CONTROLLING EXECUTION | SEC204

Overview

- Introduction
- Unconditional branches
- Conditional branches

INTRODUCTION

BRANCHES

- EIP advances sequentially, unless otherwise directed
  - With prefetching/preloading, out-of-order execution, variable CISC instructions length, this is harder than simply incrementing by a static value
- EIP cannot be modified directly by the program (ie with a MOV instruction)
- Instructions that alter the value of EIP are called branches
  - Unconditional branches
  - Conditional branches

UNCONDITIONAL BRANCHES

UNCONDITIONAL BRANCHES

- Jumps
- Calls
- Interrupts

JUMP

- `jmp location`
  Jump EIP to memory address `location`


```
# jumptest.s - An example of the jmp instruction
.section .text
.globl _start
_start:
    nop
    movl $1, %eax
    jmp overhere
    movl $10, %ebx
    int $0x80
overhere:
    movl $20, %ebx
    int $0x80
```

At command line:

```
$ as –o jumptest.o jumptest.s
$ ld –o jumptest jumptest.o
$ ./jumptest
$ echo $?
$ objdump –D jumptest
```

CALLS

- `call address`
  Similar to jump, but it remembers where it jumped from and can return if needed
- `ret`
  Return to the original part of the code before the call function was called
  The stack is used to save the return address, local variables, function parameters

- Create calltest.s file with the following contents. Assemble, link, and run it on command line.


```
# calltest.s - An example of using the CALL instruction
.section .data
output:
   .asciz "This is section %d\n"
.section .text
.globl _start
_start:
   pushl $1
   pushl $output
   call printf
   add  $8, %esp
   call overhere
   pushl $3
```

Cont…

```
   pushl $output
   call printf
   add  $8, %esp
pushl $0
   call exit
overhere:
   pushl %ebp
   movl %esp, %ebp
   pushl $2
   pushl $output
   call printf
   add  $8, %esp
   movl %ebp, %esp
   popl %ebp
   ret
```

LEA – LOAD EFFECTIVE ADDRESS

- Often when using the stack, you will see the LEA instruction
- `lea operand1, operand2`
  Computes the effective address of operand1 and loads it into register operand2
- Example: `lea -4(%ebp), %eax`
  The memory address to be loaded into %eax will be 4 bytes less than the address stored in %ebp.

INTERRUPTS

- Hardware interrupts
  - Hardware devices generate hardware interrupts (ie an I/O port receives an incoming signal)
- Software interrupts
  - Programs generate software interrupts to hand off control to another program
  - When a program is called by an interrupt, the calling program is put on hold and the called program takes over
  - A lot of Microsoft DOS interrupts use the 0x21 software interrupt code
  - A lot of Linux programs use the 0x80 interrupt to provide low-level kernel functions

CONDITIONAL BRANCHES

CONDITIONAL BRANCHES

- Will only execute on condition (and value of EFLAGS register)
  - Carry Flag (CF)
  - Overflow (OF)
  - Parity Flag (PF)
  - Sign Flag (SF)
  - Zero Flag (ZF)
- Conditional jumps depend on these values
  - A subset of these jumps are shown in the table

Instruction	Description
JA	Jump if above (unsigned numbers)
JAE	Jump if above or equal (unsigned)
JB	Jump if below (unsigned)
JBE	Jump if below or equal (unsigned)
JCXZ	Jump if CX register is 0
JE	Jump if equal
JL	Jump if less (signed numbers)
JG	Jump if greater (signed numbers)
JGE	Jump if greater or equal (signed numbers)
JNE	Jump if not equal
JZ	Jump if zero

CMP

- `cmp operand1, operand2`
  compares operand2 to operand1. Subtracts (operand2-operand1) and sets EFLAGS based on the result (the original operands 1 and 2 are not modified)


```
# cmptest.s - An example of using
# the CMP and JGE instructions
.section .text
.globl _start
_start:
   nop
   movl $15, %eax
   movl $10, %ebx
   cmp %eax, %ebx
   jge greater
   movl $1, %eax
   int $0x80
greater:
   movl $20, %ebx
   movl $1, %eax
   int $0x80
```

At command line:

```
$ as –o cmptest.o cmptest.s
$ ld –o cmptest cmptest.o
$ ./cmptest
$ echo $?
```

LOOPS

- `Loop address`
  Loop to memory label address until the ECX register is zero.

```
# loop.s - An example of the loop instruction
.section .data
output:
   .asciz "The value is: %d\n"
.section .text
.globl _start
_start:
   movl $100, %ecx
   movl $0, %eax
loop1:
   addl %ecx, %eax
   loop loop1
   pushl %eax
   pushl $output
   call printf
   add $8, %esp
   movl $1, %eax
   movl $0, %ebx
   int $0x80
```


At command line:

```
$ as –o loop.o loop.s
$ ld –dynamic-linker /lib/ld-
linux.so.2 –lc –o loop loop.o
$ ./loop
```

If then else statement

1. Create an assembly program for the following c code:

```
int main()
{
   int a = 100;
   int b = 25;
   int greater = 0;
   if (a > b)
   {
      greater = a;
   } else greater = b;
   return 0;
}
```


In (gdb), watch the value of greater change:

```
break *main
run
x/4d &greater
step
x/4d &greater
cont
```

FURTHER READING

- Professional Assembly Language, chapter 6