

Embedded systems

Laboratory 4 – ARM Procedure Call Standard¹

1 Introduction

1.1 Goals

After this laboratory exercise, you should understand the use of C to perform low-level programming, the ARM procedure call standard, modularizing the program in several files and how C and assembler routines can work together.

1.2 Literature

Study the following literature before starting the lab homework or exercises:

- Chapters 5.1-5.5 in: Wolf, Computers as Components.
- [Chapter 6 from the ARM Software Development Toolkit user's manual, about the ARM procedure call standard](#)

2 Preparations

Read the literature and this laboratory exercise manual in detail, and solve the home assignments. Note that you must solve the home assignments, or you will not be allowed to start the laboratory exercise.

2.1 Home Assignment 1

Study the following (not so) simple assembly program, which calls a subroutine that finds the largest number in a vector with N elements:

```
        AREA lab4d, DATA

Test    DCD 1, 3, 5, 7, 9, 8, 6, 4, 2, 0

TextA   DCB  "Lab 4, Home Assigment 1\n",0
TextB   DCB  "The max is: ",0
TextC   DCB  "\nDone\n",0
```

¹ Edited for the course year 2004/2005 by Ingo Sander based on the earlier version of Mats Brorsson and Jan Eric Larsson

```

        AREA lab4c, CODE

; This function prints a null-terminated string
; a1 points to the string
wrline  MOV a2,a1          ; Copy the pointer to a2
        MOV a1,#0x4       ; Set the SYS_WRITE0 code
        SWI 0x123456      ; Make the system call
        MOV pc,lr        ; return from this function

; This function prints a number (less then 10)
; a1 contains the number
wrnum   ADD a2,a1,#0x30    ; Copy the number to a2 and add 0x20 to
                          ; make an ASCII character
        STMFD sp!,{a2}    ; Push character on stack
        MOV a2,sp         ; Put address to character in a2
        MOV a1,#0x3       ; Set the SYS_WRITEC code
        SWI 0x123456      ; Make the system call
        ADD sp,sp,#4      ; Pop without mem.ref.
        MOV pc,lr        ; return from this function

        GLOBAL FindMax    ; Make the symbol FindMax visible outside
                          ; this file

FindMax
        STMFD sp!,{v1,v2} ; Push v1 and v2 (modify as needed)

        ;;; Add code to find maximum value element here! ;;;

        LDMFD sp!,{v1,v2} ; Pop v1 and v2
        MOV pc,lr        ; Jump back to calling routine

        ;;EXTERN FindMaxC ; Uncomment if FindMaxC is used
        ENTRY            ; Comment if main in C program is used

main
        LDR a1,=TextA     ; Load address to welcome text
        BL wrline
        LDR a1,=Test      ; Load address to vector
        BL FindMax        ; Call FindMax subroutine
        MOV v1,a1         ; Save the result in v1

        LDR a1,=TextB     ; Load address to result text
        BL wrline
        MOV a1,v1         ; Copy result to a1
        BL wrnum          ; print the result

        LDR a1,=TextC     ; Load address to goodbye text
        BL wrline

stop    MOV r0,#0x18

```

```

LDR r1,=0x20026
SWI 0x123456

END

```

Read the literature carefully and make sure that you understand the program above in detail. Then write the missing code of the *FindMax* subroutine in assembly language. Note that arguments and results are transferred according to the ARM procedure call standard, as described in the literature. In the subroutine you can use the argument registers a1-a4 as you wish. For variables such as *n* and *Max* you can use the variable registers v1-v8, but if you do, their contents must first be stored on the stack and then restored before the subroutine returns. In the code above, the subroutine *FindMax* pushes the old values of v1 and v2 in it. At the end, these values are restored.

3 Subroutines and the Stack

In high-level languages, the concept of subroutines is important, because it allows for structuring the code into smaller parts. In this laboratory exercise we will study how subroutines are supported in assembly and machine language.

3.1 Assignment 1

Study the following C program and make sure that you understand what it does and how, and all the C language constructions. The declaration of the vector *Test* below uses the C syntax for initialization of vector elements.

```

int Test[10] = { 1, 3, 5, 7, 9, 8, 6, 4, 2, 0 };

int FindMaxC(int Value[])
{
    int n, Max;

    Max = Value[0];
    for (n = 1; n < 10; n = n + 1) {
        if (Value[n] > Max) Max = Value[n];
    }
    return Max;
}

main ()          /* rename this id entry point in assembler is used */
{
    printf("Lab 4, Assignment 1\n");
    printf("The max is %d\n", FindMaxC(Test));
    printf("Done\n");
}

```

This program contains a vector *Test* of ten integer variables initialized with ten single digit numbers in random order. Next, it contains a subroutine *FindMaxC* which takes a vector as input argument and loops through the vector to find the largest number. Finally, the main function, which is called when the program is started, prints a few messages and calls the subroutine.

3.2 Assignment 2

Create a project in ARM project manager, type in the program of Assignment 1, save, build, load to simulator, and test run it. Does it run correctly?

3.3 Assignment 3

Now test the assembly program of Home Assignment 1. Create a project, build, load to simulator, and run. Use the disassembler and step facilities to debug your program, correct all bugs, and verify that it works correctly.

3.4 Assignment 4

The C compiler `armcc` can translate C programs to machine code. It is possible to investigate the result of this translation by inspecting the generated assembly code. Open a command line window (Normally Start -> Programs -> Accessories -> Command prompt). Change directory to wherever your files are stored. Compile your C-program with “`armcc -S file.c`”, assuming your file is named `file.c`. Study the assembly code produced by `armcc` in Assignment 2, and make sure you understand everything. Compare the generated code with the assembly program of Home Assignment 1.

3.5 Assignment 5

Combine the C main program from Assignment 1 with the assembly *FindMax* subroutine of Home Assignment 1. Create a new project containing both a C and an assembly part, and make the necessary changes in the C and assembly source codes. Test the program and verify that it works correctly. Note that C and assembler source code files in the same project must have different names, (i.e., different extensions are not enough).

In this assignment you used program parts in both C and ARM assembly language. In what language is the program that is executed on the computer or in the simulator?

3.6 Assignment 6

Combine the assembly main program from Home Assignment 1 with the C subroutine *FindMaxC* from Assignment 1. Test the program and verify that it works correctly. Note that when using an assembly main program with a C project, you must replace the *start* label with *main*. The C routines already contain a start label, and will execute some C-specific initializations, before they call the main routine.

4 Conclusions

Before you pass the laboratory exercise, think about the questions below and explain to your supervisor:

- What is demanded for C and assembly programs to be able to call each other?
- What is demanded for two different languages to be able to call each other?
- Explain the interface between high-level and assembly language.
- What are the advantages of high-level compared to assembly languages?
- What are the advantages of assembly compared to high-level languages?
- Under what circumstances is assembly programming useful?
- Can it be useful to understand machine code even if you are not using assembly language for programming?