631875

FORM 1
REGULATION 9

COMMONWEALTH OF AUSTRALIA

PATENTS ACT 1952-1973

APPLICATION FOR A PATENT

We TANDEM COMPUTERS INCORPORATED

of 19333 Vallco Parkway, Cupertino, CALIFORNIA 95014, U.S.A.

hereby apply for the grant of a Patent for an invention entitled:

PAIRED INSTRUCTION PROCESSOR PRECISE EXCEPTION HANDLING MECHANISM

which is described in the accompanying complete specification. This Application is a Convention Application and is based on the Application numbered: 357,238 for a Patent or similar protection made in U.S.A. on 24 May 1989.

Our address for service is:

GRIFFITH HACK & CO
71 YORK STREET
SYDNEY NSW 2000

DATED this 17th day of May 1990

TANDEM COMPUTERS INCORPORATED
By their Patent Attorney

GRIFFITH HACK & CO

TO: THE COMMISSIONER OF PATENTS
COMMONWEALTH OF AUSTRALIA

9235A:rk

10577- 211 AU

COMMONWEALTH OF AUSTRALIA

Patents Act 1952

Regulation 12(2)

DECLARATION IN SUPPORT OF A CONVENTION APPLICATION FOR A PATENT

To be signed by the applicant(s) or in the case of a body corporate to be signed by a person authorised by the body corporate.

In support of the Convention application made for a patent for an invention entitled

(a) Insert title of invention.

(a) PAIRED INSTRUCTION PROCESSOR PRECISE EXCEPTION HANDLING MECHANISM

I, Theodore S. Park C/O.

(b) Insert full name(s) of declarant(s).

~~I/We~~ (b) TANDEM COMPUTERS INCORPORATED 19333 VALLCO PARKWAY

(c) Insert address(es) of declarant(s).

of (c) CUPERTINO, CALIFORNIA 95014 UNITED STATES OF AMERICA

do solemnly and sincerely declare as follows:–

1. ~~I am/We are the applicant(s) for the patent~~

(OR, IN THE CASE OF AN APPLICATION BY A BODY CORPORATE.)

1. I am/~~We are~~ authorised by TANDEM COMPUTERS INCORPORATED the applicant for the patent to make this declaration on its behalf.

2. The basic application(s) as defined by Section 141 of the Act was/were made in the following country or countries on the following date(s) namely:–

(d) Insert country in which basic application(s) was/were filed.

(e) Insert date of basic application(s).

(f) Insert full names of basic applicant(s).

in (d) the U.S.A. on (e) MAY 24, 1989
by (f) ~~Jardine, Lynch, Manela and Horst~~ Robert L. Jardine; Shannon J. Lynch; Philip R. Manela and Robert W. Horst
in (d) ..... on (e) .....
by (f) .....
in (d) ..... on (e) .....
by (f) .....

3. ~~I am/We are the actual inventor(s) of the invention referred to in the basic application~~.

(OR, WHERE A PERSON OTHER THAN THE INVENTOR IS THE APPLICANT)

(g) Insert full name(s) of actual inventor(s)

(h) Insert address(es) of actual inventor(s).

3. (g) Robert L. Jardine, Shannon J. Lynch, Philip R. Manela, Robert W. Horst
of (h) 10500 Mira Vista Avenue, Cupertino, California 95014, 646 Camellia Way, Los Altos, California 94022, 406 Iris Street, Redwood City, California 94062, 2804 Robeson Park Drive, Champaign, Illinois 61821, USA
is/are the actual inventor(s) of the invention and the facts upon which the applicant(s) is/are entitled to make the application are as follows:

(i) Set out how applicant(s) derive(s) title from actual inventor(s) i.e., assignee of the invention from the actual inventor(s).

(i) by virtue of assignment between the applicant and the inventors for the said invention.

4. The basic application(s) referred to in paragraph 2 of this Declaration was/were the first application(s) made in a Convention country in respect of the invention the subject of the application.

Attestation or legalization not required.

Declared at Cupertino California this 15 day of May 1990

To:

The Commissioner of Patents

By: [signature]
THEODORE S. PARK

Signature of Declarant(s)

(12) PATENT ABRIDGMENT (11) Document No. AU-B-55154/90
(19) AUSTRALIAN PATENT OFFICE (10) Acceptance No. 631875

(54) Title
PAIRED INSTRUCTION PROCESSOR PRECISE EXCEPTION HANDLING MECHANISM

International Patent Classification(s)
(51)[5] G06F 009/38

(21) Application No. : 55154/90 (22) Application Date : 17.05.90

(30) Priority Data

(31) Number	(32) Date	(33) Country
357238	24.05.89	US UNITED STATES OF AMERICA

(43) Publication Date : 29.11.90

(44) Publication Date of Accepted Application : 10.12.92

(71) Applicant(s)
TANDEM COMPUTERS INCORPORATED

(72) Inventor(s)
ROBERT L. JARDINE; SHANNON J. LYNCH; PHILIP R. MANELA; ROBERT W. HORST

(74) Attorney or Agent
GRIFFITH HACK & CO., GPO Box 4164, SYDNEY NSW 2001

(57) Claim

1. In a data processor that issues a family of instructions fetched from a single sequence of instructions during a single clock and includes a pipeline for concurrently executing the family of instructions in several stages, the pipeline including a first given stage for calculating memory references and performing ALU operations and a second given stage for writing data, and with the data processor including exception handling procedures for handling exceptions resulting from the execution of a single instruction, an exception handling procedure for handling an exception that occurs during the execution of a given family of instructions issued during a single clock period and prevents the execution from being completed comprising the steps of:

generating an indication whether the exception resulting from executing an instruction in the given family issued during the single clock period in the first given pipeline stage and preventing the execution from being completed, has occurred;

testing said indication to detect the occurrence of the exception;

inhibiting, during the second given pipeline stage, the data write associated with the execution of the given family if the presence of the exception is detected;

.../2

flushing instructions from all the stages of the pipeline if the presence of the exception is detected; and

reissuing a first instruction in the given family to be executed singly so that the included exception handling procedure may be utilised to handle the exception if the exception resulted from executing said first instruction.

631875

COMMONWEALTH OF AUSTRALIA

PATENTS ACT 1952 Form 10

COMPLETE SPECIFICATION

FOR OFFICE USE

Short Title:

Int. Cl:

Application Number:
Lodged:

Complete Specification-Lodged:
Accepted:
Lapsed:
Published:

Priority:

Related Art:

TO BE COMPLETED BY APPLICANT

Name of Applicant: TANDEM COMPUTERS INCORPORATED

Address of Applicant: 19333 Vallcc Parkway, Cupertino, CALIFORNIA 95014, U.S.A.

Actual Inventor: Robert L. Jardine; Shannon J. Lynch; Philip R. Manela and Robert W. Horst

Address for Service: GRIFFITH HACK & CO
71 YORK STREET
SYDNEY NSW 2000

Complete Specification for the invention entitled:

PAIRED INSTRUCTION PROCESSOR PRECISE EXCEPTION HANDLING MECHANISM

The following statement is a full description of this invention, including the best method of performing it known to us:-

16223-CG:COS:RK

9235A:rk

PAIRED INSTRUCTION PROCESSOR
PRECISE EXCEPTION HANDLING MECHANISM

BACKGROUND OF THE INVENTION

1. Field of the Invention

The present invention relates generally to digital processors and, more particularly, to the instruction issuing and execution units of a digital processor.

2. Description of the Relevant Art

A primary goal in the design of digital processors is to increase the throughput, i.e., the number of instructions processed per unit time, of the processor. One approach has been to improve the hardware design of the processor to reduce the machine cycle time. Another approach has been to develop architectures and instruction sets designed to process one instruction per machine cycle. Both of these approaches are limited to a theoretical maximum throughput of one instruction per machine cycle due to basic policy of sequentially issuing at most one instruction per cycle.

Systems for issuing more than one instruction per cycle are described in a paper by Ditzel et al. entitled "The Hardware Architecture of the CRISP Microprocessor", 1098 ACM 0084-7495 87, pp. 309-319 and in a paper by Acosta et al. entitled "An instruction issuing Approach to Enhancing Performance in Multiple Functional Unit Processors", IEEE Transactions on Computers, Vol. C-35, No. 9, September 86, pp. 815-828.

One limitation on concurrent issuing of instructions is that the instructions must not require the use of the same functional unit of the processor during the same machine cycle. This limitation is related to the resources included in the processor architecture and can be somewhat obviated by providing additional copies of heavily used functional units.

The paper by Acosta et al. presents an approach to concurrently issuing instructions to take advantage of the existence of multiple functional units. Further, the CRISP architecture, described in the above-referenced paper, allows the execution of a branch instruction concurrently with another instruction. Additionally, mainframes have allowed concurrent dispatching of integer and floating point instructions to different functional units.

However, all of these systems require that the instructions issued concurrently not be dependent on each other. Types of dependencies will be discussed fully below, but a fundamental dependency between a pair of instructions is that the second instruction in the pair processes data resulting from the execution of the first instruction in the pair. Accordingly, the first instruction must be processed prior to the second.

Thus, these existing processors may concurrently issue and execute very few combinations of instructions. A branch instruction is a special case where no memory reference is required and requires only that a new address be calculated. Similarly, floating point and integer instructions require only ALU resources and no memory reference. Thus, data dependencies between the instructions do not exist.

In view of the above limitations, the type of instructions that may be concurrently issued in these systems is extremely limited and, although in certain limited situations two instructions may be issued in one clock, the average throughput cannot significantly exceed one clock per instruction.

SUMMARY OF THE INVENTION

A family of instructions is a set of sequential instructions in a program that may be issued concurrently in one clock. In embodiments of the present invention, the number of types of instructions that may be included in a family is greater than allowed in prior art processors.

AUSTRALIAN PATENT OFFICE

In embodiments of the present invention, a family of instructions that includes, for instance, instructions of the ALU and memory reference type may be issued during a single clock. A special pipeline includes resources that facilitate the acceptance and processing of the issued family. Thus, embodiments of the present invention provide for an instruction processing throughput of greater than one instruction per clock.

In one aspect of the present invention there is provided a data processor that issues a family of instructions fetched from a single sequence of instructions during a single clock and includes a pipeline for concurrently executing the family of instructions in several stages, the pipeline including a first given stage for calculating memory references and performing ALU operations and a second given stage for writing data, and with the data processor including exception handling procedures for handling exceptions resulting from the execution of a single instruction, an exception handling procedure for handling an exception that occurs during the execution of a given family of instructions issued during a single clock period and prevents the execution from being completed comprising the steps of:

generating an indication whether the exception resulting from executing an instruction in the given family issued during the single clock period in the first given pipeline stage and preventing the execution from being completed, has occurred;

testing said indication to detect the occurrence of the exception;

inhibiting, during the second given pipeline stage, the data write associated with the execution of the given family if the presence of the exception is detected;

flushing instructions from all the stages of the pipeline if the presence of the exception is detected; and

reissuing a first instruction in the given family to be executed singly so that the included exception handling procedure may be utilised to handle the exception if the exception resulted from executing said first instruction.

In a second aspect of the present invention there is provided a data processor that issues a family of instructions fetched from a single sequence of instructions during a single clock and includes a pipeline for concurrently executing the family of instructions in several stages, the pipeline including a first given stage for calculating memory references and performing ALU operations and a second given stage for writing data, and with the data processor including exception handling procedures for handling exceptions resulting from the execution of a single instruction, an exception handling system for handling an exception that occurs during the execution of a given family of instructions issued during a single clock period and prevents the execution from being completed comprising:

means for generating an indication whether the exception, resulting from executing an instruction in the given family issued during the single clock period in the first given pipeline stage and preventing the execution from being completed, has occurred;

means, responsive to the output of said generating means, for testing said indication to detect the occurrence of the exception;

means, responsive to the output of said testing means, for inhibiting, during the second given pipeline stage, the data write associated with the execution of the given family if the presence of the exception is detected;

means, responsive to the output of said testing means, for flushing instructions from all the stages of the pipeline if the presence of the exception is detected; and

means, responsive to the output of said flushing means, for reissuing a first instruction in the given family to be executed singly so that the included exception handling procedure may be utilised to handle the exception if the exception resulted from executing said first instruction.

Other features and advantages of the invention will become apparent in view of the figures and following detailed description.

BRIEF DESCRIPTION OF THE DRAWINGS

Fig. 1 is a high-level block diagram of a preferred embodiment of the present invention ;

Fig. 2 is a schematic diagram illustrating a specific example of sequential and concurrent execution of a first pair of instructions;

Fig. 3 is a schematic diagram illustrating a specific example of sequential and concurrent execution of a second pair of instructions;

Fig. 4 is a block diagram of a prior art three-stage pipeline;

Fig. 5 is a schematic diagram of a six-stage pipeline utilized in an embodiment of the invention;

Fig. 6 is a schematic diagram of a multiported register file with bypass circuitry; and

Fig. 7 is a schematic diagram of the pairing logic utilized in an embodiment of the invention;

Fig. 8 is a block diagram of the memory map of a microstore utilized in an embodiment of the invention;

Fig. 9 is a flow chart of an embodiment of the exception handling procedure of the present invention;

Fig. 10 is a flow chart of an embodiment of the unpaired restart procedure of the present invention;

Figs. 11A - 11J are detailed schematic diagrams illustrating the pipeline stages for exception and branch processing procedures of a preferred embodiment of the present invention;

Fig. 11K is a block diagram of exception handling control system.

Fig. 12 is a flow chart of an embodiment of the branching procedure of the present invention; and

Fig. 13 is a flow chart of an embodiment of the procedure for handling a branch misprediction.

Fig. 14 is a block diagram of a branch prediction mechanism control system.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

A preferred embodiment executes a target instruction set utilized by an existing processor not designed for issuing more than one instruction per cycle. Thus, the embodiment is downwardly compatible with the existing system and may operate programs written for the system. However, as will be demonstrated below, the ability of the present system to concurrently process families of instructions dramatically increases throughput.

A preferred embodiment is a microprogrammed machine where the control signals for implementing each object code, or macro, instruction are provided by a microcode routine unique to the instruction.

A subset of pair candidate (PC) instructions from the target instruction set is selected and special microcode routines for concurrently executing families of two PCs are stored in the control store. The selection of which instructions are included in the subset of PCs depends on

various factors including the frequency of the occurrence of the instructions in application programs, the difficulty of concurrent execution of the instructions, and the resources required to concurrently execute the instructions. These special routines are dependent on special resources, to be described below, in the processor.

However, even for families consisting of only PCs, concurrent execution is not possible if resource conflicts occur. Accordingly, the system cancels concurrent execution in these instances. The execution unit is configured so that such instances occur infrequently and do not significantly reduce throughput.

The present system is object code compatible with non-pairing processors that execute the target instruction set. This requires that an object code program comprising an ordered sequence of object code instructions that is executed by the non-pairing machine must also be executed by the pairing machine without any modification to the program.

Thus, in the present system, ordered pairs of instructions occurring in the program are issued concurrently if predetermined conditions are satisfied. In a preferred embodiment the occurrence of these conditions are indicated by status bits generated when the instructions are decoded.

Referring now to the drawings, where like reference numerals identify identical or corresponding parts throughout the several views, Fig. 1 is a high-level block diagram of a preferred embodiment.

In Fig. 1, a fetch unit (FU) 10 is coupled to an instruction cache (IC) 12. The FU 12 includes address calculation hardware controlled by an address state machine that increments addresses during normal sequential operation and calculates branch target (BRANCH-TARG) addresses for conditional branch and jump instructions. The output of the IC 12 is coupled to instruction queue registers (IQR) 14 and 16. The IQRs 14 and 16 and first and second rank 0 pipeline registers (ROS and ROF) 18 and 20 are connected in series to

form an instruction queue 21. The outputs of ROS and ROF 18 and 20 are coupled, respectively, to the inputs of first and second rank 1 registers and to the inputs of a second decode unit and a first decode unit (DCS and DCF) ~~22~~ 24 and ~~24~~ 22. The output of the two decode units 22 and 24 are connected to the data inputs of a pairing logic unit (PLU) 26 by first and second DCO buses 28 and 30. Status bits on these buses are connected to control inputs of the PLU 26 by first and second status buses 32 and 34. The PLU ~~16~~ 26 includes a data output coupled to an execution unit (EU) 36 by a merged decode bus (MDB) 38, an FU control output coupled to the FU ~~20~~ 10 by an FUC bus 40, and a PU control output coupled to the PU 25 by a PUC bus 42. The EU 36 also assert the below described FIssue signal.

The operation of the system of Fig. 1 will now be described. The first and second instructions in the family of instructions are stored in ROF and ROS 20 and 18, respectively. These instructions are decoded at the respective decode units 22 and 24 and the decode results are output on the respective DCO buses 28 and 30. The first decode result includes a first entry point field (EPT-F) and a first set of status bits and the second decode result includes a second entry point field (EPT-S) and a second set of status bits. The status bits in each decode result indicate whether the respective instruction is a PC and what resources are required to execute the instruction. The status information from both decode results is routed to the PLU control ports on the status buses 32 and 34.

A first status bit is designated PC and indicates whether the instruction is a pair candidate. A second status bit is designated EPTIssue and indicates whether the macro instruction requires only one clock or requires multiple clocks. If so, then the following instruction may be issued during the next clock. If not, then the data in ROF and ROS 20 and 18 doesn't change until an FI issue occurs. This signal is received by the FU 10 to restart prefetching.

Other status bits indicate whether the instruction requires the ALU or memory addressing logic to execute.

The output of the PLU 26 depends on the status information and will be one of three possibilities. First, if the first instruction is not a PC the decode result of the first instruction is output on the MDB 38 to access the microcode routine for singly executing the first instruction. Additionally, the signals issued on the FUC line 40 and the PUC line 42 cause a next single instruction to be fetched from the IC 12 and only the first instruction (stored in ROF 20) to be issued to the next stage of the PU 25. Thus, the concurrent issuing capability of the processor is not used.

Secondly, if both the first and second instructions are PCs and are pairable, i.e., no resource conflicts exist, the decode results of the first and second instructions are merged and output on the MDB 38. The merged decode results access microcode for executing the first and second instructions as a pair. Additionally, the signals issued on the FUC line 40 and the PUC line 42 cause the next two instructions in the program to be fetched from the IC 12 and the pair of instructions stored in the Rank 0 registers 18 and 20 to be issued to the next stage of PU 25.

Third, if the first instruction is a PC, but the first and second instructions are not pairable, e.g., because of a resource conflict, a subfield of the decode result of the first instruction is output on the MDB 38 to access a microcode routine for singly executing the first instruction. Additionally, the signals issued on the FUC line 40 and the PUC line 42 cause a next single instruction to be fetched from the IC 12 and only the first instruction to be issued from the ROF register 20 to the next stage of the PU 25. Thus, the concurrent issuing capability of the processor is not used.

The operation of the system is best understood by considering concrete examples. These examples are for a processor that utilizes a stack as the data source and data

sink for ALU operations. Data is transferred between memory and the stack by load and store operations. The stack includes eight physical registers (R0-R7) and a register pointer (RP) which points to the physical register logically assigned to the top of the stack. The stack registers are given logic designations A-H, defined by the value of RP, with the register at the top of the stack designated A. In this type of processor, data dependencies are characterized by stack register conflicts. Another type of dependency is an RP conflict. Instructions generally increment or decrement RP. Thus, a family of instructions may have conflicting requirements for the change of RP.

Figs. 2 and 3 are schematic diagrams depicting the stack configurations resulting during the execution of a family of two instructions in a standard manner and listing a routine for concurrently executing the instructions according to the present invention.

Referring to Fig. 2, the stack configurations for the sequential single execution of an LDI-LOAD family of two instructions are depicted. Both of these instructions write data to the H register defined relative to the RP. During the execution of LDI, immediate data is written to H (R0) and RP is incremented to change H to R1. Then, during the execution of LOAD, cache data is written to H and RP is incremented again to change H to R2.

In terms of a standard three stage pipeline depicted in Fig. 4, each of these instructions writes data to the H register and increments RP during the R3 pipeline stage. Thus, concurrent execution would not be possible without special resources and techniques to obviate the effects of these register and RP conflicts.

The concurrent execution of the pair is described in terms of the relatively deep six stage pipeline, utilized in the preferred embodiment, depicted in Fig. 5. To concurrently execute the pair, the data cache address for the load instruction is generated during Rank 2, the operands are fetched, cache data is written to G (R1) and

immediate data is written to H (R0) during Rank 3, and RP is increased by 2 to define R? as A and R2 as H during Rank 4. Thus, the microcode and resources of the present invention allow concurrent issuance and execution of the pair of instructions.

Referring to Fig. 3, the stack configurations for the sequential single execution of an LDD-DADD family of two instructions are depicted. Referring again to the three stage pipeline of Fig. 4, during execution of LDD, RP is incremented to define A as R2, the high field of OP-2 data is written to A (R2), and the low field of OP-2 data is written to H (R3) during Rank 2. Finally, RP is incremented again to define A as R3 during Rank 3. Then, during the execution of DADD, the data in C (R1) and in A (R3) are summed and written to C (R1) and the data in D (R0) and B (R2) are summed and written to D (R0) during Rank 2 and RP is decremented by 2 during rank 3.

In terms of the three stage pipeline of Fig. 4, if the pair were executed concurrently the writing of data and summation of data would occur during rank 3 thereby causing an unresolvable data conflict. A further conflict would occur during rank 3 because of conflicting requirements for changing RP.

Referring to the six stage pipeline of Fig. 5, to concurrently execute the pair, OP-2 data is read from the data cache during Rank 3, summed with the contents of registers A (R1) and B (R0) during Rank 4, and the sum written to A (R1) and B (R0) as well as the original OP-2 data written to H (R2) and G (R3) during Rank 5. Thus, no unresolvable register or RP conflicts occur.

From the above it is clear that concurrent execution requires access to data before it is written to registers in the stack. Fig. 6 depicts a register configuration facilitating the required access. In Fig. 6 a register file 60 includes four write ports and four read ports to allow concurrent transfer of double words. Additionally, bypass circuitry 62, bypass buses 64, wide

muxes and cross connections 65 between cache data 65a, immediate data 65b, and the ALU input buses 65c and SBus 65d allow direct access to data prior to its being written to the multiported register file 60. Thus, the deep pipeline and multiported register file 60 allow microcode to execute many combinations of instructions as families.

For example, during the execution of the LDD-DADD pair illustrated in Fig. 3, the cache data is transferred directly, during Rank 3, from the data cache 66 to the CDHi 67a and CDLo 67b registers prior to being written to the register file while OP-1 H is transferred from RegA to KHReg 67c and OP-1 L is transferred from RegB to KLReg 67d. The sum of transferred data is calculated by the ALU 68 during Rank 4 and written to RegA and RegB during Rank 5. Additionally, the data in CDHi 67a and CDLo 67b is written to RegH and RegG during Rank 5.

For example, a family including a load and load immediate instruction can be issued during one clock and processed. Similarly, a family including a load and an add instruction can be issued during one clock and processed.

Fig. 7 is a detailed block diagram illustrating the architecture of an embodiment of the invention. The decode units are entry point tables (EPTs) 22 and 24 that respond to the instructions in ROF and ROS 20 and 18 to generate first and second entry points EPF and EPS.

The output of the EPTs 22 and 24 are coupled, respectively, to the first and second DCO buses 28 and 30. The first EP bit field F<0:15> is transferred from EPTF 22 to the first DCO bus 28 and the second EP bit field S<4:15> is transferred from EPTS 24 to the second DCO bus 30. The bit field F<1,2,14,15> is transferred to the Pairing Logic 72 by the first status bus 32 and the bit field S<5,6,7,8> is transferred by the second status bus 34.

The bit field F<9:15> is transferred to the 0 input of a MUX/LOGIC UNIT (MLU) 74 by an FLSB bus 76 and the bit field S<9:15> is transferred to the 1 input of the MLU 74 by an SLSB bus 78.



The bit field F<2:8> is transmitted on an MMSB bus 80 and the output of the MLU 74 is transferred to an MLSB bus 82. The MMSB and MLSB buses 80 and 82 are merged to form MSB and LSB sections of the merged EPT (MEPT) bus 38. The MEPT is used to access microcode from a microstore 39.

The Pairing Logic generates PC and EPTIssue signals, transferred on FUC bus 40, that direct the pipeline to issue an instruction pair or a single instruction. Additionally, a NonPaired PC signal is transmitted on an PCNP line 84 coupled to the 09 input of the MLU 74 and the bit F<2> is transmitted on a mux control line 86 to the control input of the MLU 74.

Fig. 8 is a memory map of the microstore 39 addressed by MEPT<2:15>. The MSB field of the MEPT is always equal to F<2:8> where F<2> is the MSB. The LSB field of the MEPT is equal to the output of MLU 74 and depends on the signals on the NPPC and mux control lines 84 and 86. The MSB, F<2>, of all PCs has a value of 1 and for all non-PCs has a value of 0.

For a first instruction that is not a pair candidate the MSB, F<2>, is 0 and the microcode address (MEPT) is located in the lower half 90 of the address space 92. For a first instruction that is a pair candidate the MSB, F<2>, is 1 and the microcode address (MEPT) is the upper half 94 of the address space.

The operation of the system depicted in Fig. 7 to generate the MEPT for the cases where the first instruction is a not a pair candidate, is a pair candidate but is not paired, and is a pair candidate and is paired will now be described.

If the first instruction is not a pair candidate then F<2> is 0 and the 0 input of the MLU 74 is coupled to the MLSB bus 82 so that the field F<9:15> is transmitted on the MLSB bus 82 and the MEPT is:

$$MEPT\langle 2{:}15\rangle = F\langle 2{:}8\rangle : F\langle 9{:}15\rangle \qquad \text{Eq. 1}$$

so that MEPT is equal to the EPTF field. This address is in the lower half 90 of the address space 92 depicted in Fig. 8. Thus, the MEPT in this case accesses microcode for executing the first instruction as a single.

If the first instruction is a pair candidate but is not pairable with the second instruction then F<2> is 1 and the signal on the NPPC line 84 is set. In this case the MLU 74 transfers the field <0000000> to the MLSB bus 82 and the MEPT is:

$$MEPT<2:15> = F<2:8>:<0000000> \qquad \text{Eq. 2}$$

so that the MEPT is equal to the MSB field of the EPTF followed by a string of seven zeros. Thus, the address of the microcode for executing a non-paired pair candidate is located in the upper half 94 of the address space 92 depicted in Fig. 8.

If the first instruction is a pair candidate and paired then F<2> is 1 and signal on the NPPC line 84 is not set. In this case the MLU 74 transfers the field S<9:15> to the MLSB bus 82 and the MEPT is:

$$MEPT<2:15> = F<2:8>:S<9:15> \qquad \text{Eq. 3}$$

so that the MEPT is equal to the LSB field of the EPTS and the MSB field of the EPTF. As depicted in Fig. 8, these addresses follow the address of the unpaired instruction in the address space.

Subsequent to the issue of an instruction pair, events, such an exception or branch misprediction, may occur that prevent the successful execution and retirement of the issued instruction pair. The following is a description of unique exception handling and branching techniques for efficiently coping with the occurrence of these events.

Turning first to exception handling, an exception is caused by a condition internal to the processor that prevents the execution of an instruction. Examples of such

conditions include arithmetic overflows for ALU type instructions and page faults for memory reference type of instructions.

For each instruction that may encounter an exception, special exception handling procedures that may include software and/or microcode, have been developed for singly issued instructions. Generally, the exception handling microcode is much more complicated than the microcode required to implement the same instruction in the absence of an exception.

In terms of the six stage pipeline depicted in Fig. 5, the exception condition occurs during rank 4 of the pipeline. Since exceptions occur when instructions are paired, one approach to exception handling would be to store special microcode for handling exceptions for each possible instructions pair.

However, such an approach has several drawbacks. Since instructions prior to the one that encounters the exception must be allowed to complete, complex fix-up code would be required in some cases to allow stores associated with the first instruction to complete while preventing stores associated with the second instruction in the pair. Further, it is possible that more than one exception could be encountered. For example, the pair (LOAD and ADD) might encounter a page fault exception for the LOAD and an overflow exception for the ADD. Additionally, the number of exception combinations becomes very large and makes the pair exception procedures extremely difficult to debug.

The present solution utilises the exception handling procedures already existing for handling exceptions for singly issued instructions. The microcode for implementing the pair detects an exception by testing for the presence of an exception condition during rank 4 of the pipeline. An instruction pair is issued 90 and ranks 1 to 3 of the pipeline are executed 92. This procedure

is depicted 94 then the unpaired restart procedure is called 96 to abort the instruction pair and reissue the first instruction in the pair singly. If there is no exception the instruction pair is retired 98.

For the existing exception handling procedures to be utilized, it required to present an internal macrostate to the single instruction exception handling procedure consistent with the macrostate that would have occurred if the program were being executed by a non-paired instruction machine.

This consistent macrostate is presented by an unpaired restart procedure, depicted in the flow chart of Fig. 10, that includes the step of inhibiting 99 the loads and stores that occur during rank 5 if an exception is detected in rank 4. Thus, the contents of the stack registers are not changed and the internal macrostate is the state resulting from the execution of the instruction in the program that immediately precedes the aborted pair. This macrostate is consistent with the state that would be presented to the exception handling procedures if the program were executed by a non-paired instruction machine.

In addition to inhibiting the loads and stores of rank 5, the unpaired restart flushes the pipeline and reissues 99a and 99b the first instruction of the pair as a single. If the exception was associated with the first instruction, then the singles microcode will again encounter the same exception and can handle the exception as in a non-paired instruction machine.

If the exception was associated with the second instruction, then the first instruction will complete without incident and the second instruction will again encounter the exception. If the second instruction is not paired this time then the singles microcode will handle the exception. If it is paired then another unpaired restart will occur.

Although the unpaired restart procedure reduces throughput, the relative infrequency of exception occurrence makes this penalty acceptable in view of the tremendous reduction in complexity.

Figs. 11A through 11G depict the various pipeline stages and procedures for handling an exception while a pair of instructions is executing.

In Fig. 11A, the pipeline 25 is divided into instruction and address sides that include instruction and address queues 21I and 21A and instruction and address rank register sets 25I and 25A. Additionally, an instruction mux 90I and an address mux 90A selectively couple the outputs of the IQ2 and IQ3 registers and rank 5 registers of the instruction and address sides, respectively, to the address calculation hardware of the FU 10. In Figs. 11B through 11G buses and registers that are enabled are indicated by bold lines.

In Fig. 11B, the instruction pair A + B is ready to issue and is output from the rank 0 registers 18I and 20I of the instruction side and the addresses of A and B from the rank 0 registers 18A and 20A on the address side. The bits of the instructions are transferred to the EPTs on DCO buses 28 and 30.

Figs. 11C through 11F depict the progress of the instruction pair from rank 1 to rank 4. At rank 4 the microcode tests for an exception such as the setting of an arithmetic overflow bit or a page fault. If no exception is detected the pair continues through rank 5 and is retired. This testing may be implemented by a conditional branch in the microcode that utilizes an exception status bit as a branch condition bit. For example, an exception indicating bit could be the arithmetic overflow bit stored in a condition code register.

Fig. 11G illustrates the unpaired restart procedure. The microcode controls the address mux 90A to transfer the address of instruction A to the address calculation hardware in the fetch unit 10. Additionally, the pair A + B and their addresses are loaded into the rank 1 through 5 registers of the instruction and address sides respectively to begin the flush of the pipeline.

In Fig. 11H, the instruction calculating hardware in the fetch unit 10 accesses the instruction A from the IC 12 and the microcode controls the IQ muxes to route the instruction to the instruction R0 registers 20I and the instruction address to address R0 register 20A.

In Fig. 11I, instruction A and its address have been transferred to the instruction R0F and the address R0F 20I and 20A, respectively.

In Fig. 11J, instruction A has been reissued as a single.

Fig. 11K is a block diagram of a control system for implementing the unpaired restart procedure. Referring to Fig. 11K, the output of the control store is coupled to an MCR3 register 100 which, in addition to MCR4 and MCR5 registers 102 and 104, form a microcode pipeline which operates in synchronism with the instruction pipeline 25 to provide required control signals to execute the instruction families in the pipeline. The rank 4 microcode includes an m-bit field which is coupled to the first input port of an AND gate 106. An exception indication test bit field generated by ALU 108 of the EU 36 is coupled to the input of a latched MUX 110. The control port of MUX 110 is coupled to a control field of the rank 5 microcode and the output is coupled to the second input port of the AND gate 106. The output of the AND gate 106 is coupled to the input of a first decoder (DEC1) 112. The outputs of DEC1 are the control signals which cause the pipeline to be flushed and the Rank 5 write operations to be inhibited in the event that an exception condition is detected.

The operation of the system depicted in Fig. 11K will now be described. The control field of rank 5 microcode causes the MUX 110 to transmit the particular exception indication test bit to be tested for the instruction family being executed. If an exception condition has occurred then the transmitted test bit is a logical "1", AND gate 106 is open, and the m-bit Rank 5 microcode field is decoded to generate the control signals

for implementing the unpaired restart procedure. If the exception condition has not occurred then the transmitted test bit is a logical "0", the AND gate 106 is closed, and the outputs of the decoder do not cause the unpaired restart procedure to be implemented.

The procedure for handling branch instructions will now be described with reference to the flow charts of Figs. 12 and 13. As in many pipelined systems, a branch prediction ram (BPR) is included. For each branch instruction the state of a branch prediction bit (BPB) in the BPR indicates the value of the branch condition the last time the associated branch instruction was executed. It is assumed that the branch condition bit will again have the same value. Referring to Fig. 12, the BPB is tested 113a and if the BPB predicts that the branch will not be taken then prefetching continues in order of ascending addresses 113b. If the branch is predicted taken, then prefetching stops while the address of the target address is formed. Once this target address is formed, prefetching resumes at the target location 113c.

As described above with reference to Fig. 7, the MEPT is uniquely determined by the bits of the first and second instructions in a pair and reflects the order of the instructions, ie. the MSB field includes bits from the EP of the first instruction and the LSB field includes bits from the EP of the second instruction. This unique MEPT accesses microcode that includes an indication of whether the first or second instruction is the branch instruction.

The branch condition is evaluated at rank 4. The BPB moves through the pipeline 113d and 113e as a PipeID bit along with the pair or instructions and is compared 113f to the branch condition but (BCB) set by a previously executed instruction. If the BPB and branch condition bit match, then the contents of the pipeline

are correct, normal processing continues 113g, and the instruction pair is retired. If the BPB and BCB do not match 113h, then the following instructions in the pipeline are incorrect and branch recovery must occur.

Referring now to Fig. 13, the first step in branch recovery depends on whether the branch instruction is the first or second instruction in the pair. As described above, the microcode routine is different for the two cases. If the mispredicted branch instruction is the first instruction of the pair, then the rank 5 stores of the second instruction must be inhibited because that instruction is not the instruction that is to be executed subsequent to the first instruction. Additionally, all subsequent instructions are flushed from the pipeline and any updates required for the branch are redone.

If the mispredicted branch instruction is the second instruction of the pair, then the Rank 5 stores associated with the instruction pair are completed. The remaining stages of the pipeline and the prefetch queue are flushed as before.

In the case of a branch misprediction, a new address must be formed to resume prefetching at the location of the correct next instruction the follows the branch instruction in the program.

TABLE 1

Branch is First/Second	Branch was Predicted Taken?	Next Address
First	Yes	R5PF + 1
First	No	TARG of R5PF
Second	Yes	R5PS + 1
Second	No	TARG of R5PS

Address recalculation is required only when the BPB does not match the BCB. Since the branch prediction was incorrect the instructions in the pipeline following the branch instruction are incorrect. Thus, if the BPB indicates that the branch is taken then the address of the instruction in the pipeline following the branch instruction is BRANCH-TARG. However, the following instruction address should be BRANCH + 1. Similarly, if the BPB indicates that

the branch is not taken then the address of the instruction in the pipeline following the branch instruction is BRANCH + 1. However, the following instruction address should be BRANCH-TARG.

Referring to Table 1, in the first row the microcode indicates that the first instruction in the pair, now resident in register R5I-F of the pipeline, is the branch instruction. Since the BPB mispredicted that the branch would be taken, the address of the next instruction should be BRANCH + 1 instead of BRANCH-TARG.

Referring back to Fig. 11A, the address of the branch instruction, now resident in register R5P-F, is transferred to the address calculation hardware of the FU 10 via the address side MUX 90A. This address is incremented to form the address, BRANCH + 1, of the next instruction to be fetched.

In the second row of table 1 the microcode indicates that the first instruction is the branch instruction. Since the BPB mispredicted that the branch would not be taken, the address of the next instruction should be BRANCH-TARG instead of BRANCH + 1.

Referring to Fig. 11A, the branch instruction and its address, now resident in registers R5I-F and R5P-F, respectively, are routed to the address calculation hardware of the FU 10 via the instruction side and address side MUXes 90I and 90B. The address calculation hardware then calculates the address, BRANCH-TARG, of the next instruction to be fetched.

Fig. 14 is a block diagram of a control system for implementing the branch prediction mechanism. The control store 39 and MCR3, MCR4, and MCR5 registers 100, 102, and 104 are coupled as described above with reference to Fig. 11K. Additionally, each rank of the control pipeline includes a Pipe ID (PID) register 120 which stores certain information regarding the instruction being executed. A branch prediction RAM (BPR) 122 has an output coupled to the PID register 120 so that a branch prediction bit (BPB) is

propogated through the pipeline for a branch instruction being executed. The BPB stored in the Rank 4 PID register 120 is coupled to a first input of a comparator 124 and a branch condition bit stored in a condition code register 126 or transferred from the ALU is coupled to the second input of the comparator 124. An n-bit field of the Rank 4 microcode is coupled to the first input and the output of the comparator 124 is coupled to the second input of an AND gate 128 via MUX 127. MUX 127 is controlled by a control field of the Rank 4 microcode. The output of the AND gate 128 is coupled to the input of a decoder (DECA) 130. The outputs of DECA 130 are the control signals which cause the branch prediction mechanism to be implemented.

The operation of the system depicted in Fig. 14 will now be described for four separate cases. The first two cases relate to conditional branch instructions where the BCB is a bit set in the condition code register 126 by a previously executed instruction. The two cases are distinguished by whether the previous instruction was included in a previously executed family or in the current family that includes the branch instruction.

The second two cases relate to a conditional branch instruction where the BCB is calculated by the ALU using data stored in the register file, for example RegA, by a previously executed instruction. Again the two cases are distinguished by whether the previous instruction is included in the current family. In the first case, the BCB in the condition code register 126 has been set during the execution of a previous instruction. If the BCB and the BPB from the PID 120 register do not match then the branch was mispredicted and the output of the comparator 124 opens AND gate 128. The n-bit Rank 4 microcode field is then passed to DECA 130 to generate the control signals required to implement the branch prediction mechanism. As described above, the n-bit field indicates the position of the branch instruction in the family and thus the output of the decoder will differ depending on this location.

If the BCB and BPB match then the output of the comparator 124 closes AND gate 128 and the control signals for implementing the branch recovery mechanism are not generated because the contents of the pipeline are correct.

If the BCB is a bit from the condition code register 126 that is being written by an instruction in the current family then the BCB bit must be provided to the comparator 124 prior to being written to the register file. The MUX 127 is controlled to provide the output of the ALU directly to the comparator 124 during Rank 4 and then the BCB is written to the condition code register during Rank 4.

If the BCB is calculated by the ALU and the register data was written by an instruction in a previous family, the register data is transferred to the ALU and the ALU output is transferred to the comparator 124 via the MUX 127.

If the register data is generated by an instruction in the current family, the data is provided to the ALU, utilizing the hardware described with reference to Fig. 6, prior to being written to the register file and the ALU output is transferred to the comparator 124 via the MUX 127 during Rank 4. The data is then written to the register file during Rank 5.

The invention has been described for a system for concurrently executing families of two instructions, however the principles of the invention are equally applicable to families of more than two instructions. The pairing logic would be modified to respond to more than two status fields and special microcode routines would be written to concurrently execute the family.

Additionally, although a microcoded processor has been described the system is applicable to machines using logic arrays to generate control signals. Both systems have advantages in certain situations and the choice is not critical to practicing the invention. Further, a register configuration other than a register stack can be utilized.

The status bits described with reference to Fig. 7 are used by the pairing logic to determine whether to merge the EPs of the family of instructions. Alternatively, the EPs could always be merged and the accessed microcode would control whether the family of instructions is issued concurrently.

Further, the selection of a subset of target instructions that may issue concurrently is not required. Microcode routines can be provided for every possible pair of instructions from the target instruction set.

Thus, it is apparent that although the invention has been described with reference to preferred embodiments, substitutions and modification would be obvious to a person of ordinary skill in the art. Accordingly, it is not intended to limit the invention except as provided by the appended claims.

THE CLAIMS DEFINING THE INVENTION ARE AS FOLLOWS:

1. In a data processor that issues a family of instructions fetched from a single sequence of instructions during a single clock and includes a pipeline for concurrently executing the family of instructions in several stages, the pipeline including a first given stage for calculating memory references and performing ALU operations and a second given stage for writing data, and with the data processor including exception handling procedures for handling exceptions resulting from the execution of a single instruction, an exception handling procedure for handling an exception that occurs during the execution of a given family of instructions issued during a single clock period and prevents the execution from being completed comprising the steps of:

generating an indication whether the exception resulting from executing an instruction in the given family issued during the single clock period in the first given pipeline stage and preventing the execution from being completed, has occurred;

testing said indication to detect the occurrence of the exception;

inhibiting, during the second given pipeline stage, the data write associated with the execution of the given family if the presence of the exception is detected;

flushing instructions from all the stages of the pipeline if the presence of the exception is detected; and

reissuing a first instruction in the given family to be executed singly so that the included exception handling procedure may be utilised to handle the exception if the exception resulted from executing said first instruction.

2. The procedure of claim 1 wherein said step of testing includes the step of:

determining whether an overflow bit has been set to indicate an arithmetic overflow exception resulting from executing an ALU instruction.

3. The procedure of claim 1 wherein said step of testing includes the step of:

determining whether a page fault bit has been set to indicate a page fault exception resulting from executing a memory reference instruction.

4. The procedure of claim 1 wherein said step of reissuing further comprises the steps of:

storing the address of each instruction in the family;

singly accessing said first instruction in the family with said stored address of said first instruction; and

reissuing said first instruction.

5. In a data processor that issues a family of instructions fetched from a single sequence of instructions during a single clock and includes a pipeline for concurrently executing the family of instructions in several stages, the pipeline including a first given stage for calculating memory references and performing ALU operations and a second given stage for writing data, and with the data processor including exception handling procedures for handling exceptions resulting from the execution of a single instruction, an exception handling system for handling an exception that occurs during the execution of a given family of instructions issued during a single clock period and prevents the execution from being completed comprising:

means for generating an indication whether the exception, resulting from executing an instruction in the given family issued during the single clock period in the first given pipeline stage and preventing the execution from being completed, has occurred;

means, responsive to the output of said generating means, for testing said indication to detect the occurrence of the exception;

means, responsive to the output of said testing means, for inhibiting, during the second given pipeline stage, the data write associated with the execution of the given family if the presence of the exception is detected;

means, responsive to the output of said testing means, for flushing instructions from all the stages of the pipeline if the presence of the exception is detected; and

means, responsive to the output of said flushing means, for reissuing a first instruction in the given family to be executed singly so that the included exception handling procedure may be utilised to handle the exception if the exception resulted from executing said first instruction.

6. The system of claim 5 wherein said means for testing includes:

means for testing whether an overflow bit has been set to indicate an arithmetic overflow exception resulting from executing an ALU instruction.

7. The system of claim 5 wherein said means for testing includes:

means for testing whether a page fault bit has been set to indicate a page fault exception resulting from executing a memory reference instruction.

8. The system of claim 5 wherein said means for reissuing further comprises:

means for storing the address of each instruction in the family;

means for singly accessing said first instruction in the family with said stored address of said first

instruction; and

means for reissuing said first instruction.

9. An exception handling procedure substantially as herein described with reference to the accompanying drawings, excluding figure 4.

10. An exception handling system substantially as herein described with reference to the accompanying drawings, excluding figure 4.

DATED this 8th day of October 1992

TANDEM COMPUTERS INCORPORATED
By their Patent Attorneys
GRIFFITH HACK & CO

55154/90

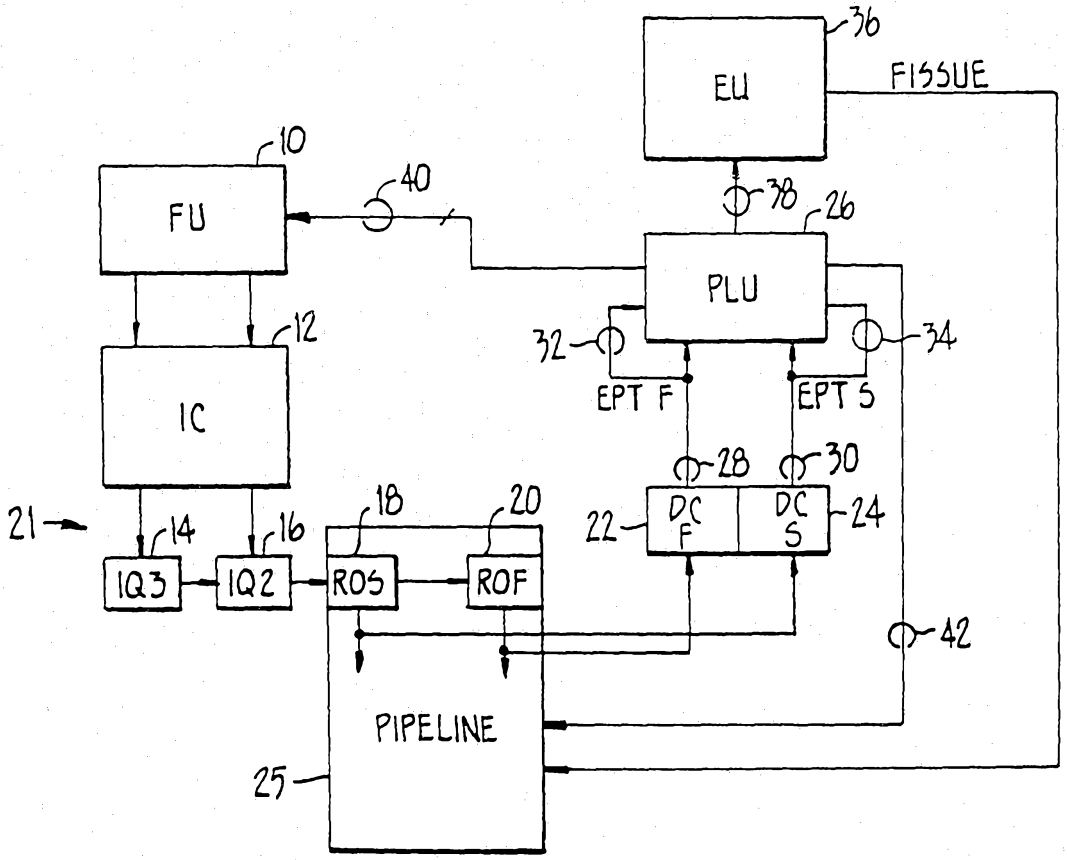


FIG._1.

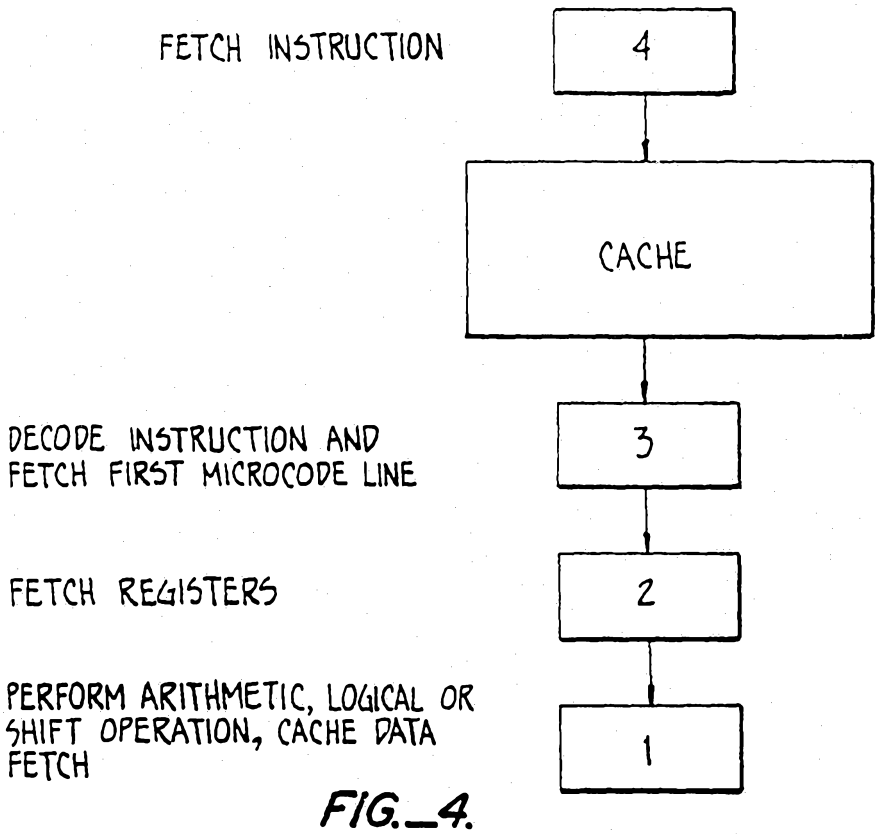
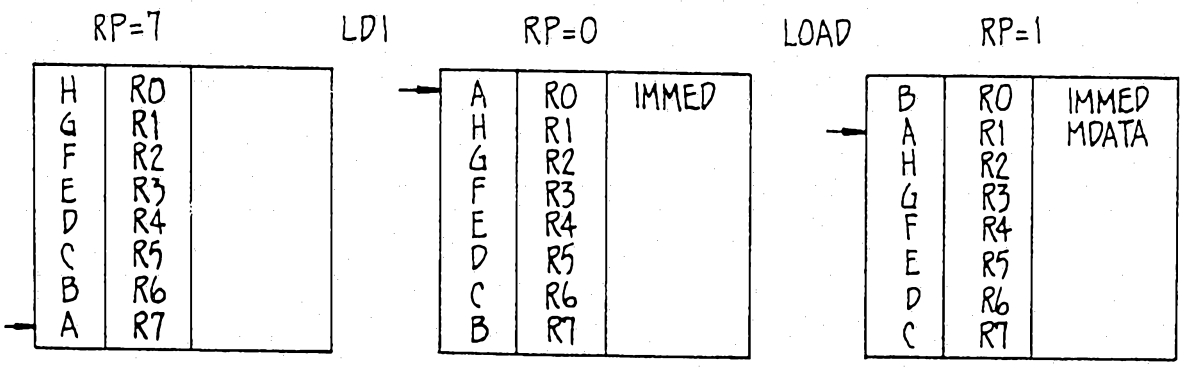


FIG._4.



SINGLE

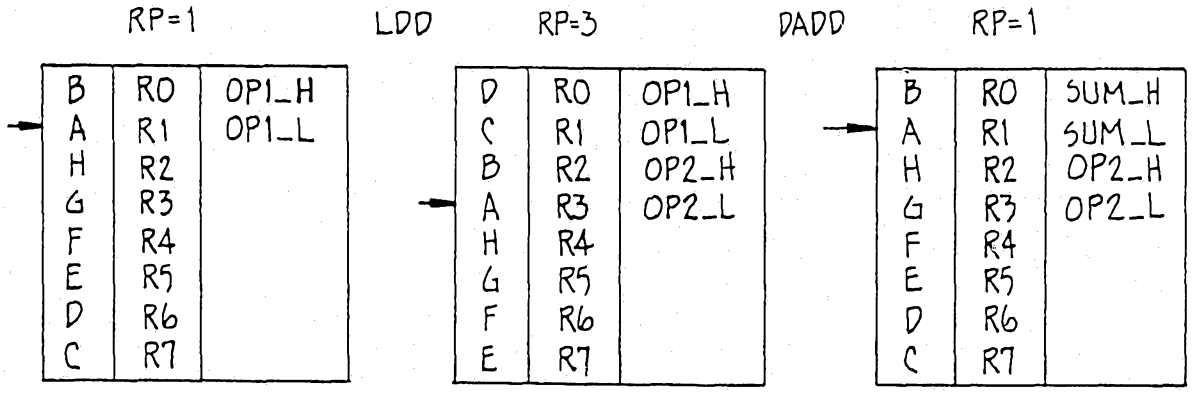
```
LDI: H: = DISP FROM K,
     INCRP
     ;
     FINI

LOAD: CDREG: = CACHE
      ;
      H: = CDREG,
      INCRP
      ;
      FINI
```

PAIR

```
LDI_LOAD: EARLY CACHE READ LOW,
          G: = CDREGLO VIA SL
          H: = IMMED_F,
          RP:= RP + 2
          ;
```

FIG._2.

LDD_DADD INSTRUCTION PAIR EXAMPLE




```
LDD: CDREG:= CACHE,
     INCRP ← R2
     ;
     A:= CDREG,
     CDREG:= CACHE
     ;
     H:= CDREG,
     INCRP ← R3
     ;
     FINI

DADD: C:= C+A, LOAD CYF,
      ;
      D:= D+B+CYF,
      DEC RP BY 2
      ;
      FINI
```

```
LDD_DADD: EARLY CACHE READ 32,
          BA:= CDREG32 + BA,
          HG:= CDREG32 VIA S32
          ;
```

FIG._3.

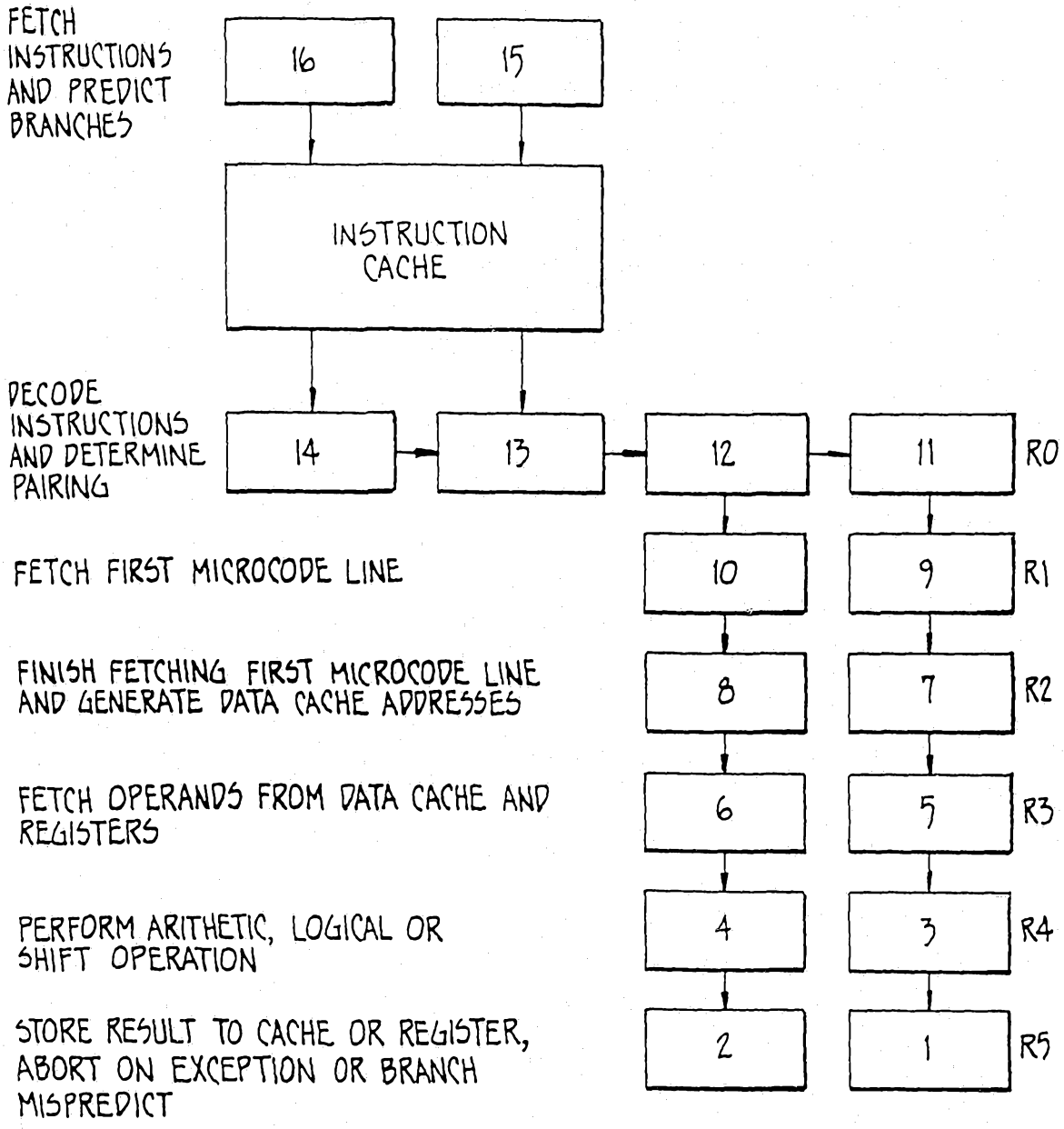

FETCH INSTRUCTIONS AND PREDICT BRANCHES
16
15
INSTRUCTION CACHE
DECODE INSTRUCTIONS AND DETERMINE PAIRING
14
13
12
11
R0
FETCH FIRST MICROCODE LINE
10
9
R1
FINISH FETCHING FIRST MICROCODE LINE AND GENERATE DATA CACHE ADDRESSES
8
7
R2
FETCH OPERANDS FROM DATA CACHE AND REGISTERS
6
5
R3
PERFORM ARITHETIC, LOGICAL OR SHIFT OPERATION
4
3
R4
STORE RESULT TO CACHE OR REGISTER, ABORT ON EXCEPTION OR BRANCH MISPREDICT
2
1
R5

FIG._5.

FIG._6A.	FIG._6B.

FIG._6.

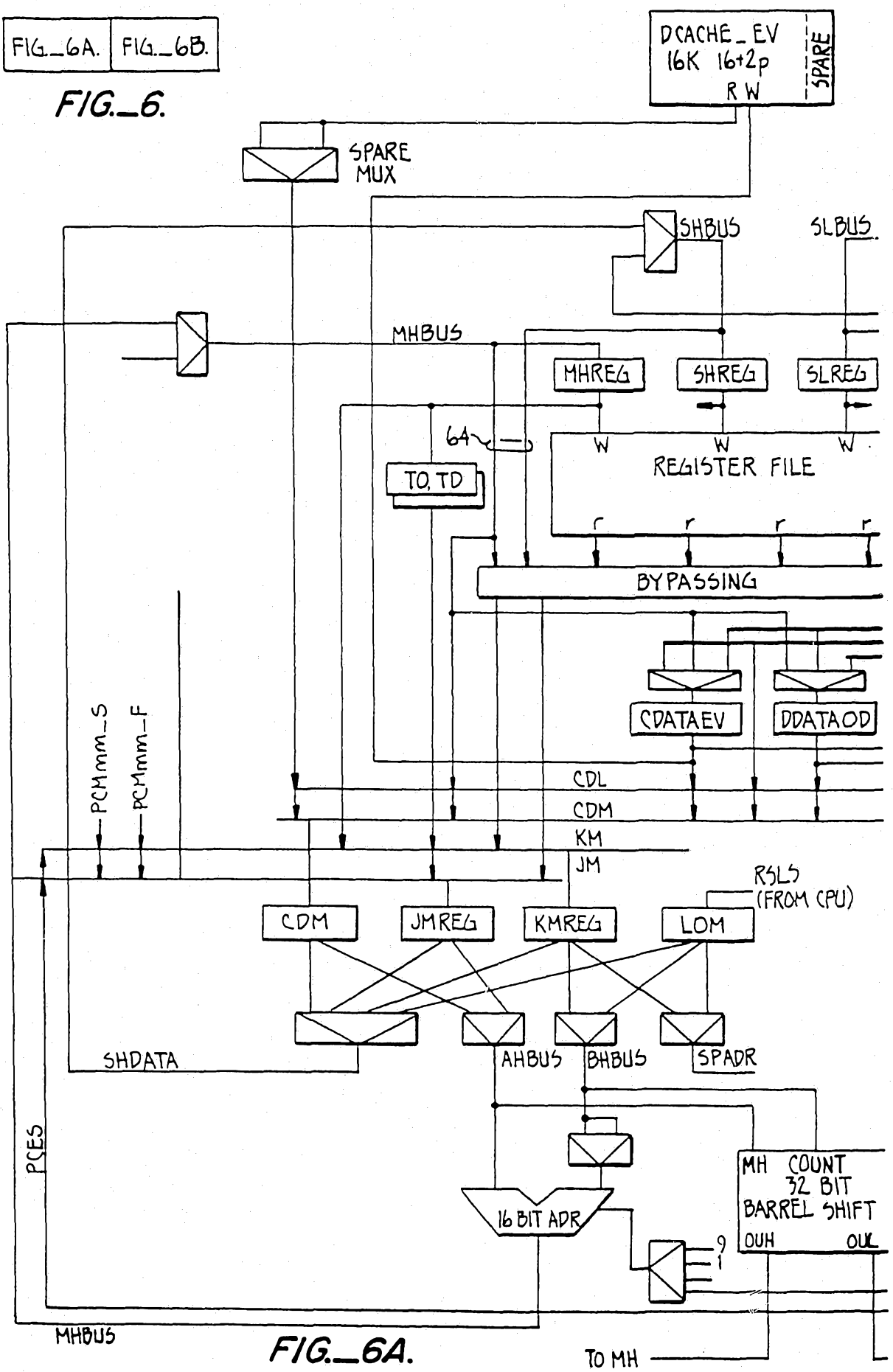


FIG._6A.

FIG._6A.	FIG._6B.

FIG._6.

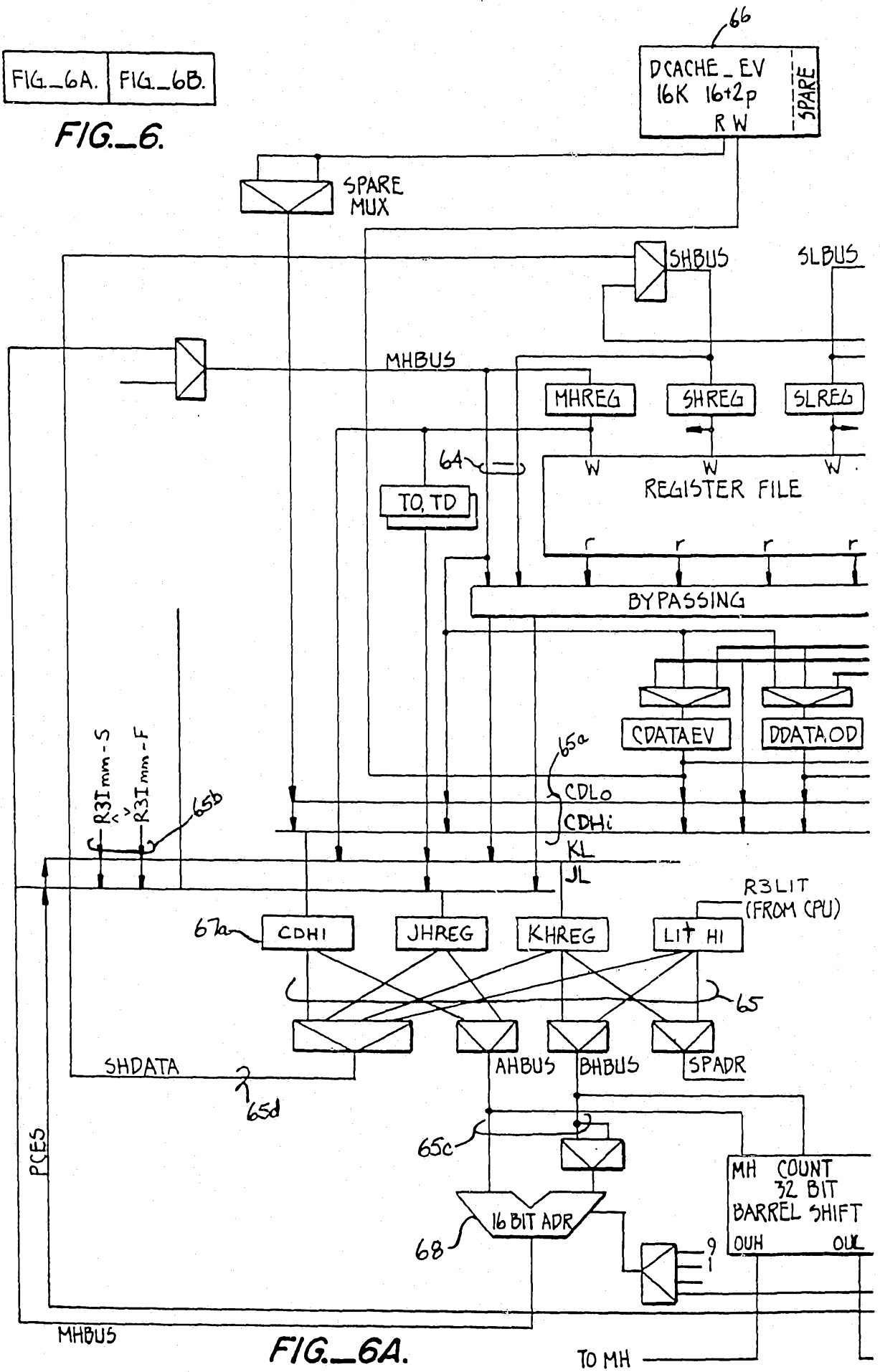


FIG._6A.

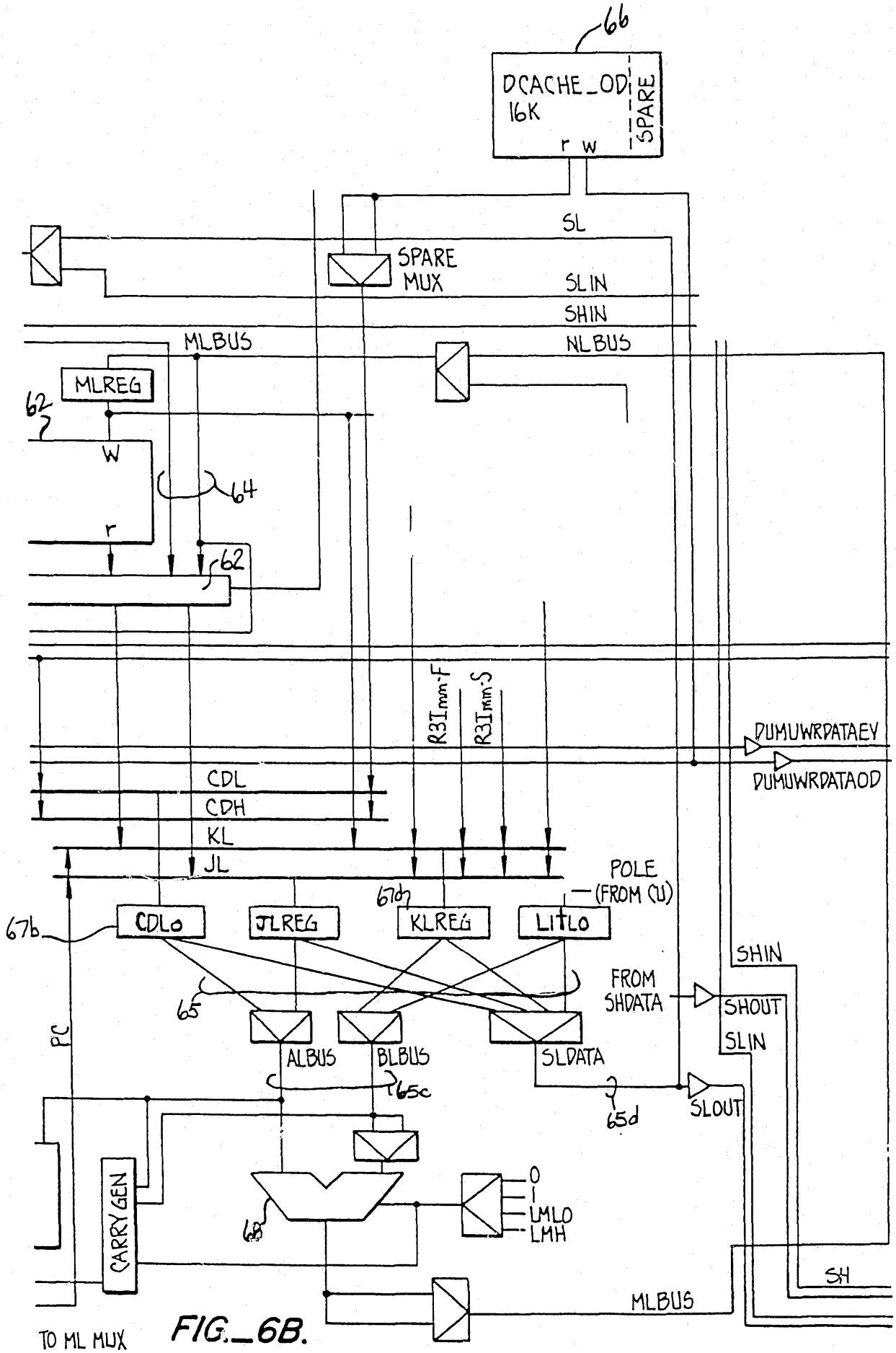

66
DCACHE_OD
16K
r w
SPARE
SL
SPARE MUX
SLIN
SHIN
MLBUS
NLBUS
MLREG
62
W
64
r
62
R3Imm-F
R3Imm-S
DUMUWRDATAEV
DUMUWRDATAOD
CDL
CDH
KL
JL
POLE
(FROM CU)
67d
67b
CDLO
JLREG
KLREG
LITLO
SHIN
FROM SHDATA
65
SHOUT
SLIN
PC
ALBUS
BLBUS
SLDATA
65c
65d
SLOUT
CARRY GEN
68
0
1
LMLO
LMH
SH
MLBUS
TO ML MUX

FIG._6B.

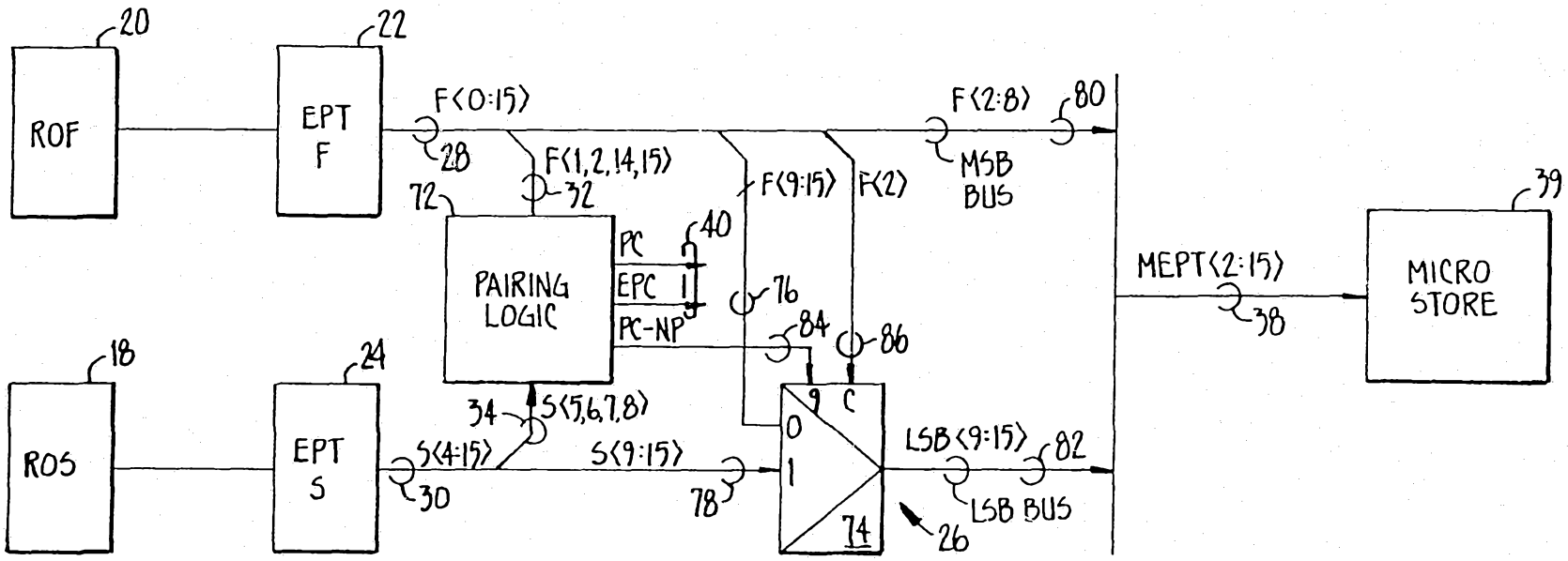

20
ROF
22
EPT F
F(0:15)
28
F(1,2,14,15)
32
72
PAIRING LOGIC
PC
EPC
PC-NP
40
F(9:15)
F(2)
F(2:8)
80
MSB BUS
76
84
86
MEPT(2:15)
38
39
MICRO STORE
18
ROS
24
EPT S
34
S(5,6,7,8)
S(4:15)
30
S(9:15)
78
9
C
0
1
74
LSB(9:15)
82
LSB BUS
26

FIG._7.

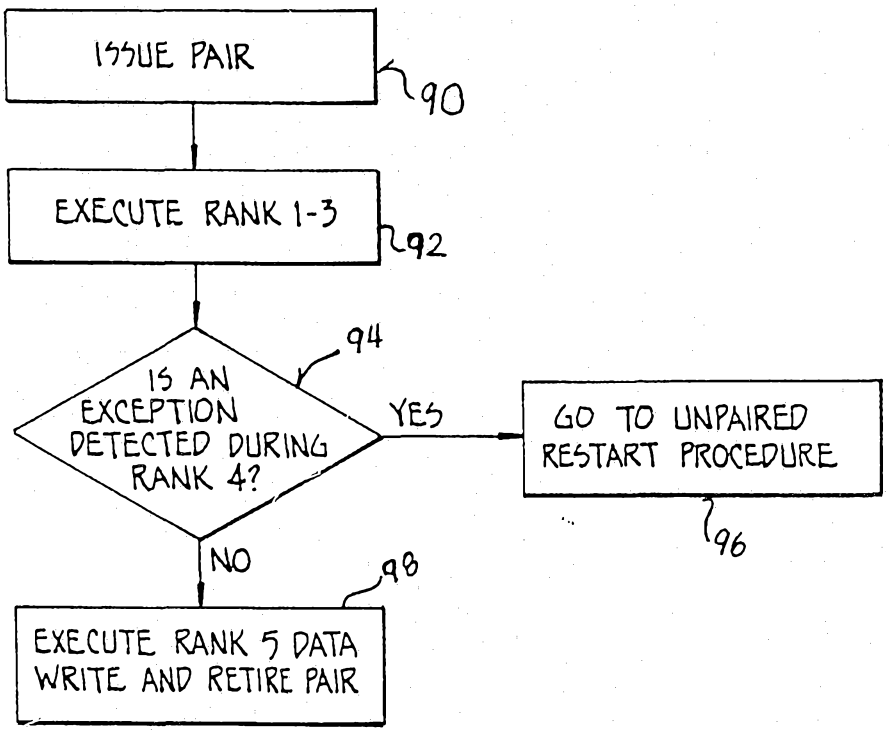


FIG._9.

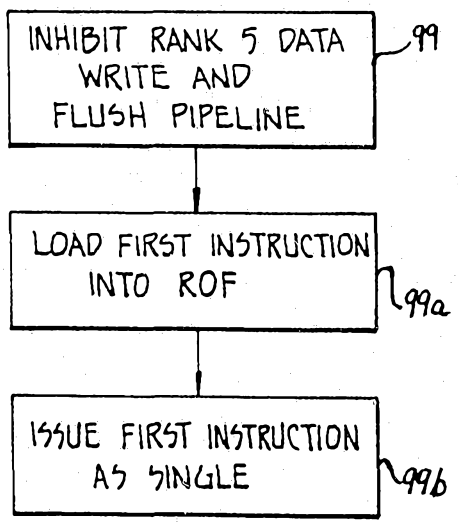


FIG._10.

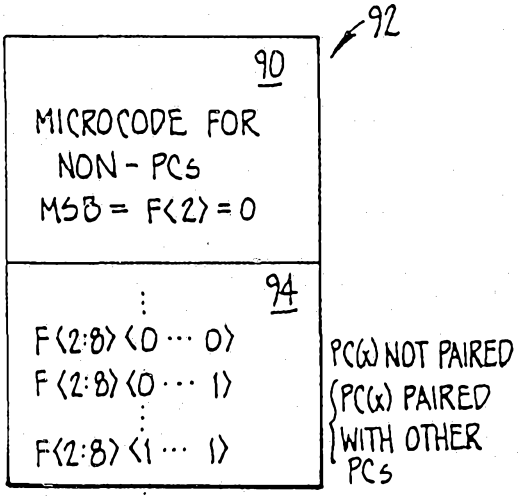
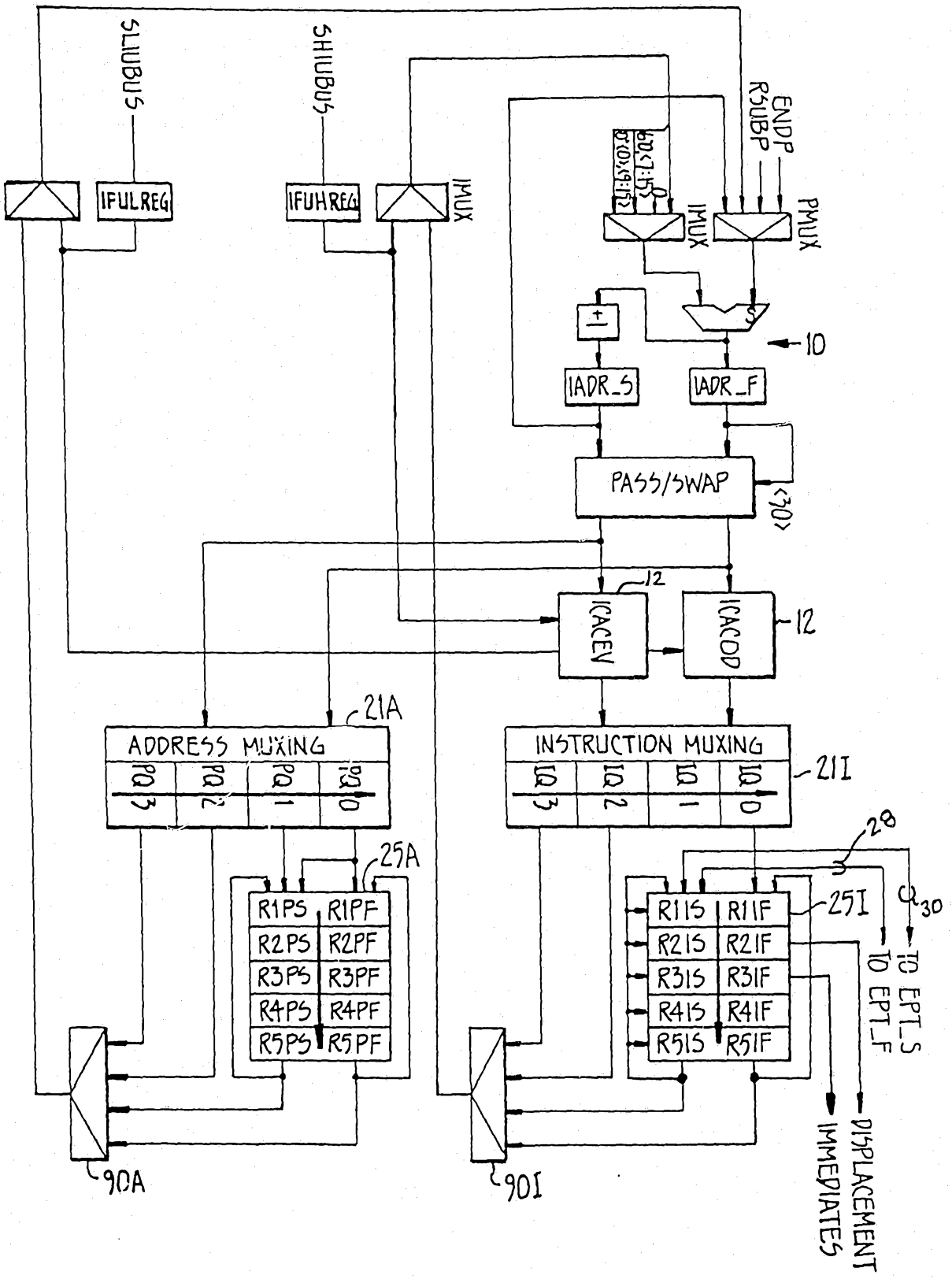
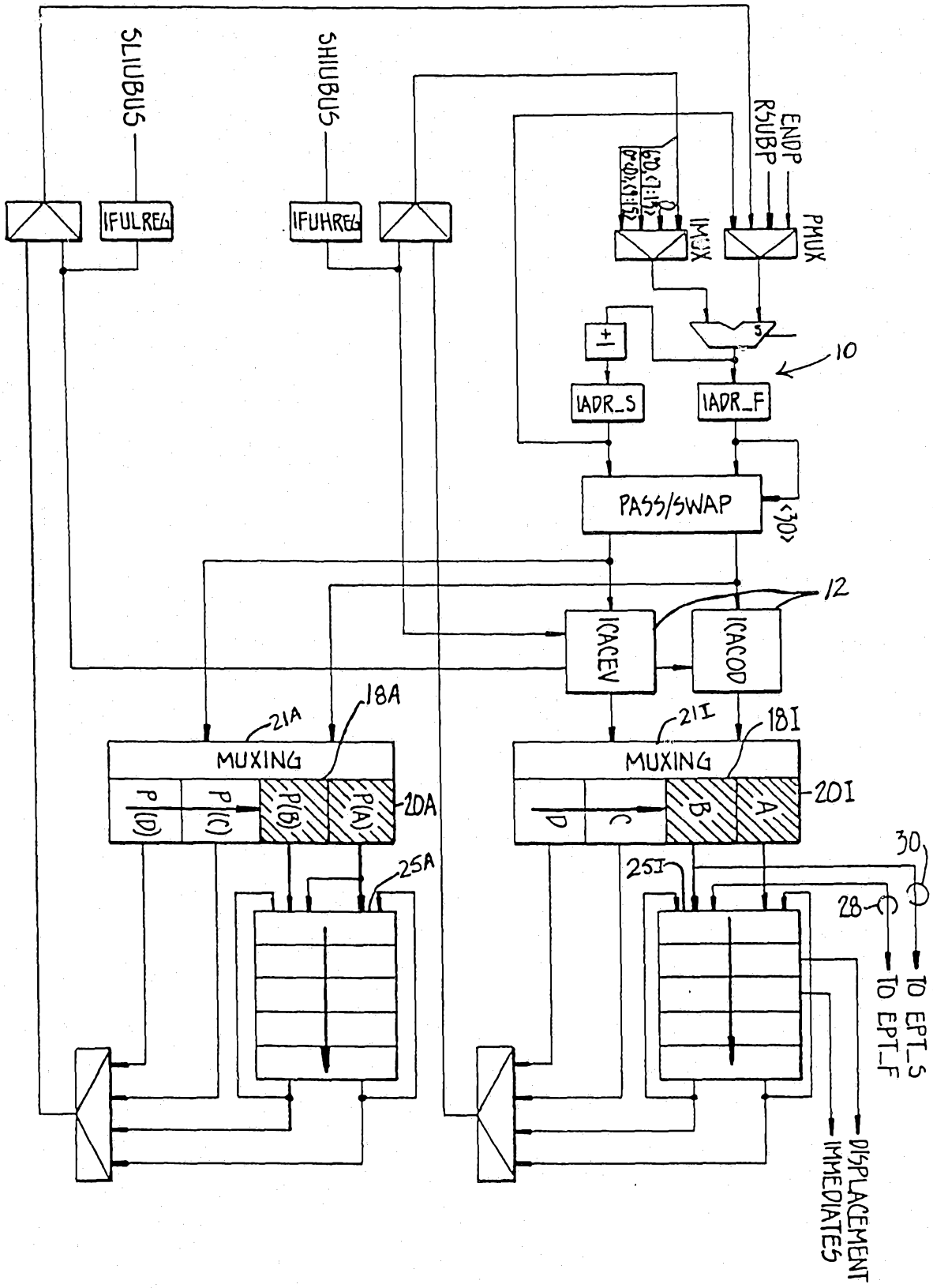


FIG._8.



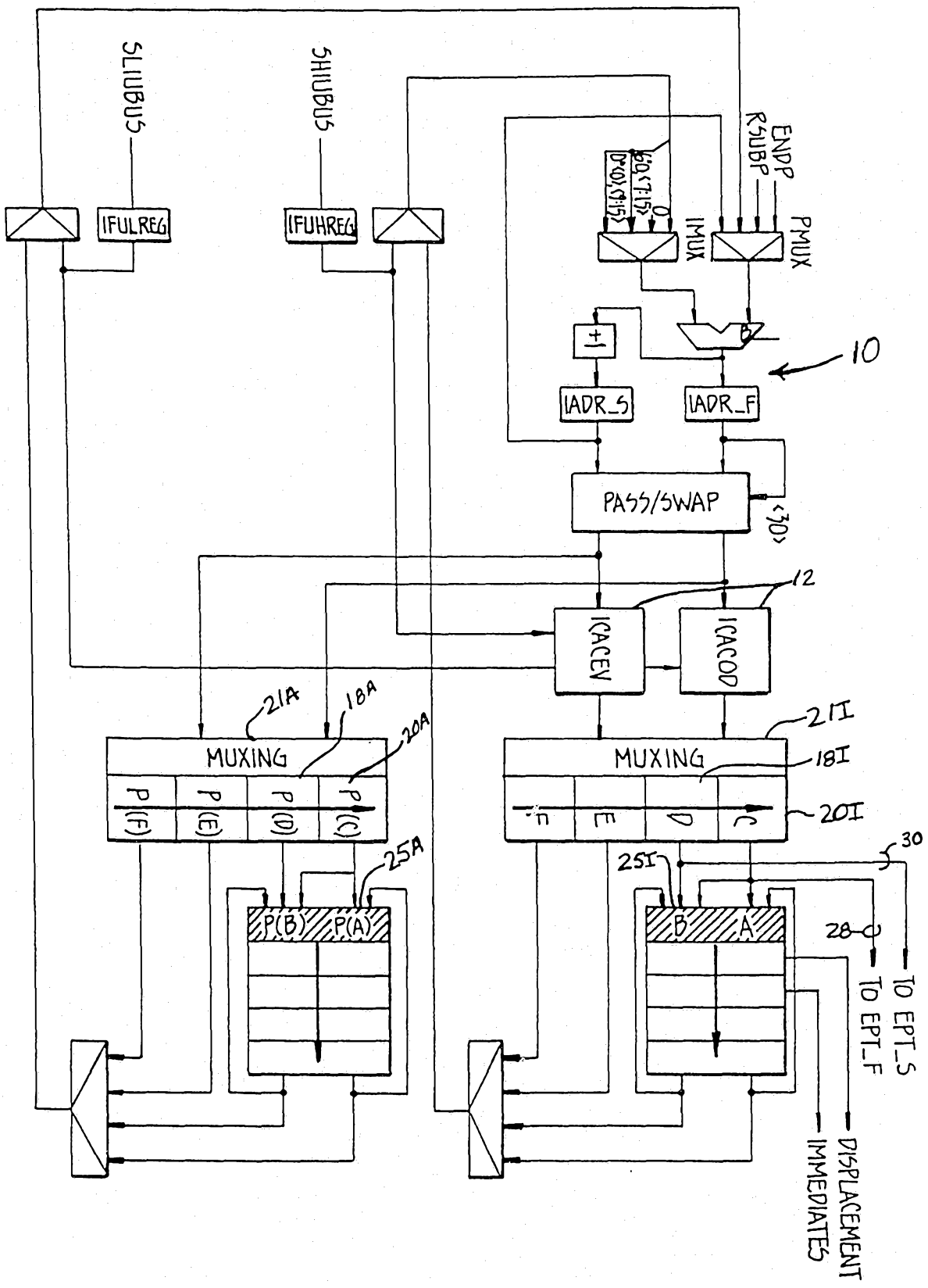
INSTRUCTION CACHE, QUEUE, AND PIPELINE

FIG._11A.



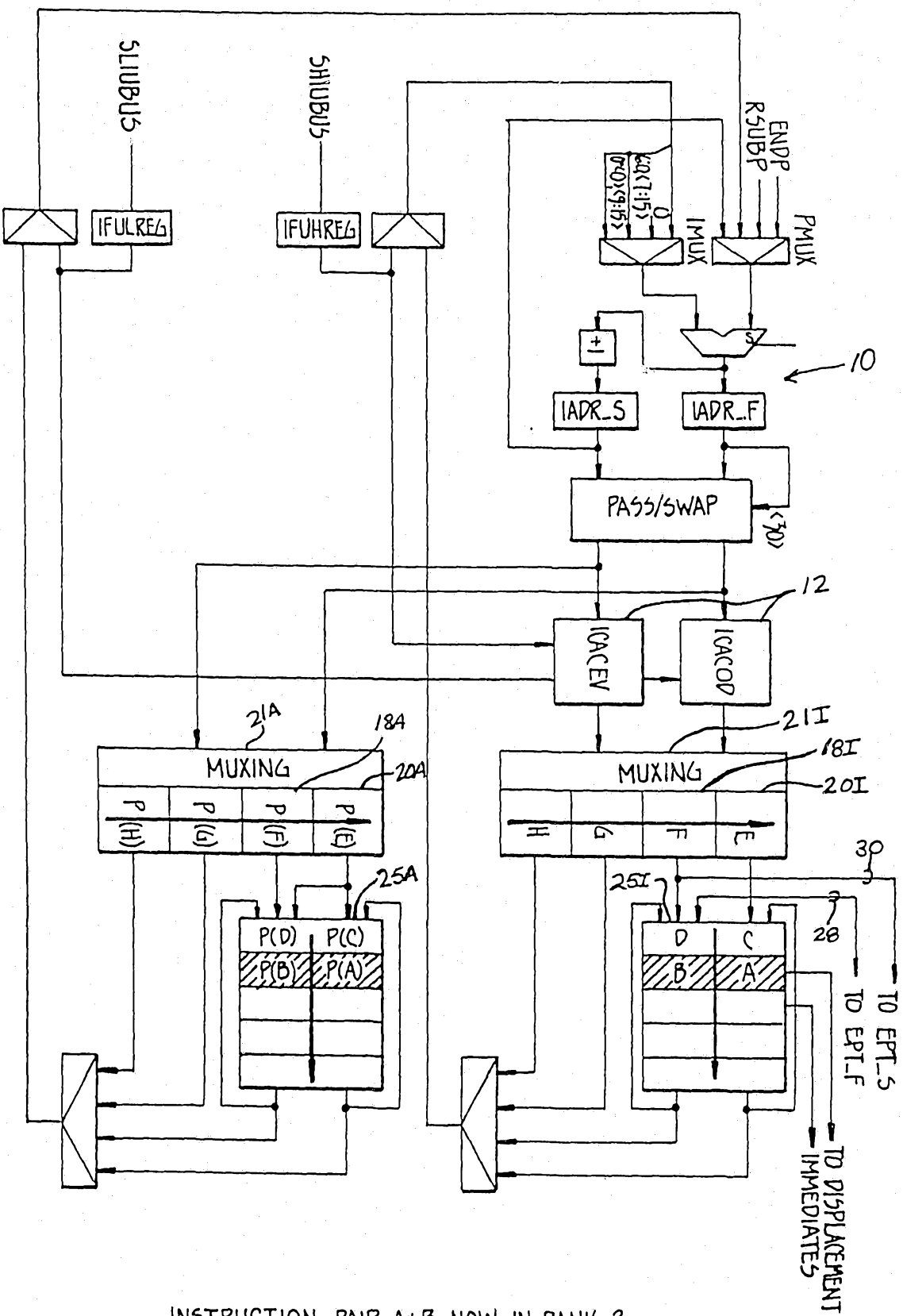
INSTRUCTION PAIR A+B READY TO ISSUE

FIG._11B.



INSTRUCTION PAIR A+B JUST ISSUED

FIG._11C.



INSTRUCTION PAIR A+B NOW IN RANK 2

FIG._11D.

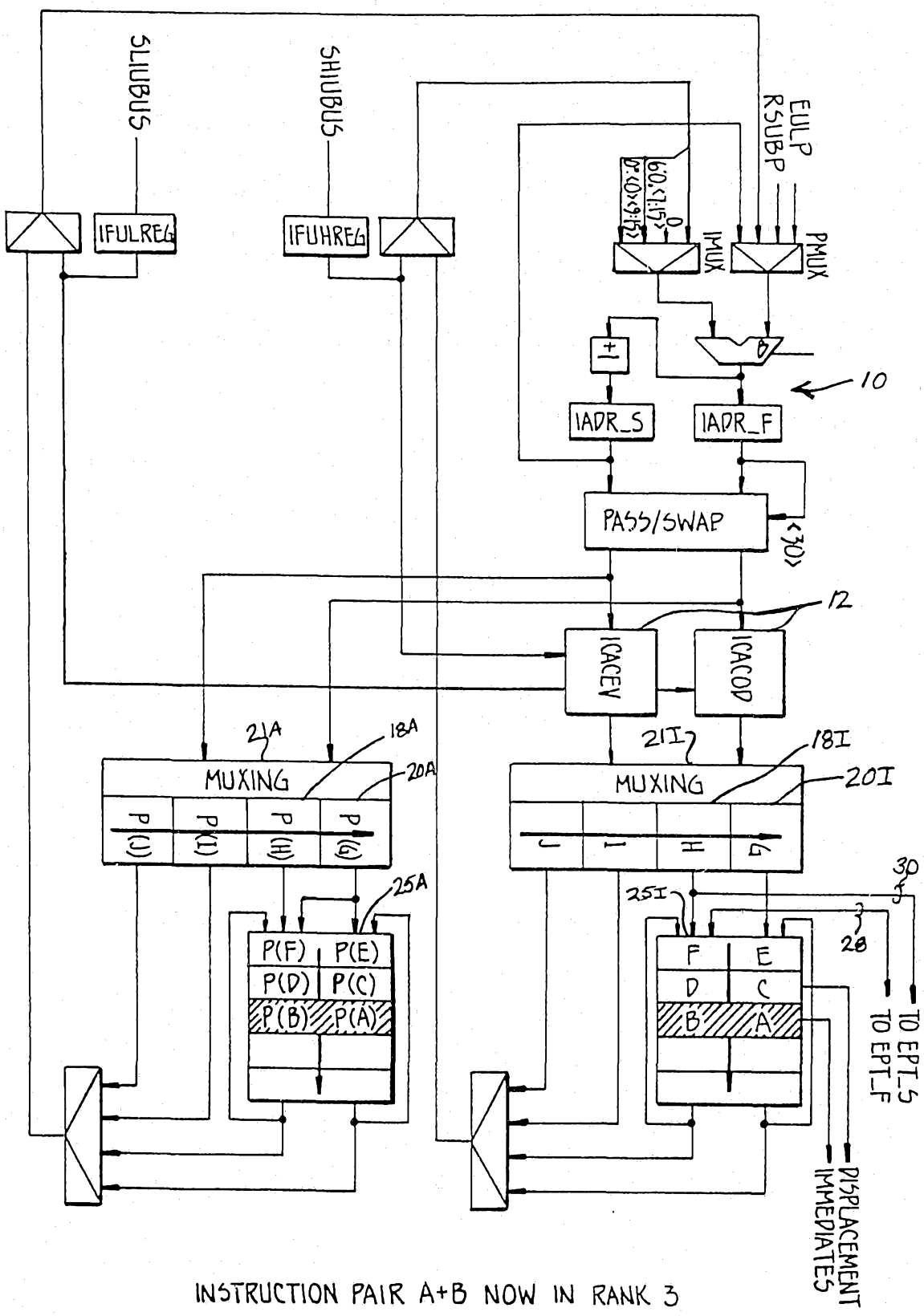
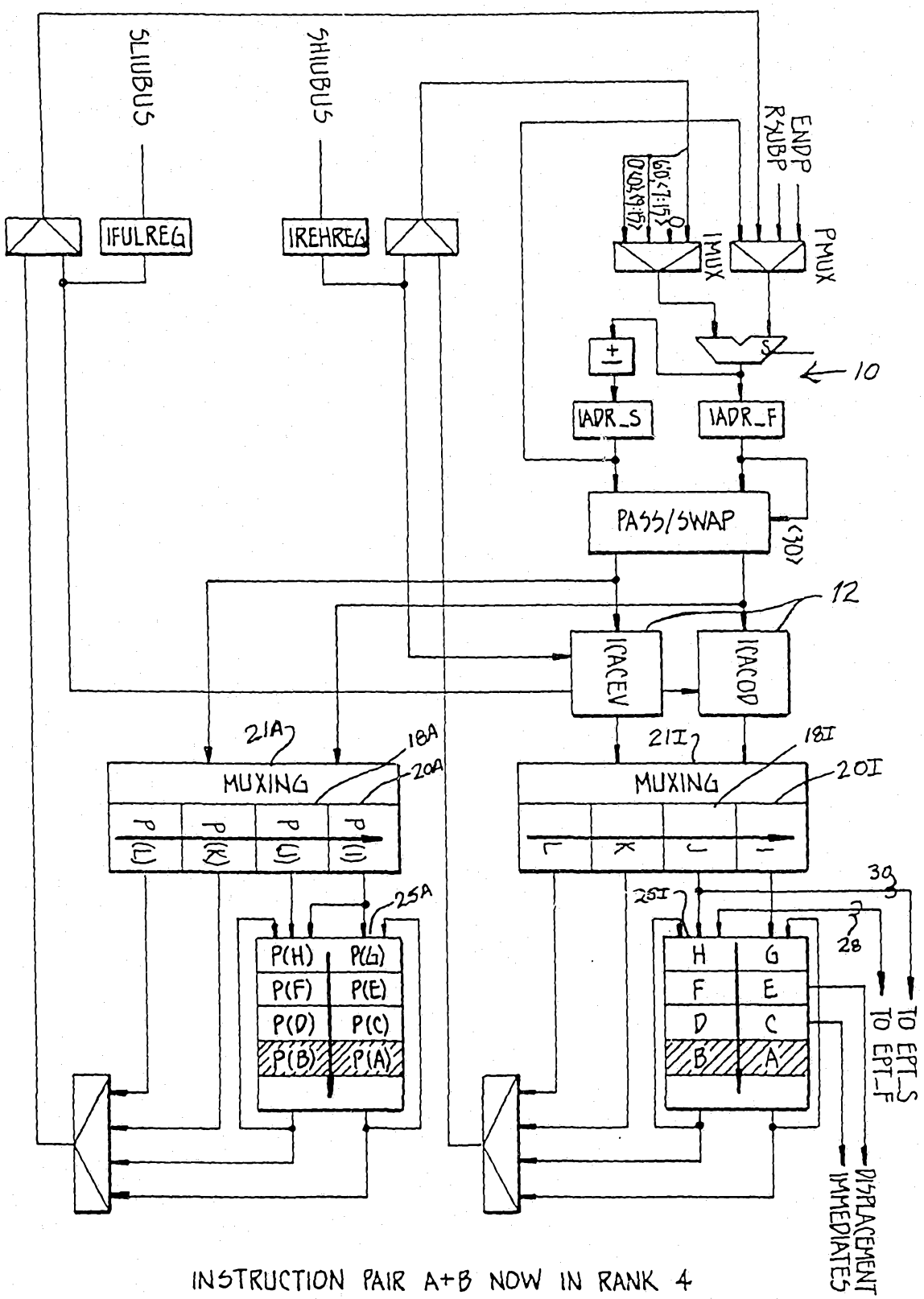
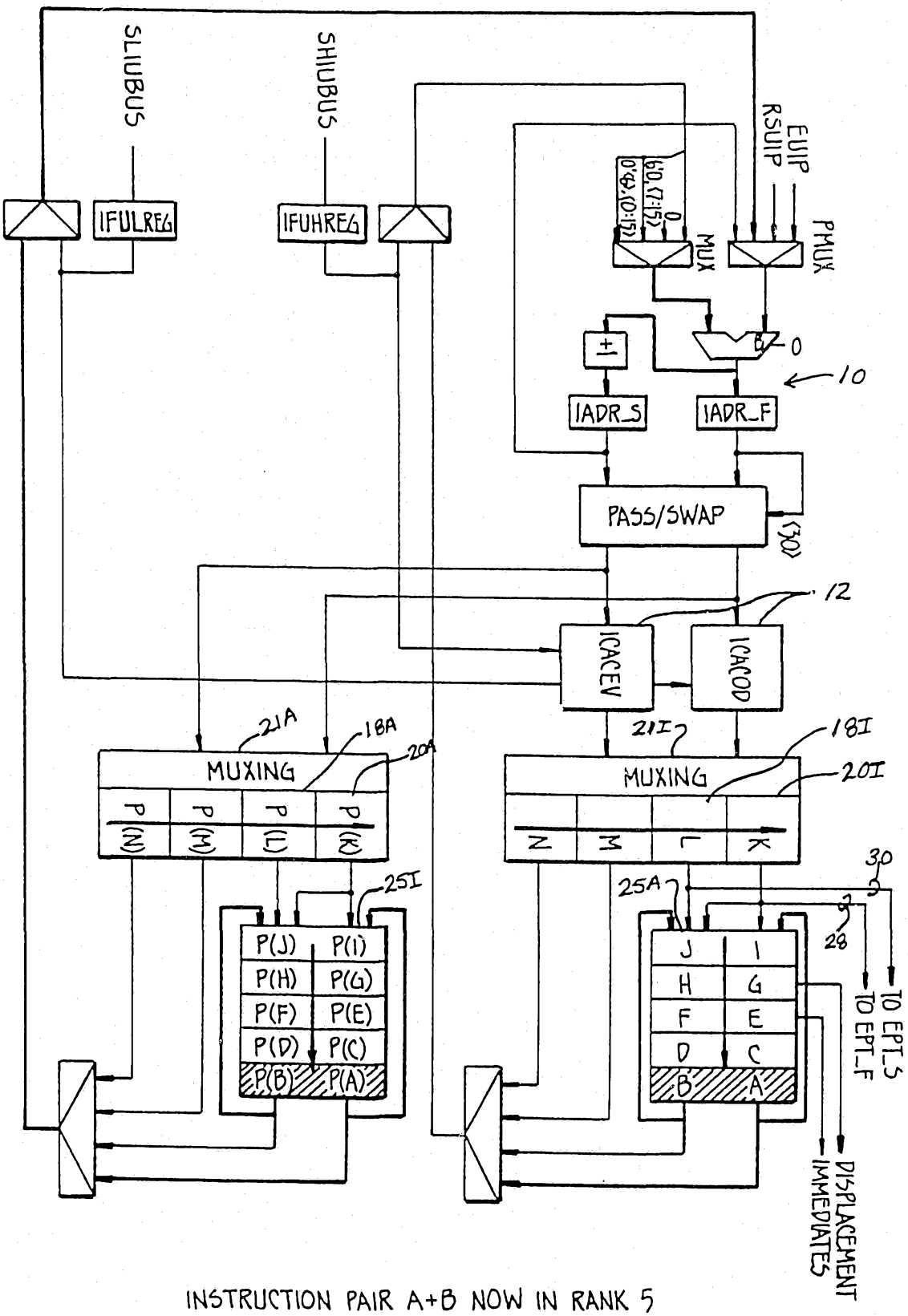

SLUBUS
SHIUBUS
EULP
RSUBP
IFULREG
IFUHREG
IMUX
PMUX
+1
IADR_S
IADR_F
10
PASS/SWAP
12
ICACEV
ICACOD
21A
18A
20A
MUXING
P(L)
P(K)
P(H)
P(G)
21I
18I
20I
MUXING
J
I
H
G
25A
P(F)
P(E)
P(D)
P(C)
P(B)
P(A)
25I
30
28
F
E
D
C
B
A
TO EPT_S
TO EPT_F
DISPLACEMENT
IMMEDIATES
INSTRUCTION PAIR A+B NOW IN RANK 3

FIG_11E.



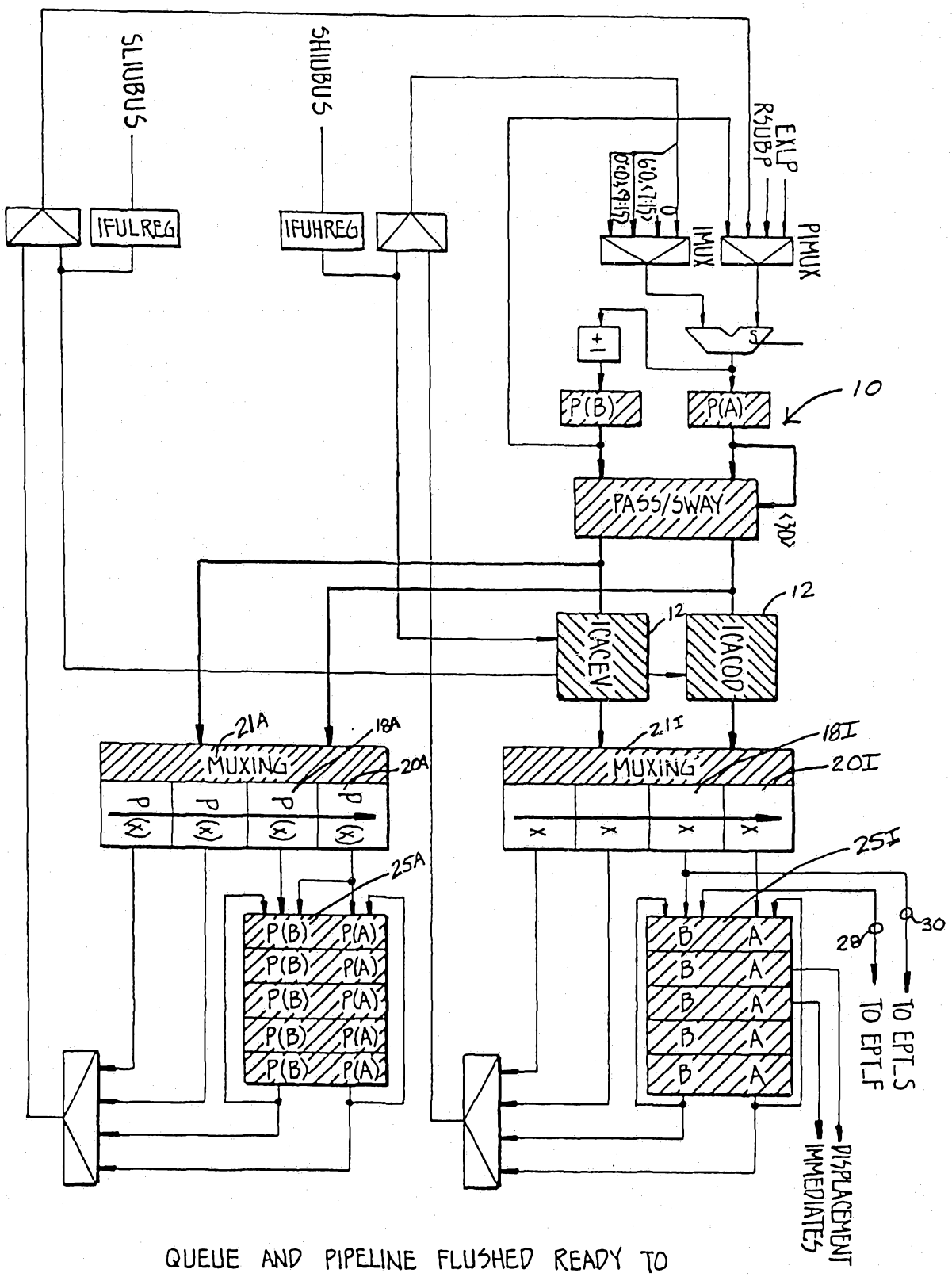
INSTRUCTION PAIR A+B NOW IN RANK 4
EXCEPTION CONDITIONS EVALUATED

FIG._11F.



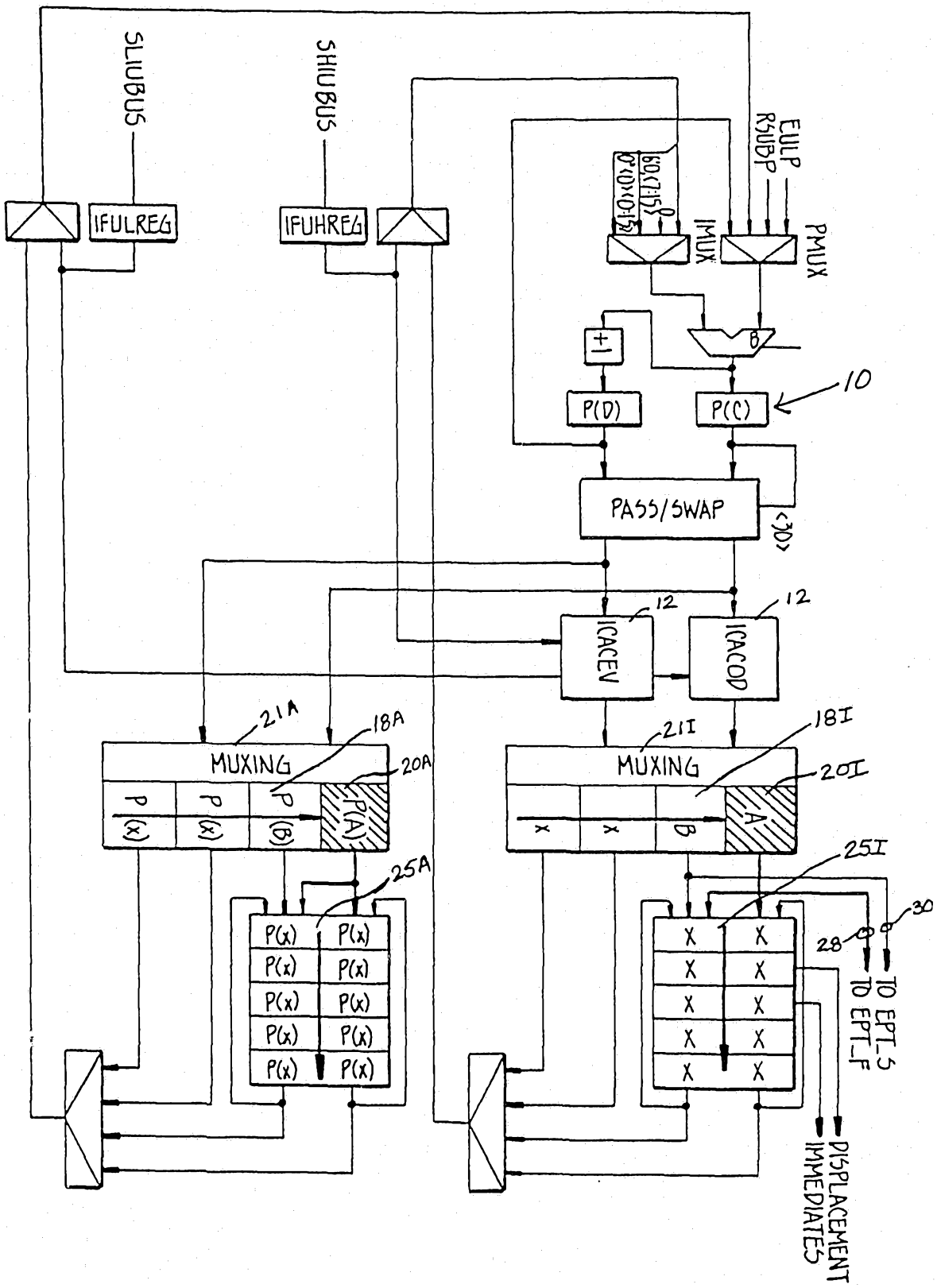
INSTRUCTION PAIR A+B NOW IN RANK 5
TIME TO DO UNPAIRED RESTART

FIG._11G.



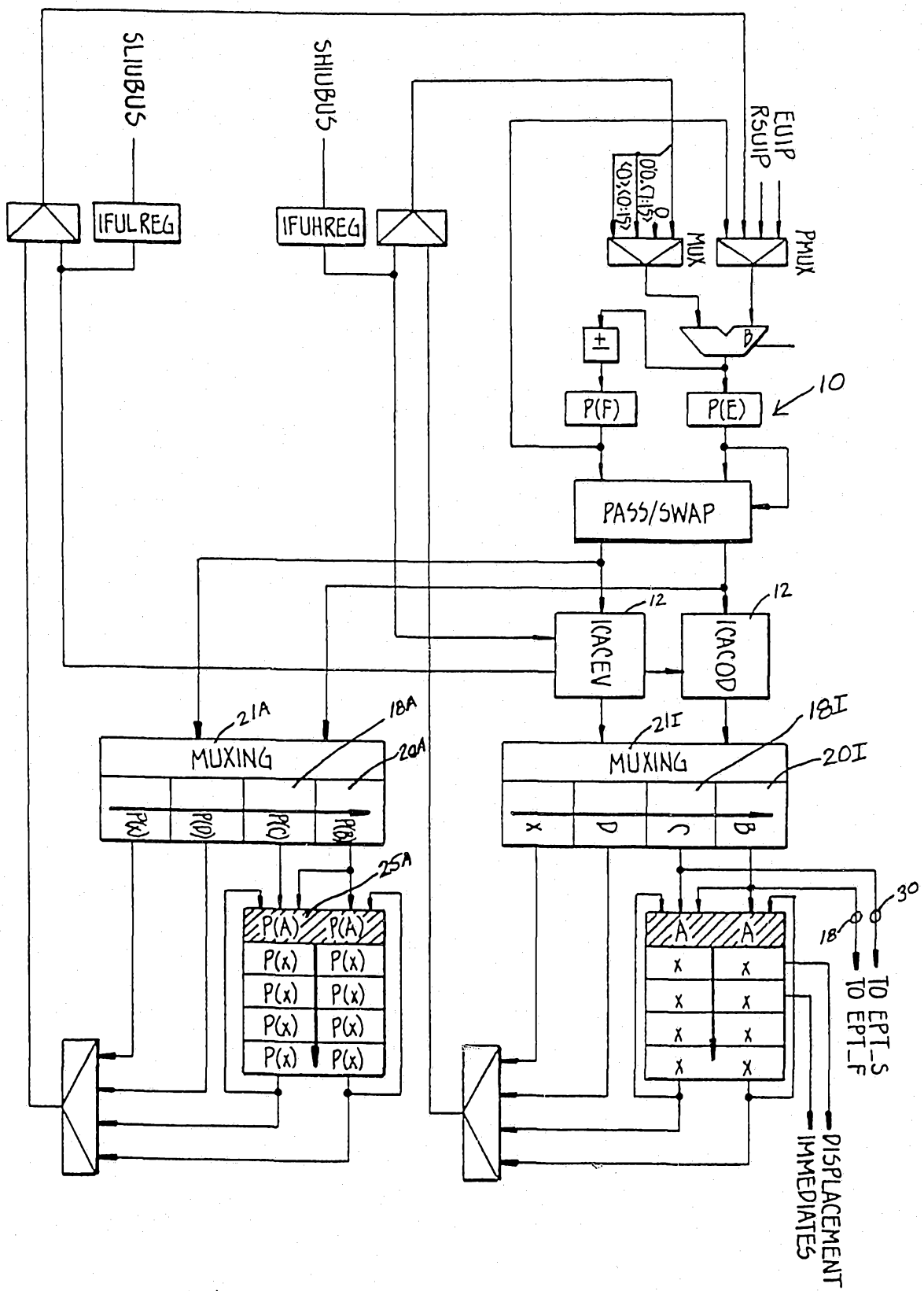
QUEUE AND PIPELINE FLUSHED READY TO FETCH INSTRUCTION AGAIN

FIG._11H.



READY TO ISSUE INSTRUCTION AGAIN
ISSUE AS A SINGLE THIS TIME

FIG._11I.



SINGLE INSTRUCTION BEGINS EXECUTION
HANDLE EXCEPTION THIS TIME

FIG._11J.

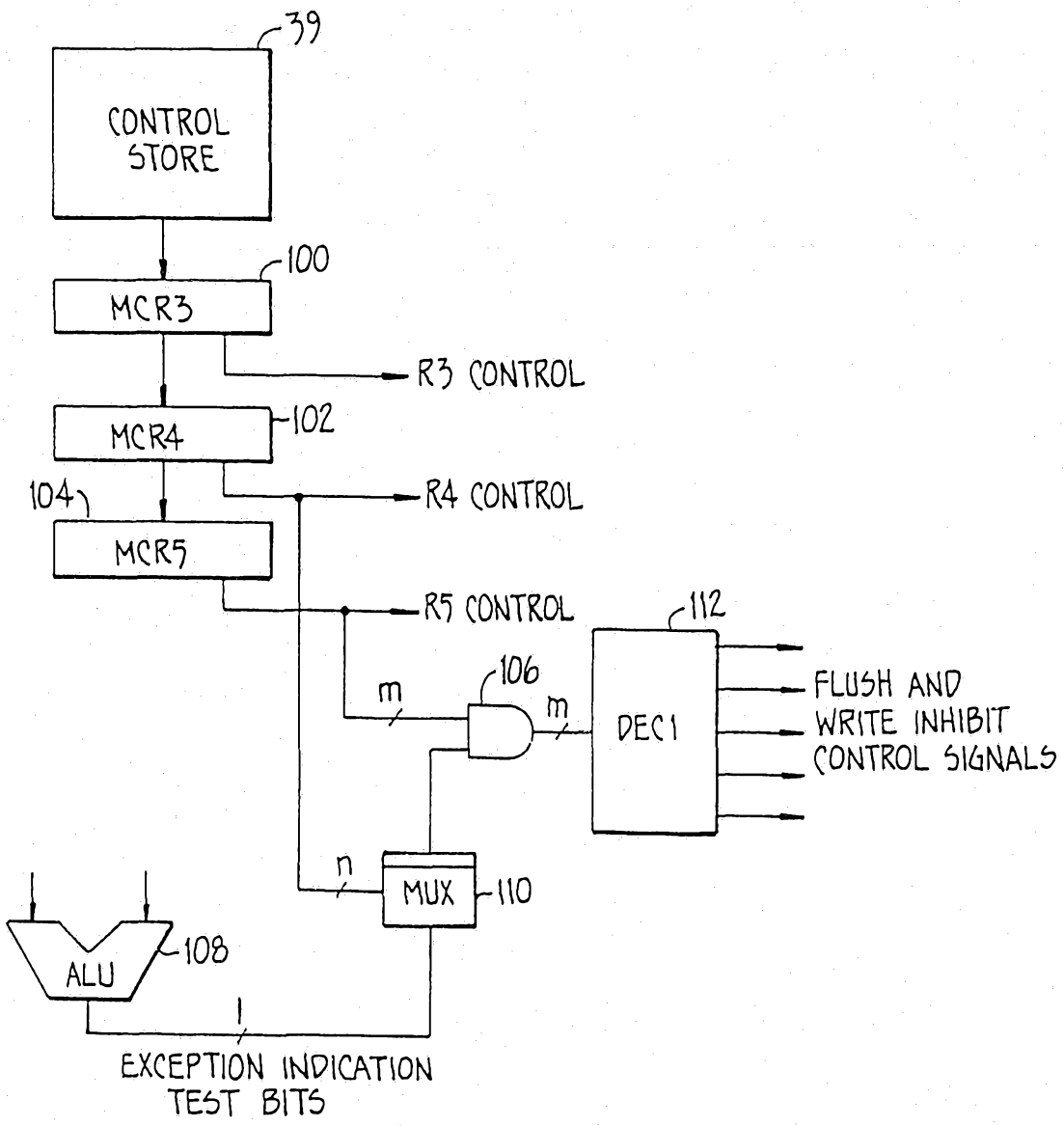

39
CONTROL STORE
100
MCR3
R3 CONTROL
102
MCR4
104
R4 CONTROL
MCR5
R5 CONTROL
112
m
106
m
DEC1
FLUSH AND
WRITE INHIBIT
CONTROL SIGNALS
n
MUX
110
ALU
108
1
EXCEPTION INDICATION
TEST BITS

FIG. 11K.

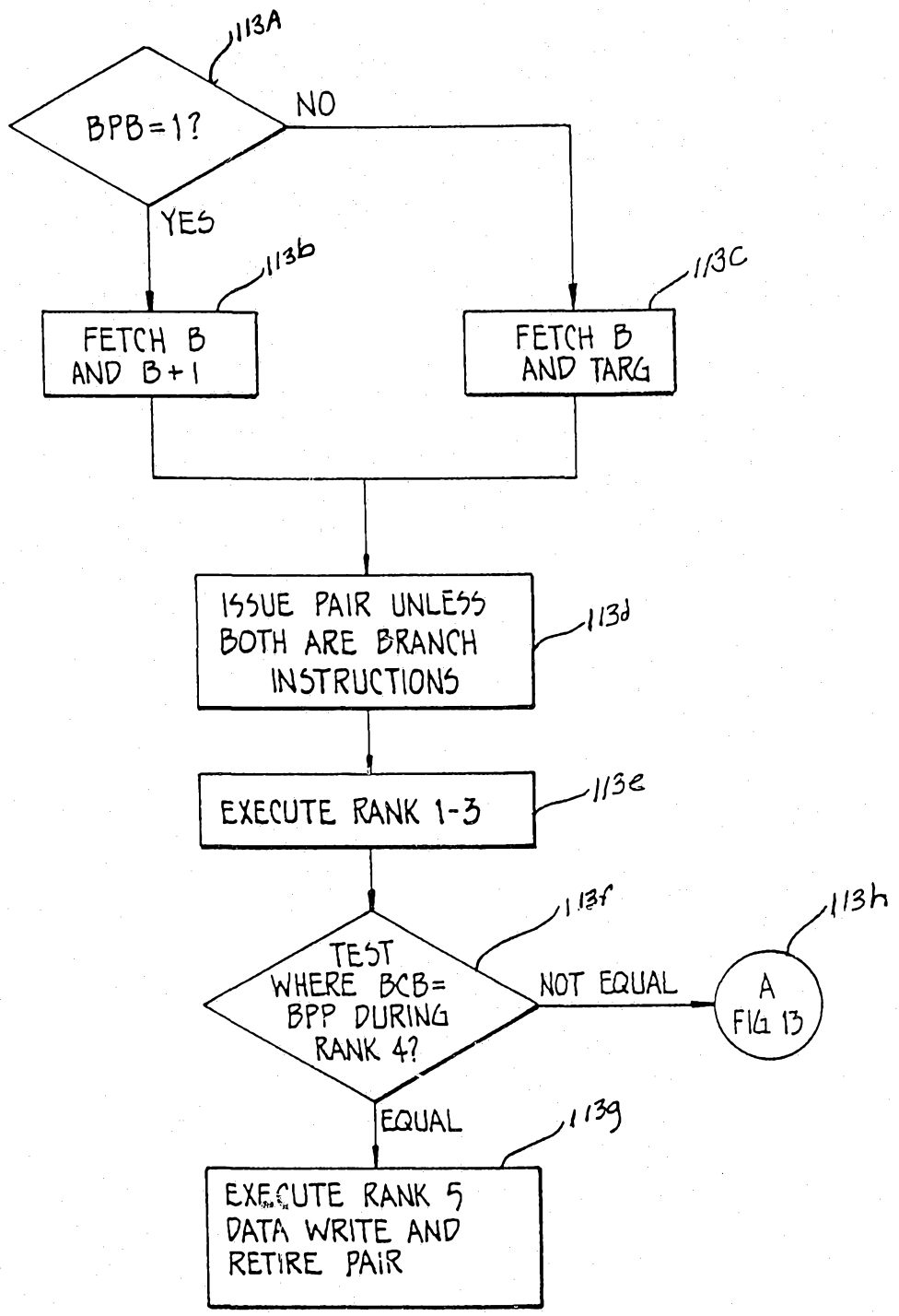

113A
BPB=1?
NO
YES
113b
FETCH B AND B+1
113C
FETCH B AND TARG
ISSUE PAIR UNLESS BOTH ARE BRANCH INSTRUCTIONS
113d
EXECUTE RANK 1-3
113e
113f
TEST WHERE BCB= BPP DURING RANK 4?
NOT EQUAL
113h
A FIG 13
EQUAL
113g
EXECUTE RANK 5 DATA WRITE AND RETIRE PAIR

FIG._12.

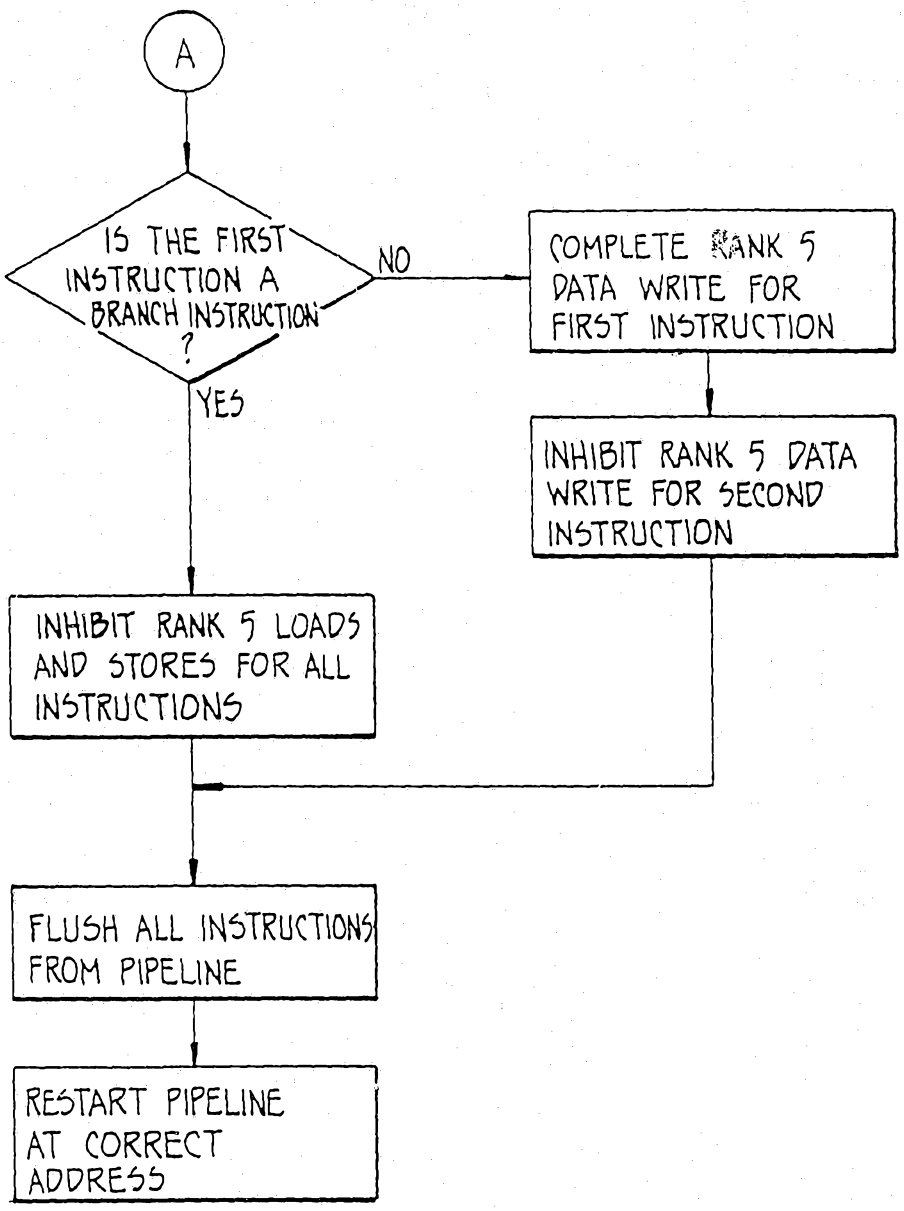

A
IS THE FIRST INSTRUCTION A BRANCH INSTRUCTION ?
NO
YES
COMPLETE RANK 5 DATA WRITE FOR FIRST INSTRUCTION
INHIBIT RANK 5 DATA WRITE FOR SECOND INSTRUCTION
INHIBIT RANK 5 LOADS AND STORES FOR ALL INSTRUCTIONS
FLUSH ALL INSTRUCTIONS FROM PIPELINE
RESTART PIPELINE AT CORRECT ADDRESS

FIG. 13.

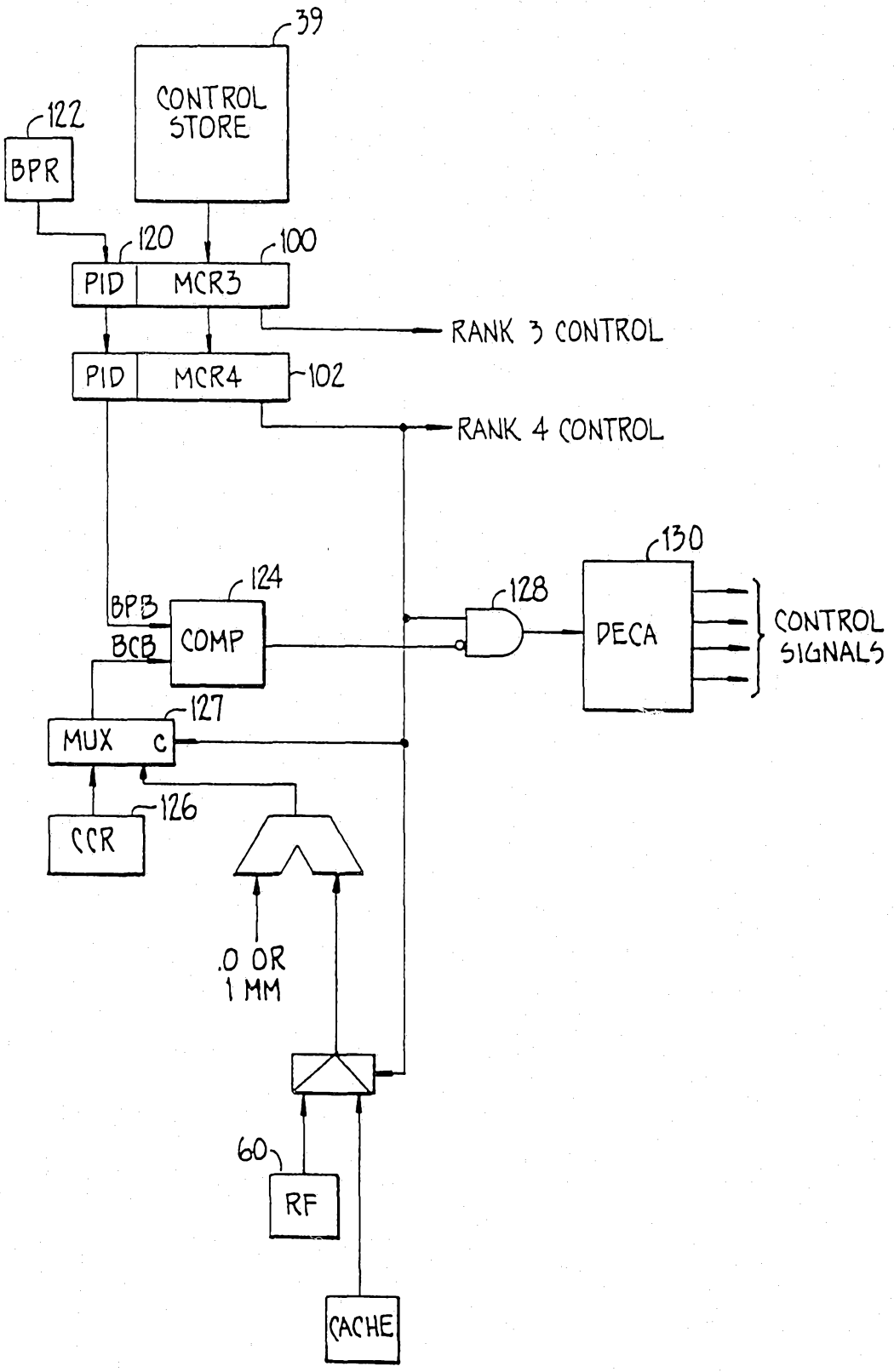

39
CONTROL STORE
122
BPR
120
100
PID
MCR3
RANK 3 CONTROL
PID
MCR4
102
RANK 4 CONTROL
130
124
128
BPB
BCB
COMP
DECA
CONTROL SIGNALS
127
MUX
C
126
CCR
.0 OR 1 MM
60
RF
CACHE

FIG._14.