Group A: Initialization & Configuration

Universal setups for defining how pins behave.

1. The "Master Setup" (All Output)

Standard setup for an LED bar graph on PORTB.

Code snippet
```
RESET:
   ; 1. Init Stack (ALWAYS DO THIS)
   LDI R16, LOW(RAMEND)
   OUT SPL, R16
   LDI R16, HIGH(RAMEND)
   OUT SPH, R16

   ; 2. Configure PORTB as Output
   LDI R16, 0xFF         ; 1111 1111
   OUT DDRB, R16         ; Direction: All Output

   ; 3. Turn Off All LEDs (Initial State)
   LDI R16, 0x00
   OUT PORTB, R16
```

2. The "Master Input" (With Pull-Ups) *Safest way to read switches. Enables internal resistors so floating pins read "1".*

Code snippet
```
SETUP_INPUT:
   ; 1. Configure PORTC as Input
   LDI R16, 0x00         ; 0000 0000
   OUT DDRC, R16         ; Direction: All Input

   ; 2. Enable Internal Pull-Up Resistors
   LDI R16, 0xFF         ; Write 1s to the PORT register of an Input
   OUT PORTC, R16        ; Now inputs default to 1 (High) when open
```

3. The "Split Port" (4 In / 4 Out) *Common exam constraint: "Use PC0-3 for Switches, PC4-7 for LEDs".*

Code snippet
```
SETUP_SPLIT:
```

```
   LDI R16, 0xF0        ; 1111 0000 (Upper=Out, Lower=In)
   OUT DDRC, R16        ; Configure Direction

   ; Optional: Enable Pull-ups on Lower 4 bits
   LDI R16, 0x0F        ; 0000 1111
   OUT PORTC, R16       ; Pull-ups active on PC0-PC3
```

4. The "Safe Reset" (Clear All) *Wipes all ports to a known safe state (Input, No Pull-ups).*

Code snippet
```
SAFE_RESET:
   LDI R16, 0x00
   OUT DDRB, R16        ; B = Input
   OUT PORTB, R16       ; B = Tri-state (Hi-Z)
   OUT DDRC, R16        ; C = Input
   OUT PORTC, R16       ; C = Tri-state
   OUT DDRD, R16        ; D = Input
   OUT PORTD, R16       ; D = Tri-state
```

Group B: Basic Input/Output Loops

Directly connecting inputs to outputs logic.

5. The "Copycat" (Switch -> LED)

Reads PORTC switches and instantly shows them on PORTB LEDs.

Code snippet
```
MAIN_COPY:
   IN R16, PINC         ; Read Switches
   OUT PORTB, R16       ; Update LEDs
   RJMP MAIN_COPY       ; Repeat forever
```

6. The "Inverter" (NOT Logic) *Shows the reverse: Switch ON (0) -> LED ON (1).*

Code snippet
```
MAIN_NOT:
   IN R16, PINC         ; Read Switches
   COM R16              ; Complement (Invert all bits)
   OUT PORTB, R16       ; Update LEDs
   RJMP MAIN_NOT
```

7. The "Nibble Masker" (Lower 4 Only)

Reads switches but forces the upper 4 LEDs to stay OFF.

Code snippet
```
MAIN_MASK:
   IN R16, PINC          ; Read all 8 bits
   ANDI R16, 0x0F        ; Mask: 0000 1111 (Clear Upper Nibble)
   OUT PORTB, R16        ; Display Result
   RJMP MAIN_MASK
```

8. The "Nibble Swapper" *Reads Switches 0-3 and lights up LEDs 4-7.*

Code snippet
```
MAIN_SWAP:
   IN R16, PINC          ; Read Input
   SWAP R16              ; Swap High/Low Nibbles (0x0F -> 0xF0)
   OUT PORTB, R16        ; Display on opposite side
   RJMP MAIN_SWAP
```

Group C: Polling & Decisions (SBIC/SBIS)

Waiting for user events.

9. Wait for Button Press (Active Low)

Blocks program until Switch 0 is pressed (GND).

Code snippet
```
WAIT_PRESS_LOW:
   SBIC PINC, 0          ; Skip next line if PC0 is CLEARED (0)
   RJMP WAIT_PRESS_LOW    ; Jump back if PC0 is SET (1)
   ; Code continues here only after press...
```

10. Wait for Button Press (Active High) *Blocks program until Switch 0 is pressed (VCC).*

Code snippet
```
WAIT_PRESS_HIGH:
   SBIS PINC, 0          ; Skip next line if PC0 is SET (1)
   RJMP WAIT_PRESS_HIGH   ; Jump back if PC0 is CLEARED (0)
```

```
    ; Code continues here only after press...
```

11. Wait for Button Release (Anti-Hold) *Ensures one action per press. Wait for Press -> Wait for Release.*

Code snippet
```
ONE_SHOT:
    ; 1. Wait for Press
    SBIC PINC, 0
    RJMP ONE_SHOT

    ; 2. Perform Action (e.g., Increment)
    INC R20
    OUT PORTB, R20

    ; 3. Wait for Release
WAIT_REL:
    SBIS PINC, 0
    RJMP WAIT_REL

    RJMP ONE_SHOT          ; Go back to start
```

12. Full Software Debounce *Filters out mechanical switch noise using a delay.*

Code snippet
```
DEBOUNCE_READ:
    SBIC PINC, 0          ; Check for Press
    RJMP DEBOUNCE_READ    ; If not pressed, keep checking

    RCALL DELAY           ; Wait 20ms (Noise passes)

    SBIC PINC, 0          ; Check AGAIN
    RJMP DEBOUNCE_READ    ; If it was just noise, go back

    ; Valid Press Confirmed:
    INC R20
    OUT PORTB, R20
```

Group D: Logic Gates (Hardware Simulation)

Combining multiple inputs.

13. The "Security Gate" (AND Logic) *LED turns ON only if Switch 0 AND Switch 1 are pressed.*

Code snippet
```
LOGIC_AND:
   IN R16, PINC          ; Read Inputs
   MOV R17, R16          ; Copy to Temp
   ANDI R17, 0x03        ; Isolate Switch 0 and 1 (0000 0011)
   CPI R17, 0x00         ; Are both 0? (Active Low Press)
   BREQ ACCESS_GRANTED

   CBI PORTB, 0          ; Else: Access Denied (LED Off)
   RJMP LOGIC_AND

ACCESS_GRANTED:
   SBI PORTB, 0          ; LED On
   RJMP LOGIC_AND
```

14. The "Emergency Switch" (OR Logic) *LED turns ON if Switch 0 OR Switch 1 is pressed.*

Code snippet
```
LOGIC_OR:
   IN R16, PINC
   ; Check Sw0
   SBRS R16, 0           ; Skip if Bit 0 is Set (Not Pressed)
   RJMP ALARM_ON         ; If Cleared (Pressed), Alarm

   ; Check Sw1
   SBRS R16, 1
   RJMP ALARM_ON

   CBI PORTB, 0          ; No Alarm
   RJMP LOGIC_OR

ALARM_ON:
   SBI PORTB, 0          ; Alarm!
   RJMP LOGIC_OR
```

15. The "Hallway Light" (XOR Logic) *LED state toggles if EITHER switch changes position.*

Code snippet
```
LOGIC_XOR:
   IN R16, PINC          ; Read Port C (Switch 1)
```

```
   IN R17, PIND          ; Read Port D (Switch 2)
   EOR R16, R17          ; XOR the two ports
   OUT PORTB, R16        ; Result on LED
   RJMP LOGIC_XOR
```

16. Threshold Comparator *Lights LED if Input Number > 100.*

Code snippet

```
COMPARE_VAL:
   IN R16, PINC          ; Read DIP Switches (Number)
   CPI R16, 100          ; Compare with 100
   BRSH LIGHT_ON         ; Branch if Same or Higher

   CBI PORTB, 0          ; LED Off
   RJMP COMPARE_VAL

LIGHT_ON:
   SBI PORTB, 0          ; LED On
   RJMP COMPARE_VAL
```

Group E: Advanced Output Patterns

Sequencing and Animations.

17. Basic Toggle (Blinker)

Flashes all LEDs On/Off forever.

Code snippet

```
BLINK_ALL:
   LDI R16, 0xFF
   OUT PORTB, R16        ; ON
   RCALL DELAY
   LDI R16, 0x00
   OUT PORTB, R16        ; OFF
   RCALL DELAY
   RJMP BLINK_ALL
```

18. LED Chaser (Shift Left) *Moving dot: 00000001 -> 00000010 -> ...*

Code snippet

```
CHASER_LEFT:
   LDI R16, 0x01         ; Start Bit
LOOP_L:
   OUT PORTB, R16        ; Display
   RCALL DELAY
   LSL R16               ; Logic Shift Left
   BRCC LOOP_L           ; If no Carry (didn't fall off edge), Loop
   RJMP CHASER_LEFT      ; Reset to 0x01
```

19. LED Chaser (Shift Right) *Moving dot: 10000000 -> 01000000 -> ...*

Code snippet

```
CHASER_RIGHT:
   LDI R16, 0x80         ; Start Bit (Bit 7)
LOOP_R:
   OUT PORTB, R16
   RCALL DELAY
   LSR R16               ; Logic Shift Right
   BRCC LOOP_R
   RJMP CHASER_RIGHT
```

20. Traffic Light Sequence (Red -> Green -> Yellow) *Simple State Machine using individual bits.*

Code snippet

```
TRAFFIC_LIGHT:
   ; RED (Bit 0)
   LDI R16, 0x01
   OUT PORTB, R16
   RCALL LONG_DELAY

   ; GREEN (Bit 1)
   LDI R16, 0x02
   OUT PORTB, R16
   RCALL LONG_DELAY

   ; YELLOW (Bit 2)
   LDI R16, 0x04
   OUT PORTB, R16
   RCALL SHORT_DELAY

   RJMP TRAFFIC_LIGHT
```