ECE 354 | 1 | 25 | 22

Course Overview

problem
- C Programming
- Assembly x86 64
- System
  - virtual Memory
  - Dynamic Memory Allocation
  - caching Interrupt + OS
  - communications and Interrupts

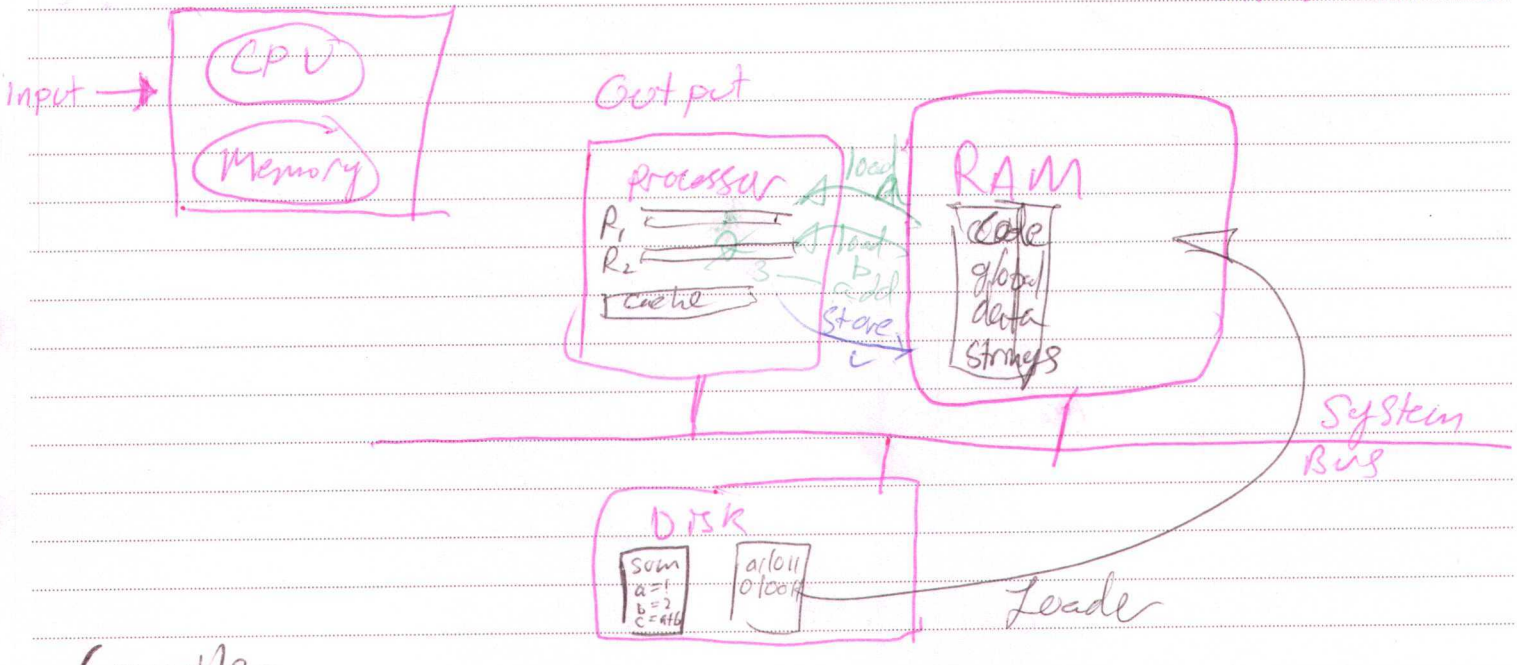
How computer Works

Application

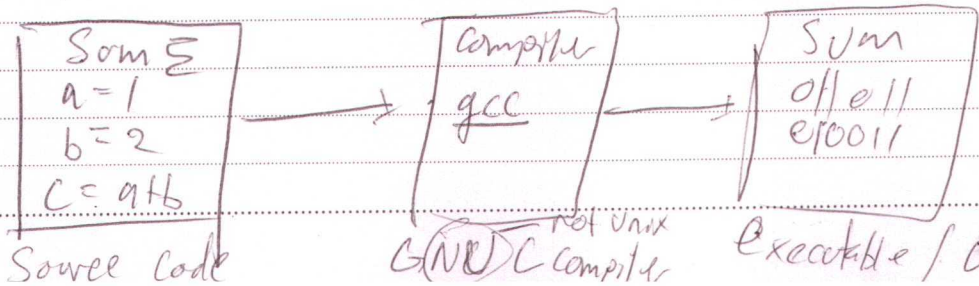
Software Systems - OS
- Drivers

Hardware - CPU
- RAM
- DISCS

What happens when we run Prog. on Von Neuman Architechture



Compiler



Build process

pre processor : Copy & paste included #def

sum.i

↓ Sum.i

Compile process : Translate → Assembly

↓ Sum.s

assembly process : generate prealtotable object file

↓ Sum.o

Linker : merge all obj file → executable

1/27/2022

Build process Review

CPP - (C pre processor)

#include #declare

↓ Sum.i src file

CC1 (compiler proper)

translation assembly

↓ Sum.s

Assembler AS

relocatable file in memory space

↓ Sum.o

LD == (Linker) "Loader" in older days move library from Ram

↓

SUM Sum.exe

gcc -g -o sum demo.c -Wall

debugger, output, source, all warnings

Stdio.h

use different names

.LFB0 .file

table assembly directive

endbr64 end of branch

%edx register

movq start

subq start

leave ~~collect / save / load~~ delete stuff

rem op code ★ 642 Comp Sec

Objdump -d
-S
~~-xd~~
~~-xd~~ -b
↳ hexadecimal dump

→ Objdump -d sum.c
uses hex.

2/1/2022

fields

ID	Title	Year	Director	Rating	Revenue

D Open White Space { for word space }
★ change it
A

format string "%f"

gcc ✓

main always returns int
so use it to verify success or fail

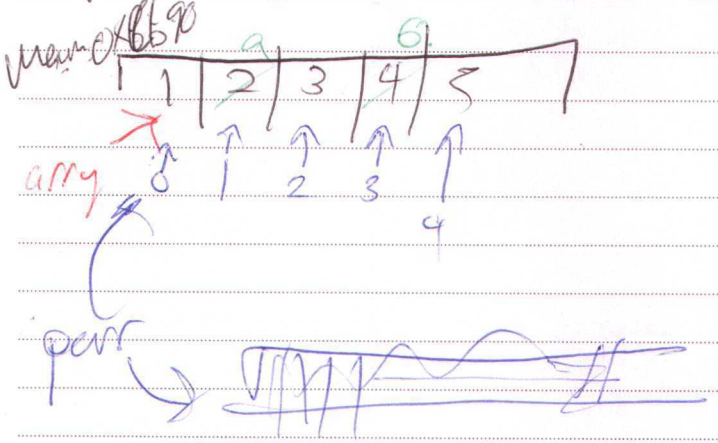
x==3 ⟺ ((x=3))

|| ∨ or

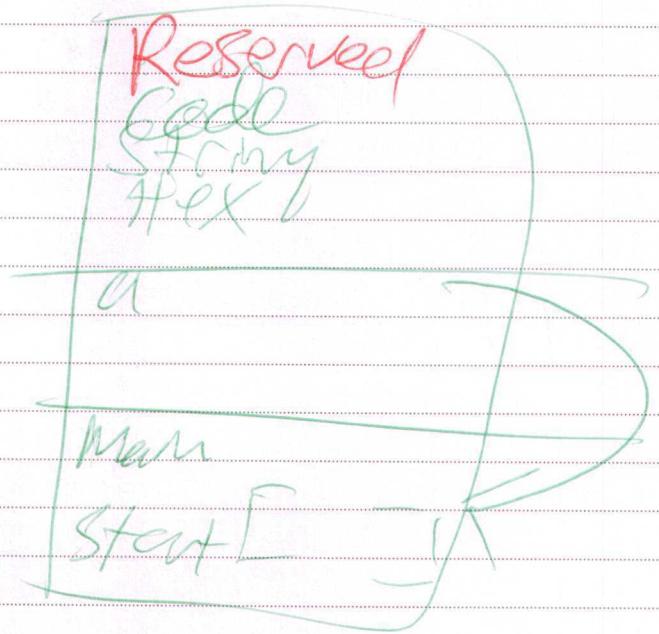
&& ∧ and

It's like modular addition

int *arr = 0x6540



Pointers are 8 bytes



Objects arr[3]

int [5] [1][2]

1*(sizeof int[5]) + 2*sizeof(int

m[rows][columns]

m[1][0] = 21/4

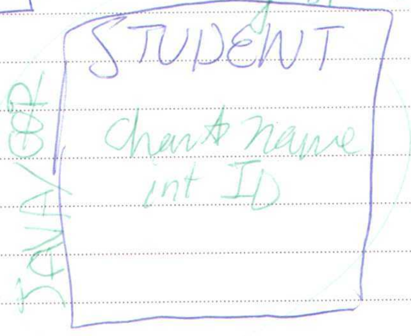
(base address + col * sizeof(int)) + j * size(int

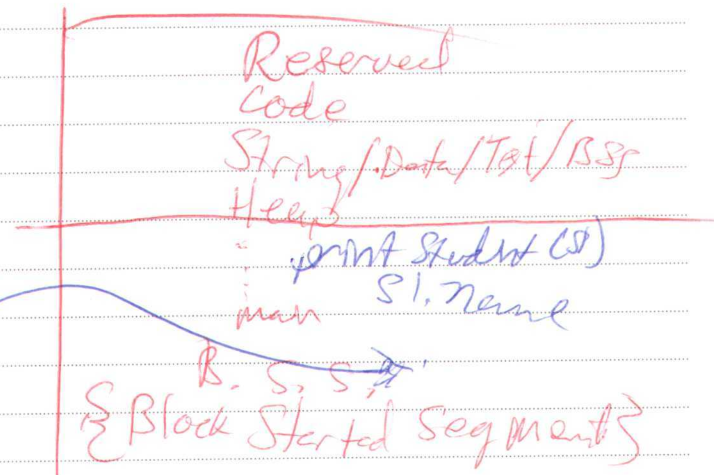
m s

*(mat + i) // pointer deference dealing at one row

+ *(j)

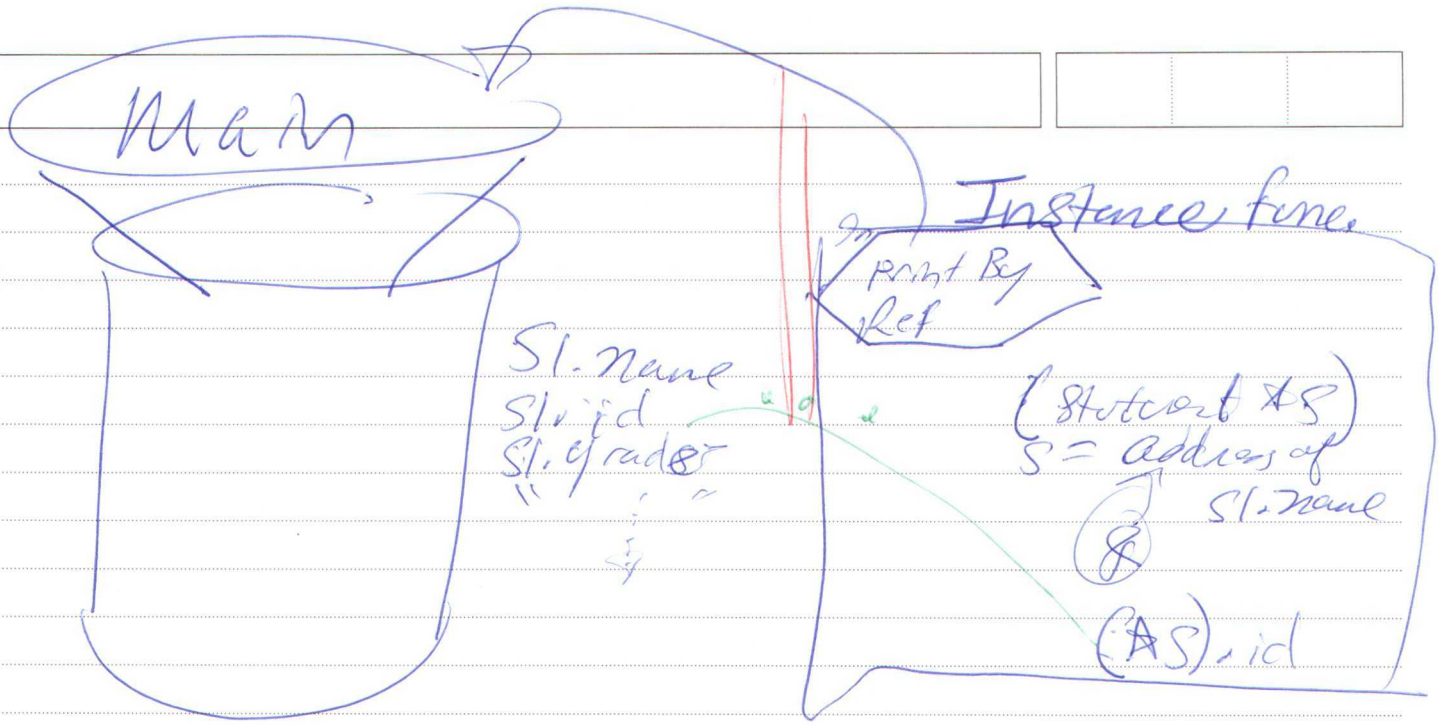
Struct — just a container for data

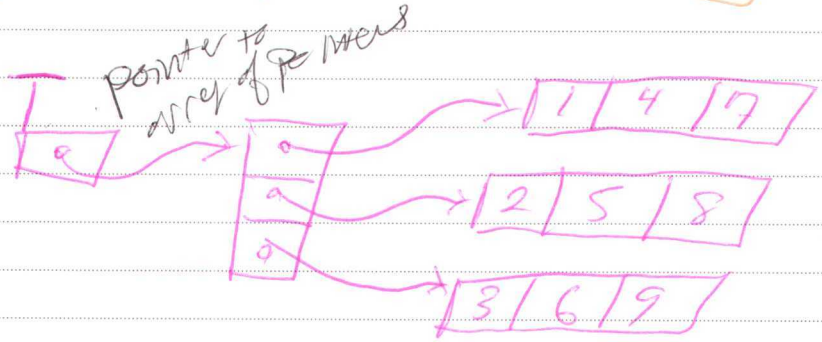




just like arrays S1

*(j) is the same as [j]



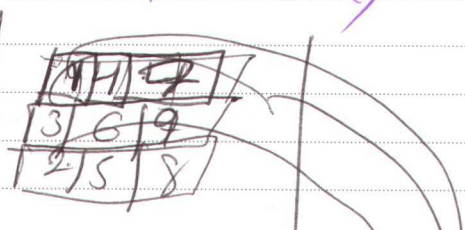


T[i][2] = 8

int x;
int *p;
p = &x;

int **T
array of pointers

Heap

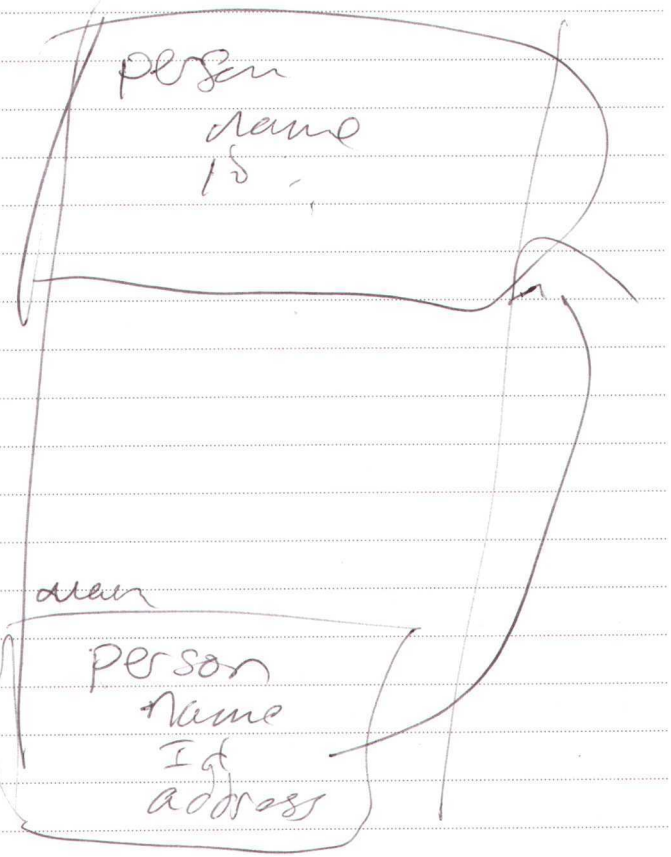


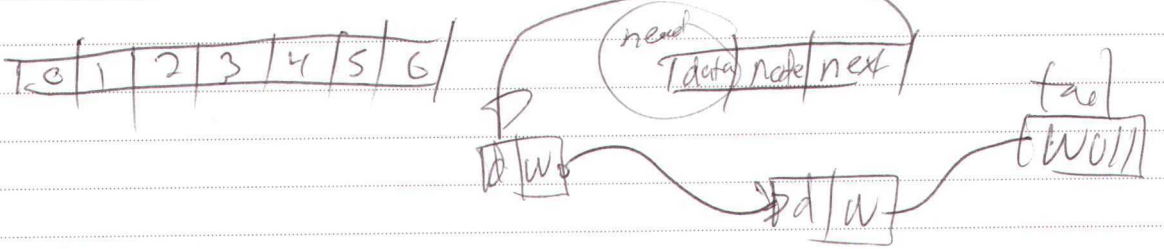
Stack

main
row(int **T)

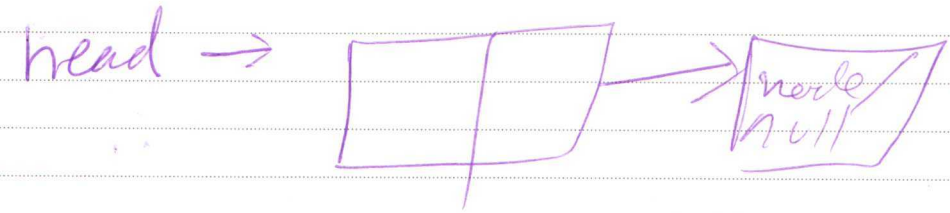
malloc = {"go get me bytes"}

always free malloc





3 · 8 · 22



Debugging Techniques

Traverse - find a node to delete or add
Prospect - return

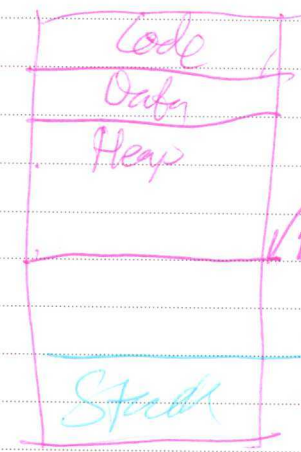
traverse to find previous next
next = current
reassign and
free

CS APP - 9.9 3/8/2022

malloc is the built in library that comes with C
Dynamic memory Allocator

address space

Because computers in the 70's needed to micromanage memory locations



BRK pointer (1970's)
Bottom of Heap
TOP of Heap

malloc

free store - first available block

int brk(void *addr)
- Let OS reset the memory offset location

Today we have sbrk

void *sbrk(int incr) size
input & return brk pointer

Implicit

- programming language with built in garbage collection system
- i.e. Java / Python

Explicit

malloc / free

heap management responsibility of programmer

C / C++

new / delete

↳ deconstruction

Super Simple memory Allocator

alignment Requirement (8)

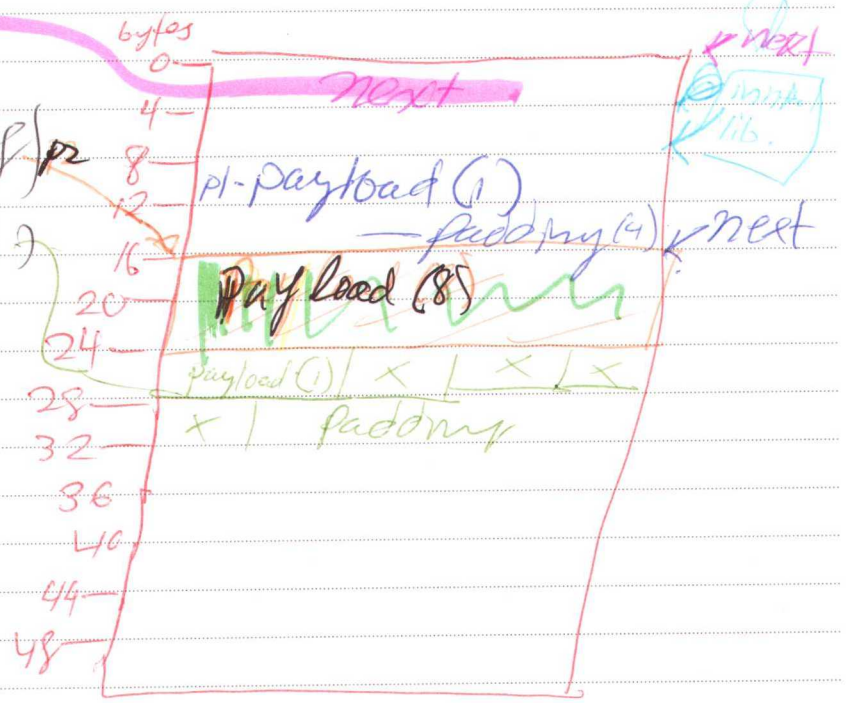
data starts at address ÷ by size

```
int * p1 = malloc(4)
int * p2 = malloc(8)
int * p3 = malloc(1)

free(p2){
    return;
}
```

Run out of space

create holes in memory we cannot reuse



3 / 10 / 2022

Malloc Review

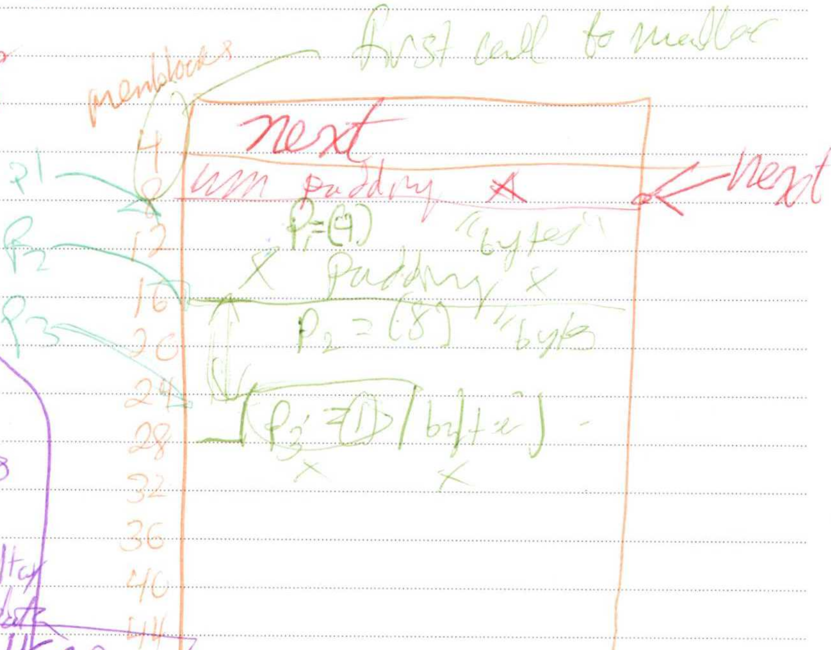
void * malloc (size * - non-bytes)

1) If this the first call
2) find a free block
   return to user
3) Return pointer or null

void free (void * p)
- check to see if p is generated by malloc
- change block from allocated to free
- coalesce adjacent free blocks

Alignment Requirement = 8

p1 = malloc (4)
p2 = malloc (8)
p3 = malloc (1)



General Allocator
1.) Keep track of blocksizes
2.) Arbitrary Sequences
3.) Requests are immediately dealt
4.) only vs. heap for metadata
5.) Block alignment
- Starts on address divisible by 2^n
6.) No modification

- HEADER
- Payload
- Padding

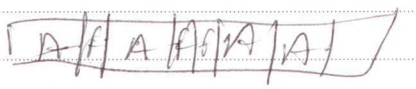
Goals Maximize Throughput - Requests per second

- minimize wasted space
  - Padding
  - holes in heap mem
  - Meta data

Fragmentation -

Internal
- Block is larger than requested Payload
- Alignment requirement

External
- have free blocks
- not large enough to Request



slivers of non-usable space

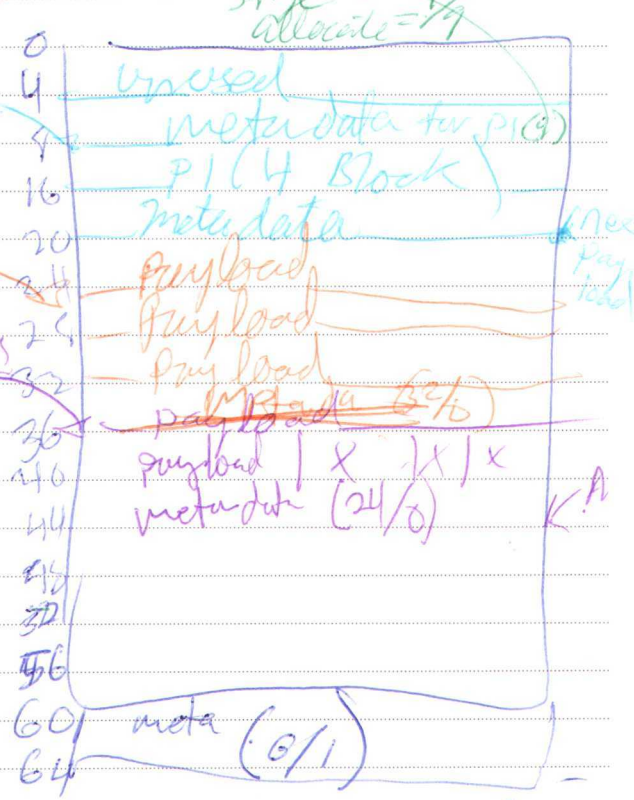
IMPLICIT FREE LIST

Size allocated
Header 4
Payload 4
Padding 8

int P = malloc(4)

int *P2 = malloc(8)

char *P3 = malloc(1)



Header stores metadata

- Heap
- Payload
- Padding

Padding Size are Free Bit

Size 8 Free Bit 1 = 9

Alignment Requirement 2

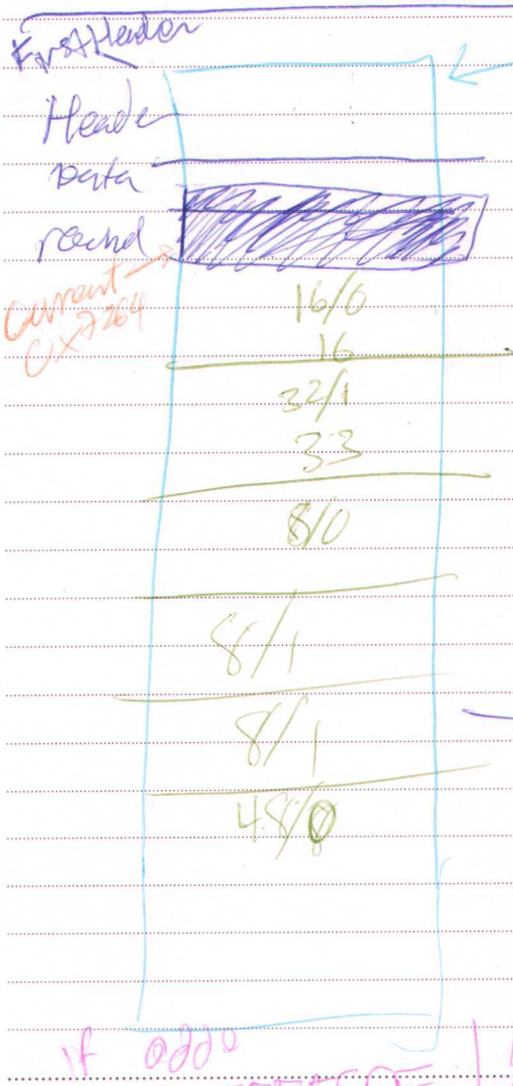
2	0010
4	0100
6	0110
8	1000
10	1010

Use this bit to store allocated (1) / free (0)

always zero if size must be a multiple of 2

Rules

1. Alignment requirement - based on largest possible primative (8 bytes)
2. minimum block size (1)
   Header size + data size + Footer size + round up (mod %)
   (0)
   Header - size / allocated
   previous allocated
   padding
   End of list marker
3. placement ~~padding~~ policy
   First fit
4. Coalesing policy



First Fit

minimum Block size

Head	Data	Footer	+ (Round)
4	1	0	3

MBS = 8

while S % ≠ 0
S++

Malloc (18)

Block ~~Size~~ = 18 + 4 + 2 (D H R)

Block Size = 24

Traverse List

~~Free Block Size~~ Current > Block Size

Current → Size Alloc & 1

if odd & FFFFFFFe | 1001 1110 1000

Bitwise (8) 9 → 1001 — 1000 1000

Continued

Split

if remainder block size is greater than minimum block size

A ⟶ A Split

; set alloc | 1 // set mem to alloc

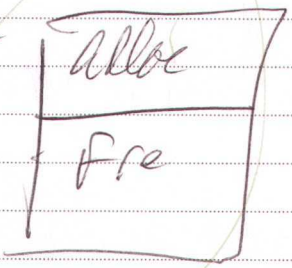
pre

Split ; add a new header where current + BS

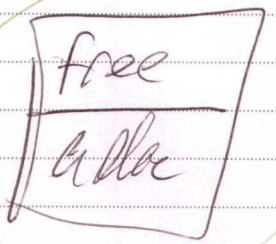
store FBS - BS → even

return ?

Split



∨



← Faster for finding free

- better for realloc
- easier to free
- easier to coalesce

Free (p) 0x7 2A0

① verify p is from malloc optional

② find pointer to header
p - 4 = 72A0
Traverse find that address

③ set alloc bit to zero (Subtract 1) (MAX Int - 1)

0xFFFFFFFE

Coalescing

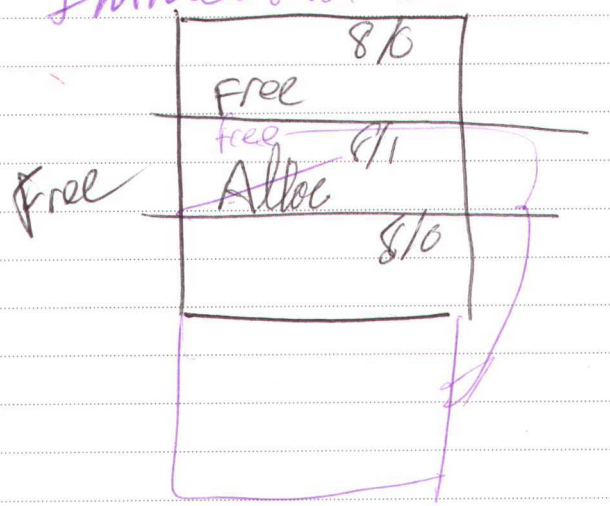
Immediate

Deferred

Immediate

Block After

Block Before



Block After

Find Next Block

current + Size is free

Change Size to Size of Block Size

8 + S

Deferred Coalescing

merge all pairs of adjacent free Blocks

- Count frees
- allocation fails

~~Block~~

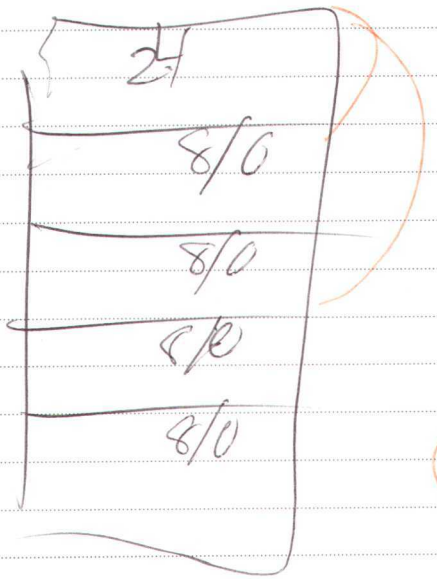
Traverse current

next

if both free then merge

current = current + Size of next

Size free



Header of Size 0

Fitting Policy Performance Analysis

First Fit

1. Retain large block at the end of list
2. Fragment beginning of list
   - "Splitting"
   - "Slivers"
3. Increase search time

Next Fit

next pointer - Start search at the last allocation
1. Faster than first fit
2. more even distribution of block size
3. Great initial performance

Best Fit

1. scan entire list
2. Smallest free block that's big enough

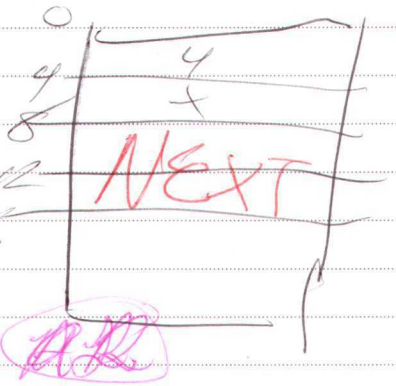
SLOW

Best memory utilization
Smallest slivers

GCD = MBS

Memory allocator for

Node { int data
Node *next }



Alignment Requirement = AR

Data Size = 16

First Fit

Immediate Boundry - Header / add footer

Min Block Size (MBS)

Header + Data + Footer + AR padding

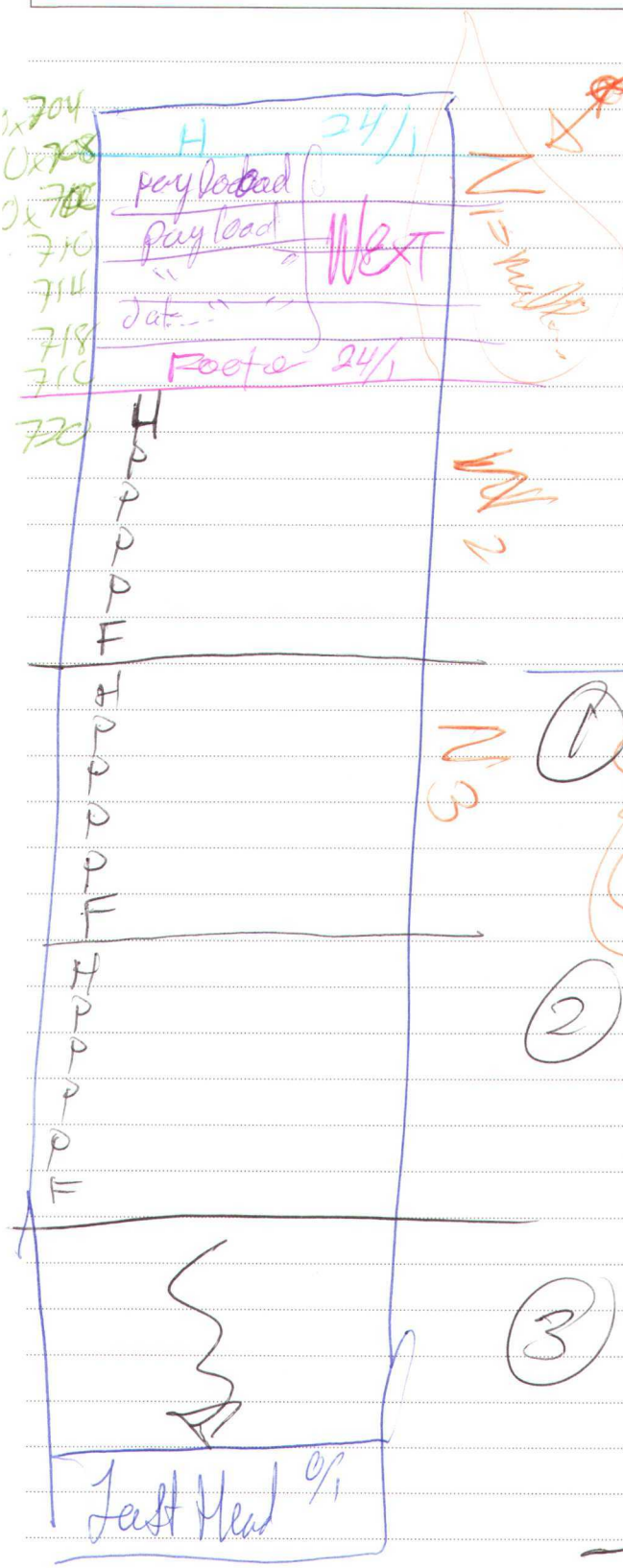
padding Size allocated

21 16 4 6

allowed Block Size

MBS + n° AR

↳ 0, 1, ...



W_1 = malloc (sizeof (node));

Node * W_2 = malloc (sizeof (node))

Node * W_3

free W_2

free W_3

free W_1

① User pointer - 4 = Header pointer

0x71C

set alloc bit to 0 in header/footer

set alloc left to coalesce

② Block alloc

same as before

add size to get to next header

→ if free → then:

add size to W_2 from Header size

③ Check previous Block

&H_2 - 4 = &F_1

check alloc & size of footer

→ if free → then →

get size from footer

size - size from = size H_1

FINISH

add size of H_2 to H_1

Improved Boundary

① Size are divisible by 8

1111 00C

↳ allocated

previous allocated

If we previous block is allocated,

then don't want to free it

So no need a footer

Minimum Block size.

Struct Node { next *8, data 4 } (12 byte)

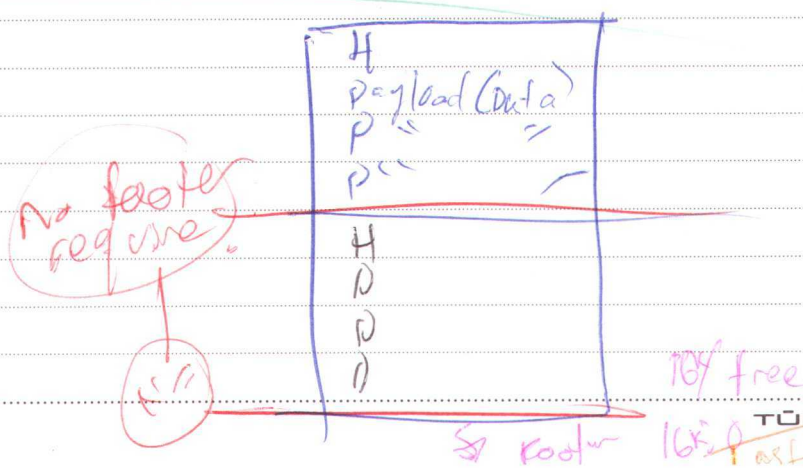
H + D + Padding =

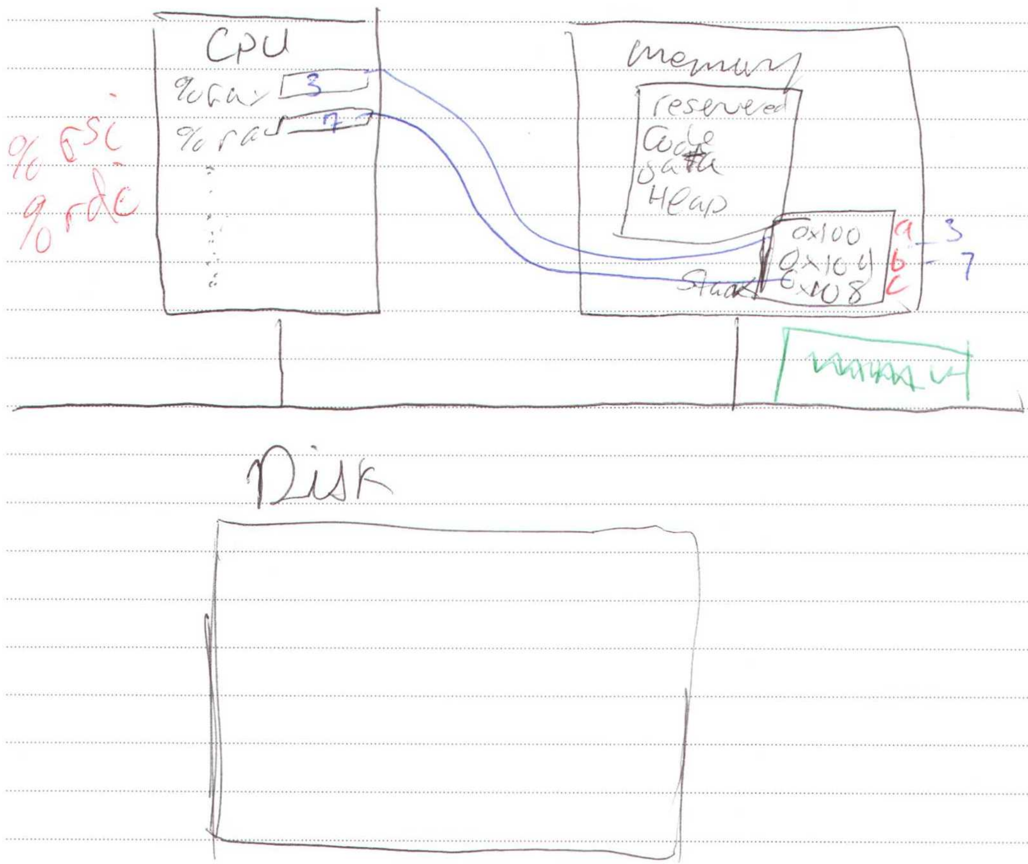
4 + 8 = (16)

H + F + Pad = (8) 8 | 16 ✓

16 ✗

MBS = (16)





Compilation — Preprocessor — sum.i
compiler proper — sum.s
assemble — relocate object file
linker — executable

Objdump -d

2 gcc - S -O

~~why should i ...~~

(3) C = (a<b) * C + (a ,) ∞ C

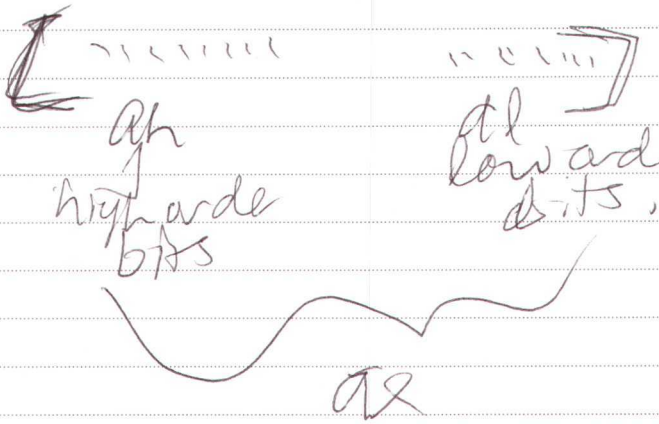
1. Reverse Engineering
2. Optimization
3. Efficient Code
4. Security
5. Parallel Processing

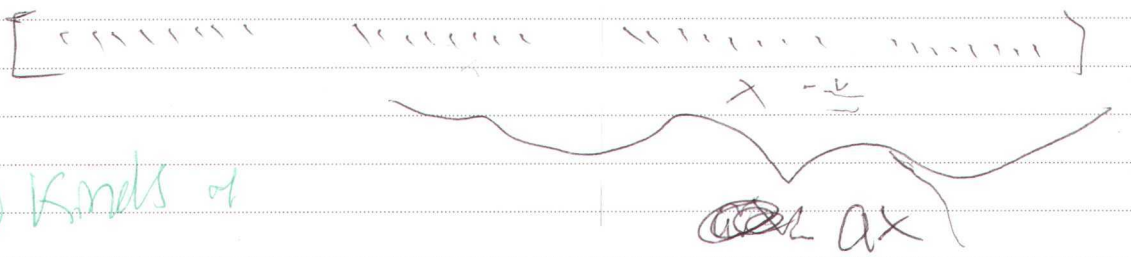
Thrust Notes

History | 4 | 5 | 22

Intel 8080 | 8086 80186, 80286, 80386, pentium

16 bit





(3) Kinds of

Data $100

char short int float double	Then b. $97, 2.9.

Immediate

Source	Dest
Imm	Reg
~~Imm~~	mem
Reg	Reg

Reg ← Imm

mem

Const removes stack pointer from stack for ref

Addressing Memory

General form: Imm (%base register, %displacement register, scale)

(index)

Immediate + base + displacement * scale

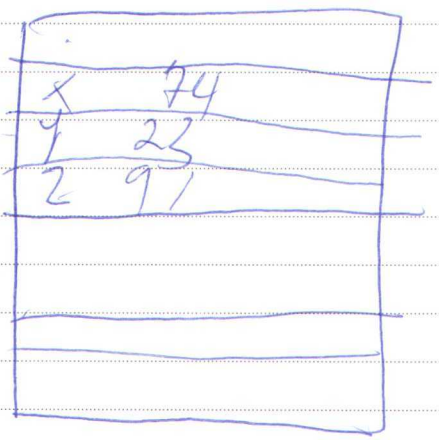
Indirect: (%base register)

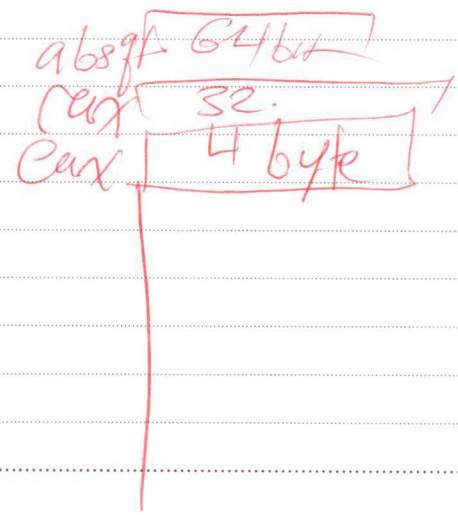
pointer variable

```
int x = 74
int y = 23
int z = 91
```

mov (%rax), %ebx

#B = *p





TO DO

Base + Displacement

1000 x = 79
1004 y = 32
1008 (A)
⋮
1BF0 PBP

① Imm(%Base Register) immediate + Base
4(%rsp) 4 + %rsp
4 + 0x1000 = 0x1004
y et sy

int @ = y;
mov 4(%rsp), %edx
mov %edx, @(%rsp)

Indexed DO

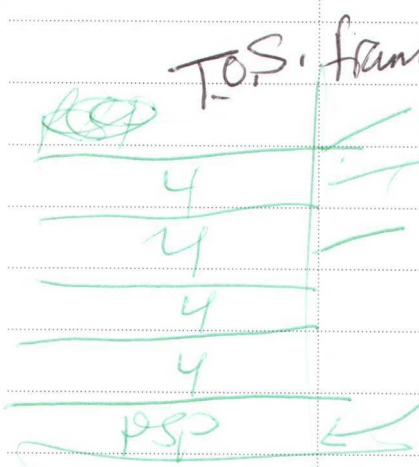
(% base register, % displacement register)

Imm (% base, % disp)

Imm + base + disp → what address in memory to get

Struct point { int x; int y; }

Struct point A
px = 5



%rax ← 0,
movq $0, %rax
-16(%rbp, %rax, 4) # x
movl $5, -16(%rbp, %rax, 4)

.py = a
movq $4, %rax
movl $9, -16(%rbp, %rax, 4)
movl 9, -12(%rbp)

-16 + 4 = -12

Immediate is hard coded # ← use registers

TOL

Scaled Index almost always used for arrays

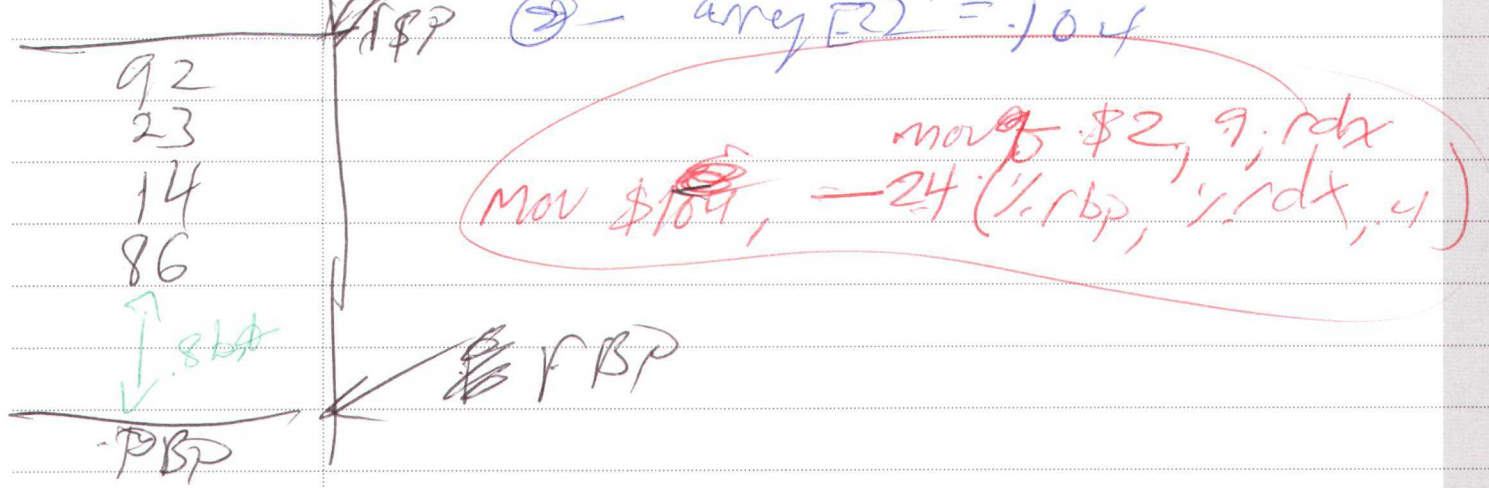
Displ(%base, %Index, Scale)

I + B + I*S

1
2
4
8

① int arr[] = {92, 23, 14, 86}

② arr[2] = 104



push & pop

addq $8, %rsp → move stack pointer up

movq data, (%rsp) → move data to the top of stack

Casting

char y = 92

int x = (int) y ⟹ 92

mov %bl | %bw | %wl

mov 5 | %bl | %wg

64

TOL

TO DO

Easy

Inc ++.
Dec --
Neg *(-1)
Not · bitwise complement
ADD
SUB D=D-S D-=S

SAL : arithmatic shift left
SHL logical shift
SAR
SHR

TO DO

		address	Val	Reg	Val
%rax	0x100	0x100	0xFF	%rax	0x100
	0xAB	0x104	0xAB	%rcx	0x1
	0x108	0x108	0x13	%rdx	0x3
	0xFF	0x10C	0x11		
	0xAB				
	0x11				
	0x13				
0xFC(%rax,4)	~~0xFF~~				
	0x11				

%rax → Imm(Base, , BP, Scale)

4/19/2020

~~Iterate~~

for

for(int i=0, __ , __){
sum+i
}

TO DO

Procedures

Control ~~continued~~ Flow
Conditionals
iteration
Functions · Calls

- These all set the instructions
- * RIP address of the next instruction

Caller vs. Callee

- returns
- args are

```
int main{
    sum(2,3)
}
```

Function Call Set Up

pg. 208

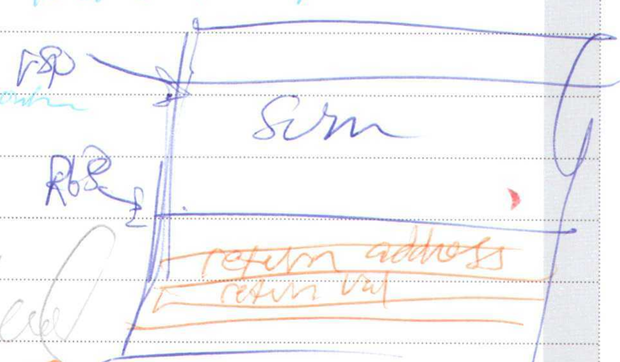
- reserve space for large return ie Struct
- Push arg. (creates more space)
- mov args
- Call → Push return address → instruction Pointer

Prologue

- ENDBR64 - Security
- push RBP & caller base pointer
- mov rsp to rbp
- Sub rad
- CANARY

Function Body

Stack Frame



Epilogue

- Stack clean up
- Keep return value in %RAX

Function Call Clean Up

- restore callee saved registers
- check canary

Return Value → %Rax + %eax

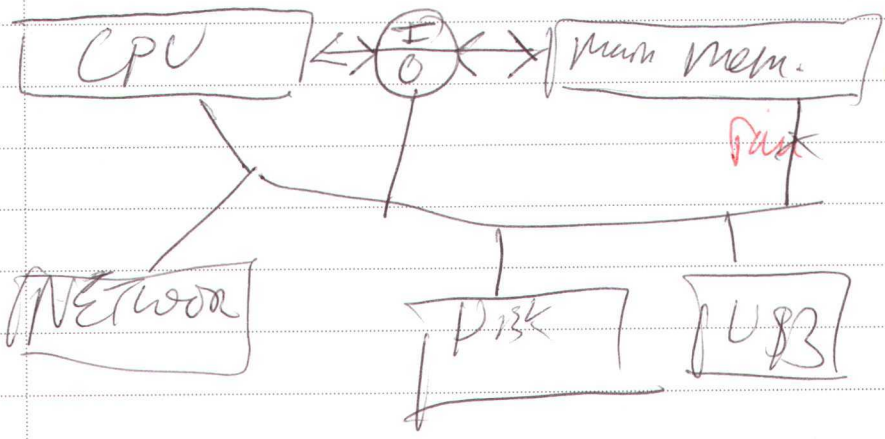
In

★ Parameters / arguments

TO DO

8byte @-40 ← 2
8byte @-32 ← 5
8byte @-24 ← 7
@-24 → RAX
7 → @-16
@-16 → RAX
@-24 RAX

4/26/2022



rip
rsp
rbp
rax
rbx
rcx

Locality - next data on mem is next to first data on mem

Temporal
use
data again

Spatial
use nearby
data

for(int i=0 ; i<n ; i++)
sum = arr[i];
}

TO DO

Register (o clock)

L1
L2
L3

Main mem
DSK
Network

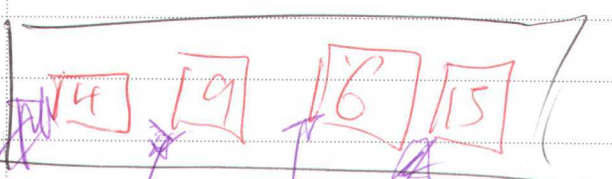
hardware reserve

4
40
50
200-300
10^7
10^9 clock cycle

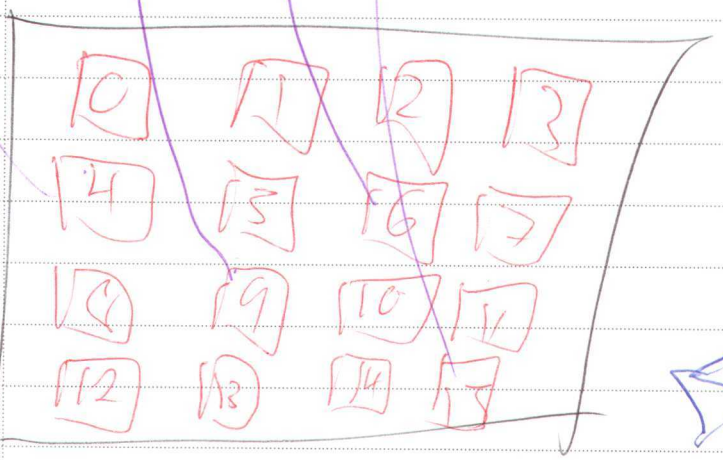
4/40 +OS flexible

Super flexible

level K

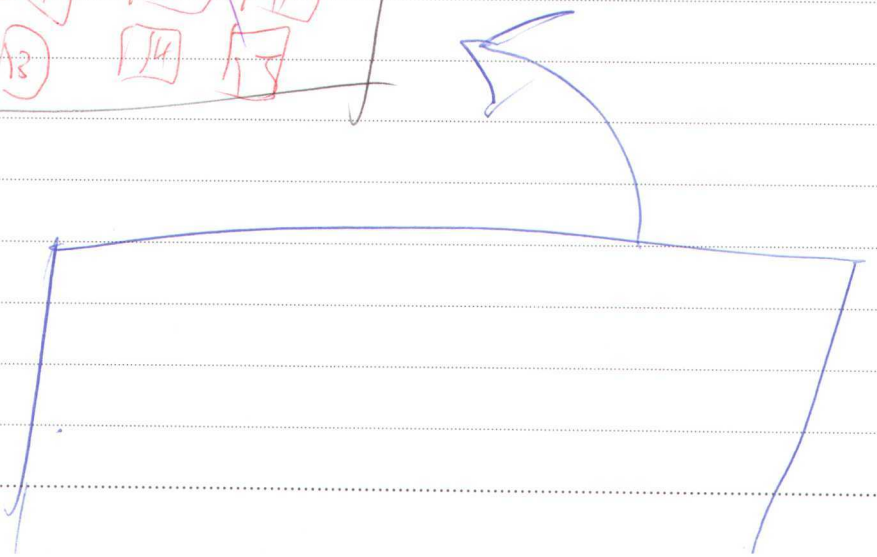


K f1



Restriction rule. Where it's stored @ each Block

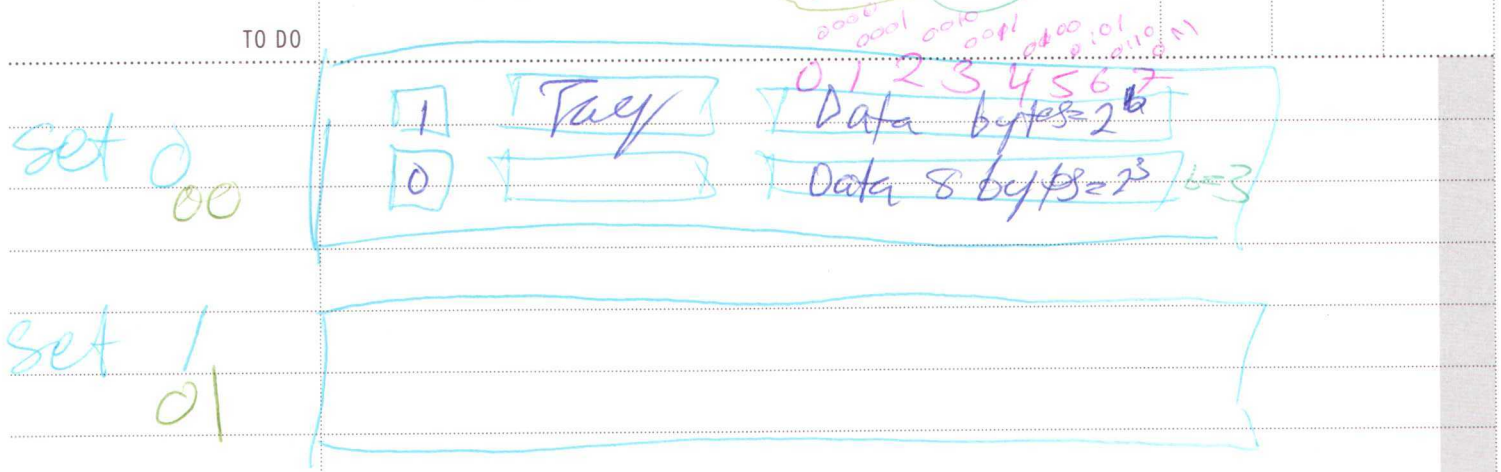
Placement Policy



TOL

0x~~E~~0C 1111 0000 1111 0000

TO DO



set 2
10

1. — check set
2. — check all tags
3. — check valid bits
4. — Read data if hit
   ↳ If miss goto K+1