This work is protected by US copyright laws and is for instructors' use only.

Online Instructor's Manual
to accompany

The x86 PC: Assembly Language, Design, and Interfacing

5th Edition

Muhammad Ali Mazidi

Janice Gillispie Mazidi

Danny Causey

Prentice Hall

Boston Columbus Indianapolis New York San Francisco Upper Saddle River

Amsterdam Cape Town Dubai London Madrid Milan Munich Paris Montreal Toronto

Delhi Mexico City Sao Paulo Sydney Hong Kong Seoul Singapore Taipei Tokyo

This work is protected by United States copyright laws and is provided solely for the use of instructors in teaching their courses and assessing student learning. Dissemination or sale of any part of this work (including on the World Wide Web) will destroy the integrity of the work and is not permitted. The work and materials from it should never be made available to students except by instructors using the accompanying text in their classes. All recipients of this work are expected to abide by these restrictions and to honor the intended pedagogical purposes and the needs of other instructors who rely on these materials.

Copyright © 2010 Pearson Education, Inc., publishing as Prentice Hall, Upper Saddle River, New Jersey and Columbus, Ohio. All rights reserved. Manufactured in the United States of America. This publication is protected by Copyright, and permission should be obtained from the publisher prior to any prohibited reproduction, storage in a retrieval system, or transmission in any form or by any means, electronic, mechanical, photocopying, recording, or likewise. To obtain permission(s) to use material from this work, please submit a written request to Pearson Education, Inc., Permissions Department, One Lake Street, Upper Saddle River, New Jersey.

Many of the designations by manufacturers and seller to distinguish their products are claimed as trademarks. Where those designations appear in this book, and the publisher was aware of a trademark claim, the designations have been printed in initial caps or all caps.

10 9 8 7 6 5 4 3 2 1

PEARSON

ISBN-13: 978-0-13-504081-2
ISBN-10: 0-13-504081-7

CHAPTER 0: INTRODUCTION TO COMPUTING

SECTION 0.1: NUMBERING AND CODING SYSTEMS

1.
(a) 1210 = 11002
(b) 12310 = 0111 10112
(c) 6310 = 0011 11112
(d) 12810 = 1000 00002
(e) 100010 = 0011 1110 10002
2.
(a) 1001002 = 3610
(b) 10000012 = 6510
(c) 111012 = 2910
(d) 10102 = 1010
(e) 001000102 = 3410
3.
(a) 1001002 = 2416
(b) 10000012 = 4116
(c) 111012 = 1D16
(d) 10102 = 0A16
(e) 001000102 = 2216
4.
(a) 2B916 = 0010 1011 10012, 69710
(b) F4416 = 1111 0100 01002, 390810
(c) 91216 = 1001 0001 00102, 232210
(d) 2B16 = 0010 10112, 4310
(e) FFFF16 = 1111 1111 1111 11112, 6553510
5.
(a) 1210 = 0C16
(b) 12310 = 7B16
(c) 6310 = 3F16
(d) 12810 = 8016
(e) 100010 = 3E816
6.
(a) 1001010 = 0011 0110
(b) 111001 = 0000 0111
(c) 10000010 = 0111 1110
(d) 111110001 = 0000 1111
7.
(a) 2C+3F = 6B
(b) F34+5D6 = 150A
(c) 20000+12FF = 212FF
(d) FFFF+2222 = 12221

8. (a) 24F-129 = 12616
(b) FE9-5CC = A1D16
(c) 2FFFF-FFFFF = 3000016
(d) 9FF25-4DD99 = 5218C16

9. (a) Hex: 30, 31, 32 , 33, 34, 35, 36, 37, 38, 39
(b) Binary: 11 0000, 11 0001, 11 0010, 11 0011, 11 0100, 11 0101, 11 0110, 11 0111, 11 1000, 11 1001.

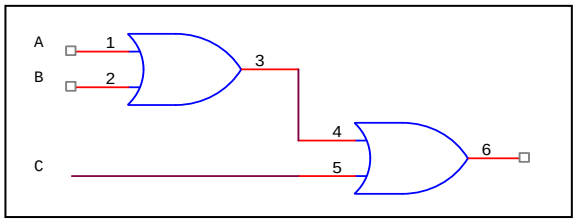
ASCII	(hex)	Binary		
	0	30	011	0000
	1	31	011	0001

2	32	011	0010
3	33	011	0011
4	34	011	0100
5	35	011	0101
6	36	011	0110
7	37	011	0111
8	38	011	1000
9	39	011	1001

10. 000000 22 55 2E 53 2E 41 2E 20 69 73 20 61 20 63 6F 75 "U.S.A. is a cou
000010 6E 74 72 79 22 0D 0A 22 69 6E 20 4E 6F 72 74 68 ntry".."in North
000020 20 41 6D 65 72 69 63 61 22 0D 0A America"..

SECTION 0.2: DIGITAL PRIMER

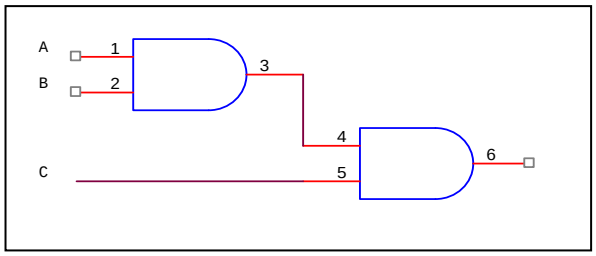
11.



12.

A	B	C	Y
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

13.

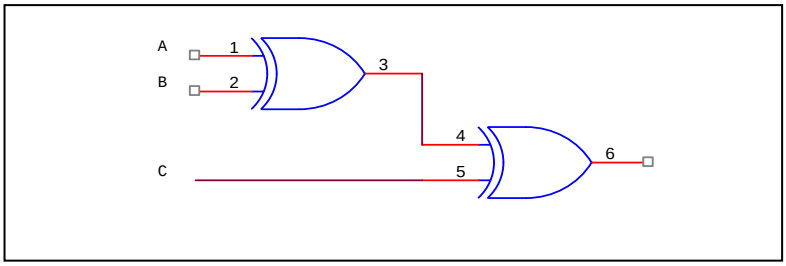


14.

A	B	C	Y
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0

1	1	0	0
1	1	1	1

15.



A	B	C	Y
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

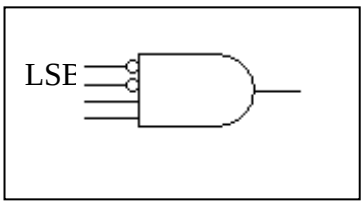
16.

A	B	C	Y
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

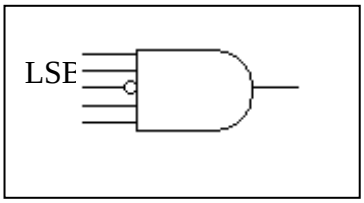
17.

A	B	C	Y
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

18.



19.



20.

CLK	D	Q
No	X	NC
Yes	0	0
Yes	1	1

SECTION 0.3: INSIDE THE COMPUTER

21. (a) 4
(b) 4
(c) 4
(d) 1 048 576, 220
(e) 1024K
(f) 1 073 741 824, 230
(g) 1 048 576 K
(h) 1024M
(i) 8388608, 8192K

22. Disk storage capacity / size of a page = (2*230) / (25*80) = 1 million pages
23. (a) 9FFFFh – 10000h = 8FFFFh = 589 824 bytes
(b) 576 kbytes
24. 232 – 1 = 4 294 967 295
25. (a) FFh, 255
(b) FFFFh, 65535
(c) FFFF FFFFh, 4 294 967 295
(d) FFFF FFFF FFFF FFFFh, 18 446 744 073 709 551 615
26. (a) 216 = 64K
(b) 224 = 16M
(c) 232 = 4096 Mega, 4G
(d) 248 = 256 Tera, 262144 Giga, 268435456 Mega
27. Data bus is bidirectional, address bus is unidirectional (exit CPU).
28. PC (Program Counter)
29. ALU is responsible for all arithmetic and logic calculations in the CPU.
30. Address, control and data

CHAPTER 1: THE x86 MICROPROCESSOR

SECTION 1.1: BRIEF HISTORY OF THE x86 FAMILY

1. 8086

2. the internal data bus of the 386SX is 32 bits, whereas
the internal data bus of the 286 is 16 bits

3. terms such as "16-bit" or "32-bit" microprocessors refer
to the internal data bus and register size of the microprocessor

4. yes

5. upward compatibility means that any program written
for a lower (earlier) system will run on more advanced (later) systems

6. the 8088 has an 8-bit external data bus but the 8086
has a 16-bit external data bus

7. the 8088 has a 4-byte queue, the 8086 has a 6-byte queue

SECTION 1.2: INSIDE THE 8088/86

8. more efficient internal architecture such as pipelining
and wider registers

9. the BIU (bus interface unit) fetches instructions into
the CPU and the EU (execution unit) executes the instruction

10. (a) 8-bit registers are: AH, AL, BH, BL, CH, CL, DH,CL
(b) 16-bit registers are: AX, BX, CX, DX

11. (a) CS (c) DS (d) SS (h) SI (i) DI

SECTION 1.3: INTRODUCTION TO ASSEMBLY PROGRAMMING

12. (b) is illegal since the value is too large
(c) is illegal since immediate addressing is not allowed for segment registers
(f) is illegal since immediate addressing is not allowed for segment registers
(i) is illegal since the operand types do not match
(j) is illegal since the value is too large for the register
(k) is illegal since the register sizes do not match
(l) is illegal since the operand sizes do not match

SECTION 1.4: INTRODUCTION TO PROGRAM SEGMENTS

13. CS is the code segment register and holds the segment address for the code section
DS is the data segment register and holds the segment address for the data section
SS is the stack segment register and holds the segment address for the stack section
ES is the extra segment register and holds the segment address for the extra segment
which is used for many string operations

14. (a) 3499:2500 (b) 36E90 (c) 34990 to 4498F

15. (a) 1296:0100 (b) 12A60 (c) 12960 to 2295F

16. (a) 38949 (b) 3499:3FB9 (c) 34990 to 4498F

17. (a) 1A648 (b) 1298:7CC8 (c) 12980 to 2297F

18. 0042:004C

19. no, because the upper range of the code segment would be 36FFF
CS should be 3777

20.

12B0:0170	12C70	B0
12B0:0171	12C71	76
12B0:0172	12C72	B7
12B0:0173	12C73	8F
12B0:0174	12C74	00
12B0:0175	12C75	C7
12B0:0176	12C76	80
12B0:0177	12C77	C7
12B0:0178	12C78	7B
12B0:0179	12C79	88
12B0:017A	12C7A	FB
12B0:017B	12C7B	00
12B0:017C	12C7C	C3

21.

12B0:0100	12C00	B0
12B0:0101	12C01	00
12B0:0102	12C02	02
12B0:0103	12C03	06
12B0:0104	12C04	00
12B0:0105	12C05	02
12B0:0106	12C06	02
12B0:0107	12C07	06
12B0:0108	12C08	01
12B0:0109	12C09	02
12B0:010A	12C0A	02
12B0:010B	12C0B	06
12B0:010C	12C0C	02
12B0:010D	12C0D	02
12B0:010E	12C0E	02
12B0:010F	12C0F	06
12B0:0110	12C10	03
12B0:0111	12C11	02
12B0:0112	12C12	02
12B0:0113	12C13	06
12B0:0114	12C14	04
12B0:0115	12C15	02

SECTION 1.5: THE STACK

22. (b)

23. (c)

24. decremented, incremented

25. (b)

26. the stack is slower than registers, since the stack is a section of RAM

27. (a) 24578 (b) 2000:4578 (c) 20000 (d) 2FFFF

28. 24FB

29. after "PUSH AX", the stack pointer = FF2C and the stack is as follows:

logical address	stack contents
SS:FF2C	91
SS:FF2D	32

after "PUSH BX", the stack pointer = FF2A and the stack is as follows:

logical address	stack contents
SS:FF2A	3C
SS:FF2B	F4
SS:FF2C	91
SS:FF2D	32

after "PUSH CX", the stack pointer = FF28 and the stack is as follows:

logical address	stack contents
SS:FF28	09
SS:FF29	00
SS:FF2A	3C
SS:FF2B	F4
SS:FF2C	91
SS:FF2D	32

30. at the conclusion of Problem 28, SP = FF28,

```
POP CX      ;then SP = FF2A
POP BX      ;then SP = FF2C
POP AX      ;then SP = FF2E
```

31. (a) SS (b) DS (c) CS (d) DS
(e) SS (f) DS

32. (a) SS overrides default register DS
(b) SS overrides default register DS
(c) DS overrides default register SS

SECTION 1.6: FLAG REGISTER

33. (a) CF = 1 indicating a carry occurred
PF = 1 indicating even parity
AF = 1 indicating a carry from bit 3
ZF = 1 indicating the result is zero
SF = 0 indicating a positive result

(b) CF = 0 indicating no carry

PF = 0 indicating odd parity
AF = 0 indicating no carry from bit 3
ZF = 0 indicating that the result is not zero
SF = 1 indicating negative result

(c) CF = 0 indicating no carry
PF = 1 indicating even parity
AF = 1 indicating a carry from bit 3
ZF = 0 indicating the result is not zero
SF = 0 indicating positive result

SECTION 1.7: x86 ADDRESSING MODES

34. (a) location 24000 (20000 + 4000) contains FF
(b) location 2A088 (20000 + 4000 + 6080 + 8) contains 25
(c) location 26080 (20000 + 6080) contains FF
location 26081 contains 25
(d) location 25006 (20000 + 5000 + 6) contains 80
location 25007 contains 60
(e) location 2B0A8 (20000 + 5000 + 6080 + 28) contains 91
location 2B0A9 contains 87
(f) location 34010 (30000 + 4000 + 10) contains 99
location 34011 contains 12
(g) location 23600 (20000 + 3600) contains FF
location 23601 contains 25
(h) location 260B0 (20000 + 6080 + 30) contains 99
location 260B1 contains 12
(i) location 37200 (30000 + 7000 + 200) contains FF
location 37201 contains 25
(j) location 3B100 (30000 + 7000 + 4000 + 100) contains 80
location 3B101 contains 60
(k) location 24050 (20000 + 4000 + 50) contains 25
(l) location 2C100 (20000 + 5000 + 7000 + 100) contains FF
location 2C101 contains 25

35. (a) register
(b) immediate
(c) direct
(d) register
(e) register indirect
(f) register indirect
(g) based index
(h) register
(i) based
(j) based index
(k) index
(l) based index

36. (a) DS:1450 contains 9F
DS:1451 contains 12
(b) DS:2348 contains 63
DS:2349 contains 8C

CHAPTER 2: ASSEMBLY LANGUAGE PROGRAMMING

SECTION 2.1: DIRECTIVES AND A SAMPLE PROGRAM
SECTION 2.2: ASSEMBLE, LINK, AND RUN A PROGRAM
SECTION 2.3: MORE SAMPLE PROGRAMS

1.

```
            .MODEL SMALL
            .STACK 64
            .DATA

            ORG     10H
DATA_IN     DW      2525H,4FFFH,8555H,1F00H,2BBBH,0C4H
            ORG     28H
COPY        DW      6 DUP(?)

            ;------------
            .CODE
MAIN        PROC    FAR
            MOV     AX,@DATA
            MOV     DS,AX
            MOV     SI,OFFSET DATA_IN       ;SI points to data to be copied
            MOV     DI,OFFSET COPY          ;DI points to copy of data
            MOV     CX,06H                  ;loop counter = 6
MOV_LOOP:   MOV     AX,[SI]                 ;move the next word area to AX
            MOV     [DI],AX                 ;move the next word to COPY area
            INC     SI                      ;increment DATA pointer
            INC     SI
            INC     DI                      ;increment COPY pointer
            INC     DI
            DEC     CX                      ;decrement LOOP counter
            JNZ     MOV_LOOP                ;jump if loop counter not zero
            MOV     AH,4CH                  ;set up to return
            INT     21H                     ;return to DOS
MAIN        ENDP
            END     MAIN
```

2. first the source file (extension "asm") must be produced with a word processor which produces an ASCII file
then the program is assembled to produce the object (extension "obj") file
then the program is linked to produce the executable (extension "exe") file

3. the linker program

4. the assembler program

5. false

7. after

SECTION 2.4: CONTROL TRANSFER INSTRUCTIONS

8. when the procedure is called, IP (which points to the next instruction to be executed after the CALL) is saved on the stack since it is a NEAR procedure. After the CALL and all PUSH instructions have been executed, the stack is as follows with SP = 1278.

1278 <- flag register
127A <- DI

127C <- SI
127E <- DX
1280 <- CX
1282 <- BX
1284 <- AX 1285 = (AH) 1284 = (AL)
1286 <- IP 1287 = (04) 1286 = (53)
1288

9. SP = 1278

```
POPF        ;now SP = 127A
POP DI      ;now SP = 127C
POP SI      ;now SP = 127E
POP DX      ;now SP = 1280
POP CX      ;now SP = 1282
POP BX      ;now SP = 1284
POP AX      ;now SP = 1286
;SP = 1288 after the RET
```

10. the address of the instruction immediately following the CALL is stored on the stack. The last instruction of a called subroutine must be RET in order to tell the system to pop off the return address from the stack.

11. CS and IP, IP

12. NEAR calls require two bytes to store IP
FAR calls require four bytes to store CS and IP

13. IP = 673D will be stored in the stack at 1295 and 1294, therefore
SS:1295 = 67 and SS:1294 = 3D

14. (a) 3F (displacement) + 6E (instruction after JNC) = E0AD, the offset of ERROR1
(b) 39 (displacement) + 74 (instruction after JNO) = E0AD,the offset of ERROR1
(c) E3 (displacement) + A9 (instruction after JMP) = E08C, the offset of C8

SECTION 2.5: DATA TYPES AND DATA DEFINITION

15. the following notation indicates "offset location: byte contents of location"

for DATA1			
0020:31	0021:2D	0022:38	0023:30
0024:30	0025:2D	0026:35	0027:35
0028:35	0029:2D	002A:31	002B:32
002C:33	002D:34		
for DATA2			
0040:4E	0041:61	0042:6D	0043:65
0044:3A	0045:20	0046:4A	0047:6F
0048:68	0049:6E	004A:20	004B:4A
004C:6F	004D:6E	004E:65	004F:73
for DATA3			
0060:35	0061:39	0062:35	0063:36
0064:33	0065:34	0066:32	
for DATA4			
0070:60	0071:25	0072:06	0073:10
for DATA5			
0074:31	0075:00		
for DATA6			
0080:6E	0081:7F	0082:69	0083:25

for DATA7

0084:F2	0085:99	0086:1C	0087:A2
0088:7B	0089:9E		

for DATA8

0090:28	0091:98	0092:99	0093:24
0094:79	0095:99	0096:39	0097:04
0098:00	0099:00		

for DATA9

009A:EE	009B:EE	009C:EE	009D:EE
009E:EE	009F:EE		

16.

```
TITLE     PROBLEM  (EXE)   PROBLEM 16 PROGRAM
PAGE      60,132
          .MODEL  SMALL
          .STACK   32
;-----------------------------
          .DATA
DATA      DW       234DH,0DE6H,3BC7H,566AH        ;leading zero
          ORG      10H
SUM       DW       ?
;-----------------------------

          .CODE

START     PROC  FAR                  ;no colon after START
          MOV  AX,DATA               ;should be @DATA
          MOV  DS,AX
          MOV  CX,04
          MOV  BX,0
          MOV  DI,OFFSET DATA
LOOP1:    ADD  BX,[DI]
          INC  DI
          INC  DI
          DEC  CX                    ;DEC CX NOT BX
          JNZ  LOOP1
          MOV  SI,OFFSET SUM         ;SUM  instead of RESULT
          MOV  [SI],BX
          MOV  AH,4CH
          INT  21H
START     ENDP
          END  START                 ;START not STRT
```

CHAPTER 3: ARITHMETIC AND LOGIC INSTRUCTIONS AND PROGRAMS

SECTION 3.1: UNSIGNED ADDITION AND SUBTRACTION

1. (a) BH = 84, CF = 0 (no carry), ZF = 0 (result not zero), AF = 1 (auxiliary carry)
(b) CX = 79F1, CF = 0 (no carry), ZF = 0 (result not zero),AF = 1 (auxiliary carry)
(c) AX = 0100, CF = 0 (no carry), ZF = 0 (result not zero),AF = 1 (auxiliary carry)
(d) BL = 00, CF = 1 (carry), ZF = 1 (result is zero),AF = 1 (auxiliary carry)
(e) CX = 0000, CF = 1 (carry), ZF = 1 (result is zero),AF = 1 (auxiliary carry)
(f) AH = FF, CF = 0 (no carry), ZF = 0 (result not zero), AF = 0 (no auxiliary carry)

2. The data is stored with directive DW. The program will loop 8 times to add the 8 words, and the loop count is set in data item COUNT, then moved to CX. SI is used to point to the data. As each word is added to AX, any carry that was generated is added to BX, then the result is stored with BX as the higher word and AX as the lower word. In this particular example, the result of the addition is 5CE4H, no carries were generated so the higher word (BX) = 0.

```
TITLE           PROB2 (EXE) ADDING 8 WORDS
PAGE            60,132
STSEG           SEGMENT
DB 64 DUP (?)
STSEG           ENDS
;-----------------------------
DTSEG           SEGMENT
COUNT           EQU  08
DATA            DW   2300,4300,1200,3700,1298,4323,5673,986
                ORG  0010H
SUM             DW   2 DUP(?)
DTSEG           ENDS
;-----------------------------
CDSEG           SEGMENT
MAIN            PROC FAR
                ASSUME CS:CDSEG,DS:DTSEG,SS:STSEG
                MOV AX,DTSEG
                MOV DS,AX
                MOV CX,COUNT              ;CX is the loop counter
                MOV SI,OFFSET DATA        ;SI is the data pointer
                MOV AX,00                 ;AX will hold the sum
                MOV BX,AX                 ;BX will hold the carries
BACK:           ADD AX,[SI]               ;add the next word to AX
                ADC BX,0                  ;add carry to BX
                INC SI                    ;increment data pointer twice
                INC SI                    ; to point to next word
                DEC CX                    ;decrement loop counter
                JNZ BACK                  ;if not finished, continue adding
                MOV SUM,AX                ;store the sum
                MOV SUM+2,BX              ;store the carries
                MOV AH,4CH
                INT 21H                   ;go back to DOS
MAIN            ENDP
CDSEG           ENDS
                END MAIN
```

3. In order to rewrite Program 3-2 for multibyte addition, the following changes were made:
(1) CX was changed from 4 (to add 4 words) to 8 (to add 8 bytes)
(2) AL was used to accumulate the sum of the two bytes instead of AX
(3) BYTE PTR was used to make the operands match in size
(4) all pointers SI, DI and BX were incremented once to point to the next byte instead of twice to point to the next word

   The result of the addition is 945D09387874H.

```
TITLE     PROB3 (EXE) MULTIBYTE ADDITION
PAGE      60,132
STSEG     SEGMENT
          DB 64 DUP (?)
STSEG     ENDS
;-----------------------------
DTSEG     SEGMENT
DATA1     DQ   548FB9963CE7H
          ORG  0010H
DATA2     DQ   3FCD4FA23B8DH
          ORG  0020H
DATA3     DQ   ?
DTSEG     ENDS
;-----------------------------
CDSEG     SEGMENT
MAIN      PROC FAR
          ASSUME CS:CDSEG,DS:DTSEG,SS:STSEG
          MOV AX,DTSEG
          MOV DS,AX
          CLC                       ;clear carry before first addition
          MOV SI,OFFSET DATA1       ;SI is pointer for operand1
          MOV DI,OFFSET DATA2       ;DI is pointer for operand2
          MOV BX,OFFSET DATA3       ;BX is pointer for the sum
          MOV CX,08                 ;CX is the loop counter
BACK:     MOV AL,BYTE PTR[SI]       ;move the first operand to AX
          ADC AL,BYTE PTR [DI]      ;add the second operand to AX
          MOV BYTE PTR [BX],AL      ;store the sum
          INC SI                    ;point to next byte of operand1
          INC DI                    ;point to next byte of operand2
          INC BX                    ;point to next byte of sum
          LOOP BACK                 ;if not finished, continue adding
          MOV AH,4CH
          INT 21H                   ;go back to DOS
MAIN      ENDP
CDSEG     ENDS
          END MAIN
```

4. This program subtracts one 8-byte number from another. The subtraction is performed one byte at a time, starting at the least significant byte. SBB is used to take care of any borrow that occurred from the previous byte subtraction. The result of the subtraction is 14C269F4015AH.

   The program is on the following page.

```
TITLE     PROB4 (EXE) MULTIBYTE SUBTRACTION
PAGE      60,132
STSEG     SEGMENT
          DB 64 DUP (?)
STSEG     ENDS
;-----------------------------
DTSEG     SEGMENT
DATA1     DQ   548FB9963CE7H
          ORG  0010H
DATA2     DQ   3FCD4FA23B8DH
          ORG  0020H
DATA3     DQ   ?
DTSEG     ENDS
;-----------------------------
CDSEG     SEGMENT
MAIN      PROC FAR
          ASSUME CS:CDSEG,DS:DTSEG,SS:STSEG
          MOV AX,DTSEG
          MOV DS,AX
          CLC                        ;clear carry before first subtraction
          MOV SI,OFFSET DATA1        ;SI is pointer for operand1
          MOV DI,OFFSET DATA2        ;DI is pointer for operand2
          MOV BX,OFFSET DATA3        ;BX is pointer for the result
          MOV CX,08                  ;CX is the loop counter
BACK:     MOV AL,BYTE PTR [SI]       ;move the first operand to AX
          SBB AL,BYTE PTR [DI]       ;add the second operand to AX
          MOV BYTE PTR [BX],AL       ;store the result
          INC SI                     ;point to next byte of operand1
          INC DI                     ;point to next byte of operand2
          INC BX                     ;point to next byte of sum
          LOOP BACK                  ;if not finished, continue subtracting
          MOV AH,4CH
          INT 21H                    ;go back to DOS
MAIN      ENDP
CDSEG     ENDS
          END MAIN
```

5. The three steps are:
(1) convert the subtrahend to two's complement
(2) add the two numbers
(3) invert the carry

(a) 23H	0010 0011	0010 0011	
- 12H	0001 0010	1110 1110	step 1: two's complement
11H		0001 0001	step2 : add the two
		step 3: invert carry CF=0	
(b) 43H	0100 0011	0100 0011	
- 51H	0101 0001	1010 1111	step1: two's complement
F2H		1111 0010	step 2: add the numbers
		step 3: invert carry CF = 1	
(c) 99	0110 0011	0110 0011	
- 99	0110 0011	1001 1101	step 1:two's complement
00		0000 0000	step 2: add the numbers
		step 3: invert carry CF = 0	

SECTION 3.2: UNSIGNED MULTIPLICATION AND DIVISION

6. (1) parts a, b, and c were combined into one program, shown below:

   (a) byte1 x byte2 BYTE1 is moved into AL, then AL is multiplied by BYTE2,the product (59D8H) is left is AX.

   (b) byte1 x word1 BYTE1 is moved into AL and AH is set to zero, then AX is multiplied by WORD1 and the product, 231694H, is in DX AX, DX = 0023 and AX = 1694.

   (c) word1 x word2 WORD1 is moved into AX, then AX is multiplied by WORD2 and the product, 2DC468H, is in DX AX, DX = 002D and AX = C468.

```
TITLE     PROB61 (EXE) MULTIPLICATION
PAGE      60,132
STSEG     SEGMENT
          DB 64 DUP (?)
STSEG     ENDS
;-----------------------------
DTSEG     SEGMENT
BYTE1     DB   230
BYTE2     DB   100
WORD1     DW   9998
WORD2     DW   300
DTSEG     ENDS
;-----------------------------
CDSEG     SEGMENT
MAIN      PROC FAR
          ASSUME CS:CDSEG,DS:DTSEG,SS:STSEG
          MOV AX,DTSEG
          MOV DS,AX
          ;(a) byte1 x byte2
          MOV AL,BYTE1
          MUL BYTE2              ;RESULT IN AX
          ;(b) byte1 x word1
          SUB AH,AH
          MOV AL,BYTE1
          MUL WORD1              ;RESULT IN DX AX
          ;(c) word1 x word2
          MOV AX,WORD1
          MUL WORD2              ;RESULT IN DX AX
          MOV AH,4CH
          INT 21H                ;go back to DOS
MAIN      ENDP
CDSEG     ENDS
          END MAIN
```

(2) parts a, b, and c were combined into one program, shown below:

(a) BYTE1 is moved into AL and AH is set to zero, then AL is divided by BYTE2 and the result (2H) is in AL, with the remainder (0) in AH.

(b) WORD1 is moved into AX and DX is set to zero, then AX is divided by WORD2and the result (21) is in AX and the remainder (62)is in DX.

(c) DBLWORD is moved into DX AX with DX = 0001 and AX = 86A0, then DX AX is divided by BYTE1 which is moved into BL (36H) with BH set to zero;
the result (1B2H) is in AX with DX = B4H, the remainder.

```
TITLE       PROB62 (EXE) DIVISION
PAGE        60,132
STSEG       SEGMENT
            DB 64 DUP (?)
STSEG       ENDS
;-----------------------------
DTSEG       SEGMENT
BYTE1       DB    230
BYTE2       DB    100
WORD1       DW    9998
WORD2       DW    300
DBLWRD      DD    100000
DTSEG       ENDS
;-----------------------------
CDSEG       SEGMENT
MAIN        PROC FAR
            ASSUME CS:CDSEG,DS:DTSEG,SS:STSEG
            MOV AX,DTSEG
            MOV DS,AX
            ;(a) byte1 / byte2
            MOV AL,BYTE1
            SUB AH,AH
            DIV BYTE2                   ;QUOTIENT IN AL     REM IN AH
            ;(b) word1 / word2
            MOV AX,WORD1
            SUB DX,DX
            DIV WORD2                   ;QUOTIENT IN AX   REM IN DX
            ;(c) dblwrd / byte1
            MOV SI,OFFSET DBLWRD
            MOV AX,[SI]
            MOV DX,[SI+2]
            SUB BH,BH
            MOV BL,BYTE1
            DIV BX                      ;QUOTIENT IN AX   REM IN DX
            MOV AH,4CH
            INT 21H                     ;go back to DOS
MAIN        ENDP
CDSEG       ENDS
            END MAIN
```

SECTION 3.3: LOGIC INSTRUCTIONS

7. (a) DX = E000 CF = 1 ZF = 0
(b) DH = F7 CF = 0 ZF = 0
(c) AL = 76 CF = 0 ZF = 0
(d) DX = E390 CF = 0 ZF = 0
(e) AX = 0 CF = 0 ZF = 1
(f) BX = F7D6 CF = 0 ZF = 0
(g) AH = F0 CF = 0 ZF = 0
(h) AX = F999 CF = 0 ZF = 0
(i) DX = 0D7E CF = 0 ZF = 0
(j) BX = 0000 CF = 0 ZF = 1
(k) AL = 00 CF = 0 ZF = 1
(l) DX = 71C8 CF = 0 ZF = 0
(m) DL = 12 CF = 0 ZF = 0
(n) BX = 8AC0 CF = 0 ZF = 0
(o) DX = E400 CF = 0 ZF = 0

8. (a) CF = 0 and ZF = 0, because 2500 > 1400
(b) CF = 0 and ZF = 0, because FF > 6F
(c) CF = 1 and ZF = 0, because 34 < 88
(d) CF = 0 and ZF = 1, because 0 = 0
(e) CF = 1 and ZF = 0, because 0 < FFFF
(f) CF = 0 and ZF = 1, because FFFF = FFFF
(g) CF = 0 and ZF = 0, because 4000 > 2378
(h) CF = 0 and ZF = 1, because 0 = 0

9. (a) yes, because no carry was generated by shifting zero
(b) no, because no carry was generated by shifting left 6D once
(c) no, because a carry was generated by shifting left 55H 5 times

10.

```
TITLE     PROB10 (EXE)
PAGE      60,132
STSEG     SEGMENT
          DB 64 DUP (?)
STSEG     ENDS
DTSEG     SEGMENT
GRADES    DB   69,87,96,45,75
          ORG  0008
LOWEST    DB   ?
DTSEG     ENDS
CDSEG     SEGMENT
MAIN      PROC FAR
          ASSUME CS:CDSEG,DS:DTSEG,SS:STSEG
          MOV AX,DTSEG
          MOV DS,AX
          MOV CX,5                    ;set up loop counter
          MOV BX,OFFSET GRADES        ;BX points to GRADE data
          MOV AL,0FFH                 ;AL holds lowest grade found so far
AGAIN:    CMP AL,[BX]                 ;compare next grade to lowest
          JB  NEXT                    ;jump if AL still lowest
          MOV AL,[BX]                 ;else AL holds new lowest
NEXT:     INC BX                      ;point to next grade
          LOOP AGAIN                  ;continue search
          MOV LOWEST,AL               ;store lowest grade
          MOV AH,4CH
          INT 21H                     ;go back to dos
MAIN      ENDP
```

```
CDSEG   ENDS
        END  MAIN
```

The following changes were made to Program 3-3 to find the lowest grade:
(1) name HIGHEST was changed to LOWEST
(2) AL was initialized to FF instead of 00
(3) JA (jump above, or jump no carry) was changed to JB (jump below, or jump carry)

11.

```
TITLE   PROB11 (EXE) UPPER TO LOWER CASE CONVERT
PAGE    60,132
STSEG   SEGMENT
        DB 64 DUP (?)
STSEG   ENDS
;-----------------------------
DTSEG   SEGMENT
DATA1   DB 'mY NAME is jOe'
        ORG 0020H
DATA2   DB 14 DUP(?)
DTSEG   ENDS
;-----------------------------
CDSEG   SEGMENT
MAIN    PROC FAR
        ASSUME CS:CDSEG,DS:DTSEG,SS:STSEG
        MOV AX,DTSEG
        MOV DS,AX
        MOV SI,OFFSET DATA1     ;SI points to original data
        MOV BX,OFFSET DATA2     ;BX points to lower case data
        MOV CX,14               ;CX is loop counter
BACK:   MOV AL,[SI]             ;get next character
        CMP AL,5AH              ;if greater than 'Z'
        JA  OVER                ;then no need to convert
        CMP AL,41H              ;if less than 'A'
        JB  OVER                ;then no need to convert
        OR AL,00100000B         ;make d5 = 1 to convert to lower case
OVER:   MOV [BX],AL             ;store lower case character
        INC SI                  ;increment pointer to original
        INC BX                  ;increment pointer to lower case data
        LOOP BACK               ;continue looping if CX 0
        MOV AH,4CH
        INT 21H                 ;go back to dos
MAIN    ENDP
CDSEG   ENDS
        END MAIN
```

The following changes were made to Program 3-4 to convert to lower case:
(1) the first compare was changed to check if the byte is greater than 'Z'
(2) the jump after the compare was changed to JA
(3) the second compare was changed to check if the byte is less than 'A'
(4) the jump after the second compare was changed to JB
(5) if the number was to be converted, the OR would make bit 5 one

12. Notice that ERRO1 is at offset E0AD, C8 is at offset E08C, C9 is at offset E0A9. The displacement is calculated
by subtracting the offset of the instruction immediately after the jump from the target offset

(a)	E05F 734C	313	JNC	ERRO1
	E0AD - E061 = 4C			
(b)	E061 75 4A	314	JNZ	ERRO1
	E0AD - E063 = 4A			
(c)	E063 7B48	315	JNP	ERRO1
	E0AD - E065 = 48			
(d)	E065 7946	316	JNS	ERRO1
	E0AD - E067 = 46			
(e)	E06C 733F	320	JNC	ERRO1
	E0AD - E06E = 3F			
(f)	E072 7139	323	JNO	ERRO1
	E0AD - E074 = 39			
(g)	E077 7634	326	JBE	ERRO1
	E0AD - E079 = 34			
(h)	E079 7832	328	JS	ERRO1
	E0AD - E07B = 32			
(i)	E07B 7A30	329	JP	ERRO1
	E0AD - E07D = 30			
(j)	E082 7229	333	JC	ERRO1
	E0AD - E084 = 29			
(k)	E086 7025	335	JO	ERRO1
	E0AD - E088 = 25			
(l)	E0A0 7307	353	JNC	C9
	E0A9 - E0A2 = 07			
(m)	E0A4 7507	355	JNZ	ERRO1
	E0AD - E0A6 = 07			
(n)	E0A7 EBE3	357	JMP	C8
	E08C - E0A9 = FFE3			

SECTION 3.4: BCD AND ASCII CONVERSION

13. The following changes were made to Program 3-7:
(1) data name DATA5_ADD was changed to DATA5_SUB
(2) procedure name BCD_ADD was changed to BCD_SUB
(3) within procedure BCD_SUB, ADC was changed to SBB
(4) within procedure BCD_SUB, DAA was changed to DAS

```
TITLE PROGB13 (EXE) ASCII - BCD CONVERSION AND SUBTRACTION
PAGE            60,132
STSEG           SEGMENT
            DB 64 DUP(?)
STSEG           ENDS
;-----------------------------
DTSEG           SEGMENT
DATA1_ASC       DB   '0649147816'
                ORG  0010H
DATA2_ASC       DB   '0072687188'
                ORG  0020H
DATA3_BCD       DB   5 DUP (?)
                ORG  0028H
DATA4_BCD       DB   5 DUP (?)
                ORG  0030H
DATA5_SUB       DB   5 DUP (?)
                ORG  0040H
DATA6_ASC       DB   10 DUP (?)
DTSEG     ENDS
```

```
;-----------------------------
CDSEG           SEGMENT
MAIN            PROC FAR
                ASSUME CS:CDSEG,DS:DTSEG,SS:STSEG
                MOV AX,DTSEG
                MOV DS,AX
                MOV BX,OFFSET DATA1_ASC     ;BX points to first ASCII data
                MOV DI,OFFSET DATA3_BCD      ;DI points to first BCD data
                MOV CX,10                   ;CX holds number bytes to convert
                CALL CONV_BCD               ;convert ASCII to BCD
                MOV BX,OFFSET DATA2_ASC     ;BX points to second ASCII data
                MOV DI,OFFSET DATA4_BCD     ;DI points to second BCD data
                MOV CX,10                   ;CX holds number bytes to convert
                CALL CONV_BCD               ;convert ASCII to BCF
                CALL BCD_SUB                ;subtract the BCD operands
                MOV SI,OFFSET DATA5_SUB     ;SI points to BCD result
                MOV DI,OFFSET DATA6_ASC     ;DI points to ASCII result
                MOV CX,05                   ;CX holds count for convert
                CALL CONV_ASC               ;convert result to ASCII
                MOV AH,4CH
                INT 21H                     ;go back to DOS
MAIN            ENDP
;-----------------------------
;THIS SUBROUTINE CONVERTS ASCII TO PACKED BCD
CONV_BCD PROC
AGAIN:          MOV AX,[BX]                 ;BX=pointer for ASCII data
                XCHG AH,AL
                AND AX,0F0FH                ;mask ASCII 3s
                PUSH CX                     ;save the counter
                MOV CL,4                    ;shift AH left 4 bits
                SHL AH,CL                   ; to get ready for packing
                OR AL,AH                    ;combine to make packed BCD
                MOV [DI],AL                 ;DI=pointer for BCD data
                ADD BX,2                    ;point to next 2 ASCII bytes
                INC DI                      ;point to next BCD data
                POP CX                      ;restore loop counter
                LOOP AGAIN
                RET
CONV_BCD ENDP
;-----------------------------
;THIS SUBROUTINE SUBTRACTS TWO MULTIBYTE PACKED BCD OPERANDS
BCD_SUB         PROC
                MOV BX,OFFSET DATA3_BCD     ;BX=pointer for operand 1
MOV DI,OFFSET DATA4_BCD                     ;DI=pointer for operand 2
                MOV SI,OFFSET DATA5_SUB     ;SI=pointer for sum
                MOV CX,05
                CLC
BACK:           MOV AL,[BX]+4               ;get next byte of operand 1
                SBB AL,[DI]+4               ;subtract next byte of operand 2
                DAS                         ;correct for BCD subtraction
                MOV [SI] +4,AL              ;save difference
                DEC BX                      ;point to next byte of operand 1
                DEC DI                      ;point to next byte of operand 2
                DEC SI                      ;point to next byte of difference
                LOOP BACK
                RET
BCD_SUB         ENDP
;-----------------------------
;THIS SUBROUTINE CONVERTS FROM PACKED BCD TO ASCII
CONV_ASC        PROC
AGAIN2:         MOV AL,[SI]                 ;SI=pointer for BCD data
                MOV AH,AL                   ;duplicate to unpack
                AND AX,0F00FH               ;unpack
                PUSH CX                     ;save counter
                MOV CL,04                   ;shift right 4 bits to unpack
                SHR AH,CL
                OR AX,3030H                 ;make it ASCII
                XCHG AH,AL                  ;swap for ASCII storage convention
                MOV [DI],AX                 ;store ASCII data
                INC SI                      ;point to next BCD data
                ADD DI,2                    ;point to next ASCII data
                POP CX                      ;restore loop counter
                LOOP AGAIN2
```

```
                RET
CONV_ASC        ENDP
CDSEG           ENDS
                END MAIN
```

14. The following changes were made to Program 3-8:
(1) procedure name ASC_ADD was changed to ASC_SUB
(2) within ASC_SUB, ADC was changed to SBB
(3) within ASC_SUB, AAA was changed to AAS

Program changes are highlighted below:

```
TITLE       PROB14 (EXE) SUBTRACTING ASCII NUMBERS
PAGE        60,132
STSEG       SEGMENT
            DB 64 DUP (?)
STSEG       ENDS
 ;-----------------------------
DTSEG       SEGMENT
VALUE1      DB '0999999999'
            ORG 0010H
VALUE2      DB '0077777775'
            ORG 0020H
RESULT1     DB 10 DUP (?)
            ORG 0030H
RESULT2     DB 10 DUP (?)
DTSEG       ENDS
;-----------------------------
CDSEG       SEGMENT
MAIN        PROC FAR
            ASSUME CS:CDSEG,DS:DTSEG,SS:STSEG
            MOV AX,DTSEG
            MOV DS,AX
            CALL ASC_SUB            ;call ASCII subtraction subroutine
            CALL CONVERT            ;call convert to ascii subroutine
            MOV AH,4CH
            INT 21H                 ;go back to DOS
MAIN        ENDP
 ;-------------------------------
 ;THIS SUBROUTINE SUBTRACTS ASCII NUMBERS, RESULT IS  UNPACKED
ASC_SUB     PROC
            CLC                     ;clear the carry
            MOV CX,10               ;set up loop counter
            MOV BX,9                ;point to LSD
BACK:       MOV AL,VALUE1[BX]       ;move next byte of operand 1
            SBB AL,VALUE2[BX]       ;add next byte of operand 2
            AAS                     ;adjust to make it ASCII
            MOV RESULT1[BX],AL      ;store ASCII sum
            DEC BX                  ;point to next byte
            LOOP BACK
            RET
ASC_SUB     ENDP
;-----------------------------
;THIS SUBROUTINE CONVERTS UNPACKED BCD TO ASCII
CONVERT     PROC
            MOV BX,OFFSET RESULT1   ;BX points to ASCII data
            MOV SI,OFFSET RESULT2   ;SI points to BCD data
            MOV CX,05               ;CX is loop counter
BACK2:      MOV AX,WORD PTR [BX]    ;get next 2 ASCII bytes
            OR AX,3030H             ;remove ASCII 3s
            MOV WORD PTR [SI],AX    ;store BCD
            ADD BX,2        ;increment ASCII pointer
            ADD SI,2                ;increment BCD pointer
            LOOP BACK2
            RET
CONVERT     ENDP
CDSEG       ENDS
            END MAIN
```

15. The program follows:

```
TITLE       PROB15 (EXE) ADDING BCD NUMBERS
PAGE        60,132
STSEG       SEGMENT
            DB 64 DUP (?)
STSEG       ENDS
;-----------------------------
DTSEG       SEGMENT
VALUE1      DT 87965141012
ORG 0010H
VALUE2      DT 31610640392
            ORG 0020H
SUM         DT  ?
DTSEG       ENDS
;-----------------------------
CDSEG       SEGMENT
MAIN        PROC FAR
            ASSUME CS:CDSEG,DS:DTSEG,SS:STSEG
            MOV AX,DTSEG
            MOV DS,AX
            CALL BCD_ADD                    ;call BCD addition subroutine
            MOV AH,4CH
            INT 21H                         ;go back to DOS
MAIN        ENDP
;-------------------------------
;THIS SUBROUTINE SUBTRACTS ASCII NUMBERS, RESULT IS  UNPACKED
BCD_ADD     PROC
            CLC                             ;clear the carry
            MOV CX,10                       ;set up loop counter
            MOV BX,0                        ;set pointer to 0
BACK:       MOV AL,BYTE PTR VALUE1[BX]      ;move next byte of operand 1
            ADC AL,BYTE PTR VALUE2[BX]      ;add next byte of operand 2
            DAA                             ;adjust to make it ASCII
            MOV BYTE PTR SUM[BX],AL         ;store ASCII sum
            INC BX                          ;point to next byte
            LOOP BACK
            RET
BCD_ADD     ENDP
CDSEG       ENDS
            END MAIN
```

16. DAA

17.

```
TITLE PROB17 (EXE)
PAGE        60,132
STSEG       SEGMENT
            DB 32 DUP(?)
STSEG       ENDS
;-----------------------------
DTSEG       SEGMENT
COUNT       DB   99
DTSEG       ENDS
;-----------------------------
CDSEG       SEGMENT
START       PROC FAR
            ASSUME CS:CDSEG,DS:DTSEG,SS:STSEG
            MOV AX,DTSEG
            MOV DS,AX
            MOV AL,COUNT  ;move count to AL
BACK:       SUB AL,1                    ;count down by 1
            DAS                         ;adjust for BCD
            CMP AL,00                   ;stop counting when = 0
            JA  BACK
            MOV AH,4CH
            INT 21H
START       ENDP
CDSEG       ENDS
            END  START
```

18.

```
TITLE           PROB18 (EXE)
PAGE            60,132
STSEG           SEGMENT
DB 64 DUP (?)
STSEG           ENDS

;-----------------------------
DTSEG   SEGMENT
GRADES          DB   81,65,77,82,73,55,88,78,51,91,86,76
NUM_GRDS        DW   12
CURVE           DB     ?
                ORG 10H
CURVED_GRD      DB  30 DUP (?)
                ORG  0040H
HIGHEST         DB   ?
DTSEG           ENDS
;-----------------------------
CDSEG   SEGMENT
MAIN            PROC FAR
                ASSUME CS:CDSEG,DS:DTSEG,SS:STSEG
                MOV AX,DTSEG
                MOV DS,AX
                CALL FIND_HIGH
                CALL ADD_CURVE
                MOV AH,4CH
                INT 21H
MAIN            ENDP
;-------------------------------------
FIND_HIGH       PROC
                MOV CX,NUM_GRDS             ;set up loop counter
                MOV BX,OFFSET GRADES        ;BX points to GRADE data
                MOV AL,0                    ;AL holds highest grade found so far
AGAIN:          CMP AL,[BX]                 ;compare next grade to highest
                JA NEXT                     ;jump if AL still highest
                MOV AL,[BX]                 ;else AL holds new highest
NEXT:           INC BX                      ;point to next grade
                LOOP AGAIN                  ;continue search
                MOV HIGHEST,AL              ;store highest grade
                RET
FIND_HIGH       ENDP
;-------------------------------------
ADD_CURVE       PROC
                MOV  CX,NUM_GRDS            ;loop count = number of grades
                MOV  AL,99
                SUB  AL,HIGHEST
                MOV  CURVE,AL               ;curve = 99 - highest
                MOV  BX,OFFSET CURVED_GRD   ;BX points to curved grades
                MOV  SI,OFFSET GRADES       ;SI points to original grades
CURVE_LP:       MOV AL,[SI]                 ;get original grade
                ADD AL,CURVE                ;add curve
                MOV [BX],AL                 ;store curved grade
                INC BX                      ;increment curved grade pointer
                INC SI                      ;increment original grade pointer
                LOOP CURVE_LP               ;loop through all grades
                RET
ADD_CURVE       ENDP
CDSEG           ENDS
                END  MAIN
```

19. (a) the quotient (500000 = 7A120H) is too large for register AX
(b) the CPU gives the "divide overflow" error when the program is executed

20. (d)

SECTION 3.5: ROTATE INSTRUCTIONS

21.

```
TITLE PROB21 (EXE)
PAGE            60,132
STSEG           SEGMENT
                DB 32 DUP(?)
STSEG           ENDS
;-----------------------------
DTSEG           SEGMENT
NUM             DW   0000H
COUNT           DW        ?
DTSEG           ENDS
;-----------------------------
CDSEG SEGMENT
START           PROC FAR
                ASSUME CS:CDSEG,DS:DTSEG,SS:STSEG
                MOV AX,DTSEG
                MOV DS,AX
                MOV CX,16               ;loop through 16 bits (1 word)
                CLC
                SUB BX,BX               ;BX will hold count of zeros
                MOV AX,NUM              ;load number into AX
BACK:           SHR AX,1                ;shift rightmost bit to CF
                JC END_LOOP             ;if CF=1, don't increment count
                INC BX                  ;if CF= 0, increment count
END_LOOP:       LOOP BACK               ;loop through all 16 bits
                MOV COUNT,BX
                MOV AH,4CH
                INT 21H
                MAIN   ENDP
CDSEG           ENDS
                END  MAIN
```

22. RCL rotates through the Carry flag. ROL does not.

SECTION 3.6: BITWISE OPERATORS IN THE C LANGUAGE

23.

```
/* Problem 23 */
#include <stdio.h>
#include <conio.h>
main()
{
   clrscr();
   unsigned char data_1 = 0x55;
   unsigned char data_2 = 0xAA;
   unsigned char temp;
   temp=data_1&0x0F;                //masking upper four bits
   printf("\nMasking the upper four bits of %X (hex) we get %X (hex)\n",data_1,temp);
   temp=data_1&data_2;              //ANDing
   printf("The result of %X hex ANDED with %X hex is %X hex\n",data_1,data_2,temp);
   temp=data_1|data_2;               //ORing
   printf("The result of %X hex ORed with %X hex is %X hex\n",data_1,data_2,temp);
   temp= data_1^data_2;             //EX-ORing
   printf("The results of %X hex EX-ORed with %X hex is %X hex\n",data_1,data_2,temp);
   temp=~data_1;                    //INVERTING
   printf("The result of %X hex inverted is %X hex\n",data_1,temp);
   temp=data_2>>2;                  //SHIFTING Right
```

```
   printf("When %X hex is shifted right twice we get %X hex\n",data_2,temp);
   temp=data_1<<4;                    //SHIFTING Left
   printf("When %X hex is shifted left %d times we get %X hex\n",data_1,4,temp);
}
```

24.

```
/* Problem 24 */
#include <stdio.h>
#include <conio.h>
main()
{
   clrscr();
   unsigned char data_1;
   unsigned data_a;
   unsigned data_b;
   unsigned data_c;
   unsigned char data_2;
   unsigned char temp;
   fflush(stdin);
   printf("Enter a Hex data of 00-FF\n");
   scanf("%X",&data_1);
   data_b=data_1;
   printf("you typed in %X",data_1);
   printf("\nEnter another Hex data of 00-FF\n");
   fflush(stdin);
   scanf("%X",&data_2);
   fflush(stdin);
   printf("you typed in %X %X",data_1,data_2);
   temp=data_1&0x0F;       //masking upper four bits
   printf("\n%x",temp);
   printf("\nMasking the upper four bits of %X (hex) we get %X (hex)\n",data_1,temp);
   temp=data_1&data_2;    //ANDing
   printf("The result of %X hex ANDED with %X hex is %X hex\n",data_1,data_2,temp);
   temp=data_1|data_2;    //ORing
   printf("The result of %X hex ORed with %X hex is %X hex\n",data_1,data_2,temp);
   temp= data_1^data_2  ;  //EX-ORing
   printf("The results of %X hex EX-ORed with %X hex is %X hex\n",data_1,data_2,temp);
   temp=~data_1;                      //INVERTING
   printf("The result of %X hex inverted is %X hex\n",data_1,temp);
   temp=data_2>>2;       //SHIFTING Right
   printf("When %X hex is shifted right twice we get %X hex\n",data_2,temp);
   temp=data_1<<4;        //SHIFTING Left
   printf("When %X hex is shifted left %d times we get %X hex\n",data_1,4,temp);
}
```

25.

```
/* Problem 25 */
/*Converts two ASCII bytes to a packed BCD data */
#include <stdio.h>
#include <conio.h>
main()
{
   clrscr();
   unsigned char asci_1= '4';          //ascii 34h
   unsigned char asci_2= '8';          //ascii 38h
   unsigned char bcd;
   unsigned char temp_1,temp_2;
   temp_1= asci_1&0x0f;         //mask upper bits
   temp_2= asci_2&0x0f;         //mask upper bits
   temp_2=temp_2<<4;          //shift left
   bcd=temp_1|temp_2;          //make it a bcd data
   printf("The ASCII %X and %X is packed BCD %Xh ",asci_1,asci_2,bcd);
}
```

26.

```
/* Problem 26 */
/* Get a byte of hex data from user and check D0 and D1 for high*/
#include <stdio.h>
#include <conio.h>
main()
{
   clrscr();
   unsigned char data_1;
   unsigned char temp;
   printf("Enter a Hex data of 00-FF to see if D0 and D1 are high\n");
   scanf("%X",&data_1);
   temp=data_1&0x03;      //mask all but D0 and D1
   switch(temp)
   {
      case 0:
         printf("Neither D0 nor D1 is high");
         break;
      case 1:
         printf("D1 is not high");
         break;
      case 2:
         printf("D0 is not high");
         break;
      default:
         printf("D0 and D1 are both high");
   }
}
```

27.

```
/* Problem 27 */
/* Get a byte of hex data from user and check D0 and D7 for high */
#include <stdio.h>
#include <conio.h>
main()
{
   clrscr();
   unsigned char data_1;
   unsigned char temp;
   printf("Enter a Hex data of 00-FF to see if D0 and D7 are high\n");
   scanf("%X",&data_1);
   temp=data_1&0x81;      //mask all but D0 and D7
   switch(temp)
   {
      case 0:
         printf("Neither D0 nor D7 is high");
         break;
      case 1:
         printf("D7 is not high");
         break;
      case 0x80:
         printf("D0 is not high");
         break;
      default:
         printf("D0 and D7 are both high");
   }
}
```

CHAPTER 4: INT 21H AND INT 10H PROGRAMMING AND MACROS

SECTION 4.1: BIOS INT 10H PROGRAMMING

1. The program follows:

```
TITLE     PROB1 (EXE)
PAGE      60,132
STSEG     SEGMENT
                          DB 64 DUP (?)
STSEG     ENDS
DTSEG     SEGMENT
DTSEG     ENDS
;-------------------------------
CDSEG     SEGMENT
MAIN      PROC FAR
          ASSUME CS:CDSEG,DS:DTSEG,SS:STSEG
          MOV AX,DTSEG
          MOV DS,AX
          CALL CLEAR      ;CLEAR THE SCREEN
          CALL CURSOR     ;SET CURSOR POSITION
          MOV AH,4CH
          INT 21H         ;GO BACK TO DOS
MAIN      ENDP
;-------------------------------
;THIS SUBROUTINE CLEARS THE SCREEN
CLEAR     PROC
          MOV AX,0600H    ;SCROLL SCREEN FUNCTION
          MOV BH,07       ;NORMAL ATTRIBUTE
          MOV CX,0000     ;SCROLL FROM ROW=00,COL=00
          MOV DX,184FH    ;TO ROW=18H,COL=4FH
          INT 10H         ;INVOKE INTERRUPT TO CLEAR SCREEN
          RET
CLEAR     ENDP
;-------------------------------
;THIS SUBROUTINE SETS THE CURSOR
CURSOR    PROC
          MOV AH,02       ;SET CURSOR FUNCTION
          MOV BH,00       ;PAGE 00
          MOV DH,5        ;ROW 5
          MOV DL,12       ;COLUMN 12
          INT 10H         ;INVOKE INTERRUPT TO SET CURSOR POSITION
          RET
CURSOR    ENDP
CDSEG     ENDS
          END MAIN
```

2. The program sets the cursor position to row 10, column 20.

3. Program changes are highlighted below:

```
MOV AH,02
MOV BH,00
MOV DH,14
MOV DL,20
INT 10H
```

4. INT 10H Function 03 puts the current cursor position in register DX in hex. In this example DH = 0C (row = 12) and DL = 0F (column = 15). The program follows:

```
TITLE     PROB4 (EXE)
PAGE      60,132
STSEG     SEGMENT
          DB 64 DUP (?)
STSEG     ENDS
;-----------------------------
DTSEG     SEGMENT

DTSEG     ENDS
;-----------------------------
CDSEG     SEGMENT
MAIN      PROC FAR
          ASSUME CS:CDSEG,DS:DTSEG,SS:STSEG
          MOV AX,DTSEG
          MOV DS,AX
          CALL CURSOR       ;set cursor position
          MOV AH,03         ;get cursor position
          MOV BH,00
          INT 10H
          MOV AH,4CH
          INT 21H           ;go back to dos
MAIN      ENDP
;-------------------------------
;THIS SUBROUTINE SETS THE CURSOR
CURSOR    PROC
          MOV AH,02         ;set cursor function
          MOV BH,00         ;page 00
          MOV DH,12         ;row 12
          MOV DL,15         ;column 15
          INT 10H           ;invoke interrupt to set cursor position
          RET
CURSOR    ENDP
;-------------------------------
CDSEG     ENDS
          END MAIN
```

5. The sequence of instructions does not matter, they simply initialize the registers needed by the interrupt. The function of the interrupt is not performed until the INT instruction is executed.

6. The changes are highlighted below:

```
MOV  AX,0600H
MOV  BH,07
MOV  CX,0000
MOV  DX,184F**H**
INT  10**H**
```

7. The program follows:

```
TITLE     PROB7 (EXE)
PAGE      60,132
STSEG     SEGMENT
          DB 64 DUP (?)
STSEG     ENDS
;-----------------------------
DTSEG     SEGMENT
MESSAGE   DB 'IBM Personal Computer','$'
DTSEG     ENDS
;-----------------------------
CDSEG     SEGMENT
MAIN      PROC FAR
          ASSUME CS:CDSEG,DS:DTSEG,SS:STSEG
          MOV AX,DTSEG
          MOV DS,AX
          CALL CLEAR                ;clear the screen
          CALL CURSOR               ;set cursor position
          CALL DISPLAY              ;display message
          MOV AH,4CH
          INT 21H                   ;go back to dos
MAIN      ENDP
;-----------------------------
;THIS SUBROUTINE CLEARS THE SCREEN
CLEAR     PROC
          MOV AX,0600H              ;scroll screen function
          MOV BH,07                 ;normal attribute
          MOV CX,0000               ;scroll from row=00,col=00
          MOV DX,184FH              ;to row=18h,col=4fh
          INT 10H                   ;invoke interrupt to clear screen
          RET
CLEAR     ENDP
;-----------------------------
;THIS SUBROUTINE SETS THE CURSOR
CURSOR    PROC
          MOV AH,02                 ;set cursor function
          MOV BH,00                 ;page 00
          MOV DH,8                  ;row = 8
          MOV DL,14                 ;column = 14
          INT 10H                   ;invoke interrupt to set cursor position
          RET
CURSOR    ENDP
;-----------------------------
;THIS SUBROUTINE DISPLAYS A STRING ON THE SCREEN
DISPLAY   PROC
          MOV AH,09                 ;display function
          MOV DX,OFFSET MESSAGE     ;dx points to output buffer
          INT 21H                   ;invoke interrupt to display string
          RET
DISPLAY   ENDP
CDSEG     ENDS
          END MAIN
```

SECTION 4.2: DOS INTERRUPT 21H

8. The following shows the memory dump before and after the program was executed. Notice that carriage return (0DH) occupies the 15th location and the string length is stored as 0EH (14). Since the expected string length was given as 15, the system did not allow any input (except carriage return) after 14 characters were typed in.

```
-d 12e0:220 23f
12E0:0220  0F FF FF FF FF FF FF FF-FF FF FF FF FF FF FF FF  ................
12E0:0230  FF 00 00  00 00 00 00  00 -00 00 00 00  00 00 00 00    ................
-g
IBM PC with 80
Program terminated normally
-d 12e0:220 23f
12E0:0220  0F 0E 49 42 4D 20 50 43-20 77 69 74 68 20 38 30 ..IBM PC with 80
12E0:0230  0D 00 00 00 00  00 00 00-00 00 00 00 00 00 00 00   ................
```

9.

```
TITLE           PROB9
PAGE            60,132
STSEG           SEGMENT
                DB 64 DUP (?)
STSEG           ENDS
;-----------------------------
DTSEG           SEGMENT
PROMPT          DB    'What is your name?','$'
BUFFER          DB     20,?,20 DUP (?)   ;buffer for keyed-in data
DTSEG           ENDS
CR              EQU 0DH
LF              EQU 0AH
;-----------------------------
CDSEG           SEGMENT
MAIN            PROC FAR
                ASSUME CS:CDSEG,DS:DTSEG,SS:STSEG
                MOV AX,DTSEG
                MOV DS,AX
                CALL CLEAR              ;clear the screen
                CALL CURSOR1            ;set cursor position
                CALL DISPLAY            ;display prompt
                CALL CURSOR2            ;set cursor position
                CALL GETDATA            ;input a string into buffer
                MOV AH,4CH
                INT 21H                 ;go back to dos
MAIN            ENDP
;-----------------------------
;THIS SUBROUTINE CLEARS THE SCREEN
CLEAR           PROC
                MOV AX,0600H            ;scroll screen function
                MOV BH,07               ;normal attribute
                MOV CX,0000             ;scroll from row=00,col=00
                MOV DX,184FH            ;to row=18h,4fh
                INT 10H                 ;invoke interrupt to clear screen
                RET
CLEAR           ENDP
;-----------------------------
```

```
;THIS SUBROUTINE SETS THE CURSOR
CURSOR1     PROC
            MOV AH,02                   ;set cursor function
            MOV BH,00                   ;page 0
            MOV DL,20                   ;column 20
            MOV DH,15                   ;row 15
            INT 10H                     ;Invoke interrupt to set cursor
            RET
CURSOR1     ENDP
;-----------------------------
;THIS SUBROUTINE SETS THE CURSOR
CURSOR2     PROC
            MOV AH,02                   ;set cursor function
            MOV BH,00                   ;page 0
            MOV DL,20                   ;column 20
            MOV DH,17                   ;row 17
            INT 10H                     ;Invoke interrupt to set cursor
            RET
CURSOR2     ENDP
;-----------------------------
;THIS SUBROUTINE DISPLAYS A STRING ON THE SCREEN
DISPLAY     PROC
            MOV AH,09                   ;display string function
            MOV DX,OFFSET PROMPT        ;dx points to message
            INT 21H                     ;invoke interrupt to display string
            RET
DISPLAY     ENDP
;-----------------------------
;THIS SUBROUTINE PUTS DATA FROM THE KEYBOARD INTO A BUFFER
GETDATA     PROC
            MOV AH,0AH                  ;Input string function
            MOV DX,OFFSET BUFFER        ;dx points to buffer
            INT 21H                     ;invoke interrupt to input string
            RET
GETDATA     ENDP
;-----------------------------
CDSEG       ENDS
            END MAIN
```

10. The program follows:

```
TITLE       PROB10
PAGE        60,132
STSEG       SEGMENT
            DB 64 DUP (?)
STSEG       ENDS
;-----------------------------
DTSEG       SEGMENT
DTSEG       ENDS
;-----------------------------
CDSEG       SEGMENT
MAIN        PROC FAR
            ASSUME CS:CDSEG,DS:DTSEG,SS:STSEG
            MOV AX,DTSEG
            MOV DS,AX
            MOV AH,00                   ;set mode
            MOV AL,04                   ;to medium resolution (320x200)
            INT 10H
            MOV CX,160                  ;middle column = 160
            MOV DX,0                    ;start at top row
            MOV AH,0CH                  ;draw pixel function
            MOV AL,01                   ;pixel value
LINE1:      INT 10H
            INC DX                      ;increment to next row
```

```
                CMP DX,200              ;draw line until row 200
                JNZ LINE1
                MOV CX,00               ;start at first  column
                MOV DX,100              ;draw line on middle row
LINE2:          MOV AH,0CH              ;draw pixel function
                MOV AL,01               ;pixel value
                INT 10H
                INC CX                  ;increment to next column
                CMP CX,320              ;draw unit column 320
                JNZ LINE2
                MOV AH,4CH
                INT 21H                 ;go back to dos
MAIN            ENDP
CDSEG           ENDS
                END MAIN
```

11. The program follows:

```
TITLE           PROB11
PAGE            60,132
STSEG           SEGMENT
                DB 64 DUP (?)
STSEG           ENDS
;-----------------------------
DTSEG           SEGMENT
SS_AREA         LABEL   BYTE
SS_SIZE         DB   12
SS_ACTUAL       DB   ?
SS_DASHED       DB   12 DUP (?)
SS_NUM          DB   9 DUP (?)
DTSEG           ENDS
DASH            EQU 2DH
;-----------------------------
CDSEG           SEGMENT
MAIN            PROC FAR
                ASSUME CS:CDSEG,DS:DTSEG,SS:STSEG
                MOV AX,DTSEG
                MOV DS,AX
                CALL CLEAR              ;clear the screen
                CALL GETDATA            ;input a string into buffer
                CALL CONVERT            ;remove the dash
                MOV AH,4CH
                INT 21H                 ;go back to dos
MAIN            ENDP
;-----------------------------
;THIS SUBROUTINE CLEARS THE SCREEN
CLEAR           PROC
                MOV AX,0600H            ;scroll screen function
                MOV BH,07               ;normal attribute
                MOV CX,0000             ;scroll from row=00,col=00
                MOV DX,184FH            ;to row=18H,4FH
                INT 10H                 ;invoke interrupt to clear screen
                RET
CLEAR           ENDP
;-----------------------------
;THIS SUBROUTINE PUTS DATA FROM THE KEYBOARD INTO A BUFFER
GETDATA         PROC
                MOV AH,0AH              ;input string function
                MOV DX,OFFSET SS_AREA   ;dx points to buffer
                INT 21H                 ;invoke interrupt to input string
                RET
GETDATA         ENDP
;-----------------------------
;THIS SUBROUTINE REMOVES THE '-' FROM SS NUMBERS
```

```
CONVERT     PROC
            MOV BX,OFFSET SS_DASHED
            MOV SI,OFFSET SS_NUM
            MOV CX,11
LOOP1:      MOV AL,[BX]
            CMP AL,DASH
            JE INCR
            MOV [SI],AL
            INC SI
INCR:       INC BX
            LOOP LOOP1
            RET
CONVERT     ENDP
CDSEG       ENDS
            END MAIN
```

12. The program follows:

```
TITLE       PROB12
PAGE        60,132
            .MODEL SMALL
            .STACK 64H
            .DATA
NUM1        DB   8,?,8 DUP (?)
NUM2        DB   8,?,8 DUP (?)
PROMPT1     DB   CR,LF,'Enter the first number','$'
PROMPT2     DB   CR,LF,'Enter the second number','$'
PROMPT3     DB   CR,LF,'The total sum is '
SUM         DB    7 DUP (?),'$'
CR          EQU          0DH
LF          EQU          0AH
            .CODE
MAIN:       MOV AX,@DATA
            MOV DS,AX
            CALL CLEAR                  ;clear screen
            MOV AH,09
            MOV DX,OFFSET PROMPT1
            INT 21H                     ;display first prompt
            MOV AH,0AH
            MOV DX,OFFSET NUM1
            INT 21H                     ;get first number
            MOV AH,09
            MOV DX,OFFSET PROMPT2
            INT 21H                     ;display second prompt
            MOV AH,0AH
            MOV DX,OFFSET NUM2
            INT 21H                     ;get second number
            MOV SI,OFFSET NUM1 + 8      ;point to LSD of number 1
            MOV DI,OFFSET NUM2 + 8      ;point to LSD of number 2
            MOV BX,OFFSET SUM + 6       ;point to LSD of sum
            MOV CX,7                    ;add 7 bytes
            CLC                         ;clear carry
ADD_LP:     MOV AL,[SI]                 ;get byte from number 1
            ADC AL,[DI]                 ;add byte from number 2
            PUSHF                       ;save any carry
            AAA                         ;ASCII adjust
            OR AL,30H                   ;make it ASCII
            POPF                        ;restore flags
            MOV [BX],AL                 ;store sum (in BCD)
            DEC SI                      ;decrement pointers
            DEC DI                      ;to point to next byte
            DEC BX
            LOOP ADD_LP                 ;loop through 7 bytes
            MOV AH,09
```

```
                MOV DX,OFFSET PROMPT3
                INT 21H                 ;display result
                MOV AH,4CH
                INT 21H                 ;go back to DOS
;-------------------------------------------------
CLEAR           PROC
                MOV AH,06               ;clear screen function
                MOV AL,00               ;page 0
                MOV BH,07               ;normal attribute
                MOV CX,0                ;entire screen
                MOV DX,184FH
                INT 10H
                RET
CLEAR           ENDP
                END MAIN
```

SECTION 4.3: WHAT IS A MACRO AND HOW IS IT USED?

13. MACRO, ENDM

14. name: WORK_HOUR

    body:
    ```
    MOV  AL,40              ;WEEKLY HRS
    ADD  AL,OVRTME_HR       ;TOTAL HRS WORKED
    ```

    dummy argument: OVRTME_HR

15. .SALL suppresses listing of the macro body and comments in the list file

    .LALL lists the macro body and comments beginning with a single ';' comments beginning with double ';;' will list as a semicolon on a line by itself

    .XALL lists only the macro body with no comments; of course, comments appearing on the same line as an instruction will appear

16.
```
TITLE       PROB4
PAGE        60,132
;-----------------------------
WAGES       MACRO   SALARY,OVERTIME,BONUSES
;FINDING THE TOTAL WAGES
;;ADDS SALARY + OVERTIME + BONUSES
                SUB AX,AX               ;CLEAR
                MOV DX,AX               ;AX AND DX
                ADD AX,SALARY
                ADD AX,OVERTIME
                ADC DX,0                ;TAKE CARE OF CARRY
                ADD AX,BONUSES
                ADC DX,0
                ENDM
;-----------------------------
STSEG       SEGMENT
                DB 64 DUP (?)
STSEG       ENDS
;-----------------------------
DTSEG       SEGMENT
DTSEG       ENDS
;-----------------------------
CDSEG       SEGMENT
```

```
MAIN    PROC FAR
        ASSUME CS:CDSEG,DS:DTSEG,SS:STSEG
        MOV AX,DTSEG
        MOV DS,AX
        WAGES   60000,25000,3000
        MOV AH,4CH
        INT 21H                         ;GO BACK TO DOS
MAIN    ENDP
CDSEG   ENDS
        END MAIN
```

The trace shows the following:

AX = 57C0
DX = 0001

17. The order of using the operands does not matter:

```
ADD AX,OVERTIME
ADD AX,SALARY
```

18. The comment beginning with a single colon listed:

;FINDING THE TOTAL WAGES

The comment beginning with a double semicolon listed only as ';'. For example,

;;ADDS SALARY + OVERTIME + BONUSES

will be listed as

;

19. The list file follows:

```
Microsoft (R) Macro Assembler Version 5.10              9/4/92   09:44:2
PROB6                                   Page   1-1

                        TITLE   PROB6
                        PAGE    60,132
                        ;------------------------------
                        WAGES   MACRO   SALARY,OVERTIME,BONUSES
                        ;FINDING THE TOTAL WAGES
                                ;;ADDS SALARY + OVERTIME + BONUSES
                                SUB AX,AX                       ;CLEAR
                                MOV DX,AX                       ;AX AND DX
                                ADD AX,SALARY
                                ADD AX,OVERTIME
                                ADC DX,0                        ;TAKE CARE OF CARRY
                                ADD AX,BONUSES
                                ADC DX,0
                                ENDM
                        ;------------------------------
0000                            STSEG  SEGMENT
0000  0040[                     DB 64 DUP (?)
??
102B F255                   ]

0040                            STSEG  ENDS
                        ;------------------------------
0000                            DTSEG  SEGMENT
0000                            DTSEG  ENDS
                        ;------------------------------
0000  CDSEG                     SEGMENT
0000                            MAIN    PROC FAR
                                ASSUME CS:CDSEG,DS:DTSEG,SS:STSEG
0000  B8 ---- R                         MOV AX,DTSEG
0003  8E D8                            MOV DS,AX
0005  BB EA60                          MOV BX,60000
0008  B9 61A8                          MOV CX,25000
000B  BE 0BB8                          MOV SI,3000
                                WAGES BX,CX,SI
000E  2B C0                     1       SUB AX,AX               ;CLEAR
0010  8B D0                     1       MOV DX,AX               ;AX AND DX
0012  03 C3                     1       ADD AX,BX
0014  03 C1                     1       ADD AX,CX
0016  83 D2 00                  1       ADC DX,0                ;TAKE CARE OF CARRY
0019  03 C6                     1       ADD AX,SI
001B  83 D2 00                  1       ADC DX,0
001E  B4 4C                             MOV AH,4CH
0020  CD 21                             INT 21H                 ;GO BACK TO DOS
                                MAIN    ENDP
0022                            CDSEG  ENDS
```

```
END MAIN
```

20. The blanks are filled in as follows:

```
SUMMING     MACRO    COUNT,VALUES
            LOCAL AGAIN
            ;;THIS MACRO ADDS AN ARRAY OF BYTE SIZE ELEMENTS.
            ;;AX WILL HOLD THE TOTAL SUM
            MOV CX,COUNT                ;SIZE OF ARRAY
            MOV SI,OFFSET VALUES        ;LOAD OFFSET ADDRESS OF ARRAY
            SUB AX,AX                   ;CLEAR AX
AGAIN:      ADD AL,[SI]
            ADC AH,0                    ;ADD BYTES AND TAKES CARE OF CARRIES
            INC SI                      ;POINT TO NEXT BYTE
            LOOP AGAIN                  ;CONTINUE UNTIL FINISHED
            ENDM
```

21. The result in AX for SUMMING is as follows:

1st invocation: AX = 1A2H SUMMING 5,DATA1
2nd invocation: AX = 299H SUMMING 8,DATA2
3rd invocation: AX = 3DEH SUMMING 10,DATA3

22.

```
.LALL
SUMMING  5,DATA1
MOV SUM1,AX
```

The above lines of code list all lines of the macro. The comments beginning with ';;' are not listed but are replaced by ';' on a blank line.

```
.XALL
SUMMING  8,DATA2
MOV SUM2,AX
```

The above lines of code cause the comments to be suppressed in the listing, only code generating statements are listed.

```
.SALL
SUMMING  10,DATA3
MOV SUM3,AX
```

The above lines of code cause no lines to be listed in the list file.

23. The macro is changed as follows:

```
SUMMING     MACRO    COUNT,VALUES, SUM
            LOCAL AGAIN
;;THIS MACRO ADDS AN ARRAY OF BYTE SIZE ELEMENTS.
;;AX WILL HOLD THE TOTAL SUM
            MOV CX,COUNT                ;SIZE OF ARRAY
            MOV SI,OFFSET VALUES        ;LOAD OFFSET ADDRESS OF ARRAY
            SUB AX,AX                   ;CLEAR AX
AGAIN:      ADD AL,[SI]
            ADC AH,0                    ;ADD BYTES AND TAKES CARE OF CARRIES
            INC SI                      ;POINT TO NEXT BYTE
            LOOP AGAIN                  ;CONTINUE UNTIL FINISHED
            MOV SUM,AX
            ENDM
```

The macro invocations are changed as follows:

```
.LALL
SUMMING  5,DATA1,SUM1
.XALL
SUMMING  8,DATA2,SUM2
.SALL
 SUMMING  10,DATA3,SUM3
```

24. The macro is changed as follows:

```
MULTIPLY        MACRO VALUE1, VALUE2, RESULT
                LOCAL BACK
;THIS MACRO COMPUTES RESULT = VALUE1 X VALUE2
;;BY REPEATED ADDITION
;;VALUE1 AND VALUE2 ARE WORD OPERANDS; RESULT IS  DWORD
                MOV BX,VALUE1                   ;BX=MULTIPLIER
                MOV CX,VALUE2                   ;CX=MULTIPLICAND
                SUB AX,AX                       ;CLEAR AX
                MOV DX,AX                       ;CLEAR DX
BACK:           ADD AX,BX                       ;ADD BX TO AX
                ADC DX,00                       ;ADD CARRIES IF THERE IS ONE
                LOOP BACK                       ;CONTINUE UNTIL CX=0
                MOV  SI,OFFSET RESULT
                MOV [SI],AX                     ;SAVE THE LOW WORD
                MOV [SI]+2,DX                   ;SAVE THE HIGH WORD
                ENDM
```

The changes in the data segment are as follows:

```
RESULT1    DD                                   ?
RESULT2    DD                                   ?
RESULT3    DD                                   ?
```

CHAPTER 5: KEYBOARD AND MOUSE PROGRAMMING
SECTION 5.1: INT 16H KEYBOARD PROGRAMMING

1. INT 16H function AH = 01
2. After return from INT 16H function AH = 01, if ZF = 1 there is no key press; if ZF = 0 then a key has been pressed. If ZF = 0, then we use INT 16H function AH = 0 to get the ASCII character for the pressed key.

SECTION 5.2: MOUSE PROGRAMMING WITH INT 33H

3.

```
TITLE   PROBLEM 13:
;this program checks the presence of mouse and displays the
;number of buttons it supports
PAGE    60,132
DISPLAY         MACRO           STRING
        MOV     AH,09H
        MOV     DX,OFFSET STRING ;LOAD STRING ADDRESS
        INT     21H
        ENDM

.MODEL SMALL
.STACK

.DATA
MESSAGE_1       DB 'THERE IS A MOUSE INSTALLED IN THIS PC ','$'
MESSAGE_2       DB 'AND IT HAS '
BUTTON          DB ?, ' BUTTONS $'
MESSAGE_3       DB 'THIS PC HAS NO MOUSE','$'
.CODE
MAIN    PROC
        MOV     AX,@DATA
        MOV     DS,AX
        MOV     AX,0600H            ;clear screen
        MOV     BH,07
        MOV     CX,0
        MOV     DX,184FH
        INT     10H
        MOV     AX,0                ;initialize mouse
        INT     33H
        CMP     AX,0                ;see if there is mouse
        JZ      OVER                ;no mouse. get out
        DISPLAY MESSAGE_1           ;it got mouse, BX= mouse info
        OR      BL,30H              ;make it ASCII
        MOV     BUTTON,BL           ;save the button number
        DISPLAY MESSAGE_2           ;show mouse buttons
        JMP     EXIT                ;exit to DOS
OVER:   DISPLAY MESSAGE_3 ;
EXIT:   MOV     AH,4CH              ;go back to DOS
        INT     21H
        MAIN    ENDP
END     MAIN
```

CHAPTER 6: SIGNED NUMBERS, STRINGS, AND TABLES

SECTION 6.1: SIGNED NUMBER ARITHMETIC OPERATIONS

1. (a) 1110 1001 (b) 0000 1100 (c) 1101 1000 (d) 0110 1111
   (e) 1000 0000 (f) 0111 1111 (g) 0000 0001 0110 1101
   (h) 1000 0000 0000 0001

2. (a) OF = 0, result = 03 (b) OF = 1, result = 06
   (c) OF = 0, result = 47H (d) OF = 0, result = 00
   (e) OF = 0, result = 00

3.
```
MOV AL,-122             ;AL = 86
CBW         ;AX = FF86
MOV AX,-999H            ;AX =  F667
CWD         ;DX = FFFF
MOV AL,17H              ;AL = 17
CBW         ;AX = 0017
MOV AL,127              ;AL = 7F
CBW         ;AX = 007F
MOV AX,-129             ;AX = FF7F
CWD         ;DX = FFFF
```

4. The changes are highlighted below:
   (1) data name LOWEST was changed to HIGHEST
   (2) after the CMP, the conditional jump was changed to JGE

```
TITLE       PROB4   ;FIND THE HIGHEST TEMPERATURE
PAGE        60,132
;-----------------------------
STSEG       SEGMENT
            DB 64 DUP (?)
STSEG       ENDS
;-----------------------------
DTSEG       SEGMENT
SIGN_DAT DB +13,-10,+19,+14,-18,-9,+12,-9,+16
            ORG 0010H
HIGHEST     DB ?
DTSEG       ENDS
;-----------------------------
CDSEG       SEGMENT
MAIN        PROC FAR
            ASSUME CS:CDSEG,DS:DTSEG,SS:STSEG
            MOV AX,DTSEG
            MOV DS,AX
            MOV CX,8                ;load counter (number items - 1)
            MOV SI,OFFSET SIGN_DAT  ;set up pointer
            MOV AL,[SI]             ;AL holds highest value found so far
BACK:       INC SI                  ;increment pointer
CMP AL,[SI]                         ;compare next byte to highest
            JGE SEARCH              ;IF AL is higher, continue search
            MOV AL,[SI]             ;otherwise save new highest
SEARCH:     LOOP BACK               ;loop if not finished
            MOV HIGHEST,AL          ;save highest temperature
            MOV AH,4CH
            INT 21H                 ;go back to DOS
```

```
MAIN        ENDP
CDSEG       ENDS
            END MAIN
```

SECTION 6.2: STRING AND TABLE OPERATIONS

5. STD is used to set the direction flag to one,
CLD is used to reset the direction flag to zero
the direction flag determines the direction of repeated string instructions, if DF = 0, the pointers (DI and SI) will be automatically incremented; if DF = 1, the pointers will be automatically decremented

6. (a) (b) (e)

7. (a) destination = DI, source = SI
(b) destination = DI, source = SI
(c) source = SI, destination = DI
(d) operand = SI, destination = AL
(e) operand = DI, source = AX
(f) operand = AX, destination = DI

8.

```
TITLE       PROB4   ;FIND THE HIGHEST TEMPERATURE
PAGE        60,132
;-----------------------------
STSEG       SEGMENT
            DB 64 DUP (?)
STSEG       ENDS
;-----------------------------
DTSEG       SEGMENT
DATA1       DB     'I pledge allegiance to the flag '
            DB     'of the United States of America,'
            DB     'and to the Republic for which it'
            DB     ' stands, one nation under God, indivisible,
            DB     'with liberty and justice for all.  '
            DB     '************************!'
            ORG 300H
DATA2       DW      200 DUP (?)
DTSEG       ENDS
;-----------------------------
CDSEG       SEGMENT
MAIN        PROC FAR
            ASSUME CS:CDSEG,DS:DTSEG,SS:STSEG,ES:DTSEG
            MOV  AX,DTSEG
            MOV  DS,AX                  ;INITIALIZE THE DATA SEGMENT
            MOV  ES,AX                  ;INITIALIZE THE EXTRA SEGMENT
            CLD                         ;CLEAR DIRECTION FLAG FOR AUTOINCREMENT
            MOV  SI,OFFSET DATA1        ;LOAD THE SOURCE POINTER
            MOV  DI,OFFSET DATA2        ;LOAD THE DESTINATION POINTER
            MOV  CX,200                 ;LOAD THE COUNTER
            REP MOVSW                   ;REPEAT UNTIL CX BECOMES ZERO
            MOV AH,4CH
            INT 21H                     ;GO BACK TO DOS
MAIN        ENDP
CDSEG       ENDS
```

END MAIN

9.

```
TITLE       PROB9
PAGE        60,132
;-----------------------------
STSEG       SEGMENT
            DB 64 DUP (?)
STSEG       ENDS
;-----------------------------
DTSEG       SEGMENT
ASC_DATA    DB   '0123456789'
            DB   '0123456789'
            DB   '0123456789'
            DB   '0123456789'
            DB   '0123456789'
            ORG 100H
COPY_DATA   DW    50 DUP (?)
DTSEG       ENDS
;-----------------------------
CDSEG       SEGMENT
MAIN        PROC FAR
            ASSUME CS:CDSEG,DS:DTSEG,ES:DTSEG,SS:STSEG
            MOV AX,DTSEG
            MOV DS,AX
            MOV ES,AX
            CLD                         ;set up DF autoincrement
            MOV SI,OFFSET ASC_DATA      ;SI points to ASCII data
            MOV DI,OFFSET COPY_DATA     ;DI points to BCD data
            MOV CX,50                   ;50 bytes will be processed
LD_LOOP:    LODSB                       ;load an ASCII byte
            AND AL,0FH                  ;convert to unpacked BCD
            STOSB                       ;store unpacked BCD
            LOOP LD_LOOP                ;continue looping
            MOV AH,4CH
            INT 21H                     ;go back to dos
MAIN        ENDP
CDSEG       ENDS
            END MAIN
```

10. REPNE

11.

```
TITLE       PROB11
PAGE        60,132
;-----------------------------
STSEG       SEGMENT
            DB 64 DUP (?)
STSEG       ENDS
;-----------------------------
DTSEG       SEGMENT
DATA_1      DB   'IbM'
DTSEG       ENDS
;-----------------------------
```

```
CDSEG       SEGMENT
MAIN        PROC FAR
            ASSUME CS:CDSEG,DS:DTSEG,ES:DTSEG,SS:STSEG
            MOV AX,DTSEG
            MOV DS,AX
            MOV ES,AX                   ;set up extra segment
            CLD                         ;set DF = 0 to autoincrement
            MOV DI,OFFSET DATA_1        ;DI points to data to be scanned
            MOV CX,03                   ;string length = 3
            MOV AL,'b'                  ;search for 'b'
            REPNE  SCASB                ;scan until 'b' is found
            JNE EXIT                    ;exit if not found
            DEC DI                      ;if found, point to it
            MOV BYTE PTR [DI],'B'       ;replace it with 'B'
EXIT:       MOV AH,4CH
            INT 21H                     ;go back to dos
MAIN        ENDP
CDSEG       ENDS
            END MAIN
```

12. For the 8086, the total clock count for XLAT is 11, and for the instructions equivalent to XLAT, the total clock count is 15.

```
XLAT                    ;clock count =  11
SUB  AH,AH              ;clock count =  3
MOV SI,AX               ;clock count =  2
MOV AL,[BX+SI]          ;clock count = 10
```

13.

```
TITLE       PROB13
PAGE        60,132
;-----------------------------
STSEG       SEGMENT
            DB 64 DUP (?)
STSEG       ENDS
;-----------------------------
DTSEG       SEGMENT
Y_TABLE  DB   5,8,13,20,29,40,53,68,85,104
X_VAL       DB          3
Y_VAL       DB          ?
DTSEG       ENDS
;-----------------------------
CDSEG       SEGMENT
MAIN        PROC FAR
            ASSUME CS:CDSEG,DS:DTSEG,ES:DTSEG,SS:STSEG
            MOV AX,DTSEG
            MOV DS,AX
            MOV BX,OFFSET Y_TABLE   ;point to table
            MOV AL,X_VAL            ;retrieve item
            XLAT                    ;place item in AL
            MOV Y_VAL,AL            ;get Y value
            MOV AH,4CH
            INT 21H                 ;go back to dos
MAIN        ENDP
CDSEG       ENDS
            END MAIN
```

CHAPTER 7: MODULES AND MODULAR PROGRAMMING

SECTION 7.1: WRITING AND LINKING MODULES

1. The underlined portions below show how the blanks were filled.

```
                .MODEL SMALL
                .STACK 100H
                .DATA
                PUBLIC      DATA1, RESULT
DATA1           DB          25, 12,34,56,98
RESULT          DW          ?
                .CODE
                EXTRN   SUM:FAR
HERE:           MOV   AX,@DATA
                MOV   DS,AX
                CALL  SUM
                MOV   AH,4CH
                INT   21H
                END   HERE
```

```
                .MODEL  SMALL
                EXTRN DATA1:BYTE
                EXTRN RESULT:WORD
COUNT           EQU     5
                .CODE
                PUBLIC SUM
SUM             PROC  FAR
                MOV   BX,OFFSET DATA1
                SUB    AX,AX
                MOV   CX,COUNT
AGAIN:          ADD    AL, BYTE PTR [BX]
                ADC    AH,0
                INC    BX
                LOOP  AGAIN
                MOV   RESULT,AX
                RET
SUM             ENDP
                END  SUM
```

2. EXTRN

3. PUBLIC

4. NEAR, FAR, PROC

5. BYTE, WORD, DWORD, FWORD, QWORD, TBYTE

6. no options are needed for the PUBLIC directive

7. no options are needed for the PUBLIC directive

8. The changes are underlined below. All procedures are declared EXTRN with type NEAR. MESSAGE is declared PUBLIC because module 3 refers to it. The procedures are moved to separate files.

```
TITLE     PROB8 SIMPLE DISPLAY PROGRAM
PAGE      60,132
          EXTRN CLEAR:NEAR
          EXTRN CURSOR:NEAR
          EXTRN DISPLAY:NEAR
          PUBLIC MESSAGE
STSEG     SEGMENT
          DB 64 DUP (?)
STSEG     ENDS
;-----------------------------
DTSEG     SEGMENT
MESSAGE   DB 'This is a test of the display routine','$'
DTSEG     ENDS
;-----------------------------
CDSEG     SEGMENT
MAIN      PROC FAR
          ASSUME CS:CDSEG,DS:DTSEG,SS:STSEG
          MOV AX,DTSEG
          MOV DS,AX
          CALL CLEAR              ;CLEAR THE SCREEN
          CALL CURSOR             ;SET CURSOR POSITION
          CALL DISPLAY            ;DISPLAY MESSAGE
          MOV AH,4CH
          INT 21H                 ;GO BACK TO DOS
MAIN      ENDP
CDSEG     ENDS
          END MAIN
```

Each subroutine is placed in a separate file. The procedure name is made PUBLIC and is placed inside a code segment. The PROC is given type FAR and the ASSUME statement refers CS to the code segment. The END directive marks the end of the module.

```
;THIS SUBROUTINE CLEARS THE SCREEN
          PUBLIC CLEAR
CDSEG     SEGMENT
CLEAR     PROC
          ASSUME CS:CDSEG
          MOV AX,0600H            ;SCROLL SCREEN FUNCTION
          MOV BH,07               ;NORMAL ATTRIBUTE
          MOV CX,0000             ;SCROLL FROM ROW=00,COL=00
          MOV DX,184FH            ;TO ROW=18H,COL=4FH
          INT 10H                 ;INVOKE INTERRUPT TO CLEAR SCREEN
          RET
CLEAR     ENDP
CDSEG     ENDS
          END
```

The changes for modules 2 and 3 are similar to those for module 1. Modules 2 and 3 are shown next

```
;THIS SUBROUTINE SETS THE CURSOR AT THE CENTER OF THE SCREEN
                PUBLIC CURSOR
CDSEG           SEGMENT
CURSOR          PROC
                ASSUME CS:CDSEG
                MOV AH,02                   ;SET CURSOR FUNCTION
                MOV BH,00                   ;PAGE 00
                MOV DH,12                   ;CENTER ROW
                MOV DL,39                   ;CENTER COLUMN
                INT 10H                     ;INVOKE INTERRUPT TO SET CURSOR POSITION
                RET
CURSOR          ENDP
CDSEG           ENDS
                END


;THIS SUBROUTINE DISPLAYS A STRING ON THE SCREEN
                EXTRN MESSAGE:BYTE
                PUBLIC DISPLAY
CDSEG           SEGMENT
DISPLAY         PROC
                ASSUME CS:CDSEG
                MOV AH,09                   ;DISPLAY FUNCTION
                MOV DX,OFFSET MESSAGE       ;DX POINTS TO OUTPUT BUFFER
                INT 21H                     ;INVOKE INTERRUPT TO DISPLAY STRING
                RET
DISPLAY         ENDP
CDSEG           ENDS
                END
```

SECTION 7.2: SOME VERY USEFUL MODULES

9.

```
TITLE           PROB9
PAGE            60,132
                EXTRN ASC2B_CON:FAR
                EXTRN B2ASC_CON:FAR
                PUBLIC TEN
STSEG           SEGMENT
                DB 64 DUP (?)
STSEG           ENDS
;-----------------------------
DTSEG           SEGMENT
TEN             DW   10
MESSAGE1        DB CR,LF,'Enter a three digit number','$'
                ORG 20H
NUMBER1         DB 4,?,4 DUP (0)
NUMBER2         DB 4,?,4 DUP (0)
                ORG 30H
MESSAGE2        DB CR,LF,'The average is '
ASC_AVG         DB 5 DUP (20H),'$'
DTSEG ENDS
CR  EQU 0DH      ;EQUATE CR WITH ASCII CODE FOR CARRIAGE RETURN
LF  EQU 0AH      ;EQUATE LF WITH ASCII CODE FOR LINE FEED
```

```
;-----------------------------

CDSEG           SEGMENT
MAIN            PROC    FAR
                ASSUME CS:CDSEG,DS:DTSEG,SS:STSEG
                MOV  AX,DTSEG
                MOV  DS,AX
                MOV AX,0600H                ;clear the screen
                MOV BH,07
                MOV CX,0000
                MOV DX,184FH
                INT 10H
                MOV  AH,09                  ;prompt for the first number
                MOV  DX,OFFSET MESSAGE1
                INT  21H
                MOV  AH,0AH                 ;get the first number
                MOV  DX,OFFSET NUMBER1
                INT  21H
                MOV AH,09                   ;prompt for the second number
                MOV  DX,OFFSET MESSAGE1
                INT  21H
                MOV  AH,0AH                 ;get the second number
                MOV  DX,OFFSET NUMBER2
                INT  21H
                ;SI and BX must be set up for ASC2B_CON
                MOV SI,OFFSET NUMBER1 + 2  ;SI points to beginning of ASCII string
                SUB BH,BH                   ;BX = string length - 1
                MOV BL,BYTE PTR [SI-1]
                DEC  BX
                CALL ASC2B_CON              ;convert number 1 to hex
                MOV CX,AX                   ;save number1 in CX
                MOV SI,OFFSET NUMBER2 + 2  ;SI points to beginning of ASCII string
                SUB BH,BH                   ;BX = string length - 1
                MOV BL,BYTE PTR [SI-1]
                DEC BX
                CALL ASC2B_CON              ;convert number 2 to hex
                ADD AX,CX       ;add the numbers
                SHR AX,1        ;divide by 2
                MOV  SI,OFFSET ASC_AVG      ;SI points to
                CALL B2ASC_CON              ;convert AX to ASCII
                MOV  AH,09                  ;display  average
                MOV  DX,OFFSET MESSAGE2
                INT  21H
                MOV  AH,4CH
                INT  21H                    ;go back to DOS
MAIN            ENDP
CDSEG           ENDS
                END MAIN


TITLE           B2ASC BINARY TO DECIMAL CONVERSION MODULE
PAGE            60,132
;this module converts a binary (hex) number up to FFFFH to decimal
; then makes it displayable (ASCII)
;CALLING PROGRAM SETS
; AX = BINARY VALUE TO BE CONVERTED TO ASCII
; SI = OFFSET ADDRESS WHERE ASCII VALUE TO BE STORED
                PUBLIC B2ASC_CON
```

```
CDSEG       SEGMENT PARA PUBLIC 'CODE'
B2ASC_CON   PROC FAR
            ASSUME CS:CDSEG
            PUSHF                   ;STORE REGS CHANGED BY THIS MODULE
            PUSH BX
            PUSH DX
            MOV BX,10               ;BX=10 THE DIVISOR
            ADD SI,4                ;SI POINTS TO LAST ASCII DIGIT
B2A_LOOP:   SUB DX,DX               ;DX MUST BE 0 IN WORD DIVISION
            DIV BX                  ;DIVIDE HEX NUMBER BY 10 (BX=10)
            OR DL,30H               ;TAG '3' TO REMAINDER TO MAKE IT ASCII
            MOV [SI],DL             ;MOVE THE ASCII DIGIT
            DEC SI                  ;DECREMENT POINTER
            CMP AX,0                ;CONTINUE LOOPING WHILE AX 0
            JA B2A_LOOP
            POP DX                  ;RESTORE REGISTERS
            POP BX
            POPF
            RET
B2ASC_CON   ENDP
CDSEG       ENDS
            END

TITLE       ASC2B ASCII TO BINARY CONVERSION MODULE
PAGE        60,132
;this module converts any ASCII number between 0 to 65535 to binary
;CALLING PROGRAM SETS SI = OFFSET OF ASCII STRING
; BX = STRING LENGTH - 1 (USED AS INDEX INTO ASCII NUMBER)
;THIS MODULE SETS AX = BINARY NUMBER
;-------------------------
            EXTRN TEN:WORD
            PUBLIC ASC2B_CON
CDSEG       SEGMENT PARA PUBLIC 'CODE'
ASC2B_CON   PROC FAR
            ASSUME CS:CDSEG
            PUSHF                   ;STORE REGS CHANGED IN THIS MODULE
            PUSH DI
            PUSH CX
            SUB DI,DI               ;CLEAR DI FOR THE BINARY(HEX) RESULT
            MOV CX,1                ;CX = WEIGHT FACTOR
A2B_LOOP:   MOV AL,[SI+BX]          ;GET THE ASCII DIGIT
            AND AL,0FH              ;STRIP OFF '3'
            SUB AH,AH               ;CLEAR AH FOR WORD MULTIPLICATION
            MUL CX                  ;MULTIPLY BY THE WEIGHT
            ADD DI,AX               ;ADD IT TO BINARY (HEX) RESULT
            MOV AX,CX               ;MULTIPLY THE WEIGHT FACTOR
            MUL TEN                 ; BY TEN
            MOV CX,AX               ; FOR NEXT ITERATION
            DEC BX                  ;DECREMENT DIGIT POINTER
            JNS A2B_LOOP            ;JUMP IF OFFSET 0
            MOV AX,DI               ;STORE BINARY NUMBER IN AX
            POP CX                  ;RESTORE FLAGS
            POP DI
            POPF
            RET
ASC2B_CON   ENDP
CDSEG       ENDS
            END
```

10. The program follows:

```
TITLE           PROB10
PAGE            60,132
                EXTRN SUBPROG1:FAR
                EXTRN SUBPROG2:FAR
                PUBLIC PRODUCT
                PUBLIC QUOTIENT
;----------------------------
STSEG           SEGMENT PARA STACK 'STACK'
                DB 100 DUP(?)
STSEG           ENDS
DTSEG           SEGMENT PARA 'DATA'
VALUE1          DW 1228
VALUE2          DW 52400
PRODUCT         DW 2 DUP (?)
QUOTIENT        DW  (?)
REMAINDER       DW  (?)
DTSEG           ENDS
CODSG_A         SEGMENT PARA 'CODE'
MAIN            PROC FAR
                ASSUME CS:CODSG_A,DS:DTSEG,SS:STSEG
                MOV AX,DTSEG
                MOV DS,AX
                PUSH VALUE1
                PUSH VALUE2
                CALL SUBPROG1           ;CALL SUBPROG TO MUL VALUE1 * VALUE2
                PUSH VALUE1
                PUSH VALUE2
                CALL SUBPROG2           ;CALL SUBPROG TO DIV VALUE2 / VALUE1
                MOV AH,4CH
                INT 21H                 ;GO BACK TO DOS
MAIN            ENDP
CODSG_A         ENDS
                END MAIN

;THIS PROGRAM MULTIPLIES TWO EXTERNALLY DEFINED WORDS
;AND STORES THE PRODUCT IN AN EXTERNALLY DEFINED DWORD

TITLE           SUBPROG1 PROGRAM TO MULTIPLY TWO WORDS
PAGE            60,132
                EXTRN PRODUCT:WORD
                PUBLIC SUBPROG1
CODSG_C         SEGMENT PARA 'CODE'
SUBPROG1 PROC FAR
                ASSUME CS:CODSG_C
                PUSH BP                 ;save SP
                MOV BP,SP               ;set up BP to access stack
                MOV AX,[BP]+8           ;get VALUE1
                MOV CX,[BP]+6           ;get VALUE2
                SUB BX,BX               ;initialize carry count
                MUL CX                  ;mul value1 * value2
                MOV PRODUCT,AX          ;store product
                MOV PRODUCT+2,DX        ;store upper word or product
                POP BP                  ;restore BP
                RET 4
SUBPROG1        ENDP
```

```
CODSG_C     ENDS
            END

;THIS PROGRAM DIVIDES TWO EXTERNALLY DEFINED WORDS
;AND STORES THE QUOTIENT IN AN EXTERNALLY DEFINED WORD
TITLE       SUBPROG2 PROGRAM TO ADD TWO WORDS
PAGE        60,132
            EXTRN QUOTIENT:WORD
            EXTRN REMAINDER:WORD
            PUBLIC SUBPROG2
CODSG_B     SEGMENT PARA 'CODE'
SUBPROG2    PROC FAR
            ASSUME CS:CODSG_B
            PUSH BP                 ;save BP
            MOV  BP,SP              ;set up BP to access stack
            MOV AX,[BP]+6           ;move VALUE2 to AX
            MOV BX,[BP]+8           ;move VALUE1 to BX
            SUB DX,DX               ;INITIALIZE REM
            DIV BX                  ;divide VALUE2/VALUE1
            MOV QUOTIENT,AX         ;store quotient
            MOV REMAINDER,DX        ;store remainder
            POP BP                  ;restore BP
            RET 4
SUBPROG2    ENDP
CODSG_B     ENDS
            END
```

CHAPTER 8: 32-BIT PROGRAMMING FOR x86

SECTION 8.1: 32-BIT PROGRAMMING IN x86

1. (a) AL = B6 AH = F4 AX = F4B6 EAX = 9823F4B6
   (b) BL = C2 BH = 85 BX = 85C2 EBX =000985C2
   (c) DL = 80 DH = 84 DX = 8480 EDX =001E8480
   (d) SI = 0000 ESI = 00120000

2. (a) EAX = 02CEFF93
   (b) EBX = 00124F80
   (c) EDX = 024B76A0
   (d) EAX = 09090804
   (e) EBX = 0F90EC52

3. (a)
   DS:2000 = 56
   DS:2001 = F4
   DS:2002 = 23
   DS:2003 = 98

   (b)
   DS:348C = 91
   DS:348D = 34
   DS:348E = F2
   DS:348F = 01

   (c) register EBX = 4CA26D92H

CHAPTER 9: 8088, 80286 MICROPROCESSORS AND ISA BUS

SECTION 9.1: 8088 MICROPROCESSOR

1. The 8088 has only pins AD0 - AD7, whereas the 8086 has AD0 - AD15. Furthermore, only the 8086 has the BHE pin. They are not interchangeable.
2. output
3. 2^{20} = 1,048,576 bytes, since the 8088 has only 20 bits for address pins
4. input
5. output for address, and both input and output for data
6 maximum
7. true
8. all of them
9. It uses less pins and consequently has a smaller package.
10. one extra clock for every data byte access
11. in the first T state
12. Due to the fact that it has a 16-bit external data bus, it requires more wire strips on the printed circuit board.
13. to low (ground)
14. IP=0000 and CS=FFFFH
15. b
16. c
17. a

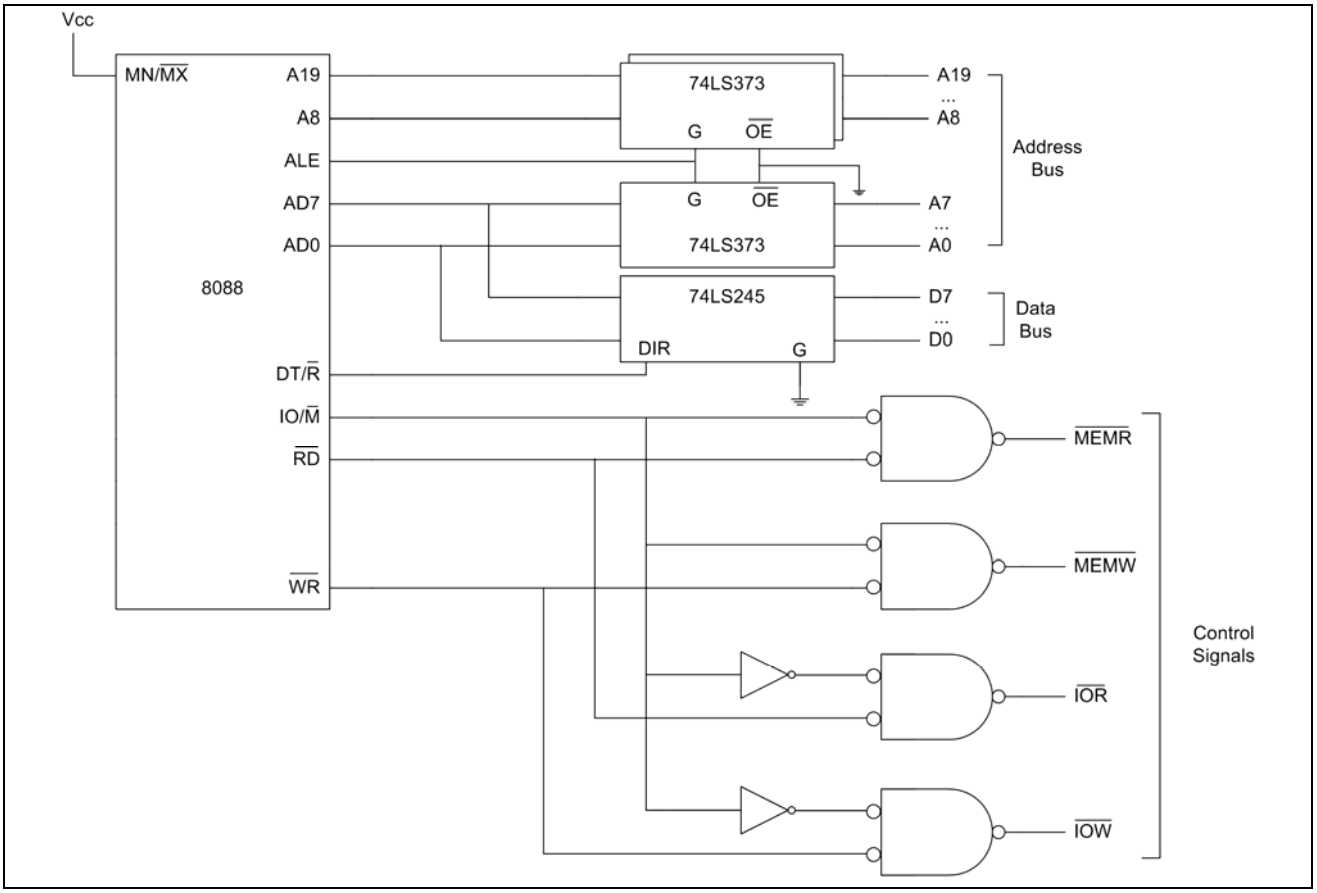
SECTION 9.2: 8284 AND 8288 SUPPORTING CHIPS

18. 8288
19. all except RESET and NMI
20. b
21. c
22. a

SECTION 9.3: 8-BIT SECTION OF ISA BUS

23. 8088
24. 3
25. the 8088; When AEN=1, DMA is in control of all the buses on the system board.
26. (a) 74LS245 (b) 74LS244
27. d

28. The following is the diagram.



29. low
30. DIR = 1 and G = 0
31. DIR = 0 and G = 0
32. the Hi-Z (impedance) bus state, since the 74LS245 is not activated and therefore no transfer of data happens for either direction
33. data
34. G = 1 and OE = 0; The data is latched when G goes from high to low.

SECTION 9.4: 80286 MICROPROCESSOR

35. true
36. No, this person is wrong. The 80286 is a byte-addressable processor, meaning that each address location holds one byte of data and not two bytes. Therefore, we have $2^{24} \times 1 = 16{,}777{,}216 = 16$ megabytes.
37. both high and low bytes: D0 - D7 and D8 - D15
38. even
39. true
40. False, only the 1M.
41. real
42. CS = F000H, IP = FFF0H and all the rest are 0000.
43. FFFFF0H
44. Since CS = F000H and IP = FFF0, we have A19 - A0=FFFF0H; A23 -A20 = 1111. That makes the physical address FFFFF0H for the first memory location upon activation of the RESET pin in the 80286.

SECTION 9.5: 16-BIT ISA BUS 246

45. 62-pin and 36-pin parts. The A side and the B side each have 31 pins (for a total of 62 pins). The C and D side each have 18 pins (for a total of 36 pins). The A side of the 62-pin and C side of the 36-pin are the component side on the expansion card.
46. the 36-pin part
47. the 36-pin part
48. to make it PC/XT compatible
49. 36-pin since it is part of the 80286 CPU
50. The physical address is FC480H + 7652H = 103AD2H. In the case of the 8088 and 8086, the wrap-around makes A19 - A0 = 03AD2H, meaning that the 1 is dropped, but for the 286 and later processors A20 =1.
51. 80286
52. MEMW and MEMR on the 36-pin part
53. true
54. true

CHAPTER 10: MEMORY AND MEMORY INTERFACING

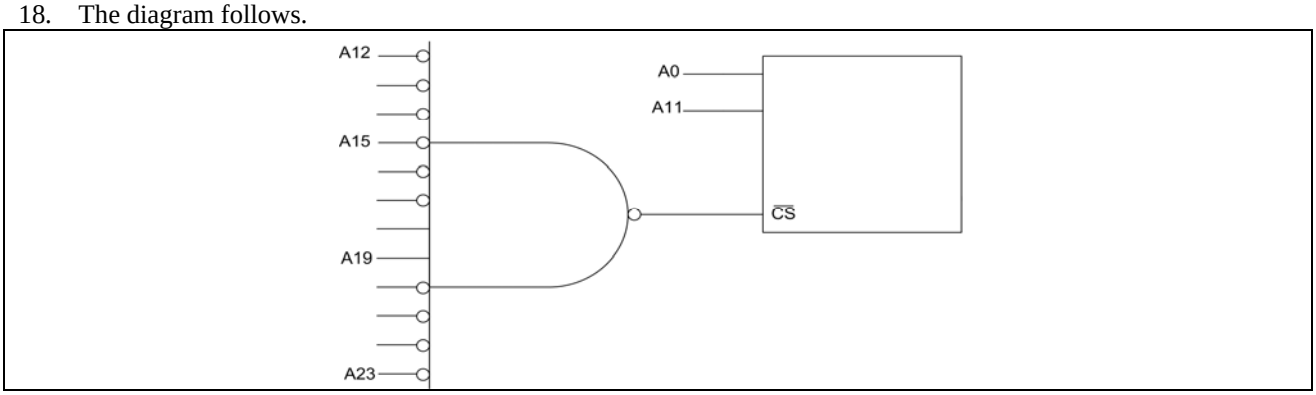
SECTION 10.1: SEMICONDUCTOR MEMORIES

1. For memory it is 4 megabits, and for the computer it is 4 megabytes.
2. true
3. true
4. false, not necessarily
5. true
6. access time
7. true
8. It need not be removed from the system board to be erased and reprogrammed.
9. true
10. DRAM
11. SRAM
12. all of them, except UV-EPROM and NV-RAM
13. c
14. c
15. (a) 32Kx8, 256K (b) 8Kx8, 64K
    (c) 4Kx8, 32K (d) 8Kx8, 64K
    (e) 4Mx1, 4M (f) 8Kx8, 64K
    (g) 4Kx8, 32K (h) 2Kx8, 16K
    (i) 256Kx4, 1M (j) 64Kx8, 512K

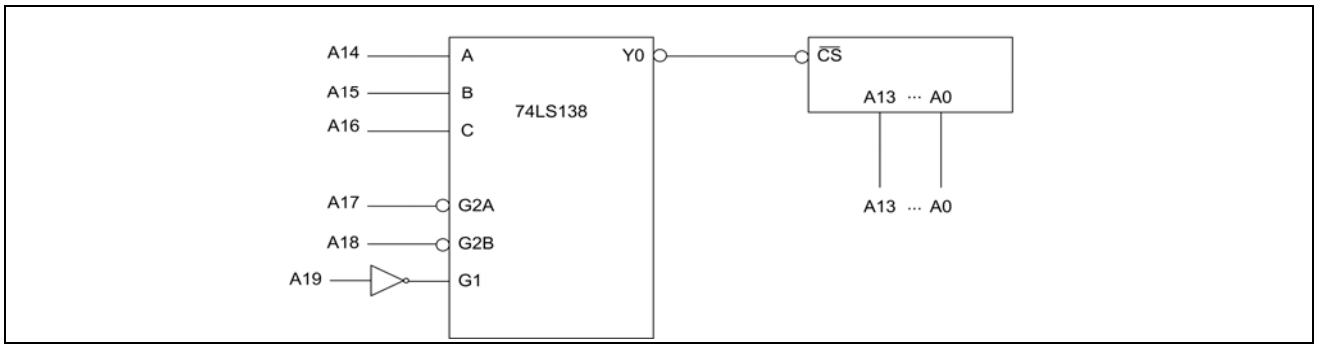
16. (a) 128K, A0 - A13, D0 - D7 (b) 256K, A0 - A14, D0 - D7
    (c) 512K, A0 - A15, D0- D7 (d) 1M, A0 - A8, D0 - D3 plus RAS and CAS
    (e) 512K, A0-A15, D0-D7 (f) 256K, A0 - A7, D0 - D3 plus RAS and CAS
    (g) 4M, A0-A9, D0 - D3 plus RAS and CAS
    (h) 16M, A0 - A10, D0 - D3 plus RAS and CAS
    (i) 512K, A0 - A7, D0 - D7 plus RAS and CAS

SECTION 10.2: MEMORY ADDRESS DECODING

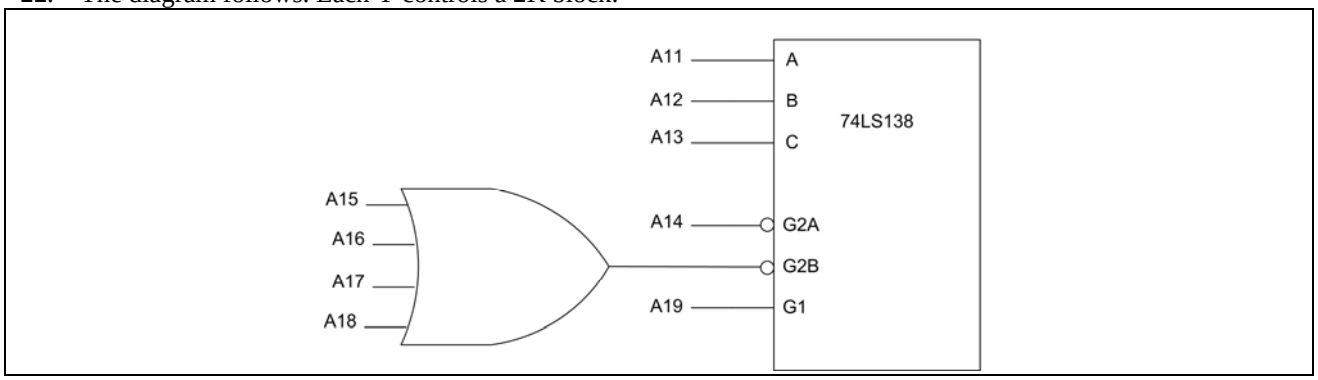
17. 98000H - 9FFFFH
18. The diagram follows.



19. for Y0 = F0000H - F1FFFH, Y3 = F6000H - F7FFFH, Y6 = FC000H - FDFFFH
20. The diagram follows. Each Y controls a 16K block.


A14
A15
A16
A17
A18
A19
A
B
C
G2A
G2B
G1
74LS138
Y0
CS
A13 ··· A0
A13 ··· A0

21. Y3 = 0C000 - 0FFFF, Y6 = 18000 - 1BFFF, Y7 = 1C000 - 1FFFF.
22. The diagram follows. Each Y controls a 2K block.



23. Y1 = 80800 - 80FFF, Y4 = 82000 - 827FF, Y5 = 82800 - 82FFF.
24. low RD is also active low.
25. CPLD

SECTION 10.3: IBM PC MEMORY MAP

26. 00000 - 9FFFFH, a total of 640K bytes for RAM; A0000H - BFFFFH, a total of 128K bytes for video RAM; C0000H - FFFFFH, a total of 256K bytes for ROM
27. from 00000 - 9FFFFH, a total of 640K bytes
28. No, DOS uses only the lowest 640K bytes. Furthermore, the addresses beyond 9FFFFH belong to video RAM; therefore, if we use them there will be a conflict with video.
29. CS = FFFFH IP = 0000
30. No. The address range 00000 - 9FFFFH is strictly managed by DOS. DOS uses whatever K bytes it needs and uses the rest for applications software. To use that area results in fragmentation of the 640K memory space and can result in a system crash.
31. From F000:FFF5 to F000:FFFD is the logical address and FFFF5 to FFFFD is the physical address.
32. Subtracting C0000 from C7FFF gives 7FFFH, but since the 0 is the first address location it will be 8000H bytes. Converting that to decimal and dividing by 1024 gives 32K bytes in ROM allocated space.
33. B0FFFH
34. BBFFFH
35. When the 8088 is powered up, the FFFF0H is the first physical location that the CPU fetches the op code from. Therefore the ROM (non-volatile) memory must be mapped at those addresses and not 00000. Furthermore, the 00000 belongs to the interrupt vector table.
36. FFFF0H; the opcode is EAH, the opcode for FAR JMP

SECTION 10.4: DATA INTEGRITY IN RAM AND ROM

37. 34H+54H+7FH+11H+E6H+99H=297H. The 2 is dropped and then the 2's complement of 97H=69H. Therefore the checksum byte is 69H.
38. (a) 29H + 1CH + 16H + 38H + 6DH =00, dropping the carries from the upper nibble. Therefore, the data is not corrupted.
    (b) 29H +1CH + 16H + 30H + 6DH = F8H, which is not zero. Therefore, some bytes are corrupted.
39. ROM, RAM
40. true
41. 320K bytes is broken down to 256K and 64K bytes blocks. It means that we need 9 of the 64Kx1 and 9 of the 256Kx1 memory chips, a total of 18 chips.
42. That gives two 256Kx4 and one 256Kx1 chip for the 256KB block, and two 64Kx4 and one 64kx1 chip for the 64KB block, for a total of 6 memory chips.

43. A higher density chip means a lower parts count, which leads to a smaller printed circuit board. Also, a lower number of parts leads to a lower number of system defects.
44. true
45. true
46. even = 1 and odd = 0
47. even = 0 and odd = 1
48. even = 1 and odd = 0

SECTION 10.5: 16-BIT MEMORY INTERFACING

49. 80286
50. 2 of each, a total of 8 chips
51. A0=1 and BHE=0
52. A0=0 and BHE=0
53. A0=0 and BHE=1
54. A0=1 and BHE=0
55. increases driving capability of the data pins
56. megabytes per second
57. the data bus width and bus cycle time
58. true
59. false
60. (a) 200 ns (2 × 100 ns) memory cycle gives (1/200 ns) × 2 bytes =10 megabytes/s
    (b) 125 ns (2 × 62.5 ns) memory cycle gives (1/125 ns) × 2 bytes = 16 megabytes/s

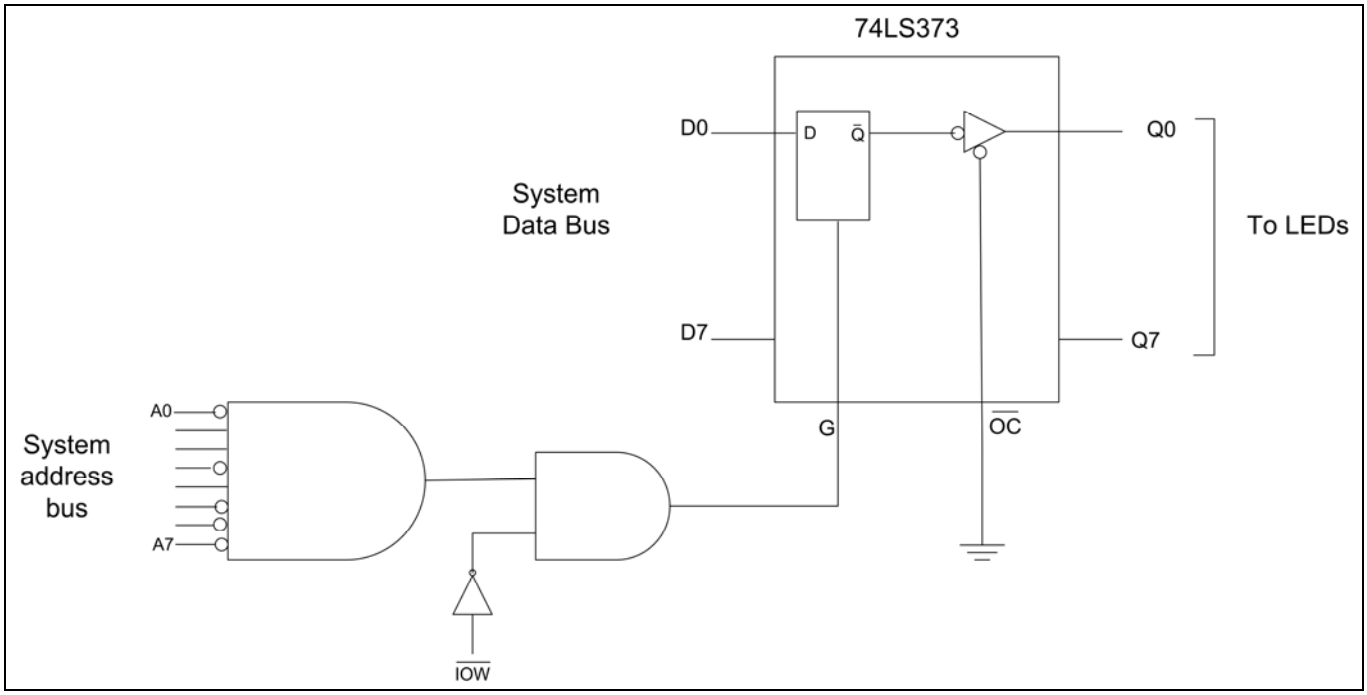
CHAPTER 11: 8255 I/O PROGRAMMING

SECTION 11.1: 8088 INPUT/OUTPUT INSTRUCTIONS

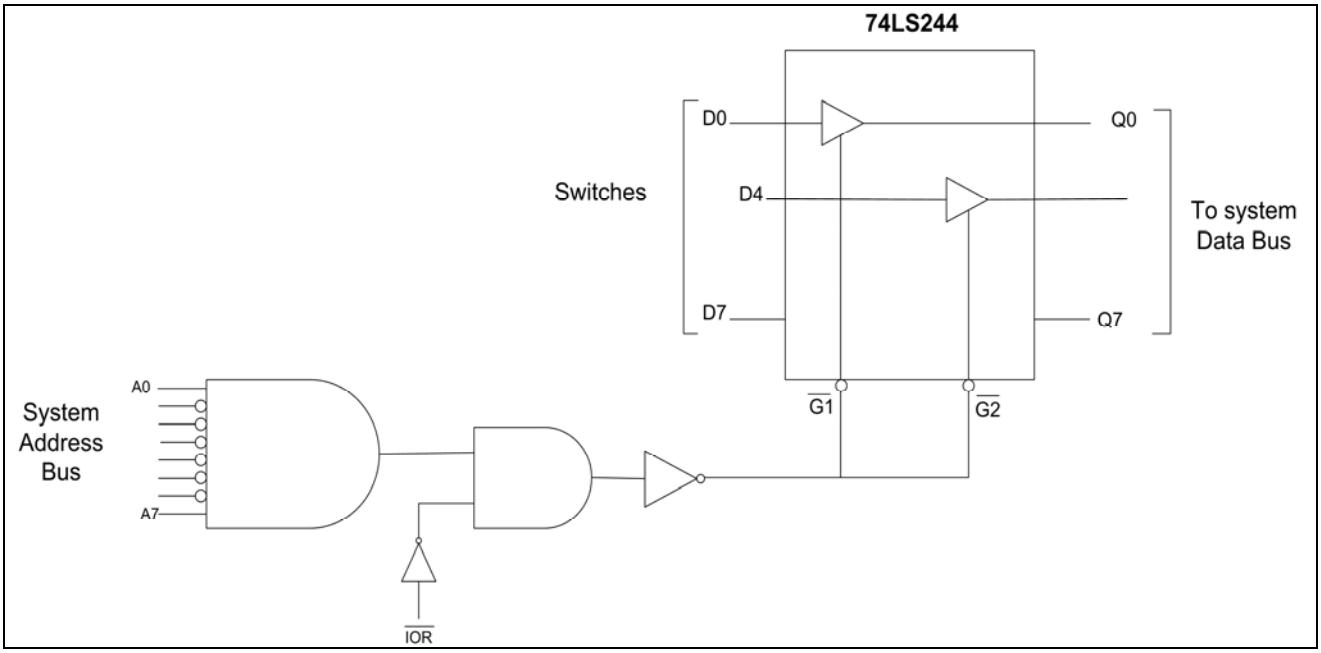
1. true
2. a
3. c, since it can go as high as 16 bits
4. true
5. The contents of port address 5FH are fetched into the CPU's AL register.
6. The contents of register AL are transferred to port addresses pointed at by DX, meaning that AL = 3BH goes to port address 300H.

SECTION 11.2: I/O ADDRESS DECODING AND DESIGN

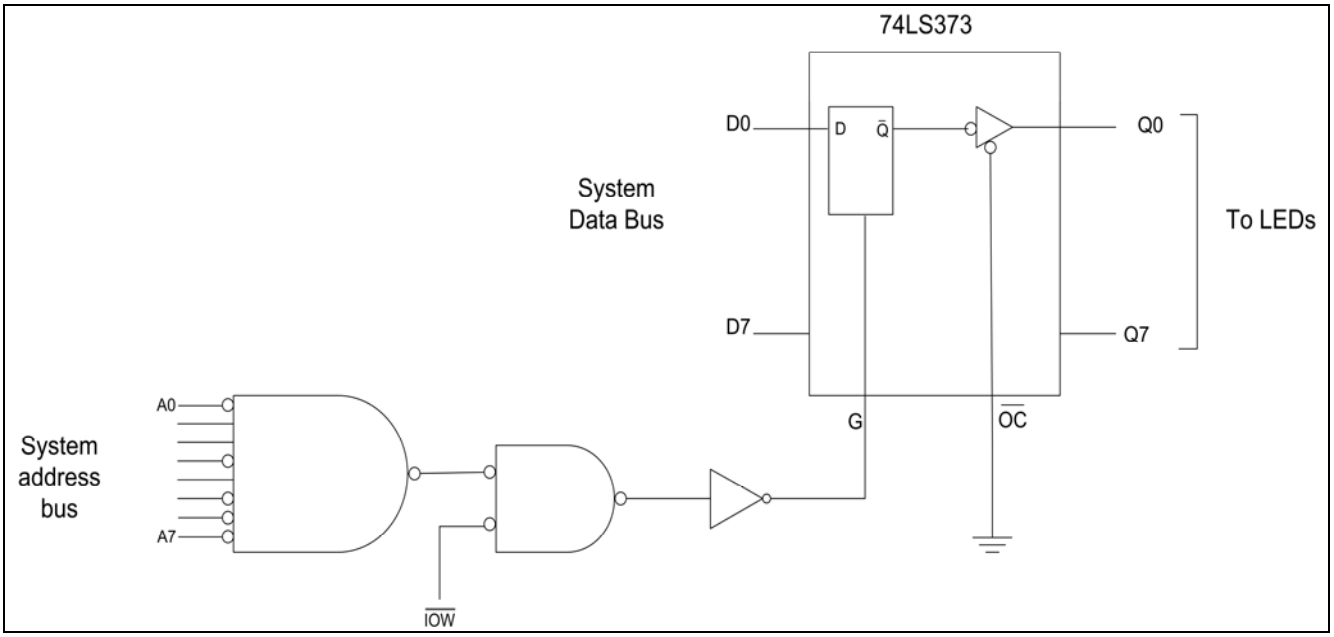
7. $\overline{IOW}$
8. $\overline{IOR}$
9. false
10. true
11. false, by the use of $\overline{IOW}$ or $\overline{IOR}$
12. true
13. tri-state buffer
14. latch
15. $\overline{IOR}$
16. $\overline{IOW}$
17. The diagram follows.



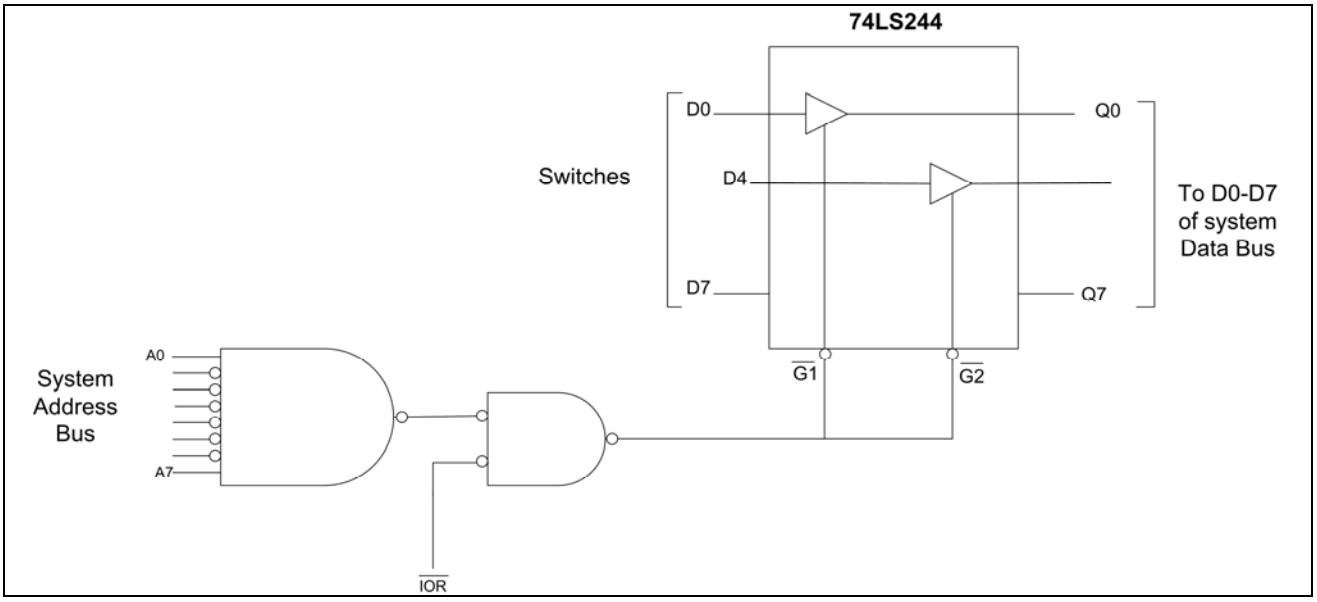
18. The diagram follows.



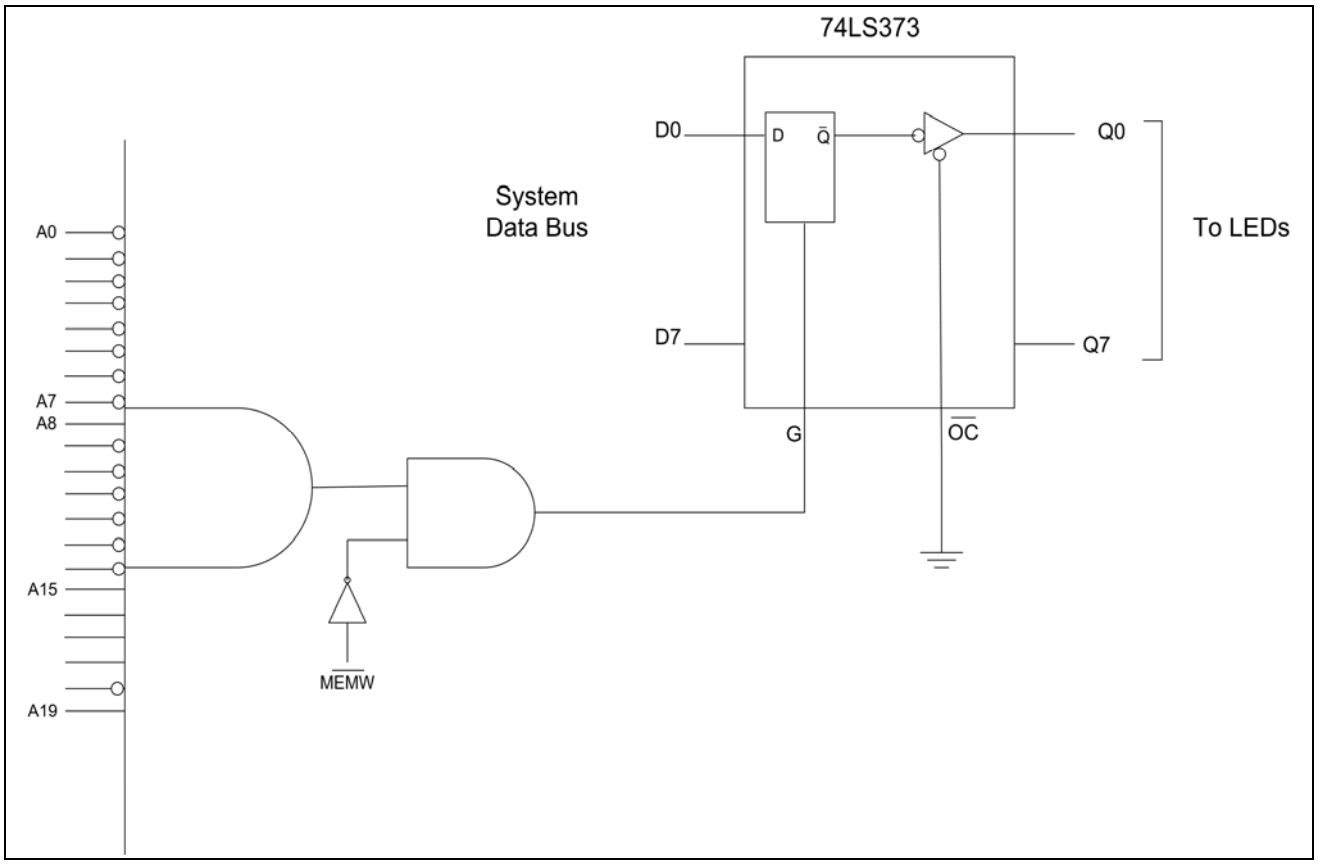
19. The diagram follows.



20. The diagram follows.



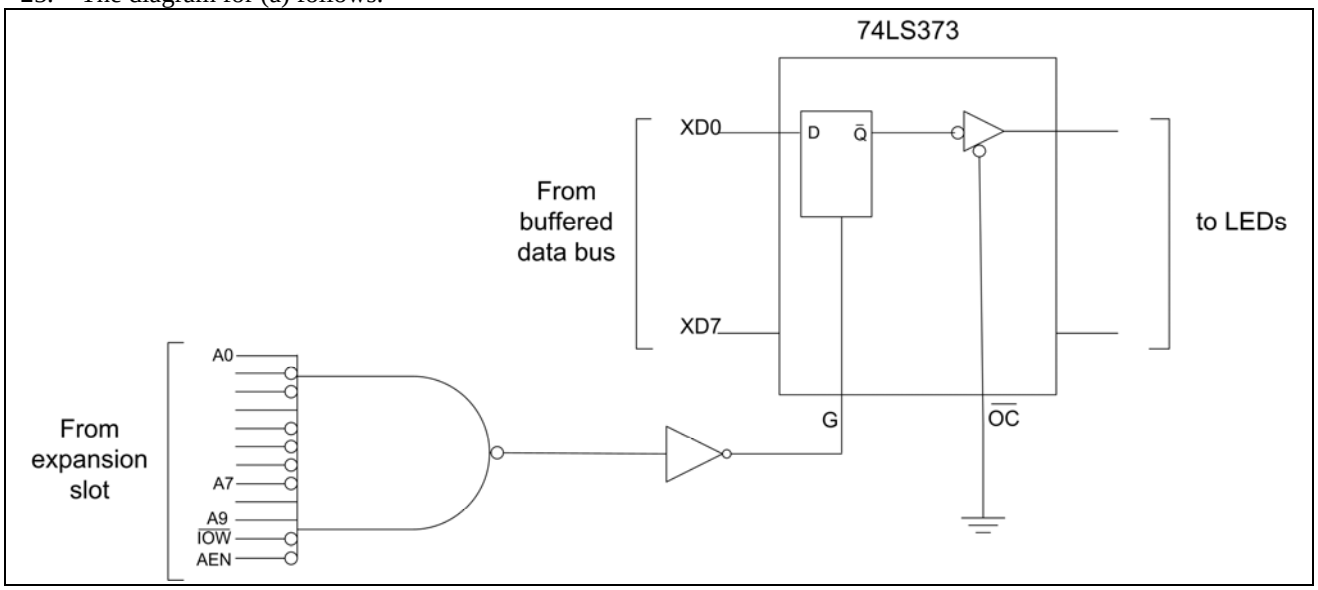
21. true
22. true
23. The physical address is B8000H + 0100H = B8100H; therefore, we have the following circuit.



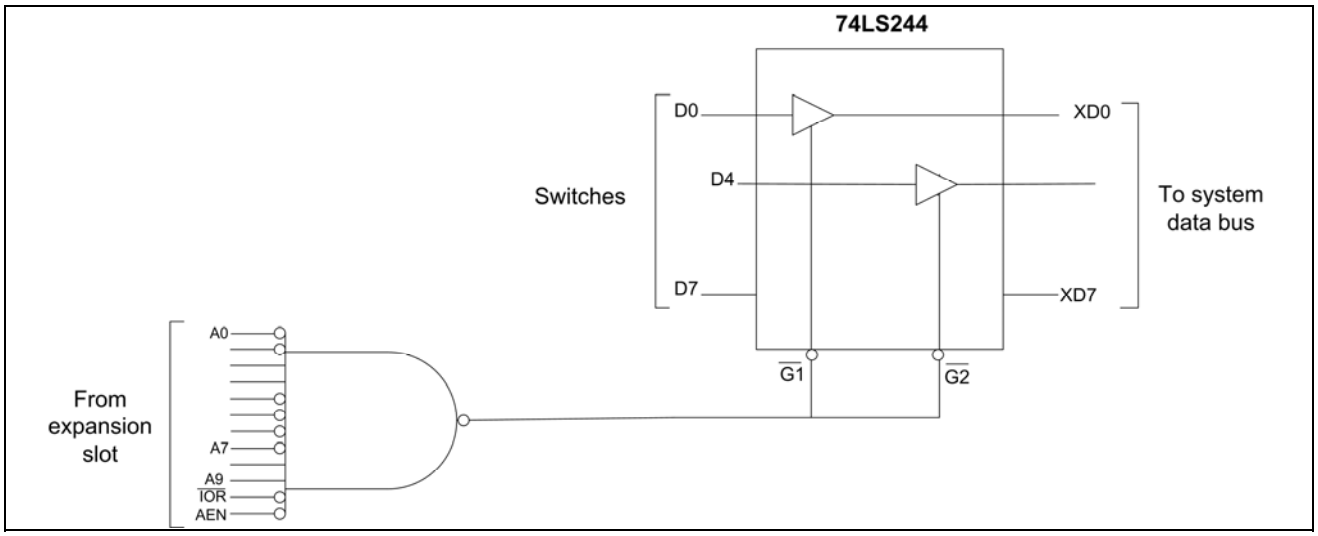
24. because decoding the entire 20-bit address requires many more inputs for the decoding circuitry and therefore is more expensive.

SECTION 11.3: I/O ADDRESS MAP OF x86 PCs

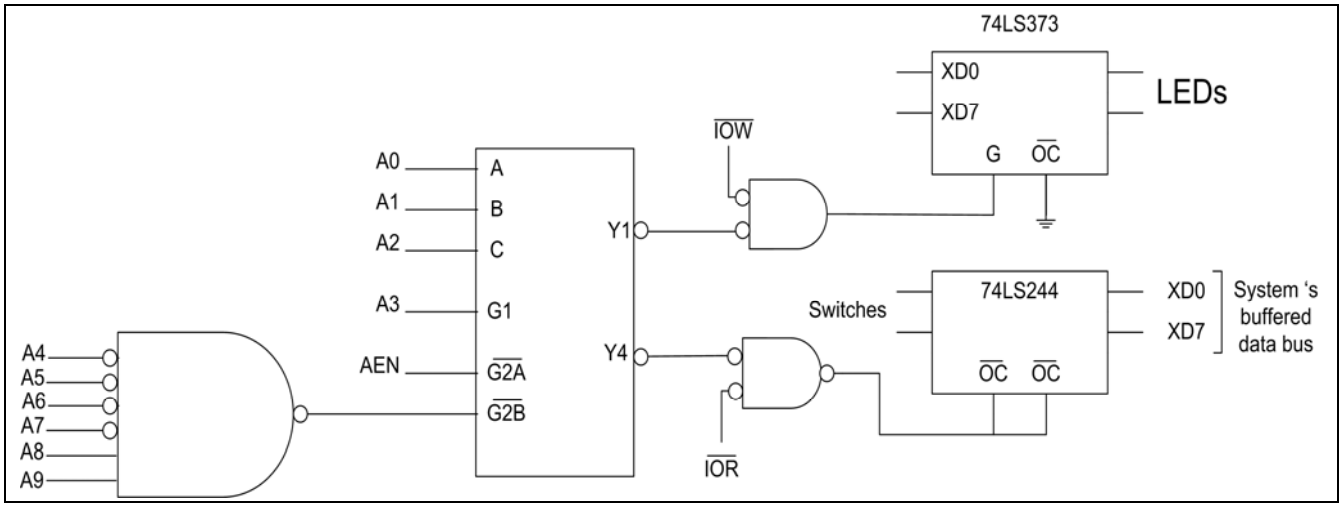
25. The diagram for (a) follows.



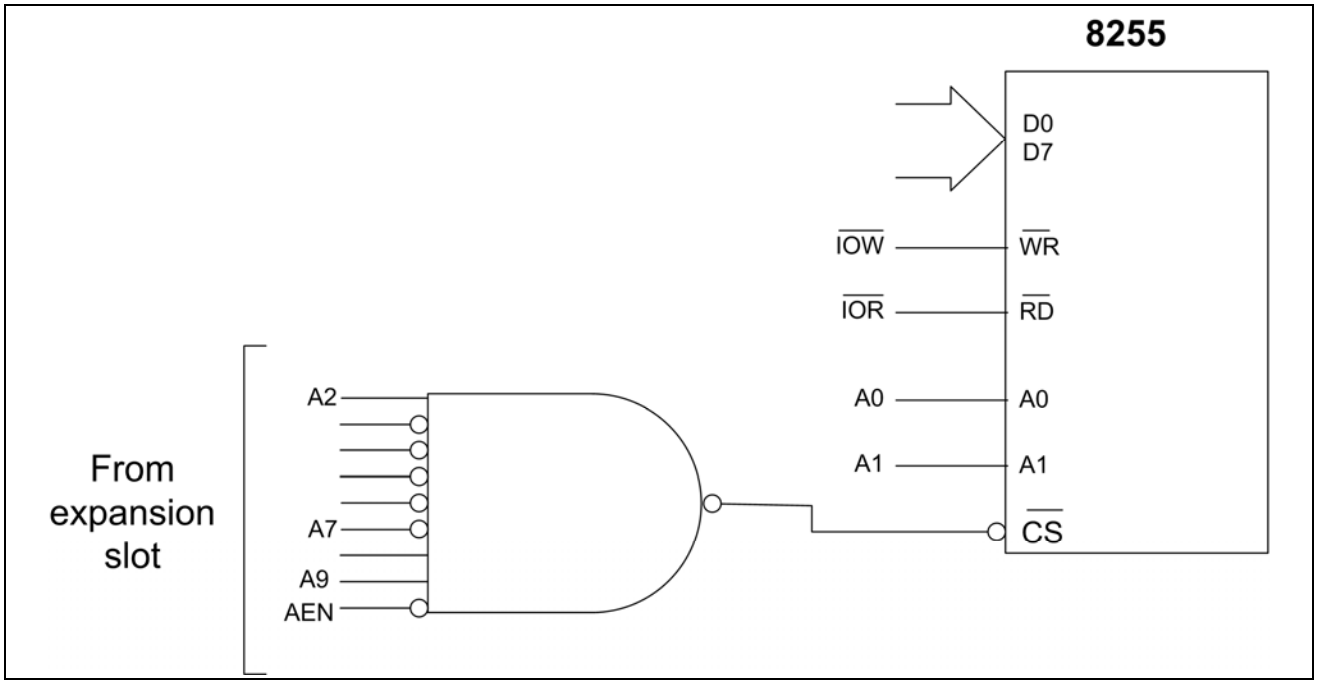
The diagram for (b) follows.



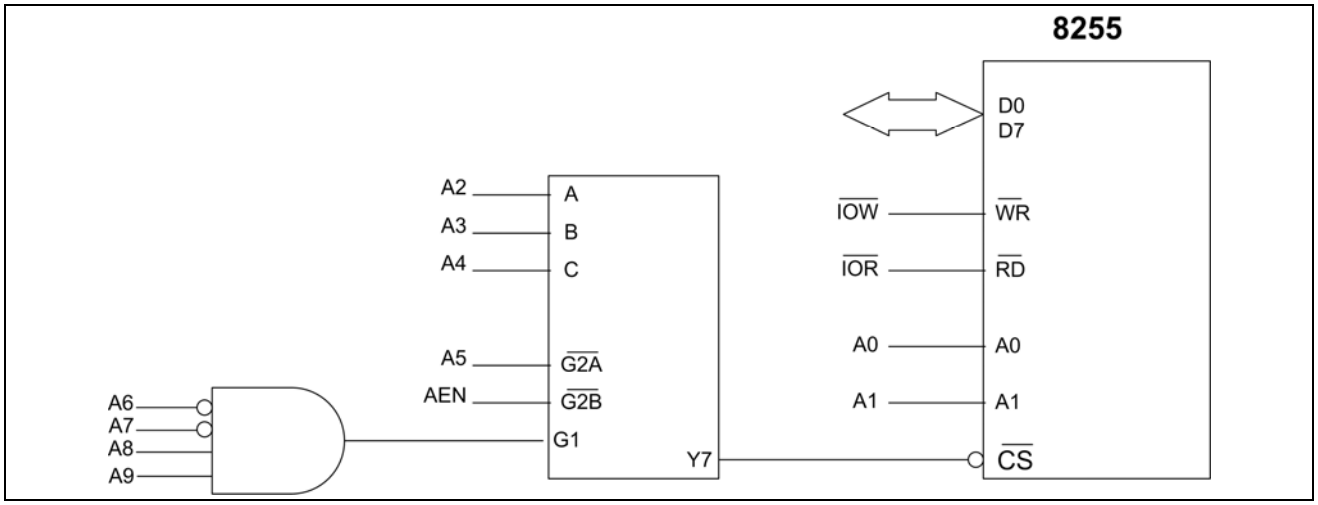
26. The diagram follows.



27. The diagram follows.



28. The diagram follows.

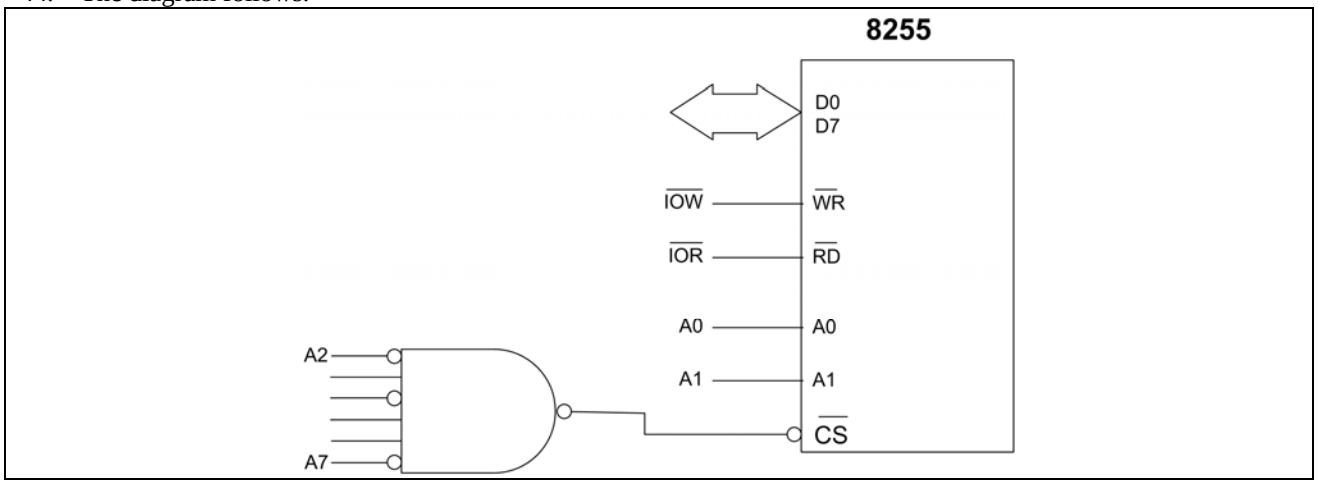


29. The port address space 300H - 31FH is set aside for prototype. It provides space for a total of 32 ports.
30. the linear select
31. Most often each port (or even memory) can be accessed by a single unique address. However, in address decoding, multiple addresses are assigned to a single port if some address lines are not used in the decoding circuitry. In this case, all the aliases refer to the same device just like a person with several names (aliases).
32. the linear select
33. true
34. The ports are programmed by the CPU. Therefore, we must use AEN = 0 to access them.
35. 61H
36. D4, 61H, 15.085 us
37. 0.25 sec/15.085 us = 16,572
38. 38,000 x 15.085 us = 573,230 us

SECTION 11.4: PROGRAMMING AND INTERFACING THE 8255

39. a total of 24 pins, 8 pins for each of PA0 - PA7, PB0 - PB7, and PC0 - PC7.
40. To transfer the control word to 8255 and data between the CPU and various ports of A, B, and C.
41. While 74LS373 and 74LS244 are fixed and static, the 8255 is programmable and therefore dynamic.
42. true
43. false, either all as input or all as output

44. The diagram follows.



45. None can be assigned since we must have A0 = 0 and A1 = 0 to access port A.
46. PA = IN, PB = OUT, PC0 - PC3 = IN and PC4 - PC7 = OUT
47. 92H
48. simple I/O mode
49. This means that the ports must be set as PA = in, PB = out, and PC = out. Therefore, we have

```
BA8255      EQU     68H             ;THE BASE ADDRESS OF 8255
            MOV     AL,10010000B    ;PA=in,PB=out, AND PC=out
            OUT     BA8255+3,AL     ;TO CONTROL REG
BACK:       IN      AL,BA8255       ;GET DATA FROM PORT A
            CMP     AL,100          ;IS IT 100?
            JNE     BACK            ;KEEP MONITORING
            MOV     BL,AL           ;IF 100 THEN SAVE IT
            MOV     AL,0AAH         ;AND SEND AAH
            OUT     BA8255+1,AL     ;TO PB
            NOT     AL              ;MAKE AL=55H
            OUT     BA8255+2,AL     ;AND SEND IT TO PC
```

50.

```
MOV     AL,90H ;PA=IN
MOV     DX,CRPORT
OUT     DX,AL
MOV     DX,PORTA
IN      AL,DX
MOV     BL,10
SUB     AH,AH
DIV     BL
MOV     CL,AH
SUB     AH,AH
DIV     BL
OR      AX,3030H
OR      CL,30H
```

CHAPTER 12: INTERFACING TO LCD, MOTOR, ADC, AND SENSOR

SECTION 12.1: INTERFACING TO AN LCD

1. 8
2. RS allows us to make a distinction between the data to be displayed and command codes. R/W allows us to read from internal registers of an LCD or write to an LCD. E is used to latch data (or command code) into an LCD.
3. VCC powers the LCD while VEE provides power for the contrast.
4. command code and it value is 01
5. 0E hex
6. RS=0, R/W=0 and a High-to-Low pulse for E
7. We put ASCII code for Z (59H) on the D0 - D7 data bus, then we make RS=1, R/W=0 and a High-to-Low pulse for E
8. (a)
9. True
10. (1) In the checking the busy flag method, we send in information (command code or data) only when the LCD is ready to accept it. In this way, no data gets lost. However, extra software is needed to monitor the busy flag. In addition, the data port must be a bidirectional port since the busy flag (BF) is D7 of the data pin.
    (2) In the second method, we put a sufficient delay in between each sending of code (or data) to the LCD. This delay makes sure that the LCD is not overwhelmed with a string of data. However, we must make sure that the delay is long enough before the next character is sent in. Otherwise, data is lost. In some cases the LCD will go blank if the delay is too short. The main advantage of this method is that it does not require a bidirectional data port; therefore, it is ideal for the unidirectional printer ports. It also requires less software. Now since the busy flag is D7 of the data bus of the LCD, to read the busy flag we must have RS=1, R/W=1, E=1. That means that we must change the direction of our port connected to D0 - D7 of the LCD to input. Then we read D0 - D7 and mask D0 -D6 of it. Now if D7 = 1, LCD is not ready. If D7 = 0, it is ready to accept new information.

11. In the 16x2 LCD, we have 16 (0 to 15) characters for each line with an address range of 0000 to 1111 (1111 in binary, 15 in decimal). Therefore, from Table 12-3 we have 1000000 (80H) for line 1's first character and 10001111 (8FH) as the last character of line 1.

12. In the 16x2 LCD, we have 16 (0 to 15) characters for each line with an address range of 0000 to 1111 (15 in decimal). Therefore, from Table 12-3 we have 1100000 (C0H) for line 2's first character and 11001111 (CFH) as the last character of line 2.

13. In the 20x2 LCD, we have 20 (0 to 19) characters for each line with an address range of 00000 to 10011 in binary (19 in decimal). Therefore, from Table 12-3 we have 1000000 (80H) for line 1's first character and 10010011 (93H) as the last character of line 1.

14. In the 20x2 LCD, we have 20 (0 to 19) characters for each line with an address range of 00000 to 10011 (19 in decimal). Therefore, from Table 12-3 we have 1100000 (C0H) for line 2's first character and 11010011 (D3H) as the last character of line 2. Therefore, the third character has an address of C2 since C0, C1 are for the first and the second characters, respectively.

15. In the 40x2 LCD, we have 40 (0 to 39) characters for each line with an address range of 000000 to 100111 in binary (39 in decimal). Therefore, from Table 12-3 we have 1000000 (80H) for line 1's first character and 10100111 (A7H) as the last character of line 1.

16. In the 40x2 LCD, we have 40 (0 to 39) characters for each line with an address range of 0000 to 100111 in binary (39 in decimal). Therefore from Table 12-3 we have 1100000 (C0H) for line 2's first character and 11100111 (E7H) as the last character of line 2.

17. It is 10001001 (89H) because if the first character's address is 10000000 (80H), the tenth (1001) one is 10001001 (89H).

18. If we have 1100000 (C0H) for line 2's first character, the twentieth character (10011 in binary or 19 in decimal) is 11010011 (D3H).

19. The following is the comparison of the old version and the new version:

OLD		**NEW**	
PUSH	DX	PUSH	DX
MOV	DX,PORTA	MOV	DX,PORTA
OUT	DX,AL	OUT	DX,AL
MOV	DX,PORTB	MOV	DX,PORTC
MOV	AL,00000100B	MOV	AL,01000000B
OUT	DX,AL	OUT	DX,AL
NOP		NOP	
NOP		NOP	
NOP		NOP	
NOP		NOP	
MOV	AL,00000000B	MOV	AL,00000000B
OUT	DX,AL	MOV	DX,AL
POP	DX	POP	DX
RET		RET	

20.

OLD		**NEW**	
PUSH	DX	PUSH	DX
MOV	DX,PORTA	MOV	DX,PORTA
OUT	DX,AL	OUT	DX,AL
MOV	DX,PORTB	MOV	DX,PORTC
MOV	AL,00000101B	MOV	AL,01010000B
OUT	DX,AL	OUT	DX,AL
NOP		NOP	
NOP		NOP	
NOP		NOP	
NOP		NOP	
MOV	AL,00000001B	MOV	AL,00010000B
OUT	DX,AL	MOV	DX,AL
POP	DX	POP	DX
RET		RET	

```
            .DATA
MYDATA      DB      "HELLO"
            .CODE
            ...
            ...
            MOV     CX,05               ;display this number of bytes
            MOV     SI,OFFSET MYDATA    ;load data address
            CALL    LCD_DISPLAY
            ...
            ...
LCD_DISPLAY:
NEXT:       MOV     AL,[SI]             ;get the char
            CALL    NDATWRIT            ;issue it to LCD
            INC     SI                  ;point to next char
            CALL    DELAY               ;wait
            LOOP    NEXT                ;now the next one until all is finished
            RET
NDATWRIT:
```

```
PUSH   DX
MOV    DX,PORTA
OUT    DX,AL
MOV    DX,PORTC
MOV    AL,01010000B
OUT    DX,AL
NOP
NOP
NOP
NOP
MOV    AL,00010000B
MOV    DX,AL
POP    DX
RET
```

SECTION 12.2: INTERFACING TO A STEPPER MOTOR

21. 4, since 360/90=4
22. 360/7.5=48
23. 0011,1001,1100,0110
24. 1100,0110,0011,1001
25. 1001,0011,0110,1100
26. 0110,1100,1001,0011
27. Since the 8255's Port A does not provide sufficient current to drive the stepper motor, we need a driver. The ULN2003 Darlington pair is one such driver, providing a maximum of 600 mA. Therefore,it cannot be used for a 3 amp motor.
28. (b)
29. First, it allows the EM field to collapse before it is activated again. This is the absolute minimum delay needed before the next sequence is issued. Second, we can control the speed of rotation with the time delay.
30. The shorter the time delay in between the sequence, the faster the motor speed. However, there is a need for a minimum delay to allow the EM field to collapse before it is activated again by the next sequence.

SECTION 12.3: INTERFACING TO A DAC

31. True, yes
32. It is 2^n
    (a) 256 (b) 1024 (c) 4096
33. If all the inputs are high, we have:
    2 mA (1/2 + 1/4 + 1/8 + 1/16 + 1/32 + 1/64 + 1/128 + 1/256) = 2mA x 255/256 = 1.99 mA
34. (a) 10011001 = 2 mA (1/2 + 1/16 + 1/32 + 1/256) = 2 mA x 153/256 = 1.195 mA
    (b) 11001100 = 2 mA (1/2 + 1/4 + 1/32 + 1/64) = 2mA x 204/256 = 1.593 mA
    (c) 11101110 = 2 mA x 238/256 = 1.859 mA
    (d) 00100010 = 2 mA x 34/256 = 0.265 mA
    (e) 00001001 = 2 mA x 9/256 = 0.07 mA
    (f) 10001000 = 2 mA x 136/256 = 1.062 mA
35. more
36. all inputs must be high

SECTION 12.4: INTERFACING TO ADC CHIPS AND SENSORS

37. CS=0 and a Low-to-High pulse for WR
38. CS=0 and WR=1 and H-to-L pulse for RD to read the data out of ADC

39. The converted data is kept inside ADC848 momentarily. In other words, after the input analog data is converted, it is kept inside and it activates the INTR (makes it low) to let us know that the data is converted and ready to be picked up.
40. it is lost
41. output. Through the INTR signal, the ADC848 lets us know analog data is converted and is ready to be picked up.
42. (a) 5 mV
    (b) 3.9 mV
    (c) 7.42 mV
43. V_{in} = 20 mV x 256 = 5.12 V; therefore, V_{ref} = 5.12 V
    (you cannot do this since it is more than V_{cc})
44. V_{in} = 5 mV x 256 = 1.28 V; therefore, V_{ref} = 1.28 V
45. Provide the channel address to the D0 - D7 pins, set CS = 0, and apply an L-to-H pulse to the WR pin.
46. 255 x 19.53 mV = 4.999 V
47. Since the step size is 1.28 V / 256 = 5 mV, we have
    (a) 11111111 = 255 and 255 x 5 mV = 1.27 V
    (b) 10011001 = 153 and 153 x 5 mV = 765 mV
    (c) 11001100 = 204 and 204 x 5 mV = 1.02 V

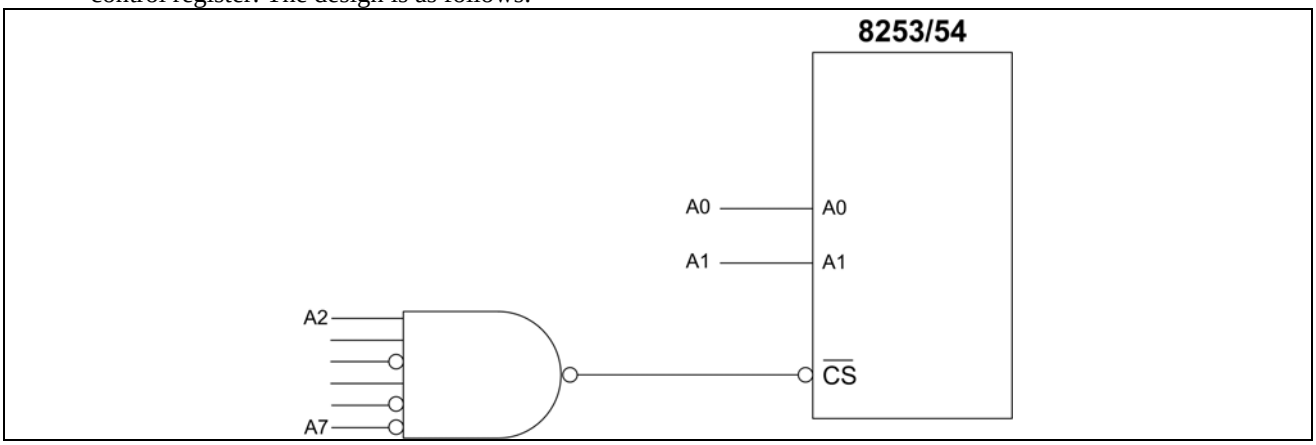
48. The output changes proportionally with the input in a straight line fashion (in contrast with non-linear, in which the change follows a curve).
49. 10 mV
50. Physical quantities such as temperature, pressure, light, and moisture need to be represented in voltage in order to be sampled by an analog-to-digital converter (ADC). It is the job of transducers to sense the physical quantity; however, their output could be in the form of current, resistor, or capacitance. Even if the output is in the form of voltage, it often needs amplification, which is the job of signal conditioning. Therefore, signal conditioning is the process of converting the output of transducers to voltage forms of sufficient magnitude in order to be sampled by A-to-D converters. There exists the following stages between physical quantities and the PC:

    Physical Quantities→Transducer→Signal Conditioning→ADC→PC

CHAPTER 13: 8253/54 TIMER

SECTION 13.1: 8253/54 TIMER

1. true
2. input, square wave
3. The base port address is 2CH. The port addresses are 2CH: counter 0;2DH: counter 1; 2EH:counter 2; and 2FH: control register. The design is as follows:



4. 23H, 97H, 51H, and 59H since counter 0 must have A0 = 0 and A1 = 0
5. In binary mode, the highest value is 65,536 and programmed as 0000. In BCD mode, it is 10,000 and programmed as 0000.
6. true
7. 76H
8.

```
MOV   AL,76H      ;countr1,mode 3,binary,lo byte,and hi byte
OUT   2FH,AL
MOV   AX,1333     ;divisor=1.6M/1200=1333
OUT   2DH,AL      ;the low byte first
MOV   AL,AH
OUT   2DH,AL      ;then the high byte
```

9.

```
MOV   AL,76H      ;countr1,mode 3,binary,lo byte,and hi byte
OUT   2FH,AL
MOV   AX,6400     ;divisor=1.6M/250=6400
OUT   2DH,AL      ;the low byte first
MOV   AL,AH
OUT   2DH,AL      ;then the high byte
```

10. In the binary option, the maximum divisor is 65,536; therefore, we have 1.6 MHz /65536 = 24 Hz. For the BCD, option we have 160 Hz since 1.6 MHz /10000 =160 Hz.

SECTION 13.2: x86 PC 8253/54 TIMER CONNECTION AND PROGRAMMING

11. CLK0, CLK1, and CLK2 are connected to a fixed frequency of 1.19 MHz.
12. The source of the square wave is the 8284's PCK which is 2.38 MHz. After it is divided by 2 using a simple D-FF, 1.19 MHz is fed to all CLKs inputs.
13. 40H - 43H. No, they cannot be changed. If a given clone does not use these port addresses, it is not compatible with the IBM PC.

14. Counter0 is used by IRQ0 to cause the interrupt 18.2 times per second. Counter1 is used for the DRAM memory refreshing circuitry. Counter2 is connected to the PC speaker.
15. true
16. GATE0 and GATE1 are connected to VCC permanently. GATE2 is controlled by D0 of port address 61H.
17. This is due to the fact that such a time delay is not fixed and can vary depending on the speed of the 80x86 processor in different PCs, plus the fact that the clock count of a given instruction varies among the 80x86 family.
18.

```
        MOV     BL,20           ;20 X 0.5SEC=10SEC
AGAIN:  MOV     CX,33144        ;0.5SEC DELAY
        CALL    WAITF
        DEC     BL
        JNZ     AGAIN
;for the WAITF subroutine see the text.
```

SECTION 13.3: GENERATING MUSIC ON THE x86 PC

19. A3 frequency is 220 Hz; therefore, 5423 is the divisor since 1.1931 MHz/5423 = 220 Hz. By the same token, we have a divisor of 1522 for G5 and 604 for B6.
20.

```
PAGE    60,132
;THIS PROGRAM USES THE BUILT-IN TIMER CLOCK #2 TO PLAY A SONG
;OVER THE INTERNAL SPEAKER
;*******************************************************
MYSTAK          SEGMENT
                DB      32 DUP(?)
MYSTAK          ENDS
;*******************************************************
MYDATA          SEGMENT
FREQS           DW      330,1,294,1,262,1,294,1,330,1,330,1
                DW      330,2,294,1,294,1,294,2,330,1,392,1
                DW      392,2,330,1,294,1,262,1,294,1,330,1
                DW      330,1,330,1,330,1,294,1,294,1,330,1
                DW      294,1,262,4
MYDATA          ENDS
;*******************************************************
MYCODE          SEGMENT
MAIN            PROC    FAR
                ASSUME  CS:MYCODE, DS:MYDATA, SS:MYSTAK
                MOV     AX,MYDATA
                MOV     DS,AX
                MOV     AL,0B6H         ;program timer clock 2
                OUT     43H,AL
                LEA     DI,FREQS        ;pointer to notes & counts
HERE:           MOV     AX,34DEH        ;1.193182 MHz = 001234DEH
                MOV     DX,0012H        ;clock freq is loaded in numerator
                MOV     BX,[DI]         ;denominator
                CMP     BX,0000H        ;check for end of song
                JE      THU             ;if end of song, back to DOS
                DIV     BX              ;calculate divide #
                OUT     42H,AL
                MOV     AL,AH           ;high byte divider
                OUT     42H,AL
                IN      AL,61H          ;save speaker status
                MOV     AH,AL
                OR      AL,03           ;turn speaker on
```

```
                OUT     61H,AL
                INC     DI
                INC     DI
                MOV     BX,[DI]         ;set note delay
                CALL    DELAY           ;note delay
                INC     DI
                INC     DI
                MOV     AL,AH           ;restore speaker status
                OUT     61H,AL          ;and turn speaker off
                CALL    DELAY2
                JMP     HERE            ;get next note
THU:            MOV     AH,4CH          ;end of song return to DOS
                INT     21H
MAIN            ENDP                    ;end main procedure
;*********************************************************
DELAY           PROC
                PUSH    AX              ;save AX
AGAIN1:         MOV     CX,16578        ;16578 * 15.08us = 250ms
AGAIN:          IN      AL,61H          ;run 15.08 us delay
                AND     AL,10H
                CMP     AL,AH
                JE      AGAIN
                MOV     AH,AL
                LOOP    AGAIN           ;end of 250 ms
                DEC     BL              ;decrement note count
                JNZ     AGAIN1          ;more time if note is longer
                POP     AX              ;restore AX
                RET
DELAY           ENDP
;*********************************************************
DELAY2          PROC
                MOV     CX,1328         ;1328 * 15.08 us = 20 ms
REPEAT:         IN      AL,61H
                AND     AL,10H
                CMP     AL,AH
                JE      REPEAT
                MOV     AH,AL
                LOOP    REPEAT          ;end of 20ms
                RET
DELAY2          ENDP
;*********************************************************
MYCODE          ENDS
                END     MAIN
```

CHAPTER 14: INTERRUPTS IN x86 PC

SECTION 14.1: 8088/86 INTERRUPTS

1. Yes. It finishes the current instruction and saves the address of the next instruction on the stack before it gets the CS:IP of the interrupt service routine. The stack provides the returning address.
2. (a) For INT 05 we have logical address = 0000:0014H and physical address = 00014H.
   (b) For INT 21H, we have LA = 0000:0084H and PA = 00084H.
3. ISR stands for interrupt service routine, and is the program associated with the INT type. When an interrupt is invoked, it is asked to run the ISR. Another name for ISR is interrupt handler.
4. the interrupt vector table
5. (a) In the CALL FAR, only 4 bytes of the stack is used, 2 for the IP and 2 for the CS of the next instruction.
   (b) In the interrupt, 6 bytes are used, 2 for the flag register, 2 for IP and 2 for the CS of the next instruction.
6. INT FEH or INT 254
7. The logical address is 0000:0000 to 0000:03FFH and physical address is 00000 to 003FFH.
8. 1024 bytes, since there are 256 interrupts and each interrupt takes 4 bytes.
9. because it is set aside for the interrupt vector table
10. INT 00 for the divide error or quotient oversize
11. (a) INT 00 (b) INT 01 (c) INT 02
12. true
13. false, only the INTR
14.
```
PUSHF
MOV     BP,SP
OR      BP+0,0000 0001 0000 0000B      ;binary position of TF
POPF
```
15. (a) The IF is set to low by instruction CLI.
    (b) It is set to high by instruction STI.
16. true
17. IRET, RETF
18. In RETF, only the 4 bytes are popped from the stack, while IRET pops off 6 bytes FR,CS, and IP.

19.

	for Interrupt	for CALL FAR
SS:FFDA	IP (low)	
SS:FFDB	IP (high)	
SS:FFDC	CS (low)	IP (low)
SS:FFDD	CS (high)	IP (high)
SS:FFDE	FR (low)	CS (low)
SS:FFDF	FR (high)	CS (high)
SS:FFE0		

20. none; the sequence is IP, CS and FR

SECTION 14.2: x86 PC AND INTERRUPT ASSIGNMENT

21. (a) This belongs to INT 07
    (b) F000:FF47H is the logical address and FFF47H is the physical address.
22. BIOS
23. c
24. CS = F000H and IP = FE6EH
25. 0000:0070 = 6EH, 0000:0071 = FEH,0000:0072 = 00H, 0000:0073 =F0H
26. locations 0000:0413H and 0000:0414H

SECTION 14.3: 8259 PROGRAMMABLE INTERRUPT CONTROLLER

27. false, A0 = 0
28. All are input, except INT.
29. The port address for ICW1 is 94H while ICW2, ICW3, and ICW4 all have the port address of 95H.
30. 00010110 = 16H for ICW1. The ICW2 is always the INT # assigned to IR0; therefore, the ICW2=88H.

31.

```
MOV     AL,16H          ;no ICW4,SNGL,x86,EDGE TRIG
OUT     94H,AL
MOV     AL,88H          ;INT 88h is assigned to IR0
OUT     95H,AL          ;output the ICW2
```

32. a, c, d, since the lower nibble must be 8 or 0. The 8259 provides the 3 bit combinations for IR0 - IR7.
33. IR0 will have the INT 18H, and IR7 the INT 1FH.
34. IR0 has 30H, IR4 has INT 34H, and IR6 has INT 36H
35. OCW1
36. OCW2
37. IR0 has the highest priority, and IR7 the lowest priority.
38. OCW1 has the port address of 95H, and OCW2 and OCW3 both have the same port address of 94H.

39.

```
MOV     AL,11101011     ;OCW1: MASK ALL EXCEPT IR2&IR4
OUT     95H,AL          ;SEND THE OCW1
```

40. Notice that D4 = 1 in ICW1, while in OCW2 and OCW3, D4 = 0. In OCW2 we have D3 = 0, while in OCW3, D3 = 1

SECTION 14.4: USE OF THE 8259 CHIP IN x86 PCs

41. because the 8259 must be programmed by the main CPU the 80x86, and not the DMA; this means that AEN = 0 and must be used in the address decoding circuitry to access the 8259 chip.
42. single
43. edge
44. INT 08 to INT 0FH
45. port addresses 20H and 21H
46. true
47. IRQ2 to IRQ7
48. side B
49. 0010 0000 = 20H and is sent to port address 20H for OCW2
50. IRQ0 since BIOS programs the 8259 in the default fully nested mode
51. true
52. true
53. Yes, through D7 of the port address A0H.
54. true

SECTION 14.5: MORE ON INTERRUPTS IN x86 PCs

55. true
56. true
57. The primary (master) has 20H and 21H for the port address. the secondary (slave) has A0H and A1H as port addresses.
58. INT 70H to 77H
59. IRQ3 to IRQ7; IRQ9, IRQ10, IRQ11, IRQ12, IRQ14, IRQ15

60. The IRQ2 is used for the cascade connection of the 2nd 8259; therefore, it is not available at the expansion slot. IRQ9 is provided at the pin where IRQ2 used to be in the PC/XT. Internally, BIOS redirects IRQ9 to IRQ2.
61. (a) The port address where the EOI is issued for the primary 8259 is 20H, and for the secondary 8259 is A0H.
(b) If any of the IRQ0, IRQ1, IRQ3 - IRQ7 is activated, the EOI is issued only to the primary 8259. However, if any of IRQ8 - IRQ15 from the secondary 8259 are activated not only must the EOI be issued to the secondary, but also to the primary 8259 since these IRQs come in through IRQ2 of the primary (master) 8259.
62. true
63. false; they have higher priority.
64. true
65. IRQ10 and IRQ6 both come to the CPU through the INTR; therefore, they have lower priority than NMI. It means that the CPU takes care of the NMI first. Then IRQ10 is serviced, and finally IRQ6 is serviced.
66. IRQ15 is serviced first, then IRQ3, and finally IRQ7. At the end of the interrupt service routine for the IRQ15, the issuing of the EOI will allow the lower priority IRQs to be responded to; therefore, the IRQ3 is serviced next. Again, at the end of the ISR for IRQ3, the issuing of the EOI allows the system to service IRQ7.

CHAPTER 15: DIRECT MEMORY ACCESS AND DMA CHANNELS IN x86 PC

SECTION 15.1: CONCEPT OF DMA

1. For the 8088 it takes 39 clocks, while for DMA it takes 4 clocks to transfer a byte. The DMA is 9 times faster.
2. 512 × 39 × 200 ns = 3.9936ms for the 8088; 512 × 4 × 200 ns = 0.4096 ms for DMA.
3. In response to INTR, the CPU finishes the current instruction, saves the address of the next instruction, and then responds by activating INTA. However, in response to HOLDR, the CPU finishes the current bus cycle and freezes the action (retains everything), then responds with HOLDA. The bus cycle could be in the middle of the execution of an instruction.
4. input, output
5. b
6. HOLDA
7. The DMA does not require an opcode interpreter and executor since it simply transfers information (code and data) and has no idea what the data is. This results in using substantially fewer transistors and shorter design time.
8. when HOLDR is pulled down (deactivated) by the DMA

SECTION 15.2: 8237 DMA CHIP PROGRAMMING

9. 16
10. (a) 88H and (c) 92H, since for the base address the lower nibble address (A3 - A0) must be 0
11. The memory address of the first location and count value. There is a single port for each.
12. Each channel must have a port for the memory address register and the count register. Since there are 4 channels,
    we need 8 port addresses for the channels.
13. 50H - 57H. 50H and 51H for channel 0, 52H and 53H for channel 1, and so on.
14. 58H - 5FH
15. command register; port address 58H
16. Channel 0 has the highest priority and channel 3 the lowest. In this mode, channel 0 always has the highest priority, but in rotating mode, after channel 0 is served once it will not get a chance until all other channels have had a chance to be served.
17. DREQ2
18. 8237 DMA
19. When the entire block of data has been transferred by a channel, the count register reaches 0. This is indicated by the EOP pin (hardware) of the 8237. It also sets the status register terminal count (TC) bit to high, allowing software monitoring of the terminal count. No, this is a read-only register.
20. The control byte is D7 - D0 = 1011 0110 = B6H.

```
        MOV   AL,0B6H          ;block mode, addr decr, autoinc, write
        OUT   5BH,AL           ;transfer, chan 2 to mode register
```

21.

```
        MOV   AL,03H           ;clear mask bit for chan 3
        OUT   5AH,AL           ;send to single mask register
```

22. Assuming that 50H is the base address of the 8237, we have:

```
        MOV   AX,1500H         ;starting address
        OUT   56H,AL           ;low byte to chan 3 memory address reg
        MOV   AL,AH
        OUT   56H,AL           ;hi byte to chan 3 memory address reg
        MOV   AX,8192          ;8K-byte block
        OUT   57H,AL           ;low byte to chan 3 count reg
        MOV   AL,AH
        OUT   57H,AL           ;hi byte to chan 3 count reg
```

SECTION 15.3: 8237 DMA INTERFACING IN THE IBM PC

23. The 8237 provides only 16 bits for the address. It provides the 8 bits of the higher part of the address through the data bus along the ADDSTB signal for the address to be latched. The lower 8 bits of the address are provided by pins A7 - A0.
24. There must be a file register to provide the most significant 4 bits of the address. The IBM PC/XT uses a 4x4 file register. That is 4 bits for each channel.
25. True
26. False, there is only one HOLD in the 8237.
27. (a) HOLD, (d) DACKs, and (f) ADDSTB are all out; (b)HOLDA and (c) DREQs are all in; the rest are bidirectional.

SECTION 15.4: DMA IN x86 PCs

28. True
29. a total of 7; DREQ1 - DREQ3 on the 62-pin part, and DRQ0 and DRQ5 - DREQ7 are available on the 36-pin part.
30. because their cycle times are not the same; The memory is on the motherboard but the disk is connected to the expansion slot through a cable and cables cannot handle such a fast data transfer rate.
31. This is because they will not be ISA compatible. The ISA bus has only A0 - A23 address bus which means it can access only 16M of memory space. In addition, to access the 4G address range by the DMA requires expanding the DMA page register from 24 bits to 32 bits.

CHAPTER 16: VIDEO AND VIDEO ADAPTERS

SECTION 16.1: PRINCIPLES OF MONITORS AND VIDEO MODES

1. (a) 909 and 262 (b) 744 and 364 (c) 799 and 525
2. (a) 269 and 62 (b) 104 and 14 (c) 159 and 45
3. For an example of the calculation, see the solution for Problem 4.

	640x480	800x600	1024x768	1280x1024
14"	10.92"	10.92"	10.98"	11.50"
15"	11.85"	11.7"	11.98"	12.14"
17"	13.41"	13.26"	13.47"	14.06"
20'	15.6"	15.6"	15.47"	15.98"

4. $[(640 \times 0.5 \text{ mm})^2 + (480\ 0.5 \text{ mm})^2]^{1/2} = 400$ mm. 400 mm x 0.039 = 15.6", which is much larger than the diagonal size of the monitor.
5. distance in between pixels; in a color monitor, the distance between two pixels of the same color
6. False, each pixel consists of red, green, and blue dots.
7. 8x8
8. 40 × 25 = 1000 characters per screen

SECTION 16.2: TEXT MODE PROGRAMMING AND VIDEO RAM

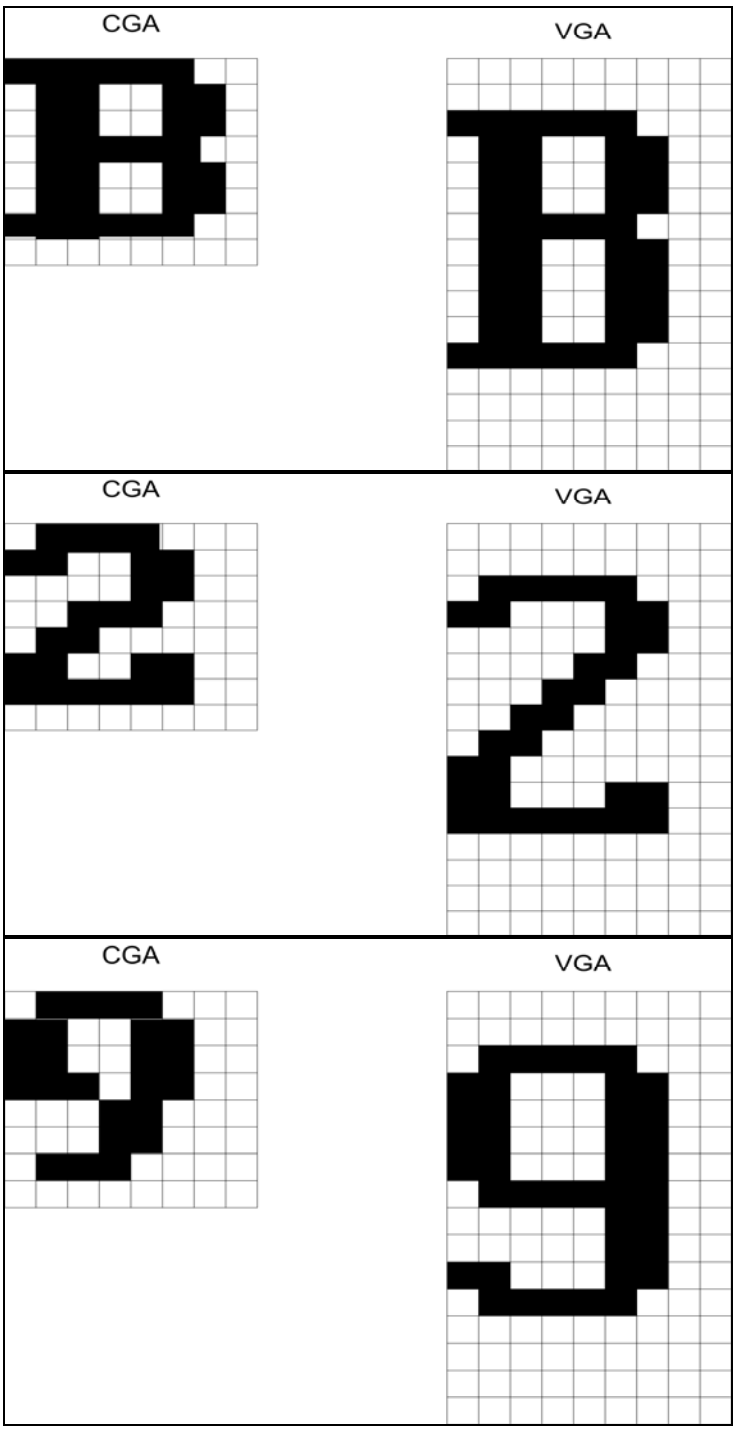
9. True
10. B0000H
11. This is similar to Example 16-7 (8-7) in the text with some modification.

```
MOV   AH,0
MOV   AL,03
INT   10H
MOV   AH,09
MOV   BH,0
MOV   AL,42H
MOV   CX,4
MOV   BL,41H
INT   10H
```

12. This is similar to Example 16-8 (8-8) in the text with some modification.

```
MOV   AH,0
MOV   AL,07
INT   10H
MOV   AH,09
MOV   BH,0
MOV   AL,42H
MOV   CX,50H
MOV   BL,0F0H
INT   10H
```

13. See the drawings below.



SECTION 16.3: GRAPHICS AND GRAPHICS PROGRAMMING

14. True
15. True
16. $2^{16} = 65,536$
17. since 620 x 200 = 124,000 bits = 15.13Kbytes and CGA can have a maximum of 16K bytes of video memory

18. It is divided into four planes of 64K bytes, where each plane is fit into the A0000 - AFFFF address space. When a bit in the address range is accessed one bit from each plane is provided.
19. 24 bits
20. (a) 640 x 480 x 4 = 150K, but due to the bit plane it uses 256K
640 x 480 x 8 = 300K, but due to the bit plane it uses 512K
640 x 480 x 16 = 600K, but due to the bit plane it uses 1M
640 x 480 x 24 = 900K, but due to the bit plane it uses 1.5M
(b) 1024 x 768 x 4 = 384K, but due to the bit plane it uses 512K
1024 x 768 x 8 = 768K, but due to the bit plane it uses 1M
1024 x 768 x 16 = 1536K = 1.5M
1024 x 768 x 24 = 2304K, but due to the bit plane it uses 2.5M
(c) 1600 x 1200 x 8 = 1.83M, but due to the bit plane it uses 2M
1600 x 1200 x 16 = 3.66M, but due to the bit plane it uses 4M

21.
```
        MOV   AX,0600H          ;clear the screen
        MOV   BH,07
        MOV   CX,0
        MOV   DX,184F
        INT   10H
        MOV   AH,00             ;set the mode to 06 (CGA high resolution)
        MOV   AL,06
        INT   10H
        ;draw line 1
        MOV   CX,213            ;col pixel=213
        MOV   DX,0              ;row pixel=0
LINE1:  MOV   AH,0C             ;0CH option to write a pixel
        MOV   AL,01             ;turn on the pixel
        INT   10H
        INC   DX                ;increment vertical position
        CMP   DX,200            ;check for the last position
        JNZ   LINE1             ;if not, continue
        ;draw line 2
        MOV   CX,426            ;col pixel=426
        MOV   DX,0              ;row pixel=0
LINE2:  MOV   AH,0C             ;0CH option to write a pixel
        MOV   AL,01             ;turn on the pixel
        INT   10H
        INC   DX                ;increment vertical position
        CMP   DX,200            ;check for the last position
        JNZ   LINE2             ;if not, continue
```

CHAPTER 17: SERIAL PORT PROGRAMMING WITH ASSEMBLY AND C#

SECTION 17.1: BASICS OF SERIAL COMMUNICATION

1. parallel data transfer
2. false
3. The even parity bit for ASCII 'Z' (01011010) is 0. Therefore, we have the following framing: 1 0 01011010 0 from left to right: 1 stop bit, even parity, 8 data bits, start bit
4. mark
5. When there is no data transfer and the line is low, it is called space.
6. With 2 stop bits,1 parity bit, and the stop bit, this makes a total of 4 bits for the overhead. Therefore, it has 4/6 = 66% overhead.
7. false
8. to convert from non-TTL voltage levels of RS232 to TTL level voltage used by UART chips
9. true
10. Since only 9 pins are of critical importance and it is much more economical in terms of space usage on the back plane of the motherboard. In addition, it will not be confused with 25-DB pins of the parallel port.
11. false
12. DTE
13. DTE, DTE
14. TxD, RxD, DSR, DTR, CTS, RTS, CDC, RI, and GND
15. Since each character takes a total of 10 bits (8 bits, 1 stop bit, 1 start bit) and each page has 25 × 80 characters, then we have the total bits for 200 pages = 200 x 25 x 80 x 10 =4,000,000

SECTION 17.2: PROGRAMMING x86 PC COM PORTS USING ASSEMBLY AND C#

16. a maximum of 4
17. 115,200
18.

```
        MOV     AH,0            ;option 0 of INT 14H
        MOV     DX,1            ;COM2
        MOV     AL,11100011     ;9600,no parity,1 stop bit,8-bit data
        INT     14H
```

19.

```
TITLE PROBLEM 19 SERIAL DATA COMMUNICATION BETWEEN TWO PCs
        .MODEL SMALL
        .STACK
        .DATA
MESSAGE  DB 'SENDING DATA VIA COM2, 4800,No P,1 Stop.',0AH,0DH
        DB ' ANY KEY PRESS IS SENT TO THE OTHER PC.',0AH,0DH
        DB ' PRESS ESC TO EXIT','$'

        .CODE
MAIN    PROC
        MOV AX,@DATA
        MOV DS,AX
        MOV AH,09
        MOV DX,OFFSET MESSAGE
        INT 21H

        ;initializing COM 2
        MOV AH,0                ;initialize COM port
        MOV DX,1                ;COM 2
        MOV AL,0C3H             ;4800 ,NO P,1 STOP,8 BIT DATA
        INT 14H
```

```
        ;checking key press and sending key to COM2 to be transferred
AGAIN:  MOV AH,01               ;check key press
        INT 16H
        JZ  NEXT                ;if no key go check COM port
        MOV AH,0                ;yes, there is a key press, get it
        INT 16H
        CMP AL,1BH              ;is it ESC?
        JE EXIT                 ;yes, exit
        MOV AH,1                ;no. send the char to COM 2 port
        MOV DX,01
        INT 14H
        ;check COM2 port to see there is char. if so get it and display it
NEXT:   MOV AH,03               ;get COM 2 status
        MOV DX,01
        INT 14H
        AND AH,01               ;AH has COM port status, mask all bits except D0
        CMP AH,01               ;check D0 to see if there is a char
        JNE AGAIN               ;no data, go to monitor keyboard
        MOV AH,02               ;yes, COM2 has data get it
        MOV DX,01
        INT 14H                 ;get it
        MOV DL,AL               ;and display it using INT 21H
        MOV AH,02               ;output char in DL reg
        INT 21H
        JMP AGAIN               ;keep monitoring keyboard
EXIT:   MOV AH,4CH              ;exit to DOS
        INT 21H                 ;
MAIN    ENDP
        END
```

CHAPTER 18: KEYBOARD AND PRINTER INTERFACING

SECTION 18.1: INTERFACING THE KEYBOARD TO THE CPU

1. 1s
2. (a)
3. (b)
4. (a) it can be any of keys in row 3 (3, 7, B, or F)
   (b) it can by any of keys in row 0 (0, 4, 8, or C)
5. (a) all rows are grounded at the same time
   (b) the columns are read
   (c) the columns data is checked to see if there is a zero in it; if there is a zero then it goes to the next step of key identification, otherwise it repeats step b and c
6. (a) ground one row at a time
   (b) read the columns
   (c) check for a zero in column data
   (d) if zero found, it goes to find out which column it belongs to and gets the scan code, otherwise it goes back to step a

7.

```
;the following look-up scan codes are in the data segment
KCOD_0          DB      0,1,2,3,4               ;key codes for row zero
KCOD_1          DB      5,6,7,8,9               ;key codes for row one
KCOD_2          DB      0AH,0BH,0CH,0DH,0EH     ;key codes for row two
KCOD_3          DB      0FH,10H,11H,12H,13H     ;key codes for row three

;the following is from the code segment
                PUSH    BX                      ;save BX
                SUB     AL,AL                   ;AL=0 to ground all rows rows at once
                OUT     PORT_A,AL               ;to ensure all keys are open (no contact)
K1:             IN      AL,PORT_B               ;read the columns
                AND     AL,00011111B            ;mask the unused bits (D7-D5)
                CMP     AL,00011111B            ;are all keys released
                JNE     K1                      ;keep checking for all keys released
                CALL    DELAY                   ;wait for 20 ms
K2:             IN      AL,PORT_B               ;read columns
                AND     AL,00011111B            ;mask D7-D5
                CMP     AL,00011111B            ;see if any key pressed?
                JE      K2                      ;if none keep checking
                CALL    DELAY                   ;wait 20 ms for debounce
;after the debounce see if still pressed
                IN      AL,PORT_B               ;read columns
                AND     AL,00011111B            ;mask D7-D5
                CMP     AL,00011111B            ;see if any key closed?
                JE      K2                      ;if none keep polling
;now ground one row at a time and read columns to find the key
                MOV     AL,11111110B            ;ground row 0 (D0=0)
                OUT     PORT_A,AL
                IN      AL,PORT_B               ;read all columns
                AND     AL,00011111B            ;mask unused bits (D7-D5)
                CMP     AL,00011111B            ;see which column
                JE      RO_1                    ;if none go to grounding row 1
                MOV     BX,OFFSET KCOD_0        ;set BX=start of table for column 0 keys
                JMP     FIND_IT                 ;identify the key
RO_1:           MOV     AL,11111101B            ;ground row 1 (D1=0)
                OUT     PORT_A,AL
                IN      AL,PORT_B               ;read all columns
                AND     AL,00011111B            ;mask unused bits (D7-D5)
                CMP     AL,00011111B            ;see which column
                JE      RO_2                    ;if none go to grounding row 2
                MOV     BX,OFFSET KCOD_1        ;set BX=Start of table for column 1 keys
                JMP     FIND_IT                 ;identify the key
RO_2:           MOV     AL,11111011B            ;ground row 2 (D2=0)
                OUT     PORT_A,AL
                IN      AL,PORT_B               ;read all columns
                AND     AL,00011111B            ;mask unused bits (D7-D5)
                CMP     AL,00011111B            ;see which column
```

```
JE          RO_3                    ;if none go to grounding row 3
```

```
                MOV     BX,OFFSET KCOD_2        ;set BX=start of table for column 2 keys
                JMP     FIND_IT                 ;identify the key
RO_3:           MOV     AL, 11110111B           ;ground row 3 (D3=0)
                OUT     PORT_A,AL
                IN      AL,PORT_B               ;read all columns
                AND     AL,00011111B            ;mask unused bits (D7-D5)
                CMP     AL,00011111B            ;see which column
                JE      K2                      ;if none then false input repeat the process
                MOV     BX,OFFSET KCOD_3        ;set BX=start of table for column 3 keys
;A key press has been detected and the row identified.  Now find which key.
FIND_IT:        RCR     AL,1                    ;rotate the column input to search for 0
                JNC     MATCH                   ;if zero, go get the code
                INC     BX                      ;if not point at the next code
                JMP     FIND_IT                 ;and keep searching
;GET THE CODE FOR THE KEY PRESSED AND RETURN
MATCH:          MOV     AL,[BX]                 ;get the code pointed by BX
                POP     BX                      ;return with AL=code for pressed key
                RET
```

8.

```
;the following look-up scan codes are in the data segment
KCOD_0          DB      0,1,2,3,4,5                     ;key codes for row zero
KCOD_1          DB      6,7,8,9,0AH,0BH                 ;key codes for row one
KCOD_2          DB      0CH,0DH,0EH,0FH,10H,11H         ;key codes for row two
KCOD_3          DB      12H,13H,14H,15H,16H,17H         ;key codes for row three
KCOD_4          DB      18H,19H,1AH,1BH,1CH,1DH         ;key codes for row four
KCOD_5          DB      1EH,1FH,20H,21H,22H,23H         ;key codes for row five

;the following is from the code segment
                PUSH    BX                      ;save BX
                SUB     AL,AL                   ;AL=0 to ground all rows at once
                OUT     PORT_A,AL               ;to ensure all keys are open (no contact)
K1:             IN      AL,PORT_B               ;read the columns
                AND     AL,00111111B            ;mask the unused bits (D7-D6)
                CMP     AL,00111111B            ;are all keys released
                JNE     K1                      ;keep checking for all keys released
                CALL    DELAY                   ;wait for 20 ms
K2:             IN      AL,PORT_B               ;read columns
                AND     AL,00111111B            ;mask D7-D6
                CMP     AL,00111111B            ;see if any key pressed?
                JE      K2                      ;if none keep checking
                CALL    DELAY                   ;wait 20 ms for debounce
;after the debounce see if still pressed
                IN      AL,PORT_B               ;read columns
                AND     AL,00111111B            ;mask D7-D6
                CMP     AL,00111111B            ;see if any key closed?
                JE      K2                      ;if none keep polling
;now ground one row at a time and read columns to find the key
                MOV     AL,11111110B            ;ground row 0 (D0=0)
                OUT     PORT_A,AL
                IN      AL,PORT_B               ;read all columns
                AND     AL,00111111B            ;mask unused bits (D7-D6)
                CMP     AL,00111111B            ;see which column
                JE      RO_1                    ;if none go to grounding row 1
                MOV     BX,OFFSET KCOD_0        ;set BX=start of table for column 0 keys
                JMP     FIND_IT                 ;identify the key
RO_1:           MOV     AL,11111101B            ;ground row 1 (D1=0)
                OUT     PORT_A,AL
                IN      AL,PORT_B               ;read all columns
                AND     AL,00111111B            ;mask unused bits (D7-D6)
                CMP     AL,00111111B            ;see which column
                JE      RO_2                    ;if none go to grounding row 2
                MOV     BX,OFFSET KCOD_1         ;set BX=Start of table for column 1 keys
                JMP     FIND_IT                 ;identify the key
RO_2:           MOV     AL,11111011B            ;ground row 2 (D2=0)
                OUT     PORT_A,AL
                IN      AL,PORT_B               ;read all columns
                AND     AL,00111111B            ;mask unused bits (D7-D6)
                CMP     AL,00111111B            ;see which column
                JE      RO_3                    ;if none go to grounding row 3
                MOV     BX,OFFSET KCOD_2        ;set BX=start of table for column 2 keys
                JMP     FIND_IT                 ;identify the key
```

```
RO_3:       MOV     AL,11110111B        ;ground row 3 (D3=0)
            OUT     PORT_A,AL
            IN      AL,PORT_B           ;read all columns
            AND     AL,00111111B        ;mask unused bits (D7-D6)
            CMP     AL,00111111B        ;see which column
            JE      RO_4                ;if none then go to grounding row 4
            MOV     BX,OFFSET KCOD_3    ;set BX=start of table for column 3 keys
            JMP     FIND_IT
RO_4:       MOV     AL,11101111B        ;ground row 4 (D4=0)
            OUT     PORT_A,AL
            IN      AL,PORT_B           ;read all columns
            AND     AL,00111111B        ;mask unused bits (D7-D6)
            CMP     AL,00111111B        ;see which column
            JE      RO_5                ;if none then go to grounding row 5
            MOV     BX,OFFSET KCOD_4    ;set BX=start of table for column 5 keys
            JMP     FIND_IT
RO_5:       MOV     AL,11011111B        ;ground row 5 (D5=0)
            OUT     PORT_A,AL
            IN      AL,PORT_B           ;read all columns
            AND     AL,00111111B        ;mask unused bits (D7-D6)
            CMP     AL,00111111B        ;see which column
            JE      K2                  ;if none then false input repeat the process
            MOV     BX,OFFSET KCOD_5    ;set BX=start of table for column 3 keys
;A key press has been detected and the row identified.  Now find which key.
FIND_IT:    RCR     AL,1                ;rotate the column input to search for 0
            JNC     MATCH               ;if zero, go get the code
            INC     BX                  ;if not point at the next code
            JMP     FIND_IT             ;and keep searching
;GET THE CODE FOR THE KEY PRESSED AND RETURN
MATCH:      MOV     AL,[BX]             ;get the code pointed by BX
            POP     BX                  ;return with AL=code for pressed key
            RET
```

9. The main advantage is that it does not tie down the microprocessor. The disadvantage is that it is not versatile.
10. To use a separate microprocessor (microcontroller) from the main CPU to scan the keys (polling) and use interrupts to inform the main CPU. This allows programming the keyboard itself which is much more versatile.

SECTION 18.2: PC KEYBOARD INTERFACING AND PROGRAMMING

11. 11 bits, 8 bits for scan code, 1 stop bit, 1 start bit, and odd parity bit
12. (a) B4H (b) 9AH (c) DFH
13. c and e are make, and the rest are break
14. The keyboard's shift status byte allows the distinction. f the shift bit is 0, it is "5". If the shift bit is 1, it is "%".
15. (a) 69H (b) 57H (c) 08H (d) 2FH (e) 49H (f) 40H
16. AH = 12H
17.

```
            EXTRN       MUSIC: NEAR
DTSEG       SEGMENT
MESS1       DB          CR,LF,'I will play a song for you if '
            DB          'you guess the key I am thinking of. '
            DB          'The key is one of the ALT Fs','$'
MESS2       DB          CR,LF,'Try again','$'
CR          EQU         0DH
LF          EQU         0AH
DTSEG       ENDS
;-----------------------------------------------
```

```
CDSEG       SEGMENT
MAIN        PROC  FAR
            ASSUME  CS:CDSEG,DS:DTSEG,SS:STSEG
            MOV     AX,DTSEG
            MOV     DS,AX
            MOV     AH,09               ;prompt user
            MOV     DX,OFFSET MESS1
            INT     21H
AGAIN:      MOV     AH,0                ;get scan code of
            INT     16H                 ;key pressed
            CMP     AX,7000H            ;is it Alt-F9 ?
            JZ      PLAY                ;if yes, play music
            CMP     AX,011BH            ;is it the Esc key?
            JZ      EXIT                ;if yes, exit to DOS
            MOV     AH,09               ;otherwise, display message
            MOV     DX,OFFSET MESS2
            INT     21H
            JMP     AGAIN               ;try again
PLAY:       CALL    MUSIC
EXIT:       MOV     AH,4CH
            INT     21H
MAIN        ENDP
CDSEG       ENDS
            END     MAIN
```

18. IRQ1
19. true
20. false
21. 00
22. the motherboard circuitry
23. Alt, Ctrl, Shift, NumLock, and CapsLock
24. (a) Home and (c) arrow
25. 0110 0000 or 60H
26. Right Shift is pressed and Insert is toggled.
27. (a) 00417H (b) 0041EH to 0043DH
    (c) 0041CH and 0041DH (d) 0041AH and 41BH
28. The keyboard buffer is empty.
29. full
30. the capacitive

SECTION 18.3: PRINTER AND PRINTER INTERFACING IN THE IBM PC

31. data lines, status signals, control signals, and ground signals
32. (a) BUSY (in) (b) STROBE (out) (c) ACKNLDG (in) (d) SLCT (in)
    (e) INIT (out) (f) PE (in)
    a, c, d, and f are all status signals while b and e are control signals.
33. PE
34. true
35. to indicate to the PC that the printer cannot accept new data
36. low
37. After the rising edge of ACKNLDG, it indicates that the printer is ready to accept another byte.
38. true, depending on the decoding circuitry

CHAPTER 19: HARD DISKS

SECTION 19.1: HARD DISK ORGANIZATION AND PERFORMANCE

1. because the microprocessor looks for code first from RAM and ROM before it looks for it in the disk
2. boot record
3. FAT
4. For MS DOS, they are IO.SYS, MSDOS.SYS, and COMMAND.COM. For IBM PC DOS, they are IBMBIO.COM, IBMDOS.COM, and COMMAND.COM.
5. They are hidden files
6. True
7. The tracks of the same radius on all the platters is called a cylinder. In the hard disk, when the heads move they are all moving at the same time on the same cylinder, allowing access to all the sectors under each head.
8. the number of heads times the number of cylinders
9. Using Example 19-1, calculate for your hard drive.
10. true
11. The amount of time that it takes to find the desired cylinder (track) is called seek time. The amount of time that it takes for the head to settle (stop vibrating) when the desired track is found is called settling time. With the advent of the voice coil it is almost zero. When the desired track is found, the amount of time it takes to locate the desired sector is called rotational latency. It is the average of 0 and the time for one rotation. The access time
    is the total sum of all these factors.
12. (c) and (a)
13. False, it is used for any kind of peripheral device such as printer and CD-ROM.
14. 7 (or 8 if we include the SCSI adapter itself)
15. (a) 1 (b) 3 (c) 5
16. *Mean time between failure* is the measure of how long the disk will last without a malfunction. It is a measure of reliability.

CHAPTER 20: THE IEEE FLOATING POINT AND x87 MATH PROCESSORS

SECTION 20.1: MATH COPROCESSOR AND IEEE FLOATING-POINT STANDARDS

1. It takes too long to execute; in addition, writing software for math functions using integer instruction is very tedious.
2. 4
3. 8
4. Bit 31 is for the sign, bits 30 - 23 are for biased exponents, and bits 22 - 0 are for the significand.
5. (a) 15.575 = 41793400H (b) 89.125 = 42B24000H
   (c) -1022.543 = C47FAB00H (d) -0.00075 = BA449000H
6. 98.23 = 42C475C3H
7. The IEEE single precision and IEEE double precision standards are called short real and long real, respectively, in Intel literature.
8. Bit 63 is for the sign, bits 62 - 52 are for biased exponents, and bits 51 - 0 are for the significand.
9. 7FH, exponent portion
10. 3FF, exponent portion
11. (a) 12.9823 = 4029F6F000000000H (b) 98.76123 = 4058B0B700000000H
12. 63 bits
13. bit 79
14. 73 bits (bits 0 to 72)
15. 10 bytes

SECTION 20.2: x87 INSTRUCTIONS AND PROGRAMMING

16. (a) DD (b) DQ (c) DT
17.

```
0000                                DTSEG   SEGMENT PARA 'Data'
0000  41793333                     Q5A     DD   15.575
0004  42B24000                     Q5B     DD   89.125
0008  C47FA2C1                     Q5C     DD   -1022.543
000C  BA449BA6                     Q5D     DD   -0.00075
0010  BBB88D06F0F62940             Q11A    DQ   12.9823
0018  FBAE08FEB7B05840             Q11B    DQ   98.76123
0020                                DTSEG   ENDS
```

18.

```
.8087
; in the data seg
X       DD      3.12            ;stored as 4047AE14H
Y       DD      5.43            ;stored as 40ADC28FH
Z       DD      ?
;in code segment

        FINIT
        FLD     X               ;load X
        FMUL    ST(0),ST(0)     ;X squared
        FLD     Y               ;load Y
        FMUL    ST(0),ST(0)     ;Y squared
        FLD     Y               ;load Y again
        FMUL    ST(0),ST(1)     ;Y cubed
        FADD    ST(0),ST(2)     ;sum X squared and Y cubed
        FSQRT                   ;calculate square root
        FST     Z               ;and store it
```

;z = 415083C2H = 13.032168

19.
```
.8087
;in the data segment
X       DD      1.25            ;stored as 3FA00000H
Y       DD      ?
C1      DD      12.34           ;stored as 414570A4H
C2      DD      5.0             ;stored as 40A00000H
;in the code segment
        FINIT
        FLD     X               ;load X
        FMUL    ST(0),ST(0)     ;square X (1.5625)
        FADD    ST(0),ST(0)     ;double the square, now ST(0)=2X**2   (3.125)
        FLD     X               ;load X again
        FLD     C2              ;load the constant 5
        FMUL    ST(0),ST(1)     ;now ST(0)=5X and ST(2)=2X**2
        FADD    ST(0),ST(2)     ;ST(0) =2X**2+5X  (3.125 + 6.25 = 9.375)
        FLD     C1              ;load the constant 12.34
        FADD    ST(0),ST(1)     ;make the sum Y   (12.34 + 9.375 = 21.715)
        FST     Y               ;and store it
;y = 41ADB852H = 21.715
```

20.
```
.8087
;in the data segment
R       DD      25.5            ;stored as 41CC0000H
AREA    DD      ?
;in the code segment
        FINIT
        FLD     R
        FMUL    ST(0),ST(0)     ;R squared = 650.25
        FLDPI
        FMUL    ST(0),ST(1)
        FST     AREA

;AREA = 44FF5A43H =2042.8206787109375
```

21.
```
.8087
;in the data segment
R       DD      25.5
Z       DD      ?
C1      DD      3.0
C2      DD      4.0
;in the code segment
        FINIT
        FLD     R
        FMUL    ST(0),ST(0)     ;R**2
        FLD     R
        FMUL    ST(0),ST(1)     ;ST(0)=R**3
        FLD     C1              ;load 3
        FMUL    ST(0),ST(1)     ;now ST(0)=3 times R**3
        FLDPI
        FMUL    ST(0),ST(1)     ;now ST(0)=3 * pi * R**3
        FLD     C2              ;load 4
        FXCH
        FDIV    ST(0),ST(1)     ;divide by 4
        FST     Z               ;and save it
;Z = 47189CF2H = 39068.94441481
```

22.

```
.8087
PAGE 60,132
;program to calculate SIN of a 45-degree angle
STACKSG         SEGMENT
        DW      32 DUP (?)
STACKSG         ENDS
;----------------------------
DATASG  SEGMENT
        ORG 00H
ANGLE   DD      0.785                   ;angle in radian for 45 degrees (3F48F5C3)
        ORG 10H
X       DD      0
        ORG 20H
Y       DD      0
        ORG 30H
R       DD      0
        ORG 40H
SIN     DD      0
        ORG 50H
DATASG  ENDS
;----------------------------
CODESG          SEGMENT
START   PROC    FAR
        ASSUME CS:CODESG,DS:DATASG,SS:STACKSG
        MOV     AX,DATASG
        MOV     DS,AX
        CALL    CALC_X_Y
        CALL    CALC_R
        CALL    CALC_SIN
        MOV     AH,4CH
        INT     21H
START   ENDP
;----------------------------
;procedure to calculate X and Y given an angle
CALC_X_Y        PROC  NEAR
        FINIT                           ;initialize 8087
        FLD     ANGLE                   ;load ANGLE onto stack
        FPTAN                           ;calculate X and Y
        FSTP    X                       ;store X and POP
        FSTP    Y                       ;store Y and POP
        RET
CALC_X_Y        ENDP
;-------------------------------
;procedure to calculate hypotenuse given X and Y
CALC_R          PROC  NEAR
        FINIT                           ;initialize 8087
        FLD     X                       ;load X onto stack
        FMUL    ST(0),ST(0)             ;square X
        FLD     Y                       ;load Y onto stack
        FMUL    ST(0),ST(0)             ;square Y
        FADD    ST(0),ST(1)             ;calculate X**2 + Y**2
        FSQRT                           ;take square root
        FST     R                       ;store R
        RET
        CALC_R                          ENDP
;-------------------------------
;procedure to calculate SIN, given R and X
CALC_SIN        PROC  NEAR
        FINIT                           ;initialize 8087
        FLD     R                       ;load R onto stack
        FLD     Y                       ;load Y onto stack
        FDIV  ST(0),ST(1)               ;SIN = Y/R
        FST   SIN                       ;store SIN
        RET
CALC_SIN        ENDP
;-------------------------------
CODESG          ENDS
        END     START
```

SECTION 20.3: x87 INSTRUCTIONS

23. c, d, and e
24. radians
25.

```
.387
;in the data segment
ANG         DD      0.523598776         ;angle in radians for 30 degrees
SINEVAL     DD      ?
COSVAL      DD      ?
;in the code segment
            FLD     ANG
            FSINCOS                     ;now ST(0)= sine and ST(1) =cosine
            FSTP    SINEVAL             ;store the sine and pop
            FSTP    COSVAL              ;store the cosine and pop.
```

CHAPTER 21: 386 MICROPROCESSOR: REAL vs. PROTECTED MODE

SECTION 21.1: 80386 IN REAL MODE

1. the 80188/186 and higher
2. The PUSHA instruction pushes onto the stack the registers AX, CX, DX, BX, SP, BP, SI, and DI. It performs the actions of the following instructions: PUSH AX, PUSH CX, PUSH DX, PUSH BX, PUSH SP, PUSH BP, PUSH SI, and PUSH DI.
3. The POPA instruction pops the contents of the top 8 bytes of the stack into registers DI, SI, BP, SP, BX, DX, CX, and AX. Although it pop off the stack the word belonging to SP, it does not store it into the SP register. This to ensure the integrity of stack frame. The POPA is equivalent to the following group of instructions: POP DI, POP SI, POP BP, POP SP, POP BX, POP DX, POP CX, and POP AX.
4. all 80188/186 and higher microprocessors of the 80x86 family
5. (a) AX=430H (b) BX=0 (c) CX= AA0AH (d) CX= A0AAH
6. true
7. 80286
8. 80286
9. a, b, d, and e
10. DS:3500=(43) DS:3501=(F5) DS:3502=(34) DS:3503=(98) All values are in hex.
11. ES:1000=(24) ES:1001=(B3) ES:1002=(07) ES:1003=(00) All values are in hex.
12. DS:EAX DS:EBX DS:ECX DS:EDX
    DS:ESI DS:EDI SS:EBP SS:ESP
13.

```
                .MODEL SMALL
                .386
                .STACK  300H
                .DATA
MYDATA  DD      10  DUP(9999H)          ;DATA ARRAY
                .CODE
                MOV     AX,@DATA
                MOV     DS,AX
                SUB     EBX,EBX
                MOV     CX,10           ;COUNTER
                MOV     EAX,100         ;THE FACTOR
BACK:           ADD     [MYDATA+EBX*4],EAX
                INC     EBX
                DEC     CX
                JNZ     BACK
                MOV     AH,4CH
                INT     21H
                END
```

14.

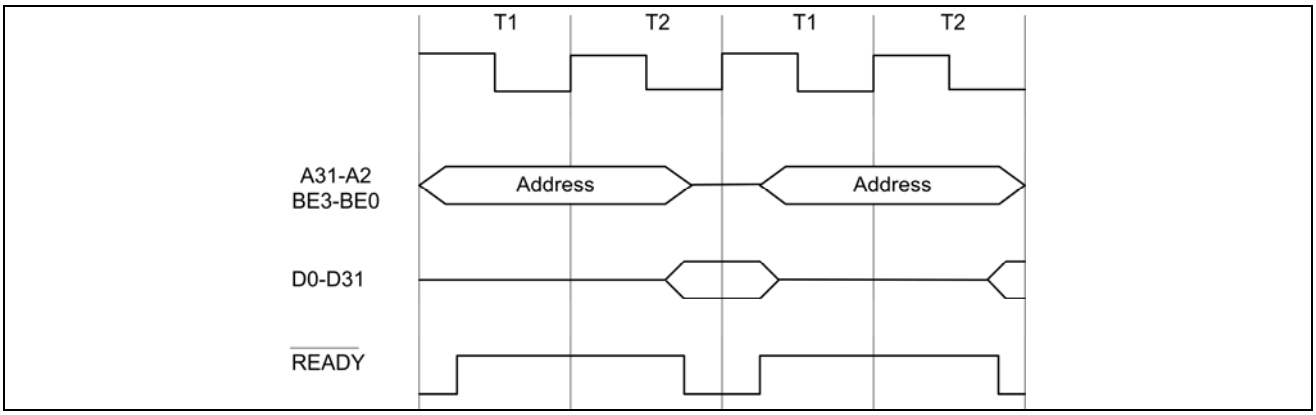
```
            .MODEL SMALL
            .386
            .STACK 300H
            .DATA
DATA1   DQ      548FB9963CE7H           ;FIRST OPERAND
DATA2   DQ      3FCD4FA23B8DH           ;SECOND OPERAND
SUM     DQ      ?
            .CODE
MAIN:   MOV     AX,@DATA
            MOV     DS,AX
            CLC
            SUB     EBX,EBX
            MOV     CX,2                    ;COUNTER
            MOV     EDX,OFFSET DATA1        ;LOAD ADDRESS
            MOV     ESI,OFFSET DATA2        ;LOAD ADDRESS
            MOV     EDI,OFFSET SUM          ;LOAD ADDRESS
BACK:   MOV     EAX,[EDX+EBX*4]         ;LOAD THE OPERAND
            ADC     EAX,[ESI+EBX*4]         ;ADD THE OPERAND
            MOV     [EDI+EBX*4],EAX         ;SAVE IT
            INC     EBX                     ;INCREMENT THE INDEX POINTER
            DEC     CX                      ;CONTINUE UNTIL COUNTER=0
            JNZ     BACK
            MOV     AH,4CH                  ;EXIT
            INT     21H                     ;TO DOS
            END     MAIN
```

15. CS:IP and CS:EIP for code, all the following for data
    DS:BX DS:SI DS:DI DS:EAX
    DS:EBX DS:ECX DS:EDX DS:ESI DS:EDI
    all the following for stack
    SS:BP SS:SP SS:EBP SS:ESP
16. (a) EBX= FFFFFFF4H (b) EDX= FFFFFFF8H
    (c) ECX= 7 (d) EBX= 99H
17. EAX= 0
    EBX= 19H (since scanning from left, D25 is the first high, 19H = 25 decimal)
18. AX= 03 DX= 7
19. In the case of MOVSX, the microprocessor sign extends (copies the sign bit to upper bits of the destination register) to prepare it for signed number arithmetic operations. MOVZX is used for unsigned arithmetic and logic instructions.
20. false

SECTION 21.2: 80386: A HARDWARE VIEW

21. low
22. false
23. D0 - D7 and D8 - D15
24. D16 - D23 and D24 - D31
25. D0 - D7 and D24 - D31
26. D8 - D15 and D16 - D23
27. 50 MHz
28. EIP = 0000FFF0H, CS = F000, A2 - A31 = all 1s, and BE0 - BE3 =all 0s.
29. The first opcode is fetched from physical address FFFFFFF0H. Therefore, ROM must be mapped into that address for the boot-up.
30. The address ranges of E0000 - FFFFF and E0000000 - FFFFFFFF have the same BIOS ROM. This allows BIOS to be accessed both in real mode and protected mode.

31.



32. Since the ECX source is 32-bit data and the destination address 2002 is not double word (32-bit) aligned, we need a total of 2 memory cycles to access them. Therefore, a total of 4 clocks (2 cycles × 2 clocks) is needed.
33. 0, 4, 8 or, C, as the least significant digits of the address (in hex)
34. 1/33 MHz = 30 ns; Since each memory cycle takes two clocks, we need a total of 60 ns for the memory cycle time.

SECTION 21.3: 80386 PROTECTED MODE

35. Virtual memory refers to memory of the hard disk. It is a way of fooling the CPU to think that it has access to an unlimited amount of RAM (main memory).
36. False. It asks from the main memory first. If it is not there, then the operating system will bring it from hard disk (virtual memory) into main memory. This is often done without any noticeable delay as far as the user is concerned.
37. DRAM and hard disk
38. In real mode, the maximum memory space is 1M but in protected mode the memory space is 4 gigabytes.
39. protected mode
40. false, segmentation only
41. true
42. false, a maximum of 64K bytes
43. true
44. 4K bytes
45. 8 bytes
46. In real mode the physical address must not exceed FFFFFH (if the operating system supports HMA, the maximum is 10FFEFH), but in protected mode the maximum address can be as high as FFFFFFFFH.
47. a total of 32 bits, the lower 24 bits (A0 - A23) are bytes 2, 3, 4, and the 8 bits of A24 - A31 are assigned to byte 7 (the last byte)
48. all 0s
49. 20 bits, bytes 0 and 1, in addition to lower nibble of byte 6
50. true
51. highest, lowest
52. To indicate if a given data or code is accessed (read). This allows the operating system to monitor if data has been used any time recently in order to make a decision to discard the unused data to make space for new data.
53. P lets the CPU know that the information it is requesting is present in main (DRAM) memory and so it can go ahead and fetch it. If P = 0 the CPU must wait until the operating system brings the information from the hard disk to main memory.

54. When D3 = 0 it is the data segment; if D1 = 0 it is read-only and D1 = 1 it is read/write data segment. When D3 = 1 it is the code segment (executable), then if D1 = 0 the code segment may not be read, but if D1 = 1 the code segment may be read.
    (a) this is an access byte for the data segment, present, accessed, privilege level = 00 (highest), and read only
    (b) this is an access byte for the data segment, present, accessed, privilege level = 11 (lowest), and read only
    (c) this is an access byte for the data segment, present, accessed, privilege level = 00 (highest), and read/write segment
    (d) this is an access byte for the code segment, present, accessed, privilege level = 11 (lowest), and can be read
    (e) this is an access byte for the code segment, present, accessed, privilege level = 00 (highest), and can be read
    (f) this is an access byte for the code segment, present, accessed, privilege level = 11 (lowest), and cannot be read
55. The translation look aside buffer is a buffer inside the CPU that keeps the top 32 most recently used page frame physical addresses. This helps the CPU to save time in translating from a linear address to physical address.
56. The physical address is an address in the range of 000000000 - FFFFFFFFH of the maximum 4 gigabytes address of the CPU and each location can be assigned to a physically existent memory such as ROM and RAM. In contrast, the linear address is an address used by the operating system to map the information for virtual memory.
57. linear to physical address
58. TLB
59. 32
60. False
61. In segmentation the memory is viewed in chunks of 1 byte to 4 gigabytes in size, but in paging it is viewed as a multiple of 4K bytes.
62. 2 levels, user and supervisor

CHAPTER 22: HIGH-SPEED MEMORY DESIGN AND CACHE

SECTION 22.1: MEMORY CYCLE TIME OF THE x86

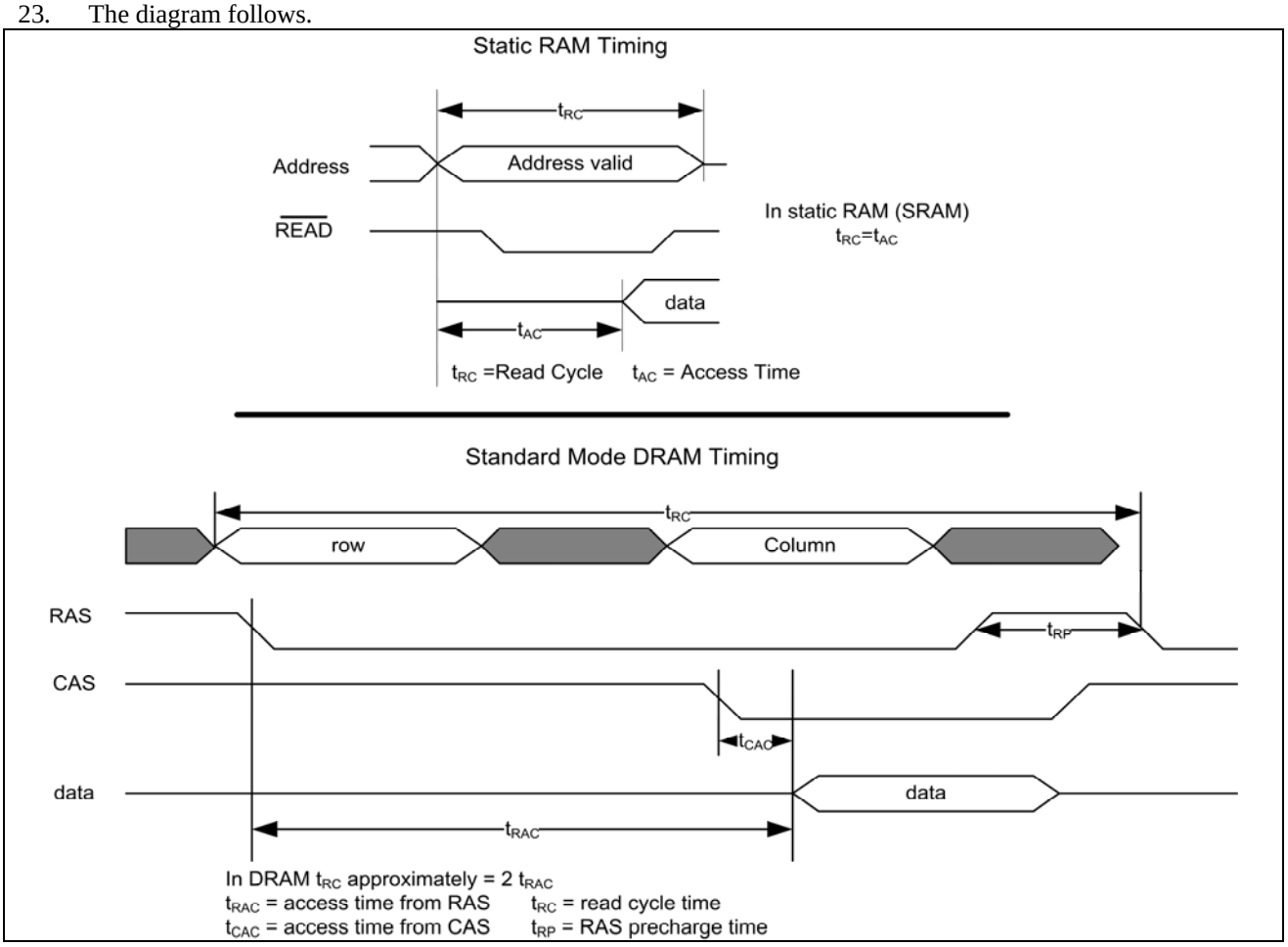
1. (a) (1 WS + 2) x 30 ns = 90 ns (b)(1WS + 2) x 20 ns = 60 ns
   (c) (2 WS + 2) x 40 ns = 120 ns (d) (1 WS + 2) x 16.66 ns = 50 ns
   (e) (0 WS + 2) x 15 ns = 30 ns
2. 1/33 MHz =30 ns for each clock period, 90 ns/30 ns = 3 clocks, therefore we have 1 WS
3. 1/50 MHz =20 ns for each clock period, 80 ns/20 ns = 4 clocks, therefore we have 2 WS
4. 1/66 MHz =15 ns for each clock period, 45 ns/15 ns = 3 clocks, therefore we have 1 WS
5. 1/20 MHz =50 ns for each clock period, 200 ns/50 ns = 4 clocks, therefore we have 2 WS
6. 1/50 MHz =20 ns and memory cycle time with 2 WS is (2WS + 2) x 20 =80 ns. This is the same as a 40 ns clock period system, or the same bus performance of a 25 MHz of 0 WS system. The bus performance degradation is 50%.
7. 1/33 MHz =30 ns and memory cycle time with 1 WS is (1WS + 2) × 30 =90 ns. Now this is the same as a 45 ns clock period system or the same bus performance of 22.2 MHz of 0 WS system. The bus performance degradation is 67%.
8. 1/60 MHz = 16.6 ns and memory cycle time with 1 WS is (1WS + 2) × 16.6 = 50 ns. This is the same as a 25 ns clock period system, or the same bus performance of a 40 MHz of 0 WS system. The bus performance degradation is 66%.
9. 60ns/(1 WS + 2) =20 ns is the period for the system clock. Therefore, the CPU frequency is 50 MHz.
10. 1/33 MHz = 30 ns is the clock period and memory cycle time is 2 × 30 = 60 ns. Out of 60 ns, 20 ns is used for path delay; therefore, the memory must have no more than 40 ns cycle time.

SECTION 22.2: PAGE AND STATIC COLUMN DRAMS

11. ROM and SRAM
12. t_{AA} is the time interval between the moment the addresses are provided to the memory chip address pins to the moment the data is available at the data pins. t_{CA} is the time interval between the moment the CS (chip select) is activated to the moment the data is available at the data pins.
13. The minimum time interval between two consecutive accesses to the memory chip.
14. From the moment the CPU provides the address to the address pins to the moment it expects the data to be at its data pins. Or, one might say the minimum time interval between two consecutive memory access requests.

15. t_{RC} (read cycle time) is the memory cycle time while t_{RAC} (access time from RAS activation) is the memory access time from the moment the RAS is activated. t_{RC} is always much larger than t_{RAC} (almost twice).
16. $t_{RC} = t_{RAC} + t_{RP}$
17. 120 ns
18. 170 ns
19. 55 ns
20. 45 ns
21. It is 2048 × t_{RC}; therefore, 2048 × 110 ns = 225280 ns.
22. It is 2048 × t_{RC}; therefore, 2048 × 130 ns = 266240 ns.

23. The diagram follows.



24. Each 8-bit data bus has 1Mx8 or 1M; therefore, we have 4M bytes for each set and since in interleaved memory two sets are used, the minimum memory addition is 8M or 2[4(1Mx8 bits)] = 8M bytes, or one might say, 2 sets of 1M double words (each double word = 32 bits).
25. Since 8M bytes is 8Mx8, (8Mx8)/(1Mx4) = 16 chips.
26. 2[4(4Mx8 bits)] = 32M bytes, or one might say, 2 sets of 4M double words
27. Since 32M is 32Mx8, (32Mx8)/(4Mx4) = 16 chips.
28.

Set B

D31 D24	D23 D16	D15 D8	D7 D0
7	6	5	4
F	E	D	C
FFFF			

SetA

D31 D24	D23 D16	D15 D8	D7 D0
3	2	1	0
B	A	9	8
			FFF8

29. time to access one page = t_{RAC} + 2047 t_{PC} = 70 ns + 2047 x 45 ns = 70 ns + 92115 = 92185 ns

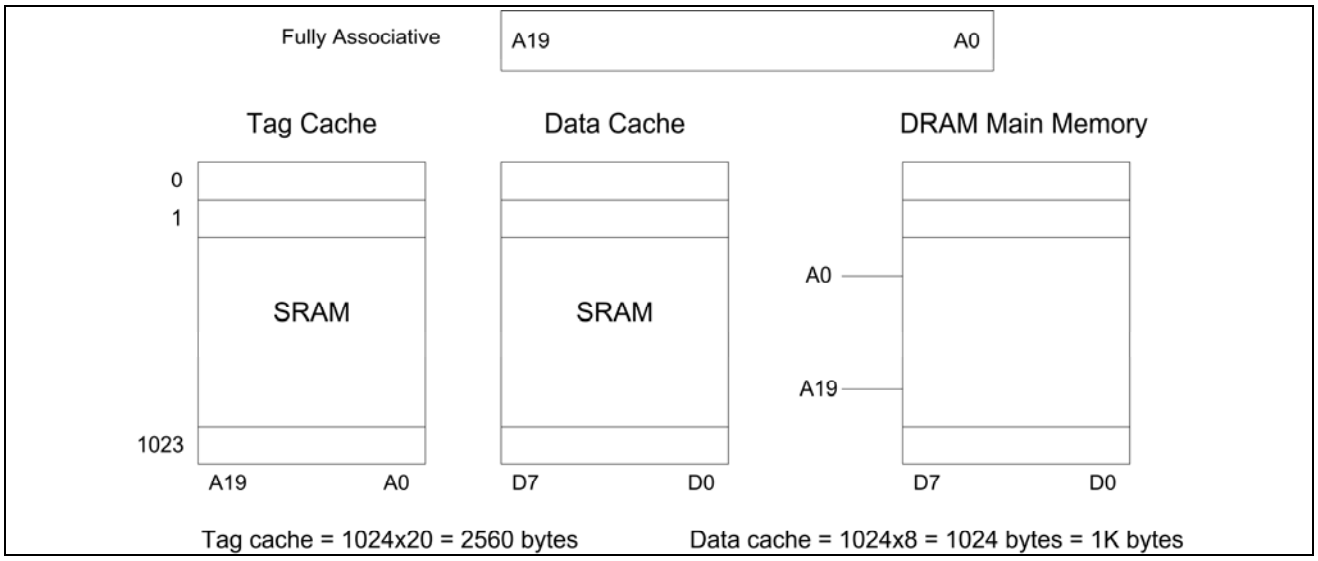
30. time to access one page = t_{RAC} + 2047 t_{PC} = 60 ns + 2047 x 40 ns = 60 ns + 81880 ns = 81940 ns
31. time to access all 2048 bits of one page = t_{RAC} + 2047 × t_{SC} = 70 ns + 2047 × 40 ns = 70 ns + 81880 ns = 81950ns
32. time to access one page = t_{RAC} + 2047 x t_{PC} = 60 ns + 2047 x 35 ns = 60 ns + 71645 ns = 71705 ns
33. false
34. true

SECTION 22.3: CACHE MEMORY

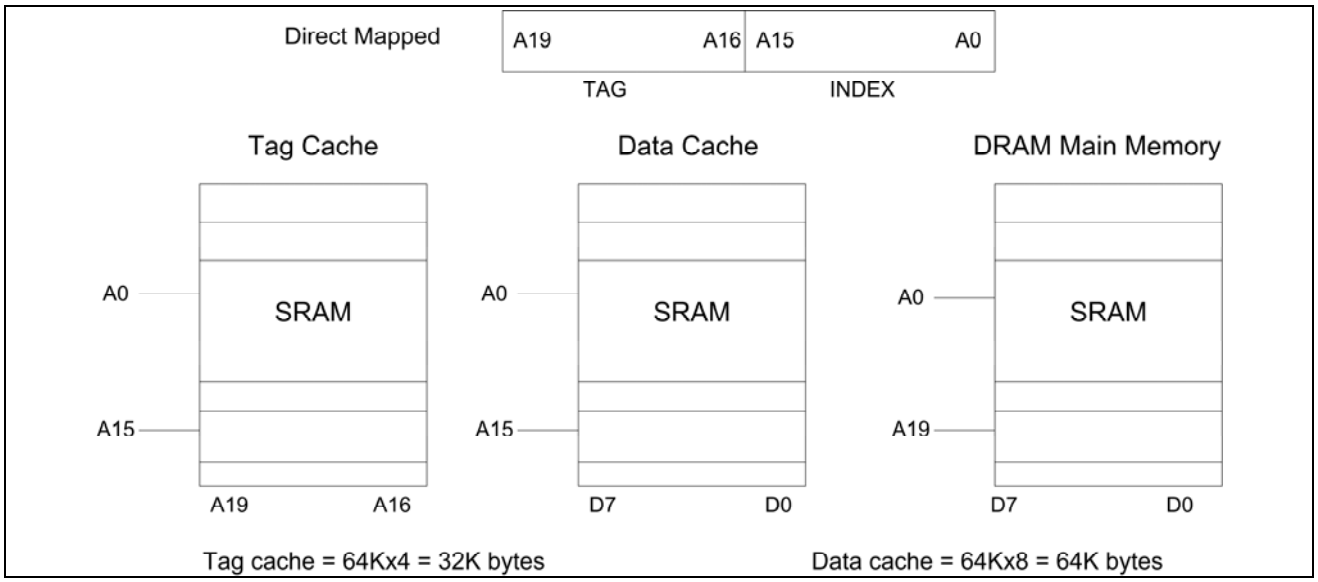
35. fully associative, direct mapped, and set associative
36. When the CPU starts to access memory locations, it is likely that subsequent memory accesses will be in the same vicinity, at least for a while.
37. *Least recently used* refers to those contents of cache that have not been accessed for a period of time and therefore they are the first candidates to be replaced in order to make room in cache for new data and code.

38. It refers to the method of updating main memory. When the CPU writes to memory, if it writes to cache and main memory both at the same time, it is called write through. If it writes to cache memory only and lets the cache controller write to the main memory at some convenient time later, it is called write back. Write through increases the main memory bus usage (traffic) but main memory is always updated. Since write-back has 30 - 40% better performance all recent CPUs use write-back instead of write-through.

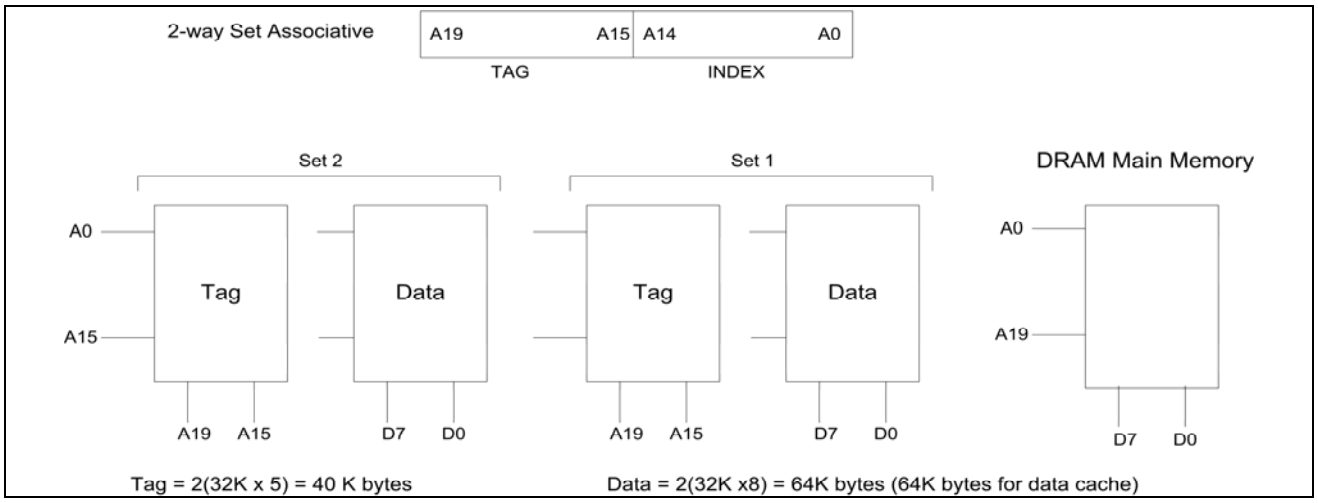
39. It means that when there is a cache miss, a minimum of 16 bytes of data is brought into cache (and into CPU) from the main memory.
40. (a) See the following diagram.



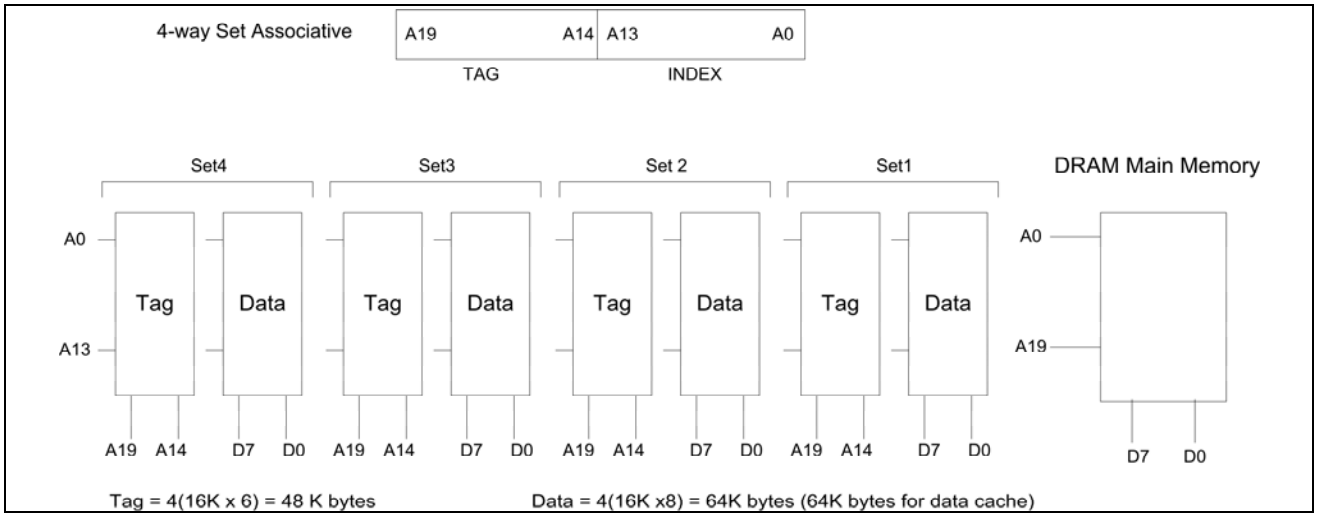
(b) See the following diagram.



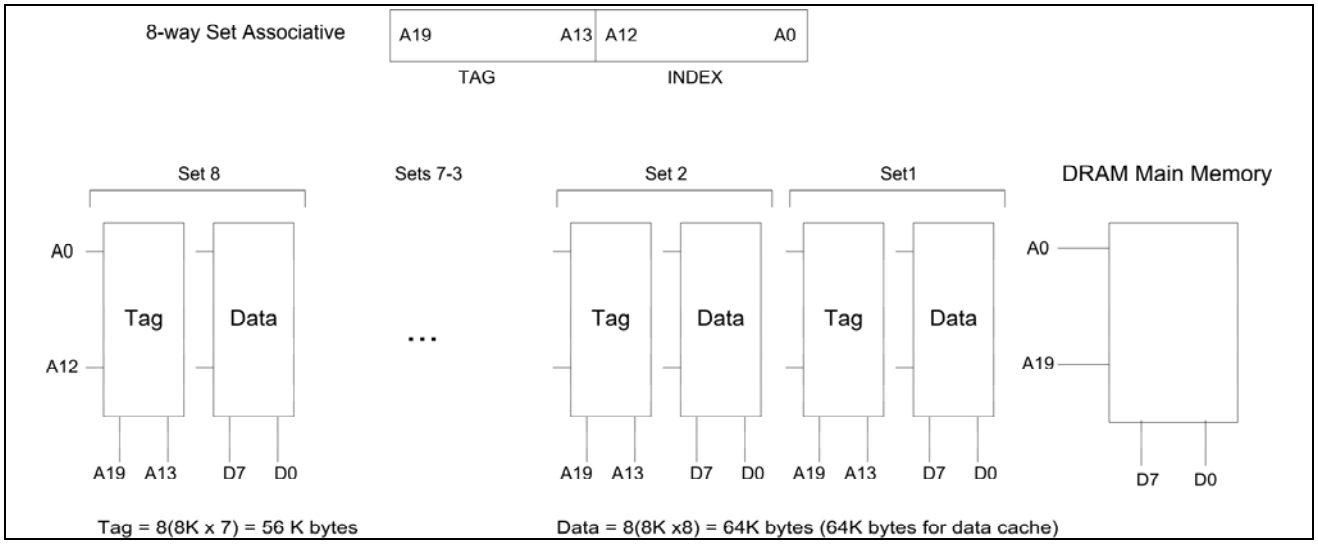
(c) See the following diagram.



(d) See the following diagram.

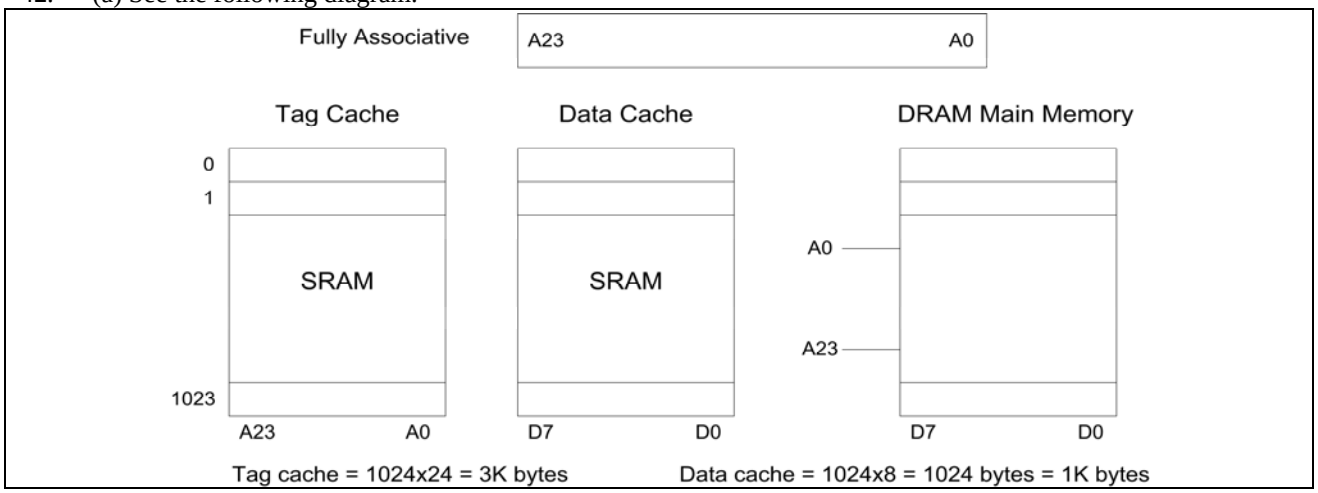


(e) See the following diagram.

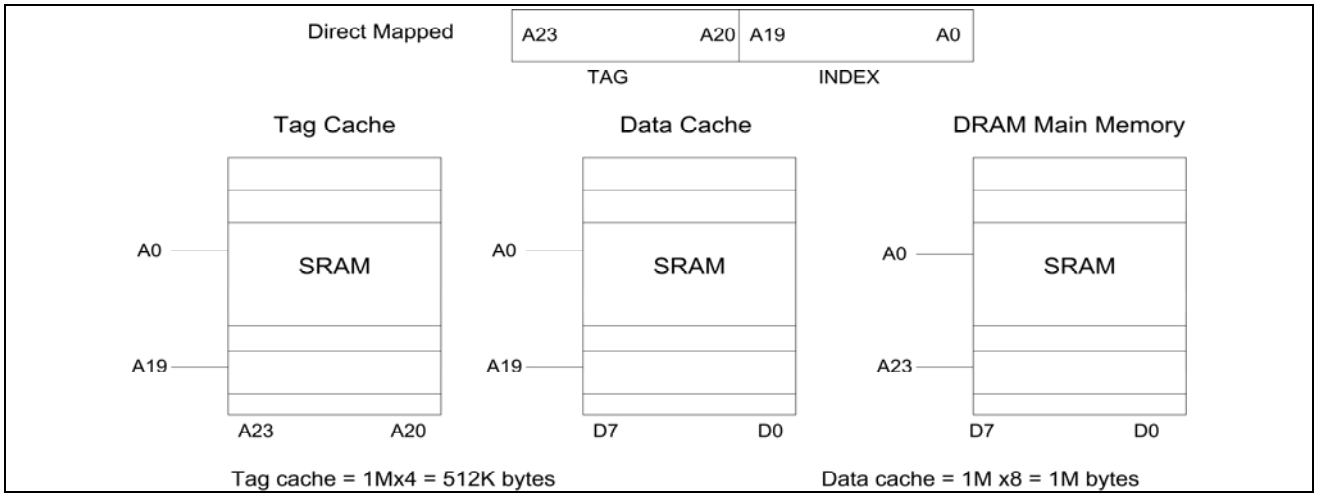


41. The data cache is the same for all of them but the tag cache goes from 32K in direct-mapped to 56K in 8-way set associative. This increase of 24K bytes (56 - 32 = 24) increases the hit rate substantially.

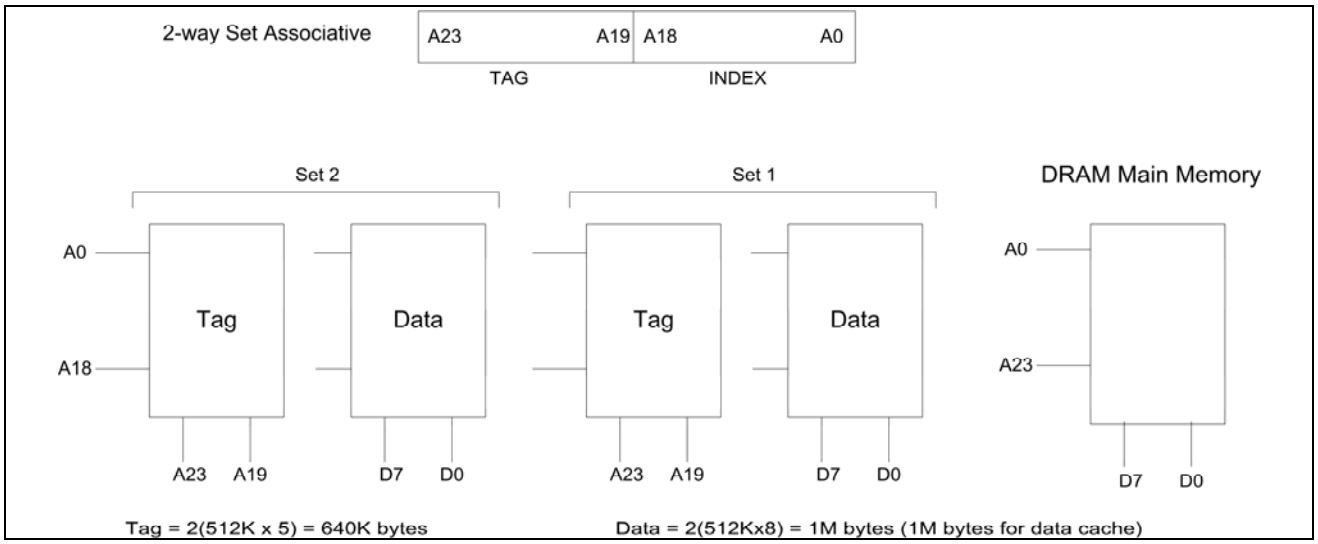
42. (a) See the following diagram.



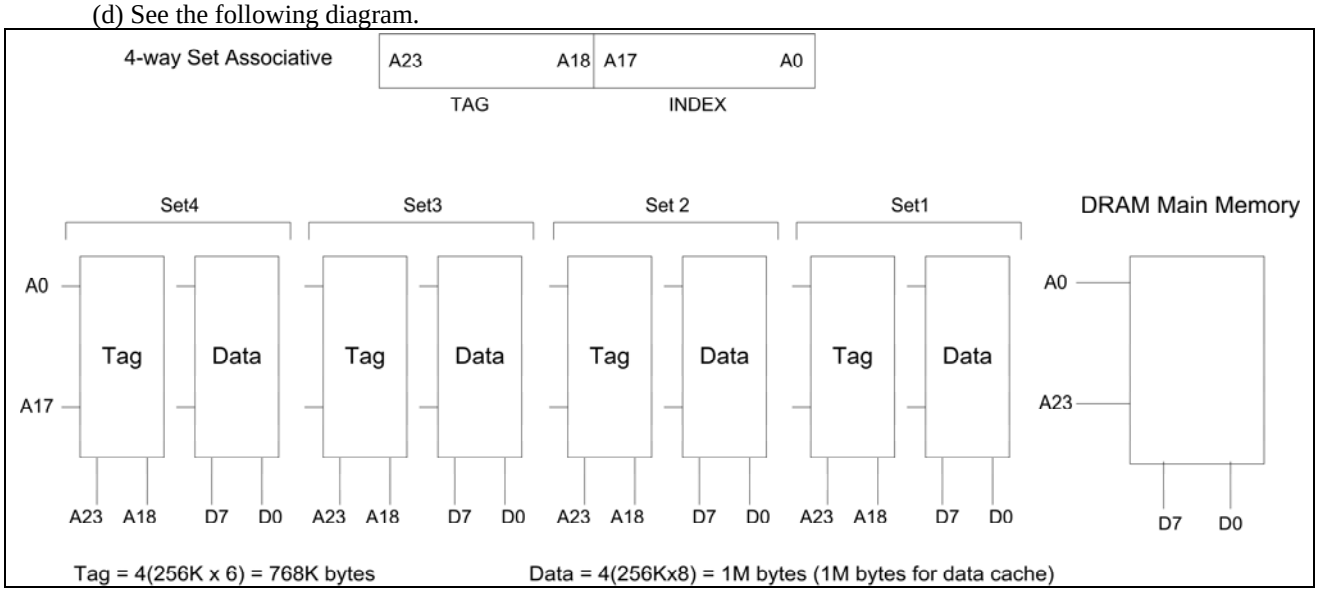
(b) See the following diagram.



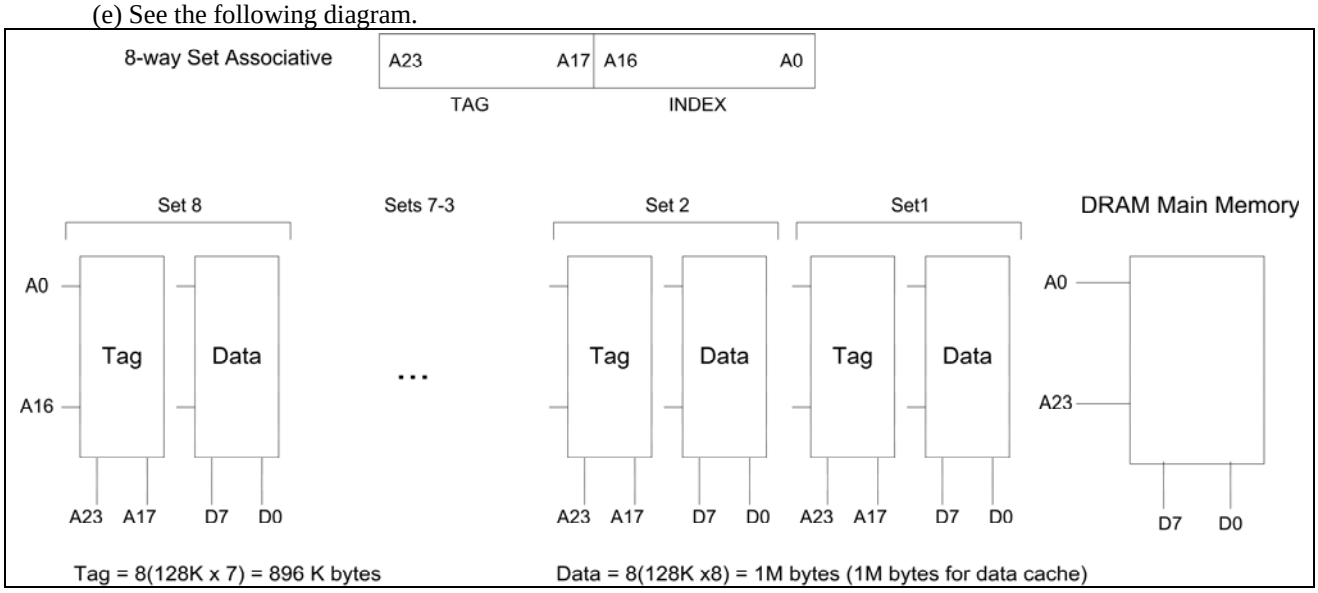
(c) See the following diagram.



(d) See the following diagram.



(e) See the following diagram.



43. The data cache is the same for all of them but the tag cache goes from 512K bytes in direct-mapped to 896K bytes in 8-way set associative. This increase of 384K bytes (896 - 512 = 384) increases the hit rate substantially.
44. cache organization (associativity), cache size, cache fill block size
45. by increasing the cache size, the effect on the cache hit is small
46. L1 works at the same speed as the CPU. L2 works at a fraction (1/3) of the frequency of the CPU.
47. true

SECTION 22.4: SDRAM, DDR RAM, AND RAMBUS MEMORIES

48. multiple
49. Since each clock has period of 1/300 MHz = 3.3 ns we have 2 x 3.3 ns = 6.6 ns for the read cycle time.
50. A memory cycle time of 6.6 ns means that the microprocessor expects data every 6.6 ns. Memory cycle time is measured from the time the CPU provides the address to the time it expects the requested data at its data pins. To design a motherboard in which the memory access time plus the data and address pathway delay is only 6.6 ns is not only technically challenging but also very expensive. It requires memory chips with an access time of 4 ns and logic gates (74XX244, 74XX138, 74XX245) of pico second speed. Furthermore, such high-speed signals going around the motherboard cause massive radiation and act like an antenna. As result, such a high-speed motherboard is not yet possible at an affordable price.
51. (b) a fraction of. This is the case now. However, Intel is planning to introduce 100 - 150 MHz bus speed in the next few years.
52. Extended Data Out, Fast Page Mode
53. True
54. It means that the row is selected and the data in consecutive columns of that row are fetched. In opening the page, we provide the row address and RAS is activated; then the column address is provided and CAS is activated. From then on subsequent data is fetched by simply toggling CAS and the access time is called t_{PC}, the page cycle time. t_{PC} is the minimum time for reading consecutive columns of a given row.
55. FPM
56. The internal sense amplifiers of DRAM shut down data flow to DRAM's data bus. This in turn will deprive the CPU of data being fetched from DRAM.
57. t_{PC} (page cycle time)
58. To allow the start of the next cycle. The faster it is done, the sooner it can start the next cycle.
59. From Table 22-11 (14-11) we have t_{PC} = 20 ns.
From Table 22-10 (14-10) and 22-11 (14-11) we see that t_{RAC} and t_{RC} are the same for both EDO and FPM DRAMs. However, the t_{PC} is different which shows the superiority of EDO over FPM DRAM.
61. less
62. t_{CAS}, t_{CP} ($t_{PC} = t_{CAS} + t_{CP}$). The t_{CP} is same across FPM and EDO DRAMs since it is the absolute minimum time that CAS has to stay high before it
is pulled down. However t_{CAS} is shorter in EDO, hence its superiority.
63. Synchronous DRAM
64. The presence of a common clock between the CPU and SDRAM memory by which bus activities are synchronized. There is no such common clock connecting EDO or FPM DRAMs to the CPU.
65. 13.3 ns since 1/75 MHz = 13.3 ns
66. 8.3 ns since 1/120 MHz = 8.3 ns
67. 125 MHz
68. Since there is burst mode read in the CPU bus cycle time there are also burst mode RAMs. See Chapter 23 (15) for burst mode READ of 486 and Pentium processors. In burst mode RAMs, the first addressed data is accessed in the normal access time; however, each subsequent data is fetched much faster. The burst length is the number of locations inside RAM that can be accessed with the shorter access time after the first location has been accessed. In many RAMs this length can be programmed. It can be 2, 4, 8, and so on.
69. It can be 2, 4, 8, up to 256, or in some cases up to an entire page. Some pages go as high as 1024.
70. True
71. In the motherboard design with interleaved memory, we lay the DRAM banks side by side in order to hide the access time of one bank behind the precharge time of the other one. However, in SDRAM of the interleaved type, this is done inside the DRAM chip on the die itself.

72. False
73. Rambus DRAM. It is mainly a proprietary bus connection for the CPU- to-memory communication. This means that DRAM makers must license the RDRAM technology from RAMBUS Inc. for a fee. However EDO and SDRAM DRAMs are industry standards with no royalties to be paid to anyone.
74. (1) Rambus interface (2) Rambus channel, and (3) Rambus DRAM
75. True
76. 9
77. slave
78. True
79. True
80. (1) request, (2) acknowledge, and (3) data
81. part of the acknowledge packet
82. When data is requested by the master, the slave sends an acknowledgment. In the acknowledgment response there are three possibilities: (a) the data does not exist, (b) the data exists but the slave is too busy to send it (nack) and finally , (c) the data exists and the slave is ready to transfer (okay).
83. 500 MHz, that is theoretically.
84. Such a CPU must be connected to the Rambus controller in order to communicate with the RDRAM type memory. In other words, we must place a Rambus controller in between this CPU and RDRAM.
85. False

CHAPTER 23: PENTIUM AND RISC PROCESSORS

SECTION 23.1: THE 80486 MICROPROCESSOR

1. 168
2. 32
3. 8K bytes
4. L2 (level 2) or secondary cache
5. true
6. The 486 has both the microprocessor and the math coprocessor all on one chip. The 486SX does not have a math coprocessor. The math coprocessor is a separate chip called the 487SX.
7. 32
8. 4
9. 4 gigabytes using its 32-bit address bus (2^32 = 4G)
10. 4: BE0 - BE3
11. 2
12. It means that in reading 4 consecutive words (32-bit size) it will take 2 clocks for each word.
13. It means that in reading 4 consecutive words (32-bit size) the first one will take 2 clocks, and each of three subsequent ones will take only one clock.
14. 4 x 4 = 16 bytes; a total of 5 clocks since we have 2-1-1-1
15. (a) With the clock period of 1/25 MHz = 40 ns, we have 2 x 40 = 80 ns for the memory cycle time; therefore, the bus bandwidth is (1/(80 ns)) × 4 bytes = 50 Mbytes/sec.
    (b) In the burst cycle, 5 clocks are used to transfer 4 consecutively located words of 32-bit size. Each 32-bit word takes a 1.25 clock period (5/4 = 1.25); therefore, the bus bandwidth is [1/(1.25 x 40 ns)] x 4 bytes = 80Mbytes/sec.
16. (a) With the clock period of 1/33 MHz = 30 ns, we have 2 x 30 = 60 ns for the memory cycle time; therefore, the bus bandwidth is (1/(60 ns)) × 4 bytes = 66.6 Mbytes/sec.
    (b) In the burst cycle, 5 clocks are used to transfer 4 consecutively located words of 32-bit size for each 32-bit 1.25 clock period (5/4 = 1.25); therefore, the bus bandwidth is [1/(1.25 x 30 nsec)] x 4 bytes = 106 Mbytes/sec.
17. BSWAP (byte swap)
18. DS:4000=(12) DS:4001=(65) DS:4002=(F4) DS:4003=(23) DS:6000=(23)
    DS:6001=(F4) DS:6002=(65) DS:6003=(12)
    all values are in hex
19. 25 MHz
20. to mask or unmask the A20 address bit; This will eliminate the need for external circuitry called the A20 gate used for 386/286 CPUs.
21. (a) When input A20M=0, the CPU's A20 address bit is masked and forced to 0. Since FFFF0H + 76A0H = 10769H; dropping the 1 we have 07690H as the physical address.
    (b) When input A20M=1, the CPU's A20 address pin provides the A20 address bit. Since FFFF0H + 76A0H = 10769H where A20=1, we have 107690H as the physical address.
22. It is 65,520 bytes of memory located above the 1M address space of FFFFFH and accessible by DOS using 286/386/486/Pentium processors in real mode. The A20M input pin of the 486 allows masking or unmasking the A20 address bus, thereby making the 486 compatible with 8088/86 or 286/386 CPUs.
23. 5
24. 1. prefetch 2. decode1 3. decode2 4. execute 5. write back

SECTION 23.2: INTEL'S PENTIUM

25. more
26. superscalar
27. 3.1 million, 273 pins
28. 64, D0 - D63
29. 32

30. It has 8 of them, BE0 - BE7, where BE0 is used for D0 - D7, BE1 for D8 - D15, and so on. Their purpose is to allow the selection of individual byte buses.
31. low
32. D0 - D31
33. D0 - D63
34. 8, one for each byte of the data bus
35. The clock period is 1/60 MHz = 16.6 ns, therefore
    (a) [1/(2 x 16.6 ns)] x 8 bytes = 240 megabytes/second
    (b) [1/(1.25 x 16.6 ns)] x 8 bytes = 385 megabytes/second

36. 16K bytes
37. 8K bytes for data and 8K bytes for code: a total of 32K bytes
38. code
39. true
40. To issue two instructions simultaneously, one to each execution unit; therefore, executing two instructions in one clock.
41. superscalar
42. when two instructions are issued, one to each execution unit at the same time; these paired instructions must not have any dependency
43. when one instruction needs the data result of another instruction in order to proceed with its own completion. It is avoided most of the time by rearranging the sequence of instruction flow (scheduling).
44.

```
        .MODEL   SMALL
        .386
        .STACK   300H
        .DATA
DATA1   DD       10 DUP(9999999H)   ;ARRAY OF 10 DWORD
SUM     DQ       ?
COUNT   EQU      10
        .CODE
MAIN:   MOV      AX,@DATA
        MOV      DS,AX
        SUB      EBX,EBX                    ;clear EBX
        MOV      EAX,EBX                    ;clear EAX
        MOV      EDX,EAX                    ;clear EDX
        MOV      CX,COUNT                   ;counter
        MOV      ESI,OFFSET DATA1           ;load address
BACK:   ADD      EAX,[ESI+EDX*4]            ;add the operand
        ADC      EBX,0                      ;save the carry
        INC      EDX                        ;increment the pointer
        DEC      CX                         ;continue until counter=0
        JNZ      BACK
        MOV      DWORD PTR SUM,EAX          ;save the sum
        MOV      DWORD PTR SUM+4,EBX
        MOV      AH,4CH                     ;exit
        INT      21H                        ;to DOS
        END      MAIN
```

45. (a) and (b)

			386	486
BACK:	ADD	EAX,[ESI+EDX*4]	6	2
	ADC	EBX,0	2	1
	INC	EDX	2	1
	DEC	CX	2	1
	JNZ	BACK	7/3	3/1
Total clock count for one iteration			19	8

(c) Since there are no detail information about the calculation of the effective address of [ESI+EDX*4] from Intel plus the fact that it is followed by the ADC we assume the first instruction will take 2 clocks (This is a speculation). Therefore with the branch prediction and instruction pairing we have:

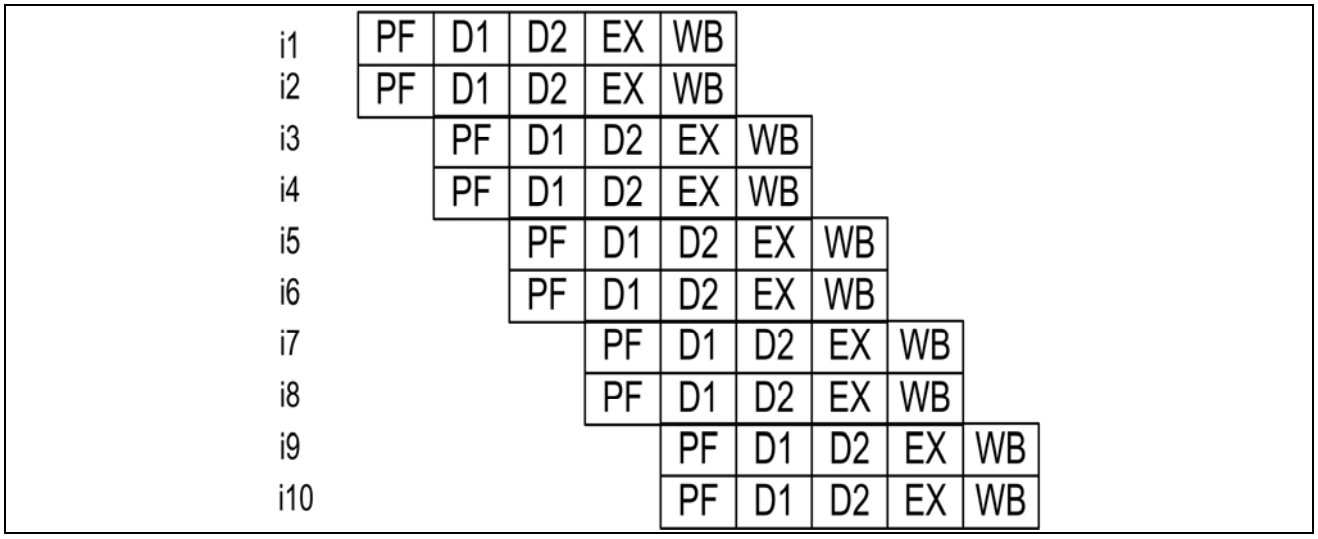
			Pentium
BACK:	ADD	EAX,[ESI+EDX*4]	
	ADC	EBX,0	2
	INC	EDX	
	DEC	CX	1
	JNZ	BACK	1
Total clock count for one iteration			4

(d) by swapping the "ADC EBX,0" and "INC EDX" the dependency of the ADC is removed. Now we have:

BACK:	ADD EAX,[ESI+EDX*4]	
	INC EDX	1
	ADC EBX,0	
	DEC CX	1
	JNZ BACK	1
Total clock count for one iteration		3

46. The clock period is 1/25 MHz =40 ns; bus bandwidth =1[(2 x 40 ns)] x 4 bytes =50 megabytes/second, 6 MB/second.

47.



48. true

SECTION 23.3: RISC ARCHITECTURE

49. because all arithmetic and logic operations must use registers for the source and destination operands; therefore to perform any operation on operands, both must be loaded from memory into register first and then after the operation it is stored back into memory
50. 4 bytes
51. OR reg,mem; In instruction "MOV reg,immediate", the immediate operand size is limited to 16 bits since the size of the instruction must not exceed 32 bits in RISC processors
52.

```
MOV     REG2,MEM                ;load the operand from memory to reg
ADD     REG2,REG1               ;add the other operand
MOV     MEM,REG2                ;now store the result back into memory
```

53. It makes the task of instruction decoding by the CPU much easier and uses less transistors.
54. because RISC uses a series of simple instructions to perform a single CISC instruction
55. 1
56. The idea of using separate buses to access the code and data sections of a program is called Harvard architecture. Since all programs, including those written for 80x86 CISC processors, use separate regions (segments) of memory for data and code, one can use the Harvard architecture for CISC as well.
57.

```
;r8 = pointer to location for SUM
;r3 = pointer to data block
            or      10,r0,r2            ;load the counter value (10) into r2
            or      r0,r0,r1            ;clear r1 (r1 contains SUM)
again       ld      0(r3),r7            ;load operand pointed to by r3 into r7
            nop                         ;r7 cannot be used right away
            add     r7,r1,r1            ;add the operand to r1
            add     4,r3,r3             ;point to next doubleword operand
            add     -1,r2,r2            ;decrement counter
            or      r0,r2,r2            ;set condition code to high if r2=0
            bnc.t   again               ;loop back
            nop
            st.1    r1,(r4)             ;store SUM
```

58. The code scheduling and clock counts for the loop section are follows.

```
                                        clock count
again       ld      0(r3),r7                1
            add     4,r3,r3                 1
            add     r7,r1,r1                1
            add     -1,r2,r2                1
            or      r0,r2,r2                1
            bnc.t   again                   1 (with branch prediction)
                                            6 clocks
```

59. the idea of executing the instruction below the jump instruction even though the program may branch to new address after the execution of the jump
60. b
61. 32
62. R0
63. Native code is written using the instructions specific to that processor. For example, DOS runs native on the x86. Emulation is where one computer mimics the hardware and instruction set of another system. For example, Macintosh 68000 emulates DOS. Porting requires rewriting or modifying the code from one platform in order for it to operate on another system.

SECTION 23.4: PENTIUM PRO PROCESSOR

64. 32, 64
65. 36
66. 2^{36} = 64 gigabytes
67. 64
68. It is the internal frequency.
69. In the Pentium, the L2 cache is a separate set of SRAM chips; however, in the Pentium Pro the CPU and L2 cache are on the same package (but different die). Intel also has a Pentium Pro without the L2 cache on the same package. In this case it is like the Pentium chip that needs SRAM to be connected to it externally.
70. No. It is only for the processor itself without the L2 cache.
71. Pentium Pro
72. No. It is used internally.
73. True
74. False
75. True
76. Retire unit
77. True
78. Both Pentium and Pentium Pro

SECTION 23.5: MMX TECHNOLOGY

79. False. It has a whole new set of instructions for that purpose.
80. In such a system, the x86 is responsible for communicating with memory and peripherals; the DSP chip is for performing digital signal processing and multimedia functions.
81. No
82. It is the idea of the same register set being used by two different sections of the CPU. Each section can use the register but not at the same time. It is like one person having two different names.
83. R0 - R7 ((ST(0) - ST(7)) of the x87 math processor
84. ST(0) - ST(7) of the x87 are accessed in the stack format meaning that the last one is the first one out (LIFO). In MMX, the registers are accessed by their names MR0 - MR7 in the same way as AX, BX, and so on.
85. (a), (b)
86. It should clear all the MMX registers and point to the top of stack.
87. It should pop all the FP registers (setting the stack at the top) to leave them cleared.
88. 64
89. 64
90. Since in the x86, a word is defined as 16 bits, 64-bit data can be used as two doublewords, four words, or 8 bytes.
91. False
92. We check for D18 of the flag bit. If it is low it is 386.
93. We try to change bit D21 of the flag register. If it cannot be changed then it is a 486.

94 & 95. First we try to see if bit D21 of the flag register can be changed. In the x86, the CPUID instruction is supported if D21 can be altered (set or reset). It is only after this determination that we use the CPUID instruction with EAX=01. Upon return from CPUID, bits D8 - D11 indicate the family number where 101 (binary) is for the Pentium and 110 (binary) is for the Pentium Pro.

96. Upon return from CPUID, we must check D23 of EDX. If it is high, it has MMX capability.
97. We can use a simple macro such as:

```
CPUID   MACRO
DB      0FH
DB      0A2H
ENDM
```

CHAPTER 24: THE EVOLUTION OF x86: FROM 32-BIT TO 64-BIT

SECTION 24.1: x86 PENTIUM EVOLUTION

1. True
2. True
3. 16KB
4. 512KB
5. 12KB
6. 1MB
7. 32KB
8. 2 MB
9. L1 cache works at the same speed as the CPU (or close to it) while L2 works at a fraction (around 1/3) of the CPU speed.
10. L3 cache is outside the CPU and sitting on the motherboard. It works at the speed of 1/6 to 1/10 of CPU.
11. There is no L3 cache.
12. L1 cache works at the same speed as the CPU (or close to it) while L2 works at a fraction (around 1/3) of the CPU speed. L3 cache is outside the CPU and sitting on the motherboard. It works at the speed of 1/6 to 1/10 of CPU.
13. False. Only Pentium 4 and on
14. False. Only Pentium 4 and on.
15. In the multithreading the CPU takes advantage of parallelism in a program while in the multitasking the CPU swaps in and out different programs(That is it takes turn to executes different programs.)
16. Multicore CPU have multiple processing cores inside the a single CPU, while the in multiprocessors we have multiple CPUs on motherboard.
17. The MMX did not support floating point operations while XMM does
18. True
19. True
20. True

SECTION 24.2: 64-BIT PROCESSORS AND VISTA FOR x86

21. False. Yes it can.
22. RAX, RBX, RCX, RDX, RSI, RDI,RBP, RSP, R9, R10, …and R15.
23. EAX, EBX,ECX, EDX, ESI, EDI, EBP, and ESP, R0…..and R15
24. AX, BX, CX, DX, SI,DI, BP, and SP, R0…and R15
25. AH, AL, BH, BL, CH, CL, DH, and DL, R0, ….and R0
26. The compatibility mode and 64-bit mode
27. 2**32
28. 2**64
29. 4GB
30. 2**44 = 8 Terabyte, but only 128 GB is implemented
31. No. Only in 64-bit
32. 128GB

CHAPTER 25: SYSTEM DESIGN ISSUES AND FAILURE ANALYSIS

SECTION 25.1: OVERVIEW OF IC TECHNOLOGY

1. When electrons move from emitter to collector, they must overcome two pin junctions, and the second one, especially, consumes energy since it is reverse bias.
2. gate input capacitance; the charge-up time adds to the delay
3. because NMOS uses electrons as carriers and electrons travel much faster than the hole which is the carrier for the PMOS
4. (a) Q2 and Q4 are "on" while Q3 is "off"
   (b) Q2 and Q4 are "off" while Q3 is "on"
5. (a) PMOS is "off" and NMOS is "on"
   (b) PMOS is "on" and NMOS is "off"
6. because the amount of time where the path from VDD to VSS is conducting is also increased as the frequency rises, therefore drawing more current
7. to prevent the transistors from going into deep saturation where it takes longer to recover to the "off" state
8. NM for "0" is =VIL - VOL = 0.8 V - 0.5 V = 0.3 V.
   NM for "1" is =VOH -VIH = 2.7 V - 2.0 V = 0.7 V.
9. NM for "0" is =VIL - VOL = 1.1 V - 0.4 V = 0.7 V.
   NM for "1" is =VOH -VIH = 3.7 V - 3.15V = 0.55 V.
10. LS
11. ALS
12. ALS
13. FCT; ACT is more power efficient
14. The FCT logic family is a CMOS version of TTL based FAST logic family.
15. For designers who are familiar with the FAST logic family, the migration to FAST is easy since FASTr has much higher IOH and IOL and is less noisy.
16. It combines speed of bipolar and power efficiency of CMOS technology.
17. true
18. False, it is used for systems with speeds in the range of 30 MHz - 66 MHz.
19. 0.8 micron
20. It is the only way one can put millions of transistors together to make a powerful CPU. Bipolar dissipates too much heat, thereby limiting the number of transistors that can be put on a single chip.

SECTION 25.2: IC INTERFACING AND SYSTEM DESIGN ISSUES

21.

$$\frac{I_{OH}}{I_{IH}} = \frac{400\mu A}{20\mu A} = 20 \text{ , } \frac{I_{OL}}{I_{IL}} = \frac{8mA}{0.2mA} = 40$$

Therefore, the fan-out is 20.

22.

$$\frac{I_{OH}}{I_{IH}} = \frac{400\mu A}{40\mu A} = 10 \text{ , } \frac{I_{OL}}{I_{IL}} = \frac{8mA}{1.6mA} = 5$$

Therefore, the fan-out is 5.

23.

$$\frac{I_{OH}}{I_{IH}} = \frac{400\mu A}{40\mu A} = 10 \text{ , } \frac{I_{OL}}{I_{IL}} = \frac{4mA}{1.6mA} = 2.5$$

Therefore, the fan-out is 2.

24.

$$\frac{I_{OH}}{I_{IH}} = \frac{3mA}{20\mu A} = 150 \text{ , } \frac{I_{OL}}{I_{IL}} = \frac{12mA}{0.36mA} = 33$$

Therefore, the fan-out is 33.

25.

$I_{OL} = \sum I_{IL} = 10 \times 0.36\ mA + 20 \times 10\ \mu A = 3.8mA$

$I_{OH} = \sum I_{IH} = 10 \times 20\ \mu A + 20 \times 10\ \mu A = 400\ \mu A$

26. true
27. high
28. A single output of the 244 goes to 64 inputs, since it goes to two banks, each with 32 inputs. Therefore, the total C_{in} = 64 x 7 pF = 448 pF.
For the derating, [448 pF - 50 pF = 398 pF since the 244 handles only 50 pF] (398/100) x 3 ns = 11.94 ns.
Address path delay = 8 ns + 11.94 ns + 15 ns = 34.94 ns.
29. A single output of the 244 goes to 32 inputs, since it goes to one bank with 32 inputs. Therefore, the total C_{in} = 32 x 7 pF = 224 pF.
For the derating, [224 pF - 50 pF = 174 pF since the 244 handles only 50 pF] (174/100) x 3 ns = 5.22 ns.
Address path delay = 8 ns + 5.22 ns + 15 ns = 28.22 ns.
30. Since a single output of the 244 drives only 16 inputs, the C_{in} = 16 x 7 pF = 112 pF. Therefore, the delay due to capacitance derating is small (2 ns) and address path delay = 8 ns + 2 ns + 15 ns = 25 ns.
31. dynamics
32. (1) it dissipates less power, (2) It allows further reduction in the line size. For example in the Pentium of V_{CC} = 5V, the line size is 0.8 micron while Pentium II of V_{CC} = 3.3V, the line size is 0.6 micron, (3) smaller line size allows higher speed chip, (4) a smaller line size also increases the number of transistors per square inch in the chip design, thereby resulting in a higher chip density.
33. $5^2 - 3.7^2 = 25 - 13.69 = 11.31$ W less or 45.24% power savings
34. (a) Using 256Kx1, we need 8 chips of 256Kx1; in order to access a byte, all 8 chips are active and there is no inactive (standby) chip; therefore we have $P = (N \times IACT + M \times I_{SB} \times 5V = (8 \times 230\ mA + 0)\ 5V = 9.2$ watts.
(b) Using 32Kx8, we need 8 banks of 32Kx8 but to access
a byte only one of the chips is active and the other 7 chips are inactive.
Therefore, we have $P = (N \times I_{ACT} + M \times I_{SB}) \times 5V = (1 \times 230\ mA + 7 \times 50mA) \times 5V = 2.9$ watts.

35. (a) $P = (N \times I_{ACT} + M \times I_{SB}) \times 3.3\ V = (8 \times 230\ mA + 0) \times 3.3V = 6.072$ watts.

(b) $P = (N \times I_{ACT} + M \times I_{SB}) \times 3.3\ V = (1 \times 230\ mA + 7 \times 50\ mA) \times 3.3\ V = 1.914$ watts.

36. Moving VCC and GND to the middle of the package reduces the length of the wire, which in turn decreases the inductance (L) and results in lower ground bounce and VCC bounce.
37. Ground bounce happens when a large number of outputs change state from high to low, causing a massive amount of current to flow through the ground pin, which raises the V_{OL} voltage level beyond the permitted noise margin. The cures are
(1) If possible one must use memories with 4 or 8 data buses instead of 16 or 32, (2) Make the length of the ground pin as small as possible, (3) Use chips which have many ground pins instead of one, (4) Use IC chips where the ground pin is located in the middle instead of the corners.

38. because the noise margin for V_{OH} is normally much higher than the V_{OL} noise margin
38. In TTL, pins 7 and 14 are used for ground and V_{CC}, respectively. The location of V_{CC} and ground pins at the corners increases the length of the wire causing a bigger V_{CC} and ground bounce. In ACT chips, the V_{CC} and ground pins are located in the middle which means a much shorter wire length and as result, less inductance and consequently, less voltage swing since V = L di/dt.
39. In totem pole TTL, there are times, however short, that both output transistors are on. This results in drawing a large amount of current, and consequently a spike on the output current. Placing a 0.01 uF (or 0.1uF) ceramic disk capacitor between V_{CC} and ground plane for each chip will reduce the transient current substantially.
40. for the following reasons, (1) All these devices must be isolated (buffered) from the main data bus the same way that the CPU must be buffered (isolated) from the main data bus, (2) each device has a different current driving capability, (3) To reduce the load on the main data bus driver.

41. The mutual inductance as a result of the two conducting lines running in parallel is called crosstalk. To reduce crosstalk, we need to (1) if possible place the parallel conducting lines at some distance from each other, (2) use dedicated ground lines for each signal line.
42. When a signal travels on a line not all the harmonics, which the square line is made of, respond the same way. In many cases the square wave looks distorted. The main reason behind the ringing is because some signals bounce back and run into the incoming signal causing distortion.
43. FCT, LS.
44. It is serial termination in order to reduce the ringing.
45. A hard error is permanent while a soft error is not.
46. A hard error, permanent damage to a storage cell, is due to wear and tear of the device and nothing can be done about it. A soft error, the inadvertent change in the contents of cell, is due to alpha particle radiation in the air and the chip's encapsulating plastic package, and power surge.
47. To implement the 16M of system memory using the 1M×1 DRAM chips, we need (16M × 8) / 1Mx1 = 128 chips. The MTBF for 16M of system memory =453 years / 128 chips = 3.53 years. (1,000,000,000 hours / 252 = 453 years MTBF for one chip)
48. 16M of memory using 32-bit words has 4 banks each with 32 1Mx1 memory chip since 128/32 = 4 banks. This means that we have 4M words of 32-bit size. Since each 32-bit word requires an extra 7 chips for EDC, a total of (16Mx8/32Mx1) x 7 =4 x 7 = 28 DRAM chips are used by EDC. This results in a total chip count of 156 (128 + 28) of 1Mx1 chips. Since MTBF for a single chip is 453, we have:

$$\text{MTBF of 156 DRAM} = \frac{453 years}{156} = 2.9 years$$

$$\text{MTBF with EDC} = 2.9 years \times \sqrt{\frac{\pi \times 4M}{2}} = 7441.7 years$$

CHAPTER 26: ISA, PC104, AND PCI BUSES

SECTION 26.1: ISA BUS MEMORY SIGNALS

1. MEMW and MEMR on the 36-pin section of the ISA bus
2. 24 megabytes since we have A0 - A23
3. low, yes
4. The 8-bit D0 - D7 is the default mode in ISA. Therefore, to use the entire D0 - D15 data bus we must assert the MEMCS16 pin low.
5. low, yes
6. The ZEROWS pin is used to tell the system board to perform the read and write cycle time with zero wait states. For 16-bit ISA, memory read and write has 1 WS, unless the ZEROWS pin is aserted low. For 8-bit ISA, if ZEROWS is asserted low, the read/write cycle has 1 WS instead of 4 WS.
7. D0 - D7 portion
8. the entire D0 - D15
9. 6 × 125 ns = 750 ns (4WS + 2)
10. 3 × 125 ns = 375 ns (1WS + 2)
11. 2 × 125 ns = 250 ns (0WS + 2)
12. 3 × 125 ns = 375 ns (1WS + 2)
13. Both need to be asserted low.
14.

MEMCS16	ZEROWS	Data bus used	Read Cycle Time	Bus Bandwidth
0	0	D0 - D15	250 ns	8MB/sec
0	1	D0 - D15	375 ns	5.33MB/sec
1	0	D0 - D7	375 ns	2.66 MB/sec
1	1	D0 - D7	750 ns	1.33MB/sec

15. More memory on smaller cards and much shorter access time than the ISA bus.

SECTION 26.2: I/O BUS TIMING IN ISA BUS

16. The ISA bus signals on the expansion slot have certain number of WS inserted as default. The ZEROWS pin is used to inform the motherboard that we want to eliminate these WS. This allows to shorten the I/O bus cycle.
17. The ISA bus on the expansion slot is an 8-bit bus as default. The IOCS16pin is used to inform the motherboard that we want to override the default and use the 16-bit buses This allows to use D0-D15 buses instead of D0-D7.
18. 8-bit
19. 8 MHz which gives 1/8Mhz=125 nsec.
20. 4 WS
21. 4WS + 2 clocks =6 clocks and 6x 125 ns= 750 nsec is the time it takes to finish one I/O cycle.
22. 1 WS
23. 1WS + 2 clocks =3 clocks and 3x 125 ns= 375 nsec is the time it takes to finish one I/O cycle when ZEROWS pin is asserted.
24. 1 WS
25. 1WS + 2 clocks =3 clocks and 3x 125 ns= 375 nsec is the time it takes to finish one I/O cycle of 16-bit default.
26. input, low
27. input, low
28. Since the I/O cycle time is 750 nsec(6x125 ns=750 ns) we have 1/750 ns x1byte=1.33 mega bytes/sec
29. Since the I/O cycle time for 16-bit standard is 375 nsec(3x125 ns=375 ns) we have 1/375 ns x2byte=5.33 mega bytes/sec
30. While ZEROWS is used to shorten the I/O bus cycle we increase its duration by using the CHANRDY pin. By asserting the CHANRDY we can insert up to 10 WS into I/O cycle time.
31. By asserting the CHANRDY pin we can insert up to 10 WS into I/O cycle time.

SECTION 26.3: PCI BUS

32. 33 MHz
33. true
34. true
35. False, it is allowed.
36. In PCI ,the bus frequency is 33 MHz, and the memory cycle is 2 clocks 2 x 30 ns = 60 ns; therefore, the bus bandwidth is [1/(60 ns)] x 4 = 66 Mbyte/sec.
37. (a) Much of the PCI documentation states that the maximum bus bandwidth of a 32-bit PCI bus at 33 MHz is 133M bytes. The reason is that in their calculations, they assume one clock per memory (or I/O) cycle using burst mode. In that case, the first clock is used for the addresses and in each of the subsequent clocks the data is transferred for hundreds of consecutive cycles. Therefore, the bandwidth is [1/(30 ns)] × 4 = 133 megabytes/second.
    (b) [1/(30 ns)] x 8 = 266 megabytes/second

CHAPTER 27: USB PORT PROGRAMMING

SECTION 27.1: USB PORTS: AN OVERVIEW

1. 127 for USB, 4 for COM and 4 for LPT
2. 25 W for ISA, 10 W for PCI, and 500 mW for USB
3. 1. Mb/sec, 12Mb/s, and 480Mb/s.
4. It means we can plug and unplug the device into the system while the system power is on
5. USB

SECTION 27.2: USB PORT EXPANSION AND POWER MANAGEMENT

6. True
7. True
8. True
9. False
10. True
11. The data going toward host is called upstream and data going from host toward peripherals is called downstream
12. Host is the master which controls the slave or peripheral devices. If a host has limited number of connecting ports for peripheral connection, then we need a hub to expand the number of connection
13. maximum 500mA
14. maximum 500mA
15. 7
16. In the self-powered power comes from external power supply, while in the bus powered the power comes from the USB host.
17. The cable with both ends of A-type is used to extend cable.
18. It is used for connecting a hub to a device.
19. 4
20. The D+ and D- are differential lines carrying data, GND and VCC provide power to the device.
21. 5 meters (15 feet)
22. NRZI (none return to zero inverted) encoding
23. True
24. The process of recognizing a device and assigning an address to it is called enumeration
25. the USB host
26. 500mA
27. The A-type is used for connecting to a hub (upstream) while the B-side is used for connecting it to a device.
28. total of 127
29. 1 - 127
30. False
31. True
32. 100mA

SECTION 27.3: USB PORT PROGRAMMING

33. See MicroDigitalEd.com for solution.