Blekinge Tekniska Högskola

DV1493 – Datorteknik

Laboration I - Rekursiv fakultetsberäknig i ARM assembler

Författare: Carina Nilsson

1 Inledning

Syftet med den här laborationen är att stifta en första bekantskap med assemblerprogrammering, samt hur subrutiner (funktioner) kan skrivas och anropas i assembler. Här ges också tillfälle att grundligt lära sig hur en stack fungerar genom att implementera en rekursiv subrutin. Labbmiljön vi använder är en online-simulator som kallas **CPUlator**. Laborationen består av tre delar, där bara den tredje delen behöver redovisas. Uppgiften görs lämpligen i grupper om två. Annan gruppstorlek ska beviljas av handledaren. Grupper med fler än tre deltagare godtas inte. Redovisningen sker genom att du visar upp programkörning och kod för handledaren. De två första delarna är till för uppvärmning. Tänk på att laborationerna i den här kursen kräver betydligt mer arbete än den tid som finns vid det handledda laborationstillfället.

2 Laborationsuppgifter

2.1 Del 1 - Kör ditt första assemblerprogram

Här finns ett litet kodexempel som kan skrivas in och köras som en första bekantskap med assemblerprogrammering. Stega igenom programmet i simulatorn.

```
.data
numbers:
    .word       2, 3, 8, 3, 9, 12, 0
sum:
    .word       0

.text
.global _start
_start:
    LDR r1, =numbers
    MOV r0, #0
again:
    LDR r2, [r1]
    CMP r2, #0
    BEQ finish
    ADD r0, r0, r2
    ADD r1, r1, #4
    BAL again
finish:
    LDR r1, =sum
    STR r0, [r1]
halt:
    BAL halt
.end
```

2.2 Del 2 - Skriv en funktion i assembler

Det här programexemplet innehåller några subrutiner också.
Huvudprogrammet gör ”anrop” (hopp) till subrutinerna i koden. Det ger ett bra tillfälle att träna på att använda stacken lite grann. Subrutinerna **`print_string`** och **`print_number`** är redan färdigskrivna och klara att använda. De gör det möjligt att skriva ut textsträngar och heltal till UART-terminalen som är kopplad till simulatorn. Funktionen `idiv` är en hjälpfunktion till `print_number`. Funktionen **`find_max`** gör egentligen ingenting ännu. Komplettera den funktionens kod så att den returnerar det största värdet i den array av heltal, som avslutas med talet noll, vars adress skickas in som parameter till funktionen. I programmet som anropar funktionen används arrayen `test` som argument till funktionsanropet.

Koden nedan finns i filen `arm_part2.s` som du kan ladda ned från kurssidan i Canvas.

```
// Constants
.equ UART_BASE, 0xff201000     // UART base address
.equ STACK_BASE, 0x10000000             // stack beginning

.equ NEW_LINE, 0x0A

.data
test:
        .word 1
        .word 3
        .word 5
        .word 7
        .word 9
        .word 8
        .word 6
        .word 4
        .word 2
        .word 0

textA: .asciz "Lab1, Assignment 2\n"
textB: .asciz "The max is "
textC: .asciz "Done\n"

.global _start
.text
```

Programmet fortsätter på nästa sida

```
print_string:
/*
-------------------------------------------------------
Prints a null terminated string.
-------------------------------------------------------
Parameters:
  r0 - address of string
Uses:
  r1 - holds character to print
  r2 - address of UART
-------------------------------------------------------
*/
    PUSH {r0-r1, r4, lr}
    LDR r2, =UART_BASE
    _ps_loop:
        LDRB r1, [r0], #1   // load a single byte from the string
        CMP  r1, #0
        BEQ  _print_string   // stop when the null character is found
        STR  r1, [r2]       // copy the character to the UART DATA field
        B    _ps_loop
    _print_string:
        POP {r0-r1, r4, pc}


idiv:
/*
-------------------------------------------------------
Performs integer division
-------------------------------------------------------
Parameters:
  r0 - numerator
  r1 - denominator
Returns:
  r0 - quotient r0/r1
  r1 - modulus r0%r1
-------------------------------------------------------
*/
    MOV r2, r1
    MOV r1, r0
    MOV r0, #0
    B _loop_check
    _loop:
        ADD r0, r0, #1
        SUB r1, r1, r2
    _loop_check:
        CMP r1, r2
        BHS _loop
    BX lr
```

Programmet fortsätter på nästa sida

```
print_number:
/*
-------------------------------------------------------
Prints a decimal number followed by newline.
-------------------------------------------------------
Parameters:
  r0 - number
Uses:
  r1 - 10 (decimal base)
  r2 - address of UART
-------------------------------------------------------
*/
    PUSH {r0-r5, lr}
    MOV r5, #0  //digit counter
    _div_loop:
        ADD r5, r5, #1   // increment digit counter
        MOV r1, #10  //denominator
        BL idiv
        PUSH {r1}
        CMP r0, #0
        BHI _div_loop

    _print_loop:
        POP {r0}
        LDR r2, =#UART_BASE
        ADD r0, r0, #0x30   // add ASCII offset for number
        STR r0, [r2]  // print digit
        SUB r5, r5, #1
        CMP r5, #0
        BNE _print_loop

    MOV r0, #NEW_LINE
    STR r0, [r2]   // print newline
    POP {r0-r5, pc}



/*****************************************************************
  Function finding maximum value in a zero terminated integer array
*****************************************************************/
find_max:
    PUSH    {lr}

/* Add code to find maximum value element here! */
/* Any registers altered by the function beside r0-r3 must be preserved */

    POP     {pc}
```

Programmet fortsätter på nästa sida

```
/**********************
 main program
**********************/
_start:
    LDR     sp, =STACK_BASE
    LDR     r0, =textA
    BL      print_string
    LDR     r0, =test
    BL      find_max
    MOV     r1, r0
    LDR     r0, =textB
    BL      print_string
    MOV     r0, r1
    BL      print_number
    LDR     r0, =textC
    BL      print_string
_end:
    B _end

.end
```

2.3 Del 3 - Rekursiv assemblerfunktion för fakultetsberäkning

Skriv en rekursiv subrutin i ARM-assembler som räknar ut fakultet på ett tal:

$$n! = n \cdot (n-1) \cdot \cdots \cdot 2 \cdot 1 = n \cdot (n-1)!$$

En sådan subrutin kan i ett högnivåspråk se ut så här:

```
1  int factorial(int number)
2  {
3      if (number > 1)
4          return(number * factorial(number - 1));
5      else
6          return(1);
7  }
```

Tänk på hur parametrar och stack ska hanteras så att rekursion blir möjlig.
Beräkningsfunktionen ska inte göra några utskrifter!
Skriv också ett huvudprogram med en loop som anropar den här funktionen och presenterar resultaten av fakultetsberäkning för talen n = 1,2,3,4,5,6,7,8,9,10.
Utgå gärna från filen `arm_part3.s` där hjälpfunktionerna för utskrift finns implementerade.