X86 Crash Course Part 2

Program Call Stack
Stack Instruction
Branch Instructions
Rep Instructions
Program Call Stack

- Each active function call has a *frame* (that stores the values of all local variables) associated with it.
- The frames of active functions are maintained on a stack.
- A new frame is pushed onto the stack each time a new function call context is entered.
- When recursive functional calls are made, multiple frames in the stack may be associated with the same function, but with different call contexts of that function.
int g(int y) {
    int gi;
    char gc;
    ...
}
int f(float x) {
    int fi;
    ...
    return g(fi)
}
int main() {
    int mx = 100;
    printf("%d\n", f(3.2));
}
Function Calling Sequence (cdecl)

- Use **push** instructions to place arguments on the stack.
- Use **call** instruction to invoke the function.
  - Address of next instruction in caller is pushed on the stack by the **call** instruction
- Use **push ebp** to store the pointer to the caller's frame on the stack.
- Use **mov ebp, esp** to update **ebp** to point to the called function's frame.
- Evaluate the instructions in the function.
- Use **pop ebp** to restore the caller's frame
- Use **ret** to pop **eip** from the stack (restoring the caller's context)
- Restore the stack pointer by adding the number of bytes occupied by the parameters to it.
Stack Instructions

• **push value**
  - Decrements `esp` by the number of bytes occupied by `value`, and copies `value` to the address now referred to by `esp`.
  - Operand types: register, memory, immediate
  - Flags set: none

• **pop destination**
  - Copies bytes in memory from the address of `esp` to the destination, then increments `esp` by the size of the destination.
  - Operand types: register, memory
  - Flags set: none
Uncommon Stack Instructions (Often Used by Malware)

- **pusha**
  - Pushes the 16-bit register values $AX, CX, DX, BX, SP, BP, SI, DI$ to the stack.

- **pushad**
  - Pushes the 32-bit register values $EAX, ECX, EDX, EBX, ESP, EBP, ESI, EDI$ to the stack.

- **popa**
  - Inverse operation of pusha

- **popad**
  - Inverse operations of pushad
Function Call Instructions

- **call location**
  - Pushes the address of the next instruction on the stack, then set eip to the specified location.
  - Operand types: register, memory, immediate
  - Flags set: none

- **ret (value)**
  - Pops the stack into eip. If an operand is given, its value is added to the stack pointer.
  - Operand types: immediate
  - Flags set: none

- **leave**
  - Equivalent to the sequence
    ```
    pop ebp, esp
    ret
    ```
Unconditional Branch Instruction

- `jmp location`
  - Unconditional jump. Usually used to join the branches of if-then-else constructs together.
  - Operand types: register, memory, immediate

- You can compile test programs on remnux to see how this works. *(set disassembly-flavor intel in .gdbinit)*
  - gcc -g -o test test.c
  - gdb test
  - list
  - break line-number
  - x/20i $eip
Conditional Instructions

- Used to set flags for conditional branch instructions

  - **test destination, source**
    - Performs same function as `and`, but does not store the result in the destination.
    - Usually used to set the `z` flag (1 if operands don't share any bits, 0 otherwise). Often used with Boolean values (1 or all 1s = true, 0 = false).

  - **cmp destination, source**
    - Performs same function as `sub`, but does not store the result in the destination. Compares destination to source.
    - Used to set the `ZF`, `CF`, `SF`, and `OF`.

<table>
<thead>
<tr>
<th>Condition</th>
<th>Z</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>dst = src</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>dst &lt; src</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>dst &gt; src</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Conditional Branches

- Example: \texttt{jz location}

- The opcode tells what conditions is checked, \texttt{jz}: jump if zero, \texttt{jg} jump if greater, etc.

- The location, specified by an immediate operand tells the location to jump to. In an assembly program, this is specified by a label. The assembler translates this to a displacement from the current \texttt{eip} value.

- A jump can be \texttt{short} (within \(+127,-128\) bytes) or \texttt{near} (offset is either 2 or 4 bytes).
Conditional Branch Instructions

jz loc ; checks Z flag
jnz loc

je loc ; same as jz
jne loc

jg loc ; signed compare
jge loc

ja loc ; like jg, unsigned
jae loc ; like jge, unsigned

jl loc ; signed compare
jle loc

jb loc ; like jl, unsigned
jbe loc ; like jle, unsigned

jo loc ; checks OF
js loc ; checks SF

jecxz loc; checks ecx
Rep Instructions

- Intel *string instructions* provide operations on sequences of data.
- Most common: *movsx, cmpsx, stosx, scasx* where \( x \) is either \( b \) (byte), \( w \) (word) or \( d \) (double word).
- These instructions use registers \( esi \) (source index) and \( edi \) (destination index).
- Instructions will apply to exactly one unit of data unless prefixed by a *repeat prefix*, either \( rep, repe \) (\( repz \)), \( repne \) (\( repnz \)).
- Instructions increment \( esi \) and \( edi \) if direction flag \( DF=0 \). Otherwise, they decrement \( esi \) and \( edi \).
Repeating `movsb` and `cmpsb`

- `rep` repeats until `ecx` = 0
  - `repe` (or `repz`) repeats until `ecx` = 0 or `ZF` = 0
  - `repne` (or `repnz`) repeats until `ecx` = 0 or `ZF` = 1

- `rep movsb`
  Equivalent to C `memcpy`. Copies data at address in `esi` to address in `edi`. If `ecx` is 0, stop, otherwise, decrement `ecx` and continue.

- `repe cmpsb`
  Equivalent to C `memcmp`. Subtracts the value at address in `edi` from the value at `esi` and updates flags. If `ecx` = 0 or `ZF` = 0, the instruction terminates.
Next Class?

- Read Chapter 5. We'll start looking at disassembly with IDA.
- You should have completed all chapter 3 labs. If you had questions, post them to the associate discussion.