Lecture 1
- text books
- any good intro to unix
- any good intro to programming
- netlib.org for the real codes
Hardware Essentials

- Architectures
- Address space (32/64 bits)
- Data registers
- Floating point units, IEEE
- Instruction execution
- Caches
- Data storage in memory (Big/Little Endian)
Modern Computers

IBM

704

electronic
data-processing
machine

manual of operation
Architecture of modern computers
CPU architectures

Figure 1-5 The Intel Pentium M Processor Microarchitecture

System Bus

Bus Unit

2nd Level Cache

1st Level Data Cache

Front End

1st Level Instruction Cache

Fetch/Decode

Execution Out-Of-Order Core

Retirement

Frequently used paths

Less frequently used paths

BTBs/Branch Prediction

Branch History Update
CPU architectures II

Figure 1-1. High Level Structural Overview of PowerPC with AltiVec Technology
CPU architectures III

AMD Opteron™
Processor Architecture

DDR Memory Controller

L1 Instr’n Cache
L1 Data Cache
L2 Cache

AMD Opteron Processor Core

HyperTransport™ technology
CPU features

- Memory controller:
  - translates virtual to hardware memory addresses
  - data transfer external memory ↔ cache
  - prefetches data

- L1/L2/L3 data/instruction caches
  - on-chip
  - very fast (L1 fastest, L3 slowest)
  - very expensive ($$$ & space!)
CPU features

- processor core
  - instruction fetch/decode/ordering/scheduler
  - branch unit(s)
  - 1 or more integer units
  - 2 or more floating point units (FPUs)
  - 0 or more vector units
- each unit includes data registers
CPU features

- address space (32/64 bits)
  - limits available virtual address space
  - → max. size of data

- data storage in memory (Big/Little Endian)
Data storage: Big Endian

Figure 1-2. Big-Endian Byte and Bit Ordering
Floating point numbers

- several ways to represent real numbers on computers:
  - *fixed point*: place radix point somewhere in the middle of the digits
  - equivalent to using integers
  - fixed point has a fixed window of representation
  - → limits it from representing very large or very small numbers
  - prone to a loss of precision when two large numbers are divided

http://research.microsoft.com/~hollasch/cgindex/coding/ieeefloat.html
Floating point numbers

- **rationals**: represent every number as the ratio of two integers
- **floating point**: represents reals in scientific notation
- decimal: $123.456 \rightarrow 1.23456 \times 10^2$
- hex: $123.abc \rightarrow 1.23abc \times 16^2$
- ’sliding window’ of precision appropriate to the scale of the number
- can represent large and small numbers:
  - $1,000,000,000,000$
  - $0.0000000000000001$
### Storage Layout

<table>
<thead>
<tr>
<th></th>
<th>Sign</th>
<th>Exponent</th>
<th>Fraction</th>
<th>Bias</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Single Precision</strong></td>
<td>1 [31]</td>
<td>8 [30-23]</td>
<td>23 [22-00]</td>
<td>127</td>
</tr>
<tr>
<td><strong>Double Precision</strong></td>
<td>1 [63]</td>
<td>11 [62-52]</td>
<td>52 [51-00]</td>
<td>1023</td>
</tr>
</tbody>
</table>

- The sign bit is used to determine whether the number is positive or negative.
- The exponent is biased to represent a wider range of values.
- The mantissa represents the significant digits of the number.
- The bias is added to the actual exponent to get the stored exponent.
- Only possible non-zero digit for the mantissa is 1 when representing binary numbers, effectively having 24 bits for mantissa.
Ranges of Floating-Point Numbers

- 32-bit integers → 32-bits of resolution
- Single-precision floating-point → only 24 bits
- → approximate 32-bit integer by effectively truncating from the lower end
- example:
  - 11110000 11001100 10101010 00001111
    - 32-bit integer
  - = +1.1110000 11001100 10101010 × 2^{31}
    - as Single-Precision Float
  - 11110000 11001100 10101010 00000000
    - Corresponding Value
Ranges of Floating-Point Numbers

- range of positive floating point numbers:
  - normalized numbers (full precision of the mantissa)
  - denormalized numbers (portion of the fractions’s precision)
## Ranges of Floating-Point Numbers

<table>
<thead>
<tr>
<th></th>
<th>Denormalized</th>
<th>Normalized</th>
<th>Approximate Decimal</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Single Precision</strong></td>
<td>$\pm 2^{-149}$ to $(1-2^{-23}) \times 2^{-126}$</td>
<td>$\pm 2^{-126}$ to $(2-2^{-23}) \times 2^{127}$</td>
<td>$\pm 10^{-44.85}$ to $10^{38.53}$</td>
</tr>
<tr>
<td><strong>Double Precision</strong></td>
<td>$\pm 2^{-1074}$ to $(1-2^{-52}) \times 2^{-1022}$</td>
<td>$\pm 2^{-1022}$ to $(2-2^{-52}) \times 2^{1023}$</td>
<td>$\pm 10^{-323.3}$ to $10^{308.3}$</td>
</tr>
</tbody>
</table>
Ranges of Floating-Point Numbers

- five distinct numerical ranges that single-precision floating-point numbers are not able to represent:
  1. Negative numbers less than $-(2 \times 2^{23}) \times 2^{127}$ (negative overflow)
  2. Negative numbers greater than $-2^{-149}$ (negative underflow)
  3. Zero
  4. Positive numbers less than $2^{-149}$ (positive underflow)
  5. Positive numbers greater than $(2 \times 2^{23}) \times 2^{127}$ (positive overflow)
## Ranges of Floating-Point Numbers

<table>
<thead>
<tr>
<th></th>
<th>Binary</th>
<th>Decimal</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Single</strong></td>
<td>$\pm (2^{-23})^{127}$</td>
<td>$\sim \pm 10^{38.53}$</td>
</tr>
<tr>
<td><strong>Double</strong></td>
<td>$\pm (2^{-52})^{1023}$</td>
<td>$\sim \pm 10^{308.25}$</td>
</tr>
</tbody>
</table>
IEEE reserves exponent field values of all 0s and all 1s to denote special values in the floating-point scheme.

Zero

- zero is not directly representable in the straight format, due to the assumption of a leading 1
- special value denoted with an exponent field of zero and a fraction field of zero
- note that $-0$ and $+0$ are distinct values, though they both compare as equal
Special Values

- **Denormalized**
  - exponent is all 0s, but the fraction is non-zero
  - does not have an assumed leading 1 before the binary point
  - represents a number \((-1)^s \times 0.f \times 2^{-126}\) where s is the sign bit and f is the fraction
  - double precision: \((-1)^s \times 0.f \times 2^{-1022}\)
  - zero as a special type of denormalized number
Special Values

- **Infinity**
  - $+\infty$ and $-\infty$ are denoted with an exponent of all 1s and a fraction of all 0s.
  - Sign bit distinguishes between negative infinity and positive infinity.
  - Allows operations to continue past overflow situations.
  - *Operations with infinite values are well defined in IEEE floating point.*
Special Values

- **Not A Number (NaN)**
  - used to represent a value that does not represent a real number
  - represented by a bit pattern with an exponent of all 1s and a non-zero fraction
- **two categories of NaN: NaNQ (Quiet NaN) and NaNs (Signalling NaN)**
  - NaNQ is a NaN with the most significant fraction bit set
  - NaNQ’s propagate freely through most arithmetic operations
  - return from an operation when the result is not mathematically defined
NaNS is a NaN with the most significant fraction bit clear
used to signal an exception when used in operations
e.g., assign to uninitialized variables to trap premature usage
Special Operations

- any operation with a NaN yields a NaN result
- comparisons with NaNs are always false:
  - NaNQ == NaNQ → false
## Special Operations

<table>
<thead>
<tr>
<th>Operation</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n / \pm \text{Infinity}$</td>
<td>0</td>
</tr>
<tr>
<td>$\pm \text{Infinity} \times \pm \text{Infinity}$</td>
<td>$\pm \text{Infinity}$</td>
</tr>
<tr>
<td>$\pm \text{nonzero} / 0$</td>
<td>$\pm \text{Infinity}$</td>
</tr>
<tr>
<td>$\text{Infinity} + \text{Infinity}$</td>
<td>Infinity</td>
</tr>
<tr>
<td>$\pm 0 / \pm 0$</td>
<td>NaN</td>
</tr>
<tr>
<td>$\text{Infinity} - \text{Infinity}$</td>
<td>NaN</td>
</tr>
<tr>
<td>$\pm \text{Infinity} / \pm \text{Infinity}$</td>
<td>NaN</td>
</tr>
<tr>
<td>$\pm \text{Infinity} \times 0$</td>
<td>NaN</td>
</tr>
</tbody>
</table>
## IEEE 754 summary

<table>
<thead>
<tr>
<th>Sign</th>
<th>Exponent ($e$)</th>
<th>Fraction ($f$)</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>00..00</td>
<td>00..00</td>
<td>+0</td>
</tr>
<tr>
<td>0</td>
<td>00..00</td>
<td>00..01</td>
<td>Positive Denormalized Real $0.f \times 2^{(e-b+1)}$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>11..10</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>00..01</td>
<td>xx..XX</td>
<td>Positive Normalized Real $1.f \times 2^{(e-b)}$</td>
</tr>
<tr>
<td></td>
<td>11.10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>11..11</td>
<td>00..00</td>
<td>+Infinity</td>
</tr>
<tr>
<td>0</td>
<td>11..11</td>
<td>00..01</td>
<td>SNaN</td>
</tr>
<tr>
<td></td>
<td></td>
<td>01..11</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>11..11</td>
<td>10..00</td>
<td>QNaN</td>
</tr>
<tr>
<td></td>
<td></td>
<td>11..11</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>00..00</td>
<td>00..00</td>
<td>-0</td>
</tr>
<tr>
<td>1</td>
<td>00..00</td>
<td>00..01</td>
<td>Negative Denormalized Real $-0.f \times 2^{(e-b+1)}$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>11..11</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>00..01</td>
<td>xx..XX</td>
<td>Negative Normalized Real $-1.f \times 2^{(e-b)}$</td>
</tr>
<tr>
<td></td>
<td>11.10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>11..11</td>
<td>00..00</td>
<td>-Infinity</td>
</tr>
<tr>
<td>1</td>
<td>11..11</td>
<td>00..01</td>
<td>SNaN</td>
</tr>
<tr>
<td></td>
<td></td>
<td>01..11</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>11..11</td>
<td>10..00</td>
<td>QNaN</td>
</tr>
<tr>
<td></td>
<td></td>
<td>11.11</td>
<td></td>
</tr>
</tbody>
</table>
Vector Registers (PPC970)

Figure 2-1. Programming Model—All Registers

1 These registers are 32-bit registers only.
2 These registers are on 64-bit implementations only.
3 These registers are on 4/8-bit implementations only.
4 In 4/8-bit implementations. TSB208 is read as a 32-bit value.
Complexity of CPU cores

**PPC 970 Features**

- **Instruction pipe**
  - 64KB L1 Inst cache, direct mapped
  - 32 entry I buffer
  - 8 instructions fetch / cycle
- **Branch prediction**
  - Highly accurate dynamic prediction
- **Dispatch, issue**
  - 1 group (4 + branch) / cycle
  - Up to 20 active groups
  - Up to 8 issue / cycle
  - Over 200 instructions in flight
- **Data pipe**
  - 32 KB L1 Data cache, 2-way sa
  - 32 x 64b GPR, FPR
  - 32 x 128b VRF
  - 512KB L2 cache, 8-way sa
  - 8 data prefetch streams
Complexity of execution

Figure 1-4 Execution Units and Ports in the Out-Of-Order Core

- Port 0
  - ALU 0 Double Speed
  - ADD/SUB Logic
  - Store Data Branches
  - FP Move
  - FP Store Data
  - FXCH

- Port 1
  - ALU 1 Double Speed
  - ADD/SUB
  - Shift/Rotate
  - Integer Operation Normal Speed
  - FP Execute
  - FP_ADD
  - FP_MUL
  - FP_DIV
  - FP_MISC
  - MMX_SHFT
  - MMX_ALU
  - MMX_MISC

- Port 2
  - Memory Load
  - All Loads Prefetch
  - Store Address

- Port 3
  - Memory Store

Note:
- FP_ADD refers to x87 FP, and SIMD FP add and subtract operations
- FP_MUL refers to x87 FP, and SIMD FP multiply operations
- FP_DIV refers to x87 FP, and SIMD FP divide and square root operations
- MMX_ALU refers to SIMD integer arithmetic and logic operations
- MMX_SHFT handles Shift, Rotate, Shuffle, Pack and Unpack operations
- MMX_MISC handles SIMD reciprocal and some integer operations
Initial programming interface
Programming

- direct programming of modern CPUs?
- use high level languages
- historically: FORTRAN (IBM, early 1950’s)
Fortran Automatic Coding System
APPENDIX B. TABLE OF FORTRAN STATEMENTS

<table>
<thead>
<tr>
<th>STATEMENT</th>
<th>NORMAL SEQUENCING</th>
</tr>
</thead>
<tbody>
<tr>
<td>a = b</td>
<td>Next executable statement</td>
</tr>
<tr>
<td>GO TO n</td>
<td>Statement n</td>
</tr>
<tr>
<td>GO TO i, n, m, n, ...</td>
<td>Statement list assigned</td>
</tr>
<tr>
<td>ASSIGN i TO n</td>
<td>Next executable statement</td>
</tr>
<tr>
<td>GO TO n, n, m, n, ...</td>
<td>Statement n</td>
</tr>
<tr>
<td>IF m &gt; n, n &gt; 0</td>
<td>Statement n, n as a less than, =, or greater than 0</td>
</tr>
<tr>
<td>SENSE LIGHT i</td>
<td>Next executable statement</td>
</tr>
<tr>
<td>IF SENSE LIGHT if n &gt; 0</td>
<td>Statement n as Sense Light i ON or OFF</td>
</tr>
<tr>
<td>IF SENSE SWITCH if n &gt; 0</td>
<td>as Sense Switch i DOWN or UP</td>
</tr>
<tr>
<td>IF ACCUMULATOR OVERFLOW n &gt; 0</td>
<td>as Accumulator Overflow Trigger ON or OFF</td>
</tr>
<tr>
<td>IF QUOTIENT OVERFLOW n &gt; 0</td>
<td>as QO Overflow Trigger ON or OFF</td>
</tr>
<tr>
<td>IF DIVIDE CHECK n &gt; 0</td>
<td>as Divide Check trigger ON or OFF</td>
</tr>
<tr>
<td>PAUSE or PAUSE n</td>
<td>Next executable statement</td>
</tr>
<tr>
<td>STOP or STOP n</td>
<td>Terminates program</td>
</tr>
<tr>
<td>DO i = m, n, or DO n i = m</td>
<td>Next executable statement</td>
</tr>
<tr>
<td>CONTINUE</td>
<td>&quot; &quot;</td>
</tr>
<tr>
<td>FORMAT (specification)</td>
<td>Not executed</td>
</tr>
<tr>
<td>READ n, c, List</td>
<td>Next executable statement</td>
</tr>
<tr>
<td>READ INPUT TAPE i, n, List</td>
<td>&quot; &quot;</td>
</tr>
<tr>
<td>PUNCH n, List</td>
<td>&quot; &quot;</td>
</tr>
<tr>
<td>PRINT n, List</td>
<td>&quot; &quot;</td>
</tr>
<tr>
<td>WRITE OUTPUT TAPE i, n, List</td>
<td>&quot; &quot;</td>
</tr>
<tr>
<td>READ TAPE i, List</td>
<td>&quot; &quot;</td>
</tr>
<tr>
<td>READ DRUM i, j, List</td>
<td>&quot; &quot;</td>
</tr>
<tr>
<td>WRITE TAPE i, List</td>
<td>&quot; &quot;</td>
</tr>
<tr>
<td>WRITE DRUM i, j, List</td>
<td>&quot; &quot;</td>
</tr>
<tr>
<td>END FILE i</td>
<td>&quot; &quot;</td>
</tr>
<tr>
<td>REWIND i</td>
<td>&quot; &quot;</td>
</tr>
<tr>
<td>BACKSPACE i</td>
<td>&quot; &quot;</td>
</tr>
<tr>
<td>DIMENSION x1, x2, ..., x100</td>
<td>Not executed</td>
</tr>
<tr>
<td>EQUIVALENCE x1 = x2, x3, ..., x100</td>
<td>&quot; &quot;</td>
</tr>
<tr>
<td>FREQUENCY m1, m2, ..., m100</td>
<td>&quot; &quot;</td>
</tr>
</tbody>
</table>
IBM System/360
and System/370
FORTRAN IV Language

This publication describes and illustrates the use of the FORTRAN IV language for IBM System/360 and System/370. It is primarily a reference manual for programmers who are familiar with the elements of the language.

FORTRAN IV is a mathematically-oriented language useful in writing programs for applications that involve manipulation of numerical data.
C PROGRAM FOR FINDING THE LARGEST VALUE
C ATTAINED BY A SET OF NUMBERS
DIMENSION A(10)
READ (5,1)(A(I),I=1,10)
1 FORMAT (10/(1E13,2X))
BIG=A(I)
DO 20 I=2,10
20 IF (B(I)-A(I))=0,10,20
10 BIG=A(I)
20 CONTINUE
WRITE (6,2)(B(I),I=1,10)
2 FORMAT (10,10H1THE LARGEST OF THESE 10 NUMBERS IS, 10E13,2X)
STOP
END