Name:

CSc 656 Quiz 1
Spring 2018

Problem 1:
Translate this C/C++ code fragment to MIPS assembly language. You are only allowed to use the MIPS instructions given in the list from your textbook, and pseudo-instructions for conditional branches. (The variable declarations are for your reference; you don't have to show the data allocation section.)

Assume &x[0] is in $xaddr, &y[0] is in $yaddr. i is in $i, j is in $j, ptr is in $ptr. Write efficient code. Obviously inefficient code will be penalized. (50 points)

```
int x[100], y[100], i, j, *ptr;
[some code not shown …]

  i=1;
  j=7;
  while (x[i] < *ptr) {
    j*=2;
    if (y[i] > j)
      ptr++;
    y[i-1] = j;
    i++;
  }
```

ANS:

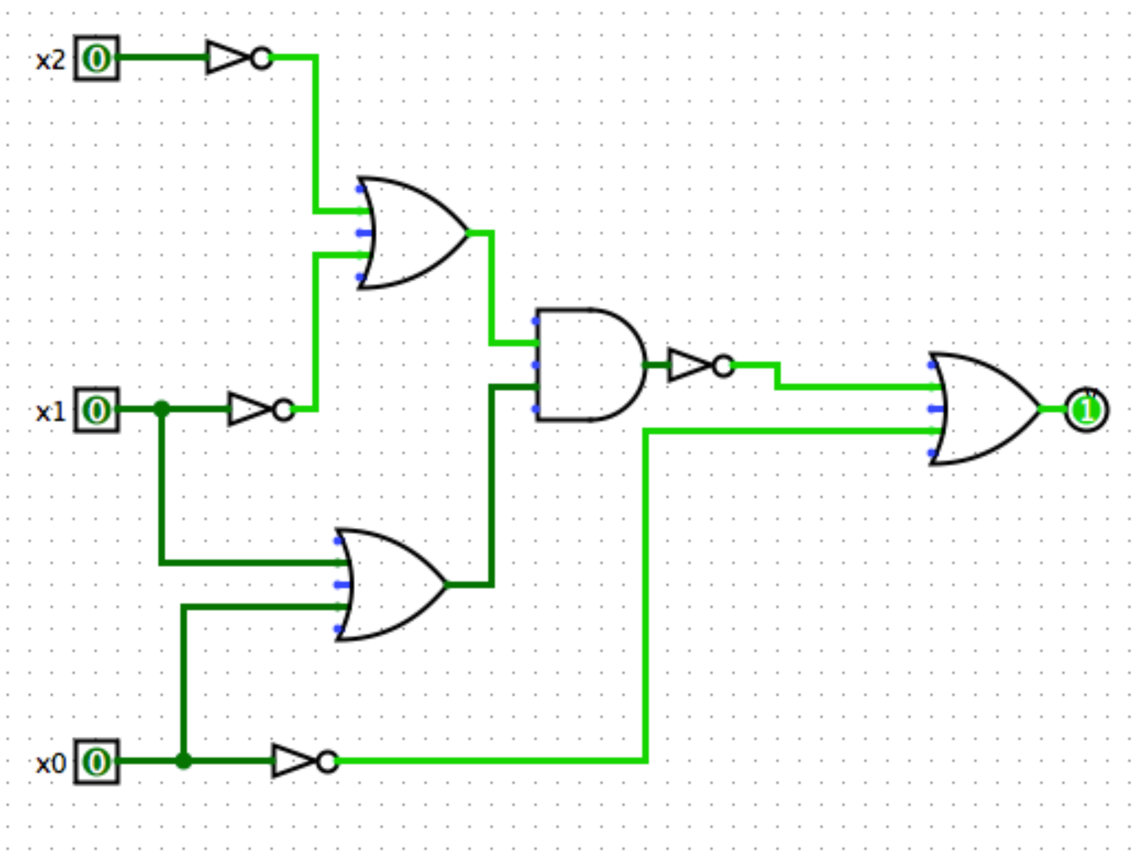
```
    addi   $i, $0, 1     #  i=1;
    addi   $j, $0, 7     #  j=7;
    sll    $t0, $i, 2    #  while (x[i] < *ptr) {
    add    $t0, $t0, $xaddr
    lw     $t0, ($t0)
    lw     $t1, ($ptr)
    bge    $t0, $t1, exit

loop:
   sll     $j, $j, 1     #    j*=2;
    sll    $t0, $i, 2    #    if (y[i] > j)
    add    $t0, $t0, $yaddr
    lw     $t1, ($t0)
    ble    $t1, $j, skip
    addi   $ptr, $ptr, 4 #      ptr++;
skip:
    sw     $j, -4($t0)    #    y[i-1] = j;
    addi   $i, $i, 1     #    i++;

    sll    $t0, $i, 2 #  }
    add    $t0, $t0, $xaddr
    lw     $t0, ($t0)
    lw     $t1, ($ptr)
    blt    $t0, $t1, loop

exit:
```

Problem 2:
Consider this digital logic circuit:



a) Fill out an equivalent truth table for the circuit.

X2	X1	X0	y
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

b) Write an equivalent logic expression for output y, in sum of products form.

ANS:

Y = ~x2~x1~x0 + ~x2x1~x0 + x2~x1~x0 + x2x1~x0 + x2x1x0

[rs, rt, rd are any registers. I is a 16-bit constant.]

```
add rd, rs, rt          rd = rs + rt
sub rd, rs, rt          rd = rs - rt
addu rd, rs, rt         rd = rs + rt (overflow ignored)
subu rd, rs, rt         rd = rs - rt (overflow ignored)
addiu rt, rs, I         rt = rs + I (sign-extended)
mult rs, rt             Hi || Lo = rs * rt
div rs, rt              Hi = rs % rt
                        Lo = rs / rt
mfhi rd                 rd = Hi
mflo rd                 rd = Lo
mthi rs                 Hi = rs
mtlo rt                 Lo = rs

and rd, rs, rt          rd = rs & rt
or rd, rs, rt           rd = rs | rt
andi rt, rs, I          rt = rs & I (zero-extended)
ori rt, rs, I           rt = rs | I (zero-extended)
sll rd, rt, I           rd = rt << I
srl rd, rt, I           rd = rt >> I

lw rt, I(rs)            rt = Mem[rs + I] (load word)
sw rt, I(rs)            Mem[rs + I] = rt (store word)

lbu rt, I(rs)           rt = mem[rs + I]
                         (load byte zero-extended)

lb rt, I(rs)            rt = mem[rs + I]
                         (load byte sign-extended)

sb rt, I(rs)            Mem[rs + I] = rt (store byte)

beq rs, rt, label       if (rs == rt) goto label
bne rs, rt, label       if (rs != rt) goto label
slt rd, rs, rt          if (rs < rt) rd = 1; else rd = 0
slti rt, rs, I          if (rs < I) rt = 1; else rd = 0

j label                 goto label
jr rs                   PC = rs
jal label               $31 = return address; goto label
```