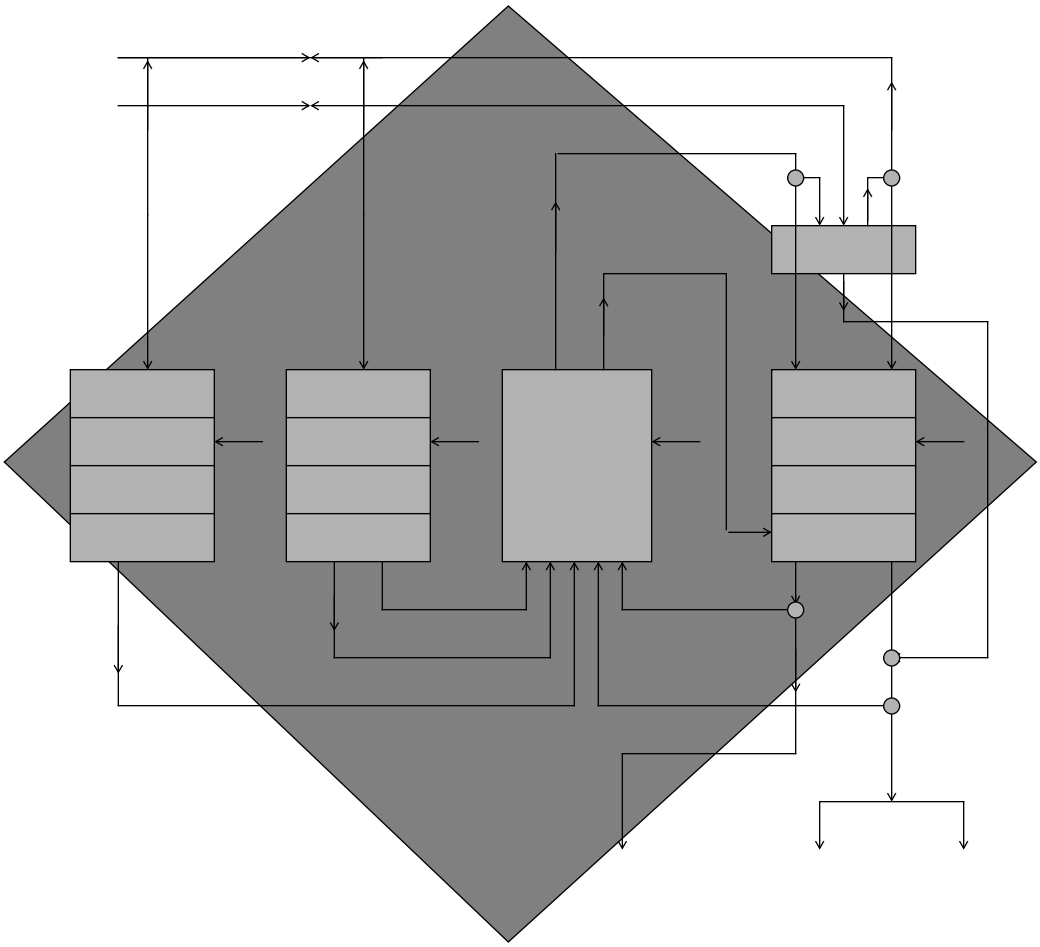
SECTION 4
ADDRESS GENERATION UNIT



SECTION CONTENTS

4.1 ADDRESS GENERATION UNIT AND ADDRESSING MODES

This section contains three major subsections. The first subsection describes the hardware architecture of the address generation unit (AGU), the second subsection describes the programming model, and the third subsection describes the addressing modes, explaining how the Rn, Nn, and Mn registers work together to form a memory address.

4.2 AGU ARCHITECTURE

The AGU is shown in the DSP56K block diagram in Figure 4-1. It uses integer arithmetic to perform the effective address calculations necessary to address data operands in memory, and contains the registers used to generate the addresses. It implements linear, modulo, and reverse-carry arithmetic, and operates in parallel with other chip resources to minimize address-generation overhead.

The AGU is divided into two identical halves, each of which has an address arithmetic logic unit (ALU) and four sets of three registers (see Figure 4-2). They are the address registers (R0 - R3 and R4 - R7), offset registers (N0 - N3 and N4 - N7), and the modifier registers (M0 - M3 and M4 - M7). The eight Rn, Nn, and Mn registers are treated as register triplets — e.g., only N2 and M2 can be used to update R2. The eight triplets are R0:N0:M0, R1:N1:M1, R2:N2:M2, R3:N3:M3, R4:N4:M4, R5:N5:M5, R6:N6:M6, and R7:N7:M7.

The two arithmetic units can generate two 16-bit addresses every instruction cycle — one for any two of the XAB, YAB, or PAB. The AGU can directly address 65,536 locations on the XAB, 65,536 locations on the YAB, and 65,536 locations on the PAB. The two independent address ALUs work with the two data memories to feed the data ALU two operands in a single cycle. Each operand may be addressed by an Rn, Nn, and Mn triplet.

4.2.1 Address Register Files (Rn)

Each of the two address register files (see Figure 4-2) consists of four 16-bit registers. The two files contain address registers R0 - R3 and R4 - R7, which usually contain addresses used as pointers to memory. Each register may be read or written by the global data bus (GDB). When read by the GDB, 16-bit registers are written into the two least significant bytes of the GBD, and the most significant byte is set to zero. When written from the GBD, only the two least significant bytes are written, and the most significant byte is truncated. Each address register can be used as input to its associated address ALU for a register update calculation. Each register can also be written by the output of its respective address ALU. One Rn register from the low address ALU and one Rn register from the high address ALU can be accessed in a single instruction.

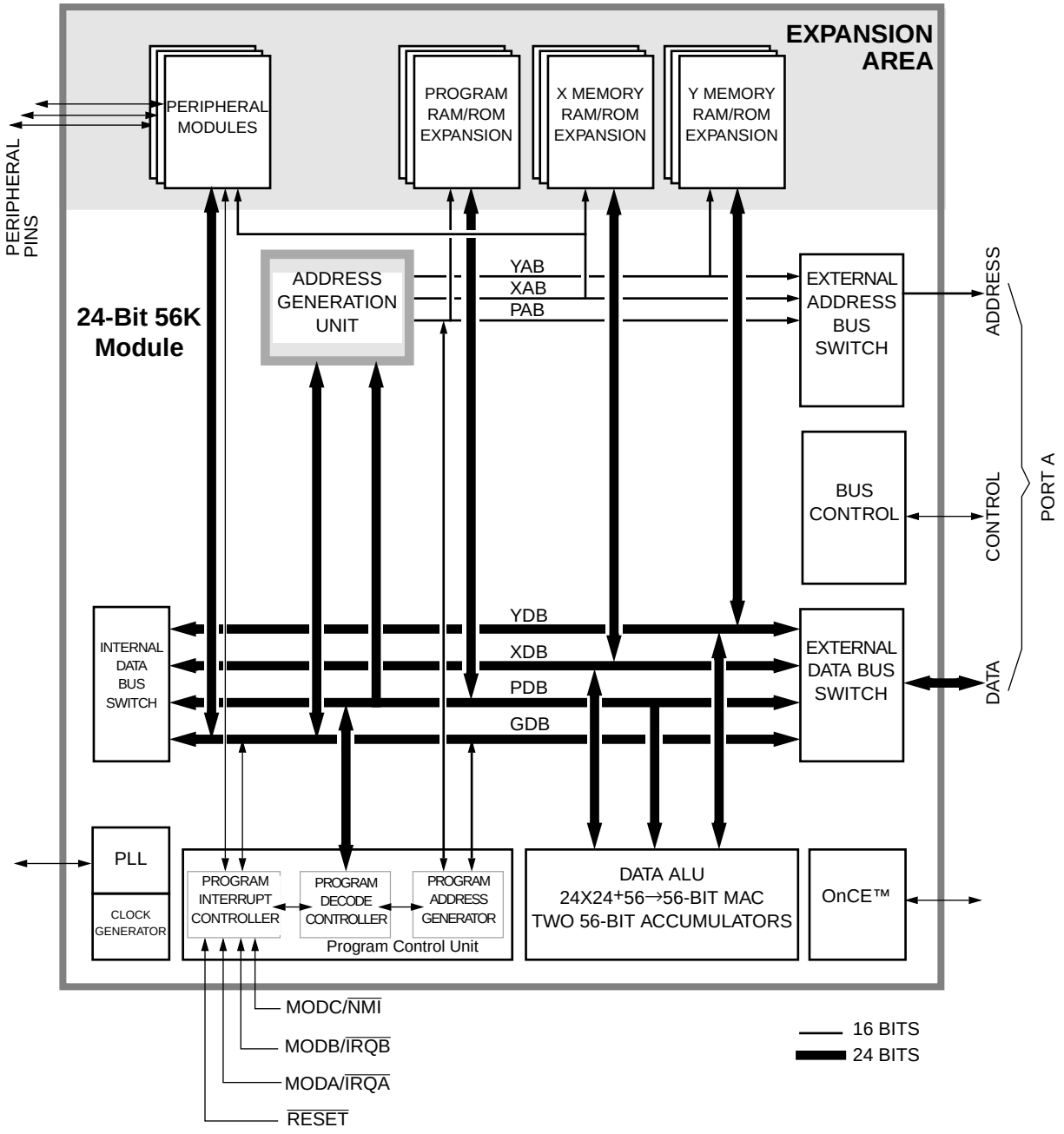


Figure 4-1 DSP56K Block Diagram

4.2.2 Offset Register Files (Nn)

Each of two offset register files shown in Figure 4-2 consists of four 16-bit registers. The two files contain offset registers N0 - N3 and N4 - N7, which contain either data or offset values used to update address pointers. Each offset register can be read or written by the

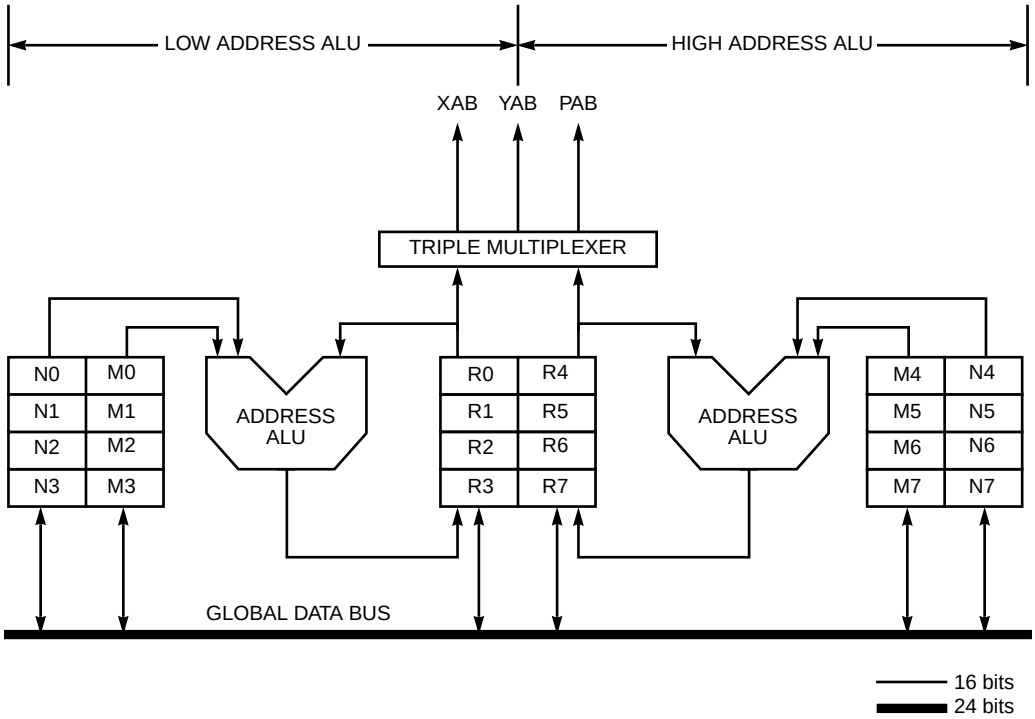


Figure 4-2 AGU Block Diagram

GDB. When read by the GDB, the contents of a register are placed in the two least significant bytes, and the most significant byte on the GDB is zero extended. When a register is written, only the least significant 16 bits of the GDB are used; the upper portion is truncated.

4.2.3 Modifier Register Files (Mn)

Each of the two modifier register files shown in Figure 4-2 consists of four 16-bit registers. The two files contain modifier registers M0 - M3 and M4 - M7, which specify the type of arithmetic used during address register update calculations or contain data. Each modifier register can be read or written by the GDB. When read by the GDB, the contents of a register are placed in the two least significant bytes, and the most significant byte on the GDB is zero extended. When a register is written, only the least significant 16 bits of the GDB are used; the upper portion is truncated. Each modifier register is preset to $FFFF during a processor reset.

4.2.4 Address ALU

The two address ALUs are identical (see Figure 4-2) in that each contains a 16-bit full adder (called an offset adder), which can add 1) plus one, 2) minus one, 3) the contents of the respective offset register N, or 4) the twos complement of N to the contents of the

selected address register. A second full adder (called a modulo adder) adds the summed result of the first full adder to a modulo value, M or minus M, where M-1 is stored in the respective modifier register. A third full adder (called a reverse-carry adder) can add 1) plus one, 2) minus one, 3) the offset N (stored in the respective offset register), or 4) minus N to the selected address register with the carry propagating in the reverse direction — i.e., from the most significant bit (MSB) to the least significant bit (LSB). The offset adder and the reverse-carry adder are in parallel and share common inputs. The only difference between them is that the carry propagates in opposite directions. Test logic determines which of the three summed results of the full adders is output.

Each address ALU can update one address register, Rn, from its respective address register file during one instruction cycle and can perform linear, reverse-carry, and modulo arithmetic. The contents of the selected modifier register specify the type of arithmetic to be used in an address register update calculation. The modifier value is decoded in the address ALU.

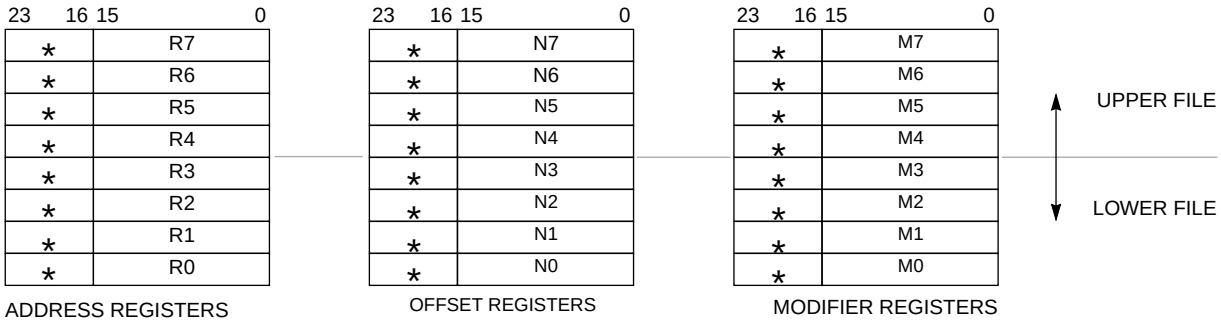
The output of the offset adder gives the result of linear arithmetic (e.g., Rn $\pm$ 1; Rn $\pm$ N) and is selected as the modulo arithmetic unit output for linear arithmetic addressing modifiers. The reverse-carry adder performs the required operation for reverse-carry arithmetic and its result is selected as the address ALU output for reverse-carry addressing modifiers. Reverse-carry arithmetic is useful for 2^k-point fast Fourier transform (FFT) addressing. For modulo arithmetic, the modulo arithmetic unit will perform the function (Rn $\pm$ N) modulo M, where N can be one, minus one, or the contents of the offset register Nn. If the modulo operation requires wraparound for modulo arithmetic, the summed output of the modulo adder gives the correct updated address register value; if wraparound is not necessary, the output of the offset adder gives the correct result.

4.2.5 Address Output Multiplexers

The address output multiplexers (see Figure 4-2) select the source for the XAB, YAB, and PAB. These multiplexers allow the XAB, YAB, or PAB outputs to originate from R0 - R3 or R4 - R7.

4.3 PROGRAMMING MODEL

The programmer's view of the AGU is eight sets of three registers (see Figure 4-3). These registers can act as temporary data registers and indirect memory pointers. Automatic updating is available when using address register indirect addressing. The Mn registers can be programmed for linear addressing, modulo addressing, and bit-reverse addressing.



* Written as don't care; read as zero

Figure 4-3 AGU Programming Model

4.3.1 Address Register Files (R0 - R3 and R4 - R7)

The eight 16-bit address registers, R0 - R7, can contain addresses or general-purpose data. The 16-bit address in a selected address register is used in the calculation of the effective address of an operand. When supporting parallel X and Y data memory moves, the address registers must be thought of as two separate files, R0 - R3 and R4 - R7. The contents of an Rn may point directly to data or may be offset. In addition, Rn can be pre-updated or post-updated according to the addressing mode selected. If an Rn is updated, modifier registers, Mn, are always used to specify the type of update arithmetic. Offset registers, Nn, are used for the update-by-offset addressing modes. The address register modification is performed by one of the two modulo arithmetic units. Most addressing modes modify the selected address register in a read-modify-write fashion; the address register is read, its contents are modified by the associated modulo arithmetic unit, and the register is written with the appropriate output of the modulo arithmetic unit. The form of address register modification performed by the modulo arithmetic unit is controlled by the contents of the offset and modifier registers discussed in the following paragraphs. Address registers are not affected by a processor reset.

4.3.2 Offset Register Files (N0 - N3 and N4 - N7)

The eight 16-bit offset registers, N0 - N7, can contain offset values used to increment/decrement address registers in address register update calculations or can be used for 16-bit general-purpose storage. For example, the contents of an offset register can be used to step through a table at some rate (e.g., five locations per step for waveform generation), or the contents can specify the offset into a table or the base of the table for indexed addressing. Each address register, Rn, has its own offset register, Nn, associated with it.

Table 4-1 Address Register Indirect Summary

Address Register Indirect	Uses Mn Modifier	Operand Reference									Assembler Syntax
		S	C	D	A	P	X	Y	L	XY	
No Update	No					X	X	X	X	X	(Rn)
Postincrement by 1	Yes					X	X	X	X	X	(Rn)+
Postdecrement by 1	Yes					X	X	X	X	X	(Rn)–
Postincrement by Offset Nn	Yes					X	X	X	X	X	(Rn)+Nn

NOTE:
S = System Stack Reference
C = Program Control Unit Register Reference
D = Data ALU Register Reference
A = Address ALU Register Reference
P = Program Memory Reference
X = X Memory Reference
Y = Y Memory Reference
L = L Memory Reference
XY = XY Memory Reference

Offset registers are not affected by a processor reset.

4.3.3 Modifier Register Files (M0 - M3 and M4 - M7)

The eight 16-bit modifier registers, M0 - M7, define the type of address arithmetic to be performed for addressing mode calculations, or they can be used for general-purpose storage. The address ALU supports linear, modulo, and reverse-carry arithmetic types for all address register indirect addressing modes. For modulo arithmetic, the contents of Mn also specify the modulus. Each address register, Rn, has its own modifier register, Mn, associated with it. Each modifier register is set to $FFFF on processor reset, which specifies linear arithmetic as the default type for address register update calculations.

4.4 ADDRESSING

The DSP56K provides three different addressing modes: register direct, address register indirect, and special. Since the register direct and special addressing modes do not necessarily use the AGU registers, they are described in SECTION 6 - INSTRUCTION SET INTRODUCTION. The address register indirect addressing modes use the registers in

the AGU and are described in the following paragraphs.

4.4.1 Address Register Indirect Modes

When an address register is used to point to a memory location, the addressing mode is called “address register indirect” (see Table 4-1). The term indirect is used because the register contents are not the operand itself, but rather the address of the operand. These addressing modes specify that an operand is in memory and specify the effective address of that operand.

A portion of the data bus movement field in the instruction specifies the memory space to be referenced. The contents of specific AGU registers that determine the effective address are modified by arithmetic operations performed in the AGU. The type of address arithmetic used is specified by the address modifier register, Mn. The offset register, Nn, is only used when the update specifies an offset.

Not all possible combinations are available, such as + (Rn). The 24-bit instruction word size is not large enough to allow a completely orthogonal instruction set for all instructions used by the DSP.

An example and description of each mode is given in the following paragraphs. SECTION 6 - INSTRUCTION SET INTRODUCTION and APPENDIX A - INSTRUCTION SET DETAILS give a complete description of the instruction syntax used in these examples. In particular, XY: memory references refer to instructions in which an operand in X memory and an operand in Y memory are referenced in the same instruction.

4.4.1.1 No Update

The address of the operand is in the address register, Rn (see Table 4-1). The contents of the Rn register are unchanged by executing the instruction. Figure 4-4 shows a MOVE instruction using address register indirect addressing with no update. This mode can be used for making XY: memory references. This mode does not use Nn or Mn registers.

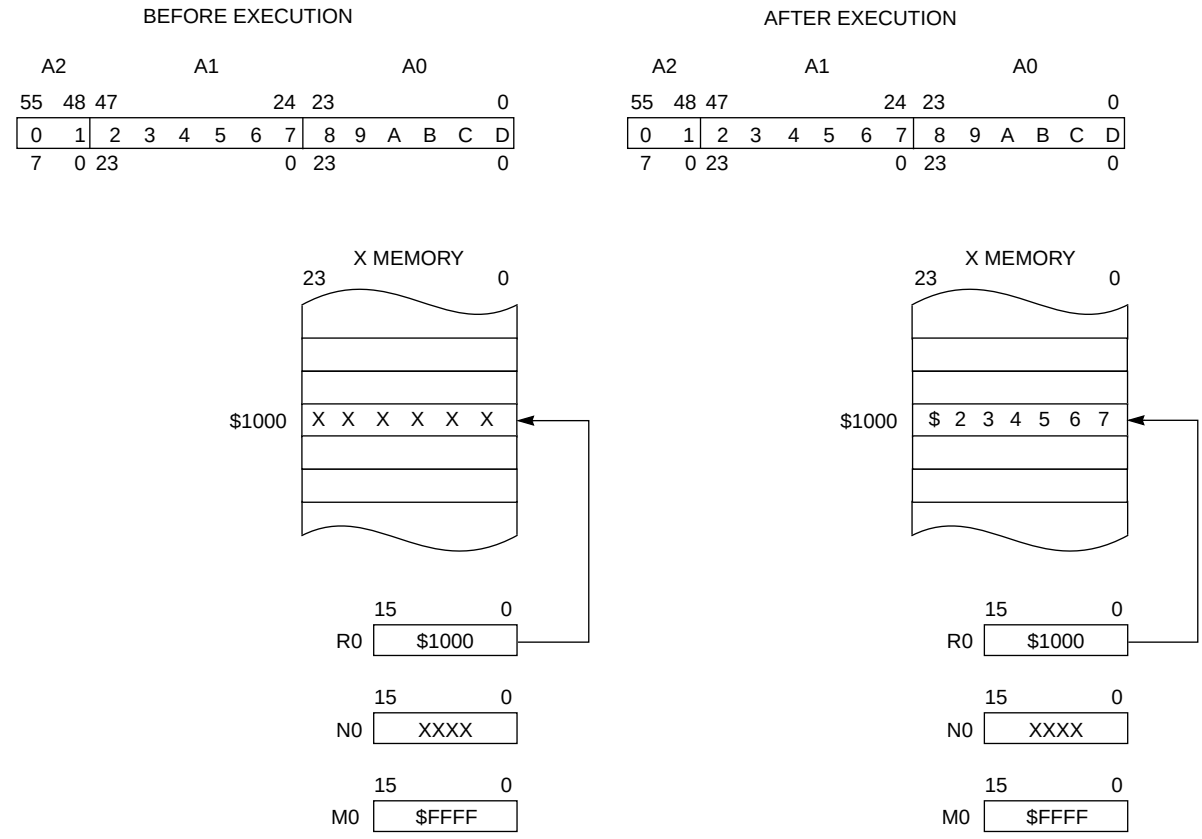
4.4.1.2 Postincrement By 1

The address of the operand is in the address register, Rn (see Table 4-1 and Figure 4-5). After the operand address is used, it is incremented by 1 and stored in the same address register. This mode can be used for making XY: memory references and for modifying the contents of Rn without an associated data move.

4.4.1.3 Postdecrement By 1

The address of the operand is in the address register, Rn (see Table 4-1 and Figure 4-6). After the operand address is used, it is decremented by 1 and stored in the same address register. This mode can be used for making XY: memory references and for

EXAMPLE: MOVE A1,X: (R0)



Assembler Syntax: (Rn)
Memory Spaces: P:, X:, Y:, XY:, L:
Additional Instruction Execution Time (Clocks): 0
Additional Effective Address Words: 0

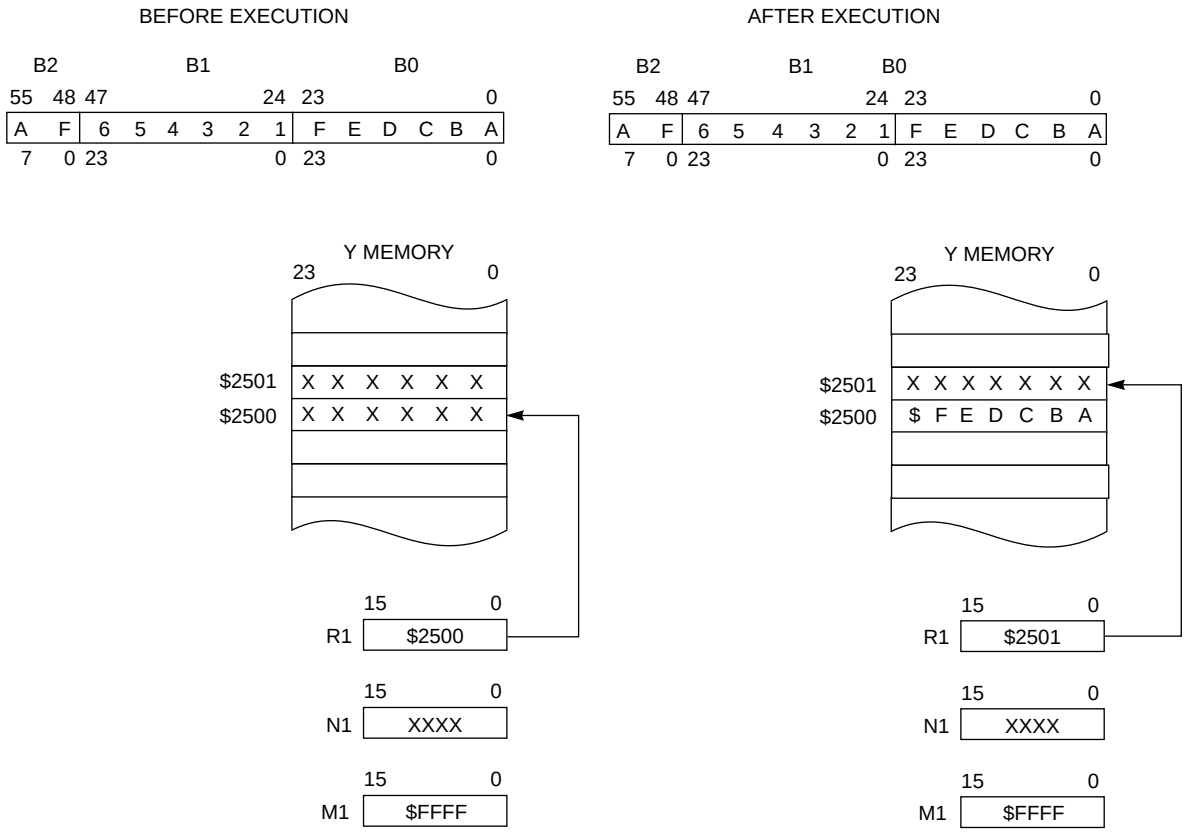
Figure 4-4 Address Register Indirect — No Update

modifying the contents of Rn without an associated data move.

4.4.1.4 Postincrement By Offset Nn

The address of the operand is in the address register, Rn (see Table 4-1 and Figure 4-7). After the operand address is used, it is incremented by the contents of the Nn register and stored in the same address register. The contents of the Nn register are unchanged. This mode can be used for making XY: memory references and for modifying the contents of

EXAMPLE: MOVE B0,Y: (R1)+



Assembler Syntax: (Rn)+
Memory Spaces: P:, X:, Y:, XY:, L:
Additional Instruction Execution Time (Clocks): 0
Additional Effective Address Words: 0

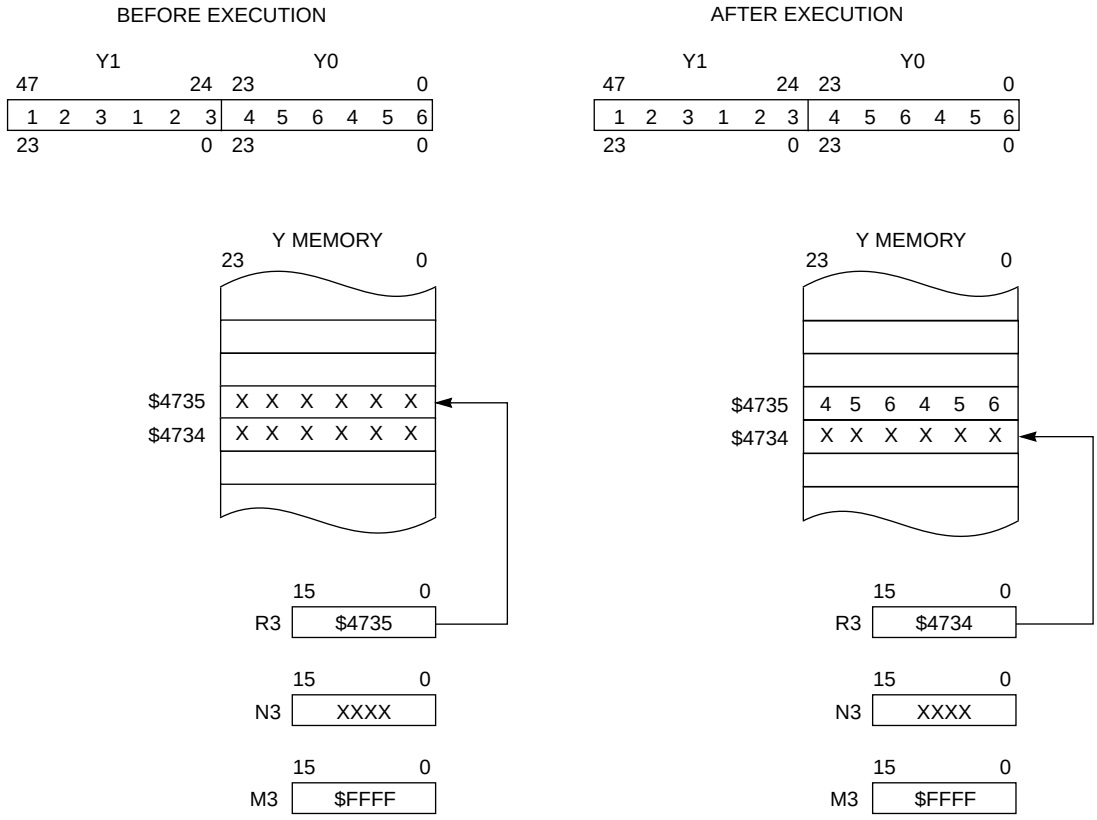
Figure 4-5 Address Register Indirect — Postincrement

Rn without an associated data move.

4.4.1.5 Postdecrement By Offset Nn

The address of the operand is in the address register, Rn (see Table 4-1 and Figure 4-8). After the operand address is used, it is decremented by the contents of the Nn register and stored in the same address register. The contents of the Nn register are unchanged. This mode cannot be used for making XY: memory references, but it can be used to mod-

EXAMPLE: MOVE Y0,Y: (R3)-



Assembler Syntax: (Rn)–
Memory Spaces: P:, X:, Y:, XY:, L:
Additional Instruction Execution Time (Clocks): 0
Additional Effective Address Words: 0

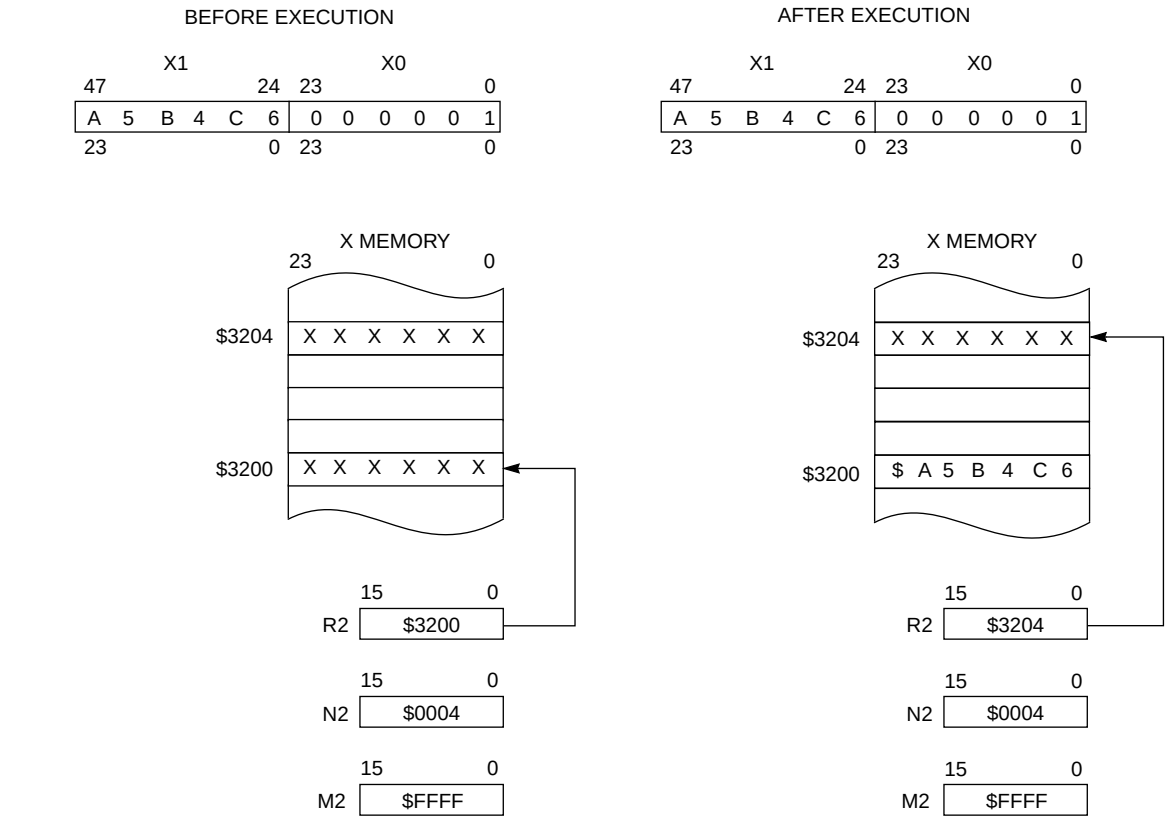
Figure 4-6 Address Register Indirect — Postdecrement

ify the contents of Rn without an associated data move.

4.4.1.6 Indexed By Offset Nn

The address of the operand is the sum of the contents of the address register, Rn, and the contents of the address offset register, Nn (see Table 4-1 and Figure 4-9). The contents of the Rn and Nn registers are unchanged. This addressing mode, which requires

EXAMPLE: MOVE X1,X: (R2)+N2



Assembler Syntax: (Rn)+Nn
Memory Spaces: P:, X:, Y:, XY:, L:
Additional Instruction Execution Time (Clocks): 0
Additional Effective Address Words: 0

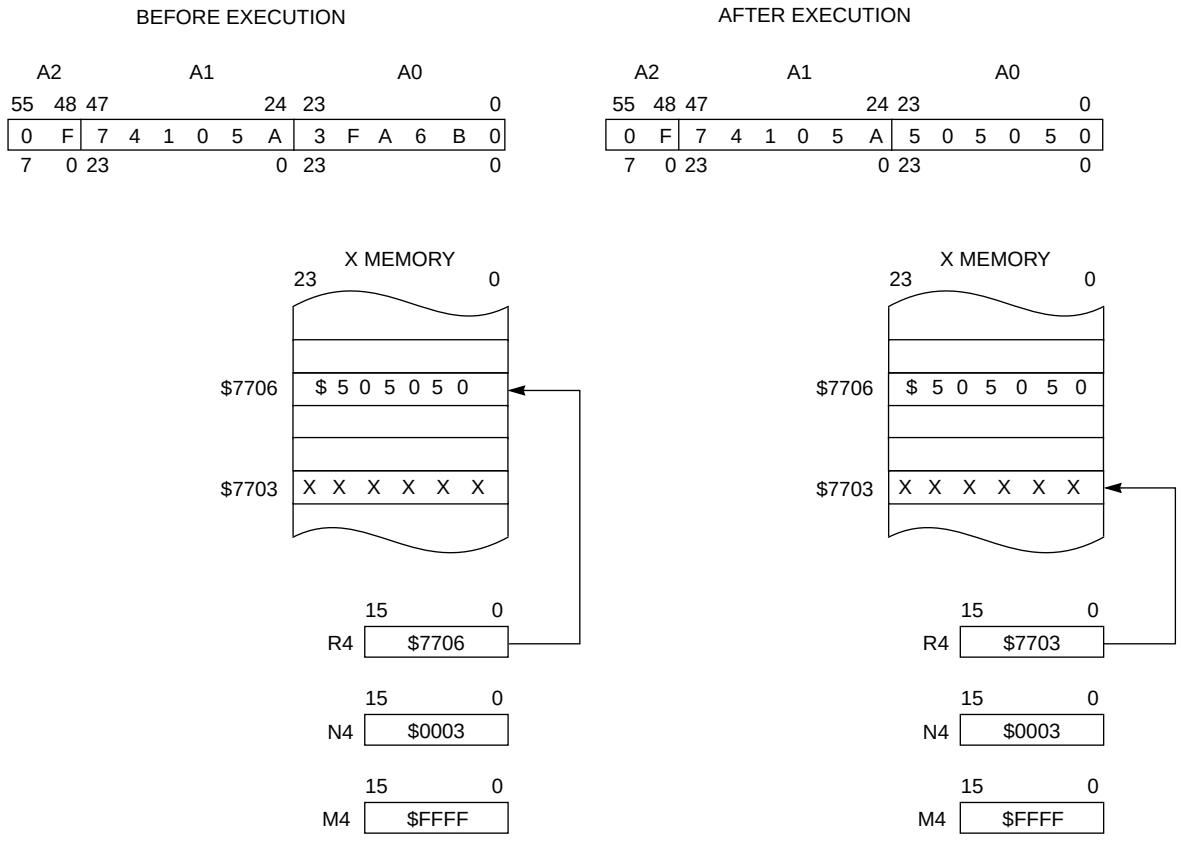
Figure 4-7 Address Register Indirect — Postincrement by Offset Nn

an extra instruction cycle, cannot be used for making XY: memory references.

4.4.1.7 Predecrement By 1

The address of the operand is the contents of the address register, Rn, decremented by 1 before the operand address is used (see Table 4-1 and Figure 4-10). The contents of Rn are decremented and stored in the same address register. This addressing mode requires an extra instruction cycle. This mode cannot be used for making XY: memory references, nor can it be used for modifying the contents of Rn without an associated data

EXAMPLE: MOVE X:(R4)–N4,A0



Assembler Syntax: (Rn)–Nn
Memory Spaces: P:, X:, Y:, L:
Additional Instruction Execution Time (Clocks): 0
Additional Effective Address Words: 0

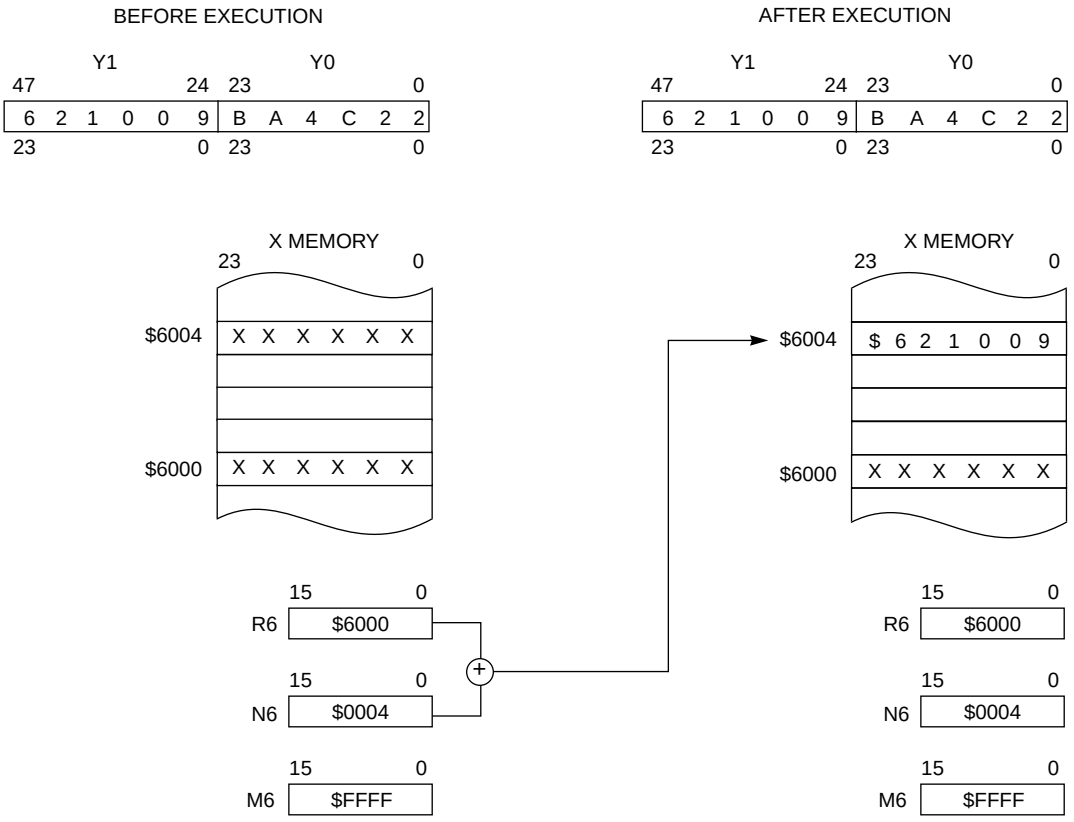
Figure 4-8 Address Register Indirect — Postdecrement by Offset Nn

move.

4.4.2 Address Modifier Arithmetic Types

The address ALU supports linear, modulo, and reverse-carry arithmetic for all address register indirect modes. These arithmetic types easily allow the creation of data structures in memory for FIFOs (queues), delay lines, circular buffers, stacks, and bit-reversed FFT buffers.

EXAMPLE: MOVE Y1,X: (R6+N6)

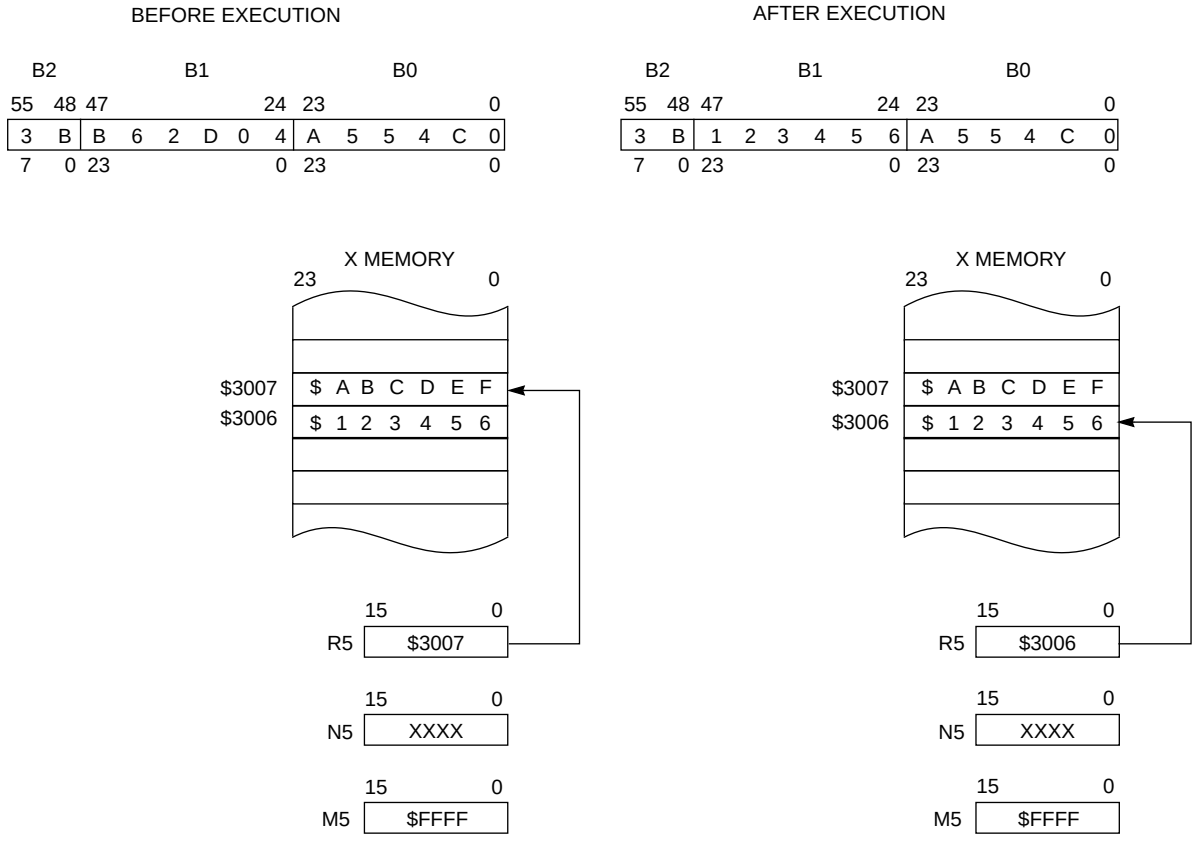


Assembler Syntax: (Rn+Nn)
Memory Spaces: P:, X:, Y:, L:
Additional Instruction Execution Time (Clocks): 2
Additional Effective Address Words: 0

Figure 4-9 Address Register Indirect — Indexed by Offset Nn

The contents of the address modifier register, Mn, defines the type of arithmetic to be performed for addressing mode calculations. For modulo arithmetic, the contents of Mn also specifies the modulus, or the size of the memory buffer whose addresses will be referenced. See Table 4-2 for a summary of the address modifiers implemented on the

EXAMPLE: MOVE X: –(R5),B1



Assembler Syntax: –Rn
Memory Spaces: P:, X:, Y:, L:
Additional Instruction Execution Time (Clocks): 2
Additional Effective Address Words: 0

Figure 4-10 Address Register Indirect — Predecrement

DSP56K. The MMMM column indicates the hex value which should be stored in the Mn register.

4.4.2.1 Linear Modifier (Mn=$FFFF)

When the value in the modifier register is $FFFF, address modification is performed using normal 16-bit linear arithmetic (see Table 4-2). A 16-bit offset, Nn, and + 1 or –1 can be used in the address calculations. The range of values can be considered as signed (Nn from –32,768 to + 32,767) or unsigned (Nn from 0 to + 65,535) since there is no arithmetic

difference between these two data representations. Addresses are normally considered unsigned, and data is normally considered signed.

4.4.2.2 Modulo Modifier

When the value in the modifier register falls into one of two ranges (Mn=$0001 to $7FFF or Mn= $8001 to $BFFF with the reserved gaps noted in the table), address modification is performed using modulo arithmetic (see Table 4-2).

Modulo arithmetic normally causes the address register value to remain within an address range of size M, whose lower boundary is determined by Rn. The upper boundary is determined by the modulus, or M. The modulus value, in turn, is determined by Mn, the value in the modifier register (see Figure 4-11).

There are certain cases where modulo arithmetic addressing conditions may cause the address register to jump linearly to the same relative address in a different buffer. Other cases firmly restrict the address register to the same buffer, causing the address register to wrap around within the buffer. The range in which the value contained in the modifier register falls determines how the processor will handle modulo addressing.

4.4.2.2.1 Mn=$0001 to $7FFF

In this range, the modulus (M) equals the value in the modifier register (Mn) plus 1. The memory buffer's lower boundary (base address) value, determined by Rn, must have zeros in the k LSBs, where $2^k \geq M$, and therefore must be a multiple of 2^k. The upper boundary is the lower boundary plus the modulo size minus one (base address plus M–1). Since $M \leq 2^k$, once M is chosen, a sequential series of memory blocks (each of length 2^k) is created where these circular buffers can be located. If $M < 2^k$, there will be a space between sequential circular buffers of (2^k)–M.

For example, to create a circular buffer of 21 stages, M is 21, and the lower address boundary must have its five LSBs equal to zero ($2^k \geq 21$, thus $k \geq 5$). The Mn register is loaded with the value 20. The lower boundary may be chosen as 0, 32, 64, 96, 128, 160, etc. The upper boundary of the buffer is then the lower boundary plus 21. There will be an unused space of 11 memory locations between the upper address and next usable lower address. The address pointer is not required to start at the lower address boundary or to end on the upper address boundary; it can initially point anywhere within the defined modulo address range. Neither the lower nor the upper boundary of the modulo region is stored; only the size of the modulo region is stored in Mn. The boundaries are determined by the contents of Rn. Assuming the (Rn)+ indirect addressing mode, if the address register pointer increments past the upper boundary of the buffer (base address plus M–1), it will wrap around through the base address (lower boundary). Alternatively, assuming the (Rn)- indirect addressing mode, if the address decrements past the lower boundary

Table 4-2 Address Modifier Summary

MMMM	Addressing Mode Arithmetic
0000	Reverse Carry (Bit Reverse)
0001	Modulo 2
0002	Modulo 3
:	:
7FFE	Modulo 32767
7FFF	Modulo 32768
8000	Reserved
8001	Multiple Wrap-Around Modulo 2
8002	Reserved
8003	Multiple Wrap-Around Modulo 4
:	Reserved
8007	Multiple Wrap-Around Modulo 8
:	Reserved
800F	Multiple Wrap-Around Modulo 2^4
:	Reserved
801F	Multiple Wrap-Around Modulo 2^5
:	Reserved
803F	Multiple Wrap-Around Modulo 2^6
:	Reserved
807F	Multiple Wrap-Around Modulo 2^7
:	Reserved
80FF	Multiple Wrap-Around Modulo 2^8
:	Reserved
81FF	Multiple Wrap-Around Modulo 2^9
:	Reserved
83FF	Multiple Wrap-Around Modulo 2^{10}
:	Reserved
87FF	Multiple Wrap-Around Modulo 2^{11}
:	Reserved
8FFF	Multiple Wrap-Around Modulo 2^{12}
:	Reserved
9FFF	Multiple Wrap-Around Modulo 2^{13}
:	Reserved
BFFF	Multiple Wrap-Around Modulo 2^{14}
:	Reserved
FFFF	Linear (Modulo 2^{15})

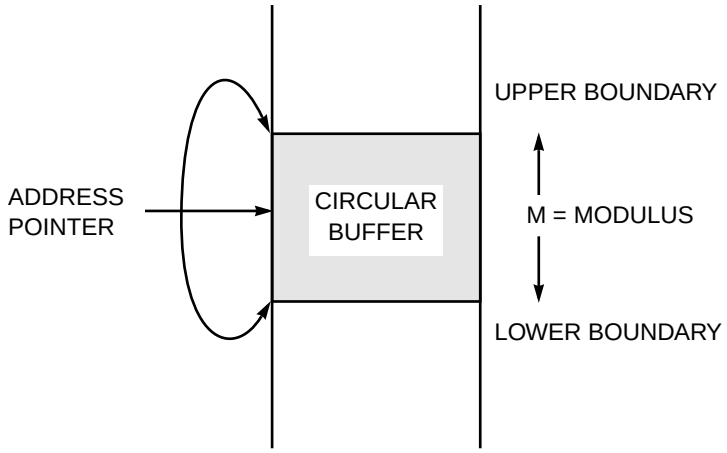


Figure 4-11 Circular Buffer

(base address), it will wrap around through the base address plus M–1 (upper boundary).

If an offset (Nn) is used in the address calculations, the 16-bit absolute value, |Nn|, must be less than or equal to M for proper modulo addressing in this range. If Nn>M, the result is data dependent and unpredictable, except for the special case where $Nn=P \times 2^k$, a multiple of the block size where P is a positive integer. For this special case, when using the (Rn)+ Nn addressing mode, the pointer, Rn, will jump linearly to the same relative address in a new buffer, which is P blocks forward in memory (see Figure 4-12).

Similarly, for (Rn)–Nn, the pointer will jump P blocks backward in memory. This technique is useful in sequentially processing multiple tables or N-dimensional arrays. The range of values for Nn is –32,768 to + 32,767. The modulo arithmetic unit will automatically wrap around the address pointer by the required amount. This type of address modification is useful for creating circular buffers for FIFOs (queues), delay lines, and sample buffers up to 32,768 words long as well as for decimation, interpolation, and waveform generation. The special case of (Rn) $\pm$ Nn mod M with $Nn=P \times 2^k$ is useful for performing the same algorithm on multiple blocks of data in memory — e.g., parallel infinite impulse response (IIR) filtering.

An example of address register indirect modulo addressing is shown in Figure 4-13. Starting at location 64, a circular buffer of 21 stages is created. The addresses generated are offset by 15 locations. The lower boundary = L x (2^k) where $2^k \geq 21$; therefore, k=5 and the lower address boundary must be a multiple of 32. The lower boundary may be chosen

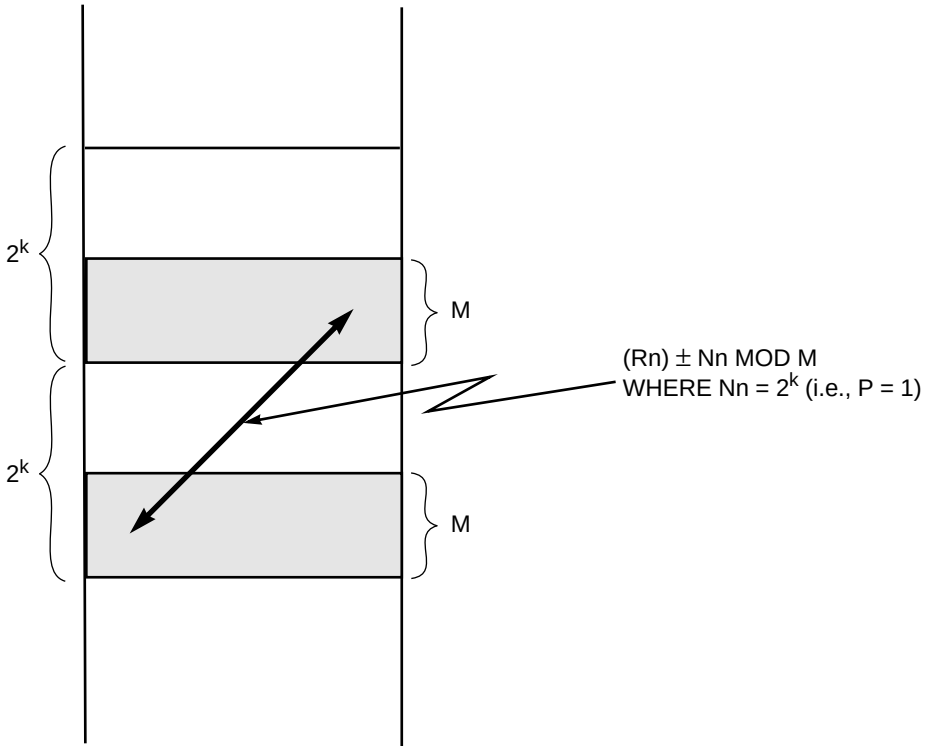


Figure 4-12 Linear Addressing with a Modulo Modifier

as 0, 32, 64, 96, 128, 160, etc. For this example, L is arbitrarily chosen to be 2, making the lower boundary 64. The upper boundary of the buffer is then 84 (the lower boundary plus 20 (M–1)). The Mn register is loaded with the value 20 (M–1). The offset register is arbitrarily chosen to be 15 (Nn≤M). The address pointer is not required to start at the lower address boundary and can begin anywhere within the defined modulo address range — i.e., within the lower boundary + (2^k) address region. The address pointer, Rn, is arbitrarily chosen to be 75 in this example. When R2 is post-incremented by the offset by the MOVE instruction, instead of pointing to 90 (as it would in the linear mode) it wraps around to 69. If the address register pointer increments past the upper boundary of the buffer (base address plus M–1), it will wrap around to the base address. If the address decrements past the lower boundary (base address), it will wrap around to the base address plus M–1.

If Rn is outside the valid modulo buffer range and an operation occurs that causes Rn to be updated, the contents of Rn will be updated according to modulo arithmetic rules. For example, a MOVE B0,X:(R0)+ N0 instruction (where R0=6, M0=5, and N0=0) would apparently leave R0 unchanged since N0=0. However, since R0 is above the upper boundary, the AGU calculates R0+ N0–M0–1 for the new contents of R0 and sets R0=0.

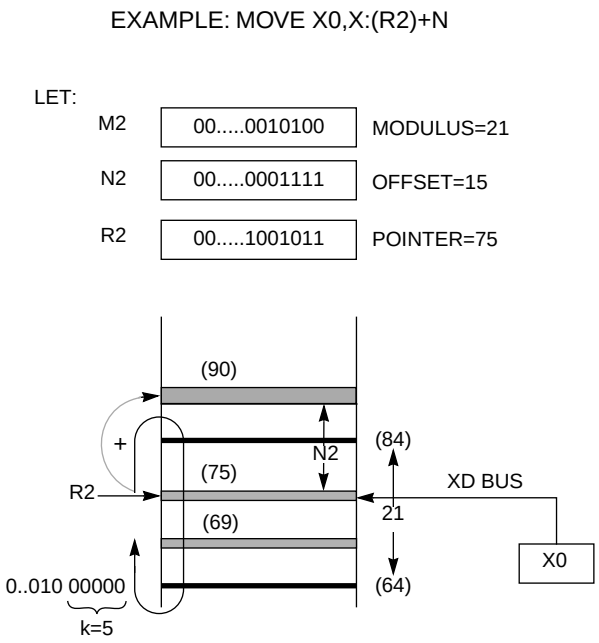


Figure 4-13 Modulo Modifier Example

The MOVE instruction in Figure 4-13 takes the contents of the X0 register and moves it to a location in the X memory pointed to by (R2), and then (R2) is updated modulo 21. The new value of R2 is not 90 (75+ 15), which would be the case if linear arithmetic had been used, but rather is 69 since modulo arithmetic was used.

4.4.2.2.2 Mn=$8001 to $BFFF

In this range, the modulo (M) equals (Mn+1)-$8000, where Mn is the value in the modifier register (see Table 4-2). This range firmly restricts the address register to the same buffer, causing the address register to wrap around within the buffer. This multiple wrap-around addressing feature reduces argument overhead and is useful for decimation, interpolation, and waveform generation.

The address modification is performed modulo M, where M may be **any power of 2** in the range from 2^1 to 2^{14}. Modulo M arithmetic causes the address register value to remain within an address range of size M defined by a lower and upper address boundary. The value M-1 is stored in the modifier register Mn least significant 14 bits while the two most significant bits are set to ‘10’. The lower boundary (base address) value must have zeroes in the k LSBs, where 2^k = M, and therefore must be a multiple of 2^k. The upper boundary is the lower boundary plus the modulo size minus one (base address plus M-1).

For example, to create a circular buffer of 32 stages, M is chosen as 32 and the lower address boundary must have its 5 least significant bits equal to zero (2^k = 32, thus k = 5). The Mn register is loaded with the value $801F. The lower boundary may be chosen as 0, 32, 64, 96, 128, 160, etc. The upper boundary of the buffer is then the lower boundary plus 31.

The address pointer is not required to start at the lower address boundary and may begin anywhere within the defined modulo address range (between the lower and upper boundaries). If the address register pointer increments past the upper boundary of the buffer (base address plus M-1) it will wrap around to the base address. If the address decrements past the lower boundary (base address) it will wrap around to the base address plus M-1. If an offset Nn is used in the address calculations, it is not required to be less than or equal to M for proper modulo addressing since multiple wrap around is supported for (Rn)+Nn, (Rn)-Nn and (Rn+Nn) address updates (multiple wrap-around cannot occur with (Rn)+, (Rn)- and -(Rn) addressing modes).

The multiple wrap-around address modifier is useful for decimation, interpolation and waveform generation since the multiple wrap-around capability may be used for argument reduction.

4.4.2.3 Reverse-Carry Modifier (Mn=$0000)

Reverse carry is selected by setting the modifier register to zero (see Table 4-2). The address modification is performed in hardware by propagating the carry in the reverse direction — i.e., from the MSB to the LSB. Reverse carry is equivalent to bit reversing the contents of Rn (i.e., redefining the MSB as the LSB, the next MSB as bit 1, etc.) and the offset value, Nn, adding normally, and then bit reversing the result. If the + Nn addressing mode is used with this address modifier and Nn contains the value $2^{(k-1)}$ (a power of two), this addressing modifier is equivalent to bit reversing the k LSBs of Rn, incrementing Rn by 1, and bit reversing the k LSBs of Rn again. This address modification is useful for addressing the twiddle factors in 2k-point FFT addressing and to unscramble 2^k-point FFT data. The range of values for Nn is 0 to + 32K (i.e., $Nn=2^{15}$), which allows bit-reverse addressing for FFTs up to 65,536 points.

To make bit-reverse addressing work correctly for a 2^k point FFT, the following procedures must be used:

1. Set Mn=0; this selects reverse-carry arithmetic.
2. Set $Nn=2^{(k-1)}$.

3. Set Rn between the lower boundary and upper boundary in the buffer memory. The lower boundary is L x (2^k), where L is an arbitrary whole number. This boundary gives a 16-bit binary number “xx . . . xx00 . . . 00”, where xx . . . xx=L and 00 . . . 00 equals k zeros. The upper boundary is L x (2^k)+ ((2^k)–1). This boundary gives a 16-bit binary number “xx . . . xx11 . . . 11”, where xx . . . xx=L and 11 . . . 11 equals k ones.
4. Use the (Rn)+ Nn addressing mode.

As an example, consider a 1024-point FFT with real data stored in the X memory and imaginary data stored in the Y memory. Since 1,024=2^{10}, k=10. The modifier register (Mn) is zero to select bit-reverse addressing. Offset register (Nn) contains the value 512 ($2^{(k-1)}$), and the pointer register (Rn) contains 3,072 (L x (2^k)=3 x (2^{10})), which is the lower boundary of the memory buffer that holds the results of the FFT. The upper boundary is 4,095 (lower boundary + (2^k)–1=3,072+ 1,023).

Postincrementing by + N generates the address sequence (0, 512, 256, 768, 128, 640,...), which is added to the lower boundary. This sequence (0, 512, etc.) is the scrambled FFT data order for sequential frequency points from 0 to 2π. Table 4-3 shows the successive contents of Rn when using (Rn)+ Nn updates.

Table 4-3 Bit-Reverse Addressing Sequence Example

Rn Contents	Offset From Lower Boundary
3072	0
3584	512
3328	256
3840	768
3200	128
3712	640

The reverse-carry modifier only works when the base address of the FFT data buffer is a multiple of 2^k, such as 1,024, 2,048, 3,072, etc. The use of addressing modes other than postincrement by + Nn is possible but may not provide a useful result.

The term bit reverse with respect to reverse-carry arithmetic is descriptive. The lower boundary that must be used for the bit-reverse address scheme to work is L x (2^k). In the previous example shown in Table 4-3, L=3 and k=10. The first address used is the lower boundary (3072); the calculation of the next address is shown in Figure 4-14. The k LSBs of the current contents of Rn (3,072) are swapped:

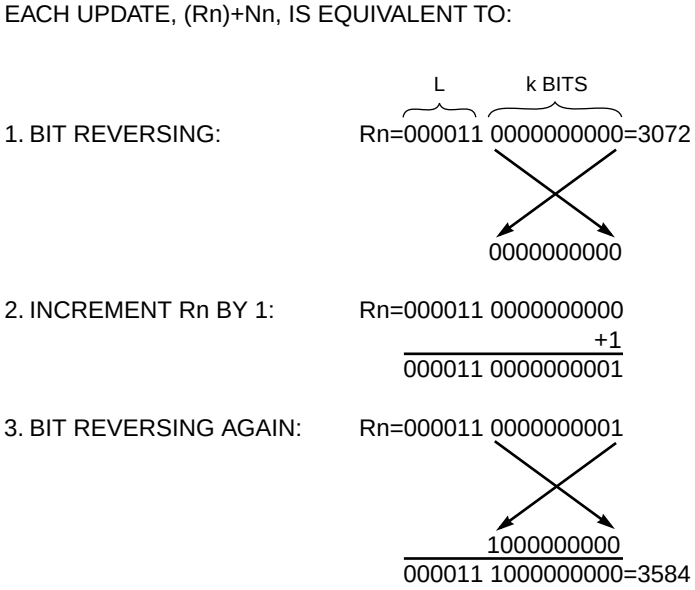


Figure 4-14 Bit-Reverse Address Calculation Example

- Bits 0 and 9 are swapped.
- Bits 1 and 8 are swapped.
- Bits 2 and 7 are swapped.
- Bits 3 and 6 are swapped.
- Bits 4 and 5 are swapped.

The result is incremented (3,073), and then the k LSBs are swapped again:

- Bits 0 and 9 are swapped.
- Bits 1 and 8 are swapped.
- Bits 2 and 7 are swapped.
- Bits 3 and 6 are swapped.
- Bits 4 and 5 are swapped.

The result is Rn equals 3,584.

4.4.2.4 Address-Modifier-Type Encoding Summary

There are three address modifier types:

- Linear Addressing
- Reverse-Carry Addressing
- Modulo Addressing

Bit-reverse addressing is useful for 2^k-point FFT addressing. Modulo addressing is useful for creating circular buffers for FIFOs (queues), delay lines, and sample buffers up to 32,768 words long. The linear addressing is useful for general-purpose addressing. There is a reserved set of modifier values (from 32,768 to 65,534) that should not be used.

Figure 4-15 gives examples of the three addressing modifiers using 8-bit registers for simplification (all AGU registers are 16 bit). The addressing mode used in the example, postincrement by offset Nn, adds the contents of the offset register to the contents of the address register after the address register is accessed. The results of the three examples are as follows:

- The linear address modifier addresses every fifth location since the offset register contains $5.
- Using the bit-reverse address modifier causes the postincrement by offset Nn addressing mode to use the address register, bit reverse the four LSBs, increment by 1, and bit reverse the four LSBs again.
- The modulo address modifier has a lower boundary at a predetermined location, and the modulo number plus the lower boundary establishes the upper boundary. This boundary creates a circular buffer so that, if the address register is pointing within the boundaries, addressing past a boundary causes a circular wraparound to the other boundary.

LINEAR ADDRESS MODIFIER

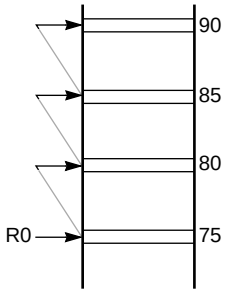
M0 = 255 = 11111111 FOR LINEAR ADDRESSING WITH R0

ORIGINAL REGISTERS: N0 = 5, R0 = 75 = 0100 1011

POSTINCREMENT BY OFFSET N0: R0 = 80 = 0101 0000

POSTINCREMENT BY OFFSET N0: R0 = 85 = 0101 0101

POSTINCREMENT BY OFFSET N0: R0 = 90 = 0101 1010



MODULO ADDRESS MODIFIER

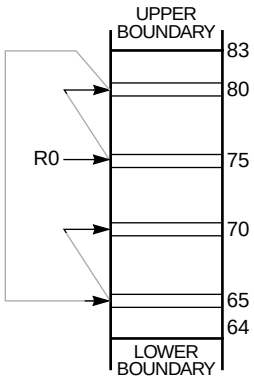
M0 = 19 = 0001 0011 FOR MODULO 20 ADDRESSING WITH R0

ORIGINAL REGISTERS: N0 = 5, R0 = 75 = 0100 1011

POSTINCREMENT BY OFFSET N0: R0 = 80 = 0101 0000

POSTINCREMENT BY OFFSET N0: R0 = 65 = 0100 0001

POSTINCREMENT BY OFFSET N0: R0 = 70 = 0100 0110



REVERSE-CARRY ADDRESS MODIFIER

M0 = 0= 0000 0000 FOR REVERSE-CARRY ADDRESSING WITH R0

ORIGINAL REGISTERS: N0 = 8, R0 = 64 = 0100 0000

POSTINCREMENT BY OFFSET N0: R0 = 72 = 0100 1000

POSTINCREMENT BY OFFSET N0: R0 = 68 = 0100 0100

POSTINCREMENT BY OFFSET N0: R0 = 76 = 0100 1100

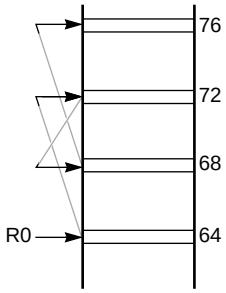


Figure 4-15 Address Modifier Summary