CS 2340 - Computer Architecture

11 Binary Arithmetic, Fixed point

Dr. Alice Wang

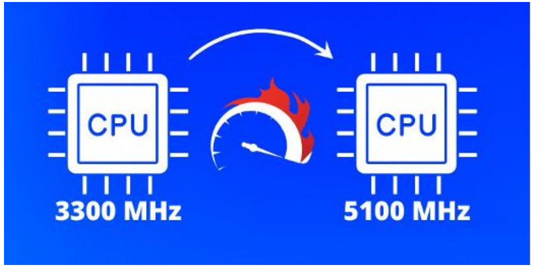
Housekeeping

- Next week Exam 1 on Tues, Oct 7
    - There is a survey planet poll in MS Teams - vote for which topics you want me to cover first
- Exam 1 Review on Thursday, 12-1pm, TI auditorium
    - MS Teams link in Course announcements
- Released Exam 1 Study Guide on eLearning
- All attendance quizzes from Lecture 2-11 are released on eLearning for Exam 1 practice
- Testing center registrations still not at 100%

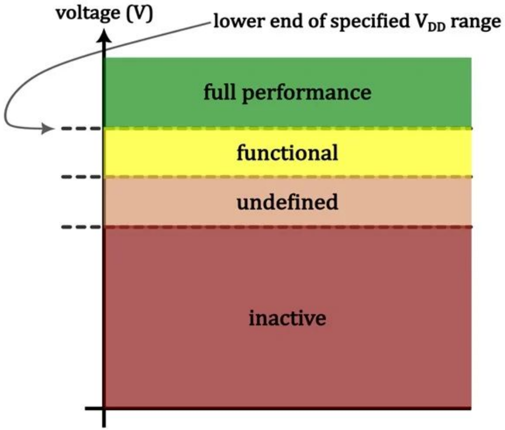
Research

Overclocking and Undervoltage

Increasing a computer's clock frequency (clock rate) beyond its factory settings, boosting performance for tasks like gaming



Decreasing a computer's voltage below its factory settings to help conserve power and reduce heat.



Warning! You may void your warranty & damage your hardware

Review

- CPU time: the best performance measure
- Clock Cycle Time or Clock Period, Frequency or Rate, Instruction Counts (IC), Cycles per Instruction (CPI) all factor into CPU Time

Clock Cycle Time = Clock Period = T_c

Clock Frequency = Clock Rate = 1/ Clock Cycle Time

CPU Time = IC x CPI x Clock Cycle Time

= IC x CPI / Clock Rate

Performance $\propto$ 1 / CPU Time

Binary Arithmetic

Binary Arithmetic (aka how your computer does math)

- Binary Addition and Subtraction
- Binary Multiply and Divide

Fractions: Fixed point

- Unsigned, Two’s Complement
- Decimal to Binary, Binary to Decimal conversions
- Fixed-point math

Review: Binary Addition

Decimal Example: 77 + 66

```
  1      ← carry
  77
+66
 143
```

Binary Example: 7 + 6

```
 0 0 1 1 0    ← carry
 000111
+000110
 001101
```

Binary Subtraction – My Turn

- Binary subtraction uses binary addition
- First apply 2s complement procedure to the second number to change the sign before adding

Binary Example: 7 - 6

Step 1: start with second #
Step 2: flip the bits
Step 3: +1 to the lsb to get -6
Step 4: Add the two numbers
(Check in decimal)

Binary Subtraction – Your Turn

Binary Example: 1100 - 0101

Step 1: take the second #
Step 2: flip the bits
Step 3: +1 to the lsb
Step 4: Add the two numbers
(Check in decimal)

Binary Subtraction (A-B) → Take the 2s complement of B then add to A

Overflow

- Overflow occurs if result is out of range

Operand A Sign	Operand B Sign	Addition Result = A+B	Subtraction Result = A-B
Positive	Positive	Overflow occurs if Result Sign bit = 1 (Result is Negative)	No overflow can occur
Positive	Negative	No overflow can occur	Overflow occurs if Result Sign bit = 1 (Result is Negative)
Negative	Positive	No overflow can occur	Overflow occurs if Result Sign bit = 0 (Result is Positive)
Negative	Negative	Overflow occurs if Result Sign bit = 0 (Result is Positive)	No overflow can occur

Example: A= 0b01000 8

B= 0b01111 +15

A+B =

Multiplication (Decimal vs Binary)

- **Partial products** formed by multiplying a single digit of the multiplier with multiplicand
- **Shifted** partial products **summed** to form result

Decimal

230	multiplicand
x 42	multiplier
460	partial
+ 920	products
9660	result

230 x 42 = 9660

= 230 x 2 + 230 x 40
= 460 + 9200
= 9660

to

Multiplication (Decimal vs Binary)

- **Partial products** formed by multiplying a single digit of the multiplier with multiplicand
- **Shifted** partial products **summed** to form result

Decimal		Binary	
230	multiplicand	0101	
x 42	multiplier	x 0111	
460	partial products	0101	**AND** the multiplier with the multiplicand to get partial products
+ 920		0101	
		0101	
		+ 0000	
9660	result	0100011	**SUM** all partial products

230 x 42 = 9660

5 x 7 = 35

Unsigned Multiply Example – My turn

1) What are the partial products?
2) What is the final product?
3) Does it check out in decimal?

Unsigned

```
  0101
x 0011
```

Unsigned Multiply Example – Your turn

1) What are the partial products?
2) What is the final product?
3) Does it check out in decimal?

Unsigned

```
  1011
x 1001
```

Multiplication - Signed numbers

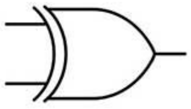
What if one of the number is negative?

If inputs are two's complement N-bit numbers

- Find the magnitudes in positive and perform binary multiply

- XOR the sign bits to get the sign

XOR



A	B	Output
0	0	0
1	0	1
0	1	1
1	1	0

A^B

$A \oplus B$

Signed Multiply Example - My turn

1) Change all numbers to positive
2) Perform binary multiplication
3) Does it check out in decimal?

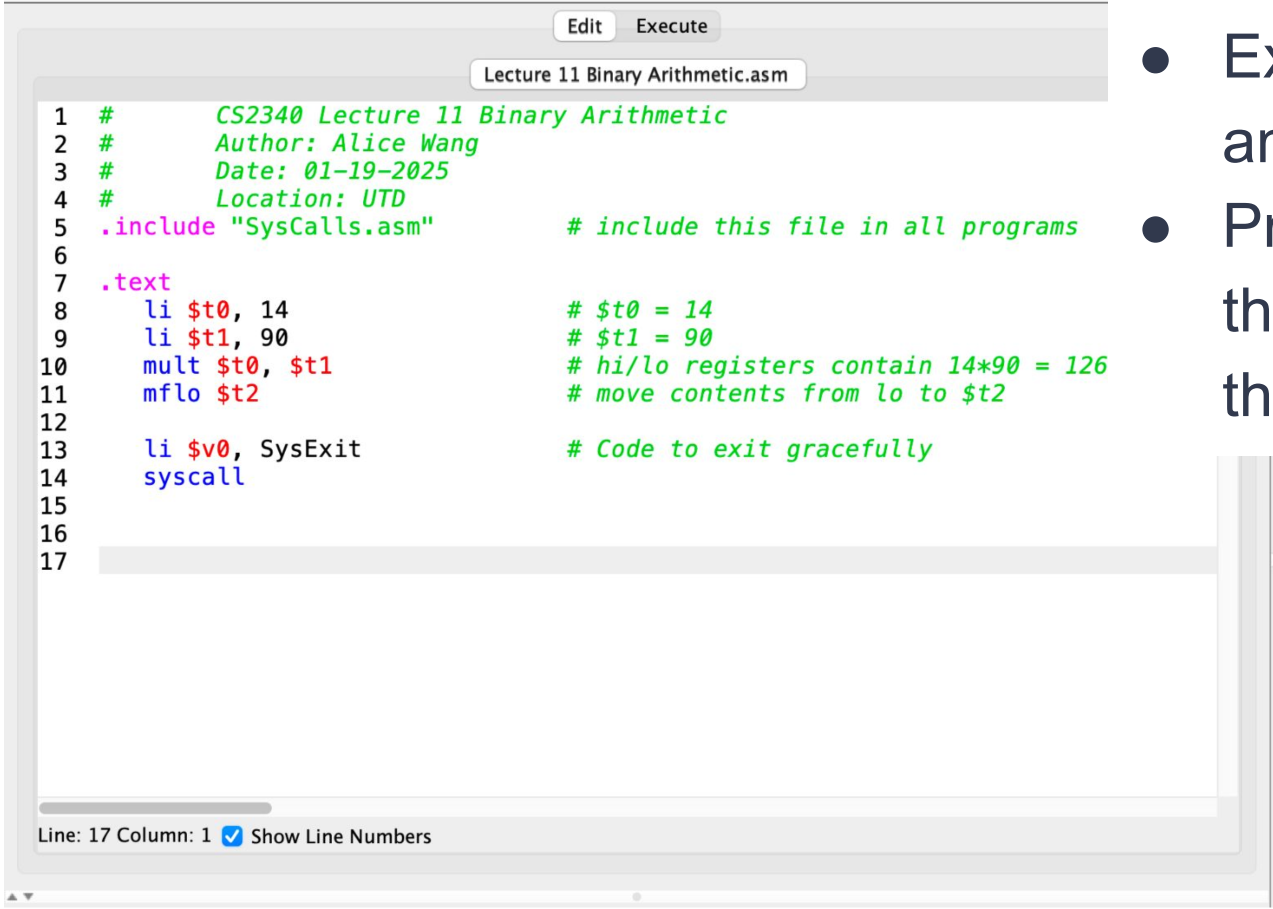
2s complement

```
  1101 →
x 0011 →
```

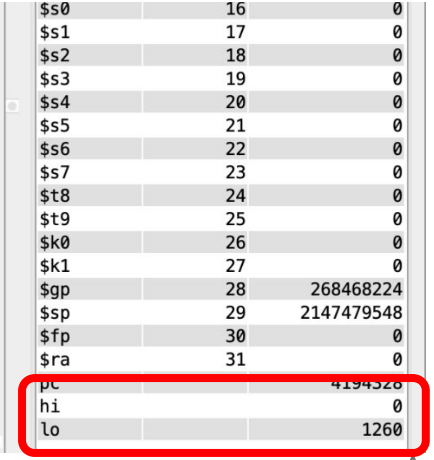
MIPS Multiplication

- One 32-bit x 32-bit multiply produces a **64-bit product**
- Two 32-bit registers for product
  - HI: most-significant 32-bits
  - LO: least-significant 32-bits
- Native Multiply Instructions
  - `mult rs, rt  /  multu rs, rt`
    - 64-bit product in HI/LO
  - `mfhi rd  /  mflo rd`
    - Move from HI/LO to rd
    - Can test HI value to see if product overflows 32 bits

Where are the Hi/Lo registers in MARS?



- Example: Multiply 14 and 90 = 1260
- Product is smaller than 2^{32}, so it fits in the lo register



Where are the Hi/Lo registers in MARS?

Edit | Execute

Lecture 11 Binary Arithmetic2.asm

```
#       CS2340 Lecture 11 Binary Arithmetic 2
#       Author: Alice Wang
#       Date: 01-19-2025
#       Location: UTD
.include "SysCalls.asm"         # include this file in all programs
.text
   li $t0, 200000000            # $t0 = 2x10^8
   li $t1, 500                  # $t1 = 500
   mult $t0, $t1                # hi/lo registers contain 2x10^8x500
   mflo $t2                     # move contents from lo to $t2
   mfhi $t3                     # move contents from hi to $t3

   li $v0, SysExit              # Code to exit gracefully
   syscall
```

Line: 2 Column: 21 ☑ Show Line Numbers

- Example: Multiply $2x10^8$ and $500 = 10^{11}$ the product overflows beyond 32-bits
- The hi register is non-zero

$s4	20	0
$s5	21	0
$s6	22	0
$s7	23	0
$t8	24	0
$t9	25	0
$k0	26	0
$k1	27	0
$gp	28	268468224
$sp	29	2147479548
$fp	30	0
$ra	31	0
hi		23
lo		1215752192

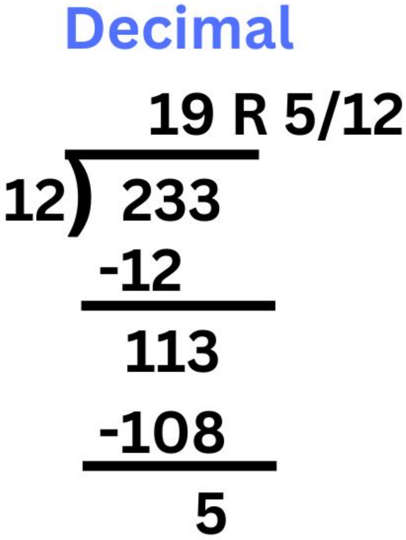
MIPS Multiplication

- Sometimes two steps for multiplication is overkill
- If you know your result will be < 32-bit you can use the following single instruction instead
  - `mul rd, rs, rt`
    - Least-significant 32 bits of product –> rd

- One instruction instead of two (lower instruction count, better performance)

Division (Decimal)

- A/B = Q + R/B → Q: Quotient, R: Remainder (as a fraction of B)
- Example in Decimal : 233 / 12 ⇒ Dividend / Divisor



Division (Binary) – Equivalent to decimal

- Translate this to Binary example (unsigned)
- 1101 / 0010 =

Dividend / Divisor

$$\begin{array}{r} \underline{0110} \\ 0010\,)\,1101 \end{array}$$

quotient

Decimal Check: ___ / ___ = ____ R ____

How your computer does division

Pseudo code

$A/B = Q + R/B$

Algorithm:

```
R' = 0
for i = N-1 to 0
  R = { R' << 1, A_i }
  D = R - B
  if D < 0,   Q_i=0,   R'=R
  else        Q_i=1,   R'=D
R=R'
```

- Uses SW to perform division
- Because hardware division is relatively expensive (area, power) and slow
- Division is rarely done

Division (Binary) – Example – My Turn

$A / B = 1010 / 0100$

Pseudo code

$A/B = Q + R/B$

Algorithm:

$R' = 0$

for i = N-1 to 0

$R = \{ R' << 1, A_i \}$

$D = R - B$

if $D < 0$, $Q_i=0$, $R'=R$

else $Q_i=1$, $R'=D$

$R=R'$

i	R'	R	D = R - B	Q
3				
2				
1				
0				

MIPS Division

- Use HI/LO registers for result
  - HI: 32-bit remainder
  - LO: 32-bit quotient
- Instructions
  - **`div rs, rt  /  divu rs, rt`**
  - Warning: No overflow or divide-by-0 checking
    - Software must perform checks if required
  - Use `mfhi`, `mflo` to access result

Numbers with Fractions

- What do we do about fractions?
- Two common notations:
  - Fixed-point
  - Floating-point
- You're most used to floating point for fractions:

	Integer	Floating point

C and Java	`int myInteger = 10;`	`float myFloat = 3.14f;`
		`// the suffix "f" is for`
		`// single precision float`

Python	`my_integer = 10` `another_integer = -5`	`my_float = 3.14`

What is fixed-point?

- Most microcontrollers don't have floating point hardware (cost $$$)
- Doing math with integers is fast, but dynamic range is limited and fractions are impossible
- Fixed point is a good compromise

8-bit Integer:	0	1	1	1	0	1	0	1.	**Binary point (assumed)**
	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	
8-bit Fixed-pt:	0	1	1	1	0	1.	0	1	**Shift binary point to the left to gain precision, sacrifice range**
	2^5	2^4	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	

Fixed-point number example - *My Turn*

Express 6.75 using 4 integer bits and 4 fraction bits:

$$\underbrace{\overset{\rule{1em}{0.4pt}}{2^3}\ \ \overset{\rule{1em}{0.4pt}}{2^2}\ \ \overset{\rule{1em}{0.4pt}}{2^1}\ \ \overset{\rule{1em}{0.4pt}}{2^0}}_{\text{Integer part}}\ .\ \underbrace{\overset{\rule{1em}{0.4pt}}{2^{-1}}\ \ \overset{\rule{1em}{0.4pt}}{2^{-2}}\ \ \overset{\rule{1em}{0.4pt}}{2^{-3}}\ \ \overset{\rule{1em}{0.4pt}}{2^{4}}}_{\text{Fractional part}}$$

Note: The number of integer and fraction bits must be specified

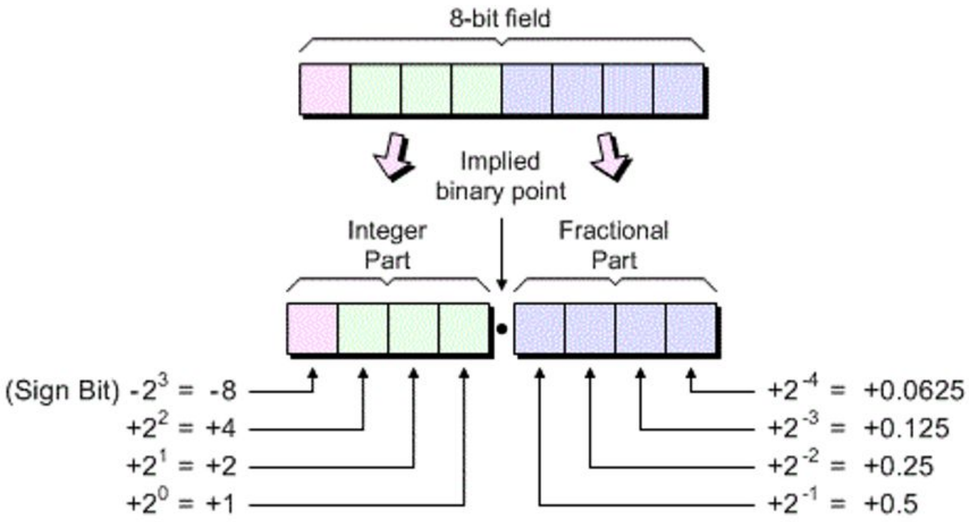
Fixed-point number example - Your Turn

Express 4.125 using 4 integer bits and 4 fraction bits:

$$\underbrace{\underset{2^3}{\underline{\quad}}\ \underset{2^2}{\underline{\quad}}\ \underset{2^1}{\underline{\quad}}\ \underset{2^0}{\underline{\quad}}}_{\text{Integer part}}\,.\,\underbrace{\underset{2^{-1}}{\underline{\quad}}\ \underset{2^{-2}}{\underline{\quad}}\ \underset{2^{-3}}{\underline{\quad}}\ \underset{2^{4}}{\underline{\quad}}}_{\text{Fractional part}}$$

Note: The number of integer and fraction bits must be specified

What about negative fractions?



- Signed fractions in Two’s complement format
  - Sign bit is negative
  - Remaining bits are the magnitude

Signed Fixed-point (Bin to Dec) – My Turn

What is two's complement **0b1100.1101** in decimal?

Integer part: __ __ __ __

-2^3 2^2 2^1 2^0

= ______

Fractional part: __ __ __ __

2^{-1} 2^{-2} 2^{-3} 2^{-4}

= ______

Result = __________

Signed Fixed-point (Bin to Dec) – Your Turn

What is two's complement **0b0100.1001** in decimal?

Integer part: __ __ __ __ **Fractional part:** __ __ __ __

-2^3 2^2 2^1 2^0 2^{-1} 2^{-2} 2^{-3} 2^{-4}

= ______ = ______

Result = __________

Fixed-point (Dec to Bin) - My turn

- Express **10.25** as a two's complement fixed point number
  - 6 integer (including the sign), 3 fractional

Integer part: __ __ __ __ __ __

-2^5 2^4 2^3 2^2 2^1 2^0

Fractional part: __ __

2^{-1} 2^{-2}

Result = ______________

More difficult Fixed-point (Dec to Bin) - My turn

Express **1.234375** as an unsigned fixed point number in binary with 2 integer bits and 6 fractional bits

- Trick: Multiply by 2^6 to get an integer, easier to convert to binary
- Then shift the binary point by 6 places

Multiply 1.234375 by 2^6: ____________________

Convert decimal to binary: ____________________

Shift Binary point left by 6 places: ____________________

Fixed-point (Dec to Bin) - Your turn

Express **6.4375** as an unsigned fixed point number in binary with 4 integer bits and 4 fractional bits

- Trick: Multiply by 2^4 to get an integer, easier to convert to binary
- Then shift the binary point by 4 places

Multiply 6.4375 by 2^4: ____________________

Convert decimal to binary: ____________________

Shift Binary point left by 4 places: ____________________

Fixed-point Addition – My Turn

- Two's complement Fixed Point Addition is similar to Integer Addition
- Before you add, **line up the binary point**
- **Zero-extend** on the LSB side, **Sign-extend** on the MSB side

Decimal Check

```
  01110.01    →
+   101.1     →  + ______
```

1. **Line up Binary point**
2. **Zero-extend on LSB**
3. **Sign-extend on MSB**

Fixed-point Addition – Your Turn

Add the 2 two's complement fixed-point binary numbers 0110.011 and 011.1. Give the result in binary with 4 integer places and 5 fractional places.

Decimal Check

```
  0110.011   →
+   01.1     →  +________
```

1. **Line up Binary point**
2. **Zero-extend on LSB**
3. **Sign-extend on MSB**

Pros/Cons with Fixed-point

- **Pros:**
  - Simple to understand
  - Simple to implement in circuits - fast, low power, low cost
- **Cons:**
  - Limited in bit precision (e.g. 4 fraction bits means precision of 0.0625)
  - Limited in range (e.g. 4 bits for the integer and 4 bits for the fractional part, the range is limited to $[-8, 7.9375]$)

Summary

How your computer does math

- Binary Addition, Subtraction, Multiplication, Division

How your computer does fractions

- Fixed-point
- Signed numbers: Two’s complement

Next Lecture

Floating point