Summer 2003 Lecture 3 06/11/03

Registers

AX = AH | AL
BX = BH | BL
CX = CH | CL
DX = DH | DL

Assembly Language Programming – New Assembler Directives

An Intel X86 assembly language program is made up of segments. All code and data for a program is declared within the context of some segment. The programmer declares segments via the **segment** directive. The first time that the segment directive is used, the segment parameters can be specified (alignment and combine class). Each subsequent time, they are not necessary, but if given, must match the original declaration. If a project has multiple source files, it is generally a good idea to put all of the segment declarations in a common include file and include that file in each source module. This ensures that the segment declarations agree between modules.

SEGMENT
ENDS
ASSUME
END
ORG
INCLUDE

What the Assembler Does

An assembly language program is made up of a sequence of statements. Each statement either

- **declares data,**
- **specifies an instruction,**
- **or is a directive to the assembler.**

Directives (or pseudo-ops) give instructions to the assembler, but do not generate any output.

The **assembler is a translator** which converts the assembly language source code into a binary object model that contains the data and code specified by the programmer, and is in a form suitable for input to a linker, which builds the actual executable program.

The assembler performs two essential operations.

- First, it translates the humanly readable source text into the appropriate binary instructions and data.
- Second, it constructs a symbol table containing addresses and symbolic constants. The assembler assigns symbolic names to a data items and locations in the program. The assembler then assigns addresses to these symbolic names, relieving the programmer of the necessity of assigning addresses, or remembering the addresses.

Most assemblers perform two passes over the source text. The first pass is used to count the sizes of the instructions and data items, and assign addresses to all of the labels in the program. The second pass is used to actually generate the output.

Types of Constants

Decimal:	A decimal number is made up of decimal digits (0-9) with no suffix
Hexadecimal:	A hex number is made up of hex digits (0-F). The first digit must be numeric to distinguish it from a symbolic name, and there is an 'h' as a suffix on the end of the number
Binary:	A binary number is made up of binary digits (0,1) with a 'b' suffix at the end.
Character:	A character constant is an ASCII character code between two single quotes as delimiters.
String:	A string constant is a sequence of ASCII characters between two double quotes as delimiters. String constants can only be used in data declaration statements. Only the characters between quotes are stored **(there is no 0 placed at the end automatically as in C)**.

Additional Pseudo-Ops

OFFSET	- get offset portion of pointer
SEG	- get segment portion of pointer
BYTE PTR	- specify data type is BYTE
WORD PTR	- specify data type is WORD
DWORD PTR	- specify data type is DWORD
FWORD PTR	- specify data type is FWORD
PUBLIC	- specifies that a symbol is visible outside of the module
EXTRN	- specified that a symbol is defined outside of the module

DOS and BIOS Function Call Interrupts:

The software interrupt mechanism is used to make calls to DOS and BIOS services. A software interrupt is like a special kind of subroutine call. The procedure is to load some parameters into certain registers and then perform the appropriate software interrupt. DOS or the BIOS will perform the operation and then return (possibly returning values in registers or flags).

Processor Instruction Set

Seven Basic types of instructions

1. Data transfer (including stack operations)
2. String operations
3. Arithmetic
4. Bit manipulation
5. Loops and jumps
6. Subroutine and interrupt
7. Processor control

Data Transfer Instructions

MOV — Move Data

General forms:

```
mov   reg,idata
mov   mem,idata
mov   reg,reg
mov   reg,mem
mov   mem,reg
```

MOV — Move Selector/Segment

General forms:

```
mov   sreg,reg
mov   sreg,mem
mov   reg,sreg
mov   mem,sreg
```

NOTE: There is no move immediate into segment register