OPIC
Office de la propriété intellectuelle du Canada

CIPO
Canadian Intellectual Property Office

(12) (19) (CA) **Brevet-Patent**

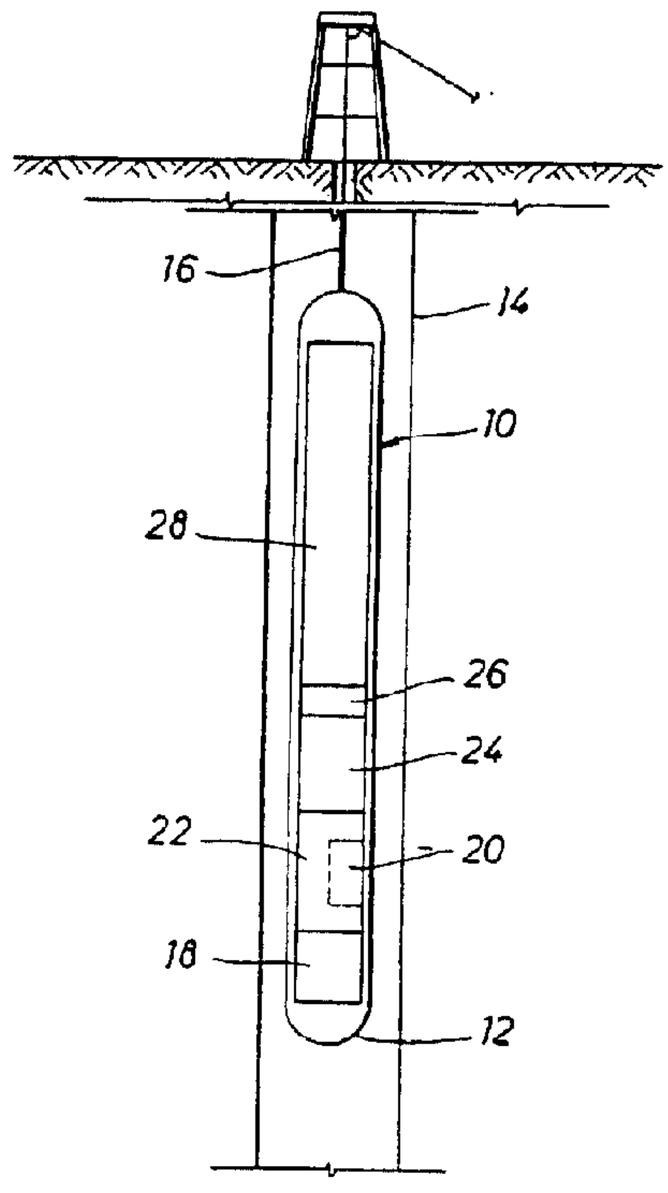
(11) (21) (C) **2,183,123**
(22) 1996/08/12
(43) 1997/05/01
(45) 2000/10/17

(72) Baron, Emilio A., US
(72) Berneking, David J., US
(72) Chisholm, John W., US
(72) Fisher, Marc Kevin, US
(72) Trainor, William F., US
(72) Foreman, James R., US
(73) CORE LABORATORIES, INC., US

(51) Int.Cl.[6] G01V 5/12
(30) 1995/10/30 (08/550,287) US

(54) **OUTIL D'ANALYSE SPECTRALE AUX RAYONS GAMMA POUR LA DIAGRAPHIE**

(54) **GAMMA RAY SPECTRAL TOOL FOR WELL LOGGING**



(57) A gamma ray spectral tool is provided that can be used in slick line systems. The tool includes a battery, a controller, and a memory. In operation, the tool compresses data from the spectral detector for storage in the memory, allowing for long runs without the memory overflowing. Further, a controller latches the self-adjusting stabilization feature of the spectral detector when gamma ray intensity exceeds a predetermined threshold, preventing erroneous "stabilization" by the stabilization feature of the spectral detector.

ABSTRACT OF THE DISCLOSURE

A gamma ray spectral tool is provided that can be used in slick line systems. The tool includes a battery, a controller, and a memory. In operation, the tool compresses data from the spectral detector for storage in the memory, allowing for long runs without the memory overflowing. Further, a controller latches the self-adjusting stabilization feature of the spectral detector when gamma ray intensity exceeds a predetermined threshold, preventing erroneous "stabilization" by the stabilization feature of the spectral detector.

APPLICATION FOR PATENT

Title: GAMMA RAY SPECTRAL TOOL FOR WELL LOGGING

Inventors: EMILIO A. BARON, DAVID J. BERNEKING, JOHN W. CHISHOLM, MARC KEVIN FISHER, WILLIAM F. TRAINOR, and JAMES R. FOREMAN

SPECIFICATION

BACKGROUND OF THE INVENTION

1. Field of the Invention

The invention relates to well logging tools, and more specifically to a gamma ray spectral memory tool.

2. Description of the Related Art

A vast array of tools are used for well logging. These tools, which can measure pressure, temperature, and a wide variety of other parameters, are typically lowered into the well at various points in drilling and production operations to determine conditions down hole and to determine the effect of various drilling procedures.

These well logging tools generally process the data they accumulate in two different ways. In the first technique, the tools are lowered on what is known as a wireline, an electric line that allows bidirectional data communication with the surface and allows power to be provided from the surface. In the second technique, the tool is instead lowered on a slick line or non-electrical cable and the tool must provide its own power and data storage.

Wireline systems allow dynamic monitoring of the well logging tools and eliminate the need for large memories within the tool. Wireline, however, is much more expensive than slick line.

A slick line is typically used in conjunction with a tool that has on-board memory of some sort and need not

communicate with the surface. A battery powered tool is lowered into the well, the tool records the data, and that data is downloaded when the tool is brought to the surface. Although less expensive to use, slick line tools must be capable of operation independent from surface control.

One parameter that well logging tools measure is the presence of gamma rays. Radioactive materials can be injected into the well and then the location of those materials tracked by the gamma ray output of the radioactive materials. Different radioactive materials output gamma rays having different energies. A well logging tool can monitor either overall gamma ray count per unit time, or a gamma ray spectral tool can further determine not only the gamma ray count, but also the energies of those gamma rays. The former technique, which simply detects gamma rays, has been used in conjunction with both wireline and slick line systems.

Spectral gamma ray tools, however, have historically been used only in wireline systems. Examples of such systems are found in United States patent 4,585,939 to Arnold et al. titled "Multi-Function Natural Gamma Ray Logging System" issued April 29, 1986; United States patent 4,857,729 to Gadeken et al. titled "Method of Radioactive Well Logging" issued Agusut 15, 1989; and United States patent 5,410,152 to Gadeken titled "Low-Noise Method for Performing Downhole Well Logging Using Gamma Ray Spectroscopy to Measure Radioactive Tracer Penetration" issued April 25, 1995. Because spectral gamma ray tools divide the gamma ray counts based on their energy spectrum, they provide the capability to distinguish between different radioactive substances in

or near a well and to determine where each is located. Such radioactive substances are used to trace the location of liquids or solids injected into a well. Solid particles used as tracers are described, for example, in U.S. patent 5,182,051.

These systems typically use an americium source as a gamma ray energy reference. Americium generates gamma rays of 60 Kev, so the spectral gamma ray tool can use this reference to determine the energy level of other gamma rays in proportion to that source. Gamma ray emissions from the americium source, as well as from other substances, strike a photomultiplier tube (PMT) in the spectral tool, which in turn outputs voltage pulses that are proportional to the energy of the gamma ray that caused the pulse. Electronics within the spectral gamma ray detector convert this pulse to a digital value and then increment a register corresponding to the energy denoted by that digital value, indicating another pulse has occurred within that portion of the energy spectrum. This is repeated over a period of time, for example 1.6 seconds and then this spectral data is then transmitted over the wireline. Typical instruments divide the gamma ray spectrum into approximately 250 energy channels, although the Halliburton TracerScan tool uses 512 energy channels, with approximately 8 bits of count data per channel. After the accumulated data is transmitted, all the registers are zeroed, and then the counting process repeats.

These gamma ray spectral tools provide a great deal of information, but historically these devices have been restricted to use in wireline systems. First, the photomultiplier tube requires a very high voltage source for its grids (around 1500 V), and has historically

consumed too much power for extended runs on a slick line. Runs can take as long as eight hours, and it was simply impossible to provide battery power for that length of time.

Second, the volume of data generated by these spectral tools is greater than non-spectral gamma ray tools by over two orders of magnitude. Typically, a gamma ray spectral tool is run through a well zone of interest five times at a logging speed of 10 feet per minute, taking up to 8 hours. Assuming around 640 bytes of data is being generated over a 1.6 second period by the tool (1 byte per energy spectrum per second), in 8 hours an extraordinarily large amount of data is generated--over 11 megabytes. These storage requirements have been excessive because memory chips require power as well as space.

Further, spectral tools sometimes encounter areas of a high gamma ray concentration. In such a case, the reference signal from the americium can be masked by those high gamma ray levels. A stabilizer circuit in the tool typically monitors the digitized PMT output in an attempt to locate the 60 Kev gamma ray spike generated by the americium, which would generally be the strongest gamma ray source found. When this spike drifts off of the register designated for 60 Kev gamma rays, the supply voltage to the PMT is adjusted, forcing the americium spike back into the proper energy channel. (Again, the energy of gamma rays from the americium are constant and known.) In this way, the americium spike provides a reference to compensate for any drift in the PMT output. But when excessive gamma ray radiation is present, this americium spike can be masked, causing this automatic stabilization to fail, leading to uncontrollable

misadjustments of the voltage level to the PMT. Gamma ray spectral tools have historically required dynamic monitoring from the surface for these "washout" conditions because otherwise the spectral tool could lose track of the reference signal, possibly causing faulty energy readings of the gamma rays actually being monitored.

For all of these reasons, spectral gamma ray instrumentation has historically been run in wireline systems, rather than slick line systems. This has resulted in increased costs because of the added expense of running wireline as opposed to slick line, which is simple to transport and simple to use.

Therefore, it would be greatly desirable to develop techniques to allow gamma ray spectral tools to be used in slick line systems.

SUMMARY OF THE INVENTION

In a well logging instrument according to the invention, a gamma ray spectral detector tool is run on a slick line rather than a wireline. The gamma ray tool includes a battery, a controller, and a memory. The controller receives digital spectral data from the gamma ray spectral detector, compresses that data, and stores that data in the memory. When the well logging tool is retrieved to the surface, that data is then downloaded to an analysis system.

Further according to the invention, the gamma ray spectral detector includes a log/calibration mode and a latched mode. In the log/calibration mode, a voltage to a photomultiplier tube is dynamically adjusted to compensate for output shifts of the photomultiplier tube as indicated by 60 Kev gamma rays from an americium

source. In the latched mode, the voltage to the photomultiplier tube is held constant. According to the invention, when the digital output of the spectral detector indicates an overall quantity of gamma rays exceeding a predetermined value, the spectral detector is switched from the log/calibration mode to the latched mode.

BRIEF DESCRIPTION OF THE DRAWINGS

A better understanding of the present invention can be obtained when the following detailed description of the preferred embodiment is considered in conjunction with the following drawings, in which:

Figure 1 is a block diagram illustrating the components of the gamma ray spectral tool in the well according to the invention;

Figure 2 is a block diagram illustrating the functional interconnection of the various components of the gamma ray spectral tool according to the invention;

Figures 3A, 3B, 3C and 3D comprise a schematic illustration of the controller circuitry according to the invention;

Figure 4 is a schematic illustration of a bank of static random access memory implemented according to the invention;

Figure 5A is a block diagram illustrating the control of a voltage source to a photomultiplier tube in a spectral detector;

Figures 5B and 5C are spectral energy graphs indicating the effect of high gamma ray concentration upon the self-stabilizing aspect of a spectral detector; and

Figure 6A-6D are flowcharts illustrations of code for execution in an apparatus according to the invention.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

Turning to Figure 1, shown is a basic diagram of a down hole gamma ray spectral instrument 10 according to the invention. Typically, the entire instrument 10 is placed in a 17 foot long case 12, which is lowered into a well 14 on a slick line 16. The instrument 10 includes a gamma ray spectral detector 18, a memory 20, a controller 22, a collar locator (CCL) 24, an RS-232 interface 26, and batteries 28.

The gamma ray spectral detector 18 is one of many gamma ray spectroscopic instruments widely available. Manufacturers include Haliburton Logging Services, Houston, Texas; Atlanta Scientific; and Schlumberger Technology Corporation, Houston, Texas. This device is preferably a TracerScan™ instrument developed by Haliburton. These devices generally use a sodium iodide crystal to capture gamma rays and emit pulses of light in response to those gamma rays. These pulses of light have an intensity proportional to the energy level of the gamma ray (measured in electron volts). The resulting light pulse then strikes a photomultiplier tube, which in turn emits a voltage pulse that is proportional to the energy level of the original gamma ray. This voltage pulse is also dependent on the photomultiplier tube's supply voltage. Further, a photomultiplier tube's gain tends to drift over time because of environmental factors. To compensate for this drift, these instruments typically use an americium source, which is a source of 60 Kev gamma rays, to adjust the voltage to the photomultiplier tube. If the peak in the photomultiplier tube output voltage that corresponds to the gamma rays from the americium source drifts, the supply voltage to the photomultiplier tube is correspondingly adjusted.

This is further discussed below in conjunction with Figure 5A.

These instruments also have various modes, such as a startup mode, a stabilization mode, a logging/calibration mode, and a latched mode. When in the logging/calibration mode, the peak from the americium source is automatically tracked, with the supply voltage to the photomultiplier being adjusted to compensate for drifts. When in the latched mode, the supply voltage to the photomultiplier tube is instead locked, preventing stabilization for drift.

The digitized output of the gamma ray spectral detector 18 is provided to the controller 22. This controller 22 is further discussed below in conjunction with Figure 3. To summarize, it includes an embedded processor, a read only memory, a random access memory, and interface circuitry to allow it to receive the spectral digital data from the gamma ray spectral detector 18, compress that data, and store the compressed data in the memory 20. When the instrument 10 is retrieved to the surface, that data is then downloaded to an analysis system via the RS-232 port 26.

The batteries 28 provide power through a power supply to various components of the instrument 10. The batteries 28 must be sufficiently powerful to last around 8 hours. For that reason, preferably special batteries are used. These are 9 volt lithium or alkaline battery sticks, preferably manufactured by Southwest Electronics and built to specifications. The lithium batteries are used in high temperature applications, preferably capable of operating at temperatures up to 150 degrees centigrade. The alkaline batteries are used in low temperature applications, preferably capable of operating

at up to 80 degrees centigrade. The batteries should be able to source 1.4 amps for 8 hours, and they should be diode protected and overload protected. For added capacity, they may be nearly 3½ feet long. The batteries 28 provide power to all of the components within the instrument 10.

Turning to Figure 2, a block diagram is shown illustrating the logical interconnection of the various components of the instrument 10. The battery 28 provides a 9 volt supply voltage to a power supply 30. The power supply 30 in turn provides +5 volts to the controller 22 and the memory 20, and provides +5 and ±15 volt to the gamma ray spectral detector 18.

The controller 22 also provides control signals to the spectral detector 18, and the spectral detector 18 similarly provides spectral data to the controller 22. The precise format of the spectral data from the spectral detector 18 varies from instrument to instrument, but in essence is a digital representation of the gamma ray pulse count detected per energy spectra. A typical spectral detector 18 will include 512 discrete energy channels, 256 being in a low energy range, and 256 being in a high energy range. Within each channel, an 8-bit registe- will represent the gamma ray pulse count within that energy spectra since the spectral detector 18 started counting. Typically, the spectral detector 18 will reset around every 1.6 seconds, restarting its counting, and then transmitting its spectral data to the controller 22. This reset time may vary, however, depending on the spectral detector's manufacturer.

As will be appreciated, 640 channels (512 of spectral data and 128 in operational data) at 8 bits per channel constitutes around 640 bytes of data every 1.6

seconds. This is a high data rate, corresponding to nearly 11.5 megabytes of data over an 8 hour period.

Depending on the spectral detector 18 used, the data rate, the number of channels, and the exact format of the data may vary. The volume of data, however, is universally large.

The controller 22 also provides control data to the spectral detector 18. A variety of signals are generally provided, but for this discussion, the most important are the controls of the log/calibration circuitry for the photomultiplier tube supply voltage. As previously discussed, the supply voltage to the photomultiplier tube affects the digitized output voltage of the photomultiplier tube as it relates to the energy of a specific gamma ray. To maintain the photomultiplier tube in calibration, that supply voltage is adjusted to keep the 60 Kev americium gamma ray peak located within its corresponding spectral energy range in the spectral data provided to the controller 22. One of the controls from the controller 22 to the spectral detector 18 allows the controller 22 to disable the automatic calibration of the spectral detector 18, latching the supply voltage to its current voltage.

Turning to Figure 3, illustrated is a schematic diagram of the controller 22. The heart of the controller 22 is a microprocessor 100, which is preferably an 80C32 embedded controller by Intel Corporation. A number of standard components provide appropriate clock and power signals to the microprocessor 100, such as a crystal 102, capacitors 104, 106, and 108, and a resistor 110. Power is provided by the 5 volt supply and the chip is grounded. The 80C32 is a well known embedded controller, and its implementation is well

known to those of ordinary skill in the embedded systems arts. A wide variety of other microprocessors could instead be used without in any way detracting from the invention.

The microcontroller 100 provides an interrupt signal INT0 to the spectral detector 18 at 400 Hz. INT0 allows the spectral detector 18 to operate properly. The microcontroller 100 also includes a synchronization input SYNCEN, which is provided by the spectral detector 18 for synchronization purposes. A pair of RS-232 lines are also provided through a port of the microcontroller 100, allowing half duplex RS-232 communications with external devices.

The microcontroller 100 has sixteen address lines and eight data lines. The high order address lines A15/8 correspond to the microcontroller's bidirectional P2.7/2.0 I/O lines. The low order address lines are multiplexed with the data lines, so those address/data lines AD7/0 correspond to the P0.7/0.0 I/O lines. The AD7/0 lines are provided to a transparent latch 112, which is latched by the address latch enable signal ALE from the microcontroller 100. The transparent latch 112 then provides the non-multiplexed low order eight address lines A7/0. The A7/0 lines are used to address and drive a variety of devices, including a decoder 114, an AND gate 116, and a scratch pad random access memory (RAM) 118, and a read only memory (ROM) 120. These lines are also used to address the static random access memory (SRAM) in the system, as is discussed below in conjunction with Figure 4. The AND gate 116 combines address lines A15 and A14 so that when both are asserted, the enable of the decoder 114 is asserted.

The decoder 114 is used to select a number of other hardware features within the controller 22. Thus, the highest portion of the address space is mapped to these physical features. These physical features include SRAM control, control of the spectral detector 18, and reading of data from the spectral detector 18. SRAM control is provided through three addresses (HBYTE, MBYTE, LBYTE) which, when written to by the microcontroller 100, cause the assertion of a set high order address byte signal HBYTE, a set middle order address byte signal MBYTE, and a set low order address byte signal LBYTE. Once the SRAM address is set using these three signals, data is written to or read from the SRAMs by writing to or reading from an SRAM address port ADD_O/P, which causes assertion of an SRAM addressing signal ADD_O/P. The spectral detector 18 is controlled by writing to an address SW corresponding to a switch signal SW, and spectral data is read from the spectral detector 18 via a first-in first-out buffer by reading from an address FIFO, which causes assertion of a first-in first-out buffer access signal FIFO.

In practice, to access the SRAMs, the address for the SRAMs is latched into three latches 122, 124, and 126 by writing the appropriate data to those latches by writing to the addresses HBYTE, MBYTE, and LBYTE. Then, to write data to the SRAMs at the latched address, 8-bit data is written to the address ADD_O/P, which enables the output of an octal transceiver 128, writing data onto an SRAM data bus D7/0. To read data, a read is directed to the address ADD_O/P, which causes a read signal to be sent to the SRAMs, as is discussed below. The SRAMs then drive data onto the SRAM data bus D7/0, and the octal transceiver 128 drives that data onto AD7/0. The

appropriate chip select signals for the SRAMs are generated by two decoders 130 and 132, which select the appropriate SRAM chip based on ADD_O/P being asserted in conjunction with the appropriate high order bits provided by the high order address byte for the SRAMs as driven by the output of the latch 126.

A FIFO 134 is similarly read by reading from the address FIFO. When the FIFO signal goes high, that enables a dual multiplexer 136, which on a read causes the read input R* of the FIFO 134 to go active, or low. The FIFO 134 then provides data to the microprocessor 100 over AD7/0.

A signal from the P1.0 output of the microcontroller 100 selects a FIFO test mode for the FIFO 134 and its related circuitry. When the signal from P1.0 is high, this switches the outputs of a multiplexer 138 to their B inputs. This in turn forces the first output of the multiplexer 138 low, enabling the B output buffers of a transceiver 140 and disabling all output buffers of a transceiver 142. This in effect couples the data bus AD7/0 of the microcontroller 100 to the data inputs of the FIFO 134. Further, the Q1* output of a decoder 136 is provided (by the multiplexer 138) to the write input W* of the FIFO 134. This causes the write output WR* of the microcontroller 100 to be coupled to the write input of the FIFO 134. Thus, when the P1.0 output of the microcontroller 100 is low, data can be written into the FIFO 134 from the microcontroller 100 via the data bus AD7/0. The capability of writing data to the FIFO is used for diagnostic purposes.

When the select input of the multiplexer 138 is instead driven high, the circuitry is in its operating mode. The outputs of the multiplexer 138 are then

provided as a switch enable signal SWENBL and a detector writing signal DETWR*, both of which are provided by the spectral detector 18. In this mode, the spectral detector 18 periodically writes data into the FIFO 134 when DETWR* goes low and SWENBL is low. The Q2* output of the decoder 144 is then forced low, so the transceiver then drives the data on the spectral detector 18 bus DBR7/0 into the FIFO 134.

When SWENBL is true, then the data latched into the octal latch 140 (the spectral detector 18 mode data) is written out to the spectral detector 18 over DBR7/0 through the transceiver 142. The octal latch 140 is loaded by writing to the address SW. SWENBL allows the spectral detector 18 to periodically poll for the switch position to determine in which mode to place the stabilization circuitry--latched, stabilizing, or logging/calibration.

Turning to Figure 4, shown is the SRAM circuitry of the memory 120. Six SRAM chips 200-210 are shown. The SRAM chips 200-210 are preferably 512K x 8 static RAMs from White Micro, part no. C50307. This is a specialty part, which is the same as a standard static RAM chip, but allows for operation at up to 200°C.

The SRAM chips 200-210 provide 3 megabytes of memory in the illustrated bank. Preferably, another bank is provided providing a total of 6 megabytes of memory. The SRAMs 200-210 are addressed by SRAM address lines SA18/0. These are provided from the controller 22, illustrated in Figure 3, and as previously discussed, are latched in by appropriate writes to HBYTE, MBYTE, and LBYTE by the microcontroller 100. Once the address is latched, an appropriate chip select signal CS5/0 is enabled by the latch 126 of Figure 3, allowing for a read from or write

to the desired SRAM 200-210 via the data bus D7/0. The remaining chip selects of CS11/6 are routed to the other bank of SRAM, not shown. Finally, appropriate decoupling capacitors 212 are used.

As discussed above, and as further illustrated by the attached software appendix, it is necessary to compress the data from the spectral detector 18 in order to store it all in the SRAMs 200-210, as well as the second bank of SRAMs not shown. This compression routine relies on the fact that a majority of the 512 spectral data energy channels will have values of zero, indicating no gamma rays of that particular energy. Further, the compression routine also relies on the fact that typically, the values within a particular energy spectrum will at least show counts less than 16, allowing for storage within 4 bits of data rather than 8.

Consecutive channel values that are equal to each other are stored as repeat strings. The first byte contains a code indicating a repeat string follows, along with a count of the number of repeat values. The next byte then contains the value that is repeated.

For a sequence of non-equal values less than 16 (i.e., can be stored in 4 bits) the first byte contains a code indicating that a string of 4-bit elements follows, the second byte contains the count of the number of elements stored in the following bytes, and then the elements themselves follow. The elements are stored two per byte, one value per nibble.

For a string of 8-bit values (that is a value greater than 15 and less than 256) each element requires one full byte to store the data. The first byte is again a code indicating a string of 8-bit elements follows, the

next byte is the number of 8-bit elements to follow, and then the actual 8-bit elements follow.

According to the preferred embodiment, if a value is a repeat value and the repeat count is less than or equal to 6, and the data it is adjacent to is a string of non-repeating elements of the same size (i.e., 4-bit elements or 8-bit elements) the repeat values are not be stored as repeat values but will instead be attached to the adjacent string of non-repeating elements, as this is more efficient.

If the value is a string of 4-bit elements and is adjacent to a string of 8-bit elements, and the 4-bit string has less than 6 elements, then the string of 4-bit values are instead treated as 8-bit values and are appended to the 8-bit string.

If a value is a repeat value and its count is less than 12, and it is preceded and followed by strings of non-repeating 4-bit elements, those repeat values will be appended to the 4-bit string as 4-bit elements, rather than treated as a repeat string.

It will be readily apparent how this code is decompressed once the data has been compressed. The decompression routine used in the surface system is further attached in the source code appendix.

Turning to Figure 5, a block diagram is shown illustrating how the spectral detector 18 uses feedback to control the voltage to a photomultiplier tube 300. This is illustrated in conjunction with Figures 5B and 5C, showing a relative number of counts from the photomultiplier tube 300 within particular energy spectrums. Assume that the photomultiplier tube 300 receives a 60 Kev gamma ray from the americium source. The energy level of that particular pulse is then

digitized by a digitizer 302. That digitizer 302 then provides a value to a voltage controller 304 internal to the spectral detector 18, which tracks the peak for the 60 Kev signal illustrated in Figure 5B. But assume, again referring to Figure 5B, that this peak drifts to the left as illustrated. The voltage controller 304 must increase the voltage to the photomultiplier tube 300, forcing the peak in Figure 5B to the right. Therefore, the voltage controller 304 then instructs a PMT voltage supply 306 to increase the voltage to the photomultiplier tube 300.

But turning to Figure 5C, sometimes the gamma ray count will be so great as to saturate many of the energy spectra, or at least to interfere with the peak found at 60 Kev. In that case, the voltage controller 304 may incorrectly adjust the PMT voltage supply 306, attempting to force a phantom peak to the proper 60 Kev energy spectra. In prior art systems, one had to manually monitor for this condition, and when it occurred take appropriate action from the surface. According to the invention, however, when the total gamma ray count in a predetermined range of channels (for example channels 100 to 489) exceeds a certain value for a 1.6 second time interval, the microcontroller 100 then writes a switch position to the switch address SW that corresponds to the latch mode in the spectral detector 18. That position is then latched into the latch 140, which is then written to the spectral detector 18 when SWENBL is next activated.

This latch mode switch position forces the spectral detector 18 into its latched mode. In this mode, the voltage controller 304 in the spectral detector 18 prevent the PMT voltage supply 306 from changing its voltage to the PMT 300. This is permissible for short

periods of time, as the drift of the output of the PMT 300 should not be excessive. Once the gamma ray saturation illustrated in Figure 5C is reduced, the controller 22 writes a different mode value corresponding to log/calibration mode to the latch 140, allowing the voltage controller 304 within the spectral detector 18 to again enter a log/calibration mode. In this way, wild excursions by the PMT voltage supply 306 are prevented when the gamma rays from the americium source stands to be "washed out" by other gamma rays.

The software for operating the tool, including compressing and decompressing data, switching operating modes, and initiating data storage, is attached in the source code appendices. Appendix 1 includes the source code for operating the instrument 10, while Appendix 2 is the source code for the decompression routine. That source code is self explanatory, but Figures 6A-6D serve to further illustrate the flow of that software.

Turning to Figure 6A, a flowchart of a main routine 400 is shown. After initialization, the routine 400 proceeds to step 402, where it checks for user input over the RS232 port 26. This would occur when the instrument 10 was at the surface, either before or after a run, and was connected to the surface system.

If input is available at step 402, control proceeds to a case 404, where appropriate action is taken depending on the type of input. If the command is a tool command relating to the spectral detector 18 itself, control proceeds to step 406, where that command is executed. These commands can include commands to change the spectral detector 18 mode, data transmission commands, status checking commands, and commands to initiate a run into the well. The effect of this last

command is further discussed below in conjunction with Figure 6B. After the command is executed, control loops to step 402.

If at step 404 the input is related to testing, control proceeds to step 408, where the appropriate test is performed. This could include, for example, various hardware testing, SRAM testing, and EEPROM testing. After the testing is complete, control then proceeds to step 402.

Other input at step 402 would be spurious, so control proceeds in that case from step 404 to step 410, where an error is indicated and control again loops to step 402.

If no input is available at step 402, control proceeds to step 412, where the routine 400 checks for a full field of spectral data. Preferably, channel data is stored to alternate fields. When one field is full, data is stored to the other field while the data from the first field is compressed to memory. This is further discussed below in conjunction with Figure 6C.

If a field is full at step 412, control proceeds to step 414, where the data is processed by being compressed to memory. Otherwise from step 412, and in any case from step 414, control proceeds to step 402.

Turning to Figure 6B, a flowchart of further details of initiating a run is shown. Beginning at step 416, a check is made for entry of a delay time, and at 418, a check is made to ensure the instrument 10 is in an appropriate (log/calibrate or latched) mode. If either condition is false, an error is displayed at step 420, and control returns to the main routine 400 at step 402.

Proceeding to step 422, the delay timer is started, and at step 424, the delay timer is counted down until

timeout. This allows for sufficient time to position the tool to begin a data acquisition run. Upon time out, control then proceeds to step 426, where the command mode is set to actually compress and save acquired data. Control then returns to the main routine 400 at step 402.

Turning to Figure 6C, a store routine 450 is shown which reflects the program flow of steps 412 and 414 of Fig. 6A. Control begins at step 452, where data for a full spectrum is read from the FIFO 134 and then written to a field in the RAM 118 at step 454. Control then proceeds to step 456, where it is determined whether a full field of data has been received from the FIFO 134. If not, control loops to step 452; if so, control proceeds to step 458.

At step 458, it is determined to which field the data is in, and a branch is taken reflecting the field. Control proceeds to step 460 if field 2 is full, where a field 2 full flag is set; control proceeds to step 462 if field 1 is full, where a field 1 full flag is set. Control then proceeds to step 464, where a return is executed to the main routine 400.

Turning to Figure 6D, a flowchart illustrates how the data is processed in step 414 of Fig. 6A. A process data routine 500 begins at step 502, where the field full flags are checked. If the second field is full, control proceeds to step 504, where the operating mode is determined (real time or storage). If in real time mode, control proceeds to step 506, where the second field is transmitted over the RS232 port 26. If in storage mode, control instead proceeds to step 508, where the field is compressed and stored in the SRAMs. Otherwise, and from both steps 506 and 508, control proceeds to step 510, where control returns to the main routine 400. Control

proceeds in a similar manner for the first field in steps 512-516.

In an actual drilling operation, the instrument 10 would typically be used as follows. First, the instrument 10 would be programmed for the particular run via the RS-232 port by connection with the surface analysis system. During this stage, the spectral detector 18 would be set in an initialization mode and would be calibrated, setting the zero position of an analog-to-digital converter in the spectral detector 18 so that a zero output of the analog-to-digital converter corresponds to a zero energy value of gamma ray. Further, the instrument 10 would be calibrated such that the 60 Kev gamma rays from the americium source correspond to the appropriate output value of the analog to digital converter within the spectral detector 18. In this way, both the zero and the gain would be properly set in the spectral detector 18.

Further, an appropriate delay would be set by the surface analysis system so that the instrument 10 would not begin acquiring data until it was properly in position. This would prevent unneeded data from being stored in the memory 20. Instead, the delay would be set, the instrument 10 would be disconnected from the surface analysis system, and the instrument 10 would be lowered into the well on a slick line. When it was at the appropriate starting depth, the operator would wait until the delay time had passed, at which point the instrument 10 would start actually storing data in the memory 20, and then the instrument 10 would be periodically lowered and raised for the desired number of passes through the zone of interest.

At this point, it is appropriate to note that not only could slick line be used, but also coiled tubing. Further, slick line could be fed through coiled tubing. The coiled tubing could be of a type, for example, disclosed in U.S. patent 5,121,827 to Legget or U.S. patent 4,984,634 to Pilla. Using such coiled tubing, appropriate tracer elements could be injected into the hole while the instrument was operating. Similarly, the coiled tubing could be lowered down hole with the slick line inside until the end of the coiled tubing was at the top of the zone of interest. Then, the tubing could be tied off at the top, and the wireline could be lowered further. Thus, a tracer element could be injected at the top of the zone of interest to the coiled tubing, while the instrument 10 is played up and down through the zone of interest on a slick line.

Further, coiled tubing alone could be used to lower the instrument 10. With coiled tubing, openings or spacers could be provided above the instrument 10 so that the instrument 10 and the tubing could be lowered into the well to a certain depth, tracer elements could be injected through the coiled tubing, and then the instrument 10 could be run through the zone of interest. In this technique, no slick line is even needed, as the coiled tubing acts a both a support for the instrument 10 and a path for injecting tracers. A wide variety of techniques for using the slick line in conjunction with the instrument 10 will be appreciated by one of ordinary skill in the art.

Although the use of the instrument 10 has been described in terms of slick line usage, if only wireline is available, the instrument 10 could of course be run on that wireline. It would not need to be coupled to a

surface system, however, but could instead use the wireline simply as another form of cabling for lowering the instrument 10.

The foregoing disclosure and description of the invention are illustrative and explanatory thereof, and various changes in the size, shape, materials, components, circuit elements, wiring connections and contacts, as well as in the details of the illustrated circuitry and construction and method of operation may be made without departing from the spirit of the invention.

APPENDIX 1

```
        TITLE ROUTINE-TSCAN.ASM

        NOLIST
        INCLUDE GLOBL.ASM
        LIST

        PAGE
        EXTERN  TBL1,iTBL1

        .CODE
;
;       PROGRAM START
;
MAIN    .SECTION OFFSET MAINADDR
;
        LJMP    INITM               ;INITIALIZATION ROUTINE
;
;****;****;****;****;****;****;****;****;****;****;****;****;****;
;
; INTERRUPT 0, SPECTRAL CHANNEL DATA PROCESSING
;
INT0    .SECTION OFFSET MAINADDR+0003H
;
        LJMP    INT0_R
;
;****;****;****;****;****;****;****;****;****;****;****;****;****;*
;
; TIMER 0 INTERRUPT
;
T0INT   .SECTION OFFSET MAINADDR+000BH
;
        LJMP    T0INTR
;
;****;****;****;****;****;****;****;****;****;****;****;****;****;**
;
INT1    .SECTION OFFSET MAINADDR+0013H
;
        CLR     EX1
        RETI
;
;****;****;****;****;****;****;****;****;****;****;****;****;****;***
;
; TIMER 1 INTERRUPT, RELOAD 400 HZ TIMER
;
T1INT   .SECTION OFFSET MAINADDR+001BH
;
        LJMP    T1INTR
;
;****;****;****;****;****;****;****;****;****;****;****;****;****;**
;
; SERIAL PORT INTERRUPT
;
TRINT   .SECTION OFFSET MAINADDR+0023H
;
        LJMP    SERINT
;
;****;****;****;****;****;****;****;****;****;****;****;****;****;**
;
; TIMER 2 INTERRUPT
;
```

```
T2INT   .SECTION OFFSET MAINADDR+002BH
;
        CLR     ET2
        RETI
;
;****;****;****;****;****;****;****;****;****;****;****;****;****;****;****;**
;
MAIN8   .SECTION OFFSET MAINADDR+0033H
;
; INTERRUPT REQUEST TO SERVICE FIFO
; USES rR0H AND rR0L DURING ITS OPERATION...
;
; THIS ROUTINE READS THE SPECTRAL CHANNEL DATA FROM THE FIFO AND WRITES
; IT TO AN INPUT BUFFER IN RAM.  THE INTERRUPT OCCURS EVERY FRAME AND IS
; INITIATED BY THE SYNCEN PULSE COMING FROM THE C31.  THE SIZE OF THE INPUT
; BUFFER IS 2 FIELDS (64 FRAMES) LONG WITH EACH FRAME CONTAINING 20 CHANNELS.
; WHEN THE BUFFER IS FULL THE BUFFER POINTER IS RESET TO THE BUFFER BASE ADDRESS
;
FRAMSIZ EQU     20                      ;# OF 8 BIT CHANNELS IN ONE FRAME OF DATA
;
INT0_R  PUSH    A                       ; SAVE ACCUMULATOR
        PUSH    PSW                     ; SAVE PROGRAM STATUS WORD
        CLR     RS0                     ; SELECT REGISTER BANK 0
        CLR     RS1                     ; SELECT REGISTER BANK 0
;
        MOV     A,mSKIPFLG
        JNZ     RDFIFO                  ;SUBSEQUENT INTERRUPT
;
; FIRST INTERRUPT
;
        MOV     mSKIPFLG,#01            ;SET SKIP FLAG
        MOV     mFRCTR,#00              ;CLEAR FRAME COUNTER
        MOV     mCHBUFH,#>IPBUFF;INITIALIZE IPBUFF PTR TO BASE ADDR
        MOV     mCHBUFL,#<IPBUFF
        JMP     XINT0B                  ;EXIT ROUTINE
;
; SUBSEQUENT INTERRUPTS
;
RDFIFO  PUSH    DPH                     ; SAVE DPH
        PUSH    DPL                     ; SAVE DPL
;
; READ FIFO DATA LOOP
;
        MOV     rR0H,#FRAMSIZ           ;LOAD FRAME SIZE INTO REG CTR
        MOV     rR0L,#SCBUF             ;LOAD TEMP SPEC CH BUF BASE ADDR
        MOV     DPTR,#FIFO              ;LOAD FIFO ADDR INTO DPTR
;
READFIF MOVX    A,@DPTR                 ;FIFO DATA
        MOV     @rR0L,A                 ;WR BYTE OF SPEC CH TO TEMP BUFF
        INC     rR0L                    ;INC TO NEXT IN BUFF ADDR
        DJNZ    rR0H,READFIF            ;DECREMENT CH COUNTER-CONTINUE READING UNTIL END
;
; WRITE FIFO DATA TO RAM BUFFER LOOP
;
        MOV     A,mSKIPFLG              ;CHECK ALIGNMENT OF FRAME
        CJNE    A,#01,LOKFOR0
        MOV     rR0L,#SCBUF             ;RESET VALUES AND LOAD PERM CHANNEL
        MOV     rR0H,#FRAMSIZ           ;BUFFER ADDRESS TO DPTR TO WRITE
        MOV     DPL,mCHBUFL             ;FROM TEMP BUFF TO PERM BUFF
        MOV     DPH,mCHBUFH
```

```
        JMP     WRMEM
;
LOKFOR0 MOV     A,CH0           ;LOOKING FOR FRAME 0 IN THE TEMP
        JNZ     XINT0A          ;STORAGE BUFFER SO AS TO REALIGN
        MOV     rR0H,#02H       ;CHANNEL 0 WITH THE FIRST LOCATION IN
        MOV     rR0L,#CH0       ;THE IP BUFFER
        MOV     DPL,mCHBUFL     ;FROM TEMP BUFF TO PERM BUFF
        MOV     DPH,mCHBUFH
;
WRMEM   MOV     A,@rR0L         ;WRITE VALUES FROM TEMP BUFF TO
        MOVX    @DPTR,A         ;PERM BUFF
        INC     DPTR
        INC     rR0L
        DJNZ    rR0H,WRMEM
;
INCFCTR MOV     A,mSKIPFLG
        CJNE    A,#01,NOFLD1
        INC     mFRCTR          ;INC FRAME CTR
        MOV     A,mFRCTR
        CJNE    A,#32,SECFLD    ;CHECK FOR END OF FIELD
        JMP     FRSTFLD         ;OR 1ST HALF OF PERM BUFF FULL

SECFLD  CJNE    A,#64,NOFIELD   ;IF FRAME COUNT < 64 (NOT 2 FIELDS READ) EXIT RO
        MOV     DPTR,#IPBUFF    ;ELSE IF = 64
        MOV     A,DPL
        ADD     A,#2            ;INC DPTR TO 3RD LOCATION IN IPBUFF
        MOV     mCHBUFL,A       ;RESET PTR TO PERM IP BUFF BASE ADDRESS FOR NEXT
        MOV     mCHBUFH,DPH
;
        MOV     DPTR,#ENDIBUF
        MOVX    A,@DPTR
        MOV     rR0H,A
        CJNE    rR0H,#00,SSKP   ;CHECK FOR FRAME 0
        INC     DPTR
        MOVX    A,@DPTR
        MOV     rR0L,A
        MOV     DPTR,#IPBUFF
        MOV     A,rR0H
        MOVX    @DPTR,A
        INC     DPTR
        MOV     A,rR0L
        MOVX    @DPTR,A
        JMP     FLD2
;
SSKP    MOV     SKIPFLG,#02H    ;IF NOT FRAME 0 WAIT FOR 0 FRAME
        MOV     DPTR,#IPBUFF
        MOV     mCHBUFH,DPH
        MOV     mCHBUFL,DPL
        MOV     mFLDFUL,#02H
        MOV     mFRCTR,#00H
        JMP     XINT0A          ;ZERO FRAME COUNTER

FLD2    MOV     mFRCTR,#00H     ;CLEAR FRAME COUNTER
        MOV     mFLDFUL,#02H    ;SET SECOND HALF FULL FLAG
        JMP     XINT0A
                                ;AND EXIT ROUTINE
;
FRSTFLD MOV     mFLDFUL,#01H    ;FIRST HALF FULL FLAG
NOFLD1  MOV     mSKIPFLG,#01H
NOFIELD MOV     mCHBUFH,DPH     ;SAVE THE I/P BUFF PTR HI ADDR
```

```
        MOV     mCHBUFL,DPL     ;SAVE THE I/P BUFF PTR LO ADDR
;
XINT0A  POP     DPL             ; RESTORE DPL
        POP     DPH             ; RESTORE DPH
;
XINT0B  POP     PSW             ; RESTORE PROGRAM STATUS WORD
        POP     A               ; RESTORE ACCUMULATOR
        RETI
;
;****;****;****;****;****;****;****;****;****;****;****;****;****;***
;
; TIMER 1 INTERRUPT ROUTINE
;
; THIS TIMER IS BEING USED TO GENERATE THE 400 Hz SIGNAL USED TO
; INTERRUPT THE C31.  THIS ROUTINE IS SET UP TO RELOAD THE TIMER.
;
; SAVE ACCUMULATOR AND PROGRAM STATUS WORD
;
T1INTR  PUSH    A               ; SAVE ACCUMULATOR
        PUSH    PSW             ; SAVE PROGRAM STATUS WORD
;
; CHECK TIMER EXTENSION FOR EXPIRATION
;
        MOV     A,mTE1          ; TIMER EXTENSION TO ACCUMULATOR
        INC     A               ; INCREMENT TIMER EXTENSION
        MOV     mTE1,A          ; STORE BACK
        JNZ     T1EXIT          ; TIMER EXTENSION NON-ZERO
;
        CLR     TR1             ; TURN TIMER OFF
        MOV     TL1,mRLOAD1+2   ; RELOAD VALUE (LSBYTE) TO ACCUMULATOR
        MOV     TH1,mRLOAD1+1   ; RELOAD VALUE (MSBYTE-1) TO ACCUMULATOR
        MOV     mTE1,mRLOAD1    ; RELOAD VALUE (MSBYTE) TO mTE0
        SETB    TR1             ; RESTART TIMER 1
;
        CLR     P1.2            ; PULL 400 Hz INT TO OTHER CPU LO
        NOP
        SETB    P1.2            ; PULL 400 Hz INT SIGNAL BACK HI
;
T1EXIT  POP     PSW             ; RESTORE PROGRAM STATUS WORD
        POP     A               ; RESTORE ACCUMULATOR
        RETI
;
;****;****;****;****;****;****;****;****;****;****;****;****;****;****;
;
; TIMER 0 INTERRUPT ROUTINE
; SAVE ACCUMULATOR AND PROGRAM STATUS WORD
;
T0INTR  PUSH    A               ; SAVE ACCUMULATOR
        PUSH    PSW             ; SAVE PROGRAM STATUS WORD
;
; CHECK TIMER EXTENSION FOR EXPIRATION
;
        MOV     A,mTE0          ; TIMER EXTENSION TO ACCUMULATOR
        INC     A               ; INCREMENT TIMER EXTENSION
        MOV     mTE0,A          ; STORE BACK
        JNZ     T0EXIT          ; TIMER EXTENSION NON-ZERO
;
; RELOAD TIMER AND EXTENSION
;
        CLR     TR0             ; STOP TIMER 0
```

```
        MOV     TL0,mRLOAD0+2   ; RELOAD VALUE (LSBYTE) TO TL0
        MOV     TH0,mRLOAD0+1   ; RELOAD VALUE (MSBYTE-1) TO TH0
        MOV     mTE0,mRLOAD0    ; RELOAD VALUE (MSBYTE) TO mTE0
        SETB    TR0             ; RESTART TIMER 0
;
        MOV     A,mMINCTRL      ; MINUTE COUNTER (LOWER)
        ADD     A,#01           ; INCREMENT
        MOV     mMINCTRL,A      ; STORE BACK
        MOV     A,mMINCTRH      ; MINUTE COUNTER (UPPER)
        ADDC    A,#00           ; FINISH INCREMENT
        MOV     mMINCTRH,A      ; STORE BACK
        MOV     T0FLG,#01H      ; SET EXPIRED FLAG
;
; CLEAN UP AND EXIT
;
T0EXIT  POP     PSW             ; RESTORE PROGRAM STATUS WORD
        POP     A               ; RESTORE ACCUMULATOR
        RETI
;
;****;****;****;****;****;****;****;****;****;****;****;****;****;****;
;
; UART HANDLER
;
; USES rR0H DURING ITS OPERATION...
;
; USER POSTS AN OUTPUT BY
; (1) LOADING THE "PICK" ADDRESS IN mRSOUT+1 AND mRSOUT+2,
; (2) LOADING mRSOUT WITH AN 80H (POST INSTRUCTION) OR C0H (PRINT INSTRUCTION)
; (3) SHOVING THE FIRST CHARACTER TO BE TRANSMITTED INTO SBUF
;
; OUTPUT AREA
;   1) CONTROL WORD
;      BIT 7   OUTPUT POSTED BIT
;   2) PICK ADDRESS
;   3) PICK ADDRESS
;
; SAVE SOME CPU REGISTERS
;
SERINT  PUSH    A               ; SAVE ACCUMULATOR
        PUSH    PSW             ; SAVE PROGRAM STATUS WORD
        PUSH    DPH             ; SAVE DPH
        PUSH    DPL             ; SAVE DPL
;
; CHECK IF OUTPUT IS WORKING
;
        JNB     TI,CKIN         ; NO TRANSMIT INTERRUPT, GO CHECK RECEIVE
        CLR     TI              ; CLEAR TRANSMIT INTERRUPT
        MOV     A,mRSOUT        ; OUTPUT CONTROL WORD
        JNB     A.7,CKIN        ; SCREW UP, GO CHECK RECEIVE
;
; FEED THE UART
;
        MOV     DPL,mRSOUT+2    ; PICK ADDRESS (LSBYTE) TO DPL
        MOV     A,mRSOUT+1      ; PICK ADDRESS (MSBYTE)
        MOV     DPH,A           ; PICK ADDRESS (MSBYTE) TO DPH
        JNZ     CKOUT1          ; EXTERNAL MEMORY TO BE TRANSMITTED
;
; OUTPUT FROM INTERNAL MEMORY
;
        CLR     RS0             ; SELECT REGISTER BANK 0
```

CA2183123

```
        CLR     RS1                 ; SELECT REGISTER BANK 0
        MOV     rR0H,DPL            ; PICK ADDRESS (INTERNAL MEMORY)
        MOV     A,@rR0H             ; CHARACTER TO BE TRANSMITTED (INTERNAL)
        SJMP    CKOUT2              ; CONTINUE
;
; OUTPUT FROM EXTERNAL MEMORY
;
CKOUT1  MOVX    A,@DPTR             ; CHARACTER TO BE TRANSMITTED (EXTERNAL)
;
CKOUT2  JZ      CKOUT3              ; IS A NULL, FINISHED
        MOV     SBUF,A              ; STORE INTO UART
        INC     DPTR                ; INCREMENT DPTR
        MOV     mRSOUT+1,DPH        ; STORE BACK INCREMENTED PICK ADDRESS (MSBYTE)
        MOV     mRSOUT+2,DPL        ; STORE BACK INCREMENTED PICK ADDRESS (LSBYTE)
        SJMP    CKEX                ; FINISHED FOR NOW
;
; TRANSMIT COMPLETE...
; CLEAR TRANSMIT FLAG AND RESTORE INPUT AREA (IF REQUIRED)
;
CKOUT3  MOV     A,mRSOUT            ; OUTPUT CONTROL WORD
        CLR     A.7                 ; CLEAR TRANSMIT FLAG
        MOV     mRSOUT,A            ; STORE BACK
        JNZ     CKEX                ; NO RESTORE REQUIRED
;
; RESTORE INPUT AREA
;
        MOV     DPTR,#IOSAV+2       ; IOSAV+2 ADDRESS VALUE TO DPTR
        MOVX    A,@DPTR             ; RSIN RESTORE LOCATION
        INC     DPTR                ; INCREMENT IOSAV BLOCK LOCATION POINTER
        MOV     RSIN,A              ; STORE INTO RSIN
        MOVX    A,@DPTR             ; RSIN+9 RESTORE LOCATION
        MOV     RSIN+9,A            ; STORE INTO RSIN+9
        MOV     DPTR,#IOSAV         ; IOSAVE ADDRESS VALUE TO DPTR
        MOVX    A,@DPTR             ; SAVED IE
        MOV     IE,A                ; RESTORE SAVED IE
        INC     DPTR                ; INCREMENT DPTR
        MOVX    A,@DPTR             ; SAVED SCON
        JNB     A.4,CKEX            ; REN WAS NOT SET
        SETB    REN                 ; RESTORE REN
;
; CLEAN UP AND EXIT
;
CKEX    POP     DPL                 ; RESTORE DPL
        POP     DPH                 ; RESTORE DPH
        POP     PSW                 ; RESTORE PROGRAM STATUS WORD
        POP     A                   ; RESTORE ACCUMULATOR
        RETI
;
;       SUBTITLE **** UART SERIAL INPUT HANDLER ****
        PAGE
;
; USER POSTS AN INPUT BY
; (1) LOADING TERMINATOR COUNT IN mRSIN+15
; (2) LOADING INPUT BUFFER LENGTH IN mRSIN+13..mRSIN+14
; (2) LOADING "PUT" ADDRESS IN mRSIN+11..mRSIN+12
; (3) LOADING "WHOAMI" IN mRSIN+10
; (4) LOADING mRSIN+9 WITH AN 80H
;
; INPUT AREA USED BY INTERRUPT HANDLER
;
```

```
; mRSIN    ) INPUT CONTROL WORD
;     BIT 7     INPUT WORKING
;     BIT 6     MY ADDRESS FLAG
;     BIT 5     FRAMING ERROR
;     BIT 4     INPUT MESSAGE EXCEEDS BUFFER LENGTH
; mRSIN+1  ) "WHOAMI"
; mRSIN+2  ) PUT ADDRESS (MSBYTE)
; mRSIN+3  ) PUT ADDRESS (LSBYTE)
; mRSIN+4  ) INPUT BUFFER LENGTH (MSBYTE)
; mRSIN+5  ) INPUT BUFFER LENGTH (LSBYTE)
; mRSIN+6  ) TERMINATOR COUNT
; mRSIN+7  ) INPUT CHARACTER COUNT (MSBYTE)
; mRSIN+8  ) INPUT CHARACTER COUNT (LSBYTE)
;
; AREA LOADED BY INPUT SUBROUTINES
;
; mRSIN+9  ) INPUT STATUS WORD
;     BIT 7     INPUT POSTED
;     BIT 6     INPUT SUCCESSFUL/MY ADDRESS FLAG
;     BIT 5     FRAMING ERROR
;     BIT 4     INPUT MESSAGE EXCEEDS BUFFER LENGTH
; mRSIN+10 ) "WHOAMI"
; mRSIN+11) PUT ADDRESS-UPPER
; mRSIN+12) PUT ADDRESS-LOWER
; mRSIN+13) INPUT BUFFER LENGTH (MSBYTE)
; mRSIN+14) INPUT BUFFER LENGTH (LSBYTE)
; mRSIN+15) TERMINATOR COUNT
;
CKIN    JNB     RI,CKEX             ; NO RECEIVE INTERRUPT
;
        CLR     RI                  ; CLEAR RECEIVE INTERRUPT
        MOV     A,mRSIN+9           ; INPUT STATUS WORD
        JNB     A.7,CKIN1           ; SCREW UP, EXIT
        MOV     A,mRSIN             ; INPUT CONTROL WORD
        JB      A.7,CKIN1           ; RECEIVE IN PROCESS
;
; INITIAL RECEIVE SETUP OR REPOST...
; MOVE mRSIN+9...mRSIN+15 TO mRSIN...mRSIN+6
; AND CLEAR mRSIN+7..mRSIN+8 (INPUT CHARACTER COUNT)
;
        MOV     mRSIN,mRSIN+9    ; INPUT STATUS WORD TO INPUT CONTROL WORD
        MOV     mRSIN+1,mRSIN+10; WHOAMI
        MOV     mRSIN+2,mRSIN+11; PUT ADDRESS (MSBYTE)
        MOV     mRSIN+3,mRSIN+12; PUT ADDRESS (LSBYTE)
        MOV     mRSIN+4,mRSIN+13; INPUT BUFFER LENGTH (MSBYTE)
        MOV     mRSIN+5,mRSIN+14; INPUT BUFFER LENGTH (LSBYTE)
        MOV     mRSIN+6,mRSIN+15; TERMINATOR COUNT
        MOV     mRSIN+7,#00      ; CLEAR CHARACTER COUNT (MSBYTE)
        MOV     mRSIN+8,#00      ; CLEAR CHARACTER COUNT (LSBYTE)
;
; CHECK FOR TOO LONG AN INPUT MESSAGE
;
CKIN1   MOV     DPH,mRSIN+7      ; INPUT CHARACTER COUNT (MSBYTE)
        MOV     DPL,mRSIN+8      ; INPUT CHARACTER COUNT (LSBYTE)
        INC     DPTR             ; INCREMENT INPUT CHARACTER COUNT
        MOV     mRSIN+7,DPH      ; STORE BACK INPUT CHARACTER COUNT (MSBYTE)
        MOV     mRSIN+8,DPL      ; STORE BACK INPUT CHARACTER COUNT (LSBYTE)

        MOV     A,mRSIN+5        ; INPUT BUFFER LENGTH (LSBYTE)
        XRL     A,DPL            ; INPUT BUFFER LENGTH .XOR. CHARACTER COUNT
```

```
        JNZ     CKIN2                   ; INPUT MESSAGE NOT   O LONG
        MOV     A,mRSIN+4               ; INPUT BUFFER LENGTH (MSBYTE)
        XRL     A,DPH                   ; INPUT BUFFER LENGTH .XOR. CHARACTER COUNT
        JZ      CKIN4                   ; INPUT MESSAGE TOO LONG
;
; READ UART AND STORE CHARACTER
;
CKIN2   MOV     DPH,mRSIN+2             ; PUT ADDRESS - UPPER
        MOV     DPL,mRSIN+3             ; PUT ADDRESS - LOWER
        MOV     A,SBUF                  ; READ UART
        MOVX    @DPTR,A                 ; STORE CHARACTER INTO PUT ADDRESS
;
; CHECK FOR SECOND CHARACTER...
; IF SECOND CHARACTER, COMPARE WITH 'WHOAMI' AND SET MY ADDRESS IF THEY DO
;
        MOV     A,mRSIN+7               ; INCREMENTED CHARACTER COUNT (MSBYTE)
        JNZ     CKIN3                   ; .GT. 255
        MOV     A,mRSIN+8               ; INCREMENTED CHARACTER COUNT (LSBYTE)
        CJNE    A,#02,CKIN3             ; NOT SECOND CHARACTER
        MOVX    A,@DPTR                 ; JUST STORED CHARACTER
        CJNE    A,mRSIN+1,CKIN3         ; 'WHOAMI' .NE. JUST STORED CHARACTER
        ORL     mRSIN,#40H              ; SET MY ADDRESS FLAG
;
; CHECK FOR TERMINATING CHARACTER
;
CKIN3   MOVX    A,@DPTR                 ; JUST STORED CHARACTER
        INC     DPTR                    ; INCREMENT PUT ADDRESS
        MOV     mRSIN+2,DPH             ; INCREMENTED PUT ADDRESS - UPPER
        MOV     mRSIN+3,DPL             ; INCREMENTED PUT ADDRESS - LOWER
        CJNE    A,#<CTERM,CKEX          ; NOT TERMINATING CHARACTER, MORE TO RECEIVE
;
; WAS TERMINATING CHARACTER, DECREMENT TERMINATOR COUNT AND CHECK FOR ZERO
;
        DJNZ    mRSIN+6,CKEX            ; NON-ZERO, MORE TO RECEIVE
;
; TERMINATOR COUNT WAS ZERO, CLEAR RECEIVE WORKING BIT
;
        MOV     A,mRSIN                 ; CONTROL WORD
        CLR     A.7                     ; CLEAR RECEIVE WORKING BIT
        MOV     mRSIN,A                 ; STORE BACK INTO INPUT CONTROL WORD
        JNZ     CKIN5                   ; PROPER RECEIVE
        SJMP    CKEX                    ; WE WILL REPOST AN INPUT ON NEXT INTERRRUPT
;
; ERROR CONDITIONS HANDLED HERE
;
CKIN4   MOV     mRSIN,#10H              ; SET INPUT TOO LONG FLAG IN CONTROL WORD
;
; MOVE CONTROL WORD TO STATUS WORD, DISABLE SERIAL PORT INTERRUPT,
; KILL REN, CLEAR MST CONTROL REGISTER, AND CLEAR CONTROL WORD
;
CKIN5   MOV     mRSIN+9,mRSIN           ; CONTROL WORD TO STATUS WORD
        CLR     ES                      ; DISABLE SERIAL PORT INTERRUPT
        CLR     REN                     ; CLEAR REN
        SETB    P1.6                    ; TURN OFF SERIAL SENSOR
        MOV     mRSIN,#00               ; CLEAR CONTROL WORD
        LJMP    CKEX                    ; EXIT
;
;****;****;****;****;****;****;****;****;****;****;****;****;****;*
;
;****;****;****;****;****;****;****;****;****;****;****;****;****;***
```

```
; INITIALIZATION ROUTINE AS A RESULT OF A POWER UP...
; SELECT REGISTER BANK 0 AND CLEAR INTERNAL RAM

INITM   CLR     RS0                 ; SELECT REGISTER BANK 0
        CLR     RS1                 ; SELECT REGISTER BANK 0
;       CLR     P1.5
        NOP
        NOP
;       SETB    P1.5                ; SIGNAL USED TO RESET OTHER CPU

        MOV     rR0H,#<TOPMEM       ; SET CLEAR ADDRESS REGISTER
INITM1  MOV     @rR0H,#00H          ; CLEAR LOCATION @R0L
        DJNZ    mR0H,INITM1         ; LOOP UNTIL DONE
;
; CLEAR PCON
;
        MOV     PCON,#00            ; CLEAR PCON
        CLR     P1.5
;
; INITIALIZE STACK POINTER (SP) AND SYSTEM INDEX REGISTER (mRAH)
;
        MOV     SP,#<vSTACK-1       ; INITIALIZE STACK POINTER
        MOV     mRAH,#<vPINDEX      ; INITIALIZE SYSTEM INDEX REGISTER
;
; SET SERIAL PORT CONTROL REGISTER FOR A 19.2K BAUD
; SET SERIAL COMM PORT TO MODE 1
;
SCONV   EQU     48H                 ;SERIAL COMM IN MODE 3
BAUDR   EQU     FFF1H               ;FFF1 BAUD ROLL OVER VALUE FOR T2, 65521D
T2CONV  EQU     34H                 ;SET T2 AS BAUD RATE GEN AND TURN ON
;
; SET UP SERIAL COMM
;
        MOV     SCON,#SCONV         ;SET UP SERIAL COMM
        MOV     RCAP2H,#>BAUDR      ;LOAD UPPER BYTE OF ROLL OVER VALUE
        MOV     RCAP2L,#<BAUDR      ;LOAD LOWER BYTE OF ROLL OVER VALUE
        MOV     T2CON,#T2CONV       ;SET UP T2 AND TURN ON
;
; SET UP PORT 1 AND PORT 3 I/O MASKS, THEN LOAD INTERRUPT ENABLE REGISTER
;
        MOV     P1,#P1V             ; PORT 1 MASK TO PORT 1
        MOV     P3,#P3V             ; PORT 3 MASK TO PORT 3
        SETB    IT0                 ; INT0 FOR EDGE TRIGGER INT
        MOV     IE,#8BH             ; ENABLE ET1 AND ETO
;
        SUBTITLE **** INTERPRETER ****
        PAGE
;
; SELECT REGISTER BANK 1 AND
; SET UP THE TABLE POINTER (ENTRY ADDRESS OF THE PROCESS TABLE)...
;
        CLR     RS1                 ; SELECT REGISTER BANK 1
        SETB    RS0                 ; SELECT REGISTER BANK 1

;
        MOV     rTABPH,#>TBL1       ; ENTRY ADDRESS TO mTABPH (MSBYTE)
        MOV     rTABPL,#<TBL1       ; ENTRY ADDRESS TO mTABPL (LSBYTE)
;
; RESTORE DPTR (TABLE POINTER) FROM TABP
```

```
;
INTRP   MOV     DPH,rTABPH      ; RESTORE DPH (TABLE POINTER) FROM TABPH
        MOV     DPL,rTABPL      ; RESTORE DPL (TABLE POINTER) FROM TABPL
;
; OBTAIN SUBROUTINE ADDRESS FROM TABLE
;
        MOVX    A,@DPTR         ; MSBYTE OF SUBROUTINE ADDRESS
        MOV     mSUBRH,A        ; MSBYTE OF SUBROUTINE ADDRESS TO SUBRH
        INC     DPTR            ; INCREMENT TABLE POINTER
        MOVX    A,@DPTR         ; LSBYTE OF SUBROUTINE ADDRESS
        MOV     mSUBRL,A        ; LSBYTE OF SUBROUTINE ADDRESS TO SUBRL
        INC     DPTR            ; INCREMENT TABLE POINTER
        MOV     mDFLDH,DPH      ; MSBYTE OF DATA FIELD POINTER TO mDFLDH
        MOV     mDFLDL,DPL      ; LSBYTE OF DATA FIELD POINTER TO mDFLDL
;
; OBTAIN ARG1 OR ARG1 INDIRECT
;
        MOVX    A,@DPTR         ; ARGUMENT (MSBYTE) OR INDIRECT FLAG
        CLR     mBIT7           ; CLEAR mBIT7 (NO INDIRECT)
        CJNE    A,#<ZI,INTRP1   ; INDIRECT ?
        SETB    mBIT7           ; SET mBIT7 (INDIRECT REQUESTED)
        INC     DPTR            ; INCREMENT TABLE POINTER PAST INDIRECT FLAG
INTRP1  MOVX    A,@DPTR         ; ARGUMENT (MSBYTE)
        MOV     rARG1H,A        ; SAVE IT IN rARG1H
        INC     DPTR            ; INCREMENT TABLE POINTER
        MOVX    A,@DPTR         ; ARGUMENT (LSBYTE)
        MOV     rARG1L,A        ; SAVE IT IN rARG1L
        INC     DPTR            ; INCREMENT TABLE POINTER
        JNB     mBIT7,INTRP3    ; NO INDIRECT REQUESTED
;
        MOV     A,rARG1H        ; MSBYTE OF ARGUMENT (INDIRECT) TO ACC
        JNZ     INTRP2          ; EXTERNAL MEMORY INDIRECT
;
        MOV     rTABPH,mARG1L   ; LSBYTE OF ARGUMENT (INDIRECT) TO rTABPH
        MOV     mARG1H,@rTABPH  ; INDIRECT (MSBYTE) TO mARG1H
        INC     rTABPH          ; INCREMENT INDIRECT POINTER
        MOV     mARG1L,@rTABPH  ; INDIRECT (LSBYTE) TO mARG1L
        SJMP    INTRP3          ; INTERNAL INDIRECT FINISHED
;
INTRP2  PUSH    DPH             ; TABLE POINTER TO STACK (MSBYTE)
        PUSH    DPL             ; TABLE POINTER TO STACK (LSBYTE)
        MOV     DPH,A           ; MSBYTE OF ARGUMENT (INDIRECT) TO DPH
        MOV     DPL,rARG1L      ; LSBYTE OF ARGUMENT (INDIRECT) TO DPL
        MOVX    A,@DPTR         ; INDIRECT (MSBYTE)
        MOV     rARG1H,A        ; SAVE IT IN rARG1H
        INC     DPTR            ; INCREMENT INDIRECT POINTER
        MOVX    A,@DPTR         ; INDIRECT (LSBYTE)
        MOV     rARG1L,A        ; SAVE IT IN rARG1L
        POP     DPL             ; TABLE POINTER FROM STACK (LSBYTE)
        POP     DPH             ; TABLE POINTER FROM STACK (MSBYTE)
;
; OBTAIN ARG2 OR ARG2 INDIRECT
;
INTRP3  MOVX    A,@DPTR         ; ARGUMENT (MSBYTE) OR INDIRECT FLAG
        CLR     mBIT7           ; CLEAR mBIT7 (NO INDIRECT)
        CJNE    A,#<ZI,INTRP4   ; INDIRECT ?
        SETB    mBIT7           ; SET mBIT7 (INDIRECT REQUESTED)
        INC     DPTR            ; INCREMENT TABLE POINTER PAST INDIRECT FLAG
INTRP4  MOVX    A,@DPTR         ; ARGUMENT (MSBYTE)
        MOV     rARG2H,A        ; SAVE IT IN rARG2H
```

```
        INC     DPTR                    ; INCREMENT TABLE POINTER
        MOVX    A,@DPTR                 ; ARGUMENT (LSBYTE)
        MOV     rARG2L,A                ; SAVE IT IN rARG2L
        INC     DPTR                    ; INCREMENT TABLE POINTER
        JNB     mBIT7,INTRP6            ; NO INDIRECT REQUESTED
;
        MOV     A,rARG2H                ; MSBYTE OF ARGUMENT (INDIRECT) TO ACC
        JNZ     INTRP5                  ; EXTERNAL MEMORY INDIRECT
;
        MOV     rTABPH,mARG2L           ; LSBYTE OF ARGUMENT (INDIRECT) TO rTABPH
        MOV     mARG2H,@rTABPH          ; INDIRECT (MSBYTE) TO mARG2H
        INC     rTABPH                  ; INCREMENT INDIRECT POINTER
        MOV     mARG2L,@rTABPH          ; INDIRECT (LSBYTE) TO mARG2L
        SJMP    INTRP6                  ; INTERNAL INDIRECT FINISHED
;
INTRP5  PUSH    DPH                     ; TABLE POINTER TO STACK (MSBYTE)
        PUSH    DPL                     ; TABLE POINTER TO STACK (LSBYTE)
        MOV     DPH,A                   ; MSBYTE OF ARGUMENT (INDIRECT) TO DPH
        MOV     DPL,rARG2L              ; LSBYTE OF ARGUMENT (INDIRECT) TO DPL
        MOVX    A,@DPTR                 ; INDIRECT (MSBYTE)
        MOV     rARG2H,A                ; SAVE IT IN rARG2H
        INC     DPTR                    ; INCREMENT INDIRECT POINTER
        MOVX    A,@DPTR                 ; INDIRECT (LSBYTE)
        MOV     rARG2L,A                ; SAVE IT IN rARG2L
        POP     DPL                     ; TABLE POINTER FROM STACK (LSBYTE)
        POP     DPH                     ; TABLE POINTER FROM STACK (MSBYTE)
;
; OBTAIN ARG3 OR ARG3 INDIRECT
;
INTRP6  MOVX    A,@DPTR                 ; ARGUMENT (MSBYTE) OR INDIRECT FLAG
        CLR     mBIT7                   ; CLEAR mBIT7 (NO INDIRECT)
        CJNE    A,#<ZI,INTRP7           ; INDIRECT ?
        SETB    mBIT7                   ; SET mBIT7 (INDIRECT REQUESTED)
        INC     DPTR                    ; INCREMENT TABLE POINTER PAST INDIRECT FLAG
INTRP7  MOVX    A,@DPTR                 ; ARGUMENT (MSBYTE)
        MOV     rARG3H,A                ; SAVE IT IN rARG3H
        INC     DPTR                    ; INCREMENT TABLE POINTER
        MOVX    A,@DPTR                 ; ARGUMENT (LSBYTE)
        MOV     rARG3L,A                ; SAVE IT IN rARG3L
        INC     DPTR                    ; INCREMENT TABLE POINTER
        JNB     mBIT7,INTRP9            ; NO INDIRECT REQUESTED
;
        MOV     A,rARG3H                ; MSBYTE OF ARGUMENT (INDIRECT) TO ACC
        JNZ     INTRP8                  ; EXTERNAL MEMORY INDIRECT
;
        MOV     rTABPH,mARG3L           ; LSBYTE OF ARGUMENT (INDIRECT) TO rTABPH
        MOV     mARG3H,@rTABPH          ; INDIRECT (MSBYTE) TO mARG3H
        INC     rTABPH                  ; INCREMENT INDIRECT POINTER
        MOV     mARG3L,@rTABPH          ; INDIRECT (LSBYTE) TO mARG3L
        SJMP    INTRP9                  ; INTERNAL INDIRECT FINISHED
;
INTRP8  PUSH    DPH                     ; TABLE POINTER TO STACK (MSBYTE)
        PUSH    DPL                     ; TABLE POINTER TO STACK (LSBYTE)
        MOV     DPH,A                   ; MSBYTE OF ARGUMENT (INDIRECT) TO DPH
        MOV     DPL,rARG3L              ; LSBYTE OF ARGUMENT (INDIRECT) TO DPL
        MOVX    A,@DPTR                 ; INDIRECT (MSBYTE)
        MOV     rARG3H,A                ; SAVE IT IN rARG3H
        INC     DPTR                    ; INCREMENT INDIRECT POINTER
        MOVX    A,@DPTR                 ; INDIRECT (LSBYTE)
        MOV     rARG3L,A                ; SAVE IT IN rARG3L
```

```
        POP     DPL                     ; TABLE POINTER FROM STACK (LSBYTE)
        POP     DPH                     ; TABLE POINTER FROM STACK (MSBYTE)
;
; CALL SUBROUTINE, BUT FIRST SAVE DPTR (TABLE POINTER) IN TABPH
;
INTRP9  MOV     rTABPH,DPH              ; SAVE TABLE POINTER (MSBYTE)
        MOV     rTABPL,DPL              ; SAVE TABLE POINTER (LSBYTE)
        MOV     DPH,mSUBRH              ; SUBROUTINE ADDRESS (MSBYTE)
        MOV     DPL,mSUBRL              ; SUBROUTINE ADDRESS (LSBYTE)
        CALL    INTRPX                  ; CALL REQUESTED SUBROUTINE
;
; REQUESTED SUBROUTINE RETURNS HERE...
; INTERPRETER EXPECTS A DECREMENT COUNT IN ACCUMULATOR
;
        CLR     RS1                     ; SELECT REGISTER BANK 1
        SETB    RS0                     ; SELECT REGISTER BANK 1
;
        JZ      INTRPA                  ; DECREMENT COUNT IS ZERO
        SETB    C                       ; SET CARRY
        CPL     A                       ; COMPLEMENT DECREMENT COUNT
        ADDC    A,mTABPL                ; mTABPL+NOT(DECREMENT COUNT)+1
        MOV     rTABPL,A                ; STORE BACK IN rTABPL
        JC      INTRPA                  ; NO BORROW, GO CHECK BREAK POINT
        DEC     rTABPH                  ; DECREMENT rTABPH
;
; CHECK FOR SOFTWARE INTERRUPT
;
INTRPA  MOV     DPTR,#SWINTM            ; SOFTWARE INTERRUPT FLAG ADDRESS
        MOVX    A,@DPTR                 ; SOFTWARE INTERRUPT FLAG
        JNB     A.6,INTRPB              ; SOFTWARE INTERRUPT NOT REQUESTED
;
; SOFTWARE INTERRUPT REQUESTED
;
        CLR     A                       ; CLEAR INTERRUPT FLAG WORD
        MOVX    @DPTR,A                 ; STORE BACK SOFTWARE INTERRUPT FLAG WORD
        INC     DPTR                    ; INCREMENT DPTR
        MOV     A,mTABPH                ; RETURN ADDRESS (MSBYTE)
        MOVX    @DPTR,A                 ; SAVE RETURN ADDRESS (MSBYTE)
        INC     DPTR                    ; INCREMENT DPTR
        MOV     A,mTABPL                ; RETURN ADDRESS (LSBYTE)
        MOVX    @DPTR,A                 ; SAVE RETURN ADDRESS (LSBYTE)
        INC     DPTR                    ; INCREMENT DPTR
        MOVX    A,@DPTR                 ; SOFTWARE INTERRUPT ADDRESS (MSBYTE)
        INC     DPTR                    ; INCREMENT DPTR
        MOV     mTABPH,A                ; SOFTWARE INTERRUPT ADDRESS (MSBYTE) TO mTABPH
        MOVX    A,@DPTR                 ; SOFTWARE INTERRUPT ADDRESS (MSBYTE)
        INC     DPTR                    ; INCREMENT DPTR
        MOV     mTABPL,A                ; SOFTWARE INTERRUPT ADDRESS (LSBYTE) TO mTABPL
;
; NOW CHECK FOR BREAK POINT
;
INTRPB  MOV     DPTR,#BREAK             ; BREAK POINT LOCATION (EXTERNAL RAM)
        MOVX    A,@DPTR                 ; BREAK POINT FLAG
        JNZ     INTRPD                  ; BREAK POINT REQUESTED
INTRPC  LJMP    INTRP                   ; GO INTERPRET
;
; BREAK POINT REQUESTED - CHECK IF HIT
;
INTRPD  INC     DPTR                    ; INCREMENT BREAK POINT POINTER (BREAK+1)
        MOVX    A,@DPTR                 ; MSBYTE OF BREAK POINT ADDRESS
```

CA2183123

```
        CJNE    A,mTABPH,INTRPC ; .NE. SAVED TABLE POINTER, GO INTERPRET
        INC     DPTR            ; INCREMENT BREAK POINT POINTER (BREAK+2)
        MOVX    A,@DPTR         ; LSBYTE OF BREAK POINT ADDRESS
        CJNE    A,mTABPL,INTRPC ; .NE. SAVED TABLE POINTER, GO INTERPRET
;
; BREAK POINT HIT...
;
        INC     DPTR                    ; INCREMENT BREAK POINT POINTER (BREAK+3)
        MOVX    A,@DPTR                 ; BREAK POINT ROUTINE ENTRY ADDRESS (MSBYTE)
        MOV     rTABPH,A                ; STUFF INTO mTABPH
        INC     DPTR                    ; INCREMENT BREAK POINT POINTER (BREAK+4)
        MOVX    A,@DPTR                 ; BREAK POINT ROUTINE ENTRY ADDRESS (LSBYTE)
        MOV     rTABPL,A                ; STUFF INTO mTABPL
        INC     DPTR                    ; INCREMENT BREAK POINT POINTER (BREAK+5)
        MOV     A,#<INTRPE              ; INTERPRETER RETURN ADDRESS (LSBYTE)
        MOVX    @DPTR,A                 ; STORE IT INTO BREAK+5
        INC     DPTR                    ; INCREMENT BREAK POINT POINTER (BREAK+6)
        MOV     A,#>INTRPE              ; INTERPRETER RETURN ADDRESS (MSBYTE)
        MOVX    @DPTR,A                 ; STORE IT INTO BREAK+6
;
; BREAK POINT RETURNS HERE
;
INTRPE  CLR     RS1                     ; SELECT REGISTER BANK 1
        SETB    RS0                     ; SELECT REGISTER BANK 1
        LJMP    INTRP                   ; GO INTERPRET
;
; SUBROUTINE CALL SUBROUTINE
;
INTRPX  CLR     A                       ; CLEAR ACCUMULATOR
        JMP     @A+DPTR
;
        TITLE ROUTINE-MAIN.ASM
        SUBTITLE TSCAN
        PAGE
;
        END     MAIN
```

```
        SUBTITLE TSCAN
        PAGE
;
        NOLIST
        INCLUDE MACEVAL.ASM
        INCLUDE MACLIB.ASM
        LIST
        INCLUDE GLOBL.ASM
;
IPOST   .MACRO  ARGA,ARGB,ARGC,ARGD
        .IFSAME ARGA,QINPUT
        .IFNDEF ZQPST
        EXTERN  ZQPST
        .ENDIF
        WORD    ZQPST   ; POST A QUALIFIED INPUT
        .IFNMA  4
        INDR2   ARGB,0
        INDRBU  1
        INDRBL  0
        INDR1   ARGC
        .ELSE
        INDR2   ARGB,0
        INDRBU  ARGD
        INDRBL  0
        INDR1   ARGC
        .ENDIF
        .MACEXIT
        .ENDIF
        .IFSAME ARGA,UINPUT
        .IFNDEF ZUPST
        EXTERN  ZUPST
        .ENDIF
        WORD    ZUPST   ; POST AN UNQUALIFIED INPUT
        .IFNMA  4
        INDR2   ARGB,0
        INDRBU  1
        INDRBL  0
        INDR1   ARGC
        .ELSE
        INDR2   ARGB,0
        INDRBU  ARGD
        INDRBL  0
        INDR1   ARGC
        .ENDIF
        .MACEXIT
        .ENDIF
;       WORD    ERROR
        .ENDM
;
QINPUT  .MACRO  ARGA,ARGB,ARGC,ARGD,ARGE
        .IFNDEF ZQINP
        EXTERN  ZQINP
        .ENDIF
        WORD    ZQINP   ; QUALIFIED INPUT @ARGA, WAIT ARGD SEC(S)
        INDR1   ARGA    ; GOTO ARGC IF INPUT PROPER
        .IFNMA  5
        INDR1   ARGC
        INDRBU  1
        INDRBL  ARGD
        INDR1   ARGB
```

```
        .ELSE
        INDR1   ARGC
        INDRBU  ARGE
        INDRBL  ARGD
        INDR1   ARGB
        .ENDIF
        .ENDM
;
UINPUT  .MACRO  ARGA,ARGB,ARGC,ARGD,ARGE
        .IFNDEF ZUINP
        EXTERN  ZUINP
        .ENDIF
        WORD    ZUINP   ; UNQUALIFIED INPUT @ARGA, WAIT ARGD SEC(S)
        INDR1   ARGA    ; GOTO ARGC IF INPUT PROPER
        .IFNMA  5
        INDR1   ARGC
        INDRBU  1
        INDRBL  ARGD
        INDR1   ARGB
        .ELSE
        INDR1   ARGC
        INDRBU  ARGE
        INDRBL  ARGD
        INDR1   ARGB
        .ENDIF
        .ENDM
;
BBUF    .MACRO  ARGA,ARGB,ARGC,ARGD,ARGE,ARGF,ARGG,ARGH,ARGI,ARGJ
        .IFNDEF ZBBUF
        EXTERN  ZBBUF
        .ENDIF
        .IFNMA  1
        WORD    ZBBUF
        INDR1   ERROR   ; TOO FEW VARIABLES
        .MACEXIT
        .ENDIF
        .IFNMA  2
        WORD    ZBBUF
        INDR1   ERROR   ; TOO FEW VARIABLES
        .MACEXIT
        .ENDIF
        .IFNMA  3
        WORD    ZBBUF   ; BUILD BUFFER AT ARGA
        INDR1   ARGA
        INDR2   ARGB,NULL
        .MACEXIT
        .ENDIF
        .IFNMA  4
        WORD    ZBBUF   ; BUILD BUFFER AT ARGA
        INDR1   ARGA
        INDR3   ARGB,ARGC,NULL
        .MACEXIT
        .ENDIF
        .IFNMA  5
        WORD    ZBBUF   ; BUILD BUFFER AT ARGA
        INDR1   ARGA
        INDR4   ARGB,ARGC,ARGD,NULL
        .MACEXIT
        .ENDIF
        .IFNMA  6
```

```
WORD    ZBBUF   ; BUILD BUFFER AT ARGA
INDR1   ARGA
INDR5   ARGB,ARGC,ARGD,ARGE,NULL
.MACEXIT
.ENDIF
.IFNMA  7
WORD    ZBBUF   ; BUILD BUFFER AT ARGA
INDR1   ARGA
INDR3   ARGB,ARGC,ARGD
INDR3   ARGE,ARGF,NULL
.MACEXIT
.ENDIF
.IFNMA  8
WORD    ZBBUF   ; BUILD BUFFER AT ARGA
INDR1   ARGA
INDR3   ARGB,ARGC,ARGD
INDR4   ARGE,ARGF,ARGG,NULL
.MACEXIT
.ENDIF
.IFNMA  9
WORD    ZBBUF   ; BUILD BUFFER AT ARGA
INDR1   ARGA
INDR4   ARGB,ARGC,ARGD,ARGE
INDR4   ARGF,ARGG,ARGH,NULL
.MACEXIT
.ENDIF
.IFNMA  10
WORD    ZBBUF   ; BUILD BUFFER AT ARGA
INDR1   ARGA
INDR4   ARGB,ARGC,ARGD,ARGE
INDR5   ARGF,ARGG,ARGH,ARGI,NULL
        .MACEXIT
        .ENDIF
        .IFNMA  11
        WORD    ZBBUF   ; BUILD BUFFER AT ARGA
        INDR1   ARGA
        INDR5   ARGB,ARGC,ARGD,ARGE,ARGF
        INDR5   ARGG,ARGH,ARGI,ARGJ,NULL
        .MACEXIT
        .ENDIF
        .IFMA   11
        WORD    ZBBUF
        INDR1   ERROR   ; TOO MANY VARIABLES
        .MACEXIT
        .ENDIF
        .ENDM
;
INTREQ0 .MACRO  ARGA
        .IFNDEF XINTR0  ;INT 0 ENABLE/DISABLE
        EXTERN  XINTR0  ;INTO FIFO TEST MODE
        .ENDIF
        WORD    XINTR0
        INDR2   ARGA,0
        .ENDM
;
FIFORS  .MACRO
        .IFNDEF XTSTMOD ;RESETS FIFO PTRS AND PUTS SYSTEM
        EXTERN  XTSTMOD ;INTO FIFO TEST MODE
        .ENDIF
        WORD    XTSTMOD
```

```
        INDR2   0,0
        .ENDM
;
FIFOEN  .MACRO
        .IFNDEF XFIFEN  ;TAKE FIFO OUT OF RESET
        EXTERN  XFIFEN
        .ENDIF
        WORD    XFIFEN
        INDR2   0,0
        .ENDM
;
FIFMPTY .MACRO  ARGA
        .IFNDEF XMPTY   ;IF FIFO EMPTY FLAG IS SET (LOW)
        EXTERN  XMPTY   ;A 1 IS RETURNED TO ARGA(1:1)
        .ENDIF
        WORD    XMPTY
        INDR2   ARGA,0
        .ENDM
;
RSTEN   .MACRO
        .IFNDEF XRSTEN  ;IF FIFO FULL FLAG IS SET (LOW)
        EXTERN  XRSTEN  ;A 1 IS RETURNED TO ARGA(1:1)
        .ENDIF
        WORD    XRSTEN
        INDR2   0,0
        .ENDM
;
RSTDIS  .MACRO
        .IFNDEF XRSTDIS ;IF FIFO FULL FLAG IS SET (LOW)
        EXTERN  XRSTDIS ;A 1 IS RETURNED TO ARGA(1:1)
        .ENDIF
        WORD    XRSTDIS
        INDR2   0,0
        .ENDM
;
FIFFUL  .MACRO  ARGA
        .IFNDEF XFUL    ;IF FIFO FULL FLAG IS SET (LOW)
        EXTERN  XFUL    ;A 1 IS RETURNED TO ARGA(1:1)
        .ENDIF
        WORD    XFUL
        INDR2   ARGA,0
        .ENDM
;
RCHECK  .MACRO  ARGA,ARGB,ARGC
        .IFNDEF ZREP
        EXTERN  ZREP
        .ENDIF
        WORD    ZREP    ; REPEAT CHECK ARGB (ARGC TIMES), RETURN @ARGA
        INDR3   ARGA,ARGB,ARGC
        .ENDM
;
SCHECK  .MACRO  ARGA,ARGB,ARGC
        .IFNDEF ZSIZ
        EXTERN  ZSIZ
        .ENDIF
        WORD    ZSIZ    ; SIZE CHECK ARGB (ARGC TIMES), RETURN @ARGA
        INDR3   ARGA,ARGB,ARGC
        .ENDM
;
SWRITE  .MACRO  ARGA,ARGB,ARGC,ARGD
```

```
        .IFNDEF ZSW
        EXTERN  ZSW
        .ENDIF
        WORD    ZSW         ; SRAM@ARGA(1:ARGC) = ARGB(1:ARGC), GOTO ARGD ON NON-ERR
        INDR4   ARGA,ARGB,ARGC,ARGD
        .ENDM
;
SREAD   .MACRO  ARGA,ARGB,ARGC,ARGD
        .IFNDEF ZSR
        EXTERN  ZSR
        .ENDIF
        WORD    ZSR         ; ARGA(1:ARGC) = SRAM@ARGB(1:ARGC), GOTO ARGD ON NON-ERR
        INDR4   ARGB,ARGA,ARGC,ARGD
        .ENDM
;
        .CODE
TABL1   .SECTION OFFSET TABLADDR

        EQUATE  FIFOSIZ,256     ;SIZE OF FIFO
        EQUATE  BTP1,AAH        ;BIT TEST PATTERN 1
        EQUATE  BTP2,55H        ;BIT TEST PATTERN 2
        EQUATE  EFLG,10H        ;EMPTY FLAG MASK FOR P1
        EQUATE  FFLG,08H        ;FULL FLAG MASK FOR P1

        EQUATE  IV15,0FH        ;INTEGER VALUE 15
        EQUATE  IV255,FFH       ;INTEGER VALUE 255
        EQUATE  BITCOD4,80H     ;LEAD CODE WORD IN DATA STRING OF 4BIT DATA
        EQUATE  BITCOD8,00H     ;LEAD CODE WORD IN DATA STRING OF 8BIT DATA
        EQUATE  REPTCOD,A0H     ;LEAD CODE WORD IN DATA STRING OF RPT DATA
        EQUATE  NOCOD,C0H       ;LEAD CODE WORD IN DATA STRING OF 16BIT DATA
        EQUATE  CHANLCT,640     ;640 NUMBER OF BYTES IN A FIELD (20 CHANNELS X 3
        EQUATE  PAT1,AAH        ;SRAM BIT PATTERN
        EQUATE  PAT2,55H        ;SRAM BIT PATTERN
        EQUATE  MAXCNT,6000     ;MAXIMUM ALLOWABLE GAMMA COUNT FOR A 32 FRAME IN
        EQUATE  MAXTIM,210      ;MAXIMUM ALLOWABLE DELAY TIME

        EQUATE  RAMERR,$
        ASCII   THERE IS AN SRAM MEMORY ERROR AT|
        BYTE    ' '
        NOP

        EQUATE  MCKSUMERR,$
        ASCII   CKSUM ERROR AT BLOCK|
        BYTE    ' '
        NOP

        EQUATE  MFIFTST,$
        ASCII   FIFO TEST|
        NOP

        EQUATE  ENTDELAY,$
        ASCII   ENTER THE LOG TIME DELAY|
        NOP

        EQUATE  MRS_ERR,$
        ASCII   FIFO RESET ERROR, THE RD AND WR POINTERS DID NOT GO TO 0|
        NOP

        EQUATE  MNF_ERR,$
        ASCII   FIFO ERROR: MEMORY DID NOT GET FILLED|
```

```
NOP

EQUATE  MMEM_ERR,$
ASCII   FIFO ERROR: ERROR READING FIFO, VALUES DID NOT EQUAL|
NOP

EQUATE  MPRAMER,$
ASCII   TESTING OF PROGRAM RAM UNSUCCESSFUL|
NOP

EQUATE  MSMODERR,$
ASCII   INVALID COMMAND|
NOP

EQUATE  MFIF_DUM,$
ASCII   FIFO TEST IS COMPLETE AND SUCCESSFUL|
NOP

EQUATE  MTSTRAM,$
ASCII   TESTING PROGRAM RAM|
NOP

EQUATE  MRTDUN,$
ASCII   PROGRAM RAM TEST COMPLETE AND SUCCESSFUL|
NOP

EQUATE  MSMOD0,$
ASCII   MODE 0|
NOP

EQUATE  AKMOD0,$
ASCII   COMMAND TO SWITCH TOOL TO MODE 0 SENT|
NOP

EQUATE  MSMOD3,$
ASCII   MODE 3|
NOP

EQUATF  AKMOD3,$
ASCII   COMMAND TO SWITCH TOOL TO MODE 3 SENT|
NOP

EQUATE  MSMOD4,$
ASCII   MODE 4|
NOP

EQUATE  AKMOD4,$
ASCII   COMMAND TO SWITCH TOOL TO MODE 4 SENT|
NOP

EQUATE  MSMOD6,$
ASCII   MODE 6|
NOP

EQUATE  AKMOD6,$
ASCII   COMMAND TO SWITCH TOOL TO MODE 6 SENT|
NOP

EQUATE  MREQADDR,$
ASCII   PLEASE ENTER THE HEX ADDRESS|
```

```
NOP

EQUATE  MADRCHEK,$
ASCII   ADDRESS TEST|
NOP

EQUATE  MCSCHEK,$
ASCII   CS TEST|
NOP

EQUATE  MCELCHEK,$
ASCII   CELL TEST|
NOP

EQUATE  MCOMM,$
ASCII   SPECTRASCAN READY|
NOP

EQUATE  MECOMMOD,$
ASCII   ENTER COMM MODE|
NOP

EQUATE  MRTIME,$
ASCII   REAL TIME|
NOP

EQUATE  AKRTIME,$
ASCII   REAL TIME COMMUNICATION COMMAND RECEIVED|
NOP

EQUATE  MSTOR,$
ASCII   STORE DATA|
NOP

EQUATE  AKSTOR,$
ASCII   DATA STORE REQUEST RECEIVED|
NOP

EQUATE  MSTOP,$
ASCII   BACK|
NOP

EQUATE  AKSTOP,$
ASCII   STOP REQUEST RECEIVED|
NOP

EQUATE  MENTRDLY,$
ASCII   DELAY TIME NOT ENTERED|
NOP

EQUATE  AKDELAY,$
ASCII   DELAY TIME AND START TIME ENTERED|
NOP

EQUATE  MSIZERR,$
ASCII   THE VALUE ENTERED IS TOO LARGE|
NOP

EQUATE  RSTDLAY,$
ASCII   THE DELAY CLOCK HAS BEEN STOPPED|
```

```
NOP

EQUATE  MDUMP,$
ASCII   DUMP|
NOP

EQUATE  AKDUMP,$
ASCII   DATA DUMP REQUEST RECEIVED|
NOP

EQUATE  MTSTALL,$
ASCII   FULL MEMORY TEST|
NOP

EQUATE  AKTSTALL,$
ASCII   FULL TEST STARTING|
NOP

EQUATE  MSRTST,$
ASCII   MEMORY TEST|
NOP

EQUATE  MTLCMD,$
ASCII   C|
NOP

EQUATE  MLOCERR,$
ASCII   MEMORY LOCATION ERROR|
NOP

EQUATE  MONELOC,$
ASCII   LOC CHECK|
NOP

EQUATE  MTSTDUN,$
ASCII   THE TEST IS COMPLETE|
NOP

EQUATE  MSYSERR,$
ASCII   INVALID COMMAND SENT|
NOP

EQUATE  MCSERR,$
ASCII   CHIP SELECT ERROR AT|
BYTE    ' '
NOP

EQUATE  MTIME,$
ASCII   SEND TIME|
NOP

EQUATE  AKFIFTST,$
ASCII   FIFO TEST|
NOP

EQUATE  MRNGE,$
ASCII   VALUE OUT OF RANGE|
NOP

EQUATE  MDONE,$
```

```
ASCII   DONE|
NOP

EQUATE  MXMDATA,$
ASCII   XMDATA|
NOP

EQUATE  MREXDATA,$
ASCII   REXDATA|
NOP

EQUATE  MTMODERR,$
ASCII   TOOL SET TO MODE|
BYTE    ' '
NOP

EQUATE  MDELAY,$
ASCII   DELAY|
NOP

EQUATE  MSC,$
ASCII   SC |
NOP

EQUATE  NP,$
NOP

EQUATE  MTOOLCMD,$
ASCII   C|
NOP

EQUATE  MTEST,$
ASCII   T|
NOP

EQUATE  MCODERR,$
ASCII   SAY AGAIN!|
NOP

EQUATE  MDEVTST,$
ASCII   CHIP TEST
NOP

EQUATE  MAS2HXERR,$
ASCII   ASCII TO HEX CONVERSION ERROR|
NOP

EQUATE  MBCD2BINE,$
ASCII   BCD TO BINARY CONVERSION ERROR|
NOP

EQUATE  MOK,$
ASCII   OK|
NOP

EQUATE  RMSG,$
ASCII   R|
NOP

EQUATE  MSCTRN,$
```

```
ASCII   ASCII TRANSLATION ERROR|
NOP

EQUATE  MSAT,$
BYTE    ' @ '
NOP

EQUATE  NOTUND,$
ASCII   SAY AGAIN!|
NOP

EQUATE  MSTATUS,$
ASCII   STAT|
NOP

EQUATE  MSWPOS,$
ASCII   TOOL MODE|
BYTE    ' '
NOP

EQUATE  MWAIT,$
ASCII   WAIT, OBTAINING DATA|
NOP

EQUATE  MPRAMTST,$
ASCII   PGM RAM TEST|
NOP

EQUATE  MTSTP
ASCII   TEST IN PROGRESS|
NOP

EQUATE  MSBK,$
ASCII   BK|
NOP

EQUATE  MERRAT,$
ASCII   MEM ERROR AT|
NOP

EQUATE  MVALWAS,$
BYTE    ' VALUE WAS '
NOP

EQUATE  MSHBE,$
BYTE    ' SHOULD BE '
NOP

EQUATE  GMSG,$
ASCII   G|
NOP

EQUATE  MSBKV,$
ASCII   BREAK POINT|
NOP

EQUATE  MSBKNA,$
BYTE    ' '
EQUATE  MSBKNA1,$
ASCII   NOT ACTIVE|
```

```
NOP

EQUATE  PLF2CR,$
ASCII   >|
EQUATE  LF2CR,$
BYTE    0AH
EQUATE  LFCR,$
BYTE    0AH
EQUATE  CR,$
BYTE    0DH
NOP

EQUATE  MSTMEM,$
ASCII   TMEM|
NOP

EQUATE  MSWVER,$
ASCII   WRITE VERIFY ERROR|
NOP

EQUATE  MSWKG,$
ASCII   WORKING...|
NOP


EQUATE  QMSG,$
ASCII   Q|
NOP


EQUATE  MSMAIN,$
ASCII   MAIN|
NOP

EQUATE  MSMEX,$
ASCII   EX|
NOP


EQUATE  PLFCR,$
ASCII   >|
BYTE    0AH
BYTE    0DH
EQUATE  NULL,$
NOP

EQUATE  MDFRST
ASCII   DUMP TOOL FIRST|
NOP

EQUATE  MXMOD,$
ASCII   DATA COMPRESSION CHECK|
NOP

EQUATE  MNODUMP,$
ASCII   NOTHING TO DUMP|
NOP

EQUATE  MXMAIN,$
ASCII   XMAIN|
```

```
NOP

EQUATE  MFIFOK
ASCII   FIFO TEST SUCCESSFUL|
NOP

EQUATE  MWRTMOD
ASCII   INCORRECT TOOL MODE ENTERED|
NOP

EQUATE  MLOC,$
ASCII   LOCATION TEST IN PROGRESS|
NOP

EQUATE  ACKCEL,$
ASCII   MEMORY CELL CHECK IN PROGRESS|
NOP

EQUATE  ACKADR,$
ASCII   ADDRESS/DATA LINE TEST IN PROGRESS|
NOP

EQUATE  ACKCS,$
ASCII   CHIP SELECT TEST IN PROGRESS|
NOP

EQUATE MSPACE,$
BYTE    ' '
NOP

EQUATE  TLFCR,$
ASCII   <|
BYTE    0AH
BYTE    0DH
NOP

EQUATE  MDTIME,$
ASCII   DELAY TIME|
NOP

EQUATE  MSBCKS,$
ASCII   STARTING|
BYTE    ' '
EQUATE  MSCKS,$
ASCII   CHKSUM|
NOP

EQUATE  MMANYE,$
ASCII   TOO MANY ERRORS, TEST TERMITATED|
NOP

EQUATE  MCHKSUM,$
ASCII   CHEKSUM|
NOP

EQUATE  MPTMEM,$
ASCII   PGMRAM TEST|
NOP

EQUATE  MM,$
```

```
        ASCII   M|
        NOP

        EQUATE  LF,$
        BYTE    0AH
        NOP


        EQUATE  MSCKS1,$
        ASCII   CHECKSUM 1|
        NOP


        EQUATE  MSEQU,$
        BYTE    ' = '
        NOP

        EQUATE  MSNST,$
        BYTE    ' '
        EQUATE  MSNST1,$
        ASCII   NOT SET|
        NOP
;
; REV/DATE MESSAGE
;
RVDATM  ASCII   SCAN.|
        WORD    REV
        BYTE    ' '
        DATE
;
        EXTERN  SERNUM
;
MSSERN  ASCII TOOL SERIAL NUMBER = |
        LWORD   SERNUM
        NOP
;
;
;
;****;****;****;****;****;**     TEST FIFO ROUTINE       ****;****;****;****;
;
;
FIFOCHK FIFORS                          ;RESET FIFO AND PUT SYSTEM
                                        ;INTO FIFO TEST MODE
        SETS    FINFLG,00H              ;CLEAR FINAL LOOP FLAG
        FIFMPTY EMPTYFLG
        BNES    EMPTYFLG,1,RS_ERR
                                        ;CHECK THE EMPTY FLAG TO
                                        ;MAKE SURE THAT BOTH PO_NTERS
                                        ;ARE AT THE SAME POSITION

        SETS            RT0,BTP1        ;LOAD BIT TEST PATTERN 1
        SETS            RT1,BTP2        ;LOAD BIT TEST PATTERN 2
FIFOLOD SETD            CTR,FIFOSIZ

FIFOLD  XFER            SWREG,RT0,1     ;WR BTP TO SW REG
        XFER            FIFO,RT0,1      ;WR BTP FROM SW REG TO FIFO
        XFER            SWREG,RT1,1     ;WR NEXT BTP TO SW REG
        XFER            FIFO,RT1,1      ;WR BTP FROM SW REG TO FIFO
        DECDBN          CTR,2,FIFOLD    ;STAY IN LOOP UNTIL FIFO FULL
```

```
        SETD            BPTR,IPBUFF
        SETD            CTR,FIFOSIZ
FIFORD  FIFFUL          FULFLG              ;ERROR ROUTINE INDICATING MEM
        BEQS            FULFLG,0,NF_ERR     ;NOT FULL

        XFER            iBPTR,FIFO,1          ;XFER 1ST BYTE FROM FIFO TO RAM
        FIFFUL          FULFLG              ;ERROR ROUTINE INDICATING READ PTR
        BNES            FULFLG,0,NF_ERR     ;NOT MOVING
        INCD            BPTR,1

CHEKRD  XFER            iBPTR,FIFO,1          ;READ NEXT BYTE
        INCD            BPTR,1
        DECDBZ          CTR,2,CKDATA          ;CHEK FOR END OF MEM
        XFER            iBPTR,FIFO,1
        INCD            BPTR,1
        GOTO            CHEKRD

CKDATA  SETD            BPTR,IPBUFF
        SETD            CTR,FIFOSIZ
CHEKLP  BNES            iBPTR,iRT0,MEM_ERR ;BREAK ON COMP TO BTP ERROR
        INCD            BPTR,1
        BNES            iBPTR,iRT1,MEM_ERR
        INCD            BPTR,1
        DECDBZ          CTR,2,FINCHEK
        GOTO            CHEKLP

FINCHEK FIFMPTY EMPTYFLG
        BNES    EMPTYFLG,1,RS_ERR
                                    ;CHECK THE EMPTY FLAG TO
                                    ;MAKE SURE THAT BOTH POINTERS
                                    ;ARE AT THE SAME POSITION

        BEQS            FINFLG,01H,FIFODON
                                    ;CHEK FINAL LOOP FLAG
        SETS            FINFLG,01H  ;SET FINAL LOOP FLAG
        SETS            RT0,BTP2    ;LOAD BTP2
        SETSBR          RT1,BTP1,FIFOLOD;LOAD BTP1

FIFODON FIFOEN                      ;TAKE SYSTEM OUT OF FIFO TEST
                                    ;MODE AND ENABLE FOR NORMAL
                                        ;PROCESSING

        PRNT    RESP,MFIFOK,LFCR
        RETURN
;
MEM_ERR PRNT    RESP,MMEM_ERR,LFCR
        RETURN
NF_ERR  PRNT    RESP,MNF_ERR,LFCR
        RETURN
RS_ERR  PRNT    RESP,MRS_ERR,LFCR
        RETURN
;
;
;****;****;****;****;* COMPRESS AND STORE ROUTINE   ****;****;****;****;***
;
;
; SET ADDRESS POINTER AT BEGINNING OF PROGRAM
;
;
```

CA2183123

```
; WHEN INPUT BUFFER HALF FULL START ROUTINE
;
;****;****;****;**        INITIALIZE CNTRS, PNTRS, AND FLAGS      ****;****;***
;
          EQUATE   IBP,XTEMPA
          EQUATE   JX,XTEMPA+2
          EQUATE   PUTADR,XTEMPA+4
          EQUATE   TEMPA,XTEMPA+6
          EQUATE   TEMPB,XTEMPA+8
          EQUATE   TEMPC,XTEMPA+10
          EQUATE   TEMPD,XTEMPA+12
          EQUATE   TEMPE,XTEMPA+14
          EQUATE   RCOUNT,XTEMPA+16
          EQUATE   SCOUNT,XTEMPA+22
;
IRPT      EQU      A000H
IVLUR     EQU      6
ISZ4      EQU      8000H
ISZ8      EQU      0
IVLU4     EQU      12
IVLU8     EQU      6
;
COMPRS    SETD     IBP,iRDPTR                ; INPUT BUFFER POINTER
          SETD     TEMPE,iRDPTR              ; INPUT BUFFER POINTER
          INCD     TEMPE,640                 ; TEMPE = IBUFF_END+1
          SETD     PUTADR,OBUFF              ; OUTPUT BUFFER POINTER
;
; REPEAT CHECKS
;
REP1      SETD     TEMPA,iTEMPE              ; IBUFF_END+1
          DECD     TEMPA,iIBP                ; IBUFF_END+1-IBUFF_PTR
;
          RCHECK   RCOUNT,iIBP,iTEMPA        ; REPEAT CHECK
          BLTD     RCOUNT,2,SIZE1            ; RCOUNT .LT. 2?
          GOTO     REP4                      ; GO STORE
;
; REPEAT LOOP
;
REP2      SETD     TEMPA,iTEMPE              ; IBUFF_END+1
          DECD     TEMPA,iIBP                ; IBUFF_END+1-IBUFF_PTR
;
          RCHECK   RCOUNT,iIBP,iTEMPA        ; REPEAT CHECK
          SETD     TEMPD,IVLU8               ; TEMPD = IVLU8
          BEQS     RCOUNT+3,80H,REP3         ; EIGHT BIT REPEAT?
          SETD     TEMPD,IVLU4               ; TEMPD = IVLU4
REP3      BLTD     RCOUNT,iTEMPD,SIZE1       ; RCOUNT .LT. TEMPD?
;
; REPEAT COUNT DICTATES A STORE
;
REP4      IORS     iPUTADR,RCOUNT+1,IRPT     ; PUTADR = IRPT .OR. RCOUNT
          INCD     PUTADR,1                  ; PUTADR = PUTADR+2
          SETS     iPUTADR,iRCOUNT+2         ; PUTADR = REPEAT_DATA
          INCD     PUTADR,1                  ; PUTADR = PUTADR+1
          INCD     IBP,iRCOUNT               ; IBUFF_PTR = IBUFF_PTR+REPEAT_COUNT
          BLTD     IBP,iTEMPE,REP2           ; LOOP IF IBUFF_PTR .LE. IBUFF_END+1
          GOTO     SIZEX                     ; NO, STOP
;
; SIZE CHECKS
;
SIZE1     SCHECK   SCOUNT,iIBP,iTEMPA        ; SIZE CHECK
```

```
;
        SETD    TEMPD,IVLU8                     ; TEMPD = IVLU8
        BEQS    SCOUNT+2,80H,SIZE2              ; EIGHT BIT DATA?
        SETD    TEMPD,IVLU4                     ; TEMPD = IVLU4
SIZE2   BLTD    SCOUNT,iTEMPD,SIZE7             ; SCOUNT .LT. TEMPD?
;
; SIZE COUNT IS LARGE...
; LOOK FOR A REPEAT STRING IN THE BUFFER
;
SIZE3   SETD    JX,iIBP                         ; JX = IBUFF_PTR
        SETD    TEMPB,iIBP                      ; TEMPB = IBUFF_PTR
        INCD    TEMPB,iSCOUNT                   ; TEMPB = IBUFF_PTR+SIZE_COUNT
        SETD    TEMPC,iTEMPB                    ; TEMPC = IBUFF_PTR+SIZE_COUNT
        DECD    TEMPB,IVLU4                     ; TEMPB = IBUFF_PTR+SIZE_COUNT-IVLU4
        DECD    TEMPC,IVLU8                     ; TEMPC = IBUFF_PTR+SIZE_COUNT-IVLU8
;
SIZE4   RCHECK  RCOUNT,iJX,iTEMPA               ; REPEAT CHECK
        BEQS    RCOUNT+3,80H,SIZE5              ; EIGHT BIT REPEAT?
        BGED    RCOUNT,IVLU4,SIZE6              ; RCOUNT .GE. IVLU4?
        BGTD    JX,iTEMPB,SIZE7                 ; JX .GT. IBUFF_PTR+SCOUNT-IVLU4
        INCDBR  JX,1,SIZE4                      ; NO, CONTINUE
SIZE5   BGED    RCOUNT,IVLU8,SIZE6              ; RCOUNT .GE. IVLU8?
        BGTD    JX,iTEMPC,SIZE7                 ; JX .GT. IBUFF_PTR+SCOUNT-IVLU8
        INCDBR  JX,1,SIZE4                      ; NO, CONTINUE
;
SIZE6   DECD    JX,iIBP                         ; JX = JX-IBUFF_PTR
        SETD    SCOUNT,iJX                      ; SCOUNT = JX
;
SIZE7   SETD    TEMPD,ISZ8                      ; TEMPD = ISZ8
        BEQS    SCOUNT+2,80H,SIZE8              ; EIGHT BIT DATA
        SETD    TEMPD,ISZ4                      ; TEMPD = ISZ4
SIZE8   IORS    iPUTADR,SCOUNT+1,iTEMPD         ; PUTADR = ISZE .OR. SCOUNT
        INCD    PUTADR,1                        ; PUTADR = PUTADR+2
        BEQD    TEMPD,ISZ8,SIZEC                ; EIGHT BIT DATA
;
; FOUR BIT PACK AND STORE LOOP
;
SIZE9   FMT12   TEMPB,iIBP                      ; TEMPB = FMT12(IBUFF_PTR)
        SHL16   TEMPB,TEMPB,4                   ; TEMPB = SHL16(TEMPB,4)
        INCD    IBP,1                           ; IBUFF_PTR = IBUFF_PTR+1
        DECDBZ  SCOUNT,1,SIZEA                  ; SCOUNT = SCOUNT-1, BRANCH IF ZERO
        IORS    TEMPB+1,iIBP,iTEMPB+1           ; TEMPB+1 = IBUFF_PTR .OR. TEMPB+1
        INCD    IBP,1                           ; IBUFF_PTR = IBUFF_PTR+1
        DECD    SCOUNT,1                        ; SCOUNT = SCOUNT-1
SIZEA   SETS    iPUTADR,iTEMPB+1                ; PUTADR = TEMPB+1
        INCD    PUTADR,1                        ; PUTADR = PUTADR+1
        BNED    SCOUNT,0,SIZE9                  ; LOOP IF SCOUNT .NE. 0
        BLTD    IBP,iTEMPE,REP2                 ; LOOP IF IBUFF_PTR .LE. IBUFF_END+1
        GOTO    SIZEX                           ; FINISHED
;
; EIGHT BIT STORE LOOP
;
SIZEC   XFER    iPUTADR,iIBP,iSCOUNT            ; PUTADR(1:SCOUNT) = IBUFF_PTR(1:SCOUNT)
        INCD    PUTADR,iSCOUNT                  ; PUTADR = PUTADR+SCOUNT
        INCD    IBP,iSCOUNT                     ; IBUFF_PTR = IBUFF_PTR+SCOUNT
        BLTD    IBP,iTEMPE,REP2                 ; LOOP IF IBUFF_PTR .LE. IBUFF_END+1
;
; FINISHED WITH COMPRESSION, STORE THE DATA
;
SIZEX   SETD    TEMPA,iPUTADR                   ; TEMPA = PUTADR
```

```
        DECD    TEMPA,OBUFF                     ; TEMPA = PUTADR-OBUFF
        SWRITE  SRAMBLK,OBUFF,iTEMPA,CDUN
        SETS    COMMOD,00H
CDUN    XFER    LASTADDR,SRAMBLK,3
        RETURN
;
;****;****;****;****;****       END OF COMPRESSION SUB  ****;****;****;****
;
;
;****;****;****;****;****;*     CHEK SRAM       ****;****;****;****;****;**
;
;
; THIS ROUTINE CHECKS THE CELLS 512K X 8 SRAM DEVICES IN THE TOOL.
; SRAMCHEK
;
;
;
CELCHEK SETS    FINFLG,00H                      ;CLR FINAL LOOP FLAG
        SETS    RT0,BTP1                        ;WR FIRST PATTERN TO MEM
        SETS    RT1,BTP2                        ;WR SECOND PATTERM TO MEM

MCLOOP  SETS    SRADDR,00H                      ;SET SRAM ADDR = 0
        SETD    SRADDR+1,0000H
        SETS    SRBUF,00H
        SETS    ENDFLG,00H                      ;CLEAR END OF MEMORY FLAG
        SHL16   CSVALU,CSVALU,3                 ;SCALE CS VALU ENTERED FOR H BYTE
        SETS    SRADDR,iCSVALU+1
;
        SETS    SRBUF,iRT0                      ;WRITE FIRST PATTERN TO MEMORY
        SCALL   MEMWR
;
WRLOOP  SCALL   INCADDR
        SETS    SRBUF,iRT1                      ;LOAD PAT 2 TO SRAM BUFFER
        SCALL   MEMWR                           ;CALL MEMORY WRITE ROUTINE
        BEQS    ENDFLG,1,RDBACK
        SCALL   INCADDR
        SETS    SRBUF,iRT0                      ;LOAD PAT 1 TO SRAM BUFFER
        SCALL   MEMWR                           ;CALL MEMORY WR ROUTINE
        GOTO    WRLOOP
;
RDBACK  SETS    ENDFLG,00H                      ;CLEAR END OF MEMORY FLAG
        SETS    SRADDR,00H
        IORS    SRADDR,CSVALU+1,iSRADDR         ;RELOAD CS VALU
        SETD    SRADDR+1,0000H                  ;CLEAR LOWER BYTES
;
        SCALL   MEMRD                           ;READ FIRST VALUE FROM MEM
        BEQS    RT0,iSRBUF,RDLOP
        SETS    TEMP1,iRT0
        SCALL   ADDRERR
;
RDLOP   SCALL   INCADDR
        SCALL   MEMRD                           ;READ FROM SRAM
        BEQS    RT1,iSRBUF,CONT4                ;COMP VALU IF NOT EQUAL OP ERROR
        SETS    TEMP1,iRT1
        SCALL   ADDRERR
        BEQS    ERRFLG,1,CCDONE
;
CONT4   BEQS    ENDFLG,1,FINCHK
        SCALL   INCADDR
        SCALL   MEMRD                           ;READ NEXT VALUE FROM SRAM
```

```
        BEQS    RT0,SRBUF,RDLOP         ;COMP VALU IN NOT EQUAL OP ERROR
        SETS    TEMP1,iRT0              ;
        SCALL   ADDRERR
        BEQS    ERRFLG,1,CCDONE
        GOTO    RDLOP
;
FINCHK  BEQS    FINFLG,1,CCDONE         ;IF ALREADY DONE FINAL PASS THEN EXIT
        SETS    RT0,BTP2                ;LOAD PATTERN 2 FIRST THIS TIME
        SETS    RT1,BTP1                ;LOAD PATTERN 1
        SETSBR  FINFLG,1,MCLOOP         ;SET FINAL LOOP FLAG
;
;
CCDONE  RETURN
;
;
;****;****;****;****;****        CELL CHECK DONE ****;****;****;****;****;***
;
; THIS PORTION PERFORMS A TEST OF THE ADDRESS AND DATA LINES
;
;
ADRCHEK SETS    ENDFLG,00H              ;CLEAR END OF MEMORY FLAG
        SETS    SRADDR,00H              ;INIT SR ADDRESS & BUFFER
        SETD    SRADDR+1,0000H
        SETS    SRBUF,00H

SETCS   SHL16   CSVALU,CSVALU,3         ;SCALE CS VALU ENTERED FOR H BYTE
        IORS    SRADDR,CSVALU+1,iSRADDR
;
;
WRLOOP1 SCALL   MEMWR                   ;CALL MEMORY WRITE ROUTINE
        BEQS    ENDFLG,1,RDBAK1         ;IF MEM CHIP FULL
        SCALL   INCADDR
        INCSBR  SRBUF,1,WRLOOP1         ;LOAD PAT 2 TO SRAM BUFFER
;
;
RDBAK1  SETS    SRADDR,00H              ;INIT ADDR AND TEMP VALUE
        SETD    SRADDR+1,0000H
        IORS    SRADDR,CSVALU+1,iSRADDR
        SETS    TEMP1,00H
;
;
RDLOOP1 SCALL   MEMRD                   ;READ MEM LOCATION
        BEQS    SRBUF,iTEMP1,ENDCHK     ;COMP VALU IN MEM TO SUPPOSED VALU
        SCALL   ADDRERR
        BEQS    ERRFLG,1,BDON
ENDCHK  BEQS    ENDFLG,1,BDON
        SCALL   INCADDR
        INCSBR  TEMP1,1,RDLOOP1
;
;
;
; THIS PORTION OF THE ROUTINE CHECKS THE ADDRESS AND DATA LINES
;
;
LINCHEK SETS    SRADDR,00H              ;INIT SR ADDRESS & BUFFER
        SETD    SRADDR+1,0000H
        SETS    SRBUF,00H
        IORS    SRADDR,CSVALU+1,iSRADDR
```

```
         SCALL    MEMWR                          ;INITIAL WRITE TO MEMORY
         INCS     SRBUF,1
         SETD     SRADDR+1,0001H
;
WRLOOP2  SCALL    MEMWR                          ;WR VALUE TO MEMORY LOC
         INCS     SRBUF,1                        ;INC DATA VALUE
         INCD     SRADDR+1,iSRADDR+1             ;SHIFT ADDR LOC LEFT 1 BIT
         BNED     SRADDR+1,0000H,WRLOOP2         ;CHECK FOR SHIFT OUT OF LOWER
                                                 ;2 BYTES IF NOT STAY IN LOOP
                                                 ;ELSE
         SETS     SRADDR,01H                     ;SET HI BYTE
WRLOOP3  SCALL    MEMWR                          ;WR VALUE TO MEM
         INCS     SRADDR,iSRADDR                 ;SHIFT ADDR LOC LEFT ONE
         INCS     SRBUF,1                        ;INC DATA VALUE
         BNES     SRADDR,00H,WRLOOP3             ;CHECK FOR SHIFT OUT OF HI BYTE
;
         SETS     SRADDR,00H                     ;INIT SR ADDR AND DATA VALUE
         SETD     SRADDR+1,0000H
         IORS     SRADDR,CSVALU+1,iSRADDR
         SETS     TEMP1,00H
         SCALL    MEMRD                          ;INITIAL READ

         COMP     SRBUF,TEMP1,1,CONT6            ;COMP VALU READ TO SUPPOSED VALU
                                                 ;IF NOT SAME GOTO ERROR
         SCALL    ADDRERR
         BEQS     ERRFLG,1,BDON
CONT6    SETD     SRADDR+1,1

RDLOOP2  INCS     TEMP1,1                        ;INC COMPARE TO VALUE
         COMP     SRBUF,TEMP1,1,CONT2            ;COMP VALU READ TO SUPPOSED VALU
;                                                ;IF NOT SAME GOTO ERROR
         SCALL    ADDRERR
         BEQS     ERRFLG,1,BDON
CONT2    INCD     SRADDR+1,iSRADDR+1             ;SHIFT ADDR LOC LEFT ONE BIT
         BNED     SRADDR+1,0000H,RDLOOP2         ;IF SHIFT OUT
;
         SETS     SRADDR,01H                     ;SET HI BYTE
RDLOOP3  INCS     TEMP1,1                        ;INC COMPARE TO VALUE
         SCALL    MEMRD                          ;READ VALUE FROM MFM
         COMP     SRBUF,TEMP1,1,CONT3            ;COMP VALUE READ FROM MEM
         SCALL    ADDRERR
         BEQS     ERRFLG,1,BDON
CONT3    INCS     SRADDR,iSRADDR                 ;SHIFT ADD LOC LEFT ONE BIT
         BNES     SRADDR,00H,RDLOOP3             ;IF SHIFT OUT EXIT
;
BDON     RETURN
;
;
; THIS ROUTINE CHECKS THE CHIP SELECT LINES BY WRITING THE CS VALUE
; TO THE FIRST LOCATION IN MEMORY OF EACH CHIP.
;
;
CSCHEK   SETS     SRADDR,00H                     ;CLEAR SR ADDR AND BUFFER
         SETD     SRADDR+1,0000H
         SETS     SRBUF,00H

WRLOOP4  SCALL    MEMWR                          ;WRITE VALUE TO MEM DEVICE
         INCS     SRADDR,8H                      ;INC CS VALUE
         INCS     SRBUF,1                        ;INC VALUE WRITTEN TO CHIP
         IANDD    SRADDR,SRADDR,7800H                    ;CLEAR ALL BUT CS VALUE
```

```
        SETS    SRADDR+2,00H                    ;
        BNES    SRADDR,60H,WRLOOP4              ;IF CS VALU > 11 THEN OUT OF RANGE
;
;
        SETS    SRADDR,00H                      ;CLEAR SR ADDR AND BUFFER
        SETD    SRADDR+1,0000H
        SETS    SRBUF,00H
        SETS    TEMP1,00H
;
RDLOOP4 SCALL   MEMRD                           ;READ BACK VALUE
        COMP    SRBUF,TEMP1,1,NOERR             ;IF NOT CORRECT VALU GOTO ERROR
        SCALL   ADDRERR
        BEQS    ERRFLG,1,RDDONE

NOERR   INCS    SRADDR,8H                       ;INC CS VALUE
        INCS    TEMP1,1                         ;INC COMP TO VALUE
        BNES    SRADDR,60H,RDLOOP4              ;IF CS VALU > 11 THEN OUT OF RANGE

RDDONE  RETURN
;
; * ADDRESS ERROR ROUTINE *
;
; * ROUTINE CALLED WHEN AN ERROR OCCURS DURING A MEMORY TEST
;
;
ADDRERR CV2ASC  OUTBUF,SRADDR,0,3
        CV2ASC  OUTBUF+9,SRBUF,0,1
        CV2ASC  OUTBUF+12,TEMP1,0,1
        PRNT    RESP,RAMERR,OUTBUF,MVALWAS,OUTBUF+9,MSHBE,OUTBUF+12,LFCR
        INCS    ERRCTR,1
        BLTS    ERRCTR,10,ADON
        SETS    ERRFLG,1
        PRNT    RESP,MMANYE,LFCR
;
ADON    RETURN
;****;****;****;****;****;****  END CHEK SRAM    ****;****;****;****;****;***
;
;****;****;****;****;****;****  TEST ADDRESS     ****;****;****;****;****;***
;
;
ONELTST SETS    RT0,55H                         ;LOAD THE 1ST VALU TO BE TESTED
TSTLP   SETS    SRBUF,iRT0                      ;
        SCALL   MEMWR                           ;WRITE VALUE TO MEM
        SETS    SRBUF,00H
        DELAY   10                              ;DELAY READ BACK
        SCALL   MEMRD                           ;
        COMP    SRBUF,RT0,1,CONTST              ;COMP VALUE READ BACK WITH VALUE
                                                ;WRITTEN
        SETS    TEMP1,iRT0
        SCALL   ADDRERR

CONTST  BEQS    RT0,AAH,TSTDUN
        SETSBR  RT0,AAH,TSTLP
;
;
TSTDUN  RETURN


;****;****;****;****;****;****  WR DATA TO SRAM          ****;****;****;****;
```

```
MEMWR    BEQS     NOWRFLG,1,WRDONE
         XFER     ADDREG,SRADDR,3          ;LOAD SR ADDR REG'S W ADDR/CS
         XFER     SRDATA,SRBUF,1           ;WR DATA TO SRAM
WRDONE   RETURN
;
;
;
INCADDR  BEQS     NOWRFLG,01H,INCDON
         TBIT0    SRADDR,7,ENDCK           ;IF CHIP NOT FULL THEN GO TO
         BNED     SRADDR+1,FFFFH,ENDCK     ;ADDR INC SUB
                                           ;ELSE
;
;
;* SET END OF MEM CHIP FLAG, RESET ADDR TO 0 AND INC CS VALUE *
;
;
         SETS     ENDFLG,1                 ;SET END OF MEMORY CHIP FLAG
         GOTO     ENDCK
;
ENDCK    BLTS     SRADDR,60H,INCADR
         SETS     NOWRFLG,01H
         GOTO     INCDON
;
INCADR   INCD     SRADDR+1,1               ;INC MID AND LO BYTES OF SRAM ADDR
         BNED     SRADDR+1,0000H,INCDON        ;IF NOT =0 (ROLL OVER) THEN EXIT
         INCS     SRADDR,1                 ;ELSE INC UPPER 3BITS OF ADDR
;
;
INCDON   RETURN
;
;
;****;****;****;****;****      SRAM RD SUB     ****;****;****;****;****;**
;
;
MEMRD    XFER     ADDREG,SRADDR,3          ;LOAD SR ADDR REG'S W ADDR/CS
         XFER     SRBUF,SRDATA,1           ;WR DATA TO SRAM BUFFER
;
;
ENDOP    RETURN
;
;****;****;****;****;****;      END SRAM RD SUB ****;****;****;****;****

;****;****;****;****;****      O/P DATA TO PC  ****;****;****;****;****
;
; THE FOLLOWING 4 ROUTINES HANDLE THE DATA TRANSFER FROM THE TOOL TO THE
; PC.
;
;****;****;****;****;****;****;****;****;****;****;****;****;****;****;*
;
;****;****;****          TRANSFER DATA          ****;****;****;****;****
;
; THIS SUB ACTUALLY TRANSMITS ONE 512 BYTE DATA BLOCK FROM THE TOOL TO THE PC.
; IT IS CALLED BY THE DUMP, XMIT, AND REXMIT SUBS.
;
;
TRDATA   SETD     OPTR,OUTTEMP             ;INIT OP PTR TO START OF OP BUFFER
         SETD     DATCTR,0000H

         SETD     CHKSUMV,0000H             ;CLEAR CHECK SUM VALUE
         SETD     CHKSUMV+2,0000H
```

```
        BNES    OPFLG,00H,POBUF1        ;CHECK OUTPUT FLAG, IF SET POST
        NPOSTO  OUTBUF                  ;OUTBUF1 ELSE POST OUTBUF
        SETS    XMBUF,00H
        GOTO    RDDATA
POBUF1  NPOSTO  OUTBUF1
        SETS    XMBUF,01H

RDDATA  SREAD   OUTTEMP,SRAMBLK,512,SETDC1      ;READ DATA FROM SRAM TO
        SETS    LTIMFLG,01H                     ;OUT BLOCK
SETDC1  SETD    DATCTR,iSRAMBLK+6
;
;
; WHEN OP BUFFER IS FULL THEN WR BUFFER TO RS232.
;
;
TRCON   BNES    OPFLG,00H,OUT1
        CV2ASCL OUTBUFH,OUTTEMP,iDATCTR
        INCD    DATCTR,iDATCTR
        CHKSUM  CHKSUMV,OUTBUFH,iDATCTR
        CV2ASC  TEMP5,CHKSUMV,0,4
        SETS    OPFLG,01H
        BEQS    LTIMFLG,01H,LPRT1
        BBUF    OUTBUF1,RESP,OUTBUFH,MSPACE,TEMP5,TLFCR
        RETURN
LPRT1   BBUF    OUTBUF1,RESP,OUTBUFH,MSPACE,TEMP5,PLFCR
        SETS    LBUFLG,01H
        RETURN
;
;
OUT1    CV2ASCL OUTBUFH,OUTTEMP,iDATCTR
        INCD    DATCTR,iDATCTR
        CHKSUM  CHKSUMV,OUTBUFH,iDATCTR
        CV2ASC  TEMP5,CHKSUMV,0,4
        SETS    OPFLG,00H
        BEQS    LTIMFLG,01H,LPRT
        BBUF    OUTBUF,RESP,OUTBUFH,MSPACE,TEMP5,TLFCR
        RETURN
LPRT    BBUF    OUTBUF,RESP,OUTBUFH,MSPACE,TEMP5,PLFCR
        SETS    LBUFLG,01H
        RETURN
;
;
;****;****;****;****;*  TSCAN MEMORY DUMP SUB   ****;****;****;****;**
;
; DATA DUMP SUB.  THIS IS THE ROUTINE CALLED WHEN THE USER ISSUES THE
; COMMAND TO TRANSFER ALL COMPRESSED SPEC CHANNEL DATA IN TOOL TO PC.
;
;
DUMP    SETS    SRADDR,00H              ;INITIALIZE HI BYTE OF SR ADDR
        SETD    SRADDR+1,0000H          ;INITIALIZE LO AND MID BYTES
        XFER    SRAMBLK+3,LASTADDR,3
        SWEEP   SRAMBLK,0,3
        SETS    LTIMFLG,00H             ;CLEAR END OF DATA FLAG
        SETS    OPFLG,00H               ;CLEAR CHECK SUM TOTAL
        SETD    DATCTR,0000H            ;CLEAR DATA COUNTER
        SETS    LBUFLG,00H
        SETD    OPTR,OUTTEMP
```

```
;
;
          SREAD    OUTTEMP,SRAMBLK,512,SETDC
          SETS     LTIMFLG,01H
SETDC     XFER     DATCTR,SRAMBLK+6,2
;
TRBUF1    CV2ASCL  OUTBUFH,OUTTEMP,iDATCTR
          INCD     DATCTR,iDATCTR
          CHKSUM   CHKSUMV,OUTBUFH,iDATCTR
          CV2ASC   TEMP5,CHKSUMV,0,4
          SETS     OPFLG,01H
          BEQS     LTIMFLG,01H,LPRTT
          BBUF     OUTBUF1,RESP,OUTBUFH,MSPACE,TEMP5,TLFCR
          GOTO     TRDATE
LPRTT     BBUF     OUTBUF1,RESP,OUTBUFH,MSPACE,TEMP5,PLFCR
          SETS     LBUFLG,01H
;
TRDATE    I2BCD1   TEMP5,DTIME+1                          ;RECONV VALUE AND XMIT BACK TO
          CV2ASC   TEMP5,TEMP5+1,0,2                      ;USER TO VERIFY
          BPOSTN   OUTBUF,DAT,MSPACE,CLKTIM,MSPACE,TEMP5
          CHKSUM   CHKSUMV,OUTBUF,24
          SETS     TEMP1,ZERO
          XFER     TEMP5,TEMP1,10
          CV2ASC   TEMP5,CHKSUMV,0,4
;
          PRNT     RESP,OUTBUF,MSPACE,TEMP5,TLFCR           ;PRINT FIRST DATA
;
;
TRDDUN    RETURN
;
;
;****;****;****;***      END DUMP SUB     ****;****;****;****;****;****;
;
;****;****;****;****;** START RETRANSMIT SUB     ****;****;****;****;***
;
;
; THIS ROUTINE IS CALLED IF A CHKSUM ERROR OCCURED.  IT RETRANSMITS THE
; BLOCK OF DATA JUST XMITED ALONG WITH AN ERROR MESSAGE.
;
;
REXMIT    BNES     XMBUF,00H,RXOUT1
          NPOSTO   OUTBUF
          GOTO     RXDUN
;
RXOUT1    NPOSTO   OUTBUF1
;
RXDUN     RETURN
;
;
;****;****;****;***      END REXMIT SUB   ****;****;****;****;****;****;**
;
;****;****;****;***      START XMIT SUB   ****;****;****;****;****;****;**
;
;
; WHERE CONTINUATION OF XMIT STARTS UPON VALIDATION OF CHKSUM
;
;
XMIT      XFER     TEMPSRA,SRADDR,3                  ;SAVE START LOCATION OF NEXT BLOCK
          SCALL    TRDATA                          ;CALL XMIT DATA SUB
;
```

```
;
;
XMITDUN RETURN
;
;****;****;****;          END XMIT SUB     ****;****;****;****;****;****;***
;
;****;****;****;          STOP LOG SUB     ****;****;****;****;****;****;***
;
;
;THIS ROUTINE SETS THE STOP FLAG USED TO INDICATE THE END OF PROCESSING OF
;SPECTRAL DATA.
;
;
STLOG   SETS    STOPFLG,1                       ;SET STOP PROCESSING DATA FLAG
;
        RETURN
;
;
;****;****;****;***      END STOP LOG SUB          ****;****;****;****;****;***
;
;****;****;****;****;*   REAL TIME COMM WITH SURFACE      ****;****;****;****;
;
;
;THIS SUB IS USED TO COMMUNICATE WITH THE PC REAL TIME.  SO THE USER CAN VIEW
;THE SPECTRAL CHANNEL DATA DURING CALIBRATION AS IT IS BEING GENERATED.
;
;
;
RTCOMM  XFER    OPTR,XPTR,2                     ;INITIALIZE OP BUFFER PTR TO
                                                ;OP BUFFER BASE ADDR
;
RDRT    CV2ASC  OUTBUF,iOPTR,0,256              ;CONVERT ONE FIELD OF DATA
        PRNT    RESP,OUTBUF                     ;(640 CHANNELS) TO ASCII AND XMIT
        INCD    OPTR,256                        ;TO SURFACE
        CV2ASC  OUTBUF,iOPTR,0,256
        PRNT    OUTBUF
        INCD    OPTR,256
        CV2ASC  OUTBUF,iOPTR,0,128
        PRNT    OUTBUF,PLFCR
;
;
        RETURN
;
;
;****;****;****;****;****;***   END REAL TIME COMM      ****;****;****;****;
;
;
;****;****;****;****;****;***   SW MODE SETTINGS        ****;****;****;****;*
;
;
; THIS SUB IS TO PUT THE TOOL INTO SW MODE 0
;
;
SWMOD0  SETS    SWREG,MOD0                      ;INITIAL POWER UP MODE
;
        RETURN
;
;
SWMOD3  SETS    SWREG,MOD3                      ;STABILIZATION MODE
```

```
        RETURN
;
;
SWMOD4  SETS    SWREG,MOD4                  ;LOG/CALIBRATION MODE

        RETURN
;
;
SWMOD6  SETS    SWREG,MOD6                  ;STABILIZER LATCHED MODE

        RETURN
;
;
;****;****;****;***     GAMMA COUNT ADDER       ****;****;****;****;****;**
;
;
; THIS SUB ADDS THE GAMMA COUNT FROM EACH FRAME AND KEEPS A RUNNING TOTAL
; FOR THE RESPECTIVE FIELD.
;
;
GAMCNT  SETS    FCTR,06H                            ;CLEAR FRAME COUNTER
        SETD    GAMTOT,00H                          ;CLEAR GAMMA COUNTER
        SETD    TEMP2,iSUMPTR
        INCD    TEMP2,CH437
        SETD    TEMP1,iSUMPTR
        INCD    TEMP1,SCOFSET
        XFER    RT0,iTEMP1,1                ;MOVE SCALE FACTOR FOR FIELD
                                            ;TO TEMP VALUE
        SETD    TEMP1,00H                   ;CLEAR TEMP1 FOR LATER USE
        SETD    W1PTR,iSUMPTR
        INCD    W1PTR,120
;
;
GAMLOOP SETS    TEMP3,00                    ;CLR TEMP TO BE USED AS CTR
        SCALL   SPFRAM                      ;ADD GAMMA FROM CHANNEL TO
;                                           ;TOTAL COUNT
        INCS    TEMP3,1                     ;INC CHANNEL COUNTER
;
        BEQS    FCTR,15,OPFRAM              ;CHECK FOR OPERATIONAL FRAMES
        BEQS    FCTR,30,XITGAM
;
SCFRAM  SCALL   SPFRAM
;
        INCS    TEMP3,1                     ;INC CHANNEL COUNTER
        BNES    TEMP3,17,SCFRAM             ;CHECK FOR END OF SPEC CHAN
;                                           ;SECTOR
        INCD    W1PTR,1                     ;INC GAMMA SUM. PTR TO NXTO
                                            ;LAST SPEC DATA CHAN IN FRAME
        SCALL   SPFRAM                      ;ADD CHANNEL COUNT TO TOTAL
        INCS    FCTR,1                      ;INCREMENT FRAME COUNTER
        INCDBR  W1PTR,1,GAMLOOP             ;SET PTR TO START OF NXT FRAM
;
OPFRAM  INCS    FCTR,1                          ;INC FRAME COUNTER
        INCD    W1PTR,19                        ;MOVE PTR TO START OF NEXT
                                                ;FRAME
        BNES    FCTR,32,GAMLOOP                 ;IF LAST FRAME EXIT SUB

XITGAM  RETURN
;
```

```
;
;****;****;****;****;****;***   INC GAMMA COUNT ****;****;****;****;***;****
;
;
; THIS ROUTINE INC THE SUM PTR TO THE NEXT SPECTRAL CHANNEL VALUE AND ADDS
; IT TO THE TOTAL GAMMA COUNT.
;
;
SPFRAM  INCD    W1PTR,1                          ;SPECTRAL FRAME LOOP
        SETS    TEMP1,00H
        XFER    TEMP1+1,iW1PTR,1                 ;LOAD SPEC CHANNEL VALUE
        BGED    W1PTR,iTEMP2,ADDGAM              ;CHECK FOR HI CH 102
        BEQS    RT0,00H,ADDGAM                   ;CHEK FOR 0 SCAL FACTOR
        SHL16   TEMP1,TEMP1,iRT0                 ;SHIFT VALUE LEFT BY APPLIED
;
ADDGAM  INCD    GAMTOT,iTEMP1                    ;KEEP RUNNING TOTAL OF GAMMA
;
        RETURN
;
;
;****;****;****;****;****;****  CHECK GAMMA COUNT  ****;****;****;****;****;*
;
;
CHEKCNT BLTD    GAMTOT,MAXCNT,CNTOK      ;COMPARE THE GAMMA COUNT WITH THE MAX
        SCALL   SWMOD6                   ;PUT TOOL INTO LOCKED MODE
        SETSBR  LOKMOD,1,CNTDUN          ;SET LOCK MODE FLAG
;
CNTOK   BEQS    LOKMOD,0,CNTDUN          ;IF TOOL NOT IN LOCK MODE THEN GO TO
                                         ;EXIT SUB
        SCALL   SWMOD4                   ;ELSE IF TOOL IN LOCKED MODE PUT
                                         ;BACK TO LOG MODE
CNTDUN  RETURN
;
;
;****;****;****;****;****;***   END GAMMA COUNT ****;****;****;****;****;****
;
;
;
;****;****;****;****;****;*     END SAVE INPUT  ****;****;****;****;****;****
;
        SUBTITLE *** BREAK POINT HERE ***
        PAGE
;
PBRK    ASCMP   LF,iBPTR1,PBRK1
        ASCMPE  CR,iBPTR1,PBRK4
PBRK1   SPOS    BREAK,PBRK3
PBRK2   CV2ASC  BREAK+9,BREAK+1,0,2
        PRNTBR  iSPTR2,MSBKV,MSEQU,BREAK+9,PLF2CR,iSPTR1
PBRK3   PRNTBR  iSPTR2,MSBKV,MSNST,PLF2CR,iSPTR1
PBRK4   ASCMPE  RMSG,iBPTR1,PBRK5
        SETS    BREAK,0
        PRNTBR  iSPTR2,MOK,PLF2CR,iSPTR1
PBRK5   ASCMPE  GMSG,iBPTR1,PBRK6
        PRNT    BKRESP,MOK,PLF2CR
        DELAY   1
        INTERV  RESTORE
        INTRPT  RESTORE
        BRET    BREAK+5
PBRK6   AS2HX   BPLOOP,iBPTR1,PBRK7,1,2
        PRNTBR  iSPTR2,MSCTRN,PLF2CR,iSPTR1
```

```
PBRK7    SETD     BREAK+1,iBPLOOP
         SETS     BREAK,80H
         XFER     BKRESP,iSPTR2,4
         SETDBR   BREAK+3,PBRK8,PBRK2
;
; ENTRY HERE WHEN BREAK POINT HIT
;
PBRK8    INTERV   SAVE/DISABLE
         INTRPT   SAVE/ENABLE
         SETD     BREAK+7,iBREAK+1
         CV2ASC   BREAK+9,BREAK+1,0,2
         SETD     SPTR1,PBRK9
         SETD     SPTR2,BKRESP
         PRNT     iSPTR2,MSBKV,MSAT,BREAK+9,PLF2CR
PBRK9    QINPUT   BPBUF,BSIZE,PBRK10,0
         GOTO     PBRK9
PBRK10   BNES     BKRESP+1,iBPBUF,PBRK9
         ASCMPE   MSMEX,BPBUF+3,PBRK11
         SETDBR   MPTR2,BPBUF+5,PMEX
PBRK11   ASCMPE   MSBK,BPBUF+3,PBRK12
         SETDBR   BPTR1,BPBUF+5,PBRK
PBRK12   PRNTBR   iSPTR2,NOTUND,PLF2CR,PBRK9
;
         SUBTITLE *** MEMORY EXAMINED HERE ***
         PAGE
;
PMEX     AS2HX    MLOOP1,iMPTR2,PMEX2,1,2
PMEX1    PRNTBR   iSPTR2,MSCTRN,PLF2CR,iSPTR1
PMEX2    SETD     MPTR1,iMLOOP1
PMEX3    CV2ASC   MOUTBF,MPTR1,7F02H
         SETS     MOUTBF+4,'='
         CV2ASC   MOUTBF+5,iMPTR1,7F01H
         SETD     MOUTBF+7,203CH
         SETS     MOUTBF+9,0
PMEX4    PRNT     iSPTR2,MOUTBF
         UINPUT   iMPTR2,BSIZE,PMEX5,0
         GOTO     PMEX4
PMEX5    ASCMP    LFCR,iMPTR2,PMEX6
         ASCMPE   CR,iMPTR2,PMEX7
PMEX6    INCDBR   MPTR1,1,PMEX3
PMEX7    ASCMPE   QMSG,iMPTR2,PMEX8
         PRNTBR   iSPTR2,MOK,PLF2CR,iSPTR1
PMEX8    DECD     MPTR2,1
         SETS     iMPTR2,SPACE
         AS2HXE   MLOOP1,iMPTR2,PMEX1,1,2
         INCD     MPTR2,1
         BLED     MLOOP1,0FFH,PMEX9
         PRNTBR   iSPTR2,NOTUND,PLF2CR,iSPTR1
PMEX9    WRITBE   iMPTR1,MLOOP1+1,1,MERR,PMEX10
         INCDBR   MPTR1,1,PMEX3
PMEX10   PRNTBR   iSPTR2,MSWVER,PLF2CR,iSPTR1
;
;****;****;****;****;****;       PROGRAM SRAM TEST        ****;****;****;****;*
;
         SUBTITLE *** RAM MEMORY TESTED AND ROM CHECKSUM HERE ***
         PAGE
;
PTMEM    PRNT     RESP,MTSTP,PLFCR
         SERIN    DISABLE
         RAMTST   PGMRAM,RAMHLF,MERR,THAF2
```

```
        PRNT    RESP,MERRAT,MERR,MVALWAS,MERR+5,MSHBE,MERR+8,PLFCR
THAF2   RAMTST  PGMRAM2,RAMHLF,MERR,TDUN
        PRNT    RESP,MERRAT,MERR,MVALWAS,MERR+5,MSHBE,MERR+8,PLFCR
TDUN    PRNTBR  RESP,MSBCKS,LFCR,PCHKS1
;
; CHECKSUM PERFORMED HERE, PRINT WORKING MESSAGE
;
PCHKS   PRNTBR  RESP,MSWKG,LFCR,PCHKS1
;
; GENERATE CHECKSUMS AND PRINT (ENTRY POINT FOR END OF RAM TEST ROUTINE)

PCHKS1  CHKSUM  TEMP10,MAINADDR,PGMRAM-MAINADDR
        CV2ASC  OUTBUF,TEMP10,7F04H
        PRNT    RESP,MSCKS1,MSEQU,OUTBUF,PLF2CR
        RETURN
;
;****;****;****;****;****;**    PROGRAM EEPROM  ****;****;****;****;****;***
;
        SUBTITLE *** ONE CARD LOAD PERFORMED HERE ***
        PAGE
;
PLOAD   INTERV  DISABLE
        SETD    XADDR,FFFFH
        PRNT    RESP,MOK,PLF2CR
;
PLOAD1  QINPUT  INBUF,ISIZE,PLOAD2,0
        GOTO    PLOAD1
;
; CHECK FOR COLON
;
PLOAD2  BNES    INBUF+3,':',PLOAD8
;
; CONVERT ADDRESS
;
        SETS    STRTMP,' '
        SETS    STRTMP+5,0DH
        XFER    STRTMP+1,INBUF+6,4
        AS2HXE  BUFTMP+2,STRTMP,PLOAD8,1,2
;
; CONVERT RECORD TYPE
;
        SETS    STRTMP+3,0DH
        SETD    STRTMP+1,iINBUF+10
        AS2HXE  BUFTMP+4,STRTMP,PLOAD8,1,2
;
; CONVERT RECORD LENGTH AND ERROR CHECK
;
        SETD    STRTMP+1,iINBUF+4
        AS2HXE  BUFTMP,STRTMP,PLOAD8,1,2
        BGTD    BUFTMP,16,PLOAD8
;
; CONVERT DATA
;
        SETD    RECL,iBUFTMP
        SETD    PTRX1,INBUF+12
        SETD    PTRX2,BUFTMP+6
;
; CONVERSION LOOP
;
PLOAD3  XFER    STRTMP+1,iPTRX1,2
```

```
        AS2HXE   STRTMP+4,STRTMP,PLOAD8,1,2
        SETS     iPTRX2,iSTRTMP+5
        BEQD     RECL,0,PLOAD4
        INCD     PTRX1,2
        INCD     PTRX2,1
        DECDBR   RECL,1,PLOAD3
;
; CHECK CHECKSUM
;
PLOAD4  SETD     RECL,iBUFTMP
        INCD     RECL,6
        CHKSUM   STRTMP,BUFTMP,iRECL
        INCD     STRTMP+2,iSTRTMP+4
        BNES     STRTMP+3,0,PLOAD8
;
; IF RECORD TYPE IS ZERO, TRANSFER LOADER TO RAM
;
        BNED     BUFTMP+4,0,PLOAD5
        XFER     iBUFTMP+2,BUFTMP+6,iBUFTMP
        PRNTBR   RESP,MSACK,PLF2CR,PLOAD1
;
; IF RECORD TYPE IS THREE, CONTAINS TRANSFER ADDRESS
;
PLOAD5  BNED     BUFTMP+4,3,PLOAD6
        BNED     BUFTMP,2,PLOAD8
        SETD     XADDR,iBUFTMP+6
        PRNTBR   RESP,MSACK,PLF2CR,PLOAD1
;
; IF RECORD TYPE IS ONE, TERMINATE
;
PLOAD6  BNED     BUFTMP+4,1,PLOAD8
        BNED     BUFTMP,0,PLOAD8
;
; CHECK FOR TRANSFER ADDRESS
;
        BNED     XADDR,FFFFH,PLOAD7
        PRNTBR   RESP,MSNAK,PLF2CR,MLOOP
;
; DISABLE INTERRUPTS AND VECTOR THROUGH TRANSFER ADDRESS
;
PLOAD7  INTRPT   DISABLE
        VECTOR   iXADDR
;
; ERRORS HANDLED HERE
;
PLOAD8  PRNTBR   RESP,MSNAK,PLF2CR,PLOAD1
;
        EQUATE   MSACK,$
        ASCII    ACK|
        NOP
;
        EQUATE   MSNAK,$
        ASCII    NAK|
        NOP
;
        EQUATE   MSOCRD,$
        ASCII    ONECARD|
        NOP
;
;****;****;****;****;****;****;**        END PROGRAM EEPROM        ****;****;***
```

```
;
;****;****;****;****;****;****  MAIN  PROG   ****;****;****;****;****;****;**
;       ;;  * INIT OF MAIN *
;
        SUBTITLE TABLES
        PAGE
;
        INDADR  TBL1
;
        SWEEP   PGMRAM,00H,RAMSIZ
        SETS    DHOLFLG,00H
        XFER    RESP,MSC,4                  ;LOAD RESPONSE STRING
        SETD    SUMPTR,IPBUFF               ;INIT SUM PTR
        SETD    RDPTR,IPBUFF                ;INIT READ POINTER TO DATA BUFF
        SETD    XPTR,IPBUFF
;
; INITIALIZE TOOL BY PUTTING INTO SW POS 0 AND STARTING 400 HZ PULSE SIGNAL
; GENERATOR USED AS AN INTERRUPT FOR OTHER CPU.
;
        FIFOEN
        SCALL   SWMOD0                      ;WR 0 TO SWITCH POS REG
        SETS    DUMPFLG,01H
        FTIMER  ENABLE,FFH,F883H            ;ENABLE 400 HZ INT GENERATOR
        DELAY   4
        IPOST   QINPUT,INBUF,ISIZE,1        ;SET INPUT POST
        PRNT    RESP,MCOMM,PLFCR            ;SEND READY MESSAGE
;
;
;     *  MAIN PROGRAM LOOP *
;
;
MLOOP   IPOST   QINPUT,INBUF,ISIZE,1        ;SET INPUT POST
MLOOP2  SNEG    OUTPUT,CKFLD                ;WHILE OUTPUTING CKFLD
                                            ;ELSE CHECK INPUT
        SNEG    INPUT,CKFLD                 ;CHECK INPUT POST
        BNES    INPUT,40H,MLOOP             ;INPUT ERROR
        GOTO    TOOLCMD                     ;GOTO COMMAND TABLE
;
; * CHECK IF DATA BUFFER HAS A FULL FIELD OF DATA *
;
CKFLD   BEQS    FLDFUL,00H,MLOOP2           ;CHECK DATA BUFFER FOR A FULL FIELD
                                            ;OF DATA (32 FRAMES)
        BEQS    FLDFUL,01H,RDAIP            ;IF 1ST HALF OF IP BUFF FULL THEN
                                            ;SET PTRS TO START OF BUFF
                                            ;ELSE
        SETD    RDPTR,IPBUFF+HAFDAT          ;MOVE RD AND SUM PTRS TO START OF 2ND
        SETD    XPTR,IPBUFF+HAFDAT           ;FIELD, 2ND HALF OF BUFFER
        SETD    SUMPTR,IPBUFF+HAFDAT
        SETSBR  FLDFUL,00H,COMMCHK          ;CLEAR FIELD FULL FLAG
;
RDAIP   SETD    RDPTR,IPBUFF                ;SET SUM & RD PTRS TO START OF IP BUF
        SETD    XPTR,IPBUFF
        SETD    SUMPTR,IPBUFF
        SETS    FLDFUL,00H                  ;CLEAR FIELD FULL FLAG
;
COMMCHK BNES    COMMOD,00H,COMTYP           ;CHECK TO SEE IF THE COMM MODE IS SET
        GOTO    MLOOP2
;
COMTYP  BEQS    COMMOD,RTIMFLG,RTIM         ;CHECK REAL TIME COMM FLAG
                                            ;(WRITE DATA TO SERIAL PORT)
```

CA2183123

```
;
; * STORE DATA MODE *
;
        BNES    COMMOD,COMPFLG,XMCHEK   ;CHECK COMPRESS FLAG
                                        ;(STORE DATA IN TOOL)
COMPR   ;SCALL   GAMCNT                  ;TOTAL GAMA CNT FOR SPEC CHANNELS
        ;SCALL   CHEKCNT                 ;CHECK FOR HIGH GAMMA COUNT
        SCALL   COMPRS                  ;COMPRESS DATA
        GOTO    MLOOP2
;
XMCHEK  BNES    COMMOD,XMFLG,MLOOP
;
;
; * XMAIN TO CHECK COMPRESSED DATA WITH RTIME DATA *
;
CXMAIN  SCALL   RTCOMM
        SCALL   COMPRS
        GOTO    MLOOP2

;
; * REAL TIME COMM MODE *
;
RTIM    SETS    COMMOD,00H              ;CLEAR COMM MODE INDICATOR
        SCALL   RTCOMM                  ;XMIT DATA REAL TIME OUT SERIAL PORT
        GOTO    MLOOP
;
;
; ****;**** COMMAND PORTION OF THE PROGRAM ****;****;****;*
;
;
; ****;**** TOOL COMMAND SECTION ****;***
;
;
TOOLCMD ASCMPE  MTLCMD,INBUF+3,MEMTST   ;CHECK FOR TYPE OF CMND TOOL/MEMORY
;
; * REAL TIME COMM *
;
RTIME   ASCMPE  MRTIME,INBUF+5,TMOD0            ;XMIT DATA REAL TIME
        SETSBR  COMMOD,RTIMFLG,MLOOP            ;SET COMM MODE TO REAL TIME
                                                ;GOTO MAIN LOOP
;
; * SWITCH POSITION 0 *
;
TMOD0   ASCMPE  MSMOD0,INBUF+5,TMOD3    ;INITIALIZATION
        SETS    COMMOD,00H
        SCALL   SWMOD0                  ;SET TOOL TO SWMODE 0
        PRNT    RESP,MWAIT,LFCR         ;PRINT WAIT MESSAGE
        DELAY   8                       ;WAIT FOR TOOL TO SWITCH
        SETD    TEMP1,iRDPTR
        INCD    TEMP1,SPOFSET
        CV2ASC  TMODA,iTEMP1,0,1
        PRNT    RESP,MSWPOS,TMODA+1,LFCR    ;OUTPUT SWITCH POSITION
        PRNTBR  RESP,MDONE,PLFCR,MLOOP

; * SWITCH POSITION 3 *
;
TMOD3   ASCMPE  MSMOD3,INBUF+5,TMOD4    ;STABILIZATION
        SETS    COMMOD,00H
        SCALL   SWMOD3
        PRNT    RESP,MWAIT,LFCR         ;PRINT WAIT MESSAGE
```

```
        DELAY   8                               ;WAIT FOR TOOL TO SWITCH
        SETD    TEMP1,iRDPTR
        INCD    TEMP1,SPOFSET
        CV2ASC  TMODA,iTEMP1,0,1
        PRNT    RESP,MSWPOS,TMODA+1,LFCR        ;OUTPUT SWITCH POSITION
        PRNTBR  RESP,MDONE,PLFCR,MLOOP
;
; * SWITCH POSITION 4 *
;
TMOD4   ASCMPE  MSMOD4,INBUF+5,TMOD6    ;LOG/CALIBRATION
        SETS    COMMOD,00H
        SCALL   SWMOD4
        PRNT    RESP,MWAIT,LFCR         ;PRINT WAIT MESSAGE
        DELAY   8                       ;WAIT FOR TOOL TO SWITCH
        SETD    TEMP1,iRDPTR
        INCD    TEMP1,SPOFSET
        CV2ASC  TMODA,iTEMP1,0,1
        PRNT    RESP,MSWPOS,TMODA+1,LFCR        ;OUTPUT SWITCH POSITION
        PRNTBR  RESP,MDONE,PLFCR,MLOOP
;
; * SWITCH POSITION 6 *
;
TMOD6   ASCMPE  MSMOD6,INBUF+5,DDUMP            ;LATCHED
        SETS    COMMOD,00H
        SCALL   SWMOD6
        PRNT    RESP,MWAIT,LFCR         ;PRINT WAIT MESSAGE
        DELAY   8                       ;WAIT FOR TOOL TO SWITCH
        SETD    TEMP1,iRDPTR
        INCD    TEMP1,SPOFSET
        CV2ASC  TMODA,iTEMP1,0,1
        PRNT    RESP,MSWPOS,TMODA+1,LFCR
        PRNTBR  RESP,MDONE,PLFCR,MLOOP
                                        ;OUTPUT SWITCH POSITION
;
; * DUMP DATA FROM ONBOARD SRAM *
;
DDUMP   ASCMPE  MDUMP,INBUF+5,SDELAY            ;XMIT COMPRESSED DATA
        BNES    DHOLFLG,01H,NODUMP
        SETS    COMMOD,00H
        SETS    NOWRFLG,00H
        PRNT    RESP,AKDUMP,LFCR
        IPOST   QINPUT,INBUF,ISIZE,1    ;SET INPUT POST
        SCALL   DUMP
        SETS    DUMPFLG,01H
        GOTO    MLOOP2
;
NODUMP  PRNTBR  RESP,MNODUMP,PLFCR,MLOOP
;
; * SET DELAY TIME *
;
SDELAY  ASCMPE  MDELAY,INBUF+5,XMDATA           ;CHECK THE COMMAND MESSAGE
;
        SETS    COMMOD,00H
        SETS    TEMP5,00H
        AS2BCE  TEMP5,INBUF+11,ASCONVE,1,3      ;CONVERT DELAY TIME FROM
        BCD2IS  DTIME,TEMP5,BCDCONE             ;ASCII TO BCD TO HEX
        I2BCD1  TEMP5,DTIME+1                   ;RECONV VALUE AND XMIT BACK TO
        CV2ASC  TEMP5,TEMP5+1,0,2               ;USER TO VERIFY
        PRNT    RESP,MDTIME,MSEQU,TEMP5,LFCR
        BLED    DTIME,MAXTIM,CONVTIM            ;VALUE <= 210 MINUTES
```

```
        PRNTBR  RESP,MSIZERR,PLFCR,MLOOP          ;IF > MAX VALUE ERROR
;
CONVTIM SETS    DTIMFLG,01H                       ;SET THE DELAY TIME FLAG
                                                  ;USED TO NOTE D'TIME ENTERED
        PRNTBR  RESP,MDONE,PLFCR,MLOOP
;
; * XMIT DATA *
;
XMDATA  ASCMPE  MXMDATA,INBUF+5,REXDATA
        IPOST   QINPUT,INBUF,ISIZE,1     ;SET INPUT POST
        SCALL   XMIT
        GOTO    MLOOP2
;
; * REXMIT DATA *
;
REXDATA ASCMPE  MREXDATA,INBUF+5,DHOLE
        IPOST   QINPUT,INBUF,ISIZE,1     ;SET INPUT POST
        SCALL   REXMIT
        GOTO    MLOOP2
;
; * DOWNHOLE *
;
DHOLE   ASCMPE  MSMAIN,INBUF+5,STAT               ;DOWNHOLE COMMAND
        BEQS    DUMPFLG,00H,DFRST
        SETS    COMMOD,00H
        IPOST   QINPUT,INBUF,ISIZE,1
        XFER    DAT,INBUF+10,10                    ;WRITE DATE TO MEMORY
        SETS    DAT+10,00H
        XFER    CLKTIM,INBUF+21,8                 ;WRITE PC CLOCK TIME TO MEM
        SETS    CLKTIM+8,00H
        BEQS    DTIMFLG,01H,NEEDMOD               ;VERIFY DELAY TIME SET
        PRNTBR  RESP,MENTRDLY,PLFCR,MLOOP          ;IF NOT PRINT MESSAGE

NEEDMOD SETD    TEMP1,iRDPTR                      ;SET TEMP1 = ADDR OF SW
        INCD    TEMP1,SPOFSET                     ;POS IN CHANNEL IP BUFFER
        BEQS    iTEMP1,4,STRTDLY                  ;VERIFY TOOL MODE SET
        BEQS    iTEMP1,6,STRTDLY
        PRNTBR  RESP,MWRTMOD,PLFCR,MLOOP
;
STRTDLY PRNT    RESP,MOK,PLFCR                     ;PRINT OK MESSEAGE
        SETD    DCTR,0000H                         ;INIT CTRS
        SETD    MINCTRH,0000H
        SETS    T0FLG,00H
        BEQD    DTIME,0000H,COMPRC
        INTERV  ENABLE,F4H,4800H                  ;START DELAY TIMER

TWAIT   BEQS    T0FLG,0,TWAIT                     ;CHECK T0 INT FLAG
        SETS    T0FLG,0                             ;CLEAR T0 INT FLAG

        SNEG    INPUT,CONCNT                    ;IF A COMMAND SENT DURING DELAY LOOP
        BEQS    INPUT,40H,TOOLCMD               ;AND I/P GOOD GO TO COMMAND TABLE
        IPOST   QINPUT,INBUF,ISIZE,1

CONCNT  BNED    MINCTRH,TFACTOR,TWAIT           ;CHECK MINUTE COUNTER(60/MINUTE)
        SETD    MINCTRH,0000H
        INCD    DCTR,1
        COMPE   DTIME,DCTR,2,TWAIT                ;CHECK FOR TIME OUT

        INTERV  DISABLE
```

```
COMPRC   SWEEP    SRAMBLK,0,8
         SETS     SRAMBLK+3,60H
         SETS     NOWRFLG,00H
         SETS     DHOLFLG,01H
         SETS     DUMPFLG,00H
         SETS     COMMOD,COMPFLG                    ;SET COMM MODE TO COMPRESS AND S
         SETS     SRADDR,00H                        ;INITIALIZE SRAM ADDRESS
         SETDBR   SRADDR+1,0000H,MLOOP              ;REGISTERS
;
DFRST    PRNTBR   RESP,MDFRST,PLFCR,MLOOP
;
;  * STATUS *
;
STAT     ASCMPE   MSTATUS,INBUF+5,XXMAIN              ;OUTPUT TOOL STATUS
         SETD     TEMP1,iRDPTR
         INCD     TEMP1,SPOFSET
         CV2ASC   TMODA,iTEMP1,0,1
         I2BCD1   TEMP5,DTIME+1                     ;RECONV VALUE AND XMIT BACK TO
         CV2ASC   TEMP5,TEMP5+1,0,2                 ;USER TO VERIFY
         PRNT     RESP,MSWPOS,MSEQU,TMODA+1,LFCR
         PRNT     RESP,MDTIME,MSEQU,TEMP5,LFCR
         PRNT     RESP,RVDATM,LFCR
         PRNT     RESP,MSSERN,LFCR
         PRNTBR   RESP,MDONE,PLFCR,MLOOP
;
;
;
XXMAIN   ASCMPE   MXMAIN,INBUF+5,STOP                ;XMAIN TO CHECK COMPRS ROUTINE
         IPOST    QINPUT,INBUF,ISIZE,1              ;SET INPUT POST
         SETS     COMMOD,XMFLG
         PRNTBR   RESP,MXMOD,LFCR,MLOOP2

;  * STOP *
;


STOP     ASCMPE   MSTOP,INBUF+5,MODERR              ;STOP PROCESSING DATA
         PRNT     RESP,AKSTOP,PLFCR
         SETSBP   COMMOD,00H,MLOOP                  ;CLEAR COMMUNICATE MODE
;
;  * ERROR MESSAGE SECTION *
;
ASCONVE  PRNTBR   RESP,MAS2HXERR,PLFCR,MLOOP         ;ASCII TO HEX CONVERSION
BCDCONE  PRNTBR   RESP,MBCD2BINE,PLFCR,MLOOP         ;BCD TO BINARY CONVERSION
MODERR   PRNTBR   RESP,MCODERR,PLFCR,MLOOP
;
;****;****;****;****;****;*      MEMORY TESTS      ****;****;****;****;****;
;
;
MEMTST   ASCMPE   MTEST,INBUF+3,SYSCHEK
;
FIFTST   ASCMPE   MFIFTST,INBUF+5,ALLTST             ;FIFO TEST
         SETS     COMMOD,00H
         PRNT     RESP,AKFIFTST,LFCR                ;ACK FIFO TEST REQUEST
         FTIMER   DISABLE                           ;DISABLE INST SECTION
         SCALL    FIFOCHK                           ;CALL FIFO TEST ROUTINE
         SETS     ERRCTR,00H
         FTIMER   ENABLE,FFH,F883H
         SCALL    SWMOD0
         PRNTBR   RESP,MDONE,PLFCR,MLOOP            ;PRINT DONE MESSAGE
```

```
;
;**** **** **** **** **** **** **** **** **** **** **** **** **** **** **** ****
;
; * START COMPLETE MEMORY TEST *
; * THIS ROUTINE PERFORMS ADDRESS/DATA LINE, MEMORY CELL, AND CS LINE
; * TEST ON ALL THE MEMORY DEVICES
;
ALLTST  ASCMPE  MTSTALL,INBUF+5,PGMLOAD
        PRNT    RESP,AKTSTALL,LFCR          ;ACK REQUEST
;
; * MEMORY CELL TEST
;
        SETS    COMMOD,00H
        FTIMER  DISABLE                     ;DISABLE INST SECTION
        SETD    CSVALU,0000H                ;CLEAR CS VALUE
        PRNT    RESP,ACKCEL,LFCR

CELOOP  SCALL   CELCHEK                     ;PERFORM MEMORY CELL CHECK
        BEQS    ERRFLG,1,ATDUN
        SNEG    INPUT,CELCON                ;CHECK TO SEE IF A COMMAND WAS SENT
        BEQS    INPUT,40H,TOOLCMD           ;DURING TESTING
                                            ;IF SO EXIT TEST LOOP AND QUERY
        IPOST   QINPUT,INBUF,ISIZE,1        ;COMMAND
CELCON  INCD    CSVALU,1
        BLTD    CSVALU,000CH,CELOOP
;
; * ADDRESS/DATA LINE TEST
;
        SETD    CSVALU,0000H                ;RESET CS VALUE TO 00
        PRNT    RESP,ACKADR,LFCR
ADRLOOP SCALL   ADRCHEK                     ;ADDRESS/DATA LINE CHECK
        BEQS    ERRFLG,1,ATDUN
        SNEG    INPUT,ADRCON                ;CHECK TO SEE IF A COMMAND WAS SENT
        BEQS    INPUT,40H,TOOLCMD           ;DURING TESTING
                                            ;IF SO EXIT TEST LOOP AND QUERY
        IPOST   QINPUT,INBUF,ISIZE,1        ;COMMAND
ADRCON  INCD    CSVALU,1
        BLTD    CSVALU,000CH,ADRLOOP
;
; * CHIP SELECT LINE TEST
;
        PRNT    RESP,ACKCS,LFCR
CSTST   SCALL   CSCHEK                      ;CS LINE CHECK
        PRNT    RESP,MDONE,PLFCR            ;DONE MESSAGE
;
        FTIMER  ENABLE,FFH,F883H            ;REENABLE INST SECTION
        SCALL   SWMOD0                      ;SET TO MODE 0
;
ATDUN   SETSBR  ERRCTR,00H,MLOOP
;
;
; * END COMPLETE MEMORY TEST *
;
;****;****;****;****;****;****;****;****;****;****;****;****;****;****;****;*
;
; * PROGRAM LOAD *
; * THIS PROGRAM LOADS THE TOOL PROGRAM INTO ROM
;
PGMLOAD ASCMPE  MSOCRD,INBUF+5,TPGMRAM
        SETS    COMMOD,00H
```

```
        FTIMER  DISABLE                                 DISABLE INST SECTION
        SCALL   PLOAD
        FTIMER  ENABLE,FFH,F883H                        ;REENABLE INST SECTION
        SCALL   SWMOD0                                  ;SET TO MODE 0
        GOTO    MLOOP
;
; * PROGRAM RAM TEST *
; * THIS CALLS THE ROUTINE TO TEST THE PROGRAM SRAM
;
TPGMRAM ASCMPE  MPTMEM,INBUF+5,CHEKSM
        SETS    COMMOD,00H
        FTIMER  DISABLE                                 DISABLE INST SECTION
        SCALL   PTMEM
        FTIMER  ENABLE,FFH,F883H                        ;REENABLE INST SECTION
        SCALL   SWMOD0                                  ;SET TO MODE 0
        GOTO    MLOOP
;
; * CHECK SUM TEST *
; * THIS ROUTINE CALLS THE CHECK SUM ROUTINE
;
CHEKSM  ASCMPE  MCHKSUM,INBUF+5,DEVTST
        SETS    COMMOD,00H
        FTIMER  DISABLE                                 DISABLE INST SECTION
        SCALL   PCHKS
        FTIMER  ENABLE,FFH,F883H                        ;REENABLE INST SECTION
        SCALL   SWMOD0                                  ;SET TO MODE 0
        GOTO    MLOOP
;
; * DEVICE TEST *
; * THIS ROUTINE PERFORMS THE ADDRESS/DATA LINE, CELL, AND CS LINE TESTS
; * ON A MEMORY DEVICE SELECTED BY THE USER
;
;
DEVTST  ASCMPE  MDEVTST,INBUF+5,ADRCHK                  ;SINGLE MEMORY DEVICE TEST

        SETS    COMMOD,00H
        FTIMER  DISABLE                                 ;DISABLE INST SECTION
CONVRT  SETD    TEMP5,0000H
        AS2HXE  TEMP5,INBUF+15,ASCONVE,1,1
        SETS    CSVALU+1,iTEMP5,BCDCONE
        BGTD    CSVALU,11,RNGER                         ;CS VALUE <= 11
; *
        PRNT    RESP,ACKCEL,LFCR
        SCALL   CELCHEK                                 ;CHECK CELLS ON DEVICE
        BEQS    ERRFLG,1,DTSTDUN
        PRNT    RESP,ACKADR,LFCR
        SCALL   ADRCHEK                                 ;CHEK ADDR DATA LINES
        BEQS    ERRFLG,1,DTSTDUN
; *
        PRNT    RESP,ACKCS,LFCR
        SCALL   CSCHEK                                  ;CHECK CS LINES

RNGER   PRNT    RESP,MRNGE,PLFCR                                ;CS VALUE OUT OF RANGE
DTSTDUN SETS    ERRCTR,00H
        FTIMER  ENABLE,FFH,F883H                        ;ENABLE INSTRUMENT SECTION
        SCALL   SWMOD0                                  ;SET TO SWMOD 0
        PRNTBR  RESP,MDONE,PLFCR,MLOOP                  ;PRINT DONE MESSAGE
;
;
;
```

```
;  * ADDRESS/DATA LINE TEST *
;
;  * THIS COMMAND ROUTINE CALLS THE ADDRESS/DATA LINE TEST FOR A SINGLE
;  * DEVICE
;
;
ADRCHK   ASCMPE   MADRCHEK,INBUF+5,CELCHK          ;ADDR/DATA LINE TEST

         SETS     COMMOD,00H
         FTIMER   DISABLE                          ;DISABLE INST SECTION
         SETD     TEMP5,0000H                      ;CONVERT CS VALUE SELECTED
         AS2BCE   TEMP5+2,INBUF+18,ASCONVE,1,1
         BCD2IS   CSVALU,TEMP5,BCDCONE
         BGTS     CSVALU,11,RNGER                  ;CS VALUE <= 11
;
         PRNT     RESP,ACKADR,LFCR
         SETS     CSVALU,00H
         SCALL    ADRCHEK
         SETS     ERRCTR,00H
         FTIMER   ENABLE,FFH,F883H                 ;REENABLE INST SECTION
         SCALL    SWMOD0                           ;SET TO MODE 0
         PRNTBR   RESP,MDONE,PLFCR,MLOOP
;
;
;  * MEMORY CELL TEST *
;
;  * THIS ROUTINE CALLS CELL CHECK ROUTINE TO TEST ALL THE MEMORY LOCATIONS
;  * IN A PARTICULAR DEVICE
;
;
CELCHK   ASCMPE   MCELCHEK,INBUF+5,CSCHK           ;MEMORY CELL TEST

         SETS     COMMOD,00H
         FTIMER   DISABLE                          ;DISABLE INST SECTION
         SETD     TEMP5,0000H                      ;CONVERT CS VALUE SELECTED
         AS2BCE   TEMP5+2,INBUF+15,ASCONVE,1,1
         BCD2IS   CSVALU,TEMP5,BCDCONE
         BGTS     CSVALU,11,RNGER                  ;CS VALUE <= 11

         PRNT     RESP,ACKCEL,LFCR
         SCALL    CELCHEK
         SETS     ERRCTR,00H
         FTIMER   ENABLE,FFH,F883H                 ;REENABLE INST SECTION
         SCALL    SWMOD0                           ;SET TO MODE 0
         PRNTBR   RESP,MDONE,PLFCR,MLOOP
;
;
;  * CHIP SELECT TEST *
;
;  * THIS ROUTINE CALLS THE CS TEST ROUTINE TO TEST ALL THE CS LINES
;
;
CSCHK    ASCMPE   MCSCHEK,INBUF+5,ONELOC           ;CS LINE CHECK
         PRNT     RESP,ACKCS,LFCR
         SETS     COMMOD,00H
         FTIMER   DISABLE                          ;DISABLE INST SECTION
         SCALL    CSCHEK
         SETS     ERRCTR,00H
         FTIMER   ENABLE,FFH,F883H                 ;REENABLE INST SECTION
         SCALL    SWMOD0                           ;SET TO MODE 0
```

```
        PRNTBR  RESP,MDONE,PLFCR,MLOOP
;
;
; * MEMORY LOCATION TEST *
;
; * THIS ROUTINE CALLS THE ONE LOCATION TEST WHICH TESTS AN INDIVIDUAL
; * ADDRESS LOCATION
;
;
ONELOC  ASCMPE  MONELOC,INBUF+5,MSGER           ;SINGLE MEMORY LOCATION TEST
        AS2HXE  SRADDR,INBUF+15,ASCONVE,1,3
        BGES    SRADDR,60H,MSGBIG
        PRNT    RESP,MLOC,LFCR
        SETS    COMMOD,00H
        FTIMER  DISABLE                         ;DISABLE INST SECTION
TSTADR  SCALL   ONELTST
        SETS    ERRCTR,00H
        PRNT    RESP,MTSTDUN,LFCR
MSGDUN  FTIMER  ENABLE,FFH,F883H                ;REENABLE INST SECTION
        SCALL   SWMOD0                          ;SET TO MODE 0
        PRNTBR  RESP,MDONE,PLFCR,MLOOP
;
MSGBIG  PRNTBR  RESP,MRNGE,PLFCR,MLOOP
MSGER   PRNTBR  RESP,MCODERR,PLFCR,MLOOP        ;COMMAND CODE ERROR MESSAGE
;
;****;****;****;****;****;      END MEM TEST     ****;****;****;****;****;*
;
;
;****;****;****;****;***        DIAG TESTS        ****;****;****;****;****;**
;
;
        SUBTITLE *** PROCESS CHECK (MEMORY FUNCTIONS) ***
        PAGE

SYSCHEK ASCMPE  MM,INBUF+3,MSGER
;
        ASCMPE  MSBK,INBUF+5,CMEX
        ASCMPE  GMSG,INBUF+7,CBRK1
        PRNTBR  RESP,MSBKV,MSBKNA,PLF2CR,MLOOP
CBRK1   SETD    BPTR1,INBUF+7
        SETD    SPTR1,MLOOP
        SETDBR  SPTR2,RESP,PBRK
;
; CHECK FOR MEMORY EXAMINE
;
CMEX    ASCMPE  MSMEX,INBUF+5,MSGER
        SETD    MPTR2,INBUF+7
        SETD    SPTR1,MLOOP
        SETDBR  SPTR2,RESP,PMEX
;
;****;****;****;****;****;*      END DIAG          ****;****;****;****;****;***
;
```

```
>>
;
; BEGIN MACRO DEFINITIONS
;
ASCMP   .MACRO  ARGA,ARGB,ARGC
        .IFNDEF ZACP
        EXTERN  ZACP
        .ENDIF
        WORD    ZACP    ; IF STRING(ARGA) .EQ. STRING(ARGB), THEN GOTO ARGC ELSE
        INDR3   ARGA,ARGB,ARGC
        .ENDM
;
ASCMPE  .MACRO  ARGA,ARGB,ARGC
        .IFNDEF ZACPE
        EXTERN  ZACPE
        .ENDIF
        WORD    ZACPE   ; IF STRING(ARGA) .NE. STRING(ARGB), THEN GOTO ARGC ELSE
        INDR3   ARGA,ARGB,ARGC
        .ENDM
;
AS2BC   .MACRO  ARGA,ARGB,ARGC,ARGD,ARGE
        .IFNDEF ZA2B
        EXTERN  ZA2B
        .ENDIF
        WORD    ZA2B    ; CONVERT ASCII @ARGB TO BCD @ARGA, GOTO ARGC IF SUCCESS
        .IFNMA  5
        INDR4   ARGA,ARGB,ARGD,ARGC
        .ELSE
        INDR2   ARGA,ARGB
        INDRBU  ARGD
        INDRBL  ARGE
        INDR1   ARGC
        .ENDIF
        .ENDM
;
AS2BCE  .MACRO  ARGA,ARGB,ARGC,ARGD,ARGE
        .IFNDEF ZA2BE
        EXTERN  ZA2BE
        .ENDIF
        WORD    ZA2BE   ; CONVERT ASCII @ARGB TO BCD @ARGA, GOTO ARGC IF UNSUCC
        .IFNMA  5
        INDR4   ARGA,ARGB,ARGD,ARGC
        .ELSE
```

```
        INDR2   ARGA,ARGB
        INDRBU  ARGD
        INDRBL  ARGE
        INDR1   ARGC
        .ENDIF
        .ENDM
;
AS2HX   .MACRO  ARGA,ARGB,ARGC,ARGD,ARGE
        .IFNDEF ZA2H
        EXTERN  ZA2H
        .ENDIF
        WORD    ZA2H    ; CONVERT ASCII @ARGB TO HEXADECIMAL @ARGA, GOTO ARGC IF
        .IFNMA  5
        INDR4   ARGA,ARGB,ARGD,ARGC
        .ELSE
        INDR2   ARGA,ARGB
        INDRBU  ARGD
        INDRBL  ARGE
        INDR1   ARGC
        .ENDIF
        .ENDM
;
AS2HXE  .MACRO  ARGA,ARGB,ARGC,ARGD,ARGE
        .IFNDEF ZA2HE
        EXTERN  ZA2HE
        .ENDIF
        WORD    ZA2HE   ; CONVERT ASCII @ARGB TO HEXADECIMAL @ARGA, GOTO ARGC IF
        .IFNMA  5
        INDR4   ARGA,ARGB,ARGD,ARGC
        .ELSE
        INDR2   ARGA,ARGB
        INDRBU  ARGD
        INDRBL  ARGE
        INDR1   ARGC
        .ENDIF
        .ENDM
;
BCD2IS  .MACRO  ARGA,ARGB,ARGC
        .IFNDEF ZBCDIS
        EXTERN  ZBCDIS
        .ENDIF
        WORD    ZBCDIS  ; BINARY INTEGER (ARGA(1:2)) = BCD INTEGER (ARGB(1:3)),
        INDR3   ARGA,ARGB,ARGC
        .ENDM
;
BCD2ID  .MACRO  ARGA,ARGB,ARGC
        .IFNDEF ZBCDID
        EXTERN  ZBCDID
        .ENDIF
        WORD    ZBCDID  ; BINARY INTEGER (ARGA(1:4)) = BCD INTEGER (ARGB(1:5)),
        INDR3   ARGA,ARGB,ARGC
        .ENDM
;
F2BCDS  .MACRO  ARGA,ARGB
        .IFNDEF ZFBCDS
        EXTERN  ZFBCDS
        .ENDIF
        WORD    ZFBCDS  ; BCD FRACTION (ARGA(1:8)) = BINARY FRACTION (ARGB(1:2))
        INDR2   ARGA,ARGB
        .ENDM
```

```
;
F2BCDD  .MACRO  ARGA,ARGB
        .IFNDEF ZFBCDD
        EXTERN  ZFBCDD
        .ENDIF
        WORD    ZFBCDD  ; BCD FRACTION (ARGA(1:16)) = BINARY FRACTION (ARGB(1:4)
        INDR2   ARGA,ARGB
        .ENDM
;
I2BCD1  .MACRO  ARGA,ARGB
        .IFNDEF ZIBCD1
        EXTERN  ZIBCD1
        .ENDIF
        WORD    ZIBCD1  ; BCD INTEGER (ARGA(1:3)) = BINARY INTEGER (ARGB(1:1))
        INDR2   ARGA,ARGB
        .ENDM
;
I2BCD2  .MACRO  ARGA,ARGB
        .IFNDEF ZIBCD2
        EXTERN  ZIBCD2
        .ENDIF
        WORD    ZIBCD2  ; BCD INTEGER (ARGA(1:3)) = BINARY INTEGER (ARGB(1:2))
        INDR2   ARGA,ARGB
        .ENDM
;
I2BCD3  .MACRO  ARGA,ARGB
        .IFNDEF ZIBCD3
        EXTERN  ZIBCD3
        .ENDIF
        WORD    ZIBCD3  ; BCD INTEGER (ARGA(1:5)) = BINARY INTEGER (ARGB(1:3))
        INDR2   ARGA,ARGB
        .ENDM
;
I2BCD4  .MACRO  ARGA,ARGB
        .IFNDEF ZIBCD4
        EXTERN  ZIBCD4
        .ENDIF
        WORD    ZIBCD4  ; BCD INTEGER (ARGA(1:5)) = BINARY INTEGER (ARGB(1:4))
        INDR2   ARGA,ARGB
        .ENDM
;
IANDS   .MACRO  ARGA,ARGB,ARGC
        .IFNDEF ZIANDS
        EXTERN  ZIANDS
        .ENDIF
        WORD    ZIANDS  ; ARGA(1:1) = ARGB(1:1) .AND. ARGC
        INDR2   ARGA,ARGB
        INDRS   ARGC
        .ENDM
;
IANDD   .MACRO  ARGA,ARGB,ARGC
        .IFNDEF ZIANDD
        EXTERN  ZIANDD
        .ENDIF
        WORD    ZIANDD  ; ARGA(1:2) = ARGB(1:2) .AND. ARGC
        INDR3   ARGA,ARGB,ARGC
        .ENDM
;
IORS    .MACRO  ARGA,ARGB,ARGC
        .IFNDEF ZIORS
```

CA2183123

```
        EXTERN  ZIORS
        .ENDIF
        WORD    ZIORS    ; ARGA(1:1) = ARGB(1:1) .OR. ARGC
        INDR2   ARGA,ARGB
        INDRS   ARGC
        .ENDM
;
IORD    .MACRO  ARGA,ARGB,ARGC
        .IFNDEF ZIORD
        EXTERN  ZIORD
        .ENDIF
        WORD    ZIORD    ; ARGA(1:2) = ARGB(1:2) .OR. ARGC
        INDR3   ARGA,ARGB,ARGC
        .ENDM
;
IXORS   .MACRO  ARGA,ARGB,ARGC
        .IFNDEF ZIXORS
        EXTERN  ZIXORS
        .ENDIF
        WORD    ZIXORS   ; ARGA(1:1) = ARGB(1:1) .XOR. ARGC
        INDR2   ARGA,ARGB
        INDRS   ARGC
        .ENDM
;
IXORD   .MACRO  ARGA,ARGB,ARGC
        .IFNDEF ZIXORD
        EXTERN  ZIXORD
        .ENDIF
        WORD    ZIXORD   ; ARGA(1:2) = ARGB(1:2) .XOR. ARGC
        INDR3   ARGA,ARGB,ARGC
        .ENDM
;
BRET    .MACRO
        .IFNDEF ZBRET
        EXTERN  ZBRET
        .ENDIF
        WORD    ZBRET    ; BREAK POINT RETURN
        INDR1   BREAK+5
        .ENDM
;
SCALL   .MACRO  ARGA,ARGB
        .IFNDEF ZJSR
        EXTERN  ZJSR
        .ENDIF
        WORD    ZJSR     ; CALL ARGA
        .IFMA   2
        INDR3   ARGA,ARGB,0
        .ELSE
        INDR3   ARGA,NORET,0
        .ENDIF
        .ENDM
;
RETURN  .MACRO  ARGA,ARGB
        .IFNDEF ZRTN
        EXTERN  ZRTN
        .ENDIF
        WORD    ZRTN     ; SUBROUTINE RETURN
        .IFMA   1
        .IFMA   2
        INDR2   ARGA,ARGB
```

```
        .ELSE
        WORD    ERROR
        .ENDIF
        .ELSE
        INDR1   NORET
        .ENDIF
        .ENDM
;
CHKSUM  .MACRO  ARGA,ARGB,ARGC
        .IFNDEF ZCKS
        EXTERN  ZCKS
        .ENDIF
        WORD    ZCKS    ; ARGA(1:4) = CHECKSUM(ARGB(1:ARGC))
        INDR3   ARGA,ARGB,ARGC
        .ENDM
;
DELAY   .MACRO  ARGA
        .IFNDEF ZDELS
        EXTERN  ZDELS
        .ENDIF
        WORD    ZDELS   ; DELAY ARGA SECOND(S)
        INDR2   ARGA,0
        .ENDM
;
MDELAY  .MACRO  ARGA
        .IFNDEF ZDELM
        EXTERN  ZDELM
        .ENDIF
        WORD    ZDELM   ; DELAY ARGA MILLISECONDS
        INDR2   ARGA,0
        .ENDM
;
BEQD    .MACRO  ARGA,ARGB,ARGC
        .IFNDEF ZBEQD
        EXTERN  ZBEQD
        .ENDIF
        WORD    ZBEQD   ; IF ARGA(1:2) .EQ. ARGB, GOTO ARGC
        INDR3   ARGA,ARGB,ARGC
        .ENDM
;
BNED    .MACRO  ARGA,ARGB,ARGC
        .IFNDEF ZBNED
        EXTERN  ZBNED
        .ENDIF
        WORD    ZBNED   ; IF ARGA(1:2) .NE. ARGB, GOTO ARGC
        INDR3   ARGA,ARGB,ARGC
        .ENDM
;
BLED    .MACRO  ARGA,ARGB,ARGC
        .IFNDEF ZBLED
        EXTERN  ZBLED
        .ENDIF
        WORD    ZBLED   ; IF ARGA(1:2) .LE. ARGB, GOTO ARGC
        INDR3   ARGA,ARGB,ARGC
        .ENDM
;
BLTD    .MACRO  ARGA,ARGB,ARGC
        .IFNDEF ZBLTD
        EXTERN  ZBLTD
        .ENDIF
```

```
        WORD    ZBLTD   ; IF ARGA(1:2) .LT. ARGB, GOTO ARGC
        INDR3   ARGA,ARGB,ARGC
        .ENDM
;
BGED    .MACRO  ARGA,ARGB,ARGC
        .IFNDEF ZBGED
        EXTERN  ZBGED
        .ENDIF
        WORD    ZBGED   ; IF ARGA(1:2) .GE. ARGB, GOTO ARGC
        INDR3   ARGA,ARGB,ARGC
        .ENDM
;
BGTD    .MACRO  ARGA,ARGB,ARGC
        .IFNDEF ZBGTD
        EXTERN  ZBGTD
        .ENDIF
        WORD    ZBGTD   ; IF ARGA(1:2) .GT. ARGB, GOTO ARGC
        INDR3   ARGA,ARGB,ARGC
        .ENDM
;
INCD    .MACRO  ARGA,ARGB
        .IFNDEF ZINCD
        EXTERN  ZINCD
        .ENDIF
        WORD    ZINCD   ; INCREMENT ARGA(1:2) BY ARGB
        INDR2   ARGA,ARGB
        .ENDM
;
INCDBR  .MACRO  ARGA,ARGB,ARGC
        .IFNDEF ZINCDB
        EXTERN  ZINCDB
        .ENDIF
        WORD    ZINCDB  ; INCREMENT ARGA(1:2) BY ARGB, GOTO ARGC
        INDR3   ARGA,ARGB,ARGC
        .ENDM
;
INCDBZ  .MACRO  ARGA,ARGB,ARGC
        EXTERN  ZINCDZ
        .ENDIF
        WORD    ZINCDZ  ; INCREMENT ARGA(1:2) BY ARGB, BRANCH ZERO TO ARGC
        INDR3   ARGA,ARGB,ARGC
        .ENDM
;
INCDBN  .MACRO  ARGA,ARGB,ARGC
        .IFNDEF ZINCDN
        EXTERN  ZINCDN
        .ENDIF
        WORD    ZINCDN  ; INCREMENT ARGA(1:2) BY ARGB, BRANCH NON-ZERO TO ARGC
        INDR3   ARGA,ARGB,ARGC
        .ENDM
;
DECD    .MACRO  ARGA,ARGB
        .IFNDEF ZDECD
        EXTERN  ZDECD
        .ENDIF
        WORD    ZDECD   ; DECREMENT ARGA(1:2) BY ARGB
        INDR2   ARGA,ARGB
        .ENDM
;
DECDBR  .MACRO  ARGA,ARGB,ARGC
```

```
        .IFNDEF ZDECDB
        EXTERN  ZDECDB
        .ENDIF
        WORD    ZDECDB   ; DECREMENT ARGA(1:2) BY ARGB, GOTO ARGC
        INDR3   ARGA,ARGB,ARGC
        .ENDM
;
DECDBZ  .MACRO  ARGA,ARGB,ARGC
        .IFNDEF ZDECDZ
        EXTERN  ZDECDZ
        .ENDIF
        WORD    ZDECDZ   ; DECREMENT ARGA(1:2) BY ARGB, BRANCH ZERO TO ARGC
        INDR3   ARGA,ARGB,ARGC
        .ENDM
;
DECDBN  .MACRO  ARGA,ARGB,ARGC
        .IFNDEF ZDECDN
        EXTERN  ZDECDN
        .ENDIF
        WORD    ZDECDN   ; DEC ARGA(1:2) BY ARGB, BRANCH NON-ZERO TO ARGC
        INDR3   ARGA,ARGB,ARGC
        .ENDM
;
SETD    .MACRO  ARGA,ARGB
        .IFNDEF ZSETD
        EXTERN  ZSETD
        .ENDIF
        WORD    ZSETD    ; SET ARGA(1:2) TO ARGB
        INDR2   ARGA,ARGB
        .ENDM
;
SETDBR  .MACRO  ARGA,ARGB,ARGC
        .IFNDEF ZSETDB
        EXTERN  ZSETDB
        .ENDIF
        WORD    ZSETDB   ; SET ARGA(1:2) TO ARGB, GOTO ARGC
        INDR3   ARGA,ARGB,ARGC
        .ENDM
;
SWEEP   .MACRO  ARGA,ARGB,ARGC
        .IFNDEF ZSWP
        EXTERN  ZSWP
        .ENDIF
        WORD    ZSWP     ; SWEEP ARGA(1:ARGC) WITH ARGB
        INDR1   ARGA
        INDRS   ARGB
        INDR1   ARGC
        .ENDM
;
FIX     .MACRO  ARGA,ARGB,ARGC
        .IFNDEF ZFIX
        EXTERN  ZFIX
        .ENDIF
        WORD    ZFIX     ; ARGA(1:9) = FIX{FP(ARGB(1:5))} OVER 2*ARGC
        INDR3   ARGA,ARGB,ARGC
        .ENDM
;
FIXPOS  .MACRO  ARGA,ARGB,ARGC
        .IFNDEF ZFIXP
        EXTERN  ZFIXP
```

```
        .ENDIF
        WORD    ZFIXP    ; ARGA(1:9) = FIX{FP(ARGB(1:5))} OVER 2*ARGC
        INDR3   ARGA,ARGB,ARGC
        .ENDM
;
FLT     .MACRO  ARGA,ARGB
        .IFNDEF ZFLT
        EXTERN  ZFLT
        .ENDIF
        WORD    ZFLT     ; ARGA(1:5) = FP(ARGB(1:4))
        INDR2   ARGA,ARGB
        .ENDM
;
FLTAB   .MACRO  ARGA,ARGB
        .IFNDEF ZFLTAB
        EXTERN  ZFLTAB
        .ENDIF
        WORD    ZFLTAB   ; ARGA(1:5) = ABS{FP(ARGB(1:4))}
        INDR2   ARGA,ARGB
        .ENDM
;
FNEG    .MACRO  ARGA,ARGB
        .IFNDEF ZNEG
        EXTERN  ZNEG
        .ENDIF
        WORD    ZNEG     ; FP(ARGA(1:5)) = -FP(ARGB(1:5))
        INDR2   ARGA,ARGB
        .ENDM
;
FABS    .MACRO  ARGA,ARGB
        .IFNDEF ZABS
        EXTERN  ZABS
        .ENDIF
        WORD    ZABS     ; FP(ARGA(1:5)) = ABS{FP(ARGB(1:5))}
        INDR2   ARGA,ARGB
        .ENDM
;
FSUB    .MACRO  ARGA,ARGB,ARGC
        .IFNDEF ZSUB
        EXTERN  ZSUB
        .ENDIF
        WORD    ZSUB     ; FP(ARGA(1:5)) = FP(ARGC(1:5)) - FP(ARGB(1:5))
        INDR3   ARGA,ARGB,ARGC
        .ENDM
;
FSUBAB  .MACRO  ARGA,ARGB,ARGC
        .IFNDEF ZSUBAB
        EXTERN  ZSUBAB
        .ENDIF
        WORD    ZSUBAB   ; FP(ARGA(1:5)) = ABS{FP(ARGC(1:5)) - FP(ARGB(1:5))}
        INDR3   ARGA,ARGB,ARGC
        .ENDM
;
FSUBSO  .MACRO  ARGA,ARGB,ARGC
        .IFNDEF ZSUBSO
        EXTERN  ZSUBSO
        .ENDIF
        WORD    ZSUBSO   ; ARGA(1:1) = SIGN{FP(ARGC(1:5)) - FP(ARGB(1:5))}
        INDR3   ARGA,ARGB,ARGC
        .ENDM
```

```
;
FADD    .MACRO  ARGA,ARGB,ARGC
        .IFNDEF ZADD
        EXTERN  ZADD
        .ENDIF
        WORD    ZADD    ; FP(ARGA(1:5)) = FP(ARGC(1:5)) + FP(ARGB(1:5))
        INDR3   ARGA,ARGB,ARGC
        .ENDM
;
FADDAB  .MACRO  ARGA,ARGB,ARGC
        .IFNDEF ZADDAB
        EXTERN  ZADDAB
        .ENDIF
        WORD    ZADDAB  ; FP(ARGA(1:5)) = ABS{FP(ARGC(1:5)) + FP(ARGB(1:5))}
        INDR3   ARGA,ARGB,ARGC
        .ENDM
;
FADDSO  .MACRO  ARGA,ARGB,ARGC
        .IFNDEF ZADDSO
        EXTERN  ZADDSO
        .ENDIF
        WORD    ZADDSO  ; ARGA(1:1) = SIGN{FP(ARGC(1:5)) + FP(ARGB(1:5))}
        INDR3   ARGA,ARGB,ARGC
        .ENDM
;
FMULT   .MACRO  ARGA,ARGB,ARGC
        .IFNDEF ZMULT
        EXTERN  ZMULT
        .ENDIF
        WORD    ZMULT   ; FP(ARGA(1:5)) = FP(ARGC(1:5)) * FP(ARGB(1:5))
        INDR3   ARGA,ARGB,ARGC
        .ENDM
;
FDIV    .MACRO  ARGA,ARGB,ARGC
        .IFNDEF ZDIV
        EXTERN  ZDIV
        .ENDIF
        WORD    ZDIV    ; FP(ARGA(1:5)) = FP(ARGC(1:5)) / FP(ARGB(1:5))
        INDR3   ARGA,ARGB,ARGC
        .ENDM
;
SQRT    .MACRO  ARGA,ARGB
        .IFNDEF ZSQRT
        EXTERN  ZSQRT
        .ENDIF
        WORD    ZSQRT   ; FP(ARGA(1:5) = SQRT{FP(ARGB(1:5))}
        INDR2   ARGA,ARGB
        .ENDM
;
FMT12   .MACRO  ARGA,ARGB
        .IFNDEF ZFM12
        EXTERN  ZFM12
        .ENDIF
        WORD    ZFM12   ; ARGA(1:2) = ARGB(1:1)
        INDR2   ARGA,ARGB
        .ENDM
;
FMT12S  .MACRO  ARGA,ARGB
        .IFNDEF ZFM12S
        EXTERN  ZFM12S
```

```
        .ENDIF
        WORD    ZFM12S  ; ARGA(1:2) = SIGN EXTENDED{ARGB(1:1)}
        INDR2   ARGA,ARGB
        .ENDM
;
FMT14   .MACRO  ARGA,ARGB
        .IFNDEF ZFM14
        EXTERN  ZFM14
        .ENDIF
        WORD    ZFM14   ; ARGA(1:4) = ARGB(1:1)
        INDR2   ARGA,ARGB
        .ENDM
;
FMT14S  .MACRO  ARGA,ARGB
        .IFNDEF ZFM14S
        EXTERN  ZFM14S
        .ENDIF
        WORD    ZFM14S  ; ARGA(1:4) = SIGN EXTENDED{ARGB(1:1)}
        INDR2   ARGA,ARGB
        .ENDM
;
FMT24   .MACRO  ARGA,ARGB
        .IFNDEF ZFM24
        EXTERN  ZFM24
        .ENDIF
        WORD    ZFM24   ; ARGA(1:4) = ARGB(1:2)
        INDR2   ARGA,ARGB
        .ENDM
;
FMT24S  .MACRO  ARGA,ARGB
        .IFNDEF ZFM24S
        EXTERN  ZFM24S
        .ENDIF
        WORD    ZFM24S  ; ARGA(1:4) = SIGN EXTENDED{ARGB(1:2)}
        INDR2   ARGA,ARGB
        .ENDM
;
GOTO    .MACRO  ARGA
        .IFNDEF ZGOTO
        EXTERN  ZGOTO
        .ENDIF
        WORD    ZGOTO   ; GOTO ARGA
        INDR1   ARGA
        .ENDM
;
SPOS    .MACRO  ARGA,ARGB
        .IFNDEF ZSPOS
        EXTERN  ZSPOS
        .ENDIF
        WORD    ZSPOS   ; MSB(ARGA) .EQ. 0, THEN GOTO ARGB
        INDR2   ARGA,ARGB
        .ENDM
;
SNEG    .MACRO  ARGA,ARGB
        .IFNDEF ZSNEG
        EXTERN  ZSNEG
        .ENDIF
        WORD    ZSNEG   ; MSB(ARGA) .NE. 0, THEN GOTO ARGB
        INDR2   ARGA,ARGB
        .ENDM
```

```
;
SZER    .MACRO  ARGA,ARGB
        .IFNDEF ZSZER
        EXTERN  ZSZER
        .ENDIF
        WORD    ZSZER   ; (ARGA) .EQ. 40H, THEN GOTO ARGB
        INDR2   ARGA,ARGB
        .ENDM
;
SNZER   .MACRO  ARGA,ARGB
        .IFNDEF ZSNZR
        EXTERN  ZSNZR
        .ENDIF
        WORD    ZSNZR   ; (ARGA) .NE. 40H, THEN GOTO ARGB
        INDR2   ARGA,ARGB
        .ENDM
;
SPOSN   .MACRO  ARGA,ARGB,ARGC
        .IFNDEF ZSPOSN
        EXTERN  ZSPOSN
        .ENDIF
        WORD    ZSPOSN  ; MSB(ARGA) .EQ. 0, THEN GOTO ARGB ELSE GOTO ARGC
        INDR3   ARGA,ARGB,ARGC
        .ENDM
;
SNEGP   .MACRO  ARGA,ARGB,ARGC
        .IFNDEF ZSNEGP
        EXTERN  ZSNEGP
        .ENDIF
        WORD    ZSNEGP  ; MSB(ARGA) .NE. 0, THEN GOTO ARGB ELSE GOTO ARGC
        INDR3   ARGA,ARGB,ARGC
        .ENDM
;
CV2ASC  .MACRO  ARGA,ARGB,ARGC,ARGD
        .IFNDEF ZHEXS
        EXTERN  ZHEXS
        .ENDIF
        WORD    ZHEXS   ; ARGA(1:2*ARGC(LWR)) = ASCII(ARGB(1:ARGC(LWR)))
        .IFNMA 4
        INDR3   ARGA,ARGB,ARGC
        .ELSE
        INDR2   ARGA,ARGB
        INDRBU  ARGC
        INDRBL  ARGD
        .ENDIF
        .ENDM
;
CV2ASCL .MACRO  ARGA,ARGB,ARGC
        .IFNDEF ZHEXL
        EXTERN  ZHEXL
        .ENDIF
        WORD    ZHEXL   ; ARGA(1:2*ARGC) = ASCII(ARGB(1:ARGC))
        INDR3   ARGA,ARGB,ARGC
        .ENDM
;
IDLE    .MACRO
        IFNDEF  ZIDLE
        EXTERN  ZIDLE
        .ENDIF
        WORD    ZIDLE
```

```
        INDR2   BEGIDL,0
        .ENDM
;
SHL16   .MACRO  ARGA,ARGB,ARGC
        .IFNDEF ZSHL2
        EXTERN  ZSHL2
        .ENDIF
        WORD    ZSHL2    ; ARGA(1:2) = LEFT SHIFT(ARGB(1:2))
        INDR3   ARGA,ARGB,ARGC
        .ENDM
;
SHL32   .MACRO  ARGA,ARGB,ARGC
        .IFNDEF ZSHL4
        EXTERN  ZSHL4
        .ENDIF
        WORD    ZSHL4    ; ARGA(1:4) = LEFT SHIFT(ARGB(1:4))
        INDR3   ARGA,ARGB,ARGC
        .ENDM
;
SHR16   .MACRO  ARGA,ARGB,ARGC
        .IFNDEF ZSHR2
        EXTERN  ZSHR2
        .ENDIF
        WORD    ZSHR2    ; ARGA(1:2) = RIGHT SHIFT(ARGB(1:2))
        INDR3   ARGA,ARGB,ARGC
        .ENDM
;
ASHR16  .MACRO  ARGA,ARGB,ARGC
        .IFNDEF ZASHR2
        EXTERN  ZASHR2
        .ENDIF
        WORD    ZASHR2   ; ARGA(1:2) = ARITHMETIC RIGHT SHIFT(ARGB(1:2))
        INDR3   ARGA,ARGB,ARGC
        .ENDM
;
SHR32   .MACRO  ARGA,ARGB,ARGC
        .IFNDEF ZSHR4
        EXTERN  ZSHR4
        .ENDIF
        WORD    ZSHR4    ; ARGA(1:4) = RIGHT SHIFT(ARGB(1:4))
        INDR3   ARGA,ARGB,ARGC
        .ENDM
;
ASHR32  .MACRO  ARGA,ARGB,ARGC
        .IFNDEF ZASHR4
        EXTERN  ZASHR4
        .ENDIF
        WORD    ZASHR4   ; ARGA(1:4) = ARITHMETIC RIGHT SHIFT(ARGB(1:4))
        INDR3   ARGA,ARGB,ARGC
        .ENDM
;
INTRPT  .MACRO  ARGA
        .IFNDEF ZINT
        EXTERN  ZINT
        .ENDIF
        .IFSAME ARGA,ENABLE
        WORD    ZINT     ; ARGA INTERRUPTS
        INDR2   3,0
        .MACEXIT
        .ENDIF
```

```
.IFSAME ARGA,DISABLE
WORD    ZINT    ; ARGA INTERRUPTS
INDR2   2,0
.MACEXIT
.ENDIF
.IFSAME ARGA,RESTORE
WORD    ZINT    ; ARGA INTERRUPTS
INDR2   0,0
.MACEXIT
.ENDIF
.IFSAME ARGA,SAVE
WORD    ZINT    ; ARGA INTERRUPTS
INDR2   4,0
.MACEXIT
.ENDIF
.IFSAME ARGA,SAVE/ENABLE
WORD    ZINT    ; ARGA INTERRUPTS
INDR2   7,0
.MACEXIT
.ENDIF
.IFSAME ARGA,SAVE/DISABLE
WORD    ZINT    ; ARGA INTERRUPTS
INDR2   6,0
.ELSE
WORD    ZINT,ERROR
.ENDIF
.ENDM
;
SERIN   .MACRO  ARGA
        .IFNDEF ZKIN
        EXTERN  ZKIN
        .ENDIF
        .IFSAME ARGA,DISABLE
        WORD    ZKIN    ; KILL ARGA
        INDR2   0,0
        .MACEXIT
        .ENDIF
        WORD    ERROR
        .ENDM
;
XMOD    MACRO   ARGA
        .IFNDEF ZMOD
        EXTERN  ZMOD
        .ENDIF
        .IFSAME ARGA,DISABLE
        WORD    ZMOD    ; DISABLE MODULATOR
        INDR2   0,0
        .MACEXIT
        .ENDIF
        .IFSAME ARGA,ENABLE
        WORD    ZMOD    ; ENABLE MODULATOR, DATA = 0
        INDR2   1,0
        .MACEXIT
        .ENDIF
        .IFSAME ARGA,ENABLE1
        WORD    ZMOD    ; ENABLE MODULATOR, DATA = 1
        INDR2   3,0
        .MACEXIT
        .ENDIF
        WORD    ERROR
```

```
        .ENDM
;
WRITBE  .MACRO  ARGA,ARGB,ARGC,ARGD,ARGE
        .IFNDEF ZWR
        EXTERN  ZWR
        .ENDIF
        WORD    ZWR       ; ARGA(1:ARGC) = ARGB(1:ARGC)
        INDR1   ARGA      ; WRITE ERROR @ARGD AND GOTO ARGE ELSE CONTINUE
        INDR4   ARGB,ARGC,ARGD,ARGE
        .ENDM
;
WRITBN  .MACRO  ARGA,ARGB,ARGC,ARGD,ARGE
        .IFNDEF ZWRE
        EXTERN  ZWRE
        .ENDIF
        WORD    ZWRE      ; ARGA(1:ARGC) = ARGB(1:ARGC)
        INDR1   ARGA      ; WRITE ERROR @ARGD AND CONTINUE ELSE GOTO ARGE
        INDR4   ARGB,ARGC,ARGD,ARGE
        .ENDM
;
PRNT    .MACRO  ARGA,ARGB,ARGC,ARGD,ARGE,ARGF,ARGG,ARGH,ARGI,ARGJ,ARGK
        .IFNDEF ZPRT
        EXTERN  ZPRT
        .ENDIF
        .IFNMA  1
        WORD    ZPRT,ERROR
        .MACEXIT
        .ENDIF
        .IFNMA  2
        WORD    ZPRT      ; PRINT
        INDR2   ARGA,NULL
        .MACEXIT
        .ENDIF
        .IFNMA  3
        WORD    ZPRT      ; PRINT
        INDR3   ARGA,ARGB,NULL
        .MACEXIT
        .ENDIF
        .IFNMA  4
        WORD    ZPRT      ; PRINT
        INDR4   ARGA,ARGB,ARGC,NULL
        .MACEXIT
        .ENDIF
        .IFNMA  5
        WORD    ZPRT      ; PRINT
        INDR5   ARGA,ARGB,ARGC,ARGD,NULL
        .MACEXIT
        .ENDIF
        .IFNMA  6
        WORD    ZPRT      ; PRINT
        INDR3   ARGA,ARGB,ARGC
        INDR3   ARGD,ARGE,NULL
        .MACEXIT
        .ENDIF
        .IFNMA  7
        WORD    ZPRT      ; PRINT
        INDR3   ARGA,ARGB,ARGC
        INDR4   ARGD,ARGE,ARGF,NULL
        .MACEXIT
        .ENDIF
```

```
.IFNMA  8
WORD    ZPRT    ; PRINT
INDR4   ARGA,ARGB,ARGC,ARGD
INDR4   ARGE,ARGF,ARGG,NULL
.MACEXIT
.ENDIF
.IFNMA  9
WORD    ZPRT    ; PRINT
INDR4   ARGA,ARGB,ARGC,ARGD
INDR5   ARGE,ARGF,ARGG,ARGH,NULL
.MACEXIT
.ENDIF
.IFNMA  10
WORD    ZPRT    ; PRINT
INDR5   ARGA,ARGB,ARGC,ARGD,ARGE
INDR5   ARGF,ARGG,ARGH,ARGI,NULL
.MACEXIT
.ENDIF
.IFNMA  11
WORD    ZPRT    ; PRINT
INDR3   ARGA,ARGB,ARGC
INDR4   ARGD,ARGE,ARGF,ARGG
INDR4   ARGH,ARGI,ARGJ,NULL
.MACEXIT
.ENDIF
.IFMA   11
WORD    ZPRT,ERROR
.ENDIF
.ENDM
;
PRNTBR  .MACRO  ARGA,ARGB,ARGC,ARGD,ARGE,ARGF,ARGG,ARGH,ARGI,ARGJ,ARGK,ARGL
.IFNDEF ZPRTBR
EXTERN  ZPRTBR
.ENDIF
.IFNMA  1
WORD    ZPRTBR,ERROR
.MACEXIT
.ENDIF
.IFNMA  2
WORD    ZPRTBR,ERROR
.MACEXIT
.ENDIF
.IFNMA  3
WORD    ZPRTBR  ; PRINT, THEN BRANCH
INDR3   ARGA,NULL,ARGB
.MACEXIT
.ENDIF
.IFNMA  4
WORD    ZPRTBR  ; PRINT, THEN BRANCH
INDR4   ARGA,ARGB,NULL,ARGC
.MACEXIT
.ENDIF
.IFNMA  5
WORD    ZPRTBR  ; PRINT, THEN BRANCH
INDR5   ARGA,ARGB,ARGC,NULL,ARGD
.MACEXIT
.ENDIF
.IFNMA  6
WORD    ZPRTBR  ; PRINT, THEN BRANCH
INDR3   ARGA,ARGB,ARGC
```

```
INDR3    ARGD,NULL,ARGE
.MACEXIT
.ENDIF
.IFNMA   7
WORD     ZPRTBR   ; PRINT, THEN BRANCH
INDR3    ARGA,ARGB,ARGC
INDR4    ARGD,ARGE,NULL,ARGF
.MACEXIT
.ENDIF
.IFNMA   8
WORD     ZPRTBR   ; PRINT, THEN BRANCH
INDR4    ARGA,ARGB,ARGC,ARGD
INDR4    ARGE,ARGF,NULL,ARGG
.MACEXIT
.ENDIF
.IFNMA   9
WORD     ZPRTBR   ; PRINT, THEN BRANCH
INDR4    ARGA,ARGB,ARGC,ARGD
INDR5    ARGE,ARGF,ARGG,NULL,ARGH
.MACEXIT
.ENDIF
.IFNMA   10
WORD     ZPRTBR   ; PRINT, THEN BRANCH
INDR5    ARGA,ARGB,ARGC,ARGD,ARGE
INDR5    ARGF,ARGG,ARGH,NULL,ARGI
.MACEXIT
.ENDIF
.IFNMA   11
WORD     ZPRTBR   ; PRINT, THEN BRANCH
INDR3    ARGA,ARGB,ARGC
INDR4    ARGD,ARGE,ARGF,ARGG
INDR4    ARGH,ARGI,NULL,ARGJ
.MACEXIT
.ENDIF
.IFNMA   12
WORD     ZPRTBR   ; PRINT, THEN BRANCH
INDR4    ARGA,ARGB,ARGC,ARGD
INDR4    ARGE,ARGF,ARGG,ARGH
INDR4    ARGI,ARGJ,NULL,ARGK
.MACEXIT
.ENDIF
.IFMA    12
WORD     ZPRTBR,ERROR
.ENDIF
.ENDM
;
BPOSTO   .MACRO  ARGA,ARGB,ARGC,ARGD,ARGE,ARGF,ARGG,ARGH,ARGI,ARGJ
         .IFNDEF ZBP
EXTERN   ZBP
.ENDIF
.IFNMA   1
WORD     ZBP
INDR1    ERROR    ; TOO FEW VARIABLES
.MACEXIT
.ENDIF
.IFNMA   2
WORD     ZBP
INDR1    ERROR    ; TOO FEW VARIABLES
.MACEXIT
.ENDIF
```

```
.IFNMA   3
WORD     ZBP       ; BUILD BUFFER AT AND POST AN OUTPUT FROM ARGA
INDR1    ARGA
INDR2    ARGB,NULL
.MACEXIT
.ENDIF
.IFNMA   4
WORD     ZBP       ; BUILD BUFFER AT AND POST AN OUTPUT FROM ARGA
INDR1    ARGA
INDR3    ARGB,ARGC,NULL
.MACEXIT
.ENDIF
.IFNMA   5
WORD     ZBP       ; BUILD BUFFER AT AND POST AN OUTPUT FROM ARGA
INDR1    ARGA
INDR4    ARGB,ARGC,ARGD,NULL
.MACEXIT
.ENDIF
.IFNMA   6
WORD     ZBP       ; BUILD BUFFER AT AND POST AN OUTPUT FROM ARGA
INDR1    ARGA
INDR5    ARGB,ARGC,ARGD,ARGE,NULL
.MACEXIT
.ENDIF
.IFNMA   7
WORD     ZBP       ; BUILD BUFFER AT AND POST AN OUTPUT FROM ARGA
INDR1    ARGA
INDR3    ARGB,ARGC,ARGD
INDR3    ARGE,ARGF,NULL
.MACEXIT
.ENDIF
.IFNMA   8
WORD     ZBP       ; BUILD BUFFER AT AND POST AN OUTPUT FROM ARGA
INDR1    ARGA
INDR3    ARGB,ARGC,ARGD
INDR4    ARGE,ARGF,ARGG,NULL
.MACEXIT
.ENDIF
.IFNMA   9
WORD     ZBP       ; BUILD BUFFER AT AND POST AN OUTPUT FROM ARGA
INDR1    ARGA
INDR4    ARGB,ARGC,ARGD,ARGE
INDR4    ARGF,ARGG,ARGH,NULL
.MACEXIT
.ENDIF
.IFNMA   10
WORD     ZBP       ; BUILD BUFFER AT AND POST AN OUTPUT FROM ARGA
INDR1    ARGA
INDR4    ARGB,ARGC,ARGD,ARGE
INDR5    ARGF,ARGG,ARGH,ARGI,NULL
.MACEXIT
.ENDIF
.IFNMA   11
WORD     ZBP       ; BUILD BUFFER AT AND POST AN OUTPUT FROM ARGA
INDR1    ARGA
INDR5    ARGB,ARGC,ARGD,ARGE,ARGF
INDR5    ARGG,ARGH,ARGI,ARGJ,NULL
.MACEXIT
.ENDIF
.IFMA    11
```

```
        WORD    ZBP     ; TOO MANY VARIABLES
        INDR1   ERROR
        .MACEXIT
        .ENDIF
        .ENDM
;
BPOSTN  .MACRO  ARGA,ARGB,ARGC,ARGD,ARGE,ARGF,ARGG,ARGH,ARGI,ARGJ
        .IFNDEF ZBNP
        EXTERN  ZBNP
        .ENDIF
        .IFNMA  1
        WORD    ZBNP
        INDR1   ERROR   ; TOO FEW VARIABLES
        .MACEXIT
        .ENDIF
        .IFNMA  2
        WORD    ZBNP
        INDR1   ERROR   ; TOO FEW VARIABLES
        .MACEXIT
        .ENDIF
        .IFNMA  3
        WORD    ZBNP    ; BUILD BUFFER AT ARGA
        INDR1   ARGA
        INDR2   ARGB,NULL
        .MACEXIT
        .ENDIF
        .IFNMA  4
        WORD    ZBNP    ; BUILD BUFFER AT ARGA
        INDR1   ARGA
        INDR3   ARGB,ARGC,NULL
        .MACEXIT
        .ENDIF
        .IFNMA  5
        WORD    ZBNP    ; BUILD BUFFER AT ARGA
        INDR1   ARGA
        INDR4   ARGB,ARGC,ARGD,NULL
        .MACEXIT
        .ENDIF
        .IFNMA  6
        WORD    ZBNP    ; BUILD BUFFER AT ARGA
        INDR1   ARGA
        INDR5   ARGB,ARGC,ARGD,ARGE,NULL
        .MACEXIT
        .ENDIF
        .IFNMA  7
        WORD    ZBNP    ; BUILD BUFFER AT ARGA
        INDR1   ARGA
        INDR3   ARGB,ARGC,ARGD
        INDR3   ARGE,ARGF,NULL
        .MACEXIT
        .ENDIF
        .IFNMA  8
        WORD    ZBNP    ; BUILD BUFFER AT ARGA
        INDR1   ARGA
        INDR3   ARGB,ARGC,ARGD
        INDR4   ARGE,ARGF,ARGG,NULL
        .MACEXIT
        .ENDIF
        .IFNMA  9
        WORD    ZBNP    ; BUILD BUFFER AT ARGA
```

```
        INDR1   ARGA
        INDR4   ARGB,ARGC,ARGD,ARGE
        INDR4   ARGF,ARGG,ARGH,NULL
        .MACEXIT
        .ENDIF
        .IFNMA  10
        WORD    ZBNP      ; BUILD BUFFER AT ARGA
        INDR1   ARGA
        INDR4   ARGB,ARGC,ARGD,ARGE
        INDR5   ARGF,ARGG,ARGH,ARGI,NULL
        .MACEXIT
        .ENDIF
        .IFNMA  11
        WORD    ZBNP      ; BUILD BUFFER AT ARGA
        INDR1   ARGA
        INDR5   ARGB,ARGC,ARGD,ARGE,ARGF
        INDR5   ARGG,ARGH,ARGI,ARGJ,NULL
        .MACEXIT
        .ENDIF
        .IFMA   11
        WORD    ZBNP
        INDR1   ERROR     ; TOO MANY VARIABLES
        .MACEXIT
        .ENDIF
        .ENDM
;
NPOSTO  .MACRO  ARGA
        .IFNDEF ZNBP
        EXTERN  ZNBP
        .ENDIF
        .IFNMA  1
        WORD    ZNBP
        INDR1   ERROR     ; TOO FEW VARIABLES
        .MACEXIT
        .ENDIF
        .IFNMA  2
        WORD    ZNBP
        INDR1   .ARGA     ; POST AN OUTPUT FROM ARGA
        .MACEXIT
        .ENDIF
        .IFMA   2
        WORD    ZPOUT
        INDR1   ERROR     ; TOO MANY VARIABLES
        .MACEXIT
        .ENDIF
        .ENDM
;
RAMTST  .MACRO  ARGA,ARGB,ARGC,ARGD
        .IFNDEF ZRAMT
        EXTERN  ZRAMT
        .ENDIF
        WORD    ZRAMT     ; TEST ARGA(1:ARGB), GOTO ARGD IF SUCCESSFUL, WRITE ERR
        INDR4   ARGA,ARGC,ARGB,ARGD
        .ENDM
;
TBIT1   .MACRO  ARGA,ARGB,ARGC
        .IFNDEF ZBIT1
        EXTERN  ZBIT1
        .ENDIF
        WORD    ZBIT1     ; IF ARGA(1:1) .AND. ARGB .NE. 0, GOTO ARGC
```

```
        INDR1   ARGA
        INDRS   ARGB
        INDR1   ARGC
        .ENDM
;
TBIT0   .MACRO  ARGA,ARGB,ARGC
        .IFNDEF ZBIT0
        EXTERN  ZBIT0
        .ENDIF
        WORD    ZBIT0       ; IF ARGA(1:1) .AND. ARGB .EQ. 0, GOTO ARGC
        INDR1   ARGA
        INDRS   ARGB
        INDR1   ARGC
        .ENDM
;
BEQS    .MACRO  ARGA,ARGB,ARGC
        .IFNDEF ZBEQS
        EXTERN  ZBEQS
        .ENDIF
        WORD    ZBEQS       ; IF ARGA(1:1) .EQ. ARGB, GOTO ARGC
        INDR1   ARGA
        INDRS   ARGB
        INDR1   ARGC
        .ENDM
;
BNES    .MACRO  ARGA,ARGB,ARGC
        .IFNDEF ZBNES
        EXTERN  ZBNES
        .ENDIF
        WORD    ZBNES       ; IF ARGA(1:1) .NE. ARGB, GOTO ARGC
        INDR1   ARGA
        INDRS   ARGB
        INDR1   ARGC
        .ENDM
;
BLES    .MACRO  ARGA,ARGB,ARGC
        .IFNDEF ZBLES
        EXTERN  ZBLES
        .ENDIF
        WORD    ZBLES       ; IF ARGA(1:1) .LE. ARGB, GOTO ARGC
        INDR1   ARGA
        INDRS   ARGB
        INDR1   ARGC
        .ENDM
;
BLTS    .MACRO  ARGA,ARGB,ARGC
        .IFNDEF ZBLTS
        EXTERN  ZBLTS
        .ENDIF
        WORD    ZBLTS       ; IF ARGA(1:1) .LT. ARGB, GOTO ARGC
        INDR1   ARGA
        INDRS   ARGB
        INDR1   ARGC
        .ENDM
;
BGES    .MACRO  ARGA,ARGB,ARGC
        .IFNDEF ZBGES
        EXTERN  ZBGES
        .ENDIF
        WORD    ZBGES       ; IF ARGA(1:1) .GE. ARGB, GOTO ARGC
```

```
        INDR1   ARGA
        INDRS   ARGB
        INDR1   ARGC
        .ENDM
;
BGTS    .MACRO  ARGA,ARGB,ARGC
        .IFNDEF ZBGTS
        EXTERN  ZBGTS
        .ENDIF
        WORD    ZBGTS     ; IF ARGA(1:1) .GT. ARGB, GOTO ARGC
        INDR1   ARGA
        INDRS   ARGB
        INDR1   ARGC
        .ENDM
;
INCS    .MACRO  ARGA,ARGB
        .IFNDEF ZINCS
        EXTERN  ZINCS
        .ENDIF
        WORD    ZINCS     ; INCREMENT ARGA(1:1) BY ARGB
        INDR1   ARGA
        INDRS   ARGB
        .ENDM
;
INCSBR  .MACRO  ARGA,ARGB,ARGC
        .IFNDEF ZINCSB
        EXTERN  ZINCSB
        .ENDIF
        WORD    ZINCSB    ; INCREMENT ARGA(1:1) BY ARGB, GOTO ARGC
        INDR1   ARGA
        INDRS   ARGB
        INDR1   ARGC
        .ENDM
;
INCSBZ  .MACRO  ARGA,ARGB,ARGC
        .IFNDEF ZINCSZ
        EXTERN  ZINCSZ
        .ENDIF
        WORD    ZINCSZ    ; INCREMENT ARGA(1:1) BY ARGB, BRANCH ZERO TO ARGC
        INDR1   ARGA
        INDRS   ARGB
        INDR1   ARGC
        .ENDM
;
INCSBN  .MACRO  ARGA,ARGB,ARGC
        .IFNDEF ZINCSN
        EXTERN  ZINCSN
        .ENDIF
        WORD    ZINCSN    ; INCREMENT ARGA(1:1) BY ARGB, BRANCH NON-ZERO TO ARGC
        INDR1   ARGA
        INDRS   ARGB
        INDR1   ARGC
        .ENDM
;
DECS    .MACRO  ARGA,ARGB
        .IFNDEF ZDECS
        EXTERN  ZDECS
        .ENDIF
        WORD    ZDECS     ; DECREMENT ARGA(1:2) BY ARGB
        INDR1   ARGA
```

```
        INDRS   ARGB
        .ENDM
;
DECSBR  .MACRO  ARGA,ARGB,ARGC
        .IFNDEF ZDECSB
        EXTERN  ZDECSB
        .ENDIF
        WORD    ZDECSB    ; DECREMENT ARGA(1:1) BY ARGB, GOTO ARGC
        INDR1   ARGA
        INDRS   ARGB
        INDR1   ARGC
        .ENDM
;
DECSBZ  .MACRO  ARGA,ARGB,ARGC
        .IFNDEF ZDECSZ
        EXTERN  ZDECSZ
        .ENDIF
        WORD    ZDECSZ    ; DECREMENT ARGA(1:1) BY ARGB, BRANCH ZERO TO ARGC
        INDR1   ARGA
        INDRS   ARGB
        INDR1   ARGC
        .ENDM
;
DECSBN  .MACRO  ARGA,ARGB,ARGC
        .IFNDEF ZDECSN
        EXTERN  ZDECSN
        .ENDIF
        WORD    ZDECSN    ; DEC ARGA(1:1) BY ARGB, BRANCH NON-ZERO TO ARGC
        INDR1   ARGA
        INDRS   ARGB
        INDR1   ARGC
        .ENDM
;
SETS    .MACRO  ARGA,ARGB
        .IFNDEF ZSETS
        EXTERN  ZSETS
        .ENDIF
        WORD    ZSETS     ; SET ARGA(1:1) WITH ARGB
        INDR1   APGA
        INDRS   ARGB
        .ENDM
;
SETSBR  .MACRO  ARGA,ARGB,ARGC
        .IFNDEF ZSETSB
        EXTERN  ZSETSB
        .ENDIF
        WORD    ZSETSB    ; SET ARGA(1:1) WITH ARGB,GOTO ARGC
        INDR1   ARGA
        INDRS   ARGB
        INDR1   ARGC
        .ENDM
;
SUPR    .MACRO  ARGA,ARGB,ARGC
        .IFNDEF ZSUPR
        EXTERN  ZSUPR
        .ENDIF
        WORD    ZSUPR     ; REPLACE ALL LEADING AND TRAILING ARGB(S) WITH ARGC(S)
        INDR3   ARGA,ARGB,ARGC
        .ENDM
;
```

```
COMP    .MACRO  ARGA,ARGB,ARGC,ARGD
        .IFNDEF ZCOMP
        EXTERN  ZCOMP
        .ENDIF
        WORD    ZCOMP     ; IF ARGA(1:ARGC) .EQ. ARGB(1:ARGC), THEN GOTO ARGD
        INDR4   ARGA,ARGB,ARGC,ARGD
        .ENDM
;
COMPE   .MACRO  ARGA,ARGB,ARGC,ARGD
        .IFNDEF ZCOMPE
        EXTERN  ZCOMPE
        .ENDIF
        WORD    ZCOMPE    ; IF ARGA(1:ARGC) .NE. ARGB(1:ARGC), THEN GOTO ARGD
        INDR4   ARGA,ARGB,ARGC,ARGD
        .ENDM
;
INTERV  .MACRO  ARGA,ARGB,ARGC
        .IFNDEF ZCTC
        EXTERN  ZCTC
        .ENDIF
        WORD    ZCTC      ; INTERVALOMETER ARGA
        .IFSAME ARGA,ENABLE
        INDRBU  0
        INDRBL  SAV0
        INDR3   1001H,ARGB,ARGC
        .MACEXIT
        .ENDIF
        .IFSAME ARGA,SETUP
        INDRBU  0
        INDRBL  SAV0
        INDR3   0001H,0,0
        .MACEXIT
        .ENDIF
        .IFSAME ARGA,DISABLE
        INDRBU  2
        INDRBL  0
        INDR1   0
        .MACEXIT
        .ENDIF
        .IFSAME ARGA,SAVE/ENABLE
        INDRBU  4
        INDRBL  SAV0
        INDR3   1001H,ARGB,ARGC
        .MACEXIT
        .ENDIF
        .IFSAME ARGA,SAVE/DISABLE
        INDRBU  6
        INDRBL  SAV0
        INDR1   0
        .MACEXIT
        .ENDIF
        .IFSAME ARGA,RESTORE
        INDRBU  8
        INDRBL  SAV0
        INDR1   0
        .MACEXIT
        .ENDIF
        WORD    ERROR
        .ENDM
;
```

```
FTIMER  .MACRO  ARGA,ARGB,ARGC
        .IFNDEF ZCTC
        EXTERN  ZCTC
        .ENDIF
        WORD    ZCTC        ; FREQUENCY TIMER ARGA
        .IFSAME ARGA,ENABLE
        INDRBU  1
        INDRBL  SAV1
        INDR3   4010H,ARGB,ARGC
        .MACEXIT
        .ENDIF
        .IFSAME ARGA,SETUP
        INDRBU  1
        INDRBL  SAV1
        INDR3   0010H,0,0
        .MACEXIT
        .ENDIF
        .IFSAME ARGA,DISABLE
        INDRBU  3
        INDRBL  0
        INDR1   0
        .MACEXIT
        .ENDIF
        .IFSAME ARGA,SAVE/ENABLE
        INDRBU  5
        INDRBL  SAV1
        INDR3   4010H,ARGB,ARGC
        .MACEXIT
        .ENDIF
        .IFSAME ARGA,SAVE/DISABLE
        INDRBU  7
        INDRBL  SAV1
        INDR1   0
        .MACEXIT
        .ENDIF
        .IFSAME ARGA,RESTORE
        INDRBU  9
        INDRBL  SAV1
        INDR1   0
        .MACEXIT
        .ENDIF
        WORD    ERROR
        .ENDM
;
INDEX   .MACRO  ARGA,ARGB,ARGC,ARGD,ARGE
        .IFNDEF ZINDX
        EXTERN  ZINDX
        .ENDIF
        WORD    ZINDX       ; ARGA(1:2) = INDEX(ARGB(1:ARGC),ARGD(1:ARGE))
        INDR5   ARGD,ARGB,ARGE,ARGC,ARGA
        .ENDM
;
XFER    .MACRO  ARGA,ARGB,ARGC
        .IFNDEF ZXFER
        EXTERN  ZXFER
        .ENDIF
        WORD    ZXFER       ; ARGA(1:ARGC) = ARGB(1:ARGC)
        INDR3   ARGA,ARGB,ARGC
        .ENDM
;
```

```
XFERBR  .MACRO  ARGA,ARGB,ARGC,ARGD
        .IFNDEF ZXFERB
        EXTERN  ZXFERB
        .ENDIF
        WORD    ZXFERB    ; ARGA(1:ARGC) = ARGB(1:ARGC), GOTO ARGD
        INDR4   ARGA,ARGB,ARGC,ARGD
        .ENDM
;
TWOCMP  .MACRO  ARGA,ARGB,ARGC
        .IFNDEF Z2CMP
        EXTERN  Z2CMP
        .ENDIF
        WORD    Z2CMP     ; ARGA(1:ARGC) = COMPLEMENT(ARGB(1:ARGC))+1
        INDR3   ARGA,ARGB,ARGC
        .ENDM
;
VECTOR  .MACRO  ARGA
        .IFNDEF ZVECT
        EXTERN  ZVECT
        .ENDIF
        WORD    ZVECT             ; VECTOR THROUGH ARGA
        INDR1   ARGA
        .ENDM
;
SWINT   .MACRO  ARGA
        .IFNDEF ZSWI
        EXTERN  ZSWI
        .ENDIF
        .IFSAME ARGA,DISABLE
        WORD    ZSWI      ; DISABLE SOFTWARE INTERRUPT
        INDR1   SWINTM
        INDRS   0
        .MACEXIT
        .ENDIF
        .IFSAME ARGA,ENABLE
        WORD    ZSWI      ; ENABLE SOFTWARE INTERRUPT
        INDR1   SWINTM
        INDRS   80H
        INDR1   SWADR
        .MACEXIT
        .ENDIF
        .IFSAME ARGA,RETURN
        WORD    ZSWI      ; ENABLE SOFTWARE INTERRUPT
        INDR1   SWINTM
        INDRS   A0H
        .MACEXIT
        .ENDIF
        WORD    ERROR
        .ENDM
;
INDADR  .MACRO  ARG
ARG     EQU     $
i|ARG   EQU     $+ZI*10000H
        GLOBAL  ARG,i|ARG
        .ENDM
;
VARLST  .MACRO  ARGA,ARGB,ARGC,ARGD
        .IFNMA  1
        WORD    ERROR
        .MACEXIT
```

```
.ENDIF
.IFNMA  2
ASCII   ARGA
ASCII   /
WORD    ARGA
BYTE    TERM
.MACEXIT
.ENDIF
.IFNMA  3
ASCII   ARGA
ASCII   /
WORD    ARGA
ASCII   ARGB
ASCII   /
WORD    ARGB
BYTE    TERM
.MACEXIT
.ENDIF
.IFNMA  4
ASCII   ARGA
ASCII   /
WORD    ARGA
ASCII   ARGB
ASCII   /
WORD    ARGB
ASCII   ARGC
ASCII   /
WORD    ARGC
BYTE    TERM
.MACEXIT
.ELSE
ASCII   ARGA
ASCII   /
WORD    ARGA
ASCII   ARGB
ASCII   /
WORD    ARGB
ASCII   ARGC
ASCII   /
WORD    ARGC
ASCII   ARGD
ASCII   /
WORD    ARGD
BYTE    TERM
.ENDIF
.ENDM
```

APPENDIX 2

```
      SUBROUTINE DECOMP (ITIME,IFWA,ILWA,IPASS,IEXIT)

      INTEGER*2         IBUF(3072),IFWA,ILWA,IPASS,IEXIT,JOPEN
      INTEGER*2         IX,JX,KX,IPICK,IPUT,JPUT,JBUF(650),MYFMT,FCHAR

      INTEGER*4         ICKSUM,ICKSUM1,ITIME(7),INCVLU

      CHARACTER*8       FILENAME

$INCLUDE: 'RESPCM.INC'

C
C  Compressed Data is in the response buffer
C
      IF (IPASS .LT. 0) GOTO 72
      IF (IPASS .GT. 0) GOTO 10

        IPUT = 1
        IPICK = 1
        JPUT = 1
        IPASS = 1

        DUMPFILE = 'DUMP.' // SERNUM

        JOPEN = 0
        FILENAME = DUMPFILE
        FCHAR = 8
        OPEN (JDISK,FILE=DUMPFILE,ACCESS='SEQUENTIAL',
     +        STATUS='UNKNOWN',IOSTAT=DISK,ERR=104)
        JOPEN = 1
C
C  Checksum the RESPONSE buffer
C
   10   ICKSUM = 0
        DO 12 IX=IFWA,ILWA-10
          ICKSUM = ICKSUM+ICHAR(RESP(IX:IX))
   12   CONTINUE

        READ (RESP(ILWA-8:ILWA-1),7012,ERR=100) ICKSUM1
 7012   FORMAT (Z8)
        IF (ICKSUM .NE. ICKSUM1) GOTO 100
C
C  Checksum passed, read the input data into IBUF
C
        ASSIGN 8012 TO MYFMT
 8012   FORMAT('Error Reading Input Data'A)

        DO 20 IX=IFWA,ILWA-10,2
          READ (RESP(IX:IX+1),9012,ERR=70) IBUF(IPUT)
 9012       FORMAT (Z2)
            IPUT = IPUT+1
   20   CONTINUE
C       WRITE (IDISK,*) 'IPUT=',IPUT
        IF (RESP(ILWA:ILWA) .EQ. '>') IPASS = 2
C
C  Decompression...
C  Check if JBUF has a frame
C  Check if done
```

CA2183123

```
C  Check if incomplete
C  Generate repeat count or word count and check if zero (error)
C
   30    IF (JPUT .GE. 641) GOTO 40
         IF (IPICK+1 .GE. IPUT .AND. IPASS .EQ. 2) GOTO 72
         IF (IPICK+1 .GE. IPUT) RETURN

         ASSIGN 9030 TO MYFMT
 9030    FORMAT ('Word Count or Repeat Count Zero',A)

         IX = ISHFT(IAND(IBUF(IPICK),16#0F),8)+IBUF(IPICK+1)
         IF (IX .EQ. 0) GOTO 70
C
C  Repeat field processing here
C
         IF (IAND(IBUF(IPICK),16#F0) .EQ. 16#A0) THEN
           IF (IPICK+2 .GE. IPUT) RETURN
             IPICK = IPICK+2

   32        JBUF(JPUT) = IBUF(IPICK)
C            WRITE (IDISK,9032) JPUT,JBUF(JPUT),IX
C 9032       FORMAT ('JBUF(',I3,')=',I4,1X,'IX=',I6)
             JPUT = JPUT+1
             IX = IX-1
             IF (IX .GT. 0) GOTO 32
             IPICK = IPICK+1
             GOTO 30
C
C  Eight bit data processing here
C
         ELSEIF (IAND(IBUF(IPICK),16#F0) .EQ. 16#00) THEN
           IF (IPICK+2+IX-1 .GE. IPUT) RETURN
             IPICK = IPICK+2

   34        JBUF(JPUT) = IBUF(IPICK)
C            WRITE (IDISK,9032) JPUT,JBUF(JPUT),IX
             JPUT = JPUT+1
             IPICK = IPICK+1
             IX = IX-1
             IF (IX .GT. 0) GOTO 34
             GOTO 30
C
C  Four bit data processing here
C
         ELSEIF (IAND(IBUF(IPICK),16#F0) .EQ. 16#80) THEN
           JX = ISHFT(IX,-1)+IAND(IX,16#0001)
           IF (IPICK+2+JX-1 .GE. IPUT) THEN
C            WRITE (6,*) IX,'-',JX
C            WRITE (6,*) IPICK,'-',IPUT
C            WRITE (6,*) IBUF(IPUT-2),IBUF(IPUT-1)
             RETURN
           ENDIF
             IPICK = IPICK+2

   36        JBUF(JPUT) = ISHFT(IBUF(IPICK),-4)
C            WRITE (IDISK,9032) JPUT,JBUF(JPUT),IX
             JPUT = JPUT+1
             IX = IX-1
             IF (IX .EQ. 0) GOTO 38
               JBUF(JPUT) = IAND(IBUF(IPICK),16#0F)
```

CA2183123

```
C               WRITE (IDISK,9032) JPUT,JBUF(JPUT),IX
                JPUT = JPUT+1
   38           IPICK = IPICK+1
                IX = IX-1
                IF (IX .GT. 0) GOTO 36
              GOTO 30
C
C  Error processing here
C
         ELSE
           ASSIGN 9038 TO MYFMT
 9038      FORMAT ('Unrecognized Control Byte',A)
           GOTO 70
         ENDIF
C
C  We are ready to file a spectral frame
C
   40    ASSIGN 9040 TO MYFMT
 9040    FORMAT ('Long Frame Error',A)

         IF (JPUT .GT. 641) GOTO 70

         IX = 1
         JX = 1
C
C  Check Frame ID...
C  Operational frame if Frame ID is 15 or 31 else data
C
   42    ASSIGN 9042 TO MYFMT
 9042    FORMAT ('SpectraScan Frame Counter Range Error',A)

         IF (JBUF(IX) .LT. 0 .OR. JBUF(IX) .GT. 31) GOTO 70

       IF (JBUF(IX) .EQ. 15 .OR. JBUF(IX) .EQ. 31) THEN
         IX = IX+1
         IOBUF(JX) = INT4(JBUF(IX))
         IX = IX+19
         JX = JX+1
       ELSE
         IX = IX+1
         DO 44 KX=2,20
           IF (KX .LE. 17 .OR. KX .GE. 20) THEN
             IOBUF(JX) = INT4(JBUF(IX))
             JX = JX+1
           ENDIF
           IX = IX+1
44       CONTINUE
       ENDIF
       IF (IX .LT. 641) GOTO 42

         ASSIGN 9044 TO MYFMT
 9044    FORMAT ('Short Frame Error',A)

         IF (JX .NE. 513) GOTO 70
C
C  Scale channels 1 thru 356 (JBUF 1 thru 356)
C
         DO 46 IX=1,356
           IOBUF(IX) = ISHFT(IOBUF(IX),JBUF(316))
   46    CONTINUE
```

```
C
C  Write the spectral record to disk
C
      JX = 0

      WRITE (JDISK,8046,ERR=108,IOSTAT=DISK)
     +   (ITIME(IX),IX=4,6),ITIME(7)/100,JX,JX,
     +    1023-JBUF(303)*32+JBUF(304),JBUF(305),JBUF(315),
     +    JBUF(317)*32+JBUF(320),JX,INCVLU,INT1(JBUF(319)-16#80)
 8046 FORMAT (2(I2.2,':'),I2.2,'.',I1.1,8(I6),2X,I4)

      DO 48 IX=1,512,16
        WRITE (JDISK,9046,ERR=108,IOSTAT=DISK)
     +    IX-1,(IOBUF(JX),JX=IX,IX+15)
 9046   FORMAT (I5,':',4X,16(I6))
   48 CONTINUE

      WRITE (JDISK,'(1X)',ERR=108,IOSTAT=DISK)
C
C  Write the ccl record to disk
C
      IX = 1
      WRITE (JDISK,9048,ERR=108,IOSTAT=DISK)
     +          IX,(INT1(JBUF(JX)-16#80),JX=19,319,20)
 9048 FORMAT (I5,':',4X,16(2X,I4))

      IX=17
      WRITE (JDISK,9048,ERR=108,IOSTAT=DISK)
     +          IX,(INT1(JBUF(JX)-16#80),JX=339,639,20)

      WRITE (JDISK,'(1X)',ERR=108,IOSTAT=DISK)
C
C  Increment time word by 1600 milliseconds
C
      INCVLU = 1600
      CALL TBUMP (ITIME,INCVLU)
C
C  Reposition the leftover spectral data in IBUF
C
C     WRITE (IDISK,*) 'IPUT-IPICK=',IPUT-IPICK,IPUT,IPICK

      JX = 1
      KX = IPUT-IPICK
      IF (KX .LE. 0) GOTO 62

        JX = 1
   60   IBUF(JX) = IBUF(IPICK)
        JX = JX+1
        IPICK = IPICK+1
        KX = KX-1
        IF (KX .NE. 0) GOTO 60

   62 IPICK = 1
      IPUT = JX
C     WRITE (IDISK,*) 'IPUT-IPICK=',IPUT-IPICK,IPUT,IPICK
C     WRITE (IDISK,*) IBUF(1),IBUF(2),IBUF(3),IBUF(4)
      JPUT = 1
      GOTO 30
C
C  Non read data errors handled here
```

```
C
   70    WRITE (CMD,MYFMT) CHAR(0)
         CALL CHRDSP1 (CMD)
         CALL CHRDSP1 (CHAR(0))
         WRITE (IDISK,MYFMT,ERR=650,IOSTAT=DISK) CHAR(32)
         WRITE (IDISK,'(1X)',ERR=650,IOSTAT=DISK)
         IEXIT = 2
C
C  Close the dump file
C
   72    IF (JOPEN .EQ. 0) RETURN
         CLOSE (JDISK,ERR=110,IOSTAT=DISK)
         JOPEN = 0
         RETURN
C
C  Checksum error handled here
C
  100    WRITE (CMD,9100) ICKSUM,RESP(ILWA-8:ILWA-1),CHAR(0)
 9100    FORMAT('Checksum Failure: Calculated ',Z8,' Sent ',2A)
         CALL CHRDSP1 (CMD)
         CALL CHRDSP1 (CHAR(0))
         WRITE (IDISK,9100,ERR=650,IOSTAT=DISK)
     +          ICKSUM,RESP(ILWA-8:ILWA-1),CHAR(32)
         WRITE (IDISK,'(1X)',ERR=650,IOSTAT=DISK)

         IEXIT = IEXIT+1
         RETURN
C
C  Non-Fatal Disk Errors Handled Here
C
  104    ASSIGN 9104 TO MYFMT
 9104    FORMAT ('Unable to Open ',A,2X,'IOSTAT = ',I7,A)
 106     WRITE (CMD,MYFMT) FILENAME(1:FCHAR),DISK,CHAR(0)
         CALL CHRDSP1 (CMD)
         CALL CHRDSP1 (CHAR(0))
         WRITE (IDISK,MYFMT,ERR=650,IOSTAT=DISK)
     +    FILENAME(1:FCHAR),DISK,CHAR(32)
         WRITE (IDISK,'(1X)',ERR=650,IOSTAT=DISK)
         IF (JOPEN .EQ. 1) CLOSE (JDISK,ERR=110,IOSTAT=DISK)
         IEXIT = -1
         RETURN

 108     ASSIGN 9108 TO MYFMT
9108     FORMAT ('Error Writing ',A,2X,'IOSTAT = ',I7,A)
         GOTO 106

 110     ASSIGN 9110 TO MYFMT
9110     FORMAT ('Unable to Close ',A,2X,'IOSTAT = ',I7,A)
         JOPEN = 0
         GOTO 106
C
C  Fatal error handled here
C
  650      WRITE (CMD,9650) SAVEFILE,DISK,CHAR(0)
 9650      FORMAT ('Error writing ',A,2X,'IOSTAT = ',I7,A)
           CALL CHRDSP1 (CMD)
           CALL CHRDSP1 (CHAR(0))
           DISKERR = 1
           IEXIT = -3
```

CLAIMS

What is claimed is:

1. A well logging instrument for gamma ray spectroscopy, comprising:

a gamma ray spectral detector that provides digital gamma ray spectral data for a plurality of energy spectra;

a controller coupled to the gamma ray spectral detector and receiving the digital gamma ray spectral data, the controller compressing the digital gamma ray spectral data to provide compressed digital gamma ray spectral data; and

a memory system coupled to the controller, the memory system receiving and storing the compressed digital gamma ray spectral data.

2. The logging instrument of claim 1, wherein the gamma ray spectral detector includes a log/calibration mode in which a voltage to a photomultiplier tube is dynamically adjusted and a latched mode in which the voltage to the photomultiplier tube is latched, and wherein the controller includes an output for activating the latched mode in the gamma ray spectral detector in response to the digital gamma ray spectral data exceeding a predetermined threshold.

3. The logging instrument of claim 2, wherein the digital gamma ray spectral data includes data corresponding to 60 Kev gamma rays from an americium gamma ray source, and wherein the predetermined threshold corresponds to the 60 Kev gamma rays from the americium gamma ray source being masked by other gamma ray sources.

4. A well logging instrument for gamma ray spectroscopy, comprising:

a gamma ray spectral detector that provides digital gamma ray spectral data for a plurality of energy spectra, that includes a log/calibration mode in which the gamma ray spectral detector dynamically adjusts a voltage to a photomultiplier tube, and that includes a latched mode in which the gamma ray spectral detector latches the voltage to the photomultiplier tube; and

a controller coupled to the gamma ray spectral detector and receiving the digital gamma ray spectral data, the controller providing an output for activating the latched mode in the gamma ray spectral detector in response to the digital gamma ray spectral data exceeding a predetermined threshold.

5. The logging instrument of claim 4, wherein the digital gamma ray spectral data includes data corresponding to 60 Kev gamma rays from an americium gamma ray source, and wherein the predetermined threshold corresponds to the 60 Kev gamma rays from the americium gamma ray source being masked by other gamma ray sources.

6. A method of logging gamma ray spectral data using a gamma ray spectroscopy instrument that includes a gamma ray spectral detector, a controller, and a memory, the method comprising the steps of:

receiving digital data from the gamma ray spectral detector;

compressing the gamma ray digital data according to a predetermined algorithm; and

storing the compressed digital data in the memory.

7. The method of claim 6 further comprising the steps of:

after the storing step, retrieving the compressed digital data, decompressing the data, and providing the data to an analysis system.

8. The method of claim 6 further comprising the steps of:

before the step of storing the data, delaying a predetermined period of time corresponding to the amount of time necessary to place the instrument into an appropriate position before the storing step.

9. The method of claim 6 in which the digital data includes a plurality of energy spectra elements, and wherein the compressing step further comprises the steps of:

if a next sequence of digital data includes a sequence of identical values, storing as the compressed data a code indicating a repeat sequence, a count of the identical value, and a single instance of the identical values;

if a next sequence of digital data includes a sequence of non-repeating values and the values are less than 16, storing as the compressed data a code indicating a sequence of 4-bit non-repeating values, a count of non-repeating 4-bit values, and a sequence of non-repeating values as a series of 4-bit values, two to an 8-bit byte; and

if a next sequence of digital data includes a sequence of non-repeating values and the values are greater than 15, storing as the compressed data a code

count of non-repeating 8-bit values, and a sequence of non-repeating values as a series of 8-bit bytes.

10. A well logging instrument for gamma spectroscopy, comprising:

a battery;

a gamma ray spectral detector that provides digital gamma ray spectral data for a plurality of energy spectra, the gamma ray spectral detector coupled to and powered by the battery;

a controller coupled to the gamma ray spectral detector and receiving the digital gamma ray spectral data, the controller being coupled to and powered by the battery; and

a memory system coupled to the controller, the memory system receiving and storing data from the controller.

11. A method of logging gamma ray spectral data in a well using a gamma ray spectroscopy instrument that includes a gamma ray spectral detector, a controller, a battery, and a memory, the method comprising the steps of:

providing power from the battery to the gamma ray spectral detector, the controller, and the memory;

lowering the gamma ray spectroscopy instrument into the well;

acquiring data from the gamma ray spectral detector in the controller; and

storing the data from the gamma ray spectral detector in the memory.

12. The method of claim 11, wherein the step of lowering the gamma ray spectroscopy instrument into the well further comprises the step of:

lowering the gamma ray spectroscopy instrument into the well using a slick line.

13. The method of claim 11, wherein the step of lowering the gamma ray spectroscopy instrument into the well further comprises the step of:

lowering the gamma ray spectroscopy instrument into the well using coiled tubing.

14. The method of claim 11, wherein the step of storing the data further comprises the step of:

compressing the data from the gamma ray spectral detector.

15. The method of claim 11, wherein before the step of storing, executing the step of:

delaying a predetermined period of time sufficient to complete the lowering step.

FIG. 1

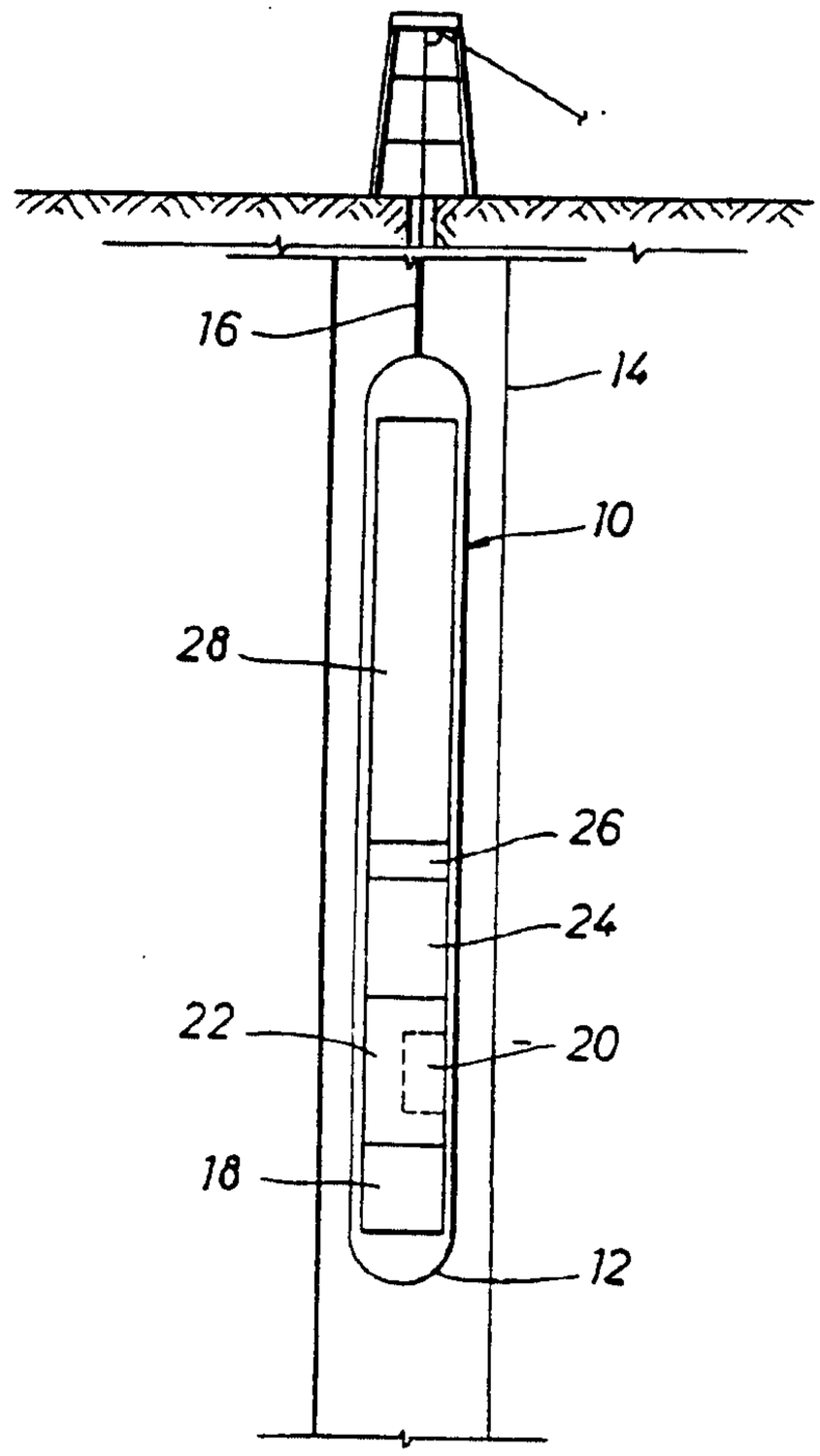
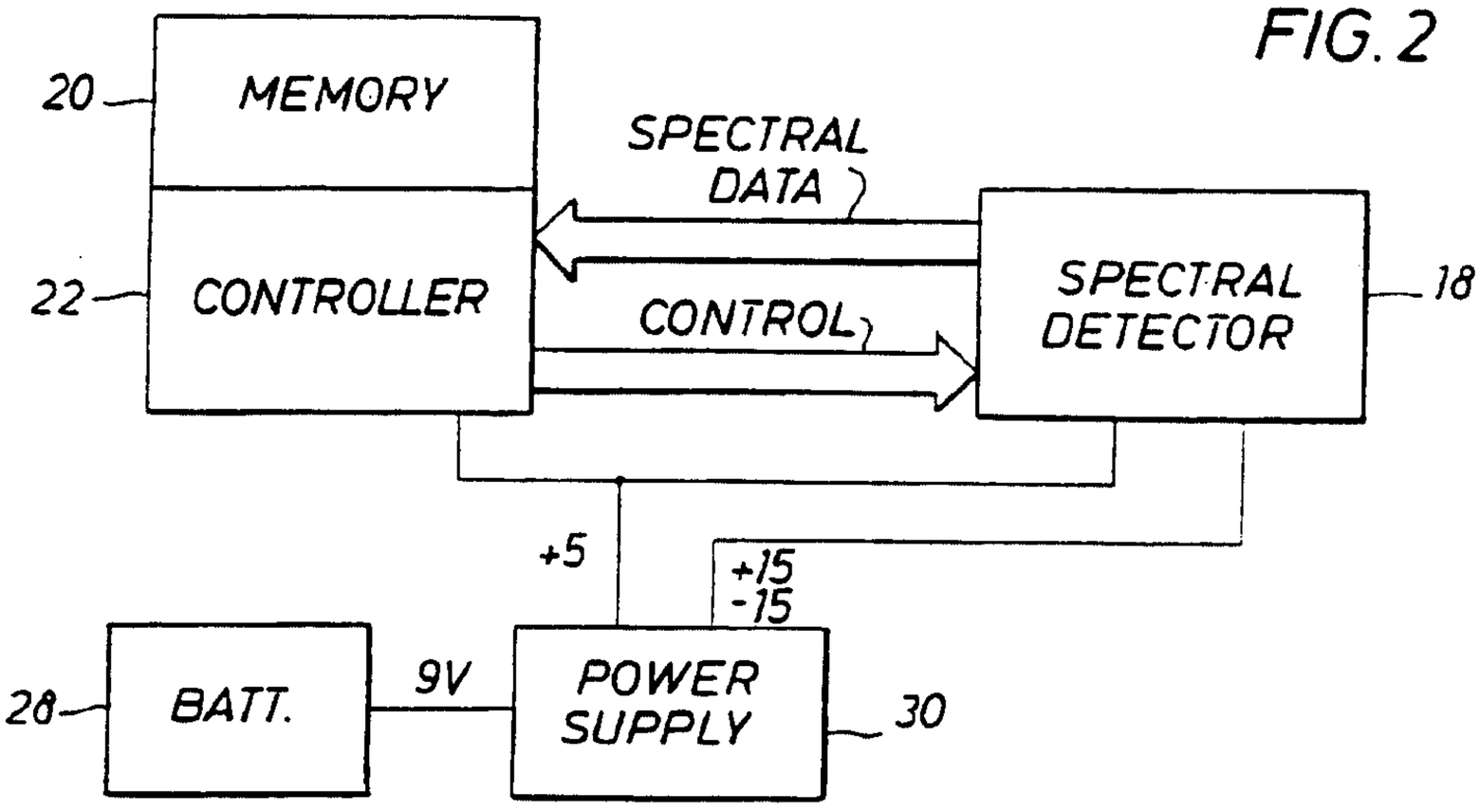

16
14
10
28
26
24
22
20
18
12

FIG. 2


20
MEMORY
SPECTRAL DATA
22
CONTROLLER
CONTROL
SPECTRAL DETECTOR
18
+5
+15
-15
28
BATT.
9V
POWER SUPPLY
30

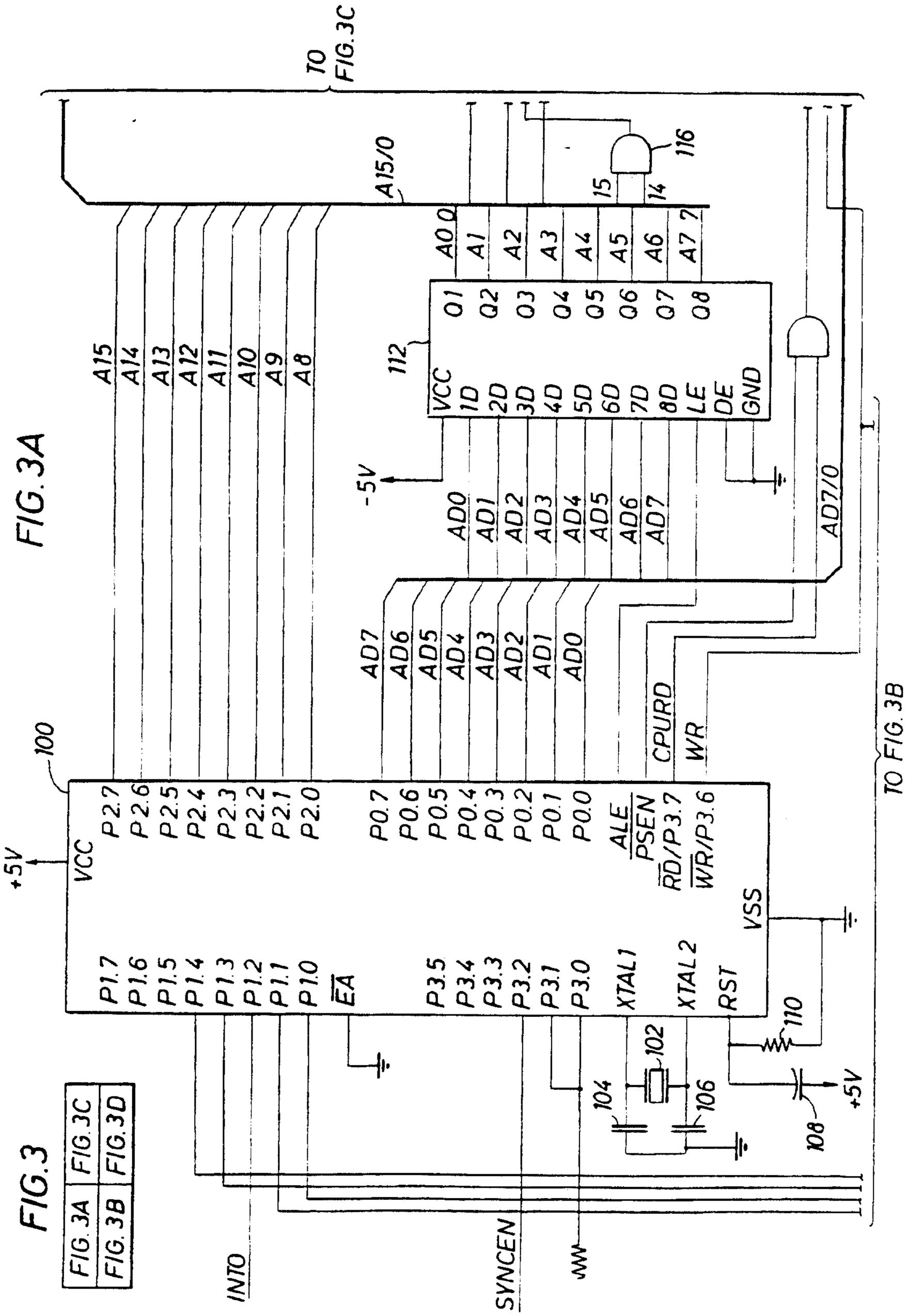


FIG. 3A

FIG. 3

FIG. 3A	FIG. 3C
FIG. 3B	FIG. 3D

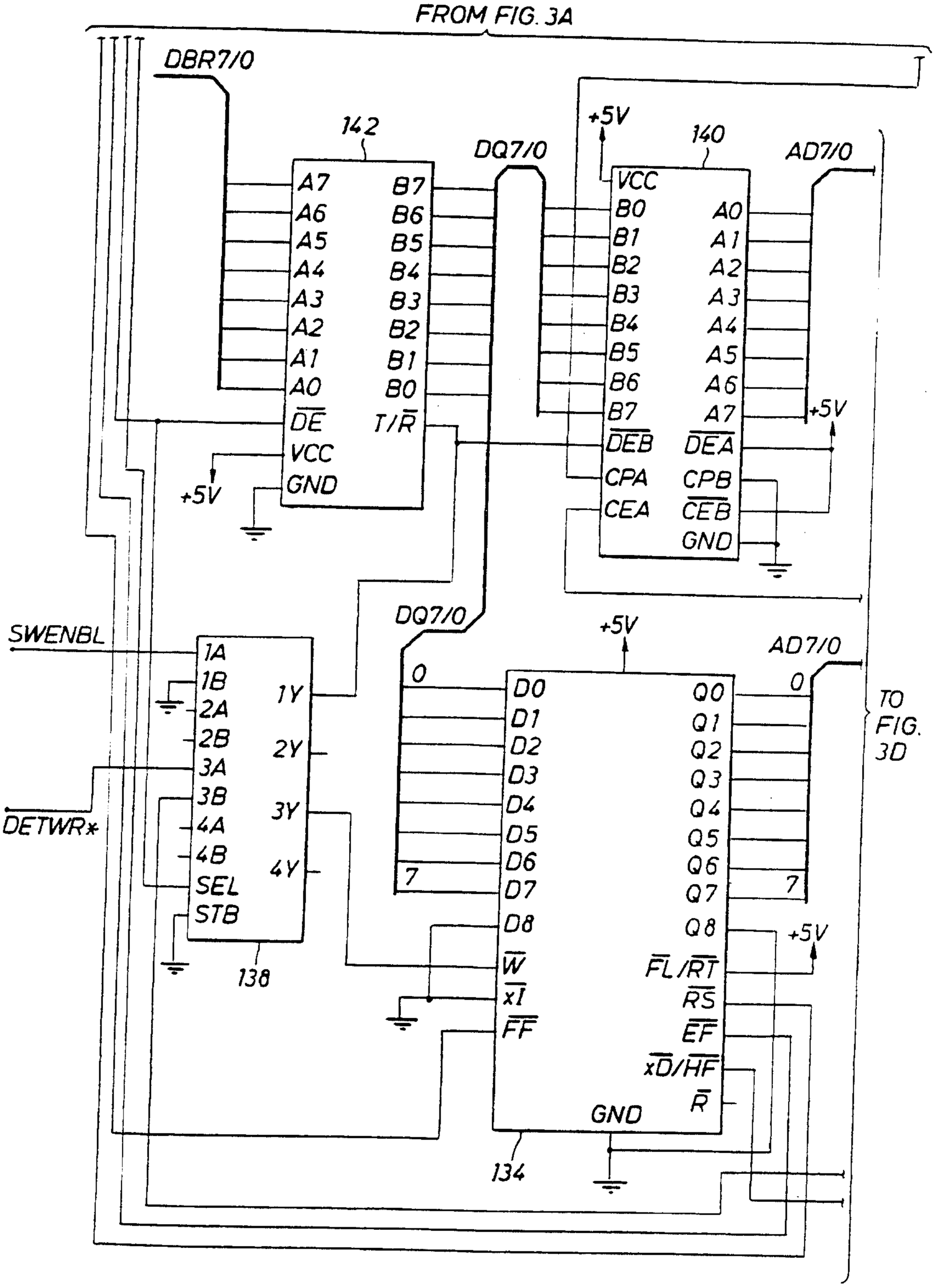

FROM FIG. 3A
DBR7/0
142
DQ7/0
+5V
140
AD7/0
A7
A6
A5
A4
A3
A2
A1
A0
DE
VCC
GND
B7
B6
B5
B4
B3
B2
B1
B0
T/R
VCC
B0
B1
B2
B3
B4
B5
B6
B7
DEB
CPA
CEA
A0
A1
A2
A3
A4
A5
A6
A7
DEA
CPB
CEB
GND
+5V
+5V
DQ7/0
SWENBL
DETWR*
1A
1B
2A
2B
3A
3B
4A
4B
SEL
STB
1Y
2Y
3Y
4Y
138
+5V
AD7/0
0
7
D0
D1
D2
D3
D4
D5
D6
D7
D8
W
xI
FF
Q0
Q1
Q2
Q3
Q4
Q5
Q6
Q7
Q8
FL/RT
RS
EF
xD/HF
R
GND
0
7
+5V
TO FIG. 3D
134

FIG. 3B

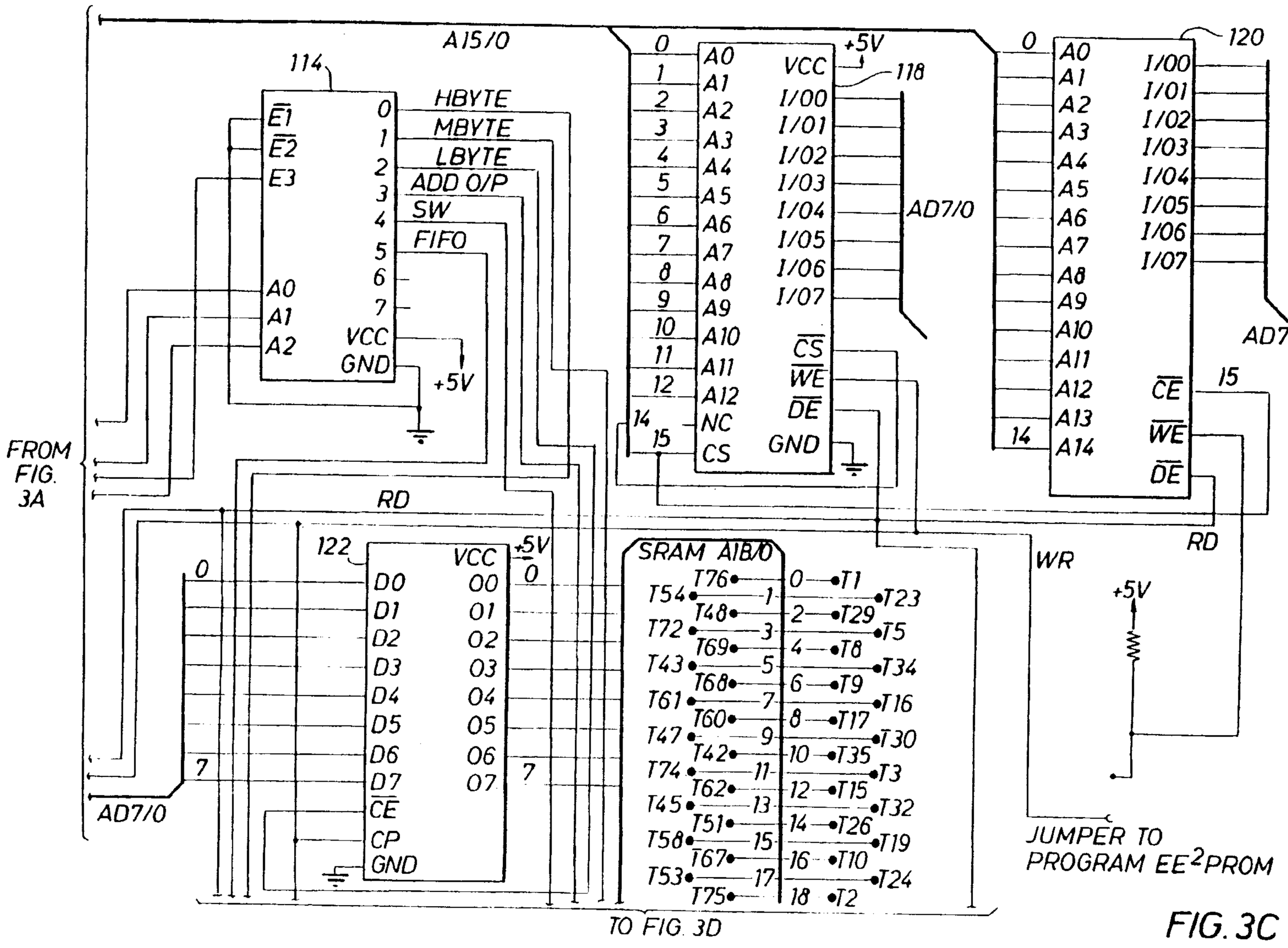


FIG. 3C

CA 02183123 2000-01-12

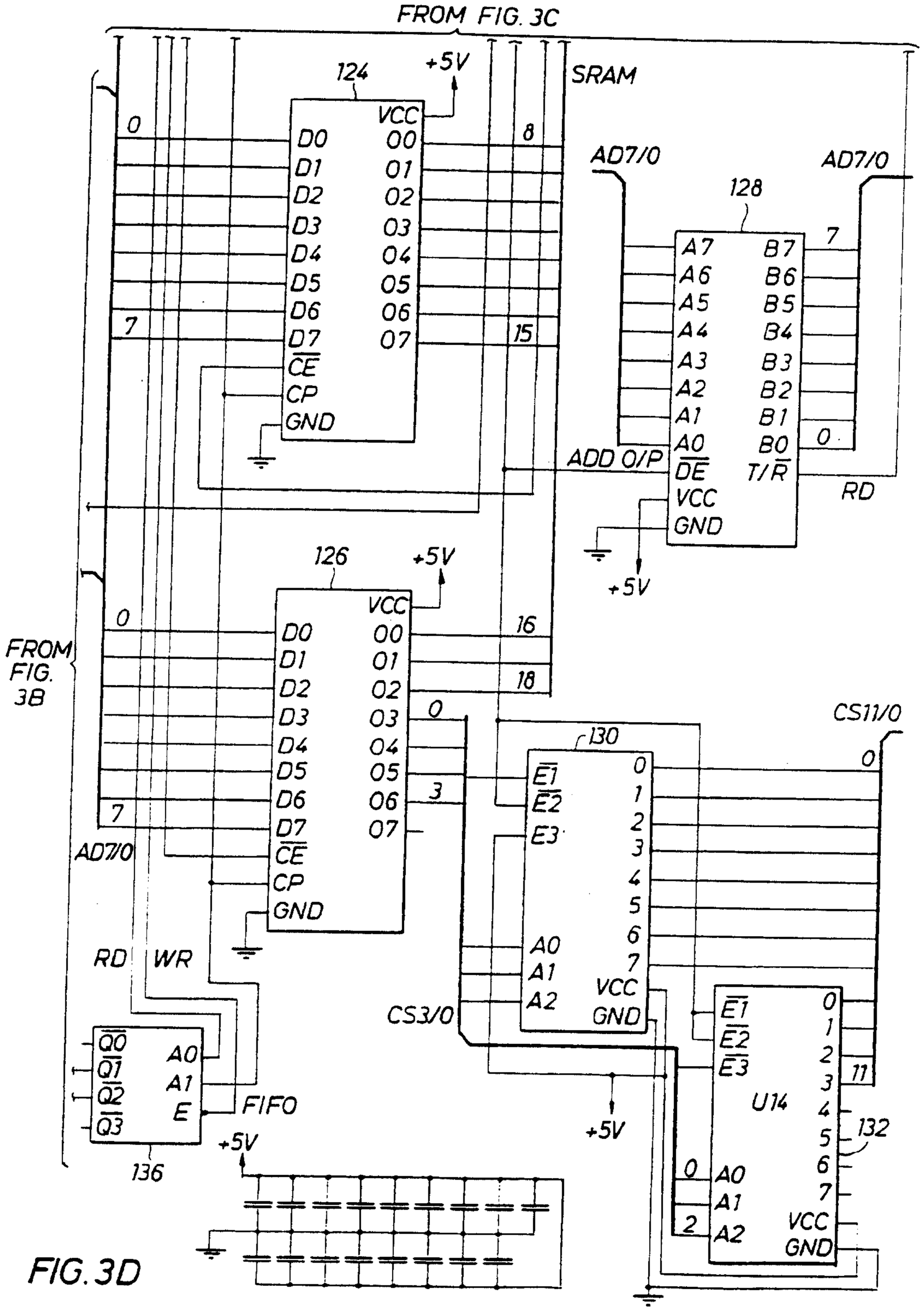

FROM FIG. 3C
SRAM
124
+5V
VCC
D0
D1
D2
D3
D4
D5
D6
D7
CE
CP
GND
O0
O1
O2
O3
O4
O5
O6
O7
0
7
8
15
AD7/0
AD7/0
128
A7
A6
A5
A4
A3
A2
A1
A0
ADD O/P
DE
T/R
VCC
GND
B7
B6
B5
B4
B3
B2
B1
B0
7
0
RD
+5V
126
+5V
16
18
FROM FIG. 3B
0
7
AD7/0
RD
WR
130
CS11/0
CS3/0
E1
E2
E3
A0
A1
A2
VCC
GND
0
1
2
3
4
5
6
7
Q0
Q1
Q2
Q3
A0
A1
E
FIFO
+5V
136
U14
11
132
0
2

FIG. 3D

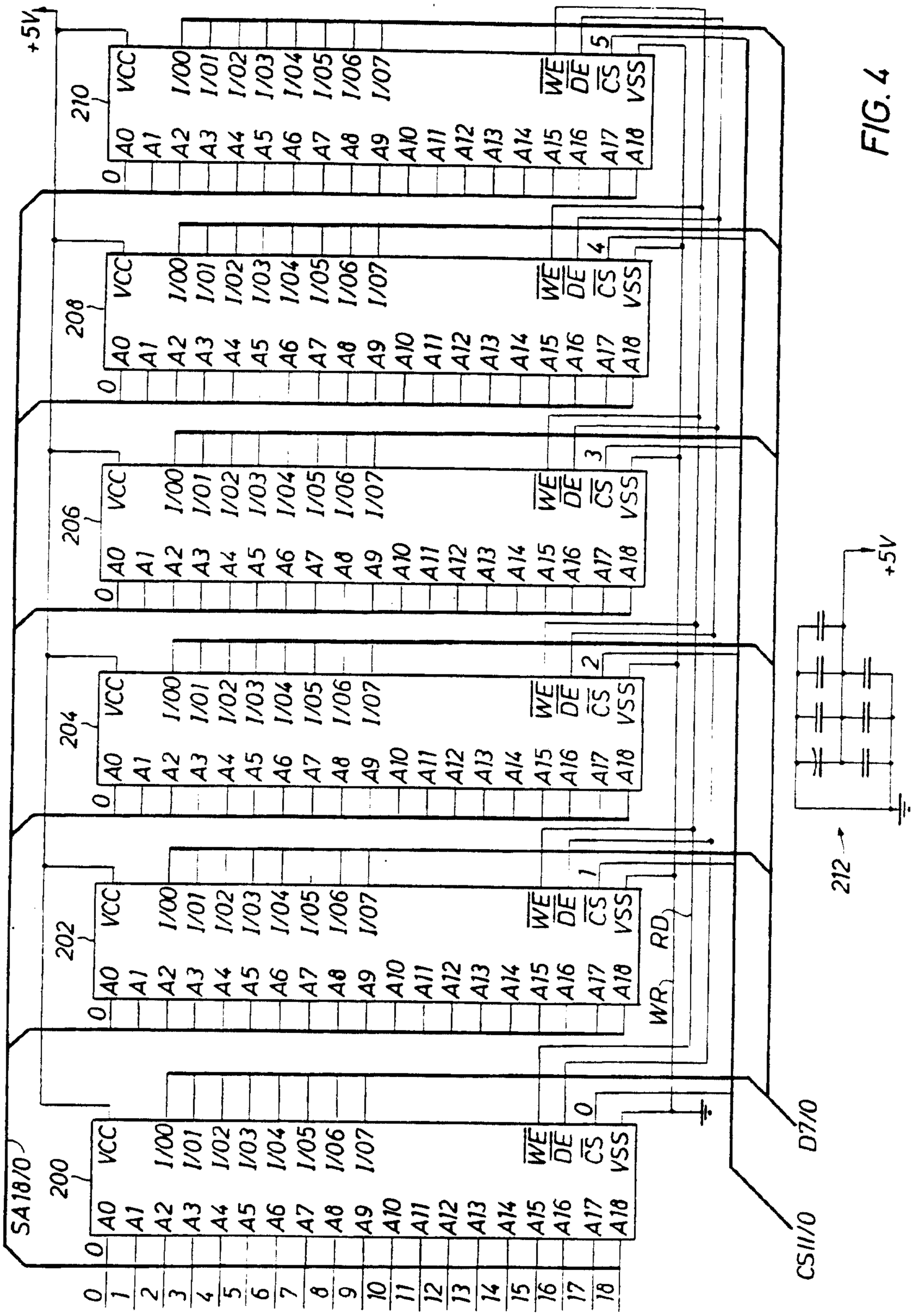

+5V
210
VCC
I/O0
I/O1
I/O2
I/O3
I/O4
I/O5
I/O6
I/O7
WE
DE
CS
VSS
5
A0
A1
A2
A3
A4
A5
A6
A7
A8
A9
A10
A11
A12
A13
A14
A15
A16
A17
A18
0
208
4
206
3
204
2
202
1
200
0
SA18/0
D7/0
CS11/0
WR
RD
212
+5V

FIG. 4

FIG. 5A

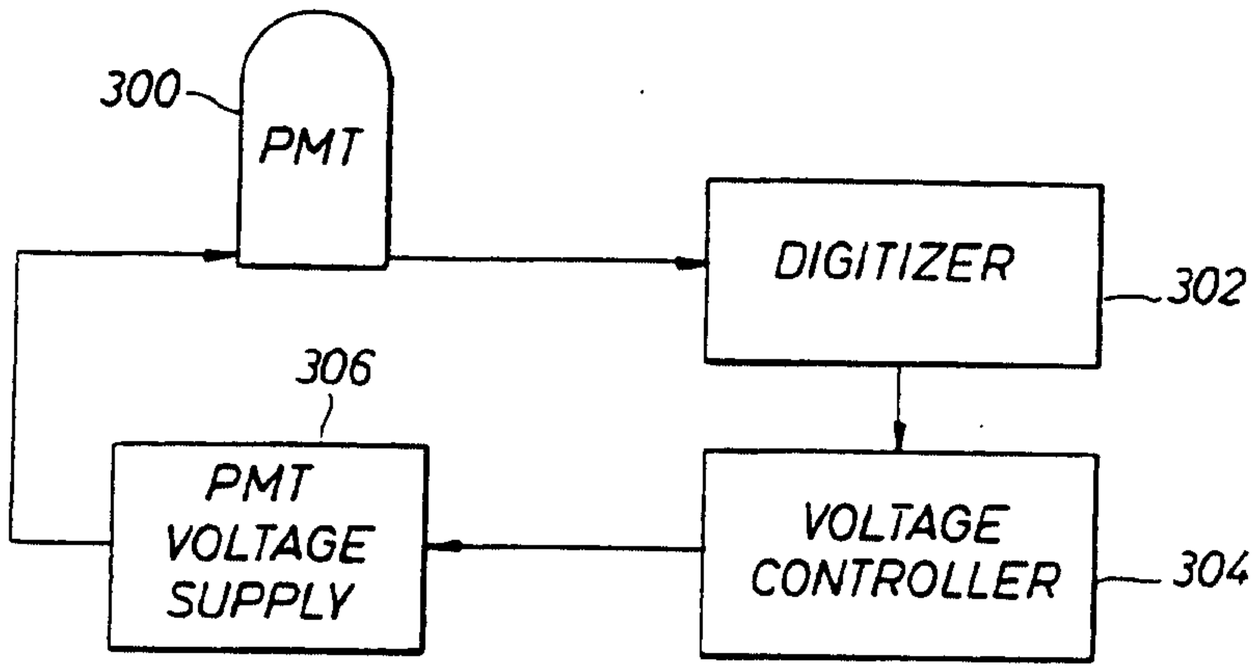
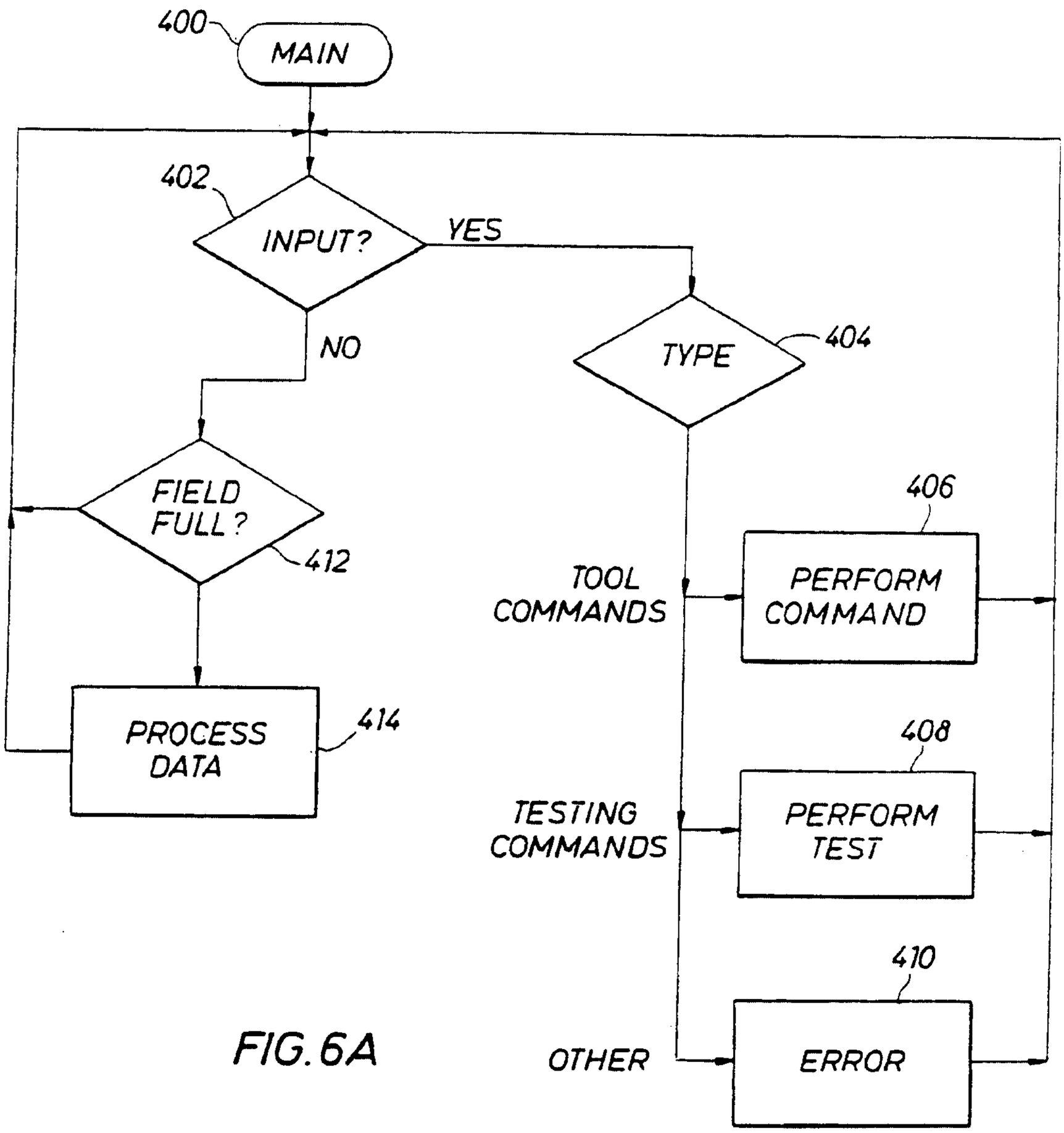

300
PMT
DIGITIZER
302
306
PMT VOLTAGE SUPPLY
VOLTAGE CONTROLLER
304

FIG. 6A


400
MAIN
402
INPUT?
YES
NO
FIELD FULL?
412
PROCESS DATA
414
TYPE
404
TOOL COMMANDS
406
PERFORM COMMAND
TESTING COMMANDS
408
PERFORM TEST
OTHER
410
ERROR

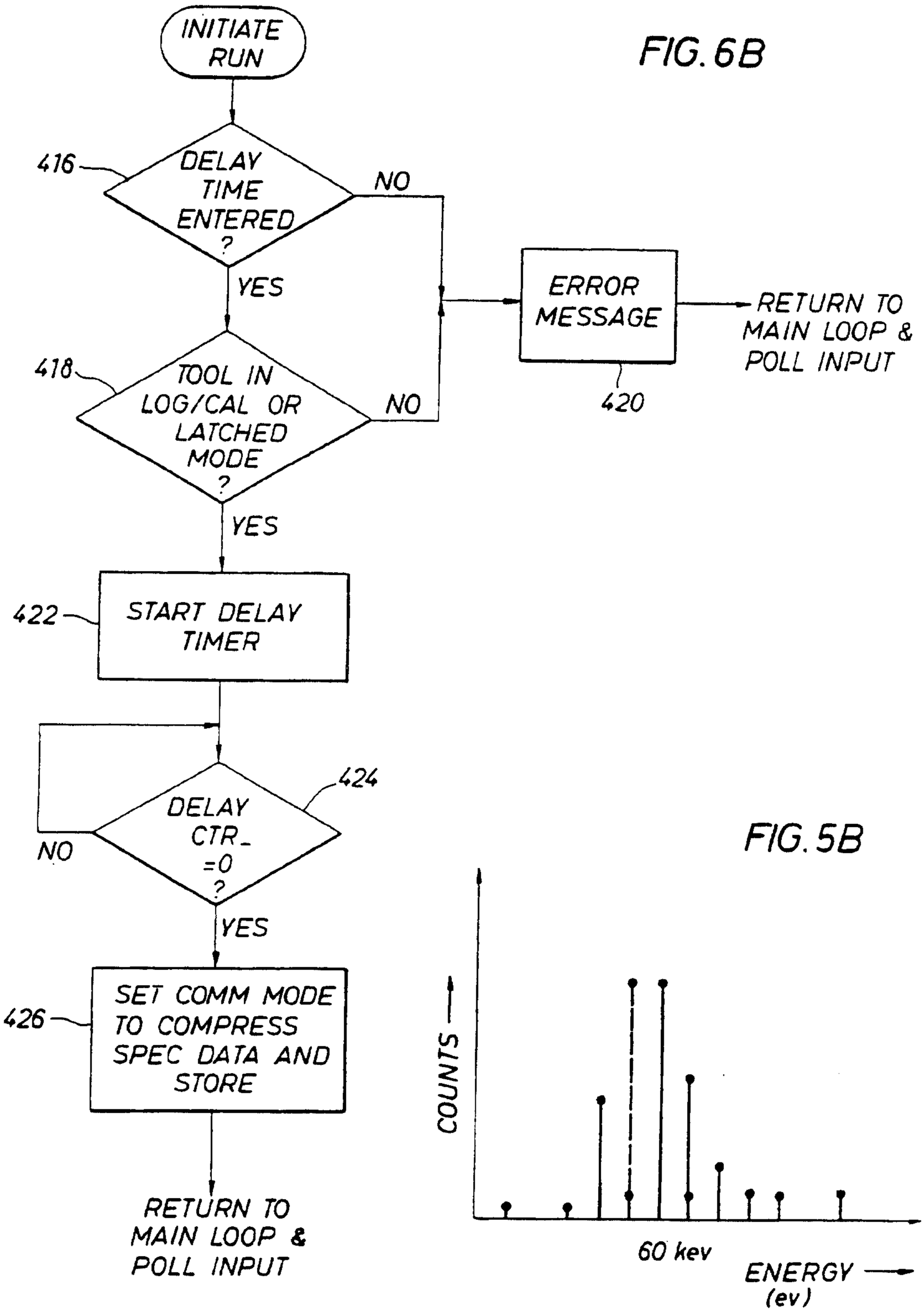

INITIATE RUN
416
DELAY TIME ENTERED ?
NO
YES
418
TOOL IN LOG/CAL OR LATCHED MODE ?
NO
YES
ERROR MESSAGE
RETURN TO MAIN LOOP & POLL INPUT
420
422
START DELAY TIMER
424
DELAY CTR_ =0 ?
NO
YES
426
SET COMM MODE TO COMPRESS SPEC DATA AND STORE
RETURN TO MAIN LOOP & POLL INPUT

FIG. 6B

FIG. 5B

COUNTS
60 kev
ENERGY
(ev)

FIG. 5C

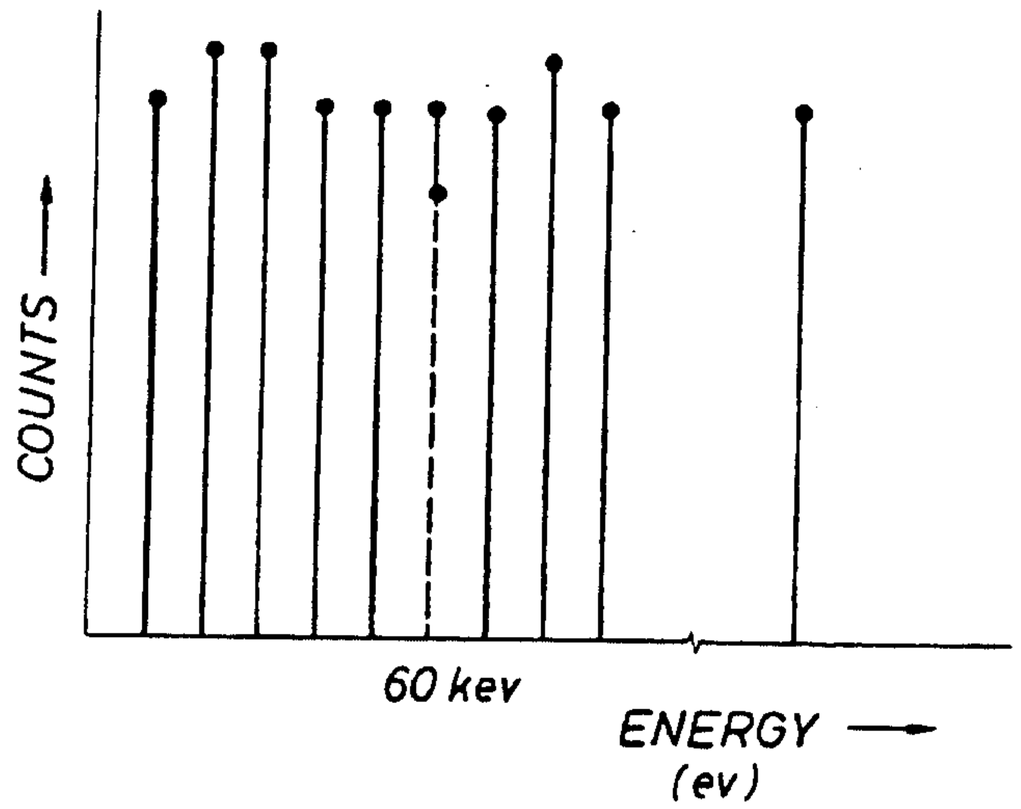

COUNTS
60 kev
ENERGY
(ev)

FIG. 6C

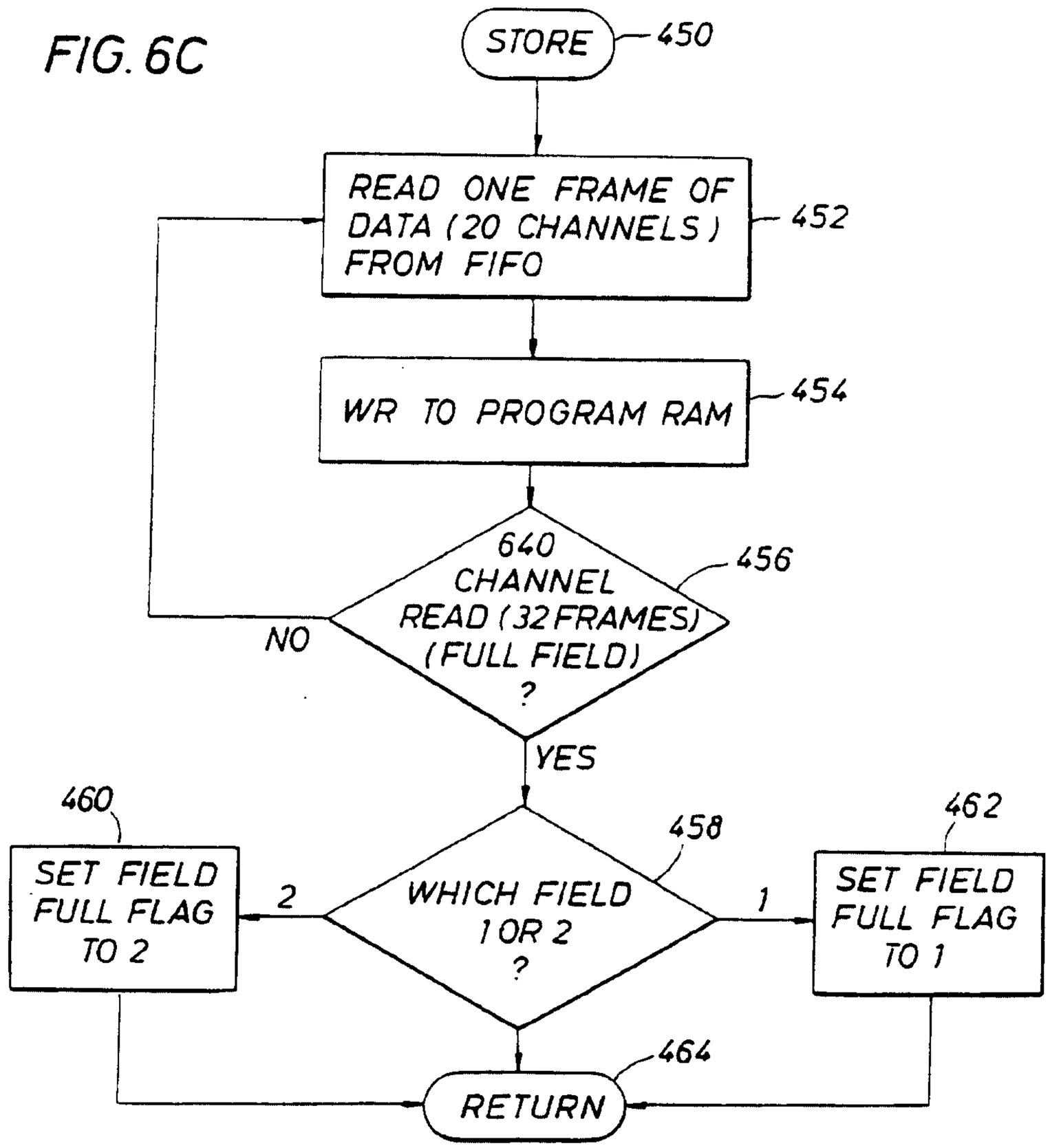
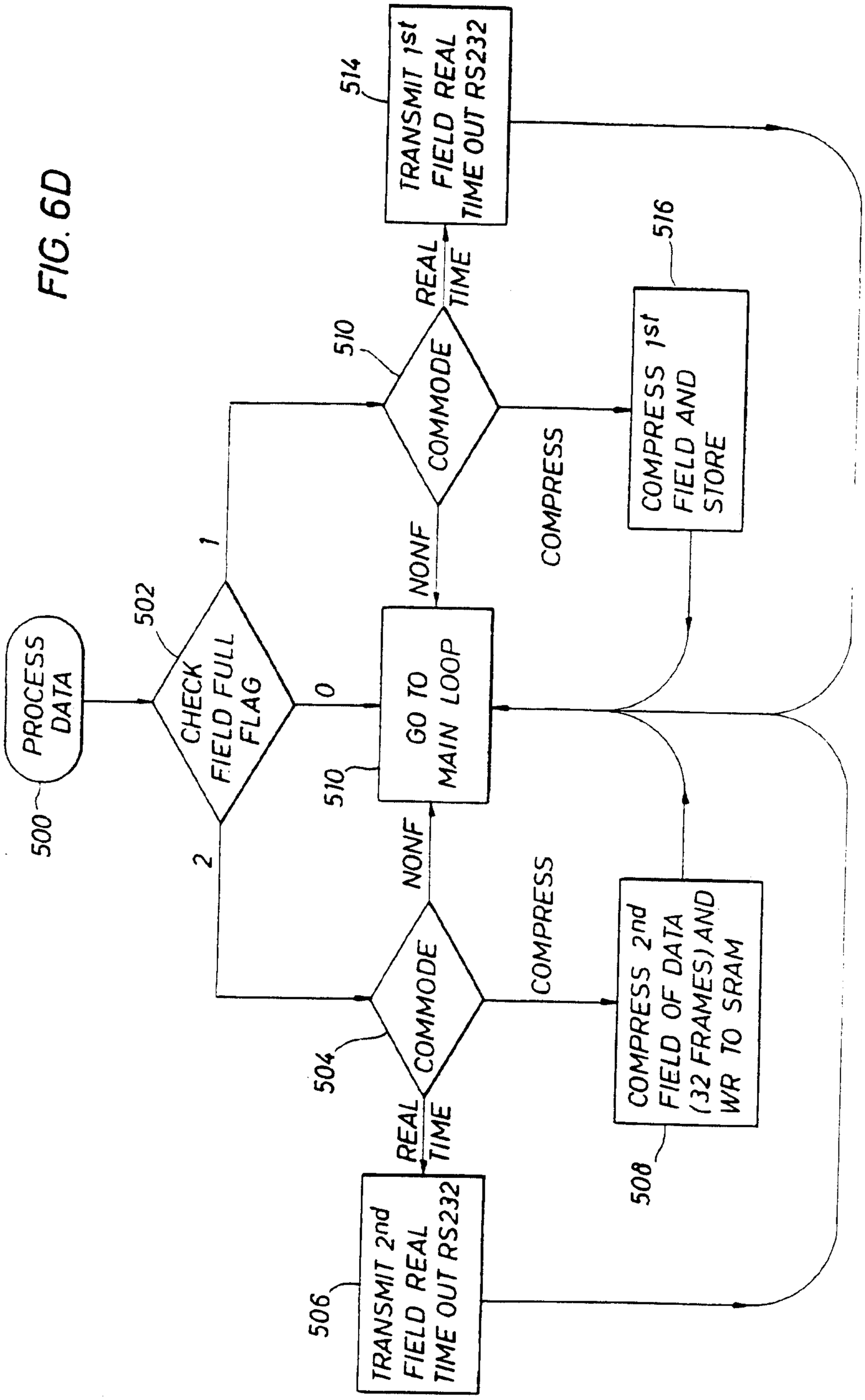
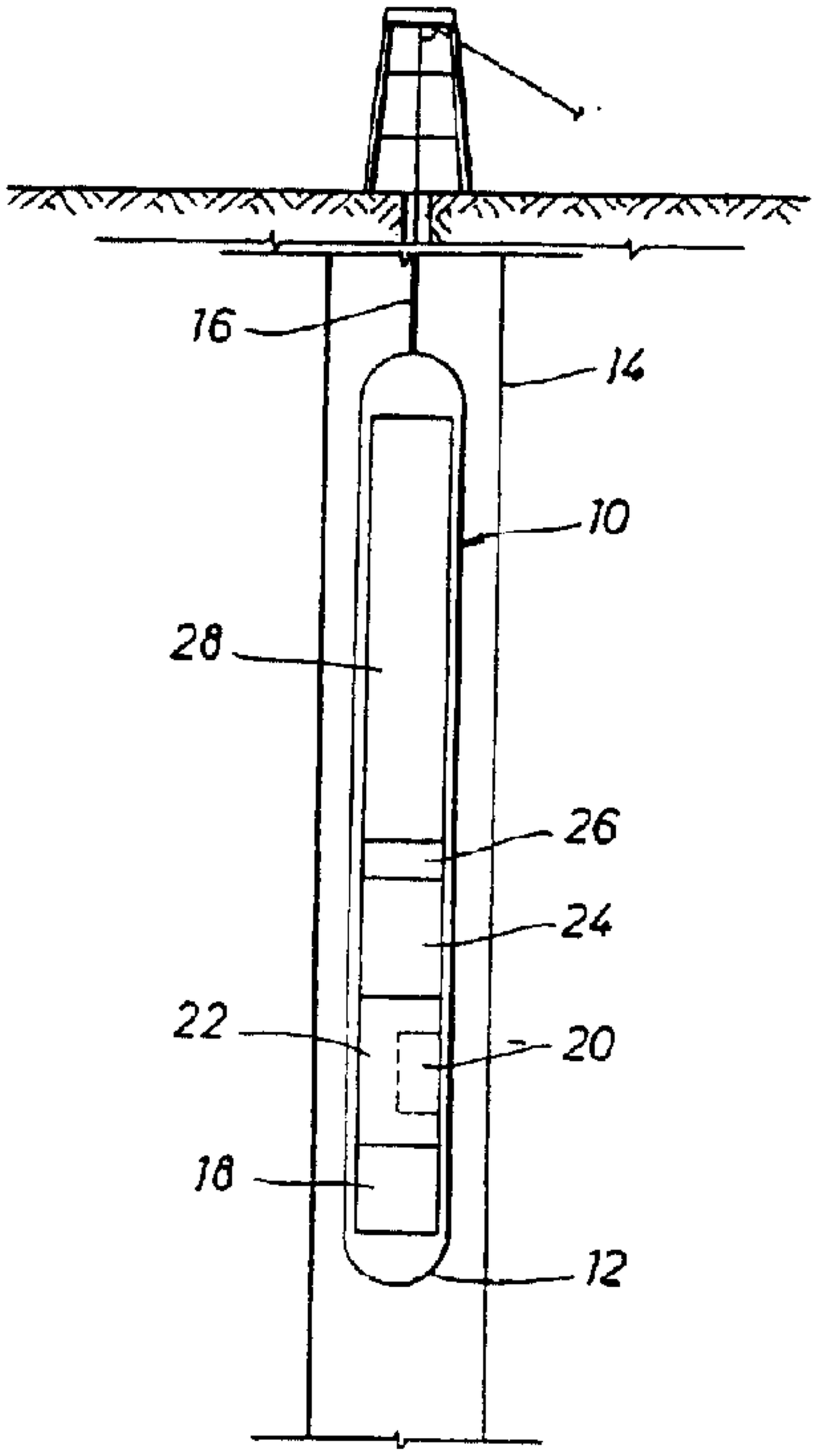

STORE
450
READ ONE FRAME OF DATA (20 CHANNELS) FROM FIFO
452
WR TO PROGRAM RAM
454
640 CHANNEL READ (32 FRAMES) (FULL FIELD) ?
456
NO
YES
458
WHICH FIELD 1 OR 2 ?
2
1
460
SET FIELD FULL FLAG TO 2
462
SET FIELD FULL FLAG TO 1
464
RETURN

FIG. 6D


PROCESS DATA
500
CHECK FIELD FULL FLAG
502
1
2
0
TRANSMIT 1st FIELD REAL TIME OUT RS232
514
COMMODE
510
REAL TIME
NONF
COMPRESS
COMPRESS 1st FIELD AND STORE
516
GO TO MAIN LOOP
510
COMMODE
504
REAL TIME
NONF
COMPRESS
COMPRESS 2nd FIELD OF DATA (32 FRAMES) AND WR TO SRAM
508
TRANSMIT 2nd FIELD REAL TIME OUT RS232
506


16
14
10
28
26
24
22
20
18
12