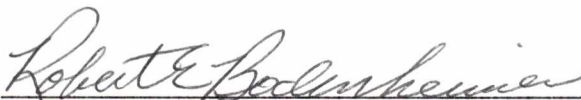
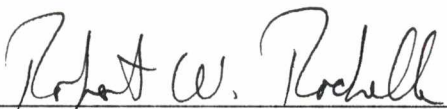


To the Graduate Council:

I am submitting herewith a thesis written by Rex C. Trammell entitled "A Microprocessor Based Data Acquisition Instrument for Gamma Ray Spectroscopy." I have examined the final copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Electrical Engineering.

  
Robert E. Bodenheimer, Major Professor

We have read this thesis  
and recommend its acceptance:

  
Robert W. Rochelle

  
Marshall Pace

Accepted for the Council:

  
Vice Provost  
and Dean of the Graduate School

# STATEMENT OF PERMISSION TO USE

In presenting this thesis in partial fulfillment of the requirements for the Master's degree at the University of Tennessee, Knoxville, I agree that the Library shall make it available to borrowers under rules of the Library. Brief quotations from this thesis are allowable without special permission, provided that accurate acknowledgement of the source is made.

Permission for extensive quotation from or reproduction of this thesis may be granted by my major professor, or in his absence, by the Head of Interlibrary Services when, in the opinion of either, the proposed use of the material is for scholarly purposes. Any copying or use of the material in this thesis for financial gain shall not be allowed without my written permission.

Signature   
Date 5/4/88

A MICROPROCESSOR BASED DATA ACQUISITION  
INSTRUMENT FOR GAMMA RAY  
SPECTROSCOPY

A Thesis  
Presented for the  
Master of Science  
Degree  
The University of Tennessee, Knoxville

Rex C. Trammell

June 1988

## ACKNOWLEDGEMENTS

The author wishes to thank the many people at Ortec who have contributed to this work. Special thanks must go to T. J. Paulus, who encouraged both the development of the Gammalyzer and pursuing the Master's Degree. In his unique way, Mario Martini provided strong inducement to enter the field of Electrical Engineering.

Bob Bodenheimer first introduced this author to microprocessors. It was his interest and enthusiasm that eventually led to this project. His teaching and support for nearly twenty years is gratefully acknowledged.

This work might never have been completed without the helpful advice and sometimes gentle persuasion of the author's wife, Murf.



## ABSTRACT

Germanium gamma ray spectrometers are extremely low noise, high resolution systems. A special purpose instrument, the multichannel analyzer (MCA), is used to digitize and histogram the data from these systems. MCA's have been in use for nearly 40 years and are highly developed devices. The subject of this thesis is a new architecture for the MCA, using a microprocessor based data acquisition system coupled with a personal computer.

A discussion of the complete germanium spectroscopy system is given, with special attention to those characteristics of the system which impact the design of the new instrument. Classic MCA architecture is discussed and contrasted to the new architecture. Design of a high speed data interface to a personal computer is given in detail.

Software is shown to be a key element in the new instrument. Methods used to design well engineered software are discussed and examples are given.

## TABLE OF CONTENTS

| CHAPTER                                           | PAGE |
|---------------------------------------------------|------|
| I. INTRODUCTION .....                             | 1    |
| II. THE GERMANIUM GAMMA RAY SPECTROMETER .....    | 3    |
| III. A NEW APPROACH - THE GAMMALYZER .....        | 20   |
| IV. THE DUAL PORT MEMORY - IBM PC INTERFACE ..... | 28   |
| V. THE SOFTWARE .....                             | 45   |
| VI. FUTURE WORK .....                             | 58   |
| LIST OF REFERENCES .....                          | 60   |
| APPENDIX .....                                    | 62   |
| VITA .....                                        | 70   |

## LIST OF FIGURES

| FIGURE |                                                 | PAGE |
|--------|-------------------------------------------------|------|
| II-1   | Ge Gamma Ray Spectroscopy System .....          | 4    |
| II-2   | Exponential Cumulative Distribution Function .. | 5    |
| II-3   | Coaxial Germanium Detector .....                | 7    |
| II-4   | Charge Sensitive Preamplifier .....             | 8    |
| II-5   | Preamplifier Output .....                       | 10   |
| II-6   | Equivalent Noise Sources .....                  | 11   |
| II-7   | Throughput Curve .....                          | 13   |
| II-8   | Classic Multichannel Analyzer .....             | 15   |
| III-1  | 7010 MCA Block Diagram .....                    | 21   |
| III-2  | Gammalyzer Block Diagram .....                  | 27   |
| IV-1   | 6116 Memory Device .....                        | 31   |
| IV-2   | Z-80 Microprocessor .....                       | 34   |
| IV-3   | IBM PC Bus Diagram .....                        | 35   |

| FIGURE                                  | PAGE |
|-----------------------------------------|------|
| IV-4 Shared Memory Block Diagram .....  | 36   |
| IV-5 Arbitrator .....                   | 39   |
| IV-6 PC Selector Circuit .....          | 40   |
| IV-7 Gammalyzer Selector Circuit .....  | 43   |
| IV-8 Completed Shared Memory .....      | 44   |
| V-1 Structured Control .....            | 49   |
| V-2 MCA Emulator Display Screen .....   | 51   |
| V-3 Full and Expanded Display .....     | 54   |
| V-4 Structured English Pseudocode ..... | 55   |

## CHAPTER I

### INTRODUCTION

In 1962 Freck and Wakefield<sup>1</sup> published a description of a new type of gamma ray spectrometer based on a germanium diode structure. After many years of constant development, the germanium detector is now the instrument of choice for gamma ray spectroscopy. The small output signal and very high resolution of these detectors place severe constraints on the electronic signal processing chain. Signals ranging from about 1 mV to 100 mV must be amplified, filtered, digitized to an accuracy of 1 part in 10000, and stored in memory at a rate up to several hundred thousand events per second. This thesis has as its subject the final element in that chain, the Multichannel Analyzer (MCA)<sup>2</sup>.

Voltage pulses with amplitudes ranging from about 100 mV to 10 volts are the input to the MCA. Each pulse is the result of the absorption of a gamma ray by the germanium detector and has a peak value proportional to the energy of the gamma ray. The task of the MCA is to detect the peak amplitude, hold that peak for the analog to digital converter (ADC), digitize the peak value, and display a histogram of number of events per digitized value. This histogram is called the gamma ray spectrum observed by the detector.

Before 1982, the MCA was a dedicated instrument with self-contained analog and digital electronics, data display, input/output (I/O) devices, and user interface. Beginning in 1982, the development of an alternative architecture was begun. This architecture, named the Gammalyzer, uses distributed microprocessor based data acquisition front ends to collect the histogrammed data. All user interface, data display, I/O, and data manipulation are performed by a standard microcomputer or minicomputer. This architecture has become quite common in many fields, but in 1982 many doubted whether the high-performance dedicated MCA could ever be emulated by the new architecture. The success of the new approach was clearly demonstrated during the 1987 IEEE Nuclear Science Symposium, when every major manufacturer of MCA's showed a similar product, and not one new dedicated MCA was demonstrated.

The subject of this thesis is the development of the Gammalyzer. The process has taken several years and the result is due to the work of many engineers and the investment of hundreds of thousands of dollars by EG&G ORTEC. For clarity and completeness a full discussion of the theory of operation of a germanium spectroscopy system and of the Gammalyzer itself is given. Care is taken, however, to separate the contributions of this author from those of the rest of the group.



## CHAPTER II

### THE GERMANIUM GAMMA RAY SPECTROMETER

In this section the theory of the modern Germanium Gamma Ray Spectrometer is developed. No new material is derived here, but certain characteristics of the system which impact the development of the Gammalyzer are emphasized. Figure II-1 shows a block diagram of the complete system.

Gamma rays are quanta of electromagnetic energy produced when a radioactive nucleus decays. They range in energy from about ten thousand electron volts (keV) to ten MeV. The decay is governed by Poisson statistics, so the time distribution between events is described by an exponential density function. If the mean rate is  $R$ , the probability that the time between two events is between  $t$  and  $t + dt$  is given by

$$f(t) = R e^{-Rt} dt.$$

The fraction of all events with intervals less than some given time  $T$  is found by integrating the above expression from 0 to  $T$  yielding

$$F(T) = 1 - e^{-RT}.$$

This characteristic of gamma ray emission is important in the design of the shaping amplifier and the MCA. Figure II-2 shows the exponential cumulative distribution for a mean rate of 40,000 per second. Note that the average time

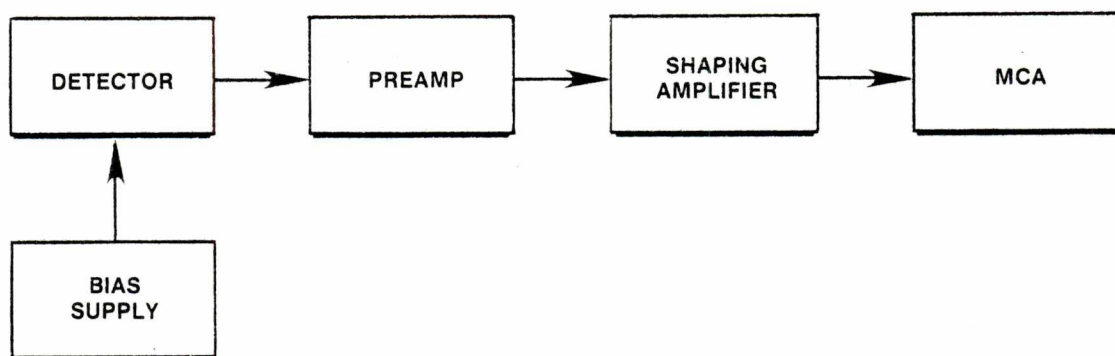


Figure II-1  
Ge Gamma Ray Spectroscopy System



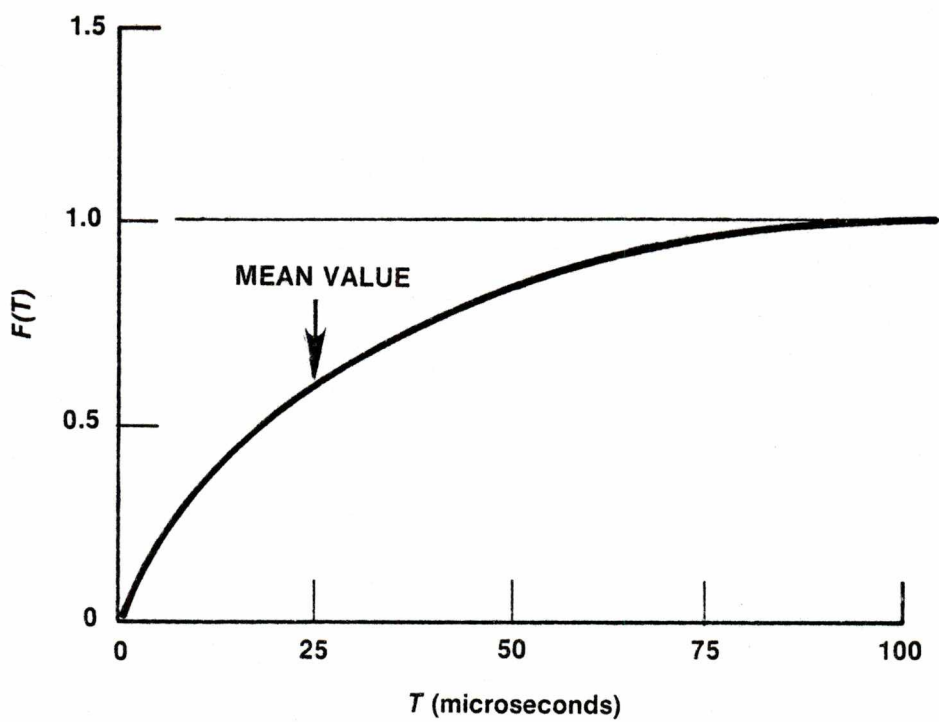


Figure II-2  
Exponential Cumulative Distribution Function

between events is 25 microseconds but that 63 percent of the events occur less than 25 microseconds after their predecessor.

In order to measure the energy of a gamma ray, a signal proportional to that energy is required. The germanium detector accomplishes this by absorbing all the energy in the depletion region of a large reverse biased germanium diode. Free electron-hole pairs are created by this process, with the number of such pairs accurately proportional to the energy absorbed. In a properly designed detector the electrons and holes are collected to the electrodes so that the collected charge provides the energy proportional signal. The collection process takes a few hundred nanoseconds, thus the detector itself could operate at rates of up to several megaHertz. It is shown later that the constraints of optimizing signal-to-noise ratio and the finite digitizing time are the rate limiting factors. As an example, a one MeV gamma ray produces about 336,000 free electron-hole pairs in germanium, so that a detector with a junction capacitance of 20 pF will produce a signal of about 2.7 mV. Figure II-3 shows the typical coaxial diode structure used to obtain large volumes with reasonable collection distances.

Figure II-4 shows schematically the charge sensitive preamplifier usually employed to process germanium detector signals. The signal current is integrated on the feedback

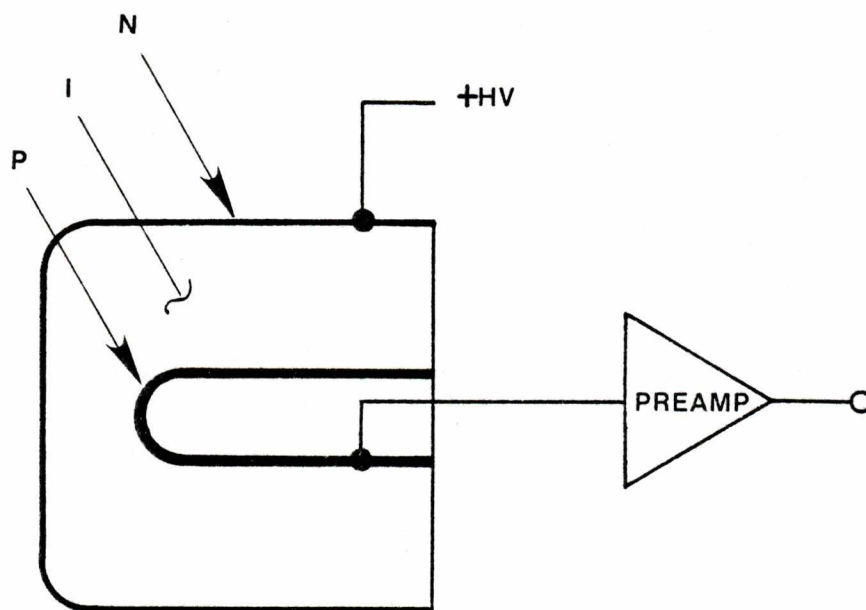


Figure II-3  
Coaxial Germanium Detector

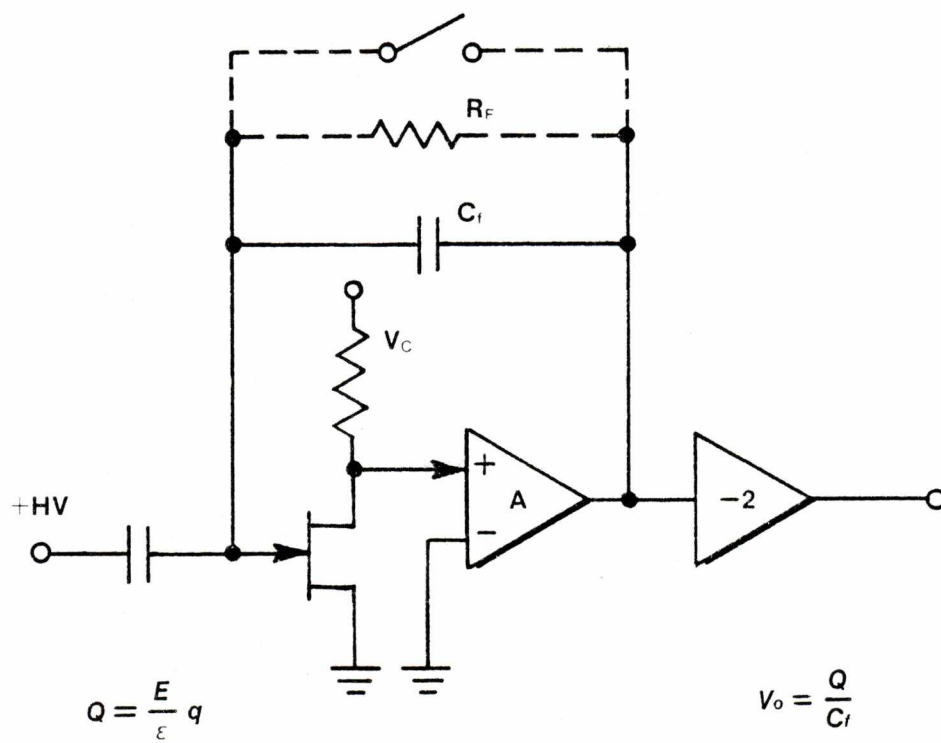


Figure II-4  
Charge Sensitive Preamplifier

capacitor, producing a voltage signal,  $V_O$ , proportional to the total charge collected and hence proportional to the gamma ray energy:

$$V_O = Q/C_f.$$

The output of the preamplifier is shown in Figure II-5. Note that the signals of interest show up as steps or jumps in the preamp output. Since the signals are all of the same polarity, some method must be provided to discharge the integrating capacitor. The usual method is to place a large resistor in parallel with the capacitor, but the use of low noise transistor switches is preferred for high counting rate systems.

In properly designed systems, the noise sources can be represented as in Figure II-6. The noise source in parallel with the detector capacitance is typically the detector reverse bias leakage current combined with the feedback resistor  $R_f$ , while the series source is typically the variation in the drain current of the input field effect transistor, referred back to the input. Because of the small bandgap of germanium, the leakage current at room temperature is many mA, thus cryogenic temperatures are required for low noise. In order to reduce the noise these components are typically operated at about 90 degrees Kelvin.

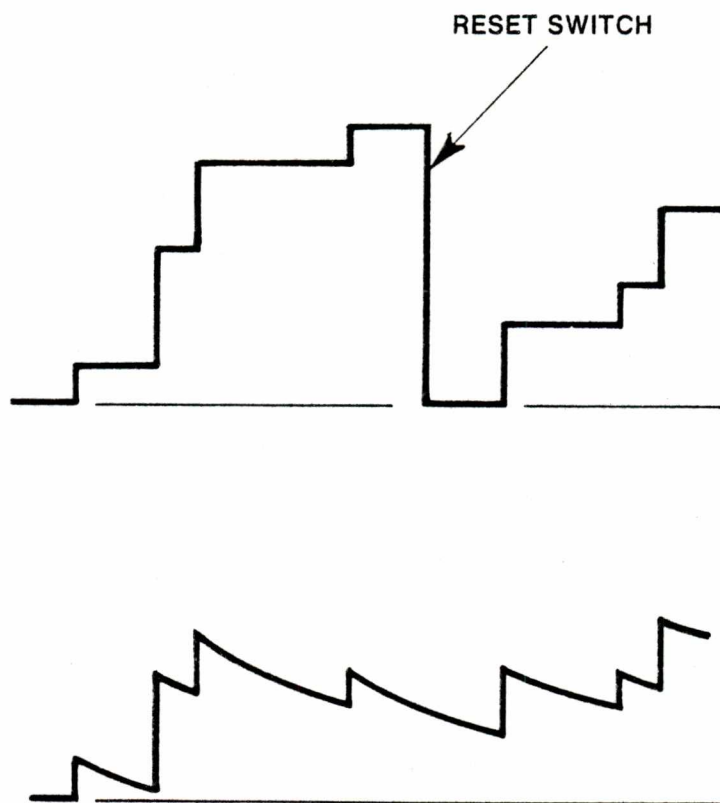


Figure II-5  
Preamplifier Output

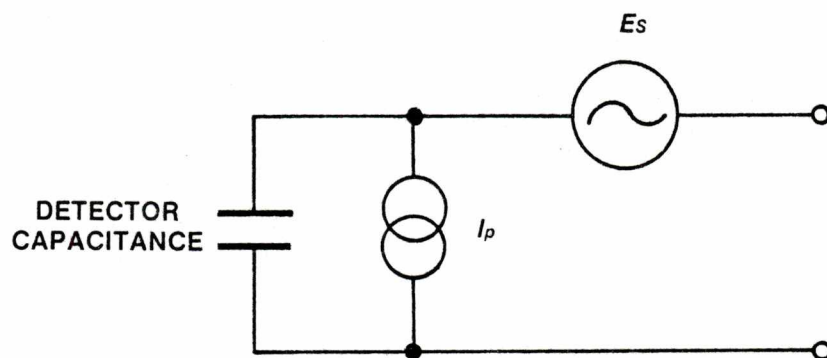


Figure II-6  
Equivalent Noise Sources



The preamplifier output is further processed by a shaping or filter amplifier. Output pulses produced by the shaping amplifier have a maximum amplitude proportional to the individual steps in the preamplifier output and are thus proportional to the energy of the gamma ray. The optimum shape of the pulse is approximately triangular, with equal rise and fall times. It can be shown that increasing the time scale of the shaping amplifier output pulse increases the parallel noise while decreasing the series noise. If no other constraints are placed on the time allowed for the shaping amplifier pulse, there is an optimum value where the two types of noise are equal. However, if a second pulse arrives before the output pulse is complete, events can be lost due to the pulses 'piling up' on each other. There thus exists a trade-off between optimizing the system for least noise and highest energy resolution and optimizing for maximum data rate. It can be shown that the output rate,  $R_O$ , of undisturbed pulses depends on the input rate,  $R_i$ , and the effective processing time,  $T_m$ , according to the following equation:

$$R_O = R_i e^{-R_i T_m}.$$

This curve is plotted in Figure II-7 for a typical system optimized for low noise and having a processing time of 25 microseconds. Note that the maximum output rate occurs when

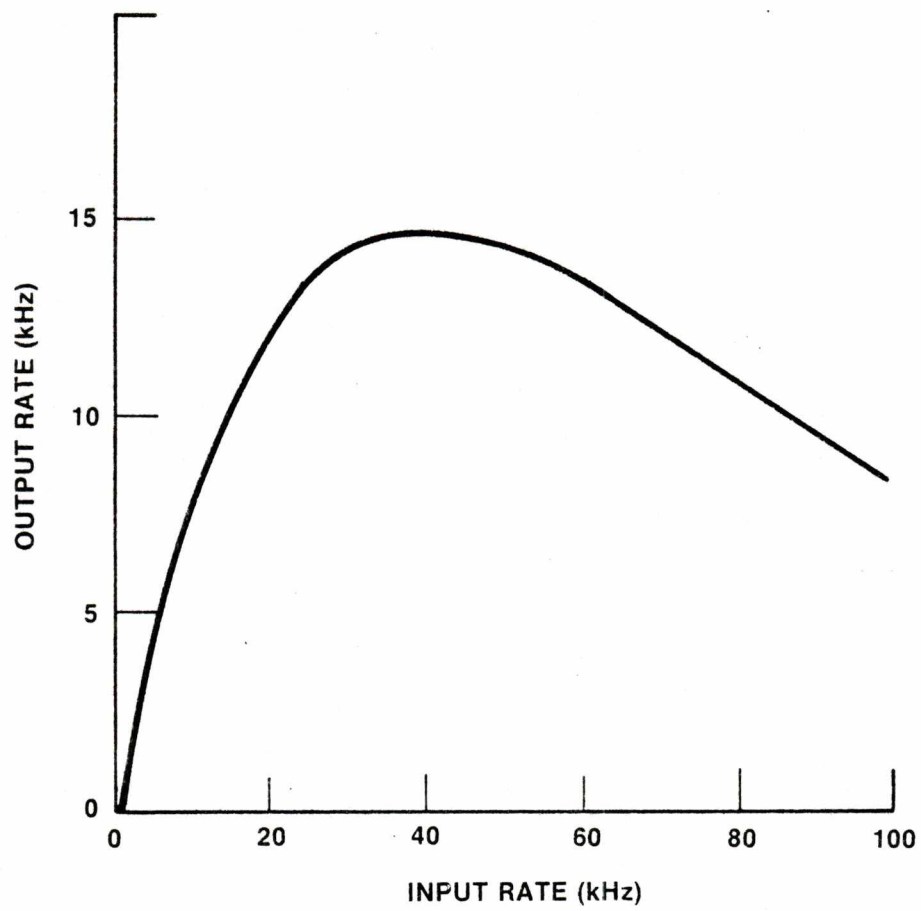


Figure II-7  
Throughput Curve

the input rate is  $1/T_m$  and is  $1/e$  times the input. Thus for this system the maximum output rate is 14,715 counts per second when the input rate is 40,000 counts per second. These rates are perhaps surprisingly low, considering the speed of the detector, and impact the design of Gammalyzer.

The maximum amplitude of the output pulse of the shaping amplifier is proportional to the gamma ray energy. Usually the desired result of the measurement is the frequency distribution of gamma ray energies emitted by the source. A ballistic measurement of the maximum amplitude of each pulse from the amplifier is performed by the multichannel pulse height analyzer (MCA), digitizing to an accuracy of typically 13 bits or 1 part in 8192. The resulting number is used to index into a memory array and increment that 'channel' of data. Since each channel corresponds to a small range of energies, the result of processing many such pulses is the desired histogram of number of events as a function of energy. Figure II-8 shows a block diagram of the classic MCA architecture.

The Linear Gate opens to process each pulse and then is held closed during the time required to digitize the pulse. Critical design characteristics are feedthrough and pedestal. Feedthrough is defined as the fraction of the input which appears at the output when the gate is in the off condition. The measurement will be disturbed unless the feedthrough is much less the digitizing accuracy of one

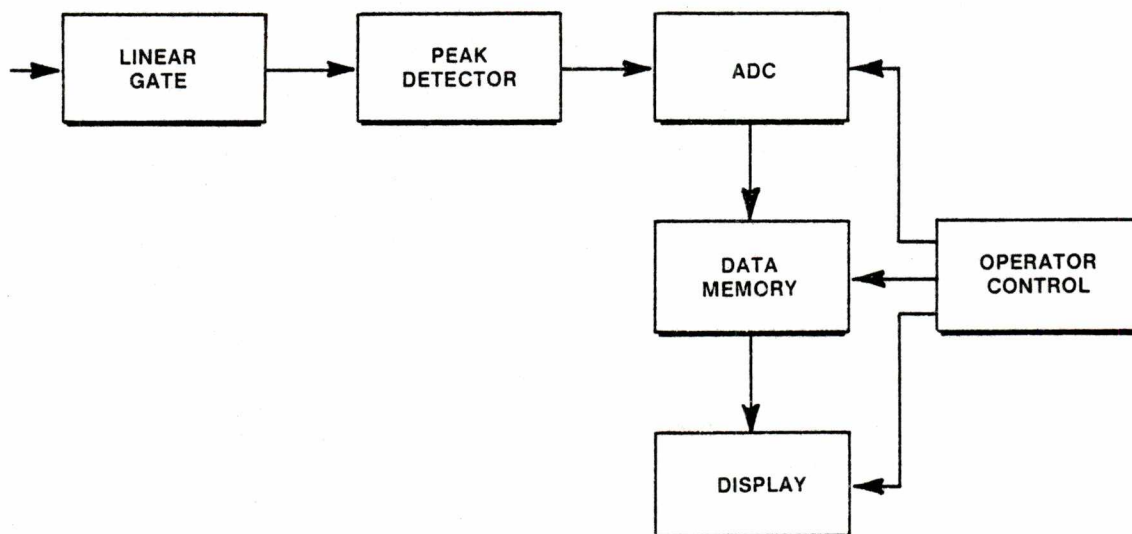


Figure II-8  
Classic Multichannel Analyzer

part in 8192. Pedestal is the DC offset of the output from the input. Pedestal is not usually a problem if it is a constant offset, but can affect the measurement if it varies with time or temperature.

The Peak Detector is a track and hold circuit triggered by the detection of the maximum amplitude (peak) of the voltage pulse. No change must occur in the peak value during the time required for digitizing. A capacitor buffered from the output by a low leakage buffer amplifier is typically used for storing the signal. The capacitor used to store the signal is especially critical, with internal leakage and dielectric absorption causing the major problems. Critical design characteristics of the Peak Detector are linearity, temperature and time stability, and droop. Droop is defined as the rate of decrease of the output with time, so is typically determined by leakage sources in parallel with the hold capacitor. The track-to-hold switching is critical for low droop design, with only high impedances such as FET gates and bipolar transistor collectors connected to the hold capacitor.

A particularly subtle design problem arises when the Peak Detector must drive an Analog to Digital Converter (ADC) of the Successive Approximation type. This type of converter has an input which changes impedance very rapidly. The output buffer amplifier must be fast enough to correct for the change in output caused by this impedance



change, typically in less than 0.5 microsec and to better than 13 bit accuracy. It is not easy to determine from published specifications whether an amplifier can meet these requirements.

The ADC in the MCA appears to be fairly straightforward. Typical requirements are 13 bit accuracy and five to ten microsecond conversion times; such specifications are routinely achieved by Successive Approximation (SA) type converters available as modular components. The width of each channel of data, called the differential non-linearity (DNL), is required to be constant to within one percent. Commercial SA type ADC's typically specify the DNL as one half of the least significant bit. Since one least significant bit corresponds to one channel, this means the commercial ADC fails to meet the DNL requirement by a factor of fifty. In order to use these relatively fast and inexpensive devices, a technique known as the Gatti slider is used.

The Gatti slider method uses a digital-to-analog converter, (DAC), to add a small analog voltage to the signal of interest. After the ADC converts the signal to its digital equivalent, the digital value of the added signal is subtracted away yielding the desired digital equivalent of the original signal. After each conversion the slider voltage is increased by the width of one channel. A total slider voltage range of typically one

percent of the total ADC range is used. The effect of this method is to replace the DNL of each channel with the average value of many adjacent channels, with excellent results for the stochastic data considered here.

The Display Unit allows the operator to view the data. A typical gamma ray spectrum is quite complex, thus high-speed graphics displays allowing for easy manipulation of vertical and horizontal ranges are required. A certain part of the data may be considered as a Region of Interest (ROI). ROI's are marked on the display as intensified or colored areas. Live display of the data during accumulation is helpful to the operator, necessitating some method of multiple access to the data memory.

The Operator Control Unit allows the operator to enter commands to the MCA. Typical commands allow starting or stopping a data acquisition, clearing old data, storing new data in mass storage, or initiating output to printers or plotters. Classical MCA's use a variety of push buttons on the front panel for this function.

The Data Memory consists of an array of memory locations with length at least equal to the maximum range of the ADC. Each location can store several million counts of data and usually a tag with ROI data. Access to the Data Memory must be allowed for the ADC, the display unit, and any output devices or mass storage units.

The accumulated histogram contains the total measured counts as a function of energy. By normalizing the histogram to the measurement time, we obtain the desired statistical distribution function. Two times are usually kept by the MCA system - the Real Time and the Live Time. Real Time is just the clock time for the acquisition. Live Time is defined to be the number which gives the most accurate data when used to normalize the accumulated histogram. The difference between Real Time and Live Time is called the Dead Time and is a measure of the time required for the Shaping Amplifier and ADC to process the pulses. Accurate estimation of the Live Time is a critical function of any MCA architecture.

The operator must be able to specify the termination of acquisition based on certain Preset conditions. Stopping after a given Preset Live Time is typical, but often there is a desire to stop after a certain number of counts has accumulated in a given channel or ROI. Monitoring the various Preset conditions is sometimes performed in the Operator Control Unit but is usually assisted by dedicated hardware.



## CHAPTER III

### A NEW APPROACH - THE GAMMALYZER

In this section an implementation of the MCA is discussed which was typical of the design approach used in 1980. The architecture and implementation of the Gammalyzer is then described and the two designs contrasted in areas of acquisition speed and cost effectiveness.

Figure III-1 shows the block diagram of the Ortec 7010 MCA. This was considered a state of the art MCA design in 1980. The complete system is a large, rack mounted instrument which sold for about ten thousand dollars. Since the 7010 had limited computational abilities and no provision for mass storage of spectral data, it was often coupled to a minicomputer. There are four, almost independent, computers embedded in the 7010. The data flow between them during a typical data acquisition is considered in the following paragraphs.

The Master Control Processor (MCP) is an 8 bit computer based on the Intel 8085 microprocessor. It provides the functions of the Operator Control Unit discussed in Chapter II and also some of the I/O functions. It is coupled to the other three computers by the Low Speed Control Bus, which is essentially the 8085 native bus. When the operator requests the start of a data acquisition, the MCP decodes the keyboard, recognizes the

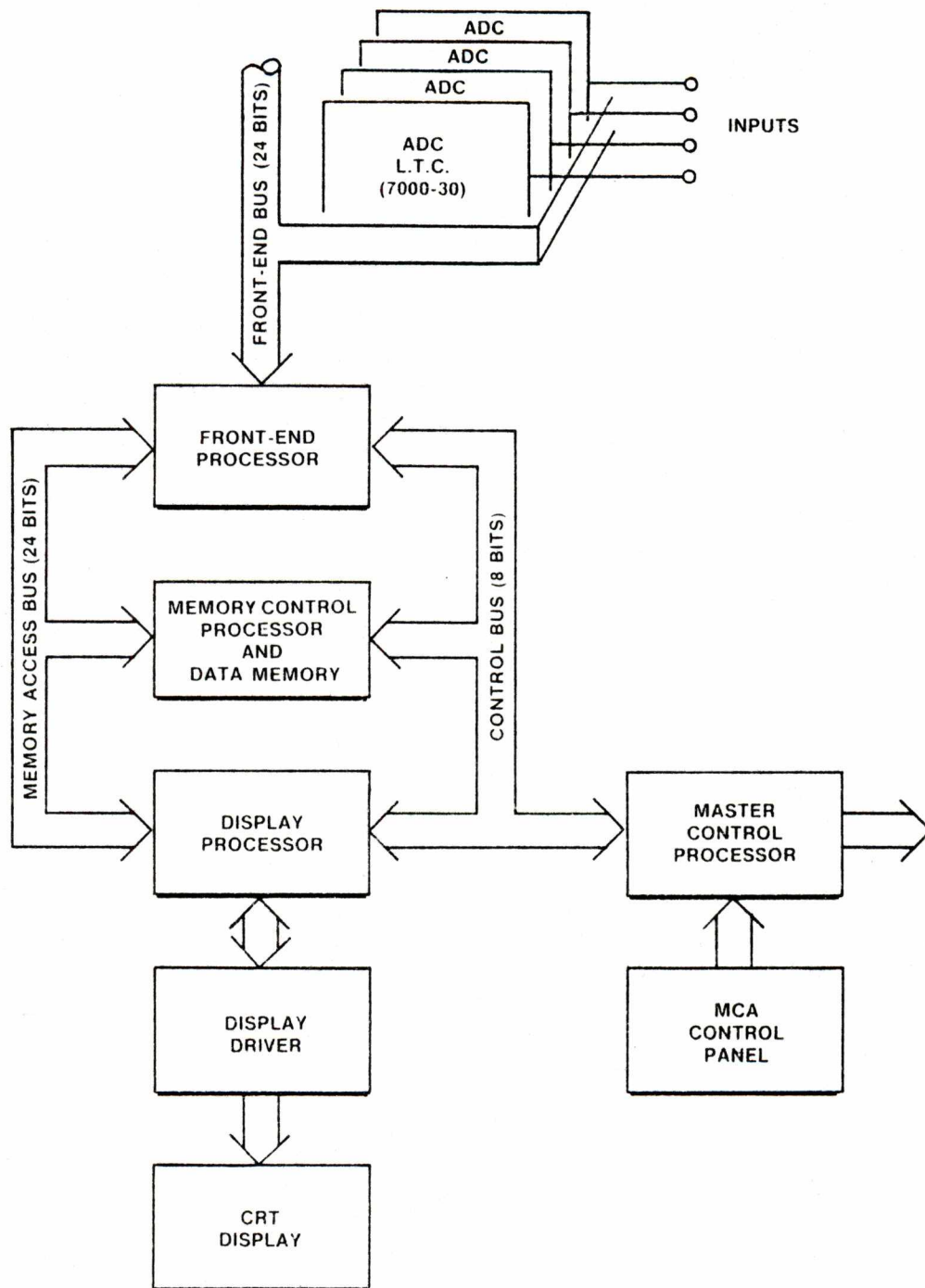


Figure III-1

7010 MCA Block Diagram

command, and sends a message to the Front End Processor to enable the acquisition of data from the ADC.

The Front End Processor (FEP) is a high-speed 16 bit special purpose computer based on micro-coded bit-slice processor design. When commanded by the MCP to allow data acquisition, it begins scanning the enabled ADC's waiting for an event. When an ADC generates an end of conversion signal, the FEP reads the data and frees the ADC for its next conversion. The FEP then signals the Memory Control Processor that it has data.

The Memory Control Processor (MCP) is also a 16 bit bit-slice design. When the FEP signals the need to update the data memory, the MCP waits until it is through with any other action, such as display update, and then signals the FEP that the memory is available. The FEP requests that the appropriate channel of data memory be incremented and the cycle is complete.

The Display Processor is a 12-bit bit-slice computer dedicated to displaying a graph of the data and textual information for the operator. It can display all or portions of the data memory with variable vertical linear or log scales. For each plotted point, it must request the contents of the appropriate channel from the MCP and calculate the scaled value to drive the analog display.

The FEP, MCP, and the Display Processor are coupled by the high-speed 24 bit Memory Access Bus and the 8-bit

Memory Instruction Bus. Since the FEP can interface to multiple input devices, the base system must be designed to handle the data throughput of the most complex configuration. The requirement for instant update of the graphics display and the resultant contention for data memory is the major reason for the complex and expensive custom programmed computers.

Personal computers provide graphics display, computational power, and operator input by means of a keyboard. These are three of the functions of the MCA, thus it seems reasonable to base an MCA on a low cost, mass produced personal computer. In 1981, there were several popular personal computers which appeared to have the characteristics needed, so a study was made to determine the requirements of a separate instrument which, when mated to a personal computer by means of an industry standard bus, could perform the function of the MCA. The new instrument was named the Gammalyzer.

Clearly the Gammalyzer would require the Linear Gate, Peak Detector and ADC functions of the classic MCA design. The Operator Control and Display functions would be in the Personal Computer but the Gammalyzer must have sufficient intelligence to accept control instructions from the Personal Computer and to send data on command. A microprocessor based design was chosen for the interface to the Personal Computer. Since 16 bit microprocessors were



still relatively new and expensive at this time, an eight bit microprocessor, the Zilog Z-80, was chosen.

The Personal Computer has memory so it could serve the Data Memory function. This design would require the Gammalyzer to communicate the ADC results to the Personal Computer each time a gamma ray is detected. Even a fast parallel bus, such as the IEEE 488 bus, would limit the count rate if this method were chosen. Even worse, so much of the Personal Computer microprocessor time would be occupied by the communication that insufficient time would remain for the operator interactions and data display. For these reasons, the Data Memory was placed in the Gammalyzer.

Two critical design issues remain. The first concerns the method used to increment the Data Memory. The second concerns the monitoring of Presets. Presets based on real time or live time could be independent of the Data Memory, but Presets based on peak counts or integrated counts in an ROI require the monitoring of the data memory or the addition of complex hardware. Adding the complex circuitry to increment the Data Memory and monitor the Presets, while allowing the Z-80 microprocessor ready access to the data so the host Personal Computer could display it, would exceed the space and cost goals of the Gammalyzer.

In Chapter II it was noted that the maximum data rates are limited by pulse pileup in the Shaping Amplifier. A

typical high-resolution system might have a maximum thruput of only about 15000 counts per second. The time available for processing each new event is thus about 67 microseconds. Could the control microprocessor in the Gammalyzer perform the function of incrementing the data memory in addition to its other tasks? A simple interrupt routine was written in Z-80 assembly language and the interrupt latency and processing times calculated for a 4 MHz clock speed. The calculations showed that the Z-80 could respond to a service request from the ADC, increment the contents of the correct channel of data memory, and return to the main program in less than 25 microseconds.

This design approach seemed to offer a highly cost effective instrument to replace the MCA, but one problem remained. As discussed in Chapter II, the emission of gamma rays is governed by Poisson statistics. Even though the Z-80 could process data at an average rate of 40000 events per second, the exponential interval distribution function predicts that 63 percent of the events would occur less than 25 microseconds after their predecessor and would thus be lost. This problem can be eliminated by placing a First In First Out (FIFO) buffer between the ADC and the Z-80. The FIFO allows the ADC to deliver its data at the uneven rate produced by the radioactive decay, while the Z-80 processes it at the rate determined by its required processing time. Using the FIFO also allows the Z-80 to



perform other short tasks without losing data. FIFO's are available as relatively low cost integrated circuits, usually 8 or 9 bits wide by 32 or 64 words deep, so two chips are required to buffer the 13 bit wide ADC data.

These considerations led to the design shown in Figure III-2. Note how simple the design is compared to classic MCA. The Z-80 microprocessor handles the ADC control functions and the management of Data Memory. All Presets are monitored by the Z-80. Timing functions are simple in hardware and periodically interrupt the Z-80 which keeps the True Time and Live Time totals. When multiple detectors are used, multiple Gammalyzers are used, so the data memory is always exactly the size needed and the data path to memory expands to keep the bandwidth per ADC constant. Data acquisition and storage are completely independent of the host computer, so it can be used for other tasks once the acquisition is started.

This design was implemented essentially as described here. The project mainly involved three engineers and took about two years to complete. This author was not involved in that portion of the development.

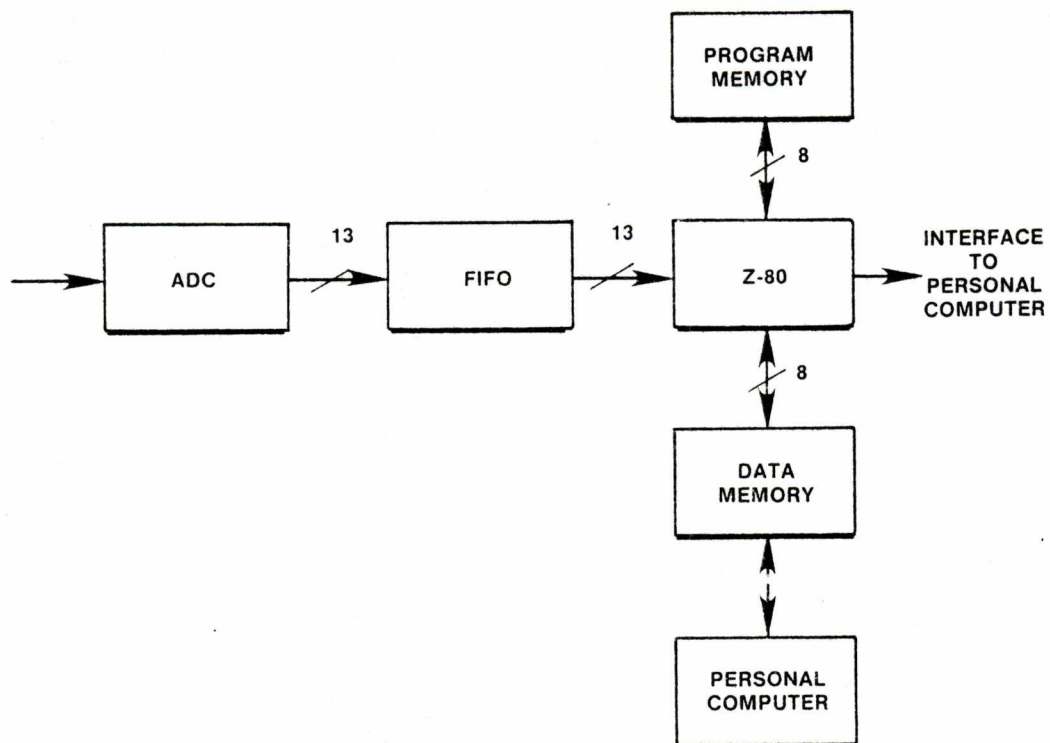


Figure III-2  
Gammalyzer Block Diagram

## CHAPTER IV

### THE DUAL PORT MEMORY - IBM PC INTERFACE

As the Gammalyzer development neared completion in early 1983 an important development occurred. The IBM Personal Computer, released in August of 1981, had begun to completely dominate the microprocessor based personal computer market. The Gammalyzer was intended to interface to a variety of personal computers, using industry standard interfaces, and both the RS-232 and IEEE 488 standard interfaces were supported. The necessary software to emulate an MCA was being written to run on a Digital Equipment Corporation (DEC) Pro 350 Personal Computer but the response time to update the graphics display was excessive. At 9600 baud, sending 8192 channels of three byte data requires nearly 30 seconds. The IEEE 488 transfer time on the 350 was limited by a poor implementation and was not much faster. It appeared that if the Gammalyzer was to replace the MCA it would have to be interfaced to the IBM PC by a very fast parallel bus and be supported by responsive, fast MCA emulation software. This project was initiated and completed by this author.

A variety of data transfer methods were considered but again analysis of the actual data flow led to a surprisingly simple solution. The data memory in the Gammalyzer is simply a part of the random read/write memory

of the Z-80 microprocessor. Approximately 25 microseconds is required for the Z-80 to increment the contents of a three byte channel of data, but only one byte read and one byte write, a total of 2 microseconds, actually involve the data memory. This results because the incrementing of the second byte occurs only when the low byte is incremented from 255 to 0 plus a carry. The carry is then added to the second byte. When the second byte generates a carry, it is added to the third byte. This occurs only once in 65,536 events. At maximum data rates the Z-80 thus only requires access to the memory about 10 percent of the time.

The graphics display should be updated about ten times per second to appear as fast as the hard-wired MCA. This means the IBM PC must read all 8192 channels every 100 milliseconds. Each channel is three bytes, requiring about 800 nanoseconds per byte, thus the entire data memory can be scanned in about 21 milliseconds. Reading the entire data memory every 100 milliseconds thus only requires about 20 percent of the available memory bandwidth.

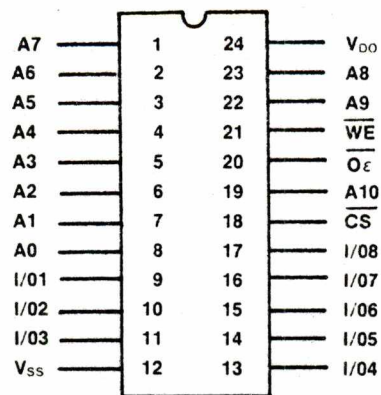
Since the Gammalyzer and the PC are both microprocessors with the capability of extending their memory access cycles by the use of wait states, it should be possible to design a very simple arbitration circuit to allow each microprocessor to consider the data memory as a part of its own read/write memory. Since the accesses to memory occur with small duty cycles, collisions are rare



and performance affected very little. When collisions do occur, the first microprocessor to access the shared memory completes a normal memory cycle, while the other has its memory cycle extended until the memory is free. Such shared data memories are sometimes called dual-ported memories. True dual-port memories are implemented in the memory chip itself and allow essentially simultaneous access to each memory location. Such devices are far too expensive for this application. The shared data memory is implemented with standard, low cost silicon static memory.

Silicon memory is available as either dynamic or static Random Access Memory (RAM). Dynamic RAM is lower in cost but more difficult to interface because of its requirement for periodic refreshing. In order to simplify the shared memory, static RAM was selected.

The first prototype of Gammalyzer used the 6116, the highest density chip then available. Figure IV-1 shows the pin-out and timing diagrams of the 6116. The device has 2048 memory locations each 8 bits wide, thus 12 chips are required for the 8K by 24 bit data memory in the Gammalyzer. Each channel of data fills 4 bytes of the Gammalyzer address space, thus allowing a simple shifting operation to map the ADC output to the correct channel of data. Since only three bytes of data are actually needed, every fourth byte is not physically present. Three of the 6116's thus provide 2048 channels of data memory mapped



TOP VIEW

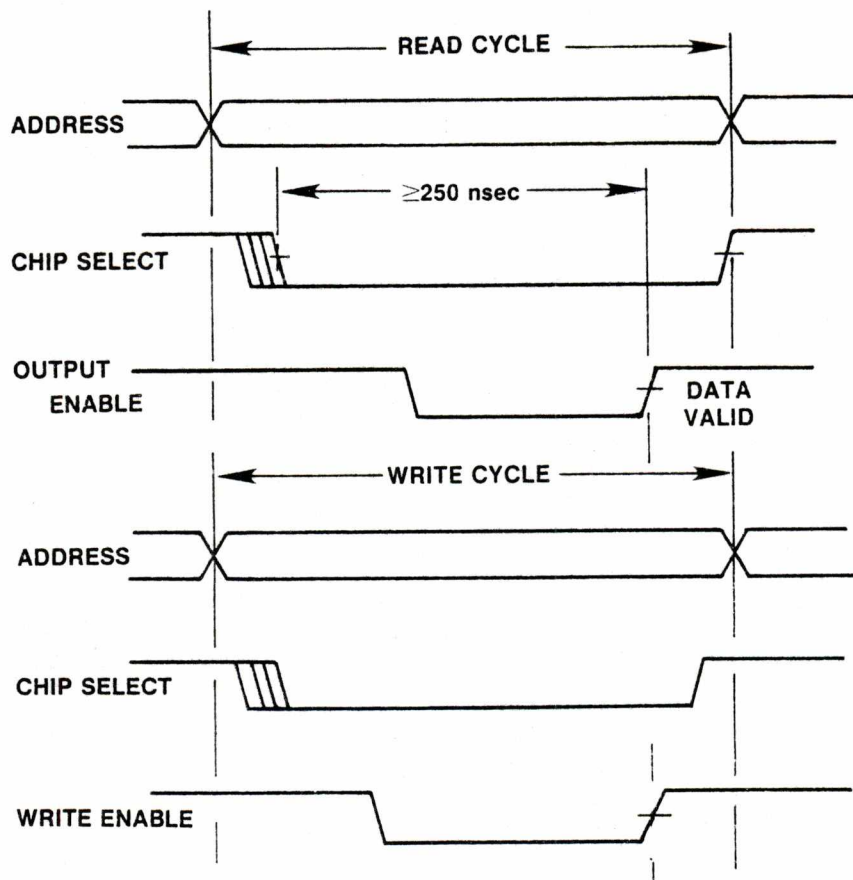


Figure IV-1

6116 Memory Device



into 8K bytes of the microprocessor's memory space. The entire 8192 channels map into 32K bytes of microprocessor memory.

Reading data from the 6116 requires the following steps :

1. Place a stable address on the address lines,
2. Assert the negative true Chip Select (-CS) line,
3. Assert the negative true Output Enable (-OE) line,
4. Wait the required Access Time before reading the data lines.

Writing data to the memory is similar :

1. Place a stable address on the address lines,
2. Assert the -CS line,
3. Assert the Write Enable (-WE) line,
4. Place stable data on the data lines,
5. Wait the required Setup Time before -WE goes high, thus strobing in the new data.

These are exactly the steps used by the Gammalyzer Z-80 and the IBM PC 8088.

Figure IV-2 shows the Z-80 microprocessor and its memory cycles. With suitable buffering the address and data busses can directly couple to the 6116 memory. The -RD signal, conditioned with -MREQ, has the required timing for the 6116 -OE signal. Similarly, -WR can serve for the 6116

-WE signal. The -WAIT line causes the memory cycle to insert wait states. Wait states leave the address lines and data lines stable and extend the -RD or -WR strobes to allow for slow responding memories. This feature allows the shared memory to work. When the Z-80 tries to access the shared memory while the 8088 has a cycle in progress, an arbitration circuit asserts -WAIT until the 8088 has completed its cycle.

The IBM PC bus, shown in Figure IV-3, is essentially a demultiplexed and buffered standard 8088 bus. The address and data lines are suitable for the 6116 memories. The -MEMW and -MEMR signals have the correct timing for the 6116 -WE and -OE signals. The I/O CH READY performs the same function as the Z-80 -WAIT line.

Figure IV-4 shows a block diagram of the entire shared memory. Signals whose name starts with an 'I' refer to the Intel 8088 bus while those that begin with a 'Z' refer to the Zilog Z-80 bus. The Data Memory consists of the twelve 6116 memory chips and a 74LS154 4 to 16 decoder. The decoder generates the individual -CS signals for the 6116's. The Address Bus, Data Bus, and Control Bus are driven by tri-state drivers from either the Gammalyzer's Z-80 or the IBM PC's 8088. Selection of the appropriate tri-state drivers is done by the Arbitrator. The Arbitrator also generates -IWAIT or -ZWAIT if simultaneous accesses to

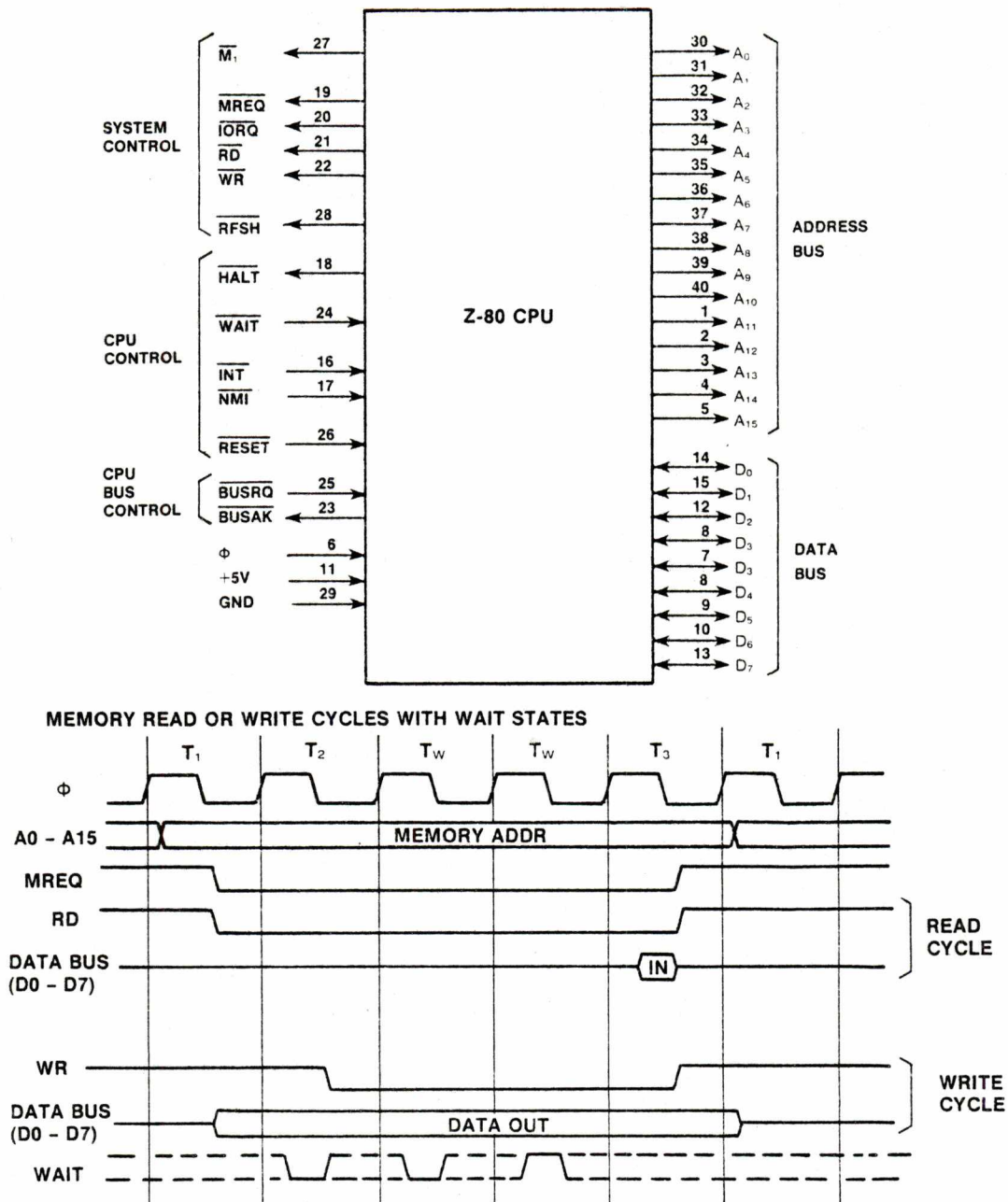


Figure IV-2  
Z-80 Microprocessor

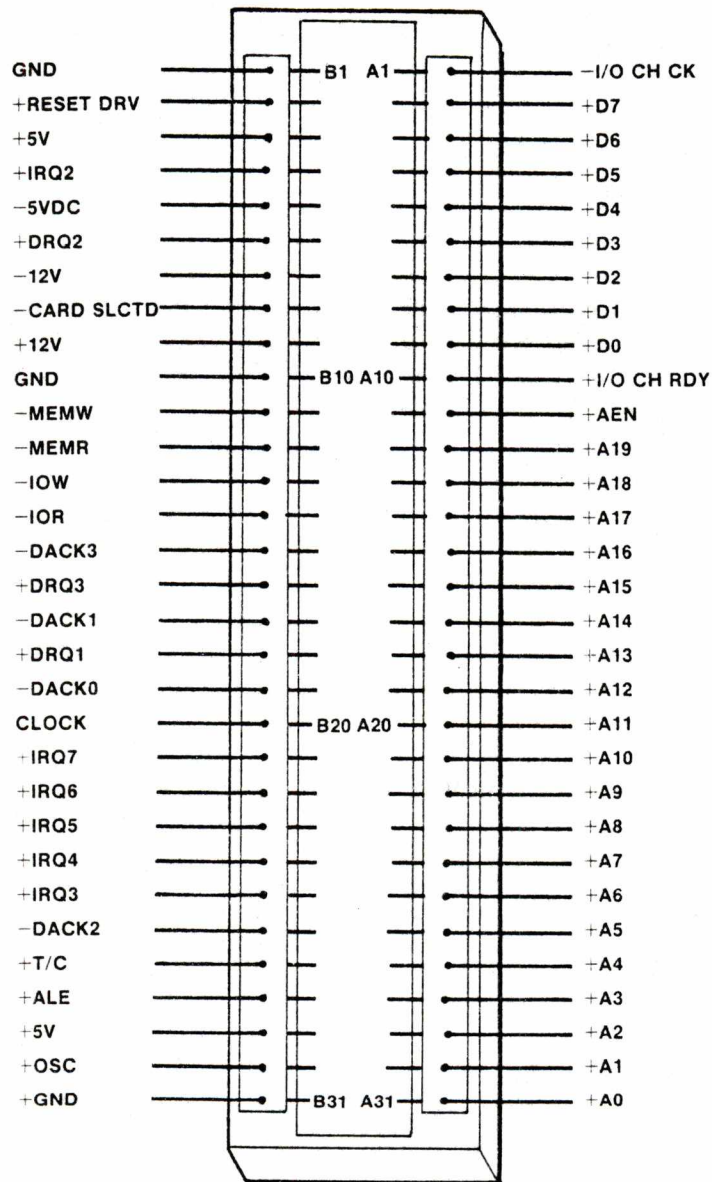


Figure IV-3  
IBM PC Bus Diagram

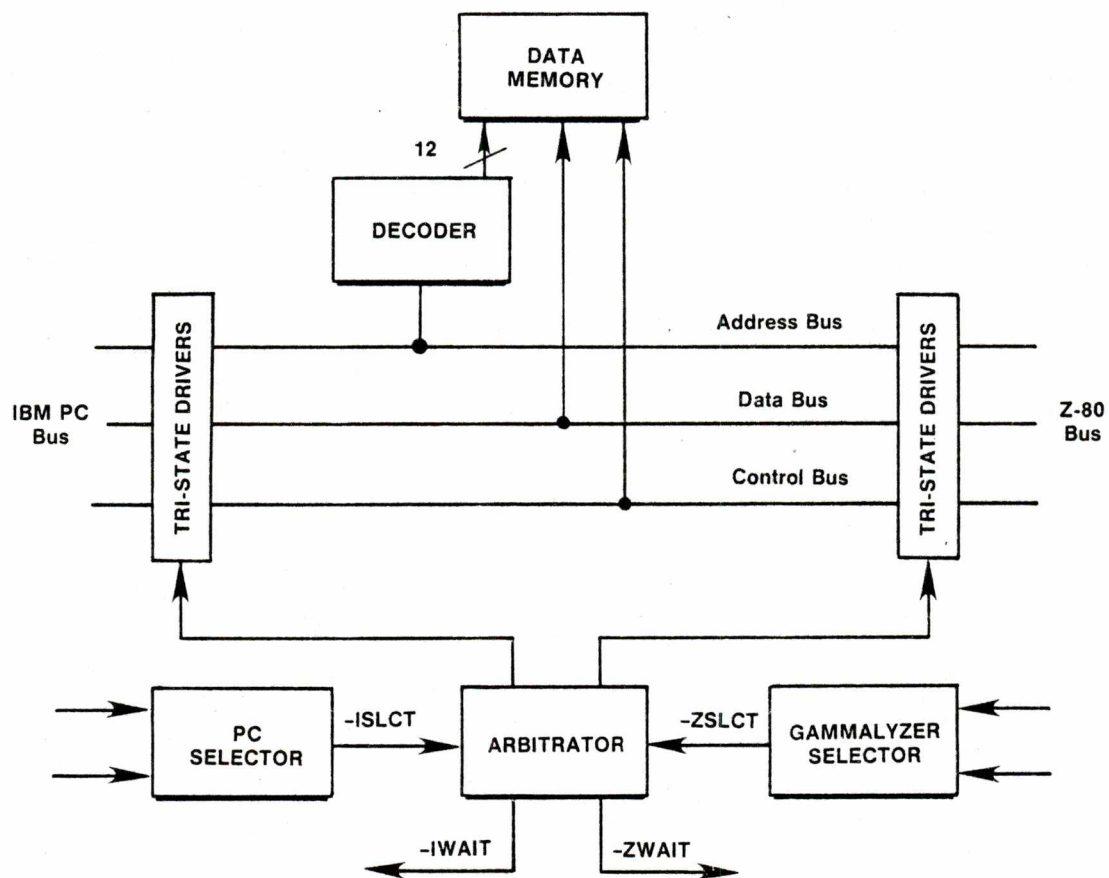


Figure IV-4  
Shared Memory Block Diagram



the shared memory requires either of the microprocessors to wait. The -ISLCT signal is generated when the PC attempts a valid read or write of the shared memory. A similar circuit generates -ZSLCT when the Gammalyzer accesses the shared memory.

Consider the sequence of events that occurs when the PC reads a byte of data from the shared memory. The PC Selector circuit confirms that a valid access is being attempted and generates -ISLCT. Assuming the memory is available, the Arbitrator generates -IBUSY. The appropriate 15 address lines of the PC bus are gated onto the Data Memory Address Bus by enabling the 74LS245 drivers. A -CS signal for one of the 12 6116 memory chips is generated by the 74LS154 decoder . Enabling the 74LS245 drivers connects the Data Memory Data Bus to the PC data bus. The PC -MEMR signal reverses the direction of the Data Bus drivers and is gated onto the Data Memory Control Bus where it connects to the -OE line of the selected 6116. The PC -MEMR signal goes high, indicating the completion of the read. When the PC begins its next cycle, the PC Selector sees a different address and removes -ISLCT. The Arbitrator then removes -IBUSY and the cycle is complete.

Now suppose the Gammalyzer attempts to access memory during the cycle just described. Gammalyzer's Selector sees a valid data memory access attempt and generates -ZSLCT. Memory is in use, so the Arbitrator sees -IBUSY asserted



and generates -ZWAIT, thus causing the Z-80 memory cycle to be extended by wait states until -ISLCT is removed. The Arbitrator then removes -IWAIT, generates -ZBUSY, and the memory cycle completes as before.

The Arbitrator, shown in Figure IV-5, consists of two 14 pin IC's costing a total of less than 50 cents. The first two NAND gates of the 74LS00 quad NAND gate act as inverters. The other two NAND gates are connected as a set-reset flip-flop. When both -ISLCT and -ZSLCT are false (high) the inputs to the second set of NAND gates are both low and -IBUSY and -ZBUSY are false. If -ISLCT is asserted while -ZSLCT is false -IBUSY goes true. Should -ZSLCT be asserted while -IBUSY is true, the flip-flop ignores it, but the negative logic AND gate generates the required wait signal. If both processors attempt to access data memory at exactly the same time the Arbitrator sometimes oscillates a few times but always stably selects one or the other.

Figure IV-6 shows the IBM PC Selector circuit. The PC bus AEN line is high when the DMA circuitry is active, typically during memory refresh cycles for the computer's dynamic RAM. The 74LS138 address decoder is enabled when AEN is low and address line A19 is high, thus selecting a valid memory address somewhere in the top half of the 8088's 1 Mbyte address space. Address lines A16, A17 and A18 are decoded to select one of the eight 64 Kbyte pages

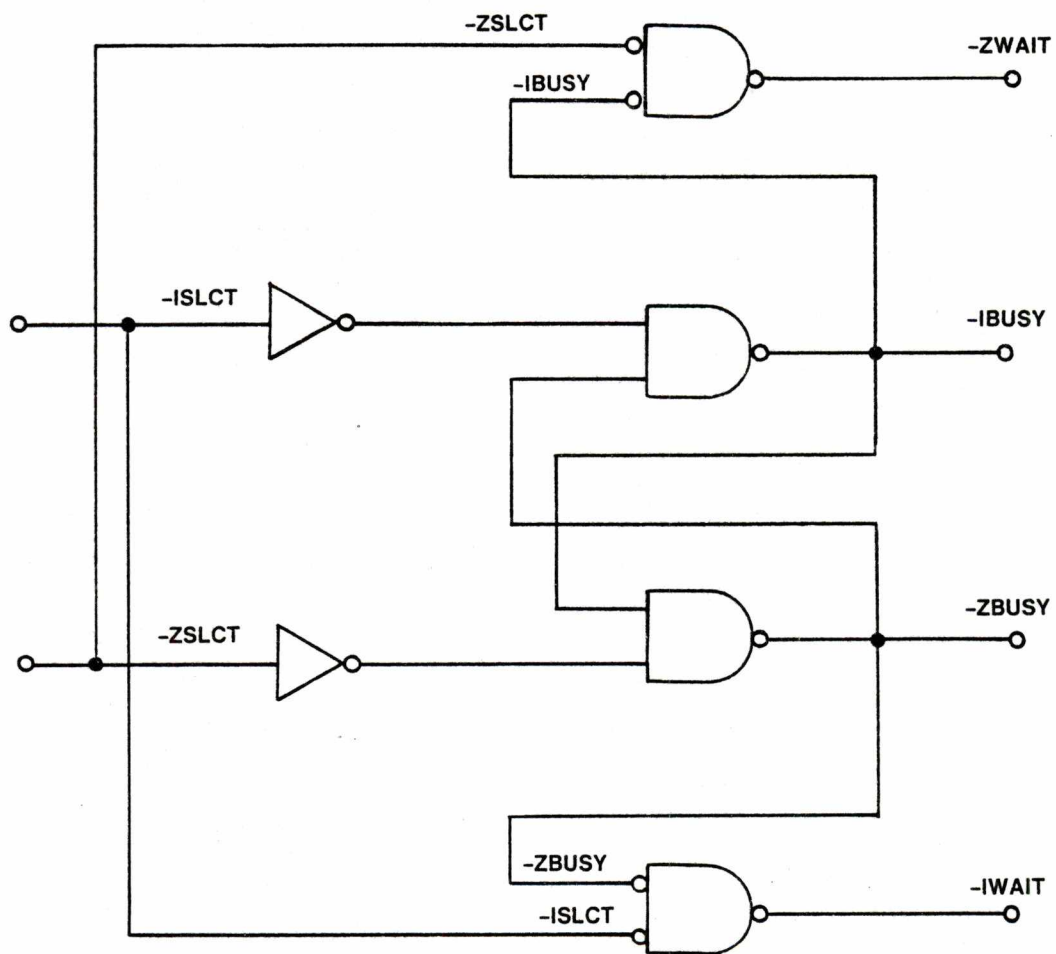


Figure IV-5  
Arbitrator

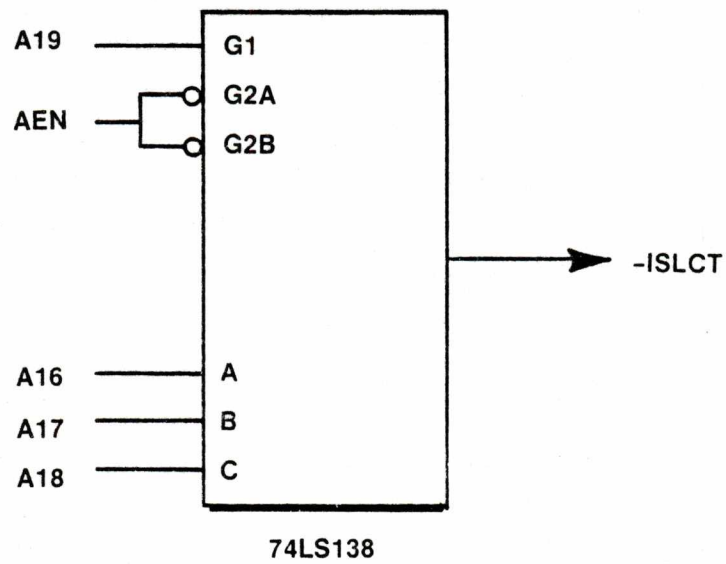


Figure IV-6  
PC Selector Circuit

of memory. The PC memory map indicates page D is not normally used, so this output from the decoder is used for -ISLCT. Since the PC address line A15 is not decoded, the 32 Kbyte data memory is mapped into both halves of page D.

The Gammalyzer Selector circuit is shown in Figure IV-7. The Z-80 indicates a valid memory access by asserting -MEMR. Any memory request to the top half of the Z-80 address space indicates a request for the shared memory. The circuit shown generates -ZSLCT with the correct timing for driving the bus transceivers. Note that if -ZSLCT terminated with -MEMR, the 74LS245 bus transceivers might tri-state before the read or write was completed.

The circuit described was constructed using a wire-wrapped prototyping board for the IBM bus and a small hand wired circuit board for the Gammalyzer bus. The memory physically resided in the PC, since more space was available there, but the production version places the data memory in the Gammalyzer itself, thus yielding a completely standalone data acquisition device. The photographs in Figure IV-8 show the completed circuits.

The circuits were tested and shown to work as predicted. Even at the highest count rates, with many collisions accessing the memory, there are no errors. The Gammalyzer increments the shared memory at essentially its maximum calculated rate even when the PC is scanning the memory as quickly as it can update the graphics display.

The circuit was implemented in production and hundreds are now in use around the world.



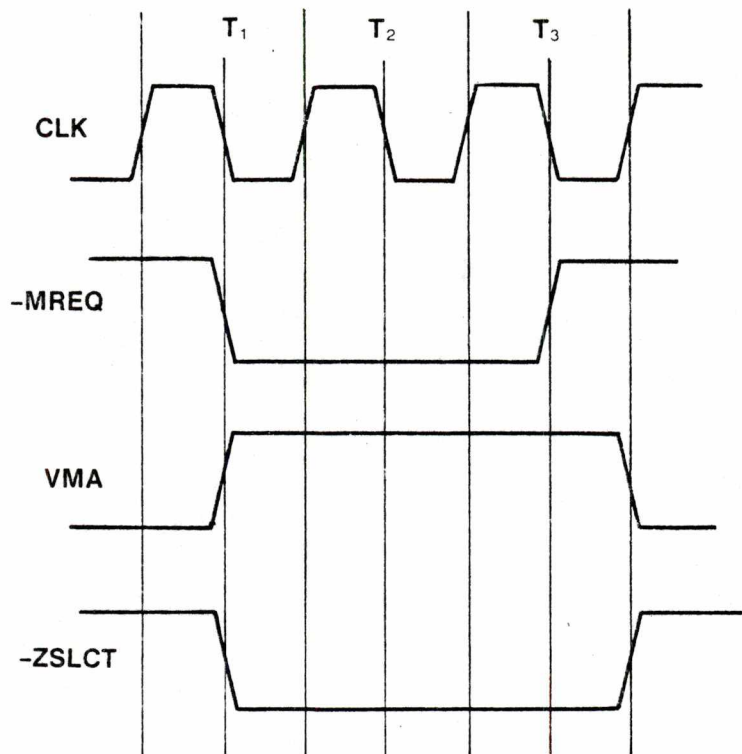
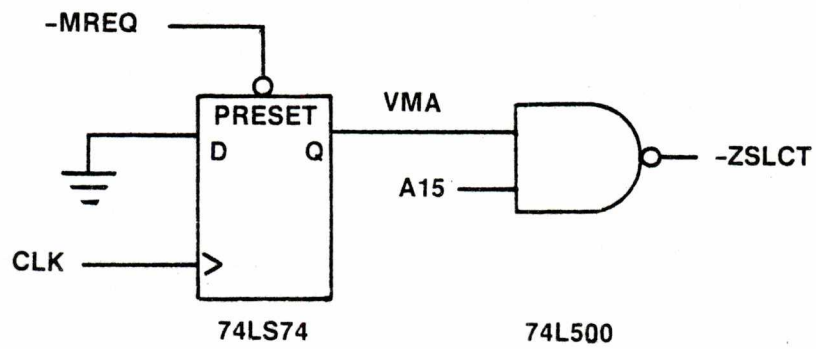


Figure IV-7  
Gammalyzer Selector Circuit

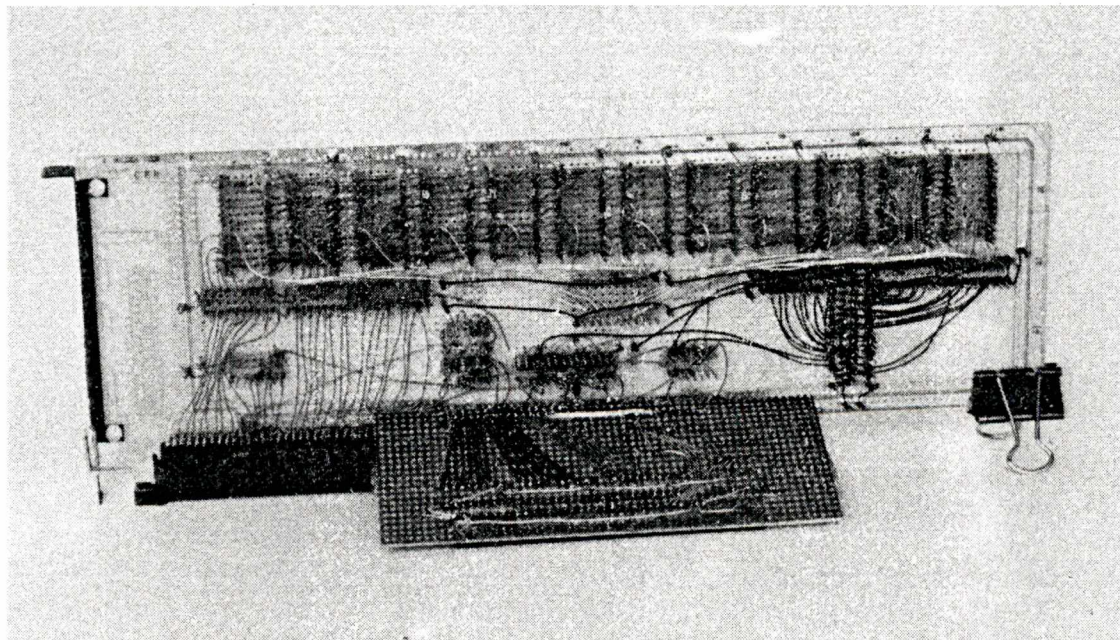
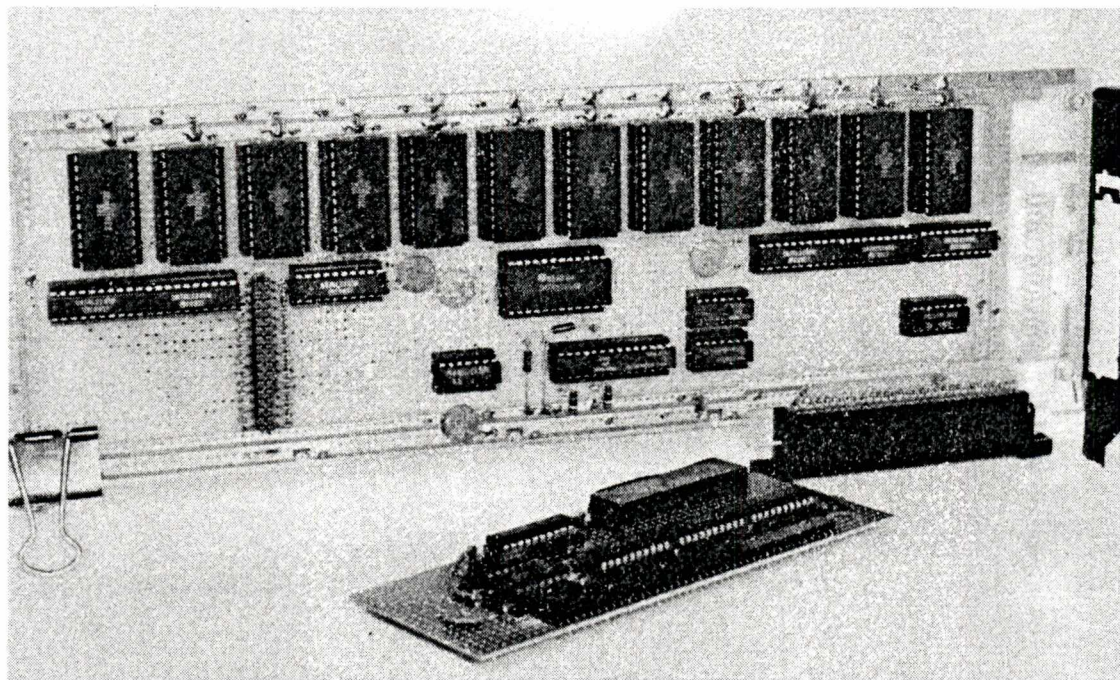


Figure IV-8  
Completed Shared Memory



## CHAPTER V

### THE SOFTWARE

The classic MCA is an easy to use instrument. Dedicated push-button controls start and stop the data acquisition, clear the data, control input/output, and manipulate the graphics display. Graphics control is particularly important because the gamma ray spectrum has many peaks and the maxima vary over several orders of magnitude. Horizontal scale can be the full 8192 channels or be expanded to display smaller regions in higher detail. Vertical ranges can be from less than 100 full scale to several million or can be logarithmic. Changing horizontal or vertical ranges is easily accomplished and the display update is essentially instantaneous. A movable cursor or marker selects a particular channel whose contents are displayed in text.

The intent of the Gammalyzer project was to produce an instrument which would replace the classic MCA. All important characteristics of the MCA, particularly the ease of use and excellent graphics, would be emulated by the software of the Personal Computer. Since the Emulator program would serve as a base for future enhancements, careful design and implementation using the most current software engineering practices was required. The first step in the project was to study the current state of software

engineering and select the tools and procedures to be used in writing the Emulator.

During the 1960's, computer software was more witchcraft than engineering. Programmers rarely had professional training and every programmer had his own set of methods for designing programs. The languages available did little to promote good design practices and, in fact, there was not much agreement as to what did constitute good design. In the late 60's and early 70's, pioneering work by Dijkstra<sup>3</sup>, Hoare<sup>4</sup>, and Wirth<sup>5</sup> led to the basis of modern software engineering. Study of these pioneering works and later software engineering texts such as those by Yourdon<sup>6</sup> and by Kernighan and Plauger<sup>7</sup> indicated several important techniques which would be used in the design of the Emulator program. These techniques are Top-down Design, Modular Design, and Structured Programming.

Top-down Design is a method of breaking up a conceptually complex task into a set of logically simpler tasks that the programmer can more easily understand. The first step is to describe, without too much detail, the steps the program needs to do to perform its task. This description is usually done in simple English, called pseudocode, instead of a programming language, so that syntactic details don't interfere with the thought processes. Each task thus generated is further broken down into finer detail, until the tasks described are simple

enough to be translated into the selected programming language. Wirth termed this process Stepwise Refinement.

The individual tasks to be performed by the program are called modules. The program is easier to understand and modify if these modules are small, logically consistent units. Modular Programming is a set of guidelines that help in the process of breaking up the work of the program into the individual modules. Usually each module can be thought of as a transformer of data; there are defined inputs and outputs with a definite algorithm for transforming the input into the output. If the separation of the program into modules is well done, each module can be understood on its own, with little knowledge of the rest of the program.

Structured Programming involves primarily the flow of control in the program. Programs written in Fortran or BASIC often use the GOTO construct to transfer control to some other part of the program. Trying to understand the effect of some particular line of code in the program, with no clear way of knowing how that line was reached and what was done before it, can be almost impossible. A flow chart showing the transfer of control can look like a bowl of spaghetti; the term 'spaghetti code' is sometimes disparagingly used to describe such programs. Structured Programming uses only simple control mechanisms such as the sequence, the loop, and the if-then-else to achieve a clear, understandable flow of control. The case statement

is an extension of the if-then-else to allow multi-way selection. Figure V-1 shows these mechanisms.

The IBM PC is a microcomputer based on the 8088 microprocessor. It uses a single user, single tasking Disk Operating System (DOS) that allows easy access to the basic hardware. The Read Only Memory (ROM) Basic Input/Output System (BIOS) provides even more hardware specific interaction with the display, keyboard, RS-232 port, and other hardware resources. Finally, if even the BIOS support is inadequate, the programmer can control the hardware directly. Typically, using DOS to perform a given task is the easiest but slowest method. Using BIOS requires somewhat more hardware knowledge but is faster. Going directly to the hardware is more difficult, more likely to require change when the computer is upgraded, but fastest of all.

The only graphics system available in 1983 was the Color Graphics Adapter (CGA). The CGA is a plug-in card for the PC that drives the Color Display. It has a variety of operating modes, including a 640x200 high resolution mode suitable for the MCA display. Each pixel in the display is controlled by a single bit in display memory. The display memory is a part of the normal memory space of the 8088.



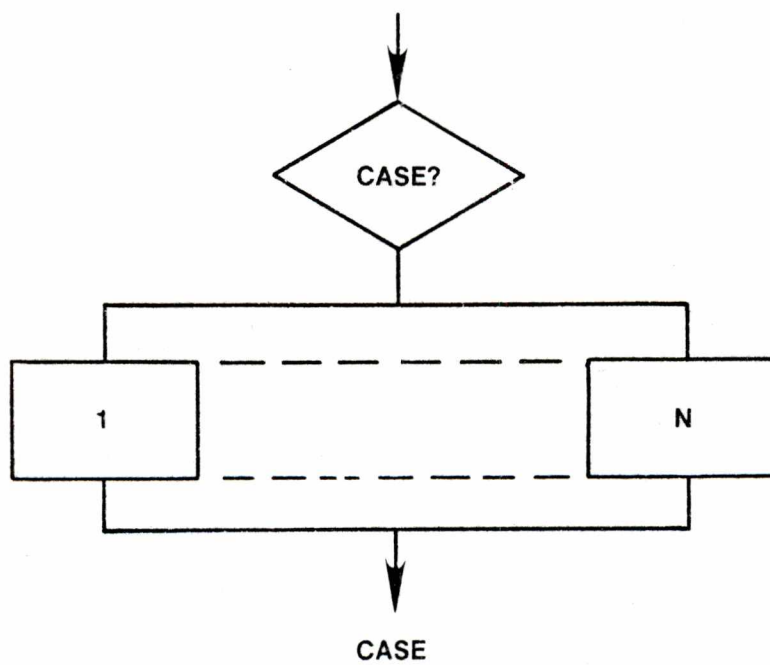
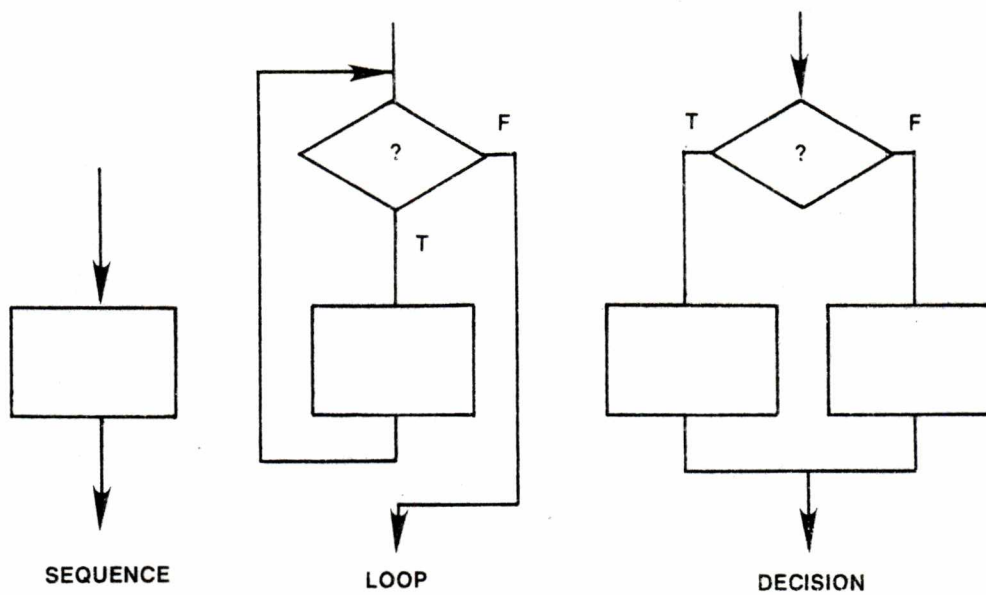


Figure V-1  
Structured Control

A Motorola 6845 generates the necessary signals to display the bit-mapped image on the Color Display. Preliminary experiments showed that using BIOS calls to generate the Emulator's MCA graphics was unacceptably slow. The graphics portion of the Emulator would require working directly with the display memory in Assembly Language.

A well-structured high level language was required in order to write a maintainable, understandable, and extensible Emulator. Direct access to the hardware of the CGA required working in the lower level and unstructured Assembly Language. Microsoft Pascal allows certain extensions to ANSI standard Pascal that make it much more suitable for programs that need close control of the hardware. The most important extension allows the declaration of a procedure as external to the Pascal environment. Such procedures can be passed data just as any normal Pascal procedure, but can be written in Assembly Language. The output units from the Pascal compiler and the Assembler are linked together by the Linker program to form an executable program. The Emulator was written in Pascal wherever possible. A collection of small, clearly defined, independent functions in Assembly Language was used where adequate speed or hardware control could not be obtained in Pascal.

Figure V-2 shows the Emulator's display screen. Ten blocks on the left side of the display correspond to the

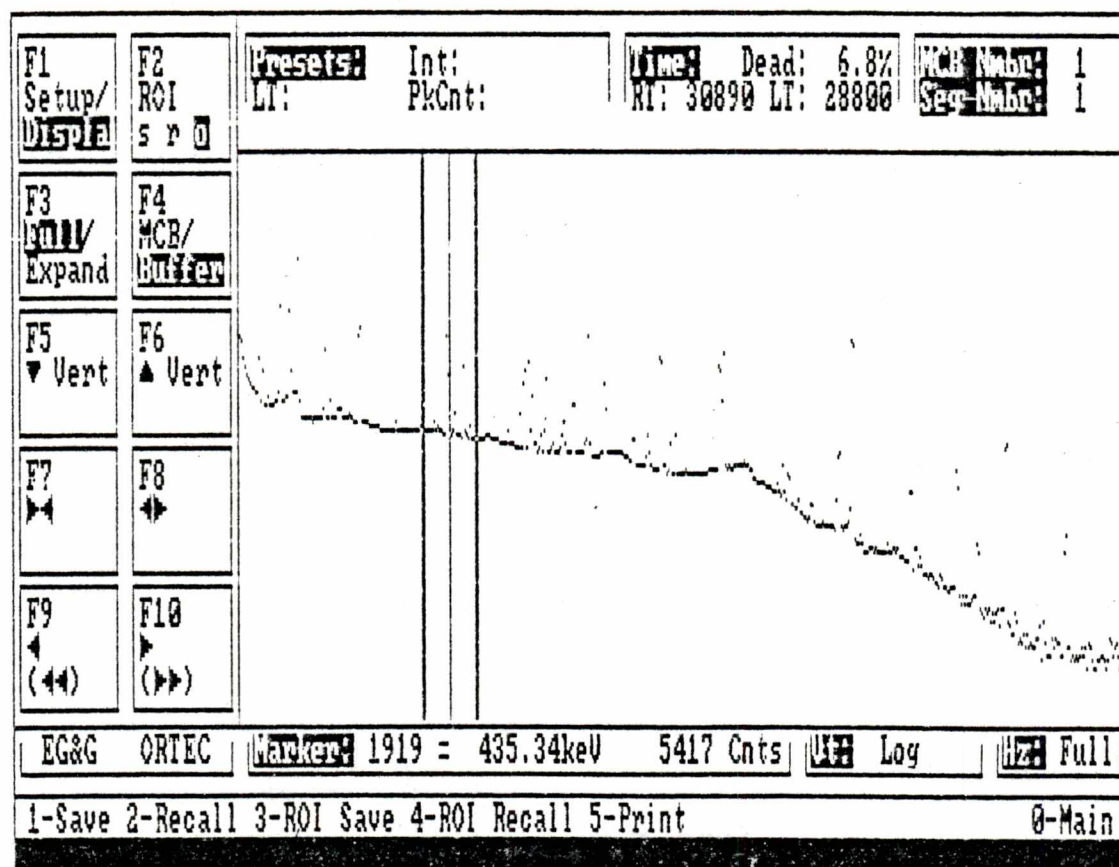


Figure V-2  
MCA Emulator Display Screen

function keys on the PC keyboard. Near the bottom of the display is a menu line showing the commands to be performed when the user presses the Alt key in combination with the number of the command. The last line of the display, in reverse video, displays prompts for data or error messages from the Emulator and also echoes the user's input when the Emulator asks for data. A graphics area shows the display of data in the currently selected format. User defined presets, the real and live time and the marker channel information are also shown.

A unique feature of the Emulator is the use of a separate data area called the Buffer. Data can be transferred from the shared memory to a private section of the Emulator's data space very quickly. All input/output and calculation functions are performed on the Buffer. This feature allows the user to view one set of data while a new set is being accumulated.

The CGA screen has only 640 columns so 512 columns are dedicated to the data display. Since there are 8192 channels of data, each pixel must display the contents of 16 channels in order to show the entire spectrum. The Emulator has two distinct display modes. In the Full Mode, the channel with the maximum number of counts in its group of 16 is displayed. This design prevents the loss of visible peaks which would occur if the average value were displayed instead. The Full Mode also displays a variable



width 'window' which is moved over the display with the function keys. The window is centered around the marker channel. When the Expand Mode display is selected the data in the window is displayed with one pixel per channel. In Expand Mode moving the marker channel causes the entire display to scroll behind the centered marker. The display is redrawn very quickly when the marker is moved or the display mode is changed, making it easy for the operator to select individual regions within the data for closer examination. Figure V-3 shows the two display modes.

The screen design and user interface were prototyped before the real program design began. This approach seems to violate the concept of Top-Down Design but such preliminary prototyping and even some Bottom-Up coding can prevent wasted effort on a conceptually flawed design. Once the basic approach is proven, proper design techniques can be applied. If properly modularized, the low level routines written in the prototyping stage may still be useful. This was the case with the Emulator program; the Assembler routines used to prototype the graphics design were incorporated in the finished program.

Figure V-4 shows the pseudocode for the Emulator. After an initializing step, the program is just an infinite loop that checks for a keypress from the operator and performs the requested command. If no key has been pressed, the default action is to update the graphics display.



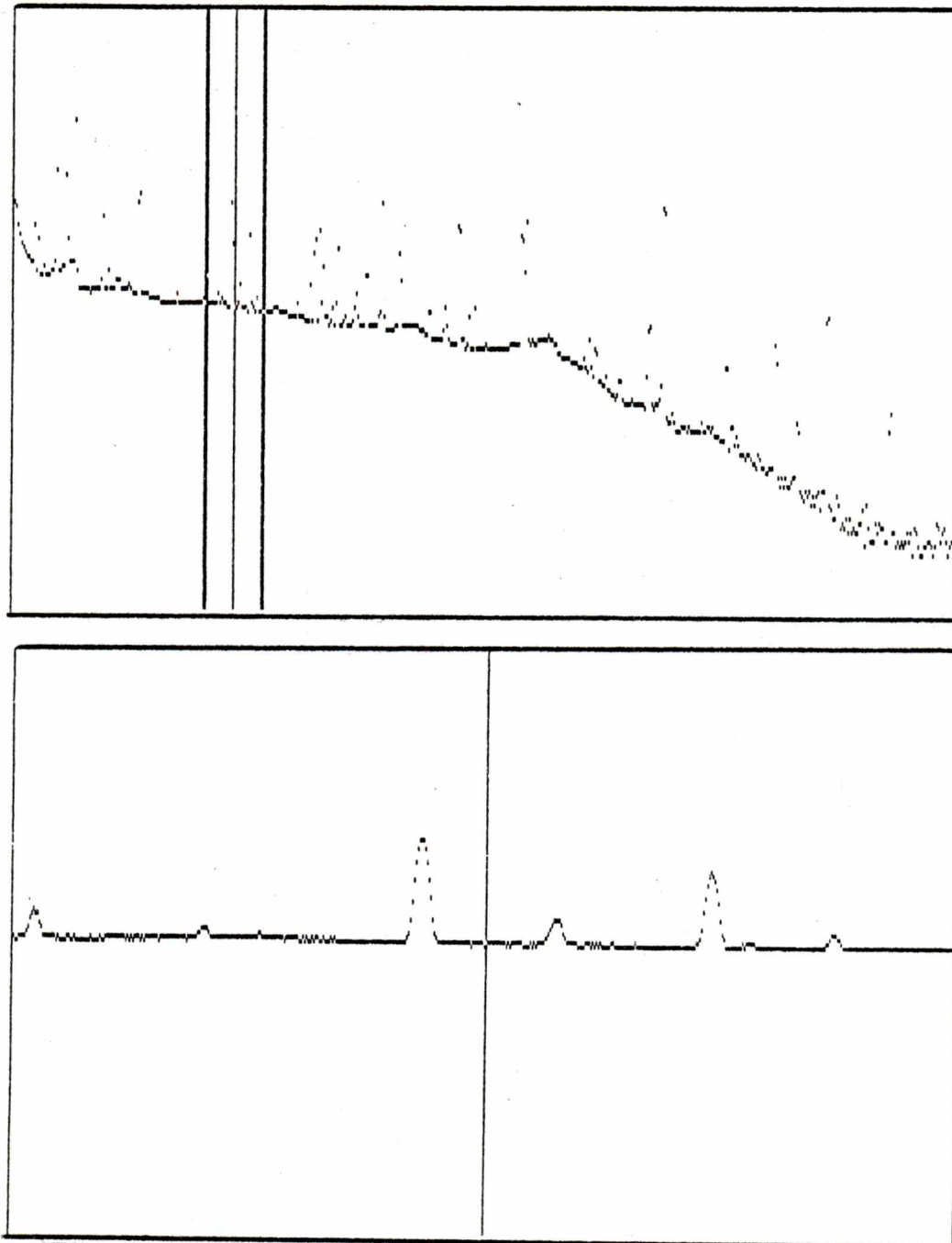


Figure V-3  
Full and Expanded Display

```
begin
  initialize;
  display_screen;
  repeat
    check_keypress;
    case keypress of
      none: update_display
        1: start;
        2: stop;
        3: clear;
        10: break;
      otherwise:
        prompt ('Not valid key');
    until forever:
      clean_up_and_exit:
  end.
```

Figure V-4  
Structured English Pseudocode

The high priority thus placed on the display update causes the accumulating data to appear to 'grow' just like the MCA. When a valid keypress is recognized, a separate module (called a procedure in Pascal) is called to perform the requested command. If a given command has not been coded yet, an empty or do-nothing procedure can be used as a 'place holder'. This technique allows testing the parts of the program as they are completed.

Since the PC has nearly infinite data storage by means of replaceable magnetic media (floppy disks), the Emulator allows spectra to be saved and recalled from disk. Each of these tasks is handled by a separate Pascal procedure and together they form a good example of the software engineering techniques used. The buffer has associated with it the real time and live time of the acquisition. The data in the buffer is thus the input for the Save procedure and the output for the Recall procedure. The disk files produced by Save are the inputs for Recall. Encapsulating the knowledge of the actual format of the data on the disk in these two routines means that if a later decision is made to include more information in the file, for example the starting date and time, only these routines have to be modified.

The Emulator program was written using the methods described. The complete listing consists of over two hundred pages and can not be duplicated here. Two sample

procedures, used to save and recall spectra, appear in the Appendix. It is not possible to completely understand these procedures, out of the context of the entire program, but much of the readability of Pascal is demonstrated.

The program has been extensively enhanced over the four years since its first release. It now supports a variety of display types, hardware configurations and even a different type of data acquisition known as multi-channel scaling. The software engineering methods used have resulted in a program which is easy to maintain and modify.

## CHAPTER VI

### FUTURE WORK

Gammalyzer has been a successful product, but the basic architecture is now over five years old. IBM has a new bus, the Micro Channel Architecture, which may become important. Newer germanium detector systems and shaping amplifiers will soon be able to process data faster than Gammalyzer's Z-80 can handle. Many low cost microprocessors are available which are more powerful than the Z-80. Work is now in progress to design a next generation Gammalyzer which will use an advanced 16-bit microprocessor.

Gammalyzer II will have more data memory and more than twice the speed of its predecessor. Interfaces will be available for both the standard PC bus and the new bus. Enhanced firmware will be developed in the 'C' programming language, instead of assembler, for more efficient development and maintenance.

As the newer Intel family processors, especially the 80386, become more popular, Microsoft's new operating system OS/2 will gradually replace the current Disk Operating System (DOS). OS/2 is a complex multi-tasking system which does not allow the easy access to the hardware allowed by DOS. New graphics oriented user interfaces use pull-down menus, often controlled by a mouse, to accept



commands. New graphics systems allow much higher resolution and more color than the CGA standard assumed by the existing Emulation software. An entirely new version of the Emulation software will be required to achieve a modern up-to-date look.

LIST OF REFERENCES

## LIST OF REFERENCES

1. D. V. Freck and J. Wakefield, *Nature* 193, 669 (1962).
2. D. Wilkinson, *Proc. Cambridge Phil. Soc.* 46, 508 (1950).
3. E. W. Dijkstra, *Communications of the ACM*, 341 (May 1968).
4. O. J. Dahl, E. W. Dijkstra, and C. A. R Hoare, Structured Programming, Academic Press, (1972).
5. N. Wirth, Systematic Programming: An Introduction, Prentice-Hall, (1973).
6. E. Yourdon, Techniques of Program Structure and Design, Prentice-Hall, (1975).
7. B. W. Kernighan and P. J. Plauger, Software Tools in Pascal, Addison-Wesley, (1981).

## APPENDIX

## APPENDIX

### SAMPLE PROGRAMMING

```

{
  Prompt for a filename,
  save the spectrum,
  update the display.
}

procedure save_spectrum;

  const
    blk_size = 512;
    max_index = blk_size - 1;

  type
    buffer = array[0..max_index] of integer4;
    header_rec = record
      type_marker : integer;
      mcb_number : integer;
      segment_number : integer;
      pass_count : integer;
      true_time : integer4;
      live_time : integer4;
      start_date : string(8);
      start_time : string(4);
      start_channel : integer;
      length : integer;
    end;

  var
    handle1, handle2, nread, nwrote, ercode : word;
    index1, index2 : integer;
    hdr_name : lstring(20);
    chn_header : header_rec;
    buf_array : ^buffer;
    max : integer;

  procedure make_hdr_name;

    const
      space = chr(32);

    var
      tmp_str : lstring(2);
      index : integer;

    begin
      hdr_name := 'mcb';
      eval(encode(tmp_str, buf_mcb:2));

```



```

concat(hdr_name,tmp_str);
eval(encode(tmp_str,buf_seg:2));
concat(hdr_name,tmp_str);
concat(hdr_name,'.hdr');

for index := 1 to upper(hdr_name) do begin
  if(hdr_name[index] = space) then begin
    hdr_name[index] := '0';
  end;{if}
end{for}
end;{procedure make_hdr_name}
procedure make_chn_header;

var
  date_string : string(8);
  time_string : string(8);
  date_record  : string(8);
  time_record  : string(4);
  month_string : string(3);
  temp_string  : string(2);

begin
  date(date_string);
  temp_string[1] := date_string[1];
  temp_string[2] := date_string[2];
  date_record[1] := date_string[4];
  date_record[2] := date_string[5];
  date_record[6] := date_string[7];
  date_record[7] := date_string[8];
  if (temp_string = '01') then
    month_string := 'JAN'
  else if (temp_string = '02') then
    month_string := 'FEB'
  else if (temp_string = '03') then
    month_string := 'MAR'
  else if (temp_string = '04') then
    month_string := 'APR'
  else if (temp_string = '05') then
    month_string := 'MAY'
  else if (temp_string = '06') then
    month_string := 'JUN'
  else if (temp_string = '07') then
    month_string := 'JUL'
  else if (temp_string = '08') then
    month_string := 'AUG'
  else if (temp_string = '09') then
    month_string := 'SEP'
  else if (temp_string = '10') then
    month_string := 'OCT'
  else if (temp_string = '11') then
    month_string := 'NOV'

```

```

else if (temp_string = '12') then
    month_string := 'DEC';

date_record[3] := month_string[1];
date_record[4] := month_string[2];
date_record[5] := month_string[3];
date_record[8] := ' ';

time(time_string);
time_record[1] := time_string[1];
time_record[2] := time_string[2];
time_record[3] := time_string[4];
time_record[4] := time_string[5];

with chn_header do begin
    type_marker := -1;
    mcb_number := buf_mcb;
    segment_number := buf_seg;
    pass_count := 0;
    true_time := buffer_true_time;
    live_time := buffer_live_time;
    start_date := date_record;
    start_time := time_record;
    start_channel := lo_limit;
    length := total_channels div buf_num_segs;
end;{with}
end;{procedure make_chn_header}

begin
    get_name(chn_name,'chn');
    createff(chn_name,0,handle1,rcode);
    if(rcode <> 0) then begin
        prompt('Error in the filename, aborting SAVE');
        return
    end;{if}
    new(buf_array);

    make_hdr_name;
    openff(hdr_name,0,handle2,rcode);
    if(rcode = 0) then begin
        readff(handle2,ads chn_header,32,nread,rcode);
        if (chn_header.type_marker = -3) then begin
            with chn_header do begin
                type_marker := -1;
                true_time := buffer_true_time;
                live_time := buffer_live_time;
            end;{with}
        end;{if}
    else begin
        prompt('Incorrect header file, storing with
            current date and time');
    end;
end;

```

```

        make_chn_header;
    end;{else}
    closeff(handle2,rcode);
end{if(rcode...)}
else begin
    prompt('No header file found, storing with current
           date and time');
    make_chn_header;
end;{else}

writeff(handle1,ads_chn_header,32,nwrote,rcode);

index1 := chn_header.start_channel;
max := index1 + (chn_header.length);
repeat
    for index2 := 0 to max_index do begin
        buf_array^[index2] := (spectrum1^[index1+index2]
                               and 16#7ffffff)
    end;{for}
    writeff(handle1,ads_buf_array^[0],
            (4*blk_size), nwrote, rcode);
    index1 := index1 + blk_size;
until ((index1 >= max) or (rcode > 0));
if (rcode <> 0) then begin
    prompt('Error in writing file, aborting SAVE');
    closeff(handle1,rcode);
    dispose(buf_array);
    return
end;{if}
closeff(handle1,rcode);
dispose(buf_array);
white_line
end;{save_spectrum}

```

```

{
  Prompt for a filename,
  get the spectrum,
  update the display
}

procedure recall_spectrum;

type
  header_rec = record
    type_marker : integer;
    mcb_number : integer;
    segment_number : integer;
    pass_count : integer;
    true_time : integer4;
    live_time : integer4;
    start_date : string(8);
    start_time : string(4);
    start_channel : integer;
    length : integer;
  end;

var
  handle, ercode, read_num : word;
  file_type_marker : integer;
  old_buf_num_segs : integer;
  chn_header : header_rec;
  buf_len : word;
  itemp_time : integer4;
  index : integer;

procedure put_chn_header(var chn_header : header_rec);

begin
  with chn_header do begin
    buf_mcb := mcb_number;
    buf_num_segs := total_channels div length;
    buf_seg := segment_number;
    buffer_true_time := true_time;
    buffer_live_time := live_time;
    lo_limit := start_channel;
    hi_limit := start_channel + length - 1;
  end; {with}
  rtime[buf_seg] := buffer_true_time;
  ltime[buf_seg] := buffer_live_time;
  buf_int[buf_seg] := 0;
  buf_slope[buf_seg] := 1;
end; {procedure put_chn_header}

```



```

begin
  get_name(chn_name,'chn');
  openff(chn_name,0,handle,rcode);
  if(rcode <> 0) then begin
    prompt('Error in the filename, aborting RECALL');
    return
  end;if}
  readff(handle,ads chn_header,32,read_num,rcode);
  if (chn_header.type_marker <> -1) then begin
    prompt('File is not a spectrum file, aborting');
    closeff(handle,rcode);
    return
  end;if}
  old_buf_num_segs := buf_num_segs;
  put_chn_header(chn_header);
  if (chn_header.length = 8192) then begin
    buf_len := 32767;
  end;if}
  else begin
    buf_len := wrd(chn_header.length * 4);
  end;if}
  readff(handle,
    ads spectrum1^[chn_header.start_channel],
    buf_len, read_num,rcode);
  if (rcode <> 0) then begin
    prompt('Error in reading file, aborting RECALL');
    closeff(handle,rcode);
    return
  end;if}
  closeff(handle,rcode);
{Clear the rest of the buffer if num_segs not compatible}
  if (old_buf_num_segs <> buf_num_segs) then begin
    for index := 0 to lo_limit-1 do
      spectrum1^[index] := 0;
    for index := hi_limit+1 to 8191 do
      spectrum1^[index] := 0;
    buf_seg := 1;
    start_channel := 0;
    mark_channel := span div 2;
    lo_limit := 0;
    hi_limit := total_channels div buf_num_segs - 1;
    clear_plot;
    if page then begin
      lo_max := lo_limit;
      hi_max := hi_limit;
      lin_plot(plot_seg,plot_offset,start_channel,
        f_s_b,span,vert_offset);
      marker(cursor_line);
    end;if}
    else begin
      lo_max := 0;
      hi_max := total_channels-1;
    end;if}

```



```

        log_plot(plot_seg,plot_offset);
        highlight(start_channel,span);
    end{else}
end{if}
else if ((mark_channel < lo_limit) or
        (mark_channel > hi_limit)) then begin
    buf_seg := 1;
    start_channel := 0;
    mark_channel := span div 2;
    lo_limit := 0;
    hi_limit := total_channels div buf_num_segs - 1;
    clear_plot;
    if page then begin
        lo_max := lo_limit;
        hi_max := hi_limit;
        lin_plot(plot_seg,plot_offset,start_channel,
                f_s_b,span,vert_offset);
        marker(cursor_line);
    end{if}
    else begin
        lo_max := 0;
        hi_max := total_channels-1;
        log_plot(plot_seg,plot_offset);
        highlight(start_channel,span);
    end;{else}
end;{else if}

```

```

intercept := 0;
slope := 1;
calib := false;
pk_chn_1 := -1;
pk_chn_2 := -1;
buffer_true_time := rtime[buf_seg];
buffer_live_time := ltime[buf_seg];
set_cursor(2,47);
itemp_time := buffer_true_time div 50;
write(itemp_time:6);
set_cursor(2,57);
itemp_time := buffer_live_time div 50;
write(itemp_time:6);
set_cursor(1,75);
writeln(buf_mcb:2);
set_cursor(2,75);
writeln(buf_seg:2);
white_line
end;{recall_spectrum}

```

## VITA

Rex C. Trammell was born in Sevier County, Tennessee, on November 9, 1938. He attended various elementary schools in Tennessee and graduated from Clinton High School, Clinton, Tennessee, in 1955.

Since 1962, he has been employed by EG&G Ortec, Oak Ridge, Tennessee. While working there, he attended the University of Tennessee, where he received a Bachelor of Science degree in Engineering Physics in 1965, a Master of Science degree in Physics in 1969, and a Master of Science degree in Electrical Engineering in 1988.