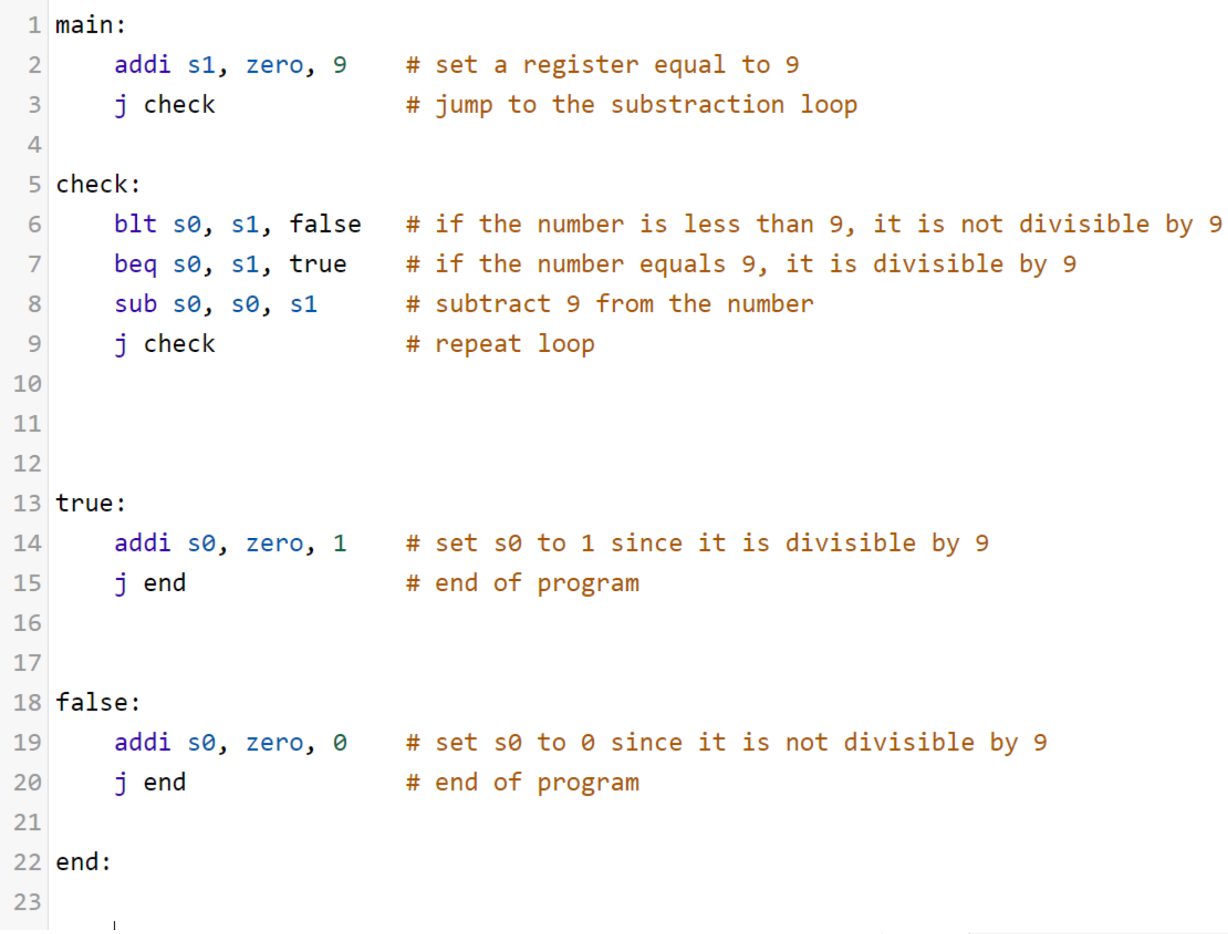
Rylan Wade

Computer Hardware Design

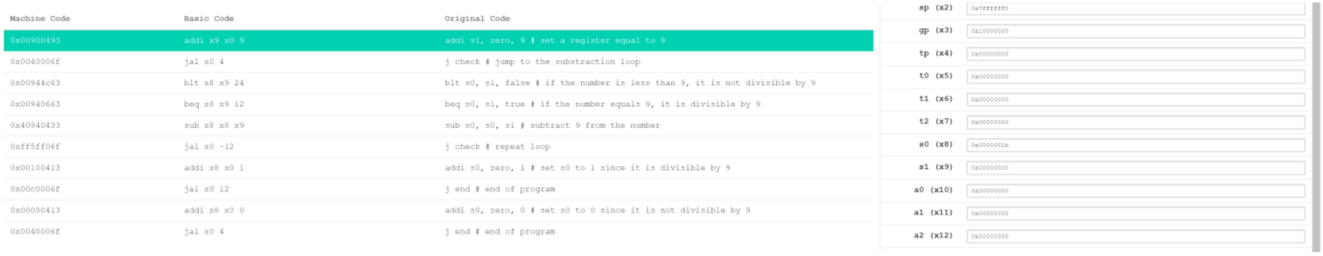
10/5/23

Lab 4 – Assembly

1. I spent 6 hours on this lab.
2. div9.s



```
main:
    addi s1, zero, 9    # set a register equal to 9
    j check             # jump to the substraction loop

check:
    blt s0, s1, false   # if the number is less than 9, it is not divisible by 9
    beq s0, s1, true    # if the number equals 9, it is divisible by 9
    sub s0, s0, s1      # subtract 9 from the number
    j check             # repeat loop



true:
    addi s0, zero, 1    # set s0 to 1 since it is divisible by 9
    j end               # end of program


false:
    addi s0, zero, 0    # set s0 to 0 since it is not divisible by 9
    j end               # end of program

end:

```

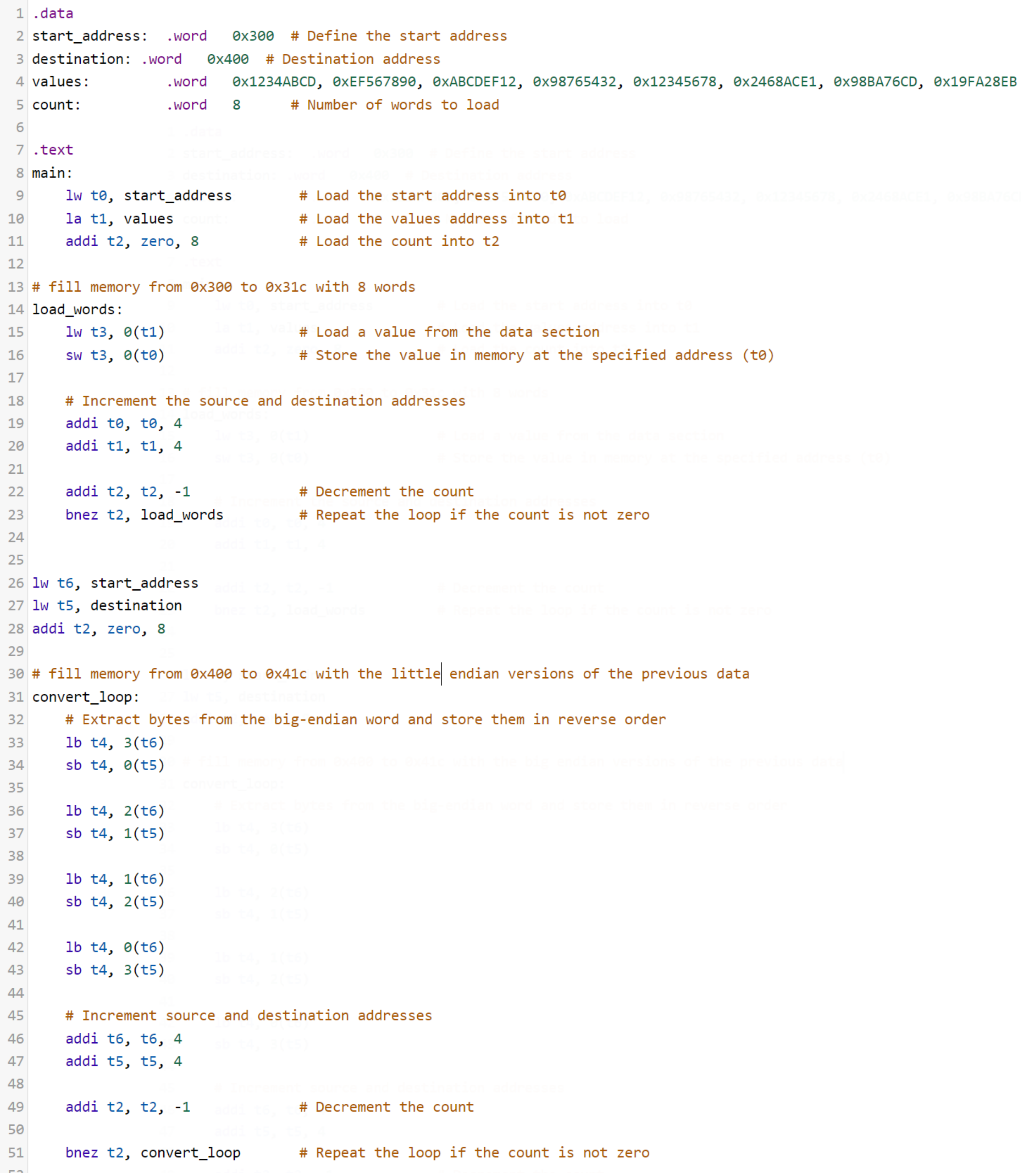


Machine Code	Basic Code	Original Code
0x00900493	addi x9 x0 9	addi s1, zero, 9 # set a register equal to 9
0x0040006f	jal x0 4	j check # jump to the substraction loop
0x00944c63	blt x8 x9 24	blt s0, s1, false # if the number is less than 9, it is not divisible by 9
0x00940663	beq x8 x9 12	beq s0, s1, true # if the number equals 9, it is divisible by 9
0x40940433	sub x8 x8 x9	sub s0, s0, s1 # subtract 9 from the number
0xff5ff06f	jal x0 -12	j check # repeat loop
0x00100413	addi x8 x0 1	addi s0, zero, 1 # set s0 to 1 since it is divisible by 9
0x00c0006f	jal x0 12	j end # end of program
0x00000413	addi x8 x0 0	addi s0, zero, 0 # set s0 to 0 since it is not divisible by 9
0x0040006f	jal x0 4	j end # end of program

sp (x2)	0x7ffffff0
gp (x3)	0x10000000
tp (x4)	0x00000000
t0 (x5)	0x00000000
t1 (x6)	0x00000000
t2 (x7)	0x00000000
s0 (x8)	0x0000001b
s1 (x9)	0x00000000
a0 (x10)	0x00000000
a1 (x11)	0x00000000
a2 (x12)	0x00000000

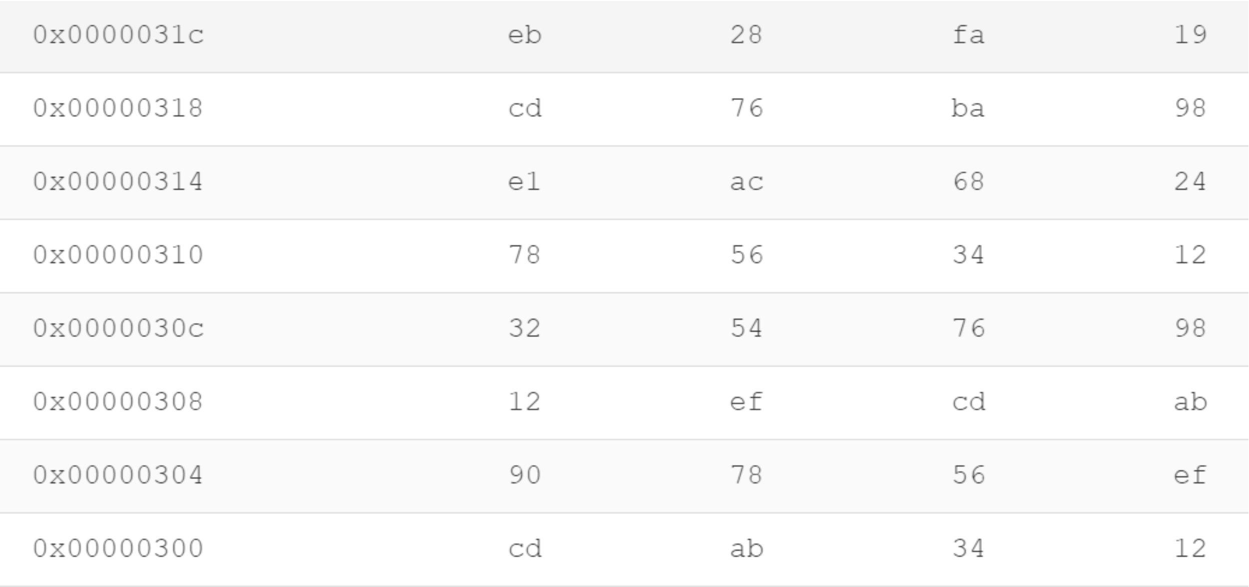
Above is my code and the simulation output. TO test my program, I first initialized s0 to 0x1B, which is 27 in decimal. After pressing the step button several times, I saw that the value of s0 went 0x1b -> 0x12 -> 0x09 -> 0x1. This is as expected since it decreased by 0x9 each time. When testing a larger input (s0 = 0x2328 = 9000 in decimal), I can see that the final value for s0 is 1 which is also as expected. With input s0 = 0x2c = 44 I decimal, I can use the step button to see that s0 equals 0x2c -> 0x23 -> 0x1a -> 0x11 -> 0x08 -> 0x0. This is as expected since 44 is not divisible by 9.

3. big2little.s

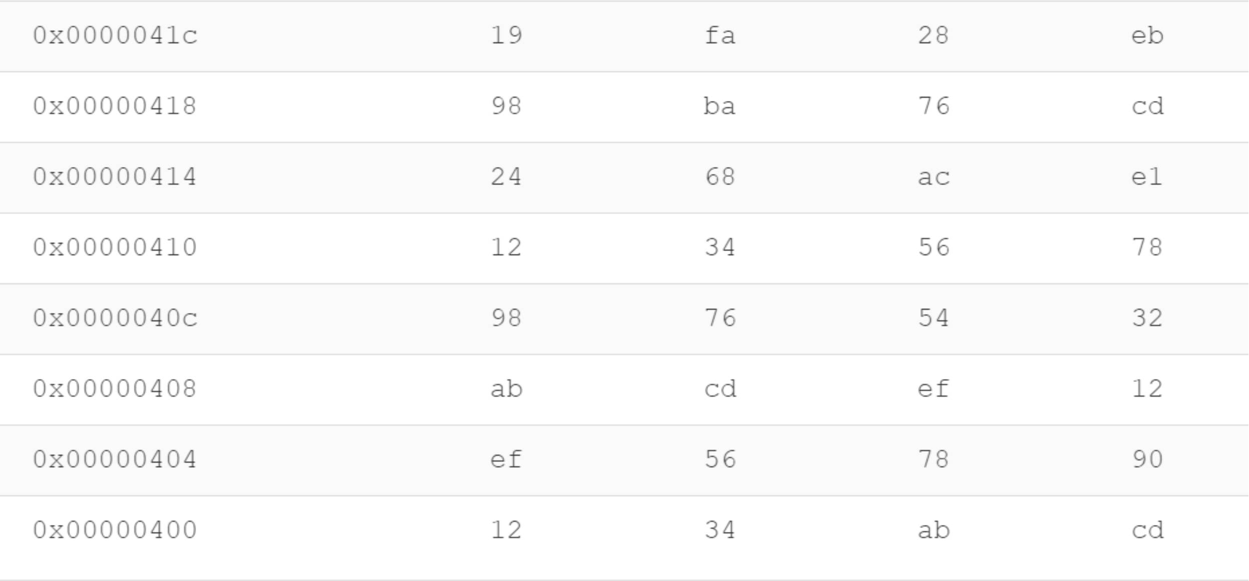


```
.data
start_address:  .word   0x300  # Define the start address
destination: .word   0x400  # Destination address
values:         .word   0x1234ABCD, 0xEF567890, 0xABCDEF12, 0x98765432, 0x12345678, 0x2468ACE1, 0x98BA76CD, 0x19FA28EB
count:          .word   8      # Number of words to load

.text
main:
    lw t0, start_address        # Load the start address into t0
    la t1, values               # Load the values address into t1
    addi t2, zero, 8            # Load the count into t2

# fill memory from 0x300 to 0x31c with 8 words
load_words:
    lw t3, 0(t1)                # Load a value from the data section
    sw t3, 0(t0)                # Store the value in memory at the specified address (t0)

    # Increment the source and destination addresses
    addi t0, t0, 4
    addi t1, t1, 4

    addi t2, t2, -1             # Decrement the count
    bnez t2, load_words         # Repeat the loop if the count is not zero


lw t6, start_address
lw t5, destination
addi t2, zero, 8

# fill memory from 0x400 to 0x41c with the little endian versions of the previous data
convert_loop:
    # Extract bytes from the big-endian word and store them in reverse order
    lb t4, 3(t6)
    sb t4, 0(t5)

    lb t4, 2(t6)
    sb t4, 1(t5)

    lb t4, 1(t6)
    sb t4, 2(t5)

    lb t4, 0(t6)
    sb t4, 3(t5)

    # Increment source and destination addresses
    addi t6, t6, 4
    addi t5, t5, 4

    addi t2, t2, -1             # Decrement the count

    bnez t2, convert_loop       # Repeat the loop if the count is not zero
```

The first part of my code (the load_words loop) loads the 8 words on line 4 into memory locations 0x300 through 0x31c. The first picture below shows the memory locations with the initial words loaded in. Then, the second loop in my code (convert_loop) takes the value from the initial memory and converts it to little endian and then stores it in the destination memory. The destination memory is 0x400 to 0x41c. This can be seen in the second picture below. Upon examining the output in the two pictures below, I can tell that my code correctly converted the data from little endian to big endian.

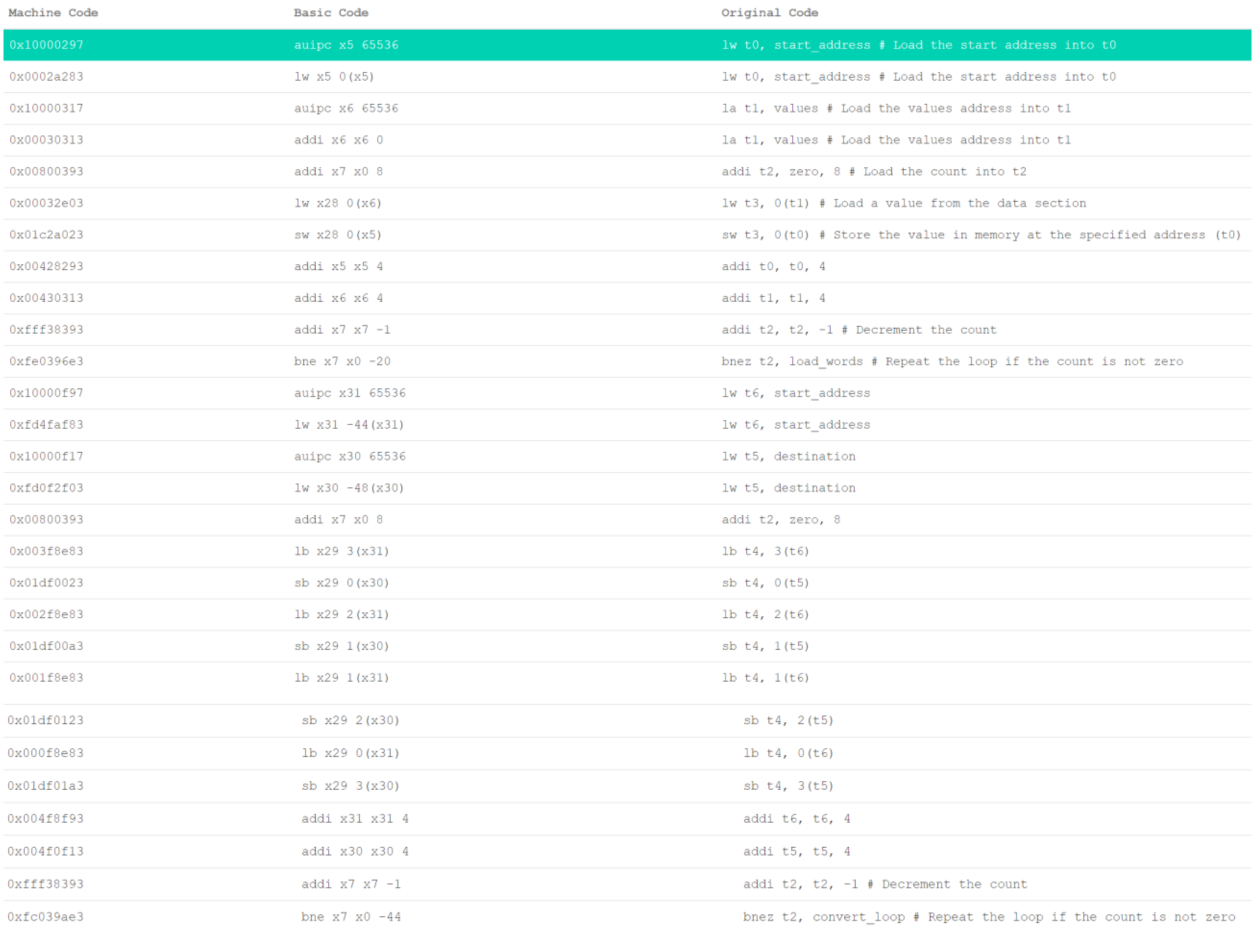


```
0x0000031c      eb    28    fa    19
0x00000318      cd    76    ba    98
0x00000314      e1    ac    68    24
0x00000310      78    56    34    12
0x0000030c      32    54    76    98
0x00000308      12    ef    cd    ab
0x00000304      90    78    56    ef
0x00000300      cd    ab    34    12
```



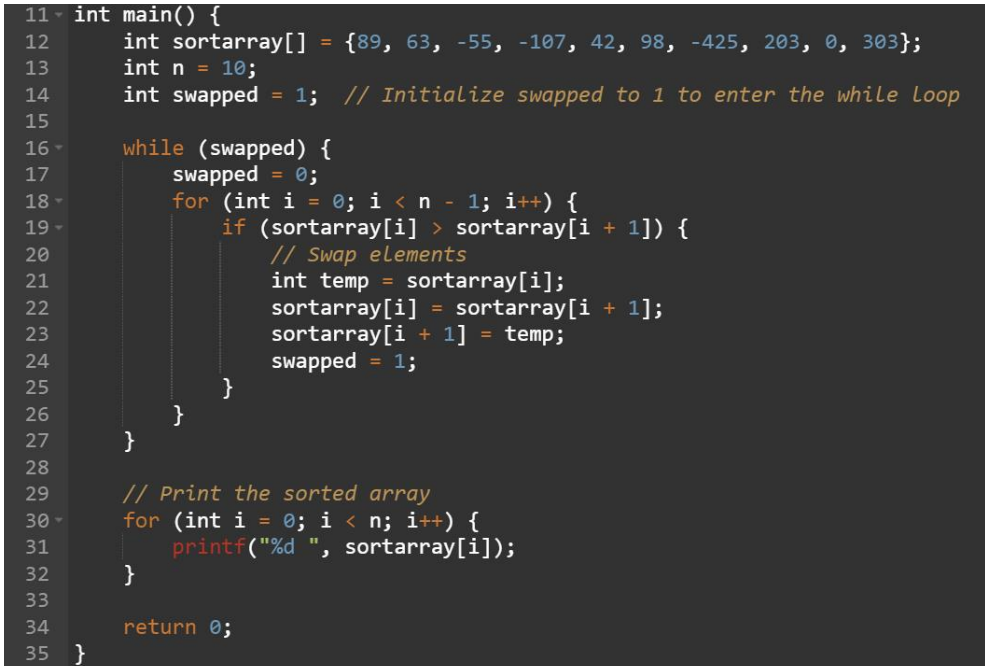
```
0x0000041c      19    fa    28    eb
0x00000418      98    ba    76    cd
0x00000414      24    68    ac    e1
0x00000410      12    34    56    78
0x0000040c      98    76    54    32
0x00000408      ab    cd    ef    12
0x00000404      ef    56    78    90
0x00000400      12    34    ab    cd
```

Below is the machine code output from the simulation of my big2little program.



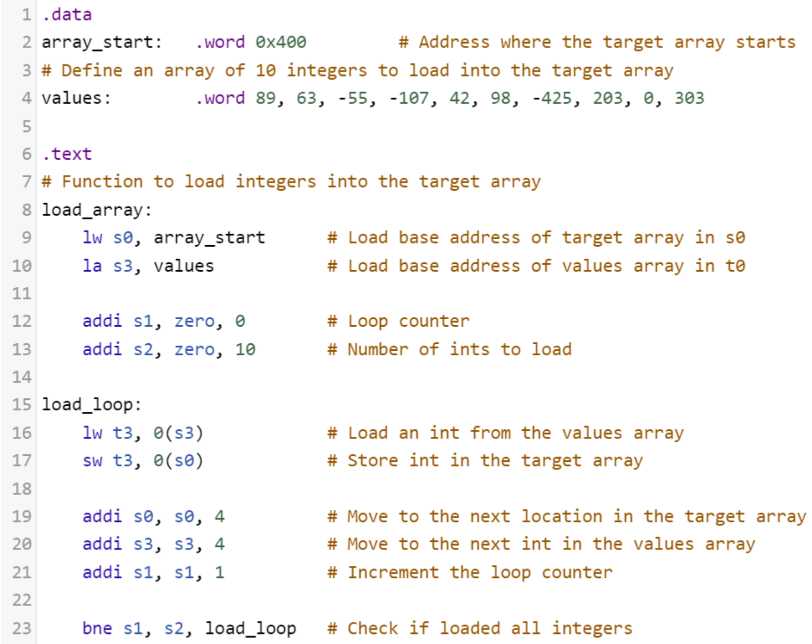
Machine Code	Basic Code	Original Code
0x10000297	auipc x5 65536	lw t0, start_address # Load the start address into t0
0x0002a283	lw x5 0(x5)	lw t0, start_address # Load the start address into t0
0x10000317	auipc x6 65536	la t1, values # Load the values address into t1
0x00030313	addi x6 x6 0	la t1, values # Load the values address into t1
0x00800393	addi x7 x0 8	addi t2, zero, 8 # Load the count into t2
0x00032e03	lw x28 0(x6)	lw t3, 0(t1) # Load a value from the data section
0x01c2a023	sw x28 0(x5)	sw t3, 0(t0) # Store the value in memory at the specified address (t0)
0x00428293	addi x5 x5 4	addi t0, t0, 4
0x00430313	addi x6 x6 4	addi t1, t1, 4
0xfff38393	addi x7 x7 -1	addi t2, t2, -1 # Decrement the count
0xfe0396e3	bne x7 x0 -20	bnez t2, load_words # Repeat the loop if the count is not zero
0x10000f97	auipc x31 65536	lw t6, start_address
0xfd4faf83	lw x31 -44(x31)	lw t6, start_address
0x10000f17	auipc x30 65536	lw t5, destination
0xfd0f2f03	lw x30 -48(x30)	lw t5, destination
0x00800393	addi x7 x0 8	addi t2, zero, 8
0x003f8e83	lb x29 3(x31)	lb t4, 3(t6)
0x01df0023	sb x29 0(x30)	sb t4, 0(t5)
0x002f8e83	lb x29 2(x31)	lb t4, 2(t6)
0x01df00a3	sb x29 1(x30)	sb t4, 1(t5)
0x001f8e83	lb x29 1(x31)	lb t4, 1(t6)
0x01df0123	sb x29 2(x30)	sb t4, 2(t5)
0x000f8e83	lb x29 0(x31)	lb t4, 0(t6)
0x01df01a3	sb x29 3(x30)	sb t4, 3(t5)
0x004f8f93	addi x31 x31 4	addi t6, t6, 4
0x004f0f13	addi x30 x30 4	addi t5, t5, 4
0xfff38393	addi x7 x7 -1	addi t2, t2, -1 # Decrement the count
0xfc039ae3	bne x7 x0 -44	bnez t2, convert_loop # Repeat the loop if the count is not zero

4. bubblesort.s



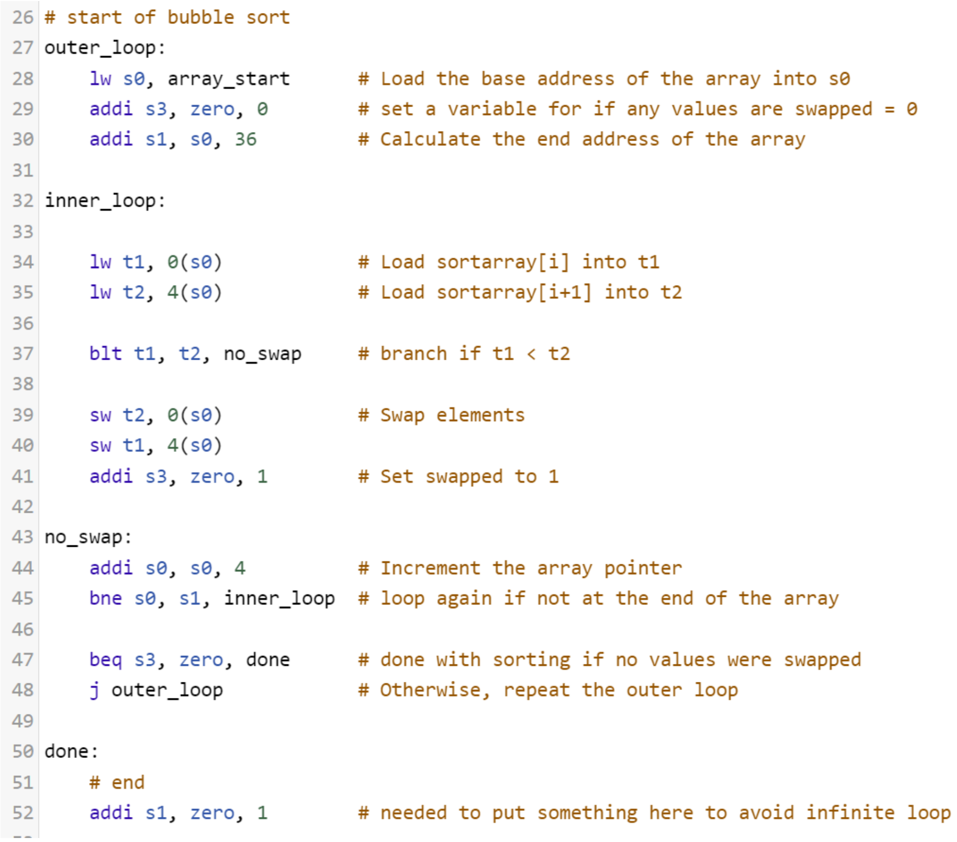
```c
int main() {
    int sortarray[] = {89, 63, -55, -107, 42, 98, -425, 203, 0, 303};
    int n = 10;
    int swapped = 1;  // Initialize swapped to 1 to enter the while loop

    while (swapped) {
        swapped = 0;
        for (int i = 0; i < n - 1; i++) {
            if (sortarray[i] > sortarray[i + 1]) {
                // Swap elements
                int temp = sortarray[i];
                sortarray[i] = sortarray[i + 1];
                sortarray[i + 1] = temp;
                swapped = 1;
            }
        }
    }

    // Print the sorted array
    for (int i = 0; i < n; i++) {
        printf("%d ", sortarray[i]);
    }

    return 0;
}
```

C code for bubble sort above.



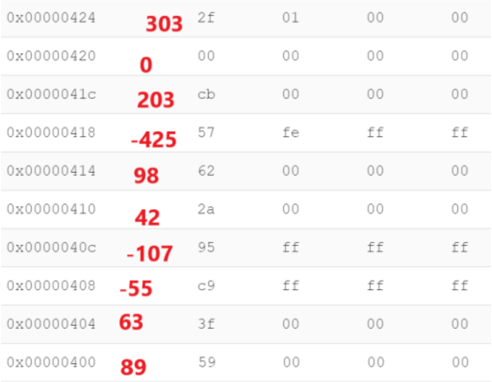
```
.data
array_start:   .word 0x400         # Address where the target array starts
# Define an array of 10 integers to load into the target array
values:        .word 89, 63, -55, -107, 42, 98, -425, 203, 0, 303

.text
# Function to load integers into the target array
load_array:
    lw s0, array_start      # Load base address of target array in s0
    la s3, values           # Load base address of values array in t0

    addi s1, zero, 0        # Loop counter
    addi s2, zero, 10       # Number of ints to load

load_loop:
    lw t3, 0(s3)            # Load an int from the values array
    sw t3, 0(s0)            # Store int in the target array

    addi s0, s0, 4          # Move to the next location in the target array
    addi s3, s3, 4          # Move to the next int in the values array
    addi s1, s1, 1          # Increment the loop counter

    bne s1, s2, load_loop   # Check if loaded all integers
```

This code above is what I used to load the values into the array. The array starts at 0x400 and contains the 10 integers mentioned in the instructions.



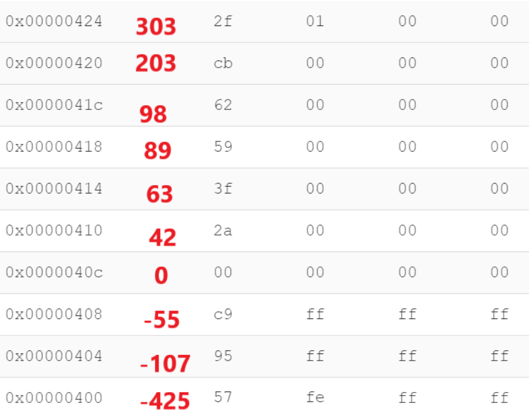
```
# start of bubble sort
outer_loop:
    lw s0, array_start      # Load the base address of the array into s0
    addi s3, zero, 0        # set a variable for if any values are swapped = 0
    addi s1, s0, 36         # Calculate the end address of the array

inner_loop:

    lw t1, 0(s0)            # Load sortarray[i] into t1
    lw t2, 4(s0)            # Load sortarray[i+1] into t2

    blt t1, t2, no_swap     # branch if t1 < t2

    sw t2, 0(s0)            # Swap elements
    sw t1, 4(s0)
    addi s3, zero, 1        # Set swapped to 1

no_swap:
    addi s0, s0, 4          # Increment the array pointer
    bne s0, s1, inner_loop  # loop again if not at the end of the array

    beq s3, zero, done      # done with sorting if no values were swapped
    j outer_loop            # Otherwise, repeat the outer loop

done:
    # end
    addi s1, zero, 1        # needed to put something here to avoid infinite loop
```

This is my assembly code for the bubble sort algorithm.



0x00000424	303	2f	01	00	00
0x00000420	0	00	00	00	00
0x0000041c	203	cb	00	00	00
0x00000418	-425	57	fe	ff	ff
0x00000414	98	62	00	00	00
0x00000410	42	2a	00	00	00
0x0000040c	-107	95	ff	ff	ff
0x00000408	-55	c9	ff	ff	ff
0x00000404	63	3f	00	00	00
0x00000400	89	59	00	00	00

Above is the 10 values loaded into the array before it has been sorted. They are originally displayed in hex and the red annotation is the decimal value.



0x00000424	303	2f	01	00	00
0x00000420	203	cb	00	00	00
0x0000041c	98	62	00	00	00
0x00000418	89	59	00	00	00
0x00000414	63	3f	00	00	00
0x00000410	42	2a	00	00	00
0x0000040c	0	00	00	00	00
0x00000408	-55	c9	ff	ff	ff
0x00000404	-107	95	ff	ff	ff
0x00000400	-425	57	fe	ff	ff

Here are the values of the array after the bubble sort algorithm has been run. The values are correctly sorted in ascending order, which matches my expectations.

On the next page is the machine code output from the simulator

Machine Code	Basic Code	Original Code
0x10000417	auipc x8 65536	lw s0, array_start # Load base address of target array in s0
0x00042403	lw x8 0(x8)	lw s0, array_start # Load base address of target array in s0
0x10000997	auipc x19 65536	la s3, values # Load base address of values array in t0
0xffc98993	addi x19 x19 -4	la s3, values # Load base address of values array in t0
0x00000493	addi x9 x0 0	addi s1, zero, 0 # Loop counter
0x00a00913	addi x18 x0 10	addi s2, zero, 10 # Number of ints to load
0x0009ae03	lw x28 0(x19)	lw t3, 0(s3) # Load an int from the values array
0x01c42023	sw x28 0(x8)	sw t3, 0(s0) # Store int in the target array
0x00440413	addi x8 x8 4	addi s0, s0, 4 # Move to the next location in the target array
0x00498993	addi x19 x19 4	addi s3, s3, 4 # Move to the next int in the values array
0x00148493	addi x9 x9 1	addi s1, s1, 1 # Increment the loop counter
0xff2496e3	bne x9 x18 -20	bne s1, s2, load_loop # Check if loaded all integers
0x10000417	auipc x8 65536	lw s0, array_start # Load the base address of the array into s0
0xfd042403	lw x8 -48(x8)	lw s0, array_start # Load the base address of the array into s0
0x00000993	addi x19 x0 0	addi s3, zero, 0 # set a variable for if any values are swapped = 0
0x02440493	addi x9 x8 36	addi s1, s0, 36 # Calculate the end address of the array
0x00042303	lw x6 0(x8)	lw t1, 0(s0) # Load sortarray[i] into t1
0x00442383	lw x7 4(x8)	lw t2, 4(s0) # Load sortarray[i+1] into t2
0x00734863	blt x6 x7 16	blt t1, t2, no_swap # branch if t1 < t2
0x00742023	sw x7 0(x8)	sw t2, 0(s0) # Swap elements
0x00642223	sw x6 4(x8)	sw t1, 4(s0)
0x00100993	addi x19 x0 1	addi s3, zero, 1 # Set swapped to 1
0x00440413	addi x8 x8 4	addi s0, s0, 4 # Increment the array pointer
0xfe9412e3	bne x8 x9 -28	bne s0, s1, inner_loop # loop again if not at the end of the array
0x00098463	beq x19 x0 8	beq s3, zero, done # done with sorting if no values were swapped
0xfcdff06f	jal x0 -52	j outer_loop # Otherwise, repeat the outer loop
0x00100493	addi x9 x0 1	addi s1, zero, 1 # needed to put something here to avoid infinite loop