COMP2121 Sample Exam Questions

1. Convert the following numbers from the original base to the specified base:

a) 123_{10} → $\_\_\_\_\_\_\_\_\_\_\_\__2$

b) 10101_2 → $\_\_\_\_\_\_\_\_\_\_\_\__{10}$

c) 1084_{10} → $\_\_\_\_\_\_\_\_\_\_\_\__{16}$

d) $A5_{16}$ → $\_\_\_\_\_\_\_\_\_\_\_\__{10}$

e) 11001001_2 → $\_\_\_\_\_\_\_\_\_\_\_\__{16}$

f) $2D5_{16}$ → $\_\_\_\_\_\_\_\_\_\_\_\__2$

2. What is the result of the following calculations?

a) 1395 + 4988 (base 16)

b) 11001001 + 00101101 (base 2)

c) A41 – 560 (base 16)

d) 11001 – 011 (base 2)

4. What number does 10010010 represent as an unsigned number? What does it represent in 2's complement notation?

5. In 2's complement addition, 11011011 + 01100000 = 00111011. Was there a 2's complement overflow? Why? What do the values in this sum represent?

6. What is the difference between performing 2's complement addition and unsigned addition in the AVR processor?

7. Represent the following numbers in IEEE 754 32-bit floating point notation:

a) 1.5

b) 1084

c) –1

d) –13.75

8. What does the following IEEE 754 FP number represent:

0	1000 0001	110 0000 0000 0000 0000 0000
Sign	Exponent	Mantissa

9. Encode the following instructions into Atmel AVR machine code:

```
a) ldi r18, 127
b) mov r18, r2
c) lds r2, 0xABCD
```

Refer to the AVR Instruction Set document on the course website (in the AVR Material section).

10. How many bits are needed to address:

a) 16 32-bit general purpose registers?
b) a memory space of 65536 bytes (assume byte addressing)?
c) a memory space of 65536 32-bit words (assume byte addressing)?

11. What do the following letters in a typical status register stand for and how are they generated?

a) Z
b) C
c) V
d) N
e) S

12. What is the main difference between the memory models of Princeton (von Neumann) and Harvard architectures?

13.

Memory address	Data
`0x00000100`	`0xAF`
`0x00000101`	`0x1B`
`0x00000102`	`0xC2`
`0x00000103`	`0x05`

Based on the above, what is the 32-bit word stored at the memory address `0x00000100` in a:

a) big-endian machine?
b) little-endian machine?

14. Can you design an 8-bit instruction format that can allow 4 2-operand instructions for a machine with 8 registers?

15. What do these notations mean in AVR assembly programming? Where are they used?

```
a) .def      d) .dseg   g) .dw

b) .set      e) .org    h) .byte

c) .cseg     f) .db     i) .equ
```

16. Where are the functions **low()** and **high()** utilised? Load -200 into a two byte number.

17. What are the differences between **Macros** and **Functions**? In what circumstances are each of them appropriate, and when should each be avoided? Write a Macro called **Invert** to invert the value of a register (Note: The register should be sent as a parameter)

18. What are ***word addressable*** and ***byte addressable***? Explain them with examples using AVR memories.

19. Consider the following AVR assembly code segment and fill the initialization part?

.dseg
array: .byte 20
.cseg
data: .dw 0x1234
*// Initialize the X pointer with **array***

____

*// Initialize the Z pointer with **data***

____

20. What are **little endian** and **big endian** representations ? Which endian is used in AVR?

21. Identify the errors in the following instructions,

```
a) ldi r1,18

b) cp r16,'L'

c) ldi zh, high(0x3476) => Word Addressable

d) ldi r40, 23

e) brge loop => for both unsigned numbers

f) brlo end => for both signed numbers
```

22. Write AVR assembly code segments for the following scenarios,

a) Initialize an array A of size 20 (each element is one byte) with values ranging from 1 to 20.
b) Initialize an array B of size 20 (each element is two bytes) with values ranging from -1 to -20.
c) Add the arrays A and B together and store the result into an array C.
d) Store the string 12345678 into program memory using .db and .dw.
e) Load the values stored in the program memory in (d) and store them into data memory in the reverse order.

23. How do you multiply a two byte number by a one byte number? (Explain using a simple example). Do we have to consider the carry bit in the STATUS register for this case?

24. Investigate the different ways of writing AVR assembly code for the following scenarios,

a) Copying a pair of registers into another pair of register.
b) Multiply a number by 4.
c) Divide a number by 4.

25. When are MUL, MULS and MULSU instructions used and how are they are used? Write AVR assembly code to perform multiplication for the following set of numbers,

a) 10, 12 (1 byte result) d) 32,258 (2 bytes result)
b) -11,11 (1 byte result) e) -352, 28 (2 bytes result)
c) -4,-14 (1 byte result) f) -27,-375 (2 byte result)

26. 1 Minimally modify the code below to add two numbers (in r17:r16 and r19:r18) when the result is bigger than 255.

```
ldi r16, 1
ldi r17, 0
ldi r18, 255
ldi r19, 0
add r16, r18
add r17, r19
```

26.2 Write AVR code to add two 32 bits values?(Using R16-R23 to hold all values.)

a = 0x00000100
b = 0x002000FF

27. Please complete the following table with instructions used for each operation.

Instructions	Registers	Stack	Memory		I/O	
Initialize			Data	Program	Separate	Mapped
Write to						
Read from						

28. How do you setup a port to act as an input port or as an output port in AVR? What instructions are used to read from an I/O port? What instructions are used to write to an I/O port?

29. Consider the following example AVR code segment:

```
Address
0x1000      .def grade=r20
0x1002      .include "m64def.inc"

0x1004      LDI r29,high(RAMEND)
0x1006      LDI r28,low(RAMEND)
0x1008      OUT SPH,r29
0x100A      OUT SPL,r28
0x100C      LDI r18,45
0x100E      RCALL GRADE_CAL
            end:
0x1010              RJMP end
            GRADE_CAL:
0x1012              PUSH r29
0x1014              PUSH r28
0x1016              CPI r18,50
0x1018              BRGE grade1
0x101A              LDI grade,2
0x101C              RJMP exit
            grade1:
0x101E              LDI grade,1
            exit:
0x1020              POP r28
0x1022              POP r29
0x1024              RET
```

What are the values of r28, r29, SPL and SPH:

a) after line "`LDI r28,low(RAMEND)`"?
b) after line "`OUT SPL,r28`"?
c) after line "`BRGE grade1`"?

d) after line "`POP r29`"?

30. The EICRA register is used to indicate what condition should be present for external interrupts to occur, and looks like this:

Bit	7	6	5	4	3	2	1	0	
	ISC31	ISC30	ISC21	ISC20	ISC11	ISC10	ISC01	ISC00	EICRA
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	

where each pair of bits ISCn1 and ISCn0 mean the following for INTn:

ISCn1	ISCn0	Description
0	0	The low level of INTn generates an interrupt request.
0	1	Reserved
1	0	The falling edge of INTn generates asynchronously an interrupt request.
1	1	The rising edge of INTn generates asynchronously an interrupt request.

The EIMSK register is used to enable the external interrupts and looks like this:

Bit	7	6	5	4	3	2	1	0	
	INT7	INT6	INT5	INT4	INT3	INT2	INT1	INT0	EIMSK
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	

In "m64def.inc", the values in these registers have been defined to their bit value. e.g., ISC00 = 0, ISC11=3 and INT2=2. Knowing this, examine the following code:

```
.def temp=r16
ldi temp, (0b10 << ISC00) | (0 << ISC10) |(0b11 << ISC20)
sts EICRA, temp
ldi temp, (1 << INT0) | (1 << INT2)
out EIMSK, temp
sei
```

a) What is the value (in binary) that is written to the EICRA register?
b) Why do we use this approach to set up the register values?
c) Which external interrupts can occur, and when will they occur?
d) What is the difference between the 'sts' instruction and the 'out' instruction?

31. This question looks at the registers associated with PORT A. The following tables might help:

DDxn	PORTxn	PUD (in SFIOR)	I/O	Pull-up	Comment
0	0	X	Input	No	Tri-state (Hi-Z)
0	1	0	Input	Yes	Pxn will source current if ext. pulled low.
0	1	1	Input	No	Tri-state (Hi-Z)
1	0	X	Output	No	Output Low (Sink)
1	1	X	Output	No	Output High (Source)

Port A Data Register – PORTA

Bit	7	6	5	4	3	2	1	0	
	PORTA7	PORTA6	PORTA5	PORTA4	PORTA3	PORTA2	PORTA1	PORTA0	PORTA
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	

Port A Data Direction Register – DDRA

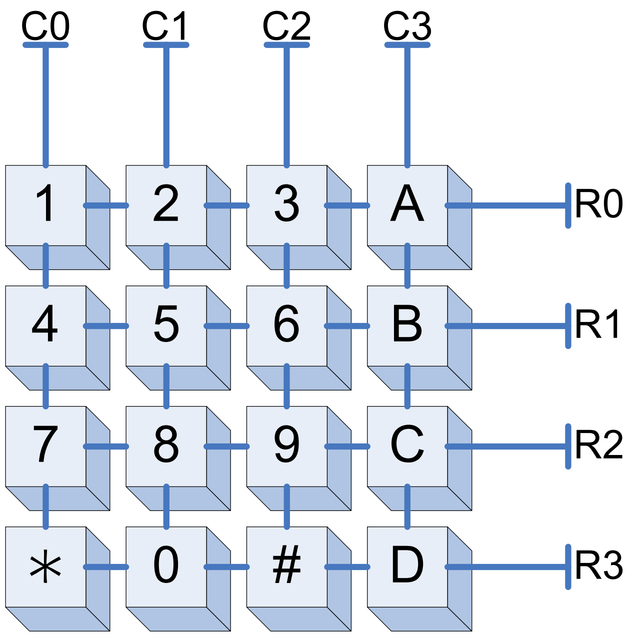
Bit	7	6	5	4	3	2	1	0	
	DDA7	DDA6	DDA5	DDA4	DDA3	DDA2	DDA1	DDA0	DDRA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Port A Input Pins Address – PINA

Bit	7	6	5	4	3	2	1	0	
	PINA7	PINA6	PINA5	PINA4	PINA3	PINA2	PINA1	PINA0	PINA
Read/Write	R	R	R	R	R	R	R	R	
Initial Value	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

a) What is the purpose of the DDRA register?
b) What is the purpose of PORTA0 when DDA0 = 1?
c) What is the purpose of PORTA0 when DDA0 = 0 and PUD = 0?
d) What is the purpose of the PINA register?

32. The Keypad on the AVR boards is a set of 16 push buttons. The keypad has four rows and four columns, accessible via the pins R0-R3 and C0-C3. When you push a button on the keypad, it connects the column of the key to the row of the key as follows:

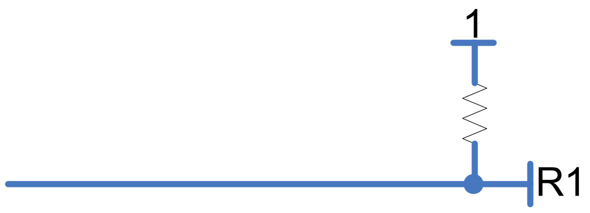


One method to correctly read what keys are being pressed is to:

1: Set up the rows so that they read a Logic 1 when none of the buttons on the row is pushed.
2: Set one column to Logic 0 and all other columns to Logic 1.
3: Read the values of the row pins. If a row reads as Logic 0, you know that the switch at that row and column must be pushed.
4: Set a different column to Logic 0 and read the rows.
5: Repeat steps 3 and 4 until a switch is found to be pressed or you run out of columns.
6: Repeat steps 2-5 again if you want to see whether a different switch is pushed.

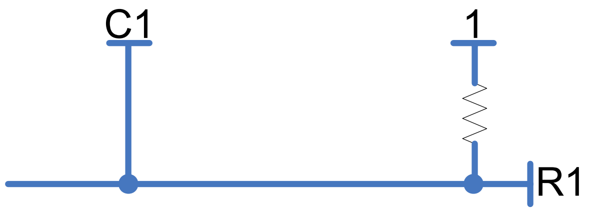
Part of your third lab requires you to perform this algorithm. Steps 2-5 should be fairly simple to code, but step 1 is not so obvious. The way to accomplish this is with pull-up resistors. A pull-up resistor ties an input pin to Logic 1 via a resistor. This means that an input pin will still read any value that is input, and will read Logic 1 if disconnected.

To further understand this, look at switch 5 in the above diagram. When none of the switches connected to row 1 are pushed, the circuit (with pull-up shown) looks like this:



If read, the port would read a Logic 1 via the pull-up resistor.

When switch 5 is pushed, the circuit looks like this:



In this case, the port connected to R1 will always read the current value of C1. When C1 is Logic 0 there will be a voltage drop across the resistor, but this will not affect the value being read. Thus, the pull-up resistor accomplishes the desired task.

a) How do you setup an AVR I/O port so that it has pull-up resistors connected to its input pins? (See question 2 of this tute)

b) Write the code to find a switch that has been pushed by scanning either the columns or rows. (You have to do this for your lab, anyway)

c) Can you see an electrical problem with this scanning method when two switches on the same row are pushed at the same time (e.g., 5 and 6)? How could you correct this? (Hint: There might be something better you can do than output logic 1s to the columns you are not testing.)