

NRL Report 7494

A Fast Fourier Transform Microcode for the Burroughs D-Machine

HAROLD H. SMITH

*Information Systems Group
Office of the Associate Director of Research for Electronics*

December 8, 1972



NAVAL RESEARCH LABORATORY
Washington, D.C.

CONTENTS

Abstract	ii
Problem Status.....	ii
Authorization	ii
GLOSSARY.....	iii
1. INTRODUCTION	1
1.1 Objective and Conclusions	1
1.2 The D-Machine	4
1.2.1 Configuration	4
1.2.2 Programming	5
1.3 The FFT Program	7
2. PROGRAM DETAILS	12
2.1 Initialization and Sine Table Generation.....	12
2.2 Bit Reversal	15
2.3 Data Transfer	19
2.4 Zero-Degree Butterfly	19
2.5 Ninety-Degree Butterfly	24
2.6 45/135-Degree Butterfly	24
2.7 General Butterfly.....	29
2.8 Sine/Cosine Lookup.....	29
2.9 Real Multiplication	36
2.10 Overflow, Real Data, and the Inverse Transform	36
3. TEST RESULTS	36
4. SUITABILITY OF THE D-MACHINE TO SIGNAL PROCESSING	41
REFERENCES.....	43
APPENDIX A — Program Listing	44
APPENDIX B — Real-Valued Input Sequence	59
APPENDIX C — Inverse Transform.....	61

ABSTRACT

A fast Fourier transform program has been developed and run on a simulation of the Burroughs D-Machine processor. This program was developed to gain insight into the use of microprogrammed structures for signal processing applications and to serve as a reference "benchmark" during the development, evaluation, and application of programmable signal processors at the Naval Research Laboratory.

The D-Machine simulator and a companion translator program have been written in Fortran to run on the Control Data Corporation's Kronos time-sharing system. This action was taken because of a paucity of software support for the D-Machine itself and because the simulator provides debugging features not found in hardware. The time-sharing terminal was also more accessible.

A novel feature of the FFT program is its ability to terminate calculations before $\log_2 N$ stages have been completed (N is the number of complex input data points), thereby providing multiple estimates of less than N spectral lines, rather than one estimate of each of N lines, as is the case when $\log_2 N$ stages are carried to completion. Several of the test spectra which have been calculated are described in Section 3 of the report. For reasons of scarcity of registers and lack of concurrency, the D-Machine is deemed poorly suited to signal processing applications.

PROBLEM STATUS

This is an interim report; work is continuing on the Problem.

AUTHORIZATION

NRL Problem B02-10
Project No. XF-21-241-015-K152

Manuscript submitted August 30, 1972.

GLOSSARY

A1, A2, A3	primary input registers for the adder
ABT	all bits true (i.e. equal to ones)
AMPCR	alternate microprogram count register
$A(n)$	complex spectral line, output of the program
AOV	adder overflow
BMAR	concatenation of BR1 or BR2 and MAR
BR1, BR2	base register 1, base register 2
B register	primary interface from main memory
Butterfly	the computational kernel which combines two complex input points with a complex weighting factor to yield two complex output points
CAJ	condition adjust
COV	counter overflow
CTR	counter
CU	control unit
EXTOP	external operations
FFT	fast Fourier Transform
i	$\sqrt{-1}$
I	index of entries in sine table; imaginary part
j	index of input data samples
k	stage number, exponent in angle denominator
LC1, LC2	local condition flags 1 and 2
LIT	literal register
LMAR	an instruction; load literal register into MAR
LSB	least significant bit
LU	logic unit
LUOP	logic unit operations
M	number of FFT stages to be completed
MAR	memory address register
MCU	memory control unit
MDOP	memory or device operations

MIR	memory information register
MPAD	microprogram address
MPM	microprogram memory
MSB	most significant bit
n	index of output spectral lines
N	number of complex input samples
P_{2R}	address of real part of the second (lower) point in a butterfly. See Fig. 1-5.
R	factor in angle numerator; real part
RDC	read complete (flag bit)
RMI	ready MIR (flag bit)
SAR	shift amount register
S-memory	main memory (containing data)
Translang	a D-Machine symbolic programming language
$X(j)$	complex input data sample, input to the program

A FAST FOURIER TRANSFORM MICROCODE FOR THE BURROUGHS D-MACHINE

1. INTRODUCTION

1.1 Objectives and Conclusions

NRL is developing a high-performance, microprogrammable Signal Processing Element (SPE) (1-4). Early in the development cycle, it was decided to microprogram a Fast Fourier Transform (FFT) code for an existing microprogrammable machine. The Burroughs D-Machine is a recently developed microprogrammable processor (5). A unit with a 32-bit word length is available at the Naval Research Laboratory (NRL).

The D-Machine cannot be classified as a high-performance signal processor because of its lack of a multiplier and sufficient high-speed registers, but it was believed that programming the D-Machine would expose the issues of bookkeeping and address calculation that must be faced in the design of an efficient signal processor but which are often obscured in general purpose machines and languages.

The D-Machine may be programmed in a symbolic language, called Translang, for which Burroughs provides a translator program that runs on their B5500 computer. Since a B5500 was not readily available for this effort, and the software support for the D-Machine hardware was nil, programs were written in Fortran to perform the translation function and to simulate the hardware on the Kronos time-sharing system. With the aid of these support programs, the FFT program was written and debugged with an emphasis on maximizing computational speed consistent with a reasonable program size. The result is the program listed in Appendix A, comprising 766 instructions. This is many times more than the bare minimum needed to perform an FFT calculation, for several reasons which are described in Section 2.

The configuration of the D-Machine, illustrated in Figs. 1-1 and 1-2, differs greatly from the general configuration recommended for high-speed signal processors (6-9). It does not take advantage of the highly structured nature of the FFT operation; instead, it performs all operations serially using a single arithmetic/logic unit. The program reflects this, being organized to minimize the number of memory accesses required.

The test results, which are described in Section 3, indicate the large numbers of clock cycles expended in each of the program tasks, and these are for relatively small sample sizes. The rapid growth of processing time as a function of sample size, as indicated in Table 1-1, made sample sizes in excess of 128 prohibitively expensive for testing.

The deficiencies of the D-Machine as a signal processor, discussed in Section 4, have been noted in the design of the NRL SPE. The Microprogrammed Control Unit (MCU) of the SPE (3,4) has been designed with a sufficient number of suitable registers to permit

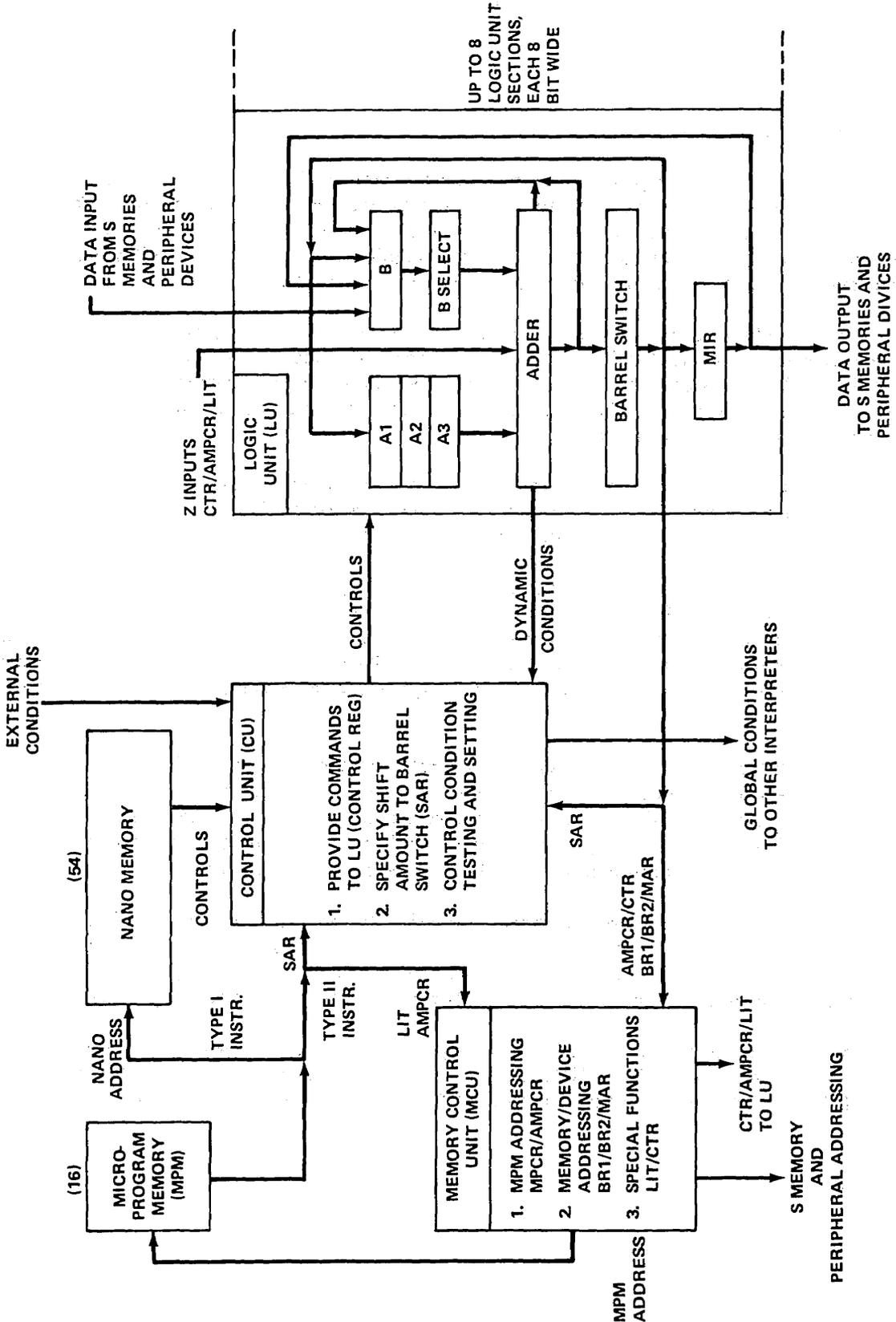


Fig. 1-1. — D-Machine major functional parts (Ref. 9)

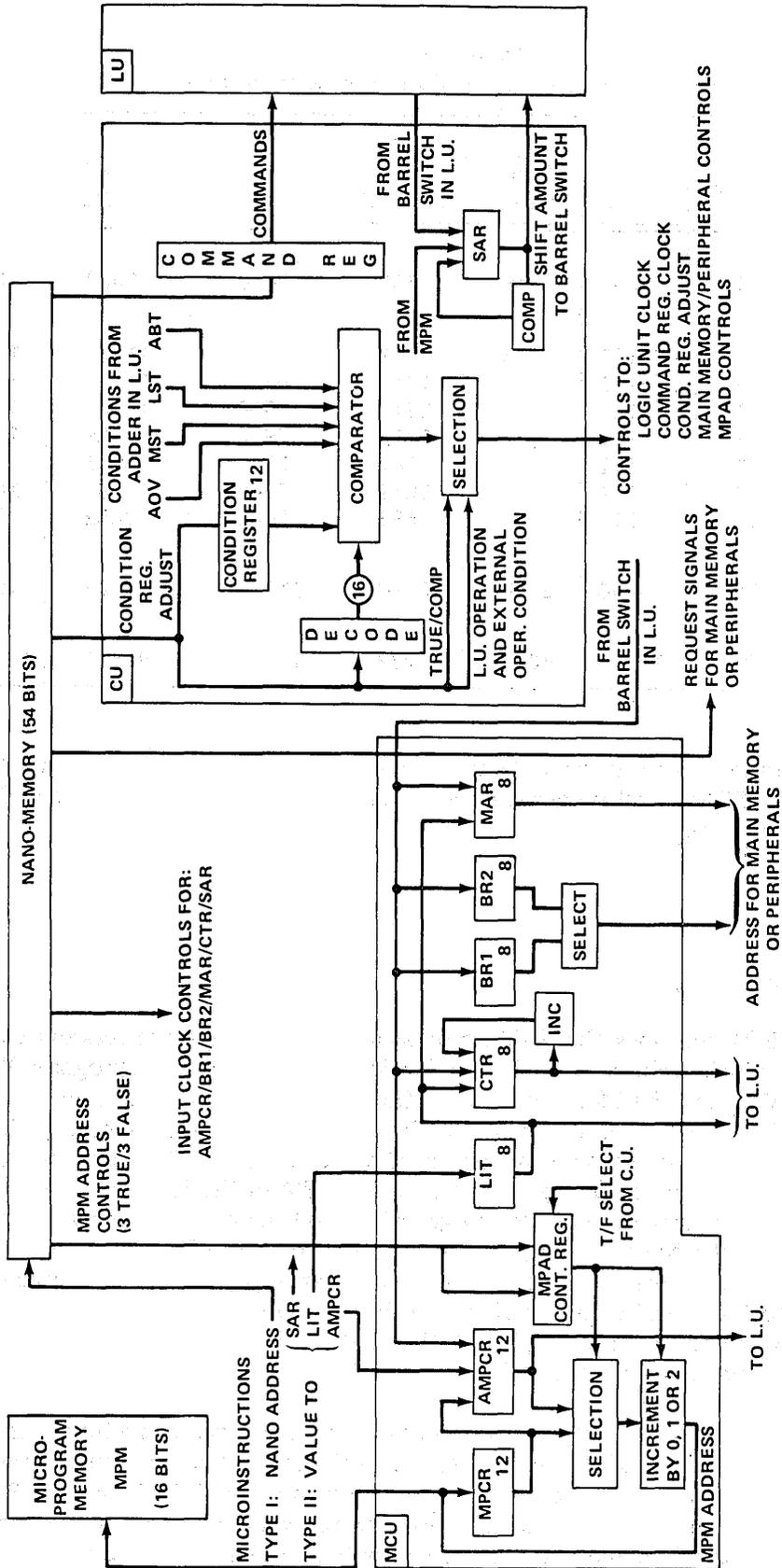


Fig. 1-2 — D-Machine logic interconnections (Ref. 9)

Table 1-1
Processing Time, in Clock Cycles, vs Sample Size

Input	(1) Initialize	(2) Sine Table	(3) Data Transfer	(4) Bit Reversal	(5) Special Butterflies	(6) General Butterflies	(1+4 +5+6)
16 points SAW 116	54	1339	548	582	3040	2522	6198
32 points							
SAW 132	59	3111	1092	1211	5953	11339	18562
REC 132	59	3111		1211	5957	11391	18618
P 2432	59	3111		1211	5956	11381	18607
CS 325	59	3111		1211	5953	11326	18549
CS 324	59	3111		582	5056	3945	9642
SIN 325	59	3111		1211	5947	11184	18401
SIN 324	59	3111		582	5052	3852	9545
64 points							
SAW 164	64	6656	2180	2480	11798	35622	49964
128 points							
SAW 1128	69	13774	4356	5029	23091	96811	125000

much higher concurrency with an only slightly wider control word, and the Signal Processing Arithmetic Unit (SPA) will be designed with high-speed parallel arithmetic and concurrent address generation.

1.2 The D-Machine

1.2.1 Configuration

Burroughs (5,9) describes the D-Machine as a family of digital processors utilizing a modular building block architecture. The major subunits are the Microprogram Memory (MPM), the Nano Memory, the Memory Control Unit (MCU), the Control Unit (CU), and the Logic Unit (LU), all of which are shown in Fig. 1-1. The LU is composed of individual sections, each eight bits wide, up to a maximum of eight sections. The simulated LU is 32 bits wide. The FFT control program resides in the MPM in the form of 16-bit "instructions." Most of these are actually pointers to generic 54-bit instructions known as Type I instructions, stored in the Nano Memory. The remainder of the MPM instructions, known as Type II instructions, are used to load the value of an arbitrary constant, contained in the instruction, into the literal register (LIT), the shift amount register (SAR), or the alternate microprogram control register (AMPCR). Data and parameters are stored in the S memory which is controlled by the MCU.

The contents of S memory are addressed in two parts; the most significant eight bits are in a base register (BR1 or BR2) and the least significant eight bits are in the memory address register (MAR). Words are read from S memory into the LU adder via a B register, and are written into S memory from the LU barrel switch via a Memory Information Register (MIR). The barrel switch permits the contents of the adder to be shifted left or

right, end off, or right circular by a number of bits equal to the contents of the SAR. The LU also contains three general purpose registers denoted by A1, A2, and A3. The MCU contains a counter (CTR) having a maximum count of 255.

The A registers, LIT and CTR, may be used as the X input to the adder, and the B register, LIT, CTR, AMPCR or the concatenated base register and MAR (BMAR), may be used as the Y input. For this purpose, the B register is separable into three parts: the most significant bit (MSB), the least significant bit (LSB), and the 30 central bits. These can be specified as either zero, one, their true value, or the complement of their true value, by means of a three-part subscript on B. For example, B_{01T} requires that bit number one of B shall be zero, bits two through 31 shall all be ones, and bit 32 shall have its current value, whether zero or one, when B is used as the Y input to the adder.

Overflows in the adder (AOV) and the counter (COV) are indicated indefinitely. The dynamic states of MST and LST are indicated for one clock pulse following each adder operation. An "all bits true" (ABT) condition is also indicated dynamically. Successful completion of a Read operation is indicated by setting a read complete (RDC) bit to one. Completion of a Write operation is marked by setting a ready MIR (RMI) bit. The simulator arbitrarily requires two clock cycles to elapse before setting RDC and RMI. Two local-condition flags (LC1 and LC2) are also available to the programmer. All of these flags can be used as the basis for making conditional branches in the program and for performing LU operations conditionally.

The method of employment of the D-Machine hardware in executing the FFT program is shown in the detailed flowcharts of Section 2 (and in the comments in the program listing). Each memory access is called out and the juggling of data in the working registers is shown.

1.2.2 Programming

Type I instructions normally require two clock cycles to be executed. The sequence of operations in these two phases is illustrated in Table 1-2. When Type I instructions occur in sequence, Phase 2 of each instruction is executed concurrently with Phase 1 of its successor. Type II instructions require only a single clock cycle and cause the Phase 2 operations of the concurrent Phase 1 instruction to be repeated during the following clock cycle. A succession of Type II instructions will cause the Phase 2 of the last preceding Type I instruction to be repeated indefinitely.

Since the contents of destinations are not changed until the end of Phase 2, it is possible to execute an LU operation before the registers involved have changed as a result of the preceding operation. For example, when using LIT to load MAR (or CTR, or any of the other registers), the instruction to load LIT should follow the loading of MAR rather than precede it. Thus, the sequence

```
1 = LIT
LMAR
2 = LIT
```

Table 1-2
Instruction Timing*

Phase 1

1. Obtain microinstruction address (MPAD) from either MPCR or AMPCR.
2. Select address increment: 0, 1, or 2.
3. Read the addressed microinstruction.
4. Decode the microinstruction for either Type I or Type II.

For a Type II:

- 5a. Use the low-order part as a literal.
- 11a. Step successor to MPAD.
- 11b. Clock literal(s) to SAR and/or LIT; AMPCR.

For a Type I:

- 5b. Use the low-order part as a nanoinstruction address.
7. Decode the nanoinstruction.
8. Select the condition to test.
- 10a. Test Logic Unit Operations, LUOP.
- 10b. Test External Operations, EXTOP.
- 11c. If EXTOP is true, enable CAJ/MDOP.
- 11d. If LUOP is true, complete destination part of Phase 2 of prior LUOP; also, decode and load command register.
- 11e. True/False successor to MPAD.
- 11f. Reset tested conditions.

Phase 2

- 1a. Select adder X input.
- 1b. Select adder Y input.
- 3a. Inhibit carry.
- 3b. Select and do adder operation.
7. Adder dynamic conditions (ABT, AOV, LST, MST) available for testing in Phase 1 of subsequent instruction.
9. Select shift direction and do shift.
- 11g. When LUOP condition is true, change destinations.

*The number preceding each item indicates its relative time of occurrence.

results in the MAR being loaded with the number 2. For this reason, instructions which load the SAR, LIT, or AMPCR should follow the instruction which uses them in a logic unit operation.

In addition to logic unit operations (LUOP), the 54-bit microinstruction recognizes memory/device operations (MDOP) and a condition adjust (CAJ). One of each of these may occur in the same instruction and the LUOP or MDOP/CAJ may be conditional while the other is unconditional. The instruction specifies the source of its successor instruction

for both true and false conditions. A four-bit field is used to specify the source of input to the B register. One such source is the MIR which can therefore serve as a convenient temporary store whose contents are readily moved into the B register for adder operations.

Memory access operations require that the desired address first be stored in BMAR, the concatenated base and memory address registers. This may be done by loading a literal or by incrementing. The MIR may be loaded for write operations while BMAR is being set up. LUOP's which do not change BMAR or the MIR (in the case of memory writing) may then be performed concurrently with the access. The access is completed when an RDC or RMI flag is set.

1.3 The FFT Program

The test program calculates a discrete Fourier transform according to Eq. (1-1) for data samples which are powers of two from 3 through 11.

$$A(n) = \sum_{j=0}^{N-1} X(j) \exp(-2\pi i j n / N). \quad (1-1)$$

The total number of complex data samples $X(j)$ is N . The program permits premature termination at the end of any stage M ($3 \leq M \leq \log_2 N$) in order to provide multiple estimates of the spectral lines $A(n)$. The four stages of a 16-point transform are illustrated in Fig. 1-3. The input data are located in an arbitrary section of memory whose first (lowest) address is a parameter of the program, and the output results are left in another arbitrary section of memory whose first address is a program parameter. When these addresses are equal, the maximum allowable value of N is 2048, a constraint imposed by the simulated 8192 locations of S memory.

Prior to executing the program, it is necessary to load S-memory locations 7936 through 7939 with the four parameters N , M , the Data Address, and the Results Address. A complete list of all parameters, both input and calculated, is given in Table 1-3. Calculations are in 16-bit, fixed-point format, although the simulated D-Machine has a 32-bit word length. The program contains no provision for arithmetic overflow, packing strictly real input data, or calculating inverse transforms. These topics are discussed in Section 2.10. Also, no window weighting was applied to the data or the output lines. Figure 1-4 is a functional flowchart which shows the major tasks in association with their program labels as they appear in the listing of Appendix A. The first two tasks, the generation of a table of trigonometric sines and the generation of a table of binary bit-reversed numbers, are included in the program for the sake of completeness. In a real-time processor operating on a continuous stream of input data, these functions would be done only once, at system startup, or not at all, by using prestored values. The FFT program described here is organized on a batch processing basis.

In executing the program, the complex points are processed in pairs, as shown in the "butterfly" diagram of Fig. 1-5, which depicts the complex multiplications of Eq. (1-1). The output pair resulting from each butterfly is stored in the locations from which the input pair to that butterfly were obtained. To reduce the number of memory accesses required, all of the butterflies that involve a particular value of angle exponent, from the

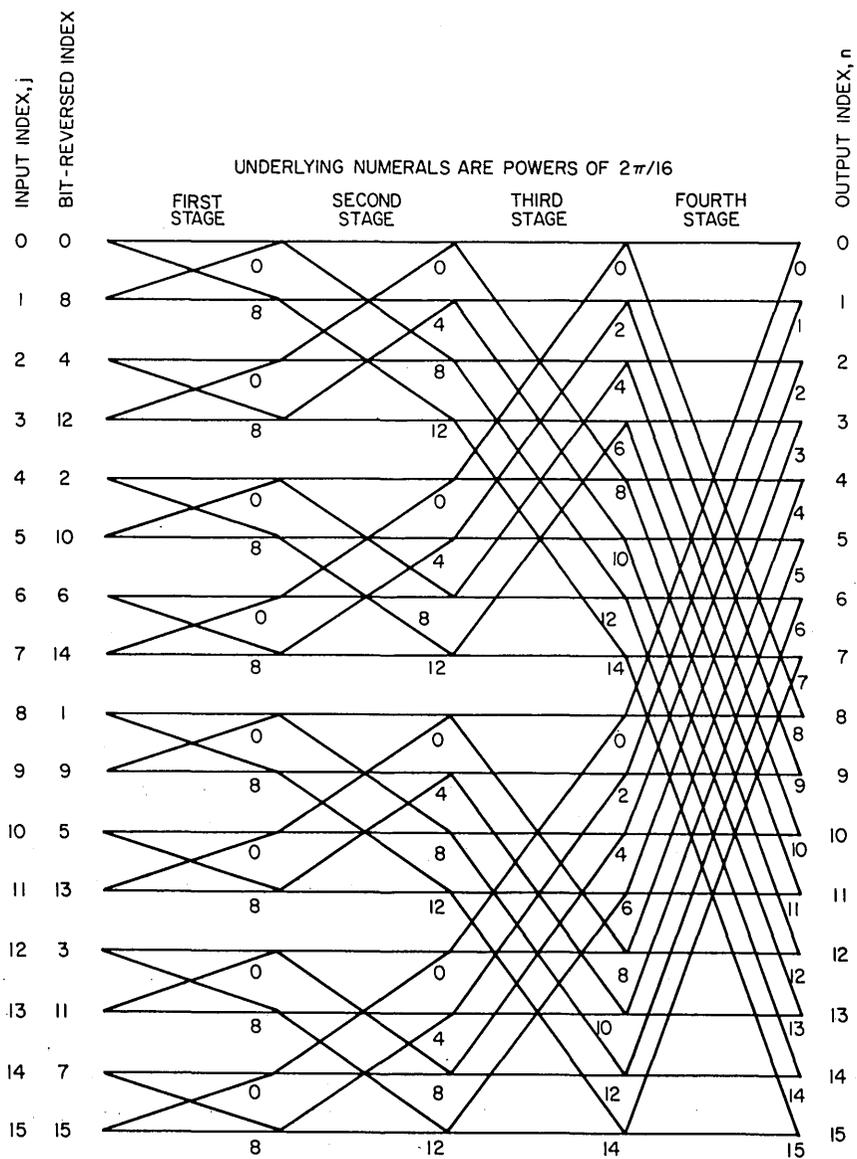


Fig. 1-3 — Sixteen-point FFT

Table 1-3
Program Parameters

Decimal Address	Parameter
7936	Number of sample points, N
7937	Number of stages to complete, M
7938	Data Address
7939	Results Address
7940	$\log_2 N$
7941	Sine table index, I
7942	Cosine of current angle
7943	Sine of current angle
7944	Angle denominator factor, K
7945	Angle numerator factor, R
7946	Unused
7947	Point spacing
7948	Address of P_{2R}
7949	Old P_{1R} value, X^2
7950	2^M
7951	$P_{2R} \sin \theta$
7952	Angle element, $N/4$
7953	Return address

first stage through the M th stage, are calculated at one time in a single group (10). Then the angle value is advanced, new values of the sine and cosine are obtained, and all butterflies involving that angle are calculated through the M th stage.

Separate coding is used for angles of 0 degrees, 90 degrees, and 45/135 degrees, to avoid the use of the 16-bit multiplication subroutine. This grouping of the butterflies has resulted in a nearly twofold increase in the number of instructions devoted to this function; 502 lines of code for the four butterfly routines, as compared to 262 lines for the general butterfly alone. Table 1-4 shows the number of lines of code used by each of the major functions in the program.

Data may be entered into the simulated S memory either directly from a data terminal keyboard, or from an existing data file. The program listing in Appendix A contains two instructions, at MPM locations 30 and 31, whose effect is to bypass the sine table generation by the program so that the sine table may be loaded from a data file. This procedure saves many clock cycles (and time-sharing service charges). If these two instructions are deleted, the reassembled FFT program will calculate and store the needed table of sine coefficients as described in detail in Section 2.1. The bit-reversal table generation was not bypassed in a similar manner because its size varies with the value of M and in various test runs M was varied for a fixed set of input data. The simulator can load the S memory from only one data file, which was reserved for the input data and sine table.

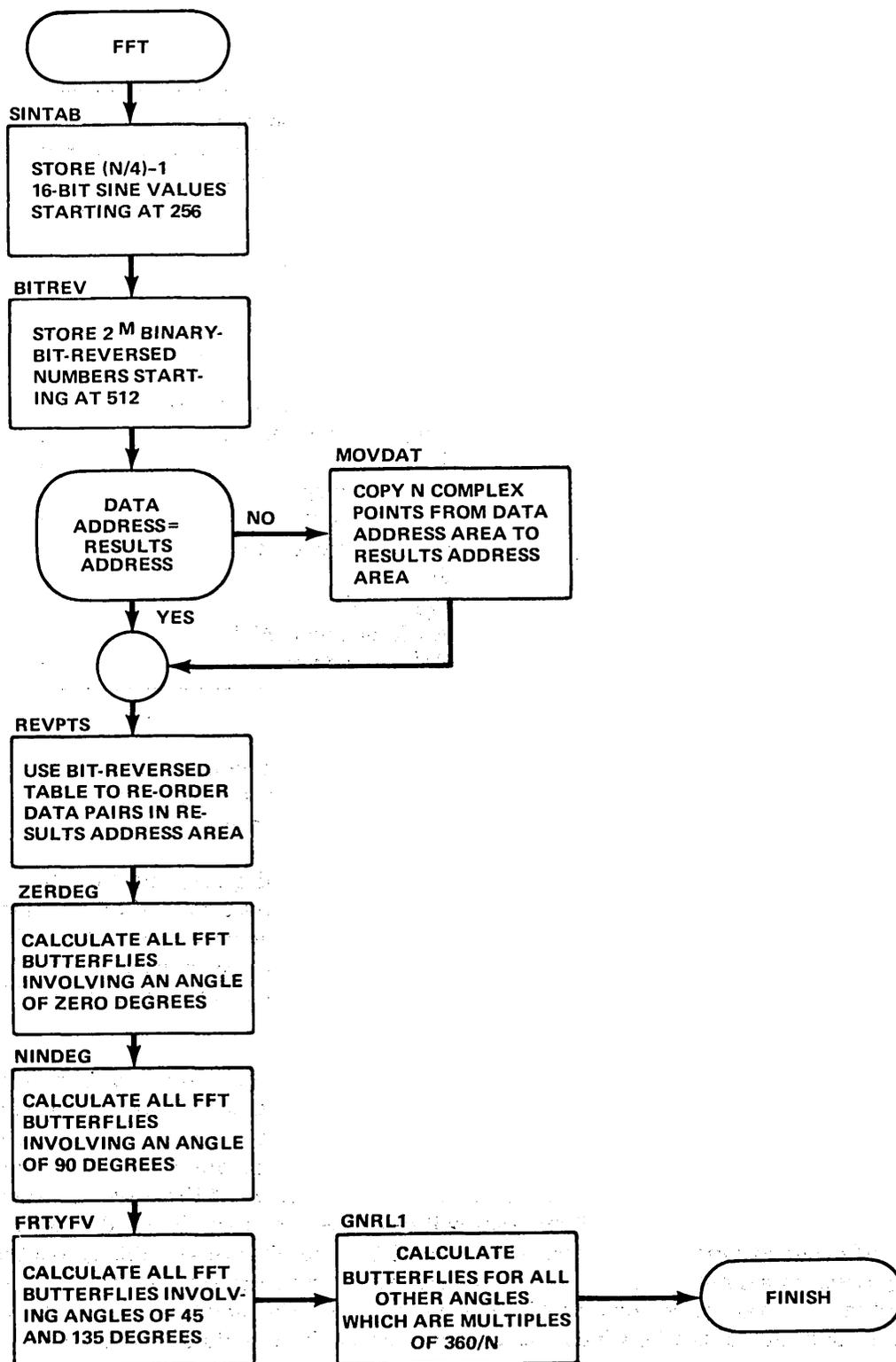
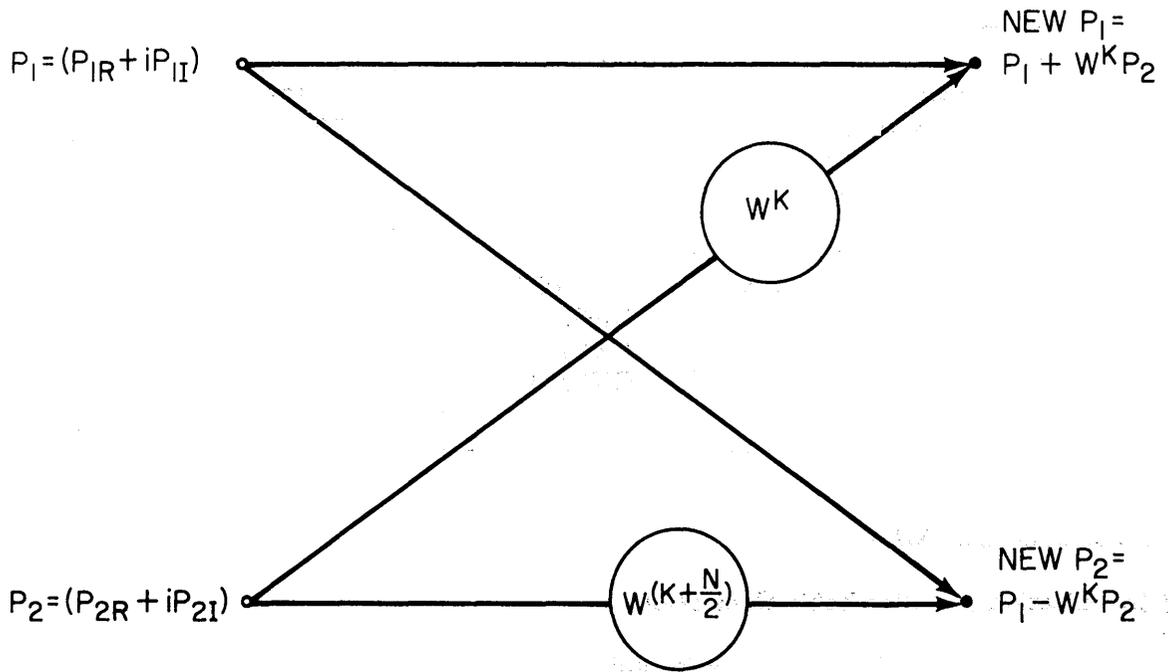


Fig. 1-4 -- Functional flowchart



$$W \triangleq \exp(2\pi i/N)$$

$$W^k \triangleq (\cos \theta - i \sin \theta)$$

$$P_{1R} \leftarrow P_{1R} + (P_{2R} \cos \theta + P_{2I} \sin \theta)$$

$$P_{1I} \leftarrow P_{1I} + (P_{2I} \cos \theta - P_{2R} \sin \theta)$$

$$P_{2R} \leftarrow P_{1R} - (P_{2R} \cos \theta + P_{2I} \sin \theta)$$

$$P_{2I} \leftarrow P_{1I} - (P_{2I} \cos \theta - P_{2R} \sin \theta)$$

Fig. 1-5 — FFT butterfly

Table 1-4
Coding Allocation

Function	Program Label	Lines of Code
Initialization	FFT	13
Sine Table	SINTAB	105
Real Multiply	MUPY	27
Bit Reversal	BITREV	88
Data Transfer	AROUND	31
Zero-degree butterfly	ZERDEG	62
90-degree butterfly	NINDEG	67
45/135-butterfly	FRTYFV	111
General butterfly	GNRL 1	<u>262</u>
		766

2. PROGRAM DETAILS

2.1 Initialization and Sine Table Generation

Figure 2-1 outlines the procedures for initialization and sine table generation. The two base registers BR1 and BR2 are assigned to data and parameters, respectively. The term *data* includes the table of sines, table of bit-reversed integers, input data, and output results. The parameters include the items listed in Table 1-3 of which the first four must be input at run time. The program begins by loading BR2 with its permanent value, 31 (= 1F hexadecimal). The first parameter N is thus located at address 1F00 hexadecimal, or 7936 decimal.

It is required that N be a power of two. Its logarithm is determined by scanning from the least significant bit (LSB) to the most significant bit (MSB) for the first nonzero bit. The counter is loaded initially with 243 and incremented each time a zero bit is detected. Because the D-Machine test of the LSB is based on the unshifted output of the adder, it is necessary to subtract 244 from the accumulated count in order to yield $\log_2 N$.

One quadrant of the complex plane corresponds to $N/4$ data points so that only $(N/4)-1$ sine values need to be stored when the end values of zero and unity are executed directly. The first entry in the sine table is located at address 256. The values are calculated from the following approximation (11):

$$\sin \frac{\pi}{2} X = C_1 X + C_3 X^3 + C_5 X^5 \quad (2-1)$$

$$C_1 = 1.5706268 \quad (2-2)$$

$$C_3 = -0.6432292 \quad (2-3)$$

$$C_5 = 0.0727102 \quad (2-4)$$

$$-1 \leq X \leq 1. \quad (2-5)$$

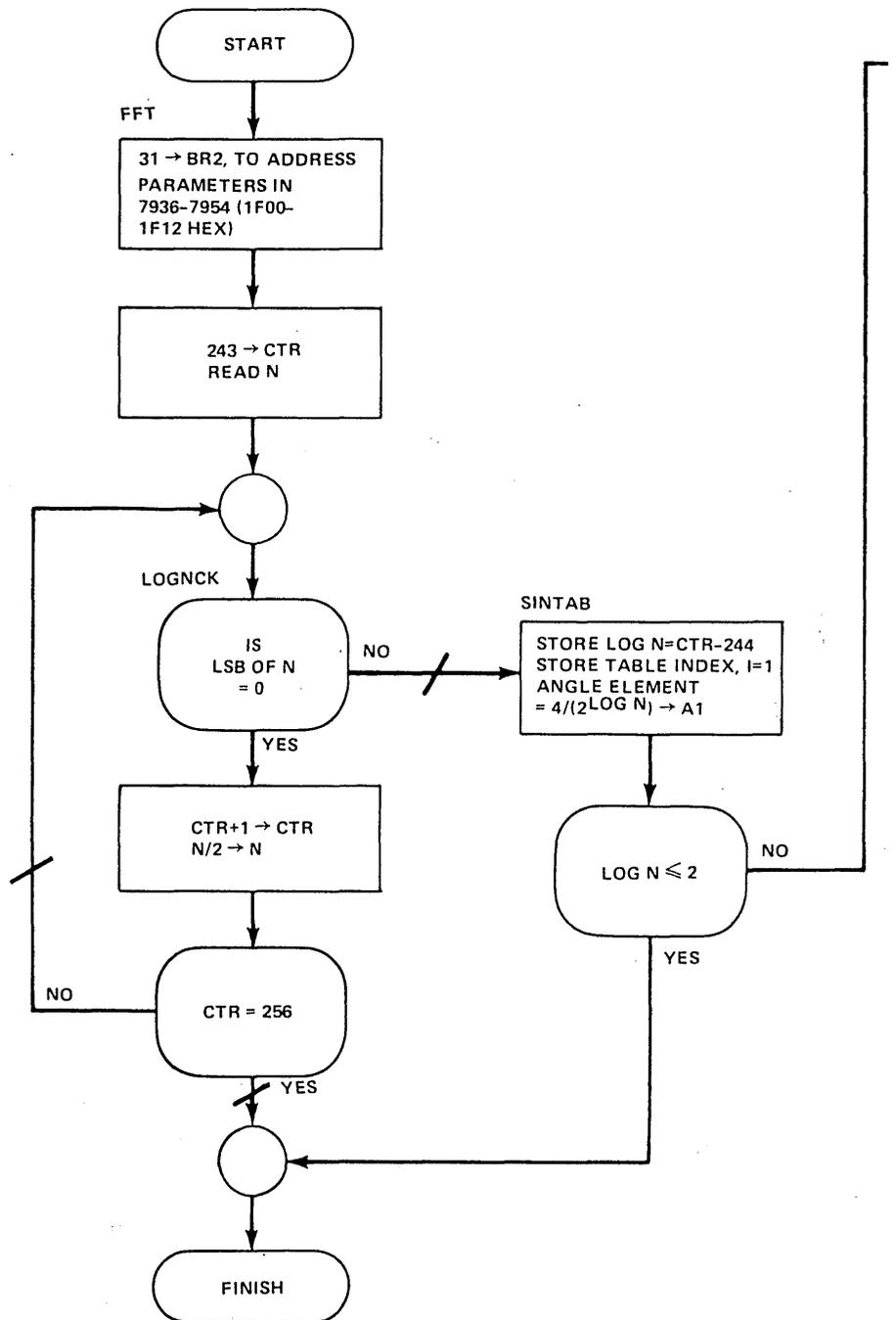
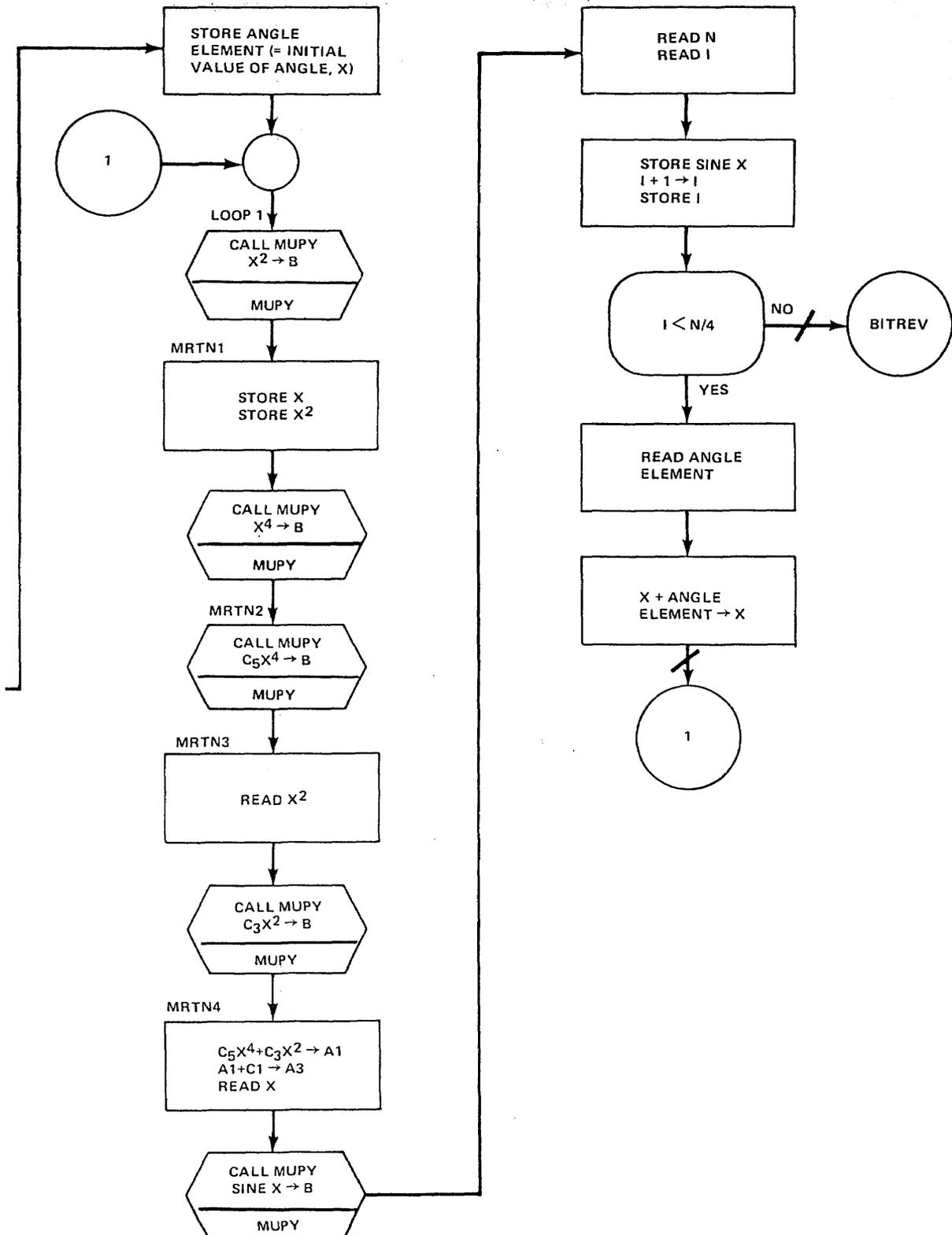


Fig. 2-1 — Initialization and



and sine table generation (/ denotes a program branch)

Values calculated from this formula are compared with the correct values in Table 2-1. In the stored sine table, the binary point is located between the 18th and 19th bits from the left in each 32-bit word. The working registers A1, A2, A3, B, and MIR cannot be spared for temporary storage so the following quantities are saved in external S memory and recalled as needed: the basic angle element ($=4/N$), the total angle, the intermediate result X^2 , the return address during subroutine calls, and the table index I . If the maximum values of N were limited to 1024 (rather than 2048), it would be possible to hold I in the counter instead of in the S memory.

Table 2-1
Calculated Sine Values

Angle, degrees	Decimal Sine	Scaled Sine	Calculated Sine
5.625	0.0980	1605	1605
11.25	0.1951	3196	3196
16.875	0.2902	4758	4755
22.5	0.3827	6271	6269
28.125	0.4714	7723	7723
33.75	0.5556	9103	9103
39.375	0.6343	10392	10394
45.0	0.7071	11586	11586
50.625	0.7730	12664	12666
56.25	0.8315	13623	13623
61.875	0.8819	14449	14450
67.5	0.9239	15137	15136
73.125	0.9569	15677	15678
78.75	0.9808	16069	16067
84.375	0.9952	16305	16305

2.2 Bit Reversal

In the in-place algorithm used in this program, the input data are permuted according to a binary bit-reversed sequence before the FFT butterflies begin. This is accomplished by first generating a table of bit-reversed integers, using a counting process whose logic is illustrated in Fig. 2-2 (12). The exact sequence depends on the size of the table, which contains 2^M entries. The first integer produced by the counter is always zero. Each integer thereafter is produced by examining the preceding integer for the first occurrence of a zero in its binary representation, starting at the MSB and scanning toward the LSB. When the first zero is detected at position J (J varies from one to M), an amount equal to $2^M/2^J$ is added to the preceding entry, and similar amounts for lesser values of J down to and including one are subtracted. The effect is to convert the first zero to a one and all of the ones in the more significant bits to zeros.

Figure 2-3 illustrates the program flow, in which the scanning process employs a mask with initial value $2^M/2$, which is repeatedly divided by two until the first zero is found. The value of J is indicated by the counter. The last entry in the table is equal to $2^M - 1$; the first entry, zero, is at S-memory location 512. When the table is complete,

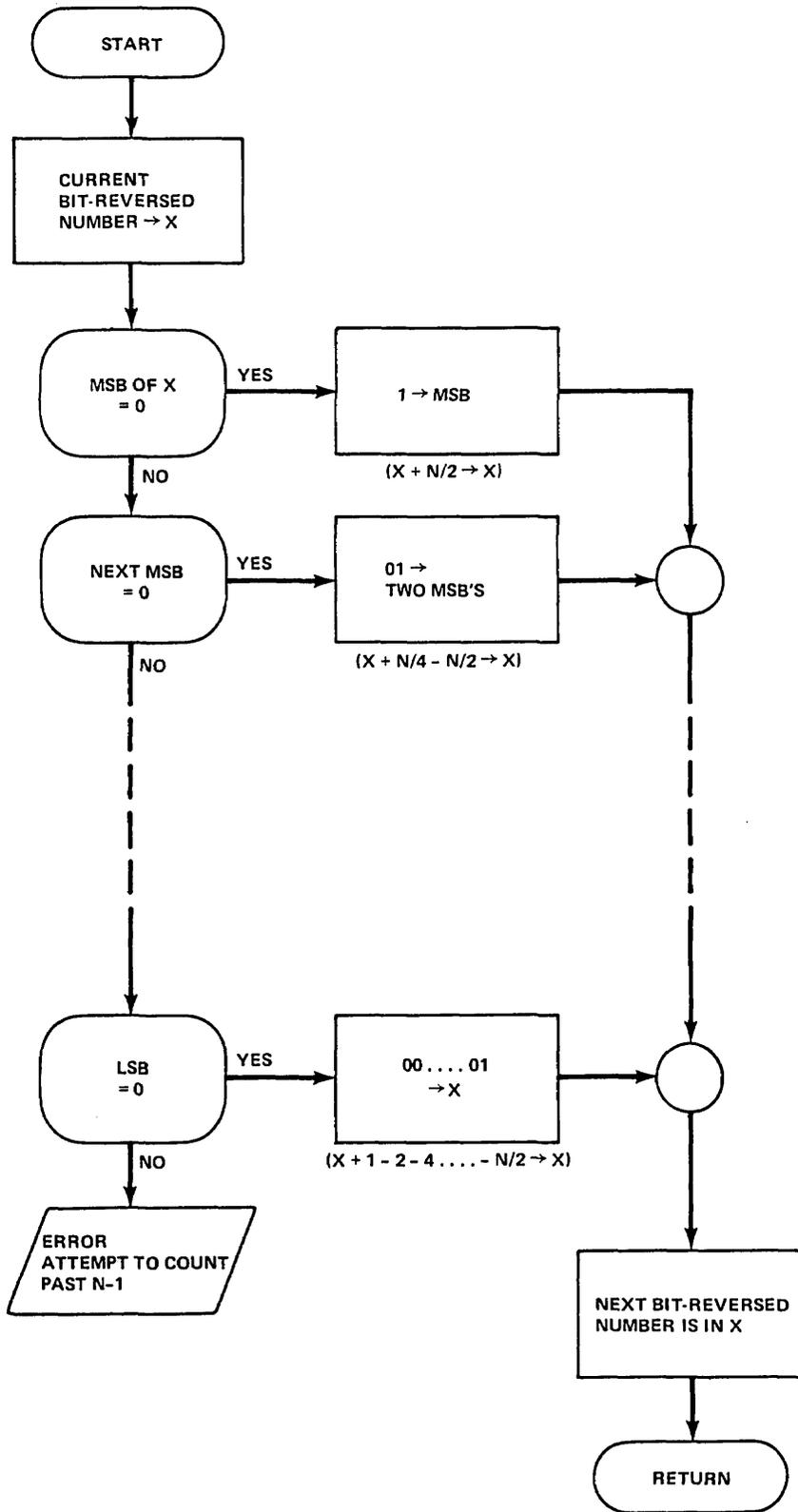


Fig. 2-2 — Bit-reversed counting

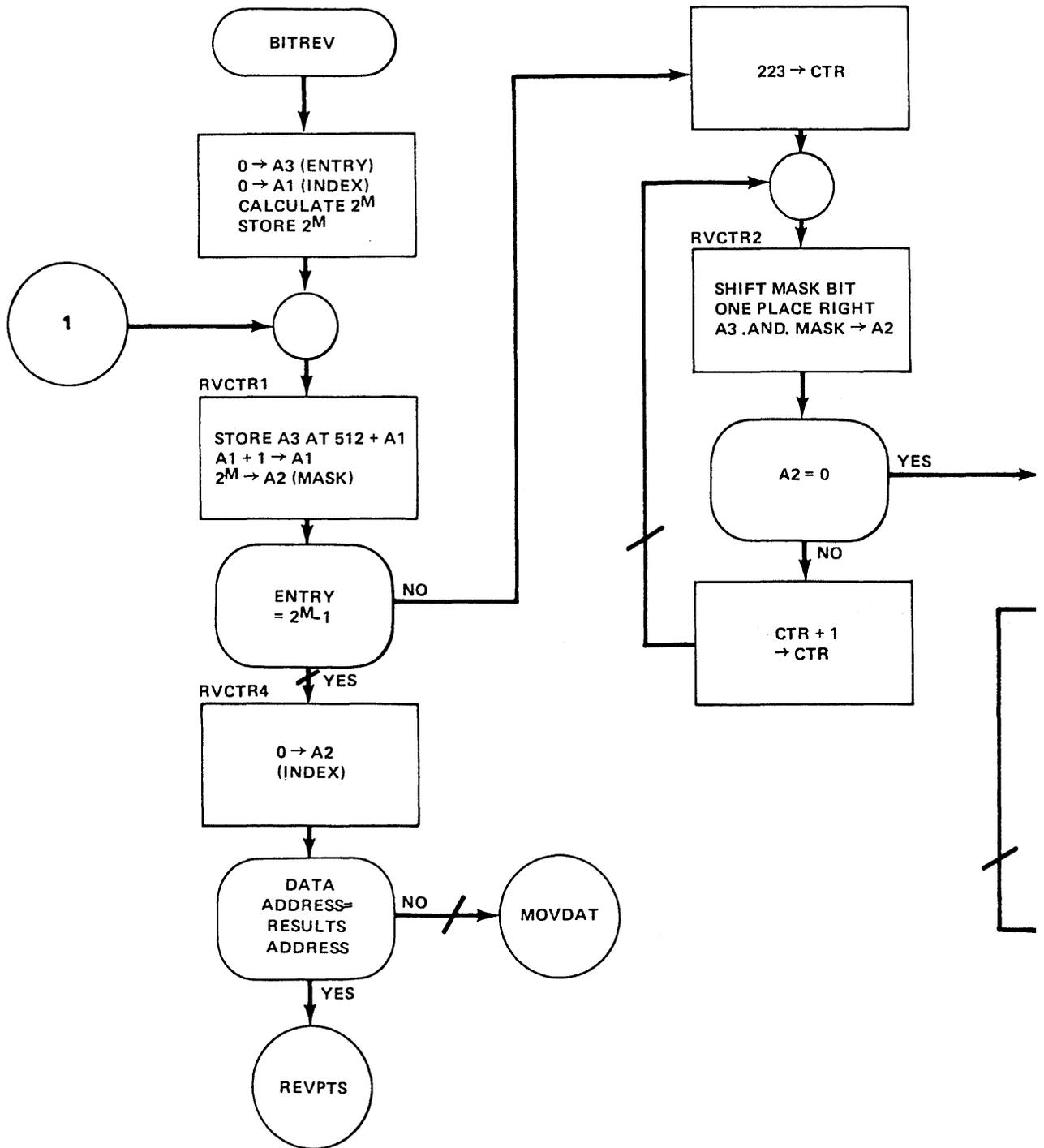
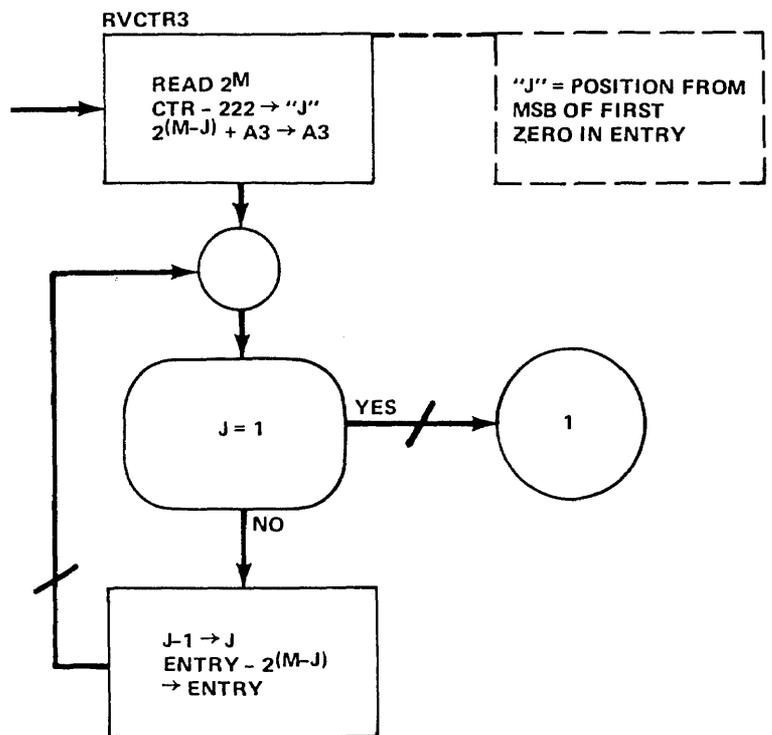


Fig. 2-3 — Bit reversal



the Data Address and Results Address are tested for equality to determine if it is necessary to move the data to a new area of memory before permuting the points.

2.3 Data Transfer

If the input Data Address parameter is unequal to the output Results Address parameter, the program copies the input data into the results area before performing the bit-reversal permutations (see Fig. 2-4). The real and imaginary components of each point are stored pairwise, in consecutive locations, with the real component at the lower address. Thus, $2N$ 32-bit words are copied from the continuous S-memory area whose lowest address is given by the contents of location 7938 to the continuous area whose lowest address is given by the contents of location 7939.

To provide multiple estimates when M is less than $\log_2 N$, the points are permuted in sections of size 2^M , a total of $N/2^M$ times. The counter keeps track of the number of sections processed. Each time a complete section is permuted the Section Start address is increased by $2 \cdot 2^M$, since there are two data words per point. An index scans the points within a section. If the index is less than the corresponding (indexed) integer in the bit-reversal table, then the data word pair that corresponds to the value of the index is swapped with the data word pair that corresponds to the integer, used as an index. In the scanning process, if the index is equal to the integer no swap is required and if it exceeds the integer then the swap has already taken place. When the final section has been permuted the counter increment results in an overflow and the program branches to the butterfly calculations.

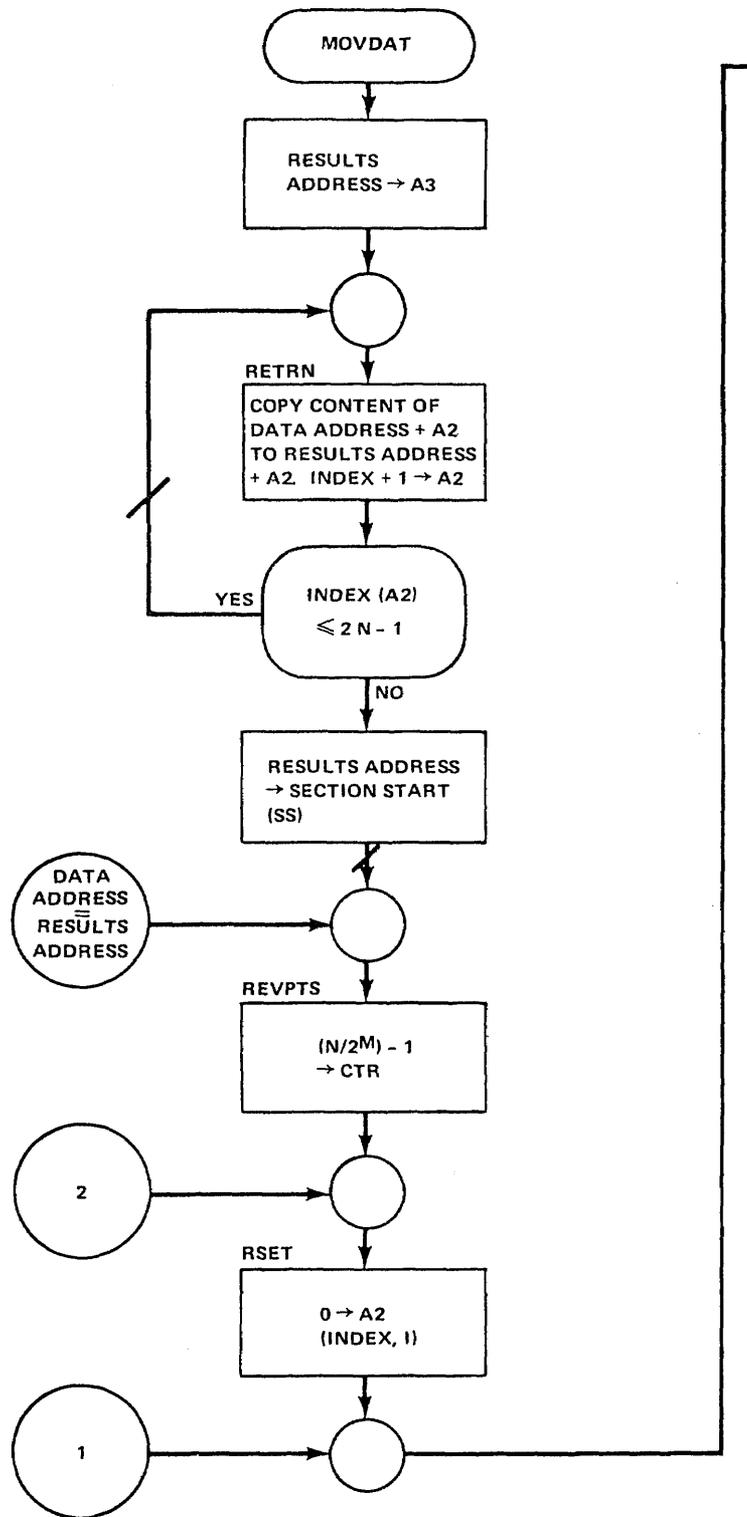
2.4 Zero-Degree Butterfly

Examination of Fig. 1-3 shows that the spacing in memory between the points in any butterfly pair doubles in each successive stage and the separation between the pairs that involve the same angle also doubles. In the zero-degree angle group the address of the real component of the first point of the first butterfly of any stage is a constant, equal to the Results Address. The first stage consists entirely of zero-degree butterflies (Fig. 2-5), $N/2$ in number; the second stage consists of $N/4$ such butterflies, and so forth, until the $\log_2 N$ stage consists of only one butterfly involving each angle.

Memory addressing for the input pairs is keyed on the real component of point 2, denoted by P_{2R} . The spacing has an initial value of two (memory locations) and the P_{2R} address is initialized to the Results Address minus the spacing value. The routine labeled B1BK generates the address pairs by adding the spacing to previous values, and as long as the P_{2R} address does not exceed the Results Address plus $2N-1$, the butterfly calculation is performed. When it does exceed this value, the spacing is doubled and immediately tested to determine if the M th stage has been passed. If not, the zero-degree butterflies are continued with the new spacing; if so, the program branches to the 90-degree calculations.

The equations of the general butterfly are

$$P_{1, 2R} = P_{1R} \pm (P_{2R} \cos \theta + P_{2I} \sin \theta) \quad (2-6)$$



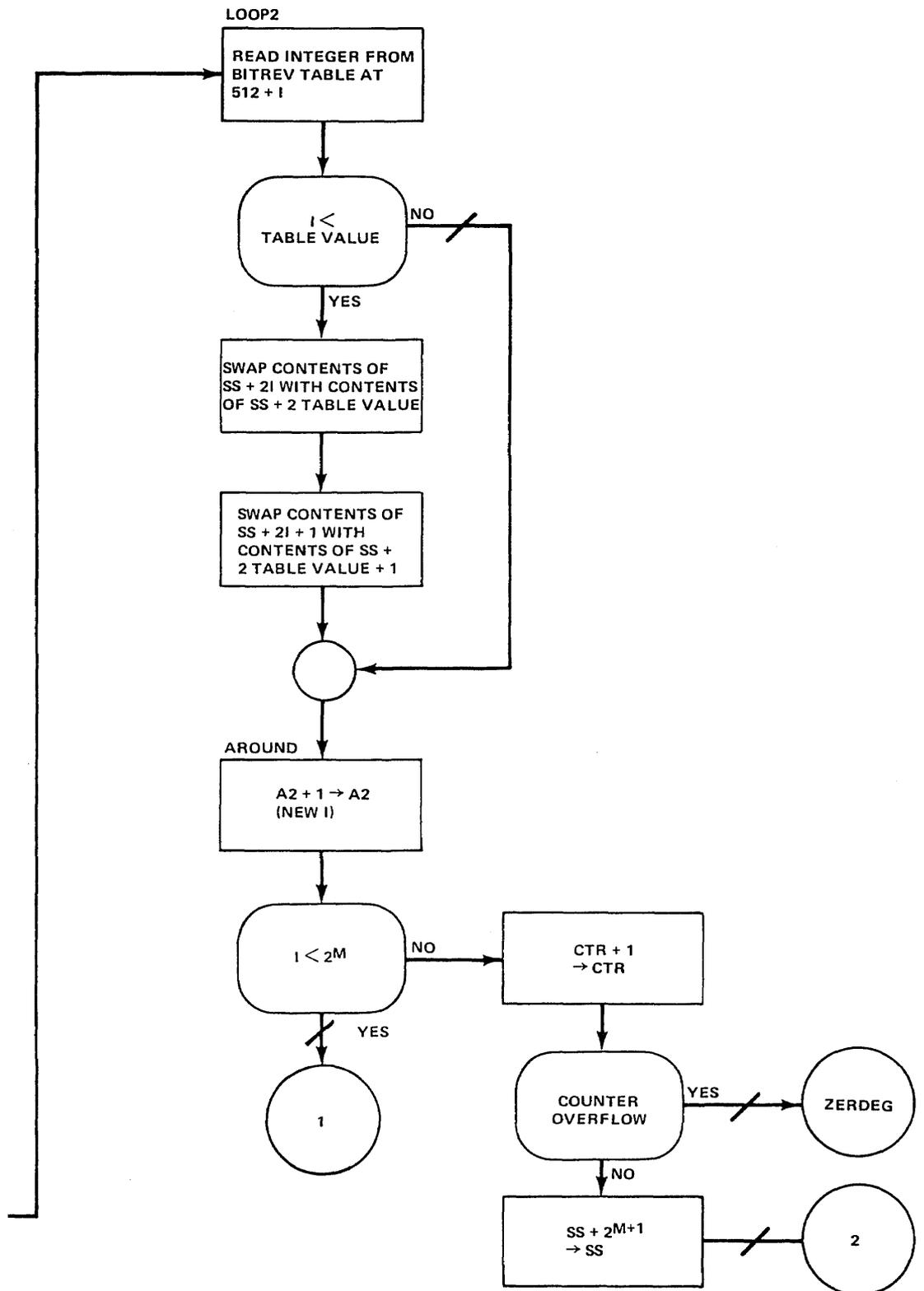


Fig. 2-4 — Data transfer

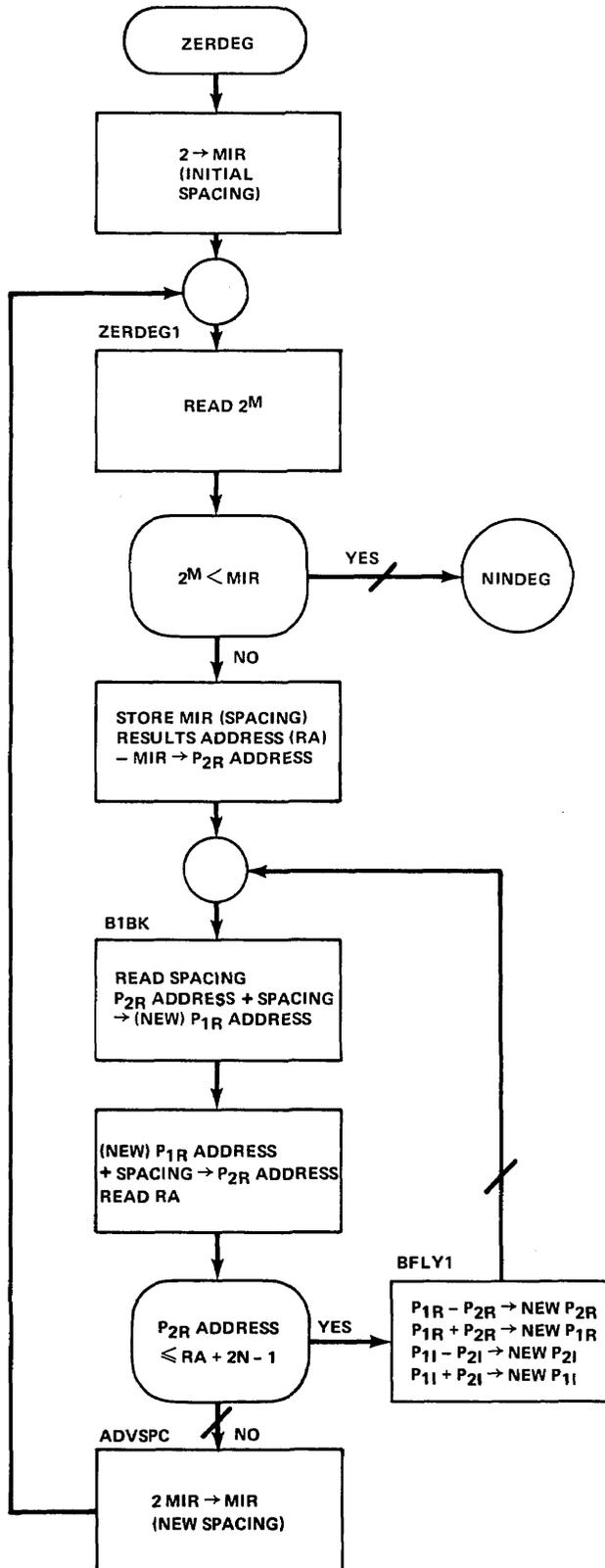


Fig. 2-5 — Zero-degree butterfly

$$P_{1, 2I} = P_{1I} \pm (P_{2I} \cos \theta - P_{2R} \sin \theta), \quad (2-7)$$

where the subscripts R and I denote the real and imaginary components, respectively, and the upper plus sign applies to point 1 and the lower minus sign applies to point 2. The angle exponent of Eq. (1-1) is denoted by θ . For the zero-degree butterfly these simplify to

$$P_{1, 2R} = P_{1R} \pm P_{2R} \quad (2-8)$$

$$P_{1, 2I} = P_{1I} \pm P_{2I}, \quad (2-9)$$

and for the 90-degree butterfly to

$$P_{1, 2R} = P_{1R} \pm P_{2I} \quad (2-10)$$

$$P_{1, 2I} = P_{1I} \pm (-P_{2R}). \quad (2-11)$$

When $\theta = 45$ degrees, the following equations are solved,

$$P_{1, 2R} = P_{1R} \pm 0.707 (P_{2R} + P_{2I}) \quad (2-12)$$

$$P_{1, 2I} = P_{1I} \pm 0.707 (P_{2I} - P_{2R}), \quad (2-13)$$

and when $\theta = 135$ degrees, the butterfly solution is

$$P_{1, 2R} = P_{1R} \pm 0.707 (P_{2I} - P_{2R}) \quad (2-14)$$

$$P_{1, 2I} = P_{1I} \pm 0.707 (-P_{2R} - P_{2I}). \quad (2-15)$$

2.5 Ninety-Degree Butterfly

The calculation of the 90-degree butterflies (Fig. 2-6) differs from those of zero degrees in only three respects. The initial spacing is four, rather than two, because these butterflies first occur in the second stage. The initial offset of the first butterfly in any stage is equal to the Results Address minus one-half of the spacing for that stage. Also, the butterfly formulas reflect an interchange of signs and real and imaginary components because of the changed values of the trigonometric functions.

2.6 45/135-Degree Butterfly

The routine 45/135-degree butterfly is entered with the LC1 flag in a reset condition, indicating an angle of 45 degrees. In this case, the initial memory address offset is $-3/4$ times the spacing and the sine and cosine are both positive. After completing the 45-degree calculations through the M th stage, the flag is set, an initial offset of $-1/4$ times the spacing is used, and the sine is positive but the cosine is negative in the butterfly formulas.

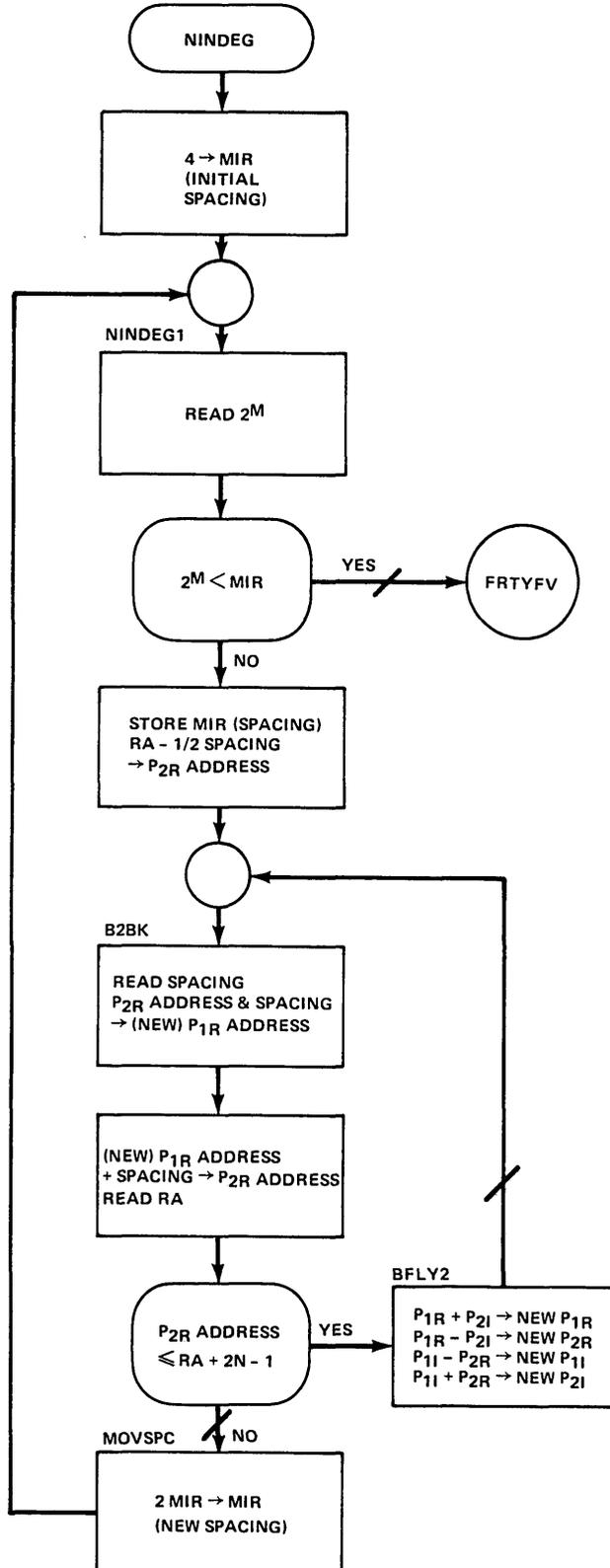


Fig. 2-6 — Ninety-degree butterfly

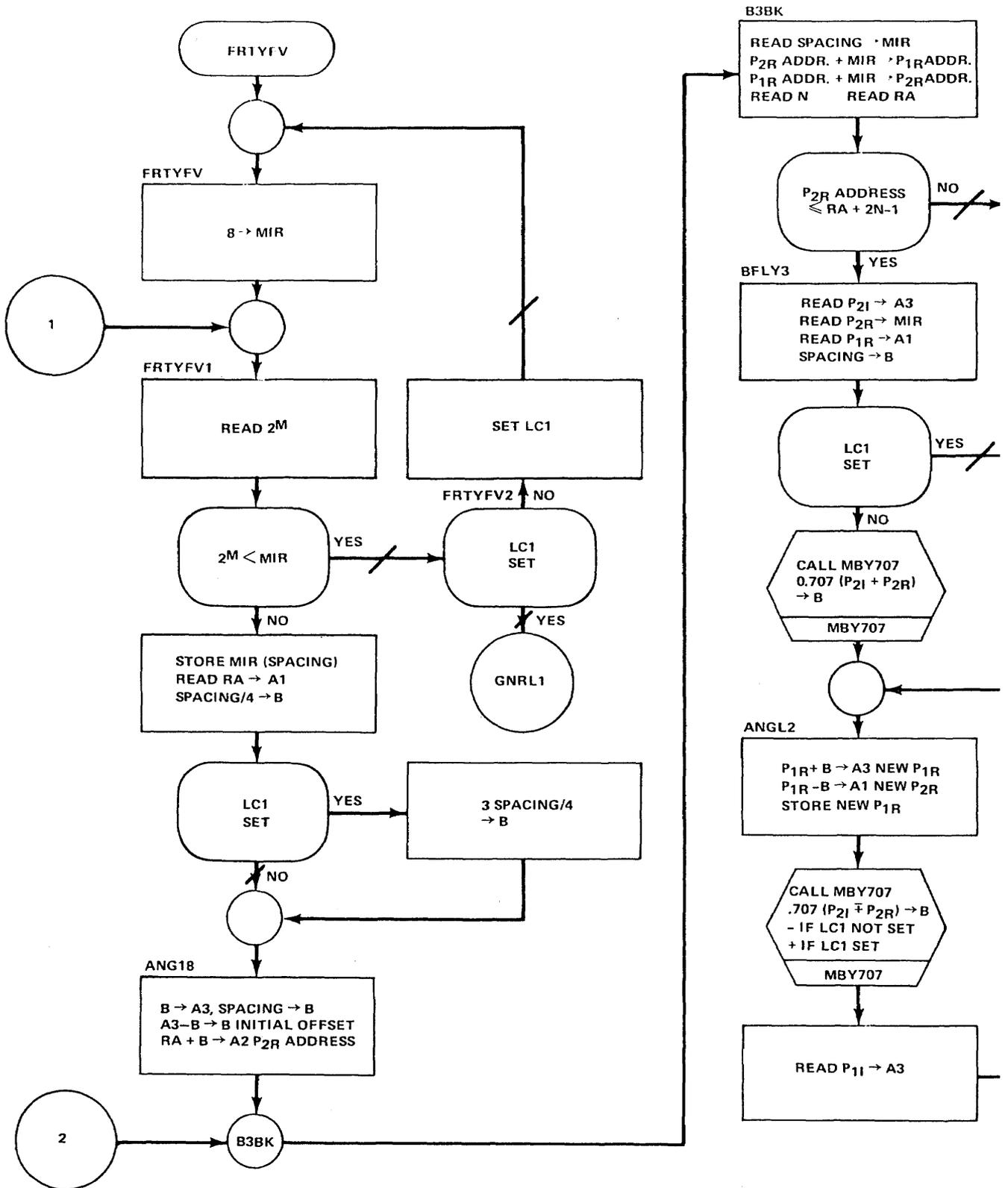
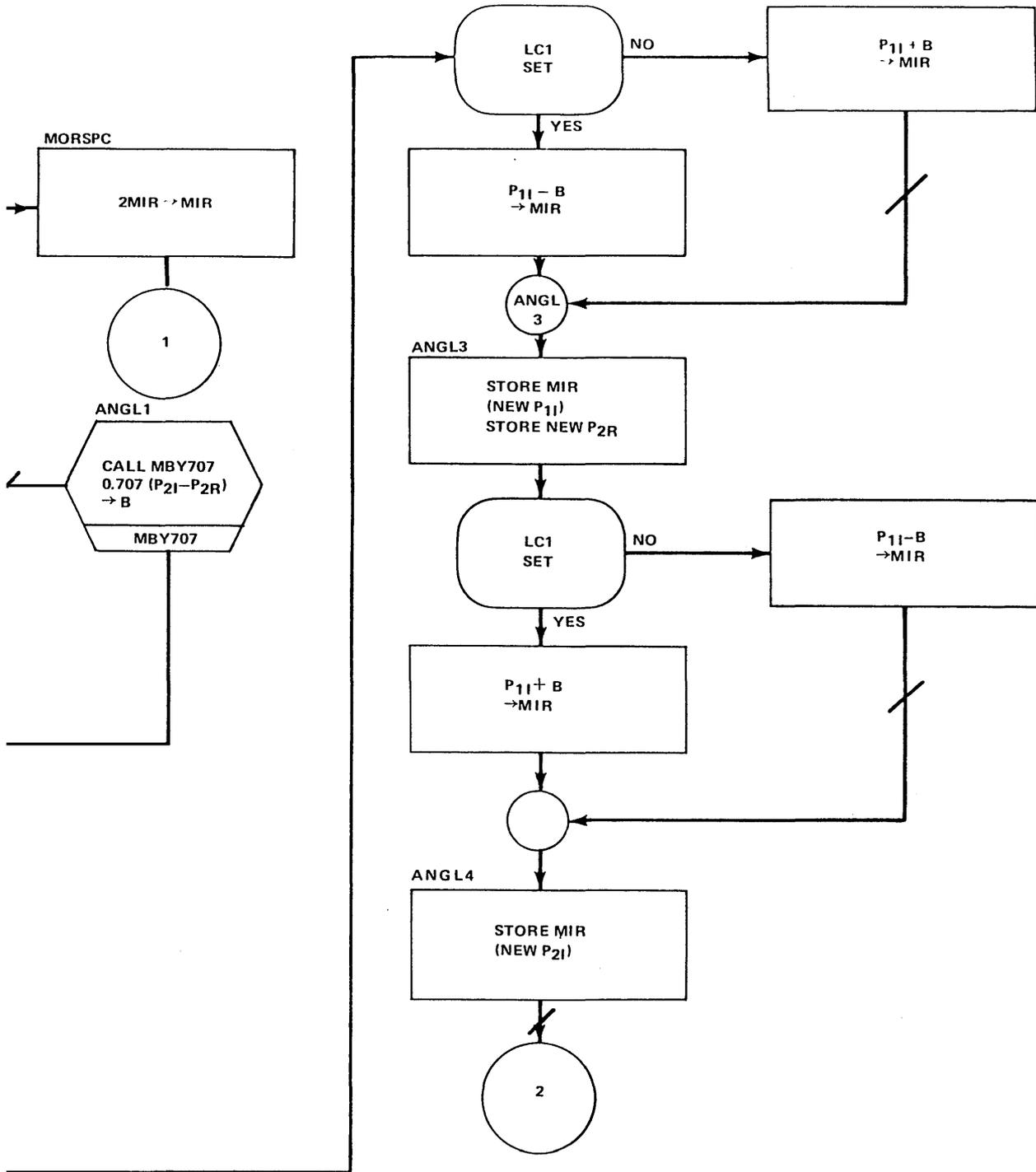


Fig. 2-7 - 45/135-deg



2.7 General Butterfly

The general butterfly (Fig. 2-8) involves angles which are odd multiples of submultiples of 45 degrees. Its first occurrence is in the fourth stage of the FFT calculations, involving multiples of 22.5 degrees. If the initial stage number is denoted by K , then the general angles for that stage are equal to $(2R + 1)2\pi/2^K$, for values of R ranging from 0 through $2^{K-2} - 1$. All of the butterflies for a given angle are calculated in a single group which generally spans several stages. When the M th stage is reached, the angle is incremented by increasing R by one. If the resultant angle is less than $\pi/2$, the calculations are resumed at the K th stage. If the angle exceeds $\pi/2$, then R is reset to zero and K is increased by one. The value of K is held constant, corresponding to the stage number at which an angle first appeared, while the spacing, which is a direct measure of the "stage" of a butterfly, is increased. When K exceeds M , the FFT is complete.

As part of the evaluation of each general butterfly a complex multiplication is performed, as shown in Fig. 2-9. This operation is carried out by performing three real multiplications and five additions, based on the following expansion:

$$\begin{aligned} (R + iI) (\cos\theta - i \sin\theta) &= (R + I) (\cos\theta + \sin\theta) - I \cos\theta \\ &\quad - R \sin\theta + i (I \cos\theta - R \sin\theta). \end{aligned} \tag{2-16}$$

The three multiplications form the terms $R \sin\theta$, $I \cos\theta$, and $(R + I) (\cos\theta + \sin\theta)$, in that order. Because of a shortage of working registers, the $R \sin\theta$ term is held in temporary storage in the S memory during this routine. And, because of a lack of sign extension on right shifts, it is necessary to test the MSB's of the real and imaginary products and extend the signs after rescaling with the aid of an OR operation in the case of negative products. Also, the previous value of the real part of point 1 must be saved in temporary storage during the butterfly calculation.

2.8 Sine/Cosine Lookup

This routine looks up the cosine and the sine of the current angle in the general butterfly and stores them at locations 7942 and 7943, respectively (see Fig. 2-10). For angles less than 90 degrees, the sine is equal to the table entry located at the starting address 255 (which corresponds to an angle of zero degrees), plus an index increment equal to the angle itself. The cosine of the same angle is obtained by indexing the starting address by $N/4$ (which corresponds to an angle of 90 degrees) minus the angle. If the angle is greater than 90 degrees, an index of $N/2$ minus the angle yields the sine and an index equal to the angle minus $N/4$ yields the cosine. In the last-named case, the cosine is complemented to a negative value before being stored.

Whenever a new stage of the FFT calculation is entered, it is necessary first to check whether this stage number exceeds M before proceeding. The stage is advanced by incrementing K . Before storing the updated K , it is compared to M , and if it does not exceed M the value of R is reset to zero and a new initial angle is calculated for the new stage. When K exceeds M the FFT program is finished.

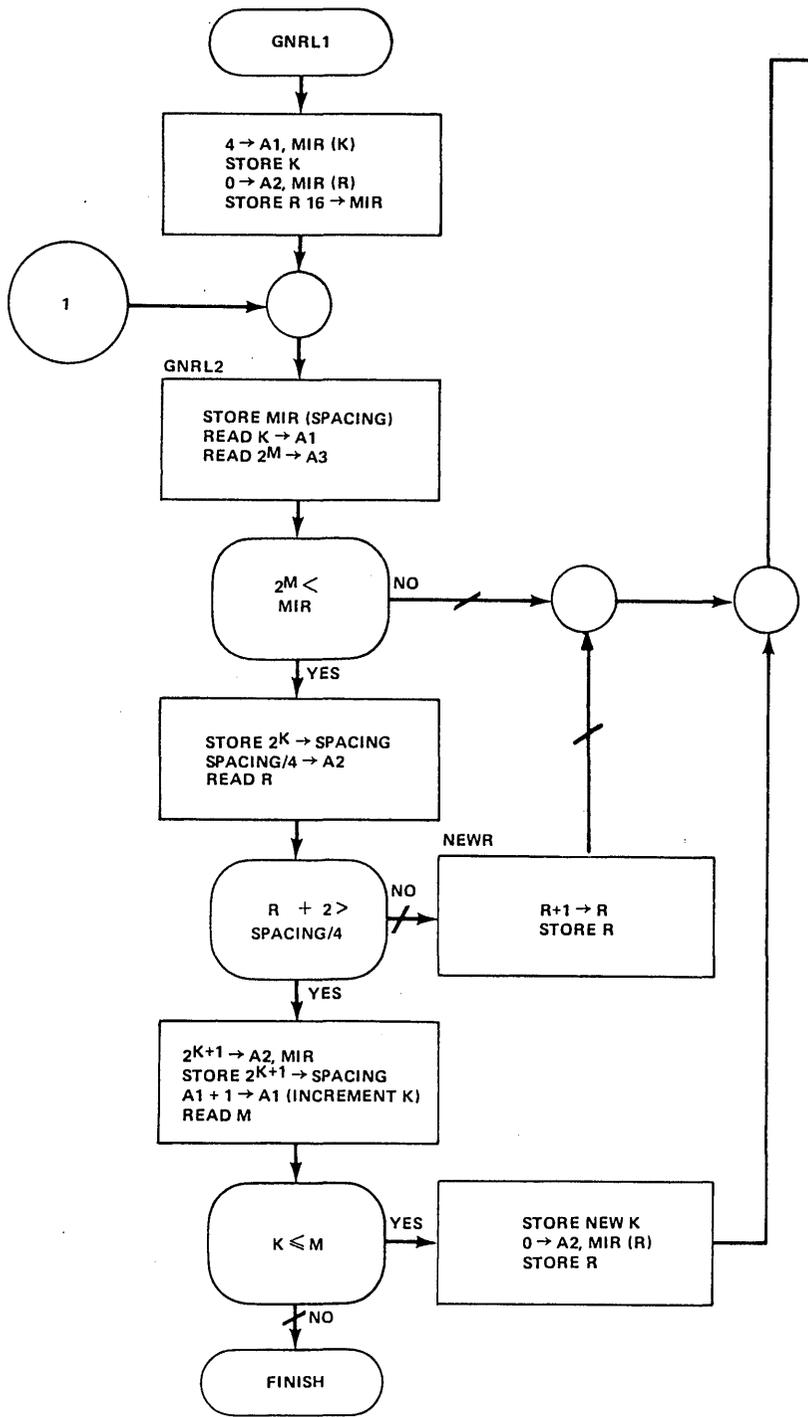


Fig. 2-8

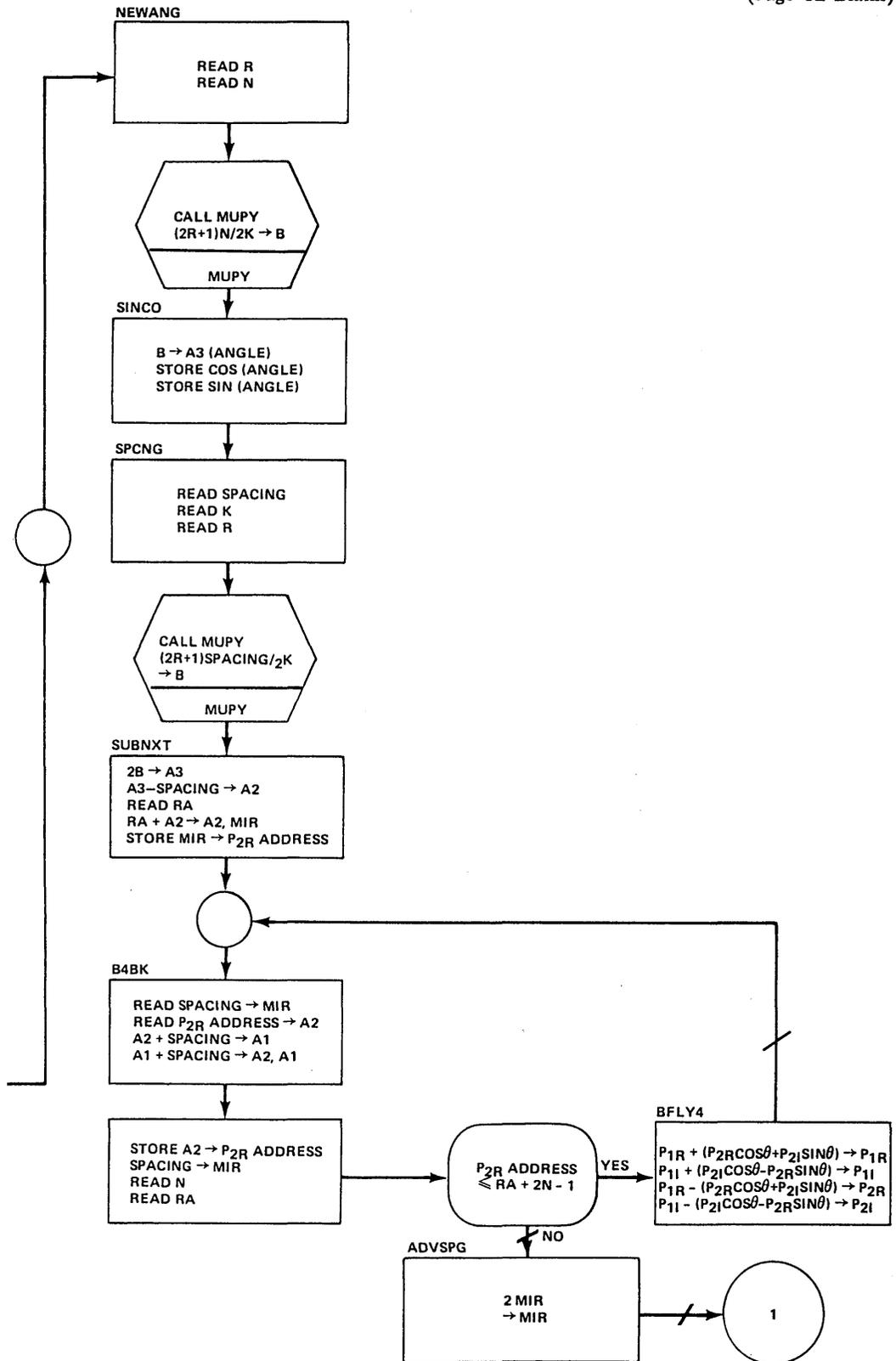


Fig. 2-8 — General butterfly

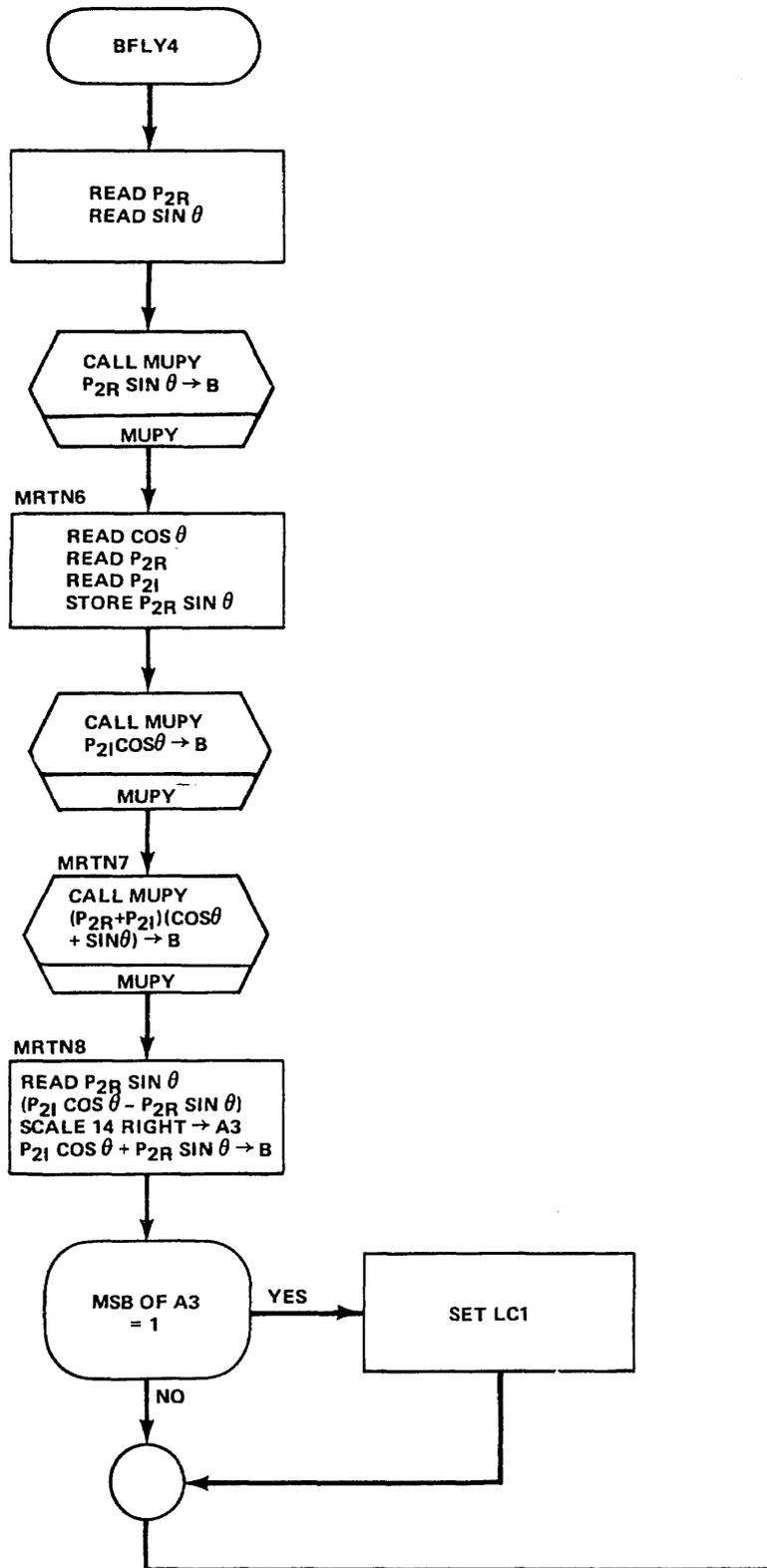
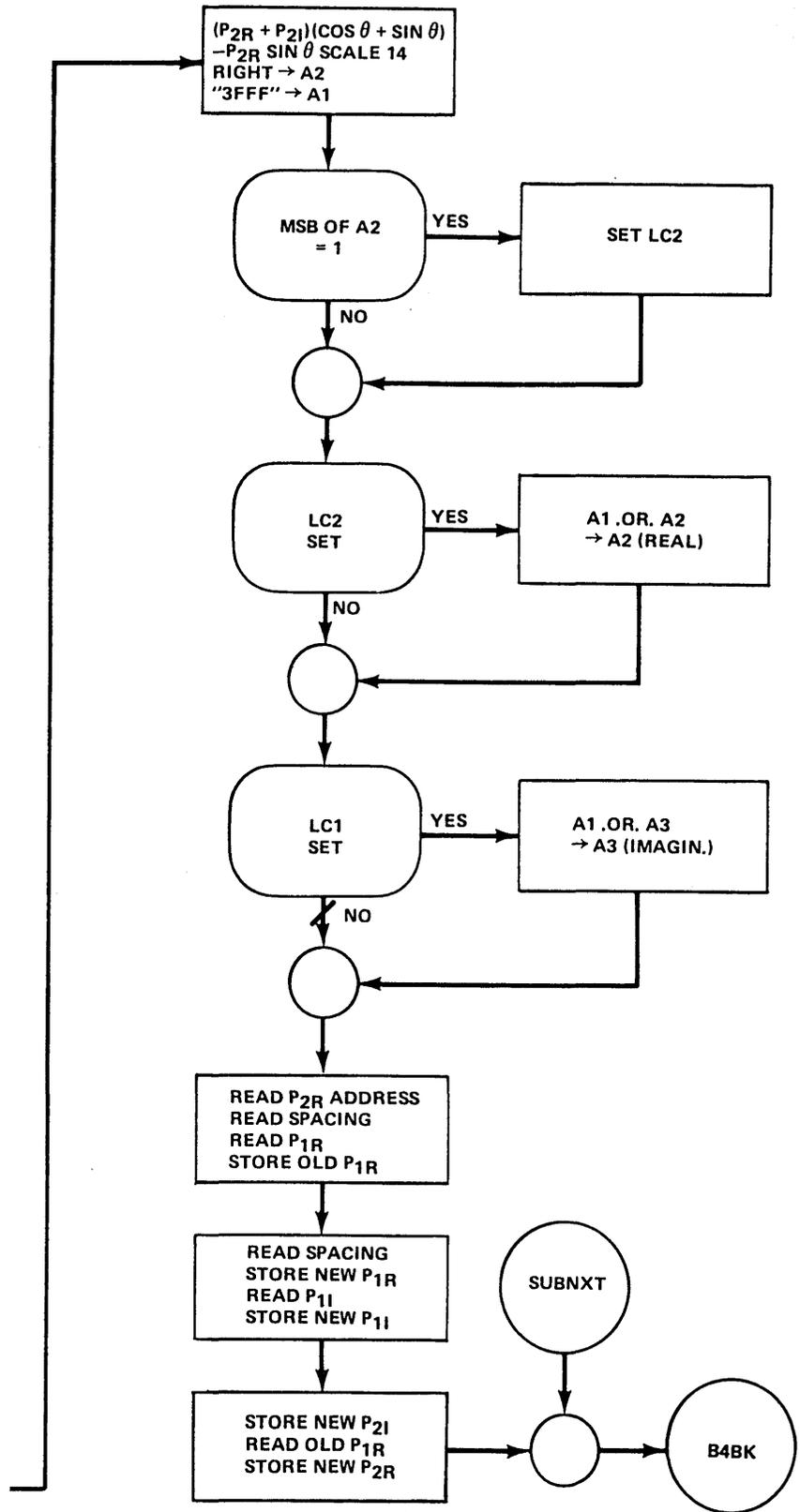


Fig. 2-9 — Complex



complex multiplication

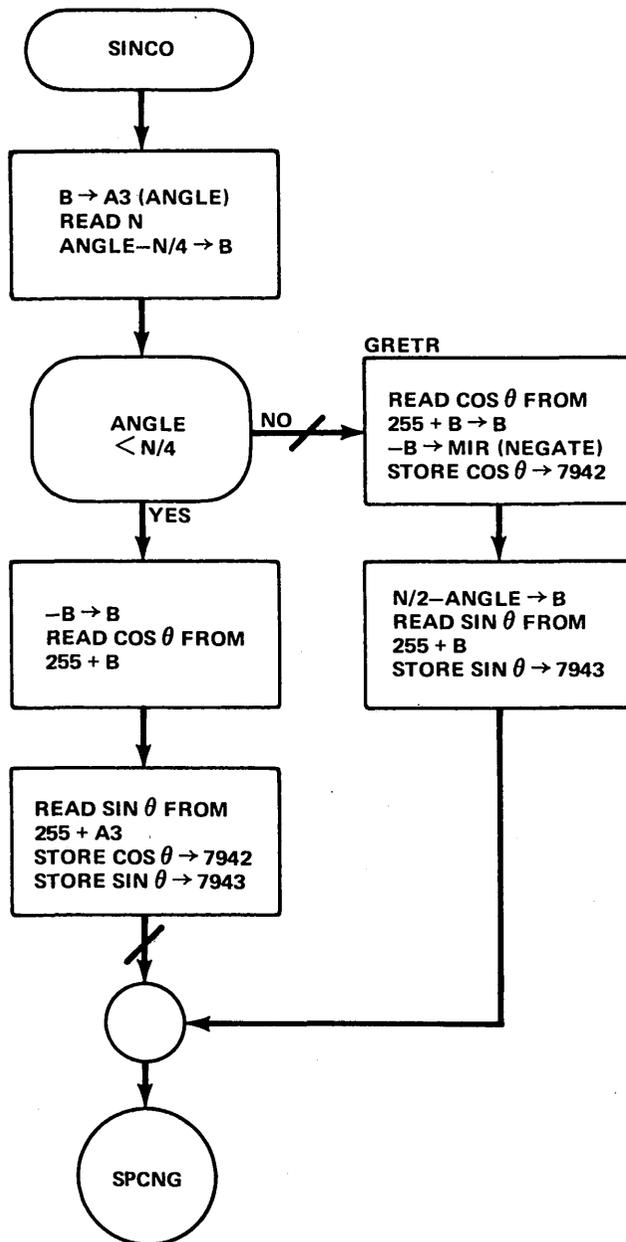


Fig. 2-10 — Sine/cosine lookup

2.9 Real Multiplication

The shift-and-add multiplication of two real 16-bit numbers is coded as shown in the flowchart of Fig. 2-11. The signs of the multiplier in the A3 register and the multiplicand in the B register are tested, and if either one is negative, it is complemented and a condition flag is set. The two positive factors are multiplied by repeatedly shifting the multiplier to the right and testing its LSB for a value of zero. An LSB of zero causes the intermediate product being formed in the B register to be shifted circularly one place right. An LSB of one causes the multiplicand to be added to the contents of B before the circular shift. The initial counter setting of 241 results in the loop being executed 16 times for counter values of 241 through 256 (i.e., 0). After the last pass, the condition flags are tested and the product is complemented if the conditions are not both the same. The routine then looks up the return address, at location 7953, and jumps to this address with the result in the B register.

2.10 Overflow, Real Data, and the Inverse Transform

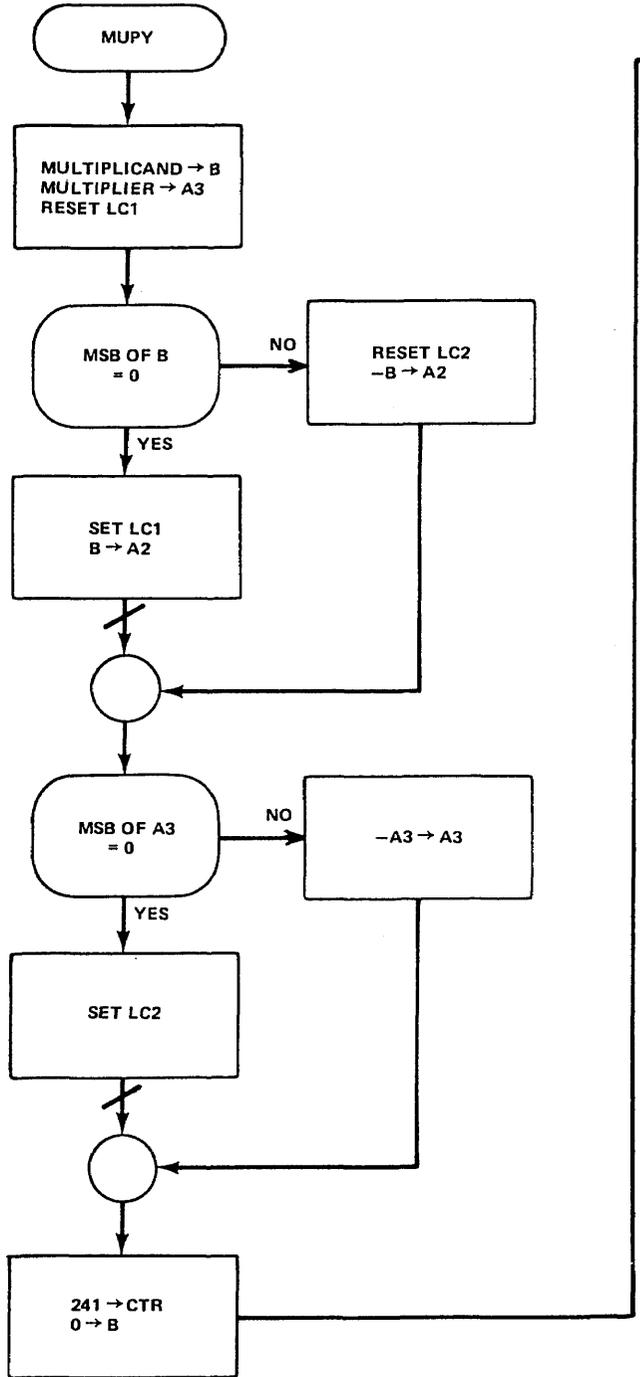
The program, as written, contains no provision for dealing with overflows in the arithmetic operations. Neither does it include a $1/N$ factor which usually appears in the transform definition, Eq. (1-1). If the entire data array is divided by two prior to each stage of calculation, overflow will be prevented and the scale factor can be introduced at the same time. Alternatively, conditional array scaling can be used to improve the accuracy of the calculations and prevent overflow by checking for data values which will indeed cause overflow and rescaling the array only when such values appear.

When the input data are purely real, the efficiency of the FFT calculation can be increased (compared to simply setting the imaginary components of the input data to zero) by packing alternate input values into the real and imaginary locations of the Data Area, and performing an N -point complex FFT on $2N$ real points, as described in Appendix B. A subroutine for performing this function has been coded but not debugged; therefore, it is not included in the present program. The coding comprises 93 instructions.

The only essential difference between the direct and inverse discrete Fourier transforms is a change in the sign of the exponent angle in the complex weighting factor. When an algorithm is suitably coded, it can be used for either procedure by changing angle increments to decrements or conversely. In the subject program, provision could be made to change the butterfly formulas according to a direct/inverse flag, or the direct transform could be used with a simple component combining routine, as described in Appendix C, to generate the inverse transform of conjugate symmetric coefficients. Neither of these procedures has been programmed.

3. TEST RESULTS

The FFT program was used to calculate a number of test spectra varying in sample size from 16 through 128 complex points. Some of the results appear in Table 1-1, which shows the number of clock cycles required to perform each of the major functions and also summarizes their required processing times. Some of the inputs and their output spectra are listed in Figs. 3-1 through 3-6. They verify the symmetry relationships of the Fourier transform.



In Fig. 3-1, the input comprises 64 points representing one cycle of a sawtooth wave which is a real, odd function with an average value of zero. The output consists of consecutive spectral lines of alternating sign which show an approximate $1/n$ envelope: 1300, 650, 433, 325, etc. The fundamental amplitude 1302 compares well with the value 1260, corresponding to a continuous waveform. The nonzero real terms indicate the magnitude of the computational error.

Figures 3-2 and 3-3 show the spectrum of a sine wave whose period is one-fourth of the measurement period. The real, odd input yields an imaginary, odd output. In Fig. 3-3, the symmetry is reflected about the $N/2$ point rather than the N point, as in spectra which are carried to completion. Figure 3-3 shows two estimates of the spectral lines.

Input

Real part:

0	2	4	6	8	10	12	14	16	18	20	22	24	26	28	30
32	34	36	38	40	42	44	46	48	50	52	54	56	58	60	62
0	-62	-60	-58	-56	-54	-52	-50	-48	-46	-44	-42	-40	-38	-36	-34
-32	-30	-28	-26	-24	-22	-20	-18	-16	-14	-12	-10	-8	-6	-4	-2

Imaginary part all zero.

Output

Real part:

0	0	-3	-3	-1	-7	-3	-4	0	-1	0	1	-1	-1	-1	-2
0	0	1	1	0	-1	0	-1	0	0	0	0	0	0	0	-1
0	0	1	1	1	1	1	0	0	1	0	1	1	1	1	0
0	0	1	1	0	3	2	1	0	0	0	-2	0	4	0	7

Imaginary part:

0	-1302	646	-434	320	-257	210	-168	154	-137	120	-108	95	-88	77	-71
64	-58	52	-49	43	-40	33	-31	26	-23	19	-16	12	-11	6	-4
0	4	-6	10	-12	15	-20	22	-26	31	-34	40	-43	48	-53	57
-64	72	-76	89	-95	106	-119	135	-154	181	-209	260	-32	435	-646	1302

Fig. 3-1 — SAW 164, Sawtooth, 1 cycle, 64 points

Input

Real part:

0	71	100	71	0	-71	-100	-71	0	71	100	71	0	-71	-100	-71
0	71	100	71	0	-71	-100	-71	0	71	100	71	0	-71	-100	-71

Imaginary part all zero.

Output

Real part all zero.

Imaginary part:

0	0	0	0	-1603	0	0	0	0	0	0	0	0	-3	0	0	0
0	0	0	0	3	0	0	0	0	0	0	0	0	1603	0	0	0

Fig. 3-2 — SIN 432, Sinusoid, 4 cycles, 32 points, $M = 5$

Output

Real part all zero.

Imaginary part:

```

0 0 -801 0 0 0 -1 0 0 0 1 0 0 0 801 0
0 0 -801 0 0 0 -1 0 0 0 1 0 0 0 801 0

```

Fig. 3-3 — SIN 432, Sinusoid, 4 cycles,
32 points, $M = 4$

Input

Real part:

```

1000 995 981 957 924 882 832 773
 707 634 556 471 383 290 195 98
  0  98 195 290 383 471 556 634
 707 773 832 882 924 957 981 995

```

Imaginary part all zero.

Output

Real part:

```

20356 6804 -1374 595 -341 222 -160 120
 -100  84  -71  64  -59  57  -56  49
 -44  50  -52  57  -59  64  -70  84
 -100 122  -159 224  -341 597  -1373 6807

```

Imaginary part:

```

0 -2 -1 -3 0 -4 0 -2
0 -1 0 -1 0 0 -1 -1
0 0 1 -1 0 0 0 0
0 3 0 5 0 4 1 3

```

Fig. 3-4 — REC 132, Rectified cosinusoid,
1 cycle, 32 points

In Fig. 3-4, one-half cycle of a cosine wave is rectified and sampled 32 times for the input. The real, even input yields a real, even output, and the numerical results, 20356, 6806, 1374, 596, etc., compare well with the values 20373, 6791, 1358, 654, corresponding to a continuous waveform.

In Fig. 3-5, the real part of the input represents one cycle of a cosine waveform and the imaginary part represents two cycles of a sine wave. Their separate spectra can be obtained from the spectrum $A(n)$ of the complex input by using the relations

$$A_{\text{real}}(n) = 1/2 [A(n) + A^*(N-n)] \quad (3-1)$$

$$A_{\text{imag}}(n) = \frac{1}{2i} [A(n) - A^*(N-n)], \quad (3-2)$$

where * denotes the complex conjugate. When these equations are applied to the $A(n)$ of Fig. 3-5, the individual spectral lines appear at their correct frequencies.

Input

Real part:

100	98	92	83	71	56	38	20	0	-20	-38	-56	-71	-83	-92	-98
-100	-98	-92	-83	-71	-56	-38	-20	0	20	38	56	71	83	92	98

Imaginary part:

0	38	71	92	100	92	71	38	0	-38	-71	-92	-100	-92	-71	-38
0	38	71	92	100	92	71	38	0	-38	-71	-92	-100	-92	-71	-38

Output

Real part:

0	1595	1596	-4	0	0	0	2	0	2	-2	-2	0	3	-6	-2
0	-1	6	4	0	-2	2	2	0	4	0	2	0	-1	-1596	1598

Imaginary part:

0	-1	-1	-3	0	-2	0	-1	0	0	-1	-1	0	-1	0	1
0	-1	1	1	0	0	0	-1	0	2	1	3	0	3	0	1

Fig. 3-5 — CS 32, One-cycle cosine/two-cycle sine, 32 points

Figure 3-6 illustrates the spectrum of a pulse (actually a square wave) which has a duty cycle of 75%. The magnitudes of the spectral lines are in close agreement with the $(\sin x)/x$ envelope expected for a pulse. The average value of 2400 is the product of the pulse amplitude, the duty cycle, and the number of points.

4. SUITABILITY OF THE D-MACHINE TO SIGNAL PROCESSING

The FFT test program reveals several deficiencies in the D-Machine as a signal processor. Despite its use of 54 bits in the control nano instructions, the effective concurrency of operations is low. Part of the reason for this is the scarcity of working registers (A1, A2, A3, B, and MIR) which could be used to hold intermediate results and to perform logic unit operations during memory accesses. As it is, many memory accesses occur as isolated operations; with more working registers, they could be run concurrently with LUOP's. And in some instances, such as the complex multiplication of B4BK through MRTN8, intermediate results are stored in S memory because there are not enough working registers available.

In a similar manner, there is a scarcity of registers for holding program branch addresses. The single AMPCR is employed in branches within subroutines, while the calling routine's return address is stored in S memory. When the subroutine does not disturb the contents of MIR and these contents are not being saved as intermediate results, the return address may be saved in MIR without writing it in S memory. However, this may require some register juggling at the end of the subroutine because MIR can be loaded only into B and B frequently (as in the case of MUPY) contains the result of the subroutine computations.

Input

Real part:

100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100
100	100	100	100	100	100	100	100	100	0	0	0	0	0	0	0	0

Imaginary part all zero.

Output

Real part:

2400	-455	100	213	0	-47	100	108
0	10	99	76	0	34	99	54
0	55	100	35	0	77	100	10
0	110	101	-44	0	216	101	-452

Imaginary part:

0	-558	-502	-113	0	-144	-150	-12
0	-91	-68	24	0	-66	-20	44
0	-44	20	65	0	-24	68	90
0	13	150	144	0	144	502	558

Spectral Magnitudes

2400	1438	1022	485	0	301	358	220
0	184	246	160	0			

Normalized Magnitudes

1.0	0.598	0.425	0.205	0	0.125	0.149	0.092
0	0.077	0.102	0.067	0			

[Sin(3nπ/4)]/(3nπ/4) Envelope

1.0	0.600	0.424	0.200	0	0.120	0.141	0.086
0	0.067	0.085	0.055	0			

Fig. 3-6 — P 2432, Pulse, 75% duty cycle, 32 points

The concurrency is also reduced by the division of the microinstructions into Types I and II, which are mutually exclusive, and the separation of the Type II instructions into those which load the AMPCR and those which load the SAR and/or LIT. The prohibition of branches to Type II instructions, the lack of sign extension on right shifts, and the constraints on the allowable X and Y inputs to the adder were also found to require excess coding. The availability of only two local condition flags was found to be sufficient.

All of these deficiencies are minor, however, compared to the fact that the D-Machine operates in an essentially serial manner. The performance figures cited in Table 1-1 can be improved by orders of magnitude by using high-speed memories in a parallel processing structure (1).

REFERENCES

1. B. Shay, "Design Considerations of a Programmable Predetection Signal Processor for Radar Applications," NRL Report 7455, Dec. 13, 1972.
2. J.D. Roberts, Jr., "Microprogrammed Control Unit (MCU) Programming Reference Manual," NRL Report 7476, Aug. 15, 1972.
3. W.R. Smith and J.P. Ihnat, "Signal Processing Element Users' Reference Manual," NRL Report 7488, Sept. 5, 1972.
4. J.P. Ihnat, W.R. Smith, J.R. Roberts, Jr., Y.S. Wu, and B. Wald, "Signal Processing Element Functional Description. Part 1, Microprogram Control Unit, Buffer Storage, and Storage Control Unit," NRL Report 7490, Sept. 12, 1972.
5. E.W. Reigel, U. Faber, and D.A. Fisher, "The interpreter — A microprogrammable building block system," Spring Joint Computer Conference, AFIPS Conf. Proc. 40, 705-723 (1972).
6. Y.S. Wu, "Architectural Considerations of a Signal Processor Under Microprogram Control," Spring Joint Comp. Conf. AFIPS Conf. Proc. 40, 675-683 (1972).
7. B. Gold, I.L. Lebow, P.G. McHugh, and C.M. Rader, IEEE Trans. Comput. C-20 (No. 1), 33-38 (Jan. 1971).
8. A.S. Zukin and S.Y. Wong, "Architecture of a Real-Time Fast Fourier Radar Signal Processor," Spring Joint Comput. Conf., AFIPS Conf. Proc. 36, 417-435 (1970).
9. "D-Machine Users' Manual," TR 70-8, Burroughs Corporation, Space Systems Division, Paoli, Pa., Apr. 1971.
10. J.W. Hartwell, "A Procedure for Implementing the Fast Fourier Transform on Small Computers," TR 54.017, IBM General Systems Division, Boca Raton, Fla., Mar. 1, 1971.
11. C. Hastings, Jr., *Approximations for Digital Computers*, Princeton University Press, Princeton, N.J., 1955.
12. B. Gold and C.M. Rader, *Digital Processing of Signals*, McGraw-Hill, New York, 1969.

Appendix A

PROGRAM LISTING

```

FFT. LIT L = BR2 $
  31 = LIT,COMP 8 = SAR $
  0 = MAR IF LC1 THEN STEP ELSE STEP $ RESET LC1
  MR2 LCTR $ TO GET LOG N SCAN N FOR FIRST '1'
  12 = LIT,1 = SAR $
  WHEN RDC THEN BEX $
LOGNCK. IF NOT LST THEN SET LC1 INC $
  SINTAB - 1 = AMPCR $
  IF LC1 THEN STEP ELSE JUMP $
  LOGNCK - 1 = AMPCR $
  B R = B IF NOT COV THEN JUMP ELSE STEP $
  FINISH - 1 = AMPCR $
  JUMP $
SINTAB. CTR - LIT = MIR A1 $ LOG N = CTR - 244
  244 = LIT $
  LMAR $
  4 = LIT,COMP 12 = SAR $ BINARY PT. AT 20/21
  MW2 $ STORE LOG N
  LIT L = A2 $
  WHEN RMI THEN BMAR + 1 = MAR2 $
  1 = MIR $
  MW2 $ STORE INDEX,I
  A1 = SAR B $ TO SHIFT RIGHT BY LOG N
  A2 R = A1 $ ANGLE ELEMENT=4/N
  FINISH - 1 = AMPCR $
  WHEN RMI THEN LIT - B = B $
  2 = LIT $
  IF NOT AOV THEN LMAR STEP ELSE JUMP $ IF LOG N< OR =2
  16 = LIT $
  A1 = MIR $
  BITREV - 1 = AMPCR $ ***** THIS JUMP SKIPS THE
  JUMP $ SINE TABLE GENERATION *****
  MW2 $
  WHEN RMI THEN STEP $ SAVE X, PT. AT 20/21
LOOP1. STEP $ CURRENT ANGLE IS IN A1, PT. AT 20/21
  MRTN1 - 1 = AMPCR $
  AMPCR = MIR LMAR $
  17 = LIT $
  MW2 $
  MUPY - 1 = AMPCR $
  WHEN RMI THEN A1 = A3 B MIR CALL $ FORM X**2,PT. AT 8/9

```

MRTN1. LMAR \$
 18 = LIT,10 = SAR \$
 MW2 \$ SAVE CURRENT ANGLE, PT. AT 20/21
 WHEN RMI THEN B R = B MIR LMAR \$ RESTORE PT. TO 18/19
 13 = LIT \$
 MW2 \$ SAVE X**2, PT. AT 18/19
 MRTN2 - 1 = AMPCR \$
 WHEN RMI THEN AMPCR = MIR LMAR \$
 17 = LIT \$
 MW2 \$ SAVE RETURN ADDRESS
 MUPY - 1 = AMPCR \$
 WHEN RMI THEN B = A3 CALL \$ FORM X**4, PT. AT 4/5
 MRTN2. B R = B \$ RESTORE PT. TO 18/19
 14 = SAR \$
 LIT L = A3 \$
 18 = LIT,COMP 8 = SAR \$ 0000.1200 HEX., PT. AT 16/17
 A3 OR LIT R = A3 \$
 157 = LIT,2 = SAR \$ 0000.129D, PT. AT 18/19
 MRTN3 - 1 = AMPCR \$
 AMPCR = MIR LMAR \$
 17 = LIT \$
 MW2 \$
 MUPY - 1 = AMPCR \$
 WHEN RMI THEN CALL \$ FORM C5X**4, PT. AT 4/5
 MRTN3. B R = A1 LMAR \$ PARTIAL SUM TO A1, PT. AT 18/19
 13 = LIT,14 = SAR \$
 MR2 \$ RECALL X**2
 LIT L = A3 \$
 164 = LIT,COMP 8 = SAR \$ 0000.A400 HEX
 A3 OR LIT R = A3 \$
 171 = LIT,2 = SAR \$ 0000.A4AB, PT. AT 18/19
 MRTN4 - 1 = AMPCR \$
 AMPCR = MIR LMAR \$
 17 = LIT \$
 WHEN RDC THEN BEX MW2 \$
 MUPY - 1 = AMPCR \$
 WHEN RMI THEN CALL \$ FORM C3X**2, PT. AT 4/5 IN A3
 MRTN4. B R = B \$
 14 = SAR \$
 A1 - B = A1 \$ PARTIAL SUM
 LIT L = A3 \$
 1 = LIT,COMP 8 = SAR \$
 A3 OR LIT L = A3 \$
 146 = LIT \$
 A3 OR LIT R = B \$
 20 = LIT,2 = SAR \$ 0001.9214, PT. AT 18/19
 A1 + B = A3 LMAR \$ FINAL SUM
 18 = LIT \$
 MR2 \$ RECALL CURRENT ANGLE VALUE
 MRTN5 - 1 = AMPCR \$
 WHEN RDC THEN AMPCR = MIR LMAR BEX \$
 17 = LIT \$

```

MW2 B = A1 $ SAVE RETURN ADDRESS
MUPY - 1 = AMPCR $ TOTAL ANGLE IS IN A1
WHEN RMI THEN CALL $ FORM SINE VALUE, PT. AT 6/7
MRTN5. B R = MIR LMAR $
Ø = LIT,12 = SAR $ MOVE PT. TO 18/19
MR2 $ READ N
WHEN RDC THEN BEX $
B = A3 LMAR $ N TO A3
5 = LIT $
MR2 $ READ INDEX
WHEN RDC THEN BEX $
LIT = A2 $ FIRST NON-ZERO ENTRY IS AT 256
255 = LIT,COMP 8 = SAR $
A2 + B = MAR1 $ CURRENT ADDRESS IN TABLE
MW1 $ STORE SINE VALUE
WHEN RMI THEN B + 1 = A2 MIR LMAR $ INCREMENTED I
5 = LIT $
MW2 A3 R = B $ STORE INDEX, I
2 = SAR $ TO GET N/4
BITREV - 1 = AMPCR $
A2 - B = B $
IF NOT AOV THEN LMAR STEP ELSE JUMP $
16 = LIT $
MR2 $
LOOP1 - 1 = AMPCR $
WHEN RDC THEN BEX $ RECALL ANGLE ELEMENT, 4/N
A1 + B = A1 JUMP $ INCREMENT ANGLE
MUPY. B = B IF LC1 THEN STEP ELSE STEP $RESET LC1
IF NOT MST THEN B = A2 SET LC1 SKIP ELSE STEP $ TEST B SIGN
Ø - B = A2 IF LC2 THEN STEP ELSE STEP $ COMPLEMENT B. RESET LC2
A3 = A3 B $
IF NOT MST THEN SET LC2 SKIP ELSE STEP $ TEST A3 SIGN
Ø - B = A3 $COMPLEMENT MULTIPLIER
Ø = B LCTR $
1 = SAR,14 = LIT $
A3 R = A3 SAVE $ TEST MULTIPLIER BIT
IF NOT LST THEN B C = B SKIP ELSE STEP $
A2 + B C = B $
INC IF NOT COV THEN STEP ELSE SKIP $
A3 R = A3 JUMP $
B C = B $
16 = SAR $
ENDMUL - 1 = AMPCR $
IF LC1 THEN STEP ELSE SKIP $
IF NOT LC2 THEN SKIP ELSE JUMP $
IF LC2 THEN STEP ELSE JUMP $
Ø - B = B $
ENDMUL. LMAR $
17 = LIT $
MR2 $
B = A2 $ SAVE PRODUCT
WHEN RDC THEN BEX $
B = AMPCR $

```

```

A2 = B JUMP $ TO RETURN ADDRESS
BITREV. 0 = A1 A3 $ A1 HOLDS TABLE INDEX, A3 IS ENTRY (BIT-REVERSED
LIT L = BR1 LMAR $ BITREV TABLE STARTS AT 512 NUMBER)
COMP 9 = SAR, 1 = LIT $
MR2 $ READ M
WHEN RDC THEN 1 = A2 BEX $
B = SAR $
CSAR $
A2 L = A2 MIR LMAR $
14 = LIT $
MW2 $ STORE 2**M
WHEN RMI THEN 0 = MAR $
RVCTR1. MR1 A3 = MIR $ A3 HOLDS THE ENTRY; MR1 SETS UP BMAR
WHEN RDC THEN A1 + BMAR = MAR1 $
MW1 A1 + 1 = A1 $ STORE ENTRY, INCREMENT INDEX
WHEN RMI THEN LMAR $
14 = LIT $
MR2 $ READ 2**M
WHEN RDC THEN BEX $
B = A2 $
A2 - 1 = B $
RVCTR4 - 1 = AMPCR $
A3 EQV B = $ ENTRY=(2**M)-1 ?
IF NOT ABT THEN LCTR STEP ELSE JUMP $
32 = LIT, 1 = SAR $
RVCTR2. A2 R = B $ MOVE MASK BIT TO RIGHT
A3 AND B = A2 $ "AND" ENTRY WITH MASK
RVCTR2 - 1 = AMPCR $ FIND POSITION OF 1ST ZERO BIT
A2 EQV 0 = $ START WITH MSB (J=1)
IF NOT ABT THEN INC JUMP ELSE STEP $
RVCTR3. MR2 CTR - LIT = SAR A2 $
222 = LIT $
WHEN RDC THEN 0 = MAR BEX $
B R = B $
A3 + B = A3 SAVE $ ADD 2**(M-J) TO OLD ENTRY
A2 EQV B001 = $
IF NOT ABT THEN SET LC1 SKIP ELSE STEP $
RVCTR1 - 1 = AMPCR $
IF LC1 THEN A2 - 1 = A2 STEP ELSE JUMP $
B L = B $
COMP 1 = SAR $ SUBTRACT 2**(M-J) FOR VALUES OF J DOWN TO 1
A3 - B = A3 JUMP $
RVCTR4. B001 + 1 = MAR $
MR2 0 = A2 $
WHEN RDC THEN BMAR + 1 = MAR2 BEX $
MR2 B = A1 $ DATA ADDRESS IN A1
WHEN RDC THEN BEX $ RESULT ADDRESS IN B
MOVDAT - 1 = AMPCR $ PREPARE TO MOVE DATA TO RESULTS AREA
A1 EQV B = $ DATA ADDRESS=RESULTS ADDRESS ?
IF ABT THEN STEP ELSE JUMP $ IF YES, DO BIT REVERSAL
REVPTS. 1 = MAR $
MR2 $ READ M

```

```

WHEN RDC THEN 0 = MAR BEX $
MR2 B = SAR $ READ N
WHEN RDC THEN BEX $
B R = A2 $
A2 + B111 = CTR $
RSET. 0 = A2 $ RESET INDEX, I
LOOP2. LIT L = B $
2 = LIT,COMP 8 = SAR $
A2 + B = MAR1 $
MR1 $ READ BITREV(I)
WHEN RDC THEN A2 L = MIR BEX $ FORM 2I IN MIR
COMP 1 = SAR $
AROUND - 1 = AMPCR $ IF I<BITREV
A2 - B = $ THEN STEP TO BIT REVERSAL
IF NOT AOV THEN B L = B STEP ELSE JUMP $ (B)=2BITREV
A1 + B = A3 BMI $ SS+2BITREV TO A3;2I TO B
A1 + B = A2 MAR1 $ SS+2I (SS=SECTION START ADDRESS)
MR1 $
WHEN RDC THEN A3 = MAR1 BEX $ READ (SS+2I) INTO B & MIR
MR1 B = MIR $
WHEN RDC THEN BEX $ READ (SS + 2BITREV) INTO B
MW1 $ STORE ( SS+2I) IN SS+2BITREV
WHEN RMI THEN B = MIR $
A2 = MAR1 $
MW1 A2 + 1 = A2 $ STORE (SS+2BITREV) IN SS+2I
WHEN RMI THEN A2 = MAR1 $
MR1 A3 + 1 = A3 $
WHEN RDC THEN A3 = MAR1 BEX $ READ (SS+2I+1) INTO B & MIR
MR1 B = MIR $
WHEN RDC THEN BEX $ READ ( SS + 2BITREV + 1) INTO B
MW1 $ STORE (SS+2I+1) IN SS+2BITREV+1
WHEN RMI THEN B = MIR $
A2 = MAR1 $
MW1 A2 - LIT = A2 $ STORE (SS+2BITREV+1)
1 = LIT,1 = SAR $
A1 = B $
WHEN RMI THEN A2 - B R = A2 $ RESTORE I TO A2
AROUND. A2 + 1 = A2 LMAR $ INCREMENT INDEX
14 = LIT,COMP 1 = SAR $
MR2 $
WHEN RDC THEN BEX $ READ 2**M
LOOP2 - 1 = AMPCR $
A2 - B = $ INDEX<2**M ?
IF AOV THEN INC STEP ELSE JUMP $ IF SO, JUMP
ZERDEG - 1 = AMPCR $
B L = B $
IF NOT COV THEN STEP ELSE JUMP $
RSET - 1 = AMPCR $
A1 + B = A1 JUMP $
MOVDAT. B = A3 $ (A3)=RESULTS ADDRESS
RETRN. A2 = B $ (A2)=INDEX, I
A1 + B = MAR1 $ (A1)=DATA ADDRESS; DA+I=SOURCE

```

```

MR1 $
WHEN RDC THEN BEX $
B = MIR $ SOURCE DATA TO MIR
A2 = B $
A3 + B = MARI $ RA+I=DESTINATION
MW1 $ MOVE SOURCE DATA TO DESTINATION
WHEN RMI THEN B + 1 = A2 LMAR $ INCREMENT INDEX
0 = LIT,COMP 1 = SAR $
MR2 $ READ N
RETRN - 1 = AMPCR $
WHEN RDC THEN BEX $
B L = B $
A2 - B = $ IS I < OR =2N-1 ?
IF NOT AOV THEN JUMP ELSE STEP $ IF SO, RETURN
REVPTS - 1 = AMPCR $ IF NOT JUMP
A3 = A1 JUMP $ RESULTS ADDRESS=FIRST SECTION START ADDRESS
ZERDEG. B001 + 1 = MIR $ INITIAL SPACING = 2 MEMORY LNCS
ZERDEG1. LMAR $
14 = LIT $
MR2 $ READ 2**M
NINDEG - 1 = AMPCR $
WHEN RDC THEN BEX $
B = A2 BMI $ (B)=SPACING; (A2)=2**M
A2 - B = $
IF AOV THEN LMAR STEP ELSE JUMP $
11 = LIT $
MW2 $
WHEN RMI THEN LMAR $
3 = LIT $
MR2 $
WHEN RDC THEN BEX $ READ RESULTS ADDRESS
B = A1 BMI $
A1 - B = A2 $ INITIALIZE P2R TO RA-SPACING
B1BK. LMAR $ RECALL SPACING
11 = LIT $
MR2 $
WHEN RDC THEN BEX $
B = MIR $ SAVE SPACING FOR ADVSPC
A2 + B = A1 $ ADD SPACING TO OLD P2R ADDRESS
A1 + B = A2 LMAR $ ADD SPACING TO NEW P1R ADDRESS
0 = LIT,COMP 1 = SAR $
MR2 $
WHEN RDC THEN BEX $ READ N
B L = A3 LMAR $ CHECK IF NEW P2 < OR=2N-1 BY
3 = LIT $ USING P2-(2N-1)-1 = P2-2N
MR2 $ READ RESULTS ADDRESS
WHEN RDC THEN BEX $
A3 + B = B $ (B)= RA+2N-1
ADVSPC - 1 = AMPCR $
A2 - B = $ IF P2R ADDRESS < OR=(B), STEP
IF NOT AOV THEN STEP ELSE JUMP $ TO BUTTERFLY
BFLY1. A1 = MARI $ P1R ADDRESS

```

```

MR1 $
WHEN RDC THEN A2 = MAR1 BEX $ P2R ADDRESS
MR1 B = A3 $ P1R TO A3
WHEN RDC THEN BEX $
A3 - B = MIR $
MW1 $ NEW P2R = P1R-P2R
WHEN RMI THEN A3 + B = MIR $
A1 = MAR1 $
MW1 $ NEW P1R = P1R+P2R
WHEN RMI THEN A1 + 1 = MAR1 $
MR1 $ READ P1I
WHEN RDC THEN A2 + 1 = MAR1 BEX $
MR1 B = A3 $ P1I TO A3
WHEN RDC THEN BEX $
A3 - B = MIR $
MW1 $ NEW P2I = P1I-P2I
WHEN RMI THEN A3 + B = MIR $
A1 + 1 = MAR1 $
MW1 $ NEW P1I = P1I+P2I
B1BK - 1 = AMPCR $
WHEN RMI THEN JUMP $
ADVSPC. BMI $
B L = MIR $ NEW SPACING
COMP 1 = SAR $
ZERDEG1 - 1 = AMPCR $
JUMP $
NINDEG. LIT = MIR $
4 = LIT $
NINDEG1. LMAR $
14 = LIT $
MR2 $
FRTYFV - 1 = AMPCR $
WHEN RDC THEN BEX $
B = A2 BMI $
A2 - B = $
IF AOV THEN B R = B LMAR STEP ELSE JUMP $
11 = LIT,1 = SAR $
MW2 $
WHEN RMI THEN B = MIR LMAR $ HALF SPACING TO MIR
3 = LIT $
MR2 $
WHEN RDC THEN BEX $ READ RA
B = A1 BMI $
A1 + B = A1 $ RA+HALF SPACING
B L = B MIR $ RESTORE FULL SPACING
COMP 1 = SAR $
A1 - B = A2 $ INITIALIZED P2R
B2BK. LMAR $
11 = LIT $
MR2 $
WHEN RDC THEN BEX $
B = MIR $ SAVE SPACING FOR MOVSPC

```

```

A2 + B = A1 $
A1 + B = A2 LMAR $
0 = LIT, COMP 1 = SAR $
MR2 $
WHEN RDC THEN BEX $ READ N
B L = A3 LMAR $
3 = LIT $
MR2 $
WHEN RDC THEN BEX $
A3 + B = B $
MOVSPC - 1 = AMPCR $
A2 - B = $ IF P2R ADDRESS < OR = (B), STEP TO
IF NOT AOV THEN STEP ELSE JUMP $ BUTTERFLY
BFLY2. A1 = MAR1 $ P1R ADDRESS
MR1 $
WHEN RDC THEN A2 + 1 = MAR1 BEX $
MR1 B = A3 $ READ P2I
WHEN RDC THEN A1 = MAR1 BEX $
A3 + B = MIR $
MW1 $ NEW P1R = P1R+P2I
WHEN RMI THEN A3 - B = MIR $ NEW P2R = P1R-P2I
A2 = MAR1 $ P2R ADDRESS
MR1 $ READ OLD P2R BEFORE IT IS LOST
WHEN RDC THEN BEX $
MW1 B = A3 $ STORE NEW P2R; (A3)=OLD P2R
WHEN RMI THEN A1 + 1 = MAR1 $ P1I ADDRESS
MR1 $
WHEN RDC THEN A3 = MIR BEX $
B = A3 BMI $ P2R TO B; P1I TO A3
A3 - B = MIR $
MW1 $ NEW P1I = P1I-P2R
WHEN RMI THEN A2 + 1 = MAR1 $ P2I ADDRESS
A3 + B = MIR $
MW1 $ NEW P2I = P1I+P2R
B2BK - 1 = AMPCR $
WHEN RMI THEN JUMP $
MOVSPC. BMI $
B L = MIR $
COMP 1 = SAR $
NINDEG1 - 1 = AMPCR $
JUMP $
FRTYFV. LIT = MIR $
8 = LIT $
FRTYFV1. LMAR $
14 = LIT $
MR2 $
FRTYFV2 - 1 = AMPCR $
WHEN RDC THEN BEX $
B = A2 BMI $
A2 - B = $
IF AOV THEN LMAR STEP ELSE JUMP $
11 = LIT $

```

```

MW2 $
WHEN RMI THEN LMAR $
3 = LIT $
MR2 $
WHEN RDC THEN BEX $
B = A1 BMI $
B R = B A2 $ (B)=SPACING/4
2 = SAR $
ANG18 - 1 = AMPCR $ IF LC1 IS SET, THE
IF LC1 THEN B = A2 STEP ELSE JUMP $ ANGLE IS 3N/8
A2 + B = A2 SET LC1 $
A2 + B = B $ (B)=3SPACING/4
ANG18. B = A3 BMI $ START ADDRESS(=1/4 OR 3/4 SPACING) IN A3
A3 - B = B $ INITIAL P1R OFFSET
A1 + B = A2 $ INITIALIZED P2R ADDRESS
B3BK. LMAR $
11 = LIT $
MR2 $
WHEN RDC THEN BEX $
B = MIR $ SAVE SPACING FOR MORSPC
A2 + B = A1 $
A1 + B = A2 LMAR $
0 = LIT, COMP 1 = SAR $
MR2 $
WHEN RDC THEN BEX $ READ N
B L = A3 LMAR $
3 = LIT $
MR2 $
WHEN RDC THEN BEX $ READ RESULTS ADDRESS
A3 + B = B $ (B)=RA+2N-1
MORSPC - 1 = AMPCR $
A2 - B = $ P2R ADDRESS:(B)
IF NOT AOV THEN STEP ELSE JUMP $ IF P2R ADDRESS < OR =
BFLY3. A2 + 1 = MAR1 $ P2R ADDRESS IN A2
MR1 $ READ P2I
WHEN RDCTHEN A2 = MAR1 BEX $
MR1 B = A3 $ READ P2R
WHEN RDC THEN A1 = MAR1 BEX $
MR1 B = MIR $ P2R TO MIR TEMPORARILY
WHEN RDC THEN BEX $ READ P1R
B = A1 BMI $
ANGL1 - 1 = AMPCR $
IF LC1 THEN JUMP ELSE STEP $
MBY707 - 1 = AMPCR $ ANGLE IS N/8
A3 - B = MIR $
A3 + B = B CALL $
STEP $
ANGL2 - 1 = AMPCR $
JUMP $
ANGL1. SET LC1 $ ANGLE IS 3N/8
MBY707 - 1 = AMPCR $
A3 + B = MIR $

```

```

A3 - B = B CALL $
ANGL2. A1 + B = A3 $ NEW P1R TO A3
A1 - B = A1 $ NEW P2R TO A1
A3 = MIR BMI $
MW1 $ NEW P1R
MBY707 - 1 = AMPCR $
WHEN RMI THEN BMAR + 1 = MAR1 CALL $
MR1 B = MIR $ PRODUCT TO MIR TEMPORARILY
WHEN RDC THEN BEX $ READ P1I
B = A3 BMI $ P1I TO A3;PRODUCT TO B
ANGL3 - 1 = AMPCR $
IF LC1 THEN SKIP ELSE STEP $
A3 + B = MIR JUMP $
A3 - B = MIR SET LC1 $
ANGL3. MW1 $ STORE NEW P1I
WHEN RMI THEN A1 = MIR $
A2 = MAR1 $
MW1 $ STORE NEW P2R
WHEN RMI THEN A2 + 1 = MAR1 $
ANGL4 - 1 = AMPCR $
IF LC1 THEN SKIP ELSE STEP $
A3 - B = MIR JUMP $
A3 + B = MIR SET LC1 $
ANGL4. MW1 $ STORE NEW P2I
B3BK - 1 = AMPCR $
WHEN RMI THEN JUMP $
MBY707. B = A3 $ MULTIPLICAND TO A3
IF MST THEN 0 - B = A3 SET LC2 STEP ELSE SKIP $
0 - B = B $
B R = B $ SHIFT FIRST PARTIAL PRODUCT
1 = SAR $
B R = B $ SECOND BIT OF B5(HEX) IS 0
A3 + B R = B $ THIRD BIT IS A 1
B R = B $
A3 + B R = B $
A3 + B R = B $
B R = B $
A3 + B R = B $
IF LC2 THEN 0 - B = B JUMP ELSE JUMP $
FRTYFV2. STEP $
GNRL1 - 1 = AMPCR $
IF NOT LC1 THEN STEP ELSE JUMP $
FRTYFV - 1 = AMPCR $
SET LC1 JUMP $
MORSPC. BMI $
B L = MIR $
COMP 1 = SAR $
FRTYFV1 - 1 = AMPCR $
JUMP $
GNRL1. LIT R = A1 MIR LMAR $
8 = LIT,1 = SAR $
MW2 $ ANGLE DENOMINATOR EXPONENT, K

```

```

WHEN RMI THEN 0 = A2 MIR LMAR $
9 = LIT,COMP 1 = SAR $
MW2 $ INITIAL NUMERATOR FACTOR, R = 0
WHEN RMI THEN LIT = MIR $
16 = LIT $
GNRL2. LMAR $
11 = LIT $
MW2 $
WHEN RMI THEN LMAR $
8 = LIT $
MR2 $ READ EXPONENT, K
WHEN RDC THEN BEX $
B = A1 SAR LMAR $ SAVE K IN A1
14 = LIT $
MR2 $
NEWANG - 1 = AMPCR $
WHEN RDC THEN BEX $
B = A3 BMI $ (A3)=2**M, (B)=SPACING
A3 - B = $
IF NOT AOV THEN 1 = A2 CSAR LMAR STEP ELSE JUMP $
11 = LIT $
A2 L = A2 MIR $
MW2 $ STORE SPACING=2**K
WHEN RMI THEN LMAR $
9 = LIT $
MR2 A2 R = A2 $ 2**(K-2)
1 = LIT,2 = SAR $
WHEN RDC THEN BEX $
NEWR - 1 = AMPCR $
LIT + B + 1 = B MIR $ NEW R + 1
A2 - B = $
IF NOT AOV THEN A2 L = A2 MIR LMAR STEP ELSE JUMP $
11 = LIT,COMP 3 = SAR $
MW2 $
WHEN RMI THEN A1 + 1 = A1 MIR LMAR $
1 = LIT $
MR2 $
FINISH - 1 = AMPCR $
WHEN RDC THEN BEX $
A1 - B - 1 = $
IF NOT AOV THEN LMAR STEP ELSE JUMP $
8 = LIT $
MW2 $ STORE UPDATED K
WHEN RMI THEN 0 = A2 MIR LMAR $
9 = LIT,COMP 1 = SAR $
MW2 $ SET R=0
WHEN RMI THEN STEP $
NEWANG. LMAR $
9 = LIT,COMP 1 = SAR $
MR2 $
WHEN RDC THEN BEX $
B L = A2 $ 2R IN A2

```

```

A2 + 1 = A3 $ SAVE IN A3 FOR LATER MULTIPLY
SINCO - 1 = AMPCR $
AMPCR = MIR LMAR $
17 = LIT $
MW2 $ STORE RETURN ADDRESS
MUPY - 1 = AMPCR $
WHEN RMI THEN 0 = MAR $
MR2 $ READ N
WHEN RDC THEN A1 = SAR BEX $ DIVIDE N BY 2**K
B R = B JUMP $ MULTIPLY BY (2R+1)
SINCO. B = A3 LMAR $
0 = LIT, 2 = SAR $
MR2 $
255 = LIT $
GRETR - 1 = AMPCR $
WHEN RDC THEN BEX $
B R = A2 B $ N/4
A3 - B = B $ IS ANGLE < N/4
IF NOT AOV THEN 0 - B = B STEP ELSE JUMP $
LIT + B = MAR1 $ INDEX SINTBL BY (N/4-ANGLE)
MR1 $ READ COS
WHEN RDC THEN BEX $
B = MIR $ COS TO MIR
A3 + LIT = MAR1 $ INDEX SINTBL BY ANGLE
MR1 $ READ SINE
WHEN RDC THEN LMAR BEX $
6 = LIT $
MW2 $ STORE COSINE
WHEN RMI THEN BMAR + 1 = MAR2 $
B = MIR $
MW2 $ STORE SINE
SPCNG - 1 = AMPCR $
WHEN RMI THEN JUMP $
GRETR. LIT + B = MAR1 $ INDEX SINTBL BY (ANGLE-N/4)
MR1 0 - B = A3 $ N/4-ANGLE IN A3 READ COS
WHEN RDC THEN BEX $
0 - B = MIR LMAR $ COS IS NEGATIVE
6 = LIT $
MW2 A2 = B $ STORE COS; (B)=N/4
A3 + B = B $ N/2-ANGLE
WHEN RMI THEN LIT + B = MAR1 $
255 = LIT $ INDEX SINTBL TO GET SINE
MR1 $ READ SINE
WHEN RDC THEN LMAR BEX $
7 = LIT $
B = MIR $
MW2 $ STORE SINE
WHEN RMI THEN STEP $
SPCNG. LMAR $
11 = LIT $
MR2 $ READ SPACING
WHEN RDC THEN BEX LMAR $

```

```

8 = LIT $
MR2 B = A3 MIR $
WHEN RDC THEN BEX $ READ K
B = SAR LMAR $
9 = LIT $
MR2 A3 R = A3 $ SPACING/2**K
WHEN RDC THEN BEX $ READ R
B = A1 $
A1 + B + 1 = A1 $ 2R+1
SUBNXT - 1 = AMPCR $
AMPCR = MIR BMI LMAR $
17 = LIT $
MW2 $
MUPY - 1 = AMPCR $
WHEN RMI THEN B = MIR $ SAVE SPACING
A1 = B JUMP $ (2R+1)SPACING/2**K
SUBNXT. B L = A3 BMI LMAR $ RECALL SPACING
COMP 1 = SAR,3 = LIT $
MR2 A3 - B = A2 $ OFFSET = (A3)-SPACING
WHEN RDC THEN BEX LMAR $
12 = LIT $
A2 + B = A2 MIR $
MW2 $ OFFSET ADDRESS TO P2 STORE
WHEN RMI THEN STEP $
B4BK. LMAR $
11 = LIT $
MR2 $ READ SPACING
WHEN RDC THEN BMAR + 1 = MAR2 BEX $
MR2 B = MIR $ SAVE SPACING IN MIR
WHEN RDC THEN BEX $ READ OLD P2R ADDRESS
B = A2 BMI $ OLD P2R TO A2, SPACING TO B
A2 + B = A1 $
A1 + B = A2 A1 MIR $ NEW P2 ADDRESS IN A2&A1
MW2 $ STORE P2R ADDRESS
WHEN RMI THEN B = MIR LMAR $
0 = LIT,COMP 1 = SAR $
MR2 $ CHECK IF NEW P2< OR=2N-1
WHEN RDC THEN BEX $ BY USING
ADVSPG - 1 = AMPCR $ P2-(2N-1)-1=P2-2N
B L = A3 LMAR $
3 = LIT $
MR2 $ READ RA
WHEN RDC THEN BEX $
A3 + B = B $ RA+2N
A2 - B = $ P2:RA+2N
IF NOT AOV THEN STEP ELSE JUMP $ JUMP IF P2>
BFLY4. A1 = MAR1 $
MR1 $READ P2R
WHEN RDC THEN LMAR BEX $
7 = LIT $
MR2 B = A3 $READ SINE
WHEN RDC THEN BEX $

```

```

MRTN6 - 1 = AMPCR $
AMPCR = MIR LMAR $
17 = LIT $
MW2 $
MUPY - 1 = AMPCR $
WHEN RMI THEN B = MIR CALL $
MRTN6. B = MIR BMI $ SINE TO B
B = A2 BMI $ HOLD SINE IN A2
MRTN7 - 1 = AMPCR $
AMPCR = MIR LMAR $
17 = LIT $
MW2 $ STORE RETURN ADDRESS
WHEN RMI THEN A2 = MIR $
B = MIR BMI LMAR $ P2RSINE TO MIR; SINE TO B
6 = LIT $
MR2 B = A3 $ SINE TO A3; READ COSINE
WHEN RDC THEN BEX $
A3 + B = A2 $SIN + COS IN A2
B = A3 $COS TO A3
A1 = MAR1 $
MR1 $READ P2R
WHEN RDC THEN BMAR + 1 = MAR1 BEX $
MR1 B = A1 $P2R TO A1
WHEN RDC THEN LMAR BEX $READ P2I
15 = LIT $
MW2 A1 + B = A1 $STORE P2RSINE
MUPY - 1 = AMPCR $
WHEN RMI THEN A2 = MIR CALL $
MRTN7. B = MIR BMI $
B = A2 BMI $
MRTN8 - 1 = AMPCR $
AMPCR = MIR LMAR $
17 = LIT $
MW2 $
WHEN RMI THEN A1 = A3 $
MUPY - 1 = AMPCR $
B = A1 $
A2 = B CALL $
MRTN8. LMAR $
15 = LIT, 14 = SAR $
MR2 B = A2 $RECALL P2RSINE
WHEN RDC THEN BEX $
A1 - B R = A3 $ PRELIMINARY IMAGINARY RESULT
A1 + B = B IF MST THEN SET LC1 $
A2 - B R = A2 B $ PRELIMINARY REAL RESULT
LIT L = A1 IF MST THEN SET LC2 $
63 = LIT, COMP 8 = SAR $
A1 OR LIT L = A1 $
255 = LIT, COMP 18 = SAR $
LMAR $ ***TO ALLOW A1 TO CHANGE***
12 = LIT $
IF LC2 THEN A1 OR B = A2 $ REAL

```

```

IF LC1 THEN A3 = B STEP ELSE SKIP $
A1 OR B = A3 $ IMAGINARY
MR2 $ READ P2R ADDRESS
WHEN RDC THEN LMAR BEX $
11 = LIT $
MR2 B = A1 $
WHEN RDC THEN BEX $
A1 - B = MAR1 $
MR1 $ READ P1R VALUE
WHEN RDC THEN BEX LMAR $
13 = LIT $
B = MIR $
MW2 $ SAVE OLD P1R
WHEN RMI THEN A2 + B = MIR LMAR $
11 = LIT $
MR2 $ READ SPACING
WHEN RDC THEN BEX $
A1 - B = MAR1 $
MW1 $ STORE NEW P1R
WHEN RMI THEN BMAR + 1 = MAR1 $
MR1 $ READ P1I VALUE
WHEN RDC THEN BEX $
A3 + B = MIR $
MW1 $ STORE NEW P1I
WHEN RMI THEN A1 + B001 = MAR1 $ P2I ADDRESS
A3 = MIR $ SAVE P2W**N IMAGINARY PART
B = A3 BMI $ P1I VALUE TO A3
A3 - B = MIR $
MW1 $ STORE NEW P2I
WHEN RMI THEN LMAR $
13 = LIT $
MR2 A2 = MIR $ SAVE P2W**N REAL PART
WHEN RDC THEN BEX $
B = A2 BMI $ P1R VALUE TO A2; REAL PART TO B
A2 - B = MIR $
A1 = MAR1 $
MW1 $ STORE NEW P2R
B4BK - 1 = AMPCR $
WHEN RMI THEN JUMP $
ADVSPG. BMI $
B L = MIR $
COMP 1 = SAR $
GNRL2 - 1 = AMPCR $
JUMP $
NEW. B = A2 LMAR $
9 = LIT, COMP 1 = SAR $
A2 - 1 = A2 MIR $
MW2 $
NEWANG - 1 = AMPCR $
WHEN RMI THEN JUMP $
FINISH. STEP $
END $
READY.

```

Appendix B

REAL-VALUED INPUT SEQUENCE

The direct transform of $2N$ real input samples may be found efficiently by storing consecutive samples in consecutive locations in the Data Area, and performing an N -dimensional (complex) transform. The input sequence, $X(q)$, $q = 0, 1, \dots, 2N-1$, can be separated into two N -dimensional sequences; $X_1(j)$, even-numbered input points, and $X_2(j)$, the odd-numbered input points. Since $X(q)$, $X_1(j)$, and $X_2(j)$ are strictly real, their transforms, denoted by $C(q)$, $A_1(n)$, and $A_2(n)$, respectively, possess the property of conjugate symmetry (*denotes the complex conjugate):

$$C(N+m) = C^*(N-m) \quad m = 0, 1, \dots, N-1 \quad (\text{B1})$$

$$A_1(N/2+n) = A_1^*(N/2-n) \quad (\text{B2})$$

$$A_2(N/2+n) = A_2^*(N/2-n) \quad n = 0, 1, \dots, N/2-1. \quad (\text{B3})$$

The relationship between C , A_1 , and A_2 may be derived by stretching X_1 and X_2 to dimension $2N$ by interspersing zeros between the original input samples. It is easy to show that this stretching introduces a factor of $1/2$ in the transforms. The input sequence $X(q)$ is then the sum of the stretched X_1 sequence and the stretched X_2 sequence delayed by one sample. Since a delay of K samples in the start of the input sequence causes the transform to be multiplied by $\exp(-2\pi ink/2N)$, the transform $C(m)$ is equal to

$$C(m) = \frac{1}{2} [A_1(m) + A_2(m) \exp(-2\pi im/2N)]. \quad (\text{B4})$$

For values of m greater than $N-1$, the values of $C(m)$ are obtained by noting that A_1 and A_2 are periodic, of period N :

$$C(N+m) = \frac{1}{2} [A_1(m) - A_2(m) \exp(-2\pi im/2N)] \quad (\text{B5})$$

$$m = 0, 1, \dots, N-1.$$

The transforms A_1 and A_2 are found from A , the transform of the input sequence, considered as N complex points, by means of the following relationships:

$$A(n) = A_1(n) + iA_2(n) \quad (\text{B6})$$

$$\begin{aligned}
 A^*(N-n) &= A_1^*(N-n) - iA_2^*(N-n) \\
 &= A_1(n) - iA_2(n)
 \end{aligned} \tag{B7}$$

$$A_1(n) = \frac{1}{2} [A(n) + A^*(N-n)] \tag{B8}$$

$$A_2(n) = \frac{1}{2i} [A(n) - A^*(N-n)] \quad n = 0, 1, \dots, N/2-1. \tag{B9}$$

The procedure for finding the transform of the input sequence, considered as $2N$ real values, is thus to use the FFT program to calculate A , then apply Eqs. (B8) and (B9) and finally (B4) and (B5). In assigning storage locations, use is made of the fact that $A_1(0)$, $A_1(N/2)$, $A_2(0)$, $A_2(N/2)$, $C(0)$, and $C(N)$ have imaginary components which are identically zero, so they may be paired. All other transform components are, in general, complex.

Appendix C

INVERSE TRANSFORM

The inverse transform of a set of Fourier coefficients $B(n)$ which possess conjugate symmetry, can be found by combining the coefficients as shown below, applying the direct transform, and combining the results of the direct transform. The conjugate symmetry is expressed as (*denotes the complex conjugate):

$$B(N-n) = B^*(n) \quad n = 0, 1, \dots, N/2. \quad (C1)$$

The inverse transform is

$$g(k) = \sum_{n=0}^{N-1} B(n) \exp(2\pi ink/N) \quad k = 0, 1, \dots, N-1. \quad (C2)$$

Making use of Eq. (C1) results in

$$g(k) = B(0) + (-1)^k B\left(\frac{N}{2}\right) + 2 \sum_{n=1}^{N/2-1} \text{Re}\{B(n)\exp(2\pi ink/N)\}. \quad (C3)$$

Combine the components to form the N real numbers,

$$f(k) = \text{Re}\{B(k)\} + \text{Im}\{B(k)\} \quad (C4)$$

$$f(N-k) = \text{Re}\{B(k)\} - \text{Im}\{B(k)\}. \quad (C5)$$

Or,

$$B(k) = \frac{f(k) + f(N-k)}{2} + i \frac{f(k) - f(N-k)}{2} \quad k = 0, 1, \dots, N/2. \quad (C6)$$

Substitute Eq. (C6) into (C3) to obtain

$$g(k) = \sum_{k=0}^{N-1} f(k) \cos(2\pi nk/N) - f(k) \sin(2\pi nk/N). \quad (C7)$$

The terms in this summation may be obtained by a direct transform of the sequence $f(k)$. Denoting the result of this transform by $A(n)$, the desired inverse transform is

$$g(n) = \text{Re}\{A(n)\} + \text{Im}\{A(n)\} \quad n = 0, 1, \dots, N/2 \quad (C8)$$

$$g(N-n) = \text{Re}\{A(n)\} - \text{Im}\{A(n)\} \quad (C9)$$

due to the symmetry properties of $A(n)$.

The procedure for finding the inverse transform of the original coefficients $B(n)$ is thus to combine the components of $B(n)$ according to Eqs. (C4) and (C5), perform a direct transform on the results to yield $A(n)$, and then to combine the results according to Eqs. (C8) and (C9).

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY (Corporate author) Naval Research Laboratory Washington, D.C. 20390		2a. REPORT SECURITY CLASSIFICATION Unclassified	
		2b. GROUP	
3. REPORT TITLE A FAST FOURIER TRANSFORM MICROCODE FOR THE BURROUGHS D-MACHINE			
4. DESCRIPTIVE NOTES (Type of report and inclusive dates) An interim report on a continuing NRL Problem.			
5. AUTHOR(S) (First name, middle initial, last name) H. H. Smith			
6. REPORT DATE December 8, 1972		7a. TOTAL NO. OF PAGES 68	7b. NO. OF REFS 12
8a. CONTRACT OR GRANT NO. NRL Problem B02-10		9a. ORIGINATOR'S REPORT NUMBER(S) NRL Report 7494	
b. PROJECT NO. XF-21-241-015-K152		9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report)	
c.			
d.			
10. DISTRIBUTION STATEMENT Approved for public release; distribution unlimited.			
11. SUPPLEMENTARY NOTES		12. SPONSORING MILITARY ACTIVITY Department of the Navy (Naval Electronics Systems Command) Washington, D.C. 20360	
13. ABSTRACT A fast Fourier transform program has been developed and run on a simulation of the Burroughs D-Machine processor. This program was developed to gain insight into the use of microprogrammed structures for signal processing applications and to serve as a reference "benchmark" during the development, evaluation, and application of programmable signal processors at the Naval Research Laboratory. The D-Machine simulator and a companion translator program have been written in Fortran to run on the Control Data Corporation's Kronos time-sharing system. This action was taken because of a paucity of software support for the D-Machine itself and because the simulator provides debugging features not found in hardware. The time-sharing terminal was also more accessible. A novel feature of the FFT program is its ability to terminate calculations before $\log_2 N$ stages have been completed (N is the number of complex input data points), thereby providing multiple estimates of less than N spectral lines, rather than one estimate of each of N lines, as is the case when $\log_2 N$ stages are carried to completion. Several of the test spectra which have been calculated are described in Section 3 of the report. For reasons of scarcity of registers and lack of concurrency, the D-Machine is deemed poorly suited to for signal processing applications.			

14. KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
Fast Fourier transform Signal processing Microcode						