Group A: Basic Math Formulas (Lab 3-D Logic)

These are the direct "Calculate X and output to Y" problems.

1. Basic Addition: `Output = Input A + Input B` *Reads two ports, adds them, and displays the result.*

Code snippet

```
CALC_ADD:
   IN R16, PINB         ; Read Input A
   IN R17, PINC         ; Read Input B
   ADD R16, R17          ; R16 = A + B
   OUT PORTD, R16        ; Display Result
   RJMP CALC_ADD
```

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

Specific problem from Lab 3-D Activity 1 .

Code snippet

```
CALC_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 (R0) to Output Reg
   OUT PORTB, R16        ; Display Low Byte
   RJMP CALC_FORMULA
```

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

Specific problem from Lab 3-D Activity 2 .

Code snippet

```
CALC_AVG:
   IN R18, PINB         ; Read B
   IN R19, PIND         ; Read D
   ADD R18, R19          ; Sum = B + D
```

```
    LSR R18             ; Logical Shift Right (Divides by 2)
    OUT PORTC, R18      ; Output Average
    RJMP CALC_AVG
```

4. Subtraction with Borrow Check *Calculates `A - B`. If result is negative, lights an error LED.*

```
Code snippet
CALC_SUB:
    IN R16, PINB        ; Load A
    IN R17, PINC        ; Load B
    SUB R16, R17        ; A - B

    BRMI IS_NEGATIVE    ; Branch if Minus Flag (N) is set
    OUT PORTD, R16      ; Display Result
    RJMP CALC_SUB

IS_NEGATIVE:
    SBI PORTD, 7        ; Turn on Error LED (Bit 7)
    RJMP CALC_SUB
```

Group B: Advanced Arithmetic (16-bit & Multiplication)

5. 16-Bit Addition (Low/High Byte)

Adds two 16-bit numbers using `ADC` (Add with Carry).

```
Code snippet
ADD_16BIT:
    ; Num1 = R17:R16 (High:Low)
    ; Num2 = R19:R18 (High:Low)

    ADD R16, R18        ; Add Low Bytes
    ADC R17, R19        ; Add High Bytes + Carry from Low

    ; Result is now in R17:R16
    RET
```

6. Multiply and Store (16-bit Result) *Multiplication always produces a 16-bit result in R1:R0. You must save both.*

```
Code snippet
```

```
DO_MUL:
   IN R16, PINB          ; Input 1
   LDI R17, 10           ; Input 2 (Constant 10)
   MUL R16, R17          ; Result in R1:R0

   MOV R20, R0           ; Save Low Byte
   MOV R21, R1           ; Save High Byte
   RET
```

7. Division by 4 (Two Shifts) *Shifting right once divides by 2. Shifting twice divides by 4.*

Code snippet

```
DIV_BY_4:
   IN R16, PINB
   LSR R16               ; / 2
   LSR R16               ; / 4
   OUT PORTD, R16
   RET
```

8. Multiplication by 2 (Left Shift) *Shifting Left (`LSL`) multiplies by 2. Cheaper than `MUL`.*

Code snippet

```
MUL_BY_2:
   IN R16, PINB
   LSL R16               ; R16 = R16 * 2
   OUT PORTD, R16
   RET
```

Group C: Program Memory (Flash) Access

Reading constant data arrays using the Z-Pointer (`LPM`).

9. Read Array from Flash (Basic Loop)

Standard setup to read a list of constants.

Code snippet

```
READ_FLASH:
   ; 1. Setup Z Pointer (Must scale by 2 for Word Address)
   LDI ZL, LOW(MYDATA*2)
   LDI ZH, HIGH(MYDATA*2)
```

```
    LDI R21, 10          ; Loop 10 times

LOOP_READ:
    LPM R16, Z+          ; Load Byte from Flash to R16, Increment Z
    OUT PORTB, R16       ; Display Data
    DEC R21
    BRNE LOOP_READ
    RET

MYDATA: .DB 10, 20, 30, 40, 50, 60, 70, 80, 90, 100
```

10. Summation of Flash Array

Adds all numbers in a stored array.

Code snippet

```
SUM_FLASH:
    LDI ZL, LOW(MYDATA*2)
    LDI ZH, HIGH(MYDATA*2)
    CLR R20              ; Sum Register (Start 0)
    LDI R21, 10          ; Counter

LOOP_SUM:
    LPM R16, Z+
    ADD R20, R16         ; Sum += Current Value
    DEC R21
    BRNE LOOP_SUM

    OUT PORTB, R20       ; Show Total
    RET
```

11. Find Maximum Value in Flash Array *Scans array to find the largest number.*

Code snippet

```
FIND_MAX:
    LDI ZL, LOW(MYDATA*2)
    LDI ZH, HIGH(MYDATA*2)
    LDI R21, 10          ; Counter
    CLR R20              ; Max Value Holder (Start 0)

LOOP_MAX:
    LPM R16, Z+          ; Read Value
    CP R20, R16          ; Compare Max vs New
```

```
    BRSH SKIP_UPDATE       ; If Max >= New, Skip
    MOV R20, R16           ; Else, New Max found

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

12. Copy Flash Data to SRAM *Moves data from Code Memory to Data Memory (e.g., to $0100).*

```
Code snippet
COPY_FLASH_RAM:
    LDI ZL, LOW(MYDATA*2)  ; Source (Flash)
    LDI ZH, HIGH(MYDATA*2)
    LDI XL, 0x00           ; Dest (SRAM $0100)
    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
```

Group D: SRAM Data Processing (X/Y Pointers)

Manipulating data in RAM.

13. Clear SRAM Block (Fill with 0) *Wipes a section of memory.*

```
Code snippet
CLEAR_RAM:
    LDI XL, 0x00           ; Start at $0100
    LDI XH, 0x01
    LDI R16, 0x00          ; Clear Value
    LDI R21, 50            ; Clear 50 bytes

LOOP_CLR:
    ST X+, R16             ; Store 0, Inc X
```

```
   DEC R21
   BRNE LOOP_CLR
   RET
```

14. Find Value in SRAM (Search) *Checks if a specific number (e.g., 5) exists in RAM.*

```
Code snippet
SEARCH_RAM:
   LDI XL, 0x00
   LDI XH, 0x01
   LDI R22, 5          ; Target Number
   LDI R21, 10         ; Size

LOOP_SEARCH:
   LD R16, X+          ; Load from RAM
   CP R16, R22         ; Compare
   BREQ FOUND_IT       ; If Equal, Jump
   DEC R21
   BRNE LOOP_SEARCH
   RET                 ; Not Found

FOUND_IT:
   SBI PORTB, 0        ; Turn on "Found" LED
   RET
```

15. Array Addition (SRAM to SRAM) *Adds two arrays stored in RAM: Array3[i] = Array1[i] + Array2[i].*

```
Code snippet
ADD_ARRAYS:
   LDI XL, 0x00        ; Array 1 at $0100
   LDI XH, 0x01
   LDI YL, 0x00        ; Array 2 at $0200
   LDI YH, 0x02
   LDI R21, 10         ; Size

LOOP_ADD_ARR:
   LD R16, X+          ; Load Array 1
   LD R17, Y           ; Load Array 2
   ADD R16, R17        ; Add
   ST Y+, R16          ; Store Result back to Array 2
   DEC R21
   BRNE LOOP_ADD_ARR
```

```
    RET
```

16. Count Occurrences *Counts how many times a number appears in an array.*

Code snippet
```
COUNT_HITS:
    LDI XL, 0x00
    LDI XH, 0x01
    LDI R22, 0xFF       ; Target (Count 255s)
    CLR R20             ; Counter
    LDI R21, 10         ; Size

LOOP_COUNT:
    LD R16, X+
    CP R16, R22
    BRNE NO_MATCH
    INC R20             ; Found one!

NO_MATCH:
    DEC R21
    BRNE LOOP_COUNT
    OUT PORTB, R20
    RET
```

Group E: Logic & BCD

17. Logic Selector (Masking)

Use this if asked to "Mask off the upper 4 bits".

Code snippet
```
LOGIC_MASK:
    IN R16, PINC
    ANDI R16, 0x0F      ; Keep Lower 4 bits only
    OUT PORTB, R16
    RET
```

18. BCD Increment (Decimal Counter) *Simulates a decimal counter (0-9) instead of Hex (0-F).*

Code snippet
```
BCD_INC:
```

```
   INC R20              ; Increment
   CPI R20, 10          ; Check if 10
   BRNE NO_RESET
   CLR R20              ; Reset to 0 if 10
NO_RESET:
   OUT PORTB, R20
   RET
```

19. ASCII to Integer Conversion *Converts character '5' (0x35) to number 5.*

Code snippet

```
ASCII_TO_INT:
   LDI R16, '5'         ; Load ASCII char
   SUBI R16, 0x30       ; Subtract '0' (0x30)
   ; R16 is now integer 5
   OUT PORTB, R16
   RET
```

20. Integer to ASCII Conversion *Converts number 5 to character '5' (0x35) for display.*

Code snippet

```
INT_TO_ASCII:
   LDI R16, 5           ; Load Integer
   SUBI R16, -0x30      ; Add '0' (Subtracting negative is adding)
   ; R16 is now '5' (0x35)
   OUT PORTB, R16
   RET
```