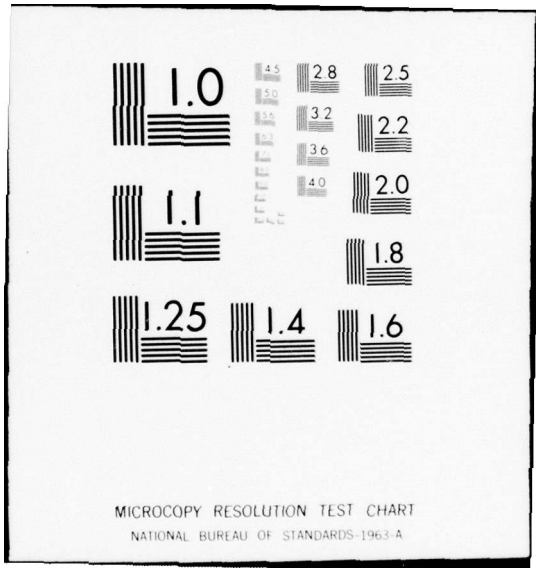
AD-A051 247 BATTELLE COLUMBUS LABS DURHAM N C F/G 9/2
MICROPROCESSING TECHNIQUES FOR ADVANCED SENSORS APPLICATIONS.(U)
SEP 77 L B OWEN, J B CLARY DAAG29-76-D-0100
UNCLASSIFIED DRDMI-T-78-25 NL

1 OF 1
AD A051247

END
DATE FILMED
4-78
DDC


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

(12)

B.S.

TECHNICAL REPORT T-78-25

MICROPROCESSING TECHNIQUES FOR ADVANCED SENSORS APPLICATIONS

L. B. Owen
Advanced Sensors Directorate
Technology Laboratory

and

James B. Clary
Research Triangle Institute
Research Triangle Park
South Carolina 27709

September 1977

U.S. ARMY MISSILE RESEARCH AND DEVELOPMENT COMMAND

Redstone Arsenal, Alabama 35809

DISPOSITION INSTRUCTIONS

DESTROY THIS REPORT WHEN IT IS NO LONGER NEEDED. DO NOT RETURN IT TO THE ORIGINATOR.

DISCLAIMER

THE FINDINGS IN THIS REPORT ARE NOT TO BE CONSTRUED AS AN OFFICIAL DEPARTMENT OF THE ARMY POSITION UNLESS SO DESIGNATED BY OTHER AUTHORIZED DOCUMENTS.

TRADE NAMES

USE OF TRADE NAMES OR MANUFACTURERS IN THIS REPORT DOES NOT CONSTITUTE AN OFFICIAL INDORSEMENT OR APPROVAL OF THE USE OF SUCH COMMERCIAL HARDWARE OR SOFTWARE.

18 DRDMI-T

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE	READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER: 78-25	2. GOVT ACCESSION NO.
3. RECIPIENT'S CATALOG NUMBER	
4. TITLE (and Subtitle): MICROPROCESSING TECHNIQUES FOR ADVANCED SENSORS APPLICATIONS.	5. TYPE OF REPORT & PERIOD COVERED: Technical Report
6. PERFORMING ORG. REPORT NUMBER: 43U1408	
7. AUTHOR(s): B. Owen (MIRADCOM) Larry; James B. Clary (Research Triangle Institute)	8. CONTRACT OR GRANT NUMBER(s): DAAG29-76-D-0100
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Research Triangle Institute, Research Triangle Park, North Carolina 27709	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: DA 1M362303A214 AMCMS 662303.1121401
11. CONTROLLING OFFICE NAME AND ADDRESS: Commander, US Army Missile Research and Development Command, Attn: DRDMI-TI, Redstone Arsenal, Alabama 35809	12. REPORT DATE: 31 Sep 77
	13. NUMBER OF PAGES: 62
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office): Commander, US Army Missile Research and Development Command, Attn: DRDMI-TE, Redstone Arsenal, Alabama 35809	15. SECURITY CLASS. (of this report): UNCLASSIFIED
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

DDC RECEIVED MAR 16 1978 F

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Microprocessor
Signal processing
Large scale integration
Task allocation

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

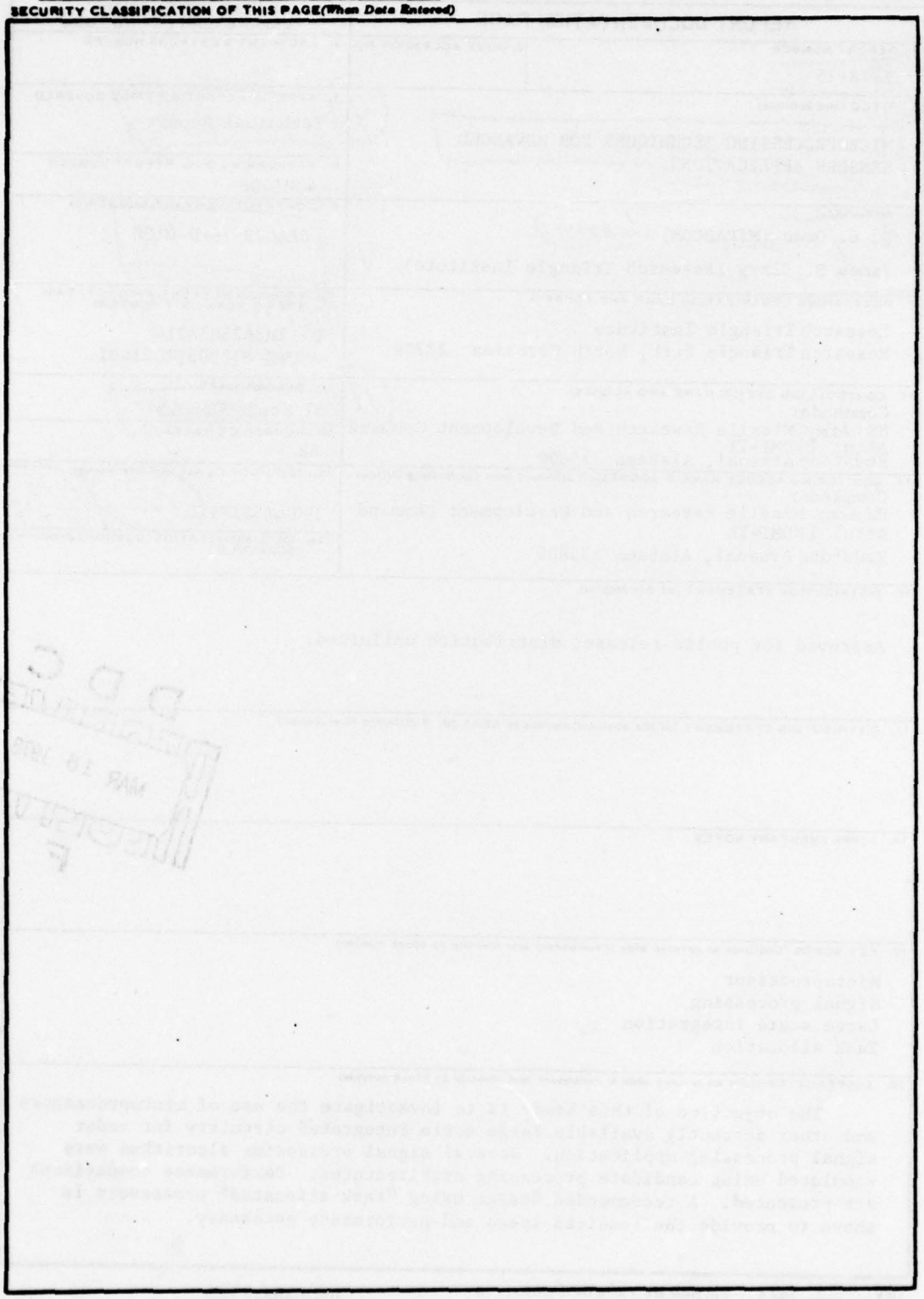
The objective of this study is to investigate the use of microprocessors and other currently available large scale integrated circuitry for radar signal processing application. Several signal processing algorithms were simulated using candidate processing architectures. Performance comparisons are presented. A recommended design using 'task allocated' processors is shown to provide the required speed and performance necessary.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

391 859



CONTENTS

		Page
I.	INTRODUCTION	3
II.	CANDIDATE CLASSES OF SIGNAL PROCESSOR ARCHITECTURES	4
III.	BASELINE PROCESSING REQUIREMENTS	15
IV.	REQUIREMENTS VERSUS SIGNAL PROCESSOR ARCHITECTURE TRADEOFFS	15
V.	*RECOMMENDATIONS AND CONCLUSIONS*	21
REFERENCES		25
Appendix A.	8080A PROCESSOR BENCHMARK CONFIGURATIONS	27
Appendix B.	8080A MICROPROCESSOR BENCHMARK PROGRAM LISTINGS	46

ACKNOWLEDGMENTS

This report was prepared for the US Army Missile Research and Development Command under Batelle Laboratories Delivery Order No. 0338. The author gratefully acknowledges the technical guidance provided during this study by Mr. Larry B. Owen of the US Army Missile Research and Development Command and the programming support of Mr. Sasan Ardalan, a graduate student in the Electrical Engineering Department at North Carolina State University.

I. INTRODUCTION

Greater understanding of the theory of digital signal processing coupled with dramatic advances in hardware technology and software engineering has led to improved remote sensing capabilities for military and civilian applications. Nowhere has the impact of theoretical concepts and hardware/software technological advances been felt more than in the field of radar remote sensing. This is due to a large extent to the inherent difficulty of the basic radar signal processing problem.

Radar signal processing differs from other signal processing problems in that very high data throughput rates as well as wide dynamic ranges are often simultaneously required. This unfortunate coincidence results directly from the physics of the remote sensing problem where noncooperative (evasive) target detection using primary radar is hampered by partically correlated noise in the form of ground, weather, and chaff clutter. As a consequence, radar signal processors must be fast (capable of high data throughput rates) and intelligent (capable of executing algorithms which can distinguish between correlated noise and targets of interest). The challenge, then, is to define a radar signal processor which can execute the resulting complex algorithms in real-time with arithmetic precision adequate to allow differentiation between small cross-section targets and large cross-section clutter.

A. Study Objectives

The objective of this study is to investigate the use of microprocessors and other currently available large scale integrated (LSI) circuitry for radar signal processing and to define a structure which is capable of executing algorithms in real-time. A processing throughput objective is provided by the Advanced Sensors Directorate's Quiet Radar parameters [1]. The general processing requirements of this radar serve as a baseline for the present study. While the study is theoretical in nature, the use of these real-world radar parameters tends to anchor the results in a context which can be meaningful in the near-term. In particular, it is envisioned that this study will serve as the basis for the later design and fabrication of a high speed, flexible radar signal processor with a broad range of applications in remote sensing.

A great deal of attention has been focused in recent years on the application of microprocessors to various data processing and control applications. More recently, advances in microprocessor technologies have presented apparent opportunities for increased data throughput and processing flexibility. As a consequence, various processors designed especially for high throughput have been proposed [2-6]. The present task has considered the application of microprocessors as well as other state-of-the-art LSI potentially suitable for use in high-speed signal processing applications.

B. Study Scope

The scope of this work encompasses the basic elements of a radar signal processor architecture analysis at the block diagram level. Tradeoffs are made between candidate approaches as they apply to remote sensing in general and to the Advanced Sensors Directorate's Quiet Radar in particular. It is envisioned that the futher reduction of these system level descriptions to fundamental logic circuit diagrams will be accomplished in a related follow-on effort.

C. Study Approach

The approach taken in this study was to investigate signal processor hardware architectures to determine their applicability to the radar processing problem. A basic assumption throughout this study was that existing state-of-the-art LSI including microprocessors and special purpose LSI hardware would be used as the processor building blocks. Analysis of candidate architectures was carried on in light of the baseline signal processing throughput requirements of the Advanced Sensors Directorate's Quiet Radar program but was not restricted to consideration of these parameters only. Consideration was also given to modular architectures which offer flexibility in terms of expansion through replication of constituant components. Such architectures have certain cost advantages as well as robustness in terms of hardware and software reliability and maintainability.

The following discussion presents the study results in a top-down fashion. Candidate classes of signal processor architectures are first discussed. Desirable attributes as well as shortcomings of microprocessor-based signal processors are then considered in relation to the high-speed radar signal processing problem. Succeeding sections relate candidate processor architectures to the baseline radar parameters of interest. Finally, a specific radar signal processor architecture is proposed as a candidate for later detailed design and fabrication.

II. CANDIDATE CLASSES OF SIGNAL PROCESSOR ARCHITECTURES

It is desirable to define a radar signal processor architecture which achieves maximum data throughput and flexibility with a minimum investment in hardware and software. Candidate processing elements to be used in this study are provided by the families of LSI circuits presently available. Of these classes, the following have been considered:

a) Eight-bit metal oxide semiconductor (MOS) microprocessors.

b) Four-bit slice microprocessors.

c) Eight-bit microprocessors + special purpose arithmetic unit.

d) Special purpose arithmetic hardware.

The performance-related characteristics of devices of this nature are as follows:

a) Instruction cycle time.

b) Data word width.

c) Instruction set.

d) Small scale integration/medium scale integration (SSI/MSI) overhead required.

e) Bus structure.

f) Input/Output (I/O) capabilities.

The performance characteristics were considered for each of the previously listed classes of microprocessor architectures. The following sections consider three specific classes of processor architectures.

A. Single Mircroprocessor Architectures

Previous works on multiprocessing systems have defined single central processing unit (CPU) computers as "Single-Instruction Single Data (SISD)" machines [7]. The majority of the general purpose computers presently in use are SISD architectures. SISD architectures use a single-control unit to route data into and out of the CPU. As a result only one arithmetic process such as addition, subtraction, multiplication, or division can occur at one time. Furthermore, the movement of data is usually accomplished by means of a single data bus in such designs.

The primary advantage of single-instruction single data architectures is the simplicity of the hardware and software structures. These machines require little in the way of hardware and are straightforward to program. Unfortunately, these attributes are achieved at the expense of data throughput and flexibility as demonstrated in later sections of this report. However, the SISD class processor remains important because it is the primary building block of more complex processors.

1. Non-Bit Slice Processors

A micorporcessor which illustrates the SISD architecture is the Intel 8080A, 8-bit machine. For purposes of this study, the 8080A has been chosen as the baseline architecture to which other designs may be compared. Specifically, the 8080A was chosen for the following reasons:

a) It is the most widely used microprocessor.

b) It is low-cost and readily available.

c) It is reasonable to consider an array of such machines in more complex architectures.

Unfortunately, the single data bus structure permits only limited data flow. Obvious variations of this basic structure therefore include multiple operational and resultant buses to permit increased flexibility in terms of data management alternatives.

2. Bit-Slice Processors

Attempts to achieve higher data throughput rates with programmable LSI have resulted in the creation of bit-slice microprocessor devices. The primary advantage of these devices is their faster instruction cycle time. Their major disadvantages include increased part counts due to support circuit requirements and the fact that they are generally harder to program. Bit-slice microprocessors have been used in two primary application areas as follows:

a) As instruction set emulators where microprogrammed bit-slice machines are made to look like other processors.

b) In moderate speed signal processing applications where advantage can be taken of their micro-instruction power and faster instruction execution time.

Bit-slice processors using the Motorola M10800 have been designed by Motorola [8], Raytheon [9], and others. These machines have instruction cycle times on the order of 100 nsec.

Another bit-slice microprocessor is the Advanced Micro Devices (AMD) AM2900, 4-bit-slice device. This processor uses Schottky bipolar LSI technology and executes instructions at a rate of approximately 250 nsec. Although the AM2900 cycle time is slower than the M10800, it is generally easier to program because the AM2900 is a 2-bus structure while the M10800 is a 3-bus design. Thus, more options and potentially more powerful instructions are available with the M10800.

At the present time, greater software support is available with the AM2900 including a cross-assembler. However, Motorola is in the process of developing a software support package for the M10800 which should be available by the end of the calendar year.*

*Balph, Tom, Motorola, Inc., Phoenix, Arizona, September 1977 (Private Communication).

Finally, a third bit-slice microprocessor has recently become available. This machine differs from the others in that it is an 8-bit-slice device. The part has been introduced by RCA and is known as the ATMAC microprocessor. Unfortunately, this device is not available as a commercial component but may be purchased only from RCA as a part of a system. On the positive side, RCA does have extensive documentation and software support available [10-12]. The ATMAC is attractive to the signal processor designer for the following reasons:

a) It combines low power with high speed by using complementary metal oxide semiconductor/silicon-on-sapphire (CMOS/SOS) technology to give a very low speed-power product.

b) It is an 8-bit-slice (versus a 4-bit-slice) device and therefore requires fewer components to realize a total system.

c) It has provisions for a peripheral special function unit (SFU) which can be a high-speed multiplier device, for example.

Unfortunately, the ATMAC 8-bit-slice microprocessor data throughput is limited by the incorporation of only a single bus I/O structure. This appears to be its greatest architectural weakness. Discussions with RCA personnel generally confirm this limitation.*

B. Multi-Microprocessor Architectures

Arrays of low-cost microprocessors performing multiple computational tasks in parallel have been considered as an alternative for achieving higher data throughput rates in radar signal processing applications. The obvious advantage of such an approach is the redundancy inherent in such a design which can lead to a more survivable processor in case of component failures. The obvious disadvantage of this approach is the difficulty in programming such a structure.

Arrays of 8-bit MOS microprocessors are potentially more attractive than arrays of 4-bit-slice bipolar designs because of the lower parts count and generally lower cost of the 8-bit processors. Unfortunately, the 8-bit devices have a single I/O bus while the 4-bit parts have one or two operand buses and a resultant bus.

C. Task-Allocated Processor Structures

In radar signal processing, various tasks of differing complexity and speed must be performed. One possible way to arrive at a radar signal processor architecture is to determine the needs of each processing subtask and to create the required computational resources necessary to accomplish those tasks. This approach will be referred to in this discussion as "task-allocated" signal processing.

---

*Helbig, Walter, RCA Advanced Technology Laboratories, Camden, New Jersey, August 1977 (Private Communication).

At the crux of the radar digital signal processing problem are the high data throughput requirements. Therefore, a logical place to begin in defining a task-allocated structure is with those subtasks which have the most demanding throughput requirements. In most coherent radars, filtering operations require the greatest number of arithmetic operations in the shortest time interval. These operations generally consist of:

1) High-pass filtering — Moving Target Indication (MTI).

2) Low-pass filtering (clutter maps).

3) Band-pass filtering (Doppler filtering).

The high-pass and low-pass filters may be synthesized using well-known finite impulse response (FIR) and infinite impulse response (IIR) techniques [13-15]. The band-pass filters required for Doppler processing are usually realized with the fast Fourier transform (FFT) algorithm [14, 15].

The top-down approach taken in this study was to determine how fast various microprocessor and special purpose hardware structures could perform the required computations necessary to accomplish these filtering operations. Filter order (weights) and transform length were used as parameters of interest. A priori knowledge of the Quiet Radar performance requirements were used to determine approximate filter orders and transform lengths of interest. Arithmetic precision was initially assumed to be 16-bits. It is expected that further work will be performed to determine the validity of this assumption.

The first task to be considered is that of MTI filtering. Such filters may be realized as either conventional N-pulse cancellers which require that

$$y_n = a_o X_n - a_1 X_{n-1} - \cdots - a_{n+1} X_{n-1} \quad , \tag{1}$$

where $X_n = N^{th}$ input data word. In the simplest case, the coefficients, a_i, are unity. Thus, the simplest two-pulse canceller requires only a single subtraction for each return pulse.

More sophisticated MTI filters may be realized using FIR synthesis techniques. Furthermore, FIR designs can be high-pass, low-pass, or band-pass, depending upon the coefficients selected. Figure 1 presents the computational requirements of an FIR design. The structure depicted in this figure computes

$$y_n = \sum_{k=0}^{K} a_k X_{n-k} \quad . \tag{2}$$

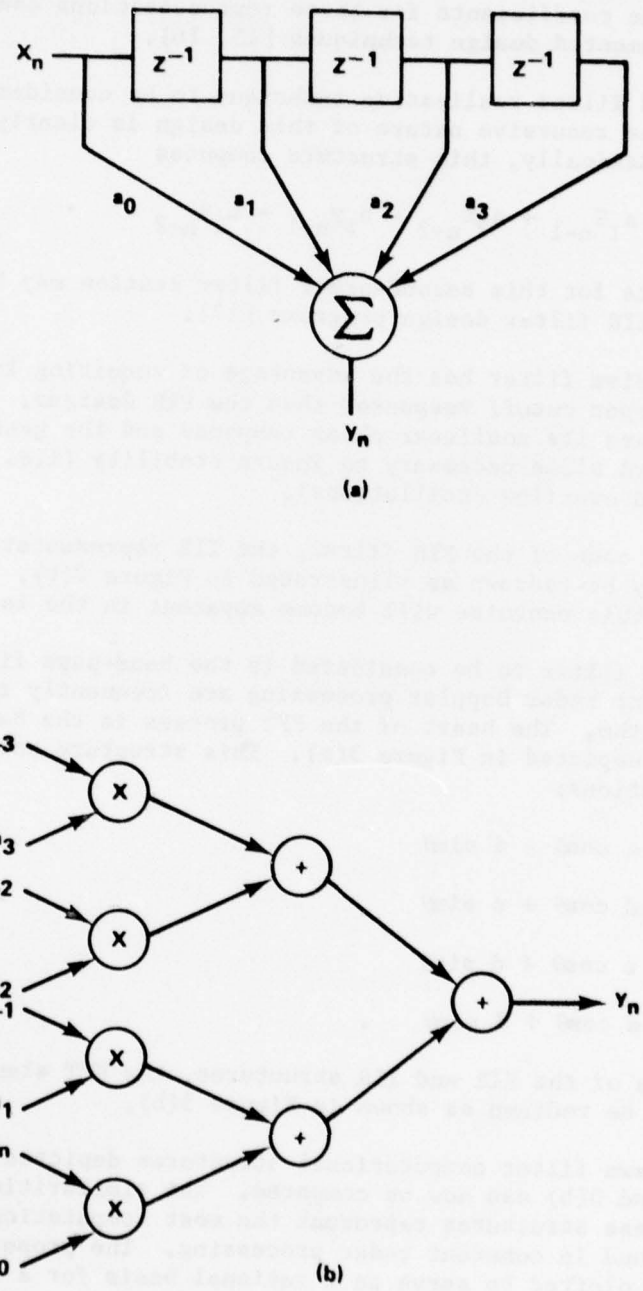


Figure 1. FIR digital filter.

The structure depicted in Figure 1(a) may be redrawn as shown in Figure 1(b). The usefulness of this representation will be shown later in this discussion. The coefficients for these representations can be determined using well-documented design techniques [15, 16].

The second filter realization technique to be considered is the IIR filter. The recursive nature of this design is clearly illustrated in Figure 2. Basically, this structure computes

$$y_n = X_n + a_1 X_{n-1} + a_2 X_{n-2} - b_1 y_{n-1} - b_2 y_{n-2} \quad . \tag{3}$$

The coefficients for this second-order filter section may be determined from existing IIR filter design programs [17].

The recursive filter has the advantage of requiring lower orders to achieve sharper cutoff responses than the FIR designs. Its major disadvantages are its nonlinear phase response and the generally greater coefficient word sizes necessary to insure stability (i.e., to minimize limit cycle and overflow oscillations).

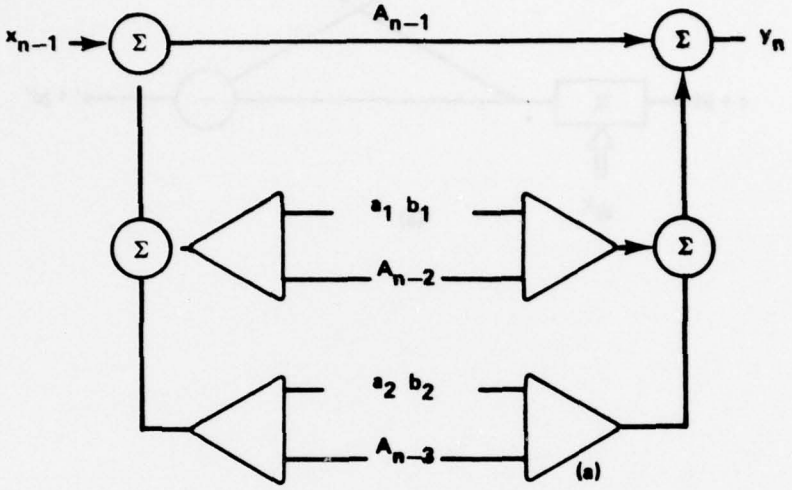
As in the case of the FIR filter, the IIR representation shown in Figure 2(a) may be redrawn as illustrated in Figure 2(b). Again, the usefulness of this exercise will become apparent in the later discussion.

The third filter to be considered is the band-pass filter. Band-pass filters for radar Doppler processing are frequently realized using the FFT algorithm. The heart of the FFT process is the basic computational element depicted in Figure 3(a). This structure solves the following equations:

$$\begin{aligned} a' &= a + c\cos\theta - d\sin\theta \\ b' &= b + d\cos\theta + c\sin\theta \\ c' &= a - c\cos\theta + d\sin\theta \\ d' &= b - d\cos\theta + d\sin\theta \quad . \end{aligned} \tag{4}$$

As in the case of the FIR and IIR structures, the FFT elemental computations may be redrawn as shown in Figure 3(b).

The redrawn filter computational structures depicted in Figures 1(b), 2(b), and 3(b) can now be compared. The similarities are quite apparent. These structures represent the most computationally demanding algorithms found in coherent radar processing. The properties evident here can be exploited to serve as a rational basis for a high-speed radar signal processor design as shown in the following discussion.



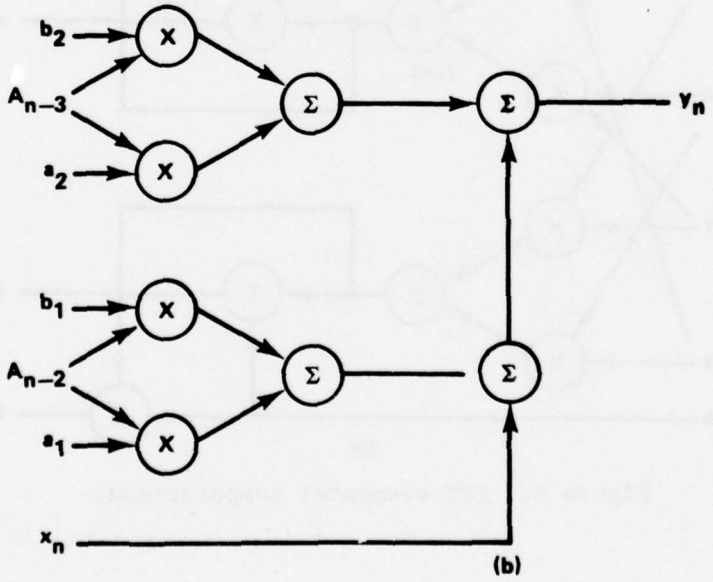
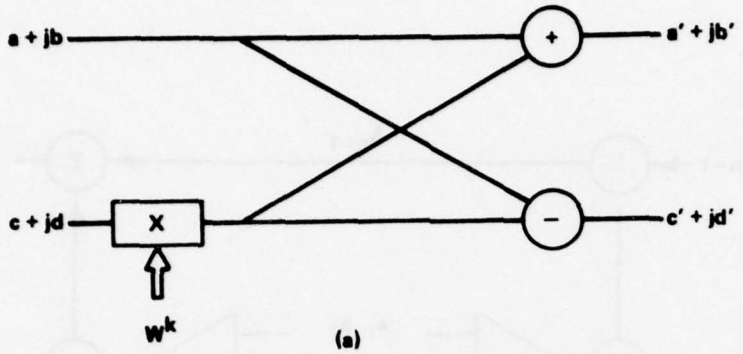


Figure 2. IIR digital filter (second-order section).



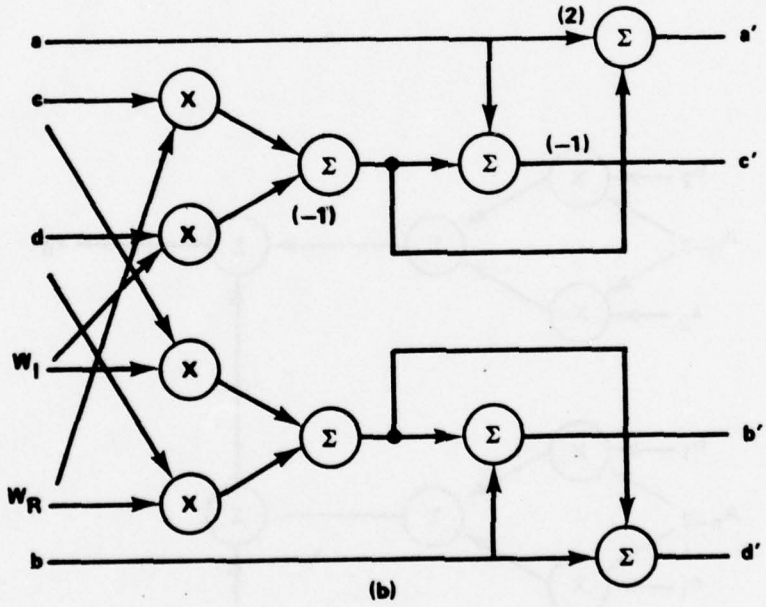


Figure 3. FFT elemental computations.

The arithmetic elements common to the redrawn FIR, IIR, and FFT structures are four multipliers and several adders. Therefore, taking the union of the three configurations shown in Figure 1(b), 2(b), and 3(b) and minimizing the functional arithmetic elements results in a basic signal processing structure which can accommodate the high throughput algorithms required by coherent radars. Two important questions are (1) how to route the data efficiently to and from the high-speed computational elements, and (2) how to store and buffer I/O and partial result data effectively.

The approach taken to data routing in the proposed special purpose processor is to use multiple parallel data buses to avoid the common problem of bus-limited data transfers. Such an approach has the potential for achieving 100% computational efficiency by supplying data continuously to the arithmetic computational elements.

The problem of special purpose processor data storage can be met with high-speed, distributed memory capable of accepting data in parallel from multiple buses. An important advantage of distributed memory, in addition to having multiple I/O ports available to accommodate pipelined data, is the ability to do data steering (switching) by clever memory addressing techniques. If the special purpose processor arithmetic unit is envisioned as a miniswitching system, the data storage elements can be used in much the same way as in large electronic switching systems such as the Bell System's Electronic Switching Systems (ESS) machine.

The final important concept to be discussed briefly in addition to arithmetic, data-bus, I/O, and memory is that of special purpose processor control. To achieve maximum flexibility and to guarantee that the resulting structure will compute the FIR, IIR, FFT, and other signal processing algorithms efficiently, it is proposed that the basic control be microprogrammable. In a signal processing structure such as that proposed here, the microprogram object code may be thought of as data switch enables/disables. Thus, the data steering function is controlled by the processor microprogram. The microprogram itself can reside either in Read-Only-Memory (ROM), assuming that all processing algorithms to be executed are known a priori or it can reside in Random Access Memory (RAM) which can be loaded by a more intelligent machine such as a microprocessor or minicomputer.

The union of all the ideas briefly outlined in the preceding paragraphs have been incorporated in a proposed radar signal processing structure which is illustrated in Figure 4. It is proposed that the special purpose arithmetic unit be constructed of computational and memory components which will permit a 200-nsec pipelined throughput. Under the assumption that such a processor can be constructed, the resulting FIR, IIR, and FFT data throughput rates as a function of filter order and FFT transform length can be determined.

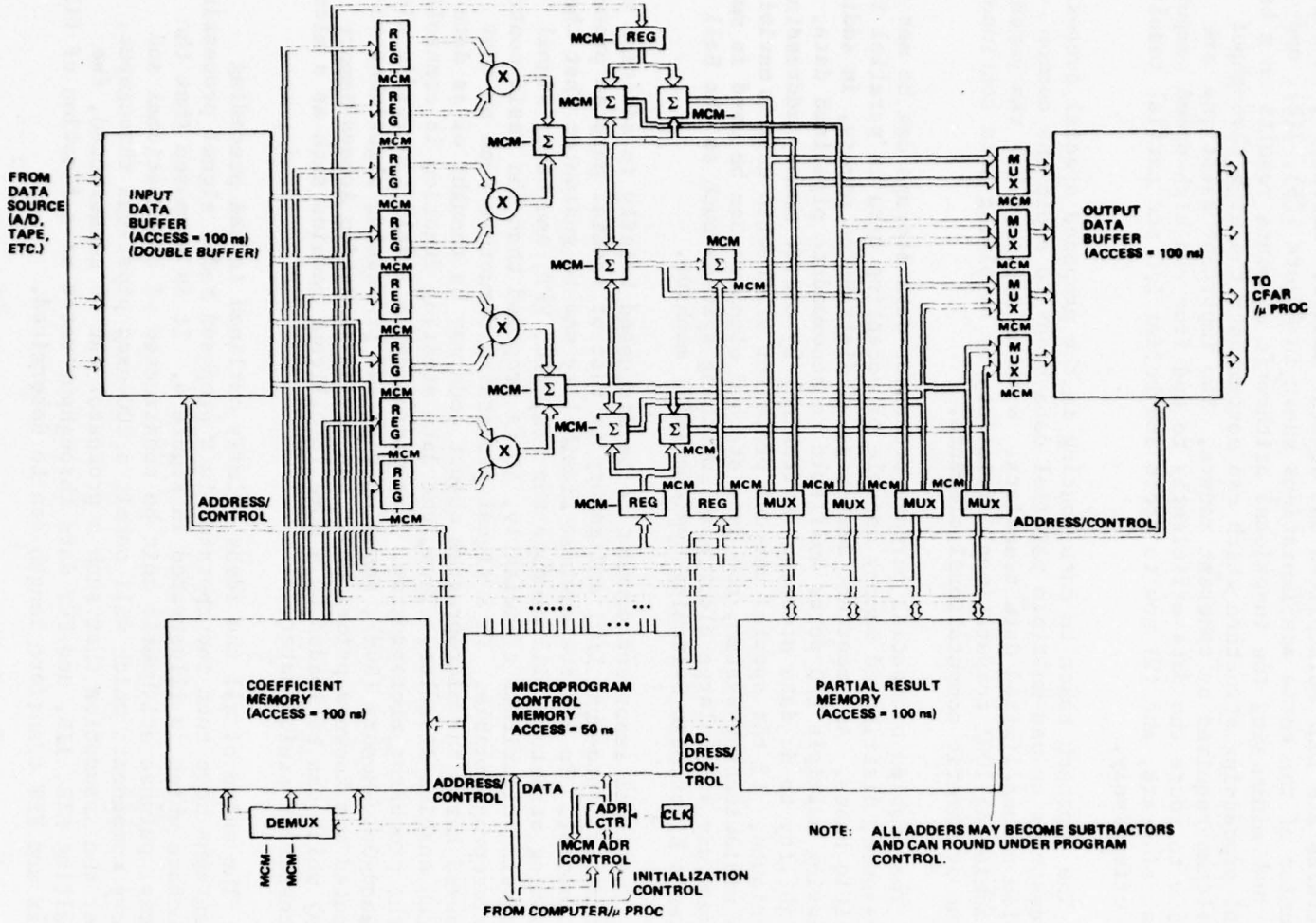


Figure 4. Proposed radar signal processor arithmetic unit.

III. BASELINE PROCESSING REQUIREMENTS

The baseline processing requirements for this study are provided by the Advanced Sensors Directorate's Quiet Radar. This is a phase-coded continuous wave (CW) radar with the following characteristics:

a) Code shift rate = 5 MHz.

b) Code length = 63 bits.

c) Antenna dwell time = 2 msec.

d) Number of Code Periods per dwell = 150.

e) Doppler Coverage = ± 25 kHz.

f) Number of Doppler lines = 100.

A number of specific Quiet Radar processor configuration alternatives have been carefully considered by the US Army Missile Research and Development Command (MIRADCOM) and therefore will not be iterated here. The important parameters to note are the code shift rate (5 MHz), the antenna dwell time (2 msec) and the Doppler cutoff frequency (25 kHz).

Based upon a processing interval of 2 msec and 63 range cells/dwell, the per range cell computation interval is 31.7 μsec complex or 15.8 μsec per real channel. This computation interval may be compared to the times required to compute various orders of FIR, IIR, and FFT transform lengths discussed in the following sections.

IV. REQUIREMENTS VERSUS SIGNAL PROCESSOR ARCHITECTURE TRADEOFFS

With the Quiet Radar processing requirements as a reference point, the signal processor architectures described earlier may be considered. The following sections discuss single processors, multiprocessors, and the task-allocated signal processor approaches.

A. Single Processors

The single microprocessors considered in this study were the 8-bit MOS, 4-bit-slice bipolar, and 8-bit-slice CMOS/SOS devices. To determine their suitability for computing the signal processing algorithms discussed earlier, i.e., the FIR filter, IIR filter, and FFT, these algorithms were either encoded and implemented to provide benchmark timing requirements or, where possible, were taken from the literature.

The Intel 8080A, which has become an industry standard 8-bit microprocessor was chosen to provide a baseline upon which the other approaches may be compared. This is a reasonable choice because many versions of the 8080A exist in the form of high-speed bipolar emulators as well as software upward compatible, higher speed devices such as the Z-80 microprocessor.

Figure 5 depicts the 8080A baseline throughput capability for three possible configurations. These configurations are as follows:

1) 8080A with software multiply.

2) 8080A with the AMD AM9511 arithmetic processor chip.

3) 8080A with a high-speed peripheral multiplier such as the TRW single chip devices.

As could be anticipated, the 8080A with software multiply only is by far the slowest configurations. For example, this configuration requires approximately 10 msec to compute a 16-weight FIR filter. The details of the 8080A configuration and the software used establish this benchmark are given in Appendices A and B of this report.

Figure 5 shows that a half-order magnitude speed-up can be achieved using the 8080A augmented with the AMD arithmetic unit (AM9511). The configuration used is described in detail in Appendix A.

The third 8080A configuration considered was the 8080A coupled with a high-speed multiplier. This configuration results in a throughput increase which is close to an order of magnitude faster than the 8080A with software multiply. It is significant that the increase in throughput achieved by the 8080A with the high-speed multiplier relative to the 8080A with the AMD device is not as great as expected. The fundamental reason for this is that as the peripheral devices become faster, the basic throughput limitation becomes I/O bound. This is true in the 8080A case even with memory mapped I/O.

It should be pointed out that faster versions of the 8080A would shift these curves down in a corresponding manner. Additional speed-up could also be achieved with an expander instruction set such as that available with the Z-80. However, as seen later in this discussion, the overall throughput increase would not be consequential relative to most coherent radar processing requirements.

Two additional single processor architectures were considered for reference. The first of these is the Motorola MOD System which is basically an 8-bit processor composed of two-slices of the M10800 microprocessor [8]. Its performance, assuming the use of Booth's algorithm to perform the software multiplies is shown in Figure 5.* It can be observed that for the FIR algorithm, the throughput can be increased by a factor of 10 over the baseline 8080A using Booth's algorithm. However, it is important to recall that this increased throughput is achieved at the expense of the much higher component parts count required by bit-slice microprocessors as well as more complex software.

*Balph, Tom, Motorola, Inc., Phoenix, Arizona, September 1977 (Private Communication).

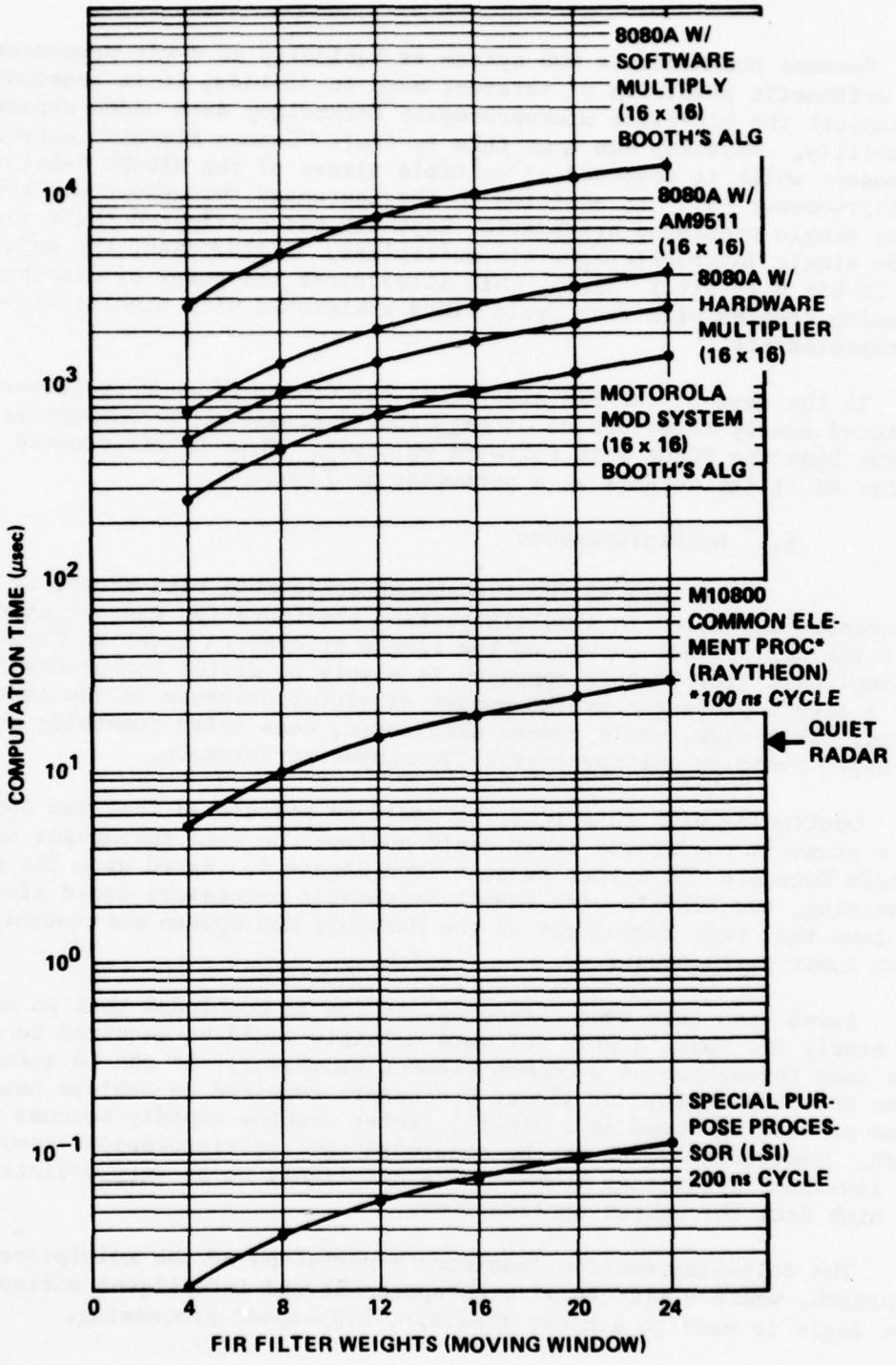


Figure 5. FIR filter throughput.

Because the Motorola MOD system is basically an 8-bit processor and the arithmetic precision of interest here is 16-bits, it is reasonable to exploit the bit-slice microprocessor technolgoy data width expansion capability. Raytheon has done this in their "Common Element" approach processor which is composed of multiple slices of the M10800 4-bit-slice microprocessor [9]. In addition to the increased throughput achieved by doing single precision arithmetic, Raytheon has configured their machine to do single instruction-per-bit multiplies. In this case, the multiplies are 12-bit $\times$ 12-bit.) Again, this illustrates the power of microprogramming coupled with fast cycle times achievable with bipolar bit-slice microprocessors.

In the case of the FIR algorithm, the Common Element processor achieved nearly three orders of magnitude increase in throughput relative to the baseline 8080A with software multiply. This is illustrated with number of filter weights as a parameter in Figure 5.

B. Multiprocessors

An idea of the throughput achievable with arrays of microprocessors assembled in a multiprocessor configuration can be inferred from the performance curves of the single processors given in Figure 5. A simplistic view of this approach is simply to divide the processing time for a single processor by the number of microprocessors in the assembled array. This view, while indeed simplistic, does infer something about an upper bound on multiprocessor throughput performance.

Continuing with this idea, it could be postulated that ten 8080A's in a properly configured array could achieve the same throughput as a single Motorola MOD System as seen from Figure 5. Based upon the same reasoning, two 8080A's with AM9511 arithmetic processors could also achieve the same throughput as the Motorola MOD System and probably at a much lower parts count.

Based upon this same reasoning, it can be concluded that an array of nearly 300 8080A's with software multiply would be required to achieve the same throughput as a Common Element processor. It can be quickly seen that the number of slower processors required to achieve nearly the same processing speed as a single, faster machine rapidly becomes very high. Consequently, it can be concluded that multiprocessors composed of low-speed processing elements are not likely to be very efficient in high data throughput applications.

The following section considers a variation of the multiprocessor approach, where a mixture of high-speed LSI and intelligent microprocessor logic is used to achieve flexible, high-speed processing.

C. Task-Allocated Processing

The lower bound on data throughput for purposes of this study is achieved by the special purpose radar processor described in Section II.C. Figure 5 illustrates that such a design can potentially realize data throughput rates which are four to five orders of magnitude faster in computing the FIR algorithm than the baseline 8080A processor discussed earlier (Section II.A and Appendices A and B). The fact that the proposed special purpose processor is approximately two orders of magnitude faster when computing the FIR algorithm than the M10800 Common Elements approach proposed by Raytheon is illustrated in Figure 5. Viewed another way, this says that if 100 Common Element processors are required to achieve the necessary data throughput, only one special purpose processor can replace all 100 Common Element processors.

The central issue here is obvious tradeoff between processor data throughput and processor flexibility. However, a great deal of flexibility can be achieved with a special purpose arithmetic unit through the use of programmable control. The control structure of the processor shown in Figure 4 essentially performs a data routing role. Thus, by making these operations microprogrammable, the special purpose unit can be made to perform a large number of different algorithms and thereby overcome processor flexibility limitations. It is proposed that additional work be undertaken to define specifically the nature of this control structure. One possibility which should be studied further is the incorporation of a microprocessor to perform such control functions.

A second potential application for a microprocessor in a task-allocated structure is as a post-processor. That is, after the high-speed algorithm processing has been accomplished by the special purpose LSI (i.e., the multipliers, adders, high-speed memory, etc.) more sophisticated, but lower throughput, processing is usually required. For example, Constant False Alarm Rate (CFAR) processing with associated thresholding and clutter map generation may be accomplished in a moderate-speed microprocessor. This area is also identified as one where additional work is needed.

Figure 6 represents the same benchmark approach to processor comparison illustrated in Figure 5 except that Figure 6 is for the IIR algorithm. The two figures are similar, but results are given for different filter orders. A comparison of the FIR and IIR algorithms shows that the basic memory access required for each is first-in, first-out (FIFO). In addition, it has already been shown that the computational elements themselves are similar (Figures 1 through 3). Therefore, it is reasonable for the benchmark throughput results to be similar.

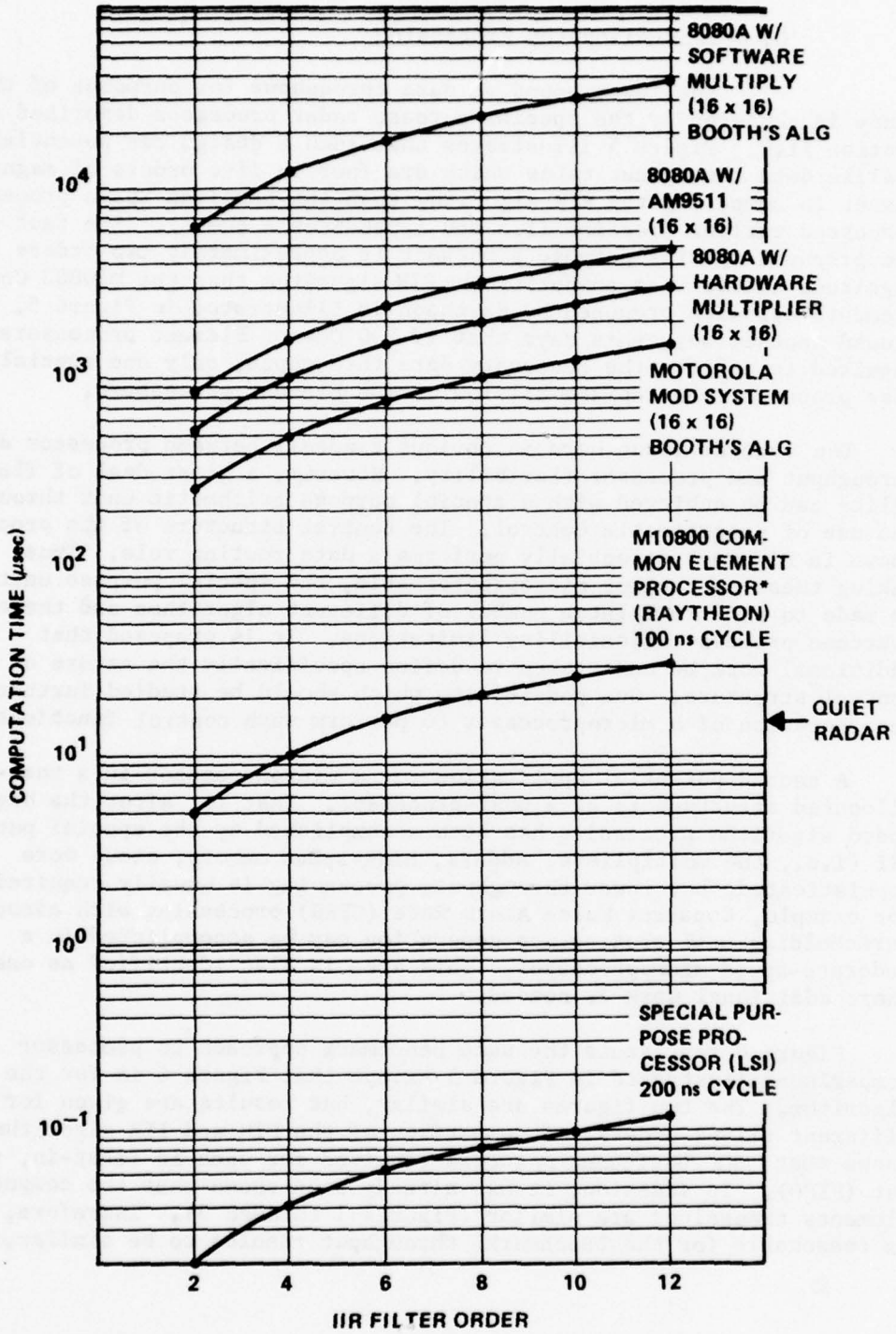


Figure 6. IIR filter throughput.

The most computationally demanding coherent radar processing algorithm is the band-pass Doppler filtering often accomplished using the FFT algorithm. Figure 7 presents the results obtained using the signal processor configurations described earlier. The length transforms considered were from 8 to 256 points. Earlier work has shown that realistic transform lengths for the Quiet Radar are from 64 to 256 points.*

It can be observed from Figure 7 that for a 128-point transform, for example, the special purpose processor is nearly four orders of magnitude faster than the baseline 8080A processor with software multiply. As in the case of the FIR and IIR algorithms, the bit-slice processors fall between these bounds. One additional interesting FFT benchmark shown in Figure 7 is the RCA 8-bit-slice ATMAC processor with SFU [18].

The throughput achieved with the RCA device is comparable to that achieved with the Raytheon Common Element approach in the case of the FFT algorithm. However, this throughput is achieved with a lower parts count because the ATMAC represents a higher level of circuit integration (i.e., 8-bit-slice versus 4-bit-slice). Details of this processor are given in References 6 and 10.

V. RECOMMENDATIONS AND CONCLUSIONS

The general problem of realizing a flexible, high throughput signal processor for coherent radar applications has been considered. The approach taken in this study has been to investigate various microprocessor configurations and to evaluate their capabilities through the use of benchmark radar signal processing algorithms. As a baseline configuration, the popular 8080A microprocessor was chosen. Various configurations of the 8080A with software multiply only and versions of the 8080A augmented with special peripheral arithmetic hardware were considered. The signal processing algorithms of interest were programmed on these machines and their throughput capabilities determined. Both the AMD AM9511 arithmetic processor and a high-speed peripheral multiplier were used to augment the basic 8080A.

The ability of bit-slice microprocessors to process the benchmark coherent radar algorithms was evaluated as a part of this study. The particular 4-bit microprocessor considered was the faster cycle time device currently commercially available, namely, the Motorola M10800. The two configurations evaluated and compared to the baseline 8080A processor were the Motorola MOD System and the Raytheon Common Element processor. Where data were available, the RCA 8-bit ATMAC processor was also considered.

*Burlage, Don, US Army Missile Research and Development Command, Redstone Arsenal, Alabama, September 1977 (Private Communication).

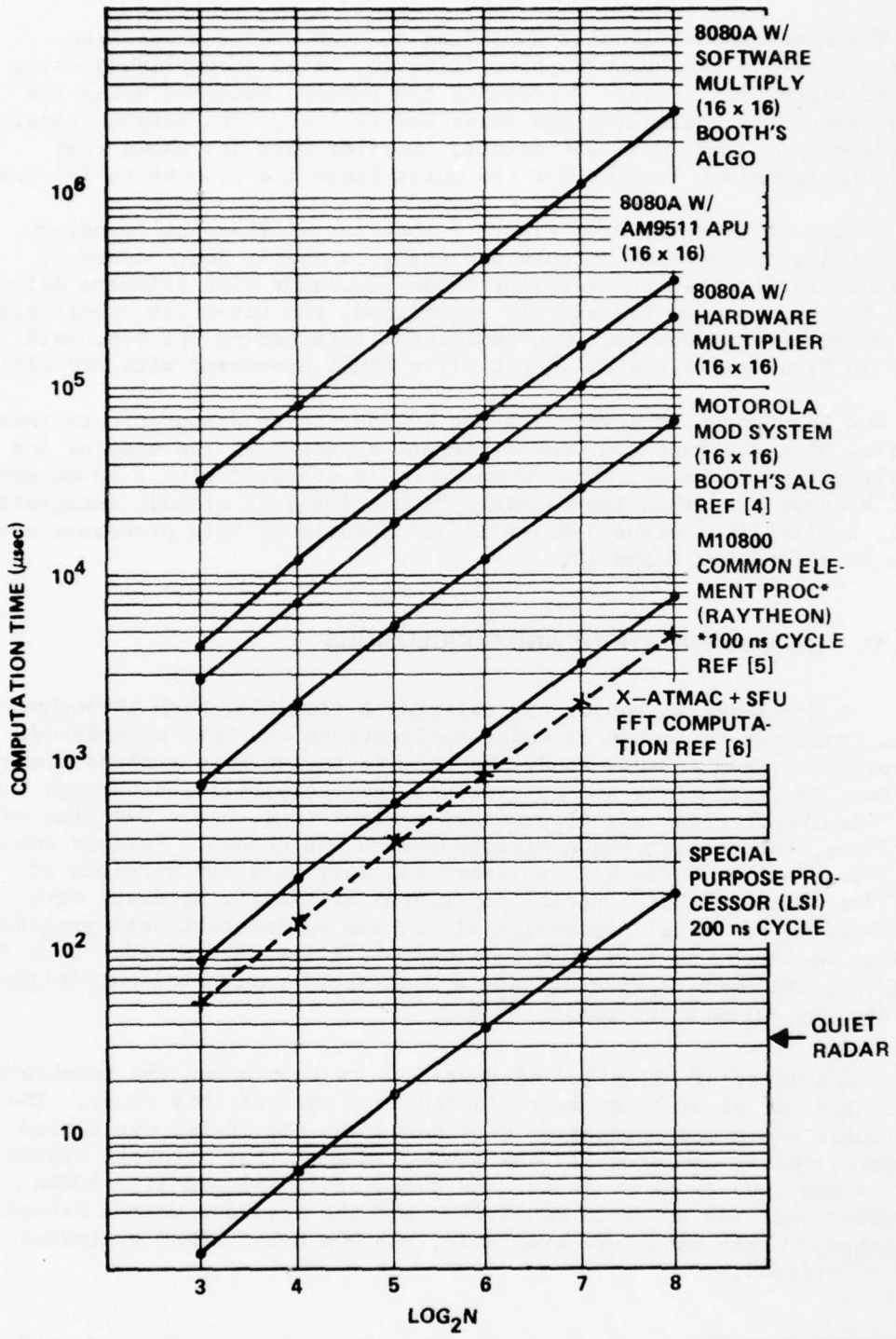


Figure 7. FFT throughput.

Finally, a high-speed arithmetic unit comprised of special purpose LSI circuitry was configured and its throughput capabilities evaluated. The rational basis for this design was the similarity of the coherent radar signal processing algorithms of interest. The throughput performance of the single processors, multiprocessors, and task-allocated processors were considered with filter order and transform length as parameters.

The major conclusion of this study is that a carefully configured combination of high-speed, special purpose LSI coupled with distributed intelligence in the form of microprocessors can effectively meet the throughput and flexibility requirements of coherent radars. More specifically, it has been determined that such an embodiment can meet the processing needs of the Quiet Radar. It is therefore recommended that additional work be undertaken to answer remaining performance questions and that following this effort, such a processor be constructed and interfaced with the Quiet Radar.

REFERENCES

1. Boothe, R. R., Fletcher, R. H., Holliday, E. M., and Kiss, C. J., Quiet Radar/Missile Guidance Technical Assessment Final Report (U), US Army Missile Command, Redstone Arsenal, Alabama, September 1975, Report No. RE-76-12.

2. Barrett, R. C., Bork, G. F., and Hattendorf, M. D., High Speed Microprocessor Study, Aerospace Group, Hughes Aircraft Company, Culver City, California, October 1975, Report No. AFAL-TR-75-44.

3. Roberts, R. F., "High Speed Sensor Signal Processing Using Microprocessor Technology," NAECON '77 Record, pp. 562-565.

4. Tao, T. F., Yehoshua, D. B., and Martinez, R., "Applications of Microprocessors in Control Problems," Proceedings of the 1977 Joint Automatic Control Conference, San Francisco, California, June 1977.

5. Cohen, H. I., "The MSP: GTE Sylvania's Microsignal Processor," EASCON '76 Record, Boston, Massachusetts.

6. Helbig, W. A., and Stringer, J. B., "The RCA ATMAC, AVLSI Computer for High-Speed Data Processing," Microcomputer '77.

7. Baer, J., "Multiprocessing Systems," IEEE Transactions on Computers, Vol. C-25, No. 12, December 1976.

8. Balph, T., "The Motorola MOD System," Motorola, Inc., Phoenix, Arizona, 1976.

9. Freedman, N., Cornwell, P., and Barr, P., "Final Report Automatic Threshold Detector Techniques," Equipment Development Laboratories, Raytheon Company, Wayland, Massachusetts, July 1976.

10. Programmer's Reference Manual — ATMAC, RCA, Camden, New Jersey, 1977.

11. Simulators User's Manual — ATMAC, RCA, Camden, New Jersey, 1977.

12. Cross-Assembler User's Guide — ATMAC, RCA, Camden, New Jersey, 1977.

13. Otnes, R. K. and Enochson, Loren, Digital Time Series Analysis, New York: John Wiley and Sons, 1972.

14. Oppenheim, A. V., and Schafer, R. W., Digital Signal Processing, Englewood Cliffs, New Jersey: Prentice-Hall, 1975.

15. Rabiner, L. R. and Gold, B., Theory and Application of Digital Signal Processing, Englewood Cliffs, New Jersey: Prentice-Hall, 1975.

16. Burlage, D. W. and Honts, R. C., Design Techniques for Improved Bandwidth Moving Target Indicator Processors In Surface Radars, US Army Missile Command, Redstone Arsenal, Alabama, June 1975, Report No. RE-75-35.

17. Peled, A. and Liu, B., Digital Signal Processing, New York: John Wiley and Sons, 1976.

18. "ATMAC FFT Program," RCA, Camden, New Jersey, 1977.

Appendix A. 8080A PROCESSOR BENCHMARK CONFIGURATIONS

The equation solved by the FIR class of digital filters is

$$y_n = \sum_{k=0}^{K} h_k \, x_{n-k} \qquad \text{(A-1)}$$

where h_k are the filter coefficients and x_n are the samples. From this expression, it is seen that two tables are required: one for storing the coefficients h_k, and one for storing x_{n-k}. The table for the coefficients could be ROM or RAM; however, the table for x_{n-k} is RAM and as will be shown later, preferably a K level circulating FIFO. If RAM is used, then this FIFO is implemented by software.

The coefficients are stored as 16-bit two-complement numbers. The samples are 8-bit two-complement numbers. Hence, the need for a 16- by 8-bit multiplier. This multiplier will be implemented presently by software.

The algorithm necessary for the computation of y_n is straightforward. A flowchart is given in Figure A-1. The software multiply algorithm chosen for implementation in the 8080A processor is the well-known Booth's algorithm. While more efficient mechanizations may be known, this algorithm is representative. The following discussion briefly describes Booth's multiplication procedure.

In two-complement form, X can be represented as

$$X = -2^n{}_n + \sum_{m=0}^{n-1} 2^m \, xm$$

and Y can be represented as

$$Y = -2^m y_m + \sum_{j=0}^{m-1} 2^j y_j$$

or

$$Y = (y_{m-1} - y_m)2^m + \sum_{j=0}^{m-1} 2^j (y_{j-1} - y_j) \; y_{-1} = 0 \quad ;$$

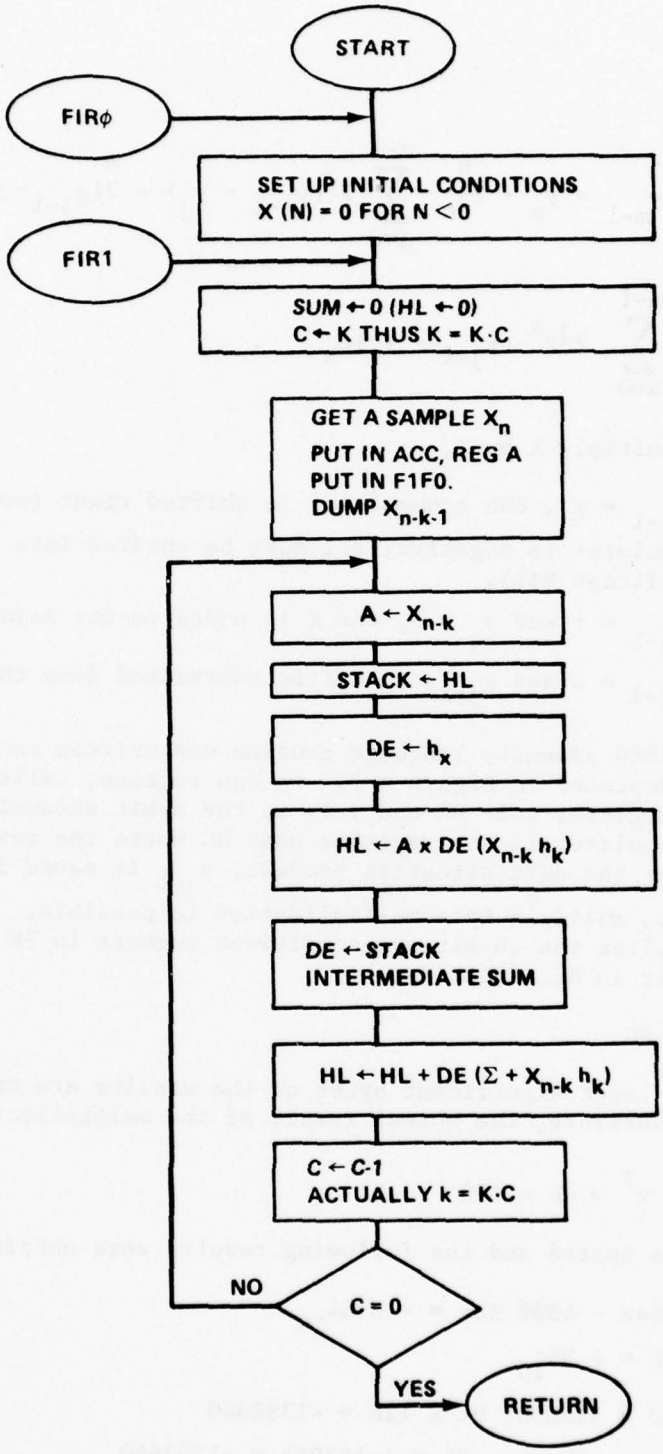


Figure A-1. FIR filter algorithm flow chart.

then,

$$XY = 2x_n^{m+m}(y_{m-1} - y_m - 2x_n^n \sum_{j=0}^{m-1} 2^j(y_{j-1} - y_j) - 2^m(y_{j-1} - y_j) \sum_{k=0}^{n-1} 2^k t_k$$

$$+ \sum_{j=0}^{n-1} \sum_{k=0}^{n-1} 2^j 2^k (y_{j-1} - y_i) n_k \quad .$$

Therefore, to multiply X by Y,

1) If $y_{j-1} = y_j$, the accumulator is shifted right (not if the accumulator is negative; a 1 must be shifted into the most significant bit).

2) If $y_{j-1} = 1$ and $y_j = 0$, the X is added to the accumulator.

3) If $y_{j-1} = 0$ and $y_{j=1}$, then X is subtracted from the accumulator.

An Intel 8080 assembly language routine was written to implement the algorithm depicted in Figure A-2. In one routine, called MULT, x is stored in the register pair DE and y is in the 8-bit accumulator, A. The 16-bit accumulator is the register pair HL where the result is obtained. After the multiplication process, y_{j-1} is saved in the carry bit. Therefore, multiple byte multiplication is possible. The routine, MULT 16, multiplies the 16-bit two-complement numbers in DE and BC and forms the result in HL. That is,

$HL \leftarrow DE \times BC$.

The seven least significant bytes of the results are truncated using MULT. Therefore, the actual result of the multiplication is

$$xy = HL \times 2^7 = HL \times 128 \quad .$$

The routine was tested and the following results were obtained:

$x = C000$ Hex - 4000 Hex = -16384_{10}

$y = 55$ Hex = $+ 85_{10}$

HL = D580H = -2A80H; HL × 128 = -1392640

Check: $85 \times (-16384) = -1392640$

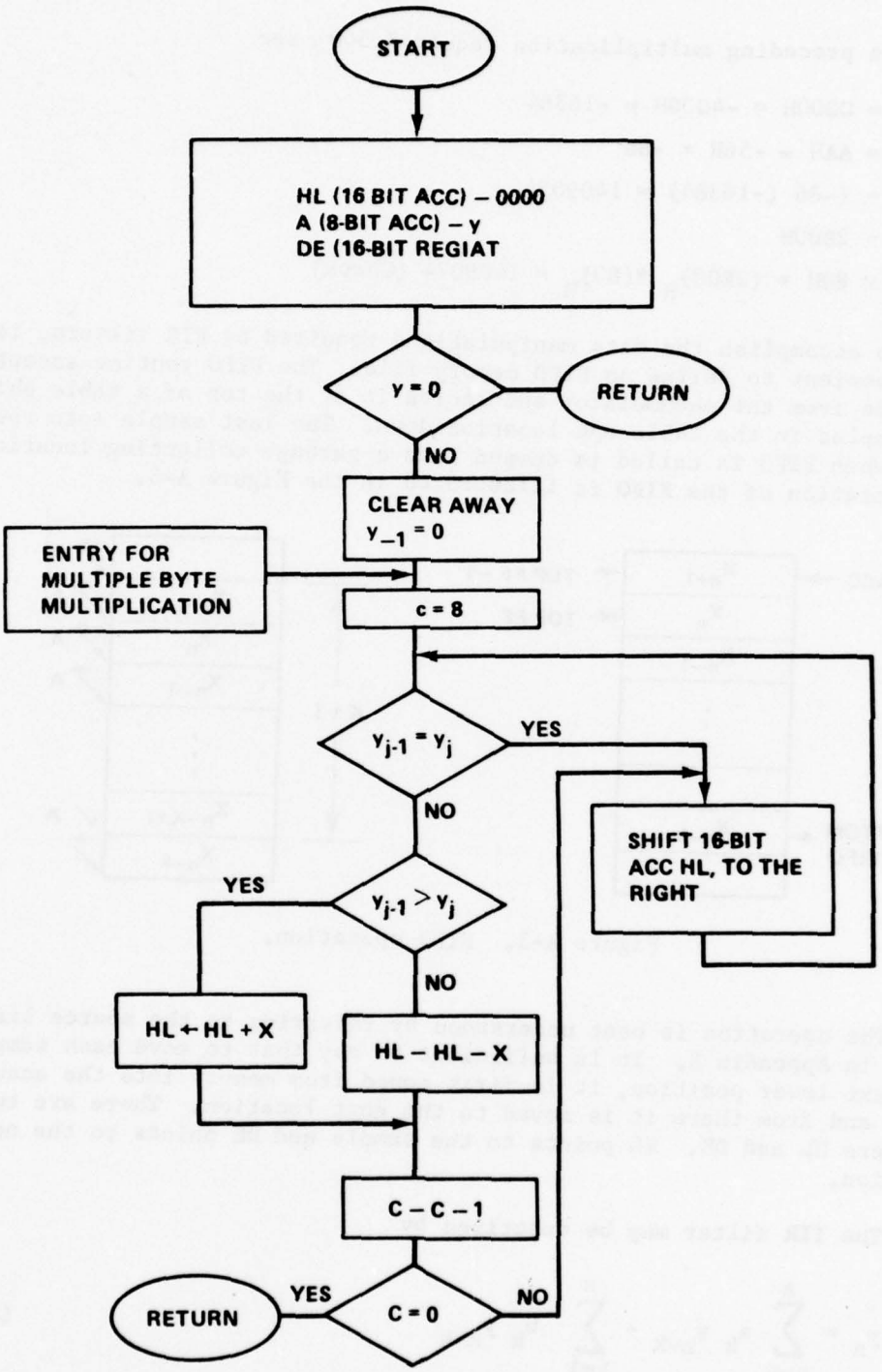


Figure A-2. Multiplication algorithm (16 × 18).

The preceding multiplication required 500 μsec

x = C000H = -4000H = -16384

y = AAH = -56H = -86

xy = (-86 (-16384) = 1409024

HL = 2B00H

HL $\times$ 80H = $(2B00)_H$ *$(80)_H$ = 1409024 (Check)

To accomplish the data manipulations required by FIR filters, it is convenient to define an FIFO memory file. The FIFO routine accepts a sample from the accumulator and stores it at the top of a table shifting all samples in the table one location down. The last sample into the table when FIFO is called is dumped into a garbage collecting location. The operation of the FIFO is illustrated in the Figure A-3.

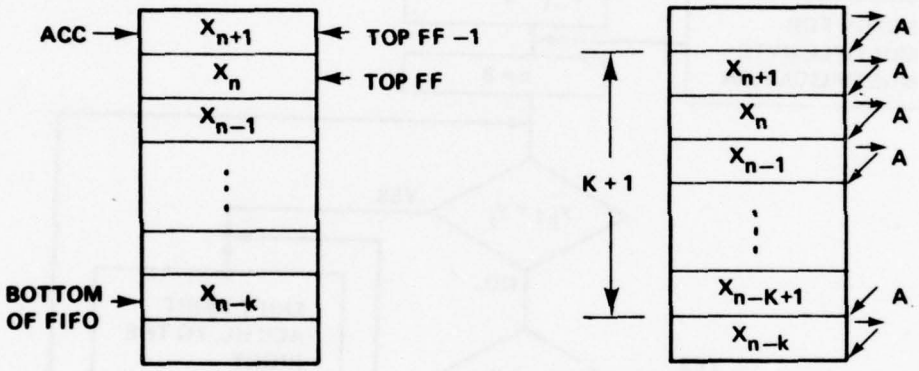


Figure A-3. FIFO operation.

The operation is best understood by referring to the source listing given in Appendix B. It is sufficient to say that to move each sample to the next lower position, it is first moved from memory into the accumulator and from there it is moved to the next location. There are two pointers HL and DE. HL points to the sample and DE points to the next location.

The IIR filter may be described by

$$y_n = \sum_{k=0}^{z} a_k \, x_{m-k} - \sum_{k=1}^{z} b_k \, y_{n-k} \qquad \text{(A-2)}$$

where a_k and b_k are coefficients and x_n are samples.

To evaluate the preceding expression, it is expanded as

$$y_n = a_o X_n + a, X_{n-1} + a_2 X_{n-2} - b, y_{n-1} - b_2 y_{n-2} \qquad \text{(A-3)}$$

where a_o, a_1, a_2, b_1, and b_2 are stored in ROM or RAM as 16-bit two-complement numbers. The results y_n are also 16-bit numbers. x_n are 8-bit two-complement numbers. It is seen that both 16- by 16-bit and 16- by 8-bit multiplication is required. Also, care must be taken when adding the elements. Scaling must be taken into account because the results of 16- by 8-bit and 16- by 16-bit multiplications are added. It is assumed that the coefficients are scaled such that the elements can be directly summed.

To store x_n, x_{n-1}, and x_{n-2} three locations called XNØ, XN1, and XN2 are used. When a new sample is given, then the sample in XNØ moves to XN1 and XN1 to XN2 while the new sample is stored in XNØ. The third sample x_{n-3} is dumped. The IIR routine evaluates the expression according to the flowchart given in Figure A-4.

On entry to the IIR after initialization by calling IIR1, BC must contain y_{n-2} and HL, y_{n-1}. This condition is true when the IIR routine returns to the calling program. Therefore, care must be taken not to destroy these registers.

1. PROCESSING TIME

The processing time of each routine is given in terms of clock cycles. For the Intel 8080A standard package, a typical clock cycle is 500 nsec. (K = K' + 1) where K' = order of filter.

Routine	Clock Cycles
MULT	MIN = 928 MAX = 1143
FIFO	88 + 42 K
FIRØ	171 + 195 K + MULT × K where MULT 114Ø
FIR1	150 + 170 K + MULT × K
IIRØ	502 + 2 × MULT 16 + 3 × MULT where MULT 16 ≈ 2300, MULT ≈ 114Ø
IIR1	427 + 2 × MULT 16 + 3 × MULT
FIR W/AMD 9511 W/O DMA	171 + 388 K

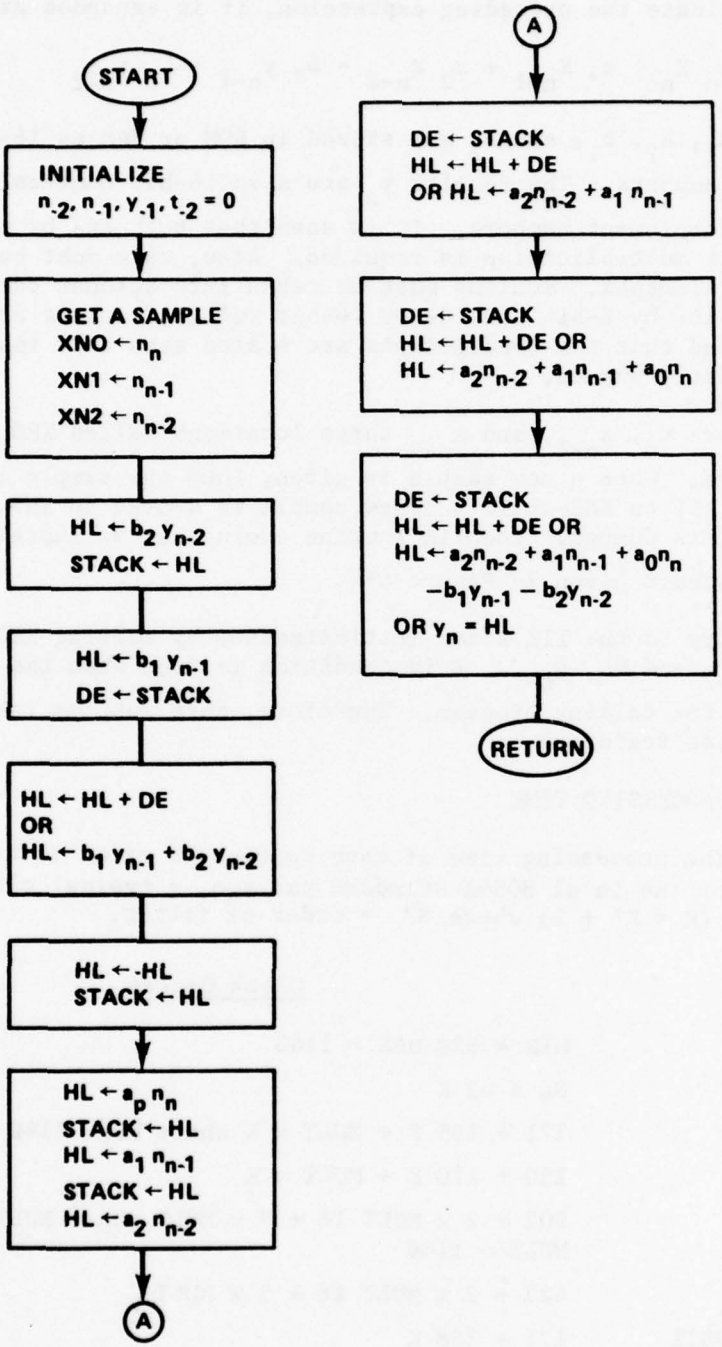


Figure A-4. IIR filter algorithm.

Routine	Clock Cycles
FIR W/AMD 9511	AM9511 requires 92 clk cycle multiply + 101 cycles load
W/DMA	400 + 345 K + FIFO
IIR W/AMD9511	1238 cycles
W/O DMA	(Total)

2. FIR, IIR FILTER TIMING

The following are the times in cycles required to process each sample for K^{th} order FIR and IIR filters.

a. FIR Filter

$$T = 55 + FIFO + (135 + MULT)\ K \qquad (A\text{-}4)$$

where FIFO is a software first-in first-out routine. (FIFO = 88 + 42K). MULT is the multiplication time. The 8-bit by 16-bit multiplication time depends on whether it is implemented in software or hardware. The following table gives the multiplication time in cycles for the configuration listed.

Configuration	MULT Cycles
Software 8- by 16-bit	1140 average
Memory mapped hardware with hardware multiplication of (11 cycles or 5.5 μsec)	58
9511 APU Memory mapped-no interrupt	172

b. IIR Filter

$$T = 82 + [761 + 5*\ MULT\ 16] \qquad (A\text{-}5)$$

where MULT 16 is a 16- by 16-bit multiplication

Configuration	Cycles
Software (16- × 16-bit)	2300
Hardware memory mapped	58
APU memory mapped-no interrupt	172

3. 16- × 16-Bit Multiplication Hardware Multiplication Alternatives

The purpose of this discussion is to outline briefly the hardware requirements of a memory mapped 16- × 16-bit hardware multiplication scheme. In addition, the advantages of memory mapped versus isolated I/O are explained by comparing the software necessary for moving data to and from the multiplication unit for each configuration. Hence, the software required for the operation HL = HL*BC for each configuration are examined first.

a. Isolated I/O

An isolated I/O configuration is considered where data are sent to and received from an I/O device through the accumulator. Figure A-5 shows a block diagram of a 16- by 16-bit hardware multiplication unit. The I/O ports XL, XH, YK, and YH can be either isolated I/O ports or memory locations (memory mapped I/O). The necessary software and the corrsponding cycles per instruction required to perform the operation HL ← HL*BC are given in Figure A-5.

Cycles/Instruction

```
 5     MOV     A,C     ;
10     OUT     XL      ; XL = C
 5     MOV     A,B     ;
10     OUT     XH      ;
 5     MOV     A,L     ; XL = B
10     OUT     YL      ;
 5     MOV     A,H     ; YL = L
10     OUT     YH      ;
10     IN      YL      ; Y4 = H
 5     MOV     L,A     ; Y = Y*X + HL *BC
10     IN      YH      ; HL = Y
 5     MOV     H,A     ; HL = HL*BC
```

In the preceding routine, XL is the address of the lower 8-bit latch of X and XH is the address of the higher 8-bit latch of X. The same applies to Y2 and YH. Assuming that the hardware multiplier has a multiplication time of less than eight cycles (this is the time in cycles, between which YH is loaded with H by the instruction OUT YH and when Y_L must be put on the data bus during the instruction IN YL), the program requires 90 cycles of execution. In the light of the preceding discussion, memory mapped I/O is examined next.

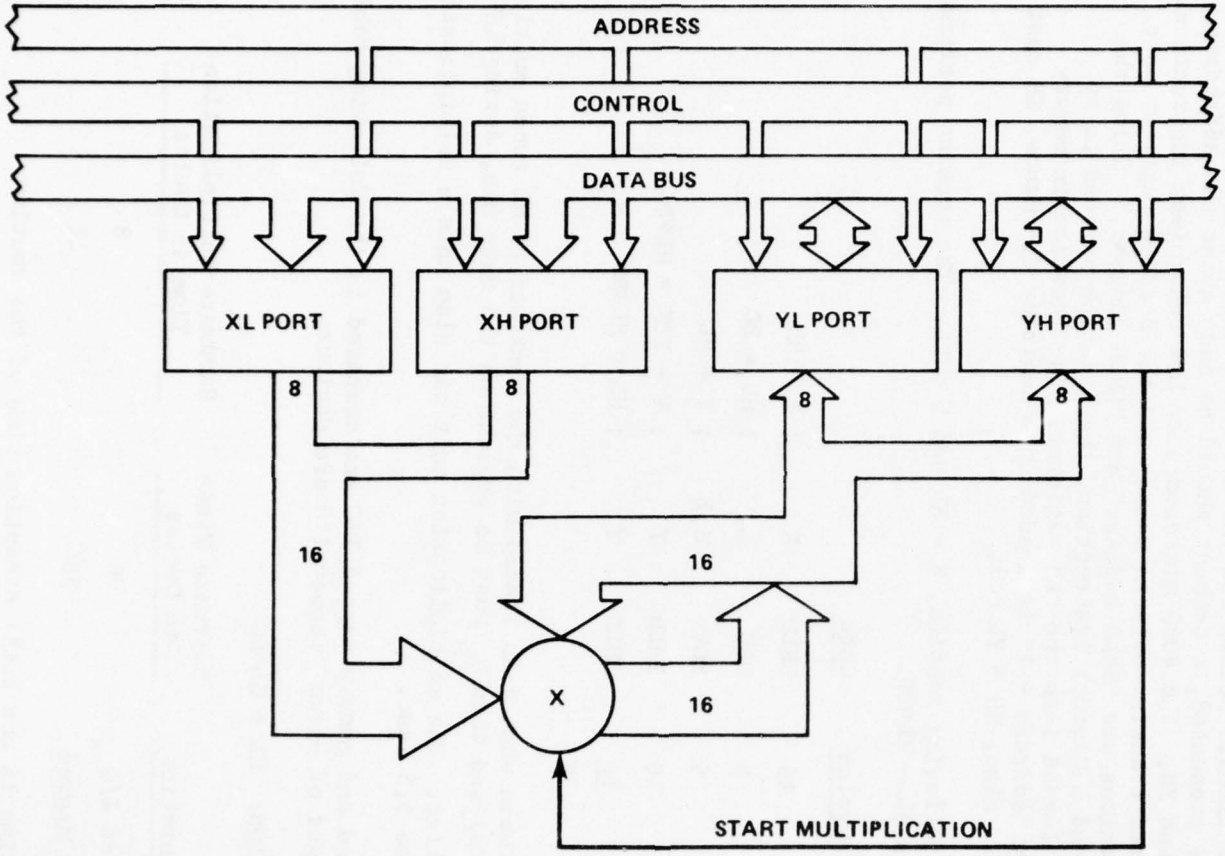


Figure A-5. I/O configuration for isolated or memory mapped hardware multiplication.

b. Memory Mapped I/O

In this case XL, XH, YL, and YH are actually locations in memory. In storing data to and from these locations, they are treated as memory locations. Before the software necessary for performing HL = HL*BC is presented, a comment should be made about the addresses XL, XH, YL, and YH. The 8080 microcomputer has convenient instructions for 16-bit data transfer between memory and the H and L register pair. These instructions are "SHLD address" and "LHLD address." Using the LHLD (Load Hand L Direct) instruction, the location pointed to by "address" is loaded into the "L" register. The location in memory pointed to by "address + 1" is loaded into Register H. Hence, XH must equal XL + 1. Also, YH = YL + 1.

In the following routine, X = XL and Y = YL. The routine performs the operation HL ← HL*BC.

```
Cycles    Code
  16      SHLD    X       ; X = HL
   5      MOV     L,C     ; HL = BC
   5      MOV     H,B     ; Y = HL
  16      SHLD    Y       ; Y = Y*X = HL*BC
  16      LHLD    Y       ; HL = HL*BC
  58
```

The time between when Y is loaded with the contents of HL (when multiplication starts) and when Y_L must be present on the data bus, during LHLD Y, is 11 cycles. The multiplication unit can also have a multiplication time of up to 5.5 μsec.

Isolated and memory mapped I/O are compared in the following table. The advantages of memory mapped I/O are obvious:

OPERATION: HL = HL*BC

Configuration	Software Time* in Cycles	Hardware Multiplication Time in Cycles
Isolated I/O	90	8
Memory Mapped	58	11

*Software time is the total execution time of the routine.

c. Hardware Implementation for Memory Mapped I/O

Figure A-6 presents a basic circuit necessary for storing the contents of the H and L registers into the latches (or shift registers) X_L and X_H using the "SALD X" instruction. The address X is actually the address of X_L and X + 1 is the address of XH. For complete grouping of the hardware requirements of output and input operations, reference is made to pages 3-8 and 3-9 of the INTEL 8080 Microcomputer Users Manual for information of memory mapped I/O, and to pages 2-16 and 2-17 for a complete description of the cycles necessary for the execution of the SHLD and LHLD instructions.

Figure A-7 shows the circuit for input and output to locations Y_L and Y_H. In these circuits, the address X is defined as A15 = 1, A7 = 1, AØ = Ø. The rest of the address bits are "don't cares" for $X_H = X + 1$; A15 = 1, A7 = 1, and AØ = 1.

The address of Y is defined as A_{15} = 1, A7 = Ø, AØ = Ø. The remaining address lines are "don't cares" (assuming that no other memory mapped I/O devices are present).

4. MEMORY MAPPED VERSUS ISOLATED I/O MULTIPLICATION USING THE Am 9511 APU

a. Introduction

The following is a comparison between memory mapped versus isolated I/O configurations of the AM 9511 APU in performing the following operations:

1) Two-complement 8- by 16-bit multiplication. That is, HL = DE*A.

2) Two-complement 16- by 16-bit multiplication. That is, HL = HL*BC.

b. Hardware Configuration:

Figure A-8 shows the hardware configuration of a memory mapped APU unit. Figure A-9 shows the configuration for an isolated I/O configuration.

c. Memory Mapped I/O Software, Operation HL = B * HL

The 8-bit two-complement number in A is multiplied by the 16-bit two-complement number in HL. The result is placed in HL.

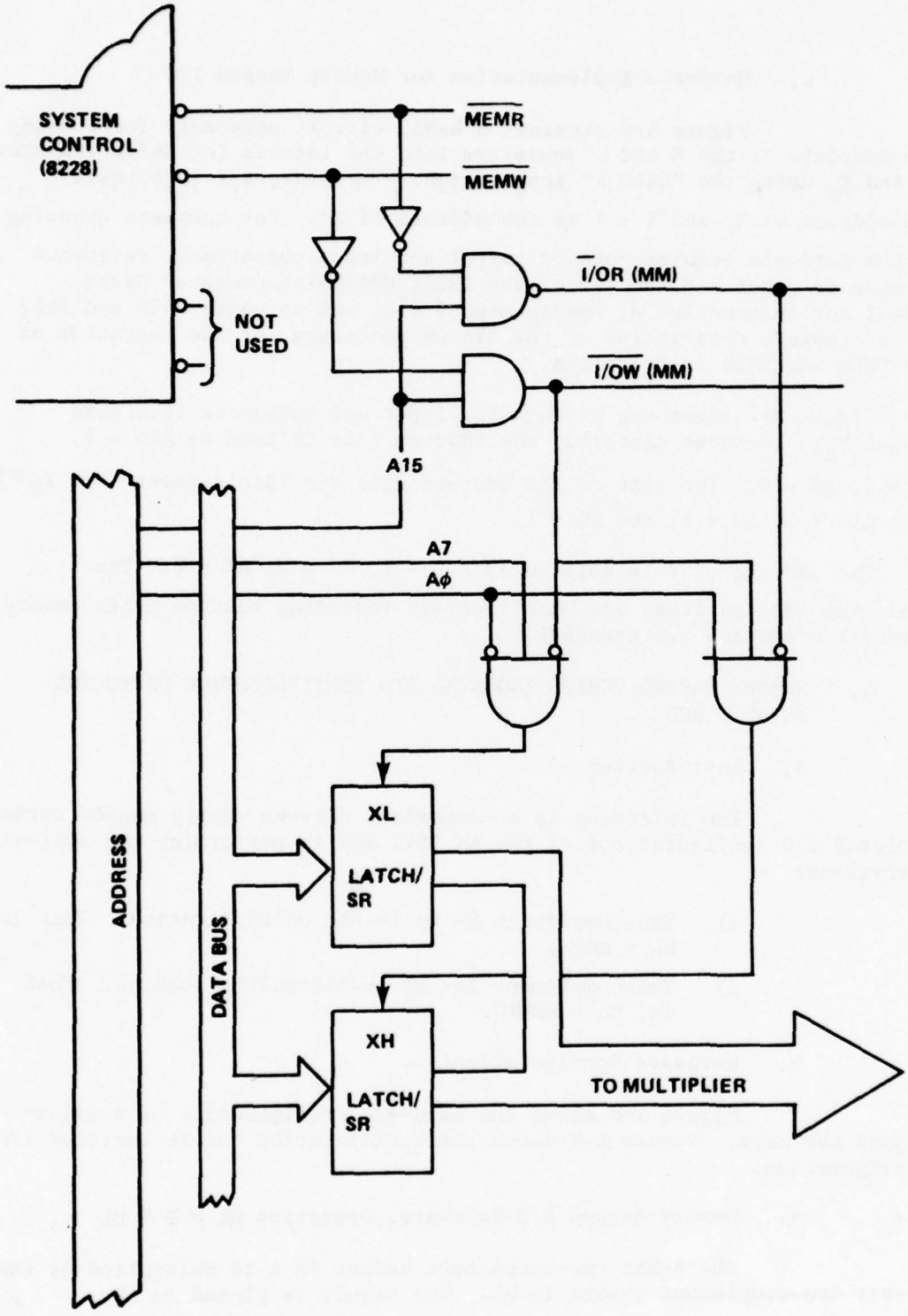


Figure A-6. Memory mapped hardware configuration for latching H and L from the data bus into the XL and XH latches.

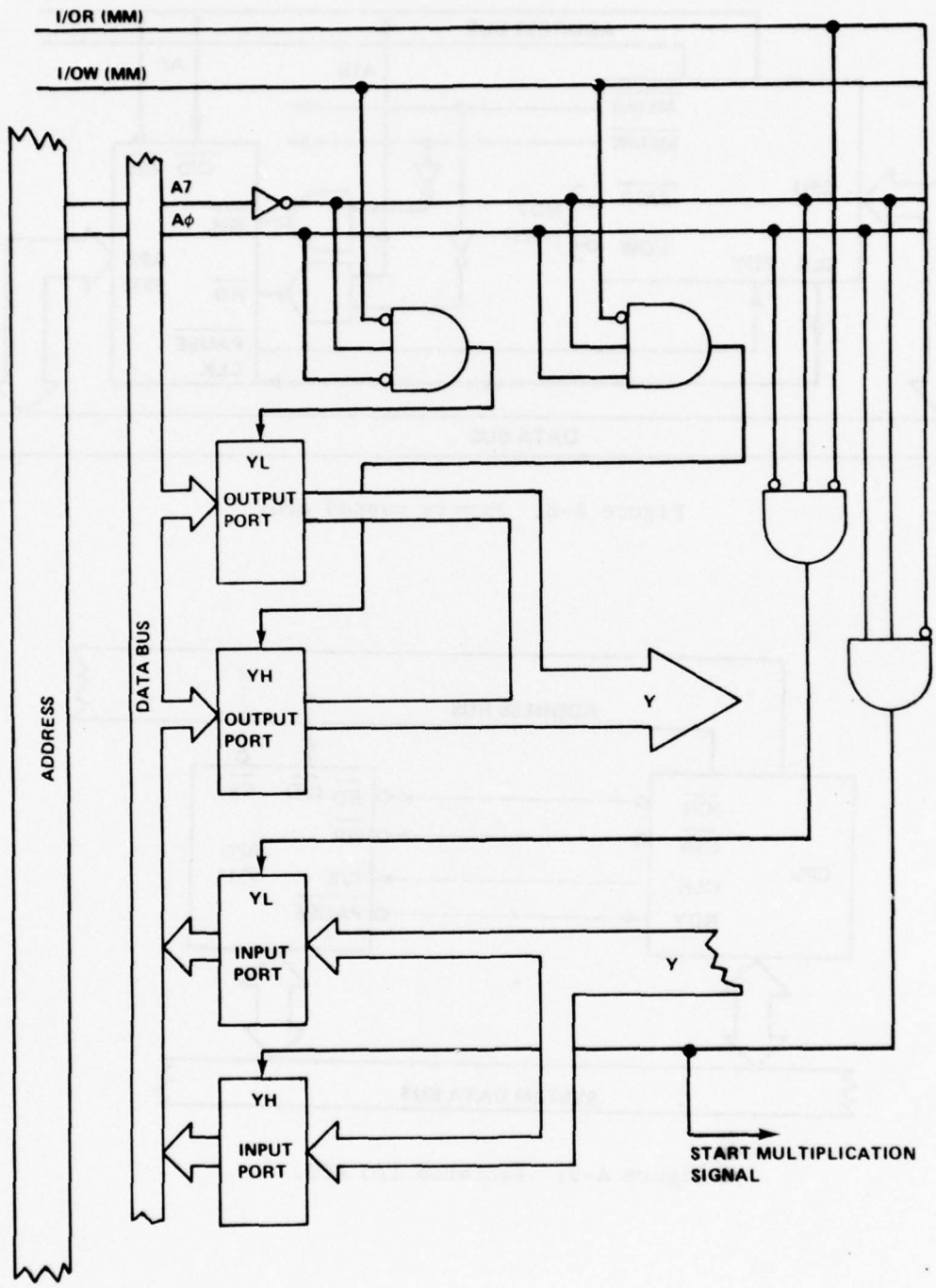


Figure A-7. Hardware configuration for latching H and L and loading Y_H and Y_L into H and L using the instructions SHLD Y, LHLD Y.

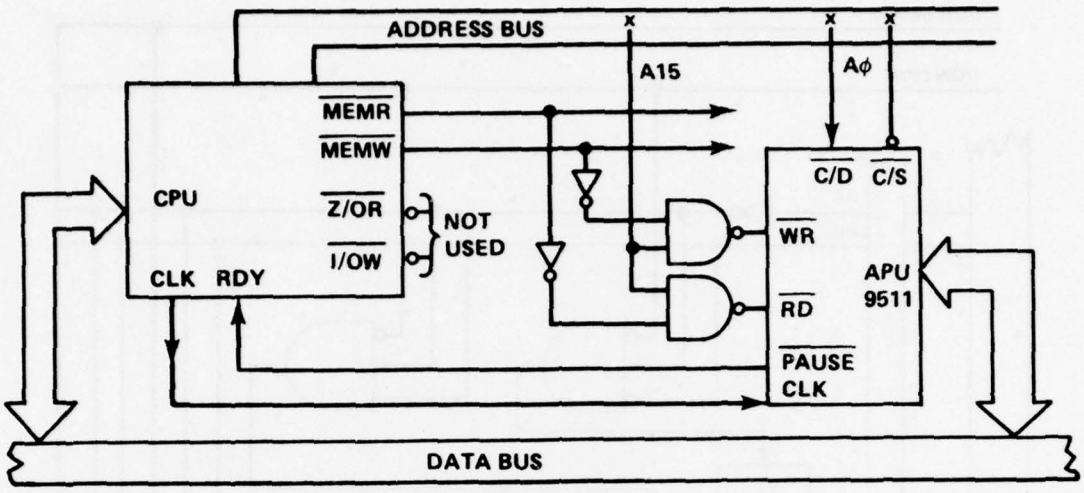


Figure A-8. Memory mapped APU.

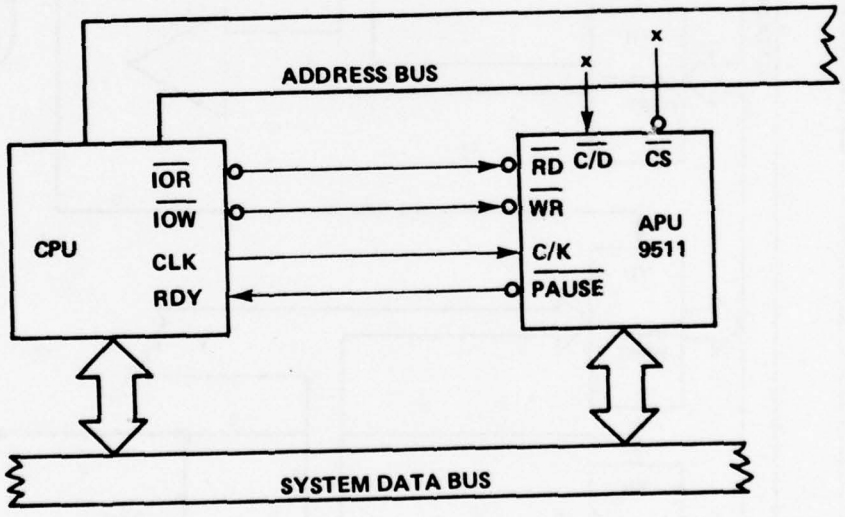


Figure A-9. Isolated I/O APU.

```
Cycles

 10        LXI D, APUAD      ; LOAD DE WITH APU ADDRESSES
  4        XCHG              ; DE HL or HL = APUAD
  7        MOV M, E          ;
  7        MOV M,D           ; TOS = DE
  7        MOV M,B           ; TOS = B, NOX = DE
  7        MVI M,Ø           ;
  5        INX H             ; POINT TO COMMAND ADDRESS
  7 (96)   MVI M, SMUL       ; STORE MULTIPLY COMMAND
  5        DCX H             ; POINT BACK TO DATA ADDRESS
  7        MOV E,M           ; DE = DE*B
  7        MOV D,M           ;
  4        XCHG              ; HL = HL*B
```

Total number of cycles = 169 (77 program, 92 MULT). When APUAD + 1 is put in the address bus, total number of cycles = 164 (72 program, 92 MULT). This method has less code and also is 5 cycles faster than method No. 1. APUAD, the address that sets CID low, and APUCM, the address that sets CID high, cannot be consecutive.

d. Isolated I/O Software Operation HL = HL*B

```
Cycles

  5          MOVE A,B        ; MOVE B into A
 10          OUT APUAT       ;
  4          XRA A           ;
 10          OUT APUDAT      ; Clear A
  5          MOV A,L         ; TOS = HL, NOS = B (16-bit)
 10          OUT APUDAT      ;
  7          MVI A, SMUL     ; MOVE MULT COMMAND INTO A
 10 (92)     OUT APUCM       ; SEND IT TO APU, CID = HIGH
 10          IN APUDAT       ; MOVE TOS TO HL
  5          MOV H,A         ; REGISTERS C/D = LOW
 10          IN APUDAT       ;
  5          MOV L,A         ; HL = HL*B
```

Total number of cycles = 190 (106 program, 92 MULT).

Operation HL = HL*BC

The software is the same as for Operation HL = HL*B except that "XRA A" which was used to clear the accumulator is replaced by "MOV A,C" so that TOS = BC. Hence, one cycle is added to the previous program. Thus, for HL = HL*BC, total cycle time equals 199 (107 program, 92 MULT). The CID line must go high indicating that the data on the data bus are a command (multiplication in this case). When APUAD is put on the address bus, CID must be low.

Operation HL = HL*BC

The two-complement 16-bit numbers in HL and BC are multiplied. The result is placed in HL in the Memory Mapped Configuration.

Cycles	Method No. 1	
10	LXI D, APUAD	; LOAD DE WITH APU ADDRESS
4	XCHG	; DE HL
7	MOV M,E	; MOVE DE (OLD HL) INTO APU
7	MOV M,D	; STACK
7	MOV M,C	; MOVE BC INTO APU STACK
7	MOV M,B	; BC = TOS, DE = NOX
5	INX H	; CID = HIGH
7	MVI M,SMUL	; SEND MULTIPLICATION COMMAND
5	DCX H	; CID = LOW
7	MOV E,M	;
7	MOV D,M	;
4	XCHG	; DE HL or HL = HL*BC

Total Number of Cycles = 169 (77 program, 92 MULT).

Cycles	Method No. 2	
10	LXI SP,APUDAT	; LOAD STACK POINTER WITH
11	PUSH H	; APU DATA ADDRESS
11	PUSH B	; HL = TOS
2	MVI A,SMUL	; BC = TOS, HL = NOS
13	STA APUCM	; TOS = NOS*TOS = HL*BC
10	POP H	; HL = TOS REVERSED
5	MOV A,L	; L H
5	MOV L,H	;
5	MOV H,A	; HL = HL*BC

4. SUMMARY

The results obtained for each operation using memory mapped I/O, or isolated I/O are tabulated with the cycle time and necessary memory storage for each method, as follows:

Operation	Configuration	No. of Bytes Storage Required	Cycles
HL = HL*B	Memory Mapped, Method No. 1	16	169
HL = HL*B	Memory Mapped, Method No. 2	16	170
HL = HL*B	Isolated I/O	22	198
HL = HL*BC	Memory Mapped, Method No. 1	15	169
HL = HL*BC	Memory Mapped, Method No. 2	14	164
HL = HL*BC	Isolated I/O	22	199

Because it takes 11 + 10 = 21 (RST + RET) cycles just to service an interrupt without performing any operation, using the multiplication feature of the APV with 92 cycles execution time with an interrupt is unreasonable.

Appendix B. 8080A MICROPROCESSOR BENCHMARK PROGRAM LISTINGS

FIR DIGITAL FILTER

```
;       ********************************************
;       *
;       *  THE FOLLOWING PROGRAM ILLUSTRATES HOW RESULTS
;       *  ARE OBTAINED FROM THE FIR FILTER
;       *  SAMCNT INITIALLY IS THE NUMBER OF SAMPLES
;       *  SCNT IS WHERE THE SAMPLE COUNT IS STORED
;       *  K IS THE NUMBER OF ELEMENTS SUMMED
;       *
;       ********************************************
SCNT    EQU     0380H
SAMCNT  EQU     10
K       EQU     7
        ORG     0200H
        LXI     SP,0800H; INITIALIZE STACK POINTER
        LXI     H,XTABL ; INITIALIZE POINTER
        SHLD    XPNTR   ; STORE X(N) TABLE ADDR. IN POINTER
        LXI     H,YTABL ; INITIALIZE POINTER
        SHLD    YPNTR   ; STORE Y(N) TABLE ADDR. IN POINTER
        MVI     A,SAMCNT-1  ; INITIALIZE SAMPLE COUNT
        STA     SCNT    ; SAVE COUNT IN LOC. SCNT
        CALL    FIR0    ; INITIALIZE
FILTER: CALL    FIR1
        LDA     SCNT    ; CHECK SAMPLE COUNT
        DCR     A       ;
        STA     SCNT    ; SAVE SCNT; SAMPLE COUNT
        JNZ     FILTER  ;
        JMP     3800H   ; JUMP TO MONITER IF THROUGH
;       ********************************************
;       *                                          *
;       *          T A B L E S                     *
;       *                                          *
;       ********************************************
        ORG     0300H
;     COEFFICIENT TABLE ...H(K)
;   COEF IS THE PAGE WHERE THE COEFF. ARE STORED
COEF    EQU     03H
COEFF:  DW      0FF45H
        DW      0FE7DH
        DW      0FF76H
        DW       0798H
        DW      0FF76H
        DW      0FE7DH
        DW      0FF45H
        DW      000000
;     FIFO STORAGE
IOPFF:  DS      15
ENDFF:  DS      1
;     X TABLE WHERE X SAMPLES ARE STORED
XPNTR:  DS      2
XTABL:  DB      100
        DB      0
        DB      0
        DB      0
        DB      0
        DB      0
        DB      0
        DB      0
        DB      0
;     Y TABLE WHERE Y SAMPLES ARE STORED
YPNTR:  DS      2
YTABL:  DS          20
        ORG     3400H
```

```
;       ********************************************
;*      *                                          *
;*      *            F I F O   R O U T I N E       *
;*      *                                          *
;       ********************************************
;
ENDFF   EQU     TOPFF+K-1
FIFO:   STA     TOPFF-1 ; TOPFF IS THE ADDRESS OF THE FIFO
        LXI     H,ENDFF ; ENDFF IS THE ADDRESS OF THE BOTTOM
        LXI     D,ENDFF+1 ; OF THE FIFO.
        MVI     C,K+1   ; K IS NUMBER OF ELEMENTS
MOVDWN: MOV     A,M     ; EACH LOCATION IS MOVED DOWN
        STAX    D       ;TO THE NEXT LOCATION STARTING FROM
        DCR     C       ; THE BOTTOM OF THE FIFO. HENCE
        RZ              ; X(N) IS MOVED TO THE TOP AND
        DCX     H       ; X(N-K-1) IS DUMPED FROM THE
        DCX     D       ; BOTTOM
        JMP     MOVDWN  ;
;       ********************************************
;       *                                          *
;       *     M U L T I P L Y   R O U T I N E      *
;       *                                          *
;       ********************************************
;   THE FOLLOWING PROGRAM MULTIPIES THE 8 BIT TWO'S COMPLEMENT
;   NUMBER IN ACC BY  THE 16 BIT TWO'S COMPLEMENT NUMBER
;   IN THE REGISTER PAIR DE AND PLACES THE RESULT IN THE
;   REGISTER PAIR HL IN TWO'S COMPLEMENT. REGISTERS BC
;   ARE SAVED
;
MULT:   LXI     H,0000H ; CLEAR H AND L
        ORA     A       ; CLEAR CARRY Y(-1)=0
        RZ              ; RETURN IF MULTIPLYER IS ZERO. HL=0000
        PUSH    B       ; SAVE BC PAIR ON STACK
MUL16:  MVI     C,8     ; INITIALIZE COUNT
        JMP     ENTER   ; START MULTIPLICATIN.ENTRY POINT
                        ; FOR MULTIBYTE MULTIPLICATION.
ADDIT:  DAD     D       ; ADDITION: HL=HL+DE
        JMP     COUNT   ; CHECK COUNT
SUBTR:  MOV     A,L     ; THIS  IS TO SUBTRACT DE FROM HL
        SUB     E       ; HL=HL-DE
        MOV     L,A     ;
        MOV     A,H     ;
        SBB     D       ; SUBTRACT D FROM H WITH BORROW
        MOV     H,A     ;GENERATED FROM L-E.
COUNT:  DCR     C       ; CHECK COUNT
        JNZ     ROTATE  ; IF NOT THROUGH THEN ROTATE HL TO THE RIGHT
        POP     PSW     ; RESTORE ACC AND CARRY
        POP     B       ; RESTORE BC REGISTERS
        RET             ; RETURN TO CALLING PROGRAM
ROTATE: MOV     A,H     ; THIS IS TO ROTATE HL TO THE RIGHT
        ORA     A       ; CHECK H TO SEE IF HL IS NEG.
        JP      POSITV  ; IF SO THEN INSERT A ONE IN MSB WHEN
        STC             ; ROTATING. THAT IS SET CARRY ON NEG. H
POSITV: RAR             ; ROTATE H TO THE RIGHT THROUGH CARRY
        MOV     H,A     ;
        MOV     A,L     ;
        RAR             ; ROTATE L THROUGH CARRY
        MOV     L,A     ;
        POP     PSW     ; RESTORE ACC AND CARRY
ENTER:  JC      CARRY   ; Y(J-1)=1 IF CARRY
        RAR             ; CARRY=Y(J)
        PUSH    PSW     ; SAVE ACC AND CARRY ON STACK
        JC      SUBTR   ; SUBTRACT DE FROM HL IF Y(J)=1 AND Y(J-1)=0
        JMP     COUNT   ; ROTATE IF Y(J)=Y(J-1)=0
CARRY:  RAR             ; Y(J-1)=1
        PUSH    PSW     ;SAVE ACC AND CARRY,Y(J), ON STACK
        JC      COUNT   ; ROTATE IF Y(J)=Y(J-1)=1
        JMP     ADDIT   ; ADD DE TO HL IF Y(J)=0, Y(J-1)=1
```

```
;       ********************************************
;       *                                          *
;       *  THIS ROUTINE MULTIPLIES TWO 16-BIT      *
;       *  TWO'S COMPLEMENT NUMBERS IN BC AND DE   *
;       *  AND PLACES THE RESULT IN HL.            *
;       *                                          *
;       ********************************************
MULT16: MOV     A,C
        CALL    MULT
        MOV     A,B
        CALL    MUL16
        RET
;         Y(N)  O U T P U T          **********
OUTPUT: XCHG                ; NOW DE CONTAINS Y(N)
        LHLD    YPNTR       ; LOAD HL WITH TABLE POINTER
        MOV     M,E         ; MOVE LEAST SIG BYTE OF Y(N) INTO
        INX     H           ; TABLE. POINT TO MOST SIG. BYTE
        MOV     M,D         ; STORE MOST SIG. BYTE INTO MEMORY
        INX     H
        SHLD    YPNTR       ; STORE TABLE POINTER IN YTABL IN MEM.
        XCHG                ; HL NOW CONTAINS Y(N)
        RET
;           A T O D                  *********
ATOD:   LHLD    XPNTR       ; LOAD HL WITH X(N) TABLE POINTER
        MOV     A,M         ; ACC=X(N)
        INX     H           ; POINT TO X(N+1)
        SHLD    XPNTR       ; SAVE POINTER
        RET
;       ********************************************
;       *                                          *
;       *          F I R   D I G I T A L           *
;       *                  F I L T E R             *
;       *                                          *
;       ********************************************
;       *                                          *
;       *  THE COEFF. ARE STORED IN A TABLE        *
;       *  CALLED COEFF AS 16-BIT TWO'S COMPLEMENT *
;       *  NUMBERS. X(N-KK) ARE STORED IN LOCATION *
;       *  FIFO AS 8-BIT TWO'S COMPLEMENT NUMBERS  *
;       *  CALL FIR0 FOR INITIALIZATION.           *
;       *  CALL FIR1 FOR SUBSEQUENT RESULTS        *
;       *                                          *
;       ********************************************
FIR0:   LXI     H, TOPFF    ; HL POINTS TO THE FIFO
        MVI     C,K         ; K IS THE NUMBER OF ADDITIONS IN SUM
        XRA     A           ; CLEAR ACC. THE FOLLOWING INITIALIZES
ZERO:   MOV     M,A         ;THE FIFO SO THAT X(N) FOR
        INX     H           ; N<0 EQUAL TO ZERO
        DCR     C           ; CHECK COUNT
        JNZ     ZERO        ; PUT ZEROES UNTIL END OF FIFO
FIR1:   CALL    ATOD        ; A=X(N)
        CALL    FIFO        ; PUT X(N) ON TOP OF FIFO AND
        LXI     H,00000H    ;DUMP X(N-K-1),CLEAR SUM
        MVI     B,0         ; B CONTAINS LOWER ADDRESS OF COEFF.
        MVI     C,K         ; K IS THE NUMBER OF SAMPLES SUMMED
        LXI     D,TOPFF     ; DE POINTS TO THE TOP OF FIFO
        PUSH    D           ; SAVE DE ON STACK
START:  POP     D           ; RESTORE DE FROM STACK
        LDAX    D           ; ACC=X(N-KK)
        INX     D           ; POINT TO X(N-KK+1)
        PUSH    D           ; SAVE SAMPLE ADDRESS ON STACK
        PUSH    H           ;SAVE INTERMIDIATE SUM ON STACK
        MOV     L,B         ;MOVE COEFF COUNT INTO REG. L
        MVI     H,COEF      ; HL NOW POINTS TO COEFF(0,KK)
        MOV     E,M         ; MOVE H(KK) INTO DE REGISTER PAIR
        INR     L           ; POINT TO MOST SIG BYTE OF COEFF.
        MOV     D,M         ; MOVE IT INTO D
        INR     L           ; POINT TO NEXT COEFF.
        MOV     B,L         ; SAVE COEFF. COUNT IN REG. B
                            ; MULT MULTIPLIES DE BY ACC PUTS
                            ; RESULT IN HL. HL=ACC*DE
        CALL    MULT        ; HL=X(N-KK)*H(KK)
        POP     D           ;PUT INTERMIDIATE SUM FROM STACK
        DAD     D           ; INTO DE REGISTERS AND ADD TO HL
        DCR     C           ; HL=SUM+X(N-KK)*H(KK), DECREMENT COUNT
        JNZ     START       ; CHECK IF K ELEMENTS SUMMED
        POP     D           ; RETURN ADDRESS NOW ON STACK
        CALL    OUTPUT      ; STORE Y(N) IN TABLE
        RET
        END
```

IIR DIGITAL FILTER

```
;*****************************************************************
;*                                                               *
;* THE FOLLOWING PROGRAM CALCULATES Y(N), THE RESPONSE OF A      *
;* K-TH ORDER DIGITAL FILTER TO AN INPUT X(N). THE FILTER        *
;* IS IMPLEMENTED IN SECTIONS. EACH SECTION IS A SECOND          *
;* ORDER IIR FILTER THAT CALCULATES THE FOLLWING:                *
;*                                                               *
;* Y(N)=A0*X(N)+A1*X(N-1)+A2*X(N-2)-B1*Y(N-1)-B2*Y(N-2)          *
;*                                                               *
;* X(N),X(N-1),...,Y(N-1),...,A0,A1,A2,B1,B2, ARE ALL            *
;* 16-BIT TWO'S COMPLEMENT NUMBERS.                              *
;*                                                               *
;* SINCE EACH SECTION NEEDS ITS OWN COEFFICIENTS AND             *
;* PREVIOUS VALUES, THE PROGRAM HAS TWO TABLES, THE COEFF.       *
;* TABLE AND THE XY TABLE.IN THE LATTER, THE PREVIOUS            *
;* VALUES  X(N-1),X(N-2),... ARE STORED.THE TABLES ARE           *
;* DIVIDED INTO SECTIONS, EACH SECTION CONTAINING COEFF.'S       *
;* AND PREVIOUS VALUES FOR A PERTICULAR FILTER SECTION.          *
;* THE CONTENTS OF THE XY TABLE ARE MODIFIED AT THE END          *
;* OF EACH SECTION CALCULATION, THAT IS, X(N) BECOMES X(N-1)*    *
;* AND X(N-1) BECOMES X(N-2) ETC. THE XY TABLE IS REFERRED       *
;* TO AS XYTBL. A POINTER CALLED XYPTR POINTS TO THE APPRO-*     *
;* PRIATE SECTION ADDRESS IN THE TABLE AT THE BEGINNING OF       *
;* EACH IIR FILTER SECTION CALCULATION.EACH SECTION IN THE       *
;* TABLE CONTAINS THE FOLLOWING PREVICUS CONDITIONS IN THE       *
;* ORDER INDICATED: Y(N-2),Y(N-1),X(N-2),X(N-1).                 *
;* THE COEFFICIENTS FOR EACH SECTION ARE STORED IN THE           *
;* FOLLOWING ORDER: B2,B1,A2,A1,A0.                              *
;* A SINGLE ROUTINE IS USED TO CALCULATE THE IIR RESPONSE        *
;* TO AN OUTPUT FROM A PREVIOUS SECTION. THIS ROUTINE            *
;* ASSUMES THAT THE PREVIOUS VALUES AND COEFF.'S FOR THAT        *
;* SECTION HAVE BEEN MOVED FROM THE TABLES TO THE FOLLOWING*     *
;* LOCATIONS: Y(N-2) INTO YNM2, Y(N-1) INTO YNM1, X(N-2)         *
;* INTO XNM2, X(N-1) INTO XNM1 AND X(N) INTO XNM0 AND THE        *
;* COEFF. B2 INTO LOC. B2, COEFF. B1 INTO LOC B1,ETC.            *
;*                                                               *
;* THE OUTPUT FROM EACH SECTION IS STORED IN LOCATION XNM0       *
;* TO BE USED AS INPUT TO THE NEXT SECTION.                      *
;* THE FINAL OUTPUT IS OBTAINED IN THE HL REGISTER PAIR.         *
;*                                                               *
;*                                                               *
;*****************************************************************
IIR:    LXI     H,XYTBL ; INITIALIZE XY TABLE POINTER,POINTING
        SHLD    XYPTR   ; TO PREVEOUS VALUES: X(N-2),...
        LXI     H,CFTBL ; INITIALIZE COEFFICIENT TABLE POINTER
        SHLD    CFPTR   ;
        LHLD    XN      ; HL=X(N) ASSUMING X(N) IN LOC. XN, (A/D MM)
        MVI     A,KD2   ; INITIALIZE SECTION COUNT
SECTN:  STA     SECNT   ;KD2=K/2 IS NUMBER OF SECTIONS, SAVE COUNT
        SHLD    XNM0    ; STORE X(N) INTO LOC. XNM0
;       *                                                       *
;       *                                                       *
;       * PREV MOVES THE PREVIOUS VALUES Y(N-2),Y(N-1),... INTO *
;       * LOCATIONS  YNM2,YNM1,... USED BY IIRS TO CALCULATE Y(N) *
;       *                                                       *
;       *                                                       *
PREV:   LHLD    XYPTR   ;LOAD HL WITH XYPTR WHICH POINTS TO PREVIOUS
        SPHL            ;VALUE SECTION IN XYTBL, MOVE TO STACK POINTR
        MOV     A,L     ;
        ADI     8       ; CALCULATE HL=HL+8
        JNC     NEXT    ; INCREMENT H IF CARRY
        INR     H       ;
NEXT:   MOV     L,A     ; HL=HL+8
        SHLD    XYPTR   ; MODIFY POINTER TO POINT TO NEXT SECTION
                        ; XYPTR=XYPTR+8
        POP     H       ; STACK POINTS TO Y(N-2) SO POP H
        SHLD    YNM2    ; LOADS Y(N-2) INTO H AND L,
        POP     H       ; POPPING AGAIN LOADS Y(N-1) INTO HL,
        SHLD    YNM1    ;STORE Y(N-2),Y(N-1),...AS THEY ARE POPPED
        POP     H       ; INTO LOCATIONS YNM2,YNM1,XNM2,XNM1.
        SHLD    XNM2    ;
        POP     H       ;
        SHLD    XNM1    ;
```

```
;     *                                                               *
;     *                                                               *
;     * CFMOV MOVES THE COEFFICIENTS FOR THE SECTION TO BE            *
;     * CALCULATED FROM CFTBL TO LOCATIONS B2,B1,...                  *
;     *                                                               *
;     *                                                               *
CFMOV:  LHLD    CFPTR   ;MOVE CFPTR WHICH POINTS TO COEFFICIENTS
        SPHL            ;FOR SECTION TO BE CALCULATED TO
        MOV     L,A     ; STACK POINTER
        ADI     10      ; CALCULATE HL=HL+10
        JNC     NEXT2   ; INCREMENT H IF CARRY
        INR     H       ;
NEXT2:  MOV     A,L     ;  HL=HL+10
        SHLD    CFPTR   ;MODIFY POINTER TO POINT TO NEXT SECTION
                        ; CFPTR=CFPTR+10
        POP     H       ; MOVE B2,B1,... INTO LOCATIONS B2,B1,...
        SHLD    B2      ; BY POPPING B2,B1,... INTO HL AND THEN
        POP     H       ; STORING THEM INTO LOCATIONS B2,B1,...
        SHLD    B1      ; USED TO CALCULATE IIR RESPONSE.
        POP     H       ;
        SHLD    A2      ;
        POP     H       ;
        SHLD    A1      ;
        POP     H       ;
        SHLD    A0      ;
;     *                                                               *
;     *                                                               *
;     * THE FOLLOWING CALCULATES Y(N) FOR AN IIR SECTION              *
;     * ACCORDING TO THE FOLLOWING:                                   *
;     *  Y(N)=A0*X(N)+A1*X(N-1)+A2*X(N-2)-B1*Y(N-1)-B2*Y(N-2)         *
;     *Y(N-2),Y(N-U),X(N-2),X(N-1),AND X(N) MUST BE IN LOCATIONS      *
;     *YNM2,YNM1,XNM2,XNM1,AND XNM0. THE COEFFICIENTS MUST BE         *
;     * IN LOCATIONS B2,B1,A2,A1,A0. Y(N) IS CALCULATED AND PUT      *
;     *IN THE H AND L REGISTERS. B2,B1,... ARE DESTROYED.            *
;     *                                                               *
;     *                                                               *
IIRS:   LXI     SP, B2  ; STACK POINTS TO B2 WHICH CONTAINS COEF. B2
        POP     B       ; BC=B2
        LHLD    YNM2    ; HL=Y(N-2)
        CALL    MUL16   ; HL=HL*BC, HL=Y(N-2)*B2
        XCHG            ; DE=HL
        POP     B       ;  BC=B1
        LHLD    YNM1    ; HL=Y(N-1)
        CALL    MUL16   ; HL=Y(N-1)*B1
        XCHG
        DAD     D       ; HL=B1*Y(N-1)+B2*Y(N-2)
        XRA     A       ; CLEAR ACC.
        SUB     L       ;  A=0-L
        MOV     L,A     ;
        MVI     A,0     ; CAREFUL NOT TO DESTROY CARRY
        SBB     H       ;
        MOV     H,A     ; HL=-HL
        XCHG            ; DE=-HL=-B1*Y(N-1)-B2*Y(N-2)
        POP     B       ; BC=A2
        LHLD    XNM2    ;  HL=X(N-2)
        CALL    MUL16   ;HL=X(N-2)*B2
        DAD     D       ; HL=A2*X(N-2)-Y(N-1)*B1-...
        XCHG            ; DE=HL
        POP     B       ; BC=A1
        LHLD    XNM1    ; HL=X(N-1)
        CALL    MUL16   ; HL=A1*X(N-1)
        DAD     D       ; HL=A1*X(N-1)+...-B1*Y(N-1)-...
        POP     B       ; BC=A0
        LHLD    XNM0    ; HL=X(N)
        CALL    MUL16   ; HL=A0*X(N)
        DAD     D       ; HL=Y(N)=A0*X(N)+A1*X(N-1)+...-B1*Y(N-1)-...
;     *                                                               *
```

```
;*****************************************************************
;*                                                               *
;*  TRANS TRANSFERS Y(N), Y(N-1), X(N), X(N-1) TO THE XYTBL      *
;*  LOCATIONS OF THE SECTION JUST CALCULATED TO BE USED AS       *
;*  Y(N-1),Y(N-2),X(N-1), AND X(N-2) THE NEXT TIME AROUND.       *
;*                                                               *
;*****************************************************************
TRANS:  XCHG            ; DE=Y(N)
        LHLD    XYPTR   ; MOVE XY TABLE POINTER WHICH POINTS TO
        SPHL            ;NEXT SECTION TO STACK.
        LHLD    XNM0    ; HL=X(N) IS PUSHED ON THE STACK.
        PUSH    H       ; HENCE IT BECOMES X(N-1) OF THE SECTION
        LHLD    XNM1    ; JUST EVALUATED. LIKEWISE X(N-1) IS
        PUSH    H       ; IS STORED INTO THE X(N-2) LOCATION OF
        PUSH    D       ; THE SECTION IN XYTBL. SO ARE
        LHLD    YNM1    ; Y(N) AND Y(N-1) THAT BECOME Y(N-1),Y(N-2)
        PUSH    H       ;THE NEXT TIME AROUND.
        XCHG            ;HL=DE=Y(N) WHICH IS X(N) FOR NEXT SECTION.
        LDA     SECNT   ; LOAD ACC. WITH SECTION COUNT
        DCR     A       ; DECREMENT SECTION COUNT
        JNZ     SECTN   ; CHECK TO SEE IF LAST SECTION.
                        ; EXIT WITH Y(N) IN H AND L REGISTER PAIR
        END
/*
```

FFT ALGORITHM

```
;*****************************************************************
;*                                                               *
;*  THE FOLLOWING IS AN 8080 PROGRAM       FOR DECIMATION-IN-TIME*
;*  RADIX 2, IN-PLACE FAST FOURIER TRANSFORM.                    *
;*  THE SAMPLES ARE STORED IN A TABLE CALLED XTABL. EACH SAMPLE  *
;*  IS STORED AS A COMPLEX NUMBER WITH 16-BIT TWO'S COMPLEMENT   *
;*  REAL AND IMAGINARY PARTS.                                    *
;*  REFERENCE IS MADE TO THE FORTRAN DECIMATION-IN-TIME, RADIX 2,*
;*  IN-PLACE FFT ROUTINE BY COOLY,LEWIS, AND WELCH(PAGE 367,     *
;*  RABINEU AND GOLD'S "DIGITAL SIGNAL PROCESSING"),             *
;*                                                               *
;*                                                               *
;*  THE MAXIMUM NUMBER OF SAMPLES,N, IS NMAX=256.                *
;*                                                               *
;*                                                               *
;*****************************************************************
;*                                                               *
;*  IN-PLACE BIT-REVERSAL SHUFFLING ROUTINE.                     *
;*  SUBROUTINE MOVE IS USED TO MOVE THE REAL AND IMAGINARY PART  *
;*  OF THE COMLEX VARIABLE ADDRESSED BY HL INTO THE LOCATION     *
;*  ADDRESSED BY DE.                                             *
;*                                                               *
;*                                                               *
MOVE:   MVI     C,4       ; INITIALIZE COUNT IN C REGISTER
;*                                                               *
;*                                                               *
MOVE1:  MOV     A,M       ; MOVE FIRST BYTE OF REAL PART INTO
        STAX    D         ; ACCUMULATOR AND FROM THERE STORE THEM
        INX     H         ; INTO LOCATION ADDRESSED BY DE.
        INX     D         ; INCREMENT HL AND DE.
        DCR     C         ; DO FOR FOUR BYTES.
        JNZ     MOVE1     ; CHECK TO SEE IF FOUR BYTES MOVED.
        RET               ;
;*                                                               *
;*                                                               *
;*  THE FOLLOWING DOES THE IN-PLACE BIT-REVERSED SHUFFLING.      *
;*  REGISTER B CONTAINS THE BIT-REVERSED CODE OF THE COUNT PLACED*
;*  IN REGISTER E.N IS THE NUMBER OF SAMPLES WHICH IS PLACED     *
;*  IN REGISTER D.                                               *
;*                                                               *
;*                                                               *
INPBRS: LXI     H,XTABLE  ; HL POINTS TO TABLE WITH SAMPLES X.
        MVI     D,N       ; MOVE NUMBER OF SAMPLES,N, INTO REG. D.
        XRA     A         ; CLEAR ACC.
        MOV     E,A       ; CLEAR REG. E.
                          ; E WILL CONTAIN COUNT AS WE MOVE
                          ; DOWN THE XTABL CONTAINING SAMPLES.
        MOV     B,A       ; CLEAR B.
                          ; B WILL CONTAIN BIT REVERSED NUMBER OF D.
SHFFL:  SUB     E         ; A=B-E. IF B>E THEN REVERSE POSITIONS.
        JZ      NONEED    ; IF B=E THEN NO NEED TO REVERSE.
        JC      NONEED    ; IF E>B THEN ALREADY REVERSED BEFORE.
REVERSE: PUSH   D         ; SAVE DE REGISTERS.
        PUSH    H         ; SAVE ADDRESSS OF CURRENT SAMPLE ADDRESS
        PUSH    H         ; SAVE TWO LEVELS DEEP ON STACK.
        LXI     D,TEMP    ; LOAD DE WITH ADD. OF TEMPORARY LOC.
        CALL    MOVE      ; MOVE SAMPLE INTO TEMP LOCATION.
        MVI     H,0       ; SET HL=A
        MOV     L,A       ;
        DAD     H         ;  HL=2*A
        DAD     H         ;  HL=4*A
        DAD     D         ; HL=CURRENT SAMPLE ADD.+ 4*A.
        PUSH    H         ; HL NOW CONTAINS ADDRESSS OF LOC. TO BE
                          ; SHUFFLED IN-PLACE. SAVE HL ON STACK
        CALL    MOVE      ; MOVE SAMPLE IN LOC. ADDRESSED BY HL TO
        POP     H         ; LOCATION IN DE( CURRENT SAMPLE)
        LXI     D,TEMPR   ; LOAD DE WITH ADDRESS OF TEMP LOC.
        XCHG              ; SWITCH DE AND HL
        CALL    MOVE      ; MOVE SAMPLE IN LOC. TEMP INTO
                          ; LOCATION OBTAINED BY BIT-REVERSAL.
        POP     H         ; RESTORE ADDRESS OF CURRENT SAMPLE.
        POP     D         ; RESTORE DE REGISTERS.
NONEED: INR     E         ; INCREMENT COUNTER E TO POINT TO NEXT SAMPLE
        CALL BITREV       ; BIT-REVERSE E. RETURN WITH BIT-REVERSED
        MOV     B,A       ; NUMBER IN ACC. PUT IT INTO B.
        INX     H         ;
        INX     H         ;
        INX     H         ;
        INX     H         ; HL=HL+4,HL NOW POINTS TO NEXT SAMPLE IN XTABL.
        DCR     D         ; CHECK TO SEE IF N SAMPLES SHUFFLED.
        JNZ     SHFFL     ; KEEP ON SHUFFELING.
```

```
;     *                                                                 *
;     * THE FOLLOWING ROUTINE OBTAINS THE NEXT BIT-REVERSED NUMBER      *
;     * FROM THE PREVIOUS ONE IN THE B REGISTER. THE METHOD USED IS     *
;     * BASED ON THE FLOW-CHART ON PAGE 365 OF RABINER AND GOLD'S       *
;     * " DIGITAL SIGNAL PROCESSING".ON EXIT THE BIT-REVERSED NUMBER    *
;     * IS IN THE ACCUMULATOR.                                          *
;     *                                                                 *
;
BITREV: MOV     A,B       ; MOVE PREVIOUS BIT-REVERSED NUMBER
        MVI     C,9-LN    ; INTO ACC. THE BIT-REVERSED NUMBER MUST BE
LROTN:  DCR     C         ; NORMALIZED. LN=LOG(BASE TWO)( N )
                          ; HENCE 8-LN ARE THE NUMBER OF TIMES THE
                          ; NUMBER MUST BE SHIFTED TO THE LEFT.
        JZ      START     ; EXIT CONDITION.
        RLC               ; ROTATE ACC. TO THE LEFT
        JMP     LROTN     ;
START:  MVI     C,N/2     ; START BIT-REVERSAL PROCESS
ROTX:   RAL               ;
ROTX:   JC      SUBONE    ;
ROTX:   MOV     A,C       ;
ROTX:   XRA     B         ; A= BIT- REVERSED NUMBER.
ROTX:   MVI     9-LK      ; BEGIN DENORMALIZATION.
RROTN:  DCR     C         ;
RR
        RZ                ; RETURN
        RRC               ; ROTATE ACC. RIGHT.
SUBONE: MOV     D,A       ; A TRICK TO PERFORM:
        MOV     A,C       ; X+(1-2-4...-N/2) BY CALCULATING
        RRC               ;(1-2-4-...-N/2) AND PUTTING IT IN C
        XRA     C         ;
        MOV     C,A       ;
        MOV     D,A       ;
        JMP     ROTX      ;
;     *                                                                 *
;     * THIS PORTION IMPLEMENTS THE EQUAVALENT OF THE FOLLOWING         *
;     * FORTRAN CODE:     DO 20 L=1,M                                   *
;     *                   LE = 2**L                                     *
;     *                   LE1=LE/2                                      *
;     *                   U = (1.0,0.0)                                 *
;     *                   W = CMPLX(COS(PI/LE1),SIN(PI/LE1))            *
;     *                   DO 10 J = 1,LE1                               *
;     *                   DO 20 I = J,N,LE                              *
;     *                   IP = I + LE1                                  *
;     *                   T = A(IP) * U                                 *
;     *                10 A(I) =A(I) + T                                *
;     *                20 U = U*W                                       *
;     * STORAGE LOCATION LL CONTAINS 'L', REGISTERS D AND E CONTAIN     *
;     * 'J' AND 'I' RESPECTIVELY. LE1 IS KEPT IN REGISTER B AND         *
;     * LE IS KEPT IN REGISTER C.                                       *
;     * ALL COMPLEX VALUES HAVE 16-BIT TWO'S COMPLEMENT REAL AND        *
;     * IMAGINARY PARTS.                                                *
;     *                                                                 *
```

```
        MVI     A,1     ; BEGIN DO LOOP WITH 'L'=1
CDE:    STA     LL      ; STORE 'L' IN LOCATION LL
        MOV     C,A     ; SAVE 'L' IN C TO CALCULATE LE LATER
        MVI     A,1     ; THIS PORTION CALCULATES
POWER:  ADD     A       ; A=2**L
        DCR     C
        DCR     C       ; THIS IS DONE BY SHIFTING A=1 TO
        JNZ     POWER   ; THE RIGHT 'L' TIMES.
        MOV     C,A     ; C=2**L=LE
        RRC             ; A=A/2=LE/2=LE1
        MOV     B,A     ; B=LE1
        LXI     H,ONE   ; LOAD HL WITH REAL 1.0
        SHLD    UREAL   ; STORE HL INTO UREAL
        SHLD    UR
        SHLD    ACCR    ; ACCR=1.0
        LXI     H,ZERO  ; LOAD H WITH 0.0
        SHLD    UIMAG   ; UIMAG=0.0
        SHLD    ACCI    ; ACCI=0.0
                        ; U=(1.0,0.0),ACC=(1.0,0.0)
        CALL    WCALC   ; CALCULATE W = CMPLX(COS(PI/LE1),SIN(PI/LE1))
        MVI     A,1     ; INITIALIZE DO LOOP
BCD:    MOV     D,A     ;    D='J'
ABC:    MOV     E,A     ;    E='I'
        ADD     B       ; 'IP'=A='I'+LE1
        CALL BUTTFLY    ; PERFORM BUTTERFLY
        MOV     A,E     ; A='I'
        ADD     C       ; A='I' +LE
        CPI     N       ; COMPARE 'I' WITH N
        JP      SKIP1   ; IF 'I'>N EXIT LOOP
        JMP     ABC     ; STAY OTHERWISE
SKIP1:  CALL    ITER    ; CALCULATE U=U*W
        INR     D       ; 'J'='J'+1
        MOV     A,B     ; A=LE1
        CMP     D       ; COMPARE 'J' WITH LE1
        JM      SKIP2   ; EXIT LOOP IF 'J'>LE1
        JMP     BCD     ; STAY OTHERWISE
SKIP2:  LDA     LL      ; A='L'
        INR     A       ; 'L'='L'+1
        CPI     LMAX    ; COMPARE 'L' WITH 'M'
        RP              ; RETURN IF 'L'>'M'
        JMP     CDE     ; STAY IN LOOP OTHERWISE
;       *                                                       *
;       * THE FOLLOWING MULTIPLIES THE COMPLEX NUMBERS ADDRESSED BY HL *
;       * (CALLED X=A+B*SQRT(-1)) BY Y=AP+BP*SQRT(-1) ADDRESSED BY DE. *
;       * Y IS IN A COMPLEX ACCUMULATOR IN MEMORY, LOCATION ACCR.  *
;       * WE HAVE Y=X*Y=(A*AP-B*BP)+(A*BP+AP*B)*SQRT(-1)          *
;       *                                                       *
```

```
;
CMULT:  PUSH    H       ; SAVE ADDRESS OF X
        MOV     C,M     ; LOAD BC REGISTERS WITH REAL PART
        INX     H       ; OF X.
        MOV     B,M     ;
        INX     H       ;
        XCHG            ; SAVE X IMAG. ADDRESS IN DE
        LHLD    ACCR    ; PUT REAL PART OF Y IN HL. HL=AP
        CALL    MUL16   ; HL=A*AP
        XCHG            ; MOVE A*AP INTO DE AND ADDRESSS OF X IMAG.
        MOV     C,M     ; PART INTO HL. MOVE X IMAG. PART,B, INTO
        INX     H       ; BC REGISTERS. BC=B
        MOV     B,M     ;
        LHLD    ACCI    ; HL=BP (IMAG. PART OF Y)
        CALL    MUL16   ; HL=HL*BC OR HL=B*BP
        MOV     A,E     ;
        SUB     L       ; START SUBTRACTING ; HL=DE-HL
        MOV     L,A     ; THAT IS HL=A*AP-B*BP
        MOV     A,D     ;
        SBB     H       ;
        MOV     H,A     ;
        XCHG            ; START PUTTING REAL PART OF Y IN DE,
        LHLD    ACCR    ; BEFORE STORING COMPUTED REAL PART
        XCHG            ; IN ACCUMULATOR.
        SHLD    ACCR    ; STORE A*AP-B*BP INTO COMPLEX ACC, REAL PART
        POP     H       ; RESTORE X REAL PART ADDRESS
        MOV     C,M     ;
        MOV     B,M     ;
        INX     H       ; MOVE  XREAL PART,A, INTO BC REGISTERS
        INX     H       ; HL NOW POINTS TO IMAG. PART OF X ,B.
        PUSH    B       ; SAVE REAL PART OF X,A, ON STACK
        MOV     C,M     ; MOVE IMAG PART OF X,B, INTO BC REGISTERS,
        INX     H       ;
        MOV     B,M     ;
        XCHG            ; DE , WHICH CONTAINED AP, IS MOVED TO HL
        CALL    MUL16   ; HL=HL*BC, OR HL=AP*B.
        XCHG            ; DE=AP*B.
        LHLD    ACCI    ; HL=BP.
        POP     B       ; RESTORE X REAL PART,A, INTO BC REGISTERS.
        CALL    MUL16   ; HL=HL*BC, OR HL=A*BP
        DAD     D       ; HL=HL+DE, OR HL=A*BP+AP*B.
        SHLD    ACCI    ; ACCUMULATOR=X*Y=(A*AP-B*BP)+(A*BP+AP*B)*I
        RET             ;
;
;  *
;  * THE FOLLOWING ROUTINE CALCULATES:                              *
;  *                          T=A(I)*U                              *
;  *                          A(IP)=A(I)-T                          *
;  *                          A(I)=A(I)+T                           *
;  * FIRST THE ADDRESSES OF A(IP) AND A(I) ARE OBTAINED FROM        *
;  * REGISTERS A=IP AND E=I RESPECTIVLY. BC WILL POINT TO A(I)      *
;  * AND DE WILL POINT TO A(IP).                                    *
;  *
```

```
;
BUTTFLY:
        PUSH    B       ;
        PUSH    D       ;
        PUSH    H       ;
        PUSH    PSW     ;
        MVI     H,0     ; PUT E INTO L AND 0 IN H. HL=E
        MOV     L,E     ;
        LXI     D,XTABL ; LOAD DE WITH XTABL.
        DAD     H       ; HL=2*HL, OR HL=E+E
        DAD     H       ; HL=2*HL, OR HL=4*E
        DAD     D       ; HL=XTABL+4*E
        MOV     B,H     ; MOVE HL INTO BC
        MOV     C,L     ; BC NOW POINTS TO A(I)
        MOV     L,A     ; LOAD HL WITH ACC.
        MVI     H,0     ;    HL=A
        DAD     H       ; HL=2*HL, OR HL=2*A
        DAD     H       ; HL=2*HL, OR HL=4*A
        DAD     D       ; HL=XTABL+4*A
        XCHG            ; DE NOW POINTS TO A(IP)
;
;*******************************************************************
;* DE POINTS TO A(IP), BC POINTS TO A(I), T IS IN THE COMPLEX      *
;* ACCUMULATOR IN MEMORY WHICH IS ADDRESSED BY BC.                 *
;*******************************************************************
;
        PUSH    D       ; SAVE A(IP) ADDRESS ON STACK
        PUSH    B       ;
        LHLD    UR      ;
        SHLD    ACCR    ; MOVE U INTO ACC.
        LHLD    UI      ;
        SHLD    ACCI    ;
        XCHG            ; HL POINTS TO A(IP)
        CALL    CMULT   ; COMPLEX MULTIPLY A(IP) BY U. ACC=A(IP)*U
        POP     B       ; RESTORE BC
        POP     D       ; RESTORE ADDRESS OF A(IP)
        PUSH    B       ; SAVE ADDRESS OF A(I)
        LHLD    ACCR    ; START CALCULATING A(IP)=A(I)-T
        LDAX    B       ; THE REAL PART OF A(I) POINTED TO
        SUB     E       ; BY BC IS SUBTRACTED FROM T IN THE
        STAX    D       ; ACCUMULATOR AND STORED IN THE REAL
        INX     D       ; PART OF A(IP) POINTED TO BY DE.
        INX     B       ; THE SAME IS DONE FOR THE IMAGINARY
        LDAX    B       ; PART.
        SUB     H       ;
        STAX    D       ;
        INX     B       ;
        INX     D       ;
        LHLD    ACCI    ; DO SAME FOR IMAGINARY PART.
        LDAX    B       ;
        SUB     L       ;
        STAX    D       ;
        INX     D       ;
        INX     B       ;
        LDAX    B       ;
        SBB     H       ;
        STAX    D       ; A(IP)=A(I)-T
        POP     B       ; RESTORE ADDRESS OF A(I)
        LHLD    ACCR    ; LOAD HL WITH REAL PART OF T
        LDAX    B       ; START CALCULATING A(I)=A(I)+T
        ADD     L       ;
        STAX    B       ; STORE BACK INTO A(I)
        INX     B       ;
        LDAX    B       ;
        ADC     H       ;
        STAX    B       ;
        INX     B       ;
        LHLD    ACCI    ; DO FOR IMAGINARY PART ALSO
        LDAX    B       ;
        ADD     L       ;
        STAX    B       ;
        INX     B       ;
        LDAX    B       ;
        ADC     H       ;
        STAX    B       ; A(I)=A(I)+T
        POP     PSW     ; RESTORE ALL REGISTERS
        POP     H       ;
        POP     D       ;
        POP     B       ;
        RET             ; RETURN
        END
```

DISTRIBUTION

	No. of Copies
Defense Documentation Center Cameron Station Alexandria, Virginia 22314	12
Commander US Army Materiel Development and Readiness Command Attn: DRCCG DRCRD 5001 Eisenhower Avenue Alexandria, Virginia 22304	 1 1
Commander Ballistic Missile Defense Systems Command Attn: BMDSC-HR P. O. Box 1500 Huntsville, Alabama 35807	1
Director Ballistic Missile Defense Advanced Technology Center Attn: ATC-R P. O. Box 1500 Huntsville, Alabama 35807	2
Commander US Army Electronics Research and Development Command Attn: DRSEL, Mr. Fishbien DRCPM-MALR Fort Monmouth, New Jersey	 1 1
Superior Technical Services, Inc. Attn: T. Ward 4308 Governors Drive Huntsville, Alabama 35805	1
DRCPM-MDE, Mr. Evans -HAE, Mr. Ams -TOL, Mr. Bishop	1 1 1
DRSMI-LP, Mr. Voigt	1
DRDMI-X -T, Dr. Kobler Mr. Fagan	1 1 1

	No. of Copies
-TE, Mr. Lindberg	1
Mr. Pittman	1
-TEO, Mr. Currie	1
-TEG, Mr. Cash	1
-TER, Mr. Low	1
Mr. Owen	50
-TG, Mr. Huff	1
-TD, Dr. McCorkle	1
-TBL	5
-TBD	3
-TI (Record Set)	1
(Reference Copy)	1