به نام خدا

آشنایی با زبان اسمبلی AVR

مدهای آدرس دهی

Dr. Aref Karimiafshar
A.karimiafshar@ec.iut.ac.ir

مدهای آدرس دهی

- CPU can access data in various ways
  - These various ways of accessing data are called addressing mode
- Addressing mode of a microprocessor are determined when it is designed
- Addressing mode in AVR:

1. Single-Register (Immediate)
2. Register
3. Direct
4. Register indirect
5. Flash Direct
6. Flash Indirect

Single-register (Immediate) addressing mode

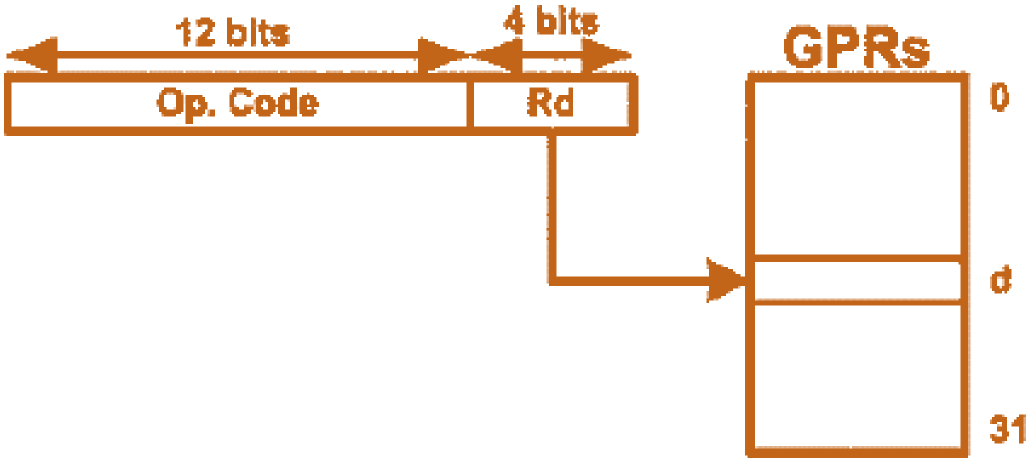
In this addressing mode, the operand is a register.

```
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
```

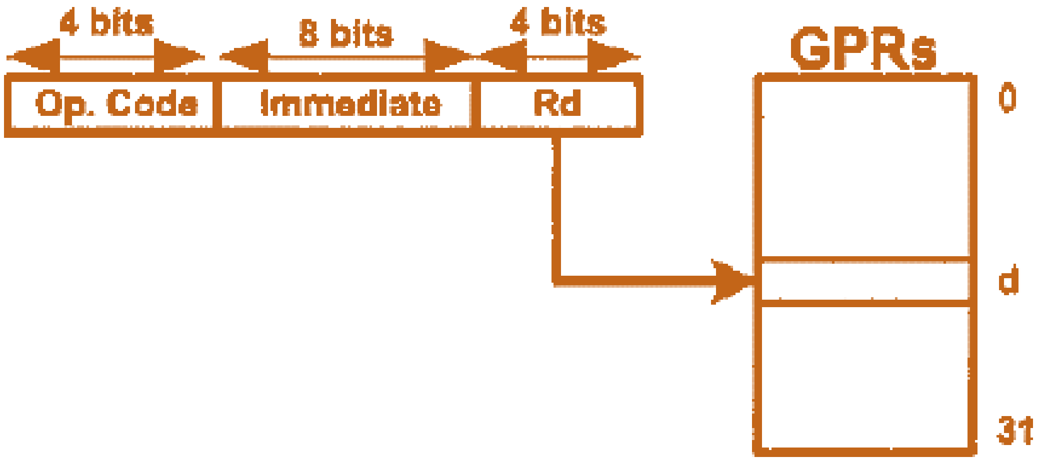
In some of the instructions there is also a constant value with the register operand.

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

Single-register (Immediate) addressing mode



NEG R18

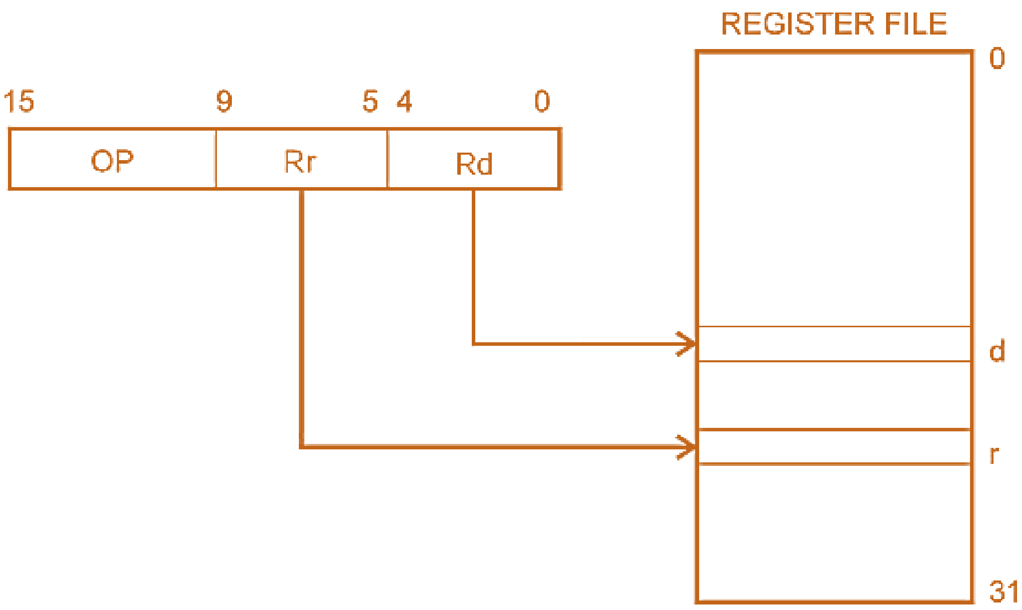


LDI R19,0x25

Two-register addressing mode

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

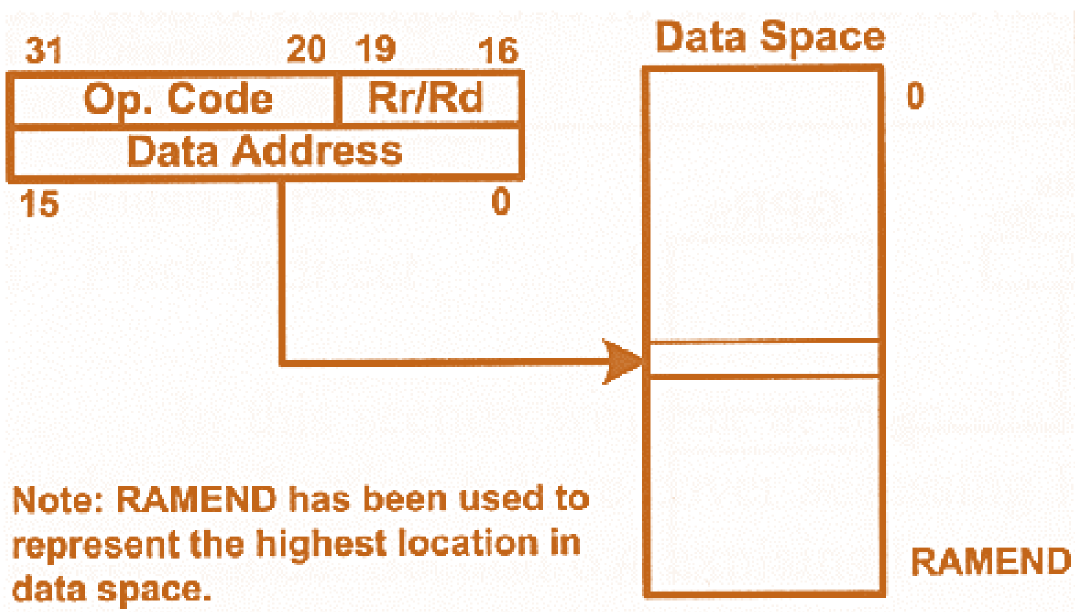
```
ADD    R20,R23          ;add R23 to R20
SUB    R29,R20          ;subtract R20 from R29
AND    R16,R17          ;AND R16 with 0x40
MOV    R23,R19          ;copy the contents of R19 to R23
```



Direct Data 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.

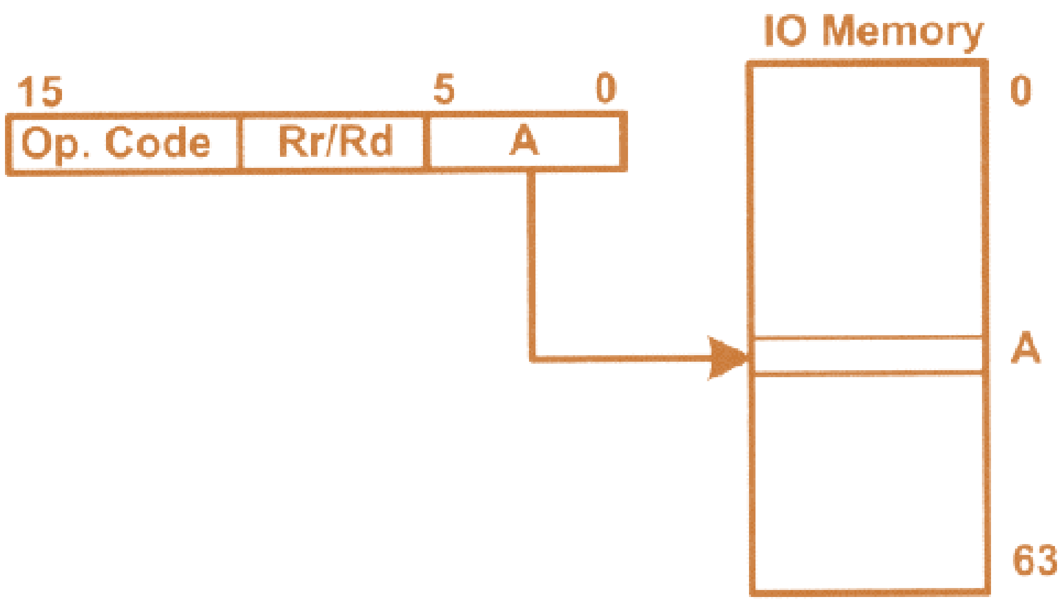
```
LDS    R19,0x560   ;load R19 with the contents of memory loc $560
STS    0x40,R19    ;store R19 to data space location 0x40
```



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.

```
IN     R18,0x16    ;R18 = contents of location $16 (PINB)
OUT    0x15,R18    ;PORTC (location $15) = R18
```



I/O Direct addressing mode

Selected ATmega32 I/O Register Addresses

Symbol	Name	I/O Address	Data Memory Addr.
PIND	Port D input pins	$10	$30
DDRD	Data Direction, Port D	$11	$31
PORTD	Port D data register	$12	$32
PINC	Port C input pins	$13	$33
DDRC	Data Direction, Port C	$14	$34
PORTC	Port C data register	$15	$35
PINB	Port B input pins	$16	$36
DDRB	Data Direction, Port B	$17	$37
PORTB	Port B data register	$18	$38
PINA	Port A input pins	$19	$39
DDRA	Data Direction, Port A	$1A	$3A
PORTA	Port A data register	$1B	$3B
SPL	Stack Pointer, Low byte	$3D	$5D
SPH	Stack Pointer, High byte	$3E	$5E

I/O Direct addressing mode

Some AVRs have less than 64 I/O registers. So, some locations of the standard I/O memory are not used by the I/O registers. The unused locations are reserved and must not be used by the AVR programmer.

Some AVRs have more than 64 I/O registers. The extra I/O registers are located above the data memory address $5F. The data memory allocated to the extra I/O registers is called *extended I/O memory*.
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. For example, the following instruction causes an error, since the I/O address must be between 0 and $3F:

```
OUT 0x65,R19          ;illegal as the address is above $3F
```

I/O Direct addressing mode

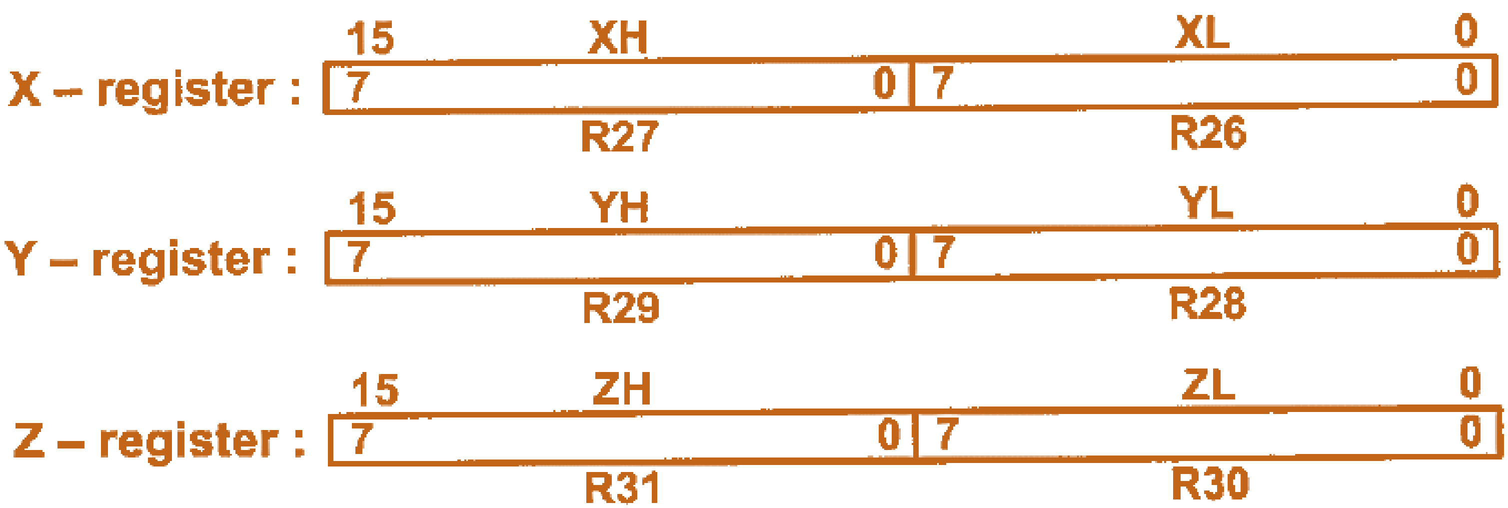
To access the extended I/O registers we can use the direct addressing mode. For example, in ATmega128, PORTF has the memory address of 0x62. So, the following instruction stores the contents of R20 in PORTF.

```
STS 0x62,R20          ;PORTF = R20
```

The I/O registers can have different addresses in different AVR microcontrollers. For example, the I/O address $2 is assigned to TWAR in the ATmega32, while the same address is assigned to DDRE in ATmega128. This means that in ATmega32, the instruction "`OUT 0x2,R20`" copies the contents of R20 to TWAR, while the same instruction, in ATmega128, copies the contents of R20 to DDRE. In other words, the same instruction can have different meanings in different AVR microcontrollers. This can cause problems if you want to run programs written for one AVR on another AVR. For example, if you have written a code for ATmega32 and you want to run it on an ATmega128, it might be necessary to change some register locations before loading it into the ATmega128.

Register Indirect addressing mode

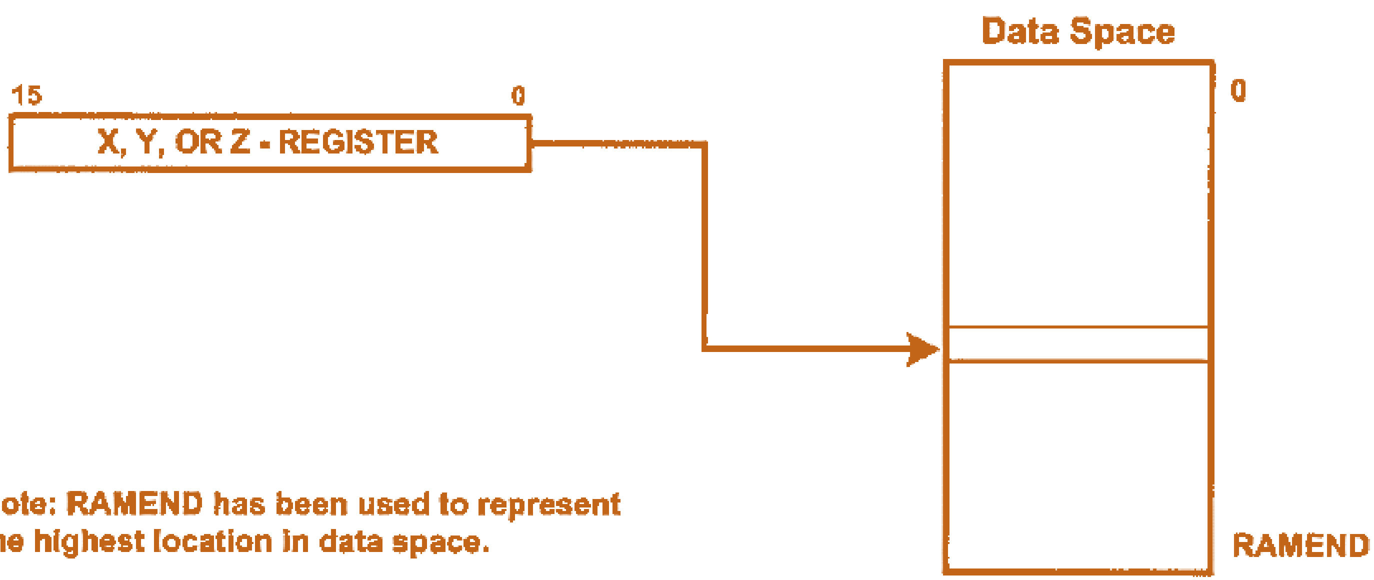
In the register indirect addressing mode, a register is used as a pointer to the data memory location. In the AVR, three registers are used for this purpose: X, Y, and Z. These are 16-bit registers allowing access to the entire 65,536 bytes of data memory space in the AVR.



Register Indirect addressing mode

```
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    X, R23      ;store the contents of location 0x139F in R23
```



Note: RAMEND has been used to represent the highest location in data space.

Register Indirect addressing mode

One of the advantages of register indirect addressing mode is that it makes accessing data dynamic rather than static, as with direct addressing mode.

```
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
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
```

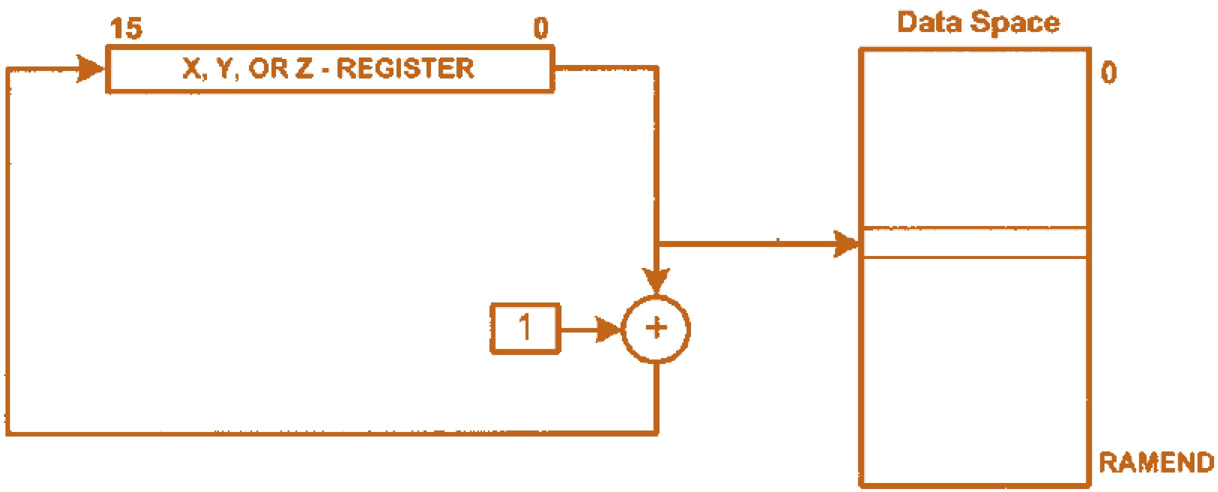
Register Indirect addressing mode

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.

AVR Auto-Increment/Decrement of Pointer Registers for LD Instruction

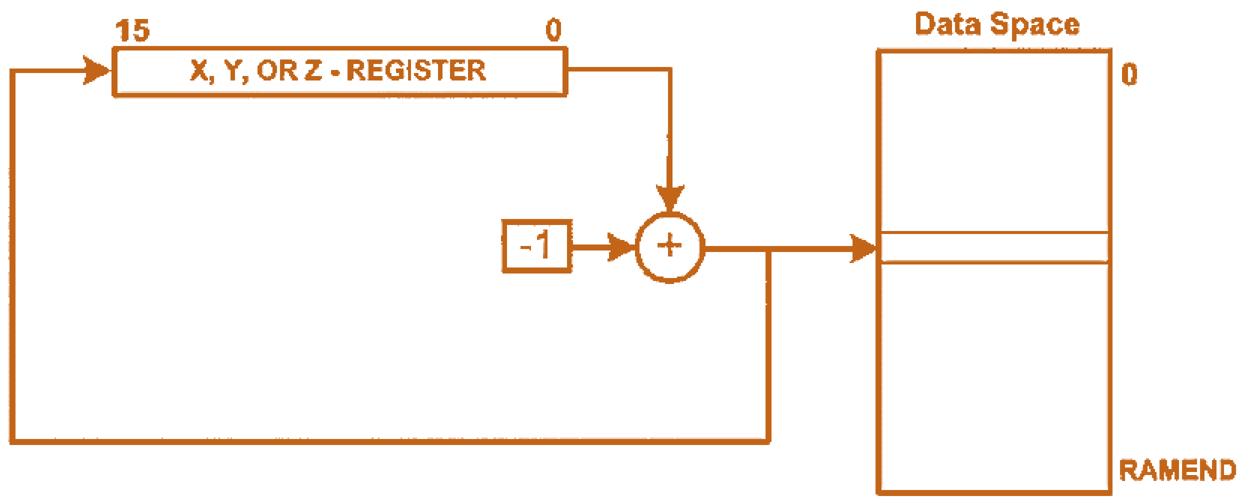
Instruction		Function
LD	Rn,X	After loading location pointed to by X, the X stays the same.
LD	Rn,X+	After loading location pointed to by X, the X is incremented.
LD	Rn,-X	The X is decremented, then the location pointed to by X is loaded.
LD	Rn,Y	After loading location pointed to by Y, the Y stays the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

Register Indirect addressing mode



LD Rn,X+

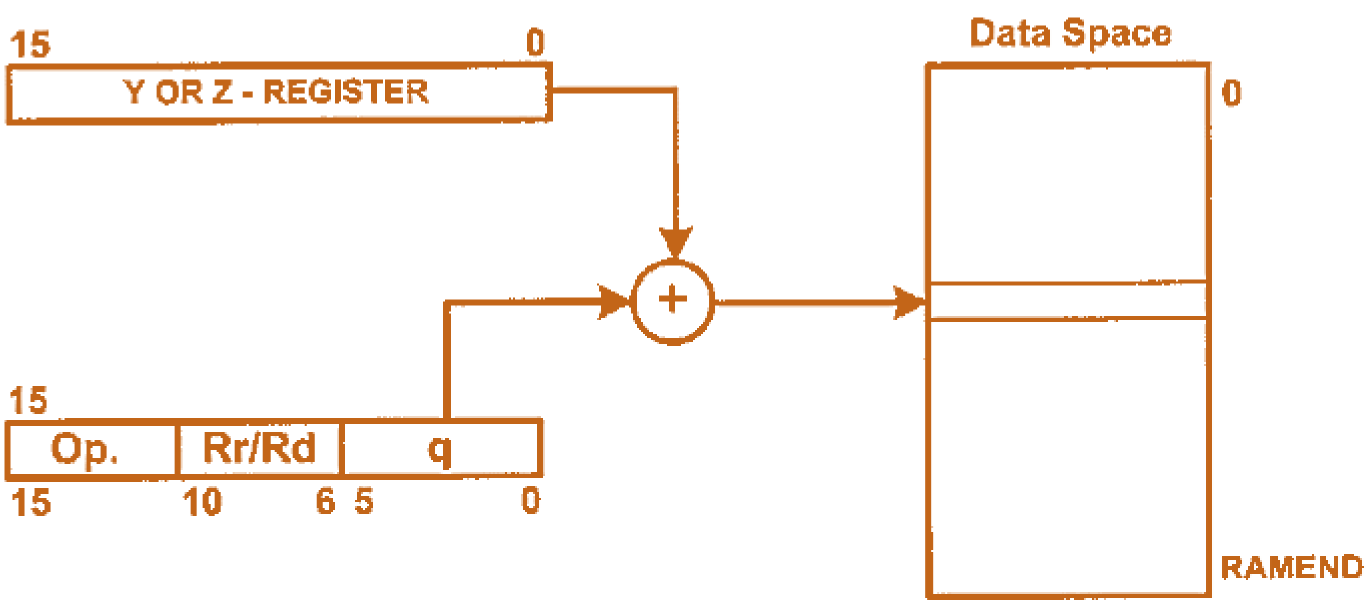
LD Rn,-X



Register Indirect with Displacement addressing mode

In this addressing mode a fixed number is added to the Z register. For example, if we want to read from the location that is 5 bytes after the location to which Z points, we can write the following instruction:

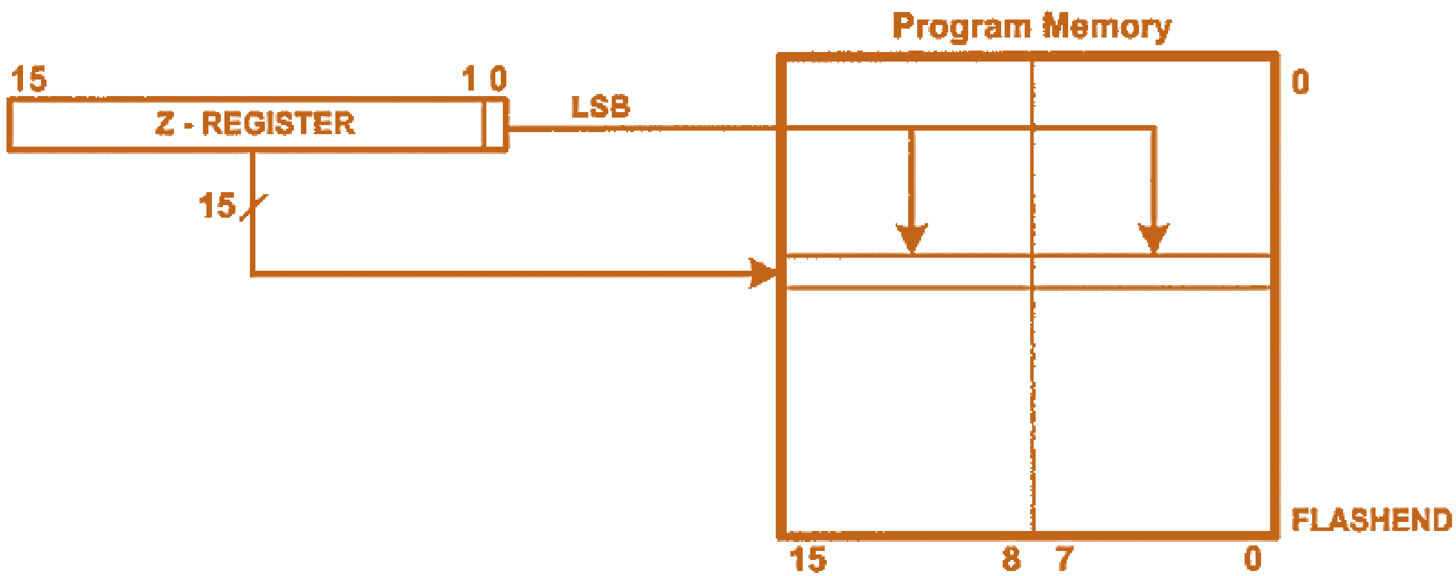
```
LDD   R20, Z+5      ;load from Z+5 into R20
```



Register Indirect Flash addressing mode

AVR Table Read Instructions

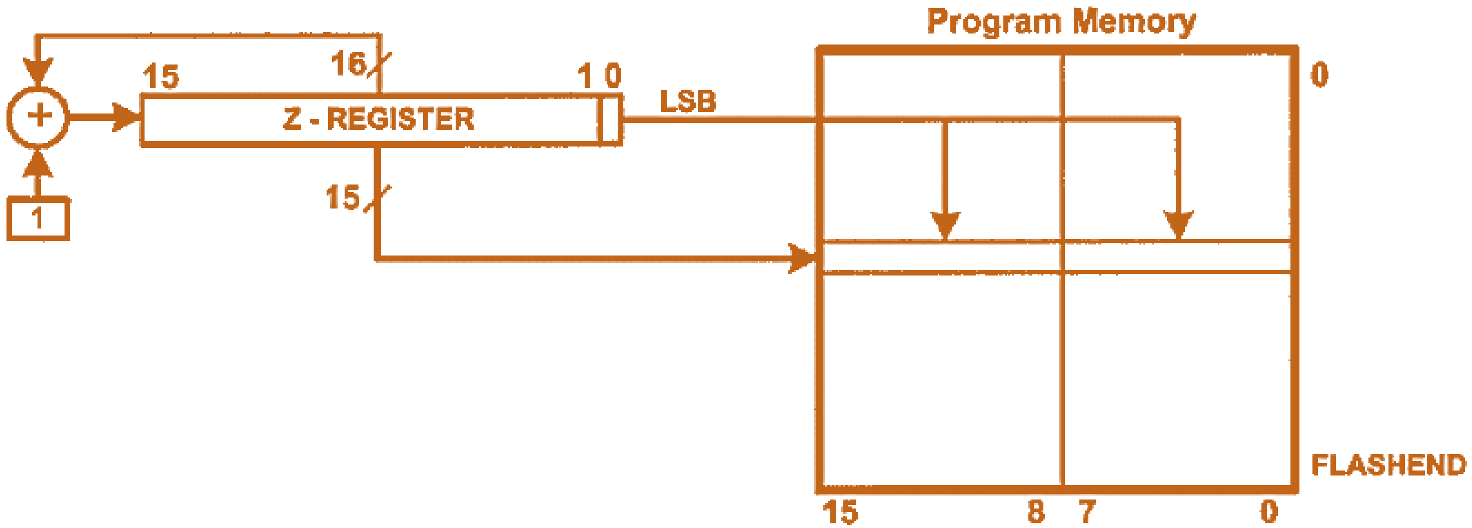
Instruction	Function	Description
LPM Rn,Z	Load from Program Memory	After read, Z stays the same
LPM Rn,Z+	Load from Program Memory with post-inc.	Reads and increments Z



Note: If LSB = 0, the low byte is selected; if LSB = 1, the high byte is selected. Bits 15 through 1 are for word address.

Register Indirect Flash addressing mode

Using the "INC ZL" instruction to increment the pointer can cause a problem when an address such as $5FF is incremented. The carry will not propagate into ZH. The AVR gives us the option of LPM Rn, Z+ (load program memory with post-increment).



Look-up Table

Assume that the lower three bits of Port C are connected to three switches. Write a program to send the following ASCII characters to Port D based on the status of the switches.

000	'0'
001	'1'
010	'2'
011	'3'
100	'4'
101	'5'
110	'6'
111	'7'

LPM – Load Program Memory

- Loads one byte pointed to by the Z-register into the destination register Rd.

	Operation:	Comment:
(i)	R0 ← (Z)	Z: Unchanged, R0 implied destination register
(ii)	Rd ← (Z)	Z: Unchanged
(iii)	Rd ← (Z) Z ← Z + 1	Z: Post incremented

	Syntax:	Operands:	Program Counter:
(i)	LPM	None, R0 implied	PC ← PC + 1
(ii)	LPM Rd, Z	$0 \le d \le 31$	PC ← PC + 1
(iii)	LPM Rd, Z+	$0 \le d \le 31$	PC ← PC + 1

LPM – Load Program Memory

16-bit Opcode:

(i)	1001	0101	1100	1000
(ii)	1001	000d	dddd	0100
(iii)	1001	000d	dddd	0101

Status Register (SREG) and Boolean Formula

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Words	1 (2 bytes)
Cycles	3

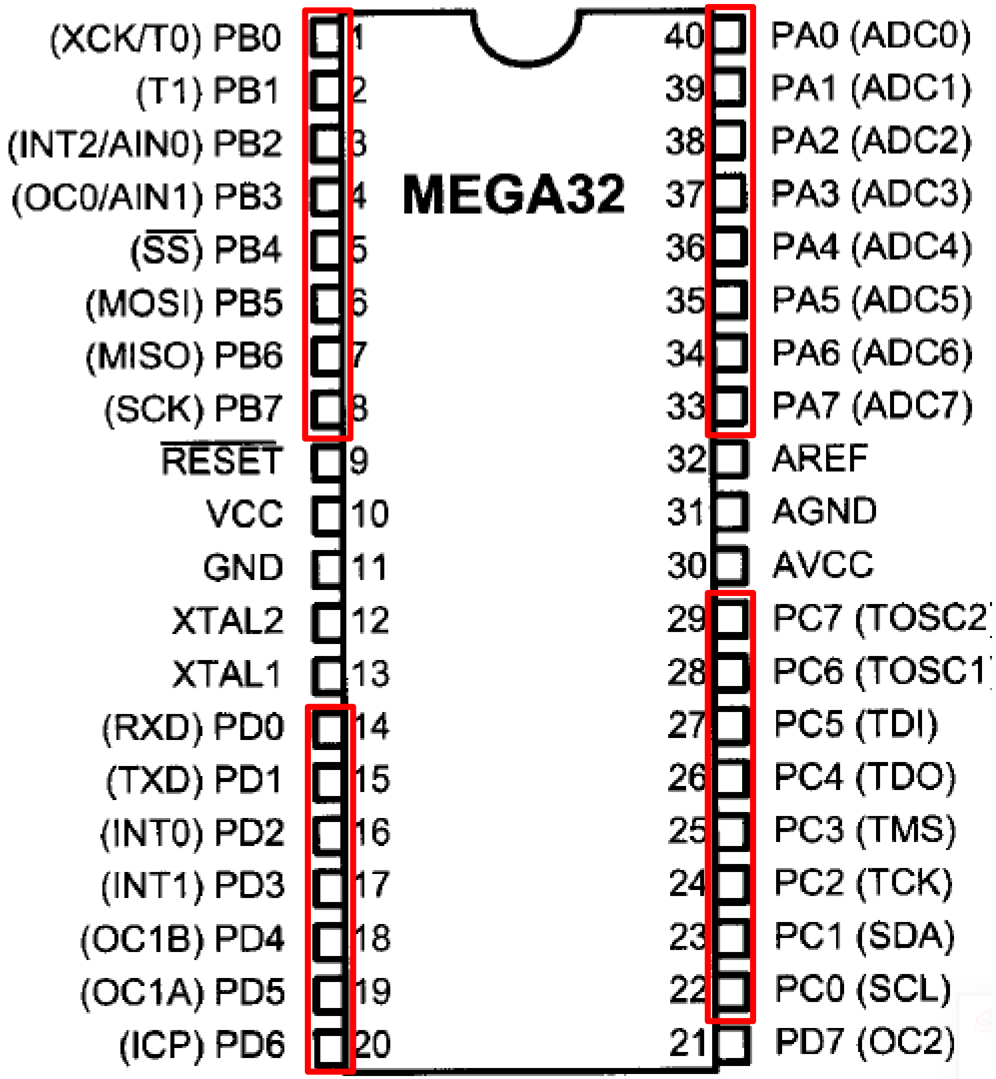
Look-up Table

000	‘0’
001	‘1’
010	‘2’
011	‘3’
100	‘4’
101	‘5’
110	‘6’
111	‘7’

```
.ORG 0
.INCLUDE "M32DEF.INC"
      LDI   R16,0x0
      OUT   DDRC,R16                 ;DDRC = 0x00 (port C as input)
      LDI   R16,0xFF
      OUT   DDRD,R16                 ;DDRD = 0xFF (port D as output)
      LDI   ZH,HIGH(ASCII_TABLE<<1)  ;ZH = high byte of addr.
BEGIN:IN    R16,PINC                 ;read from port C into R16
      ANDI  R16,0b00000111           ;mask upper 5 bits
      LDI   ZL,LOW(ASCII_TABLE<<1)   ;ZL = the low byte of addr.
      ADD   ZL,R16                   ;add PINC to the addr
      LPM   R17,Z                    ;get ASCII from look-up table
      OUT   PORTD,R17
      RJMP  BEGIN

;look-up table for ASCII numbers 0-7
.ORG 0x20
ASCII_TABLE:
      .DB '0','1','2','3','4','5','6','7'
```

I/O Port Programming



I/O Port Programming

Number of Ports in Some AVR Family Members

Pins	**8-pin**	**28-pin**	**40-pin**	**64-pin**	**100-pin**
Chip	**ATtiny25/45/85**	**ATmega8/48/88**	**ATmega32/16**	**ATmega64/128**	**ATmega1280**
Port A			X	X	X
Port B	6 bits	X	X	X	X
Port C		7 bits	X	X	X
Port D		X	X	X	X
Port E				X	X
Port F				X	X
Port G				5 bits	6 bits
Port H					X
Port J					X
Port K					X
Port L					X

Note: X indicates that the port is available.

I/O Port Programming

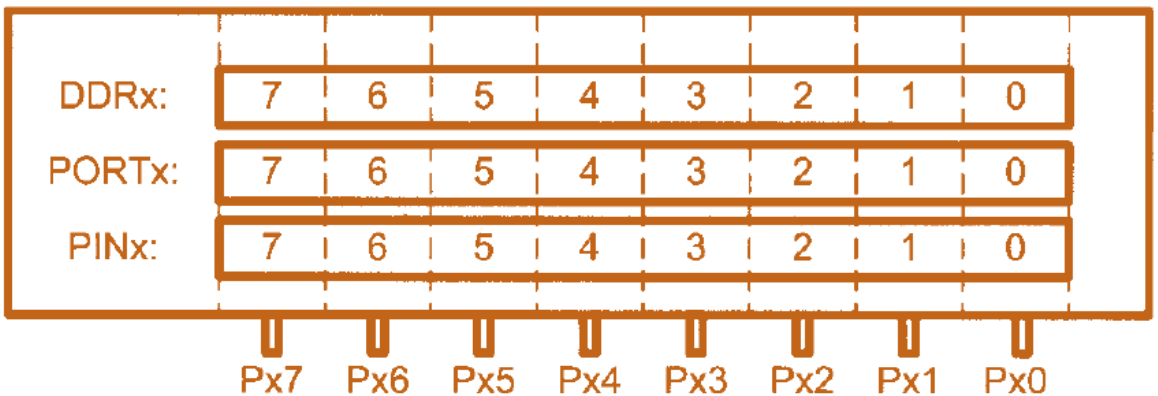
To use any of these ports as an input or output port, it must be programmed.

Each port has three I/O registers

PORTx, DDRx, and PINx.

PORTB, DDRB, and PINB.

Port	Address	Usage
PORTA	$3B	output
DDRA	$3A	direction
PINA	$39	input
PORTB	$38	output
DDRB	$37	direction
PINB	$36	input
PORTC	$35	output
DDRC	$34	direction
PINC	$33	input
PORTD	$32	output
DDRD	$31	direction
PIND	$30	input



DDR Register Role

Each of the ports A–D in the ATmega32 can be used for input or output. The DDRx I/O register is used solely for the purpose of making a given port an input or output port. For example, to make a port an output, we write 1s to the DDRx register. In other words, to output data to all of the pins of the Port B, we must first put 0b11111111 into the DDRB register to make all of the pins output.

To make a port an input port, we must first put 0s into the DDRx register for that port, and then bring in (read) the data present at the pins. As an aid for remembering that the port is input when the DDR bits are 0s, imagine a person who has 0 dollars. The person can only get money, not give it. Similarly, when DDR contains 0s, the port gets data.

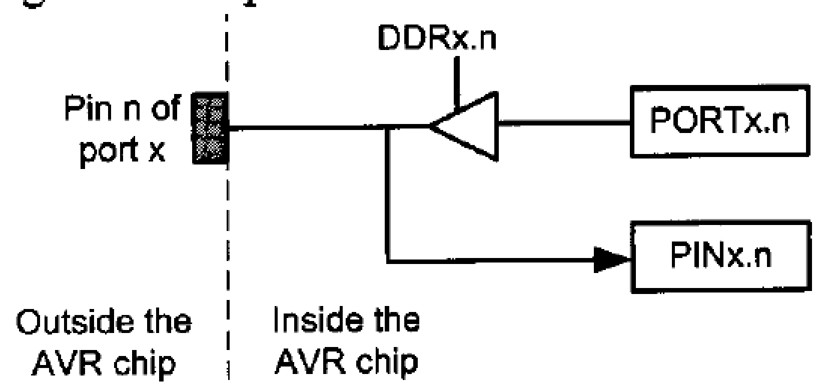
Notice that upon reset, all ports have the value 0x00 in their DDR registers. This means that all ports are configured as input .

DDR Register Role

Each of the ports A–D in the ATmega32 can be used for input or output. The DDRx I/O register is used solely for the purpose of making a given port an input or output port. For example, to make a port an output, we write 1s to the DDRx register. In other words, to output data to all of the pins of the Port B, we must first put 0b11111111 into the DDRB register to make all of the pins output.

To make a port an input port, we must first put 0s into the DDRx register for that port, and then bring in (read) the data present at the pins. As an aid for remembering that the port is input when the DDR bits are 0s, imagine a person who has 0 dollars. The person can only get money, not give it. Similarly, when DDR contains 0s, the port gets data.

Notice that upon reset, all ports have the value 0x00 in their DDR registers. This means that all ports are configured as input .

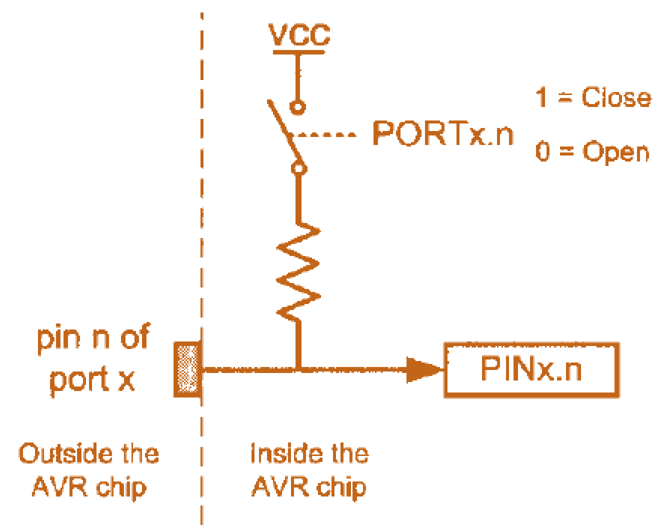


Port Register Role

To read the data present at the pins, we should read the PIN register. It must be noted that to bring data into CPU from pins we read the contents of the PINx register, whereas to send data out to pins we use the PORTx register.

There is a pull-up resistor for each of the AVR pins. If we put 1s into bits of the PORTx register, the pull-up resistors are activated. In cases in which nothing is connected to the pin or the connected devices have high impedance, the resistor pulls up the pin.

If we put 0s into the bits of the PORTx register, the pull-up resistor is inactive.



Port Register Role

The pins of the AVR microcontrollers can be in four different states according to the values of PORTx and DDRx .

PORTx \ DDRx	0	1
0	Input & high impedance	Out 0
1	Input & pull-up	Out 1

پایان

موفق و پیروز باشید