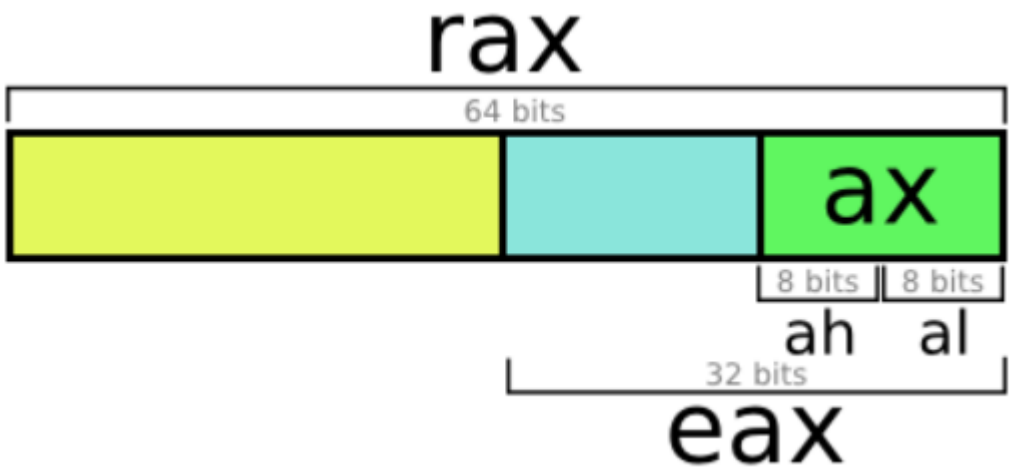
64 bit registers

Modern CPU's all have 64 bit registers

- For backward compatibility it is still capable of 32 bit addressing
- The register names:



Comparison between 64 bit and 32 bit instructions

Assembly Output (32 Bit or x86 Code)

```
mov eax, 10
add eax, 20 ;OUTPUT IS SAVED TO "EAX" REGISTER WHICH IS A 32 BITS REGISTER
```

Assembly Output (64 Bit or x86_64 Code)

```
mov rax, 10
add rax, 20 ;OUTPUT IS SAVED TO "RAX" REGISTER WHICH IS A 64 BIT REGISTER
```

64-bit register	Lower 32 bits	Lower 16 bits	Lower 8 bits
rax	eax	ax	al
rbx	ebx	bx	bl
rcx	ecx	cx	cl
rdx	edx	dx	dl
rsi	esi	si	sil
rdi	edi	di	dil
rbp	ebp	bp	bpl
rsp	esp	sp	spl
r8	r8d	r8w	r8b
r9	r9d	r9w	r9b
r10	r10d	r10w	r10b
r11	r11d	r11w	r11b
r12	r12d	r12w	r12b
r13	r13d	r13w	r13b
r14	r14d	r14w	r14b
r15	r15d	r15w	r15b

x64 extends x86's 8 general-purpose registers to be 64-bit, and adds 8 new 64-bit registers. The 64-bit registers have names beginning with "r", so for example the 64-bit extension of **eax** is called **rax**.

The new registers are named **r8** through **r15**.

Passing arguments to functions

- Method 1: Using Registers
- Method 2: Using Global Data
- Method 3: Using the Stack

Method 1: registers

- Values are passed on some of the registers:

```
    mov     ecx,5       ; argument
    mov     edx,2       ; argument
    call    my_func
```

```
my_func:
    mov     eax,ecx
    sub     eax,edx
    ret
```

- Sometimes referred to as FASTCALL.
- Very common in 64 bit long mode.
  - There are more registers.

credits: xorpd.net

Method 2:
global data

- Values are passed through a global memory location:

```
section '.bss' readable writeable
    arg1          dd      ?
    arg2          dd      ?
```

```
    mov     dword [arg1],5     ; argument
    mov     dword [arg2],2     ; argument
    call    my_func
```

```
my_func:
    mov     eax,dword [arg1]
    sub     eax,dword [arg2]
    ret
```

credits: xorpd.net

Method 3: stack

- We pass arguments over the stack:

```
    push    5       ; argument
    push    2       ; argument
    call    my_func
    add     esp,8 ; clean stack.
```

```
my_func:
    mov     eax,dword [esp + 8]
    sub     eax,dword [esp + 4]
    ret
```

credits: xorpd.net

Thank you