RISC-V ECE253 ONE-PAGE CHEATSHEET

Subroutine Template (Fully Commented)

```
myfunc:
   addi sp, sp, -16 # allocate stack frame
   sw ra, 12(sp) # save return address
   sw s0, 8(sp) # save s0 (callee-saved)

   # function body here

   lw s0, 8(sp) # restore s0
   lw ra, 12(sp) # restore ra
   addi sp, sp, 16 # deallocate stack
   ret # return to caller
```

Recursive Sum n + (n-1) + ... + 1

```
sum:
   addi sp, sp, -16 # make stack frame
   sw ra, 12(sp) # save ra

   beq a0, x0, base # if n == 0, go to base case
   addi a0, a0, -1 # compute n-1
   jal ra, sum # recursive call
   add a0, a0, a1 # add returned value to n
   j done # skip base case

base:
   mv a0, x0 # return 0

done:
   lw ra, 12(sp) # restore ra
   addi sp, sp, 16 # pop frame
   ret
```

Recursive Fibonacci

```
fib:
   addi sp, sp, -32 # big frame for storage
   sw ra, 28(sp) # save ra
   sw a0, 24(sp) # save n

   li t0, 1
   ble a0, t0, base # if n <= 1 return n

   addi a0, a0, -1
   jal ra, fib # fib(n-1)
   mv t1, a0 # store fib(n-1)

   lw a0, 24(sp)
   addi a0, a0, -2
   jal ra, fib # fib(n-2)

   add a0, a0, t1 # fib(n) = f(n-1) + f(n-2)
   j done

base:
   lw a0, 24(sp) # return n

done:
   lw ra, 28(sp)
   addi sp, sp, 32
   ret
```

Interrupt Enabling

```
li t0, 0x80
csrs mie, t0 # enable machine timer interrupt

li t0, 0x8
csrs mstatus, t0 # set global MIE bit
```

Check Cause of Interrupt

```
csrr t0, mcause # bit31 = interrupt flag, lower bits=cause
```

FULLY COMMENTED POLLING EXAMPLE

```
.equ TIMER, 0xFF202000 # base addr timer
.equ TEMP, 0xFFFF0010 # temperature sensor
.equ TRANSMIT, 0xFFFF0080 # transmitter base

_start:
   la s0, TIMER # s0 = timer base address

   li t0, 50000000 # 0.5 seconds at 100MHz
   slli t1, t0, 16 # move low half up (prep masking)
   srli t1, t0, 16 # extract low 16 bits
   sw t1, 8(s0) # store low start count

   srli t0, t0, 16 # extract high 16 bits
   sw t0, 12(s0) # store high start count

   li t0, 0x6 # START + CONT bits = 0110
   sw t0, 4(s0) # write control register

polling_loop:
   lw t0, 0(s0) # read status register
   and t0, t0, 0x1 # check timeout bit
   beqz t0, polling_loop # wait until done

   sw zero, 0(s0) # acknowledge timer timeout

   la s1, TEMP
   lw t0, 0(s1) # read temperature

   la s2, TRANSMIT # transmitter base

polling_transmit:
   lw s3, 0(s2) # read status (ready bit)
   and s3, s3, 0x1 # isolate ready bit
   beqz s3, polling_transmit# wait until ready

   sw t0, 4(s2) # write temperature to transmit reg
   j polling_loop # repeat forever
```