Gnu Assembly Manual for Intel Pentium Processors

Prof. Godfrey C. Muganda
Department of Computer Science
North Central College

General information: Most assembly language instructions have either one or two operands. An operand is characterized as being a *source* operand if it supplies data for the instruction, and it is characterized as a *destination* if it receives data that is the result of the instruction execution. An immediate operand cannot be a destination. An instruction may reference at most one memory operand. In an instruction that has both source and destination operand, the source operand is written first.

Instructions for floating point numbers are not covered.

Data Movement Instructions: These move data from a source to a destination, or exchanges the contents of two operands (register or memory).

```
movl source, dest
movb
movw
xchgl dest, dest
xchgw
xchgb
```

Arithmetic Instructions: These add a source to a destination, subtract a source from a destination, increment or decrement a destination by 1, negate a destination, and perform multiplication and division of signed integers. Other opcodes, `mul` and `div` exist to perform multiplication and division of unsigned integers.

```
addl source, dest
addw
addb
subl source, dest
subw
subb
incl dest
incw
incb
decl dest
decw
decb
negl dest
negw
negb
imull source
imulw
imulb
idivl source
idivw
idivb
```

In the multiplication and division case, the destination is assumed to be the accumulator (with size of the operand being taken into account), and the source operand may not be an immediate operand. Multiplication results in a double length result and division requires a double length result. The size extension opcodes discussed below are useful in converting an operand to double length prior to a division.

Size extension opcodes: All the following opcodes require the operand to be converted to be in the accumulator.

```
cbw
cbtw     //convert byte to word
cwd
cwtd     //word to double
cdq      //double word to quad word
cltd     //long to double long
```

Logical and Bitwise Operations: These perform logical (boolean) operations on a bitwise basis.

```
andl source, dest
andw
andb
orl source, dest
```

```
orw
orb
xorl source, dest
xorw
xorb
notl dest
notw
notb
```

Comparison: These opcodes simulate subtracting the source from the destination, and set the condition flags in the flags register so they can be tested by a subsequent conditional jump instruction. The flags reflect the result of the simulated subtraction. Thus the greater than flag is set if the result would have been greater than 0, or equivalently, if the destination is greater than the source.

```
cmpl source, dest
cmpw
cmpb
```

Conditional Jumps: These check the condition flags for the result of the last ALU operation, and jump to the specified address if the tested condition is satisfied.

```
jz address //zero
je        //equal
jne     //not equal
jnz
jl      //less
jle     //less or equal
jg      //greater
jge
jcxz    //cx is zero
jecxz   //ecx is zero
```

Looping Instructions: Decrements count register and jumps to specified address if count is not zero.

```
loop address
```

Address Manipulation: Computes the effective address of the source using whatever addressing mode is specified, and stores the result in the destination. Since the source is a memory operand (it does have an effective address) the destination must be a register.

```
leal  source, dest
```