Chapter 4
Address Generation Unit

The Address Generation Unit (AGU) is one of three execution units on the DSP56300 core. The AGU performs the effective address calculations (using integer arithmetic) necessary to address data operands in memory and contains the registers used to generate the addresses. To minimize address-generation overhead, the AGU operates in parallel with other chip resources. It implements four types of arithmetic:

- Linear
- Modulo
- Multiple wrap-around modulo
- Reverse-carry

4.1 AGU Architecture

The AGU is divided into halves, each with its own Address Arithmetic Logic Unit (Address ALU). Each Address ALU has four sets of register triplets, and each register triplet is composed of an address register, an offset register, and a modifier register. The two Address ALUs are identical. Each contains a 24-bit full adder—an offset adder—which can perform the following additions/subtractions on an address register:

- Plus one
- Minus one
- Plus the contents of the respective offset register N
- Minus the contents of the respective offset register N

A second full adder—a modulo adder—adds the summed result of the first full adder to a modulo value, M or minus M, where M is stored in the respective modifier register. A third full adder—a reverse-carry adder—can perform the following additions, with the carry propagating in the reverse direction (that is, from the Most Significant Bit (MSB) to the Least Significant Bit (LSB):

- Plus one
- Minus one

- The offset N (stored in the respective offset register)
- Minus N to the selected address register

The offset adder and the reverse-carry adder operate 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. **Figure 4-1** shows a block diagram of the AGU.

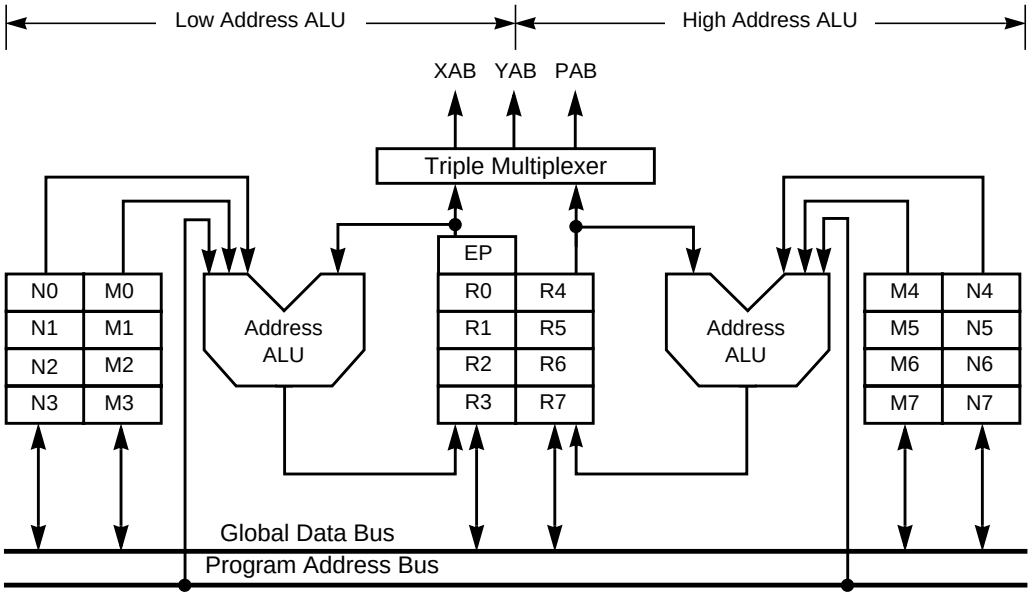

Figure 4-1 AGU Block Diagram

Each Address ALU can update one address register from its respective address register file during one instruction cycle. The contents of the associated modifier register specify the type of arithmetic to be used in the address register update calculation. The modifier value is decoded in the Address ALU.

The two Address ALUs can generate up to two addresses every instruction cycle:

- One for the PAB, or
- One for the XAB, or
- One for the YAB, or
- One for the XAB and one for the YAB

The AGU can directly address 16,777,216 locations on each of the XAB, YAB, and PAB. Using a register triplet to address each operand, the two independent ALUs can work with the two data memories to feed two operands to the Data ALU in a single cycle.

The registers are:

- Address Registers R0 – R3 on the Low Address ALU and R4 – R7 on the High Address ALU
- Offset Registers N0 – N3 on the Low Address ALU and N4 – N7 on the High Address ALU
- Modifier Registers M0 – M3 on the Low Address ALU and M4 – M7 on the High Address ALU

These registers are referred to as Rn for any address register, Nn for any offset register, and Mn for any modifier register. The Rn, Nn, and Mn registers are register triplets—that is, the offset and modulo registers of one triplet can be used only with an address register that belongs to the same triplet. For example, only N2 and M2 can be used only with R2. The eight triplets are as follows:

- Low Address ALU register triplets
  - — R0:N0:M0
  - — R1:N1:M1
  - — R2:N2:M2
  - — R3:N3:M3
- High Address ALU register triplets
  - — R4:N4:M4
  - — R5:N5:M5
  - — R6:N6:M6
  - — R7:N7:M7

The Global Data Bus (GDB) can read from or write to each register. The address output multiplexers select the address for the XAB, YAB, and PAB, where the address originates from the R0 — R3 or R4 — R7 registers.

4.2 Sixteen-bit Compatibility Mode

When the Sixteen-bit Compatibility (SC) mode bit is set in the Status Register (SR)[1], AGU operations are modified in the following ways:

1. For details on the Status Register (SR), see **Section 5.4.1.2**, "Status Register (SR)."

- MOVE operations to/from any of the AGU registers (R0 – R7, N0 – N7 and M0 – M7) clear the eight MSBs of the destination.
- The eight MSBs of any AGU address calculation result are cleared.
- The sign bit of the selected N register is Bit 15 instead of Bit 23.
- The eight MSBs of the address are ignored in the calculations of memory regions.

In Sixteen-bit Compatibility (SC) mode, proper memory access is not guaranteed for an address register in which the eight MSBs are not all zeros. If SC mode is invoked dynamically, take care to ensure that the eight MSBs of an address register used to access memory are cleared, since the switch to SC mode does not automatically clear these bits. Due to pipelining, a change in the SC bit takes effect only after three additional instruction cycles. Therefore, to ensure proper operation, insert three NOP instructions after the instruction that sets the SC bit.

4.3 Programming Model

The programmer views the AGU as eight sets of three registers, as shown in **Figure 4-2**. These registers can be used as temporary data registers and indirect memory pointers. Automatic updating is available when address register indirect addressing is in use. The address registers can be programmed for linear addressing, modulo addressing (regular or multiple wrap-around), and bit-reverse addressing.

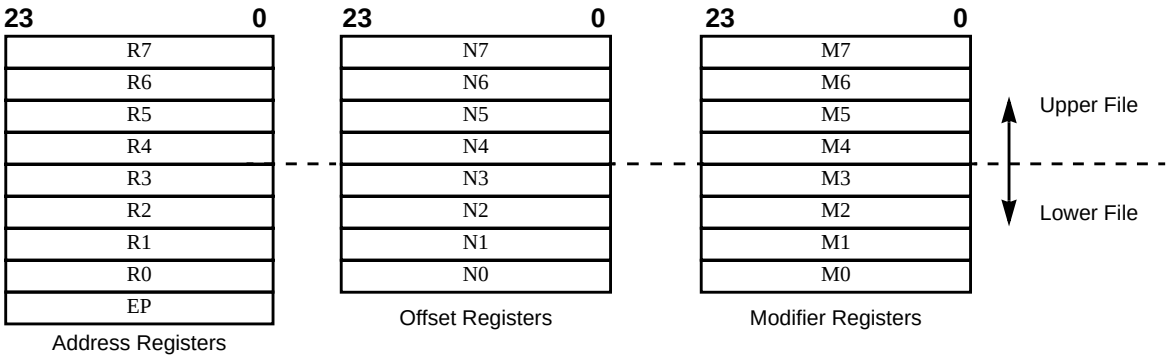

Figure 4-2 AGU Programming Model

4.3.1 Address Register Files

The eight 24-bit address registers R0 – R7 can contain addresses or general-purpose data. The 24-bit address in a selected address register is used in calculating the effective address of an operand. During parallel X and Y data memory moves, the address registers must be programmed as two separate files, R0 – R3 and R4 – R7. The contents of an address register can point directly to data, or they can be offset.

In addition, an address register (Rn) can be pre-updated or post-updated according to the addressing mode selected. If an address register (Rn) is updated, the corresponding modifier register (Mn) specifies 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, the associated modulo arithmetic unit modifies its contents, and the register is written with the appropriate output of the modulo arithmetic unit. The contents of the offset and modifier registers control the form of address register modification performed by the modulo arithmetic unit. These registers are discussed in **Section 4.3.3.** and **Section 4.3.4.**

4.3.2 Stack Extension Pointer

The hardware stack is an area in internal memory that provides temporary storage during program execution. The stack exists in either the X data memory or the Y data memory, as selected by the XYS bit in the Operating Mode Register (OMR) (refer to **Chapter 5,** *Program Control Unit* for a detailed description of the OMR). The stack uses push operations to add data to the stack and pull operations to retrieve data from the stack.

The contents of the 24-bit stack Extension Pointer (EP) register point to the stack extension whenever the stack extension is enabled and move operations to or from the on-chip hardware stack are needed. The EP register points to the next available location to which a push can be made (that is, it points just past the last item on the stack). The EP register is a read/write register and is referenced implicitly (for example, by the DO, JSR, or RTI instructions) or directly (for example, by the MOVEC instruction). The EP register is not initialized during hardware reset, and must be set (using a MOVEC instruction) prior to enabling the stack extension. For more information on the operation of the stack extension, see **Chapter 5,** *Program Control Unit.*

4.3.3 Offset Register Files

The eight 24-bit offset registers, N[0 – 7], contain offset values to increment or decrement address registers in address register update calculations. For example, the contents of an offset register are used to step through a table at some rate (for example, 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 has its own associated offset register. Each offset register can also be used for 24-bit general-purpose storage if it is not required as an address register offset.

4.3.4 Modifier Register Files

The eight 24-bit modifier registers, M0–M7, define the type of address arithmetic performed for addressing mode calculations. 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 has its own associated modifier register. Each modifier register is set to $FFFFFF on processor reset, which specifies linear arithmetic as the default type for address register update calculations. Each modifier register can also be used for 24-bit general purpose storage if it is not required as an address register modifier.

4.4 Addressing Modes

As listed in **Table 4-5**, the DSP56300 family core provides four different addressing modes:

- Register Direct
- Address Register Indirect
- PC-relative
- Special

Table 4-5 Addressing Modes Summary

Addressing Modes	Uses Mn Modifier	Operand Reference									Assembler Syntax
		S	C	D	A	P	X	Y	L	XY	
Register Direct											
Data or Control Register	No		√	√							
Address Register Rn	No				√						
Address Modifier Register Mn	No				√						
Address Offset Register Nn	No				√						
Address Register Indirect											
No Update	No					√	√	√	√	√	(Rn)
Post-increment by 1	Yes					√	√	√	√	√	(Rn) +
Post-decrement by 1	Yes					√	√	√	√	√	(Rn) –
Post-increment by Offset Nn	Yes					√	√	√	√	√	(Rn) + Nn
Post-decrement by Offset Nn	Yes					√	√	√	√		(Rn) – Nn
Indexed by Offset Nn	Yes					√	√	√	√		(Rn + Nn)
Pre-decrement by 1	Yes					√	√	√	√		– (Rn)
Short/Long Displacement	Yes						√	√	√		(Rn + displ)
PC-relative											
Short/Long Displacement PC-relative	No					√					(PC + displ)
Address Register	No					√					(PC + Rn)

Table 4-5 Addressing Modes Summary (Continued)

Addressing Modes	Uses Mn Modifier	Operand Reference									Assembler Syntax
		S	C	D	A	P	X	Y	L	XY	
Special											
Short/Long Immediate Data	No					√					
Absolute Address	No					√	√	√	√		
Absolute Short Address	No						√	√	√		
Short Jump Address	No					√					
I/O Short Address	No						√	√			
Implicit	No	√	√			√					

Note: Use this key to the Operand Reference columns:

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

4.4.1 Register Direct Modes

The Register Direct addressing modes specify that the operand is in one or more of the ten Data ALU registers, 24 address registers, or seven control registers.

- Data or Control Register Direct: The operand is in one, two, or three Data ALU register(s), as specified in a portion of the data bus movement field in the instruction. This addressing mode also specifies a control register operand for special instructions. This reference is classified as a register reference.
- Address Register Direct: The operand is in one of the 24 address registers specified by an effective address in the instruction. This reference is classified as a register reference.

4.4.2 Address Register Indirect Modes

The Address Register Indirect modes specify that the address register points to a memory location. The term “indirect” signifies that the register contents are not the operand itself, but rather the operand address. These addressing modes specify that an operand is in memory and give the effective address of that operand. In several of the following calculations, the type of arithmetic used to calculate the address is determined by the Mn register.

- No Update (Rn)—The operand address is in the address register. The contents of the address register are unchanged by executing the instruction.

  ```
  Example: MOVE x:(Rn),x0
  ```

- Post-Increment By One (Rn) + —The operand address is in the address register. After the operand address is used, it is incremented by one and stored in the same address register. The Nn register is ignored.

  ```
  Example: MOVE x:(Rn)+,x0
  ```

- Post-Decrement By One (Rn) – —The operand address is in the address register. After the operand address is used, it is decremented by one and stored in the same address register. The Nn register is ignored.

  ```
  Example: MOVE x:(Rn)-,x0
  ```

- Post-Increment By Offset Nn (Rn) + Nn—The operand address is in the address register. 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.

  ```
  Example: MOVE x:(Rn)+Nn,x0
  ```

- Post-Decrement By Offset Nn (Rn) – Nn—The operand address is in the address register. 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.

  ```
  Example: MOVE x:(Rn)-Nn,x0
  ```

- Indexed By Offset Nn (Rn + Nn)—The operand address is the sum of the contents of the address register and the contents of the address offset register, Nn. The contents of the Rn and Nn registers are unchanged.

  ```
  Example: MOVE x:(Rn+Nn),x0
  ```

- Pre-Decrement By One -(Rn)—The operand address is the contents of the address register decremented by one. The contents of Rn are decremented by one and stored in the same address register before the memory access. The Nn register is ignored.

  ```
  Example: MOVE x:-(Rn),x0
  ```

- Short Displacement (Rn + Short Displacement)—The operand address is the sum of the contents of the address register Rn and a short signed displacement occupying seven bits in the instruction word. The displacement is first sign-extended to 24 bits (16 bits in SC mode) and then added to Rn to obtain the operand address. The contents of the Rn register are unchanged. The Nn register is ignored. This reference is classified as a memory reference.

  ```
  Example: MOVE x:(Rn+63),x0
  ```

- Long Displacement (Rn + Long Displacement)—This addressing mode requires one word (label) of instruction extension. The operand address is the sum of the

contents of the address register and the extension word. The contents of the address register are unchanged. The Nn register is ignored. This reference is classified as a memory reference.

```
Example: MOVE x:(Rn+64),x0
```

4.4.3 PC-relative Modes

In the PC-relative addressing modes, the operand address is obtained by adding a displacement, represented in twos-complement format, to the value of the Program Counter (PC). The PC points to the address of the instruction opcode word. The Nn and Mn registers are ignored, and the arithmetic used is always linear.

- Short Displacement PC-relative—The short displacement occupies nine bits in the instruction operation word. The displacement is first sign-extended to 24 bits and then added to the PC to obtain the operand address.
- Long Displacement PC-relative—This addressing mode requires one word of instruction extension. The operand address is the sum of the contents of the PC and the extension word.
- Address Register PC-relative—The operand address is the sum of the contents of the PC and the address register. The Mn and Nn registers are ignored. The contents of the address register are unchanged.

4.4.4 Special Address Modes

The special address modes do not use an address register in specifying an effective address. These modes either specify the operand or the operand address in a field of the instruction, or they implicitly reference an operand.

- Immediate Data—This addressing mode requires one word of instruction extension. The immediate data is a word operand in the extension word of the instruction. This reference is classified as a program reference.
- Immediate Short Data—The 8-bit or 12-bit operand is part of the instruction operation word. An 8-bit operand is used for an immediate move to register, ANDI, and ORI instructions. It is zero-extended. A 12-bit operand is used for DO and REP instructions. It is also zero-extended. This reference is classified as a program reference.
- Absolute Address—This addressing mode requires one word of instruction extension. The operand address is in the extension word. This reference is classified as a memory reference and a program reference.

- Absolute Short Address—The operand address occupies six bits in the instruction operation word, and it is zero-extended. This reference is classified as a memory reference.
- Short Jump Address—The operand occupies 12 bits in the instruction operation word. The address is zero-extended to 24 bits. This reference is classified as a program reference.
- I/O Short Address—The operand address occupies 6 bits in the instruction operation word, and it is one-extended. The I/O short addressing mode is used with the bit manipulation and move peripheral data instructions.
- Implicit Reference—Some instructions make implicit reference to the Program Counter (PC), System Stack (SSH, SSL), Loop Address Register (LA), Loop Counter (LC), or Status Register (SR). These registers are implied by the instruction, and their use is defined by the individual instruction descriptions. See **Chapter 12,** *Guide to the Instruction Set* for more information.

4.5 Address Modifier Types

The DSP56300 family core Address ALU supports linear, reverse-carry, modulo, and multiple wrap-around modulo arithmetic types for all address register indirect modes. These arithmetic types easily allow the creation of data structures in memory for First-In, First-Out (FIFO) queues, delay lines, circular buffers, stacks, and bit-reversed Fast Fourier Transform (FFT) buffers. Data is manipulated by updating address registers (pointers) rather than moving large blocks of data. The contents of the address modifier register define the type of arithmetic to be performed for addressing mode calculations. For modulo arithmetic, the address modifier register also specifies the modulus. Each address register has its own associated modifier register. All address register indirect modes can be used with any address modifier type. The following address modifier types are available:

- Linear addressing—Useful for general-purpose addressing
- Reverse-carry addressing—Useful for 2^k-point FFT addressing
- Modulo addressing—Useful for creating circular buffers for FIFO queues, delay lines and sample buffers
- Multiple wrap-around modulo addressing—Useful for decimation, interpolation, and waveform generation, since the multiple wrap-around capability can be used for argument reduction

Table 4-6 lists the address modifier types.

Table 4-6 Address Modifier Type Encoding Summary

Modifier Mn	Address Calculation Arithmetic
$XX0000	Reverse-Carry (Bit-Reverse)
$XX0001	Modulo 2
$XX0002	Modulo 3
:	:
$XX7FFE	Modulo 32767 (2^{15}-1)
$XX7FFF	Modulo 32768 (2^{15})
$XX8001	Multiple Wrap-Around Modulo 2
$XX8003	Multiple Wrap-Around Modulo 4
$XX8007	Multiple Wrap-Around Modulo 8
:	:
$XX9FFF	Multiple Wrap-Around Modulo 2^{13}
$XXBFFF	Multiple Wrap-Around Modulo 2^{14}
$XXFFFF	Linear (Modulo 2^{24})
Notes: 1. All other combinations are reserved. 2. XX can be any value.	

4.5.1 Linear Modifier (Mn = $XXFFFF)

Address modification is performed using normal 24-bit linear (modulo 16,777,216) arithmetic. A 24-bit offset, Nn, and ±1 can be used in the address calculations. The range of values can be considered as signed (Nn from –8,388,608 to +8,388,607) or unsigned (Nn from 0 to +16,777,216), since there is no arithmetic difference between these two data representations.

4.5.2 Reverse-Carry Modifier (Mn = $000000)

Reverse carry is selected by setting the modifier register to zero. Address modification is performed in hardware by propagating the carry in the reverse direction (that is, from the MSB to the LSB). Reverse carry is equivalent to bit reversing the contents of Rn (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 a value $2^{(k-1)}$ (a power of two), this addressing modifier is equivalent to bit reversing the k LSBs of Rn, incrementing Rn by one, and bit reversing the k LSBs of Rn again. This address modification is useful for addressing the two middle factors in 2^k-point FFT addressing and unscrambling 2^k-point FFT data. The range of values for Nn is 0 to + 8 M (that is, Nn = 2^{23}), which allows bit-reverse addressing for FFTs up to 16,777,216 points.

4.5.3 Modulo Modifier (Mn = Modulus – 1)

Address modification is performed using modulo M, where M ranges from 2 to +32,768. 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 = M – 1 is stored in the modifier register. The lower boundary (base address) value 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 + 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 is a space between sequential circular buffers of $(2^k) – M$.

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 Address Register Indirect with post-increment addressing mode, (Rn)+, if the address register pointer increments past the upper boundary of the buffer (base address + M – 1), it wraps around through the base address (lower boundary). Alternatively, assuming the Address Register Indirect with post-decrement addressing mode, (Rn)-, if the address decrements past the lower boundary (base address), it wraps around through the base address + M – 1 (upper boundary).

If an offset, Nn, is used in the address calculations, the 24-bit absolute value, |Nn|, must be less than or equal to M for proper modulo addressing. 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, jumps linearly to the same relative address in a new buffer, which is P blocks forward in memory. Similarly, for (Rn) – Nn, the pointer jumps 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 –8,388,608 to +8,388,607. The modulo arithmetic unit automatically wraps around the address pointer by the required amount. This type of address modification is useful for creating circular buffers for FIFO queues, delay lines, and sample buffers up to 8,388,607 words long, and for decimation, interpolation, and waveform generation. The special case of (Rn) ± Nn modulo M with $Nn = P \times 2^k$ is useful for performing the same algorithm on multiple blocks of data in memory, for example, when performing parallel Infinite Impulse Response (IIR) filtering.

4.5.4 Multiple Wrap-Around Modulo Modifier

The Multiple Wrap-Around Addressing mode is selected by setting bit 15 of the Mn register to one and clearing bit 14 to zero, as shown in **Table 4-6**. The address modification is performed using modulo M, where M is a 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 Mn register's 14 Least Significant Bits (bits 13–0), while bit 15 is set to one and bit 14 is cleared to zero. The lower boundary (base address) value must have zeros 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 + M – 1).

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 + M – 1), it wraps around to the base address. If the address decrements past the lower boundary (base address), it wraps around to the base address + 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.