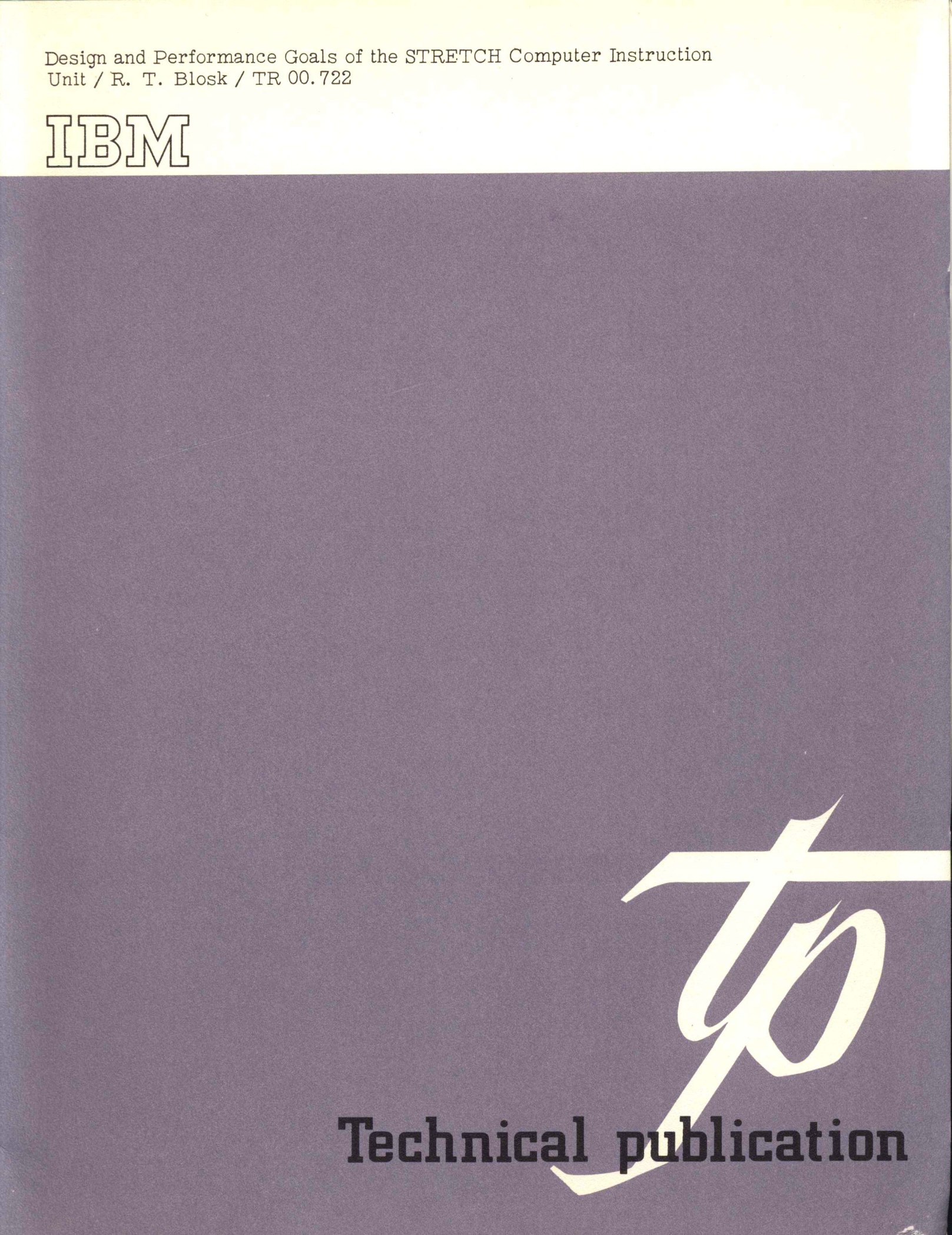
Design and Performance Goals of the STRETCH Computer Instruction Unit / R. T. Blosk / TR 00.722

IBM



May 18, 1960 TR 00.722

DESIGN AND PERFORMANCE GOALS OF THE STRETCH COMPUTER INSTRUCTION UNIT

R. T. Blosk

ABSTRACT

The Instruction unit is a large, complex, high speed computer unit, designed and built to provide the major functional ability and control for the STRETCH computer. The size is determined by the instruction buffering, the fast access index registers, the extensive overlapping and simultaneity of operations, and the innumerable combinations and variations of instructions that have to be controlled. The performance is achieved by a synchronous clock-controlled network of variable sequence execution control triggers. This performance is shown for a few typical and particularly important operations.

Product Development Laboratory, Data Systems Division
International Business Machines Corporation, Poughkeepsie, New York

TABLE OF CONTENTS

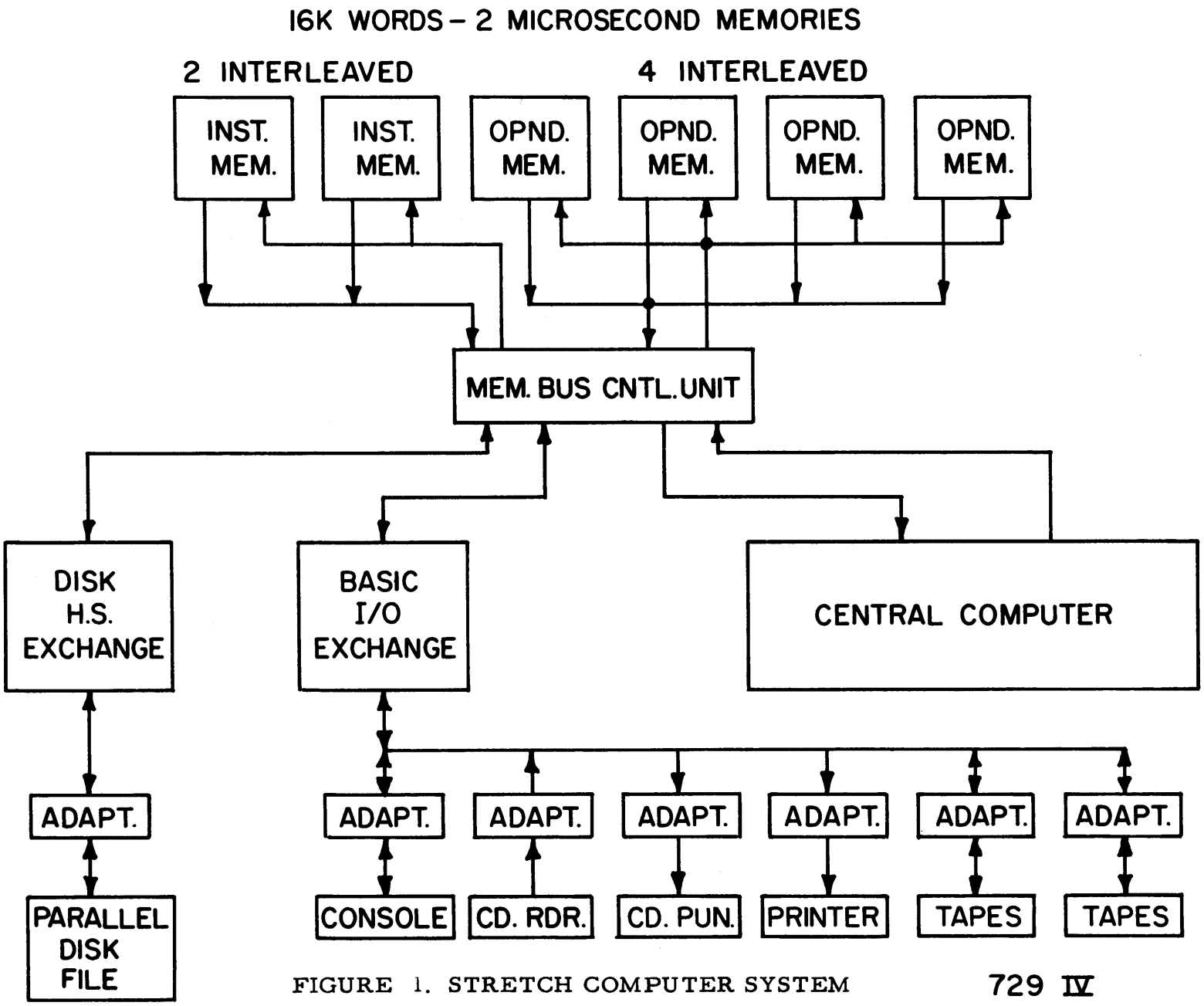


FIGURE 1. STRETCH COMPUTER SYSTEM

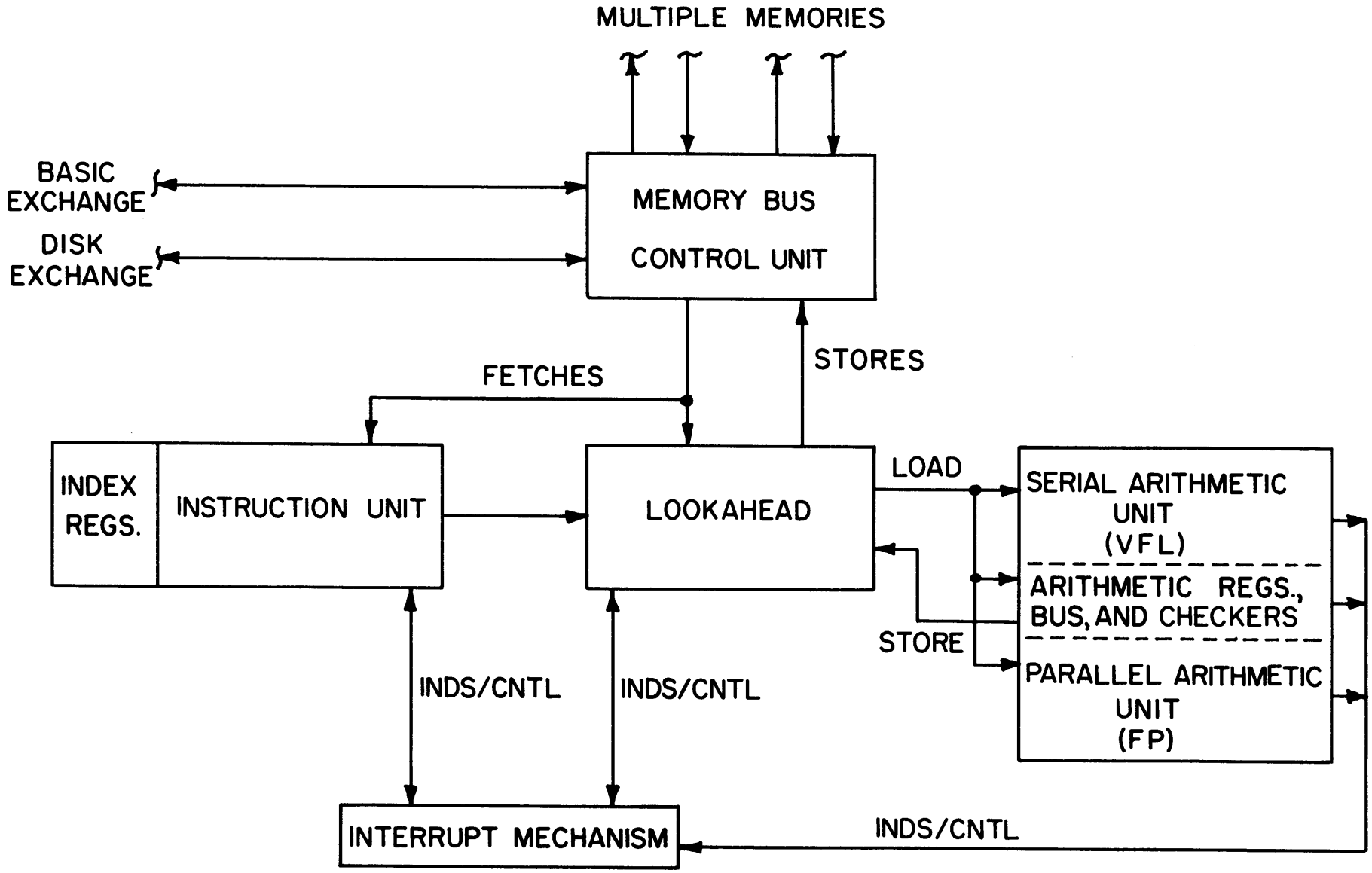


FIGURE 2. STRETCH COMPUTER

1. It had to handle a large diversified instruction set in a wide variety of word formats.[2] (See Appendix A.)

2. It had to prepare half-word instructions, full-word instructions, and full-word instructions across memory word boundaries.

3. In order to achieve the desired performance, it had to process several internal instructions simultaneously.

4. It had to be completely interruptable and recoverable on every instruction.[3]

5. It had to test for many exception conditions, set corresponding indicators, and be capable of no-opping (not execute) the associated instruction if necessary.

6. It had to differentiate between three types of memory (external memory EM, index storage XS, or internal register IR) on all fetches and stores.

7. It had to update the time clocks every millisecond.

8. It had to be a reliable and thoroughly checked unit.

9. It had to be virtually instantaneously stoppable to provide meaningful automatic error scans.

10. It had to be constructed with standard circuits, panels, and frames.

As a result of these requirements, plus the many assigned program functions, the instruction unit was designed and built to occupy five full 20 inch standard frames and parts of three others. It fills 46 standard panels, uses approximately 1275 standard circuit double cards and 6385 standard circuit single cards, and requires about 53, 680 transistors.

FUNCTIONS

The instruction unit has as one of its two primary functions the fetching and preparation of every instruction executed by the computer. The fetching carries with it the responsibility for checking and possibly correcting the word after it is received from memory. The preparation involves the indexing[4] (address modification) of the instruction, if required, the partial decoding to determine unit destination within the computer for execution, plus the actual operand fetch and loading of the instruction into lookahead. It also requires various tests of the instruction for indicator setting and possible no-opping and/or interrupt. Some indicators require that the instruction not be executed (no-opped), while others permit the full execution

of the instruction but may cause an automatic interrupt at the completion of the instruction. The actual point of execution and interrupt testing is not until some time after the I unit processed the instructions. In order to know the memory address of every instruction as it is interrupt tested, the I unit loads the instruction counter value into lookahead with each instruction. If later an interrupt is detected, the program can store the IC value of the instruction following the one being interrupted. This enables the program to know where to return to the main program after the interrupt.

Two types of indexing can be specified: normal and progressive. The normal mode modifies the operand address by the value of the index word; whereas the progressive mode modifies the index value (+ or -) by the operand, addressing and replacing the operand address in the instruction with the original index value. This mode can also specify a stepping of the index count field and may or may not call for an automatic refill operation if the count goes to zero. All instructions to be executed by the floating point (FP) unit, the variable field length (VFL) unit, or the exchange unit are loaded into lookahead to await operand return and subsequent execution. As the instruction is loaded, an operand, if required, is fetched; and any indicators set by the particular instruction accompany it into the lookahead.

The second primary function of the I unit is the execution of a large number of the instructions in the STRETCH instruction set. These instructions include all the direct and immediate index arithmetic, branch type and word transmission type, plus some miscellaneous instructions. The miscellaneous include indirect addressing, geometric addressing, and execution of direct and indirect subject instructions. A complete list of the instruction set executed by the I unit is located in Appendix B.

Another function assigned to the I unit is the monitoring of all memory addresses for an out-of-bounds condition. All memory fetches for the computer are initiated by the unit and each memory address is compared with an upper and lower boundary register. Depending upon the state of an outside-inside control trigger, appropriate indicators are set corresponding to the type of fetch (instruction or data). All stores are performed by the lookahead. However, before loading lookahead with a store operation the I unit must test the address and set a store indicator, if out of bounds. These indicators are later tested for interrupt and may cause an automatic branch to a corrective routine.

Another function of the I unit is the interrupt system. At the completion of every instruction execution, a test must be made on the indicators to determine if an automatic program interrupt is called for or not. If not, then normal program operation is continued. If an interrupt is called for, then the mechanism initiates a recovery operation in both the lookahead and the I unit. The I unit must then determine the indicator causing the interrupt, reset the indicator, and locate the "free instruction" associated with

that particular indicator. It then fetches the instruction, prepares it, and either executes it or loads it into lookahead. If it is not a successful branch instruction, then the I unit returns to the original program and continues normal operation; hence, the term "free instruction." However, if it is a successful branch, then the I unit branches to the new program routine and proceeds normally until a new branch instruction returns it to the original program.

Time-clock operation is another function assigned to the I unit. Every 1024 cycles per second, the interval timer and the real time clock must be stepped. If the interval timer goes to zero, a corresponding indicator is set. There is no indicator associated with the real time clock, and both timers wrap around.

The last important function of the I unit is the providing of manual controls for direct intervention by the operator console and the customer engineering maintenance console. Since the I unit has complete control over all operations to be performed by the computer, it was found to be the logical place for the majority of the manual controls. These controls provide the ability to:

1. Start or halt the machine

2. Load new programs

3. Display or store memory

4. Single step the program; an operation or a cycle at a time

5. Enter a particular instruction into the machine

6. Put the machine in a repeat instruction mode

Repeat instruction mode causes the machine to continually repeat the fetching, preparation, and execution of one particular instruction word. Also included in the manual controls is a set of controls for continuous read-write testing of the index storage.

DATA PATHS ORGANIZATION

The organization of the I unit was arrived at after a great deal of study and evaluation of various alternative schemes. The primary objective of the design was to achieve the correct balance of three major factors. These factors are performance, cost and reliability.

The first step was to design a system with a minimum of hardware that accomplished all the functions assigned to the unit. The next step was to

determine the amount of time-sharing that was possible, and the amount of parallelism and simultaneous operations required to achieve the high performance goals desired; always trying to keep the cost to a minimum. Finally, after having arrived at a satisfactory compromise of the first two points, the system was carefully studied and checking-correcting ciucuitry added to obtain the high degree of reliability desired.

The result was a machine organization as shown in Figures 3 and 4. Figure 3 is a block diagram of the data paths for the principal part of the I unit, and Figure 4 is a block diagram of the interrupt mechanism. The basic I unit organization consists mainly of six transistor registers, two adder units with checking, six data paths, two address busses, and one checker-corrector unit. In addition to these, there are indicator and tag storage positions, a boundary compare unit, a program store compare unit, lookahead load data transfer busses, and a special purpose, left-most one, detector-encoder for the geometric load (load value effective) instruction. The interrupt mechanism consists of two transistor data registers, one unit address register, and a left-most one detect-encode mechanism. These data paths account for approximately 40 percent of the hardware. The remaining 60 percent is taken up by extensive control logic (with its associated timing) and decoder circuitry.

The six main registers in the I unit are the instruction counter register (ICR), the instruction-data word buffer registers (1Y and 2Y), the index register (XR), the preparation and execution register (ZR), and the multi-purpose working register (WR).

Instruction Counter Register (ICR)

The IC system was designed to provide the actual memory address of the instruction currently being executed or prepared for lookahead in the Z register, and at the same time fetch succeeding instructions into the Y registers. Since a number (6) of instructions may be located in the Y and Z registers simultaneously, some means had to be developed to keep track of the instruction addresses as the instructions proceed through the I unit. It was found that we could eliminate the multiple IC registers by using the ICR to keep track of the instruction being processed in the ZR. Output combinations of the unit's position inverter and the IC adder could be used to fetch following instructions.

The ICR contains 21 bit positions, two of which are parity bits. The low order two bits are separated from the high order seventeen bits. They have their own individual advancing mechanism in order to provide the flexibility needed for advancing by half and full words and fetching succeeding instructions. The high order seventeen (0-16) bits feed a plus-one, parallel, carry lookahead adder, which actually provides the ICR quantity plus-two (+ 2) address. The seventeenth bit position is available in true

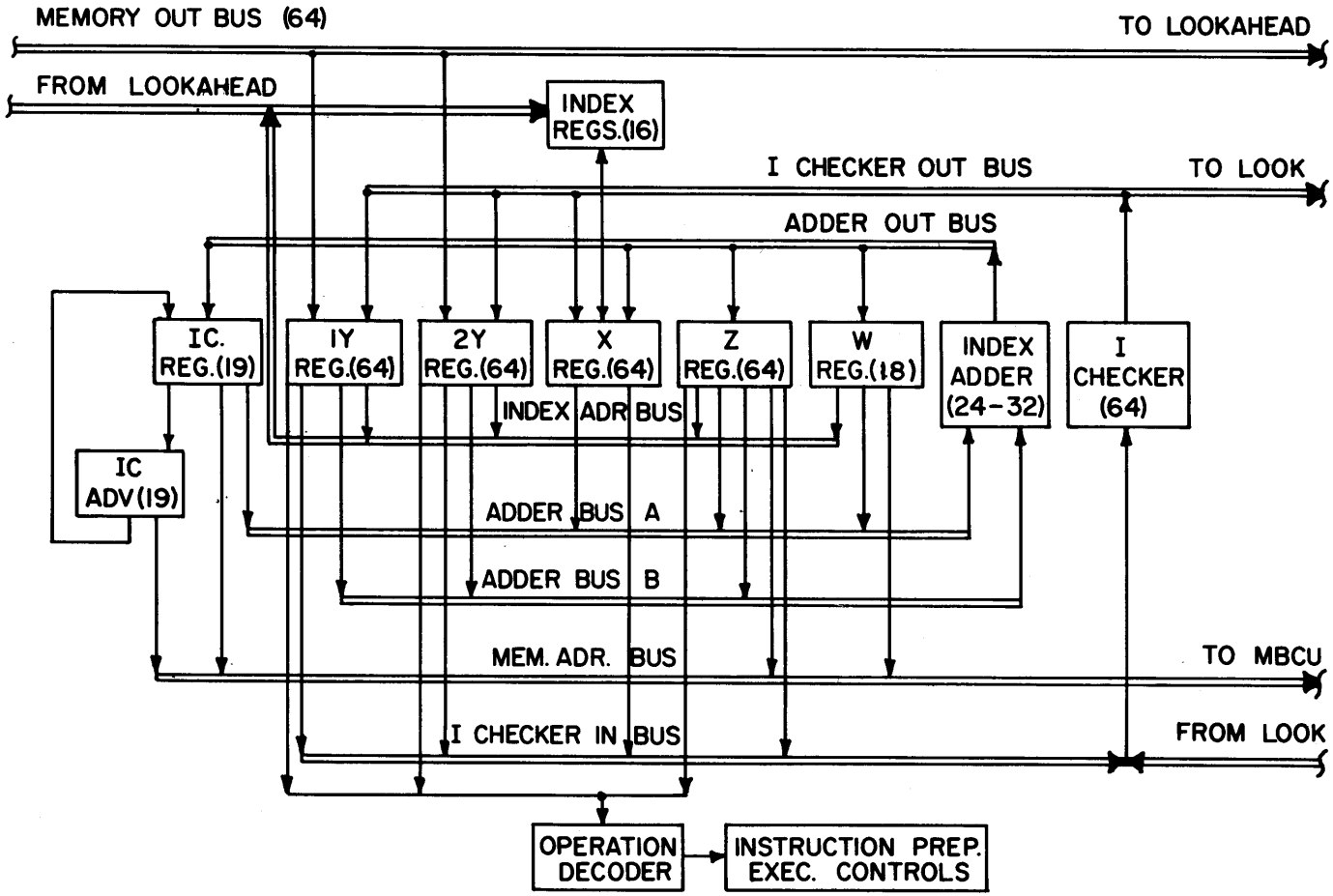


FIGURE 3. INSTRUCTION UNIT

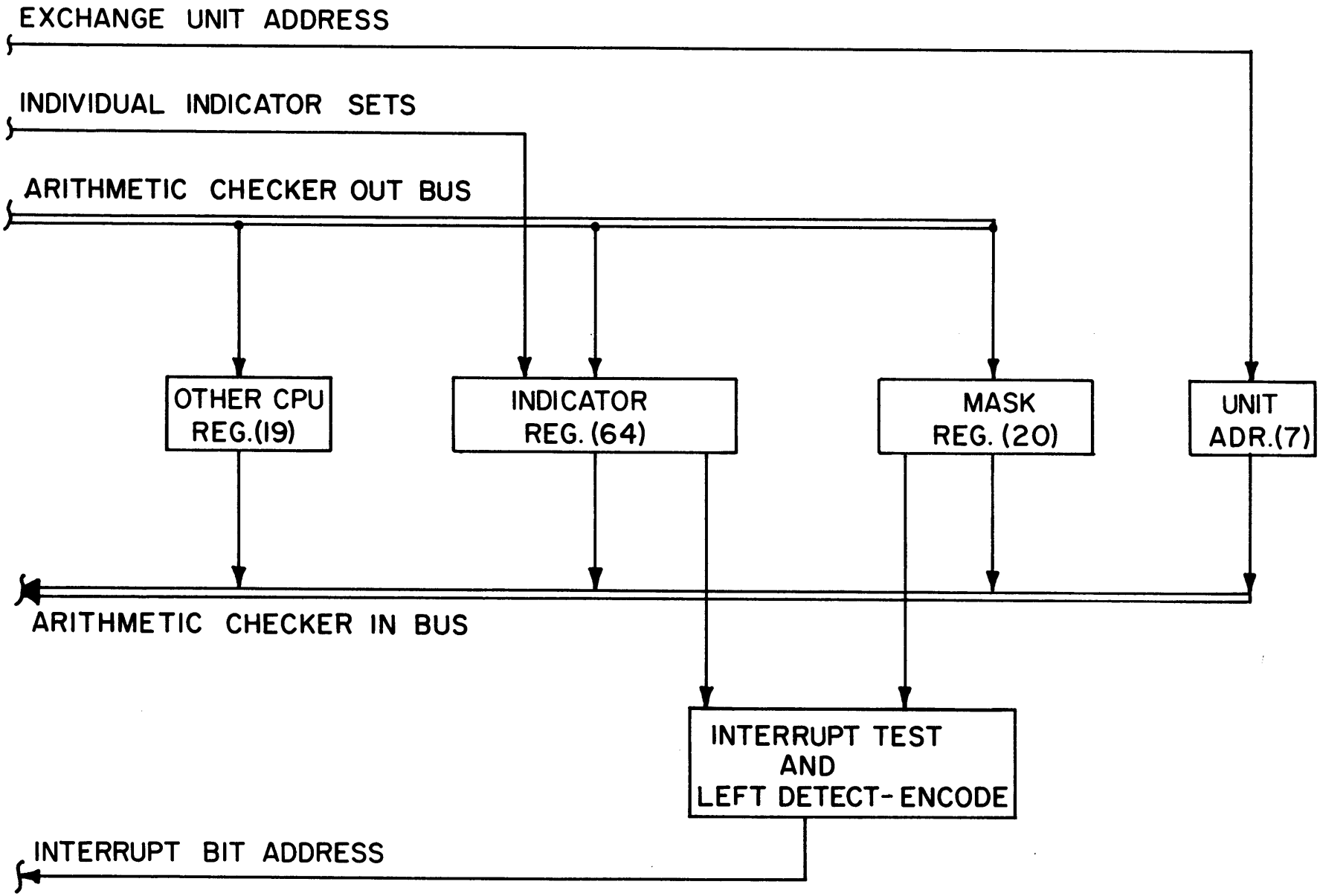


FIGURE 4. INTERRUPT MECHANISM

and complement form. By selecting combinations of ICR and adder outputs, we can obtain four addresses: n, n+1, n+2, and n+3, where n is the ICR address. This provides all the lookahead addresses needed for pre-accessing instructions into the Y register instruction buffers.

The ICR has a set of in-gates (21) from the lookahead IC buffer for use in recovery operations. It also has a set of in-gates from the index adder out bus for branch operations. Finally, there is a set of in-gates for setting the IC adder output into the register for a full advance of the IC system.

The ICR positions 0 and 16, both true and complement outputs of position 17, and the IC adder output positions 0 and 16 can all be gated out to the memory address bus for instruction fetching. The ICR also can be gated to the index adder in bus A (ABA) for store instruction counter and branch operations. The ICR has a complete set of d-c output lines to the lookahead IC input gates. These output lines are for loading the associated IC address, along with the instructions, into lookahead and for program store testing. It has a set of d-c output lines to the IC adder and several special d-c lines to the control area. A full set of d-c lines go to the maintenance console for indicating the contents of the ICR and the IC adder output.

Instruction and Data Buffer Registers (1Y and 2Y)

In order to achieve high speed instruction preparation, particularly floating point instructions, it was necessary to provide two instruction buffer registers. These two registers are identical and are used alternately for receiving instruction words fetched from memory by the IC. They are also used for data operands required for the execution of instructions performed in the I unit. They each contain 73 positions; 64 data bits, 8 check bits, and 1 memory check bit. The 8 check bit positions may contain error correcting code (ECC) bits or parity bits. All words in memory contain error correcting code bits, but during the checking-and-correcting operation in the I unit the ECC bits are replaced with parity bits for checking internal operations. The memory check bit indicates whether or not an error occurred during the memory access and, therefore, whether the word received is valid or not.

The Y registers have two complete sets of input gates; one from the memory out bus (IMOB) for memory fetch returns, and one from the checker out bus (ICOB) for check-correct operations and internal word transfers.

Each Y register has a full set of out-gates to the checker in-bus (ICIB), again for checking-correcting or internal word transferring. Each register also has two sets of half-word (36 bit) gates to the adder in-bus B (ABB) for direct transfer or logical operation through the index adder. Four sets of out-gates permit addressing index storage (XAB) from the two I fields of each register.

Both registers have 73 d-c lines to the maintenance console for indication, plus a number of d-c lines to the control areas for instruction pre-decoding and special memory addresses.

The parity fields are split up across the Y registers in the following manner:

P_0 (0 - 17)	P_4 (32 - 49)
P_1 (18 - 23)	P_5 (50 - 55)
P_2 (24 - 27)	P_6 (56 - 59)
P_3 (28 - 31)	P_7 (60 - 63)

These fields were found to be the best combination for effectively handling all the different word formats encountered in the system.

Index Storage (XS and XR)

The specifications for the STRETCH computer called for 16 index words located in high speed storage. The first approach was to use 16 transistor registers, but this was soon found to be undesirable for several reasons. One was that it was extremely expensive, particularly if each register was to be capable of directly operating with the index adder to perform all instruction indexing and all index arithmetic operations. Another reason was that the large amount of hardware presented a packaging problem and detracted from the anticipated high performance. It was apparent that a buffer register would be required, which alone would have all the necessary logical capabilities. Each of the 16 index registers could be transferred into it prior to execution. This still presented a large and expensive piece of hardware. The ideal solution was found by providing a compact, high-speed, 16-word, non-destructive-read, core memory for the index words. It has one data register (XR) to read into, store out of, and perform all the logical operations required.

The index storage (XS) was actually designed with 17 words of 73 bits each. The seventeenth word contains the interval timer and the elapsed time clock for rapid access and for advancing of these values. It is a two dimensional array (17 x 73) and has a read-out time of approximately 200 mμs. Total access time including address gating, transmission, and decoding takes up one machine cycle of approximately 500 mμs. To store the contents of the XR into XS requires two cycles. The first one destructively reads the selected word, thereby resetting it to zeros; the second cycle writes the XR contents into the selected row of cores. The index address is checked prior to decoding. The XR is checked during the following logical operation for which the index word was fetched.

The index register was designed with two principal objectives in mind. One was to provide the function of a data register for fetching and storing to/from index storage. The second was to provide the ability to perform all the full-word, half-word, field transfer, shift, and logical operations required to execute all the I unit instructions. The result was that the XR has five sets of in-gates and out-gates with extensive splitting of control fields.

This register contains 73 bit positions including 64 data and nine parity check bits. Eight of the check bits correspond to the eight in the Y registers. The ninth is the parity on bits 46 to 49 to provide means of obtaining a parity check on the index count field.

The primary input to the XR is directly from the sense amplifiers of the index storage during an index fetch operation. A full set of gates (73) allow gating from the checker out bus straight into XR for checking and full-word transfer operations. Four sets of gates allow gating from the adder out bus into four different fields of the XR. These fields are: The left half of XR beginning at position 0, the right half of XR beginning at position 32, the count field beginning at position 28, and the refill field beginning at position 46. These gates are all used in the execution of various types of index instructions. In addition to these inputs there is a direct reset of all 73 positions for setting the XR to zero prior to a read-out of index storage.

The XR has a full set of out-gates (73) to the checker in bus for checking and full-word transfer operations. It has three sets of partial gates to the adder in bus "A." These provide the ability of gating the value field, the count field, and the refill field to the adder for logical operation or transfer through this unit. One set of gates (24-27) in bus B provide the ability to check the sign of the value field during operations in the adder.

There are three d-c detector circuits connected to the XR. These circuits provide the following indications:

X value less 0	X count equal 1
X value equal 0	X count equal 0
X value greater 0	X refill equal all 1's

These are used to set indicators or modify execution controls for various operations. In addition, there are a number of special d-c outputs of the XR which feed adder true/complement controls, execution controls, and parity adjust logic in the parity checker/generators. A full set of d-c outputs go to the maintenance console for indicator purposes.

Preparation - Execution Register (ZR)

This register is the basic operating register of the I unit. Every instruction is placed in this register from the Y registers for indexing, decoding, execution, and lookahead loading. It is full-word in width to accommodate full-word instructions and to speed up floating point half-word instructions. All full-word instructions appear straight, left to right in the register, regardless as to how they were received from memory and placed in the Y registers. There are a large number of output gates on the register because of the many special functions it performs. These include indexing (normal and progressive), executing I unit instructions, fetching of operands, and loading of the instructions into lookahead.

The ZR contains 74 bit positions, including 64 data, and ten parity bits. Eight of the ten parity bits correspond to the standard eight in the Y and X registers. The other two are for parity on the channel address field (12 - 18) for I/O instructions, and the length field (35 - 40) of VFL instructions. Associated with the ZR is a small three bit P register which is used solely for retaining the progressive indexing code (bits 32 - 34) during an index modification of the right half of a VFL instruction.

The only in-gates provided on the ZR are for gating the adder out-bus into various positions of the ZR. There are two sets of these gates, one for the left half of Z and the other for the right half. These gates have split control to provide for partial gating. This takes care of all the in-gating required by instruction transfers from Y to Z, plus all arithmetic result gating into Z.

The out-gating of the ZR is more extensive and complicated. There are two sets of out-gates for gating the left half or the right half of Z to the adder in bus B for arithmetic operations on the left or right operand addresses. There is a separate gate for gating the length field (35 - 40) to the adder in-bus A for word boundary cross-over test. Another gate sends the immediate count field (50 - 55) in transmit instructions to the W register for counting purposes. There are two sets of out-gates provided for gating the left (0 - 17) or right (32 - 49) operand address to the memory address bus for operand fetches. There are five sets of gates for: gating the left or right operand address, the left or right J field (index operand in index instructions), and the left I field (index address for index modification) to the index address bus. These permit fetching of index word operands and the index fetch for the delayed modification of the left half of Z.

There is also a set of ten out-gates to the checker in bus for rearranging the fields for loading instructions into lookahead. These also have split control for selecting the width of the fields.

In addition to the in and out-gates, the ZR has 41 positions line driven to the control area for operation and memory area decoding. As in the case

of all the registers, all positions of the ZR have a d-c line to the maintenance console for indicator purposes.

Working Register (WR)

This register is only 19 positions long (including one parity) and is used primarily for operand address storage, and secondarily, as the counting register for transmission type instructions. It is used for storing the second operand address for VFL instructions which cross memory word boundaries, for geometric address decoding, and automatic refill and interrupt address operations. In transmit operations the direct or immediate count field is placed in the W register for counting purposes. The from and to operand address remains in Z for fetching, storing, and stepping operations.

The WR has three in-gates: one from the adder out-bus for transfer and arithmetic operations; one from the maintenance console keys for manual insertion of an address; and one from the interrupt bit address encoder for automatic interrupt operations.

The WR has three out-gates; one to the adder in-bus A for arithmetic operations such as counting; one to the memory address bus for word boundary cross-over and refill fetches; and one to the index address bus for index fetches.

The d-c outputs of the WR feed a left-most one detect logical unit for geometric address decoding. The output of the detect circuit feeds an address encoder, the output of which is set into a five bit register called the geometric load address register (GLAR). The detect and encoder logic is checked.

The d-c outputs of the WR also feed various decoder circuits for determining special address and contents equal to one condition. The output of the left-most one detector feeds the adder in bus B. This resets the current geometric address bit by a subtract operation through the index adder. The output of the GLAR feeds the index address bus for index fetching during goemetric load execution. All 19 positions of the WR and five positions of the GLAR go to the maintenance console for indicator purposes.

Index Adder Unit (IAU)

The specifications for the I unit called for arithmetic operations on fields up to 24 positions wide. High performance required a parallel adder of advanced design with a minimum of logical levels. In order to guarantee complete reliability it had to be thoroughly checked. Since the many operations of the I unit required that all the registers be capable of feeding the adder, it was found that the transfer and adder paths could be combined and time-

shared to provide the most economical and yet completely checked system. This was accomplished by providing an eight bit bypass path around the 24 bit adder to permit half-word (32 bit) transfers. A further study indicated that the best performance could be gained by providing a controlled complementer on one input, with automatic recomplementing ability on the output. The basic 24 bit parallel adder was broken up into six four bit blocks with parallel carry lookahead and carry propagate detect logic for each. Carries from block to block and end around carries are detected early and propagated through. The 24 bit adder consists of five logical levels. Input checking is accomplished by comparing input parities with the half-sum parity. The rest of the adder is checked by a carry prediction checking method.

Standard I unit parity is generated on the output in parallel with special memory address decoding. The bypass eight positions are parity checked and transferred to the output bus. The output is completely latched at sample time to prevent race conditions along the d-c paths when gating the result back into one of the input registers.

The adder in-bus A (ABA) has a complementer on the input and is accomplished in the same logical level as the ORing. The inputs to ABA are the IC, WR, XR, and ZR. The ABB is not complemented and the inputs come from the YR's and the ZR. In addition there are numerous lines to control the complementing, recomplementing, and parity adjustments for various fields and operations. The adder out-bus (IAOB) has 32 data bit positions plus 11 parity bit positions for selection, depending upon the fields involved. The IAOB feeds the left and right halves of the ZR, the left and right halves count, and refill fields of the XR, the WR, and the ICR.

The I Checker

One of the earliest requirements of the STRETCH system was the automatic error correction of memory words. The method adopted was that of using the Hamming[5] error correcting code (ECC) which required eight ECC bits with a 64 bit data word. It was necessary to be able to check and correct memory words (instructions and operands) in the I unit and to convert them to the unit's parity system. It was also necessary to provide a full-word transfer-bus for high speed transfer operations in executing many of the I unit instructions. It was found that these two operations could share equipment by combining the transfer-bus with the full-word ECC checking-correcting and parity checking-generating logic. It was also found that the ECC checking-correcting operation on instructions could overlap with the initial pre-decoding required on the new instructions when they are received in the Y registers.

Lookahead had a similar problem with ECC checking-correcting and parity checking-generating on operands, fetched to lookahead by the I unit, and with storing result operands to memory. A study of the two units showed

that one I checker unit could be time-shared between the I unit and lookahead, provided a fast priority system could be designed to guarantee little loss in performance. This was done, and the result is a single I checker with separate in-busses from the I unit and lookahead, OR'd at the input to the checker. The checker out-bus (ICOB) is a 64 bit data-bus, plus 29 parity and ECC lines. These are to enable the controlling unit to select the proper parity or ECC bits for the receiving register. This bus feeds the four lookahead levels first and then the XR and the 2Y registers. The output of the checker is completely latched at sample time to prevent race conditions along the d-c paths while gating the result into the receiving register.

The I checker is located in the lookahead unit and was not a part of -- but coordinated with -- the I unit design.

Interrupt Mechanism

Figure 4 showed a block diagram of the interrupt mechanism which was designed by the I unit group and packaged in the lookahead area. It consists largely of a 64 position indicator register (IR), a 28 position mask register, and a left-most one detect and encoder circuit.

The indicator register has two inputs: one is from the arithmetic checker out-bus (ACOB), and the other is the individual turn on-line from each position's particular logical area. These logical areas include the I unit, the VFL and FP execution units, lookahead, exchange, and memory. There is no parity on the contents of this register because it is continually changing, due to the many asynchronous inputs. The only out-gate on the IR is to the arithmetic checker in-bus (ACIB) and is used for transferring the contents of the register to another location. Similarly, the input gate from ACOB is for bringing in a new word to the IR. Certain d-c outputs from a portion of the register feed the updated indicator register in the I unit for recovery purposes. A full set of d-c outputs feed the left-most one detector and the maintenance console.

The mask register has its only input from the ACOB for bringing in a new mask word. It can also be gated to the ACIB for storing purposes, and has d-c outputs to the left-most one detector and the maintenance console. Logically, the first 20 positions (0 - 19) of the mask register are always one, and the last 16 positions (48 - 63) are always zero. They are fixed and cannot be programmed. There are four parity bits associated with the mask register positions 21 - 47, conforming to the parity fields of the arithmetic bus and checker.

The left-most one detect circuit has two functions: first to test rapidly for any match between an indicator positon and its associated mask bit, and second, to determine, in the case of multiple matches, which one has a

higher priority. Priority is established from left to right (0 - 63) and the match with highest priority blocks the remaining ones from being effective. The test for any match is done in only a few levels by ORing all the "compare and" circuits, and signalling an interrupt if the mechanism is enabled. The enabling/disabling is controlled by a single trigger which is set on/off by programming. Once an interrupt is signalled, the left-most one detect logic is allowed time to establish priority. It encodes the matching pair of indicator and mask bits into the register bit address of the particular indicator; then transfers them into the WR. The bit address contains a six bit address plus one parity bit.

Included in this logical area is a seven bit channel address register which holds the address of the I/O unit which last set I/O status bits into the indicator register. The register may be set by either the basic or the high speed exchanges and includes two parity bits. It can be gated to the ACIB (positions 12 - 18) for transfer purposes.

Also designed for this area but not packaged was the other CPU register. This register of 19 positions is used for systems involving more than one computer. It can be gated out to the ACIB and gated in from the ICOB for transfer purposes. There are four parity bits associated with the 19 bit field to conform with the arithmetic bus and checker requirements.

Updated Indicator Register

There are eight triggers in the I unit which contain the status of the index register value, count, and refill fields up to the last instruction executed by the I unit. These indicators differ from the status of the corresponding main indicator register triggers by the effect of the instructions which were processed by the I unit and which are still in lookahead, awaiting to be processed by the VFL or FP units. Hence, the term updated indicators. This UIR is used to test for conditional branches on these indicators.

A listing of the indicators contained in the updated indicator register is as follows:

1. Index low - XL
2. Index equal - XE
3. Index high - XH
4. Index count zero - XCZ
5. Index value less than zero - XVLZ
6. Index value zero - XVZ
7. Index value greater than zero - XVGZ
8. Index flag - XF

Indicator and Address Tag Triggers

Due to the multiplicity of instructions that are contained and processed simultaneously in the I unit, it is necessary to tag (store with the particular instruction) each instruction with information regarding certain conditions which may arise during their processing. Examples of these are memory and ECC checks on new instructions, parity checks on instruction transfers from Y to Z, and special memory addresses decoded during the transfer through the index adder. These triggers are mostly located in the control area so that they can immediately condition the control trigger outputs for the following cycle. There are some 26 triggers of this nature in the control area of the I unit. These include the following:

A. Y Register Tags

1.	1Y instruction fetch indicator	1YIF
2.	2Y instruction fetch indicator	2YIF
3.	1Y operand address invalid	1YAD
4.	2Y operand address invalid	2YAD
5.	1Y identifiable check	1YIDC
6.	2Y identifiable check	2YIDC
7.	1Y memory check	1YMC
8.	2Y memory check	2YMC

B. Z Register Tags

1.	Z left operand address-special (0-15)	ZLSA
2.	Z left operand address-index (16-31)	ZLXA
3.	Z left operand address-non-existent	ZLNA
4.	Z right operand address-special (0-15)	ZRSA
5.	Z right operand address-index (16-31)	ZRXA
6.	Z right operand address-non-existent	ZRNA
7.	Z left instruction fetch indicator	ZLIF
8.	Z right instruction fetch indicator	ZRIF
9.	Z left operand address invalid	ZLAD
10.	Z right operand address invalid	ZRAD
11.	Z left contains identifiable check	ZLIDC
12.	Z right contains identifiable check	ZRIDC
13.	Z contains data store condition	ZDS
14.	Z contains data fetch condition	ZDF

C. W Register Tags

1.	W operand address-special (0-15)	WSA
2.	W operand address-index (16-31)	WXA
3.	W operand address-non-existent	WNA

D. General

1. I unit contains non-identifiable check NIDC

Memory Addressing

Every memory address has to be decoded as to the type of memory involved before any fetch or store operation can be completed. There are three types of memory that may be referred to: internal transistor registers, index storage words, and large external core memory. The first thirty-two words of addressable memory are reserved for special purposes and have permanently assigned addresses. (See Appendix C.) These addresses may be located in any of the three types of memory.

In the majority of cases, the memory address is pre-decoded in the index adder and tagged as to the type of memory referred to. Subsequent fetches or stores with the address are conditioned by the tag triggers to refer to the proper type of memory. In some cases where there was insufficient time to decode, notably in rapid FP instruction preparation, a "guess" is made that the operand address refers to external memory and the required fetch initiated. The memory area decoding is completed before the fetch is actually executed. This operation can either permit the fetch to be completed or cancel it. If cancelled, the proper fetch is initiated to the correct memory area on the following cycle. Since most operand addresses refer to external memory, this "guess" should save a decode cycle in the majority of cases. In the cases where the "guess" is incorrect, no time is lost since the decode time was required anyway.

All external memory fetches are executed by the I unit and involve sending an 18 bit word address and a three bit return address to the bus control unit. The return address indicates which Y register in this I unit, or which of the four lookahead levels is to receive the word on return. All stores to external memory are done by lookahead. The I unit loads the store address into the lookahead address register and a store type operation code into the op code field. The operand may be loaded at the same time, in case of an I unit instruction, or may be loaded later as a result of a VFL or FP arithmetic operation. As the store memory address is transferred to lookahead via the memory address-bus, it is tested for a data store boundary alarm (DS). If "out of bounds," the lookahead may cancel (no-op) the actual store operation.

All index word fetches are done by the I unit whether for I unit operations or for VFL and FP operands. The fetch is made by gating a four bit address to the index address-bus, and controlling the read-out of the index storage and the reset of the index register. Index stores can be made by either the I unit or the lookahead unit. The operation is similar to the fetch except that two cycles are required, one for resetting, and the other for writing the contents of the XR into the selected row of cores. In the case of VFL

and FL stores to index storage, the I unit loads the store type instruction into lookahead; then waits until the entire operation is completed, including the resultant store back to index storage.

Internal transistor register fetches and stores are made by lookahead on the arithmetic checker bus. The fetches and stores are set up by the I unit in loading a lookahead level with the address and properly coding the operation code field. If the fetch is for the I unit, lookahead will first transfer the contents of the register to the lookahead level on the arithmetic checker bus, and then relay it on to one of the Y registers via the I checker bus. If it is a VFL or an FP operand fetch, it is left in the lookahead level for later transfer to arithmetic operand register. The I unit stores to internal registers by loading the word into the lookahead level along with the address and operation code.

Memory Address Monitoring

The specifications called for a comparison of every address with upper and lower boundary registers sent to memory. These 18 bit registers are located in the lookahead area. The compare logic, however, was designed by the I unit and located on the memory address-bus in the bus control unit.

The I unit does all memory address monitoring by comparing each fetch address as the fetch is being made and all store addresses while loading lookahead with the store instruction. The logic consists of two separate units, each comparing the memory address-bus against the upper boundary in one case and the lower boundary in the other case. The comparison is made 18 bits parallel in five logical levels and indicates whether the contents of MAB is inside the bounds or outside. Inside bounds means equal to or greater than the lower boundary and less than the higher boundary A programmable control trigger determines whether an alarm should occur for the inside or the outside bounds condition. The compare has to be executed rapidly to insure that the proper indication and action can be taken during the same cycle that the fetch or store transfer to lookahead is occurring, and the particular memory address is on the MAB.

The alarm signal can cause one of three different indications to occur. These indications are stored in tag triggers associated with the instruction either in one of the Y registers, the Z register, or the lookahead level that the instruction is being loaded into. They are later transferred directly into the main indicator register and tested for an interrupt. If an instruction fetch is in progress, the IF tag trigger of 1Y, 2Y, ZL, or ZR may be set. If an operand fetch is being made, the DF tag trigger of ZL or ZR, or the lookahead level being loaded may be set. If an operand store is to be made, then the DS indicator in the lookahead level may be set during the actual lookahead load cycle.

Program Store Compare

Certain problems are encountered in a system which preaccesses instructions. A store type instruction, for example, presently being executed, can store into the address of an instruction which has already been fetched. In order to detect this condition and be able to correct it, it was necessary to design two logical units for comparing the memory address-bus with the IC register and the IC adder output. Whenever a store is loaded into lookahead this comparison is made. If an equal condition results it indicates that the contents of that location have been fetched already and loaded into the I unit. A recovery is always made under these conditions.

This does not solve the problem entirely. Since lookahead has four levels, all subsequent fetches have to be compared against the store address, once it is loaded into lookahead. If an instruction fetch or an I unit operand fetch compares equal, the fetch is held up until the store is completed. If an I unit operand fetch for lookahead compares equal with the store address, appropriate controls are effected. These initiate a "forwarding" operation of the store operand to the designated fetch level, without first requiring a store to memory. This comparison is done in another compare circuit between the memory address-bus and the store address register in lookahead. None of these comparisons cause recoveries of the I unit.

CONTROL PHILOSOPHY

One of the earliest studies made in the project was on the relative merits of synchronous and asynchronous control methods. Several asynchronous control techniques were proposed and evaluated both on paper and in data flow models. At the same time, work was done on developing a synchronous control system which would best suit the size, complexity, and speeds of the STRETCH computer. Initially asynchronous control looked attractive, because it was thought to be inherently faster, more reliable, and more adaptable to changes in circuit components. Added to this was a general apprehension that synchronous control of such a large, high-speed system was impractical . It was found very difficult, however, to achieve these advantages at a reasonable cost. The additional logic required to determine operation completion and sequence interlocking was such, that it not only added greatly to the cost of the system, but detracted from the anticipated speed and reliability. The major objection to the synchronous approach was removed by proposing a system which would operate asynchronously between units, but synchronously within the units. The clock distribution skew could be minimized by delay line techniques, and by distributing a minimum number of master clock lines to the main units of the system where they could then be powered up and distributed internally. Since a satisfactory asynchronous system could not be found, and the feasibility of synchronous control was established, the decision was made to implement the latter type in the STRETCH computer.

The next step was to define a basic system of synchronous control for the I unit. Several factors greatly affected the final form of this control. These included the type of circuits used, the amount of overlapped simultaneous operation, the degree of asynchronism and interlocking between control areas, the basic operating cycle time desired, and the overall packaging aspects of the data paths and controls.

High speed, direct coupled, drift transistor current mode circuits were used throughout the design with an estimated delay of 10 to 20 millimicroseconds per logic block. To achieve fast execution times, all the paths were designed in parallel with a minimum number of serial logic levels. This gave rise to the problem of "race" conditions along successive d-c paths, whenever data or control logic was advanced. Several methods of avoiding this were developed and used in different areas. One method (used in the stepping of the control stages) was to split the clock pulses into A and B pulses, and thus alternate the control sequencing. Another method (used in data paths) was to provide a latch circuit in each path. During clock sample time, this stored the state of the line until the clock pulse disappeared. The third solution proposed was the use of delay lines and short clock pulses, which would keep the logic from advancing more than one step during any one sample time. This was never adopted because of the time-dependent nature of the solution and the individual "custom tailoring" implied.

The controls for the I unit were split up into five major areas: instruction preparation, lookahead loading, IC controls, instruction execution, and miscellaneous. The controls were designed separately in each area but closely coordinated between areas to provide a maximum sharing of common equipment, accurate and efficient interlocking, and overall continuity of control design.

The execution of each operation was broken up into individual logical steps. Each step or cycle required a control trigger to condition and select the data paths associated with the particular logical operation, and to condition the turn-off of the previous--and the turn-on of the next -- control stage. Because of the variety of operations in the I unit, it was found that no fixed sequence of steps could reasonably be specified. Instead, a very flexible system of control was defined, where the stepping could proceed from any one control stage to any other. As the sequence for executing each function was detailed, the requirements for the control triggers became evident.

In many cases the stepping of the controls is interlocked with a condition from another control area. In this case, a memory trigger must be associated with the control stage to remember that the operation has been completed and the next step is desired. This fact, combined with the "race" conditions previously described, made it desirable to supply each control stage with two triggers (sometimes more, depending on the branching factor), and to use the A-B clocking method. This eliminated the need for numerous latches in control stepping.

The two triggers are called the E (execution) and the M (memory) triggers, since one controls primarily the data paths for the present operation while the other controls the selection and advancing to the following cycle (either immediately following or some time later). A typical control stage is shown in Figure 5. The E trigger is turned on by an A clock pulse. Its outputs control the output gating of the data paths, the turn-off of the previous M stage and -- at the next B pulse -- the turn-on of its own. When the M trigger comes on, it causes the turn-off of the E trigger, and the turn-on of the next E trigger at the following A pulse. The entire cycle, then, is defined as the time from the rise of the A pulse, which turns it on, to its completion which turns it off. This is twice the period of the clock, since A and B pulses arrive alternately from the master clock in a symmetrical sequence. The register in-gates are sampled by continuous A and B pulses. The select condition from the control stage is formed by ANDing the E and M outputs, always selecting the in-gate at A time. In-gate controls are latched to prevent race conditions at M turn-on and E turn-off times. Control overlap is accomplished by selecting and turning on control stages simultaneously with the result in-gating of the previous cycle. This means that all conditions arising out of the logical operation of any cycle must be available early enough to travel to and condition the input logic of the following control stage.

The control paths and data paths were all designed for a maximum of twenty-four logical levels. This included the cabling between frames, which was estimated at 15 feet of coax equivalent to the delay through one logic block.

Assuming a maximum delay of 20 mμs per block means that a control cycle should take approximately 500 mμs, allowing 20 mμs for clock distribution skew. This means further that the master clock should operate at a frequency of four Mc and supply a symmetrical waveform, 125 mμs up and 125 mμs down, with a period of 250 mμs. The precise operating times will not be established until the system is completely assembled and tested.

CONTROL ORGANIZATION

The remaining hardware in the I unit is taken up by this extensive and complex control system which regulates the flow of instructions and operands, in many different combinations of simultaneous and asynchronous operations, in the data paths previously described. The system consists primarily of many control triggers, commonly referred to as control stages or sequencers, plus their input and output switching logic and ORing of control lines to the data paths. Also adding to the hardware is the extensive decoder logic required to determine which operation and variation is called for in instruction preparation and execution. Extensive powering is required for the distribution of the clock pulses associated with the controls. The control hardware amounts to approximately 60% of the total I unit.

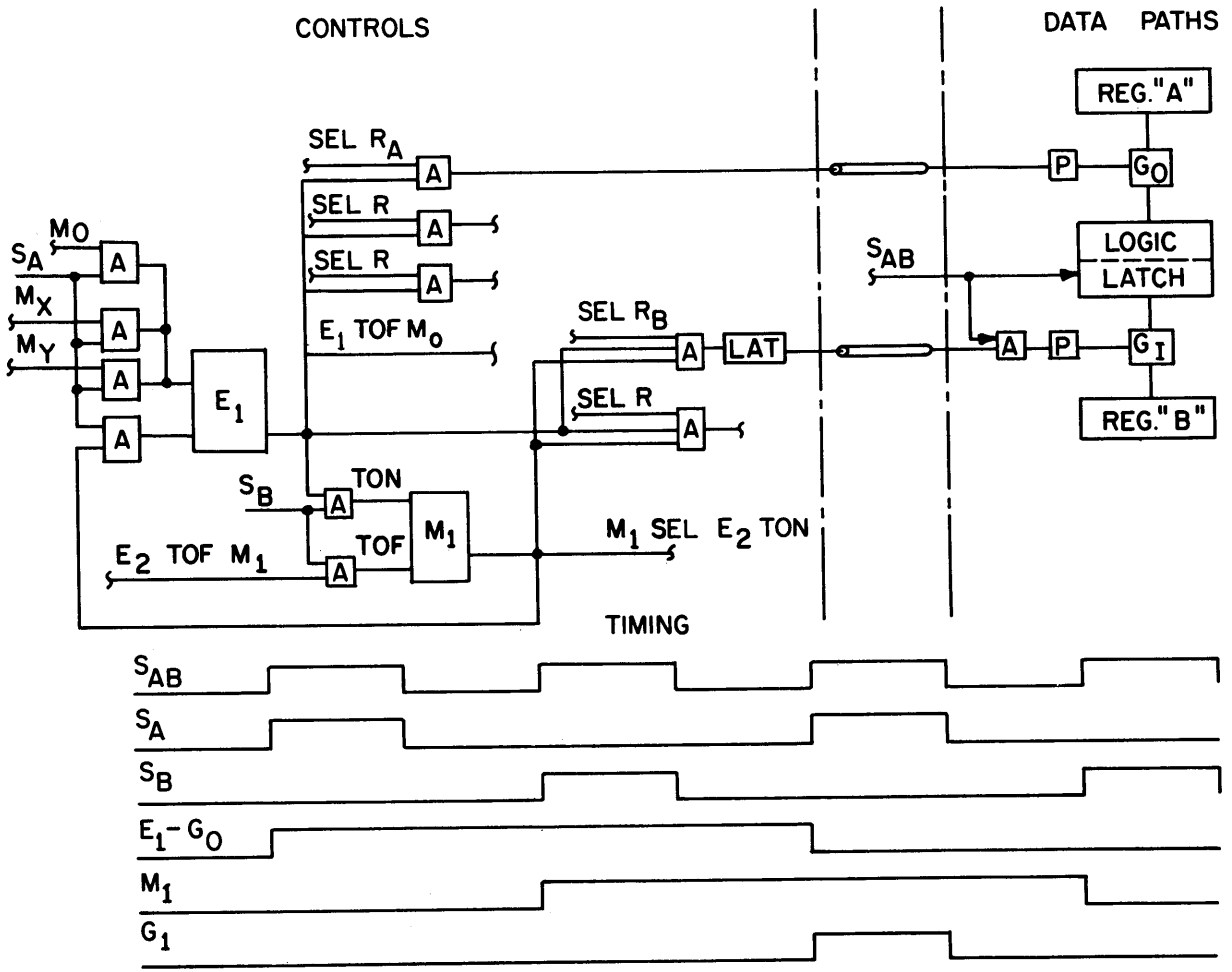


FIGURE 5. GENERAL CONTROL STAGE AND TIMING

The controls are divided logically into the following categories:

1. Instruction counter
2. Instruction preparation
3. Lookahead loading
4. Instruction execution
5. Miscellaneous

IC Controls

The instruction counter controls consist of eight control stages for sequencing the fetching of instructions to the Y registers, checking-correcting them through the I checker, and advancing the IC register. There are thirteen supervisory control storage triggers to condition the sequences as to whether a fetch is in progress (outstanding), 1Y or 2Y is empty, ZL or ZR is empty, a branch to 1Y or 2Y is required, or a recovery is required. Six tag triggers indicate whether there are any checks, instruction fetch alarms, or invalid addresses associated with the instructions in the Y registers. There are eight block-and-suspend triggers for instantaneously interrupting the normal IC operation to allow the I unit instruction execution controls use of the Y registers.

These controls attempt to keep the Y registers filled with new and checked instructions. They signal the preparation controls whenever a Y register is ready for transfer to Z. In branch and recovery operations the IC controls are designed for rapid resetting and restarting at the new address in order to minimize the time required to refill the Y registers.

These controls time-share the I checker and the memory address-bus with other controls, but operate simultaneously with the preparation and look-ahead load controls as long as no interlock occurs. The latter may indicate that no YR is empty; that an I unit instruction requires execution; or that an interrupt is required.

Preparation Controls

The preparation controls consist of eight control sequencers which regulate the preparation of all instructions to the computer. This preparation involves the transfer of instructions from the YR's to the ZR; the index fetching and address modification, if required; and the word boundary crossover test for VFL instructions. In addition to the sequencers, there are six supervisory type control triggers which condition the selection of the right or left half of the current YR and the corresponding half of the ZR. Instruction pre-decoding as to type of instruction and indexing requirements is stored in eleven supervisory control triggers. There are eight additional tag triggers associated with the two halves of the ZR to indicate the class of floating point instruction in Z. This permits the lookahead load controls to take over rapidly and load without a delay for decoding.

Preparation controls signal the IC controls when a YR is empty, and the lookahead load controls or I unit execution controls when the ZR has a prepared instruction. These controls are a completely independent set of hardware and operate simultaneously with the IC and lookahead load controls. Their function is basically to empty the YR's, and to prepare and fill the ZR as rapidly as possible in order to insure a high rate of instruction flow through the computer.

Lookahead Load Controls

The lookahead load controls consist of fourteen sequencers which regulate the loading of all I/O, VFL, and FP instructions into lookahead.

Five of these deal solely with the loading of VFL instructions and are in the form of a five stage execution timer. Every VFL load begins with the first stage and may then step to any one of the remaining four, so that every VFL instruction requires anywhere from two to five steps. Each sequencer loads a different lookahead level so that a VFL instruction may occupy two to five levels. If an operand address refers to an index address, the sequence is broken in order to fetch the index from a common index fetch sequencer, and control is returned to the VFL load sequencers.

There are two complete sets of four FP load sequencers for each half of the ZR. This was necessary to guarantee the fast switching and loading from the two halves of Z. Each set of four FP sequencers control the loading of one level and two level FP instructions, depending upon whether one or two operands are involved, as well as the actual fetching of the operands from either external memory, index storage, or internal register.

The prime consideration in the design of these controls was to achieve a high rate of transfer for FP instructions. It was necessary to "guess" that the operand address referred to main memory, and begin the fetch immediately. If the guess is wrong, then the fetch is blocked and the next cycle is controlled by another sequencer which fetches the operand from the correct memory.

The majority of FP instructions will require only one control step where the operand is fetched from main memory at the same time that the instruction is loaded into the single lookahead level.

During loading, the IC and the preparation controls may be operating simultaneously to maintain the instruction rate. The lookahead load controls signal the preparation controls when an instruction is completely loaded, and the instruction in the ZR can be replaced with a new one.

Instruction-Execution Controls

To execute the large number of I unit instructions and all their variations required the design of a large control system. The instructions were cate-

gorized by type, then divided into logical operations and analyzed for maximum sharing of common controls. Circuit limitations and packaging rules often prevented sharing as much as was desired. As the control logic grew, it became necessary to package it in two frames instead of one which, in turn, created a communication problem in critically timed areas. In order to keep the number of logical levels to a minimum, it was necessary to design a large amount of parallel logic into and out of each control stage.

The instruction set was divided into four categories: index arithmetic, branch, transmit, and miscellaneous operations. Each group was worked on separately, and the control sequences and logic required was designed. These controls were then compared for similarities, and wherever possible, common control logic was combined. The result was that sixty-one control sequencers and nine supervisory triggers, plus an operation decoder were required to execute the instructions in satisfactory performance times. This hardware occupied twelve standard panels and was packaged in two frames. These controls provide for fetching and checking operands from memory (main memory, index storage, or internal registers), and regulate index adder logical operations; partial and full word transfer, shift, and checking operations; as well as the loading of lookahead with a large variety of instruction formats and control tags. Most of these operations have a large number of input and output conditions whose combinations can cause each step to be performed in a wide variety of ways. Other requirements of these controls included the ability to stop immediately on errors, to be recoverable in case of no-opping and interrupt conditions, and to be able to step manually a cycle at a time. Every control stage has a line to the maintenance console for indication.

Miscellaneous Controls

These controls are used primarily in manual operations and special recovery routines. They consist mostly of supervisory control triggers (approximately 35) which initiate, condition, and terminate control sequences. The actual operation within the sequences are controlled by sequencers, which are shared for this purpose in the instruction execution area. For this reason the execution decoder must be blocked so as not to affect the stepping of the sequences for these special operations. Some special control functions are also handled in this area, such as store wait control, when lookahead contains a store to XS, IR, or MR; time clock operation triggers; I unit recovery because of program store test; and others. Some of these are very critically timed since they must block normal operation immediately, otherwise unrecoverable damage may be done. These controls are packaged in the IC control area so as to minimize the communication delay of the interlocks.

PERFORMANCE CHARACTERISTICS

The STRETCH computer was originally contracted for the Los Alamos Atomic

Energy Commission, and as such had certain performance goals which were particularly important to the customer. Of primary importance was the complete Floating Point operation, including the instruction fetching, preparation, operand fetching, and actual execution. The variable field length operation was desirable and attractive, but its performance need not be, nor could it be economically as high as FP. The index arithmetic instructions were all very important to the program indexing plans of the customer, and particular emphasis was placed on the execution of the count and branch instructions. Heavy reliance was to be placed on this type of instruction for controlling the many iterative program loops characteristic of scientific computing. Another general purpose type of instruction in which the customer showed special interest was transmit. This instruction would be used extensively in data and program manipulation, particularly in multi-programming operations. Conditional branch instructions were of particular interest since a decision had to be made as to which way the I unit should "guess" that the branch would go. This was required because the conditional data is not generally available at the time the I unit prepares the instruction. A study and program simulation was made to determine whether to assume the branch or not. The results did not clearly indicate an advantage either way, and depended largely upon how the program was written. It was decided to assume that the branch would not be successful, and if the assumption proves to be incorrect, then a recovery is made.

How the computer achieves the desired performance goals in these areas will follow. These examples are chosen because of the customer's special interest in them; and it does not mean to infer that all the other remaining operations were considered unimportant or are not of a comparable performance level. The high overall general purpose performance goals of the STRETCH computer[6] required that all the operations be executed in a much more powerful manner than on previous machines. To show how all the operations are performed, and the execution times, is not within the scope of this report.

Floating Point Instruction Preparation

A timing diagram showing the preparation of continuous FP instructions is shown in Figure 6. The preparation includes the fetching of instruction words from 2 μs memory, the ECC checking/correcting of the instruction, the Y to Z transfer and index fetch, the address modification, and finally the operand fetch and lookahead load. It can be seen how successive instruction fetches, preparations, and lookahead loads are overlapped to achieve a performance goal of one FP instruction every two cycles (one microsecond). Each instruction is assumed to require indexing. Otherwise instantaneous rates would achieve one FP instruction every half microsecond; the average rate, however, would still remain at one per microsecond. This is because the maximum rate of instruction fetches is balanced with the maximum indexing rate and the lookahead rates.

Figure 6 starts out by assuming a start-up operation (either program start or recovery operation) where two rapid successive fetches are made by the IC

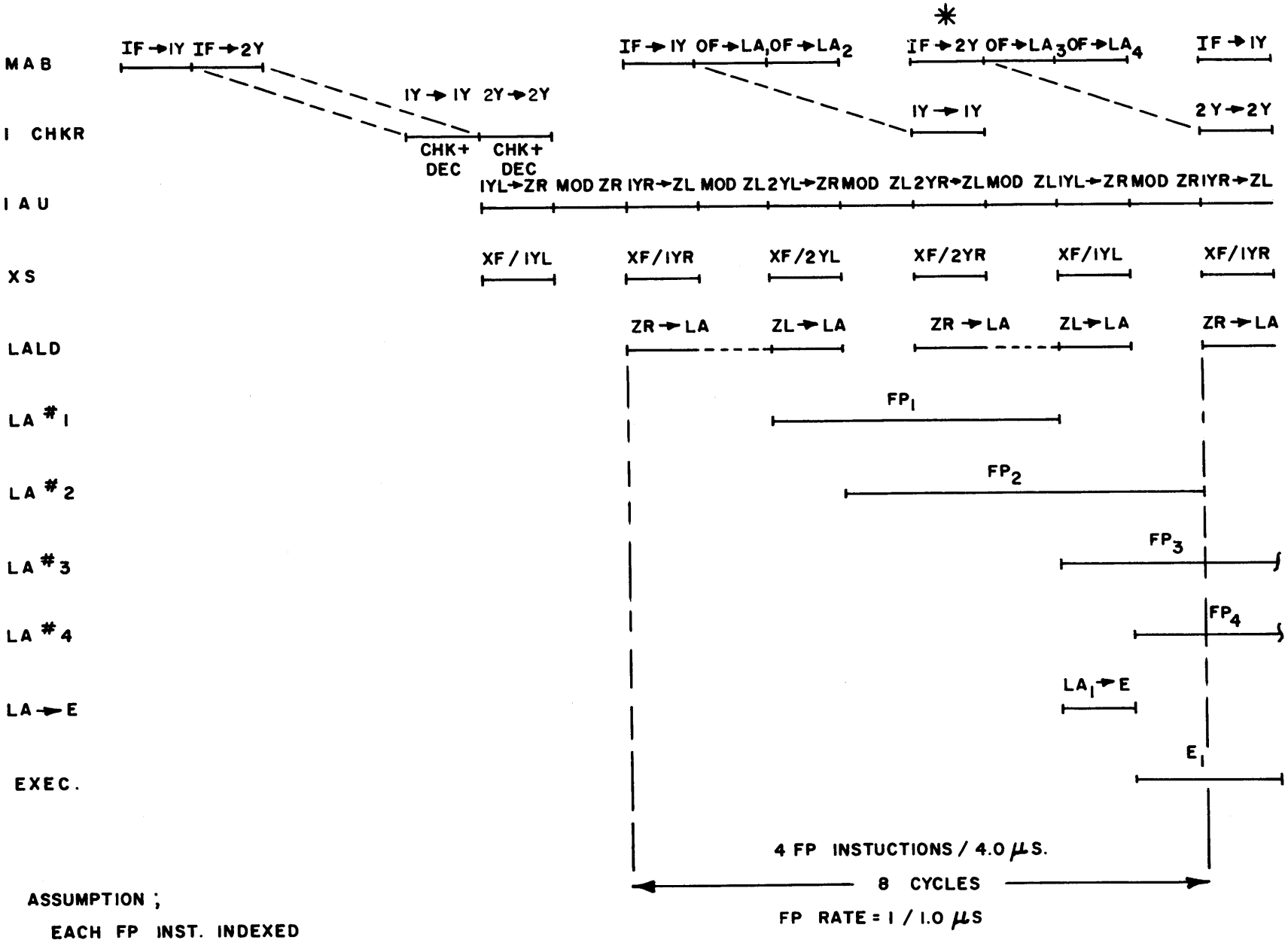


FIGURE 6. FP INSTRUCTION PREPARATION

to the Y registers. When the words are received from memory they are immediately checked, and here we are assuming no errors so an extra correct cycle is not required. During the check cycle the ECC is converted to parity and the preparation controls pre-decode the type of instruction. In this case each word contains two FP instructions. The next cycle is the transfer from the left half of 1Y to the right half of Z. The criss-crossing of half word locations was adopted as the simplest way of handling all the different combinations that occur. During this transfer, any addressed index word is fetched into the XR. Assuming that each instruction is to be indexed, the next cycle is the actual address modification. Here the index value in X is added through the index adder to the operand address in ZR. These last two cycles tied up the IAU so the next instruction in 1YR waited. The first instruction is now ready for loading into lookahead and fetching the operand. Since the IAU is not busy, we can also transfer the next instruction to Z and fetch its index word. So both operations are attempted. Since the 1Y register is now being emptied, the IC will try to fetch the next instruction word (IC + 3) into it. This conflicts with the operand fetch of the lookahead load cycle. Since IC controls have priority, the IC fetch is made simultaneously with the 1YR transfer to Z and the index fetch. The lookahead load cycle holds up its fetch during this cycle and completes it with the load operation on the following cycle, overlapped with the modification of the second instruction. This early instruction fetch guarantees that the 1Y register will be filled and checked by the time the 2Y register is emptied. In this manner the flow of instructions can be maintained. The degree of simultaneity in the I unit is best illustrated by the asterisked cycle which shows an instruction fetch to 2Y, a check cycle on 1Y, a 2 YR transfer to Z, an index fetch, and an attempted lookahead load, all occurring at the same time.

The degree of overlap in the computer is best shown by the last cycle, which shows the execution of the first instruction E1, three of the four lookahead levels filled with instructions 2 and 3, instruction 4 being loaded into lookahead and its operand being fetched, and instruction 5 being transferred from Y to Z and its index being fetched. This is but one variation of FP preparation. Other variations develop when the operand address refers to index storage or internal register; when a second operand is implied, as in multiply cumulative and load cumulative multiplicand; and in the many combinations of different types of instructions that may be intermingled with the FP instructions.

The diagram not only indicates how one performance goal is achieved, but shows also the control complexity required to interlock and execute efficiently the I unit instruction preparation function. It illustrates further how complete overlap of instruction fetching, indexing, and lookahead loading is achieved.

VFL Instruction Preparation

The preparation of a variable field length instruction is shown in the timing

diagram of Figure 7. Starting at the same point as Figure 6, it shows the sequence when the second instruction is a full-word VFL instruction located across memory word boundaries. In this case the left half of the first instruction word fetched is an FP instruction, and the right half is the left half of theVFL instruction. The left half of the second instruction word fetched contains the right half of the VFL instruction, while the right half of this word contains another FP instruction.

The first instruction is processed exactly as in the previous example. Notice that this time when we transfer the right half of 1Y to Z left we provide for the eventual correct alignment of the full-word instruction in Z. Again, any index word addressed by the half-word in 1YR is fetched into the XR. Before proceeding into the modification cycle, however, we must determine whether normal or progressive indexing is called for. This information is contained in the right half of the instruction, and is not available until the 2Y register has been filled, checked, and pre-decoded. In Figure 7, this is completed by the end of the 1YR to ZL transfer, so no wait is required. Assuming normal indexing, the next cycle is the modification cycle for the left half. If progressive indexing (PX) had been specified, a much different sequence would be required, completing the operation in two steps. The first one would replace the left half operand address with the value field of the index word, and the instruction preparation would continue on, and be loaded into, lookahead. Following the lookahead loading, a control sequence would be entered where the increment, count, and refill operations, if called for, would be executed. The results would then be loaded into lookahead as the second step of PX operations.

After normal modification of the left half, the right half is transferred from 2YL to ZR and its index fetch executed. The next cycle then modifies the right half of the instruction. The indexing and instruction operation codes are preserved during the modification. The next step is to determine if one or two memory words are required to obtain the operand field. The operand field could start at any bit position in a memory word and extend into the next, as long as the total field length is 64 bits or less. The word boundary crossover test (WBC) is done by adding, in the index adder, the length field to the full 24 bit operand address field. If a carry into position 17 of the operand address field occurs, then the instruction operand does, in fact, cross memory word boundaries. The first 18 bits of the result are gated into the WR, and this is actually the operand address for the second memory word.

This completes the preparation, and all that is left is to load the instruction into lookahead and fetch the operands. In this case three cycles are required. The first loads the instruction into one lookahead level. Since the data field is used for part of the instruction, no operand fetch can accompany this level. The next two cycles are for fetching the two operands of the instruction into two more levels of lookahead. The number of cycles required to load VFL type instructions varies from two to five, depending on whether the instruction

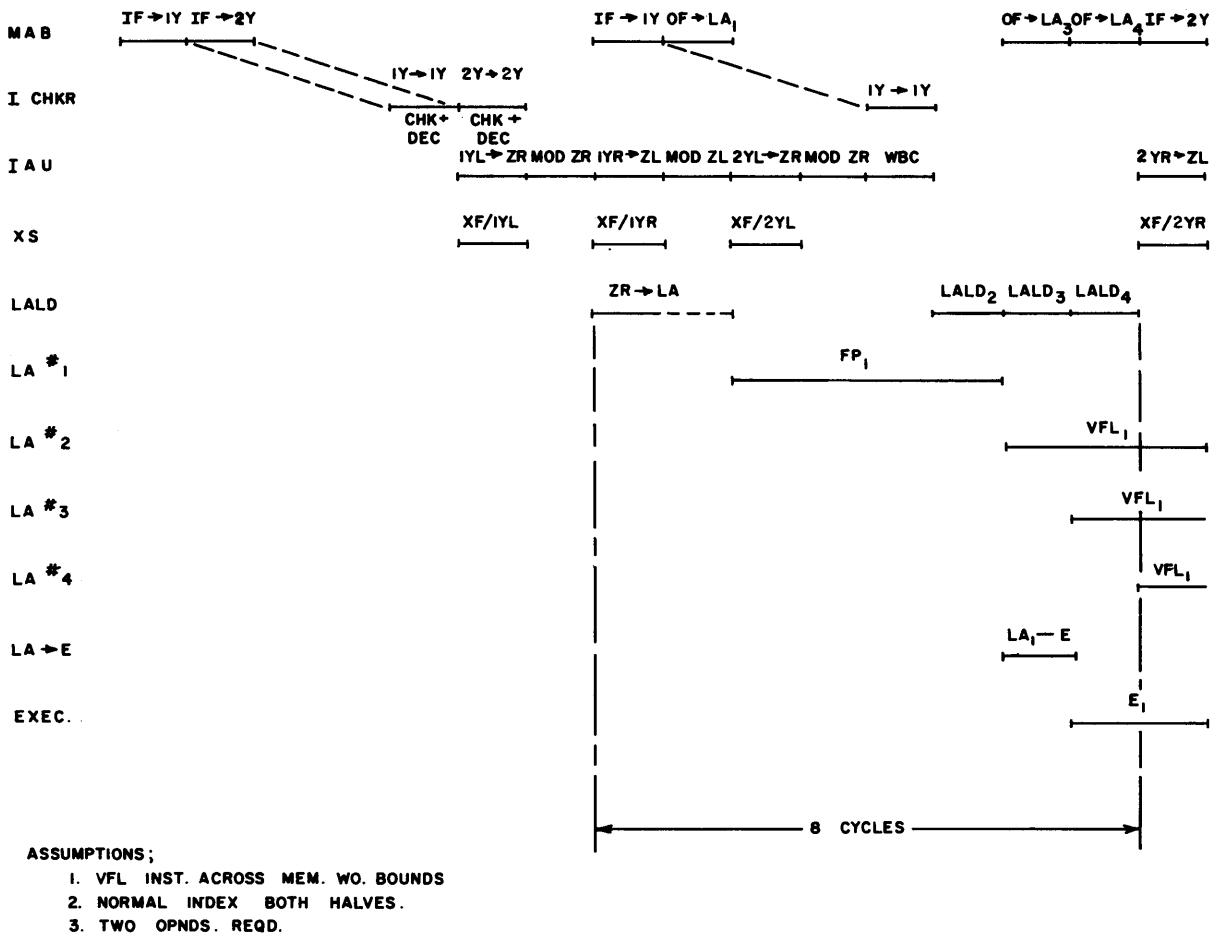


FIGURE 7. VFL INSTRUCTION PREPARATION

requires one or two operands, whether it is a fetch or a store to memory instruction, and whether any special registers are implied in addition to the operand address.

Having successfully loaded the instruction, the next half-word FP instruction can be transferred from 2YR to ZL and the normal FP preparation continued.

It can be seen from Figure 7 that the VFL instruction in this case took eight cycles or 4.0 microseconds to prepare and load lookahead, as compared to two cycles or 1.0 microsecond for a normal FP. If the instruction had no indexing specified and only one operand memory word was required, the operation would have taken only five cycles or 2.5 microseconds. Again, there are many variations of these instructions, depending on the type of indexing required; whether the instruction arrives straight or across memory words; whether the operands are in XS, EM, or IR; and whether it is a store to memory operation or not.

Count and Branch Execution

The execution sequence for a count and branch (CB) instruction is shown in Figure 8. Assuming a continuation of the FP preparation of Figure 6, the fifth instruction is defined as a CB instruction. In the example, it is assumed that the instruction requires indexing, and therefore it is possible to overlap the operation decoding with the modification cycle. Otherwise it would have been necessary to take a separate decode cycle. Once decoded, the operation enters an execution sequence controlled by the decoder outputs and by conditions arising out of the individual operations. The first cycle is a fetch of the index word whose count field will determine whether the branch is successful or not. In the next cycle, the fetch of the branch address instruction is initiated, and at the same time the count field of the index word in the XR is decoded for a XC = 1 condition. If this condition for branching exists, the fetch is completed; if the condition does not exist, the fetch is blocked. This permits as rapid a fetch of the next instruction as possible in order to begin filling up the Y registers with the new sequence of instructions. The instruction fetch is overlapped with the loading of the index word, before modification, into lookahead. This is referred to as a pseudo-store level, and is only used in recovery operations where the index registers have to restore to some previous level as a result of an interrupt or no-op condition. The next cycle then is the actual counting down through the index adder of the count field of the index word in the XR. After this, the value field can be advanced or diminished by one or one-half, if called for by the instruction. This is also effected through the index adder. At the same time, the advanced IC value (address of the instructions following the CB) is loaded into the same lookahead level as the pseudo-store. This is done because the next cycle destroys the IC contents by transferring into it the branch address from Z. If a no-op of this instruction is required, the I unit can continue on in the program. After the advance or diminish cycle, the updated index word is returned to XS. This is

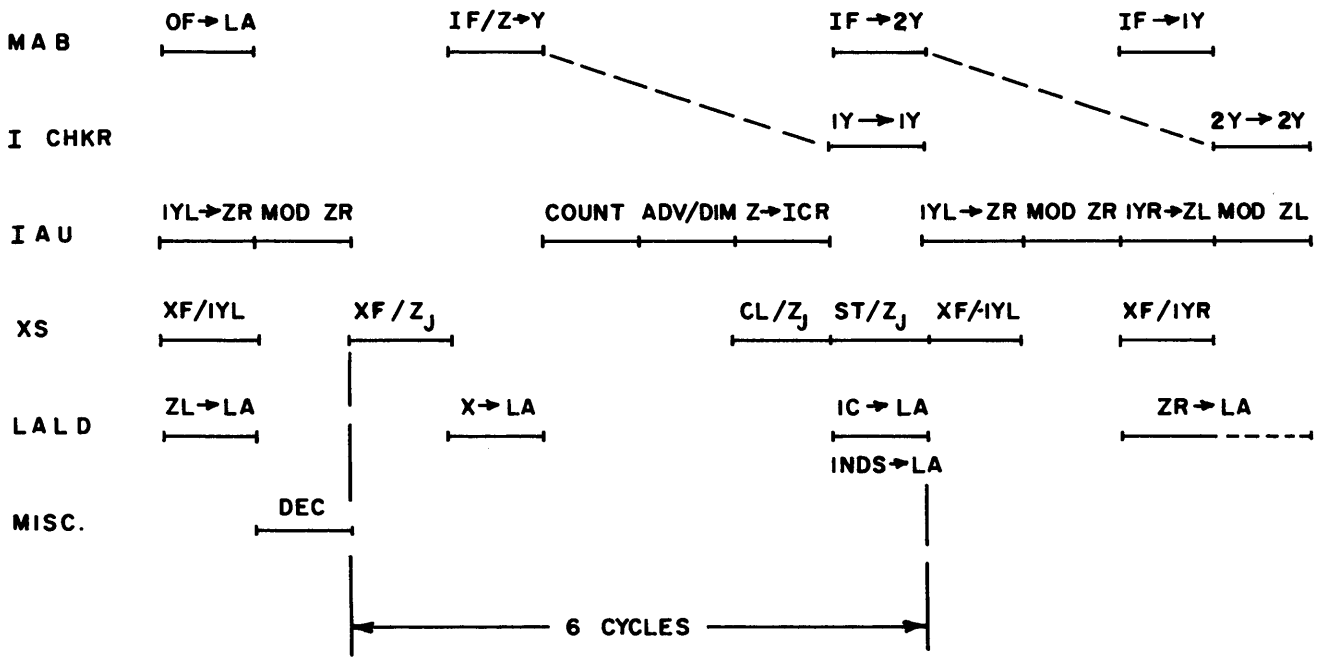


FIGURE 8. COUNT AND BRANCH EXECUTION

done by a clear cycle for resetting the word during the same cycle that transfers the branch address into the IC. It is followed by a store cycle which sets the contents of the XR into the previously reset word in the array. To complete the operation, all indicators associated with this instruction and the new IC value are loaded into lookahead to be tested later in proper instruction sequence for an interrupt. If a condition occurred which would no-op the instruction, then the new IC field is not loaded, only the indicators and a no-op tag. This would cause any branching already done to be cancelled by a recovery operation later.

Since the branch instruction was fetched early in the operation, the new instruction word has arrived and has been checked by the time the CB instruction is completely executed. The normal instruction preparation is restarted immediately and, in this case, the next FP instruction is loaded into lookahead four cycles later.

The entire execution of the instruction took six cycles equal to 3.0 microseconds, and is overlapped with the operand fetch and execution of instruction 1. Another way of saying this is that it took ten cycles or 5.0 microseconds from the completion of one instruction in the program to branch to another program location, and to prepare and load the first new FP instruction into lookahead.

Variations of this instruction execution sequence occur with respect to the branch condition 0 or 1; the advance or diminish modifiers; and the setting of indicators that may occur and cause a no-op or interrupt. In addition, there are other instructions in the same category which are more complicated and time consuming. These include the store instruction counter if count and branch (STICI-CB); and the count, branch, and refill (CBR) as well as the STICI-CBR variations.

Transmit Execution

An execution sequence diagram for the Transmit Direct instruction is shown in Figure 9. The instruction is a full word with each half being indexable. The "from" and the "to" address are located in the left and right halves respectively. The count of the number of words to be transmitted is contained in the count field of the index word, addressed by the J field of the instruction. The operation code can specify stepping the addresses forward or backward.

In the example the instruction is assumed to be contained entirely in the 1Y register, without crossing word boundaries. The left and right halves are then transferred to the ZR, and any indexing specified is done in the normal manner. As in the case of the count and branch instruction, the operation decoding is done in parallel with the indexing of the right half of the instruction, if called for. After decoding, the first cycle fetches the index word that contains the count of the number of words to be transmitted. Then the count field

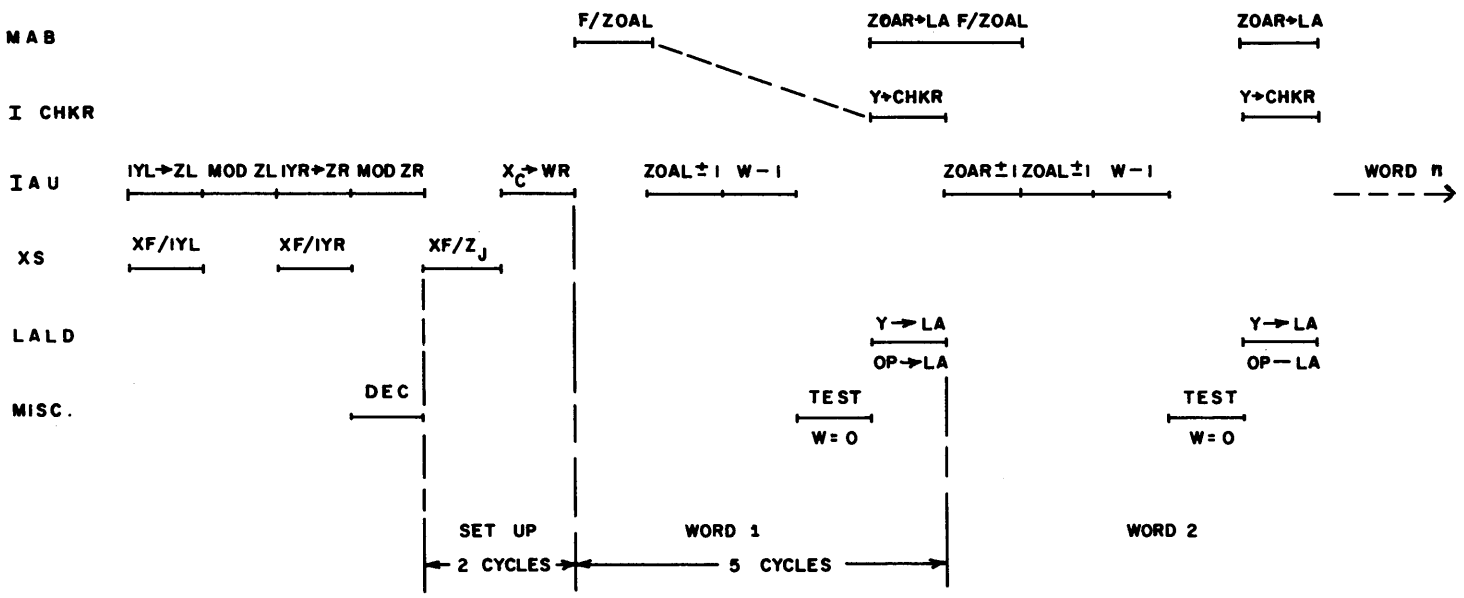


FIGURE 9. TRANSMIT EXECUTION

is transferred through the index adder from the XR to the WR, where the counting and the test for W = O is performed. The instruction is now set up for successive executions of individual word transmissions. In this case the from and to addresses are assumed to refer to external memory. The left operand address (from) is used to fetch the first word into a Y register. Then, the from address is stepped (+1 or -1) through the index adder. The next cycle subtracts one from the count field in the WR and puts it back in W. A d-c detect circuit indicates in the following cycle whether the count went to zero. By then the fetched word has been received in the Y register and is immediately loaded into lookahead through the I checker, completing the ECC check. The ECC code is left with the word instead of converting it to parity, since lookahead will immediately attempt to store it to memory. When loading the word into lookahead, the "to" operand address in the ZR is gated to the memory address-bus (MAB). This, in turn, goes to the "effective address register for storing" in lookahead. So by the end of the cycle the level contains the word to be transmitted, the store memory address, as well as sufficient operation and indicator tags to tell lookahead what to do with them. If an error or a data store indicator appears, the entire operation will be terminated at that point. Neither recovery nor no-opping of the instruction can be effected, since irrecoverable damage could have been done.

After loading the store level, the "to" address in ZR is stepped (+1 or -1) and the operation is repeated for the next word. The loop continues until the count goes to zero. When this happens, the instruction execution is complete and a final lookahead level is loaded with the advanced IC value and any final indicators.

It can be seen, from this example, that the rate of transmission is one word per five cycles or 2.5 microseconds. The initial set-up time took two cycles or 1.0 microseconds. This is only one of nine different variations of this instruction which can occur due to the combinations of the three different types of memory addresses in each of the two addresses. They all require a different sequence for execution, and therefore their transmission rates vary. In addition, there is the transmit immediate instruction which contains the count field within the instruction itself instead of in an index word. Corresponding to these two transmit instructions, there are the two swap instructions. These are similar except that the two operand address refers to two words in memory, which have to exchange locations. This is done by fetching both addresses and then reversing the addresses when storing. The counting and address stepping is done in the same manner as transmit.

Conditional Branch Operation

An example of a conditional branch operation where we "guess" that it will be unsuccessful is shown in Figure 10. The instruction is a branch on indicator, where the indicator specified is not an index result indicator. If it were, we would not have to guess but could test for the branch in the I unit's updated

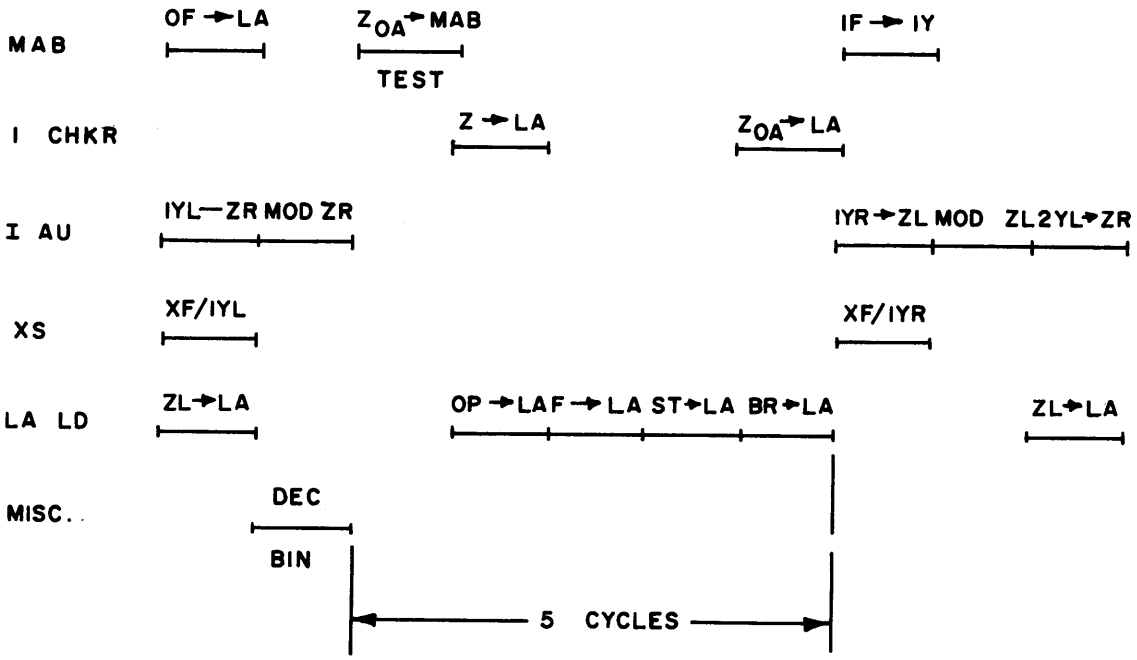


FIGURE 10. BRANCH ON INDICATOR EXECUTION

indicator register. However, in this case the test cannot be made until all preceding instructions in lookahead have been executed and their indicators set. Rather than wait the I unit tests the instruction for certain indicator conditions and then loads it into lookahead. This enables the VFL unit to select, test, and modify the specified indicator. The I unit continues in the program, assuming that in the majority of cases the branch is unsuccessful.

The diagram (Figure 10), begins as a continuation of the FP preparation in Figure 5, and FP instruction four (I_4) is prepared in the normal manner. Instruction five is the half-word branch on indicator instruction, and it is prepared and decoded in the identical manner as count and branch in Figure 8. Once decoded, the actual execution sequence begins with a test fetch cycle. This is not an actual fetch, it merely places the branch address on the MAB for boundary compare purposes and the setting of possible indicators. Having determined whether the branch address is legitimate or not, the instruction is then loaded into lookahead.

The loading is done in four steps and occupies four levels. The first is the instruction level with the operation codes and indicator address field, which tell the VFL unit what operation is required. The second cycle is the fetch level for fetching the indicator register to the lookahead level as the operand. The third level is a store to memory operand level with the store address of the indicator register. This notifies the VFL unit that after selecting, testing, and possibly modifying the indicator bit, the contents of the indicator register must be returned. The fourth and final level is the recovery level which contains the branch address, the IC value, and various operation and indicator tags. This is in case the branch test proves successful and a branch recovery operation is required. If that happens, the branch address is returned to the IC and all intervening operations are cancelled.

In the example, the branch is assumed unsuccessful and the I unit continues with the preparation of the following instruction (I_6). It can be seen that the actual execution of the instruction took five cycles or 2.5 microseconds. Again there are considerable variations to this type of instruction. It could have been a STICI-branch, where the instruction occupies a full-word and the left half contains the store instruction counter data. The branch could be conditional upon the indicator being on or off, and could specify resetting the indicator or leaving it alone. Another very similar instruction which can test any bit of memory for a branch condition is the branch on bit. This instruction is executed very similarly to the branch on indicator.

ACKNOWLEDGMENTS

To give individual credit to the many people who have contributed to the design of the instruction unit would be impossible. However, major design contributions were made by the following individuals: Mr. S.F. Anderson for the index adder and branch instruction execution controls; Mr. L. L. Headrick for the interrupt mechanism, index arithmetic, and transmit instruction execution controls; Mr. C. R. Holleran for the IC system and lookahead load controls; Mr. S. L. Lindauer for a portion of the data paths and the instruction preparation controls; and Mr. L. F. Winter for his assistance in the design of the data paths.

The overall engineering effort was under the supervision of Messrs. E. Bloch and R. E. Merwin.

REFERENCES

1. Erich Bloch, "The Engineering Design of the STRETCH Computer," Eastern Joint Computer Conference Proceedings, December, 1959.

2. W. Buchholz, "Selection of an Instruction Language," Western Joint Computer Conference Proceedings, May, 1958, p. 128.

3. F. P. Brooks, Jr., "A Program-Controlled Program Interruption System," Eastern Joint Computer Conference Proceedings, December, 1957, p. 128.

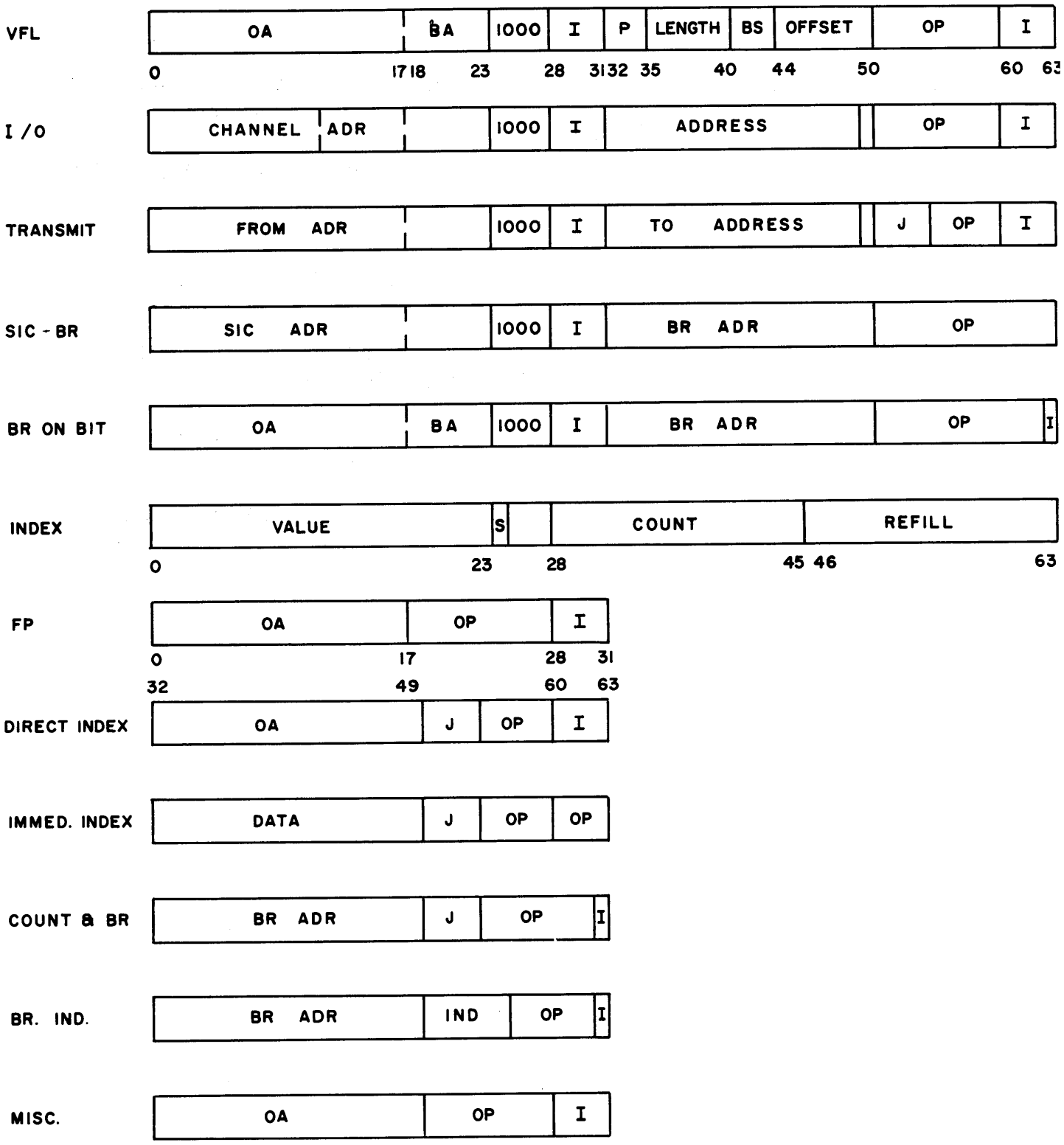
4. G. A. Blaauw, "Indexing and Control-Word Techniques," IBM Journal of Research and Development, July, 1959.

5. R. W. Hamming, "Error Correcting and Error Detecting Codes," Bell System Technical Journal, 1950, pp. 29, 147.

6. S. W. Dunwell, "Design Objectives for the IBM STRETCH Computer," Eastern Joint Computer Conference Proceedings, December, 1956, p. 20.

APPENDIX A

Instruction and Index Word Formats.



APPENDIX B

I Unit Instruction List

A. Direct Index Arithmetic

1. Load index .
2. Load value .
3. Load count.
4. Load refill.
5. Store index.
6. Store value.
7. Store count.
8. Store refill.
9. Add to value.
10. Add to value and count.
11. Compare value.
12. Compare count.
13. Rename.
14. Load value effective.
15. Store value in address.

B. Immediate Index Arithmetic

1. Load value immediate.
2. Load count immediate.
3. Load refill immediate.
4. Load value negative immediate.

5. Add immediate to value.

6. Add immediate to value and count.

7. Add immediate to value, count, and refill.

8. Subtract immediate from value.

9. Subtract immediate from value and count.

10. Subtract immediate from value, count, and refill.

11. Add immediate to count.

12. Subtract immediate from count.

13. Compare value immediate.

14. Compare value negative immediate.

15. Compare count immediate.

16. Load value with sum.

C. Unconditional Branching

1. Branch.

2. Branch relative.

3. Branch enabled.

4. Branch disabled.

5. Branch enabled and wait.

6. No operation

D. Indicator Branching

1. Branch on Indicator

Modifiers:

a) Leave indicator.
b) Set indicator to zero.
c) Branch if off.
d) Branch if on.

E. Bit Branching

1. Branch on bit

Modifiers:

a) Leave bit.
b) Invert bit.
c) Set bit to zero.
d) Branch if off.
e) Branch if on.

F. Index Branching

1. Count and branch

2. Count, branch, and refill.

Modifiers:

a) Branch if count non-zero.
b) Branch if count zero.
c) Leave value unchanged.
d) Add half to value.
e) Add one to value.
f) Subtract one from value.

G. Store Instruction Counter If

H. Transmit Operations

1. Transmit.

2. Swap.

Modifiers:

a) Forward
b) Backward
c) Direct count
d) Immediate count

J. Miscellaneous Operations

1. Refill.

2. Refill on count zero.

3. Execute.

4. Execute indirect and count

5. Store zero.

APPENDIX C

Memory Address Assignments

Location	Name	Length	Bit Address	Type
0	Zero	64	0 - 63	EM
1 P, a	Interval timer	19	0 - 18	XS
1 P, b	Time clock	36	28 - 63	XS
2 P	Interruption address	18	0 - 17	EM
3 P	Upper boundary	18	0 - 17	IR
3 P	Lower boundary	18	32 - 49	IR
3 P	Boundary control bit	1	57	IR
4	Maintenance bits	64	0 - 63	EM/MC
5 b	Channel address	7	12 - 18	IR
6	Other CPU	19	0 - 18	IR
7	Left zeros count	7	17 - 23	IR
7	All ones count	7	44 - 50	IR
8	Left half of accumulator	64	0 - 63	IR
9	Right half of accumulator	64	0 - 63	IR
10	Accumulator sign byte	8	0 - 7	IR
11 c	Indicators	64	0 - 63	IR
12 d	Mask	64	0 - 63	IR
13	Remainder	64	0 - 63	EM
14	Factor	64	0 - 63	EM
15	Transit	64	0 - 63	EM
16-31	Index registers XO - X15	64	0 - 63	XS
32-k	Normal external memory	64	0 - 63	EM

P — Permanently protected area of memory.

a — Read-only except for STORE VALUE, STORE COUNT, STORE REFILL, and STORE ADDRESS.

b — Read-only.

c — Bit positions 0 - 19 are read-only.

d — Bit positions 0 - 19 are always ones, and bit positions 48 - 63 are always zeros.

k — Last word address in a particular memory configuration.

IR — Internal Register

XS — Index Storage
EM — External Memory
MC — Maintenance Console

IBM®

Prepared by Laboratory Communications, Data Systems Division, Product Development Laboratory
International Business Machines Corporation, Poughkeepsie, New York Printed in U. S. A. 1959