2GA3 Tutorial #6

DATE: October 29th, 2021

TA: Jatin Chowdhary

Midterm #1

- What went wrong?
  - ~66% average
    - It should have been ~86%
- Am I doing something wrong?
  - Some students had trouble calculating the base address, offset, etc.
    - How is this possible? I went over this 3 times
      - Yet, some people still had trouble
      - There are clear-cut examples in the slides
  - Hopefully it was nobody in this class!

Take Action

- **You need to take your education into your own hands**
  - You're paying a lot of money; so extract as much as you can!
- DO whatever works for you
  - Don't listen to others
    - You know yourself better than anyone (I hope)
- Example:
  - *ST##.MP4*

QUESTIONS

???

Solution Manual

- I found one
  - But it's for the previous edition
    - (1st edition) (2017)
- If you want it, here's the link:
  - https://pastebin.com/XiA209tH
    - It's only up/active for 30 days
      - After ~30 days it will be automatically deleted
  - Do not take their answers at face value
    - There are typos/mistakes in the solution manual

Get Out…

- … of your comfort zone
  - If you don't know the answer, just say it
    - Ain't nothing wrong with this
  - If you have a rough idea, but you're unsure:
    - Take a deep breath
    - Slowly explain what your thinking
- NO MORE MUMBLING
- No one gets *it*, the first time
  - Literally no one – I don't care if you're Warren Buffett or Jimmy Buffett.
    - **So if you don't understand something, keep asking until you get it**
- Make a lot of mistakes in class (or in practice)
  - Then you'll make less mistakes on the test, or when it really counts

Feedback

- T01
  - Literally no complaints
    - Sad!
- T02
  - Only complaint: Time management
    - I’m working on it

Invisible Slides

- Used for homework questions
- The answer will be on the next slide (or the same slide)
  - But it will be invisible (kind of)
  - Why? To get you to actually do the question
- Reading the answer is totally different from answering the question from scratch
- Example on next slide

Example (Invisible Slide)

- **Question:** If Johnny had 3 apples, and he gave 2 apples to Timmy, and Timmy had a heart attack. Calculate the mass of the sun and show all of your work.
- **Answer:**
  - The value for G is 6.67×10^{-11} N m^2/kg^2 (where N is Newtons)
  - The distance separating the Earth and the Sun (the orbital radius of the Earth around the Sun), r, is 1.5×10^{8} km
  - The Earth's velocity around the Sun is just the total distance travelled divided by the time required for the Earth to make one complete orbit around the Sun, T
  - The mass of the sun is: 1.98955×10^{30} kg (rounded to 5 decimals)

Unary Number System

- Recall unary is base-1
  - i.e. 111111
    - $1 \times 1^6 + 1 \times 1^5 + 1 \times 1^4 + 1 \times 1^3 + 1 \times 1^2 + 1 \times 1^1$

      $(= 1 + 1 + 1 + 1 + 1 + 1)$
- Can we use a unary number system in a computer?
  - I asked around and got mixed answers
    - Some said yes, others said no, and some said it's pointless
- Turns out…
  - Can have a voltage *difference*
  - Cannot represent 0 or negative numbers without *tricks*
    - Tricks may require compromises
  - Lots of stuff needs to be re-invented
    - i.e. Addition, Subtraction, Logic, etc.

QUESTIONS

???

Hardware VS. Software (1)

- Last tutorial there was a question (3.14) about calculating how many cycles an operation would take in hardware and in software
  - *Answer:* Hardware takes less cycles than software
    - However, both have advantages and disadvantages
- Summary:
  - On next slide

Hardware VS. Software (2)

- Summary:
  - Things implemented in software can be done in hardware
    - What cannot be done? Self-modifying code
  - Hardware implementation yields better performance, and it is more efficient
    - Why? Because less abstraction (closer to the *metal*)
  - It's easy to change software, but difficult to change hardware
    - Changing microcode requires *re-flashing*
      - For consumer electronics this is basically impossible
  - Software implementations are less *reliable*
    - Possibility of more bugs/errors

QUESTIONS

???

Real Life Example

- Apple just released their M1 Plus/Pro/Max chips
  - The battery life/performance is extremely misleading
    - It does not represent real world use
  - They claim *up to* ~20 hours of battery life
    - But their testing methods are flawed!
      - Why?
        - Next slide!

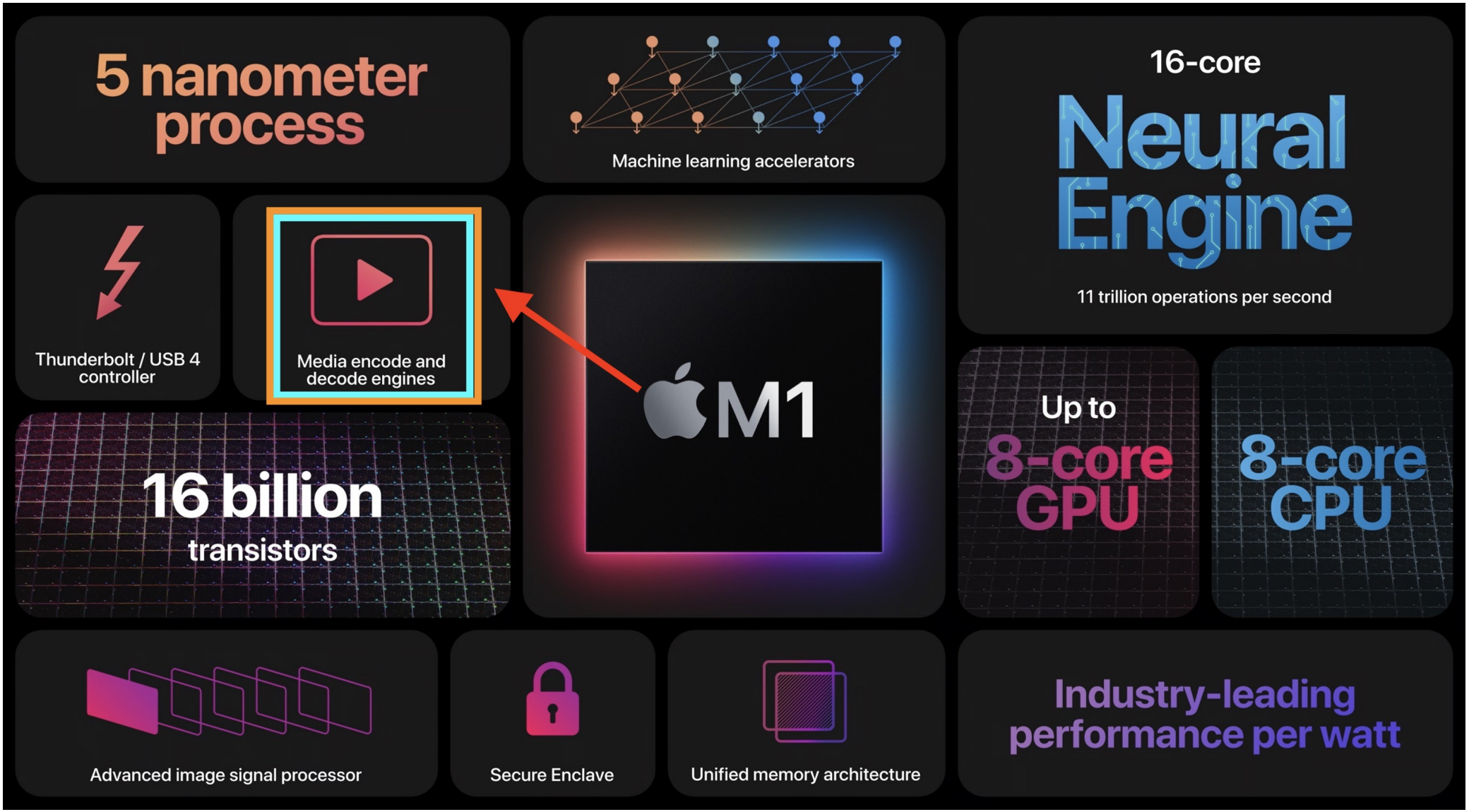
Apple M1 chip	Apple M1 Pro chip or Apple M1 Max chip	Apple M1 Pro chip or Apple M1 Max chip
8-core CPU	Up to **10-core** CPU	**10-core** CPU
8-core GPU	Up to **32-core** GPU	Up to **32-core** GPU
Up to 16GB unified memory	Up to 64GB unified memory	Up to 64GB unified memory
2TB Maximum configurable storage[1]	**8TB** Maximum configurable storage[1]	**8TB** Maximum configurable storage[1]
13.3″ Retina display[2]	**14.2″** Liquid Retina XDR display[2]	**16.2″** Liquid Retina XDR display[2]
Up to 20 hours battery life[3]	Up to 17 hours battery life[4]	Up to 21 hours battery life[5]

* Trade-in values will vary based on the condition, year and configuration of your trade-in device. You must be at least 18 years old to be eligible for credit or to trade in for an Apple Store Gift Card. Not all devices are eligible for credit. More details are available from Apple's Mac trade-in partner. Restrictions and limitations may apply. Trade-in values are based on the received device matching the description you provided when your estimate was made. Apple reserves the right to refuse or limit the quantity of any device for any reason. In the Apple Store: Offer only available on presentation of a valid, government-issued photo ID (local law may require saving this information). Value of your current device may be applied toward purchase of a new Apple device. Offer may not be available in all stores. Some stores may have additional requirements. Full terms apply.

1. 1TB = 1 trillion bytes; actual formatted capacity less.
2. Screen size is measured diagonally.
3. Testing conducted by Apple in October 2020 using preproduction 13-inch MacBook Pro systems with Apple M1 chip, 8GB of RAM and 512GB SSD. The Apple TV app movie playback test measures battery life by playing back HD 1080p content with display brightness set to 8 clicks from bottom. Battery life varies by use and configuration. See apple.com/ca/batteries for more information.
4. Testing conducted by Apple in September 2021 using preproduction 14-inch MacBook Pro systems with Apple M1 Pro, 8-core CPU, 14-core GPU, 16GB of RAM and 512GB SSD. The Apple TV app movie playback test measures battery life by playing back HD 1080p content with display brightness set to 8 clicks from bottom. Battery life varies by use and configuration. See apple.com/ca/batteries for more information.
5. Testing conducted by Apple in September 2021 using preproduction 16-inch MacBook Pro systems with Apple M1 Pro, 10-core CPU, 16-core GPU, 16GB of RAM and 1TB SSD. The Apple TV app movie playback test measures battery life by playing back HD 1080p content with display brightness set to 8 clicks from bottom. Battery life varies by use and configuration. See apple.com/ca/batteries for more information.

- Apple's battery life test does not mimic real world use
- Playing a video on loop until the battery dies is not representative of normal use

- If you thought this was misleading, wait till you see the next slide!



- According to this infographic (from Apple) the M1 chip is able to encode and decode media (i.e. Video) in hardware
  - But wait, didn't we learn that things implemented in hardware have better ***performance*** than their software counterparts
    - We also learned that things implemented in hardware are more ***efficient*** than software

That's Misleading!

- First of all:
  - Apple's battery life test revolves around looping a video for hours until the machine dies
    - This does not represent real world use
  - The M1 Plus/Pro/Max chip has a dedicated engine for encoding and decoding video
    - It's going to be (power) efficient
      - Remember: Hardware implementation takes less **cycles** than software
- So, battery life is going to be phenomenal
  - But this is not accurate; it's misleading

QUESTIONS

???

Review Question #1

- **Question:** Assume we have an *8-bit* number stored in *sign magnitude format*. What is the range of possible numbers?
- **Options:**
  - A. [0, 127]
  - B. [0, 128]
  - C. [0, 255]
  - D. [0, 256]
  - E. [-127, 127]
  - F. [-256, 255]

Explanation For #1

- 8 bit = 2^8 = ???
  - Number is signed, so values can be *negative* or *positive*
- Sign-magnitude format: There are _ ways to represent 0
  - From **-127** to **-0**, there are 128 numbers
    - [**1111 1111**, **1000 0000**]
  - From **+0** to **+127**, there are 128 numbers
    - [**0000 0000**, **0111 1111**]
- 128 + 128 = 256

Review Question #2

- **Question:** Assume we have an *8-bit* number stored in *two's complement*. What is the range of possible numbers?
- **Options:**
  - A. [-128, 127]
  - B. [-127, 127]
  - C. [0, 255]
  - D. [0, 256]
  - E. [-128, 128]

Explanation For #2

- Same logic as before, but:
  - In two's complement, there's only _ way to represent 0
    - Which is all 0's
- From [**-128**, **-1**], there are 128 numbers
  - [**1000 0000**, **1111 1111**]
- From [**0**, **127**], there are 128 numbers
  - [**0000 0000**, **0111 1111**]
- 128 + 128 = 256

Review Question #3

- **Question:** What is the largest possible *unsigned* 10-bit number? What if the number was *signed*?
- **Options:**
  - A. $2^{10} - 1$
  - B. $2^{9} - 1$
  - C. $2^{8} - 1$
  - D. 2^{10}
  - E. 2^{9}
  - F. 2^{8}

Review Question #4

- **Question:** What is the largest possible *unsigned* 16-bit number?
- **Options:**
  - A. 32768
  - B. 32767
  - C. 0
  - D. 65536
  - E. 65535
  - F. $2^{16} - 1$

Review Question #5

- **Question:** Assume we are adding two numbers, *A* and *B*, using *saturation arithmetic*. Both numbers are *unsigned* and *12-bits*. If there is *overflow* (or saturation), what is the result?
- **Options:**
  - A. 4096
  - B. 4095
  - C. 0
  - D. $A + B$
  - E. $2^{12} - A$
  - F. $2^{12} - B$
  - G. None of the above

QUESTIONS

???

Recap About IEEE

IEEE Floating-Point Format

single =: 8 bits
double: 11 bits

single: 23 bits
double: 52 bits

$$x = (-1)^S \times (1 + \text{Fraction}) \times 2^{(\text{Exponent} - \text{Bias})}$$

- S: sign bit (0 ⇒ non-negative, 1 ⇒ negative)
- Normalize significand: $1.0 \le |\text{significand}| < 2.0$
  - Always has a leading pre-binary-point 1 bit, so no need to represent it explicitly (hidden bit)
  - Significand is Fraction with the "1." restored
- Exponent: excess representation: actual exponent + Bias. Ensures exponent is unsigned
  - Single: Bias = 127; Double: Bias = ~~1283~~ 1023

- This is from the lecture slides
- The IEEE FP format is a way to rep. decimal num-bers

Tutorial Question #1

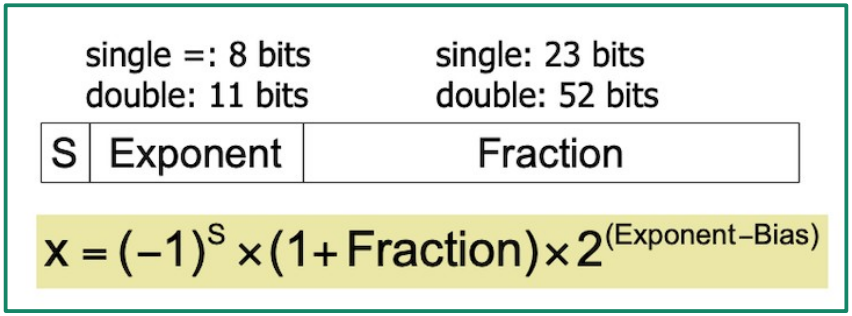
- **Question:** What *decimal* number does the following bit pattern represent? *0×0C000000*
- **Recap:**
  - What is *0x*?
    - Hint: *Hexadecimal*
  - What is the size of this bit pattern? *(In bits)*
    - Hint: Recall that a hex character (letter/number) is 4 bits
  - Convert the number to *binary*
  - 0x**0**C000000
    - The sign is *positive* because the first bit is 0
      - *We know the first bit is 0, because the first 4 bits are 0*

Answer For Question #1

- Convert ***0×0C000000*** to binary
  - 0000 1100 0000 0000 0000 0000 0000 0000
- Remember that the *IEEE FP* format is:
  - $(-1)^S$ **x** (1 + Fraction) **x** $2^{(\text{Exponent}-\text{Bias})}$
    - 1-bit for the sign
    - 23-bits for the fraction
    - 8-bits for the exponent
    - 1 + 23 + 8 = **32**
- Now, let's break this binary number down based on the *IEEE FP format*
  - Next slide!

Answer For Question #1

- The *IEEE FP* format is:
  - $(-1)^S$ **x** (1 + Fraction) **x** $2^{(Exponent-Bias)}$
- Bit-for-bit, the number is:
  - 0 0001 1000 0000 0000 0000 0000 0000 000
- **0**
  - This is the sign bit
    - 0 → Positive (non-negative)
    - 1 → Negative
  - We apply it to the first part of the IEEE FP format:
    - $(-1)^S \rightarrow (-1)^0 \rightarrow$ **1**
  - Now, we have:
    - **(1)** **x** *(1 + Fraction)* **x** $2^{(Exponent-Bias)}$



Answer For Question #1

- **0001 1000**
  - This is the exponent part
  - The number, in *decimal*, is: 24
    - But we need to apply the bias
  - Actual exponent = Biased exponent – bias
    - *Recall: The bias for a 32-bit IEEE FP number is 127 (= $2^7 - 1$)*
      - *What's the point of a bias?*
    - **Actual exponent = 24 – 127 = -103**
- So far, we have:
  - **(1) x** *(1 + Fraction)* **x (2^{-103})**

Answer For Question #1

- **0000 0000 0000 0000 0000 000**
  - This is the fraction
    - Value = x.00000000 (…)
  - But wait! There's a *hidden 1*
    - Remember, it's **(1 + Fraction)**
      - *The mantissa is one bit longer than its sequence of bits written in the number*
- Now, we have:
  - **(1) x (1 + 0.0) x (2^{-103})**
- Putting it all together, we get:
  - **(1.0) x (2^{-103})**

QUESTIONS

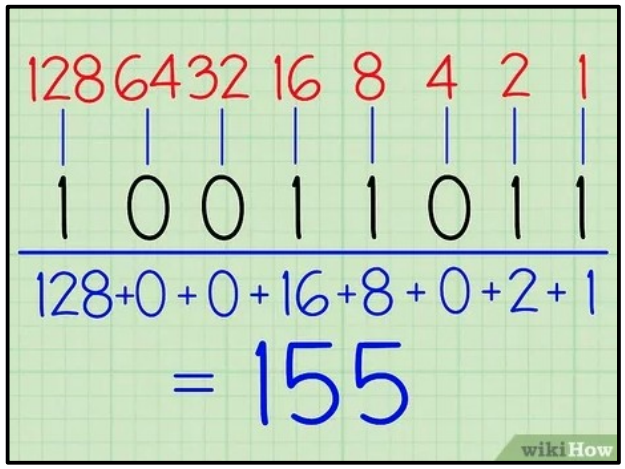
???

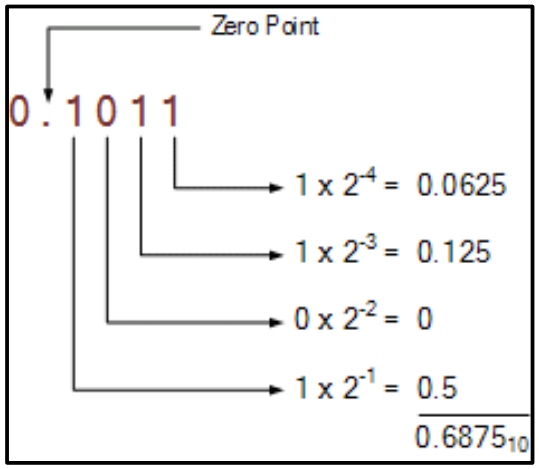
Tutorial Question #2

- **Question:** Write down the binary representation of the decimal number *63.25* assuming the *IEEE 754 single precision format*.
- **Recap:**
  - The number is 63.25
  - Convert to *binary*
    - Using IEEE 754 single precision format
  - Sign is *positive*
  - Bias = 127
    - Because single precision format

Answer For Question #2

- Normalize 63.25
  - 63.25×10^0
- Convert to binary
  - $(63)_{10} = (111111)_2$
    - $2^5 + 2^4 + 2^3 + 2^2 + 2^1 + 2^0$
  - $(0.25)_{10} = (0.01)_2$
    - $0 \times 2^{-1} + 1 \times 2^{-2}$
- Putting these 2 together, we get:
  - **111111.01×2^0**
- Move the floating point to the left, 5 times
  - $111111.01 \times 2^0 \rightarrow 1.1111101 \times 2^5$





- Everytime we move the decimal point to the *left*, the exponent is *incremented*
- If we move the decimal point to the *right*, the exponent is *decremented*
- *It's like we're taking from one side and giving it to the other*

Answer For Question #2

- So far, we have:
  - 1.1111101×2^5
- Recall: What is the sign bit?
  - Positive or negative?
- Now, we calculate the *biased exponent:*
  - *Recall from last question:*
    - *Actual exponent* = *biased exponent* – *bias*
  - **Biased Exponent = Actual exponent + bias**
    - Biased Exponent = 5 + 127 = 132

Answer For Question #2

- Let's put it together!
  - Currently, we have: 1.1111101×2^5
- Remember that the *IEEE FP* format for binary is:
  - S ++ Exponent ++ Fraction
    - Sign bit *(1-bit)*
      - 0 (Because the number is positive)
    - Biased exponent *(8-bits)*
      - 132 = 127 + 5
      - $(132)_{10} = (1000\ 0100)_2$
    - Fraction *(23 bits)*
      - The fraction component is everything to the right of the decimal point
        - $1.\mathbf{1111101} \times 2^5$
      - What happens to the **1**?
        - *"It's a hidden 1"*
          - *(1 + Fraction)*
- **Answer: 0 1000 0100 1111 101 0000 0000 0000 0000**
  - *Fill in the remaining spots with 0's*

- Note: The ++ means concatenation
  - It does not mean addition
- IFF the number was negative, like -63.25, then the sign bit would be 1
- We can add as many zeros as we want to the *right* side of the decimal point
  - The value does not change
    - *i.e. 1.0 == 1.00 == 1.000*
      - *But some people may argue that it affects precision*

Answer For Question #2

- Answer: 0 1000 0100 1111 101 0000 0000 0000 0000
- Convert to hex:
  - Remove spaces:
    - 01000010011111010000000000000000
  - Add a space every after 4th bit, starting from *right* side
    - 0100 0010 0111 1101 0000 0000 0000 0000
  - Every bit corresponds to a hex character (number/letter)
    - 0100 0010 0111 1101 0000 0000 0000 0000

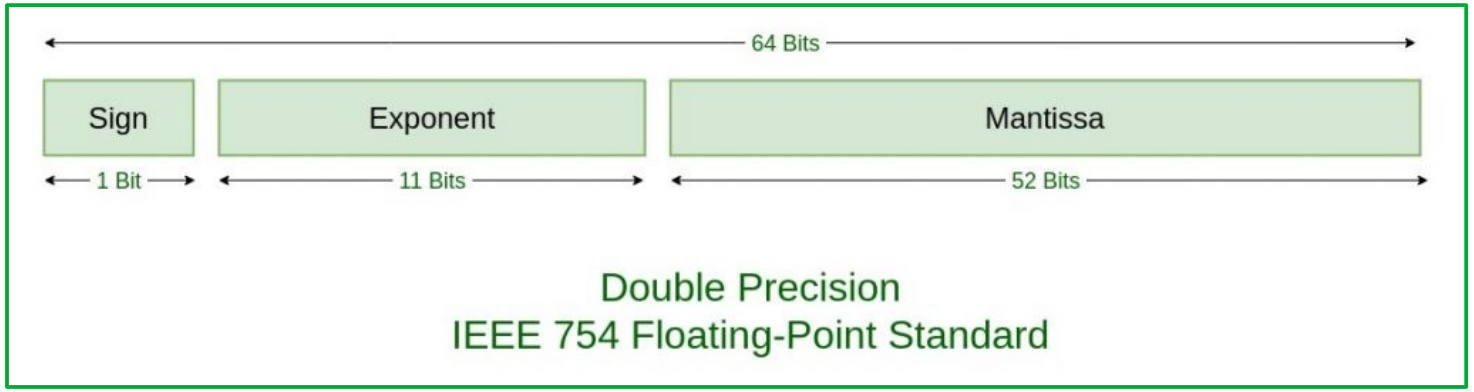
      4 2 7 D 0 0 0 0
  - Final answer (in hex): 0x427D0000
    - *What does 0x mean?*

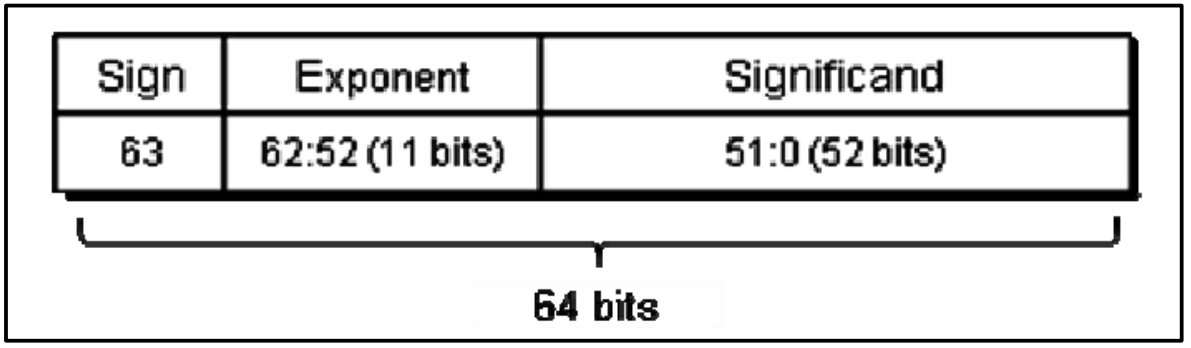
Tutorial Question #3

- **Question:** Write down the binary representation of the decimal number 63.25 assuming the IEEE 754 double precision format.
- **Answer:**
  - Homework!
  - *Logic is similar to previous question*
    - *If you can do single precision format, you can do double*
  - Answer on next slide, but don't blatantly read it!

Answer For Question #3

- **Precursor:**
  - The number given is $(63.25)_{10}$
  - Convert to *IEEE 754 double precision format*
  - For *double* precision:
    - Sign
      - 1-bit
    - Exponent
      - 11-bits
    - Fraction
      - 52 bits
    - **1** + **11** + **52** = 64
    - Bias = 1023
      - $1023 = 2^{10} - 1$





Answer For Question #3

- **Answer (1):**
  - Rewrite:
    - 63.25 → 63.25 **x** 10^0
  - Convert to binary
    - 63.25 **x** 10^0 → 111111.01 **x** 2^0
  - Normalize (move the decimal point)
    - 111111.01 **x** 2^0 → 1.1111101 **x** 2^5
      - *Decimal point moves 5 places to the left*
      - *For every left shift, the exponent increases by 1*
        - *For every right shift, the exponent decreases*
  - Sign (1-bit)
    - 63.25 is *positive*, so sign bit is **0**
      - *If the sign bit is 0, the number is positive*
      - *If the sign bit is 1, the number is negative*
  - Exponent (11-bits)
    - See right side

- **Answer (2):**
  - Exponent (11-bits)
    - Biased exponent = Actual exponent + bias
    - Biased exponent = 5 + 1023 = 1028
      - $(1028)_{10} \rightarrow (10000000100)_2$
  - Fraction = 1.1111101 **x** 2^5
    - *The fraction is 52-bits*
    - *The fraction is everything to the right of the decimal point*
    - *The 1 becomes "hidden"*
  - IEEE Format:
    - **S** ++ **Exponent** ++ **Fraction**
      - *The plus is concatenation, not addition!*
  - Put it all together:
    - **0 10000000100 1111101 0 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000**
  - Convert to hex:
    - 0x**404FA00000000000**

QUESTIONS

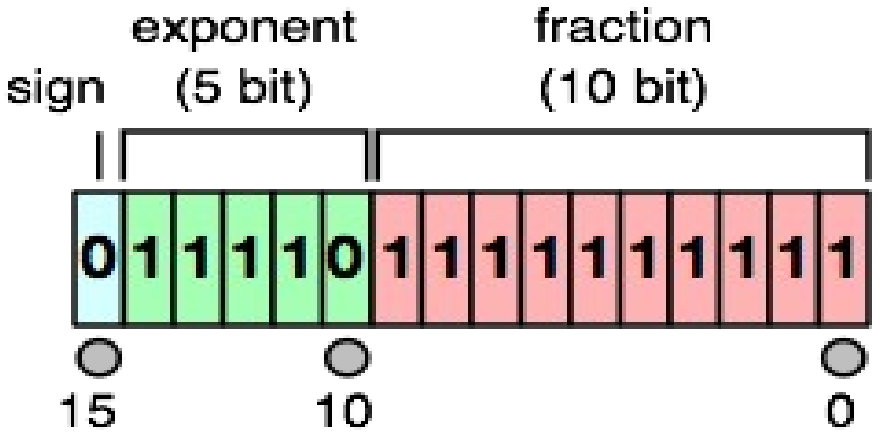
???

Tutorial Question #4

- **Question:** *IEEE 754-2008* contains a half precision that is only *16-bits* wide. The leftmost bit is still the sign bit, the exponent is *5-bits* wide and has a bias of 15, and the mantissa is *10-bits* long. A hidden 1 is assumed. Write down the bit pattern to represent -1.5625×10^{-1} assuming a version of this format, which uses an excess-16 format to store the exponent.

Answer For Question #4

- First of all, we are doing the exact same thing as the previous questions. The only difference is the precision, which is basically the size.
  - Since we are dealing with a 16-bit format, the exponent and fraction will be smaller. However, the sign bit is still 1-bit.
    - Regardless of size, the sign-bit will always be 1.
      - This assumes that the size is big enough to accommodate a decent.
  - So far, you've dealt with single (32-bits) and double (64-bits) precision – now you are dealing with half a precision (16-bits)
    - But, the exact same logic applies!



Answer For Question #4

- **Recap:**
  - The format is IEEE-754 2008
  - 16-bits wide
  - Leftmost bit is sign-bit
    - Sign bit is 1-bit
  - Exponent = 5-bits
    - Bias = 15 = $2^4 - 1$
  - Mantissa = 10-bits
    - Hidden **1** is assumed

Answer For Question #4

- So what's different?
  - Instead of 32-bits wide, it's now 16-bits wide
  - Exponent is 5-bits, instead of 8-bits
  - Fraction is 10-bits, instead of 23-bits
- What hasn't changed?
  - The sign bit
    - It's still 1-bit
- BUT, it's the same logic/format, but different numbers/size
  - Refer to image, below:

Double precision	0100000000001001001000011111101101010100010001000010110100011000
Single precision	01000000100100100001111111011011
Half precision	0100001001001000

Answer For Question #4

- **Answer:**
  - Rewrite (to represent 10^0)
    - $-1.5625 \times 10^{-1} \rightarrow -0.15625 \times 10^{0}$
  - Convert to binary and normalize
    - $-0.00101 \times 2^{0} \rightarrow -1.01 \times 2^{-3}$
      - *When normalizing a binary number, the format should be:*
        - $(1.\text{fraction}) \times 2^{y}$
  - Calculate the bias exponent
    - Biased exponent = Actual exponent + bias

      Biased exponent = $-3 + 15 = 12$
  - Sign bit
    - **–**1.5625 is a negative number
      - Therefore, the sign bit is **1**

- **Answer:**
  - Exponent
    - The biased exponent is 12
    - $(12)_{10} = (01100)_2$
  - Fraction
    - -1.01×2^{-3}
    - The fraction is $(01)_2$
  - Put it all together:
    - S ++ Exponent ++ Fraction
      - 1 01100 01 0000 0000
        - *We fill in the remaining 8-bits with zeros, because it is after the floating/decimal point*
  - Convert to hex:
    - $(\mathbf{1011\ 0001\ 0000\ 0000})_2 \rightarrow (\mathbf{B100})_{16}$

Midterm Review

- Is there time?
  - Question #4 was the *hardest*
- Let's do it from scratch, without referencing solutions
  - *Note: I wrote the solution on the board. If you have any questions, please email me.*

4)
The table below shows some machine code of RISC-V RV32I in memory

Address	+0	+1	+2	+3
0x00000018	00	00	00	00
0x00000014	00	00	00	00
0x00000010	00	00	00	00
0x0000000c	00	00	00	00
0x00000008	13	05	32	00
0x00000004	33	05	52	00
0x00000000	93	42	43	00

Translate machine code at address 0x00000000 into assembly instruction. You need to use RISK V Instruction Set Manual to identify each component of the instruction like "opcode", "rd"...etc.

THE END