The ACE FORTRAN 77 Compiler for the MC68000

Willem Wakker

(c) ACE - Associated Computer Experts bv,
Nieuwezijds Voorburgwal 314
1012 RV Amsterdam.

ABSTRACT

This report describes the implementation details of the ACE FORTRAN 77 compiler for the MC68000. The ACE f77 compiler is an upgraded version of the UNIX f77 compiler. This compiler implements fully the FORTRAN language as specified in the American National Standard programming language FORTRAN 77, ANSI X3.9-1978, so only extensions and deviations to the definition are mentioned.

Information about data-types (sizes and alignment) will be given, together with a description of the calling sequence, parameter passing and register usage.

February 10, 1984

NAME

f77 Fortran 77 compiler

SYNOPSIS

f77 [option] ... file ...

DESCRIPTION

F77 is the ACE Fortran 77 compiler. It accepts several types of arguments:

Arguments whose names end with `.f' are taken to be Fortran 77 source programs; they are compiled, and each object program is left on the file in the current directory whose name is that of the source with `.o' substituted for `.f'.

In the same way, arguments whose names end with `.c' or `.s' are taken to be C or assembly source programs and are compiled or assembled, producing a `.o' file.

The following flags are understood.

-c Suppress loading and produce `.o' files for each source file.

-w Suppress all warning messages.

-w66 Only Fortran 66 compatibility warnings are suppressed.

-p Prepare object files for profiling, see *prof*(1).

-S Compile the named programs, and leave the assembler-language output on corresponding files suffixed `.s'. (No `.o' is created).

-o output
Name the final output file output instead of `a.out'.

-onetrip
Compile DO loops that are performed at least once if reached. (Fortran 77 DO loops are not performed at all if the upper limit is smaller than the lower limit.)

-U Do not convert upper case letters to lower case. The default is to convert Fortran programs to lower case.

-u Make the default type of a variable `undefined' rather than using the default Fortran rules.

-I2 On machines which support short integers, make the default integer constants and variables short. (-I4 is the standard value of this option). All logical quantities will be short.

-C Compile code to check that subscripts are within declared array bounds.

-V The Verbose or View switch. Shows the various passes of the compiler as they are called by f77, with their switches and intermediate files.

-O<d>
Invoke the assembly code optimiser after the assembly code generation. When the O is followed by a numeric digit, <d> data registers are (at most) allocated by the f77 front-end for holding loop variables.

-N This switch allocates more space for several tables. It is normally used after the front-end has given a fatal compiler error. Usage: -Ntddd where t is one of 'q', 'x', 'c', 'n' (for equivalence table, external name table, control-structure table and name table) while ddd is the new number of elements this table at least must have (the fatal compiler error message has given the current number of entries). When more than one table has to be enlarged, the option looks like "-Nq300-Nx400".

FILES

file.[fsc]	input file
file.o	object file
a.out	loaded output
/usr/bin/f77	the driver program
/usr/lib/f77pass1	front-end of the f77 compiler
/usr/lib/int_conv	a conversion program for intermediate files
/usr/lib/pcc2	the code generator
/usr/lib/o68	the assembly code optimizer
/usr/lib/as_conv	a conversion program that modifies the assembly
/usr/lib/libF77.a	intrinsic function library
/usr/lib/libI77.a	Fortran I/O library
/lib/libc.a	C library, see section 3
/lib/libm.a	C library, see section 3

SEE ALSO

S.I. Feldman, P.J. Weinberger, A Portable Fortran 77 Compiler
prof(1), cc(1), ld(1)

DIAGNOSTICS

The diagnostics produced by f77 itself are intended to be self-explanatory. Occasional messages may be produced by the loader.

1. Introduction

The ACE f77 compiler for the MC68000 is an upgraded version of the UNIX f77 compiler [7]. It implements fully the FORTRAN 77 language as specified in [3]. Users of the f77 compiler are advised to get a copy of this definition, since many common dialects of FORTRAN deviate from this specification.

The f77 runtime environment is almost completely explained in [6], together with the interface for C routines and functions.

2. Data types, sizes and alignment

The f77 data types are specified in [3], the correspondence between the f77 data types and C data types are given in paragraph 4.2 of [7]. The f77 data types inherit (when not conflicting with [3]) their alignment for the C types.

Note that separately compiled f77 routines must all have the same assumption about the integer size, so that when one file is compiled with the -I2 switch, all files must be compiled with this switch.

The MC68000 data organisation implies that all addresable data elements are addressed via the address of the byte which contains the most significant bit of the data element (so the address of the least significant byte of the data element is always equal to or higher than the address of the data element).

The addressing scheme of the 68000 also requires that multi-byte data (2 bytes or 4 bytes) are always accessed on word (even byte) boundaries. This implies the alignment of the various data elements.

For more detailed information on the 68000 data organisation and addressing capabilities, see chapter 2 of [5].

The following data types are implemented:

character

These values occupy 8 bits (1 byte) and can be aligned on any byte boundary. The value of a char ranges from -128 to +127.

integer*2

These values occupy 16 bits (2 bytes) and are (must be) aligned on word (even byte) boundaries. The value of a short ranges from -32768 to +32767.

integer
logical

These values occupy 32 bits (4 bytes) and are (must be) aligned on word (even byte) boundaries. The value of an integer ranges from -2147483648 to +2147483647.

real

Elements of the type real occupy 32 bits (4 bytes) and are (must be) aligned on word (even byte) boundaries. All real values are converted to double precision values for arithmetic operations. A real value consists of a sign bit (most significant bit), followed by an 8-bit biased exponent and a 23 bit mantissa.

double precision

Elements of the type double precision occupy 64 bits (8 bytes) and are (must be) aligned on word (even byte) boundaries. A double precision value consists of a sign bit (most significant bit), followed by an 8-bit biased exponent and a 55 bit mantissa.

complex

The complex data type consists of two floating point (real) numbers, the first (lowest address) being the real part and the second the imaginary part.

double complex

As for the complex, but with each element being a double precision number.

arrays

Elements of an array may be of any of the types mentioned above and are stored contiguously in memory. The (base) address of an array is always aligned on a word (even byte) boundary. Elements of multi-dimensional arrays are stored in column major order.

3. Calling sequence and register usage

All parameter passing in f77 is done by reference (according to the standard). In normal use, the default register allocation scheme of the code generator is used. With the -o switch, at most six loop-variables are put in data registers.

4. Additions and remarks

The ACE f77 compiler front-end for the 68000 reflects completely the standard [3], its implementation is fully described in [7].

Here we only give some examples of problems an average f77 programmer will get, when trying to conform to the standard. This list is by no means complete, it only gives hints (see also the list of differences between F66 and f77 in the appendix of [7]).

(1) Upper - Lower case
The compiler expects lower case source input. Upper case is converted to lower case except in character constants. Be careful: a format string, as interpreted by the run time package, has to be in lower case, so implicit format strings (= character constant) or arrays with format information, cannot use upper case letters.

(2) Integer size
The standard size for an integer is four bytes. This does not only effect the size of integer variables, but also the size of integer constants, and the results of integer expressions. With the -I2 compiler switch this standard can be set to two bytes. It will effect the size of integer variables, integer constants and the results of integer expressions, but integer intermediate expressions will stay four bytes long.

In the MC68000 the address of an object is the address of its high order byte. This means that it is not possible to access a two byte integer value through the address of a four byte integer. As a consequence: routines which communicate with other routines, using integer constants or expressions as parameters, have to be compiled with the same integer size.

(3) Logical size
A logical*1 size is not defined in the standard, and consequently not supported. The only way to introduce objects of single byte size is through the character*n construct.

(4) Hollerith
The Hollerith construct from FORTRAN 66 is not available in FORTRAN 77. The f77 compiler supports Hollerith in data initialisation statements. In general Hollerith cannot be used in integer expressions, but no error or warning is generated. When a Hollerith is used as in:

```
     IVAR = 2Hab        or        IVAR .EQ. 2hab
```

a two byte string constant is generated.
The address of the high order byte is used for access, but with a move or compare instruction for a single byte only.

(5) Block structures
The only block structure supported by the standard is the Block If. Other commonly used constructs for Do loops (like Do ... Enddo, Do While, or Do ... Until) are not supported.

(6) Pre-connected files

Unit	FORTRAN 77	FD	UNIX	Status
0	not known	2	standard error	pre-connected
5	read	0	standard input	pre-connected
6	write	1	standard output	to be opened
x	any file	y	fort.x (0<x<10)	to be opened

(7) Do loops
A step size of zero is forbidden by the standard, and not supported in f77.
Statements which change the index variable of the loop are also forbidden, but this is possible in f77 and does not generate a warning or error.

(8) Variable addresses
The mapping of local variables onto the private address space of a routine is not straightforward, so variables which have been declared in some order will in general not be assigned to memory in that same order.
The only way to force the allocation of memory in a certain order is through the use of a set of equivalence statements between individual variables and the elements of an array.

(9) Array order
The order in which elements of a multi-dimensional array are stored in memory is different in f77 and C.

(10) End of record
The end of a record in a formatted file is always given by the newline character (UNIX: a linefeed or decimal 10).
It is not possible to suppress the automatic generation of this character at the end of an output record.

(11) Open statement
In addition to paragraph 6.8.1 of [7] and 12.10.1 of [3]: when no file parameter is given and no status parameter is given, status is assumed scratch (instead of old).

(12) Record length
When a file is declared to have records of length r (recl = r), while only n bytes (n < r) are written per record, then only n bytes can be read from the last record of the file (record with the highest sequence number).

5. Invocation and compiler switches

The f77 driver program accepts all switches as given in the f77 documentation ([7], [9]), except those affecting M4, Ratfor and EFL files and programs; the latter ones are not implemented.
Specials and remarks:

-V The Verbose or View switch. Shows the various passes of the compiler as they are called by f77, with their switches and intermediate files.

-O<d> Invoke the assembly code optimiser after the assembly code generation. When the O is followed by a numeric digit, <d> data registers are (at most) allocated by the f77 front-end for holding loop variables.

-N This switch allocates more space for several tables. It is normally used after the front-end has given a fatal compiler error.
Usage: -Ntddd where t is one of 'q', 'x', 'c', 'n' (for equivalence table, external name table, control-structure table and name table) while ddd is the new number of elements this table at least must have (the fatal compiler error message has given the current number of entries). When more than one table has to be enlarged, the option looks like "-Nq300-Nx400".

6. The MOTOROLA 68000 code generator

The code generator produces assembly code for several, language dependant, front-ends. Most of the typing and specific register usage is dictated by the front-ends.
The code generator produces assembly code, that includes almost all available instructions and addressing modes (including the Address Register Indirect With Index addressing mode).

The UNIX system requires a dynamically growing stack. Unfortunately the 68000 can produce address errors, when addressing outside the stack, that cannot be backed-up. Therefore provisions have to be made to prevent these errors. For this purpose, the code generator produces "tst.b -XX(sp)" instructions each time the stack pointer is decremented (on procedure entry, and before procedure calls with parameters). When such a tst.b instruction produces an address error, the stack segment can be enlarged, and any subsequent stack references cannot cause an error anymore.

7. The stack layout

On each procedure/function entry the following code is produced:

```
        tst.b     -<disp>(sp)               stack test
        link      (a6),#-<disp>             install new frame
        movem.l   -<disp>(a6),<reg list>    save used registers
```

While on return the following code is produced

```
        movem.l   <reg list>,-disp>(a6)     restore used registers
        unlk      a6
        rts
```

In these pieces of code <disp> is the total amount of space (in bytes) needed for this procedure to store its automatics, its temporary results and to save the registers; <reg list> is a list of registers used (allocated) local to this function, so effectively only those registers are saved that are actually used. Both <disp> and <reg list> are generated by the code generator.
Note that the code generator NEVER saves the following registers: D0, D1, A0, A1, A6 and A7.

This implies the following stack lay-out

```
       new sp -> |-----------------|
                 | saved registers |
                 |-----------------|
                 | temporaries     |  allocated by code generator
                 |-----------------|
                 | automatics      |
                 |-----------------|
       new A6 -> | old A6          |
                 |-----------------|
                 | return address  |
                 |-----------------|
                 | argument 1      |
                 |-----------------|
                 | argument 2      |
       old sp -> |-----------------|
```

All 'stack' variables (automatics, function arguments) are addressed via A6 (automatics having a negative offset, and arguments having a positive offset).

The clean-up of the stack after a procedure call with parameters is done by the calling procedure.

8. References

[3] American National Standard programming language FORTRAN, ANSI X3.9-1978.

[5] The MC68000 16-bit microprocessor, User's Manual, Motorola.

[6] 68343 Fast Floating-point reference manual, M68KFFP(D1) July 1981, Motorola.

[7] S.I.Feldman and P.J.Weinberger, A Portable Fortran 77 Compiler, UNIX programmer's manual, 7th edition, volume 2, chapter 21.

[9] UNIX programmer's manual, 7th edition, Volume 1.

NAME

f77 Fortran 77 compiler

SYNOPSIS

f77 [option] ... file ...

DESCRIPTION

F77 is the ACE Fortran 77 compiler. It accepts several types of arguments:

Arguments whose names end with `.f' are taken to be Fortran 77 source programs; they are compiled, and each object program is left on the file in the current directory whose name is that of the source with `.o' substituted for `.f'.

In the same way, arguments whose names end with `.c' or `.s' are taken to be C or assembly source programs and are compiled or assembled, producing a `.o' file.

The following flags are understood.

-c Suppress loading and produce `.o' files for each source file.

-w Suppress all warning messages.

-w66 Only Fortran 66 compatibility warnings are suppressed.

-p Prepare object files for profiling, see prof(1).

-S Compile the named programs, and leave the assembler-language output on corresponding files suffixed `.s'. (No `.o' is created).

-o output
Name the final output file output instead of `a.out'.

-onetrip
Compile DO loops that are performed at least once if reached. (Fortran 77 DO loops are not performed at all if the upper limit is smaller than the lower limit.)

-U Do not convert upper case letters to lower case. The default is to convert Fortran programs to lower case.

-u Make the default type of a variable `undefined' rather than using the default Fortran rules.

-I2 On machines which support short integers, make the default integer constants and variables short. (-I4 is the standard value of this option). All logical quantities will be short.

-C Compile code to check that subscripts are within declared array bounds.

-V The Verbose or View switch. Shows the various passes of the compiler as they are called by f77, with their switches and intermediate files.

-O<d>
Invoke the assembly code optimiser after the assembly code generation. When the O is followed by a numeric digit, <d> data registers are (at most) allocated by the f77 front-end for holding loop variables.

-N This switch allocates more space for several tables. It is normally used after the front-end has given a fatal compiler error. Usage: -Ntddd where t is one of 'q', 'x', 'c', 'n' (for equivalence table, external name table, control-structure table and name table) while ddd is the new number of elements this table at least must have (the fatal compiler error message has given the current number of entries). When more than one table has to be enlarged, the option looks like "-Nq300-Nx400".

FILES

file.[fsc]	input file
file.o	object file
a.out	loaded output
/usr/bin/f77	the driver program
/usr/lib/f77pass1	front-end of the f77 compiler
/usr/lib/int_conv	a conversion program for intermediate files
/usr/lib/pcc2	the code generator
/usr/lib/o68	the assembly code optimizer
/usr/lib/as_conv	a conversion program that modifies the assembly
/usr/lib/libF77.a	intrinsic function library
/usr/lib/libI77.a	Fortran I/O library
/lib/libc.a	C library, see section 3
/lib/libm.a	C library, see section 3

SEE ALSO
S.I. Feldman, P.J. Weinberger, A Portable Fortran 77 Compiler
prof(1), cc(1), ld(1)

DIAGNOSTICS
The diagnostics produced by f77 itself are intended to be self-explanatory. Occasional messages may be produced by the loader.

The ACE FORTRAN 77 Compiler for the MC68000

Willem Wakker

(c) ACE - Associated Computer Experts bv,
Nieuwezijds Voorburgwal 314
1012 RV Amsterdam.

ABSTRACT

This report describes the implementation details of the ACE FORTRAN 77 compiler for the MC68000. The ACE f77 compiler is an upgraded version of the UNIX f77 compiler. This compiler implements fully the FORTRAN language as specified in the American National Standard programming language FORTRAN 77, ANSI X3.9-1978, so only extensions and deviations to the definition are mentioned.

Information about data-types (sizes and alignment) will be given, together with a description of the calling sequence, parameter passing and register usage.

February 10, 1984

1. Introduction

The ACE f77 compiler for the MC68000 is an upgraded version of the UNIX f77 compiler [7]. It implements fully the FORTRAN 77 language as specified in [3]. Users of the f77 compiler are advised to get a copy of this definition, since many common dialects of FORTRAN deviate from this specification.

The f77 runtime environment is almost completely explained in [6], together with the interface for C routines and functions.

2. Data types, sizes and alignment

The f77 data types are specified in [3], the correspondence between the f77 data types and C data types are given in paragraph 4.2 of [7]. The f77 data types inherit (when not conflicting with [3]) their alignment for the C types.

Note that separately compiled f77 routines must all have the same assumption about the integer size, so that when one file is compiled with the -I2 switch, all files must be compiled with this switch.

The MC68000 data organisation implies that all addresable data elements are addressed via the address of the byte which contains the most significant bit of the data element (so the address of the least significant byte of the data element is always equal to or higher than the address of the data element).

The addressing scheme of the 68000 also requires that multi-byte data (2 bytes or 4 bytes) are always accessed on word (even byte) boundaries. This implies the alignment of the various data elements.

For more detailed information on the 68000 data organisation and addressing capabilities, see chapter 2 of [5].

The following data types are implemented:

character

These values occupy 8 bits (1 byte) and can be aligned on any byte boundary. The value of a char ranges from -128 to +127.

integer*2

These values occupy 16 bits (2 bytes) and are (must be) aligned on word (even byte) boundaries. The value of a short ranges from -32768 to +32767.

integer
logical

These values occupy 32 bits (4 bytes) and are (must be) aligned on word (even byte) boundaries. The value of an integer ranges from -2147483648 to +2147483647.

real

Elements of the type real occupy 32 bits (4 bytes) and are (must be) aligned on word (even byte) boundaries. All real values are converted to double precision values for arithmetic operations. A real value consists of a sign bit (most significant bit), followed by an 8-bit biased exponent and a 23 bit mantissa.

double precision

Elements of the type double precision occupy 64 bits (8 bytes) and are (must be) aligned on word (even byte) boundaries. A double precision value consists of a sign bit (most significant bit), followed by an 8-bit biased exponent and a 55 bit mantissa.

complex

The complex data type consists of two floating point (real) numbers, the first (lowest address) being the real part and the second the imaginary part.

double complex

As for the complex, but with each element being a double precision number.

arrays

Elements of an array may be of any of the types mentioned above and are stored contiguously in memory. The (base) address of an array is always aligned on a word (even byte) boundary. Elements of multi-dimensional arrays are stored in column major order.

3. Calling sequence and register usage

All parameter passing in f77 is done by reference (according to the standard). In normal use, the default register allocation scheme of the code generator is used. With the -o switch, at most six loop-variables are put in data registers.

4. Additions and remarks

The ACE f77 compiler front-end for the 68000 reflects completely the standard [3], its implementation is fully described in [7].

Here we only give some examples of problems an average f77 programmer will get, when trying to conform to the standard. This list is by no means complete, it only gives hints (see also the list of differences between F66 and f77 in the appendix of [7]).

(1) Upper - Lower case
The compiler expects lower case source input. Upper case is converted to lower case except in character constants. Be careful: a format string, as interpreted by the run time package, has to be in lower case, so implicit format strings (= character constant) or arrays with format information, cannot use upper case letters.

(2) Integer size
The standard size for an integer is four bytes. This does not only effect the size of integer variables, but also the size of integer constants, and the results of integer expressions. With the -I2 compiler switch this standard can be set to two bytes. It will effect the size of integer variables, integer constants and the results of integer expressions, but integer intermediate expressions will stay four bytes long.

In the MC68000 the address of an object is the address of its high order byte. This means that it is not possible to access a two byte integer value through the address of a four byte integer. As a consequence: routines which communicate with other routines, using integer constants or expressions as parameters, have to be compiled with the same integer size.

(3) Logical size
A logical*1 size is not defined in the standard, and consequently not supported. The only way to introduce objects of single byte size is through the character*n construct.

(4) Hollerith
The Hollerith construct from FORTRAN 66 is not available in FORTRAN 77. The f77 compiler supports Hollerith in data initialisation statements. In general Hollerith cannot be used in integer expressions, but no error or warning is generated. When a Hollerith is used as in:

```
    IVAR = 2Hab        or        IVAR .EQ. 2hab
```

a two byte string constant is generated.
The address of the high order byte is used for access, but with a move or compare instruction for a single byte only.

(5) Block structures
The only block structure supported by the standard is the Block If. Other commonly used constructs for Do loops (like Do ... Enddo, Do While, or Do ... Until) are not supported.

(6) Pre-connected files

```
Unit FORTRAN 77     FD UNIX                   Status
---- ----------     -- ----                   ------
  0  not known       2 standard error         pre-connected
  5  read            0 standard input         pre-connected
  6  write           1 standard output        to be opened
  x  any file        y fort.x (0<x<10)        to be opened
```

(7) Do loops
A step size of zero is forbidden by the standard, and not supported in f77.
Statements which change the index variable of the loop are also forbidden, but this is possible in f77 and does not generate a warning or error.

(8) Variable addresses
The mapping of local variables onto the private address space of a routine is not straightforward, so variables which have been declared in some order will in general not be assigned to memory in that same order.
The only way to force the allocation of memory in a certain order is through the use of a set of equivalence statements between individual variables and the elements of an array.

(9) Array order
The order in which elements of a multi-dimensional array are stored in memory is different in f77 and C.

(10) End of record
The end of a record in a formatted file is always given by the newline character (UNIX: a linefeed or decimal 10).
It is not possible to suppress the automatic generation of this character at the end of an output record.

(11) Open statement
In addition to paragraph 6.8.1 of [7] and 12.10.1 of [3]: when no file parameter is given and no status parameter is given, status is assumed scratch (instead of old).

(12) Record length
When a file is declared to have records of length r (recl = r), while only n bytes (n < r) are written per record, then only n bytes can be read from the last record of the file (record with the highest sequence number).

5. Invocation and compiler switches

The f77 driver program accepts all switches as given in the f77 documentation ([7], [9]), except those affecting M4, Ratfor and EFL files and programs; the latter ones are not implemented.
Specials and remarks:

-V The Verbose or View switch. Shows the various passes of the compiler as they are called by f77, with their switches and intermediate files.

-O<d> Invoke the assembly code optimiser after the assembly code generation. When the O is followed by a numeric digit, <d> data registers are (at most) allocated by the f77 front-end for holding loop variables.

-N This switch allocates more space for several tables. It is normally used after the front-end has given a fatal compiler error.
Usage: -Ntddd where t is one of 'q', 'x', 'c', 'n' (for equivalence table, external name table, control-structure table and name table) while ddd is the new number of elements this table at least must have (the fatal compiler error message has given the current number of entries). When more than one table has to be enlarged, the option looks like "-Nq300-Nx400".

6. The MOTOROLA 68000 code generator

The code generator produces assembly code for several, language dependant, front-ends. Most of the typing and specific register usage is dictated by the front-ends.
The code generator produces assembly code, that includes almost all available instructions and addressing modes (including the Address Register Indirect With Index addressing mode).

The UNIX system requires a dynamically growing stack. Unfortunately the 68000 can produce address errors, when addressing outside the stack, that cannot be backed-up. Therefore provisions have to be made to prevent these errors. For this purpose, the code generator produces "tst.b -XX(sp)" instructions each time the stack pointer is decremented (on procedure entry, and before procedure calls with parameters). When such a tst.b instruction produces an address error, the stack segment can be enlarged, and any subsequent stack references cannot cause an error anymore.

7. The stack layout

On each procedure/function entry the following code is produced:

```
        tst.b     -<disp>(sp)                stack test
        link      (a6),#-<disp>              install new frame
        movem.l   -<disp>(a6),<reg list>     save used registers
```

While on return the following code is produced

```
        movem.l   <reg list>,-disp>(a6)      restore used registers
        unlk      a6
        rts
```

In these pieces of code <disp> is the total amount of space (in bytes) needed for this procedure to store its automatics, its temporary results and to save the registers; <reg list> is a list of registers used (allocated) local to this function, so effectively only those registers are saved that are actually used. Both <disp> and <reg list> are generated by the code generator.
Note that the code generator NEVER saves the following registers: D0, D1, A0, A1, A6 and A7.

This implies the following stack lay-out

```
      new sp -> |-----------------|
                | saved registers |
                |-----------------|
                | temporaries     |   allocated by code generator
                |-----------------|
                | automatics      |
                |-----------------|
      new A6 -> | old A6          |
                |-----------------|
                | return address  |
                |-----------------|
                | argument 1      |
                |-----------------|
                | argument 2      |
      old sp -> |-----------------|
```

All 'stack' variables (automatics, function arguments) are addressed via A6 (automatics having a negative offset, and arguments having a positive offset).

The clean-up of the stack after a procedure call with parameters is done by the calling procedure.

8. References

[3] American National Standard programming language FORTRAN, ANSI X3.9-1978.

[5] The MC68000 16-bit microprocessor, User's Manual, Motorola.

[6] 68343 Fast Floating-point reference manual, M68KFFP(D1) July 1981, Motorola.

[7] S.I.Feldman and P.J.Weinberger, A Portable Fortran 77 Compiler, UNIX programmer's manual, 7th edition, volume 2, chapter 21.

[9] UNIX programmer's manual, 7th edition, Volume 1.