Chapter 6 **Introduction to Assembly language programming and the 68K family**

We will look at a family of chips produced by Motorola. The family consists of the 68000, 68020, 68030, 68040 and the 68060 chips. Code written for the 68000 (for the most part) will run on later chips but code written in assembly for later chips may not run on older chips because the instruction sets have been enhanced.

Why do we look at the 68k family architecture?

- Found in a wide variety of microcomputers
- Has a relatively sophisticated architecture
- As far as assembly languages in general, this one is relatively easy to learn.

Recall, when we speak of architecture we are talking about an abstract view of the computer and a description of what it can do. This is the assembly language programmer's view of machine.

Structure of an assembly language program

We will start by looking at an assembly language program to implement the operations :

P=2, Q=4
R=P+Q

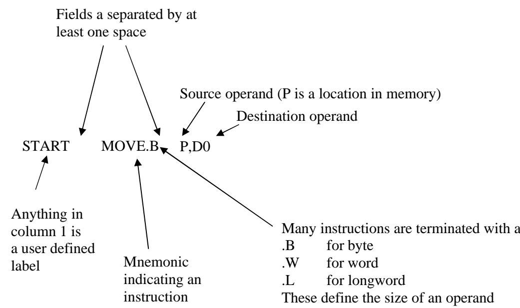
Some rules before we write our program (creating the source file):

- Leftmost column is for user defined labels
- A line beginning with an asterisk represents a comment
- A mnemonic and its operand(s) must be separated by at least one space
- No embedded spaces may be located within the mnemonic or operand fields
- Numbers may be represented in
  - Hexadecimal – start with a $
  - Binary – start with a %
  - In decimal –just write normal format
- Prefixing an operand with # indicates immediate addressing mode. Just think of this as the operand is an actual value (or literal)

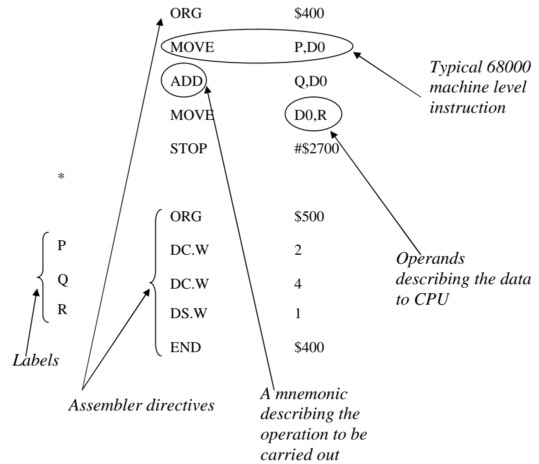
The following are equivalent operations:

```
ADD   #24,D0
ADD   #$18,D0
ADD  #%11000,D0
```

Typical assembly language instruction:



An assembly language program:



We divide the assembly language statements into two types:

assembler directives and **executable statements**

The executable statements of an assembly language program get translated into machine code. In our sample program the executable statements are:

```
MOVE  P,D0
ADD  Q,D0
MOVE  D0,R
STOP #$2700
```

The STOP # $2700 is a special code to terminate the program running on the simulator and is used to load the 68K's status register (SR) with $2700 – a special code used to initialize the 68K.

The assembler directives tell the assembler things it needs to know about the program it is assembling. ORG stands for origin and tells the assembler where instructions or data are to be loaded into memory.

```
ORG  $400
```

tells the assembler to start loading instructions at address 400_{16} which is 1024_{10}. The reason for this is the 68k uses the first 1024 bytes of memory, 0 – $3FF_{16}$ for a special purpose.

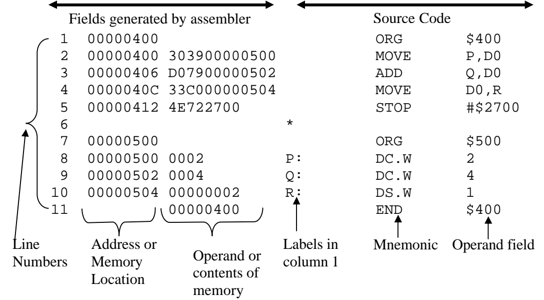
Next we run a source file through the assembler:

Source:

```
        ORG  $400
        MOVE P,D0
        ADD  Q,D0
        MOVE D0,R
        STOP #$2700
*
        ORG  $500
P       DC.W 2
Q       DC.W 4
R       DS.W 1
        END  $400
```

What the Assembler does: (A listing of what went on – a bin file is actually the product we run)

```
Source file: EXAMP.X68
Assembled on: 01-10-23 at: 14:21:47
          by: X68K PC-2.2 Copyright (c) University of Teesside
1989,99
Defaults: ORG $0/FORMAT/OPT A,BRL,CEX,CL,FRL,MC,MD,NOMEX,NOPCO
```



Notice all instructions, addresses and data are represented in hexadecimal form.

Looking at one of the lines from our assembled program:

```
2  00000400 303900000500          MOVE   P,D0
```

On line 2 we have the instruction `MOVE P,D0`

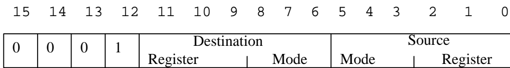
It is stored in memory location 400 and the instruction translated into machine code is `303900000500`

How did `MOVE P,D0` translate to that particular machine code?

The move instruction for the 68000 takes one of three forms:

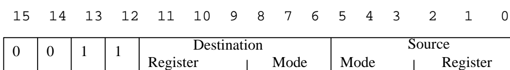
MOVE.B

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



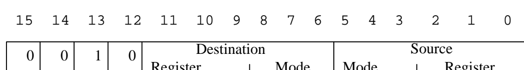
MOVE.W

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



MOVE.L

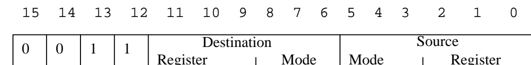
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0



MOVE P,D0

Since we are dealing with a WORD operation we use:

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



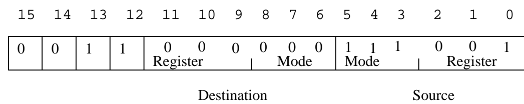
Mode is determined from the following Table:

Addressing Mode	Mode	Register
Data Register Direct	000	Register number
Address Register Direct	001	Register number
Address Register Indirect	010	Register number
Address Register Indirect with Postincrement	011	Register number
Address Register Indirect with Predecrement	100	Register number
Address Register Indirect with Displacement	101	Register number
Address Register Indirect with Index	110	Register number
Absolute Short	111	000
Absolute Long	111	001
Program Counter with Displacement	111	010
Program Counter with Index	111	011
Immediate or Status Register	111	100

The destination is a data register thus MODE: 000 , Register 000

MOVE P,D0

Since we are dealing with a WORD operation we use:



To convert 0011000000111001 to hexadecimal we group by 4 digits:

(0011)(0000)(0011)(1001) but this must be followed by the address of source, that is P. Looking back at the assembled code we see P is stored strting at: 00000500 (this is already HEX)

Thus the command becomes

303900000500 (HEX) in 68K machine language.

Try converting

```
MOVE     D0,R
```

Where R is reserved memory starting at : 00000504

The assembler itself has a variable called the ***location counter***, which keeps track of where the next instruction or data element is to be located in memory. The

ORG $400

set its value initially.

Another of the compiler directives is DC, which stands for define constant. This directive allows us to load a constant into memory before execution. The DC command can be followed by one of the following qualifiers :

.L, .W , or .B

In the line 8 of our program we have the label P

```
8  00000500 0002       P:    DC.W      2
```

This allows us to refer to memory location 500_{16} as P. Since we indicated we were storing a word (2 bytes or 16 bits) what is stored is 0000000000000010_2 or regrouping in sets of 4 , 0002_{16}

DS , for define storage, is another compiler directive. It too takes the qualifiers .L, .W , or .B .

```
10  00000504 00000002    R:      DS.W      1
```

DS.W N instructs the assembler to reserve N words of memory for use. The address of the first word is associated with the label R. The location counter is moved forward the required amount to accommodate the requested memory block.

The last assembler directive is the END directive. It tells the assembler the end of program has been reached and there is nothing left to assemble.

```
11          00000400           END     $400
```

Our assembler (Teesside cross assembler) requires END to take one parameter indicating the address of the first instruction of the program.

A compiler directive we did not use is the EQ directive. It equates a symbolic name with a numeric value

```
Inum  EQU 3
```

Would allow us to use Inum instead of 3.

`ADD #Inum,D0` is equivalent to `ADD #3,D0`

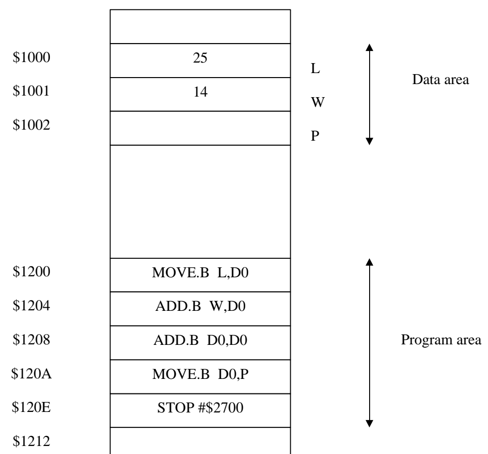
An Example of assembly language code, memory map and assembled program:

```
         ORG      $1000
L        DC.B         25
W        DC.B     14
P        DS.B     1
*
         ORG      $1200
         MOVE.B   L,D0
         ADD.B        W,D0
         ADD.B        D0,D0
         MOVE.B   D0,P
         STOP     #$2700
         END      $1200
```

Assembled Program:

```
 1  00001000                           ORG       $1000
 2  00001000 19           L:           DC.B      25
 3  00001001 0E           W:           DC.B      14
 4  00001002 00000001     P:           DS.B      1
 5                        *
 6  00001200                           ORG       $1200
 7  00001200 10381000                  MOVE.B    L,D0
 8  00001204 D0381001                  ADD.B     W,D0
 9  00001208 D000                      ADD.B     D0,D0
10  0000120A 11C01002                  MOVE.B    D0,P
11  0000120E 4E722700                  STOP      #$2700
12           00001200                  END       $1200
```

Memory Map:



A note of caution:
68K's instructions vary from 2 bytes in size to 10 bytes in size so it is difficult to estimate the size of a program by knowing the number of instructions it has. In practice you could consult Motorola’s manuals to determine the number of bytes for a particular instruction.