Register and Direct Addressing Modes

Hoda Roodaki
hroodaki@kntu.ac.ir

Register and Direct Addressing Modes

- The CPU can access data in various ways.
  - in a register
  - in memory
  - provided as an immediate value
- These various ways of accessing data are called addressing modes.
- The various addressing modes of a microprocessor are determined when it is designed, and therefore cannot be changed by the programmer.

AVR Addressing Modes

- The AVR addressing modes:
  - Single-Register (Immediate)
  - Register
  - Direct
  - Register indirect

Single-register (immediate) addressing mode

- In this addressing mode, the operand is a register.

```
NEG   R18              ;negate the contents of R18
COM   R19              ;complement the contents of R19
INC   R20              ;increment R20
DEC   R21              ;decrement R21
ROR   R22              ;rotate right R22
```

- In some of the instructions there is also a constant value with the register operand.

```
LDI   R19,0x25         ;load 0x25 into R19
SUBI  R19,0x6          ;subtract 0x6 from R19
ANDI  R19,0b01000000   ;AND R19 with 0x40
```

Two-register addressing mode

- Two-register addressing mode involves the use of two registers to hold the data to be manipulated.

```
ADD   R20,R23         ;add R23 to R20
SUB   R29,R20         ;subtract R20 from R29
AND   R16,R17         ;AND R16 with 0x40
MOV   R23,R19         ;copy the contents of R19 to R23
```

Direct addressing mode

- In direct addressing mode, the operand data is in a RAM memory location whose address is known, and this address is given as a part of the instruction.

```
LDS    R19,0x560  ;load R19 with the contents of memory loc $560
STS    0x40,R19   ;store R19 to data space location 0x40
```

Direct addressing mode

- **I/O direct addressing mode**
- To access the I/O registers there is a special mode called ***I/O direct addressing*** *mode.*
- The I/O direct addressing mode can address only the standard I/O registers.
  – The IN and OUT instructions use this addressing mode.
  – The addresses between $20 and $5F of the data space have been assigned to standard I/O registers in all of the AVRs.
  – These I/O registers have two addresses:
    - I/O address
    - Data memory address
    - The I/O address is used when we use the I/O direct addressing mode, while the data memory address is used when we use the direct addressing mode.
    - In other words, the standard I/O registers can be accessed using both the direct addressing and the I/O addressing modes.

Direct addressing mode

- **I/O direct addressing mode**
  - Example

```
OUT 0x15,R19 ;PORTC=R19 (0x15 is the I/O addr. of PORTC)
STS 0x35,R19 ;PORTC=R19 (0x35 is the data memory addr. of PORTC)

IN  R19,0x16 ;R19=PINB (0x16 is the I/O addr. of PINB)
LDS R19,0x36 ;R19=PINB (0x36 is the data memory addr. of PINB)
```

Direct addressing mode

- **I/O direct addressing mode**
- **<u>First Challenge</u>**
  - Some AVRs have more than 64 I/O registers.
  - The extra I/O registers are located above the data memory address $5F.
    - *extended I/O memory*
  - In the I/O direct addressing mode, the address field is a 6-bit address and can take values from $00-$3F, which is from 00 to 63 in decimal.
    - So, it can address only the standard I/O register memory, and it cannot be used for addressing the extended I/O memory.
    - To access the extended I/O registers we can use the direct addressing mode.
      - OUT 0x65, R19 ;illegal as the address is above $3F
      - STS 0x65, R19

Direct addressing mode

- **I/O direct addressing mode**
- **<u>Second Challenge</u>**
  - The I/O registers can have different addresses in different AVR microcontrollers.
    - The same instruction can have different meanings in different AVR microcontrollers.
    - The best way to solve this problem is to use the names of the registers instead of their addresses.

Example 6-4

Write code to send $55 to Port B. Include
(a) the register name,
(b) the I/O address, and
(c) the data memory address.

Solution:

(a)

```
LDI   R20,0xFF     ;R20 = 0xFF
OUT   DDRB,R20     ;DDRB = R20 (Port B output)
LDI   R20,0x55     ;R20 = $55
OUT   PORTB,R20    ;Port B = 0x55
```

(b) From Table 6-4, DDRB I/O address = $17 and PORTB I/O address = $18.

```
LDI   R20,0xFF     ;R20 = 0xFF
OUT   0x17,R20     ;DDRB = R20 (Port B output)
LDI   R20,0x55     ;R20 = $55
OUT   0x18,R20     ;Port B = 0x55
```

(c) From Table 6-4, DDRB data memory address = $37 and PORTB data memory address = $38.

```
LDI   R20,0xFF     ;R20 = 0xFF
STS   0x37,R20     ;DDRB = R20 (Port B output)
LDI   R20,0x55     ;R20 = $55
STS   0x38,R20     ;Port B = 0x55
```

Register Indirect Addressing Mode

- In the register indirect addressing mode, a register is used as a pointer to the data memory location.
- In the AVR, three registers are used for this purpose: X, Y, and Z.
  - These are 16-bit registers allowing access to the entire 65,536 bytes of data memory space in the AVR.

Register Indirect Addressing Mode

- Each of the registers is made by combining two specific GPRs.
  - X register: combining R26 and R27
    - R26 is the lower byte of X.
    - R27 is the higher byte of X.
  - Y register: combining R28 and R29
  - Z register: combining R30 and R31
    - The R26, R27, R28, R29, R30, and R31 GPRs can be referred to as XL, XH, YL, YH, ZL, and ZH, respectively.
      - For example, "LDI XL,0x31" is the same as "LDI R26,0x31" since XL is another name for R26.

Register Indirect Addressing Mode

- The 16-bit registers X, Y, and Z are used as pointers.
- We can use them with the LD instruction to read the value of a location pointed to by these registers.
  - The following program loads the contents of location 0x130 into R18:

```
LDI    XL, 0x30     ;load R26 (the low byte of X) with 0x30
LDI    XH, 0x01     ;load R27 (the high byte of X) with 0x1
LD     R18, X       ;copy the contents of location 0x130 to R18

LDI    ZL, 0x9F     ;load 0x9F into the low byte of Z
LDI    ZH, 0x13     ;load 0x13 into the high byte of Z (Z=0x139F)
ST     Z, R23       ;store the contents of location 0x139F in R23
```

Advantages of register indirect addressing mode

- It makes accessing data dynamic rather than static, as with direct addressing mode.

Example 6-5

Write a program to copy the value \$55 into memory locations \$140 through \$144 using
(a) direct addressing mode,
(b) register indirect addressing mode without a loop, and
(c) a loop.

Solution:

(a)
```
LDI   R17,0x55        ;load R17 with value 0x55
STS   0x140,R17       ;copy R17 to memory location 0x140
STS   0x141,R17       ;copy R17 to memory location 0x141
STS   0x142,R17       ;copy R17 to memory location 0x142
STS   0x143,R17       ;copy R17 to memory location 0x143
STS   0x144,R17       ;copy R17 to memory location 0x144
```

(b)
```
LDI   R16,0x55   ;load R16 with value 0x55
LDI   YL,0x40    ;load R28 with value 0x40 (low byte of addr.)
LDI   YH,0x1     ;load R29 with value 0x1 (high byte of addr.)
ST    Y,R16      ;copy R16 to memory location 0x140
INC   YL         ;increment the low byte of Y
ST    Y,R16      ;copy R16 to memory location 0x141
INC   YL         ;increment the pointer
ST    Y,R16      ;copy R16 to memory location 0x142
INC   YL         ;increment the pointer
ST    Y,R16      ;copy R16 to memory location 0x143
INC   YL         ;increment the pointer
ST    Y,R16      ;copy R16 to memory location 0x144
```

Example 6-5

```
(c)   LDI   R16,0x5     ;R16 = 5 (R16 for counter)
      LDI   R20,0x55    ;load R20 with value 0x55 (value to be copied)
      LDI   YL,0x40     ;load YL with value 0x40
      LDI   YH,0x1      ;load YH with value 0x1
L1:   ST    Y,R20       ;copy R20 to memory pointed to by Y
      INC   YL          ;increment the pointer
      DEC   R16         ;decrement the counter
      BRNE  L1          ;loop while counter is not zero
```

Use the AVR Studio simulator to examine memory contents after the above program is run.

$140 = ($55) $141 = ($55) $142 = ($55) $143 = ($55)   144 = ($55)

Auto-increment and auto-decrement options for pointer registers

- Because the pointer registers (X, Y, and Z) are 16-bit registers, they can go from $0000 to $FFFF, which covers the entire 64K memory space of the AVR.
  - Using the “INC ZL" instruction to increment the pointer can cause a problem when an address such as $5FF is incremented.
  - The instruction “INC ZL," will not propagate the carry into the ZH register.
  - The AVR gives us the options of auto-increment and auto-decrement for pointer registers to overcome this problem.

Auto-increment and auto-decrement options for pointer registers

Instruction		Function
LD	Rn,X	After loading location pointed to by X, the X stays the same.
LD	Rn,X+	After loading location pointed to by X, the X is incremented.
LD	Rn,-X	The X is decremented, then the location pointed to by X is loaded.
LD	Rn,Y	After loading location pointed to by Y, the Y stays the same.
LD	Rn,Y+	After loading location pointed to by Y, the Y is incremented.
LD	Rn,-Y	The Y is decremented, then the location pointed to by Y is loaded.
LDD	Rn,Y+q	After loading location pointed to by Y+q, the Y stays the same.
LD	Rn,Z	After loading location pointed to by Z, the Z stays the same.
LD	Rn,Z+	After loading location pointed to by Z, the Z is incremented.
LD	Rn,-Z	The Z is decremented, then the location pointed to by Z is loaded.
LDD	Rn,Z+q	After loading location pointed to by Z+q, the Z stays the same.

- This table shows the syntax for the LD instruction, but it works for all such instructions.
- The auto-decrement or auto-increment affects the entire 16 bits of the pointer register and has no effect on the status register. This means that pointer register going from FFFF to 0000 will not raise any flag.

Example 6-6

Assume that RAM locations \$90–\$94 have a string of ASCII data, as shown below.

\$90 = ('H')   \$91 = ('E')   \$92 = ('L')   \$93 = ('L')   \$94 = ('O')

Write a program to get each character and send it to Port B one byte at a time. Show the program using:

(a) Direct addressing mode.

(b) Register indirect addressing mode.

Solution:

(a) Using direct addressing mode

```
LDI   R20,0xFF
OUT   DDRB,R20          ;make Port B an output
LDS   R20,0x90          ;R20 = contents of location 0x90
OUT   PORTB,R20         ;PORTB = R20
LDS   R20,0x91          ;R20 = contents of location 0x91
OUT   PORTB,R20         ;PORTB = R20
LDS   R20,0x92          ;R20 = contents of location 0x92
OUT   PORTB,R20         ;PORTB = R20
LDS   R20,0x93          ;R20 = contents of location 0x93
OUT   PORTB,R20         ;PORTB = R20
LDS   R20,0x94          ;R20 = contents of location 0x94
OUT   PORTB,R20         ;PORTB = R20
```

Example 6-6

(b) Using register indirect addressing mode

```
        LDI     R16,0x5         ;R16=0x5 (R16 for counter)
        LDI     R20,0xFF
        OUT     DDRB,R20        ;make Port B an output
        LDI     ZL,0x90         ;the low byte of address (ZL = 0x90)
        LDI     ZH,0x0          ;the high byte of address (ZH = 0x0)
L1:     LD      R20,Z           ;read from location pointed to by Z
        INC     ZL              ;increment pointer
        OUT     PORTB,R20       ;send to PortB the contents of R20
        DEC     R16             ;decrement counter
        BRNE    L1              ;if R16 is not zero go to L1
```

Example 6-7

Write a program to clear 16 memory locations starting at data memory address $60.
Use the following:
(a) INC Rn
(b) Auto-increment

Solution:

```
(a)     LDI     R16, 16         ;R16 = 16 (counter value)
        LDI     XL, 0x60        ;XL = the low byte of address
        LDI     XH, 0x00        ;XH = the high byte of address
        LDI     R20, 0x0        ;R20 = 0
L1:     ST      X, R20          ;clear location X points to
        INC     XL              ;increment pointer
        DEC     R16             ;decrement counter
        BRNE    L1              ;loop until counter = zero

(b)     LDI     R16, 16         ;R16 = 16 (counter value)
        LDI     XL, 0x60        ;the low byte of X = 0x60
        LDI     XH, 0x00        ;the high byte of X = 0
        LDI     R20, 0x0        ;R20 = 0
L1:     ST      X+, R20         ;clear location X points to
        DEC     R16             ;decrement counter
        BRNE    L1              ;loop until counter = zero
```

Example 6-8

Assume that data memory locations $240–$243 have the following hex data. Write a program to add them together and place the result in locations $220 and $221.

$240 = ($7D) $241 = ($EB) $242 = ($C5) $243 = ($5B)

Solution:

```
      .INCLUDE "M32DEF.INC"
      .EQU L_BYTE = 0x220      ;RAM loc for L_Byte
      .EQU H_BYTE = 0x221      ;RAM loc for H_Byte
      LDI   R16,4
      LDI   R20,0
      LDI   R21,0
      LDI   XL, 0x40    ;the low byte of X = 0x40
      LDI   XH, 0x02    ;the high byte of X = 02
L1:   LD    R22, X+     ;read contents of location where X points to
      ADD   R20, R22
      BRCC  L2          ;branch if C = 0
      INC   R21         ;increment R21
L2:   DEC   R16         ;decrement counter
      BRNE  L1          ;loop until counter is zero
      ST    L_BYTE, R20 ;store the low byte of the result in $220
      ST    H_BYTE, R21 ;store the high byte of the result in $221
```

Example 6-9

Write a program to copy a block of 5 bytes of data from data memory locations starting at $130 to RAM locations starting at $60.

Solution:

```
        LDI     R16, 16         ;R16 = 16 (counter value)
        LDI     XL, 0x30        ;the low byte of address
        LDI     XH, 0x01        ;the high byte of address
        LDI     YL, 0x60        ;the low byte of address
        LDI     YH, 0x00        ;the high byte of address
L1:     LD      R20, X+         ;read where X points to
        ST      Y+, R20         ;store R20 where Y points to
        DEC     R16             ;decrement counter
        BRNE    L1              ;loop until counter = zero
```

Before we run the above program.

```
130 = ('H') 131 = ('E') 132 = ('L') 133 = ('L') 134 = ('O')
```

After the program is run, the addresses $60–$64 have the same data as $130–$134.

```
130 = ('H') 131 = ('E') 132 = ('L') 133 = ('L') 134 = ('O')
60 = ('H')  61 = ('E')  62 = ('L')  63 = ('L')  64 = ('O')
```

Example 6-10

Two multibyte numbers are stored in locations \$130–\$133 and \$150–\$153. Write a program to add the multibyte numbers and save the result in address \$160–\$163.

```
        $C7659812
+       $2978742A
```

Solution:

```
        .INCLUDE "M32DEF.INC"
        LDI   R16, 4          ;R16 = 4 (counter value)
        LDI   XL, 0x30
        LDI   XH, 0x1         ;load pointer. X = $130
        LDI   YL, 0x50
        LDI   YH, 0x1         ;load pointer. Y = $150
        LDI   ZL, 0x60
        LDI   ZH, 0x1         ;load pointer. Z = $160
        CLC                   ;clear carry
L1:     LD    R18, X+         ;copy memory to R18 and INC X
        LD    R19, Y+         ;copy memory to R19 and INC Y
        ADC   R18,R19         ;R18 = R18 + R19 + carry
        ST    Z+,R18          ;store R18 in memory and INC Z
        DEC   R16             ;decrement R16 (counter)
        BRNE  L1              ;loop until counter = zero
```

Example 6-11

Write a function that adds the contents of three continuous locations of data space and stores the result in the first location. The Z register should point to the first location before the function is called.

Solution:

```
.INCLUDE "M32DEF.INC"
        LDI   R16,HIGH(RAMEND)  ;initialize the stack pointer
        OUT   SPH,R16
        LDI   R16,LOW(RAMEND)
        OUT   SPL,R16
        LDI   ZL,0x00           ;initialize the Z register
        LDI   ZH,2
        CALL  ADD3LOC           ;call add3loc
HERE:   JMP   HERE              ;loop forever
ADD3LOC:
        LDI   R21,0             ;R21 = 0
        LD    R20,Z             ;R20 = contents of location  Z
        LDD   R16,Z+1           ;R16 = contents of location Z+1
        ADD   R20,R16           ;R20 = R20 + R16
        BRCC  L1                ;branch if carry cleared
        INC   R21               ;increment R21 if carry occurred
L1:     LDD   R16,Z+2           ;R16 = contents of location Z+2
        ADD   R20,R16           ;R20 = R20 + R16
        BRCC  L2                ;branch if carry cleared
        INC   R21               ;increment R21
L2:     ST    Z,R20             ;store R20 into location Z
        STD   Z+1,R21           ;store R21 into location Z+1
        RET
```