Tutorial 4 – Week of Oct. 4th 2021

Question 1) 2.27
Translate the following loop into C. Assume that the C-level integer `i` is held in register `x6`, `x5` holds the C-level integer called `result`, and `x10` holds the base address of the integer

```
.data
MemArray:                    // array of 100 words
          .word     0
          .word     1
// …
          .word     99

.text
     addi x6, x0, 0
     addi x5, x0, 0
     addi x29, x0, 100
LOOP:lw   x7, 0(x10)
     add  x5, x5, x7
     addi x10, x10, 4
     addi x6, x6, 1
     blt  x6, x29, LOOP
```

Solution:

```
int i;
int result = 0;
for (i = 0; i < 100; i++) {
     result += *MemArray;
     MemArray++;
}
return result;
```

// Many people would write the code this way

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

Question 2) 2.31

Translate function f into RISC-V assembly language. Assume the function declaration for `g` is
`int g(int a, int b).`

The code for function f is as follows:

```
int f(int a, int b, int c, int d){
     return g(g(a,b), c+d);
}
```

Solution:

```
f:
     addi x2, x2, -8      // Allocate stack space for 2 words
     sw   x1, 0(x2)       // Save return address
     add  x5, x12, x13    // x5 = c+d
     sw   x5, 4(x2)       // Save c+d on the stack
     jal  x1, g           // Call x10 = g(a,b)
     lw   x11, 4(x2)      // Reload x11= c+d from the stack
     jal  x1, g           // Call x10 = g(g(a,b), c+d)
     lw   x1, 0(x2)       // Restore return address
     addi x2, x2, 8       // Restore stack pointer
     jalr x0, 0(x1)
```

Question 3) 2.37

Write the RISC-V assembly code to implement the following C code as an atomic "set max" operation using the `lr.d` and `sc.d` instructions. Here, the argument `shvar` contains the address of a shared variable which should be replaced by `y` if `y` is greater than the value it points to. Assume `x10` is address of integer pointed by `shvar` and value of `y` is in `x11`.

```
void setmax(int* shvar, int y) {
     // Begin critical section
     if (y > *shvar)
          *shvar = y;
     // End critical section}
}
```

Solution:

```
setmax:
     try:
     lr.d x5, (x10)         # Load-reserve *shvar
     bge  x5, x11, release # Skip update if *shvar > y
     addi x5, x11, 0

release:
     sc.d x7, x5, (x10)
     bne  x7, x0, try # If store-conditional failed, try again
     jalr x0, 0(x1)
```