Student Name(s): Alejandra Kudo Student Number(s): 10136014

# CISC-221 Computer Architecture Lab Machine Representation of Programs Lab 5: Bracketing Assembly Code February, 2018

Instructor: Dave Dove ( [dove@cs.queensu.ca](mailto:dove@cs.queensu.ca) ).

**1 Introduction**

Analyzing assembly code is a very useful skill if you can step back and group clusters of instructions into blocks that can then be interpreted as performing some action at a higher level of abstraction, such as performing a transformation on a memory variable, or altering the control flow of a program such as conditionally performing some action, or calling a function. In this lab, you are given both the high level program (in C) and the resulting compilation to assembly language. The objective is to define blocks of assembly language code and relate them to specific statements in the C code. This is also what is required in Project 1, so it’s important to resolve any difficulties in acquiring this skill before you proceed with Project 1.

**2 Logistics**

You may work individually or in pairs for this assignment. All submissions are electronic to the course website.

If you are working as a pair, only one submission is required for two students but make sure that both your names are included in the submission. Either student may make the submission.

**Before the due date and time, submit a commented Word, .txt or pdf file to the Lab5 Submission item on the course web site.**

**3 Instructions**

This is the only file you need for this assignment. Add your comments to the assembly code below.

For each statement of the C program, define the corresponding block of code in the assembly language translation by adding an opening bracket comment @( at the start of the block of instructions, the C statement, and a closing bracket at the end of the block of instructions @). For an example of the format of the required block level comments, see the file: SampleC2Arm.pdf file in the Extras folder. The only comments that you are to add, are the C code statements. For complex C statements, such as a for loop, break up your comments into the individual components of the statement. I have done this for you as an example in the code below – see the for loop in the main function. There is no need to comment global variable declarations.

**4 Helpful Details**

There is nothing in this lab that isn't done in the *HLL Translations - C to ARM* item found in the Extras folder.

You may place the C code either after the opening bracket of the comment block, or before the closing bracket of the comment block.

------------------lab5 C code----------------------

#include <stdio.h>

#define arrsize 12

int sum;

int x;

char msg[]="done!\n";

int darr[arrsize];

int fsum(int arr[],int sz){

int i;

int rslt = 0;

for(i = 0; i<=sz;i++){

rslt = rslt + arr[i];

}

return rslt;

}

void main(){

for(x = 1; x <=12;x++){

darr[x] = x<<1;

}

sum = fsum(darr,arrsize);

printf("sum = %d\n",sum);

printf("%s\n",msg);

}

**Compile & Run Code**

sum = 156

done!

-----------------Lab 5 assembly code-----------------------

.arch armv6

.file "lab5.c"

.comm sum,4,4 int sum;

.comm x,4,4 int x;

.global msg

.data

.align 2

.type msg, %object

.size msg, 7

msg:

.ascii "done!\012\000" char msg[]=”done!\n”;

.comm darr,48,4 int darr[arrsize];

.text

.align 2

.global fsum

.type fsum, %function

fsum:

@ args = 0, pretend = 0, frame = 16

@ frame\_needed = 1, uses\_anonymous\_args = 0

@ link register save eliminated.

str fp, [sp, #-4]! Store fp into -4

add fp, sp, #0 fp = sp + 0

sub sp, sp, #20 sp = sp – 20

str r0, [fp, #-16] store r0 into -16

str r1, [fp, #-20] store r1 into -20

mov r3, #0 r3=0

str r3, [fp, #-12] store r3 into -12 int rslt=0;

mov r3, #0 r3=0

str r3, [fp, #-8] store r3 in -8 int i;

b .L2 branch to .L2

.L3:

ldr r3, [fp, #-8] r3 = 0

mov r3, r3, asl #2 r3 << 2

ldr r2, [fp, #-16] r0

add r3, r2, r3 r3=r2+0

ldr r3, [r3, #0] r3=0

ldr r2, [fp, #-12] r2 =0 rslt=rslt + arr[i];

add r3, r2, r3 r3=0

str r3, [fp, #-12] r3 = 0

ldr r3, [fp, #-8] r3=0

add r3, r3, #1 r3=1 i++

str r3, [fp, #-8] r3=0

.L2:

ldr r2, [fp, #-8] r2=0

ldr r3, [fp, #-20] r3=r1

cmp r2, r3 0 > r3 i<sz;

ble .L3 if 0>r3 then branch to L3

ldr r3, [fp, #-12] r3=0

mov r0, r3 r0=r3

add sp, fp, #0 sp = fp + 0

ldmfd sp!, {fp}

bx lr return from the function return rslt;

.size fsum, .-fsum

.section .rodata

.align 2

.LC0:

.ascii "sum = %d\012\000"

.text

.align 2

.global main

.type main, %function

main:

@ args = 0, pretend = 0, frame = 0

@ frame\_needed = 1, uses\_anonymous\_args = 0

stmfd sp!, {fp, lr}

add fp, sp, #4 fp = sp + 4

ldr r3, .L7 r3 = &x

mov r2, #1 @( r2 = 1

str r2, [r3, #0] for (x = 1; )

b .L5 branch to .L5

.L6:

ldr r3, .L7 r3=&x

ldr r2, [r3, #0] r2=r3=x

ldr r3, .L7 r3=&x L7=x

ldr r3, [r3, #0] r3=x

mov r1, r3, asl #1 x<<1 =2x

ldr r3, .L7+4 r3=&darr

str r1, [r3, r2, asl #2] r2<<2 + darr 🡪 darr[x] darr[x] = x<<1;

ldr r3, .L7 @( r3=&x

ldr r3, [r3, #0] r3=x

add r2, r3, #1 r2 = r3 + 1 🡪 x+ 1

ldr r3, .L7 r3=&x x=r2 🡪 x+1

str r2, [r3, #0] @ x++) )

.L5:

ldr r3, .L7 @( r3=&x

ldr r3, [r3, #0] r3=x x<=12

cmp r3, #12 x > 12

ble .L6 @ x <=12; ) branch less than or equal to

ldr r0, .L7+4 r0=darr

mov r1, #12 r1=12

bl fsum sum= fsum(darr,arrsize);

mov r2, r0 r2=r0

ldr r3, .L7+8 r3=sum

str r2, [r3, #0] r2=sum

ldr r2, .L7+12 r2= “sum =%d\012\000” printf(“sum = %d\n”, sum);

ldr r3, .L7+8 r3=sum

ldr r3, [r3, #0] r3=sum

mov r0, r2 r0=“sum =%d\012\000”

mov r1, r3 r1=sum

bl printf printf(“sum = %d\n”, sum);

ldr r0, .L7+16 r0=”done!\012\000” printf(“%s\n”, msg);

bl puts printf(“%s\n”, msg);

ldmfd sp!, {fp, pc}

.L8:

.align 2

.L7:

.word x 0

.word darr +4

.word sum +8

.word .LC0 +12

.word msg +16

.size main, .-main

.ident "GCC: (Debian 4.6.3-14+rpi1) 4.6.3"

.section .note.GNU-stack,"",%progbits