ECE-GY 6843 Real Time Embedded Systems Exam 1

Yogya Sharma
(ys5250@nyu.edu)

1. This is the common reg1.S file that is used in all the parts of the question:

```
.syntax unified

.global GetReg

GetReg:

   cmp r0, #0
   beq Reg0
   cmp r0, #1
   beq Reg1
   cmp r0, #2
   beq Reg2
   cmp r0, #3
   beq Reg3
   cmp r0, #4
   beq Reg4
   cmp r0, #5
   beq Reg5
   cmp r0, #6
   beq Reg6
   cmp r0, #7
   beq Reg7
   cmp r0, #8
   beq Reg8
   cmp r0, #9
   beq Reg9
   cmp r0, #10
   beq Reg10
   cmp r0, #11
   beq Reg11
   cmp r0, #12
   beq Reg12
   cmp r0, #13
   beq Reg13
```

```
cmp r0, #14
beq Reg14
cmp r0, #15
beq Reg15

cmp r0, #15
bgt end

cmp r0, #0
blt end

Reg0: bx lr
Reg1: mov r0,r1
bx lr
Reg2: mov r0,r2
bx lr
Reg3: mov r0,r3
bx lr
Reg4: mov r0,r4
bx lr
Reg5: mov r0,r5
bx lr
Reg6: mov r0,r6
bx lr
Reg7: mov r0,r7
bx lr
Reg8: mov r0,r8
bx lr
Reg9: mov r0,r9
bx lr
Reg10: mov r0,r10
bx lr
Reg11: mov r0,r11
bx lr
Reg12: mov r0,r12
bx lr
Reg13: mov r0,r13
bx lr
Reg14: mov r0,r14
bx lr
Reg15: mov r0,r15
```

```
    bx lr

    end: mov r0, #-1234
    bx lr
```

Following are the C files for each required operation:

a) main_one_reg.c: Function to find value of 1 register.

```
#include <mbed.h>

extern "C" uint32_t GetReg(uint32_t a);

int main() {

    uint32_t integer;

    //Variable declaration to capture returned value from GetReg()
    uint32_t result_asm;

    //Calling GetReg() to get values of the register
    result_asm = getReg(integer);

    //Checking if invalid input was given to the function.
    if(result_asm == -1234)
    {
       printf("Invalid input value");
    }
    else
    {
       printf("Value of the register called for = %d", result_asm)
    }

}
```

b) main_all_reg.c: Updated function where we print the values of all registers r0 through r15.

```
#include <mbed.h>
#include <stdio.h>

extern "C" uint32_t GetReg(uint32_t a);
```

```
int main() {

   //Array to store valued of register 0-15.
   uint32_t reg_value_array[16];

   //Variable declaration to capture returned value from GetReg()
   uint32_t result_asm;

   //Calling GetReg() to get values of all 16 registers
   for(int i = 0; i<16; i = i+1)
   {
     result_asm = 0;
     result_asm = GetReg(i);
     reg_value_array[i] = result_asm;
   }

   //Printing the values of all the registers
   for(int j = 0; j<16; j= j+1)
   {
      printf(reg_value_array[j])
   }

}
```

c) Attached files: main_one_reg.c, main_all_reg.c, reg1.S

2. This is the reg2.S file for this question:

```
.syntax unified

.global GetReg

GetReg:

   str r0, [r0], #4
   str r1, [r0], #4
```

```
    str r2, [r0], #4
    str r3, [r0], #4
    str r4, [r0], #4
    str r5, [r0], #4
    str r6, [r0], #4
    str r7, [r0], #4
    str r8, [r0], #4
    str r9, [r0], #4
    str r10, [r0], #4
    str r11, [r0], #4
    str r12, [r0], #4
    str r13, [r0], #4
    str r14, [r0], #4
    str r15, [r0], #4

    MRS r0, CPSR
    bx lr
```

main_arr_reg.c:

```
#include <mbed.h>

extern "C" uint32_t GetReg(uint32_t* arr);

int main(){
    uint32_t ar[16];
    uint32_t result_asm = GetReg(ar);
    for(int i = 0; i<16; i= i+1)
    {
        printf("reg%d: %d", i, ar[i])
    }
    printf("CPSR Value %d:", result_asm)
}
```

3. The given function reads the byte values from str (‘123’,0 -> meaning that the string ‘123’ is null terminated). The ASCII values of each character are being stored in contiguous memory locations. This function converts them to their integer value one by one, while checking that that byte is a correct number by checking if it is greater than 0x30 (for '0') and less than/equal to 0x39 (for '9'). If any of these conditions are not met the code branches to STOP, which in turn exits the code. The initial content of register r2 can be assumed as 0. So, the program is multiplying the value of r2 with 10, adding it with the number obtained from the string and storing the result in register r2 in each iteration.

   Iteration 1:

   ‘1’ is loaded into r0 as 0x31
   r0 = r0 – 30 = 1
   r1: address of the label
   r3: r2 + r2*4 = 0
   r2: r0 + r3*2 = 1

   Iteration 2:

   ‘2’ is loaded into r2 as 0x32
   r0: r0 – 30 = 2
   r1: address of the label + 1
   r3: r2 + r2*4 = 1 + 1*4 = 5
   r2: r0 + r3*2 = 3 + 5*2 = 12

   Iteration 3:

   ‘3’ is loaded into r2 as 0x33
   r0: r0 – 30 = 3
   r1: address of the label + 2
   r3: r2 + r2*4 = 12 + 12*4 = 60
   r2: r0 + r3*2 = 3 + 60*2 = 123

   Iteration 4:

   As mentioned above that the string is null terminated, the loop will stop once it reaches the null character and the code ends.

   r0: 3
   r1: 0x0 (null)
   r3: 60
   r2: 123

4. Pins 0 to 16 are configured as inputs with enabled pull-up. Pins 17 to 31 are configured as Totem pole output with input enabled.
   following configuration is required:

Pins	DIR	IEN	PULLEN	OUT
0 -16	0	1	1	1
17-31	1	1	0	X

```
#define PORTA_BASE_ADDR     0x41004400
#define DIRECTION           0x00
#define DIRECTION_CLEAR     0x04
#define DIRECTION_SET       0x08
#define DIRECTION_TOGGLE    0x0c
#define OUTPUT              0x10
#define OUTPUT_CLEAR        0x14
#define OUTPUT_SET          0x18
#define OUTPUT_TOGGLE       0x1c
#define INPUT               0x20
#define CONTROL             0x24
#define W_CONFIG            0x28
#define PIN_CONFIG0         0x40

volatile unsigned uint32_t* PORTA_DIRECTION         = (uint32_t *)(PORTA_BASE_ADDR + DIRECTION);
volatile unsigned uint32_t* PORTA_DIRECTION_CLEAR   = (uint32_t *)(PORTA_BASE_ADDR + DIRECTION_CLEAR);
volatile unsigned uint32_t* PORTA_DIRECTION_SET     = (uint32_t *)(PORTA_BASE_ADDR + DIRECTION_SET);
volatile unsigned uint32_t* PORTA_DIRECTION_TOGGLE  = (uint32_t *)(PORTA_BASE_ADDR + DIRECTION_TOGGLE);
volatile unsigned uint32_t* PORTA_OUTPUT            = (uint32_t *)(PORTA_BASE_ADDR + OUTPUT);
volatile unsigned uint32_t* PORTA_OUTPUT_CLEAR      = (uint32_t *)(PORTA_BASE_ADDR + OUTPUT_CLEAR);
volatile unsigned uint32_t* PORTA_OUTPUT_SET        = (uint32_t *)(PORTA_BASE_ADDR + OUTPUT_SET);
volatile unsigned uint32_t* PORTA_OUTPUT_TOGGLE     = (uint32_t *)(PORTA_BASE_ADDR + OUTPUT_TOGGLE);
volatile unsigned uint32_t* PORTA_INPUT             = (uint32_t *)(PORTA_BASE_ADDR + INPUT);
volatile unsigned uint32_t* PORTA_CONTROL           = (uint32_t *)(PORTA_BASE_ADDR + CONTROL);
volatile unsigned uint32_t* PORTA_W_CONFIG          = (uint32_t *)(PORTA_BASE_ADDR + W_CONFIG);
volatile unsigned uint32_t* PORTA_PIN_CONFIG0       = (uint32_t *)(PORTA_BASE_ADDR + PIN_CONFIG0);

void configIO(){
*(PORTA_DIRECTION_CLEAR) = 0x1fff;
*(PORTA_DIRECTION_SET) = 0xfffe0000;
*(PORTA_OUTPUT_SET) = 0x1ffff;
*(PORTA_W_CONFIG) = 0x4006ffff;
*(PORTA_W_CONFIG) = 0xc0060001;
*(PORTA_W_CONFIG) = 0xc002fffe;}
```