920M EXPLANATIONS

Terry Froggatt, October 2023.

Very Brief Introduction

These notes relate to the 920M digital computers manufactured by the Mobile Computing Division of Elliott Automation (later GEC) at Borehamwood in the late 1960s and through the 1970s. Specifically these notes relate to the MCM7 variant of the 920M.

The 920M is a rugged computer measuring 32cm x 19cm x 19cm, using mostly DTL flatpack logic, derived from the successful 19inch rack-mounted 920B and 903 discrete transistor computers, and using essentially the same instruction set. Both use an 18-bit word and have an internal store capacity of 8192 words.

The 920M is in three hinged sections which open out to form a "Z" or "N". The upper section contains the Ferrite Store, the middle section contains the Registers, and the lower section contains the Control logic. Each section contains a multi-layer motherboard or backplane, although I've used the term "deck", and each deck comprises a Front half-deck and Back half-deck. All pins are uniquely identified by printing on the decks.

At the time of writing in 2023, there are still several 903s in working order, and importantly all of the 920B/903 engineering documentation is available, at several places and now on line. In contrast, hardly any engineering information was in the public domain for the 920Ms, hence these notes and associated documents.

I wrote software to support the Sepecat Jaguar aircraft's flight program development. Initially code was developed on the MCM2 (with a 5usec store & 1usec logic cycle) and was switched to the MCM5 (with a 2usec store & 1usec logic cycle) when it became available. I recently found a note written by me in about 1970, implying that I was then aware of a future MCM7, but saying no more about it.

The observed MCM7 timings match those of the MCM5, and my working hypothesis is that an MCM7 is an MCM5, with the extra store connector and some associated components removed. But my memory is that the MCM2 & MCM5 control & display panels looked the same as those on a 920B/903, although I've found that the MCM7 interfaces would not be directly compatible with them.

Information Sources

I've had the following since I worked at Elliotts, none of which contain any circuit information, beyond a mention of "DTL":

- "Programming Compatibility of 920 Series Computers"
  typewritten note by Peter Lawrence, November 1965.
- "Elliott-Automation 900 Series Small Low-cost Computers"
  yellow sales brochure including photographs
  of a 920M under construction and of an LSA 566.
- "Elliott MCS 920 M Facts" orange card, October 1968,
  which rather complicates the instruction set.
- "Elliott MCS920M Computer" yellow flyer for MCM2.
- "Elliott MCS920M" gold brochure for MCM2.
- "Elliott MCS920M" yellow brochure for MCM5.

My own synopsis of the Elliott 900-Series, written in 2004, is on my website at www.tjfroggatt.plus.com.

In July 2021, Dr Erik Baigar found:

"Technical Information" typewritten notes & diagrams for
the 920M, by the Aviation Service & Repair Division
of Elliott Automation at Rochester Kent, including
a full page "920M CONTROL SEQUENCE (MICROPROGRAM)".
The discovery of a potentially appropriate microprogram is what
persuaded me that I might be able to understand how the 920M worked.
In places it is barely readable, but I've been able to produce a
readable A3 version of it, in UPCHART.PDF.

In February 2022, Dr Andrew Herbert added:
"AP 101B-3101-3A Chapter 71" Jaguar Navigation Attack System
mechanical installation diagrams, of no immediate use, and
"AP 101B-3101-10KB1 Chapter 71" Jaguar Navigation Attack System
electrical installation diagrams, which confirmed that I
had correctly deduced the Application Interface on PLC.

In May 2023, Bob Eager pointed me at:
"MCS 920C Computer Specification, January 1968", which confirmed
some of my findings about the control panel interface.

I have had 920M MCM7 serial number 5343 here, one of three MCM7s
purchased by Dr Erik Baigar from eBay, since September 2019.
I have had 920M MCM7 serial number 88 here, one of three MCM7s
from the Rochester Avionic Archives, since Summer 2021.
Both were working during the summer of 2020,
neither are working at the time of writing in 2023,
but it has still been possible to deduce their wiring,
and to check some aspects of them by powering them up.

I've regularly reported my progress in the Computer Conservation
Society's "Resurrection" journal, available on line at
www.computerconservationsociety.org/resurrection.htm.

I've produced the following documents which effectively
define the control & register decks of the 920M MCM7:
What the Logic Sub-Assemblies (LSAs) on these decks do,
see LOGICSUB.PDF, also CONJECTU.PDF;
Where the LSAs are on these two decks,
see CON-DECK.PDF and REG-DECK.PDF;
The wiring within all three decks,
see WIRING.PDF;
The external interfaces, and the interface with the store,
see INTERFAC.PDF.

WIRING.PDF is a searchable text database, showing all pins
connected to a given signal. It was derived from MCM7 5343 and
was checked against MCM7 88. It includes every LSA on all three
decks (control, register, & store), and it also notes where
signals cross the half-deck bridges or the inter-deck hinges, and
it gives some information about power and marginal test wiring.
(I believe I have covered all important pins, but I've not covered
"pin 19" on LSAs J1 to P1 or most of rows T1/T2/Z1/Z2, in the
store. These might be related to the 920C "test forcing signals").

From this I've derived the "dual" representation, namely,
conventional circuit diagrams, showing all signals connected
to a given LSA or gate: in CON-CIRC.PDF and REG-CIRC.PDF.
These do not include the store deck, or the bridge & hinge data,
or direct links between connectors not otherwise connected.
In the event of any conflict, WIRING.PDF is more trustworthy
both than the notes in this file and than the circuit diagrams.

The signal names used in INTERFAC.PDF are locally meaningful and
are relatively wordy, whereas the signal names used in WIRING.PDF
are generally shorter, more cryptic, and are globally unique.
The signal names used in CON-CIRC.PDF and REG-CIRC.PDF are almost all
derived automatically from WIRING.PDF and so are the same, except
that "~" for "not" in WIRING.PDF becomes an overbar in the Circuits.

Where signal names appear in CON-DECK.PDF and REG-DECK.PDF, they are often further abbreviated to fit.
The hash sign "#" represents "not equal".

The file names on my website are case-sensitive and need to be in upper case as shown above.
The files are all searchable PDF.

920M changes relative to 920B/903

Some instructions on various machines in the 900-Series corrupt the Q-register. On 920B, function 9 (jump if negative) corrupts the Q-register, but on the 920M function 9 leaves the Q-register unchanged. This is a minor improvement.
(Function 7, jump if zero, corrupts the Q-register on both, and function 8, unconditional jump, corrupts the Q-register on neither).

On both the 920B & 920M, the Block Transfer instruction uses the A-register & Q-register to define the store start address & the length of the transfer, and on both, Block Output leaves the last word transferred in the A-register. The A-register on Block Input is left unchanged on 920B, but is set to the last word transferred on 920M. On both input & output, the Q-register is left unchanged on 920B but is altered on 920M. This is a minor degradation, making it wise to make no assumptions about the final values of the two registers.

Whilst there is a "Block Transfer" signal on both the 920B & 920M, I've not found the 920B's related "Last Word" signal on the 920M.

On 920B, shifts in either direction can go to 2047 places.
After shifting 35 places, the A-register & Q-register will correctly be either all 0s or all 1s (although it would be more accurate to detect shifts over 36 places and set the result to zero by software), but longer shifts can be usefully used for delays.
On 920M, the Facts Card limits shifts to 36 places. In fact, shifts of up to 63 places will give correct results, longer shifts can give incorrect results, and shifts for delays are only linear in time up to 47 places. I believe that this significant limit on shift distances prompted a change to the floating-point package QF.

By inspecting the single sheet 920M microprogram summary (above), and by looking at the wiring of 920M 5343 here, I spotted a glitch in Function 3, and Dr Erik Baigar ran a test program for me on his working 920M, which confirmed this. In the Facts Cards for the 903 and for the 920M, and on an actual 920B/903, Function 3, "Store Q", is defined to read the Q-register, shift this right one place, and store it in the designated location.
On the 920M, it shifts the Q-register right one place, stores it in the designated location, then shifts the register back: the least-significant bit is not recovered from the "Q00" latch and so is lost. (It would be possible to correctly recover Q00 into Q01 for left shifts only in Function 3, just as A01 is copied into Q18 for right shifts except in Function 3 by D66/07 & D66/14: there are some spare gates).
Function 3 is often used after a multiplication, to save the lower 17 bits of the sign-and-34-bits result, where the right shift is useful. This lost bit is actually defined, in "Programming Compatibility of 920 Series Computers" to be the sign bit of the A-register before the multiply (although the 920ATC, where it is the sign bit of the other operand, got it wrong). But the Compatibility document, which includes the 920M, and the Facts Cards, make no reference to losing this bit from the Q-register itself.
Does it matter? Well, losing a bit of the Q-register unexpectedly certainly would, and in Airborne Computing Division we used interrupt code which preserved all 18 bits of A and Q, rather than the code shown in the Facts Cards. I've also used Function 3

to halve something, or to clear a word of store without losing the A-register. And it is perfectly legal to hold six characters in six-bit code in A and Q. But presumably nobody has ever felt the need to use Function 3 in a situation where they knowingly also have useful information in that bottom bit.

Dr Baigar has shown that this glitch is not present in his later AMD-based 920MEs. We cannot tell whether this is because the glitch was spotted and corrected, or it was never spotted and the 920ME was simply implemented from the specification.

(The hand-written note by Function 3 on the original microprogram diagram looks something like "Trigger Read" & "Trigger Write" so is unrelated).

In addition to the above differences which are visible to the programmer, there are two major differences in implementation.

Firstly the 920B has a 12-bit Process Counter and the 920M does not.

On the 920B the Process Counter is used to count the words in a Block Transfer whereas on the 920M the Q-register is used (and hence corrupted) to hold the negated final store address. On the 920B the Process Counter is used to count the 18 steps of Multiply or Divide, and to count the number steps in a shift. The 920B has a 3-bit microprogram counter (AA1,AA2,AA3) and a Control-v-Function flag.

On the 920M, the microprogram counter is extended to six bits, with values 0 to 7 for Control and 8 to 15 for most Functions, and with values 16 to 48 reserved to perform these counts (hence the limitation on shifts).

Secondly the 920B has a G-register, which is used to latch the results of the Adder or Collator, and inputs from the PTS or IFU, prior to writing them to the intended register (A,Q,M,J), so potentially overwriting the inputs to the Adder or Collator. On the 920M, each of the LSAs implementing the registers contains a two-stage latch (so each effectively has its own G-register).

An important consequence of this is that the 920M can perform a one-step left or right shift of the A-register (via the Adder as on 920B), and on the Q-register (via local wiring) at the same time. This enables the Multiply, Divide, and Shift loops to go twice as fast as they might otherwise do (and as the later 920ATC does in some cases) and it simplifies transferring bits between Q18 & A01.

(Another consequence is that Q shifted is not available, without shifting Q itself, but this is no excuse for the Function 3 problem).

Notes about the Interfaces

The pin-outs of the external plugs & sockets and of the internal store interface are given in my INTERFAC.PDF.

Some there are some 61-way connectors, and some 91-way connectors where only the first 82 pins are populated. Usefully, the relationship between the connector's pins and its four rows of deck pins is the same for all connectors, with the last connector pin (61 or 82) usually (but not always) connected to the chassis and not connected to its deck pin.

Power supply interface on PLA

The 920M MCM7 uses four power supplies, their nominal voltages are: -5v +5v +12v +18v.

There are two +5v supply rails inside the 920M, both must be connected:
+5v(store) supplies the store and all transmitters (type 573 LSAs)
+5v(logic) supplies the rest of the control & register decks.

There are also +5v(aux) and +5v(aux), related to LSAs at G52,G53,G55-57, which must be connected.

Some pins are designated "(sense)". These look like the pins in the Jaguar wiring, which are used

to provide voltage feedback to the computer power supply in the aircraft, (either to regulate the power supply or to force a shutdown if things go awry?).
There is a substantial load, especially on the +5v and ground wiring, warranting thick &/or short connections.

In my own home-built power supply I have connected:

```
   -5v:  4 pins: 25 28 55 56
   Gnd: 10 pins: 01 48 57 58 59 71 72 73 74 75
   +5v:  6 pins: 15 23 42 43 66 67
   12v:  2 pins: 60 61
   18v:  3 pins: 33 35 62
```

and I use +5v(sense) pin 47 to supply power to my TTL interface circuitry at the same voltage as seen by the 920M.

Righly or wrongly, we have been using pin 21, called ~PowerGood, as a means of stopping, resetting, & starting the 920Ms.

The +5v(logic) on single pins PLB/52 PLC/48 SKTH/43 indicates that CPU power is on, and perhaps powers a relay in the relevant device. It is not intended to power the device itself. In contrast, +5v is supplied on several pins to the control panel, and 10v is supplied on several pins (between -5v & +5v) to the display unit.

External store interface on PLB

Although PLB and some associated LSAs are not present in an MCM7, the wiring is all present. The receivers for the 18 data inputs from the extra store are present within type 585 LSAs and are known to be inverting, confirming that the "data from store" is not inverted.

I've assumed that the 18 data & 16 address outputs to the store would be provided by 17 type 573 LSAs each containing two similar inverting transmitters, in which case "data to store" is not inverted, but the address outputs are inverted. (Non-inverted address outputs could have been obtained by connecting the transmitter input to the inverse rather than direct J-register bit: without needing a different LSA type). The actual sense of the lower 13 address bits is unimportant (unless the store is going to be connected elsewhere), but the upper 3 address bits do have to be decoded correctly to select the store.

Regarding the control signals, I've also assumed that:
F40 would be a type 583 dual receiver (for ExtRRpy & ExtWRpy) and
F41 would be a type 573 dual transmitter (for ExtTw & ~ExtIst).
On these assumptions, the External TR & TW are upward pulses (whereas Internal TR & TW are downward), the External Replies are upward (same as Internal), and Ignore Store is down for Ignore (same as Internal).

In CON-DECK.PDF and REG-DECK.PDF, I have included the missing LSAs. But the modules with a struck-through position code are actually dummies in 5343 and 88. Module A61 is dummy even if PLB is present.

Input & Output interfaces on PLC & SKTH

SKTH is the Paper Tape Station interface (PTS),
   most of which has been confirmed by use.
Address bits "Teletype select" (bit 3, adds 4) and
"Status select" (bit 1, adds 1) are provided, thus:

```
   15 2048 Reader Input   15 6144 Punch Output
   15 2049 Status Input   15 6145 Control Output
   15 2052 TTY Input      15 6148 TTY Output
```

The "~PTS Present" pin is grounded by connecting the PTS.

PLC is the Application interface (IFU),
   pin-outs confirmed by AP 101B-3101-10KB1 above.
Whilst there is a "Block Transfer" signal
   there is no "Last Word" signal.

Requests (SIP,SOP,RRQ,PRQ) & Replies (RIP,ROP,RAQ,PAQ)
pulse upwards. Replies must come down promptly.

The reader/punch/input/output Replies have separate receivers
whose outputs are wire-ored together. So the four replies are
interchangeable: In my rig I combine the punch & reader replies.
The four reply pins themselves are not directly connected,
if they were they could not be driven separately by TTL.

Input instructions 15 3072 to 15 4095 appear to cause the
920M to wait for a reply without having raised any request.

Display Unit interface on SKTE & SKTF

My memory from the Jaguar days is that we used the same display unit
on the 920M variants MCM5 & MCM2 as we had been using on the 920B,
However, the 920M MCM7 SKTE & SKTF interface is not
directly compatible with a 920B/903 display unit; for example
the signals are all inverted on the 920B but not on the 920M MCM7.
It is certainly not compatible with the multiplexed display interface
described in "MCS 920C Computer Specification, January 1968".

In Airborne Computing Division we also had an AMU, "Accumulator
Monitor Unit", which sat in the three cables that connect the 920M
and the display unit. This could snapshot the A-register value
of a running program when a given instruction/operand/peripheral
address was used. It may well have included the required inverters.
I cannot remember if we ever used the 920M and display unit without
the AMU, although I do remember that the engineers cursed that the
3.9Kohm resistors in the 920B had been changed to 10Kohm in the 920M.

There are five -5volt pins and five +5volt pins but only four
ground pins. I would imagine that the +/-5volt pins provides the
10volts for the inverter which drives the neons, as on the 920B,
whereas the ground pins provide the reference for the signals.

As usual the final pin 61 of SKTF is Chassis, but
unusually the final pin 61 of SKTE carries data.

Control Panel interface on SKTG

My memory from the Jaguar days is that we used the same control panel
unit on the 920M variants MCM5 & MCM2 as we had been using on the 920B,
as shown in a photograph on page 246 of Wireless World May 1970.
However, the 920M MCM7 SKTG interface is not
directly compatible with a 920B/903 control panel;
in fact it looks closer to the 920C/905 control panel
described in "MCS 920C Computer Specification, January 1968".

The control panel of a 903 is marked "Elliott 903", and the
electrically identical panel of a 920B is marked either
generically "Elliott 920" or specifically "Elliott 920B",
I cannot remember which, (but there are some in the Science Museum
collection at Wroughton). And I cannot remember whether the 920M
control panel which we used was marked "920" or "920B" or "920M".

So there are three possibilities:

- The SKTG interface has changed,
  the MCM2 & MCM5 control panel is electrically the
  same as a 920B/903 control panel (however marked),
  whereas the MCM7 uses a 920C/905 style panel.
- The SKTG interfaces for all MCMs are the same, but for Jaguar
  development we used a control panel which was electrically
  the same as a 920B/903 control panel (however marked),
  with an interface box not visible to the programmers.
- The SKTG interfaces for all MCMs are the same, but for Jaguar

```
      development we used a control panel which looked like a
      920B/903 control panel but was electrically different.

On the 920B/903, there is a Stop latch, set by (for example) the
   Stop button and cleared by the Restart button on the control panel,
   and there is a Reset latch, set by the Reset button
   and cleared by the Jump button on the control panel.
   The state of both latches is shown on control panel lamps.
In the 920M MCM5, there is a Stop latch, but I've not found a
   Reset latch, although there are outputs to drive both lamps.

I've found that SKTG pin 33 performs the "Reset Key" function. It
   is pulled low by a CPU resistor, and whilst it is held high by the
   control panel, it turns the Reset lamp on, but it does not latch.
   Either there is a latch in the control panel, or the Reset and
   Jump buttons are replaced by a single Reset/Jump toggle switch.
I've labelled SKTG pin 08 the "~Jump Key". It floats high and
   when it is pulled low it inhibits the reset function of pin 33.
   This suggests that earlier 920Ms did have the Reset latch.
   (On 920C, pin 33 is "JUMP S.S" and pin 85 is "RESET S.S.").

When no control panel is attached, all 18 word generator inputs
   float high, and so appear as the instruction "/15 8191". But
   pin 25, "Pull F down", and pin 27, "Pull N down" also float high.
   These drive gates whose outputs pull the word generator down to
   "8 8177" if no PTS is attached or to "8 8181" if it is attached.
If power is applied to the 920M, with no control panel attached,
   it will start at one of these addresses.

SKTG/43 Panel to CPU: grounded for Cycle Repeat.
SKTG/37 Panel to CPU: grounded for Cycle Stop.
SKTG/15 Panel to CPU: grounded for Order Stop.
SKTG/32 Panel to CPU: grounded for Enter Number Generator.
SKTG/44 Panel to CPU: grounded for Obey Number Generator.
SKTG/38 Panel to CPU: pulled low by a CPU resistor,
                      pulsed upwards for Restart.
SKTG/16 CPU to Panel: goes low if either SKTG/32 or SKTG/44 is low.
SKTG/67 CPU to Panel: pulsed upwards when the CPU executes a jump function.

On the 920C interface, pin 26 is the "Ready" lamp,
   CPU to Panel, and is normally high.
On the 920M, pin SKTG/26 is normally low, being the inverse of a
   signal also present at U1/23 U1/31 U1/32 PLA/37 PLA/54 PLB/22.
```

Internal store interface on U1 & U2

```
U2/08, CPU to Store: Suppress Internal Store (SIS), as on 920B/903.
   This tells the internal store to ignore all Read & Write requests
   whenever any of J14,J15,J16 is high. On all 920M MCM7s seen,
   SIS is strapped low by a pink link between E40/02 & E40/01.

U2/31, CPU to Store: ~Read Request (~IntTR), pulses downwards.
U2/44, Store to CPU: Read Reply (IntRRpy), pulses upwards.
U2/42, CPU to Store: ~Write Request (~IntTW), pulses downwards.
U1/50, Store to CPU: Write Reply (IntWRpy), pulses upwards.

There is no "Store To M-Register" gating signal in the microprogram.
   Rather: data from the Store to the CPU is strobed at the correct
   time within the Read cycle within the store circuitry. But:
U2/30, Store to CPU: ~Strobe Initial Instructions (~Strobe), pulses downwards.
   The store strobe signal is sent to the CPU and gated with "J within
   Initial Instructions", to form "Initial Instructions to M-Register".

U2/43, CPU to Store: ~Ignore Store Output (~IntIst).
   If this is low during a Read cycle. the read cycle is performed
   but the store output is discarded, so forming a Clear cycle,
   (as is required before writing new data from the M-register).
   This signal is also pulled low by "J within Initial Instructions".
```

Notes about the Microprogram diagram

The "920M CONTROL SEQUENCE (MICROPROGRAM)" on a single page
(page 16 of 24 in "Technical Information") was very hard to read,
but based on my knowledge of the 920B/903 microcode,
I was able to understand a good many blocks of the 920M microcode.
Next I found the gates in the MCM7 5343 hardware which implemented
those blocks, which gave me a fuller understanding of some LSAs.
It was then possible to work the other way, from the hardware,
to work out what the rest of the microprogram diagram said.
It does appear to be appropriate to the MCM7 variant of the 920M,
for example it correctly shows the error in Function 3.

I've assumed that the term which often occurs before TR is "IST"
(rather than "TST") and it Inhibits or Ignores the STore output.
The hard-to-read two-character waveform which appears with ETJ in
B-Modification and Set B-Register is "K1" (as on page 15 of 24
in "Technical Information" and as on 920B). This sets J1 and so
selects the B-Register rather than the Sequence Control Register.

The microprogram diagram consists of blocks,
each of which represents a step of the microprogram;
each block is split into a left part then a right part,
which I've called "Phase1" and "Phase2".
The left part of each block shows the gating waveforms which route
data into the adder or into the input side of a register latch.
The right part of each block shows the waveforms which clock data
from the input part to the output part of a register latch,
or which initiate a store transfer.
Before each step, the input part of all bits of all registers are
cleared (including the counter's 551 LSAs but not the counter's
560 LSAs); this is not shown in the microprogram (and there are
no 920B/903-style "0T" waveforms to clear specific registers);
but sometimes registers are sneakily cleared by the global
clear and their clock waveform with no gating waveform.
A microprogram step number is written above each box.

The microprogram decodes the 6-bit microprogram Counter
(C-register) and the 4-bit I-register (Instruction
Function Latch), and some other conditionals including C7.
By default the counter is incremented by one between steps,
and it wraps from 63 to 0, it does not carry into C7.
Despite this, the step number in microprogram diagram is
formed by adding 64 to the C-register to represent C7 true.

Counter values 0 to 7 are "Control", shared by all functions;
thereafter most functions only use counter values 8 to 10;
multiply, divide, & shift also use values 16 to 63;
11 to 14 are only used by (single or block) input and output.

The STOP waveform in steps 01or65 and 11or75 does not stop the CPU;
it merely indicates points where the microprogram is prepared
to stop if asked to do so by ORDER STOP on the control panel.

If the M-register bit 18 is zero in control step 4,
the counter steps to 5 and C2(=2) is explicitly set, causing
a jump to step 7, so as to bypass instruction modification.

For functions 0,1,2,4,6,12,13, which require an operand from store,
step 7 performs a store cycle to read this, to reduce the number
of inputs on MtF FtJ TR TW by six. The step consumes a logic cycle
even when no operand is required, probably because the I-register
is not loaded soon enough (in step 4) to test for and skip it.

Most instructions finish with a CLEARC1-6 waveform,
but function 0, load B-register, also sets C1(=1) and so
bypasses control step 0, which services interrupts.

Multiply, divide, and shifts finish when the counter wraps to zero.

In Multiply (function 12) step 9, the counter steps to 10, and
   C3(=4) & C6(=32) are explicitly set, causing a jump to step 46, where
   18 multiply steps (46-63) are obeyed. (It's not clear why step 8 does
   not include SUPPRESSCOUNT,C2,C3,C6, enabling step 9 to be omitted).
In Divide (function 13) step 9, the counter does not step to 10, but
   C3(=4) & C6(=32) are explicitly set, causing a jump to step 45, where
   18 division steps (45-62) are obeyed, then a final transfer step.

I was puzzled by the 2tQ waveform in the Division Algorithm.
The 920B & 920M both start by shifting Q right then left to clear Q1.
In the loop, if the two tested bits are the same:
   the 920B/903 performs QtF 1tF LtG GtQ,
      so the 1tF get shifted into Q2 and Q1 remains zero,
   the 920M performs 2tQ QSL:
      QSL will not set Q2 to one because Q1 is zero throughout,
      but 2tQ will do so explicitly via inverse input on C08/06.
So the algorithms are the same.
The waveform "F18 TO something" appears twice in the left parts of
   division steps, with a corresponding "something" in the right parts.
   This could be an "X" (as on 920B), but I've represented it as a
   "*" in my microprogram diagram and I call it "star" elsewhere.

In Shift (function 14, M12=M13) step 9, the negated shift distance
   is written into the counter (by CLEARC1-6 to clear all 6 bits, not shown
   on the microprogram diagram, and by JtC which just sets the required 1s).
But if this would set the counter in the range 1 to 15 (steps reserved for
   control or other functions) then 16 is added (see circuit diagram)
   which reduces the shift distance by 16 but still gives the correct
   result (of all 0s or all 1s).
If the shift distance is zero then the 16 is not added
   and the instruction correctly exits to step 0.
The "24to63" appearing above two microprogram steps should be "16to63".
The shift direction is retained throughout in M13.

Block Transfer (function 14, M12#M13) uses the M-register for transfers,
   so the transfer direction in M13 is recorded in the bistable C7.
The only other use of C7 is to distinguish the first pass
   through the control sequence from subsequent passes.
In Block Transfer step 14, the counter steps to 15, and
   C3(=4) is explicitly reset, causing a jump back to step 11.
The Block Transfer signal to PLC/68 is turned on by C=12or76 and is
   turned off by CLEARC7, although neither is clear from the diagram.

Input/Output (function 15) step 9 does nothing. It is not clear
why steps 10 & 11 are not numbered 9 & 10. It is unlikely to be
to reduce the number loads on the step 9 decode. It is possibly
to allow the Peripheral Address to settle, (Block Transfer has
two nonempty steps between setting and using the address).

I deciphered Input/Output (function 15) step 10 by
   reference to page 17 of 24 in "Technical Information".
   "Z1" is described as the "select/reply latch", and
   "Z2" is described as the "clock suppression latch".
From the conditional (Z1,P11,P12,P13), "RCP" must be
   "Program Terminate", although it is not clear what RCP stands for.

There are two unrelated meanings of "D":
   DtF: to gate the -1 input to the adders,
   DtE: to set the program level, given the interrupts.

Notes about the Circuit diagrams

On the circuit diagrams, I have printed the signal name for
all inputs from other pages and all outputs to other pages.
If a signal is generated and used locally within the same page,
sometimes I join them with a line and omit the signal name,
sometimes I join them with a line and give the signal name
at one or both ends, and sometimes (for example in Initial
Instructions) I omit the line but give the signal name at both ends.
Where "Reset" or "Phase1" or "Phase2" signals occur more than once
on a given page, I usually don't attempt to join them with lines.

I appreciate that it is sometimes hard to spot that a signal which
is named and is used locally might also be used on another page.
For example the register diagrams do not explicitly show that outputs
A18 M18 M13 M12 Q01 are also used on the OTHER CONDITIONALS page.

The circuit diagrams do not explicitly show the sharing of inputs on
type 561 LSAs, even when two gates are near one another on the diagram.
So for example ~Fn12.exit and ~Fn13.exit, which each have a C=63 input,
appear to have a redundant C6=1 input because it shared with another gate.

The split between the circuit diagrams in CON-CIRC and REG-CIRC
is not quite the same as the split between the Control Deck and
Circuit Deck themselves. For example the I-Register and Peripheral
Address Register, and some inverters, all on the Register Deck,
are shown in CON-CIRC.

REG-CIRC has one page per bit of the 18-bit word, and shows the
A,Q,J,M registers, and the Adder and Collate circuit for that bit,
as well as connections to the internal & external stores, to the
Paper Tape Station, Application Interface, and Word Generator.

Across the 920M Control & Register Decks, there are 8 columns of LSAs,
marked "A" to "H", and in each column there are rows numbered 01 to 70.
Rows 01-04 and 67-70 are used by the connectors, and 33-38 are
used by the bridge between the half-decks, leaving 56 rows for
the hinge connectors between the full-decks and for the LSAs.

For reasons probably related to heat flow, the LSAs in rows 05-19 & 39-51
are inserted the opposite way round to the LSAs in rows 20-32 & 52-66,
and the pin numbering follows this, with pin 01 still usually ground.
The three rows for each of bits 5 and 9 actually straddle a reversal.
As a consequence, the gating signals, which run down the columns
across 9 or 18 bits, might all have to cross over one another.
However, the register latches and the adder LSAs have several
equivalent inputs, which are exploited to minimise the crossovers;
as a result, the pin number for a given gating signal can vary.

A further complication is that there are four flavours of the
LSA which implement carry with lookahead, see CONJECTU.PDF.
These necessarily have some different pin-outs, but even those
aspects of these LSAs which are common to all four flavours (such as
the PTS & IFU inputs) use different pin-outs in different flavours.

Because of these two complications, all 18 bits are different.
Bits 1 to 8 have a J-bit and PTS I/O,
they use each type of carry LSA twice, and within each pair,
bits 1 to 4 use different adder input pins to bits 5 to 8.
Bits 9 to 16 have a J-bit but no PTS.
they use each type of carry LSA twice, and within each pair,
bits 10 to 13 use different adder input pins to the other bits.
Bits 17 & 18 have no J-bit or PTS,
and they use two different carry LSAs.
And these are not the only differences,
for example some WG input bits have pull down gates,
also WG inputs to M toggle differently from adder inputs.

In REG-CIRC, I have drawn lines to show the classical "highway" signals
from the adder and collate unit outputs back round to the registers.
Of course the left-shift & right-shift inputs to the A & Q registers
cannot be drawn as lines because they come from the adjacent pages.

Each register page shows 3 or 4 "Phase1" signals, a total of 70 inputs,
which are driven by 8 buffers: "Phase1a" to "Phase1h". But whereas
gating signals are necessarily connected columnwise, Phase1 signals
are connected in groups bitwise. (For example, "Phase1d" provides:
2 inputs to bit 7, all 4 inputs to bit 8, and 2 inputs to bit 9).
Given that a single LSA type 565 buffer gate output can drive up to 18
loads (for example F25/05 drives 18 AtF inputs) it looks as though
Phase1 & Phase2 inputs to LSA types 551 & 552 each present two loads.

As stated earlier, before each microprogram step, the input
parts of all type 551 & 552 one-bit latches are cleared,
The type 551 & 552 one-bit register latches can take their input
from one of three sources, each having a 2-input gate for data
and gating signals. Where an input is unused, one of these
two signals (but never both) is appropriately grounded.
These input signals may be present before or during the phase1
clearing pulse, as well as necessarily being present after it.
There is an ungated fourth input on 551 pin 6 which is inverted,
so needs no grounding where it is unused. The corresponding
552 pin 11 is always used (for inverted data from the store).

The adder can take one of its inputs from the A or Q register
(gated by AtF or QtF) and the other input from the M register
or its inverse (gated by MtF or NtF) or from the J-register.
Bit 1's carry-in "1TF" can add +1 to either or both sides.

There are two different ways of subtracting +1.
In control step C=3,
MtF DtF FtJ subtracts the +1 which has just been added to the SCR in M,
so DtF must inject -1 (all 1s) into the other (A or Q) side of the adder.
In Function 7, Jump if Zero,
DtF is on the wrong adder side to add -1 to the A-register.
Step 8, the phase2 M without any phase1 ?tM waveforms clears M,
Step 9, NtF adds ~M = -1 (all 1s) and A, so placing A-1 in M,
but will also exit now without a jump if A < 0,
Step 10, exits with a jump if M < 0, namely if A < 1.

The gating of the J-register by JtF is performed outside the adder.
(Moving it inside would need two extra pins, and none are spare.
Presumably J was chosen for eviction because it is only 16 bits.
I think that this leaves an unused gate in each adder which could
provide the inverted output needed by 9 bits: this would need two
flavours of adder, given that there are no pins for dual outputs).
Pin 5 on the LSA type 554 adders is used bidirectionally:
M or its inverse is carried out from the adder into the carry module,
J is carried from the external J-gate into the adder module.

Wherever RtA or LtA occurs, so does QSR or QSL,
but QSR & QSL also appear without RtA & LtA.
The QSR gating waveform shifts:
Q01 into a latch which I've called "Q00", for use in Multiply,
Q02-Q18 into Q01-Q17, and
Adder01 into Q18 if F>=8 (Multiply Divide & Shift),
otherwise Q18 is cleared (Function 3),
(using gates D66/07 & D66/14 to give "ToQreg18").
The RtA gating waveform shifts:
Adder02-Adder18 into A01-A17, and
a signal which I've called "Adder19" into A18, this being generated
from the carry LSA on the same page (although not joined by a line)
by altering the 4-bit carry LSA sequence, replacing the usual
558 by a 559, which gives the required sign-bit regeneration.
The LtA gating waveform shifts:

Adder17-Adder01 into A18-A02, and
Q18 into A01.
The QSL gating waveform shifts:
Q17-Q01 into Q18-Q02, and
Zero into Q01 by grounding C05/07 but allowing the btQ on C05/02.
(As noted above, Function 3 should shift Q00 in).

Two steps in the microprogram for Division show waveform "F18t*"
which clock ~Adder18 into the inverse input pin 6 of latch F42,
which I've variously called "Star" or "Areg19".
Actually waveform AtF is used, which occurs in exactly the
right division steps (and which is harmless in other functions).
~Adder18 is also tested within the Block Transfer loop.
~Adder18 was originally derived from D66/02, but on all 920M MCM7s
seen, this pin has been isolated, and the signal is now taken from
D66/04 via a pink link, which slightly improves the worst-case time.

The type 551 one-bit register latches each have 4 outputs:
direct outputs on pins 10 & 13 and inverse outputs on pins 12 & 14.
The pin 10 & 14 outputs appear about 240nsec before pin 12 & 13 outputs,
possibly on the leading & trailing edges of the Phase2 pulse.
Possibly, only the slower signals can be used around the highway,
whereas it is better to use the faster signals elsewhere.

Specifically in the case of the J-register:
Pin 13 "safe slow direct", all 16 bits, go to the display and to the adder.
Pin 12 "safe slow inverse", bits 1-6, go via inverters on the register deck,
to the microprogram counter, where the direct signal is both used to
load the counter and is inverted again to isolate it for wire-oring.
Pin 10 "extra early direct", goes three ways:
bits 1-4 and 14-16 via some inverters on the register deck,
to initial instructions,
all 16 bits via other inverters on the register deck,
bits 1-13 to internal store inverted & bits 14-16 to nowhere,
all 16 bits via Transmitters on the register deck,
to extra store, presumably inverted.
Pin 14 "extra early inverse", goes two ways:
bits 1-13 via some inverters on the register deck,
to initial instructions,
all 16 bits via other inverters on the register deck,
bits 1-13 to internal store direct & bits 14-16 to nowhere,
Some of the inverters listed above are shown on the circuit pages
where their outputs are used rather than on the register pages,
I've left the unused direct & inverted bit 14-16 inverter outputs
unlabelled; they might be provision for a larger internal store.

Where the waveforms M TR J all occur in the same right half of
a microprogram step, the J is an input to the TR (Read) and the
M is the output, which may be why J's "early" outputs go to store.
(The need to specify J implies that the early outputs are not
available in phase 1; rather that the early & late outputs are
available on the leading & trailing edges of the phase 2 pulse).

On the "INITIAL INSTRUCTIONS" circuit page:
The 509 LSA E47 is shown in three parts: the 2-input & 4-input gates
shown were originally just a type 564 LSA, to which the latch has
been added, probably adding just two more gates: the LSA is now
probably two 2*930. This latch is not displayed on SKTE or SKTF,
indeed the LSA has no such output.
The 1st column outputs provide the 2nd column inputs (suffixed "i"),
the 2nd column outputs provide the 3rd column inputs (~addresses),
but joining them with lines would be counterproductive.
Apart from SIS at E40, none of them are used elsewhere,

On the "MICROPROGRAM COUNTER" circuit page:
There are gates to ensure that shift instructions can
only set the counter to zero or to between 16 and 63.
There are the usual 2 signals from Counter bit 6's LSA 551 to LSA 560,

```
      CtA6 from H30/13 to H29/06 and ~CtA6 from H30/12 to H29/09 (just like
      the other five counter bits), but there are also some extra signals:
         uninverted CtA6 to H21/02 (calculation of C#63), and
         inverted ~CtA6 to H24/11 giving uninverted CtA6a on
         H24/14 to G20/09 (~Fn13.sub) & G20/13 (~Fn13.add).
      I'm not sure why these are used in preference to
         the C6=0 and C6=1 outputs from LSA 560 H29.
         or why H24 was used to get CtA6a, given CtA6.
   I have drawn lines connecting the "~Clear6" signal, but to
      avoid clutter I've not connected: Phase1 Phase2 JtC Reset.
   JtC is used by twice by C5 on H28, but once by the other bits,
      here seven gates are driven by a non-buffer gate.
   The 551 LSAs are all cleared by the Phase1 signal, so it
      follows that the 560 LSAs must remember the counter value,
      confirmed by the ~~Reset signal, which clears them initially.
   CYCLE REPEAT is implemented by grounding SKTG/43,
      which, like the SUPPRESSCOUNT waveform, changes
      the carry-in to the 560 LSA chain from 1 to 0.

As on the 920B/903, the 2 upper bits of the Instruction Function
   are fully decoded and the 2 lower bits of the Instruction
   Function are fully decoded, see "I-REGISTER & FUNCTION DECODE".
   It is not clear why Irem4#3 is calculated twice, by G40/05 & G42/02.
Also fully decoded are counter states 8 to 10, see "COUNTER DECODE & C7".
So the individual steps of most instructions can be decoded
   by a 3-input gate, typically a type 561 LSA with shared inputs.
A 561 also has one 4-input gate, sometimes used to test a conditional
   (for example, output G16/04 has the same inputs as G16/02 plus A18).

The maximum number of inputs seen on any gate is 14, on a
   564 or 565 4-input gate plus all 10 inputs of a 566 expander.
Some steps have been taken in the microcode to achieve this.
As described earlier, for functions 0,1,2,4,6,12,13, which require
   an operand from store, step 7 performs a store cycle to read this,
   which reduces the number of inputs on MtF FtJ TR TW by 6.
There are two further optimisations which cannot be deduced
   from the microprogram diagram.
For functions 0 to 7: step 8, the decode ~Fn<8.08 on G11/02
   generates an MtF, reducing the number of MtF inputs by 7.
For functions 0 1 8 9: step 8: the decode ~Fn0189.08 on G11/04
   generates Clear6, reducing the number of Clear6 inputs by 3.

On the two "MICROPROGRAM DECODE" pages:
   The waveform representing a microprogram step goes LOW
   when the appropriate Function & Counter decode inputs,
   and any "OTHER CONDITIONALS" used, are ALL HIGH.
On the two "GATING SIGNALS" pages (and elsewhere):
   A primary gating signal (like FtA LtA RtA) goes HIGH if ANY
      of the microprogram step waveforms which use it is LOW.
   A secondary gating signal (like btA or ttA) goes HIGH if Phase2
      is high and any corresponding primary signal is HIGH.
The purpose of the ~~Z2 input to ~Fn15.10getPTS on page 2 is
   explained below the "INPUT OUTPUT & TERMINATE" page below.

The page "GATING SIGNALS 1 of 2" shows gating inputs to
   the A-register Q-register & J-register, the Peripheral
   Address register (P), the Instruction register (I), the
   microprogram counter (C), and the program level latch (E),
The P & I registers use LSA 553 single-stage latches, so
   MtP & MtI are only used to generate their clocks. tP & tI.
"jcA" (my name) is "Just Clear A", used in Multiply step 8.

On the page "GATING SIGNALS 2 of 2":
   the lower part relates to the gating the Adder inputs, and
   the upper part relates to the M-register and to store control.
Although the microprogram diagram contains terms like TR and TW, the
microprogram hardware uses four different signals, which I've named:
   IstTrTW: a read/write cycle, transferring M to Store,
   MTrTw:   a read/write cycle, transferring Store to M,
   WxR:     Write without Read,
   RxW:     Read without Write.
"jcM" (my name) is "Just Clear M", used in Jump-if-Zero step 8.

I only partially understand the "READ WRITE & CLEAR" circuit page.
The general scheme is clear enough:
   uTr from the GATING SIGNALS sends a TR to the store(s),
      and sets the "Read" latch which is cleared by their reply,
   uTw from the GATING SIGNALS sends a TW to the store(s),
      and sets the "Write" latch which is cleared by their reply,
   both of these send a signal "To Timing", to hold the CPU,
   ~IstPh2 from the GATING SIGNALS sets the "Ignore Store"
      latch, which is cleared at the end of cycle.
Less clear is:
   why another latch which I've called "Pending Write" is also required,
   why there appear to be further read & write latches, driving display
      outputs, built around the -75 LSA on the "RESET & TIMING" page,
   and the overall sequence of signals between the two pages.

I only partially understand the "RESET & TIMING" circuit page.
   <?> means I'm not sure whether this pin is an input or output.
This page includes the three wide LSAs.
   Each occupies three rows, but uses only two rows of 13 pins.
      LSA type "-G?" at G55-57 has pins in G55 & G56
      LSA type "-75" at H44-46 has pins in H45 & H46
      LSA type "-76" at H48-50 has pins in H49 & H50
   I've printed "H46" & "H50" and the associated pin numbers
      in italic; none of the pins in row G55 are used.
   The normal ground pin 1 and +5v pin 15 are not available;
      on -75 & -76, both pin 14s and a pin 2 are used respectively,
      as I've shown, (the reverse of what might be expected).
This page also shows all uses of the +5Vaux & -5Vaux rails,
   to the LSAs at G52 & G53, and to the wide LSA -G? above.
The two F54 LSAs are related to the marginal testing of LSAs
   -75 & -76, each of which has 6 adjustment potentiometers.
CYCLE STOP is implemented by grounding SKTG/37, or by a low
   on ~CycleStopA from the Reset circuitry, but neither take
   effect until CycleStopC goes high within the timing cycle.
   (It is not clear whether this can happen between
   Control steps 1 TR & 2 TW or Function 10 steps 8 TR & 9 TW;
   if it can, the store contents could get lost).
ORDER STOP is implemented by grounding SKTG/15, but it does
   not take effect until a step marked STOP in the microprogram
   is reached, and PhaseC goes high within the timing cycle.
The Stopped latch can be cleared by a pulse generated by H45
   from a high on RESTART at SKTG/38, which is normally low;
   it is also cleared by Reset, although the CPU will
   not run until the Reset signal itself is removed.
The SKTG/33 RESETK and SKTG/08 ~JUMPK inputs are covered
   under "Control Panel interface on SKTG" above.

On the "PERIPHERAL ADDRESS REGISTER" circuit page:
One half of the 2-bit latch D31 is unused.
Although I've named some type 553 LSA two-bit latches pins
   as in & ~in, and out & ~out, for some bits the direct and
   inverse signals (for both input & output) are reversed.
I've shown that named pins "Preg11 Preg12 Preg13 go to
   I/O request routing", but although I have also named
   pins ~Preg12 and ~Preg13, they are not used anywhere.

On the "INPUT OUTPUT & TERMINATE" circuit page:
It is not clear why G61/02 is required here to invert Preg13,
   given that there is an unused ~Preg13 at D46/02 (as above).
I've called this signal ~Preg13a,
   but I've not drawn lines to its two uses on this page,
   just as I've not drawn lines to other Preg or ~Preg uses.
As said earlier, I deciphered Input/Output (function 15) step 10
```

```
   by reference to page 17 of 24 in "Technical Information", as:
      PTA1 (notM12)   |   A (notM13)
      PTA2 (M12)      |
      SUPPRESS        |
      COUNT (notZ2)   |
      SZ1             |
      RCP (Z1,P11     |
      P12,P13)        |
This implies that either PTA1 (IFUtA) or PTA2 (PTStA) is gated into
   the input side of the A-register according to M12, but this only
   gated through to the output side if M13 indicates an input.
But it is not wired like that. IFUtA or PTStA themselves take M13
   into account, and both cause "tA" in the second phase as usual.
Terminate (RPC, 15 7168) is treated much like an output, but one
   which generates an immediate reply. The "Term" signal goes to
   three INTERRUPT LOGIC gates, and by a pink wire to the II latch.
For each input & output, initially, bistables Z1 and Z2 are clear.
   Functions 14 & 15 initiate an I/O operation using "SZ1"
   which sets Z1 and raises one of the four request signals.
   The same microprogram steps also use "SUPPRESSCOUNT if not Z2",
   so the step is repeated until Z2 is set by a reply. The next
   microstep always includes "0TZ" which clears both Z1 & Z2.
For block transfer, this sequence is repeated; the 0TZ in step
   14or78 being harmless when it follows the 0TZ in step 77.

When data is read from the Application Interface (PLC), by Function
   14 or 15, whatever data is on PLC is repeatedly copied into the
   A-register until Z2 is set, and the final value is retained.
When data is read from the Paper Tape Station (SKTH), by Function
   15, a different approach is used, because each use of the PTStA
   waveform shifts the A-register 7 places. This explains the ~~Z2
   input to ~Fn15.10getPTS on page 2 of the MICROPROGRAM DECODE:
   reader data is only transferred, once, when the reply sets Z2.
There is an aspect of I/O which I've not understood.
   The inputs to SZ1 (F32/02) & SUPPRESSCOUNTif~Z2 (F23/02) are:
      Fn14bt.12+C7      Fn14bt.12+C7-Z2
      Fn14bt.13-C7      Fn14bt.13-C7-Z2
      Fn15.10getIFU     Fn13.09 which is irrelevant here
      Fn15.10-Z2        Fn15.10-Z2
   Note that the last signal in both columns is the same.
   Why does SZ1 have an input which depends on Z2? And why does
      IFU input differ from IFU output, PTS input or PTS output?
   If this Z2 is deleted then the last two terms in the left column
      reduce to just Fn15.10, as might have been expected.
   Conversely, given that Z2 is initially zero, why not add Z2
      to the other terms in the left column, saving some gates?
```

Partial M-register Operations

```
A 920B/903, and a 920M, can have up to 8*8K store modules,
   and "Module bits" 14-16 are used to address them.
A 920C/905 can have up to 16*8K store modules,
   and "Module bits" 14-17 are used to address them.

To form the operand address, the microcode has to take only
  bits 1-13 of the instruction, and splice in the Module bits
  (except, on 920C/905, in absolute mode for a non-jump).
It does not matter what the remaining high bit(s) are set to.
If the instruction is modified, the 18-bit contents of the
   current level B-register are added later.

In Function 11, the microcode has to save the SCR thus:
   bits 1-13 into the operand, setting the other bits to zero,
   module bits into the Q-register, setting the other bits to zero.

On the 920B/903,
   To form the operand address,
      getting bits 1-13 is achieved by omitting DTF2, and
      the module bits 14-16 are gated by MtI, sheet A19.
   In function 11,
      getting bits 1-13 is achieved by omitting DTF2, and
      getting bits 14-18 is achieved by omitting DTF1.

How is this done on the 920M?
   The gating signal to transfer the M-register input
   into the M-register output is split three ways:
      8btM: the least-significant 8 bits,
            to reduce the fan-out as for other transfer signals,
      3ttM: (or Mpart) the 3 module bits, and
      7ttM: the remaining 7 more-significant bits.
   Generally all three signals are pulsed together,
   but the 3ttM or Mpart signal is also pulsed by:
      control step C=4, which also pulses the I-register transfer:
         step C=1 reads the SCR into M
         step C=2 sets M to M+1 (if not NG) and writes it back
         step C=3 sets J to M-1 and reads instruction at J into M
         step C=4 copies J to M-input,
            but Mpart only transfers the module bits to M-output,
            thus leaving the local operand address in M bits 1-16,
            (and M18 still holds the modifier flag if relevant).
            | JTF FTM CLEARC7 (C2 if M18=0) | Mpart I |
      or by function 11 step C=10:
         step C=8 copies M into Q and reads the SCR into M
         step C=9 copies Q into J
         step C=10 cleverly does the rest with
            | MtFpart FtQ CLEAR6 | Mpart Q Ist TR TW |
            MtFpart gates the 3 module bits via the adder to Q, and
            since M-input is cleared and not loaded here, Mpart will
            clear the 3 module bits of M-output before TW writes it.

Insofar as I can read the original microcode diagram,
   both "Mpart"s look the same, and indeed are wired to the
      same gate, but labelled something shorter than "M14-16",
   the "MTFpart" differs and looks longer, probably "MTF14-16".

On the page "GATING SIGNALS 2 of 2":
   The "~Control.04-NGA" waveform appears three times:
      at F07/05 to generate the FtM waveform,
      at E11/13 to suppress the tM waveforms which normally follows,
      at F19/04 to generate just the 3ttM waveform specifically.
   The "Fn11.10" waveform appears twice:
      at F19/03 to generate just the 3ttM waveform specifically,
         the FtM waveform is not required, and so tM is not generated.
      at E28/10 to generate the MtF14-16 waveform.
   The gates used to generate MtF14-16 are slightly odd.
      LSA E28 is the only type 502 in the 920M,
         which appears to function identically to an LSA type 562.
      The only other LSAs with types starting "50" are 504 & 509,
         both of which occur only once and both of which
         have deck wiring which suggests evolution.
      LSA E05 is a long way from E28, suggesting that the inverter
         used here was pressed into service late in the day.


*** EOF ***
```