Available online www.jsaer.com

Journal of Scientific and Engineering Research, 2016, 3(1):17-20

Review Article

ISSN: 2394-2630
CODEN(USA): JSERBR

The Forgotten Assembly Programming Language

Hieu Vu

American University of Nigeria, School of Information Technology and Computing, Yola, Adamawa State, Nigeria

Abstract Assembly language was one of the earliest programming languages developed and used in business. It is the one and only one classified as low-level language, and difficult to learn and to program. As with the rapidly improvement in high technology, the programming paradigms is also change, and the Assembly language seems to be forgotten. However, Assembly programming is considered as an art in programming, because the programmers must know the logic of the program and internal detail of the hardware system.

Keywords Assembly, Programming language, Low-level language, High-level languages, CPU, C++, Java.

1. Introduction

It is important to know that the lifeblood of computer science is software development. For a computer scientist, programming is a primary tool to communicate with a computer and to program the computer what to do. This is why a part of computer science curricula is the study of programming. As computing hardware becomes faster, and more powerful, application software also improved and becomes more complicated. New generation of computer's software can do a lot more than the previous generation can, and for the programmers, the job of developing the application becomes more complicate and challenging. The heart of software development is problem solving [1]. To develop a software application, the developers or programmers need problem solving skill and a programming language. The programming languages have evolved over the time, from machine language to low-level language (Assembly) then to high-level language (COBOL, FORTRAN, C, etc...) and to object oriented language such as C++, Java, VB, and C#. There are numerous programming languages that have been used today such as C++, Java, Visual Basic, COBOL, Assembly, etc. Each programming language has its own characteristics, and usability. This research paper introduces, studies some special characteristics of the Assembly language and its relationships with high level languages such as C++ and Java.

2. Problems

Programming languages have been used by programmers to develop applications for many years. These professionals were trained to work with a particular programming language, which was adopted by their firms. The programmers are knowledgeable, and very good in doing their job. But when you ask a programmer questions such as "How the computer executes the instructions?" or "What really happen inside the computer's CPU, and the RAM memory?". He or she might not be able to provide clear answers to these questions, because they don't know the internal details of the system. The answers for these questions are embedded inside the Assembly language, and it is sadly to know that many colleges and universities are dropping the Assembly language from their computer science curricula. A computer-related professional should also be aware of the development of new programming language. What are the new features, and advantages of a new programming language? How does it change and impact the programming model?

3. The Assembly Language

Since the development of computer, in the early 1950's all programming was done in machine language. Machine language is also called Binary language because it includes only numbers 0s or 1s. The Assembly language is classified and the only low-level language. It is not a complex language but is awkward for human to work with. In the late 1950s, IBM developed the Assembly language as one of the first programming language to run on its System 360. The Assembly language assembles a series of symbolic representations of machine language operation codes. These symbolic representations are called Mnemonic operation codes [2].

From the early years, Assembly language had been used in commerce to develop applications in banking, accounting, and insurance industries. Today, few of these applications are still being used most were converted to the other newer, easier to program, and more powerful programming language.

A sample Assembly program

Programming problem 3.1, page 60. (Assembly Language by Jones) [3]

```
; This program “datePgm” will use macro _getDate and
; prints out the current date (Today date)
INCLUDE PCMAC.INC        ; Include Macros
      .MODEL SMALL       ; For IBM/PC
      .STACK 100h        ; Reserve for Stack
;- Data for the program -;
      .DATA              ; Data segment
CR    EQU 13             ; Declare carriage return constant
LF    EQU 10             ; Declare line feed constant
NSTR  EQU '$'            ; Declare end-string constant
msg   DB  'Today is: $'  ; Declare string output
saveAX DW ?              ; Variable to save register AX
saveDX DW ?              ; Variable to save register DX
;-     Main procedure    -;
      .CODE              ; Code segment
      EXTRN putDec:NEAR  ; Use subprogram putDec
datePgm PROC ; Begin procedure
      mov ax, @data      ; Requirement
      mov ds, ax         ; Boiler place
      _putStr msg        ; Write output message
      _getDate           ; Get System Date
      mov saveDX, dx     ; Save Month and Day
      mov al, dh         ; Move month to register AL
      mov ah, 0          ; Clear high value in AH
      call putDec        ; Write Month
      _putCh '/'         ; Write '/'
      mov dx, saveDX     ; Restore Month, Day
      mov al, dl         ; Move day to register AL
      mov ah, 0          ; Clear high value in AH
      call putDec        ; Write Day
      _putCh '/'         ; Write '/'
      mov ax, cx         ; Move year to register AX
      call putDec        ; Write Year
      _Exit 0            ; Exit program
```

```
datePgm ENDP            ; End datePgm procedure
END DatePgm             ; End program
```

Output

```
C:\ASM>datePgm
Today is: 2/3/2004
C:\ASM>
```

This program uses macro _getDate to get the system date. After the call, the assembler will automatically place the month in register DH, day in DL, and the year in register CX. The other macros _putCh used to print the slash (/) will destroy the contents of both registers AX, and DX therefore we need to save values in both registers before calling the macro. To display numeric value such as the month, first we need to move the month into register AX then call subprogram call putDec to print the value (whatever value currently in register AX).

As we can see, the Assembly language can directly accesses and manipulates the registers inside the CPU (Central Processor Unit, the heart of the computer). Not only the logic of a program, the Assembly programmers must also know how to use these registers, to store data and where the result is stored (register AX) after the calculation. This is an art in programming.

3.1. Special Features

The Assembly can work at the “bit-level” to manipulate data, one set of operations is grouped into Logical instruction, the other set is Shift instructions that can be used instead of multiply or divide an unsigned number by an integer multiple of two (2).

3.1.1. Logical Instructions

Logical data can assume only “true or false” value, a single bit can represent these values, typically, value zero (0) represents false and one (1) is true. There are five logical instructions: and, or, not, xor (exclusive or), and test. The following examples illustrate the use of: and, or, and xor. - The and instruction. The most frequently use of and instruction is to clear bits. The example below shows how to clear the three least significant bits (right most).

```
            AL = 11010101 ← operand to be manipulated
            BL = 11111000 ← mask byte
    and AL, BL = 11010000 (The three right most bits are set to 0)
```

After the operation, AL = 11010000

- The or instruction. The or is used to set one or more bits to one (1).

```
            AL = 11010000 <- result from previous example
            BL = 00000111 <- mask byte
     or Al, BL = 11010111 (the last three bits are set to 1)
```

After the operation, AL = 11010111

- The xor instruction. Can be used to initialize registers to zero, the following two instructions are equivalence.

```
mov AX, 0
xor AX, AX
```

Both operants are the same, therefore content of register AX is set to zero (0)

3.1.2. Shift Instructions

The shift operations can be useful in: (1) bits manipulations and (2) to multiply or divide an unsigned number by a power of 2.

- Bits manipulation. Moving one or more bits to left or right positions.

```
mov   AH,   AL
shl   AL,   4      ; move lower nibble to upper
shr   AH,   4      ; move upper nibble to lower
or    AL,   AH     ; paste them together
```

The example above shows how to exchange the contents of lower and upper nibbles. This technique is an algorithm for data encryption.

- Multiplication and division an unsigned number by a power of 2. We know, in binary, position of a bit is a power of 2, therefore shifting to the left has the effect as a multiplication, and shift to the right is equivalence to a division of power of 2.

```
            ; multiplicand is in AX
            mov    CX, 32       ; multiplier is in CX (25)
            mul    CX
            ; or only one shift instruction
            sal    AX, 5        ; shift arithmetic by 5 position [4]
```

The same for shift to the right: (`sar AX, 5`) will restore value in AX.

4. Assembly versus High Level Languages

Assembly is a "One on one" corresponding with machine language, while a high-level language statement may generate many machine language instructions. It is classified as a low-level language that works close to the computer hardware component such as the Central Processor Unit (CPU), and the memory system. On the other hand, the Assembly language unlocks the secret hardware, and provides answers to some questions that might be difficult to other high level language programmers. Despite the fact that writing program in high-level language such as C++, or Java is much easier and more productive, the Assembly still has some advantages:

- More control in handling particular hardware requirements.
- Very small compiler, requires less memory to run.
- Often results in faster execution [5].

On the other hand, the Assembly has some disadvantages in comparing with other high-level languages:

- It is hard to learn. This is obvious when you learn a high-level language such as Pascal.
- It is hard to read and understand. Yes, Assembly statements are brief and conformed to a strictly format.
- It is hard to debug. The Assembly programming logic is more complicated.
- It is hard to maintain.
- It is not portable. Yes, the Assembly was built on a particular hardware (CPU) [6]

Conclusion

To be an expert on the field of computer programming, a programmer should also know the hardware system, understand how a computer executes instructions. The answers for these questions stated in "Problem" paragraph. Only the Assembly programmers can claim that they are truly the master in programming and programming is an art. But, it is sadly to know that many colleges and universities are dropping the Assembly language from their computer science curricula. The conclusion for this essay is left for the readers. Should Assembly language be instated back as a core course in computer science curriculum?

References

[1]. Riley, David D., (2003). The Object of JAVA, Blue Jay Edition. Addition Wesley, Pearson Education, Inc. Pp: xxv.

[2]. Cashman, T. J., & Sheily, G. B. (1971). Introduction to computer programming IBM system/360 assembler language. Pp: 3.2.

[3]. William Jones, (2005). Assembly Language for the IBM PC Family (3e). Addition Wesley. ISBN10: 1-57676-058-8. Pp: 60.

[4]. Sivarama P. Dandamudi (2005). Introduction to Assembly Language Programming. Springer. ISBN 0-387-20636-1. Pp: 272-283.

[5]. Abel, Peter, (1998). IBM PC Assembly language and Programming (4e). Prentice Hall. Upper Saddle River, NJ. 07458. Pp: 50.

[6]. Randall Hyde, (2010). The Art of Assembly Language (2e). No Starch Press. ISBN: 978-1-59327-4. Pp: 2.