Basic computer operation and organization

Use hex to represent memory locations as seen by the microcomputer. Memory can be organized as:

- bytes

address	memory
$0	byte 0
$1	byte 1
$2	byte 2
$3	byte 3
$4	byte 4
$5	byte 5

- words

address	memory		
$0	byte 0	byte 1	word 0
$2	byte 2	byte 3	word 1
$4	byte 4	byte 5	word 2
$6	byte 6	byte 7	word 3
$8			word 4
$A			word 5

- long words

address	memory			
$0	byte 0	byte 1	byte 2	byte 3
$4	byte 4	byte 5	byte 6	byte 7
$8	byte 8	byte 9	byte A	byte B
$C	byte C	byte D	byte E	byte F

Machine code (stored program execution)

z := x + y high level C or Pascal representation
where x,y,and z will represent words in memory

Data:

z is at memory address $1204
x is at memory address $1200
y is at memory address $1202

address	memory				contents
$1200	0001	0010	0011	0100	$1234
$1202	0100	0011	0010	0001	$4321
$1204	0000	0000	0000	0000	$0000

For some reason we decided to use words (16 bits) for all operations.

Instructions:

address	memory		meaning
$1000	3A	38	move a word from $1200
	12	00	to D5
$1004	DA	78	add the word at $1202

			to
	12	02	the contents of D5
$1008	31	C5	move the contents of D5
	12	04	to $1204
$100C	4E	40	stop

Coding of an instruction. This is an opcode word as defined by Motorola for the 68000. See The MC68000 Programmer's reference Manual.

15 14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
op code	destination		source	
	register	mode	mode	register
first four bits are the op code, indicates a move in this case	Dn = data register Abs.w = absolute word Abs.L=absolute long word	how to manipulate data:		Dn = data register Abs.w = absolute word Abs.L=absolute long word

In this case the $3A38 instruction at $1000 would be interpreted as a MOVE instruction. (see p.4-116 of the Programmer's Reference Manual, latest edition)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	1	1	0	1	0	0	0	1	1	1	0	0	0

first four bits are the op code for a move	word length	D5 data register	put data into register	get data from memory address (word length) which follows <u>EXTENSION WORD</u>	word length address

Disassembly is the interpretation of coded instructions
The instructions we just used are interpreted as:

<u>$3A38 1200</u>

0011 1010 0011 1000		rewrite as binary
0011 101 000 111 000		regroup into the appropriate fields: op code, destination, and source
op code	00XX	indicates a MOVE, move data from source to destination
size	11	indicates word length
destination	101 000	
mode	000	indicates to a data register
register	101	indicates to register D5
source	111 000	
mode	111	indicates one of several possible modes: absolute and PC relative
register	000	indicates that Abs.W is being used requiring a 16-bit extension word

→ instruction is a word length move of the contents of $1200 to D5

$DA78 1202

1101 1010 0111 1000 rewrite as binary

The form of this instruction is different from that of the MOVE.

The 1101 op code indicates that this is an ADD.

15	14	13	12	11 10 9	8 7 6	5 4 3	2 1 0
1	1	0	1	register	Op-mode	Effective Address mode	Effective Address register
op code, indicates an ADD in this case				specifies one of the eight data registers			

1101 101 001 111 000 regroup into the appropriate fields: op code, register, op-mode, and effective address

op code	1101	indicates an ADD, add binary
register	101	indicates register D5
op-mode	001	indicates word length add of the form (<Dn>) + (<ea>) → <Dn>

parentheses are used to indicate the contents of

effective address

mode	111	indicates absolute or PC relative addressing
register	000	indicates that Abs.W is being used

→ instruction is a word length add of the contents of $1202 to the contents of D5 with the result being put into D5

$31C5 1204

0011 0001 1100 0101		rewrite as binary
0011 000 111 000 101		regroup into the appropriate fields: op code, destination, and source
op code	00XX	indicates a MOVE, move data from source to destination
size	11	indicates word length
destination	000 111	
mode	111	indicates one of several possible modes: absolute and PC relative
register	000	indicates that Abs.W is being used
source	000 101	
mode	000	indicates from a data register
register	101	indicates from register D5

→ instruction is a word length move of the contents of D5 to $1204

Your textbook (p.54-55) lists several common instructions:

MOVE

copy 16-bit word specified by the source into the location specified by the destination operand

303C <number>	MOVE.W	#N,D0
33FC <number>,<address>	MOVE.W	#N,<address>
3039 <address>	MOVE.W	<address>,D0
33C0 <address>	MOVE.W	D0,<address>

ADD

adds the 16-bit word specified by the source and the 16-bit contents of the destination. The result is stored in the destination:

(<source>) + (<destination>) → <destination>

0640 <number>	ADDI.W	#N,D0
0679 <number>,<address>	ADDI.W	#N,<address>
D079 <address>	ADD.W	<address>,D0
D179 <address>	ADD.W	D0,<address>

SUB

subtracts the 16-bit word specified by the source from the 16-bit contents of the destination. The result is stored in the destination:

(<destination>) - (<source>) → <destination>

0440 <number>	SUBI.W	#N,D0
0479 <number>,<address>	SUBI.W	#N,<address>

9079 <address>	SUB.W	<address>,D0
9179 <address>	SUB.W	D0,<address>

Example: Chapter 3, problem 25

```
00010A      303C     000A
00010E      33C0     0000 020A
000114      0679     00C3 0000 020C
00011C      9079     0000 020C
000122      0679     0AF3 0000 020A
00012A      0640     00F8
...

00020A      0036
00020C      03FA
```

All numbers are in hex. Each line indicates an individual instruction.

Initially, (D0) = 0000 003B, ($020A)=$0036, ($020C)=$03FA

The disassembled program:

```
MOVE.W    #10,D0          put $A into D0
MOVE.W    D0,$020A        put the contents of D0
                          ($A) into address $20A
ADDI.W    #$C3,$020C      add $C3 to the contents
                          of $020C ($3FA) and put
                          the result into $20C
SUB.W     $20C,D0         subtract what's in $20C
                          from the contents of D0
```

		($A) and put the result in D0
ADDI.W	#$0AF3,$020A	add $0AF3 to the contents of $20A
ADDI.W	#$F8,D0	add $F8 to the contents of D0

The detailed disassembly:

<u>303C 000A</u>

0011 0000 0011 1100	rewrite as binary
0011 000 000 111 100	regroup

op code	00XX	indicates a MOVE
size	11	indicates word length MOVE
destination	000 000	
mode	000	indicates data register
register	000	register D0
source	111 100	
mode	111	any of several modes
register	100	indicates immediate mode, designated as Imm, i.e. a constant contained in an extension word

→ MOVE.W #10,D0

<u>33C0 0000 020A</u>

0011 0011 1100 0000	rewrite as binary

0011 001 111 000 000 regroup

op code	00XX	indicates a MOVE
size	11	indicates word length MOVE
destination	001 111	
mode	111	indicates any of several modes
register	001	indicates Abs.L, a long word address requiring two extension words
source	000 000	
mode	000	indicates a data register
register	000	register D0

→ MOVE.W D0, $0000 020A

0679 00C3 0000 020C

0000 0110 0111 1001		rewrite as binary
0000 0110 01 111 001		regroup
op code	0000 0110	indicates an ADDI
size	01	word operation, i.e. one 16-bit extension word
effective address		
mode	111	
register	001	indicates Abs.L, requires a 32-bit longword address, i.e. two 16-bit extension words

→ ADDI.W #$C3, $0000 020C

9079 0000 020C

1001 0000 0111 1001		rewrite as binary
1001 000 001 111 001		regroup
op code	1001	indicates a SUB
register	000	indicates D0
op-mode	001	word operation, i.e. (<Dn>)-(<ea>)→<Dn>
effective address		
mode	111	

	register	001	indicates Abs.L, requires a 32-bit longword address, i.e. two 16-bit extension words

→ SUB.W $0000 020C, D0

0679 0AF3 0000 020A

0000 0110 0111 1001 rewrite as binary
0000 0110 01 111 001 regroup

op code 0000 0110 indicates an ADDI
size 01 word operation, i.e. one 16-bit extension word
effective address
mode 111
register 001 indicates Abs.L, requires a 32-bit longword address, i.e. two 16-bit extension words

→ ADDI.W #$0AF3, $0000 020A

0640 00F8

0000 0110 0100 0000 rewrite as binary
0000 0110 01 000 000 regroup

op code 0000 0110 indicates an ADDI
size 01 word operation, i.e. one 16-bit extension word
effective address
mode 000 indicates data register
register 000 indicates D0

→ ADDI.W #$F8, D0

The final program is then
If (D0)=$3B, ($20A) = $36, ($20C) = $3FA

MOVE.W	#10,D0	(D0)=$A
MOVE.W	D0,$020A	($20A)=$000A
ADDI.W	#$C3,$020C	($020C) = ($020C)+$C3 = $3FA + $C3 = $4BD
SUB.W	$20C,D0	(D0) = (D0)-($20C) = $A-$4BD = $ FB4D
ADDI.W	#$0AF3,$020A	($20A) = ($20A)+$0AF3 = $A + $0AF3 = $0AFD
ADDI.W	#$F8,D0	(D0) = (D0)+$F8= $FB4D + $F8 = $FC45

Math:

000A	0000 0000 0000 1010
04BD	0000 0100 1011 1101
complement	1111 1011 0100 0010
add 1	1111 1011 0100 0011
000A	0000 0000 0000 1010
-04BD	1111 1011 0100 0011
FB4D	1111 1011 0100 1101
+00F8	0000 0000 1111 1000
FC45	1111 1100 0100 0101