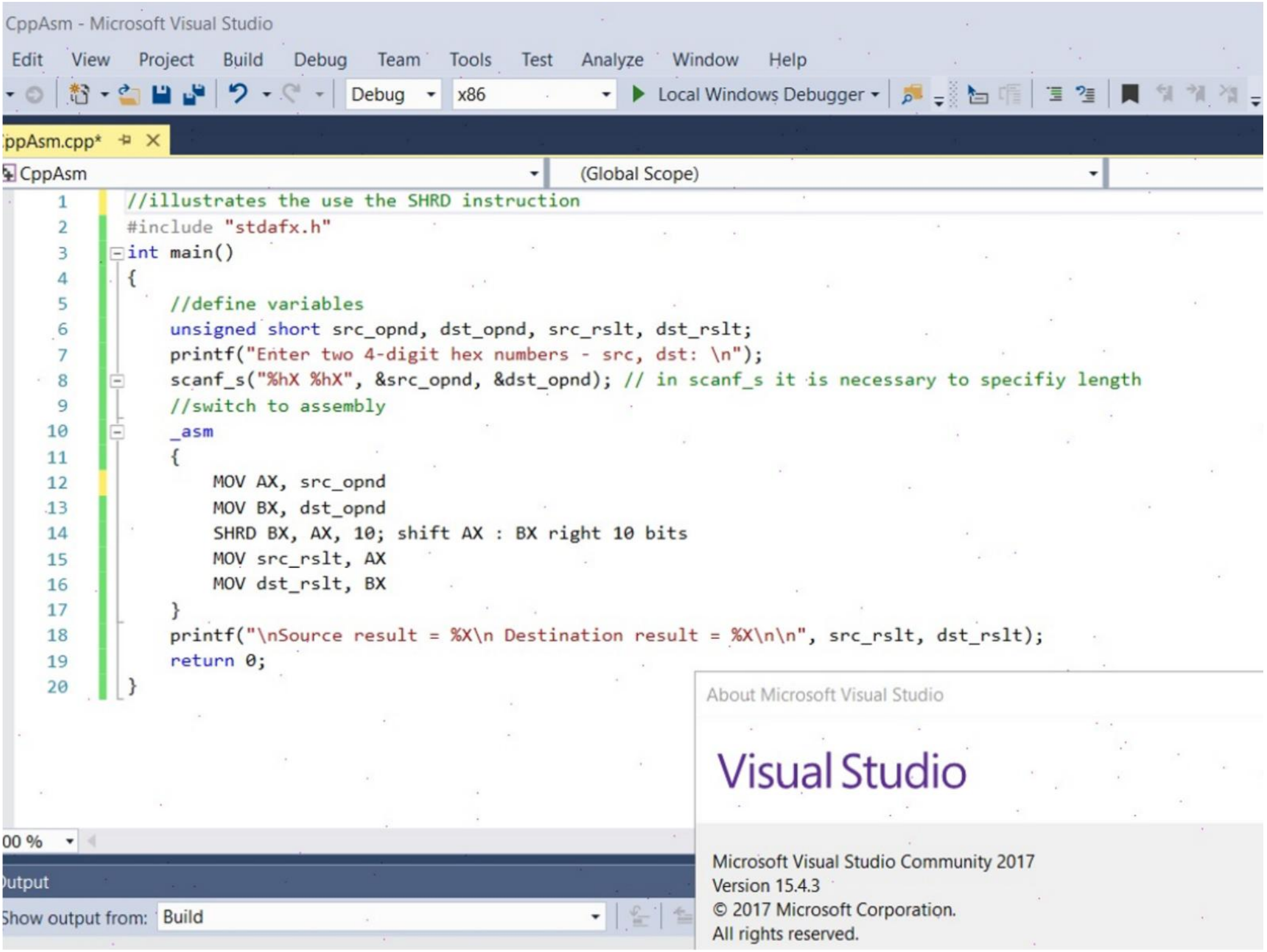

CppAsm - Microsoft Visual Studio
Edit View Project Build Debug Team Tools Test Analyze Window Help
Debug
x86
Local Windows Debugger
ppAsm.cpp*
CppAsm
(Global Scope)
1 //illustrates the use the SHRD instruction
2 #include "stdafx.h"
3 int main()
4 {
5 //define variables
6 unsigned short src_opnd, dst_opnd, src_rslt, dst_rslt;
7 printf("Enter two 4-digit hex numbers - src, dst: \n");
8 scanf_s("%hX %hX", &src_opnd, &dst_opnd); // in scanf_s it is necessary to specifiy length
9 //switch to assembly
10 _asm
11 {
12 MOV AX, src_opnd
13 MOV BX, dst_opnd
14 SHRD BX, AX, 10; shift AX : BX right 10 bits
15 MOV src_rslt, AX
16 MOV dst_rslt, BX
17 }
18 printf("\nSource result = %X\n Destination result = %X\n\n", src_rslt, dst_rslt);
19 return 0;
20 }
00 %
utput
Show output from: Build
About Microsoft Visual Studio
Visual Studio
Microsoft Visual Studio Community 2017
Version 15.4.3
© 2017 Microsoft Corporation.
All rights reserved.

reserved for the operating system; levels 1 and 2 are for less critical operating system functions; level 3 is reserved for application programs.

Multitasking can execute several programs simultaneously. These programs (or tasks) may have separate segments or may share segments. The programs are written as if on a dedicated microprocessor. The multitasking software then simulates a dedicated microprocessor by allocating a *virtual processor*. The substitution of these virtual processors creates the appearance of a dedicated processor, where each task has an allotment of CPU time. If a task must wait for an I/O operation, it suspends its operation. The computer must be able to switch rapidly between tasks, save and load the entire machine state, prevent interference of tasks, and prioritize tasks.

Protected mode allows reliable multitasking and prevents tasks from overwriting code or data of another task. Thus, if a program fails, the effects are confined to a limited area and the operating system will not be affected. Protection is applied to segments and pages.

3.9.2 Real Mode

Real mode (or real-address mode), is an operating mode that implements the 8086 architecture and is initialized when power is applied or the system is reset. Thus, real mode allows compatibility with programs that were written for the 8086 processor. Real mode allows access to the 32-bit register set and protected mode when processor extensions are in effect. Real mode does not support multitasking.

The real-address mode implements a segmented memory consisting of 64 kilobytes in each segment. Each segment register is associated with either code, data, or stack, which are contained in separate segments. Real mode segments begin on 16-byte boundaries. Memory is accessed using a 20-bit address that is calculated by adding a right-aligned 16-bit offset to a segment register that is multiplied by 16; that is, the segment register is shifted left four bits. This provides an address space of 1 megabyte.

3.10 Problems

3.1 Specify the addressing mode for each operand in the instructions shown below.

(a) ADD AX, [BX]

(b) ADD CX, [BP + 8]

(c) ADD EBX, ES:[ESI – 4]

(d) ADD [EBP +EDI + 6], 10

3.2 Let DS = 1000H, SS = 2000H, BP = 1000H, and DI = 0100H. Using the real addressing mode with a 20-bit address, determine the physical address memory address for the instruction shown below.

```
MOV   AL, [BP + DI]
```

3.3 Let DS = 1100H, DISPL = –126, and SI = 0500H. Using the real addressing mode with a 20-bit address, determine the physical address memory address for the instruction shown below.

```
MOV   DISPL[SI], DX
```

3.4 Let the directive shown below be located in the data segment at offset 04F8H, where DW 10 DUP(?) specifies that a word is defined with an unknown value and duplicated ten times.

Let BX = 04F8H, SI = 04FAH, and DI = 0006H. Let the 10 reserved words beginning at location VALUE be labelled

WORD1, WORD2, . . . , WORD10

Indicate the word that is referenced by the memory operand in each of the move instructions shown below.

```
(a)    MOV   AX, VALUE + 2
(b)    MOV   AX, [BX]
(c)    MOV   AX, [SI]
(d)    MOV   AX, [BX + DI – 2]
```

3.5 Differentiate between the operation of the following two move instructions:

```
(a)    MOV   EAX, 1234ABCDH
(b)    MOV   AX, [1234H]
```

3.6 Differentiate between the operation of the following two move instructions:

```
(a)    MOV   EBX, 1234ABCDH
(b)    MOV   [EBX], 1234H
```

3.7 Use base and index plus displacement addressing to obtain the physical address of the memory operand for the following conditions:

BX = 6F30H, SI = 1000H, DS = 85H, Displacement = 2106H

3.8 Obtain the real (physical) address that corresponds to each segment:offset pair shown below.

	Segment:	Offset
(a)	2B8C:	8D21
(b)	F000:	FFFF
(c)	3BAC:	90DF

Pop 16 low-order flags (POPF) The POPF instruction is used when the operand-size attribute is 16. It pops the top word from the stack into the low-order 16 bits of the EFLAGS register, then increments the SP register by two to point to the new stack top. The POPF instruction, in conjunction with the PUSHF instruction, allows a procedure to save the calling program's flags and then to restore the flags. The following flag bits are affected: OF, DF, TF, SF, ZF, AF, PF, and CF. The IOPL flag and the IF flag are also affected depending on the privilege level.

Pop all 32 flags (POPFD) The POPFD instruction pops a doubleword off the stack into the EFLAGS register if the operand-size attribute is 32 bits, then increments the ESP stack pointer by four. The POPF and POPFD instructions have the same operation code and their operation is identical except for the registers that are loaded from the stack — FLAGS or EFLAGS — which are a function of the operand-size attribute. Some X86 assemblers may cause the operand size to be 16 bits for a POPF instruction or to be 32 bits when a POPFD instruction is encountered. Other X86 assemblers use the current value of the operand-size attribute.

7.4 Problems

7.1 Given DS = 2800H, BX = 0400H, SP = 1000H, SS = 2F00H, and memory location 28400H = A020H, find the real (physical) data address of the source operand and the real address of the stack top when the PUSH [BX] instruction is executed. Show the contents of the stack top in memory and determine the new contents of the stack pointer SP.

7.2 Given DS = FF00H, SI = 0008H, SP = 0FEAH, SS = 2F00H, and memory location 2FFEAH = 3BC5H, find the real (physical) data address of the destination operand and the real address of the stack top when the POP [SI] instruction is executed. Show the contents of the stack top in memory and determine the new contents of the stack pointer SP.

7.3 Determine the result of each instruction for the following program segment:

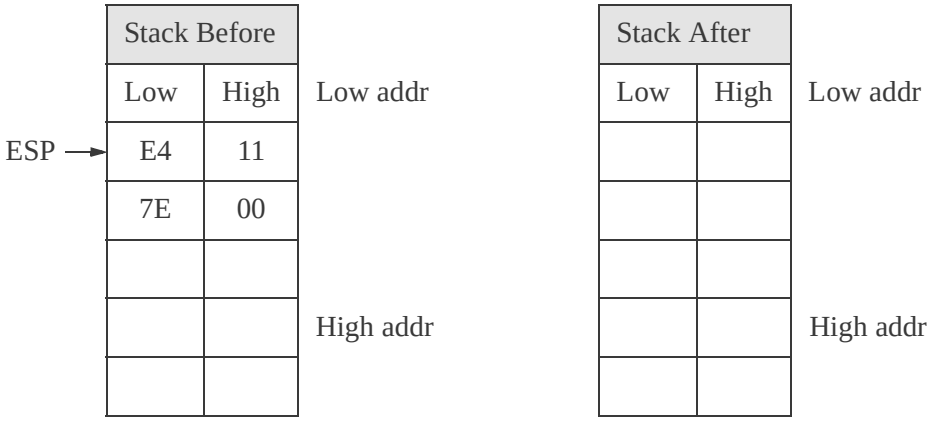
```
PUSH  EBP
MOV   EBP, ESP
PUSH  EAX
PUSH  EBX
PUSH  ECX
            . . .
              //continued on next page
```

```
            . . .
MOV   EAX, [EBP – 12]
MOV   EBX, [EBP – 8]
MOV   ECX, [EBP – 4]
            . . .
ADD   ESP, 12
POP   EBP
```

7.4 Determine the result of each instruction for the following program segment:

```
PUSH  EAX
PUSH  EBX
PUSH  ECX
PUSH  EBP
MOV   EBP, ESP
            . . .
MOV   EAX, [EBP + 4]
MOV   EBX, [EBP + 8]
MOV   ECX, [EBP + 12]
            . . .
POP   EBP
ADD   ESP, 12
```

7.5 The partial contents of a stack are shown below before execution of the program segment listed below. Determine the contents of the stack after the program has been executed and indicate the new top of stack.

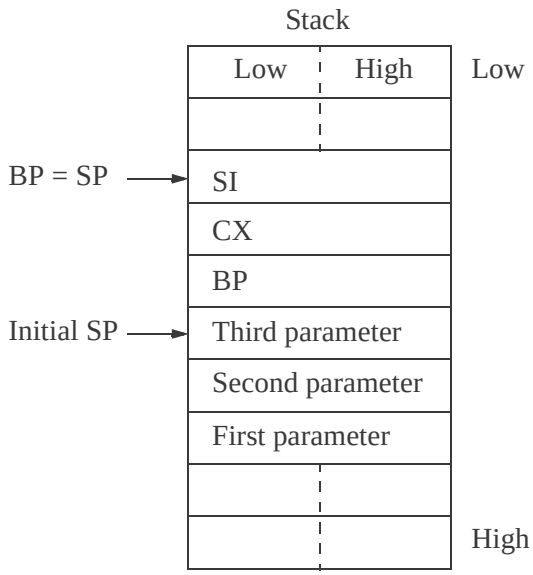


```
POP   BX
MOV   AH, BH
ADD   AH, BL
MOV   BH, AH
PUSH  BX
```

7.6 Why will a PUSH AL instruction cause an error message to be displayed?

7.7 Assume that a stack contains three parameters and that SP points to the third parameter, as shown in the diagram below. Determine how each parameter can be accessed if the following program segment is executed:

```
PUSH  BP
PUSH  CX
PUSH  SI
MOV   BP, SP
```



7.8 Write an assembly language program — using the PUSH and POP instructions — that adds four decimal integers, then displays their sum. Embed the assembly module in a C program. The decimal integers are entered from the keyboard.

7.9 Write an assembly language program — using the PUSH and POP instructions — that adds five hexadecimal integers, then displays their sum. Embed the assembly module in a C program. The hexadecimal integers are entered from the keyboard. Enter hexadecimal integers that range from one character to eight characters. Display the sum as upper-case hexadecimal characters.

```
Enter an 8-digit hexadecimal number:
AB34CD56

Immediate rotate 1
Before rotate = AB34CD56
After rotate = 559A66AB

Immediate byte rotate 6
Before rotate = AB34CD56
After rotate = 5AACD335

CL register rotate 3
Before rotate = AB34CD56
After rotate = D56699AA

Rotate through carry 4
Before rotate = AB34CD56
After rotate = CAB34CD5

Press any key to continue . . . _
```

Figure 8.32 (Continued)

8.11 Problems

8.1 For each of the sections shown below, where the values are in hexadecimal, obtain the resulting values for register AX and the flags. Then write an assembly language module that is embedded in a C program to verify the results and print the results and the flags. When inserting hexadecimal numbers in an assembly language program manually, the first digit must be a number 0 through 9, then the digits A through F, if required, followed by the hexadecimal radix specifier H.

(a) AX = FA75 **AND** AX, 000FH

(b) AX = FA75 **OR** AX, 0FFF0H

(c) AX = FA75 **XOR** AX, 0FFFFH

8.2 Use an XOR instruction to do the following:

(a) Exclusive-OR the memory operand addressed by register BX with the contents of register DX. Save the result in the memory location addressed by BX.

(b) Change the contents of register CL from 53H to 73H.

8.3 Using only logic instructions, write one instruction for each of the following parts that will perform the indicated operations.

(a) Set the low-order four bits of register AX.

(b) Reset the high-order three bits of register AX.

(c) Invert bits 7, 8, and 9 of register AX.

8.4 Write an assembly language program — not embedded in a C program — to perform an AND operation of two single hexadecimal characters that are entered from the keyboard. There is no space between the characters. Display the result of the AND operation and the low-order four bits of the flags register.

8.5 Assume that register AL contains an ASCII code for an uppercase letter. Write a single instruction to change the contents to the corresponding lowercase letter.

8.6 Determine the contents of registers AX, AH, and BL after the following program segment has been executed:

```
MOV     CL, 3
MOV     AX, 7FH
MOV     BX, 0505H
ROL     AX, CL
AND     AH, BH
OR      BL, AL
```

8.7 Write an assembly language module embedded in a C program that executes the bit test instructions BTS, BTR, and BTC. Enter a 4-digit hexadecimal number for the operand and a 2-digit hexadecimal number for the bit offset.

Store the operand in register AX. The program consists of two segments: one segment for an immediate bit offset and one segment for an offset in register BX. Include bit offsets greater than 16 so that the bit offset modulo-16 will generate a bit position in register AX. Display the results of the BTS, BTR, and BTC instructions for both the immediate bit offsets and the bit offset in register BX.

8.8 Write an assembly language module embedded in a C program that uses the *bit scan forward* (BSF) instruction to detect whether the first bit detected is in the low-order half or the high-order half of a 32-bit operand. Do not determine the bit position.

8.9 Repeat problem 8.8 using the *bit scan reverse* (BSR) instruction.

8.10 Determine the contents of register AX after execution of the following program segment:

```
MOV   AX, -15
MOV   CL, 3
SAL   AX, CL
```

8.11 Determine the contents of register AX after execution of the following program segment:

```
MOV   AX, -32,668
MOV   CL, 5
SAR   AX, CL
```

8.12 Determine the contents of register EBX and the flags register after execution of the program segment shown below. Then write an assembly language module embedded in a C program to verify the results.

```
MOV   EBX, OFFFFA85BH
SAL   EBX, 20
MOV   EBX, OFFFFA85BH
SAR   EBX, 20
```

8.13 Determine the contents of the destination register EBX, the source register EDX, and the flags register after execution of the program segment shown below and on the following page. Then write an assembly language module embedded in a C program to verify the results.

```
MOV   EBX, 1234ABCDH
MOV   EDX, OABCD1234H
```

```
MOV   CL, 16
SHLD  EBX, EDX, CL

MOV   EBX, 1234ABCDH
MOV   EDX, OABCD1234H
MOV   CL, 16
SHRD  EBX, EDX, CL
```

8.14 Write an assembly language module embedded in a C program that will multiply and divide a decimal number by 8 using arithmetic shift instructions. When dividing, some numbers will have the fraction truncated.

8.15 Write an assembly language program — not embedded in a C program — that requests one of three 1-digit numbers (1, 2, or 3) to be entered from the keyboard. Determine which number was entered, then display the number. Use the AND instruction to remove the ASCII bias.

8.16 Let register EAX contain AD3E14B5H and the carry flag be set. Determine the contents of register EAX after the following instruction is executed:

```
RCL   EAX, 6
```

8.17 Let register EAX contain 12345678H and the carry flag be set. Determine the contents of register EAX after the following instruction is executed:

```
RCR   EAX, 6
```

8.18 Let register EAX contain 1C78FDA5 and the carry flag be set. Determine the contents of register EAX after the following instruction is executed:

```
RCR   EAX, 6
```

8.19 Write an assembly language module embedded in a C program that illustrates the *shift logical left* (SHL), *shift logical right* (SHR), *rotate left* (ROL), and *rotate right* (ROR) instructions. Enter an 8-digit hexadecimal number for the destination operand with a count of 40 stored in register CL.

8.20 This problem is similar to Problem 8.15. Write an assembly language module embedded in a C program that requests one of three 1-digit numbers (3, 4, or 5) to be entered from the keyboard. Determine which number was entered, then display the number. If an incorrect number is entered, then display that information.

9.6 Determine the contents of registers AL and BL after the program segment shown below has executed. Then write an assembly language module embedded in a C program to verify the results.

```
        MOV   AL, 00H
        MOV   BL, -5
LOOP1:  ADD   BL, 2
        INC   AL
        ADD   BL, -1
        JNZ   LOOP1
```

9.7 Let AX = 47E7H and BX = 4BED, then add the two registers and obtain the state of the following flags: AF, CF, OF, PF, SF, and ZF.

9.8 Write an assembly language module — not embedded in a C program — that requests two characters to be entered from the keyboard. Display the first character unchanged and display the second character shifted left logically one bit position. The characters can be numbers or special characters in the range 21H to 3FH.

9.9 Write an assembly language program — not embedded in a C program — that removes all nonletters from a string of characters that are entered from the keyboard.

9.10 Write an assembly language module embedded in a C program that adds two 4-digit hexadecimal numbers and displays the sum and the low-order two bytes of the EFLAGS register.

9.11 Write an assembly language module embedded in a C program that uses the *add with carry* (ADC) instruction in the addition of four 4-digit hexadecimal numbers: two for the augend and two for the addend. Display the high-order sum and the low-order sum. Also display the high-order byte and the low-order byte of the EFLAGS register.

9.12 Given the program shown below, determine the contents of the RSLT field after each of the following hexadecimal characters are entered from the keyboard:

123456_{16} $ABCDEF_{16}$ $4a5b6c_{16}$ $UVWXYZ_{16}$

9.18 Let register DL = F3H and register BH = 72H. Then execute the instruction shown below to obtain the difference and the state of the following flags: OF, SF, ZF, AF, PF, and CF.

```
SUB   DL, BH
```

9.19 Determine the contents of register AL after the program shown below has executed.

```
//decrement.cpp
//program to illustrate the DEC instruction.

#include "stdafx.h"

int main (void)
{

//define variables
   char result;

//switch to assembly
      _asm
      {
         MOV   AL, 3AH
         MOV   CH, 0A9H
         ADD   CH, 06H
         ADD   AL, CH

         NEG   AL
         DEC   AL
         MOV   result, AL
      }

   printf ("\nAL = %X\n\n", result);

   return 0;
}
```

```
LP1:    MOV   AH, 0
        MOV   AL, [SI]
        ADC   AL, [DI]
        AAA
        OR    AL, 30H
        MOV   [BX], AL
        DEC   SI
        DEC   DI
        INC   BX
        LOOP  LP1

        MOV   AH, 09H
        LEA   DX, RSLT
        INT   21H

BEGIN   ENDP
        END   BEGIN
```

9.21 Write an assembly language program — not embedded in a C program — to reverse the order and change to uppercase all letters that are entered from the keyboard. The letters that are entered may be lowercase or uppercase.

9.22 Eight digits, either 0 or 1, are entered from the keyboard. Write an assembly language program — not embedded in a C program — to determine the parity of the byte. If there are an even number of 1s in the byte, then a flag bit (PF) is set to a value of 1 maintaining odd parity over all nine bits — the eight bits plus the parity flag; otherwise, the parity flag is reset to 0. Display the parity flag. Draw a flowchart that represents the operation of the odd parity generator. Remember that a 0 and a 1 are represented in ASCII as 30H and 31H, respectively.

9.23 Write an assembly language program — not embedded in a C program — to sort *n* single-digit numbers in ascending numerical order that are entered from the keyboard. A simple exchange sort is sufficient for this program. The technique is slow, but appropriate for sorting small lists. The method is as follows:

1. Search the list to find the smallest number.
2. After finding the smallest number, exchange this number with the first number.
3. Search the list again to find the next smallest number, starting with the second number.
4. Then exchange this (second smallest) number with the second number.
5. Repeat this process, starting with the third number, then the fourth number, and so forth.

9.24 Let register AL = 6CH and register BL = A7H. After execution of

```
IMUL    BL
```

determine the hexadecimal contents of register AX.

9.25 Determine the contents of register pair DX:AX after the following program segment has been executed:

```
OP1    DW     8080H
OP2    DB     0A2H
MOV    AL,    OP2
CBW
MUL    OP1
```

9.26 Determine the results of each of the following operations:

Before	Instruction
AX = FF FFH	MUL AX
AX = FF E4H BX = 04 C2H	MUL BX
AX = 00 17H CX = 00 B2H	IMUL CX
AX = FF E4H BX = 04 C2H	IMUL BX

9.27 Determine the hexadecimal contents of register AX after the following program segment has been executed:

```
       MOV    CX, 3
       MOV    AX, 2
LP1:   MUL    AX
       LOOP   LP1
```

9.28 Write an assembly language module embedded in a C program that multiplies two unsigned (MUL) hexadecimal numbers. Enter numbers from the keyboard that produce a product where the high-order half is all zeroes and where the high-order half is not all zeroes. The overflow flag and the carry flag should be set if the high-order half of the product is not all zeroes. Display the products and the low-order 16 bits of the EFLAGS register.

9.35 Write an assembly language program — not embedded in a C program — for a sequence detector that detects the number of times that the four-bit sequence 0111 appears in the keyboard input buffer (OPFLD) when n characters are entered. The variable n has the following range: $1 \le n \le 30$.

The bit configuration 0111 may be in the high-order four bits of the keyboard character, the low-order four bits, or both high- and low-order four bits. Display the number of times that the sequence occurs. The DIV instruction can be used in this program. Draw a flowchart prior to designing the program. An example of input data is shown below.

7	5	g	q	T	W
0011 **0111**	0011 0101	0110 **0111**	**0111** 0001	0101 0100	**0111 0111**

0111 occurs five times

9.36 Write an assembly language program — not embedded in a C program — that calculates the surface area of a rectangular solid box. The length, width, and height are single-digit numbers that are entered from the keyboard. The program will use the ADD, MUL, and DIV instructions. Enter at least three sets of numbers and display the corresponding surface area.

9.37 Determine the contents of register pair DX:AX for parts (a) and (b) after execution of the following program segment:

```
MOV   AX, dvdnd
CWD
IDIV  dvsr
```

(a) dvdnd = 059A; dvsr = FFC7

(b) dvdnd = 7C58; dvsr = FFA9

9.38 Write an assembly language module embedded in a C program that evaluates the expression shown below, where $x = 400_{10}$ (0190_{16}), $y = 5_{10}$ (0005_{16}), $z = 1000_{10}$ ($03E8_{16}$), and $v = 4000_{10}$ ($0FA0_{16}$).

$$[v - (x \times y + z - 500)] / x$$

The register stack for the program of Figure 12.7 is shown in Figure 12.8. The calling program first pushes the operand in AX onto the stack and then pushes the operand in BX onto the stack. The CALL to the near CALC procedure pushes IP onto the stack. The procedure then pushes the base pointer, BP, onto the stack to be used as an offset in the stack segment. In order to accomplish this, BP is made equal to the stack pointer, SP, so that BP points to the top of stack.

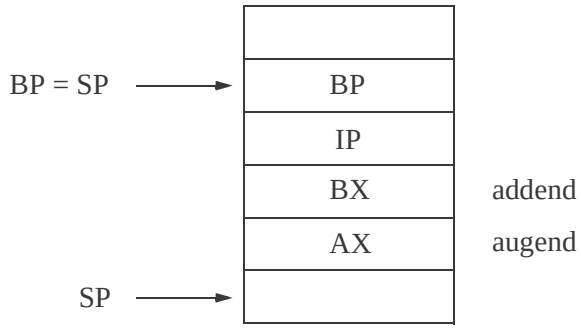


Figure 12.8 Stack usage for the program of Figure 12.7.

12.4 Problems

12.1 Write an assembly language program — not embedded in a C program — that uses a procedure to multiply two single-digit operands. Enter several operands for the multiplicand and multiplier and display the products.

12.2 Write an assembly language program — not embedded in a C program — that uses a procedure to obtain the area of a triangle from two integers that are entered from the keyboard. Enter several sets of single-digit numbers for the base and height and display the areas.

12.3 Calculate the area of a triangle using an assembly language module embedded in a C program. Floating-point numbers for the base and height are entered from the keyboard. In this problem, a procedure is not necessary. Unlike Problem 12.2, odd-valued bases will not be truncated. Enter several floating-point numbers for the base and height and display the areas.

12.4 Write an assembly language program — not embedded in a C program — that uses a procedure to exclusive-OR six hexadecimal characters that are entered from the keyboard. The following characters are exclusive-ORed: the first and third; the second and fourth; the third and fifth; and the fourth and sixth. The keyboard data can be any hexadecimal characters; for example, 2T/}b*.

13.3 Determine the contents of RSLT after execution of the following program:

```
;movs_byte_rev.asm
;-----------------------------------------------------
.STACK

;-----------------------------------------------------
.DATA
RSLT    DB  ODH, OAH, '123456789    $'

;-----------------------------------------------------
.CODE
BEGIN PROC  FAR

;set up pgm ds
      MOV   AX, @DATA       ;get addr of data seg
      MOV   DS, AX          ;move addr to ds
      MOV   ES, AX          ;move addr to es

;-----------------------------------------------------
;move string  elements
      STD                   ;right-to-left
      MOV   CX, 4           ;count in cx
      LEA   SI, RSLT+10     ;addr of rslt+10 -> si as src
      LEA   DI, RSLT+8      ;addr of rslt+8 -> di as dst

REP   MOVSB                 ;move bytes to dst

;-----------------------------------------------------
;display result
      MOV   AH, O9H         ;display string
      LEA   DX, RSLT        ;put addr of rslt in dx
      INT   21H             ;dos interrupt

BEGIN ENDP
      END   BEGIN
```

13.4 Write an assembly language program — not embedded in a C program — that receives six hexadecimal characters entered from the keyboard and stores them in the OPFLD area. The result area contains a second string of ABC-DEF. Then the first three characters from OPFLD are moved to replace the last three characters in the result area. Display the resulting contents of the result area. This program is similar to Problem 13.1, except that the second string is given; therefore, only one LOOP instruction is required. The REP prefix is not used.

13.5 Although this problem does not use the MOVS instruction, it does move modified characters from a source to a destination. Assume that the following characters are entered from the keyboard:

```
OFF, OO, OEE, 22, OC6, OF5
```

Then show the result after the program shown below has been executed.

```
//array_xor2.cpp
#include "stdafx.h"
int main (void)
{
//define variables
   unsigned char hex1, hex2, hex3, hex4, hex5, hex6;

   printf ("Enter six 2-digit hexadecimal characters:
            \n");
   scanf ("%X %X %X %X %X %X", &hex1, &hex2, &hex3,
            &hex4, &hex5, &hex6);

//switch to assembly
      _asm
      {
         MOV   AL, hex1
         XOR   hex3, AL

         MOV   AL, hex2
         XOR   hex4, AL

         MOV   AL, hex3
         XOR   hex5, AL

         MOV   AL, hex4
         XOR   hex6, AL
      }

   printf ("\nhex1 = %X", hex1);
   printf ("\nhex2 = %X", hex2);
   printf ("\nhex3 = %X", hex3);
   printf ("\nhex4 = %X", hex4);
   printf ("\nhex5 = %X", hex5);
   printf ("\nhex6 = %X\n\n", hex6);
   return 0;
}
```

13.6 Write an assembly language module embedded in a C program using the LODS instruction with explicit operands that receive byte, word, and doubleword operands that are entered from the keyboard. Then display the operands.

13.7 Given the program shown below, obtain the result when the following four-digit hexadecimal characters are entered from the keyboard separately:

```
1233  9999  2E+F  )2<=
```

```
;lods_stos.asm
;illustrates using the load string
;and store string no operand instructions
;-------------------------------------------------------
.STACK
;-------------------------------------------------------
.DATA
PARLST   LABEL BYTE
MAXLEN   DB 10
ACTLEN   DB ?
OPFLD    DB  10 (?)
PRMPT    DB ODH, OAH, 'Enter four 1-digit hex chars: $'
RSLT     DB ODH, OAH, 'Result =              $'
;-------------------------------------------------------
.CODE
BEGIN PROC FAR

;set up pgm ds
      MOV   AX, @DATA      ;get addr of data seg
      MOV   DS, AX         ;move addr to ds

;read prompt
      MOV   AH, 09H        ;display string
      LEA   DX, PRMPT      ;put addr of prompt in dx
      INT   21H            ;dos interrupt
;-------------------------------------------------------
;keyboard request rtn to enter characters
      MOV   AH, 0AH        ;buffered keyboard input
      LEA   DX, PARLST     ;put addr of parlst in dx
      INT   21H            ;dos interrupt
;-------------------------------------------------------
      MOV   CX, 3          ;# of iterations for loop
      LEA   SI, OPFLD      ;addr of opfld -> si
      LEA   DI, OPFLD+1    ;addr of opfld+1 -> di
      LEA   BX, RSLT+11    ;addr of rslt+11 -> bx
      CLD                  ;left-to-right transfer

                           //continued on next page
```