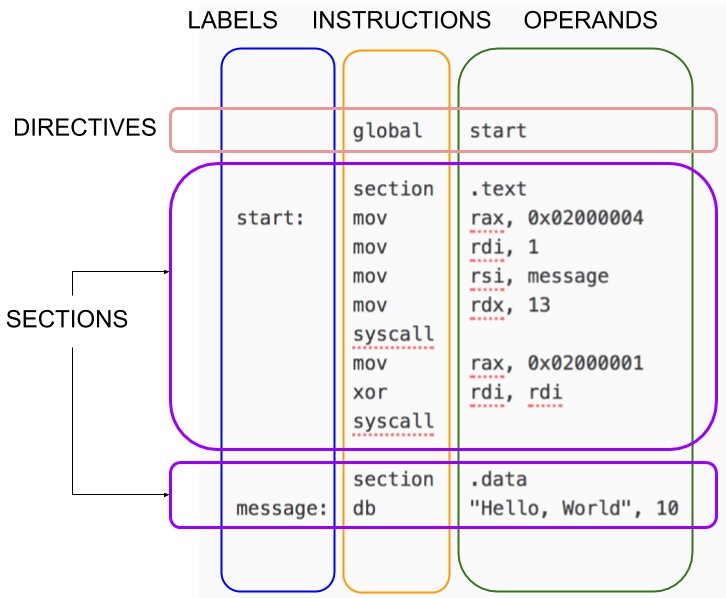
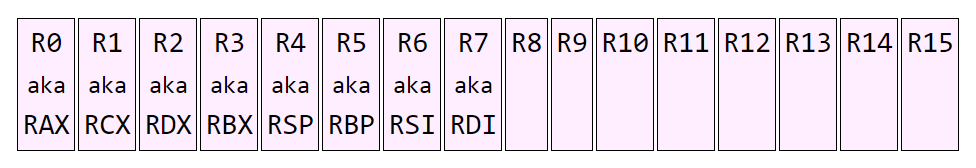
**Chapter 13: MISCELLANEOUS**

**Topic – 1: Structure Of Program**

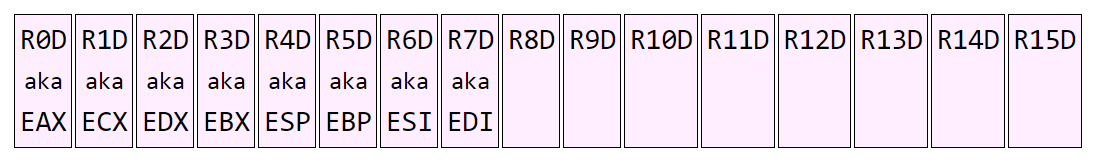


**Topic – 2: Register Operand Aliases**

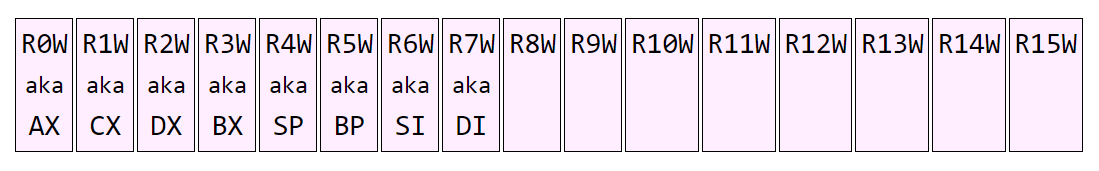
**64-Bit**



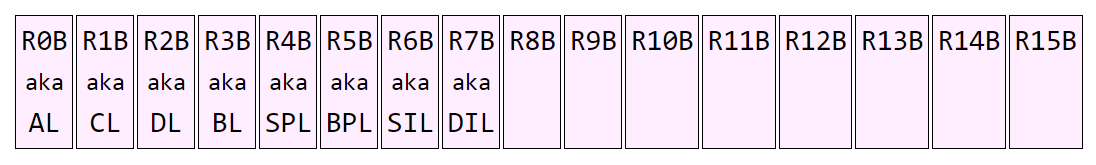
**32-Bit**



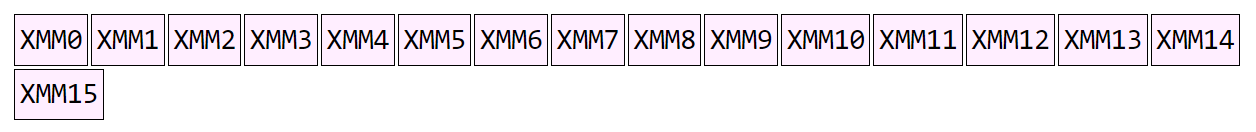
**16-Bit**



**8-Bit**



**128-Bits (XMM Registers)**



**Topic – 3: Inline Assembly Code**

**Introduction**

* We can write **inline assembly** code using ***\_\_asm\_\_*** function.
* By **default**, the syntax is read as **AT&T GAS** syntax, but this can be changed to **Intel** syntax using code we will see.
* Here, each line must end with **semicolon** somehow.
* **There are three parts of program –** Code, output transfers & input transfers.
* The code if converted to **Intel syntax**, must be **reverted back** to **GAS syntax** before the **output transfers** & **input transfers** sections arrive.
* And there is **no** separate standard library required to do it.

**Example Inline Code**

* In this example, we will **add two numbers** & display it on screen.

***#include <stdio.h>***

***int main()***

***{***

***int a=5, b=3, res;***

***\_\_asm\_\_(***

***".intel\_syntax nonprefix;" // Read syntax as Intel assembly***

***"add eax, ebx;"***

***".att\_syntax" // Read syntax as AT&T assembly***

***: "=a" (res) // Outputs (stores EAX in res)***

***: "a" (a), "b" (b) // Inputs (stores 'a' in EAX & 'b' in EBX)***

***);***

***printf("Result = %d\n", res);***

***return 0;***

***}***

* **"a"** above refers to register **EAX** & **"b"** so to **EBX**.
* While result uses **"=a"**.

**GPR Register Constraints**

* Let’s see some register constraints used in **output** & **input transfers**.

|  |  |
| --- | --- |
| **Constraint** | **Register** |
| **a** | **RAX** |
| **b** | **RBX** |
| **c** | **RCX** |
| **d** | **RDX** |
| **S** | **RSI** |
| **D** | **RDI** |
| **r** | **Any GPR** |

**Special Register Constraints**

|  |  |
| --- | --- |
| **Constraint** | **Register** |
| **q** | **Any (lower 8-bits only)** |
| **A** | **RAX, RDX** |

**I/O Register Constraints**

|  |  |
| --- | --- |
| **Modifier** | **Meaning** |
| **=** | **Operand is output.** |
| **+** | **Operand is both input & output.** |
| **&** | **Operands can be modified before inputs.** |
| **%** | **Operand can occupy only registers unused by other operands.** |

**Topic – 4: Linking Libraries**

**C Function In Assembly**

* We use ***extern*** instruction to link a **C function** to our **NASM** code.
* We pass ***string*** ***format*** to **RDI** register & ***message*** to **RSI** register.

***section .data***

***format db "Hello, %s!", 10, 0 ; 0 is newline***

***msg db "World", 0***

***section .text***

***global \_start***

***extern printf***

***\_start:***

***mov rdi, format***

***mov rsi, msg***

***call printf ; Calling printf***

***; Exiting program***

***mov rax, 60***

***xor rdi, rdi***

***syscall***

* When creating an **executable**, we have to **link** the **C standard library** with our program.
* We use flag ***-lc*** for that, which means linking against **libc** (C standard library).
* Also, we use ***no-pie*** with **GCC** to **disable Position Independent Index (PIE)**.
* **PIE** is **disabled** to avoid possible linking related issues.
* For using **math functions** from **math** library, we can add ***-lm*** (from **libm**) after ***-lc***.

***nasm -f elf64 -o myprogram.o myprogram.asm***

***gcc -no-pie -o myprogram myprogram.o -lc***

***./myprogram***

**Assembly Function In C**

***section .text***

***global add\_two\_numbers ; Declare function as global***

***add\_two\_numbers:***

***mov rax, rdi ; Move the first argument (a) into rax***

***add rax, rsi ; Add the second argument (b) to rax***

***ret ; Return the result in rax***

* We made the function **global** to make it available to the linker.
* Notice that we are again passing values to **RDI** & **RSI** registers.

**Note!**

**🡪 RDI & RSI are used for passing arguments as per global "System V ABI" convention.**

***#include <stdio.h>***

***// Declare external assembly function***

***extern long add\_two\_numbers(long a, long b);***

***int main()***

***{***

***long number = add\_two\_numbers(5, 3);***

***printf("Result: %ld\n", result);***

***return 0;***

***}***

* Now we will **assemble** & **link** the problem.

***nasm -f elf64 -o add.o add.asm***

***gcc -c -o main.o main.c***

***gcc -o myprogram main.o add.o***

***./myprogram***

**Assembly Code In Assembly Program**

* First, we will write the program to be linked later on.

***; add.asm***

***section .text***

***global add\_numbers ; Make the function available for linking***

***add\_numbers:***

***mov rax, rdi ; Move first argument (rdi) into rax***

***add rax, rsi ; Add second argument (rsi) to rax***

***ret ; Return the result in rax***

* Now we will write the program calling it.

***; main.asm***

***section .data***

***msg db "Result: %d", 0***

***section .text***

***global \_start***

***extern add\_numbers ; Declare external function from add.asm***

***extern printf***

***\_start:***

***mov rdi, 5 ; First argument for add\_numbers***

***mov rsi, 7 ; Second argument for add\_numbers***

***call add\_numbers ; Call the add\_numbers function***

***; Prepare to call printf to display the result***

***mov rsi, rax ; Move result of add\_numbers into rsi (2nd arg)***

***mov rdi, msg ; Move format string into rdi (1st arg)***

***xor rax, rax ; Clear rax for variadic printf***

***call printf ; Call printf***

***; Exit the program***

***mov rax, 60***

***xor rdi, rdi***

***syscall***

* Time to **assemble** both the files!

***nasm -f elf64 -o add.o add.asm***

***nasm -f elf64 -o main.o main.asm***

***gcc -no-pie -o myprogram main.o add.o -lc***

***./myprogram***