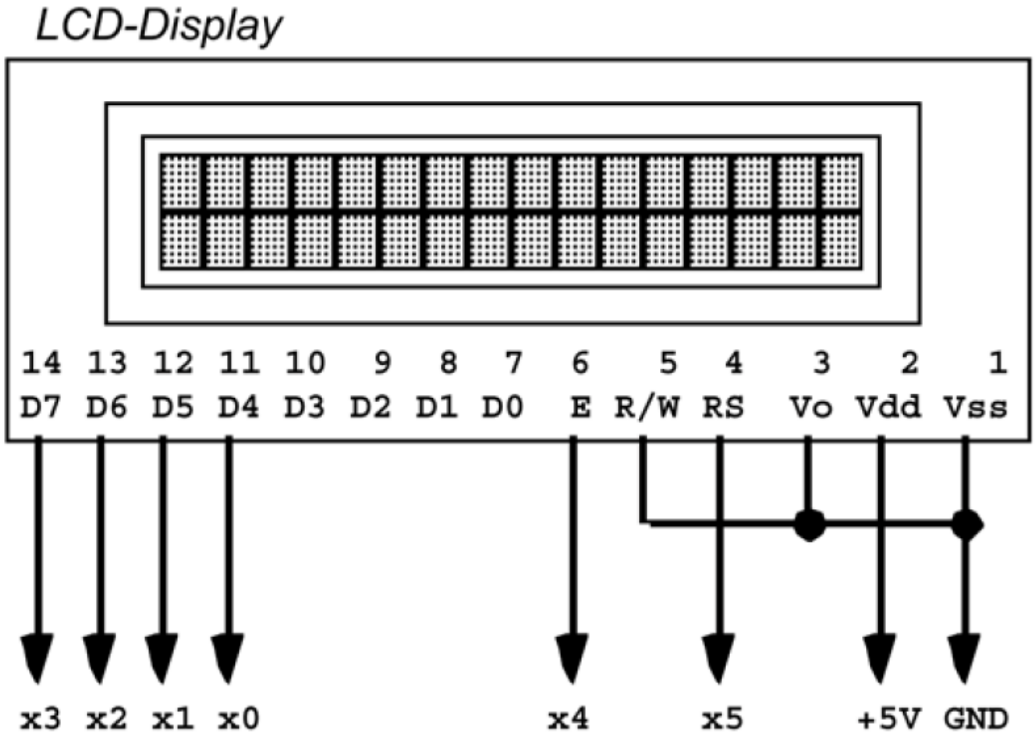
1DT301, Computer Technology

Tuesday, October 1, 2019

- Introduction to lab 5
- Addressing modes
- Program examples

Lab 5, LCD Display



Figur 8.1: LCD-displayen och dess anslutning mot datorkortet

Lab 5, LCD Display

Task 1: Write a program that displays a character on the display.

Write a program in Assembly that displays the character %. Look in the data sheet how to initiate the display. The data sheet you'll find on MyMoodle. The display will be connected as in the figure above. 4-bit-mode should be used, since only RS, E, D7, D6, D5 and D4 are connected to I/O-pins on the STK500 or STK600.
(The program *lab5_init_display.asm* gives you a good start…)

Lab 5, LCD Display

Task 2: Electronic bingo machine.

You should create an electronic bingo generator. The generator should create random numbers between 1 and 75. The numbers should be displayed on the display. Clear the display before a new value is displayed. Use interrupt and a pushbutton for the input.

Lab 5, LCD Display

Task 3: Serial communication and display.

Use program modules from lab 4 and write a program that receives a character on the serial port and displays each character on the display.

Lab 5, LCD Display

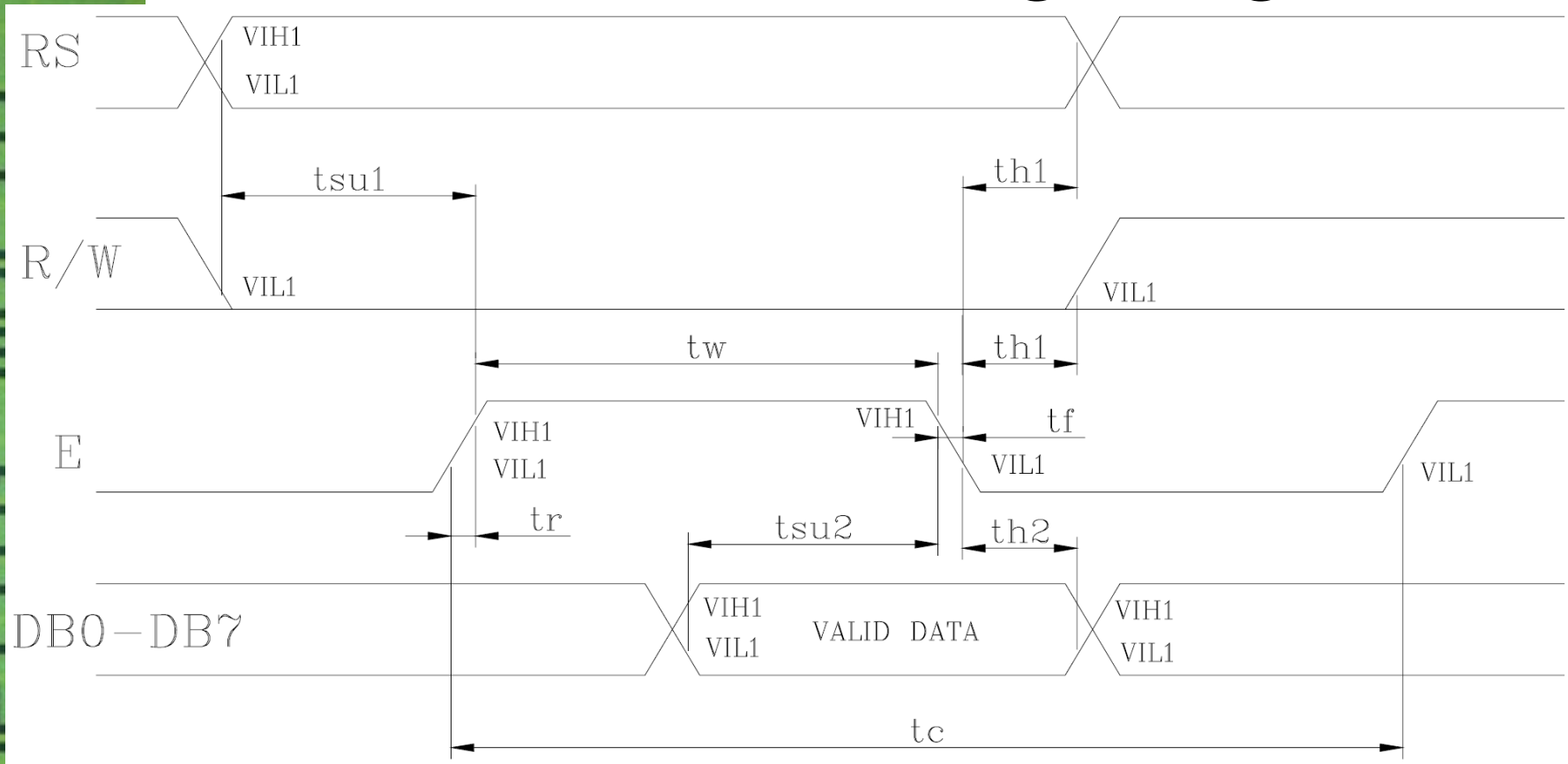
Task 4: Modify the program in task 3.

Modify the program in task 3 so 4 lines of text can be displayed. Each textline should be displayed during 5 seconds, after that the text on line 1 should be moved to line 2 and so on. The text should be entered from the terminal program, via the serial port.

GDM1602A,
Interface Pin Description

Pin no.	Symbol	External connection	Function
1	V_{SS}	Power supply	Signal ground for LCM (GND)
2	V_{DD}		Power supply for logic for LCM
3	V_0		Contrast adjust
4	RS	MPU	Register select signal
5	R/W	MPU	Read/write select signal
6	E	MPU	Operation (data read/write) enable signal
7~10	DB0~DB3	MPU	Four low order bi-directional three-state data bus lines. Used for data transfer between the MPU and the LCM. These four are not used during 4-bit operation.
11~14	DB4~DB7	MPU	Four high order bi-directional three-state data bus lines. Used for data transfer between the MPU
15	LED+	LED BKL power supply	Power supply for BKL
16	LED-		Power supply for BKL (GND)

Write mode timing diagram



RS – Instruction/Data Register Selection.

RS = 0: Instruction Register, RS =1: Data Register

E = Enable Signal

Instruction table

Instruction	Instruction code										Description	Execution time (fosc= 270 KHZ
	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0		
Clear Display	0	0	0	0	0	0	0	0	0	1	Write "20H" to DDRA and set DDRAM address to "00H" from AC	1.53ms
Return Home	0	0	0	0	0	0	0	0	1	-	Set DDRAM address to "00H" From AC and return cursor to Its original position if shifted. The contents of DDRAM are not changed.	1.53ms
Entry mode Set	0	0	0	0	0	0	0	1	I/D	SH	Assign cursor moving direction And blinking of entire display	39us
Display ON/ OFF control	0	0	0	0	0	0	1	D	C	B	Set display (D), cursor (C), and Blinking of cursor (B) on/off Control bit.	
Cursor or Display shift	0	0	0	0	0	1	S/C	R/L	-	-	Set cursor moving and display Shift control bit, and the Direction, without changing of DDRAM data.	39us
Function set	0	0	0	0	1	DL	N	F	-	-	Set interface data length (DL: 8-Bit/4-bit), numbers of display Line (N: =2-line/1-line) and, Display font type (F: 5x11/5x8)	39us

Instruction table

Function set	0	0	0	0	1	DL	N	F	-	-	Set interface data length (DL: 8-Bit/4-bit), numbers of display Line (N: =2-line/1-line) and, Display font type (F: 5x11/5x8)	39us
Set CGRAM Address	0	0	0	1	AC5	AC4	AC3	AC2	AC1	AC0	Set CGRAM address in address Counter.	39us
Set DDRAM Address	0	0	1	AC6	AC5	AC4	AC3	AC2	AC1	AC0	Set DDRAM address in address Counter.	39us
Read busy Flag and Address	0	1	BF	AC6	AC5	AC4	AC3	AC2	AC1	AC0	Whether during internal Operation or not can be known By reading BF. The contents of Address counter can also be read.	0us
Write data to Address	1	0	D7	D6	D5	D4	D3	D2	D1	D0	Write data into internal RAM (DDRAM/CGRAM).	43us
Read data From RAM	1	1	D7	D6	D5	D4	D3	D2	D1	D0	Read data from internal RAM (DDRAM/CGRAM).	43us

C:\Document__Kurser\D2_E2\Datorteknik\HT_2016\Program_examples\Lab_5_2016\Lab_5_init_HT_2016.asm

```
;    1DT301
;    Lab 5, Initialize display JHD202A.
;
;    Date: 2016-09-30
;    Author, Anders Haggren
;    Modified:
;
;    Function
;    --------
;    Initialize display JHD202 connected to PORTE
;
;    (run @ 1.8432 MHz clk frequency)
;

.include    "m2560def.inc"
.def    Temp    = r16
.def    Data    = r17
.def    RS  = r18

.equ    BITMODE4    = 0b00000010        ; 4-bit operation
.equ    CLEAR   = 0b00000001            ; Clear display
.equ    DISPCTRL    = 0b00001111        ; Display on, cursor on, blink on.

.cseg
.org    0x0000              ; Reset vector
    jmp reset

.org    0x0072

reset:

    ldi Temp, HIGH(RAMEND)  ; Temp = high byte of ramend address
    out SPH, Temp           ; sph = Temp
    ldi Temp, LOW(RAMEND)   ; Temp = low byte of ramend address
    out SPL, Temp           ; spl = Temp

    ser Temp                ; r16 = 0b11111111
    out DDRE, Temp          ; port E = outputs ( Display JHD202A)
    clr Temp                ; r16 = 0
    out PORTE, Temp

loop:   nop
    rjmp loop           ; loop forever

; **
; ** init_display
; **
init_disp:
    rcall power_up_wait     ; wait for display to power up
```

C:\Document__Kurser\D2_E2\Datorteknik\HT_2016\Program_examples\Lab_5_2016\Lab_5_init_HT_2016.asm

```
; **
; ** init_display
; **
init_disp:
    rcall power_up_wait      ; wait for display to power up

    ldi Data, BITMODE4       ; 4-bit operation
    rcall write_nibble       ; (in 8-bit mode)
    rcall short_wait         ; wait min. 39 us
    ldi Data, DISPCTRL       ; disp. on, blink on, curs. On
    rcall write_cmd          ; send command
    rcall short_wait         ; wait min. 39 us
clr_disp:
    ldi Data, CLEAR          ; clr display
    rcall write_cmd          ; send command
    rcall long_wait          ; wait min. 1.53 ms
    ret

; **
; ** write char/command
; **

write_char:
    ldi RS, 0b00100000       ; RS = high
    rjmp write
write_cmd:
    clr RS                   ; RS = low
write:
    mov Temp, Data           ; copy Data
    andi Data, 0b11110000    ; mask out high nibble
    swap Data                ; swap nibbles
    or Data, RS              ; add register select
    rcall write_nibble       ; send high nibble
    mov Data, Temp           ; restore Data
    andi Data, 0b00001111    ; mask out low nibble
    or Data, RS              ; add register select

write_nibble:
    rcall switch_output      ; Modify for display JHD202A, port E
    nop                      ; wait 542nS
    sbi PORTE, 5             ; enable high, JHD202A
    nop
    nop                      ; wait 542nS
    cbi PORTE, 5             ; enable low, JHD202A
    nop
    nop                      ; wait 542nS
    ret

; **
; ** busy_wait loop
```

C:\Document__Kurser\D2_E2\Datorteknik\HT_2016\Program_examples\Lab_5_2016\Lab_5_init_HT_2016.asm

```
    nop                         ; wait 542nS
    ret

; **
; ** busy_wait loop
; **
short_wait:
    clr zh                      ; approx 50 us
    ldi zl, 30
    rjmp wait_loop
long_wait:
    ldi zh, HIGH(1000)          ; approx 2 ms
    ldi zl, LOW(1000)
    rjmp wait_loop
dbnc_wait:
    ldi zh, HIGH(4600)          ; approx 10 ms
    ldi zl, LOW(4600)
    rjmp wait_loop
power_up_wait:
    ldi zh, HIGH(9000)          ; approx 20 ms
    ldi zl, LOW(9000)

wait_loop:
    sbiw z, 1                   ; 2 cycles
    brne wait_loop              ; 2 cycles
    ret

; **
; ** modify output signal to fit LCD JHD202A, connected to port E
; **

switch_output:
    push Temp
    clr Temp
    sbrc Data, 0                ; D4 = 1?
    ori Temp, 0b00000100        ; Set pin 2
    sbrc Data, 1                ; D5 = 1?
    ori Temp, 0b00001000        ; Set pin 3
    sbrc Data, 2                ; D6 = 1?
    ori Temp, 0b00000001        ; Set pin 0
    sbrc Data, 3                ; D7 = 1?
    ori Temp, 0b00000010        ; Set pin 1
    sbrc Data, 4                ; E = 1?
    ori Temp, 0b00100000        ; Set pin 5
    sbrc Data, 5                ; RS = 1?
    ori Temp, 0b10000000        ; Set pin 7 (wrong in previous version)
    out porte, Temp
    pop Temp
    ret
```

2. Machine code

Below is part of a program, cut out from the disassembler. Machine code is printed with the most significant byte first, ie as it is described in the Instruction Set Manual.
Example: For the instruction RET, the machine code is 9508.
On lines E0 – E6 the assembler code is removed.

a) Recreate the assembler code by interpreting machine code. Use the enclosed examples from the Instruction Set Manual. **5p**

b) Explain the function of the complete program. **5p**

```
+000000DB:   95C8        LPM
+000000DC:   2D10        MOV       R17,R0
+000000DD:   0F21        ADD       R18,R17
+000000DE:   DFF2        RCALL     PC-0x000D

+000000DF:   9631        ADIW      R30,0x01
+000000E0:   95C8
+000000E1:   2D10
+000000E2:   0F21
+000000E3:   3010
+000000E4:   F011
+000000E5:   DFEB
+000000E6:   CFF8

+000000E7:   9508        RET
```

LPM – Load Program Memory

Description:

Loads one byte pointed to by the Z-register into the destination register Rd. This instruction features a 100% space effective constant initialization or constant data fetch. The Program memory is organized in 16-bit words while the Z-pointer is a byte address. Thus, the least significant bit of the Z-pointer selects either low byte (Z_{LSB} = 0) or high byte (Z_{LSB} = 1). This instruction can address the first 64K bytes (32K words) of Program memory. The Z-pointer Register can either be left unchanged by the operation, or it can be incremented. The incrementation does not apply to the RAMPZ Register.

Devices with Self-Programming capability can use the LPM instruction to read the Fuse and Lock bit values. Refer to the device documentation for a detailed description.

Not all variants of the LPM instruction are available in all devices. Refer to the device specific instruction set summary. The LPM instruction is not implemented at all in the AT90S1200 device.

The result of these combinations is undefined:

```
LPM r30, Z+
LPM r31, Z+
```

	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 ≤ d ≤ 31	PC ← PC + 1
(iii)	LPM Rd, Z+	0 ≤ d ≤ 31	PC ← PC + 1

16-bit Opcode:

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

0x95C8

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Example:

```
        ldi   ZH, high(Table_1<<1); Initialize Z-pointer
        ldi   ZL, low(Table_1<<1)
        lpm   r16, Z              ; Load constant from Program
                                  ; Memory pointed to by Z (r31:r30)
...
Table_1:
.dw 0x5876                        ; 0x76 is addresses when Z_LSB = 0
                                  ; 0x58 is addresses when Z_LSB = 1
...
```

```
+000000DB:    95C8          LPM
+000000DC:    2D10          MOV         R17,R0
+000000DD:    0F21          ADD         R18,R17
+000000DE:    DFF2          RCALL       PC-0x000D

+000000DF:    9631          ADIW        R30,0x01
+000000E0:    95C8
+000000E1:    2D10
+000000E2:    0F21
+000000E3:    3010
+000000E4:    F011
+000000E5:    DFEB
+000000E6:    CFF8

+000000E7:    9508          RET
```

MOV – Copy Register

Description:

This instruction makes a copy of one register into another. The source register Rr is left unchanged, while the desti
register Rd is loaded with a copy of Rr.

Operation:

(i) Rd ← Rr

	Syntax:	Operands:	Program Counter:
(i)	MOV Rd,Rr	0 ≤ d ≤ 31, 0 ≤ r ≤ 31	PC ← PC + 1

16-bit Opcode:

0010	11rd	dddd	rrrr
2	D	1	0
0010	1101	0001	0000

r = 00000 r = R0
d = 10001 d = R17

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
-	-	-	-	-	-	-	-

Example:

```
            mov     r16,r0    ; Copy r0 to r16
            call    check     ; Call subroutine
            ...
check:      cpi     r16,$11   ; Compare r16 to $11
            ...
            ret               ; Return from subroutine
```

Words: 1 (2 bytes)
Cycles: 1

```
+000000DB:    95C8          LPM
+000000DC:    2D10          MOV         R17,R0
+000000DD:    0F21          ADD         R18,R17
+000000DE:    DFF2          RCALL       PC-0x000D

+000000DF:    9631          ADIW        R30,0x01
+000000E0:    95C8
+000000E1:    2D10
+000000E2:    0F21
+000000E3:    3010
+000000E4:    F011
+000000E5:    DFEB
+000000E6:    CFF8

+000000E7:    9508          RET
```

ADD – Add without Carry

Description:

Adds two registers without the C Flag and places the result in the destination register Rd.

Operation:

(i) Rd ← Rd + Rr

	Syntax:	Operands:	Program Counter:
(i)	ADD Rd,Rr	0 ≤ d ≤ 31, 0 ≤ r ≤ 31	PC ← PC + 1

16-bit Opcode:

0000	11rd	dddd	rrrr
0	F	2	1
0000	1111	0010	0001

r = 10001 r = R17
d = 10010 d = R18

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
–	–	⇔	⇔	⇔	⇔	⇔	⇔

H: $Rd3 \bullet Rr3 + Rr3 \bullet \overline{R3} + \overline{R3} \bullet Rd3$
Set if there was a carry from bit 3; cleared otherwise

S: N ⊕ V, For signed tests.

V: $Rd7 \bullet Rr7 \bullet \overline{R7} + \overline{Rd7} \bullet \overline{Rr7} \bullet R7$
Set if two's complement overflow resulted from the operation; cleared otherwise.

N: R7
Set if MSB of the result is set; cleared otherwise.

Z: $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
Set if the result is $00; cleared otherwise.

BREQ – Branch if Equal

Description:

Conditional relative branch. Tests the Zero Flag (Z) and branches relatively to PC if Z is set. If the instruction is executed immediately after any of the instructions CP, CPI, SUB or SUBI, the branch will occur if and only if the unsigned or signed binary number represented in Rd was equal to the unsigned or signed binary number represented in Rr. This instruction branches relatively to PC in either direction (PC - 63 ≤ destination ≤ PC + 64). The parameter k is the offset from PC and is represented in two's complement form. (Equivalent to instruction BRBS 1,k).

Operation:

(i) If Rd = Rr (Z = 1) then PC ← PC + k + 1, else PC ← PC + 1

	Syntax:	Operands:	Program Counter:
(i)	BREQ k	-64 ≤ k ≤ +63	PC ← PC + k + 1 PC ← PC + 1, if condition is false

16 bit Opcode:

1111	00kk	kkkk	k001

F 0 1 1

1111 0000 0001 0001

k= 00 0001 0 = 2

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Example:

```
        cp    r1,r0     ; Compare registers r1 and r0
        breq  equal     ; Branch if registers equal
        ...
equal:  nop             ; Branch destination (do nothing)
```

Words: 1 (2 bytes)

Cycles: 1 if condition is false

2 if condition is true

RJMP – Relative Jump

Description:

Relative jump to an address within PC - 2K +1 and PC + 2K (words). In the assembler, labels are used instead of rela
operands. For AVR microcontrollers with Program memory not exceeding 4K words (8K bytes) this instruction can addr
the entire memory from every address location.

Operation:

(i) PC ← PC + k + 1

	Syntax:	Operands:	Program Counter:	Stack
(i)	RJMP k	-2K ≤ k < 2K	PC ← PC + k + 1	Unchanged

16-bit Opcode:

1100	kkkk	kkkk	kkkk
C	F	F	8
1100	1111	1111	1000

k= 1111 1111 1000 = ?

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

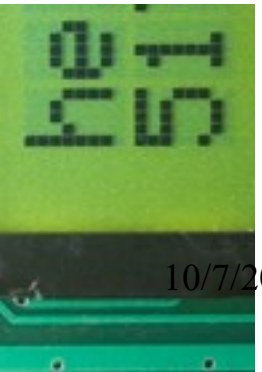
Example:

```
            cpi     r16,$42   ; Compare r16 to $42
            brne    error     ; Branch if r16 <> $42
            rjmp    ok        ; Unconditional branch
error:      add     r16,r17   ; Add r17 to r16
            inc     r16       ; Increment r16
ok:         nop               ; Destination for rjmp (do nothing)
```

Words: 1 (2 bytes)
Cycles: 2

xplain the function of the complete program.

```
00DB: print_string
00DB:    95C8          LPM                    Load program memory
00DC:    2D10          MOV       R17,R0       Copy register
00DD:    0F21          ADD       R18,R17      Add without carry
00DE:    DFF2          RCALL     PC-0x000D    Relative call subroutine
00DF: next_character
00DF:    9631          ADIW      R30,0x01     Add immediate to word
00E0:    95C8          LPM                    Load program memory
00E1:    2D10          MOV       R17,R0       Copy register
00E2:    0F21          ADD       R18,R17      Add without carry
00E3:    3010          CPI       R17,0x00     Compare with immediate
00E4:    F011          BREQ      PC+0x03      Branch if equal
00E5:    DFEB          RCALL     PC-0x0014    Relative call subroutine
00E6:    CFF8          RJMP      PC-0x0007    Relative jump
00E7: slutone
00E7:    9508          RET                    Subroutine return
```



Example 2:

Analyze the program below. RX2Int is an interrupt routine, which will be executed each time a new character is received on serial port, RS232. (USART1)
After reading the character in UDR1, the routine will call the subroutine **check**.

Make a flowchart for the subroutine **check**, which explains the function. 5p

b) Explain with words what the subroutine check will do.
An ASCII-code table is enclosed on next page. 5p

```
;>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
;    1ED022, Computer Technology I
;    Date: 2013-02-16
;    Anders Haggren
;    Function:     ?
;    Exam example, task 5
;
;    RX2Int - Interrupt routine for receiving
;    characters from USART1 data register UDR1
;<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<

RX2Int:
     push Temp              ; Save Temp on stack
     in Temp, SREG          ; Save SREG on stack
     push Temp

     lds Char, UDR1         ; Read character from receive buffer

     rcall check            ; Call subroutine check

     com Char               ; Invert bits
     out portb, Char        ; Write value to PortB

     pop Temp
     out SREG, Temp         ; Restore SREG
     pop Temp               ; Restore Temp
     reti
```

Example 2:

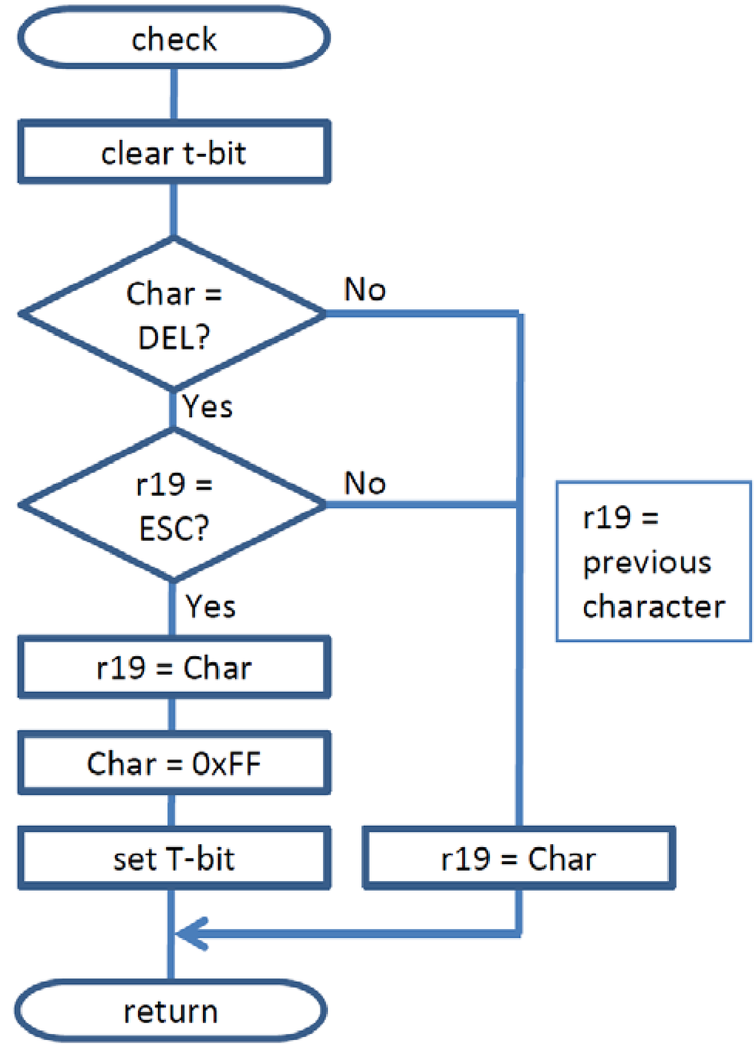

```
;>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
;    1ED022, Computer Technology I
;    Date: 2013-02-16
;    Anders Haggren
;    Function:     ?
;    Subroutine check
;<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<

check:
     clt
     cpi Char, 0x7F
     brne label_1

     cpi r19, 0x1B
     brne label_1

     mov r19, Char
     ldi Char, 0xFF
     set
     rjmp check_end

label_1:
     mov r19, Char

check_end:
     ret
```

ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[END OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

Example 2:

```
;>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
;    1ED022, Computer Technology I
;    Date: 2013-02-16
;    Anders Haggren
;    Function: Looking for sequence ESC -> DEL
;    Subroutine check
;<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<

check:
     clt
     cpi Char, 0x7F
     brne label_1

     cpi r19, 0x1B
     brne label_1

     mov r19, Char
     ldi Char, 0xFF
     set
     rjmp check_end

label_1:
     mov r19, Char

check_end:
     ret
```



Example 1, ASCII-table:

ASCII Table.

Dec	Hex	Name	Char	Ctrl-char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	0	Null	NUL	CTRL-@	32	20	Space	64	40	@	96	60	`
1	1	Start of heading	SOH	CTRL-A	33	21	!	65	41	A	97	61	a
2	2	Start of text	STX	CTRL-B	34	22	"	66	42	B	98	62	b
3	3	End of text	ETX	CTRL-C	35	23	#	67	43	C	99	63	c
4	4	End of xmit	EOT	CTRL-D	36	24	$	68	44	D	100	64	d
5	5	Enquiry	ENQ	CTRL-E	37	25	%	69	45	E	101	65	e
6	6	Acknowledge	ACK	CTRL-F	38	26	&	70	46	F	102	66	f
7	7	Bell	BEL	CTRL-G	39	27	'	71	47	G	103	67	g
8	8	Backspace	BS	CTRL-H	40	28	(	72	48	H	104	68	h
9	9	Horizontal tab	HT	CTRL-I	41	29	)	73	49	I	105	69	i
10	0A	Line feed	LF	CTRL-J	42	2A	*	74	4A	J	106	6A	j
11	0B	Vertical tab	VT	CTRL-K	43	2B	+	75	4B	K	107	6B	k
12	0C	Form feed	FF	CTRL-L	44	2C	,	76	4C	L	108	6C	l
13	0D	Carriage feed	CR	CTRL-M	45	2D	-	77	4D	M	109	6D	m
14	0E	Shift out	SO	CTRL-N	46	2E	.	78	4E	N	110	6E	n
15	0F	Shift in	SI	CTRL-O	47	2F	/	79	4F	O	111	6F	o
16	10	Data line escape	DLE	CTRL-P	48	30	0	80	50	P	112	70	p
17	11	Device control 1	DC1	CTRL-Q	49	31	1	81	51	Q	113	71	q
18	12	Device control 2	DC2	CTRL-R	50	32	2	82	52	R	114	72	r
19	13	Device control 3	DC3	CTRL-S	51	33	3	83	53	S	115	73	s
20	14	Device control 4	DC4	CTRL-T	52	34	4	84	54	T	116	74	t
21	15	Neg acknowledge	NAK	CTRL-U	53	35	5	85	55	U	117	75	u
22	16	Synchronous idle	SYN	CTRL-V	54	36	6	86	56	V	118	76	v
23	17	End of xmit block	ETB	CTRL-W	55	37	7	87	57	W	119	77	w
24	18	Cancel	CAN	CTRL-X	56	38	8	88	58	X	120	78	x
25	19	End of medium	EM	CTRL-Y	57	39	9	89	59	Y	121	79	y
26	1A	Substitute	SUB	CTRL-Z	58	3A	:	90	5A	Z	122	7A	z
27	1B	Escape	ESC	CTRL-[	59	3B	;	91	5B	[	123	7B	{
28	1C	File separator	FS	CTRL-\	60	3C	<	92	5C	\	124	7C	|
29	1D	Group separator	GS	CTRL-]	61	3D	=	93	5D	]	125	7D	}
30	1E	Record separator	RS	CTRL-^	62	3E	>	94	5E	^	126	7E	~
31	1F	Unit separator	US	CTRL-_	63	3F	?	95	5F	_	127	7F	DEL

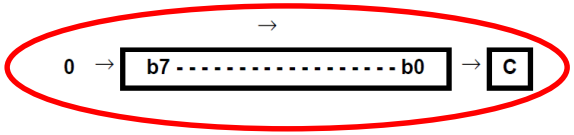
```
main:

    ldi r17, 20             ; counter = 20

    ldi ZH, HIGH(row_1)     ; pointer to row 1
    ldi ZL, LOW(row_1)

    ldi r16, 'A'            ; first character
next:
    st Z, r16               ; store character

    adiw ZL, 1              ; increase pointer

    inc r16                 ; increase char
    dec r17                 ; decrease counter
    brne next



out_text:

    ldi r17, 20

    ldi ZH, HIGH(row_1)     ; pointer to row 1
    ldi ZL, LOW(row_1)

    ldi YH, HIGH(row_3)     ; pointer to row 3
    ldi YL, LOW(row_3)

next_again:
    ld r20, Z               ; read character in row 1

    st Y, r20               ; store character in row 3

    adiw ZL, 1              ; increase pointer
    adiw YL, 1              ; increase pointer
    dec r17
    brne next_again

    rjmp main
```

LSR – Logical Shift Right

Description:

Shifts all bits in Rd one place to the right. Bit 7 is cleared. Bit 0 is loaded into the C Flag of the SREG. This operation effectively divides an unsigned value by two. The C Flag can be used to round the result.

Operation:



	Syntax:	Operands:	Program Counter:
(i)	LSR Rd	0 ≤ d ≤ 31	PC ← PC + 1

16-bit Opcode:

1001	010d	dddd	0110

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
–	–	–	⇔	⇔	0	⇔	⇔

S: N ⊕ V, For signed tests.

V: N ⊕ C (For N and C after the shift)

N: 0

Z: $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
Set if the result is $00; cleared otherwise.

C: Rd0
Set if, before the shift, the LSB of Rd was set; cleared otherwise.

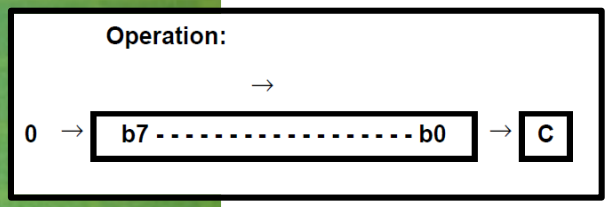
R (Result) equals Rd after the operation.

Example:

```
add     r0,r4     ; Add r4 to r0
lsr     r0        ; Divide r0 by 2
```

Words: 1 (2 bytes)
Cycles: 1

Number, positive numbers:



2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	
128	64	32	16	8	4	2	1	
1	0	1	1	0	1	1	0	
128		32	16		4	2		=182

Shift one step with lsr, logical shift right

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	
128	64	32	16	8	4	2	1	
0	1	0	1	1	0	1	1	
	64		16	8		2	1	=91

lsr, logical shift right = division of positive numbers with 2.

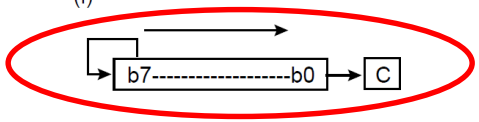
ASR – Arithmetic Shift Right

Description:

Shifts all bits in Rd one place to the right. Bit 7 is held constant. Bit 0 is loaded into the C Flag of the SREG. This operation effectively divides a signed value by two without changing its sign. The Carry Flag can be used to round the result.

Operation:

(i)



	Syntax:	Operands:	Program Counter:
(i)	ASR Rd	$0 \le d \le 31$	PC ← PC + 1

16-bit Opcode:

1001	010d	dddd	0101

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
–	–	–	⇔	⇔	⇔	⇔	⇔

S: N ⊕ V, For signed tests.

V: N ⊕ C (For N and C after the shift)

N: R7
Set if MSB of the result is set; cleared otherwise.

Z: $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
Set if the result is $00; cleared otherwise.

C: Rd0
Set if, before the shift, the LSB of Rd was set; cleared otherwise.

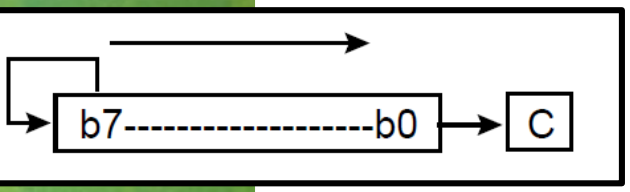
R (Result) equals Rd after the operation.

Example:

```
ldi   r16,$10   ; Load decimal 16 into r16
asr   r16       ; r16=r16 / 2
ldi   r17,$FC   ; Load -4 in r17
asr   r17       ; r17=r17/2
```

Words: 1 (2 bytes)
Cycles: 1

Number, signed numbers



(2-complement form):

-2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	
-128	64	32	16	8	4	2	1	
1	0	1	1	0	1	1	0	
-128		32	16		4	2		= - 74

Shift one step with asr, arithmetic shift right

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	
128	64	32	16	8	4	2	1	
1	1	0	1	1	0	1	1	
-128	64		16	8		2	1	= -37

asr, arithmetic shift right = division of signed numbers with 2.

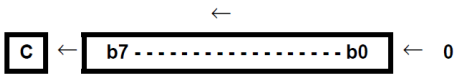
LSL – Logical Shift Left

Description:

Shifts all bits in Rd one place to the left. Bit 0 is cleared. Bit 7 is loaded into the C Flag of the SREG. This operation effectively multiplies signed and unsigned values by two.

Operation:

(i)



	Syntax:	Operands:	Program Counter:
(i)	LSL Rd	0 ≤ d ≤ 31	PC ← PC + 1

16-bit Opcode: (see ADD Rd,Rd)

0000	11dd	dddd	dddd

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
–	–	⇔	⇔	⇔	⇔	⇔	⇔

H: Rd3

S: N ⊕ V, For signed tests.

V: N ⊕ C (For N and C after the shift)

N: R7
Set if MSB of the result is set; cleared otherwise.

Z: $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
Set if the result is $00; cleared otherwise.

C: Rd7
Set if, before the shift, the MSB of Rd was set; cleared otherwise.

R (Result) equals Rd after the operation.

Example:

```
add     r0,r4     ; Add r4 to r0
lsl     r0        ; Multiply r0 by 2
```

Words: 1 (2 bytes)
Cycles: 1

Number, positive numbers:

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	
128	64	32	16	8	4	2	1	
1	0	1	1	0	1	1	0	
128		32	16		4	2		=182

Shift one step with lsl, logical shift left

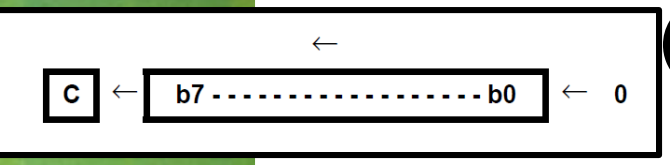
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	
128	64	32	16	8	4	2	1	
0	1	1	0	1	1	0	0	
	64	32		8	4			=108

Cy = 1 = 256. 256 + 108 = 364

lsl, logical shift left = multply with 2.

Number, signed numbers

(2-complement form):



-2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	
-128	64	32	16	8	4	2	1	
1	1	1	1	0	0	0	0	
-128	64	32	16					= - 16

Shift one step with lsl, logical shift left

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	
128	64	32	16	8	4	2	1	
1	1	1	0	0	0	0	0	
-128	64	32						= -32

lsl, logical shift left, multiply signed and unsigned values with 2.

Exam example, 2013.12.13.

3. Assembler

a) Make a flowchart of the Assembler program below. **4p**

b) Analyze the program and explain how the program works and what it will do. Input data in register R16 and output data in R1. Analyze what the output will be for all possible input values. **6p**

```
december_2013:
     push r0
     push r17
     in r17, SREG
     push r17

     ldi r17, 205
     mulsu r16, r17


     lsl r0
     brcc no_increase
     inc r1

no_increase:
     add r1, r16
     ldi r17, 32
     add r1, r17

     pop r17
     out SREG, r17
     pop r17
     pop r0
     ret
```

AVR – Little Endian

W Endianness - Wikipedia, th...

wiki/Endianness

Search

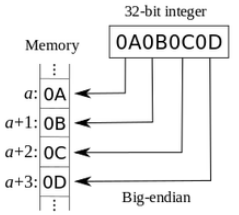
urther reading
xternal links

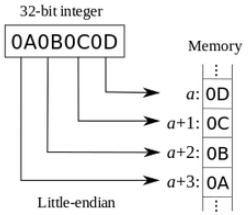
stration [edit]

ndianness may be demonstrated by writing a decimal number, say one hundred twenty-three, on paper in the usual positional notation understood by a numerate reader: The digits are written starting from the left and to the right, with the most significant digit, *1*, written first. This is analogous to the lowest address of memory being used This is an example of a big-endian convention taken from daily life.

ttle-endian way of writing the same number, one hundred twenty-three, would place the hundreds-digit *1* in the right-most position: *321*. A person following conventional ndian place-value order, who is not aware of this special ordering, would read a different number: three hundred and twenty one. Endianness in computing is similar, but it y applies to the ordering of bytes, rather than of digits.

ustrations to the right, where ***a*** is a memory address, show big-endian and little-endian storage in memory.





mology [edit]

Cohen introduced the terms Little-Endian and Big-Endian for byte ordering in an article from 1980.[2][3] In this technical and political examination of byte ordering issues, ndian" names were drawn from Jonathan Swift's 1726 satire, *Gulliver's Travels*, in which civil war erupts over whether the big or the small end of a soft-boiled egg is the r end to crack open.[4][5]

dware [edit]

uter memory consists of a sequence of storage cells. Each cell is identified in hardware and software by its memory address. If the total number of storage cells in ry is *n*, then addresses are enumerated from *0* to *n-1*. Computer programs often use data structures of fields that may consist of more data than is stored in one memory cell. For the purpose of this article its use as an operand of an instruction is relevant, a field consists of a consecutive sequence of bytes and represents a simple data value. In addition to that, it has to be of numeric type in some onal number system (mostly base-10 or base-2 — or base-256 in case of 8-bit bytes).[6] In such a number system the "value" of a digit is determined not only by its value as a single digit, but also by the on it holds in the complete number, its "significance". These positions can be mapped to memory mainly in two ways:[7]

creasing numeric significance with increasing memory addresses (or increasing time), known as *little-endian*, and
creasing numeric significance with increasing memory addresses (or increasing time), known as *big-endian*[8]

ory [edit]

the Intel microprocessor product line (most notable amongst others) has become the dominant desktop/server architecture, many historical and extant processors use a big-endian memory sentation, commonly referred to as network order, as used in the Internet protocol suite, either exclusively or as a design option; others use yet another scheme called "middle-endian", "mixed-endian" or 11-endian".

IBM System/360 used big-endian order, as do its successors System/370, ESA/390, and z/Architecture. The PDP-10 also uses big-endian addressing for byte-oriented instructions. The IBM Series/1

nd Little Endian | (Little) Endianess Guide fo... | +

blog.coldtobi.de/1_coldtobis_blog/archive/87_little_endianess_guide_for_atmel_avr.html

Search

(Little) Endianess Guide for Atmel AVR

Posted by coldtobi | 5 Jul, 2007, 16:59

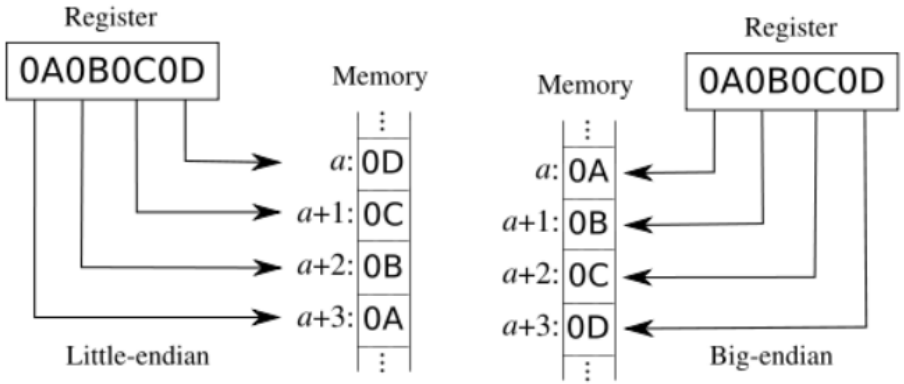
As I frequently need that information, also frequently I forget it again. (Usually, the C-Compiler has to care about it, but if you have to take over e.g a serial line, you have to know it.

> **Atmel AVR Endianess Cheat Sheet:**
>
> ```
> Example Value: 0x0A0B
>
> All 8-bt AVRs:   Little Endian     in Memory 0x0B 0x0A (*adr: 0x0B, *(adr+1): 0x0A)
>
> AVR32:           Big Endian        in Memory 0x0A 0x0B
> ```

LittleEndian is popular at Intel CPUs,

Big Endianess e.g at Motorola.

Here are some images from Wikipedia to illustrate the memory layout of the different schemes, with the example 32-bit value 0x0A0B0C0D



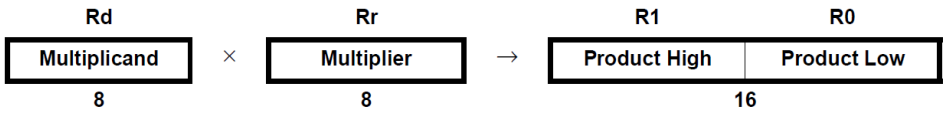
ASIP

Electronics and Atmel AVR, Tips and Tricks | Comments (2) | Trackbacks (0)

MUL – Multiply Unsigned

Description:

This instruction performs 8-bit × 8-bit → 16-bit unsigned multiplication.



The multiplicand Rd and the multiplier Rr are two registers containing unsigned numbers. The 16-bit unsigned product is placed in R1 (high byte) and R0 (low byte). Note that if the multiplicand or the multiplier is selected from R0 or R1 the result will overwrite those after multiplication.

This instruction is not available in all devices. Refer to the device specific instruction set summary.

Operation:

(i) R1:R0 ← Rd × Rr (unsigned ← unsigned × unsigned)

	Syntax:	Operands:	Program Counter:
(i)	MUL Rd,Rr	0 ≤ d ≤ 31, 0 ≤ r ≤ 31	PC ← PC + 1

16-bit Opcode:

1001	11rd	dddd	rrrr

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	⇔	⇔

C: R15
Set if bit 15 of the result is set; cleared otherwise.

Z: $\overline{R15} \bullet \overline{R14} \bullet \overline{R13} \bullet \overline{R12} \bullet \overline{R11} \bullet \overline{R10} \bullet \overline{R9} \bullet \overline{R8} \bullet \overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
Set if the result is $0000; cleared otherwise.

R (Result) equals R1,R0 after the operation.

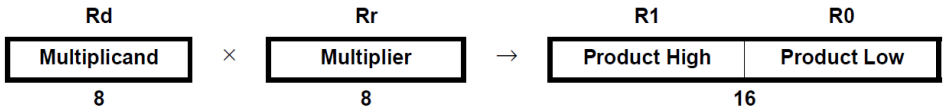
Example:

```
mul   r5,r4     ; Multiply unsigned r5 and r4
movw  r4,r0     ; Copy result back in r5:r4
```

Words: 1 (2 bytes)
Cycles: 2

MULS – Multiply Signed

Description:
This instruction performs 8-bit × 8-bit → 16-bit signed multiplication.



The multiplicand Rd and the multiplier Rr are two registers containing signed numbers. The 16-bit signed product is placed in R1 (high byte) and R0 (low byte).

This instruction is not available in all devices. Refer to the device specific instruction set summary.

Operation:

(i) R1:R0 ← Rd × Rr (signed ← signed × signed)

	Syntax:	Operands:	Program Counter:
(i)	MULS Rd,Rr	16 ≤ d ≤ 31, 16 ≤ r ≤ 31	PC ← PC + 1

16-bit Opcode:

0000	0010	dddd	rrrr

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	⇔	⇔

C: R15
Set if bit 15 of the result is set; cleared otherwise.

Z: $\overline{R15} \bullet \overline{R14} \bullet \overline{R13} \bullet \overline{R12} \bullet \overline{R11} \bullet \overline{R10} \bullet \overline{R9} \bullet \overline{R8} \bullet \overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
Set if the result is $0000; cleared otherwise.

R (Result) equals R1,R0 after the operation.

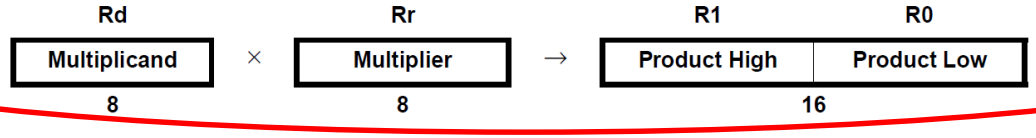
Example:

```
muls  r21,r20   ; Multiply signed r21 and r20
movw  r20,r0    ; Copy result back in r21:r20
```

Words: 1 (2 bytes)
Cycles: 2

MULSU – Multiply Signed with Unsigned

Description:
This instruction performs 8-bit × 8-bit → 16-bit multiplication of a signed and an unsigned number.



The multiplicand Rd and the multiplier Rr are two registers. The multiplicand Rd is a signed number, and the multiplier Rr is unsigned. The 16-bit signed product is placed in R1 (high byte) and R0 (low byte).

This instruction is not available in all devices. Refer to the device specific instruction set summary.

Operation:

(i) R1:R0 ← Rd × Rr (signed ← signed × unsigned)

	Syntax:	Operands:	Program Counter:
(i)	MULSU Rd,Rr	16 ≤ d ≤ 23, 16 ≤ r ≤ 23	PC ← PC + 1

16-bit Opcode:

0000	0011	0ddd	0rrr

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	⇔	⇔

C: R15
Set if bit 15 of the result is set; cleared otherwise.

Z: $\overline{R15} \bullet \overline{R14} \bullet \overline{R13} \bullet \overline{R12} \bullet \overline{R11} \bullet \overline{R10} \bullet \overline{R9} \bullet \overline{R8} \bullet \overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
Set if the result is $0000; cleared otherwise.

R (Result) equals R1,R0 after the operation.

Example:

```
;***************************************************************************
;* DESCRIPTION
;*Signed multiply of two 16-bit numbers with 32-bit result.
```

3. Assembler

a) Make a flowchart of the Assembler program below.

b) Analyze the program and explain how the program works and what it will do. Input data i
R16 and output data in R1. Analyze what the output will be for all possible input values.

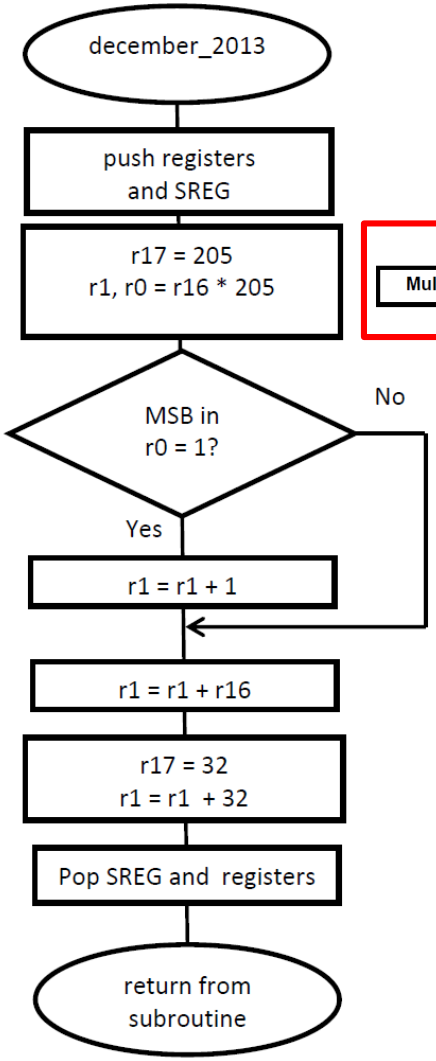
```
december_2013:
     push r0
     push r17
     in r17, SREG
     push r17

     ldi r17, 205
     mulsu r16, r17


     lsl r0
     brcc no_increase
     inc r1

no_increase:
     add r1, r16
     ldi r17, 32
     add r1, r17

     pop r17
     out SREG, r17
     pop r17
     pop r0
     ret
```

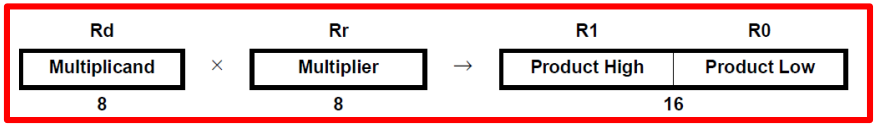
3. Assembler

a) Make a flowchart of the Assembler program below.

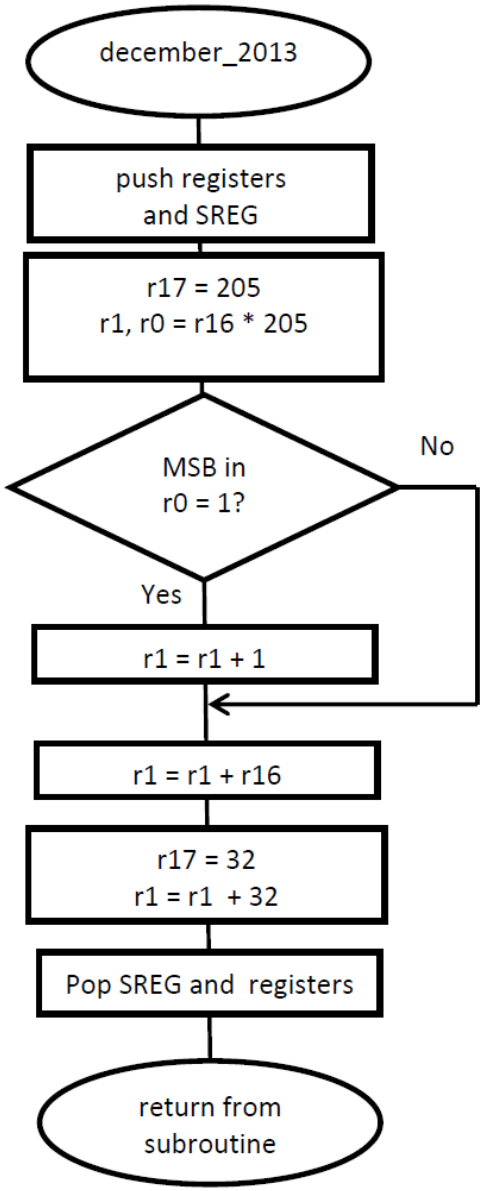
b) Analyze the program and explain how the program works and what it will do. Input data i
R16 and output data in R1. Analyze what the output will be for all possible input values.

```
december_2013:
     push r0
     push r17
     in r17, SREG
     push r17

     ldi r17, 205
     mulsu r16, r17


     lsl r0
     brcc no_increase
     inc r1

no_increase:
     add r1, r16
     ldi r17, 32
     add r1, r17

     pop r17
     out SREG, r17
     pop r17
     pop r0
     ret
```





Exam example, 2013.12.13.



Celsius temperature conversion formulae

	from Celsius	to Celsius
Fahrenheit	[°F] = [°C] × 9⁄5 + 32	[°C] = ([°F] − 32) × 5⁄9
Kelvin	[K] = [°C] + 273.15	[°C] = [K] − 273.15
Rankine	[°R] = ([°C] + 273.15) × 9⁄5	[°C] = ([°R] − 491.67) × 5⁄9

R16 = input value, °C
R1 = r16 * 205/256

(skip least significant byte =
right shift 8 bits = division with 256)
Check i f most significant bit
in least significant byte, = 1.

If so, add 1 to most significant byte, r1.
(round up)

r1 = r1 + r16

r1 = r1 + 32

R1 = r16 + r16*205/ 256 + 32, dvs

°F = °C + °C*205/256 + 32,
Approximately the same as:

°F = °C + °C*4/5 + 32 = °C*9/5 + 32 dvs
°F = 1,8 * °C + 32

3. Assembler

a) Make a flowchart of the Assembler program below. **4p**

b) Analyze the program and explain how the program works and what it will do. Input data in register R16 and output data in R1. Analyze what the output will be for all possible input values. **6p**

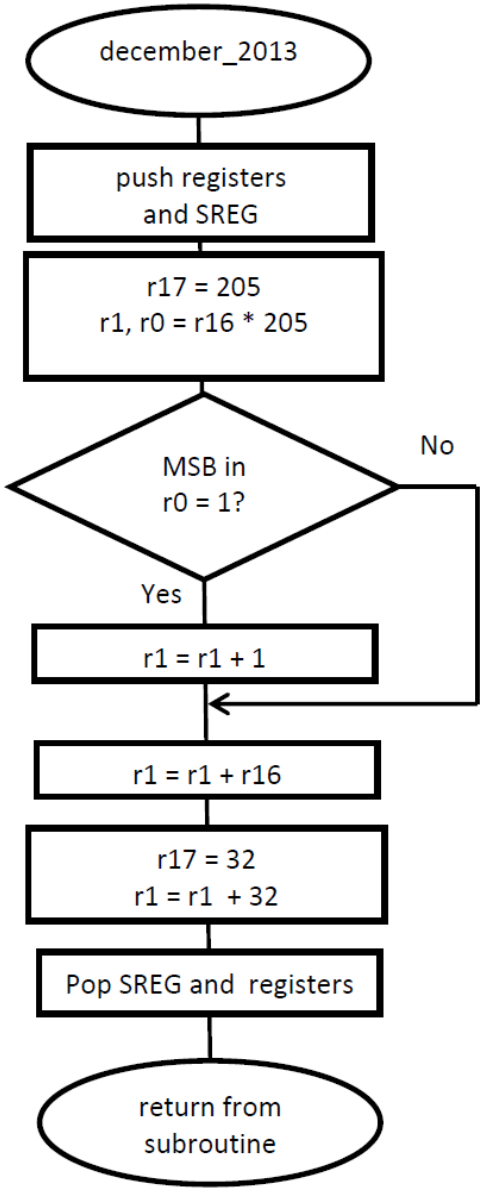
```
;>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
;    1ED022, Computer Technology I
;    Date: 2013-12-11
;    Anders Haggren
;    Function:
;    Subroutine: Celsius to Fahrenheit conversion
;    Input value in r16, -20 to 100, two's complement form
;    return value in r1, 0 to 200,
;
;    Calculation:
;    F = C + C*205/256 + 32 = C*461/256 + 32
;
;    Approximately the same as:
;    F = C*9/5 + 32 (Correct formula for transfer from C to F)
;
;<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
```

Celsius temperature conversion formulae

	from Celsius	to Celsius
Fahrenheit	[°F] = [°C] × 9⁄5 + 32	[°C] = ([°F] − 32) × 5⁄9
Kelvin	[K] = [°C] + 273.15	[°C] = [K] − 273.15
Rankine	[°R] = ([°C] + 273.15) × 9⁄5	[°C] = ([°R] − 491.67) × 5⁄9

```
december_2013:
     push r0
     push r17
     in r17, SREG
     push r17

     ldi r17, 205
     mulsu r16, r17      ; C*205 to r1, r0

; check if LS byte greater or equal 0x80, if so round up MS byte

     lsl r0              ; shift msb to cy-flag
     brcc no_increase
     inc r1              ; round up if cy = 1, r1 = C*205/256

no_increase:
     add r1, r16         ; r1 = C + C*205/256
     ldi r17, 32
     add r1, r17         ; r1 = C + C*205/8 + 32

     pop r17
     out SREG, r17
     pop r17
     pop r0
     ret
```

Exam example, 2013.12.13.



R16 = input value, ^{o}C
R1 = r16 * 205/256

(skip least significant byte =
right shift 8 bits = division with 256)
Check i f most significant bit
in least significant byte, = 1.

If so, add 1 to most significant byte, r1.
(round up)

r1 = r1 + r16

r1 = r1 + 32

R1 = r16 + r16*205/ 256 + 32, dvs

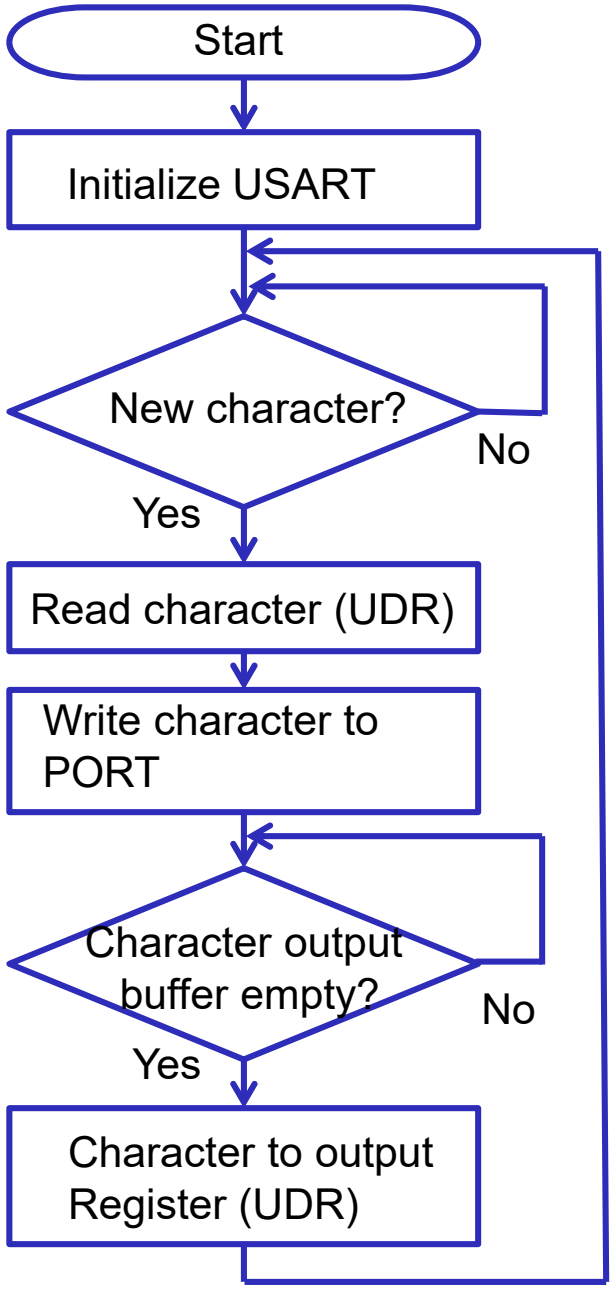
^{o}F = ^{o}C + ^{o}C*205/256 + 32,
Approximately the same as:

^{o}F = ^{o}C + ^{o}C*4/5 + 32 = ^{o}C*9/5 + 32 dvs
^{o}F = 1,8 * ^{o}C + 32

Flowcharts!

NOTE!
Written report of laboratory work to be done for all tasks of each group.

The description shall contain program code, flow chart and description of program code. The source code should be well commented. Each item in each lab should be reported. The reports will be compared in the antiplagiarism system Urkund. Note that copying of code, text or other material between groups or from other sources is considered cheating and will be reported to the Disciplinary Committee. Keep in mind that the report takes the time to write - do not wait to start writing!



1DT301, Computer Technology
Timers and USART
Wednesday, September 30, 2015

- Introduction to lab 4
- Code examples for lab 4
- Introduction to lab 5

AVR Assembler User Guide

Secti

AVR Assembler User G

4.1 Introduction

Welcome to the Atmel *AVR* Assembler. This manual describes the usage of th
bler. The Assembler covers the whole range of microcontrollers in the AT90S

The Assembler translates assembly source code into object code. The genera
code can be used as input to a simulator or an emulator such as the Atmel *A*
cuit Emulator. The Assembler also generates a PROMable code and ar
EEPROM file which can be programmed directly into the program me
EEPROM memory of an *AVR* microcontroller.

The Assembler generates fixed code allocations, consequently no linking is ne

The Assembler runs under Microsoft Windows 3.11, Microsoft Windo
Microsoft Windows NT. In addition, there is an MS-DOS command line ver
Windows version of the program contains an on-line help function covering m
document.

shift left, shift right

4.6.3.8 Shift left

Symbol: <<

Description: Binary operator which returns the left expression shifted left a number of times given by the right expression

Precedence: 11

Example:
```
ldi    r17,1<<bitmask ;Load r17 with 1 shifted
                      ;left bitmask times
```

4.6.3.9 Shift right

Symbol: >>

Description: Binary operator which returns the left expression shifted right a number of times given by the right expression.

Precedence: 11

Example:
```
ldi    r17,c1>>c2     ;Load r17 with c1 shifted
                      ;right c2 times
```

bitwise AND, XOR, OR

4.6.3.16 Bitwise And

Symbol: &

Description: Binary operator which returns the bitwise And between two expressions

Precedence: 8

Example:
```
ldi    r18,High(c1&c2)    ;Load r18 with an expression
```

4.6.3.17 Bitwise Xor

Symbol: ^

Description: Binary operator which returns the bitwise Exclusive Or between two expressions

Precedence: 7

Example:
```
ldi    r18,Low(c1^c2)     ;Load r18 with an expression
```

4.6.3.18 Bitwise Or

Symbol: |

Description: Binary operator which returns the bitwise Or between two expressions

Precedence: 6

Example:
```
ldi    r18,Low(c1|c2)     ;Load r18 with an expression
```

logical and, logical or

4.6.3.19 Logical And

Symbol: &&

Description: Binary operator which returns 1 if the expressions are both nonzero, 0 otherwise

Precedence: 5

Example:
```
ldi    r18,Low(c1&&c2)    ;Load r18 with an expression
```

4.6.3.20 Logical Or

Symbol: ||

Description: Binary operator which returns 1 if one or both of the expressions are nonzero, 0 otherwise

Precedence: 4

Example:
```
ldi    r18,Low(c1||c2)    ;Load r18 with an expression
```

doc2549_ATmega2560

ATMega2560_8-bit Microcontroller

Features

- High Performance, Low Power Atmel® AVR® 8-Bit Microcontroller
- Advanced RISC Architecture
  - 135 Powerful Instructions – Most Single Clock Cycle Execution
  - 32 × 8 General Purpose Working Registers
  - Fully Static Operation
  - Up to 16 MIPS Throughput at 16MHz
  - On-Chip 2-cycle Multiplier
- High Endurance Non-volatile Memory Segments
  - 64K/128K/256KBytes of In-System Self-Programmable Flash
  - 4Kbytes EEPROM
  - 8Kbytes Internal SRAM
  - Write/Erase Cycles:10,000 Flash/100,000 EEPROM
  - Data retention: 20 years at 85°C/ 100 years at 25°C
  - Optional Boot Code Section with Independent Lock Bits
    - In-System Programming by On-chip Boot Program
    - True Read-While-Write Operation
  - Programming Lock for Software Security
    - Endurance: Up to 64Kbytes Optional External Memory Space
- Atmel® QTouch® library support
  - Capacitive touch buttons, sliders and wheels
  - QTouch and QMatrix® acquisition
  - Up to 64 sense channels
- JTAG (IEEE std. 1149.1 compliant) Interface
  - Boundary-scan Capabilities According to the JTAG Standard
  - Extensive On-chip Debug Support
  - Programming of Flash, EEPROM, Fuses, and Lock Bits through the JTAG Interface
- Peripheral Features
  - Two 8-bit Timer/Counters with Separate Prescaler and Compare Mode
  - Four 16-bit Timer/Counter with Separate Prescaler, Compare- and Capture Mode
  - Real Time Counter with Separate Oscillator
  - Four 8-bit PWM Channels
  - Six/Twelve PWM Channels with Programmable Resolution from 2 to 16 Bits (ATmega1281/2561, ATmega640/1280/2560)
  - Output Compare Modulator
  - 8/16-channel, 10-bit ADC (ATmega1281/2561, ATmega640/1280/2560)
  - Two/Four Programmable Serial USART (ATmega1281/2561, ATmega640/1280/2560)
  - Master/Slave SPI Serial Interface
  - Byte Oriented 2-wire Serial Interface
  - Programmable Watchdog Timer with Separate On-chip Oscillator
  - On-chip Analog Comparator
  - Interrupt and Wake-up on Pin Change
- Special Microcontroller Features
  - Power-on Reset and Programmable Brown-out Detection
  - Internal Calibrated Oscillator
  - External and Internal Interrupt Sources
  - Six Sleep Modes: Idle, ADC Noise Reduction, Power-save, Power-down, Standby, and Extended Standby
- I/O and Packages
  - 54/86 Programmable I/O Lines (ATmega1281/2561, ATmega640/1280/2560)
  - 64-pad QFN/MLF, 64-lead TQFP (ATmega1281/2561)
  - 100-lead TQFP, 100-ball CBGA (ATmega640/1280/2560)
  - RoHS/Fully Green

ATMEL

8-bit Atmel Microcontroller with 64K/128K/256K Bytes In-System Programmable Flash

ATmega640/V
ATmega1280/V
ATmega1281/V
ATmega2560/V
ATmega2561/V

Preliminary

doc2466_ATmega16.pdf

Timer/Counter Interrupt Mask Register – TIMSK

Bit	7	6	5	4	3	2	1	0	
	OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	OCIE0	TOIE0	TIMSK
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	

- **Bit 1 – OCIE0: Timer/Counter0 Output Compare Match Interrupt Enable**

When the OCIE0 bit is written to one, and the I-bit in the Status Register is set (one), the Timer/Counter0 Compare Match interrupt is enabled. The corresponding interrupt is executed if a compare match in Timer/Counter0 occurs, i.e., when the OCF0 bit is set in the Timer/Counter Interrupt Flag Register – TIFR.

- **Bit 0 – TOIE0: Timer/Counter0 Overflow Interrupt Enable**

When the TOIE0 bit is written to one, and the I-bit in the Status Register is set (one), the Timer/Counter0 Overflow interrupt is enabled. The corresponding interrupt is executed if an overflow in Timer/Counter0 occurs, i.e., when the TOV0 bit is set in the Timer/Counter Interrupt Flag Register – TIFR.

m16def.inc

```
; TIMSK - Timer/Counter Interrupt Mask Register
.equ    TOIE0   = 0     ; Timer/Counter0 Overflow Interrupt Enable
.equ    OCIE0   = 1     ; Timer/Counter0 Output Compare Match Interrupt register

; TIFR - Timer/Counter Interrupt Flag register
.equ    TOV0    = 0     ; Timer/Counter0 Overflow Flag
.equ    OCF0    = 1     ; Output Compare Flag 0

; SFIOR - Special Function IO Register
.equ    PSR10   = 0     ; Prescaler Reset Timer/Counter1 and Timer/Counter0
```

ldi

```
ldi r16, 0b 0000 0011
ldi r16, 0x 03
out TIMSK, r16

ldi r16, 1<<OCIE0 | 1<<TOIE0
out TIMSK, r16

clr r16
sbr r16, 1
sbr r16, 2
out TIMSK, r16
```

Lab 4

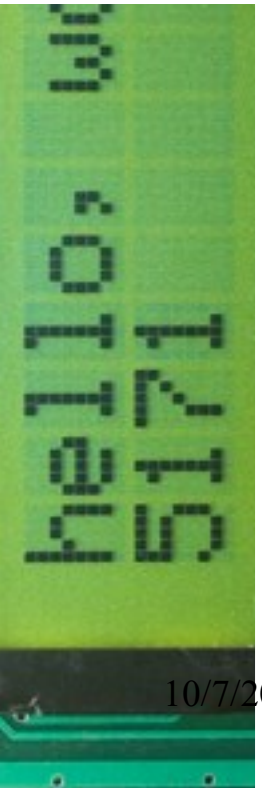
- Square wave generator
- PWM
- Serial communication, RS 232

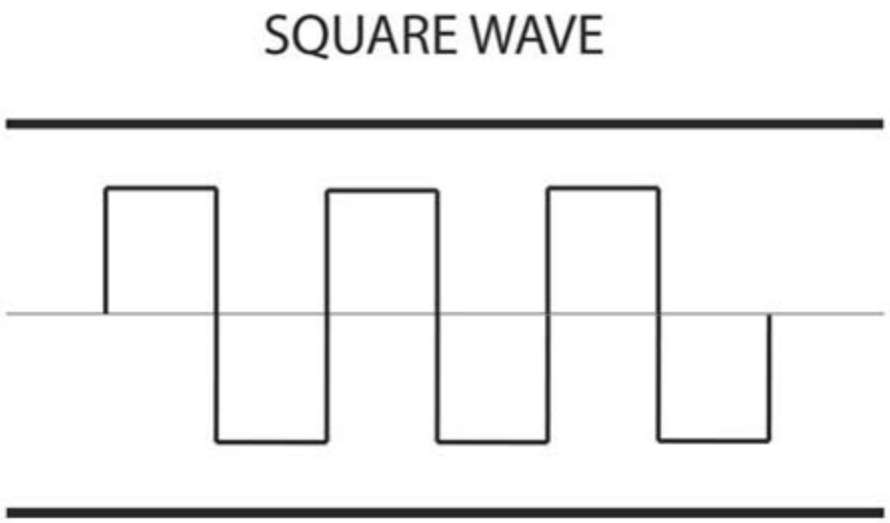
Lab 4, task 1:

Task1: Square wave generator

Write a program in Assembly that creates a square wave. One LED should be connected and switch with the frequency 1 Hz. Duty cycle 50%. (On: 0.5 sec, Off: 0.5 sec.)

Use the timer function to create an interrupt with 2 Hz, which change between On and Off in the interrupt subroutine.





Lab 4, task 1:

- Create an interrupt using a 8-bit timer, for example TIMER0.
- Select a prescaler value, that will divide the oscillator frequency. Example: osc. = 1 MHz, prescaler = 1024 => timer counts every ms. (1000 times / second)
- Let the timer give an interrupt with a delay that you choose, for example 10 ms.
  Maximum value is 255 = 255 ms.
- To get a delay of 500 ms, count 50 timerinterrupts, and toggle the LED.
- Alternatively, use a 16-bit timer, for example TIMER/COUNTER1, which can count up to 65 535 = 65 535 ms = 65,5 s.

TIMER0

interrupt vector table (ATMega2560)

```
; ***** INTERRUPT VECTORS ********************************************
.equ    INT0addr        = 0x0002        ; External Interrupt Request 0
.equ    INT1addr        = 0x0004        ; External Interrupt Request 1
.equ    INT2addr        = 0x0006        ; External Interrupt Request 2
.equ    INT3addr        = 0x0008        ; External Interrupt Request 3
.equ    INT4addr        = 0x000a        ; External Interrupt Request 4
.equ    INT5addr        = 0x000c        ; External Interrupt Request 5
.equ    INT6addr        = 0x000e        ; External Interrupt Request 6
.equ    INT7addr        = 0x0010        ; External Interrupt Request 7
.equ    PCI0addr        = 0x0012        ; Pin Change Interrupt Request 0
.equ    PCI1addr        = 0x0014        ; Pin Change Interrupt Request 1
.equ    PCI2addr        = 0x0016        ; Pin Change Interrupt Request 2
.equ    WDTaddr = 0x0018        ; Watchdog Time-out Interrupt
.equ    OC2Aaddr        = 0x001a        ; Timer/Counter2 Compare Match A
.equ    OC2Baddr        = 0x001c        ; Timer/Counter2 Compare Match B
.equ    OVF2addr        = 0x001e        ; Timer/Counter2 Overflow
.equ    ICP1addr        = 0x0020        ; Timer/Counter1 Capture Event
.equ    OC1Aaddr        = 0x0022        ; Timer/Counter1 Compare Match A
.equ    OC1Baddr        = 0x0024        ; Timer/Counter1 Compare Match B
.equ    OC1Caddr        = 0x0026        ; Timer/Counter1 Compare Match C
.equ    OVF1addr        = 0x0028        ; Timer/Counter1 Overflow
.equ    OC0Aaddr        = 0x002a        ; Timer/Counter0 Compare Match A
.equ    OC0Baddr        = 0x002c        ; Timer/Counter0 Compare Match B
.equ    OVF0addr        = 0x002e        ; Timer/Counter0 Overflow
.equ    SPIaddr = 0x0030        ; SPI Serial Transfer Complete
.equ    URXC0addr       = 0x0032        ; USART0, Rx Complete
.equ    UDRE0addr       = 0x0034        ; USART0 Data register Empty
.equ    UTXC0addr       = 0x0036        ; USART0, Tx Complete
```

TIMER0
initialization

```
.org 0x30
restart:
        ldi temp, LOW(RAMEND)        ; initialize SP, Stackpointer
        out SPL, temp
        ldi temp, HIGH(RAMEND)
        out SPH, temp

        ldi temp, 0x01               ; initialize DDRB
        out DDRB, temp
```

In the file m16def.inc, we can find:

```
.equ    RAMEND =$45F
```

TIMER0
initialization

```
        ldi temp, 0x05          ; prescaler value to TCCR0
        out TCCR0, temp         ; CS2 - CS2 = 101, osc.clock / 1024

        ldi temp, (1<<TOIE0)    ; Timer 0 enable flag, TOIE0
        out TIMSK, temp         ; to register TIMSK

        ldi temp, 100           ; starting value for counter
        out TCNT0, temp         ; counter register

        sei                     ; enable global interrupt

start:
        rjmp start              ; main loop
```

TIMER0
interrupt routine

```
timer0_int:
        push temp                       ; timer interrupt routine
        in temp, SREG                   ; save SREG on stack
        push temp

; additional code to create the square output

        ldi temp, 100                   ; starting value for counter
        out TCNT0, temp

        pop temp                        ; restore SREG
        out SREG, temp
        pop temp                        ; restore register
        reti                            ; return from interrupt
```

TIMER0 – TCCR0

Timer/Counter Control Register – TCCR0

Bit	7	6	5	4	3	2	1	0	
	FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00	TCCR0
Read/Write	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	

- **Bit 2:0 – CS02:0: Clock Select**

The three Clock Select bits select the clock source to be used by the Timer/Counter.

Table 42. Clock Select Bit Description

CS02	CS01	CS00	Description
0	0	0	No clock source (Timer/Counter stopped).
0	0	1	$clk_{I/O}$/(No prescaling)
0	1	0	$clk_{I/O}$/8 (From prescaler)
0	1	1	$clk_{I/O}$/64 (From prescaler)
1	0	0	$clk_{I/O}$/256 (From prescaler)
1	0	1	$clk_{I/O}$/1024 (From prescaler)
1	1	0	External clock source on T0 pin. Clock on falling edge.
1	1	1	External clock source on T0 pin. Clock on rising edge.

If external pin modes are used for the Timer/Counter0, transitions on the T0 pin will clock the counter even if the pin is configured as an output. This feature allows software control of the counting.

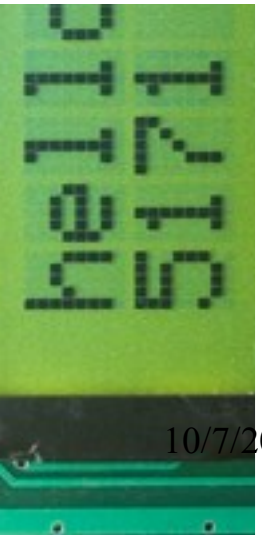
TIMER0 - TIMSK

Timer/Counter Interrupt Mask Register – TIMSK

Bit	7	6	5	4	3	2	1	0	
	OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	OCIE0	TOIE0	TIMSK
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	

- **Bit 1 – OCIE0: Timer/Counter0 Output Compare Match Interrupt Enable**

When the OCIE0 bit is written to one, and the I-bit in the Status Register is set (one), the Timer/Counter0 Compare Match interrupt is enabled. The corresponding interrupt is executed if a compare match in Timer/Counter0 occurs, that is, when the OCF0 bit is set in the Timer/Counter Interrupt Flag Register – TIFR.



- **Bit 0 – TOIE0: Timer/Counter0 Overflow Interrupt Enable**

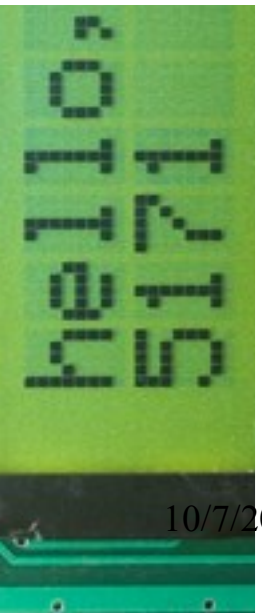
When the TOIE0 bit is written to one, and the I-bit in the Status Register is set (one), the Timer/Counter0 Overflow interrupt is enabled. The corresponding interrupt is executed if an overflow in Timer/Counter0 occurs, that is, when the TOV0 bit is set in the Timer/Counter Interrupt Flag Register – TIFR.

TIMER0 – TCNT0

Timer/Counter Register – TCNT0

Bit	7	6	5	4	3	2	1	0	
	TCNT0[7:0]								TCNT0
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	

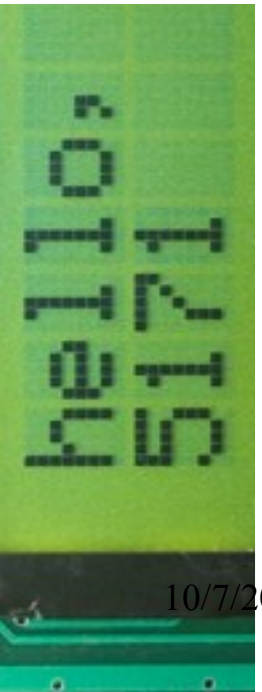
The Timer/Counter Register gives direct access, both for read and write operations, to the Timer/Counter unit 8-bit counter. Writing to the TCNT0 Register blocks (removes) the compare match on the following timer clock. Modifying the counter (TCNT0) while the counter is running, introduces a risk of missing a compare match between TCNT0 and the OCR0 Register.



Lab 4, task 2:

Task 2: Pulse Width Modulation (PWM)

Modify the program in Task 1 to obtain Pulse Width Modulation (PWM). The frequency should be fixed, but the duty cycle should be possible to change. Use two push buttons to change the duty cycle up and down. Use interrupt for each pushbutton. The duty cycle should be possible to change from 0 % up to 100 % in steps of 5 %. Connect the output to an oscilloscope, to visualize the change in duty cycle.



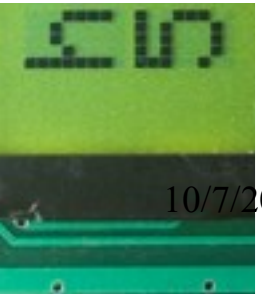
Lab 4, task 3, 4:

Task 3: Serial communication

Write a program in Assembly that uses the serial communication port0 (RS232). Connect a computer to the serial port and use a terminal emulation program. (Ex. Hyper Terminal)
The program should receive characters that are sent from the computer, and show the code on the LEDs. For example, if you send character A, it has the hex code $65, the bit pattern is 0110 0101 and should be displayed with LEDs On for each 'one'. Use polled UART, which means that the UART should be checked regularly by the program.

Task 4: Serial communication with echo

Modify the program in task 3 to obtain an echo, which means that the received character should also be sent back to the terminal, This could be used as a confirmation in the terminal, to ensure that the character has been transferred correctly.

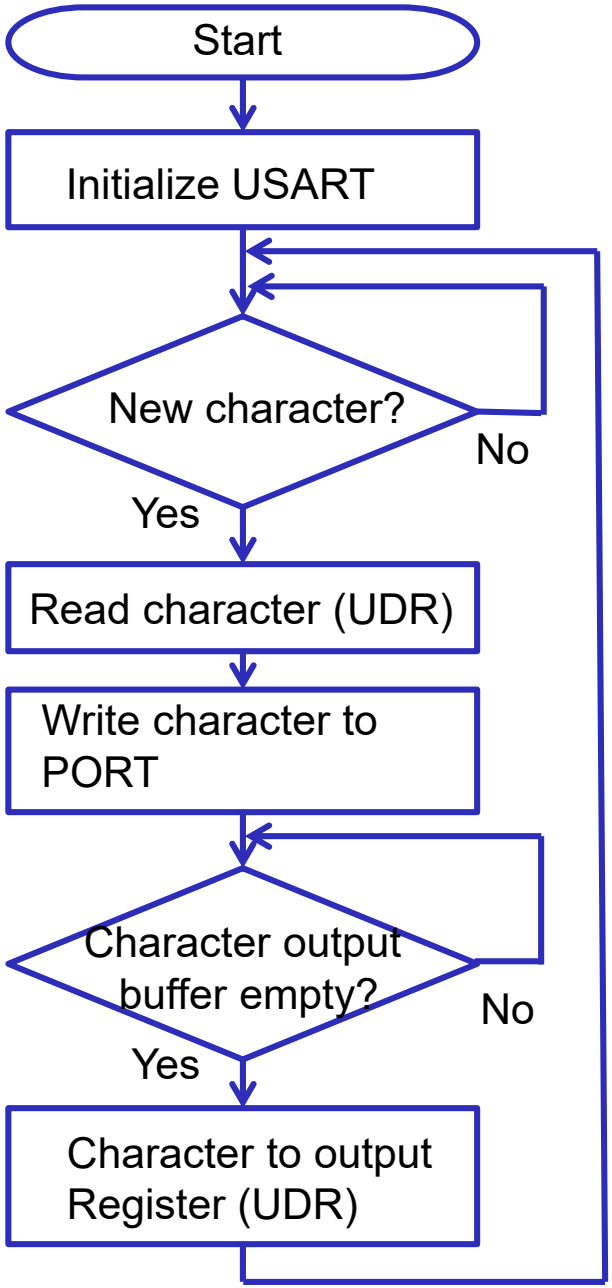


Lab 4, task 5:

Task 5: Serial communication using Interrupt
Do task 3 and 4, but use Interrupt instead of polled UART.
(USART, Rx Complete, USART Data Register Empty and USART, Tx Complete)

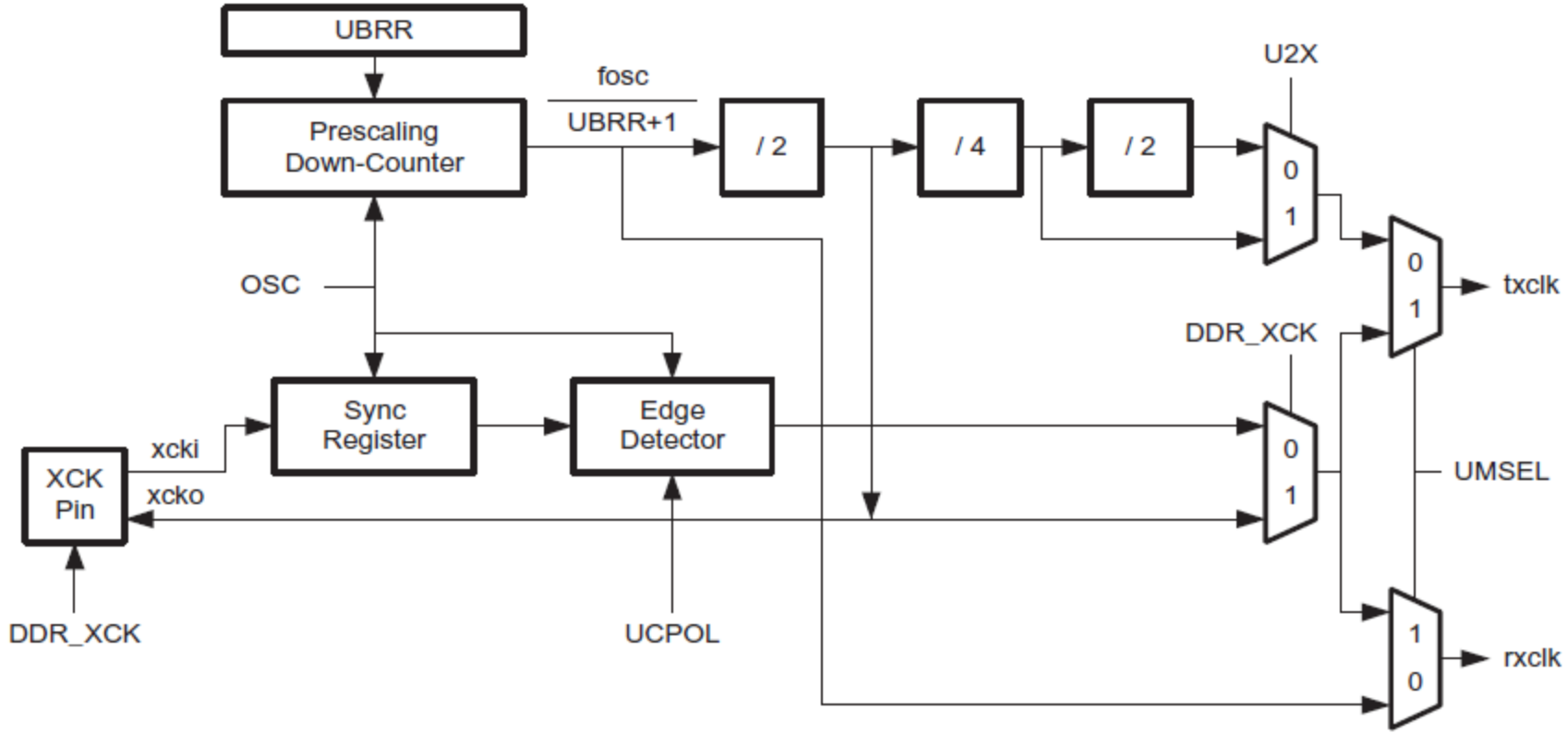
Lab 4, task 3

Example,
polled reading of
serial port.



USART

Figure 70. Clock Generation Logic, Block Diagram



USART

Table 60. Equations for Calculating Baud Rate Register Setting

Operating Mode	Equation for Calculating Baud Rate[1]	Equation for Calculating UBRR Value
Asynchronous Normal Mode (U2X = 0)	$BAUD = \frac{f_{OSC}}{16(UBRR+1)}$	$UBRR = \frac{f_{OSC}}{16BAUD} - 1$
Asynchronous Double Speed Mode (U2X = 1)	$BAUD = \frac{f_{OSC}}{8(UBRR+1)}$	$UBRR = \frac{f_{OSC}}{8BAUD} - 1$
Synchronous Master Mode	$BAUD = \frac{f_{OSC}}{2(UBRR+1)}$	$UBRR = \frac{f_{OSC}}{2BAUD} - 1$

Note: 1. The baud rate is defined to be the transfer rate in bit per second (bps).

BAUD Baud rate (in bits per second, bps)

f_{OSC} System Oscillator clock frequency

UBRR Contents of the UBRRH and UBRRL Registers, (0 - 4095)

Some examples of UBRR values for some system clock frequencies are found in Table 68 (see page 168).

USART

Table 68. Examples of UBRR Settings for Commonly Used Oscillator Frequencies

Baud Rate (bps)	f_{osc} = 1.0000 MHz				f_{osc} = 1.8432 MHz				f_{osc} = 2.0000 MHz			
	U2X = 0		U2X = 1		U2X = 0		U2X = 1		U2X = 0		U2X = 1	
	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error
2400	25	0.2%	51	0.2%	47	0.0%	95	0.0%	51	0.2%	103	0.2%
4800	12	0.2%	25	0.2%	23	0.0%	47	0.0%	25	0.2%	51	0.2%
9600	6	-7.0%	12	0.2%	11	0.0%	23	0.0%	12	0.2%	25	0.2%
14.4k	3	8.5%	8	-3.5%	7	0.0%	15	0.0%	8	-3.5%	16	2.1%
19.2k	2	8.5%	6	-7.0%	5	0.0%	11	0.0%	6	-7.0%	12	0.2%
28.8k	1	8.5%	3	8.5%	3	0.0%	7	0.0%	3	8.5%	8	-3.5%
38.4k	1	-18.6%	2	8.5%	2	0.0%	5	0.0%	2	8.5%	6	-7.0%
57.6k	0	8.5%	1	8.5%	1	0.0%	3	0.0%	1	8.5%	3	8.5%
76.8k	–	–	1	-18.6%	1	-25.0%	2	0.0%	1	-18.6%	2	8.5%
115.2k	–	–	0	8.5%	0	0.0%	1	0.0%	0	8.5%	1	8.5%
230.4k	–	–	–	–	–	–	0	0.0%	–	–	–	–
250k	–	–	–	–	–	–	–	–	–	–	0	0.0%
Max (1)	62.5 Kbps		125 Kbps		115.2 Kbps		230.4 Kbps		125 Kbps		250 Kbps	

1. UBRR = 0, Error = 0.0%

UBRR

- The prescaler value stored in UBRR is a value that depends on oscillator frequency and selected BAUD rate.
- If the value is lower than 255, it will fit in one register, UBRRL. (UBRR = UBRRL in ATmega16)
- Example:
  BAUD rate is 4800 bps,
  oscillator frequency is 1 MHz => UBRR is 12

UBRR

```
.equ UBRR_val = 12              ; osc. = 1MHz, BAUD rate 4800 bps

        ldi Temp, UBRR_val      ; store Prescaler value in UBBRRL
        out UBRRL, Temp

.def Temp = r17

        ldi Temp, (1<<TXEN) | (1<<RXEN)
        out UCR, Temp           ; set TX and RX Enable flags
```

USART

Assembly Code Example[1]

```
USART_Init:
  ; Set baud rate
  out   UBRRH, r17
  out   UBRRL, r16
  ; Enable receiver and transmitter
  ldi   r16, (1<<RXEN)|(1<<TXEN)
  out   UCSRB,r16
  ; Set frame format: 8data, 2stop bit
  ldi   r16, (1<<URSEL)|(1<<USBS)|(3<<UCSZ0)
  out   UCSRC,r16
  ret
```

USART

USART Control and Status Register B – UCSRB

Bit	7	6	5	4	3	2	1	0	
	RXCIE	TXCIE	UDRIE	RXEN	TXEN	UCSZ2	RXB8	TXB8	UCSRB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7 – RXCIE: RX Complete Interrupt Enable**

Writing this bit to one enables interrupt on the RXC Flag. A USART Receive Complete Interrupt will be generated only if the RXCIE bit is written to one, the Global Interrupt Flag in SREG is written to one and the RXC bit in UCSRA is set.

- **Bit 6 – TXCIE: TX Complete Interrupt Enable**

Writing this bit to one enables interrupt on the TXC Flag. A USART Transmit Complete Interrupt will be generated only if the TXCIE bit is written to one, the Global Interrupt Flag in SREG is written to one and the TXC bit in UCSRA is set.

- **Bit 5 – UDRIE: USART Data Register Empty Interrupt Enable**

Writing this bit to one enables interrupt on the UDRE Flag. A Data Register Empty Interrupt will be generated only if the UDRIE bit is written to one, the Global Interrupt Flag in SREG is written to one and the UDRE bit in UCSRA is set.

- **Bit 4 – RXEN: Receiver Enable**

Writing this bit to one enables the USART Receiver. The Receiver will override normal port operation for the RxD pin when enabled. Disabling the Receiver will flush the receive buffer invalidating the FE, DOR, and PE Flags.

USART

USART Control and Status Register B – UCSRB

Bit	7	6	5	4	3	2	1	0	
	RXCIE	TXCIE	UDRIE	RXEN	TXEN	UCSZ2	RXB8	TXB8	UCSRB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 3 – TXEN: Transmitter Enable**

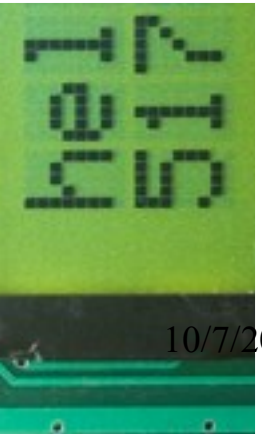
Writing this bit to one enables the USART Transmitter. The Transmitter will override normal port operation for the TxD pin when enabled. The disabling of the Transmitter (writing TXEN to zero) will not become effective until ongoing and pending transmissions are completed, that is, when the transmit Shift Register and transmit Buffer Register do not contain data to be transmitted. When disabled, the transmitter will no longer override the TxD port.

- **Bit 2 – UCSZ2: Character Size**

The UCSZ2 bits combined with the UCSZ1:0 bit in UCSRC sets the number of data bits (Character Size) in a frame the receiver and transmitter use.

- **Bit 1 – RXB8: Receive Data Bit 8**

RXB8 is the ninth data bit of the received character when operating with serial frames with nine data bits. Must be read before reading the low bits from UDR.



USART

USART Control and Status Register C – UCSRC

Bit	7	6	5	4	3	2	1	0	
	URSEL	UMSEL	UPM1	UPM0	USBS	UCSZ1	UCSZ0	UCPOL	UCSRC
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	1	0	0	0	0	1	1	0	

The UCSRC Register shares the same I/O location as the UBRRH Register. See the "Accessing UBRRH/ UCSRC Registers" on page 162 section which describes how to access this register.

- **Bit 7 – URSEL: Register Select**

This bit selects between accessing the UCSRC or the UBRRH Register. It is read as one when reading UCSRC. The URSEL must be one when writing the UCSRC.

- **Bit 6 – UMSEL: USART Mode Select**

This bit selects between Asynchronous and Synchronous mode of operation.

Table 63. UMSEL Bit Settings

UMSEL	Mode
0	Asynchronous Operation
1	Synchronous Operation

USART

```
.equ    UDR     =$0c
.equ    UCSRA   =$0b
.equ    USR     =$0b    ; For compatibility with S8535
.equ    UCSRB   =$0a
.equ    UCR     =$0a    ; For compatibility with S8535
.equ    UCSRC   =$20    ; Note! UCSRC equals UBRRH
.equ    UBRR    =$09
.equ    UBRRL   =$09    ; New name for UBRR
```

Dvs:

UBRR = UBRRL = $ 09
UCR = UCSRB = $ 0A
USR = UCSRA = $ 0B
UDR = $ 0C
UCRC = UBRRH = $ 20

UDR

USART Register Description

USART I/O Data Register – UDR

Bit	7	6	5	4	3	2	1	0	
	RXB[7:0]								UDR (Read)
	TXB[7:0]								UDR (Write)
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	

The USART Transmit Data Buffer Register and USART Receive Data Buffer Registers share the same I/O address referred to as USART Data Register or UDR. The Transmit Data Buffer Register (TXB) will be the destination for data written to the UDR Register location. Reading the UDR Register location will return the contents of the Receive Data Buffer Register (RXB).

For 5-bit, 6-bit, or 7-bit characters the upper unused bits will be ignored by the Transmitter and set to zero by the Receiver.

The transmit buffer can only be written when the UDRE Flag in the UCSRA Register is set. Data written to UDR when the UDRE Flag is not set, will be ignored by the USART Transmitter. When data is written to the transmit buffer, and the Transmitter is enabled, the Transmitter will load the data into the transmit Shift Register when the Shift Register is empty. Then the data will be serially transmitted on the TxD pin.

The receive buffer consists of a two level FIFO. The FIFO will change its state whenever the receive buffer is accessed. Due to this behavior of the receive buffer, do not use read modify write instructions (SBI and CBI) on this location. Be careful when using bit test instructions (SBIC and SBIS), since these also will change the state of the FIFO.

UCSRA - USR

USART Control and Status Register A – UCSRA

Bit	7	6	5	4	3	2	1	0	
	RXC	TXC	UDRE	FE	DOR	PE	U2X	MPCM	UCSRA
Read/Write	R	R/W	R	R	R	R	R/W	R/W	
Initial Value	0	0	1	0	0	0	0	0	

- **Bit 7 – RXC: USART Receive Complete**

This flag bit is set when there are unread data in the receive buffer and cleared when the receive buffer is empty (that is, does not contain any unread data). If the receiver is disabled, the receive buffer will be flushed and consequently the RXC bit will become zero. The RXC Flag can be used to generate a Receive Complete interrupt (see description of the RXCIE bit).

- **Bit 6 – TXC: USART Transmit Complete**

This flag bit is set when the entire frame in the transmit Shift Register has been shifted out and there are no new data currently present in the transmit buffer (UDR). The TXC Flag bit is automatically cleared when a transmit complete interrupt is executed, or it can be cleared by writing a one to its bit location. The TXC Flag can generate a Transmit Complete interrupt (see description of the TXCIE bit).

- **Bit 5 – UDRE: USART Data Register Empty**

The UDRE Flag indicates if the transmit buffer (UDR) is ready to receive new data. If UDRE is one, the buffer is empty, and therefore ready to be written. The UDRE Flag can generate a Data Register empty Interrupt (see description of the UDRIE bit).

UDRE is set after a reset to indicate that the transmitter is ready.

```
;   1ED022, Computer Tehnology
;   Lab 4, task 3 and 4
;   Date: November 27, 2011
;   Anders Haggren
;   Function: Receive data on RS232 Spare and echo
;   Polled receive on USART.
;   Target systen: STK600, ATMega2560

.include "m2560def.inc"

.def Temp = r17
.def char = r16
.equ UBRR_val = 12      ; osc.=1MHz, 4800 bps => UBBRR = 12

.org 0x00
    rjmp start

.org 0x30
start:
    ldi Temp, 0xFF      ; PORTB outputs
    out DDRB, Temp
    ldi Temp, 0X55      ; Initial value to outputs
    out PORTB, Temp

    ldi Temp, UBRR val  ; store Prescaler value in UBRR1L
    sts UBRR1L, Temp

    ldi Temp, (1<<TXEN1) | (1<<RXEN1)
    sts UCSR1B, Temp    ; set TX and RX enable flags

GetChar:                ; receive data
```

```
.org 0x30
start:
    ldi Temp, 0xFF       ; PORTB outputs
    out DDRB, Temp
    ldi Temp, 0X55       ; Initial value to outputs
    out PORTB, Temp

    ldi Temp, UBRR val   ; store Prescaler value in UBRR1L
    sts UBRR1L, Temp

    ldi Temp, (1<<TXEN1) | (1<<RXEN1)
    sts UCSR1B, Temp     ; set TX and RX enable flags

GetChar:                 ; receive data

    lds Temp, UCSR1A     ; read UCSR1A I/O register to r20
    sbrs Temp, RXC1      ; RXC1=1 => new character
    rjmp GetChar         ; RXC1=0 => no character received
    lds Char, UDR1       ; read character in UDR

Port_output:

    com Char
    out PORTB, Char      ; write character to PORTB
    com Char

PutChar:                 ; send data

    lds Temp, UCSR1A     ; read UCSR1A I/O register to r20
    sbrs Temp, UDRE1     ; UDRE1=1 => buffer is empty
    rjmp PutChar         ; UDRE1=0 => buffer is not empty
    sts UDR1, Char       ; write character to UDR1
    rjmp GetChar         ; return to loop
```

Initialize USART

Select BAUD rate, register UBBRL

STK500:

```
.equ UBRR_val = 12              ; osc. = 1MHz, BAUD rate 4800 bps

        ldi Temp, UBRR_val      ; store Prescaler value in UBBRRL
        out UBRRL, Temp
```

STK600:

```
.equ UBRR_val = 12              ; osc. = 1MHz, BAUD rate 4800 bps

        ldi Temp, UBRR_val      ; store Prescaler value in UBBRRL
        sts UBRR1L, Temp
```

Initialize USART
Set enable flags in UCR

STK500:

```
.def Temp = r17

        ldi Temp, (1<<TXEN) | (1<<RXEN)
        out UCR, Temp           ; set TX and RX Enable flags
```

STK600:

```
.def Temp = r17

        ldi Temp, (1<<TXEN1) | (1<<RXEN1)
        sts UCSR1B, Temp        ; set TX and RX Enable flags
```

Polled input,
check data received flag

STK500:

```
GetChar:                    ; receive data

        sbis USR, RXC       ; wait for character input
        rjmp GetChar        ; loop if no character input
        in r16, UDR         ; read character in UDR to r16
```

STK600:

```
GetChar:                    ; receive data
        lds r20, UCSR1A
        andi r20, 0x80      ; mask out bit 7 = RXC1
        cpi r20, 0x80       ; RXC1 = bit 7 in UCSR1A
        brne GetChar        ; RXC1 = 0, no character received
        lds r16, UDR1       ; read character in UDR to r16
```

Polled input / output,
send data

STK500:

```
PutChar:                    ; send data
        sbis USR, UDRE      ; wait if last character not yet sent
        rjmp PutChar        ; loop until UDRE flag  = 1, in USR
        out UDR, r16        ; write character in UDR
```

STK600:

```
PutChar:                    ; send data

        lds r20, UCSR1A
        andi r20, 0x20      ; mask out bit 5 = UDRE1
        cpi r20, 0x20       ; mask out bit 7 = RXC1
        brne PutChar        ; UDRE1 = 1, buffer is empty
        sts UDR1, r16       ; write character to UDR1
```

Alternate Functions of Port D

The Port D pins with alternate functions are shown in Table 13-12.

Table 13-12. Port D Pins Alternate Functions

Port Pin	Alternate Function
PD7	T0 (Timer/Counter0 Clock Input)
PD6	T1 (Timer/Counter1 Clock Input)
PD5	XCK1 (USART1 External Clock Input/Output)
PD4	ICP1 (Timer/Counter1 Input Capture Trigger)
PD3	INT3/TXD1 (External Interrupt3 Input or USART1 Transmit Pin)
PD2	INT2/RXD1 (External Interrupt2 Input or USART1 Receive Pin)
PD1	INT1/SDA (External Interrupt1 Input or TWI Serial DAta)
PD0	INT0/SCL (External Interrupt0 Input or TWI Serial CLock)

The alternate pin configuration is as follows:

- **INT3/TXD1 – Port D, Bit 3**

INT3, External Interrupt source 3: The PD3 pin can serve as an external interrupt source to the MCU.

TXD1, Transmit Data (Data output pin for the USART1). When the USART1 Transmitter is enabled, this pin is configured as an output regardless of the value of DDD3.

- **INT2/RXD1 – Port D, Bit 2**

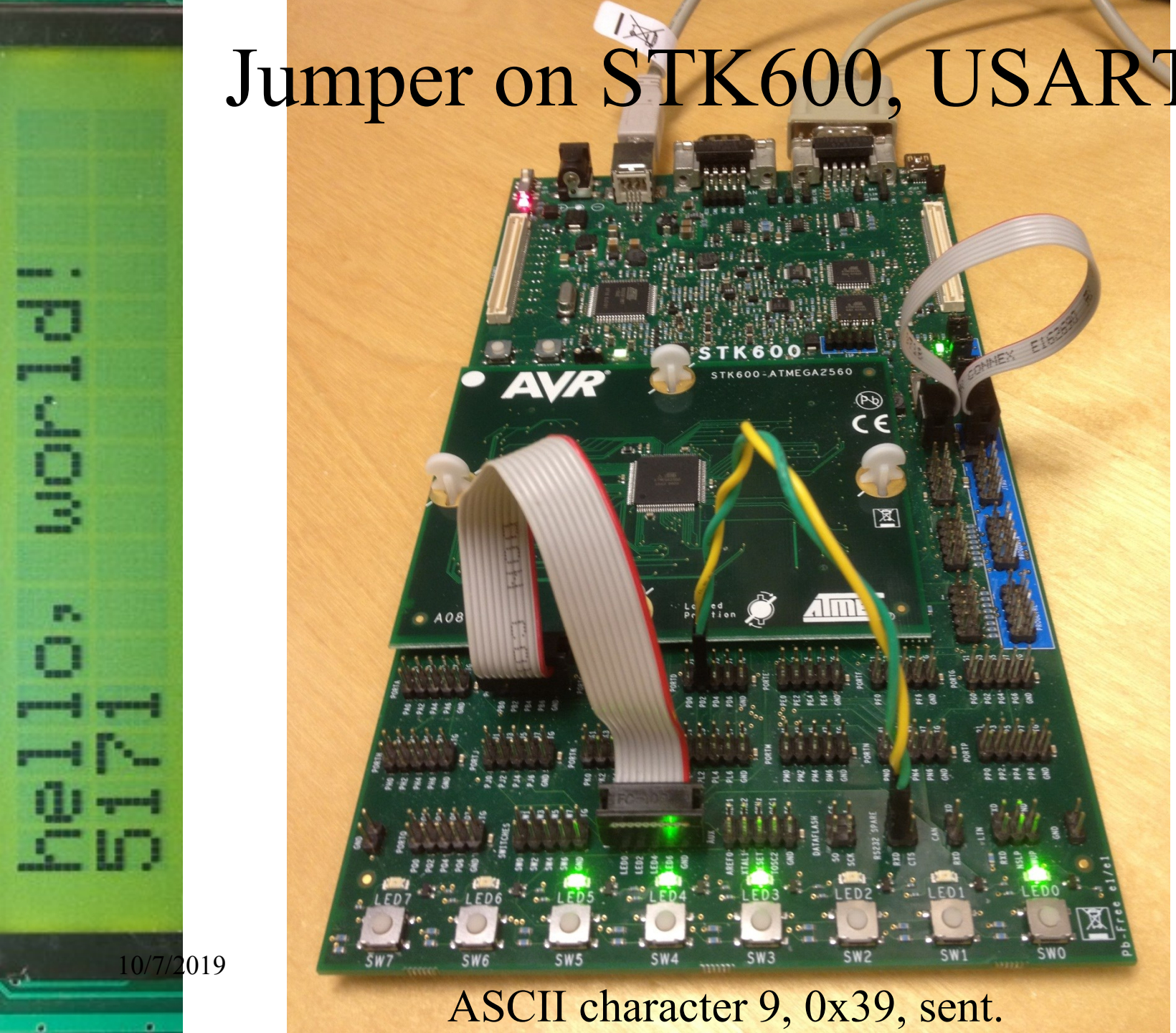
INT2, External Interrupt source 2. The PD2 pin can serve as an External Interrupt source to the MCU.

RXD1, Receive Data (Data input pin for the USART1). When the USART1 receiver is enabled this pin is configured as an input regardless of the value of DDD2. When the USART forces this pin to be an input, the pull-up can still be controlled by the PORTD2 bit.

TX Complete Interrupt Enable
USART Data register Empty Interrupt En...
Receiver Enable
Transmitter Enable
Character Size
Receive Data Bit 8
Transmit Data Bit 8
USART Mode Select: Asynchronous USART
Parity Mode Bits: Disabled

Name	Address	Value	Bits
UBRR1	na (0xCC)	0x000C	
UBRR1H	na (0xCD)	0x00	
UBRR1L	na (0xCC)	0x0C	
UCSR1A	na (0xC8)	0x20	
RXC1		0x00	
TXC1		0x00	
UDRE1		0x01	
FE1		0x00	
DOR1		0x00	
UPE1		0x00	
U2X1		0x00	
MPCM1		0x00	
UCSR1B	na (0xC9)	0x98	
RXCIE1		0x01	
TXCIE1		0x00	
UDRIE1		0x00	
RXEN1		0x01	
TXEN1		0x01	
UCSZ12		0x00	
RXB81		0x00	
TXB81		0x00	
UCSR1C	na (0xCA)	0x06	
UMSEL1		0x00	
UPM1		0x00	
USBS1		0x00	
UCSZ1		0x03	
UCPOL1		0x00	
UDR1	na (0xCE)	0x00	

Jumper on STK600, USART1



ASCII character 9, 0x39, sent.

1ED022, Computer Technology

- Disassembler
- Two’s complement for
  (for negative values stored binary)

C:\Document__Kurser\D2_E2\Datorteknik\HT_2014\Test_Example_Lectures...

```
.INCLUDE "m2560def.inc"
.DEF temp = r16
.DEF Output = r17
.DEF count = r18

.EQU timer_init = 100
.EQU timer_divide = 100

.ORG 0x00
    rjmp restart                ; reset interrupt
.ORG OVF0addr                   ; timer 0 interrupt
    rjmp timer0_int

    .org 0x72
restart:
    ldi temp, LOW(RAMEND)       ; initialize SP, Stackpointer
    out SPL, temp
    ldi temp, HIGH(RAMEND)
    out SPH, temp

    ldi r24, LOW(timer_divide)
    ldi r25, HIGH(timer_divide)

    ldi temp, 0xFF              ; initialize DDRB
    out DDRB, temp
    ldi temp, 0xFF              ; initialize DDRB
    out DDRE, temp
    ldi output, 0x0F
    ldi count, 0
    ldi temp, 0b000000101       ; prescaler value to TCCR0
    out TCCR0B, temp            ; CS2 = 101, osc.clock / 1024
    ldi temp, (1<<TOIE0)        ; Timer 0 enable flag, TOIE0
    sts TIMSK0, temp            ; to register TIMSK
    ldi temp, timer_init        ; starting value for counter
    out TCNT0, temp             ; counter register
    sei

start: ; main loop
    out PORTB, Output
    out PORTE, Output
    rjmp start

timer0_int:
    push temp                   ; timer interrupt routine
    in temp, SREG               ; save SREG on stack
    push temp
; additional code to create the square output

    sbiw r24, 1
    brne cont
```

C:\Document__Kurser\D2_E2\Datorteknik\HT_2014\Test_Example_Lectures...

```
restart:
    ldi temp, LOW(RAMEND)        ; initialize SP, Stackpointer
    out SPL, temp
    ldi temp, HIGH(RAMEND)
    out SPH, temp

    ldi r24, LOW(timer_divide)
    ldi r25, HIGH(timer_divide)

    ldi temp, 0xFF               ; initialize DDRB
    out DDRB, temp
    ldi temp, 0xFF               ; initialize DDRB
    out DDRE, temp
    ldi output, 0x0F
    ldi count, 0
    ldi temp, 0b0000000101       ; prescaler value to TCCR0
    out TCCR0B, temp             ; CS2 = 101, osc.clock / 1024
    ldi temp, (1<<TOIE0)         ; Timer 0 enable flag, TOIE0
    sts TIMSK0, temp             ; to register TIMSK
    ldi temp, timer_init         ; starting value for counter
    out TCNT0, temp              ; counter register
    sei

start: ; main loop
    out PORTB, Output
    out PORTE, Output
    rjmp start

timer0_int:
    push temp                    ; timer interrupt routine
    in temp, SREG                ; save SREG on stack
    push temp
; additional code to create the square output

    sbiw r24, 1
    brne cont

    com Output
    ldi r24, LOW(timer_divide)
    ldi r25, HIGH(timer_divide)

cont:
    ldi temp, timer_init         ; starting value for counter
    out TCNT0, temp
    pop temp                     ; restore SREG
    out SREG, temp
    pop temp                     ; restore register
;   com Output                   ; 1-complements (inverts) "Output"
    reti                         ; return from interrupt
```

C:\Document__Kurser\D2_E2\Datorteknik\HT_2014\Test_Example_Lectures...

```
restart:
    ldi temp, LOW(RAMEND)        ; initialize SP, Stackpointer
    out SPL, temp
    ldi temp, HIGH(RAMEND)
    out SPH, temp

    ldi r24, LOW(timer_divide)
    ldi r25, HIGH(timer_divide)

    ldi temp, 0xFF               ; initialize DDRB
    out DDRB, temp
    ldi temp, 0xFF               ; initialize DDRB
    out DDRE, temp
    ldi output, 0x0F
    ldi count, 0
    ldi temp, 0b0000000101       ; prescaler value to TCCR0
    out TCCR0B, temp             ; CS2 = 101, osc.clock / 1024
    ldi temp, (1<<TOIE0)         ; Timer 0 enable flag, TOIE0
    sts TIMSK0, temp             ; to register TIMSK
    ldi temp, timer_init         ; starting value for counter
    out TCNT0, temp              ; counter register
    sei

start: ; main loop
    out PORTB, Output
    out PORTE, Output
    rjmp start

timer0_int:
    push temp                    ; timer interrupt routine
    in temp, SREG                ; save SREG on stack
    push temp
; additional code to create the square output

    sbiw r24, 1
    brne cont

    com Output
    ldi r24, LOW(timer_divide)
    ldi r25, HIGH(timer_divide)

cont:
    ldi temp, timer_init         ; starting value for counter
    out TCNT0, temp
    pop temp                     ; restore SREG
    out SREG, temp
    pop temp                     ; restore register
;   com Output                   ; 1-complements (inverts) "Output"
    reti                         ; return from interrupt
```

C:\Document__Kurser\D2_E2\Datorteknik\HT_2014\Test_Example_Lectures...

```
.INCLUDE "m2560def.inc"
.DEF temp = r16
.DEF Output = r17
.DEF count = r18

.EQU timer_init = 100
.EQU timer_divide = 100

.ORG 0x00
    rjmp restart                ; reset interrupt
.ORG OVF0addr                   ; timer 0 interrupt
    rjmp timer0_int

    .org 0x72
restart:
    ldi temp, LOW(RAMEND)       ; initialize SP, Stackpointer
    out SPL, temp
    ldi temp, HIGH(RAMEND)
    out SPH, temp

    ldi r24, LOW(timer_divide)
    ldi r25, HIGH(timer_divide)

    ldi temp, 0xFF              ; initialize DDRB
    out DDRB, temp
    ldi temp, 0xFF              ; initialize DDRB
    out DDRE, temp
    ldi output, 0x0F
    ldi count, 0
    ldi temp, 0b0000000101      ; prescaler value to TCCR0
    out TCCR0B, temp            ; CS2 = 101, osc.clock / 1024
    ldi temp, (1<<TOIE0)        ; Timer 0 enable flag, TOIE0
    sts TIMSK0, temp            ; to register TIMSK
    ldi temp, timer_init        ; starting value for counter
    out TCNT0, temp             ; counter register
    sei

start: ; main loop
    out PORTB, Output
    out PORTE, Output
    rjmp start

timer0_int:
    push temp                   ; timer interrupt routine
    in temp, SREG               ; save SREG on stack
    push temp
; additional code to create the square output

    sbiw r24, 1
    brne cont

    com Output
    ldi r24, LOW(timer_divide)
```

I/O View

ANALOG_COMPAR

Name	Value
EXTERNAL_INTERRUPT	
JTAG	
PORTA	
PORTB	
Port B Data Register	0x0F
Port B Data Direction Register	0xFF
Port B Input Pins	0x00
PORTC	
PORTD	
PORTE	
PORTF	
PORTG	
PORTH	
PORTJ	
PORTK	
PORTL	
SPI	
TIMER_COUNTER_0	
Timer/Counter0 Output Compare Regis...	0x00
Timer/Counter0 Output Compare Regis...	0x00
Timer/Counter0	0x64

Name	Address	Value	Bits
GTCCR	0x23 (0x43)	0x00	□■■■■■■□
OCR0A	0x27 (0x47)	0x00	□□□□□□□□
OCR0B	0x28 (0x48)	0x00	□□□□□□□□
TCCR0A	0x24 (0x44)	0x00	□□□□■■□□
TCCR0B	0x25 (0x45)	0x05	□□■■□■□■
TCNT0	0x26 (0x46)	0x64	□■■□□■□□
TIFR0	0x15 (0x35)	0x00	■■■■■□□□
TIMSK0	na (0x6E)	0x01	■■■■■□□■