IBM® Reference Manual

PERFORMANCE CHARACTERISTICS

7030 Data Processing System

volume 1

VOLUME 1

IBM® Reference Manual

PERFORMANCE CHARACTERISTICS

7030 Data Processing System

March 8, 1961

INTERNATIONAL BUSINESS MACHINES CORPORATION

Contents

Contents

Illustrations

Tables

1
Foreword

The performance of a system as complex in concept as STRETCH is extremely difficult to evaluate. The operation time for each unit is basically dependent on the particular operation it is performing; the performance of the system as a whole is dependent on a complex mix of the individual units and their interactions with one another. Furthermore, the performance of the system with respect to a synchronous machine will vary, depending on the particular program which is being run, and no single simple measure of performance can be obtained.

There are many independent criteria for determining machine performance. Each user is ultimately interested in the ability of the machine to solve competently, in a feasible financial and technical fashion, the problem or problems peculiar to individual requirements. While we do not have exhaustive information on total system performance, all the problem applications that are available for release at this time are discussed in the following pages. Other problems have been run, but IBM does not have permission at this time to distribute this data.

The STRETCH remains, in our honest judgment, the most powerful and potentially productive piece of computing machinery available in the world today. It represents a real challenge to those with problems heretofore unprocessable on older machines with anything like reasonable efficiency. It is a machine ideally suited for special large problems drawn from the areas of matrix multiplication and inversion, linear programming, Leontieff input/output models, three dimensional non-linear partial differential equations, and certain areas of simulation. The area of application potential is discussed in greater detail in the following pages.

The information in this report shows many aspects of the performance of the STRETCH system and contains examples of test programs which have been run and timed.

2
Programming

7030 PROGRAMMABILITY

Unlike earlier IBM data processing systems, the 7030 makes very extensive use of overlapped operations. While this overlapping adds significantly to the over-all speed of the machine, it does complicate the problem of writing optimized programs because the time taken to perform a given operation depends on whether or not the look-ahead feature has been able to locate and bring from memory the required data while previous operations were being performed. Thus, the order in which instructions are written can be important if an optimal program is required. In many instances the programmer can forget about such considerations without significant loss of speed. There are, however, some situations to which the programmer must pay special attention if maximum speed is to be attained.

In order to help programmers use the 7030 system efficiently, a number of hints have been drawn up, and these are given below. The basis of these hints is twofold: a theoretical study of the 7030 logical organization, and limited practical experience of running the 7030 programs. As more operating experience on the 7030 is gained, it must be expected that further rules or hints for good programming will be developed.

Hints Towards Good 7030 Programming

Generalities

1. Efficiency in computer problem solving involves the balancing of the following factors:

a. Accuracy of results
b. Analysis effort
c. Programming time
d. Debugging time
e. Production run time
f. Effectiveness in repeated use of program (possibly by a stranger)

The relative weights of these factors vary from problem to problem, individual to individual, and from installation to installation. For small one-shot problems the trend is towards the emphasis on a, b, c, d.

2. Timing is important for much traversed inner loops, but usually less important elsewhere.

3. There are usually many ways of doing the same problem.

4. The 7030 will not be efficiently used when the programmer tries to make it look like machine X.

5. Advantage should be taken of special features in STRAP and MCP to minimize errors and to simplify debugging.

6. Machine efficiency is gained by distributing the work over as many major units as possible so that at any given time no major unit is idle.

7. Memory conflict can be largely removed by putting instructions and data in separate memory box groups.

8. Information transmittal between autonomous major units is through buffer registers. The I-box buffers 1Y, 2Y and the look-ahead buffer levels LA0, LA1, LA2, and LA3 should not be left empty over extended lengths of time. Nor should they be constantly crowded by data with little information content.

9. It is perfectly permissible to use floating point operations on VFL quantities or binary operations on decimal quantities.

Specifics

1. Floating point operations are usually E-box limited in timing (exceptions *L*, *LWF*, *DL*, *DLWF* and *ST*). VFL operations involve extensive decoding and execution time, and are usually much slower than the floating point counterparts. I-box operations usually do not involve the E-box, and the I-box time can be covered largely by neighboring floating point operations.
2. I-box fetches are less efficient than E-box fetches, since the latter are greatly enhanced by look-ahead buffering.
3. *SV*, *SC*, *SR* are more time consuming than *SX*, for the latter does not call for an I-box fetch.
4. Information transfer from the E-box to the I-box is relatively time consuming, but is still faster than, say from the E-box to main memory, then immediately from main memory to the I-box.
5. Immediate operands require no fetch and are to be preferred, particularly for I-box operations.
6. All VFL stores are fetch-and-stores. Every *BB* involves a fetch, VFL arithmetic, and a store. *B(ind)* for non-index indicators is similar to *BB* except that I-box activity is less extensive, and the fetch-store involve an internal operand ($IND).
7. VFL information will be processed more efficiently if word boundary crossover is not present. Otherwise there will be over-exercising of the memory and LA.
8. Take advantage of forwarding, but avoid (if easily accomplished) other types of store-close-to-fetch.

9. Avoid consecutive stores since they are time consuming, as is forwarding more than once. The I-box otherwise would be standing still and LA gradually drained.
10. All successful branch instructions will temporarily remove the I-box buffer.
11. The following branches are considered unconditional by the I-box:

    CB and varients

    B(ind) for *XF*, *XVLZ*, *XVZ*, *XVGZ*, *XCZ*, *XL*, *XE*, *XH*

    and are performed correctly by the I-box.
12. The following are considered by I-box to be truly conditional branches:

    B(ind) for non-index indicators

    BB

    I-box makes the tentative assumption that the branch is not successful, and processes ahead. If the assumption proved wrong, branch-recovery will be performed, which requires the cleaning of I-box, restoring of pre-processed index registers, and cleaning of LA before resumption of normal activities. Conditional branches should be largely unsuccessful, even if an additional (unconditional) branch instruction has to be added to the program.
13. *BD*, *RNX*, *T* and *SWAP* require cleaning of LA.
14. Interruption involves cleaning of the I-box, restoring of index registers, execution of a pseudo *B(ind)* instruction, fetching, and execution of a free instruction before the resumption of normal activities. Judicious use of this feature, however, allows the writing of inner loops with few time consuming branch instructions.

15. VFL instructions require from 2 to 6 levels of LA and involve the slower byte processing. The power of such instructions (particularly the logical connective instructions), however, frequently compensates for the slow speed.
16. Begin a loop with a full-word address, even if a CNOP has to be placed just prior to the loop. This avoids repeatedly fetching an instruction which is not used.
17. For optimum speed, fetch-type I-box instructions should occupy the second half of a full word.
18. The following special registers are *bona fide* memory locations, and are subject to the usual memory restrictions:
    0. ($Z), 4. ($MB), 13. ($RM), 14. ($FT), 15. ($TR).
    The load factor instruction thus involves a fetch and a store.

APPLIED PROGRAMMING SYSTEMS FOR THE IBM 7030

Listed below are the systems programs which have been designed and are now being written and tested for the 7030 system. Their purpose is to provide efficient and productive use of the computer system, to permit applications to be programmed easily, and to assist IBM 704, 709, and 7090 users in making the transition to the 7030.

Briefly, the programs being provided are: 704, 709, and 7090 programs for simulating the 7030 and assembling for it; programs for simulating the 704 or 709 on the 7030; a master control program for 7030 operations; and processors for STRAP symbolic, SMAC macro, and FORTRAN language programming. Programs will be released in field test version according to the schedule in Table 1.

IBM 7030 Packages for the 704, 709, and 7090.

These packages permit programming to be partially checked in advance of 7030 installation. They consist of an assembly program, STRAP I, and a 7030 simulator. Versions are available for each of the three different machines: 704, 709, and 7090.

> Note
>
> STRAP I was designed in a joint effort by Los Alamos Scientific Computing Laboratory and International Business Machines Corporation personnel, and it was programmed by Los Alamos personnel.

STRAP I accepts all 7030 instructions plus some of the pseudo-instructions of STRAP II. All programs written for STRAP I are accepted by STRAP II. The output of STRAP I can be executed directly on the 7030 or (via the simulator) on the 704, 709, or 7090. The speed of execution on the 704 or 709 is several thousand times slower than on the 7030. No attempt is made to simulate the timing details of I/O operations.

IBM 704 and 709 Simulators for the 7030.

These programs permit 704 and 709 programs to be run on the 7030 without reprogramming. The simulated machine has 32K memory, 8 logical drums, 10 tape units, printer, punch, and operator console. The console is simulated on the 7030 console and includes all features of the 704 or 709 console.

Note

If the size of 7030 memory is reduced, the size of the simulated machine is also reduced. The following table obtains:

7030 Memory	704 or 709
16K	4K or 8K without drum
32K	8K without drum
48K	32K without drum
65K	Maximum configuration

One tape per tape to be simulated.

There is no attempt to simulate CRT output. In addition, the PSE instructions to the printer exit hubs are handled according to the SHARE II board. Since there is no standard board for the punch, the PSE instructions to the punch exit hubs are NOP's. Otherwise the simulation duplicates as closely as possible the actual performance of the 704 or 709.

Due to the different word lengths, it is necessary to pre- or post-process binary tapes to be communicated between the 704 or 709 and the 7030. The 7030 programs to perform these operations are included as part of the package.

In general, the simulator is about three times slower than the 704 or 709. Floating point instructions are somewhat faster than this and fixed point instructions somewhat slower. Because of the speeds of the 7030 I/O units, programs can run faster on the 7030 than on the 704 or 709 if they are I/O limited on those machines.

Master Control Program

The master control program is an automatic operating system which runs job after job automatically. It plans the actual assignment of symbolic I/O units in advance, so as to minimize conflicts and delays between successive jobs, issuing tape mounting and demounting instructions to the operator and checking (through reel labels) that tapes are mounted correctly. For each job, MCP offers the options of COMPILE, GO, and COMPILE and GO. Here COMPILE can refer to any of the several language processors listed below. MCP also arranges for checkout and postmortem procedures where needed. Also, it is the agent by which further service routines (such as an installation logging routine) can be easily added to the operating system.

In the individual program, MCP provides a complete input/output system. In particular, an option is provided for buffered operation of the card reader and high-density blocked input and output SPOOL tapes, permitting easy and efficient overlap of computing with the input of program and data and the output of results. Standard methods for reading, printing, and punching data are provided. Also, all interrupts are monitored by MCP. I/O interrupts are returned to the program in a form convenient to manipulate. The programmer can designate how maskable interrupts are to be processed.

Including buffer areas, MCP occupies 8K of storage space. An IBM 1401 tape system is required for off-line operations. The time required for the system to exercise its supervisory functions is not yet known in detail, but is estimated to be small compared with the time saved by its use.

STRAP II Assembly

STRAP II is a symbolic programming system for the 7030. It defines a complete set of mnemonics for the 7030 instructions, together with the pseudo-operations for data definition and such assembly operations as origin setting, space reservation, listing control, and identification of output.

Features provided for are as follows: acceptance of programmer symbols up to 128 characters in length; acceptance of source language numerical information written with radix 2 through 10, and 16; address arithmetic involving addition, subtraction, multiplication, and division; error messages on the output listing; extensive tailing facilities that permit up to 10 unique levels of tails to be appended to programmer symbols; and the option of saving the symbol table for subsequent purposes.

STRAP II operates on a minimum size 7030 computer (24K plus disk). The first version operates independently; a later version will be adapted to operation by MCP.

SMAC

SMAC processes macro-instructions of the simple substitution type (but permitting macros within macros), thus adding the next higher level to the machine language of STRAP II. The result is a conveniently open-ended machine-oriented language.

SMAC runs on a minimum 7030 and is operated by MCP.

FORTRAN

The FORTRAN processor handles the FORTRAN II statements. Any standard FORTRAN II program is accepted and converted into a

program suitable for further processing by SMAC. Much of the processor is of a general purpose nature and is expected to be useful in other advanced programming systems the user may care to develop.

The speed of compilation from FORTRAN to machine language is expected to be approximately twice that of the new 7090 FORTRAN now being developed by applied programming. The object programs which result are expected, for typical FORTRAN applications, to run at an average of 75 percent as fast as equivalent programs which have been carefully hand-coded. This degree of efficiency is obtained by including throughout the processor-chain a large number of the criteria for efficient 7030 programming listed earlier. Further such rules are likely to be discovered in the future, and the structure of the compiler is expected to be flexible enough to accommodate most of them.

FORTRAN runs on a minimum 7030 and is operated by MCP.

TABLE 1. STATUS OF 7030 PROGRAMMING SYSTEMS

Systems	Status as of March 1, 1961	Approx. Number of Instructions	Manual Availability	Estimated Program Completion Date
704/709/7090 Package	Operational	24,000	Available now as form No. C22-6531-1	---
704/709 Simulator	Coding completed	5,000	Aug. 1961	Aug. 1961
STRAP II	Coding completed	14,000	Reference Manual April 1961. Operators Bulletin June 1961.	July 1961
MCP	Coding completed	8,000	Preliminary edition of user's guide now available as form No. J22-6559. Reference Manual Oct. 1961	Oct. 1961
SMAC	Coding completed	4,000	Sept. 1961	Sept. 1961
FORTRAN	Coding in progress	50,000	Preliminary Bulletin July 1961	March 1962

3
Execution Times

RAW SPEEDS AND THEIR INTERPRETATIONS

E-Box Times for Basic Instructions

Floating Point Instruction Times

Following is a summary of "raw" floating point execution. The times are predicated upon the availability of data and instructions when needed; that is, the times given are the maximum speed at which the floating point unit may operate. If some memory access (look-ahead, I-box, etc.) factors enter such that data and instructions are not available when the floating point unit is able to accept them (or requires them), the extra delay thus caused is added to the times below.

The times are broken up into two types of cycles: pre-execution and execution, defined as follows:

- Pre-Execution – That part of the instruction which may be executed before any modification of an addressable register occurs. Floating point instructions are begun as soon as data paths are free and the instruction and initial addressed operand is made available. This time will overlap checking of a previous instruction, indicator setting, interrupt testing, memory storage, etc. These cases may lead to a condition such that the operation should never have been started (interrupt occurs as result of previous instructions). In such a

case the pre-execution is terminated and no addressable registers would be modified. (As far as programmer is concerned pre-execution never occurred.)

- Execution – The part of the instruction which follows the initial modification of an addressable register up to the point when the next floating point instruction may begin.

The times are listed (table 2) in terms of the basic machine cycles, which is presently 0.3 microsecond.

TABLE 2. FLOATING POINT INSTRUCTION TIMES

Instruction	Pre-execution	Execution
+(Add) *+MG (Add to Mag.)* *M+ (Add to Mem.)* *M+MG (Add Mag to Mem.)* *K (Compare)* *KMG (Compare Mag.)* *KR (Compare for Range)* *KMGR (Compare Mag. for Range)*	4 cycles (pre shift ≤ 3) 1 cycle for each Additional pre-shift of 4	1 cycle (norm ≤ 6) 2 cycles for each additional norm. of 6 2 cycles if recomplement is necessary - note 1 2 cycles if forced zero occurs as result of "mag." operation.
D+ (Add Double) *D+MG (Add Double to Mag.)* *F+ (Add Fraction)*	Same as Add Class	Same as Add class plus 1 cycle if immediate next instruction is floating point
L (Load) *LWF (Load with Flag)* *LFT (Load Factor)* *ST (Store)*	2 cycles	1 cycle (norm ≤ 6) 2 cycles for each additional norm. of 6
DL (Load Double) *DLWF (Load Double with Flag*	2 cycles	1 cycle (norm ≤ 6) 2 cycles for each additional norm of 6 1 cycle if immediate next instruction is floating point
SRD (Store Rounded)	3 cycles	1 cycle (norm ≤ 6) 2 cycles for each additional norm of 6
SLϕ (Store Low Order)	2 cycles	17 cycles - unnormalized 15 cycles if normalized and there are no leading zeros in intermediate fraction 2 cycles for each norm. of 6 or less

TABLE 2. FLOATING POINT INSTRUCTION TIMES (cont'd)

Instruction	Pre-execution	Execution
SHF (Shift Fraction)	Applies to both left and right shift	
	4 cycles (shift ≤ 3) 1 cycle for each additional shift of 4	1 cycle 1 cycle if immediate next operation is floating point
E+ (Add Exponent) *E+I (Add Immed. to Exp.)* *- note 2*	1 cycle	5 cycles (norm ≤ 6) 2 cycles for each additional norm. of 6 1 cycle if immediate next operation is floating point
**(Multiply)*	5 cycles	4 cycles (norm. ≤ 6) 2 cycles for each additional norm. of 6
D (Multiply Double)*	5 cycles	4 cycles (norm ≤ 6) 2 cycles for each additional norm. of 6 1 cycle if immediate next operation is floating point
**+(Multiply and Add)*	3 cycles cannot enter execution cycles until Factor is available	17 cycles (pre shift ≤ 3 and norm ≤ 6) 1 cycle for each additional pre-shift of 4 2 cycles for each additional norm. of 6 2 cycles if recomplement is necessary - note 1 1 cycle if immediate next operation is floating point
R/ (Reciprocal Divide)	2 cycles	2 cycles plus all pre-execution and execution cycles of Divide

TABLE 2. FLOATING POINT INSTRUCTION TIMES (cont'd)

Instruction	Pre-execution	Execution
/(Divide)	2 cycles (divisor shift ≤ 6) 2 cycles for each additional divisor shift of 6	2 cycles (dividend shift ≤ 6) 2 cycles for each additional dividend shift of 6 or total of 3 cycles if dividend is all zero 2 cycles for initial reduction loop if dividend is normalized 3 cycles for initial dividend pass if dividend has leading zeros. 2 cycles for each additional reduction loop (number of cycles is data dependent). 2 cycles if final remainder has to be complemented 1 cycle - all
	If zero divisor 3 cycles pre-execution and	2 cycles execution instead of previous
D/ (Divide Double)	Same as *Divide*	Same as *Divide* plus 6 cycles (remainder norm ≤ 6) 2 cycles for each additional remainder norm. of 6
SRT (Store Root)	2 cycles	106 cycles (norm. ≤ 6) 1 cycle if operation began with a "B" pulse 2 cycles for each additional norm of 6

NOTE 1 - When fraction signs are unlike, the operand which is pre-shifted will be complemented. Recomplementing will only occur if the complemented (following pre-shift) fraction is larger in magnitude than unshifted fraction.

If fraction signs are alike no complementing occurs.

NOTE 2 - Add Exponent and Add Immediate to Exponent will most likely be changed in the near future to three pre-execution and three execution cycles.

Serial Arithmetic Execution Times

The execution times of SAU instructions can be presented in terms of some basic equations shown in table 2 below.

The column headings are defined as follows:

- Operation Code - instruction abbreviations.
- Pre-execution Time - time required to decode operation and set up controls.
- Execution Time - time required to perform the instructed function.
- Termination Time - time required to set indicators and clear unit.
- Full Word Total - a computed time in microseconds for the operations using unsigned full word operands which produce no arithmetic carries.
- Comments - variations or additions to the execution time equations.

All equations were developed for unsigned operations. Signed operations can be computed by adding .6 microseconds to the pre-execution time. An additional .6 microseconds must be added if the result of the operation requires complementing. The following operations require no additional time when signed: *SRD, C, CM, CT, CV, DCV, LCV (D-B), LTRCV (D-B), LTRS, LFT*.

TABLE 3. EQUATIONS FOR SAU INSTRUCTIONS

Operation Code	Pre-execution Time	Execution Time	Termination Time	Full Word Total	Comments
+	1.8	$.6(\frac{x}{y} + z)$	.6	7.2 us	Unlike signs CD > AB, add $.6(\frac{os}{r} + \frac{x}{y} + Z + 1)$ to the execution time.
+MG	1.8	$.6(\frac{x}{y} + z)$	.6	7.2 us	Unlike signs CD > AB, add .6 to the execution time.
L	1.8	$.6(\frac{x}{y})$	.6	7.2 us	
LWF	1.8	$.6(\frac{x}{y})$	.6	7.2 us	
M+	1.8	$.6(\frac{x}{y})$	.6	7.2 us	Unlike signs AB > CD, add $.6(\frac{x}{y} + 1)$ to the execution time.
M+MG	1.8	$.6(\frac{x}{y})$	.6	7.2 us	Unlike signs AB > CD, add $.6(\frac{x}{y} + 1)$ to the execution time.
M+1	1.8	.6(c + 1)	.6	3.0 us	Unlike signs, execution time = $.6(\frac{x}{y})$ when true result or $.6(2\frac{x}{y} + 1)$ when complement result.
ST	1.8	$.6(\frac{x}{y})$	.6	7.2 us	
SRD	3.0	$.6(\frac{x}{y})$	.6	8.4 us	No additional time required when operation is signed.
K	1.8	$.6(\frac{x}{y} + \frac{w}{r})$	.6	7.2 us	
KF	1.8	$.6(\frac{x}{y})$	.6	7.2 us	
KE	1.8	$.6(\frac{x}{y} + \frac{w}{r})$	.6	7.2 us	
KFE	1.8	$.6(\frac{x}{y})$	.6	7.2 us	

TABLE 3. EQUATIONS FOR SAU INSTRUCTIONS (cont'd)

Operation Code	Pre-execution Time	Execution Time	Termination Time	Full Word Total	Comments
KR	1.8	$.6(\frac{x}{y} + \frac{w}{r})$	.6	7.2 us	
KFR	1.8	$.6(\frac{x}{y})$	.6	7.2 us	
C	1.8	$.6(\frac{x}{y})$	.6	7.2 us	
CM	1.8	$.6(\frac{x}{y})$	.6	7.2 us	
CT	1.8	$.6(\frac{x}{y})$	.6	7.2 us	
LTRS	2.4	$.6(\frac{v}{y} + 3)$	.6	9.6	If the effective FL is greater than 48 bits, add $.6(\frac{t}{8} - 1)$ to the execution time.
LFT	2.4	$.6(\frac{v}{y}) + 3)$	.6	9.6	If the effective FL is greater than 48 bits, then add $.6(\frac{t}{8} - 1)$ to the execution time.
B - D LCV	2.4	$.6(x + \frac{u}{8} + 1)$	.6	45.3	
D - B LCV	2.4	$.6(\frac{x}{y} + \frac{t}{4} + \frac{u}{8} + 4)$	.6	14.7	
B - D LTRCV	2.4	$.6(x + 3)$	.6	29.4	If the converted field length is greater than 48 bits, add $.6(\frac{FLc}{8} - 1)$ to the execution time.
D - B LTRCV	2.4	$.6(\frac{x}{y} + 16)$	.6	21.6	If the field length to be converted after all zone bits have been removed is greater than 48 bits, add $.6(\frac{t}{8} - 1)$ to the execution time.
B - D CV	2.4	$.6(s + \frac{u}{8} + 1)$	.6	37.2	

TABLE 3. EQUATIONS FOR SAU INSTRUCTIONS (cont'd)

Operation Code	Pre-execution Time	Execution Time	Termination Time	Full Word Total	Comments
B - D DCV	2.4	$.6(s + \frac{u}{8} + 1)$	.6	63.0	
D - B CV	2.4	$.6(\frac{s}{4} + 1)$	.6	10.8	
D - B DCV	2.4	$.6(\frac{s}{4} + 2)$	.6	18.6	
*	6.0	3.9	.6	10.5	
* +	7.8	3.0 + *Add* exec. time	.6	18.6	
/	5.4	Q	.6	24.0	$Q.6(\frac{DR\ L\ "O"}{6} + \frac{DDL\ or\ DRL}{8} + A + \frac{97 - DRL}{8} + 1)$ A = 1. Zero DD $-\frac{DDL - DRL}{6}$ 2. Leading zero's $-(1 + \frac{\#L\ "O"}{6} + \frac{DDL - DRL - \#L\ "O"}{4})$ 3. Complement Result. -Add 1 to A2.

Glossary of Terms

DD	Dividend
DR	Divisor
DDL	Dividend length
DRL	Divisor length
L "O"	Leading zero's
#L "O"	Number of leading zero's
DDL or DRL	Dividend or divisor length which ever is greater
OS	Offset
FL	Field length
BS	Byte size
B - D	Binary to decimal
D - B	Decimal to binary
R	8 in Binary - 4 in decimal
S	Accumulator field specified by the offset minus all leading zero's
T	Field length less all zone bits
U	Result field
V	96 in binary - 92 in decimal
W	The number of accumulator significant bits greater than the field length
X	Field length in unsigned operations - field length minus the byte size in signed operations
Y	8 in binary operations and byte size in decimal operations
Z	One for a carry out of the last $\frac{x}{y}$ byte plus one for a carry out of each succeeding 8 bit byte in binary or 4 bit byte in decimal
C	The number of times a carry results from an 8 bit byte in binary or a 4 bit byte in decimal

Timing of I-Box Instructions

The following tables list the I-box times for decoding for execution of I-box instruction.

TABLE 4. I-BOX TIMES – DIRECT INDEX

Instruction	Address in EM	Address in XS	Address in IR
LX	4.8 μsec + dec	4.8 μsec + dec	4.2 μsec + dec + LAdr
LV, LC, LR	4.2 μsec + dec	4.2 μsec + dec	3.6 μsec + dec + LAdr
SX	1.2 μsec + dec + LAAR	4.2 μsec + dec	1.2 μsec + dec + LAAR
SV, SC, SR	4.2 μsec + dec + LAAR	5.4 μsec + dec	3.6 μsec + dec + LAdr + LAAR
SV, SC, SR to TC		6.0 μsec + dec + LAdr	
V +	4.2 μsec + dec	4.2 μsec + dec	3.6 μsec + dec + LAdr
V + C	4.2 μsec + dec	4.8 μsec + dec	4.2 μsec + dec + LAdr
V + CR (EM)	8.4 μsec + dec	9.0 μsec + dec	8.4 μsec + dec + LAdr
V + CR (XS)	7.2 μsec + dec	7.8 μsec + dec	7.2 μsec + dec + LAdr
KV, KC	4.2 μsec + dec	4.2 μsec + dec	3.6 μsec + dec + LAdr
RNX	9.0 μsec + dec + LAdr + LAAR		
LVE	7.8 μsec + dec	6.6 μsec + dec	6.6 μsec + dec + LAdr
LVE as object INSN of LVE	Additional 4.8 μsec	Additional 3.6 μsec	Additional 3.0 μsec + LAdr (4.2 if LAMT)
SVA	4.2 μsec + dec + LAAR	5.4 μsec + dec	3.6 μsec + dec + LAdr + LAAR
SVA to TC		6.0 μsec + dec + LAdr	

ABBREVIATIONS: dec = decade
LAAR = Look-ahead address register
LAdr = Look-ahead drain
LAMT = Look-ahead empty
TC = Time clock

TABLE 5. I-BOX TIMES– IMMEDIATE INDEX

Instruction	Basic Time	Variations
LVI, LCI, LRI, LVNI	2.4 μsec + dec	
V + I, V - I	2.4 μsec + dec	
V + IC, V - IC	3.0 μsec + dec	Also V ± ICR where Count ≠ 0
V + ICR, V - ICR	7.2 μsec + dec	1.2 μsec less if Refill from X5
C + I, C - I	2.4 μsec + dec	
KVI, KVNI, KCI	2.4 μsec + dec	
LVS	5.4 μsec + dec	Additional 1.8 μsec per ADD

TABLE 6. I-BOX TIMES – MISCELLANEOUS INSTRUCTIONS

Instruction	Address in EM	Address in XS	Address in IR
R (from EM)	7.8 μsec + dec + LAAR	7.2 μsec + dec	6.6 μsec + dec + LAAR + LAdr
R (from XS)	5.4 μsec + dec + LAAR	5.4 μsec + dec	4.2 μsec + dec + LAAR + LAdr
RCZ (C ≠ 0)	4.2 μsec + dec	3.0 μsec + dec	3.0 μsec + dec + LAdr
Z	1.2 μsec + dec + LAAR	3.6 μsec + dec	1.2 μsec + dec + LAAR
EX	1. Decode time, plus 2. Always begins with Look-ahead drain, plus 3. Instruction fetch time as follows: 4. Add decode and execution time for given instruction 5. Always ends with LA drain, plus 6. Add 3.0 μsec for fetch of next instruction		

Address	Time
0.0, 0.32	4.8 μsec
1.0, 1.32	6.0 μsec
2.0, 2.32	7.8 μsec
3.0, 3.32	7.8 μsec
4.0, 4.32	7.8 μsec
5.0 → 11.32	11.2 μsec
12.0, 12.32	7.8 μsec
13.0 → 14.32	5.4 μsec
15.0, 15.32	4.8 μsec
16.0 → 30.32	4.2 μsec
31.0, 31.32	6.0 μsec
32.0 → up	5.4 μsec

TABLE 6. I-BOX TIMES – MISCELLANEOUS INSTRUCTIONS (cont'd)

EXIC	1. Decode time, plus 2. Always begins with a Look-ahead drain a) 4.2 μsec to fetch the psuedo-instruction counter are overlapped with the Look-ahead drain if the ps IC is in EM. b) 3.0 μsec to fetch the psuedo-instruction counter are overlapped with the Look-ahead drain if the ps IC is in XS. 3. Instruction fetch time = 6.0 μsec 4. Stepping of psuedo-instruction counter a) Add LAAR time if psIC in EM b) Add .6 μsec time if psIC in XS 5. Add decode and execution time for given instructions 6. Always ends with LA drain, plus 7. Add 3.0 μsec for fetch of next instruction

TABLE 7. I-BOX TIMES – UNCONDITIONAL BRANCH

Instruction	Basic Time	SIC to EM	SIC to XS
B	3.6 μsec + dec	4.8 μsec + dec + LAAR	4.2 μsec + dec
BR	4.8 μsec + dec	6.0 μsec + dec + LAAR	5.4 μsec + dec
BE	3.6 μsec + dec + LAdr (if previously disabled)	4.8 μsec + dec + LAAR + LAdr (if previously disabled)	4.2 μsec + dec + LAdr (if previously disabled)
BEW	Setup time as above on BE		
NOP	1.2 μsec + dec	Same as Basic	Same as Basic
BD	4.8 μsec + dec + 2 LA drains	6.0 μsec + dec + 2 LA drains	5.4 μsec + dec + 2 LA drains

TABLE 8. I-BOX TIMES – INDEX BRANCHES

Instruction	Basic Time	SIC to EM	SIC to XS
CB (Successful)	4.2 μsec + dec	6.6 μsec + dec + LAAR	7.2 μsec + dec
CB (Unsuccessful)	3.6 μsec + dec	3.6 μsec + dec	3.6 μsec + dec
CBR (EM) (Succ)	6.6 μsec + dec	9.0 μsec + dec + LAAR	9.6 μsec + dec
CBR (XS) (Succ)	6.0 μsec + dec	8.4 μsec + dec + LAAR	9.0 μsec + dec
CBR (EM) (Unsucc)	6.0 μsec + dec	6.0 μsec + dec	6.0 μsec + dec
CBR (XS) (Unsucc)	5.4 μsec + dec	5.4 μsec + dec	5.4 μsec + dec

NOTE: CBR behaves like CB if Refill is not to be taken.
Ex: CBR, branch on count ≠ zero

TABLE 9. I-BOX TIMES - TRANSMIT INSTRUCTIONS

Instruction	Setup time	Loop time	Termination time
T (EM → EM)	1.2 usec + dec + LA dr	3.0 usec	3.6 usec
T (EM → XS)	1.2 usec + dec + LA dr	4.2 usec	.6 usec
T (XS → EM)	1.2 usec + dec + LA dr	3.0 usec	3.6 usec
T (XS → XS)	1.2 usec + dec + LA dr	4.2 usec	.6 usec
T (IR → EM)	1.2 usec + dec + LA dr	7.2 usec	3.6 usec
T (IR → XS)	1.2 usec + dec + LA dr	7.2 usec	.6 usec

Note: Transmit to EM and IR identical in time.

TABLE 9. I-BOX TIMES - TRANSMIT INSTRUCTIONS (cont'd)

Instruction	Setup time	Loop time	Termination time
S (EM → EM)	1.2 usec + dec + LA dr	6.0 usec	1.8 usec
S (EM → XS)	1.2 usec + dec + LA dr	6.6 usec	1.8 usec
S (EM → IR)	1.2 usec + dec + LA dr	9.6 usec	1.8 usec
S (XS → EM)	1.2 usec + dec + LA dr	6.0 usec	1.8 usec
S (XS → XS)	1.2 usec + dec + LA dr	7.2 usec	1.8 usec
S (XS → IR)	1.2 usec + dec + LA dr	10.8 usec	1.8 usec
S (IR → EM)	1.2 usec + dec + LA dr	9.6 usec	1.8 usec
S (IR → XS)	1.2 usec + dec + LA dr	10.2 usec	1.8 usec
S (IR → IR)	1.2 usec + dec + LA dr	15.0 usec	1.8 usec
Note: The immediate/direct and the forward/backward options do not affect the timing of either transmit or swap.			

Analysis and Interpretation of the New Speeds Relative Machine Environment

Machine Organization

In conventional machines the instruction time is dependent upon the total length and delays along information paths, and the hardware places a severe limitation on performance.

The organization of the 7030 has been devised to free the machine from such limitations. To a large extent, the number of obstacles along the information path is not of crucial importance; the speed and performance of the machine is governed by the average frequency of information access. For example, a 7030 box has a readout time of 1 μs and a recycle time of 2.2 μs, yet in the 7030 system information can be obtained at the rate of one word every 0.3 μs.

The 7030 system organization is characterized by the local autonomy of the major units: the memory bus control unit (MBCU), the I-box, the look-ahead (LA), the E-box, the exchange, and the disk exchange. (For definitions of these terms the reader is referred to Appendix A.)

Each major unit is responsible for processing at top speed as long as there is data to process. The local autonomy of the MBCU, exchange, and disk exchange means that the central processing unit (CPU) can operate independent of I/O operations. Within the CPU this local autonomy means that temporary delays up to several μs in one unit can be tolerated without slowing down the entire pipe line. Extensive buffering of information is needed to absorb such temporary delays, and within the 7030 CPU at any given time up to ten instructions can be in various stages of processing. Within wide limits, the times for instruction processing is not the sum of: Instruction fetch, instruction error check, decoding, operand fetch, operand error check, and execution but is the maximum of these times averaged over several instructions.

The E-box times of the instructions listed in the previous section are the times realizable if the LA levels (which are buffers to the E-box) can always supply needed information to the E-box. The machine organization is such that this is usually the case.

For such a loosely-coupled machine the information paths are actually longer and the number of obstacles larger than conventional tightly-organized machines. It is possible to create situations to make the information path time influence instruction time. All of these situations, so far as the CPU is concerned, have an effect on the LA buffering, which in turn affects E-box performance.

Effect of Memory Interleaving

With the instruction buffers 1Y, 2Y (capable of housing four half-word instructions), in the I-box, and the four LA levels, the advantage of memory interleaving is fully exploited.

The demand of the E-box on LA is usually no higher than one word per 1.5 μs (sequence of floating adds) and on the average, the memory is not a factor. In terms of the four-level look-ahead, the requirement is satisfied if any four E-box demands are fulfilled in 6 μs. The memory interleave scheme allows four words every 2.2 μs (four box interleaving) or four words every 4.4 μs (two box interleaving).

In actual computations, with the instructions occupying the two lower memory boxes and data occupying the four upper memory boxes, memory conflict is not expected to be an important factor, even if the I/O units are in full operation. Delays due to repeated demands of the same memory box are expected to be quite infrequent.

There are conflicts due to fetches and stores, independent of memory access. These are related to the machine measures at preserving the logical integrity of a program sequence and will be discussed elsewhere.

Overlapping of Decoding

The I-box decoding, address indexing, and operand request continues as long as instructions are available and as long as LA levels are available for loading. For a sequence of floating point instructions the decoding rate is one instruction per 1.2 μs. With few exceptions, this is faster than the execution rate. Thus the decoding time for average floating point sequences is completely overlapped by concurrent E-box execution time. In other words, floating point instructions are E-box limited. Address indexing has no measurable effect on the timing of floating point instructions.

VFL instruction decoding is much more complicated. The minimum time is 3.6 μs which includes the loading of two LA levels. Each addition level requires an additional 0.6 μs, and each indexing operation requires 0.6 μs. Of course, there may be further slow down in the decoding process if the LA levels are not available for loading, or if the second half of the instruction is not available when needed.

The decoding time of I-box instructions is 0.6 μs. Since these instructions are executed in the I-box, the decoding time has been included in the execution times. The processing time of I-box instructions can be largely overlapped by concurrent E-box action, since their E-box time is only 1.2 μs.

Whenever a branch instruction is successfully executed, the prefetched instructions in 1Y, 2Y must be replaced by new instructions. The latter have to be checked prior to use, and the decoding of the next instruction will be delayed. This delay has been taken into account in the timing given. Again, concurrent E-box action can overlap much of this.

Look-Ahead Levels

In order to use the look-ahead (LA) levels efficiently, they must not be allowed to be empty over extended lengths of time, nor should they be crowded with data with little information content.

The following instructions empty the LA completely: *T, TI, SWAP, SWAP I, RNX, BD.*

All I-box instructions load LA levels for index register recovery or for indicator register updating. These levels are not useful to the E-box and have the effect of reducing the number of LA levels.

VFL data with word boundary crossover represent inefficient use of LA levels.

Frequent demands on LAAR (such as in consecutive STORE instructions)and prolonged decoding delays in the I-box in general often lead to a half-empty look-ahead.

When the number of effective LA levels are reduced, the E-box may become idle. The I-box processing time can then no longer be absorbed.

The I-Checker

The I-Checker is shared between the I-box and LA. It processes information in 0.6 μs and is used for the following functions:

1. Instruction word error check with concurrent ECC to I–parity conversion.
2. I-box data error check with concurrent ECC to I-parity conversion.
3. Passage of data and *VFL* operation code from I-box to LA, with concurrent I-parity to LA-parity conversion.
4. E-box fetch operand error check and check code conversion.
5. Store operand error check and check code conversion.
6. As part of data path between LA and I-box during, say, branch recovery.
7. As part of I-box internal data path.
8. As part of LA internal data path.

The great majority of the demands on the I-Checker is due to 1, 2, 3, 4, and 5 above. It is therefore conceivable that I-Checker conflicts may occur. The situation has not been completely studied (because of the difficulty in subjecting the variables to program control), but it does not seem to have had much effect on floating point operations. When an I-Checker conflict occurs, the processing of one piece of data may have to be delayed by 0.3 to 0.6 μs.

The LAAR, Stores and Internal Operand Fetches

All "to-memory" operations involving the main memory from the 7030 CPU are prepared by the I-box and accomplished by the look-ahead. I/O stores are, however, performed directly between the MBCU and exchange units.

The look-ahead address register (LAAR) is created for the purpose of containing the store address. LA levels are made available for the store operand.

In the case of I-box store type instructions (*SX, SV, SC, SR, SVA, R, RXZ, T, SWAP,* etc.) the store operands are already available during I-box processing time and can be converted to ECC check bits during shipment to LA, simplifying the store action in LA. In practically all other cases the store operand will not be available during I-box processing. In any case, the LAAR will remain "busy" from the time of look-ahead loading until the proper operand is available, is fetched, checked, and accepted by the MBCU.

In an instruction, the address may refer to the main memory, index register storage, or an internal register. The I-box, upon decoding an instruction whose fetch operand is an internal register, uses the LAAR to store the internal operand address. This is because the needed operand is not available during the decoding stage, and the mechanism for internal operand fetch already exists for the handling of store instructions. Unlike a standard store, no memory bus request is needed, and the LAAR is freed sooner.

(By an internal operand address is meant address 3, or an address between 5 and 12 inclusive. $Z, $1A, $MB, and $RM, $FT and $TR are *bona fide* main memory locations. $IT , $TC and $0 through $15 are in index storage.)

Since there is only one LAAR, if the latter is busy whenever the I-box decodes an instruction requiring LAAR, a wait must occur until the LAAR is free for reloading. Therefore, LAAR-requiring instructions should preferably be reasonably far apart, ideally with three or more time consuming instructions in between to ensure smooth I-box decoding. Measurements on consecutive floating-point-to-memory operations (*ST, MT, LFT,* etc.) do not therefore yield realistic timing information. On the other hand, the placement of such instructions is usually beyond the programmers control.

A store into index storage is a time-consuming operation. Whenever such an instruction is decoded, since the next instruction may make use of the new index contents, the I-box does no further decoding until the new index contents arrives.

Store Close to Fetch

Whenever I-box requests a memory fetch, a comparison is made with the contents of LAAR to avoid logical conflicts. In all cases, logical conflicts will not produce wrong results, although some delays are to be expected.

One such conflict is a store into a location corresponding to a prefetched instruction, such as in the sequence:

```
A     ST (u),   A  +. 0.32
      * + (n),  1000. ($5)                    (1)
```

The second instruction, having been previously fetched into 1 Y or 2 Y, clearly cannot have the correct information until the store is complete. When such a store is decoded, I-box invalidates the prefetched instructions and waits until the correct information is available.

Another conflict is a store into a nearby I-box operand address:

```
ST (u),     1003. ($12)
V +, $15,   1003. ($12)                                    (2)
```

The execution of the second instruction is delayed until the correct operand arrives.

A third such conflict is a store into a nearby E-box operand address:

```
ST (n),     1007.
* (n),      1007.                                          (3)
```

The I-box comparison with the LAAR shows that the operand will be in the LA prior to its arrival at the memory. A forwarding mechanism is activated to make the store operand available to the fetch level. Further LA loading in the I-box is delayed until the forwarding is completed.

Forwarding is available also for consecutive fetch of the same memory operand. Whenever a fetch-type instruction is decoded, the fetch address is gated into LAAR unless the latter is busy. This act does not make LAAR busy but can allow the forwarding on successive fetches.

I-box Instructions

The instructions executed by the I-box can largely be absorbed in an E-box limited environment. The following points, however, are to be noted:

1. Some I-box instructions require the emptying of the LA. Concurrent E-box operations would not be possible.
2. Each I-box instruction results in one (sometimes more) level of LA being loaded. This is done to enable the convenient updating of certain indicators and to allow interruption action (if needed) to take place in step with other interruptions. The loaded level may be an index recovery level containing the old

contents of an index register to enable the restoration in case of unexpected interruptions. It takes 0.9 to 1.2 μs to process an LA level corresponding to I-box instructions. This time cannot be overlapped by concurrent E-box action.

3. The E-box overlapping cannot be effective unless I-box instructions are well dispersed.
4. I-box operand fetches are made when the instruction is decoded. There is no look-ahead buffering to reduce the effective fetch time. I-box immediate instructions, not requiring memory access, are therefore faster than the "direct index arithmetic instructions".
5. Some I-box instructions require more than one memory reference.
6. All successful branch instructions require the fetch of new instructions. The instructions previously buffered into 1Y and 2Y are invalidated.
7. The treatment of some conditional branches requires extensive E-box action and will be discussed in the next section. The following branches, however, involve conditions known to the I-box and are executed entirely within the I-box:
   CB and variants
   Bind for *XF, XVLZ, XVZ, XVGZ, XCZ, XL, XE, XH.*

Recovery Action

The I-box is usually ahead by several instructions during a machine run of the E-box. There are certain situations, however, which may force the I-box to refer to the same instruction as the E-box:

1. "Wrong branch" recovery, the I-box having made a wrong assumption about the path to be chosen on conditional branch.
2. Interruption.

These cases may require the I-box to move backwards in time to be in step with the E-box. All changes in the index registers and/or index indicators due to the advance processing by the I-box must now be undone.

For conditional branches where the condition is unknown to the I-box, the latter assumes the branch to be unsuccessful (to avoid unnecessary invalidation of 1 Y, 2 Y contents) and processes ahead. Steps are taken, however, to perform the actual test and to facilitate the alternative path to be taken. Altogether four LA levels are used for each such branch instruction: three levels not unlike those for a *VFL connect-to-memory* instruction plus a branch recovery level. It is noted that there is always a store type level, whether the programmer specifies a change of the tested bit or not. For *Bind*, two of these four levels involve the LAAR.

When the I-box guess proved correct, no particular action is taken aside from setting of the tested bit. If, however, the guess proved incorrect by E-box arithmetic, the I-box has already processed ahead, and recovery action has to be taken to ensure the logical integrity of the program. The re-setting of the I-box to the previous state of ten requires shipment of index register information back to the I-box. The correct instruction counter value is also shipped back to the I-box.

The interruption action is quite similar to branch recovery except that in addition a new instruction has to be fetched and executed.

4
Computational Performance

MATRIX INVERSION - GERB2

Problem

The inversion of Hilbert matrix segments of increasing size, using Jordan's elimination method. In each case the determinant of the matrix is evaluated as a by-product.

Program

See Volume 2.

Timing

10 x 10 matrix	0.02 seconds
20 x 20 matrix	0.14 seconds
30 x 30 matrix	0.43 seconds
40 x 40 matrix	0.99 seconds
50 x 50 matrix	1.89 seconds
60 x 60 matrix	3.23 seconds
70 x 70 matrix	5.10 seconds
80 x 80 matrix	7.55 seconds
90 x 90 matrix	10.69 seconds
100 x 100 matrix	14.61 seconds

Comparison With Other Machines

The 96K memory allows the convenient inversion of matrices up to 300 x 300 without using drums or tapes. The extra word length is also an asset in inversion. The present program uses the relatively slow *+ (*multiply and add*) instruction to obtain 96-bit intermediate fraction accuracy.

Hilbert matrices are extremely ill-conditioned, and no claim is made about the accuracy of the 100 x 100 inversion result. For an ordinary matrix of size beyond 40 x 40, it is probably fair to say 27 fraction bits (as in the 7090) would not be adequate. Double precision cost is a 6-fold decrease in speed on the 7090. The 7030 with 48 fraction bits is adequate for a much larger range.

Additional Remarks

There is a faster matrix inversion program which gives the following results:

50 x 50 matrix	1.1 seconds
100 x 100 matrix	10 seconds
150 x 150 matrix	31 seconds
200 x 200 matrix	79 seconds
250 x 250 matrix	144 seconds
300 x 300 matrix	250 seconds

A 128 x 128 linear equation program with two sets of unknowns requires 8.6 seconds.

There is also a double precision version of GERB2, called GERB3, with the following speeds:

10 x 10 matrix	0.05 seconds
20 x 20 matrix	0.41 seconds
30 x 30 matrix	1.36 seconds
40 x 40 matrix	3.19 seconds
50 x 50 matrix	6.21 seconds

It is seen that the double precision computation time is roughly triple that of single precision. 7030 double precision is almost equivalent to quadruple precision on the 7090 in terms of the number of fraction bits.

MATRIX MULTIPLICATION - MXM16

Problem

Multiplication of two nxn matrices.

Program

Listing (including test program) are included in Volume 2.

Assume n = 17 k+m. A "major" inner loop is traversed k times, then a short inner loop is traversed m times, for each vector multiplication.

The matrix elements used for the test are all equal to normalized floating point 1.0 and are placed in the upper memory (32768.0 and beyond).

Timing

25 x 25 matrices	0.20 seconds
50 x 50 matrices	1.35 seconds
75 x 75 matrices	4.74 seconds
100 x 100 matrices	10.71 seconds
125 x 125 matrices	21.44 seconds

Additional Remarks

A simple version (taken directly from the 7030 programming example book) requires 15.5 seconds for 100 x 100 matrices. The timing cost was traceable to the use of the more accurate but relatively slow LFT; *+ sequence and the fact that two adjacent I-box instructions are executed for each traversal of the inner loop.

In the present program the ratio of I-box instructions to arithmetic instruction is greatly reduced, but no other attempt has been made to speed up the program.

PRIME NUMBER GENERATION PROGRAM PRIMC

Problem

To generate prime numbers by the sieve of Eratosthenes, using VFL arithmetic and automatic program interruption.

In a very large memory segment (octal 17740.40 through octal 272777.63), consecutive bits represent consecutive odd integers. The bit at distance d from the beginning of the string thus represents the odd number 2d + 1. In the beginning all bits in the string are set to 1's.

A working prime P is represented by a 1 bit to the right of the previous working prime. In the beginning the first working prime is 3, occupying the second bit of the bit sequence.

The bits representing the number $p^2 + np$, n = integer ≥ 0, are systematically made zero by what appears to be an infinite loop, starting from the case n = 0. When the prescribed upper memory boundary is exceeded, an interruption causes exit from the loop. The next working prime is then found, and the process is repeated, unless an end condition is encountered. The end condition is met when, for a working prime p, the bit corresponding to p^2 lies beyond the upper boundary. the non-zero bits remaining in the interval represent prime numbers if the first bit is reinterpreted to represent the even prime number 2.

Program (See Volume 2)

It is to be noted that 2/3 of the program consists of an interruption table.

Timing – 106.7 Seconds

The largest prime is $(52,307,665)_8$, or about 11 million. A slight change in the inner loop (replacing the progressive indexing by ordinary indexing and adding a V + I instruction) leads to 102.7 seconds.

Comparison With Other Machines.

The 7030 machine can process bits very conveniently. Each bit zeroing on the 7030 takes only one instruction. The corresponding operation on the 7090 or similar machines requires very careful programming and relatively long computation time (about 48 μs), even if only $2^5 = 32$ bits are used per word to avoid a divide instruction. Also, other machines do not have as many bits in the memory. For this problem, the memory capacity of the 7030 is almost six times the 7090.

A MONTE CARLO PROGRAM

Problem

The physics and method of solution of the problem is described by Davis, Journal of Applied Physics, 1960. The original problem was coded in Livermore on the 709 and 7090.

Briefly, the problem is the passage of particles through a right-angle bent pipe of circular cross-section at such a low density that only wall collisions are important. It is assumed that the angle of rebound is random and uncorrelated with the angle of incidence. The two ends of the pipe are each divided into 4 areas, and statistics are accumulated to determine the distribution of exit areas as a function of entry areas.

The calculation on the 7090 differs in one respect to that described in the paper. The calculation of random input angles and rebound angles follows the "cosine law". The choice of these angles in the 7030 code differs from the 709 in method but not in result.

A point is chosen from a uniform distribution in the rectangular parallelopiped (-0.7, 0.7) x (-0.7, 0.7) x (0.0, 1.0). Unless

$$(x^2 + y^2 + z^2)^2 < z,$$

the point is discarded and a new one is tried. If the inequality is satisfied the vector is used as a velocity vecotr relative to the X-Y wall.

Program

See Volume 2.

Timing

10,000 particles were run and the measured time was 33 seconds. This is to be contrasted with a known 709 run of 5000 particles in 10 minutes.

For the machine run 20 cards had to be loaded and 36 numbers were printed on line.

WEATHER FORECASTING STUDY

The following weather forecasting study summarizes the results, to date, of timing experiments performed on STRETCH.

Inner Loop

This program steps in time one floating-point meteorological variable at one ground point (i, j) of grid, and four floating-point meteorological variables at each vertical k-level above the ground point. The equations are non-linear integro-differential.

1. First form (AO, see below), lower memory: 6.4 ms.
2. Pseudo j-level (see below), lower memory: .7 ms.
3. AO, upper memory: 6.13 ms.
4. AO, lower (instructions) & upper (data) memory: 5.98 ms.

5. Second form (A1, see below), lower (instructions) and upper (data) memory: 5.57 ms.
6. Al Modified (see below), lower (instructions) and upper (data) memory: 7.045 ms.
7. Third form (A2, see below), lower (instructions) and upper (data) memory: 7.0135 ms.

The above times for runs 1, 3, 4, 5 are for one pseudo j-level, two i-levels, and three k-levels per i-level. The time for run No. 2 is for one pseudo-j-level only, and the times for runs Nos. 6 and 7 are for greatly reduced pseudo-j-level, one i-level, and nine k-levels.

AO: Coding prior to any changes for timing improvement.

A1: Same as AO except for coding changes to improve timing with respect to accumulate multiply, division, add to memory, separation of indexing instructions, loop entry, CNOP, one-word transmits, and associated minor changes.

A2: Same as A1 Modified except for additional coding changes to improve timing with respect to spacing of store instructions, and more-than-one-word transmits.

Summary

With respect to run No. 1, the above times provide the following approximate improvements: Run No. 3 - 4.2 percent; run No. 4 - 6.6 percent; run No. 5 - 13 percent. With respect to run No. 6, run No. 7 provides an approximate improvement of .4 percent. This last suggests that this program is insensitive to the space of the store instruction and the multi-word transmit.

Grid Parameters

The execution of this program constitutes a neglible part of execution time required for the Weather program. Also, the program performs a very specialized function, arising from the particular grid-geometry and interpolation chosen for the Weather Project. Consequently, no coding changes in grid parameters were made for purposes of timing improvement or further study. Grid parameters computes, as a function of the grid size (parameter N), certain grid measurements and interpolation data (VFL and floating-point).
Using N = 3, lower memory: 7.07 ms

Elementary Function Test

$\ln(e^x)$ is compared against x, then a polynomial evaluation of $e^{\ln x}$ is compared against x. The evaluations are based on 8-decimal accuracy subroutines, employing polynomial methods. The program uses lower memory exclusively.

In the following chart, certain similar operations have been grouped together (e.g., / and R/; V+, V-, V+I, V-I; +, M+ under Fl. Pt.). Also, the time given for AO is that for the more favorable memory arrangement (i.e., instructions in lower, data in upper).

TABLE 10. EXECUTION FREQUENCY CHART FOR STRETCH PROGRAMS

Operation		A0	Pseudo j-level	A1	Modified	A2	Grid Param.	Elem. Functions 1st Form	Elem. Functions 2nd Form
Fl. Pt.	LFT	144					39		
	L	257	1	437	633	651	243	7	7
	ST	261	1	465	659	677	202	15	15
	+	250		394	581	581	195	28	28
	*	220		434	637	637	187	26	26
	/	88		42	51	51	42	2	2
	*+	144					39		
	D*							2	
	K						45	2	2
	SHF							2	
	E+						69	4	4
	SRT						4		
	Total	1364	2	1772	2561	2597	1065	88	84
VFL	L	10	8	10	9	9	128	4	2
	ST	2		2	1	1	93	6	2
	+	20	18	20	19	19	20		
	CM	66	66	70	71	71	9		
	*						8		
	LFT						4		
	*+						4		
	-MG						4		
	M+1						37		
	K						64		
	Total	98	92	102	100	100	371	10	4
Indexing	LZ,LV,LC,SV	13	9	13	11	11	6		
	V+	60		60	78	78	117		
	CB	15	1	15	20	20	13		
	KCI	12		12	18	18			
	Total	100	10	100	127	127	136	0	0
Branching	B,BD,BEW	77	5	68	91	91	6	8	10
	BIND	18		18	27	27	274	3	3
	BB	2		2	1	1	1		
	Total	97	5	88	119	119	280	11	13
Miscell.	Z,TI	52		28	38	29	31	1	1
	Total								
Grand Total		1711	109	2090	2945	2972	1883	114	106
Execution Time (ms)		5.98	.7	5.57	7.045	7.0135	7.07	.38	.309
Aver. time (μs)/opn		3.5	6.4	2.7	2.4	2.36	3.8	3.3	2.9

5
Reliability and Serviceability

The capability of a large computer complex to satisfy operational requirements must extend beyond the scope of engineering design and programming flexibility. Maintainability, achieved through the application of reliability and serviceability principles during the design stage and the planning of maintenance procedures, is a key to maximum system availability. Among the special features of STRETCH are extensive error-checking and error-correcting circuitry, test panels, marginal (bias) checking, and automatic scanning and recording. Maintenance aids include diagnostic programs for testing and maintaining the equipment and system, selection and training of field maintenance personnel, unique distribution and function of spare parts depots, comprehensive maintenance manuals, and applied programming support.

PERFORMANCE CHARACTERISTICS

Based on statistical analyses of this class of large scale computer systems and correlation of the analyses with an operating 7030 system, it is estimated that the typical 7030 Data Processing System will have the following operating characteristics within 19 months after installation of the first system and provided the system on which they are measured has been installed for at least 6 months. These estimates are based upon the best available information on component failure rates, and best estimates of the impact of the automatic error correction feature.

7030 System Availability for Acceptance Purposes

Availability for acceptance purposes is indirectly based on a specific allotment of time for preventive maintenance during each 24 hour period

of operation. It is defined as the ratio of productive customer hours to scheduled customer hours (defined as that portion of a regularly scheduled shift assigned to customer operation.)

$$\text{Availability} = \frac{\text{(Scheduled Customer Hours) - (Unscheduled Maintenance Hours)}}{\text{Scheduled Customer Hours}}$$

Average Time Between Unscheduled Maintenance Periods

On the average, it is estimated that 6 or more hours of customer operation will normally occur between unscheduled maintenance calls. This suggests one unscheduled call per operating shift.

Duration of Unscheduled Maintenance Calls

The average duration of an unscheduled maintenance call is estimated to be 1.3 hours. Ninety percent of the unscheduled calls can be expected to be less than 2.5 hours in duration.

Scheduled Maintenance

Based on purely technical equipment requirements, the duration of a scheduled maintenance period will be less than four hours. Twice each calendar year, a 24 hour period must be reserved for scheduled maintenance. There should be 16 or more hours of customer operation between scheduled maintenance periods. On a regularly planned basis, a four hour period of scheduled maintenance is required during every 24 hour operating period.

DIAGNOSTIC PROGRAMMING

Automatic diagnostic programming, the most advanced technique devised for rapidly isolating difficult and intermittent system incompatibilities, has been scientifically approached in SAGE I, refined for

other large scale computer systems, and is an integral part of the 7030 Data Processing System. Rapid isolation of malfunctions to a limited area as diagnosed automatically by the computer itself and multiple replacement substantially increase system efficiency.

Good diagnostic programming provides three major advantages:

1. A means of computer analysis which maximizes the success of customer machine program performance.
2. A means of isolating failures faster to maximize customer machine time availability.
3. A means for establishing an intimate man-to-machine relationship.

Diagnostic programs are written with reference to the machine logic, as well as with reference to machine specifications. Thus these programs ensure that all hardware is tested and that the machine will operate properly with worst-case patterns and timing relationships.

Diagnostic testing under program control proceeds with the building block technique where possible. This is a technique which starts by testing the smallest amount of hardware, and then gradually adds subsequent tests involving the minimum increment of additional circuitry possible. In this fashion a failure at any one point is isolated to those logical blocks added in the failing test. Testing continues until all logical blocks are covered. At this point, worst patterns and timing relationships are introduced to test for electronic interactions between areas of equipment.

Diagnostic programs are in use to test and maintain the present 7030 systems. In addition, a SEVA (Systems Evaluation) program will

provide a comprehensive over-all test of the 7030 System under conditions similar to those expected for an operational program. A brief description of these diagnostic programs is presented below followed by a description of the SEVA program.

- Diagnostic Control Program – This is an executive program to control the running of all diagnostic programs providing standard options and common utility routines.
- I-Box – Tests all the controls and transfer paths necessary for the execution of instructions in the instruction unit.
- SAU – Tests the transfer paths, arithmetic elements, and controls of the serial arithmetic unit.
- PAU – Tests the transfer paths, arithmetic elements, and controls in the parallel arithmetic unit.
- Look Ahead – Tests the transfer paths and controls of the look-ahead unit.
- Memory – Tests the memories and memory bus, including a worst-patterns test.
- CPU Scan and Miscellaneous – Tests the central processor unit (CPU) scan circuitry, clocks, and boundary registers.
- BX 0 – This program provides for manually testing basic functions of the exchange without using the central processor.
- BX 1 – Tests those portions of the exchange which can be tested without using any I/O devices.
- BX 2 – Tests the common circuitry in the exchange using available devices which can be selected by the operator.
- BX 3 – Tests the Exchange under worst conditions of simultaneous operation.

- Tapes – Tests the tape adapters and tape drives, including the control, information transfer, and timing circuitry.
- Printer – Tests the printer control unit and the chain printer. Carefully selected patterns will be printed for the operator to verify results.
- Console – Tests the console control unit and the operator's console. The patterns typed on the typewriter are verified by the operator, while the operator's inputs are verified by the computer.
- Reader, Punch – This program provides a punch to reader loop for testing the reader, reader control unit, punch, and punch control unit.
- High-Speed Exchange – This program utilizes the disk file in an elementary fashion to test the disk synchronizer.
- Disk – Tests the disk and includes worst patterns and timing tests.
- I/O Scan – Tests the circuitry associated with the exchange scan and the disk scan.

SEVA Program

The 7030 SEVA (Systems Evaluation) program is designed primarily to test the interrupt, asynchronous capabilities, and capacity of the 7030 system in a manner similar to that in which a customer will operate the system. SEVA is made up of a number of analytical and mathematical checking routines and is designed so that the I/O routines will cycle independently and concurrently of each other and of the central processor unit routines. Because of the asynchronous operation of the 7030 system, phase shifts will occur, producing an almost infinite variety of timing conditions within the system. During operation,

progress of the SEVA program, required operation action, and inconsistencies are printed out by the printer for evaluation and analysis purposes.

The SEVA program will contain three general types of routines: control routines, central processor unit routines, and input-output routines. Figure 1 illustrates the inter-relationship between the various routines. A brief description of these routines follows:

- Initiator – This routine establishes initial conditions, as required, for all of the other routines.
- I/O Initiator – This routine operates with the initiator routine to start from one to eight tape routines and one disk routine and is used only when first starting or restarting the over-all SEVA program.
- Central Processor Unit Routine Sequencer – This routine establishes the order in which the other central processor unit and memory routines will be run. After each pass through all of the central processor unit routines, the order is permutated so that the central processor unit routines will not be run in the same order on each successive pass.
- Central Processor Unit Routines – Central processor unit routines will calculate mathematical problems such as binomial expansions and solutions of quadratic equations. These routines will contain four sections to accommodate both variable field length and floating point calculation, using different techniques so that the results can be compared. Provision is made within the SEVA program to add other routines (indicated as variable routines in figure 1) as desired.

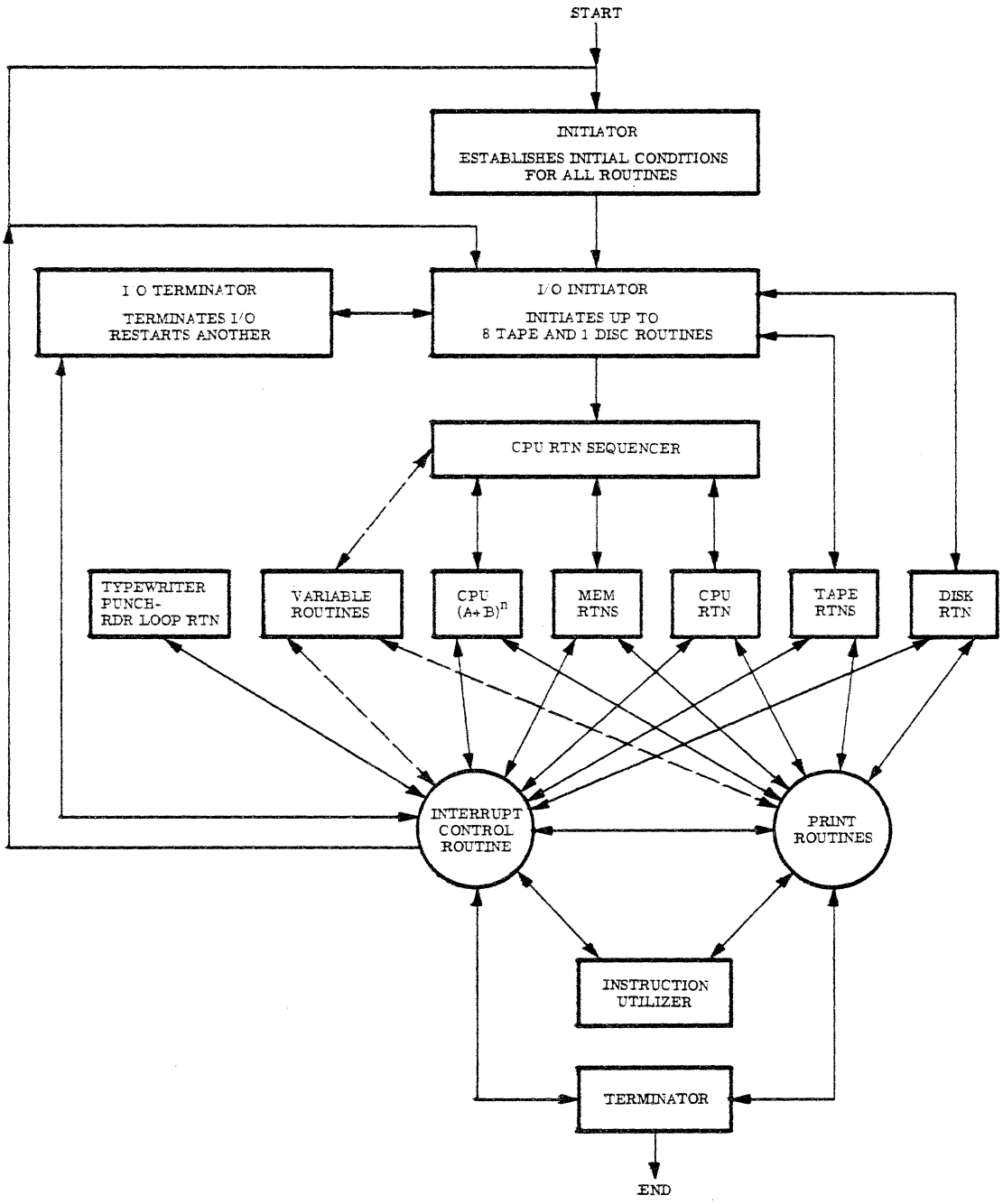

Figure 1. 7030 SEVA Program

- Memory Routines – Memory routines will perform instruction fetch (IF) and data store (DS) functions in all available core storage units. Independently, the result will be generated in a fixed memory area for comparison to the other result generated by this routine.
- Tape Routines – There are eight identical tape routines one for each tape adaptor, and they will exercise every possible operation.
- Disk Routine – This routine is similar to the tape routines in that it will exercise all possible disk operations.
- Typewriter, Punch, Reader Loop Routine – With this routine, data entered through the typewriter will be typed and punched. Punched data from the reader will be typed again to verify results.
- Input-Output Terminator – When a tape adaptor or disk has completed a routine, the input-output terminator routine will temporarily delete the equipment from the over-all SEVA and start another tape adaptor or disk in its place. When available, the disk and eight tapes will be running simultaneously with the central processor unit routine.
- Interrupt Control – This routine will handle all interrupts. If two error interrupts occur within a 30-second period, the program will be dumped. For single error interrupts within a 30-second period, the error will be logged and the operation will continue.

Appendix A

7030 CPU CONFIGURATION

Main Memory

The main memory, sometimes called extended memory, is composed of six boxes of 16,384 extended words each (for the Los Alamos configuration). Each extended word contains 64 information bits plus 8 error-check bits.

The main memory is controlled by the memory bus control unit (MBCU) which, in addition to initiating all accesses to main memory, also monitors the submitted addresses for the Address Invalid (AD) condition. The MBCU unit has direct contact with the following units: instruction unit, Look-Ahead, memory boxes, exchange, and disk exchange.

Instruction Unit

The instruction unit (I-box) contains the instruction counter (IC), the 16 index registers ($0-$15), the time clock ($TC) and interval timer ($IT), the "originals" of the index condition indicators ($XF, $XVLZ, $XVZ, $XLGZ, $XCZ, $XL, $XE, $XH), and many registers and circuits needed for efficient decoding and execution of instructions. The I-box executes the following activities:

- Generates all instruction-fetch requests on the basis of IC contents.
- Develops effective addresses by adding the pertinent index value to the numerical address.
- Generates E-box operand requests for look-ahead.
- Partially decodes E-box instructions and converts the latter into information suitable for look-ahead processing. This information is then loaded into look-ahead.

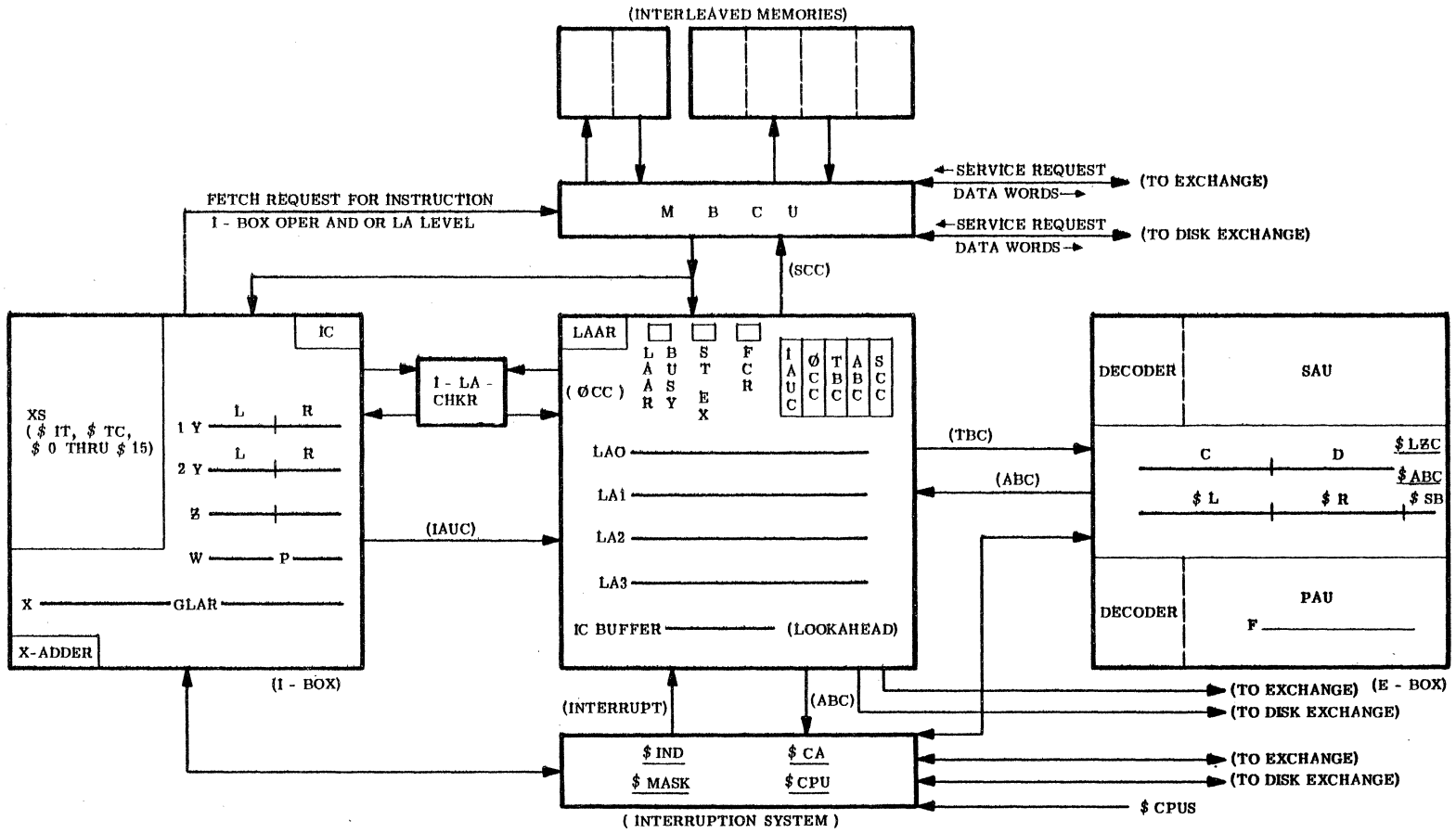

Figure A-1. The 7030 Data Processing Unit

- Decodes and executes all index arithmetic instructions as well as the following: *Z, R, RCZ, EX, EXIC, T, SWAP,* except that Stores are performed with the help of look-ahead. If non-I-box operands are required, they will be fetched from the MBCU or the look-ahead.
- Decodes and executes the following branch instructions: *B, BE, BD, BR, BEW, CB, CBR,* and *Bind* for non-index conditions require the assistance of the look-ahead and the E-box.
- Submits indicator conditions for the following indicators to look-ahead for updating of the indicator register: $IJ,$OP, $AD (from MBCU), $DS, $DF, $IF + index indicators. These indicators plus a conditional machine check may lead to interruption during the updating. Other I-box-generated indicators are gated directly to the indicator register.
- Updates $TC and $IT every 1/1024 second.

Look-Ahead (LA)

Look-ahead contains four buffer levels, an address register (LAAR) and five counters, IAUC (I-box), OCC (operand check), TBC (transfer bus into E-box), ABC (arithmetic bus and interruption system), and SCC (store check). Look-ahead functions under the following conditions:

- When IAUC refers to a level, that level may accept I-box loading, although the required operand may come later from MBCU.
- During OCC time, the operand from MBCU may be checked for error.
- During TBC time, the level may be shipped into E-box.
- During ABC time, the interruption system is updated and signal is given to E-box for execution of the instruction just loaded.

- During SCC time, the store operand (if any) is checked and sent to MBCU on the basis of the contents of LAAR.
- Provides interlocks, plus close contact with the interruption system, to ensure smooth, autonomous, and error-free operations of the various units in the computer.

Interrupt System

The interrupt system contains the indicator register ($IND), the mask register ($MASK), and $CA and $CPU. It has direct connections with I-box, LA, E-box, and the exchange units to receive updated indicator information.

- Interruption occurs if:
  a. System is enabled
  b. A masked indicator bit is a 1.
- Interruption sequence:
  a. The left-most masked indicator bit position (for instance, 0.K) is noted.
  b. I-box is house-cleaned except the index storage.
  c. LA house-cleaning is performed. Recovery information is shipped back to the I-box. This includes all index register recovery information and the interrupted IC value.
  d. Contents of $IA is fetched and added to K.0.
  e. Instruction beginning at the address C($IA) + K.0 is fetched from MBCU without disturbing $IF indicator.
  f. The "free" instruction is performed with all masked interruption conditions enforced. This instruction may or may not alter IC in I-box.
  g. I-box fetches new instructions on the basis of IC.
  h. Resumption of normal operations.

E-Box (Arithmetic Unit)

The E-box contains a parallel arithmetic unit (PAU) for floating-point fraction operations, as well as all executed *, /, *+, and binary-decimal conversions. It contains a serial arithmetic unit (SAU) for variable field length operations (except *, /, *+ and conversions), as well as for floating point exponent arithmetic.

The E-box:

- Receives instructions and operands from LA for decoding and execution.
- Submits store operands to LA.
- Submits arithmetic indicator bits to the interrupt system.

The E-box also contains the following registers:

1. Accumulator ($L, $R) and sign byte register ($SB).
2. Buffer registers C, D.
3. PAU buffer register F.
4. Left zero counter ($LZC) and all ones counter ($AOC).

Exchange

The exchange contains up to 32 channels (32-63) for simultaneous I/O processing. Through adaptors each channel can be connected to eight tape units or with one non-tape I/O unit. Each channel is represented by one control word and two data words in the exchange memory. The channels communicate with the change memory thru a multiplexer.

The exchange unit contains a main memory address register (MMAR) and a buffer register to communicate with the MBCU. It also contains an interruption address for the channel which has created an interrupt condition, as well as triggers (EOP, UK, EK, EE, and CS) to indicate the reason for interrupt. These triggers and IAR contents are

set until the interruption system accepts the conditions. When IAR is busy (Interrupt Wait trigger on), other channels cannot use it to cause other interrupts.

The exchange unit:

- Accepts I/O instructions from LA (2 levels per instruction).
- Fetches and stores control words and data words directly from MBCU, subject to $AD restrictions but not $DF and $DS since these are performed by the I-Box.
- Communicates with I/O units (thru adaptors and multiplexer) in 8-bit bytes.
- Has its own clocking circuit (1.0 us cycles divided into 10 equal pulses) and ECC check-bit generator-comparer. Maximum word rate is 1 extended word/10 μs for the entire exchange.

The disk exchange contains 32 channels (0-31), only one of which can be in operation at any given time. It contains enough memory for 1 control word and 1 data word, and each channel can be attached to one disk unit. Disk word rate is 1 extended word/8 μs. No direct chaining of the I/O control words is permitted. In addition, the copy-control-word operation cannot be performed when reading or writing. Otherwise the disk exchange functions in much the same way as the exchange.

LIST OF IMPORTANT REGISTERS IN THE 7030

I-Box

Table A-1 indicates important I-box registers in the 7030.

TABLE A-1. I-BOX REGISTERS

Code	Name	Bit Information	Remarks
1Y*	Instruction buffer	64 + check bits	Even-addressed full words
2Y*	Instruction buffer	64 + check bits	Odd-addressed full words
Z	Instruction preparation and execution register	64 + check bits	
XS	Index register	17 words, each with 64 bits + check bits	Contains location 1.0, 16.0-31.0
X	Index data register	64 + check bits	Buffer register for XS
X-Adder	Index Adder	32 + check bits	Capable of 24-bit additions
W	Work register	18 + check bits	Serves miscellaneous functions in I-box: *LVS* address decoding; second operand address in VFL; refill and interruption address; count for *T* and *SWAP*; *LVS* address decoding.
IC	Instruction counter	19 + check bit	
GLAR	Left zeros counter for *LVS* instruction execution		Geometric-load address counter
Originals of: $XF, $XVLZ, $XVZ, $XVGZ, $XCZ, $XL, $XE, $XH			

*Both 1Y and 2Y may be used as I-box operand buffer

I-Checker (shared between I-box and Look-Ahead)

Look-Ahead (LA)

LA0, LA1, LA2, LA3 — Look-Ahead buffer levels, each has:

- Op code field (10 bits + check)
- Operand field (64 bits + checks)
- Indicator bit field (15 bits)
- Instruction counter field (19 bits + checks)

plus these bits for each buffer level:

NOOP	no-op bit
WBC	word boundary crossover bit
LAOP	LA op code bit
IC	Instruction counter bit
INT	Internal fetch bit
LC	Level checked bit
LF	Level filled bit
FF	"Forward from" bit
DISC	Disconnect bit

LAAR Look-Ahead address register (18 + check)

LAAR-Busy bit

Store-executed bit

Forward-cycle-required bit

IC buffer (19 + checks)

Counters: (4 bit rings)

- IAUC (Instruction-arithmetic unit counter) For LA loading from I-box
- OCC (Operand check counter). For check of opnd arrived from MBCU
- TBC (Transfer bus counter). For loading of E-box.
- ABC (Arithmetic bus counter). For interrupt system updating, internal opnd fetch.
- SCC (Store check counter). For storing into main memory

Interruption System

$IND - Indicator register.

$MASK - Mask register.

$CPUS - Other CPU. (Not available for LASL and BuShips systems)

$CA - Channel address register.

Left zeros counter to handle interrupts.

E-box

$L, $R - Accumulator.

$SB - Sign byte register.

C, D - Operand buffer register (each 64 + check bits).

$LZC - Left zeros counter.

$AOC - All ones counter.

SAU - Serial arithmetic unit

SAU - decoder

SAU - arithmetic - logical unit

PAU - Parallel arithmetic unit

PAU - decoder

PAU - arithmetic - logical unit:

PAU - adder

PAU - multiplier

(PAU) - F-register

Exchange

Exchange storage (EM)

EMAR. Exchange memory address register. (7 bits + check)

Word register (communicates with EM) (76 bits)

MMAR. Main memory address register (18 + check bits) for dealing with MBCU Buffer register (72 bits) to handle traffic with MBCU

Interrupt address register (7 + check) to contain address of interrupting channel.

Interrupt triggers for 5 exchange interrupt conditions.

Interrupt wait bit.

Multiplexer for dealing with individual channels.

ECC generator and comparing circuits.

TABLE A-2. SUMMARY OF 7030 INSTRUCTION EXECUTION*

Operation Code	I-box opnd fetch	I-box execution	opnd fetch for LA	LA levels	LAAR needed?	LAAR not needed but used if not bury	SAU	PAU	Comments
LX, LV, LC, LR, V+, V+C V+CR(C≠0), *V ± ICR* (C=0) *R* (index), *RC≠* (index, C = 0)	yes	yes	no	IX Rec.	no	no	no	no	*V+CR* (c= 0) involves 2 I-box fetches.
LVI, LVNI, LCI, LRI, C ± I V± I, V± IC, V±ICR (C ≠ 0) *RC≠* (C ≠ 0) (index)	no	yes	no	IX Rec.	no	no	no	no	
KV, KC	yes	yes	no	NOOP	no	no	no	no	
KVI, KCI	no	yes	no	NOOP	no	no	no	no	
SX, ≠	no	partly	no	store	yes	no	no	no	
SV, SC, SR, SVA	yes	partly	no	store	yes	no	no	no	*SVA* requires extra decoding
R(memory), *RC≠* (memory), (C = 0)	2	partly	no	store	yes	no	no	no	
RC≠ (memory, C ≠ 0)	no	yes	no	NOOP	no	no	no	no	
LVE	N	N	no	IX Rec.	no	no	no	no	extra decoding for each instruction fetch
LVS	no	yes	no	IX Rec.	no	no	no	no	repeated addition of index value fields.
RNX	yes	partly	no	test store NOOP	no yes no	no no no	no	no	LA is pre-cleared by first level.
EX (exclusive of subject instruction) (repeated EX assumed)	N	yes	no	NOOP	no	no	no	no	extra decoding for each instruction fetch.
EXIC (exclusive of subject instruction) (repeated EXIC assumed)	2N	partly	no	N stores	No times	no	no	no	extra decoding for each instruction fetch.
T, SWAP	N	partly	no	test stores NOOP	no Ntimes no	no no no	no	no	LA is pre-cleared by first level. Each N is doubled in *SWAP*
LA level designation INT = internal operand fetch INT STORE = internal opnd store Store = Store IX Rec. = index register recovery (also called psuedo-store) B Rec. = branch recovery LAOP = LA operation NOOP = no op; indicator transfer only									LA level designation (cont'd) Op = operation code level (VFL) opnd = operand level op + opnd = op code plus operand (usually F.P.)

*(Addresses are assumed to refer to Main Memory unless specified)

TABLE A-2. SUMMARY OF 7030 INSTRUCTION EXECUTION (cont'd)

Operation Code	I-box operand fetch	I-box execution	opnd fetch for LA	LA levels	LAAR needed?	LAAR not needed but used if not bury	SAU	PAU	Comments
1Y, 2Y are cleared to receive new instruction for all successful branches									
B, BR, BE, NϕP	no	yes	no	NOOP	no	no	no	no	
BD	no	yes	no	test test	no	no	no	no	LA pre-cleared
BEW	no	partly	no	NOOP NOOP	no	no	no	no	
				2 test levels	no	no			
CB, CBR (no refill)	no	yes	no	IX Rec.	no	no	no	no	
CBR (refill)	yes	yes	no	IX Rec	no	no	no	no	
Bind (XF, XC≠, XVL≠, XV≠, XVGZ, XL, XE, XH)	no	yes	no	noop	no	no	no	no	
Bind (non-index conditions)	no	partly	INT	OP.	no	no	yes	no	
				INT	yes	no			
				INT. stores	no	no			
				B Rec.	no	no			
BB	no	partly	yes	OP	no	no	yes		
				opnd	no	yes			
				store	yes	no			
				B Rec.	no	no			
SIC B, SIC BR, XK BE, SIC BD	yes	partly	no	store	yes	no	no	no	LA pre-cleared in SIC BD by 2 test levels
SIC BEW	yes	partly	no	store	yes	no	no	no	
				2 test levels	no	no			
SIC CB, SIC CBR (no refill) if branch is taken	yes	partly	no	IX Rec.	yes	no	no	no	SIC store level will not exist if branch is not taken
				Store	no	no			
SIC CBR (if refill) if branch is taken	2	partly	no	IX Rec.	yes	no	no	no	SIC store level will not exist if branch is not taken
				store	no	no			
SIC Bind (index conditions) if branch is taken	yes	partly	no	store	yes	no	no	no	store level replaced by noop if branch is not taken
SIC Bind (non-index conditions)	yes	partly	no	op	no	no	yes	no	
				INT	yes	no			
				INT store	no	no			
				B Rec.	no	no			
				store	yes	no			

TABLE A-2. SUMMARY OF 7030 INSTRUCTION EXECUTION (cont'd)

Operation Code	1-box operand fetch	1-box execution	opnd fetch for LA	LA levels	LAAR needed?	LAAR not needed but used if not bury	SAU	PAU	Comments
F. P.									
±, *L, LWF, ± MG* *D±, DL, DLWF, D± MG* *K, KMG, KR, KMGR* **, /, R/* *F±, E ±*	no	no	yes	op + opnd	no	yes	exp.	frac.	
E ± 1	no	no	no	op + opnd	no	no	exp	frac	
SHF	no	no	no	op + count	no	no	no	frac	
ST, SLθ, SRD, SRT	no	no	no	store	yes	no	exp	frac	
M±, M± MG	no	no	yes	op + opnd	no	yes	exp	frac	
				store	yes	no			
**+,*	no	no	2	op + opnd	no	yes	exp	frac	
				C($FT)	no	yes			
LFT, D/	no	no	yes	op + opnd	no	yes	exp	frac	
				store	yes	no			
VFL *+, L, LWF, ± MG* *K, KMG, KR, KMGR* *KE, KF, KFE, KFR.* *C, CT, LCV*	no	no	yes	op	no	no	yes	no	For all VFL operations, if opnd crosses over full word boundary, the no. of LA opnd and/or store levels is doubled Progressive indexing required one more IX box.
				opnd	no	yes			
* (Binary)	no	no	yes	op	no	no	yes	yes	
				opnd	no	yes			
CV, DCV	no	no	no	op	no	no	yes	no	
				INT					
ST, SRD *CM, M± MG, M+1*	no	no	yes	op	no	no	yes	no	
				opnd	no	yes			
				store	yes	no			
**+* (Binary)	no	no	2	op	no	no	yes	yes	
				opnd	no	yes			
				C($FT)	no	yes			
LFT, LTRS, LTRCV **(Dec.), / (Dec.), *+(Dec.)*	no	no	yes	op	no	no	yes	no	
				opnd	no	yes			
				store	yes	no			
I/O	no	no	no	op	no	no	no	no	LA communicates with exchange directly
				LAθP	no	no			