Microprocessor and Assembly Lab: Lab 5

Dr. Upama Kabir, Professor
Computer Science and Engineering, University of Dhaka

March 27, 2023

Contents

1 Objectives **1**

2 Assessment Policy **1**
2.1 Notes 1

3 Stack **3**

4 Multiple Loads and Stores **3**

5 What to do? **4**

6 Useful Hints **5**

7 Submission **5**

1 Objectives

The objectives of this lab is to understand and have familiarize with register based assembly programming for Cortex M4 processor for handling non-recursive, recursive and nested function call.

2 Assessment Policy

The student must complete and demonstrate the lab in the laboratory during the lab hour. After completing the lab student must submit a report and upload the code to the Microprocessor and Microcontroller google classroom.

2.1 Notes

- What happens when we call a function?
  - The Caller is suspended, hands over the control to the Callee
  - Callee performs the requested task

Register	Synonym	Special	Role in the procedure call standard
r15		PC	The Program Counter.
r14		LR	The Link Register.
r13		SP	The Stack Pointer.
r12		IP	The Intra-Procedure-call scratch register.
r11	v8		Variable-register 8.
r10	v7		Variable-register 7.
r9		v6 SB TR	Platform register. The meaning of this register is defined by the platform standard.
r8	v5		Variable-register 5.
r7	v4		Variable register 4.
r6	v3		Variable register 3.
r5	v2		Variable register 2.
r4	v1		Variable register 1.
r3	a4		Argument / scratch register 4.
r2	a3		Argument / scratch register 3.
r1	a2		Argument / result / scratch register 2.
r0	a1		Argument / result / scratch register 1.

Table 2, Core registers and AAPCS usage

Figure 1

  - Callee returns control to the Caller

- Basic Function calls on Cortex M4
  - BL instruction uses R14 (LR) to store the return address
  - So in the simple case, at the end of the function call , MOV PC, LR
  - But, to handle the function call properly, we need to follow APCS (ARM Procedure Call Standard)

- R0-R3 (a1-a4) are used to pass argument values into a function and to return a result value from a function. They may also be used to hold intermediate values within a function (but, in general, only between function calls).
- R12 (IP) may be used by a linker as a scratch register between a main function and any other functions it calls . It can also be used within a routine to hold intermediate values between function calls.
- R4-R8, R10 and R11 (v1-v5, v7 and v8) are used to hold the values of a function's local variables. Of these, only v1-v4 can be used uniformly by the whole Thumb instruction set, but the AAPCS does not require that Thumb code only use those registers.
- The above are the registers which any called function is required to save.
- That means they must have unchanged values when control returns to the calling function (i.e main program)
- If the called function needs these registers for extra workspace then it must preserve the contents of the registers R4-R8, R10, R11 and SP (callee saved).

- If they have been saved then they must be restored before returning to the caller.
- Registers are limited in number. Memory has much larger capacity. We need a organized way to save stuff in memory. Best solution is stack.
- In all variants of the function call standard, registers R12-R15 have special roles. In these roles they are labeled IP, SP, LR and PC.
- The memory of a process can normally be classified into five categories:
  - code (the program being executed), which must be readable, but need not be writable, by the process.
  - read-only static data.
  - writable static data.
  - the heap.
  - the stack.

3 Stack

- Stack provides last in, first out storage
- Two operation: PUSH and POP
- Do we grow the stack downwards (descending address) or upwards (ascending address)in memory?
- We need SP to hold address of the top of the stack (SP is R13)
- The question is: should it point to topmost filled location (stack full) or should it point to next empty location just beyond top of the stack (stack empty)?
- ARM uses " full descending" approach
- Universal stack constraints
  - stack-limit<SP<=stack-base. The stack pointer must lie within the extent of the stack.
  - SP mod 4 = 0. The stack must at all times be aligned to a word boundary.
  - A process may only access (for reading or writing) the closed interval of the entire stack delimited by [SP, stack-base – 1] (where SP is the value of register r13).

4 Multiple Loads and Stores

- Storing multiple register values in a block is much more efficient than handling handling STR and LDR instruction
- ARM has two instructions: LDM and STM
- In a stack based structure, we use SP (R13) as the memory indexer
- APCS uses full descending STMFD and LDMFD

- Lets assume we want to retrieve data from the stack into registers: **LDMFD SP, R0-R3**. Here, SP holds the base address.
- The overall effect:
- **LDR R0, [SP]**
  LDR R0, [SP, #4]
  LDR R0, [SP, #8]
  LDR R0, [SP, #12]

- But notice , in the above sequence, the SP itself has not been changed.
- If we want to alter SP , we will write **LDMFD SP!, R0-R3**
- Data stored on the stack as a part of a function call forms part of stack frame for that function invocation.
- A stack frame can have stored register value, and also allocated space for local variables declared in the function.
- The stack frame contents need to be popped when the function is exited and return to the caller
- All the local variables are vanished when the above situation occurs
- Remember, lowest-address item goes to the lowest numbered registers.
- If we are in a leaf function, we don't need to store LR. But in all other cases we need to do that.
- main function− >BL func1− >BL func2
- BL finc1 in main program stores the return add in LR
- BL func2 inside func1 will overwrite it
- func2 to return to func1 is fine but func1 returns to main using MOV PC, LR would be wrong

5 What to do?

The lab has the following tasks:

1. Write an assembly language program to identify prime numbers from a list of array by calling a function called **prime**.
2. Write an assembly language program to perform the summation of two numbers by call a function **sum(arg1, arg2)** using call-by-value
3. Write an assembly language program to perform the summation of two numbers by call a function **sum(arg1, arg2)** using call-by-reference.
4. Write an assembly language program to find and return the minimum element in an array, where the array and its size are given as parameters by using recursive function.
5. Write an assembly language program to perform the nested function call..
6. You have to submit a detailed design of each program which will contains the status of stack (pictorially) and all other registers .
7. All the documents will be in latex format.

6 Useful Hints

Use the following link for further reading

- About the ARM Procedure Call Standard - Arm Developer: `https://github.com/ARM-software/abi-aa/releases`

7 Submission

You must demonstrate your solution, submit the code, and report (Latex source files and code) to the course website. The submission date is by midnight April 03, 2023 .