Answers to Review Questions

Prepared by the author. Recent revision history:

- 9/13/2002: 1.3 #5(b), 1.3 #6(a), 1.3 #10(c), 1.3 #21(c), 1.4 #2
- 9/16/2002: 1.3 #19(a, b), 1.3 #20(a, b)
- 9/25/2002: 3.4 #1, 4.1 #8(a), 4.1 #10(d), 4.2 #4, 6.2 #1(a), 4.2 #10, 6.2 #2(a,b,c,d)
- 9/30/2002: 2.2 #14, 4.3 #6, 5.3 #6, 5.3 #10, 5.4 #1, 5.4 #8, 7.2 #11, 7.2 #12

1 Basic Concepts

1.1 Welcome to Assembly Language

1. An assembler is a program that converts source-code programs from assembly language into machine language. A linker combines individual files created by an assembler into a single executable program.

2. Assembly language is a good tool for learning how application programs communicate with the computer's operating system, via interrupt handlers, system calls, and common memory areas. Assembly language programming also helps when learning how the operating system loads and executes application programs.

3. In a one-to-many relationship, a single statement expands into multiple assembly language or machine instructions.

4. A language whose source programs can be compiled and run on a wide variety of computer systems are said to be portable.

5. No. Each assembly language is based on either a processor family or specific computer.

6. Some examples of embedded systems applications are automobile fuel and ignition systems, air-conditioning control systems, security systems, flight control systems, hand-held computers, modems, printers, and other intelligent computer peripherals.

7. Device drivers are programs that translate general operating system commands into specific references to hardware details that only the manufacturer would know.
8. C++ does not allow a pointer of one type to be assigned to a pointer of another type. Assembly language has no such restriction regarding pointers.

9. Applications suited to assembly language: Hardware device driver, and embedded systems and computer games requiring direct hardware access.

10. A high-level language may not provide for direct hardware access. Even if it does, awkward coding techniques must often be used, resulting in possible maintenance problems.

11. Assembly language has minimal formal structure, so structure must be imposed by programmers who have varying levels of experience. This leads to difficulties maintaining existing code.

12. Code for the expression: \( X = (Y \times 4) + 3 \):

\[
\begin{align*}
\text{mov} & \quad \text{eax}, Y & \quad \text{; move } Y \text{ to } \text{EAX} \\
\text{mov} & \quad \text{ebx}, 4 & \quad \text{; move } 4 \text{ to } \text{EBX} \\
\text{imul} & \quad \text{ebx} & \quad \text{; } \text{EAX} = \text{EAX} \times \text{EBX} \\
\text{add} & \quad \text{eax}, 3 & \quad \text{; add } 3 \text{ to } \text{EAX} \\
\text{mov} & \quad X, \text{eax} & \quad \text{; move } \text{EAX} \text{ to } X
\end{align*}
\]

1.2 Virtual Machine Concept

1. Computers are constructed in layers, so that each layer represents a translation layer from a higher-level instruction set to a lower level instruction set.

2. It is enormously detailed and consists purely of numbers. Hard for humans to understand.

3. True

4. An entire L1 program is converted into an L0 program, by an L0 program specifically designed for this purpose. Then the resulting L0 program is executed directly on the computer hardware.

5. The IA-32’s virtual-86 operating mode emulates the architecture of the Intel 8086/8088 processor, used in the original IBM Personal Computer.

6. Java byte code is a low-level language that is quickly executed at run time by a program known as a Java virtual machine (JVM).

7. Digital logic, microarchitecture, instruction set architecture, operating system, assembly language, high-level language.

8. The specific microarchitecture commands are often a proprietary secret. Also, microcode programming is impractical because it often requires 3 or 4 microinstructions to carry out a single primitive operation.

9. Instruction-set architecture
10. Levels 2 and 3.

1.3 Data Representation

1. Least significant bit (bit 0).
2. Most significant bit (the highest numbered bit).
3. (a) 248  (b) 202  (c) 240
4. (a) 53  (b) 150  (c) 204
5. (a) 00010001  (b) 101000000  (c) 00011110
6. (a) 110001010  (b) 110010110  (c) 100100001
7. (a) 2  (b) 4  (c) 8
8. (a) 16  (b) 32  (c) 64
9. (a) 7  (b) 9  (c) 16
10. (a) 12  (b) 16  (c) 22
11. (a) CF57  (b) 5CAD  (c) 93EB
12. (a) 35DA  (b) CEA3  (c) FEDB
13. (a) 1110 0101 1011 0110 1010 1110 1101 0111
   (b) 1011 0110 1001 0111 1100 0111 1010 0001
   (c) 0010 0011 0100 1011 0110 1101 1001 0010
14. (a) 0000 0001 0010 0110 1111 1001 1101 0100
   (b) 0110 1010 1100 1101 1111 1010 1001 0101
   (c) 1111 0110 1001 1011 1101 1100 0010 1010
15. (a) 58  (b) 447  (c) 16534
16. (a) 98  (b) 457  (c) 27227
17. (a) FFE6  (b) FE3C
18. (a) FFE0  (b) FFC2
19. (a) +31915  (b) −16093
20. (a) +32667  (b) −32208
21. (a) −75  (b) +42  (c) −16
22. (a) $-128$  (b) $-52$  (c) $-73$

23. (a) $11111011$  (b) $11011100$  (c) $11110000$

24. (a) $10111000$  (b) $10011110$  (c) $11100110$

25. $58h$ and $88d$

26. $4Dh$ and $77d$

27. To handle international character sets that require more than 256 codes.

28. $2^{256} - 1$

29. $+2^{255} - 1$

### 1.4 Boolean Operations

1. (NOT X) OR Y

2. X AND Y

3. T

4. F

5. T

6. Truth table:

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>$A \lor B$</th>
<th>$\neg(A \lor B)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
<td>F</td>
</tr>
</tbody>
</table>

7. Truth table:

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>$\neg A$</th>
<th>$\neg B$</th>
<th>$\neg A \land \neg B$</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
</tbody>
</table>
8. 16, or \(2^4\)

9. 2 bits, producing the following values: 00, 01, 10, 11

2 IA-32 Processor Architecture

2.1 General Concepts

1. Control Unit, Arithmetic Logic Unit, and the clock.

2. Data, Address, and Control buses.

3. Conventional memory is outside the CPU, and it responds more slowly to access requests. Registers are hard-wired inside the CPU.

4. Fetch, decode, execute.

5. Fetch memory operands, store memory operands

6. During the fetch step

7. Executing processor stages in parallel, making possible the overlapped execution of machine instructions.

8. 10 clock cycles

9. 12 cycles \((5 + (8 – 1))\)

10. A superscalar processor contains two or more execution pipelines

11. 15 clock cycles \((5 + 10)\)

12. Section 2.1.4.1 mentions the file name, file size, starting location on the disk. (Most directories also store the file’s last modification date and time.)

13. The OS executes a branch (like a GOTO) to the first machine instruction in the program.

14. The CPU executes multiple tasks (programs) by rapidly switching from one program to the next. This gives the impression that all programs are executing at the same time.

15. The OS scheduler determines how much time to allot to each task, and it switches between tasks.
16. The program counter, the task’s variables, and the CPU registers (including the status flags).

2.2 IA-32 Processor Architecture

1. Real-address mode, Protected mode, and System Management mode
2. EAX, EBX, ECX, EDX, ESI, EDI, ESP, EBP
3. CS, DS, SS, ES, FS, GS
4. Loop counter
5. EBP
7. Carry
8. Overflow
9. Sign
10. Floating-Point Unit
11. 80 bits
12. The Intel 80386
13. The Pentium
14. The Pentium (see the Chapter corrections on the book’s Web site)
15. CISC means complex instruction set. A large collection of instructions, some of which perform sophisticated operations that might be typical of a high-level language.
16. The term RISC stands for reduced instruction set. A small set of very simple (atomic) instructions that may be combined into more complex operations.

2.3 IA-32 Memory Management

1. 4 GB (0 to FFFFFFFFh)
2. 1 MB (0 to FFFFh)
3. linear (absolute)
4. 09600h
5. 0CFF0h
6. 32 bits
7. SS register
8. Local descriptor table
9. Global descriptor table
10. The total size of all programs loaded into memory can exceed the amount of physical memory installed in the computer.

11. This is an open-ended question, of course. It is a fact that MS-DOS first had to run on the 8086/8088 processors, which only supported Real-address mode. When later processors came out which supported Protected mode, my guess is that Microsoft wanted MS-DOS to continue to run on the older processors. Otherwise, customers with older computers would refuse to upgrade to new versions of MS-DOS.

12. The following segment-offset addresses point to the same linear address: 0640:0100, and 0630:0200.

2.4 Components of an IA-32 Microcomputer

1. External cache memory is high-speed (static RAM) that is connected to the CPU via a special bus that permits data transfer at higher speeds than conventional memory.

2. Pentium

3. The 8259 is the interrupt controller chip, sometimes called PIC, that schedules hardware interrupts and interrupts the CPU.

4. On either the video board, or on the motherboard (special memory area).

5. A beam of electrons illuminates phosphorus dots on the screen called pixels. Starting at the top of the screen, the gun fires electrons from the left side to the right in a horizontal row, briefly turns off, and returns to the left side of the screen to begin a new row. Horizontal retrace refers to the time period when the gun is off between rows. When the last row is drawn, the gun turns off (called the vertical retrace) and moves to the upper left corner of the screen to start all over.

6. Dynamic RAM, Static RAM, Video RAM, and CMOS RAM.

7. Static RAM

8. The computer can query a device connected via USB to find out its name, device type, and the type of driver it supports. The computer can also suspend power to individual devices. None of these capabilities are possible with serial and parallel ports.

9. Upstream and downstream

10. 16550 UART (universal asynchronous receiver transmitter)
2.5 Input-Output System

1. The application program level

2. BIOS functions communicate directly with the system hardware. They are independent of the operating system.

3. New devices are invented all the time, with capabilities that were often not anticipated when the BIOS was written.

4. The BIOS level.

5. The operating system, BIOS, and hardware levels.

6. Game programs often try to take advantage of the latest features in specialized sound cards. It should be noted that MS-DOS game applications were more prone to do this than games running under MS-Windows. In fact, Windows-NT, 2000 and XP all prevent applications from directly accessing system hardware.

7. No. The same BIOS would work for both operating systems. Many computer owners install two or three operating systems on the same computer. They would certainly not want to change the system BIOS every time they rebooted the computer!

3 Assembly Language Fundamentals

3.1 Basic Elements of Assembly Language

1. h,q,o,d,b,r,t,y

2. No (a leading zero is required)

3. No (they have the same precedence)

4. Expression: 10 MOD 3

5. Real number constant: +3.5E-02

6. No, they can also be enclosed in double quotes

7. directives

8. 247 characters

9. True

10. True

11. False
12. True

13. label, mnemonic, operand(s), comment

14. True

15. True

16. Code example:

    Comment !
    This is a comment
    This is also a comment
    !

17. Because the addresses coded in the instructions would have to be updated whenever new variables were inserted before existing ones.

### 3.2 Example: Adding Three Integers

1. The INCLUDE directive copies necessary definitions and setup information from the *Irvine32.inc* text file. The data from this file is inserted into the data stream read by the assembler.

2. The .CODE directive marks the beginning of the code segment.

3. code, data, and stack.

4. By calling the DumpRegs procedure

5. The exit statement

6. The PROC directive

7. The ENDP directive

8. It marks the last line of the program to be assembled, and the label next to END identifies the program’s entry point (where execution begins).

9. PROTO declares the name of a procedure that is called by the current program.

### 3.3 Assembling, Linking, and Running Programs

1. Object (.OBJ) and listing (.LST) files.

2. True

3. True

4. Loader
5. Executable (.EXE) and map (.MAP).

6. The /Fl option

7. The /Zi option

8. It tells the linker to produce a Win32 Console application.

9. There are two many to mention here, but you can view their names by opening Kernel32.lib using the TextPad editor supplied on the book's CD-ROM. The file will display in hexadecimal. Scroll down to offset 1840h, and look at the various function names listed from that point on.

10. /ENTRY sets the program’s starting address (the entry point). For example, suppose you wanted your program to begining execution at the Startup procedre. The link command line would be:

    link32 /ENTRY:Startup

    This is a challenging question because you cannot find the answer in Appendix A. Instead, you can read about the Microsoft 32-bit linker command-line options by visiting the MSDN Web site and searching for linker reference.

### 3.4 Defining Data

1. var1 SWORD ?

2. var2 BYTE ?

3. var3 SBYTE ?

4. var4 QWORD ?

5. SDWORD

6. var5 SDWORD -2147483648

7. wArray WORD 10,20,30

8. myColor BYTE "blue",0

9. dArray DWORD 50 DUP(?)

10. myTestString BYTE 500 DUP("TEST")

11. bArray BYTE 20 DUP(0)

12. 21h,43h,65h,87h
### 3.5 Symbolic Constants

1. `BACKSPACE = 08h`
2. `SecondsInDay = 24 * 60 * 60`
3. `ArraySize = ($ – myArray)`
4. `ArraySize = ($ – myArray) / TYPE DWORD`
5. `PROCEDURE TEXTEQU <PROC>`
6. Code example:
   ```
   Sample TEXTEQU <"This is a string">
   MyString BYTE Sample
   ```
7. `SetupESI TEXTEQU <mov esi,OFFSET myArray>`

### 4 Data Transfers, Addressing, and Arithmetic

#### 4.1 Data Transfer Instructions

1. Register, immediate, and memory
2. False
3. False
4. True
5. A 32-bit register or memory operand
6. A 16-bit immediate (constant) operand
7. (a) not valid  (b) valid  (c) not valid  (d) not valid  (e) not valid  (f) not valid  (g) valid  (h) not valid
8. (a) FCh  (b) 01h
9. (a) 1000h  (b) 3000h  (c) FFF0h  (d) 4000h
10. (a) 00000001h  (b) 00001000h  (c) 00000002h  (d) FFFFFFFCh

#### 4.2 Addition and Subtraction

1. `inc val2`
2. `sub eax,val3`
3. Code:
    
    ```assembly
    mov ax, val4
    sub val2, ax
    ```

4. CF = 0, SF = 1

5. CF = 1, SF = 1

6. Write down the following flag values:
   
   (a) CF = 1, SF = 0, ZF = 1, OF = 0
   
   (b) CF = 0, SF = 1, ZF = 0, OF = 1
   
   (c) CF = 0, SF = 1, ZF = 0, OF = 0

7. Code example:
    
    ```assembly
    mov ax, val2
    neg ax
    add ax, bx
    sub ax, val4
    ```

8. No

9. Yes

10. Yes (for example, mov al, −128... followed by... neg al)

11. No

12. Setting the Carry and Overflow flags at the same time:
    
    ```assembly
    mov al, 80h
    add al, 80h
    ```

### 4.3 Data-Related Operators and Directives

1. False

2. False

3. True

4. False

5. True

6. Data directive:
    
    ```assembly
    .data
    ALIGN 2
    myBytes BYTE 10h, 20h, 30h, 40h
    ```
Data Transfers, Addressing, and Arithmetic

7. (a) 1   (b) 4   (c) 4   (d) 2   (e) 4   (f) 8   (g) 5

8. mov dx, WORD PTR myBytes

9. mov al, BYTE PTR myWords+1

10. mov eax, DWORD PTR myBytes

11. Data directive:
    myWordsD LABEL DWORD
    myWords WORD 3 DUP(?),2000h
    .data
    mov eax,myWordsD

12. Data directive:
    myBytesW LABEL WORD
    myBytes BYTE 10h,20h,30h,40h
    .code
    mov ax,myBytesW

### 4.4 Indirect Addressing

1. False
2. True
3. False
4. False

5. True - (the PTR operator is required)

6. True

7. (a) 10h (b) 40h (c) 003Bh (d) 3 (e) 3 (f) 2

8. (a) 2010h (b) 003B008Ah (c) 0 (d) 0 (e) 0044h

### 4.5 JMP and LOOP Instructions

1. True
2. False
3. 4,294,967,296 times

4. False
5. True
6. CX
7. ECX

8. False (−128 to +127 bytes from the current location)

9. This is a trick! The program does not stop, because the first LOOP instruction decrements ECX to zero. The second LOOP instruction decrements ECX to FFFFFFFFh, causing the outer loop to repeat.

10. Insert the following instruction at label L1: push ecx. Also, insert the following instruction before the second LOOP instruction: pop ecx. (Once you have added these instructions, the final value of eax is 1Ch.)

5 Procedures

5.1 Introduction

(no review questions)

5.2 Linking to an External Library

1. False - (it contains object code)

2. Code example:
   MyProc PROTO
3. Code example:
   call MyProc

4. Irvine32.lib
5. Kernel32.lib

6. Kernel32.dll is a dynamic link library that is a fundamental part of the MS-Windows operating system.

7. %1

5.3 The Book’s Link Library

1. RandomRange procedure
2. WaitMsg procedure
Procedures

3. Code example:

```plaintext
mov eax, 700
call Delay
```

4. WriteDec procedure

5. Gotoxy procedure

6. INCLUDE Irvine32.inc

7. PROTO statements (procedure prototypes) and constant definitions. (There are also text macros, but they are not mentioned in this chapter.)

8. ESI contains the data's starting address, ECX contains the number of data units, and EBX contains the data unit size (byte, word, or doubleword).

9. EDX contains the offset of an array of bytes, and ECX contains the maximum number of characters to read.

10. Carry, Sign, Zero, and Overflow, and EFL displays the flag bits in hexadecimal.

11. Code example:

```plaintext
.data
str1 BYTE "Enter identification number: ", 0
idStr BYTE 15 DUP(?)
.code
    mov edx, OFFSET str1
    call WriteString
    mov edx, OFFSET idStr
    mov ecx, (SIZEOF idStr) - 1
    call ReadString
```

5.4 Stack Operations

1. SS and ESP

2. The runtime stack is only type of stack that is managed directly by the CPU. For example, it holds the return addresses of called procedures.

3. LIFO stands for "last in, first out". The last value pushed into the stack is the first value popped out from the stack.

4. ESP is decremented by 4.

5. True

6. False - (you can push both 16-bit and 32-bit values)

7. True
8. False (yes, it can, from the 80186 processor onwards).

9. PUSHAD

10. PUSHFD

11. POPFD

12. NASM’s approach permits the programmer to be specific about which registers are to be pushed. PUSHAD, on the other hand, does not have that flexibility. This becomes important when a procedure needs to save several registers, and at the same time return a value to its caller in the EAX register. In this type of situation, EAX cannot be pushed and popped because the return value would be lost.

5.5 Defining and Using Procedures

1. True

2. False

3. Execution would continue beyond the end of the procedure, possibly into the beginning of another procedure. This type of programming bug is often difficult to detect!

4. Receives indicates the input parameters given to the procedure when it is called. Returns indicates what value, if any, the procedure produces when it returns its caller.

5. False - (it pushes the offset of the instruction following the call)

6. True

7. True

8. False - (there is no NESTED operator)

9. True

10. False

11. True - (it also receives a count of the number of array elements)

12. True

13. False

14. False

15. The following statements would have to be modified:

   add eax,[esi]  becomes -->  add ax,[esi]
   add esi,4      becomes -->  add esi,2
5.6 Program Design Using procedures

1. functional decomposition, or top-down design
2. Clrscr, WriteString, ReadInt, and WriteInt
3. A stub program contains all of its important procedures, but the procedures are either empty or nearly empty.
4. False - (it receives a pointer to an array)
5. The following statements would have to be modified:

\[
\begin{align*}
\text{mov [esi],eax} & \quad \text{becomes} \quad \text{mov [esi],ax} \\
\text{add esi,4} & \quad \text{becomes} \quad \text{add esi,2}
\end{align*}
\]
6. Flowchart of the PromptForIntegers procedure:

6. Conditional Processing

6.1 Introduction

(no review questions)

6.2 Boolean and Comparison Instructions

1. (a) 00001011  (b) 01001000  (c) 01101111  (d) 10100011

2. (a) 85h  (b) 34h  (c) BFh  (d) AEh
3. (a) CF=0, ZF=0, SF=0
   (b) CF=0, ZF=0, SF=0
   (c) CF=1, ZF=0, SF=1
4. and ax, 00FFh
5. or ax, 0FF00h
6. xor eax, 0FFFFFFFFh
7. test eax, 1 ; (low bit set if eax is odd)
8. Code example:
   .data
   memVal DWORD ?
   .code
   mov al, BYTE PTR memVal
   xor al, BYTE PTR memVal+1
   xor al, BYTE PTR memVal+2
   xor al, BYTE PTR memVal+3

6.3 Conditional Loops

1. JA, JNBE, JAE, JNB, JB, JNAE, JBE, JNA
2. JG, JNLE, JGE, JNL, JL, JNGE, JLE, JNG
3. JECXZ
4. Yes
5. No (JB uses unsigned operands, whereas JL uses signed operands.)
6. JNBE
7. JL
8. No (8109h is negative, and 26h is positive.)
9. Yes
10. Yes (The unsigned representation of –42 is compared to 26.)
11. Code:
    cmp dx, cx
    jbe L1
12. Code:
cmp ax,cx
jg L2

13. Code:

and al,11111100b
jz L3
jmp L4

6.4 Conditional Loop Instructions

1. False
2. True
3. True
4. Code example:

.data
array SWORD 3,5,14,-3,-6,-1,-10,10,30,40,4
sentinel SWORD 0
.code
main PROC
mov esi,OFFSET array
mov ecx,LENGTHOF array
next:
test WORD PTR [esi],8000h ; test sign bit
pushfd ; push flags on stack
add esi,TYPE array
popfd ; pop flags from stack
loopz next ; continue loop while ZF=1
jz quit ; none found
sub esi,TYPE array ; ESI points to value

5. If a matching value were not found, ESI would end up pointing beyond the end of the array. This could cause data to be corrupted if ESI were dereferenced and used to modify memory.

6.5 Conditional Structures

(We will assume that all values are unsigned in this section).

1. Code example:

cmp bx,cx
jna next
mov X,1
next:
2. Code example:

```
cmp dx, cx
jnbe L1
mov X, 1
jmp next
L1: mov X, 2
next:
```

3. Code example:

```
cmp val1, cx
jna L1
cmp cx, dx
jna L1
mov X, 1
jmp next
L1: mov X, 2
next:
```

4. Code example:

```
cmp bx, cx
ja L1
cmp bx, val1
ja L1
mov X, 2
jmp next
L1: mov X, 1
next:
```

5. Code example:

```
cmp bx, cx
jna L1 ; bx > cx?
cmp bx, dx
jna L1 ; no: try condition after OR
jmp L2 ; yes: set X to 1
;-----------------OR(dx > ax)------------------------
L1: cmp dx, ax
jna L3 ; dx > ax?
L2: mov X, 1
jmp next ; yes: set X to 1
L3: mov X, 2
next: ; and quit
```
6.6 Application: Finite-State Machines

1. A directed graph (also known as a diagraph).
2. Each node is a state.
3. Each edge is a transition from one state to another, caused by some input.
4. State C
5. An infinite number of digits.
6. The FSM enters an error state.
7. No. The proposed FSM would permit a signed integer to consist of only a plus (+) or minus (–) sign. The FSM in Section 6.6.2 would not permit that.
8. FSM that recognizes real numbers without exponents:

```
start   digit   digit   digit
A -->   +,-    (.)   C
```

6.7 Using the .IF Directive (Optional)

(no review questions)

7 Integer Arithmetic

7.1 Introduction

(no review questions)

7.2 Shift and Rotate Instructions

1. ROL
2. RCR
3. SAR
4. RCL
5. Code example:

```
shr al,1 ; shift AL into Carry flag
jnc next ; Carry flag set?
```