INTRODUCTION TO ASSEMBLY LANGUAGE

CS 348
Implementation of Programming Languages Lab
Computer Science and Engineering Department
Indian Institute of Technology
Guwahati

Assembly Language Statements

- Three types of statements in assembly language are:

  [label:] mnemonic [operands] [;comment]

1. Executable Instructions
   - Generate machine code for the processor to execute on runtime
   - Instructions tell the processor what to do.
2. Assembler Directives
   - Provide information to the assembler while translating a program
   - Used to define data, select memory model, etc.
   - Non executable: directives are not part of instruction set.
3. Macros
   - Shorthand notations for a group of statements.
   - Sequence of instructions, directives, or other macros.

Instruction Examples

- No operand

```
stc                 ; set carry flag
```

- One operand

```
inc eax             ; increment register eax
call clrscr         ; call procedure clrscr
jmp L1              ; jump to instruction with label L1
```

- Two operand

```
add ebx, ecx        ; register ebx = ebx + ecx
sub var1, 25        ; memory variable var1 = var1 - 25
```

- Three operand

```
imul eax, ebx, 5    ; register eax = ebx *5
```

Comments

- Comments are very important!
  - Explain the program's purpose
  - When it was written, revised, and by whom
  - Explain data used in the program
  - Explain instruction sequences and algorithm used
  - Application-specific explanations
- Single-line comments
  - Begin with a semicolon and terminate at end of line
- Multi-line comments
  - Begin with COMMENT directive and a chosen character.
  - End with the same chosen character.

Instructions

- Assembly language instructions have the format:
- Instruction Label (optional)
  - Marks the address of an instruction, must have a colon :
  - Used to transfer program execution to a labelled instruction.
- Mnemonic
  - Identifies the operation (e.g. MOV, ADD, SUB, JMP, CALL)
- Operands
  - Specify the data required by the operation
  - Executable instructions can have zero to three operands
  - Operands can be registers, memory variables, or constants

Directives

- .data directives
  - Defines an area in memory for the program data
  - The program's variables should be defined under this directive
  - Assembler will allocate and initialize the storage of variables
- .text directives
  - Defines the code section of a program containing instructions
  - Assembler will allocate and initialize the storage of variables

Numeric constants

```
mov     eax,200          ; decimal
mov     eax,0200         ; still decimal
mov     eax,0200d        ; explicitly decimal
mov     eax,0d200        ; also decimal
 mov     eax,0c8h         ; hex
 mov     eax,$0c8         ; hex again: the 0 is required
 mov     eax,0xc8         ; hex yet again
mov     eax,0hc8         ; still hex
 mov     eax,310q         ; octal
 mov     eax,310o         ; octal again
 mov     eax,0o310        ; octal yet again
mov     eax,0q310        ; octal yet again
 mov     eax,11001000b    ; binary
 mov     eax,1100_1000b   ; same binary constant
```

Character and String Constants

- A character constant consists of a string up to eight bytes long.

- A character constant with more than one byte will be arranged with little-endian order.
  - Eg. mov eax, 'abcd'
- String constants are character strings used in the context of pseudo instructions, like the db family.
- A string constant is treated as a concatenation of maximum size character constants.
  - db 'hello' ; string constant
  - db 'h','e','l','l','o' ;equivaent character constant

Floating Point Constants

- Floatting point constants are acceptable as arguments to db, dw, dd, dq, dt and do.
- Floating point can be used as special operators like __float8__, __float16__, etc.
- Floating points are expressed as digits, followed by a decimal, then one more digit, and e followed by an exponent.
- The special operators are used to produce floating point numbers in other contexts.
- NASM cannot perform compile time arithmetic on floating point constants.

Examples

```
db    -0.2                        ; "Quarter precision"
dw    -0.5                        ; IEEE 754r/SSE5 half precision
dd    1.2                         ; an easy one
dd    1.222_222_222               ; underscores are permitted
dd    0x1p+2                      ; 1.0x2^2 = 4.0
dq    0x1p+32                     ; 1.0x2^32 = 4 294 967 296.0
dq    1.e10                       ; 10 000 000 000.0
dq    1.e+10                      ; synonymous with 1.e10
 dq    1.e-10                      ; 0.000 000 000 1
 dt    3.141592653589793238462 ; pi
do    1.e+4000                    ; IEEE 754r quad precision
```

Expressions

- Expressions are similalr to syntax in C.
- Bitwise OR Operator: The | operator gives a bitwise OR.
- Bitwise XOR Operator: ^ provides the bitwise XOR operation.
- Bitwise AND Operator: & provides the bitwise AND operation.
- Bit Shift Operators: << and >> are bitwise shift operators.
- The operators for Add, Substract, Multiply, Divide and Modulo are same as C. Signed division operator is // and signed modulo operator is %%.

Macros

- NASM supports two form of macros.
- Single line macros are defined using %.
  - Eg. %define isTrue 1
- Multi line macros are defined similar to a function in C.
  - Eg. %macro prologue 1

```
Push ebp
 Mov ebp,,esp
Sub esp,%1
%endmacro
```

Assembler Directives

- NASM directives are of two types:
  - User Level Directives: They are implemented as macros that call primitive forms.
  - Primitive Directives
- BITS: Specifying Target Processor Mode
  - The BITS directive specifies whether NASM should generate code designed to run on a processor operating in 16-bit mode, 32-bit mode or 64-bit mode.
  - The syntax is BITS XX, where XX is 16,32,64.
  - Changes need to be made accordingly.

Assembler Directives Contd.

- DEFAULT: Change the assembler defaults.
    - The DEFAULT directive changes the assembler defaults.
- SECTION or SEGMENT: Changing and defining sections.
    - The SECTION directive changes which section of the output file the code you write will be assembled into.
- ABSOLUTE:
    - The ABSOLUTE directive can be thought of as an alternative form of SECTION: it causes the subsequent code to be directed at no physical section, but at the hypothetical section starting at the given absolute address.
- EXTERN:
    - EXTERN is similar to the C keyword `extern`: it is used to declare a symbol which is not defined anywhere in the module being assembled, but is assumed to be defined in some other module and needs to be referred to by this one.

Assembler Directives Contd.

- GLOBAL:
  - `GLOBAL` is the other end of `EXTERN`: if one module declares a symbol as `EXTERN` and refers to it, then in order to prevent linker errors, some other module must actually *define* the symbol and declare it as `GLOBAL`. Some assemblers use the name `PUBLIC` for this purpose.
- COMMON:
  - The `COMMON` directive is used to declare *common variables*. A common variable is much like a global variable declared in the uninitialized data section
- CPU
  - The CPU directive restricts assembly to those instructions which are available on the specified CPU.

Assembler Directives Contd.

- FLOAT:
  - By default, floating-point constants are rounded to nearest, and IEEE denormals are supported. There are options that can be set to alter their behaviour.
- WARNING:
  - The `[WARNING]` directive can be used to enable or disable classes of warnings in the same way as the `-w` option

Program Template

```
TITLE  Program Template (Template.asm)
;Program Description:
;Author:
;Creation Date:
;Modification Date:
Global _start
section .data
                        ; (Insert variables here)
section .text
_start:
```

Program Example 1

```
global _start
_start:

Mov eax,1
Mov ebx,42
Int 0x80
```