Assembly Cheat Sheet

Registers

64 bit	32 bit	16 bit	8 bit high	8 bit low	Special Uses
RAX	EAX	AX	AH	AL	Return
RCX	ECX	CX	CH	CL	Counter
RDX	EDX	DX	DH	DL	General Purpose
RBX	EBX	BX	BH	BL	Non-volatile
RDI	EDI	DI	N/A	N/A	Destination (string instructions)
RSI	ESI	SI	N/A	N/A	Source (string instructions)
RSP	ESP	SP*	N/A	N/A	Stack Pointer (Top of the stack)
RBP	EBP	BP*	N/A	N/A	Base Pointer (Typically top of stack frame)
RIP	EIP	IP*	N/A	N/A	Instruction Pointer (or program counter)
R8-R15	N/A	N/A	N/A	N/A	Additional 64-bit general purpose registers

*Probably not very usable in practice (since it contains a 16 bit pointer)

Calling Conventions

System V (x64)

Param 1	Param 2	Param 3	Param 4	Param 5	Param 6
RDI	RSI	RDX	RCX	R8	R9

Volatile Registers (the rest must be preserved):
RAX, RDI, RSI, RDX, RCX, R8, R9, R10, R11

Microsoft (x64)

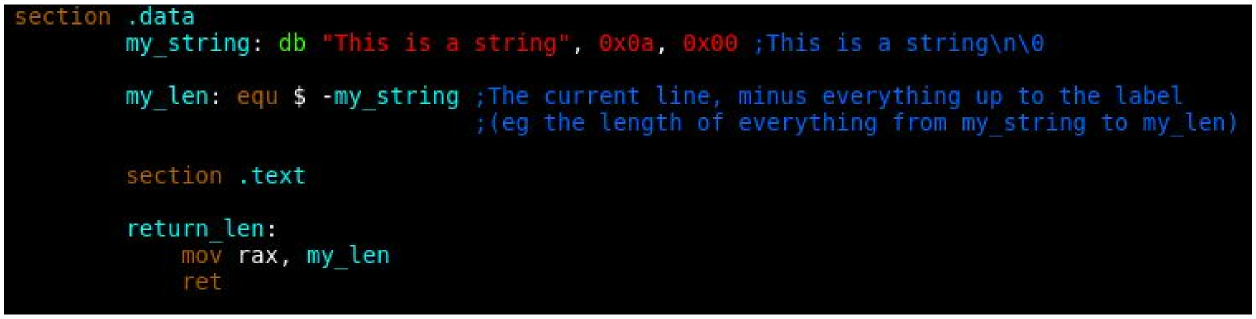
Param 1	Param 2	Param 3	Param 4
RCX	RDX	R8	R9

Volatile Registers (the rest must be preserved):
RAX, RCX, RDX, R8, R9, R10, R11

X86 Non-Volatile Registers (Must be saved by callee):
EBX, EDI, ESI, ESP, EBP

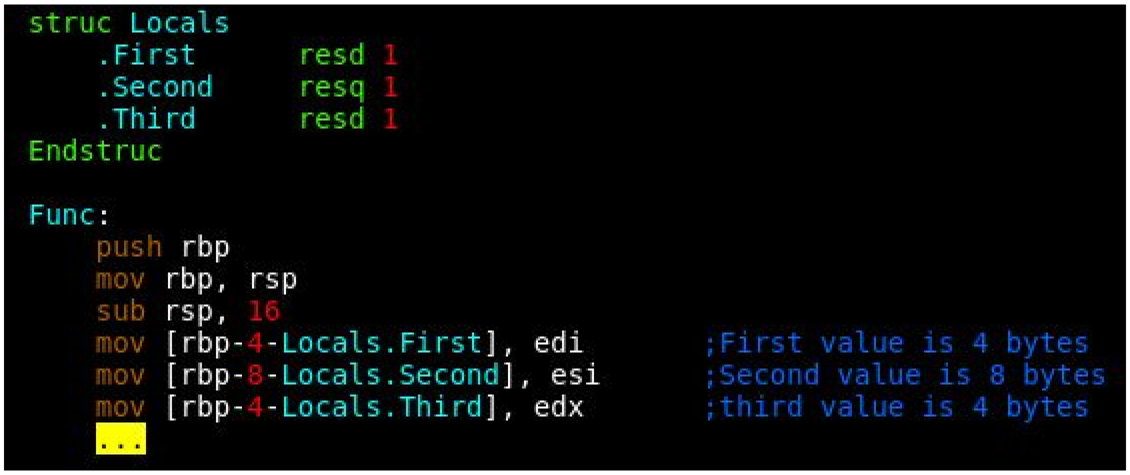
Useful NASM Features:

- **res*** : Reserve space for; e.g., resd would reserve space for a DWORD, resq would reserve space for a QWORD, etc.
- **d*** : Declare; db followed by a string would declare a string of byres, dd 10 would declare a DWORD containing the value “10”, etc
- **equ** : Perform some computation, store the result. Ex:



```
section .data
        my_string: db "This is a string", 0x0a, 0x00 ;This is a string\n\0

        my_len: equ $ -my_string ;The current line, minus everything up to the label
                                 ;(eg the length of everything from my_string to my_len)

        section .text

        return_len:
            mov rax, my_len
            ret
```

- **Struct usage in NASM:**



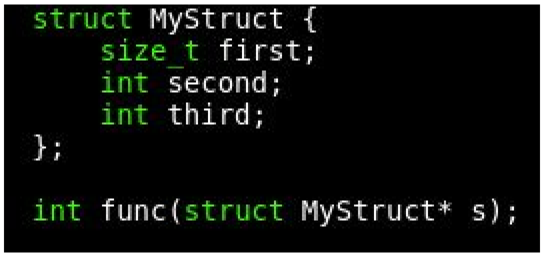
```
struc Locals
    .First      resd 1
    .Second     resq 1
    .Third      resd 1
Endstruc

Func:
    push rbp
    mov rbp, rsp
    sub rsp, 16
    mov [rbp-4-Locals.First], edi       ;First value is 4 bytes
    mov [rbp-8-Locals.Second], esi      ;Second value is 8 bytes
    mov [rbp-4-Locals.Third], edx       ;third value is 4 bytes
    ...
```

Offset Formula (if going backward in memory, eg for stack variable access):
`Base - sizeof(element) - Struct.Field (eg Locals.Second)`

Offset Formula (if going forward in memory, eg interfacing with a C structure):
`Base + Struct.Field`

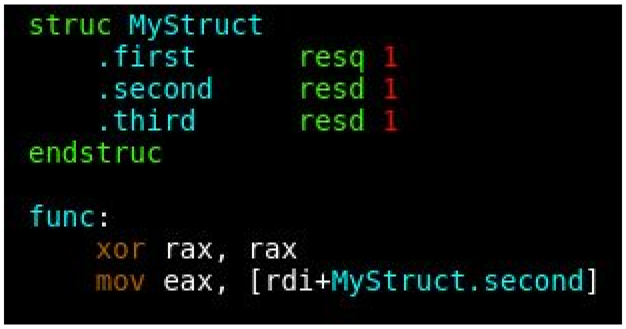
Example:

C (assuming no padding):



```c
struct MyStruct {
    size_t first;
    int second;
    int third;
};

int func(struct MyStruct* s);
```

ASM:



```
struc MyStruct
    .first      resq 1
    .second     resd 1
    .third      resd 1
endstruc

func:
    xor rax, rax
    mov eax, [rdi+MyStruct.second]
```

Sections

- **.text** : Executable code
- **.data** : Typically pre-initialized data (eg declared strings)
- **.bss** : Uninitialized data (eg reserved space)

Instructions and More:

https://www.cs.uaf.edu/2017/fall/cs301/reference/x86_64.html