Elliott ALGOL Recovery Project

Andrew Herbert

25th March 2015

1. Aims

The first aim of the Elliott ALGOL recovery project is to reconstruct the source code of all three versions of Elliott 903 ALGOL: the 8K two-pass system, the 16K load and go system (LG) and the 16K large program system (LP).

The second aim is to produce a revision of each version that contains:

- The last issued versions of the Elliott 903 ALGOL Translator and Interpreter (Issue 6) and Elliott 903 ALGOL library (Issue 7)
- Don Hunter's extensions to Elliott ALGOL
- Terry Froggatt's patches to allow use of both 900 and 903 telecode
- Terry Froggatt's new ALGOL plotting library
- Terry Froggatt's improved code for relational and Boolean operations
- Terry Froggatt's improved integer divide code
- Minor bug fixes and improvements.

The starting point is

- the set of sum-checked binary and relocatable binary ALGOL tapes distributed by Elliotts for 903 ALGOL:
    - ALGOL Tape 1 (Translator) Issue 6 (sum-checked binary)
    - ALGOL Tape 2 (Interpreter) Issue 6 (sum-checked binary)
    - ALGOL Tape 3 (Library) Issue 6 (relocatable binary)
    - ALGOL Tape 3 (Library) Issue 7 (relocatable binary)
    - ALGOL 16K Large Program Interpreter Issue 6 (sum-checked binary)
- Don Hunter's 16K Load and Go ALGOL System (sum-checked binary)
- Don Hunter's 8K Two Pass ALGOL System:
  ALGOL TAPE 1 (Translator) 30/12/1976 (sum-checked binary)
  ALGOL TAPE 2 (Interpreter) 30/12/1976 (sum-checked binary)
- the SIR source code of the Issue 6 ALGOL Translator
- the SIR source code of the Issue 6 ALGOL Interpreter
- the SIR source code of the 920 ALGOL Relocatable Binary (RLB)
- the SIR source code for Don Hunter's modifications and extensions to Elliott Algol
- Terry Froggatt's SIR source code patches for Elliott Algol
- Terry Froggatt's SIR source code for Algol plotting

- Terry Froggatt's SIR source code for Algol interpreter relational and Boolean operations
- Terry Froggatt's SIR source code for integer division.

Note 1: the 920 ALGOL RLB Loader is for an earlier release of 920 ALGOL than Issue 6 – it references locations in the Interpreter using the addresses specified in the Elliott Technical Manual for earlier issues than Issue 6 but the code is otherwise identical – e.g., FP (current stack frame pointer) is at location 38 rather than 138.

Note 2: Elliotts never released an Issue 6 Load and Go (LG) ALGOL system. The last released LG system is Issue 5. However Don Hunter's LG ALGOL is an Issue 6 system (the tape is labelled as such and it works with Issue 6 and 7 libraries). The objective for the LG system is therefore to recover Don's version.

Note 3: Don Hunter never produced a Large Program (LP) version of his ALGOL.

Note 4: The Herbert and Froggatt tape archives contain several different copies of Don's tapes for patching the Elliott Issue 6 ALGOL tapes. The specific tape versions used are noted in the text following.

2. Method

The general approach was to start by assembling the relevant source tapes. Where there is code missing this was recreated by using the simulator's DUMPASSIR and RLBTOSIR commands to extract an image of the system and from that derive a readable source. This process was deemed complete when the reconstructed source produces an identical image in store as the binary tape (checked using the simulator's VERIFYIMAGE command).

Part way through the project the original Elliott design documents were discovered: reading these gives useful background to understanding the structure of the ALGOL system.

3. Two Pass ALGOL Issue 6

Script: SCRIPTS/903ALGOL/BUILD_2PASS(ISS6).DAT

This is the easiest reconstruction as we have the source code for the translator and for the interpreter and the loader.

Tape 1 consists of the translator, a dictionary of built-in routines and a dump utility. The translator is in the lower half of store, the dictionary and dump utility are in the top half of the store.

The translator source tapes came from Don Hunter and are labeled "ALGOL TRANSLATOR SOURCE ISSUE 6 TAPE n OF 4 COPIED 25 JUN 1976" (n = 1,2,3,4).

No source has been found for the initialized dictionary or the dump utility so these have been reconstructed by reverse compilation.

Tape 2 consists of the interpreter, the relocatable binary loader, the built-in library routines, the loader dictionary and a dump utility. The interpreter is in the lower half of store, followed by the library routines. The dump utility, loader and dictionary are in the top half of the store.

The interpreter source tapes came from Don Hunter and are labeled "ALGOL INTERPRETER SOURCE ISSUE 6 TAPE n OF 6 COPIED JAN 1978" (n=1,2,..,6).

The loader source tape came from Terry Froggatt and is labeled ""ALGOL LOADER 19/5/69, Telecode".

The distributed Tape 3 contains the library in relocatable binary form. There is also an Issue 7 Tape 3. This contains the same routines with the exception of those concerned with plotting (QAPLT1, QAPLT2, CENTCH, DRAWLI, MOVEPE, SETORI, WAY) which are omitted and an updated QASTRI that does bounds checking on the arguments of INSTR and OUTSTR. No source for either tape has been found. Fortunately the distributed RLB is sufficient to set up the built-in library.

3.1 Tape 1

The translator source tapes do not include:

- The source code for the preloaded dictionary of built-in library routines.
- The source code for the dump facility.

The source code for the dictionary was recovered by inspection of the binary image of the Issue 6 Tape 1 and Don Hunter's notes on the design of the interpreter. Note that the translator workspace PERM points to the final entry of the dictionary.

The dump utility consists of two parts: code to dump out the content of store in T23 format and a copy of the ALGOL T23 loader which is punched out at the head of the dump.

The ALGOL T23 loader is a variant of the T23 loader found in the Elliott 903 T22-23 utility. It differs in how the checksums are validated – at the end of the tape the two checksums in store should become zero. It also outputs an initial clear store. The dump routine calculates the values required and initializes the checksum locations in T23 appropriately.

The dump utility source was created manually by editing disassembled SIR. The T23 part was left in absolute form.

Tape 1 is reconstructed in four steps.

First the separate source tapes are merged into one tape. Then SIR in upper store is used to assemble the interpreter and the assembled code is dumped out as sum-checked binary using T22-23.

Next, SIR in lower store is used to assemble the dictionary, dump utility and ALGOL T23 loader together with QBINOUT, which is then used to output the dictionary, dump utility and T23 loader as a sum-checked binary tape. (The dictionary grows down in store from the base of the dump utility; the translator stack grows upwards in store from the top of the translator.)

To exactly match the image loaded by the distributed tape, locations 2-7 of store are initialized to the required values even though the contents of these locations are insignificant to the execution of the translator.

Then the two tapes are then loaded in sequence under initial instructions and the new Tape 1 punched by entry at 8001.

3.2 Tape 2

The interpreter source tapes do not include:

- The source code of the built-in library routines
- The source code for the dump facility.

The library routines are those from the Issue 6 Tape 3 excluding the graph plotting routines (QAPLT, QAPLT1, CENCHA, DRAWLI, MOVEPE, SETORI).

The source tape for the RLB loader is for a version of 920 ALGOL earlier than Issue 6 so the addresses for linkage to the interpreter are wrong and need to be changed, e.g., 32 becomes 132 etc.

The dump utility is the same as used for Tape 1.

Programs and code procedures are loaded into start starting from the top of the interpreter. The RLB loader dictionary grows down from the base of the loader.

Tape 2 is reconstructed in six steps.

First the separate source tapes are merged into one tape, then SIR in upper store is used to assemble the interpreter and the assembled code is dumped out as sum-checked binary using T22-23. Next, SIR in lower store is used to assemble the RLB loader, dump utility and ALGOL T23 loader together with QBINOUT, which is then used to output the loader, dump utility and T23 loader as a sum-checked binary tape.

The built-in library is input as RLB. This was created by editing the Issue 6 library file distributed by Don Hunter with his simulator. This library is in decimal encoded binary format with textual labels at the head of each routine in the library. By careful cut and paste it was possible to create a binary file that loads and identical image to the built-in library of the distributed tape.

Next, the two tapes are then loaded in sequence under initial instructions and the built-in library is read in at 12 as an RLB tape.

To exactly match the image loaded by the distributed tape, locations 2-7 of store are initialized to the required values even though the contents of these locations are insignificant to the execution of the interpreter.

Finally the new Tape 2 is dumped out by entry at 14.

Tape 3

The original binaries are used as is for TAPE 3 Issue 6 and 7.

4. Large Program ALGOL (Issue 6)

Script: SCRIPTS/903ALGOL/BUILD_ALG16KLP(ISS6).DAT

The LP system consists of a modified interpreter, dump utility, default library routines, RLB loader and dictionary.

No original source code of this system has been found so a source was manually constructed by manually comparing the binary image of the issue 6 2-pass interpreter with that of the Elliott distributed LP interpreter binary. From this a modified and annotated version of the issue 6 2-pass interpreter was written which assembled to produce an identical binary to the LP system. With hindsight it would have been better to have written a Bowdler steering tape to make the changes.

The interpreter is in the lower half of the first store module, followed by the dump utility then the library routines. The RLB loader and dictionary are in the top half of the second store module.

The LP interpreter is a heavily modified version of the 8K system interpreter due to:

- The need to modify some interpreter primitives to allow programs to extend across store modules (subject to a restriction on the total length of the ALGOL program being <= 8192 words).
- The need for cross store module linkages between the interpreter and the RLB loader (since they are now in different store modules).
- different entry points.
- The teleprinter control interface in place of entry points.

The LP RLB loader is a heavily modified version of the 8K system RLB loader due to:

- differences in how ALGOL intermediate code is loaded (addressed relative to the start of the code rather than absolutely).
- changes to allow the dictionary and forward reference chains to extend across store modules.
- changes to enable to RLB loader to access the interpreter (since they are now in different store modules).

Note that all code procedures (including library routines) must fit in the first 8179 words of store. The ALGOL intermediate code starts immediately following the library and code procedures and can be up to 8192 words long. The base address for loading a code procedure is its ABSOLUTE starting address in store, whereas the base address for the ALGOL intermediate code is RELATIVE to its starting location.

There is code in the interpreter to support device numbers other than 1 and 3, but this appears to depend on some sort of run-time setup of a data structure in the locations N, N+5.

There is further code in the LP RLB Loader the purpose of which is unknown. It is called when the Loader is first initialized and would appear to search through the dictionary for the names QIN, QOUT, QCHOP, QCHIN and EX, and if found patch the interpreter to use them instead of its built in basic I/O routines. As distributed, the LP System binary has the flag that enables this initialization step unset, so the code is never executed. (One theory is this code is something to do with building a version of the LP System for the FAS operating system.)

Programs and code procedures are loaded upwards in store from above the dump utility which itself follows the interpreter. The RLB loader dictionary grows down store from the base of the loader.

The recovered version of the 16K LP binary is built using ACD 2-pass SIR for convenience: it is highly unlikely that Elliotts used this method: there strong evidence, from the three spurious items of data in words 15, 16, 17 that MASIR was used (these words are a "signature" of sumchecked binary tapes produced by the MASIR Loader).

An issue arises with literals in the Interpreter: there are two occurrences each for the constants +28 and +8192 in the literals area in the distributed version.
A similar issue arises with the literal +8112 in the RLB loader. This is addressed by substituting dummy literal values in the source code and using the simulator ENTER command to patch in the correct values. This is probably also a MASIR artifact due to the handling of symbolic references between store modules in the original source. As a follow on project it would be worth further editing the source to replace absolute inter-module references by symbolic ones – as has been done in the derived AJH ALGOL LP interpreter.

The distributed 16K LP binary has the remains of the ALGOL T23 loader in the top end of the first store module. This is patched into the simulator store using the Enter command so that VERIFYIMAG doesn't complain, but serves no useful purpose.

The resulting store image is dumped by entry at 14. The output tape loads the same image in store as the distributed binary, with the exception of having different ALGOL T23 Loader checksums, which is not significant.

4 2 Pass HUNTER ALGOL System

Script: SCRIPTS/HUNTER/BUILD_2PASS(HUNTER).DAT

Don Hunter's tapes include an ALGOL Tape 1 and Tape 2, labeled "ALGOL TAPE 1 AS PATCHED 30/12/76" and "ALGOL TAPE 1 AS PATCHED 30/12/76" respectively.

These tapes can be reconstructed by taking the corresponding Elliott distributed Issue 6 tape and applying Don Hunter's patches.

The source code for the Tape 1 patches is in the file TAPE1PATCH(301276).900 and was extracted from the paper tape "PATCHES SOURCE 25 NOV 76 (26 NOV MOD) 30 DEC 1976" from my collection of Don Hunter paper tapes. A single error (misspelled identifier INDA) was corrected and the source compiled successfully. Once assembled, the patch must be entered at 256 and then 512 to relocate some of the patch into high store (i.e., where the assembler sits). The store is then dumped using T22-23. The steering tape for this was derived by inspection of the contents of store as the steering tapes in "PATCHES SOURCE 25 NOV 76 (26 NOV MOD) 30 DEC 1976" are somewhat garbled.

The result of loading the dumped patch tape over the 903 ALGOL Issue 6 Tape 1 and then dumping this out by entry at 8001 is a tape that loads an identical image to "ALGOL TAPE 1 AS PATCHED 30/12/76".

The source code for the Tape 2 patches is in the file TAPE2PATCH(301276).900 and was extracted from the paper tape "PATCHES SOURCE 25 NOV 76 (26 NOV MOD) 30 DEC 1976" from my collection of Don Hunter paper tapes. A few minor garbled textual errors were corrected and the source compiled successfully. Once assembled, the patch must be entered at 512 to cause part of the patch to be moved into high store (i.e., where the assembler sits). The store is then dumped using T22-23. The steering tape for this was derived by inspection of the contents of store as the steering tapes in "PATCHES SOURCE 25 NOV 76 (26 NOV MOD) 30 DEC 1976" is somewhat garbled.

The result of loading the dumped patch tape over the 903 ALGOL Issue 6 Tape 2 and then dumping this out by entry at 8001 is a tape that loads an identical image to "ALGOL TAPE 1 AS PATCHED 30/12/76".

Note: entry point 15 of the Interpreter is concerned with how call-by-name implicit procedures (thunks) are handled and is not for use by the machine operator (see separate document "ALGOL Interpreter Notes").

5 Hunter 16K Load and Go System

Script: SCRIPTS/HUNTER/BUILD-ALG16KLG(HUNTER).DAT

The Hunter 16K Load and Go System is distributed in two forms: a paper tape "903 ALGOL ISS 6 16K LG VERSION 22 NOV 84" in my collection of Don Hunter tapes, and as two "module" files MO.DAT, M1.DAT on Don's web site for his Elliott 903 ALGOL simulator. These two produce similar store images and the differences between them are not significant, i.e., they are in store not used by the system itself but by initial instruction and T23 style loaders.

The paper tape gives the most information about the construction process. Using the simulator SCB command I discovered the tape is actually two tapes back to back. The first loads the translator and copies it to the upper store module: the second loads the interpreter. Both sub-tapes are in the form output by the standard ALGOL dump routine, with the minor change that the loader jumps to location 10 after loading rather than entering a dynamic stop. This is to initialize each sub-system.

The loaded system is clearly derived from the two-pass system tapes, i.e., the translator is 903 ALGOL Issue 6 with TAPE1PATCH(301276).900 applied and the interpreter is 903 ALGOL Issue 6 with TAPE2PATCH(301276).900 applied.

An additional patch is applied to the Hunter LG translator: this has been identified as the tape "TAPE 1 PATCH switch list Nov 77 + May 80 Fix Source 15 Oct 83" in my collection of Don Hunter tapes. The file LABELPATCH(151083).900 is an annotated version and reformatted copy. The patch adds code to allow labels to be used without requiring them to be declared in switch statements.

After assembly this patch must be initialized by entry at 1024. On loading the initialized patch must be entered at 10 to complete the initialization (by copying code to location 8000 onwards). This patch can therefore be applied also to the 8K 2 pass System Tape 1, but in so doing will overwrite the dump utility.

Two additional patches are applied to the LG interpreter. These have been identified as corresponding to two further tapes in my collection of Don Hunter tapes: "For block / actual parameters type proc problem source" (and the source for the binary tape "FOR BLOCK/ACTUAL PARAMETERLESS TYPE PROCEDURE PROBLEM TAPE 2 PATCH 23 SEP 1978") and "MAY 1980 patch reworked for 8K system TAPE 2 part 15 OCT 1983".

The file FORPATCH(230978).900 is a reformatted copy of the tape "For block / actual parameters type proc problem source". It appears to be a bug fix to TAPE2PATCH(301276), correcting one instruction at location 34.

The file TAPE2PATCH(151083).900 is a reformatted copy of MAY 1980 patch reworked for 8K system TAPE 2 part 15 OCT 1983". It similarly appears to be further corrections to the handling of procedure parameters.

The T22-23 steering tapes for these patches can be found in the file BUILD_ALG16KLG(HUNTER).DAT.

The two files MOD0PATCH(0878).900 and MOD1PATCH(0878).900 are further patches that build the load and go system. The first is derived from the tape "Module 0 patch (source) 26 AUG 1978" and the second from the tape "MODULE 1 PATCH SOURCE 19 AUG 1968" from my collection of Don Hunter paper tapes.

These two patches set up the interface by which the translator feeds ALGOL intermediate code directly to the interpreter, and the entry points for the load and go system.

The code in MOD1PATCH(0878).900 has to be initialized after assembly by entry at 1024 before dumping out.

The T22-23 steering tapes for these patches can be found in the file BUILD_ALG16KLG(HUNTER).DAT.

Both module patches must be entered at location 10 when the system starts to finalize the initialization (and in the case of MOD1PATCH, copy the program to the second store module).

This presents a problem as LABELPATCH also needs to be entered at 10 as part of the start up. This is overcome by using the simulator to patch the loop stop at the end of LABELPATCH's initialization with an "8 10" instruction.

Once the translator is started the interpreter needs to be read in. This is achieved by using the simulator to patch an "8 8181" at the end of the MOD1PATCH start up code.

We cannot use the normal ALGOL dump utility at location 8001 to produce the output tape as the various patches overwrite it. Therefore a standalone version which loads at location 6144 has been created: see script SCRIPTS/HUNTER/BUILD_DUMP(HUNTER). This dumps out 2-6144 and 6319-8135 of module 0. The T23 loader at the head of the output terminates with a jump to location 10.

Note: the comments in MOD0PATCH refer to dumping out the entire system using a modified version of T22-23, i.e., one that finishes with a jump to location 10.

The translator should be constructed first as follows:

1. Load 903 ALGOL Tape 1 Issue 6

2. Load TAPE1PATCH(301276).BIN
3. Load MOD1PATCH(0878).BIN
4. Patch location 6824 to =8 8181 (so patch will trigger loading of second part of tape)
5. Load LABELPATCH(151083).BIN
6. Patch location 7009 to = 8 9 so that when label patch is entered at 10 by the loader, it then goes on to start MODPATCH1.
7. Dump out using DUMP(HUNTER).BIN.

DUMP(HUNTER) creates a dump of the translator and patches in T23 format, preceded by a clear store and then jumps to location 10, which will start the label patch. When the label patch is started, it in turns starts MODPATCH1 by entry at 9. MODPATCH1 copies the modified translator to the second store module and then jumps to 8181 to read in another tape (i.e., the interpreter).

The interpreter portion of the LG system tape is produced as follows:

1. Load 903 ALGOL Tape 2 Issue 6
2. Load TAPE2PATCH(301276).BIN
3. Load TAPE2PATCH(151083).BIN
4. Load FORPATCH(230978).BIN
5. Load MOD0PATCH(0878).BIN
6. Dump out using DUMP(HUNTER).BIN

DUMP(HUNTER) will produce a tape containing the interpreter and patches and then start MOD0PATCH by entry at 10.

The end result is an initialized system.

6. Large Program ALGOL (AJH Version)

This is a revised version of the Issue 6 LP System with the following modifications:

- Significant tidying up of the source.
- Removal of redundant code.
- Addition of Don Hunter modifications.
- Addition of Terry Froggatt modifications.

Script: SCRIPTS/AJHALGOL/BUILD_ALG16KLP(AJH).DAT, SCRIPTS/AJHALGOL/BUILD_ALG64KLP(AJH).DAT

The baseline for this was the 903 ALGOL Issue 6 Large Program (16K LP) System described in Section 3 above.

All workspaces in the program were reviewed and those that did not require a specific initial value converted to skips.

Code identified as redundant was removed:

1. in the RLB Loader the call to `INIT` was removed as it was essentially a do nothing,

2. the subroutine `FINDIO` was removed as it was essentially a do nothing, and

3. the code in the interpreter for handling additional devices in `RUNALG2` was removed.

All inter-module address references were made symbolic.

A bug was removed in the handling of references to unlocated global labels: the original code treated the 8191 end of fixup chain marker as relative rather than absolute when loading code procedures.

The location of the base of the library and of the base of the RLB Loader dictionary was shifted up to maximize the free space available for programs.

ALGOL Tape 3 Issue 7 has been used to create the built-in library (to pick up improved parameter checking in `INSTRI` and `OUTSTR`).

The character decoding table (global identifier `TABLE`) was updated as follows to handle 900 telecode as well as 903 telecode:

`^TABLE+92` from `&170074 (7 4156)` to `&170003`
this converts old £ to new \ on input. (Note old ½ is new £).
`^TABLE+96` from `&600102` (`/8 66`) to `&300040`
this permits new ` on input and converts the old ` to new ` on output.

```
^TABLE+123 from &666100 (/11 3136) to &666007
this permits { on input as an alternative to '
^TABLE+124 from &270300 (11 4288) to &506300
this converts old £ to new £ on output
^TABLE+125 from &672100 (/11 5184) to &672040
this permits } on input as an alternative to the old `
```

The sequence

```
        0     +26
        /1    0
```

occurs several times in the original (e.g., TA primitive). These were replaced by

```
        1  STKMOD which achieves the same result.
```

Similarly a sequence

```
        0     +28
        /4    0
```

was replaced by

```
        4  WARNAD.
```

Changes to the Interpreter primitives embodied in TAPE2PATCH(301276).900, TAPE2PATCH(151083).900 and FORPATCH(230978).900 were consolidated and edited into the source code. This was non-trivial as the Hunter changes had to be merged with the changes required to convert the normal Interpreter into the Large Program version. In general terms the LP Interpreter is complicated by the fact the loaded ALGOL program can be up to 8192 words long, straddling the store module 0, module 1 boundary. Therefore the addresses in ALGOL Interpreter instructions are relative to the base of the program and the absolute location has to be computed.

An important difference between the LP system and the 2-pass system is in the handling of the CF operation (Call Function, /5). To deal with the fact that ALGOL procedures are stored with relative addresses whereas SIR code procedures are stored with absolute addresses it is necessary to distinguish between call of ALGOL versu SIR procedures.

The RLB loader in the LP system assumes that CF used with a global identifier as its address (e.g., CF SIN+0) is a call to a SIR code procedure, and in this case edits the function code to become /9. In the 2-pass system /9 jumps to TFVR which is shared code with TFAI, TFAR and TFVI, whereas in the LP system it jumps to new code (which I've called CFSIR) to handle calls of code procedures. (All of which suggests TFVR is an unused interpreter function in Issue 6 ALGOL). The LP code at CFSIR deals with the fact that code procedures need to use module relative addresses, whereas ALGOL code is relative to the start of the program.

Don Hunter's Algol also steals the `/9` function but binds it to new code called `GETADD` which implements aspects of call-by-name. This conflict in the use of `/9` has been resolved as follows:

1. `/9` is bound to `GETADD`
2. `/14` (formerly `PEM`) is bound to `CFSIR`
3. `CF <label>` is translated as `/14 <label>+32`
4. `CFSIR` first checks to see if the address is less than or equal to 31 (i.e., a parameter count) and if so transfers control to `PEM`. This is safe as procedures are limited to 14 parameters. If the address is 32 or greater, 32 is subtracted from it and added to the base address for loading code procedures to calculate the module relative address of the global label.

A similar complication arises with the `TA (0)` function: Hunter Algol will generate, for example `TA SQRT+0` if `sqrt` is used as an actual argument for a real call-by-name formal parameter. (This situation cannot arise in 903 Algol Issue 6 where call-by-name arguments are limited to scalars and constants.) To deal with this, `TA <label>` is translated as `15 <label>+32`. In the decoding of `15 (INOUT)` an address greater than 31 is assumed to be `TA <address>-32` and control passed to `TASIR` accordingly.

Note that because of the overloading of `CF` and `TA` on `PEM` and `INOUT` respectively for code procedures, the highest address that can be given a global label is 8159. Code procedures can still extend through to location 8191, but no global label should be defined in the final 32 words. The loader produces a `FG` message if such a label definition is encountered.

Don Hunter's translator emits a `HALT 15` RLB directive after a thunk has been generated. In his interpreter, this jumps to location 15 where there is code that modifies the thunk to its final form and then resumes the loader to continue with loading the rest of the program. In the `LP` system `HALT 15` is specifically detected and an inter-module call made to the thunk processing code.

The `Halt 15` code looks for certain instructions at certain offsets in the generated thunk. The offsets don't match generated code examples, nor have some of the instructions been seen in thunks. The apparently redundant code has been removed, but may have to be replaced if examples needing it are found.

There is a bug in the Hunter Algol interpreter patches concerned with call-by-name label parameters. The translator permits label formal parameters to be either by name or value However, the shared code in `GT` for searching for the label assumed only call-by-value label arguments. This has been corrected with additional code for call-by-name, and within that code a special case for handling thunks.

A designational expression is permitted as argument when referencing a formal label parameter by name. However the generated thunk didn't work. The `Halt 15` code has been modified to detect this case and replace the generatedINDS n by GTS. N instead.

The operator GETADD calculates the address of a call-by-name formal parameter and included code for evaluating thunks. I have replaced this with code to signal run-time error 21 “attempt to assign to a constant”.

CFSIR previously contained complicated code, which tested whether or not the address of the code block is above or below location 4096. This code has been removed. (I have yet to find any reason why it was needed – perhaps to call functions built into the loader in the second module, such as FINDIO?).

Don’s revised code for locating formal parameters (FINDFP) didn’t work on some complex recursive examples with embedded procedure parameters or thunks. It has been completely rewritten, as has the similar code in the GT instruction to unwind the stack back to the parent block of the target label.

The code for the RETURN operation has been edited to update the EVN variable, this was missing from Don’s code and caused errors with some test cases.

Two changes have been made to the primitives:

1. New code for integer division (‘div”) from Terry Froggatt has been incorporated – the new code correctly handles -131072 as a numerator
2. New code for integer relational operations from Terry Froggatt has been included – the new code optimizes overflow handling, making it shorter and faster.

7. 2-Pass ALGOL (AJH Version)

This is an improved version of the Hunter 2-Pass System.

Script: SCRIPTS/AJHALGOL/BUILD-ALG2PASS(AJH).DAT

Translator

The translator source is constructed by editing the 903 Algol Issue 6 translator as follows:

1. Converted for assembly by ACD 2-pass SIR.
2. Redundant blanks lines removed, and variable workspace left uninitialized by replacing +0 constants by skips (i.e., >1). Since the tape starts with an initial clear store, this achieves the same effect and helps distinguish constants and initialized variables from general workspace.
3. Remove redundant holes in the code (presumed required by e.g., FAS, spare space for patching etc).
4. Remove redundant data from the name list and moved it upwards by 2 locations, to release additional workspace.
5. Editing in the Hunter patches from TAPE1PATCH(301276).900, LABELPATCH(151083).900 and _ (delete line) facility from MOD1PATCH(0878).900.
6. Modify the character decoding table to handle 900 telecode:
   `^TABLE+92` from `&170074 (7 4156)` to `&170003`
   this converts old `£` to new \ on input. (Note old ½ is new £).
   `^TABLE+96` from `&600102 (/8 66)` to `&300040`
   this permits new ` on input and converts the old ` to new ` on output.
   `^TABLE+123` from `&666100 (/11 3136)` to `&666007`
   this permits `{` on input as an alternative to ‘
   `^TABLE+124` from `&270300 (11 4288)` to `&506300`
   this converts old `£` to new `£` on output
   `^TABLE+125` from `&672100 (/11 5184)` to `&672040`
   this permits `}` on input as an alternative to the old `
7. Reposition the workspaces (`W`, `INBUF`, `CODL`) to be clear of the translator program.
8. A new pair of entry points (17, 18) have been provided to enable call-by-name to be enable or disabled, rather than by patching.
9. The name list has been converted from absolute to symbolic addressing allowing it to be repositioned and redundant data removed.

In applying the patches the original source is edited to include the patch inline where possible. This has required the introduction of new global labels `FPARAM`, `UJ`, `UJON`, `UJOFF`.

Since the patches are inline, the translator DUMP function remains available at location 8001.

The AJH Algol Tape 1 comprises:

1. The modified translator as described above.
2. The same dump utility and T23 package as the issue 6 Tape 2.
3. The tidied up name list.

The correctness of the translator has been verified by checking it compiles a large set of demonstration programs, including several that show off the extended features of Hunter Algol.

Note: when modifying the AJH translator be sure to check that the translator literals have not overflowed into the area allocated for `W`, `INBUF`, `CODL`.

Interpreter

The interpreter was constructed by editing the issue 6 interpreter to include all the relevant changes made to create the AJH LP system, suitably modified to remove those aspects concerned with supporting large programs. Note in particular the CFSIR, TASIR changes are not required in this version.

The interpreter thus embodies:

1. Conversion for assembly by ACD 2-pass SIR.
2. Redundant blanks lines removed, and variable workspace left uninitialized by replacing +0 constants by skips (i.e., >1).
3. Removal of redundant holes in the code (e.g., spare space for patching etc).
4. Incorporation of the Hunter patches with my improvements and corrections from the LP system.
5. The character decoding table (global identifier `TABLE`) was updated as follows to handle 900 telecode as well as 903 telecode:

    `^TABLE+92` from `&170074 (7 4156)` to `&170003`

    this converts old `£` to new \ on input. (Note old ½ is new £).

    `^TABLE+96` from `&600102` (`/8 66`) to `&300040`

    this permits new ` on input and converts the old ` to new ` on output.

    `^TABLE+123` from `&666100 (/11 3136)` to `&666007`

    this permits `{` on input as an alternative to `‘`

    `^TABLE+124` from `&270300 (11 4288)` to `&506300`

    this converts old `£` to new `£` on output

    `^TABLE+125` from `&672100 (/11 5184)` to `&672040`

    this permits `}` on input as an alternative to the old `.

Loader

The loader was constructed by editing the issue 6 loader to include a Hunter patch to handle the use of `HALT  15` to fix up thunks – the patch suppresses FIRST NEXT output.

Since the loader and interpreter are assembled as MASIR program modules, the references between them for entry points etc have all been made symbolic.

The AJH Algol Tape 2 comprises:

4. The modified interpreter as described above.
5. The modified interpreter as described above.
6. The same dump utility and T23 package as the issue 6 Tape 2.
7. A built-in library derived from the issue 7 Tape 3.

8. 16K Load and Go System (AJH Version)

General

The load and go system has the interpreter and loader in store module 0 and the translator in store module 1. When a translation entry point is triggered, the interpreter computes to translator option, sets a “first time” flag and calls the loader. In the loader, the first time flag directs entry to the translator with the required option set. This is done in the BLANKS routine. A second flag is set to indicate the call is from BLANKS. The translator rlb punching routine (PUNGRP) transfers control to RETGRP in the loader that detects if the return is from either the BLANKS or READTH code. In either case the rlb triple is copied across from the translator to the interpreter and the routine resumed. The rlb triple is decoded, and if further triples are required, the loader calls READTH, setting the second flag accordingly. Once the full rlb command has been read, the second flag is reset and the next triple called for by returning to the BLANKS routine.

The 16K LG system is built using MASIR to enable cross store module references.

Because the MASIR loader overwrites locations 15-18 and treats the instruction 14, /14, 15 or `/15  LABEL` as special cases a small fixup program has been added to repair such problems and is located in free store. This program is run when the entire system is loaded after assembly, repairs the problems and then punches out a paper tape containing the corrected system image.

The final output tape consists of modified versions of the AJH 2-Pass translator, name list table, interpreter, built-in library and loader.

Because the code uses global identifiers to link the program modules it has been possible to simplify the Don Hunter code for communication between interpreter, loader and translator and delete a considerable amount of redundant code. Also with the ALGOL dump utilities and T23 loaders not required, further space has been made available to maximize the size of programs that can be translated.

Translator

The LG translator was made by editing the AJH Algol 2-Pass translator to achieve the effect of the Hunter “MOD1PATCH” tape.

1. A number of global labels have been made sub-global to avoid conflict with labels in the interpreter or loader.
2. All translator entry points are removed as they are now part of the interpreter.

3. The `W`, `INBUF` and `CODL` workspaces have been repositioned clear of the translator code and literals.
4. There is no dump utility required at 8001 so the name list has been moved up in store to create more space for CODL and name list entries.

Interpreter

The LG interpreter was made by editing the AJH Algol 2-Pass interpreter to achieve the effect of the Hunter “MOD0PATCH” tape.

The value of BASE was adjusted to be the first free location after the interpreter.

Because no DUMP program is required at 8001, the loader has been moved as high as possible in store (the MASIR loader starts at 8130).

Loader

Since there is no Dump utility, the loader has been moved to the top of store module 0 to increase program and data space.

9. UNSIGNED and REVERT Code Procedures

Some of the Hunter Algol demonstration programs uses two complementary code procedures called "unsigned" and "revert" to modify integer printing. These routines patch locations within the INOUT code for integer printing. These procedure do not work in AJH Algol because the relevant code moves to different places in the various versions (2-Pass, Load-and-Go, Large Program). However some of the locations in the INOUT code are referenced from the Interpreter standard interface, in particular the address OUTI for printing integers and OSTSB for printing strings, so a new version of unsigned has been written addressing the location to be patched via these addresses. This enables a version to be written that works for all 903 Algol Issue 6 library compatible systems, i.e., 903 Algol, MASD Algol, Hunter Algol and AJH Algol, excepting the load-and-go systems in the first two.

One of the examples using unsigned can be further improved by printing leading zeros, so a suite of routines has been written:

```
"code" "procedure" unsigned; "algol";
"comment" print integers without sign (space or -);
"code" "procedure" leadzero; "algol";
"comment" print leading zeros of integers;
"code" "procedure" revrtu;   "algol";
"comment" revert to normal sign printing;
"code" "procedure" revrtl;   "algol";
"comment" revert to suppressing leading zeros;
```

10. Known Bugs

1. AJH Algol:
    1. Use of undeclared variable as actual parameter is reported as jump to undefined label (error 79)
    2. Call of a function outside an expression does not compile code (should give an error), e.g.,
       `“for” i, i+1 “while” i<5 “do” function(x);`
       where function is e.g., an `integer` function.

11. Measurements

Measured savings in memory locations:

	Issue 6	AJH Version	Saving
Interpreter	8-3960	8-3905	+55
Loader etc	7542-7999	7543-7999	+1
TOTAL			+56
LG Interpreter	N/A	8-3907	
LG Loader etc	N/A	7581-8129	
TOTAL			
LP Interpreter	8-4429	8-4232	-3
LP Loader etc	7333^1-8190^1	7530^1-8190^1	+197
TOTAL			+194

The gap between the top of the interpreter and the bottom of the loader represents the free store for programs and data (and associated loader dictionary). The "saving" figure shows the additional space available in the AJH versions.

Measured speeds on Whetstone benchmark (20 cycles)

2-Pass ALGOL

System	Instructions Executed	CPU Time (secs)	CPU Time Factor
ALGOL Issue 6	83,895,190	2,294.992	1.00
Hunter	87,471,626	2,391.718	1.04
AJH version	86,417,406	2,365.627	1.03

Large Program ALGOL

System	Instructions Executed	CPU Time (secs)	CPU Time Factor
ALGOL Issue 6	85,266,050	2,330.17	1.02
Hunter	N/A	N/A	N/A
AJH version	86,794,181	2,374.384	1.03

Measured speeds on recursive QUICKSORT with 3000 numbers:

	Instructions Executed	CPU Time (secs)	Factor
16KLG(HUNTER)	93,604,565	2,540.682	1.00
16KLG(AJH)	90,634,645	2,465.838	0.97

Measured speeds on ALMAT examples with 51*51 matrix inversion numbers:

	Instructions Executed	CPU Time (secs)	Factor
16KLP(ISS6)	192,415,632	5,275.927	1.00
16KLP(AJH) - normal	195,701,588	5,382.639	1.02
16KLP(AJH) - optimised	195,575,706	5,379.202	1.01

Optimised: call-by-name thunk generation switched off.