MID-TERM

ECE 375

Computer Organization and Assembly Language Programming

Winter 2018

Instructions

This exam consists of four hand-graded problems that should be presented in a well-organized and readable form. Be sure to state assumptions and explanatory comments as needed to clarify your work. For reference, you may use one 8.5x11 sheet of notes.

Name_

I.D.__

Problem #1 (25 pts) 24

Problem #2 (25 pts) 25

Problem #3 (25 pts) 25

Problem #4 (25 pts) 20

Total 94

Problem #1 [25 pts]
Answer the following AVR related questions:
[5 pts]
(a) Which of the machine code shown below is equivalent to the AVR instruction `ldi r26,30`?

(1) 1110 1101 0100 0110
(2) 1110 0001 1010 1110
(3) 1110 0011 1010 0000
(4) 1110 0000 1010 0010
(5) None of the above

1110 KKKK dddd KKKK
(leading 1 implied)

16+8+4+2= 30
11010
1010
16+8+2=26
11110
0001 1110

[5 pts]
(b) What *decimal* values will be contained in registers r1 and r2 after the execution of the following AVR code:

```
  ldi   r16, $18
1 mov   r2, r16
2 mov   r1, r2
3 rol   r2          ; Assume carry bit is 0
4 sub   r2, r1
```

(1) r1 = 18, r2 = 18
(2) r1 = 24, r2 = 0
(3) r1 = 24, r2 = 24
(4) r1 = 24, r2 = 48
(5) None of the above

R16 $18
R1 (2) $18
R2 $18 (1) ⇒ 0001 1000 = 0011 0000
carry 0
32+16=48
$30 (3)

$30
-$18

48
-24
24 ⇒ $18 (4)
00011000

[15 pts]
(c) Based on the initial register and data memory contents shown below (represented in hexadecimal), show how these contents are modified (in *hexadecimal*) after executing each of the following AVR assembly instructions. Do not be concerned about what happens to the Status Register (SREG) *after* the operation. *Instructions are unrelated.*

```
(i)   CP     R2, R1
(ii)  ST     -Y, R3
(iii) LSR    R2
(iv)  NEG    R3
(v)   SBIW   R29:R28, $1F
```

Registers

R0	01	
R1	05	
R2	1B	$0D (iii)
R3	07	$F9 (iv)
R4	01	
X	0106	
Y	0102	(v) $00D3
SREG	FF	

Data Memory

0100	01	
0101	BE	$07 (ii)
0102	35	
0103	EC	
0104	48	
0105	2D	
0106	04	
0107	02	

(i) Compares contents of R2 and R1 (R2−R1), uses result to set status register flags. No registers/data memory are modified.

(ii) Stores the value contained in R3 ($07) into the pre-decremented address pointed to by the Y register.
Y is initially pointing to $0102, so $0102 − 1 = $0101.
Therefore, after this operation, address $0101 in the data memory contains $07.

(iii) performs the logical shift right operation on R2, shifting a 0 into the most significant bit (and putting the lsb into the carry bit in SREG)

R2 ⇒ $1B = 0001 1011
0 >>
= 0000 1101 (1 in carry)
= $0D is the new value in R2

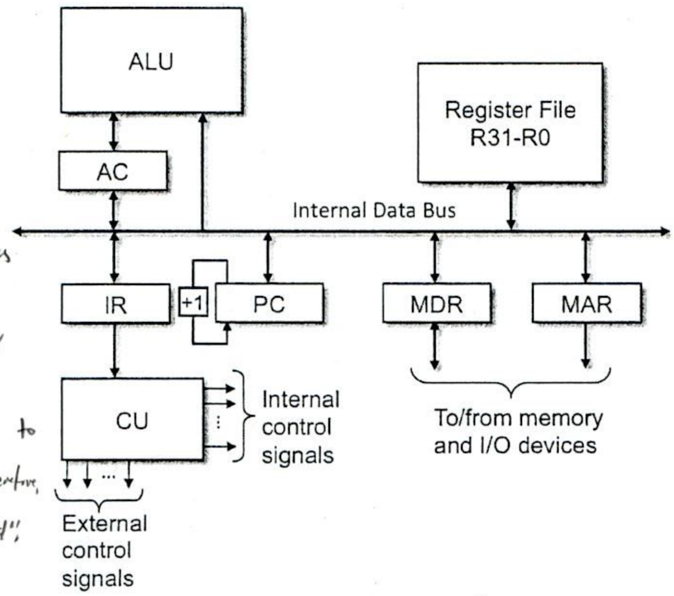
(iv) The two's compliment of R3 is taken.
Rd ← $00 − Rd
R3 $07 = 0000 0111
1111 1000
+ 1
1111 1001 ⇒ $F9 is now stored in R3.

-1
(v) The immediate (constant) value is subtracted from R29:R28, the Y register.
$0102 − $1F ⇒
258
− 31
227 ⇒ ~~$D3~~ **$E3** ⇒ $00D3
The value contained in Y is now $00D3

Problem #2 [25 pts]

Consider the internal structure of the pseudo-CPU discussed in class augmented with a *single-port register file* (i.e., only one register value can be read at a time) containing 32 8-bit registers (R0-R31). Suppose the pseudo-CPU can be used to implement the AVR instruction `LD R3, Y+`. Give the sequence of microoperations required to Fetch and Execute AVR's `LD R3, Y+` instruction. *Your solutions should result in exactly 6 cycles for the fetch cycle and 7 cycles for the execute cycle.* You may assume only the AC and PC registers have the capability to increment/decrement itself. Assume MDR is 8-bit wide (which implies that memory is organized into consecutive addressable bytes), and AC, PC, IR, and MAR are 16-bit wide. Also, assume Internal Data Bus is 16-bit wide and thus can handle 8-bit or 16-bit (as well as portion of 8-bit or 16-bit) transfers in one microoperation.

load value from memory pointed to by Y into R3, post-inc Y.



★ assumption: the instruction is 2 bytes

★ assumption: PC is currently pointing to the current command.

★ assumption: All 16 bits need to go to IR, since it is 16 bits wide. Therefore, the registers used are "hard-coded".

Fetch:

(1) MAR ← PC ; fetch first byte

(2) MDR ← M(MAR), PC ← PC+1 ; read first byte, increment PC

(3) IR(15:8) ← MDR ; send first byte to high (IR)

(4) MAR ← PC ; fetch second byte

(5) MDR ← M(MAR), PC ← PC+1 ; read second byte, increment PC

(6) IR(7:0) ← MDR ; send second byte into low (IR)

★ Now, IR has the whole instruction.

Execute:

(1) MAR(15:8) ← R29, AC(15:8) ← R29

(2) MAR(7:0) ← R28, AC(7:0) ← R28

(3) MDR ← M(MAR), AC ← AC+1

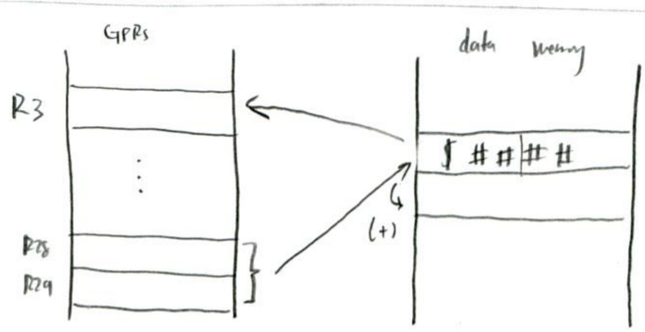
(4) R29 ← AC(15:8)

(5) R28 ← AC(7:0)

(6) R3 ← MDR

(7)

⇓



; (1)/(2): put address stored in Y into MAR and AC

; (3) load value in address into MDR, increment AC

; (4)/(5) store the incremented address back into Y

25/25

Problem #3 [25 pts]
Consider the following AVR assembly code for Lab. #5 that performs 16-bit by 16-bit multiplication (with some information missing). Assume the data memory locations \$0100 through \$0107 initially have the following values:

Data Memory Address	content	
0100	0C	* AddrA (x)
0101	00	
0102	0F	* AddrB (y)
0103	02	
0104	00	LAddrP
0105	00	
0106	00	
0107	00	

```
    .include "m128def.inc"              ; Include definition file
    .def   rlo = r0                     ; Low byte of MUL result
    .def   rhi = r1                     ; High byte of MUL result
    .def   zero = r2                    ; Zero register
    .def   A = r3                       ; An operand
    .def   B = r4                       ; Another operand
    .def   oloop = r17                  ; Outer Loop Counter
    .def   iloop = r18                  ; Inner Loop Counter
    .org   $0000
                rjmp   INIT
                .org   $0046
INIT:           clr    zero             ; Set zero register to zero
MAIN:           ldi    YL, low(addrB)   ; Load low byte
                ldi    YH, high(addrB)  ; Load high byte   } operand 1
                ldi    ZL, low(LAddrP)  ; Load low byte
                ldi    ZH, high(LAddrP) ; Load high byte
                ldi    oloop, 2         ; Load counter
MUL16_OLOOP:    ldi    XL, low(addrA)   ; Load low byte
                ldi    XH, high(addrA)  ; Load high byte   } operand 2
                ldi    iloop, 2         ; Load counter
MUL16_ILOOP:    ld     A, X+            ; Get byte of A operand
                ld     B, Y             ; Get byte of B operand
                mul    A,B              ; Multiply A and B
                ld     A, Z+            ; Get a result byte from memory
                ld     B, Z+            ; Get the next result byte from memory
                add    rlo, A           ; rlo <= rlo + A
                adc    rhi, B           ; rhi <= rhi + B + carry
                ld     A, Z             ; Get a third byte from the result
                adc    A, zero          ; Add carry to A
                st     Z, A             ; Store third byte to memory
                st     -Z, rhi          ; Store second byte to memory
                st     -Z, rlo          ; Store first byte to memory
                adiw   ZH:ZL, 1         ; Z <= Z + 1
                dec    iloop            ; Decrement counter
                brne   MUL16_ILOOP      ; Loop if iLoop !=
                sbiw   ZH:ZL, 1         ; Z <= Z - 1
                adiw   YH:YL, 1         ; Y <= Y + 1
                dec    oloop            ; Decrement counter
                brne   MUL16_OLOOP      ; Loop if oLoop != 0
Done:           rjmp   Done
    .org   $0100
    addrA:      .byte  2
    addrB:      .byte  2
    LAddrP:     .byte  4
```

1) 00

000C
× 020F

25

X → 00
A = 00
B = 0F
mul = 00 00
rlo 00
rhi 00

A 00 00 00
B 00
rlo =
rhi =

A = 0C
× B = 0F
$A4

loop 2 st+
00 A4
00 00
00
00

rlo rhi
r0 r1
A4 0
0 0
A+Z, R+Z

00 0C
Y = 02
X = 0C, 00
A = 0C
B = 02
A = A4
B = 00
rhi rlo
00 A4 00

25/25

Problem #3 (cont.)

[5 pts]

(a) What are the two 16-bit values (in hexadecimal) being multiplied? ~~$000C~~ and $020F

[5 pts]

(b) What are the contents of memory locations pointed to by `LAddrP`, `LAddrP+1`, `LAddrP+2`, and `LAddrP+3` after the loop `MUL16_ILOOP` (lines 11-25) completes for the first time?

[5 pts]

(c) What are the contents of memory locations pointed to by `LAddrP`, `LAddrP+1`, `LAddrP+2`, and `LAddrP+3` after the loop `MUL16_ILOOP` (lines 11-25) completes for the second time?

[5 pts]

(d) What are the contents of memory locations pointed to by `LAddrP`, `LAddrP+1`, `LAddrP+2`, and `LAddrP+3` after the loop `MUL16_ILOOP` (lines 11-25) completes for the third time?

[5 pts]

(e) What are the contents of memory locations pointed to by `LAddrP`, `LAddrP+1`, `LAddrP+2`, and `LAddrP+3` after the loop `MUL16_ILOOP` (lines 11-25) completes for the fourth time?

a) The two operands are $000C and $020F.

b)

LaddrP	B4
LaddrP+1	00
+2	00
+3	00

c)

+0	B4
+1	00
+2	00
+3	00

Same because X(high) = 00

d)

+0	B4
+1	18
+2	00
+3	00

e)

+0	B4
+1	18
+2	00
+3	00

Same because X(high) = 00

Problem #4 [25 pts]
Consider the AVR assembly code shown below with its equivalent (partially completed) binaries on the right for Problem #3. Determine the values for the following:

(a) `rd dddd rrrr` (@ address $0046)
(b) `KKKK dddd KKKK` (@ address $0049)
(c) `KKKK dddd KKKK` (@ address $004B)
(d) `d dddd` (@ address $004F)
(e) `rd dddd rrrr` (@ address $0055)
(f) `KKdd     KKKK` (@ address $005B)
(g) `kk kkkk k` (@ address $005D)
(h) `kkkk kkkk kkkk` (@ address $0062)

			Address	Binary			
	.org	$0000					
	rjmp	INIT	0000:	1100	kkkk	kkkk	kkkk
	.org	$0046	...		...		
INIT:	a clr	r2	0046:	0010	01rd	dddd	rrrr
MAIN:	ldi	YL, low(addrB)	0047:	1110	KKKK	dddd	KKKK
	ldi	YH, high(addrB)	0048:	1110	KKKK	dddd	KKKK
	b ldi	ZL, low(LAddrP)	0049:	1110	KKKK	dddd	KKKK
	ldi	ZH, high(LAddrP)	004A:	1110	KKKK	dddd	KKKK
	c ldi	r17, 2	004B:	1110	KKKK	dddd	KKKK
MUL16_OLOOP:	ldi	XL, low(addrA)	004C:	1110	KKKK	dddd	KKKK
	ldi	XH, high(addrA)	004D:	1110	KKKK	dddd	KKKK
	ldi	r18, 2	004E:	1110	KKKK	dddd	KKKK
MUL16_ILOOP:	d ld	r3, X+	★004F:	1001	000d	dddd	1101
	ld	r4, Y	0050:	1000	000d	dddd	1000
	mul	r3,r4	0051:	1001	11rd	dddd	rrrr
	ld	r3, Z+	0052:	1001	000d	dddd	0001
	ld	r4, Z+	0053:	1001	000d	dddd	0001
	add	r0, r3	0054:	0000	11rd	dddd	rrrr
	e adc	r1, r4	0055:	0001	11rd	dddd	rrrr
	ld	r3, Z	0056:	1000	000d	dddd	0000
	adc	r3, r2	0057:	0001	11rd	dddd	rrrr
	st	Z, r3	0058:	1000	001d	dddd	0000
	st	-Z, r1	0059:	1001	001d	dddd	0010
	st	-Z, r0	005A:	1001	001d	dddd	0010
	f adiw	ZH:ZL, 1	005B:	1001	0110	KKdd	KKKK
	dec	r18	005C:	1001	010d	dddd	1010
	~~f~~ brne	MUL16_ILOOP	005D:	1111	01kk	kkkk	k001
	g sbiw	ZH:ZL, 1	005E:	1001	0111	KKdd	KKKK
	adiw	YH:YL, 1	005F:	1001	0111	KKdd	KKKK
	dec	r17	0060:	1001	010d	dddd	1010
	brne	MUL16_OLOOP	0061:	1111	01kk	kkkk	k001
Done:	~~g~~ rjmp	Done	0062:	1100	kkkk	kkkk	kkkk
	h .org	$0100					
addrA:	.byte	2					
addrB:	.byte	2					
LAddrP:	.byte	4					

a) ✓ 0010 01 rd dddd rrrr (because clr Rd
↳ 0010 01 00 0010 0010 = Rd ← Rd ⊕ Rd)

b) −1 1110 KKKK dddd KKKK ; low(LAddrP) = $04
↳ 1110 0000 1110 0100 ZH = 31 = 11111
(leading 1 assumed)

c) ✓ 1110 KKKK dddd KKKK $02 loaded in
↳ 1110 0000 0001 0010 17 = 10001

d) 1001 000d dddd 1101 R3 = 00011
↳ ✓ 1001 000 0 0011 1101

e) ✓ 0001 11 rd dddd rrrr r1 = 00001
↳ 0001 11 00 0001 0100 r4 = 00100

−1
~~f)~~ g) 1111 01kk kkkk k001 q = −15
↳ 1111 01 1 1 1000 1 001 = 15 = 01111

h ~~g)~~ 1100 kkkk kkkk kkkk q = −1
↳ 1100 ~~0~~1 111 1111 1111
↳ two's comp encoding.

f) −3

ECE 375 Midterm 1 Solutions (only for stuff I got wrong).

- from #1: mis-converted 227 from dec to hex.

- from #4:

b) binary encoding of ldi ZL, low(LAddrP) → 30 = 1 1110 (actual)

1110 KKKK dddd KKKK

= 1110 0000 1110 0100 (mixed, thought it said ZH)

f) adiw ZH:ZL, 1

1001 0110 KKdd KKKK

1001 0110 00 11 0001

Z register, Y, X, and 25:24 are all valid.

(Z = 11, Y = 10, X = 01, 25:24 = 00).

g) brne MUL16_ILoop (back up to $004F)

1111 01kk kkkk k001

1111 01 11 1000 1 001

$004F - (\$0050 + 1)$ (PC+1)

= - $000F = - 0000 0000 0000 1111

↳ two's comp = 0b 1111 1111 1111 0001

h) rjmp Done (addr 0062)

1100 kkkk kkkk kkkk

1100 1111 1111 1111

$0062 - ($0062 + 1) = -1 = - 0000 0000 0000 0001

↳ two's comp = 1111 1111 1111 1111

19/20

ECE 375 Winter 2018 Quiz #2 Name:

Consider the following AVR code segment:

```
.include "m128def.inc"
.def mpr = r16
.def zero = r17
.equ BASEADDR = $01F0

START:
.org $0000
     RJMP INIT
.org $000A
     RCALL ISR
     RETI
INIT:
     CLR  zero
     ...                    ; Set up stack pointer, I/O, and Interrupts

ISR:
     ldi  r27,  HIGH(BASEADDR)     (1)} (1) - load BASEADDR into X, one byte at a time.
     ldi  r26,  LOW (BASEADDR)     (2)}
     in   mpr,  PIND               (3)} (2) - read input from Port D
     add  r26,  mpr                (4)} (3) - add input to X (address arithmetic)
     adc  r27,  zero               (5)}
     ld   mpr,  X                  (6)} (4) - load value in X
     out  PORTB, mpr               (7)} (5) - output value
     ret                           (8)} (6) - return from interrupt.
```

-1

31 } z
30 }

24 } y
25 }

27 } x
26 }

The eight lines of code for the sub-routine ISR shown above should function as follows:

1. Initialize X-pointer to BASEADDR ✓
2. Read the input from Port D.
3. Add the input to the X-pointer ✓
4. Load the value pointed to by X
5. Output that value to Port B ✓
6. Go back to whatever it was doing before the interrupt occurred ✓

Write the code for lines (1-8) so that the subroutine ISR works properly. Assume the subroutine INIT has already set up the stack, I/O, and interrupts.

18/20

ECE 375 Winter 2018 Quiz #1 Name:

Based on the initial register and data memory contents shown below (represented in hexadecimal), show how these contents are modified (in *hexadecimal*) after executing each of the following AVR assembly instructions. Do not be concerned about what happens to the Status Register (SREG) *after* the operation. *Instructions are unrelated*.

```
(a) ldi   r26, 36
(b) ldd   r4, Y+0x05
(c) lsl   r2
(d) mul   r1, r3
(e) sts   $0107, r29
```

Registers

R0	01	
R1	05	
R2	1B	
R3	07	
R4	01	
X	27: 01	06 :26
Y	29: 01	02 :28
SREG	FF	

Data Memory

0100	01
0101	BE
0102	35
0103	EC
0104	48
0105	2D
0106	04
0107	02

$24 ↑

(a) Register 26 (lower half of register X) is loaded with a constant ("immediate") decimal value of 36.
(∴ X now contains $0124)

(b) Register 04 is loaded with the contents of Y register's effective address offset by 5;
($0102) ($0107)

∴ register 4 is loaded with the value $02.

(c) The "logical shift left" operation is applied to register 02, where-in a 0 is shifted into the least significant bit, bumping up every other bit. The previous most significant bit is no longer contained w/in R02, it's placed into the "C" SR flag.

- Contents of R2 prior to shift: $05 : 0000 0101
- Contents of R2 after shift: 0000 1010 = ~~$0A~~ $36 −1

(d) The contents of R1 and R3 are multiplied, and placed/stored within ~~R1~~. −1

($05)($07) = (5 · 7) = 35 = $23

R0 = $23
R1 = $00

(e) The contents of R29 (upper half of Y register) are stored directly into SRAM address $0107. The address $0107 now contains the value $01.