Lab 1 (Week 2- Week 3)

1. Objectives:

- Learn to write AVR assembly programs using AVR assembly language instructions (arithmetic and logic instructions, program control instructions and data transfer instructions).
- Understand how operations on multi-byte integers are implemented by software.

2. Programming Style

When you write an assembly program, make sure that your program is easy to read. You are encouraged to adopt the following rules, especially for this course:

- Start each source code file with a heading.
- Include your name and e-mail address.
- Write the date of last modification and a version number.
- Include appropriate comments that explain what each line of assembly code does. In AVR assembly language, comments start with ";".
- Use a sensible layout for your code. This makes it easy to see the assembly statements, and any labels in your code easy to see.

3. TASKS

There are four tasks. Each task is worth 5 marks.

3.1 TASK A: Computing the Greatest Common Divisor (5 marks)

The C program shown in Figure 1 computes the greatest common divisor of two unsigned integers a and b. Assume that both a and b are two-bytes long. Manually convert the C code into AVR assembly code. You may assume that the variables a and b are already stored in some general registers.

```
void main()
  {
    unsigned int a, b; /* Initialized elsewhere */
    while(a!=b)
      {
       if(a>b)
          a=a-b;
       else
          b=b-a;
      }
     return 0;
   }
```

Figure 1: The C program for computing GCD

3.2 Task B: String to Integer Conversion (5 marks)

The program atoi.c in Figure 2 converts a string into a number. Write an AVR assembly program with the same function as atoi.c. Your program must satisfy the following requirements:

a. The string "325658" is stored in the program memory.
b. The result is stored in the data memory.

```
int main()
   {
      char s[]="325658";
      char i;
      unsigned int n;
      n=0;
      for(i=0;i<=5;i++)
         n=10*n+(s[i]-'0');
      return 0;
   }
```

Figure 2: Program atoi.c

Note that each character in C is stored in its ASCII code. The ASCII table is available here http://www.cse.unsw.edu.au/~cs2121/AVR/ascii.bmp. In the above program, '0' denotes the ASCII code of 0, and (s[i]-'0') gives the value of the digit s[i]. The sizes of char and unsigned int are one byte and four bytes, respectively.

For this task, you need to figure out how many bytes are required to store the resulting integer of the string "325658". Notice that the range of unsigned integers represented using n bits is between 0 and 2^n-1.

In order to implement n*10, you can use either shift and rotation instructions, or multiplication instructions. The former is easier to understand and the later requires a method for multiplying a one-byte integer with a multi-byte integer.

3.3 TASK C: Computing the sum of an array (5 marks)

The C program in Figure 3 computes the sum of all elements of an array of unsigned integers. Assume that each element of the array A and sum are two-bytes long. Write an AVR assembly program with the same function as the C program. Assume that the array is stored in the data (SRAM) memory.

```
void main()
   {
    unsigned int A[10],sum,i;
    for(i=0;i<10;i++)
       A[i]=200*i;
    sum=0;
    for(i=0;i<10;i++)
       sum=sum+A[i];
   return 0;
   }
```

Figure 3: The C program for computing the sum of an array.

3.4 Task D: Matrix Multiplication (5 marks)

A two-dimensional array in C is stored in row-majored order. In row-major order, a two-dimensional array is stored as an array of rows and the row i+1 is stored after the row i. For example, a 3*3 array A is stored at 0x1F0 in the memory as shown in Figure 4, where each element of A is one byte long.
Assume that the size of each element in an m*n array A is c bytes. The address X of A[i][j] is computed as follows:

$$X = Y + (i*n+j)c$$

where Y is the starting address of A. **However, you do not need to compute the address of A[i][j] using this formula. Instead, you need to observe how the pointers for the arrays A, B and C change when they are accessed and update the pointers dynamically using addition instructions or subtraction instructions only.**

The program *matrix-product.c* in Figure 5 computes the product of two 5*5 matrices A and B and stores the result in the array C. Write an AVR assembly program to implement the C program. All the variables in the C program must be stored in the data memory.

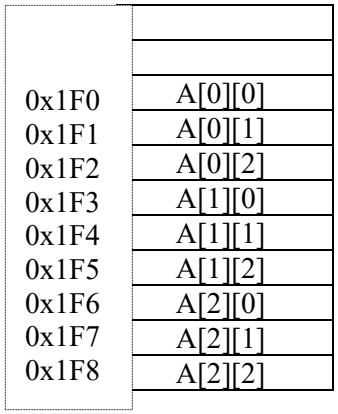

Figure 4: The layout of A[3][3] in the data memory

```
char i, j, k;
signed char A[5][5], B[5][5];
short C[5][5];
int main()
   {
     for (i=0; i<5; i++)
         for (j=0; j<5; j++)
            {
              A[i][j]=i+j;
              B[i][j]=i-j;
              C[i][j]=0;
            }
     for(i=0;i<5;i++)
        for(j=0;j<5;j++)
            for(k=0;k<5;k++)
               C[i][j]+= A[i][k]*B[k][j];
     return 0;
   }
```

Figure 5: matrix-product.c

Assume that the size of (signed) char is one byte, and the size of short is two bytes.

4. Deadline

The deadline of this lab is Week 3, i.e., you must finish all the tasks before the end of your lab class in Week 3.