Jarone Jabonillo

**Fill out this document, copy all items from Concepts 1 through 8, and the Comments/Questions, and paste into your created discussion thread.**

1. **If statements**

*C++*

int x = 5, y= 3, z = 0;

if(x > y) //r0 > r1

z = x+y; //r2 = r0 + r1

*Assembly*

r0 = x

r1 = y

r2 = z

mov r0, #5

mov r1, #3

mov r2, #0

cmp r0, r1

ble next\_1 //Branch if less than

next\_1:

add r2, r0, r1

Affected registers: r0, r1, r2

Code results: 8

1. **If/Else statements**

*C++*

int x = 5, y= 3, z = 0;

if(x > y) //r0 > r1

z = y-x; //r2 = r1 – r0

else

z = x+y; //r2 = r0 + r1

done:

*Assembly*

r0 = x

r1 = y

r2 = z

cmp r0, r1

ble part\_2

sub r2, r1, r0

bal done

part\_2:

add r2, r0, r1

bal done

Affected registers: r0, r1, r2

Code results: Runs calculations for subtraction or addition based on x and y values

1. **While loops**

*C++*

x = 1;

y = 1;

loop:

while (x >= 0){

y = y+1;

x --;

}

done:

*Assembly*

r0 = x

r1 = y

loop:

cmp r0, #0

blt done //Branch if less than

add r1, #1

sub r0, #1

bal loop

done:

Affected registers: r0, r1

Code results: loops through the code incrementing r1 until r0 == 0

1. **Do While loops**

Assume x ,y unsigned

*C++*

loop:

do{

y = y\*4;

x = x/8;

} while (x>0);

*Assembly*

loop:

mov r1, r1 lsl #2 //Multiplies 2\*#

mov r0, r0 lsr #3 //Divides 2/#

cmp r0, #0

bgt loop

…

Affected registers: r0, r1

Code results: runs through the loop label, multiplying r1 by 4 and dividing r0 by 8. It compares the value on r0 with 0 and loops if it’s false

1. *Assembly Notes for functions*

Function\_name\_label: //entry code

push {lr} //Pushes lr and any other registers we want to preserve

…

//Function code

…

pop {lr} //exit code: popping preserved registries and lr into pc

//also putting return value in r0

*C++*

int add\_range(int lower\_limit, int upper\_limit){ //Param(r0, r1)

//adds all integers between lower\_limit and upper\_limit

//(inclusive, meaning including both limits)

int sum = 0; //r2

int counter = lower\_limit; //r3

loop:

while(count <= upper\_limit){

sum = sum+ counter;

counter++;

}

done: return sum;

}

*Assembly*

//Take function name and make it a label

add\_range:

//Doesn’t require push {lr} since lr doesn’t change

//First param r0, second r1

mov r2, #0 //Sum

mov r3, r0 //r3 = lower\_limit

loop:

cmp r3, r1

bgt done

add r2, r3

add r3, #1

bal loop

done:

mov r0, r2

mov pc, lr

Affected registers: r0, r1, r2, r3

Code results: Function takes two values for the parameter, copies r0 to r3, then adds r3 to r2 and decrements it by one before looping. Loop ends once r3 is greater than r1

6.

Affected registers:

Code results:

7.

Affected registers:

Code results:

8.

Affected registers:

Code results:

Comments/Questions?