



Assembly Language Lecture Series: **x86-64 Introduction to SASM: Memory**

Sensei RL Uy, College of Computer Studies,
De La Salle University, Manila, Philippines

Copyright Notice

This lecture contains copyrighted materials and is use solely for instructional purposes only, and not for redistribution.

Do not edit, alter, transform, republish or distribute the contents without obtaining express written permission from the author.

SASM using memory I/O macro (**output**)

PRINT_DEC (decimal output)

Syntax: PRINT_DEC size, data

size: size of **data** in bytes

data: number or symbol constant,
name of variable, register or
address

Note:

- Print number **data** in *signed* decimal representation

Example:

PRINT_DEC 4, var1

SASM using memory I/O macro (**output**)

PRINT_UDEC (decimal output)

Syntax: PRINT_UDEC size, data

size: size of **data** in bytes

data: number or symbol constant,
name of variable, register or
address

Note:

- Print number **data** in *unsigned* decimal representation

Example:

PRINT_UDEC 4, var1

SASM using memory I/O macro (**output**)

PRINT_HEX
(hexadecimal output)

Syntax: PRINT_HEX size, data

size: size of **data** in bytes

data: number or symbol constant,
name of variable, register or
address

Note:

- Print number **data** in hexadecimal representation

Example:

```
PRINT_HEX 4, var1
```

The screenshot displays the SASM application window. The main editor contains assembly code for a 64-bit program. To the right, there are 'Input' and 'Output' panels. At the bottom, a status bar shows the execution progress and completion time.

SASM Interface Components:

- Menu Bar:** File, Edit, Build, Debug, Settings, Help
- Toolbar:** Icons for file operations (open, save, print), editing (undo, redo), and execution (run, stop, step through).
- Code Editor:** Displays assembly code with line numbers 1 through 19.
- Input Panel:** A text area for providing input to the program.
- Output Panel:** Displays the program's output.
- Status Bar:** Shows execution logs and timing.

Assembly Code:

```
1 %include "io64.inc"
2 section .data
3 var1 db -1
4 var2 dw 0x2710
5 var3 dd 0x1CAFEBAD
6 var4 dq 0x7fffffffffffffff
7 section .text
8 global CMAIN
9 CMAIN:
10 ;write your code here
11 PRINT_DEC 1, var1
12 NEWLINE
13 PRINT_UDEC 2, var2
14 NEWLINE
15 PRINT_HEX 4, var3
16 NEWLINE
17 PRINT_HEX 8, var4
18 xor rax, rax
19 ret
```

Macros:

- db:** define byte macro – declare memory as a byte
- dw:** define word macro – declare memory as a word (2 bytes)
- dd:** define doubleword macro – declare memory as a doubleword (4 bytes)
- dq:** define quadword macro – declare memory as a quadword (8 bytes)

Output:

```
-1
10000
1cafebad
7fffffffffffffff
```

Status Bar Log:

```
[07:01:11] Build started...
[07:01:11] Built successfully.
[07:01:13] The program is executing...
[07:01:14] The program finished normally. Execution time: 0.046 s
```

SASM using memory I/O macro (**input**)

GET_DEC (decimal input)

Syntax: GET_DEC size, data

size: size of data in bytes

data: number or symbol constant,
name of variable, register or
address

Note:

- Input number **data** in ***signed*** decimal representation from stdin (command line interface) or input window (SASM)

Example:

GET_DEC 4, var1

SASM using memory I/O macro (input)

GET_UDEC (decimal input)

Syntax: GET_UDEC size, data

size: size of **data** in bytes

data: number or symbol constant,
name of variable, register or
address

Note:

- Input number **data** in *unsigned* decimal representation from stdin (command line interface) or input window (SASM)

Example:

```
GET_UDEC 4, var1
```


SASM using register I/O macro (input)

GET_HEX (hexadecimal input)

Syntax: GET_HEX size, data

size: size of **data** in bytes

data: number or symbol constant,
name of variable, register or
address

Note:

- Input number **data** in hexadecimal representation (**with 0x**) from stdin (command line interface) or input window (SASM)

Example:

```
GET_UDEC 4, var1
```

The screenshot shows the SASM (Simple Assembler) interface. The main window displays an assembly file named `w64_sasm1.asm` with the following code:

```
1 %include "io64.inc"
2 section .data
3 var1 db 0x00
4 var2 dw 0x0000
5 var3 dd 0x00000000
6 var4 dq 0x0000000000000000
7 section .text
8 global CMAIN
9 CMAIN:
10 ;write your code here
11 GET_DEC 1, var1
12 GET_UDEC 2, var2
13 GET_HEX 4, var3
14 GET_HEX 8, var4
15 PRINT_DEC 1, var1
16 NEWLINE
17 PRINT_UDEC 2, var2
18 NEWLINE
19 PRINT_HEX 4, var3
20 NEWLINE
21 PRINT_HEX 8, var4
22 xor rax, rax
23 ret
```

On the right side, there are two panels: "Input" and "Output". Both panels display the same text:

```
-1
10000
abcd1234
ef10abcd12345678
```

At the bottom, a status bar shows the following messages:

```
[07:01:14] The program finished normally. Execution time: 0.046 s
[07:13:06] Build started...
[07:13:06] Built successfully.
[07:13:07] The program is executing...
[07:13:08] The program finished normally. Execution time: 0.04 s
```

SASM using register I/O macro (input)

GET_CHAR (character input)

Syntax: GET_CHAR data

data: name of variable, or address

Note:

- Input **1 character** and store to **data**

Example:

GET_CHAR var1

SASM

File Edit Build Debug Settings Help

w64_sasm2.asm

w64_sasm1.asm

```
1 %include "io64.inc"
2 section .data
3 var1 db 0
4 section .text
5 global CMAIN
6 CMAIN:
7 ;write your code here
8     GET_CHAR var1
9     PRINT_CHAR var1
10    xor rax, rax
11    ret
```

Input

R

Output

R

[07:22:26] The program finished normally. Execution time: 0.047 s

[07:22:30] Build started...

[07:22:30] Built successfully.

[07:22:30] The program is executing...

[07:22:31] The program finished normally. Execution time: 0.04 s

SASM using register I/O macro (input)

GET_STRING (string input)

Syntax:

GET_STRING data, maxsz

data: name of variable, or address

maxsz: register or number
constant

Note:

- *maxsz* – support up to *maxsz*-1 characters. The remaining character is to store the null character (0x00) (SASM version).
- *maxsz* – support up to *maxsz*-2 characters. The remaining 2 characters are “line feed (0x0A)” and null character (0x00) (CLI/DOS version).

SASM using register I/O macro (input)

GET_STRING (string input)

Syntax:

GET_STRING data, maxsz

data: name of variable, or address

maxsz: register or number constant

Example:

```
GET_STRING var1, 11
```

The screenshot shows the SASM application window. The main editor displays assembly code for a program that reads a string and prints it. The code is as follows:

```
1 %include "io64.inc"
2 section .data
3 var1 times 11 db 0
4 section .text
5 global CMAIN
6 CMAIN:
7 ;write your code here
8 GET_STRING var1,11
9 PRINT_STRING var1
10 xor rax, rax
11 ret
```

Annotations in red text explain the code:

- `var1 times 11 db 0`: variable to support 10 chars + 1 (null) [for SASM]
- `times`: "times" macro means repeat the declare data n times

On the right side, the 'Input' and 'Output' panels both display the string 'ABCDEFGHJI'.

The bottom status bar shows the following execution logs:

```
[08:04:41] The program finished normally. Execution time: 0.038 s
[08:10:50] Build started...
[08:10:51] Built successfully.
[08:10:51] The program is executing...
[08:10:53] The program finished normally. Execution time: 0.048 s
```