ECE 391 Discussion Week 2

Announcements & Reminders

- Auditors
  - It is YOUR responsibility to register for the course when space opens (this is not done automatically for you!)
- PS1
  - Due 2pm next Tuesday Sep 11th (ECEB third floor drop box #54)
  - MUST be done in groups of 3 (at least)
  - TYPE your solutions
  - Submit ONE copy
- MP1
  - Will be posted soon…
  - Start early
  - Lab will be full

Problem Set 1

- C and assembly do NOT have to have a 1-to-1 correspondence
- Where to look up x86 assembly instructions (GAS or AT&T Syntax)
  - https://courses.engr.illinois.edu/ece391/secure/references/doc-x86-asm.pdf
  - http://en.wikibooks.org/wiki/X86_Assembly
- Important things to remember
  - Initialize register before using it (xorl %eax, %eax)
  - Dollar sign "$" in front of immediate number (movl $5, %eax)
  - Star "*" is not dereference in x86 assembly
- Other stuff
  - "extern" in C
  - .GLOBAL or .GLOBL in x86 assembly

Function Pointers

- What does “typedef” do
  - typedef char int8_t;
  - typedef char * int8ptr_t;
- What is a “function prototype”
  - int func (int arg);
- Example of a function pointer in C

```
typedef int (*func_t) (int arg);
func_t my_func;
int foo (int arg);
my_func = foo;
```

Displacement

- displacement(base register, offset register, scalar multiplier)
- displacement + base register + (offset register * scalar multiplier)
  - Base register can be any register
  - Default value of displacement is 0
  - Offset register can NOT be esp
  - Scalar multiplier can be 1,2,4,8, default value is 1
- Example of displacement
  - movl -4(%ebp, %edx, 4), %eax
  - eax ← M[ebp – 4 + edx * 4]

Jump Table

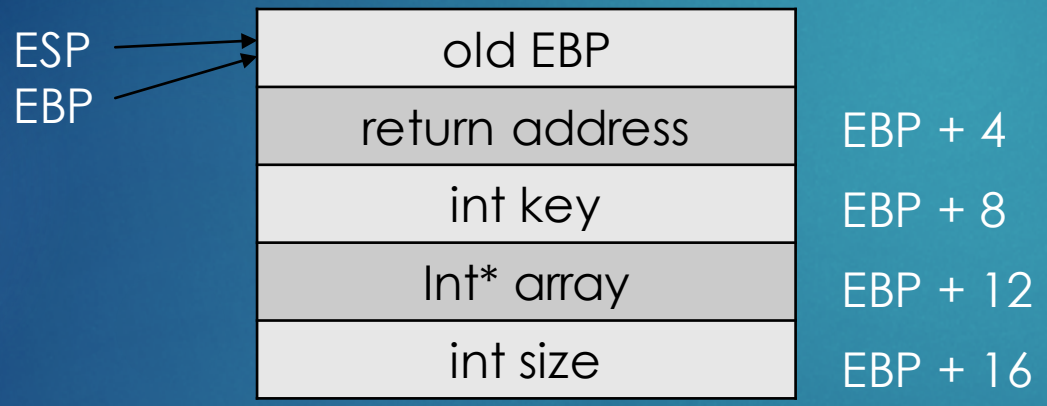
- Jump to the function whose address is given by a value
- Example of jump table in x86 assembly
  - jmp *operations-4(, %ecx, 4)
  - Remember "*" here is not for dereference
  - operations-4(, %ecx, 4) = M[operations – 4 + ECX * 4]
    = M[operations + (ECX-1) * 4]

C Calling Convention

- Rules for C's subroutine interface structure
  - How information is passed to a subroutine
  - How information is returned to the caller
  - Who owns (responsible for) which registers
- Understand Stack structure (be careful of diagrams online!)
  - Lower/Higher memory addresses toward top/bottom of stack, respectively
  - Push/Pop instruction decrements/increments ESP, respectively
- Caller sequence
- Callee sequence

C Calling Convention

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



Caller Saved	Callee Saved
EAX	EBX
ECX	ESI
EDX	EDI

Translation from C to x86 Assembly

- Binary Search
  - Task: find key by shrinking search space by half each time
  - Initialization: set search space to be entire array
  - Stopping conditions: search space is empty or middle item is one we want to shrink search space based on relative value of middle element
  - Return value: -1 on failure, otherwise index of element

	left		middle		right
Index →	**0**	**1**	**2**	**3**	**4**
Element →	1	3	6	7	9

Translation from C to x86 Assembly

```c
int bin_search(int key, int * array, int size) {

    /* declare and init variables */
    int left = 0;
    int right = size - 1;
    int middle;

    while (1) {

        /* check stopping condition */
        if (left > right) {
            return -1;
        }

        /* compute middle */
        middle = (left + right) / 2;

        /* check stopping condition */
        if (array[middle] == key) {
            return middle;
        }

        /* update search space */
        if (array[middle] < key) {
            left = middle + 1;
        } else {
            right = middle - 1;
        }
    }
}
```

Variable	Location
left	EAX
right	ECX
middle	EDX
array	EBX*
key	ESI*
size	EDI*

* = callee-saved register

Translation from C to x86 Assembly

```
bin_search:

    # initialization
    xorl    %eax, %eax
    movl    %edi, %ecx
    decl    %ecx

    # while loop
    loop:

        # check stopping condition
        cmpl    %ecx, %eax
        jg      not_found

        # compute middle
        leal    (%eax, %ecx), %edx
        shrl    $1, %edx

        # check stopping condition
        cmpl    %esi, (%ebx, %edx, 4)
        je      found
        jl      change_left
```

```
    change_right:
        leal    -1(%edx), %ecx
        jmp     loop

    change_left:
        leal    1(%edx), %eax
        jmp     loop

    not_found:
        movl    $-1, %eax
        jmp     return

    found:
        movl    %edx, %eax

    return:
        ret
```