Day 8

- Review: CPU works
- x86 call/return
- x86 (more: arrays, beyond)

- Memory Allocator (malloc lib)

- Admin: Survey
- Exam Review \[\text{(Mid: Thurs)}\] \[\text{Not Tues}\]
CPU:

while loop:

- fetch addr in eip
- decode: figure out what inst does
- execute: do the inst
call/return

cdecl (x86)
(calling
convention)

Questions:

1- How to call function?
2- How to return?
3- Return values?
4- Parameters (arguments)
5- Local variables
6- Access params/local in uniform way?
7- How to use registers safely?
1/2 - call/return

. \text{S} \Rightarrow \text{label}

\text{Call target}
\downarrow
\text{address}

\Rightarrow \text{change eip to target address}

\Rightarrow \text{pushes ret addr eip} \Rightarrow 2000
\text{inst after call}
\text{onto stack esp} \\
\text{1004}

\Rightarrow \text{ret [no arguments]}
\text{pops ret addr off stack}
\text{jumps to it}
3 - ret value

simple: 32 bits or less

\[ \Rightarrow \text{ %eax } \]

complex: other means

(larger) (stack)

4/5 - parameters / space for local variables

params/args:

push onto stack before call in "reverse" order

\[ \text{func (10, 20)} \]

N arguments:

- push N-1
- push N-2
- push 0

changes esp

moves value onto stack

subl *%esp
void func (int a, int b) {
    int c;
    int d;
    c = 0;
}

func (3, 6)

consistent way to refer to locals, args

new register: [ebp] base pointer
stable (some value)
while func executes
esp: 2008 2000
ebp: 2008

protocol:

1) establish ebp

⇒ called into a function
want to point ebp to
top of stack
+ after: make room for local var...

what inst?

[\text{movl \%esp,\%ebp}]

2) Problem: lost caller's ebp
upon entry to function:

before overwrite/update

(new) ebp

save old ebp

upon return:

restore old ebp

subl $4, %esp

movl source, (%esp)

subl $8, %esp  \( \text{room for locals} \)
entry:

pushl %ebp  
save old ebp

movl %esp, %ebp

exit:

movl %ebp, %esp  
deallocates locals
popl %ebp (dest)  
restores old ebp, decreases stack
ret

pop:

movl (%esp), dest
addl $4, %esp

esp ↑

ebp ↓
old ebp
ret addr

esp ↓

esp ↑

ebp ↓
old ebp
ret addr
Last Problem: use of registers

(\text{ebp} \rightarrow \text{save/restore})

\text{movl} \ $1, \%ecx \\
\text{call} \ \text{func} \\
\rightarrow \text{what happened to ecx?}

\underline{\text{caller-save}} \hspace{2cm} \underline{\text{callee-save}}

\text{if you care about these registers, save before call} \\
(\text{eax, ecx, edx}) \\
\text{if needed, save/restore registers (onto stack)} \\
\text{inside func, if func wants to use regs, save old, restore} \\
(\text{ ebx, esi, edi}) \text{ before return}
C : int increment (int x) {
    return x + 1;
}

Goal: write assembly version (save/restore ebp)

```
[ movl  ] = ret
[ addl  ]
[ pushl %ebp ]
[ movl %esp, %ebp ]
[ movl 8(%ebp), %eax ]
[ addl $1, %eax ]
[ movl %ebp, %esp ]
[ popl %ebp ]
[ ret ]
```

[Diagram:ebp stack frame with esp, ebp, old ebp, ret addr]
void increment (int* p) {
    *p = *p + 1;
}

write assembly!

pushl %ebp
movl %esp, %ebp
movl $1, (%eax)
addl (%eax)
movl %ebp, %esp
popl %ebp
ret

increment(3x);
```
addl $1, (%eax)
```

```
movl (%eax), %ecx
addl $2, %ecx
movl %ecx, (%eax)
```

RISC

```
eax : (address)
```

```
old ebp
ret addr
10
```

Diagram:
- esp
- ebp
- 10
Intro to C
(Some assembly)

C book

K+R: [Chaps 1, ..., 7] C book

B/O'H: [Chaps: (2, 3)]

Free: C operator precedence
x86 inst summary
x86 call/ret

Add: (no charge)
1 8.5"x11" sheet (both sides)

⇒ ...
C:

library:
- `malloc()`
- `free()`

Internals: (hwy)

```c
#include <stdlib.h>

void *malloc(size_t size)
{
  return fail(NULL);
}

void *malloc(size_t size)
{
  return new chunk of allocated memory

  (contiguous) [ptr, len]

  free(ptr)
}
```

makes that chunk available again
Implementation

track:
free := [free list]

node per free chunk
⇒ start addr
⇒ length

⇒ malloc (50)
⇒ look thru free list for size > requested amt

⇒ strategies:
First fit: find first free chunk > size req'd
Strategies

Worst Fit:
find biggest block

Best Fit:
find smallest block that can satisfy request

\[ p_1 = malloc(200); \]
\[ p_2 = malloc(200); \]
\[ p_3 = malloc(200); \]
\[ free(p_2); \]
\[ free(p_1); \]
\[ free(p_3); \]
\[ malloc(300); \]

\[ \Rightarrow \text{need: coalesce free chunks} \]
neighbor \( c_1, c_2 \Rightarrow C \text{ (total size)} \)
```c
malloc (150);
=> fail

heap: [fragmented]
chopped up into non-contiguous free spaces
Embed list: (into heap)

\[ \Rightarrow \text{ask OS for chunk of memory} \]

\[ \Rightarrow [\text{mmap( )}] \]

\[ \Rightarrow \text{contiguous region of mem of some size} \]

\[ \text{void* ptr = mmap( ..., )}; \]

\[ \text{malloc( ), free( )} \]

\[ \text{init: make list} \]

\[ 1 \text{ element} \]

\[ \text{start @ ptr} \]

\[ \text{size: 4096} \]

\[ \text{struct node \{ } \]

\[ \text{int length}; \]

\[ \text{struct node* next}; \]

\[ \}}; \]
void * ptr = mmap(...)

struct node * n = (struct node*) ptr;

n->length = 4096;
n->next = NULL;

malloc library:

[struct node * head;]

head = n;

malloc (100) & r

Head:

void * r = head;

head = (void*) head + 100;

& head->length = 3996;
head->next = NULL;

return r;
p = malloc (100)

free(p) ∪ free(2000);

struct header {
  int length);
  int magic;
};