

THE VIRTUAL PC

A TOOL FOR STUDYING HARDWARE AND SOFTWARE

Pierre A. von Kaenel
Skidmore College
Saratoga Springs, NY 12866
pvonk@skidmore.edu

INTRODUCTION

This paper introduces the Virtual PC (or VPC), a Windows 95 application that simulates a personal digital assistant based on Intel's 8086 architecture, and several supporting applications. The purpose of the VPC is to provide students with a realistic model of a complete hardware and software system. This system is used to study the different levels of a computer as presented in a computer organization course. Instead of just reading about a hypothetical machine such as the Mic-1 and Mac-1 described by Tanenbaum [3], students can write and execute microprograms, study the machine program level, and write and compile assembly programs that are interpreted by a machine level interpreter written in microcode. Since the simulator provides a complete system, students gain a better understanding of how a PC works.

The applications that comprise the simulator are the VPC (the hardware simulator that includes debugging capabilities), MicroASM (compiles micro assembly code into a binary file that is then loaded into the VPC), 8086 (a microprogram that interprets machine code), and VASM (an assembler that supports a subset of the Intel 8086 assembly language). The two assemblers provide complete programming environments by incorporating an editor, help files, and compiler. In addition reference manuals for each of these components have been written. To obtain a copy of the simulator's files, contact the author via email.

THE VPC'S HARDWARE

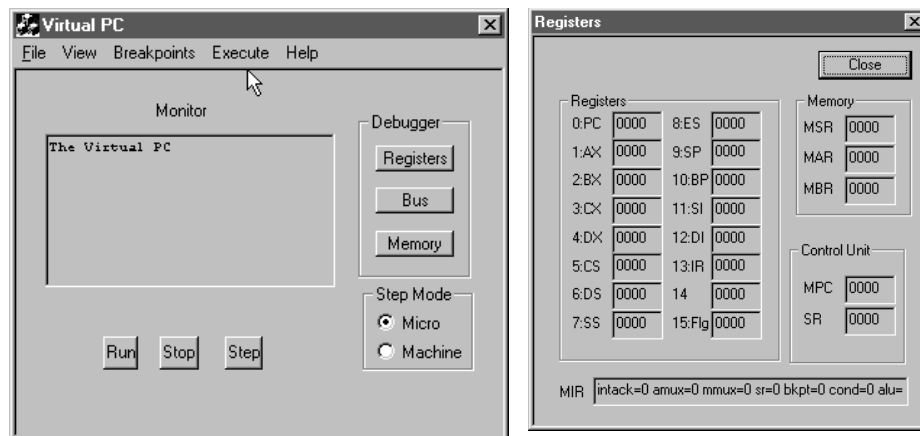
The VPC evolved from the digital watch [4], a simple DOS-based microprogram simulator that supported a limited assembly language. Unfortunately, the watch's organization did not support a rich microprogram language and did not closely model the Intel memory architecture. The result was a simulator of a fictitious computer system that bore a minimal resemblance to an Intel-based PC. Since Skidmore had decided to eliminate the assembly programming course and introduce assembly in the computer organization course, the author decided to redesign the simulator from scratch so that it would conform more to the Intel architecture and support a subset of the Intel assembly language.

The VPC consists of a microprocessor based on the Intel 8086, 256K RAM, video circuit, interrupt controller, keyboard controller, and a system clock/timer. The heart of the microprocessor is the control unit where the microcode is stored and

executed. Similar to Tanenbaum's Mic-Mac processor in its general layout, the VPC's microprocessor (VCP) is designed to operate more like the 8086 [2]. It uses 16-bit registers with a 16-bit data bus. Memory access is also similar to the Intel chip in that writing a word to memory can only be made to an even address [1]. Writing two bytes starting at an odd address involves two separate writes of one byte each. Other similarities are the segmented architecture, hardware interrupts, and use of ports. One very different aspect is the ALU. Unlike the Intel chip, the ALU can perform all supported operations (logical operators and arithmetic operators including multiplication and division) in one clock cycle. Clearly some compromises had to be made to avoid making the VPC too complex to program.

The similarities to the Intel architecture make the VPC an ideal lab tool. Students gain a better understanding of the internal workings of a computer system when they can do some hands-on exploring with a working model. The simulator's debugging features allow the user to step through a micro or machine program and observe current register, memory, and bus values. Many of the mysteries that students encounter in an assembly programming course are unveiled when they study the VPC. For example, once students get down to the nuts and bolts of the hardware, they gain a clearer understanding of the use of interrupts and interrupt priorities.

The VPC simulator appears on screen as a window containing a small video monitor (displaying the VPC output), and a set of buttons for running and debugging programs. The VPC's window includes a menu that provides for setting breakpoints, performing a reset, and loading microcode and machine programs into the simulator. In addition, other pop up windows can be invoked to peek at the registers, memory, and the system bus. The user's keyboard is used for input when the main window is in focus. The simulator's keyboard controller receives the input and initiates a hardware interrupt. It's up to the microprogram to decide what, if anything, should be done with the key.

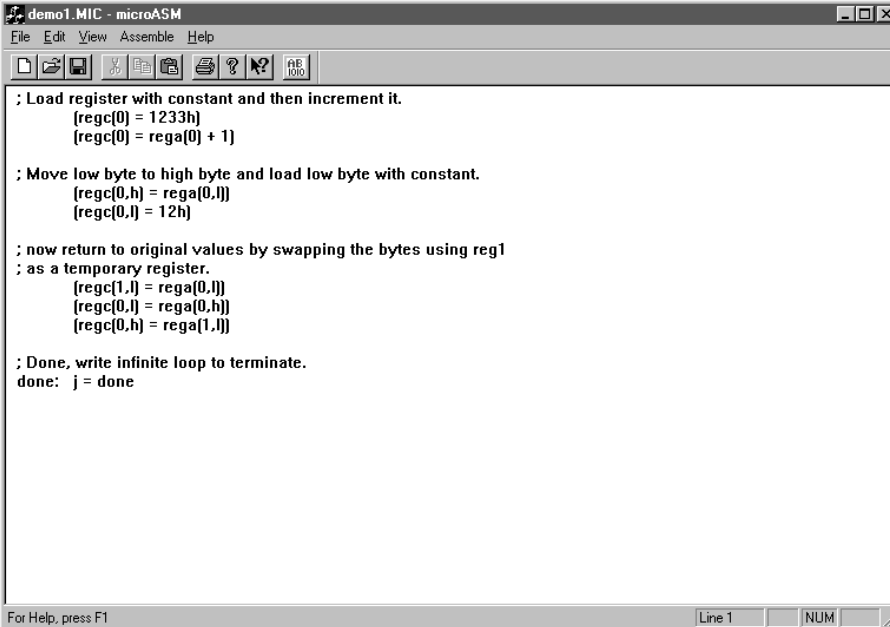


Views of the VPC and Registers windows.

THE MICROPROGRAM LEVEL

The Virtual PC is controlled by a microprogram stored in the VCPU. Each instruction, consisting of 82 bits divided into 31 fields, controls the flow of data in the microprocessor during one clock cycle. To simplify the process of writing a microprogram, a micro assembly language and a micro assembler have been created. The micro assembler (microASM) runs on a PC and provides an integrated programming environment that includes an editor and help system. The compiled code can then be synced to the VPC and executed.

Students can write microprograms that perform specific functions such as display and update a clock, or they can define a machine language and write an interpreter for that language. For example, in one course students created a simple language used to write game applications – turning the VPC into a hand-held game machine. However, writing an interpreter can be a large project when the language isn't small. Such an undertaking is better suited as an independent study.

A screenshot of a software window titled "demo1.MIC - microASM". The window has a menu bar with "File", "Edit", "View", "Assemble", and "Help". Below the menu bar is a toolbar with icons for file operations, editing, and assembly. The main text area contains assembly code with comments. The code includes instructions for loading a register with a constant and incrementing it, moving bytes between registers, and swapping bytes. At the bottom of the window, there is a status bar with the text "For Help, press F1" on the left and "Line 1" and "NUM" on the right.

```
; Load register with constant and then increment it.  
[regc(0) = 1233h]  
[regc(0) = rega(0) + 1]  
  
; Move low byte to high byte and load low byte with constant.  
[regc(0,h) = rega(0,l)]  
[regc(0,l) = 12h]  
  
; now return to original values by swapping the bytes using reg1  
; as a temporary register.  
[regc(1,l) = rega(0,l)]  
[regc(0,l) = rega(0,h)]  
[regc(0,h) = rega(1,l)]  
  
; Done, write infinite loop to terminate.  
done: j = done
```

View of MicroASM's programming environment

THE MACHINE AND ASSEMBLY LEVELS

The author has defined a machine language that supports most of the basic operations found in typical machine languages. This language, consisting of 134 opcodes, is introduced in the computer organization course as a prelude to the module on assembly programming. An interpreter was written using MicroASM, however, aside from a brief discussion of how the interpreter is organized, the actual source code is not studied in the course. Using the interpreter, it is possible to use the VPC as a platform for studying just assembly programming.

The assembly language supports a subset of the Intel 8086 assembly language and includes several extensions that are needed to work with the VPC. Assembly programs are compiled into object code that can then be synced into the VPC, much like programs are synced from a PC to a PDA like the PalmPilot. VASM is also a Windows 95 application that incorporates an editor, help system, and compiler. While the microcode is stored inside the VCPU, a VASM object file is loaded into the VPC's RAM. Both MicroASM and VASM generate listing files that contain source code, assembled code along with addresses, and error statements. Since there is no separate operating system, an assembly program must include such data as the interrupt table. The following do-nothing program serves as a template for assembly programs. The first segment (system) is required for defining the system data.

```

; template.asm - a bare bones .ASM file
; ===== system segment =====
.system
    ; the interrupt table-
    dw 8 dup(0)           ; 0-F       : Int 0-3: (unused)
    dw seg clk           ; 10-11     : Int 4:  clock segment
    dw offset clk       ; 12-13     :          clock offset
    dw seg kbd          ; 14-15     : Int 5:  kbd segment
    dw offset kbd       ; 16-17     :          kbd offset
    dw 4 dup(0)         ; 18-1F     : unused
    dw seg main         ; 20-21     : main proc segment
    dw offset main      ; 22-23     : main proc offset
    dw 2 dup(0)         ; 24-27     : unused

; ===== data segment =====
.data    dataseg
; ===== code segment =====
.code    codeseg
;----- proc : clk -----
proc clk           ; if interrupts are not activated,
    iret           ; the clk and kbd procs can be
endp              ; deleted from this segment.
;----- proc : kbd -----
proc kbd
    iret
endp
;----- proc : main -----
proc main          ; The operating system main loop.

    sti           ; delete this line to deactivate interrupts.
    mov ds, seg dataseg ;delete if no data segment is used.
    ; PLACE YOUR PROGRAM HERE
done:
    jmp done
endp

```

The following code demonstrates some of language's commands:

```
proc main
    mov ds, seg dataseg
    mov es, seg dataseg
    ; Rotate array ar1 into ar2.
    mov cx, 4 ; counter.
    mov si, offset ar1 ; ar1 and ar2 are defined in
    mov di, offset ar2 ; the data segment.
    inc di
loop:
    mov al, [si]
    mov [di], al
    inc si
    inc di
    dec cx
    jcxz finishup
    jmp loop
finishup:
    mov al, [si]
    mov ar2, al
done: jmp done
endp
```

OPERATING SYSTEMS

Because of VASM's richness and the VPC's simple architecture, students have the ability to write simple operating system components. For example, a typical assignment is to write a keyboard interrupt handler in which incoming keys are stored in a small buffer. In addition a software interrupt can be written to provide I/O functions similar to Intel's interrupt 21h (the DOS functions). Typically interrupt 6 or 7 is used on the VPC since these entries in the interrupt table are unused. For an advanced assignment, a task manager can be written to provide multiprocessing for two or more simple applications, one of them being the command processor. The VPC provides endless possibilities for simple yet significant projects at the microprogram, assembly, and operating system levels.

REFERENCES

- [1] Holt, Charles A., Microcomputer Systems, Macmillan Publishing, 1986.
- [2] Morse, Stephen P., The 8086/8088 Primer, 2nd Edition, Hayden Books, 1982.
- [3] Tanenbaum, Andrew S., Structured Computer Organization, 3rd Edition, Prentice-Hall, 1990.
- [4] von Kaenel, Pierre A. "Microprogramming a Watch: Tools for a Course in Computer Organization", ACM SIGCSE Bulletin, Feb. 1988 (vol. 20, no. 1), pp. 309.