PCS3432 - Laboratório de Processadores

Relatório - E7

Bancada B8	
Bruno Mariz	11261826
Roberta Andrade	11260832

7.5.1 Displaying the hex digits in binary to the surface-mounted LEDs

Write ARM assembly to flash the hex digits in binary form to the surface-mounted LEDs in ascending order. Now slightly modify the code to flash the digits in descending order. Make sure to use a delay so that the digits can be seen. The digits should not stop flashing.

Código utilizado no exercício:

```
@ Exercicio 7.5.1

@ 7seg = P[16:10]   set for output
@ leds = P[7:4]     set for output
@ dip  = P[3:0]     clear for input

@ IOPMOD  = 0x3FF5000
@ IOPDATA = 0x3FF5008

    .text
    .global main

main:
    LDR     r0, =0x3FF5000      @ IOPMOD
    LDR     r1, =0xF0           @ Define leds como output
    STR     r1, [r0]

    B       ascending
    B       descending
    BAL     fim

fim:
    MOV     r0, #0x18
    LDR     r1, =0x20026
    SWI     0x0

ascending:
    LDR     r0, =0x3FF5008      @ IOPDATA
    MOV     r1, #0

ascending_loop:
    MOV     r1, r1, LSL #4
```

```
    STR     r1, [r0]
    MOV     r1, r1, LSR #4
    BL      delay
    ADD     r1, r1, #1
    CMP     r1, #16
    MOVGE   r1, #0
    B       ascending_loop

descending:
    LDR     r0, =0x3FF5008      @ IOPDATA
    MOV     r1, #15

descending_loop:
    MOV     r1, r1, LSL #4
    STR     r1, [r0]
    MOV     r1, r1, LSR #4
    BL      delay
    SUB     r1, r1, #1
    CMP     r1, #-1
    MOVLE   r1, #15
    B       descending_loop

delay:
    STMFD   sp!, {r0, lr}
    LDR     r0, =0xFFFFF
    BL      delay_loop
    LDMFD   sp!, {r0, lr}
    MOV     pc, lr

delay_loop:
    CMP     r0, #0
    MOVEQ   pc, lr              @ Retorna da subrotina caso o r0 tenha
chegado em 0
    SUB     r0, r0, #1          @ Decrementa o r0 até o valor de 0 para
aplicar o delay
    B       delay_loop
```

7.5.2 Displaying the contents of a memory location to the surface-mounted LEDs

Write ARM assembly to inspect memory location 0x4000. If the location contains a decimal number 0-15, display the contents in binary on the surface-mounted LEDs. If the location holds any other value, blank the display. As an example, if 0x4000 contains 0xE, then turn on D1, D2, and D3, and turn off D4 to display b1110.

Código utilizado no exercício:

```
@ 7seg = P[16:10]   set for output
@ leds = P[7:4]     set for output
@ dip  = P[3:0]     clear for input
```

```
@ IOPMOD  = 0x3FF5000
@ IOPDATA = 0x3FF5008

    .text
    .globl main
main:
    BL  setup

    LDR r0, dado

    @ checa limite hex
    CMP r0, #0
    MOVLT   r0, #0      @ se menor, valor a ser escrito eh zero
    CMP r0, #15
    MOVGT   r0, #0      @ se maior, valor a ser escrito eh zero

    STMFD   sp!, {r0}   @ salva estado da main
    BL  limparSaida
    LDMFD   sp!, {r0}   @ restaura estado da main

    STMFD   sp!, {r0}   @ passagem de parametro
    BL  printLED

fim:    SWI 0x0

setup:
    @ usa r0-r1
    @ recebe nada
    @ retorna nada

    LDR r0, =0x3FF5000  @ IOPMOD
    LDR r1, =0xF0   @ set leds
    STR r1, [r0]    @ leds = out

    MOV pc, lr

limparSaida:
    @ usa r0-r1
    @ recebe nada
    @ retorna nada

    LDR r0, =0x3FF5008  @ IOPDATA
    LDR r1, [r0]
    MOV r1, r1, LSL #28
    MOV r1, r1, LSR #28
    STR r1, [r0]

    MOV pc, lr

printLED:
    @ usa r0-r2
    @ recebe valor a ser printado
    @ retorna nada
```

```
    LDMFD   sp!, {r0}   @ r0 = valor a ser printado

    @ alinha com IOPDATA
    MOV r0, r0, LSL #4

    LDR r1, =0x3FF5008  @ IOPDATA
    LDR r2, [r1]
    ADD r2, r2, r0
    STR r2, [r1]

    MOV pc, lr

dado:
    .word 7
```

7.5.3 Displaying the contents of a memory location to the seven-segment display

Write ARM assembly to inspect memory location 0x4000. If the location contains a decimal number in the range 0-15, display the contents in hex on the seven-segment LED display. As an example, if 0x4000 contains 14, display an E.

Código utilizado no exercício:

```
@ Exercicio 7.5.3

@ 7seg = P[16:10]   set for output
@ leds = P[7:4]     set for output
@ dip  = P[3:0]     clear for input

@ IOPMOD  = 0x3FF5000
@ IOPDATA = 0x3FF5008

    .text
    .global main

main:
    LDR     r0, =0x3FF5000  @ IOPMOD
    LDR     r1, =0x1FC00    @ Define o display de 7 segmentos como output
    STR     r1, [r0]

    LDR     r0, =dados
    LDR     r2, [r0]

    LDR     r0, =0x3FF5008  @ IOPDATA

    CMP     r2, #0
    BLT     fim
    CMP     r2, #15
    BGT     fim
```

```
    MOV     r2, r2, LSL #10
    STR     r2, [r0]
    MOV     r2, r2, LSR #10

fim:
    MOV     r0, #0x18
    LDR     r1, =0x20026
    SWI     0x0

dados:      .word 10
```

7.5.4 Displaying the contents of an array of memory location to the seven-segment display

Write ARM assembly to inspect memory location 0x3000 to 0x300A. For each location that contains a decimal number in the range 0-15, display the contents in hex on the seven-segment display with long enough delays so that the display is easy to read.

Código utilizado no exercício:

```
@ Exercicio 7.5.4

@ 7seg = P[16:10]   set for output
@ leds = P[7:4]     set for output
@ dip  = P[3:0]     clear for input

@ IOPMOD  = 0x3FF5000
@ IOPDATA = 0x3FF5008

    .text
    .global main

main:
    LDR     r0, =0x3FF5000  @ IOPMOD
    LDR     r1, =0x1FC00    @ Define o display de 7 segmentos como output
    STR     r1, [r0]

    LDR     r0, =0x3FF5008  @ IOPDATA

    LDR     r1, =dados
    LDR     r3, =N
    LDR     r3, [r3]
    ADD     r3, r1, r3, LSL #2

    BL      loop

fim:
    MOV     r0, #0x18
    LDR     r1, =0x20026
    SWI     0x0
```

```
loop:
    CMP     r1, r3
    MOVGE   pc, lr

    LDR     r2, [r1]
    ADD     r1, r1, #4

    CMP     r2, #0
    BLT     loop
    CMP     r2, #15
    BGT     loop

    MOV     r2, r2, LSL #10
    STR     r2, [r0]
    MOV     r2, r2, LSR #10

    B       loop

delay:
    STMFD   sp!, {r0, lr}
    LDR     r0, =0xFFFFF
    BL      delay_loop
    LDMFD   sp!, {r0, lr}
    MOV     pc, lr

delay_loop:
    CMP     r0, #0
    MOVEQ   pc, lr
    SUB     r0, r0, #1
    B       delay_loop

N:          .word 4
dados:      .word 1, 2, 3, 4
```

7.5.5 Displaying the value of the DIP switches to the surface-mounted LEDs

Write ARM assembly to inspect DIP1 to DIP4, which act like four binary digits. Display the contents in binary on the surface-mounted LEDs. See Figure 2-10 of Evaluator-7T User Guide for bit assignments.

Código utilizado no exercício:

```
@ Exercicio 7.5.5

@ 7seg = P[16:10]   set for output
@ leds = P[7:4]     set for output
@ dip  = P[3:0]     clear for input

@ IOPMOD  = 0x3FF5000
```

```
@ IOPDATA = 0x3FF5008

    .text
    .global main

main:
    LDR     r0, =0x3FF5000      @ IOPMOD
    LDR     r1, =0xF0           @ Define leds como output, dip como input
    STR     r1, [r0]

    LDR     r0, =0x3FF5008      @ IOPDATA

    LDR     r1, [r0]
    MOV     r1, r1, LSL #28
    MOV     r1, r1, LSR #28
    MOV     r1, r1, LSL #4

    STR     r1, [r0]

fim:
    MOV     r0, #0x18
    LDR     r1, =0x20026
    SWI     0x0
```

7.5.6 Displaying the value of the DIP switches to the surface-mounted LEDs continuously

Write an ARM assembly program to inspect DIP1 to DIP4 continuously, which act like four binary digits. Display the contents in binary continuously using the surface-mounted LEDs. The program must be stopped manually.

Código utilizado no exercício:

```
@ Exercicio 7.5.6

@ 7seg = P[16:10]   set for output
@ leds = P[7:4]     set for output
@ dip  = P[3:0]     clear for input

@ IOPMOD  = 0x3FF5000
@ IOPDATA = 0x3FF5008

    .text
    .global main

main:
    LDR     r0, =0x3FF5000  @ IOPMOD
    LDR     r1, =0xF0       @ Seta leds como output, dip como input
    STR     r1, [r0]

    LDR     r0, =0x3FF5008  @ IOPDATA
```

```
    B       loop

fim:
    MOV     r0, #0x18
    LDR     r1, =0x20026
    SWI     0x0

loop:
    LDR     r1, [r0]

    MOV     r1, r1, LSL #28
    MOV     r1, r1, LSR #28
    MOV     r1, r1, LSL #4

    STR     r1, [r0]

    B       loop
```

7.5.7 Storing the value of the DIP switches to a memory location

Write ARM assembly to inspect DIP1 to DIP4, which act like four binary digits. Store the contents in memory location 0x4000.

Código utilizado no exercício:

```
@ 7-5-7
@ 7seg = P[16:10]   set for output
@ leds = P[7:4]     set for output
@ dip  = P[3:0]     clear for input

@ IOPMOD  = 0x3FF5000
@ IOPDATA = 0x3FF5008

    .text
    .globl main
main:
    @ setando I/O
    BL  setup

    @ leitura de dip
    BL  lerDIP
    LDMFD   sp!, {r0}       @ r0 = valor lido

    @ store na memoria
    LDR r1, =gaveta     @ r1 = &gaveta[0]
    STR r0, [r1]

fim:    SWI 0x0
```

```
setup:
    @ usa r0-r1
    @ recebe nada
    @ retorna nada

    LDR r0, =0x3FF5000      @ IOPMOD
    LDR r1, =0x1FCF0        @ set 7seg, set leds, clear dip
    STR r1, [r0]        @ 7seg=out, leds=out, dip=in

    MOV pc, lr

lerDIP:
    @ usa r0-r1
    @ recebe nada
    @ retorna valor de DIP

    LDR r0, =0x3FF5008      @ IOPDATA
    LDR r1, [r0]
    MOV r1, r1, LSL #28
    MOV r1, r1, LSR #28

    @ return
    STMFD   sp!, {r1}
    MOV pc, lr

gaveta:
    .word 999
```

7.5.8 Displaying the value of the DIP switches to the seven-segment display

Write ARM assembly to inspect DIP1 to DIP4, which act like four binary digits. Display the hex digit to the seven-segment display.

Código utilizado no exercício:

```
@ Exercicio 7.5.8

@ 7seg = P[16:10]   set for output
@ leds = P[7:4]     set for output
@ dip  = P[3:0]     clear for input

@ IOPMOD  = 0x3FF5000
@ IOPDATA = 0x3FF5008

    .text
    .global main

main:
    LDR     r0, =0x3FF5000  @ IOPMOD
```

```
    LDR     r1, =0x1FC00    @ Seta leds como output, dip como input
    STR     r1, [r0]

    LDR     r0, =0x3FF5008  @ IOPDATA

    LDR     r1, [r0]
    MOV     r1, r1, LSL #26
    MOV     r1, r1, LSR #26
    MOV     r1, r1, LSL #10

    STR     r1, [r0]

fim:
    MOV     r0, #0x18
    LDR     r1, =0x20026
    SWI     0x0
```

7.5.9 Displaying the value of the DIP switches to the seven-segment display continuously

Write an ARM assembly program to continuously inspect DIP1 to DIP4, which act like four binary digits. Display the hex digit to the seven-segment display. The program

Código utilizado no exercício:

```
@ Exercicio 7.5.9

@ 7seg = P[16:10]   set for output
@ leds = P[7:4]     set for output
@ dip  = P[3:0]     clear for input

@ IOPMOD  = 0x3FF5000
@ IOPDATA = 0x3FF5008

    .text
    .global main

main:
    LDR     r0, =0x3FF5000  @ IOPMOD
    LDR     r1, =0x1FC00    @ Seta leds como output, dip como input
    STR     r1, [r0]

    LDR     r0, =0x3FF5008  @ IOPDATA

    B       loop

fim:
    MOV     r0, #0x18
    LDR     r1, =0x20026
    SWI     0x0
```

```
loop:
    LDR     r1, [r0]

    MOV     r1, r1, LSL #28
    MOV     r1, r1, LSR #28
    MOV     r1, r1, LSL #10

    STR     r1, [r0]

    B       loop
```

7.5.10 Displaying an array of memory locations by multiplexing

1. Write ARM assembly to inspect DIP1 to DIP4, which act as a multiplexor. The multiplexor determines access to an array of memory locations starting at 0x4000 and ending at 0x400F. Continuously display the contents of the multiplexed memory location to the seven- segment display.

2. Now make slight modifications to the code so that the contents are displayed to the segment display and the surface-mounted LEDs.

Código utilizado no exercício:

```
@ 7-5-10-pt2
@ 7seg = P[16:10]   set for output
@ leds = P[7:4]     set for output
@ dip  = P[3:0]     clear for input

@ IOPMOD  = 0x3FF5000
@ IOPDATA = 0x3FF5008

    .text
    .globl main
main:
    BL  setup

loop:
    @ leitura de DIP
    BL  lerDIP
    LDMFD   sp!, {r0}       @ r0 = valor em DIP


    @ obter saida do mux
    STMFD   sp!, {r0}       @ passagem de parametro
    BL  MUX         @ chamada de funcao
    LDMFD   sp!, {r0}       @ r0 = saida do MUX (retorno)

    @ limpeza da saida
    STMFD   sp!, {r0}       @ salvar estado
    BL  limparSaida     @ chamada de funcao
```

```
    LDMFD   sp!, {r0}       @ restituir estado

    @ escrita em 7Seg
    STMFD   sp!, {r0}       @ salvar estado
    STMFD   sp!, {r0}       @ passa parametro
    BL  printSevenSeg       @ chamada de funcao
    LDMFD   sp!, {r0}       @ restituir estado

    @ escrita em leds
    STMFD   sp!, {r0}       @ salvar estado
    STMFD   sp!, {r0}       @ passa parametro
    BL  printLED        @ chamada de funcao
    LDMFD   sp!, {r0}       @ restituir estado

    B   loop
fim:    SWI 0x0

setup:
    @ usa r0-r1
    @ recebe nada
    @ retorna nada

    LDR r0, =0x3FF5000      @ IOPMOD
    LDR r1, =0x1FCF0        @ set 7seg, set leds, clear dip
    STR r1, [r0]        @ 7seg=out, leds=out, dip=in

    MOV pc, lr

lerDIP:
    @ usa r0-r1
    @ recebe nada
    @ retorna valor de DIP

    LDR r0, =0x3FF5008      @ IOPDATA
    LDR r1, [r0]
    MOV r1, r1, LSL #28
    MOV r1, r1, LSR #28

    @ return
    STMFD   sp!, {r1}
    MOV pc, lr

MUX:
    @ usa r0, r1, r2
    @ recebe um seletor do mux
    @ retorna a saida do mux


    @ obter entradas
    LDR r0, =entradasMUX

    @ obter seletor
    LDMFD   sp!, {r1}
```

```
    @ obter saida
    LDR r2, [r0, r1, LSL #2]

    @ return
    STMFD   sp!, {r2}

    MOV pc, lr
entradasMUX:
    .word 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0

limparSaida:
    @ usa r0-r1
    @ recebe nada
    @ retorna nada

    LDR r0, =0x3FF5008  @ IOPDATA
    LDR r1, [r0]
    MOV r1, r1, LSL #28
    MOV r1, r1, LSR #28
    STR r1, [r0]

    MOV pc, lr

printSevenSeg:
    @ usa r0, r1, r2, r3, r4
    @ recebe valor a ser impresso
    @ retorna nada

    @ obter valor a ser impresso
    LDMFD   sp!, {r0}

    @ codificar o valor a ser impresso (ja esta alinhado com IOPDATA)
    LDR r1, =sevenSegVals
    LDR r2, [r1, r0, LSL #2]

    @ ler IOPDATA
    LDR r3, =0x3FF5008
    LDR r4, [r3]

    @ escrever campo sevenSeg com novo valor
    ADD r4, r4, r2

    @ imprimir
    STR r4, [r3]

    MOV pc, lr
sevenSegVals:
    .word 0xFC00, 0x1800, 0x16C00, 0x13C00, 0x19800, 0x1B400, 0x1F400,
0x1C00
    .word 0x1FC00, 0x19C00, 0x1DC00, 0x1F000, 0x1E400, 0x17800, 0x1E400,
0x1C400

printLED:
    @ usa r0-r2
```

```
    @ recebe valor a ser printado
    @ retorna nada

    LDMFD   sp!, {r0}   @ r0 = valor a ser printado

    @ alinha com IOPDATA
    MOV r0, r0, LSL #4

    LDR r1, =0x3FF5008  @ IOPDATA
    LDR r2, [r1]
    ADD r2, r2, r0
    STR r2, [r1]

    MOV pc, lr
```

7.5.11 Counting DIP switch state changes

Write ARM assembly to count the number of times DIP switch 4 changes state up to the hex digit F. Display the continuous count to the seven-segment display.

Código utilizado no exercício:

```
@ 7seg = P[16:10]   set for output
@ leds = P[7:4]     set for output
@ dip  = P[3:0]     clear for input

@ IOPMOD  = 0x3FF5000
@ IOPDATA = 0x3FF5008

@ r8 = IOPDATA
@ r1 = count
@ r2 = start
@ r3 = next

    .text
    .global main

main:
    MOV     r0, #15
pronto:
    LDR     r8, =0x3FF5000  @ IOPMOD
    LDR     r1, =0x1FC00    @ Seta 7 segment como output, dip4 como input
    STR     r1, [r8]

    LDR     r8, =0x3FF5008  @ IOPDATA
    LDR     r2, [r8]
    MOV     r2, r2, LSL #28
    MOV     r2, r2, LSR #28
    MOV     r2, r2, LSR #3
```

```
    MOV     r1, #0

    BL      loop

fim:
    SWI     0x0

loop:
    LDR     r3, [r8]
    MOV     r3, r3, LSL #28
    MOV     r3, r3, LSR #28
    MOV     r3, r3, LSR #3

    CMP     r2, r3
    ADDNE   r1, r1, #1
    MOVNE   r2, r3

    STMFD   sp!, {lr}
    BL      delay
    LDMFD   sp!, {lr}

    STMFD   sp!, {r8, lr}
    MOV     r8, r1
    BL      printSevenSeg
    LDMFD   sp!, {r8, lr}

    CMP     r1, r0
    MOVGE   pc, lr

    B       loop


printSevenSeg:
    STMFD   sp!, {r1, r2}
    @ codificar o valor a ser impresso (ja esta alinhado com IOPDATA)
    LDR r1, =sevenSegVals
    LDR r2, [r1, r8, LSL #2]

    LDR     r1, =0x3FF5008  @ IOPDATA
    MOV     r2, r2, LSL #10
STR     r2, [r1]

    LDMFD   sp!, {r1, r2}

    MOV pc, lr
sevenSegVals:
    .word 0b1011111, 0b0000110, 0b0111011, 0b0101111, 0b1100110, 0b1101101,
0b1111101, 0b0000111, 0b1111111
    .word 0b1100111, 0b1110111, 0b1111100, 0b1011001, 0b0111110, 0b1111001,
0b1110001

delay:
        STMFD   sp!, {r8, lr}
```

```
        LDR             r8, =0xFFFFF
        BL              delay_loop
        LDMFD   sp!, {r8, lr}
        MOV             pc, lr

delay_loop:
        CMP             r8, #0
        MOVEQ   pc, lr
        SUB             r8, r8, #1
        B               delay_loop
```