1. Objectives

The objectives of this experiment are to
• Implement functions using AVR assembly language.
• Use a stack to store dynamic data such as parameters and local variables.
• Use AVR macros.
• Use nite to download a program to the AVR development board and run it.

2. Tasks

There are three tasks in this experiment. The first two tasks are about implementing functions in AVR assembly language and the third task is about using AVR development board to run a program.

2.1 Task A: Evaluating A Polynomial

The C program polynomial.c shown in Figure 1 evaluates the polynomial function $F(x)=a_n \cdot x^n + a_{n-1} \cdot x^{n-1} + \ldots + a_1 \cdot x + a_0$. Assume that $a_i$ (i=$0$, 1, ..., n), x and n are ONE BYTE long and the result $F(x)$ is TWO BYTE long. Write an AVR program to implement the C program. Your program must satisfy the following requirements:

- All constants are stored in the program (FLASH) memory.
- All global variables are stored in SRAM.
- All local variables and parameters are stored in the stack.
- The return value of each function is stored in the register pair r25:r24.

Checkpoint A: | Signature:
3.2 Task B: Hanoi Tower Problem

The following is an excerpt from the book Concrete Mathematics - A Foundation for Computer Science (2nd Edition) by Graham, Knuth and Patashnik: A neat little puzzle called the Tower of Hanoi, invented by the French mathematician Edouard Lucas in 1883. We are given a tower of eight unequally sized disks, initially stacked in decreasing size on one of three pegs. The objective is to transfer the entire tower to one of the other pegs, moving only one disk at a time and never moving a larger one onto a smaller. Lucas furnished his toy with a romantic legend about a much larger Tower of Brahma, which supposedly has 64 disks of pure gold resting on three diamond needles. At the beginning of time, he said, God placed these golden disks on the first needle and ordained that a group of priests should transfer them to the third, according to the rules above. At the end of the third move, the Tower will crumble and the world will end.

Now the question arises: How many moves are necessary and sufficient to perform the task?
The C program shown in Figure 2 solves this problem, where n is the number of disks and A, B and C denote the three pegs. After the program is completed, the value of counter is the total number of moves.

```c
#include <stdio.h>
int counter = 0;

void move(int n, int A, int C, int B)
{
    if (n==1)
        counter++;
    else
    {
        move(n-1,A,B,C);
        move(1,A,C,B);
        move(n-1,B,C,A);
    }
}

int main()
{
    int n=8, A=1, B=3, C=2;
    move(n,A,B,C);
    return 0;
}
```

Figure 2: Program tower-Hanoi.c

Assume that the number of disks is at most 15. Write an AVR assembly language program using the above C code. Your program must satisfy the following requirements:

- All constants are stored in the program (FLASH) memory.
- All global variables are stored in SRAM.
- All local variables and parameters are stored in the stack.
- No overflow occurs during the execution of your program.

How many bytes of SRAM memory are needed for the stack to run this program correctly?

Checkpoint B:  
Signature:
2.3 Task C: AVR Development Board and Downloader

This task introduces the AVR Microcontroller Development Board and the downloader nite. In the subsequent labs you will use the development board to run your program instead of only simulating it using AVR Studio.

2.3.1 Board Handling Precautions

Two students in each group share one AVR development board due to the limited number of boards. You are required to keep the board properly.

A few precautions are necessary when you use the AVR development board:

- Electrostatic discharge can destroy electronic equipment without giving any sign of doing so! Please always discharge yourself before handling the AVR microcontroller board, and observe any handling precautions as advised by the demonstrator. You can do this painlessly by touching a grounded conductor using a coin, a key or another metallic object instead of your finger.
- Short circuits may damage certain devices. Please remove any metal watch straps, buckles, rings and so on before handling the board.
- Always turn the power off before connecting or disconnecting any I/O subsystems.

2.3.2 Examining the AVR Microcontroller Board

Figure 3 shows a picture of the assembled system. It also shows how cables attach between the microcontroller connectors and the I/O connectors. Figure 4 is a layout diagram of the assembled system:

As you can see from the figures, the AVR Microcontroller Board has one printed circuit board which contains most of the electronics, peripherals and connectors. This board is designed by David Johnson in the School of Computer Science and Engineering, University of New South Wales, Australia.

The particular microcontroller used on this board is Atmel ATmega64 device, one of the dozens available from different manufacturers world-wide.
Figure 3: AVR Development Board

Figure 4: AVR Development Board Block Diagram
Take a closer look at the particular microcontroller used on the AVR Microcontroller Board. Refer to page 2 of the Atmel ATmega64 Datasheet for the pin diagram for this microcontroller (You can find this datasheet in the AVRDOC folder on your desktop). Now identify the following components on the board, observe them on the system block diagram and try to trace the connections between them and other major components:

- AVR processor
- USB Interface
- Flash Memory
- SRAM
- Reset & Loader Switches
- Speaker
- I/O Pins
- Keyboard
- LED bargraph
- LCD Display
- Microphone
- Filter-Amplifier
- Motor, Shaft Encoder
- LDR, LED, POT, PB1, PB0

### 2.3.3 Create New File

Create a new file for this checkpoint and name it led.asm. led.asm is shown in Figure 5.

After creating the above AVR assembly program, you are now ready to advance to the next step, which simulates the code using AVR Studio.

### 2.3.4 Simulating the Code

At this point you have generated the file needed to simulate the code. Now Build and Run your program. To view the contents of I/O registers and ports, select View→Toolbars→I/O and double click on “I/O ATMEGA64” tree to expand it. This is used to inspect and modify the contents of the I/O registers. The standard configuration gives you a quick overview of the hardware with the ability to expand particular items for more information. Now expand PORTC tree in the list as before. You can see three I/O memory locations associated with Port C. They are: Port C Data Register (PORTC), Data Direction Register (DDRC) and Input Pins (PINC).

- **PORTC**: PORTC register is a read/write register. It is initialized to $00 at reset. If it is programmed as an output, writing to PORTC will allow you to change the logic state at the PORTC pins.
- **DDRC**: This register is used to control the direction of each of the pins of the PORTC. Writing a “0” (which is also the reset value) to any bit of this register will make the corresponding PORTC bit as input, and writing a “1” will make it an output bit.
• PINC: PINC is read-only. With PINC you can read the logic of each physical pin of PORTC. As shown in I/O View each bit in the registers is represented by a white square box. A logical zero (0) is represented by a white square box and a logical one (1) is represented by a black square box. These boxes will be updated dynamically during program execution, and show the current state of each bit. You may also set and clear these bits by clicking on the appropriate box at any time during the program execution.

```assembly
; led.asm
; Author:
; Date:
.include "m64def.inc"
def temp =r16
equ PATTERN1 = 0x5B
equ PATTERN2 = 0xAA
ser temp
out PORTC, temp ; Write ones to all the LEDs
out DDRC, temp ; PORTC is all outputs
out PORTA, temp ; Enable pull-up resistors on PORTA
clr temp
out DDRA, temp ; PORTA is all inputs
switch0:
sbic PINA, 0 ; Skip the next instruction if switch0 is pushed
rjmp switch1 ; If not pushed, check the other switch
ldi temp, PATTERN1 ; Store PATTERN1 to the LEDs if the switch was pushed
out PORTC, temp
switch1:
sbic PINA, 1 ; Skip the next instruction if switch 1 is pushed
rjmp switch0 ; If not pushed, check the other switch
ldi temp, PATTERN2 ; Store PATTERN2 to the LEDs if the switch was pushed
out PORTC, temp
rjmp switch0 ; Now check switch 1 again
```

Figure 5: Program led.asm

Now single step down to the last line of the program by repeatedly pressing the F11 key or by selecting “Step Into” from the “Debug” menu. Notice how the color changes from black to red when the value of a register changes.

### 3.7.5 Download Program

After having finished simulating the program, you need to download the program to the board and run it there by using nite. nite is a downloader which downloads your program onto the board. It was developed by Jayasooriah. The detailed descriptions of nite can be
found at http://www.cse.unsw.edu.au/~jayas/esdk/nite.html. Only two commands i.e. path and send will be used in the lab.

An "Intel Hex Format" file called led.hex has been generated after "Build and Run". Here are the steps you need to follow: (Note: Don't have the USB power cable plugged in before performing any wiring.)

1. Use the provided patch cables to connect pins PC0-PC7 to the LED0-LED7, and PB0-PB1 to the PA0-PA1. (Make sure you have the right order).
2. Plug the USB cable into the board and the PC
3. Open the downloader nite by double clicking the nite icon on the desktop.
4. Hold the Loader push-button down on the AVR Microcontroller Board
5. While holding the Loader push-button down push the Reset button and release the Reset button.
6. Release the loader button. The board is now waiting for a program to be downloaded.
7. When you see the logo in the nite window appears, then you can download the new version of the program.
8. Now change the directory by typing :path directory name, where directory name is the name of the directory which contains the hex file of your program.
9. To download the program, type :send file.hex (ie: :send led.hex). This downloads the program to the AVR board, and some output is also generated.
10. Press the Reset switch to execute the program.

Note that every time you download a new or modified program to the AVR Microcontroller Board, you have to RESET the board first by pressing the Loader button and the Reset button at the same time.

3.7.6 LED Bit Pattern

Your first task is to work out the bit patterns by examining the AVR assembly program given in figure 5.

Answer the following questions:

1. Does a 1 bit turn the LED on or does it turn it off? Why?
2. Which registers change their value? How are the values changed?
3. Which instruction turns on the LEDs?

Explain your answers to the laboratory assessor.

Checkpoint C: Signature: