Category A: Counters & Timing Logic

These blocks handle counting up/down and managing time delays.

1. The Standard "Forever" Up-Counter (0x00 to 0xFF) The base code for Lab 3-C. Counts up continuously on LEDs.

Code snippet

```
  LDI R20, 0x00        ; Initialize Counter
LOOP_UP:
   OUT PORTB, R20      ; Display Count
   RCALL DELAY         ; Wait for visibility
   INC R20             ; Increment
   RJMP LOOP_UP        ; Repeat forever (overflows automatically)
```

2. Bounded Counter (0 to Limit, then Reset)

Counts 0 to 9 (or any fixed number), then restarts.

Code snippet

```
  LDI R20, 0x00        ; Start at 0
   LDI R21, 10         ; Set Limit (e.g., 10 for Decimal)
LOOP_LIMIT:
   OUT PORTB, R20      ; Display
   RCALL DELAY
   INC R20
   CP R20, R21         ; Compare Counter with Limit
   BRNE LOOP_LIMIT     ; If Not Equal, keep looping
   CLR R20             ; If Equal, Reset to 0
   RJMP LOOP_LIMIT
```

3. Dynamic Counter (Limit set by Switches)

Reads PORTC switches to decide when to reset.

Code snippet

```
LOOP_DYNAMIC:
   IN R19, PINC        ; Read Limit from Switches
   OUT PORTB, R20      ; Display Counter
   RCALL DELAY
   INC R20
   CP R20, R19         ; Compare Counter vs Switch Value
   BRSH RESET_CTR      ; If Counter >= Switch, Reset
```

```
   RJMP LOOP_DYNAMIC
RESET_CTR:
   CLR R20
   RJMP LOOP_DYNAMIC
```

4. The "Smart" Up/Down Counter (Scenario 1) *Counts UP if Switch 0 is OFF, DOWN if Switch 0 is ON.*

Code snippet

```
  LDI R20, 0          ; Init Counter
LOOP_SMART:
   OUT PORTB, R20
   RCALL DELAY
   SBIC PINC, 0        ; Check Switch 0
   RJMP GO_DOWN        ; If Set (Logic 1), Jump to Down
   INC R20             ; Else, Increment (Up)
   RJMP LOOP_SMART
GO_DOWN:
   DEC R20             ; Decrement (Down)
   RJMP LOOP_SMART
```

5. Standard Nested Delay Subroutine (1 Second approx)

The massive delay loop required to make LEDs visible.

Code snippet

```
DELAY:
   LDI R21, 50         ; Outer Loop (Adjust for total time)
D1: LDI R22, 255       ; Middle Loop
D2: LDI R23, 255       ; Inner Loop
D3: DEC R23
   BRNE D3             ; Loop Inner
   DEC R22
   BRNE D2             ; Loop Middle
   DEC R21
   BRNE D1             ; Loop Outer
   RET
```

6. Variable Speed Delay (Fast/Slow) *Checks a switch to decide delay length (Scenario 2).*

Code snippet

```
DELAY_VAR:
```

```
   SBIC PINC, 0          ; Check Switch 0
   RJMP SET_FAST         ; If pressed, go Fast
   LDI R21, 100          ; Load Slow Count
   RJMP DO_DELAY
SET_FAST:
   LDI R21, 10           ; Load Fast Count
DO_DELAY:
   ; (Insert standard Inner/Middle loops here using R21)
   LDI R22, 255
D2_VAR: DEC R22
   BRNE D2_VAR
   DEC R21
   BRNE DO_DELAY
   RET
```

7. One-Shot Button Press (Anti-Hold) *Increments ONLY ONCE per press (waits for release). Vital if Ajarn asks "Count button presses".*

Code snippet

```
WAIT_PRESS:
   SBIC PINC, 0          ; Wait for Press (Active Low)
   RJMP WAIT_PRESS

   INC R20               ; Action: Increment Counter
   OUT PORTB, R20

WAIT_RELEASE:
   SBIS PINC, 0          ; Wait for Release
   RJMP WAIT_RELEASE
   RCALL DELAY           ; Small debounce delay
   RJMP WAIT_PRESS
```

Category B: Arithmetic & Logic

These blocks solve math formulas and logic problems.

8. Basic Adder: (Input A + Input B)

Reads two ports, adds them, displays result.

Code snippet

```
MAIN_ADD:
```

```
    IN R16, PINC          ; Read Value A
    IN R17, PIND          ; Read Value B
    ADD R16, R17          ; R16 = A + B
    OUT PORTB, R16        ; Display Result
    RJMP MAIN_ADD
```

9. The Formula: `(PORTC + 4) * PORTD`

The exact Lab 3-D Activity 1 problem.

Code snippet

```
MAIN_FORMULA:
    IN R18, PINC          ; Read C
    LDI R20, 4            ; Load Constant 4
    ADD R18, R20          ; R18 = C + 4
    IN R19, PIND          ; Read D
    MUL R18, R19          ; Multiply: Result in R1:R0
    MOV R16, R0           ; Move Low Byte result
    OUT PORTB, R16        ; Display Low Byte
    RJMP MAIN_FORMULA
```

10. The Average: `(PORTB + PORTD) / 2`

Uses logical shift right (LSR) to divide.

Code snippet

```
MAIN_AVG:
    IN R18, PINB          ; Read B
    IN R19, PIND          ; Read D
    ADD R18, R19          ; Sum = B + D
    LSR R18               ; Divide by 2 (Bitwise Shift)
    OUT PORTC, R18        ; Display Average
    RJMP MAIN_AVG
```

11. 16-Bit Addition (Big Numbers)

Adding two 16-bit numbers (Low+Low, High+High+Carry).

Code snippet

```
ADD_16:
    ; Num1 in R17:R16, Num2 in R19:R18
    ADD R16, R18          ; Add Low Bytes
```

```
   ADC R17, R19          ; Add High Bytes + Carry
   ; Result in R17:R16
   RET
```

12. Logic Selector (The "Scenario 4" Logic) *Performs ADD if Switch=0, AND if Switch=1.*

```
Code snippet
LOGIC_SEL:
   IN R16, PINC          ; Read A
   IN R17, PIND          ; Read B
   SBIC PINE, 0          ; Check Control Switch
   RJMP DO_AND           ; If 1, do AND

   ADD R16, R17          ; Default: ADD
   RJMP DISP_RES
DO_AND:
   AND R16, R17          ; Alternate: AND
DISP_RES:
   OUT PORTB, R16
   RJMP LOGIC_SEL
```

13. Multiplication of Array Constants

Multiply a value by 10 (common exam task).

```
Code snippet
MUL_BY_10:
   LDI R18, 10           ; Multiplier
   IN R16, PINC          ; Input
   MUL R16, R18          ; R1:R0 = Input * 10
   ; R0 holds low byte, R1 holds high byte
   OUT PORTB, R0         ; Display Lower part
   OUT PORTD, R1         ; Display Upper part
   RET
```

14. Signed Number Handling (Negative Check) *Checks if a result is negative (Bit 7 is 1) and lights an error LED.*

```
Code snippet
CHECK_NEG:
   IN R16, PINC          ; Get Number
   SBRC R16, 7           ; Skip if Bit 7 (Sign Bit) is Cleared
```

```
   SBI PORTB, 0          ; Turn on "Negative Indicator" LED
   SBRS R16, 7
   CBI PORTB, 0          ; Turn off LED if positive
   RET
```

Category C: Memory & Data Processing

These blocks use Z, X, Y pointers to handle arrays (Lab 3-D Activity 4).

15. Summation of Array (Flash Memory)

Adds 10 numbers stored in Program Memory.

Code snippet

```
SUM_ARRAY:
   LDI ZL, LOW(MYDATA*2)  ; Setup Z Pointer Low
   LDI ZH, HIGH(MYDATA*2) ; Setup Z Pointer High
   CLR R20                ; Clear Result Register (Sum)
   LDI R21, 10            ; Loop Counter (10 Items)
LOOP_SUM:
   LPM R16, Z+            ; Load Byte, Inc Pointer
   ADD R20, R16           ; Add to Sum
   DEC R21
   BRNE LOOP_SUM
   OUT PORTB, R20         ; Display Total
   RET
```

16. Find Maximum Value in Array (Flash) *Scans memory to find the largest number (Scenario 3).*

Code snippet

```
FIND_MAX:
   LDI ZL, LOW(MYDATA*2)
   LDI ZH, HIGH(MYDATA*2)
   LDI R21, 10            ; Count
   CLR R20                ; R20 stores Current Max (Start 0)
LOOP_MAX:
   LPM R16, Z+
   CP R20, R16            ; Compare Max vs New
   BRSH SKIP_UPD          ; If Max >= New, Skip
   MOV R20, R16           ; Else, Update Max
SKIP_UPD:
```

```
    DEC R21
    BRNE LOOP_MAX
    OUT PORTB, R20        ; Display Max
    RET
```

17. Find Minimum Value in Array *Same as Max, but logic reversed (`BRLO` instead of `BRSH`).*

```
Code snippet
FIND_MIN:
    LDI R20, 0xFF         ; Start Min at Max possible value (255)
    ; (Setup Z pointer here...)
LOOP_MIN:
    LPM R16, Z+
    CP R16, R20           ; Compare New vs Current Min
    BRSH SKIP_MIN         ; If New >= Min, Skip
    MOV R20, R16          ; Else, Update Min
SKIP_MIN:
    DEC R21
    BRNE LOOP_MIN
    OUT PORTB, R20
    RET
```

18. Copy Flash Data to SRAM

Moves data from Program Memory (Source) to SRAM (Dest).

```
Code snippet
COPY_MEM:
    LDI ZL, LOW(MYDATA*2)  ; Source (Flash)
    LDI ZH, HIGH(MYDATA*2)
    LDI XL, 0x00          ; Dest (SRAM 0x0100)
    LDI XH, 0x01
    LDI R21, 10           ; Count
LOOP_COPY:
    LPM R16, Z+           ; Read Flash
    ST X+, R16            ; Write SRAM
    DEC R21
    BRNE LOOP_COPY
    RET
```

19. Clear SRAM (Fill with Zeros) *Loops through an SRAM area and zeros it out.*

Code snippet

```
CLEAR_RAM:
   LDI XL, 0x00
   LDI XH, 0x01          ; Start at $0100
   LDI R21, 50           ; Clear 50 bytes
   CLR R16               ; Zero value
LOOP_CLR:
   ST X+, R16            ; Store 0, Inc Pointer
   DEC R21
   BRNE LOOP_CLR
   RET
```

20. Count Occurrences (How many 5s?) *Counts how many times a specific number appears in an array.*

Code snippet

```
COUNT_HITS:
   ; (Setup Z Pointer...)
   LDI R22, 5            ; Target Number to find
   CLR R20               ; Hit Counter
   LDI R21, 10           ; Array Size
LOOP_FIND:
   LPM R16, Z+
   CP R16, R22           ; Compare Data vs Target
   BRNE SKIP_INC         ; If Not Equal, Skip
   INC R20               ; Else, Count++
SKIP_INC:
   DEC R21
   BRNE LOOP_FIND
   OUT PORTB, R20        ; Show total count
   RET
```