CPSC 240 notes

<http://www.egr.unlv.edu/~ed/assembly64.pdf> - textbook

[Week 1 - Intro to assembly1](#_f5mxxeav9hvi)

[Day 1:](#_w809zq9iqjrq)

[Day2: compile, linker, cpu architecture.](#_ripay4brj0d0)

[Week 2: Programming + how numbers are represented internally](#_3bs5ico5u5mf)

[Day 1:](#_cjwy8d62etm3)

[Day 2:](#_39iiuni3mlnw)

[Week 3 Terminology and functionality of registers:](#_msp9i23mb2zt)

## Week 1 - Intro to assembly1

### Day 1:

* Hybrid not pure assembly in this course
* Designing programs in blocks
* Assignment 1 - allow user to input 2 numbers
  + Compare which of the two is smaller and send it back to the driver
  + Bye stuff
* Driver -> control -> isFloat + atof
  + Idea:
  + Make spots for 2 floats on stack
  + Get inputs for them and store them
  + Call isFloat on those to confirm or crash program
  + If both are numbers then convert the string to float and store in xmms
  + If a > b return a
  + Else return b
* Imports and exports + constants above segment.data
* Segment.data - “variables”
  + Strings db …. 10, 0
  + 10 new line, 0 null
* Segment.bss - unintialized arrays
* Segment.text - user assembly code
  + Name of function first line
* Function arguments - rdi, rsi, rdx, rcx, r8, r9 …
* Rbp - address of back end of stack/AR
  + push rbp (put the current address of the back end of the stack)
  + mov rbp, rsp (move into rbp the current address of the top of the stack, which will be the back end of the current AR)
  + 8 bytes - 64 bits

Mov rax, 0 on code blocks - explain this - how many xmm used

Push qword 0

Mov rax, 0

Mov rdi, string\_form

Sub rsp, 1024 (grows the stack, adding goes backwards)

Mov rsi, rsp

Call scanf

Pop rax

Mov rdi, rsp

Call isFloat (returns a bool in rax)

Cmp rax, 0 (set up these two for comparison)

Je Handle\_False (written as another function below)

; handle it’s a float (atof)

Movsd xmm0, rsp + whatever (depends on where the string is 1024, 8, etc…) (no deref necessary)

Call atof

Movsd xmm9, xmm0

Handle\_False:

; tell user to rerun

Low xmm nums are volatile (change often), you want to use xmm15 and work downwards

ALU

### Day2: compile, linker, cpu architecture.

To compile:

G++ -c -o something.o -std=c++17 your\_file.cpp

Nasm -f elf - m64 -o control.o controller.asm -l control.lis

G++ -o final -std=c++17 tf.o control.o isfloat.o -f no-pie

* Just put the output binaries at the end of the linker

CPU architecture



ALU - arithmetic logic unit

16 gprs - r (64 bit), e (32 bit)

FPU - 8 registers (80 bits a register)

SSE = Streaming SIMD Extensions - SIMD - Single Instruction Multiple Data

Xmm … registers

Using the low qword (64 bits). Xmm - 128 bits

AVX - Advanced vector extensions - ymm15…ymm0

* Split in 4 64 bit

Important registers

Rsp - top of the stack (address)

Rbp - back of current AR (address)

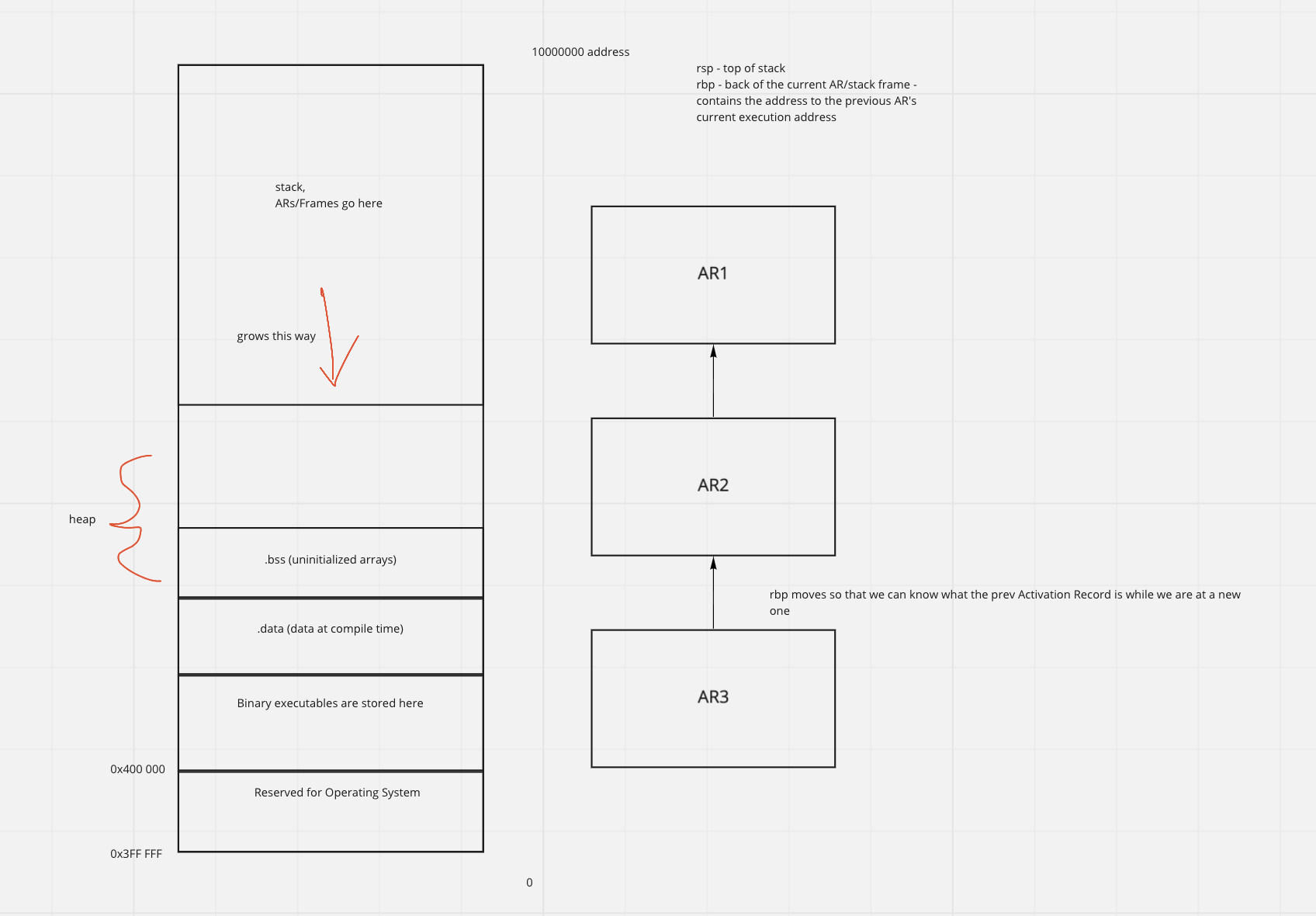
Rax - garbage collector/return values non-float

Rip - address of next instruction

8 bytes at a time not bits

Stack





Stack

Space

data

Mov rax, 0

Mov rdi, format

Push qword 0

Mov rsi, rsp

Call scanf

Movsd xmm15, [rsp]

Mov rax, 0

Mov rdi, stringformat

Sub rsp, 2048

Mov rsi, rsp

Call scanf

; validate the input

Mov rax, 0

Mov rdi, rsp

Call isFloat

Cmp rax, 0

Je endprogram

Mov rax, 0

Mov rdi, rsp

Call atof

Movsd xmm15, xmm0 (save the value returned to xmm0)

Jmp end

Endprogram:

; goodbye

Jmp done

End:

print(“success”)

done:

Add rsp, 2048

Movsd xmm0, xmm15

Ret

## Week 2: Programming + how numbers are represented internally

### Day 1:

Ja, jb, jbe, jae - xmm ?

Jl, jg, jge, jle - gpr ?

Xor - exclusively or

Pie - position independent execution bc nasm only uses object files that are non-pie

M64 - 64 bit machines

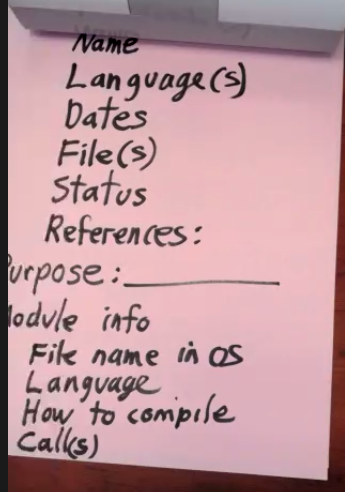
-c compile don’t link

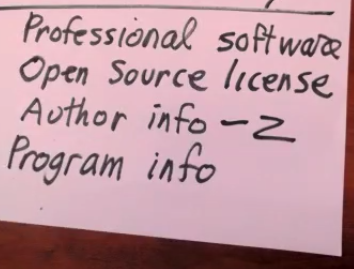
Wall show all warnings

-o object file name

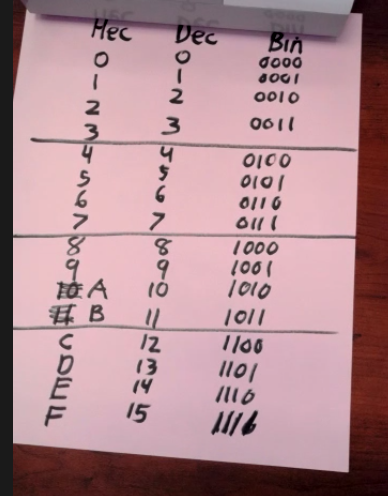
### Day 2:

What to put at the top of program



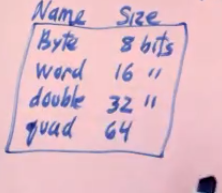


Different representations for numbers:



1 hex - 4 bits

Chart of bits to name terminologies



1 byte = 2 hex digits

23497 - decimal

Converting to hex - base\_num = 16

23497/base\_num continually, taking remainder

23497/16 = 1468 R**9**

1468/16 = 91 R**12**

91/16 = 5 R **11**

5/16 = 0 R **5**

= 0… 5 B C 9

64 bit register has 16 hex numbers (64/4)

Qword - 8 bytes (8\*8 = 64 bits)

Doubleword - 4 bytes (64/2 = 32 bits)

Word - 2 bytes (64 bits/4 = 16 bits)

## 

MSD, LSD(lost during rounding)

“Which digit contains the most ‘weight’?”

Natural Order

Internally, read numbers from right to left

Big endian - MSD first (right to left)

**Little endian - LSD first (right to left)**

**Lscpu - shows info about ur processor**

How to read the LIS



The hex represents different operations/registers

0x80 = 0000000000080

80000000000000 in LIS

Basically Flip 2 hexes at a time (a byte)

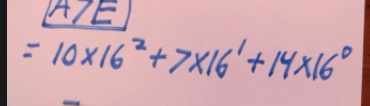
LIS is left to right

That’s why

0x 4048 78FF

In LIS is FF 78 48 40

Both are little endian but read in different “natural order”



CISC vs RISC

**Complex instruction set computer - x86 - 709 assembly instructions**

* **Many instructions overlap eachother**

Reduced Instruction set computer

* No duplication of assembly instructions

Chris Sawyer

Roller Coaster tycoon 2

1:1 correspondence - action in ALU and line of code

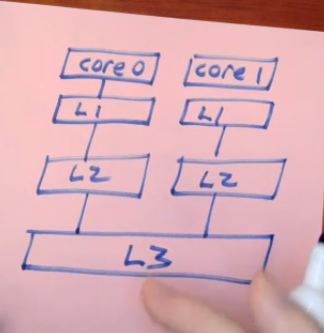
GPRs

2.3 in the book

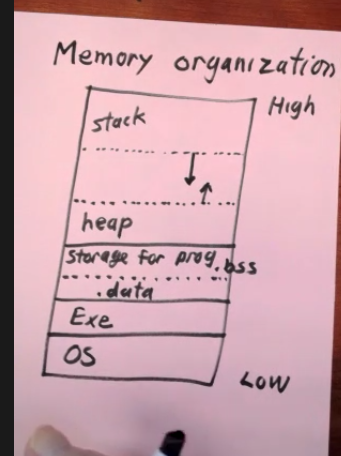
Rbx - ebx - bx - bl

Rsp - esp - sp - spl

R15, r15d, r15w, r15b



Memory organization

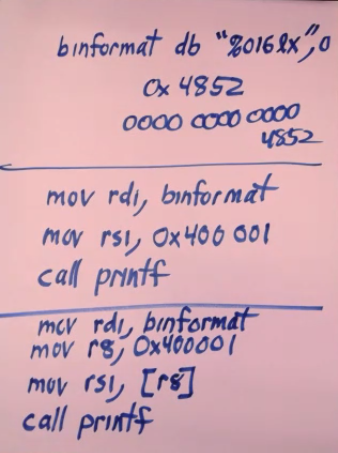


Stack grows when qword pushed, when recursive calls

Heap grows if space is dynamically allocated during run time

Double \* test = double[20]

<= 0x400 000 is OS



## Week 3 Terminology and functionality of registers:

day1:

Von Neumann machine

Quantum architecture

Names sizes of data

Components of Intel/Amd cpus

Special registers (rbp, rsp, rdi, rip) rax

Preserved and volatile registers

Pg 177 preserved and volatile registers

MSB, LSB

Big, little endians

SSE(xmm registers)

Cache L1, L2, L3

* Where and what size is the cache p 15
* Command to show cache

Org of memory p17

Stack, gap, heap, .data .bss

2’s complement

Left most digit (msb) is the sign

0 111 = 7

0 110

0 101

0 100

0 011

0 010

0 001 = 1

0 000 = 0

**Negatives**

1 111 = -1

1 110

1 101

1 100

1 011

1 010

1 001 = -7

1 000 = -8

N = # bits, 4 here

Range -8 to 7

-(2^ (N - 1) ) to (2^(N - 1) -1)

Smallest int for N bits = -2^(N-1)

Largest int 2^(N-1) -1

Two’s complement - flip all the bits, then add 1 to get the opposite number

Word - 16 bits

0x6b29 4f8d 309a

To flip hex, same logic, flip according to the 0 1 … F, then add 1

0xABCD1234

MSB -> LSB

Little endian (intel) -

MSB <- LSB

34 12 CD AB

Natural order - left to right.

I think of computer numbers in right to left

8 bytes address every block

ASSIGNMENT 2:

Fgets to input the name

; put 2 into xmm153

Mov rbc, 0x4000 0000 0000 0000 (2 as a float)

Push rbc

Movxsd xmm15, [rsp]

Pop rax

Can’t move stuff between the components of cpu

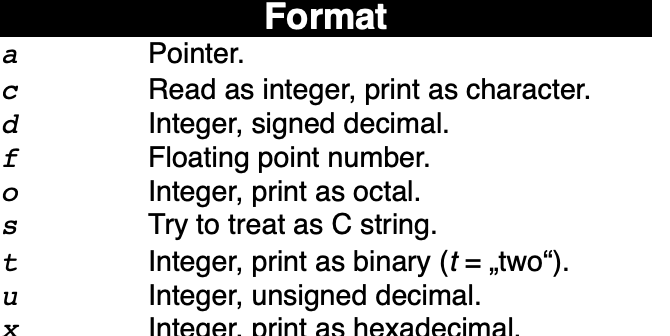
Week 4

GDB

p/format

In c++: double x = 12.4

p/format x

****

[**https://darkdust.net/files/GDB%20Cheat%20Sheet.pdf**](https://darkdust.net/files/GDB%20Cheat%20Sheet.pdf)

[**https://cs.brown.edu/courses/cs033/docs/guides/gdb.pdf**](https://cs.brown.edu/courses/cs033/docs/guides/gdb.pdf)

zCreating break points:

B file.txt:line#

B func\_name

B line#

R - run program

N

Ni

S

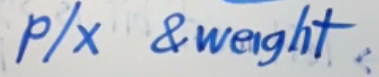
Si

c

N - Executes the current instruction and shows the next instruction

Ni - shows the next instruction to execute

S - steps into function call if there



Prints the address of weight in hex

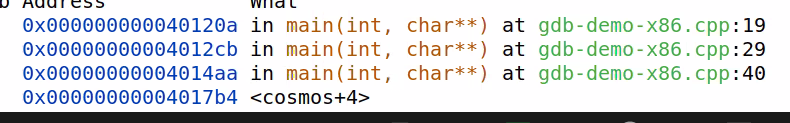
Int \* w = new int(187)

p/x w = address on the heap

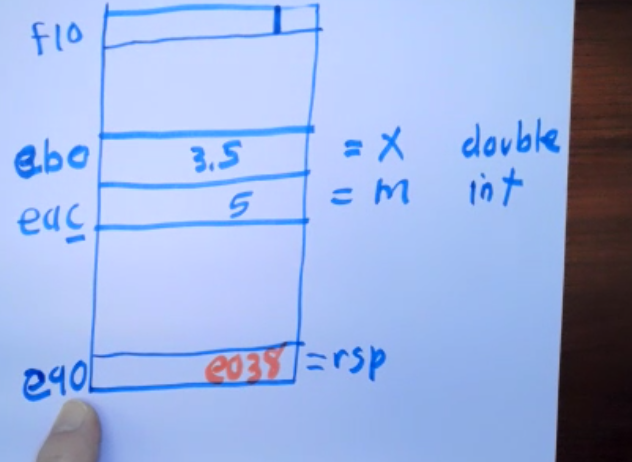
p/x \*w = the stored value at w address

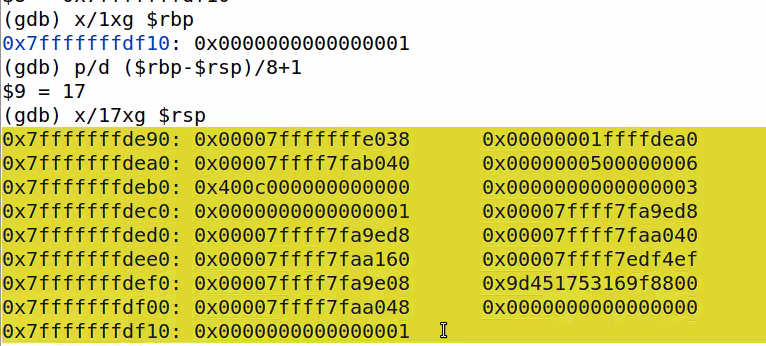
p/d &w = the address of the pointer variable (on the stack, not the heap)

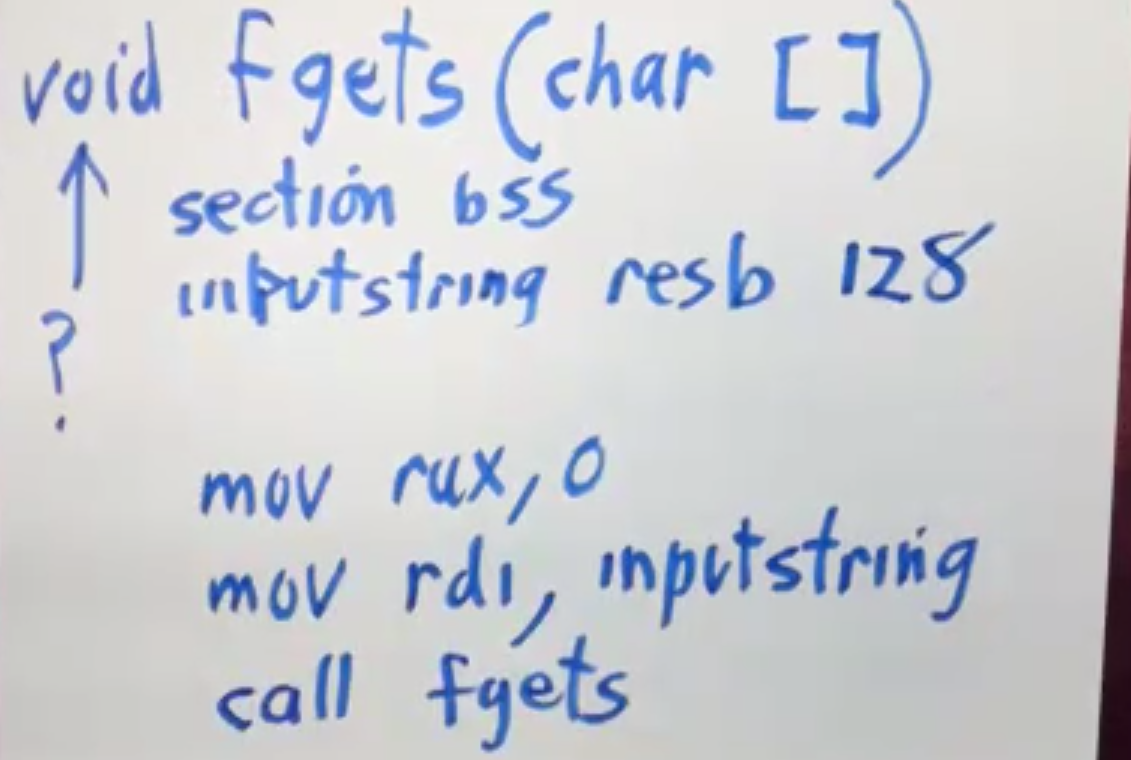
0x400000 - OS, Binaries are right above that



7fffff (near the top of memory\



)

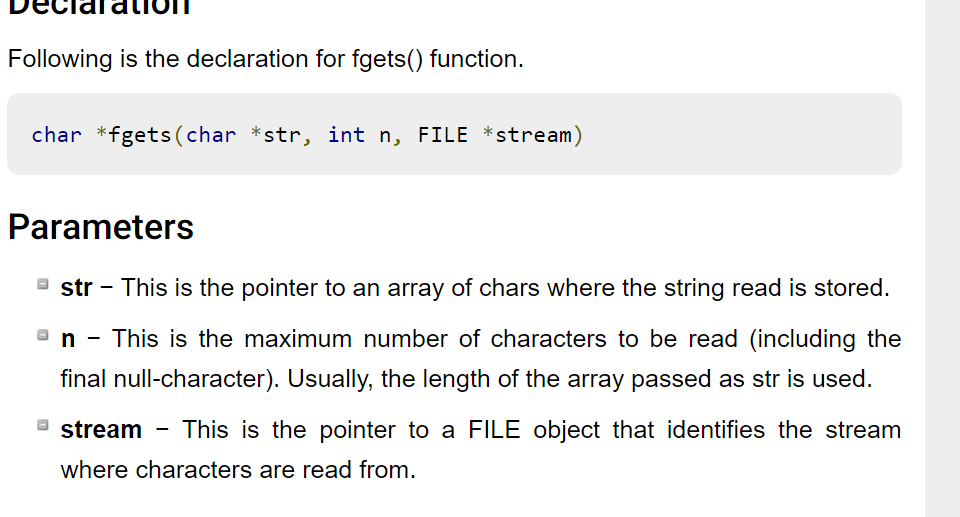


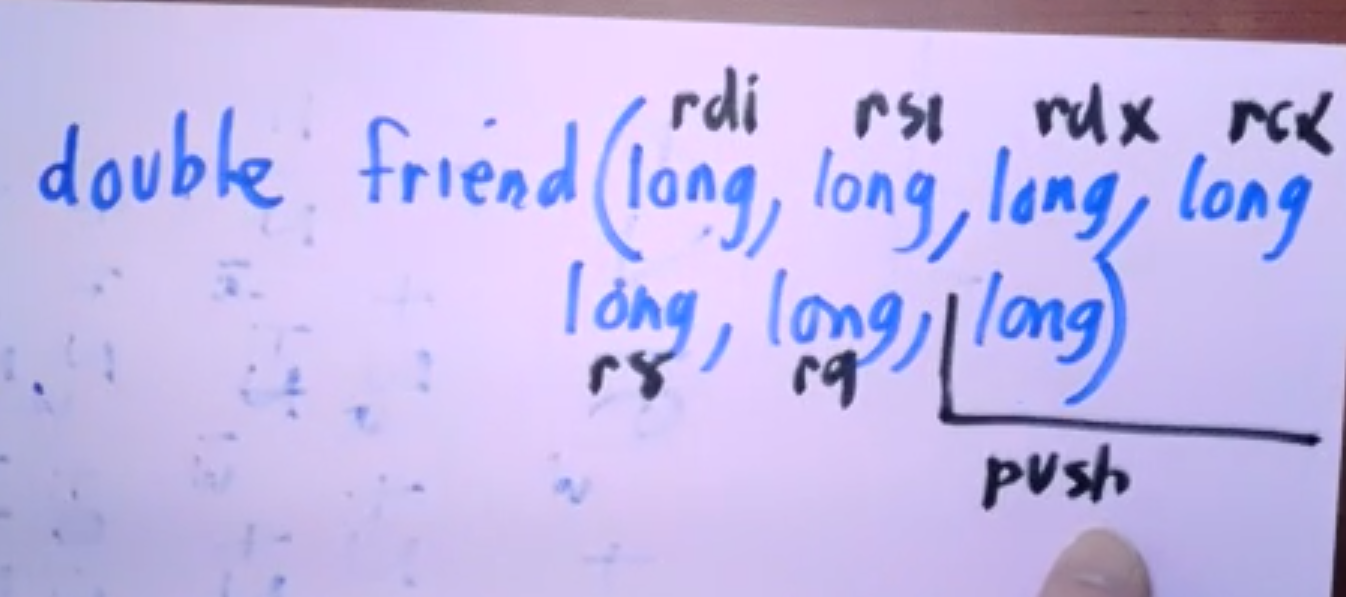
p/s (char\*)&string\_name - print out a string in .data of asm file

Everything is treated as memory in assembly, so you need to cast a char\* to it to print it as a string

To remove the /n symbol,

Use strlen(char[]) -1 to get the index and move a 0 to that spot to replace

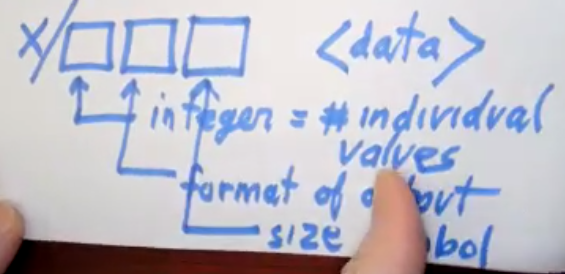




Activation records + stack dumps

Showing memory at a certain location

Identify variables in the stack



Boundaries

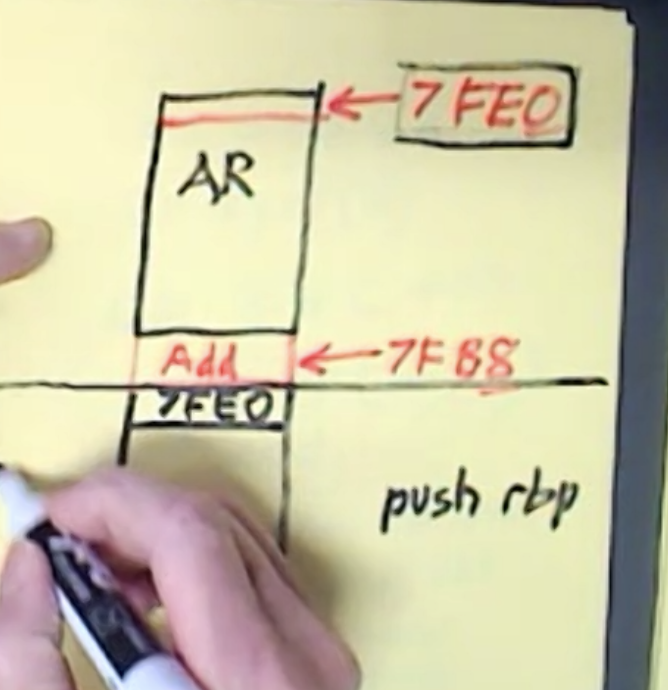
The backend will always start on a 16-byte boundary. (ends with 0 on the end of the hex address, aka the “16th” number of 16)

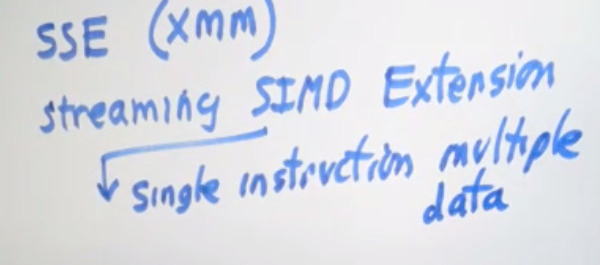
When a new function is called, a new qword is added onto the stack in the previous AR. Other words, 8 bytes are added to the front end of the previous AR

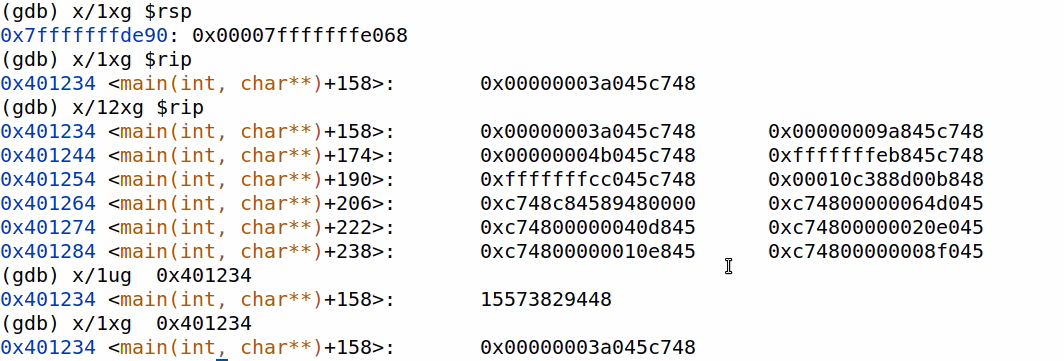
These 8 bytes carry the address of the next instruction that will execute after the new AR finishes execution **(push rip) RIP - NEXT INSTRUCTION**

This adds 8 bytes onto the stack, which misaligned it, however we always **push rbp as the first instruction, making sure that the new AR is on the boundary.**

The very first instruction of a new Activation Record is **push rbp. Rbp contains the back address of current AR.**

****

****

****

**Gdb ./out inputfile**

## Week 5 - GDB with arrays and pointers

Where are pointers and arrays

Num qwords in AR - p/x ($rbp - $rsp)/8 + 1

Static vs dynamic

At compile time, knows how much storage is needed for these objects ( static )

Doesn’t know - dynamic heap

ex)

Double \* p = new double[15]

The p variable is on the stack, but p’s value is in the heap

Double a[7];

High addresses - stack space

Low addresses - heap space

-1 in hex 0xffff….e (integer in 2’s complement, NOT floating point)

Self note (johnson):

Go over rip, recognizing when something is on stack and when something is on heap

400 187

Rsp, rax, rip, rbx, rdi, rsi, rdx

Day2:

Un signed ints =/= twos complement integers ( they have signs)

IEEE Floating point numbers

[Double (IEEE754 Double precision 64-bit) (binaryconvert.com)](https://www.binaryconvert.com/result_double.html?decimal=051052046050)

Scientific notation

1.0111 x 2^a

Numbers after the decimal point - significand

1st bit sign

Next 11 bits - exponent

52 bits - number (mantissa)

conversion

14.75

1. Convert 14 to binary (divide by 2 until you hit zero)
2. Convert .75 to binary (times 2 until you hit 1)

Bias number - 1023 - 64 bits

127 - 32 bits

1000 0000 00 - 1024

5

1023 (bias) + 6(tru exponent) = 1029 = 1000 0001 01

1029 - 1024 = 5

1023 - 0111 1111 11 + 1 = 1024 = 1000 0000 00

1. Shift the decimal point so you have 1.aaaaaaaa
2. Times that resulting number by 2^(how many decimal points moved)
3. Bias number + decimal points moved
   1. Convert to binary
   2. Those are the 11 bits
4. Get the mantissa and plug that into the 52 bits.
5. Combine the sign, exponent, and mantissa together and convert into hex.
6. How would you convert in the other direction?

The repeating case

83.1 = 1010011.00011001100110011 0011…

1.01001100011 0011… x 2^6(tru exp)

1023 + 6 = 1029

1029 -> 100 0000 0000 = 1024, 1029 = 100 0000 0101

Internally,

0(sign) 100 0000 0101 (stored exponent) 01001100011 0011… (mantissa, repeating until 52 bits)

If the 53rd bit is 1, then round the 52 bit up.

Bias #

N = #bits

2^n-1

Bias - 10 1’s

[How do I check if a C++ string is an int? - Stack Overflow](https://stackoverflow.com/questions/2844817/how-do-i-check-if-a-c-string-is-an-int)

Week 6

King of Rock

Elvis

King of Jazz

Benny Goodman

King of Hollywood

Douglass Fairbanks

King of Hearts

Rudolf Valentino

Queen of Rock

Janis Joplin - D

Tina Turner - alive

Grace Slick - alive

King of Linux

Linus Torvalds L

King of FOSS

Richard Stallman L

Chris sawyer - king of assembly - Roller Coaster Tycoon 2

King of assembly education - Chris Creel

Applications - Word

Library functions - sqrt

Licenses

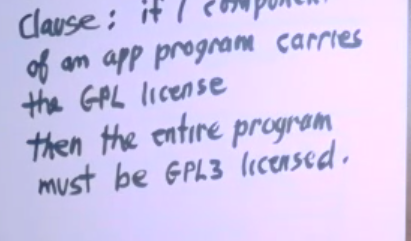
Proprietary - for profit, insure developer’s right

OPEn source license

* Insure user’s right
* Source + executable together
* Install and run multiple copies
* Modify source
* Cannot remove previous author’s info
* Avoid going to court
* Open Source license

GPL3 - application programs

If 1 component



LGPL3

* Don’t need to carry the license over

Intellectual property

Creative commons

Non comercial

Distributing similar work can only be under the same license

Concept test

Programming test=

Blank - 20%

Charles Babbage

1820 - 40 worked on

Difference machine fails

Analytical machine

Invented 10’s complement

0

1

2

3

4

5

6

7

8

9

6831 259

3168 740

* 1

3168 741

Subtracting in regular arith

sub

8 423 791

6 831 259

1 592 532

10’s complement subtracting

Add them

8 423 791

3 168 741

1 1592532

The left most digit is lost in the computer,

Ada lovelace - first computer prototype

First programmer, function calling

1991 IBM recreates the computer

Bias number in hex - 3ff

Converting decimal to hex and hex to dec

Geometric sym

N

Summation r^k

K = 0

1 - R^(n+1)/

1 - R

Find the “pattern” in the binary (R)

Determine how many times the pattern occurs (how many bits left/how many bits in R)

Compute the summation.

1 + (0.5)^1 + (0.5)^3 + (0.5)^5 + … + (0.5)^51

Must follow R^0 + R^1 + R^3 + …

1 + 0.5(0.5^0 + 0.5^2 + 0.5^4 + … + 0.5^50)

1 + 0.5( 0.25^0 + 0.25^1 + 0.25^2 + … + 0.25^25 ) = answer

Subnormal, normal, nans

A number is subnormal if it’s stored exponent is all 0’s

A number is a Nan if its stored exponent is all 1’s

0.0 - 0x 0000 0000 0000 0000

Nans

0 x 7FF0’s - positive infinity

Decimal value of positive infinity?

7ff -3ff = 400

2^10 (tru exp)

2^1024 - positive infinity on 64 bit computers

1.0 - 3FF0 0000 0000 0000

4000 0000 0000 0000 - 2.0

-1.0 = 0x BFF0…….00000

Grace Hopper phd from yale - death 1985

Joined the navy - first digital computer

In university of Iowa - 1939 - 1940

Project first computer

Harvard Team

Aiken’s Mark 1

Programmer 1940 - 1950

Program crashed cuz there was a moth in the computer “bug”

Debugging

Rear admiral

Bug added to Journal in smithsonian museum

Speaker at conferences

Wires 11.8 inches

Speed of light = 11.8 inches/nanosecond



Sudo apt-get --upgradeable

Chris creel - assembly education

Unsigned int

2’s comp

Tens comp babbage

16’s comp

IEEE754

Number lines (sub normal…normal that stuff)

Bias num = 1023

Base num = -1022

Smallest true exponent of normal numbers

0 < Stored exp < 111 1111 1111 = 2047

Sub normals @ 0

2047 reserved for infinity and nans

Inbetween normals

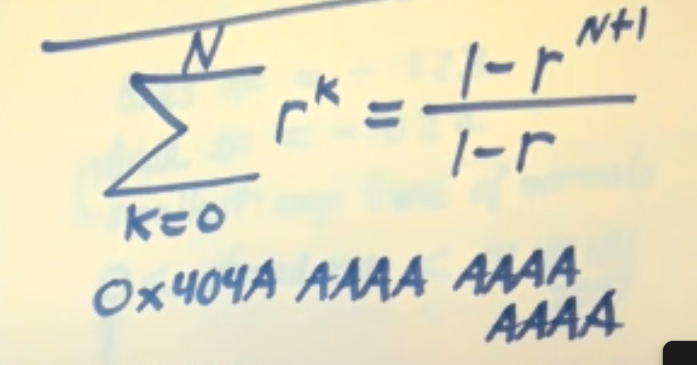
The smallest normal number?

1 is the stored exponent in IEEE form

000 0000 0001

1 - 1023 = -1022

1 x 2^-1022

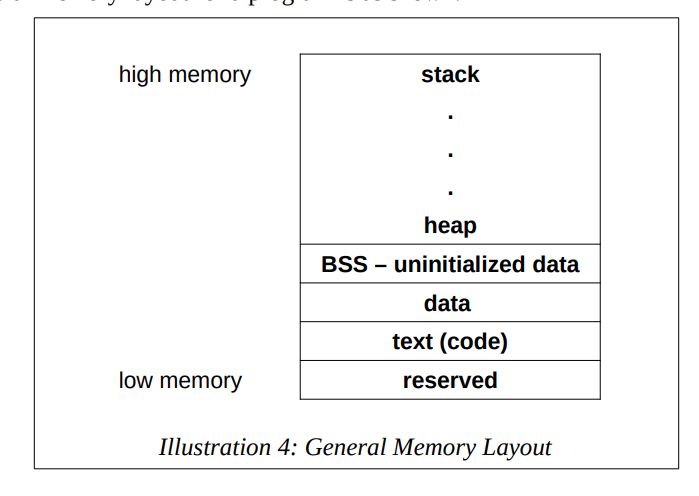


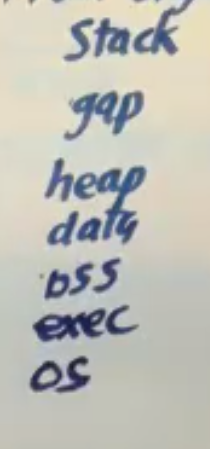
LSB

MSB

Big Endian, Little Endian

Memory organization





Famous computers

Difference Machines

Analytical Machines

ABC first electronic computer

Mark 1 (Harvard) - first bug by grace hopper

GDB

<https://darkdust.net/files/GDB%20Cheat%20Sheet.pdf>

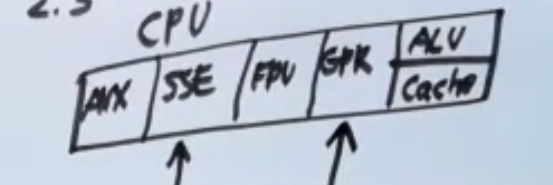
<https://cs.brown.edu/courses/cs033/docs/guides/gdb.pdf>

Variables registers

Segments of memory

Components of array

Printing addresses of values



Dividing integers

Get the remainder

Mov rax, r13

Cqo - expands rax to rdx

Idiv - signed integers division

Idiv r14 (numerator is rdx, denom is operand in idiv)

Quotient goes into rax

Remainder goes into rdx

Full ex:

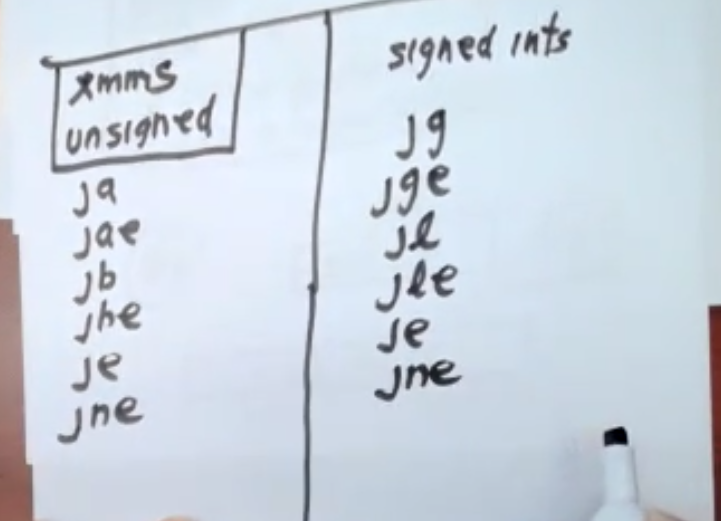
mov rax, r11

Cqo ;(expands rax to rdx)

Idiv r9 ;(rax/r9)

; rax = quotient

; rdx = remainder



Cmp rbx, rcx

Ucomisd for xmm

Static vs dynamic

Rbp, rsp, rip

