THE INSTRUCTION SET OF OUR PROCESSOR IS AS FOLLOWS (SEE FIG. 6.5 P.244)

	OP CODE	OP CODE MNEMONIC	COMMENT
0	0000	LOAD X	CON(X) → R
1	0001	STORE X	CON(R) → X
2	0010	CLEAR X	0 → X
3	0011	ADD X	CON(R)+CON(X) → R
4	0100	INCREMENT X	CON(X)+1 → X
5	0101	SUBTRACT X	CON(R)−CON(X) → R
6	0110	DECREMENT X	CON(X)−1 → X
7	0111	COMPARE X	SET CONDITION CODES

IF	GT	EQ	LT
CON(X)>CON(R)	1	0	0
CON(X)=CON(R)	0	1	0
CON(X)<CON(R)	0	0	1

8	1000	JUMP X	TRANSFER TO X
9	1001	JUMPGT X	IF GT=1 TRANSFER TO X
10	1010	JUMPEQ X	IF EQ=1 TRANSFER TO X
11	1011	JUMPLT X	IF LT=1 TRANSFER TO X
12	1100	JUMPNEQ X	IF EQ=0 TRANSFER TO X
13	1101	IN X	STANDARD INPUT → X
14	1110	OUT X	CON(X) → STANDARD OUTPUT
15	1111	HALT	STOP EXECUTION.

WE ALSO HAVE PSEUDO OPS .BEGIN, .END, AND .DATA.

NOTE THAT .DATA RESULTS IN AN INTEGER VALUE. IN PARTICULAR ALL OF OUR NUMERICAL VALUES ARE INTEGERS.

EX.1

WRITE AN ASSEMBLY LANGUAGE PROGRAM WHICH DOES THE FOLLOWING: GET THREE INPUT VALUES FROM THE USER a, b, c, AND PRINT THE VALUE a+b−c.

ALGORITHM

1.) GET VALUES FOR a, b, c FROM USER
2.) SET d TO a+b−c
3.) PRINT d

PROGRAM

```
        .BEGIN
        IN   A
        IN   B
        IN   C
        LOAD A
        ADD  B
        SUBTRACT C
        STORE  D
        OUT    D
        HALT
A:      .DATA  0
B:      .DATA  0
```

```
C:      .DATA   0
D:      .DATA   0
        .END
```

EX2 FIND THE ABSOLUTE VALUE OF A NUMBER.

ALGORITHM

1.) GET A VALUE FOR a FROM USER
2.) SET b TO |a|
3.) PRINT b

REFINEMENT OF STEP 2:

2.1) IF a < 0 SET b TO 0 - a
2.2) ELSE SET b TO a

PROGRAM

```
        .BEGIN
        IN      A
        LOAD    ZERO
        COMPARE A
        JUMPLT  NEG     -- A is NEGATIVE
        LOAD    A       -- A is NON-NEGATIVE
        STORE   B
        JUMP    PRINT
NEG:    SUBTRACT A
        STORE   B
PRINT:  OUT     B
        HALT
```

```
A:      .DATA    0
B:      .DATA    0
ZERO:   .DATA    0
        .END
```

EX3.

FIND THE SUM OF A LIST OF INTEGERS ENTERED BY THE USER. THE PROGRAM WILL TAKE ANY NUMBER OF INPUT VALUES AND WILL STOP WHEN THE SENTINAL VALUE 0 IS ENTERED. THE SUM IS THEN PRINTED.

ALGORITHM

1.) SET SUM TO 0
2.) GET item FROM USER
3.) while item ≠ 0
4.) SET SUM TO SUM + item
5.) GET item.
6.) END loop
7.) PRINT SUM
8.) STOP

PROGRAM

```
        .BEGIN
        CLEAR   SUM
LOOP:   IN      ITEM
        LOAD    ZERO
        COMARE  ITEM
```

```
        JUMPEQ  PRINT
        LOAD    SUM
        ADD     ITEM
        STORE   SUM
        JUMP    LOOP
PRINT:  OUT     SUM
        HALT
SUM:    .DATA   0
ITEM:   .DATA   0
ZERO:   .DATA   0
```

EX. 4

GET AN INPUT VALUE a FROM THE USER, ADD TWO TO IT REPEATEDLY UNTIL THE RESULT IS GREATER THAN FIFTY. PRINT OUT THE NUMBER OF TIMES TWO WAS ADDED.

ALGORITHM

1.) GET a
2.) SET COUNT TO 0
3.) SET r TO a
4.) WHILE r ≤ 50
5.) SET r TO r+2
6.) SET COUNT TO COUNT+1
7.) END LOOP
8.) PRINT COUNT
9.) STOP

USE OF THE VARIABLE ^ SUGGESTS THAT WE EMPLOY THE REGISTER R FOR THE ACCUMULATOR.

PROGRAM

```
          .BEGIN
          IN          A
          CLEAR       COUNT
          LOAD        A
LOOP:     COMPARE     FIFTY
          JUMPLT      PRINT
          ADD         TWO
          INCREMENT   COUNT
          JUMP        LOOP
PRINT:    OUT         COUNT
          HALT
A:        .DATA       0
COUNT:    .DATA       0
TWO:      .DATA       2
FIFTY:    .DATA       50
```

PROGRAMMING ASSIGNMENT 3

IN THIS ASSIGNMENT YOU WILL WRITE MACHINE LANGUAGE AND ASSEMBLY LANGUAGE PROGRAMS FOR A SIMPLE PROCESSOR WITH THE INSTRUCTION SET ^ GIVEN IN FIG. 6.5 P. 244

NOTE HOWEVER THAT THE MACHINE AND ASSEMBLY LANGUAGES USED IN PAT DIFFER IN SOME RESPECTS FROM THOSE DESCRIBED IN SECTION 6.3.

FIRST, THE MACHINE LANGUAGE INSTRUCTION FORMAT CONSISTS OF AN 8 BIT OP CODE FOLLOWED BY A SINGLE 8 BIT ADDRESS FIELD. THUS WE STILL HAVE ONE ADDRESS FIELD AND TWO BYTES PER INSTRUCTION, BUT THE BITS ARE DISTRIBUTED DIFFERENTLY.

WITH 8 BITS FOR THE OP CODE WE COULD HAVE AS MANY AS $2^8 = 256$ OPERATIONS IN OUR INSTRUCTION SET, BUT WE DO NOT. ONLY THE 16 OPERATIONS DESCRIBED ON P. 244 ARE AVAILABLE. MEMORY IS NOW LIMITED TO $2^8 = 256$ BYTES.

THE DIFFERENT DISTRIBUTION OF BITS IMPLIES THAT EACH BYTE OF MEMORY CONTAINS EITHER AN OP CODE, A WHOLE ADDRESS FIELD, OR A DATA VALUE. A MACHINE LANGUAGE PROGRAM WILL ALWAYS START AT ADDRESS 0, SO THAT OP CODES ARE STORED IN EVEN CELLS AND ADDRESS FIELDS ARE STORED IN ODD CELLS. DATA VALUES CAN BE STORED IN ANY CELL.

WE WILL WRITE OUR SIMULATED MACHINE CODE IN DECIMAL RATHER THAN BINARY.

NOW WE CAN TRANSLATE EXAMPLE 1 INTO MACHINE CODE BY HAND.

EX 1.

	ASSEMBLY CODE	
	.BEGIN	
	IN	A
	IN	B
	IN	C
	LOAD	A
	ADD	B
	SUBTRACT	C
	STORE	D
	OUT	D
	HALT	
A:	.DATA	0
B:	.DATA	0
C:	.DATA	0
D:	.DATA	0
	.END	

MACHINE CODE

ADDRESS	OPCODE	ADDRESS FIELD
0	13	17
2	13	18
4	13	19
6	0	17
8	3	18
10	5	19
12	1	20
14	14	20
16	15	
17	0	
18	0	
19	0	
20	0	

TO RUN THIS MACHINE LANGUAGE PROGRAM SAVE THE LIST:

13 17 13 18 13 19 0 17 3 18 5 19 1 20 14 20 15 0 0 0

AS A TEXT FILE AND START THE SIMULATOR (RUN).

TRANSLATING EXAMPLE 2 WE GET

EX2.

	ASSEMBLY CODE		ADDRESS	MACHINE CODE: OP CODE	MACHINE CODE: ADDR FIELD
	.BEGIN				
	IN	A	0	13	21
	LOAD	ZERO	2	0	23
	COMPARA	A	4	7	21
	JUMPLT	NEG	6	11	14
	LOAD	A	8	0	21
	STORE	B	10	1	22
	JUMP	PRINT	12	8	18
NEG:	SUBTRACT	A	14	5	21
	STORE	B	16	1	22
PRINT:	OUT	B	18	14	22
	HALT		20	15	
A:	.DATA	0	21	0	
B:	.DATA	0	22	0	
ZERO:	.DATA	0	23	0	
	.END				

EXCERCISE

TRANSLATE EX3 AND EX4 INTO MACHINE LANGUAGE BY HAND.

THE ASSEMBLY LANGUAGE USED IN PAS IS ALSO SLIGHTLY DIFFERENT FROM THAT DESCRIBED IN THE BOOK.

THE SYNTAX FOR COMMENTS IS THE SAME AS FOR THE C LANGUAGE:

/* comments */

NOT

-- COMMENTS

SEE THE WEB PAGE FOR FURTHER DETAILS.

- READ THE REST OF CHAPTER 6: 6.1, 6.2, 6.4 (OPERATING SYSTEMS, SYSTEM SOFTWARE.)

HW 7 CHAPTER 6 P. 285
4 a-f, 5 a-d, 7, 8, 9, 10, 11, 13, 15