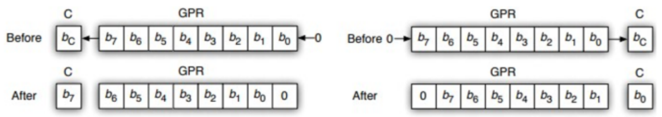
```
ADD Rd, Rr Add two Registers Rd←Rd+Rr 0000, 11rd, dddd, rrrr
ADIW Rdl,K Add Immediate to Word Rdh:Rdl←Rdh:Rdl+K Z,C,N,V,S 2: 1001, 0110, KKdd, KKKK
AND Rd, Rr Logical AND Registers Rd← Rd∧Rr Z,N,V 1: 0010, 00rd, dddd, rrrr
ANDI Rd, K Logical AND Register and Constant Rd←Rd∧K Z,N,V 1: 0111, KKKK, dddd, KKKK
ASR Rd Arithmetic Shift Right Rd(n) ← Rd(n+1), n=0..6 Z,C,N,V: 11001, 010d, dddd, 0101
BCLR s Flag Clear SREG(s) ← 0: 1001, 0100, 1sss, 1000
BLD Rd, b Bit load from T to Register Rd(b) ← T None: 11111, 100d, dddd, 0bbb
BRBC s, k Branch if Status Flag Cleared if (SREG(s)=0) then PC←PC+k+1 None 1/2: 1111, 01kk, kkkk, ksss
BRBS s, k Branch if Status Flag Set if (SREG(s)=1) then PC←PC+k+1 None 1/2 :1111, 00kk, kkkk, ksss
BRCC k Branch if Carry Cleared if (C=0) then PC←PC+k+1 None 1/2 : 1111, 01kk, kkkk, k000
BRCS k Branch if Carry Set if (C=1) then PC←PC+k+1 None 1/2  : 1111, 00kk, kkkk, k000
BREAK Break For On-chip Debug Only None N/A : 1001, 0101, 1001, 1000
BREQ k Branch if Equal if (Z=1) then PC←PC+k+1 None 1/2 : 1111, 00kk, kkkk, k001
BRGE k Branch if Greater or Equal, Signed if (N⊕V= 0) then PC←PC+k+1 None 1/2 : 1111, 01kk, kkkk, k100
BRHC k Branch if Half Carry Flag Cleared if (H=0) then PC←PC+k+1 None 1/2 : 1111, 01kk, kkkk, k101
BRHS k Branch if Half Carry Flag Set if (H=1) then PC←PC+k+1 None 1/2 : 1111, 00kk, kkkk, k101
BRID k Branch if Interrupt Disabled if (I = 0) then PC←PC+k+1 None 1/2 : 1111, 01kk, kkkk, k111
BRIE k Branch if Interrupt Enabled if (I=1) then PC←PC+k+1 None ½ : 1111, 00kk, kkkk, k111
BRLO k Branch if Lower if (C=1) then PC←PC+k+1 None 1/2 : 1111, 00kk, kkkk, k000
BRLT k Branch if Less Than Zero, Signed if (N⊕V= 1) then PC←PC+k+1 None 1/2 : 1111, 00kk, kkkk, k100
BRMI k Branch if Minus if (N=1) then PC←PC+k+1 None 1/2 : 1111, 00kk, kkkk, k010
BRNE k Branch if Not Equal if (Z=0) then PC←PC+k+1 None 1/2 : 1111, 01kk, kkkk, k001
BRPL k Branch if Plus if (N=0) then PC←PC+k+1 None 1/2 : 1111, 01kk, kkkk, k010
BRSH k Branch if Same or Higher if (C=0) then PC←PC+k+1 None 1/2 : 1111, 01kk, kkkk, k000
BRTC k Branch if T Flag Cleared if (T=0) then PC←PC+k+1 None 1/2 : 1111, 01kk, kkkk, k110
BRTS k Branch if T Flag Set if (T=1) then PC←PC+k+1 None 1/2 : 1111, 00kk, kkkk, k110
BRVC k Branch if Overflow Flag is Cleared if (V=0) then PC←PC+k+1 None 1/2 : 1111, 01kk, kkkk, k011
BRVS k Branch if Overflow Flag is Set if (V=1) then PC←PC+k+1 None 1/2 : 1111, 00kk, kkkk, k011
BSET s Flag Set SREG(s) ← 1 SREG(s) 1 : 1001, 0100, 0sss, 1000
BST Rr, b Bit Store from Register to T T ← Rr(b) T 1 : 1111, 101d, dddd, 0bbb
CALL k Direct Subroutine Call PC←k None 4 : 1001, 010k, kkkk, 111k, kkkk, kkkk, kkkk, kkkk
CBI P,b Clear Bit in I/O Register (P,b) ← 0 None 2: 1001, 1000, AAAA, Abbb
CBR Rd, K Clear Bit(s) in Register Rd←Rd∧($FF-K) Z,N,V 1 One Register : 0111, KKKK, dddd, KKKK (K, C)
CLC Clear Carry C ← 0 C 1 : 1001, 0100, 1000, 1000
CLH Clear Half Carry Flag in SREG H ← 0 H 1 : 1001, 0100, 1101, 1000
 CLI Global Interrupt Disable I ← 0 I 1 : 1001, 0100, 1111, 1000
CLN Clear Negative Flag N ← 0 N 1 : 1001, 0100, 1010, 1000
CLR Rd Clear Register Rd←Rd⊕Rd Z,N,V : 0010, 01d, dddd, dddd
CLS Clear Signed Test Flag S ← 0 S 1 : 1001, 0100, 1100, 1000
CLT Clear T in SREG T ← 0 T 1 : 1001, 0100, 1110, 1000
CLV Clear Twos Complement Overflow V ← 0 V 1 : 1001, 0100, 1011, 1000
CLZ Clear Zero Flag Z ← 0 Z 1 : 1001, 0100, 1001, 1000
COM Rd One's Complement Rd←$FF-Rd Z,C,N,V 1 : 1001, 010d, dddd, 0000
CP Rd,Rr Compare Rd - Rr Z, N,V,C,H 1 : 0001, 01rd, dddd, rrrr
CPC Rd,Rr Compare with Carry Rd - Rr - C Z, N,V,C,H 1 : 0000, 01rd, dddd, rrrr
CPI Rd,K Compare Register with Immediate Rd - K Z, N,V,C,H 1 : 0011, KKKK, dddd, KKKK
CPSE Rd,Rr Compare, Skip if Equal if (Rd = Rr) PC←PC+2 or 3 None 1/2/ 3  : 0001, 00rd, dddd, rrrr
DEC Rd Decrement Rd←Rd-1 Z,N,V 1 : 1001, 010d, dddd, 1010
ELPM Extended Load Program Memory R0 ← (RAMPZ:Z) None 3  : 1001, 0101, 1101, 1000
ELPM Rd, Z Extended Load Program Memory Rd ← (RAMPZ:Z) None 3 : 1001, 000d, dddd, 0110
ELPM Rd, Z+ Extended Load Program Memory and Post-Inc Rd ← (RAMPZ:Z), RAMPZ:Z ← RAMPZ:Z+1 None 3 Store to
Program Memory : 1001, 000d, dddd, 0111
EOR Rd, Rr Exclusive OR Registers Rd←Rd⊕Rr Z,N,V 1 : 0010, 01rd, dddd, rrrr
FMUL Rd, Rr Fractional Multiply Unsigned R1:R0←(Rd×Rr)<<1 Z,C 2 : 0000, 0011, 0ddd, 1rrr
FMULS Rd, Rr Fractional Multiply Signed R1:R0←(Rd× Rr)<<1 Z,C 2 : 0000, 0011, 1ddd, 0rrr
FMULSU Rd, Rr Fractional Multiply Signed with Unsigned R1:R0←(Rd× Rr)<<1 Z,C 2 : 0000, 0011, 1ddd, 1rrr
ICALL Indirect Call to (Z) PC←Z None 3 : 1001, 0101, 0000, 1001
IJMP Indirect Jump to (Z) PC←Z None 2 : 1001, 0100, 0000, 1001
IN Rd, P In Port Rd ← P None 1 : 1011, 0AAd, dddd, AAAA
INC Rd Increment Rd←Rd+1 Z,N,V 1 : 1001, 010d, dddd, 0011
JMP k Direct Jump PC←k None 3 Subroutine Calls and Return : 1001, 010k, kkkk, 110k, kkkk, kkkk, kkkk, kkkk
LD Rd, X Load Indirect Rd ← (X) None 2 : 1001, 000d, dddd, 1100
LD Rd, X+ Load Indirect and Post-Inc. Rd ← (X), X ← X + 1 None 2 : 1001, 000d, dddd, 1101
LD Rd, -X Load Indirect and Pre-Dec. X ← X - 1, Rd ← (X) None 2 : 1001, 000d, dddd, 1110
LD Rd, Y Load Indirect Rd ← (Y) None 2 : 1000, 000d, dddd, 1000
LD Rd, Y+ Load Indirect and Post-Inc. Rd ← (Y), Y ← Y + 1 None 2 : 1001, 000d, dddd, 1001
LD Rd, -Y Load Indirect and Pre-Dec. Y ← Y - 1, Rd ← (Y) None 2 : 1001, 000d, dddd, 1010
LD Rd, Z Load Indirect Rd ← (Z) None 2 : 1000, 000d, dddd, 0000
LD Rd, Z+ Load Indirect and Post-Inc. Rd ← (Z), Z ← Z+1 None 2 : 1001, 000d, dddd, 0001
LD Rd, -Z Load Indirect and Pre-Dec. Z ← Z - 1, Rd ← (Z) None 2 : 1001, 000d, dddd, 0010
LDD Rd,Y+q Load Indirect with Displacement Rd ← (Y + q) None 2 : 10q0, qq0d, dddd, 1qqq
LDD Rd, Z+q Load Indirect with Displacement Rd ← (Z + q) None 2 : 10q0, qq0d, dddd, 0qqq
LDI Rd, K Load Immediate Rd ← K None 1 Load from Memory : 1110, KKKK, dddd, KKKK
LDS  Rd, k Load Direct from data Rd ← (k) None 2 : 1001, 000d, , dddd, 0000, kkkk, kkkk, kkkk, kkkk
LDS  Rd, k Load Direct from SRAM Rd ← (k) None 2 : 1010, 0kkk, dddd, kkkk
LPM Load Program Memory R0 ← (Z) None 3 : 1001, 0101, 1100, 1000
LPM Rd, Z Load Program Memory Rd ← (Z) None 3 ;  1001, 000d, dddd, 0100
LPM Rd, Z+ Load Program Memory and Post-Inc Rd ← (Z), Z ← Z+1 None 3 : 1001, 000d, dddd, 0101
LSL Rd Logical Shift Left Rd(n+1)←Rd(n), Rd(0)←0 Z,C,N,V 1 : 0000, 11dd, dddd, dddd
LSR Rd Logical Shift Right Rd(n)←Rd(n+1), Rd(7)←0 Z,C,N,V 1 : 1001, 010d, dddd, 0110
MOV Rd, Rr Move Between Registers Rd ← Rr None 1 : 0010, 11rd, dddd, rrrr
MOVW Rd, Rr Copy Register Word Rd+1:Rd ← Rr+1:Rr None 1 : 0000, 0001, dddd, rrrr
MUL Rd, Rr Multiply Unsigned R1:R0←Rd×Rr Z,C 2 : 1001, 11rd, dddd, rrrr
MULS Rd, Rr Multiply Signed R1:R0←Rd×Rr Z,C 2 : 0000, 0010, dddd, rrrr
MULSU Rd, Rr Multiply Signed with Unsigned R1:R0←Rd×Rr Z,C 2 : 0000, 0011, 0ddd, 0rrr
NEG Rd Two's Complement Rd←$00-Rd Z,C,N,V,H 1 : 1001, 010d, dddd, 0001
NOP No Operation None 1 : 0000, 0000, 0000, 0000
OR Rd, Rr Logical OR Registers Rd← RdVRr Z,N,V 1 : 0010, 10rd, dddd, rrrr
ORI Rd, K Logical OR Register and Constant Rd← RdVK Z,N,V 1 : 0110, KKKK, dddd, KKKK
OUT P, Rr Out Port P ← Rr None 1 Stack Manipulation : 1011, 1AAr, rrrr, AAAA
POP Rd Pop Register from Stack Rd ← STACK None 2 : 1001, 000d, dddd, 1111
PUSH Rr Push Register on Stack STACK ← Rr None 2 : 1001, 001d, dddd, 1111
RCALL k Relative Subroutine Call PC←PC+k+1 None 3 : 1101, kkkk, kkkk, kkkk
RET Subroutine Return PC←STACK None 4 : 1001, 0101, 0000, 1000
RETI Return from Interrupt PC←STACK I 4 Compare : 1001, 0101, 0001, 1000
RJMP k Relative Jump PC←PC+k+1 None 2 : 1100, kkkk, kkkk, kkkk
ROL Rd Rotate Left Through Carry Rd(0)←C, Rd(n+1)←Rd(n), C←Rd(7) Z,C,N,V 1 :  0001, 11dd, dddd, dddd
ROR Rd Rotate Right Through Carry Rd(7)←C, Rd(n)←Rd(n+1), C←Rd(0) Z,C,N,V 1 : 1001, 010d, dddd, 0111
SBC Rd, Rr Subtract with Carry two Registers Rd←Rd-Rr-C Z,C,N,V,H 1 0000, 10rd, dddd, rrrr
SBCI Rd, K Subtract with Carry Constant from Reg. Rd←Rd-K-C Z,C,N,V,H 1 : 0100, KKKK, dddd, KKKK
SBI P,b Set Bit in I/O Register (P,b) ← 1 None 2 : 1001, 1010, AAAA, Abbb
SBIC P, b Skip if Bit in I/O Register Cleared if (P(b)=0) PC←PC+2 or 3 None 1/2/3 : 1001, 1001, AAAA, Abbb
SBIS P, b Skip if Bit in I/O Register is Set if (P(b)=1) PC←PC+2 or 3 None 1/2/3 : 1001, 1011, AAAA, Abbb
SBIW Rdl,K Subtract Immediate from Word Rdh:Rdl←Rdh:Rdl-K Z,C,N,V,S 2 : 1001, 0111, KKdd, KKKK
SBR Rd, K Set Bit(s) in Register Rd←RdVK Z,N,V 1 : 0110, KKKK, dddd, KKKK
SBRC Rr, b Skip if Bit in Register Cleared if (Rr(b)=0) PC←PC+2 or 3 None 1/2/3 : 1111, 110r, rrrr, 0bbb
SBRS Rr, b Skip if Bit in Register is Set if (Rr(b)=1) PC←PC+2 or 3 None 1/2/3 : 1111, 111r, rrrr, 0bbb
SEC Set Carry C ← 1 C 1 : 1001, 0100, 0000, 1000
SEH Set Half Carry Flag in SREG H ← 1 H 1 : 1001, 0100, 0101, 1000
SEI Global Interrupt Enable I ← 1 I 1 : 1001, 0100, 0111, 1000
SEN Set Negative Flag N ← 1 N 1 : 1001, 0100, 0010, 1000
SER Rd Set Register Rd←$FF None 1 : 1110, 1111, dddd, 1111
SES Set Signed Test Flag S ← 1 S 1 ; 1001, 0100, 0100, 1000
SET Set T in SREG T ← 1 T 1 : 1001, 0100, 0110, 1000
SEV Set Twos Complement Overflow. V ← 1 V 1 : 1001, 0100, 0011, 1000
SEZ Set Zero Flag Z ← 1 Z 1 : 1001, 0100, 0001, 1000
SLEEP Sleep (see specific descr. for Sleep function) None 1 ; 1001, 0101, 1000, 1000
 SPM Store Program Memory (Z) ← R1:R0 None - Load/Store from/to I/O Register : 1001, 0101, 1110, 1000
ST X, Rr Store Indirect (X) ← Rr None 2 : 1001, 001r, rrrr, 1100
ST X+, Rr Store Indirect and Post-Inc. (X) ← Rr, X ← X + 1 None 2 : 1001, 001r, rrrr, 1101
ST -X, Rr Store Indirect and Pre-Dec. X ← X - 1, (X) ← Rr None 2 : 1001, 001r, rrrr, 1110
ST Y, Rr Store Indirect (Y) ← Rr None 2 : 1000, 001r, rrrr, 1000
ST Y+, Rr Store Indirect and Post-Inc. (Y) ← Rr, Y ← Y + 1 None 2 : 1001, 001r, rrrr, 1001
ST -Y, Rr Store Indirect and Pre-Dec. Y ← Y - 1, (Y) ← Rr None 2 : 1001, 001r, rrrr, 1010
ST Z, Rr Store Indirect (Z) ← Rr None 2 :1000, 001r, rrrr, 0000
ST Z+, Rr Store Indirect and Post-Inc. (Z) ← Rr, Z ← Z + 1 None 2 : 1001, 001r, rrrr, 0001
ST -Z, Rr Store Indirect and Pre-Dec. Z ← Z - 1, (Z) ← Rr None 2 : 1001, 001r, rrrr, 0010
STD Y+q,Rr Store Indirect with Displacement (Y + q) ← Rr None 2 : 10q0, qq1r, rrrr, 1qqq
STD Z+q,Rr Store Indirect with Displacement (Z + q) ← Rr None 2 10q0, qq1r, rrrr, 0qqq
STS k, Rr Store Direct to data space (k) ← Rr None 2 : 1001, 001d, dddd, 0000, kkkk, kkkk, kkkk, kkkk
STS k, Rr Store Direct to SRAM (k) ← Rr None 2 : 1010, 1kkk, rrrr, kkkk
SUB Rd, Rr Subtract two Registers Rd←Rd-Rr Z,C,N,V,H 1 : 0001, 10rd, dddd, rrrr
SUBI Rd, K Subtract Constant from Register Rd←Rd-K Z,C,N,V,H 1 : 0101, KKKK, dddd, KKKK
SWAP Rd Swap Nibbles Rd(3..0)←Rd(7..4), Rd(7..4)←Rd(3..0) None 1 : 1001, 010d, dddd, 0010
SRAM (k) ← Rr None 2 Load from Program Memory
TST Rd Test for Zero or Minus Rd←Rd∧Rd Z,N,V 1 : 0010, 00dd, dddd, dddd
WDR Watchdog Reset (see specific descr. for WDR/timer) None 1 : 1001, 0101, 1010, 1000
```

C – bit 0 – carry flag, Z – bit 1 – Zero flag, N – bit 2 – Negative flag, V – bit 3 – two's compliment flag, S – bit 4 – Sign bit, H – bit 5, half carry flag, T – bit 6 – bit copy storage, I – bit 7 – global interrupt enable.
Immediate – operand is in instruction, Direct – operand is in memory locaton or register Is in an instruction, Memory indirect – operand is in a effective addres is in an address, Regiister indirect – operand is in a register is in an instruction.
Rrrrr – source register, rrrr – source register 16-31, rrr source register 16-23, RRRR – source register pair, ddddd – destination register, ddd – destination register 16-31, DDDD – destination register pair, pp – register pair XYZ, y – Y/Z register pair (0=Z Y = 1), u – FMUL signed (0=s 1=us), s – store/load (0=load 1=store), c – call/jump (0=jump 1=call), cy with carry (0=no 1=Y), aaaaaa – io space address, aaaaa – first 32 io space address's, bbb – bit number, B – bit value, kkkkkk – 6-bit unsigned constant, KKKKKKKK – 8 bit constant. All branches k's are in two's compliment form, CBR is the same as ANDI but the k's are complemented. Branches and calls are address - Pc + 1

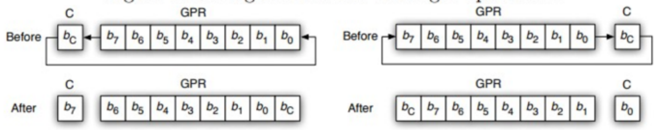
```
STA     -(x)      ; M(x) ← M(x) - 1, M(M(x)) ← AC
Fetch Cycle
Step 1:  MAR ← PC;
Step 2:  MDR ← M(MAR), PC ← PC + 1            ; Read inst. & increment PC
Step 3:  IR ← MDR_opcode, MAR ← MDR_address   ;

Execute Cycle
Step 1: MDR ← M(MAR), TEMP ← AC               ; Read M(x) (i.e., EA+1) and store AC
Step 2: AC ← MDR                               ;
Step 3: AC ← AC – 1                            ; Decrement EA+1
Step 4: MDR ← AC                               ;
Step 5: M(MAR) ← MDR, MAR ← AC (or MDR)        ; Store EA back in M(x) & MAR
Step 6: AC ← TEMP, MDR ← TEMP                  ; Restore AC and store it in MDR
Step 7: M(MAR) ← MDR                           ; Store operand in AC to memory location pointed to by EA
```



(a) Shift left. (b) Shift right.

Figure 4.16: Logical shift left and right operations.



(a) Rotate left. (b) Rotate right.

Figure 4.17: Rotate left and right operations.

ASR is the same as LSR except it maintains bit 7 by copying bit 7 and shifting it right once.

Table 3.2: Instructions in the pseudo-ISA.

Category	Instruction	Description
Data transfer	`LDA x`	Load accumulator
	`STA x`	Store accumulator
Arithmetic & logic	`ADD x`	Add to accumulator
	`SUB x`	Subtract from accumulator
	`NAND x`	Logical NAND to accumulator
	`SHFT`	Shift accumulator
Control transfer	`J x`	Jump
	`BNZ x`	Branch conditionally

The following shows the sequence of micro-operations required for each instruction in Table 3.2.

Execute Cycle:

Cycle 1: $MDR \leftarrow M[MAR]$
Cycle 2: $AC \leftarrow \overline{AC \wedge MDR}$

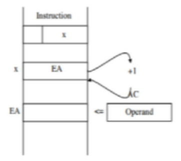
Execute Cycle:

Cycle 1: $AC \leftarrow sl\ AC$

Note that SHFT can also be described as

Cycle 1: $AC(n...1) \leftarrow AC(n\text{-}1...0),\ AC(0) \leftarrow 0$

`.BYTE`	Reserve byte to a variable
`.CSEG`	Code Segment
`.DB`	Define constant byte(s)
`.DEF`	Define a symbolic name on a register
`.DEVICE`	Define which device to assemble for
`.DSEG`	Data Segment
`.DW`	Define constant words
`.ENDMACRO`	End macro
`.EQU`	Set a symbol equal to an expression
`.ESEG`	EEPROM segment
`.EXIT`	Exit from a file
`.INCLUDE`	Read source from another file
`.LIST`	Turn listfile generation on
`.LISTMAC`	Turn macro expression on
`.MACRO`	Begin Macro
`.NOLIST`	Turn listfile generation off
`.ORG`	Set program origin
`.SET`	Set a symbol to an expression



These steps can be implemented by the following microoperations:

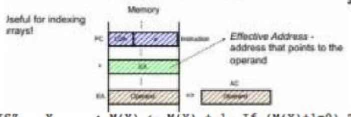
```
Execute Cycle
Step 1:  MDR ← M(MAR), TEMP ← AC   ; Read effective address (EA) and store AC
Step 2:  AC ← MDR                  ; Move EA to AC
Step 3:  AC ← AC +1                ; Increment EA
Step 4:  MDR ← AC                  ; Move EA+1 to MDR
Step 5:  M(MAR) ← MDR, AC ← AC - 1 ; Store EA+1 back and regenerate EA
Step 6:  MAR ← AC                  ; Move EA to MAR
Step 7:  MDR ← TEMP, AC ← TEMP     ; Move the original AC into MDR and restore AC
Step 8:  M(MAR) ← MDR              ; Store AC
```

`LOW(`*expression*`)`	Returns the low-byte of an expression
`HIGH(`*expression*`)`	Returns the high-byte of an expression
`BYTE2(`*expression*`)`	Is the same function as `HIGH`
`BYTE3(`*expression*`)`	Returns the third byte of an expression
`BYTE4(`*expression*`)`	Returns the fourth byte of an expression
`LWRD(`*expression*`)`	Returns bits 0-15 of an expression
`HWRD(`*expression*`)`	Returns bits 16-31 of an expression

Execute Cycle

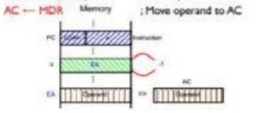
Step 1: MDR ← M(MAR) ; Read effective address (EA)
Step 2: MAR ← MDR ;
Step 3: MDR ← M(MAR) ; Read operand
Step 4: AC ← MDR ; Move operand to AC
Return to the beginning of the instruction cycle

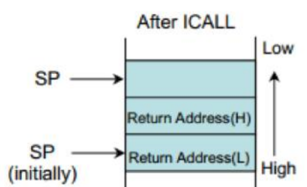
Jseful for indexing arrays!



Execute Cycle

Step 1: MDR ← M(MAR) ; Read effective address (EA)
Step 2: MDR ← MDR - 1 ; Decrement EA
Step 3: M(MAR) ← MDR ; Store it back in x
Step 4: MAR ← MDR ;
Step 5: MDR ← M(MAR) ; Read operand
Step 6: AC ← MDR ; Move operand to AC





```
; Fetch cycle
Step 1:  MAR ← PC;
Step 2:  MDR ← M(MAR), PC ← PC + 1 ; Get the high byte of the instruction and increment PC
Step 3:  IR(15…8) ← MDR
Step 4:  MAR ← PC;
Step 5:  MDR ← M(MAR), PC ← PC + 1 ; Get the low byte of the instruction and increment PC
Step 6:  IR(7…0) ← MDR               ; At this point, CU knows this is an ICALL

; Execute cycle
; The return address is pushed onto the stack
Step 7:  MDR ← PC(7…0)
Step 8:  MAR ← SP
Step 9:  M(MAR) ← MDR, SP ← SP - 1  ; Push the lower byte of return address onto stack
Step 10: MDR ← PC(15…8)
Step 11: MAR ← SP
Step 12: M(MAR) ← MDR, SP ← SP - 1    ; Push the higher byte of return address onto stack

; Put H and L target addresses of the Z register to the PC
Step 13: PC(15…8) ← R31              ; Put the H address of the target into PC
Step 14: PC(7…0) ← R30               ; Put the L address of the target into PC
```

```
MIN:
    LDI     XH, low(DATA)    ; Initialize the X register to point to
    LDI     XL, high(DATA)   ; the beginning of the array
    LD      R18, X+          ; Load the first array value and post-increment X
    LDI     R19, 7           ; initialize count (8-1)
LOOP:
    LD      R17, X+          ; Load the array value and post-increment X
    CP      R17, R18         ; Compare loaded value with previous smallest value
    BRGE    SKIP             ; If greater than or equal to, then skip.
    MOV     R18, R17         ; Else, move loaded value to temporary register
SKIP:
    DEC     R19              ; Decrement count
    BRNE    LOOP             ; Check if count has reached zero, if not jump to loop
DONE:
    ST      X, R18           ; When done, store the smallest value in location $0008
    RET                      ; Return from subroutine
```

```
MIN:   LDI    XH, high(DATA)   0060: 1110 0000 1011 0001
       LDI    XL, low(DATA)    0061: 1110 0000 1010 0000
       LD     R18, X+          0062: 1001 0001 0010 1101
       LDI    R19, 7           0063: 1110 0000 0011 0111
LOOP:  LD     R17, X+          0064: 1001 0001 0001 1101
       CP     R17, R18         0065: 0001 0111 0001 0010
       BRGE   SKIP             0066: 1111 0100 0000 1100
       MOV    R18, R17         0067: 0010 1111 0010 0001
SKIP:  DEC    R19              0068: 1001 0101 0011 1010
       BRNE   LOOP             0069: 1111 0111 1101 0001
DONE:  ST     X, R18           006A: 1001 0011 0010 1100
       RET                     006B: 1001 0101 0000 1000
```

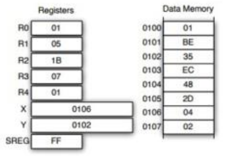
```
ISZ    Y       ; M(Y) ← M(Y) + 1, If (M(Y)+1=0) Then PC ← PC + 1
```

Fetch Cycle
Step 1: MAR ← PC;
Step 2: MDR ← M(MAR), PC ← PC+1 ; Read inst. & increment PC
Step 3: IR ← MDR_{opcode}, MAR ← $MDR_{address}$

Execute Cycle
Step 1: MDR ← M(MAR), TEMP ← AC ; Read M(Y) from memory and save the content of A

Step 2: AC ← MDR ; Transfer M(Y) into AC
Step 3: AC ← AC +1 ; Increment M(Y)
Step 4: MDR ← AC ; Transfer M(Y)+1 into MDR
Step 5: M(MAR) ← MDR, AC ← TEMP, ; Store the M(Y)+1 back into memory and restore AC, and at
if (AC=0) then PC ← PC + 1 ; the same time increment PC if AC =0

```
(i)   CPI    R28, 2
(ii)  STD    Y+0x05,R2
(iii) LD     R3, -X
(iv)  EOR    R1, R3
(v)   SBIW   R29:R28, 8
(vi)  SBR    R26, 7
```



Solution

(i) Nothing happens to any of the registers or memory locations. CPI basically performs a subtract and only sets the appropriate C, V, N, and S flags in SREG register.

(ii) Effective address is \$0102 + \$0005 = \$0107, and thus M[\$0107] changes \$1B.

(iii) X is pre-decremented to \$0105. R3 changes to M[\$0105] = \$2D.

(iv) Since 00000101 ⊕ 00000111 = 00000010, R1 changes to 02_{H}.

(v) Since R29:R28 is equal to Y, Y changes to \$0102 + \$0008 = \$00FA. Thus, R29 = \$00 and R28 = \$FA.

(vi) Since R26 = 0b00000110, R26 becomes 0b10000110 = \$86.

Control Signals	IF	Instructions										
		ADD	ORI	BREQ		LD		ST		CALL		
		EX1	EX1	EX1		EX1	EX2	EX1	EX2	EX1	EX2	EX3
				Z=0	Z=1							
MJ	0	x	x	1	1	x	x	x	x	x	x	1
MK	0	x	x	x	x	x	x	x	x	1	x	x
ML	0	x	x	x	x	x	x	x	x	0	x	x
IR_en	1	x	x	x	x	0	x	0	x	0	0	x
PC_en	1	0	0	0	1	0	0	0	0	x	x	1
PCh_en	0	0	0	0	0	0	0	0	0	0	0	0
PCl_en	0	0	0	0	0	0	0	0	0	0	0	0
NPC_en	1	x	x	x	x	x	x	x	x	1	0	x
SP_en	0	0	0	0	0	0	0	0	0	0	1	1
DEMUX	x	x	x	x	x	x	x	x	x	x	x	x
MA	x	1	0	x	x	x	x	x	x	x	x	x
MB	x	0	0	x	x	x	1	x	x	x	x	x
ALU_f	xxxx	0000	1001	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx	xxxx
MC	xx	00	00	xx	xx	xx	00	xx	xx	xx	xx	xx
RF_wA	0	0	0	0	0	0	0	0	0	0	0	0
RF_wB	0	1	1	0	0	0	1	0	0	0	0	0
MD	x	x	x	x	x	x	x	x	1	x	0	0
ME	x	x	x	x	x	x	1	x	1	x	0	0
DM_r	x	x	x	x	x	x	1	x	0	x	0	0
DM_w	0	0	0	0	0	0	0	0	1	0	1	1
MF	x	x	x	0	0	x	x	x	x	x	x	x
MG	x	x	x	0	0	1	x	1	x	x	x	0
Adder_f	xx	xx	xx	00	00	11	xx	11	xx	xx	xx	11
Inc_Dec	x	x	x	x	x	x	x	x	x	x	1	1
MH	x	x	x	x	x	1	x	1	x	x	x	x
MI	x	x	x	x	x	x	x	x	x	x	0	1

RF R/W Ports	IF	Instructions									
		ADD	ORI	BREQ	LD		ST		CALL		
		EX1	EX1	EX1	EX1	EX2	EX1	EX2	EX1	EX2	EX3
wA	x	x	x	x	x	x	x	x	x	x	x
wB	x	Rd	Rd	x	x	Rd	x	x	x	x	x
rA	x	Rd	Rd	x	Yh	x	Yh	x	x	x	x
rB	x	Rr	x	x	Yl	x	Yl	Rd	x	x	x

Stage	Normal	Displacement	Pre-Decrement	Post-Increment
EX1	DMAR ← AR	DMAR ← AR+q	DMAR ← AR-1, AR ← AR-1	DMAR ← AR, AR ← AR+1
EX2	Loads		Stores	
	Rd ← M[DMAR]		M[DMAR] ← Rr	

Mode	WGM01	WGM00	TOP	Update of OCR0 at	TOV0 Flag Set on
Normal	0	0	0xFF	Immediate	MAX
Phase Correct PWM	0	1	0xFF	TOP	BOTTOM
CTC	1	0	OCR0	Immediate	MAX
Fast PWM	1	1	0xFF	TOP	MAX

0	0000	Normal	0xFFFF	Immediate	MAX
4	0100	CTC	OCR1A	Immediate	MAX
5	0101	Fast PWM, 8-bit	0x00FF	TOP	TOP
6	0110	Fast PWM, 9-bit	0x01FF	TOP	TOP
7	0111	Fast PWM, 10-bit	0x03FF	TOP	TOP
12	1100	CTC	ICR1	Immediate	TOP
14	1110	Fast PWM	ICR1	TOP	TOP
15	1111	Fast PWM	OCR1A	TOP	TOP

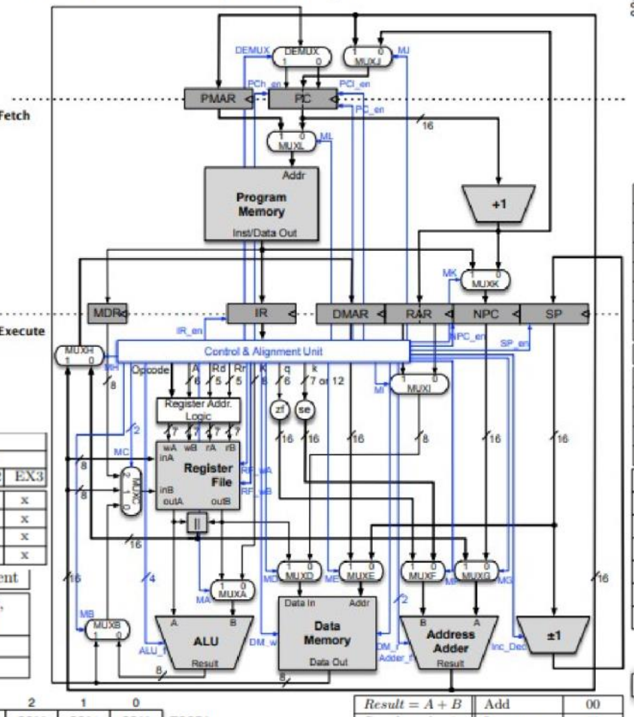
Since the memory contains 128K (2^{17}), 17 bits are required for the address field. There are 32 registers, thus 5 bits are needed for the register field. This leaves 32 minus 17+5+2 (addressing mode) = 24, or 8 bits for the opcode field. Thus, this allows 2^8 = 256 different opcodes (or instructions). The instruction format is shown below.

8	2	5	17
Opcode	AM	Reg	Address

From (a), we see that 17 bits are required to address the entire memory, thus MAR and PC each have 17 bits. Since a memory words consists of 32 bits, MDR and AC each have 32 bits. The number of bits required for IR depends on whether you assume the IR can hold the entire instruction or just the opcode. If IR can hold the entire instruction, then it is 32 bits. If it just holds the opcode, then it is 8 bits. Depending on the assumption, the other possible answers are 10 bits (include the AM field) and 15 bits (include the AM field and the Reg field).

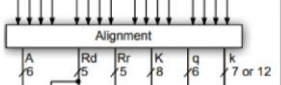
Fetch cycle
Step 1: MAR ← PC;
Step 2: MDR ← M(MAR), PC ← PC+1 ; Get the high byte of the instruction and increment PC
Step 3: IR ← MDR ; At this point, CU knows this is ADIW
Step 4: MAR ← PC;
Step 5: MDR ← M(MAR), PC ← PC+1 ; Get the low byte of the instruction and increment PC

Execute cycle
Step 6: AC ← R30
Step 7: AC ← AC + MDR ; Add 32 to ZL
Step 8: R30 ← AC ; Write back to register file
Step 9: AC ← R31
Step 10: If (C==1) then AC ← AC +1 ; Increment ZH if there was a carry
Step 11: R31 ← AC ; Write it back to register file



$Result = A + B$	Add	0000
$Result = A + B + C$	Add with carry (C)	0001
$Result = A + \bar{B} + 1$	Subtract	0010
$Result = \bar{B} + 1$	Negate	0011
$Result = A + 1$	Increment A	0100
$Result = A - 1$	Decrement A	0101
$Result = A$	Move A	0110
$Result = B$	Move B	0111
Logic Operations		
$Result = A \wedge B$	AND	1000
$Result = A \vee B$	OR	1001
$Result = A \oplus B$	Exclusive-OR (EOR)	1010
$Result = \bar{B}$	Complement	1011
$Result = B - 1$	Decrement B	1100

----	--rd	dddd	rrrr	Two-operand
----	KKKK	dddd	KKKK	Immediate
----	-AAd	dddd	AAAA	I/O
----	---d	dddd	----	Single-operand
--q-	qq-d	dddd	-qqq	Displacement
----	kkkk	kkkk	kkkk	12-bit PC-relative
----	--kk	kkkk	k---	7-bit PC-relative



$Result = A + B$	Add	00
$Result = A + 1$	Increment	01
$Result = A - 1$	Decrement	10
$Result = A$	Move A	11

7	6	5	4	3	2	1	0	
FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00	TCCR0
W (0)	R/W (0)	R/W (0)	R/W (0)	R/0 (0)	R/W (0)	R/W (0)	R/W (0)	

Bit 7 - Force Output Compare
Bit 6, 3 - Waveform Generation Mode
Bit 5, 4 - Compare Output Mode
Bit 2:0 - Clock Select

7	6	5	4	3	2	1	0	
COM1A1	COM1A0	COM1B1	COM1B0	COM1C1	COM1C0	WGM11	WGM10	TCCR1A
R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/0 (0)	R/W (0)	R/W (0)	R/W (0)	

Bit 7:6 - Compare Output Mode for Channel A
Bit 5:4 - Compare Output Mode for Channel B
Bit 3:2 - Compare Output Mode for Channel C
Bit 1:0 - Waveform Generation Mode bits 1:0

7	6	5	4	3	2	1	0	
ICNC1	ICES1	-	WGM13	WGM12	CS12	CS11	CS10	TCCR1B
R/W (0)	R/W (0)	R (0)	R/W (0)	R/0 (0)	R/W (0)	R/W (0)	R/W (0)	

Bit 7 - Input Capture Noise Canceler
Bit 6 - Input Capture Edge Select
Bit 5 - Reserved
Bit 4:3 - Waveform Generation Mode bits 3:2
Bit 2:0 - Clock Select

7	6	5	4	3	2	1	0	
FOC1A	FOC1B	FOC1C	-	-	-	-	-	TCCR1C
W (0)	W (0)	W (0)	R (0)	R (0)	R (0)	R (0)	R (0)	

Bit 7 - Force Output Compare for Channel A
Bit 6 - Force Output Compare for Channel B
Bit 5 - Force Output Compare for Channel C
Bit 4:0 - Reserved

COM1A1:0 COM1B1:0 COM1C1:0	Normal	CTC	Fast PWM
00	Normal port operation, OC1A-C disconnected		
01	Toggle OC1A-C on compare match		Reserved
10	Clear OC1A-C on compare match		Clear OC1A-C on compare match, set OC1A-C at BOTTOM (non-inverting)
11	Set OC1A-C on compare match		Set OC1A-C on compare match, clear OC1A-C at BOTTOM (inverting)

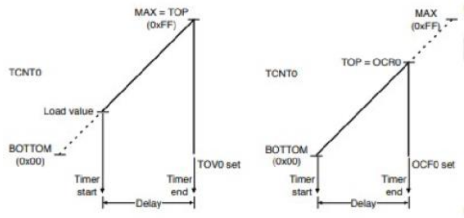
0	0	Normal port operation, OC0 disconnected	
0	1	Toggle OC0 on compare match	Reserved
1	0	Clear OC0 on compare match	Clear OC0 on compare match, set OC0 at TOP (non-inverting)
1	1	Set OC0 on compare match	Set OC0 on compare match, clear OC0 at TOP (inverting)

$$Delay_{Normal} = \frac{(MAX + 1 - value) \cdot prescale}{clk_{I/O}},$$
$$Delay_{CTC} = \frac{(TOP + 1) \cdot prescale}{clk_{I/O}}, \quad f_{PWM} = \frac{clk_{I/O}}{prescale \cdot 256},$$

UMSEL*n*	0	Asynchronous mode
	1	Synchronous mode

UCPOL*n* (synchronous)	0	Data changes/sampled at rising/falling XCK edge
	1	Data changes/sampled at falling/rising XCK edge

BOTTOM	The Timer/Counter reaches the BOTTOM when it becomes zero (i.e., 0x00 or 0x0000).
MAX	The Timer/Counter reaches its MAX when it becomes 0xFF or 0xFFFF.
TOP	The Timer/Counter reaches the TOP when it becomes the highest value in the count sequence. The TOP value can be the value stored in one of the OCRs, i.e., OCR0, OCR1A, OCR1B, and OCR1C, or assigned to 0xFF or 0xFFFF (i.e., MAX). The assignment is dependent on the mode of operation.



(a) Normal mode (b) CTC mode

1	\$0000	RESET	Hardware Reset
2	\$0002	INT0	External Interrupt Request 0
3	\$0004	INT1	External Interrupt Request 1
4	\$0006	INT2	External Interrupt Request 2
5	\$0008	INT3	External Interrupt Request 3
6	\$000A	INT4	External Interrupt Request 4
7	\$000C	INT5	External Interrupt Request 5
8	\$000E	INT6	External Interrupt Request 6
9	\$0010	INT7	External Interrupt Request 7
10	\$0012	TIMER2 COMP	Timer/Counter2 Compare Match
11	\$0014	TIMER2 OVF	Timer/Counter2 Overflow
12	\$0016	TIMER1 CAPT	Timer/Counter1 Capture Event
13	\$0018	TIMER1 COMPA	Timer/Counter1 Compare Match A
14	\$001A	TIMER1 COMPB	Timer/Counter1 Compare Match B
15	\$001C	TIMER1 OVF	Timer/Counter1 Overflow
16	\$001E	TIMER0 COMP	Timer/Counter0 Compare Match
17	\$0020	TIMER0 OVF	Timer/Counter0 Overflow
18	\$0022	SPI, STC	SPI Serial Transfer Complete
19	\$0024	USART0, RX	UASRT0, Rx Complete
20	\$0026	USART0, UDRE	USART0 Data Register Empty
21	\$0028	USART0, TX	USART0, Tx Complete
22	\$002A	ADC	ADC Conversion Complete
23	\$002C	EE READY	EEPROM Ready
24	\$002E	ANALOG COMP	Analog Comparator
25	\$0030	TIMER1 COMPC	Timer/Counter1 Compare Match C
26	\$0032	TIMER3 CAPT	Timer/Counter3 Capture Event
27	\$0034	TIMER3 COMPA	Timer/Counter3 Compare Match A
28	\$0036	TIMER3 COMPB	Timer/Counter3 Compare Match B
29	\$0038	TIMER3 COMPC	Timer/Counter3 Compare Match C
30	\$003A	TIMER3 OVF	Timer/Counter3 Overflow
31	\$003C	USART1, RX	USART1, Rx Complete
32	\$003E	USART1, UDRE	USART1 Data Register Empty
33	\$0040	USART1, TX	USART1, Tx Complete
34	\$0042	TWI	Two-wire Serial Interface
35	\$0044	SPM READY	Store Program Memory Ready

CS02	CS01	CS00	Description
0	0	0	No clock source
0	0	1	$clk_{I/O}$
0	1	0	$clk_{I/O}/8$
0	1	1	$clk_{I/O}/64$
1	0	0	$clk_{I/O}/256$
1	0	1	$clk_{I/O}/1024$
1	1	0	External clock source on T1 pin. Clocked on falling edge
1	1	1	External clock source on T1 pin. Clocked on rising edge

Clock Select bits in TCCR0.			
CS02	CS01	CS00	Description
0	0	0	No clock source
0	0	1	$clk_{I/O}$
0	1	0	$clk_{I/O}/8$
0	1	1	$clk_{I/O}/32$
1	0	0	$clk_{I/O}/64$
1	0	1	$clk_{I/O}/128$
1	1	0	$clk_{I/O}/256$
1	1	1	$clk_{I/O}/1024$

USART Control and Status Register A (UCSRnA)

7	6	5	4	3	2	1	0
RXCn	TXCn	UDREn	FEn	DORn	UPEn	U2Xn	MPCMn
R (0)	R/W (0)	R (1)	R (0)	R (0)	R (0)	R/W (0)	R/W (0)

Bit 7 - USART Receive Complete
Bit 6 - USART Transmit Complete
Bit 5 - USART Data Register Empty
Bit 4 - Frame error
Bit 3 - Data OverRun
Bit 2 - Parity Error
Bit 1 - Double USART Transmission Speed
Bit 0 - Multi-Processor Communication Mode

USART Control and Status Register B (UCSRnB)

7	6	5	4	3	2	1	0
RXCIEn	TXCIEn	UDRIEn	RXENn	TXENn	UCSZn2	RXB8n	TXB8n
R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R (0)	R/W (0)

Bit 7 - RX Complete Interrupt Enable
Bit 6 - TX Complete Interrupt Enable
Bit 5 - USART Data Register Empty Interrupt Enable
Bit 4 - Receiver Enable
Bit 3 - Transmitter Enable
Bit 2 - Character Size (combine with UCSZn1:0 in UCSRnC)
Bit 1 - Receive Data Bit 8
Bit 0 - Transmit Data Bit 8

USART Control and Status Register C (UCSRnC)

7	6	5	4	3	2	1	0
-	UMSELn	UPMn1	UPMn0	USBSn	UCSZn1	UCSZn0	UCPOLn
R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (1)	R/W (1)	R/W (0)

Bit 7 - Reserved Bit
Bit 6 - USART Mode Select
Bit 5:4 - Parity mode
Bit 3 - Stop bit select
Bit 2:1 - Character size UCSZn2:0
Bit 0 - Clock Polarity

External Interrupt Flag Register (EIFR)

7	6	5	4	3	2	1	0	
INTF7	INTF6	INTF5	INTF4	INTF3	INTF2	INTF1	INTF0	EIFR
R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	

INTFn = 1 Triggers interrupt request

External Interrupt Mask Register (EIMSK)

7	6	5	4	3	2	1	0	
INT7	INT6	INT5	INT4	INT3	INT2	INT1	INT0	EIMSK
R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	

INTn = 1 Enables interrupt

External Interrupt Control Register A (EICRA)

7	6	5	4	3	2	1	0	
ISC31	ISC30	ISC21	ISC20	ISC11	ISC10	ISC01	ISC00	EICRA
R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	

External Interrupt Control Register B (EICRB)

7	6	5	4	3	2	1	0	
ISC71	ISC70	ISC61	ISC60	ISC51	ISC50	ISC41	ISC40	EICRB
R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	

ISCn1:0 External Interrupt n Sense Control Bits
00 - Low level generates an interrupt request.
01 - Reserved (for ISC3-0)
Any logical change on generates an interrupt request (ISC7-4)
10 - Falling edge generates an interrupt request.
11 - Rising edge generates an interrupt request.

```
; Initialize stack
ldi   mpr, high(RAMEND)
out   SPH, mpr
ldi   mpr, low(RAMEND)
out   SPL, mpr

; Initialize I/O Ports
ldi   mpr, (1<<PE1)        ; Set Port E pin 0 (RXD0) for input ar
out   DDRE, mpr            ; Port E pin 1 (TXD0) for output

; Initialize USART0
ldi   mpr, (1<<U2X0)       ; Set double data rate
out   UCSR0A, mpr          ;

; Set baudrate at 2400
ldi   mpr, high(832)       ; Load high byte of 0x0340
sts   UBRR0H, mpr          ; UBRR0H in extended I/O space
ldi   mpr, low(832)        ; Load low byte of 0x0340
out   UBRR0L, mpr          ;

; Set frame format: 8 data, 2 stop bits, asynchronous
ldi   mpr, (0<<UMSEL0 | 1<<USBS0 | 1<<UCSZ01 | 1<<UCSZ00)
sts   UCSR0C, mpr          ; UCSR0C in extended I/O space

; Enable both transmitter and receiver, and receive interrupt
ldi   mpr, (1<<TXEN0 | 1<<RXEN0 | 1<<RXCIE0)
out   UCSR0B, mpr          ;

sei                        ; Enable global interrupt
```

$$\text{Baud Rate} = \frac{f_{CLK}}{16 \times (UBRR + 1)},$$

$$UBRR = \frac{f_{CLK}}{16 \times (\text{Baud Rate})} - 1$$

UCSZn2:0	0	0	0	5-bit
	0	0	1	6-bit
	0	1	0	7-bit
	0	1	1	8-bit
	1	0	0	Reserved
	1	0	1	Reserved
	1	1	0	Reserved
	1	1	1	9-bit
UPMn1:0	0	0		No parity
	1	0		Even parity
	1	1		Odd parity
USBSn	0			1 stop bit
	1			2 stop bits

a

RXCn	0	Receive incomplete
	1	Receive complete
TXCn	0	Transmit incomplete
	1	Transmit complete
UDREn	0	UDRn (trasnmit buffer) full
	1	UDRn (transmit buffer) empty