AGH

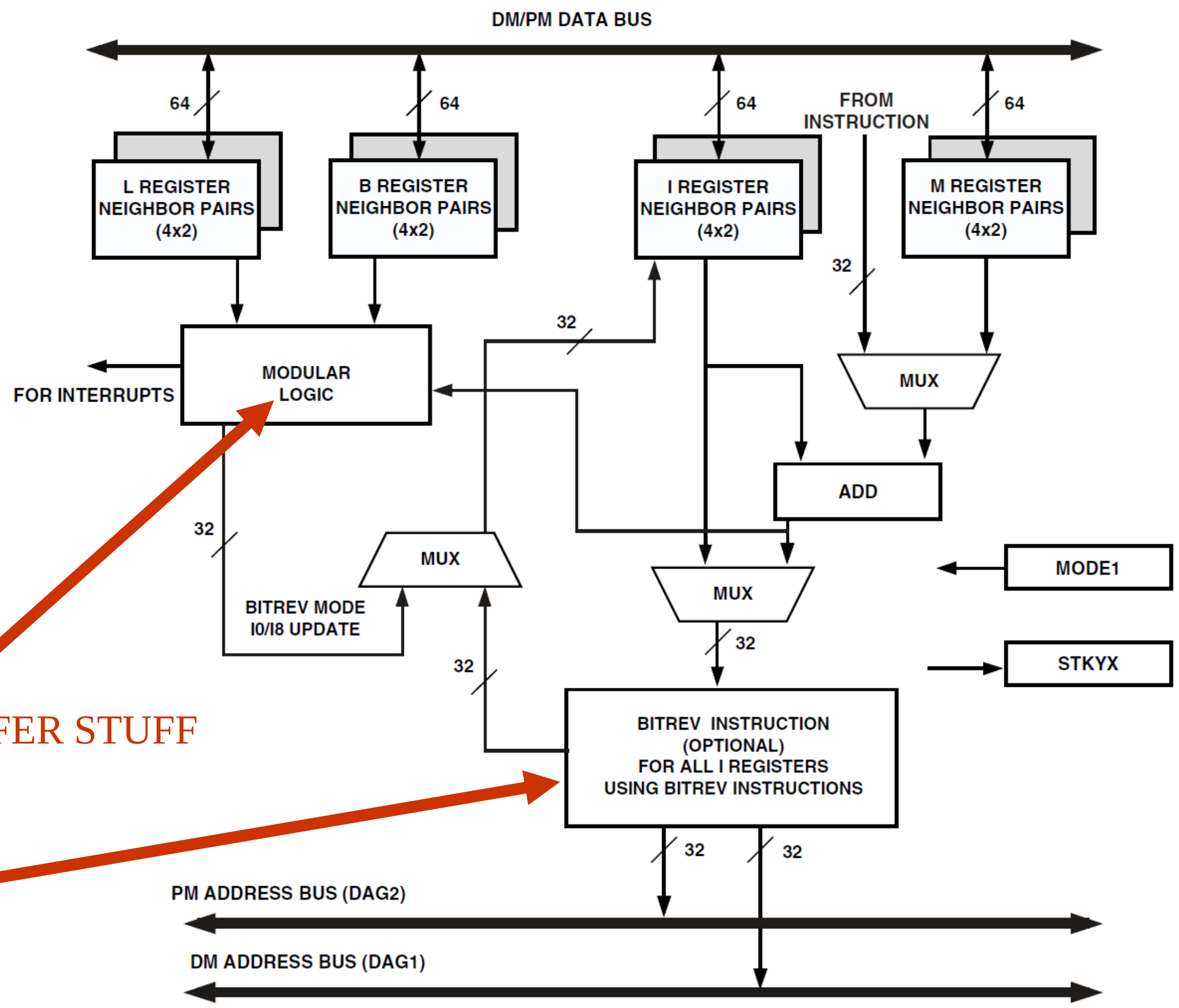
AKADEMIA GÓRNICZO-HUTNICZA
IM. STANISŁAWA STASZICA W KRAKOWIE

Procesory Sygnałowe w aplikacjach przemysłowych

Adresowanie i obliczenia

IET
Katedra Elektroniki
Kraków 2015
dr inż. Roman Rumian

Register and Register Ops in DAG1


DM/PM DATA BUS
64
64
64
FROM INSTRUCTION
64
L REGISTER NEIGHBOR PAIRS (4x2)
B REGISTER NEIGHBOR PAIRS (4x2)
I REGISTER NEIGHBOR PAIRS (4x2)
M REGISTER NEIGHBOR PAIRS (4x2)
32
32
MODULAR LOGIC
FOR INTERRUPTS
MUX
ADD
32
MUX
MUX
MODE1
BITREV MODE I0/I8 UPDATE
32
STKYX
32
BITREV INSTRUCTION (OPTIONAL) FOR ALL I REGISTERS USING BITREV INSTRUCTIONS
32
32
PM ADDRESS BUS (DAG2)
DM ADDRESS BUS (DAG1)

SPECIAL CIRCBUFFER STUFF

SPECIAL FFT BIT

DAG register info

- Index registers
  - I0 -- I7 (dm -- data mem), I8 -- I15 (pm -- program mem)
- Modify registers M0 -- M7, M8 -- M15
  - Can be used for high speed post increment
- Special Hardware for Circular Buffers
  - Base registers   B0 -- B7,   B8 -- B15
  - Length registers L0 -- L7,   L8 -- L15

Address Versus Word Size

The processor’s internal memory accommodates the following word sizes:

- 64-bit long word data (LW)
- 40-bit extended-precision normal word data (NW, 48-bit)
- 32-bit normal word data (NW, 32-bit)
- 16-bit short word data (SW, 16-bit)

ⓘ **Only the address space determines which memory word size is accessed.** An important item to note is that the DAG automatically adjusts the output address per the word size of the address location (short word, normal word, or long word). This address adjustment allows internal memory to use the address directly as shown in the following example.

```
I15=LW_addr;
pm(i15,0)=r0; /* 64-bit transfer */

I7=NW_addr;
dm(i7,0)=r8; /* 32-bit transfer */

I7=SW_addr;
dm(i7,0)=r14; /* 16-bit transfer */
```

Internal memory map

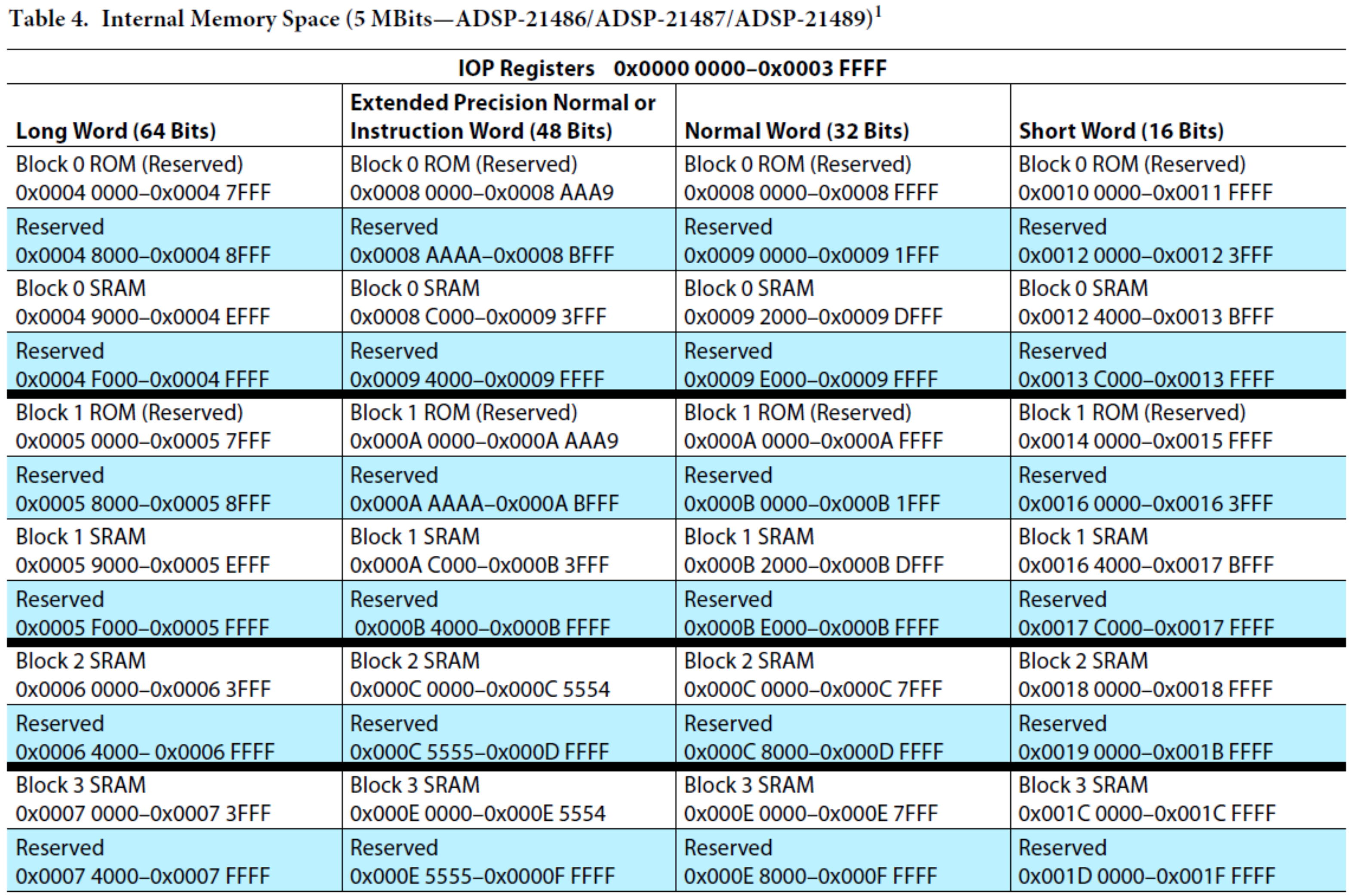


Table 4. Internal Memory Space (5 MBits—ADSP-21486/ADSP-21487/ADSP-21489)[1]

IOP Registers 0x0000 0000–0x0003 FFFF			
Long Word (64 Bits)	**Extended Precision Normal or Instruction Word (48 Bits)**	**Normal Word (32 Bits)**	**Short Word (16 Bits)**
Block 0 ROM (Reserved) 0x0004 0000–0x0004 7FFF	Block 0 ROM (Reserved) 0x0008 0000–0x0008 AAA9	Block 0 ROM (Reserved) 0x0008 0000–0x0008 FFFF	Block 0 ROM (Reserved) 0x0010 0000–0x0011 FFFF
Reserved 0x0004 8000–0x0004 8FFF	Reserved 0x0008 AAAA–0x0008 BFFF	Reserved 0x0009 0000–0x0009 1FFF	Reserved 0x0012 0000–0x0012 3FFF
Block 0 SRAM 0x0004 9000–0x0004 EFFF	Block 0 SRAM 0x0008 C000–0x0009 3FFF	Block 0 SRAM 0x0009 2000–0x0009 DFFF	Block 0 SRAM 0x0012 4000–0x0013 BFFF
Reserved 0x0004 F000–0x0004 FFFF	Reserved 0x0009 4000–0x0009 FFFF	Reserved 0x0009 E000–0x0009 FFFF	Reserved 0x0013 C000–0x0013 FFFF
Block 1 ROM (Reserved) 0x0005 0000–0x0005 7FFF	Block 1 ROM (Reserved) 0x000A 0000–0x000A AAA9	Block 1 ROM (Reserved) 0x000A 0000–0x000A FFFF	Block 1 ROM (Reserved) 0x0014 0000–0x0015 FFFF
Reserved 0x0005 8000–0x0005 8FFF	Reserved 0x000A AAAA–0x000A BFFF	Reserved 0x000B 0000–0x000B 1FFF	Reserved 0x0016 0000–0x0016 3FFF
Block 1 SRAM 0x0005 9000–0x0005 EFFF	Block 1 SRAM 0x000A C000–0x000B 3FFF	Block 1 SRAM 0x000B 2000–0x000B DFFF	Block 1 SRAM 0x0016 4000–0x0017 BFFF
Reserved 0x0005 F000–0x0005 FFFF	Reserved 0x000B 4000–0x000B FFFF	Reserved 0x000B E000–0x000B FFFF	Reserved 0x0017 C000–0x0017 FFFF
Block 2 SRAM 0x0006 0000–0x0006 3FFF	Block 2 SRAM 0x000C 0000–0x000C 5554	Block 2 SRAM 0x000C 0000–0x000C 7FFF	Block 2 SRAM 0x0018 0000–0x0018 FFFF
Reserved 0x0006 4000– 0x0006 FFFF	Reserved 0x000C 5555–0x000D FFFF	Reserved 0x000C 8000–0x000D FFFF	Reserved 0x0019 0000–0x001B FFFF
Block 3 SRAM 0x0007 0000–0x0007 3FFF	Block 3 SRAM 0x000E 0000–0x000E 5554	Block 3 SRAM 0x000E 0000–0x000E 7FFF	Block 3 SRAM 0x001C 0000–0x001C FFFF
Reserved 0x0007 4000–0x0007 FFFF	Reserved 0x000E 5555–0x0000F FFFF	Reserved 0x000E 8000–0x000F FFFF	Reserved 0x001D 0000–0x001F FFFF

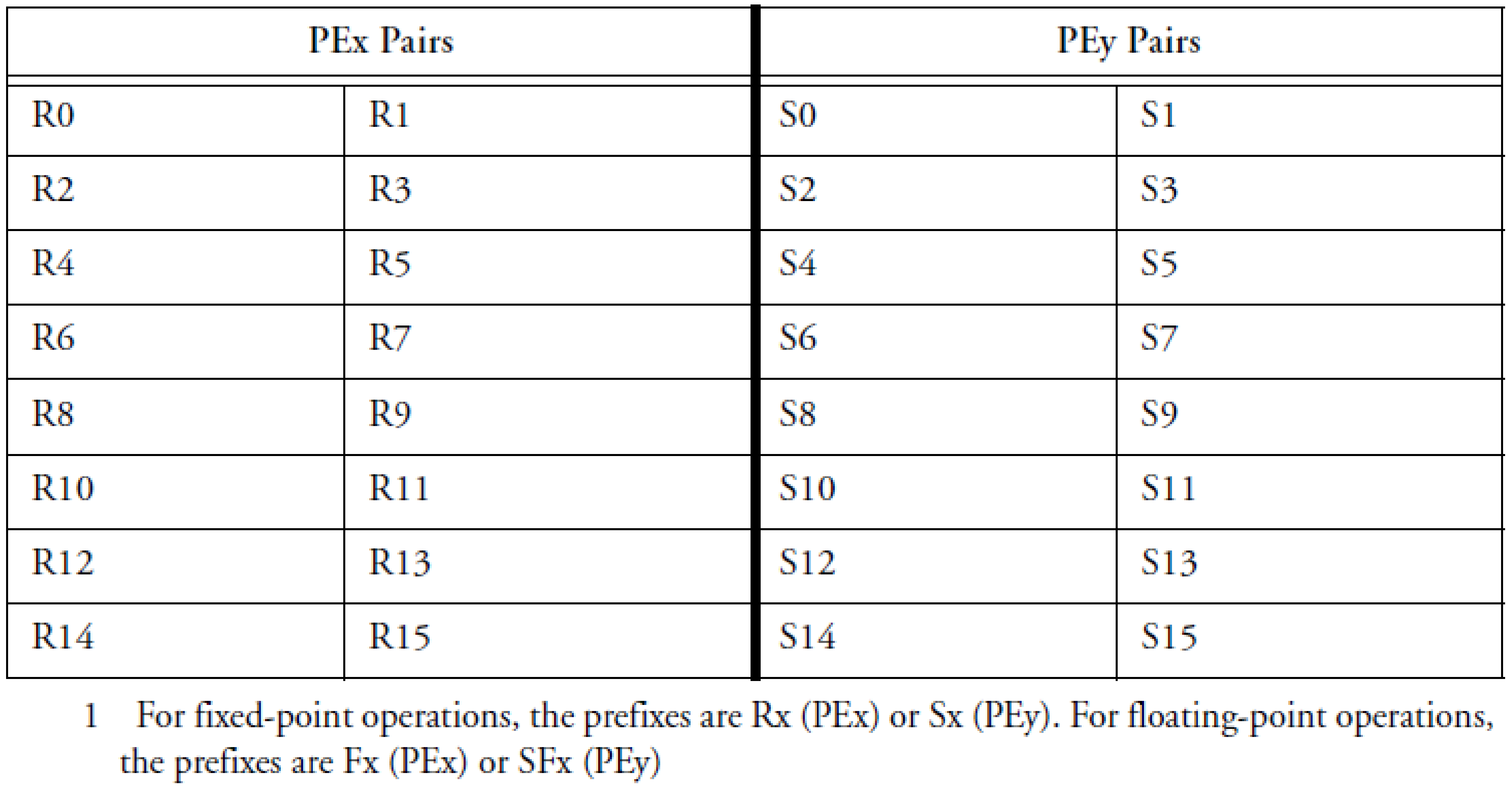
Instrukcje skoków bezwarunkowych i warunkowych

```
JUMP etykieta //skok bezwarunkowy

CALL etykieta //bezwarunkowe wywołanie podprogramu



IF NE JUMP etykieta //skok warunkowy

IF AC CALL etykieta //warunkowe wywołanie podprogramu
```

Niektóre pozostałe instrukcje także mogą być wykonywane warunkowo, np. :

```
IF EQ DM(I0,M0) = R2;
IF EQ R8 = R2;
```

Data Register Pairs for SIMD and LW Access



PEx Pairs		PEy Pairs	
R0	R1	S0	S1
R2	R3	S2	S3
R4	R5	S4	S5
R6	R7	S6	S7
R8	R9	S8	S9
R10	R11	S10	S11
R12	R13	S12	S13
R14	R15	S14	S15

1 For fixed-point operations, the prefixes are Rx (PEx) or Sx (PEy). For floating-point operations, the prefixes are Fx (PEx) or SFx (PEy)

Data and Complementary Data Register Access Priorities

If writes to the same location take place in the same cycle, only the write with higher precedence actually occurs. The processor determines precedence for the write operation from the source of the data; from highest to lowest, the precedence is:

1. DAG1 or universal register (UREG)
2. DAG2
3. PEx ALU
4. PEy ALU
5. PEx Multiplier
6. PEy Multiplier
7. PEx Shifter
8. PEy Shifter

Example:

```
r0=r1+r2, r0=dm(i0,m0), r0=pm(i8,m8); /* r0 is loaded from i0*/
r0=r1+r2, r0=pm(i8,m8); /* r0 is loaded from i8 */
```

Data and Complementary Data Register Swaps

Registers swaps use the special swap operator, <->. A register-to-register swap occurs when registers in different processing elements exchange values; for example R0 <-> S1. Only single, 40-bit register-to-register swaps are supported. Double register operations are not supported as shown in the example below.

```
R7 <-> S7;
R2 <-> S0;
```

Regardless of SIMD/SISD mode, the processor supports bidirectional register-to-register swaps. The swap occurs between one register in each processing element's data register file.

Processor supports unidirectional and bidirectional register-to-register transfers with the Conditional Compute and Move instruction.

System Register Bit Manipulation

The system registers (SREG) support fast bit manipulation. The next example uses the shifter for bit manipulations:

```
R1 = MODE1;
R1 = BSET R1 by 21; /* sets PEYEN bit */
R1 = BSET R1 by 24; /* sets CBUFEN bit */
MODE1 = R1;
```

However the following example is more efficient.

```
BIT SET MODE1 PEYEN|CBUFEN; /* change both modes */
Nop; /* effect latency */
```

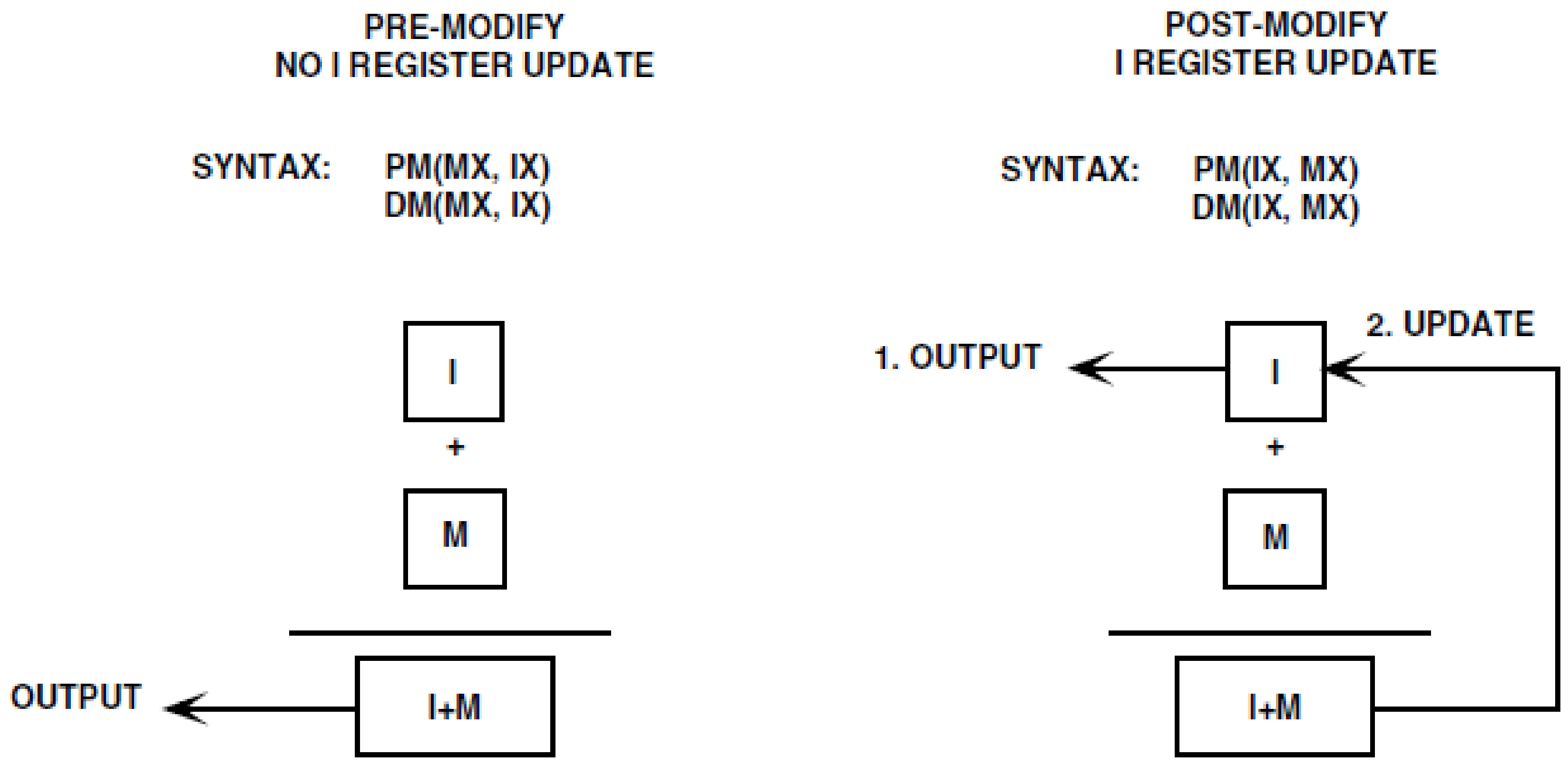
To set or test individual bits in a control register using the shifter:

```
R1 = dm(SYSCTL);
R1 = BSET R1 by 11; /* sets IMDW2 bit 11 */
R1 = BSET R1 by 12; /* sets IMDW3 bit 12 */
dm(SYSCTL) = R1;
BTST R1 by 11; /* clears SZ bit */
IF SZ jump func;
BTST R1 by 12; /* clears SZ bit */
IF SZ jump func;
```

The core has four user status registers (USTAT4–1) also classified as system registers but for general-purpose use. These registers allow flexible manipulation/ testing of single or multiple individual bits in a register without affecting neighbor bits as shown in the following example.

```
USTAT1= dm(SYSCTL);
BIT SET USTAT1 IMDW2|IMDW3; /* sets bits 12-11 */
dm(SYSCTL)=USTAT1;
USTAT1= dm(SYSCTL);
BIT TST USTAT1 IMDW2|IMDW3; /* test bits 12-11 */
IF TF r15=r15+1; /* BTF = 1 PEx OR PEy */
```

Pre-Modify and Post-Modify Operations



Post-Modify Addressing

```
BIT CLR MODE1 CBUFEN; /* clear circular buffer*/
nop;
I1 = buffer; /* Index Pointer */
M1 = 1; /* Modify */
instruction; /* stall, any non-DAG instruction */
instruction; /* stall, any non-DAG instruction */
R3 = dm(I1,M1); /* 1st access */
R3 = dm(I1,M1); /* 2nd access */
```

Modify Instruction

The MODIFY instruction modifies addresses in any DAG index register (I0-I15) without accessing memory.
The MODIFY instruction accepts either a 32-bit immediate value or an M register as the modifier. The following example adds 4 to I1 and updates I1 with the new value.

```
MODIFY(I1,4);
```

Modify Instruction

If the I register's corresponding B and L registers are set up for circular buffering, a MODIFY instruction performs the specified buffer wraparound (if needed).

```
B0 = 0x40000;
L0 = 0x10000;
I0 = 0x4ffff;
I1 = modify(I0, 2); // I1 == 0x40001
```

`Ib = MODIFY(Ia,Mc);` is an enhanced version of the MODIFY instruction. This instruction loads the modified index pointer into another index register.

If the source and destination registers are different, then:

- The source register (Ia) is not updated.
- The destination register (Ib) receives the result of the modify.

The following example instruction accepts up to 32-bit modifiers:

```
R1 = DM(0x40000000,I1); /* DM address = I1 + 0x4000 0000 */
```

The following example instruction accepts up to 6-bit modifiers:

```
PM(I8,0x0B)= ASTATx; /* PM address = I8, I8 = I8 + 0x0B */
```

Bit-Reverse Instruction

The BITREV instruction modifies and bit-reverses addresses in any DAG index register (I0–I15) without accessing memory. This instruction is independent of the bit-reverse mode. The BITREV instruction adds a 32-bit immediate value to a DAG index register, bit-reverses the result, and writes the result back to the same index register. The following example adds 4 to I1, bit-reverses the result, and updates I1 with the new value:

```
BITREV(I1,4);
```

An enhanced version of the BITREV instruction (ADSP-214xx), that loads the bit reversed index pointer into another index register is shown below:

```
I6 = BITREV(I1,0);
```

Bit-Reverse Mode

The BR0 and BR8 bits in the MODE1 register enable the bit-reverse addressing mode where addresses are output in reverse bit order. When BR0 is set (= 1), DAG1 bit-reverses 32-bit addresses output from I0. When BR8 is set (= 1), DAG2 bit-reverses 32-bit addresses output from I8. The DAGs bit-reverse only the address output from I0 or I8; the contents of these registers are not reversed. Bit-reverse addressing mode effects post-modify operations.

```
BIT SET MODE1 BR0; /* Enables bit-rev. addressing for DAG1 */
IO = 0x83000 /* Loads I0 with the bit reverse of the buffer's base
address DM(0xC1000) */
M0 = 0x4000000; /* Loads M0 with value for post-modify, which is the
bit reverse value of the modifier value M0 = 32 */
R1 = DM(I0,M0); /* Loads R1 with contents of DM address DM(0xC1000),
which is the bit-reverse of 0x83000, then post-modifies I0 for the
next access with (0x83000 + 0x4000000) = 0x4083000, which is the bit-
reverse of DM(0xC1020) */
```

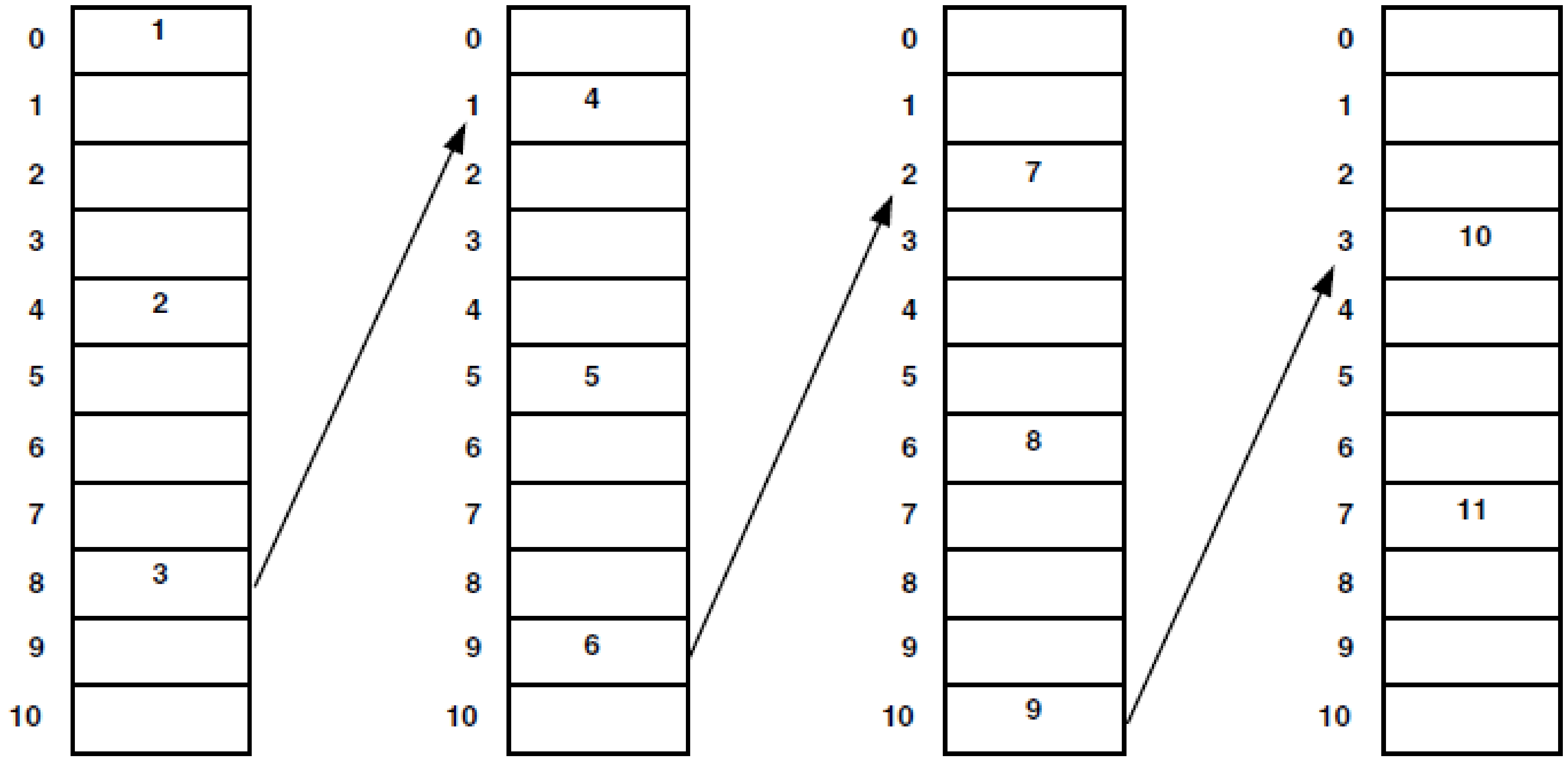
Circular Buffering Mode

The CBUFEN bit in the MODE1 register enables circular buffering—a mode where the DAG supplies addresses that range within a constrained buffer length (set with an L register). Circular buffers start at a base address (set with a B register), and increment addresses on each access by a modify value (set with an M register).
The circular buffer enable bit (CBUFEN) in the MODE1 register is cleared (= 0) at processor reset.

```
Bit Set Mode1 CBUFEN;
```

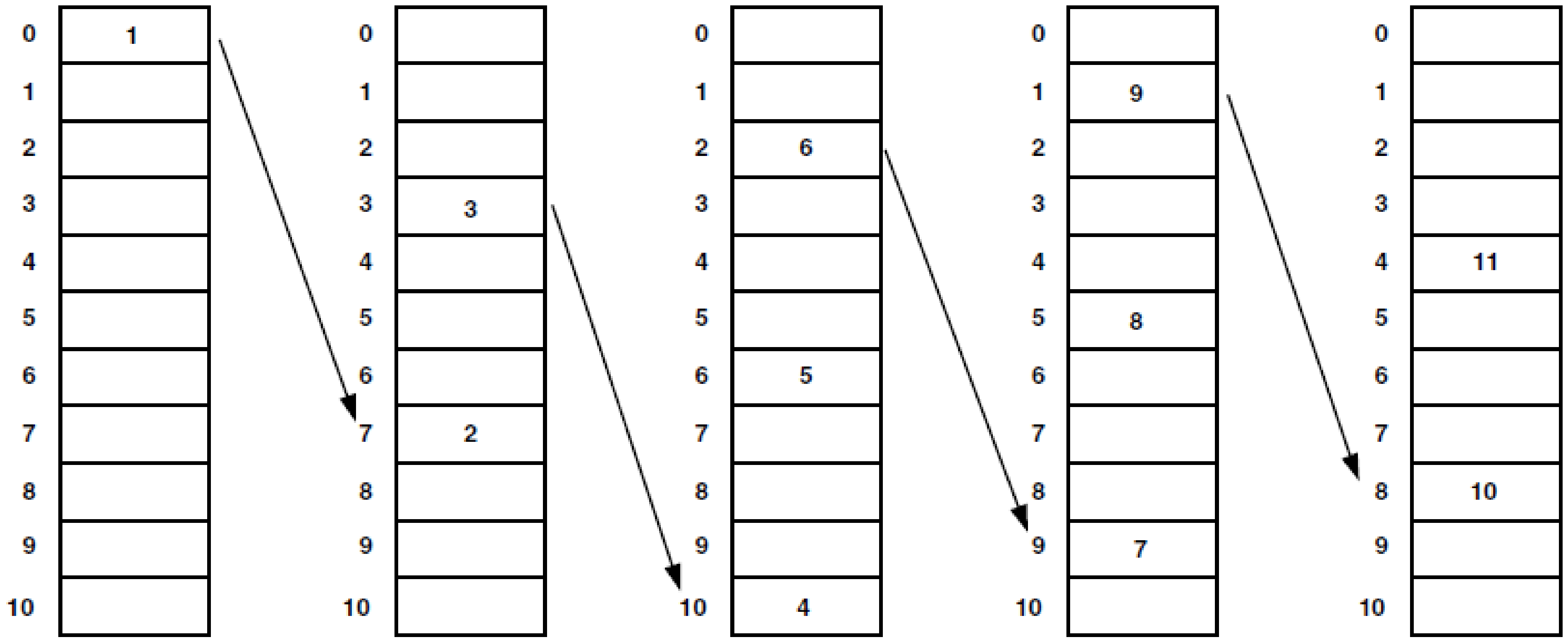
When using circular buffers, the DAGs can generate an interrupt on buffer overflow (wraparound).
programs use the following steps to set up a circular buffer:
1. Enable circular buffering (`BIT SET MODE1 CBUFEN;`). This operation is only needed once in a program.
2. Load the buffer's base address into the B register. This operation automatically loads the corresponding I register. If an offset is required the I register can be changed accordingly.
3. Load the buffer's length into the corresponding L register. For example, L0 corresponds to B0.
4. Load the modify value (step size) into an M register in the corresponding DAG. For example, M0 through M7 correspond to B0.
Alternatively, the program can use an immediate value for the modifier.

Circular Buffering Mode



THE COLUMNS ABOVE SHOW THE SEQUENCE IN ORDER OF LOCATIONS ACCESSED IN ONE PASS
NOTE THAT “0” ABOVE IS BASE ADDRESS. THE SEQUENCE REPEATS ON SUBSEQUENT PASSES

Circular Buffering Mode



Broadcast Load Mode

The processor’s BDCST1 and BDCST9 bits in the MODE1 register control broadcast register loading. When broadcast loading is enabled, the procesor writes to complementary registers or complementary register pairs in each processing element on writes that are indexed with DAG1 register I1 (if BDCST1 =1) or DAG2 register I9 (if BDCST9 =1). Broadcast load accesses are similar to SIMD mode accesses in that the processor transfers both an explicit (named) location and an implicit (unnamed, complementary) location. However, broadcast loading only influences writes to registers and writes identical data to these registers. Broadcast Load Mode performs memory reads only. Broadcast mode only operates with data registers (DREG) or complement data registers (CDREG). Enabling either DAG register to perform a broadcast load has no effect on register stores or loads to universal registers (Ureg). For example:

```
R0=DM(I1,M1); /* I1 load to R0 and S0 */
S10=PM(I9,M9); /* I9 load to S10 and R10 */
```

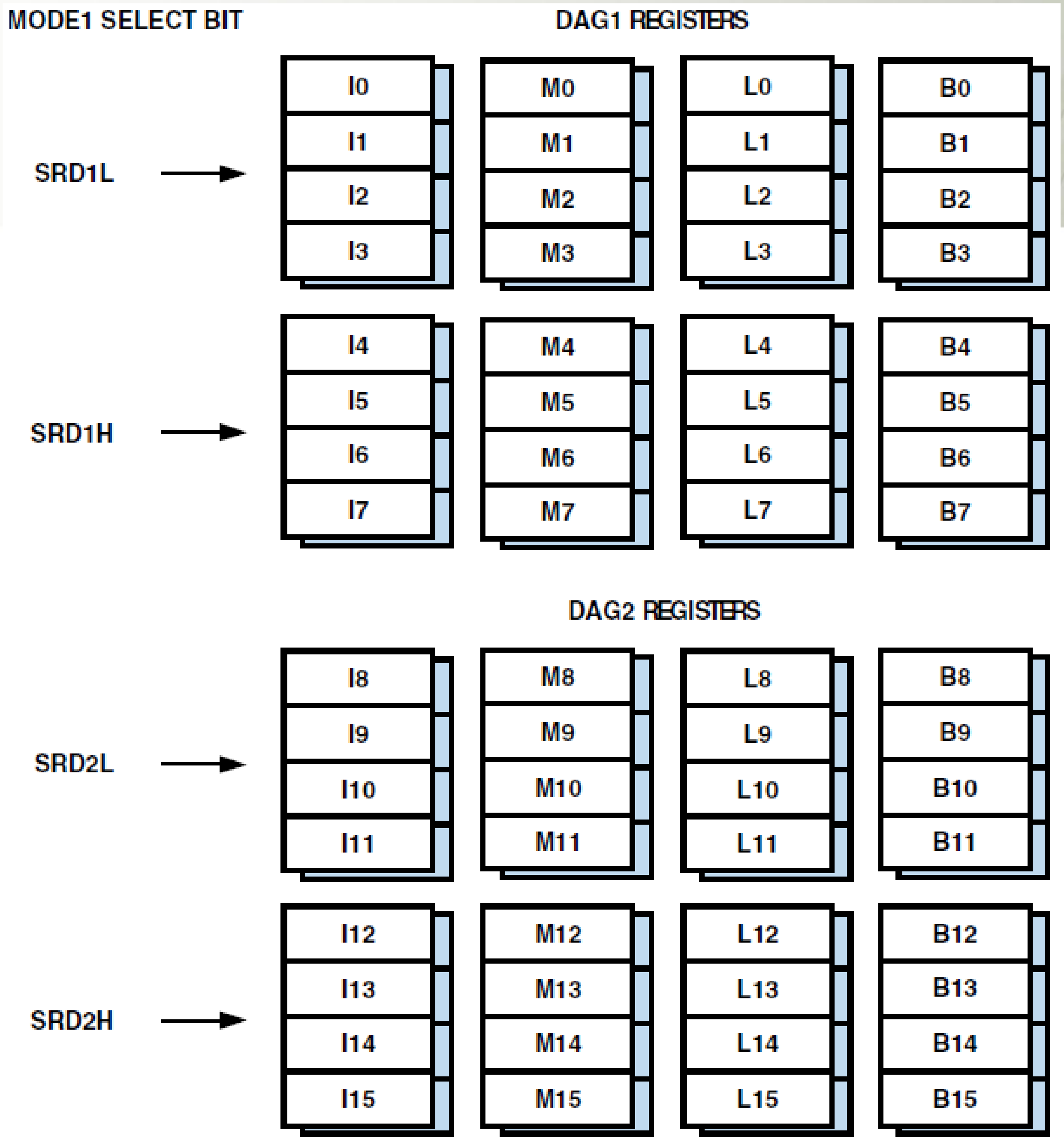
Instruction Summary Broadcast Load

Explicit, PEx Operation	Implicit, PEy operation
Rx = dm(i1,ma);	Sx = dm(i1,ma);
Rx = pm(i9,mb);	Sx = pm(i9,mb);
Rx = dm(i1,ma), Ry = pm(i9,mb);	Sx = dm(i1,ma), Sy = pm(i9,mb);

Alternate (Secondary) DAG Registers

To facilitate fast context switching, the processor includes alternate register sets for all DAG registers. Bits in the MODE1 register control when alternate registers become accessible. While inaccessible, the contents of alternate registers are not affected by processor operations. Note that there is a one cycle latency between writing to MODE1 and being able to access an alternate register set. The alternate register sets for the DAGs are described in this section. For more information on alternate data and results registers, see “Alternate (Secondary) Data Registers” on page 2-14.
Bits in the MODE1 register can activate alternate register sets within the DAGs: the lower half of DAG1 (I, M, L, B0–3), the upper half of DAG1 (I, M, L, B4–7), the lower half of DAG2 (I, M, L, B8–11), and the upper half of DAG2 (I, M, L, B12–15). Figure 6-8 shows the primary and alternate register sets of the DAGs.

```
Example 1
BIT SET MODE1 SRD1L; /* Activate alternate dag1 lo regs */
NOP; /* Wait for access to alternates */
R0 = DM(i0,m1);
Example 2
BIT SET MODE1 SRD1L; /*activate alternate dag1 lo registers */
R13 = R12 + R11; /* Any unrelated instruction */
R0 = DM(I0,M1);
```


MODE1 SELECT BIT
DAG1 REGISTERS
SRD1L
SRD1H
I0
I1
I2
I3
M0
M1
M2
M3
L0
L1
L2
L3
B0
B1
B2
B3
I4
I5
I6
I7
M4
M5
M6
M7
L4
L5
L6
L7
B4
B5
B6
B7
DAG2 REGISTERS
SRD2L
SRD2H
I8
I9
I10
I11
M8
M9
M10
M11
L8
L9
L10
L11
B8
B9
B10
B11
I12
I13
I14
I15
M12
M13
M14
M15
L12
L13
L14
L15
B12
B13
B14
B15