Programmer's Guide

to ALGOL W

Howard Koslow

June 20, 1977

Computer Science Program

Brown University

Providence, Rhode Island 02912

This research is being supported by the National Science Foundation, Grant GJ-41539, the Office of Naval Research, Contract N00014-75-C-0427, and the Brown University Division of Applied Mathematics. Principal Investigator: Professor Andries van Dam.

TABLE OF CONTENTS

1 Compilation and Execution.....2
- 1.1 Deck Setup.....2
- 1.2 %ALGOL Cards.....2
- 1.3 Option Cards.....3
- 1.4 Debug Options.....4
- 1.5 Dump Options.....5
- 1.6 Compiling and Executing Under OS/360.....6
- 1.7 Compiling and Executing Under CMS.....7
- 1.8 Assembling Programs to Interface with ALGOL W.....9
- 1.9 Compiling/Link-editing Programs for the Meta4A.....9
- 1.10 OCCAM: Handling ALGOL W Text Decks.....10

2 ALGOL W on the Meta4A.....12
- 2.1 Program Execution.....12
- 2.2 Input/Output.....13
- 2.3 Debugging.....14
- 2.4 Runtime Error Messages.....14
- 2.5 Implementation Restrictions.....16

3 ALGOL Internals.....17
- 3.1 Internal Data Representation.....17
- 3.2 Meta4A Cross-compiler Considerations.....18
- 3.3 Runtime Environment.....19
- 3.4 Standard Functions.....21

4 Interfacing ALGOL W with Assembler.....23
- 4.1 External Procedures.....23
- 4.2 FORTRAN Linkage Conventions.....23
  - 4.2.1 On the /360.....24
  - 4.2.2 On the Meta4A.....24
- 4.3 ALGOL W Linkage Conventions.....25
- 4.4 The AWX-Macros.....25
  - 4.4.1 AWXMAIN.....26
  - 4.4.2 AWXPROC.....26
  - 4.4.3 AWXENDAU.....28
  - 4.4.4 AWXCALL.....28
  - 4.4.5 AWXEXIT.....29
  - 4.4.6 AWXLHV and AWXRHV.....30
  - 4.4.7 AWXARRAY, List Form.....31
  - 4.4.8 AWXARRAY, Execute Form.....32
  - 4.4.9 AWXALLO.....32
  - 4.4.10 AWXFREE.....33
  - 4.4.11 Sample Program.....34

5 Programming and Coding Hints.....35

6 References.....37

0 Preface

This document is a programmer's guide to ALGOL W on the Brown University computer system. Knowledge of the language as described in the reference manual [4] is assumed. For those using ALGOL W on BUGS, a knowledge of GMS, the Meta4A instruction set, and the Meta4A assembler is also assumed.

The notation used in this manual to specify command and macro formats is consistent with IBM standards. Lowercase symbols denote parameters to be specified by the user; uppercase symbols denote keywords; brackets, [ ], indicate a choice of one or none of the enclosed items; braces, {}, indicate that exactly one of the enclosed items should be specified. Default options are underlined.

1 COMPILATION AND EXECUTION

The ALGOL W compiler is supported under OS/PCP and CP/CMS on the 360/67 at Brown University. This section describes the format of card decks and the procedures to compile and execute ALGOL W programs. This information is contained in more detail in [4].

1.1 DECK SETUP

The deck (or source file) setup for an ALGOL W compilation is as follows:

(1) an optional %ALGOL card, see section 1.2.

(2) the source program, interspersed with option cards. See section 1.3.

(3) a %OBJECT card, followed by ALGOL W procedure text decks. This step is optional, but if included then $DEBUG,0 must also be specified.

(4) a %DATA card, followed by data cards. This step is optional.

Steps 1 thru 4 may be repeated, but the second and subsequent %ALGOL cards are required.

1.2 %ALGOL CARDS

%ALGOL cards delimit programs in a batch submission, and allow the default compilation/execution options to be specified by the user. The format of an %ALGOL card is:

%ALGOL [<options>]

The options may be specified in any order, separated by commas. Available options (with abbreviations and defaults underlined) are:

TIME=sss — specifies the execution time limit (which does not include compilation or post-processing time). If this limit is exceeded, an error message is printed giving the coordinate of the statement executing at the time, and execution is terminated with a post-mortem dump. The

default time limit is ten seconds. This option is not supported under CMS or on the Meta4A.

PAGES=ppp — specifies the execution page limit (which does not include compilation or post-processing printed output). If this limit is exceeded, execution is terminated in the same manner as when the time limit is exceeded. The default value is nine pages, 60 lines per page. This option is not supported on the Meta4A.

MARGIN={72|80} — specifies the last column of data cards that READ and READON should scan. The default is MARGIN=80; however, if the program source cards are sequence numbered, it is assumed that the data cards are also sequence numbered and that MARGIN=72.

SIZE=xxxK — limits the maximum amount of dynamic storage requested by the compiler or runtime library to xxx*1024 bytes. This option should only be used in rare cases to prevent the compiler from using all of the available core.

1.3 OPTION CARDS

Any of the following cards can appear between a %ALGOL card and the next %-card:

Option	Effect
$NOLIST	Do not list subsequent source cards.
$LIST	List subsequent source cards. This is the default.
$TITLE,"<text>"	Eject the page and use "<text>" (up to 45 characters) as a title.
$SYNTAX	Analyze the program for syntax errors, but do not execute.
$STACK	Dump the current parsing stack if a pass 2 error occurs, with the most recent syntactic element listed last.
$DUMP*ab,cc	Dump certain internal tables during the compilation.

$NOCHECK	Omit the checking of subscript ranges, case-index ranges, and reference compatability, and omit the initializaton of variables to "undefined" (X'FB').
$DEBUG,n(m)	Activate the tracing, statement counting, and post-mortem dumping facilities of the ALGOL W system.
$NORM,a,b	Activate the floating point significance tracing facilities of ALGOL W. This facility is only available on the /360. See [4] for more details.

1.4 DEBUG OPTIONS

In the $DEBUG,n(m) option, the meaning of the single digit n is as follows:

Value of n	Meaning
0	Print the coordinate number of the statement in error if execution terminates abnormally.
1	The above plus a post-mortem dump of all the program's variables.
2	The above plus counts of how often each statement was executed.
3	The above plus a statement by statement trace of each value stored.
4	The above plus a trace of each value fetched.

If tracing is specified, i.e., n > 2, and the standard procedure TRACE is not used, then each ALGOL W statement will be traced in symbolic form the first m times it is executed. $DEBUG,4(2) may be abbreviated "$DEBUG", and $DEBUG,n(2) may be abbreviated "$DEBUG,n". The default is $DEBUG,1. See [4] for a detailed explanation of the debugging facilities and how to interpret debugging output.

1.5 DUMP OPTIONS

The $DUMP option specifies what information is to be dumped and which segments in the program will be dumped. The format is:

```
$DUMP*ab[ ,cc]
```

where:

(1) a is a single digit and is ignored.

(2) b is a single digit signifying which combination of the 5 tables given below is to be dumped.

(3) cc is a number in the range of 00 to 63; exactly 2 digits must be specified. If cc is not specified, then tables for all segments will be dumped. If cc is specified, then the object code for only that segment will be dumped. Multiple dump cards may be used to specify more than one segment. If the b digits are different, the last one is used.

Value of b	Parse Tree	Symbol Table	Edit Code	Intermediate Object Code	Final Object Code
0	no	no	no	no	no
1	no	no	no	no	yes
2	no	no	no	yes	yes
3	no	yes	no	no	no
4	no	yes	no	no	yes
5	no	yes	no	yes	yes
6	yes	yes	no	no	no
7	yes	yes	yes	no	yes
8	yes	yes	yes	yes	yes
9	yes	yes	yes	no	yes

Note that b = 9 has the same effect as b = 7.

1.6 COMPILING AND EXECUTING UNDER OS/360

Under OS/360, there are actually two versions of the ALGOL W compiler; both versions use exactly the same code for the various phases of the compiler and the runtime library, but the monitor phase is slightly different. The compile, load, and go incore version is called ALGOLW; it supports all debugging features. The compile only version is called ALGOLY; it produces standard /360 text decks, but cannot pass any debugging information, so $DEBUG,0 is forced. The output from ALGOLY can be link-edited with other text decks or load modules, including those produced by FORTRAN.

Before execution the text decks from ALGOLY must be link-edited or loaded with the ALGOL library and runtime monitor ALGOLX. To facilitate this, all object decks for ALGOL W mainline programs contain external references to the monitor and the library.

The restricted object deck facility for the compile, load, and go version only handles text decks of ALGOL W procedures, compiled without debugging features (i.e., $DEBUG,0).

If a procedure declaration is compiled and a //SYSPUNCH DD card is supplied, then a /360 text deck for that procedure will be produced. This deck can then be used with the link-editor or loader as described above, or it can be read back into the compile, load, and go incore version when the main program is compiled. The text decks are placed after the %OBJECT card.

Four catalogued procedures are provided: ALGOLW (compile, load, and go incore), ALGOLCG (compile and go), ALGOLC (compile only), and ALGOLG (load only). In the latter three cases, the text decks are passed in the same way that FORTRAN text decks are passed; thus ALGOLC and FORTHC can be intermixed and followed by ALGOLG. The step names are COMP and GO. Parameters given on an %ALGOL card are not passed to the GO step; the EXEC card parameter field is instead decoded in the same way. An example of an EXEC card is:

```
//STEPA EXEC ALGOLCG,PARM.GO='MAP,EP=ALGOLX/T=3,P=15'
```

1.7 COMPILING AND EXECUTING UNDER CMS

The format of the CMS command to invoke the ALGOL W compiler is:

ALGOLW <filename> [(<options>]

where <filename> is the name of a file of filetype ALGOL, and <options> is one or more of the compiler options listed below. The minimum abbreviation of the default is underlined.

Option	Meaning
ATR|NOATR	ATR causes the compiler to produce an attribute listing.
DECK|NODECK	DECK causes a TEXT deck to be produced.
DIAG|NODIAG	DIAG causes compilation diagnostics to be typed online.
LIST|NOLIST	LIST causes a compilation listing to be generated. The listing is written to disk with a filetype of LISTING unless the PRINT option is specified.
PRINT|NOPRINT	PRINT causes the compilation listing to be written to the printer rather than to disk.
SEQ|NOSEQ	SEQ causes the compiler to check the sequence field (columns 73-80) of input records for ascending order, printing a warning message when sequence is out of order.
XEQ|NOXEQ	XEQ causes the object code produced by the compiler to be loaded directly into core and executed.
XREF|NOXREF	XREF causes the compiler to produce a cross-reference listing.
M4A	M4A causes a TEXT deck for the Meta4A to be produced, and forces $DEBUG,0. If M4A is omitted, /360 code will be produced.

OCCAM|NOOCCAM[1] — OCCAM causes the OCCAM utility to be invoked if the DECK option was specified or implied (by M4A).

ICP|NOICP — ICP causes the OCCAM utility to be invoked with the ICP option if the DECK option was specified or implied.

NS|NONS — NS causes the SPIE normally issued by ALGOL W to be suppressed.

ALLINT|NOALLINT — ALLINT causes a PSW and register dump to be printed when any program interrupt occurs.

The source file must reside on the P-disk or on a read-only extension, and must be of filetype ALGOL. If the DECK option is specified, the file "<filename> TEXT" is created containing the object deck. If the PRINT option is not specified, the file "<filename> LISTING" is create; it contains the compilation listing (and the map produced by OCCAM if it was invoked). The /360 deck produced by means of the DECK option is identical to that produced by the OS/360 version. Note that the Meta4A version of the compiler is supported only under CMS.

During program execution under ALGOLW (with the XEQ option), the files SYSIN, SYSPRINT, and SYSPUNCH are normally FILEDEFed (with the NOCHANGE option) to <fn> ALGOL, <fn> LISTING, and DUMMY respectively; however, the user may override them by explicitly issuing FILEDEF's. The ALGOL W runtime library routines are contained in ALGOLW TXTLIB, which should be globaled before loading the user program(s).

To load and execute previously compiled ALGOL W programs, issue:

```
GLOBAL T ALGOLW [<other TXTLIB's>]
LOAD <filename> (NOMAP
START ALGOLX [<options>]
```

The options that may be specified are NS and A. Note that programs loaded and executed in this manner may call subroutines written in other languages, e.g., FORTRAN and Assembler. During program execution under ALGOLX, the files SYSIN, SYSPRINT, and SYSPUNCH are normally FILEDEFed to CON, CON, and DUMMY, respectively.

The %ALGOL TIME parameter is not supported under CMS. As an alternative, the CP EXTERNAL command will cause program termination as if time had expired.

---

[1] Both the OCCAM and ICP options cause the compiler to return unique codes to CMS which are then used by the ALGOLW EXEC on the COMMON disk to invoke OCCAM. See Section 1.10 for details.

1.8 ASSEMBLING PROGRAMS TO INTERFACE WITH ALGOL W

Assembler subroutines may be interfaced with ALGOL W programs using either FORTRAN or ALGOL W linkage conventions. Details on how to interface assembler programs with ALGOL W will be discussed in Section 4.

To assemble (or compile) ALGOL W programs for the Meta4A, the Computer Science COMMON disk must be logged in (at device address 199 F,P). The minidisk contains EXEC's for compiling, assembling, and creating MODU's (executable modules for BUGS). It also contains two sets of libraries: AWXALL, AWX360, and AWXM4A MACLIB's (macros to interface assembler routines with ALGOL W) and AWXM4A TXTLIB (Meta4A runtime subroutines).

To assemble programs for the /360, issue:

```
GLOBAL M AWXALL AWX360 SYSLIB [<other MACLIB's>]
ASMG <filename> [ (<options>]
```

To assemble programs for the Meta4A, issue:

```
BUGSGBL A AWXALL AWXM4A [<other MACLIB's>]
BUGSASM AL <filename> [ (<ASMG options>]
```

The BUGSGBL command globals the macro libraries used by the Meta4A assembler, and up to four additional ones. Text decks produced by BUGSASM may be treated as ALGOL W text decks and processed as described in section 1.9.

1.9 COMPILING/LINK-EDITING PROGRAMS FOR THE META4A

To prepare programs for the Meta4A, all procedures must be compiled and/or assembled, and then link-edited. This section describes how to perform each of these steps.

To compile an ALGOL W mainline and/or procedure(s), type:

```
ALGOLW <filename> (M4A [<other options>]
```

This command invokes the ALGOL W compiler to produce a file "<filename> TEXT".

All external procedures must be explicitly declared in ALGOL W programs. The format of the procedure declaration is:

```
[type] PROCEDURE name [ (<parameter list>) ] ;
    {ALGOL|FORTRAN} "<string>";
```

where <string> is the one to eight character ESD name of the CSECT to be called. For FORTRAN procedures, <string> may be any valid FORTRAN name. For ALGOL W procedures, <string> may

be of two forms: if the procedure was compiled by the ALGOL W compiler then <string> is of the form "xxxxx001", where "xxxxx" are the first five letters of the procedure name extended if necessary with #'s; if the procedure was assembled (using the AWX-macros) then <string> is any valid ESD name.

To link-edit a mainline and/or external procedure(s) and create an executable module (MODU) for BUGS, issue:

```
ALGLINK <moduname> <name1> ... <name8>
```

where <moduname> is the name given to the MODU and MAP files created by GMSLINK and each name(i) is a file of filetype SYSLIN, TEXT, TXTLIB or MODU. If TXTLIB's are specified, they must appear last in the list of names. Usually the symbol AWXERROR (and AWXSL006, if the TIME function is used) will be unresolved until the MODU is loaded on BUGS.

1.10 OCCAM: HANDLING ALGOL W TEXT DECKS

The text decks produced by the ALGOL W compiler are quite large and complex. For each program segment (i.e., procedure or begin block with declares), ESD, END, and RLD cards are generated (in addition, of course, to the TXT cards). These occupy extra online storage space and, unfortunately, make the inclusion of ALGCL W compiled text decks in TXTLIB's virtually impossible. Since each program segment is given a standard name, separately compiled procedures may contain identical ESD names, making it impossible to include them in one TXTLIB. Moreover, updating a TXTLIB for one ALGOL W source procedure would entail updating each program segment separately.

To eliminate these problems, a utility program, OCCAM[2], has been written by Russell Burns. Basically, it takes a TEXT file produced by the ALGOL W compiler that contains extra ESD, END and RLD cards and merges them into one text deck at a space savings of up to fifty percent. OCCAM performs the following functions: resolves all cross-references (external) between program segments in one compilation; consolidates all external references (and thus eliminates duplicate ESD entries); consolidates all RLD entries; and consolidates all TXT cards, creating a single CSECT. In addition, duplicate record table entries (which are generated for RECORD datatypes to facilitate their allocation/garbage collection) are eliminated; thus, if multiple external procedures are contained in one compilation, the user must be sure that the first procedure (in the lexical sense) declares every record type which the other procedures might use.

---

[2]Occam's Razor: to take the simpler of two hypotheses

OCCAM is invoked in CMS by specifying the OCCAM option to the ALGOLW compiler or by issuing the command:

OCCAM <textname> (ICP

where <textname> is the name of a file of filetype TEXT produced by the ALGOL W compiler. The (ICP option causes the ESD name of a compiled procedure to be changed from the standard format (xxxxx001) to the actual name of the procedure (for use by ICOPS). TEXT files from many separate ALGOL W compilations may be combined (using CMS utilities) and processed by OCCAM. Note that a text file produced by the assembler should not be processed by OCCAM; if such a file is used as input to OCCAM, an error will be indicated and the file will remain unchanged.

Output from OCCAM is the file "<textname> TEXT", which replaces the input file; it contains the input file "compressed" as previously described. In addition, a file "<textname> MAP" is produced which may be OFFLINE PRINTCCed. It contains two statement maps of the ALGOL W program(s) whose TEXT file was processed. Each map lists for each source statement (or coordinate, in ALGOL W terminology) the offset, from the start of the first program segment in the compilation, of the first instruction generated for that statement; one map is sorted by statement numbers, the other by statement addresses. If TEXT files from separate compilations are combined and processed, the statement maps will contain duplicate statement numbers, corresponding to the order in which the compiled TEXT files were combined.

The statement map is essential for debugging compiler-generated programs which have been processed by OCCAM (particularly those compiled for the Meta4A, since the /360 debugging features are not supported on the Meta4A). In order to set breakpoints and display compiler-generated code and data, one must specify addresses relative to the first program segment rather than relative to each program segment, since OCCAM combines all program segments. The statement map is used to determine the offset from the first compiled program segment where a breakpoint or display point is desired; then the desired operation (setting a breakpoint, displaying core) is performed using an interactive debugger (LSDBUG, NEWBUG, or FUDD). In addition, when a runtime error occurs which is <u>not</u> intercepted by the runtime error handler (such as when the NS option is specified on either the /360 or Meta4A), then only the interrupt address will be available; the actual statement in which the error occured can only be determined from the statement map after calculating the offset of the interrupt address from the load point of the first program segment.

2 ALGOL W ON THE META4A

ALGOL W was obtained from Stanford University in the fall of 1975 to be used as a tool for graphics and distributed processing (ICOPS) research. By the spring of 1976, the compiler had been modified to generate code for the Meta4A satellite processor, and the runtime package had been rewritten for that machine. This section describes the Meta4A implementation of ALGOL W.

2.1 PROGRAM EXECUTION

The current version of the ALGOL W runtime monitor exists on BUGS disk CS276. To run an ALGOL W program, type:

ALGOL <moduname> [(<options>]

where <moduname> is the name of the MODU file created by the ALGOL or ALGLINK EXEC's. Available execution options are:

NS allows FUDD to be used for debugging (similar to the NOSPIE option on the /360) by causing all interrupts to be trapped by FUDD instead of the runtime monitor, and by allowing breakpoints/checkpoints to be interpreted as such rather than as invalid op-codes.

DEBUG is a synonym for NS.

OUTF directs program controlled output to disk file "<moduname> OUTF" (however, runtime error messages and the execution time message still appear on the console rather than in this file). This file is a listing file, and may be shipped to the /360 and printed offline using the OFFLINE PRINTCC command.

SF=xx specifies the amount of runtime storage (in units of 1K bytes) to be allocated in the stack frame during ALGOL W program execution. The default is 4K. If records or recursion are used, or if runtime error 5005 or 5007 occurs, specify a larger size. The maximum size is 56 - <programsize>.

The ALGOL command invokes the runtime monitor, which establishes the runtime environment and loads and executes the user program. Execution continues until normal completion, a fatal error (see Error Messages, section 2.4), or the INTERRUPT key on the Meta4A programmer's panel is hit. In the latter case, if the NS option was specified, FUDD is entered, and typing 'GO' will resume normal execution; otherwise, a time limit exceeded error will occur, terminating execution.

2.2 INPUT/OUTPUT

The Input/Output facilities were designed to be compatible with the /360, given the limitations of CMS.

Stream input to a program comes initially from the console unless a file "<moduname> DATA" exists on the BUGS disk, in which case input initiates from this file. Subsequently, input can be routed from any file by use of the &INCLUDE facility; when the line

&INCLUDE <filename> <filetype>

(where "&INCLUDE" begins in the first column of the record) is read from the console or from a disk file, input from the current source is suspended and subsequent input comes from the file specified in the &INCLUDE line; if the &INCLUDE'd file is exhausted before the program terminates (or if the file is not found), the original source is returned to for input. A file may be &INCLUDE'd from within another file, as long as no more than five files are open for stream input at any one time; if five files are already open, an &INCLUDE will be ignored (except for an error message to the console). Note, however, that a file is closed immediately upon reading the last record in it. Therefore, if the last record of a file "A" were to &INCLUDE a file "B", the number of open files would not change since "A" would first be closed, and then "B" would be opened. By &INCLUDEing it with the <u>last</u> record in "A", "B" becomes a logical extension of "A". Any number of files can be linked in this manner.

Input files may be of any record length up to eighty bytes. If less than eighty bytes long, each record is padded with blanks; if greater, each record is truncated. Upon reading the last record in a file, input is routed back to the console or the file from which the exhausted file was &INCLUDE'd. Note that an end-of-file condition is never reached: if a null line is read (either from the console or a disk file) another read is issued in order to satisfy the input request. The user should use a unique data item to flag the end of the input data.

All output is normally printed at the console (long lines are folded). If the OUTF option is specified, all program controlled output will be written to the disk file "<moduname> OUTF". (Runtime error messages and the execution time message still go to the console rather than the disk file). If the disk file overflows due to a full or fragmented disk, output is routed back to the console. The OUTF file is equivalent to a CMS LISTING file, and may be sent to the /360 and printed with the OFFLINE PRINTCC command.

Note that ALGOL W assembles output lines in a buffer that is not flushed until a new "WRITE" forces the previous line out -- thus, output to the console is always one line "behind"

the program. If the user expects to see the output of a WRITE statement immediately after it is executed (e.g., if subsequent input is to be conditioned on messages to the console), he should follow each WRITE with an IOCONTROL(2) to force immediate flushing of the buffer.

2.3 DEBUGGING

The process of debugging an ALGOL W program on BUGS is similar to debugging any program written for the Meta4A. A knowledge of FUDD, the symbolic debugger, is assumed. To debug an ALGOL program, issue the sequence:

```
*FUDD
-TEST moduname
    .
    .
    set breakpoints/checkpoints; note that
    the name of an ALGOL mainline is AWXSC001
    .
    .
-GO
*ALGOL moduname (NS
```

Debugging of ALGOL W code on the Meta4A should not be necessary; however, if it is, the $DUMP*01[,cc] option may be specified in the compilation to obtain a listing of the generated code. If there are problems with the runtime environment (I/O, procedure linkage, runtime subroutines, etc.), document the error and contact a member of the BUGS or ALGOL W group.

2.4 RUNTIME ERROR MESSAGES

Runtime error messages are of the form:

RUN ERROR xxxx [(yyyy)] NEAR COORDINATE zzzz IN procname

where xxxx is the error code; yyyy is an additional error code for certain error messages, as described below; zzzz is the coordinate number near where the error occurred; and procname is the name of the procedure or block in which the error occurred. Error codes greater than 5100 indicate runtime subroutine errors (non-fatal on the Meta4A, fatal on the /360), while those in the range 5000-5099 indicate fatal execution errors. In the latter case, FUDD will be entered. The error codes are as follows:

5001 substring out of range

5002 index in a case statement or expression is less than one or greater than the number of cases
5003 array subscript out of bounds
5004 attempt to assign to a name parameter whose actual argument is not a variable; yyyy is the parameter number in error
5005 data area overflow; caused by excessive recursion or not enough free memory; try specifying a larger value for SF=xx
5006 actual parameter passed is not assignment compatible with the formal parameter; yyyy is the parameter number in error
5007 no more storage exists for records; try specifying a larger value for SF=xx
5008 string read on input longer that variable length
5009 value read for a LOGICAL variable was not TRUE or FALSE
5010 the number read was not assignment compatible with the variable in the READON or READ statement
5011 REFERENCE values cannot be read
5012 reader EOF (cannot occur on BUGS)
5013 the number of actual parameters in a procedure call differs from the number of formal parameters declared
5014 array is too large; see [1]
5015 a null string or a string longer than 255 characters was read
5016 attempt to access a record field using a NULL or undefined REFERENCE
5017 page estimate exceeded (cannot occur on BUGS)
5018 time estimate exceeded; occurs if the INTERRUPT key on the Meta4A panel is hit and the NS or DEBUG option was not specified
5019 runtime monitor or library error
5020 program interrupt; yyyy is the GMS interrupt code in hexadecimal, as follows:

Meta4A D002 operation exception
D004 stack frame overflow
D006 stack frame underflow
D008 integer overflow
D00A conversion error
D00C integer divide by zero
D00E alignment
D010 register specification
D012 priviledge
D014 stack overflow (PSH instructions)
D016 stack underflow (POP instructions)
D018 execute exception
D01A invalid UCB addr
D01C invalid unit addr
D020 no free memory
D022 invalid free
D024 invalid CPC
D026 invalid /360 UCB addr
D030 exponent overflow

	D032	exponent underflow
	D034	floating point divide by zero
	D036	timeout (floating point hardware error)
	D060	addressing
	D062	protection
Meta4B	1040	operation
	1041	alignment
	1042	register specification
	1043	divide by zero
	1044	sqrt operand negative
Simale	1080	Simale error

5021 attempt to access a field of a record, but the REFERENCE is not defined to designate a record of the corresponding class; may occur if the REFERENCE is null or undefined

5022 assertion yyyy failed

5101 division by zero or 0**(-n); (result set to zero on Meta4A)

5102 SQRT of a negative number; (absolute value of argument is used on Meta4A)

5103 argument to EXP function must be less than 174.67; (result set to MAXREAL on Meta4A)

5104 LN/LOG of a negative or zero number; (absolute value of argument is used on Meta4A)

5105 SIN/COS function domain error; (result set to zero on Meta4A)

2.5 IMPLEMENTATION RESTRICTIONS

- Because there is only short (32 bit) precision floating point on the Meta4A, the LONG versions of standard functions produce the same results as the SHORT versions.

- The predeclared variable MAXINTEGER is equal to 32767 on the Meta4A, and 16,777,215 on the /360.

- None of the debugging features are supported on the Meta4A (primarily because the symbol table cannot be saved externally). The TRACE procedure has no effect (and $DEBUG,0 is forced for a Meta4A compile, anyway).

- Exceptional conditions on the Meta4A are handled entirely by the runtime monitor and operating system. Exception records (INTREF's) are not supported; using them may produce weird side-effects.

3 ALGOL INTERNALS

3.1 INTERNAL DATA REPRESENTATION

The internal data formats on the Meta4A are similar to that on the /360. All datatypes are stored in the same amount of space which they occupy on the /360. However, since the Meta4A is a halfword machine, it was decided to ignore high-order halfwords for all but REAL, LOGICAL, and STRING datatypes. The following chart summarizes storage and access conventions on both machines:

datatype	stored in	Meta4A accesses[3]
integer	fullword	2nd hword
real	fullword	fullword
complex	2 fullwords	2 fullwords
bits	fullword	2nd hword
reference	fullword	2nd hword
logical	byte	byte (X'00' or X'01')
string(n)	n bytes	n bytes

Long datatypes are stored in doublewords, but floating-point operations on the Meta4A use only the high-order fullword (single precision). Arrays are stored in column-major order (i.e., the same as FORTRAN arrays; the first subscript varies first) in contiguous memory locations, following the above storage and alignment rules for each datatype. Array descriptors (dope vectors) are created by the compiler and allocated in each procedure's automatic storage to access individual elements of arrays and for parameter passing (see section 4.4.7 for the format of array descriptors).

Up to fifteen record types (structures) may be defined in ALGOL W. The fields in records are allocated in a compiler-defined order for alignment and to facilitate garbage collection. The order is: 1) reference fields; 2) integer, real, complex, and bitstring fields; 3) string and logical fields; 4) a mandatory 0-7 bytes of padding for doubleword alignment; and 5) long real and long complex fields. If more than one field of a given alignment level is in a record, fields are taken from left to right in the order declared.

Record allocation is managed entirely by runtime library subroutines. Records are implicitly allocated in fixed-length doubleword-aligned pages; when no free records exist, those

---

[3]Warning: the first halfword of integer, bits and reference datatypes may be overwritten on assignment to them as RESULT or VALUE RESULT parameters.

not currently pointed at by REFERENCE variables are collected for reuse by the garbage collector. Note that the first fullword in a record is reserved for runtime record management. On the Meta4A the high order byte of this fullword contains the record type (X'01' thru X'0F') and the second halfword contains the allocation number of that record type (for debugging). On the /360, the record type is always stored in the high order byte of REFERENCE variables and is usually stored in the first byte of a record, while the second halfword of a record contains the record number.

3.2 META4A CROSS-COMPILER CONSIDERATIONS

The changes made to the ALGOL W compiler and runtime environment to convert from the /360 version to the BUGS version are complicated by the differences in architecture and register usage, since the Meta4A is a halfword machine with an extended /360 instruction set and a more functionally restrictive register set. The problem of data structures was solved by keeping the same storage format on the Meta4A as on the /360; this was discussed in section 3.1.

The primary architectural differences between the /360 and the Meta4A are: (1) the Meta4A has a register length of 16 bits, not 32 bits as on the /360; (2) the non-RR Meta4A branch instructions branch relative to the program counter, rather than to an absolute address; (3) instructions that set the condition code on the /360 do not set the condition code in the same manner on the Meta4A; (4) the condition code value for a given situation on the Meta4A is different than that on the /360.

Register allocation on the Meta4A presents problems in that registers R0, R1, and R15 have special meanings: the machine status register, the program counter, and the stack frame pointer, respectively. In addition, the Meta4A CLC instruction alters R2. These problems were alleviated both by altering the register allocators of the compiler to avoid certain registers, and by mapping the /360 registers into Meta4A registers as described below:

360 Register	Meta4A Register	"Q-reg"
R0	R4 or R0	Q0
R1 - R5	R5 - R9	Q1 - Q5
R6 - R7	void	Q6 - Q7
R8 - R12	R10 - R14	Q8 - Q12
R13	R15	Q13
R14	R3	Q14
R15	void	Q15
void	R2	QX

The register corresponding to /360 register Rn on the Meta4A is referred to as Qn (or a "Q-reg" to those in the know), e.g., Q1 is R5. See [5] for an explanation of how the mapping was derived.

3.3 RUNTIME ENVIRONMENT

ALGOL W requires a non-trivial runtime environment because of its block structure and stringent type-checking. Library routines are called to perform: I/O; the building of parameter lists and type-checking of parameters; the built-in arithmetic functions; error checking of array bounds, case indices, references to uninitialized variables (unless the $NOCHECK option is specified); and automatic storage allocation for local variables and parameters upon each procedure's invocation.

Data segments (parameters, declared variables) and program segments (parameter descriptors, branch table, literals, executable code) are stored separately. Each data segment is stored in a fixed compiler-defined format; only those maintaining the compiler and runtime support need be concerned with the precise layout. Each program segment is also stored in a fixed compiler-defined format; for assembly language programmers, the information stored in these segments may be useful. The format of a program segment is as follows:

offset	contents
000	branch to procedure entry code
004	F'-1' or a branch instruction (/360 only)
008	AL4(entry point)
00C	X'00001000'
010	offset to CL8'<procedure name>'
012	offset to end of coordinate table
014	AL4(0)
018	current CLN (data segment base) register
019	program segment base register
01A	offset to end of auto storage
01C	procedure type, see chart below (byte)
01D	<length>-1 of function value (byte)
01E	number of parameters (halfword)
020	parameter descriptor(s) (fullword)
bbb	branch table (a fullword for each source label)
lll	literal table
ppp	procedure entry, body, and exit code
ccc	CL8'<procedure name>' and coordinate table

For each parameter, a fullword descriptor appears (starting at offset X'020'); no descriptor fullwords are allocated if there are no parameters. The format of a parameter descriptor is:

byte 0 B'000000rv', parameter type: r=1 if result, v=1 if value (any combinations are allowed)
byte 1 number of dimensions, zero if scalar
byte 2 <length>-1 of string parameters
byte 3 parameter type (see chart below)

The procedure and parameter types are:

0	untyped (proper or formal procedures only)
1	integer
2	real
3	long real
4	complex
5	long complex
6	logical
7	string
8	bits
9	reference
16+<type>	function procedure

Procedures passed as parameters have the same format as name parameters (type 0). For more details on the internals of procedures and parameters, see [3].

Registers are allocated at runtime as follows:

360 Register	Meta4A Register	Function
R0 - R1	R4 - R5	system linkage, scratch (pair)
R2	R6	expressions
R3 - R4	R7 - R8	thunks and expressions
R5	R9	procedure entry, expressions
R6 - R7	void	auto storage base, expressions
R8 - R12	R10 - R14	auto storage base, expressions
R13	SFP	common data area address
R14	R3	current program segment base address
R15	void	procedure calls, extra program base
F0 - F6	F0 - F6	expressions

To generalize this discussion to both machines, the Q-reg notation of section 3.2 will be used. ALGOL W is a block-structured language; to allow blocks to access variables allocated in enclosing blocks, registers Q13 down to Q5 on the /360, and Q13 down to Q8 on the Meta4A, may be allocated as data segment (auto storage) base registers. This imposes a limit (of eight levels on the /360 and five levels on the Meta4A) on the static (i.e., source-level) nesting of procedures and blocks that contain declarations. If cln is the current static nesting level, then Q13 to Q(13-cln) are used as data segment base registers, and R2 (on both the /360 and Meta4A) up to Q(13-cln-1) are used for computation.

Note that Q13, which corresponds to nesting level zero (the standard ALGOL W environment), and Q12, which corresponds to nesting level one (the mainline data segment), are always constant. The standard ALGOL W environment is allocated by the monitor prior to execution of the user program. It contains information used by the runtime monitor and library subroutines, such as buffers, flags, constants, and record pages. On the /360 this area is GETMAINed in free core; on the Meta4a it is DC'd into the monitor, and its address is loaded into Q13 (R15) so as to make the area look like an entry in the stack frame.

3.4 STANDARD FUNCTIONS

The ALGOL W runtime library routines (standard functions) for the /360 reside in ALGOLW TXTLIB, and those for the Meta4A reside in AWXM4A TXTLIB. On the /360, these routines must execute in the ALGOL W environment because they use constants stored in the common data area and may call AWXERROR for domain errors. However, on the Meta4A, the standard functions have been written to execute outside of the ALGOL W environment. They may generate calls to AWXERROR (which must execute in the ALGOL W environment), but a dummy routine can be used to "trap" such calls and allow the functions to be used by any assembly language programmer.

The standard functions of analysis are invoked by a standard calling sequence for each machine, as follows:

/360		Meta4A	
LE	F0,arg	LE	F0,arg
L	R15,=V(entry)	LI	Q1,V(entry)
BALR	R1,R15	BALR	Q1,Q1
DC	H'<code>'		

The result of each function replaces the argument in F0. The functions alter the contents of floating point registers F0, F2 and F4; the (LONG) ARCTAN and (LONG) EXP functions alter F6 in addition. In other words, they do not save the previous contents of the floating-point registers.

The names and locations of the functions are as follows:

function	/360 entry pt	/360 code	Meta4A text deck	Meta4A entry pt
SQRT	AWXSL012	0001	AWXSQRT	AWXSL501
LONGSQRT	AWXSL013	0001	AWXSQRT	AWXSL401
SIN	AWXSL012	0005	AWXTRIG	AWXSL405
LONGSIN	AWXSL013	0005	AWXTRIG	AWXSL505
COS	AWXSL012	0006	AWXTRIG	AWXSL406
LONGCOS	AWXSL013	0006	AWXTRIG	AWXSL506
ARCTAN	AWXSL012	0007	AWXATAN	AWXSL407
LONGARCTAN	AWXSL013	0007	AWXATAN	AWXSL507
LN	AWXSL012	0003	AWXLOGLN	AWXSL403
LONGLN	AWXSL013	0003	AWXLOGLN	AWXSL503
LOG	AWXSL012	0004	AWXLOGLN	AWXSL404
LONGLOG	AWXSL013	0004	AWXLOGLN	AWXSL504
LOG2	unimplemented[4]		AWXLOGLN	AWXSL413
EXP	AWXSL012	0002	AWXEXPON	AWXSL402
LONGEXP	AWXSL013	0002	AWXEXPON	AWXSL502
complex *,/,abs,**	AWXSL014	xxxx	AWXCMPLX	AWXLS6xx

Assembly language programmers may want to call the runtime error handler, AWXERROR, directly. The standard calling sequence is:

/360		Meta4A	
LA	R0,<error code>	LI	Q0,<error code>
L	R15,=V(AWXERROR)	LI	Q2,V(AWXERROR)
BALR	R2,R15	BALR	Q2,Q2

The possible error codes contained in Q0 are listed in [4] and section 2.4; the absolute value of the code is 5000 less than the printed error message. If the code is negative (a standard function error), AWXERROR will return to the caller after attempting a fixup; if the code is positive, an error message will be printed and execution will terminate; a zero error code is not used for a direct call of AWXERROR (and will produce unpredictable results). The error handler assumes upon entry that: Q0 contains the error code; Q1 contains the object code return address (i.e., the return address from a standard function); Q2 contains the address AWXERROR should return to; Q13 contains the address of the runtime data area; and Q14 contains the current program segment base address.

---

[4] Added to the Meta4A subroutine library for convenience; not a standard function.

4 INTERFACING ALGOL W WITH ASSEMBLER

4.1 EXTERNAL PROCEDURES

In a program which calls an external procedure, a dummy procedure declaration and body are used to establish the proper correspondence to the external procedure name, as follows:

```
[type] PROCEDURE name [(<parameter list>)] ;
    {ALGOL|FORTRAN} "<string>";
```

The symbols ALGOL and FORTRAN specify what kind of linkage conventions will be used by the compiler in generating a call to the procedure; the <string> following the symbol specifies the ESD name of the external procedure. Note that it is not necessary to use ALGOL linkage conventions in assembler routines unless the user intends to either

(1) Call an ALGOL W procedure from the assembler routine, or

(2) Use ALGOL's storage management facilities (i.e., the AWXALLO and AWXFREE macros described below).

4.2 FORTRAN LINKAGE CONVENTIONS

The symbol FORTRAN indicates the use of standard operating system linkage conventions on the /360 or the Meta4A. The string following the symbol is extended with blanks or truncated to eight characters and is used as the ESD name of the external procedure. The following formal parameter types are allowed and are interpreted as follows:

(1) <simple type>: The corresponding actual parameter is examined. If that parameter is a variable, the address of the variable is computed once and transmitted. Otherwise, the expression which is the actual parameter is evaluated, the value is assigned to an anonymous local variable, and the address of that variable is transmitted.

(2) <simple type> {value|result|value result}: As in ALGOL W procedures, a local variable unique to the call is created, and the address of that variable is transmitted.

(3) <simple type> *array*: The address of the actual array element with unit indices in each subscript position is computed and transmitted, even if that element lies outside the declared bounds of the ALGOL W array. Arrays with only one dimension and arrays with unit lower subscript bounds will have elements with indices which are identical in ALGOL W and FORTRAN routines. Array cross-sections should not be used as actual parameters to FORTRAN subprograms, since the array descriptor will not be handled properly.

4.2.1 ON THE /360

FORTRAN linkage conventions are defined to be standard OS/360 linkage conventions. Caller's registers should be saved in the standard OS save area. String functions with FORTRAN linkage conventions are not permitted. If FORTRAN I/O or error handling facilities are used, the subroutine package IBCOM or a suitable substitute is required.

4.2.2 ON THE META4A

FORTRAN linkage conventions on BUGS are defined to be standard GMS conventions. The following calling sequence is generated:

```
              LA     R2,PLIST                  optional
              LR     R5,R14
              LI     R14,V(<procname>)
              BALR   R14,R14
              LR     R14,R5
                     .
                     .
                     .
              DS     0H
     PLIST    DC     AL2(<param>..,)           optional
              DC     X'FFFF'                   optional
```

Items marked optional are included only if the procedure has a formal parameter list; if there are no parameters, R2 will contain garbage. The called routine should use ENT/RET instructions for entry/exit (thus saving the caller's registers automatically). The construction of the parameter list proceeds in the manner described in section 4.1 above. Parameters are stored according to the internal data representations described in section 3.1; on the Meta4A, an offset of two must be added to the parameter addresses of those datatypes whose values are stored in the second half of fullwords.

4.3 ALGOL W LINKAGE CONVENTIONS

The symbol ALGOL in the dummy body indicates the use of ALGOL W linkage conventions. The string following the symbol indicates the ESD name of the main entry point of the procedure and may be of two forms. For an independently <u>compiled</u> ALGOL W procedure, <string> takes the form "xxxxx001", where xxxxx are the first five characters of the procedure identifier extended if necessary with #'s; the name of a mainline is AWXSC001. For an <u>assembled</u> ALGOL W procedure or mainline, <string> is the label appearing on the CSECT or ENTRY DS.

ALGOL W linkage conventions are nonstandard and compiler-dependent. They are basically identical on the /360 and Meta4A, except for minor changes made to the Meta4A version because of architectural differences. Parameter lists are built at run time prior to every procedure call. A parameter list is a list of addresses; each address points to a parameter or a thunk. Due to the complexity of these mechanisms, macros have been written to simplify parameter passing in assembler. In general, these AWX-macros make assembler programs look like ALGOL W procedures (or mainlines); thus, the user can ignore the details of compiler-dependent code.

The assembler programmer must be careful to adhere to register usage conventions. ALGOL W assumes that the following registers will always contain their assigned contents: 1) Q13, the common data area pointer; 2) Q12 and Q11, the auto storage registers for the mainline and first level procedure; and 3) Q14, the program base register. Any other registers needed by an ALGOL W procedure are saved by the caller and restored by the caller upon return from an ALGOL W subroutine. The user need save the above registers only if they are used explicitly in an assembler program. The AWX-macros (AWXLHV, AWXRHV, AWXCALL, and AWXARRAY, in particular) will not change the contents of these registers; however, they may change the contents of any other register, so the programmer must be wary of assuming register contents over a call of one of the macros. <u>To be safe, do not alter</u> R11 - R14 on the /360, or R3 and R13 - SFP on the Meta4A.

4.4 THE AWX-MACROS

The AWX-macros need only be used if ALGOL linkage conventions have been elected by the user. These macros, written by Charles Sorgie, reside in three macro libraries on the Computer Science COMMON disk. They are:

AWXALL   macros common to the /360 and Meta4A interfaces
AWX360   macros to interface /360 assembler with ALGOL W

AWXM4A macros to interface Meta4A assembler with ALGOL W

One assembler "mainline" (in the ALGOL W sense) and any number of assembler procedures (again, in the ALGOL W sense) may appear in an assembly. Nesting of mainlines and procedures is strictly forbidden, and the macros will check for incorrectly sequenced macro calls. The macros automatically generate EQU's for the R-, F-, and Q-registers for the Meta4A assembler; do not include a BUGSREGS macro or the assembler will get extremely confused. Symbols beginning with the characters "AWX" and "&AWX" are used extensively by the macros, so the user is well-advised to avoid these.

The user should not necessarily expect the contents of all registers to remain constant across macro calls: certain macros cream certain registers. The specific register(s) (if any) affected by a macro call are explicitly mentioned in the macro descriptions below.

4.4.1 AWXMAIN

AWXMAIN generates the entry sequence for an ALGOL W mainline. Only one mainline may occur in an assembly. This macro begins the definition of automatic storage (i.e., a DSECT) for the mainline, and _must_ be followed by an AWXENDAU macro. The format is:

```
AWXMAIN    label[,CSECT=YES|NO]
```

label — the actual ESD name of the procedure

CSECT= — CSECT=YES to generate a CSECT, or NO to generate a "DS 0H".

4.4.2 AWXPROC

AWXPROC generates the code sequence to enter a procedure using ALGOL W linkage conventions. It begins the definition of automatic storage for the procedure and _must_ be followed by an AWXENDAU macro for CSECT=YES|NO, but _not_ for CSECT=EXTERNAL. Automatic storage contains parameter descriptors and storage used by the procedure itself. Note that registers are not automatically saved; however, Q12 (the outermost auto storage register) _is_ saved by the accompanying AWXENDAU macro in automatic storage for later use by the AWXCALL and AWXEXIT macros. The name of the savearea is "xxxxx#SA", where "xxxxx" are the first five (or less) letters of the procedure name. The user may place data in automatic storage (by inserting DS's between the AWXPROC and AWXENDAU macros); this is not required, but

will reduce the size of static storage since data will be allocated at runtime instead of assembly time. The macro format is:

```
[proctype] AWXPROC label[,(plist)][,CSECT=YES|NO|EXTERNAL]
```

proctype — an optional ALGOL W procedure type (INTEGER, REAL, LONGREAL, COMPLEX, LONGCOMPLEX, STRING, STRING(n), LOGICAL, BITS, or REFERENCE)

label — the actual ESD name of the procedure

(plist) — a list of keywords enclosed in parentheses specifying an ALGOL W formal parameter list

CSECT= — CSECT=YES to generate a CSECT, CSECT=NO to generate a "DS 0H", or CSECT=EXTERNAL to indicate that the actual body of the procedure is specified in a separately compiled or assembled program (i.e., an ESD name will be generated and later resolved to the external procedure).

The parameter list uses the same basic format as an ALGOL W formal parameter list, except the symbols in the list are separated by commas and the specification of certain keywords is different. The format is:

```
(dtype1[,ptype1],nlist1,...dtypeN[,ptypeN],nlistN)
```

where dtype is: 1) one of the datatypes INTEGER, REAL, LONGREAL, COMPLEX, LONGCOMPLEX, BITS, REFERENCE, LOGICAL, STRING, or STRING(n); 2) a datatype followed by the keyword PROCEDURE, e.g., INTEGERPROCEDURE; or 3) the keyword PROCEDURE; ptype is one of the parameter types VALUE, RESULT, VALUERESULT, or ARRAY, and if not specified defaults to a name parameter; nlist specifies a list (one or several separated by commas) of symbolic parameter "names", the last of which may be the string "(*,...,*)" denoting an array parameter, with one asterisk for each dimension.

Parameter "names" have the form "#xxxxxxx" where "xxxxxxx" is a unique name. The symbolic parameters (#-names's) may be used as arguments to the AWXRHV, AWXLHV and AWXCALL macros. Each name is EQU'd to the ordinal number of the parameter as specified in the formal parameter list; a name may also be specified as simply "#", in which case no EQU will be generated. The symbols which are generated by the macro to define the actual parameters (which are allocated in the procedure's automatic storage) are of the form "xxxxx#n", where "xxxxx" are the first five (or less) letters of the procedure name, and n is the ordinal number of the parameter.

Examples:

```
INTEGER AWXPROC  INDEX,(STRING(256),VALUE,#STR,#PAT)

        AWXPROC  SUM,(REAL,ARRAY,#NUMBER,(*,*),
                 REAL,RESULT,#SUM),CSECT=NO
```

4.4.3 AWXENDAU

AWXENDAU ends the automatic storage DSECT defined in the AWXMAIN and AWXPROC macros. The format is:

```
AWXENDAU
```

4.4.4 AWXCALL

AWXCALL generates the code to call a procedure using ALGOL W linkage conventions. This macro alters registers (see the SA option described below). As part of the calling sequence, Q12 (the outermost auto storage register) is restored if the AWXCALL is not in a mainline. The format is:

```
[label]   AWXCALL {proc|(proc,r)|#name|(#name,r)}
                  [,(plist)][,SA=area]
```

label — an optional label

proc — the ESD name of the procedure being called; this is either the name appearing in quotes in an ALGOL external procedure declaration, or the procedure name in an AWXPROC macro.

(proc,rn) — the name of the procedure being called and the general-purpose register already containing the address of the procedure being called.

#name — the parameter which represents the procedure being called. "#name" must be defined with an AWXPROC macro.

(#name,r) — the parameter being called and the register containing its address. "#name" must be defined with an AWXPROC macro.

(plist) — the actual parameter list consisting of a list of one or more parameter names enclosed in parentheses. The legal parameters are: 1) name; 2) #name as described in section 4.4.2; 3) name(), indicating a parameterless procedure; and 4) name(*), indicating a parameterized procedure. In the later two cases, the user must define

"name" with an EXTRN statement if "name" is external.

SA=area specifies the name of a user-declared save area in which the calling procedure's registers (R4-R15 on both machines) will automatically be saved and restored as part of the calling sequence; the save area should be twelve fullwords long on the /360 and twelve halfwords long on the Meta4A.

In the "#name" and "(#name,r)" cases, the parameter list is not checked at runtime to insure proper correspondence; this is the user's responsibility.

Examples:

```
AWXCALL  SUM,(STRING,PAT)

AWXCALL  SUM,(#VECTOR,#RESULT),SA=MYSAVE
```

4.4.5 AWXEXIT

AWXEXIT generates the code sequence to exit from a procedure using ALGOL W linkage conventions, optionally returning a value to the caller. As part of the exit sequence, Q12 is restored if the EXIT is not in a mainline. Return values are only valid for typed procedures (functions). The format is:

```
[label]   AWXEXIT  [value|(reg)|(*)]
```

label an optional label

value a label or base-displacement address of the value being returned

(reg) a general-purpose register containing the address of the return value or the value itself

(*) indicates that the return value or its address is already in Q3 or F0.

Either the address of the location containing the function value or the value itself is returned to the caller in Q3 (R3 on the /360, R7 on the Meta4A) depending on the type of the value and whether the procedure has parameters. The address is placed in Q3 if there are no parameters or if there are parameters and the type of the returned value is LOGICAL or STRING. Addresses conform to the storage conventions discussed in section 3.1. The value is placed in Q3 if the function has parameters and the type of the returned value is INTEGER, REFERENCE, or BITS; the

value is placed in floating-point register F0 if the type is REAL/LONGREAL, or F0-F2 if COMPLEX/LONGCOMPLEX.

4.4.6 AWXLHV AND AWXRHV

AWXLHV and AWXRHV[5] generate subroutine calls which return the address of a parameter to a procedure using ALGOL W linkage conventions. Both macros return the parameter address in Q3, (R7 on the Meta4A, R3 on the /360). In addition, AWXLHV checks a flag in the parameter list to determine if it is legal to assign to the parameter, i.e., if the actual parameter can appear on the left-hand side of an assignment; if not, the runtime error handler is called and program execution terminates. These macros alter registers Q3 and Q4 (R3, R4 on the /360; R7, R8 on the Meta 4A). The format of the macros is:

```
[label]   AWXxHV   {parm#|(r,A)|(r,N)}[,copy]
```

label — an optional label

parm# — the ordinal number of the parameter whose address is desired, either an integer or a "#name" symbol defined in the parameter list of an AWXPROC macro

(r,A) — indicates a register containing the address of the parameter list entry (i.e., DPD) of the parameter whose address is desired; "(r,ADDRESS)" is also accepted.

(r,N) — indicates a register containing the ordinal number of the parameter whose address is desired; "(r,NUMBER)" is also accepted.

copy — an optional label or base-displacement address of where the actual parameter is to be copied to (AWXRHV) or assigned from (AWXLHV); the assembler length attribute of the label is used in an MVC. Note that the local copy of the parameter must conform to the storage conventions discussed in section 3.1; in particular, the values of REFERENCE, INTEGER and BITS datatypes are stored in the <u>low-order</u> halfword of a halfword pair on the Meta4A.

---

[5]"Left-hand-value" and "right-hand-value" of an assignment operation, respectively.

4.4.7 AWXARRAY, LIST FORM

The list form of the AWXARRAY macro builds an array descriptor, i.e., a dope vector. An array descriptor is used to access elements of arrays, and to pass arrays as parameters (a parameter list entry for an array contains a pointer to an array descriptor). The format is:

```
label      AWXARRAY  length,name,(bounds),[MF=L]
```

label — a unique label to reference an array descriptor as a parameter or for the execute form of this macro

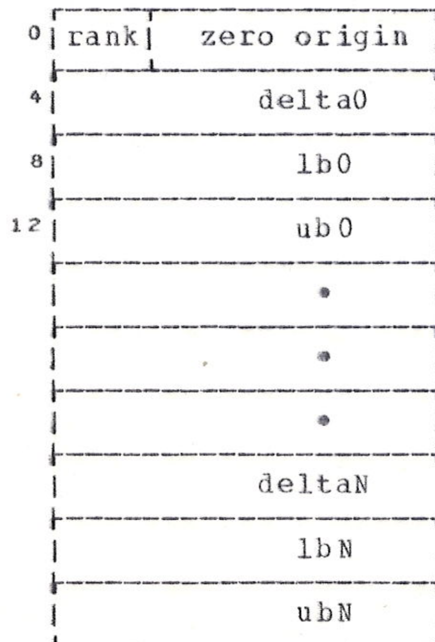
length — the length of a single array element

name — the label of the area allocated for the array

(bounds) — a list (enclosed in parentheses) of lower-bound/upper-bound pairs for each dimension, in the form (lb1,ub1,...,lbN,ubN), where each bound is an unsigned or negative integer

MF=L — indicates this is the list form of the macro.

The format of an array descriptor is:



where rank is the number of dimensions in the array, stored in the high-order byte of the first fullword (not yet implemented on the /360); origin is the address of the zeroth element (i.e., the element with zero indices) in the array; delta0 is the length in bytes of a single array element, and in general, delta(i) is the number of bytes

required to store the first i dimensions of the array; and lb0,ub0,...,lbN,ubN are the values of the lower and upper bounds of each array dimension (stored in the second halfword of a fullword).

4.4.8 AWXARRAY, EXECUTE FORM

The execute form of the AWXARRAY macro allocates (in ALGOL W's runtime automatic storage) an array defined by an array descriptor generated by an AWXARRAY list form macro. It alters registers Q0-Q4 (R0-R4 on the /360, R4-R8 on the Meta4a). The format is:

```
[label]   AWXARRAY  MF=(E,listname)
```

label — an optional label

listname — the label on an AWXARRAY macro of the MF=L form describing the array being allocated.

4.4.9 AWXALLO

This macro allocates ALGOL W runtime automatic storage similar to the way that AWXARRAY allocates an array. It alters Q1-Q3 (R1-R3 on the /360, R5-R7 on the Meta4a). The format is:

```
[label]   AWXALLO r,{size|(size)}
```

label — an optional label

r — indicates the register to be returned with the address of the area; may not be Q1, Q2, or Q3 (the macro alters these registers).

size — indicates the amount of storage to be allocated. The macro will round it up to a multiple of eight.

(size) — indicates a register containing the amount of storage to be allocated. The macro will round it up to a multiple of eight.

4.4.10 AWXFREE

This macro frees ALGOL W runtime automatic storage. It alters Q1-Q3 (R1-R3 on the /360, R5-R7 on the Meta4a). The format is:

```
[label]   AWXFREE {size|(size)}
```

label — an optional label

size — indicates the amount of storage to be freed. The macro will round it up to a multiple of eight.

(size) — indicates a register containing the amount of storage to be freed. The macro will round it up to a multiple of eight.

4.4.11 SAMPLE PROGRAM

The following is a simple function procedure to sum the elements of an ALGOL W integer array of one dimension, demonstrating the use of AWX-macros on the /360.

```
INTEGER  AWXPROC  SUM,(INTEGER,ARRAY,#ARR,(*)),CSECT=YES
         AWXENDAU  ,          END OF AUTOMATIC STORAGE

         AWXRHV  #ARR         R3 = ADDR(ARRAY DESCRIPTOR)
         L       R5,12(R3)    COMPUTE #ELEMENTS IN ARRAY
         S       R5,8(R3)     UPPER BOUND - LOWER BOUND + 1
         LA      R5,1(R5)     .
         L       R4,8(R3)     COMPUTE 1ST ARRAY ELEM ADDR
         SLA     R4,2         OFFSET = LOWER BOUND * 4
         A       R4,0(R3)     ADD ZERO ORIGIN TO GET ADDR
         SR      R3,R3        ZERO SUM

LOOP     EQU     *            DO #ELEMENTS TIMES
         A       R3,0(R4)        SUM = SUM + ARR(I)
         LA      R4,4(R4)        GET NEXT ARRAY ELEMENT
         BCT     R5,LOOP      END

         AWXEXIT (*)          SUM ALREADY IN R3
         END
```

5 PROGRAMMING AND CODING HINTS

- For testing programs the $DUMP*01[,cc] option is suggested; this causes the ALGOL W compiler to print a listing of generated machine code, from which it is easy to set checkpoints at source statements.

- For production runs, the $NOCHECK compiler option is recommended. It prevents the compiler from generating code to check for array bounds, illegal references, and the like, and may result in a considerable savings in the size of a program.

- Runtime debugging is not supported on the Meta4A, so the debug option is forced to $DEBUG,0 when the M4A option is specified. Normally, programs should be debugged on the /360, and then compiled for the Meta4A (if so desired).

- Constant expressions are not folded. Thus assigning 1.0/2.0 to a real is less efficient than assigning 0.5 to a real.

- Constants are converted to the proper type at run time, not at compile time. Thus adding 1 to a real is less efficient than adding 1.0 to a real.

- IF- and CASE-expressions and multiple assignments are compiled efficiently and should be used when applicable.

- When calling external procedures with FORTRAN linkage conventions, the most efficient way to pass parameters is by name, since only the address of the parameter need be computed.

- If error 2012 or 3102 occurs in a statement involving a call to a procedure having expressions as actual parameters, try assigning the expressions to local variables and passing the variables as the parameters.

- As an alternative to <u>bits</u>, <u>bits</u>(32) may be specified.

- When using FORTRAN linkage conventions, passing parameters by name is generally more efficient than passing them by value or result. When using ALGOL linkage conventions, the opposite is true.

- If an array declaration contains more than one array identifier, the compiler does not recalculate the bound pair list. Thus "INTEGER ARRAY A (1::10); INTEGER ARRAY B (1::10)" is less efficient than "INTEGER ARRAY A,B (1::10)".

- The compiler will not permit you to compile an external procedure that contains a REFERENCE parameter, since the RECORD to which it refers must be <u>global</u> to the procedure. To

get around this restriction, the following kludge may be applied:

(1) Code a dummy procedure, declaring the RECORD type, around the procedure that is to receive the reference parameter; e.g.,

```
PROCEDURE DUMMY;
     BEGIN
     RECORD CAR(INTEGER LICENSE);
     PROCEDURE REALPROC(REFERENCE(CAR) PTR);
          BEGIN
          .
          .
          .
          END
     END.
```

(2) Compile without OCCAM, but with the DECK and NOXEQ options.

(3) Edit the resulting text deck and delete down to and including the first END card -- this removes the dummy procedure.

(4) The deletion leaves you at the first ESD entry. Still in the editor, issue "c/002/001" (in this example making REALPROC, now DUMMY002, into DUMMY001 -- the main entry point), and "FILE".

(5) Issue "OCCAM <fn> (ICP" -- the ICP option will (continuing our example) make the entry name REALPROC instead of DUMMY001.

6 REFERENCES

[1] Bauer, H., S. Becker, and S. Graham, "ALGOL W Implementation", Stanford University (1968).

[2] Koslow, Howard, and D. Taffs, "ALGOL W Runtime Support for BUGS", Brown University (1976).

[3] Satterthwaite, Edwin H., Jr., "Source Language Debugging Tools", Ph.D. Thesis, Computer Science Dept., Stanford University (May 1975), STAN-CS-75-494

[4] Sorgie, Charles D., "ALGOL W Reference Manual", Brown University (1976).

[5] Sorgie, Charles D., "An ALGOL W /360 to Meta4A Cross-Compiler", Sc.M. Thesis, Brown University (1976).