DEFINING AND MANIPULATING SYMBOLIC BIT FLAGS IN ASSEMBLY LANGUAGE

Thomas R. Wilcox

TR 75 - 235

April 1975

Department of Computer Science
Cornell University
Ithaca, New York 14853

Defining and Manipulating Symbolic Bit Flags in Assembly Language

by

Thomas R. Wilcox[+]

Despite significant advances in designing and implementing high level languages for systems programming, assembly language is still the language of choice when uniformly efficient coding is required in a programming project such as a production compiler or an operating system. Unfortunately such projects happen also to proceed on a grand scale - i.e. being produced and maintained by more than one programmer over a period of several years. In such an environment, a macro processing facility is an absolute necessity if the project is to be the least bit manageable. If nothing else, macros provide minimal project-wide control over access to data structures and program modules by providing flexible abbreviations for calling sequences and repetitive address computations. On the other hand, if the efficiency of the final product is not to be jeopardized, macros are practically the only high level language feature permissible.

Since macros are the _de facto_ workhorse of structured programming and software reliability, attempts to improve systems programming standards will be most fruitful if implemented using existing macro processing facilities. A non-trivial example of this is the use of macros to facilitate the manipulation of bit flags in an OS/360 assembly language program. A _flag_ is a bit or collection of bits in the same byte. When a large number of flags are used -- for example, in a system I/O control block or a compiler's symbol table -- the flags must be named to reduce the possibility of errors in their use and hence to improve the reliability of the software.

Standard assembler facilities -- DC's and EQU's -- are helpful in improving reliability in that they permit names to be assigned to flags. This is not quite satisfactory since two names usually must be defined for each flag -- one name for the byte containing the flag and another name for the bits of the byte assigned to the flag. Using two names does not eliminate one of the more frequent errors encountered when using flags: testing the setting of a flag in the wrong byte of the data structure. Using similar names for all flags in a byte can help, but this practice severely restricts the range of mnemonic names possible. Mechanically undetectable typo - graphical errors become more probable in this unnecessarily dense name space.

To solve this problem, a set of macros have been defined for the OS/360 Assembler that permits using one name for a flag. Flags are defined using the DF and EQF macros. (These are used like the DS and EQU pseudo-ops

---

+ present address: Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, Illinois, 61801.

respectively.) These macros associate with each flag an absolute symbol and/or a relocatable symbol which can be used to reference the flag in an assembly. Each of these symbols encode both the byte in which the flag is located and the bits of the flag in that byte. Using these symbols, it is impossible to test or change bits in the wrong byte. Macros are also included for setting and testing flags defined in this way. Working together these macros efficiently and effectively implement the new machine data type BIT(n) and numerous operations defined on this new type. These macros have been used quite extensively in the coding of the PL/C compiler.

The implementation of the flag macros uses the length attribute associated with each flag symbol to record the position of the flag within its byte. The address of the flag symbol is the address of the byte that contains the flag. The macros hide from the user the gory details of (1) computing a length from the desired bit placement information, (2) generating the appropriate DS OAL(length) command to define the length of the flag symbol, and (3) using L' prefixes to retrieve the position information from the length attribute when the symbol is used in an instruction.

The rest of this report is divided into three sections. In the first section the macros for defining flags are presented. In the second section the macros for testing and changing flags are defined. The third section briefly defines the lower level macros used by the macros of sections I and II. The last section should be of interest only to persons wishing to expand the set of macros or those trying to locate a rare lingering error in the macros defined here.

SECTION I: MACROS FOR DEFINING FLAGS

FSECT Begin Flag Definition

An FSECT macro delimits the beginning of a contiguous block of bytes in which flags are to be located. The end of the block is delimited by the FEND macro (see below). Only DF, EQF, and ORF macros may appear between matching FSECT/FEND delimiters. They may only appear between these delimiters. A maximum of 100 flags may be defined within an FSECT. An FSECT may have at most 32 bytes.

The FSECT macro may be used to align the flag section on a specific boundary and to specify its size. The defaults for generating absolute and relocatable symbols for flags may also be specified. If absolute symbols are to be generated, the FSECT must appear in a named CSECT or DSECT. The absolute symbols so defined give the offset of the flag's byte from the beginning of this CSECT or DSECT.

The FSECT macro is written as follows:

```
[Label]  FSECT [ALIGN={X|H|F|D} ] [,WARN=YES ]
               [,SIZE=SYMBOL] [,BYTES={n| (n,MAX)} ]
               [,RELNM={ |*|prefix*|symbol} ]
               [,ABSNM={_|*|prefix*|symbol} ]
```

Label
    If present, will be defined as the first byte of the flag section.

ALIGN=
    specifies the alignment for the flag section. Byte alignment (X) is the default if ALIGN= is not coded.

BYTES=n
    specifies the number of bytes to be allocated to the flag section. The maximum allowed is 32. If BYTES=n is not coded, just enough bytes to accommodate all flags will be allocated.

BYTES=(n,MAX)
    specifies that up to n bytes are to be allocated to the FSECT. Only enough bytes to accommodate the flags defined will actually be allocated.

WARN=YES
    If this option is coded, MNOTE messages will be generated identifying bits within the flag section to which no flags have been assigned or to which more than one flag has been assigned.

SIZE=symbol
    if present, the symbol specified will be equated to the

RELNM and ABSNM specify the default action to be taken by the DF and EQF macros when generating absolute and relocatable symbols for a flag. RELNM specifies the default action for generating the relocatable symbol for a flag. ABSNM specifies the default action for generating the absolute symbol for a flag. RELNM and/or ABSNM may also be specified on the DF and EQF macros to override these defaults. The same conventions are followed for both RELNM and ABSNM. For brevity, these conventions will be given in terms of RELNM only. The conventions apply to the use of RELNM and ABSNM on the DF and EQF macros as well as to their use on the FSECT macro.

RELNM=
: specifies that no relocatable symbol for the flag is to be generated.

RELNM=*
: specifies that the name field of the DF or EQF macro is to be used as the relocatable symbol for the flag.

RELNM=prefix*
: specifies that the relocatable symbol for the flag is to be generated by prefixing the characters in the name field of the DF or EQF macro with the specified prefix. If the name field has a prefix (one mentioned in an ABSNM or RELNM option on either the FSECT macro or current DF or EQF macro) that prefix will be removed before the prefix in this option is applied. (Programming note: to generate the relocatable symbol 'prefixprefix...' under this option, 'prefixprefix...' must appear in the name field of the DF or EQF macro.)

RELNM=symbol
: specifies that the relocatable symbol for the flag is to be the symbol given with this option. It is not advisable to use this form of the RELNM option on the FSECT macro as it could easily lead to multiply-defined symbols.

The default values for ABSNM and RELNM on the FSECT macro are

ABSNM=,RELNM=*

DF Define Flag

The DF macro instruction is used to reserve storage for a flag in much the same way a DS reserves storage for other data structures.

The DF macro is written as follows:

```
[Symbol] DF    [BITS=n] [,POS=n] [,INIT=bits]
               [,RELNM={ |*|prefix*|symbol} ]
               [,ABSNM={ |*|prefix*|symbol} ]
```

Symbol
: is the name of the flag. References to this flag in following EQF or ORF macros must use this name. This symbol may also serve as the stem for the absolute or relocatable symbol for the flag.

BITS=n
: specifies the number of bits in the flag. n must not be greater than 8 or less than 1. The default is 1.

POS=n
: specifies the bit position within a byte where the flag is to start. Bit position 0 is the high order, or leftmost, bit of a byte. If POS specifies a bit position to the left of the current bit position, a new byte is started and the flag is placed in that byte. If POS=n is not coded, the flag is positioned at the current setting of the location counter. (See the discussion of the ORF macro.) If there are not enough bits remaining in this byte for the entire flag, the flag is positioned at bit 0 of the next byte.

INIT=bits
: 'bits' is a string of 0's and 1's specifying the initial value of the flag. If INIT=bits is coded, DC statements will be generated to do the initialization. The FSECT should be in a CSECT, not a DSECT, in this case. If INIT= is not coded, the flag is not initialized.

RELNM
: overrides the action of the RELNM option on the FSECT macro. See the description of the FSECT macro for details.

ABSNM
: overrides the action of the ABSNM option on the FSECT macro. See the description of the FSECT macro for details.

EQF Equate Flag

The EQF macro is used to equate a flag to another flag or to a collection of flags in the same byte.

The EQF macro is written as follows:

```
Symbol   EQF    {Flag[.n] | (Flag[.n],...,Flag[.n])}
                [,RELNM={ |*|prefix*|symbol} ]
                [,ABSNM={ |*|prefix*|symbol} ]
```

Symbol
: is the name of the flag to be defined. References to the flag in following EQF or ORF macros must specify this name. This symbol can also form the stem of the absolute or relocatable symbol for the flag.

Flag
: is a symbol which has appeared in the name field of a <u>preceding</u> DF or EQF macro instruction. Forward references are not permitted. The flag defined by this EQF macro will have the same bits in the same byte as that flag.

Flag.n
: When the bit qualifier, .n, is used the flag defined by the EQF macro is equivalent to bit n of the flag named in the operand field. n must be a single digit less than the number of bits in the flag. The leftmost bit of a flag is bit 0.

(Flag[.n],...,Flag[.n])
: The flag defined by the EQF macro when this format is used is the union of the bits in all the flags listed. The bits of the flags listed must not overlap. The flags must be defined in the same byte.

RELNM
: overrides the action of the RELNM option on the FSECT macro. See the description of the FSECT macro for details.

ABSNM
: overrides the action of the ABSNM option on the FSECT macro. See the description of the FSECT macro for details.

ORF Set Flag Location Counter

The ORF macro repositions the location counter in the PSECT in much the same way the ORG statement resets the location counter of the assembler. The location counter is initially at bit 0 of byte 0. It is moved to the bit following the flag defined by a DF macro after each DF macro is expanded. An EQF macro leaves the location counter unchanged.

The ORF macro is written as follows:

```
ORF    [{Flag | (Byte,Bit)}]
```

Flag

positions the location counter to bit zero of flag 'Flag'. 'Flag' must have appeared in the name field of an EQF or DF macro appearing before the ORF macro and in the same PSECT.

Byte

positions the location counter to the byte specified. 'Byte' may be written as b, *, *+n, or *-n, where 'b' and 'n' are unsigned decimal constants. 'b' indicates the b'th byte of an PSECT. (The first byte is byte 0.) '*' references the current byte position, and 'n' indicates a number of bytes before (-) or after (+) that byte.

Bit

positions the location counter to the <u>bit</u> specified. 'Bit' may be written as p, *, *+m, or *-m, where 'p' and 'm' are unsigned decimal constants. 'p' indicates the p'th bit of the previously specified byte. (The leftmost bit of a byte is bit 0.) 'p' must be less than 8. '*' references the current bit position, and 'm' indicates a number of bits before (-) or after (+) that bit.

If no option is coded on the ORF macro, the location counter is positioned to the bit following the last flag presently defined in the PSECT.

FEND End Flag Section

The FEND macro delimits the end of a flag section. If it is omitted, no flags in the FSECT will be defined.

The FEND macro is written as follows:

[Label] FEND

Label
: if present will define the last byte of the FSECT (after padding).

SECTION II: MACROS FOR TESTING AND CHANGING FLAGS

Flag Manipulation Macros

A number of macros are provided to assist in the testing and changing of flags defined by DF and EQF macros. All of these macros use the relocatable and/or absolute symbols for a flag in a consistent manner. How these symbols are used will now be explained.

In all of the macros defined below, the leftmost expression in the operand field is used as the address of the byte containing the flags mentioned in the macro. Normally this expression will be the relocatable or absolute symbol for a flag. In general, however, the leftmost expression may be any assembler expression which is valid in an RS instruction. The leftmost symbol of this expression, however, must be a flag symbol. If a relocatable symbol is used in this position the PSECT in which the flag was defined must be covered by some USING command at the point of macro expansion. If an absolute symbol is used it must be followed immediately with a register designation enclosed in parentheses. (This is exactly the format used by the assembler for an explicit base/displacement address.) The register so designated should contain the address of the DSECT or CSECT in which the flag was defined.

In the following descriptions, 'Flags' will mean either a single flag symbol or a list of flag symbols enclosed in parentheses. Bit qualifiers can not be used with these macros. The bits involved in an operation in which the list notation is used are the bits in the union of all the flags in the list. If the list notation is used, it is assumed that all flags have been defined in the same byte and do not overlap. This can not be checked by these macros. When 'Flags' coincides with the leftmost expression in the macro, the leftmost flag symbol may be replaced by an expression of the type defined in the previous paragraph. Only the leftmost flag symbol of that expression will determine which bits are to be contributed by that expression.

Examples in the following text assume the following FSECT has been defined:

```
B1       FSECT ABSNM=ABS*
F1       DF
F2       DF
F3       DF    BITS=3
F30      EQF   F3.0
F31      EQF   F3.1
F32      EQF   F3.2
F312     EQF   (F3.1,F32)
F4       DF    POS=0             Start a new byte
F5       DF    BITS=4
         ORF   F5                Overlay flag F5
F6       DF    BITS=2
F7       DF    BITS=4
         FEND
```

Based on this definition, the following would be legal 'Flags':

```
         F1
         F1+1                (Only as leftmost expression)
         F4-F4+*             (Only as leftmost expression)
         (F1,F2)
         (F4-1,F6)           (Only as leftmost expression)
         ABSF1(2)            (Only as leftmost expression)
         (ABSF6(2),ABSF7)    (Only as leftmost expression)
         (ABSF3(2),F2)       (Only as leftmost expression)
         ABSF2
         (ABSF1,ABSF3)
```

Note: if the last two examples were to appear in the leftmost expression, the instruction generated would reference the byte at absolute location 0.

The following would not be valid 'Flags':

```
         B1                   (Not defined as a flag)
         F1(2)                (F1 is not an absolute symbol)
         (F1,F5)              (Flags are not in same byte)
         (ABSF1(2),ABSF2(2))  (Two register designations)
         (F1,ABSF3(4))        (Rightmost item is not a symbol)
         (F5,F6)              (Flags overlap)
         *+F4-F4              (Leftmost symbol is not a flag)
```

SFZ Set Flag(s) Zero
SFO Set Flag(s) Ones
SFM Set Flag(s) Mixed

The SFZ, SFO, and SFM macros are used to explicitly set the value of a flag or flags in the same byte.

These macros are written as follows:

```
[Label]  SFZ    Flags1
[Label]  SFO    Flags2
[Label]  SFM    Flags1,Flags2
```

Flags1
: specifies the flags which are to be set to zero. All bits of these flags will be set zero.

Flags2
: specifies the flags which are to be set to ones. In the SFM macro, Flags2 should contain flag symbols only, not expressions. If Flags1 and Flags2 overlap, Flags2 has precedence.

Programming note: The only way to set a multi-bit flag to a value other than all ones or all zeros is to define a flag for each bit or group of bits of the multi-bit flag which must be set to one independently and then use the SFM macro. For example:

```
         SFM    F3,F30              Set F3 to 100
         SFM    F3,F312             Set F3 to 011
         SFM    F31,(F30,F32)       Set F3 to 101
```

INVF Invert Flag

The INVF macro is used to invert the setting of the bits of a flag or flags in the same byte. Bits which are ones will be changed to zeros and vice-versa.

The INVF macro is written as follows:

```
[Label]  INVF  Flags
```

Flags
: specifies the flags which are to be changed.

SFB Set Flag Byte

The SFB macro is used to set all flags in a single byte at the same time. Where applicable, SFB is more efficient than SFM.

The SFB macro is written as follows:

```
[Label]  SFB   Byte,{0|Flags}
```

Byte
: specifies the byte which is to be set. This may be a flag symbol or any assembler expression which is valid in an RS instruction. Even if a flag symbol is used, all bits of the byte will be affected, not just the bits of the flag.

Flags
: specifies the flags of the byte which are to be set to ones. The remaining bits in the byte will be set to zero.

For example:

```
         SFB    B1,F3              Set byte B1 to 00111000
         SFB    F3,(F1,F312)       Set byte B1 to 10011000
         SFB    F2,0               Set byte B1 to 00000000
```

TF Test Flags

The TF macro is used to test the value of a flag or group of flags. The code generated by this macro sets the condition code to indicate whether the flags tested were all ones, all zeros or mixed. The standard BC instructions may be used following the TF macro to branch according to the result of the test.

The TF macro is written as follows:

```
[Label]  TF    Flags
```

Flags
: specifies the flags to be tested.

For example:

```
         TF     F1
         BO     L1                 -> F1 is 1
         TF     (F1,F2)
         BZ     *+12               -> F1 and F2 are 0.
         BO     L1                 -> F1 and F2 are 1.
         BM     L1                 -> F1 and F2 are not the same
         TF     F3
         BMR    9                  -> F3 is neither 111 nor 000
```

```
BFZ     Branch If Flag(s) Zero
BFNZ    Branch If Flag(s) Not Zero
BFO     Branch If Flag(s) One
BFNO    Branch If Flag(s) Not One
BFM     Branch If Flag(s) Mixed
BFNM    Branch If Flag(s) Not Mixed
```

These macros generate code to branch to a specified location under the conditions indicated.

They are written as follows:

```
[Label] BFx   Flags,{Location|(Register Designation)}
```

Flags
specifies the flags to be tested.

Location
specifies the location to which control is to pass if the condition being tested is found to be true. This may be any assembler expression valid in a BC instruction. * notation may be used. Assume an instruction length of 4 when using the * notation.

(Register Designation)
specifies that the location to branch to is in a register. Any assembler expression valid in an RR instruction is valid.

The following is equivalent to the example given above:

```
BFO   F1,L1            -> F1 is 1
BFZ   (F1,F2),*+12     -> F1 and F2 are 0
BO    L1               -> F1 and F2 are 1
BM    L1               -> F1 and F2 are not the same
BFM   F3,(9)           -> F3 is neither 111 nor 000
```

```
DM      Define Mask
DCM     Define Complement Mask
```

These macros assemble into a DC statement defining a byte which contains a one (zero) in each bit position of a flag or group of flags in the same byte. The remaining bits of the byte will be zero (one).

These macros are written as follows:

```
[Label] DM    Flags
[Label] DCM   Flags
```

Flags
specifies the bits which are to be set to ones (zeros) in the generated byte.

CF Compare Flag

The CF macro is used to test a multi-bit flag for a specific value. The code generated will set the condition code to indicate whether the comparison was equal or not.

The CF macro is written as follows:

```
[Label]  CF    Flags1,Flags2
```

Flags1
: specifies the flags to be tested.

Flags2
: specifies the bits in Flags1 which are to be tested for ones. The bits in Flags1, but not in Flags2 will be tested for zero. Flags2 should contain flag symbols only, not expressions.

For example:

```
         USING B1,9
         CF    ABSF3(9),F31
         BE    L1                  -> F3 has the value 010
         CF    ABSF3(9),(F30,F32)
         BNE   L1                  -> F3 is not 101
         CF    (ABSF7(9),ABSF6),F5
         BER   8                   -> F6 is 11 and F7 is 1100
```

CFB Compare Flag Byte

The CFB macro compares an entire flag byte to a specific value. It is equivalent to a CF macro in which Flags1 contains all 8 bits of a byte, except that CFB is more efficient, and sets the condition code to indicate an ordering for the operands, not just their equality.

The CFB macro is written as follows:

```
[Label]  CFB   Byte,{0|Flags}
```

Byte
: specifies the byte to be tested. It may be a flag symbol or any assembler expression valid in an RS instruction.

Flags
: specifies the one bits of a mask to be compared with the byte. The bits not in Flags will be zero in the mask.

For example:

```
         CFB   B1+1,F7
         BE    L1                  -> Byte B1+1 is 00011110
         CFB   F5,(F6,F7)
         BL    L1                  -> Byte B1+1 is < 01111110
```

BFE Branch Flag(s) Equal
BFNE Branch Flag(s) Not Equal

These macros generate code to branch to a specified location under the condition specified.

They are written as follows:

```
[Label]  BFx   Flags1,Flags2,{Location|(Register Designation)}
```

Flags1
specifies the flags to be tested.

Flags2
specifies the bits of Flags1 which are to be tested for ones. The remaining bits of Flags1 will be tested for zeros. Flags2 should contain flag symbols only, not expressions.

Location
same as for BFZ. Note: because the Location option is used in two instructions generated by the BFE macro, * notation should not be used in the Location field of this macro.

(Register Designation)
same as for BFZ.

The following example is equivalent to the example given for the CP macro:

```
     BFE   F3,F31,L1           -> F3 has the value 010
     BFNE  F3,(F30,F32),L1     -> F3 is not 101
     BFE   (F7,F6),F5,(8)      -> F6 is 11 and F7 is 1100
```

Programming note: When testing a flag for all zeros or all ones, the macros BFZ, BFNZ, BFO, and BFNO should be used since the code they generate is more efficient than that generated by BFE and BFNE.

SECTION III: LOWER LEVEL MACROS

QFIX

The QFIX macro is called by the DF and EQF macros to generate the absolute and relocatable symbols for the flag.

The QFIX macro is written as follows:

```
QFIX  Symbol,Relnm,Absnm
```

Symbol
: is the symbol from the name field of the calling macro.

Absnm
: is the ABSNM= parameter from the calling macro.

Relnm
: is the RELNM= parameter from the calling macro.

QFIND

The QFIND macro is called by ORF and EQF to locate the internal table entry for a previously defined flag name. The search may be confined to a single byte.

The QFIND macro is written as follows:

```
QFIND Symbol,[BYTE=n]
```

Symbol
: is the argument of the search.

BYTE=n
: if coded, and n is not zero, indicates that the name must be defined in byte n-1.

QFAX

The QFAX macro is called by QFIX to decode and merge the ABSNM and RELNM options from the FSECT macro and the macro which called QFIX. The result is a generated symbol - relocatable or absolute.

The QFAX macro is written as follows:

```
QFAX  Symbol,Option1,Default1,Option2,Default2
```

Symbol
: is the symbol from the name field of the macro calling QFIX. This forms the stem of the generated symbol.

Option1
: is the RELNM or ABSNM option from the macro calling QFIX depending on whether a relocatable or an absolute symbol is to be generated. It determines how the generated symbol is to be constructed from the stem according to the rules given in the FSECT macro.

Default1
: is the RELNM (ABSNM) option from the FSECT macro. It controls generation of the symbol if Option1 is '?'. It also supplies a prefix which may have to be removed from Symbol before the proper prefix is applied.

Option2
: is the ABSNM (RELNM) option from the macro calling QFIX. It supplies a prefix which may have to be removed from Symbol before the proper prefix is applied.

Default2
: is the ABSNM (RELNM) option from the FSECT macro. It serves the same purpose as Option2.

QF

The QF macro is called by all of the macros in section II to generate flag referencing instructions.

The QF macro is wriiten as follows:

```
[Label]  QF     Op,Flags,[S={+|-}][,INIT=string]
                [,TYPE=string][,BYTE=RSexpression]
```

Label
: specifies the label field of the generated instruction.

Op
: is the operation field of the instruction to be generated.

Flags
: is a list of flag expressions (following the rules discussed in section II) specifying the flags (bits of a byte) to be tested or changed by the instruction.

S={+|-}
: indicates whether the flag bits are to be added to or subtracted from the INIT value given. The default is S=+.

INIT=string
: specifies the initial value for the immediate field of the SI instruction generated. This value will be modified by the addition or subtraction of flag bits according to the S= parameter. Currently only INIT=255- is used, in conjunction with S=-, to generate complement masks.

TYPE=string
: specifies the instruction format. If TYPE is not coded or TYPE= is coded, an SI instruction is generated. Otherwise, a 'DS-type' instruction is generated as follows:

```
Label    Op    TYPE(decoded Flags)
```

Currently, TYPE=AL1 with Op a DC or DS and TYPE=0+ with Op an EQU are used by the macros in section II.

BYTE=RSexpression
: specifies the address field of the SI instruction to be generated. If not coded, Flags(1) is used for this field.