BRANCH AT EXIT OPERATIONS

Branch-at-Exit instructions form the basic set which permits alteration of sequential execution of instructions.

To specify a change in the sequence (i. e., a branch) three decisions are required: (1) whether or not the branch is to be taken, that is, the condition determination; (2) when the branch is to be taken, the exit point specification; and (3) the address to which the branch is to be made, the effective address calculation.

Condition Determination

The conditional Branch-at-Exit instructions have the long format:

op	i	j	k	h

The i- and j-fields designate the bits of the condition register used to determine whether or not the branch is taken. The k-field designates an X-register which with the literal h-field is used to compute the effective branch address.

Whether or not the branch is to be taken is computed as a function of two bits selected from the condition register c (special register S^0). The i- and j-fields select the bits of c; the function which is computed is specified by the operation code. If the value of the function is TRUE (1), the branch is called successful and the alteration of sequence is effected at the next EXIT instruction. If the value of the function is FALSE (0), the branch is called unsuccessful and no alteration of sequence occurs.

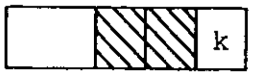
Eight functions can be specified:

$c_i \wedge c_j$	$c_i \vee c_j$
$c_i \wedge \bar{c}_j$	$c_i \vee \bar{c}_j$
$\bar{c}_i \wedge \bar{c}_j$	$\bar{c}_i \vee \bar{c}_j$
$c_i = c_j$	$c_i \neq c_j$

A branch controlled by a single bit may be specified by setting j equal to i. An unconditional branch may be specified by the true function $c_i = c_i$ for any i.

If any of the (non-existent) condition bits 24 through 31 is addressed, the bit value 0 is used.

There is a single unconditional Branch-at-Exit instruction which has the short format:



The i- and j-fields of this instruction are ignored, and the condition value TRUE is used so that this branch is always successful.

Exit Point

The sequential nature of instruction execution is not altered by the Branch-at-Exit instruction itself. Rather, the branch point is marked by an EXIT instruction, and, when a branch is successful, the actual alteration of instruction flow occurs at the EXIT. Instructions between the branch instruction and the EXIT are executed normally, independent of whether the branch is successful or unsuccessful.

When two or more branch instructions occur without an intervening EXIT, the branch instructions are examined in order. The first branch which is successful governs the next EXIT; the other branch instructions which follow the successful branch but preceed the EXIT are ignored. The set of branch instructions which relate to a single EXIT need not be in adjacent storage locations but may be interspersed with other instructions (except EXITs).

If an EXIT occurs without a successful branch having been executed since the last previous EXIT, the instruction flow continues in a sequential manner.

Effective Branch Address

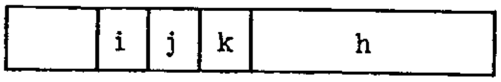
The effective branch address, eba, designates the location of the instruction to which the instruction execution sequence will be altered if the branch is successful. The point of alteration is determined by an EXIT instruction.

The eba may be specified in either of two ways: in the 24-bit unconditional branch instruction eba is given directly by the contents of index register k; in 48-bit instructions eba is the modulo 2^{24} sum of index register k and the 24-bit literal field of the instruction.

instruction format	eba calculation
short	eba $\leftarrow X^k$
long	eba $\leftarrow X^k + h$

If the branch is successful and if the eba designates a missing address, at the next EXIT exception bit MI is set to 1 and the program is interrupted (see the section on Sequencing for further details). If the branch is unsuccessful, no exception can occur.

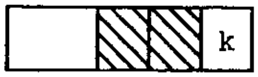
Branch at Exit, Conditional



mnemonic	function
BAND	$c_i \wedge c_j$
BTAF	$c_i \wedge \bar{c}_j$
BFAF	$\bar{c}_i \wedge \bar{c}_j$
BOR	$c_i \vee c_j$
BTOF	$c_i \vee \bar{c}_j$
BFOF	$\bar{c}_i \vee \bar{c}_j$
BEQ	$c_i = c_j$
BXOR	$c_i \neq c_j$

Exceptions: none

Branch at Exit, Unconditional



mnemonic	function
BU	identically TRUE

Exceptions: none

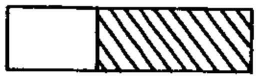
Exit Operations

An EXIT instruction serves to mark a branch point, where one sequential pattern of instruction execution terminates and another sequential pattern begins.

Two exit operations are provided. The EXIT instruction serves only to designate a branch point. The EXITL instruction does three functions in the following logical order: it sets the skip state to "not skipping", it performs the function of the MLX instruction, and it designates a branch point.

A branch point designation cannot be skipped. Thus, if an EXIT instruction is flagged as skippable, the flag is ignored. If an EXITL is flagged, its first two functions may be skipped but the branch point designation may not.

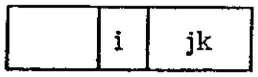
Exit EXIT



The branching action for any previous branch instruction occurs at the point designated by this instruction.

Exceptions: none

Exit, Save Location and Stop Skipping EXITL



This instruction is logically identical to the three instructions:

```
SKTAF  31,31
MLX    i,jk
EXIT
```

Exceptions: none

Skip Operations

Skip operations provide the ability to inhibit the execution of a set of instructions following the skip instruction. The skipping action is conditional on a function of two bits of the condition register; the instructions to be skipped are indicated by a special bit in the operation code. Thus, to specify a skip two parameters are required: (1) whether or not the skip is to be made, the condition determination; and (2) which instructions are to be skipped, the skip scope.

Condition Determination

Whether or not the skip is to be taken is computed as a function of two bits selected from the condition register c (special register S^0). The i- and j-fields select the bits of c; the function which is computed is specified by the operation code. If the value of the function is TRUE (1), the skip is called successful and the flagged instructions within the scope of the skip will be ignored. If the value of the function is FALSE (0), the skip is called unsuccessful and instructions within the scope of the skip are executed normally.

Eight functions can be specified:

$c_i \wedge c_j$	$c_i \vee c_j$
$c_i \wedge \bar{c}_j$	$c_i \vee \bar{c}_j$
$\bar{c}_i \wedge \bar{c}_j$	$\bar{c}_i \vee \bar{c}_j$
$c_i = c_j$	$c_i \neq c_j$

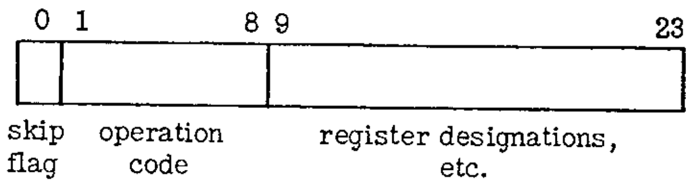
A skip controlled by a single bit may be specified by setting j equal to i.

If any of the (non-existent) condition bits 24 through 31 are addressed, the bit value 0 is used.

It will be noted that skip condition is determined exactly the same as the branch-at-exit condition.

Scope of the Skip

The scope of a skip instruction is those instructions between the SKIP and the next SKIP instruction which is not skipped. Those instructions within the scope which may be skipped are designated by setting a special bit in the instruction to 1. One bit position in the operation code of all instructions is designated as the skip flag; it is bit number 0 in the format which is common to all instructions:



The mnemonic means of designating an instruction with its skip flag set to is to proceed the instruction's menmonic by an asterisk (*).

If the skip condition is TRUE, all instructions within the scope with this skip flag set to 1 are ignored.

If the skip condition is FALSE, all instructions within the scope are executed normally (independent of the value of their skip flag).

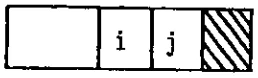
The instructions within the scope of a SKIP which are designated as skippable by having their skip flags set to 1 need not be in adjacent storage locations. They may be interspersed with other unflagged (and hence unconditionally executed) instructions.

All instructions except an EXIT instruction may be flagged as skippable. In particular a skip or branch instruction may be skipped.

The skip state (i. e., "skipping": ignore flagged instructions, or "not skipping": execute all instructions) is altered only as shown in the following table:

Instruction	New Skip State
SKIP	determined by condition determination
EXITL	not skipping
SVC, IC	not skipping
SVR, IC	determined by bit S_6^{11}
SCAN	determined by scan-in data

Skip



menmonic	function
SKAND	$c_i \wedge c_j$
SKTAF	$c_i \wedge \bar{c}_j$
SKFAF	$\bar{c}_i \wedge \bar{c}_j$
SKOR	$c_i \vee c_j$
SKTOF	$c_i \vee \bar{c}_j$
SKFOF	$\bar{c}_i \vee \bar{c}_j$
SKEQ	$c_i = c_j$
SKXOR	$c_i \neq c_j$

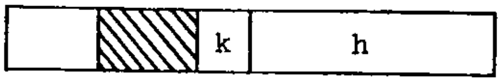
Exceptions: none

Special Purpose Branch Instructions

The special purpose branch instructions are included primarily for use in interruption servicing routines and for changing the status of the supervisory-problem mode. These situations require special treatment because the concurrency of operation in the MPM creates circumstances not normally encountered in a non-overlapped computer. To treat these situations without these instructions would be both awkward and excessively time consuming.

Many instructions in this class are essentially unconditional branch instructions. The formation of the effective branch address is different for each instruction. However the point at which the branch is to occur is marked by an EXIT, as usual.

<u>Invalidate Instruction Buffers and Branch</u> IVIB



At the next EXIT the contents of all instruction buffers are invalidated. Any instructions which had been prefetched into the instruction buffers and any instructions in the dispatch registers or contender registers following the EXIT are fetched from storage again.

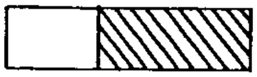
For the branching action, IVIB appears as a successful branch instruction. That is, unless there is an outstanding successful branch instruction, a branch occurs at the next EXIT to the location designated by the effective branch address, eba. The eba is calculated as

$$eba \leftarrow X^k + h.$$

If a successful branch is outstanding, all IVIB functions are suppressed.

Exceptions: none

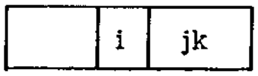
Pause

PAUSE 

The execution of all instructions preceding PAUSE are completed. Any interruptions occasioned by these instructions are also taken. After all these actions are accomplished, the execution of the next instruction in sequence is begun.

Exceptions: none

Pause with Exception

PI 

Index register X^i is replaced by the value specified by the 10-bit literal jk-field. Before the replacement the 10-bit quantity is extended to 24 bits by appending 14 high order bits equal in value to the high order bit of the jk-field. Also the PI exception bit is set to 1. Then a PAUSE is executed, so that an interruption is taken before the execution of the next instruction in sequence is begun.

Exception	Exception bit
always set	PI

Supervisor Call SVC

	i	jk

If there are no outstanding successful branch instructions, SVC is performed as follows:

1. Index register X^i is replaced by the value specified by the 10-bit literal jk-field. Before the replacement the 10-bit quantity is extended to 24 bits by appending 14 high order bits equal in value to the high order bit of the jk-field.

At the next EXIT the following are also performed:

2. The current values of the MPM mode bits $S^{11}_{0,1,2}$ replace the values of the bits $S^{11}_{13,14,15}$.

3. The MPM is placed in the following mode:

   a. supervisory
   b. concurrent

(Note that the disable/enable mode is not altered.)

4. The internal branch-skip-MPC state is saved in bits 3 through 9 of S^{11}. (Note that the recorded branch state is always "no outstanding branches".)

5. The internal branch-skip-MPC state is set as follows:

   a. no outstanding branches
   b. not skipping
   c. no carry

6. A branch is taken to fixed location 256 with respect to the supervisory normal key.

The setting of the mode bits is interlocked with the execution of other instructions to give the effect of sequential execution, so that concurrency problems associated with the entrance to the supervisory mode are avoided.

If a successful branch is outstanding, all SVC functions are suppressed.

Exceptions: none

Supervisor Return SVR

	▨	k	h

If there are no outstanding successful branch instructions, SVR is performed as follows at the next EXIT:

1. The MPM mode bits $S^{11}_{0,1,2}$ are set to the values $S^{11}_{13,14,15}$.

2. The internal branch-skip-MPC state is set to the values designated by bits 3 through 9 of the machine state register S^{11}. This setting of the branch state neither effects nor is effected by the branching action of step 3.

3. A branch is taken to the address designated by the eba where

$$eba \leftarrow X^k + h.$$

The branch is with respect to the normal key of the mode specified by S^{11}_{13}.

The setting of the mode bits is interlocked with the execution of other instructions to give the effect of sequential execution, so that concurrency problems associated with the return are avoided.

If a successful branch is outstanding, all SVR functions are suppressed.

Exception	Exception bit
in problem mode	PV

Interrupt Call IC ▨

The interrupt call instruction is internally generated and inserted into the instruction stream to effect an interruption. IC is not available for use as a programmed instruction. IC is performed as follows:

1. The current values of the MPM mode bits $S^{11}_{0,1,2}$ replace the values of bits $S^{11}_{10,11,12}$.
2. The MPM is placed in the following mode:
   a. supervisory
   b. disabled
   c. concurrent
3. The interruption return address register S^9 is set to the address to which a return should be made in order to resume the interrupted program in its proper logical sequence.
4. The internal effective branch address is saved in register S^{10}.
5. The internal branch-skip-MPC state is saved in bits 3 through 9 of register S^{11}.
6. The internal branch-skip-MPC state is set as follows:
   a. no outstanding branches
   b. not skipping
   c. no carry
7. A branch is taken to fixed location 0, with respect to the supervisory normal key.

IC is interlocked so that it is executed in strict sequence with each stream; that is, it cannot pass any instructions ahead of it (instructions in the program being interrupted), nor can it be passed by any instructions behind it (instructions in the program at location 0).

Interrupt Return IR

If there are no outstanding successful branch instructions, IR is performed as follows at the next EXIT:

1. The MPM mode bits $S^{11}_{0,1,2}$ are set to the values of $S^{11}_{10,11,12}$.

2. The internal branch-skip-MPC state is set to the values designated by bits 3 through 9 of the machine state register S^{11}.

3. The internal effective branch address is set to the value of S^{10}.

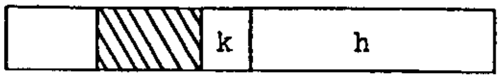
4. A branch is taken to the address designated by the interruption return address register S^9. The branch is with respect to the normal key of the mode specified by S^{11}_{10}. This branching action neither effects nor is effected by the actions of steps 2 and 3.

The setting of the mode bits and the branch-skip-MPC state are interlocked with the execution of other instructions to give the effect of sequential execution, so that concurrency problems associated with the return are avoided.

If a successful branch is outstanding, all IR functions are suppressed.

Exception	Exception bit
in problem mode	PV

Scan In

SCAN 

The MPM registers and control triggers are reset to state specified by the contents of storage starting at the effective address ea, where

$$eal \leftarrow X^k + h$$

$$eak \leftarrow \text{alternate key}$$

The nine low order bits of ea are ignored and assumed to be 0's. Thus the scan data is assumed to be aligned on a 256-word boundary.

The storage arrangement of the registers and triggers is specified in the section "MPM Interruptions".

After completing SCAN, execution is resumed according to the state specified by the scanned-in data.

Exceptions	Exception bit
in problem mode	PV
missing address	MA