5. The Stack

The Stack is a memory section of the SRAM that follows First-In-First-Out (FIFO) principles. It is generally used to store temporary data for quick and easy access.

The top of the stack is tracked by the 16 bit (adjusting to memory addressing) **Stack Pointer** that stores the address. It increments and decrements accordingly.

rjhcoding.com - Stack pointer

The stack grows downwards meaning that newer data is placed on lower memory address.

> The top, is in fact, the bottom

The stack is used during subroutine (think of functions) calling to store the previous program counter when called with `RCALL` or `ICALL`. The program then restores the program counter to return to the previous location with `RET`.

Aside from subroutines, you can utilize the stack for your own use. The following are instructions that affects the stack.

Instruction	Stack Pointer	Description
PUSH Rd	SP = SP - 1	Value in Rd is pushed onto the top of the stack.
POP Rr	SP = SP + 1	The top of the stack is popped into Rr
RCALL ICALL	SP = SP - 2	PC is stored onto the stack. Program jumps to the subroutine.
RET RETI	SP = SP + 2	The top of the stack is popped back into PC. The program returns to the caller address.

Push Operation

rjhcoding.com - Push

Pop Operation

rjhcoding.com - Pop

Using Stacks

A very useful use case for stacks is as temporary storage without using up temporary registers to reduce mental overhead. For example, you might swap the contents of two registers `R16` and `R17`.

```
PUSH R16        ; temporarily store R16
MOV  R16, R17   ; overwrite R16 with R17
POP  R17        ; return the temporary data to R17
```

This form of temporary container may become handy as your program grows. Even though AVR provides 32 general purpose registers, keeping track each and every single one of them may become harder as your program grows with more subroutines. One subroutine might unintentionally modify a register that you were using during its execution which might produce unexpected results.

For example, given a subroutine that interact with USART using R24

```
SER_send_byte:
  LDS   R24, UCSR0A     ; R24 is filled with the content of UCSR0A
  SBRS  R24, UDRE0
  ...
  RET
```

Calling it from main would unexpectedly modify R24

```
main:
  LDI   R24, 0b00001010
  ...
  RCALL SER_send_byte
  OUT   TCCR0B, R24      ; this behaviour might be unexpected since R24 is no longer the same
```

Aside from giving proper documentations to subroutines which provides information on affected registers, the stack can be used to store values in affected register and retrieve it before returning.

```
; Sends R16 to USART
; Blocks while UDRE0 is not ready.
; Registers Affected: R24
SER_send_byte:
  PUSH  R24             ; stores R24
  LDS   R24, UCSR0A
  SBRS  R24, UDRE0
```

```
RJMP  SER_send_byte
STS   UDR0, R16
POP   R24             ; retrieve the value back to R24
RET
```

Do keep in mind that this would add an overhead by using additional cycles and memory for storing into the stack.

Revision #2
Created 2026-02-25 09:09:05 UTC by MF
Updated 2026-02-25 09:28:02 UTC by MF