Group A: Stack Initialization (Crucial)

1. Standard Stack Initialization (RAMEND) *Most common start for any program. Sets SP to the very end of SRAM ($08FF).*

Code snippet

```
LDI R16, LOW(RAMEND)   ; Load low byte of RAMEND address
OUT SPL, R16           ; Set Stack Pointer Low
LDI R16, HIGH(RAMEND)  ; Load high byte of RAMEND address
OUT SPH, R16           ; Set Stack Pointer High
```

2. Custom Stack Address Initialization

Required if the exam asks "Set SP to $29D" .

Code snippet

```
LDI R16, LOW(0x29D)    ; Load Low byte of target address
OUT SPL, R16
LDI R16, HIGH(0x29D)   ; Load High byte of target address
OUT SPH, R16
```

3. Re-setting Stack Pointer (Reset) *Useful if you need to "clear" the stack manually inside a loop.*

Code snippet

```
CLI                ; Disable interrupts (safety)
LDI R16, LOW(RAMEND)
OUT SPL, R16
LDI R16, HIGH(RAMEND)
OUT SPH, R16
SEI                ; Re-enable interrupts
```

4. Copy Stack Pointer to Index Register (Y) *Use this if you need to read what is currently on the stack without Popping.*

Code snippet

```
IN R28, SPL          ; Copy SP Low to Y Low (R28)
IN R29, SPH          ; Copy SP High to Y High (R29)
; Now Y points to the top of the stack
```

Group B: Basic PUSH and POP

5. Basic Push (Single Register)

Saves a register and decrements SP.

+1
Code snippet

```
PUSH R20          ; Store R20 on stack, SP = SP - 1
```

6. Basic Pop (Single Register) Restores a register and increments SP. **Must match PUSH order in reverse**.

+1
Code snippet

```
POP R20           ; Restore R20 from stack, SP = SP + 1
```

7. Storing Immediate Value to Stack *You cannot PUSH a number directly; you must load it to a register first.*

Code snippet

```
LDI R16, 0x55     ; Load value to temp register
PUSH R16          ; Push onto stack
```

8. Discarding Top of Stack *If you Pushed something but don't need it anymore (e.g., cleaning up).*

Code snippet

```
POP R0            ; Pop into a "trash" register (R0 is often safe)
```

Group C: Context Saving (Subroutines/Interrupts)

9. Push All Working Registers (Context Save) *Use this at the start of a complex subroutine.*

Code snippet

```
PUSH R16
PUSH R17
PUSH R18
PUSH R19
```

10. Pop All Working Registers (Context Restore) *Use at the end.* ***Strict Reverse Order!***

Code snippet

```
POP R19
POP R18
POP R17
POP R16
```

11. Saving the Status Register (SREG) *CRITICAL for math/logic exams. Preserves flags (Carry, Zero) across calls.*

Code snippet

```
IN R0, SREG          ; Read Status Register into R0
PUSH R0             ; Save SREG on stack
```

12. Restoring the Status Register (SREG)

Code snippet

```
POP R0              ; Recover SREG value
OUT SREG, R0          ; Write back to Status Register
```

13. Saving a 16-bit Pointer (Z Register) *Saving ZL (R30) and ZH (R31).*

Code snippet

```
PUSH R30             ; Save Low byte
PUSH R31             ; Save High byte
```

14. Restoring a 16-bit Pointer (Z Register) *Reverse order.*

Code snippet

```
POP R31             ; Restore High byte
POP R30             ; Restore Low byte
```

Group D: SRAM & "Activity 2" Tricks

15. Store Direct to SRAM (STS)

Writing to specific memory address without using Stack.

Code snippet

```
LDI R16, 0xFF         ; Load value
STS 0x0100, R16        ; Write 0xFF to SRAM address $0100
```

16. Load Direct from SRAM (LDS) *Reading from a specific memory address.*

Code snippet

```
LDS R17, 0x0100        ; Read value from SRAM address $0100 into R17
```

17. "The Activity 2 Trick": Fill SRAM then POP This simulates a pre-filled stack. You write to memory, move SP there, then POP .

Code snippet

```
; 1. Write data to SRAM manually
LDI R16, 0xAA
STS 0x029D, R16        ; Put 0xAA at $29D

; 2. Set Stack Pointer to point BELOW that data
LDI R16, LOW(0x29C)    ; SP points to $29C
OUT SPL, R16
LDI R16, HIGH(0x29C)
OUT SPH, R16

; 3. POP (SP increments to $29D, reads 0xAA)
POP R20                ; R20 becomes 0xAA
```

Group E: Advanced Pointer Access (X, Y, Z)

18. Store Indirect with Post-Increment (ST X+)

Great for filling arrays in SRAM (Activity 6).

Code snippet

```
LDI XL, 0x00          ; Setup X Pointer ($0100)
LDI XH, 0x01
LDI R16, 0x05          ; Data
ST X+, R16            ; Store 0x05 at $0100, X becomes $0101
```

19. Load Indirect with Pre-Decrement (LD -Y) *Simulates a "PUSH" operation using the Y pointer.*

Code snippet

```
LDI R16, 0x10          ; Data
ST -Y, R16            ; Decrement Y, then Store (Push simulation)
```

20. Swap Two Registers using Stack *Swap R16 and R17 without using a third temp register.*

Code snippet

```
PUSH R16              ; Stack: [R16]
PUSH R17              ; Stack: [R16, R17]
POP R16              ; R16 gets old R17 value
POP R17              ; R17 gets old R16 value
```