USING NUMBERS | SEC204

Overview

- Introduction
- Integer arithmetic

INTRODUCTION

NUMERIC DATA TYPES

- The core numeric data types for the IA32 platform are:
  - Unsigned integers
  - Signed integers
  - Binary-coded decimal
  - Packed binary-coded decimal
  - Single-precision floating point
  - Double-precision floating point
  - Double-extended floating point

STANDARD INTEGER SIZES

- The basic IA-32 platform supports 4 integer sizes:
  - Byte: 8 bits
  - Word: 16 bits
  - Doubleword: 32 bits
  - Quadword: 64 bits

- What is the range of unsigned integers you can represent with a word?
- What is the range of signed integers (two's complement) you can represent with a word?

INTEGER ARITHMETIC

ADDITION

- `add source, desti`
  Adds source to destinati lt of the addition is placed in destination
- Can define the size of data element to be moved
  - **addl**: `l for 32-bit long word value`
  - **addw**: `w for 16-bit word value`
  - **addb**: `b for 8-bit byte value`
- Create an assembly program addnum.s that performs the following functionality


```
int main() {
   int data = -40;
   int b = 0;
   data = data +(-10)+(-200)+80+210
return 0;
}
```

In command line:

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

SUBTRACTION

- `sub source, desti`
  Subtracts source from de[illegible]estination-source). The result of the subtraction is placed in destination

- Can define the size of data element to be moved
  - **subl**: `l for 32-bit long word value`
  - **subw**: `w for 16-bit word value`
  - **subb**: `b for 8-bit byte value`

- Create an assembly program subnum.s that performs the following functionality

```
int main() {
   int data = 100;
   int b = 20;
   data = data –b –b –b –b –b –b -b
return 0;
}
```

In command line:

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

INCREMENTING, DECREMENTING

- `inc destination`
  Increases destination by
- `dec destination`
  Decreases destination by 1.

- Create an assembly program count50.s that performs the following functionality:


```
int main() {
   int data = 50;
   int b = 20;
   for (b=20; b<data; b=b+1) {
         printf("value of b: %d\n", b);
   }
return 0;
}
```


In command line:

```
$ as –o count50.o count50.s
$ ld –dynamic-linker /lib/ld-
linux.so.2 –lc –o count50
count50.o
$ ./count50
$ echo $?
```

MULTIPLICATING

- `mul source`
  For unsigned integers. The destination is implied (DX:AX) and is double the size of source.
- ```
  mov $5, %eax
  mull $10
  movl %eax, result
  ```
  Result has value of 50.
- Assemble, link and run program multest.s from DDE.
  ```

MULTIPLICATING

```
# multest.s - An example of                UL instruction
.section .data
data1:
   .int 315814
data2:
   .int 165432
#quad is 64-bits
result:
   .quad 0
output:
   .asciz "The result is %qd\n"
.section .text
.globl _start
_start:
   nop
   movl data1, %eax
   mull data2
   movl %eax, result
   movl %edx, result+4
   pushl %edx
```

Cont:

```
pushl %eax
  pushl $output
  call printf
  add $12, %esp
  pushl $0
  call exit
```

MULTIPLICATING

- `imul source`
  For signed integers. The destination is implied (DX:AX) and is double the size of source.
- `imul source, destination`
  For signed integers. The destination must be a register.
- `imul multiplier, source, destination`
  For signed integers. Multiplier is a value, source can be a register or value in memory, destination must be a register. Multiplier * source = destination

DIVISION

- `div divisor`
  For unsigned integers. The dividend is implied and must be stored in the AX register (if 16-bits), the DX:AX registers (if 32-bits), or the EDX:EAX (if 64-bits).

Dividend	Divident Size	Quotient	Remainder
AX	16bits	AL	AH
DX:AX	32bits	AX	DX
EDX:EAX	64bits	EAX	EDX

DIVIDING

```
# divtest.s - An example of                truction
.section .data
dividend:
   .quad 8335
divisor:
   .int 25
quotient:
   .int 0
remainder:
   .int 0
output:
   .asciz "The quotient is %d, and the remainder is %d\n"
.section .text
.globl _start
_start:
   nop
   movl dividend, %eax
   movl dividend+4, %edx
   divl divisor
```


Cont:

```
   movl %eax, quotient
   movl %edx, remainder
   pushl remainder
   pushl quotient
   pushl $output
   call printf
   add  $12, %esp
   pushl $0
   call exit
```

SIGNED DIVISION

- `idiv divisor`
  For signed integers. The [illegible] implied and must be stored in the AX register (if 16-bits), the DX:AX registers (if 32-bits), or the EDX:EAX (if 64-bits).

Dividend	Divident Size	Quotient	Remainder
AX	16bits	AL	AH
DX:AX	32bits	AX	DX
EDX:EAX	64bits	EAX	EDX

MULTIPLYING, DIVIDING BY SHIFTING

- `sal destination`
  `sal shifter, destination`
  Shift arithmetic left. If used, shifter specifies the number of bits to shift. In binary, sal multiplies by 2.

- `sar destination`
  `sar shifter, destination`
  Shift arithmetic right. If used, shifter specifies the number of bits to shift. In binary, sar divides by 2.

LOGICAL OPERATIONS

- `AND, OR, XOR`
  `xor source, desti`
  Performs logical XOR function. Destination holds the result. Same format for AND, OR.

- `not destination`
  Performs a NOT instruction. Each bit of destination is inverted.

FURTHER READING

- Professional Assembly Language, chapters 7-8