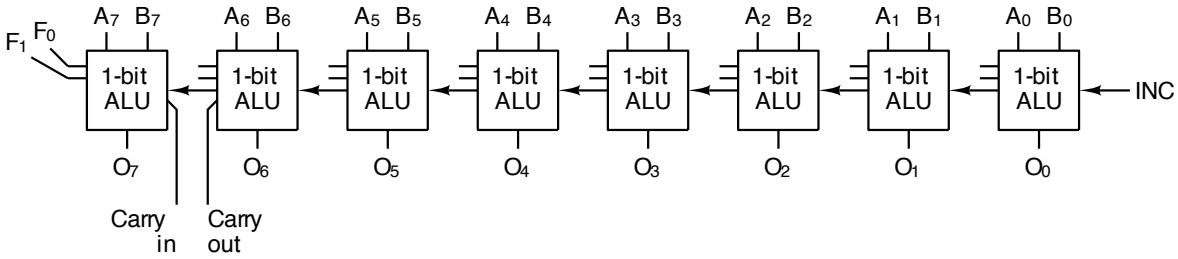
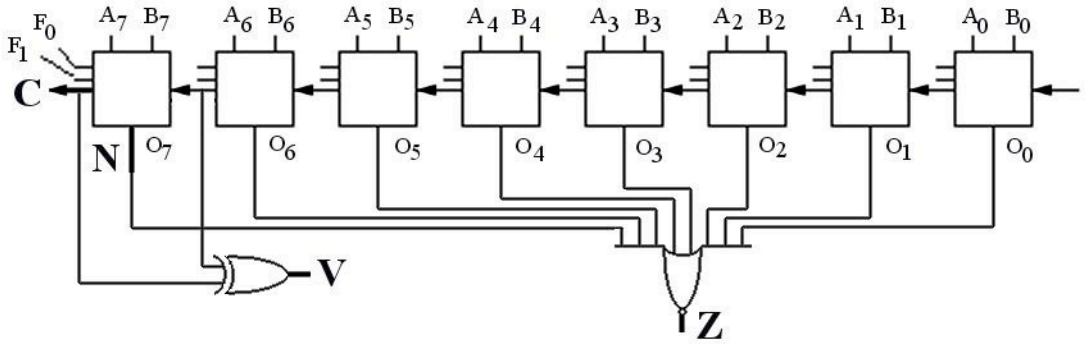
# CSSE2010 / CSSE7201 – Introduction to Computer Systems
# Answers to Exercises – Week Six
# Control Unit, AVR Introduction

## *Answers*

Some of the questions below are taken from or based on questions in Tanenbaum, Structured Computer Organisation, 5th edition.

1. Consider the 8-bit ALU below (from Tanenbaum, also shown in lectures) where each 1-bit ALU corresponds to the bit-slice shown in lectures. The ALU has six control inputs (F1,F0,ENA, ENB, INVA, INC (carry-in)) which (except for the carry-in) are common to all the bit slices. Show the logic circuitry that can be used to generate status register bits Z, N, C, V (as defined in the lecture).




2. After each of the following operations, what would be the values of the Z (zero), V (overflow), C (carry) and N (negative) bits? (Numbers given are decimal.)
   (a) 7 + 9 (in a 5 bit ALU)
   *(00111 + 01001 = 10000)*
   *Z: 0 V: 1 C: 0 N: 1*

   (b) 7 + 9 (in an 8 bit ALU)
   *(00000111 + 00001001 =00010000)*
   *Z: 0 V: 0 C: 0 N: 0*

   (c) -16 – 16 (in a 5 bit ALU)
   *(10000 + 10000 = 00000)*
   *Z: 1 V: 1 C: 1 N: 0*

   (d) -16 – 16 (in an 8 bit ALU)
   *(11110000 + 11110000 = 11100000)*
   *Z: 0 V: 0 C: 1 N: 1*

3. How can we perform 16-bit wide arithmetic operations with an ALU that supports only 8-bit wide operations? Specifically, if two 8-bit registers of a CPU contain two halves of a 16-bit number and another two registers contain two halves of another 16-bit number, what sequence of operations do we need to undertake in order to
   (a) Add the two numbers together
   (b) Negate the first number
   (Think of this question in terms of any 8-bit CPU, not specifically the Atmel AVR. The question is essentially about how to build 16-bit operations out of 8-bit operations, and importantly, what to do with carry bits.)

   *(a) First, we need to add the least significant 8-bits, with a carry-in of 0. The carry-out of this addition will be placed in the C-bit (carry flag) in the status register. (This is the carry-out from the most significant bit). Next, we add the most significant 8-bits, with the carry-in set to the value of the C-bit. The carry out of this latter add will be saved in the status register but is irrelevant in terms of the answer generated.*

   *[ Some CPUs only have an ADD instruction which always adds the C-bit (carry flag) into the sum. In this case, one would have to set the C-bit to 0 before the first addition.]*

   *(b) Two's complement negation is the same as inversion (flipping all the bits) and adding one. To do this to a 16-bit quantity using 8-bit operations, we can*
   *(i) invert each 8-bit half*
   *(ii) add one to the least-significant 8 bits and store the carry-out into the C-bit*
   *(iii) add the C-bit to the most-significant 8 bits*

   *Some CPUs will have a negate instruction but we can NOT use 8-bit negation on each half since this does not correctly handle what happens at the boundary between the two bytes.*

   *[ It may be possible to use a negation instruction on the least significant byte (in place of the inversion of the least significant bit and adding one) provided that this correctly sets the C-bit in the status register. We would still need to invert the most significant byte and add in the carry from the operation on the least significant byte.]*

4. Write down Atmel AVR instructions (in assembly language and machine code form) which will perform the following operations (not all of these instructions are covered in lecture – use the AVR instruction reference to attempt to locate these operations and also see the machine code equivalent; some of the concepts will be covered in future weeks):
   (a) increment register 5

```
inc r5
1001 0100 0101 0011
```
   *(1001 010d dddd 0011 where dddd = 00101)*

   (b) load the value 24 into register 17

```
ldi r17, 24
```
   *or*

```
ldi r17, $18
```
   *(using hexadecimal notation)*

```
1110 0001 0001 1000
```
   *(1110 KKKK dddd KKKK where KKKKKKKK = 00011000 = $24_{10}$ and dddd = 0001 = $1_{10}$ , plus an implied 16)*

   (c) logically AND registers 23 and 5 and put the result in register 5

```
and r5, r23
0010 0010 0101 0111
```
   *(0010 00rd dddd rrrr where rrrrr=10111 = $23_{10}$ and ddddd= 00101 = $5_{10}$)*

   (d) increment register 7

```
inc r7
1001 0100 0111 0011
```
   *(1001 010d dddd 0011 where dddd = 00111)*

(e) load the value 35 into register 18

```
ldi r18, 35
1110 0010 0010 0011
```
*(1110 KKKK dddd KKKK where KKKKKKKK = 00100011 = $35_{10}$ and dddd = 0010 = $2_{10}$ , plus an implied 16)*

(f) set all the bits of register 18 to 1

```
ldi r18, 255 or ser r18
1110 1111 0010 1111
```
*(same machine code for both assembly language instructions – the ser instruction also only works on registers 16 to 31)*

(g) set all the bits of register 18 to 0

```
ldi r18,0 or clr r18
1110 0000 0010 0000
0010 0111 0010 0010
```
*(for ldi r18,0)*

*(for clr r18: 0010 01rd dddd rrrr where rrrrr = ddddd = 10010. This is actually the same instruction as eor r18,r18 – i.e. exclusive or of a register with itself and put the result back in the register)*

(h) negate register 10

```
neg r10
1001 0100 1010 0001
```
*(1001 0100d dddd 0001 where ddddd = 01010)*

(i) invert register 10

```
com r10
1001 0100 1010 0000
```
*(1001 010d dddd 0000 where ddddd = 01010)*

(j) move the contents of register 3 to register 4

```
mov r4, r3
0010 1100 0100 0011
```
*(0010 11rd dddd rrrr where ddddd = 00100 and rrrrr = 00011)*