IPOS copy



# SIMD CORRECTION CIRCUIT FOR ARITHMETIC/SHIFT OPERATIONS

## BACKGROUND OF THE INVENTION

### 1. Field of the Invention

This invention relates generally to data registers in microprocessor circuitry and more particularly to a Single Instruction Multiple Data (SIMD) correction circuit for modifying the results of an arithmetic/shift operation.

### 2. Description of Related Art

Heretofore, logic circuits have been proposed to improve performance of arithmetic/shift operations in data processing. With the increasing need for processing large amounts of data at ever increasing speed, improved efficiency of arithmetic/shift operations is very important. In particular, one of the difficulties of Multi-media, especially relating to graphics, is the large number of data that must be processed. An attribute of the Single Instruction Multiple Data (SIMD) is that each SIMD instruction can perform an operation on each 8 bit, 16 bit, 32 bit, or 64 bit field of a 64 bit operand independently.

A SIMD ADD, for example, would perform an add on the first, second, third and fourth 16 bit section of the register operands as if the SIMD ADD were 4 independent 16 bit add instructions. A SIMD SHIFT, for example, would perform a shift on the first, second, third, and fourth 16 bit section of the register operands as if the SIMD SHIFT were 4 independent 16 bit

SHIFT instructions. Also, the SHIFT operations include shift left, shift right logical, shift right arithmetic.

SIMD has gained recent popularity with the announcement of the Intel MMX Extension. The MMX is a SIMD architecture. Implementing MMX extensions to the X86 architecture costs additional execution units dedicated to the MMX format. Converting a standard execution unit to perform both standard and SIMD operations introduces difficulties that have not heretofor been addressed. First, additional execution units adds delay to critical paths such as carry propagate paths since in SIMD the carry between SIMD sub-operands (16 bit or 32 bit sections) must be suppressed. Second, additional execution units requires additional silicon real estate (area). Third, additional execution units increases the development time and cost because the execution units are highly specialized circuits.

## SUMMARY OF THE INVENTION

This invention includes a correction circuit to convert a standard logic execution unit to perform both standard operations and SIMD operations. This invention has the advantages of alleviating the difficulty of delay to critical paths such as carry propagate paths; requiring less silicon space than if additional execution units were added; and does not require developing further highly specialized execution units. Execution units are highly specialized and thus to make a change to one is very labor intensive. This invention improves the performance of processing large amounts of data in applications such as Multi-media and signal processing.

This invention eliminates the critical paths, because logic is not added in the critical paths; saves time and silicon real estate because the same execution unit is being reused; and does not require the restructuring of a complex unit with added logic therein. Another operation that requires processing large amounts of data which has application in both Multi-media and signal processing is matrix multiply. The present invention may be used on the standard logic of arithmetic/shift operations (e.g., ADD, SUBTRACT, DIVIDE, MULTIPLY, SHIFT) of Arithmetic Logic Units (ALUs) and Shift logic.

This invention includes a microprocessor circuit having an execution unit for execution of standard instructions in an arithmetic/shift operation and a correction circuit responsive to the execution unit for modifying the standard instructions provided by the execution unit to results required by a SIMD instruction being executed. This improves efficiency of the operations because the correction circuit modification may be performed in a second cycle and the arithmetic/shift operation is free to execute a second instruction in the second cycle. The arithmetic/shift operation results from an instruction provided by either an Arithmetic Logic Unit (ALU) or by a shift function. The correction circuit passes data, unchanged for standard logical instructions but provides condition codes according to the SIMD instruction. The correction circuit corrects arithmetic operations by operating on standard data based on significant bits and carry bits for sub-unit boundaries.

In the case of a Shift operation, this invention includes a Shifter performing standard

operations on instructions in a first cycle of operation, a mask generating circuit in parallel with the Shifter circuit, and a correction circuit responsive to the Shifter and the mask generating circuit for modifying the standard results provided by the Shifter to results required by a SIMD Shift operation being executed. This modification may be performed by an address overlay mask in a second cycle operation and the Shifter is free to execute a second instruction in the second cycle.

## BRIEF DESCRIPTION OF THE DRAWINGS

The preferred exemplary embodiment of the present invention will hereinafter be described in conjunction with the appended drawings, where like designations denote like elements, and:

FIG. 1 depicts a block diagram showing an execution unit for performing an arithmetic/shift operation and correction circuit of a first preferred embodiment in accordance with the present invention.

FIG. 2 depicts a block diagram showing an execution unit for performing an arithmetic/shift operation unit and correction circuit of a second preferred embodiment in accordance with the present invention.

FIG. 3 depicts a high level view of a SIMD operation in accordance with a preferred embodiment of the present invention.

FIG. 4 depicts a flow diagram disclosing a correction unit detail in accordance with a preferred embodiment of the present invention.

FIG. 5 depicts a flow diagram of a shift function in accordance with a preferred embodiment of the present invention.

FIG. 6 depicts mask generator detail in accordance with the present invention as depicted in Fig. 5.

FIG. 7 depicts AND/OR mask (AOM) detail in accordance with the present invention as depicted in Fig. 5.

FIG. 8 depicts a block diagram showing an execution unit/correction unit for a third preferred embodiment in accordance with the present invention.

## BEST MODE FOR CARRYING OUT THE INVENTION

Referring to Figure 1, a microprocessor circuit 10 of the present invention is shown with a correction circuit 20. The correction circuit 20 modifies the results of either standard operations of an ALU (Figure 4) or of standard operations of a Shifter (Figures 5-7) to results required by a SIMD instruction being executed. Standard execution unit 16 receives an instruction to be executed from a reservation station (s1) 15 which holds the instruction 14 to be executed. The execution unit 16 provides access to registers 12 and performs either an ALU operation or a shifter operation. Finish stage (s2) 18 holds results to be written when the instruction 14 completes in the write-back stage.

A correction circuit 20 is shown including correction unit 22 and finish stage (s3) 24 for SIMD applications. In a first embodiment in Figure 1, the execution unit is depicted as a two-stage pipeline. The first stage 17 to the pipeline enters the correction circuit 20 and the second stage 19 to the pipeline bypasses the correction circuit 20 to MUX 26 for non-SIMD operations. The results to registers 28 of the two pipelines 17, 19 are provided to the registers 12.

Figure 2 depicts a correction circuit 30 of a second embodiment of the present invention. This embodiment is similar to Figure 1 except that the MUX 26 is eliminated, permitting a SIMD or correction operation in the same cycle as a non-SIMD operation. A SIMD correction instruction is passed through flow line 29 and a non-SIMD instruction bypasses the correction unit 22 through flow line 31.

In yet another implementation Figure 8 shows a combined execution unit/correction unit 100. The correction stage could be performed by the ALU or shift operator of the execution unit/correction unit 100 by feeding the result 17, 101 of the first pass of the ALU or shift operator stage back to the ALU or shift operator with appropriate control hardware added. All of these implementations become obvious to one skilled in the art when taught the present invention and are therefore claimed by this disclosure.

SIMD Implementation Using Standard ALU

Figure 3 shows a high level view of a SIMD operation 40. An instruction 14 is depicted. A single SIMD instruction 44 performs operations simultaneously on subsets 7, 6, 5, 4, 3, 2, 1, 0 of the registers 12. As shown, Register operand (R2) 44 points to a 64 bit register 12. The

operand instruction OP specifies the operation, for example, R1<-R1 + R2 ADD instruction. The instruction 14 will perform independent adds on each subset 7 through 0. Each subset 7 through 0 could be a single byte such that 8 independent adds are performed by the instruction using each of the 8 bytes in the 64 bit register operand, or the subset could be 16 or 32 bits.

Refering to Figure 4, the detail of the correction unit 20 of Figures 1 and 2 are depicted for use with a standard ALU. A 64 bit execution unit 16 has an eight byte result register 24. B7 is the high order byte, B(6),... with B(0) being the low order byte. Each of these bytes has three additional latches C(n), CI(n), and ZB(n). C(n) is the carry out of the high order bit of B(n) (the carry out of the byte), CI(n) is the carry into the high order bit of B(n), and ZB(n) indicates that all bits of B(n) are zero.

Each of these B(n) registers is connected to a box labeled F(n), the force box. F(n) has two input control lines plus the data from B(n). The input control lines are Force 8(n) and Force 0(n). Force 8(n) forces hex '80' on the output FB(n) of this box while Force 0(n) forces hex '00'. If both control lines are off then the input bus is passed to the output bus unmodified. The FB(n) byte output goes to a increment/decrement box ID(n). This box has two control inputs IB(n) and DB(n). IB(n) increments the input bus by one and DB(n) decrements the input bus by one. Both control lines being off passes the input bus to the output bus R(n) unmodified. ID(n) thus has a byte output bus and a output control line CB(n). This output is the CARRY/BORROW NOT line for the byte increment/decrement. That is, it should be active when DB(n) is active AND FB(n)=X'00', OR when IB(n) is active and FB(n)=X'FF'.

The following explain examples of the operations to be performed. While all

combinations of these functions are not possible in the current SIMD definition the present invention contemplates all possibilities.

A number of instruction parameters are provided during the correction stage. First, Byte(B), Word(W), Double word(DW), and Quadword(QW) data size (where a word is 16 bits) is provided. Second, an ADD or SUBTRACT instruction is provided. Third the instruction may be in signed or unsigned format. Signed numbers are standard two's complement format, while unsigned assumes only positive numbers. Fourth, these instructions are performed with either saturation or without. Without saturation means that results wrap if they exceed the specified size. With saturation means that if the results exceed the size then the largest or smallest number possible given the format is inserted. For example, in the case of an unsigned byte an overflow produces a result of 255 or hex 'FF' while an underflow (for subtract only) produces a result of 0 or hex '00'. For signed byte an overflow produces a result of 127 or X'7F' while an under flow (possible with add and subtract) produces a result of -128 or hex '80'. The structure defined in Figures 1-4 allow for implementation of correction for all combinations of the above instruction parameter scenarios. Note that while the F-box produces the X'00' and X'80' directly the constants X'FF' and X'7F' are produced indirectly taking the X'00' and X'80' and decrementing by one.

In the simplest scenario ADD UNSIGNED BYTE (no saturation). The low order byte B(0) needs no correction. The next byte B(1) needs to be decremented by one if C(0)=1. Thus DB(1)=C(0). In general B(n) needs to be decremented by one if C(n-1)=1 or DB(n)=C(n-1).

For the ADD UNSIGNED WORD (no saturation) scenario, the result B(1)/B(0) needs no correction. The next word B(3)/B(2) needs to be decremented by one if C(1)=1. This means that DB(2)=C(1) and that DB(3)=CB(2). In general then DB(n)=C(n-1) and DB(n+1)=CB(n) where n=2, 4, 6.

ADD UNSIGNED DOUBLE WORD follows a similar pattern while ADD UNSIGNED QUADWORD needs no correction. It should also be obvious that the above pattern is identical for ADD SIGNED (no saturation).

In ADD UNSIGNED BYTE WITH SATURATION only overflow is possible, so only forcing X'FF' is required. The low order byte B(0) needs no decrementing but if C(0)=1 then X'FF' should be forced. This is done by setting F0(0)=1 and DB(0)=1. Thus F0(0)=C(0) and DB(0)=C(0). The next byte B(1) needs to be decremented if C(0)=1 and needs to be forced to X'FF' if C(1)=1. Thus F0(1)=C(1) and DB(1)=C(0) OR C(1). Note that it is acceptable to force X'FF' in the case where C(1)=1 was caused by C(0)=1 because the result ignoring C(0) must have been X'FF' anyway. So in general F0(n)=C(n) and DB(n)=C(n) OR C(n-1).

With regard to ADD UNSIGNED WORD WITH SATURATION, the low order word B(1)/B(0) does not need to be decremented but if C(1)=1 then both bytes need to be forced to X'FF'. Thus DB(1)=DB(0)=FO(0)=FO(1)=C(1). The next word B(3)/B(2) needs to be decremented if C(1)=1 and forced to X'FFFF' if C(3)=1. For B(2) then DB(2)=C(3) OR C(1) and FO(2)=C(3). For B(3) then FO(3)=C(3) and DB(3)=C(3) OR CB(2). In general FO(n+1)=FO(n)=C(n+1) and DB(n)=C(n+1) OR C(n-1) and DB(n+1)=C(n+1) OR CB(n) where n=2, 4, 6.

ADD UNSIGNED DW/QW WITH SATURATION follows a similar pattern.

In unsigned addition C(n) represents overflow, while under flow could not happen. In signed addition underflow and overflow are possible so it is more complicated. Let OV(n)=C(n) NOT AND CI(n) represent overflow, while UV(n)=C(n) AND NOT CI(n) represent underflow and V(n)=UV(n) OR OV(n) represent some overflow/underflow condition.

ADD SIGNED BYTE WITH SATURATION requires forcing X'7F' or X'80'. The low order byte B(0) needs no decrementing but if OV(0)=1 then B(0)needs to be forced to X'7F' while if UV(0)=1 then B(0) needs to be forced to X'80'. Thus and X'7F' if OV(1)=1. Thus F8(1)=V(1) and DB(1)=OV(1) OR C(0). In general F8(n)=V(n) and DB(n)=OV(n) )R C(n-1).

ADD SIGNED WORD WITH SATURATION adds complication. When detecting underflow or overflow the word must be forced to X'8000' and X'7FFF' respectively. Thus the low order byte must be forced to X'00' and X'FF' while the high order byte must be forced to X'80' and X'7F'. The low order word B(1)/B(0) does not need to be decremented but if OV(1)=1 then B(1)/B(0) needs to be forced to X'7FFF' and if UV(1)=1 then B(1)/B(0) needs to be forced to X'8000'. Thus F8(1)=F0(0)=V(1), DB(0)=DB(1)=OV(1). The next word B3/B2 must be decremented if C(1)=1, forced to X'7FFF' if OV(3)=1, and forced to X'8000' if UV(3)=1. Thus F)(3)=F0(2)=V(3), DB(2)=OV(3) OR C(1) and DB(3) =OV(3) OR DB(2). In general F8(n=1)=F0(n)=V(n+1), DB(n)=)V(n=1) OR C(n), and DB(n+1)=OV(n+1) OR DB(n) for n=2, 4, 6.

ADD SIGNED DW/QW WITH SATURATION follows a similar pattern. The

SUBTRACT scenarios are the same as ADD except that instead of the correction being decremented by one when carry in is one, it is incremented by one when carry in is zero (borrow is one). Underflow and overflow are defined the same.

COMPARE FOR EQUAL and COMPARE FOR GREATER THAN take two operands in the signed B, W, DW, or QW length and perform a compare. The result field is set to all ones if true and all zeros if false. Given the ZB(n), C(n), and CI(n) signals it is trivial to determine if the Byte, Word, Double word, or Quadword is EQUAL or GREATER THAN. The structure defined already permits forcing all zeroes and all ones.

Referring to Figure 4, the carry logic can be generated in many ways known in the art. The figure implies a "ripple" arrangement but carry predict and carry look-ahead techniques (for instance) may be used within the scope of the invention.

SIMD Shift Instruction With Standard Shifter

As shown in Figure 1 a reservation station (s1) 15 which holds the instruction to be executed, an execution unit 16 includes a SHIFTER (FIG. 5), access to registers 12, and a finish stage (s2) 18 which holds results to be written when the instruction completes in the write-back stage. Figure 2 shows the present embodiment of the invention where the execution unit 16 is a two stage pipeline and SIMD correction is bypassed for non-SIMD instructions. This embodiment is more fully described above. The SIMD stage takes the result of the Shifter operation and corrects the sub-units to conform with the SIMD operation.

In another implementation of the invention the MUX 26 could be eliminated, permitting a

SIMD instruction to complete in the same cycle as a single cycle instruction following it.

In yet another implementation of the present invention the correction stage 20 could be performed by the Shifter stage by feeding the result of the first pass of the Shifter stage back to the Shifter with appropriate control hardware added. All of these implementations become obvious to one skilled in the the art when taught the present invention and are therefore encompassed by this disclosure.

Figure 3 shows the concept of the SIMD operation, a single SIMD instruction performs operations simultaneously on subsets of the register operands. Register (R2) 44 points to a 64 bit operand 12. The OP code specifies the operation (R1 < -R2 SHIFT instruction for instance). The SIMD instruction will perform independent SHIFTS on each subset. The subset could be a single byte such that 8 independent SHIFTS are performed by the instruction using each of the 8 bytes in the 64 bit register operand, or the subset could be 16 or 32 bits.

Figure 5 shows a high level view of a Shifter stage 60 of a preferred embodiment of the present invention. The Shift Count (SCNT) 62, Operand 64 and Result regs 82 are part of the standard Shift Unit function for Non-SIMD instructions. A Mask Generator 76 uses the Operand 64 and Shift Count 62 to generate a Mask Reg (MR) 80 for the AND-OR Mask (AOM) 84 in parallel with the standard shift result. The Mask generator, MR, and AOM are the correction circuit as depicted in Figs. 1 and 2. The result of the AOM 84 is latched in the final result (second stage) register.

The details of the Mask Generator 76 are shown in Figure 6. The Left or Right shift

indicator (L/R) 66 in conjunction with the Shift Count Register (SCNT) 62 creates a Shift Count Mask (SCNTM) 62. The Byte Shift Mask (BSM) 78 generates the shift mask for each byte of

**Table 1. Byte Shift Mask (BSM) Function**

| SCNTM | BSM (7:0) |
|---|---|
| 210 | 76543210 |
| 000 | 11111111 |
| 001 | 11111110 |
| 010 | 11111100 |
| 100 | 11110000 |
| 101 | 11100000 |
| 110 | 11000000 |
| 111 | 10000000 |

the 64 bit mask. This mask would be correct if it was a byte left shift with a count less than eight. ANDing the mask with the data produces the proper result for a shift left. Table 2 desrcibes the equations for deriving when the shift count is greater than or equal to a certain number. For example SCT8 is the equation for SCNT greater than or equal to 8. They are used in Table 3 to define the force X'00' (F0Sn) function for shift left and the force x'ff' (F1Sn) function for shift right.

**Table 2. Shift Count Equations**

SCT8 = S3+S4+S5+S6
SCT16 = S4+S5+S6
SCT24 = S5+S6+(S3 A S4)
SCT32 = S5+S6
SCT40 = S6+S5 A (S3 + S4)
SCT48 = S6+ (S5 A S4)
SCT56 = S5 A S4 A S3
SCT64 = S6

**Table 3. SHIFT LEFT AND SHIFT RIGHT**

| | | FOS7 | FOS6 | FOS5 | FOS4 | FOS3 | FOS2 | FOS1 | FOS0 |
|---|---|---|---|---|---|---|---|---|---|
| | RIGHT | F1S7 | F1S5 | F1S5 | F1S4 | F1S3 | F1S2 | F1S1 | F1S1 |
| BYTE | LEFT | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | RIGHT | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| WORD | LEFT | SCT16 | SCT8 | SCT16 | SCT8 | SCT16 | SCT8 | SCT16 | SCT8 |
| | RIGHT | SCT8 | SCT16 | SCT8 | SCT16 | SCT8 | SCT16 | SCT8 | SCT16 |
| DWORD | LEFT | SCT32 | SCT24 | SCT16 | SCT8 | SCT32 | SCT24 | SCT16 | SCT8 |
| | RIGHT | SCT8 | SCT16 | SCT24 | SCT32 | SCT8 | SCT16 | SCT24 | SCT32 |
| QWORD | LEFT | SCT64 | SCT56 | SCT48 | SCT40 | SCT32 | SCT24 | SCT16 | SCT8 |
| | RIGHT | SCT8 | SCT16 | SCT24 | SCT32 | SCT40 | SCT48 | SCT56 | SCT64 |

At this point when the mask in the MR is AND/OR red with the 64 bit shifted data in the result register according to the function code defined in figure 7 the proper result is generated. The mask generated for shift left has ones where the data should be preserved and zeroes where it should be zeroed out. For shift right zeroes indicate the data should be preserved and ones indicate that zeroes or ones should be padded depending upon the kind of shift (arithmetic or

logical) and the appropriate high order bit.

Table 4 shows the Generation of the function field which goes to the AOM. Referring to Figure 7, the AOM receives the MR and Result register for each byte and performs the function indicated in Table 5 based on the FN(X) field.

**Table 4. FN(X) GENERATION[1]**

| Operation: | FN7 | FN6 | FN5 | FN4 | FN3 | FN2 | FN1 | FN0 |
|---|---|---|---|---|---|---|---|---|
| SL (B,W,DW,QW) | 01 | 01 | 01 | 01 | 01 | 01 | 01 | 01 |
| SRL (B,W,DW,QW) | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 |
| SRA(B)* | R63 | R55 | R47 | R39 | R31 | R23 | R15 | R7 |
| SRA(W)* | R63 | R63 | R47 | R47 | R31 | R31 | R15 | R15 |
| SRA(DW)* | R63 | R63 | R63 | R63 | R31 | R31 | R31 | R31 |
| SRA(QW)* | R63 | R63 | R63 | R63 | R63 | R63 | R63 | R63 |

[1] SL=SHIFT LEFT
SRL=SHIFT RIGHT LOGICAL
SRA=SHIFT RIGHT ARITHMETIC
B-BYTE
W=2 BYTES
DW=4 BYTES
QW=8 BYTES

*For SRA, the two bit FN code is a "1" concatenated with R(y) where y is a bit position (i.e., R15 is bit 15).

**Table 5. AOM FUNCTIONS**

| FN (1:0) | FUNCTION | DESCRIPTION |
|---|---|---|
| 0 1 | R AND MR | SHIFT LEFT |
| 1 0 | R AND $MR_{not}$ | SHIFT RIGHT W/ ZEROES |
| 1 1 | R OR MR | SHIFT RIGHT W/ ONES |

This invention has been described in terms of specific embodiments, set forth in detail. It should be understood, however, that these embodiments are presented by way of illustration only, and that the invention is not necessarily limited thereto. Modifications and variations within the spirit and scope of the claims that follow will be readily apparent from this disclosure, as those skilled in the art will appreciate.

* C L M S *



THIS SHEET HAS BEEN REPLACED

What is claimed is:

1. A microprocessor circuit comprising:

a). arithmetic/shift function performing standard operations on instructions in a first cycle of operation; and

b). a correction circuit responsive to said arithmetic/shift function for modifying the standard results provided by said arithmetic/shift function to results required by a Single Instruction Multiple Data (SIMD) instruction being executed.

2. The circuit of claim 1, wherein the correction circuit modification is performed in a second cycle.

3. The circuit of claim 2, wherein the arithmetic/shift function is free to execute a second instruction in the second cycle.

4. The circuit of claim 1, wherein the arithmetic/shift function and the correction circuit can each complete an instruction in the same cycle.

5. The circuit of claim 1, wherein the arithmetic/shift function is an instruction provided by an Arithmetic Logic Unit (ALU).

This Sheet Has Been Replaced

* C L M S *



THIS SHEET HAS BEEN REPLACED

6. The circuit of claim 1, wherein the correction circuit passes data, unchanged for logical instructions but provides condition codes according to the SIMD instruction.

7. The circuit of claim 5, wherein the correction circuit corrects arithmetic operations by operating on standard data based on significant bits and carry bits for sub-unit boundaries.

8. The circuit of claim 5, wherein the standard ALU includes an ADD operation.

9. The circuit of claim 1, wherein the arithmetic/shift function is a SHIFT instruction.

This Sheet Has Been Replaced

* C L M S *

THIS SHEET HAS BEEN
REPLACED



10. A microprocessor circuit for executing Multi-Media instructions comprising:

a). an Arithmetic Logic Unit (ALU) performing standard operations on instructions in a first cycle of operation; and

b). a correction circuit responsive to the ALU for modifying the standard results provided by the ALU to results required by Single Instruction Multiple Data (SIMD).

11. The circuit of claim 1, wherein the correction circuit modification is performed in a second cycle.

12. The circuit of claim 2, wherein the ALU is free to execute a second instruction in the second cycle.

13. The circuit of claim 1, wherein the ALU and correction circuit can each complete an instruction in the same cycle.

This Sheet Has Been Replaced

* C L M S *

THIS SHEET HAS BEEN
REPLACED



14. A microprocessor circuit comprising:

a). a Shifter performing standard operations on instructions in a first cycle of operation;

b). a mask generating circuit in parallel with the Shifter circuit; and

c). a correction circuit responsive to said Shifter and said mask generating circuit for modifying the standard results provided by said Shifter to results required by a SIMD SHIFT instruction being executed.

15. The circuit of claim 13, wherein the modification is performed in a second cycle.

16. The circuit of claim 14, wherein the modification is performed by an address overlay mask circuit.

17. The circuit of claim 14, wherein the Shifter is free to execute a second instruction in the second cycle.

18. The circuit of claim 13, wherein the Shifter and correction circuit can each complete an instruction in the same cycle.

19. The circuit of claim 13, wherein the Shifter performs a SHIFT operation.

This Sheet Has Been Replaced

# SIMD CORRECTION CIRCUIT FOR ARITHMETIC/SHIFT OPERATIONS

## ABSTRACT

A microprocessor circuit (10) is disclosed for instructions on an arithmetic/shift function performing standard operations (e.g., ALU instructions or Shift instructions) on instructions in a first cycle of operation, and a correction circuit (20) responsive to the arithmetic/shift function for modifying the standard results provided by the arithmetic/shift function to results required by a SIMD instruction being executed. The arithmetic/shift function is an instruction provided by either an Arithmetic Logic Unit (ALU) or by a shift instruction. The correction circuit (20) passes data, unchanged for logical instructions but provides condition codes according to the SIMD instruction.

Fig. 1

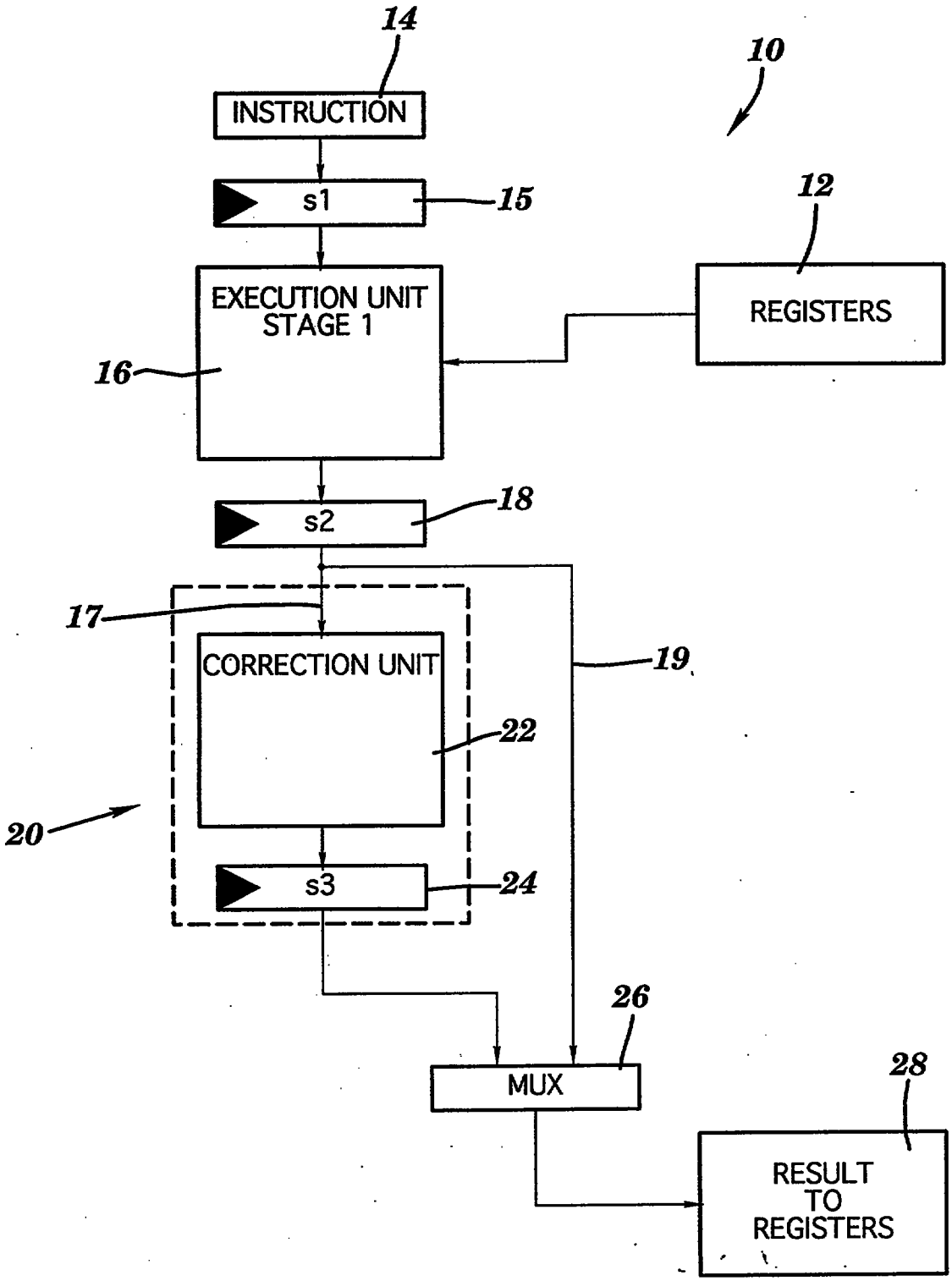

14
INSTRUCTION
10
s1
15
12
REGISTERS
EXECUTION UNIT
STAGE 1
16
s2
18
17
CORRECTION UNIT
19
22
20
s3
24
26
MUX
28
RESULT
TO
REGISTERS

*FIG. 1*

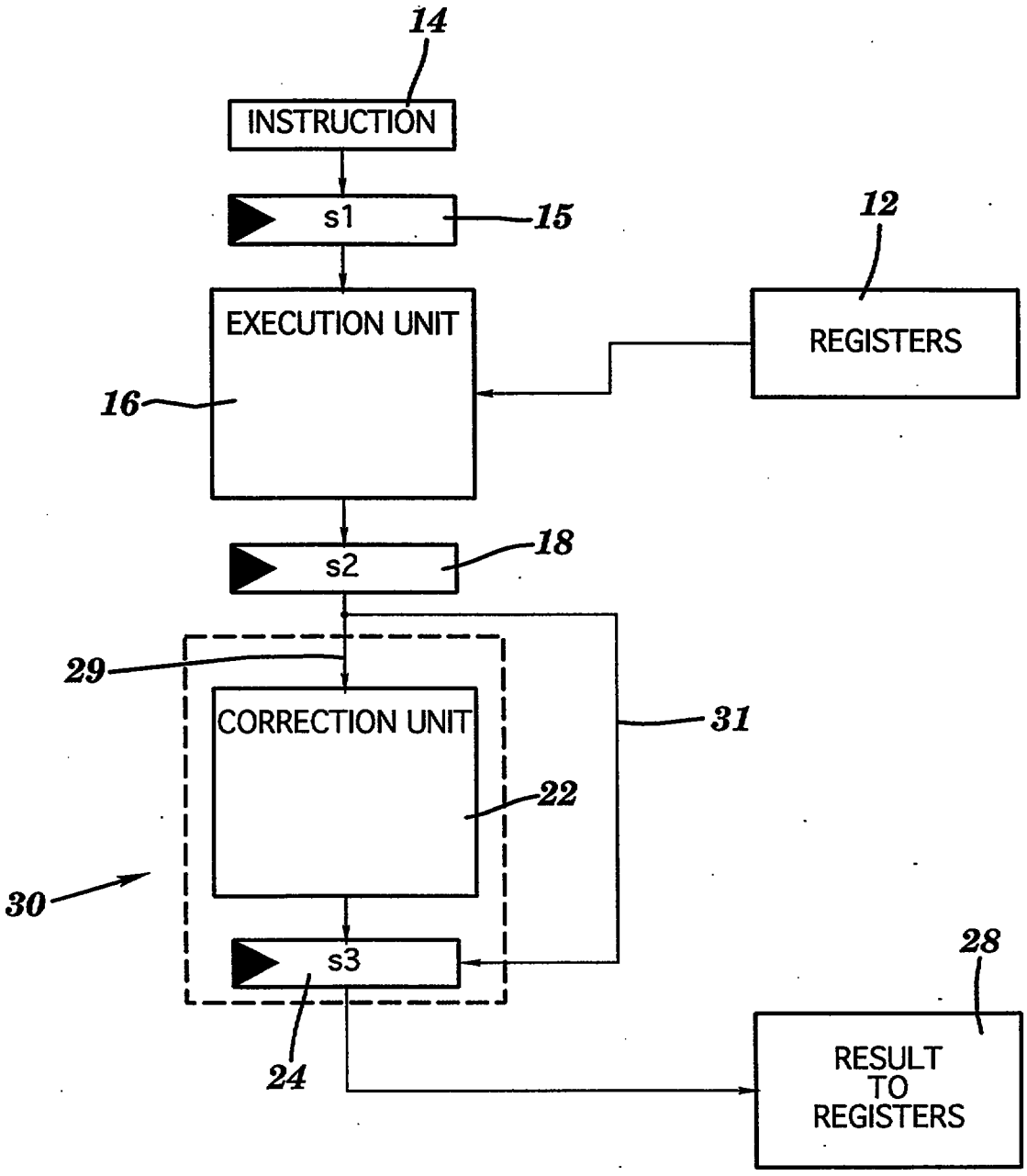

14
INSTRUCTION
s1
15
12
EXECUTION UNIT
REGISTERS
16
s2
18
29
CORRECTION UNIT
31
22
30
s3
28
24
RESULT
TO
REGISTERS

*FIG. 2*

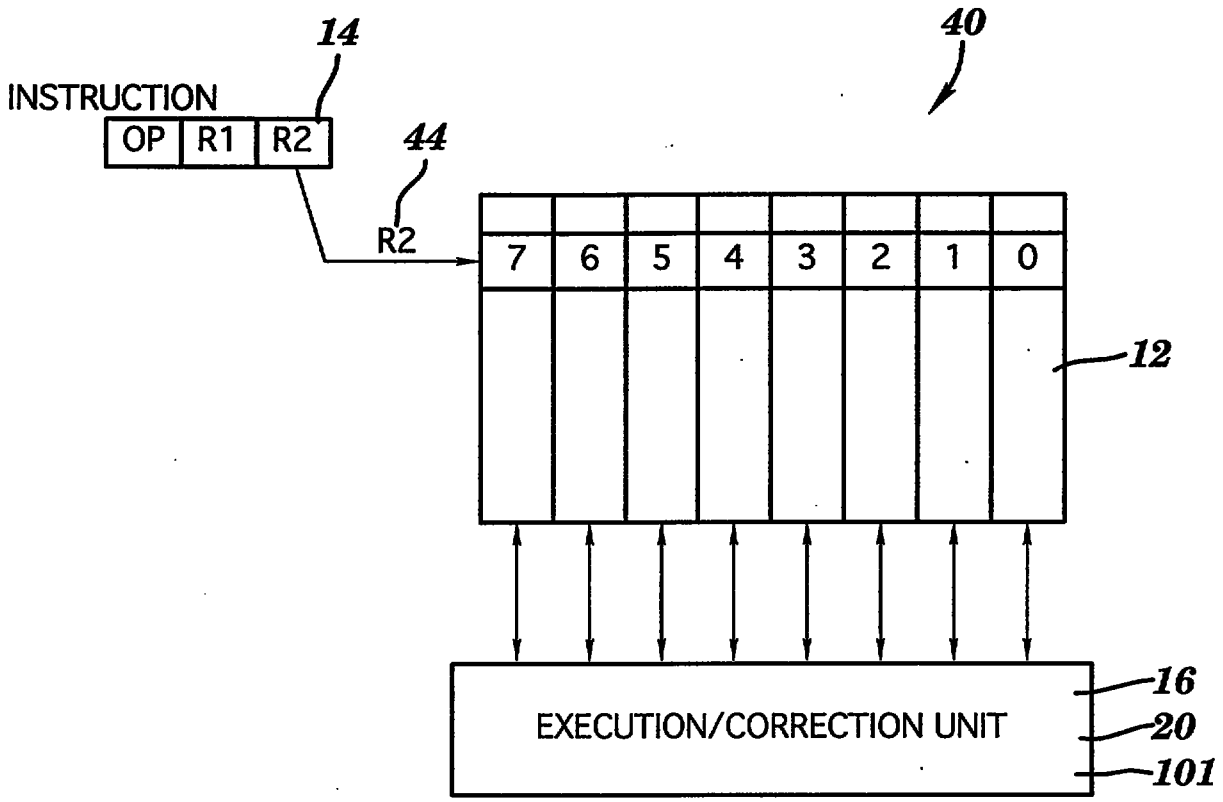

14
40
INSTRUCTION
OP
R1
R2
44
R2
7
6
5
4
3
2
1
0
12
EXECUTION/CORRECTION UNIT
16
20
101

*FIG. 3*

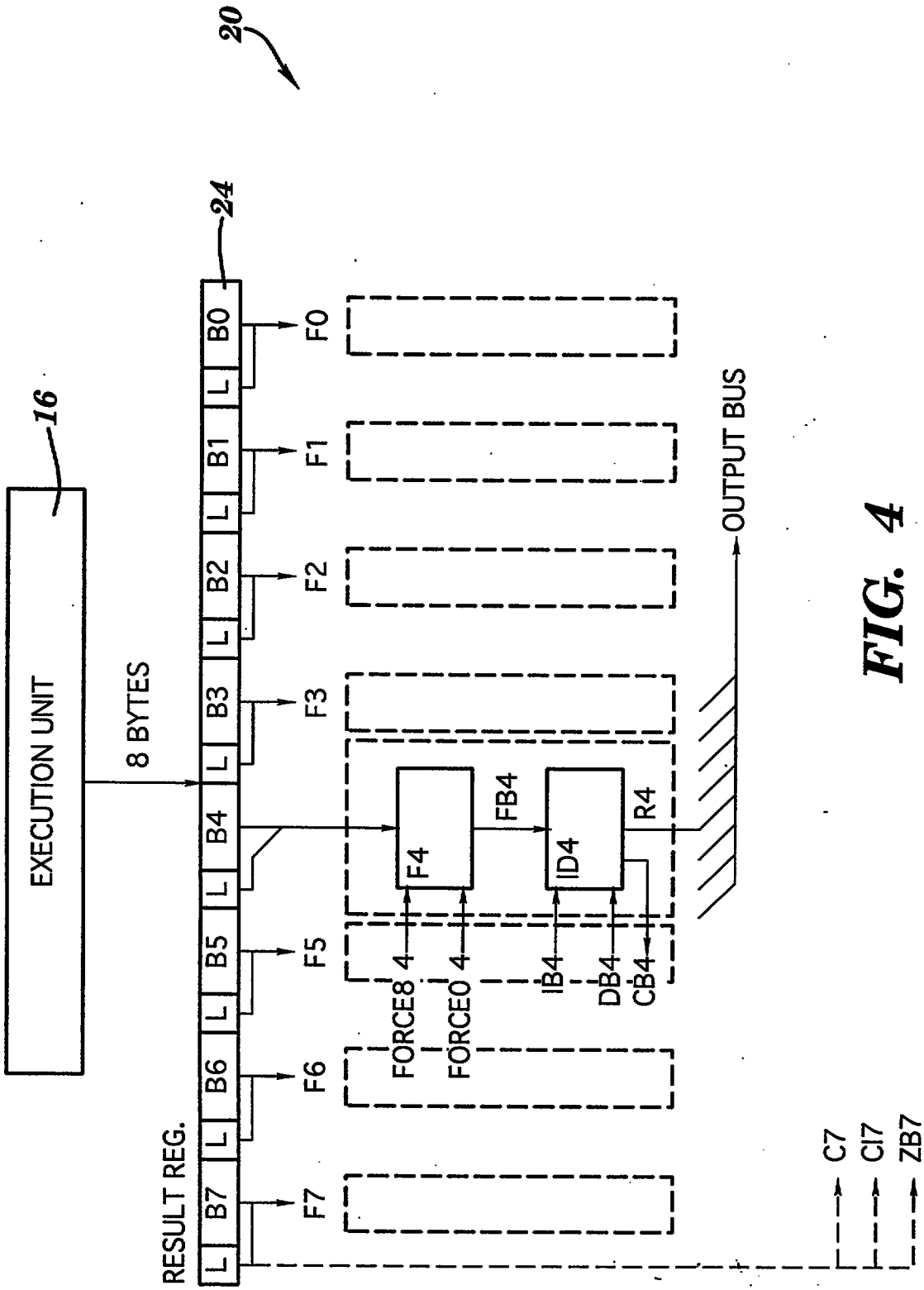

20
24
16
EXECUTION UNIT
8 BYTES
RESULT REG.
B0
B1
B2
B3
B4
B5
B6
B7
L
F0
F1
F2
F3
F4
F5
F6
F7
FB4
ID4
R4
FORCE8 4
FORCE0 4
IB4
DB4
CB4
OUTPUT BUS
C7
CI7
ZB7

FIG. 4

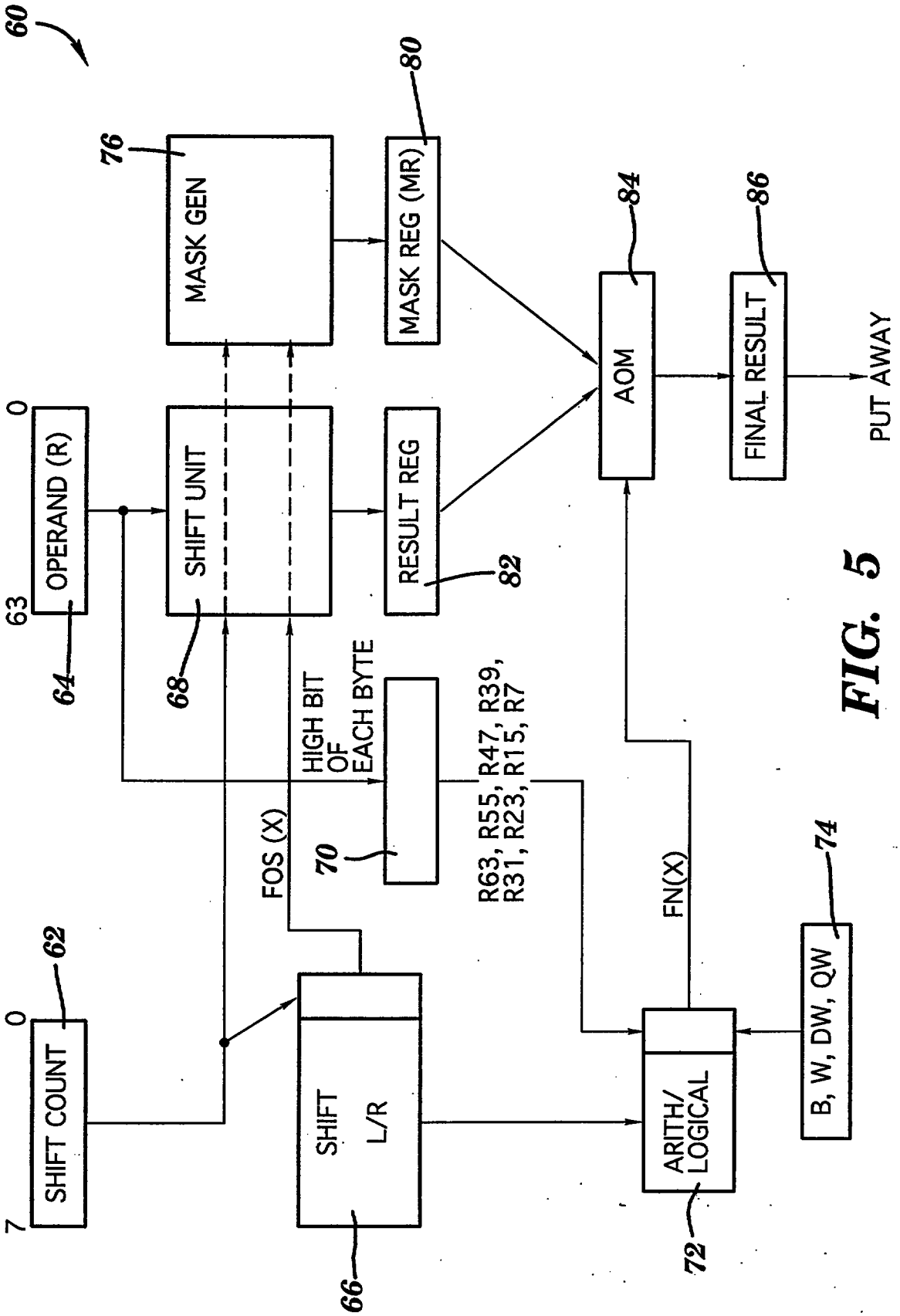

60
76
MASK GEN
80
MASK REG (MR)
84
AOM
86
FINAL RESULT
PUT AWAY
0
63
OPERAND (R)
64
SHIFT UNIT
68
RESULT REG
82
HIGH BIT OF EACH BYTE
70
FOS (X)
R63, R55, R47, R39, R31, R23, R15, R7
FN(X)
0
7
SHIFT COUNT
62
SHIFT L/R
66
ARITH/ LOGICAL
72
B, W, DW, QW
74

FIG. 5

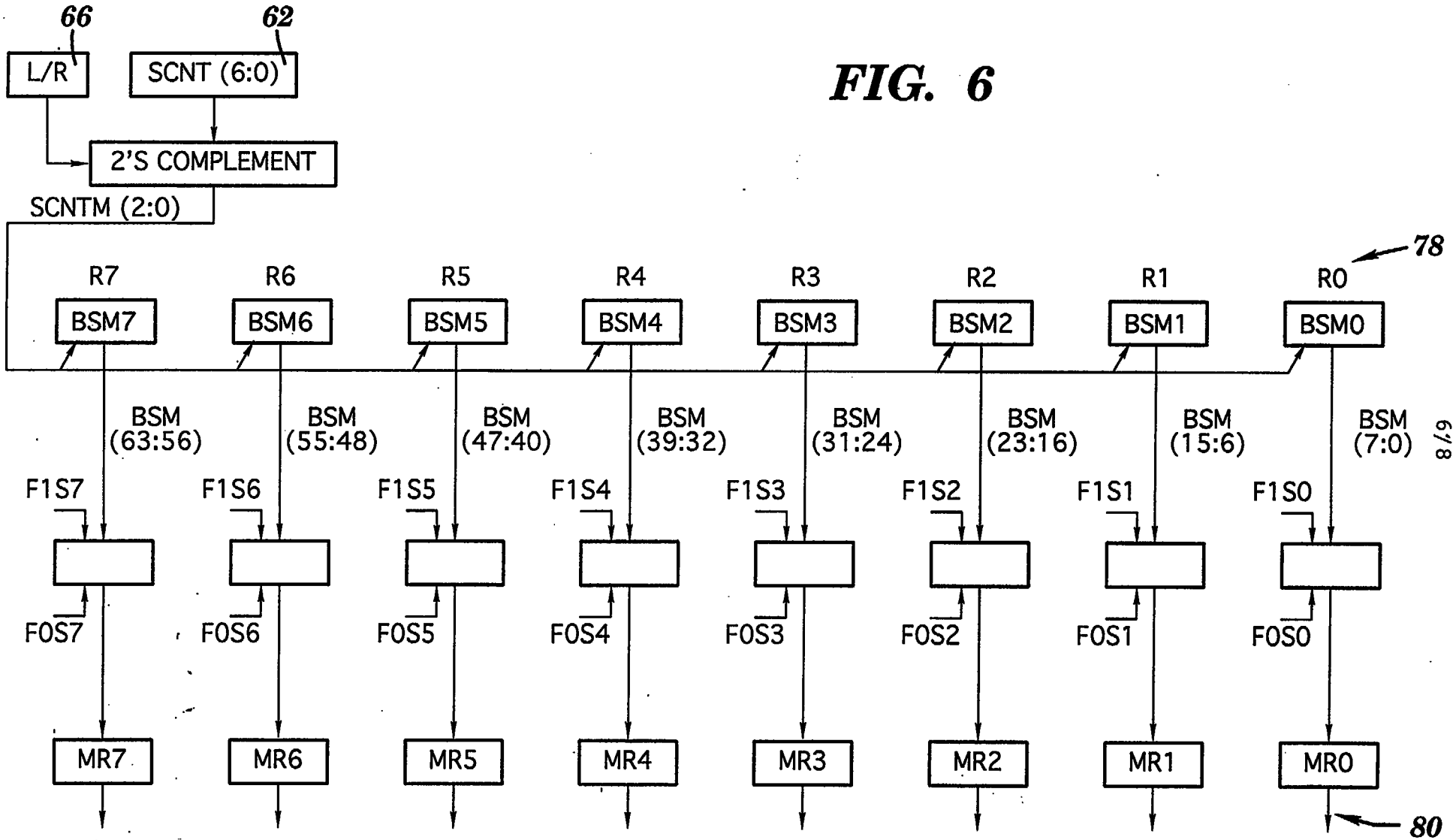

66
62
L/R
SCNT (6:0)
2'S COMPLEMENT
SCNTM (2:0)
78
R7
R6
R5
R4
R3
R2
R1
R0
BSM7
BSM6
BSM5
BSM4
BSM3
BSM2
BSM1
BSM0
BSM (63:56)
BSM (55:48)
BSM (47:40)
BSM (39:32)
BSM (31:24)
BSM (23:16)
BSM (15:6)
BSM (7:0)
F1S7
F1S6
F1S5
F1S4
F1S3
F1S2
F1S1
F1S0
F0S7
F0S6
F0S5
F0S4
F0S3
F0S2
F0S1
F0S0
MR7
MR6
MR5
MR4
MR3
MR2
MR1
MR0
80

*FIG. 6*

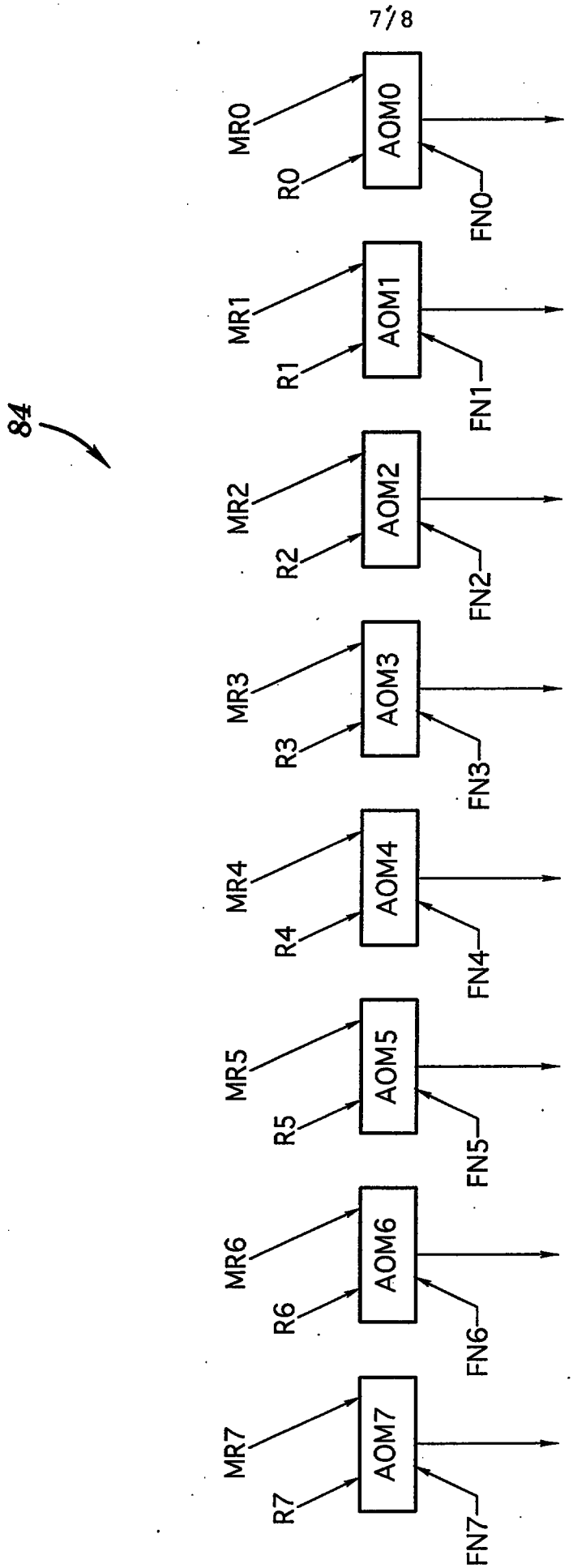

84
MR0
R0
AOM0
FN0
MR1
R1
AOM1
FN1
MR2
R2
AOM2
FN2
MR3
R3
AOM3
FN3
MR4
R4
AOM4
FN4
MR5
R5
AOM5
FN5
MR6
R6
AOM6
FN6
MR7
R7
AOM7
FN7

*FIG. 7*

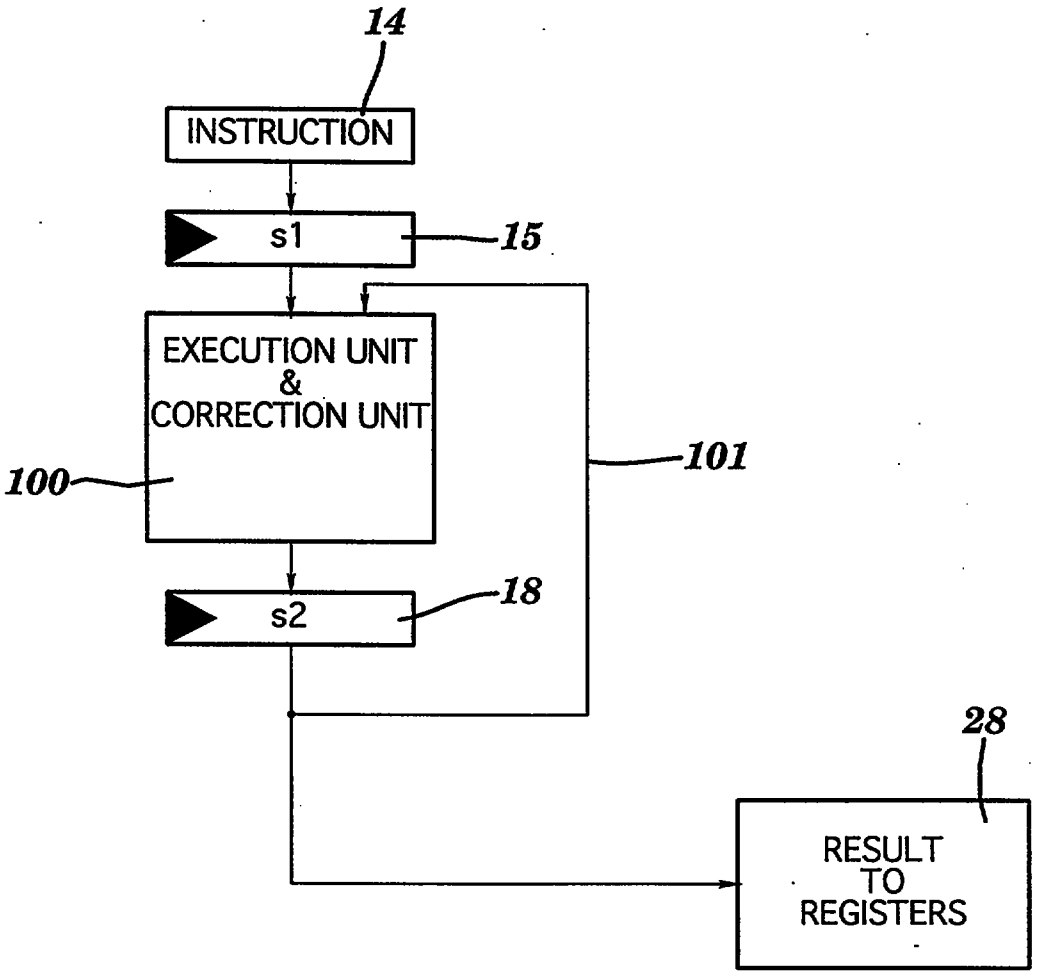

14
INSTRUCTION
s1
15
EXECUTION UNIT
&
CORRECTION UNIT
100
101
s2
18
28
RESULT
TO
REGISTERS

*FIG. 8*

* C L M S *



REPLACEMENT SHEET
11 MAR 1999

What is claimed is:

1. A microprocessor circuit for executing a single instruction multiple data (SIMD) instruction, comprising:

a standard execution unit for receiving SIMD data and performing a non-SIMD operation on the SIMD data during a first cycle and outputting a result in a non-SIMD format; and

a correction system for receiving the result in the non-SIMD format and for modifying the result to a SIMD format.

2. The circuit of claim 1, wherein the correction system modifies the result during a second cycle.

3. The circuit of claim 2, wherein the standard execution unit can perform a second non-SIMD operation on a second set of non-SIMD data during the second cycle.

4. The circuit of claim 1, wherein the standard execution unit comprises an arithmetic logic unit (ALU).

5. The circuit of claim 4, wherein the ALU includes an adder circuit.

* C L M S *

REPLACEMENT SHEET
11 MAR 1999



6. The circuit of claim 4, wherein the correction system comprises:

a register for storing data containing the result in the non-SIMD format; and

a plurality of correction circuits coupled to the register, each correction circuit operating on a portion of the data stored in the register.

7. The circuit of claim 6, wherein each portion of data comprises a subset of the result in the non-SIMD format and a plurality of carry data, wherein the carry data are calculated during the execution of the non-SIMD operation.

8. The circuit of claim 7, wherein each of the plurality of correction circuits comprises a force circuit that can force the respective subset to at least one predetermined value based upon the carry data.

9. The circuit of claim 8, wherein each of the plurality of correction circuits comprises an increment/decrement circuit that can increment and decrement the output of the force circuit.

10. The circuit of claim 1, wherein the standard execution unit is a shifter.

* C L M S *



REPLACEMENT SHEET
11 MAR 1999

11. A microprocessor circuit for executing a Single Instruction Multiple Data (SIMD) arithmetic instruction on SIMD data, comprising:

a standard execution unit for receiving the SIMD arithmetic instruction, for performing a non-SIMD arithmetic operation on the SIMD data during a first cycle, and for outputting a non-SIMD result;

a register for storing data subsets of the non-SIMD result, wherein each data subset further includes a set of related carry data calculated during the non-SIMD arithmetic operation; and

a correction system, comprising a plurality of correction circuits, wherein each correction circuit operates on a unique one of the data subsets, and wherein the correction system outputs a SIMD result.

12. The circuit of claim 11, wherein each correction circuit comprises:

a force circuit that can force the respective data subset to at least one predetermined value based upon the related carry data; and

an increment/decrement circuit that can increment and decrement the output of the force circuit based upon the related carry data.

13. The circuit of claim 12, wherein the arithmetic operation is an add operation.

14. The circuit of claim 11, wherein the correction system operates during a second cycle.

* C L M S *



REPLACEMENT SHEET
11 MAR 1999

15. The circuit of claim 12, wherein the related carry data comprises:

a bit containing a carry out value of a high order bit of the respective data subset;

a bit containing a carry into value of the high bit of the respective data subset; and

a bit for indicating if all bits of the respective data subset are zero.

16. A microprocessor circuit for executing a Single Instruction Multiple Data (SIMD) shift instruction on a SIMD operand comprised of a plurality of data subsets, comprising:

a standard execution unit for receiving the SIMD shift instruction, for performing a non-SIMD shift operation on the SIMD operand during a first cycle, and for outputting a non-SIMD result;

a register for storing the non-SIMD result; and

a correction system for converting the non-SIMD result to a SIMD result, wherein the correction system comprises a mask generator and an and-or mask (AOM).

17. The circuit of claim 16, wherein the mask generator generates a data mask comprised of a plurality of bytes, and wherein each byte corresponds to one of the data subsets of the SIMD operand.

18. The circuit of claim 17, wherein the mask generator includes a plurality of force circuits capable of forcing each of the plurality of bytes to a value of either hex FF or hex 00.