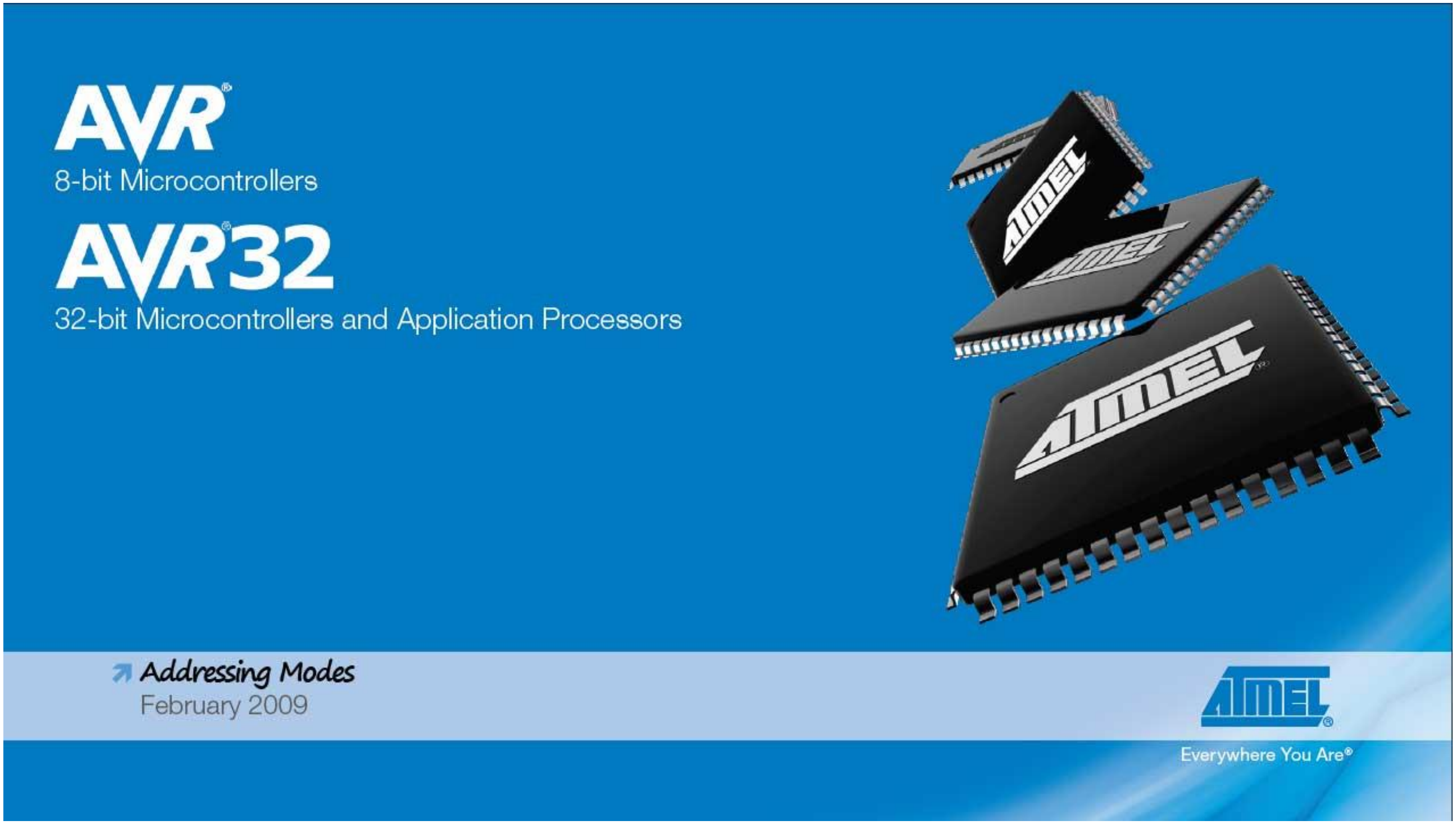


AVR®

8-bit Microcontrollers

AVR®32

32-bit Microcontrollers and Application Processors

Addressing Modes

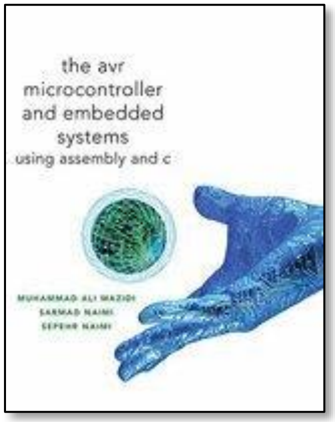
February 2009

ATMEL®

Everywhere You Are®

Addressing Modes Part II – AVR Addressing Indirect

READING



The AVR Microcontroller and Embedded Systems using Assembly and C)
by Muhammad Ali Mazidi, Sarmad Naimi, and Sepehr Naimi

Chapter 6: AVR Advanced Assembly Language Programming

Section 6.1: Introducing some more assembler directives

Section 6.3: Register Indirect Addressing Mode

Section 6.4: Look-up Table and Table Processing

CONTENTS

Working with Arrays

- Here is a C++ code example which adds 5 numbers contained within array foo.
- Foo in computer science acts as a placeholder for any command, file, directory, variable, function, and procedure.
- Source: http://www.cplusplus.com/doc/tutorial/arrays/

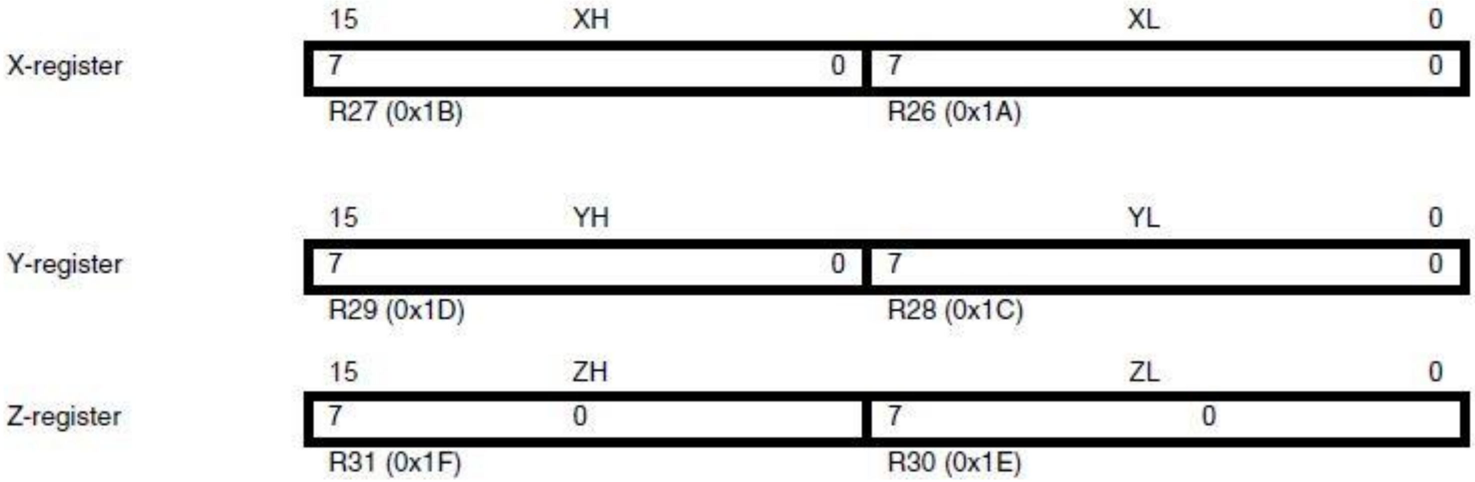
```
// arrays example
#include <iostream>
using namespace std;

uint8_t foo[] = {16, 2, 77, 40, 107};
uint8_t n, result=0;

uint8_t addArray ()
{
  for ( n=0 ; n<5 ; ++n )
  {
    result += foo[n];
  }
  return result;
}
```

The X-register, Y-register, and Z-register – *Review*

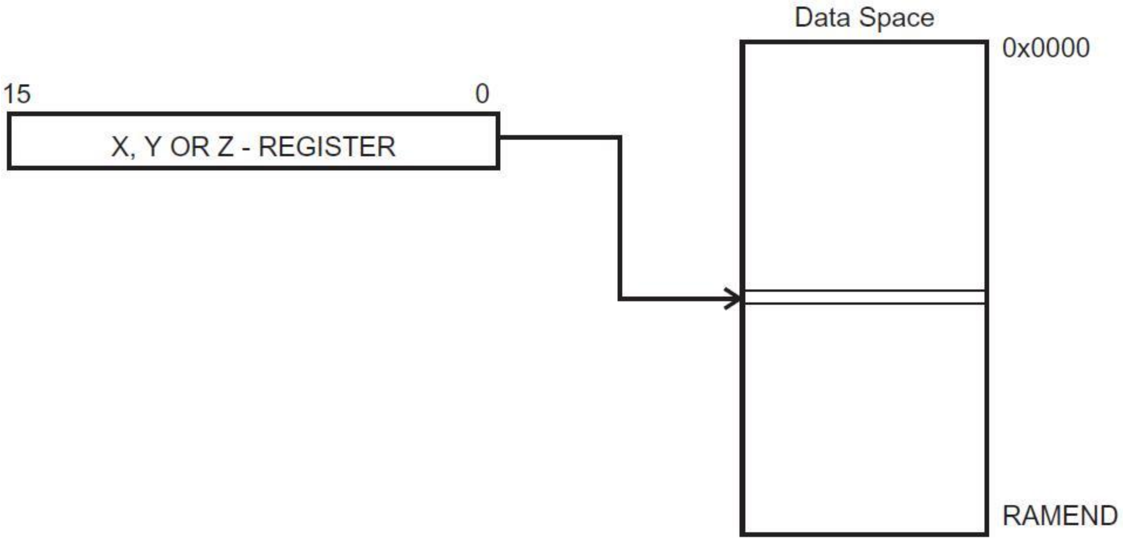
The registers R26..R31 have some added functions to their general purpose usage. These registers are 16-bit address pointers for indirect addressing of the data space. The three indirect address registers X, Y, and Z are defined as described here.



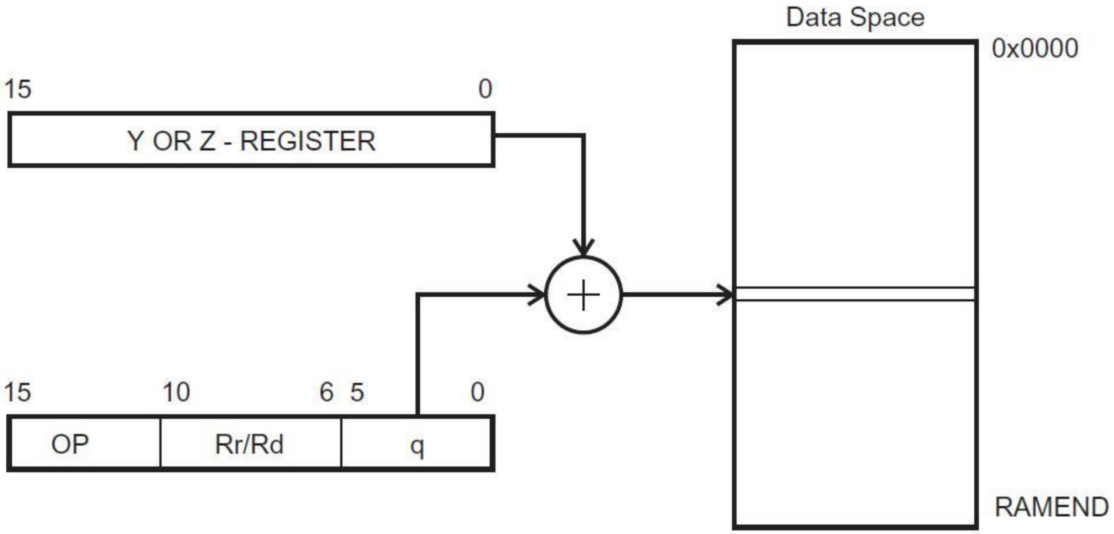
In the different addressing modes these address registers have functions as fixed displacement, automatic increment, and automatic decrement (see the instruction set reference for details).

DATA INDIRECT

```
ld     r16, X
st     X, r16
```

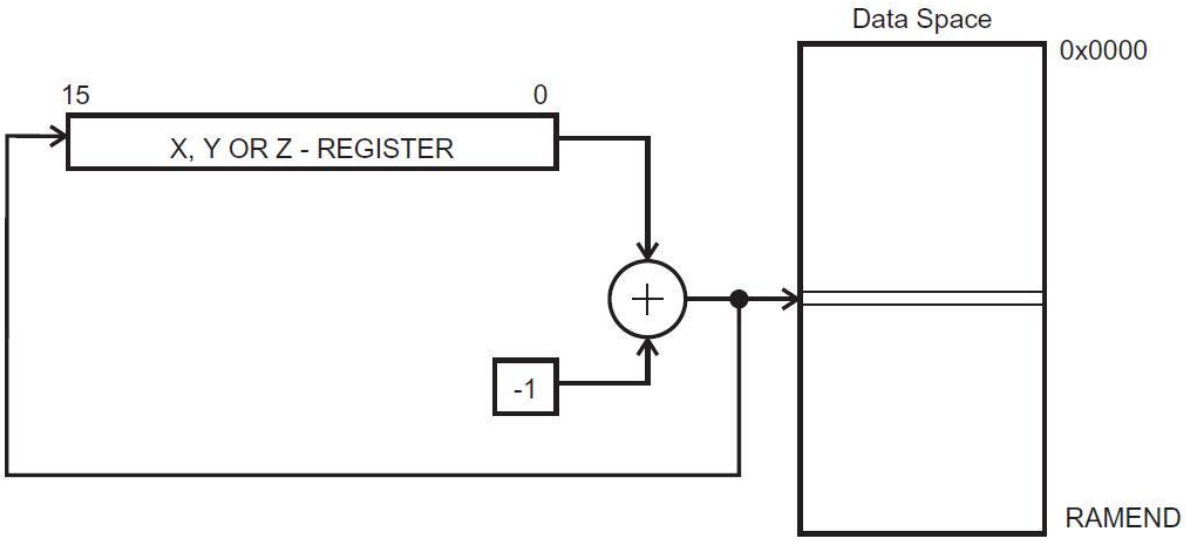


```
ldd     r4, Z+2
std     Y+2, r4
```

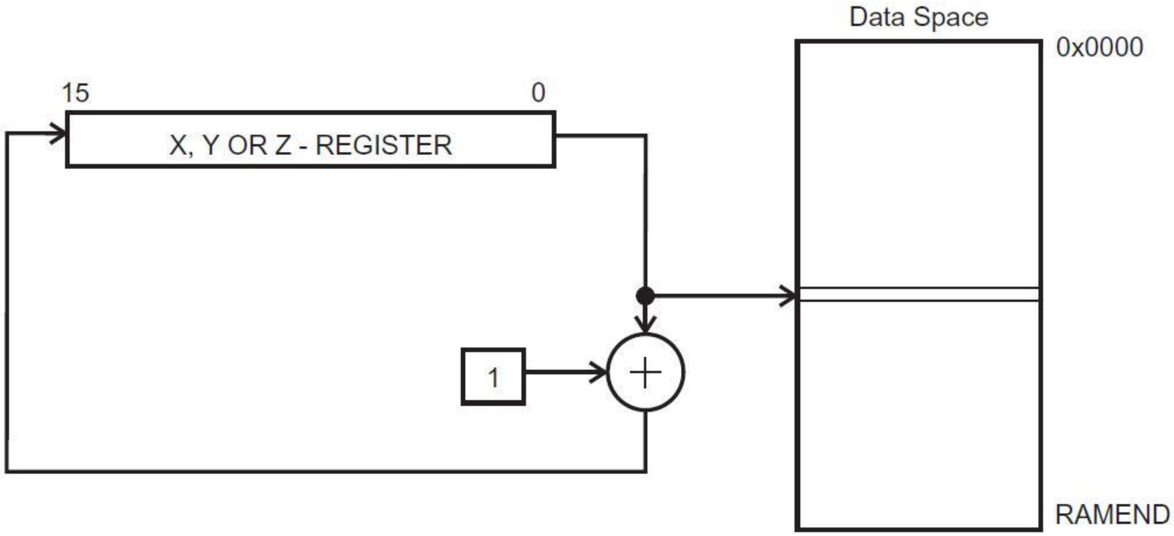


Data Indirect with Pre-decrement/Post-Increment

```
ld     r16, -Y
st     -Y, r16
```



```
ld     r4, Y+
st     Y+, r4
```

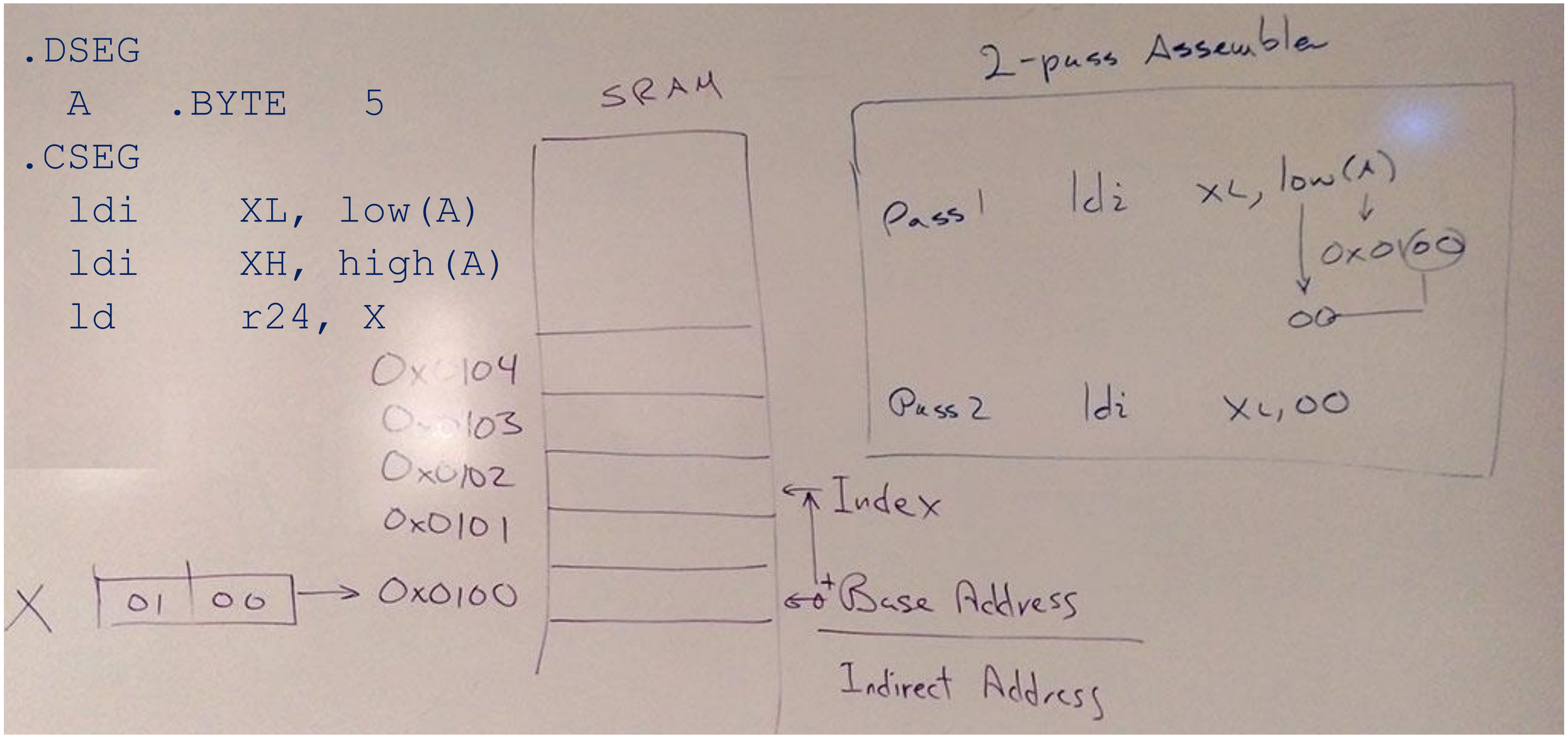


SRAM Data Indirect – Example 1
Base Address

C++ Code

```
uint8_t result, A[5] = {16, 2, 77, 40, 107};
result = A[0];
```

Assembly Code



```
.DSEG
  A    .BYTE   5
.CSEG
  ldi      XL, low(A)
  ldi      XH, high(A)
  ld       r24, X
```

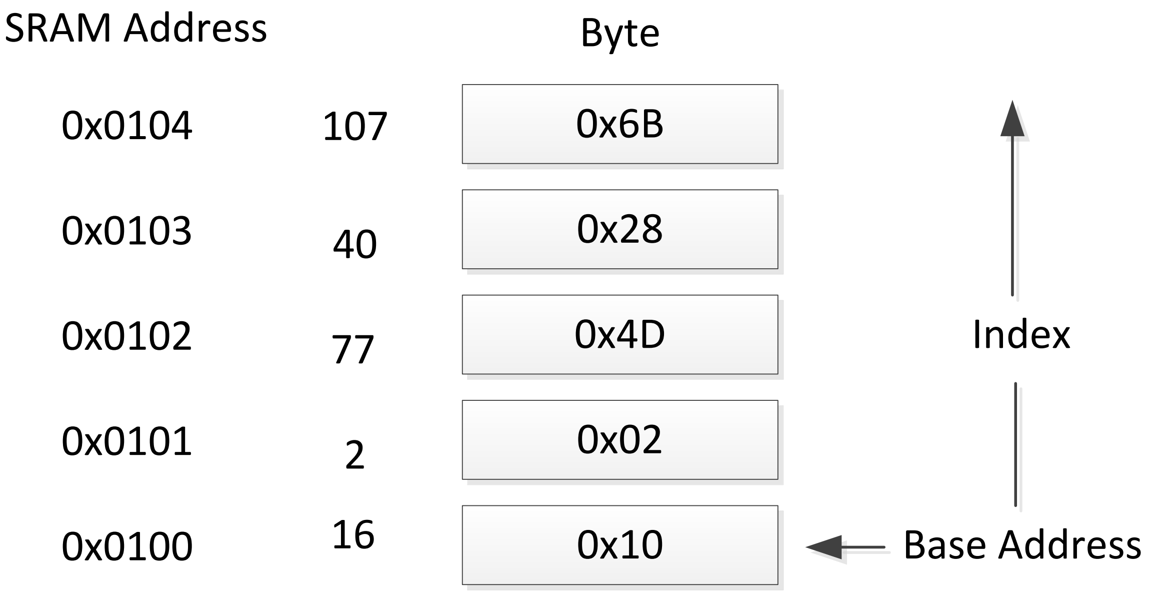
SRAM Data Indirect – Example 2
8-bit Sum Looping

C++ Code

```
uint8_t n, result, A[5] = {16, 2, 77, 40, 107};
for ( n=0 ; n<5 ; n++ ){
  result += A[n];
}
```

Assembly Code

```
.DSEG
  A    .BYTE   5
.CSEG
  ldi   XL, low(A)
  ldi   XH, high(A)
  clr   r24
  ldi   r17,0x05
loop:
  ld    r18, X+
  add   r24, r18
  dec   r17
  brne loop
```



SRAM Address		Byte
0x0104	107	0x6B
0x0103	40	0x28
0x0102	77	0x4D
0x0101	2	0x02
0x0100	16	0x10

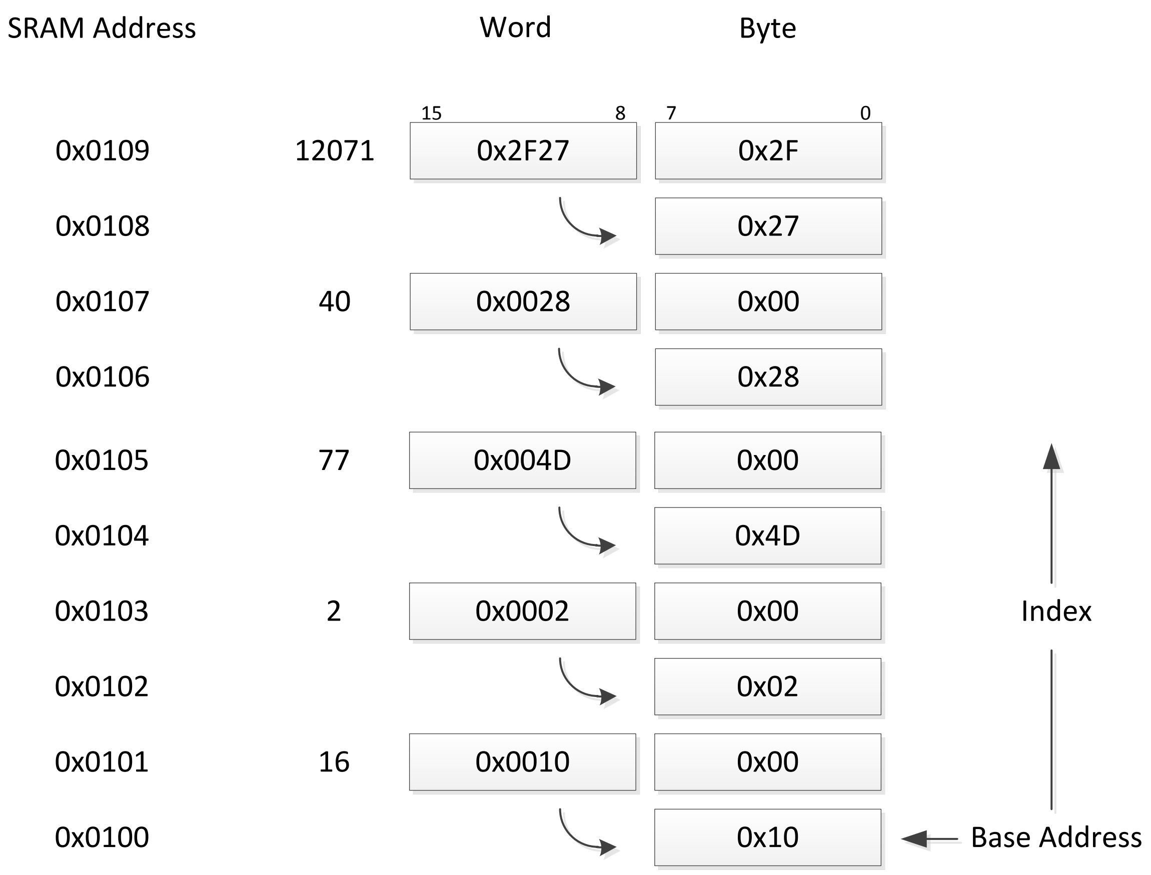
SRAM Data Indirect – Example 3
16-bit Sum Looping

C++ Code

```
uint16_t X, A[5] = {16, 2, 77, 40, 12071}; // 16-bit unsigned integer
for ( n=0 ; n<5 ; n++ ){
  result += A[n];
}
```

Assembly Code

```
.DSEG
  A    .BYTE   10
.CSEG
  ldi  YL, low(A)
  ldi  YH, high(A)
  clr  r25
  clr  r24
  ldi  r17,0x05
loop:
  ld   r18, Y+
  add  r24, r18
  ld   r18, Y+
  adc  r25, r18
  dec  r17
  brne loop
```



SRAM Address		Word (15 … 8)	Byte (7 … 0)
0x0109	12071	0x2F27	0x2F
0x0108			0x27
0x0107	40	0x0028	0x00
0x0106			0x28
0x0105	77	0x004D	0x00
0x0104			0x4D
0x0103	2	0x0002	0x00
0x0102			0x02
0x0101	16	0x0010	0x00
0x0100			0x10

SRAM Data Indirect– Example 4

- Write a program to display the 16-bit result of a 8 x 8 multiplication, where the result is stored in the r1:r0 register pair. Save result into SRAM using Little Endian byte ordering.

```
.DSEG
buffer:    .BYTE     4               // blink status

.CSEG
.ORG 0x0000

LoadBuffer:
    ldi     XL,low(buffer)    // load address of look-up
    ldi     XH,high(buffer)
    st      X+, r0
    swap    r0
    st      X+, r0
    st      X+, r1
    swap    r1
    st      X+, r1
    ret

DisplayBuffer:
    ldi     XL,low(buffer+4)    // load address of look-up
    ldi     XH,high(buffer+4)
    ldi     r20, 4
cont:
    ld      r0, -X
    rcall   BCD_to_7SEG
    rcall   Delay1S
    dec     r20
    brne    cont
    ret
```

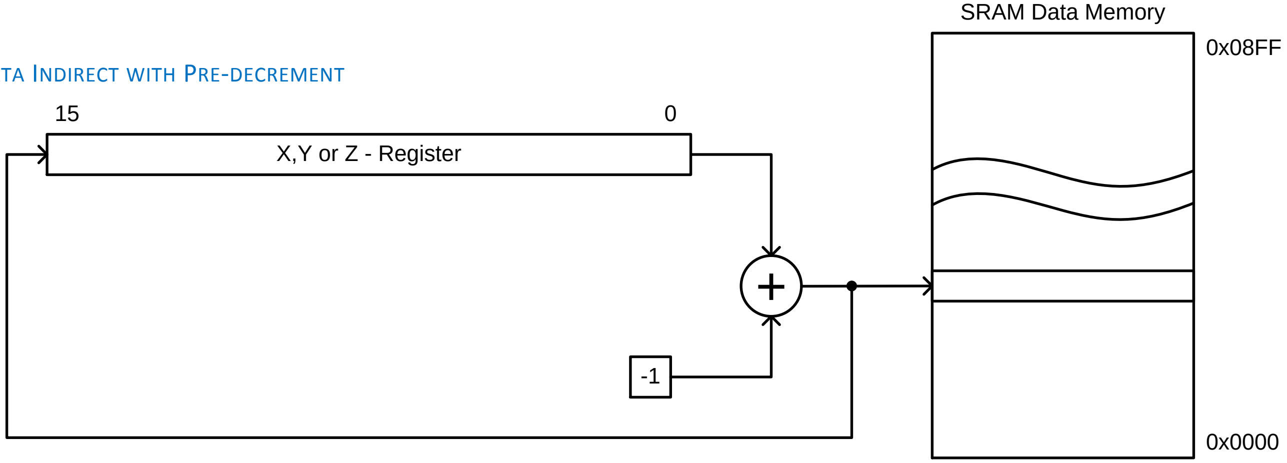
Appendix A Data Indirect with Pre-decrement/Post-Increment – Instruction Encoding

	Rn
0000	
0001	Z+
0010	-Z
0100	*see lpm*
0101	*see lpm*
1001	Y+
1010	-Y
1100	X
1101	X+
1110	-X
1111	pop (+SP) push (SP-)

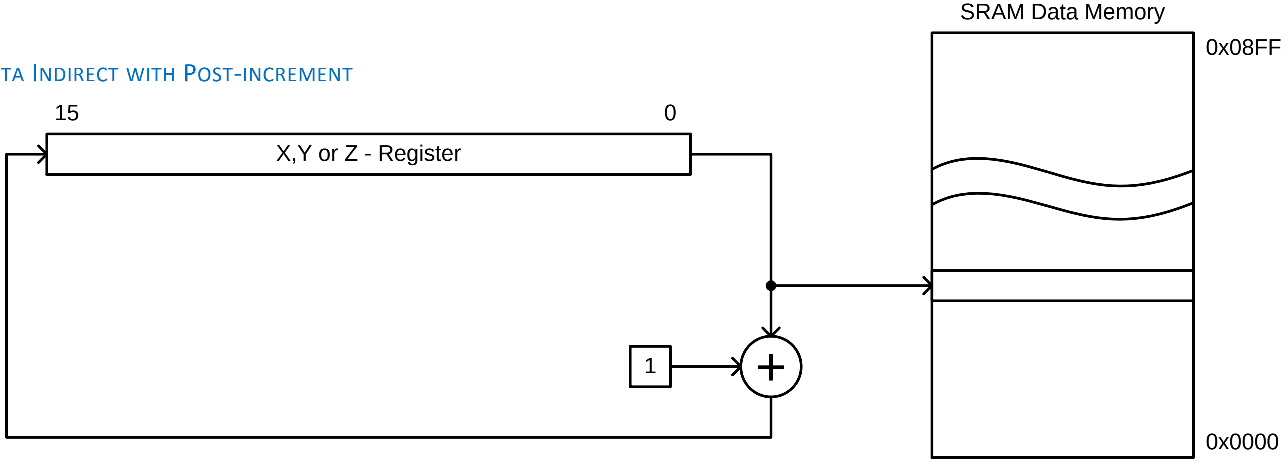
```
ld    Rd, Rn
pop   Rd
st    Rn, Rr
push  Rr
```

15 … 12	11 … 8	7 … 4	3 … 0
1001	000d	dddd	nnnn
1001	000d	dddd	1111
1001	001r	rrrr	nnnn
1001	001r	rrrr	1111

Data Indirect with Pre-decrement



Data Indirect with Post-increment


SRAM Data Memory
0x08FF
15
0
X,Y or Z - Register
1
+
0x0000