COMP9032 Lab 2

August, 2017

1. Objectives

In this lab, you will learn AVR programming on

- memory access
- stack and function

2. Tasks

2.1 Task 1: Positional Division (For Week 4, due Week 5)

Hand-division uses a series of left shifts, magnitude checks and multiple subtractions to get the final answer. For example, the **decimal** division 3217/16 can be calculated as:

1. Shift the divisor 16 to the left as many times as possible, until just before the resulting value becomes greater than the dividend 3217. This means it is left-shifted by two (decimal) digits; the shifted divisor is 1600.
2. Subtract multiples of this shifted divisor (2 × 1600 = 3200) from the dividend, leaving 17 as the partial remainder and 200 the partial quotient.
3. In the second iteration, shift the new divisor 1600 right by one digit to become 160. This is greater than the partial remainder 17, so do not subtract anything.
4. In the third iteration, shift the new divisor 160 right by one digit to become 16.
5. Subtract a multiple of this shifted divisor (1 × 16 = 16) from the new dividend (the previous partial remainder) 17, leaving 1 as the new partial remainder. Add the multiple 1 to the previous partial quotient of 200, giving 201.
6. Finally, stop the iteration here, as no more right shifts are possible. The old partial quotient of 201 becomes the actual quotient (result); the old partial remainder becomes the actual remainder.

This hand-division approach can be applied for the binary division. The program in Figure 3 is an implementation of this positional division algorithm for the 16-bit **binary division**. Manually translate the C program into assembly program. Assume the dividend and the divisor are stored in the program memory and that the quotient is saved in the data memory.

```
int posdiv(unsigned int dividend, unsigned int divisor) {
        unsigned int quotient;
        unsigned int bit_position = 1;
        quotient = 0;
        while ((dividend > divisor) && !(divisor & 0x8000)) {
                divisor = divisor << 1;
                bit_position = bit_position << 1;
        }
        while (bit_position > 0) {
                if (dividend >= divisor) {
                        dividend = dividend - divisor;
                        quotient = quotient + bit_position;
                }
                divisor = divisor >> 1;
                bit_position = bit_position >> 1;
        }
        return quotient;
}
```

Figure 1: binary_positional_division.c

2.2 Task 2: String to Integer Conversion (For Week 5, due Week 6)

The C program in Figure 4 **implements the function** of converting a string to an integer. The string is given in main() and its integer is obtained by calling function atoi(). Manually translate the program into an assembly program **with *atoi* implemented as a function.** Assume the string is stored in the program memory and that an integer takes two bytes.

```
int main(void) {
        char s[ ] = "12345";
        int number;
        number = atoi(s);
        return 0;
}

int atoi(char *a) {
        char i;
        char c;
        int n;

        n = 0;
        c = *a;
        for (i=1; ((c >='0') && (c<='9') && (n<65536)); i++){
                n = 10 * n + (c -'0');
                c = *(a+i);
        }
        return n;
}
```

Figure 2 string_to_number.c

Note: Each task is worth 6 marks. All your programs should be well commented and easy to read. Up to 1 mark will be deducted for each program without proper and sufficient comments.