1. Objectives

In this lab, you will learn AVR programming on
- Memory access; and
- Stack, stack frame design and functions

2. Preparation

Before coming to the laboratory, you should:
- obtain one USB memory on which to store your work;
- read through this experiment in detail, trying to understand what you will be doing; and
- write your programs at home in order to finish the experiment on time.

3. Tasks

3.1 Task 1: Matrix Multiplication

In C, a two dimensional matrix is stored in a row-major order, that is, row i+1 is stored after row i. For example, a 3x3 array A is stored at 0xF0 in the memory as shown in Figure 1, where each element of A is one byte long.

![Matrix Multiplication Diagram]

Figure 1: The layout of A[3][3] in memory
Figure 2 gives a C implementation of matrix multiplication. Manually translate the C program into assembly program. Assume B is stored in the program memory; A and C are stored in the data memory. All values are one byte long. The address of element A[i][j] can be calculated by the formula:

\[ a + i \times \text{row \_ size} + j, \]

where a is the address of the matrix and row\_size is the row size in bytes.

```c
unsigned char main(void) {
    unsigned char i, j, k;
    unsigned char A[4][4], C[4][4];
    const unsigned char B[4][4]={{1, 2, 3, 4}, {3, 4, 5, 6}, {1, 2, 1, 2}, {3, 4, 4, 3}};
    for (i=0; i<4; i++) {
        for (j=0; j<4; j++) {
            A[i][j] = i+j;
            C[i][j]=0;
        }
    }
    for (i=0; i<4; i++) {
        for (j=0; j<4; j++) {
            for (k=0; k<4; k++) {
                C[i][j] += A[i][k]*B[k][j];
            }
        }
    }
    return 0;
}
```

Figure 2: matrix_multiplication.c

Check Point A: Signature:

3.2 Task 2: Positional Division

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 it becomes greater than the dividend 3217. This means it is left-shifted by two digits; the shifted divisor is 1600.
2. Subtract multiples of this shifted divisor \((2 \times 1600 = 3200)\) from the dividend, leaving 17 as the partial quotient of 200.
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 \((1 \times 16 = 16)\) 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 binary division. The program in Figure 3 is an implementation of this positional division algorithm for 16-bit \textbf{binary division}.

Manually translate the C program into assembly program. Assume that the register pair r16:r15 store the dividend, r18:r17 store the divisor and r20:r19 store the quotient.

```c
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 3 positional_division.c
3.3 Task 3: String to Integer Conversion

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. Assume the string is stored in program memory and an integer takes two bytes.

```c
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 4  string_to_number.c

Check Point C:  Signature:

Note: Each task is worth 5 marks. All your programs should be well commented. Up to 1 mark will be deducted for each program without proper and sufficient comments.

Due Time: your lab class in week 5.