CSC231 - Assembly

Week #6

Dominique Thiébaut
dthiebaut@smith.edu
Misc. Notes & Review
You Can Add A Register to Itself

; x = 2 * (a-b)

mov eax, dword[a]  ; eax <- a
sub eax, dword[b]  ; eax <- a-b
add eax, eax       ; eax <- eax+eax
                   ; eax <- 2*(a-b)
mov dword[x], eax   ; x <- 2*(a-b)
Speed of Computation
Frequency & Cycle
Time
Frequency: 3.2 GHz

cycle: 1/3.2 GHz
=0.3125 ns

sec
ms
us
ns
Arduino

Clock speed: **16 MHz**

~1/200 speed of Pentium

```asm
;hello.asm
; turns on an LED which is connected to PB5 (digital out 13)

.include "./m328Pdef.inc"

  ldi r16,0b00100000
  out DDRB,r16
  out PortB,r16
Start:
  rjmp Start
```

Raspberry Pi

Clock speed: **1.4 GHz**

~1/3 speed of Pentium

```c
/* -- first.s */
/* This is a comment */
.global main /* 'main' is our entry point and must be global */

main: /* This is main */
    mov r0, #2 /* Put a 2 inside the register r0 */
    bx lr /* Return from main */
```

N-Queens Problem

Afternoon Plans?

Let’s go to Boston this afternoon. I’ll drive!
Afternoon Plans?

Let’s go to Boston this afternoon. I’ll drive!
Afternoon Plans?

Let's go for a drive in the woods. I'll drive!
Know your toolbox!
Summit: Fastest Supercomputer
Oak Ridge National Laboratory

https://www.olcf.ornl.gov/summit/
Decimal

- Number of digits, the base
- Count in decimal
- Express number as sum of products
- Add two digits
- Add two numbers
Binary

- Number of digits, the base
- Count in binary
- Express number as sum of products
- Add two digits
- Add two numbers
Base 3

- Number of digits, the base
- Count in base 3
- Express number as sum of products
- Add two digits
- Add two numbers
Hexadecimal

- Number of digits, the base
- Count in hex
- Express number as sum of products
- Add two digits
- Add two numbers
We Stopped Here Last Time...
<table>
<thead>
<tr>
<th>Binary</th>
<th>Decimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>0</td>
</tr>
<tr>
<td>0001</td>
<td>1</td>
</tr>
<tr>
<td>0010</td>
<td>2</td>
</tr>
<tr>
<td>0011</td>
<td>3</td>
</tr>
<tr>
<td>0100</td>
<td>4</td>
</tr>
<tr>
<td>0101</td>
<td>5</td>
</tr>
<tr>
<td>0110</td>
<td>6</td>
</tr>
<tr>
<td>0111</td>
<td>7</td>
</tr>
<tr>
<td>1000</td>
<td>8</td>
</tr>
<tr>
<td>1001</td>
<td>9</td>
</tr>
<tr>
<td>1010</td>
<td>A</td>
</tr>
<tr>
<td>1011</td>
<td>B</td>
</tr>
<tr>
<td>1100</td>
<td>C</td>
</tr>
<tr>
<td>1101</td>
<td>D</td>
</tr>
<tr>
<td>1110</td>
<td>E</td>
</tr>
<tr>
<td>1111</td>
<td>F</td>
</tr>
</tbody>
</table>
Decimal to Binary Conversion

Review

Shifting
$101001 = 1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$
101001 = 1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 \\
= 32 + 0 + 8 + 0 + 0 + 1 \\
= 41_{10}
101001 = 1\times2^5 + 0\times2^4 + 1\times2^3 + 0\times2^2 + 0\times2^1 + 1\times2^0 \\
= 32 + 0 + 8 + 0 + 0 + 1 \\
= 41_{10}

\[
\begin{align*}
\frac{101001}{2} &= \frac{1\times2^5 + 0\times2^4 + 1\times2^3 + 0\times2^2 + 0\times2^1 + 1\times2^0}{2} \\
&= 32 + 0 + 8 + 0 + 0 + 1 \\
&= 41_{10}
\end{align*}
\]
101001 = 1\times2^5 + 0\times2^4 + 1\times2^3 + 0\times2^2 + 0\times2^1 + 1\times2^0 \\
= 32 + 0 + 8 + 0 + 0 + 1 \\
= 41_{10}

\[
\begin{align*}
101001 & = 1\times2^5 + 0\times2^4 + 1\times2^3 + 0\times2^2 + 0\times2^1 + 1\times2^0 \\
& = \frac{1\times2^5}{2} + \frac{0\times2^4}{2} + \frac{1\times2^3}{2} + \frac{0\times2^2}{2} + \frac{0\times2^1}{2} + \frac{1\times2^0}{2}
\end{align*}
\]
\[
101001 = 1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 \\
= 32 + 0 + 8 + 0 + 0 + 1 \\
= 41_{\text{d}}
\]

\[
\frac{101001}{2} = \frac{1 \times 2^5}{2} + \frac{0 \times 2^4}{2} + \frac{1 \times 2^3}{2} + \frac{0 \times 2^2}{2} + \frac{0 \times 2^1}{2} + \frac{1 \times 2^0}{2} \\
= 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 0 \times 2^0
\]
101001 = 1x2^5 + 0x2^4 + 1x2^3 + 0x2^2 + 0x2^1 + 1x2^0
= 32 + 0 + 8 + 0 + 0 + 1
= 41d

\[
\frac{101001}{2} = \frac{1x2^5 + 0x2^4 + 1x2^3 + 0x2^2 + 0x2^1 + 1x2^0}{2}
\]
\[
= \frac{1x2^5}{2} + \frac{0x2^4}{2} + \frac{1x2^3}{2} + \frac{0x2^2}{2} + \frac{0x2^1}{2} + \frac{1x2^0}{2}
\]
\[
= 1x2^4 + 0x2^3 + 1x2^2 + 0x2^1 + 0x2^0
\]
\[
= 10100
\]
\[
101001 = 1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 \\
= 32 + 0 + 8 + 0 + 0 + 1 \\
= 41_{10}
\]

\[
\frac{101001}{2} = \frac{1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0}{2} \\
= \frac{1 \times 2^5}{2} + \frac{0 \times 2^4}{2} + \frac{1 \times 2^3}{2} + \frac{0 \times 2^2}{2} + \frac{0 \times 2^1}{2} + \frac{1 \times 2^0}{2} \\
= 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 0 \times 2^0 \\
= 10100 \\
= 16 + 0 + 4 + 0 + 0 \\
= 20_{10}
\]
Dividing by the base extracts the least significant digit
# prompts user for an integer
# decomposes the integer into binary

x = int( input( "> " ) )
binary = ""

while True:
    if x==0:
        break

    remainder = x % 2
    quotient  = x // 2

    if remainder == 0:
        binary = "0" + binary
    else:
        binary = "1" + binary

    print( "%5d = %5d * 2 + %d    quotient=%5d remainder=%d binary=%16s" % (x, quotient, remainder, quotient, remainder, binary ) )

    x = quotient
```python
# prompts user for an integer
# decomposes the integer into binary
x = int( input( "> " ) )
binary = ""

while True:
    if x==0:
        break
    remainder = x % 2
    quotient  = x // 2
    if remainder == 0:
        binary = "0" + binary
    else:
        binary = "1" + binary
    print( "%5d = %5d * 2 + %d    quotient=%5d remainder=%d binary=%16s" % (x, quotient, remainder, quotient, remainder, binary ) )
    x = quotient
```
Binary to Decimal
101001 = 1\times2^5 + 0\times2^4 + 1\times2^3 + 0\times2^2 + 0\times2^1 + 1\times2^0
= 32 + 0 + 8 + 0 + 0 + 1
= 41d
Binary to Hex
1010010 =
1010010 = 01010010
1010010 = 01010010
    = 0101\,0010\wedge
\[ 1010010 = 01010010 \\
\quad = 01010010 \]
<table>
<thead>
<tr>
<th>Binary</th>
<th>Decimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>0</td>
</tr>
<tr>
<td>0001</td>
<td>1</td>
</tr>
<tr>
<td>0010</td>
<td>2</td>
</tr>
<tr>
<td>0011</td>
<td>3</td>
</tr>
<tr>
<td>0100</td>
<td>4</td>
</tr>
<tr>
<td>0101</td>
<td>5</td>
</tr>
<tr>
<td>0110</td>
<td>6</td>
</tr>
<tr>
<td>0111</td>
<td>7</td>
</tr>
<tr>
<td>1000</td>
<td>8</td>
</tr>
<tr>
<td>1001</td>
<td>9</td>
</tr>
<tr>
<td>1010</td>
<td>A</td>
</tr>
<tr>
<td>1011</td>
<td>B</td>
</tr>
<tr>
<td>1100</td>
<td>C</td>
</tr>
<tr>
<td>1101</td>
<td>D</td>
</tr>
<tr>
<td>1110</td>
<td>E</td>
</tr>
<tr>
<td>1111</td>
<td>F</td>
</tr>
</tbody>
</table>

$1010010 = 01010010$

$= 01010010$

$= 5 \ 2$
Hex to Binary
<table>
<thead>
<tr>
<th>Hexadecimal</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0000</td>
</tr>
<tr>
<td>1</td>
<td>0001</td>
</tr>
<tr>
<td>2</td>
<td>0010</td>
</tr>
<tr>
<td>3</td>
<td>0011</td>
</tr>
<tr>
<td>4</td>
<td>0100</td>
</tr>
<tr>
<td>5</td>
<td>0101</td>
</tr>
<tr>
<td>6</td>
<td>0110</td>
</tr>
<tr>
<td>7</td>
<td>0111</td>
</tr>
<tr>
<td>8</td>
<td>1000</td>
</tr>
<tr>
<td>9</td>
<td>1001</td>
</tr>
<tr>
<td>A</td>
<td>1010</td>
</tr>
<tr>
<td>B</td>
<td>1011</td>
</tr>
<tr>
<td>C</td>
<td>1100</td>
</tr>
<tr>
<td>D</td>
<td>1101</td>
</tr>
<tr>
<td>E</td>
<td>1110</td>
</tr>
<tr>
<td>F</td>
<td>1111</td>
</tr>
</tbody>
</table>

$5A1F = 1010101000011111_2$
<table>
<thead>
<tr>
<th>Binary</th>
<th>Hex</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>0</td>
</tr>
<tr>
<td>0001</td>
<td>1</td>
</tr>
<tr>
<td>0010</td>
<td>2</td>
</tr>
<tr>
<td>0011</td>
<td>3</td>
</tr>
<tr>
<td>0100</td>
<td>4</td>
</tr>
<tr>
<td>0101</td>
<td>5</td>
</tr>
<tr>
<td>0110</td>
<td>6</td>
</tr>
<tr>
<td>0111</td>
<td>7</td>
</tr>
<tr>
<td>1000</td>
<td>8</td>
</tr>
<tr>
<td>1001</td>
<td>9</td>
</tr>
<tr>
<td>1010</td>
<td>A</td>
</tr>
<tr>
<td>1011</td>
<td>B</td>
</tr>
<tr>
<td>1100</td>
<td>C</td>
</tr>
<tr>
<td>1101</td>
<td>D</td>
</tr>
<tr>
<td>1110</td>
<td>E</td>
</tr>
<tr>
<td>1111</td>
<td>F</td>
</tr>
</tbody>
</table>

$5A1F = 0101\ 1010\ 0001\ 1111$
Hex to Decimal
1A2F =
1A2F = 1\times 16^3 + A\times 16^2 + 2\times 16^1 + F\times 16^0
=

\begin{align*}
1\text{A2F} &= 1 \times 16^3 + A \times 16^2 + 2 \times 16^1 + F \times 16^0 \\
&= 1 \times 4096 + 10 \times 256 + 2 \times 16 + 15 \times 1 \\
&= 
\end{align*}
1A2F = 1 \times 16^3 + A \times 16^2 + 2 \times 16^1 + F \times 16^0
= 1 \times 4096 + 10 \times 256 + 2 \times 16 + 15 \times 1
= 4096 + 2560 + 32 + 15
= 

<table>
<thead>
<tr>
<th>Hex</th>
<th>Binary</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0000</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0001</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>0010</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>0011</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>0100</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>0101</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>0110</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>0111</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>1000</td>
<td>8</td>
</tr>
<tr>
<td>9</td>
<td>1001</td>
<td>9</td>
</tr>
<tr>
<td>A</td>
<td>1010</td>
<td>10</td>
</tr>
<tr>
<td>B</td>
<td>1011</td>
<td>11</td>
</tr>
<tr>
<td>C</td>
<td>1100</td>
<td>12</td>
</tr>
<tr>
<td>D</td>
<td>1101</td>
<td>13</td>
</tr>
<tr>
<td>E</td>
<td>1110</td>
<td>14</td>
</tr>
<tr>
<td>F</td>
<td>1111</td>
<td>15</td>
</tr>
</tbody>
</table>
1A2F = 1 \times 16^3 + A \times 16^2 + 2 \times 16^1 + F \times 16^0 \\
= 1 \times 4096 + 10 \times 256 + 2 \times 16 + 15 \times 1 \\
= 4096 + 2560 + 32 + 15 \\
= 6703
Decimal to Hex
Do:
Decimal $\rightarrow$ Binary
Binary $\rightarrow$ Hex
Exercises

Convert these **hexadecimal** numbers to **binary**, and to **decimal**.

1111
1234
AAAA
F001
FFFF
Convert these **decimal** numbers to **binary**, and **hexadecimal**

65

127
What comes after these **hexadecimal** numbers, logically?

aaAA
9999
19F
1ABF
FFEF
F00F
ABCDEF
Perform the following additions in hex

\[
\begin{align*}
1000 & \quad + \quad 1234 & \quad + \quad 1234 & \quad + \quad FFFF \\
AAAA & \quad + \quad F & \quad + \quad ABCD & \quad + \quad FFFF
\end{align*}
\]
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 bits

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. Database switch & Bypass 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 (2x120)
- Register Alias Tables: uOp Queue

Instruction Trace Cache
- Microcode Sequencer
- Microcode ROM & Flash
- Trace Cache
- Distributed Tag comparators 24 bit virtual tags

Trace Cache Access, next Address Predict
- Trace Cache Branch Prediction Table (BTB), 512 entries
- Return Stacks (2x16 entries)
- Trace Cache next IPS (2x)
- Miscellaneous Tag Data

Instruction Decoder
- Up to 4 decoded uOps/cycle out (from max. one x86 instrucycle)
- 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 2048 entry, fully associative for 4k and 1M pages. In: Virtual address [31:12]
- Out: Physical address [31:12] + 2 page level bits

Front Side Bus Interface, 400..800 MHz

256 kByte L2 Cache Block

Front End Branch Prediction Tables (BTB), shared, 4096 entries in total
Instruction TLB's 2048 entry, fully associative for 4k and 1M pages. In: Virtual address [31:12]
Out: Physical address [31:12] + 2 page level bits

ROB Reorder Buffer 3x42 entries
8 kByte Level 1 Data cache four way set associative, 1R/1W
Cache Line Read / Write Transfer buffers and 256 bit wide bus to and from L2 cache

April 19, 2003 www.chip-architect.com
Right now, we are dealing only with **UNSIGNED** integers!
inc operand

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
### Decrement Instructions

<table>
<thead>
<tr>
<th>Register</th>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>reg8</td>
<td>dec</td>
<td>Decrement 8-bit register</td>
</tr>
<tr>
<td>reg16</td>
<td>dec</td>
<td>Decrement 16-bit register</td>
</tr>
<tr>
<td>reg32</td>
<td>dec</td>
<td>Decrement 32-bit register</td>
</tr>
<tr>
<td>mem8</td>
<td>dec</td>
<td>Decrement 8-bit memory</td>
</tr>
<tr>
<td>mem16</td>
<td>dec</td>
<td>Decrement 16-bit memory</td>
</tr>
<tr>
<td>mem32</td>
<td>dec</td>
<td>Decrement 32-bit memory</td>
</tr>
</tbody>
</table>

### Sample Usage

<table>
<thead>
<tr>
<th>Register</th>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>alpha</td>
<td>db</td>
<td>3</td>
</tr>
<tr>
<td>beta</td>
<td>dw</td>
<td>4</td>
</tr>
<tr>
<td>x</td>
<td>dd</td>
<td>6</td>
</tr>
</tbody>
</table>

```assembly
dec al       ; al <- al - 1
dec cx       
dec ebx      
```

```assembly
dec word[beta] ; beta <- 3
dec dword[x]   ; x <- 5
```
mul

mul operand

<table>
<thead>
<tr>
<th>mul</th>
<th>reg8</th>
</tr>
</thead>
<tbody>
<tr>
<td>mul</td>
<td>reg16</td>
</tr>
<tr>
<td>mul</td>
<td>reg32</td>
</tr>
<tr>
<td>mul</td>
<td>mem8</td>
</tr>
<tr>
<td>mul</td>
<td>mem16</td>
</tr>
<tr>
<td>mul</td>
<td>mem32</td>
</tr>
</tbody>
</table>
Observation

\[
\begin{array}{c}
1001 \\
\times \ 1110 \\
\end{array}
\]
**mul**

**mul operand**

- `edx: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`
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
Same Thing in Python...

```bash
$ 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
```

D. Thiebaut, Computer Science, Smith College
Same Thing in Python...

```
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
```
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
alpha db  3  
beta  dw  4  
x     dd  6  

;compute beta/alpha
    mov   ax, word[beta]
    div   byte[alpha]

;quotient in al
;remainder in ah
Exercise

Compute

\[ x = 2*\text{alpha} + 3*\text{beta} + x - 1 \]

\begin{align*}
\text{alpha} & \quad \text{db} & \quad 3 \\
\text{beta} & \quad \text{dw} & \quad 4 \\
x & \quad \text{dd} & \quad 6
\end{align*}