Agenda

- Quiz 1
- Quiz 2
- Go over PS2
Nehalem-EX Xeon 7560

- 2010
- octo-core
- 2.3 Billion transistors
- 45 nm
- 2.26 GHz
- 24 MB L3 cache (on-chip!!)
- 130 Watts (TDP)
- $3,692 (per die, when bought in 1k units)
  - (naturally slightly slower versions with smaller L3 caches go from $900-1400)
Quiz 1

Individual Question Breakdown

<table>
<thead>
<tr>
<th></th>
<th>Q1 (10)</th>
<th>Q2 (25)</th>
<th>Q3 (20)</th>
<th>Q4 (24)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Avg</td>
<td>8.2</td>
<td>14.0</td>
<td>12.1</td>
<td>18.4</td>
</tr>
<tr>
<td>Std</td>
<td>2.5</td>
<td>6.4</td>
<td>7.0</td>
<td>5.8</td>
</tr>
</tbody>
</table>

Total (80 points)
Avg: 53 points
Std: 16 points
struct Node {
    int    value;
    Node*  next;
}

void AddToLinkedList(Node* head_ptr, Node* new_ptr) {
    Node* curr_ptr = head_ptr;

    while (curr_ptr->next != 0) {
        curr_ptr = curr_ptr->next;
    }

    curr_ptr->next = new_ptr;
}

ATTL:   A  <-  rs1
Loop:   MA  <-  A  +  4
         :   A  <-  Mem
         :   if(A!=0) jmp to Loop
Done:   Mem  <-  rs2
         :   jmp to FETCH
## ATTL - final solution

**ATTL:** \( A \leftarrow rs1 \)

**Loop:** \( MA \leftarrow A + 4 \)
- \( A \leftarrow Mem \)
- if\( (A!=0) \) jmp to Loop

**Done:** \( Mem \leftarrow rs2 \)
- jmp to FETCH

| State       | PseudoCode          | IdIR | Reg Sel | Reg Wr | en Reg | ldA | ldB | ALUOp | en ALU | ld MA | Mem Wr | en Mem | Ex Sel | en Imm | µBr | Next State |
|-------------|---------------------|------|---------|--------|--------|-----|-----|-------|--------|-------|--------|--------|--------|------|------------|
| ATTL: A <- rs1 | 0 rs1 0 1 1 * * | 0 * * | 0 * * | 0 * * | 0 N |
| LOOP: MA<- A+4 | 0 * * 0 * 0 INC_A4 | 0 1 * 0 * 0 N |
| A <- Mem | 0 * * 0 1 * | 0 0 0 0 1 * 0 S |
| If A!=0 | 0 * * 0 0 * COPYA | 0 0 * 0 * 0 S |
| j loop | 0 * * 0 0 * | 0 0 * 0 0 NZ LOOP |
| DONE: Mem <- rs2 | 0 rs2 0 1 * * | 0 0 1 1 * 0 S |
| Jmp to Fetch | * * * 0 * * | 0 * * 0 * 0 J FETCH |

Friday, March 2, 2012
ATTL - Exceptions

• Precise Exceptions
  – PC = PC - 4
  – restart from the beginning
  – no state needed to be saved!

  – but if too long, we can’t guarantee forward progress! (keep having to restart...)
0x2000: ADDI x4, x0, 0
0x2004: ADDI x5, x0, 1
0x2008: BEQ x4, x5, 0x2000
0x200c: LW x7, 4(x6)
0x2010: OR x5, x7, x5
0x2014: XOR x7, x7, x3
0x2018: AND x3, x2, x3

Friday, March 2, 2012
Question 3. Part a

<table>
<thead>
<tr>
<th>PC</th>
<th>Instr</th>
<th>t1</th>
<th>t2</th>
<th>t3</th>
<th>t4</th>
<th>t5</th>
<th>t6</th>
<th>t7</th>
<th>t8</th>
<th>t9</th>
<th>t10</th>
<th>t11</th>
<th>t12</th>
<th>t13</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x2000</td>
<td>ADDI</td>
<td>F</td>
<td>D</td>
<td>X</td>
<td>M</td>
<td>W</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x2004</td>
<td>ADDI</td>
<td>F</td>
<td>D</td>
<td>X</td>
<td>M</td>
<td>W</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x2008</td>
<td>BEQ</td>
<td>F</td>
<td>D</td>
<td>X</td>
<td>M</td>
<td>W</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x200c</td>
<td>LW</td>
<td>F</td>
<td>D</td>
<td>X</td>
<td>M</td>
<td>W</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x2010</td>
<td>OR</td>
<td>F</td>
<td>D</td>
<td>D</td>
<td>X</td>
<td>M</td>
<td>W</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x2014</td>
<td>XOR</td>
<td>F</td>
<td>F</td>
<td>D</td>
<td>X</td>
<td>M</td>
<td>W</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x2018</td>
<td>AND</td>
<td>F</td>
<td>D</td>
<td>X</td>
<td>M</td>
<td>W</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Question 3. Part b

0x2000: ADDI x4, x0, 0
0x2004: ADDI x5, x0, 1
0x2008: BEQ x4, x5, 0x2000
0x200c: LW x7, 4(x6)
0x2010: OR x5, x7, x5
0x2014: XOR x7, x7, x3
0x2018: AND x3, x2, x3

<table>
<thead>
<tr>
<th>PC</th>
<th>Instr</th>
<th>t1</th>
<th>t2</th>
<th>t3</th>
<th>t4</th>
<th>t5</th>
<th>t6</th>
<th>t7</th>
<th>t8</th>
<th>t9</th>
<th>t10</th>
<th>t11</th>
<th>t12</th>
<th>t13</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x2000</td>
<td>ADDI</td>
<td>F</td>
<td>D</td>
<td>X</td>
<td>M</td>
<td>W</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x2004</td>
<td>ADDI</td>
<td>F</td>
<td>D</td>
<td>X</td>
<td>M</td>
<td>W</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x2008</td>
<td>BEQ</td>
<td>F</td>
<td>D</td>
<td>X</td>
<td>M</td>
<td>W</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x200c</td>
<td>LW</td>
<td>F</td>
<td>D</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x2010</td>
<td>OR</td>
<td>F</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x2000</td>
<td>ADDI</td>
<td>F</td>
<td>D</td>
<td>X</td>
<td>M</td>
<td>W</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x2004</td>
<td>ADDI</td>
<td>F</td>
<td>D</td>
<td>X</td>
<td>M</td>
<td>W</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

![kill 2 bubbles (2 stages got killed)](image-url)
Question 3. Part c

0x2000: LW  x7, 0(x6)
0x2004: ADDI x2, x2, 1
0x2008: BEQ  x2, x3, 0x2000
0x200c: SW  x7, 0(x6)
0x2010: OR  x5, x5, 4
0x2014: OR  x7, x7, 5
Question 3. Part c

<table>
<thead>
<tr>
<th>PC</th>
<th>Instr</th>
<th>t1</th>
<th>t2</th>
<th>t3</th>
<th>t4</th>
<th>t5</th>
<th>t6</th>
<th>t7</th>
<th>t8</th>
<th>t9</th>
<th>t10</th>
<th>t11</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x2000</td>
<td>LW x7, 0(x6)</td>
<td>F</td>
<td>D</td>
<td>X</td>
<td>M</td>
<td>W</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x2004</td>
<td>ADDI x2, x2, 1</td>
<td>F</td>
<td>D</td>
<td>X</td>
<td>M</td>
<td>W</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x2008</td>
<td>BEQ x2, x3, 0x2000</td>
<td>F</td>
<td>D</td>
<td>X</td>
<td>M</td>
<td>W</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x200c</td>
<td>SW x7, 0(x6)</td>
<td>F</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x2000</td>
<td>LW x7, x7, 5</td>
<td>F</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Question 3. Part d

0x2000: LW x7, 0(x6)
0x2004: ADDI x2, x2, 1
0x2008: BEQ x2, x3, 0x2000
0x200c: SW x7, 0(x6)
0x2010: OR x5, x5, 4
0x2014: OR x7, x7, 5
Question 3. Part d

0x2000: LW x7, 0(x6)
0x2004: ADDI x2, x2, 1
0x2008: BEQ x2, x3, 0x2000
0x200c: SW x7, 0(x6)
0x2010: OR x5, x5, 4
0x2014: OR x7, x7, 5

<table>
<thead>
<tr>
<th>PC</th>
<th>Instr</th>
<th>t1</th>
<th>t2</th>
<th>t3</th>
<th>t4</th>
<th>t5</th>
<th>t6</th>
<th>t7</th>
<th>t8</th>
<th>t9</th>
<th>t10</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x2000</td>
<td>LW</td>
<td>F</td>
<td>D</td>
<td>X</td>
<td>M</td>
<td>W</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x2004</td>
<td>ADDI</td>
<td>F</td>
<td>D</td>
<td>X</td>
<td>M</td>
<td>W</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x2008</td>
<td>BEQ</td>
<td>F</td>
<td>D</td>
<td>X</td>
<td>M</td>
<td>W</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x2000</td>
<td>LW</td>
<td>F</td>
<td>D</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x2004</td>
<td>ADDI</td>
<td>F</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x200c</td>
<td>SW</td>
<td>F</td>
<td>D</td>
<td>X</td>
<td>M</td>
<td>W</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>F</td>
<td>D</td>
<td>X</td>
<td>M</td>
<td>W</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>F</td>
<td>D</td>
<td>X</td>
<td>M</td>
<td>W</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
• Quiz 2 on Tuesday!
  – closed book
  – will look just like pset (and past tests)
Quiz 2 - Possible Topics

• Memory Hierarchies
  – caches
    • direct-mapped, set-associative
    • types of misses (compulsary, conflict, capacity)
    • AMAT
    • critical paths, physical design

• Virtual Memory
  – address translation, protection
  – TLBs
    • pages, page tables, page walking
  – aliasing

• software interplay
  – analyze software behavior
  – optimizing for memory/VM
Quiz 2 - How to study?

• fully understand pset #2
• go over past tests
  – time yourself!
• go over past psets
PSet 2

- Out of six points
- (but is worth same amount overall as pset#1)
PSet 2 - Problem 1

Diagram showing the flow of input address through tag and index to tag decoder, data decoder, and MUX to output driver.

Tag  | Status
-----|-------
   *  |      *
   *  |      *
   *  |      *

Input Address

2^b: cache line size (bytes)

Tag Decoder

Data Decoder

2^(b-2) data words

MUX

Data Output Driver

Valid Bit

Comparator

Valid Output Driver

Friday, March 2, 2012
PSet 2 - Problem 1
• Can the 3rd and 4th stages be combined?
  – yes, just insert NOP
  – we aren’t changing state, so it’s okay

• What about D$ tag check, write-back?
  – technically, yes
  – in reality, pretty difficult to get right
    • could write incorrect value into register file
    • need to devise way to undo this
    • e.g., can you handle LD r1, o(r1)?
Figure 2. R10000 block diagram (a) and pipeline timing diagram (b). The block diagram shows pipeline stages left to right to correspond to pipeline timing.

note: reg-read happens at end of cycle, reg-write happens at beginning of cycle
Can the 3rd and 4th stages be combined?
- yes, just insert NOP
- we aren’t changing state, so it’s okay

What about D$ tag check, write-back?
- technically, yes
- in reality, pretty difficult to get right
  - could write incorrect value into register file
  - need to devise way to undo this
  - e.g., can you handle LD r1, 0(r1)?
What about writes? Does this work?

- need to ensure we don’t write to memory if tag check misses

solutions:

- add fourth stage (but then ST,LD structural hazard)
- delayed write-buffer (much better solution, but now loads must also check the write-buffer)
Does this work for set-associative instruction cache?
- cycle time hit, improved miss rate
- no, tag check required to get data out of the correct way

Does this work for set-associative data cache?
- yes, we don’t use data until after tag check
- (not that this is a good idea. in particular, you can’t begin to drive data out until tag check completed)
### PSet 2 - Problem 2.D

<table>
<thead>
<tr>
<th>I-Cache Address Decode</th>
<th>I-Cache Array Access</th>
<th>I-Cache Tag Check, Instruction Decode &amp; Register Fetch</th>
<th>Execute</th>
<th>D-Cache Address Decode</th>
<th>D-Cache Array Access</th>
<th>D-Cache Tag Check</th>
<th>Write-Back</th>
</tr>
</thead>
</table>

- Branches resolved in Execute...
- What is the branch delay?
  - 3 cycles
### PSet 2 - Problem 2.E

<table>
<thead>
<tr>
<th>I-Cache Address Decode</th>
<th>I-Cache Array Access</th>
<th>I-Cache Tag Check, Instruction Decode &amp; Register Fetch</th>
<th>Execute</th>
<th>D-Cache Address Decode</th>
<th>D-Cache Array Access</th>
<th>D-Cache Tag Check</th>
<th>Write-Back</th>
</tr>
</thead>
</table>

- Bypassing speculated loads...
- How can an interlock in Decode handle this?
  - stall if dependent on load in EXE, DAD,DAA
- How many cycles is the load-use delay? (assuming hit)
  - 3 cycles
• Bypassing speculated loads...
• Instead let’s speculatively execute. What state do we need to store to restart the pipeline on a misspeculation?
  – track the PC of the bad load
  – restart pipeline when data comes back
  – (it’s basically an exception!)
• load delay is now how many cycles?
  – 2 cycles
PSet 2 - Problem 2
PSet 2 - Problem 1
The matrix A is stored contiguously in memory in row-major order. Row major order means that elements in the same row of the matrix are adjacent in memory as shown in the following memory layout:

A[i][j] resides in memory location [4*(64*i + j)]

Memory Location:

<table>
<thead>
<tr>
<th>0</th>
<th>4</th>
<th>252</th>
<th>256</th>
<th>4*(64*127+63)</th>
</tr>
</thead>
</table>
Loop A

```c
sum = 0;
for (i = 0; i < 128; i++)
    for (j = 0; j < 64; j++)
        sum += A[i][j];
```

Loop B

```c
sum = 0;
for (j = 0; j < 64; j++)
    for (i = 0; i < 128; i++)
        sum += A[i][j];
```

- 4KB direct-mapped, 8-word lines (32B)
- How many lines?
  - 128 lines
- $64 \times 128 = 8192$ elements, or 32768 bytes (32KB)
- Loop A misses:
  - 1024 (notice the 4KB capacity is irrelevant)
- Loop B misses:
  - 8192 (100% misses, cache is too small to save us)
### Problem 3.B

**Loop A**

```plaintext
sum = 0;
for (i = 0; i < 128; i++)
    for (j = 0; j < 64; j++)
        sum += A[i][j];
```

**Loop B**

```plaintext
sum = 0;
for (j = 0; j < 64; j++)
    for (i = 0; i < 128; i++)
        sum += A[i][j];
```

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>4</th>
<th>252</th>
<th>256</th>
<th>4*(64*127+63)</th>
</tr>
</thead>
</table>

- 4KB direct-mapped, 8-word lines (32B), 128 lines
- 64*128 = 8192 elements, or 32768 bytes (32KB)
- Cache size required?
  - 1 cache line
  - Cache size required?
  - 1024 cache lines
### Loop A

```c
sum = 0;
for (i = 0; i < 128; i++)
    for (j = 0; j < 64; j++)
        sum += A[i][j];
```

### Loop B

```c
sum = 0;
for (j = 0; j < 64; j++)
    for (i = 0; i < 128; i++)
        sum += A[i][j];
```

<table>
<thead>
<tr>
<th>0</th>
<th>4</th>
<th>252</th>
<th>256</th>
<th>4*(64*127+63)</th>
</tr>
</thead>
</table>

- 4KB fully-associ, 8-word lines (32B), 128 lines, FIFO
- $64 \times 128 = 8192$ elements, or 32768 bytes (32KB)
- Loop A misses?
  - **1024** (still only compulsory, can’t fix this)
- Loop B misses?
  - **1024** (only compulsory, we can fit an entire column of matrix \( A \) (8x128 elements)
**PSet 2 - Problem 4-Critical Path?**

![Diagram of a cache memory system with components such as Tag, Index, Data Decoder, and MUXes.](image-url)
PSet 2 - Problem 4-Critical Path?

tag decoder -> tag read -> comparator -> 2-in AND -> buffer driver -> data output driver

240+330+500+50+200+300 = 1620ps, or 1.62 ns
PSet 2 - Problem 4
tag check = 1320ps

final answer (through utags): 1350ps, or 1.35 ns.

data up to data output driver = 930ps

microtag to buffer driver = 1050ps
• AMAT?
• 95% hit rate
• miss penalty = 20ns
  – AMAT = hit_time + miss_rate*miss_penalty
  – AMAT = hit_time + (0.05)*(20ns)
  – AMAT = hit_time + 1ns
  – AMAT_old = 1.62ns + 1ns = 2.62ns
  – AMAT_new = 1.35ns + 1ns = 2.35ns
new constraint:
  – all microtags in set must be unique!

What type of miss does this affect?
  – conflict miss

How does miss_rate compare to 4-way?
  – worse... more conflict misses

How does miss_rate compare to direct-mapped?
  – at least as good (better)... similiar microtags in a given set would already alias and kick each other out
Problem:
- an alias will see a HIT (physical tag already present), **BUT** it will not see its microtag
  -> data_out will float and the CPU gets incorrect data

Physical tags
Virtual microtags
Virtual indices
PSet 2 - Problem 4.E - Fix?
PSet 2 - Problem 4.E - Fix?

Solution
- if tag check hits, but utag misses, then you have an alias (also must verify both physical tag and microtag correspond to the same way)
- evict first alias
• $2x$ associativity (halves # of sets)
  – compulsory misses
    • no effect
  – conflict misses
    • reduces
    • more places to put in a set
  – capacity misses
    • no effect
    • capacity unchanged
• halving line size (halves capacity)
  – compulsory misses
    • increases
    • less “prefetching” data on same line
    • less able to exploit spatial locality
  – conflict misses
    • no effect
  – capacity misses
    • increase
    • capacity has been halved
• double number of sets (halves associativity)
  – compulsory misses
    • no effect
  – conflict misses
    • increase
    • less associativity
  – capacity misses
    • no effect
• adding prefetching
  – compulsory misses
    • decreases
    • good prefetecher brings in data before we need it
  – conflict misses
    • increase
    • prefetched data could pollute the cache
  – capacity misses
    • increase
    • prefetched data could pollute the cache
• double associativity (halves # of sets)
  – hit time
    • increases
    • sets decrease -> larger tags
    • more tags to check, more ways to mux out
  – miss rate
    • decrease
    • less conflict misses
  – miss penalty
    • no effect
    • dominated by the other parts of the memory hierarchy
• halving line size (halves capacity)
  – hit time
    • decreases
    • cache becomes physically smaller (x2), so quicker access probably dominates larger tag check (by 1 bit)
  – miss rate
    • increases
    • smaller capacity, less spatial locality
  – miss penalty
    • decreases
    • uses less bandwidth out to memory
• double # of sets (halves associativity)
  – hit time
    • decreases
    • less logic getting in the way, smaller tags
  – miss rate
    • increases
    • more conflict misses
  – miss penalty
    • no effect
    • this is dominated by outer memory heirarchy
• adding prefetching
  – hit time
    • no effect
    • isn’t on critical path for the hit
  – miss rate
    • decrease
    • this is the purpose of a prefetcher!
    • bring in data a head of time
  – miss penalty
    • decreases
    • prefetch could be inflight when miss occurs