Bryan Martinez

CSE 3666

4 February 2024

1)

1.)

add t0, s0, s1 1001 1000 1010 1011 1100 1101 0110 1010
0001 0000 1111 1111 0101 1010 1001 1000
= 1011 1001 1010 1011 0010 1000 0000 0010

t0 = 0x B9AB2802

and t1, s0, s1 1001 1000 1010 1011 1100 1101 0110 1010
0010 0000 1111 1111 0101 1010 1001 1000
0000 0000 1010 1011 0100 1000 0000 1000

t1 = 0x00AB4808

or t2, s0, s1 1001 1000 1010 1011 1100 1101 0110 1010
0010 0000 1111 1111 0101 1010 1001 1000
= 1011 1000 1111 1111 1101 1111 1111 1010

t2 = 0xB8FFDFFA

xor t3, s0, s1 1001 1000 1010 1011 1100 1101 0110 1010
0010 0000 1111 1111 0101 1010 1001 1000
= 1011 1000 0101 0100 1001 0111 1111 0010

t3 = 0xB85497F2

add t4, s0, 0x2FA 1001 1000 1010 1011 1100 1101 0110 1010
+ 0000 0000 0000 0000 0000 0010 1111 1010
= 1001 1000 1010 1011 1101 0000 0110 0100

t4 = 0x98ABD064

And t5, S0, -16 1001 1000 1010 1011 1100 1101 0110 1010
1111 1111 1111 1111 1111 1111 1111 0000
1001 1000 1010 1011 1100 1101 0110 0000

t5 = 0x98ABCD60

Sll: t6, S0, 12 1001 1000 1010 1011 1100 1101 0110 1010

= 1011 1100 1101 0110 1010 0000 0000 0000

t6 = 0xBCD6A000

Srai S2, S0, 8 1001 1000 1010 1011 1100 1101 0110 1010

= 1111 1111 1001 1000 1010 1011 1100 1101

S2 = 0xFF98ABCD

2)

```
        .globl  main

        .text
main:   lui     s2, 0x12345     # load upper 20 bits to s2
        addi    s2, s2, 0x678   # load the rest of 12 bits

        add     t0, x0, s2      # create a copy of s2 to t0

        addi    t2, x0, 0       # counter -> i = 0
        addi    t3, x0, 4       # max -> t4 = 4

loop:   beq     t2, t3, exit    # if the counter (i) is equal to t3 (4), then exit
        addi    t4, x0, 0xFF    # assign to t4 binary sequence of all zeros except for the last 8 bits being 1's
        and     t5, t0, t4      # use and to compare the last 8 bits of t0 - extract the last 8 bits of t0 and assign to t5

        slli    s4, s4, 8       # shift bit to left 8 bits to make space for new orientation
        add     s4, s4, t5      # add t5 to s4
        srli    t0, t0, 8       # shift bit right 8 bits to move the next 8 bits into place

        addi    t2, t2, 1       # increment t2 by 1
        beq     x0, x0, loop    # go back to beginning of loop

exit:   addi    a7, x0, 34      # syscall 34 to print hex
        addi    a0, s4, 0       # assign s4 to a0 as input
        ecall

        addi    a7, x0, 10      # exit
        ecall
```

0x12345678 is loaded into s2 using lui and addi, and a copy is made into t0. A loop is then used where it runs for 4 times to perform the correct number of iterations to rearrange the entire hex sequence. Bit shifting is used throughout the code to move the last 8 bits into place after each iteration, and then extract it to add it to s4 by using an and operation with 0xFF. At the end, system call 34 is used to print the hexadecimal number.

3)

a) If s0 is 0xFF00FF00, 146 instructions are going to be executed. The number of executed instructions depends on the number of 1’s in s0, but it does not depend on the location of the 1’s since whenever a 1 is detected in t0, the increment instruction will run. For all other bits that are 0, 2 instructions are going to run in the loop label but will skip the addi instruction. Then, all bits are going to run the skip label regardless of whether it is a 0 or 1. Thus, 4 are guaranteed to run, plus the two addi in the beginning. Then, the number of times s1 will be incremented is based on the number of 1’s. Thus, the equation to find the number of executed instructions is:

Number of instructions = 2 + (4 * number of bits) + (number of 1’s).

0xFF00FF00 in binary is: $11111111000000001111111100000000_2$ and with the equation: Number of instructions = 2 + (4 * 32) + 16 which equals 146.

b)

```
        .globl  main

        .text
main:   addi    s1, x0, 0       # initialize s1 to 0
        addi    t1, x0, 31      # tracks the bits to shift

loop:   srl     t0, s0, t1      # shift content of t0 t1 bits to right
        andi    t0, t0, 1       # mask to isolate bit
        beq     t0, x0, skip    # if the bit is 0, do not increment s1
        addi    s1, s1, 1       # increment the counter

skip:   addi    t1, t1, -1      # decrement by 1
        bge     t1, x0, loop    # if counter is greater than or equal to 0, then return to loop
```

The number of instructions when s0 is 0xFF00FF00 is 178 instructions. This number can be obtained by the fact that when there is a 0, the loop will run 3 instructions until it reaches the beq instruction, then it will run the two instructions at the skip label. This would then have 0s always running 5 instructions. When t0 is a 1, then all the instructions under the loop label will be run, which will be 4 instructions. Then, the instructions under the skip label will run, which will be 2. Thus, 6 instructions in total will run for 1s. Then, two instructions will run in the beginning. Thus, we can multiply the total number of digits and then add by the total number of 1s in the binary sequence and then a + 2. The equation is then the following for 0xFF00FF00:

Number of instructions = 2 + (5 * 32) + 16

4)

```
        .globl  main

        .text

# a = s1, i = s2, r = s3

loop:   bge     s2, s1, exit    # if i >= a (s2 >= s1), exit loop
        andi    t0, s2, 0xA5    # perform and instruction with s2 (i) = store into t0
        beq     t0, x0, else    # if t0 is equal to 0, move to else label
        slli    t2, s2, 8       # bit shift to the left by 8, and store at t2
        xor     s3, s3, t2      # r ^= (i << 8)

        addi    s2, s2, 1       # increment s2 by 1
        beq     x0, x0, loop    # return to beginning of loop

else:   srli    t1, s2, 4       # shift bit to the right by 4
        add     s3, s3, t1      # add t1 to s3 (s3 += t1)

        addi    s2, s2, 1       # increment s2 by 1
        beq     x0, x0, loop    # return to beginning of loop

```

5)

```
        .globl  main

        .text

main:   addi    a7, x0, 5       # syscall for taking in an integer as input
        ecall

        addi    s1, a0, 0       # s1 = input() which was stored in a0
        addi    t0, x0, 1       # t0 = 1 for comparison in loop
        addi    s2, x0, 0       # counter for number of times function runs

loop:   beq     s1, t0, exit    # if s1 is equal to 1, then exit the loop
        andi    t1, s1, 1       # checks to see if the final bit of s1 is a 1 where 1 means its odd and 0 means even
        beq     t1, x0, even    # if it is a 0, go to even label

        addi    t2, s1, 0       # create a copy of s1
        slli    s1, s1, 1       # bit shift to the left by 1 in order to multiply by 2 - then add t2
        add     s1, s1, t2      # s1 += t2 = 3n
        addi    s1, s1, 1       # add by 1 (3n + 1)

        addi    s2, s2, 1       # incrememnt counter by 1
        beq     x0, x0, loop    # return to beginning of loop


even:   srli    s1, s1, 1       # shift right by 1 bit = divides by 2
        addi    s2, s2, 1       # incrememnt s2 counter by 1
        beq     x0, x0, loop    # return back to the loop

exit:   addi    a7, x0, 1       # syscall for printing an integer
        addi    a0, s2, 0       # store s2 into a0
        ecall

        addi    a7, x0, 10      # exit program with code 0
        ecall
```