# Homework 4: Function Calls in ARM CPE221

**Instructor: Rahul Bhadani**

**Due: March 21, 2025, 11:59 PM**
**100 points**

You are allowed to use a generative model-based AI tool for your assignment. However, you must submit an accompanying reflection report detailing how you used the AI tool, the specific query you made, and how it improved your understanding of the subject. You are also required to submit screenshots of your conversation with any large language model (LLM) or equivalent conversational AI, clearly showing the prompts and your login avatar. Some conversational AIs provide a way to share a conversation link, and such a link is desirable for authenticity. Failure to do so may result in actions taken in compliance with the plagiarism policy.

Additionally, you must include your thoughts on how you would approach the assignment if such a tool were not available. Failure to provide a reflection report for every assignment where an AI tool is used may result in a penalty, and subsequent actions will be taken in line with the plagiarism policy.

### Submission instruction:

Upload a .pdf on Canvas with the format {firstname_lastname}_CPE221_hw04.pdf. For example, if your name is Sam Wells, your file name should be sam_wells_CPE221_hw04.pdf. If there is a programming assignment, then you should include your source code along with your PDF files in a zip file {firstname_lastname}_CPE221_hw04.zip. Your submission must contain your name, and UAH Charger ID or the UAH email address. Please number your pages as well.

---

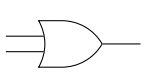
## 1 Assembly Program for Function Calls to Sort Numbers (30 Points)

Write an ARM assembly program that:

1. **(10 points)** Initializes an array x with the following integers:

$$\{1, 2, 4, 5, 3, 6, 8, 7, 10, 9\}.$$





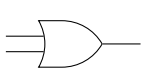
2. **(20 points)** Divides the integers in `x` into two separate arrays, `y` and `z`:
    - The array `y` should store all even integers from `x`.
    - The array `z` should store all odd integers from `x`.
3. **(30 points)** Use a function named `intsort` to perform the sorting. The function must:
    - Take as input pointers to arrays `x`, `y`, and `z`, the index `i`, and the size of the array `x`.
    - Loop through the elements of `x` and separate even and odd integers using modulus operations.
    - Increment the appropriate counters (`j` for `y`, `k` for `z`) as each value is stored in `y` or `z`.
4. **(20 points)** At the end of execution:
    - `y` should contain all even numbers from `x` in the same order as they appear in `x`.
    - `z` should contain all odd numbers from `x` in the same order as they appear in `x`.
5. **(10 points)** Ensure that proper stack usage is maintained in the function `intsort` to save and restore the values of registers.

## Additional Requirements (10 points)

- Use ARM assembly directives to define and reserve memory for the arrays and variables.
- Ensure the program uses proper branching instructions (`B`, `CMP`, etc.) to handle loop conditions and function calls.
- Include comments in your code to explain the logic behind each section.

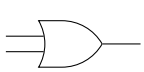
Answer





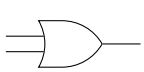
```
.GLOBAL _START
@ MAIN PROGRAM
_START:
      @ Stack Pointer Initialization
      MOV SP, #0x200
         @ LOAD ADDRESS OF ARRAY X
         ADR R0, ARRX
         @ LOAD ADDRESS OF ARRAY Y
         ADR R1, ARRY
         @ LOAD ADDRESS OF ARRAY Z
         ADR R2, ARRZ
         @ Array size = 10
         MOV R3, #10
         @ Push size onto stack
         PUSH {R3}
         @ BRANCH TO FUNCTION
         BL INTSORT
         @ Clean up stack
         ADD SP, SP, #4
DONE: B DONE
```





```
@ FUNCTION DEFINITION
INTSORT:
        @ Save registers (including LR for nested calls)
        @ as we will use these registers within the function
        @ so we need to store the older content of these
        @ registers onto the stack
        PUSH {R3, R6, LR}
        @ R3 = index i (loop counter)
        MOV R3, #0
        @ Load array size (passed via stack)
        LDR R6, [SP, #12]
LOOP:
        @ Compare i with array size
        CMP R3, R6
        BEQ EXIT
        @ INCREMENT LOOP COUNTER
        ADD R3, R3, #1
        @ LOAD NEXT VALUE FROM X, INCREMENT INDEX TO NEXT WORD IN ARRAY AFTER LOAD
        LDR R12, [R0], #4
        @ TEST TO SEE IF VALUE FROM X IS EVEN OR ODD
     TST R12, #0B1
        @ MOVE VALUE TO EVEN ARRAY AND INCREMENT INDEX TO NEXT WORD IN ARRY
        STRNE R12, [R1], #4
        @ MOVE VALUE TO ODD ARRAY AND INCREMENT INDEX TO NEXT WORD IN ARRZ
        STREQ R12, [R2], #4
        @ UNCONDITIONAL BRANCH BACK TO LOOP LABEL
        B LOOP

EXIT:
        @ Restore registers
   POP {R3, R6, LR}
        @ LOAD VALUE FROM LINK REGISTER BACK INTO PC TO RETURN TO MAIN
        BX LR

        @ VARIABLE DECLARATION
ARRX: .WORD 1, 2, 4, 5, 3, 6, 8, 7, 10, 9
ARRY: .SPACE 20
ARRZ: .SPACE 20
```





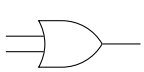
## 2 Recursive Function in ARM (20 Points)

Write a recursive ARMv7 function to compute the factorial of a number. Use the stack to save and restore registers.

Answer

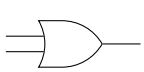
```
@ Write a recursive ARMv7 function to compute
@ the factorial of a number.
@ Use the stack to save and restore registers.

.global _start
_start:
   @ Stack Pointer Initialization
   MOV SP, #0x1000
   LDR R0, =number
   LDR R0, [R0]
   LDR R1, =result
   PUSH {R0}
   BL Factorial
   ADD SP, SP, #4
   STR R0, [R1]

done: B done
```





```
Factorial:
    @ prologue
    PUSH {R4, FP, LR}
    MOV FP, SP
    @ Load argument from stack
          @ 12 = 3 registers pushed * 4 bytes
    LDR R0, [FP, #12]
    @ Base case: if n == 1, return 1
    CMP R0, #1
    MOVEQ R0, #1
    BEQ factorial_exit
    @ Recursive case: n * factorial(n-1)
          @ Set argument for recursive call
          @ Preserve R0, i.e.(n) in R4
    MOV R4, R0
          @ R0 = n-1
    SUB R0, R4, #1
          @ Save current n
    PUSH {R0}

          @ Call factorial(n-1)
    BL Factorial
    @ After recursive call returns
    @ Restore original n
          ADD SP, SP, #4
          @ R0 = n * factorial(n-1)
    MUL R0, R4, R0

factorial_exit:
    @ epilogue
    MOV SP, FP
    POP {R4, FP, LR}
    BX LR

.data
number: .word 5
result: .word 0
```





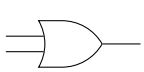
# 3 Nested Function Calls (20 Points)

Write a program in ARMv7 that computes `max(min(a,b,d,),f,g)` by nested function call. You must use frame pointers, stack pointers, and stack frames.

Answer

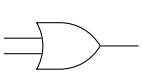
```
@ Write a program in ARMv7 that computes max(min(a,b,d),f,g)
@ by nested function call.
@ You must use frame pointers, stack pointers, and stack frames.

.data

a: .word 5
b: .word 7
d: .word 3
f: .word 8
g: .word 6
result: .word 0

.text
.global _start
_start:

@ initialize the stack
MOV SP, #0x2000
LDR R1,=a
LDR R1, [R1]
LDR R2,=b
LDR R2, [R2]
LDR R3,=d
LDR R3, [R3]
LDR R4,=f
LDR R4, [R4]
LDR R5,=g
LDR R5, [R5]

BL max
```





```
LDR R6, =result
STR R0, [R6]

done: B done

max:
  @ prologue
  PUSH {FP, LR}
  MOV FP, SP

  BL min
  CMP R0, R4
  BGT r0greaterthan
  B r4greaterthan

r0greaterthan:
  CMP R0, R5
  MOVGT R0, R0
  MOVLT R0, R5
  B exitfuncmax

r4greaterthan:
  CMP R4, R5
  MOVGT R0, R4
  MOVLT R0, R5
  B exitfuncmax

exitfuncmax:
  @ epilogue
  MOV SP, FP
  POP {FP, LR}
  BX LR

min:
  @ prologue
  PUSH {FP, LR}
```





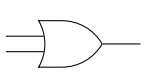
```
    MOV FP, SP

    CMP R1, R2
    BLT r1lessthan
    B r2lessthan

  r1lessthan:
    CMP R1, R3
    MOVLT R0, R1
    MOVGT R0, R3
    B exitfuncmin

  r2lessthan:
    CMP R2, R3
    MOVLT R0, R2
    MOVGT R0, R3
    B exitfuncmin

  exitfuncmin:
    @ epilogue
    MOV SP, FP
    POP {FP, LR}
    BX LR
```

## 4 C++ to ARM (20 Points)

Given the following C++ Code demonstrating a nested function call, write an equivalent ARMv7 code:

```
void inner() { x = 10; }
void outer() {
    int x = 5;
    inner();
}
```





You must also write a program that calls the `outer` function with the appropriate value.

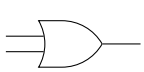
Write down the link register value (LR) as the execution progresses.

Answer

**Code:**

```
.global _start
_start:

        MOV R1, #0
        BL outer

done: B done

outer:
        @prologue
        PUSH {FP, LR}
        MOV FP, SP

        MOV R2, #5
        BL inner

        @epilogue
        MOV SP, FP
        POP {FP, LR}
        BX LR

inner:
        @prologue
        PUSH {FP, LR}
        MOV FP, SP

        @ R1 is declated globally
        MOV R1, #10
        @epilogue
        MOV SP, FP
        POP {FP, LR}
```





```
BX LR
```

**Link Register Values:**

- Initial Value: 00
- After calling the function `outer`, the link register value is set to the address of `B done` line, which is 0x8 in this case.
- After calling the `inner` function from within the `outer` function, link register value is set to the address of the instruction `MOV SP, FP` of the `outer` function, which is 0x1c for the given code.
- Once, function returns from the `inner` and executes `POP {FP, LR}`, the link register value becomes again 0x8.
- That value stays even after returning from the `outer`.

Hence for the entire life-cycle of the program link register value goes from 0x0 to 0x8 to 0x1C to 0x8.