2022 Spring Term (1) Fundamentals of Computer Systems CSC 215 5000[3335] (Borough of Manhattan CC)

     Assembly Language

Assembly Language

Setting up Assembly Language in Visual Studio

Attached Files: Assembly Language Program Template.txt (184 B)

Videos have been added to the end of this item for informational purposes. We will follow instructions found by following the Textbook Website link below.

Visual Studio is an Integrated Development Environment (IDE) that supports numerous languages and environments. The links found below present ways of setting up Visual Studio to support Microsoft's Assembly Language using the routines provided by the text's author Kip Irvine.

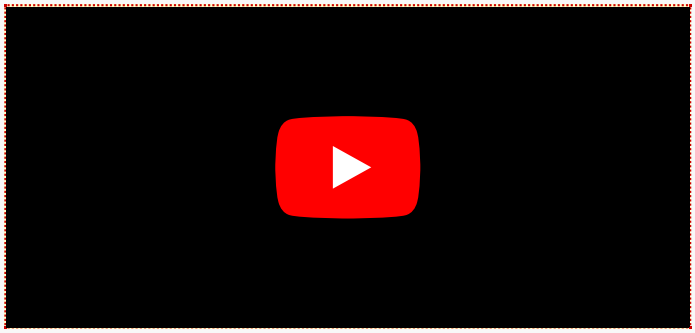
Learning an Assembly language is something of a chicken and egg scenario suggested by the famous question "Which came first? A chicken or an egg". In the chicken's evolutionary development, the question is valid and unanswerable! Which did come first? Learning Assembly language is similar because in order to learn an Assembly language, a learning programmer must be familiar with ideas and concepts he/she has likely not yet become familiar.

The author provides a solution to the problem by providing a set of routines that perform common tasks that require a level of operating system knowledge as well as fore knowledge of advanced Assembly language. The links below provide various descriptions of setting up Visual Studio to easily support both Irvine's routines and the learning Assembly language program.

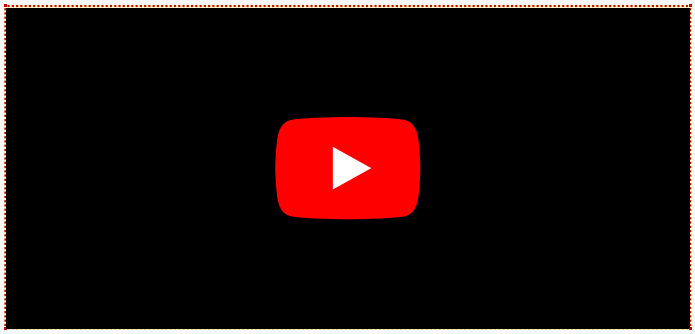
- Textbook Website. The easiest way of setting up Visual Studio is to start with a tutorial provided by the text's author. Here you can download the routines as well as a Visual Studio project with the appropriate settings changed to support Assembly language. This is by far the easiest method to setting up Visual Studio.
- Setting up Assembly Language in VS 2019. This web page presents a more comprehensive description of changes that need to be made in the Visual Studio environment to support Assembly language. This page is presented to those students that want a more comprehensive knowledge of the Visual Studio environment.

Videos

- How to setup Visual Studio for Assembly Language.



- Setting up Irvine



Introduction

Attached Files:
ANATOMY OF AN ASSEMBLER LANGUAGE PROGRAM.pdf (154.323 KB)

Introduction to Assembly Language

- Types/Categories of Programming languages

    1. Compiled - A compiled language uses a program called a compiler that translates computer code (source file) into an executable format. Typically, the source file is a high level language, such as C++, that is translated to an executable form known as machine code. Machine code is executed natively on a CPU.

    2. Assembled - An assembler language is a symbolic language that is closest to machine language. It performs a function similar to a compiler but rather than being a high level language, an assembly language is considered a low level programming language. Assembly languages are

associated with specific chip sets. When assembled and linked, the resulting machine code, known as an executable file, is executed natively on a CPU.

3. Interpreted - An interpreted language is said to be compiled in real time in a process known as just in time (JIT) compilation. Interpreted languages are read by a program known as an interpreter. Some interpreters create a virtual machine (VM) environment in which programs are read, compiled, and executed. Interpreted languages are not tied to a chip set but can be interpreted and run on many computers.

- The Process of Assembling a Program

  The process of assembling a program, sometimes call the compilation process, is the process of converting a source file (a text file containing the text of an Assembly language program) into an executable program capable of being executed on a computer. These are the steps:

  1. A programmer uses a text editor to create an ASCII text file containing the text of the program. This file is called the *source file* and in MASM, the version of Assembly language we are using, the file typically has the `.asm` extension. The Visual Studio Integrated Development Environment, or IDE, will create this file.

  2. The Assembler is a program that reads in one or more source files check for syntax errors and, if none are found, will create an *object file*. The object file has a `.obj` extension and contains the machine translation of the source file. If there are errors, the programmer must correct those errors before attempting to assemble the program again.

  3. The Linker is a program that reads the object file to determine if the program contains calls to procedures in a link library. The linker will copy an required procedures from the link library and combines them with the object file. If successful, an *executable file* is created.

     (I think the text makes it seem that the assembling and execution of a program are dependent activities when they are not. The compilation process produces an executable file that can be executed whenever it needs to be without the need to assemble it each time.)

- Basics of Assembly Language
  - Integer Constants - An integer constant (integer literal) has the syntax:

    ```
    [ (+ | -) ] digits [radix]
    ```

    where *radix* may be one of the following (upper or lower case)

Radix	Meaning
`h`	Hexadecimal
`o | q`	Octal
`d | t`	Decimal
`b | y`	Binary
`r`	Encoded real

Examples

Constant Value	Meaning
`26`	Decimal 26(constants without an explicit radix are considered decimal)
`26d`	Decimal 26
`10111101b`	Binary 189_{10}
`10111101y`	Binary 189_{10} (Alternate Form)
`237o`	Octal 159_{10}
`237q`	Octal 159_{10}
`237h`	Hexadecimal 567_{10}

- Integer Expressions - An integer expression is a mathematical expression involving integer values and operators. Operators follow and order of precedence as follows:

Operator	Name	Precedence
`( )`	Parentheses Used to clarify the order of operations	1
`+, -`	Unary plus and minus Positive and negative	2
`*, /`	Multiplication and Division	3
`mod`	Modulus Returns remainder of integer division	4
`+, -`	Addition and Subtraction	5

- Real Number Constants - A real number constant represents a number with a decimal component. Within memory they are represented quite differently than an integer value. When representing a real number constant, a decimal point and at least one digit are required. The syntax of a real number constant is:

```
[(+ | -)] integer.[integer] [exponent]
```

Where `exponent` is represented as

```
E[(+ | -)]integer
```

Examples

Value	Meaning
`2.`	Real number positive 2 Positivity is implicitly defined The decimal point and at least one number is required
`+2.0`	Real number positive 2 Positivity is explicitly defined
`-2.0E+5`	Real number -200,000
`2.E5`	Real number 200,000

- Character Constants - A character constant is a single character enclosed, or encapsulated, in single or double quotes. Most languages understand a single character as being enclosed in single quotes and that is the standard we will use in class. There are one or two single bit characters that should enclosed in double quotes rather than single quotes - the single quote character for example should be represented as `"'"`.

- String Constants - A string constant is a sequence of ASCII characters enclosed, or encapsulated, in single or double quotes. Most languages understand a string being enclosed in double quotes and that is the standard we will used in class. Other languages understand this concept of a string more in line as a character array while a string is a sequence of characters where the last character is binary zero or null. To differentiate between this idea of a character array and string, we will assume a string is terminated by a null.
- Reserved Words - Reserved words have special meaning to the Assembler and preprocessor and should only be used in their original context. Programmers should not use them in whole or as part of an identifier. Other languages use reserved words and key words. These are types of reserved words:
    - Instruction mnemonics
    - Register names
    - Directives
    - Attributes
    - Operators
    - Predefined symbols
    - Preprocessor directives

- Identifiers - An identifier is a programmer defined name that identifies variables, constants, procedures, or labels. There are rules for creating identifiers:
    - may be 1-247 characters
    - are not case sensitive
    - first character must be a letter [`A-Z`, `a-z`], an underscore [_],`@`, `?`, `$`. Subsequent characters may include digits [`0-9`] (The first character may not be a digit)
    - cannot be the same as a reserved word. You may not want to include the text of a reserved word as part of an identifier.

- Directives - A directive is a command to the Assembler that is embedded in the source code that is recognized and acted upon by the Assembler. Directives do not execute a runtime. Directives can define variables, macros, and procedures.

- Instructions - An instruction is a statement that becomes executable when a program is assembled. There are four parts:
    - Label (optional)
    - Instruction Mnemonic (required)
    - Operand(s)(as required by the instruction mnemonic)
    - Comment (optional)

- Label - A label is an identifier that acts as a place marker for instructions and data.
    - Data label - A data label identifies the location of a variable; it gives a name to the address of a construct such as a variable
    - Code label - A code label, place in the .code portion of a program ends in a semi-colon (`:`) and identifies targets for jumping and looping.

- Instruction Mnemonic - An instruction mnemonic is a short word that

describes an instruction. In English, a mnemonic is a device that assists in remembering.

- Operands - An operand is that upon which an instruction operates. In Assembly language, an instruction may have zero to three operands depending on the instruction mnemonic.

- Comments - Comments are an important part of documenting a program and of communicating identification information about a program. Organizations may have strict rules about the quality and position of comments within a program.
  - A single line comment begins with a semi-colon (`;`) and continues until the end-of-line (EOL)
  - Block comments begin with the COMMENT directive and a user specified symbol used to encapsulate the comment. An example:

```
COMMENT !
     a comment
     another comment
!
```

    You should be consistent with a program and define the user specified symbol that can be used throughout the program rather than using a different symbol for each comment.

The Process of Assembling a Program

The process of assembling a program, sometimes call the compilation process, is the process of converting a source file (a text file containing the text of an Assembly language program) into an executable program capable of being executed on a computer. These are the steps:

1. A programmer uses a text editor to create an ASCII text file containing the text of the program. This file is called the *source file* and in MASM, the version of Assembly language we are using, the file typically has the `.asm` extension. The Visual Studio Integrated Development Environment, or IDE, will create this file.
2. The Assembler is a program that reads in one or more source files check for syntax errors and, if none are found, will create an *object file*. The object file has a `.obj` extension and contains the machine translation of the source file. If there are errors, the programmer must correct those errors before attempting to assemble the program again.
3. The Linker is a program that reads the object file to determine if the program contains calls to procedures in a link library. The linker will copy an required procedures from the link library and combines them with the object file. If successful, an *executable file* is created.

I think the text makes it seem that the assembling and execution of a program are dependent activities when they are not. The compilation process produces an executable file that can be executed whenever it needs to be without the need to assemble it each time.