The assembler maintains a location counter to assign storage addresses to our program statements. You can refer to the current value of the location counter by using an asterisk (*) as a term in an operand. As the artifacts (fields, constants, DCB's, …) in our programs are assigned space and addresses (denoted by the address of the first byte of each artifact), the location counter is a usually updated by adding the length of the artifact. Slack bytes can also be inserted to bring the location counter to the required boundary. Slack bytes for a DS instruction are uninitialized. Slack bytes for a DC instruction are set to zero. CNOP, another instruction often coded inside macros, will insert a single byte of zeroes when starting on an odd boundary. Otherwise it will insert a series of one or more NOPs to bring alignment to a chosen halfword.

Boundaries

Even-numbered addresses are associated with boundaries (addresses with certain numeric qualities). For example,

Halfword boundaries are addresses that are evenly divisible by two.
Fullword boundaries are addresses that are evenly divisible by four.
Doubleword boundaries are addresses that are evenly divisible by eight.
Quadword boundaries are addresses that are evenly divisible by sixteen.

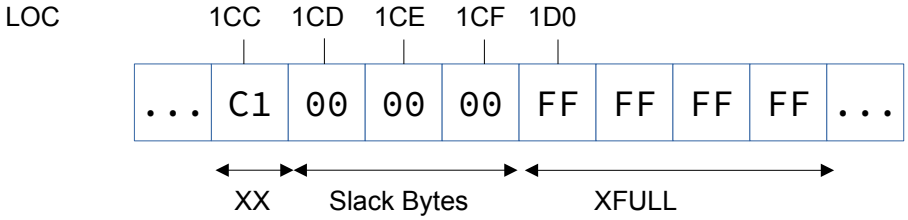
Slack Bytes

In order to provide proper boundaries (or alignment) for an artifact we define, the assembler will generate slack bytes when needed to bring the location counter to the next proper boundary. Consider the example below,

```
   Loc Object Code Addr1 Addr2 Stmt Source Statement

0001CC C1                       208 XXX     DC    C'A'
0001CD 000000
0001D0                          209 XFULL   DC    F'-1'
```

We defined a one-byte character constant "A" that is located at hex address 0001CC. This address is on a fullword boundary since X'1CC' = 460 base 10, and 460 is evenly divisible by 4. It is not a doubleword boundary since 460 is not evenly divisible by 8. Next we define XFULL, a fullword, which requires a fullword boundary. Since XXX occupies one byte, the next fullword is at 464 = X'1D0'. The assembler inserted three slack bytes of zeroes after XXX so that XFULL would begin on a fullword boundary. The slack bytes were initialized with zeroes because the subsequent instruction was a DC. This is pictured below. Had XFULL been defined with DS, the slack bytes would have been uninitialized.

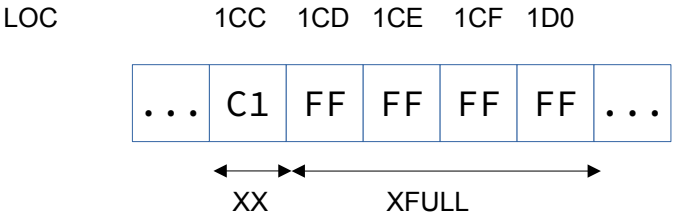


Overriding Alignment

In some cases, we may **not** want the assembler to generate slack bytes. We can override this feature by coding a length attribute for the artifact which requires alignment. Consider the example below.

```
    Loc Object Code Addr1 Addr2 Stmt Source Statement

0001CC C1                       208 XXX     DC    C'A'
0001D0                          209 XFULL   DC    FL4'-1'
```



Coding XFULL as FL4 signals the assembler to suppress generation of slack bytes for fullword alignment. As a result, XFULL is not properly aligned. Nevertheless, this may be what is required in certain situations.

Slack bytes are uninitialized if the field is being defined with DS. Slack bytes are set to zero if the field is being defined with DC.

Artifact Boundaries

The list below defines the boundaries for various artifact types coded without length attributes.

Field or Constant	**Type**	**Alignment**
Address	Y	Halfword
Address	A	Fullword
Address	S	Halfword
Address	V	Fullword
Address	J	Fullword
Address	Q	Fullword
Address	R	Fullword
Binary	B	Byte
Character	C	Byte
Graphic	G	Byte
Hexadecimal	X	Byte
Binary	H	Halfword
Binary	F	Fullword
Packed Decimal	P	Byte
Zoned Decimal	Z	Byte
Floating Point	E	Fullword
Floating Point	D	Doubleword
Floating Point	L	Doubleword

In addition to the type subfield, when coding DS or DC, you can also code a subtype for some data types. The subtype adds a one character suffix to the normal type.

The list below defines the boundaries for various artifact types **and** extensions (coded without length attributes).

Field or Constant	Type/Extension	Data	Alignment
Character	CA	ASCII Character	Byte
Character	CE	EBCDIC Character	Byte
Character	CU	Unicode UTF-16	Byte
Floating Point	EH	Hexadecimal	Fullword
Floating Point	EB	Binary	Fullword
Floating Point	ED	Binary	Fullword
Floating Point	DH	Hexadecimal	Doubleword
Floating Point	DB	Binary	Doubleword
Floating Point	DD	Decimal	Doubleword
Floating Point	LH	Hexadecimal	Doubleword
Floating Point	LB	Binary	Doubleword
Floating Point	LD	Decimal	Doubleword
Floating Point	LQ	Hexadecimal	Quadword
Fixed Point	FD	Binary	Doubleword
Address	AD	Binary	Doubleword
Address	VD	Binary	Doubleword
Address	JD	Address	Doubleword
Address	QD	Offset	Doubleword
Address	QY	Offset	Halfword
Address	RD	Address	Doubleword
Address	SY	Address	Halfword

Some Unrelated DCs

Consider the following definitions and their alignment.

```
                                                                    ALIGNMENT
0001CC 0A                          212 AAA     DC    B'1010'              BYTE
0001CD C3C8C1D9C1C3E3C5            213 BBB     DC    C'CHARACTER STRING'  BYTE
0001DD C1C2C3                      214 EEE     DC    X'C1C2C3'            BYTE
0001E0 000A                        215 FFF     DC    H'10'                HALFWORD
0001E2 0000          Slack Bytes
0001E4 FFFFFFFF                    216 GGG     DC    F'-1'                FULLWORD
0001E8 0000000000000000            217 HHH     DC    FD'0'                DBLEWORD
0001F0 123C                        218 III     DC    P'123'               BYTE
0001F4          <-- Unitialized Slack Bytes
0001F8 425F000000000000            220 KKK     DC    D'95'                DBLWORD
000200 77270BB7E1DB8FE4            221 LLL     DC    L'2.57E65'           DBLWORD
000210 0001                        222 MMM     DC    Y(BBB-AAA)           HALFWORD
000212 0000          Slack Bytes
000214 000001CC                    223 NNN     DC    A(AAA)               FULLWORD
000218 00000000000001CC            224 OOO     DC    AD(AAA)              DBLWORD
000220 C1C6                        225 PPP     DC    S(AAA)               HALFWORD
000222 0000          Slack Bytes
000224 00000000                    226 QQQ     DC    V(DOG)               FULLWORD
000228 0000000000000000            227 RRR     DC    VD(DOG)              DBLWD
```

Look through the DCs listed above. Do they begin on a proper boundary? Does the assembler generate slack bytes when needed? Are the slack bytes all zero?

Assembler Options That Affect Alignment

There are a number of assembler options that can affect alignment. In most cases you will want to take the default assembler alignment options that were chosen when the assembler was installed. Examine an assembler listing to see how your options are set.

ALIGN / NOALIGN – The assembler to checks (ALIGN) or not (NOALIGN) the alignment of addresses in machine instructions for consistency with the requirements of the operation code type. DC, DS, DXD, and CXD are to be aligned on the correct boundaries.

FLAG(ALIGN)- Instructs the assembler to issue diagnostic message ASMA033I when an inconsistency is detected between the operation code type and the alignment of addresses in machine instructions. Assembler option ALIGN describes when the assembler detects an inconsistency.

FLAG(NOALIGN) - Instructs the assembler not to issue diagnostic message ASMA033I when an inconsistency is detected between the operation code type and the alignment of addresses in machine instructions. Assembler option ALIGN describes when the assembler detects an inconsistency.

SECTALGN – Specifies the alignment for all sections. Alignment must be a power or 2 between 8 (the default) and 4096 (page alignment).

Requesting Alignment within Storage Using DS

When defining storage areas, you can use the DS instruction to force alignment for the next field. Each DS instruction begins with a duplication factor. When this factor is zero, the assembler forces alignment for the data type specified in the Type part of the operand. Here are four unrelated examples. In each case, X starts on the boundary defined in the previous DS. To provide the correct boundary, the assembler may generate slack bytes.

```
         DS    0F  THE NEXT FIELD, X, STARTS ON A FULLWORD BOUNDARY
X        DC    CL3

         DS    0H  THE NEXT FIELD, X, STARTS ON A HALFWORD BOUNDARY
X        DC    CL3

         DS    0D  THE NEXT FIELD, X, STARTS ON A DOUBLEWORD BOUNDARY
X        DC     CL3

         DS    0LQ THE NEXT FIELD, X, STARTS ON A QUADWORD BOUNDARY
X        DC     CL3
```

Requesting Alignment within Executable Code Using CNOP (Conditional NOP)

To get an instruction (or a storage area within a sequence of instructions) to align on a given halfword boundary, use the CNOP instruction. CNOP is often used inside macros to define local fields that are needed. If the location counter contains an odd address, CNOP will generate a single byte of zeroes to provide halfword alignment. Then, CNOP generates one or more NOP instructions until the proper alignment is reached. This ensures a continuous sequence of instructions. The generated NOP instructions may be a combination of four-byte BCs or two-byte BCRs. NOP branches are coded so that execution falls through to the next instruction.

CNOP is coded with two integer operands.

```
CNOP  byte,boundary
```

byte - An absolute expression that specifies which even-numbered byte in a fullword, doubleword, or quardword the location counter is set. The byte value must be an non-negative even integer less than the boundary value.

boundary - An absolute expression that specifies the type of boundary: fullword, doubleword or quadword. The possible values for boundary are 4,8, and 16. Assembler option SECTALGN may affect the values you can use for the boundary value.

Here are some examples:

```
CNOP  6,8   Align on the third halfword in a doubleword
CNOP  2,4   Align on the middle of a word
CNOP  4,8   Align on the middle of a doubleword
CNOP  0,16  Align on the beginning of a quadword
```

Tips

1) When determining alignment,

   Halfword addresses written in hexadecimal end in 0, 2, 4, 6, 8, A, C, or E.

   Fullword addresses written in hexadecimal end in 0, 4, 8, or C.

   Doubleword addresses written in hexadecimal end in 0 or 8.

   Quadword addresses written in hexadecimal end in 0.