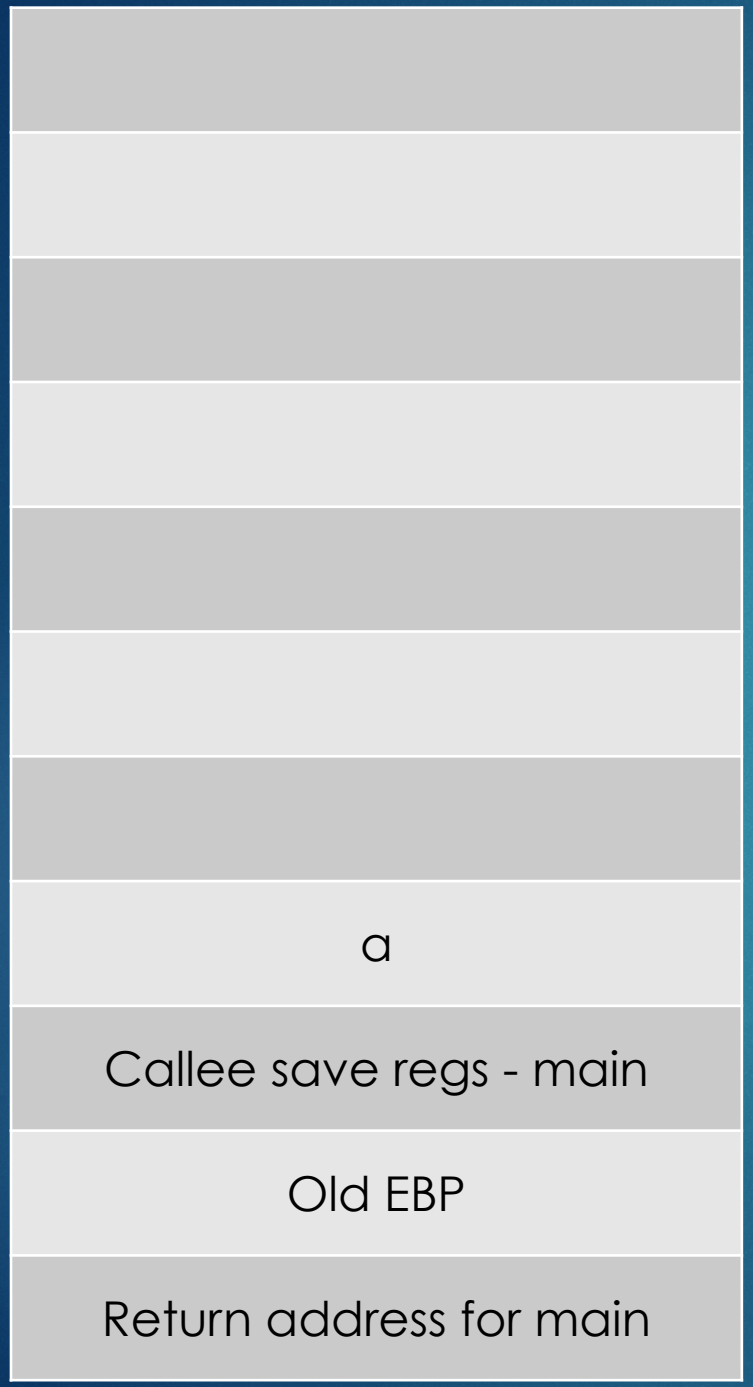
ECE 391 Discussion Week 3

Announcements & Reminders

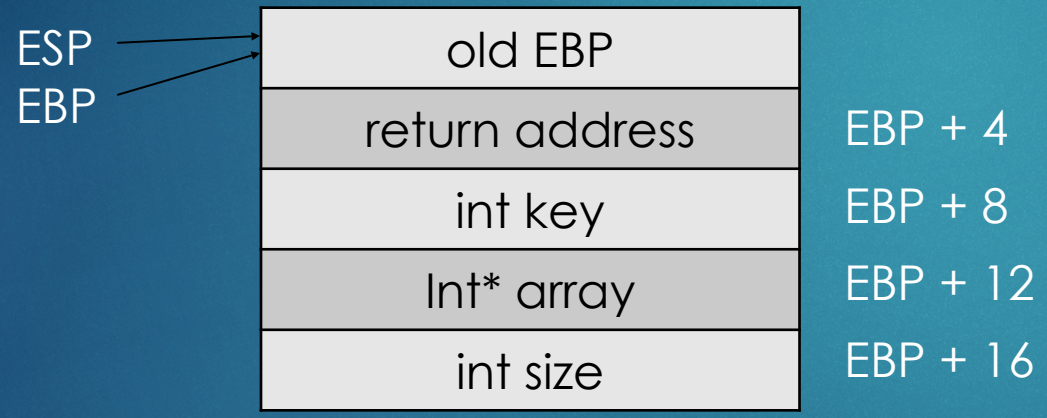
- MP1 Due next Tuesday, Sep 14 at 5:59pm (commit on GitLab)
  - Master branch for grading
  - Don't be late and show up to your assigned demo day!
  - Rubric will be posted soon – ask if something is unclear
- Read the whole document carefully before you start – there's a lot!
  - Document suggests an order to write the functions
  - Do NOT write the whole MP at once – write each IOCTL and then test it
  - Demo questions will be based off of the documents and what you wrote for the MP




```
int main() {
    int a;
    …
    a = magic( 7, 8, 9);
    …
}


// int magic(int a, int b, int c)
// Performs magical operation on
// the input args
magic:

    _______________

    _______________
    …

    _______________

    _______________
```

C Calling Convention

int binary_search(int key, int* array, int size);



Caller Saved	Callee Saved
EAX	EBX
ECX	ESI
EDX	EDI

Tasklet

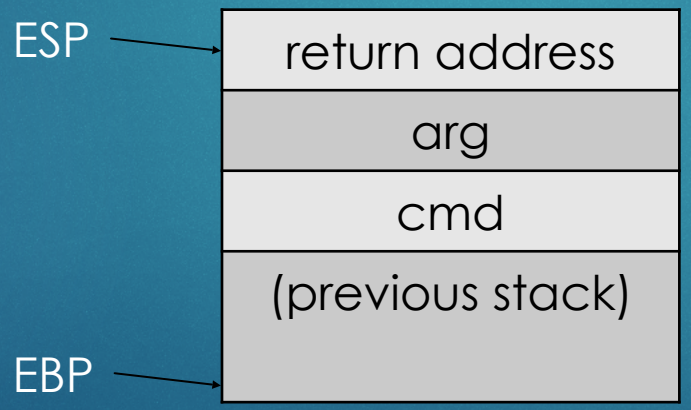
- A computer's real-time clock (RTC) can generate interrupts at a settable frequency
- Tasklets are interrupt handlers' way of deferring work
  - Interrupt handlers need to be as short as possible – why?
- In MP1, the tasklet is scheduled by the periodic RTC interrupt
- The tasklet is the main part of the MP1 game, which aggregates information and updates the state of the game and the screen

Virtual Memory Issue

- Virtual memory allows each user-level program to have the illusion of its own memory address space – why?
- When passing memory addresses between a user-level program and the kernel, a translation is needed
- Translation is provided by mp1_copy_to_user and mp1_copy_from_user
- If you improperly use them, things may still work in user space, but will break in kernel space

Dispatcher

- Dispatcher does **NOT** modify the stack
  - No need to setup stack frame
  - Other functions don't need to return to the dispatcher
  - Use the JMP instruction instead of the CALL instruction for a function call – why?



Debugging

- Debugging will be hard, especially in kernel
- Your code may run in user-space, but it may not run in kernel space
- In kernel mode, when it fails, it crashes everything
- Suggestion: identify issues in kernel mode, debug and fix in user space; or comment out the IOCTLs and test one by one

Coding Style in x86 Assembly

- Bad

  done1:

  done2:

  done3:

- Good

  success:

  free_mem:

  bad_param:

Coding Style in C

- Bad
  - tmp1, tmp2, tmp3
- Good
  - velocity_x, delta_y, acceleration_z

Magic Numbers

- Any number that appears in your code without a comment or a meaningful name
  - movl $1234, %ebx
  - int i = 5678;
- If a number appears only once, you can use a comment
  - cmpl $100, %ebx # loop for 100 times, so compare ebx with 100
- If a number appears more than once, define a constant (or make a label if you are using x86 assembly)
  - #define LOOP_COUNTER 100
- Note that the rules above does not apply to numbers that are used in conventions. E.g. movl 8(%ebp), eax is fine, as everyone who knows C calling convention should recognize the 8 in the code.

Other Things Related to Style

- No spaghetti code
  - Don't jump back and forth
  - Keep programs and functions short - create helper functions
- Comments
  - You have to comment your code
  - But don't comment on every line
- Function interface
  - Every function you write must have an interface