CHW 469 : Embedded Systems

Instructor:
Dr. Ahmed Shalaby
http://bu.edu.eg/staff/ahmedshalaby14#

I/O Ports in AVR

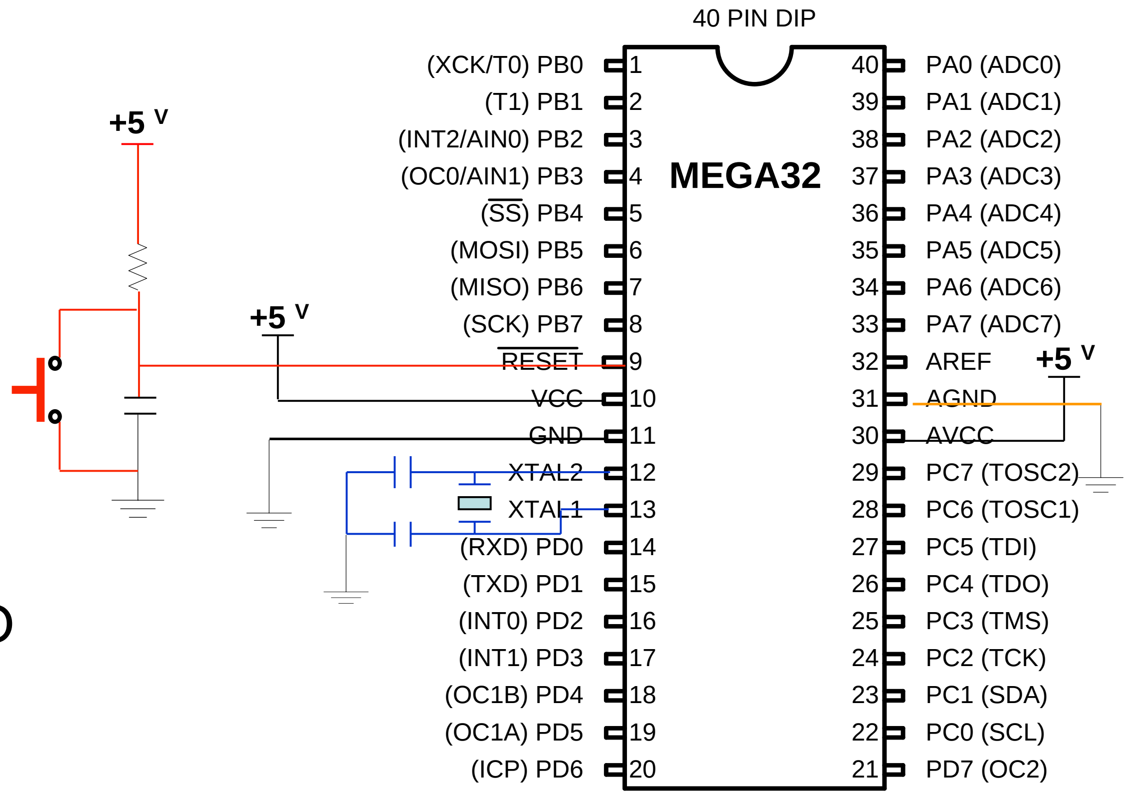
The AVR microcontroller and embedded systems
using assembly and c

Topics

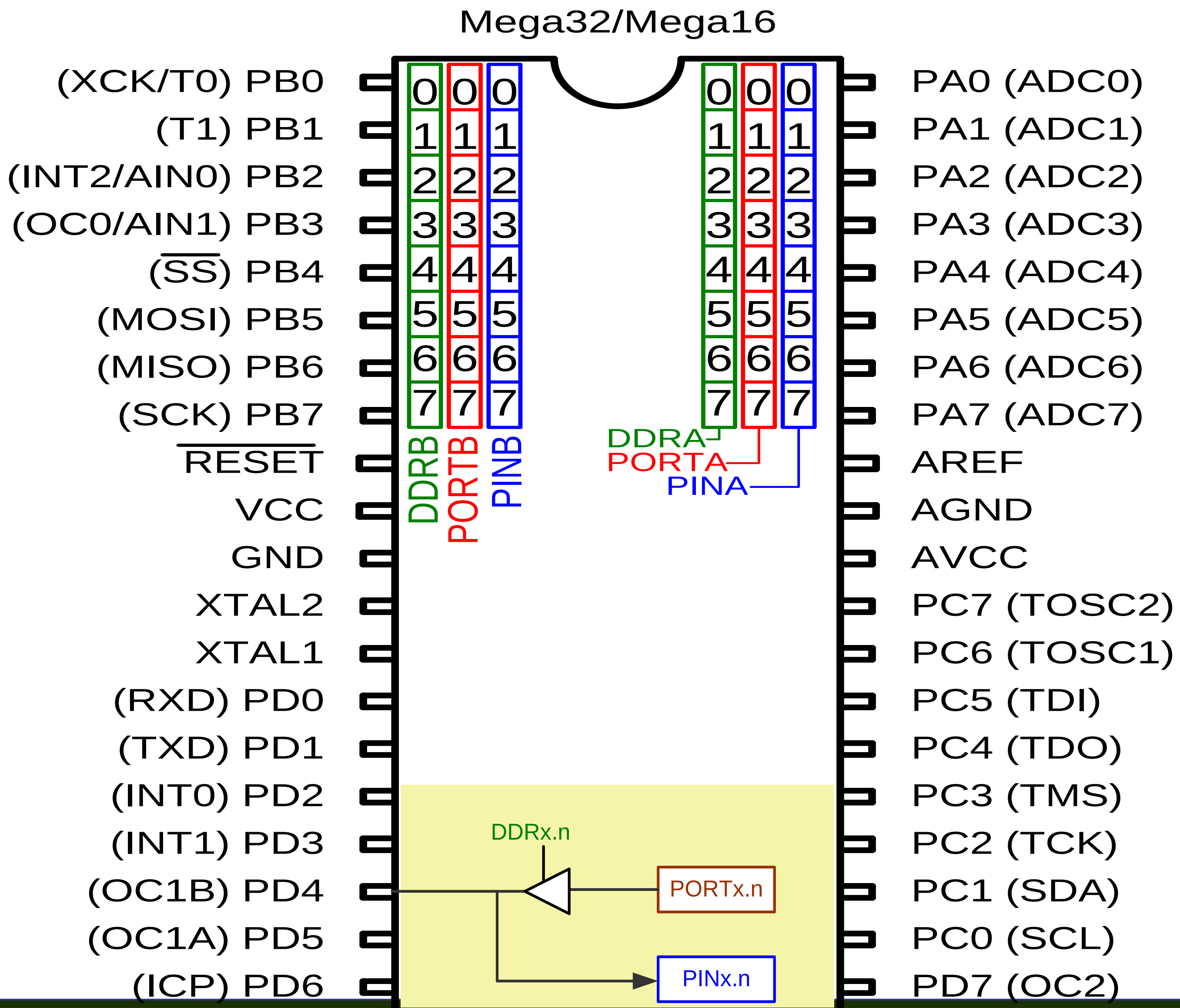
- AVR pin out
- The structure of I/O pins
- I/O programming
- Bit manipulating

ATmega16/mega32 pinout

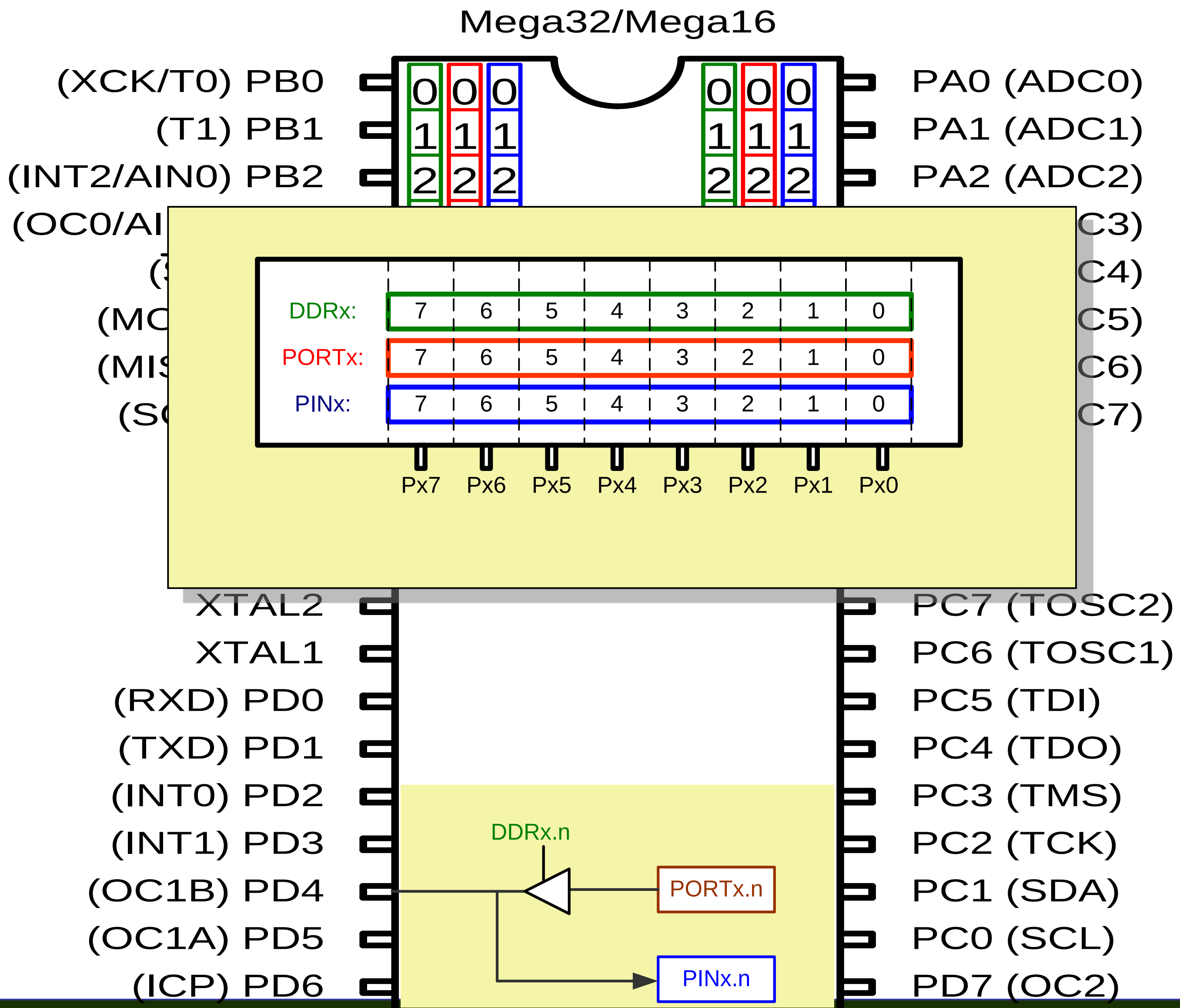
1. Vital Pins:
   1. Power
      - VCC
      - Ground
   2. Crystal
      - XTAL1
      - XTAL2
   3. Reset
2. I/O pins
   - PORTA, PORTB, PORTC, and PORTD
3. Internal ADC pins
   - AREF, AGND, AVCC



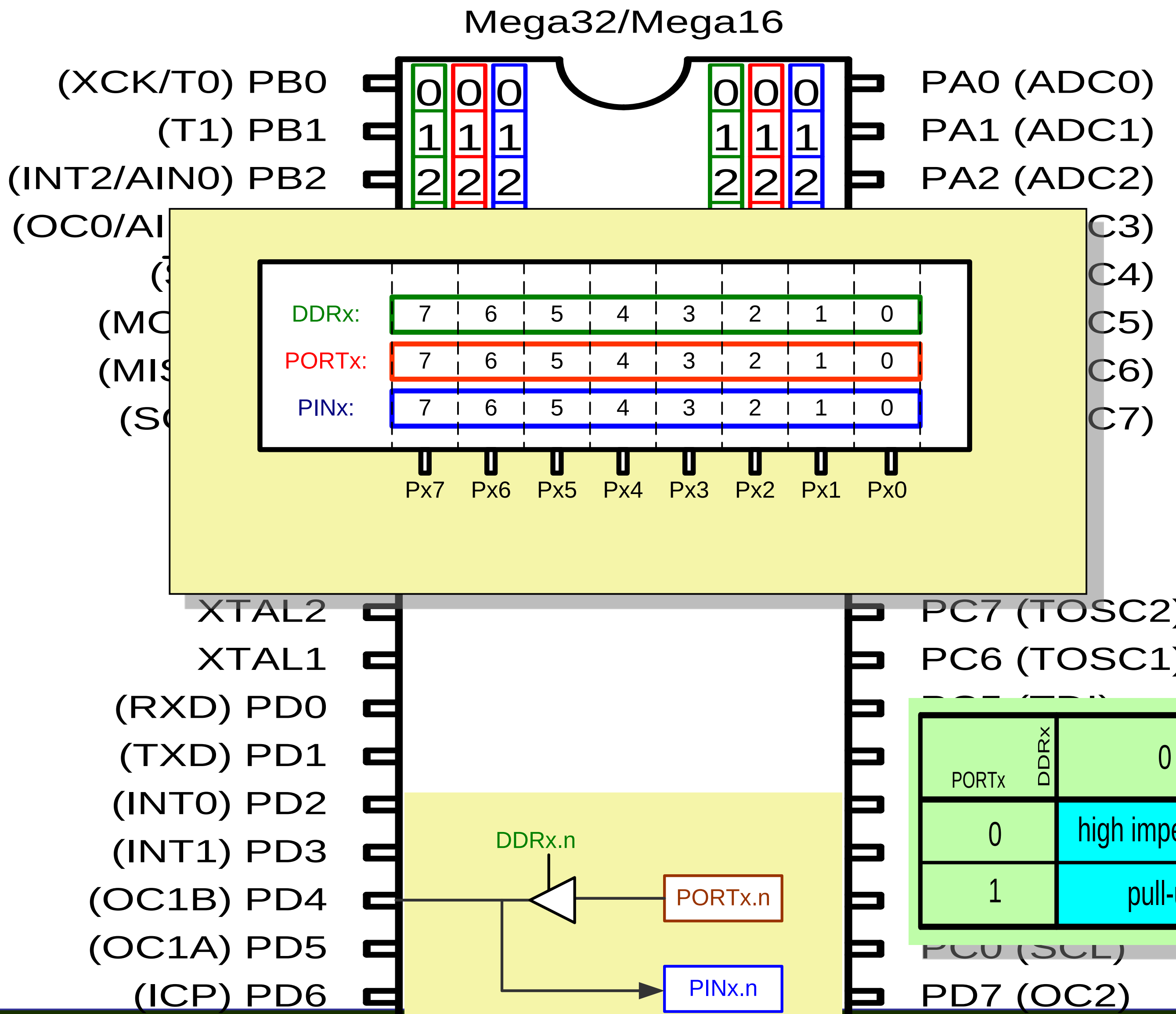
The structure of IO pins



The structure of IO pins



The structure of IO pins

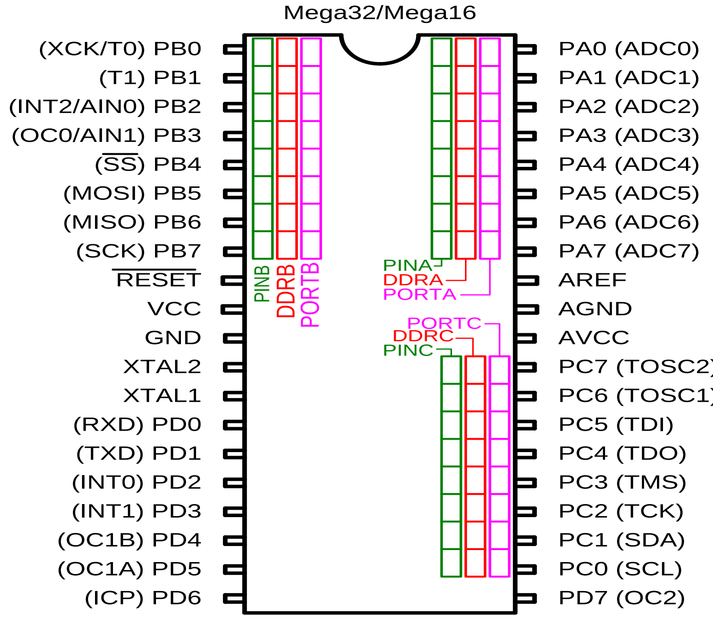


PORTx \ DDRx	0	1
0	high impedance	Out 0
1	pull-up	Out 1

Example 1

- Write a program that makes all the pins of PORTA one.

DDRA:	1	1	1	1	1	1	1	1
PORTA:	1	1	1	1	1	1	1	1




```
.INCLUDE "M32DEF.INC"
LDI   R20,0xFF   ;R20 = 11111111 (binary)
OUT   PORTA,R20 ;PORTA = R20
OUT   DDRA,R20   ;DDRA = R20
```

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

Example 2

- The following code will toggle all 8 bits of Port B forever with some time delay between “on” and “off” states:

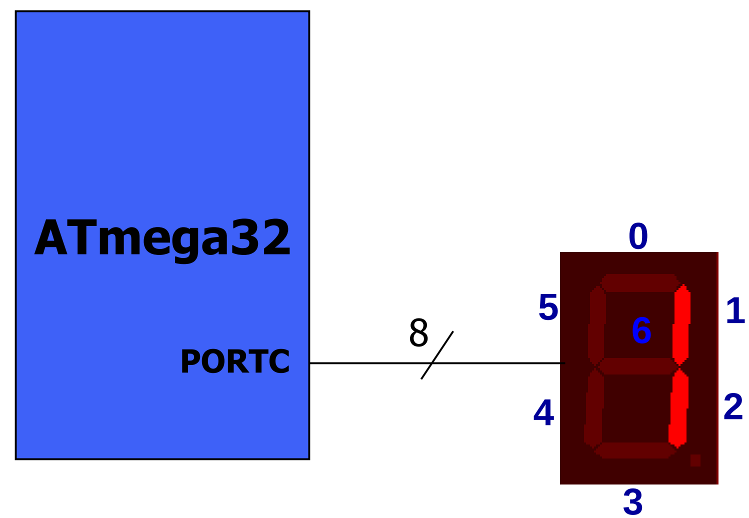
```
    LDI   R16,0xFF     ;R16 = 0xFF = 0b11111111
    OUT   DDRB,R16     ;make Port B an output port (1111 1111)
L1: LDI   R16,0x55     ;R16 = 0x55 = 0b01010101
    OUT   PORTB,R16    ;put 0x55 on port B pins
    CALL  DELAY
    LDI   R16,0xAA     ;R16 = 0xAA = 0b10101010
    OUT   PORTB,R16    ;put 0xAA on port B pins
    CALL DELAY
    RJMP L1
```

Example 3

- A 7-segment is connected to PORTA. Display 1 on the 7-segment.

DDRC	1	1	1	1	1	1	1	1
PORTC:	0	0	0	0	0	1	1	0

```
    .INCLUDE "M32DEF.INC"
    LDI   R20,0x06   ;R20 = 00000110 (binary)
    OUT   PORTC,R20  ;PORTC = R20
    LDI   R20,0xFF   ;R20 = 11111111 (binary)
    OUT   DDRC,R20   ;DDRC = R20
L1: RJMP L1
```



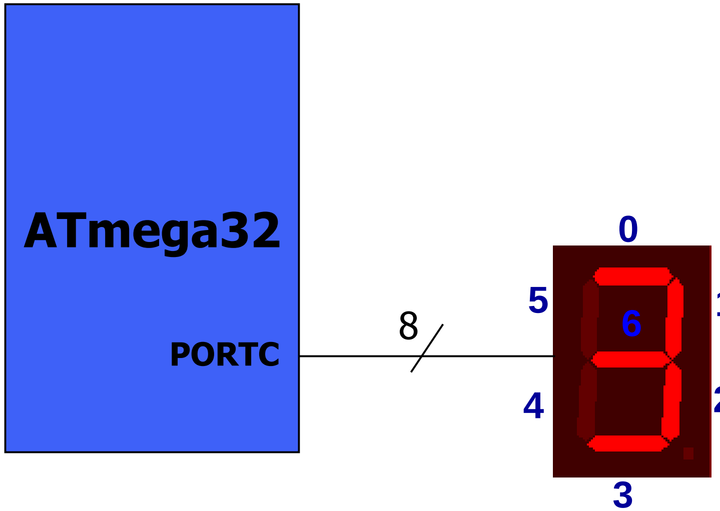
PORTx \ DDRx	0	1
0	high impedance	Out 0
1	pull-up	Out 1

Example 4

- A 7-segment is connected to PORTA. Display 3 on the 7-segment.

DDR:	1	1	1	1	1	1	1	1
PORTC:	0	1	0	0	1	1	1	1

```
    .INCLUDE "M32DEF.INC"
    LDI   R20,0x4F   ;R20 = 01001111 (binary)
    OUT   PORTC,R20  ;PORTC = R20
    LDI   R20,0xFF   ;R20 = 11111111 (binary)
    OUT   DDRC,R20   ;DDRC = R20
L1: RJMP L1
```



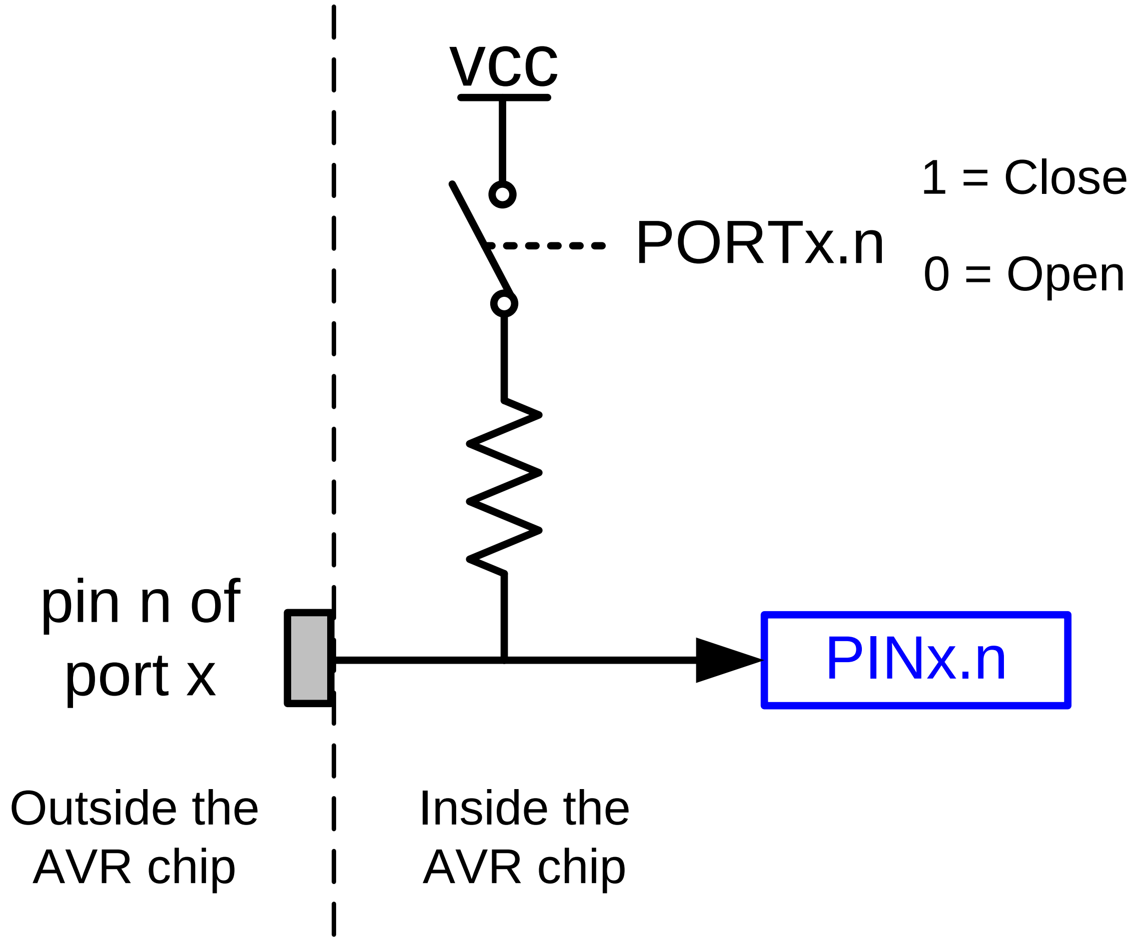
PORTx \ DDRx	0	1
0	high impedance	Out 0
1	pull-up	Out 1

Example 5: Input

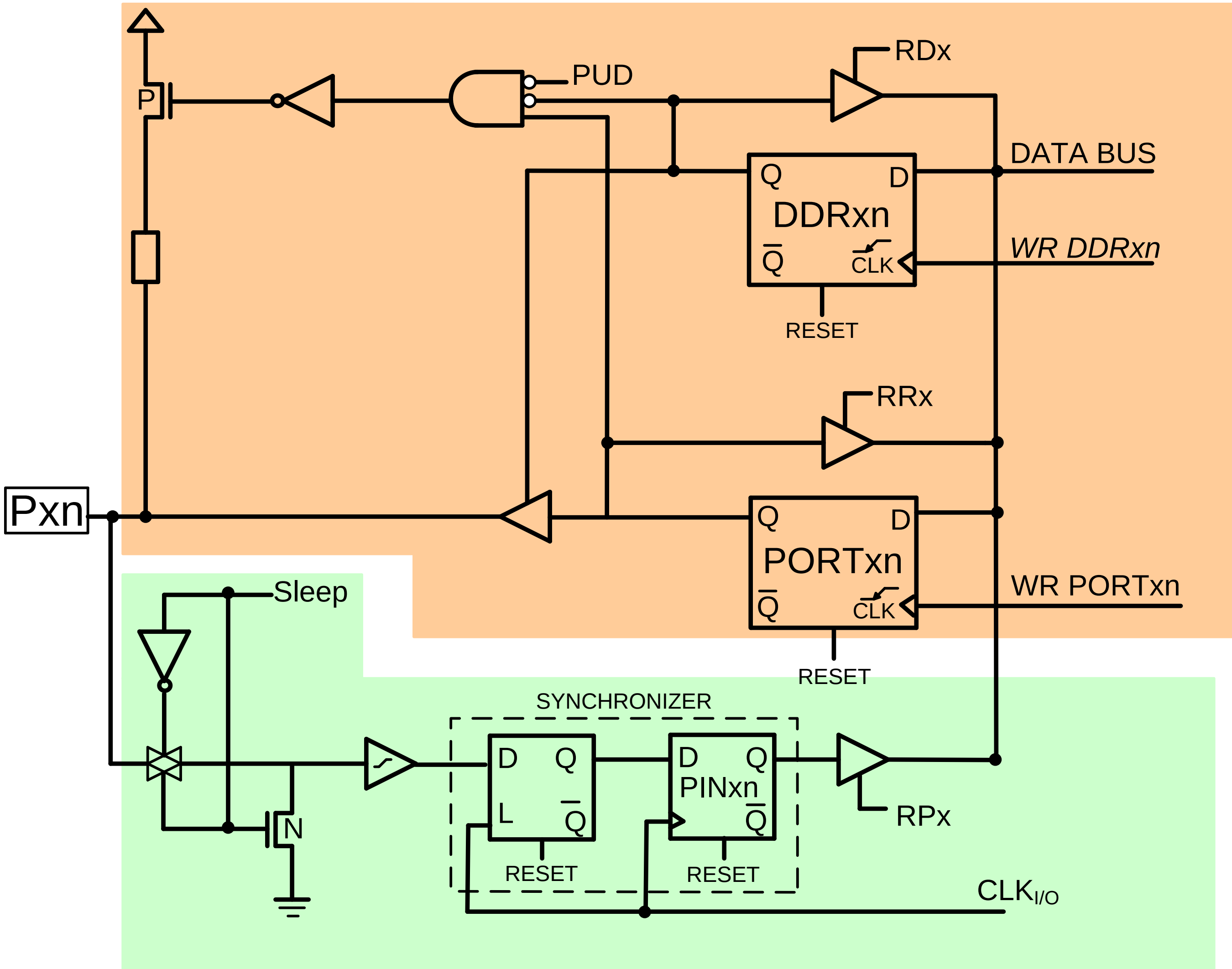
- The following code gets the data present at the pins of port C and sends it to port B indefinitely, after adding the value 5 to it:

```
        .INCLUDE "M32DEF.INC"
        LDI     R16,0x00        ;R16 = 00000000 (binary)
        OUT     DDRC,R16        ;make Port C an input port
        LDI     R16,0xFF        ;R16 = 11111111 (binary)
        OUT     DDRB,R16        ;make Port B an output port(1 for Out)
L2:     IN      R16,PINC        ;read data from Port C and put in R16
        LDI     R17,5
        ADD     R16,R17         ;add 5 to it
        OUT     PORTB,R16       ;send it to Port B
        RJMP    L2              ;continue forever
```

Pull-up resistor



The structure of I/O pins



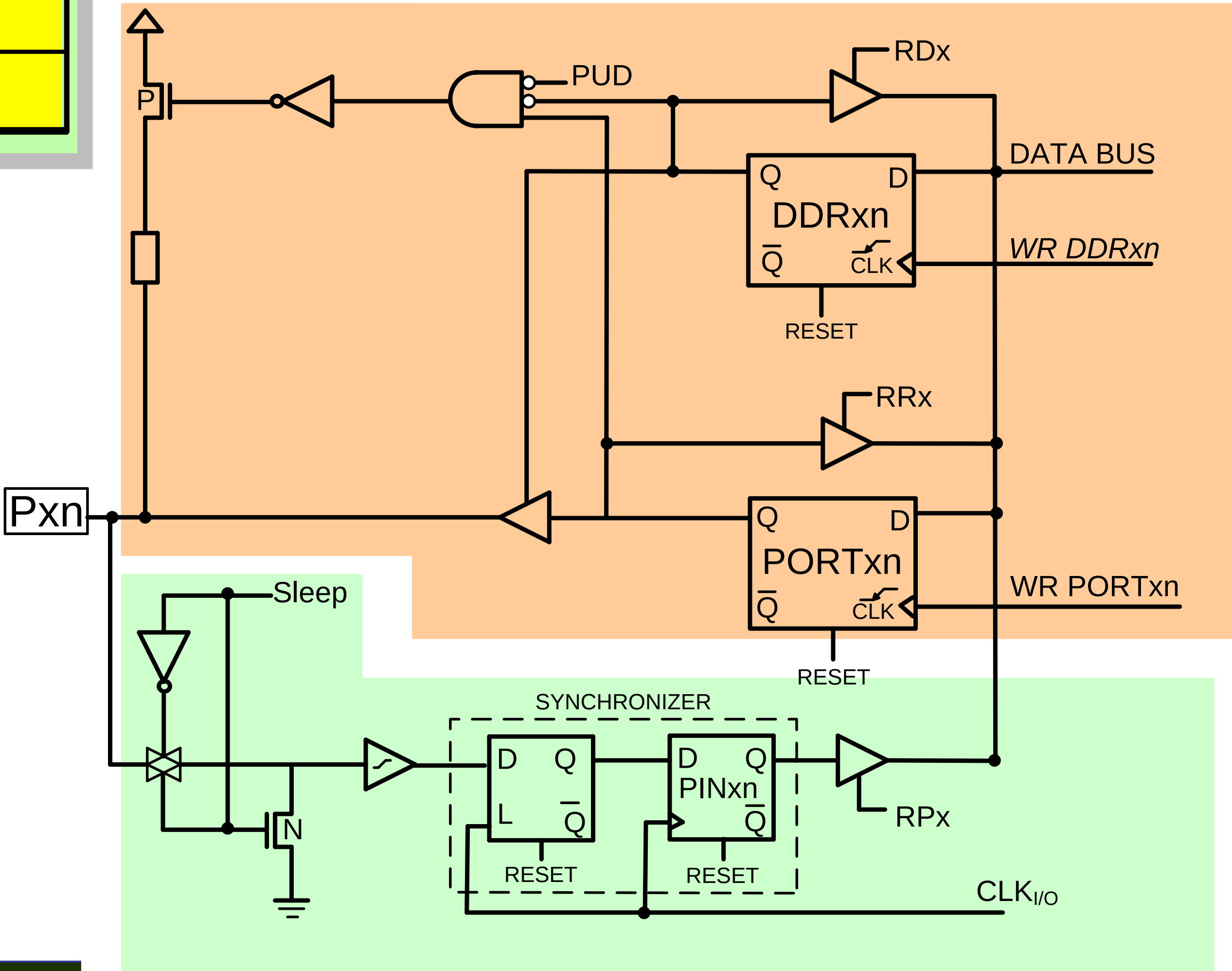
Example 6

- Write a program that continuously sends out to Port C the alternating values of 0x55 and 0xAA.

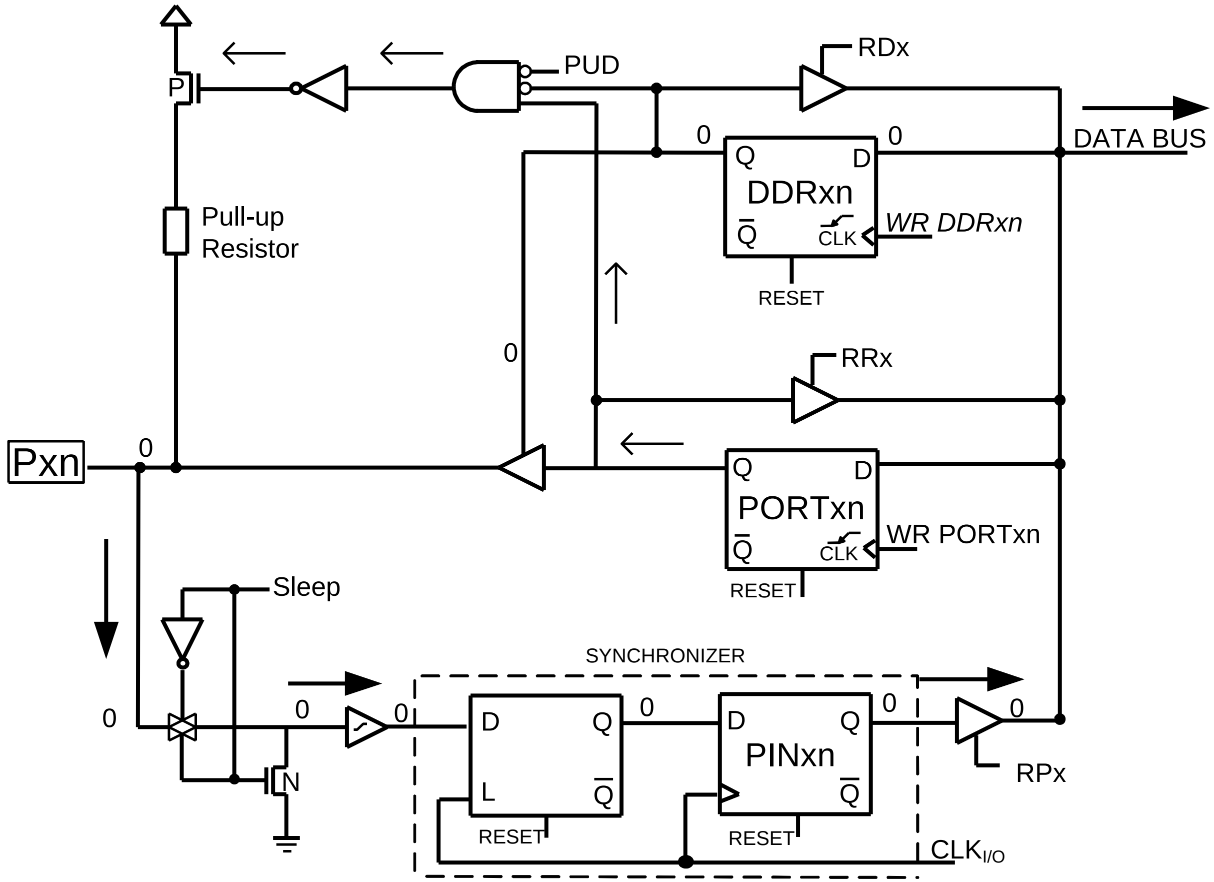
```
     .INCLUDE "M32DEF.INC"
     LDI   R16,0xFF     ;R16 = 11111111 (binary)
     OUT   DDRC,R16     ;make Port C an output port
L1:  LDI   R16,0x55     ;R16 = 0x55
     OUT   PORTC,R16    ;put 0x55 on Port C pins
     LDI   R16,0xAA     ;R16 = 0xAA
     OUT   PORTC,R16    ;put 0xAA on Port C pins
     RJMP  L1
```

The structure of IO pins

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



Input (Tri-state vs. pull up)

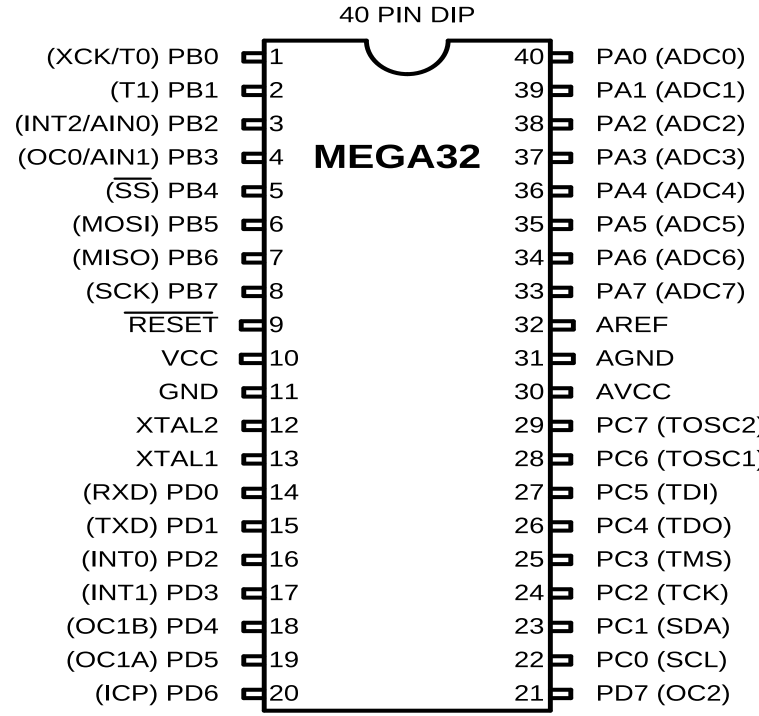


The ⟵ represents how the content of PORTx register affects the pull-up resistor;
while the ⟶ shows how a data can be read from a pin

Example 7

- Write a program that reads from port A and writes it to port B.

```
      .INCLUDE "M32DEF.INC"
      LDI   R20,0x0    ;R20 = 00000000 (binary)
      OUT   DDRA,R20   ;DDRA = R20
      LDI   R20,0xFF   ;R20 = 11111111 (binary)
      OUT   DDRB,R20   ;DDRB = R20
L1:   IN    R20,PINA   ;R20 = PINA
      OUT   PORTB,R20  ;PORTB = R20
      RJMP  L1
```



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

I/O bit manipulation programming

SBI and CBI instructions

- SBI (Set Bit in IO register)
  - SBI ioReg, bit ;ioReg.bit = 1
  - Examples:
    - SBI PORTD,0 ;PORTD.0 = 1
    - SBI DDRC,5 ;DDRC.5 = 1
- CBI (Clear Bit in IO register)
  - CBI ioReg, bit ;ioReg.bit = 0
  - Examples:
    - CBI PORTD,0 ;PORTD.0 = 0
    - CBI DDRC,5 ;DDRC.5 = 0

Example

- Write a program that toggles PORTA.4 continuously.

```
     .INCLUDE "M32DEF.INC"
     SBI  DDRA,4
L1:  SBI  PORTA,4
     CBI  PORTA,4
     RJMP L1
```

Example

- An LED is connected to each pin of Port D. Write a program to turn on each LED from pin D0 to pin D7. Call a delay module before turning on the next LED.

```
.INCLUDE "M32DEF.INC"
        LDI     R20, 0xFF
        OUT     DDRD, R20        ;make PORTD an output port
        SBI     PORTD,0          ;set bit PD0
        CALL    DELAY            ;delay before next one
        SBI     PORTD,1          ;turn on PD1
        CALL    DELAY            ;delay before next one
        SBI     PORTD,2          ;turn on PD2
        CALL    DELAY
        SBI     PORTD,3
        CALL    DELAY
        SBI     PORTD,4
        CALL    DELAY
        SBI     PORTD,5
        CALL    DELAY
        SBI     PORTD,6
        CALL    DELAY
        SBI     PORTD,7
        CALL    DELAY
```

SBIC and SBIS

- SBIC (Skip if Bit in IO register Cleared)
  - SBIC ioReg, bit ; if (ioReg.bit = 0) skip next instruction
  - Example:

```
SBIC   PORTD,0   ;skip next instruction if PORTD.0=0
INC    R20
LDI    R19,0x23
```

- SBIS (Skip if Bit in IO register Set)
  - SBIS ioReg, bit ; if (ioReg.bit = 1) skip next instruction
  - Example:

```
SBIS   PORTD,0   ;skip next instruction if PORTD.0=1
INC    R20
LDI    R19,0x23
```

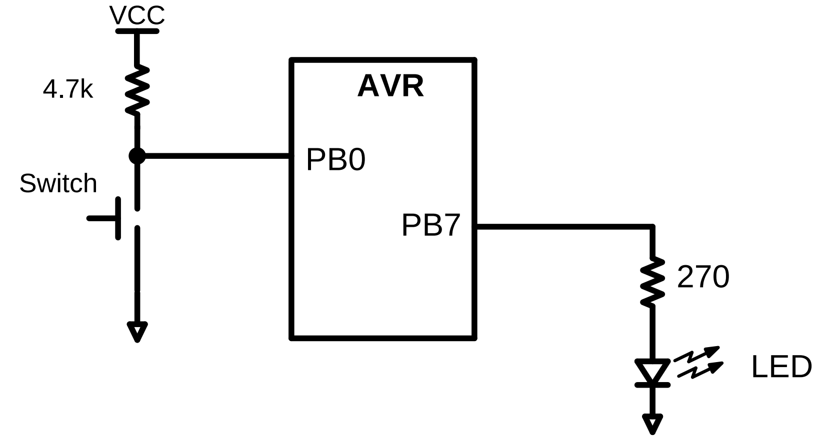
Example

- Write a program to perform the following:
- (a) Keep monitoring the PB2 bit until it becomes HIGH;
- (b) When PB2 becomes HIGH, write value $45 to Port C, and also send a HIGH-to-LOW pulse to PD3.

```
.INCLUDE "M32DEF.INC"
        CBI  DDRB, 2        ;make PB2 an input
        SBI  PORTB,2
        LDI  R16, 0xFF
        OUT  DDRC, R16      ;make Port C an output port
        SBI  DDRD, 3        ;make PD3 an output
AGAIN:  SBIS PINB, 2        ;Skip if Bit PB2 is HIGH
        RJMP AGAIN          ;keep checking if LOW
        LDI  R16, 0x45
        OUT  PORTC, R16     ;write 0x45 to port C
        SBI  PORTD, 3       ;set bit PD3 (H-to-L)
        CBI  PORTD, 3       ;clear bit PD3
HERE:   RJMP HERE
```

Example

- A switch is connected to pin PB0 and an LED to pin PB7. Write a program to get the status of SW and send it to the LED.




```
.INCLUDE "M32DEF.INC"
        CBI   DDRB,0          ;make PB0 an input
        SBI   DDRB,7          ;make PB7 an output
AGAIN:  SBIC  PINB,0          ;skip next if PB0 is clear
        RJMP  OVER            ;(JMP is OK too)
        CBI   PORTB,7
        RJMP  AGAIN           ;we can use JMP too
OVER:   SBI   PORTB,7
        RJMP  AGAIN           ;we can use JMP too
```

Arithmetic and Logic
Chapter 5

The AVR microcontroller and embedded systems

using assembly and c

Objectives

- The concept of signed numbers and 2'complement
- Addition and subtraction instructions
- Carry and overflow
- Logical instruction and masking
- Compare instruction and branching
- Shift, Rotate and Data serialization
- BCD, Packed BCD and ASCII conversion.

Some ideas

- 100 – 34 = ?
- 99 – 34 = ?
- 100 – 34 = (99-34) + 1

- 34 – 19 = ?
- 34 +100 -19 – 100 = 34 + (99-19)+1 -100

- 100000000 – 00101101 = ?
- =011111111 – 00101101 + 1 =A

- 010110110 – 00101101 =
- 010110110 + A – 100000000 =

ADD instructions

ADD Rd,Rr ;Rd = Rd + Rr (Direct or immediate are not supported)

Example 5-1

Show how the flag register is affected by the following instructions.

```
LDI   R21,0xF5    ;R21 = F5 hex
LDI   R22,0xB     ;R22 = 0xB hex
ADD   R21,R22     ;R21 = R21+R22 = F5+0B = 00 and C = 1
```

Solution:

```
   F5H          1111 0101
 + 0BH        + 0000 1011
  100H          0000 0000
```

After the addition, register R21 contains 00 and the flags are as follows:

C = 1 because there is a carry out from D7.

Z = 1 because the result in destination register (R1) is zero.

H = 1 because there is a carry from D3 to D4.

ADD instructions

ADD Rd,Rr   ;Rd = Rd + Rr   (Direct or immediate are not supported)

Example 5-2

Assume that RAM locations 400H have the value of 22H. Write a program to find the sum of location 400H of RAM and 12. At the end of the program, R21 should contain the sum.

Solution:

```
LDS   R2,0x400     ; R2 = 22H (location 0x400 of RAM)
LDI   R21,0x12     ; R21 = 12
ADD   R21,R2       ; R21 = R21 + R2 = 12H + 22H = 34H, C = 0
```

ADC instructions

```
   1
  3C  E7
  3B  8D
  ------
  78  74
```

Example 5-3

Write a program to add two 16-bit numbers. The numbers are 3CE7H and 3B8DH. Assume that R1 = (8D), R2=(3B), R3=(E7) and R4 = (3C). Place the sum in R3 and R4; R3 should have the lower byte.

Solution:

```
;R1 = (8D)
;R2 = (3B)
;R3 = (E7)
;R4 = (3C)

ADD   R3,R1      ;R3 = R3 + R1 = E7 + 8D = 74 and C = 1
ADDC  R4,R2      ;R4 = R4 + R2 + carry, adding the upper byte
                 ;with Carry from lower byte
                 ;R4 = 3C + 3B + 1 = 78H (all in hex)
```

Notice the use of ADD for the lower byte and ADDC for the higher byte.

SUB instruction

SUB Rd,Rr ;Rd = Rd - Rr (immediate are not supported)
SUB Rd,Rr ; Rd = Rd – K

Example 5-4

Show the steps involved in the following.

```
LDI   R20, 0x23          ;load 23H into R20
LDI   R21, 0x3F          ;load 3FH into R21
SUB   R21, R20           ;R21 <- R21-R20
```

Solution:

```
  R21 = 3F   0011 1111        0011 1111
- R20 = 23   0010 0011    +   1101 1101  (2's complement)
        1C                  1 0001 1100
                          C = 0, D7 = N = 0 (result is positive)
```

The flags would be set as follows: N = 0, C = 0 (notice that there is a carry but C=0. we will discuss it more in the next section) . The programmer must look at the N (or C) flag to determine if the result is positive or negative.

SBC instruction

```
SBC     Rd,Rr     ;Rd = Rd – Rr-C     ( immediate are not supported)
SBIc    Rd,Rr     ;Rd = Rd – K-C
```

```
  27 62  (H)
- 11 96  (H)
-----------
  11 CC (H)
```

```
;R26 = (62)
;R27 = (27)

LDI   R28,0x96     ;load the low byte (R28 = 96H)
LDI   R29,0x12     ;load the high byte (R29 = 12H)
SUB   R26,R28      ;R26 = R26 - R28 = 62 - 96 = CCH
                   ;C = borrow = 1, N = 1
SBC   R27,R29      ;R27 = R27 - R29 - b
                   ;R27 = 27 - 12 - 1 = 14H
```

After the SUB, R26 has = 62H – 96H = CCH and the carry flag is set to 1, indicating there is a borrow (notice, N = 1). Because C = 1, when SBC is executed the R27 has 27H – 12H – 1 = 14H. Therefore, we have 2762H – 1296H = 14CCH.

Multiplication and Division

```
MUL    Rd,Rr        ;Multiply Unsigned R1:R0 = Rd * Rr
MULS   Rd,Rr        ;Multiply Signed R1:R0 = Rd * Rr
MULSU  Rd,Rr        ;Multiply Signed with Unsigned R1:R0 =Rd*Rr
```

```
.DEF  NUM = R20
.DEF  DENOMINATOR = R21
.DEF  QUOTIENT = R22

      LDI   NUM,95              ; NUM = 95
      LDI   DENOMINATOR,10      ; DENOMINATOR = 10
      CLR   QUOTIENT            ; QUOTIENT = 0

L1:   INC   QUOTIENT
      SUB   NUM, DENOMINATOR
      BRCC  L1

      DEC   QUOTIENT
      ADD   NUM, DENOMINATOR
```

Logic Instructions

AND	Rd,Rr	;Rd = Rd AND Rr
OR	Rd,Rr	;Rd = Rd OR Rr
EOR	Rd,Rr	;Rd = Rd XOR Rr (immediate are not supported)
COM	Rd,Rr	;Rd = 1' Complement of Rd (11111111 – Rd)
NEG	Rd,Rr	;Rd = 2' Complement of Rd (100000000 – Rd)

- ***AND is used to clear an specific bit/s of a byte***
- ***OR is used to set an specific bit/s of a byte***

Example 5-15

Show the results of the following.

```
LDI    R20,0x35    ;R20 = 35H
ANDI   R20,0x0F    ;R20 = R20 AND 0FH (now R20 = 05)
```

Solution:

```
35H   0 0 1 1 0 1 0 1
0FH   0 0 0 0 1 1 1 1
05H   0 0 0 0 0 1 0 1   ;35H AND 0FH = 05H, Z = 0, N = 0
```

Setting and Clearing bits

AND	Rd,Rr	;Rd = Rd AND Rr
OR	Rd,Rr	;Rd = Rd OR Rr
EOR	Rd,Rr	;Rd = Rd XOR Rr (immediate are not supported)
COM	Rd,Rr	;Rd = 1' Complement of Rd (11111111 – Rd)
NEG	Rd,Rr	;Rd = 2' Complement of Rd (100000000 – Rd)

- ***AND is used to clear an specific bit/s of a byte***
- ***OR is used to set an specific bit/s of a byte***

AND

35H	0	0	1	1	0	1	0	1
0FH	0	0	0	0	1	1	1	1
05H	0	0	0	0	0	1	0	1

OR

04H	0000	0100
30H	0011	0000
34H	0011	0100

Branch and CP Instructions

CP Rd,Rr ;Rd – Rr (only flags are set)

Table 5-2: AVR Compare Instructions

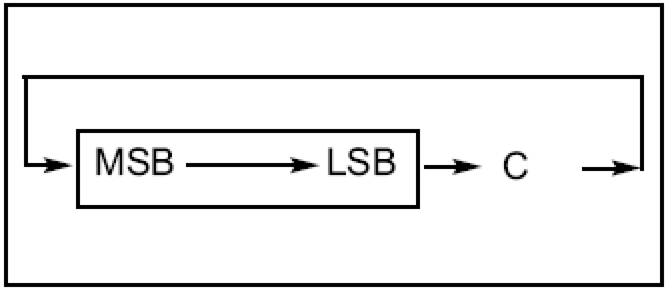
BREQ	Branch if equal	if(Z==1) PC = PC + k + 1
BRNE	Branch if not equal	if(Z==0) PC = PC + k + 1
BRSH	Branch if same or higher	if(C==0) PC = PC + k + 1
BRLO	Branch if lower	if(C==1) PC = PC + k + 1
BRLT	Branch if less than (signed)	if(S==1) PC = PC + k + 1
BRGE	Branch if greater than or equal (signed)	if(S==0) PC = PC + k + 1

- ***BRVC is used to branch when oVerflow is clear to zero***
- ***BRVS is used to branch when oVerflow is set to one***

ROR instruction

```
ROR     Rd                     ;Rd  (only flags are set)
```

In ROR, as bits are rotated from left to right, the carry flag enters the MSB and the LSB exits to the carry flag. In other words, **in ROR the C is moved to the MSB, and the LSB is moved to the C.**



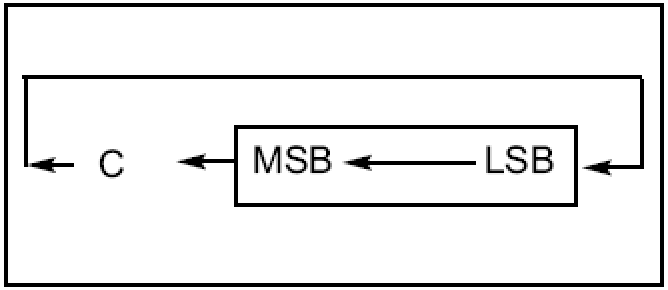
See what happens to 0010 0110 after running 3 ROR instructions:

```
CLC                        ;make C = 0 (carry is 0 )
LDI     R20 , 0x26         ;R20 = 0010 0110
ROR     R20                ;R20 = 0001 0011 C = 0
ROR     R20                ;R20 = 0000 1001 C = 1
ROR     R20                ;R20 = 1000 0100 C = 1
```

ROL instruction

ROR Rd ;Rd (only flags are set)

ROL. In ROL, as bits are shifted from right to left, the carry flag enters the LSB and the MSB exits to the carry flag. In other words, **in ROL the C is moved to the LSB, and the MSB is moved to the C.**

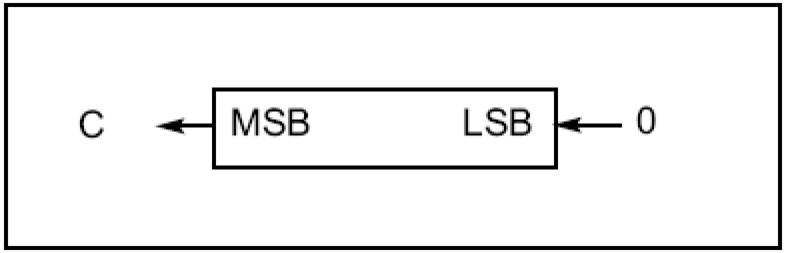



```
SEC              ;make C = 1 (carry is 0)
LDI R20,0x15     ;R20 = 0001 0101
ROL R20          ;R20 = 0010 1011 C = 0
ROL R20          ;R20 = 0101 0110 C = 0
ROL R20          ;R20 = 1010 1100 C = 0
ROL R20          ;R20 = 0101 1000 C = 1
```

LSL instruction

```
LSL    Rd              ;logical shift left
```

In LSL, as bits are shifted from right to left, 0 enters the LSB and the MSB exits to the carry flag. In other words, **in LSL 0 is moved to the LSB, and the MSB is moved to the C.**



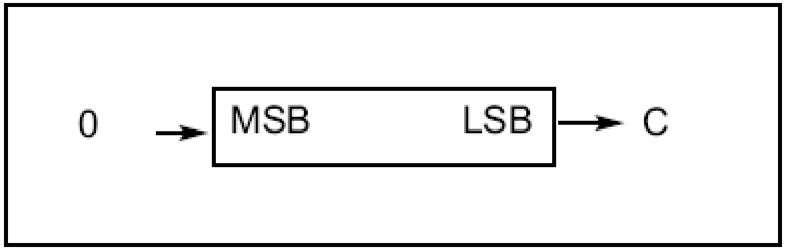
this instruction multiplies content of the register by 2 assuming that after LSL the carry flag is not set.

In the next code you can see what happens to 00100110 after running 3 LSL instructions.

```
CLC              ;make C = 0 (carry is 0 )
LDI R20 , 0x26   ;R20 = 0010 0110(38) c = 0
LSL R20          ;R20 = 0100 1100(74) C = 0
LSL R20          ;R20 = 1001 1000(148) C = 0
LSL R20          ;R20 = 0011 0000(98) C = 1 as C=1 and content of R20
                 ;is not multiplied by 2
```

```
ROR       Rd                  ;Rd  (only flags are set)
```

In LSR, as bits are shifted from left to right, 0 enters the MSB and the LSB exits to the carry flag. In other words, **in LSR 0 is moved to the MSB, and the LSB is moved to the C.**



this instruction divides content of the register by 2 and carry flag contains the remainder of division.

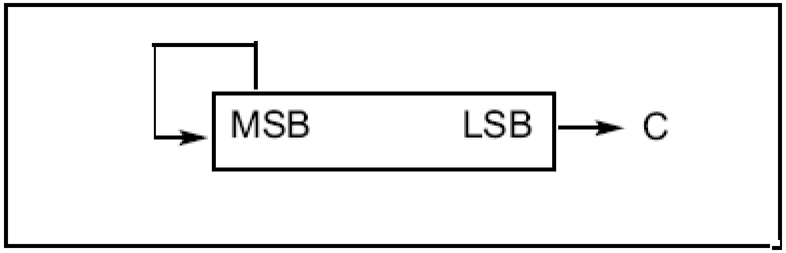
In the next code you can see what happens to 0010 0110 after running 3 LSL instructions.

```
LDI R20,0x26          ;R20 = 0010 0110 (38)
LSR R20               ;R20 = 0001 0011 (19) C = 0
LSR R20               ;R20 = 0000 1001 (9) C = 1
LSR R20               ;R20 = 0000 0100 (4) C = 1
```

ASR Instruction

```
ROR     Rd                    ;Rd  (only flags are set)
```

ASR means *arithmetic shift right*. ASR instruction can divide signed number by 2. In LSR, as bits are shifted from left to right, MSB is held constant and the LSB exits to the carry flag. In other words **MSB is not changed but is copied to D6, D6 is moved to D5, D5 is moved to D4 and so on.**



In the next code you can see what happens to 0010 0110 after running 5 ASL instructions.

```
LDI    R20 , 0D60          ;R20 = 1101 0000(-48) c = 0
LSL   R20                  ;R20 = 1110 1000(-24) C = 0
LSL   R20                  ;R20 = 1111 0100(-12) C = 0
LSL   R20                  ;R20 = 1111 1010(-6) C = 0
LSL   R20                  ;R20 = 1111 1101(-3) C = 0
LSL   R20                  ;R20 = 1111 1110(-1) C = 1
```

BCD, Packed BCD and ASCII conversion.

- **ASCII Codes**

BCD Codes

Packed BCD

Key	ASCII (hex)	Binary	BCD (unpacked)
0	30	011 0000	0000 0000
1	31	011 0001	0000 0001
2	32	011 0010	0000 0010
3	33	011 0011	0000 0011
4	34	011 0100	0000 0100
5	35	011 0101	0000 0101
6	36	011 0110	0000 0110
7	37	011 0111	0000 0111
8	38	011 1000	0000 1000
9	39	011 1001	0000 1001

ASCII and BCD Codes for Digits 0–9

Packed BCD to ASCII conversion

To convert packed BCD to ASCII:

- you must first convert it to unpacked BCD.
- Then the unpacked BCD is tagged with 011 0000 (30H).

Packed BCD = 1001 0010

Un packed BCD = 0000 1001 , 0000 0010

ACSII = 0011 1001 , 0011 0010

Advanced Assembly
Chapter 6

The AVR microcontroller and embedded systems
using assembly and c

Topics

- Assembler directives
- Addressing modes
- Macro
- EEPROM memory
- Checksum

Some Assembler directives

	Example
+	LDI R20,5+3 ;LDI R20,8
-	LDI R30,9-3 ;LDI R30,6
*	LDI R25,5*7 ;LDI R25,35
/	LDI R19,8/2 ;LDI R19,4

	Example
&	LDI R20,0x50&0x10 ;LDI R20,0x10
|	LDI R25,0x50|0x1 ;LDI R25,0x51
^	LDI R23,0x50^0x10 ;LDI R23,0x40

	Example
<<	LDI R16, 0x10<<1 ;LDI R16,0x20
>>	LDI R16, 0x8 >>2 ;LDI R16,0x2

HIGH and LOW

```
LDI     R20, LOW(0x1234)          LDI     R20, $34
LDI     R21, HIGH(0x1234)         LDI     R21, $12
```

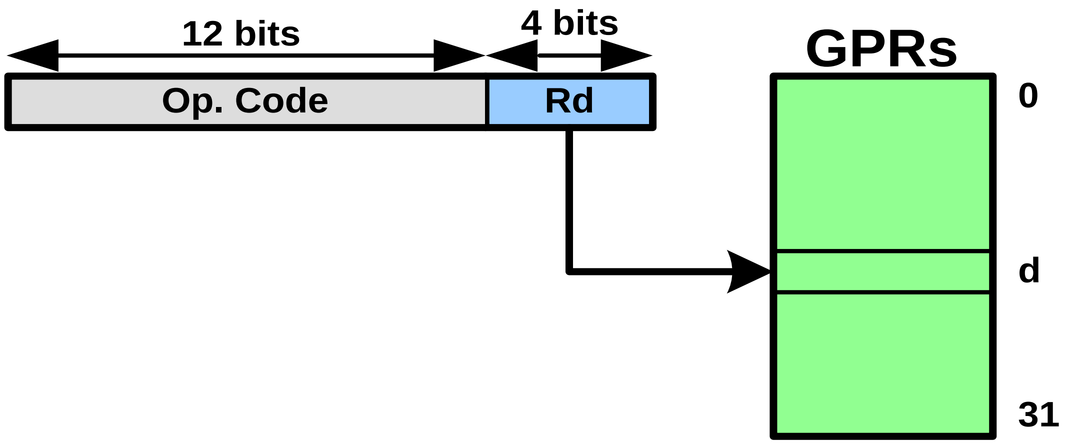
$1234 → HIGH: $12, LOW: $34

```
LDI     R20, LOW(-200)            LDI     R20, $FF
LDI     R21, HIGH(-200)           LDI     R21, $38
```

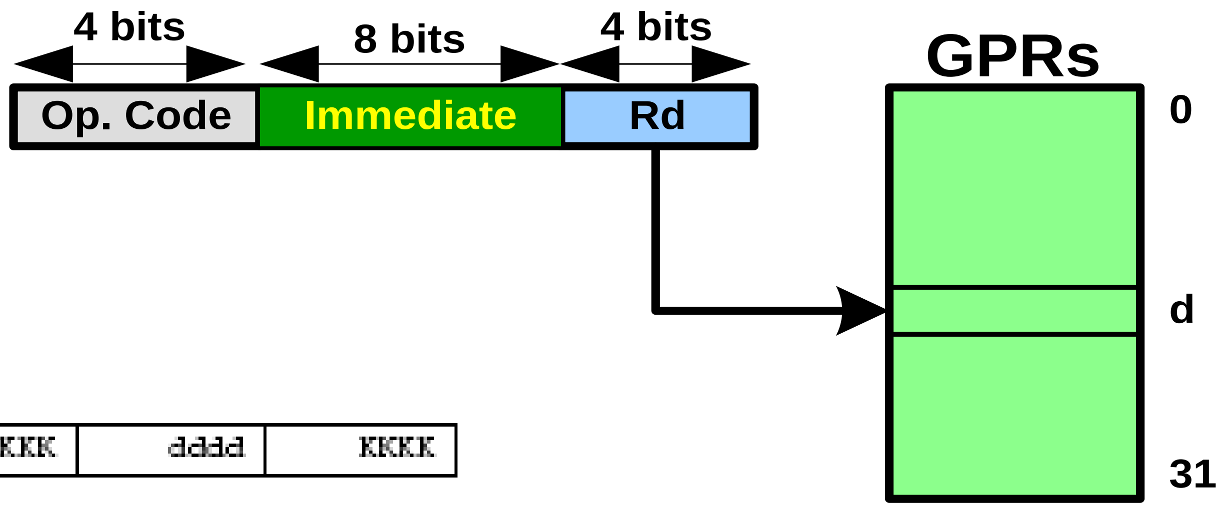
-200 = $FF38 → HIGH: $FF, LOW: $38

Single Register Addressing Mode

- Single Register Addressing Mode
  - INC Rd
    - INC R19
  - DEC Rd
    - DEC R23   ;R23 = R23 – 1



Immediate Addressing Mode (Single register with immediate)



- LDI Rd,K
  - LDI R19,25

1110	KKKK	dddd	KKKK

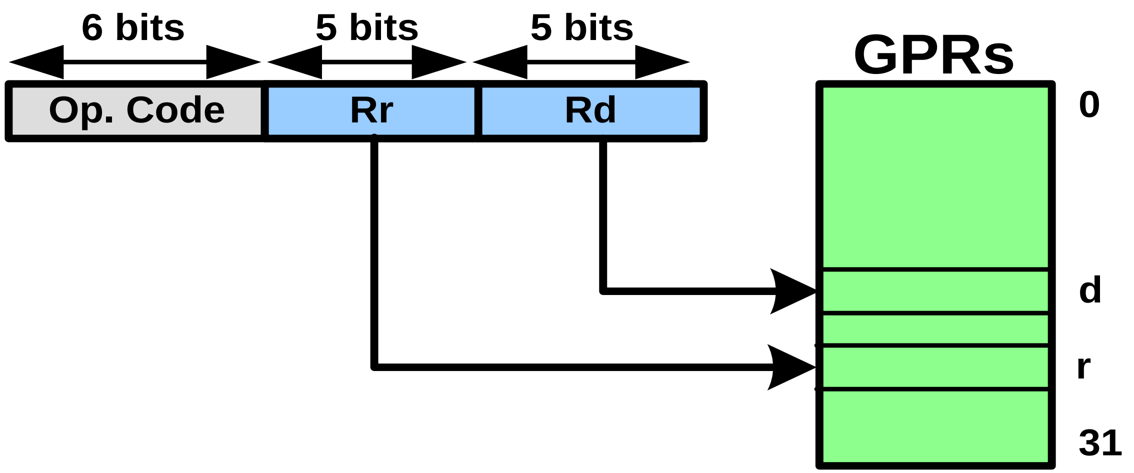
- SUBI Rd,K
  - SUBI R23,5 ;R23 = R23 – 5

0101	KKKK	dddd	KKKK

- ANDI Rd,K
  - ANDI R21,0x15

0111	KKKK	dddd	KKKK

Two-register addressing mode



- ADD Rd,Rr
  - ADD R26,R23
- SUB Rd,Rr
  - LDI R20,R10

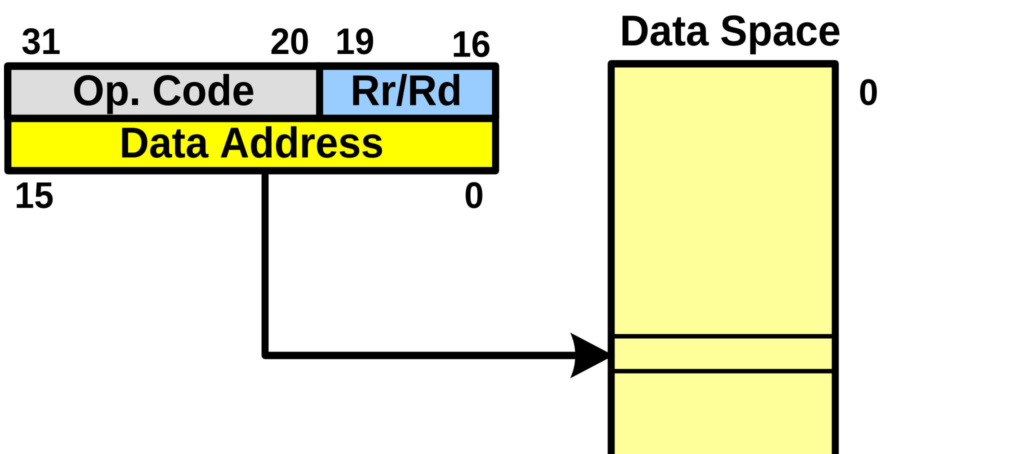
Direct addressing mode

- LDS Rd,address
  - LDS R19,0x313

1001	000d	dddd	0000
kkkk	kkkk	kkkk	kkkk

- STS address,Rs
  - STS 0x95,R19

1001	001d	dddd	0000
kkkk	kkkk	kkkk	kkkk



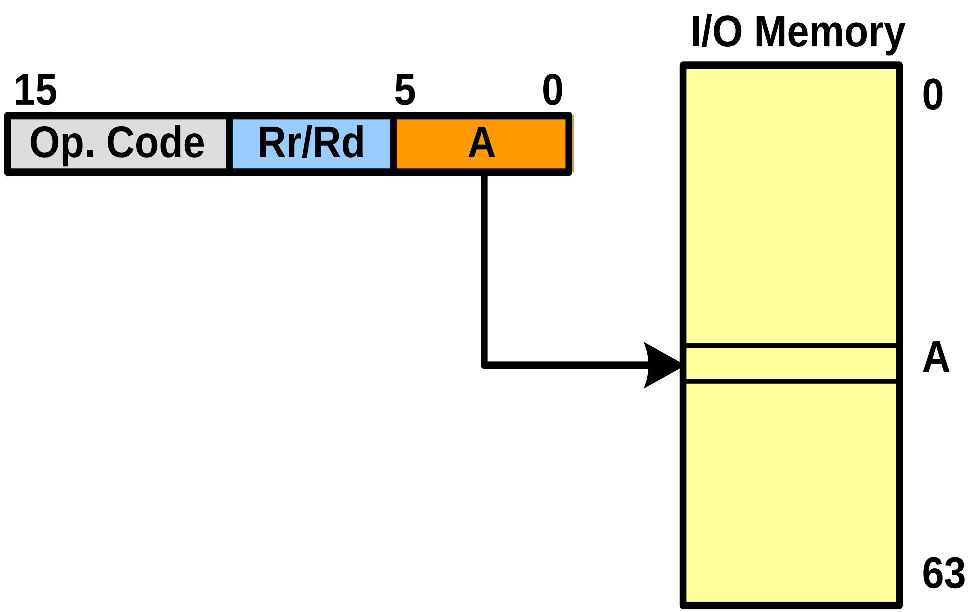
Note: RAMEND has been used to represent the highest location in data space.

I/O direct addressing mode

- OUT address, Rs
  - OUT 0x70,R16

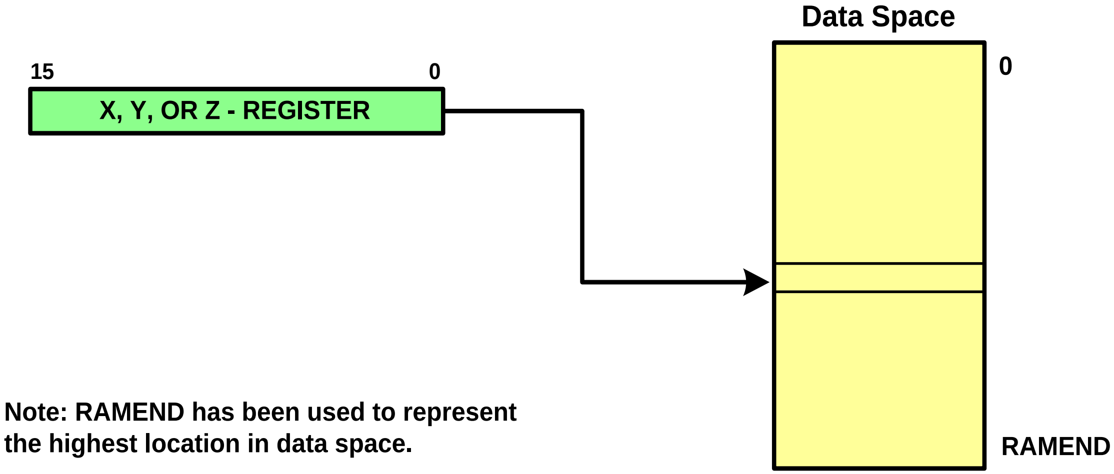
1011	1AAr	rrrr	AAAA

- IN Rs,address
  - IN R19,0x90

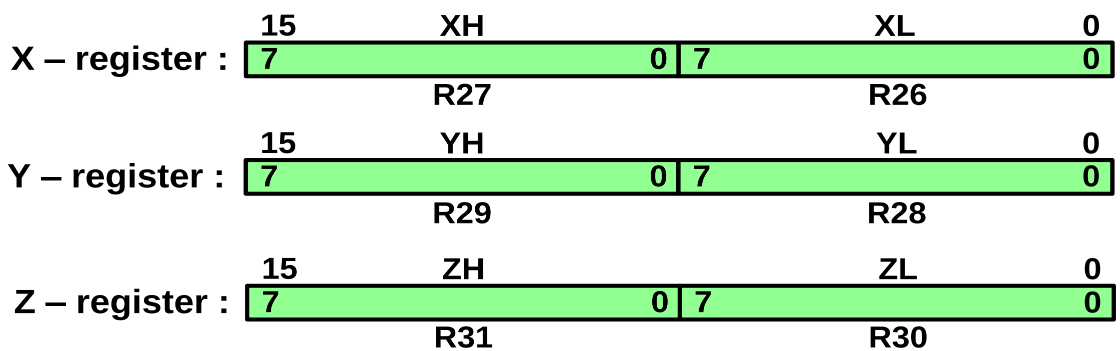


Register indirect addressing mode

- LD Rd,X
  - LD R24,X
  - LD R19,Y
  - LD R20,Z
- ST X,Rd
  - ST X,R18
  - ST Y,R20



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



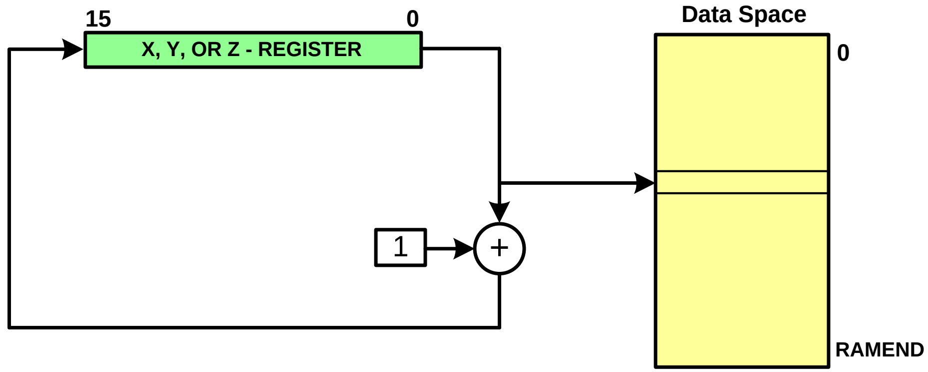
Example

- Write a program to copy the value $55 into memory locations $140 to $144

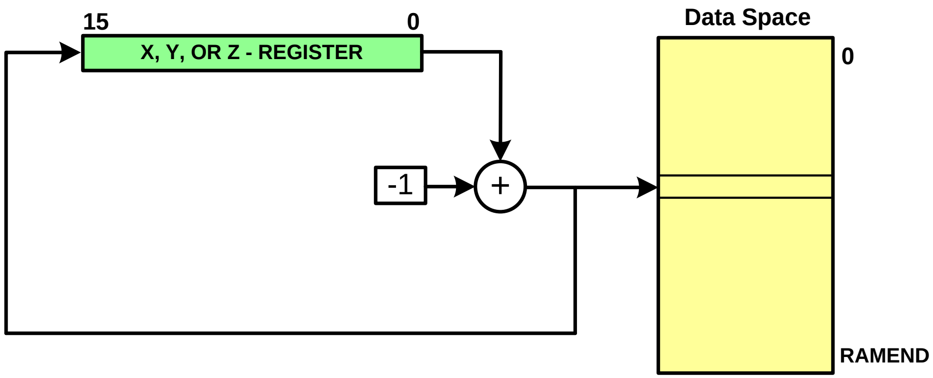
```
     LDI  R19,0x5        ;R19 = 5 (R19 for counter)
     LDI  R16,0x55       ;load R16 with value 0x55 (value to be copied)
     LDI  YL,0x40        →  LDI  YL,LOW(0x140)
     LDI  YH,0x1             LDI  YH,HIGH(0x140)
L1:  ST   Y,R16          ;copy R16 to memory location 0x140
     INC  YL             ;increment the low byte of Y
     DEC  R19            ;decrement the counter
     BRNE L1             ;loop until counter = zero
```

Auto-increment and Auto decrement

- Register indirect addressing with Post-increment
  - LD Rd, X+
    - LD R20,X+
  - ST X+, Rs
    - ST X+, R8



- Register indirect addressing with Pre-decrement
  - LD Rd, -X
    - LD R19,-X
  - ST –X,R31



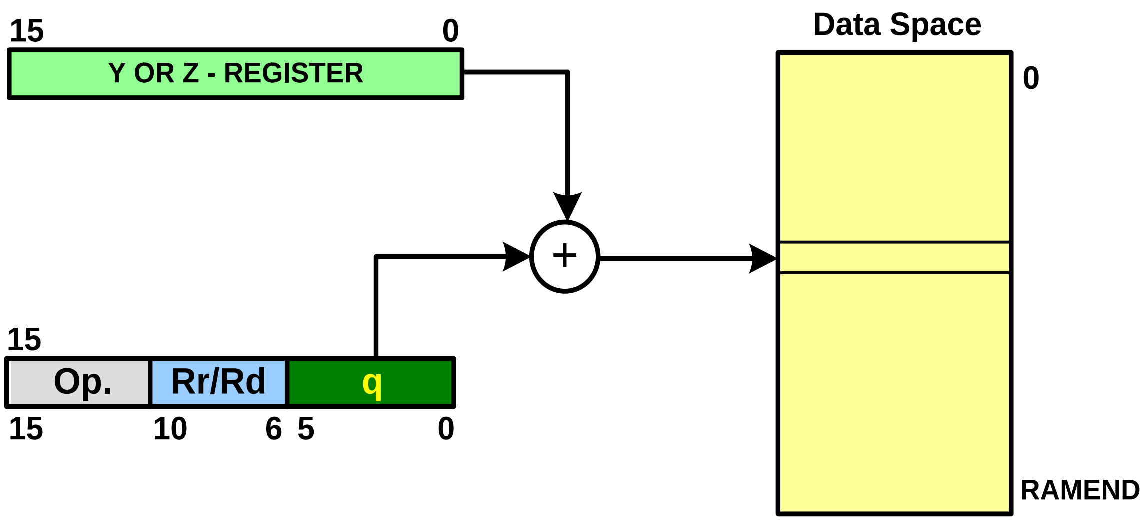
Example

- Write a program to copy the value $55 into memory locations $140 to $444

```
    LDI  R19,0x5          ;R19 = 5 (R19 for counter)
    LDI  R16,0x55         ;load R16 with value 0x55 (value to be copied)
    LDI  YL,LOW($140)     ;load the low byte of Y with value 0x40
    LDI  YH,HIGH($140)    ;load the high byte of Y with value 0x1
L1: ST   Y+,R16           ;copy R16 to memory location Y
    DEC  R19              ;decrement the counter
    BRNE L1               ;loop until counter = zero
```

Register indirect with displacement

- STD Z+q,Rr ;store Rr into location Z+q
  - STD Z+5,R20 ;store R20 in location Z+5
- LDD Rd, Z+q ;load from Z+q into Rd
  - LDD R20, Z+8 ;load from Z+8 into R20

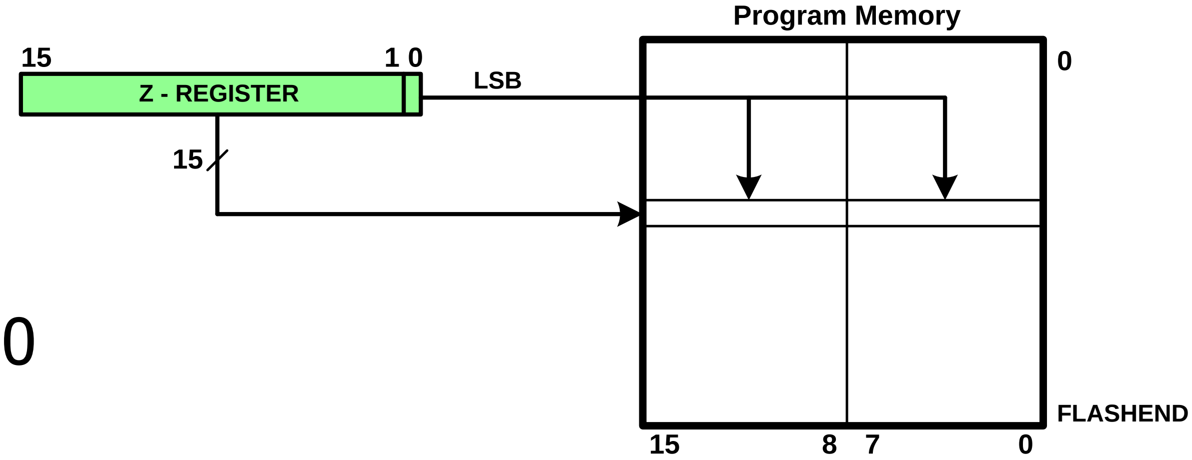


Storing fixed data in flash memory

```
DATA1: .DB 28                 ;DECIMAL(1C in hex)
DATA2: .DB 0b00110101         ;BINARY (35 in hex)
DATA3: .DB 0x39               ;HEX
DATA4: .DB 'Y'                ;single ASCII char
DATA6: .DB "Hello ALI";ASCII string
```

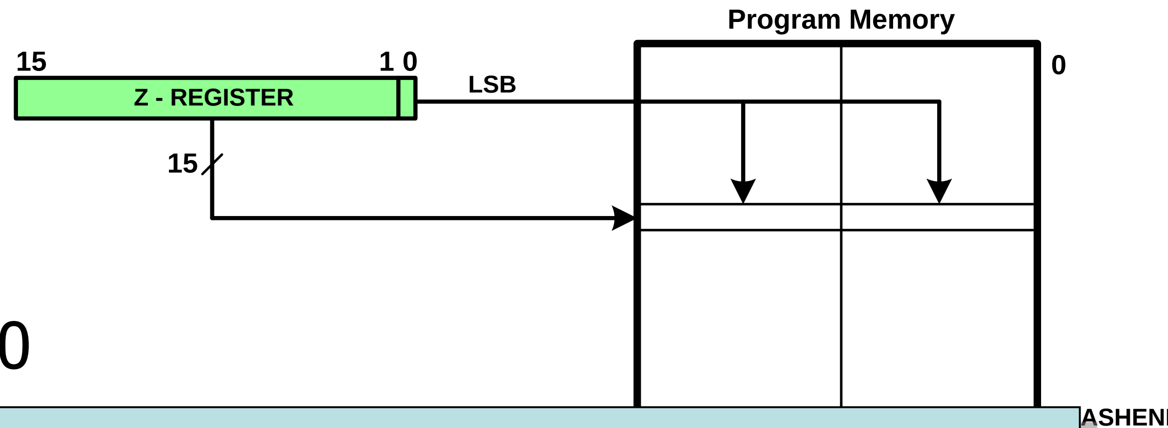
Storing fixed data in flash memory

- LPM Rd, Z
  - LPM R15, Z
  - Example:
    - LDI R30,0x80
    - LDI R31,0
    - LPM R18,Z ;read from the low byte of loc 0x40
- LPM Rd, Z+
  - LPM R20,Z



Storing fixed data in flash memory

- LPM Rd, Z
  - LPM R15, Z
  - Example:
    - LDI R30,0x80



Low	High	Address
0000 0000 0000 0000	0000 0000 0000 0001	0000 0000 0000 0000
0000 0000 0000 0010	0000 0000 0000 0011	0000 0000 0000 0001
0000 0000 0000 0100	0000 0000 0000 0101	0000 0000 0000 0010
0000 0000 0000 0110	0000 0000 0000 0111	0000 0000 0000 0011
0000 0000 0000 1000	0000 0000 0000 1001	0000 0000 0000 0100
0000 0000 0000 1010	0000 0000 0000 1011	0000 0000 0000 0101
⋮	⋮	
1111 1111 1111 1100	1111 1111 1111 1101	0111 1111 1111 1110
1111 1111 1111 1110	1111 1111 1111 1111	0111 1111 1111 1111

Low	High	Address
$0000	$0001	$0000
$0002	$0003	$0001
$0004	$0005	$0002
$0006	$0007	$0003
$0008	$0009	$0004
$000A	$000B	$0005
$FFFC	$FFFD	$7FFE
$FFFE	$FFFF	$7FFF

Example

- Analyze the following program; then rewrite it using LPM R20,Z+

```
.ORG    $0000                   ;burn into ROM starting at 0
        LDI     R20,0xFF
        OUT     DDRB,R20            ;make PB an output
        LDI     ZL, LOW(MYDATA<<1)  ;ZL = 0  look-up table low-byte addr
        LDI     ZH,HIGH(MYDATA<<1)  ;ZH = 0A look-up table high-byte addr
        LPM     R20,Z
        OUT     PORTB,R20
        INC     ZL
        LPM     R20,Z
        OUT     PORTB,R20
        INC     ZL
        LPM     R20,Z               ;load R20 with 'A' char pointed to by Z
        OUT     PORTB,R20           ;send it to Port B
HERE:   RJMP    HERE                ;stay here forever
;data is burned into code(program) space starting at $500
        .ORG    $500
MYDATA: .DB     "USA"
```

```
LPM     R20,Z+
OUT     PORTB,R20

LPM     R20,Z+
OUT     PORTB,R20
```

Example

- Assume that ROM space starting at $500 contains the message "The Promise of World Peace". Write a program to bring it into CPU one byte at a time and place the bytes in RAM locations starting at $140.

```
        .ORG  0                          ;burn into ROM starting at 0
        LDI   ZL, LOW(MYDATA<<1)         ;R30 = 00 low-byte addr
        LDI   ZH, HIGH(MYDATA<<1)        ;R31 = 0A, high-byte addr
        LDI   XL, LOW(0x140)             ;R26 = 40, low-byte RAM address
        LDI   XH, HIGH(0x140)            ;R27 = 1, high-byte RAM address
AGAIN:  LPM   R16, Z+                    ;read the table, then increment Z
        CPI   R16,0                      ;compare R16 with 0
        BREQ  END                        ;exit if end of string
        ST    X+, R16                    ;store R16 in RAM and inc X
        RJMP  AGAIN
END:    RJMP  END
.ORG    0x500                    ;data burned starting at 0x500
MYDATA: .DB "The Promise of World Peace",0
```

Macro

```
.MACRO  INITSTACK
    LDI   R16,HIGH(RAMEND)
    OUT   SPH,R16
    LDI   R16,LOW(RAMEND)
    OUT   SPL,R16
.ENDMACRO

INITSTACK
```

Macro

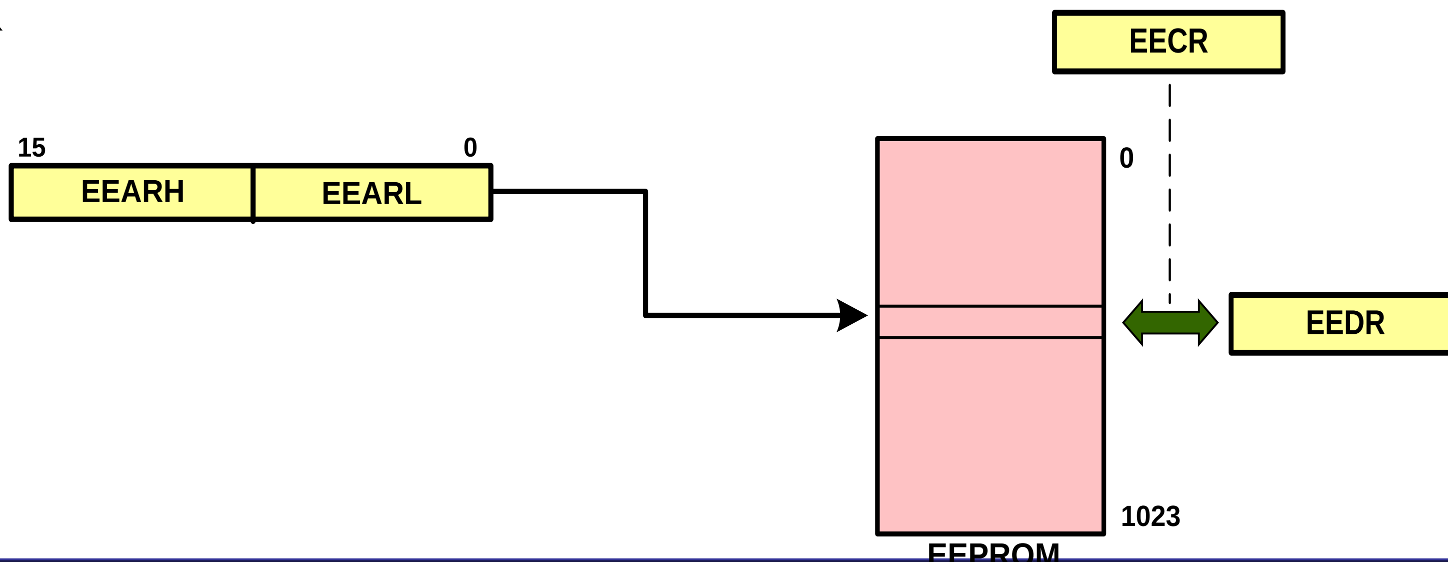
```
.MACRO LOADIO
   LDIR20,@1
   OUT       @0,R20
.ENDMACRO

LOADIO  DDRB,0xFF
LOADIO  PORTB,0x55
```

EEPROM

EEPROM Address Register

Bit	15	14	13	12	11	10	9	8
EEARH	-	-	-	-	-	-	EEAR9	EEAR8
EEARL	EEAR7	EEAR6	EEAR5	EEAR4	EEAR3	EEAR2	EEAR1	EEAR0
Bit	7	6	5	4	3	2	1	0

- EEARH:EEARL
- EEDR
- EECR



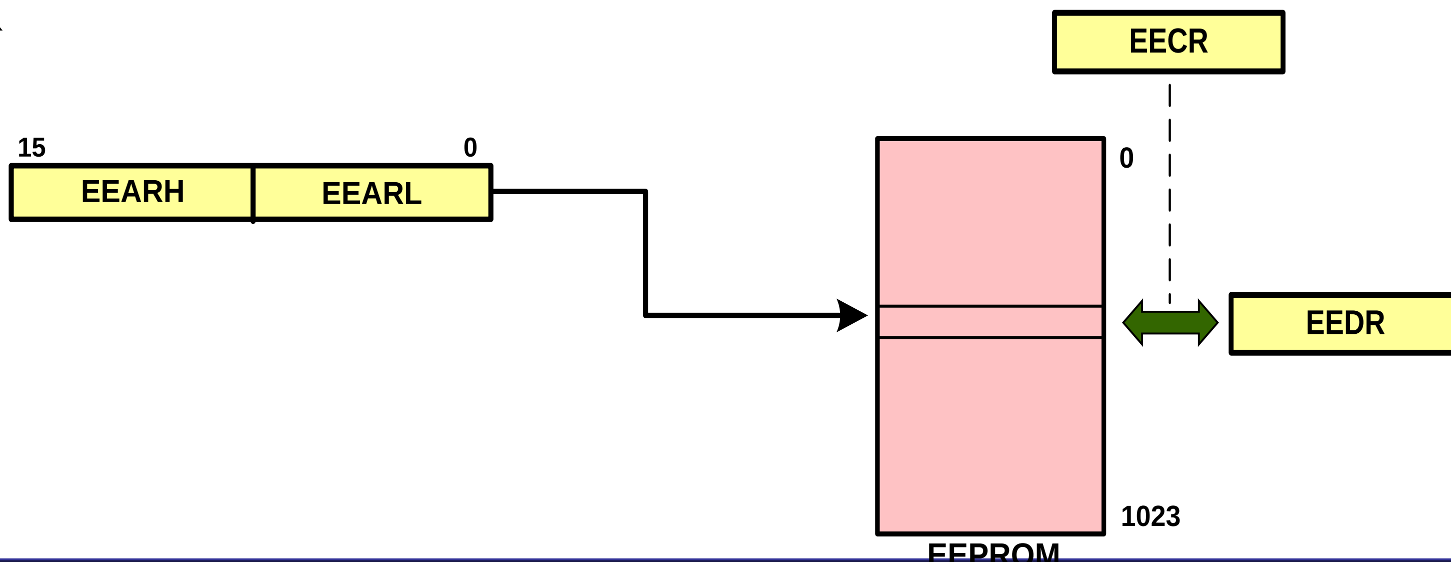
EEPROM

EEPROM Address Register

EEPROM Data Register

Bit	7	6	5	4	3	2	1	0	
	MSB							LSB	EEDR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- EEDR
- EECR



EEPROM

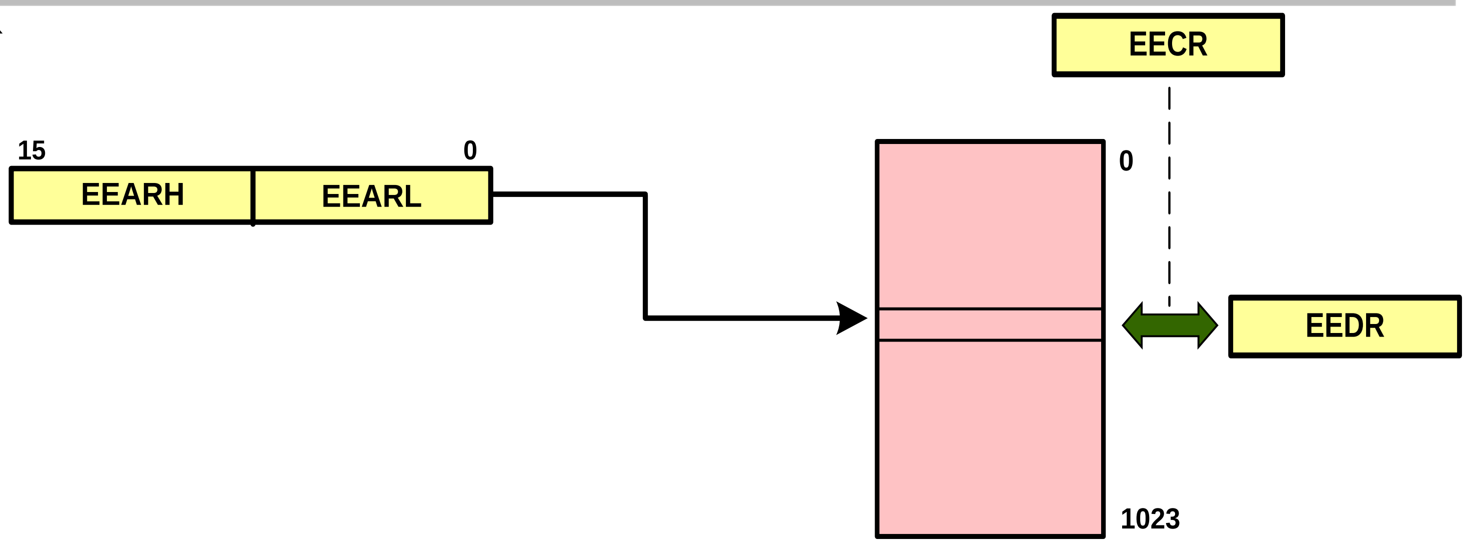
EEPROM Address Register

EEPROM Data Register

EEPROM Control Register

Bit	7	6	5	4	3	2	1	0	
	–	–	–	–	EERIE	EEMWE	EEWE	EERE	EECR
Read/Write	R	R	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	X	0	

- EEDR
- EECR



Reading from EEPROM

Example 6-29

Write an AVR program to read the content of location 0x005F of EEPROM into PORTB.

Solution:

```
.INCLUDE "M16DEF.INC"
      LDI    R16,0xFF
      OUT    DDRB,R16
WAIT:                   ;wait for last write to finish
      SBIC   EECR,EEWE  ;check EEWE to see if last write is finished
      RJMP   WAIT       ;wait more
      LDI    R18,0      ;load high byte of address to R18
      LDI    R17,0x5F   ;load low byte of address to R17
      OUT    EEARH, R18 ;load high byte of address to EEARH
      OUT    EEARL, R17 ;load low byte of address to EEARL
      SBI    EECR,EERE  ;set Read Enable to one
      IN     R16,EEDR   ;load EEPROM Data Register to R16
      OUT    PORTB,R16  ;out R16 to PORTB
```

Writing into EEPROM

- 1. Wait until EEWE becomes zero.
- 2. Write new EEPROM address to EEAR (optional).
- 3. Write new EEPROM data to EEDR (optional).
- 4. Set EEMWE bit to one.
- 5. Within four clock cycles after setting EEMWE, set EEWE to one.

Writing into EEPROM

- 1
- 2
(c
- 3 I).
- 4
- 5 /E,
se

Example 6-28

Write an AVR program to store 'G' into location 0x005F of EEPROM .

Solution:

```
.INCLUDE "M16DEF.INC"

WAIT:                 ;wait for last write to finish
SBIC   EECR,EEWE      ;check EEWE to see if last write is finished
RJMP   WAIT           ;wait more
LDI    R18,0          ;load high byte of address to R18
LDI    R17,0x5F       ;load low byte of address to R17
OUT    EEARH, R18     ;load high byte of address to EEARH
OUT    EEARL, R17     ;load low byte of address to EEARL
LDI    R16,'G'        ;load 'G' to R16
OUT    EEDR,R16       ;load R16 to EEPROM Data Register
SBI    EECR,EEMWE     ;set Master Write Enable to one
SBI    EECR,EEWE      ;set Write Enable to one
```

Run and simulate the code on AVR Studio to see how the content of the EEPROM changes after the last line of code. Enter four NOP instructions before the last line, change the 'G' to 'H', and run the code again. Explain why the code doesn't store 'H' at location 0x005F of EEPROM.

Checksum

- To detect data corruption
- Calculating checksum byte:
  - Add the bytes together and drop the carries
  - Take the 2's complement of the total sum
- Testing checksum
  - Add the bytes together and drop the carries
  - Add the checksum byte to the sum
  - If the result is not zero, data is corrupted

Example

- Find the checksum byte for the followings:

  $25, $62, $3F, $52

Solution:

```
    $25
+   $62
+   $3F
+   $52
  -----
  $1 18
```

Checksum byte = 2's complement of $18 = $E8

Example

- The checksum byte is $E8. Test checksum for the following data:

  $25, $62, $3F, $52

Solution:

```
     $25
+    $62
+    $3F
+    $52
+    $E8
  -------
     $00    ➔ not corrupted
```