CPE300: Digital System Architecture and Design

Fall 2011

MW 17:30-18:45 CBC C316

Arithmetic Unit

10032011

http://www.egr.unlv.edu/~b1morris/cpe300/

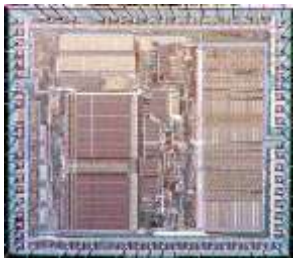
Outline

- Recap Chapter 3
- Number Systems
- Fixed Point Arithmetic

The Motorola MC68000

chip

- Introduced in 1979
  - Computers
    - Apple Lisa 2, Apple Macintosh 128, Atari 520STfm and 1040STfm, Commodore Amiga 500 and 1000
  - Still in use today (now Freescale Semiconductor)
- Very early 32-bit microprocessor
  - Most operations on 32-bit internal data
  - Some operations may use different number of bits
  - External datapaths may not all be 32 bits wide
    - 24-bit address bus for MC68000
- Complex instruction set computer
  - Large instruction set
  - 14 addressing modes



die

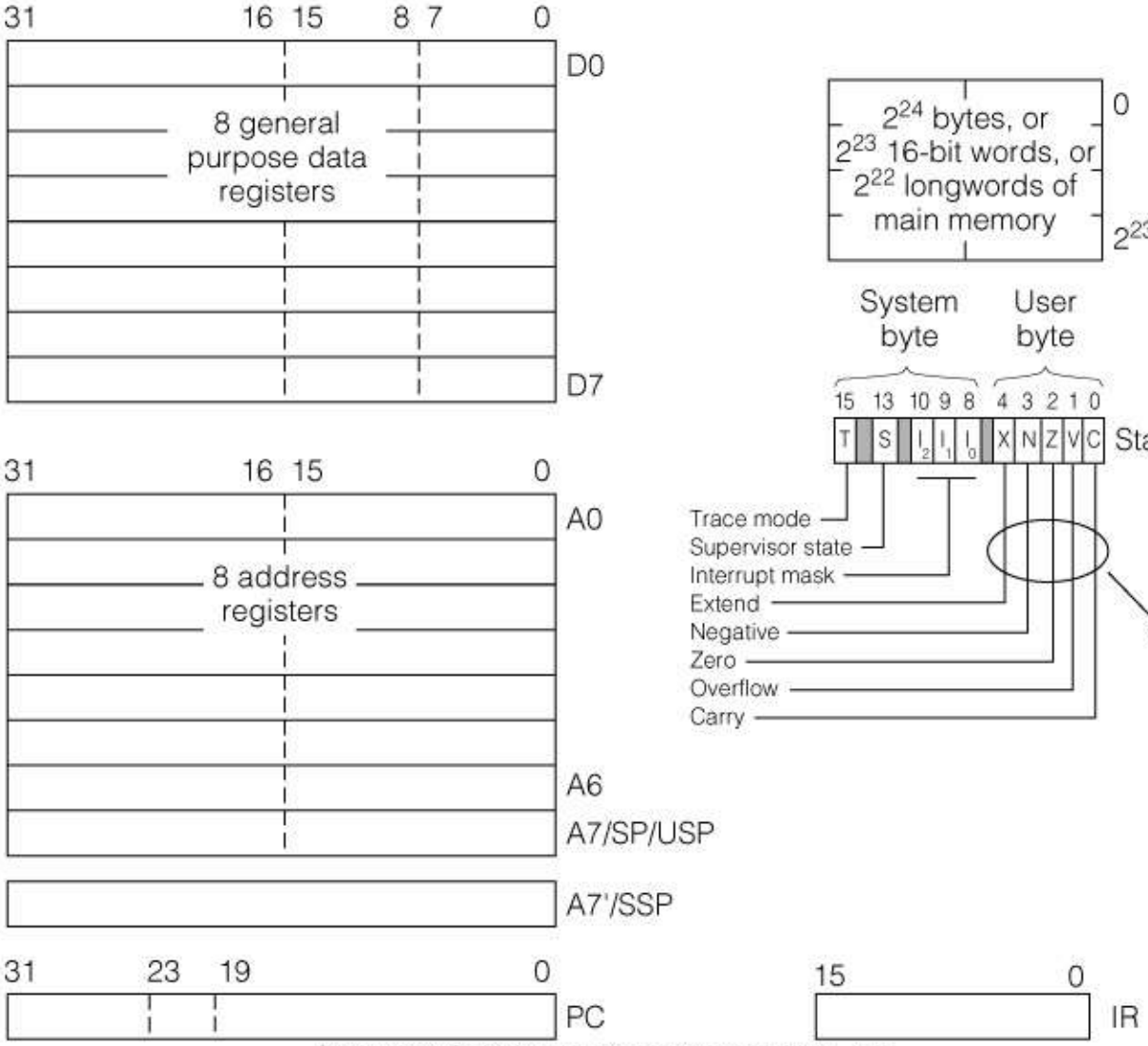
Motorola MC68000 Highlights

- CISC – has many addressing modes and instruction formats
  - Pack as much functionality as possible into small word size
- 16-bit instruction load
  - Some instructions multiple words
- Interrupts and traps (a real machine)
- Memory mapped I/O

New Concepts from MC68000

- Variable length instructions
  - Large instruction set, variable format
- Operation on many different types
  - Must specify byte, word, longword
- Effective address (EA) calculation
  - 14 Addressing modes
- Subroutines
  - E.g. function calls
- Exceptions
  - Interruption of normal sequential instruction execution
- Memory-mapped I/O
  - Part of CPU memory reserved for I/O

MC68000 Programmer's Model



Features of Processor State

- Distinction between 32-bit data registers and 32-bit address registers
- 16 bit instruction register
  - Variable length instructions handled 16 bits at a time
- Stack pointer registers
  - User and system stack pointers
- Condition code register: System & User bytes
  - Arithmetic status (N, Z, V, C, X) is in user status byte
  - System status has Supervisor & Trace mode flags and the Interrupt Mask

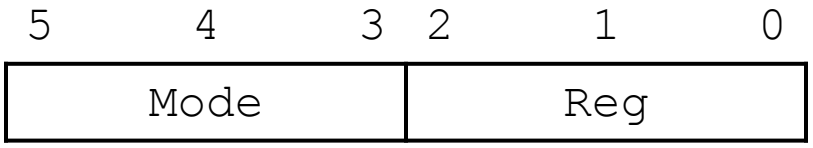
Main Memory

- Main memory:
  - `Mb[0..2^24-1]<7..0>:` memory as bytes
  - `Mw[ad]<15..0> := Mb[ad]#Mb[ad+1]:` memory as words
  - `Ml[ad]<31..0> := Mw[ad]#Mw[ad+2]:` memory as longwords

- Word and longword forms are big-endian
  - Lowest numbered byte contains most significant bit of word
- Hard word alignment constraints
  - Not described in the RTN
  - Word addresses must in end in on binary 0
  - Longword addresses end in two binary 0

Addressing Mode Highlights

- General address of operand specified by 6-bit effective address field



- Modes 0-6 use a register to calculate a memory address
  - Based offset modes (5-6) require an extra word (16-bits) to specify address
- Mode 7 does not use a register
  - Functionality is expanded by repurposing `reg` field
  - All variants require extra words to complete the instruction and specify the memory address

MC68000 Instruction Types

- Instruction fields not standardized
  - Maximize instructions in limited word size (bits)
  - Operates on different types (`B, W, L`)
- Data movement instructions
  - CC can be set during move
- ALU instructions
  - 1 EA, 1 Dn operand
  - Destination specified by 3-bit mode field
- Program control instructions
  - Use 16 condition codes
  - Has subroutine specific instructions

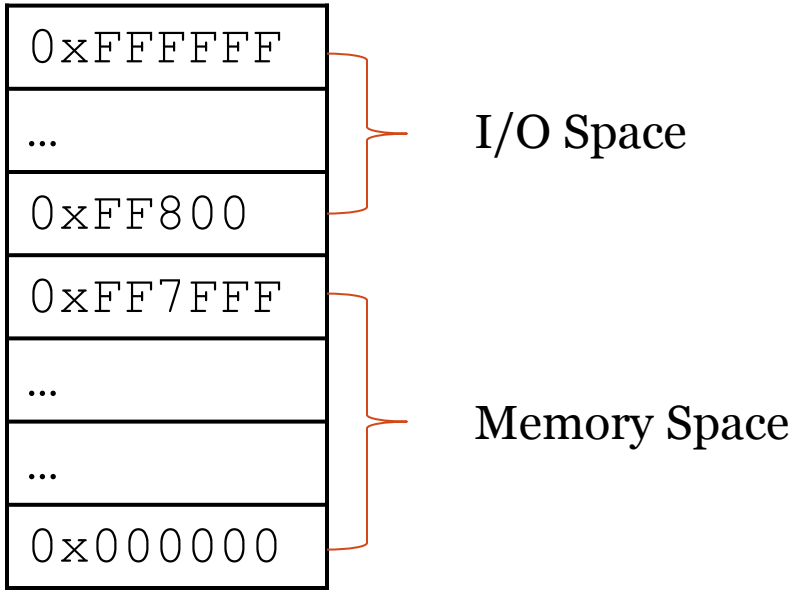
Exceptions

- Changes sequential instruction execution
  - Next instruction fetch not from PC location
  - Exception vector
    - Address supplying the next instruction
  - 7 levels of priority
- Arise from instruction execution, hardware faults, external conditions
  - Interrupts – externally generated exceptions
  - ALU overflow, power failure, completion of I/O operation, out of range memory access, etc.
- Trace bit = 1 causes exception after every instruction
  - Used for debugging

Memory-Mapped I/O

- Part of CPU memory is devoted/reserved for I/O
  - No separate I/O space
  - Not popular for machines having limited address bits
- Single bus needed for memory and I/O
  - Less packaging pins
- Size of I/O and memory spaces independent
  - Many or few I/O devices may be installed
  - Much or little memory may be installed
- Spaces are separated by putting I/O at the top end of address space

24-bit address space with top 32K reserved for I/O



Notice top 32K can be addressed by a negative 16-bit value

The SPARC Microprocessor

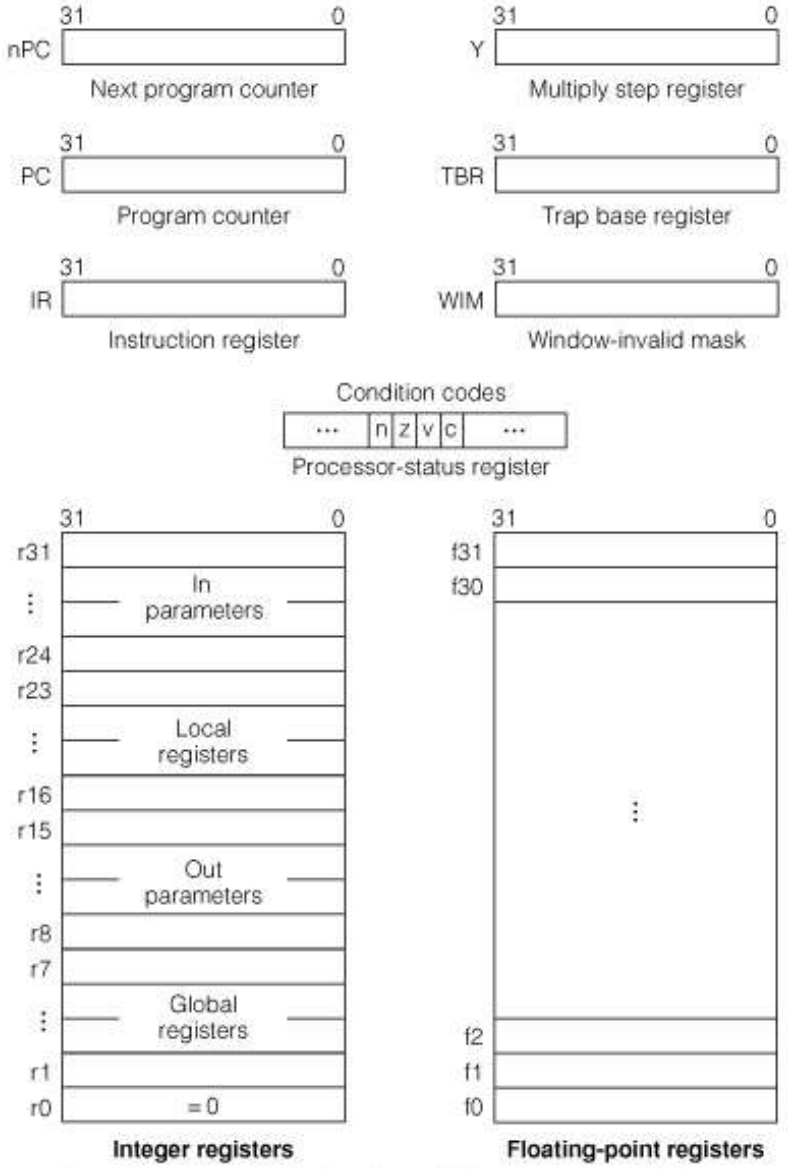
- Scalable Processor Architecture (SPARC)
  - RISC microprocessor architecture
  - Not a machine – specification for implementation
- General register, load/store architecture
- Only 2 addressing modes
  - Reg + Reg
  - Reg + 13-bit constant
- Only 69 basic instructions
  - 32-bit instruction length
  - Separate floating point handling
    - 3 processing units – integer unit, FP unit, coprocesser

SPARC Highlights

- RISC machine has fewer simple instructions
  - Multistep arithmetic operations happen in special units
  - Regular instruction formats and few addressing modes simplify instruction decode
- Load/store machine with ALU only on registers
- Use of branch delays for 4 stage pipeline
- Use of register windows
  - Extend register space for fewer memory operations

SPARC Processor State

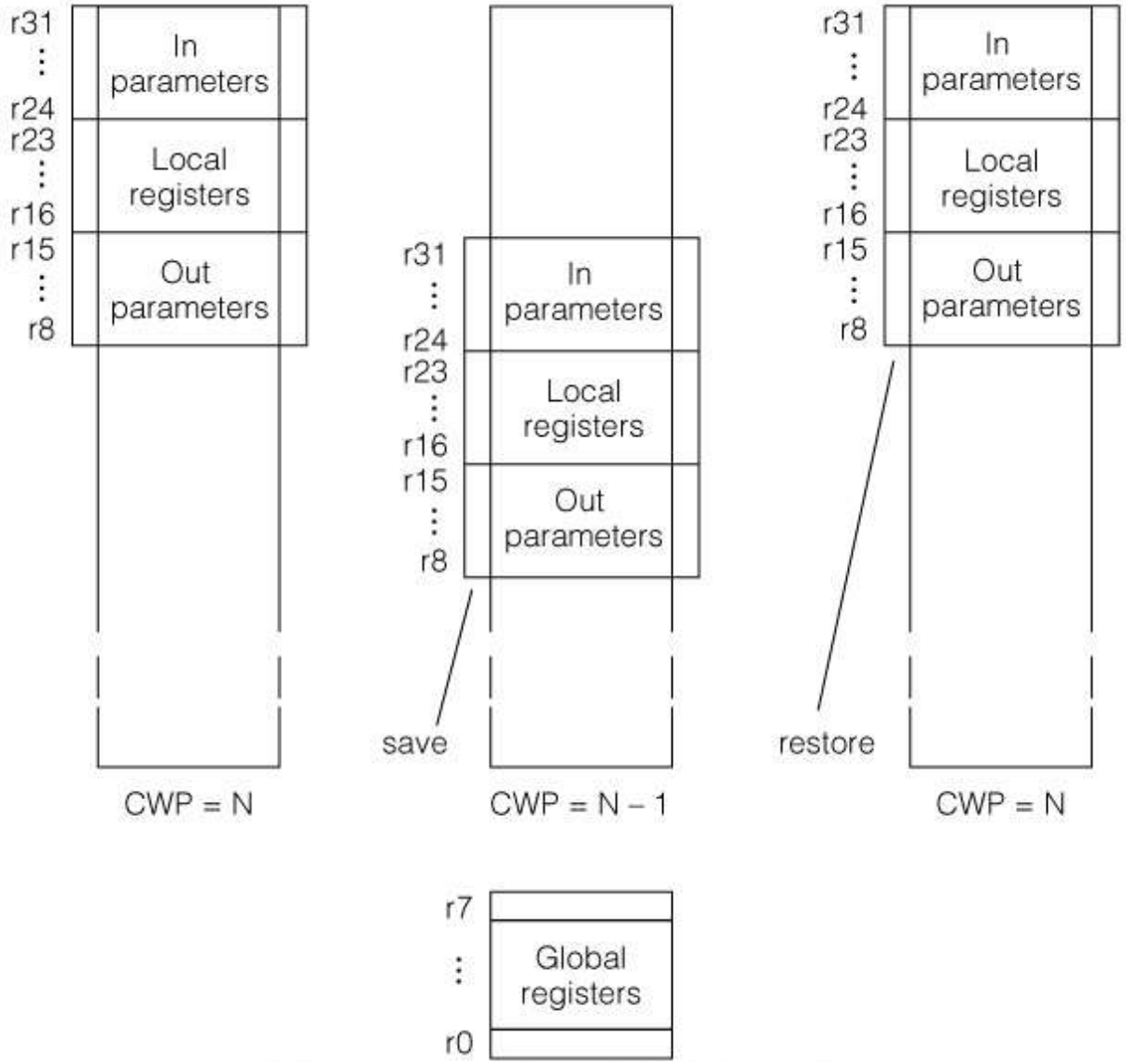
- 32-bit general registers
  - Integer and floating point separate
- Brach delays
  - Requires 2 program counters
- Processor-status register (PSR)
  - Condition codes
- Window-invalid mask (WIM)
  - Used for register windows
- Trap base register
  - Traps and interrupts



Register Windows

- High percentage of memory traffic for saving and restoring registers during procedure calls
  - More registers = less memory traffic
  - Reduce overhead of calls
- Only a small subset of registers is visible to the programmer at a given time (within procedure)
  - Dedicated but overlapping registers groups
    - Global
    - Input parameters
    - Output parameters
    - Local registers
  - Overlap designed to prevent swapping of registers
    - Output parameters in one window become input parameters in the next

Register Windows Mechanism



Window Specifics

- CWP points to register currently called `r8`
  - `save` moves CWP to former r24
  - `restore` reverses process
- Parameters placed in `r24..r31` by caller are available in `r8..r15` by callee
- Spill := attempt to `save` when all windows have been used
  - `save` traps to routine to store registers to memory
  - Window wraps around like a circular buffer
    - On overflow, first window is reused

Main Memory

- Main memory:
  - `Mb[0..2^32-1]<7..0>:` memory as bytes
  - `Mh[ad]<15..0> := Mb[ad]#Mb[ad+1]:` memory as halfwords
  - `Mw[ad]<31..0> := Mh[ad]#Mh[ad+2]:` memory as words

- Word and halfword forms are big-endian
  - Lowest numbered byte contains most significant bit of word
- Hard word alignment constraints
  - Not described in the RTN
  - Word addresses must in end in binary 00

Addressing Modes

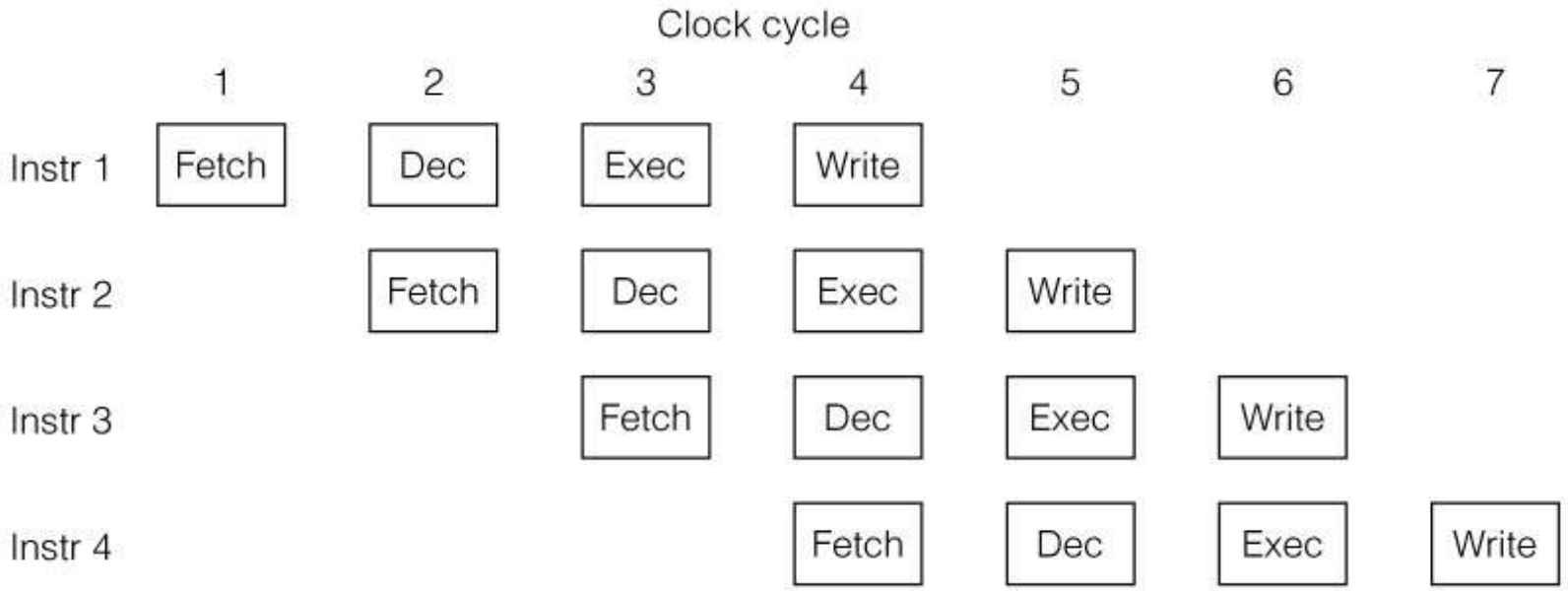
- Only 2 modes for load/store
  - Sum of two registers
  - Sum of register and sign extended 13-bit constant
- Allows for a variety of addressing modes can be synthesized
  - Indexed
    - Base in one register, index in another
  - Register indirect
    - `g0 + rn` `; r0 = 0`
  - Displacement
    - `rn + const.` `; n≠0`
  - Absolute
    - `g0 + const.`
    - Can only reach the bottom or top 4K bytes of memory

RTN for Instruction Interpretation

- ```
  Instruction_interpretation := (
      IR ← Mw[PC] ; instruction_execution;
      update_PC_and_nPC; instruction_interpretation):
  ```

- Notice execution occurs before PC updates
  - 2 PC values to update because of delayed branch
- Interrupts not mentioned in this simple RTN statement
  ```

SPARC MB86900 Pipeline



- 4 stage pipeline
  - Results written to registers in write stage
  - A new inst. is started (issued) before previously issued instructions are complete
  - Instructions guaranteed to complete in order

RISC vs. CISC Designs

- CISC: Complex Instruction Set Computer
  - Many complex instructions and addressing modes
  - Some instructions take many steps to execute
    - Varying lengths of time
  - Not always easy to find best instruction for a task
- RISC: Reduced Instruction Set Computer
  - Pipeline friendly
    - Few, simple instructions, addressing modes
    - Usually one word per instruction
    - May take several instructions to accomplish what CISC can do in one
    - Should be able to finish (nearly) one instruction per clock cycle
  - Complex address calculations may take several instructions
  - Usually has load-store, general register ISA

Problem Solving

- Homework problems
- 3.1
- 3.2
- 3.3

Chapter 6

- Number Systems and Radix Conversion
- Fixed-Point Arithmetic
- Seminumeric Aspects of ALU Design
- Floating-Point Arithmetic

Digital Number Systems

- Expanded generalization of lecture 07 topics

- Number systems have a base (radix) b
- Positional notation of an m digit base b number
  - $x = x_{m-1}x_{m-2} \ldots x_1x_0$
  - $\text{Value}(x) = \sum_{i=0}^{m-1} x_i b^i$

Range of Representation

- Largest number has all digits equal to largest possible base b digit, $(b-1)$
- Max value in closed form for unsigned m digit base b number
  - $x_{\max} = \sum_{i=0}^{m-1}(b-1)b^i$
  - $x_{\max} = (b-1)\sum_{i=0}^{m-1} b^i = (b-1)\left(\frac{b^m-1}{b-1}\right)$
  - $x_{\max} = b^m - 1$

- Sum of geometric series
  - $\sum_{i=0}^{m-1} b^i = \left(\frac{b^m-1}{b-1}\right)$

Radix Conversion

- Conversion between different number systems involves computation
  - Base of calculation is c (10 typical for us humans)
  - Other base is b
- Calculation based on division
  - For integers a and d, exist integers q and r such that
  - $a = q \cdot d + r$
    - $0 \leq r \leq b - 1$
- Notation:
  - $q = \lfloor a/d \rfloor$
  - $r = a \bmod b$ (mod is remainder)

Digit Symbol Correspondence Between Bases

- Each base (b or c) has different symbols to represent digits
- Lookup table given for correspondence between symbols
  - Provides mapping between base b and base c symbols
  - May be more than one digit required to represent a larger base symbol

Base 12	0	1	2	3	4	5	6	7	8	9	A	B
Base 3	0	1	2	10	11	12	20	21	22	100	101	102

Base Conversion 1

- Convert base b integer to calculator base c

1. Start with base b
   - $x = x_{m-1}x_{m-2} \ldots x_1x_0$
2. Set $x = 0$ in base c
3. Left to right, get next symbol x_i
4. Lookup base c number D_i for symbol x_i
5. Calculate in base c
   - $x = x \cdot b + D_i$
6. Repeat step 3 until no more digits

- Example:
- Convert `0x3AF` to base 10
  - $x = 0$
  - $x = 16 \cdot x + 3 = 3$
  - $x = 16 \cdot 3 + 10\ (= A) = 58$
  - $x = 16 \cdot 58 + 15\ (= F) = 943$
- `0x3AF` = 943_{10}

Base Conversion 2

- Convert calculator base c integer to base b

1. Start with base c integer
   - $x = x_{m-1}x_{m-2} \ldots x_1x_0$
2. Initialize
   - $i = 0$
   - $v = x$
   - Get digits right to left
3. Set
   - $D_i = v \bmod b$
   - $v = \lfloor v/b \rfloor$
   - Lookup D_i to get x_i
4. Set
   - $i = i + 1$
   - Repeat step 3 if $v \neq 0$

- Example:
- Convert 3587_{10} to base 12
  - $\frac{3587}{12} = 298\ (rem = 11) \Rightarrow x_0 = B$
  - $\frac{298}{12} = 24\ (rem = 10) \Rightarrow x_1 = A$
  - $\frac{24}{12} = 2\ (rem = 0) \Rightarrow x_2 = 0$
  - $\frac{2}{12} = 0\ (rem = 2) \Rightarrow x_3 = 2$
- $3587 = 20AB_{12}$

Fractions and Fixed Point Numbers

- Base b fraction
  - f= .f_{-1} f_{-2}… f_{-m}
  - Value is integer f_{-1} f_{-2}… f_{-m} divided by b^m
- Mixed fixed point number
  - $x_{n-1}x_{n-2} \ldots x_1x_0.x_{-1}x_{-2} \ldots x_{-m}$
  - Value of n+m digit integer
    - $x_{n-1}x_{n-2} \ldots x_1x_0x_{-1}x_{-2} \ldots x_{-m}$
  - Divided by b^m
- Moving radix point one place left divides by b
  - Right shift for fixed radix point position
- Moving radix point one place right multiplies by b
  - Left Shift for fixed radix point position

Converting Fractions to Calculator Base

- Can use integer conversion and divide result by b^m
- Alternative algorithm

1. Let base b number be
   - $f = .f_{-1}f_{-2}\cdots f_{-m}$
2. Initialize
   - $f = 0.0$
   - $i = -m$
3. Find base c equivalent of D of digit f_i
4. Update
   - $f = \frac{f+D}{b}$
   - $i = i + 1$
5. If $i = 0$, result is f; otherwise repeat step 3

- Example
- Convert 0.413_8 to base 10
  - $f = \frac{0+3}{8} = 0.375$
  - $f = \frac{(0.375+1)}{8} = 0.171875$
  - $f = \frac{0.171875+4}{8} = 0.521484375$
- Notice: there will be precision errors due to numerical round-off
  - Only a fixed number of digits can be retained

Converting Fractions to Base b

1. Start with fraction f in base c
   - $f = .f_{-1}f_{-2} \cdots f_{-m}$
2. Initialize
   - $v = f$
   - $i = 1$
3. Set
   - $D_{-i} = \lfloor b \cdot v \rfloor$
   - $v = b \cdot v\text{-}D_{-i}$
   - Get base b digit f_{-i} for D_{-i} with table
4. Increment
   - $i = i + 1$
   - Repeat Step 3 until
     - $v = 0$
     - Enough digits generated

- Example
- Convert 0.31_{10} to base 8
  - $0.31 \times 8 = 2.48 \Rightarrow f_{-1} = 2$
  - $0.48 \times 8 = 3.84 \Rightarrow f_{-2} = 3$
  - $0.84 \times 8 = 6.73 \Rightarrow f_{-3} = 6$
- Notice:
  - Since $8^3 > 10^2$, 0.236_8 has more accuracy than 0.31_{10}

Digit Grouping for Related Bases

- Base $b = c^k$
- Can convert between bases by replacing single digit symbol in base b with corresponding digits in base c

- (Our favorite method to change base e.g. binary to hex)