Exercise 1

Yes, $s0 - $s7 can be used in a procedure. However, the caller function expects the values to stay the same at the end. So then you would have to save the contents on the stack first before using $s0 - $s7. When the contents are on the stack, you can use the registers as much as you want, as long as you restore the original values before returning. Found by information from the lecture and the Green Sheet.

So yes, it is possible, but it is cumbersome to do so.

Exercise 2

No, $at, $k0 and $k1 cannot be used as temporary storage. The register $at is used as an assembler temporary value, so it is reserved for the assembler. $k0 and $k1 are reserved for the OS Kernel. Hence, all of these registers cannot be used for temporary storage.

Exercise 3

```
# for(int i = 0; i < 100; i++)
#   if(A[i] != B[i]) count++;

# $S0 = i
# $S1 = count
# $S2 = A
# $S3 = B

# $t0 = 4*i
# $t1 = A[i]
# $t2 = B[i]
# $t3 stores address of A[i] or B[i]


li $s0, 0

loop:
    beq $S0, 100, exit
# load A
    sll $t0, $S0, 2
    add $t3, $t0, $s2
    lw $t1, 0($t3)
# load B
    add $t3, $t0, $s3
    lw $t2, 0($t3)

# If B != A
    bne $t1, $t2, inccount
    j skip
inccount:
    addi $s1, $s1, 1
skip:
    addi $s0, $s0, 1
    j loop
exit:
```

Exercise 4

```
# int main(){
#   int i=3, j=5, n=2, m=8;

#   A(i,j,n,m);
#   return 0;
# }

# int A(int i, int j, int n, int m){
#   return B(i,n) + B(j,m);
# }

# int B(int i, int n){
#   return i*n;
# }


# Main
main:
    li      $s0, 3      # $s0 = int i = 3
    li      $s1, 5      # $s1 = int i = 5
    li      $s2, 2      # $s2 = int i = 2
    li      $s3, 8      # $s3 = int i = 8

    # Setting arguments
    move    $a0, $s0    # $a0 = i
    move    $a1, $s1    # $a1 = j
    move    $a2, $s2    # $a2 = n
    move    $a3, $s3    # $a3 = m

    # A(i, j, n, m)
    # Stack snapshot 1
    jal A_func # Stack snapshot 2
    # Stack snapshot 9
    # Don't have to save $v0 because it is also not saved in the int main()

    j Exit

# Function A
A_func:
    # Stack snapshot 2
    # $a0 = i, $a1 = j, $a2 = n, $a3 = m

    # Store arguments j and n, because they have to be moved around for new arguments
    # m does not have to be stored, since B_func will not interact with $a3
    # $a0 = i so that is fine for first B_func call: it does not have to be stored
    move    $t0, $a1            #   $t0 = j
    move    $t1, $a2            #   $t1 = n
    # Don't know if it is handy that we don't save $a3, but in this case it works
    # I don't know MIPS coventions


    # Push $ra because of upcoming nested function call
    sub     $sp, $sp, 4
    sw      $ra, 0($sp)

    # load second argument (n) for B_func
    # $a0 = i so that is fine for first B_func call
    # B(i, n)
    move    $a1, $t1            # $a1 = n

    # Stack snapshot 3
    jal     B_func              # Stack snapshot 4
    # Stack snapshot 5
    move    $t3, $v0            # $t3 = B(i,n)

    # B(j, m)
    move    $a0, $t0            # $a0 = j
    move    $a1, $a3            # $a1 = m
    # Stack snapshot 6
    jal     B_func              # Stack snapshot 7
    # Stack snapshot 8
    move    $t4, $v0            # $t4 = $v0

    add     $v0, $t3, $t4       # $v0 = B(i,n) + B(j,m)

    # Pop $ra
    lw      $ra, 0($sp)
    addi    $sp, $sp, 4

    jr      $ra                 # jump to $ra


# Function B
B_func:
    # Stack snapshots 4 and 7
    # $a0 = i, $a1 = n
    mult    $a0, $a1            # $a0 * $a1 = Hi and Lo registers
    mflo    $v0                 # copy Lo to $v0
    # No overflow check, because how to do error checking?
    jr      $ra                 # jump to $ra

Exit:
```

Stack snapshots 1 & 2

Adress in $sp	0
	0
	0

Stack snapshots 3 - 8

	Value from $ra from jal A_func
Adress in $sp	0
	0

Stack snapshot 9

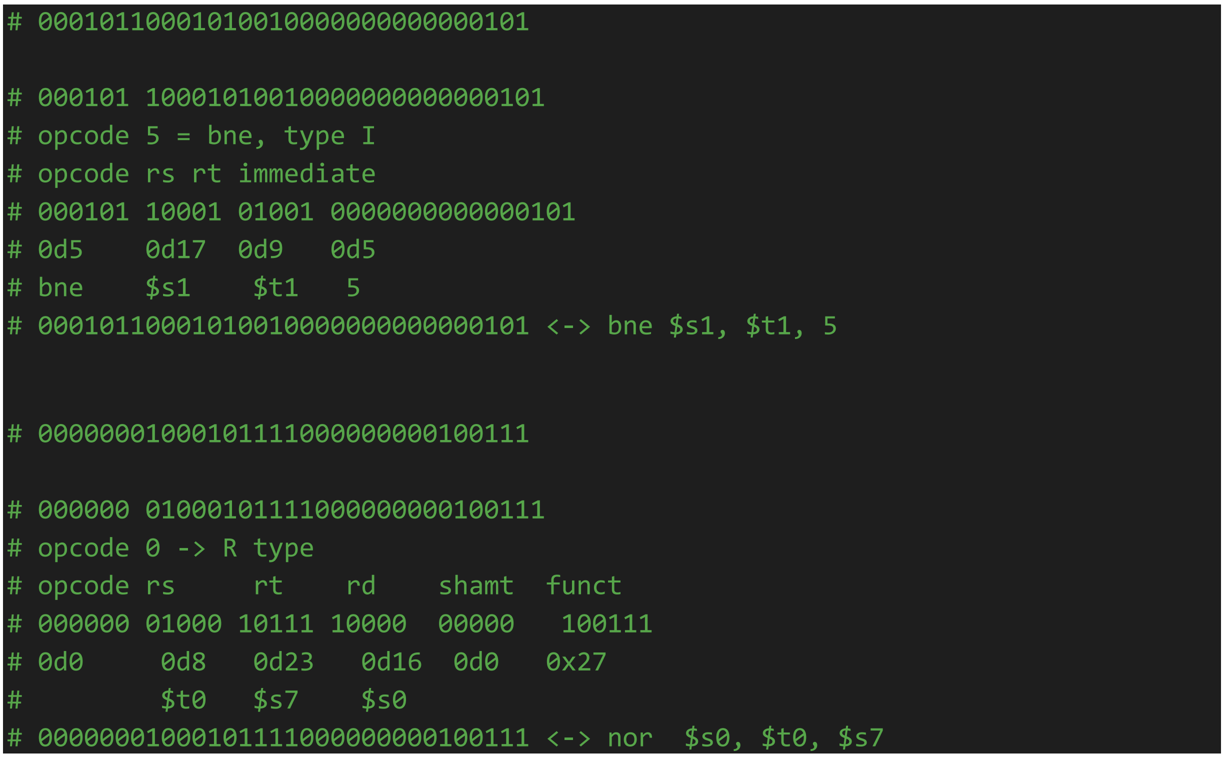
Adress in $sp	0
	0
	0

Exercise 5

```
# slt $t0, $s1, $t8
# R type: opcode rs rt rd shamt funct
# rd = $t0, rs = $s1, rd = $t8
# 0       0d17  0d24  0d8    shamt  0x2a
# 000000 10001 11000  01000  00000 101010
# 0000 0010 0011 1000 0100 0000 0010 1010
# slt $t0, $s1, $t8 <-> 0000 0010 0011 1000 0100 0000 0010 1010

# j 4660
# j 0d4660
# J type: opcode address
# 0x2 0d4660
# 000010 00000000000001001000110100
# 0000 1000 0000 0000 0001 0010 0011 0100
# j 4660 <-> 0000 1000 0000 0000 0001 0010 0011 0100
```

Exercise 6



```
# 00010110001010010000000000000101

# 000101 10001010010000000000000101
# opcode 5 = bne, type I
# opcode rs rt immediate
# 000101 10001 01001 0000000000000101
# 0d5    0d17  0d9   0d5
# bne    $s1    $t1   5
# 00010110001010010000000000000101 <-> bne $s1, $t1, 5


# 00000001000101111000000000100111

# 000000 01000101111000000000100111
# opcode 0 -> R type
# opcode rs     rt    rd    shamt  funct
# 000000 01000 10111 10000  00000   100111
# 0d0     0d8   0d23   0d16  0d0   0x27
#         $t0   $s7    $s0
# 00000001000101111000000000100111 <-> nor  $s0, $t0, $s7
```