Lecture Notes on Assembly Language - J. Vaughan

7. Instruction Layout

The usual way of writing an assembly language instruction is to lay it out on a single line according to the following fields:

label:		**opcode**	*operands*	*;comments*

Depending on the assembler, the colon following the label may not be required. The idea of fields dates back to the days of punched cards, when the fields were fixed in length (number of characters). Modern assemblers do not depend on fixed-length fields. Nevertheless, a fixed field approach, giving a columnar layout, helps to make an assembly language program more readable. The exception to this is when a series of lines containing only comments are placed at the beginning of a block of code in order to explain its operation. Each line then begins with a semicolon. Some assemblers also require that any blank lines be "commented out" with a semicolon.

8. Program Translation

An assembly language program is a text file that can be read by a program translator called an assembler. The assembler produces an object file containing machine code, as well as other information. The assembly language program contains instructions to the assembler, called directives or pseudo-instructions, as well as instructions for the target CPU. One entity that does not affect the object file is a comment. Comments are included in the source code for the sake of readibility. A comment in nasm is anything following a semicolon that occurs on the same line as the semicolon.

Any symbol occurring in the label field of a data definition directive or an instruction is given a value and stored by the assembler in a table called the s*ymbol table*. The value of a label on an equate directive is the value of the operand of the equate directive. The value of a label on a data definition directive or an instruction is the current value of the location counter at the time the label is inserted in the symbol table.

9. Program Structure: Preamble

A program should begin with an extensive comment section explaining such things as the name of the program, its dates of creation and last revision, the names of its creator and reviser, the commands required to assemble, link and run it, the inputs required and how these are provided, and the outputs produced and where these are stored and/or displayed. For example:

```
;
; hello.asm       a first program for nasm for Linux, Intel, gcc
;
; created:        12 January 2007     Revised: -
; creator:        P. Jones            Reviser: -
;
; assemble:       nasm -f elf -l hello.lst hello.asm
; link:           gcc -o hello hello.o
; run:          hello
;
; input:        none
; output:       "Hello World" appears on stdout, followed by a newline.
;
```

10. Program Structure: Data Definition

Following the introductory comments, the program is divided into sections defining data and executable instructions. The nasm assembler uses the section directive for this purpose.

The first section is that dealing with data definition. There are four general types of data: constants for which storage is not reserved in the data definition section, constants for which storage is reserved (such as strings), initialised variables and uninitialised variables. Constants for which memory is not reserved in the data section tend to be defined as a correspondence between some value and an appropriately meaningful symbol, for example in the following equate directive that allows the ASCII values for line feed, space and colon to be represented by their customary symbolic names:

```
LF          equu 0xa
SPACE       equ 20h
COLON       equ 3ah
```

As the assembler translates the program, it constructs a memory image, comprising addresses and contents of blocks of memory locations where the program is to be stored during execution. In order to keep track of the address of the next available location in the memory image, the assembler maintains a variable called the location counter. The value of this variable can be referenced from within an nasm assembly language program by the dollar symbol, $. Note that this variable only exists for the duration of the program translation process. It is completely different from the similarly-named processor register called the Program Counter, or PC. The most common use of a program reference to the location counter is for easy calculation of string length, as in the following example:

```
msg1:  db "REGISTER CONTENTS: ", 0ah
msg1len equ $-msg1
```

Since storage is not immediately reserved for such constants in the memory image, they do not need to be preceded by a section directive.

When storage is reserved in the memory image, a section-defining directive is usually necessary. In nasm, there are two sections that fulfil this function: the .bss and the .data sections.

The .data section contains definitions for constants needing reserved storage and for variables that need to contain initial values. The directives in this section are those that reserve and initialise memory as follows:

Reserve and initialise a byte or series of bytes	db
Reserve and initialise a word or series of words	dw

Reserve and initialise a doubleword or series of doublewords	dd
Reserve and initialise a quadword or series of quadwords	dq
Reserve and initialise ten bytes or series of ten bytes	dt

Examples

```
        SECTION .data
BLKSIZE   equ 64
msg:      db "Hello World", 10
eaxlab:   db " eax: "
var1      dw 65000
var2      dd 135678
hexnum:   db "0123456789ABCDEF"
patrn     db 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15
patrn1:   times (BLKSIZE/4) db 0xa5
```

The .bss section is used to declare variables whose size is defined but whose value is not defined at the time of program translation. The directives used in this section are as follows:

Reserve a byte or series of bytes	resb
Reserve a word or series of words	resw
Reserve a doubleword or series of doublewords	resd
Reserve a quadword or series of quadwords	resq
Reserve ten bytes or series of ten bytes	rest

Examples

```
        SECTION .bss
eaxsave:  resb 4
ebxsave:  resw 2
ecxsave   resd 1
hexdump:  resb BLKSIZE*3 + 7*BLKSIZE/16
```