A New Role of Assembly Language in Computer Engineering/Science Curriculum

Afsaneh Minaie
Assistant Professor
minaieaf@uvsc.edu

Reza Sanati-Mehrizy
Associate Professor
sanatire@uvsc.edu

Computing and Networking Sciences Department
Utah Valley State College

Abstract:

A separate assembly language course in computer engineering/science curriculum is not required by curriculum guidelines anymore[4]. This is because assembly language programmer is not needed in industry and the curriculum does not afford to include a separate course for assembly language programming. However, it is essential for students to be exposed to assembly language to understand the different concepts in computer engineering/science.

In our introductory computer architecture and assembly language course, we are teaching assembly language using 8086 architecture and Turbo Assembler's Ideal mode for about seven weeks in order to introduce the basic concepts of computer architecture and organization. The students will benefit from knowledge of assembly language programming early in the curriculum not only for better understanding of computer organization and architecture, but it will help them with the concepts such as data representation, instruction interpretation, compiler design, system programming, cost of language abstractions and hardware/software tradeoffs. In this paper, we elaborate the detail content of our introductory computer architecture & assembly language course and the teaching strategies and analyze its outcome.

Introduction

Computer engineering and computer science fields are expanding in all directions. All the subject areas have grown and new subject areas have been added. Since, there are a limited number of courses that can be included in a curriculum model; some of the existing courses will have to be dropped to introduce new ones. As software applications become more complex, more industries use high level languages. The lack of need in industry makes assembly language programming to be a good candidate for elimination from the curriculum. The newer curriculum standards[4] now

recommend the diminution of traditional assembly language programming to make way for a glut of new curriculum topics such as software engineering, object oriented programming, security, computer graphics, and the World Wide Web[1]. This paper argues that assembly language is a vital component of computer engineering/science; however, its role in
the traditional curriculum should be evaluated. The assembly language can be used as a tool for better understanding computer architecture and to prepare students for abstract courses to come. The intention of teaching assembly language programming is not to make students experts in assembly language programming, however; to use it to understand abstract materials.

The Case for Assembly Language

Assembly language concepts are fundamental for the understanding of many areas of computer engineering/science. During a student's career, he or she will encounter lots of abstract concepts in subjects ranging from programming languages, to operating systems, to real time programming, to artificial intelligence, to computer interfacing and to compiler design. The foundation of many of these concepts lies in assembly language programming and computer architecture. One might say that assembly language provides bottom-up support for the top-down methodology we teach in high-level languages[6].

Some of the topics which can be explained further using assembly language concepts are: data representation, computer organization, instruction interpretation and encoding, compiler construction, system programming, overhead of data structures, overhead of parameter passing in procedure abstraction, space/time tradeoffs, hardware/software tradeoffs, input/output programming, interrupts, and etc.[7]. Krishnaprasad[7] claims that the role of assembly language course is similar to that of the discrete structures course in computer science curriculum.

Teaching assembly language is going to help students learn the abstract computer engineering/science subjects easier. However, due to the lack of need in the industry for assembly programmers and the fact that the computer engineering/science is expanding every day, it will not be affordable to offer a separate course for teaching assembly language. It will be feasible and essential to teach assembly for few weeks in the introductory computer architecture class because it is a viable tool for understanding computer architecture.

Assembly Language and Computer Architecture at Utah Valley State College (UVSC)

Students at UVSC majoring in computer science or computer engineering take an introductory assembly language and computer architecture course after completing their first object oriented course in C++. The goal of this course is to use assembly language as a tool for better understanding computer architecture and to prepare them for abstract courses to come. The intention is not to make students experts in assembly language programming. The introductory course that we teach uses two text books. The primary text is Computer Organization and Architecture by William Stallings[7]. The second text is Mastering Turbo Assembler by Tom Swan[8]. The software package used is Borland's Turbo Assembler for the PC.

We start teaching the course by introducing the number systems and data representation and integer arithmetic. Then, we introduce assembly language basics. Our students learn the basic organization and internal functioning of 8086 microprocessor by using Turbo Assembler. As students use the Turbo Debugger to step through an assembly language program, they will observe the various changes in the registers and status flags. Important teaching points can be emphasized by having them step through an assembly language program while observing the various changes in the registers and status flags. Then, we move to the computer architecture part of the course and teach the structure and function of computers from a top down view. As the need comes for more assembly material, in order for students to do their programming assignments or to understand a computer architecture concept; we move back to the assembly part of the course. The organization of the course is that we mix assembly language concepts and computer architecture materials. Following is an outline for the course that we teach:

Course Outline:

- Number Systems
- Data Representation and Integer Arithmetic
- Introduction to Assembly Language
- 8086 architecture
- 8068 Assembly Language Features
- The History of Computer Architecture Technology
- Addressing Modes for 8086
- Stack
- Digital Logic
- A top-level View of Computer Function and Interconnection
- Subroutines and parameter passing
- Cache Memory
- Logical Instructions
- Internal Memory
- Finite State Machines
- Input / Output Techniques
- Interrupts and Interrupt handling
- Instruction Sets: Characteristics and Functions
- Instruction Sets: Addressing Modes and Formats
- CPU Structure and Function

At UVSC, we have six open computer labs which are open 7:00 a.m. – 10:00 p.m. This class is not held in the lab. However, in the beginning of the semester, we take students to the lab. The purpose of taking students to the lab is for them to become familiar with the turbo assembler environment and learn how to use the debugger. In the lab, the students will type in an assembly program and assemble and link it and use the debugger to step through the program. There is a tutor for the course that sits in the lab to help students with their programming assignments. The tutor-student relationship benefits the tutor as well. Tutor solidifies his/her knowledge while helping their peers. Laboratory programming assignments are given electronically. Assignments are submitted and graded electronically. Following a sample project is given:

Sample Laboratory Project

In this assignment, students will write an interrupt service routine to replace the existing ISR for Control - Break. Normally, the <ctrl-C> key press will terminate the currently running DOS program. Their assignment is to write a new ISR for the <ctrl-C> key. Their program should declare a global variable in their main program data segment where they will keep a count of the number of times the <ctrl-C> key is pressed. Each time <ctrl-C> is pressed their interrupt service routine will get the contents of this variable and add one, then return control to their main program. Their ISR must save the registers that it uses. The interrupt number for Control - Break is 23h, which is a software interrupt.

In their main routine, their program will first save the default interrupt vector then load the interrupt vector to their interrupt service routine. Once the interrupt is set up, they will display the following message:

Enter a character from the keyboard:

Their program will enter a loop, inspects to see if it is a printable character then prints out the present character input with a space character. It will continue printing out the same character until a different printable character is inputted. If a non-printable character is inputted, stops printing and waits until a printable character is inputted then continues printing the new character. This will continue until an <!> character is entered. Their screen output should look something like this:

Enter a character from the keyboard: a
a a a a a a a a a a a a a a a a a a ^c a a a a a a a a a a a a
a a a a a a b b b b b b b b b b b b b b b b b ^c b b b b b b b
b b b b b b b b b x x x x x x x x !

<ctrl-C> was pressed 02 times

The characters input in this example were in this order:
<a> <^c> <b> <^c> <^z> <x> <!>

When a <ctrl-C> is pressed their ISR will be called automatically, and the count updated.

When an <!> is pressed, their program should exit the loop and print out the number of times <ctrl-C> was pressed. This count is a byte count and should print out at least 2 ASCII digits in a message something like

<ctrl-C> was pressed 13 times

Their program should then restore the original interrupt that they saved and then exit to DOS.

A student's implementation of the assignment follows:

```
%TITLE "CNS 1380 Programming Project #7"
;--------------------------------------------------------
```

```
;           Replace Interrupt Routines for ctrl-c with my own.
;           Display prompt, accept input from keyboard.
;-------------------------------------------------------
                IDEAL                       ; Turbo Assembler mode
                MODELsmall                  ; Program model
                STACK 100h                  ; Stack size
;-------------------------------------------------------
DOS             EQU     21h                 ; DOS interrupt code
KEY             EQU     01h                 ; KEY reads keyboard
WRITE           EQU     02h                 ; WRITE for video display
CR              EQU     13                  ; ASCII number for carriage return
LF              EQU     10                  ; ASCII for line feed
STR_WRITE       EQU     09h                 ; Used to write out a string
ENTER_KEY       EQU     0Dh                 ; value for Enter Key
CTRLC  EQU      23h                         ; value for CTRL-C
;-------------------------------------------------------
                DATASEG

exCode          DW      04C00h              ; return code
msg1            DB      'Enter a character from the keyboard: $'
quitmsg         DB      '<ctrl-c> was pressed $'
quitmsg2        DB      ' times$'           ; quit message
charIn          DB      ?                   ; used to save input character
vStore1         DW      ?                   ; used to store bx of original vector
vStore2         DW      ?                   ; used to store es of original vector
ctrlcount1      DB      '0'                 ; used to count how many times ctrl-c was pressed
ctrlcount2      DB      '0'                 ; used to count how many times ctrl-c was pressed
keyFlag  DB     0FFh                        ; value returned if key has been pressed

;-------------------------------------------------------
                CODESEG

Main:           mov     ax, @data           ; get ds
                mov     ds, ax              ; initialize ds
;-------------------------------------------------------
;               Used to save interrupt vector
;
                mov     ah, 35h             ; get interrupt vector
                mov     al, CTRLC           ; for INT 23 for CTRL-C
                int     21h                 ; call DOS
                mov     [vStore1], bx       ; Store the offset
                mov     [vStore2], es       ; Store the segment

;-------------------------------------------------------
;               Used to set new interrupt vector

                push    ds                  ; save ds
                push    cs                  ; store cs
                pop     ds                  ; put cs in ds
                mov     dx, OFFSET ISRoutine
                mov     ah, 25h             ; Set interrupt vector
                mov     al, CTRLC           ; for INT 23 for CTRL-C
                int     DOS                 ; call DOS
                pop     ds                  ; restore ds
```

```
                mov     dx, OFFSET msg1          ; prepares msg1 to be displayed
                call    PutStr                   ; calls subroutine to display string
                jmp     Start
;-----------------------------------------------------------
;               New interrupt code
ISRoutine:
                cmp     [ctrlcount1], '9'
                je      Count10s                 ; jumps to increment 10's spot
                inc     [ctrlcount1]             ; increments ctrlcount1 by 1
                iret
Count10s:       mov     [ctrlcount1], '0'        ; resets ctrlcount1 back to 0
                inc     [ctrlcount2]             ; increments ctrlcount2 by 1
                iret
;-----------------------------------------------------------
;       Displays prompt, waits for character input
;       Goes to first of new line, outputs character that was input
Start:
                call    GetChar                  ; calls routine to accept input character
                pop     bp                       ; pops value for character input from stack
                mov     ax, bp                   ; moves content of bp to ax
                mov     [charIn], al             ; copies input character to memory
                cmp     [charIn], '!'            ; if ! is entered then exit
                je      exitIn                   ; jumps to exitIn
                mov     dl, " "                  ; used for spacer
                call    PutChar                  ; puts spacer on screen
                call    Ischar                   ; checks to see if al is a printable character
                jz      Loopy                    ; jumps to Loopy
                jmp     Start                    ; jumps back to Start if an invalid key was pressed
Loopy:
                mov     ah, 0Bh                  ; used to see if there has been a key press
                int     DOS                      ; call DOS
                cmp     al, [keyFlag]            ; sees if key was pressed
                je      Start                    ; if key was pressed goes back to start
                mov     dl, [charIn]             ; used for PutChar when displaying prompt
                call    PutChar                  ; calls routine to display character
                mov     dl, " "                  ; used for PutChar when displaying prompt
                call    PutChar                  ; calls routine to display character
                jmp     Loopy
exitIn:                                          ; used to exit when ! is entered
                call    NewLine
                mov     dx, OFFSET quitmsg       ; used for first part of exit message
                call    PutStr
                mov     dl, [ctrlcount2]         ; displays 10's location of number
                call    PutChar
                mov     dl, [ctrlcount1]         ; displays 0-9 for times ctrl-c pressed
                call    PutChar
                mov     dx, OFFSET quitmsg2      ; end of exit message
                call    PutStr
                push    dx                       ; pushes current dx register
                push    ds                       ; pushes current ds register
                mov     dx, [vStore1]            ; used to restore original vector
                mov     ds, [vStore2]            ; used to restore original vector
                mov     ah, 25h                  ; Set interrupt vector
```

```
        mov     al, CTRLC           ; for INT 23
        int     DOS                 ; call DOS
        pop     ds                  ; pop ds from stack
        jmp     Exit                ; exits

Exit:   mov     ax, [exCode]        ; return code
        int     DOS                 ; function call to DOS

PutStr: mov     ah, STR_WRITE
        int     DOS
        ret
GetChar:                            ; used to input character from keyboard
        push    bp                  ; used to store future input character
        push    bp                  ; used to store updated pointer
        push    ax                  ; backs up old ax
        mov     bp, sp              ; movs sp to bp
        mov     ax, [bp+6]          ; movs old bp into ax
        mov     [bp+4], ax          ; mov ax into new bp stack location
        mov     ah, KEY                     ; dos command to key input
        int     DOS                 ; calls dos
        mov     [bp+6], ax          ; saves character input onto stack
        pop     ax                  ; pops old ax
        pop     bp                  ; pops updated bp
        ret
PutChar:                                    ; used to move to new line and display character
        push    bp                  ; stores old bp
        push    dx                  ; stores old dx
        mov     bp, sp              ; mov sp to bp
        mov     dx, [bp]            ; moves the old dx info into dx reg
        mov     ah,WRITE            ; dos call to write character
        int     DOS                 ; calls dos
        pop     dx                  ; pops old dx content
        pop     bp                  ; pops old bp content
        ret
NewLine: mov    dl, CR              ; used for carriage return
        mov     ah, WRITE           ; dos call to write character
        int     DOS                 ; calls dos
        mov     dl, LF              ; used for line feed
        mov     ah, WRITE           ; dos call to write character
        int     DOS                 ; calls dos
        ret
Ischar: cmp     [charIn], '!'       ;used to see if character value is ASCII
        jb      A1
        cmp     [charIn], '~'
        ja      A1
        test    ax,0
A1:     ret
        END     Main                ; end of program
```

The course material is available at ftp://cseftp.uvsc.edu/csn/minaieaf/CNS%201380

Conclusion:

Assembly language concepts are fundamental to understanding many fields of computer engineering/science. Since we can not afford to offer a stand alone course in assembly language, in our introductory course in computer architecture, we teach assembly language for less than half of the semester. By teaching assembly language in that course, our students can understand the computer architecture concepts better and it will also prepare them for abstract courses to come.

Bibliography:

[1] Bredlau, Carl and Deremer, Dorothy, Assembly Language through the Java Virtual Machine, Proceedings of the thirty second SIGCSE technical symposium on Computer Science Education, 2001, pp. 194–198.

[2] Tanenbaum, A. Structured Computer Organization, 1999. Prentice-Hall, pp. 483-488.

[3] Computing Curricula 1991, Report of the CM/IEEE-CS Joint Curriculum Task Force (1991) Available WWW. http://www.computer.org/education/cc1991/.

[4] Computing Curricula 2001, Report of the Joint Task Force on Computing Curricula (2000). Available WWW. http://www.computer.org/education/cc2001/report/index.html .

[5] Than, Soe, Development and Use of an Assembler in Computer Systems Course, JCSC, May 2001, pp 145-152.

[6] The Case for and Against Assembly Language, http://wheelie.tees.ac.uk/users/a.clements/CaseFor.htm

[7] Stallings, William, Computer Organization & Architecture, 6th edition, Prentice Hall, 2003. ISBN # 0-13-035119-9

[8] Swan, Tom, Mastering Turbo Assembler, 2nd edition, Sams publishing, 1995.

AFSANEH MINAIE is an assistant professor in the Computing and Networking Sciences Department at Utah Valley State College. She received a B.S., M.S. and Ph.D. all in Electrical Engineering from University of Oklahoma in 1981, 1984 and 1989 respectively. Her current interests are in computer architecture, digital design, and computer interfacing.

REZA SANATI MEHRIZY is an associate professor of the Computing and Networking Sciences Dept. at Utah Valley State College, Orem, Utah. He received his MS and PhD in Computer Science from University of Oklahoma, Norman, Oklahoma. His research focuses on diverse areas such as: Database Design, Data Structures, Artificial Intelligence, Robotics, and Computer Integrated Manufacturing.