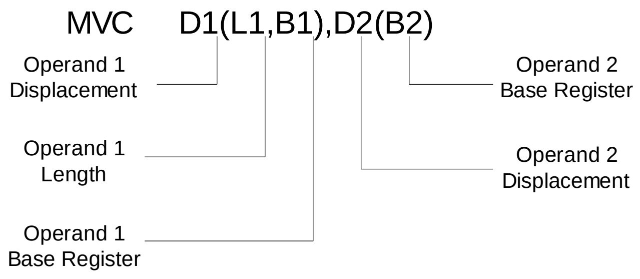
EXPLICIT ADDRESSING

The term **explicit addressing** refers to the practice of representing the address of a byte in memory by specifying a base register and a displacement, or a base and index register and a displacement. If at all possible, we would prefer to represent a byte in memory by a symbolic name. Usually this is possible. For instance, consider the following declaration,

```
XFIELD   DS    CL5
```

While it does have a length attribute of five, the symbol "XFIELD" represents the address of the **first** byte of that field. This is important to understand because the assembler will convert this symbolic address to a base/displacement format representing the address of the **first** byte of the field. This base/displacement address (BDDD in hexadecimal) occupies two bytes in the assembled code as evidenced in the instruction formats. (See the instruction format for an SS instruction.)

Unfortunately, sometimes the use of a symbolic name like XFIELD is inconvenient or impossible to create. At these times we must resort to an explicit address. The explicit format for each instruction in this text is provided at the top left of the title page for each instruction. For example, the MVC instruction lists the following explicit format.



Consider the following example instruction.

```
MVC   4(8,12),32(8)
```

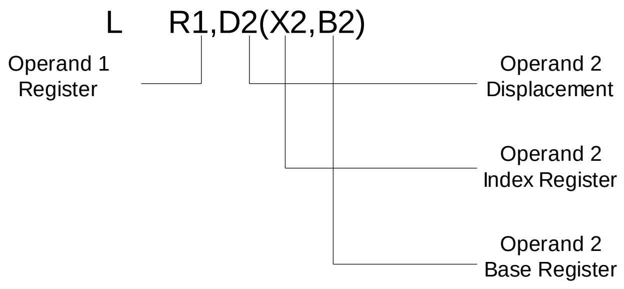
Using the explicit format above we can decipher what the instruction will do. First, we can see that it is a move instruction that will transfer 8 bytes. Where will the data that is to be moved come from? The answer is given explicitly as 32(8). This is a 32 byte displacement off register 8. At execution, the effective address is computed by adding the contents of base register 8 plus the 32 byte displacement:

Effective address = Contents(Base register 8) + 32

At "execution time" it is imperative that register 8 contains a "known" address and that the second operand field that we are currently addressing be 32 bytes away from this known address.

Where will the data be moved? Again, the answer is given explicitly as 4(12). This represents a 4 byte displacement off register 12. The effective address is computed to be the contents of base register 12 plus 4. The field which is to receive the data must be at an address which is 4 bytes larger than the “execution time” address in register 12.

Each instruction format has its own explicit format. For instance, the load instruction appears below.



Consider the following explicitly coded instruction,

```
L   10,4(3,5)
```

Obviously, register 10 is being loaded with a fullword in memory. Where is the fullword stored? The explicit format gives the answer. The base register is 5, register 3 is an index register, and 4 is a displacement. The effective address can be computed by adding the contents of the base register, plus the contents of the index register, plus the displacement:

Effective Address = Contents(register 5) + Contents(register 3) + 4

A knowledge of the explicit notation is helpful in understanding many instructions. For instance, it is common practice for an assembler programmer to code an instruction similar to the one below.

```
LA   R5,10
```

Without prior knowledge, what are we to make of this code? First we could look up a load address instruction and discover that it is an RX instruction and has an explicit format similar to the load instruction described above. The second operand for an RX instruction appears as D2(X2,B2). The instruction above has no parentheses - they were omitted. This means that 10 is a displacement. What is the effective address? Since the base and index registers were omitted, zero was chosen for the base and index registers. But the machine never uses register zero as a base or index register. This means that the effective address is simply 10. The effect of the instruction above is to load a “small” number into register 5. Experienced programmers understand this and choose the method as an effective way to load “small” numbers. What is meant by “small”? Since the number we are loading must appear as a displacement, the largest number we could code in this way is 4095 = X’FFF’ which is the maximum displacement allowed.

Mixing Symbolic and Explicit Notation

It is also possible to mix symbols and explicit notation within the same instruction. For example,

```
L    8,XWORD(5)
```

In the load instruction above, 8 is the Operand 1 register where a fullword will be loaded. What does the notation XWORD(5) mean? In all cases, the assembler can process a symbol like

XWORD and produce a base register and a displacement. This means that the 5 is an index register. A related example appears below.

```
                    MVC    AFIELD(8),BFIELD
```

Operand 1 appears as AFIELD(8). Again, the assembler can process the symbol AFIELD and produce a base register and a displacement. Checking the explicit notation for a move character instruction we see that the 8 must be the length of Operand 1. Normally, the length would be taken from the length attribute of AFIELD. Mixing notations allows us to overide the “implicit” length with an “explicit” length.

Examples

```
          LM     14,12,12(R13)  14 AND 12 REPRESENT A RANGE OF
REGISTERS.
                                WORDS ARE LOADED BEGINNING WITH THE WORD
                                12 BYTES OFF REGISTER 13.

          MVC    X(9),Y         A 9 BYTE EXPLICIT LENGTH
          L      R8,9           AN ERROR - TRIES TO LOAD THE WORD AT
                                ADDRESS 9.
          LA     R8,9           BETTER!  LOADS 9 INTO REGISTER 8
          AP     X(4),Y(3)      SS2 INSTRUCTIONS HAVE TWO LENGHTS
          MVI    0(R8),C’ ’     MOVES A BLANK TO THE ADDRESS IN R8
          BASR   R12,R0         RR INSTRUCTIONS ARE ALWAYS EXPLICIT
```

Tips

1) Avoid the use of explicit notation whenever possible to simplify your code.

2) If you must use explicit notation, always code base and index registers using equate names (R0, R1, ...). This practice was not followed in the notes above (examples excluded) in order to emphasize the explicit nature of the code. Using equate names forces the assembler to record the use of each register in the cross reference listing at the end of your program. For large programs this is a necessity.

3) One of the primary uses for explicit addresses involves parameter passing. In this area, the use of explicit addresses is a commonly accepted practice. Read **Program Linkage** and become familiar with this important use of explicit addresses.

4) Consider the use of a **DSECT** as a means of using symbolic names instead of coding explicit addresses.

5) Remember that explicit addresses must be used if you have not issued a **USING** directive or if no base register is available. (See **Base Displacement Addressing**.)