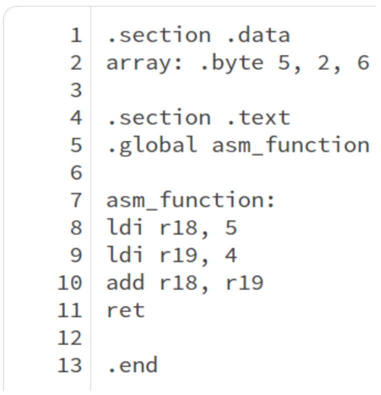
AVR Simulator Guide

Code

AVR code requires the following lines (highlighted in red) to run the code.

```
.section .data
[Replace this line with data definitions]
.section .text
.global asm_function
asm_function:
[Replace this line with AVR instruction code]
ret
.end
```

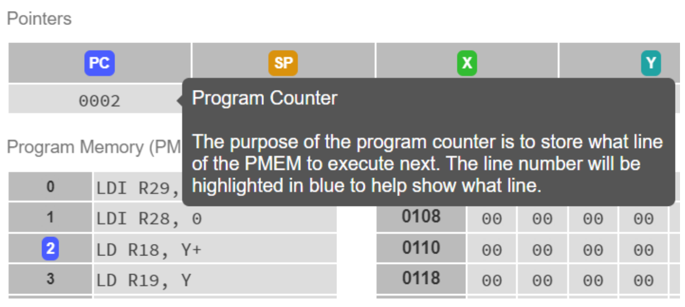
The following picture shows an example of this code.



```
.section .data
array: .byte 5, 2, 6

.section .text
.global asm_function

asm_function:
ldi r18, 5
ldi r19, 4
add r18, r19
ret

.end
```

Popups

Further information is available for some parts of the simulator if you click on something you are uncertain about. Some examples are below.



Pointers

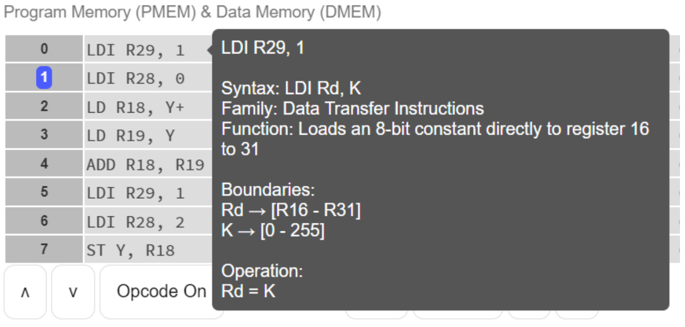
```
0002
```

Program Counter

The purpose of the program counter is to store what line of the PMEM to execute next. The line number will be highlighted in blue to help show what line.

Program Memory (PM

```
0    LDI R29,
1    LDI R28, 0           0108   00   00   00   00
2    LD R18, Y+           0110   00   00   00   00
3    LD R19, Y            0118   00   00   00   00
```



Program Memory (PMEM) & Data Memory (DMEM)

```
0    LDI R29, 1
1    LDI R28, 0
2    LD R18, Y+
3    LD R19, Y
4    ADD R18, R19
5    LDI R29, 1
6    LDI R28, 2
7    ST Y, R18
```

LDI R29, 1

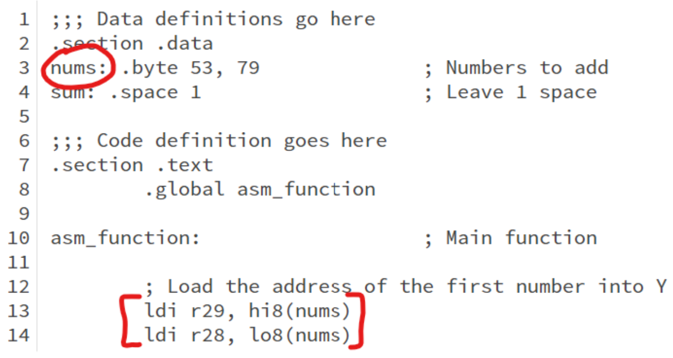
Syntax: LDI Rd, K
Family: Data Transfer Instructions
Function: Loads an 8-bit constant directly to register 16 to 31

Boundaries:
Rd → [R16 - R31]
K → [0 - 255]

Operation:
Rd = K

Using X, Y, and Z

X, Y, and Z are used as pointers to DMEM addresses in AVR. In order to load a DMEM value into them you can reference its label in the data section of the code using the following.



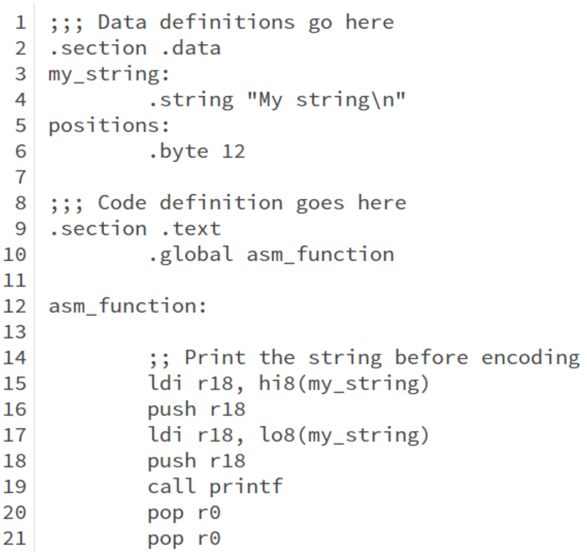
```
;;; Data definitions go here
.section .data
nums: .byte 53, 79              ; Numbers to add
sum: .space 1                   ; Leave 1 space

;;; Code definition goes here
.section .text
        .global asm_function

asm_function:                   ; Main function

        ; Load the address of the first number into Y
        ldi r29, hi8(nums)
        ldi r28, lo8(nums)
```

Printing to the Console

Printing to the console requires pushing the 2 byte address to the stack in the order *hi8* then *lo8*.

Then call the *printf* function and pop the address back of the stack if desired.

The *printf* function prints each value in its ascii form until it reaches a *00*, where it will stop printing and return.



```
;;; Data definitions go here
.section .data
my_string:
        .string "My string\n"
positions:
        .byte 12

;;; Code definition goes here
.section .text
        .global asm_function

asm_function:

        ;; Print the string before encoding
        ldi r18, hi8(my_string)
        push r18
        ldi r18, lo8(my_string)
        push r18
        call printf
        pop r0
        pop r0
```