8086 Assembler in C

P - 101

SUBMITTED IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE AWARD OF THE DEGREE OF BACHELOR OF ENGINEERING IN COMPUTER TECHNOLOGY AND INFORMATICS ENGINEERING

KUMARAGURU COLLEGE OF TECHNOLOGY COIMBATORE-6 1984 CHARACTER IS LIFE

SUBMITTED BY

Elangovan .D
Saravanan .K .B
Sendhil Murugan .K

UNDER THE GUIDANCE OF

K. Mala, B.E(Hons)

Department of Computer Technology and Informatics Engineering
Kumaraguru College of Technology
Coimbatore-641 006

1989-90

Department of Computer Technology & Informatics Engineering

Kumaraguru College of Technology

Coimbatore-641 006

Certificate

This is to certify that the project entitled

8086 Assembler in C

has been submitted by

Mr. ....

In partial fulfilment for the Award of Bachelor of Engineering in the Computer Technology & Informatics Engineering Branch of the Bharathiar University, Coimbatore-641 006 During the Academic Year 1989-'90

[illegible]

....

Guide Head of the Department

Certified that the candidate was Examined by us in the project work Viva-Voce Examination held on .... and the University Register Number was ....

....

Internal Examiner External Examiner

Acknowledgement

ACKNOWLEDGEMENT

First of all, we wish to thank our Principal **Prof.R.Palanivelu,** B.E., M.Sc. for his support and encouragement for completion of this project.

With due respects, we wish to thank our Head of Department **Prof.P.Shanmugam,** M.S. (Hawaii) for his valuable guidance and facilities provided for the successful completion of the project.

We would like to express our sincere gratitude and indebtedness to **Miss. K.Mala,** B.E. (Hons.) faculty CTI Department for her thought provoking inspiration and guidance with-out whom, this project would not have been completed.

Also we would like to express our sincere thanks to **Miss.G.Andal Jayalakshmi,** B.E. for giving us her valuable ideas towards the successful completion of the project.

Last but not the least we would like to thank all the faculty members of the CTI department who had lent their helping hands for the completion of the project.

Contents

CONTENTS

Synopsis

SYNOPSIS

Assembler is a system software that converts Assembly instructions to machine code. Our project named AS86 is an Assembler for 8086 machine. Which is developed in 'C' language. 'C' language is chosen because of its capability of accessing the memory and registers as a low level language. AS86 generates opcodes for data transfer. Processor Control, Control transfer, logic and arithmetic instructions involving registers and memory.

AS86 is a 2 pass Assembler. In the first pass, symbol table is created. Symbol table consist of lable and the corresponding addresses. In second pass opcodes are generated for each instruction. The address displacement for control transfer instructions are calculating by referring the symbol Table the output of the form.

Address Label Mnemonics Opcode is stored in an output file will be which mentioned by the user. When he runs the AS86, the Input should be kept as a file containing the mnemonics (ie) the assembly language program. The running environment can be included by simply typing AS86.

We have followed modular programming approach in designing AS86. AS86 contains separate functions for each mnemonics type.

The flow control will be transferred appropriately by function calls.

The Modular Programming otherwise called, the structured programming is a "Goto-less" programming which is followed in designing AS86 specifically, because such structured programming approach is required for quick understanding of the flow of the program.

Introduction

CHAPTER - 1
INTRODUCTION

We, in our system software have concentrated on producing opcode which is the main process in an assembler. The language 'C' which is generally used for system software development has been used for the development of our package.

8086 : Processor

The 8086 is a 16 bit processor that is available as a 40 pins integrated circuit which has thirteen 16 bit registers and nine flags. They are

1. General registers.
2. Pointerand index registers.
3. Segment register.
4. Introduction pointer and flags.

The various addressing modes available in the 8086 processor are

1. Register Addressing
2. Immediate Addressing
3. Direct Addressing

4. Indirect Addressing through an index or a base register
5. Indirect addressing through the sum of a base and an Index Register.

'C' Language

'C' Lanugage was developed in the early seventies by programmers at Bell laboratories. 'C' language was designed by a small group of highly talented programmers. 'C' is a small language it is a structured language and it can be used to construct programs that execute very rapidly.

8086 Assembler in 'C'

The system software has been developed using structure and file operation. It is a two pass assembler in which mnemonics are stored in a structure (i.e.) mnemonics table. Input file is checked to find whether it is a label instruction (or) ordinary instruction if it is a label instruction pass. 1. functions is called else the instruction is stored in temp file. In Pass I symbol table is created.

In pass 2 temporary file is opened and the mnemonics are compared with mnemonics table, if they match then the corresponding function is called and the opcode is generated. The opcode address generated are stored in the output file.

Assembler

CHAPTER - 2

ASSEMBLER

2.1 INTRODUCTION

At one time programmers found it difficult to write or read programs in a machine language. In their output for a more convenient language they began to use a mnemonic for each machine instructions, which they would subsequently translate into machine language such a mnemonic machine language is now called an assembly language. Programs known as assemblers were written to automate the translation of assembly language into machine language. The input to an assembler program is source program; the output is a machine language translations (Object program).

2.2 GENERAL DESIGN PROCEDURE

Before going in for the detailed design of an assembler let us examine the general problem of designing of software. The six main steps to be followed in the design procedure are given below.

1. Specify the problem.
2. Specify data structures.
3. Define format of data structures.

4. Specify algorithm.
5. Look for modularity.
6. Repeat 1 through 5 on modular.

2.3 TWO PASS ASSEMBLER

In assembly language program symbols can appear before they are defined so it is convenient to make two passes over the input. The first pass has only to define the symbols; the second pass can generate instructions and addresses.

2.3.1 Functions of an Assembler

An Assembler must do the following

1. Generate instructions.
   a. Evaluate the mnemonic in the operation field to produce its machine code.
   b. Evaluate the subfields-find the value of each symbol process literals.
2. Process pseudo ops.

We can group these tasks into two passes or sequential scans over the input.

Pass 1

1. Determine length of machine instructions (MOTGETI)
2. Keep track of location counter (LC)
3. Remember values of symbols until pass 2 (STSTO)
4. Process some pseudo ops.
5. Remember literals.

Pass 2

1. Look up value of symbols (STGET)
2. Generate instructions
3. Generate data
4. Process Pseudo ops

The step involved in pass 1 and pass 2 are outlined in fig. 2.1 and fig 2.2.

2.3.2 Data Structures

The second step in our design procedure is to establish data bases that we have to work with

1. Input source program.

2. A location counter (LC) used to keep track of each instructions location.

3. A table, the machine operation table MOT, that indicates the symbolic mnemonic for each instruction and its length.

4. A table, the pseudo operation table (POT) that indicate the symbolic mnemonic and action to be taken for each pseudo-op in pass 1.

5. A table the symbol (ST) that is used to store each label and it is corresponding values.

6. A table, the literal table (LT) that is used to store each literal encountered and its corresponding assigned location.

7. A copy of the input to be used later by pass 2.

Pass 2 Data bases

Many of the databases used in pass 1 are common to pass 2. Those of them not found in pass 1 but used in pass 2 are given below.

1. Copy of source program input to pass 1.
2. A base table, that indicates which registers are currently specificed as base register by using pseudo-ops and what are the specified contents of these registers.
3. A work-space, INST, that is used to hold each instruction as it various parts are being assembled together.
4. A work-space, PRINTLINE, used to produce a printed listing.
5. A work space, pouch card used prior to actual out putting for converting the assembled instruction into format needed by the loader.
6. An output deck of assembled instruction in format needed by loader.

2.3.3 Format of Data bases

The third step in our design procedure is to specify the format and contents of each of data base.

The Pseudo operation table format has been shown in Fig. (2.3) in which each Pseudo-op is listed with an associated pointer to the assembler routine for processing Pseudo-op similarly other formats are specified.

2.3.4 Algorithm

The purpose of first pass is to assign a location to each instruction and data defining Pseudo-instruction and then to define values for symbols appearing in the lable fields of the source program.

After all the symbols have been defined by pass 1 it is possible to finish the assembly by processing each card and determining value for its operation code, and its operand field. In addition, pass 2 must structure the generated code into the appropriate format for later processing by loader, and print and assembly listing containing the original source and hexadecimal equivalent of the bytes generated.

2.3.5 Look for modularity

We now review our design looking for function that can be isolated. Typically such functions full into two categories (1) multi-use and (2) unique likely choices for modularity are selected and listed each of these likely choice should go through the entire design process.

8086 Processor

CHAPTER - 3
8086 PROCESSOR

3.1 ORGANIZATION OF THE 8086

8086 is a 16 bit processor that is available as a 40 pin integrated circuit. Three main organization in 8086 are

* Memory organization.
* Register Structure.
* Addressing Modes.

which has been detaited disqussed in the proceeding section.

3.2 MEMORY ORGANIZATION

The 8086 provides 20 lines on which address and data is multiplexed. The organization is shown in fig. 3.1. The AD0-AD15 are lines are used for transmitting 16 least significant bits of the 20 bit address. The remaining four most significants bits are transmitted over the A16-A19 lines.

One word of data, 16 bits is sent or received over the AD0-AD15 lines. The least significant byte of the word is transmitted on AD0-AD7 and the most significant byte on AD8-

AD15. When only a byte of data is transferred and not a complete word, then either AD0-AD7 or AD8-AD15 are used depending on which the byte corresponds to even or odd memory respectively so the largest address is $(2^{20}-1)$ which implies that a total of one million bytes can be addressed directly by 8086.

Addressing bytes and words

Most instructions of the 8086 operate both on byte and word operands. It is possible to move contents of the byte at location 02 to the AL register. It is also possible to move the contents of the word starting at location 02 to the AX Register.

Physically the memory can be viewed to be organized as two banks of 512 bytes each. The D0-D7 lines being used for data transfer involving the low bank and D8-D15 for high bank and there is involved data address transfer also so the A0 and BHE lines are used for determining one of these possible reference to memory which is shown in Table 3.1.

3.3 REGISTER STRUCTURE

The 8086 has thirteen 16 bit register and nine flage. The registers can be classified.

1. General Registers.
2. Pointer and Index Registers.
3. Segment Registers.
4. Instructions Pointer and Flags.

General Registers

The four general registers are used as operand in arithmetic and logic operation whose structure are as shown in fig. 3.2.

These registers can also be used as scartch pad while evaluating arithmetic expressions.

Pointers and Index Registers

These 16 bit registers serve to hold offsets of operands residing in the memory. The organization is shown in fig. 3.3 SP acts as stack points as in 8085. It points to the current top of the stack.

The BP register can be used for holding the offset of the base of a data area in the current stack segment. This enables accesses to variables within a procedure to be specified conveniently.

There are certain string operations provided by 8086. Typical amongst them is the one to move a string of bytes or words from one area to another. The SI and DI are used implicitly in such instructions.

Segment Registers

The 8086 memory is logically divided into several segments. In all there are four types of segments.

1. Code Segment
2. Data Segment
3. Stack Segment
4. Extra Segment.

Each segment can be at most 64 K bytes long. The starting address of currently active segments in contained in the segment registers. The CS, DS and ES contains respectively start of code, data stack and extra segments. The figure 3.4 illustrates segmentations.

The figure 3.4 shows four currently active code, data, stack and extra segments as pionted by the segment registers. Program instructions reside in the code segment, Data area of the program variables and constants may reside in the data segment. Any stack used resides in the stack segment some data like string operated on by special instructions reside in the extra segment.

Instruction pointer and flags

The instruction pointer is a 16 bit register and points to the next instruction to be executed with in the current code segment it is automatically updated by the 8086 as program execution proceeds.

There are nine flags available as shown in figure 3.5. Based on the use of these flags it is possible to categorise them into states and control flags. Status flag such as CF (Carry) ZF (Zero) are generaly set or (reset) after an arithmetic or logical instructions in executed control flags such as DF (Direction Flag). If (Interrupt enable) and TF(Trace Flag). Control the execution of the 8086. The TF flag forces the 8086 to generate an interrupt after execution of each instruction.

3.4 ADDRESSING MODES IN 8086

Several powerful addressing modes are available for specifying the operands in an instruction there are

1. Register addressing
2. Immediate addressing
3. Direct addressing
4. Indirect addressing through an index or a base register

Indirect addressing through the sum of a base and an index register operands in memory can be specified using any of the addressing modes mentioned above.

Register Addressing

This is the simplest of all addressing mode for example.

MOV AL, BL	:	Move contents of BL to AL
INC SI	:	Increment SI content by 1
ADD AX, BX	:	Add BX to AX and result is in AX

Immediate Addressing

Is as similar in 8085 examples of instructions using this mode are given below.

MOV AL, OF AH	:	Move OFAH to AL
ADD BX,4	:	Add 4 to BX

Direct Addressing

The direct addressing mode is available in the 8086 a few of which are given below.

MOV AL, X	:	Move the 8 bit value of X of AL
MUL Y	:	Multiply contents of AX by the 16 bit value of Y

Indirect Addressing through Base or Index Register

It is possible to indirectly address is memory operand using any of the following four ways.

* through an index register
* through a base register
* through an index register summed up with an 8 or 16 bit offset
* through a base register summed up with an 8 or 16 bit offset.

Indirect Addressing through Sum of Base and Index Register

It is possible to form an indirect address in the 8086 using following modes.

* sum of base and an index register
* sum of base, index register and displaement.

3.5 BUS STRUCTURE

Bus Structure and Timing of 8086

In this section we shall discuss about the address data and control bus structure and timings for the 8086.

The signals available are as shown in fig. 3.6. several function depend on MN/$\overline{\text{MX}}$. When MN/$\overline{\text{MX}}$ pin tied to VCC the 8086 is in minimum mode and when it is grounded in the 8086 is in the maximum mode.

Bus Interface and Execution Unit

The 8086 consists of two units which interact with each other to perform various functions. The bus interface unit (BIU) is responsible for transferring instruction and data, bytes between the processor and memory or I/O Devices. The other unit named execution unit perform instruction decoding and execution. The BIU fetches successive instruction bytes from the memory and inserts them into a 6 byte instruction queue. The EU remove instructions from this Queue, decodes and execute them. The BIU fetches the instruction bytes independent of the EU. In fact whenever the bus is idle and there is space for atleast 2 bytes in the instruction Queue, the BIU would fetch 2 bytes pointed at by IP from the current code segment.

Using BIU and EU to perform bus operations and instruction executions independend of each other the 8086 is able to achieve a high degree of overlap between instruction execution and fetching. This architectural features of the 8086 is known as instruction pipeline. Pipelining is used to improve performance of the µp.

3.6. OPCODE GENERATION

Opcodes are coded manually in 8086 which has a specific format to calculate opcode.

The format to calculate opcode is show in fig. 3.7. the coding could be explained with an example.

Coding MOV, SP, BX

This instruction will copy a word from the BX register to sp register. Consulting the instruction set in Appendix A. You find that the 6 bit opcode for this instruction 'is 100010. Make the W bit a 1 because you are moving a word. The D bit for this instruction may be some what confusing however since two registers are involved you can think of the move as to sp, or from BX. If you think of the instruction as moving a word to sp, then make the D bit as 1 and put 100 in the REG field to represent sp. The mod field will be 11 to represent register addressing mode. Make the R/M field 011 to represent other register, BX. The resultant code for the instruction MOV SP, BX will be 1000101111100011. Figure 3.8 shows the meaning of all these bits.

‘C’ Language

CHAPTER - 4

'C' LANGUAGE

4.1 ABOUT 'C'

'C' is a general purpose programming language. It has been closely associated with the UNIX system, since it was developed on that system and since UNIX and its soft ware are written in 'C'. The language however is not tied to any one operating system or machine and although it has been called a "system programming language" because it is useful fcr writing operating system it has been equally well to write major numerical text processing and data-base programs.

'C' language was developed in the early seventies by an independent hand ful of programmes at Bell laboratories 'C' language was designed by a small group of highly talented programmers. It was designed as a short hand form of assembly language that offered a high degree of portability of expression.

'C' mainly deal with the same sort of objects that most computers do, namely characters, numbers, and addresses. These may be combined and moved about with the usual arithmetic and logical operations implemented by actual machines.

Similarly 'C' offers straight forward single thread control flow construction : tests, loops, grouping and subprograms .

'C' is a small language, it is a structured language and it can be used to construct programs that execute very rapidly. It has small number of statements and a compact set of functions would be early grasped.

'C' offers you access to machine level with a rare blend of efficiency and elegance 'C' is undoubtedly the language for most system programers.

4.2 'C' FOR SYSTEM SOFTWARE DEVELOPMENT

There are humpty number of reasons for choosing 'C' for software development of which some are listed below.

'C' language is used mainly in commercial quality software development. Its portability, compactcode fast execution time and expressiveness are its strong points. It is used to address many different operating environment. 'C' language compiler offer a high degree of option that may be called upon to address many different operating situation.

Also because the data types and control structure provided by 'C' are supported directly by most existing computers, the run-time library required to implement self-contained program is tiny.

Again because the language reflects the capabilities of current computer. 'C' programs tend to be efficient enough that there is no compulsion to write assemble language instead.

Also in 'C' the fundamental data objects are characters, integers of several sizes, and floating point numbers. In addition there is a hierarchy of derived data types created with pointer arrays structure, unions and functions.

'C' provides the fundamental flow-control constructions required for will structured programs. Statement grouping decision making (if) looping with the termination test at the top (while, for) or at the bottom (do) and selecting one of a set of possible cases (switch).

'C' provides pointer and the ability to do address arithmetic. Any function may be called recursively and its local variables are typically automatic or created a new with each invocation.

'C' also includes pascal when the emphasis on clearly structuring programs the programs in 'C' tend to be better much more understandable and therefore requires 'C' much less time to develop then the program written in earlier language such as COBOL and FORTRAN.

A Sample 'C' Program is an follows

```
While (True);
Puts ("Input File Name");
Gets ("Filename");
If open (Filename, 0) = ERRoR))
Continue;
ELSE
Break;
```

Since this language includes the facilities that are very close to machine language we can get fairly close to the machine which helps tremendously in making programs more efficeint when it comes to execution speed. In a sense we get the best both words, namely a high level language and machine language.

For these reasons 'C' has become the execeeding popular language today many compiler and themselves written in 'C' and so are many operation system such as UNIX.

Since the language 'C' speech up both development and execution time our assembler is written in 'C'.

Software Development

CHAPTER - 5
SOFTWARE DEVELOPMENT

This chapter outlines the development of software and gives a detailed algorithm. It has 2 sections which are

1. Software Outline.
2. Flow Chart.

5.1 SOFTWARE OUTLINE

This software includes files, pointers, structures and tables etc. For input, output and intermediate operations files are used. Each file operation includes a pointer in it. Structures are used here to store symbols and its corresponding addresses. Tables are used for searching the particular instructions etc.

First the program gets instruction from the input file and checks its fifth character to see whether it is a label instruction. If the fifth character of the instruction is going to be a ':' then it decides it is a label instruction. Then it calls the pass 1 function to check it is a correct Label and finds its corresponding address by calling the symbol function and then the rest of the instruction is stored in the temporary file. If the fifth character is not an ':' character then the instruction is stored in the temporary file. Then pass 2 function is called.

Here the pass 1 function finds all the Labels and checks whether it is a correct label or not. The first character of a Label should be an alphabet. Rest of the characters may be a character (alphabet or digit) but it should not be an special character. Once the correct label is found then the corresponding instructions address also calculated by calling symbol function. All the Lables in the input file and its addresses are stored in the structure.

The function of pass 2 is to generate opcodes for each instruction which appear in the input file. Temporary file is the input file for pass 2. It takes only one instruction at a time and matches with the instructions stored in the label. Once there is a match found then it calls the corresponding function which will give the corresponding opcode for that particular instruction. This is stored in the output file.

In the case of unconditional Jump instructions, the Label is found and it is searched in the structure for matching. Once there is a match then the actual address is compared with the label address. If actual address is greater than the label address then the displacement is found and the two's complement for the displacement is placed at the jump instruction. Else the displacement is placed.

The format in the output file is ADDRESS LABEL MNEMONICS OPCODE. The o/p file can be read through an Editor. The opcodes generated can be run on the machine by creating the running environment.

5.2 FLOW CHART

MAIN

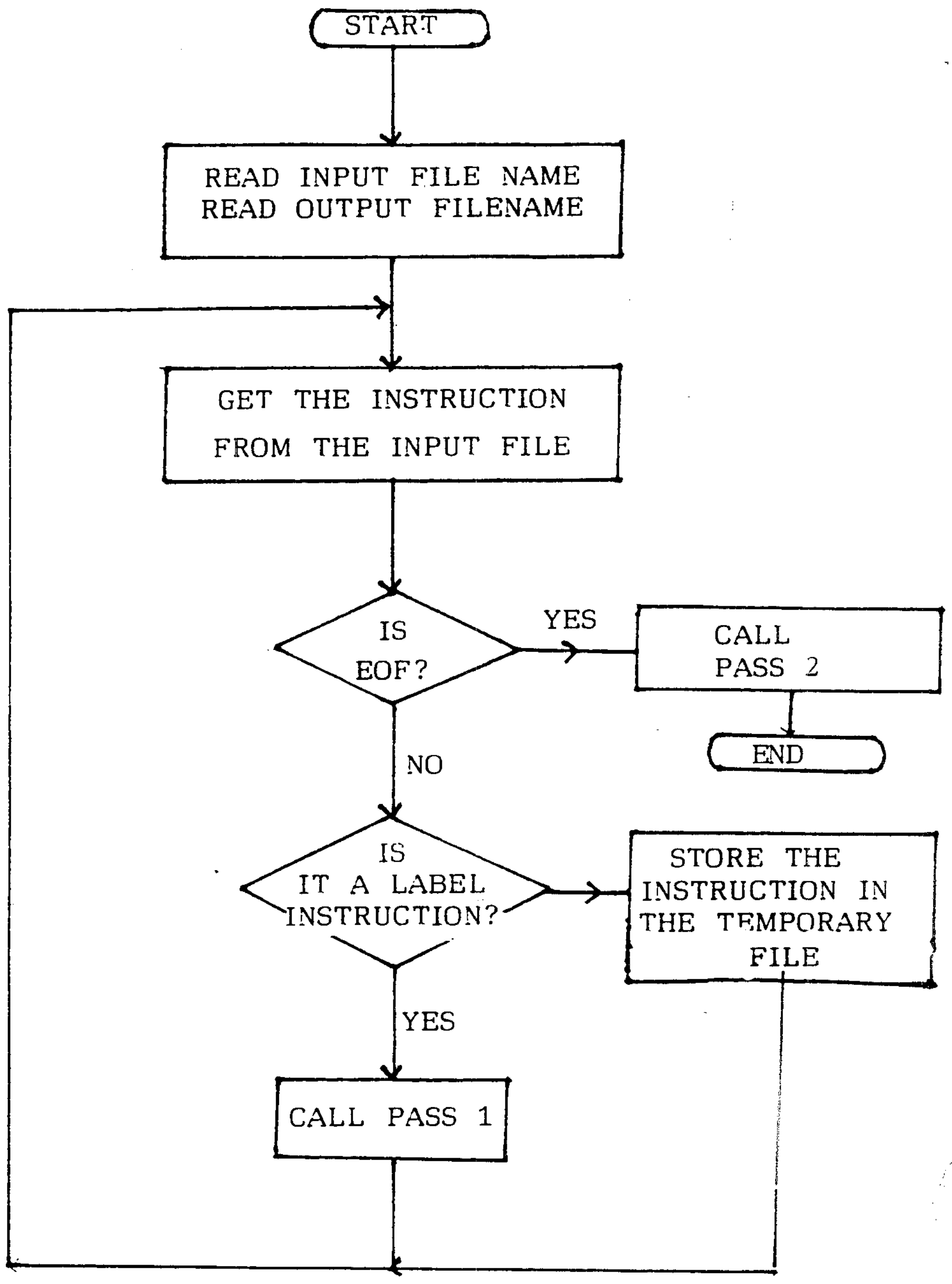


FIG. 5.1.

This software includes 4 major flow charts with it. Which are shown in the following diagrams.

PASS 1 FLOWCHART

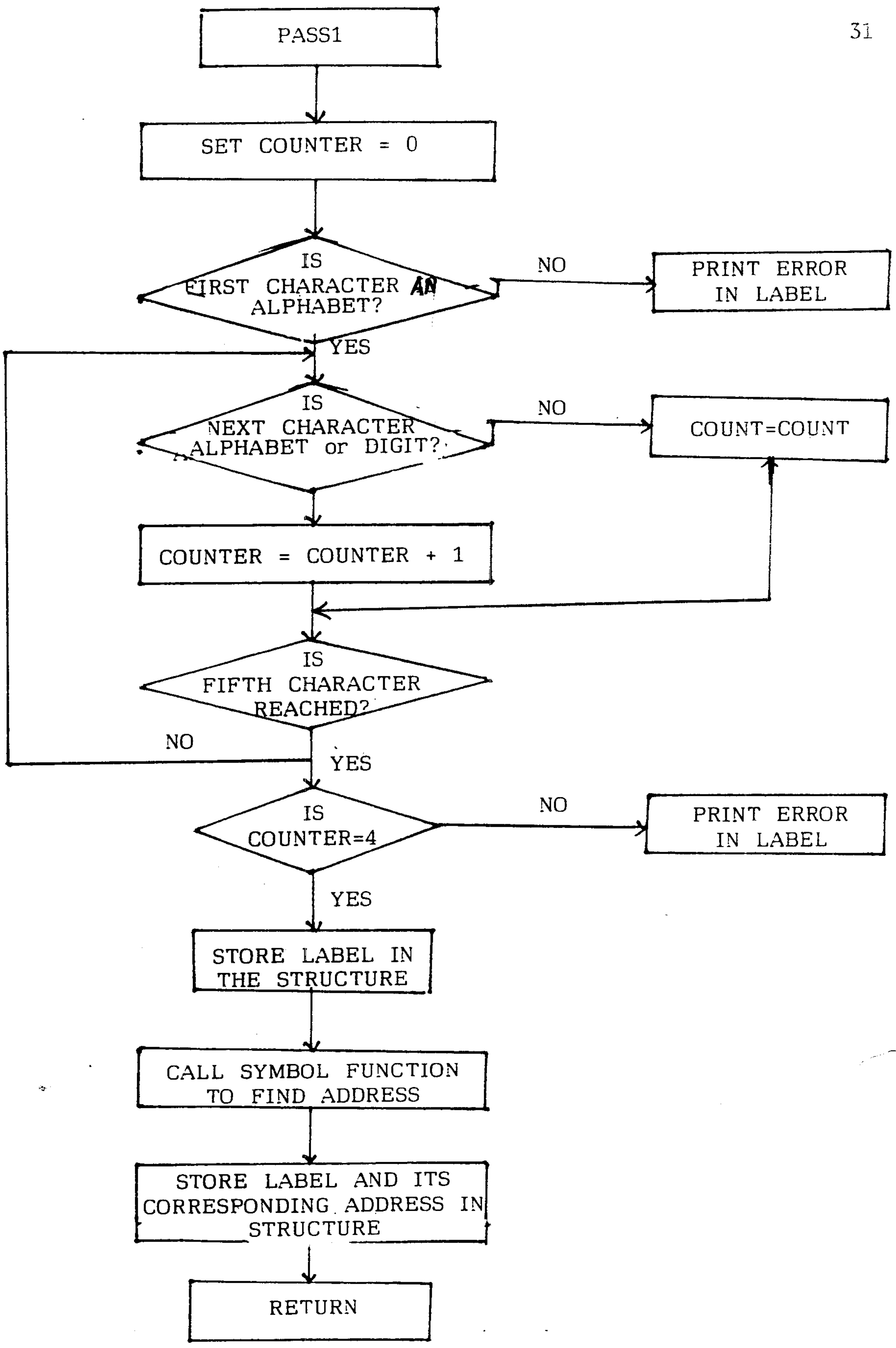

PASS1
SET COUNTER = 0
IS FIRST CHARACTER AN ALPHABET?
NO
PRINT ERROR IN LABEL
YES
IS NEXT CHARACTER ALPHABET or DIGIT?
NO
COUNT=COUNT
COUNTER = COUNTER + 1
IS FIFTH CHARACTER REACHED?
NO
YES
IS COUNTER=4
NO
PRINT ERROR IN LABEL
YES
STORE LABEL IN THE STRUCTURE
CALL SYMBOL FUNCTION TO FIND ADDRESS
STORE LABEL AND ITS CORRESPONDING ADDRESS IN STRUCTURE
RETURN

FIG. 5.2.

PASS 2 FLOW CHART

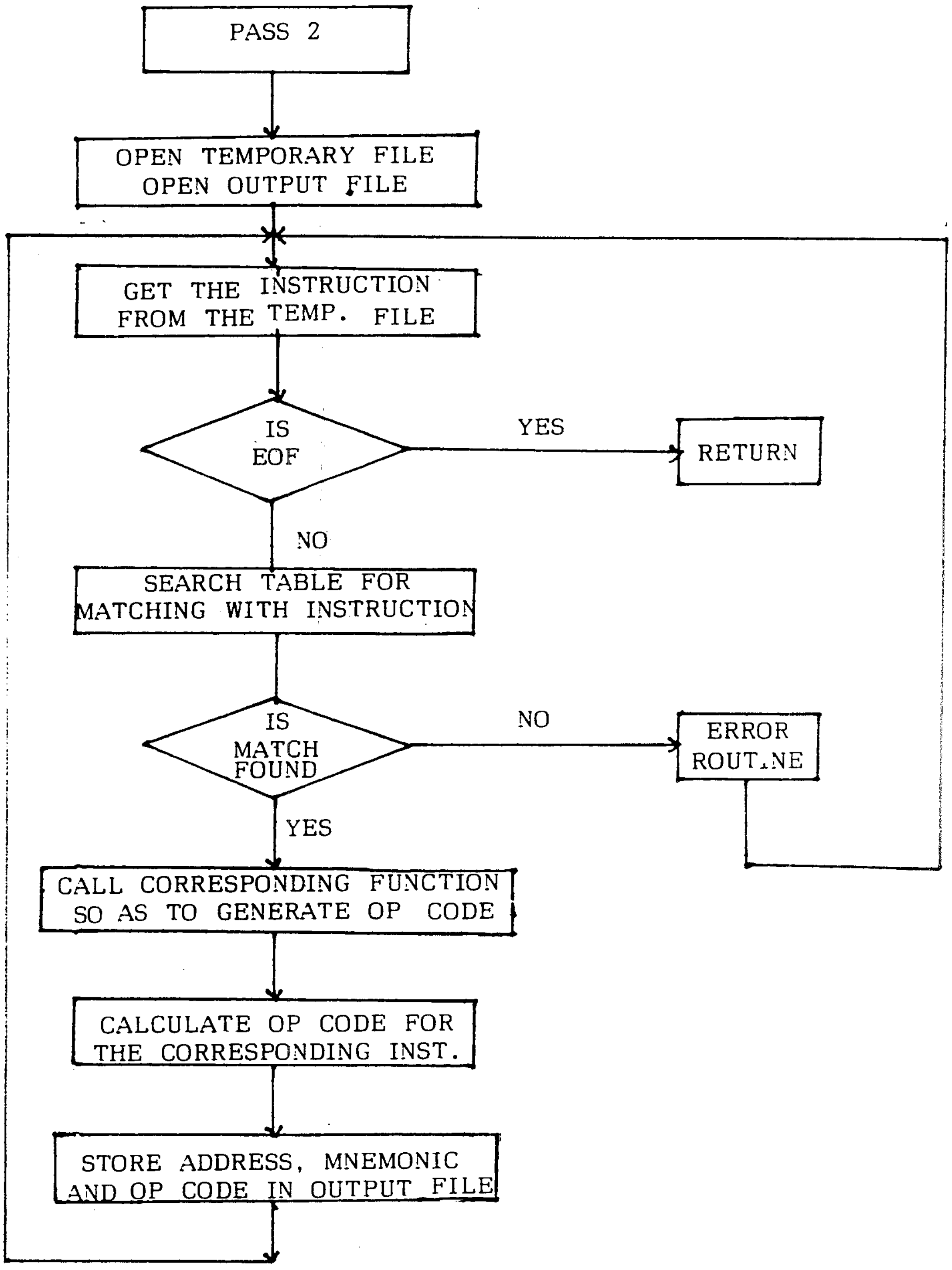

PASS 2
OPEN TEMPORARY FILE
OPEN OUTPUT FILE
GET THE INSTRUCTION FROM THE TEMP. FILE
IS EOF
YES
RETURN
NO
SEARCH TABLE FOR MATCHING WITH INSTRUCTION
IS MATCH FOUND
NO
ERROR ROUTINE
YES
CALL CORRESPONDING FUNCTION SO AS TO GENERATE OP CODE
CALCULATE OP CODE FOR THE CORRESPONDING INST.
STORE ADDRESS, MNEMONIC AND OP CODE IN OUTPUT FILE

FIG. 5.3

JUMP FLOWCHART

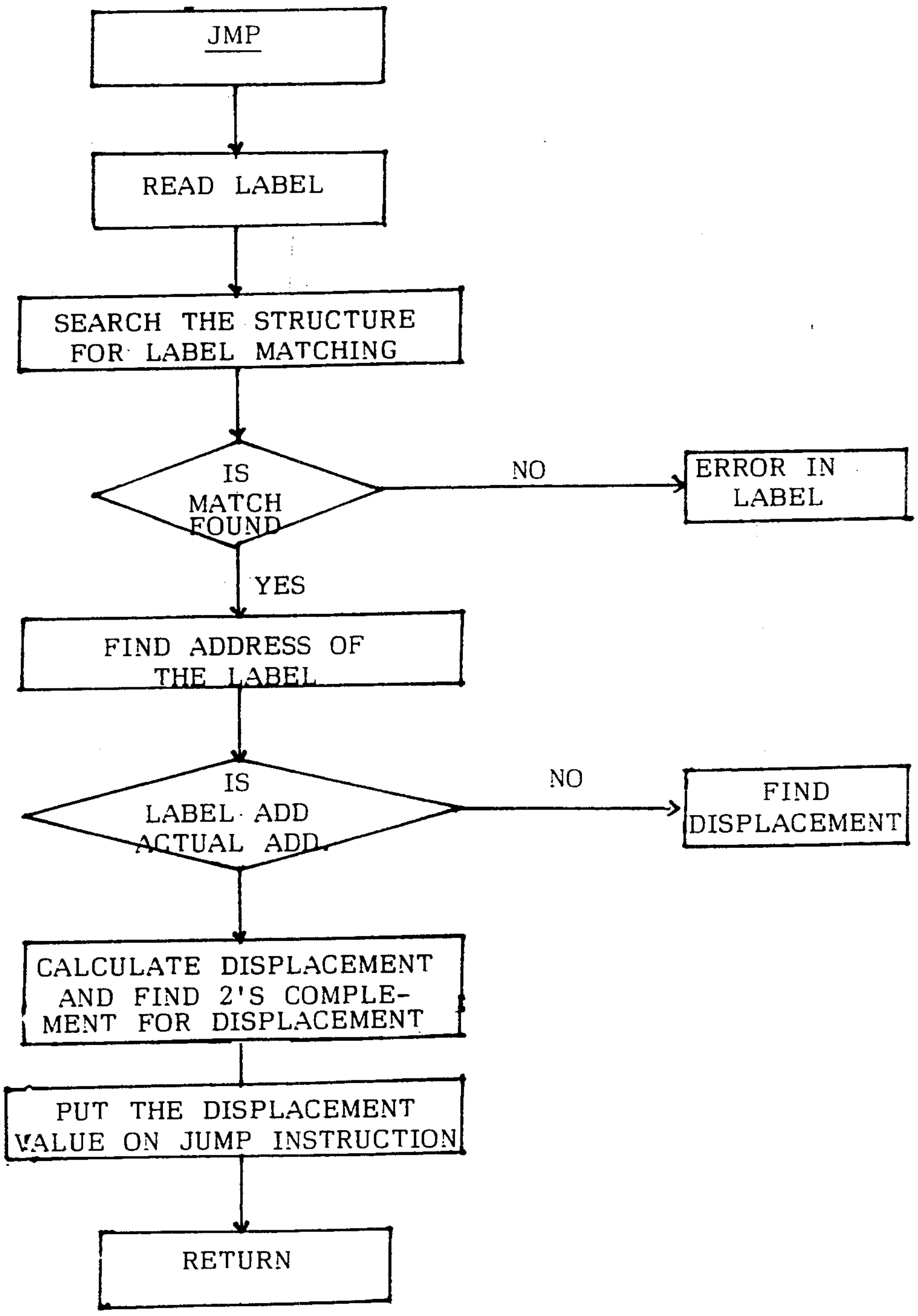

JMP
READ LABEL
SEARCH THE STRUCTURE FOR LABEL MATCHING
IS MATCH FOUND
NO
ERROR IN LABEL
YES
FIND ADDRESS OF THE LABEL
IS LABEL ADD ACTUAL ADD.
NO
FIND DISPLACEMENT
CALCULATE DISPLACEMENT AND FIND 2'S COMPLEMENT FOR DISPLACEMENT
PUT THE DISPLACEMENT VALUE ON JUMP INSTRUCTION
RETURN

FIG. 5.4

5.3 TRIAL RUN

5.3.1 INPUT

Input File-1

```
mov ax,1000
mov bx,1000
add ax,bx
mov dx,ax
add dx,2000
mul ax
xor cx,dx
test dx,bx
cmp ax,8000
and bx,dx
stc
sub ax,dx
sbb ax,0060
inc ax
adc bx,0400
cmc
clc
or ax,bx
hlt
```

5

Input File-2

```
clc
mov ax,1000
mov bx,ax
mul bx
wait
aam
loop1:add ax,bx
or ax,bx
mov sp,bx
mov si,bx
dec ax
jmp loop1
hlt
```

Input File-3

```
mov dx.0050
mov bp,1000
sub bp,dx
mov ax,0020
jmp loop2
dec ax
sbb ax,dx
das
loop2:test ax,0010
or ax,bp
push ax
pop ax
hlt
```

5.3.2 OUTPUT

Output File-1

```
*************************************************************
  ADDRESS    LABEL    MNEMONICS                OPCODE
*************************************************************

   1000               mov ax,1000              C7    0     10   00
   1004               mov bx,1000              C7    3     10   00
   1008               add ax,bx                 3   C3
   100A               mov dx,ax                8B   D0
   100C               add dx,2000              83    2     20   00
   1010               mul ax                   F7   40
   1012               xor cx,dx                8B   CA
   1014               test dx,bx               85   D3
   1016               cmp ax,8000              83   38     80   00
   101A               and bx,dx                23   DA
   101C               stc                      F9
   101D               sub ax,dx                2B   C2
   101F               sbb ax,0060              83   18     00   60
   1023               inc ax                   40
   1024               adc bx,0400              83   13     04   00
   1028               cmc                      FF
   1029               clc                      F8
   102A               or ax,bx                  B   C3
   102C               hlt                      74
```

Output File-2

```
*************************************************************
  ADDRESS     LABEL     MNEMONICS                  OPCODE
*************************************************************

   1000                   clc                       F8
   1001                   mov ax,1000               C7     0      10   00
   1005                   mov bx,ax                 8B    D8
   1007                   mul bx                    F7    43
   1009                   wait                      FA
   100A                   aam                       FD
   100B       loop1       add ax,bx                  3    C3
   100D                   or ax,bx                   B    C3
   100F                   mov sp,bx                 8B    E3
   1011                   mov si,bx                 8B    F3
   1013                   dec ax                    48
   1014                   jmp loop1                 D9    F4
   1016                   hlt                       74
```

Output File-3

```
*************************************************************
  ADDRESS     LABEL     MNEMONICS                   OPCODE
*************************************************************

   1000                   mov dx.0050               C7    2     00   50
   1004                   mov bp,1000               C7    5     10   00
   1008                   sub bp,dx                 2B   EA
   100A                   mov ax,0020               C7    0     00   20
   100E                   jmp loop2                 D9    0
   1010                   dec ax                    48
   1011                   sbb ax,dx                 1B   C2
   1013                   das                       FC
   1014       loop2       test ax,0010              F7    0     00   10
   1018                   or ax,bp                   B   C5
   101A                   push ax                   50
   101B                   pop ax                    58
   101C                   hlt                       74
```

Conclusion

CHAPTER - 6
CONCLUSION

The Assembler package was developed in the DOS Environment. The package named AS86 works successfully for a certain set of instructions.

AS86 has been developed to cover all instructions in the 8086 processor. We have included all instructions involving register to register and memory instruction.

Structures and file pointers have been used to speed up the generation of opcode. The user just specifies the input program in an input file (file name of his choice). The AS86 does the remaining and gives a fulformatted output in the output file.

Control transfer instructions have been included in the software ie., forward and backward looping instructions like jump in the II pass.

The software aimed at generating opcode for all instructions has been done and provision been made to include other leftout instruction.

Appendix

A. PROGRAM LISTING

```
#include <graphics.h>
#include <stdlib.h>
#define ESC 0x1b
char c;
int col,l=0,k;
struct viewporttype vp;
int maxx,maxy;
main()
{
  int GraphDriver = CGAC0,GraphMode;

  initgraph(&GraphDriver,&GraphMode,"");
  maxx = getmaxx();
  maxy = getmaxy();

  setviewport(0,0,maxx,maxy,1);
  getviewsettings(&vp);
  setcolor(3);
  setbkcolor(LIGHTCYAN);
  rectangle(0,0,vp.right-vp.left,vp.bottom-vp.top);

  for(k=0;k<=80;k=k+5)  {
  rectangle(5+k,5+k,vp.right-vp.left-k,vp.bottom-vp.top-k);
  delay(200); }
   settextstyle(GOTHIC_FONT,HORIZ_DIR,3);
  outtextxy(160,90," s y s t e m   s o f t w a r e");
  delay(10000);
  cleardevice();
  for(k=0;k<=10000;k++){
  putpixel(random(maxx),random(maxy),1); delay(1); }
  delay(1000);
  cleardevice();
  setbkcolor(YELLOW);
  setfillstyle(10,1);
  settextstyle(TRIPLEX_FONT,HORIZ_DIR,2);
  bar(0,0,vp.right-vp.left,vp.bottom-vp.top-30);
  outtextxy(140,35,"WELCOME TO THE 8086 ASSEMBLER DEMO");
  settextstyle(DEFAULT_FONT,HORIZ_DIR,1);
  delay(500);
  outtextxy(maxx/2,maxy-110,"BY");
  delay(100);
  outtextxy(maxx/2,maxy-100,"ELANGOVAN");delay(500);
  outtextxy(maxx/2,maxy-90,"SARAVANAN");delay(500);
  outtextxy(maxx/2,maxy-80,"SENDHIL MURUGAN");delay(500);
  outtextxy(maxx/2,maxy-30,"Esc  Aborts,   Press a Key  to
  cont.....");
   c=getch();

   escfn(1);
          outtextxy(200,10,"INSTRUCTION FORMAT");
          delay(1000);
          outtextxy(20,30,"MOV reg,reg"); delay(300);
          outtextxy(20,40,"MOV reg,memory"); delay(300);
          outtextxy(20,50,"ADD reg,reg"); delay(300);
          outtextxy(20,60,"ADD reg,memory"); delay(300);
          outtextxy(20,70,"SUB reg,reg");  delay(300);
          outtextxy(20,80,"SUB reg,memory"); delay(300);
          outtextxy(20,90,"ADC reg,reg");    delay(300);
          outtextxy(20,100,"ADC reg,memory");delay(300);
          outtextxy(20,110,"SBB reg,reg");   delay(300);
          outtextxy(20,120,"SBB reg,memory");delay(300);
          outtextxy(210,30,"AND reg,reg");  delay(300);
          outtextxy(210,40,"AND reg,memory"); delay(300);
          outtextxy(210,50,"CMP reg,reg");    delay(300);
          outtextxy(210,60,"CMP reg,memory"); delay(300);
          outtextxy(210,70,"TEST reg,reg");   delay(300);
          outtextxy(210,80,"TEST reg,memory");delay(300);
          outtextxy(210,90,"XOR reg,reg");    delay(300);
          outtextxy(210,100,"XOR reg,memory");delay(300);
          outtextxy(210,110,"OR reg,reg");    delay(300);
          outtextxy(210,120,"OR reg,memory"); delay(300);
          outtextxy(420,30,"PUSH reg");       delay(300);
          outtextxy(420,40,"POP reg");      delay(300);
          outtextxy(420,50,"INC reg");      delay(300);
          outtextxy(420,60,"DEC reg");      delay(300);
          outtextxy(420,70,"HLT");          delay(300);
          outtextxy(420,80,"JMP label"); delay(300);
          outtextxy(420,90,"MUL reg");delay(300);
          outtextxy(420,100,"CLC");delay(300);
          outtextxy(420,110,"CMC");delay(300);
          outtextxy(420,120,"STC");delay(300);
          outtextxy(520,60,"WAIT");delay(300);
          outtextxy(520,70,"AAS");delay(300);
          outtextxy(520,80,"DAS");delay(300);
          outtextxy(520,90,"AAM");delay(300);
        c=getch();
 closegraph();

}

escfn(col)
{
if(ESC == c)
{
 closegraph();
 exit(1);
 }
 else
 {
   cleardevice();
   setcolor(1);
   if(col == 1)
   setbkcolor(GREEN);
   else if(col == 2)
   setbkcolor(BROWN);
   else   { setbkcolor(CYAN);
           setfillstyle(0,1);   }
   settextstyle(DEFAULT_FONT,HORIZ_DIR,1);
```

```
      bar(0,0,vp.right-vp.left,vp.bottom-vp.top-30);
      outtextxy(maxx/2,maxy-30,"Esc  Aborts,  Press a Key to
Cont....");

    }
}
```

```
#include <stdio.h>
#include <ctype.h>
int i,j,s,w,op[10],e,q1,v,k;
int ax,cx,dx,bx,sp,bp,si,di,o1,o2,addr=0x1000;
char *p,*p1,infile[30],outfile[30],m[80],n[80],c;
struct label
        {
          char name[10];
          int locat;    }   lab[15];
struct mne
      {
          char mnemo[80];
          int index;
          int bit;
      };
struct mne tab[] = {
                        "mov ",1,4,
                        "add ",2,4,
                        "sub ",3,4,
                        "cmp ",4,4,
                        "test ",5,5,
                        "inc ",6,4,
                        "dec ",7,4,
                        "push ",8,5,
                        "pop ",9,4,
                        "xor ",10,4,
                        "or ",11,3,
                        "and ",12,4,
                        "sbb ",13,4,
                        "adc ",14,4,
                        "jmp ",15,4,
                        "mul ",16,4,
                        "clc",17,3,
                        "cmc",18,3,
                        "stc",19,3,
                        "wait",20,4,
                        "aas",21,3,
                        "das",22,3,
                        "aam",23,3,
                        "hlt",24,3
                                     };
FILE *fp,*fi,*ft;
FILE *fp1;
FILE *fp3;
main()
{
char m[80];
clrscr();
printf("NAME OF THE INPUT FILE:     ");
scanf("%s",infile);
printf("NAME OF THE OUTPUT FILE :     ");
scanf("%s",outfile);
fi = fopen(infile,"r");
```

```
ft = fopen("temp","w");
clrscr();
   fgets(m,80,fi);
   while(!feof(fi))
   {
       i=0;
       p=m; p1=m;
       if(m[5] ==':')
       { v=0;
        pass1();
                 for(i=6;m[i]!='\n';i++)
                  fprintf(ft,"%c",m[i]);

                fprintf(ft,"\n");
       }
    else
     {  v=1;
     fprintf(ft,"%s",m);
     }
     symbol1();
     fgets(m,80,fi);
    }  /* end of while */
    fclose(fi);
    fclose(ft);
     addr= 0x1000;
     pass2();

     } /* end of main */


 pass2()
   {
    fp1 = fopen("temp","r");
    fp3 = fopen(infile,"r");
    fp = fopen(outfile,"w");
    fprintf(fp,"***********************************\n");
    fprintf(fp," ADDRESS   LABEL  MNEMONICS   OPCODE\n");
    fprintf(fp,"***********************************\n\n");
    fgets(m,80,fp1);
    fgets(n,80,fp3);
    while(!feof(fp1))
    {
      p=m;
      i=0;

          while(strncmp(p,tab[i].mnemo,tab[i].bit) != 0)
                 i+=1;
                 switch(tab[i].index)
              {
                  case 1:
                         movfn();
                         break;
                  case 2:addfn();
                         break;
                fprintf(fp,"%c",m[i]);
                if(w==2)
                fprintf(fp,"\t\t%10X",op[1]);
                else
                fprintf(fp,"\t%10X%5X\t",op[1],op[2]);
                if(w==0)
                fprintf(fp,"%5c%c\t%c%c",op[3],op[4],op[5],op[6]);
                fprintf(fp,"\n");
                fgets(m,80,fp1);
                fgets(n,80,fp3);
                locfn();
                w = 1;
                }

            fclose(fp);
            fclose(fp1);
            fclose(fp3);
     }


pass1()
{
int q1=0,ct=0;
if(isalpha(*p))
{
for(i=1;i<5;i++)
 {
  p=p+1;
  if(isalpha(*p) !! isdigit(*p))
  ct=ct+1;
 }
}
     if(ct ==4)
     {
      p=p-4;
      for(i=0;i<5;i++)
      {
       lab[q1].name[i] =*p;
       *p++;
      }
     }

        lab[q1].locat =addr;
        q1++;
 }

symbol1()
{
  i=0;
  if(v==1){
             while(strncmp(p1,tab[i].mnemo,tab[i].bit) != 0)
                     i+=1;
            }
  else    {
      p1 =p1 +6;

      while(strncmp(p1,tab[i].mnemo,tab[i].bit)  != 0)
          i+=1;
     }
             e= s = tab[i].index;
  if((s==6)   ¦¦(s==7)   ¦¦(s==8)  ¦¦(s==9)   ¦¦(s==17)
  ¦¦(s==18) ¦¦(s==19))
  e=6;
   if((s==20) ¦¦(s==21) ¦¦(s==22) ¦¦(s==23) ¦¦(s==24))
    e =6;
          switch(e)  {
           case 6:
              addr = addr + 0x01;
              break;
           case 5:

              if((isxdigit(m[9]))¦¦(isxdigit(m[15]

              addr = addr +0x04;
              else addr = addr +0x02;
           case 15:
               addr = addr +0x02;
               break;
           case 11:
               if((isxdigit(m[7]))¦¦(isxdigit(m[13

               addr=addr +0x04;
               else addr=addr+0x02;
               break;
      default:
              if((isxdigit(m[8]))¦¦(isxdigit(m[14]

              addr = addr +0x04;
              else addr = addr +0x02;
               break;
               }
       }

jmpfn()
{
 char aaa[10];
 int var1,temp;
 op[1]=0xd8;
 p=p+4;
   for(i=0;i<5;i++)
   {
    aaa[i] =*p;
    p=p+1;
   }
   aaa[i] = '\0';
   i=0;

   while(strcmp(lab[i].name,aaa) !=0) {
   i++;                               }
   var1= lab[i].locat;
   if(addr >var1)
```

```
        {
         temp=addr-var1;
         temp=255-temp;
         op[2]=temp;
        }
        else
          {
             temp=var1-addr;
             op[2]=temp;
           }

}

 movfn()
 {
 p+=4;
 switch(*p++)
 {
 case 'a':
         if(*p++ == 'x')
                   movafn();
                   break;
 case 'c':
           if(*p++ =='x')
             movcfn();
           break;
 case 'd':
          if(*p++ =='x')
           movdfn();
          else
             movdifn();
           break;
 case 'b':
         if(*p++ =='x')
            movbfn();
         else
             movbpfn();
            break;

 case 's':
          if(*p++ =='p')
              movspfn();
          else
             movsifn();
              break;
  default:
             break;

             }
   }

 imme()
 {
        p = p+1;
     if(isxdigit(*p))
     {*p++;
     if(isxdigit(*p))
     w = 0;
     else w = 1;  }
     else{
     *p++;
     w = 1;}
     p = p-1; }

  movafn()
  {
     imme();
     switch(w)  {
     case 0 :
     op[1] =0x0c <<4;
     op[1] =op[1] ¦0x07;
     op[2] =0x00;
     op[3] =*p;
     *p++;
     op[4] = *p;*p++;
     op[5] = *p;*p++;
     op[6] = *p;
     break;
     case 1:
         op[1] =0x08 << 4;
         op[1] =op[1] ¦0x0b;
     switch(*p++)
     {
     case ´c´:
        if(*p++ == ´x´)
        {
         op[2] =0xc1;
         }
         break;
  case´d´:
         if(*p++ == ´x´)
           {
              op[2] = 0xc2;

              }
           else       {
            op[2] =0xc7;
            }
            break;
case ´b´:
             if(*p++ == ´x´)
             {
              op[2] =0xc3;
              }
             else
             {
               op[2] =0xc5;
              }
              break;
case 's':
          if(*p++ == 'p')
         {
          op[2] =0xc4;
          }
        else
        {
          op[2] =0xc6;
          }
     break;
     default: printf("error in mnemonic\n");
              break;
     }
     }
     }
movcfn()
{
 imme();
  switch(w)  {
  case 0 :
  op[1] =0x0c <<4;
  op[1] =op[1] |0x07;
  op[2] =0x01;
  op[3] =*p;
  *p++;
  op[4] = *p;*p++;
  op[5] = *p;*p++;
  op[6] = *p;
  break;
  case 1:
      op[1] =0x08 << 4;
      op[1] =op[1] |0x0b;
  switch(*p++)
  {
  case 'a':
     if(*p++ == 'x')
     {
      op[2] =0xc8;
     }
      break;
 case'd':
      if(*p++ == 'x')
        {
            op[2] = 0xca;
        }
            else
            {
         op[2] =0xcf;
             }
         break;
case 'b':
           if(*p++ == 'x')
```

```
              {
               op[2] =0xcb;
               }
              else
              {
                op[2] =0xcd;
                }
                break;
case 's':
          if(*p++ == 'p')
         {
          op[2] =0xcc;
          }
        else
        {
          op[2] =0xce;
          }
      break;
      default: printf("error in mnemonic\n");
              break;
      }
      }
      }
 movdfn()
 {
    imme();
   switch(w)  {
   case 0 :
   op[1] =0x0c <<4;
   op[1] =op[1] |0x07;
   op[2] =0x02;
   op[3] =*p;
   *p++;
   op[4] = *p;*p++;
   op[5] = *p;*p++;
   op[6] = *p;
   break;
   case 1 :
       op[1] =0x08 << 4;
       op[1] =op[1] |0x0b;
   switch(*p++)
   {
   case 'a':
      if(*p++ == 'x')
      {
       op[2] =0xd0;
       }
       break;
 case'c':
       if(*p++ == 'x')
         {
            op[2] = 0xd1;
            }
          break;
```

```
case 'b':
                if(*p++ == 'x')
                {
                 op[2] =0xd3;
                 }
                else
                {
                  op[2] =0xd5;
                  }
                  break;
 case 's':
              if(*p++ == 'p')
             {
              op[2] =0xd4;
              }
           else
           {
              op[2] =0xd6;
              }
              break;
 case 'd':
                 if(*p++ == 'i')
                {
                  op[2] = 0xd7;
                  }
        break;
        default: printf("error in mnemonic\n");
        break;
        }
        }
        }
 movbfn()
 {
   imme();
   switch(w)  {
   case 0 :
   op[1] =0x0c <<4;
   op[1] =op[1] |0x07;
   op[2] =0x03;
   op[3] =*p;
   *p++;
   op[4] = *p;*p++;
   op[5] = *p;*p++;
   op[6] = *p;
   break;
   case 1:
        op[1] =0x08 << 4;
        op[1] =op[1] |0x0b;
   switch(*p++)
   {
   case 'a':
      if(*p++ == 'x')
      {
       op[2] =0xd8;
```

55

```
        }
        break;
  case'c':
        if(*p++ == 'x')
          {
             op[2] = 0xd9;

             }
           break;
case 'b':
             if(*p++ == 'p')
             {
              op[2] =0xdd;
              }
               break;
 case 's':
          if(*p++ == 'p')
         {
          op[2] =0xdc;
          }
        else
        {
          op[2] =0xde;
        }
          break;
 case 'd':
             if(*p++ == 'x')
            {
              op[2] = 0xda;
              }
               else
             {
               op[2] =0xdf;
               }
     break;
     default: printf("error in mnemonic\n");
     break;
     }
     }
     }
 movspfn()
 {
   imme();
   switch(w)  {
   case 0 :
   op[1] =0x0c <<4;
   op[1] =op[1] ¦0x07;
   op[2] =0x04;
   op[3] =*p;
   *p++;
   op[4] = *p;*p++;
   op[5] = *p;*p++;
   op[6] = *p;
   break;
   case 1 :
        op[1] =0x08 << 4;
        op[1] =op[1] |0x0b;
   switch(*p++)
   {
   case 'a':
       if(*p++ == 'x')
       {
        op[2] =0xe0;
        }
        break;
 case'c':
        if(*p++ == 'x')
          {
              op[2] = 0xe1;
              }
            break;
case 'b':
              if(*p++ == 'x')
              {
               op[2] =0xe3;
               }
               else
                {
               op[2] =0xe5;
               }
                break;
 case 's':
           if(*p++ == 'i')
          {
           op[2] =0xe6;
           }
           break;
 case 'd':
               if(*p++ == 'x')
             {
                op[2] = 0xe2;
                }
                 else
               {
                 op[2] =0xe7;
                 }
      break;
      default: printf("error in mnemonic\n");
      break;
      }
      }
      }
  movbpfn()
  {
    imme();
    switch(w)  {
    case 0 :
    op[1] =0x0c <<4;
    op[1] =op[1] ¦0x07;
    op[2] =0x05;
    op[3] =*p;
    *p++;
    op[4] = *p;*p++;
    op[5] = *p;*p++;
    op[6] = *p;
    break;
    case 1 :
        op[1] =0x08 << 4;
        op[1] =op[1] ¦0x0b;
    switch(*p++)
    {
    case ´a´:
       if(*p++ == ´x´)
       {
        op[2] =0xe8;
        }
        break;
  case´c´:
        if(*p++ == ´x´)
          {
             op[2] = 0xe9;
             }
           break;
 case ´b´:
              if(*p++ == ´x´)
              {
               op[2] =0xeb;
               }
                break;
  case ´s´:
           if(*p++ == ´p´)
          {
           op[2] =0xec;
           }
         else
         {
           op[2] =0xee;
         }
           break;
  case ´d´:
              if(*p++ == ´x´)
             {
               op[2] = 0xea;
               }
                else
              {
                op[2] =0xef;
                }
      break;
      default:printf("error in mnemonic\n");
      break;
      }
      }
      }
 movsifn()
 {
   imme();
   switch(w)  {
   case 0 :
   op[1] =0x0c <<4;
   op[1] =op[1] ¦0x07;
   op[2] =0x06;
   op[3] =*p;
   *p++;
   op[4] = *p;*p++;
   op[5] = *p;*p++;
   op[6] = *p;
   break;
   case 1:
       op[1] =0x08 << 4;
       op[1] =op[1] ¦0x0b;
   switch(*p++)
   {
   case 'a':
      if(*p++ == 'x')
      {
       op[2] =0xf0;
       }
       break;
 case'c':
       if(*p++ == 'x')
         {
            op[2] = 0xf1;

            }
           break;
case 'b':
            if(*p++ == 'x')
            {
             op[2] =0xf3;
             }
             else
              {
             op[2] =0xf5;
             }
              break;
 case 's':
          if(*p++ == 'p')
         {
          op[2] =0xf4;
          }
          break;
 case 'd':
             if(*p++ == 'x')
           {
             op[2] = 0xf2;
```

9

```
                }
                  else
                {
                  op[2] =0xf7;
                  }
        break;
        default: printf("error in mnemonic\n");
        break;
        }
        }
        }
  movdifn()
  {
    imme();
    switch(w)  {
    case 0 :
    op[1] =0x0c <<4;
    op[1] =op[1] |0x07;
    op[2] =0x07;
    op[3] =*p;
    *p++;
    op[4] = *p;*p++;
    op[5] = *p;*p++;
    op[6] = *p;
    break;
    case 1:
         op[1] =0x08 << 4;
         op[1] =op[1] |0x0b;
    switch(*p++)
    {
    case 'a':
       if(*p++ == 'x')
       {
        op[2] =0xf8;
        }
        break;
  case'c':
        if(*p++ == 'x')
          {
             op[2] = 0xf9;

             }
           break;
 case 'b':
             if(*p++ == 'x')
             {
              op[2] =0xfa;
              }
              else
                 {
              op[2] =0xfd;
              }
               break;
  case 's':
```

```
            if(*p++ == 'p')
           {
            op[2] =0xfc;
            }
            else
          {
            op[2] =0xfe;
          }
            break;
case 'd':
              if(*p++ == 'x')
             {
               op[2] = 0xfa;
               }
     break;
     default: printf("error in mnemonic\n");
     break;
     }
     }
     }
addfn()
{
p+=4;
switch(*p++)
{
case 'a':
            if(*p++ == 'x')
              addafn();
              break;
case 'c':
              if(*p++ =='x')
                addcfn();
              break;
case 'd':
             if(*p++ =='x')
              adddfn();
              else
                adddifn();
              break;
case 'b':
            if(*p++ =='x')
               addbfn();
            else
                addbpfn();
               break;

case 's':
             if(*p++ =='p')
                  addspfn();
            else
                 addsifn();
                  break;
 default:
                break;
                }
    }

   addafn()
   {
     imme();
     switch(w)  {
     case 0 :
     op[1] =0x08 <<4;
     op[1] =op[1] |0x03;
     op[2] =0x00;
     op[3] =*p;
     *p++;
     op[4] = *p;*p++;
     op[5] = *p;*p++;
     op[6] = *p;
     break;
     case 1:
         op[1] =0x00 << 4;
         op[1] =op[1] |0x03;
     switch(*p++)
     {
     case 'c':
        if(*p++ == 'x')
        {
         op[2] =0xc1;
         }
         break;
  case'd':
         if(*p++ == 'x')
           {
                op[2] = 0xc2;
                }
           else       {
            op[2] =0xc7;
            }
            break:
case 'b':
             if(*p++ == 'x')
             {
              op[2] =0xc3;
              }
             else
             {
               op[2] =0xc5;
               }
               break:
 case 's':
           if(*p++ == 'p')
          {
           op[2] =0xc4;
           }
        else
```

2

```
          {
            op[2] =0xc6;
            }
        break;
        default: printf("error in mnemonic\n");
                 break;
        }
        }
        }
    addcfn()
    {
     imme();
      switch(w)  {
      case 0 :
      op[1] =0x08 <<4;
      op[1] =op[1] |0x03;
      op[2] =0x01;
      op[3] =*p;
      *p++;
      op[4] = *p;*p++;
      op[5] = *p;*p++;
      op[6] = *p;
      break;
      case 1:
      op[1] =0x00 << 4;
       op[1] =op[1] |0x03;
      switch(*p++)
      {
      case 'a':
         if(*p++ == 'x')
         {
          op[2] =0xc8;
          }
          break;
    case'd':
          if(*p++ == 'x')
            {
               op[2] = 0xca;
               }
               else
               {
             op[2] =0xcf;
             }
             break;
   case 'b':
               if(*p++ == 'x')
               {
                op[2] =0xcb;
                }
               else
               {
                 op[2] =0xcd;
                 }
                 break;
```

```
  case 's':
          if(*p++ == 'p')
         {
          op[2] =0xcc;
          }
        else
        {
          op[2] =0xce;
          }
      break;
      default: printf("error in mnemonic\n");
               break;
      }
      }
      }
  adddfn()
  {
     imme();
    switch(w)  {
    case 0 :
    op[1] =0x08 <<4;
    op[1] =op[1] |0x03;
    op[2] =0x02;
    op[3] =*p;
    *p++;
    op[4] = *p;*p++;
    op[5] = *p;*p++;
    op[6] = *p;
    break;
    case 1:
        op[1] =0x00 << 4;
        op[1] =op[1] |0x03;
    switch(*p++)
    {
    case 'a':
       if(*p++ == 'x')
       {
        op[2] =0xd0;
        }
        break;
 case'c':
        if(*p++ == 'x')
          {
             op[2] = 0xd1;
             }
            break;
case 'b':
             if(*p++ == 'x')
             {
              op[2] =0xd3;
              }
             else
             {
               op[2] =0xd5;
```

```
                }
                break;
case 's':
           if(*p++ == 'p')
          {
           op[2] =0xd4;
           }
         else
         {
           op[2] =0xd6;
           }
           break;
case 'd':
             if(*p++ == 'i')
            {
              op[2] = 0xd7;
              }
     break;
     default: printf("error in mnemonic\n");

     break;
     }
     }
      }
addbfn()
{
    imme();
   switch(w)  {
   case 0 :
   op[1] =0x08 <<4;
   op[1] =op[1] |0x03;
   op[2] =0x03;
   op[3] =*p;
   *p++;
   op[4] = *p;*p++;
   op[5] = *p;*p++;
   op[6] = *p;
   break;
   case 1:
        op[1] =0x00 << 4;
        op[1] =op[1] |0x03;
   switch(*p++)
   {
   case 'a':
       if(*p++ == 'x')
       {
        op[2] =0xd8;
        }
        break;
case'c':
        if(*p++ == 'x')
          {
              op[2] = 0xd9;
              }
```

```
                break;
        case 'b':
                  if(*p++ == 'p')
                  {
                   op[2] =0xdd;
                   }
                     break;
         case 's':
                    if(*p++ == 'p')
                  {
                    op[2] =0xdc;
                    }
                 else
                 {
                    op[2] =0xde;
                 }
                    break;
         case 'd':
                      if(*p++ == 'x')
                    {
                      op[2] = 0xda;
                      }
                       else
                     {
                       op[2] =0xdf;
                       }
            break;
            default: printf("error in mnemonic\n");
            break;
            }
            }
            }
addspfn()
{
 imme();
  switch(w)  {
  case 0 :
  op[1] =0x08 <<4;
  op[1] =op[1] |0x03;
  op[2] =0x04;
  op[3] =*p;
  *p++;
  op[4] = *p;*p++;
  op[5] = *p;*p++;
  op[6] = *p;
  break;
  case 1:
       op[1] =0x00 << 4;
       op[1] =op[1] |0x03;
  switch(*p++)
  {
  case 'a':
      if(*p++ == 'x')
      {
         op[2] =0xe0;
         }
         break;
  case´c´:
         if(*p++ == ´x´)
           {
              op[2] = 0xe1;
              }
            break;
 case ´b´:
             if(*p++ == ´x´)
             {
              op[2] =0xe3;
              }
              else
               {
              op[2] =0xe5;
              }
               break;
  case ´s´:
           if(*p++ == ´i´)
          {
           op[2] =0xe6;
           }
           break;
  case ´d´:
             if(*p++ == ´x´)
            {
              op[2] = 0xe2;
              }
               else
             {
               op[2] =0xe7;
               }
      break;
      default: printf("error in mnemonic\n");
      break;
      }
      }
      }
  addbpfn()
  {
    imme();
    switch(w)  {
    case 0 :
    op[1] =0x08 <<4;
    op[1] =op[1] ¦0x03;
    op[2] =0x05;
    op[3] =*p;
    *p++;
    op[4] = *p;*p++;
    op[5] = *p;*p++;
    op[6] = *p;
    break;
```

```
      case 1:
           op[1] =0x00 << 4;
           op[1] =op[1] ¦0x03;
      switch(*p++)
      {
      case 'a':
          if(*p++ == 'x')
          {
           op[2] =0xe8;
           }
           break;
    case'c':
           if(*p++ == 'x')
             {
                op[2] = 0xe9;
                }
              break;
   case 'b':
                if(*p++ == 'x')
                {
                 op[2] =0xeb;
                 }
                  break;
    case 's':
             if(*p++ == 'p')
            {
             op[2] =0xec;
             }
           else
           {
             op[2] =0xee;
           }
             break;
    case 'd':
                if(*p++ == 'x')
               {
                 op[2] = 0xea;
                 }
                  else
                {
                  op[2] =0xef;
                  }
         break;
         default:printf("error in mnemonic\n");
         break;
         }
         }
         }
   addsifn()
    {
     imme();
      switch(w)  {
      case 0 :
      op[1] =0x08 <<4;
    op[1] =op[1] ¦0x03;
    op[2] =0x06;
    op[3] =*p;
    *p++;
    op[4] = *p;*p++;
    op[5] = *p;*p++;
    op[6] = *p;
    break;
    case 1:
        op[1] =0x00 << 4;
        op[1] =op[1] ¦0x03;
    switch(*p++)
    {
    case 'a':
       if(*p++ == 'x')
       {
        op[2] =0xf0;
        }
        break;
  case'c':
        if(*p++ == 'x')
          {
             op[2] = 0xf1;
             }
           break;
 case 'b':
              if(*p++ == 'x')
              {
               op[2] =0xf3;
               }
      .        else
                {
               op[2] =0xf5;
               }
                break;
  case 's':
            if(*p++ == 'p')
           {
            op[2] =0xf4;
            }
            break;
  case 'd':
              if(*p++ == 'x')
             {
               op[2] = 0xf2;
               }
                else
              {
                op[2] =0xf7;
                }
      break;
      default: printf("error in mnemonic\n");
      break;
      }
      }
      }
adddifn()
 {
   imme();
   switch(w)  {
   case 0 :
   op[1] =0x08 <<4;
   op[1] =op[1] ¦0x03;
   op[2] =0x07;
   op[3] =*p;
   *p++;
   op[4] = *p;*p++;
   op[5] = *p;*p++;
   op[6] = *p;
   break;
   case 1:
       op[1] =0x00 << 4;
       op[1] =op[1] ¦0x03;
   switch(*p++)
   {
   case 'a':
      if(*p++ == 'x')
      {
       op[2] =0xf8;
       }
       break;
 case'c':
        if(*p++ == 'x')
          {
             op[2] = 0xf9;
             }
            break;
case 'b':
              if(*p++ == 'x')
              {
             → op[2] =0xfb;
               }
               else
                   {
               op[2] =0xfd;
               }
                break;
 case 's':
          if(*p++ == 'p')
         {
          op[2] =0xfc;
          }
          else
        {
          op[2] =0xfe;
        }
          break;
 case 'd':
```

```
            if(*p++ == 'x')
           {
             op[2] = 0xfa;
             }
      break;
      default: printf("error in mnemonic\n");
      break;
      }
      }
       }
subfn()
{
p+=4;
switch(*p++)
{
case 'a':
          if(*p++ == 'x')
            subafn();
            break;
case 'c':
            if(*p++ =='x')
              subcfn();
            break;
case 'd':
           if(*p++ =='x')
            subdfn();
           else
              subdifn();
            break;
case 'b':
          if(*p++ =='x')
             subbfn();
          else
              subbpfn();
             break;

case 's':
           if(*p++ =='p')
                subspfn();
           else
               subsifn();
                break;
 default:
              break;

              }
   }


subafn()
{
imme();
  switch(w)  {
  case 0 :
    op[1] =0x08 <<4;
    op[1] =op[1] |0x03;
    op[2] =0x28;
    op[3] =*p;
    *p++;
    op[4] = *p;*p++;
    op[5] = *p;*p++;
    op[6] = *p;
    break;
    case 1:
        op[1] =0x00 << 4;
        op[1] =op[1] |0x2b;
    switch(*p++)
    {
    case 'c':
       if(*p++ == 'x')
       {
        op[2] =0xc1;
        }
        break;
  case'd':
        if(*p++ == 'x')
          {
             op[2] = 0xc2;
             }
          else       {
           op[2] =0xc7;
           }
           break;
 case 'b':
             if(*p++ == 'x')
             {
              op[2] =0xc3;
              }
             else
             {
               op[2] =0xc5;
               }
               break;
  case 's':
          if(*p++ == 'p')
         {
          op[2] =0xc4;
          }
        else
        {
          op[2] =0xc6;
          }
      break;
      default: printf("error in mnemonic\n");
               break;
      }
      }
      }
 subcfn()
 {
   imme();
   switch(w)  {
   case 0 :
   op[1] =0x08 <<4;
   op[1] =op[1] |0x03;
   op[2] =0x29;
   op[3] =*p;
   *p++;
   op[4] = *p;*p++;
   op[5] = *p;*p++;
   op[6] = *p;
   break;
   case 1:
       op[1] =0x00 << 4;
       op[1] =op[1] |0x2b;
   switch(*p++)
   {
   case 'a':
      if(*p++ == 'x')
      {
       op[2] =0xc8;
       }
       break;
 case'd':
       if(*p++ == 'x')
         {
            op[2] = 0xca;
            }
            else
            {
          op[2] =0xcf;
          }
          break;
case 'b':
            if(*p++ == 'x')
            {
             op[2] =0xcb;
             }
            else
            {
              op[2] =0xcd;
              }
              break;
 case 's':
         if(*p++ == 'p')
        {
         op[2] =0xc6;
         }
       else
       {
         op[2] =0xce;
         }
```

```
     break;
     default: printf("error in mnemonic\n");
               break;
     }
     }
     }
subdfn()
{
 imme();
  switch(w)  {
  case 0 :
  op[1] =0x08 <<4;
  op[1] =op[1] ¦0x03;
  op[2] =0x0a;
  op[3] =*p;
  *p++;
  op[4] = *p;*p++;
  op[5] = *p;*p++;
  op[6] = *p;
  break;
  case 1:
       op[1] =0x00 << 4;
       op[1] =op[1] ¦0x2b;
  switch(*p++)
  {
  case 'a':
      if(*p++ == 'x')
      {
       op[2] =0xd0;
       }
       break;
 case'c':
       if(*p++ == 'x')
          {
              op[2] = 0xd1;
              }
            break;
case 'b':
             if(*p++ == 'x')
             {
              op[2] =0xd3;
              }
             else
             {
               op[2] =0xd5;
               }
               break;
 case 's':
           if(*p++ == 'p')
          {
           op[2] =0xd4;
           }
          else
          {
```

```
          op[2] =0xd6;
          }
          break;
 case 'd':
            if(*p++ == 'i')
           {
             op[2] = 0xd7;
             }
     break;
     default: printf("error in mnemonic\n");
     break;
     }
     }
     }
 subbfn()
 {
  imme();
   switch(w)  {
   case 0 :
   op[1] =0x08 <<4;
   op[1] =op[1] ¦0x03;
   op[2] =0x0b;
   op[3] =*p;
   *p++;
   op[4] = *p;*p++;
   op[5] = *p;*p++;
   op[6] = *p;
   break;
   case 1:
       op[1] =0x00 << 4;
       op[1] =op[1] ¦0x2b;
   switch(*p++)
   {
   case 'a':
      if(*p++ == 'x')
      {
       op[2] =0xd8;
       }
       break;
 case'c':
       if(*p++ == 'x')
         {
             op[2] = 0xd9;
             }
           break;
case 'b':
            if(*p++ == 'p')
            {
             op[2] =0xdd;
             }
              break;
 case 's':
          if(*p++ == 'p')
         {
          op[2] =0xdc;
          }
         else
         {
          op[2] =0xde;
         }
          break;
  case 'd':
           if(*p++ == 'x')
          {
            op[2] = 0xda;
            }
             else
           {
             op[2] =0xdf;
             }
      break;
      default: printf("error in mnemonic\n");
      break;
      }
      }
      }
  subspfn()
  {
      imme();
    switch(w)  {
    case 0 :
    op[1] =0x08 <<4;
    op[1] =op[1] |0x03;
    op[2] =0x2c;
    op[3] =*p;
    *p++;
    op[4] = *p;*p++;
    op[5] = *p;*p++;
    op[6] = *p;
    break;
    case 1:
        op[1] =0x00 << 4;
        op[1] =op[1] |0x2b;
    switch(*p++)
    {
    case 'a':
       if(*p++ == 'x')
       {
        op[2] =0xe0;
        }
        break;
  case'c':
        if(*p++ == 'x')
          {
             op[2] = 0xe1;
             }
           break;
case 'b':
```

```
              if(*p++ == 'x')
              {
               op[2] =0xe3;
               }
               else
                {
               op[2] =0xe5;
               }
                break;
case 's':
          if(*p++ == 'i')
         {
          op[2] =0xe6;
          }
          break;
case 'd':
              if(*p++ == 'x')
             {
               op[2] = 0xe2;
               }
                else
              {
                op[2] =0xe7;
                }
     break;
     default: printf("error in mnemonic\n");
     break;
     }
     }
     }
subbpfn()
{
   imme();
   switch(w)  {
   case 0 :
   op[1] =0x08 <<4;
   op[1] =op[1] |0x03;
   op[2] =0x2d;
   op[3] =*p;
   *p++;
   op[4] = *p;*p++;
   op[5] = *p;*p++;
   op[6] = *p;
   break;
   case 1:
       op[1] =0x00 << 4;
       op[1] =op[1] |0x2b;
   switch(*p++)
   {
   case 'a':
      if(*p++ == 'x')
      {
       op[2] =0xe8;
       }
```

```
         break;
      case'c':
         if(*p++ == 'x')
           {
              op[2] = 0xe8;
              }
             break;
    case 'b':
               if(*p++ == 'x')
               {
                op[2] =0xeb;
                }
                 break;
      case 's':
             if(*p++ == 'p')
            {
             op[2] =0xec;
             }
         else
         {
           op[2] =0xee;
         }
           break;
      case 'd':
               if(*p++ == 'x')
              {
                op[2] = 0xea;
                }
                 else
               {
                 op[2] =0xef;
                 }
        break;
        default:printf("error in mnemonic\n");
        break;
        }
        }
        }
    subsifn()
     {
        imme();
      switch(w)  {
      case 0 :
      op[1] =0x08 <<4;
      op[1] =op[1] |0x03;
      op[2] =0x0e;
      op[3] =*p;
      *p++;
      op[4] = *p;*p++;
      op[5] = *p;*p++;
      op[6] = *p;
      break;
      case 1:
          op[1] =0x00 << 4;
      op[1] =op[1] |0x2b;
   switch(*p++)
   {
   case 'a':
      if(*p++ == 'x')
      {
       op[2] =0xf0;
       }
       break;
 case'c':
       if(*p++ == 'x')
          {
              op[2] = 0xf1;
              }
             break;
case 'b':
              if(*p++ == 'x')
              {
               op[2] =0xf3;
               }
               else
                {
               op[2] =0xf5;
               }
                break;
 case 's':
             if(*p++ == 'p')
            {
             op[2] =0xf4;
             }
             break;
 case 'd':
               if(*p++ == 'x')
              {
                op[2] = 0xf2;
                }
                 else
               {
                 op[2] =0xf7;
                 }
      break;
      default: printf("error in mnemonic\n");
      break;
      }
      }
      }
 subdifn()
  {
    imme();
    switch(w)  {
    case 0 :
    op[1] =0x08 <<4;
    op[1] =op[1] |0x03;
    op[2] =0x0f;
```

```
      op[3] =*p;
      *p++;
      op[4] = *p;*p++;
      op[5] = *p;*p++;
      op[6] = *p;
      break;
      case 1:
          op[1] =0x00 << 4;
          op[1] =op[1] |0x2b;
      switch(*p++)
      {
      case 'a':
         if(*p++ == 'x')
         {
          op[2] =0xf8;
          }
          break;
    case'c':
          if(*p++ == 'x')
            {
                op[2] = 0xf9;
                }
              break;
   case 'b':
                if(*p++ == 'x')
                {
                 op[2] =0xfb;
                 }
                 else
                    {
                 op[2] =0xfd:
                 }
                  break:
    case 's':
              if(*p++ == 'p')
             {
              op[2] =0xfc:
              }
              else
            {
              op[2] =0xfe:
            }
              break:
    case 'd':
                if(*p++ == 'x')
               {
                 op[2] = 0xfa:
                 }
        break:
        default: printf("error in mnemonic\n");
        break:
        }
        }
        }
```

```
cmpfn()
{
printf("inside subfn");
p+=4;
switch(*p++)
{
case 'a':
         if(*p++ == 'x')
           cmpafn();
           break;
case 'c':
           if(*p++ =='x')
             cmpcfn();
           break;
case 'd':
          if(*p++ =='x')
           cmpdfn();
          else
             cmpdifn();
           break;
case 'b':
         if(*p++ =='x')
            cmpbfn();
         else
             cmpbpfn();
            break;

case 's':
          if(*p++ =='p')
               cmpspfn();
          else
              cmpsifn();
               break;
 default:
               break;

               }
  }

cmpafn()
{
  imme();
  switch(w)  {
  case 0 :
  op[1] =0x08 <<4;
  op[1] =op[1] !0x03;
  op[2] =0x38;
  op[3] =*p;
  *p++;
  op[4] = *p;*p++;
  op[5] = *p;*p++;
  op[6] = *p;
  break;
  case 1;
```

```
          op[1] =0x00 << 4;
          op[1] =op[1] |0x3b;
      switch(*p++)
      {
      case 'c':
         if(*p++ == 'x')
         {
          op[2] =0xc0;
          }
          break;
   case'd':
          if(*p++ == 'x')
            {
               op[2] = 0xc2;
               }
            else        {
             op[2] =0xc7;
             }
             break;
  case 'h':
              if(*p++ == 'x')
              {
               op[2] =0xc3;
               }
              else
              {
                op[2] =0xc5;
                }
                break;
    case 's':
             if(*p++ == 'p')
            {
             op[2] =0xc4;
             }
           else
           {
             op[2] =0xc6;
             }
         break;
         default: printf("error in mnemonic\n");
                  break;
         }
         }
         }
        cmpcfn()
     {
        imme();
       switch(w)  {
       case 0 :
       op[1] =0x08 <<4;
       op[1] =op[1] |0x03;
       op[2] =0x39;
       op[3] =*p;
       *p++;
     op[4] = *p;*p++;
     op[5] = *p;*p++;
     op[6] = *p;
     break;
     case 1:
          op[1] =0x00 << 4;
          op[1] =op[1] ¦0x3b;
     switch(*p++)
     {
     case ´a´:
         if(*p++ == ´x´)
         {
          op[2] =0xc8;
          }
          break;
   case´d´:
          if(*p++ == ´x´)
             {
                 op[2] = 0xca;
                 }
                 else
                 {
              op[2] =0xcf;
              }
              break;
  case ´b´:
                if(*p++ == ´x´)
                {
                 op[2] =0xcb;
                 }
                else
                {
                  op[2] =0xcd;
                  }
                  break;
   case ´s´:
            if(*p++ == ´p´)
           {
            op[2] =0xcc;
            }
          else
          {
            op[2] =0xce;
            }
        break;
        default: printf("error in mnemonic\n");
                  break;
        }
        }
        }
  cmpdfn()
   {
      imme();
     switch(w)  {
```

```
    case 0 :
    op[1] =0x08 <<4;
    op[1] =op[1] ¦0x03;
    op[2] =0x3a;
    op[3] =*p;
    *p++;
    op[4] = *p;*p++;
    op[5] = *p;*p++;
    op[6] = *p;
    break;
    case 1:
        op[1] =0x00 << 4;
        op[1] =op[1] ¦0x3b;
    switch(*p++)
    {
    case ´a´:
       if(*p++ == ´x´)
       {
        op[2] =0xd0;
        }
        break;
  case´c´:
        if(*p++ == ´x´)
          {
             op[2] = 0xd1;
             }
           break;
 case ´b´:
              if(*p++ == ´x´)
              {
               op[2] =0xd3;
               }
              else
              {
                op[2] =0xd5;
                }
                break;
  case ´s´:
          if(*p++ == ´p´)
         {
          op[2] =0xd4;
          }
        else
        {
          op[2] =0xd6;
          }
          break;
  case ´d´:
            if(*p++ == ´i´)
           {
             op[2] = 0xd7;
             }
      break;
      default: printf("error in mnemonic\n");
```

```
     break;
     }
     }
     }

cmpbfn()
 {
    imme();
   switch(w)  {
   case 0 :
   op[1] =0x08 <<4;
   op[1] =op[1] ¦0x03;
   op[2] =0x3b;
   op[3] =*p;
   *p++;
   op[4] = *p;*p++;
   op[5] = *p;*p++;
   op[6] = *p;
   break;
   case 1:
       op[1] =0x00 << 4;
       op[1] =op[1] ¦0x3b;
   switch(*p++)
   {
   case 'a':
      if(*p++ == 'x')
      {
       op[2] =0xd8;
       }
       break;
  case'c':
       if(*p++ == 'x')
         {
            op[2] = 0xd9;
            }
           break;
 case 'b':
              if(*p++ == 'p')
              {
               op[2] =0xdd;
               }
                break;
  case 's':
            if(*p++ == 'p')
           {
            op[2] =0xdc;
            }
          else
          {
            op[2] =0xde;
          }
            break;
  case 'd':
              if(*p++ == 'x')
              {
                op[2] = 0xda;
                }
                 else
               {
                 op[2] =0xdf;
                 }
      break;
      default: printf("error in mnemonic\n");
      break;
      }
      }
      }
cmpspfn()
 {
   imme();
   switch(w)  {
   case 0 :
   op[1] =0x08 <<4;
   op[1] =op[1] ¦0x03;
   op[2] =0x3c;
   op[3] =*p;
   *p++;
   op[4] = *p;*p++;
   op[5] = *p;*p++;
   op[6] = *p;
   break;
   case 1:
       op[1] =0x00 << 4;
       op[1] =op[1] ¦0x3b;
   switch(*p++)
   {
   case 'a':
      if(*p++ == 'x')
      {
       op[2] =0xe0;
       }
       break;
 case'c':
       if(*p++ == 'x')
         {
            op[2] = 0xe1;
            }
           break;
case 'b':
              if(*p++ == 'x')
              {
               op[2] =0xe3;
               }
               else
                {
               op[2] =0xe5;
               }
                break;
 case ´s´:
          if(*p++ == ´i´)
         {
          op[2] =0xe6;
          }
          break;
 case ´d´:
            if(*p++ == ´x´)
           {
             op[2] = 0xe2;
             }
              else
            {
              op[2] =0xe7;
              }
      break;
      default: printf("error in mnemonic\n");
      break;
      }
      }
      }
cmpbpfn()
 {
   imme();
   switch(w)  {
   case 0 :
   op[1] =0x08 <<4;
   op[1] =op[1] |0x03;
   op[2] =0x3d;
   op[3] =*p;
   *p++;
   op[4] = *p;*p++;
   op[5] = *p;*p++;
   op[6] = *p;
   break;
   case 1:
       op[1] =0x00 << 4;
       op[1] =op[1] |0x3b;
   switch(*p++)
   {
   case ´a´:
      if(*p++ == ´x´)
      {
       op[2] =0xe8;
       }
       break;
 case´c´:
       if(*p++ == ´x´)
         {
            op[2] = 0xe9;
            }
          break;
case ´b´:
           if(*p++ == ´x´)
```

```
               {
                op[2] =0xeb;
                }
                 break;
    case ´s´:
            if(*p++ == ´p´)
           {
            op[2] =0xec;
            }
          else
          {
            op[2] =0xee;
          }
            break;
     case ´d´:
                if(*p++ == ´x´)
               {
                 op[2] = 0xea;
                 }
                  else
                {
                  op[2] =0xef;
                  }
         break;
         default:printf("error in mnemonic\n");
         break;
         }
         }
         }
    cmpsifn()
     {
       imme();
       switch(w)  {
       case 0 :
       op[1] =0x08 <<4;
       op[1] =op[1] |0x03;
       op[2] =0x3e;
       op[3] =*p;
       *p++;
       op[4] = *p;*p++;
       op[5] = *p;*p++;
       op[6] = *p;
       break;
       case 1:
           op[1] =0x00 << 4;
           op[1] =op[1] |0x3b;
       switch(*p++)
       {
       case ´a´:
          if(*p++ == ´x´)
          {
           op[2] =0xf0;
           }
           break;
```

```
 case´c´:
       if(*p++ == ´x´)
         {
            op[2] = 0xf1;
            }
          break;
case ´b´:
           if(*p++ == ´x´)
           {
            op[2] =0xf3;
            }
            else
             {
            op[2] =0xf5;
            }
             break;
 case ´s´:
         if(*p++ == ´p´)
        {
         op[2] =0xf4;
         }
         break;
 case ´d´:
           if(*p++ == ´x´)
          {
            op[2] = 0xf2;
            }
             else
           {
             op[2] =0xf7;
             }
     break;
     default: printf("error in mnemonic\n");
     break;
     }
     }
     }
cmpdifn()
 {
   imme();
   switch(w)  {
   case 0 :
   op[1] =0x08 <<4;
   op[1] =op[1] ¦0x03;
   op[2] =0x3f;
   op[3] =*p;
   *p++;
   op[4] = *p;*p++;
   op[5] = *p;*p++;
   op[6] = *p;
   break;
   case 1:
       op[1] =0x00 << 4;
       op[1] =op[1] ¦0x3b;
```

89

```
   switch(*p++)
   {
   case ´a´:
      if(*p++ == ´x´)
      {
       op[2] =0xf8;
       }
       break;
 case´c´:
       if(*p++ == ´x´)
         {
            op[2] = 0xf9;
            }
          break;
case ´b´:
           if(*p++ == ´x´)
           {
            op[2] =0xfb;
            }
            else
               {
            op[2] =0xfd;
            }
             break;
 case ´s´:
         if(*p++ == ´p´)
        {
         op[2] =0xfc;
         }
         else
       {
         op[2] =0xfe;
       }
         break;
 case ´d´:

           if(*p++ == ´x´)
          {
            op[2] = 0xfe;
            }
     break;
     default: printf("error in mnemonic\n");
     break;
     }
     }
     }

  testfn()
 {
 p+=5;
 switch(*p++)
 {
 case ´a´:
         if(*p++ == ´x´)
```

^10

```
          printf("calling test");getch();
            testafn();
            break;
case 'c':
            if(*p++ =='x')
              testcfn();
            break;
case 'd':
           if(*p++ =='x')
            testdfn();

           else
              testdifn();
            break;
case 'b':
          if(*p++ =='x')
             testbfn();
          else
              testbpfn();
             break;

case 's':
           if(*p++ =='p')
                testspfn();
           else
               testsifn();
                break;
 default:
              break;

              }
  }

 testafn()
 {
    imme();
   switch(w)  {
   case 0 :
   op[1] =0x0f <<4;
   op[1] =op[1] |0x07;
   op[2] =0x00;
   op[3] =*p;
   *p++;
   op[4] = *p;*p++;
   op[5] = *p;*p++;
   op[6] = *p;
   break;
   case 1:
       op[1] =0x00 << 4;
       op[1] =op[1] |0x85;
   switch(*p++)
   {
   case 'c':
      if(*p++ == 'x')
```

```
        {
         op[2] =0xc0;
         }
         break;
  case´d´:
         if(*p++ == ´x´)
           {
              op[2] = 0xc2;
              }
           else       {
            op[2] =0xc7;
            }
            break;
 case ´b´:
             if(*p++ == ´x´)
             {
              op[2] =0xc3;
              }
             else
             {
               op[2] =0xc5;
               }
               break;
  case ´s´:
          if(*p++ == ´p´)
         {
          op[2] =0xc4;
          }
        else
        {
          op[2] =0xc6;
          }
      break;
      default: printf("error in mnemonic\n");
                break;
      }
      }
      }
testcfn()
 {
     imme();
   switch(w)  {
   case 0 :
   op[1] =0x0f <<4;
   op[1] =op[1] ¦0x07;
   op[2] =0x01;
   op[3] =*p;
   *p++;
   op[4] = *p;*p++;
   op[5] = *p;*p++;
   op[6] = *p;
   break;
   case 1:
       op[1] =0x00 << 4;
     imme();
   switch(w)  {
   case 0 :
   op[1] =0x0f <<4;
   op[1] =op[1] |0x07;
   op[2] =0x03;
   op[3] =*p;
   *p++;
   op[4] = *p;*p++;
   op[5] = *p;*p++;
   op[6] = *p;
   break;
   case 1:
       op[1] =0x00 << 4;
       op[1] =op[1] |0x85;
   switch(*p++)
   {
   case 'a':
      if(*p++ == 'x')
      {
       op[2] =0xd8;
       }
       break;
 case'c':
       if(*p++ == 'x')
         {
              op[2] = 0xd9;
              }
            break;
case 'b':
             if(*p++ == 'p')
             {
              op[2] =0xdd;
              }
               break;
 case 's':
           if(*p++ == 'p')
          {
           op[2] =0xdc;
           }
         else
         {
           op[2] =0xde;
         }
           break;
 case 'd':
             if(*p++ == 'x')
            {
              op[2] = 0xda;
              }
               else
             {
               op[2] =0xdf;
               }
```

```
        break;
        default: printf("error in mnemonic\n");
        break;
        }
        }
        }
 testspfn()
  {
      imme();
    switch(w)  {
    case 0 :
    op[1] =0x0f <<4;
    op[1] =op[1] ¦0x07;
    op[2] =0x04;
    op[3] =*p;
    *p++;
    op[4] = *p;*p++;
    op[5] = *p;*p++;
    op[6] = *p;
    break;
    case 1:
        op[1] =0x00 << 4;
        op[1] =op[1] ¦0x85;
    switch(*p++)
    {
    case 'a':
       if(*p++ == 'x')
       {
        op[2] =0xe0;
        }
        break;
 case'c':
        if(*p++ == 'x')
          {
              op[2] = 0xe1;
              }
             break;
case 'b':
              if(*p++ == 'x')
              {
               op[2] =0xe3;
               }
               else
                {
               op[2] =0xe5;
               }
                break;
 case 's':
          if(*p++ == 'i')
         {
          op[2] =0xe6;
          }
          break;
 case 'd':
```

```
              op[2] =0xec:
              }
            else
            {
              op[2] =0xee:
            }
              break:
    case 'd':
                 if(*p++ == 'x')
                {
                  op[2] = 0xea:
                  }
                   else
                 {
                   op[2] =0xef:
                   }
         break:
         default:printf("error in mnemonic\n");
         break:
         }
         }
         }
   testsifn()
    {
     imme():
      switch(w)  {
      case 0 :
      op[1] =0x0f <<4:
      op[1] =op[1] |0x07:
      op[2] =0x06:
      op[3] =*p;
      *p++:
      op[4] = *p;*p++:
      op[5] = *p:*p++:
      op[6] = *p;
      break;
      case 1:
           op[1] =0x00 << 4;
           op[1] =op[1] |0x85:
      switch(*p++)
      {
      case 'a':
         if(*p++ == 'x')
         {
          op[2] =0xf0:
          }
          break:
     case'c':
          if(*p++ == 'x')
            {
                op[2] = 0xf1:
                }
              break:
    case 'b':
```

```
              if(*p++ == 'x')
              {
               op[2] =0xf3;
               }
               else
                {
               op[2] =0xf5;
               }
                break;
  case 's':
             if(*p++ == 'p')
            {
             op[2] =0xf4;
             }
             break;
  case 'd':
               if(*p++ == 'x')
              {
                op[2] = 0xf2;
                }
                 else
               {
                 op[2] =0xf7;
                 }
      break;
      default: printf("error in mnemonic\n");
      break;
      }
      }
      }
testdifn()
 {
  imme();
   switch(w)  {
   case 0 :
   op[1] =0x0f <<4;
   op[1] =op[1] |0x07;
   op[2] =0x07;
   op[3] =*p;
   *p++;
   op[4] = *p;*p++;
   op[5] = *p;*p++;
   op[6] = *p;
   break;
   case 1:
       op[1] =0x00 << 4;
       op[1] =op[1] |0x85;
   switch(*p++)
   {
   case 'a':
      if(*p++ == 'x')
      {
       op[2] =0xf8;
       }
```

```
        break;
  case'c':
        if(*p++ == 'x')
          {
             op[2] = 0xf9;
             }
           break;
case 'b':
             if(*p++ == 'x')
             {
              op[2] =0xfb;
              }
              else
                 {
              op[2] =0xfd;
              }
               break;
 case 's':
          if(*p++ == 'p')
         {
          op[2] =0xfc;
          }
          else
        {
          op[2] =0xfe;
        }
          break;
 case 'd':

             if(*p++ == 'x')
            {
              op[2] = 0xfe;
              }
     break;
     default: printf("error in mnemonic\n");
     break;
     }
     }
     }
 mulfn()
{
w = 1;
m[6]=' ';m[7]=' ';m[8]='\n';
p+=4;
op[1] = 0xf0 ;
op[1] = op[1]|0x07;
switch(*p++)
{
case 'a':
        if(*p++ == 'x')
        op[2] = 0x40;
          break;
case 'c':
           if(*p++ =='x')
```

```
              op[2] = 0x41;
              break;
case 'd':
             if(*p++ =='x')
             op[2] = 0x42;
            else
            op[2] = 0x47;
              break;
case 'b':
            if(*p++ =='x')
            op[2] = 0x43;
           else
           op[2] = 0x45;
               break;

case 's':
             if(*p++ =='p')
             op[2] = 0x44;
            else
            op[2] = 0x46;
                  break;
 default:
                break;

                }
  }

 incfn()
{
w = 2;
p+=4;
switch(*p++)
{
case 'a':
            if(*p++ == 'x')
            op[1] = 0x40;
              break;
case 'c':
              if(*p++ =='x')
              op[1] = 0x41;
              break;
case 'd':
             if(*p++ =='x')
             op[1] = 0x42;
            else
            op[1] = 0x47;
              break;
case 'b':
            if(*p++ =='x')
            op[1] = 0x43;
           else
           op[1] = 0x45;
               break;
```

```
case 's':
         if(*p++ =='p')
         op[1] = 0x44;
        else
        op[1] = 0x46;
              break;
 default:
            break;

            }
  }
 decfn()
{
w = 2;
p+=4;
switch(*p++)
{
case 'a':
        if(*p++ == 'x')
        op[1] = 0x48;
          break;
case 'c':
          if(*p++ =='x')
          op[1] = 0x49;
          break;
case 'd':
         if(*p++ =='x')
         op[1] = 0x4a;
        else
        op[1] = 0x4f;
          break;
case 'b':
        if(*p++ =='x')
        op[1] = 0x4b;
       else
       op[1] = 0x4d;
           break;

case 's':
         if(*p++ =='p')
         op[1] = 0x4c;
        else
        op[1] = 0x4e;
              break;
 default:
            break;

            }
  }
      pushfn()
{
w = 2;
p+=5;
switch(*p++)
```

```
  {
  case 'a':
          if(*p++ == 'x')
          op[1] = 0x50;
            break;
  case 'c':
            if(*p++ =='x')
            op[1] = 0x51;
            break;
  case 'd':
          if(*p++ =='x')
          op[1] = 0x52;
         else
         op[1] = 0x57;
           break;
  case 'b':
          if(*p++ =='x')
          op[1] = 0x53;
         else
         op[1] = 0x55;
             break;

  case 's':
          if(*p++ =='p')
          op[1] = 0x54;
         else
         op[1] = 0x56;
               break;
  default:
             break;

             }
  }
     popfn()
{
w = 2;
p+=4;
switch(*p++)
{
case 'a':
        if(*p++ == 'x')
        op[1] = 0x58;
          break;
case 'c':
          if(*p++ =='x')
          op[1] = 0x59;
          break;
case 'd':
         if(*p++ =='x')
         op[1] = 0x5a;
        else
        op[1] = 0x5f;
          break;
case 'b':
```

```
          if(*p++ =='x')
          op[1] = 0x5b;
         else
         op[1] = 0x5d;
             break;

   case 's':
           if(*p++ =='p')
           op[1] = 0x5c;
          else
          op[1] = 0x5e;
                break;
    default:
               break;

               }
     }
     clcfn()     {
     w=2;
     m[3]=' ';m[4]=' ';m[5]='\n';
     op[1] =0xf8; }

     stcfn()     {
     w=2;
     m[3]=' ';m[4]=' ';m[5]='\n';
     op[1] =0xf9;
     }

     waitfn()     {
     w=2;
     m[4]=' ';m[5]='\n';
     op[1] =0xfa; }

    aasfn()     {
     w=2;
     m[3]=' ';m[4]=' ';m[5]='\n';
     op[1] =0xfb; }

     dasfn()     {
     w=2;
     m[3]=' ';m[4]=' ';m[5]='\n';
     op[1] =0xfc; }

      aamfn()     {
     w=2;
     m[3]=' ';m[4]=' ';m[5]='\n';
     op[1] =0xfd; }

     cmcfn()
     {
      w=2;
      m[3]=' ';m[4]=' ';m[5]='\n';
     op[1]=0xff; }
    hltfn()     {
   w=2;
   m[3]=' ';m[4]=' ';m[5]='\n';
   op[1] =0x74; }
locfn()
{
  switch(w) {
    case 0:
           addr=addr+0x04;
           break;
    case 1:
            addr=addr+0x02;
            break;
    default:
            addr=addr+0x01;
            break;
                       }
}
 xorfn()
 {
 p+=4;
 switch(*p++)
 {
 case 'a':
          if(*p++ == 'x')
            xorafn();
            break;
 case 'c':
            if(*p++ =='x')
              xorcfn();
            break;
 case 'd':
           if(*p++ =='x')
            xordfn();
           else
              xordifn();
            break;
 case 'b':
           if(*p++ =='x')
             xorbfn();
          else
              xorbpfn();
             break;

 case 's':
           if(*p++ =='p')
                xorspfn();
           else
               xorsifn();
                break;
  default:
               break;

               }
   }

 xorafn()
 {
  imme();
   switch(w)  {
   case 0 :
   op[1] =0x08<<4;
   op[1] =op[1] ¦0x01;
   op[2] =0x30;
   op[3] =*p;
   *p++;
   op[4] = *p;*p++;
   op[5] = *p;*p++;
   op[6] = *p;
   break;
   case 1:
       op[1] =0x08 << 4;
       op[1] =op[1] ¦0x0b;
   switch(*p++)
   {
   case ´c´:
      if(*p++ == ´x´)
      {
       op[2] =0xc1;
       }
       break;
 case´d´:
       if(*p++ == ´x´)
         {
              op[2] = 0xc2;

             }
         else       {
          op[2] =0xc7;
          }
          break;
case ´b´:
           if(*p++ == ´x´)
           {
            op[2] =0xc3;
             }
           else
           {
             op[2] =0xc5;
             }
             break;
 case ´s´:
         if(*p++ == ´p´)
        {
         op[2] =0xc4;
         }
       else
       {
            op[2] =0xc6;
            }
        break;
        default: printf("error in mnemonic\n");
                 break;
        }
        }
        }
    xorcfn()
    {
      imme();
      switch(w)  {
      case 0 :
      op[1] =0x08 <<4;
      op[1] =op[1] |0x01;
      op[2] =0x31;
      op[3] =*p;
      *p++;
      op[4] = *p;*p++;
      op[5] = *p;*p++;
      op[6] = *p;
      break;
      case 1:
          op[1] =0x08 << 4;
          op[1] =op[1] |0x0b;
      switch(*p++)
      {
      case 'a':
         if(*p++ == 'x')
         {
          op[2] =0xc8;
          }
          break;
    case'd':
          if(*p++ == 'x')
            {
               op[2] = 0xca;
               }
               else
               {
             op[2] =0xcf;
             }
             break;
   case 'b':
              if(*p++ == 'x')
              {
               op[2] =0xcb;
               }
              else
              {
                op[2] =0xcd;
                }
                break;
    case 's':
          if(*p++ == 'p')
         {
          op[2] =0xcc;
          }
        else
        {
          op[2] =0xce;
          }
      break;
      default: printf("error in mnemonic\n");
              break;
      }
      }
      }
  xordfn()
  {
    imme();
    switch(w)  {
    case 0 :
    op[1] =0x08 <<4;
    op[1] =op[1] |0x01;
    op[2] =0xba;
    op[3] =*p;
    *p++;
    op[4] = *p;*p++;
    op[5] = *p;*p++;
    op[6] = *p;
    break;
    case 1 :
        op[1] =0x08 << 4;
        op[1] =op[1] |0x0b;
    switch(*p++)
    {
    case 'a':
       if(*p++ == 'x')
       {
        op[2] =0xd0;
        }
        break;
  case'c':
        if(*p++ == 'x')
          {
             op[2] = 0xd1;
             }
           break;
 case 'b':
             if(*p++ == 'x')
             {
              op[2] =0xd3;
              }
             else
             {
               op[2] =0xd5;
               }
```

```
                break;
     case ´s´:
              if(*p++ == ´p´)
             {
              op[2] =0xd4;
              }
            else
            {
              op[2] =0xd6;
              }
              break;
     case ´d´:
                 if(*p++ == ´i´)
                {
                  op[2] = 0xd7;
                  }
          break;
          default: printf("error in mnemonic\n");
          break;
          }
          }
          }
     xorbfn()
     {
       imme();
       switch(w)  {
       case 0 :
       op[1] =0x08 <<4;
       op[1] =op[1] |0x01;
       op[2] =0x32;
       op[3] =*p;
       *p++;
       op[4] = *p;*p++;
       op[5] = *p;*p++;
       op[6] = *p;
       break;
       case 1:
            op[1] =0x08 << 4;
            op[1] =op[1] |0x0b;
       switch(*p++)
       {
       case ´a´:
          if(*p++ == ´x´)
          {
           op[2] =0xd8;
           }
           break;
    case´c´:
           if(*p++ == ´x´)
             {
                op[2] = 0xd9;

                }
              break;
```

```
case 'b':
           if(*p++ == 'p')
           {
            op[2] =0xdd;
            }
             break;
 case 's':
         if(*p++ == 'p')
        {
         op[2] =0xdc;
         }
       else
       {
         op[2] =0xde;
       }
         break;
 case 'd':
            if(*p++ == 'x')
           {
            op[2] = 0xda;
            }
             else
           {
             op[2] =0xdf;
             }
     break;
     default: printf("error in mnemonic\n");
     break;
     }
     }
     }
 xorspfn()
 {
   imme();
   switch(w)  {
   case 0 :
   op[1] =0x08 <<4;
   op[1] =op[1] |0x01;
   op[2] =0x33;
   op[3] =*p;
   *p++;
   op[4] = *p;*p++;
   op[5] = *p;*p++;
   op[6] = *p;
   break;
   case 1 :
       op[1] =0x08 << 4;
       op[1] =op[1] |0x0b;
   switch(*p++)
   {
   case 'a':
      if(*p++ == 'x')
      {
       op[2] =0xe0;
```

```
        }
        break;
 case'c':
        if(*p++ == 'x')
          {
             op[2] = 0xe1;
             }
           break;
case 'b':
             if(*p++ == 'x')
             {
              op[2] =0xe3;
              }
              else
               {
              op[2] =0xe5;
              }
               break;
 case 's':
           if(*p++ == 'i')
          {
           op[2] =0xe6;
           }
           break;
 case 'd':
             if(*p++ == 'x')
            {
              op[2] = 0xe2;
              }
               else
             {
               op[2] =0xe7;
               }
     break;
     default: printf("error in mnemonic\n");
     break;
     }
     }
     }
 xorbpfn()
 {
   imme();
   switch(w)  {
   case 0 :
   op[1] =0x08 <<4;
   op[1] =op[1] |0x01;
   op[2] =0x34;
   op[3] =*p;
   *p++;
   op[4] = *p;*p++;
   op[5] = *p;*p++;
   op[6] = *p;
   break;
   case 1 :
       op[1] =0x08 << 4;
       op[1] =op[1] |0x0b;
    switch(*p++)
    {
    case 'a':
       if(*p++ == 'x')
       {
        op[2] =0xe8;
        }
        break;
  case'c':
        if(*p++ == 'x')
          {
             op[2] = 0xe9;
             }
           break;
 case 'b':
             if(*p++ == 'x')
             {
              op[2] =0xeb;
              }
               break;
  case 's':
           if(*p++ == 'p')
          {
           op[2] =0xec;
           }
        else
        {
           op[2] =0xee;
        }
           break;
  case 'd':
     .       if(*p++ == 'x')
            {
              op[2] = 0xea;
              }
               else
             {
               op[2] =0xef;
               }
      break;
      default:printf("error in mnemonic\n");
      break;
      }
      }
      }
  xorsifn()
  {
    imme();
    switch(w)  {
    case 0 :
    op[1] =0x08 <<4;
    op[1] =op[1] |0x01;
```

\2

```
   op[2] =0x36;
   op[3] =*p;
   *p++;
   op[4] = *p;*p++;
   op[5] = *p;*p++;
   op[6] = *p;
   break;
   case 1:
        op[1] =0x08 << 4;
        op[1] =op[1] |0x0b;
   switch(*p++)
   {
   case 'a':
       if(*p++ == 'x')
       {
        op[2] =0xf0;
        }
        break;
 case'c':
        if(*p++ == 'x')
          {
             op[2] = 0xf1;

             }
            break;
case 'b':
              if(*p++ == 'x')
              {
               op[2] =0xf3;
               }
               else
                {
               op[2] =0xf5;
               }
                break;
 case 's':
            if(*p++ == 'p')
           {
            op[2] =0xf4;
            }
            break;
 case 'd':
              if(*p++ == 'x')
             {
               op[2] = 0xf2;
               }
                else
              {
                op[2] =0xf7;
                }
       break;
       default: printf("error in mnemonic\n");
       break;
       }
```

```
      }
      }
 xordifn()
 {
   imme();
   switch(w)  {
   case 0 :
   op[1] =0x08 <<4;
   op[1] =op[1] ¦0x01;
   op[2] =0x37;
   op[3] =*p;
   *p++;
   op[4] = *p;*p++;
   op[5] = *p;*p++;
   op[6] = *p;
   break;
   case 1:
       op[1] =0x08 << 4;
       op[1] =op[1] ¦0x0b;
   switch(*p++)
   {
   case 'a':
      if(*p++ == 'x')
      {
       op[2] =0xf8;
       }
       break;
 case'c':
       if(*p++ == 'x')
         {
              op[2] = 0xf9;

              }
           break;
case 'b':
             if(*p++ == 'x')
             {
              op[2] =0xfa;
              }
              else
                 {
              op[2] =0xfd;
              }
               break;
 case 's':
           if(*p++ == 'p')
          {
           op[2] =0xfc;
           }
           else
         {
           op[2] =0xfe;
         }
           break;
```

4

```
case 'd':
          if(*p++ == 'x')
         {
           op[2] = 0xfa;
           }
    break;
    default: printf("error in mnemonic\n");
    break;
    }
    }
    }

 adcfn()
{
p+=4;
switch(*p++)
{
case 'a':
        if(*p++ == 'x')
          adcafn();
          break;
case 'c':
          if(*p++ =='x')
            adccfn();
          break;
case 'd':  if(*p++ =='x')
          adcdfn();
         else
            adcdifn();
          break;
case 'b':
        if(*p++ =='x')
           adcbfn();
        else
            adcbpfn();
           break;

case 's':
         if(*p++ =='p')
              adcspfn();
         else
             adcsifn();
              break;
 default:
             break;

             }
  }

adcafn()
{
   imme();
  switch(w)  {
  case 0 :
     op[1] =0x08 <<4;
     op[1] =op[1] ¦0x03;
     op[2] =0x10;
     op[3] =*p;
     *p++;
     op[4] = *p;*p++;
     op[5] = *p;*p++;
     op[6] = *p;
     break;
     case 1:
         op[1] =0x01 << 4;
         op[1] =op[1] ¦0x03;
     switch(*p++)
     {
     case 'c':
        if(*p++ == 'x')
        {
         op[2] =0xc0;
         }
         break;
   case'd':
         if(*p++ == 'x')
           {
              op[2] = 0xc2;
              }
           else       {
            op[2] =0xc7;
            }
            break;
  case 'b':
             if(*p++ == 'x')
             {
              op[2] =0xc3;
              }
             else
             {
               op[2] =0xc5 <<4;
               }
               break;
   case 's':
           if(*p++ == 'p')
          {
           op[2] =0xc4;
           }
         else
         {
           op[2] =0xc6;
           }
       break;
       default: printf("error in mnemonic\n");
                break;
       }
       }
       }
```

```
adccfn()
 {
     imme();
   switch(w)  {
   case 0 :
   op[1] =0x08 <<4;
   op[1] =op[1] |0x03;
   op[2] =0x11;
   op[3] =*p;
   *p++;
   op[4] = *p;*p++;
   op[5] = *p;*p++;
   op[6] = *p;
   break;
   case 1:
       op[1] =0x01 << 4;
       op[1] =op[1] |0x03;
   switch(*p++)
   {
   case 'a':
      if(*p++ == 'x')
      {
       op[2] =0xc8;
       }
       break;
 case'd':
       if(*p++ == 'x')
         {
            op[2] = 0xca;
            }
            else
            {
          op[2] =0xcf;
          }
          break;
case 'b':
           if(*p++ == 'x')
           {
            op[2] =0xcb;
            }
           else
           {
             op[2] =0xcd;
             }
             break;
 case 's':
         if(*p++ == 'p')
        {
         op[2] =0xcc;
         }
       else
       {
         op[2] =0xce;
         }
     break;
     default: printf("error in mnemonic\n");
              break;
     }
     }
     }
adcdfn()

 {
   imme();
   switch(w)  {
   case 0 :
   op[1] =0x08 <<4;
   op[1] =op[1] ¦0x03;
   op[2] =0x12;
   op[3] =*p;
   *p++;
   op[4] = *p;*p++;
   op[5] = *p;*p++;
   op[6] = *p;
   break;
   case 1:
       op[1] =0x01 << 4;
       op[1] =op[1] ¦0x03;
   switch(*p++)
   {
   case 'a':
      if(*p++ == 'x')
      {
       op[2] =0xd0;
       }
       break;
 case'c':
       if(*p++ == 'x')
         {
             op[2] = 0xd1;
             }
           break;
case 'b':
           if(*p++ == 'x')
           {
            op[2] =0xd3;
            }
           else
           {
             op[2] =0xd5;
             }
             break;
 case 's':
         if(*p++ == 'p')
        {
         op[2] =0xd4;
         }
       else
```

```
          {
            op[2] =0xd6;
            }
            break;
   case 'd':
              if(*p++ == 'i')
             {
               op[2] = 0xd7;
               }
        break;
        default: printf("error in mnemonic\n");
        break;
        }
        }
        }
  adcbfn()
   {
    imme();
     switch(w)  {
     case 0 :
     op[1] =0x08 <<4;
     op[1] =op[1] |0x03;
     op[2] =0x13;
     op[3] =*p;
     *p++;
     op[4] = *p;*p++;
     op[5] = *p;*p++;
     op[6] = *p;
     break;
     case 1:
         op[1] =0x01 << 4;
         op[1] =op[1] |0x03;
     switch(*p++)
     {
     case 'a':
        if(*p++ == 'x')
        {
         op[2] =0xd8;
         }
         break;
   case'c':
         if(*p++ == 'x')
           {
              op[2] = 0xd9;
              }
            break;
  case 'b':
              if(*p++ == 'p')
              {
               op[2] =0xdd;
               }
                break;
   case 's':
           if(*p++ == 'p')
          {
           op[2] =0xdc;
           }
         else
         {
           op[2] =0xde;
         }
           break;
   case 'd':
               if(*p++ == 'x')
              {
                op[2] = 0xda;
                }
                 else
               {
                 op[2] =0xdf;
                 }
       break;
       default: printf("error in mnemonic\n");
       break;
       }
       }
       }
  adcspfn()
   {
    imme();
     switch(w)  {
     case 0 :
     op[1] =0x08 <<4;
     op[1] =op[1] |0x03;
     op[2] =0x14;
     op[3] =*p;
     *p++;
     op[4] = *p;*p++;
     op[5] = *p;*p++;
     op[6] = *p;
     break;
     case 1:
         op[1] =0x01 << 4;
         op[1] =op[1] |0x03;
     switch(*p++)
     {
     case 'a':
        if(*p++ == 'x')
        {
         op[2] =0xe0;
         }
         break;
   case'c':
         if(*p++ == 'x')
           {
              op[2] = 0xe1;
              }
            break;
```

```
case ´b´:
                if(*p++ == ´x´)
                {
                 op[2] =0xe3;
                 }
                 else
                  {
                 op[2] =0xe5;
                 }
                  break;
 case ´s´:
            if(*p++ == ´i´)
           {
            op[2] =0xe6;
            }
            break;
 case ´d´:
               if(*p++ == ´x´)
              {
                op[2] = 0xe2;
                }
                 else
               {
                 op[2] =0xe7;
                 }
      break;
      default: printf("error in mnemonic\n");
      break;
      }
      }
      }
adcbpfn()
 {
  imme();
   switch(w)  {
   case 0 :
   op[1] =0x08 <<4;
   op[1] =op[1] |0x03;
   op[2] =0x15;
   op[3] =*p;
   *p++;
   op[4] = *p;*p++;
   op[5] = *p;*p++;
   op[6] = *p;
   break;
   case 1:
        op[1] =0x01 << 4;
        op[1] =op[1] |0x03;
   switch(*p++)
   {
   case ´a´:
      if(*p++ == ´x´)
      {
       op[2] =0xe8;
```

```
        }
        break;
  case'c':
        if(*p++ == 'x')
          {
             op[2] = 0xe8;
             }
           break;
case 'b':
            if(*p++ == 'x')
            {
             op[2] =0xeb;
             }
              break;
 case 's':
          if(*p++ == 'p')
         {
          op[2] =0xec;
          }
        else
        {
          op[2] =0xee;
        }
          break;
 case 'd':
            if(*p++ == 'x')
           {
             op[2] = 0xea;
             }
              else
            {
              op[2] =0xef;
              }
     break;
     default:printf("error in mnemonic\n");
     break;
     }
     }
     }
adcsifn()
 {
  imme();
   switch(w)  {
   case 0 :
   op[1] =0x08 <<4;
   op[1] =op[1] |0x03;
   op[2] =0x16;
   op[3] =*p;
   *p++;
   op[4] = *p;*p++;
   op[5] = *p;*p++;
   op[6] = *p;
   break;
   case 1:
        op[1] =0x01 << 4;
        op[1] =op[1] ¦0x03;
    switch(*p++)
    {
    case ´a´:
       if(*p++ == ´x´)
       {
        op[2] =0xf0;
        }
        break;
  case´c´:
        if(*p++ == ´x´)
          {
             op[2] = 0xf1;
             }
           break;
 case ´b´:
              if(*p++ == ´x´)
              {
               op[2] =0xf3;
               }
               else
                {
               op[2] =0xf5;
               }
                break;
  case ´s´:
            if(*p++ == ´p´)
           {
            op[2] =0xf4;
            }
            break;
  case ´d´:
              if(*p++ == ´x´)
             {
               op[2] = 0xf2;
               }
                else
              {
                op[2] =0xf7;
                }
     break;
     default: printf("error in mnemonic\n");
     break;
     }
     }
     }
adcdifn()
 {
  imme();
   switch(w)  {
   case 0 :
   op[1] =0x08 <<4;
   op[1] =op[1] ¦0x03;
   op[2] =0x17;
   op[3] =*p;
   *p++;
   op[4] = *p;*p++;
   op[5] = *p;*p++;
   op[6] = *p;
   break;
   case 1:
       op[1] =0x01 << 4;
       op[1] =op[1] |0x03;
   switch(*p++)
   {
   case ´a´:
      if(*p++ == ´x´)
      {
       op[2] =0xf8;
       }
       break;
 case´c´:
       if(*p++ == ´x´)
         {
            op[2] = 0xf9;
            }
          break;
case ´b´:
            if(*p++ == ´x´)
            {
            op[2] =0xfb;
            }
            else
               {
            op[2] =0xfd;
            }
             break;
 case ´s´:
         if(*p++ == ´p´)
        {
         op[2] =0xfc;
         }
         else
       {
         op[2] =0xfe;
       }
         break;
 case ´d´:

            if(*p++ == ´x´)
           {
            op[2] = 0xfe;
            }
     break;
     default: printf("error in mnemonic\n");
     break;
     }
     }
     }
orfn()
{
p+=3;
switch(*p++)
{
case 'a':
         if(*p++ == 'x')
           orafn();
           break;
case 'c':
           if(*p++ =='x')
             orcfn();
           break;
case 'd':
           if(*p++ =='x')
           ordfn();
          else
             ordifn();
           break;
case 'b':
          if(*p++ =='x')
            orbfn();
         else
             orbpfn();
            break;

case 's':
          if(*p++ =='p')
               orspfn();
          else
              orsifn();
               break;
 default:
             break;

             }
  }

orafn()
{
  imme();
  switch(w)  {
  case 0 :
  op[1] =0x08 <<4;
  op[1] =op[1] |0x01;
  op[2] =0x08;
  op[3] =*p;
  *p++;
  op[4] = *p;*p++;
  op[5] = *p;*p++;
  op[6] = *p;
  break;
```

```
    case 1:
        op[1] =0x00 << 4;
        op[1] =op[1] |0x0b;
    switch(*p++)
    {
    case 'c':
       if(*p++ == 'x')
       {
        op[2] =0xc1;
        }
        break;
  case'd':
        if(*p++ == 'x')
          {
             op[2] = 0xc2;
             }
          else       {
           op[2] =0xc7;
           }
           break;
 case 'b':
            if(*p++ == 'x')
            {
             op[2] =0xc3;
             }
            else
            {
              op[2] =0xc5;
              }
              break;
  case 's':
          if(*p++ == 'p')
         {
          op[2] =0xc4;
          }
        else
        {
          op[2] =0xc6;
          }
      break;
      default: printf("error in mnemonic\n");
               break;
      }
      }
      }
  orcfn()
  {
    imme();
    switch(w)  {
    case 0 :
    op[1] =0x08 <<4;
    op[1] =op[1] |0x01;
    op[2] =0x09;
    op[3] =*p;
    *p++;
    op[4] = *p;*p++;
    op[5] = *p;*p++;
    op[6] = *p;
    break;
    case 1:
        op[1] =0x00 << 4;
        op[1] =op[1] |0x0b;
    switch(*p++)
    {
    case 'a':
       if(*p++ == 'x')
       {
        op[2] =0xc8;
        }
        break;
  case'd':
        if(*p++ == 'x')
          {
             op[2] = 0xca;
             }
             else
             {
           op[2] =0xcf;
           }
           break;
 case 'b':
              if(*p++ == 'x')
              {
               op[2] =0xcb;
               }
              else
              {
                op[2] =0xcd;
                }
                break;
  case 's':
          if(*p++ == 'p')
         {
          op[2] =0xcc;
          }
        else
        {
          op[2] =0xce;
          }
      break;
      default: printf("error in mnemonic\n");
               break;
      }
      }
      }
  ordfn()
  {
    imme();
    switch(w)  {
    case 0 :
    op[1] =0x08 <<4;
    op[1] =op[1] |0x01;
    op[2] =0x0a;
    op[3] =*p;
    *p++;
    op[4] = *p;*p++;
    op[5] = *p;*p++;
    op[6] = *p;
    break;
    case 1:
        op[1] =0x00 << 4;
        op[1] =op[1] |0x0b;
    switch(*p++)
    {
    case 'a':
       if(*p++ == 'x')
       {
        op[2] =0xd0;
        }
        break;
  case'c':
        if(*p++ == 'x')
          {
             op[2] = 0xd1;
             }
           break;
 case 'b':
              if(*p++ == 'x')
              {
               op[2] =0xd3;
               }
              else
              {
                op[2] =0xd5;
                }
                break;
  case 's':
           if(*p++ == 'p')
          {
           op[2] =0xd4;
           }
         else
         {
           op[2] =0xd6;
           }
           break;
  case 'd':
             if(*p++ == 'i')
            {
              op[2] = 0xd7;
              }
      break;
    default: printf("error in mnemonic\n");
    break;
    }
    }
      }
orbfn()
{
  imme();
  switch(w)  {
  case 0 :
  op[1] =0x08 <<4;
  op[1] =op[1] |0x01;
  op[2] =0x0b;
  op[3] =*p;
  *p++;
  op[4] = *p;*p++;
  op[5] = *p;*p++;
  op[6] = *p;
  break;
  case 1:
      op[1] =0x00 << 4;
      op[1] =op[1] |0x0b;
  switch(*p++)
  {
  case 'a':
     if(*p++ == 'x')
     {
      op[2] =0xd8;
      }
      break;
 case'c':
      if(*p++ == 'x')
        {
           op[2] = 0xd9;
           }
         break;
case 'b':
          if(*p++ == 'p')
          {
           op[2] =0xdd;
           }
            break;
 case 's':
        if(*p++ == 'p')
       {
        op[2] =0xdc;
        }
      else
      {
        op[2] =0xde;
      }
        break;
 case 'd':
           if(*p++ == 'x')
```

```
              {
                op[2] = 0xda;
                }
                 else
               {
                 op[2] =0xdf;
                 }
        break;
        default: printf("error in mnemonic\n");
        break;
        }
        }
        }
  orspfn()
  {
    imme();
    switch(w)  {
    case 0 :
    op[1] =0x08 <<4;
    op[1] =op[1] !0x01;
    op[2] =0x0c;
    op[3] =*p;
    *p++;
    op[4] = *p;*p++;
    op[5] = *p;*p++;
    op[6] = *p;
    break;
    case 1:
         op[1] =0x00 <<4 ;
         {
         op[2] =0xe0;
         }
         break;
  case'c':
         if(*p++ == 'x')
           {
               op[2] = 0xe1;
               }
             break;
 case 'b':
              if(*p++ == 'x')
              {
               op[2] =0xe3;
               }
               else
                {
               op[2] =0xe5;
               }
                break;
  case 's':
            if(*p++ == 'i')
           {
            op[2] =0xe6;
            }
```

```
          break:
case 'd':
               if(*p++ == 'x')
              {
                op[2] = 0xe2;
                }
                 else
              {
                 op[2] =0xe7;
                 }
      break;
      default: printf("error in mnemonic\n");
      break;
      }
      }
      }
orbpfn()
{
   imme();
   switch(w)  {
   case 0 :
   op[1] =0x08 <<4:
   op[1] =op[1] !0x01:
   op[2] =0x0d:
   op[3] =*p:
   *p++:
   op[4] = *p:*p++:
   op[5] = *p:*p++:
   op[6] = *p;
   break;
   case 1:
        op[1] =0x00 << 4:
        op[1] =op[1] !0x0b:
   switch(*p++)
   {
   case 'a':
      if(*p++ == 'x')
      {
       op[2] =0xe8;
       }
       break;
case'c':
       if(*p++ == 'x')
         {
            op[2] = 0xe9;
            }
          break;
case 'b':
            if(*p++ == 'x')
            {
             op[2] =0xeb;
             }
              break;
 case 's':
```

```
           if(*p++ == 'p')
          {
           op[2] =0xec;
           }
         else
         {
           op[2] =0xee;
         }
           break;
 case 'd':
              if(*p++ == 'x')
             {
               op[2] = 0xea;
               }
                else
              {
                op[2] =0xef;
                }
      break;
      default:printf("error in mnemonic\n");
      break;
      }
      }
      }
orsifn()
 {
   imme();
   switch(w)  {
   case 0 :
   op[1] =0x08 <<4;
   op[1] =op[1] |0x01;
   op[2] =0x0e;
   op[3] =*p;
   *p++;
   op[4] = *p;*p++;
   op[5] = *p;*p++;
   op[6] = *p;
   break;
   case 1:
       op[1] =0x00 << 4;
       op[1] =op[1] |0x0b;
   switch(*p++)
   {
   case 'a':
      if(*p++ == 'x')
      {
       op[2] =0xf0;
       }
       break;
 case'c':
       if(*p++ == 'x')
         {
            op[2] = 0xf1;
            }
           break;
case ´b´:
            if(*p++ == ´x´)
            {
             op[2] =0xf3;
             }
             else
              {
             op[2] =0xf5;
             }
              break;
 case ´s´:
          if(*p++ == ´p´)
         {
          op[2] =0xf4;
          }
          break;
 case ´d´:
            if(*p++ == ´x´)
           {
             op[2] = 0xf2;
             }
              else
            {
              op[2] =0xf7;
              }
     break;
     default: printf("error in mnemonic\n");
     break;
     }
     }
     }
ordifn()
 {
   imme();
   switch(w)  {
   case 0 :
   op[1] =0x08 <<4;
   op[1] =op[1] ¦0x01;
   op[2] =0x0f;
   op[3] =*p;
   *p++;
   op[4] = *p;*p++;
   op[5] = *p;*p++;
   op[6] = *p;
   break;
   case 1:
       op[1] =0x00 << 4;
       op[1] =op[1] ¦0x0b;
   switch(*p++)
   {
   case ´a´:
      if(*p++ == ´x´)
      {
        op[2] =0xf8;
        }
        break;
 case'c':
        if(*p++ == 'x')
          {
             op[2] = 0xf9;
             }
            break;
case 'b':
             if(*p++ == 'x')
             {
              op[2] =0xfb;
              }
              else
                 {
              op[2] =0xfd;
              }
               break;
 case 's':
          if(*p++ == 'p')
         {
          op[2] =0xfc;
          }
          else
        {
          op[2] =0xfe;
        }
          break;
 case 'd':
             if(*p++ == 'x')
            {
              op[2] = 0xfa;
              }
      break;
      default: printf("error in mnemonic\n");
      break;
      }
      }
       }
andfn()
 {
 p+=4;
 switch(*p++)
 {
 case 'a':
          if(*p++ == 'x')
            andafn();
            break;
 case 'c':
            if(*p++ =='x')
              andcfn();
            break;
 case 'd':
```

```
           if(*p++ ==´x´)
            anddfn();
           else
              anddifn();
            break;
case ´b´:
          if(*p++ ==´x´)
             andbfn();
          else
              andbpfn();
             break;

case ´s´:
           if(*p++ ==´p´)
                andspfn();
           else
               andsifn();
                break;
 default:
              break;

              }
  }

andafn()
{
  imme();
  switch(w)  {
  case 0 :
  op[1] =0x08 <<4;
  op[1] =op[1] ¦0x01;
  op[2] =0x20;
  op[3] =*p;
  *p++;
  op[4] = *p;*p++;
  op[5] = *p;*p++;
  op[6] = *p;
  break;
  case 1:
      op[1] =0x02 << 4;
      op[1] =op[1] ¦0x03;
  switch(*p++)
  {
  case ´c´:
     if(*p++ == ´x´)
     {
      op[2] =0xc1;
      }
      break;
case´d´:
      if(*p++ == ´x´)
        {
           op[2] = 0xc2;
           }
          else        {
           op[2] =0xc7;
           }
           break;
case 'b':
             if(*p++ == 'x')
             {
              op[2] =0xc3;
              }
             else
             {
               op[2] =0xc5;
               }
               break;
 case 's':
          if(*p++ == 'p')
         {
          op[2] =0xc4;
          }
        else
        {
          op[2] =0xc6;
          }
     break;
     default: printf("error in mnemonic\n");
              break;
     }
     }
     }
 andcfn()
 {
   imme();
   switch(w)  {
   case 0 :
   op[1] =0x08 <<4;
   op[1] =op[1] |0x01;
   op[2] =0x21;
   op[3] =*p;
   *p++;
   op[4] = *p;*p++;
   op[5] = *p;*p++;
   op[6] = *p;
   break;
   case 1:
       op[1] =0x02 << 4;
       op[1] =op[1] |0x03;
   switch(*p++)
   {
   case 'a':
      if(*p++ == 'x')
      {
       op[2] =0xc8;
       }
       break;
 case´d´:
        if(*p++ == ´x´)
          {
             op[2] = 0xca;
             }
             else
             {
          op[2] =0xcf;
          }
          break;
case ´b´:
            if(*p++ == ´x´)
            {
             op[2] =0xcb;
             }
            else
            {
              op[2] =0xcd;
              }
              break;
 case ´s´:
          if(*p++ == ´p´)
         {
          op[2] =0xc6;
          }
        else
        {
          op[2] =0xce;
          }
      break;
      default: printf("error in mnemonic\n");
                break;
      }
      }
      }
 anddfn()
 {
    imme();
   switch(w)  {
   case 0 :
   op[1] =0x08 <<4;
   op[1] =op[1] |0x01;
   op[2] =0x22;
   op[3] =*p;
      *p++;
   op[4] = *p;*p++;
   op[5] = *p;*p++;
   op[6] = *p;
   break;
   case 1:
       op[1] =0x02 << 4;
       op[1] =op[1] |0x03;
   switch(*p++)
   {
    case 'a':
       if(*p++ == 'x')
       {
        op[2] =0xd0;
        }
        break;
  case'c':
        if(*p++ == 'x')
          {
             op[2] = 0xd1;
             }
          break;
 case 'b':
            if(*p++ == 'x')
            {
             op[2] =0xd3;
             }
            else
            {
              op[2] =0xd5;
              }
              break;
  case 's':
          if(*p++ == 'p')
         {
          op[2] =0xd4;
          }
        else
        {
          op[2] =0xd6;
          }
          break;
  case 'd':
            if(*p++ == 'i')
           {
             op[2] = 0xd7;
             }
      break;
      default: printf("error in mnemonic\n");
      break;
      }
      }
      }
  andbfn()
  {
   imme();
    switch(w)  {
    case 0 :
    op[1] =0x08 <<4;
    op[1] =op[1] |0x01;
    op[2] =0x23;
    op[3] =*p;
    *p++;
    op[4] = *p;*p++;
   op[5] = *p;*p++;
   op[6] = *p;
   break;
   case 1:
       op[1] =0x02 << 4;
       op[1] =op[1] |0x03;
   switch(*p++)
   {
   case 'a':
      if(*p++ == 'x')
      {
       op[2] =0xd8;
       }
       break;
 case'c':
       if(*p++ == 'x')
         {
            op[2] = 0xd9;
            }
           break;
case 'b':
            if(*p++ == 'p')
            {
             op[2] =0xdd;
             }
              break;
 case 's':
          if(*p++ == 'p')
         {
          op[2] =0xdc;
          }
        else
        {
          op[2] =0xde;
        }
          break;
 case 'd':
            if(*p++ == 'x')
           {
             op[2] = 0xda;
             }
              else
            {
              op[2] =0xdf;
              }
     break;
     default: printf("error in mnemonic\n");
     break;
     }
     }
     }
 andspfn()
 {
   imme();
```

```
      switch(w)  {
      case 0 :
      op[1] =0x08 <<4;
      op[1] =op[1] ¦0x01;
      op[2] =0x24;
      op[3] =*p;
      *p++;
      op[4] = *p;*p++;
      op[5] = *p;*p++;
      op[6] = *p;
      break;
      case 1:
          op[1] =0x02 << 4;
          op[1] =op[1] ¦0x03;
      switch(*p++)
      {
      case ´a´:
         if(*p++ == ´x´)
         {
          op[2] =0xe0;
          }
          break;
    case´c´:
          if(*p++ == ´x´)
            {
               op[2] = 0xe1;
               }
             break;
   case ´b´:
              if(*p++ == ´x´)
              {
               op[2] =0xe3;
               }
               else
                {
               op[2] =0xe5;
               }
        break;
    case ´s´:
            if(*p++ == ´i´)
           {
            op[2] =0xe6;
            }
            break;
    case ´d´:
              if(*p++ == ´x´)
             {
               op[2] = 0xe2;
               }
                else
              {
                op[2] =0xe7;
                }
       break;
```

```
        default: printf("error in mnemonic\n");
        break;
        }
        }
        }
    andbpfn()
    {
       imme();
      switch(w)  {
      case 0 :
      op[1] =0x08 <<4;
      op[1] =op[1] ¦0x01;
      op[2] =0x25;
      op[3] =*p;
      *p++;
      op[4] = *p;*p++;
      op[5] = *p;*p++;
      op[6] = *p;
      break;
      case 1:
          op[1] =0x02 << 4;
          op[1] =op[1] ¦0x03;
      switch(*p++)
      {
      case 'a':
         if(*p++ == 'x')
         {
          op[2] =0xe8;
          }
          break;
    case'c':
          if(*p++ == 'x')
            {
               op[2] = 0xe9;
               }
             break;
   case 'b':
               if(*p++ == 'x')
               {
                op[2] =0xeb;
                }
                 break;
    case 's':
            if(*p++ == 'p')
           {
            op[2] =0xec;
            }
          else
          {
            op[2] =0xee;
          }
            break;
    case 'd':
               if(*p++ == 'x')
           {
             op[2] = 0xea;
             }
              else
            {
              op[2] =0xef;
              }
      break;
      default:printf("error in mnemonic\n");
      break;
      }
      }
      }
 andsifn()
  {
    imme();
    switch(w)  {
    case 0 :
    op[1] =0x08 <<4;
    op[1] =op[1] ¦0x01;
    op[2] =0x26;
    op[3] =*p;
    *p++;
    op[4] = *p;*p++;
    op[5] = *p;*p++;
    op[6] = *p;
    break;
    case 1:
        op[1] =0x02 << 4;
        op[1] =op[1] ¦0x03;
    switch(*p++)
    {
    case 'a':
       if(*p++ == 'x')
       {
        op[2] =0xf0;
        }
        break;
  case'c':
        if(*p++ == 'x')
          {
            op[2] = 0xf1;
            }
           break;
 case 'b':
             if(*p++ == 'x')
             {
              op[2] =0xf3;
              }
              else
               {
              op[2] =0xf5;
              }
               break;
```

12

```
case 's':
        if(*p++ == 'p')
       {
        op[2] =0xf4;
        }
        break;
case 'd':
          if(*p++ == 'x')
         {
           op[2] = 0xf2;
           }
            else
          {
            op[2] =0xf7;
            }
    break;
    default: printf("error in mnemonic\n");
    break;
    }
    }
    }
anddifn()
 {
  imme();
   switch(w)  {
   case 0 :
   op[1] =0x08 <<4;
   op[1] =op[1] |0x01;
   op[2] =0x27;
   op[3] =*p;
   *p++;
   op[4] = *p;*p++;
   op[5] = *p;*p++;
   op[6] = *p;
   break;
   case 1:
       op[1] =0x02 << 4;
       op[1] =op[1] |0x03;
   switch(*p++)
   {
   case 'a':
      if(*p++ == 'x')
      {
       op[2] =0xf8;
       }
       break;
 case'c':
       if(*p++ == 'x')
         {
            op[2] = 0xf9;
            }
          break;
case 'b':
           if(*p++ == 'x')
```

```
              {
               op[2] =0xfb;
               }
               else
                  {
               op[2] =0xfd;
               }
                break;
case 's':
            if(*p++ == 'p')
           {
            op[2] =0xfc;
            }
            else
          {
            op[2] =0xfe;
          }
            break;
case 'd':
              if(*p++ == 'x')
             {
               op[2] = 0xfa;
               }
    break;
    default: printf("error in mnemonic\n");
    break;
    }
    }
    }
sbbfn()
{
p+=4;
switch(*p++)
{
case 'a':
            if(*p++ == 'x')
              sbbafn();
              break;
case 'c':
              if(*p++ =='x')
                sbbcfn();
              break;
case 'd':
              if(*p++ =='x')
              sbbdfn();
             else
                sbbdifn();
              break;
case 'b':
            if(*p++ =='x')
               sbbbfn();
            else
                sbbbpfn();
               break;
```

```
 case ´s´:
           if(*p++ ==´p´)
                sbbspfn();
           else
               sbbsifn();
                break;
  default:
              break;

              }
   }

 sbbafn()
 {
   imme();
   switch(w)  {
   case 0 :
   op[1] =0x08 <<4;
   op[1] =op[1] ¦0x03;
   op[2] =0x18;
   op[3] =*p;
   *p++;
   op[4] = *p;*p++;
   op[5] = *p;*p++;
   op[6] = *p;
   break;
   case 1:
       op[1] =0x01<< 4;
       op[1] =op[1] ¦0x0b;
   switch(*p++)
   {
   case ´c´:
      if(*p++ == ´x´)
      {
       op[2] =0xc1;
       }
       break;
 case´d´:
       if(*p++ == ´x´)
         {
            op[2] = 0xc2;

            }
         else        {
          op[2] =0xc7;
          }
          break;
case ´b´:
           if(*p++ == ´x´)
           {
            op[2] =0xc3;
            }
           else
```

```
                {
                  op[2] =0xc5;
                  }
                  break;
case 's':
              if(*p++ == 'p')
             {
              op[2] =0xc4;
              }
            else
            {
              op[2] =0xc6;
              }
        break;
        default: printf("error in mnemonic\n");
                 break;
        }
        }
        }
sbbcfn()
{
 imme();
  switch(w)  {
  case 0 :
  op[1] =0x08 <<4;
  op[1] =op[1] |0x03;
  op[2] =0x19;
  op[3] =*p;
  *p++;
  op[4] = *p;*p++;
  op[5] = *p;*p++;
  op[6] = *p;
  break;
  case 1:
      op[1] =0x01 << 4;
      op[1] =op[1] |0x0b;
  switch(*p++)
  {
  case 'a':
     if(*p++ == 'x')
     {
      op[2] =0xc8;
      }
      break;
case'd':
      if(*p++ == 'x')
        {
           op[2] = 0xca;
           }
           else
           {
         op[2] =0xcf;
         }
         break;
```

```
case 'b':
              if(*p++ == 'x')
              {
               op[2] =0xcb;
               }
              else
              {
                op[2] =0xcd;
                }
                break;
 case 's':
           if(*p++ == 'p')
          {
           op[2] =0xcc;
           }
         else
         {
           op[2] =0xce;
           }
      break;
      default: printf("error in mnemonic\n");
                break;
      }
      }
      }
 sbbdfn()
 {
   imme();
   switch(w)  {
   case 0 :
   op[1] =0x08 <<4;
   op[1] =op[1] |0x03;
   op[2] =0x1a;
   op[3] =*p;
   *p++;
   op[4] = *p;*p++;
   op[5] = *p;*p++;
   op[6] = *p;
   break;
   case 1 :
        op[1] =0x01 << 4;
        op[1] =op[1] |0x0b;
   switch(*p++)
   {
   case 'a':
       if(*p++ == 'x')
       {
        op[2] =0xd0;
        }
        break;
case'c':
        if(*p++ == 'x')
          {
              op[2] = 0xd1;
```

```
               }
             break;
case 'b':
              if(*p++ == 'x')
              {
               op[2] =0xd3;
               }
              else
              {
                op[2] =0xd5;
                }
                break;
 case 's':
            if(*p++ == 'p')
           {
            op[2] =0xd4;
            }
           else
           {
            op[2] =0xd6;
            }
            break;
 case 'd':
              if(*p++ == 'i')
             {
               op[2] = 0xd7;
               }
        break;
        default: printf("error in mnemonic\n");
        break;
        }
        }
        }
 sbbbfn()
 {
   imme();
   switch(w)  {
   case 0 :
   op[1] =0x08 <<4;
   op[1] =op[1] ¦0x03;
   op[2] =0x1b;
   op[3] =*p;
   *p++;
   op[4] = *p;*p++;
   op[5] = *p;*p++;
   op[6] = *p;
   break;
   case 1:
       op[1] =0x01 << 4;
       op[1] =op[1] ¦0x0b;
   switch(*p++)
   {
   case 'a':
      if(*p++ == 'x')
```

```
        {
         op[2] =0xd8;
         }
         break;
   case´c´:
         if(*p++ == ´x´)
           {
              op[2] = 0xd9;

              }
            break;
  case ´b´:
              if(*p++ == ´p´)
              {
               op[2] =0xdd;
               }
                break;
   case  ´s´:
           if(*p++ == ´p´)
          {
           op[2] =0xdc;
           }
         else
         {
           op[2] =0xde;
         }
           break;
   case  ´d´:
              if(*p++ == ´x´)
             {
               op[2] = 0xda;
               }
                else
              {
                op[2] =0xdf;
                }
       break;
       default: printf("error in mnemonic\n");
       break;
       }
       }
       }
   sbbspfn()
   {
     imme();
     switch(w)  {
     case 0 :
     op[1] =0x08 <<4;
     op[1] =op[1] ¦0x03;
     op[2] =0x1c;
     op[3] =*p;
     *p++;
     op[4] = *p;*p++;
     op[5] = *p;*p++;
```

```
    op[6] = *p;
    break;
    case 1 :
        op[1] =0x01 << 4;
        op[1] =op[1] |0x0b;
    switch(*p++)
    {
    case ´a´:
       if(*p++ == ´x´)
       {
        op[2] =0xe0;
        }
        break;
 case´c´:
        if(*p++ == ´x´)
          {
             op[2] = 0xe1;
             }
           break;
case ´b´:
            if(*p++ == ´x´)
            {
             op[2] =0xe3;
             }
             else
              {
             op[2] =0xe5;
             }
              break;
 case ´s´:
          if(*p++ == ´i´)
         {
          op[2] =0xe6;
          }
          break;
 case ´d´:
            if(*p++ == ´x´)
           {
            op[2] = 0xe2;
            }
             else
            {
             op[2] =0xe7;
             }
     break;
     default: printf("error in mnemonic\n");
     break;
     }
     }
     }
 sbbbpfn()
 {
   imme();
   switch(w)  {
      case 0 :
      op[1] =0x08 <<4;
      op[1] =op[1] |0x03;
      op[2] =0x1d;
      op[3] =*p;
      *p++;
      op[4] = *p;*p++;
      op[5] = *p;*p++;
      op[6] = *p;
      break;
      case 1 :
          op[1] =0x01 << 4;
          op[1] =op[1] |0x0b;
      switch(*p++)
      {
      case 'a':
         if(*p++ == 'x')
         {
          op[2] =0xe8;
          }
          break;
    case'c':
          if(*p++ == 'x')
            {
               op[2] = 0xe9;
               }
             break;
   case 'b':
              if(*p++ == 'x')
              {
               op[2] =0xeb;
               }
                break;
   case 's':
            if(*p++ == 'p')
           {
            op[2] =0xec;
            }
          else
          {
            op[2] =0xee;
          }
            break;
   case 'd':
             if(*p++ == 'x')
            {
              op[2] = 0xea;
              }
               else
             {
               op[2] =0xef;
               }
       break;
       default:printf("error in mnemonic\n");
      break;
      }
      }
      }
 sbbsifn()
 {
   imme();
   switch(w)  {
   case 0 :
   op[1] =0x08 <<4;
   op[1] =op[1] |0x03;
   op[2] =0x1e;
   op[3] =*p;
   *p++;
   op[4] = *p;*p++;
   op[5] = *p;*p++;
   op[6] = *p;
   break;
   case 1:
       op[1] =0x01 << 4;
       op[1] =op[1] |0x0b;
   switch(*p++)
   {
   case 'a':
      if(*p++ == 'x')
      {
       op[2] =0xf0;
       }
       break;
 case'c':
       if(*p++ == 'x')
         {
            op[2] = 0xf1;

            }
           break;
case 'b':
             if(*p++ == 'x')
             {
              op[2] =0xf3;
              }
              else
               {
              op[2] =0xf5;
              }
               break;
 case 's':
          if(*p++ == 'p')
         {
          op[2] =0xf4;
          }
          break;
 case 'd':
             if(*p++ == 'x')
```

```
                {
                  op[2] = 0xf2;
                  }
                   else
                 {
                   op[2] =0xf7;
                   }
          break;
          default: printf("error in mnemonic\n");
          break;
          }
          }
          }
      sbbdifn()
      {
        imme();
        switch(w)  {
        case 0 :
        op[1] =0x08 <<4;
        op[1] =op[1] |0x03;
        op[2] =0x1f;
        op[3] =*p;
        *p++;
        op[4] = *p;*p++;
        op[5] = *p;*p++;
        op[6] = *p;
        break;
        case 1:
            op[1] =0x01 << 4;
            op[1] =op[1] |0x0b;
        switch(*p++)
        {
        case 'a':
           if(*p++ == 'x')
           {
            op[2] =0xf8;
            }
            break;
    case'c':
            if(*p++ == 'x')
              {
                 op[2] = 0xf9;

                 }
              break;
  case 'b':
               if(*p++ == 'x')
               {
                op[2] =0xfa;
                }
                else
                   {
                op[2] =0xfd;
                }
```

```
                  break;
case ´s´:
              if(*p++ == ´p´)
             {
              op[2] =0xfc;
              }
              else
           {
              op[2] =0xfe;
           }
              break;
case ´d´:
                if(*p++ == ´x´)
               {
                 op[2] = 0xfa;
                 }
     break;
     default: printf("error in mnemonic\n");
     break;
     }
     }
     }
```

B. 8086 INSTRUCTION SET

Translation of Assembler Instructions

Machine code for the 8086/8088 instructions. (Reprinted by permission of Intel Corporation. Copyright 1979.)

AL = 8-bit accumulator
AX = 16-bit accumulator
CX = Count register
DS = Data segment
ES = Extra segment
Above/below refers to unsigned value
Greater = more positive;
Less = less positive (more negative) signed values
if d = 1 then "to" reg; if d = 0 then "from" reg
if w = 1 then word instruction; if w = 0 then byte instruction

if s w = 01 then 16 bits of immediate data form the operand
if s w = 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" in (CL)
x = don't care
z is used for string primitives for comparison with ZF FLAG

SEGMENT OVERRIDE PREFIX

0 0 1 reg 1 1 0

if mod = 11 then r/m is treated as a REG field
if mod = 00 then DISP = 0*, disp-low and disp-high are absent
if mod = 01 then DISP = disp-low sign-extended to 16-bits, disp-high is absent
if mod = 10 then DISP = disp-high: disp-low

if r/m = 000 then EA = (BX) + (SI) + DISP
if r/m = 001 then EA = (BX) + (DI) + DISP
if r/m = 010 then EA = (BP) + (SI) + DISP
if r/m = 011 then EA = (BP) + (DI) + DISP
if r/m = 100 then EA = (SI) + DISP
if r/m = 101 then EA = (DI) + DISP
if r/m = 110 then EA = (BP) + DISP*
if r/m = 111 then EA = (BX) + DISP
DISP follows 2nd byte of instruction (before data if required)

*except if mod = 00 and r/m = 110 then EA = disp-high: disp-low

REG is assigned according to the following table

16 Bit (w = 1)		8 Bit (w = 0)		Segment	
000	AX	000	AL	00	ES
001	CX	001	CL	01	CS
010	DX	010	DL	10	SS
011	BX	011	BL	11	DS
100	SP	100	AH		
101	BP	101	CH		
110	SI	110	DH		
111	DI	111	BH		

Instructions which reference the flag register file as a 16-bit object use the symbol FLAGS to represent the file

FLAGS = X X X X (OF) (DF) (IF) (TF) (SF) (ZF) X (AF) X (PF) X (CF)

8086/8088 Instruction Encoding

DATA TRANSFER

MOV = Move:

	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0
Register/memory to/from register	1 0 0 0 1 0 d w	mod reg r/m	(DISP-LO)	(DISP-HI)		
Immediate to register/memory	1 1 0 0 0 1 1 w	mod 0 0 0 r/m	(DISP-LO)	(DISP-HI)	data	data if w = 1
Immediate to register	1 0 1 1 w reg	data	data if w = 1			
Memory to accumulator	1 0 1 0 0 0 0 w	addr-lo	addr-hi			
Accumulator to memory	1 0 1 0 0 0 1 w	addr-lo	addr-hi			
Register/memory to segment register	1 0 0 0 1 1 1 0	mod 0 SR r/m	(DISP-LO)	(DISP-HI)		
Segment register to register/memory	1 0 0 0 1 1 0 0	mod 0 SR r/m	(DISP-LO)	(DISP-HI)		

PUSH = Push:

Register/memory	1 1 1 1 1 1 1 1	mod 1 1 0 r/m	(DISP-LO)	(DISP-HI)
Register	0 1 0 1 0 reg			
Segment register	0 0 0 reg 1 1 0			

POP = Pop:

Register/memory	1 0 0 0 1 1 1 1	mod 0 0 0 r/m	(DISP-LO)	(DISP-HI)
Register	0 1 0 1 1 reg			
Segment register	0 0 0 reg 1 1 1			

Continued.

DATA TRANSFER (Cont'd.)

	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0
XCHG = Exchange:						
Register/memory with register	1 0 0 0 0 1 1 w	mod reg r/m	(DISP-LO)	(DISP-HI)		
Register with accumulator	1 0 0 1 0 reg					
IN = Input from:						
Fixed port	1 1 1 0 0 1 0 w	DATA 8				
Variable port	1 1 1 0 1 1 0 w					
OUT = Output to:						
Fixed port	1 1 1 0 0 1 1 w	DATA-8				
Variable port	1 1 1 0 1 1 1 w					
XLAT = Translate byte to AL	1 1 0 1 0 1 1 1					
LEA = Load EA to register	1 0 0 0 1 1 0 1	mod reg r/m	(DISP-LO)	(DISP-HI)		
LDS = Load pointer to DS	1 1 0 0 0 1 0 1	mod reg r/m	(DISP-LO)	(DISP-HI)		
LES = Load pointer to ES	1 1 0 0 0 1 0 0	mod reg r/m	(DISP-LO)	(DISP-HI)		
LAHF = Load AH with flags	1 0 0 1 1 1 1 1					
SAHF = Store AH into flags	1 0 0 1 1 1 1 0					
PUSHF = Push flags	1 0 0 1 1 1 0 0					
POPF = Pop flags	1 0 0 1 1 1 0 1					

ARITHMETIC

ADD = Add:						
Reg/memory with register to either	0 0 0 0 0 0 d w	mod reg r/m	(DISP-LO)	(DISP-HI)		
Immediate to register/memory	1 0 0 0 0 0 s w	mod 0 0 0 r/m	(DISP-LO)	(DISP-HI)	data	data if s: w=01
Immediate to accumulator	0 0 0 0 0 1 0 w	data	data if w=1			
ADC = Add with carry:						
Reg/memory with register to either	0 0 0 1 0 0 d w	mod. reg r/m	(DISP-LO)	(DISP-HI)		
Immediate to register/memory	1 0 0 0 0 0 s w	mod 0 1 0 r/m	(DISP-LO)	(DISP-HI)	data	data if s: w=01
Immediate to accumulator	0 0 0 1 0 1 0 w	data	data if w=1			
INC = Increment:						
Register/memory	1 1 1 1 1 1 1 w	mod 0 0 0 r/m	(DISP-LO)	(DISP-HI)		
Register	0 1 0 0 0 reg					
AAA = ASCII adjust for add	0 0 1 1 0 1 1 1					
DAA = Decimal adjust for add	0 0 1 0 0 1 1 1					

Continued.

ARITHMETIC (Cont'd.)

SUB = Subtract:	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0
Reg/memory and register to either	0 0 1 0 1 0 d w	mod reg r/m	(DISP-LO)	(DISP-HI)		
Immediate from register/memory	1 0 0 0 0 0 s w	mod 1 0 1 r/m	(DISP-LO)	(DISP-HI)	data	data if s: w=01
Immediate from accumulator	0 0 1 0 1 1 0 w	data	data if w=1			
SBB = Subtract with borrow:						
Reg/memory and register to either	0 0 0 1 1 0 d w	mod reg r/m	(DISP-LO)	(DISP-HI)		
Immediate from register/memory	1 0 0 0 0 0 s w	mod 0 1 1 r/m	(DISP-LO)	(DISP-HI)	data	data if s: w=01
Immediate from accumulator	0 0 0 1 1 1 0 w	data	data if w=1			
DEC Decrement:						
Register/memory	1 1 1 1 1 1 1 w	mod 0 0 1 r/m	(DISP-LO)	(DISP-HI)		
Register	0 1 0 0 1 reg					
NEG Change sign	1 1 1 1 0 1 1 w	mod 0 1 1 r/m	(DISP-LO)	(DISP-HI)		
CMP = Compare.						
Register/memory and register	0 0 1 1 1 0 d w	mod reg r/m	(DISP-LO)	(DISP-HI)		
Immediate with register/memory	1 0 0 0 0 0 s w	mod 1 1 1 r/m	(DISP-LO)	(DISP-HI)	data	data if s: w=01
Immediate with accumulator	0 0 1 1 1 1 0 w	data	data if w=1			
AAS ASCII adjust for subtract	0 0 1 1 1 1 1 1					
DAS Decimal adjust for subtract	0 0 1 0 1 1 1 1					
MUL Multiply (unsigned)	1 1 1 1 0 1 1 w	mod 1 0 0 r/m	(DISP-LO)	(DISP-HI)		
IMUL Integer multiply (signed)	1 1 1 1 0 1 1 w	mod 1 0 1 r/m	(DISP-LO)	(DISP-HI)		
AAM ASCII adjust for multiply	1 1 0 1 0 1 0 0	0 0 0 0 1 0 1 0				
DIV Divide (unsigned)	1 1 1 1 0 1 1 w	mod 1 1 0 r/m	(DISP-LO)	(DISP-HI)		
IDIV Integer divide (signed)	1 1 1 1 0 1 1 w	mod 1 1 1 r/m	(DISP-LO)	(DISP-HI)		
AAD ASCII adjust for divide	1 1 0 1 0 1 0 1	0 0 0 0 1 0 1 0				
CBW Convert byte to word	1 0 0 1 1 0 0 0					
CWD Convert word to double word	1 0 0 1 1 0 0 1					

LOGIC

NOT Invert	1 1 1 1 0 1 1 w	mod 0 1 0 r/m	(DISP-LO)	(DISP-HI)
SHL/SAL Shift logical/arithmetic left	1 1 0 1 0 0 v w	mod 1 0 0 r/m	(DISP-LO)	(DISP-HI)
SHR Shift logical right	1 1 0 1 0 0 v w	mod 1 0 1 r/m	(DISP-LO)	(DISP-HI)
SAR Shift arithmetic right	1 1 0 1 0 0 v w	mod 1 1 1 r/m	(DISP-LO)	(DISP-HI)
ROL Rotate left	1 1 0 1 0 0 v w	mod 0 0 0 r/m	(DISP-LO)	(DISP-HI)

Continued.

LOGIC (Cont'd)	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0
ROR Rotate right	1 1 0 1 0 0 v w	mod 0 0 1 r/m	(DISP-LO)	(DISP-HI)		
RCL Rotate through carry flag left	1 1 0 1 0 0 v w	mod 0 1 0 r/m	(DISP-LO)	(DISP-HI)		
RCR Rotate through carry right	1 1 0 1 0 0 v w	mod 0 1 1 r/m	(DISP-LO)	(DISP-HI)		

AND = And						
Reg/memory with register to either	0 0 1 0 0 0 d w	mod reg r/m	(DISP-LO)	(DISP-HI)		
Immediate to register/memory	1 0 0 0 0 0 0 w	mod 1 0 0 r/m	(DISP-LO)	(DISP-HI)	data	data if w=1
Immediate to accumulator	0 0 1 0 0 1 0 w	data	data if w=1			

TEST = And function to flags no result:						
Register/memory and register	1 0 0 0 0 1 0 w	mod reg r/m	(DISP-LO)	(DISP-HI)		
Immediate data and register/memory	1 1 1 1 0 1 1 w	mod 0 0 0 r/m	(DISP-LO)	(DISP-HI)	data	data if w=1
Immediate data and accumulator	1 0 1 0 1 0 0 w	data	data if w=1			

OR = Or:						
Reg/memory and register to either	0 0 0 0 1 0 d w	mod reg r/m	(DISP-LO)	(DISP-HI)		
Immediate to register/memory	1 0 0 0 0 0 0 w	mod 0 0 1 r/m	(DISP-LO)	(DISP-HI)	data	data if w=1
Immediate to accumulator	0 0 0 0 1 1 0 w	data	data if w=1			

XOR = Exclusive or:						
Reg/memory and register to either	0 0 1 1 0 0 d w	mod reg r/m	(DISP-LO)	(DISP-HI)		
Immediate to register/memory	1 0 0 0 0 0 0 w	mod 1 1 0 r/m	(DISP-LO)	(DISP-HI)	data	data if w=1
Immediate to accumulator	0 0 1 1 0 1 0 w	data	data if w=1			

STRING MANIPULATION	
REP = Repeat	1 1 1 1 0 0 1 z
MOVS = Move byte/word	1 0 1 0 0 1 0 w
CMPS = Compare byte/word	1 0 1 0 0 1 1 w
SCAS = Scan byte/word	1 0 1 0 1 1 1 w
LODS = Load byte/wd to AL/AX	1 0 1 0 1 1 0 w
STDS = Stor byte/wd from AL/A	1 0 1 0 1 0 1 w

Continued

CONTROL TRANSFER						
CALL = Call:	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0
Direct within segment	1 1 1 0 1 0 0 0	IP-INC-LO	IP-INC HI			
Indirect within segment	1 1 1 1 1 1 1 1	mod 0 1 0 r/m	(DISP-LO)	(DISP-HI)		
Direct intersegment	1 0 0 1 1 0 1 0	IP-lo	IP-hi			
		CS-lo	CS hi			
Indirect intersegment	1 1 1 1 1 1 1 1	mod 0 1 1 r/m	(DISP-LO)	(DISP-HI)		
JMP = Unconditional Jump:						
Direct within segment	1 1 1 0 1 0 0 1	IP-INC-LO	IP-INC-HI			
Direct within segment-short	1 1 1 0 1 0 1 1	IP-INC8				
Indirect within segment	1 1 1 1 1 1 1 1	mod 1 0 0 r/m	(DISP-LO)	(DISP-HI)		
Direct intersegment	1 1 1 0 1 0 1 0	IP-lo	IP-hi			
		CS lo	CS-hi			
Indirect intersegment	1 1 1 1 1 1 1 1	mod 1 0 1 r/m	(DISP LO)	(DISP HI)		
RET = Return from CALL:						
Within segment	1 1 0 0 0 0 1 1					
Within seg adding immed to SP	1 1 0 0 0 0 1 0	data-lo	data-hi			
Intersegment	1 1 0 0 1 0 1 1					
Intersegment adding immediate to SP	1 1 0 0 1 0 1 0	data-lo	data hi			
JE/JZ = Jump on equal/zero	0 1 1 1 0 1 0 0	IP-INC8				
JL/JNGE = Jump on less/not greater or equal	0 1 1 1 1 1 0 0	IP-INC8				
JLE/JNG = Jump on less or equal/not greater	0 1 1 1 1 1 1 0	IP-INC8				
JB/JNAE = Jump on below/not above or equal	0 1 1 1 0 0 1 0	IP-INC8				
JBE/JNA = Jump on below or equal/not above	0 1 1 1 0 1 1 0	IP-INC8				
JP/JPE = Jump on parity/parity even	0 1 1 1 1 0 1 0	IP-INC8				
JO = Jump on overflow	0 1 1 1 0 0 0 0	IP-INC8				
JS = Jump on sign	0 1 1 1 1 0 0 0	IP-INC8				
JNE/JNZ = Jump on not equal/not zer0	0 1 1 1 0 1 0 1	IP-INC8				
JNL/JGE = Jump on not less/greater or equal	0 1 1 1 1 1 0 1	IP INC8				
JNLE/JG = Jump on not less or equal/greater	0 1 1 1 1 1 1 1	IP-INC8				
JNB/JAE = Jump on not below/above or equal	0 1 1 1 0 0 1 1	IP-INC8				
JNBE/JA = Jump on not below or equal/above	0 1 1 1 0 1 1 1	IP INC8				
JNP/JPO = Jump on not par/par odd	0 1 1 1 1 0 1 1	IP INC8				
JNO = Jump on not overflow	0 1 1 1 0 0 0 1	IP-INC8				

Continued.

CONTROL TRANSFER (Cont'd.)

	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0
JNS = Jump on not sign	0 1 1 1 1 0 0 1	IP INC8				
LOOP = Loop CX times	1 1 1 0 0 0 1 0	IP INC8				
LOOPZ/LOOPE = Loop while zero/equal	1 1 1 0 0 0 0 1	IP-INC8				
LOOPNZ/LOOPNE = Loop while not zero/equal	1 1 1 0 0 0 0 0	IP INC8				
JCXZ = Jump on CX zero	1 1 1 0 0 0 1 1	IP-INC8				

INT = Interrupt:

Type specified	1 1 0 0 1 1 0 1	DATA-8
Type 3	1 1 0 0 1 1 0 0	
INTO = Interrupt on overflow	1 1 0 0 1 1 1 0	
IRET = Interrupt return	1 1 0 0 1 1 1 1	

PROCESSOR CONTROL

CLC = Clear carry	1 1 1 1 1 0 0 0			
CMC = Complement carry	1 1 1 1 0 1 0 1			
STC = Set carry	1 1 1 1 1 0 0 1			
CLD = Clear direction	1 1 1 1 1 1 0 0			
STD = Set direction	1 1 1 1 1 1 0 1			
CLI = Clear interrupt	1 1 1 1 1 0 1 0			
STI = Set interrupt	1 1 1 1 1 0 1 1			
HLT = Halt	1 1 1 1 0 1 0 0			
WAIT = Wait	1 0 0 1 1 0 1 1			
ESC = Escape (to external device)	1 1 0 1 1 x x x	m o d y y y r / m	(DISP-LO)	(DISP-HI)
LOCK = Bus lock prefix	1 1 1 1 0 0 0 0			
SEGMENT = Override prefix	0 0 1 reg 1 1 0			

would be assembled as

00111101	10100010	00000101

instead of

10000001	11111000	10100010	00000101

C. DIAGRAMS

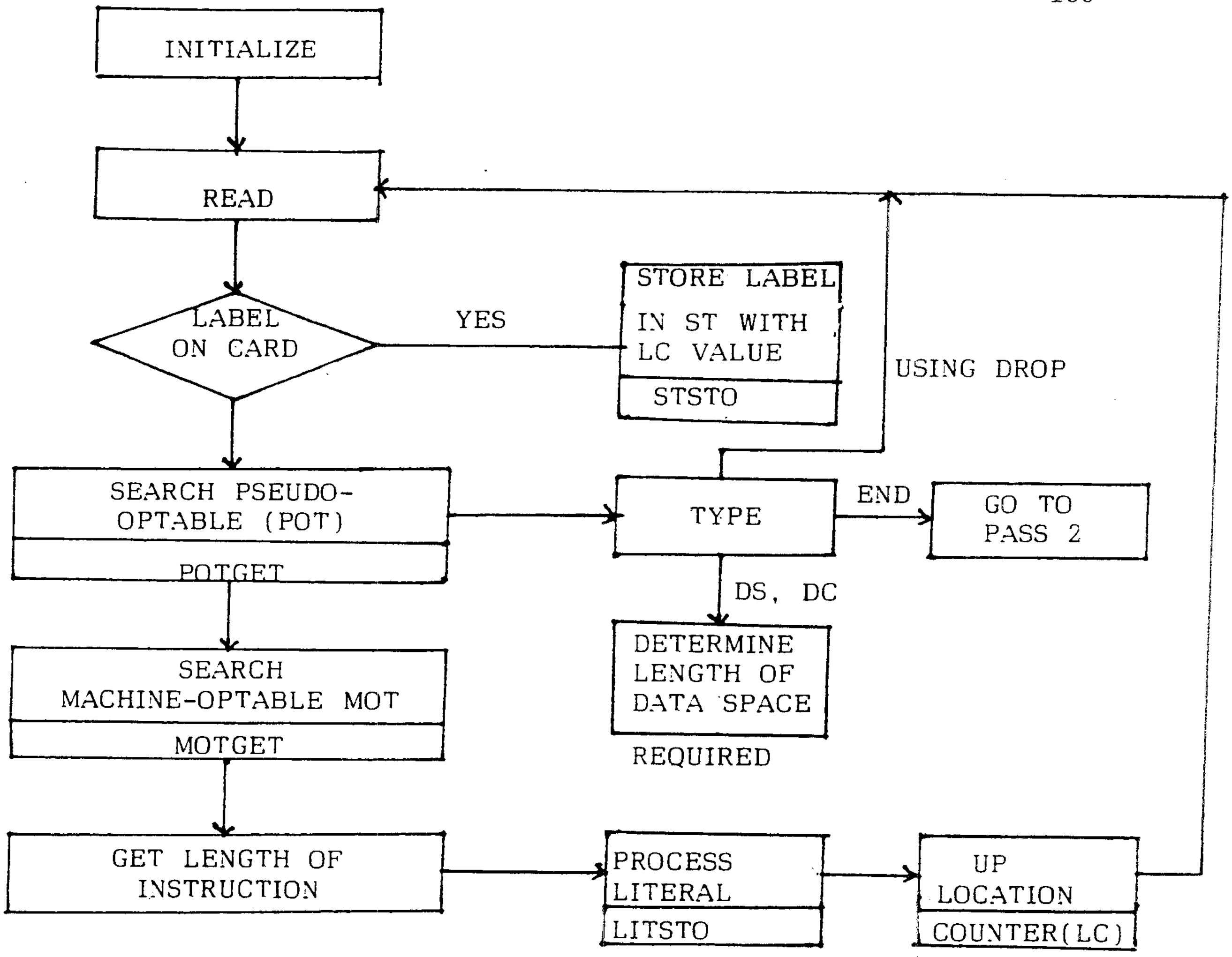
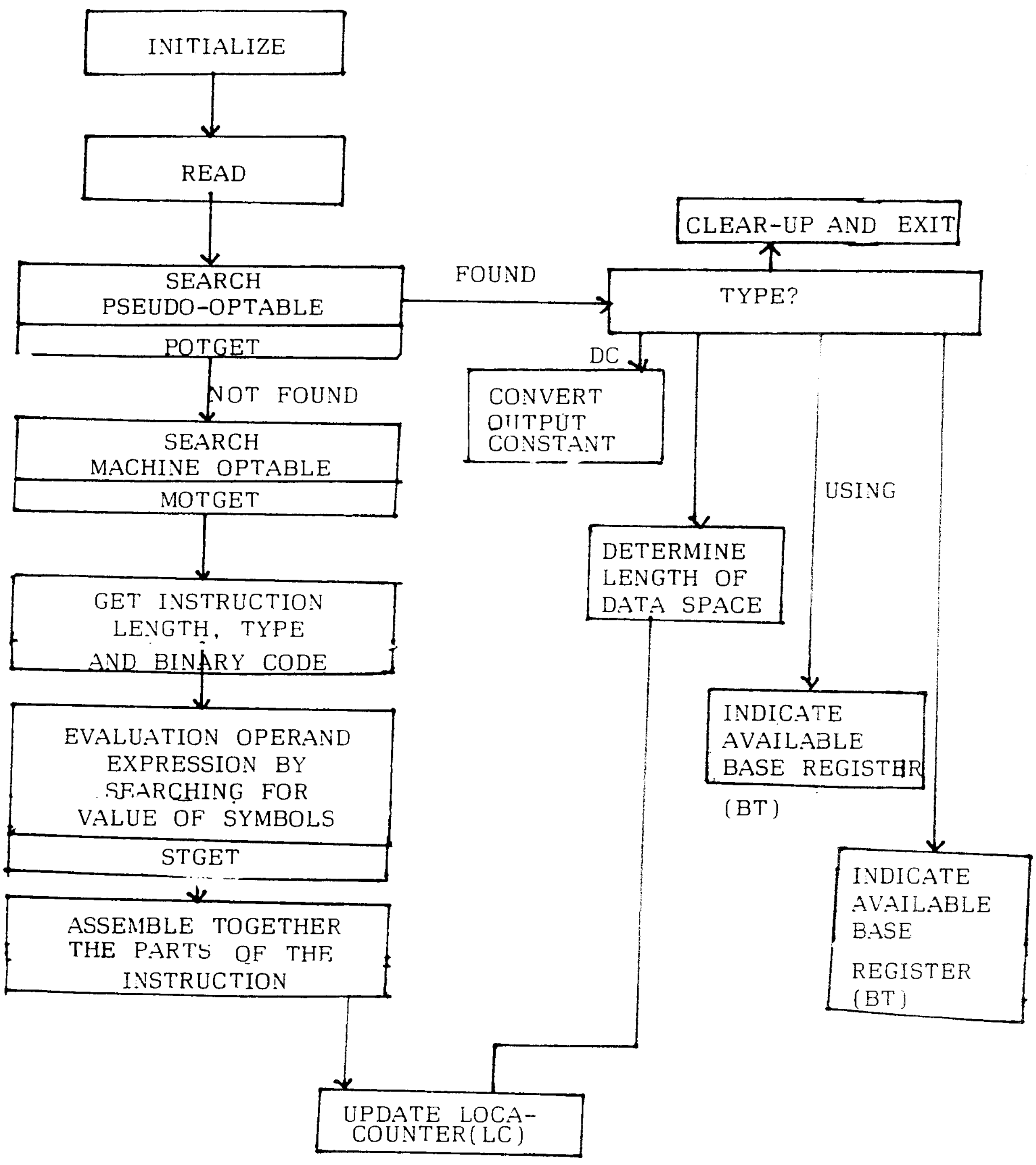

INITIALIZE
READ
LABEL ON CARD
YES
STORE LABEL IN ST WITH LC VALUE
STSTO
USING DROP
SEARCH PSEUDO-OPTABLE (POT)
POTGET
TYPE
END
GO TO PASS 2
DS, DC
DETERMINE LENGTH OF DATA SPACE
REQUIRED
SEARCH MACHINE-OPTABLE MOT
MOTGET
GET LENGTH OF INSTRUCTION
PROCESS LITERAL
LITSTO
UP LOCATION
COUNTER(LC)

FIG. 2.1.

Flow charts given outline the steps involved in pass 1 and pass 2, which specifies the algorithm and data structure involved.


INITIALIZE
READ
SEARCH PSEUDO-OPTABLE
POTGET
FOUND
TYPE?
CLEAR-UP AND EXIT
DC
CONVERT OUTPUT CONSTANT
NOT FOUND
SEARCH MACHINE OPTABLE
MOTGET
USING
DETERMINE LENGTH OF DATA SPACE
GET INSTRUCTION LENGTH, TYPE AND BINARY CODE
INDICATE AVAILABLE BASE REGISTER
(BT)
EVALUATION OPERAND EXPRESSION BY SEARCHING FOR VALUE OF SYMBOLS
STGET
INDICATE AVAILABLE BASE REGISTER (BT)
ASSEMBLE TOGETHER THE PARTS OF THE INSTRUCTION
UPDATE LOCA-COUNTER(LC)

PASS 2 OVERVIEW EVALUATE FIELDS AND GENERATE CODE

FIG. 2.2

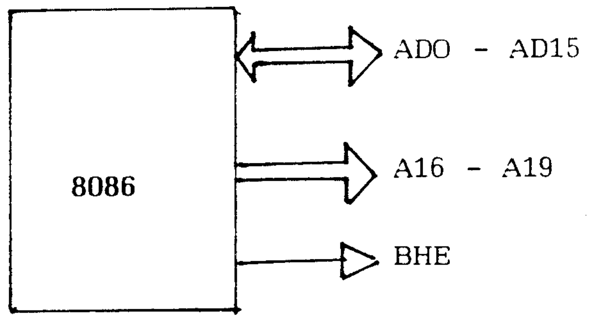
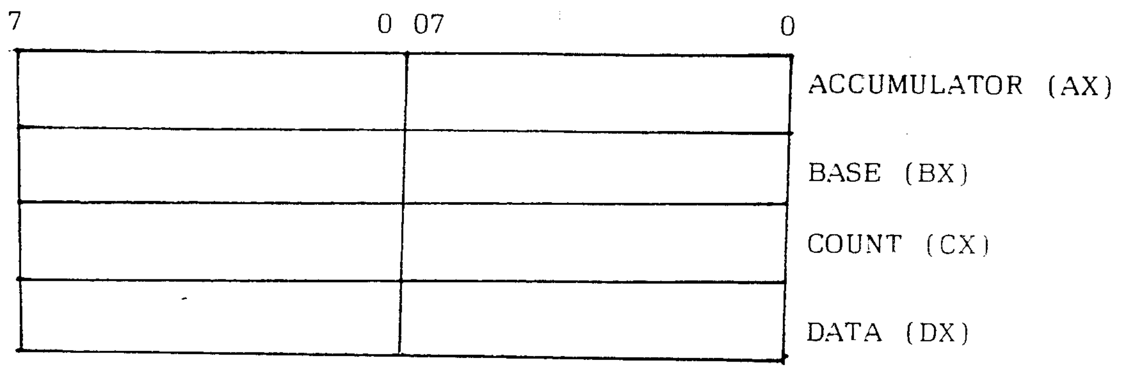


FIG. NO. 3.1

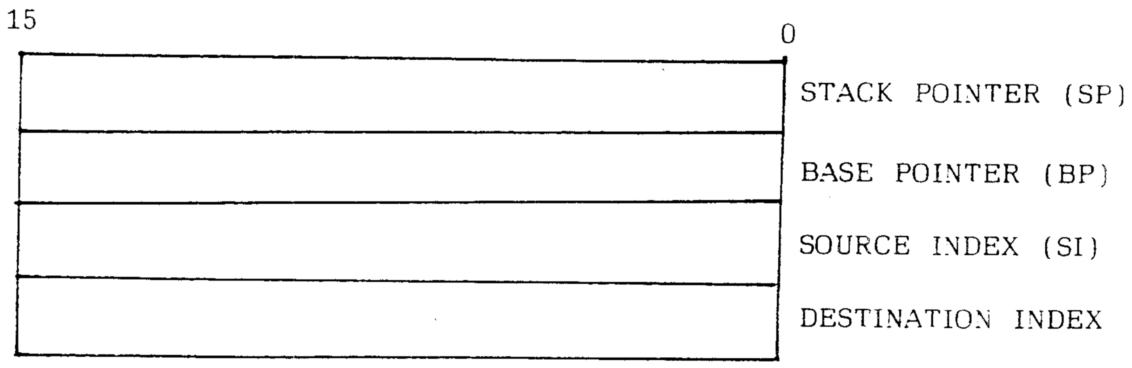
BHE	AO	What is read/written
0	0	116 bit word
0	1	1 byte from /to odd address
1	0	1 byte from/to even address
1	1	NONE

TABLE No. 3.1.



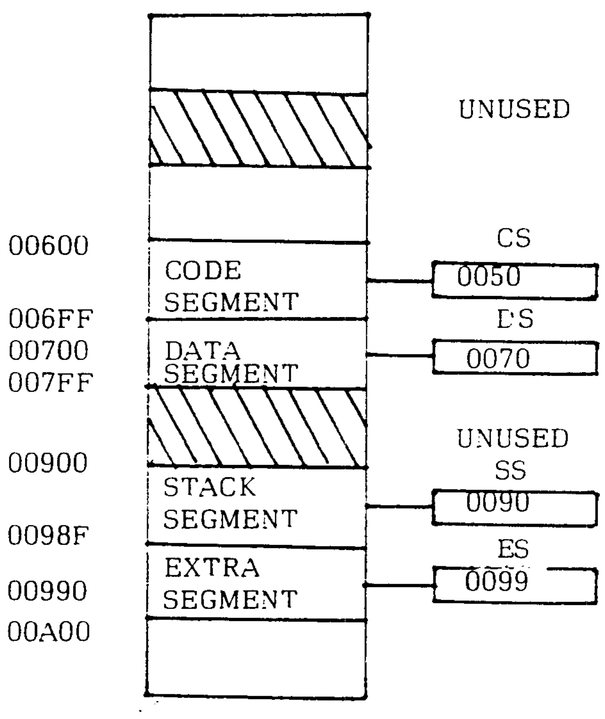
GENERAL PURPOSE REGISTERS

FIG. No. 3.2.



POINTERS AND INDEX REGISTERS

FIG. No. 3.3



MEMORY SEGMENTATION IN 8086

FIG. 3.4

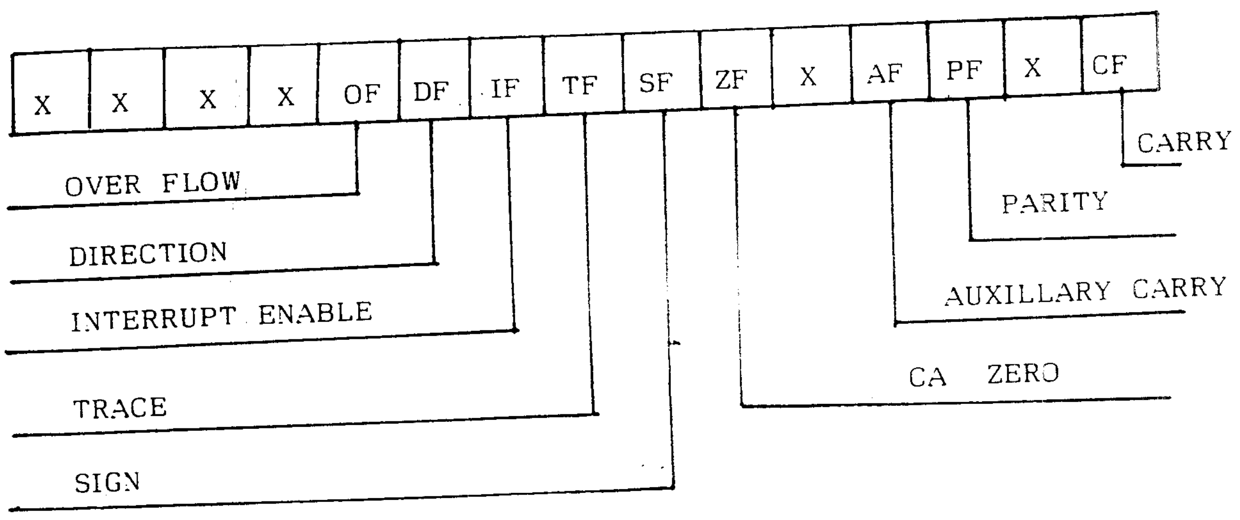
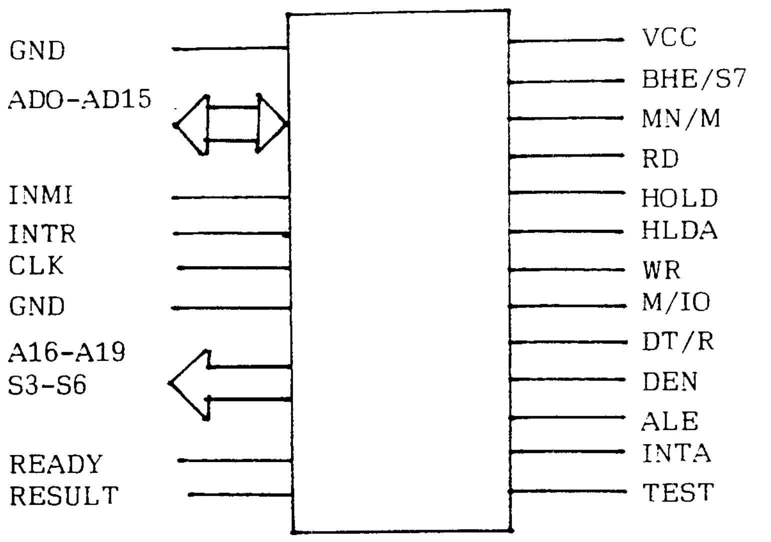
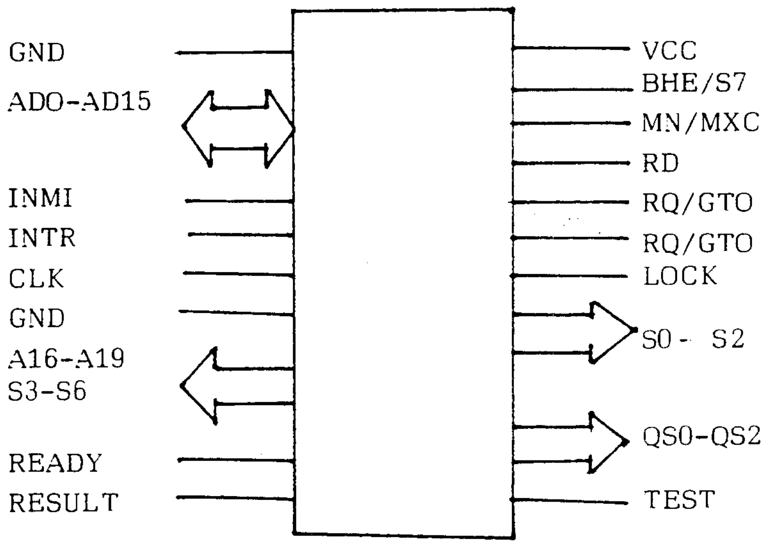

X
X
X
X
OF
DF
IF
TF
SF
ZF
X
AF
PF
X
CF
CARRY
OVER FLOW
PARITY
DIRECTION
AUXILLARY CARRY
INTERRUPT ENABLE
CA ZERO
TRACE
SIGN

FIG. No. 3.5



(a)

MINIMUM MODE



(b)

MAXIMUM MODE

FIG. No. 3.6

MOV SP,BX

FIG. No. 3.8

Bibliography

BIBLIOGRAPHY

1.	Systems Programming	:	John J. Donovan
2.	The 'C' Programming Language	:	Brain W. Kernighan, Dennis M.Ritchie
3.	Programming in 'C'	:	Stephen G. Kochan
4.	Micro Computer system The 8086/8088 family Architecture, Programming and design	:	Yu-Cheng Liu Glenn A,. Gibson
5.	Microprocessors and Interfacing Programming and Hardware	:	Douglas V. Hall
6.	Introduction to Microprocessors	:	Aditya P. Mathur