The LX-1 Microprocessor and Its Application to Real-Time Signal Processing

GARY D. HORNBUCKLE, MEMBER, IEEE, AND
ENRICO I. ANCONA, MEMBER, IEEE

Abstract—LX-1 is an integrated circuit prototype of a microprocessor which is being used as a design vehicle to study the problems associated with the design and implementation of a similar computer constructed with large-scale integrated circuits. The organizational simplicity of LX-1 is emphasized and the supporting microprogramming and simulation facilities are discussed and examples are given.

The major portion of the control of the microprocessor is implemented in a 256-word by 64-bit control memory, and the remaining logic is partitioned into a very few unique functional logic circuits. There are 16 general-purpose 16-bit registers and a 256- by 16-bit scratch memory. The cycle time is 70 ns, and logic is provided for addition, bit logical, array shift, and array multiply operations.

To demonstrate the feasibility of using small, general-purpose, microprogrammed computers for real-time digital signal processing, the application of LX-1 as a digital vocoder was investigated. The advantages of using a microprogrammed processor rather than special-purpose hardware for this application are the lower cost of such a general-purpose computer and the flexibility provided by the ability to change the microprogram.

An implementation of the spectrum analyzer portion of the vocoder and its operation on the LX-1 simulator is described. The results show that real-time operation of the spectrum analyzer on the LX-1 prototype could be expected with minimal additional hardware, e.g., a 200-sample input buffer. Moreover, the proposed LSI version of LX-1 could be operated in half duplex mode for both analysis and synthesis, or to process several input channels.

Index Terms—Computer design, LSI computers, microprogramming, real-time digital signal processing, spectrum analysis, vocoders.

INTRODUCTION

THERE is currently much interest in machines designed to take advantage of large-scale integrated circuit (LSI) technology. This paper discusses a microprocessor which was designed and built at the M.I.T. Lincoln Laboratory. The organization and regularity of layout were central considerations because of the constraints posed by large-scale integration. The major portion of the control of the microprocessor is implemented in a control memory and the remaining logic is partitioned into a very few unique functional circuits. The technique of control implemented in a control memory (often read-only memory) is well known and is called microprogramming, and such processors are referred to as microprocessors.

The paper describes a prototype processor, called LX-1, which has been built primarily to test the applicability of microprogramming, as well as the particular design, to a variety of problems. However, the prototype was constructed from commercially available emitter-coupled logic circuits which are similar to those anticipated for an LSI version. The logic was partitioned as nearly as possible in the same manner that an LSI version might be. This paper emphasizes the organizational simplicity of LX-1 and discusses the supporting microprogramming and simulation facilities that have been programmed on the TX-2 computer at the M.I.T. Lincoln Laboratory. Specifically, it discusses in detail the application of LX-1 to the problem of real-time, digital encoding of speech.

LX-1 is a processor intended for use in a multiprocessor environment with a hierarchical, general-purpose memory system. It has three levels of memory—register, scratch, and control—but has no core or bulk memory as these are treated as I/O devices. LX-1 can be used in a variety of ways. For example, it can be used as a digital controller, such as a drum or tape controller, or as a special-purpose processor to rapidly compute such functions as Fourier transforms, floating point arithmetic, or trigonometric functions. The more common use for microprocessors has been in the role of a Central Processor to interpret (or emulate) so-called machine languages, such as is done in several models of the IBM System/360, and microprocessors have been proposed which could interpret higher level languages such as LISP [1] and Euler [2]. A wide variety of applications is made possible by changing the contents of the control memory, i.e., the microprogram. In a general sense, LX-1 is like other processors if one considers the (read-only)[1] program to reside in the control memory. With this view, LX-1 is a four-address machine with limited (read-only) program memory. The four addresses contained in a typical microinstruction are the location of two operands, the location of the result, and the location of the next instruction.

HARDWARE DESIGN

Partitioning

In the design of LX-1, the emphasis was placed upon simplifying the backpanel wiring, minimizing the number of different kinds of circuits, and simplifying testing, since these are major considerations in large-scale integration. The processor was not designed to simplify microprogram-

Manuscript received September 18, 1969; revised January 25, 1970. This work was sponsored by the U. S. Advanced Research Projects Agency.

G. D. Hornbuckle was with the Massachusetts Institute of Technology, Lincoln Laboratory, Lexington, Mass. He is now with Applicon Inc., Burlington, Mass., 01803.
E. I. Ancona is with the Massachusetts Institute of Technology, Lincoln Laboratory, Lexington, Mass. 02173.

[1] Read-only is placed in parenthesis here to emphasize that microprogram memory need not be read-only. The control memory of LX-1 is read-only with respect to the microprogram, but is externally writeable.

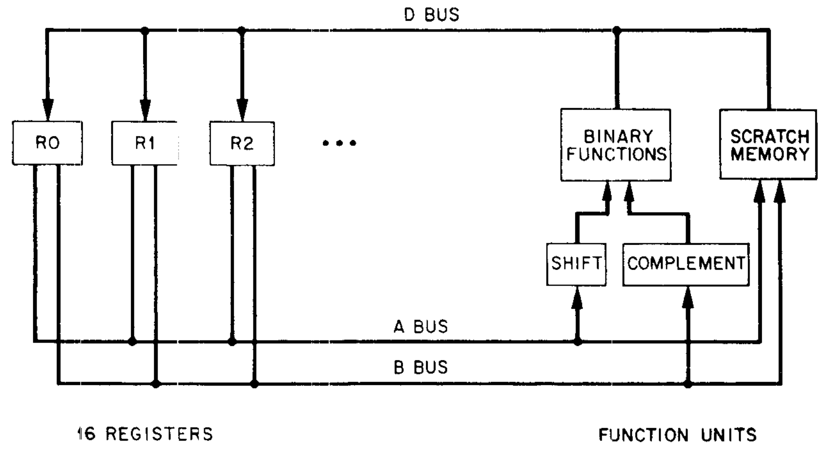

Fig. 1. Simplified block diagram for the LX-1 microprocessor.

ing, although considerable attention was given to making possible to concisely write and conveniently debug microprograms.

The three major components of LX-1 are the control memory, a set of 16 16-bit registers, and several function units which perform operations on the data in the registers. The function units realized in LX-1 include a logic and adder unit, a scratch memory, and a multiplier unit.

To simplify the backpanel wiring a three-bus structure was used. Fig. 1 gives a simplified block diagram, Fig. 2 is more detailed. Each of the 16 registers is connected to the A- and B-bus which are the data paths from the registers to the function units, and each register is connected to the D-bus which is the data path from the function units to the registers. These 48 bus wires consume one layer of parallel wiring on the backpanel. The remaining wires, which consist of selection lines, input/output lines, and other control wires, cause all the backpanel wiring crossover problems. An attempt was made to minimize the number of these wires by suitable partitioning of the logic and by simplifying the basic design.

The number of different kinds of circuits was minimized by partitioning all flip-flop memory into identical registers, one-half register per card or LSI chip. The logic and arithmetic function units are partitioned into three types, each of which is iteratively repeated for the function. One type implements four bits of a function unit which can generate the arithmetic sum or Boolean sum or product of two operands. A second type implements one-eighth of an array shift function unit. A third type generates the arithmetic product. The remaining two frequently used logic circuits are a select circuit which selects one of eight inputs or decodes three inputs into eight outputs, and a select circuit which is used to select between pairs of inputs such as for the B-bus complement function.

The remaining miscellaneous circuits, such as bus drivers, either would be fundamentally different for LSI or would be added to the frequently used circuits to avoid new circuit types. The circuits directly associated with the control memory would also be different for LSI. The scratch memory of LX-1 is constructed from one printed circuit card type which contains 16 16-bit integrated circuit memory elements. A similar LSI scratch memory circuit would contain 256 memory bits with decoding. Table I gives the number of each circuit type used and Fig. 1 shows where they are used.

Register slice, rather than bit-slice, partitioning was chosen for reasons of expandability. It was felt that it would be desirable to be able to increase the number of registers and function units with only minor revisions in the layout. With bit-slice the number of bits per register could be expanded easily, but this was considered to be of secondary importance, although with LX-1, the number of bits could be conveniently increased in four-bit increments. Another important reason for using register slice partitioning was to isolate all sequential circuits, that is, all flip-flops, to one circuit type. All other circuits, other than the scratch memory and control memory buffer, are combinational, and well developed techniques for diagnosis of combinational circuits are applicable [3].

Operation

During a typical microinstruction, the contents of one of the 16 registers is placed on the A-bus, the contents of the same or another register is placed on the B-bus, an operation is performed in a function unit, and the result which appears on the D-bus is stored in the same or a third register. This entire sequence occurs within a single 70-ns clock cycle; the clock is only used to strobe the function results from the D-bus into a register and to strobe the results of control memory read cycles into the control memory buffer. The control memory, which is constructed from the same circuit as the scratch memory, acts as a combinational circuit during reading, and the read cycle is generally overlapped with the register operations.

The data on the A-bus can be shifted or rotated from zero to 15-bit positions, right or left, and the data on the B-bus can be complemented, prior to the arithmetic and logic function operations. The latter include binary addition and bit-by-bit AND, OR, and EXCLUSIVE-OR. Subtraction of the B-bus data is performed by complement addition. Both one's and two's complement arithmetic are made possible

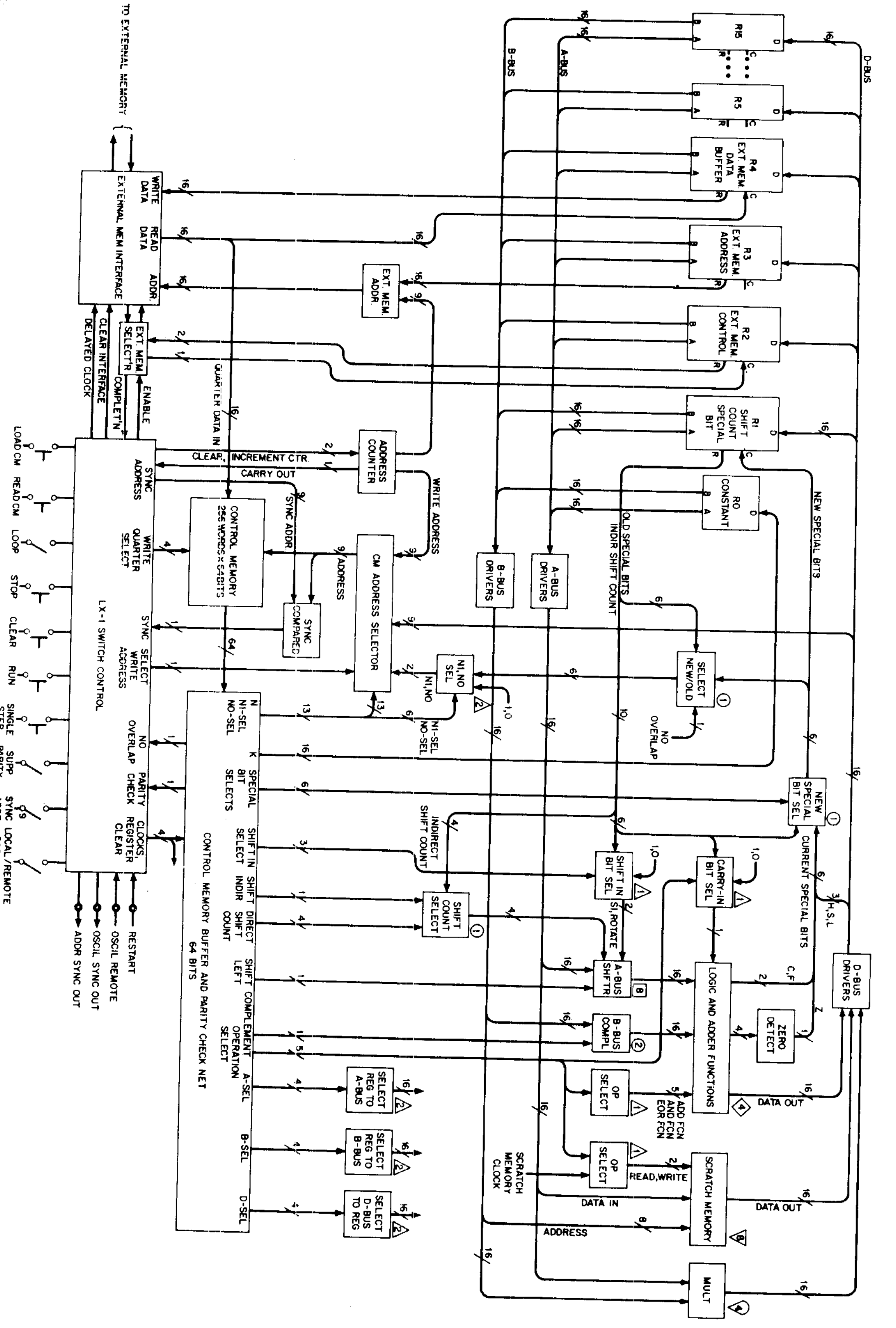

Fig. 2. Block diagram for the LX-1 microprocessor.

TABLE I

LX-1 CIRCUIT TYPES

Circuit Type	Fig. 1	Circuits Used	Gates per Circuit	Gates Used	Signal Pins per Circuit
Register		32	72	2304	47
Adder/Logic	◇	4	71	284	21
Shift	□	8	75	600	24
Select 1	△	12	13	156	26
Select 2	○	4	49	196	33
Multiply	▽	4	103*	412	57
Scratch Memory	△	8	37†	296	20
		72		4248	

* Includes 64 full adder circuits.
† Excludes 256 memory bits.

by allowing the carry-in to the lowest order adder position to be a logical one, logical zero, or one of six special bits. All types of shift operations, such as arithmetic and logical, are possible because the bits shifted into the vacated bit positions can be selected from logical one or zero, one of five special bits, or the bits shifted out (for rotate or cycle shifts). Also, the shift count can be stated explicitly in the microinstruction or can come indirectly from register one.

The second function unit of LX-1 is a 256-word scratch memory. In a single microinstruction, a word can be read from or written into the scratch memory. The data to be written come from the A-bus, and the address comes from the low-order eight bits of the B-bus. The write-clock is derived from the basic machine clock.

A third function unit, an array multiplier, generates the high-order or low-order 16 bits of the product. Two instructions are required to store the results for full precision. The multiplier is attached to the A-, B-, and D-buses in a manner similar to the other function units.

All overflow, carry indicators, and condition bits used for carry-in, shift-in, and conditional branches are stored in six bits of register one. These special bits are as follows:

H—high-order bit of D-bus
C—carry-out of add operation
Z—zero detect from adder/logic unit
L—low-order bit of D-bus
S—second-from-low-order bit of D-bus
F—overflow from add operation or multiply operation.

By selecting any two of the special bits, a logical one, or a logical zero, one can cause absolute, two-way conditional, or four-way conditional branches. All next-address selection bits are included in every microinstruction and a program counter is not used. The address counter shown in Fig. 2 is for external loading of the control memory. In each microinstruction, one can elect to save any subset of the special bits just generated or to save the old values. The conditional branch can use the old values or the newly generated values of the special bits. In the latter case an extended cycle is caused because overlap of control memory reading and function execution is not possible. Since an extended cycle is also used for addition to allow for worst-case carry propagation, long extended cycles can occur. A special case of the next-instruction bits allows the next control-memory address to come from the D-bus. This extended cycle feature allows microprogram subroutines and branch tables.

Sixteen bits of each microinstruction are reserved for an immediate constant. The constant will appear on the A-bus or B-bus if register zero is selected. A standard register circuit is used for register zero to allow uniform gating onto the A- and B-buses, although it is wired to act as a nonstorage data transfer circuit.

Each register, in addition to its A-, B-, and D-bus connection, has 16 output and 16 input lines, one for each bit. These connections are used internally in the processor in the case of register one, but the remaining 448 leads can be attached to external equipment. For an LSI version some rather small connectors will be required. Although a multiplexed I/O bus would avoid these connections, the timing problems and complexities of such logic would be greater. Even if one were to construct an I/O bus, further logic would be needed to tap onto the bus for each external device. If such logic were to be constructed with LSI, one would have a problem similar to connecting 448 leads to the microprocessor. Furthermore, for LX-1 most of the timing problems associated with input/output equipment can be solved in the microprogram. It is unlikely that all 14 registers will be attached to I/O equipment, although at least two are required if an external memory is connected to the processor. The interconnection of two processors can be made by simply attaching the input leads of a register in one processor to the output leads of another, and vice versa. Communication control can be handled with the microprograms.

FIRMWARE DESIGN

A programming package has been created on the TX-2 computer that allows a user to write and run microprograms written in a symbolic assembly language. Statements in this language, called ML, correspond to the control instructions for the LX-1 processor. ML programs are free-format text files. The language allows symbolic labels, register names, literal and address constants, and is highly readable. The ML programs are compiled by the ML assembler, which is implemented in VITAL [4], into files of control memory bit patterns. Side-by-side formatted listings are produced. The control memory files created drive the LX-1 simulator. Eventually, they will be read into the LX-1 control memory or be used to determine read-only memory bit patterns in subsequent LSI versions. The syntax and description of the language are included in the Appendix.

The LX-1 microprogram checkout and debugging package (LXSIM) provides a cycle-by-cycle simulation of the LX-1 prototype processor. The simulator was designed to allow maximum user flexibility. Snapshots of the current machine status are given via an on-line display. Breakpoints can be put on microinstructions or on memory locations. Individual instructions can be altered or the whole program can be edited and recompiled. Memory locations can be examined and filled. Conditional execution or single stepping of the simulator is possible. New control memory or main memory files can be loaded at will. Different versions of the simulator, corresponding to various machine configurations, can be selected.

Since its purpose is to assist in debugging microprograms, the simulator is controlled interactively in a very flexible way. This flexibility is achieved because the simulator is designed to accept a large number of low-level text-encoded commands. If desired, the user can enter lists of these commands directly, via the outline keyboard, but to do so is quite tedious. These basic commands, though quite flexible, are relatively primitive. For example, 72 characters must be typed to clear the 16 general registers. To combat inevitable user fatigue, a run-time command package has been written to allow the directed expansion of user-defined succinct commands into primitive commands.

User commands are defined by procedures written in the DOMEX (Display Oriented Macro EXpander) language, which has been based on the TRAC language [5]. Like TRAC, DOMEX is a language for text manipulation. Strings of characters may be named, parameter markers inserted, and strings called by name with argument lists. Character strings can be treated arbitrarily as executable procedures, names, or as text. Recursive function calls are also possible.

Unique to DOMEX is its ability to use the on-line display and Sylvania tablet [6]. Pseudo light buttons can be defined and given procedural value. When a light button is pointed at, the procedure is executed, at times producing a new selection of light buttons. The procedure executed may also send strings of commands to the simulator. It can interrogate the status of the 16 registers in the simulated machine and use this information to control the generation of the command sequence sent. The procedure may also choose to demand characters from the tablet. The character recognizer [7] will be called and its output made available to the procedure.

Part of the flexibility of DOMEX commands is their run-time definition. When the simulator package is first called, the only defined DOMEX command is one that will read and execute text files. An initialization text file might contain a DOMEX procedure which, when executed, defines other commands. Different users may have different initialization files. The user is free to drop or edit old definitions or to create new ones. DOMEX procedures can define new ones or be self-modifying. New initialization files can be written out at any time.

Typical simulator control procedures that can be defined might include:

1) single step until a selected register contains a given value,
2) load one register from another,
3) remember the machine status so it can be restored later—even at a different session,
4) print read-only memory in symbolic format,
5) trace a register, typing the value and the microprogram instruction counter every time it is changed,
6) edit, recompile, and reload the control memory file,
7) write out a text file which, when read in, redefines all strings currently in use,
8) trace an instruction, printing out selected register values every time it is executed.

The selection and form are, of course, up to the user. The command package can be tailored as desired.

Application to Real-Time Signal Processing

The usefulness of the LX-1 design to real-time digital signal processing was investigated to demonstrate LX-1's capabilities and flexibility. In particular, its application to the spectrum analyzer and coding portions of a digital channel vocoder was investigated. The vocoder algorithm was designed by Anderson [9] and Bially [10]. This particular application was chosen because of the current interest in digital signal processing and also because the computational load and real-time constraints imposed by the spectrum analyzer algorithm are well-matched to the microprocessor's capabilities.

The advantages of using microprogrammed processors rather than special-purpose hardware are twofold: first, general-purpose LSI processors like LX-1 are potentially less expensive than special-purpose devices. Second, the flexibility provided by the microprogrammed control enables designers to alter their algorithms by changing only the control memory contents.

The LX-1 Simulator

The vocoder spectrum analyzer and coding were simulated using the LXSIM package on the TX-2 computer. Except for input/output, and input buffer maintenance, the vocoder simulator microcode is identical to that of the real vocoder microcode. The above operations, however, do not add significant overhead to the computations, and thus the values obtained from simulation are reasonable.

Channel Vocoder

The overall vocoder analyzer and synthesizer structure is shown in Fig. 3. The input to the analyzer consists of 10-bit samples of speech entering at a 10-kHz rate, or every 100 μs. Speech bandwidth compression in the vocoder is achieved by coding two parameters of the speech: the first parameter is the pitch period, which is a measure of the excitation to the vocal tract. The second parameter is the spectrum, which is a measure of the characteristics of the vocal tract. The spectrum analyzer calculates the magnitude of the response of the vocal tract at 32 frequencies. These two parameters are coded into 2400 bits/second and are transmitted to the synthesizer.

There, the impulse response of the vocal tract, or the inverse discrete Fourier transform of the spectrum, is con-

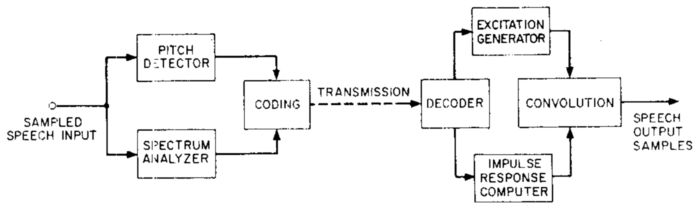

Fig. 3. Digital vocoder system.

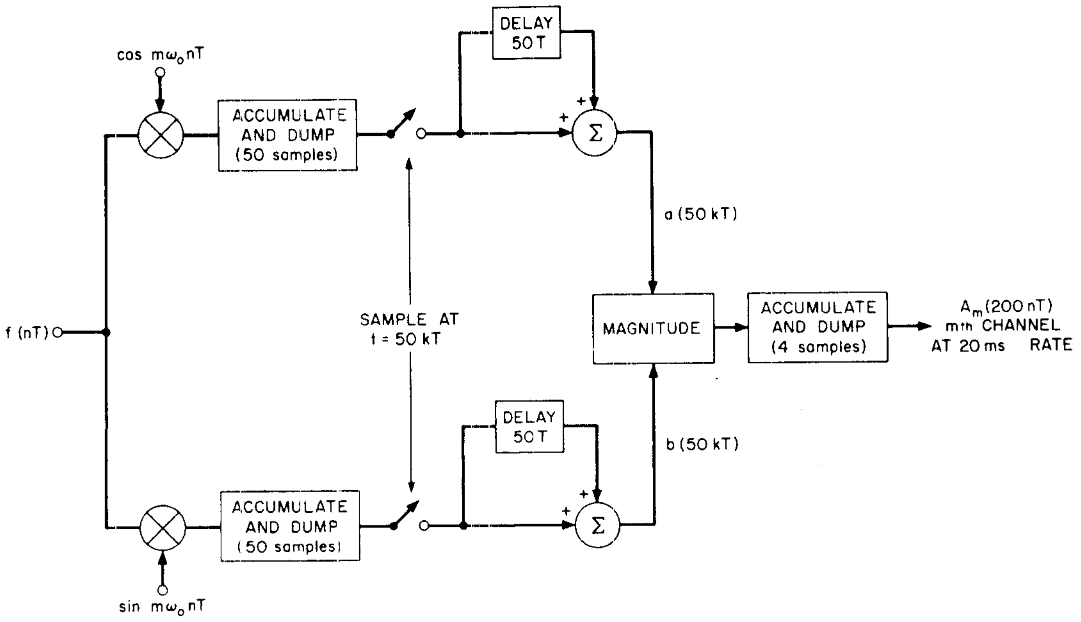

Fig. 4. Simplified *m*th channel of vocoder analyzer.

volved with an appropriate excitation to produce speech. Only the spectrum analyzer and coding portions of the vocoder are described here in detail and the results of simulation on LXSIM will be discussed. The reader is referred to Gold [11] for a description of pitch detection and to Anderson [9] and Bially [10] for a description of the synthesizer.

Vocoder Spectrum Analyzer

The spectrum analyzer consists of 32 bandpass filters centered at 100, 200, · · · , 3200 Hz. Each bandpass filter is of the form shown in Fig. 4. The 10-bit samples of speech are input into each channel at a 10-kHz rate and are modulated by a sine and cosine function at the center frequency of the bandpass filter in question. Fifty successive samples are accumulated and saved, and are then added to the sum of the previous 50 samples. The magnitude of the function is then taken. Thus, every 5 ms the magnitude of the discrete Fourier transform at the center frequency of the filter for the last 10 ms of input has been found. This corresponds to

$$\sum_{n=0}^{99} f(nT)e^{jm\omega_0 nT}$$

for channel m (center frequency $m\omega_0$), where $\omega_0 = 2\pi \times 100$ Hz.

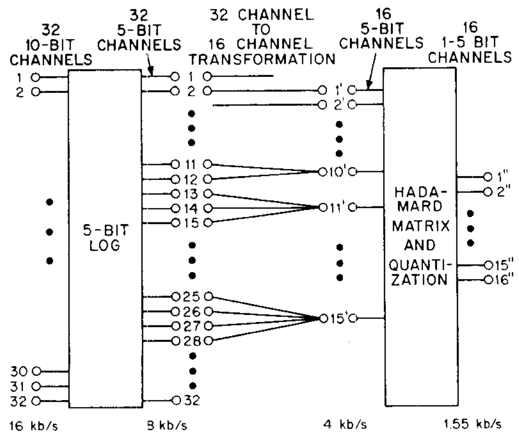

Fig. 5. 2400-bit coding of 32 vocoder channels.

The second portion of the spectrum analyzer's function is to take the average of the last four 5-ms outputs and to code the output of the 32 channels to enable transmission at 2400 bits/second, as shown in Fig. 5. First, the 5-bit logarithm of each of the 32 10-bit channels is taken. Then the 32 channels are converted to 16 by means of a linear combination of less significant channels, and finally the channels are multiplied by a Hadamard matrix to decorrelate the channels and quantized to 1 to 5 bits per channel. This coding yields a total of 1550 bits/second. The remaining 850 bits/second are used for pitch information.

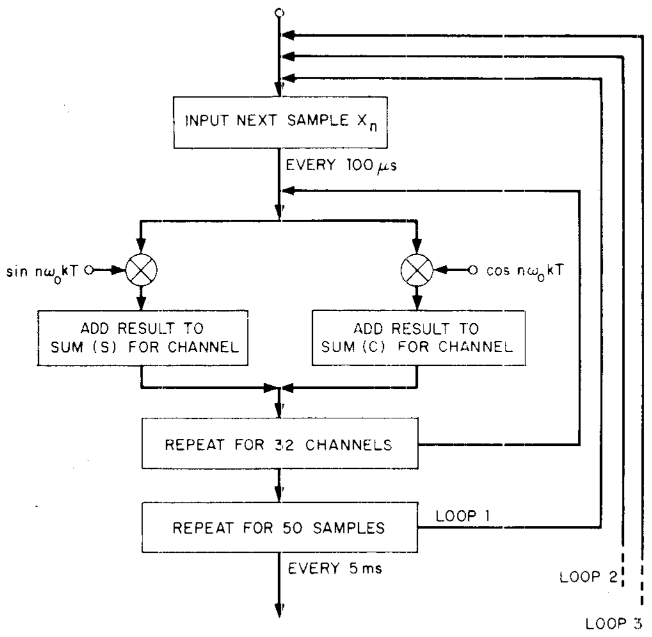

Fig. 6. Loop 1 of microprogrammed analyzer.

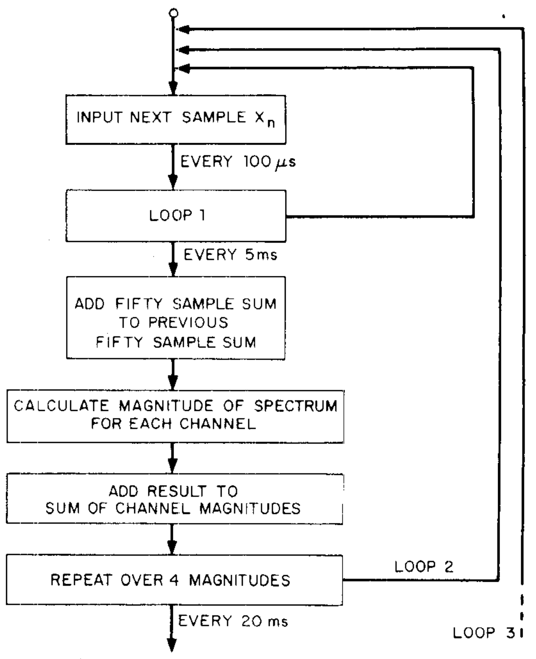
INPUT NEXT SAMPLE X_n
EVERY 100 μs
LOOP 1
EVERY 5 ms
ADD FIFTY SAMPLE SUM TO PREVIOUS FIFTY SAMPLE SUM
CALCULATE MAGNITUDE OF SPECTRUM FOR EACH CHANNEL
ADD RESULT TO SUM OF CHANNEL MAGNITUDES
REPEAT OVER 4 MAGNITUDES
LOOP 2
EVERY 20 ms
LOOP 3

Fig. 7. Loop 2 of microprogrammed analyzer.

Microprogrammed Analyzer

There are three main loops in the microcode.

Loop 1: Every 100 μs, the discrete Fourier transform of the current sample at the 32 frequencies is computed. Next, the running sum over the last 50 samples, as shown in Fig. 6, is computed as follows:

$$\sum_{n=0}^{49} x_n \sin m\omega_0 nT, \tag{1}$$

and

$$\sum_{n=0}^{49} x_n \cos m\omega_0 nT \tag{2}$$

where m is the channel number and ω_0 is the base frequency (100 Hz). The first quadrant of the lowest frequency sine wave (100 Hz) is stored in 25 words of the read-only control memory. Each value is coded into 6 bits. All other sine and cosine values may be obtained from this table by appropriate functions for the sign and table lookup. The multiplication is done by means of the array multiplier mentioned earlier. Thus, the input to Loop 2 is the 64 sums of the form shown in (1) and (2).

Loop 2: Every 5 ms, the 64 sums are added to the sums of the previous 50 samples, which were stored in the scratchpad memory, and the magnitude of the spectrum for each channel is calculated (see Fig. 7). Next, the sum of four spectrum magnitudes for each channel is calculated. Thus, every 20 ms, the input to Loop 3 consists of 32 10-bit spectrum magnitudes.

Loop 3: Every 20 ms, the logarithm of the sum of the spectra is calculated and the 32 channels are compressed to 16 by appropriate linear combinations of the logarithms. Finally, the 16 results are multiplied by a Hadamard matrix

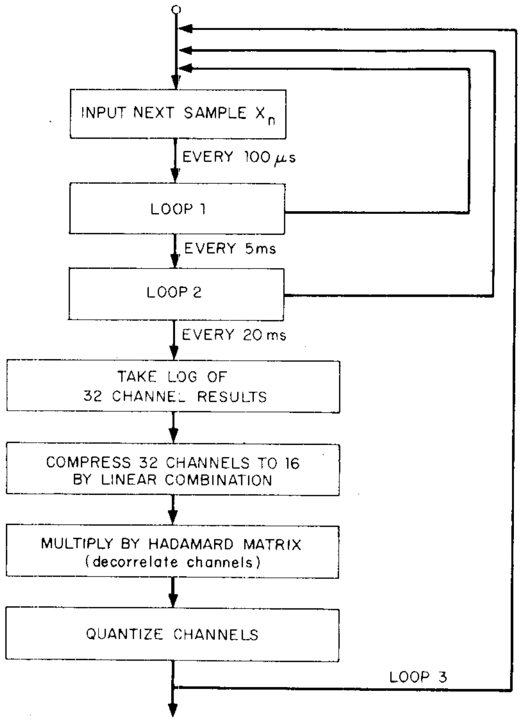
INPUT NEXT SAMPLE X_n
EVERY 100 μs
LOOP 1
EVERY 5 ms
LOOP 2
EVERY 20 ms
TAKE LOG OF 32 CHANNEL RESULTS
COMPRESS 32 CHANNELS TO 16 BY LINEAR COMBINATION
MULTIPLY BY HADAMARD MATRIX (decorrelate channels)
QUANTIZE CHANNELS
LOOP 3

Fig. 8. Loop 3 of microprogrammed analyzer.

to decorrelate the channels and the channels are appropriately quantified to yield a total of 32 bits every 20 ms (see Fig. 8). The magnitude of the Hadamard coefficients is always one and their sign is calculated by means of

$$h_{ij} = \alpha_1 \cdot \beta_4 \oplus \alpha_2 \cdot \beta_3 \oplus \alpha_3 \cdot \beta_2 \oplus \alpha_4 \cdot \beta_1$$

where $i=\alpha_1\alpha_2\alpha_3\alpha_4$ and $j=\beta_1\beta_2\beta_3\beta_4$ are the row and column indices, respectively. If $h_{ij}=1$, the sign is negative. If $h_{ij}=0$

TABLE II
RESULTS OF SIMULATION

	Length (μinstructions)	Best Time (μs)	Worst Time (μs)
op 1	50	55	80
(every sample)		(55 μs/sample)	(80 μs/sample)
op 2	30	45	50
(every 50 samples)		(1 μs/sample)	(1 μs/sample)
op 3	120	650	700
(every 200 samples)		(3.2 μs/sample)	(3.5 μs/sample)
erall	200	60	85
(per sample basis)			

e sign is positive. Note that $\cdot$ is logical AND, and $\oplus$ is gical EXCLUSIVE-OR.

esults

The results, using a 70-ns cycle time for LX-1, are shown Table II. All times marked "per sample" assume a 200-mple input buffer. This buffer is necessary because, as own in Table II, the 5-ms and 20-ms calculations take considerable amount of time. The 100-μs calculation is st enough, however, to enable the analyzer to catch up ith the input every 200 samples at most. Depending on e values of the speech input samples, the computation mes vary within the limits described in Table II.

The above implementation shows that the analysis of put speech can be done in real-time. Two factors, however, have not been taken into account. One is the maintenance of the input buffer and the other is the section of the alyzer dealing with pitch detection which operates in arallel with the spectrum analyzer.

It can also be seen that Loop 2 and Loop 3, even though ey are long, add at most a 10 percent overhead to the verall calculation if a 200-sample input buffer is assumed.

CONCLUSION

In summary, the simplicity of organization of LX-1 lends self to large-scale integration, yet the instructions are omplex enough to be interesting and useful. The price for mplicity appears to be in the long words required in the ontrol memory. All attempts to date to decrease the number of bits per microinstruction have resulted in rather d hoc and cumbersome additions to the present logic. For xample, the 16-bit constant field could be eliminated by eferencing a second read-only memory indirectly. This ould eliminate approximately ten of the 64 bits. With a ardware program counter one could also eliminate about ght bits from the next-instruction address. However, program counters must be incremented, and if the adder is sed in order to avoid a new chip type, the cycle time might crease. One could also allow fewer operations per instruction. Rather than allow a shift followed by an add, these ould be separate instructions. In fact, if one were to ttempt to decrease the number of bits in each microinstruction to 16, he might be led to an instruction set typically found in 16-bit minicomputers. The 64-bit microinstructions of LX-1 are, on the average, four to six times more powerful than typical 16-bit instruction sets.

There appear to be many possible applications of LX-1. Microprograms have been written to simulate two small machines; the Digital Equipment Corporation PDP-8, and the Hewlett–Packard 2115A. Both took approximately 100 microinstructions and ran at real-time with respect to the machine being simulated. A 64-terminal communications line multiplexer has been coded. It also required less than 100 microinstructions and took about 1 μs per bit input or output (all serial/parallel conversion was done in the microprogram).

The usefulness of the design to a vocoder analyzer has been demonstrated, and the only required hardware addition is a 200-sample (slow) input buffer memory. The LSI version, which should be five times faster, could be used with no hardware modifications and should be fast enough to handle both analysis and synthesis simultaneously.

Thus, real-time digital signal processing is feasible in small microprogrammed computers, and the expected performance of LSI technology should make this method of operation even more attractive because of the potential cost advantage and the flexibility provided.

APPENDIX

Syntax for ML

The syntax for ML is based upon the extended BNF notation developed by Cheatham [8]. The metasymbol $\Rightarrow$ is equivalent to the BNF ::= and means "is defined as" or "consists of." Upper case alphabetics and all other characters (other than metasymbols) are terminal symbols, whereas strings of lower case alphabetic characters (in some cases separated with dashes) are used to define nonterminal syntactic elements. Exceptions are that space, tab, and cr stand for the obvious terminal symbols. The metasymbol | is used to separate and specify alternatives, brackets enclose alternatives, and brackets with subscripts and superscripts mean the following:

$[\ ]_j^i$	choose at least j but at most i of the alternatives
$[\ ]^i$	choose none or at most i
$[\ ]_j$	choose at least j with no upper limit
$[\ ]$	is equivalent to $[\ ]^1$
$\{\ \}$	is equivalent to $[\ ]_1^1$.

The following example means that a jwak is defined as a snex followed by at most 3 zats followed by exactly one carriage return:

$$\text{jwak} \Rightarrow \text{snex}[\text{zats}]^3\{\text{cr}\}.$$

In general, the microprogram source text may be liberally sprinkled with separators; only those places where separators are necessary are indicated in the following.

Preliminary

alphabetic-character ⇒ A | B | C | · · · | Z | 1
alphanumeric-character ⇒ alphabetic-character | 0 | 1 | · · · | 9 2
separator ⇒ space | tab 3
comment ⇒ $_*$ [any-character-but-cr]$^\infty$ 4
lname, cname, rname ⇒ alphabetic-character [alphanumeric-character]^{15} 5

General Program Structure

microprogram ⇒ definition-section assignment-section END cr 6
definition-section ⇒ [definition-line cr]$_1$ 7
definition-line ⇒ comment | definition [comment] 8
assignment-section ⇒ [assignment-line cr]$_1$ 9
assignment-line ⇒ comment | assignment-statement [comment] 10

Definitions

definition ⇒ register-declaration | constant-declaration 11
register-declaration ⇒ rname = octal-number-modulo-16 12
constant-declaration ⇒ cname ≡ octal-number-modulo-2^{16} | cname ≡ label 13

Assignments

assignment-statement ⇒ [label] [separator]$_1$ assignment [/[goto]] 14
assignment ⇒ left-side [→ right-side] 15
right-side ⇒ [dregsel] [, special-bit]^6 16
left-side ⇒ [aregsel [shiftop] logop] [−] bregsel | 17
aregsel [shiftop] + [−] bregsel [, carryin] | 18
aregsel [shiftop] − bregsel | 19
[aregsel] λ bregsel | 20
aregsel $\{\bar{x} | \underline{x}\}$ bregsel 21
bregsel, aregsel ⇒ rname | cname | octal-number-modulo-2^{16} 22
dregsel ⇒ rname 23
logop ⇒ ∨ | ∧ | ⊻ 24
shiftop ⇒ {/ | $_*$} {octal-number-modulo-16 | #} [, {special-bit | 0 | 1}] 25
carryin ⇒ {special-bit | 1 | 0} 26
label ⇒ lname [({1 | 0} [, {1 | 0}])] 27
goto ⇒ lname [({special-bit | 1 | 0} [, {special-bit | 1 | 0}]) [‖]] 28
special-bit ⇒ C | Z | H | L | S | F 29

Description of ML

A microprogram consists of a set of definitions (or declarations) and a set of assignment statements. The definitions allow the use of symbolic names for register identification and literal constants.

Example: Define register 5 to have the name XYZ:

$$XYZ = 5 \qquad [SX - 12].^2$$

Each assignment statement is translated into a 64-bit microinstruction for which the bit assignments are given in Table III. The basic form of an assignment statement is:

⟨A-bus select⟩ ⟨operation⟩ ⟨B-bus select⟩ →
⟨D-bus select⟩/⟨next instruction select⟩.

Example: Store the binary sum of the contents of registers ABC and XYZ into register XYZ:

$$ABC + XYZ \rightarrow XYZ \qquad [SX - 15, 16, 18].$$

If the next instruction field is missing, the next sequential instruction is assumed and the appropriate bits are filled in by the assembler.

Of the 16 addressable registers, register zero is wholly dedicated for immediate constants, register one is partly dedicated for the special bits and indirect shift count, and the remaining 14 registers are available for general use.

Example: Define the immediate constant PQ and load it plus register ABC into register XYZ. The three statements following the definition are equivalent.

$$PQ \equiv 177326 \qquad [SX - 13]$$
$$ABC + PQ \rightarrow XYZ \qquad [SX - 15, 16, 18]$$
$$ABC + 177326 \rightarrow XYZ$$
$$PQ + ABC \rightarrow XYZ$$

Of the 32 possible operations selectable, eight are for addition (eight different carry-in options), three are for the

TABLE III
MICROINSTRUCTION BIT ASSIGNMENTS

Key Syntax Line	Fig. 1 Name	Number of Bits	Function
22	A-Sel	4	Select 1 of 16 registers for A-bus
22	B-Sel	4	Select 1 of 16 registers for B-bus
23	D-Sel	4	Select 1 of 16 registers for D-bus
17	Operation-Select	5	Select 1 of 32 operations
16	Special-Bit-Select	6	Update or ignore 6 special bits
28	N	6	Select next address modulo 4
28	N1-Sel	3	Select second bit of next address
28	N0-Sel	3	Select low-order bit of next address
25	Shift-Indir	1	Select indirect shift count
25	Dir-Shift-Count	4	Direct shift count
25	Shiftin-Sel	3	Select shift-in
25	Shift-Left	1	Shift left or right
18	Complement	1	Complement B-bus
13	K	16	16-bit constant
—	No Overlap	1	Execution dependent next address
—	Parity	1	Control memory parity
	Spare	1	
		64	

^{2} SX—12 refers to syntax line 12, above. Only the key syntax lines are referenced. Numbers used in programming examples are octal; numbers used in the text are decimal.

gical operations AND, OR, and EOR, two are for reading nd writing the scratch memory, two are for the multiply perations, and the remaining are unassigned.

Example: Form the logical OR of ABC and XYZ and put e result in register XYZ.

ABC ∨ XYZ → XYZ [SX − 17].

Example: Increment register ABC. The first increments ith a carry-in, the second with a constant. The third adds ne or two to ABC depending on the state of special bit H.

ABC + 0, 1 → ABC [SX − 18]
ABC + 1 → ABC
ABC + 1, H → ABC

Example: Subtract ABC from XYZ assuming 2's comlement encoding:

XYZ − ABC → XYZ [SX − 19].

Example: Multiply ABC and XYZ and place the loworder 16 bits of the product in UVW:

ABC x XYZ → UVW [SX − 21].

Any subset of the special bits which are generated each ycle can be ignored or saved in register one. In the latter ase, the old values are lost.

Example: Save the carry (C) resulting from the addition of ABC and XYZ, and also save the low-order bit (L) of the sum.

ABC + XYZ → XYZ, C, L [SX − 16].

Program sequencing information is contained in each microinstruction. The location of the next instruction, modulo 4, is given explicitly, and the low-order two bits which complete the next instruction address may be given explicitly or selected from the special bits. Unconditional branches, two-way branches, or four-way branches are provided.

Example: Clear register ABC and branch to L1:

0 → ABC / L1 [SX − 14].

Example: Test the sign bit (H) of ABC and branch to L2(0) if the sign bit is 0, or branch to L2(1) if the sign bit is 1:

ABC → , H / L2(H) [SX − 14,28].

Example: Test the carry bit (C) and branch to L3(0, 1) if the carry bit is 0, or branch to L3(1, 1) if the carry bit is 1.

ABC / L3 (C, 1) [SX − 15,28].

The data on the A-bus can be shifted or rotated, right or left, up to 15 positions, and the data on the B-bus can be ones complemented prior to the addition or logical operations.

Example: Shift XYZ left three bits (shifting in zeros) and AND with the complement of ABC:

XYZ * 3 ∧ − ABC → PQR [SX − 17,25].

Example: Divide the 2's complement number in ABC by 16:

ABC → , H *save the sign-bit
ABC / 4, H → ABC.

Example: Rotate ABC left an amount equal to the number of zeros in XYZ. Assume COUNT is register 1 where the indirect shift count must be stored:

```
0 → COUNT
− XYZ → TEMP, Z, L / L3 (L, Z)
L3 (0, 0)  TEMP / 1 → TEMP, L, Z, / L3 (L, Z)
L3 (1,0)   COUNT + 1 → COUNT / L3 (0, 0)
L3 (0, 1)  ABC * #, F → ABC *shiftin = F causes rotate.
```

The low-order eight bits of data on the B-bus select the word to be written into or read from the scratch memory. For writing, the A-bus contains the data to be written, and for reading, the scratch memory output appears on the D-bus.

Example: Swap word 26 of the scratch memory with ABC:

ABC → TEMP
λ 26 → ABC [SX − 20].
TEMP λ 26

A special case of the next microinstruction selection bits has the effect of causing an unconditional branch of the microprogram to the location whose value is on the D-bus. This feature makes branch tables and subroutines possible.

Example: Call subroutine SINE. The subroutine will return to L2:

L2 ≡ L2 [SX − 13]
L2 → RETREG / SINE

*Subroutine return within SINE

RETREG → / [SX − 14].

ACKNOWLEDGMENT

The authors wish to express their gratitude to the many members of the Digital Computers Group who contributed to the LX-1 project. In particular, they wish to acknowledge the work of R. Kalin, who was responsible for the simulator and DOMEX, and S. Pezaris, who was responsible for the engineering.

REFERENCES

[1] H. Webber, "A microprogrammed implementation of Euler on IBM system/360 model 30," *Commun. ACM*, vol. 10, pp. 549–558, September 1967.
[2] G. D. Hornbuckle, "Representation, generation, and manipulation of computer graphical information," Ph.D. dissertation, Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, Calif., 1967.
[3] G. D. Hornbuckle and R. N. Spann, "Diagnosis of single-gate fail-

ures in combinational circuits," *IEEE Trans. Computers*, vol. C-18, pp. 216–220, March 1969.

[4] L. F. Mondshein, "Vital: compiler-compiler systems reference manual," M.I.T. Lincoln Laboratory, Lexington, Mass., Tech. Note 1967-12, February 1967.

[5] C. N. Mooers, "TRAC, a procedure-describing language for the reactive typewriter," *Commun. ACM*, vol. 9, pp. 215–219, March 1966.

[6] J. F. Teixeira and R. P. Sallen, "The Sylvania data tablet: a new approach to graphic data input," *1968 Spring Joint Computer Conf., AFIPS Proc.*, vol. 32. Washington, D. C.: Thompson, 1968, pp. 315–321.

[7] J. E. Curry, "A tablet input facility for an interactive graphics system," *Proc. Internatl. Conf. on Artificial Intelligence*. Washington, D. C., May 1969.

[8] T. E. Cheatham, Jr., "The theory and construction of compilers," Massachusetts Computer Associates, Inc., Wakefield, Mass., 1966.

[9] W. M. Anderson, Jr., "Specification of a digital vocoder system," presented at the Acoustical Society of America, Cleveland, Ohio, November 19–22, 1968.

[10] T. Bially, "Structure of a digital channel vocoder," presented at the Internatl. Conf. on Communications, Boulder, Colo., June 9–11, 1969.

[11] B. Gold, "Computer program for pitch extraction," *J. Acoust. Soc. Am.*, vol. 34, no. 7, pp. 916–921, July 1962.

Design of the Arithmetic Units of ILLIAC III: Use of Redundancy and Higher Radix Methods

DANIEL E. ATKINS, MEMBER, IEEE

Abstract—In keeping with the experimental nature of the Illinois Pattern Recognition Computer (ILLIAC III), the arithmetic units are intended to be a practical testing ground for recent theoretical work in computer arithmetic. This paper describes the use of redundant number systems and the design of a structure with which multiplication and division are executed radix 256. The heart of the unit is the stored-sign subtracter, a recently discovered member of the family of borrow–save subtracters and carry–save adders. A cascade of these subtracters, controlled by a multiplier recoder, provides multiplication. The same structure, controlled by a "model division" (a quotient recoder), performs division.

Index Terms—Arithmetic unit, computer arithmetic, division, higher radix arithmetic, ILLIAC III, multiplication, redundant number systems, signed-digit subtracter, stored-sign subtracter.

Introduction

In keeping with the experimental nature of the Illinois Pattern Recognition Computer (ILLIAC III), the arithmetic units are intended to be a practical testing ground for some recent theoretical work in computer arithmetic. The bulk of this work centers upon the use of redundant number systems and/or the use of higher radix methods. The design of the arithmetic units of ILLIAC III exhibits both techniques. They are of primary importance in the adder–subtracter, multiplication, and division structures.

ILLIAC III is a multiprocessor computer system. The four central processors share the two floating-point arithmetic units by way of an exchange net. The arithmetic units are therefore autonomous units with limited input–output facilities. A brief description of the ILLIAC III computer system together with the details of logic design is presented in [10].

Manuscript received June 16, 1969; revised February 20, 1970. This work was supported in part by the U. S. Atomic Energy Commission under Contract USAEC AT(11-1-1018), and in part by the National Science Foundation under Grant NSF-GP-46.

The author may be contacted in care of the Department of Computer Science, University of Illinois, Urbana, Ill. 61801. He is currently on active duty in the U. S. Army.

Adder–Subtracter

Redundancy is introduced into the adder–subtracter structure in order to limit carry–borrow propagation. This property is a key factor in the rapid execution of multiplication, since during the iterative steps, the partial product is formed without carry or borrow propagation. The product is converted to a nonredundant representation during a terminal step in which propagation does take place. One input, and the output of the adder–subtracter use a signed-digit representation. Two bits, one a sign, the other a magnitude, are associated with each digital position. This structure also exhibits several properties not found in the conventional carry–save adder or borrow–save subtracter.

Multiplication

Elsewhere than in the adder–subtracter complex, high-speed operation is also obtained by extensive use of redundancy and by executing operations in radices greater than 2. Multiplication, for example, is performed radix 256, i.e., eight bits of the multiplier are retired in one pass from the primary to the secondary rank of the accumulator. By recoding, redundancy is introduced into the multiplier in such a manner that all the required multiples of the multiplicand may be formed merely by shifting.

Division

In division, redundancy is introduced into the representation of the quotient. As a consequence, quotient digits