

EE 466 Computer Architecture

Fall 2019

Instructor: Dr. Chen Liu

Project 2: Recursive Calls

Name: Lewis Collum

Student ID: 0621539

Major: EE & CE

Email Address: colluml@clarkson.edu

Date of the report: November 19, 2019

Contents

1 Design	2
1.1 Recursive Sum Basics	2
1.2 Software Environment	2
1.3 Interface Between C and Assembly	2
2 Flow Chart	2
3 Program	2
3.1 Recursive Sum in Assembly	2
3.2 Interface in C	3
4 Result	4
5 Screenshot of the Stack	4
6 Self-Evaluation	4
7 Appendix: Code	4
7.1 Makefile	4

Abstract

We will be implementing a simple recursive sum algorithm in A64 (ARMv8) assembly.

1 Design

1.1 Recursive Sum Basics

For any given element, f_n , is the sum of natural numbers,

$$f_n = 1 + 2 + 3 + \dots + n.$$

So,

$$f_1 = 1.$$

1.2 Software Environment

We will be using the A64 Linaro cross compiler toolchain. Our project source code will be maintained with a simple GNU makefile. The DS-5 (Eclipse) IDE is used for debugging purposes. **Note that the code may not compile unless the makefile we provide is used.**

The makefile can be seen in section 7.1.

1.3 Interface Between C and Assembly

Our recursive sum function, `mysum`, is written in assembly, and called in C. It has a single input parameter `n`, which is the number we want to sum up to. The assembly function then returns the sum.

2 Flow Chart

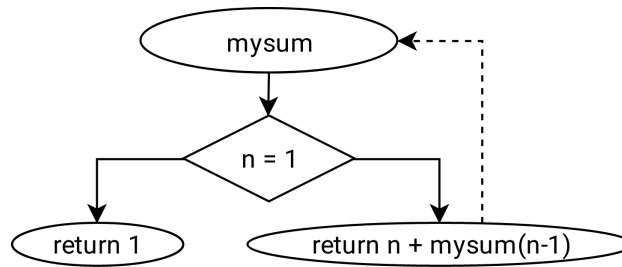


Figure 1: Flowchart depicting the simple recursive sum algorithm, where n is the input parameter.

3 Program

3.1 Recursive Sum in Assembly

```
/*
EE 466 Computer Architecture
Fall 2019
Instructor: Dr. Chen Liu
Project 2: Recursive Calls
Name: Lewis Collum
Student ID: 0621539
Major: EE & CE
Email Address: collum1@clarkson.edu
Date of the report: November 19, 2019
*/

#define n x0
#define return x0
#define recursiveResult x11

.global mysum
mysum:
    // if n == 1, branch to "baseCase"
    cmp n, 1
    b.eq baseCase

    // push n and lr to stack
    sub sp, sp, #(8*2)
    str n, [sp, #0]
    str x30, [sp, #8]

    // recursive function call
    sub n, n, 1
    bl mysum
    mov recursiveResult, return

    // pop n and lr from stack
    ldr x30, [sp, #8]
    ldr n, [sp, #0]
    add sp, sp, #(8*2)

    // return n + mysum(n-1)
    add return, n, recursiveResult
    ret

baseCase:
    //return 1
    mov return, 1
```

3.2 Interface in C

```

/*
  EE 466 Computer Architecture
  Fall 2019
  Instructor: Dr. Chen Liu
  Project 2: Recursive Calls
  Name: Lewis Collum
  Student ID: 0621539
  Major: EE & CE
  Email Address: colluml@clarkson.edu
  Date of the report: November 19, 2019
*/

#include <stdio.h>

extern long long int mysum(long long int n);

int main() {
    int willContinue = 1;
    while(willContinue) {
        // get user input for parameter n.
        int n;
        printf("Input (n): ");
        scanf("%d", &n);

        if (n >= 1) {
            // print result. Then ask if user wants to continue.
            printf("%d\nContinue? (1/0): ", mysum(n));
            scanf("%d", &willContinue);
        } else {
            printf("n must be larger than or equal to 1.\n");
        }
    }
    return 0;
}

```

4 Result

```

Input (n): 0
n must be larger than or equal to 1.
Input (n): -1
n must be larger than or equal to 1.
Input (n): 1
1
Continue? (1/0): 1
Input (n): 2
3
Continue? (1/0): 1
Input (n): 5
15
Continue? (1/0): 1
Input (n): 10
55
Continue? (1/0): 0

```

5 Screenshot of the Stack

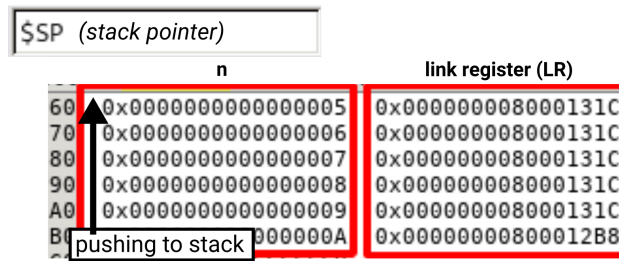


Figure 2: Input (n) starts at 10, or in hex, A. This can be seen in the first push to the stack at the bottom of the figure.

6 Self-Evaluation

The difficult part was getting the DS-5 environment set up without linker errors and debugger errors. Also, we learned that passing the `-g` flag when assembling allows debugging assembly files.

7 Appendix: Code

7.1 Makefile

```

IMAGE=02_recursiveSum.afx
OBSJ = 02_recursiveSum.o mysum.o

CC=aarch64-elf-gcc
LD=aarch64-elf-gcc
AS=aarch64-elf-gcc
CFLAGS=-march=armv8-a -O0

# Select build rules based on Windows or Unix
ifdef WINDIR
    DONE=@if exist $(1) echo Build completed.
    RM=if exist $(1) del /q $(1)
    SHELL=$(WINDIR)\system32\cmd.exe
else
    ifdef windir
        DONE=@if exist $(1) echo Build completed.
        RM=if exist $(1) del /q $(1)
        SHELL=$(windir)\system32\cmd.exe
    else
        DONE=@if [ -f $(1) ]; then echo Build completed.; fi
        RM=rm -f $(1)
    endif
endif

all: $(IMAGE)
    $(call DONE,$(IMAGE))

rebuild: clean all

clean:
    $(call RM,*.o)
    $(call RM,$(IMAGE))
    $(call RM,linkmap.txt)

%.o: %.S
    $(AS) -q -c $< -o $@

```

```
$(IMAGE) : $(OBJS)
# Link with specific base address to suit VE model memory layout
$(CC) $(OBJS) $(CFLAGS) --specs=aem-ve.specs -Wl,--build-id=none,-Map=linkmap.txt -o $@
```