CSE2223 EMBEDDED SYSTEMS SCHEME OF EVALUATION

Q1. After the execution of the following instructions, R3=____.

```
MOV R0,#2_1111 0100
MOV R3,R0,ASR #2 (0.5)
```

1. #2_1111 1010
2. #2_1110 1000
3. **#2_1111 1101
4. #2_1111 0010

Q2. R1=____ after the execution of the following instructions.

```
MOV R1,#0x55
RSB R0,R1,#0 (0.5)
```

1. **0xffffffab
2. 0xfffffeab
3. 0xffffeead
4. 0xffff1ead

Q3. The values of R5 and R6 after execution of the following instructions are ____ and _____ .

```
LDR R0,=0XE635 26AF
LDR R1,=0x4F12 69B3
MOV R2,#0x35
MOV R3,#0x21
SUBS R5,R1,R0
SBC R6,R3,R2 (0.5)
```

1. **68DD 4304 and FFFF FFEB
2. 68DE 5404 and FAAF FFAB
3. 68EE 4334 and FBBF FFBB
4. 68FF 4399and F66F F9EB

Q4. For the instruction STR R1, [R0, #4]! \\ Assume R0= 0x2000 8004, R1= 0x8745 230. ____ is the type of instruction and _____ value of R0 after its execution. (0.5)

1. Post indexed with autoindexing and 0x8745 230
2. Pre-indexed with autoindexing and 0x8745 230
3. **Pre-indexed with autoindexing and 0x2000 8008
4. Post indexed with autoindexing and 0x2000 8004

Q5. ARM Cortex M3 is based on _____ architecture (0.5)

1. Von Neumann

2. **Harvard
3. Cambridge
4. Stanford

Q6. The flags; C & Z, after the execution of the following code is,

```
LDR R2,=0xFFFFFFF1
MOV R3,#0x0F
ADDS R3,R3,R2
ADD R3,R3,#0x7
MOV R1,R3
```
(0.5)

1. C=1 & Z=0
2. **C=1 & Z=1
3. C=0 & Z=0
4. C=0 & Z=1

Q7. Which of the following statement is correct when LPC 1768 is connected to a keypad of in 4X4; rows X columns, matrix fashion. Note row is treated as outputs and columns are treated as inputs. (0.5)

1. All the rows are made high at once and scan the columns for high.
2. The rows are made high sequentially and scan the columns for high.
3. All the columns are made high at once and scan the rows for high.
4. ** The columns are made high sequentially and scan the rows for high.

Q8. If STMDB is used to PUSH the contents into stack the ____ is used as POP instruction. (0.5)

1. STMIB
2. **STMDB
3. STMIA
4. STMDA

Q9. The instructions used to configure P0.6 to P0.13 as GPIO outputs ____ (0.5)

1. LPC_PINCON->PINSEL0 &= 0xF00003FF AND LPC_GPIO0->FIODIR |= 0x0000CF30
2. ** LPC_PINCON->PINSEL0 &= 0xFFC003FF AND LPC_GPIO0->FIODIR |= 0x00003FC0
3. LPC_PINCON->PINSEL0 &= 0xFFD005FF AND LPC_GPIO0->FIODIR |= 0x00005E60
4. LPC_PINCON->PINSEL0 &= 0xFEE03FE AND LPC_GPIO0->FIODIR |= 0x00333FC0

Q10. The maximum delay that can be achieved when P_{CLK} = 51MHz and PR = 50. (0.5)

1. **1 hr 19 min
2. 1 hr 45 min
3. 1 hr 10 min

4. 1 hr 01 min

Q11. List and explain the special function register in ARM. Why they are called special function registers?

4M

Ans:

In ARM the R13, R14, R15, and CPSR (current program status register) registers are called special function registers since each one is dedicated to a specific function.

1M

The R13 is set aside for stack pointer. Stack pointer is used to access the stack. A stack location can be accessed by pointing stack pointer to that location.

1M

The R14 is designated as link register which holds the return address when the CPU calls a subroutine. While returning from subroutine, the link register content is used to resume back.

1M

The R15 is the program counter (PC). The PC (program counter) points to the address of the next instruction to be executed.

0.5M

The CPSR (current program status register) is used for keeping condition flags. The status of the current execution can be reflected in the CPSR.

0.5M

Q12. Write an ARM assembly language program to find whether a given 32-bit hexadecimal number in code memory is palindrome. Store 1 in the data memory if the number is palindrome and 0 if not.

4M

Ans:

```
        Reset_Handler
        LDR R0, =src
        LDR R1, =palin
        LDR R2, [R0]
        MOV R3, R2
        MOV R4, #8          ;Counter
        MOV R5, #28         ;No. of shifts
L1      AND R6, R3, #0xF
        LSL R6, R5
        ORR R7, R6
        SUB R5, #4
        LSR R3, #4
        SUBS R4, #1
        BNE L1
        CMP R7, R2
        BEQ Store
        MOV R8, #0
        STR R8, [R1]
        B STOP
```

```
Store   MOV R8, #1
        STR R8, [R1]
STOP
        B STOP        ; Be there
src  dcd  0xFFFAAFFF

        AREA mydata, DATA, READWRITE
palin   DCD 0
        END
```

Q13. Write an embedded C program for BCD up/down counter using LEDs. The delay between each count is 0.5s. Use P2.20 to P2.23 to drive the LEDs and P2.12 as up/down controller. Assume PCLK=CCLK=3MHz and T_{RES} = 1µs.

3M

Ans:

```
#include <LPC17xx.h>
int main(void)
{
unsigned int j;
unsigned long LED;

//Configure P0.20-P0.23 as output port
LPC_GPIO0->FIODIR |= 0xF00000;
LPC_GPIO2->FIODIR &= 0xFFFFEFFF;   //P2.12 as input port  (0.5M)
        while(1)
        {
                if(LPC_GPIO2->FIOPIN & 1<<12) //switch not pressed    (0.5M)
                {
                        for(LED=15;LED>=0;LED--) //Down Counter
                        {
                                LPC_GPIO0->FIOPIN = LED;
                                delay_lcd();// (0.5M)


                        }
                }

                else
                {
                        for(LED=0;LED<15;LED++) //Up counter
                        {
                                LPC_GPIO0->FIOPIN = LED;
                                delay_lcd();// (0.5M)

                        }
                }
        }
}
void initTimer0(void) (1M)
```

```
{
    LPC_TIM0->CTCR = 0x0;   //timer mode
    LPC_TIM0->TCR = 0x02; //Reset Timer
    LPC_TIM0->PR = 2; //Increment TC at every 2+1 clock cycles, Tres = 1us
    LPC_TIM0->MR0 = 499999; // for 500 ms delay TC counts 0 to 999
    LPC_TIM0->MCR = 2; // reset TC after 500ms delay
    LPC_TIM0->EMR = 0x20; // when match occurs bit 0 of EMR will set
    LPC_TIM0->TCR = 0x01; //enable timer0
}
void delay(void)
{
        initTimer0();
    while(!(LPC_TIM0->EMR &1));
}
```

Q14. Write an embedded C program for the 4-bit mode LCD initialisation process. Use P0.20 to P0.23 for the data lines and P0.24 to P0.25 for the command lines. (3M)

Ans:

```
#include <lpc17xx.h>
void lcd_init(void);
void write_cmd(void);
void delay_lcd(unsigned int);
void lcd_com(int);
unsigned long int temp1=0, temp2=0 ;

int main(void)
{
/* Setting the directions as output */
 LPC_GPIO0->FIODIR |= 0x0F<<20 | 1<<24 | 1<<25  | 1<<26;       (0.5M)

delay_lcd(3200);
lcd_init();
}

//lcd initialization
    void lcd_init() (0.5M)

    {
        //4 bit mode
        lcd_com(0x33);
        delay_lcd(800);

        lcd_com(0x32);
        delay_lcd(800);

        lcd_com(0x28);              //function set
```

```
                delay_lcd(800);

                lcd_com(0x0c);              //display on cursor off
                delay_lcd(800);

                lcd_com(0x06);              //entry mode set increment cursor right
                delay_lcd(800);

                lcd_com(0x01);              //display clear
                delay_lcd(10000);

                return;
        }

void lcd_com(int temp1)
{
        temp2 = temp1 & 0xf0;       // get MS nibble
        temp2 = temp2 << 16;        //data lines from 23 to 26
        write_cmd();
        delay_lcd(30000);
        temp2 = temp1 & 0x0f;       //get LS nibble
        temp2 = temp2 << 20;
        write_cmd();
        delay_lcd(30000);
  return; (1M)
}

// command nibble o/p routine
 void write_cmd(void)            //write command reg
{
LPC_GPIO0->FIOPIN = temp2;    // Assign the value to the data lines
LPC_GPIO0->FIOCLR = 1<<24;  // clear bit RS
LPC_GPIO0->FIOCLR = 1<<25;  // clear bit RW
LPC_GPIO0->FIOSET = 1<<26;  // EN=1
delay_lcd(25);
LPC_GPIO0->FIOCLR  = 1<<26;       // EN =0
 return; (1M)
 }

void delay_lcd(unsigned int r1)
{
 unsigned int r;
 for(r=0;r<r1;r++);
 return;
 }
```

Q15. List and explain the functions of registers available in the LPC1768 timer Module in detail.

3M

Ans:

The LPC1768 microcontroller, part of the LPC17xx family from NXP, features a comprehensive Timer module that is crucial for a wide range of timing and counting applications in embedded systems. Each timer in this module is equipped with several registers that control its operation, from basic counting to complex pulse width modulation. For a university-level engineering examination, understanding the functions of these registers is essential. Here's a detailed explanation of the primary registers found in the LPC1768 Timer Module:

1. Timer Counter (TC)

The Timer Counter register is the heart of the timer module. It increments on every tick derived from the prescaled clock. The rate at which this register increments is determined by the Prescale Register (PR) value. The TC can be used to measure time intervals or generate time delays.

2. Prescale Register (PR)

The Prescale Register determines the rate at which the Timer Counter (TC) increments. It effectively divides the input clock frequency to a lower rate before it is fed to the TC. The value set in PR is the number of peripheral clock pulses minus one that will increment the TC once. For example, if PR is set to 0, the TC increments on every peripheral clock pulse. If PR is set to 1, the TC increments every two pulses, and so on. This allows for fine control over the timer's operation speed.

3. Match Register (MR)

The LPC1768 timer contains multiple Match Registers (MR0, MR1, MR2, MR3), which are used to generate actions based on a match condition. When the TC matches the value in a Match Register, several actions can be configured to occur, such as resetting the TC, stopping the timer, or triggering an interrupt. This feature is widely used for precise timing operations, like generating periodic interrupts or PWM signals.

4. Interrupt Register (IR)

The Interrupt Register indicates which interrupts are pending. This register flags when a match or capture event occurs (based on configuration in the MCR and CCR). Each bit corresponds to a specific event, for instance, a match on MR0 or a capture event on CAP0. The software can check this register to determine the cause of an interrupt and must clear the appropriate bit to reset the interrupt condition.

5. Timer Control Register (TCR)

This register controls the operation of the timer/counter. It allows the timer to be enabled or disabled and provides a reset function. Setting a bit in this register can either halt the TC from

incrementing or reset the TC to zero. This register is essential for starting, stopping, or resetting the timer as the application requires.

6. Capture Register (CR)

In timer modules that support input capture (used for measuring time intervals or detecting external events), the Capture Register stores the value of the TC at the moment an external signal event occurs (e.g., a rising edge on a selected pin). This feature is essential for applications requiring precise time measurement between external events.

7. Match Control Register (MCR)

It configures what action should be taken when a match occurs between the TC and any of the MRs. Actions include interrupting the processor, resetting the TC, and stopping the timer. This register is crucial for setting up periodic operations and timed tasks within an embedded application.

These registers collectively allow the LPC1768 Timer Module to serve various timing-related functions in embedded systems, from simple delays to complex pulse width modulation and event counting. Understanding how to configure and utilize these registers is fundamental for developing time-sensitive applications in embedded systems engineering.

Q16. If the LPC1768's timer is set up with a pre-scale value (PR) of 4999, what is the frequency of the timer interrupts? Assume the CCLK is running at 100 MHz.

3M

Ans:

The LPC1768 microcontroller's timer interrupts' frequency can be calculated based on the CCLK (CPU clock) frequency, the timer's pre-scale value (PR), and the timer counter's increment rate.

Given:

(a) CCLK (CPU Clock) = 100 MHz (which means the clock has a period of 10 ns, since 1 second / 100,000,000 Hz = 10^-7 seconds = 10 ns).
(b) Pre-scale value (PR) = 4999.

The pre-scale register (PR) is used to scale down the CPU clock frequency to a lower frequency by skipping a specified number of CCLK pulses before incrementing the Timer Counter (TC). The actual number of CCLK pulses skipped is one more than the programmed value in PR since the counting starts from 0. Therefore, for a PR value of 4999, the timer counter (TC) increments every 5000 CCLK pulses.

To find the frequency of the timer interrupts, calculate the period of one increment of the timer counter (TC) and then take the reciprocal to get the frequency.

Therefore

1. Calculate the period of one TC increment:

Period of one CCLK pulse = 10 ns

Period of one TC increment = CCLK pulse period * (PR value + 1) = 10 ns * 5000

2. Convert the period of one TC increment into a frequency to find the timer interrupts' frequency:

Frequency = 1 / Period of one TC increment

The frequency of the timer interrupts, with a pre-scale value (PR) of 4999 and a CCLK running at 100 MHz, is **20 kHz.**

Q17. Rewrite the following code snippets using conditional execution instructions.

```
i)      MOV R0, #0
        LDR R1, =0x99999999
        MOV R2, #10
        ADDS R0, R1
        BCC Next
        ADD R1, #1
Next    SUBS R2, #1
ii)     MOV R1, #10
        MOV R2, #20
        MOV R4, #0
        CMP R2, R1
        BEQ L1
        BCC L2
L1      MOV R4, #20
Here    B Here
L2      ADD R4, #10
```

3M

Ans:

```
i)      MOV R0, #0
        LDR R1, =0x99999999
        MOV R2, #10
        ADDS R0, R1
        ADDCS R1, #1
        SUBS R2, #1
```

Removal of BCC – 0.5, ADDS to ADDCS – 0.5, SUBS to be retained-0.5M.

```
ii)     MOV R1, #10
        MOV R2, #20
        MOV R4, #0
        CMP R2, R1
        MOVEQ R4, #20
Here    B Here
```

L2 ADDCC R4, #10
Removal of BCC and BEQ – 0.5M, MOVEQ -0.5, ADDCC – 0.5

Q18. Identify the addressing modes in the following instructions. Also, write the operation performed in each case.

(i) LDR R2, [R4], #4

(ii) LDR R0,[R1,R2,LSL#2]

2M

Ans:

(i) LDR R2, [R4], #4
Addressing mode: Postindexed addressing mode with fixed offset 0.5M
Operation: Load R2 from [R4], R4 = R4+4 0.5M
(ii) LDR R0,[R1,R2,LSL#2]
Addressing mode: Preindexed address mode with offset of a shifted register 0.5M
Operation: Content of location R1+(R2×4) (i.e; R1 + R2 LSL by 2) is loaded to R0 0.5M