Eric Tashji

Homework Assignment 2

Problem 1 (20 points) Convert the following code into LEGv8. Assume g and h are in registers X19, and X20 respectively:

If (g < h)
g = g + h;
else
h = h * 2

Answer: The following lines of code will complete the same task as above, but is in LEGv8:

```
        SUBS XZR, X19, X20 //Compare g and h
        B.GE L2 //If g is greater than or equal to h, move to the else statement (L2)
        ADD X19, X19, X20 //g = g + h
L1:     B L3 //Skip the Else statement (L2) by going to L3
L2:     LSL X20, X20, #1 //Perform the else statement: h = h * 2
L3:
```

Explanation: The first instruction will compare X19(g) and X20(h). If X19(h) is greater than or equal to X20 (g), then it will skip the next line and jump to L2 (the else statement), which will set X20(h) equal to X20(h) * 2. Otherwise, it will set X19(g) equal to X19(g) + X20(h). Then, it will use the B L3 line to skip L2 and go to L3, where it will end.

Problem 2 (20 points) Consider the following high-level code. Assume the array temp is initialized before it is used, and that register X20 holds the base address of temp. Convert the code to LEGv8.

for(i = 0; i < 100; i = i + 1)
temp[i] = temp[i] + 1

Answer: The following lines of code will complete the same task as above, but is in LEGv8:

Eric Tashji

```
        ADDI X10, XZR, #0 //Initialize i to 0
Loop:   SUBIS XZR, X10, #100 //compare i and 100
        B.GE Exit //If i is greater than or equal to 100, exit
        LDUR X9, [X20, #0] //load current index of temp (temp[i]) into the temp register X9
        ADDI X9, X9, #1 //Add 1 to the value in X9
        STUR X9, [X20, #0] //Add X9 back into the current index of temp (temp[i])
        ADDI X20, X20, #8 //Add 8 to X20 address to get next element in temp[i]
        ADDI X10, X10, #1 //Increment i
        B Loop //Go back to the loop
Exit:
```

Explanation: The first line initializes X10 (i) to 0. The next line compares X10 (i) to 100. If it is greater than 100, the following line will force the code to move to the exit. Otherwise, it will continue with the loop. It will then draw the value stored at memory address X20 (temp[i]) into X9. Then, it will add 1 to X9 before storing it back into the memory address at X20 (temp[i]). Then, X20 will be incremented by 8 to find the next index in the array temp. Finally, X10 (i) will be incremented by 1.

Problem 3 (15 points) Consider the high level code:

```
int f, g, y //global 64-bit variables

int sum (int a, int b) {

        return(a+b)
}

int main (void) // at memory address X0 + 800
{
  f=2;
  g=3;
  y= sum(f, g)
  return y;
}
```

Convert this code to LEGv8, making valid assumptions about registers and register use. Note that brackets and global variable declarations are not affecting the addresses of the instructions in memory.

Answer: The following code will perform the same task as the code above, but is in LEGv8:

```
        ADDI X19, XZR, #2 //Initialize f to 2
        ADDI X20, XZR, #3 //Initialize f to 3
        ADD X0, XZR, X19 //Save f into X0 for use as an argument with sum (a)
        ADD X1, XZR, X20 //Save g into X1 for use as an argument with sum (b)
        BL Sum //Go to sum and save return address
        ADD X21, X2, XZR //Save y as X21
        ADD X0, X21, XZR //return y
        B Exit //Exit the program
Sum:
        ADD X2, X0, X1 //add and return a + b
        BR X30 //Return to the return address
Exit:
```

Explanation: The code begins by initializing X19 (f) and X20 (g) to their initial values in main. Then, it copies them into X0 (a) and X1 (b) for use with the sum procedure. Then, it branches to the Sum procedure, where it adds X0 (a) and X1 (b) into X2 before returning it. Then, it will return to the return address in main. From there, main will save X2 into X21 (setting y to the result of sum) before returning y. Finally, it will use B Exit to exit the program.

Problem 4 (15 points) Provide the instruction type, assembly language instruction, and binary representation of instruction described by the following LEGv8 fields:

Op=0x7c2 Rn=11 Rt=4 const=0x4

Answer: This is a D-type instruction. The specific instruction is LDUR X4, [X11, #32].
The binary representation of this instruction is: 11111000010001000000000101100100. This can be broken down as: opcode (11 bits): 11111000010, address (9 bits): 001000000, op2 (2 bits): 00, Rn (5 bits): 01011, Rt (5 bits): 00100.

Problem 5 (15 points) For the following C statement, write a minimal sequence of LEGv8 assembly instructions that performs the identical operation. Assume value of B is stored in register X9, and the base address of the array C is X11:

B = C[2] << 4;

Answer: The following code will perform the objective above, but will be in LEGv8

Eric Tashji

```
LDUR X10, [X11, #16] //Retrieve the element at the 2nd index of C
LSL X9, X10, #4 //Shift left by 4 and save as B
```

Explanation: The first line of the program takes element 2 in the array C (X11) by shifting the memory locations by 16 (2 * 8 = 16) and stores it in register X10. Then, it uses LSL to make the binary shift necessary and store it in B (X9), as according to the code above.

Problem 6 (15 Points) Translate the following LEGv8 code into C. Assume the C-level integer I is held in register X10, X0 holds the C-level integer called result, and X1 holds the base address of the integer MemArray.

```
        ORR   X10, XZR, XZR
LOOP: LDUR X11, [X1, #0]
        ADD   X0, X0, X11
        ADDI  X1, X1, #8
        ADDI  X10, X10, #1
        CMPI X10, 100
        B.LT LOOP
```

Answer: The following code performs the tasks of the code above but is written in C:

```
for (int i = 0; i < 100 ++i) {
        result =  result + memArray[i]
}
```

Explanation: The first line of the LEGv8 code establishes i (X10) as equal to 0. The second line of the LEGv8 code loads the data in the current index of MemArray (X1) into the temporary value X11. Then, that data is added to result (X0). X1 is incremented to retrieve the next element of MemArray and i is incremented. The next line checks to see if i is less than 100. If so, the loop will continue. If not, the loop will end.