ECE243

Computer Organization
Prof. Natalie Enright Jerger

Introduction

Professor Natalie Enright Jerger

PhD, University of Wisconsin - Madison

Intel, IBM

Research:

Processor architecture and memory systems

Multi- and many-core processors

Interconnection networks

Contact info:

enright@eecg.toronto.edu

Office: PT 374A

Office hours: TBA

Purpose of this course

- Understand how a microprocessor system works:

① How to directly program such a system @ the machine code level

② How to expand such systems by connecting H/W to them

③ How typical system is organized & how we build one that works

- Course: 2 thematic sections
  - ① Learn how a computer works
    - How to program it
    - How it behaves
  - ② Learn how to build a processor that works

THIS COURSE IS DIFFERENT

- not about:
  - memorization
  - equations and formulae
- about:
  - gaining experience
  - building systems that work
- like an art class:
  - no mind blowing concepts or math
  - need attention to detail
- but you need to practice!

ECE243 IN THE CURRICULUM

compiled languages (C, C++) ✓ expect you to know

machine code / assembly
processor architecture } 243
gates, flip-flops - √ 241
transistors
quantum effects, electrons

WHY TAKE THIS COURSE?

Embedded system design
Computer architecture & H/W design
S/W design
- performance
Computer systems security

GRADES

- Grade Distribution
  - Labs:15%
  - Design Project:10%
  - Midterm Exam:25%
  - Final Exam:50%
- EXAMS
  - worth 75%!!
  - open book/notes
- Policy on regrade requests:
  - submit cover sheet explaining where and why
  - Entire exam regraded, grade may go up **or down**

LABS

- DE2 board
  - FPGA with NIOS II soft processor, peripherals, host PC
  - peripherals: keypads, LED display, audio, video, robotic lego, other ports
- 3 hours every week (except midterm week)
  - preparation, demo for TA
  - with a partner (chosen first lab)
    - do labs yourself anyway!

Start Jan 17

- PROJECT
  - you decide what to do (meet minimum complexity)
  - last 3 lab periods
  - examples: printer, scanner, elevator, robots (typically involves lego)

COURSE WEBSITE/Materials

- No textbook, no lab manual---everything online
  - Use saved $$$ to buy a DE1/DE2 board! (~$100)
- All course material (CoursePeer) and questions (Piazza)
  - please post **all** questions (rather than email)
  - Do not post any lab code -- this constitutes cheating and will not be tolerated
  - grades will be posted to grade center on BlackBoard

- Handouts
  - General: general info, policy (labs & tests), background
  - Lab: general info, other
  - supplementary materials
- DESL: http://www-ug.eecg.toronto.edu/desl
  - DE2 and NIOS manuals
  - sample codes & diagnostic programs
  - getting DE1/DE2 working at home

WHAT MAKES AN IDEAL PROF?

not hand holding - teach you tools, techniques to problem solve, design, think critically

Resource: expert, ask questions

in class, piazza, office hours - get to know your profs

If you don't understand something ask!

HOW TO SUCCEED AT 243

What makes an Ideal student:

do labs yourself

both you & your partner, combine later

learn to do labs right

read instructions carefully

do exactly what is asked for

know what is going on - you will be quizzed in labs by TA

keep up w/ course material

Number Representation

There are 10 kinds of people in this world: those who understand binary and those who don't.

binary: understands only 2 values
represented as 1 & 0

bundling bits together can represent more things
use n bits to represent 2^n things

map 1s & 0s to meaning

What is this?

10010010

- machine instruction
- audio sample
- image code
- video code
- ascii char.
- floating point value

• memory address

RECALL: BASE 10 (decimal)

956 - trained to interpret as

$9 \times 100 + 5 \times 10 + 6 \times 1$ or

$9 \times 10^2 + 5 \times 10^1 + 6 \times 10^0$

Every digit is multiplied by base 10 raised to power of digit's position

NOTE: not all languages/cultures use base10

Ndom, Frederik Hendrik Island: base6
nimbia: base12
Huli, Papua New Guinea: base15
Tzotzil, Mexico: base20
Computers use base2 (binary)

USING BINARY (BASE 2)

- denoted by '0b' (in this course, in C-code)
- Converting from binary to decimal:

$$0b1011 = 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$$
$$= 1 \times 8 + 0 \times 4 + 1 \times 2 + 1 \times 1$$
$$= 8 + 2 + 1 = 11$$

Questions

- How many decimal numbers can you represent with N bits?

2^N Ex: 2 bits → 4 numbers
4 bits → 16 numbers

- How many binary 'bits' do you need to represent M decimal numbers?

$\text{ceil}[\log_2 M]$
20 nums ⇒ 5 bits (w/ waste)

- NOTE: computers can only represent finite numbers

TWO'S COMPLEMENT

- used to reprsent negative numbers
- converting to 2's complement:
  - invert all the bits, then add one
  - NOTE: the max size in bits is very important!!
- NOTE: most significant bit (MS-bit) is the 'sign bit'
  - for signed representation MS-bit=1 means negative
- **Example (4-bit max size):**

-5

5 = 0b0101

2scomp(0b0101) = 0b1010 + 0b0001
= 0b1011

2scomp(A) + A = 0 (-A + A = 0)

0b1011
+0b0101
0b0000

SUBTRACTION USING 2'S COMP

- to compute A – B, add A to 2's comp of B
- **example (4-bit max size):**

5-3 = 5 + 2scomp(3)

= 0b0101 + 2scomp(0b0011)

= 0b0101 + 0b1100 + 0b0001

= 0b0101 + 0b1101

= 0b0010 = 2

Underflow/Overflow

- Can only represent a fixed range of signed numbers (max size) (n)

$-2^{(n-1)}$ to $2^{(n-1)}-1$

Ex 4 bits: -2^3 to 2^3-1

-8 to 7

- overflow: very large positive number
- underflow: very small negative number
- How can we tell?

Add 2#s w/ diff signs?

under/overflow impossible

detect when result has diff sign than 2#s being added

Ex: 6 + 7 (max size: 4 bit)

0b0110
+ 0b0111
0b1101

check answer:
0b1101 = 0b0010 + 0b0001
= 0b0011 = -3

Sign-Extension

- Convert a 4-bit value into 8-bit value?
- Replicate sign bit (MS-bit) of narrower representation into vacant positions of wider representation

0b0010 = 0b0000 0010

HEXADECIMAL (BASE 16)

- large binary numbers can be a pain:
  0b10010100010111101101
- need a more compact representation
- 'hexadecimal' coined by IBM in 1963
- denoted by '0x' (this course, C-Code)

Converting to Hex

0	0b0000	0x0
1	0b0001	0x1
2	0b0010	0x2
3	0b0011	0x3
4	0b0100	0x4
5	0b0101	0x5
6	0b0110	0x6

Notice:
0b1111 is 0xf
4 binary bits = 1 hex char
hence 1 byte = 2 hex chars

Example: conv. to hex:

7	0b0111	0x7
8	0b1000	0x8
9	0b1001	0x9
10	0b1010	0xA
11	0b1011	0xB
12	0b1100	0xC
13	0b1101	0xD
14	0b1110	0xE
15	0b1111	0xF

0b11010110101101101101

3 5 a e d

0x35aed

REPRESENTING TEXT

- use binary to represent keyboard characters:
  - a,b,c,d…A,B,C,D, !@#$%
- "ASCII" code:
  - 7 bits represent each character
  - example: 'a' = 65 = 0b100 0001 = 0x41

REPRESENTING REAL NUMBERS

- how do we represent 1.75 using binary?
- **Option1: Fixed point:**
  - choose a point in the binary number to be the decimal point
  - Ex: convert fixed-pt to dec:
    - 5-bit max size, 2 bits of whole, 3 bits of fractional

0b01.110 ?

$0\times2^{1}+1\times2^{0}+1\times2^{-1}+1\times2^{-2}+0\times2^{-3}$

$0+1+0.5+0.25+0=1.75$

Note: only accurate to nearest $2^{-3} = 0.125$

Converting decimal to fixed point

- Ex: convert 9.3 to fixed point
  - 8-bit max size: 5 bits of whole plus 3 bits of fractional

X.Y : separately convert X & Y

- Step1: find the 5-bit whole portion:

0b01001

- Step2: find the 3-bit fractional portion:

0.Y - repeatedly multiply by 2 → gives either 0.W or 1.W
binary representation given by integer portion of these #s, repeat w/ 0.W

$0.3 \times 2 = 0.6 \Rightarrow 0$ ← ms-bit

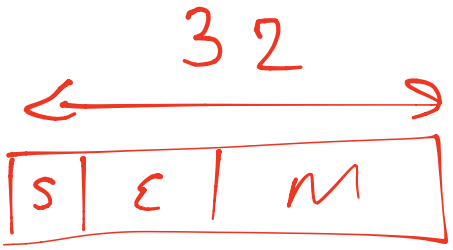
$0.6 \times 2 = 1.2 \Rightarrow 1$
$0.2 \times 2 = 0.4 \Rightarrow 0$
Continue but only have 3 bits

- Final number: 0b01001.010
- Actual value:

$1 \times 2^3 + 1 \times 2^0 + 1 \times 2^{-2}$
$8 + 1 + 0.25 = 9.25$ — not quite original val

Floating Point Representation

32-bit IEEE standard: three parts:



- sign bit (S, 1 bit)
- exponent (E, 8 bits)
- mantissa (M, 23 bits)
- Value = $(-1)^S * 2^{(E-127)} *$ 0b1.mantissa

Ex:

- S = 0b1, E = 0b1000 0001=129,
- M = 0b100 0000 0000 0000 0000 0000

Value = $-1^1 \times 2^{(129-127)} \times$ 0b1.100000....

$= -1 \times 2^2 \times (1 \times 2^0 + 1 \times 2^{-1})$

$= -1 \times 4 \times 1.5$

$= -6$

Converting dec to floating point:

- Ex: 100.625
- whole part in binary: repeatedly divide by 2 & record remainder until result is 0

Remainder

100/2 = 50 → 0 ← least significant bit

50/2 = 25 → 0

25/2 = 12 → 1

12/2 = 6 → 0

6/2 = 3 → 0

3/2 = 1 → 1

1/2 = 0 → 1 ← MS-bit

0b1100100

- Fractional part in binary:

0.625 x 2 = 1.25 → 1 ← ms-bit
0.25 x 2 = 0.5 → 0
0.5 x 2 = 1.0 → 1
0 x 2 = 0 → 0 stop

- final binary value: 0b1100100.101
- binary value with 24 bits of precision:

0b1100100.1010 0000 0000 0000 0

- normalized (to look like 1.something):

shift by 6 ⇒ 0b1.1001 0010 1000 0000 0000 000
x 2^6

- sign bit: S = 0
- exponent: E = 6 = E - 127 → E = 133

133/2 = 66 → 1
66/2 = 33 → 0
33/2 = 16 → 1
16/2 = 8 → 0
8/2 = 4 → 0
4/2 = 2 → 0
2/2 = 1 → 0
1/2 = 0 → 1

133 = 0b1000 0101

- mantissa M: = 1001 0010 1000 0000 0000 000

Floating Point Details

How to represent zero?

special case: if E = 0 and M = 0 (all zeros) means zero

There are many other Special cases:

denormalized numbers

infinity

not-a-number (NaN)