Univerzitet u Sarajevu
Elektrotehnički fakultet
Ugradbeni sistemi 2023 / 24

Izvještaj za laboratorijsku vježbu br. 9

Prekidi i tajmeri

Ime i prezime: **Ismar Muslić**
Broj index-a: **19304**

23. april 2024.

Sadržaj

1 Analiza programskog rješenja

1.1 Zadatak 1

Prvi zadatak predstavljao je upoznavanje sa online simulatorom za pokretanje koda u ARM Assembly-u. Bilo je potrebno napisati assembly kod koji računa prvih N Fibonaccijevih brojeva, te ih upisuje u uzastopne memorijske lokacije. Broj N se definira unutar samog programa, na početku, a u ovom slučaju iznosio je 48. Najprije se u dva registra smjeste prva dva Fibonaccijeva broja, a zatim se petljom računa sljedeći kao zbir dva prethodna, te se upisuje u memoriju. Nakon toga se računa nova adresa na koju će se upisati sljedeći broj, te se sprema prethodni u memoriju da bi se mogao računati naredni.

S obzirom na to da ovaj simulator ne poznaje sistemske pozive za ARM assembly, program se „završava“ beskonačnom petljom u kojoj se vrti nakon što su svi brojevi uspješno generisani.

2.2 Zadatak 2

U drugom zadatku bilo je potrebno napisati assembly kod koji preko *stdin* omogućava u konzoli unos elemenata niza kao ASCII karaktera, te zatim date elemente sortira po rastućem redoslijedu i ispisuje ponovno u konzolu preko *stdout*. Od korisnika se zahtijeva unos karaktera (maksimalno 20), a zatim se isti sortiraju po ASCII vrijednosti i tako sortirani ispisuju u konzoli. Korišteni algoritam sortiranje je Selection Sort.

2.3 Zadatak 3

Treći zadatak je zahtijevao unos cjelobrojne varijable od strane korisnika putem konzole (ponovno *stdin*), koju je zatim bilo potrebno upisati u memoriju na neku lokaciju. Sistemskim pozivom *read* je pročitana 32-bitna cjelobrojna vrijednost, te je ista upisana u memoriju instrukcijom str. Na kraju je korišten sistemski poziv *exit* za izlaz iz programa sa exit kodom 0.

2.3 Zadatak 4

Tražena implementacija u četvrtom zadatku je unos određenog broja cijelih brojeva sa konzole, sortiranje istih te za relevanvni niz ispis najmanjeg i najvećeg elementa, razlike istih, te medijana datog niza. Rješenje je implementirano ponovno traženjem unosa sistemskim pozivom *read* , ovaj put za broj elemenata niza koji će se unijeti, a zatim i samih elemenata. Implementirano je ignorisanje whitespace charactera tako da su uneseni elementi odvojeni pritiskom tipke „Enter“ (Return). Nakon unosa elemenata i spremanja istih u memoriju, pristupa se sortiranju istih. Ponovno je korišten algoritam Selection Sort. Nakon sortiranje jednostavno je i pronaći najmanji i najveći element (prvi i posljednji), kao i njihovu razliku, te medijan niza kao srednji element (ukoliko je neparan broj elemenata u nizu) ili kao prosjek srednja dva elementa (ukoliko je broj elemenata niza paran). Ove vrijednosti se upisuju u konzolu pozivom funkcije *printf* koja je dostupna iz standardne biblioteke ukoliko se asemblira koristeći GCC.

2 Korišteni hardverski resursi

Za potrebe ove laboratorijske vježbe korišteni sistemi su Raspberry Pi Model 2B, te LPC1114ETF. Od dodatne opreme korišteno je sljedeće:

Za LPC1114ETF:

- LED (8x)
- Fizički taster (2x)

ULAZI	IZLAZI
Potenciometar (analogni; DP9, LPC1114ETF)	LED0 – LED7 (digitalni; DP23, DP24, DP25, DP26, DP27, DP5, DP6, DP28, LCP1114ETF)
Tasteri (ugrađeni na LPC1114ETF) (digitalni; DP1 i DP2)	

3 Zaključak

Postojale su izvjesne poteškoće prilikom izvedbe laboratorijske vježbe br. 9, s obzirom na to da je ovo prvi susret sa pisanjem i pokretanjem asemblerskog koda. Bilo je potrebno mnogo referiranja na dokumentaciju kako ARM arhitekture tako i sintaktičkih detalja asemblerskog koda i instrukcijskog seta. Uprkos tome, svi zadaci sa laboratorijske vježbe su uspješno implementirani i u potpunosti funkcionalni.

U implementaciji zadataka 2 i 4 korišteni algoritam sortiranja je Selection Sort. Bez nekih specifičnih razloga je odabran ovaj algoritam, možda primarno zbog bliskosti sa implementacijom istog, mada je bitno naglasiti da postoje i jednostavniji algoritmi sortiranja koji su se mogli implementirati (recimo Bubble Sort), mada je razlika u težini implementacije u assembly-u marginalna.

Cilj ove vježbe bio je upoznavanje sa GNU ARM Assembly jezikom, što je uspješno i postignuto.

4 Prilog

4.1 Zadatak 1/Izvorni kod

U prvom zadatku, s obzirom na to da simulator ne poznaje sistemske pozive, program je „prekinut“ beskonačnom petljom. Da bi se program uspješno pokrenuo na stvarnom sistemu, dovoljno je ukloniti liniju koja vrti beskonačnu petlju (b done), te otkomentirati linije koje izvršavaju sistemski poziv za izlaz iz programa.

```
.section .data
N:          .word 48          @ Ovdje definiramo konstantu N (može biti 47 ili 48)
fibonacci:  .space 192        @ Rezerviramo dovoljno prostora za 48 Fibonaccijevih
brojeva (48 * 4 = 192 bajta)

.section .text
.global _start
.global generate_fibonacci
.global done

_start:
    ldr r1, =N                @ Učitaj adresu N u r1
    ldr r2, [r1]              @ Učitaj vrijednost N u r2
    ldr r3, =fibonacci        @ r3 pokazuje na početak niza Fibonaccijevih brojeva

    mov r4, #1                @ Prvi broj je 1
    str r4, [r3], #4          @ Spremi prvi broj u memoriju i povećaj r3 za 4

    mov r4, #1                @ Drugi broj je 1
    str r4, [r3], #4          @ Spremi drugi broj u memoriju i povećaj r3 za 4

    subs r2, r2, #2           @ Smanji N za 2 jer smo već generirali prva dva broja
    cmp r2, #0
    beq done                  @ Ako je N bilo 2, završavamo jer smo već generirali
dva broja

generate_fibonacci:
    ldr r4, [r3, #-4]         @ Učitaj prethodni broj (n-1)
    ldr r5, [r3, #-8]         @ Učitaj pretprethodni broj (n-2)
    add r4, r4, r5            @ Računaj trenutni broj kao zbroj prethodna dva
    str r4, [r3], #4          @ Spremi trenutni broj u memoriju i povećaj r3 za 4

    subs r2, r2, #1           @ Smanjuj N za 1
    cmp r2, #0
    bne generate_fibonacci    @ Ponavljaj dok r2 ne postane 0
    beq done

done:
    @mov r0, #0               @ Status kod 0
    @mov r7, #1               @ Syscall za exit
    @swi 0                    @ Prekid za syscall
    b done
```

4.2 Zadatak 2/Izvorni kod

```
.data
prompt1:
    .asciz "Unesite proizvoljan tekst:\n"
prompt2:
    .asciz "Unesen je tekst:\n"
tekst:
    .asciz "                    "        @ Buffer to store the input (20 spaces)

.text
.global _start

_start:
    @ Display the first prompt
    mov     r7, #4                      @ syscall number for sys_write
    mov     r0, #1                      @ file descriptor 1 (stdout)
    ldr     r1, =prompt1                @ address of prompt1
    mov     r2, #27                     @ length of prompt1 string
    swi     #0                          @ make the syscall

    @ Read the input text
    mov     r7, #3                      @ syscall number for sys_read
    mov     r0, #0                      @ file descriptor 0 (stdin)
    ldr     r1, =tekst                  @ address of tekst buffer
    mov     r2, #20                     @ number of bytes to read (assuming 20
characters max)
    swi     #0                          @ make the syscall

    @ Selection sort the tekst array
    ldr     r1, =tekst                  @ address of tekst buffer
    mov     r2, #20                     @ number of elements in the array

selection_sort:
    mov     r3, #0                      @ i = 0

outer_loop:
    cmp     r3, r2                      @ if (i >= 20)
    bge     end_sort                    @ break

    mov     r4, r3                      @ min_index = i
    add     r5, r3, #1                  @ j = i + 1

inner_loop:
    cmp     r5, r2                      @ if (j >= 20)
    bge     update_min                  @ break

    ldrb    r6, [r1, r5]                @ r6 = tekst[j]
    ldrb    r7, [r1, r4]                @ r7 = tekst[min_index]
    cmp     r6, r7                      @ if (tekst[j] < tekst[min_index])
    bge     skip_update                 @ skip if r6 >= r7
    mov     r4, r5                      @ min_index = j
```

```
skip_update:
    add     r5, r5, #1                  @ j = j + 1
    b inner_loop                        @ continue inner loop

update_min:
    cmp     r3, r4                      @ if (i != min_index)
    beq     no_swap                     @ skip swap if i == min_index

    ldrb    r6, [r1, r3]                @ r6 = tekst[i]
    ldrb    r7, [r1, r4]                @ r7 = tekst[min_index]
    strb    r7, [r1, r3]                @ tekst[i] = tekst[min_index]
    strb    r6, [r1, r4]                @ tekst[min_index] = tekst[i]

no_swap:
    add     r3, r3, #1                  @ i = i + 1
    b outer_loop                        @ continue outer loop

end_sort:
    @ Display the second prompt
    mov     r7, #4                      @ syscall number for sys_write
    mov     r0, #1                      @ file descriptor 1 (stdout)
    ldr     r1, =prompt2                @ address of prompt2
    mov     r2, #17                     @ length of prompt2 string
    swi     #0                          @ make the syscall

    @ Display the sorted tekst
    mov     r7, #4                      @ syscall number for sys_write
    mov     r0, #1                      @ file descriptor 1 (stdout)
    ldr     r1, =tekst                  @ address of tekst buffer
    mov     r2, #20                     @ length of tekst buffer (20 characters)
    swi     #0                          @ make the syscall

    @ Exit the program
    mov     r7, #1                      @ syscall number for sys_exit
    swi     #0                          @ make the syscall
```

4.3 Zadatak 3/Izvorni kod

```
.section .data
    buffer_size: .int 12            @ Definiramo veličinu bafera za unos
    input_buffer: .space 12         @ Rezerviramo prostor za unos broja

.section .bss
    number: .word 0                 @ Memorijska lokacija za cijeli broj

.section .text
    .global _start

_start:
    @ Učitavanje broja sa stdin
    ldr r1, =input_buffer           @ Učitaj adresu bafera
    ldr r2, =buffer_size            @ Učitaj veličinu bafera
    ldr r2, [r2]                    @ Učitaj vrijednost veličine bafera

    @ Syscall za čitanje sa stdin
    mov r7, #3                      @ Syscall broj za read
    mov r0, #0                      @ File descriptor za stdin
    mov r2, #12                     @ Maksimalni broj bajtova za čitanje
    swi 0                           @ Poziv syscall

    @ Konverzija ASCII na integer
    ldr r0, =input_buffer           @ Adresa bafera sa unesenim podacima
    mov r1, #0                      @ Inicijaliziraj rezultat na 0
    mov r2, #0                      @ Inicijaliziraj privremeni registar za rezultat

convert_loop:
    ldrb r3, [r0], #1               @ Učitaj naredni karakter iz bafera
    cmp r3, #10                     @ Provjeri je li kraj linije (ASCII kod za novi
red)
    beq store_number                @ Ako je kraj linije, spremi broj
    sub r3, r3, #48                 @ Pretvori ASCII karakter u cifru (0-9)

    @ r1 = r1 * 10
    mov r4, r1, lsl #3              @ r4 = r1 * 8
    add r1, r4, r1, lsl #1          @ r1 = r4 + r1 * 2 = r1 * 10

    add r1, r1, r3                  @ Dodaj cifru u rezultat
    b convert_loop                  @ Nastavi konverziju

store_number:
    ldr r0, =number                 @ Učitaj adresu memorijske lokacije
    str r1, [r0]                    @ Spremi konvertirani broj u memorijsku lokaciju

    @ Završetak programa
    mov r7, #1                      @ Syscall broj za exit
    mov r0, #0                      @ Status kod 0
    swi 0                           @ Prekid za syscall
```

4.4 Zadatak 4/Izvorni kod

```
.data
prompt_num_elements:
    .asciz "Unesite broj elemenata (maksimalno 10):\n"
prompt_elements:
    .asciz "Unesite elemente niza:\n"
prompt_min:
    .asciz "Najmanji broj: %d\n"
prompt_max:
    .asciz "Najveci broj: %d\n"
prompt_range:
    .asciz "Opseg: %d\n"
prompt_median:
    .asciz "Medijan: %d\n"
input_buffer:
    .asciz "                    " @ Buffer for input (20 spaces)
array:
    .space 40                     @ Array to hold up to 10 integers (10 * 4 bytes)
max_elements:
    .word 10
num_elements:
    .word 0

.text
.global _start
.extern printf

_start:
    @ Prompt for the number of elements
    mov     r7, #4                @ syscall number for sys_write
    mov     r0, #1                @ file descriptor 1 (stdout)
    ldr     r1, =prompt_num_elements
    mov     r2, #40               @ length of prompt_num_elements string
    swi     #0

read_num_elements:
    @ Read number of elements
    mov     r7, #3                @ syscall number for sys_read
    mov     r0, #0                @ file descriptor 0 (stdin)
    ldr     r1, =input_buffer     @ address of input buffer
    mov     r2, #20               @ number of bytes to read
    swi     #0

    @ Convert input string to integer (num_elements)
    ldr     r1, =input_buffer     @ address of input buffer
    mov     r2, #0                @ initialize result to 0
    mov     r3, #0                @ initialize sign to positive

    @ Skip leading whitespace
skip_whitespace_num:
    ldrb    r4, [r1], #1          @ load byte and increment r1
    cmp     r4, #' '              @ check if byte is a space
```

```
    beq     skip_whitespace_num   @ skip if it is a space
    cmp     r4, #'-'              @ check if byte is '-'
    bne     check_digit_num       @ if not '-', check if digit
    mov     r3, #1                @ set sign to negative
    b       skip_whitespace_num   @ continue skipping

check_digit_num:
    sub     r4, r4, #'0'          @ convert ASCII to digit
    cmp     r4, #9                @ check if valid digit (0-9)
    bhi     end_conversion_num    @ if not valid, end conversion

    mov     r5, r2, lsl #3        @r5 = r2 * 8
    add     r2, r5, r2, lsl #1    @r2 = r5 + r2 * 2 = r2 * 10

    add     r2, r2, r4            @ add the digit to result
    b       skip_whitespace_num   @ continue conversion

end_conversion_num:
    cmp     r3, #0                @ check if number is negative
    beq     store_num_elements    @ if not, skip negation
    rsb     r2, r2, #0            @ negate the result

store_num_elements:
    ldr     r0, =max_elements     @ load maximum number of elements
    ldr     r1, [r0]              @ load max value into r1
    cmp     r2, r1                @ compare num_elements with max_elements
    bhi     exit_program_error    @ if num_elements > max_elements, read again
    ldr     r0, =num_elements     @ load address of num_elements
    str     r2, [r0]              @ store number of elements

    @ Prompt for the elements
    mov     r7, #4                @ syscall number for sys_write
    mov     r0, #1                @ file descriptor 1 (stdout)
    ldr     r1, =prompt_elements
    mov     r2, #23               @ length of prompt_elements string
    swi     #0

read_elements:
    ldr     r0, =num_elements     @ load address of num_elements
    ldr     r3, [r0]              @ load num_elements into r3
    mov     r4, #0                @ initialize index to 0
read_element_loop:
    cmp     r4, r3                @ compare index with num_elements
    bge     selection_sort        @ if index >= num_elements, exit loop

    @ Read each element
    mov     r7, #3                @ syscall number for sys_read
    mov     r0, #0                @ file descriptor 0 (stdin)
    ldr     r1, =input_buffer     @ address of input buffer
    mov     r2, #20               @ number of bytes to read
    swi     #0
```

```
    @ Convert input string to integer
    ldr     r1, =input_buffer     @ address of input buffer
    mov     r2, #0                @ initialize result to 0
    mov     r5, #0                @ initialize sign to positive

    @ Skip leading whitespace
skip_whitespace:
    ldrb    r6, [r1], #1          @ load byte and increment r1
    cmp     r6, #' '              @ check if byte is a space
    beq     skip_whitespace       @ skip if it is a space
    cmp     r6, #'-'              @ check if byte is '-'
    bne     check_digit           @ if not '-', check if digit
    mov     r5, #1                @ set sign to negative
    b       skip_whitespace       @ continue skipping

check_digit:
    sub     r6, r6, #'0'          @ convert ASCII to digit
    cmp     r6, #9                @ check if valid digit (0-9)
    bhi     end_conversion        @ if not valid, end conversion

    mov     r7, r2, lsl #3        @r7 = r2 * 8
    add     r2, r7, r2, lsl #1    @r2 = r7 + r2 * 2 = r2 * 10

    add     r2, r2, r6            @ add the digit to result
    b       skip_whitespace       @ continue conversion

end_conversion:
    cmp     r5, #0                @ check if number is negative
    beq     store_element         @ if not, skip negation
    rsb     r2, r2, #0            @ negate the result

store_element:
    ldr     r0, =array            @ load address of array
    add     r0, r0, r4, lsl #2    @ calculate address of array[r4]
    str     r2, [r0]              @ store the element
    add     r4, r4, #1            @ increment index
    b       read_element_loop     @ repeat for the next element

selection_sort:
    mov     r4, #0                @ i = 0
    ldr     r0, =num_elements     @ load address of num_elements
    ldr     r3, [r0]              @ load num_elements into r3

outer_loop:
    cmp     r4, r3                @ if (i >= 20)
    bge     print_results         @ break

    mov     r6, r4                @ min_index = i
    add     r5, r4, #1            @ j = i + 1

inner_loop:
    cmp     r5, r3                @ if (j >= 20)
```

```
    bge     update_min            @ break
    ldr     r1, =array            @ load address of array
    ldr     r8, [r1, r5, LSL #2]  @ r8 = tekst[j]
    ldr     r7, [r1, r6, LSL #2]  @ r7 = tekst[min_index]
    cmp     r8, r7                @ if (tekst[j] < tekst[min_index])
    bge     skip_update           @ skip if r8 >= r7
    mov     r6, r5                @ min_index = j

skip_update:
    add     r5, r5, #1            @ j = j + 1
    b inner_loop                  @ continue inner loop

update_min:
    cmp     r4, r6                @ if (i != min_index)
    beq     no_swap               @ skip swap if i == min_index

    ldr     r1, =array
    ldr     r8, [r1, r4, LSL #2]  @ r8 = tekst[i]
    ldr     r7, [r1, r6, LSL #2]  @ r7 = tekst[min_index]
    str     r7, [r1, r4, LSL #2]  @ tekst[i] = tekst[min_index]
    str     r8, [r1, r6, LSL #2]  @ tekst[min_index] = tekst[i]

no_swap:
    add     r4, r4, #1            @ i = i + 1
    b outer_loop                  @ continue outer loop

print_results:
    @ Calculate min, max, range, and median
    ldr     r3, =num_elements
    ldr     r3, [r3]
    ldr     r0, =array            @ load address of array
    ldr     r1, [r0]              @ load min element (first element)
    add     r2, r0, r3, lsl #2    @ address of array[num_elements-1]
    sub     r2, r2, #4            @ address of the last element
    ldr     r2, [r2]              @ load max element (last element)
    sub     r3, r2, r1            @ calculate range (max - min)

    @ Calculate median
    ldr     r4, =num_elements     @ load address of num_elements
    ldr     r5, [r4]              @ load num_elements into r5
    mov     r6, r5, lsr #1        @ calculate index for median (r5 / 2)
    tst     r5, #1                @ check if num_elements is odd
    beq     even_median

    @ Odd number of elements
    ldr     r0, =array            @ load address of array
    add     r7, r0, r6, lsl #2    @ address of array[r6]
    ldr     r7, [r7]              @ load median element
    b       median_calculated

    even_median:
    @ Even number of elements: take the average of the two middle elements
```

```
    ldr     r0, =array            @ load address of array
    add     r7, r0, r6, lsl #2    @ address of array[r6]
    ldr     r7, [r7]              @ load element at array[r6]
    sub     r6, r6, #1            @ index for the previous middle element
    add     r0, r0, r6, lsl #2    @ address of array[r6-1]
    ldr     r6, [r0]              @ load element at array[r6-1]
    add     r7, r7, r6            @ sum the two middle elements
    mov     r7, r7, lsr #1        @ divide by 2 to get the average

median_calculated:
    nop
exit_program:
    @ Write the result and exit the program

    @ Copy the results into safe registers
    mov r10, r1
    mov r9, r2
    mov r8, r3
    mov r6, r7

    @ Print results using printf:

    @ Print min
    push {fp, lr}
    add     fp, sp, #4
    ldr     r0, =prompt_min
    mov     r1, r10
    bl      printf                @ call printf with r0 text and r1 %d
    nop
    sub     sp, fp, #4
    pop     {fp, lr}

    @ Print max
    push {fp, lr}
    add     fp, sp, #4
    ldr     r0, =prompt_max
    mov     r1, r9
    bl      printf                @ call printf with r0 text and r1 %d
    nop
    sub     sp, fp, #4
    pop     {fp, lr}

    @ Print range
    push {fp, lr}
    add     fp, sp, #4
    ldr     r0, =prompt_range
    mov     r1, r8
    bl      printf                @ call printf with r0 text and r1 %d
    nop
    sub     sp, fp, #4
    pop     {fp, lr}
```

```
    @ Print median
    push {fp, lr}
    add     fp, sp, #4
    ldr     r0, =prompt_median
    mov     r1, r6
    bl      printf                  @ call printf with r0 text and r1 %d
    nop
    sub     sp, fp, #4
    pop     {fp, lr}

    @ Exit the program
    mov     r7, #1                  @ syscall number for sys_exit
    mov     r0, #0                  @ exit code
    swi     #0                      @ make the syscall

exit_program_error:
       @ mov     r7, #1                  @ syscall number for sys_exit
       @ mov     r0, #12                 @ exit code
       @ swi     #0                      @ make the syscall
       b         _start                  @ ask for user input again
```