NATIONAL UNIVERSITY OF SCIENCE AND TECHNOLOGY

FACULTY OF ENGINEERING

DEPARTMENT OF ELECTRONIC ENGINEERING

MICROPROCESSORS

TEE 3112 / TEE 2112

Special Supplementary Final Examination Paper

August 2024

This examination paper consists of 4 pages and 10 pages of attached appendices

Time Allowed: 3 hours

Total Marks: 100

Special Requirements: Intel 8086 Instruction Set and ASCII Tables

Examiner's Name: Reginald Gonye

INSTRUCTIONS AND INFORMATION TO CANDIDATES

1. Answer **ALL** questions.
2. All microprocessor instructions and programming questions refer to the Intel 8086 microprocessor unless specifically stated otherwise.
3. Show all your steps clearly in any calculations.
4. Start each new question on a fresh page.

Question 1

What are uses of each of the following

a) Arithmetic logic unit in a microprocessor [2 marks]
b) Memory address register (MAR) in a microprocessor [2 marks]
c) Read control signal in a microprocessor-based system. [2 marks]
d) A segment register in the 8086 microprocessor [2 marks]

Question 2

The following is a program segment executing on the Intel 8086 microprocessor at some given time. Before the start of the program segment, registers AX, DX and DI contain zeros. The carry flag is cleared and the memory locations 0871H, 0872H and 0873H also contain zeros. The voltages at port pins of port address 06H are 5V, 5V, 0V, 0V, 5V, 0V, 5V, 0V starting with the most significant position.

```
MOV   DI,0870H
IN    AL,06H
MOV   DH,AL
AND   AL,3FH
MOV   (DI+03H),AL
DEC   DI
MOV   AL,DH
ADD   AL,3FH
MOV   (DI+03H),AL
OUT   07H,AL
```

What will be the contents of the registers AX, DX and DI; memory locations 0871H, 0872H and 0873H; the state of the carry flag; and the states of the pins on port 07h after the segment has finished executing? [10 marks]

Question 3

Assemble the following program segment by hand. [7 marks]

```
portA equ 80h

org   1800h
MOV   AX,[BX]
AND   AX,0FH
ADD   AX,DX
OUT   portA,AX
```

Question 4

Convert the following section of a C program statement into a suitable block of Intel 8086 assembly language instructions. [10 marks]

```
x = b*b - 4ac;
```

a, *b* and *c* are 8-bit values in memory locations at addresses 0990h, 0991h and 0992h respectively. *x* is a 16-bit value in memory locations 1993h and 1994h.

Question 5

What is direct memory access (DMA)? Describe in detail what happens from the time a DMA request is received by a Microprocessor the DMA completes. [7 marks]

Question 6

Briefly outline what you understand as floating point numbers in digital systems. Outline how microprocessors can be designed to perform operations on floating point numbers. [5 marks]

Question 7

What do you understand as parallel processing in a microprocessor? You may use diagrams to aid your explanation. [5 marks]

Question 8

A microprocessor has an address space of **1M words** where each word is 16 bits wide. It has 16 data lines, 24 control lines, 4 pins for power and 4 pins for the clock.

a) How many byte-enable pins are required on the microprocessor and memory at once? [1 mark]
b) What is the address bus width? [2 marks]
c) How many pins in total does the microprocessor have? [2 marks]
d) The memory chips exist as 128k × 8-bit chips. Determine the total number of chips required to provide the memory. [5 marks]

Question 9

An Intel 8086 program is required to read ASCII characters from a parallel input port and store them sequentially in memory starting at address **3000H**. Any lower case letters read from the port need to be converted to lower case before storage. All other characters are stored as they are. A new character at the input port is signaled to the microprocessor by means of an INTR interrupt. The interrupt type number is 36 (i.e. causes branching to address defined by vector at address 90H).

a) Design a suitable interrupt handler to enable the system to log the received characters. Present the design as an assembly program.

b) Design a main program to configure the microprocessor for response to the interrupt request and initialize it for data the logging.

[20 marks]

Question 10

a) Figure Q10 shows the internal structure of a simple microprocessor.

i. Suggest a sequence of micro operations in the device to execute the instruction `ADC A,[D]`. [6 marks]

ii. Given the instruction in part (i) is two bytes long and the data bus is one byte wide, suggest a sequence of micro operations to fetch the instruction. [6 marks]

b) Briefly outline how a micro-programmed control unit works [6 marks]

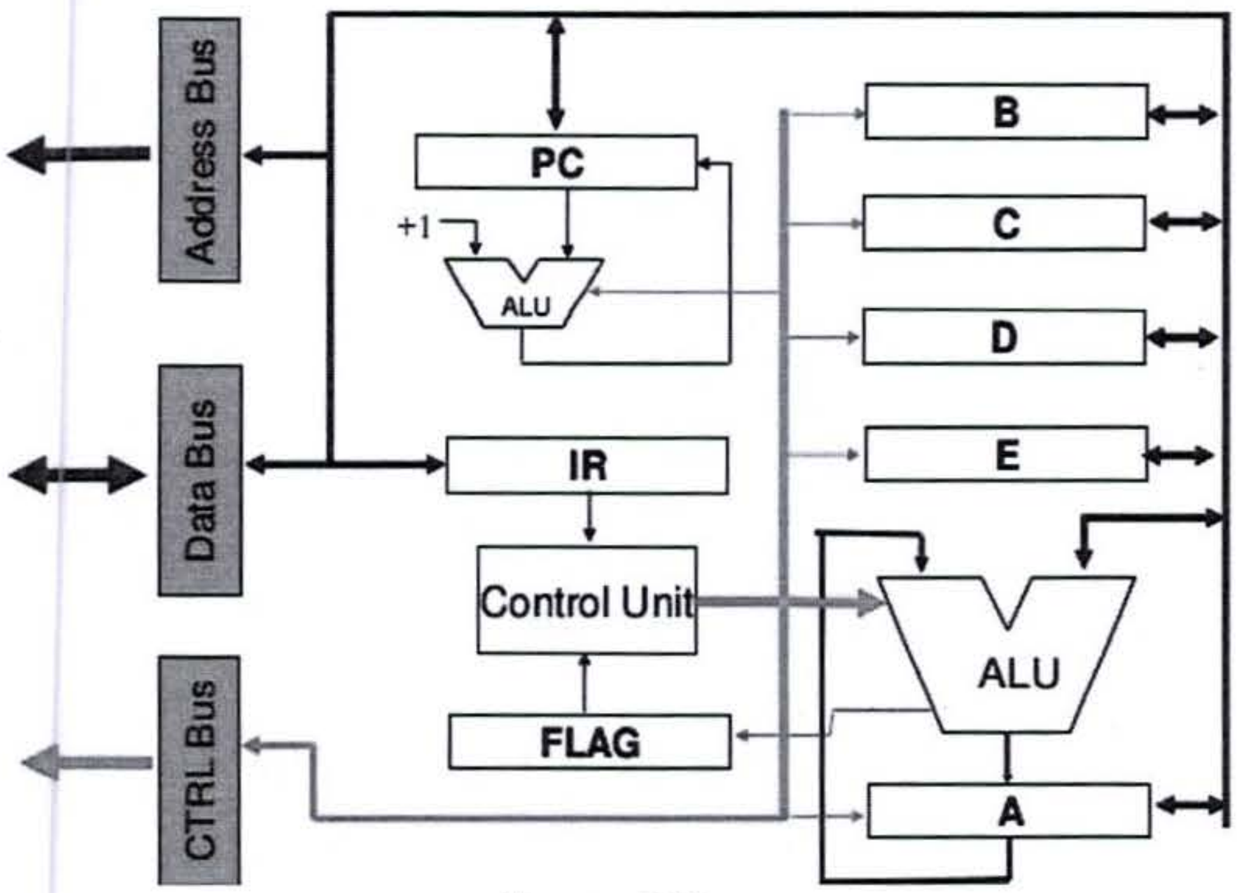


Figure Q10

END OF PAPER

Appendix A: 8086 Instruction Set

Data Transfer Instructions

x86 Mnemonic		Operation	x86 Mnemonic		Operation
MOV	reg16,reg16	reg16 ← reg16	MOV	reg8,reg8	reg8 ← reg8
MOV	reg16,mem	reg16 ← mem	MOV	reg8,mem	reg8 ← mem
MOV	mem,reg16	mem ← reg16	MOV	mem,reg8	mem ← reg8
MOV	reg16,imm16	Reg16 ← imm16	MOV	reg8,imm8	reg8 ← imm8
MOV	mem,imm16	mem ← imm16	MOV	mem,imm8	mem ← imm8
MOV	sreg,reg16	sreg ← reg16		-	-
MOV	sreg,mem	sreg ← mem		-	-
MOV	reg16,sreg	reg16 ← sreg		-	-
MOV	mem,sreg	mem ← sreg		-	-
XCHG	reg16,mem	reg16 ↔ mem		-	-
LEA	reg16	reg16 ← value of mem		-	-
LEA	mem	mem ← value of mem		-	-
LDS	reg16,mem	DS:reg16 ← mem		-	-
LES	reg16,mem	ES:reg16 ← mem		-	-
IN	AX,addr8	AX ← port [addr8]	IN	AL,addr8	AL ← port [addre8]
IN	AX,DX	AX ← port [DX]	IN	AL,DX	AL ← port [DX]
OUT	addr8, AX	Port [addr8] ← AX	OUT	addr8, AL	Port [addr8] ← AL
OUT	DX,AX	Port [DX] ← AX	OUT	DX,AL	Port [DX] ← AL
PUSH	reg16	[SP+2] ← reg16; SP ← SP - 2		-	-
PUSH	mem	[SP+2] ← mem; SP ← SP - 2		-	-
PUSH	sreg16	[SP+2] ← sreg; SP ← SP - 2		-	-
PUSH	imm16	[SP+2] ← imm16; SP ← SP - 2		-	-
POP	reg16	reg16 ← [SP]; SP ← SP + 2		-	-
POP	mem	reg16 ← [SP]; SP ← SP + 2		-	-
POP	sreg16	reg16 ← [SP]; SP ← SP + 2		-	-
PUSHF		[SP+1] ← FLAGS; SP ← SP - 2		-	-
POPF		FLAGS ← [SP] ; SP ← SP + 2		-	-
	-	-	LAHF		AH ← FLAGS
	-	-	SAHF		AH ← FLAGS
	-	-	XLAT		AL ← [BX+AL]

Arithmetic Instructions

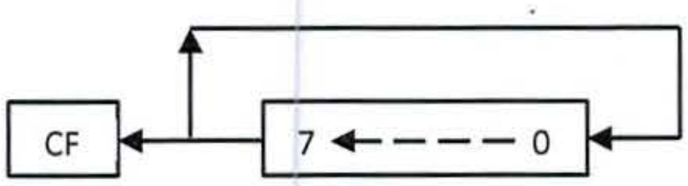
x86 Mnemonic		Operation	x86 Mnemonic		Operation
ADD	reg16,reg16	reg16 ← reg16 + reg16	ADD	reg8,reg8	reg8 ← reg8 + reg8
ADD	reg16,mem	reg16 ← reg16 + mem	ADD	reg8,mem	reg8 ← reg8 + mem
ADD	mem,reg16	mem ← mem + reg16	ADD	mem,reg8	mem ← mem + reg8
ADD	reg16,imm16	reg16 ← reg16 + imm16	ADD	reg8,imm8	reg8 ← reg8 + imm16
ADD	mem,imm16	mem ← mem + imm16	ADD	mem,imm8	mem ← mem + imm8
ADC	reg16,reg16	reg16 ← reg16 + reg16 + CF	ADC	reg8,reg8	reg8 ← reg8 + reg8 + CF
ADC	reg16,mem	reg16 ← reg16 + mem + CF	ADC	reg8,mem	reg8 ← reg8 + mem + CF
ADC	mem,reg16	mem ← mem + reg16 + CF	ADC	mem,reg8	mem ← mem + reg8 + CF
ADC	reg16,imm16	reg16 ← reg16 + imm16 + CF	ADC	reg8,imm8	reg8 ← reg8 + imm8 + CF
ADC	mem,imm16	mem ← mem + imm16 + CF	ADC	mem,imm8	mem ← mem + imm8 + CF
SUB	reg16,reg16	reg16 ← reg16 - reg16	SUB	reg8,reg8	reg8 ← reg8 - reg8
SUB	reg16,mem	reg16 ← reg16 - mem	SUB	reg8,mem	reg8 ← reg8 - mem
SUB	mem,reg16	mem ← mem - reg16	SUB	mem,reg8	mem ← mem - reg8
SUB	reg16,imm16	reg16 ← reg16 - imm16	SUB	reg8,imm8	reg8 ← reg8 – imm8
SUB	mem,imm16	mem ← mem - imm16	SUB	mem,imm8	mem ← mem - imm8
SBB	reg16,reg16	reg16 ← reg16 - reg16 - CF	SBB	reg8,reg8	reg8 ← reg8 - reg8 - CF
SBB	reg16,mem	reg16 ← reg16 - mem - CF	SBB	reg8,mem	reg8 ← reg8 - mem - CF
SBB	mem,reg16	mem ← mem - reg16 - CF	SBB	mem,reg8	mem ← mem - reg8 - CF
SBB	reg16,imm16	reg16 ← reg16 - imm16 - CF	SBB	reg8,imm8	reg8 ← reg8 - imm8 - CF
SBB	mem,imm16	mem ← mem - imm16 - CF	SBB	mem,imm8	mem ← mem - imm8 - CF
CMP	reg16,reg16	reg16 - reg16	CMP	reg8,reg8	reg8 - reg8
CMP	reg16,mem	reg16 - mem	CMP	reg8,mem	reg8 - mem
CMP	mem,reg16	mem - reg16	CMP	mem,reg8	mem - reg8
CMP	reg16,imm16	reg16 - imm16	CMP	reg8,imm8	reg8 - imm8
CMP	mem,imm16	mem - imm16	CMP	mem,imm8	mem - imm8
INC	reg16	Reg16 ← reg16 + 1	INC	reg8	reg8 ← reg8 + 1
INC	mem	mem ← mem + 1	INC	mem	mem ← mem + 1
DEC	reg16	reg16 ← reg16 - 1	DEC	reg8	reg8 ← reg8 - 1
DEC	mem	mem ← mem + 1	DEC	mem	mem ← mem + 1
NEG	reg16	reg16 ← -reg16	NEG	reg8	reg8 ← -reg8
NEG	mem	mem ← -mem	NEG	mem	mem ← -mem
MUL	reg16	DX:AX ← AX × reg16	MUL	reg8	AX ← AL × reg8
MUL	mem	DX:AX ← AX × mem	MUL	mem	AX ← ALX × mem
IMUL	reg16	DX:AX ← AX × reg16	IMUL	reg8	AX ← AL × reg8
IMUL	mem	DX:AX ← AX × mem	IMUL	mem	AX ← AL × mem
DIV	reg16	AX ← DX:AX / reg16, DX←rem	DIV	reg8	AL ← AX / reg16; AH←rem
DIV	mem	AX ← DX:AX × mem	DIV	mem	AL ← AX × mem
IDIV	reg16	AX ← DX:AX × reg16	IDIV	reg8	AL ← AX × reg16
IDIV	mem	AX ← DX:AX × mem	IDIV	mem	AL ← AX × mem
CWD			CBW		
			AAA		
			DAA		
			AAS		
			DAS		
			AAM		
			AAD		

Logic Instructions

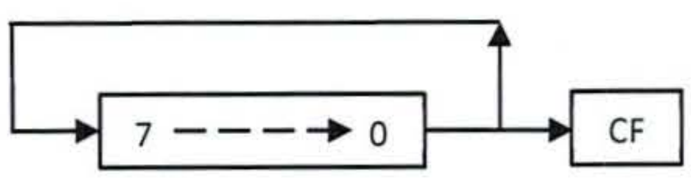
x86 Mnemonic		Operation	x86 Mnemonic		Operation
AND	reg16,reg16	reg16 ← reg16 • reg16	AND	reg8,reg8	reg8 ← reg8 • reg8
AND	reg16,mem	reg16 ← reg16 • mem	AND	reg8,mem	reg8 ← reg8 • mem
AND	mem,reg16	mem ← mem • reg16	AND	mem,reg8	mem ← mem • reg8
AND	reg16,imm16	reg16 ← reg16 • imm16	AND	reg8,imm8	reg8 ← reg8 • imm16
AND	mem,imm16	mem ← mem • imm16	AND	mem,imm8	mem ← mem • imm8
OR	reg16,reg16	reg16 ← reg16 | reg16	OR	reg8,reg8	reg8 ← reg8 | reg8
OR	reg16,mem	reg16 ← reg16 | mem	OR	reg8,mem	reg8 ← reg8 | mem
OR	mem,reg16	mem ← mem | reg16	OR	mem,reg8	mem ← mem | reg8
OR	reg16,imm16	reg16 ← reg16 | imm16	OR	reg8,imm8	reg8 ← reg8 | imm8
OR	mem,imm16	mem ← mem | imm16	OR	mem,imm8	mem ← mem | imm8
XOR	reg16,reg16	reg16 ← reg16 - reg16	XOR	reg8,reg8	reg8 ← reg8 - reg8
XOR	reg16,mem	reg16 ← reg16 - mem	XOR	reg8,mem	reg8 ← reg8 - mem
XOR	mem,reg16	mem ← mem - reg16	XOR	mem,reg8	mem ← mem - reg8
XOR	reg16,imm16	reg16 ← reg16 - imm16	XOR	reg8,imm8	reg8 ← reg8 – imm8
XOR	mem,imm16	mem ← mem - imm16	XOR	mem,imm8	mem ← mem - imm8
TEST	reg16,reg16	reg16 • reg16	TEST	reg8,reg8	reg8 • reg8
TEST	reg16,mem	reg16 • mem	TEST	reg8,mem	reg8 • mem
TEST	mem,reg16	mem • reg16	TEST	mem,reg8	mem • reg8
TEST	reg16,imm16	reg16 • imm16	TEST	reg8,imm8	reg8 • imm8
TEST	mem,imm16	mem • imm16	TEST	mem,imm8	mem • imm8

Shift and Rotate Instructions

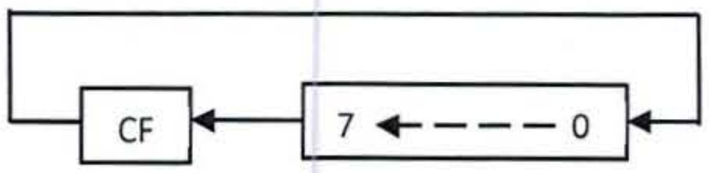
x86 Mnemonic		Operation	x86 Mnemonic		Operation
NOT	reg16		NOT	reg8	
NOT	mem		NOT	mem	
SHL/SAL	reg16,count		SHL/SAL	reg8,count	
SHL/SAL	mem,count		SHL/SAL	mem,count	
SHR	reg16,count		SHR	reg8,count	
SHR	mem,count		SHR	mem,count	
SAR	reg16,count		SAR	reg8,count	
SAR	mem,count		SAR	mem,count	
ROL	reg16,count		ROL	reg8,count	
ROL	mem,count		ROL	mem,count	
ROR	reg16,count		ROR	reg8,count	
ROR	mem,count		ROR	mem,count	
RCL	reg16,count		RCL	reg8,count	
RCL	mem,count		RCL	mem,count	
RCR	reg16,count		RCR	reg8,count	
RCR	mem,count		RCR	mem,count	



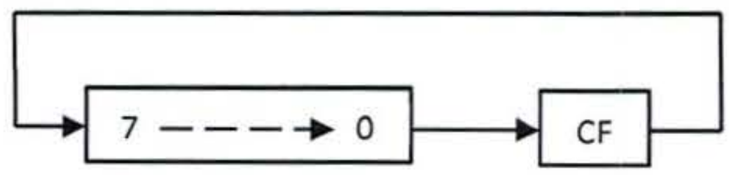
ROL



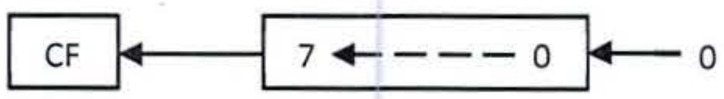
ROR



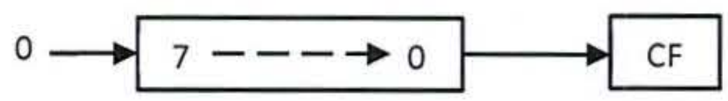
RCL



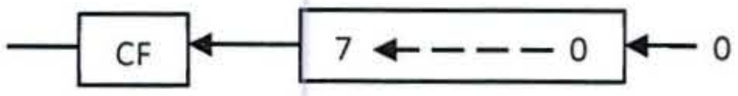
RCR



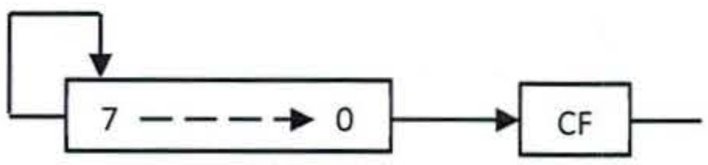
SHL



SHR



SAL



SAR

Control Transfer Instructions

x86 Mnemonic		Operation	x86 Mnemonic		Operation
					If condition met:
JMP	addr16	IP ← addr16	JE/JZ	disp8	IP ← IP + disp8 - 2
JMP	reg16	IP ← reg16	JL/JNGE	disp8	IP ← IP + disp8 - 2
JMP	mem	IP ← mem	JLE/JNG	disp8	IP ← IP + disp8 - 2
			JB/JNAE/JC	disp8	IP ← IP + disp8 - 2
JMP	seg:addr16	IP ← addr16; CS ← seg	JP/JPE	disp8	IP ← IP + disp8 - 2
JMP	seg:reg16	IP ← reg16; CS ← seg	JO	disp8	IP ← IP + disp8 - 2
JMP	seg:mem	IP ← mem; CS ← seg	JS	disp8	IP ← IP + disp8 - 2
			JNE/JNZ	disp8	IP ← IP + disp8 - 2
CALL	addr16	IP ← addr16	JNL/JGE	disp8	IP ← IP + disp8 - 2
CALL	reg16	IP ← reg16	JNLE/JG	disp8	IP ← IP + disp8 - 2
CALL	mem	IP ← mem	JNB/JAE/JNC	disp8	IP ← IP + disp8 - 2
			JNP/JPO	disp8	IP ← IP + disp8 - 2
CALL	seg:addr16	IP ← addr16; CS ← seg; SP ← SP -2	JNO	disp8	IP ← IP + disp8 - 2
CALL	seg:reg16	IP ← reg16; CS ← seg; SP ← SP -2	JNS	disp8	IP ← IP + disp8 - 2
CALL	seg:mem	IP ← mem; CS ← seg; SP ← SP -2	JCXZ	disp8	IP ← IP + disp8 - 2
INT			LOOP	disp8	IP ← IP + disp8 - 2
INTO			LOOPZ	disp8	IP ← IP + disp8 - 2
			LOOPE	disp8	IP ← IP + disp8 - 2
RET		IP ← [SP]; CS ← [SP+2]; SP ← SP + 2	LOOPNZ	disp8	IP ← IP + disp8 - 2
IRET		IP ← [SP]; CS ← [SP+2]; FLAGS ← [SP + 2]; SP ← SP + 4	LOOPNE	disp8	IP ← IP + disp8 - 2

String Manipulation

x86 Mnemonic	Operation	x86 Mnemonic	Operation
MOVSW	ES:DI ← DS:SI; CX ← CX - 1	MOVSB	ES:DI ← DS:SI; CX ← CX - 1
LODSW	AX ← DS:SI; CX ← CX - 1	LODSB	AL ← DS:SI; CX ← CX - 1
STOSW	ES:DI ← AX; CX ← CX - 1	STOSB	ES:DI ← AL; CX ← CX - 1
CMPSW	ES:DI - DS:SI; CX ← CX - 1	CMPSB	ES:DI - DS:SI; CX ← CX - 1
SCASW	AX - DS:SI; CX ← CX - 1	SCASB	AXL- DS:SI; CX ← CX - 1
REP	Repeat prefix		
REPE/REPZ	Repeat if equal/zero prefix	REPNE/REPNZ	Repeat if not equal/not zero prefix

Processor Control

x86 Mnemonic	Operation	x86 Mnemonic	Operation
CLC	CF ← 0	STC	CF ← 1
CMC	CF ← not CF		
CLD	DF ← 0	STD	DF ← 1
CLI	IF ← 0	STI	IF ← 1
HALT	Halt instruction execution	WAIT	Wait if busy line asserted
ESC	Escape to processor extension	LOCK	Lock instruction prefix

Notes:

1. **reg8** is an 8-bit register: AL, CL, DL, BL, AH, CH, DH or BH
2. **reg16** is a 16-bit register: AX, CX, DX, BX, SI, DI, BP or SP
3. **sreg** is a segment register: CS, DS, SS or ES
4. **imm8** is an 8-bit immediate value
5. **imm16** is a 16-bit immediate value
6. **addr8** is an 8-bit port address
7. **addr16** is a 16-bit memory address
8. **disp8** is an 8-bit offset to be branched to or 8-bit memory address displacement
9. **disp16** is a 16-bit offset to be branched to or 16-bit memory address displacement
10. **mem** is a memory pointer in any of the following memory addressing modes:
    [addr16], [disp16],
    [BX], [SI], [DI],
    [BX + disp8], [BP + disp8], [SI + disp8], [DI + disp8],
    [BX + disp16], [BP + disp16], [SI + disp16], [DI + disp16],
    [BX + SI], [BX + DI], [BP + SI], [BP + DI],
    [BX +SI + disp8], [BX + DI + disp8], [BP + SI + disp8], [BP + DI + disp8],
    [BX + SI + disp16], [BX + DI + disp16], [BP + SI + disp16], [BP + DI + disp16]
11. **seg** is a 16-bit segment selector value
12. **count** is either **1** or ***register CL to represent value in CL : number of times = remainder of*** $\frac{value\ in\ CL}{8}$

Appendix B: 8086 Instruction Construction
(Extract)

Mnemonic and Description	Instruction Code			
DATA TRANSFER				
MOV = Move:	**76543210**	**76543210**	**76543210**	**76543210**
Register/Memory to/from Register	100010dw	mod reg r/m		
Immediate to Register/Memory	1100011w	mod000r/m	data	data if w = 1
Immediate to Register	1011w reg	data	data if w = 1	
Memory to Accumulator	1010000w	addr-low	addr-high	
Accumulator to Memory	1010001w	addr-low	addr-high	
Register/Memory to Segment Register	10001110	mod0reg r/m		
Segment Register to Register/Memory	10001100	mod0reg r/m		
PUSH = Push:				
Register/Memory	11111111	mod110r/m		
Register	01010reg			
Segment Register	000reg110			
POP = Pop:				
Register/Memory	10001111	mod000r/m		
Register	01011reg			
Segment Register	000reg111			
XCHG = Exchange:				
Register/Memory with Register	1000011w	mod reg r/m		
Register with Accumulator	10010reg			
IN = Input from:				
Fixed Port	1110010w	port		
Variable Port	1110110w			
OUT = Output to:				
Fixed Port	1110011w	port		
Variable Port	1110111w			
XLAT = Translate Byte to AL	11010111			
LEA = Load EA to Register	10001101	mod reg r/m		
LDS = Load Pointer to DS	11000101	mod reg r/m		
LES = Load Pointer to ES	11000100	mod reg r/m		
LAHF = Load AH with Flags	10011111			
SAHF = Store AH into Flags	10011110			
PUSHF = Push Flags	10011100			
POPF = Pop Flags	10011101			

Mnemonic and Description	Instruction Code			
ARITHMETIC	**76543210**	**76543210**	**76543210**	**76543210**
ADD = Add:				
Reg./Memory with Register to Either	000000dw	mod reg r/m		
Immediate to Register/Memory	100000sw	mod000r/m	data	data if s: w = 01
Immediate to Accumulator	0000010w	data	data if w = 1	
ADC = Add with Carry:				
Reg./Memory with Register to Either	000100dw	mod reg r/m		
Immediate to Register/Memory	100000sw	mod010r/m	data	data if s: w = 01
Immediate to Accumulator	0001010w	data	data if w = 1	
INC = Increment:				
Register/Memory	1111111w	mod000r/m		
Register	01000reg			
AAA = ASCII Adjust for Add	00110111			
BAA = Decimal Adjust for Add	00100111			
SUB = Subtract:				
Reg./Memory and Register to Either	001010dw	mod reg r/m		
Immediate from Register/Memory	100000sw	mod101r/m	data	data if sw = 01
Immediate from Accumulator	0010110w	data	data if w = 1	
SSB = Subtract with Borrow				
Reg./Memory and Register to Either	000110dw	mod reg r/m		
Immediate from Register/Memory	100000sw	mod011r/m	data	data if sw = 01
Immediate from Accumulator	0001111w	data	data if w = 1	
DEC = Decrement:				
Register/memory	1111111w	mod001r/m		
Register	01001reg			
NEG = Change sign	1111011w	mod011r/m		
CMP = Compare:				
Register/Memory and Register	001110dw	mod reg r/m		
Immediate with Register/Memory	100000sw	mod111r/m	data	data if sw = 01
Immediate with Accumulator	0011110w	data	data if w = 1	
AAS = ASCII Adjust for Subtract	00111111			
DAS = Decimal Adjust for Subtract	00101111			
MUL = Multiply (Unsigned)	1111011w	mod100r/m		
IMUL = Integer Multiply (Signed)	1111011w	mod101r/m		
AAM = ASCII Adjust for Multiply	11010100	00001010		
DIV = Divide (Unsigned)	1111011w	mod110r/m		
IDIV = Integer Divide (Signed)	1111011w	mod111r/m		
AAD = ASCII Adjust for Divide	11010101	00001010		
CBW = Convert Byte to Word	10011000			
CWD = Convert Word to Double Word	10011001			

Mnemonic and Description	Instruction Code			
LOGIC	76543210	76543210	76543210	76543210
NOT = Invert	1111011w	mod010 r/m		
SHL/SAL = Shift Logical/Arithmetic Left	110100vw	mod100 r/m		
SHR = Shift Logical Right	110100vw	mod101 r/m		
SAR = Shift Arithmetic Right	110100vw	mod111 r/m		
ROL = Rotate Left	110100vw	mod000 r/m		
ROR = Rotate Right	110100vw	mod001 r/m		
RCL = Rotate Through Carry Flag Left	110100vw	mod010 r/m		
RCR = Rotate Through Carry Right	110100vw	mod011 r/m		
AND = And:				
Reg./Memory and Register to Either	001000dw	mod reg r/m		
Immediate to Register/Memory	1000000w	mod100 r/m	data	data if w = 1
Immediate to Accumulator	0010010w	data	data if w = 1	
TEST = And Function to Flags, No Result:				
Register/Memory and Register	1000010w	mod reg r/m		
Immediate Data and Register/Memory	1111011w	mod000 r/m	data	data if w = 1
Immediate Data and Accumulator	1010100w	data	data if w = 1	
OR = Or:				
Reg./Memory and Register to Either	000010dw	mod reg r/m		
Immediate to Register/Memory	1000000w	mod001 r/m	data	data if w = 1
Immediate to Accumulator	0000110w	data	data if w = 1	
XOR = Exclusive or:				
Reg./Memory and Register to Either	001100dw	mod reg r/m		
Immediate to Register/Memory	1000000w	mod110 r/m	data	data if w = 1
Immediate to Accumulator	0011010w	data	data if w = 1	
STRING MANIPULATION				
REP = Repeat	1111001z			
MOVS = Move Byte/Word	1010010w			
CMPS = Compare Byte/Word	1010011w			
SCAS = Scan Byte/Word	1010111w			
LODS = Load Byte/Wd to AL/AX	1010110w			
STOS = Stor Byte/Wd from AL/A	1010101w			
CONTROL TRANSFER				
CALL = Call:				
Direct within Segment	11101000	disp-low	disp-high	
Indirect within Segment	11111111	mod010 r/m		
Direct Intersegment	10011010	offset-low	offset-high	
		seg-low	seg-high	
Indirect Intersegment	11111111	mod011 r/m		

Encodings for the REG field

reg	For w = 0	For w = 1
000	AL	AX
001	CL	CX
010	DL	DX
011	BL	BX
100	AH	SP
101	CH	BP
110	DH	SI
111	BH	DI

Encodings for the R/M field

r/m	For mod = 00	For mod = 01	For mod = 10	For mod = 11 & w = 0	For mod = 11 & w = 1
000	[BX + SI]	[BX + SI + disp8]	[BX + SI + disp16]	AL	AX
001	[BX + DI]	[BX + DI + disp8]	[BX + DI + disp16]	CL	CX
010	[BP + SI]	[BP + SI + disp8]	[BP + SI + disp16]	DL	DX
011	[BP + DI]	[BP + DI + disp8]	[BP + DI + disp16]	BL	BX
100	[SI]	[SI + disp8]	[SI + disp16]	AH	SP
101	[DI]	[DI + disp8]	[DI + disp16]	CH	BP
110	[disp16]	[BP + disp8]	[BP + disp16]	DH	SI
111	[BX]	[BX + disp8]	[BX + disp16]	BH	DI

Notes:

- Disp8 or disp16 follows 2nd byte of instruction (before data if required).
- 16-bit displacement and data values are encoded in little endian style (reverse order of bytes)

- if d = 1 then destination of operation is specified by reg field; if d = 0 then destination is specified by r/m field
- if w = 1 then instruction operates on 16-bit values; if w = 0 then it operates on 8-bit values (bytes)

- if sw = 01 then 16 bits of immediate data form the operand; if sw = 11 then an immediate data byte is sign extended to form the 16-bit operand
- if v = 0 then ``count'' = 1; if v = 1 then ``count'' is reminder of dividing value in CL by 8
- z is used for string primitives for comparison with ZF FLAG
- x = don't care

- Segment override prefixes: ES – 26h; CS – 1Eh; SS – 36h; DS – 3Eh

Appendix C: ASCII Table

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	00	NUL	32	20	SPACE	64	40	@	96	60	`
1	01	SOH	33	21	!	65	41	A	97	61	a
2	02	STX	34	22	"	66	42	B	98	62	b
3	03	ETX	35	23	#	67	43	C	99	63	c
4	04	EOT	36	24	$	68	44	D	100	64	d
5	05	ENQ	37	25	%	69	45	E	101	65	e
6	06	ACK	38	26	&	70	46	F	102	66	f
7	07	BEL	39	27	'	71	47	G	103	67	g
8	08	BS	40	28	(	72	48	H	104	68	h
9	09	HT	41	29	)	73	49	I	105	69	i
10	0A	LF	42	2A	*	74	4A	J	106	6A	j
11	0B	VT	43	2B	+	75	4B	K	107	6B	k
12	0C	FF	44	2C	,	76	4C	L	108	6C	l
13	0D	CR	45	2D	-	77	4D	M	109	6D	m
14	0E	SO	46	2E	.	78	4E	N	110	6E	n
15	0F	SI	47	2F	/	79	4F	O	111	6F	o
16	10	DLE	48	30	0	80	50	P	112	70	p
17	11	DC1	49	31	1	81	51	Q	113	71	q
18	12	DC2	50	32	2	82	52	R	114	72	r
19	13	DC3	51	33	3	83	53	S	115	73	s
20	14	DC4	52	34	4	84	54	T	116	74	t
21	15	NAK	53	35	5	85	55	U	117	75	u
22	16	SYN	54	36	6	86	56	V	118	76	v
23	17	ETB	55	37	7	87	57	W	119	77	w
24	18	CAN	56	38	8	88	58	X	120	78	x
25	19	EM	57	39	9	89	59	Y	121	79	y
26	1A	SUB	58	3A	:	90	5A	Z	122	7A	z
27	1B	ESC	59	3B	;	91	5B	[	123	7B	{
28	1C	FS	60	3C	<	92	5C	\	124	7C	|
29	1D	GS	61	3D	=	93	5D	]	125	7D	}
30	1E	RS	62	3E	>	94	5E	^	126	7E	~
31	1F	US	63	3F	?	95	5F	_	127	7F	DEL