Outline

• More arithmetic instructions
• MUL is special…
• DIV is too!
• Mixing data types
• Logical Instructions (AND, OR, NOT, XOR)
• Negative Numbers (next week?)
More Arithmetic Instructions
Intel Pentium 4 Northwood

Buffer Allocation & Register Rename
Instruction Queue (for less critical fields of the uOps)
General Instruction Address Queue & Memory Instruction Address Queue
(queues register entries and latency fields of the uOps for scheduling)
Floating Point, MMX, SSE2
Renamed Register File
128 entries of 128 bit.

uOp Schedulers
FP Move Scheduler:
(8x8 dependency matrix)
Parallel (Matrix) Scheduler
for the two double pumped ALU's
General Floating Point and Slow Integer Scheduler:
(8x8 dependency matrix)  
Load / Store uOp Scheduler:
(8x8 dependency matrix)
Load / Store Linear Address Collision History Table

Integer Execution Core
(1) uOp Dispatch unit & Replay Buffer
Dispatches up to 6 uOps/cycle
(2) Integer Renamed Register File
128 entries of 32 bit + 6 status flags
12 read ports and six write ports
(3) Databus switch & Bypasses to and from the Integer Register File.
(4) Flags, Write Back
(5) Double Pumped ALU 0
(6) Double Pumped ALU 1
(7) Load Address Generator Unit
(8) Store Address Generator Unit
(9) Load Buffer (48 entries)
(10) Store Buffer (24 entries)

Execution Pipeline Start
Register Alias History Tables (2x126)  
Register Alias Tables uOp Queue
Micro code Sequencer
Micro code ROM & Flash
Trace Cache Fill Buffers
Distributed Tag comparators
24 bit virtual Tags

Instruction Trace Cache

Trace Cache Access, next Address Predict
Trace Cache Branch Prediction Table (BTB), 512 entries.
Return Stacks (2x16 entries)
Trace Cache next IP's (2x)
Miscellaneous Tag Data

Instruction Decoder
Up to 4 decoded uOps/cycle out.
(from max. one x86 instr/cycle)
Instructions with more than four are handled by Micro Sequencer
Trace Cache LRU bits
Raw Instruction Bytes in
Data TLB, 64 entry fully associative, between threads dual ported (for loads and stores)

Instruction Fetch from L2 cache and Branch Prediction
Front End Branch Prediction Tables (BTB), shared, 4096 entries in total
Instruction TLB's 2x64 entry, fully associative for 4k and 4M pages.
In: Virtual address [31:12]
Out: Physical address [35:12] + 2 page level bits

Front Side Bus Interface, 400..800 MHz

April 19, 2003  www.chip-architect.com

D. Thiebaut, Computer Science, Smith College
Right now, we are dealing only with **UNSIGNED** integers!
inc

alpha db 3
beta dw 4
x dd 0

inc al
inc cx
inc ebx

inc word[beta] ;beta <- 5
inc dword[x] ;x <- 1
```assembly
alpha db 3
beta dw 4
x dd 6

; Decrement operands
dec al ; al <- al - 1
dec cx
dec ebx

word[beta] dw 3
word[x] dw 5

```
mul

mul operand

- mul reg8
- mul reg16
- mul reg32
- mul mem8
- mul mem16
- mul mem32
Observation

\[
\begin{array}{c}
1001 \\
x \quad 1110
\end{array}
\]
mul

mul operand

edx:eax ← operand₃₂ * eax
dx:ax ← operand₁₆ * ax
ax ← operand₈ * al

alpha db 3
beta dw 4
x dd 6

mul byte[alpha] ; ax ← al*alpha
mul ebx ; edx:eax ← ebx*eax
This has tremendously important consequences!
public class JavaLimits {
    public static void main(String[] args) {
        // -----------------------------------------------
        // a multiplication of ints
        int x = 0x30000001;
        int y = 0x30000001;

        System.out.println( "x = " + x );
        System.out.println( "y = " + y );

        int z = x * y;

        System.out.println( "z = " + z );
        System.out.println();
    }
}
public class JavaLimits {

    public static void main(String[] args) {
        // -----------------------------------------------
        // a multiplication of ints
        int x = 0x30000001;
        int y = 0x30000001;

        System.out.println( "x = " + x );
        System.out.println( "y = " + y );

        int z = x * y;

        System.out.println( "z = " + z );
        System.out.println();
    }
}

x = 805306369
y = 805306369
z = 1610612737
public class JavaLimits {

    public static void main(String[] args) {
        // -----------------------------------------------
        // a multiplication of ints
        int x = 0x30000001;
        int y = 0x30000001;

        System.out.println( "x = " + x );
        System.out.println( "y = " + y );

        int z = x * y;

        System.out.println( "z = " + z );
        System.out.println();
    }

}
int x, y, z;
x = ...
y = ...
z = x * y;
int x, y, z;
x = ...  
y = ...  
z = x * y;
Java's Attitude

```java
int x, y, z;
x = ...;
y = ...;
z = x * y;
```
Java's Attitude

```java
int x, y, z;
x = ...
y = ...
z = x * y;
```
Same Computation in Python...

```bash
cs231a@marax:~/handout$ python3
Python 3.5.2 (default, Nov 23 2017, 16:37:01)
[GCC 5.4.0 20160609] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> x = 0x30000001
>>> y = 0x30000001
>>> z = x*y
>>> print(x, y, z, sep="\n")
805306369
805306369
648518347951964161
>>> 
cs231a@marax:~/handout$ 
```
Same Computation in Python...

```sh
$ python3
Python 3.5.2 (default, Nov 23 2017, 16:37:01)
[GCC 5.4.0 20160609] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> x = 0x30000001
>>> y = 0x30000001
>>> z = x*y
>>> print(x, y, z, sep="\n")
805306369
805306369
648518347951964161
```
How big is a 32-bit int?
Ranges
(Unsigned Integers)

- 8 bits: 0 - 255
- 16 bits: 0 - 65,535
- 32 bits: 0 - 4,294,967,295
Puzzling Behavior
What is the output?

```java
int x = 1;

for (int i=0; i<40; i++) {
    System.out.println(x);
    x = x * 2;
}
```

getcopy JavaMulBy2.java
mul operand

dx:eax ← operand_{32} * eax
dx:ax ← operand_{16} * ax
ax ← operand_{8} * al

alpha db 3
beta dw 4
x dd 6

mul byte[alpha] ;ax ← al*alpha
mul ebx ;edx:eax ←
            ; ebx*eax
; compute \( \beta / \alpha \)
```
mov ax, word[beta]
div byte[alpha]
```
; quotient in al
; remainder in ah
Exercise

Compute $x = 2\alpha + 3\beta + x - 1$

$$\begin{align*}
\alpha & \text{ db} & 3 \\
\beta & \text{ dw} & 4 \\
x & \text{ dd} & 6
\end{align*}$$
section .text
global _start

_start:

;; 2*alpha
mov eax, 0
mov al, byte[alpha] ;eax <- alpha
add eax, eax ;eax <- 2*alpha

;; 3*beta
mov ebx, 0
mov bx, word[beta] ;ebx <- beta

;; 2*alpha + 3*beta
add eax, ebx ;eax <- 2*alpha + beta
add eax, ebx ;eax <- 2*alpha + 2*beta
add eax, ebx ;eax <- 2*alpha + 3*beta

;; 2*alpha + 3*beta + x +1
add eax, dword[x] ;eax <- 2*alpha + 3*beta
add eax, 1 ;eax <- 2*alpha + 3*beta+1

;; x = 2*alpha + 3*beta + x +1
mov DWORD[x], eax
Logical Instructions

AND, OR, NOT, XOR
Is \((x,y)\) inside Rectangle?

\((x_1,y_1)\)

\((x_2,y_2)\)
<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>a and b</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>a</td>
<td>b</td>
<td>a and b</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---------</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>a and b</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>a</td>
<td>b</td>
<td>a and b</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---------</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>a and b</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>a or b</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>a</td>
<td>b</td>
<td>a and b</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---------</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>a and b</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>a or b</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>a xor b</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
### Boolean Operations

#### a and b

<table>
<thead>
<tr>
<th></th>
<th></th>
<th><strong>a and b</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
</tbody>
</table>

#### a or b

<table>
<thead>
<tr>
<th></th>
<th></th>
<th><strong>a or b</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

#### a xor b

<table>
<thead>
<tr>
<th></th>
<th></th>
<th><strong>a xor b</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

#### not a

<table>
<thead>
<tr>
<th></th>
<th><strong>not a</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Instruction</td>
<td>Feature</td>
</tr>
<tr>
<td>-------------</td>
<td>----------------------------------------------</td>
</tr>
<tr>
<td>AND</td>
<td>Good for setting bits to 0</td>
</tr>
<tr>
<td>OR</td>
<td>Good for setting bits to 1</td>
</tr>
<tr>
<td>XOR</td>
<td>Good for flipping bits</td>
</tr>
<tr>
<td>NOT</td>
<td>Good for complementing all the bits</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Feature</th>
</tr>
</thead>
<tbody>
<tr>
<td>AND</td>
<td>Good for setting bits to 0</td>
</tr>
<tr>
<td>OR</td>
<td>Good for setting bits to 1</td>
</tr>
<tr>
<td>XOR</td>
<td>Good for flipping bits</td>
</tr>
<tr>
<td>NOT</td>
<td>Good for complementing all the bits</td>
</tr>
</tbody>
</table>

**Examples:**

- **AND:**
  - `10010` and `11100` = `10000`

- **OR:**
  - `10010` or `11100` = `11110`

- **XOR:**
  - `10010` xor `11100` = `01110`

- **NOT:**
  - `10010` not `11100` = `00011`
and dest, src

\[
\begin{align*}
\text{alpha} & \equiv \text{db} 0xf3 \\
\text{beta} & \equiv \text{dw} 4 \\
x & \equiv \text{dd} 6 \\
\text{and} & \equiv \text{byte[alpha]}, 7 \\
\text{mov} & \equiv \text{ax, 0x1234} \\
\text{and} & \equiv \text{ax, 0xFF00} \\
\text{and} & \equiv \text{dword[x]}, 2
\end{align*}
\]
or dest, src

alpha db 3
beta dw 4
x dw 0x0F06

or byte[alpha], 4
mov ax, 0x1234
or ax, 0xFF00

or word[x], 15
xor
xor dest, src

alpha db 3
beta dw 4
x dd 0xF06

xor byte[alpha], 0x0F
mov ax, 0x1234
xor ax, 0xFF00
xor dword[x], 15
not oprnd

alpha db 3
beta dw 4
x dd 0xF06

not byte[alpha]
mov ax,0x1234
not ax

not dword[x]
We Stopped Here Last Time...