CS 33: Computer Organization

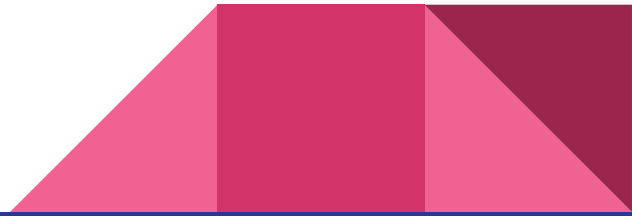
Dis 1B: Week 2 Discussion

Agenda

- Data Lab
- More Data Lab
- x86

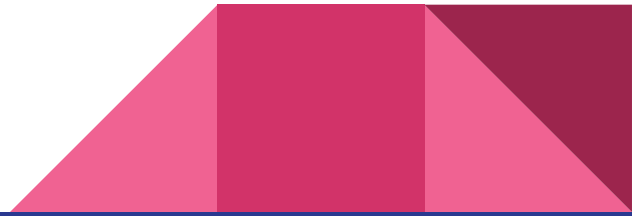
Piazza

- You asked for it, you got it!
- Please post any questions that you have on Piazza



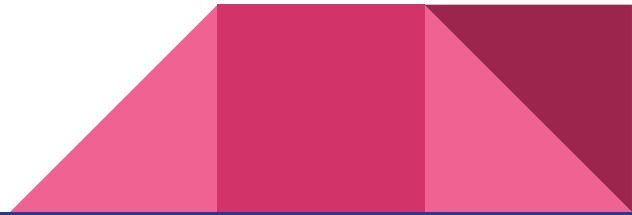
Lab 0

- How was it?
  - Easy? Medium? Hard? 0xFFFFFFFF?
- Lab 0 was intended to help you guys with the setup of Linux environment
- And to teach you guys some basic tricks for Lab 1



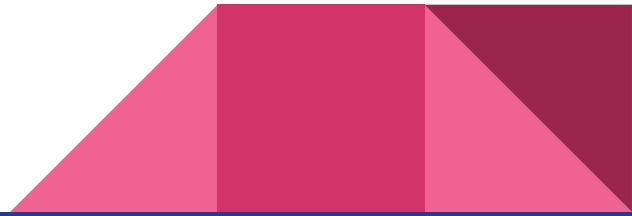
Lab 0 Tricks

- How can you create an “==” operator with “! ~ & ^ | + << >>”
- What happens if you use logical operators to numbers?
- How can you check if a number is 0 or not?
- How can you convert a positive number to a negative number?



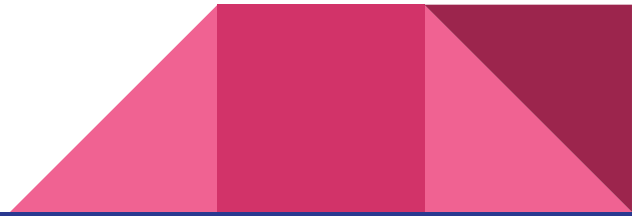
Lab 1

- Uploaded on SEASnet
- cp -r /w/class.1/cs/cs33/cs33w17/lab1-handout .
- 11 problems
  - howManyBits is an extra credit problem
- Much harder than Lab 0
  - Probably the hardest lab for CS 33
  - If you had a hard time with Lab 0… Start NOW!



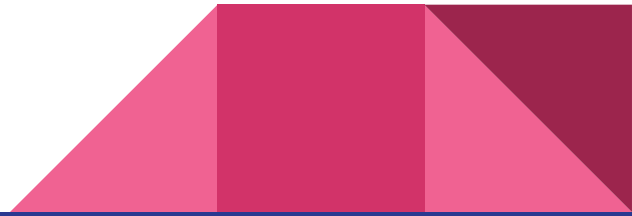
Lab 1 Tricks

- Exercise 1
  - How can you create an “&” operator with “~ |”?
- Exercise 2
  - How can you create an “|” operator with “~ &”?



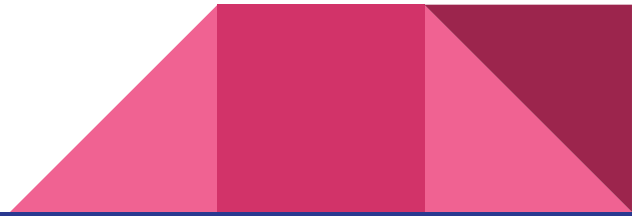
Lab 1 Tricks

- Exercise 3
  - Write a function that takes in an integer and returns the 2 least significant bits of an integer
  - ex) 0x12345678 returns 00_2
  - ex) 0xFF12AA55 returns 01_2
  - ex) 0x12FF55AA returns 10_2
  - ex) 0xFFFFFFFF returns 11_2



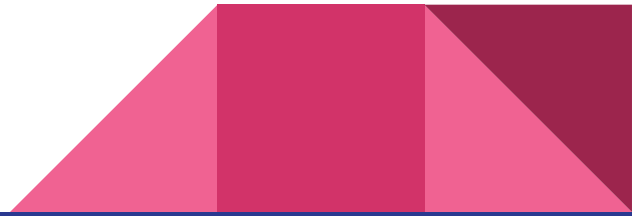
How can we create a constant?

- Data lab spec says…
  - Integer constants 0 through 255 (0xFF), inclusive. You are not allowed to use big constants such as 0xFFFFFFFF



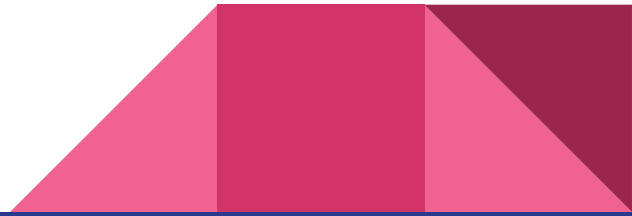
Lab 1 Tricks

- Exercise 4
  - How can you create a number -1?



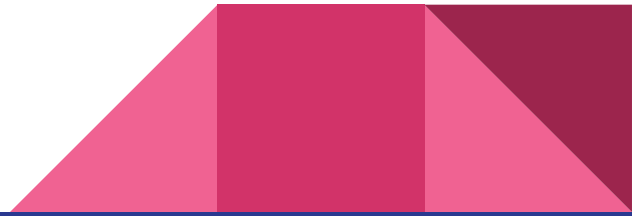
Lab 1 Tricks

- Exercise 5
  - How can you create a number 0xFFFF0000?



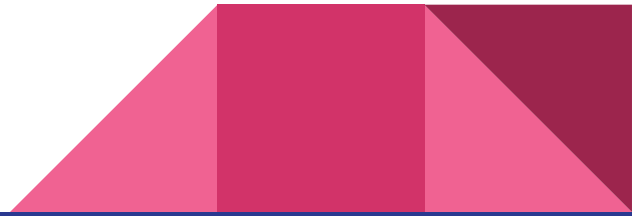
Lab 1 Tricks

- Exercise 6
  - Write a function that takes in an integer and returns the 8 most significant bits of an integer



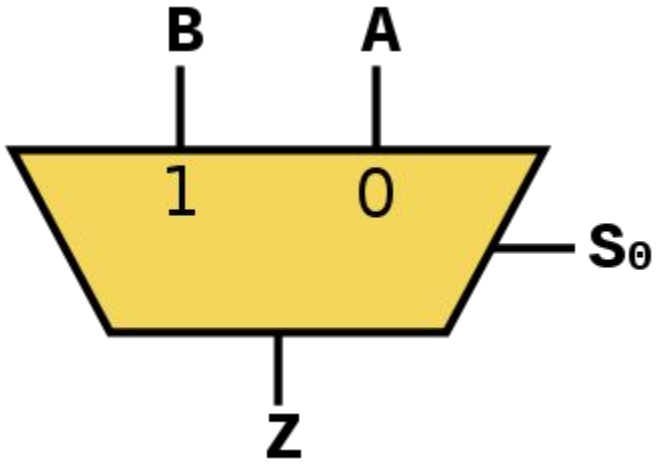
What if I tell you...

- There is a way of creating if-statement with “! ~ & ^ | + << >>”



What is this?

- How many EE majors do we have?
- Tell me what this thing is and what it does please



If-statement in Data Lab

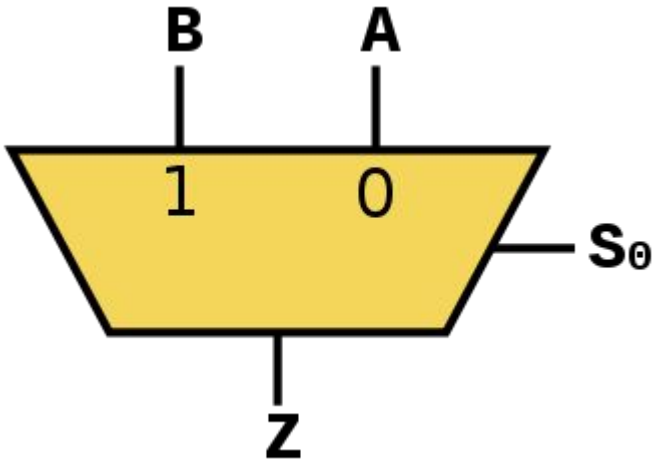
- if (S0)

  Z = B;

  else

  Z = A;

  return Z;



If-statement in Data Lab

- if (S0)

  Z = B;

  else

  Z = A;

  return Z;

- When do we want to return B?
- When do we want to return A?
- How can we combine these previous two statements?

If-statement in Data Lab

- if (S0)

  Z = B;

  else

  Z = A;

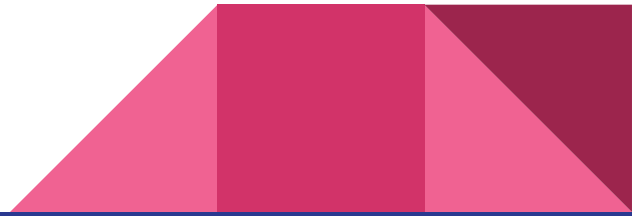
  return Z;

- When do we want to return B?
- When do we want to return A?
- How can we combine these previous two statements?
- return (S0 & B) | (~S0 & A)

Machine-level Programming

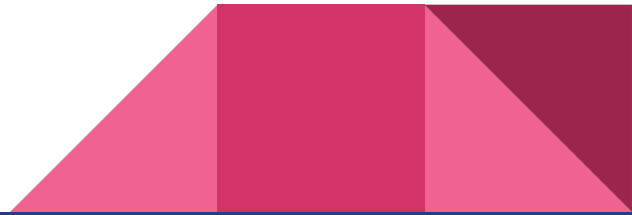
Memory

- Memory is a huge physical container
- However, it is too slow to keep up the demands of a processor



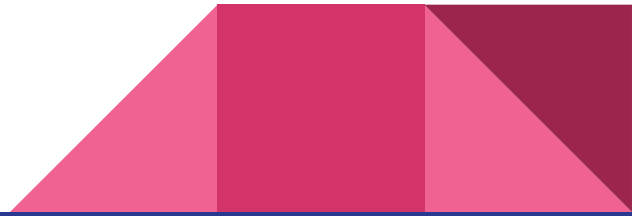
Registers

- Registers are extremely small physical containers that each store a number of bits
- A 64-bit addressable machine will have registers that are 64-bits
- In a such case, each register only able to hold up to 8 bytes, but the access time is extremely fast
- When a program needs to work on a piece of data, it will bring it into a register first
- x86-64 contains 16 general purpose registers



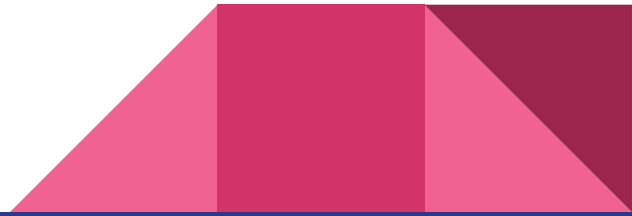
x86 Assembly Syntax

- [operation] [source] [destination]
- ex)
  - movq %rax %rbx
  - addq %rbx %rcx



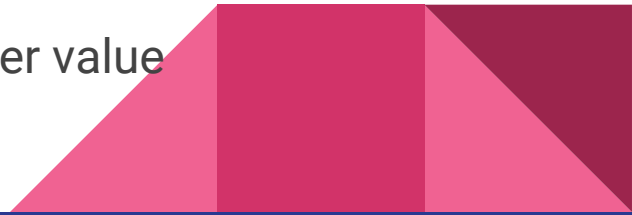
mov instruction

- mov instruction: moves data from the source to the destination
- movb: move a byte
- movl: move 32 bits
- movq: move 64 bits



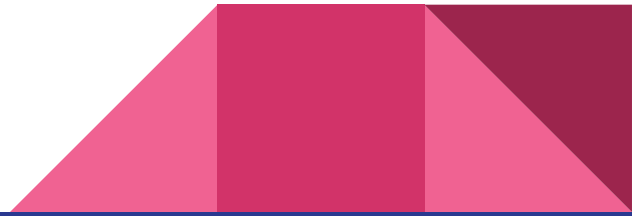
movq instruction

- movq %rax, %rbx
- movq %rax, (%rbx)
- movq (%rax), %rbx
- The parentheses indicate a memory operation
  - Treat the bit vector inside of a register as a memory address
  - Follow that address into memory and get the value at that address
- Memory to memory transfer is not allowed in a single instruction!
- ‘$’ indicates an “immediate” which is constant number value
  - movq $0xFFFFFFFF, %rax



Addressing Mode

- D(Rb, Ri, S)
  - movq $0xAAAAAAAA 8(%rax, %rbx, 2)
  - Data stored in Rb + data stored in Ri * S + D



More instructions

`addq`	*Src,Dest*	Dest = Dest + Src
`subq`	*Src,Dest*	Dest = Dest – Src
`imulq`	*Src,Dest*	Dest = Dest * Src
`salq`	*Src,Dest*	Dest = Dest << Src
`sarq`	*Src,Dest*	Dest = Dest >> Src
`shrq`	*Src,Dest*	Dest = Dest >> Src
`xorq`	*Src,Dest*	Dest = Dest ^ Src
`andq`	*Src,Dest*	Dest = Dest & Src
`orq`	*Src,Dest*	Dest = Dest | Src

Also called shlq

Arithmetic

Logical

leaq instruction

- leaq [src] [dest]
- src is an addressing mode expression
- Set dest to address denoted by expression
- leaq 4(%rax, %rbx, 2), %rcx