

Chapter 4



Operand Addressing Modes

Operand addressing modes

- ❑ Ways to specify operands for an instruction:
 - Inherent: operand location is implied by the instruction mnemonic
 - Immediate: operand immediately follows instruction code in memory
 - Memory: operand must be read from or written to a designated memory address
 - ❑ Direct – specify the memory address directly
 - ❑ Indirect – specify a way to determine the address

Inherent (Register) Addressing

- No “operands” listed with the instruction
 - No memory is accessed
- Operand is in a default register.

clra ; 0 → a
aba ; a + b → a
decb ; b - 1 → b
inx ; x + 1 → x

EXAMPLE 2-9

0000 1806	1	aba		; A + B -> A
0002 08	2	inx		; X+1 -> X
0003 B781	3	exg	a,b	; A <-> B

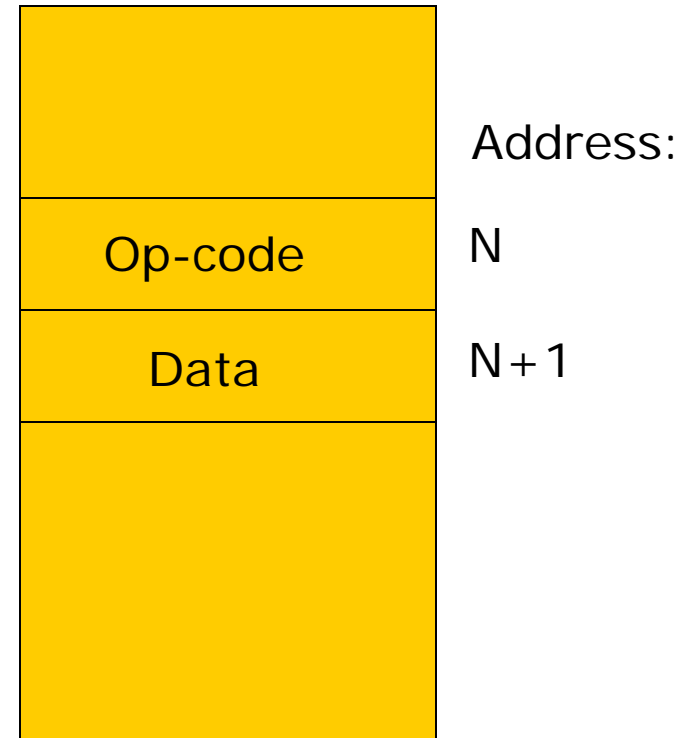
Immediate addressing.

Figure 4-4

Data immediately follows op-code in memory.

If 16-bit data, then the op-code is followed by first the high byte, and then the low byte of the data.

“big endian” format



Example 4-2 Immediate Addressing

0000 8640
0002 8664
0004 CE1234

```
1   ldaa    #64      ; Decimal 64 -> A
2   ldaa    #$64     ; Hexadecimal 64 -> A
3   ldx     #$1234   ; Hexadecimal 1234 -> X
```

“Source” assembly language statements.

Program
addresses

Instruction
codes

Created by the assembler

Note:

CASM12 designates decimal by !
(example: !64)

Code Warrior and most other
assemblers assume decimal by
default (no special designator).

Immediate addressing with labels

```
Value1    equ    $34
           org    $0800
Bob:      dc.1   $56
           .....
           ldaa  #Value1    ; $34 → a
           ldx   #Bob       ; $0800 → x
           ldaa  Bob        ; $56 → a (Extended)
           ldaa  0,x        ; $56 → a (Indexed)
```

Label "Bob" is a symbolic reference to address \$0800.
After "ldx", register IX also contains address \$0800.

Direct/Extended Addressing

- The instruction “directly” specifies the address of the operand
- The address of the operand follows the op-code in memory
 - “Extended” => full 16-bit address
 - “Direct” => address range \$0000-\$00FF.
Instruction includes only low byte of address

Examples

ldaa \$1000 ;extended

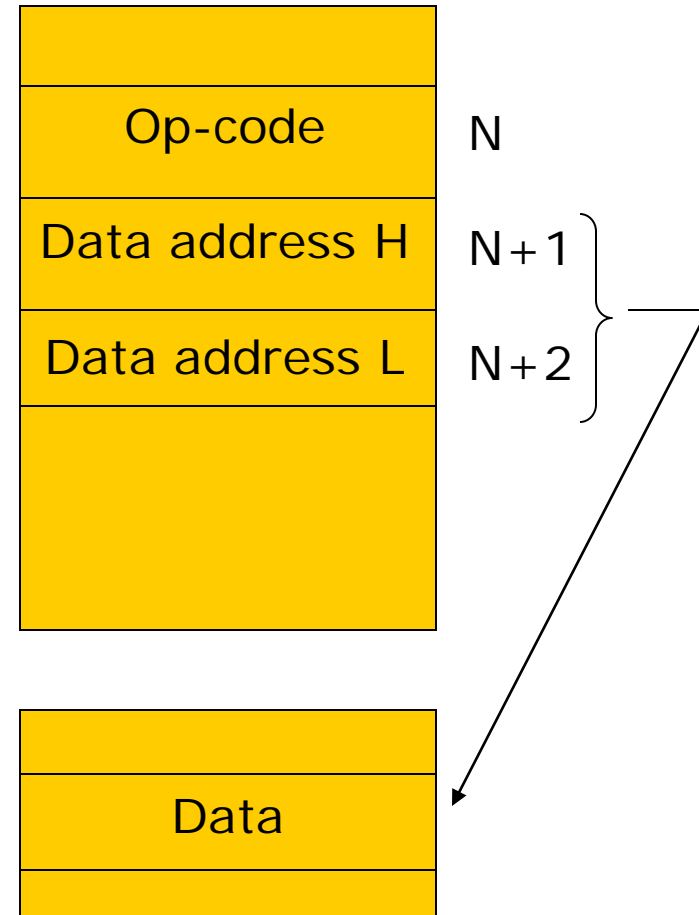
ldaa \$45 ;direct

Extended memory addressing.

Figure 4-5

Operand address follows op-code in memory (big-endian format)

Address "points" to location of operand



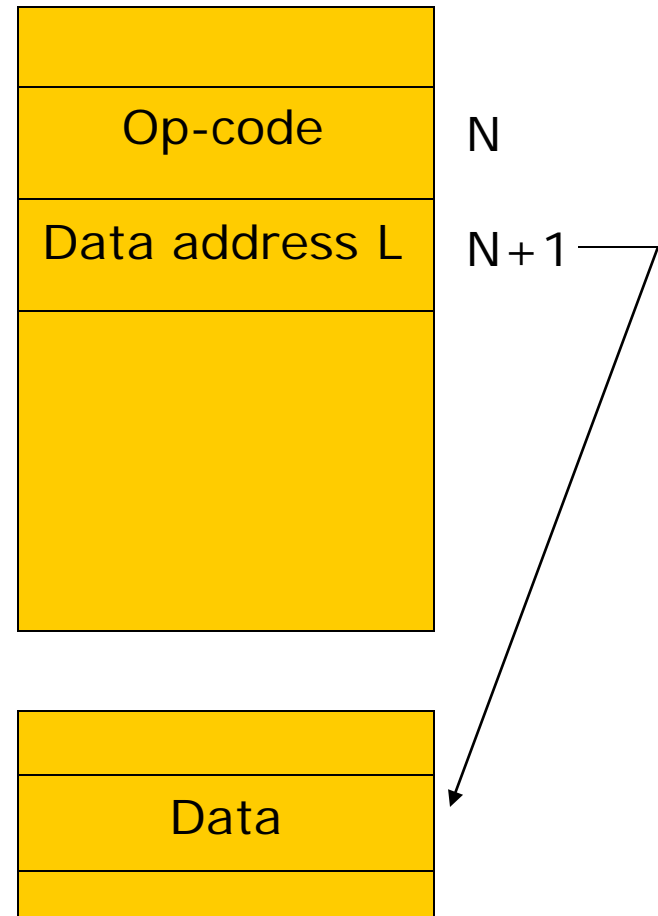
Direct memory addressing.

Low byte of operand address (XY) follows op-code in memory.

Operand address = \$00XY
Range: [\$0000...\$00FF]

Save memory by not storing upper byte of address if known to be \$00.

Direct & extended are identical from programmer's perspective.



Examples 4-3 & 4-4

Direct Addressing

```
0000 9664      1   ldaa   $64    ; ($0064) -> A
0002 5BFF      2   stab   255    ; B -> ($00FF)
0004 DE0A      3   ldx    10     ; ($000A:000B) -> X
```

Extended Addressing

```
0000 B61234    1   ldaa   $1234   ; ($1234) -> A
0003 FC1234    2   ldd    $1234   ; ($1234:1235) -> D
0006 7EC000    3   stx    $c000   ; X -> ($C000:C001)
```

The assembler decides which format to use.
(Direct used if \$0000-\$00FF, otherwise extended.)

Source: Cady/Sibrigoth
"Software and Hardware
Engineering

Symbolic labels as addresses

- Symbolic labels are usually used to specify direct/extended memory addresses

```
    org  $0000
bill:  dc.b 8,9      ; "bill" – address $0000
    org  $0800
bob:   dc.b 5,6      ; "bob" = address $0800
.....
ldaa  bob           ; 5 → a   Extended address
ldaa  bill          ; 8 → a   Direct address
ldab  bob+1         ; 6 → b   Extended address
```

Assembler allows “expressions” for creating addresses.

C vs. assembly language

□ C example:

```
char k;      /* declare variable k */  
k = 5;      /* assign value 5 to k */
```

□ Assembly language equivalent:

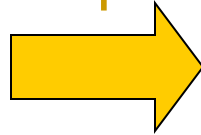
```
k:      ds.b  1      ; allocate space for k  
...  
      {  
      ldaa #5      ; 5 → a (immediate)  
      staa  k      ; store at location k  
      ; (direct or extended address)
```

Variables in C vs. Assembly Language

C Example

```
int i,j,k;  
char c;  
char a[10];  
int n[5];
```

Compiler



Assembly Language

```
i:  ds.w  1  
j:  ds.w  1  
k:  ds.w  1  
c:  ds.b  1  
a:  ds.b 10  
n:  ds.w  5
```

Indirect (indexed) addressing

- Instruction “indirectly” specifies operand location
 - tell CPU how to determine the address
- Use to:
 - specify an operand address unknown at the time of writing the instruction
 - pass an address to a subroutines
 - create an address that can be modified during program execution
 - create “array addresses”, similar to C/Java
- CPU12 calls this “indexed” addressing

Indexed addressing

- Basic format:

operation offset, index_register

- Operand address = (index_register) + offset

- Index_register is one of x , y , SP , PC

(write x instead of IX , and y instead of IY)

- Offset is a 5, 9, or 16-bit signed constant
(offset from address in index register)

Indexed addressing examples

ldx # \$1000

ldy # \$2000

ldaa \$10,x ; address = \$1000 + \$10

ldaa -\$10,y ; address = \$2000 - \$10

ldaa ,x ; address = \$1000 + \$00



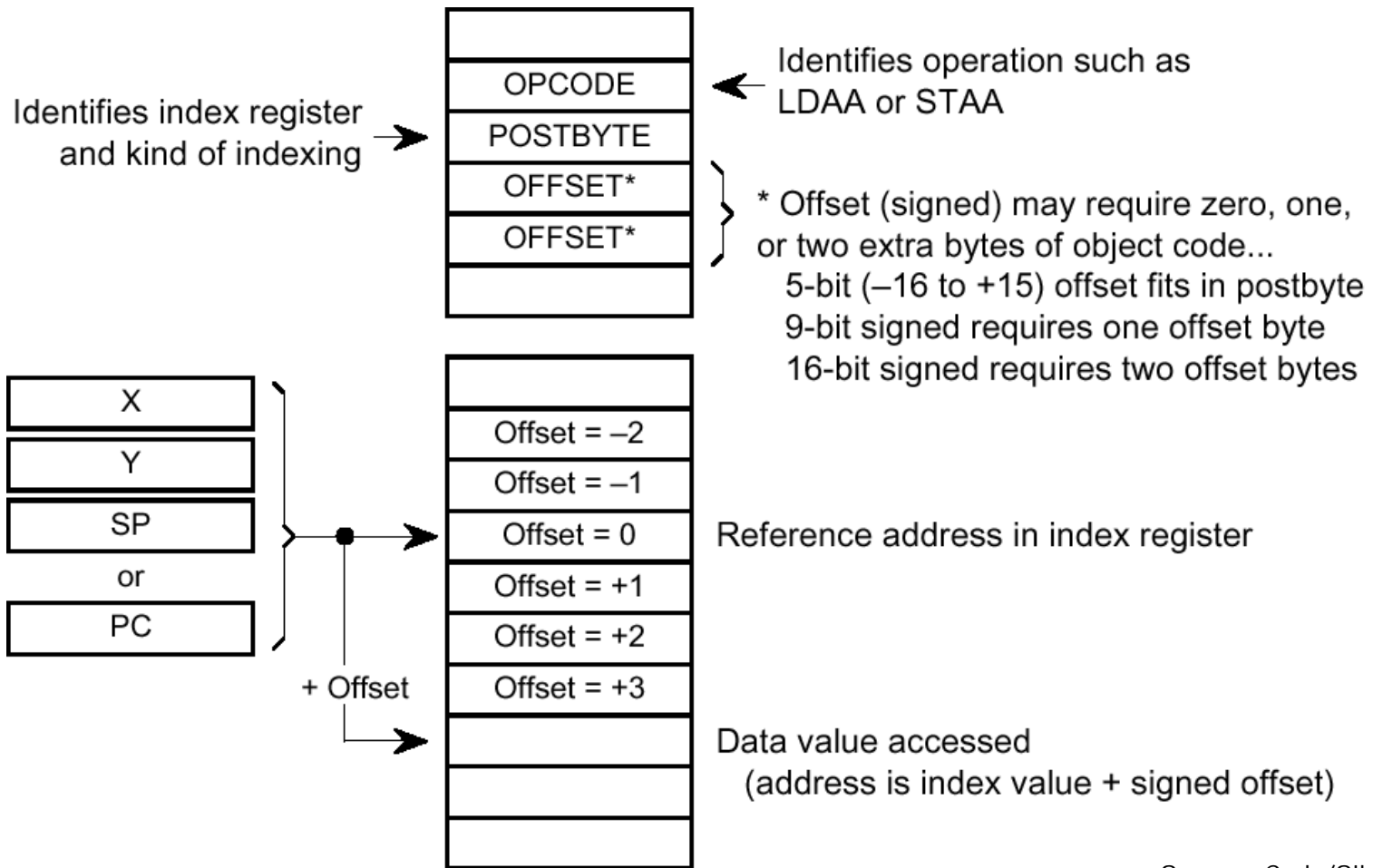
If offset is omitted, it defaults to 0

EXAMPLE 2-4

0000	A600	1	ldaa	,x	; (X+0) 5-bit offset -> A
0002	A600	2	ldaa	0,x	; (X+0) 5-bit offset -> A
0004	A6E040	3	ldaa	64,x	; (X+64) 9-bit offset -> A
0007	A6E9C0	4	ldaa	-64,y	; (Y-64) 9-bit offset -> A
000A	6A9F	5	staa	-1,SP	; A -> (SP-1) 5-bit offset
000C	A6FA1388	6	ldaa	5000,PC	; (PC+5000) 16-bit offset -> A

M68HC12 indexed addressing.

Figure 4-6



Array example

C:

```
char a[10]
```

```
char b;
```

```
...
```

```
b = a[5];
```

Assembly:

```
a: ds.b 10
```

```
b: ds.b 1
```

```
....
```

```
ldx #a
```

```
ldaa 5,x
```

```
staa b
```

IX contains address of a



C pointer example

```
char m,n;  
int  p;  
....  
p = &m;    // &a = address of m  
n = *p;    // *p = value pointed to by p
```

Assembly Language:

```
ldx #m      ; x = address of m  
ldaa ,x     ; a = value of variable m  
inx         ; x + 1 → x (point to n)  
ldab ,x     ; b = value of variable n
```

Indexed with autoincrement/decrement

- Auto-increment/decrement the index register by a given amount after using it (postincrement)

Idx #1000

ldaa 1,x+ ; [1000] → a, 1000 + 1 → x

ldaa 2,x- ; [1001] → a, 1001 - 2 → x

- Auto-increment/decrement the index register before using it (preincrement)

ldaa 2,+x ; 1000+2 → x, [1002] → a

ldaa 1,-x ; 1002-1 → x, [1001] → a

Incr/decr value must be 1...8 for these modes.

Autoincrement example

```
ary: ds.b 10
      clra          ; 0 → a
      ldx #ary      ; address of ary → x
      ldab #10      ; 10 → b (size of array)
lp:   adda 1,x+     ; a+ary[n]→a, x+1→x
      decb          ; b - 1 → b
      bne lp        ; jump if b <> 0
```

Example 4-6 Indexed addressing with post- and preincrementing/decrementing

```
0000 A629      1 ; Pre-decrement
                2   ldaa    7,-X   ; X-7 -> X, (X) -> A
0002 A63E      3 ; Post-decrement
                4   ldaa    2,X-   ; (X) -> A, X-2 -> X
0004 A620      5 ; Pre-increment
                6   ldaa    1,+X   ; X+1 -> X, (X) -> A
0006 A630      7 ; Post-increment
                8   ldaa    1,X+   ; (X) -> A, X+1 -> X
```

Indexing with offset in an accumulator

- Offset may be in accumulator A, B, or D
- Address: **register, index-register**

Examples

ldaa B,X ; [B + IX] -> A

ldx A,Y ; [A + IY] -> IX

ldaa D,X ; [D + IX] -> A

EXAMPLE 4-7

```
0000 A6E5      1      ldaa    B,X      ; (X+B) -> A
0002 E6EC      2      ldab    A,Y      ; (Y+A) -> B
0004 EDE6      3      ldy     D,X      ; (X+D:X+D+1) -> Y
```

Example – array index

□ In C

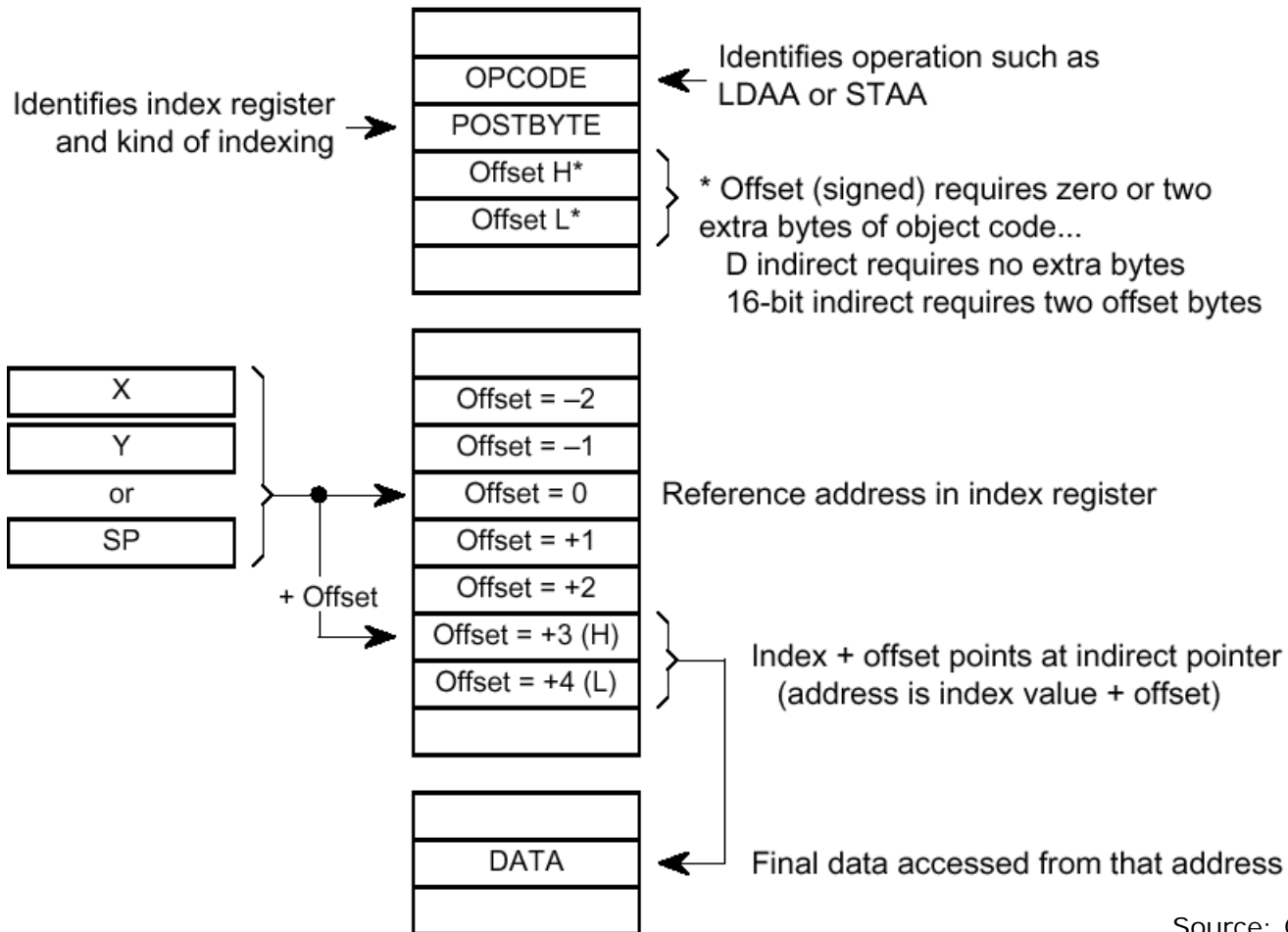
```
int i;  
char aa,cc[10];  
aa = cc[i];
```

□ In CPU12 assembly language

```
i:      ds.b 1  
aa:     ds.b 1  
cc:     ds.b 10  
        ldx  #cc    ;X=base address of cc  
        ldaa i      ;A=value of index i  
        ldab A,X    ;B=c[i]
```

M68HC12 indexed-indirect addressing.

Figure 4-7



Indirect memory addressing

Instruction specifies address of a memory location that contains the address of the operand.

Example 4-8 16-Bit Constant Indexed-Indirect Addressing

```
0000 CE5000      1    ldx      #$5000    ; $5000 -> X, Initialize X
0003 A6E30064    2    ldaa     [$64,X]    ; (($5064)) -> A
0007 6AE3FFFF    3    staa     [-1,X]    ; A -> (($4fff))
```

Example 4-9 Indexed-Indirect Addressing Using Accumulator B

```
0000 CE5000      1    ldx      #$5000    ; $5000 -> X, initialize X
0003 CC0064      2    ldd      #$064     ; $0064 -> D, initialize D
0006 EFE7        3    lds      [D,X]     ; (($5064)) -> SP
```

Source: Cady/Sibrigoth
"Software and Hardware
Engineering

Example: Relative Addressing

Branch instructions specify memory addresses that are some displacement from the current Instruction in the program

```
0000 2002      1  THERE:    bra      WHERE    ; Forward branch
0002 A7        2           nop
0003 A7        3           nop
0004 22FA      4  WHERE:    bhi      THERE    ; Conditional branch back
0006 18260256  5           lbne     LONG_BRANCH
000A          6           DS      256      ; Simulate instructions
          7  LONG_BRANCH:
0260 A7        8           nop
```

Source: Cady/Sibrigoth
"Software and Hardware
Engineering

Summary of HCS12 Indexed Operations

Operand Syntax	Comments
ldaa ,r ldaa n,r	5-, 9- or 16-bit, signed, constant offset n = -16 to +15 for 5-bit n = -256 to +255 for 9-bit n = -32,768 to 32,767 for 16-bit ⁴ r can be X, Y, SP, or PC; r is not changed by the instruction.
ldaa n,-r	Automatic pre-decrement n = 1 to 8 and is subtracted from the contents of register r before the data value is fetched. r can be X, Y, or SP (not PC); r is modified by the instruction.
ldaa n,+r	Automatic pre-increment n = 1 to 8 and is added to the contents of register r before the data value is fetched. r can be X, Y, or SP (not PC); r is modified by the instruction.

Summary of M68HC12 Indexed Operations (cont)

Operand Syntax	Comments
ldaa n,r-	Automatic post-decrement n = 1 to 8 and is subtracted from register r after the data value is fetched. r can be X, Y, or SP (not PC); r is modified by the instruction.
ldaa n,r+	Automatic post-increment n = 1 to 8 and is added to register r after the data value is fetched. r can be X, Y or SP (not PC); r is modified by the instruction.
ldaa A,r ldaa B,r ldaa D,r	Accumulator offset The contents of A, B, or D are used as a 16-bit, unsigned offset. r can be X, Y, SP, or PC; r is not changed by the instruction.

Summary of M68HC12 Indexed Operations (cont)

Operand Syntax	Comments
Idaa [n,r]	16-bit offset indexed-indirect r can be X, Y, SP, or PC; r is not changed by the instruction.
Idaa [D,r]	Accumulator D offset indexed-indirect r can be X, Y, SP, or PC; r is not changed by the instruction.

Source: Cady/Sibrigoth
"Software and Hardware
Engineering