Version 3 – September 2024.

Ferranti Ltd. and minicomputers.

General background.

The electrical engineering company Ferranti Ltd. was founded in 1882 by Sebastian Ziani de Ferranti, an inventor who was born in Liverpool. Ferranti Ltd. soon achieved substantial success in the fields of electrical generation and supply, transformers and electricity meters. This was followed by work on electrical measuring instruments and, in due course, wireless technology and radar.

In 1948, when the government was looking for a company to produce a re-engineered production version of a pioneering digital computer produced at Manchester University, Ferranti was given the contract. This led to the Ferranti Mark I which, when it was delivered in February 1951, was the world's first production computer to have been installed at a users' site. Throughout the 1950s and 1960s Ferranti Ltd. continued to build computers, initially aimed at the scientific and engineering sectors of the market. These mainframe machines are described in other sections of the *Our Computer Heritage* site.

The mainframe Ferranti Computer Department merged with ICT in 1963, which became ICL in 1968 but the process-control activities continued under the Ferranti name at various locations, especially at Bracknell (Berkshire) and Wythenshawe (south Manchester). Ferranti Ltd continued to manufacture smaller computers for industrial process control and for real-time military applications.

This account of Ferranti minicomputers is divided into two sections:

the **Argus range**, produced at Wythenshawe, pages 2 - 10;
the **F1600 and FM1600 series** produced at Bracknell, pages 11 - 21.

The Argus description is based on a paper supplied by John Steele, who worked for Ferranti from June 1963 until about 1969. The F1600/FM1600 description has been written by Simon Lavington, based on information supplied by Peter Niblett and Ellis Thomas – who both worked at Ferranti Bracknell.

Ferranti Argus Computer – Evolution & Architecture

1 Argus - later renamed Argus 200

This system was originally developed prior to 1963 as the launch control computer for the Bloodhound Mark 2 anti-aircraft missile.

The renamed Argus 200 computer was expensive and had limited program and data memory but its architecture made it faster than Argus 100 (see section 2 below).

Technology	Germanium transistor
Construction	Standardised logic assemblies installed into a card cage. The card cages were rack mounted into standard 19-inch racks, 5ft. high[1].

The logic basis for the design was NOR logic gates.

Word length	12 bit data, 24 bit program.
Program store	Ferrite pegs plugged into holes in a tray organised as 64 rows of 24 bits each plus 3(?) parity bits. Trays were installed horizontally into a rack mounted arrangement. The trays contained horizontal loops, one around each row (program instruction), with vertical loops to read the data. Presence of a peg indicated a 1 bit, absence indicated a 0 bit.
Working storage	1024 12 bit words
Internal architecture	Serial/parallel 2 bits at a time

1.1 Initial Commercial process control applications

A personal note

Prior to 1963 an Argus computer was installed at ICI Fleetwood and was used to control a Soda Ash plant. It was widely thought (believed?) in Ferranti that this was the World's first true on-line process control application[2]. It was experimental at the time and ran for a few years until the plant was closed down.

The only other commercial Argus installation that I recall (and I did visit this site during my first 2 weeks at Ferranti while acceptance trials were running) was at West Thurrock power station. Installation took place in 1963. In this case the application was to control the start-up and shut-down of one of the five turbines and to monitor boiler temperatures.

These limited industrial applications led directly to the development of the Process Control Division at Ferranti and the development of a range of computers specifically designed for this purpose.

[1]

2. Ferranti Argus 100, 300, 400 and 500 systems

2.1 Evolution

The Argus 100, 300, 400 and 500 computers were a family starting with the Argus 100. This was a machine developed using the experience gained from the commercial industrial applications of the original Argus computer (renamed as Argus 200) as an industrial process control computer.

Process Control v. Data Processing

Process Control applications, when compared with Data Processing, are characterized by a need to address a large number of external devices. In the 1960s, Data Processing required only paper tape and card readers as input devices and teleprinters and line printers as output devices, together with some form of bulk storages devices such as magnetic drum/disk and tapes. Process Control, however, required these devices plus the ability to directly access a range of plant sensors and controls, such as :-

1. Temperature sensors (thermocouples) read in as voltage through an A/D converter. Many plants had hundreds to thousands of these.

2. Contact closure sensors read in as single-bit inputs. These were often provided as a spare set of contacts on an actuator.

3. Single bit outputs to control relays or light indicator lamps.

4. Analogue outputs to act as inputs into servomechanisms controlling the plant.

For economy a single storage technology was used for both program and data stores. Non-volatile memory was desirable so that the computer would hold program and data in the event of a power failure. Core store was therefore the technology chosen.

It was logical to build on the Argus 200 experience and the same germanium transistor logic boards and card cages developed for the earlier Argus 200 were used for the Argus 100 and 300 computers.

2.2 Argus 100

The 12-bit word length of the original Argus 200 was extended to 24 bits and the first computer in the family, the Argus 100, was implemented as a serial computer. The instruction set used was a subset of 30 instructions derived from the Ferranti Pegasus computer designed a few years earlier. The first production machine went to Jodrell Bank to control its Mark 2 radio telescope in 1963. It is interesting to note that an Argus 400 computer was later supplied to Jodrell Bank to retrofit into its Mark 1 telescope. That computer was last seen in the Jodrell Bank Museum and was personally commissioned by myself.

The Argus 100 computer was (I believe) designed by David Senior and Mike Eyres with assistance from Stan Redshaw.

[1] As far as I recall [2]Perhaps it should be noted that Elliott introduced process control systems in 1959.

Even as the Argus 100 computer was going into production it was realised that it was perhaps a little slow and that an alternative offering was needed. Development of a parallel version was started in 1963/4. This was the Argus 300.

2.3 Argus 100, 300 construction

The basic unit was a plug-in printed circuit board with a gold-plated edge connector. These were known as a 'package', and each was roughly the same size as a single Eurocard. Each card had a grey plastic handle at the front so the card could easily be removed. The circuit cards were single sided and all produced in-house from layout through bare board manufacture to component assembly.

The logic cards plugged into a card cage and used soldered wire connections at the rear to form the interconnections between logic cards in the same card cage. A Ferranti standard pink wire was used throughout. As the insulation was PVC great care was required when making connections, particularly for modifications, as the heat radiated from a soldering iron would melt nearby insulation if one lingered too long!

To enable card cages to be pre-wired on the bench connections which passed between card cages went to wire wrap posts with one row at the top and one at the bottom. These could be laid flat for bench assembly to make access easier.

Inter cage wiring in the racks was made to wire wrap posts which were a static fixture in the rack.

When a card cage was installed in the rack a "U" link wire wrap completed the connection. This made it possible to remove a complete card cage in a few minutes for modification. Depending on where the wires were and how many changes were required a skilled wireman would often choose to make changes in situ without removing the card cage. Removal and reinsertion, although easy, took between 30 minutes to an hour. Removal was quick – a pair of side cutters would be used to snip all the U links. The time consuming task was individually unwrapping each of the wraps from the posts. This was a chore often left to the designer who had asked for the modification! There was a good relationship between the skilled craftsmen and the designers that made such things possible. There was not much evidence of job demarcation.

The logic cards were based on diode transistor technology. By convention logic one was ground and logic zero was -6 volts. This was very logical in the process control world where a "one" could be used with an open collector power driver to switch on an indicator lamp or close a relay. The logic function we used throughout was NOR. As the outputs were single ended it was possible to connect two outputs together producing a logical OR function. Connecting two standard outputs together however reduced the signal driving capability so we had open collector versions of the logic gates to be used for this purpose.

As logic designers we had a choice of configurations. These varied according to the number of inputs into a single gate. From memory there were four outputs and up to 12 inputs. The design was common and the final assembly selected which combination was supported by the circuit board. It was never done but it would have been possible to reconfigure a standard card to a different combination.

Registers were built up with flip-flop packages. These were latched on the clock and I think both phases of input were needed. We had two forms available. A “Triple” which would latch data from 3 different sources depending on which clock signal was used and a “Single” which provided 4 flip-flops on a single card.

The core store electronics was built using the same technology but required a double card cage. Circuit boards containing the core matrix drive circuits and the read amplifiers were fitted at the front. The actual core store matrix was mounted on the back.

The core store was designed as a 6 microsecond access time module.

Argus 100 and 300

Technology	Germanium transistor (Argus 100, 300)
Construction	Standardised logic card assemblies installed into a card cage. Most of these logic cards were identical to those used for the Argus 200 computer. Card cages were rack-mounted into standard 19-inch racks. The main computer used 3-foot high racks with a Formica topped work surface. The control console was mounted on the work surface.
Logic	Design based on NOR logic

Architecture common to models 100, 300:

Word length	24 bit program and data
Storage	6 microsecond core store (Argus 100) organised in modules of 4096 24 bit words. In fact the matrix had 26 bit planes with the intention that the two additional bits would be used for parity. It was however discovered that the reliability of the parity generation/decode was less than the core store and the parity logic, although designed, was never installed. The two additional planes however did provide a useful spare that was used at least once. The Argus 300 had 2 microsecond core store modules.
Working storage	1 to three modules of 4096 * 24 bit bit words or 12 to 36 kilobytes
Internal Architecture	Large I/O address space (0-7777 octal) Memory space 10000-37777 octal 7 Accumulators held in main storage (octal addresses 10001 to 10007)

The technology and construction of the Argus 400 and 500 is covered in detail below.

2.4 Argus 400 – New integrated circuit technology

In parallel with the development of the germanium transistor based components the whole World was meanwhile investigating the benefits of silicon solid-state technology. Ferranti had developed a rather weird Diode/Transistor logic family called Micronor 1. This suffered from too many deficiencies to be useful and a parallel development of a new integrated circuit family, Micronor 2, and a computer to exploit it, the Argus 400, was started in 1963. This development was just starting as I joined Ferranti in June 1963 and the development of the logic design of Argus 400 was my first ever design.

Development of an integrated circuit based computer in 1963 was an exciting adventure. It was evident that single sided circuit boards would not be adequate and that multilayer boards and some means of joining layers together would be needed. The technology for bonding multilayer boards was bought in and the techniques for plating through were developed initially in-house.

Meanwhile the drawing office was developing techniques for laying out multilayer boards and gaining experience. They used tape on Mylar film at 4 times full size for each of the layers. All camera work involved in producing the masters for etching were naturally (being Ferranti) also done in-house.

Various topologies were tried with differing numbers of ICs on each circuit board. Eventually a size of 39 TO5 cans mounted in three rows of 13 was adopted. The Argus 400 required 20 such boards. The boards were interconnected by a backplane. As one of the initial markets was the RAF a rugged design was required and it was decided to use wire wrap U links to connect the individual cards to the backplane. There were two rows of 35 pins making 70 U links in all. The principle was the tried and tested technique used for connecting the Argus 100 card cages to the frame wiring but the wire wraps were considerably smaller.

Core store technology was used again. The core store cycle time was 2 microseconds. This was built in a similar manner to the processor backplane and provision was made, as in the Argus 100, for up to three 4k word store modules to be fitted.

There was a final backplane that was originally planned to contain the specialised I/O equipment required for aircraft use. When it was realised that the hoped-for airborne market was not ready for such a machine this became the operating console interface plus device drivers for interfacing to Process Control I/O equipment.

The 3 to 5 backplane modules (depending on how many core store modules were fitted) were interconnected via a flexible printed circuit so that the whole assembly could be opened out for maintenance. The whole assembly was designed to fit into a standard aircraft instrumentation case.

2.5 Argus 500

The Argus 500 used a parallel computing architecture but used the same integrated circuits as the Argus 400. It used 2 microsecond core store but the architecture was extended to enable memory bank switching to address four times as much memory.

The architecture also included a similar technique for mapping the accumulator addresses providing eight sets of accumulators. This greatly reduced the time taken to process external interrupts.

The Argus 500 was packaged into a larger vertical plug-in module rather than the aircraft case style used by the original Argus 400. The memory banks were also made into a similar form factor and plugged into processor frame. The Argus 400 was then repackaged in this style making the two machines plug compatible.

The I/O interface was identical between the Argus 400 and 500 so a processor upgrade was a simple swap of the processor module.

3 Integrated Circuit Details

3.1 Micronor 1

This was the first integrated circuit family produced by Ferranti. It predates 1963. The conceptual circuit diagram (reproduced from memory) is illustrated in Figure 1 below.

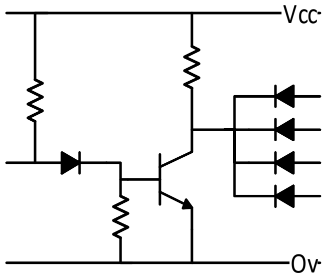


Figure 1 – Micronor 1

It looks unconventional in that the output stage contains the diodes forming the logic gate. These are connected to the input transistor base. This has a major benefit in that it is impossible to connect the output stage to more places than it is capable of driving. This is however the only benefit and the noise immunity of the circuit was poor due to the transistor base being connected to an external pin and to potentially several tracks on the PCB. It would be therefore unsuitable for building a large design such as a computer.

Encapsulation was in a multi-pin (probably 8 pin) TO5 metal can. It was probably designed to run off a 4.5 volt rail but I cannot confirm that. As far as I am aware it was never produced in commercial quantities.

3.2 Micronor 2

Recognising the limitations of Micronor 1, and looking at alternative experimental devices that were appearing in 1963 such as a range of integrated circuits from Fairchild (probably Resistor Transistor Logic) Ferranti decided to embark on the development of a new logic family. This was to be fast, for its day, and have high noise immunity to ease circuit layout. Micronor 2 was born. Conceptually the circuit configuration is shown in Figure 2 below.

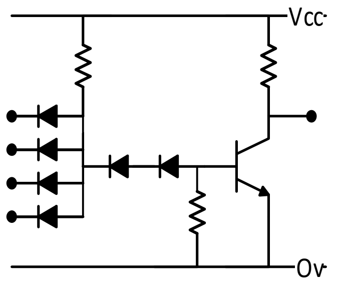


Figure 2 – Micronor 2

The design, in its original form (actually close to that shown above) was designed to drive four loads. An alternative device, a power gate could drive 25 loads. It should be noted that there is some degree of optimisation here. The design was to be used for a 24 bit computer and hence one power gate could drive all the elements of a 24 bit register eliminating any additional problems of skew that could arise from clocks arriving from different sources.

Noise immunity was provided by the two diodes in series feeding the base. The combination provided (from memory) about 1.5 volts of noise immunity on the inputs. Far better than the 74 series TTL from Texas Instruments that came later. The family was designed to run off a nominal voltage of 4.5 volts. The commercial specification devices would run over a range of 4.25 to 4.75 volts. The military versions would run from 4 volts to 5 volts. The temperature range for commercial was 0 to 50 degrees C, for the military versions -55 to 70 degrees C. The typical delay through a gate was better than 7 nanoseconds but one had to be careful about layout to minimise capacitance with the resistive pull-up. The value was 750 ohms from memory.

A JK flip-flop was produced using a number of such circuits cross-connected together internally. This device caused early production problems as it required too high a gain (beta of 25) from the internal transistors. With the silicon purity that was then achievable we were lucky to get one working device from a diffusion of 200 devices on a single wafer. At the time a single JK flip-flop cost 1.5 times that of a graduate engineer's weekly wage.

A eureka moment happened when the designer realised that the diodes closest to the transistor base could be converted into a transistor this forming a Darlington Pair. With a beta of 5 for each transistor, which was readily achievable on an integrated circuit, the required composite beta of 25 was now feasible and yields increased. At the same time the first design of the Argus 400 had been completed and it was realised that a greater fan-out than 4 (3 for JK flip-flops) would significantly reduce the chip count. The increased gain of the Darlington Pair permitted this to be raised to 8 (6 for JK flip-flops) and this is the final specification for the basic IC.

During the Argus 400 design there was also a realisation that one particular grouping of interconnections was occurring repeatedly. See Figure 3 below.

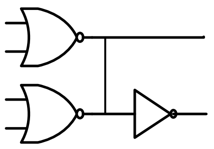


Figure 3 – Small Scale integration

The initial function identified enabled the designer to select one of two sources to feed a JK flip-flop. Both polarities of output were provided to ease this process. A section from the main register of the Argus 400 is illustrated in Figure 4 below. The design opened the register to the core store data when memory was being read. Otherwise the register was connected as a serial register.

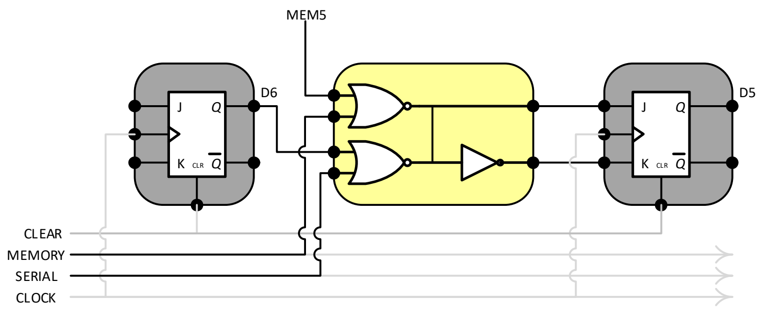


Figure 4 – Section of main register

Much later another use for this SSI device was found. By sequencing reset and set signals this useful device could be turned into a simple latch. In many cases this was more convenient than a JK flip-flop and it was certainly cheaper. This is illustrated in Figure 5 below.

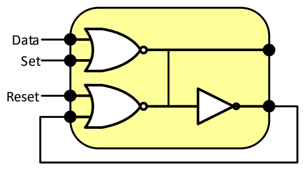


Figure 5 – SSI simple latch

As one of the target markets for the Argus 400 and hence the Micronor 2 family was the military and alternative manufacture was required to second source the devices. Marconi was chosen and did manufacture some devices. Texas Instruments, I was told then approached Ferranti for a license to manufacture and market the devices. Ferranti said no. This led to TI developing 74 series TTL and another marketing opportunity lost. 74 series TTL had significantly worse noise immunity than Ferranti DTL and did nasty things to the power supply when it switched state as a result of its totem pole output stage. It was also slower at nominal 10 nanoseconds compared with Micronor's 7 nanoseconds.

Micronor 2 was almost indestructible. It would survive its output being short-circuited to the Vcc rail permanently without apparent damage. I have received a number of round burn marks on fingers particularly from power gates that were designed to carry 25 loads. In one case it took about 2 weeks of debugging on a new design before an artwork fault was discovered where such an output was directly connected to the 5-volt rail. The aluminium can had by then lost its bright shiny appearance but when the offending track was cut the gate carried on working.

When it became obvious that TI 74 series TTL was taking over the world Ferranti started to manufacture the range under license. It then became more cost effective for us to use the 74 series family but there was a problem – 74 series used a nominal voltage of 5 volts compared with 4.5 volts for Micronor 2. We agonised for a time about this dilemma until the chip designer decided to see just how well Micronor 2 would work at 5 volts. He went to the production line and reset the test criteria upper limit to 5.25 volts from 4.75 volts and ran a large production batch through the test equipment again. There were no failures. The problem was solved by retrospectively changing the voltage specification to be compatible with 74 series. Over a period of time all machines were adjusted to the new range and, as far as I am aware no failures occurred.

The first versions of Micronor 2 were packaged in TO5 8 pin metal cans. Later 14 pin DIL versions were also made.

Development of this IC and pilot production was undertaken at Ferranti in Wythenshawe on a prototype production line. Large scale production eventually moved to Ferranti Gem Mill with Marconi being licensed as a second source. Ferranti suffered from being unable to produce pure enough silicon wafers and were left behind when TI was able to improve the purity and the size of the wafers. Ferranti used the smallest size chips for this reason – it was their only way of obtaining adequate yields.

The chip designer was Peter Bagnall who eventually left Ferranti to work for Motorola.

Document authorship info.:
John Steele, submitted 2000, revised John Steele & D. Bew 4th March 2016.

The Ferranti F1600 and FM1600 descriptions follows overleaf on page 11.

Ferranti F1600 and FM1600 series of computers.

1. Introduction.

The Ferranti F1600 series was first conceived at Ferranti's Bracknell Laboratory at the start of the 1960s, following on from the success of the Hermes and Poseidon computers for ship-borne naval applications. – (see *Our Computer Heritage* 'mainframes': https://www.ourcomputerheritage.org/Maincomp/Fer/ccs-f6x1.pdf etc.). For contextual comments on the close relationship between Admiralty research establishments and Ferranti Bracknell, see reference 5.

After some initial updates and name-changes, it was decided to standardise on the name F1600, with an 'M' added when the use of integrated circuits enabled smaller, faster implementations of the instruction set to be designed. The FM1600B will be taken in these notes as an example of the power of the range and also as one example of Ferranti's entry into the minicomputer arena. As time went by, enhancements were made. The FM1600C, and FM1600D followed. FM1600E installations, which had a 650 nanosecond main store cycle time, were still in service at sea in 2010.

The Ferranti FM1600B minicomputer.

For all FM1600 series, instruction-set compatibility was maintained with the Ferranti Hermes computer. All Hermes/FM1600 instructions could be expressed in FIXPAC, a fixed-point language somewhat similar to Pegasus Autocode. Additionally, FLOPAC was available as a double-word length floating-point Autocode. Higher-level languages, for example Algol, Fortran and Coral, are also implemented on the FM1600 computers.

2. Overall architecture of the FM1600B.

The FM1600B is a 24-bit word length parallel computer, providing both fixed-point and floating-point instructions. The central processor includes 4K words of 1 microsecond core storage; there can be a total of 65K storage in 4K increments connected to the system. Later, 650 nanosecond core storage became available. The central processor, based on Ferranti Micronor II integrated circuits in TO5 cans, is formed of nineteen 6-layer printed-circuit boards which plug into a 12-layer backplane. Input/output is available using up to 22 'Christchurch' Ferranti B channels via separate modular Computer Interrupt Equipment (CIE) – see Section 9 below.

Here are some sample FM1600B instruction times in microseconds:

Operation	*Fixed-point time*	*Floating-point time*
Add/subtract	2.7μsec to 4.3μsec	6.3μsec to 7.0μsec
Multiply	11.3μsec to 13.3μsec	14.0μsec
Divide	13.0μsec to 16.0μsec	15.3μsec
Shifts	2.7μsec to 8.7μsec	-
Jumps	2.0μsec to 5.7μsec	-

3. Instruction format and addressing conventions.

3.1. The 24-bit word is divided into seven fields, thus:

F	I	D	S	A	B	C
3	2	1	3	5	5	5

When an instruction is fetched from memory, the nine {F,I,D,S} bits are inspected and are used to set appropriate bits in a lower-level 12-bit Executive Register (see later). These 12 bits are then used in strict sequence to control the step-by-step execution of the instruction. The actions are somewhat equivalent to microprogram control.

The nine {F,I,D,S} bits give 512 possible combinations, each combination defining the action of a particular FM1600B instruction. Only about 350 combinations are used in practice. The two I bits generally (but not necessarily) give the identity of an index register and the D bit generally (but not necessarily) controls whether an address is to be modified or not. It can be deduced that the {F,I,D,S} bits are coded in a complex manner that relates directly to the underlying logic circuits rather than to a normal programmer's top-level view. For this reason, programmers are usually advised to use the FM1600B Autocode, known as FIXPAC, which is a type of symbolic assembler language – see Section 5 below.

3.2. Addressing.

In general, B and C are each 5-bit addresses of source operands and A is the 5-bit destination address into which the answer of an operation is written. The short (5-bit) A, B and C addresses are often used as offsets to base addresses held in index (ie modifier) registers, the results being 24-bit full-length addresses. The B and C fields can also be used to hold 5-bit two's complement constants (ie literals). There are four 24-bit index registers, known as n0, n1, n2 and n3 but the contents of n0 is treated specially so in effect there are just three active index registers in the conventional sense. Register n3 is also treated specially under certain circumstances. Specifically:

n0 is the program counter (or instruction address register) so this implicitly addresses the next instruction to be obeyed.
n1 is a general address register, but can be altered as a side effect of certain operations on other index registers.
n2 is a general address register.
n3 can be a general address register.
nL is the Link Nest register for subroutine calling and exit. Whether the use of 3 in the I field means n3 or nL is determined by the staticiser Q1 (see below), and this also applies to n3 and Vn3 references in the A,B,C fields.

Direct operations on index registers apart from n3 are not allowed in the A and B fields. However, in [ref. 2] the phrase *modify instructions* is used to describe a set of operations where an offset from the A, B, C field respectively is applied to an index register. Specifically, when the F bits in the {FIDS} op code of an instruction take on the values F = 0,1,2, or 3 and I = 1, 2, or 3 and the {D,S} bits are set to 11, 12, or 13 then an offset from the A, B, C field respectively is applied to the relevant index register in I. These instructions are useful in processing tables of data.

As well as the obvious uses for Vn1, Vn2, and Vn3, use of Vn0 provides access to words (constants) embedded between instructions. Instructions containing Vn0 cause an extra increment of n0, so the constant is not itself obeyed as an instruction!

3.3. Special addresses.
The meaning of the 32 values as used in the A/B/C fields are given special interpretations, some of which are conditional on the state of control bits not covered in these brief notes. The original FM1600 manuals used the following notations:
Vx means ‘the value held in store address x’;
nx means ‘index register x’;
n means ‘a store address’, eg as held in an index register.
nL denotes a Push-Down (LIFO) stack pointer, implementing a *Link Nest* store.
Using these notations, and indicating whether the A, B or C address-fields of an instruction are being used to reference each address, a more detailed interpretation of the first 32 store locations is as follows:

Addr	***A***	***B***	***C***
0	V0 (unstored)	V0 = 0	V0 = -1 or zero (Note a)
1 - 23	V1 – V23	V1 – V23	V1 – V23
24	V24 (tape punch)	Index register n0, (= zero)	V24 (handswitches)
25	V25 (output devices)	Index register n1	V25 (tape reader)
26	V26 (output devices)	Index register n2	V26 (input devices)
27	Index reg. N3 or stack pointer NL (note b)	Index register n3 or stack pointer NL (note b)	V27 (input devices)
28	VN0	VN0	VN0
29	VN1	VN1	VN1
30	VN2	VN2	VN2
31	VN3	VN3	VN3

Note (a) and Note (b): some detailed actions are defined by control bits (eg Staticisers), some of which are described in Section 3.4. See references 1, 2 and 3, given at the end, for further information.

The actual significance of the first locations of core store is that earlier F1600 series machines used memory addresses 1-23 to store the registers V1 to V23 (which therefore operated at the speed to the core store). It is believed that index registers were always fast registers, so memory addresses 24-27 were never used for n1-n3.

3.4. Staticisers (control bits).

The FM1600B contains control bits that can be tested, set, and cleared by certain instructions. Of these, Staticisers Q1, Q2, and Q4 have special meanings. Q1 has already been mentioned above.

Q2 becomes set when an interrupt is handled, and further interrupts are delayed until it becomes cleared. It can be directly set and cleared for use in interrupt handlers, and restored by return to link.

Q4 becomes set when overflow occurs in various arithmetic operations.

Q1, Q2 and Q4 are treated specially by the link-storing jump instruction and the return to link instruction (see below). Their current values are stored in the top bits of the word used to store the return link address (as only 16 bits are used for the address), and they are restored on return to link. Q1 is set and Q4 cleared at subroutine entry; Q1 thus enables the link nest to be used to save one or more V registers, to be restored on subroutine exit. (Both actions need explicit coding, such as: Vn3 = V1, n3-1 and V1=Vn3, n3+1). Some care is needed that the Link Nest is placed in a large enough store area to accommodate all links and saved registers if subroutine calls became deeply nested. The starting value for nL needs to be selected to cater for all planned decrements in using this stack. There was no hardware detection of stack overflow.

When an interrupt occurs, the effect of a link storing jump is made, saving the Q2 staticiser and setting it at entry to the interrupt handler. By software convention, register V2 was permitted for use in handlers, without saving and restoring, so could not be relied on for use in normal code.

4. Overview of the FM1600B's instruction set.

In this section we give just a general guide to what is in reality a very complex, but rich, repertoire of orders. See references 1, 2 and 3 for further information.

By way of reminder, the 24-bit word is divided into seven fields:

F	I	D	S	A	B	C
3	2	1	3	5	5	5

4.1. Arithmetic and logical instructions.

Single-length fixed-point arithmetical and logical instructions are provided. The general form is as follows, where: <function> can be: **+** or **–** or **&** or **Non-equivalent to (exclusive or)**:

VA' := VB <function> VC

Double-length multiply and divide instructions are catered for. Also, there are double-length instructions where a multiplier is a specified power of two; this is in effect an arithmetical shift. Here's an example:

$V(A, A+1) := V(B, B+1) \times 2^{VC}$

Single-length floating-point arithmetic instructions are available, using 24-bit words with 6-bit exponent and 18-bit mantissa.

4.2. Shift instructions.

There is a range of single-length and double-length shifts, both left and right, logically or circularly. Here are two examples:

VA := VB shifted left logically VC places
V(A, A+1) := V(B, B+1) shifted left logically VC places

There are several normalise instructions, of which this is an example:

Normalise VB. VA := no. of shifts. V(A+1) := normal form.

Many shift operations used hardware that shifts multiple places in a single clock beat, so a shift of more places do not necessarily take longer. This in turn improved the speed of some of the operations for floating point, including shifts to standardise the results of calculated values.

4.3. Control transfers instructions.

There is a total of 96 control transfer orders, during which the {F, I, D, S} bits in an instruction can take on the following values: F can be 0 -> 5, whilst I = 0 and D can be 0 or 1 and S can be 0 -> 7. So 6 x 1 x 2 x 8 = 96 combinations in all.
Both absolute and relative jumps are catered for. In addition, there are three special control transfer instructions, described later.

The general formats for the 96 jump instructions are:
Absolute jump to <address> if <operand 1> <function> <operand 2> <condition>;
Relative jump A places if <operand 1> <function> <operand 2> <condition>.
where: <function> can be **+** or **–** or **&** or **≠**
and <condition> can be **≥ 0** or **< 0** or **≠ 0**

Some illustrative examples are:
Jump to VA if VB <fn> VC <cond>
Relative jump A places if VB <fn> VC <cond>
Jump to VA if VB <fn> C <cond>
Relative jump A places if VB <fn> C <cond>
VB := VB <fn> VC; then Jump to VA if VB <fn> VC <cond>
VB := VB <fn> VC; then Relative jump A places if VB <fn> VC <cond>

Another set of control transfer instructions test single bits, which could be set (ie = 1) or clear (ie = 0). Examples are:
Jump to VA if bit VC of VB is clear
Jump to VA if bit VC of VB is set
Relative jump to VA if bit VC of VB is clear
Relative jump to VA if bit VC of VB is set
Jump to VA if bit VC of VB is clear; then set the bit
Jump to VA if bit VC of VB is set; then clear the bit
Relative jump to VA if bit VC of VB is clear; then set the bit
Relative jump to VA if bit VC of VB is set; then clear the bit

A similar set of control transfer instructions test the specific staticisers (ie control bits) QC and QVC.

4.4. Instructions which manipulate index registers and the Link register.
In the following example instructions, <function> can be **+, -, &, NEQ**:
N1 := VA; Vn1 := VB <function> VC … and similar.
Vα := VB <function> VC; N1 := N1 + 1; VA := Vα … and similar.
There are also instructions such as:
N1 := N1 + A; VN1 := VB + VC … and similar.
N1 := N2 + A; Vn1 := VB NEQ VC … and similar.

Here are the subroutine entry and exit instructions using the Link register and a 16-bit jump address ‘JA’. The 16 bits of ‘JA’ comprise one bit from the ‘S’ field of an instruction plus 15 bits from the three A, B and C five-bit fields.

NL := NL – 1; VNL := 0; N0 := JA; store Q1, Q2, Q4; set Q1; clear Q4.
N0 := VNL; NL := NL + 1; restore Q1, Q2, Q4.

4.5. Special instructions.
There is a Sideways Add (or population count) instruction:

VA := number of ones in VB

Some other special instructions cause a ‘wait’. See references 1, 2 and 3 for further information.

5. FIXPAC.

5.1. General observations. Programming in FIXPAC provided a convenient translation mechanism between the complicated encoding of the {F,I,D,S} bits in machine-level instructions and clearer Autocode-like instructions. For example, FIXPAC filled in relative or absolute addresses of symbolic labels for jumps and conditional jumps. FIXPAC programs could be made clearer by adding text to the right of a FIXPAC statement in single quotes ('). The FIXPAC compiler could also accept instructions given directly in the {FIDS A B C} machine-level format. This was useful when an operation is required that the compiler itself did not easily offer. It was also useful in providing a bit pattern, rather than expressing the pattern in octal or decimal.

By software convention, register V2 was permitted for use in interrupt handlers, without saving and restoring, so could not be relied on for use in normal code. The FIXPAC compiler was allowed to use V1 when it translated an instruction into a series of {FIDS} equivalent instructions. V1 was otherwise a normal register, but usually treated as a temporary value in calculations rather than holding a value needed for an extended time.

5.2. Subroutines in FIXPAC.
FIXPAC programs were compiled as numbered subroutines, built together by the linker (MISS). The naming convention *Snnn* provided references to the subroutine and acted as a symbolic reference filled in by the linker. A subroutine could be executable code or a data area – but there was nothing to indicate which.

The normal FIXPAC way of calling a subroutine used the right arrow character (→), which was available on some input devices and on the line printer, but later ISO code allocations replaced this by underline (_). Constructs such as
→S27601
called the subroutine using a link storing jump. The normal way to exit back to the link was →L.
It was also possible to use
→SNL27601, which jumped to the subroutine without storing a link. This was sometimes used when the calling code wanted to exit from itself immediately after, so equivalent to:
→S27601
→L
For data area subroutines, the address was obtained by use of ASnnn, meaning "Address of Subroutine nnn".

Some utility subroutines were provided for public use. In particular, S74 was provided to control output to the line printer. Several predefined subroutines such as this one offered multiple entry points for performing specific operations. These were called by FIXPAC by using a format such as:
→S74,3
and the Ferranti documentation provided for the subroutine detailed the actions. Here is a list of the more useful line printer operations provided by various entry points to S74:

S74,0	Output single character from V20.
S74,1	Output multiple characters from an addressed buffer.
S74,2	Output a tab character.
S74,3	Output multiple newlines, count in V20.
S74,4	Reset the printer.
S74,5	Wait for printer to finish.
S74,6	Return a value to show whether the printer was busy.

The numbered 'public' subroutines described above were separate compilations. It was also possible to include local subroutines within FIXPAC programs by making link storing jumps to labels within the same program. For example

```
→30,L
...
[30]
...
→L
```

In a similar way to data area subroutines, a label could be used to provide the address of a local data area embedded within the FIXPAC program. The construct AL20 was used for "Address of Label 20".

5.3. Stacks

The link nest employed by link storing jumps was addressed using nL (which replaced n3 when Q1 was set). This is a LIFO stack of locations with decrementing addresses. It was a common usage to push values to be saved onto this stack, using code such as:
Vn3=V20, n3-1
and later to pop them off the stack using code such as:
V20=Vn3, n3+1
Any index register could be used to manage similar stacks.

5.4. Loops and Tests

The ‘test count and jump’ instruction was used in the common style of loop for a given number of times. The easiest way to loop "n" times was to put "-n" in a register, and count up to zero. The numerical test instructions used ">0" and "<0", but note that ">0" meant greater than or equal to zero. This example loops eight times:

```
V15=-8
[7]
...
→7, V15=V15+1 <0
```

Sometimes such a count and jump instruction was used when the result of the test was "fixed" – for example an index register cannot be negative. Thus it can be used so it never jumps, or always jumps. Remember on the FM1600B the normal A=B+C instructions did not allow A to be an index register. Here are some examples:

```
→42, n1=n1-1 >0          'Decrement n1 and always jump'
→19, n1=n1+24<0          'Increment n1, but never jump'
```

5.5. STOP Instructions

When a STOP instruction was obeyed, it had a number that was shown on the FM1600B control panel. The number displayed could be used by the operator to decide what to do, such as setting a value on the handswitches before continuing. Several different stop codes are available.

5.6. Some sample FIXPAC statements.

Below we give the first few lines of FIXPAC code from an Auto Core Dump utility subroutine. This subroutine provides a diagnostic printout of the state of the machine during the operation of a "parent" program into which it was linked.

This Auto Core Dump utility was given the name S27601, and the title in the original Ferranti documentation describes it as: "AUTO CORE DUMP FROM AS27000 TO AS2. WITH NO STOP ENTRY 1+ 23. 1 73". So the using program would need to be built with S27000 as a data subroutine to mark the start of the area of main store that was of interest. In addition, Auto Core Dump used "S2000 / BLANK AREA FOR REGISTERS" which is listed as a 60-word area (771 to 831); it was used as a working area to store the machine state that was saved on entry.

5.6.1. Overall behaviour of the Core Dump utility.

Referring to the FIXPAC instructions given later, it can be seen that on entry the program goes to considerable pains to preserve the machine state so that it can re-instate it on exit. The data area S2000 is used to capture the following:

V1 to V23
n1, n2, n3, nL
Store locations 128 to 160 (for link nest)
Printer interrupt word.

This allows the program to use these registers freely, and to use a link nest for its own subroutine calls running downwards from 160. While printing store locations, it checks that the address is not within 128 to 160 – and substitutes the saved values if it is.

When that has been done, the utility proceeds with generating the diagnostic output in the form:

Saved registers in octal, eight per line,
Specified area of store eight to a line, skipping blocks of consecutive zeros,
Each line starts with the address in decimal and octal, then the values in
octal.

5.6.2. Initial entry and exit – (see FIXPAC statements below).

Entry 0+ goes directly to label [0], while entry 1+ arranges to record a marker to indicate that it was used, before continuing at label [42] on line 18 as for 0+. Both these paths use {FIDS} code to perform Vn0=n1, saving n1 into the following location of the code, so it can be placed later into AS2000. Entry 1+ picks up the saved n1 value from label [40] on line 3, sets its sign bit (which is never set in n1) and transfers it to label [1] at line 15, to be the saved value of n1 the same as used by entry 0+. The sign bit is set by the code V1[23] £0 on line 9, where £ means ≠ . (This would be achieved by using this machine-code instruction:
4016 0 1 23
even if the bit was already set; this jumps zero places). Entry 1+ reinstates n2 and n1 as needed when continuing from label [42].

The first few lines of the FIXPAC Auto Core Dump utility follow on the next page.

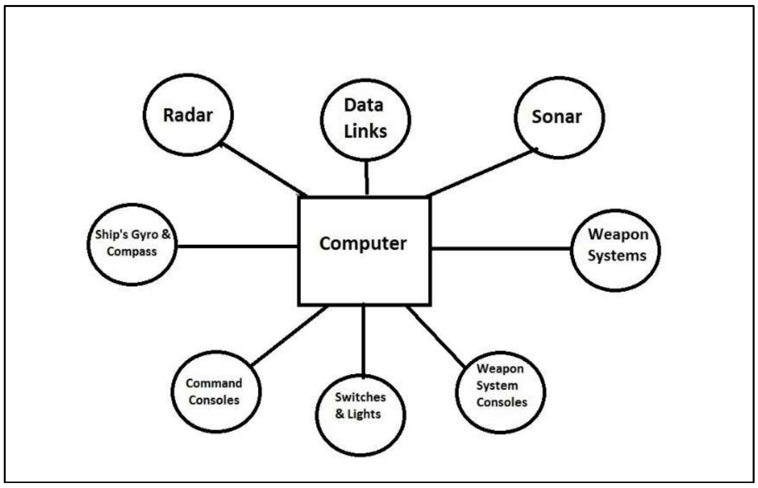
```
   1     →0                         'normal entry'
   2                                'no stop entry'
2            0110   28    25    0   'VN0 = N1'
3      [40]  +0
4            NI = VN0 , VN1 = V1
5            AS2000
6            VN1 = N2 , N1+1
7            N2 = VN0 , V1 = VN2
8            AL40
9            V1[23]  £0             'set marker for entry'
10           N2=VN0 , VN2 = V1
11           AL1
12           N2=VN1
13           →42 , N1 -1 > 0        'reset N1 and jump'
14      [0]  0110  28  25  0        'VN0=N1'
15      [1]  +0
…
…
```

9. Overview of input/output in the FM1600B.

Input/output devices are allowed to interrupt between steps of Executive Register activity. This yields a maximum wait-time for an I/O interrupt of 3 microseconds. Since there may be many independent, asynchronous, peripherals connect to an FM1600B-based system, a priority-based assessment of all outstanding requests is performed rapidly via fast shift instructions.

The FM1600B is designed to work with a companion unit called the *Computer Interrupt Equipment* (CIE). Two CIE versions are available. The smaller one can handle up to 12 Ferranti B interface channels (so-called *Christchurch* channels, named after the Signals Research and Development Establishment at Christchurch, Dorset). The larger CIE can handle up to 22 channels. Each channel can handle a single complex peripheral device (for example a quad magnetic tape system), or a number of simpler peripherals (for example a few tens of teleprinters) multiplexed onto one channel.

On board ship, the flow of practical input/output information covered a variety of real-time devices, as shown in the diagram below, which comes from [ref. 4].



In naval terminology, variations of the above schematic covered Computer Aided Action Information Systems (CAAIS), Action Data Automation (ADA) and Action Data Automation Weapon Systems (ADAWS). A fuller list of abbreviations used for such systems is given in reference 5.

10. Deliveries of FM1600 and FM1600B computers.

The main customers were naval warships, for a variety of on-board *action data processing* systems. Ferranti Bracknell maintained close contact with Admiralty Surface Weapons Establishment (ASWE) Portsmouth and Admiralty Underwater Weapons Establishment (AUWE) Portland Harbour for system specification and personnel training. There were therefore some shore-based F1600 series installations for training personnel and for prototyping ship-based systems.

Ship-board applications included the following types of vessel: Type 12, Type 21, Type 22 and Leander class Frigates, Type 42 and Type 82 destroyers, various aircraft carriers and submarines. FM1600 computers were installed in some of these deliveries (covering 1963 to 1985). Deliveries of FM1600B computers covered the years 1966 to 1990. In total, there were about 88 F1600 and FM1600B ship-born installations between 1963 and 1990. Later FM1600 models such as the FM1600E were still at sea in 2010. The data on ship-board deliveries comes from reference 4. Further practical comments on ship-board deliveries are given in reference 5, together with many relevant photographs.

Of the few non-naval deliveries, one F1600 computer went to the Army at Fort Halstead, Sevenoaks, where it was installed in a trailer, presumably for moving about a battlefield.

Other deliveries to static sites included Wembury, Aberporth and Woomera - all for weapon system test firings. As far as is known, there were no F1600 series installations delivered to the RAF or to civil applications in control and industrial automation.

11. References and acknowledgement.

1. Ferranti FM1600B Microcircuit computer. List DSD/68/6, September 1968, revised October 1968. Digital Systems Department, Bracknell, Berks. See: https://www.sba.unipi.it/sites/default/files/2015_05_29_08_44_132.pdf

2. FM1600B Instruction Code Document. Published by Information Services, Ferranti Digital Systems, Bracknell. 1972.

3. FIDS to FIXPAC. Hand-written document produced by E N Thomas, 9th October 1969.

4. The data on F1600/FB1600B deliveries comes from a talk given by Peter Niblett to the Computer Conservation Society on 17th January 2013. See: https://www.youtube.com/watch?v=I9zLvsa7P40

5. *Post-war AIO and Command systems in the Royal Navy*. Peter Marland. Warship 2016, pages 76 – 98. (2016).

The help of Peter Niblett and Ellis Thomas in preparing this note on the FM1600B is gratefully acknowledged. The finished text has been researched and produced by Simon Lavington in August 2024, who remains responsible for any errors or omissions.