Answers to End of Chapter Reviews and Exercises

for Assembly Language for x86 Processors, 7th Edition

by Kip R. Irvine

Chapters 1 to 13

*Revision date: 1/18/2014*

Chapter 1

1.7.1 Short Answer Questions

1. Most significant bit (the highest numbered bit).

2. (a) 53 (b) 150 (c) 204

3. (a) 110001010 (b) 110010110 (c) 100100001

4. 00000110

5. (a) 8 (b) 32 (c) 64 (d) 128

6. (a) 12 (b) 16 (c) 16

7. (a) 35DA (b) CEA3 (c) FEDB

8. (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

9. (a) 58 (b) 447 (c) 16534

10. (a) 98 (b) 1203 (c) 671

11. (a) FFE8 (b) FEB5

12. (a) FFEB (b) FFD3

13. (a) 27641 (b) −16093

14. (a) 19666 (b) −32208

15. (a) −75 (b) +42 (c) −16

16. (a) −128 (b) −52 (c) −73

17. (a) 11111011 (b) 11010110 (c) 11110000

18. (a) 10111000 (b) 10011110 (c) 11100110

19. (a) AB2 (b) 1106

20. (a) B82 (b) 1316

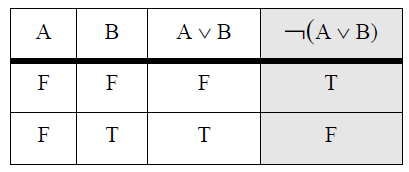
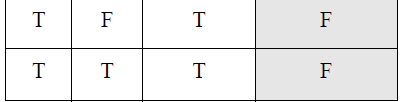
21. 42h and 66d

22. 47h and 71d

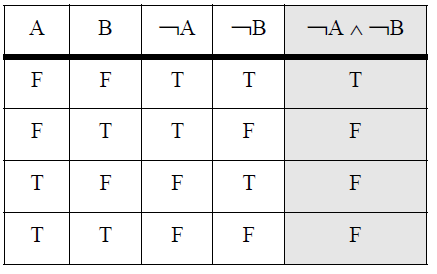
23. 229 − 1, or 6.8056473384187692692674921486353 X 1038

24. 286 − 1, or 77371252455336267181195263

25. Truth table:



26. Truth table: (last column is the same as #25)



27. It requires 24 (16) rows.

28. 2 bits, producing the following values: 00, 01, 10, 11

1.7.2 Algorithm Workbench

1. Code example (C++)

int toInt32(string s) {

int num = 0;

for(int i = 0; s[i] >= '0' && s[i] <= '1'; i++) {

num = num \* 2 + s[i]-'0';

}

return num;

}

2. Code example (C++)

int hexStrToInt32(string s) {

int num = 0;

for(int i = 0; ; i++) {

if( s[i] >= '0' && s[i] <= '9' )

num = num \* 16 + s[i]-'0';

else if( s[i] >= 'A' && s[i] <= 'F' )

num = num \* 16 + (s[i]-'A'+10);

else

break;

}

return num;

}

3. Code example (C++)

string intToBinStr( int n ) {

vector<int> stack;

do {

int quotient = n / 2;

int remainder = n % 2;

stack.push\_back(remainder);

n = quotient;

} while( n > 0 );

string s;

while( stack.size() > 0 ) {

s += (stack.back() + '0');

stack.pop\_back();

}

return s;

}

4. Code example (C++)

string intToHexStr( int n ) {

vector<int> stack;

do {

int quotient = n / 16;

int remainder = n % 16;

stack.push\_back(remainder);

n = quotient;

} while( n > 0 );

string s;

while( stack.size() > 0 ) {

int d = stack.back();

if( d >= 0 && d <= 9 )

s += (stack.back() + '0');

else // probably a hex digit

s += (stack.back() - 10 + 'A');

stack.pop\_back();

}

return s;

}

5. Code example (C++)

string addDigitStrings( string s1, string s2, int base ) {

string sumStr;

int carry = 0;

for(int i = s1.size() - 1; i >= 0; i--) {

int dval = (s1[i] - '0') + (s2[i] - '0') + carry;

carry = 0;

if( dval > (base - 1) ) {

carry = 1;

dval = dval % base;

}

sumStr.insert(sumStr.begin(), (dval + '0'));

}

if( carry == 1 )

sumStr.insert( sumStr.begin(), 1 + '0');

return sumStr;

}

Chapter 2

2.8.1 Short Answer Questions

1. EBP

2. Choose 4 from: Carry, Zero, Sign, Direction, Aux Carry, Overflow, Parity.

3. Carry flag

4. Overflow flag

5. True

6. Sign flag

7. Floating-point unit

8. 80 bits

9. True

10. False

11. True

12. False

13. True

14. False

14. False

15. False

16. True

17. False

18. True

19. False

20. False

21. True

22. True

23. False

24. False

25. Hardware, BIOS, and OS

26. It gives them more precise control of hardware, and execution is faster.

Chapter 3

3.9.1 Short Answer Questions

1. ADD, SUB, MOV

2. A calling convention determines how parameters are passed to subroutines, and how the stack is restored after the subroutine call.

3. By subtracting a value from the stack pointer register.

4. Assembler means the program that translates your source code. A more correct term is "assembly language".

5. Little endian places the least significant bit in position 0, on the right side of the number. Big endian does the opposite.

6. An integer literal, such as 35, has no direct meaning to someone reading the program's source code. Instead, a symbolic constant such as MIN\_CAPACITY can be assigned an integer value, and is self-documenting.

7. A source file is given as input to the assembler. A listing file has additional text that will not assemble. It is a file that iscreated by the assembler.

8. Data labels exist in the data segment as variable offsets. Code labels are in the code segment, and are offsets for transfer of control instructions.

9. True

10. False (this notation is used in C, but not in assembly language).

11. False

12. True

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

14. True

15. True

16. Code example:

COMMENT !

this is the first comment line

this is the second comment line

!

17. You do not use numeric addresses (offsets) for variables because the addresses would change if new variables were inserted before the existing ones.

3.9.2 Algorithm Workbench

1. Code example:

one = 25

two = 11001b

three = 31o

four = 19h

2. Yes, it can have multiple code and data segments.

3. Storing the value 01020304h

myVal LABEL DWORD

BYTE 04h,03h,02h,01h

4. Yes, you can. The assembler does not check the number's sign.

5. Code example. The two instructions have different opcodes.

add eax,5

add edx,5

6. Little endian order: ABh, 89h, 67h, 45h

7. **myArray DWORD 120 DUP(?)**

8. **firstFive BYTE "ABCDE"**

9. smallVal SDWORD 80000000h ; -32,768

10. wArray WORD 1000h, 2000h, 3000h

11. favColor BYTE "blue",0

12. dArray DWORD 50 DUP(?)

13. msg BYTE 500 DUP("TEST")

14. bArray BYTE 20 DUP(0)

Chapter 4

4.9.1 Short Answer

1. a. edx = FFFF8002h b. edx = 00004321h

2. eax = 10020000h

3. eax = 3002FFFFh

4. eax = 10020001h

5. Parity Even (1)

6. eax = FFFFFFFFh, SF = 1 (the result is negative)

7. −1 + 130 = 129, which is outside the range of a signed positive byte. Therefore, the Overflow flag is set.

8. rax = 0000000044445555h

9. rax = FFFFFFFF84326732h

10. eax = 00035678h

11. eax = 12375678h

12. No.

13. Yes.

14. Yes, for example:

mov al,−128

neg al ; OF = 1

15. No.

16. (a) not valid, (b) valid, (c) not valid, (d) not valid, (e) not valid, (f) not valid, (g) valid, (h) not valid

17. (a) FCh (b) 01h

18. (a) 1000h, (b) 3000h, (c) FFF0h, (d) 4000h

19. (a) 00000001h, (b) 00001000h, (c) 00000002h, (d) FFFFFFFCh

4.9.2 Algorithm Workbench

1. Code example:

mov ax,word ptr three

mov bx,word ptr three+2

mov three,bx

mov word ptr three+2,ax

2. Code example: convert al,bl,cl,dl to bl,cl,dl,al

xchg al,bl

xchg al,cl

xchg al,dl

3. Code example:

mov al,01110101

add al,0 ; PF = 0 (odd)

4. Code example:

mov al,-127

add al,-1 ; OF = 1

5. Code example (set Zero and Carry)

mov al,0FFh

add al,1

6. Code example:

mov al,3

sub al,4

7. Code example: AX = (val2 + BX) − val4

mov ax,val2

add ax,bx

sub ax,val4

8. Code example:

mov al,80h

add al,80h

9. Code example:

mov ax,val2

neg ax

add ax,bx

sub ax,val4

10. Setting the Carry and Overflow flags at the same time:

mov al,80h

add al,80h

11. Setting the Zero flag after INC and DEC to indicate unsigned overflow:

mov al,0FFh

inc al

jz overflow\_occurred

mov bl,1

dec bl

jz overflow\_occurred

12. Data directive:

.data

ALIGN 2

myBytes BYTE 10h, 20h, 30h, 40h

etc.

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

14. Code:

mov dx, WORD PTR myBytes

15. Code:

mov al, BYTE PTR myWords+1

16. Code:

mov eax, DWORD PTR myBytes

17. Data directive:

myWordsD LABEL DWORD

myWords WORD 3 DUP(?),2000h

.code

mov eax,myWordsD

18. Data directive:

myBytesW LABEL WORD

myBytes BYTE 10h,20h,30h,40h

.code

mov ax,myBytesW

Chapter 5

5.7.1 Short Answer

1. pusha

2. pushf

3. popf

4. Because you might not want to push all the general-purpose registers when eax is being used to pass a return value back to the subroutine's caller.

5. Code example (32-bit mode):

sub esp,4

mov [esp],eax

6. True

7. False

8. True

9. False

10. Yes, the pointer is in the ESI register

11. True

12. False

13. False

14. The following statements would have to be modified:

mov [esi],eax becomes --> mov [esi],ax

add esi,4 becomes --> add esi,2

15. EAX = 5

16. (d)

17. (c)

18. (c)

19. (a)

20. The array will contain 10, 20, 30, 40

5.7.2 Algorithm Workbench

1. Exchange registers using push and pop:

push ebx

push eax

pop ebx

pop eax

2. Modify a subroutine's return address:

pop eax ; get the return address

add eax,3 ; add 3

push eax ; put it back on the stack

ret

3. Create and assign local variables:

sub esp,8 ; space for two variables

mov [esp],1000h

mov [esp+4],2000h

4. Copy an array element backwards

mov edi,esi

dec edi

mov edx,array[esi\*4]

mov array[edi\*4],edx

5. Display a subroutine's return address

mov eax,[esp]

call WriteHex

Chapter 6

6.10.1 Short Answer

1. BX = 006Bh

2. BX = 092h

3. BX = 064BBh

4. BX = A857h

5. EBX = BFAFF69Fh

6. RBX = 0000000050509B64h

7. AL = 2Dh, 48h, 6Fh, A3h

8. AL = 85h, 34h, BFh, AEh

9. a. CF= 0 ZF= 0 SF=0

b. CF= 0 ZF= 0 SF=0

c. CF= 1 ZF= 0 SF=1

10. JECX

11. JA and JNBE jump to the destination if ZF = 0 and CF = 0.

12. EDX = 1

13. EDX = 1

14. EDX = 0

15. True

16. True

17. 0FFFFFFFFFFFFFF80h

18. 0FFFFFFFFFF808080h

19. 0000000080808080h

6.10.2 Algorithm Workbench

1. and al,0Fh

2. Calculate parity of a doubleword:

.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

3. Generate a bit string in EAX that represents members in SetX that are not members of SetY:

.data

SetX DWORD ?

SetY DWORD ?

.code

mov eax,SetX

xor eax,SetY ; remove all SetY from SetX

4. Jump to label L1 when the unsigned integer in DX is less than or equal to the integer in CX:

cmp dx,cx

jbe L1

L1:

5. Write instructions that jump to label L2 when the signed integer in AX is greater than the

integer in CX:

cmp ax,cx

jg L2

L2:

6. First clear bits 0 and 1 in AL. Then, if the destination operand is equal to zero, the code should jump to label L3. Otherwise, it should jump to label L4:

and al,00000011b

jz L3

jmp L4

mov edx,0

L3:

L4:

7. Code example:

cmp val1,cx

jna L1

cmp cx,dx

jna L1

mov X,1

jmp next

L1: mov X,2

next:

8. Code example:

cmp bx,cx

ja L1

cmp bx,val1

ja L1

mov X,2

jmp next

L1: mov X,1

next:

9. Code example:

cmp bx,cx ; bx > cx?

jna L1 ; no: try condition after OR

cmp bx,dx ; yes: is bx > dx?

jna L1 ; no: try condition after OR

jmp L2 ; yes: set X to 1

;-----------------OR(dx > ax) ------------------------

L1: cmp dx,ax ; dx > ax?

jna L3 ; no: set X to 2

L2: mov X,1 ; yes:set X to 1

jmp next ; and quit

L3: mov X,2 ; set X to 2

next:

10. Code example:

Exercise10Test proc

; use these registers to hold the logical variables:

mov eax,4 ; A

mov ebx,5 ; B

mov edx,10 ; N

call Exercise10Test

ret

Exercise10Test endp

Exercise10 proc

whileloop:

cmp edx,0

jle endwhile

cmp edx,3 ; if N != 3

je elselabel

; check N < eax OR N > ebx

cmp edx,eax ; N < A?

jl orlabel ; if true, jump

cmp edx,ebx ; or N > B?

jg orlabel ; if true, jump

jmp elselabel

orlabel:

sub edx,2

jmp whileloop

elselabel:

sub edx,1

jmp whileloop

endwhile:

ret

Exercise10 endp

Chapter 7

7.10.1 Short Answer

1. (a) 6Ah (b) EAh (c) FDh (d) A9h

2. (a) 9Ah (b) 6Ah (c) 0A9h (d) 3Ah

3. DX = 0002h, AX = 2200h

4. AX = 0306h

5. EDX = 0, EAX = 00012340h

6. The DIV will cause a divide overflow, so the values of AX and DX cannot be determined.

7. DX = 0016h

8. The DIV will cause a divide overflow.

9. In correcting this example, it is easiest to reduce the number of instructions. You can use a single register (ESI) to index into all three variables. ESI should be set to zero before the loop because the integers are stored in little endian order with their low-order bytes occurring first:

mov ecx,8 ; loop counter

mov esi,0 ; use the same index register

clc ; clear Carry flag

top:

mov al,byte ptr val1[esi] ; get first number

sbb al,byte ptr val2[esi] ; subtract second

mov byte ptr result[esi],al ; store the result

inc esi ; move to next pair

loop top

Of course, you could easily reduce the number of loop iterations by adding doublewords rather than bytes.

10. (Shift each bit two positions to the left) = 4080C10140000h

Shown in binary:

0001 0000 0010 0000 0011 0000 0100 0000 0101 0000 0000 0000 0000 (before)

0100 0000 1000 0000 1100 0001 0000 0001 0100 0000 0000 0000 0000 (after)

7.10.2 Algorithm Workbench

1. Code example:

shl eax,16

sar eax,16

2. Code example:

shr al,1 ; shift AL into Carry flag

jnc next ; Carry flag set?

or al,80h ; yes: set highest bit

next: ; no: do nothing

3. shl eax,4

4. shr ebx,2

5. ror dl,4 (or: rol dl,4)

6. shld dx,ax,1

7. This problem requires us to start with the high-order byte and work our way down to the lowest byte:

byteArray BYTE 81h,20h,33h

.code

shr byteArray+2,1

rcr byteArray+1,1

rcr byteArray,1

8. This problem requires us to start with the low-order word and work our way up to the highest word:

wordArray WORD 810Dh,0C064h,93ABh

.code

shl wordArray,1

rcl wordArray+2,1

rcl wordArray+4,1

9. Code example:

mov ax,3

mov bx,-5

imul bx

mov val1,ax ; product

// alternative solution:

mov al,3

mov bl,-5

imul bl

mov val1,ax ; product

10. Code example:

mov ax,-276

cwd ; sign-extend AX into DX

mov bx,10

idiv bx

mov val1,ax ; quotient

11. Implement the unsigned expression: val1 = (val2 \* val3) / (val4 – 3).

mov eax,val2

mul val3

mov ebx,val4

sub ebx,3

div ebx

mov val1,eax

(You can substitute any 32-bit general-purpose register for EBX in this example.)

12. Implement the signed expression: val1 = (val2 / val3) \* (val1 + val2).

mov eax,val2

cdq ; extend EAX into EDX

idiv val3 ; EAX = quotient

mov ebx,val1

add ebx,val2

imul ebx

mov val1,eax ; lower 32 bits of product

(You can substitute any 32-bit general-purpose register for EBX in this example.)

13. Code example (displays binary value in AX):

out16 proc

aam

or ax,3030h

push eax

mov al,ah

call WriteChar

pop eax

call WriteChar

ret

out16 endp

14. After AAA, AX would equal 0108h. Intel says: First, if the lower digit of AL is greater than 9 or the AuxCarry flag is set, add 6 to AL and add 1 to AH. Then in all cases, AND AL with 0Fh. Here is their pseudocode:

IF ((AL AND 0FH) > 9) OR (AuxCarry = 1) THEN

add 6 to AL

add 1 to AH

END IF

AND AL with 0FH

15. Calculate x = n mod y, given n and y, where y is a power of 2:

.data

dividend DWORD 1000

divisor DWORD 32 ; must be a power of 2

answer DWORD ?

.code

mov edx,divisor ; create a bit mask

sub edx,1

mov eax,dividend

and eax,edx ; clear high bits, low bits contain mod value

mov answer,eax

16. Calculate absolute value of EAX without using a conditional jump:

mov edx,eax ; create a bit mask

sar edx,31

add eax,edx

xor eax,edx

Chapter 8

8.10.1 Short Answer

1. Code example:

mov esp,ebp

pop ebp

2. EAX

3. It passes an integer constant to the RET instruction. This constant is added to the stack pointer right after the RET instruction has popped the procedure’s return address off the stack.

4. LEA can return the offset of an indirect operand; it is particularly useful for obtaining the offset of a stack parameter.

5. Four bytes

6. The C calling convention allows for variable-length parameter lists.

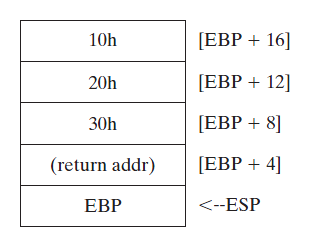
7. False

8. False

9. True (because the immediate value will be interpreted as an address)

8.10.2 Algorithm Workbench

1. Stack frame:



2. Code example:

AddThree PROC

; modeled after the AddTwo procedure

push ebp

mov ebp,esp

mov eax,[ebp + 16] ; 10h

add eax,[ebp + 12] ; 20h

add eax,[ebp + 8] ; 30h

pop ebp

ret 12

AddThree ENDP

3. Declaration: LOCAL pArray:PTR DWORD

4. Declaration: LOCAL buffer[20]:BYTE

5. Declaration: LOCAL pwArray:PTR WORD

6. Declaration: LOCAL myByte:SBYTE

7. Declaration: LOCAL myArray[20]:DWORD

8. Code example:

SetColor PROC USES eax,

forecolor:BYTE, backcolor:BYTE

movzx eax,backcolor

shl eax,4

add al,forecolor

call SetTextColor

ret

SetColor ENDP

9. Code example:

WriteColorChar PROC USES eax,

char:BYTE,forecolor:BYTE, backcolor:BYTE,

INVOKE SetColor, forecolor, backcolor

mov al,char

call WriteChar

ret

WriteColorChar ENDP

10. Code example:

DumpMemory PROC USES esi ebx ecx,

address:DWORD, ; starting address

units:DWORD, ; number of units

unitType:DWORD ; unit size

mov esi,address

mov ecx,units

mov ebx,unitType

call DumpMem

ret

DumpMemory ENDP

11. Code example:

MultArray PROC USES esi ebx ecx,

array1:PTR DWORD, array2:PTR DWORD,

count:DWORD

MultArray PROTO,

array1:PTR DWORD, array2:PTR DWORD,

count:DWORD

Chapter 9

9.9.1 Short Answer

1. 1 (set)

2. 2 is added to the index register

3. Regardless of which operands are used, CMPS still compares the contents of memory pointed to by ESI to the memory pointed to by EDI.

4. 1 byte beyond the matching character.

5. REPNE (REPNZ).

6. 1 (set)

7. JNE is used to exit the loop and insert a null byte into the string when no more characters are to be trimmed.

8. The digit is unchanged.

9. REPNE (REPNZ).

10. The length would be (EDIfinal − EDIinitial) − 1.

11. The maximum comparisons for 1,024 elements is 11.

12. The Direction flag is cleared so that the STOSD instruction will automatically increment the EDI register. Instead, if the flag were set, EDI would decrement and move backwards through the array.

13. EDX and EDI were already compared.

14. Change each JMP L4 instruction to JMP L1.

9.9.2 Algorithm Workbench

1. [ebx + esi]

2. array[ebx + esi]

3. Code example:

mov esi,2 ; row

mov edi,3 ; column

mov eax,[esi\*16 + edi\*4]

4. CMPSW example:

mov ecx,count

mov esi,offset sourcew

mov edi,offset targetw

cld

repe cmpsw

5. SCASW example:

cld

mov ecx,4

mov edi,offset wordArray

repne scasw

sub edi,4 ; adjust the offset

mov eax,edi

6. Str\_compare example:

.data

str1 byte "ABCDE",0 ; larger string

str2 byte "ABCCD",0

.code

invoke Str\_compare, offset str1, offset str2

ja str1bigger

mov edx,offset str2

jmp print

str1bigger:

mov edx,offset str1

print:

call WriteString

call Crlf

7. Str\_trim example:

.data

prob7string byte "ABCD@@@@",0

.code

invoke Str\_trim, offset prob7string, '@'

8. Str\_lcase example (converts string to lower case):

Str\_lcase PROC USES eax esi,

pString:PTR BYTE

mov esi,pString

L1:

mov al,[esi] ; get char

cmp al,0 ; end of string?

je L3 ; yes: quit

cmp al,'A' ; below "A"?

jb L2

cmp al,'Z' ; above "Z"?

ja L2

or BYTE PTR [esi],00100000b ; convert to lower case

L2:inc esi ; next char

jmp L1

L3: ret

Str\_lcase ENDP

9. 64-bit Str\_trim procedure:

;-----------------------------------------------------------

Str\_trim PROC uses rax rcx rdi

; Removes all occurences of a given character from

; the end of a string.

; Receives: RCX points to the string, AL contains the trim character

; Returns: nothing

;-----------------------------------------------------------

.data

strtrimchar byte ?

.code

mov strtrimchar,al

mov rdi,rcx ; save pointer to string

call Str\_length ; puts length in RAX

cmp rax,0 ; length zero?

je L3 ; yes: exit now

mov rcx,rax ; no: RCX = string length

dec rax

add rdi,rax ; point to null byte at end

L1: mov al,[rdi] ; get a character

cmp al,strtrimchar ; character to be trimmed?

jne L2 ; no: insert null byte

dec rdi ; yes: keep backing up

loop L1 ; until beginning reached

L2: mov BYTE PTR [rdi+1],0 ; insert a null byte

L3: ret

Str\_trim ENDP

10. Base-index operand in 64-bit mode:

array[rsi\*TYPE array]

11. Two-dimensional array indexing, 32-bit mode:

mov eax,table[ebx + edi\*TYPE myArray]

12. Two-dimensional array indexing, 64-bit mode:

mov rax,table[rbx + rdi\*TYPE myArray]

Chapter 10

10.7.1 Short Answer

1. Structures are essential whenever you need to pass a large amount of data between procedures. One variable can be used to hold all the data.

2. Answers:

a. Yes.

b. No.

c. Yes.

d. Yes.

e. No.

3. False.

4. To permit the use of labels in a macro that is invoked more than once by the same program.

5. ECHO (also, the %OUT operator, which is shown later in the chapter).

6. ENDIF.

7. List of relational operators:

LT Less than

GT Greater than

EQ Equal to

NE Not equal to

LE Less than or equal to

GE Greater than or equal to

8. The substitution (&) operator resolves ambiguous references to parameter names within a macro.

9. The literal-character operator (!) forces the preprocessor to treat a predefined operator as an ordinary character.

10. The expansion operator (%) expands text macros or converts constant expressions into their text representations.

10.7.2 Algorithm Workbench

1. Structure definition:

MyStruct STRUCT

field1 WORD ?

field2 DWORD 20 DUP(?)

MyStruct ENDS

2. Code example:

.data

time SYSTEMTIME <>

.code

mov ax,time.wHour

3. Code example:

myShape Triangle < <0,0>, <5,0>, <7,6> >

4. Declare and initialize an array of Triangle structures:

.data

ARRAY\_SIZE = 5

triangles Triangle ARRAY\_SIZE DUP(<>)

.code

mov ecx,ARRAY\_SIZE

mov esi,0

L1: mov eax,11

call RandomRange

mov triangles[esi].Vertex1.X, ax

mov eax,11

call RandomRange

mov triangles[esi].Vertex1.Y, ax

add esi,TYPE Triangle

loop L1

5. Code example:

mPrintChar MACRO char,count

LOCAL temp

.data

temp BYTE count DUP(&char),0

.code

push edx

mov edx,OFFSET temp

call WriteString

pop edx

ENDM

6. Code example:

mGenRandom MACRO n

mov eax,n

call RandomRange

ENDM

7. mPromptInteger:

mPromptInteger MACRO prompt,returnVal

mWriteprompt

call ReadInt

mov returnVal,eax

ENDM

8. Code example:

mWriteAt MACRO X,Y,literal

mGotoxy X,Y

mWrite literal

ENDM

9. Code example:

mWriteStr namePrompt

1 push edx

1 mov edx,OFFSET namePrompt

1 call WriteString

1 pop edx

10. Code example:

mReadStr customerName

1 push ecx

1 push edx

1 mov edx,OFFSET customerName

1 mov ecx,(SIZEOF customerName) - 1

1 call ReadString

1 pop edx

1 pop ecx

11. Code example:

;------------------------------------------------

mDumpMemx MACRO varName

;

; Displays a variable in hexadecimal, using the

; variable's attributes to determine the number

; of units and unit size.

;------------------------------------------------

push ebx

push ecx

push esi

mov esi,OFFSET varName

mov ecx,LENGTHOF varName

mov ebx,TYPE varName

call DumpMem

pop esi

pop ecx

pop ebx

ENDM

; Sample calls:

.data

array1 BYTE 10h,20h,30h,40h,50h

array2 WORD 10h,20h,30h,40h,50h

array3 DWORD 10h,20h,30h,40h,50h

.code

mDumpMemx array1

mDumpMemx array2

mDumpMemx array3

12. Macro using a default argument initializer:

mWriteLn MACRO text:=<" ">

mWrite text

call Crlf

ENDM

13. Macro using IF, ELSE, ENDIF:

mCopyWord MACRO intVal

IF (TYPE intVal) EQ 2

mov ax,intVal

ELSE

ECHO Invalid operand size

ENDIF

ENDM

14. Macro using IF to check value of a parameter

mCheck MACRO Z

IF Z LT 0

ECHO \*\*\*\* Operand Z is invalid \*\*\*\*

ENDIF

ENDM

15. Macro uses the & operator when parameter is embedded in a string:

CreateString MACRO strVal

.data

temp BYTE "Var&strVal",0

.code

ENDM

16. Source code generated by the mLocate macro:

mLocate -2,20

;(no code generated because xval < 0)

mLocate 10,20

1 mov bx,0

1 mov ah,2

1 mov dh,20

1 mov dl,10

1 int 10h

mLocate col,row

1 mov bx,0

1 mov ah,2

1 mov dh,row

1 mov dl,col

1 int 10h

Chapter 11

11.7 Review Questions

11.7.1 Short Answer

1. BOOL = byte, COLORREF = DWORD, HANDLE = DWORD, LPSTR = PTR BYTE, WPARAM = DWORD.

2. GetStdHandle.

3. ReadConsole.

4. The COORD structure contains X and Y screen coordinates in character measurements.

5. SetFilePointer.

6. SetConsoleTitle.

7. SetConsoleScreenBufferSize.

8. SetConsoleCursorInfo.

9. SetConsoleTextAttribute.

10. WriteConsoleOutputAttribute.

11. Sleep.

12. (A program that calls CreatewindowEx is shown in Section 11.2.6.) The prototype for CreateWindowEx is located in the GraphWin.inc file:

CreateWindowEx PROTO,

classexWinStyle:DWORD,

className:PTR BYTE,

winName:PTR BYTE,

winStyle:DWORD,

X:DWORD,

Y:DWORD,

rWidth:DWORD,

rHeight:DWORD,

hWndParent:DWORD,

hMenu:DWORD,

hInstance:DWORD,

lpParam:DWORD

The fourth parameter, winStyle, determines the window’s style characteristics. In the WinApp.asm program in Section 11.2.6, when we call CreateWindowEx, we pass it a combination of predefined style constants:

MAIN\_WINDOW\_STYLE = WS\_VISIBLE + WS\_DLGFRAME + WS\_CAPTION

+ WS\_BORDER + WS\_SYSMENU + WS\_MAXIMIZEBOX + WS\_MINIMIZEBOX

+ WS\_THICKFRAME

The window described here will be visible, and it will have a dialog box frame, a caption bar, a border, a system menu, a maximize icon, a minimize icon, and a thick surrounding frame.

13. Choose any two of the following (from GraphWin.inc):

MB\_OK, MB\_OKCANCEL, MB\_ABORTRETRYIGNORE, MB\_YESNOCANCEL, MB\_YESNO,

MB\_RETRYCANCEL, MB\_CANCELTRYCONTINUE

14. Icon constants (choose any two):

MB\_ICONHAND, MB\_ICONQUESTION, MB\_ICONEXCLAMATION, MB\_ICONASTERISK

15. Tasks performed by WinMain (choose any three):

* Get a handle to the current program.
* Load the program’s icon and mouse cursor.
* Register the program’s main window class and identify the procedure that will process event messages for the window.
* Create the main window.
* Show and update the main window.
* Begin a loop that receives and dispatches messages.

16. The WinProc procedure receives and processes all event messages relating to a window. It decodes each message, and if the message is recognized, carries out application-oriented (or application-specific) tasks relating to the message.

17. The following messages are processed:

• WM\_LBUTTONDOWN, generated when the user presses the left mouse button.

• WM\_CREATE, indicates that the main window was just created.

• WM\_CLOSE, indicates that the application’s main window is about to close.

18. The ErrorHandler procedure, which is optional, is called if the system reports an error during the registration and creation of the program’s main window.

19. The message box is shown before the application’s main window appears.

20. The message box appears before the main window closes.

21. A linear address is a 32-bit integer ranging between 0 and FFFFFFFFh, which refers to a memory location. The linear address may also be the physical address of the target data if a feature called paging is disabled.

22. When paging is enabled, the processor translates each 32-bit linear address into a 32-bit physical address. A linear address is divided into three fields: a pointer to a page directory entry, a pointer to a page table entry, and an offset into a page frame.

23. The linear address is automatically a 32-bit physical memory address.

24. Paging makes it possible for a computer to run a combination of programs that would not otherwise fit into memory. The processor does this by initially loading only part of a program in memory while keeping the remaining arts on disk.

25. The LDTR register.

26. The GDTR register.

27. One.

28. Many (each task or program has its own local descriptor table).

29. Choose any four from the following list: base address, privilege level, segment type, segment present flag, granularity flag, segment limit.

30. Page Directory, Page Table, and Page (page frame).

31. The Table field of a linear address (see Figure 11-4).

32. The Offset field of a linear address (see Figure 11-4).

11.7.1 Algorithm Workbench

1. Example from the ReadConsole.asm program in Section 11.1.4:

INVOKE ReadConsole, stdInHandle, ADDR buffer,

BufSize - 2, ADDR bytesRead, 0

2. Example from the Console1.asm program in Section 11.1.5:

INVOKE WriteConsole,

consoleHandle, ; console output handle

ADDR message, ; string pointer

messageSize, ; string length

ADDR bytesWritten, ; returns num bytes written

0 ; not used

3. Calling CreateFile when reading an input file:

INVOKE CreateFile,

ADDR filename, ; ptr to filename

GENERIC\_READ, ; access mode

DO\_NOT\_SHARE, ; share mode

NULL, ; ptr to security attributes

OPEN\_EXISTING, ; file creation options

FILE\_ATTRIBUTE\_NORMAL, ; file attributes

0 ; handle to template file

4. Calling CreateFile to create a new file:

INVOKE CreateFile,

ADDR filename,

GENERIC\_WRITE,

DO\_NOT\_SHARE,

NULL,

CREATE\_ALWAYS,

FILE\_ATTRIBUTE\_NORMAL,

0

5. Calling ReadFile:

INVOKE ReadFile, ; read file into buffer

fileHandle,

ADDR buffer,

bufSize,

ADDR byteCount,

0

6. Calling WriteFile:

INVOKE WriteFile, ; write text to file

fileHandle, ; file handle

ADDR buffer, ; buffer pointer

bufSize, ; number of bytes to write

ADDR bytesWritten, ; number of bytes written

0 ; overlapped execution flag

7. Calling MessageBox:

INVOKE MessageBox, hMainWnd, ADDR GreetText,

ADDR GreetTitle, MB\_OK

Chapter 12

12.6.1 Short Answer

1. 1101.01101 = 13/1 + 1/4 + 1/8 + 1/32

2. 0.2 generates an infinitely repeating bit pattern.

3. 11011.01011 = 1.101101011 X 24

4. 0000100111101.1 = 1.001111011 X 2−8

5. Quiet NaN and Signaling NaN

6. REAL10 80 bits

7. It pops ST(0) off the stack.

8. FCHS.

9. None, m32fp, m64fp, stack register

10. FISUB converts the source operand from integer to floating-point.

11. FCOM, or FCOMP

12. Code example:

fnstsw ax

lahf

13. FILD

14. RC field

12.6.2 Algorithm Workbench

1. +1110.011 = 1.110011 X 23, so the encoding is 0 10000010 11001100000000000000000

2. 5/8 = 0.101 binary

3. 17/32 = 0.10001 binary

4. +10.75 = +1010.11 = +1.01011 X 23, encoded as 0 10000010 01011000000000000000000

5. −76.0625 = −01001100.0001 = −1.0011000001 X 2−6, encoded as:

1 10000101 00110000010000000000000

6. Code example:

fnstsw ax

lahf

7. 1.010101101

8. 1.010101101 rounded to nearest even becomes 1.010101110.

9. Assembly language code:

.data

B REAL8 7.8

M REAL8 3.6

N REAL8 7.1

P REAL8 ?

.code

fld M

fchs

fld N

fadd B

fmul

fst P

10. Assembly language code:

.data

B DWORD 7

N REAL8 7.1

P REAL8 ?

.code

fld N

fsqrt

fiadd B

fst P

11. (a) 8E (b) 8A (c) 8A (d) 8B (e) A0 (f) 8B

12. (a) 06 (b) 56 (c) 1D (d) 55 (e) 84 (f ) 81

13. Machine language bytes:

a. 8E D8

b. A0 00 00

c. 8B 0E 01 00

d. BA 00 00

e. B2 02

f. BB 00 10

Chapter 13

13.7 Review Questions

1. The memory model determines whether near or far calls are made. A near call pushes only the 16-bit offset of the return address on the stack. A far call pushes a 32-bit segment/offset address on the stack.

2. C and C++ are case sensitive, so they will only execute calls to procedures that are named in the same fashion.

3. Yes, many languages specify that EBP (BP), ESI (SI), and EDI (DI) must be preserved across procedure calls.

4. Yes.

5. No.

6. No.

7. A program bug might result because the \_\_fastcall convention allows the compiler to use general-purpose registers as temporary variables.

8. Use the LEA instruction.

9. The LENGTH operator returns the number of elements in the array specified by the DUP operator. For example, the value placed in EAX by the LENGTH operator is 20:

myArray DWORD 20 DUP(?), 10, 20, 30

.code

mov eax,LENGTH myArray ; 20

10. The SIZE operator returns the product of TYPE (4) \* LENGTH.

11. **printf PROTO C, pString:PTR BYTE, args:VARARG.**

12. X will be pushed last.

13. To prevent the decoration (altering) of external procedure names by the C++ compiler. Name decoration (also called name mangling) is done by programming languages that permit function overloading, which permits multiple functions to have the same name.

14. If name decoration is in effect, an external function name generated by the C++ compiler will not be the same as the name of the called procedure written in assembly language. Understandably, the assembler does not have any knowledge of the name decoration rules used by C++ compilers.

15. Virtually no changes at all, showing that array subscripts can be just as efficient as pointers when manipulating arrays.