Loop —A conditional Branch, condition met, in the stack.
Jump —An unconditional Branch, or condition Branch, condition met, not in the stack.
No Branch—A conditional Branch, condition not met.

To test for the destination in the Stack, the contents of the program address register, P, must remain set equal to the address which contains the Branch instruction. Therefore, P is not changed until after the "third parcel" instruction is issued, that being the last possible location of a 30-bit branch instruction in a word.

An additional problem of the Branch is the condition of the issue control mechanism following a "fall-through," or no-branch, condition. Instructions, following the Branch instruction in the same instruction word, are held in the registers U1 and U2 after the issue of the Branch instruction. Then, if the branch condition is not met, these instructions are brought to ISSUE in the normal manner. As a result, the control over the instruction stack output must track directly with the ISSUE command.

D. SCOREBOARD

A unique and essential part of the 6600 Central Processor control is the Unit and Register Reservation Control, or the Scoreboard. What is intended by this design is the simultaneous operation of functional units on a single instruction stream. Many operations in these units are quite independent of others, due to the relative simplicity of the instructions. It is often particularly apparent that a sequence of arithmetic or logical operations can be executed simultaneously with a sequence of control or housekeeping operations. Again, examples will be shown in which considerable overlap is possible even in the single sequence.

One major premise of the Scoreboard design is that each new instruction be issued to its functional unit as early as possible in order to allow following instructions to be issued. In some cases, an issued instruction may be held up after issue awaiting input operands, while a following instruction may proceed without restraint.

Three types of conflict can be described in the usage of functional units and registers, which must be resolved by the Scoreboard.

1. First Order Conflict
   This is a conflict between instructions which require the same functional unit or the same result registers.

   Example one. Functional unit conflict

   X6 = X1 + X2
   X5 = X3 + X4

Both instructions use the Add functional unit, a situation in which the second instruction must wait for the first to be completed.

In the case of multiply or increment instructions, two units are provided reducing the probability of this conflict.

Example two. Result register conflict

```
X6 = X1 + X2
X6 = X4 * X5
```

Both instructions call for register X6 for the result, another situation in which the second instruction must wait for the first to be completed. Although the example shown is a trivial case, it will be seen in later discussion that many nontrivial cases are possible.

The control over this conflict is simply that of not issuing the second instruction until the first is completed. At issue time, the condition must be determined early enough to stop the ISSUE command.

2. Second Order Conflict
This conflict occurs when an instruction requires the result of a previously issued, and as yet uncompleted, instruction as a source or input operand.

Example:

```
X6 = X1 + X2
X7 = X5/X6
```

Register X6 in this example is used as the result of the Add instruction and then as the divisor in the Divide instruction. The second instruction is issued but held in the Divide Unit until result X6 is ready.

The second order conflict does not halt issuing of instructions but is resolved by the scoreboard control over the functional unit.

3. Third Order Conflict
This conflict occurs when an instruction is called on to store its result in a register which is to be used as an input operand for a previously issued, but as yet unstarted, instruction.

Example:

```
X3 = X1/X2
X5 = X4 * X3
X4 = X0 + X6
```

In this example the third order conflict on the use of register X4 is a direct result of a second order conflict on register X3. Because the instructions are issued on consecutive minor cycles and because the

Add function is much faster than Divide or Multiply, the addition is accomplished and ready for entry in the result register X4 well in advance of the start of Multiply. The second order conflict on register X3 causes the Multiply to hold until that input operand is ready. This holds up the entry of register X4 into the Multiply Unit also.

Third order conflicts are resolved by holding the result in the functional unit.

Scoreboard control thus directs the functional unit in starting, obtaining its operands, and storing its result. Each unit, once started, proceeds independently until just before the result is produced. The unit then sends a signal to the Scoreboard requesting permission to release its result to the result register. The Scoreboard determines that the path to the result register is clear and signals the requesting unit to release its result. The releasing units reservations are then cleared, and all units waiting for the result are signaled to read the result for their respective computations.

DESIGNATORS

The Scoreboard gets its name from the number of designators and identifiers used in performing the job of reservation control. Figure 75 on page 129 diagrams the number of designators associated with one functional unit. Shown is the Add Unit which is given the number 17 with function designators, reservation identifiers and flags as described below.

Fm	—Function to be performed (ADD)
Fi	—Designates register Xi for result
Fj	—Designates register Xj as addend
Fk	—Designates register Xk as augend
Qj	—Identifies the functional unit, by number, producing a result to be used as addend
Qk	—Identifies the functional unit, by number, producing a result to be used as augend
Read Flag j	—A single-bit flag indicating that the addend is ready
Read Flag k	—A single-bit flag indicating that the augend is ready
Xi	—Identifies that the Add Unit, number 17, has reserved register Xi for its result. (Bi and Ai for other units)

All functional units are assigned a number to be used in the identifiers Q for the units, X for the operand registers, B for the increment registers, and A for the Address registers. These numbers are assigned as follows:

Designator (Octal)	Functional Unit
00	Branch
01	Increment 1
02	Increment 2
03	Shift
04	Boolean
05	Divide
06	Multiply 1
07	Multiply 2
10	—
11	Read Storage Channel 1
12	Read Storage Channel 2
13	Read Storage Channel 3
14	Read Storage Channel 4
15	Read Storage Channel 5
16	Fixed Add
17	Add

The Scoreboard operation is described in two parts; first, placing reservations, and second, directing the read operand and store result operations of each unit.

PLACING RESERVATIONS

This portion of the Scoreboard operation is executed in four sequential steps at the time an instruction is issued. These steps are as follows.

1. Reserve the functional unit, Set its "busy" flag, and enter the operating mode (fm).
2. Set the register designators in the functional unit, Fi, Fj and Fk.
3. Enter any previous result reservations on the entry operands, Qj and Qk.
4. Set the result register identifier, Xi, Bi, or Ai with the functional unit number.

Step one, SET UNIT BUSY, is rather straightforward except as the determination of unit "busy" is made. As an example, two consecutive instructions to the same unit must be handled such that the second instruction ISSUE is disallowed. Since these are one minor cycle apart, the setting of unit "busy" flag by the first instruction followed by the test for busy by the second instruction must be accomplished in one minor cycle.

Step two, SET F, transfers the i, j, and k fields of the instruction to the designators of the functional unit. These are then used to designate operand and result registers to be used by the unit. Figure 76 shows how these designators are transferred from U1 to U2 and then to the respective functional units.

Notice that the Branch instructions cause a right shift of the designators i and j in register U1 to j and k respectively in register U2. This ma-

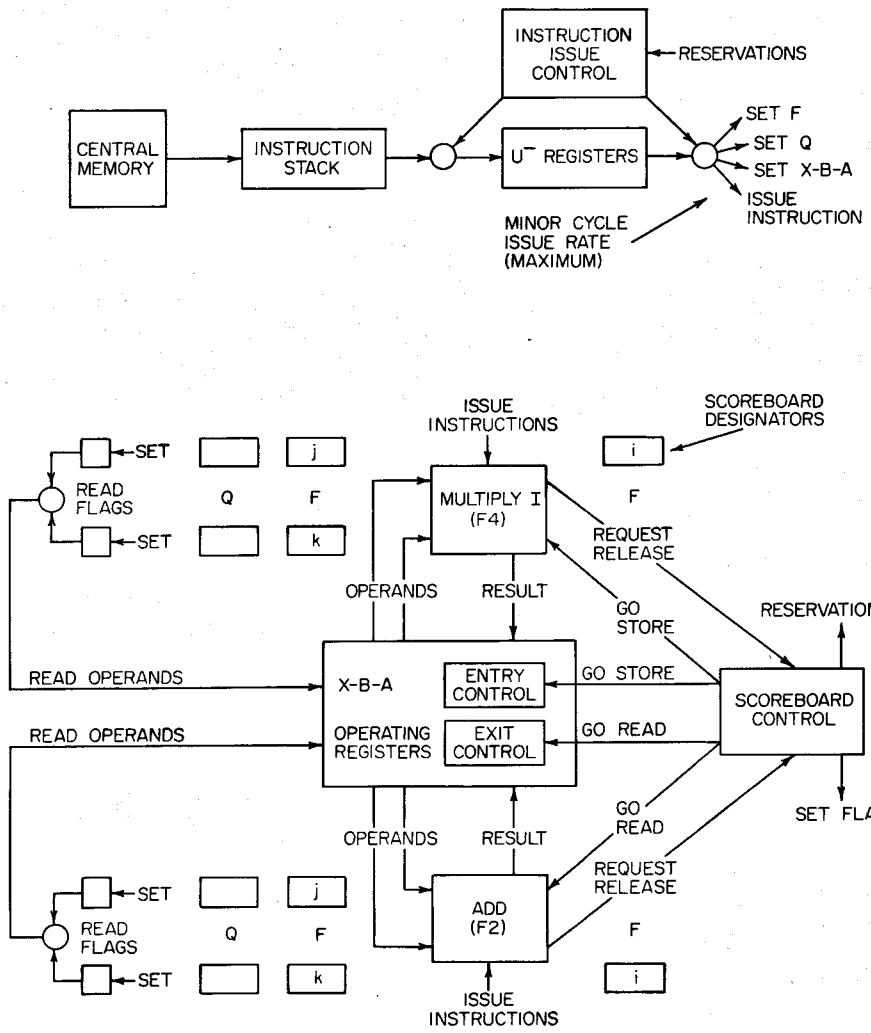


FIGURE 75 Reservation designators.

neuver is convenient to allow a direct usage of the Increment and Fixed Add Units as partner units to the Branch unit for conditional branch instructions.

Step three in placing reservations, SET Q, is essentially a copying operation from one of the 24 XBA identifiers related to the 24 operating registers. The identifier contains the functional unit number of the unit which has reserved that register for a result. Since there are usually two Q identifiers, one for each input operand, there may be two independent settings. See Figure 77. Following this step, the essential link between a previous result and an input operand is established.

Step four, the final step in placing reservations, SET XBA, places the

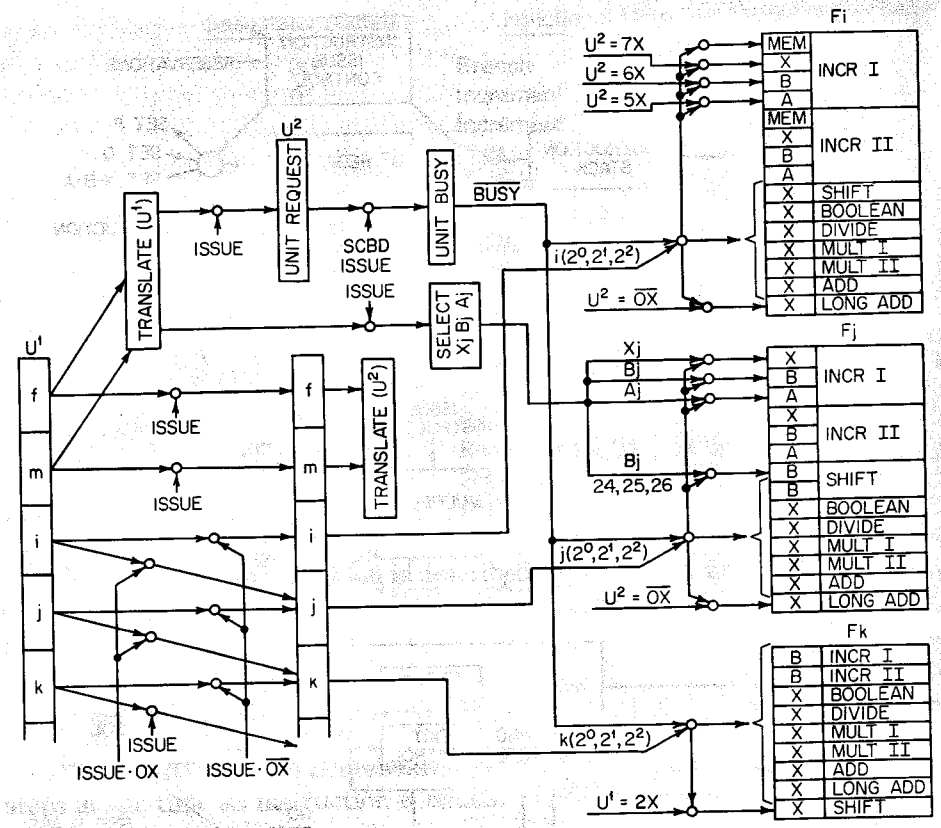


FIGURE 76 Set F

functional unit number in the identifier associated with the result register. Translations of the function to be performed are necessary in order to select the correct register group, X, B or A, along with the correct register in the group. Note that the unit numbers were chosen such that only two bits are necessary for the B and A registers, whereas four bits are needed for the X registers. Only three units cause results in B and A registers, whereas up to ten "units" cause results in X registers. This, of course, includes the Read Storage channels into registers X1 through X5. The unit number generator produces the necessary unit numbers to be entered. In the case of Read Storage instructions, which produce a new result in registers A1 through A5, the A identifier is set with the Increment Unit number, and the partner X identifier is set to the Read Storage Channel number. See Figure 78.

SET READ FLAGS

After issuing the instruction and placing reservations, the Scoreboard proceeds to control the functional unit in reading operands and storing re-

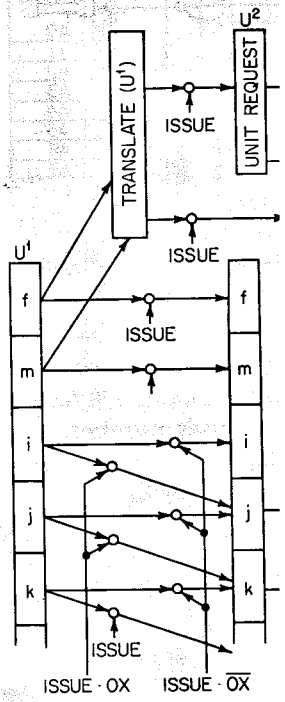
sults. The first activity
of input operands. The
be read. Both Read Fla

The conditions for
fier associated with tha
functional unit identifie
operation with the inpu
second order conflict is

X6

X7

When the second i
tions, SET Q, causes the
the Qk identifier for the
Add Unit, having been p
When the Add Un



sults. The first activity in the functional unit is the simultaneous "reading" of input operands. The unit may not start until both operands are ready to be read. Both Read Flags must, therefore, be set.

The conditions for setting a Read Flag are determined by the Q identifier associated with that input operand and by the Release signal from the functional unit identified by Q. The effect is to link the result of the previous operation with the input operand. The example used in the description of second order conflict is repeated here to show the effect.

X6 = X1 + X2
X7 = X5/X6

When the second instruction is issued, the third step in placing reservations, SET Q, causes the unit number found in identifier X6 to be placed in the Qk identifier for the divisor. The unit number is, of course, 17 for the Add Unit, having been placed there at the time of issue of the first instruction.

When the Add Unit requests release and receives permission, a Release

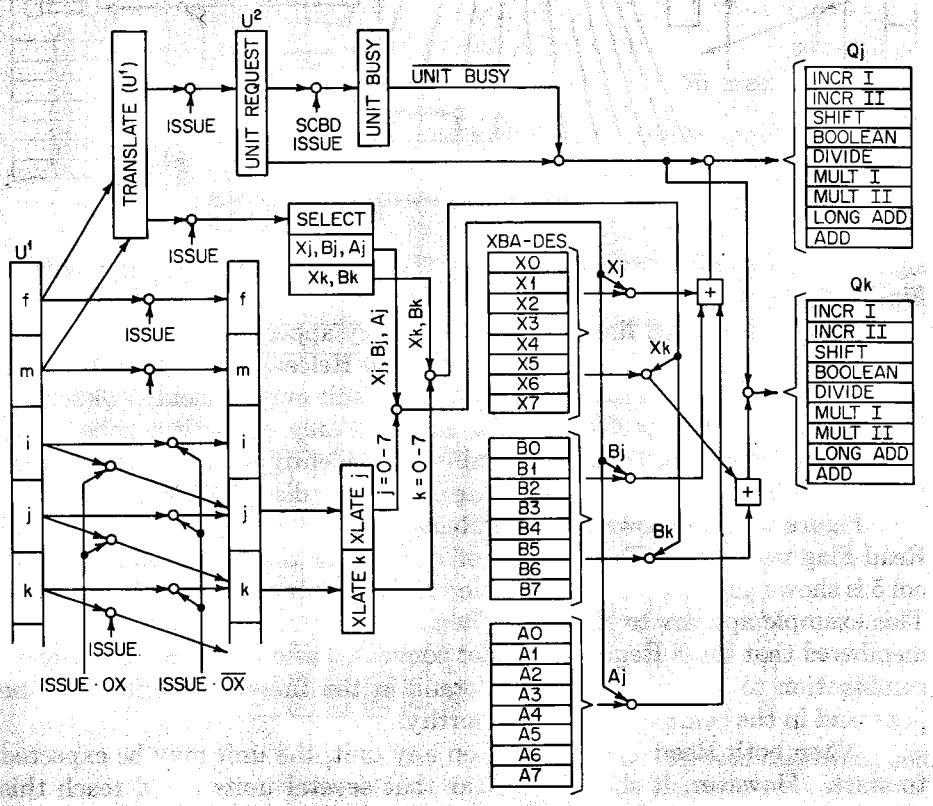
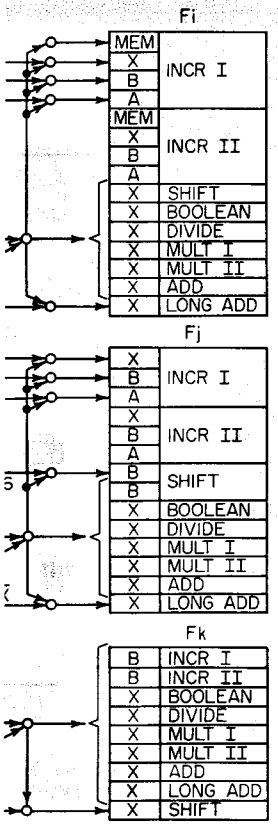


FIGURE 77 Set Q



ith the result register.
ssary in order to select
correct register in the
ı that only two bits are
s are needed for the X
egisters, whereas up to
ırse, includes the Read
unit number generator
In the case of Read
gisters A1 through A5,
ber, and the partner X
. See Figure 78.

ations, the Scoreboard
erands and storing re-

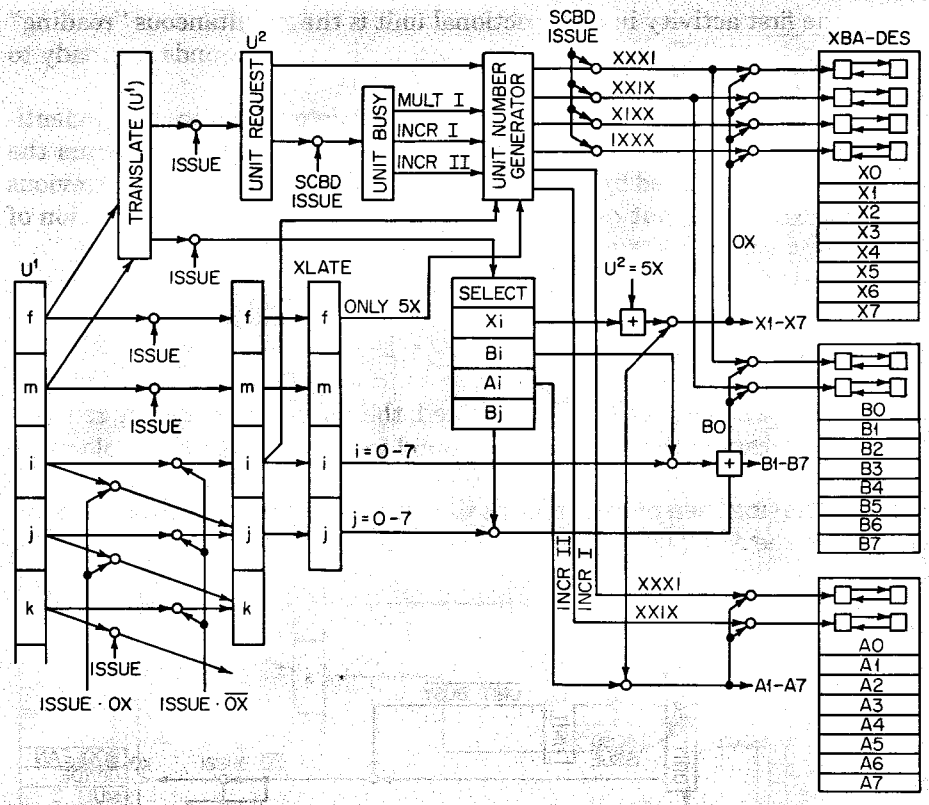


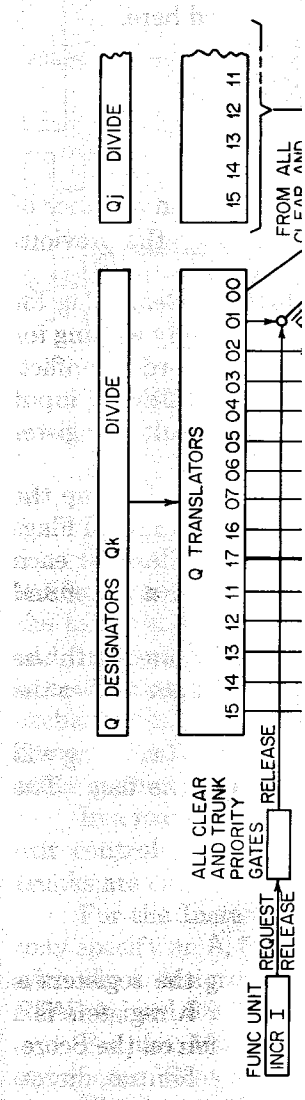
FIGURE 78 Set XBA

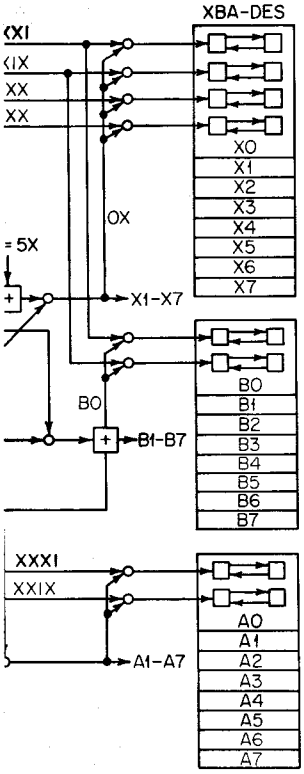
signal is sent to all units, among them the Divide Unit. This is shown in Figure 79.

All Release signal lines are shown as they appear to the divisor input to the Divide Unit. The case in point is the Release signal from the Add Unit, which AND's with the translation of the unit number held in Qk for the Divide Unit. Since the Qk identifier can hold only one unit number, only one Release signal is selected. Assuming the Q identifier is set to zero, meaning no wait is necessary, the Read Flag is set immediately after issue.

Figure 79 also shows how the Release signals are actually sent to all Read Flag networks. The example of Release for the Read Storage Channel 5 is shown going to the Q translation for unit number 15 on all nine units. This example appears to skip some Read Flag circuits, but it should be remembered that the X Registers are not connected as input operands in every combination to all units. The k operand in the Increment Units and the j operand in the Shift Unit are noteworthy.

When both Read Flags are set on any unit, the unit may be expected to start. However, it should be clear that several units could reach this condition simultaneously. For units which share data trunks (Chapter V),





Unit. This is shown in

pear to the divisor input
ease signal from the Add
number held in Qk for the
ly one unit number, only
ntifier is set to zero, mean-
diately after issue.
ls are actually sent to all
the Read Storage Chan-
umber 15 on all nine units.
cuits, but it should be re-
as input operands in every
Increment Units and the

the unit may be expected
al units could reach this
data trunks (Chapter V),

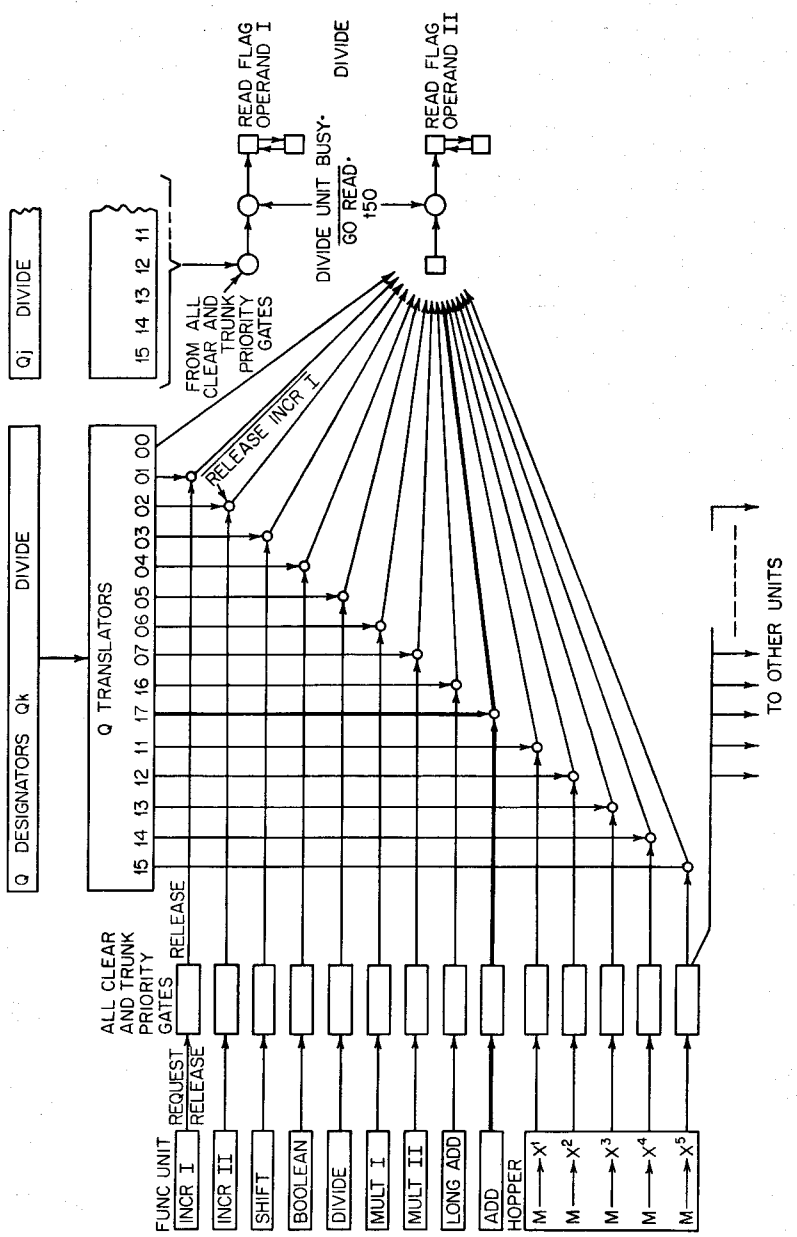
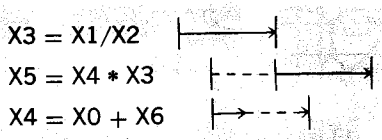


FIGURE 79 Set read flags divide unit.

this would mean simultaneous traffic on the trunk. Therefore, the data trunk priority condition also controls the start of the unit.

RELEASE

An additional factor in the Scoreboard control has to do with the Release signal. The release of the result to the result register would be uncomplicated were it not for the third order conflict and the result data trunk conflict. The third order conflict described before is repeated here.



In this example the third instruction is completed well in advance of the first two but cannot release its result to register X4 until the previous Read is accomplished.

Close examination of the example will show that the Read Flag for Multiply j input, corresponding to the X4 input, is set and simply waiting for the k Read Flag. The k Read Flag is held up by a second order conflict. Note that the third instruction would not be issued if the Multiply j input Read Flag were not set since that would indicate a previous result to register X4 not yet completed.

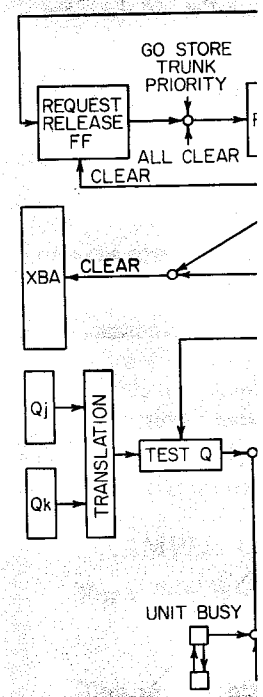
This is a form of proof that a Read Flag can be cause to hold up the Release signal. Each register can be described as "all clear" if no Read Flags are set corresponding to that register. To generate the All Clear for each register, the Fj and Fk designators are translated to the register number and ANDed with the associated j or k Read Flag.

These ALL CLEAR signals for each register are then combined with the translation of the result designator, Fi, for the unit to determine whether the unit should be allowed to release its result.

Assuming that the unit is held back, some time later the Read Flag will be cleared as a result of its unit starting, thereby clearing the flag. The entire case is presented in Figure 80.

E. REGISTER ENTRY/EXIT CONTROL

The secondary control over data entering and leaving the registers is provided by a rather simple system. Entry to the X, B or A registers is a direct result of the Release mechanism described in the section on the Scoreboard. A "GO STORE" signal is generated by the release mechanism, directing the requesting functional unit to transmit its result to the registers via its result trunk. At the same time, the result designator, usually Fi, is also sent to the register end of that trunk. This designator is translated and control is initiated to clear the result register and transfer the data from the



trunk to the correc
the fixed-time natu
and data trunks. I
isters X, B, and A.
exchange jump for i
isters used to hold t
order conflict case (

In a fixed, syn
exit control provid
trunks are controlle

For the Increr
may specify an A, E
with the k designa
READ Aj, and GO RI

For the Mult
are needed, GO RE/

Three control
pair because of the

All GO READ
described previousl

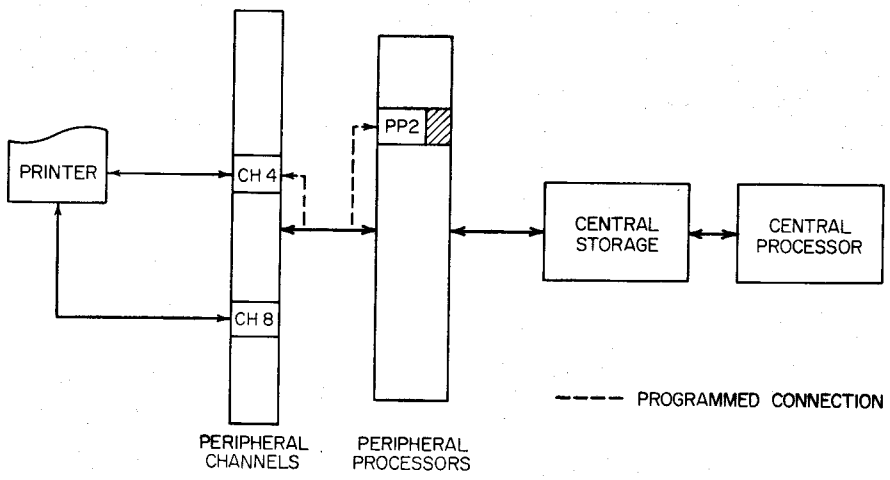


FIGURE 3

C. CENTRAL PROCESSOR—CPU

The Central Processor of the Control Data 6600 Computer is based on a high degree of functional parallelism. This is provided by the use of many functional units and a number of essential supporting properties, as shown in Figure 4.

The ten functional units are independent of each other and may operate

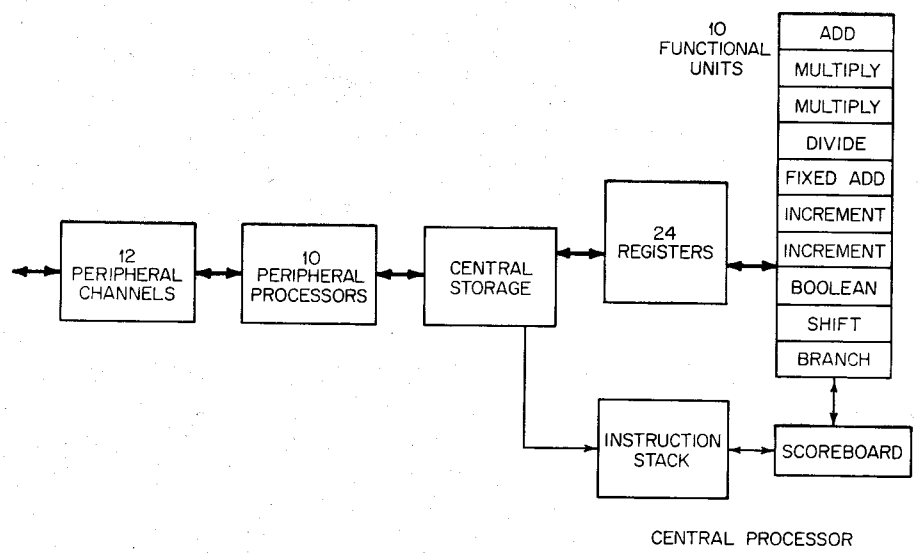


FIGURE 4

simultaneously. In a ty
functional units will be i

- Floating Add
- Floating Multiply (2)
- Floating Divide
- Fixed Add
- Increment (2)
- Boolean
- Shift
- Branch

Twenty-four regist
these are assigned as o
Eight are assigned as in
are assigned as address
metic functions are exec
turned to the registers.
efficient instruction pac
eighteen-bit registers pr
Instructions in the
dress for each of two o

contains two operands,
The use of registe
handling of partial or in
be used for these values.
eration would be requir
Instructions are lo
under control of a Pro
ceeds, up to a maximun
some circumstances, the
memory. An obvious c

Locatic

Program Addr
Program Addr
Program Addr
Program Addr
Program Addr

In this example a
location n + 4 calls for
cumstances, this entire
The program can loop w
any storage references