University of Victoria
Department of Computer Science
CSC 230 - Introduction to Computer Architecture

Midterm Exam, October 25, 2019

NAME (print): [redacted]

STUDENT ID: V0[redacted]

SIGNATURE:

TIME: 50 minutes

Question	1	2	3	4	5	6	Total
Points	1	2	2	3	2/2	13	25
Grade	1	0	2	3	2	10	18

STUDENTS MUST CHECK THE NUMBER OF PAGES IN THE EXAMINATION BOOKLET BEFORE BEGINNING TO WRITE AND REPORT ANY CONCERNS IMMEDIATELY TO THE INVIGILATOR

◇ Calculators and other electronic devices are **NOT** permitted.

◇ Turn **OFF** your electronic devices and place them in your bag and under your seat. Do **NOT** reach into your bag for anything during the exam.

◇ Use **ink** to fill out your name, ID, and signature on this page.

◇ All questions must be answered on the examination paper (this booklet).

◇ A copy of the *AVR Instruction Set Summary* is provided with this exam.

◇ Each question's weight is listed in brackets next to each question's number.

◇ Read through the entire exam from beginning to end before starting to answer the questions.

◇ This exam has six pages including this cover page plus nine pages from the AVR Instruction Set (pp. 11-15, p. 17, pp. 148-149). There are 14 pages in total.

This page is provided to you as scrap paper. Exam questions begin on the next page.

1. **[1 pt.]** Today, most personal computers follow the architectural design where the CPU retrieves (fetches) one instruction at a time from the data memory via the same bus as it uses to store and retrieve information to and from the data memory. Which architecture does the statement above describe? Pick one:

   (A) Systolic
   (B) Harvard
   [✓ (C) von Neumann]
   (D) Both A and C.

1

2. Give the **decimal** representation of 0xFB when interpreted as:

   b. **[1 pt.]** an unsigned 8-bit binary integer;

   0b11111011 α 251

   a. **[1 pt.]** a signed 2's complement 8-bit binary integer.

   00000100
   0b00000101 α -5

0
01
10
11
100
101
110
111
1000
1001
1010
1011
1100
1101
1110
1111

3. **[2 pt.]** Specify the missing mask and the bitwise operation that are necessary to complete the following expression. The mask is an 8-bit binary number and the operation must be one of the following operations (in AVR notation): either AND, ASR, COM, EOR, LSL, LSR, OR, ROL, ROR.

   0b01010000 = 0b01010000 AND 0b01011110
   result mask op. operand

2

4. **[3 pt.]** What are the contents of the status register bits H (half-carry), S (sign), V (overflow), N (negative), C (carry), Z (zero), after the following three instructions are executed:

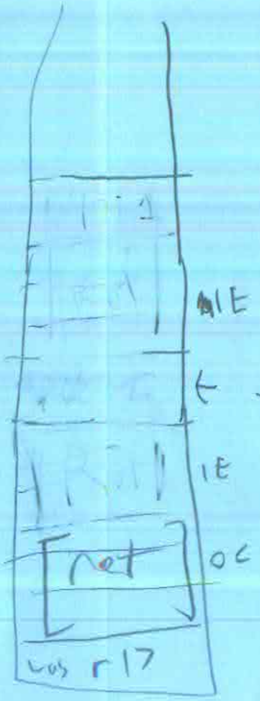
```
LDI  r16, 0b11011001
LDI  r17, 0b01101010
ADD  r16, r17
```

```
 1 1 1 1
  11011001
+ 01101010
  01000011
```

3/3

SR bits	H	S	V	N	C	Z
Value	1	0	0	0	1	0
	✓	✓	✓	✓	✓	✓

5. The next two questions (5a and 5b) are about the following syntactically correct AVR program, which computes the N^{th} triangle number using a recursive function and stores the entire sequence of the intermediate values in memory, starting from 1 and ending with the final sum. The function parameters are passed in registers r24, r30, and r31. Remember, computing the N^{th} triangle number is like computing factorial but using addition instead of multiplication. E.g., calling the function with $n = 5$ will result in the following sequence: 1, 3, 6, 10, 15.



```
0x02       LDI ~~r10~~ 30,  low(input<<1)
0x04       LDI r31 ,  high(input<<1)
0x06       LPM r24 ,  Z
0x08       LDI r30 ,  low(output)
0x0A       LDI r31 ,  high(output)
0x0C       CALL nth_triangle

      done:
→0x10      RJMP done

      nth_triangle:
0x14       PUSH r17
0x16       CPI r24 , 0
0x18       BREQ base_case
0x1A       MOV r17 , r24
0x1C       DEC r24
0x1E       CALL nth_triangle
0x20       ADD r24 , r17
0x22       ST Z+, r24
      base_case:
0x24       POP r17
0x26       RET

0x2A  input: .db 2, 0

      .dseg
      .org 0x200
      output: .BYTE 100
```

Z

r30	r31	r24	r17		
~~lowinput~~ lowout	~~hiinput~~ hi out	~~2~~ ~~1~~ ~~1~~ ~~0~~ ~~1~~ 3	~~2~~ ~~1~~ 2		

1	200
3	201
	202

0x--

a. **[2 pt.]** Using 8-bit hexadecimal notation, report the contents of XRAM at memory address 0x0201 when the program reaches line 0x10 for the first time during execution?

0x03 ✓

b. **[2 pt.]**What is the binary encoding of the instruction on line 0x22 when the program reaches this line for the first time during execution?

$\underbrace{1001}_{\text{fixed}}\ \underbrace{\underbrace{001}_{\text{fixed}}\ \underbrace{0}_{r}}\ \underbrace{0010}_{rrrr}\ \underbrace{0001}_{\text{fixed}}$

α α α

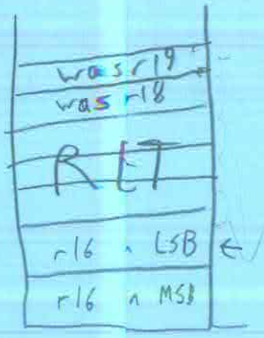
0b1000 0001 1001 0011

6. **[13 pt.]** Complete the following program by writing an AVR assembly function rotate_16, which would rotate a 16-bit number by one bit to the right. The number is provided by the caller via the stack with the least significant byte on top (last in) and the result is returned in the same spot with the least significant byte on top (first out). Upon reaching the end of your function (just before the RET instruction), the contents of all of the registers must be the same as they were before the start of your function execution (i.e. protect all registers that your function uses). If you need it, there is more space on the next page.

10

```
0x02      LDI r16, 0b11001001
0x04      PUSH r16
0x06      LDI r16, 0b01000101
0x08      PUSH r16
0x0A      CLR r16
0x0C      CALL rotate_16
          POP r16
          POP r17
     done:
0x0E      RJMP done

     rotate_16:
```



protect Z & Y

```
          Push r18
          Push r19
          ld ZL, SL
          ld ZH, SH
          ldd YL, ZL + 5
          ldd YH, ZH + 5

          ld r18, Y+
          ld r19, Y

          lsr r18
          ror r19
          cpi C, 1
          brne nocarry
          push r20
          ldi r20, 0b10000000
          OR r18, r20
          pop r20
          jmp endfunc
 nocarry: push r20
          ldi r20, 01111111
          AND r18, r20
          pop r20
endfunc:
          str r19, Y-
          sts r18, Y
          pop r19
          pop r18
          ret
```

LDS or IN

X

✓

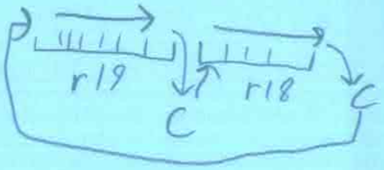
?

=

✓

1
1
1
1
0
1
1
3

need to rotate right; 2 bytes



use carry flag to do so

the only issues

Complete Instruction Set Summary

Instruction Set Summary

Mnemonics	Operands	Description	Operation	Flags	#Clocks	#Clocks XMEGA
Arithmetic and Logic Instructions						
ADD	Rd, Rr	Add without Carry	Rd ← Rd + Rr	Z,C,N,V,S,H	1	
ADC	Rd, Rr	Add with Carry	Rd ← Rd + Rr + C	Z,C,N,V,S,H	1	
ADIW	Rd, K	Add Immediate to Word	Rd ← Rd + 1:Rd + K	Z,C,N,V,S	2	
SUB	Rd, Rr	Subtract without Carry	Rd ← Rd - Rr	Z,C,N,V,S,H	1	
SUBI	Rd, K	Subtract Immediate	Rd ← Rd - K	Z,C,N,V,S,H	1	
SBC	Rd, Rr	Subtract with Carry	Rd ← Rd - Rr - C	Z,C,N,V,S,H	1	
SBCI	Rd, K	Subtract Immediate with Carry	Rd ← Rd - K - C	Z,C,N,V,S,H	1	
SBIW	Rd, K	Subtract Immediate from Word	Rd + 1:Rd ← Rd + 1:Rd - K	Z,C,N,V,S	2	
AND	Rd, Rr	Logical AND	Rd ← Rd • Rr	Z,N,V,S	1	
ANDI	Rd, K	Logical AND with Immediate	Rd ← Rd • K	Z,N,V,S	1	
OR	Rd, Rr	Logical OR	Rd ← Rd v Rr	Z,N,V,S	1	
ORI	Rd, K	Logical OR with Immediate	Rd ← Rd v K	Z,N,V,S	1	
EOR	Rd, Rr	Exclusive OR	Rd ← Rd ⊕ Rr	Z,N,V,S	1	
COM	Rd	One's Complement	Rd ← $FF - Rd	Z,C,N,V,S	1	
NEG	Rd	Two's Complement	Rd ← $00 - Rd	Z,C,N,V,S,H	1	
SBR	Rd,K	Set Bit(s) in Register	Rd ← Rd v K	Z,N,V,S	1	
CBR	Rd,K	Clear Bit(s) in Register	Rd ← Rd • ($FFh - K)	Z,N,V,S	1	
INC	Rd	Increment	Rd ← Rd + 1	Z,N,V,S	1	
DEC	Rd	Decrement	Rd ← Rd - 1	Z,N,V,S	1	
TST	Rd	Test for Zero or Minus	Rd ← Rd • Rd	Z,N,V,S	1	
CLR	Rd	Clear Register	Rd ← Rd ⊕ Rd	Z,N,V,S	1	
SER	Rd	Set Register	Rd ← $FF	None	1	
MUL(1)	Rd,Rr	Multiply Unsigned	R1:R0 ← Rd x Rr (UU)	Z,C	2	
MULS(1)	Rd,Rr	Multiply Signed	R1:R0 ← Rd x Rr (SS)	Z,C	2	
MULSU(1)	Rd,Rr	Multiply Signed with Unsigned	R1:R0 ← Rd x Rr (SU)	Z,C	2	
FMUL(1)	Rd,Rr	Fractional Multiply Unsigned	R1:R0 ← Rd x Rr<<1 (UU)	Z,C	2	
FMULS(1)	Rd,Rr	Fractional Multiply Signed	R1:R0 ← Rd x Rr<<1 (SS)	Z,C	2	
FMULSU(1)	Rd,Rr	Fractional Multiply Signed with Unsigned	R1:R0 ← Rd x Rr<<1 (SU)	Z,C	2	

DES	K	Data Encryption	if (H = 0) then R15:R0 ← Encrypt(R15:R0, K)
			else if (H = 1) then R15:R0 ← Decrypt(R15:R0, K)
						1/2

Branch Instructions						
RJMP	k	Relative Jump	PC ← PC + k + 1	None	2	
IJMP(1)		Indirect Jump to (Z)	PC(15:0) ← Z, PC(21:16) ← 0	None	2	
EIJMP(1)		Extended Indirect Jump to (Z)	PC(15:0) ← Z, PC(21:16) ← EIND	None	2	
JMP(1)	k	Jump	PC ← k	None	3	

Mnemonics	Operands	Description	Operation	Flags	#Clocks	#Clocks XMEGA
RCALL	k	Relative Call Subroutine	PC ← PC + k + 1	None	3 / 4[3][5]	2 / 3[3]
ICALL[1]		Indirect Call to (Z)	PC(15:0) ← Z, PC(21:16) ← 0	None	3 / 4[3]	2 / 3[3]
EICALL		Extended Indirect Call to (Z)	PC(15:0) ← Z, PC(21:16) ← EIND	None	4 [3]	3 [3]
CALL[1]	k	call Subroutine	PC ← k	None	4 / 5[3]	3 / 4[3]
RET		Subroutine Return	PC ← STACK	None	4 / 5[3]	
RETI		Interrupt Return	PC ← STACK	I	4 / 5[3]	
CPSE	Rd,Rr	Compare, Skip if Equal	if (Rd = Rr) PC ← PC + 2 or 3	None	1 / 2 / 3	
CP	Rd,Rr	Compare	Rd - Rr	Z,C,N,V,S,H	1	
CPC	Rd,Rr	Compare with Carry	Rd - Rr - C	Z,C,N,V,S,H	1	
CPI	Rd,K	Compare with Immediate	Rd - K	Z,C,N,V,S,H	1	
SBRC	Rr, b	Skip if Bit in Register Cleared	if (Rr(b) = 0) PC ← PC + 2 or 3	None	1 / 2 / 3	
SBRS	Rr, b	Skip if Bit in Register Set	if (Rr(b) = 1) PC ← PC + 2 or 3	None	1 / 2 / 3	
SBIC	A, b	Skip if Bit in I/O Register Cleared	if (I/O(A,b) = 0) PC ← PC + 2 or 3	None	1 / 2 / 3	2 / 3 / 4
SBIS	A, b	Skip if Bit in I/O Register Set	If (I/O(A,b) =1) PC ← PC + 2 or 3	None	1 / 2 / 3	2 / 3 / 4
BRBS	s, k	Branch if Status Flag Set	if (SREG(s) = 1) then PC ← PC + k + 1	None	1 / 2	
BRBC	s, k	Branch if Status Flag Cleared	if (SREG(s) = 0) then PC ← PC + k + 1	None	1 / 2	
BREQ	k	Branch if Equal	if (Z = 1) then PC ← PC + k + 1	None	1 / 2	
BRNE	k	Branch if Not Equal	if (Z = 0) then PC ← PC + k + 1	None	1 / 2	
BRCS	k	Branch if Carry Set	if (C = 1) then PC ← PC + k + 1	None	1 / 2	
BRCC	k	Branch if Carry Cleared	if (C = 0) then PC ← PC + k + 1	None	1 / 2	
BRSH	k	Branch if Same or Higher	if (C = 0) then PC ← PC + k + 1	None	1 / 2	
BRLO	k	Branch if Lower	if (C = 1) then PC ← PC + k + 1	None	1 / 2	
BRMI	k	Branch if Minus	if (N = 1) then PC ← PC + k + 1	None	1 / 2	
BRPL	k	Branch if Plus	if (N = 0) then PC ← PC + k + 1	None	1 / 2	
BRGE	k	Branch if Greater or Equal, Signed	if (N ⊕ V= 0) then PC ← PC + k + 1	None	1 / 2	
BRLT	k	Branch if Less Than, Signed	if (N ⊕ V= 1) then PC ← PC + k + 1	None	1 / 2	
BRHS	k	Branch if Half Carry Flag Set	if (H = 1) then PC ← PC + k + 1	None	1 / 2	
BRHC	k	Branch if Half Carry Flag Cleared	if (H = 0) then PC ← PC + k + 1	None	1 / 2	
BRTS	k	Branch if T Flag Set	if (T = 1) then PC ← PC + k + 1	None	1 / 2	
BRTC	k	Branch if T Flag Cleared	if (T = 0) then PC ← PC + k + 1	None	1 / 2	
BRVS	k	Branch if Overflow Flag is Set	if (V = 1) then PC ← PC + k + 1	None	1 / 2	
BRVC	k	Branch if Overflow Flag is Cleared	if (V = 0) then PC ← PC + k + 1	None	1 / 2	
BRIE	k	Branch if Interrupt Enabled	if (I = 1) then PC ← PC + k + 1	None	1 / 2	
BRID	k	Branch if Interrupt Disabled	if (I = 0) then PC ← PC + k + 1	None	1 / 2	
		Data Transfer Instructions				
MOV	Rd, Rr	Copy Register	Rd ← Rr	None	1	
MOVW[1]	Rd, Rr	Copy Register Pair	Rd+1:Rd ← Rr+1:Rr	None	1	
LDI	Rd, K	Load Immediate	Rd ← K	None	1	
LDS[1]	Rd, k	Load Direct from data space	Rd ← (k)	None	1[5]/2[3]	2[3][4]
LD[2]	Rd, X	Load Indirect	Rd ← (X)	None	1[5]2[3]	1[3][4]

Mnemonics	Operands	Description	Operation	Flags	#Clocks	#Clocks XMEGA
LD(2)	Rd, X+	Load Indirect and Post-Increment	Rd ← (X) X ← X + 1	None	$2^{(3)}$	$1^{(3)(4)}$
LD(2)	Rd, -X	Load Indirect and Pre-Decrement	X ← X - 1, ← X - 1 Rd ← (X) ← (X)	None	$2^{(3)}/3^{(5)}$	$2^{(3)(4)}$
LD(2)	Rd, Y	Load Indirect	Rd ← (Y) ← (Y)	None	$1^{(5)}/2^{(3)}$	$1^{(3)(4)}$
LD(2)	Rd, Y+	Load Indirect and Post-Increment	Rd ← (Y) Y ← Y + 1	None	$2^{(3)}$	$1^{(3)(4)}$
LD(2)	Rd, -Y	Load Indirect and Pre-Decrement	Y ← Y - 1 Rd ← (Y)	None	$2^{(3)}/3^{(5)}$	$2^{(3)(4)}$
LDD(1)	Rd, Y+q	Load Indirect with Displacement	Rd ← (Y + q)	None	$2^{(3)}$	$2^{(3)(4)}$
LD(2)	Rd, Z	Load Indirect	Rd ← (Z)	None	$1^{(5)}/2^{(3)}$	$1^{(3)(4)}$
LD(2)	Rd, Z+	Load Indirect and Post-Increment	Rd ← (Z), Z ← Z+1	None	$2^{(3)}$	$1^{(3)(4)}$
LD(2)	Rd, -Z	Load Indirect and Pre-Decrement	Z ← Z - 1, Rd ← (Z)	None	$2^{(3)}/3^{(5)}$	$2^{(3)(4)}$
LDD(1)	Rd, Z+q	Load Indirect with Displacement	Rd ← (Z + q)	None	$2^{(3)}$	$2^{(3)(4)}$
STS(1)	k, Rr	Store Direct to Data Space	(k) ← Rd	None	$1^{(5)}/2^{(3)}$	$2^{(3)}$
ST(2)	X, Rr	Store Indirect	(X) ← Rr	None	$1^{(5)}/2^{(3)}$	$1^{(3)}$
ST(2)	X+, Rr	Store Indirect and Post-Increment	(X) ← Rr, X ← X + 1	None	$1^{(5)}/2^{(3)}$	$1^{(3)}$
ST(2)	-X, Rr	Store Indirect and Pre-Decrement	X ← X - 1, (X) ← Rr	None	$2^{(3)}$	$2^{(3)}$
ST(2)	Y, Rr	Store Indirect	(Y) ← Rr	None	$1^{(5)}/2^{(3)}$	$1^{(3)}$
ST(2)	Y+, Rr	Store Indirect and Post-Increment	(Y) ← Rr, Y ← Y + 1	None	$1^{(5)}/2^{(3)}$	$1^{(3)}$
ST(2)	-Y, Rr	Store Indirect and Pre-Decrement	Y ← Y - 1, (Y) ← Rr	None	$2^{(3)}$	$2^{(3)}$
STD(1)	Y+q, Rr	Store Indirect with Displacement	(Y + q) ← Rr	None	$2^{(3)}$	$2^{(3)}$
ST(2)	Z, Rr	Store Indirect	(Z) ← Rr	None	$1^{(5)}/2^{(3)}$	$1^{(3)}$
ST(2)	Z+, Rr	Store Indirect and Post-Increment	(Z) ← Rr Z ← Z + 1	None	$1^{(5)}/2^{(3)}$	$1^{(3)}$
ST(2)	-Z, Rr	Store Indirect and Pre-Decrement	Z ← Z - 1	None	$2^{(3)}$	$2^{(3)}$
STD(1)	Z+q,Rr	Store Indirect with Displacement	(Z + q) ← Rr	None	$2^{(3)}$	$2^{(3)}$
LPM(1)(2)		Load Program Memory	R0 ← (Z)	None	3	3
LPM(1)(2)	Rd, Z	Load Program Memory	Rd ← (Z)	None	3	3
LPM(1)(2)	Rd, Z+	Load Program Memory and Post-Increment	Rd ← (Z), Z ← Z + 1	None	3	3
ELPM(1)		Extended Load Program Memory	R0 ← (RAMPZ:Z)	None	3	
ELPM(1)	Rd, Z	Extended Load Program Memory	Rd ← (RAMPZ:Z)	None	3	
ELPM(1)	Rd, Z+	Extended Load Program Memory and Post-Increment	Rd ← (RAMPZ:Z), Z ← Z + 1	None	3	
SPM(1)		Store Program Memory	(RAMPZ:Z) ← R1:R0	None	-	-
SPM(1)	Z+	Store Program Memory and Post-Increment by 2	(RAMPZ:Z) ← R1:R0, Z ← Z + 2	None	-	-
IN	Rd, A	In From I/O Location	Rd ← I/O(A)	None	1	
OUT	A, Rr	Out To I/O Location	I/O(A) ← Rr	None	1	
PUSH(1)	Rr	Push Register on Stack	STACK ← Rr	None	2	$1^{(3)}$
POP(1)	Rd	Pop Register from Stack	Rd ← STACK	None	2	$2^{(3)}$

Mnemonics	Operands	Description	Operation	Flags	#Clocks	#Clocks XMEGA
XCH	Z, Rd	Exchange	(Z) ← Rd, Rd ← (Z)	None	1	
LAS	Z, Rd	Load and Set	(Z) ← Rd v (Z) Rd ← (Z)	None	1	
LAC	Z, Rd	Load and Clear	(Z) ← ($FF – Rd) • (Z) Rd ← (Z)	None	1	
LAT	Z, Rd	Load and Toggle	(Z) ← Rd ⊕ (Z) Rd ← (Z)	None	1	
Bit and Bit-test Instructions						
LSL	Rd	Logical Shift Left	Rd(n+1) ← Rd(n), Rd(0) ← 0, C ← Rd(7)	Z,C,N,V,H	1	
LSR	Rd	Logical Shift Right	Rd(n) ← Rd(n+1), Rd(7) ← 0, C ← Rd(0)	Z,C,N,V	1	
ROL	Rd	Rotate Left Through Carry	Rd(0) ← C, Rd(n+1) ← Rd(n), C ← Rd(7)	Z,C,N,V,H	1	
ROR	Rd	Rotate Right Through Carry	Rd(7) ← C, Rd(n) ← Rd(n+1), C ← Rd(0)	Z,C,N,V	1	
ASR	Rd	Arithmetic Shift Right	Rd(n) ← Rd(n+1), n=0..6	Z,C,N,V	1	
SWAP	Rd	Swap Nibbles	Rd(3..0) ↔ Rd(7..4)	None	1	
BSET	s	Flag Set	SREG(s) ← 1	SREG(s)	1	
BCLR	s	Flag Clear	SREG(s) ← 0	SREG(s)	1	
SBI	A, b	Set Bit in I/O Register	I/O(A, b) ← 1	None	1[5]2	1
CBI	A, b	Clear Bit in I/O Register	I/O(A, b) ← 0	None	1[5]/2	1
BST	Rr, b	Bit Store from Register to T	T ← Rr(b)	T	1	
BLD	Rd, b	Bit load from T to Register	Rd(b) ← T	None	1	
SEC		Set Carry	C ← 1	C	1	
CLC		Clear Carry	C ← 0	C	1	
SEN		Set Negative Flag	N ← 1	N	1	
CLN		Clear Negative Flag	N ← 0	N	1	
SEZ		Set Zero Flag	Z ← 1	Z	1	
CLZ		Clear Zero Flag	Z ← 0	Z	1	
SEI		Global Interrupt Enable	I ← 1	I	1	
CLI		Global Interrupt Disable	I ← 0	I	1	
SES		Set Signed Test Flag	S ← 1	S	1	
CLS		Clear Signed Test Flag	S ← 0	S	1	
SEV		Set Two's Complement Overflow	V ← 1	V	1	
CLV		Clear Two's Complement Overflow	V ← 0	V	1	
SET		Set T in SREG	T ← 1	T	1	
CLT		Clear T in SREG	T ← 0	T	1	
SEH		Set Half Carry Flag in SREG	H ← 1	H	1	
CLH		Clear Half Carry Flag in SREG	H ← 0	H	1	
MCU Control Instructions						
BREAK[1]		Break	(See specific descr. for BREAK)	None	1	

Mnemonics	Operands	Description	Operation	Flags	#Clocks	#Clocks XMEGA
NOP		No Operation		None	1	
SLEEP		Sleep	(see specific descr. for Sleep)	None	1	
WDR		Watchdog Reset	(see specific descr. for WDR)	None	1	

Notes:
1. This instruction is not available in all devices. Refer to the device specific instruction set summary.
2. Not all variants of this instruction are available in all devices. Refer to the device specific instruction set summary.
3. Cycle times for Data memory accesses assume internal memory accesses, and are not valid for accesses via the external RAM interface.
4. One extra cycle must be added when accessing Internal SRAM.
5. Number of clock cycles for Reduced Core tinyAVR.

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:

000	11rd	dddd	rrrr

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.

C: $Rd7\bullet Rr7+Rr7\bullet\overline{R7}+\overline{R7}\bullet Rd7$
Set if there was carry from the MSB of the result; cleared otherwise.

R (Result) equals Rd after the operation.

Example:

```
    add  r1,r2     ; Add r2 to r1 (r1=r1+r2)
    add  r28,r28   ; Add r28 to itself (r28=r28+r28)
```

Words: 1 (2 bytes)
Cycles: 1

ST (STD) – Store Indirect From Register to Data Space using Index Z

Description:

Stores one byte indirect with or without displacement from a register to data space. For parts with SRAM, the data space consists of the Register File, I/O memory and internal SRAM (and external SRAM if applicable). For parts without SRAM, the data space consists of the Register File only. The EEPROM has a separate address space.

The data location is pointed to by the Z (16 bits) Pointer Register in the Register File. Memory access is limited to the current data segment of 64K bytes. To access another data segment in devices with more than 64K bytes data space, the RAMPZ in register in the I/O area has to be changed.

The Z-pointer Register can either be left unchanged by the operation, or it can be post-incremented or pre-decremented. These features are especially suited for Stack Pointer usage of the Z-pointer Register, however because the Z-pointer Register can be used for indirect subroutine calls, indirect jumps and table lookup, it is often more convenient to use the X or Y-pointer as a dedicated Stack Pointer. Note that only the low byte of the Z-pointer is updated in devices with no more than 256 bytes data space. For such devices, the high byte of the pointer is not used by this instruction and can be used for other purposes. The RAMPZ Register in the I/O area is updated in parts with more than 64K bytes data space or more than 64K bytes Program memory, and the increment/decrement/displacement is added to the entire 24-bit address on such devices.

Not all variants of this instruction is available in all devices. Refer to the device specific instruction set summary.

The result of these combinations is undefined:

ST Z+, r30
ST Z+, r31
ST -Z, r30
ST -Z, r31

Using the Z-pointer:

	Operation:		Comment:
(i)	(Z) ←Rr		Z: Unchanged
(ii)	(Z) ← Rr	Z ← Z+1	Z: Post incremented
(iii)	Z ← Z - 1	(Z) ← Rr	Z: Pre decremented
(iv)	(Z+q) ← Rr		Z: Unchanged, q: Displacement

	Syntax:	Operands:	Program Counter:
(i)	ST Z, Rr	0 ≤ r ≤ 31	PC ← PC + 1
(ii)	ST Z+, Rr	0 ≤ r ≤ 31	PC ← PC + 1
(iii)	ST -Z, Rr	0 ≤ r ≤ 31	PC ← PC + 1
(iv)	STD Z+q, Rr	0 ≤ r ≤ 31, 0 ≤ q ≤ 63	PC ← PC + 1

16-bit Opcode :

(i)	1000	001r	rrrr	0000
(ii)	1001	001r	rrrr	0001
(iii)	1001	001r	rrrr	0010
(iv)	10q0	qq1r	rrrr	0qqq

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Example:

```
        clr     r31             ; Clear Z high byte
        ldi     r30,$60         ; Set Z low byte to $60
        st      Z+,r0           ; Store r0 in data space loc. $60(Z post inc)
        st      Z,r1            ; Store r1 in data space loc. $61
        ldi     r30,$63         ; Set Z low byte to $63
        st      Z,r2            ; Store r2 in data space loc. $63
        st      -Z,r3           ; Store r3 in data space loc. $62(Z pre dec)
        std     Z+2,r4          ; Store r4 in data space loc. $64
```

Words: 1 (2 bytes)

Cycles: 2

Cycles XMEGA: (i) 1
(ii) 1
(iii) 2
(iv) 2

Cycles Reduced Core tinyAVR: (i) 1
(ii) 1
(iii) 2