LINUX VOICE TUTORIAL

ATLAS: THE UK'S SUPERCOMPUTER

JULIET KEMP

In the 1950s came the transistor, and with the transistor came the supercomputer – here's how to program one of the first.

Atlas, in Manchester, was one of the first supercomputers; it was said that when Atlas went down, the UK's computing capacity was reduced by half. Today supercomputers are massively parallel and run at many, many times the speed of Atlas. (The fastest in the world is currently Tianhe-2, in Guangzhou, China, running at 33 petaflops, or over a thousand million times faster than Atlas.) But some of the basics of modern computers still owe something to the decisions made by the Atlas team when they were trying to build their 'microsecond engine'.

The computers of the early 1950s were built with vacuum tubes, which made for machines which were enormous, unreliable, and very expensive. The University of Manchester computing team already had one of these, the Manchester Mark 1 (which Alan Turing worked with), which began operation in April 1949. They were working on a smaller version when Tom Kilburn, director of the group, set a couple of his team to designing a computer which used transistors.

"Germanium transistors were even less reliable than valves, but were a lot cheaper to run."

The result was the Transistor Computer, the world's first transistorised computer, first operational in April 1953. It used germanium point-contact transistors, the only type available at the time, which were even less reliable than valves; but they were a lot cheaper to run, using much less power. It did still use valves for memory read/write and for the clock cycle, so it wasn't fully transistorised. (The first fully transistorised computer was the Harwell CADET, in 1955.) Once junction transistors became available, the second version of the machine was more reliable.

The University of Manchester's ATLAS machine, photographed on 1 January 1963. Photo: Iain MacCallum

Building Atlas

After the success of the Transistor Computer, the next challenge the team set themselves was to build a "microsecond engine" – a computer that could operate at one microsecond per instruction (or close to it), so managing a million instructions a second. (This is not quite the same as one megaflop, as FLOPS measure floating-point operations, not instructions, and are a little slower than instructions.)

The machine was initially called MUSE (after the Greek letter μ, meaning one-millionth), but was renamed Atlas once the Ferranti company became involved in 1958. When Atlas was officially first commissioned, in December 1962, it was one of the most powerful computers in the world, running at (at peak) 1.59 microseconds per instruction, or around 630,000 instructions/second.

Atlas was an asynchronous processing machine, with 48-bit words, 24-bit addressing, and (in the Manchester installation) 16k word core store and 96k word drum store. It also had over a hundred index registers to use for address modification. It was fitted up for magnetic tape (a big novelty at the time and much faster than paper tape).

One important feature was instruction pipelining, which meant being able to speed up programs beyond merely running instructions more quickly. With instruction pipelining, the CPU begins to fetch the next instruction while the current one is still processing. Instead of holding up the whole CPU while a single instruction goes through the CPU's various parts, pipelining means that instructions follow one another from point A to point B to point C through the CPU, maximising the amount of work being done by the CPU at a particular time, and minimising the overall time. Obviously this does require appropriate programming to take advantage of it.

Atlas' "Extracode" setup also allowed certain more complex instructions to run as software rather than be included in the hardware. The most significant bit of the top 10 bits of a word determined whether an instruction was a normal hardware instruction, or an Extracode instruction. An Extracode instruction meant that the program would jump to what was basically a subroutine in the ROM, and run that. This was a way of reducing the complexity of the hardware while still being able to provide those complicated instrutions 'baked in' to the machine (and thus easy to use for programmers). Extracode instructions were used particularly for calculations like sine/cosine and square root (inefficient to wire into the hardware); but they were also used for operating system functions like printing to output or reading from a tape. (See the next section for more on the Atlas Supervisor.)

The first production Atlas, the Manchester installation, started work in 1962, although the OS software, Atlas Supervisor (see below) wasn't fully operational until early 1964. Ferranti and the University shared the available time on Atlas, running between them up to a thousand user programs in a 20-hour 'day'. The value of the machine to the University was estimated at £720,000 per year in 1969, if they'd had to buy it in externally.

Software

A 48-bit Atlas instruction was divided into four parts: a 10-bit function code, 7-bit Ba (bits 10-16) and 7-bit Bm (bits 17-25) index registers, and a 24-bit address. There were two basic types of instruction: B-codes, which used Bm as a modifier and Ba as a register address and did integer operations; and A-codes, which provided floating point arithmetic.

The B index registers were used to modify the given address to get the 'correct' one (useful for moving through a series of memory locations); having two index fields meant Atlas could be double-indexed. You could also test the Bm register and then do something specific with the contents of the Ba register, depending on the result of the test. Specifically, there was a general form of:

"if CONDITION then load register Ba with address N (and optionally act on Bm); otherwise do nothing"

Since register B127 was the program counter, this could be used as a program operation transfer. Simply set N to the location you want to jump to, and set Ba to B127. If the condition is true, B127 now contains address N, and the program jumps to N.

Other B registers also had specific roles, and there is a comprehensive list of these registers and many other details of the system in the short book *The Story of ATLAS* by Iain Stinson (**http://elearn.cs.man.ac.uk/~atlas/docs/london%20atlas%20book.pdf**).

Atlas Supervisor was the Atlas operating system, which managed resources and allocated them between user programs and other tasks, including managing virtual memory. It's been called "the first recognisable modern OS" in terms of how it managed jobs and resources. At any given time, multiple user programs could be running, and it was Atlas Supervisor's responsibility to manage resources and workload. The Scheduler and Job Assembler would assemble all parts of a job and sort it into one of two queues (requires its own magnetic tape, or does not). The Central Executive took care of program-switching, error-monitoring, Extracodes, and memory management. Output Assembly handled output potentially onto many different devices, maintaining a list of documents to be output.

One of the radical innovations of Atlas was virtual memory. Computers had (and still have) two levels of memory: main (working) memory and secondary (disk; or drums/tapes back in the 1950s) memory. A program can only deal directly with main memory. For a programmer trying to perform a calculation (such as matrix multiplication) that couldn't fit into main memory, a large part of the job became working out how to switch data in and out of secondary memory, how to do it most efficiently, what blocks (pages) to divide it into, how to swap it in and out, and so on. The designers of Atlas were working programmers (as was everyone working in computers at the time), and it was very clear to them that automating this process would make programmers' lives much easier. Atlas' virtual memory had three important features:

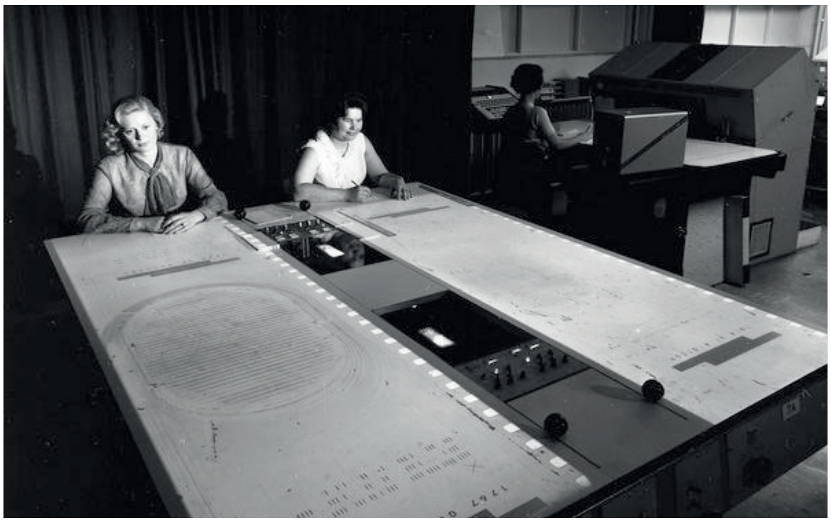
- It translated addresses automatically into memory locations (so the programmer didn't need to keep

Transitors

Vacuum tubes, used in all the 1940s computers, were far from ideal. Everyone in the industry was keen for something different. In particular, Bell (the telephone company) wanted a more reliable component for telephone systems. It put together a team to research transistors, based on an idea patented in the late 1920s by physicist Julius Lilienfeld. The Bell Labs team produced a working transistor in 1947 (a French team repeated this independently in 1948). Bell Labs' Shockley, Bardeen, and Brattain won the Nobel Prize in Physics in 1956 for their work.

Fundamentally, transistors control and direct the flow of electricity. They act as switches (sending current one way or another, or switching it off), and they can also amplify current, making the output power greater than the input power. The first transistors were made from germanium, which when pure is an insulator, but when slightly impure becomes a semiconductor, which is what is needed for a transistor to work. The amount of impurity must be tightly controlled to create the correct effect. Germanium transistors were very quickly replaced by junction transistors, which are more robust and easier to make.

Transistors are an essential part of nearly all modern electronics, although most modern transistors are silicon rather than germanium. The fact that they can be easily mass-produced at low cost (more than ten million transistors can be made per US cent) has been a crucial part of the development of mass modern technology. These days they are usually part of an integrated circuit rather than wired together as with early transistor computers, but they're still at the core of all practical electronics. Estimates of transistors made per year vary between about half a billion and a billion per person on the Earth, and those numbers are still going up.


Ann Moffat worked with Ferranti on the Manchester Atlas from 1962, and in 1966 was one of the earliest teleworkers – here seen writing programs to analyse Concorde's black box, with her daughter. Copyright Rutherford Appleton Laboratory and the Science and Technology Facilities Council (STFC). **www.chilton-computing.org.uk**.

The Atlas machine room at Chilton in 1967. Copyright Rutherford Appleton Laboratory and the Science and Technology Facilities Council (STFC). **www.chilton-computing.org.uk**.

track of memory locations by hand).

- It had demand paging: the address translator would automatically load a required page of data into main memory when it was required.
- It had an algorithm which identified the currently least-required pages and moved them back into secondary memory.

Fundamentally, this is still what virtual memory does today, and it does, as expected, make programming massively more straightforward. It's also vital for running multiple programs at the same time, allowing the OS to swap parts of jobs in and out of memory as they are required.

Emulator

An Atlas simulator is available from the Institute for Computing Systems Architecture (University of Edinburgh – **www.icsa.inf.ed.ac.uk/research/groups/hase/models/atlas/index.html**). You can download their three sample programs from their website. This is a simulator rather than an emulator, in that it simulates the operation of the Atlas architecture by modelling its internal state, but doesn't pretend to give the experience of operating the whole machine.

To run the simulator, you'll first need to download and install HASE III. There are detailed instructions **www.icsa.inf.ed.ac.uk/research/groups/hase/models/use.html**, but basically you download the **jar** file, then type:

```
java -jar Setup_HASE_3.5.jar
```

at a terminal window. Run as root to be able to install for all users of the machine, or as a user to install for just that user. You can then run the **bin/Hase** executable from wherever you installed the program.

To run one of the Atlas projects, download one of the samples and unzip it, choose Open Project from the HASE menu, then choose the relevant **.edl** file. So for Atlas_V1.3, the project that demonstrates each of the various instructions, choose **V1.3/atlas_v1.3.edl**.

To compile the project, first, if you installed HASE as root, you'll need to make sure that the user as which you're running has write access to **hase/hase-iii/lib**. (This seems only to be necessary for the first compile.) Next, go to Project > Properties > Compiler, and make sure that the **Hase** directory is set correctly to where you installed HASE. Finally, hit the Compile and Build buttons on the menu bar.

Having compiled the code, you can run it (with the green running person icon), then load the tracefile back into the simulator and watch it run (use the clock icon, and choose the relevant results file). Run it to watch changes happen in the simulated construction. You can also watch the pipelining happen, and the virtual pages being requested and loaded.

If you want to look at the program instructions themselves, they are found in the **DRUM_STORE.pageX.mem** files in the **model** directory, starting with page 0. They are structured as:

```
instruction Ba Bm address
```

The Drum Store contains the program code in page 0, fixed-point integers in page 2, and floating-point reals in page 3. The Core Store is empty at the start of the simulation, with Block 0 modelled as an instruction array, Block 1 as an integer array, and Block 2 as a floating-point array. As each array of code/integers/floating-point number is needed, it is fetched in from the Drum Store.

For an explanation of the instructions used in each model, check out the HASE Atlas simulation webpage (**www.icsa.inf.ed.ac.uk/cgi-bin/hase/atlas.pl?menu.html,atlas.html)**, which also has more details of the simulation itself. The listing of the first demonstration program doubles as a list of Atlas instructions.

You can also try the other demonstration programs, both of which do actual mathematics. V3.2 is a Sum of Squares program, which should report in the Output window the result 3, 4, 5, (then print **stopping**). This is the solution of the equation **a2 + b2 = c 2 for a, b, c < 8**. We couldn't find any output for the Matrix Multiplication program (updates would be welcome if any readers do experiment with it!). An explanation of the model is at the link above.

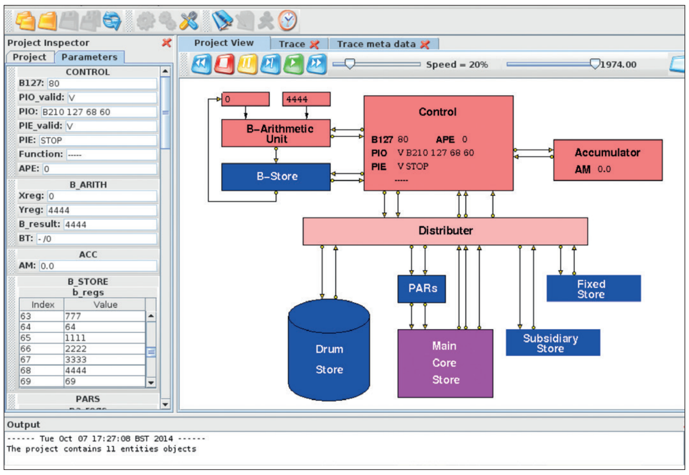
More information about HASE, a user guide, and how to create your own models, is available here: **www.icsa.inf.ed.ac.uk/research/groups/hase/manuals/index.html**.

Building your own program

If you copy the contents of one of the sample directories wholesale into another directory, and rename **atlas_v*.*** to **my_project.*** (so you're renaming the **.edl .elf .params** files), you can edit the **DRUM_STORE.page0.mem** file to produce your own small program. Here's one example:

```
A314 0 0 12288
A320 0 0 12296
A346 0 0 12304
STOP
nop 0 0 0 ... to end of file (must be 256 lines)
```

This loads the value in word 1536 into the accumulator (**A314**), adds the value in word 1537 to it (**A320**), and then stores the result in word 1538 (**A346**). Words 1536 onwards are found at the start of



CONTROL
B127: 80
PIO_valid: V
PIO: B210 127 68 60
PIE_valid: V
PIE: STOP
Function: -----
APE: 0

B_ARITH
Xreg: 0
Yreg: 4444
B_result: 4444
BT: - /0

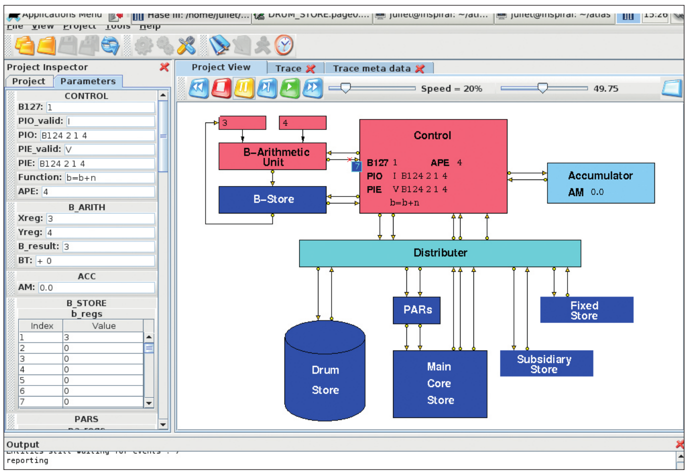
ACC
AM: 0.0

B_STORE
b_regs

Index	Value
63	777
64	64
65	1111
66	2222
67	3333
68	4444
69	69

PARS

Output
```
------ Tue Oct 07 17:27:08 BST 2014 ------
The project contains 11 entities objects
```



CONTROL
B127: 1
PIO_valid: I
PIO: B124 2 1 4
PIE_valid: V
PIE: B124 2 1 4
Function: b=b+n
APE: 4

B_ARITH
Xreg: 3
Yreg: 4
B_result: 3
BT: + 0

ACC
AM: 0.0

B_STORE
b_regs

Index	Value
1	3
2	0
3	0
4	0
5	0
6	0
7	0

PARS

Output
```
reporting
```

The simulator after running the v1.3 model. The blue fast-forward button runs the whole thing as fast as possible. The green button allows you to watch more slowly, or you can step through one process at a time.

DRUM_STORE.page3.mem, and are loaded into the core store block 2 when needed.

To run it, load the project, compile it, run, and then you can watch the trace. If editing, reload the project, then recompile.

Here's the same example (adding two numbers) in B-instructions:

```
B121 1 0 3
B124 2 1 4
E1064 0 0 0
E1067 2 0 0
STOP
```

This loads the value 3 (not a memory address) into B1 in line 0 (**B121**), then in line 1 adds 4 modified by the contents of B1 to B2:

```
B124 B-register B-modification-register Number
```

So in practice this adds 4 + B1 = 4 + 3 = 7 to B2 (which starts as zero). Line 2 uses an Extracode instruction:

```
E1064
```

to output a newline, then line 3 uses another Extracode instruction:

```
E1067
```

to output the contents of B2.

You can see the output 7 in the bottom window, and in the main window, the 7 in the process of being returned to Control as part of instruction 1.

As mentioned above, the first test program listing in the Atlas model information (**www.icsa.inf.ed.ac.uk/cgi-bin/hase/atlas.pl?menu.html,atlas.html**) is effectively an instruction listing. The earlier B instructions, for example, access memory locations rather than absolute values. Remember that A instructions managed floating point operations, and B instructions the integer operations. This means that A instructions operate only on the floating-point values (from block 2 of the core store, word 1536 onwards, memory location 12288 onwards), and B instructions only on integers (block 1 of the core store, word 1024 onwards, location 8192 onwards). We're not sure to what extent this exactly mirrors the real setup of Atlas memory and to what extent it is a feature of the organisation of the simulator, but do bear it in mind to avoid getting really frustrated with memory locations that won't load! Two more Atlas machines were built alongside the Manchester one; one shared by BP and the University of London, and one for the Atlas Computer Laboratory in Chilton near Oxford, which provided a shared research computing service to British scientists.

After Atlas

Ferranti also built a similar system, called Titan (aka Atlas 2), for Cambridge University. Its memory was organised a little differently, and it ran a different OS written by the Computer Lab folk at Cambridge. Titans were also delivered to the CAD Centre in Cambridge, and to the Atomic Weapons Establishment at Aldermaston. The Manchester Atlas was decommissioned in 1971, and the last of the other two closed down in 1974. The Chilton Atlas main console was rediscovered earlier this year and is now at the Rutherford Appleton Laboratory in Chilton; National Museums Scotland in Edinburgh also has a couple of its parts. The Titans closed down between 1973 and 1974.

The Atlas team were responsible for the start of numerous concepts (such as pipelining, virtual memory and paging, as well as some of the OS ideas behind Atlas Supervisor) which are still important in modern computing; and, of course, at the time, the machines themselves were of huge research importance. It's rather a shame that it seems largely to have been forgotten in the shadow of other supercomputers such as those made by Cray and by IBM. It was certainly a very successful British project at the time.

In 2012, Google produced a short film remembering the Atlas, which is available online. There's also a collection of links and memories available on the Manchester University website. There's some documentation on the Chiltern Computing website, too, including this brochure from 1967 (**www.chilton-computing.org.uk/acl/literature/acl/p003.htm**).

Juliet Kemp is a scary polymath, and is the author of O'Reilly's *Linux System Administration Recipes*.

LV PRO TIP

There is an emulator (which copies external behaviour) of the whole thing available, but it's Windows-only; see Dik Leatherdale's webpage at **www.dikleatherdale.webspace.virginmedia.com/atlas.html**.