US 20060117167A1

(19) United States
(12) Patent Application Publication
Evrard et al.
(10) Pub. No.: US 2006/0117167 A1
(43) Pub. Date: Jun. 1, 2006

(54) PROCESSING ACTIVITY MASKING IN A DATA PROCESSING SYSTEM

(76) Inventors: **Christophe Justin Evrard**, Letignet (FR); **Julie-Anne Francoise Marie Pruvost**, Antibes (FR)

Correspondence Address:
NIXON & VANDERHYE, PC
901 NORTH GLEBE ROAD, 11TH FLOOR
ARLINGTON, VA 22203 (US)

(21) Appl. No.: **10/527,812**

(22) PCT Filed: **Oct. 6, 2003**

(86) PCT No.: **PCT/GB03/04261**

(30) **Foreign Application Priority Data**

Dec. 12, 2002	(GB)	0229068.2
Feb. 5, 2003	(GB)	0302646.5
Feb. 5, 2003	(GB)	0302650.7
Apr. 4, 2003	(GB)	0307823.5

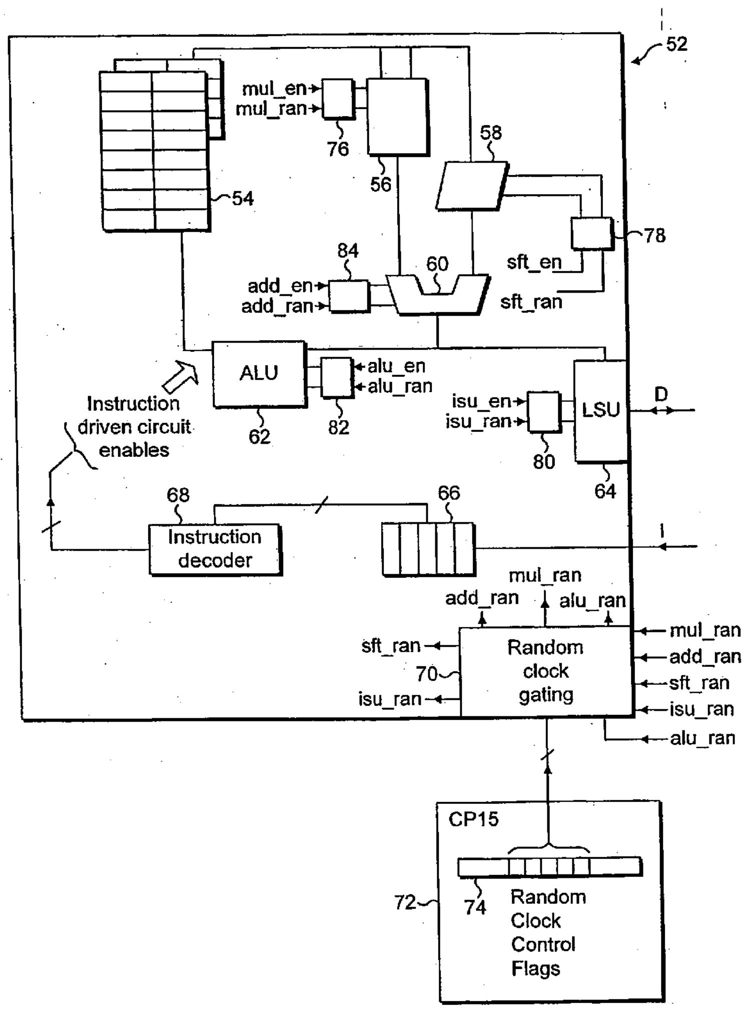
Publication Classification

(51) **Int. Cl.**
G06F 9/30 (2006.01)

(52) **U.S. Cl.** **712/208**

(57) **ABSTRACT**

Within a data processing systems supporting conditional write processing operations, a trash register is provided such that when non-write conditions are encountered a register write is made to the trash register rather than the data register specified by the conditional write operation. Thus the power signature associated with whether or not a register write does or does not occur is masked. The trash register activity may be programmable enabled and disabled by a configuration parameter stored within a system configuration register.



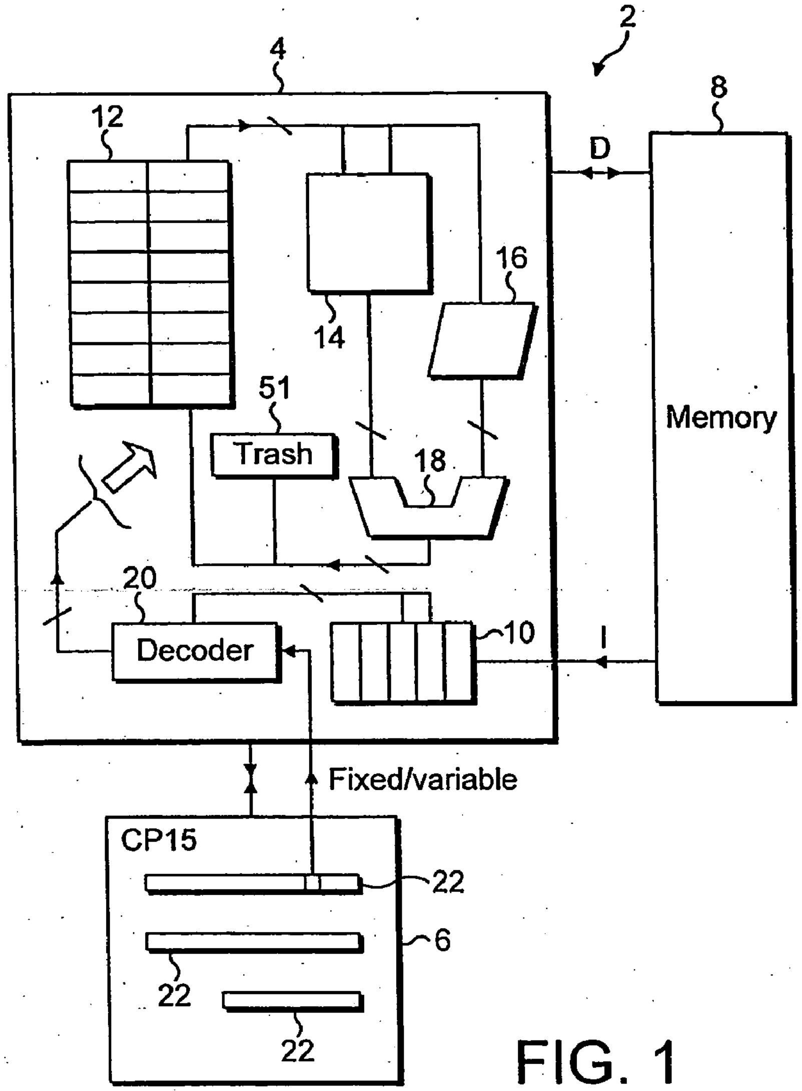

2
4
8
12
D
16
14
Memory
51
Trash
18
20
Decoder
10
I
Fixed/variable
CP15
22
6
22
22

FIG. 1

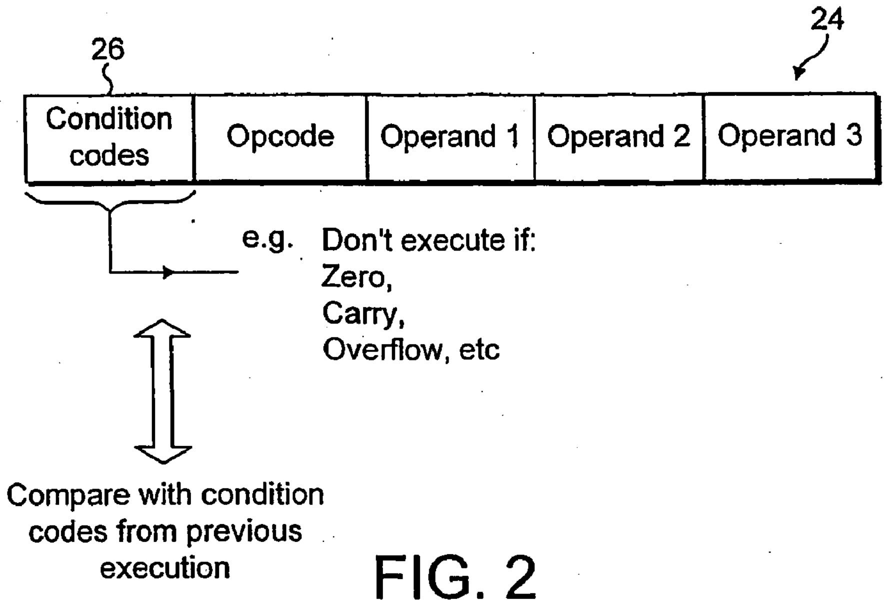

24
26
Condition codes
Opcode
Operand 1
Operand 2
Operand 3
e.g. Don't execute if:
Zero,
Carry,
Overflow, etc
Compare with condition codes from previous execution

FIG. 2

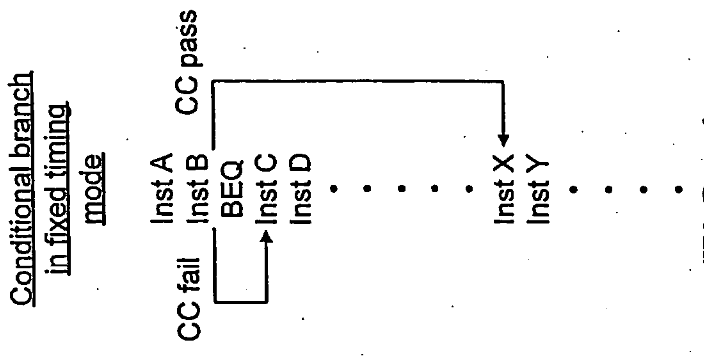

Conditional branch in fixed timing mode
CC fail
CC pass
Inst A
Inst B
BEQ
Inst C
Inst D
Inst X
Inst Y

FIG. 4

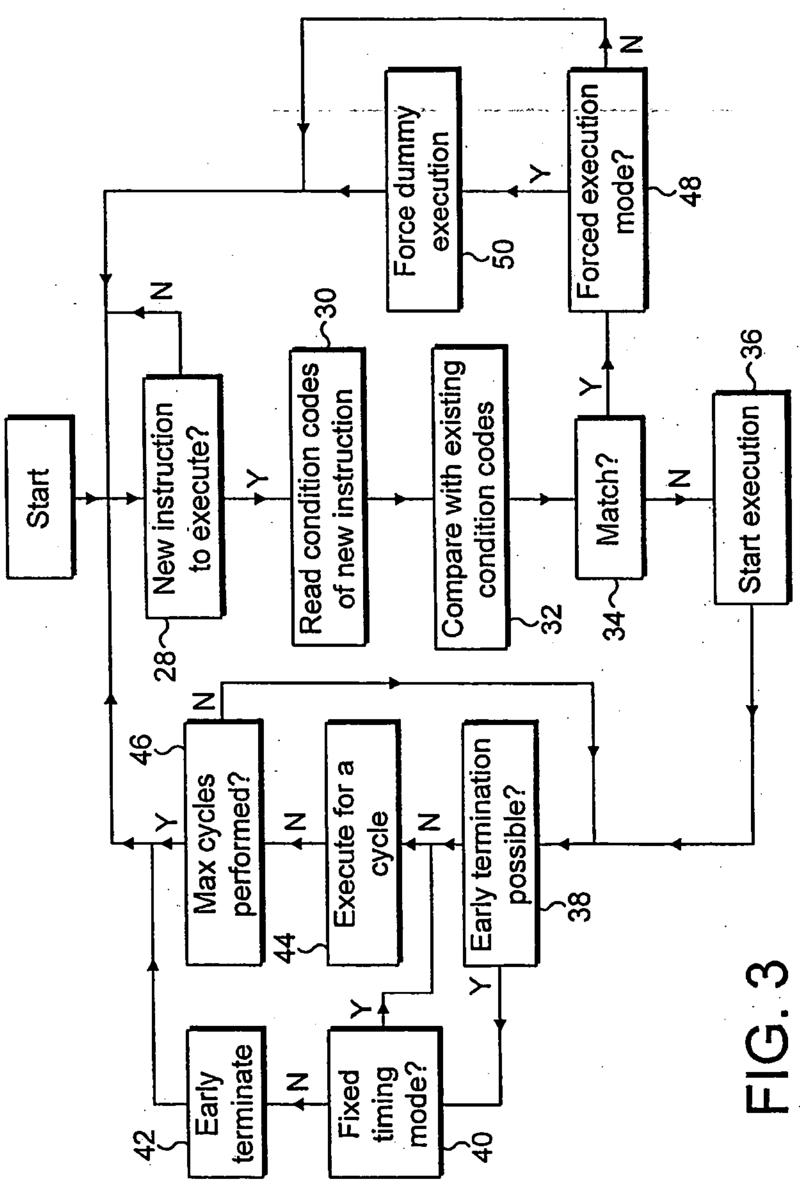

Start
New instruction to execute?
28
Y
N
Read condition codes of new instruction
30
Compare with existing condition codes
32
Match?
34
Y
N
Forced execution mode?
48
Y
N
Force dummy execution
50
Start execution
36
Early termination possible?
38
N
Y
Fixed timing mode?
40
Y
N
Early terminate
42
Execute for a cycle
44
N
Max cycles performed?
46
Y
N

FIG. 3

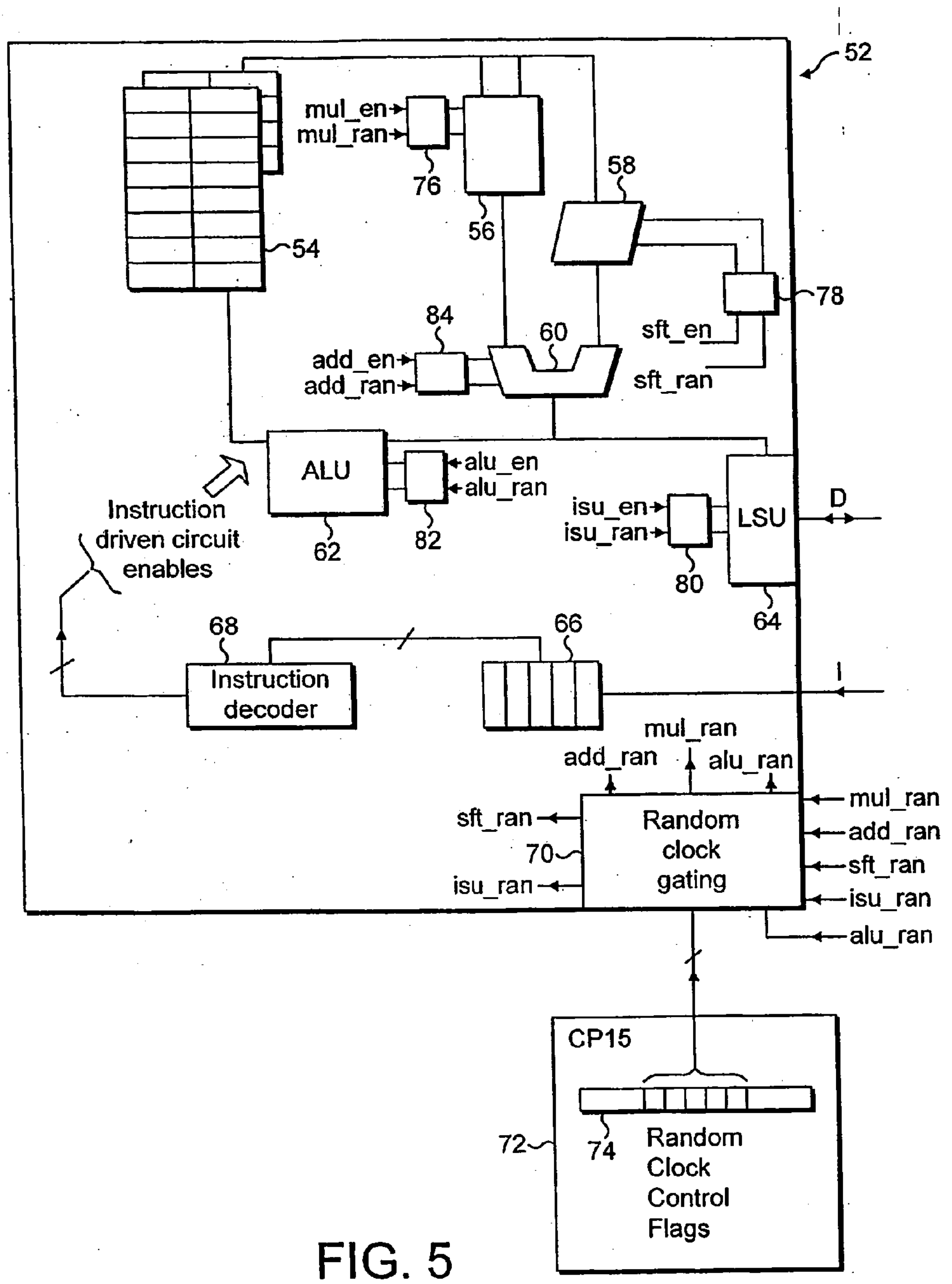

52
mul_en
mul_ran
76
56
58
54
78
84
60
sft_en
add_en
add_ran
sft_ran
ALU
alu_en
alu_ran
Instruction driven circuit enables
62
82
isu_en
isu_ran
LSU
D
80
64
68
66
Instruction decoder
I
mul_ran
add_ran
alu_ran
sft_ran
Random clock gating
mul_ran
add_ran
sft_ran
isu_ran
alu_ran
70
isu_ran
CP15
72
74
Random Clock Control Flags

FIG. 5

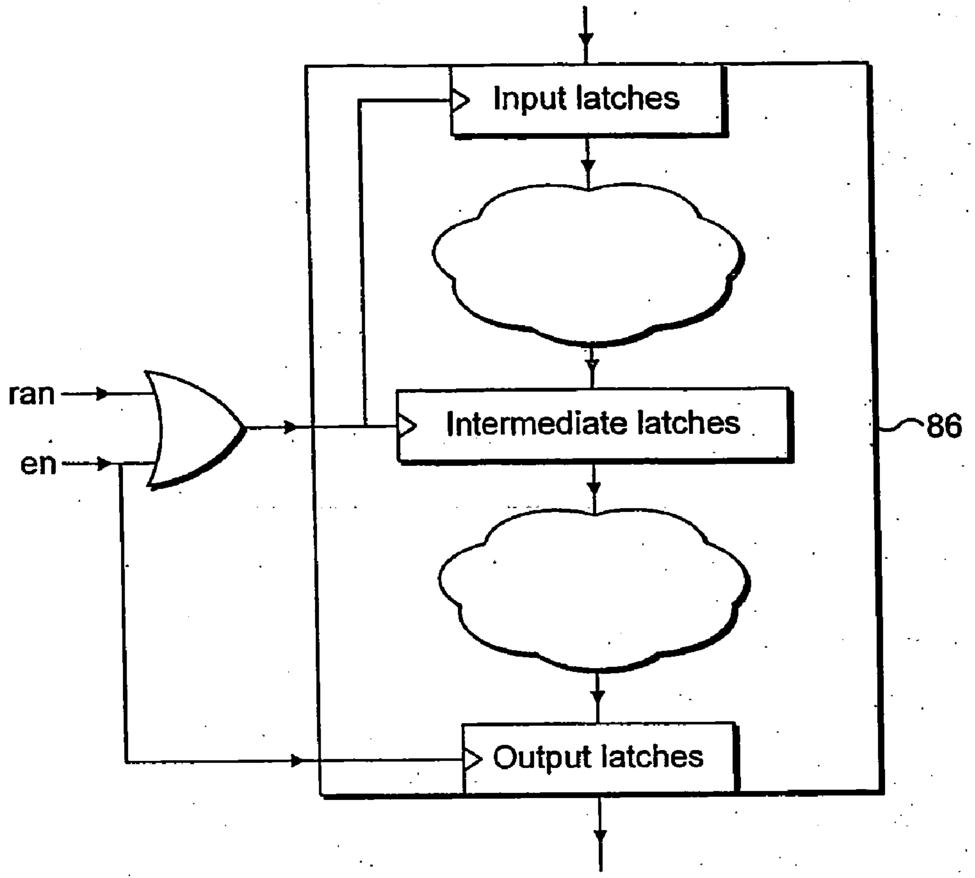

Input latches
Intermediate latches
Output latches
ran
en
86

FIG. 6

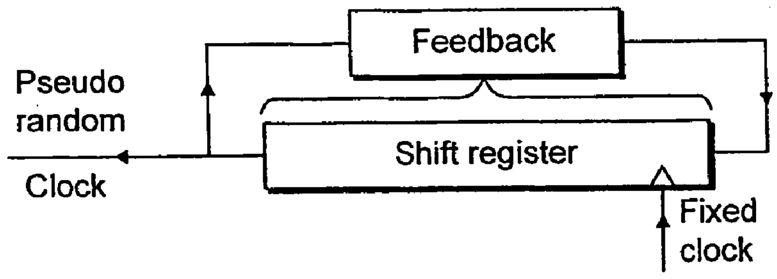

Feedback
Shift register
Pseudo random
Clock
Fixed clock

FIG. 7

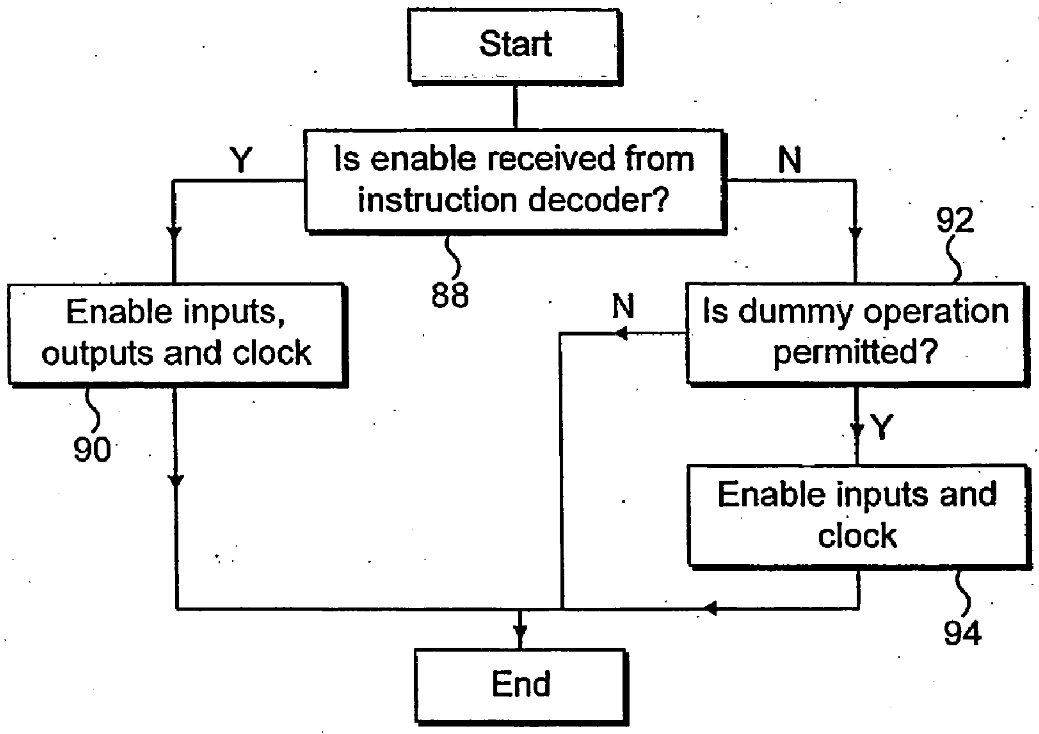

Start
Is enable received from instruction decoder?
Y
N
88
Enable inputs, outputs and clock
90
92
Is dummy operation permitted?
N
Y
Enable inputs and clock
94
End

FIG. 8

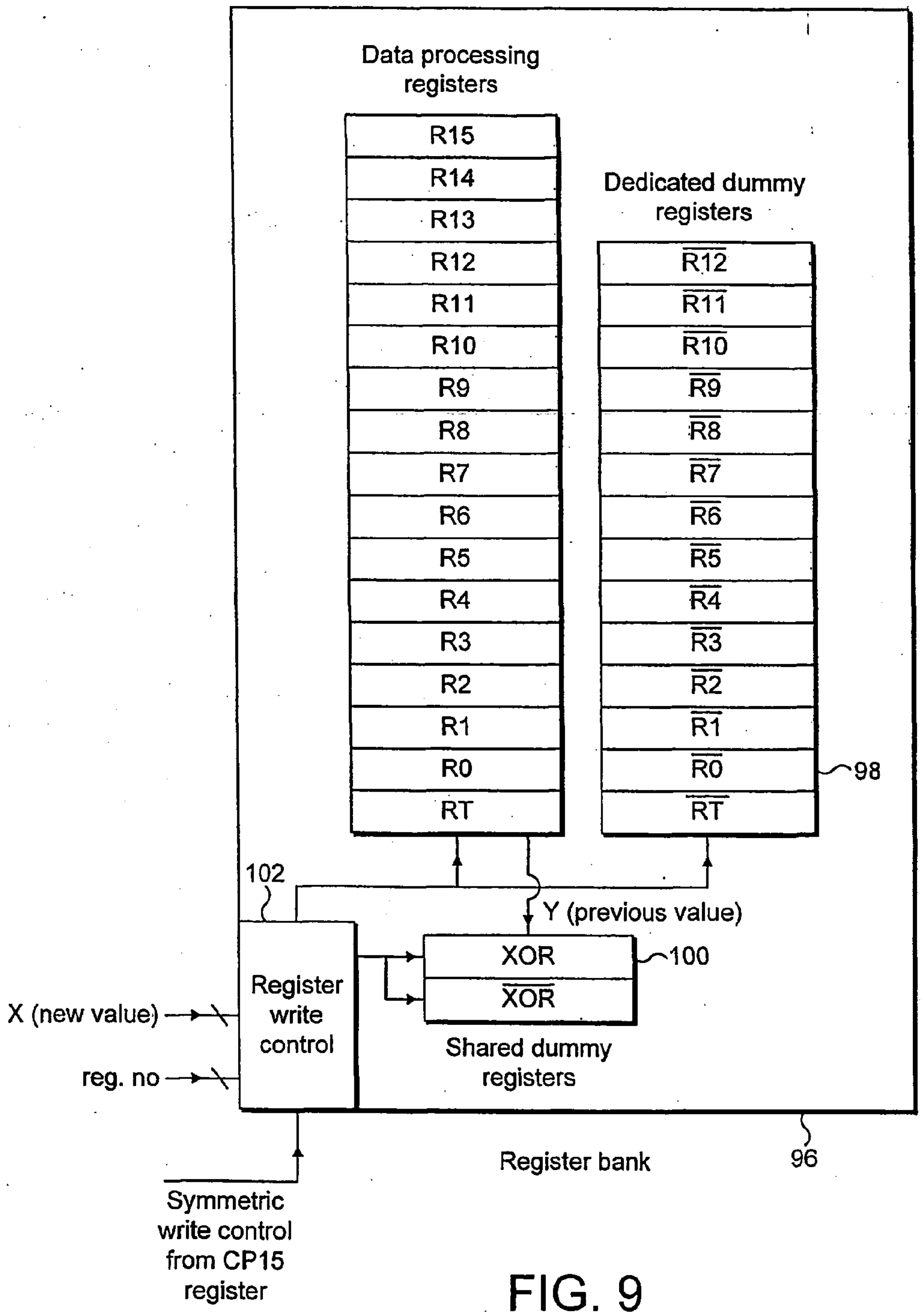

Data processing registers
R15
R14
R13
R12
R11
R10
R9
R8
R7
R6
R5
R4
R3
R2
R1
R0
RT
Dedicated dummy registers
$\overline{R12}$
$\overline{R11}$
$\overline{R10}$
$\overline{R9}$
$\overline{R8}$
$\overline{R7}$
$\overline{R6}$
$\overline{R5}$
$\overline{R4}$
$\overline{R3}$
$\overline{R2}$
$\overline{R1}$
$\overline{R0}$
$\overline{RT}$
98
102
Y (previous value)
XOR
$\overline{XOR}$
100
Register write control
X (new value)
reg. no
Shared dummy registers
Register bank
96
Symmetric write control from CP15 register

FIG. 9

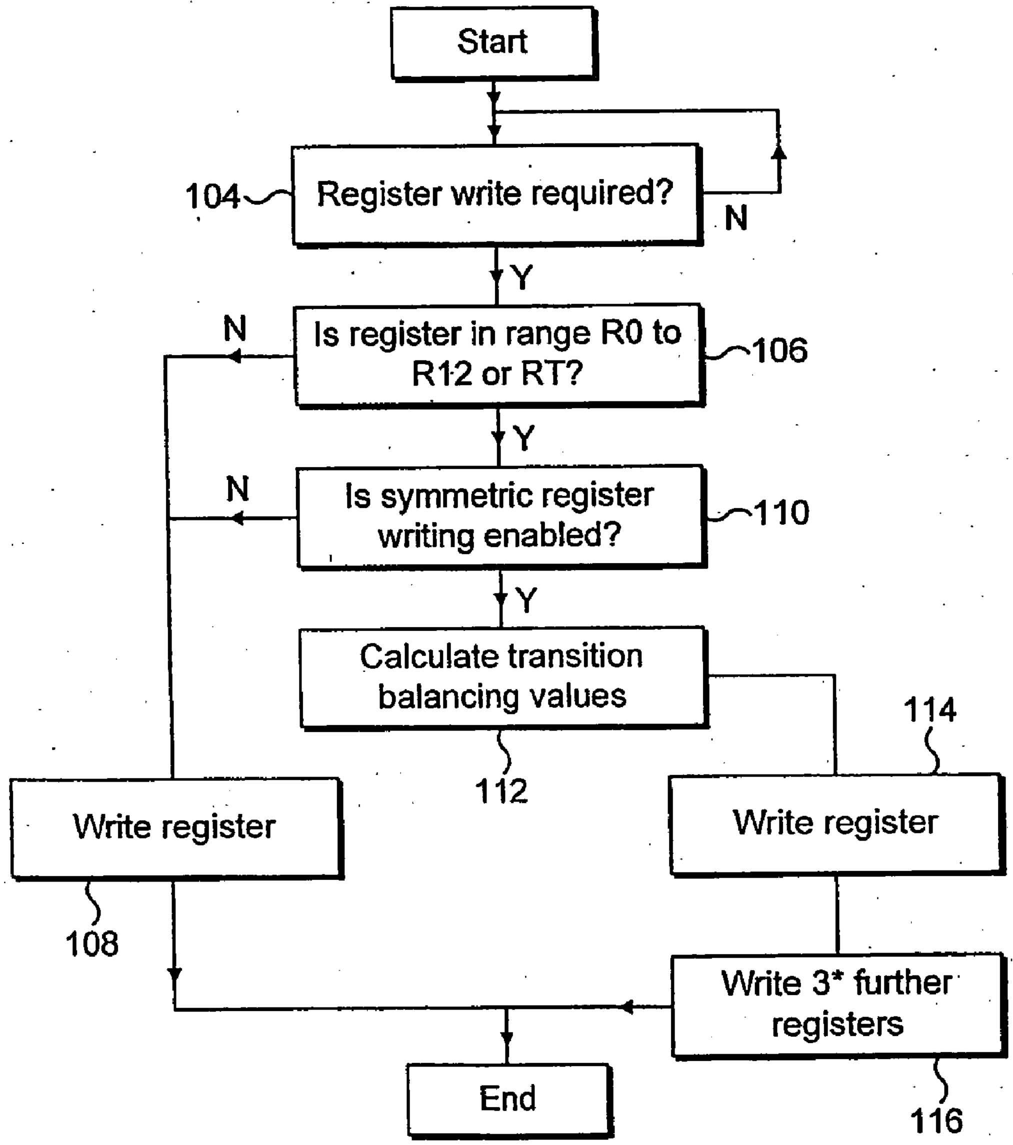

Start
Register write required?
104
N
Y
Is register in range R0 to R12 or RT?
106
N
Y
Is symmetric register writing enabled?
110
N
Y
Calculate transition balancing values
112
114
Write register
108
Write register
Write 3* further registers
116
End

FIG. 10

								Transitions			
Rn[i]t	Rn[i]t+1	$\overline{\text{Rn[i]t}}$	$\overline{\text{Rn[i]t+1}}$	Rd[i]t	Rd[i]t+1	$\overline{\text{Rd[i]t}}$	$\overline{\text{Rd[i]t+1}}$	Rn	$\overline{\text{Rn}}$	Rd	$\overline{\text{Rd}}$
0	0	1	1	0	1	1	0	—	—	↑	↓
0	0	1	1	1	0	0	1	—	—	↓	↑
0	1	1	0	0	0	1	1	↑	↓	—	—
0	1	1	0	1	1	0	0	↑	↓	—	—
1	0	0	1	0	0	1	1	↓	↑	—	—
1	0	0	1	1	1	0	0	↓	↑	—	—
1	1	0	0	0	1	1	0	—	—	↑	↓
1	1	0	0	1	0	0	1	—	—	↓	↑

$$\text{Rd[i]t+1} = \left(\overline{\text{Rn[i]t XOR Rn[i]t+1}}\right) \text{ XOR Rd[i]t}$$

FIG. 11

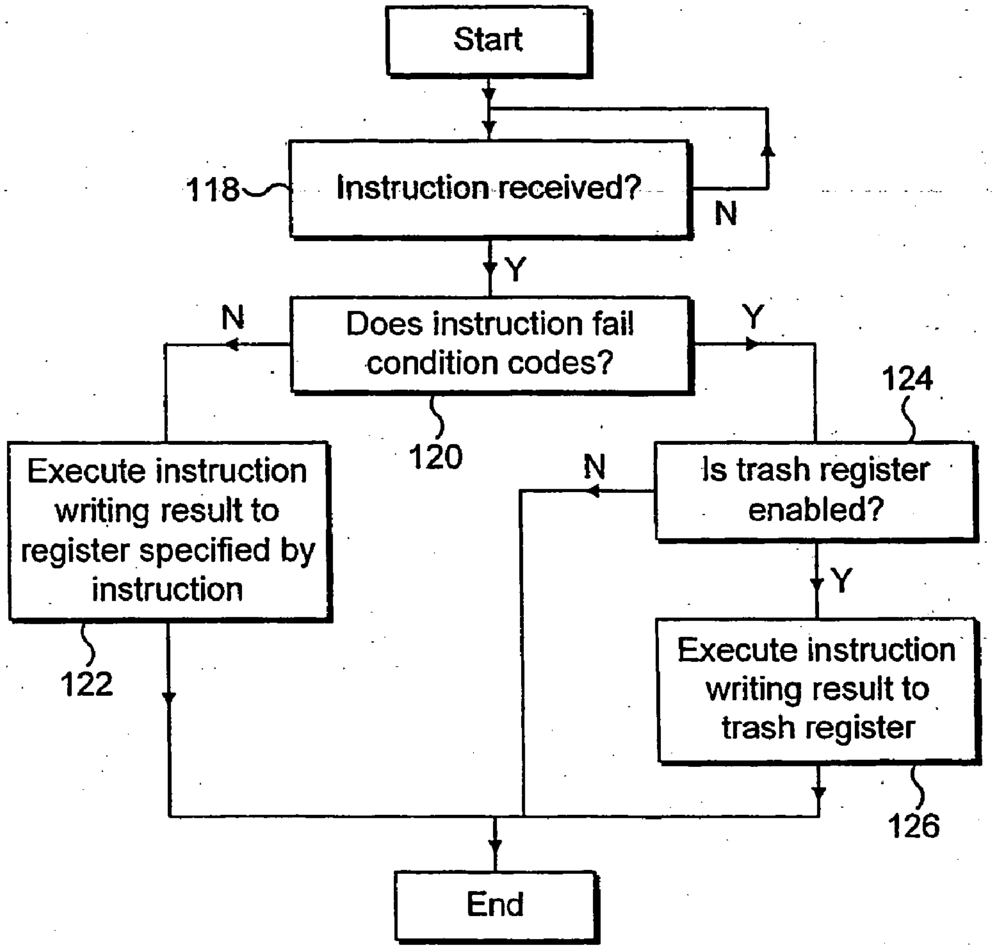

Start
118
Instruction received?
N
Y
Does instruction fail condition codes?
N
Y
120
124
Execute instruction writing result to register specified by instruction
122
N
Is trash register enabled?
Y
Execute instruction writing result to trash register
126
End

FIG. 12

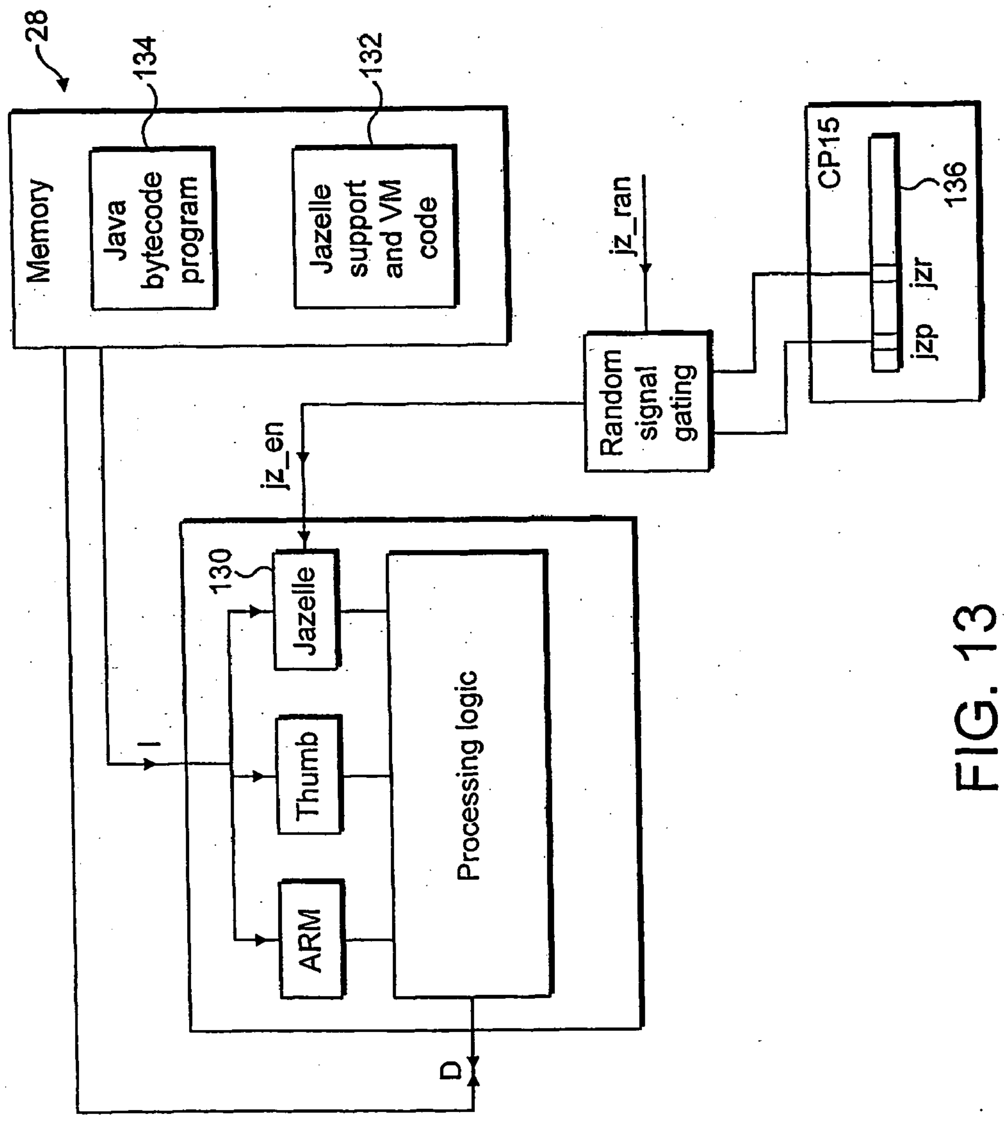

28
Memory
134
Java bytecode program
132
Jazelle support and VM code
jz_ran
CP15
136
Random signal gating
jzr
jzp
jz_en
130
Jazelle
Thumb
ARM
Processing logic
I
D

FIG. 13

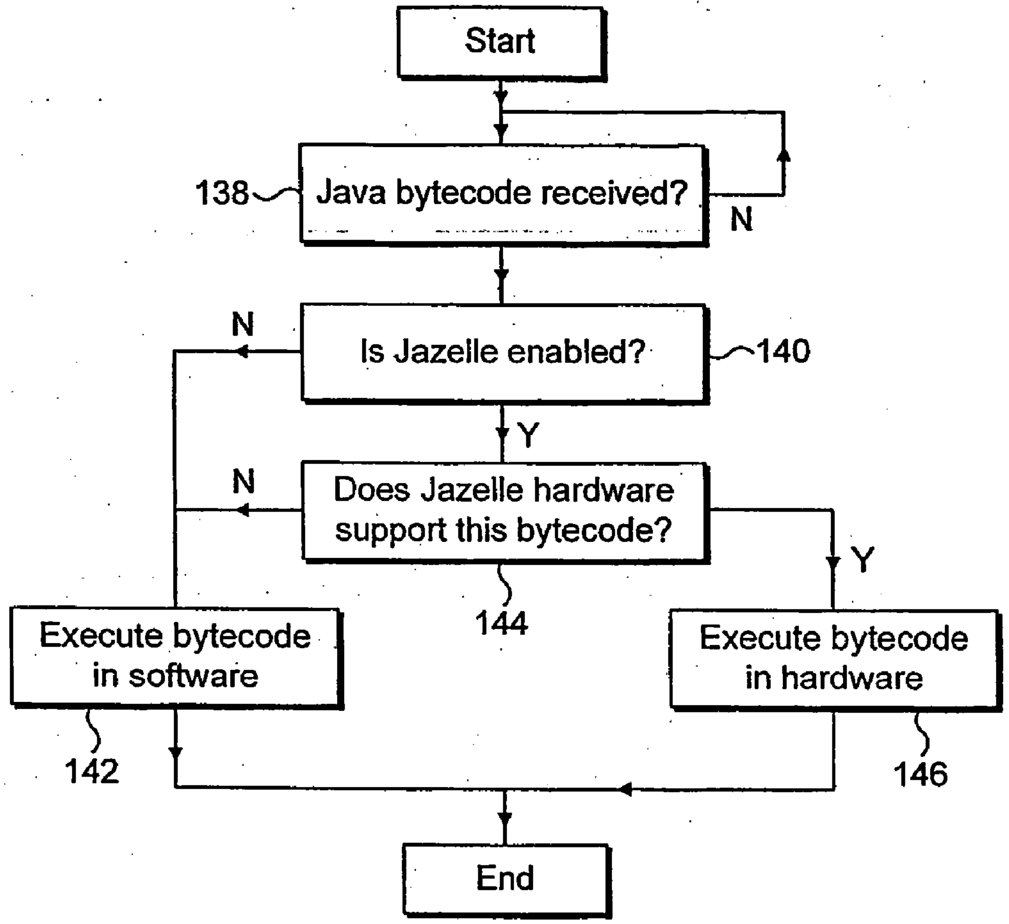

Start
138
Java bytecode received?
N
Is Jazelle enabled?
140
N
Y
Does Jazelle hardware support this bytecode?
N
Y
144
Execute bytecode in software
142
Execute bytecode in hardware
146
End

FIG. 14

PROCESSING ACTIVITY MASKING IN A DATA PROCESSING SYSTEM

[0001] This invention relates to the field of data processing systems. More particularly, this invention relates to the masking of processing activity within data processing systems, for example, in order to increase security.

[0002] It is known to provide data processing systems which manipulate secure data and for which it is desirable to ensure a high degree of security. As an example, it is known to provide smart cards which include a data processing system which manipulates secure data, such as secret cryptographic keys, and this data must be kept secret in order to prevent fraud.

[0003] Known ways of attacking the security of such systems include timing analysis and power analysis. By observing the tiling behaviour and/or the power consumption behaviour of such a system in response to inputs, information concerning the processing being performed and the data being manipulated can be determined in a way that can compromise security. It is strongly advantageous to provide resistance against such security attacks.

[0004] Viewed from one aspect the present invention provides apparatus for processing data, said apparatus comprising:

[0005] a result data value generating circuit operable to generate a result data value upon execution of a data processing operation; and

[0006] a data processing register to which said result data value is written; wherein

[0007] at least one data processing operation executed by said result data generating circuit is a conditional write data processing operation for which a result data value is not written to said data processing register when non-write conditions are met; and further comprising

[0008] a trash register to which a result data value may be written upon execution of said conditional write data processing operation when said non-write conditions are met.

[0009] This invention recognises that there is a characteristic power consumption signature associated with a write to a data processing register and accordingly information concerning the data processing being performed in association with conditional write data processing operations can be externally observed, i.e. information upon whether or not the conditional write did or did not occur. The invention address this problem by providing a trash register to which a result value (which is preferably the true value) is written when the conditional write data processing operation meets its non-write conditions and a write would not otherwise occur. Accordingly, a write to a register whether the true register or the trash register, always occurs irrespective of whether or not the write conditions or non-write conditions are met and thus the security of this system is enhanced.

[0010] The data register to which the write is normally made when the write conditions are met is preferably part of a register bank containing a plurality of such registers. In this circumstance, a common trash register(s) may be used for dummy writes irrespective of how many real data registers are provided within the register bank.

[0011] Preferably, the trash register is physically located as part of the register bank so as to avoid leakage of information by observing which part of a circuit is active at any given time.

[0012] Whilst the present technique is applicable to a wide variety of systems, such a microprocessor based systems, digital signal processing system and the like, the invention is particularly well suited to systems including a processor core responsive to data processing instructions as these often incorporate conditional write instructions which give rise to the potential vulnerability addressed by this technique.

[0013] It will be appreciated that a conditional write operation may be arranged to either occur or not occur when particular conditions are met.

[0014] It will be appreciated that the normal technical prejudice in this field is to reduce power consumption as much as possible. Accordingly, it would conventionally be considered that not performing a register write when a conditional write operation did not require one would be an advantageous feature since it would reduce the amount of power consumed. The present technique moves against this technical prejudice in the field by deliberately performing a trash register write and consuming power even though this is not required for the real processing activities of the system.

[0015] In preferred embodiments of the invention the trash register activity can be selectively enabled and disabled depending upon a control signal stored in a system configuration register. This allows programmable activity of the trash register activity such that power can be saved by disabling this feature when non secure processing is taking place and yet security improved when required, such as when handling cryptographic keys, decoding passwords etc.

[0016] As mentioned above, whilst the trash register may be physically located within the register bank with the normal data registers, in preferred embodiments the trash register is unmapped to a register number such that it cannot be specified by any program instruction and accordingly is invisible as a register from the programmer's model point of view. The trash register is however visible in the sense that its activity can be enable or disabled in preferred embodiments by a configuration parameter.

[0017] Viewed from another aspect the present invention provides a method of processing data, said method comprising the steps of:

[0018] generating a result data value upon execution of a data processing operation, at least one data processing operation executed being a conditional write data processing operation, wherein

[0019] a result data value is not written to a data processing register when non-write conditions are met but is instead written to a trash register.

[0020] Embodiments of the invention will now be described, by way of example only, with reference to the accompanying drawings in which:

[0021] **FIG. 1** schematically illustrates a data processing system operable in a fixed timing mode and a variable timing mode;

[0022] **FIG. 2** schematically illustrates a conditional programming instruction;

[0023] **FIG. 3** is a flow diagram schematically illustrating part of the processing operations performed by an instruction decoder operating in accordance with the present techniques;

[0024] **FIG. 4** schematically illustrates the execution of a conditional branch instruction in a fixed timing mode;

[0025] **FIG. 5** is a diagram schematically illustrating a data processing system including multiple circuit portions which may be selectively enabled to perform required processing operations or dummy processing operations;

[0026] **FIG. 6** schematically illustrates a circuit portion and its associated dummy activity enabling circuit which may be responsive to both required enable signals and random dummy activity enable signals;

[0027] **FIG. 7** schematically illustrates a linear shift back feed register which may be used as a pseudo-random signal generator:

[0028] **FIG. 8** is a flow diagram schematically illustrating control of a circuit portion to perform required processing activity and dummy processing activity;

[0029] **FIG. 9** schematically illustrates a portion of a register bank including multiple data processing registers, multiple dummy registers, multiple shared dummy registers and a non-mapped trash register RT to which a dummy register write is made when a conditional write operation fails its condition codes;

[0030] **FIG. 10** is a flow diagram schematically illustrating a register write controlling circuit which seeks to balance the number of high to low and low to high transitions occurring when a register write occurs;

[0031] **FIG. 11** is a table illustrating the relationships between bit transitions for a particular bit within the data register and three further registers which are configured to balance the high to low and low to high transitions occurring in association with a register writet;

[0032] **FIG. 12** is a flow diagram schematically illustrating control of writing to a trash register on a condition code fail of a write operation;

[0033] **FIG. 13** is a diagram schematically illustrating a system having multiple execution mechanisms for an instruction and pseudo random selection of the execution mechanism employed for at least some instructions; and

[0034] **FIG. 14** is a flow diagram schematically illustrating control of the system of **FIG. 13**.

[0035] **FIG. 1** illustrates a data processing system **2** including a processor core **4**, a coprocessor **6** and a memory **8**.

[0036] In operation, the processor core **4** fetches instructions and data from the memory **8**. The instructions are fed to an instruction pipeline **10** where they occupy successive pipeline stages such as, for example, fetch, decode, execute, memory and write back on successive processing cycles. Pipelined processors are in themselves well known as a way of effectively executing a number of program instructions in a partially overlapping fashion in order to improve processor performance.

[0037] The data values read from the memory **8** by the processor core **4** are supplied to a register bank **12** from where they may be manipulated under program instruction control using one or more of a multiplier **14**, a shifter **16** and an adder **18**. Other data manipulating circuits may be provided, such as circuits performing logical operations, such as ANDs, Ors, count leading zeros etc.

[0038] **FIG. 1** also illustrates an instruction decoder **20** within the processor core **4** which is responsive to a program instruction within the instruction pipeline **10** to generate execution control signals that are applied to the various processing elements, such as the register bank **12**, the multiplier **14**, the shifter **16** and the adder **18** in order to control the data processing operations performed. As an example, the control signals generated by the decoder **20** may cause the appropriate operands to be read from the register bank **12** and supplied and acted upon by the appropriate ones of the multiplier **14**, the shifter **16** and the adder **18** so as to generate a result which is written back into the register bank **12**.

[0039] The coprocessor **6** is a system configuration coprocessor containing a number of configuration registers **22** which may be written under program control to set up configuration controlling parameters. These configuration controlling parameters can specify many aspects of the configuration of the processing system **2**, such as for example the endianess and the like. Included within one of these configuration controlling registers **22** is a bit which specifies whether or not the processor core should operate in a fixed timing mode or a variable timing mode. This bit is illustrated as being supplied as an input to the instruction decoder **20**, but it will be appreciated that this bit may also be supplied to various other points within the processor core **4** as required to control their behaviour. In dependence upon this fixed/variable bit, the processor core **4** operates in either a fixed timing mode or a variable timing mode. When in the fixed timing mode at least one program instruction which has a variable timing (i.e. takes a variable number of processing cycles to complete) in the variable timing mode, is instead forced to have a fixed timing (e.g. take the maximum possible number of processing cycles to complete irrespective of whether or not it could have been suppressed in its entirety or completed in less than the maximum number of processing cycles. As the instruction decoder **20** is primarily responsible for decoding the program instructions and instructing the activity of the other elements of the processor core **4**, the instruction decoder **20** can take the major role in controlling the processor core **4** to either operate in the fixed timing mode or the variable timing mode. Not all variable timing instruction need be provided with a fixed timing mode type of operation.

[0040] It will be appreciated that in the above description a single bit in the configuration controlling register **22** is shown as switching between fixed and variable timing modes. Alternatively, multiple bits within the configuration controlling register **22** may be provided to separately enable and disable the fixed or variable timing behaviour of different types of instruction, such as conditional instruction behaviour, uniform branch behaviour, disabling early terminate, etc.

[0041] **FIG. 2** schematically illustrates a conditional instruction **24**. This conditional instruction may be part of an instruction set which includes only some conditional instructions or part of an instruction set, such as the ARM instruction set, which is substantially fully conditional. The condition codes **26** encode a set of processor state conditions in which the associated instruction either will or will not be executed. As an example, the condition codes **26** can be arranged to specify that the instruction **24** will not execute if the condition codes currently set in the system indicate a zero result, a carry has occurred, an overflow has occurred or the like. This type of instruction can be utilised to provide efficient program coding. The fixed/variable bit at least partially suppresses the conditional behaviour in that the instruction will execute irrespective of its condition codes, but may not write its result in a way that has an effect upon the processor state.

[0042] **FIG. 3** is a flow diagram schematically illustrating part of the processing operations performed by the instruction decoder **20**. It will be appreciated that **FIG. 3** illustrates these processing operations as a logical sequence, whereas in practice these processing operations may be performed at least partially in parallel or in a different order.

[0043] At step **28**, the instruction decoder **20** waits for a new instruction to execute. When a new instruction is received processing proceeds to step **30** at which the condition codes associated with the new instruction are read. At step **32** these condition codes are compared with the currently existing condition codes in the system. These condition codes currently existing in the system are the result of previous processing activity, either in the immediately preceding instruction or in the last instruction which would have updated those condition codes.

[0044] At step **34**, a check is made for a match between the condition codes **26** of the current instruction being executed and the existing condition codes. If a match does not occur, then processing proceeds to step **36** where execution of the current instruction is started. It will be appreciated that **FIG. 3** illustrates a system in which execution occurs when a match does not occur, but alternative embodiments could equally well be ones in which execution occurs when a match occurs.

[0045] Following step **36**, processing proceeds to step **38** where a check is made as to whether or not early termination of the instruction is possible. This early termination may, for example, be because one of the operands has a particular value, such as zero or unity, or on subsequent processing cycles that a particular partial result has been produced. If early termination is possible, then processing proceeds to step **40** where a check is made as to whether or not the processor core **4** is currently operating in the fixed or variable timing mode. If the processor is in the variable timing mode, then processing proceeds to step **42** and the instruction concerned is early terminated with the result being returned as appropriate and processing returns to step **28**.

[0046] If the determination at step **40** is that the system is in the fixed timing mode, then processing proceeds to step **44** irrespective of the fact that early termination is possible. Step **44**, which may also be reached by a determination at step **38** that early termination is not possible, executes the instruction concerned for one processing cycle. In the case of a multicycle processing instruction, such as a multiplication, a divide, an add or a subtraction, these typically take several cycles to execute and so after step **44** processing proceeds to step **46** at which a determination is made as to whether or not the maximum number of cycles associated with that instruction has yet been performed. If the maximum number of cycles has been performed, then the result will have been generated. If early termination was possible and the system was being forced to continue to execute for further processing cycles, then step **46** will still indicate that this forced execution should cease when the maximum possible number of processing cycles for that type of instruction has been reached. If the maximum number of processing cycles has not yet been performed, then processing is returned to step **38**.

[0047] If the match tested for at step **34** was positive, then processing proceeds to step **48**. In this example, the positive detection of a match at step **34** indicates that execution of the particular instruction should be suppressed. Step **48** determines whether or not the system is currently in the forced execution mode. If in the forced execution mode, then processing proceeds to step **50** where a forced dummy execution of the instruction will occur. When dummy execution is performed the result is written to a trash register (see trash register **51** in **FIG. 1**), rather than the destination specified in the instruction itself so as to prevent the state of the system being modified by a program instruction which should not have executed as it should have been suppressed whilst also keeping a substantially unaltered power consumption. If at step **48** the determination is that the system is not in the forced execution mode but is in the variable timing mode, then processing bypasses step **50** and returns to step **28** with the program instruction being suppressed in the normal way.

[0048] It will be appreciated that **FIG. 3** illustrates a generic system in which dummy execution is applied to all condition code failed instructions and all early termination of instructions is suppressed. In practice, it is also possible for these techniques to be applied to a subset of conditional instructions and instructions capable of early termination. The multiple configuration controlling bits mentioned above could be used to selectively turn on features such as early terminate suppression, but not others, such as dummy execution following a condition code fail.

[0049] **FIG. 4** schematically illustrates the execution of a conditional branch instruction in the fixed timing mode. A sequence of instructions AB are executed until a conditional branch instruction BEQ (branch upon equal) is reached. This instruction encodes the behaviour that the specified branch will be performed if the flag indicating an equal result from previous processing is set and will be suppressed if this flag is not set. When the condition codes are passed, i.e. a condition code match, then the branch is taken and processing proceeds to instructions X, Y, etc. If the condition codes fail, then instead of being suppressed in its entirety, the BEQ instruction performs a branch to the immediately following instruction C. This is the same instruction which would have been reached if the BEQ instruction had been suppressed and not executed at all. However, in the fixed timing mode, the BEQ will have executed consuming the same number of processing cycles irrespective of whether or not the condition codes were passed or failed. This helps obscure the results of data processing operations previously performed from a person trying to gain access to secure data.

[0050] **FIG. 5** schematically illustrates a data processing system **52** in the form of a programmable processor core which responds to program instructions I and manipulates data D. The data processing system **52** includes a register bank **54**, a multiplier **56**, a shifter **58**, an adder **60**, and arithmetic logic unit **62**, a load store unit **64**, a data pipeline **66**, and instruction decoder **68** and a random clock gating circuit **70**. A system configuration coprocessor CP15 **72** is coupled to the processor core. The system configuration coprocessor **72** includes a system configuration register **74** holding multiple flag values which respectively serve to enable or disable the pseudo random dummy activity of different circuit portions of the data processing system **52**. It will be appreciated that the data processing system **52** will typically include many more circuit elements, but these have been omitted for the sake of clarity from **FIG. 5**.

[0051] Associated with the multiplier **56** is a dummy activity enable circuit **76** which serves to enable dummy activity in the multiplier **56** as appropriate or alternatively pass the required activity enable signals to activate the multiplier **56** when the program instruction being executed so requires. Similar dummy activity enables circuits **78**, **80**, **82**, **84** are associated with the respective other circuit portions **58**, **60**, **62**, **64** previously mentioned.

[0052] In operation, instructions to be executed are to be passed to the instruction pipeline **66** and on to the instruction decoder **68** to generate instruction driven enable signals which are applied to respective circuit portions. These enable signal serve to select the data path through the data processing system **52** and to activate the circuit portions concerned to read their inputs, to perform the specified processing and to generate their associated output signals. As an example, a multiplier-accumulate operation might read data values from the register bank **54**, apply these to the multiplier **56** and adder **60** and then write back the result to the register bank **54**. Thus, the register bank **54**, the multiplier **56** and the adder **60** would all be subject to required activity enable signals which both enabled their operation and selected them to form a complete data path. The different circuit portions have different power consumption characteristics and timing characteristics such that external observation could reveal which instruction were being executed by observing such parameters. Accordingly, pseudo random dummy activity of the other circuit portions not required for the instruction being executed is also enabled. Thus, even though the shifter **58** may not being used by the particular multiplier accumulate instruction being executed, it may nevertheless be pseudo randomly enabled such that it will consume power by shifting whatever value is applied to its input. Its output latches will not be enabled to avoid this dummy activity altering the circuit state an undesired way which may interfere with required operation e.g some circuit portions might assume persistence of output values. The dummy activity is enabled for time periods matching the normal operation timings for the circuit portions concerned.

[0053] The random clock gating circuit **70** serves to receive a plurality of pseudo random enable signals for different respective circuit portions and gate these for applying to respective circuit portions under the control of configuration parameters read from the system configuration register **74** within the system configuration coprocessor **72**. These configuration flags may indicate that dummy activity should be enabled for the shifter **58**, ALU **62** and multiplier **56**, but not for the adder **60** or the load store unit **64**. The different pseudo random enable signals allow different pseudo random characteristics to be applied in a manner that can match these respective circuit portion concerned. As an example, there may a different minimum enable time associated with the normal timing of the different circuit portions.

[0054] At an overall level, it will be seen that the instruction decoder **68** will serve as a required activity enabling circuit which will enable the circuit portions required to perform the data processing operation specified by the instruction currently being executed. superimposed upon this required activity, various dummy activities within the other circuit portions will be enabled/stimulated by the dummy activity control circuitry provided in various places in the rest of the data processing system **52**. The dummy activity serves to mask the power consumption and timing characteristics associated with the required activity.

[0055] **FIG. 6** schematically illustrates a circuit portion **86** which may be subject to both a required enable signal en and a dummy enable signal rnd. This circuit portion **86** can be considered as a sequence of latches between which processing logic manipulates data values. When genuine required activity is necessary, all of the latches that provide a data path through the circuit portion **86** are enabled and the required processing will be performed between the input latches and the output latches. When dummy activity is instructed, then only the input latches and the intermediate latches are enabled. Thus, a data path is not provided through the full circuit portion and the output values generated by that circuit portion are not altered.

[0056] **FIG. 7** illustrates a linear feedback shift register of the type which may be used to generate pseudo random clock signal. These clock signals can be provided to the random clock dating circuit **72** of **FIG. 5**. Separate pseudo random signal generators may be provided for the different circuit portions. The fixed clock frequency associated with the different pseudo random generators may be altered so as to match the characteristics of the circuit portion concerned and further obscure the masking operation as required.

[0057] **FIG. 8** schematically illustrates the control of an enable signal for a circuit portion. At step **88** a determination is made as to whether or not an enable signal en has been received from the instruction decoder **68**. If such an enable signal has been received, then processing passes to step **90**. An enable signal from the instruction decoder **68** indicates that required processing operation is necessary in accordance with a genuine program instruction being decoded. Thus, step **90** enables the input, output and clock signal to the circuit portion concerned. If at step **88** no enable signal en is received from the instruction decoder, then processing passes to step **92** where a determination is made as to whether or not dummy operation of that circuit portion is permitted. If dummy operation is permitted, then processing proceeds to step **94** at which the inputs and clock to the circuit portion are enabled, but the outputs from the circuit portion are not enabled. The circuit portion then undertakes dummy activity. If the determination at step **92** was that dummy operation was not permitted, as indicated by the system configuration parameter(s), then processing terminates by passing to step **94**.

[**0058**] It will be appreciated that the process illustrated in **FIG. 8** is in the form of a sequential flow diagram. In practice, this control may be performed in a different sequence and use circuit elements spread throughout the data processing system **52**. The operations illustrated as being sequentially performed may in fact be performed in parallel or the control functions modified. At an overall level an individual circuit portion will be enabled to perform its normal required operation in response to an appropriate program instruction and will be enabled to perform dummy activity when permitted by the associated configuration parameter.

[**0059**] **FIG. 9** schematically illustrates a register bank **96**. This register bank is based upon the ARM processor programmer's model for user mode operation in accordance with processors designed by ARM Limited, Cambridge, England). In practice, further registers may be provided for other processor modes, but these have been omitted for the sake of clarity. The normal data registers R**0** to R**15** are provided for holding data values. The registers R**13**, R**14** and R**15** typically serve to store the program counter value, the branch return address value and the stack pointer, which tend to be none security related data values. Accordingly, transition balancing upon data writes is not necessary for R**13**, R**14** and R**15**. A trash register RT is provided within the register bank **96** for use in association with conditional writes which fail their condition codes thus, a conditional write instruction which fails its condition code would not normally make any write. However, with this system such a failed conditional write instruction nevertheless writes a data value to the trash data register RT even though the condition codes have failed. This masks any difference in power consumption or timing that might be associated with condition code failure or condition code passing of a conditional write operation. The trash data register RT does not appear in the programmer's model in a way that enables it to be addressed with a register specifying operand within an instruction.

[**0060**] As well as the trash data register RT, further registers **98**, **100** are also provided for the purpose of balancing the high to low and low to high transitions. Dedicated dummy registers **98** are provided in respect of the data registers R**0** to R**12** as well as the trash data register RT. Shared dummy registers **100** are provided for storing an exclusive OR value as well as the inverse of the exclusive OR value in response to each write to a data register subject to the transition balancing technique. A register write control circuit **102** serves to generate the appropriate data values to be written to the further registers **98**, **100** in response to a data value write to a data register. This symmetric write control is selectively enabled and disabled by an appropriate system configuring controlling flag signal from the system configuring coprocessor **72**.

[**0061**] **FIG. 10** is a flow diagram schematically illustrating the operation of the register write control circuit **102**. At step **104** the circuit waits for a register write operation to be instructed. Step **106** determines whether this register write is to one of the data registers or the trash data registers RT for which the symmetric write control system is applied. If the register write is not to such a register, then processing proceeds to step **108** and a simple write of the required data value X is made to one of the registers R**13**, R**14** and R**15**.

[**0062**] If the register to which the write is being made is potentially subject to symmetric register writing then step **110** serves to determine whether or not this feature is currently enabled. If this feature is not currently enabled, then processing proceeds to step **108**. If this feature is enabled, then processing proceeds to step **112**.

[**0063**] At step **112**, the register controls circuit calculates for each bit position within the data value a value being the inverse exclusive OR of the current bit being written at that position and the previously stored bit at that position which is then exclusive Ored with the previously stored dummy register value for that bit position (see **FIG. 11**). The register control circuit **102** also calculates the inverse of the determination as well the inverse of the bit being written as the data value to the data register. These values are calculated for all of the bits being written (e.g. 3 dummy 32-bit values).

[**0064**] At step **114** the data value is written to the data register in a similar manner to step **108**. At step **116** the three further values determined for each bit position within the registers concerned are written to the three further registers. Steps **114** and **116** take place simultaneously. As will be discussed in relation to **FIG. 11**, this results in a balance number of high to low and low to high transitions and thus power consumed.

[**0065**] **FIG. 11** illustrates a table of possible bit values before and after a data write operation. The data value is being written to a register Rn, which is a register to which the symmetric write operation function is applied. The values at time t and time t+1 are illustrated. The inverse of these values is simply determined. Each of the data registers subject to this symmetric operation is provided with a dedicated dummy register **98** which stores the inverse of the data value currently held in the data register.

[**0066**] The shared dummy registers **100** are indicated in **FIG. 11** as registers Rd. For each bit position on within the shared dummy register Rd the new value to be written at that bit position when a data write occurs is determined by the function shown at the bottom of **FIG. 11**. This function ensures that when a change does not occur in the data value and the inverse of the data value, then a change is guaranteed to occur in the corresponding bit within the shared dummy register and accordingly its inverse. The table shows the changes in the shared dummy register values which occur when the data value does not change and the shared values in the dummy register values not changing when the data value does change. Thus, there is a guaranteed fixed number of transitions for every write, i.e. a balanced equal number of transitions high to low and low to high.

[**0067**] **FIG. 13** is a flow diagram illustrating the action of the dummy data register RD to provide writes when a write operation fails its condition code(s). At step **118** the control logic waits for an instruction to be received. This control logic may be the instruction decoder **68** or other logic. Step **120** determines whether or not the instruction failed its condition codes. If the instruction does not fail its condition code, then it is normally executed at step **122** and makes its write to the register specified by the register operand within that instruction. If the instruction does fail its condition codes, then processing proceeds to step **124** at which a determination is made as to whether or not dummy data register writes are enabled. If these are not enabled, then processing terminates. If dummy data register writes are enabled, then processing proceeds to step **126** at which a write of the data value calculated by the condition code failed instruction is written to the trash data register RT even though the condition codes failed. This balances the power consumption and timing irrespective of a condition code pass or a condition code fail. It will be appreciated that the trash data register RT is also subject to the transition balancing mechanisms previously discussed.

[0068] **FIG. 13** illustrates a data processing system **128** in which multiple instruction execution mechanisms are provided for at least some instructions. The data processing system **128** is one which supports the native execution of at least o some Java bytecode instructions. This type of data processing system and native execution is described in published PCT Patent Application Number WO-A-02/29555. The disclosure of this published application as a whole and in respect of the native hardware execution and selective software emulation of more complex Java bytecodes in particular is incorporated here in by reference.

[0069] The Java bytecode decoder **130** may be selectively enabled and disabled by an input signal. When the Java bytecode decoder **130** is disabled a received Java bytecode will trigger an exception which starts execution of software emulation code for handling Java bytecodes using the native ARM Thumb instruction sets. This support code is stored within memory in area **132** as illustrated. The Java bytecode program **134** is also stored within memory. When it is desired to obscure the nature of the Java program execution, the Java bytecode decoder **130** may be subject to a pseudo random signal which selectively enables and disables this element so as to effectively switch the instruction execution mechanism for the Java bytecodes between a mixed hardware and emulation execution mechanism and a purely emulation mechanism. Configuration controlling values within a system configuration register **136** specify whether or not the Java decoder **130** is present and whether or not random enabling and disabling of this Java decode **130** is permitted.

[0070] **FIG. 14** schematically illustrates the handling of a received Java bytecode. At step **138** a Java bytecode is received. Step **140** determines whether or not the Java decoder **130** is enabled. The pseudo random enabling and disabling of the Java decoder **130** effectively causes a branch to either step **142** at which the bytecode is always emulated or an attempt to execute the instruction in hardware at step **146**. This obscures/masks the power signature associated with Java bytecode execution. If the determination at step **146** is that the particular Java bytecode concerned is not supported by the Java decoder **130**, then this Java bytecode will also be emulated in software at step **142**. However, if the Java bytecode is supported in hardware, then it is executed in hardware at step **146**.

1. Apparatus for processing data, said-apparatus comprising:

a process core (**4**) operable to execute data processing instructions to generate result data values; and

data processing registers (**12**) holding data values defining state of said processor core to which said result data values are written; wherein

at least one data processing instruction executed by said processor core is a conditional write data processing instruction encoding condition codes (**26**) specifying conditions under which said conditional write data processing instruction will or will not be permitted to write data to effect a change in state of said processor core: and further comprising

a trash register (**51**) to which a result data value may be written instead of a data processing register upon execution of said conditional write data processing instruction when said condition codes within said conditional write data processing instruction do not permit a write to effect a change in state of said processor core.

2. Apparatus as claim in claim 1, comprising a register bank (**12**) having a plurality of data registers to which result data values are written.

3. Apparatus as claimed in claim 1, wherein writing to said trash register (**51**) is programmably disabled by a trash register control signal.

4. Apparatus as claimed in claim 3, wherein said trash register control signal is stored in a system configuration register.

5. Apparatus as claimed in claim 2, wherein said trash register (**51**) is part of said register bank, said trash register being unmapped to a register number such that said trash register may not be specified by a register specifying operand value.

6. A method of processing data, said method comprising the steps of:

generating result data values upon execution by a processor core (**4**) of data processing instructions, at least one data processing instruction executed being a conditional write data processing instruction encoding condition codes (**26**) specifying conditions under which said conditional write data processing instruction will or will not be permitted to write data to effect a change in state of said processor core and wherein

a result data value is not written to a data processing register holding a data value defining state of said processor core when condition codes within said condition write data processing instruction do not permit a write to effect a change in state of said processor core but is instead written to a trash register (**51**).

7. A method as claimed in claim 6, wherein said data processing register is part of a register bank (**12**) having a plurality of data registers to which result data values are written.

8. A method as claimed in claim 6, wherein writing to said trash register (**51**) is programmable disabled by a trash register control signal.

9. A method as claimed in claim 8, wherein said trash register control signal is stored in a system configuration register.

10. A method as claimed in claim 7, wherein said dummy register is part of said register bank, said trash register being unmapped to a register number such that said trash register may not be specified by a register specifying operand value.

* * * * *