APPENDIX D

Answers to Selected Even-Numbered Questions and Problems

CHAPTER 1

2. Herman Hollerith
4. Konrad Zuse
6. ENIAC
8. Augusta Ada Byron
10. A machine that stores the instructions of a program in the memory system.
12. 200 million
14. 16M bytes
16. 1993
18. 2000
20. Millions of instructions per second
22. A binary bit stores a 1 or a 0.
24. 1024K
26. 1024
28. System area and transient program area
30. 640K
32. 1M
34. 80386, 80486, Pentium, Pentium Pro, PII, PIII, P4, and Core2
36. The basic I/O system
38. The XT was used with the 8088 and 8086 and beginning with the 80286, the AT became the name of the system.
40. 8-bit and 16-bit
42. The advanced graphics port is designed to support video cards.
44. The serial ATA interface is designed to support disk drive memory.
46. 64K
48. See Figure 1–6.
50. Address, data, and control buses.
52. $\overline{\text{MRDC}}$
54. Memory read operation
56. (a) 8-bit signed number (b) 16-bit signed number (c) 32-bit signed number (d) 32-bit floating-point number (e) 64-bit floating-point number
58. (a) 156.625 (b) 18.375 (c) 4087.109375 (d) 83.578125 (e) 58.90625
60. (a) 10111_2, 27_8, and 17_{16} (b) 1101011_2, 153_8, and 6B (c) 10011010110_2, 2326_8, and $4D6_{16}$ (d) 1011100_2, 134_8, and $5C_{16}$ (e) 10101101_2, 255_8, and AD
62. (a) 0010 0011 (b) 1010 1101 0100 (c) 0011 0100 . 1010 1101 (d) 1011 1101 0011 0010 (e) 0010 0011 0100 . 0011
64. (a) 0111 0111 (b) 1010 0101 (c) 1000 1000 (d) 0111 1111
66. Byte is an 8-bit binary number, word is a 16-bit binary number, doubleword is a 32-bit binary number.
68. Enter is a 0DH and it is used to return the cursor/print head to the left margin of the screen or page of paper.
70. LINE1 DB ‘What time is it?’
72. (a) 0000 0011 1110 1000 (b) 1111 1111 1000 1000 (c) 0000 0011 0010 0000 (d) 1111 0011 0111 0100
74. char Fred1 = –34
76. Little endian numbers are stored so the least significant portion is in the lowest numbered memory location and big endian numbers are stored so the most significant part is stored in the lowest numbered memory location.
78. (a) packed = 00000001 00000010 and unpacked 00000001 00000000 00000010 (b) packed = 01000100 and unpacked 00000100 00000100 (c) packed = 00000011 00000001 and unpacked 00000011 00000000 00000001 (d) packed = 00010000 00000000 and unpacked 00000001 00000000 00000000 00000000
80. (a) 89 (b) 9 (c) 32 (d) 1
82. (a) +3.5 (b) −1.0 (c) +12.5

CHAPTER 2

2. 16
4. EBX
6. Holds the offset address of the next step in the program.
8. No, if you add +1 and –1 you have zero, which is a valid number.
10. The I-flag.
12. The segment register addresses the lowest address in a 64K memory segment.
14. (a) 12000H (b) 21000H (c) 24A00H (d) 25000H (e) 3F12DH
16. DI
18. SS plus either SP or ESP
20. (a) 12000H (b) 21002H (c) 26200H (d) A1000H (e) 2CA00H
22. All 16M bytes
24. The segment register is a selector that selects the descriptor from a descriptor table. It also sets privilege level of the request and chooses either the global or local table.
26. A00000H–A01000H
28. 00280000H–00290FFFH
30. 3
32. 64K
34.

0000 0011	1101 0000
1001 0010	0000 0000
0000 0000	0000 0000
0010 1111	1111 1111

36. Through a descriptor stored in the global table
38. The program invisible registers are the cache portions of the segment registers and also the GDTR, LDTR, and IDTR registers.
40. 4K
42. 1024
44. Entry zero or the first entry
46. The TLB caches the most recent memory accesses through the paging mechanism.
50. 1T

CHAPTER 3

2. AL, AH, BL, BH, CL, CH, DL, and DH
4. EAX, EBX, ECX, EDX, ESP, EBP, EDI, and ESI
6. CS, DS, ES, SS, FS, and GS
8. You may not specify mixed register sizes.
10. (a) MOV AL,12H (b) MOV AX,123AH
(c) MOV CL,0CDH (d) MOV RAX,1000H
(e) MOV EBX,1200A2H
12. Selects an assembly language programming model that contains a single segment that compiles as a .COM program.
14. A label is a symbolic memory address.
16. A label may begin with a letter and some special characters, but not with a number.
18. The .TINY model creates a .COM program.
20. A displacement is a distance and in MOV DS:[2000H],AL the displacement of 2000H is added to the contents of DS times 10H to form the memory address.
22. (a) 3234H (b) 2300H (c) 2400H
24. MOV BYTE PTR [2000H],6
26. MOV DWORD PTR DATA1, 5
28. The MOV BX,DATA instruction copies the word from memory location data into the BX register where the MOV BX,OFFSET DATA instruction copies the offset address of DATA into BX.
30. Nothing is wrong with the instruction; it just uses an alternative addressing style.
32. (a) 11750H (b) 11950H (c) 11700H
34. BP or as an extended version EBP
36.
```
FIELDS     STRUC
F1         DW   ?
F2         DW   ?
F3         DW   ?
F4         DW   ?
F5         DW   ?
FIELDS     ENDS
```
38. Direct, relative, and indirect
40. The intersegment jump allows jumps between segments or to anywhere in the memory system while the intrasegment jump allows a jump to any location within the current code segment.
42. 32
44. Short
46. JMP BX
48. 2
50. AX, CX, DX, BX, SP, BP, DI, and SI in the same order as listed
52. PUSHFD

CHAPTER 4

2. The D-bit indicates the direction of flow for the data (REG to R/M or R/M to REG) and the W-bit indicates the size of the data (byte or word/doubleword).
4. DL
6. DS:[BX+DI]
8. MOV AL,[BX]
10. 8B 77 02
12. The REX prefix, which is used in the 64-bit flat mode, is the register extension that allows the 64-bit registers to be addressed in an instruction.

14.
```
MOV  AX,1000H
MOV  DS,AX
```
16. PUSH RAX
18. AX, CX, DX, BX, SP, BP, SI, and DI
20. (a) AX is copied to the stack. (b) A 32-bit number is retrieved from the stack and placed into ESI. (c) The word contents of the data segment memory location addressed by BX is pushed onto the stack. (d) EFLAGS are pushed onto the stack. (e) A word is retrieved from the stack and placed into DS. (f) A 32-bit number 4 is pushed onto the stack.
22. Bits 24–31 of EAX are stored in location 020FFH, bits 16–23 of EAX are stored into location 020FEH, bits 8–15 of EAX are stored into location 020FDH, and bits 0–7 of EAX are stored into location 020FCH. SP is then decremented by 4 to a value of 00FCH.
24. There are many possible locations, but SP = 0200H and SS = 0200H is one of them.
26. Both instruction load the address of NUMB into DI. The difference is that the MOV DI,OFFSET NUMB assembles as a move immediate and the LEA DI,NUMB assembles as an LEA instruction.
28. The LDS BX,NUMB instruction loads BX with the word stored at data segment memory location NUMB and DS is loaded from the data segment memory location addressed by NUMB+2.
30.
```
MOV  BX,NUMB
MOV  DX,BX
MOV  SI,DX
```
32. CLD clears the direction flag and STD sets the direction flag.
34. The LODSB instruction copies a byte of data from the data segment memory location addressed by SI into the AL register and then increments SI by one if the direction flag is cleared.
36. The OUTSB instruction sends the contents of the data segment memory location addressed by SI to the I/O port address by DX, then SI is incremented by one if the direction flag is cleared.
38.
```
MOV  SI,OFFSET SOURCE
MOV  DI,OFFSET DEST
MOV  CX,12
REP  MOVSB
```
40. XCHG EBX,ESI
42. The LAHF and SAHF instructions in non-64-bit application with the arithmetic coprocessor.
44. The XLAT instruction passes the contents of AL to BX to form an offset address that accesses a memory location whose content is then copied into AL.
46. The OUT DX,AX instruction copies the 16-bit contents of AX into the data segment memory location addressed by the DX register.
48. `MOV AH,ES:[BX]`
50. An assembly language directive is a special command to the assembler that may or may not generate code or data for the memory.
52. The directives, DB, DW, and DD, are used to define memory as a byte (DB), a word (DW), and a doubleword (DD).
54. The EQU (equate) directive allows a memory location to be equated to another memory location.
56. The .MODEL directive specifies the type of memory model used for a program.
58. Full segment definitions
60. PROC indicates the start of a procedure and ENDP indicates its end.
62.
```
STORE  PROC   NEAR
       MOV    [DI],AL
       MOV    [DI+1],AL
       MOV    [DI+2],AL
       MOV    [DI+3],AL
       RET
STORE  ENDP
```
64.
```
COPY  PROC   FAR
      MOV    AX,CS:DATA4
      MOV    BX,AX
      MOV    CX,AX
      MOV    DX,AX
      MOV    SI,AX
      RET
COPY  ENDP
```

CHAPTER 5

2. You cannot use mixed-size registers.
4. AX = 3100H, C = 0, A = 1, S = 0, Z = 0, and O = 0.
6.
```
ADD  AX,BX
ADD  AX,CX
ADD  AX,DX
ADD  AX,SP
```
8.
```
MOV  DI,AX
MOV  R12,RCX
ADD  R12,RDX
ADD  R12,RSI
```
10. INC SP
12. (a) SUB CX,BX (b) SUB DH,0EEH (c) SUB SI,DI (d) SUB EBP,3322H (e) SUB CH,[SI] (f) SUB DX,[SI+10] (g) SUB FROG,AL (h) SUB R10,R9
14.
```
MOV  BX,AX
SUB  BX,DI
SUB  BX,SI
SUB  BX,BP
```
16. The contents of DX and the carry flag are subtracted from the 16-bit contents of the data segment memory addressed by DI – 4 and the result is placed into DX.
18. AH (most significant) and AL (least significant)
20. The O and C flags contain the state of the most significant portion of the product. If the most significant part of the product is zero, then C and O are zero.
22.
```
MOV  DL,5
MOV  AL,DL
MUL  DL
MUL  DL
```
24. BX = DX times 100H

26. AX
28. The errors detected during a division are a divide overflow and a divide by zero.
30. AH
32.
```
MOV  AH,0
MOV  AL,BL
DIV  CL
ADD  AL,AL
MOV  DL,AL
MOV  DH,0
ADC  DH,0
```
34. It divides by AL by 10. This causes numbers between 0 and 99 decimal to be converted to unpacked BCD in AH (quotient) and AL (remainder).
36.
```
PUSH DX
PUSH CX
MOV  CX,1000
DIV  CX
MOV  [BX],AL
MOV  AX,DX
POP  CX
POP  DX
PUSH AX
AAM
MOV  [BX+1],AH
MOV  [BX+2],AL
POP  AX
MOV  AL,AH
AAM
MOV  [BX+3],AH
MOV  [BX+4],AL
```
38. Neither the BCD or the ASCII instructions function in the 64-bit mode.
40.
```
MOV  BH,DH
AND  BH,1FH
```
42.
```
MOV  SI,DI
OR   SI,1FH
```
44.
```
OR   AX,0FH
AND  AX,1FFFH
XOR  AX,0380H
```
46. TEST CH,4
48. (a) SHR DI,3 (b) SHL AL,1 (c) ROL AL,3 (d) RCR EDX,1 (e) SAR DH,1
50. Extra
52. The SCASB instruction is repeated while the condition is equal as long as CX is not zero.
54. CMPSB compares the byte contents of the byte in the data segment addressed by SI with the byte in the extra segment addressed by DI.
56. In DOS the letter C is displayed.

CHAPTER 6

2. A near JMP instruction
4. A far jump
6. (a) near (b) short (c) far
8. The IP or EIP register
10. The JMP AX instruction jumps to the offset address stored in AX. This can only be a near jump.
12. The JMP [DI] instruction jumps to the memory location addressed by the offset address stored in the data segment memory location addressed by DI. The JMP FAR PTR[DI] instruction jumps to the new offset address stored in the data segment memory location addressed by DI and the new segment addressed by the data segment memory location address by DI+2. JMP [DI] is a near jump and JMP FAR PTR [DI] is a far jump.
14. JA tests the condition of an arithmetic or logic instruction to determine if the outcome is above. If the outcome is above a jump occurs, otherwise no jump occurs.
16. JNE, JE, JG, JGE, JL, or JLE
18. JA and JBE
20. SETZ or SETE
22. ECX
24.
```
        MOV  DI,OFFSET DATAZ
        MOV  CX,150H
        CLD
        MOV  AL,00H
L1:     STOSB
        LOOP L1
```
26.
```
        CMP  AL,3
        JNE  @C0001
        ADD  AL,2
@C0001:
```
28.
```
        MOV SI,OFFSET BLOCKA
        MOV DI,OFFSET BLOCKB
        CLD
        .REPEAT
                LODSB
                STOSB
        .UNTIL AL == 0
```
30.
```
        MOV AL,0
        MOV SI,OFFSET BLOCKA
        MOV DI,OFFSET BLOCKB
        CLD
        .WHILE AL != 12H
                LODSB
                ADD AL,[DI]
                MOV [DI],AL
                INC DI
        .ENDW
```
32. A procedure is a reusable group of instructions that ends with a RET.
34. RET
36. By using NEAR or FAR to the right of the PROC directive
38.
```
CUBE  PROC  NEAR USES AX DX
      MOV   AX,CX
      MUL   CX
      MUL   CX
      RET
CUBE  ENDP
```
40.
```
SUMS  PROC  NEAR
      MOV   EDI,0
      ADD   EAX,EBX
      ADD   EAX,ECX
      ADD   EAX,EDX
      ADC   EDI,0
      RET
SUMS  ENDP
```
42. An interrupt is a hardware-initiated function call.
44. INT 0 through INT 255
46. The interrupt vector is used to detect and respond to divide errors.

48. The IRETD instruction is a 32-bit return that pops the return address into EIP.
50. When overflow is a 1
52. CLI and STI
54. If the value in the register or memory location under test in the destination operand is below or above the boundaries stored in the memory address by the source operand.
56. BP

CHAPTER 7

2. No, bytes must be defined in C++ using char.
4. EAX, EBX, ECX, EDX, and ES
6. Floating-point coprocessor stack
8. Data are accessed in array string using register SI to index the string element.
10. If no headers are used for a C++ program, it will be much smaller.
12. No. INT 21H is a 16-bit DOS call that cannot be used in the Windows 32-bit environment.
14.
```
#include “stdafx.h”
#include <conio.h>

int _tmain(int argc, _TCHAR* argv[])
{
        char a = 0;
        while ( a != ?@?)
        {
              a = _getche();
              _putch(a);
        }
        return 0;
}
```
16. The _putch(10) instruction displays the new line function and the _putch(13) returns the cursor to the left margin of the display.
18. Separate assembly modules are the most flexible.
20. The flat model must be used with the C prototype as in .MODEL FLAT,C and the function that is linked to C++ must be made public.
22. A 16-bit word is defined with the short directive.
24. Examples of events are mouse move, key down, etc. Event handlers catch these events so they can be used in a program.
26. Yes. The C++ editor can be used to edit an assembly language module, but the module must use the .TXT extension instead of .ASM.
28. `#define RDTSC _asm _emit 0x0f _asm _emit 0x31`
30.
```
;
;External function rotates a byte 3 places
 left
;
.586                  ;select Pentium and 32-
                      bit model
.model flat, C        ;select flat model with
                      C/C++ linkage
.stack 1024           ;allocate stack space
.code                 ;start code segment

public RotateLeft3    ;define RotateLeft3 as a
                      public function

RotateLeft3 proc      ;define procedure
Rotatedata:byte       ;define byte

       mov  al,Rotatedata
       rol  al,3
       ret

RotateLeft3 endp
```
32.
```
;Function that converts
;
.model flat,c
.stack 1024
.code

Public Upper

Upper proc
Char:byte

      mov  al,Char
      .if al >= 'a' && a; <= 'z'
             sub al,30h
      .endif
      Ret
Upper endp
```
34. Properties contains information about an object such as the foreground and background colors, etc.
36. `_asm inc ptr;`

CHAPTER 8

2. The TEST.ASM file, when assembled, generates the TEST.OBJ file and the TEST.EXE file if no switches appear on the command line.
4. PUBLIC indicates that a label is available to other program modules.
6. EXTRN
8. MACRO and ENDM
10. Parameters are passed to a macro through a parameter list that follows the MACRO keyword (on the same line).
12. The LOCAL directive defines local labels and must be on the line immediately following the MACRO line.
14.
```
ADDM  MACRO  LIST,LENGTH
      PUSH CX
      PUSH SI
      MOV CX,LENGTH
      MOV SI,LIST
      MOV AX,0
      .REPEAT
           ADD AX,[SI]
           INC SI
      .UNTILCXZ
      POP SI
      POP CX
      ENDM
```
16.
```
private: System::Void textBox1_KeyDown
(System::Object^ sender,System::
Windows::Forms::KeyEventArgs^ e)
```

```
{
      // this is called first
      keyHandled = true;
      if (e->KeyCode >= Keys::D0 &&
            e->KeyCode <= Keys::D9 &&
            e->Shift == false)
      {
            keyHandled = false;
      }
}
```

18. `private: System::Void textBox1_KeyDown (System::Object^ sender,System::Windows:: Forms::KeyEventArgs^ e)`

```
{
    {
        RandomNumber++;  //a global variable
        if ( RandomNumber == 63 )
             RandomNumber = 9;
    }
}
```

20. `bool direction;`

```
private: System::Void button1_Click(System
::Object^ sender,System::EventArgs^ e)
{
      label1->Text = "Shift Left = ";
      shift = true;
      data1 = 1;
      label2->Text = "00000001";
      timer1->Start();
}

private: System::Void button1_Click(System::
Object^ sender,System::EventArgs^ e)
{
      label1->Text = "Rotate Left = ";
      shift = false;
      data1 = 1;
      label2->Text = "00000001";
      timer1->Start();
}

private: System::Void radiobutton1_Click
(System::Object^ sender,System::EventArgs^ e)
{
      // left button
      direction = true;  //new bool variable
      if ( shift )
            label1->Text = "Shift Left = ";
      else
            label1->Text = "Rotate Left = ";
}

private: System::Void radiobutton2_Click
(System::Object^ sender,System::EventArgs^ e)
{
      // left button
      direction = false;  //new bool variable
      if ( shift )
            label1->Text = "Shift Right = ";
      else
            label1->Text = "Rotate Left = ";
}

private: System::Void timer1_Tick(System::
Object^ sender,System::EventArgs^ e)
{
      String^ temp = "";
      char temp1 = data1;
      if ( shift )
                 if ( direction )
                       _asm shl temp1,1;
                 else
                       _asm shr temp1,1;
        else
                 if ( direction )
                       _asm rol temp1,1;
                 else
                       _asm ror temp1,1;
      data1 = temp1;
      for (int a = 128; a > 0; a>>=1)
      {
                 if ( ( temp1 & a ) == a )
                       temp += "1";
                 else
                       temp += "0";
      }
      label2->Text = temp;
}
```

22. The MouseEventArgs in MouseDown contains the Button state that is tested against ::mouses::Mouse Buttons::Right to intercept the right mouse button in a program.

24. `private: System::Void Form1_MouseDown (System::Object^ sender,System::Windows:: Forms::MouseEventArgs^ e)`

```
{
      if (e->Button ==
       ::mouses::MouseButtons::Left &&
       e->Button == ::mouses::MouseButtons::
       Right)
      {
                 //left and right
      }
}
```

26. ForeColor sets the color of the text or characters in a control.

28. A large number is converted by repeated divisions by the number 10. After each digit the remainder is saved as a significant digit of the BCD result.

30. 30H

32. `int GetOct(void)`

```
{
      String^ temp;
      int result = 0;
      char temp1;
      temp = textBox1-≥Text;
      for ( int a = 0; a < temp->Length; a++ )
      {
            temp1 = temp.[a];
            _asm
            {
                  shl result,3
                  mov eax,0
                  mov al,temp1
                  sub al,30h
                  or result,eax
            }
      }
      return result;
}
```

34. `char Up(char temp)`

```
{
      if ( temp >= 'a' && temp <= 'z' )
            _asm sub temp,20h;
      Return temp;
}
```

36. The boot sector is where a bootstrap loader program is located that loads the operating system. The FAT is a table that contains numbers that indicate whether a cluster is free, bad, or occupied. If occupied, an

FFFFH indicates the end of a file chain or the next cluster number in the file chain. The director holds information about a file or a folder.

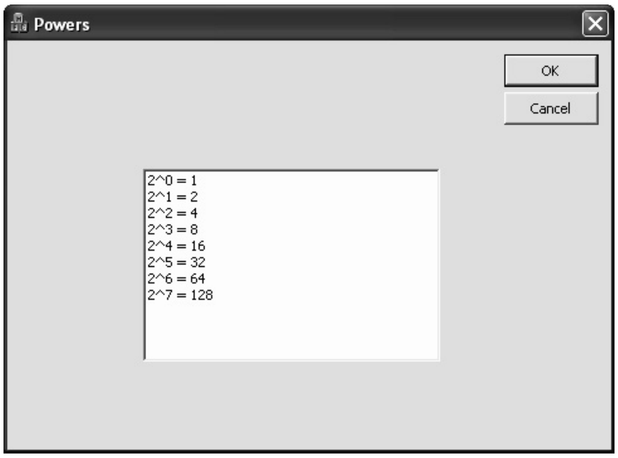
38. Sectors.
40. A cluster is a grouping of sectors.
42. 4G bytes
44. 8
46. 256
48. `File::Replace(?TEST.LST?, ?TEST.LIS?, ?TEST.BAK?);`

```
// Creates TEST.LIS and TEST.BAK, Deletes
TEXT.LST
```

50. A control is a common object that can be used in any visual programming language.
52. See Figure D–1 for the output (the stock ListBox contains the output).

```
String CPowersDlg::GetNumb(int temp)
{
      char numb[10];
      _asm
      {
            mov  eax,temp
            mov  ebx,10
            push ebx
            mov  ecx,0
loop1:
            mov  edx,0
            div  ebx
            push edx
            cmp  eax,0
            jnz  loop1
loop2:
            pop  edx
            cmp  edx,ebx
            je   loop3
            add  dl,30h
            mov  numb[ecx],dl
            inc  ecx
            jmp  loop2
loop3:
            mov  byte ptr numb[ecx],0
     }
      return numb;
}

//code placed in the OnInitDlg function
int tempval = 1;
for ( int a = 0; a < 8; a++ )
{
      CString temp = "2^ = ";
      temp.SetAt(2, a + 0x30);
      temp += GetNumb(tempval);
      List.InsertString(a, temp);
      tempval <<= 1;
}
```



```
2^0 = 1
2^1 = 2
2^2 = 4
2^3 = 8
2^4 = 16
2^5 = 32
2^6 = 64
2^7 = 128
```

FIGURE D–1

54. `private: System::Void Clear()`

```
{
      panel1->Visible = false;
      panel2->Visible = false;
      panel3->Visible = false;
      panel4->Visible = false;
      panel5->Visible = false;
      panel6->Visible = false;
      panel7->Visible = false;
}
private: System::Void Form1_KeyDown(System::
Object^  sender,System::Windows::Forms::
KeyEventArgs^ e)
{
      char lookup[] = {0x3f, 6, 0x5b, 0x4f,
             0x66, 0x6d, 0x7d, 7, 0x7f,0x6f,
             0x77, 0x7c, 0x39, 0x5e, 0x79,
             0x71};
      if (e->KeyCode >= Keys::D0 &&
              e->KeyCode <= Keys::D9)
      {
            ShowDigit(lookup
                  [e->KeyValue -
                  0x30]);  //display
                           the digit
      }
      else if (e->KeyCode >= Keys::A &&
              e->KeyCode <= Keys::F)
      {
            ShowDigit(lookup
                  [e->KeyValue -
                  0x37]);  //display
                           letter
      )
}
private: System::Void ShowDigit(unsigned
char code)
{
      Clear();
      if (( code & 1 ) == 1)  //test a
                              segment
            panel1->Visible = true;
      if (( code & 2 ) == 2)  //test b
                              segment
            panel4->Visible = true;
      if (( code & 4 ) == 4)  //test c
                              segment
            panel5->Visible = true;
      if (( code & 8 ) == 8)  //test d
                              segment
            panel3->Visible = true;
      if (( code & 16 ) == 16)  //test e
                                segment
            panel6->Visible = true;
      if (( code & 32 ) == 32)  //test f
                                segment
            panel7->Visible = true;
      if (( code & 64 ) == 64)  //test g
                                segment
            panel2->Visible = true;
}
```

```
private: System::Void Form1_Load(System::
Object^ sender,System::EventArgs^ e)
{
      Clear();
}
```

CHAPTER 9

2. Yes and no. The current drive of a logic zero is reduced to 2.0 mA and the noise immunity is reduced to 350 mV.
4. Address bits A_0–A_7
6. A read operation
8. The duty cycle must be 33%.
10. A write is occurring.
12. The data bus is sending data to the memory or I/O.
14. IO/$\overline{M}$, DT/$\overline{R}$, and $\overline{SS0}$
16. Signals to the coprocessor, which indicate what the microprocessor queue is doing.
18. 3
20. 14 MHz/6 = 2.33 MHz
22. Address bus connection A_0–A_{15}
24. 74LS373 transparent latch
26. If too many memory and/or I/O devices are attached to a system the buses must be buffered.
28. 4
30. Fetch and execute
32. (a) The address is output along with ALE (b) Time is allowed for memory access and the READY input is sampled. (c) The read or write signal is issued. (d) Data are transferred and read or write is deactivated. (e) Wait allows additional time for memory access.
36. Selects one or two stages of synchronization for READY
38. Minimum mode operation is most often used in embedded applications and maximum mode operation was most often used in early personal computers.

CHAPTER 10

2. (a) 256 (b) 2K (c) 4K (d) 8K (e) 1M
4. Select the memory device
6. Cause a write to occur
8. The microprocessor allows 460 ns for memory at 5 MHz, but because there is a small delay in the connections to the memory, it would be best not to use a 450 ns memory device in such a system without one wait state.
10. Static random access memory
12. 250 ns
14. The address inputs to many DRAMs are multiplexed so one address input accepts two different address bits, reducing the number of pins required to address memory in a DRAM.
16. Generally the amount of time is equal to a read cycle and represents only a small amount of time in a modern memory system.
18. See Figure D–2.
20. One of the eight outputs becomes a logic zero as dictated by the address inputs.

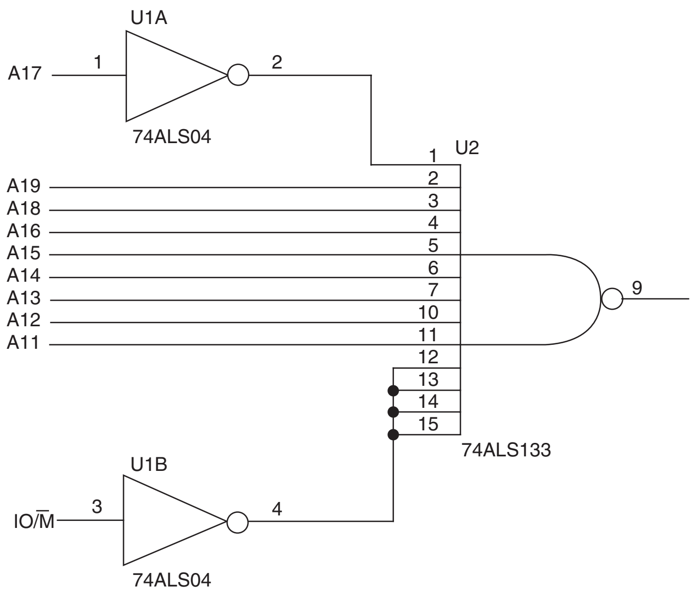


FIGURE D–2

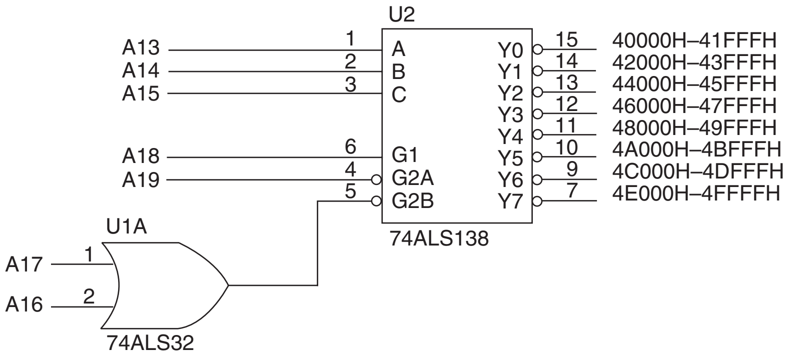


FIGURE D–3

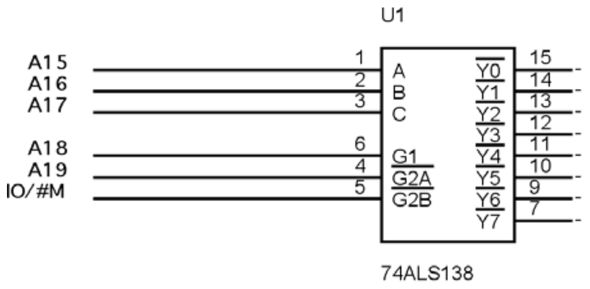


FIGURE D–4

22. See Figure D–3.
24. Verilog hardware description language
26. The architecture block between begin and end
28. $\overline{\text{MRDC}}$ and $\overline{\text{MWTC}}$
30. See Figure D–4.
32. 5
34. 1
36. $\overline{\text{BHE}}$ selects the upper memory bank and A_0 selects the lower memory bank.
38. Separate decoders and separate write signals
40. Lower memory bank
42.
```
library ieee;
use ieee.std_logic_1164.all;
entity DECODER_10_28 is
port (
      A23, A22, A21, A20, A19, A18, A17,
      A16, A0, BHE, MWTC: in STD_LOGIC;
      SEL, LWR, HWR: out STD_LOGIC
);

end;

architecture V1 of DECODER_10_28 is

begin

     SEL <= A23 or A22 or A21 or A20 or A19
     or A18 or (not A17) or (not A16);
     LWR <= A0 or MWTC;
     HWR <= BHE or MWTC;

end V1;
```
44. See Figure D–5.
48. Yes, as long as a memory location on the DRAM is not accessed.
50. 128 bits wide

CHAPTER 11

2. The I/O address is stored in the second byte of the instruction.
4. DX
6. The OUTSB instruction transfers the data segment byte addressed by SI to the I/O port addressed by DX, then SI is incremented by one.
8. Memory mapped I/O uses any instruction that transfers data to or from the memory for I/O, while isolated I/O requires the use of the IN or OUT instruction.
10. The basic output interface is a latch that captures output data and holds it for the output device.
12. Lower
14. 4
16. It removes mechanical bounces from a switch.
18. See Figure D–6.
20. See Figure D–7.
22. See Figure D–8.
24. If the port is 16 bits wide, there is no need to enable either the low or high half.
26. D_{47}–D_{40}
28. Group A is port A and PC_4–PC_7, while group B is port B and PC_3–PC_0.
30. $\overline{\text{RD}}$
32. Inputs
34. The strobe input latches the input data and sets the buffer full flag and interrupt request.
36.
```
DELAY  PROC  NEAR USES ECX

       MOV ECX, 7272727
D1:
       LOOPD D1
       RET

DELAY  ENDP
```

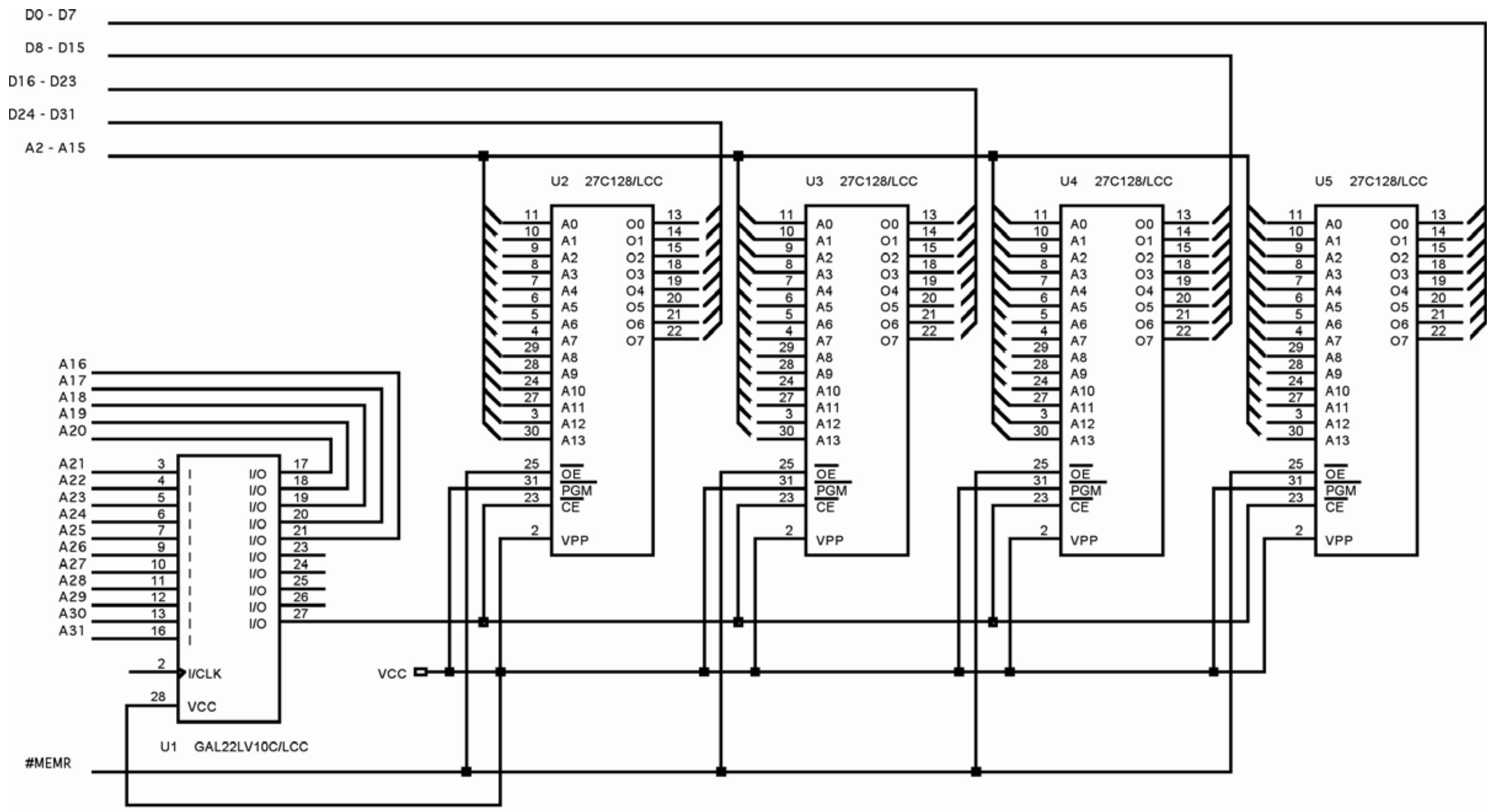

D0 - D7
D8 - D15
D16 - D23
D24 - D31
A2 - A15
U2 27C128/LCC
U3 27C128/LCC
U4 27C128/LCC
U5 27C128/LCC
A0
A1
A2
A3
A4
A5
A6
A7
A8
A9
A10
A11
A12
A13
O0
O1
O2
O3
O4
O5
O6
O7
OE
PGM
CE
VPP
A16
A17
A18
A19
A20
A21
A22
A23
A24
A25
A26
A27
A28
A29
A30
A31
I/O
I/CLK
VCC
U1 GAL22LV10C/LCC
#MEMR

FIGURE D–5

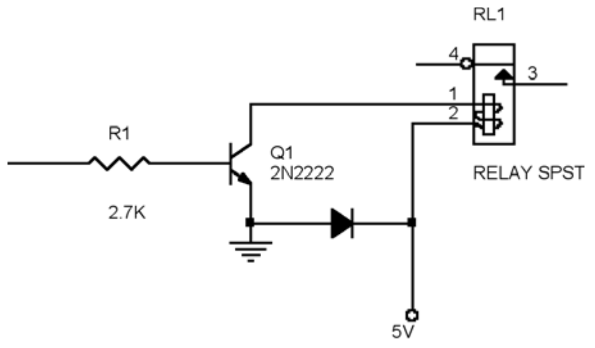


FIGURE D–6

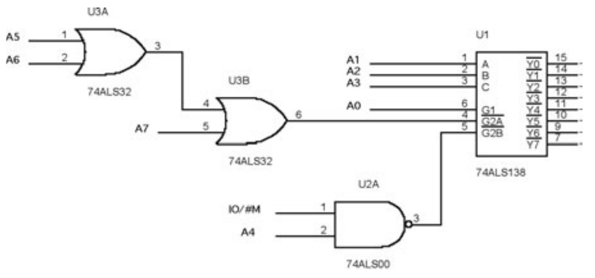


FIGURE D–7

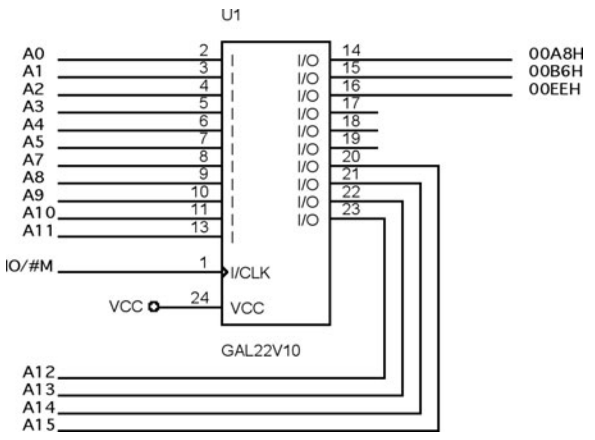


FIGURE D–8

38. The strobe signal ($\overline{STB}$)
40. The INTR pin is enabled by setting the INTE bit in PC_4 (port A) or PC_2 (port B).
42. When data are output to the port $\overline{OBF}$ becomes a 0 and when $\overline{ACK}$ is sent to the port $\overline{OBF}$ becomes a 1.
44. Group or port A contains the bidirectional data.
46. The 01H command is sent to the LCD display.
48.
```
;Displays the null terminated string addressed
by DS:BX
;uses a macro called SEND to send data to
the display
;
DISP      PROC  NEAR  USES BX
          SEND  86H,2,1     ;move cursor to
                            position 6
          .WHILE BYTE PTR [BX] != 0
                 SEND [BX],0,1
                 INC  BX
          .ENDW
          RET
DISP      ENDP
```
50. The only changes that need to be made are that instead of four rows there are three rows and three pull-up resisters connected to port A and five columns to connect to port B. Of course, the software also needs some minor changes.
54. 6
58. Least significant
62. Data that are sent a bit at a time without any clocking pulses
64.
```
LINE  EQU  023H
LSB   EQU  020H
MSB   EQU  021H
FIFO  EQU  022H

       MOV  AL,10001010B ;enable baud
divisor
      OUT  LINE,AL

      MOV  AL,60         ;program baud rate
      OUT  LSB,AL
      MOV  AL,0
      OUT  MSB,AL

      MOV  AL,00011001B   ;program 7-data, odd
      OUT  LINE,AL        ;parity, one stop

      MOV  AL,00000111B   ;enable transmitter
                           and
      OUT  FIFO,AL        ;and receiver
```
66. Simplex = receiving or sending data; half-duplex = receiving and sending data; but only one direction at a time; and full-duplex = receiving and sending data at the same time.
68.
```
SENDS  PROC  NEAR

MOV     CX,16
       .REPEAT
          .REPEAT
             IN    AL,LSTAT ;get line
                             status register
             TEST  AL,20H   ;test TH bit
          .UNTIL !ZERO?
          LODSB              ;get data
          OUT  DATA,AL       ;transmit data
       .UNTILCXZ

   RET

SENDS ENDP
```
70. 0.01V
72.
```
       .MODEL  TINY
       .CODE
       .STARTUP
              MOV     DX,400H
              .WHILE  1
              MOV     CX,256
              MOV     AL,0
              .REPEAT
                  OUT     DX,AL
                  INC     AL
                  CALL    DELAY
              .UNTILCXZ
              MOV   CX,256
              .REPEAT
                  OUT     DX,AL
                  DEC     AL
                  CALL    DELAY
              .UNTILCXZ
              .ENDW
DELAY           PROC   NEAR

; 39 microsecond time delay

DELAY   ENDP
        END
```

74. INTR indicates that the converter has completed a conversion.
76. See Figure D–9.

CHAPTER 12

2. An interrupt is a hardware- or software-initiated subroutine call.
4. Interrupts only use computer time when the interrupt is activated.
6. INT, INT3, INTO, CLI, and STI
8. The first 1K byte of the memory system in real mode and anywhere in protected mode.
10. 00H through 1FH
12. Anywhere in the memory system
14. A real mode interrupt pushes CS, IP, and the FLAGS onto the stack, while a protected mode interrupt pushes CS, EIP, and the EFLAGS onto the stack.
16. The INTO occurs if overflow is set.
18. The IRET instruction pops the flags and the return address from the stack.
20. The state of the interrupt structure is stored on the stack, so when the return occurs, it is restored. Both the interrupt and trace flags are cleared.
22. The IF flag controls whether the INTR pin is enabled or disabled.
24. The IF flag is enabled or disabled by using the STI or CLI instructions.
26. 2
28. Level sensitive
30. Vector
32. See Figure D–10.
34. The pull-up resistors guarantee that vector number fetched from the data bus during an interrupt acknowledge is an FFH.
36. Because the interrupt request signal (INTR) is generated by ORing all the requests together, the software must ask or poll each device to determine which device caused the request.
38. 9
40. The CAS pins are cascade pins used to cascade 8259s for more than eight interrupt inputs.
42. An OCW is an operational control word for the 8259.
44. ICW_2

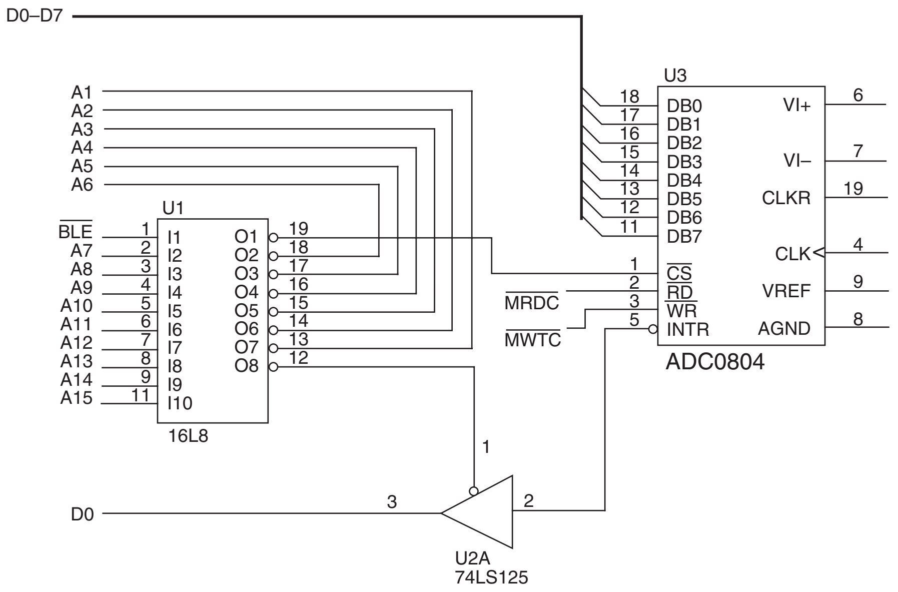


FIGURE D–9

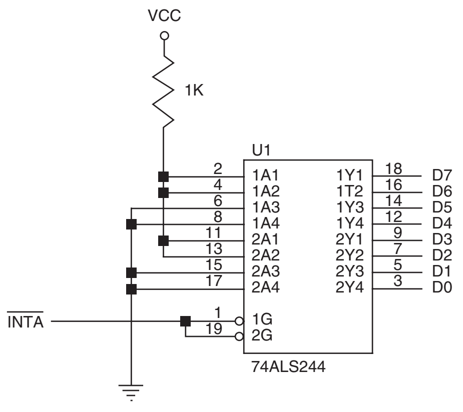


FIGURE D–10

46. Program sensitivity and single or multiple 8259s
48. The most recent interrupt request level becomes the lowest level interrupt after being serviced.
50. INT 8 through INT 0FH

CHAPTER 13

2. When a 1 is placed on HOLD, the program stops executing and the address, data, and control buses go to their high-impedance state.
4. I/O to memory
6. DACK
8. The microprocessor is in its hold state and the DMA controller has control of the buses.
10. 4
12. The command register
16. A pen drive is a USB device that acts as a storage device using a flash memory.
18. Tracks
20. Cylinder
22. See Figure D–11.
24. The heads in a hard disk drive are aerodynamically designed to ride on a cushion of air as the disk spins and are therefore called flying heads.
26. The stepper motor positioning mechanism is noisy and not very precise, while the voice coil positioning mechanism is silent and very accurate because its placement can be continuously adjusted.
28. A CD-ROM is an optical device for storing music or digital data and has a capacity of about 660M or 700M (80 minute) bytes.
30. A TTL monitor uses TTL signals to generate a display and an analog monitor uses analog signals.
32. Cyan, magenta, and yellow
34. 1024 lines with 1280 horizontal elements per line
36. The DVI-D and HDMI connectors are the latest style of digital video input connectors for all types of video equipment.
38. 16 million colors

CHAPTER 14

2. Word (16-bits, ±32K), doubleword (32-bits, ±2G), and quadword (64-bits, $\pm 9 \times 10^{18}$)
4. Single-precision (32 bits), double-precision (64 bits), and temporary-precision (80 bits)
6. (a) −7.75 (b) .5625 (c) 76.5 (d) 2.0 (e) 10.0 (f) 0.0
8. The microprocessor continues executing microprocessor (integer) instructions while the coprocessor executes a floating-point instructions.
10. It copies the coprocessor status register to AX.
12. By comparing the two registers and then by transferring the status word to the AX register. If the SAHF instruction is next executed, a JZ instruction can be used to test the outcome of the coprocessor compare instruction.
14. FSTSW AX
16. Data are always stored as an 80-bit temporary precision number.
18. 0
20. Affine allows positive and negative infinity, while projective assumes infinity is unsigned.
22. Extended (temporary) precision
24. The contents of the top of the stack are copied into memory location DATA as a floating-point number.
26. FADD ST,ST(3)
28. FSUB ST(2),ST

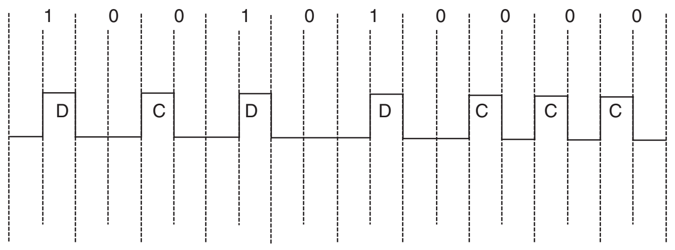


FIGURE D–11

30. Forward division divides the top of the stack by the contents of a memory location and returns the quotient to the top of the stack. Reverse division divides the top of the stack into the contents of the memory location and returns the result to the top of the stack. If no operand exists, then forward division divides ST(1) by ST and reverse division divides ST by ST(1).
32. It performs a MOV to ST if the condition is below.
34.
```
RECIP   PROC     NEAR
        MOV    TEMP,EAX
        FLD    TEMP
        FLD1
        FDIVR
        FSTP TEMP
        MOV  EAX,TEMP
RECIP   ENDP
TEMP    DD  ?
```
36. Finds the function $2^X - 1$.
38. FLDPI
40. It indicates that register ST(2) is free.
42. The state of the machine
44.
```
CAPR    PROC   NEAR
        FLDPI
        FADD ST,ST(1)
        FMUL F
        FMUL C1
        FLD1
        FDIVR
        FTSP XC
        RET
CAPR    ENDP
```
46. In modern software it is never used.
48.
```
TOT     PROC   NEAR
        FLD  R2
        FLD1
        FDIVR
        FLD  R3
        FLD1
        FDIVR
        FLD  R4
        FLD1
        FDIVR
        FADD
        FADD
        FLD1
        FDIV
        FADD  R1
        FSTP  RT
        RET
TOT     ENDP
```
50.
```
PROD    PROC   NEAR
        MOV  ECX,100
        .REPEAT
              FLD   ARRAY1[ECX*8-8]
              FUML  ARRAY2[ECX*8-8]
              FSTP  ARRAY3[ECX+8-8]
        .UNTILCXZ
        RET
PROD    ENDP
```
52.
```
POW     PROC    NEAR
        MOV  TEMP,EBX
        FLD  TEMP
        F2XM1
        FLD1
        FADD
        MOV   TEMP,EAX
        FLD   TEMP
        FYL2X
        FSTP  TEMP
        MOV   ECX,TEMP
        RET
POW     ENDP
```
54.
```
GAIN    PROC   NEAR
        MOV  ECX,100
        .REPEAT
              FLD  DWORD PTR VOUT[ECX*4-4]
              FDIV DWORD PTR VIN[ECX*4-4]
              CALL LOG10
              FIMUL TWENTY
              FSTP DWORD PTR DBG[ECX*4-4]
        .UNTILCXZ
        RET
TWENTY  DW    20
GAIN    ENDP
```
56. The EMMS instruction clears the coprocessor stack to indicate that the MMX unit has completed using the stack.
58. Signed saturation occurs when byte-sized numbers are added and have values of 7FH for an overflow and 80H for an underflow.
60. The FSAVE instruction stores all the MMX registers in memory.
62. Single-instruction, multiple-data instructions
64. 128 bits
66. 16
68. Yes

CHAPTER 15

2. 8- or 16-bit depending on the socket configuration.
4. See Figure D–12.
6. See Figure D–13.
8. See Figure D–14.
12. 16 bits
14. The configuration memory identifies the vendor and also information about the interrupts.
16. This is the command/bus enable signal that is high to indicate the PCI bus contains a command and low for data.
18.
```
MOV  AX,0B108H
MOV  BX,0
MOV  DI,8
INT  1AH
```
20. 2.5 GHz
22. Yes
24. COM_1
30. 1.5 Mbps, 12 Mbps, and 480 Mbps
32. 5 meters
34. 127
36. An extra bit that is thrown in the data stream if more than six ones are sent in a row.
38. 1 to 1023 bytes
40. The PCI transfers data at 33 MBs, while AGP transfers data at 2 GBps (8 ×).

CHAPTER 16

2. The hardware enhancements include internal timers, additional interrupt inputs, chip selection logic, serial communications ports, parallel pins, DMA controller, and an interrupt controller.
4. 10 MHz
6. 3 mA
8. The point at which the address appears
10. 260 ns for the 16 MHz version operated at 10 MHz
12.
```
MOV  AX,1000H
MOV  DX,0FFFEH
OUT  DX,AX
```
14. 10 on most versions of the 80186/80188 including the internal interrupts.

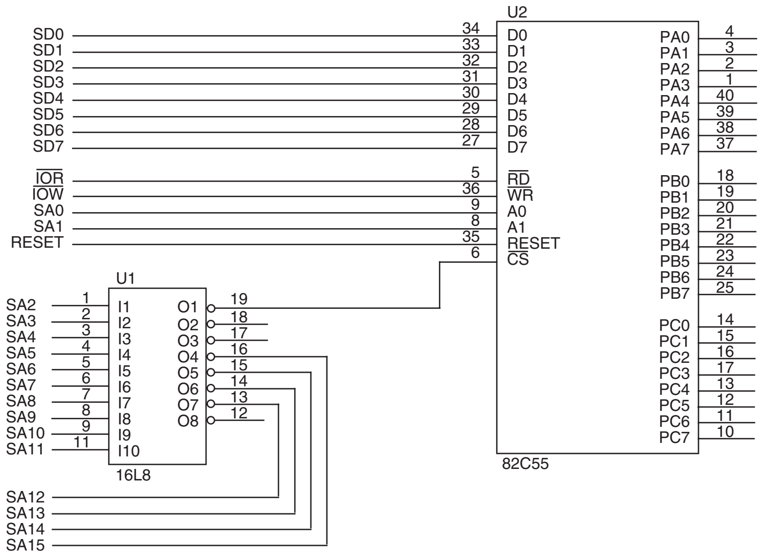


FIGURE D–12

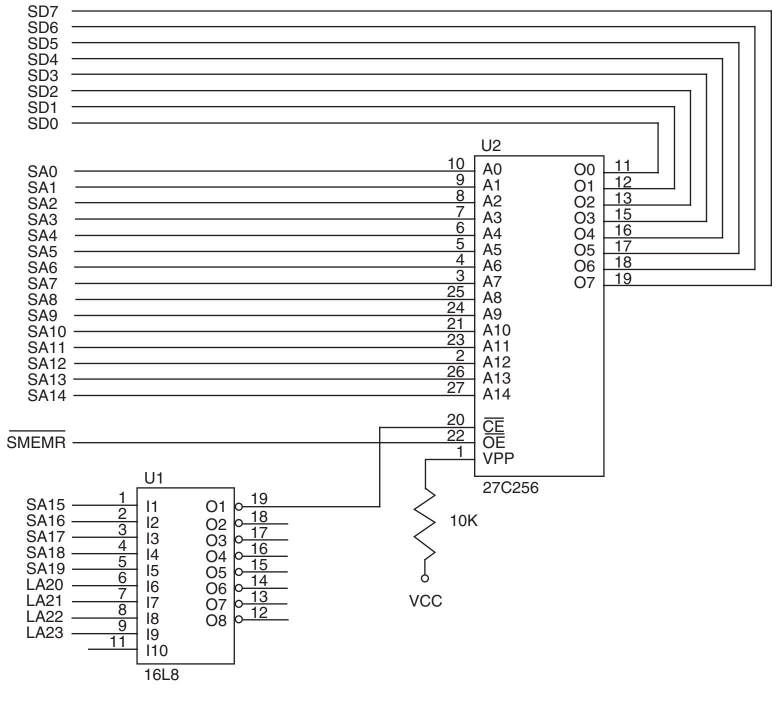


FIGURE D–13

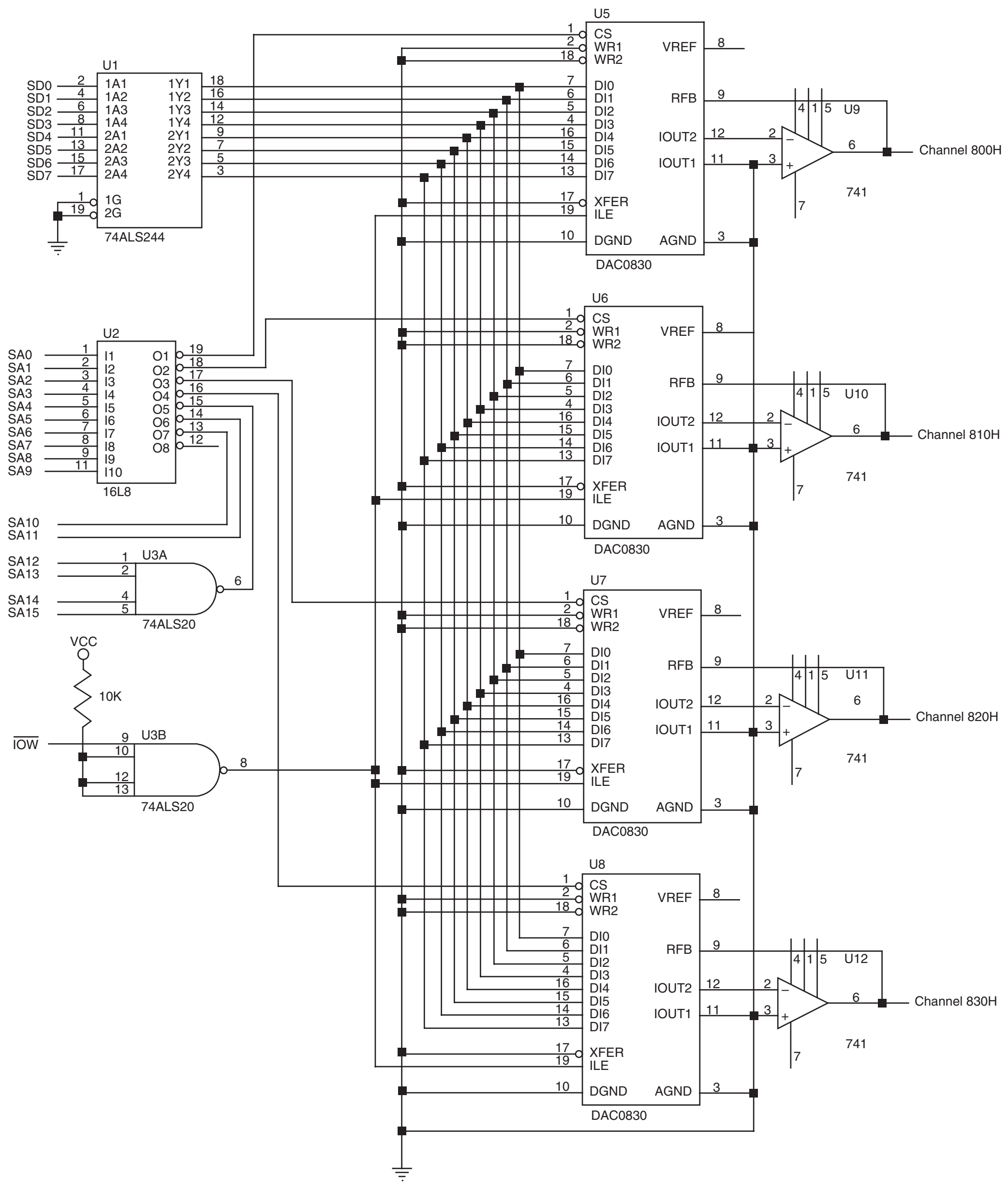


FIGURE D–14

16. The interrupt control registers control a single interrupt.
18. The interrupt poll register acknowledges the interrupt, while the interrupt poll status register does not acknowledge the interrupt.
20. 3
22. Timer 2
24. It determines whether the enable counter bit functions.
26. The ALT bit selects both compare registers so the duration of the logic 1 and logic 0 output times can be programmed.
28.
```
MOV  AX,123
MOV  DX,0FF5AH
OUT  DX,AX
MOV  AX,23
ADD  DX,2
OUT  DX,AX
MOV  AX,0C007H
MOV  DX,0FF58H
OUT  DX,AX
```
30. 2
32. Place a logic 1 in both the CHG/$\overline{\text{NOCHG}}$ and START/$\overline{\text{STOP}}$ bits of the control register.
34. 7

36. Chip
38. 15
40. It determines the operation of the $\overline{PCS5}$ and $\overline{PCS6}$ pins.
42.
```
MOV  AX,1001H
MOV  DX,0FF90H
OUT  DX,AX
MOV  AX,1048H
OUT  DX,AX
```
44. 1G
46. Verify for read access.
48. An RTOS is a real-time operating system that has a predictable and guaranteed time for threads access.

CHAPTER 17

2. 64T
4. See Figure D–15.
6. The memory system has up to 4G bytes and the bank enable signals select one or more of the 8-bit-wide banks of memory.
8. The pipeline allows the microprocessor to send the address of the next memory location, while it fetches the data from the prior memory operation. This allows the memory additional time to access the data.
10. 0000H–FFFFH
12. I/O has the same address as earlier models of the microprocessor. The difference is that the I/O is arranged as a 32-bit-wide space with four 8-bit banks that are selected by the bank enable signals.
14. The $\overline{BS16}$ pin causes the microprocessor to function with an 8-bit-wide data bus.
16. The first four debug registers (DR_0–DR_3) contain breakpoint addresses; registers DR_4 and DR_5 are reserved for Intel's use; DR_6 and DR_7 are used to control debugging.

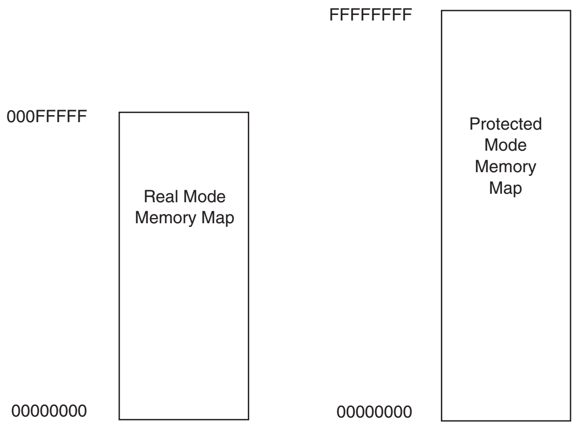


FIGURE D–15

18. The test registers are used to test the translation look-aside buffer.
20. The PE bit switches the microprocessor into protected mode if set and real mode if cleared.
22. Scaled-index addressing used a scaling factor of 1, 2, 4, or 8 times to scale addressing from byte, word, doubleword, or quadword.
24. (a) the address in the data segment at the location pointed to by EBX times 8 plus ECX (b) the address in the data segment array DATA pointed to by the sum of EAX plus EBX (c) the address at data segment location DATA (d) the address in the data segment pointed to by EBX
26. Type 13 (0DH)
28. The interrupt descriptor table and its interrupt descriptors
30. A selector appears in a segment register and it selects a descriptor from a descriptor table. It also contains the requested privilege level of the request.
32. The global descriptor table register
34. Because a descriptor addresses up to 4G of memory and there are 8K local and 8K global descriptor available at a time, 4G times 16K = 64T.
36. The TSS holds linkages and registers of a task so tasks can be switched efficiently.
38. The switch occurs when a logic 1 is placed into the PE bit of CR_0.
40. Virtual mode, which simulates DOS in protected mode, sets up 1M memory spans that can operate in the real mode.
42. 4K
44. The 80486 has an internal 8K cache and also contains a coprocessor.
46. The register sets are virtually identical.
48. $\overline{PCHK}$ and DP_0–DP_3
50. 8K
52. A burst is when four 32-bit numbers are read or written between the cache and memory.
54. Built-in self test

CHAPTER 18

2. 64G bytes
4. These pins generate and check parity.
6. The burst ready pin is used to insert wait state into the bus cycle.
8. 18.5 ns
10. T_2
12. An 8K byte data cache and an 8K-byte instruction cache.
14. Yes, if one is a coprocessor instruction and the integer instructions are not dependent.

16. The SSM mode is used for power management in most systems.
18. 38000H
20. The CMPXCH8B instruction compares the 64-bit number in EDX:EAX with a 64-bit number stored in memory. If they are equal, ECX:EBX is stored in memory. If not equal, the contents of memory are moved into EDX:EAX.
22. ID, VIP, VIF, and AC
24. To access 4M pages, the page tables are dropped and only the page directory is used with a 22-bit offset address.
26. The Pentium Pro is an improved version of the Pentium that contains three integer units, an MMX unit, and a 36-bit address bus.
28. 36 address bits on A_3 through A_{35} (A_0–A_2 are encoded in the bank selection signals)
30. The access time in a 66 MHz Pentium is 18.5 ns and in the Pentium Pro at 66 MHz access time is 17 ns.
32. SDRAM that is 72 bits wide is purchased for ECC memory applications instead of 64-bit-wide memory.

CHAPTER 19

2. 512K, 1M, or 2M
4. The Pentium Pro cache is on the main board and the Pentium 2 cache is in the cartridge and operates at a high speed.
6. 64G bytes
8. 242
10. The read and write signals are developed by the chip set instead of the microprocessor.
12. 8 ns after the first quadword is accessed. The first quadword still requires 60 ns for access.
14. Model-specific registers have been added for SYSENTER_CS, STSENTER_SS, and SYSENTER_ESP.
16. The ECX register address the MSR number when the RDMSR instruction executes. After execution, EDX:EAX contains the contents of the register
18.
```
TESTS  PROC    NEAR
       CPUID
       BT    EDX,800H
       RET
TESTS  ENDP
```
20. EDX to the EIP register and the value in ECX to the ESP register.
22. Ring 3
24. Pentium Pro
26. The Pentium 4 or Core2 requires a power supply with an additional 12 V connector for the main board. A Pentium 4–compliant supply must be used.
28.
```
bool Hyper()
{
     _asm
     {
           bool State = true;
           mov  eax,1
           cpuid
           mov  temp1,31h
           bt   edx,28     ;check for hyper-
                            threading
           jc   Hyper1
           mov  State, 0
Hyper1:
      }
      return State;
}
```