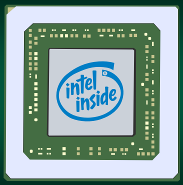
Introduction to the Intel x64

Part 2

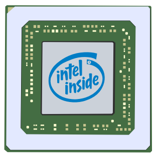
The Intel x64



It was simple at first…

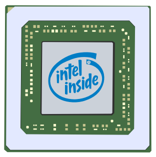
The Intel x64

- The Intel x64 is the main processor used by servers, laptops, and desktops
- It has evolved continuously over a 40 year period
- The term "x86" refers to the 32-bit and 16



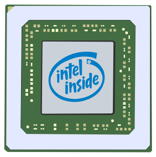
What to call the processor

- The classic term "x86" refers to the 32-bit and 16-bit processor family
- With move to 64-bit, the term "x64" is used to differentiate the newest design from the previous



The Original x86

- First "x86" was the Intel 8086 released in 1978
- Attributes:
  - 16-bit processor (registers were 16-bit)
  - 16 registers
  - can access of 1MB of RAM



Original x86 Registers

- The x86 processor has evolved continuously over the last 4 decades
- It jumped to 32-bit, and then, finally, to 64-bit
- The result is many of the registers have strange names

Original x86 Registers

It was simple at first…

Original x86 Registers

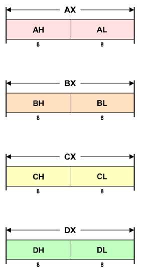
- The original x86 contained 16 registers
- 8 can be used by your programs
- The other 8 are used for memory management

Original x86 Registers

- 8 Registers can be used by your programs
  - Four General Purpose: AX, BX, CX, DX
  - Four pointer index: SI, DI, BP, SP
- The remaining 8 are restricted
  - Six segment: CS, DS, ES, FS, GS, SS
  - One instruction pointer: IP
  - One status register – used in computations

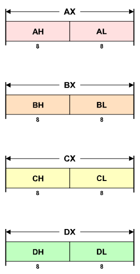
Original General Purpose Registers

- However, back then (and now too) it is very useful to store 8-bit values
- So, Intel chopped 4 of the registers in half
- These registers have generic names of A, B, C, D



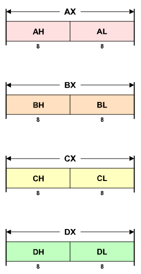
Original General Purpose Registers

- The first and second byte can be used separately or used together
- Naming convention
  - high byte has the suffix "H"
  - low byte has the suffix "L"
  - for both bytes, the suffix is "X"

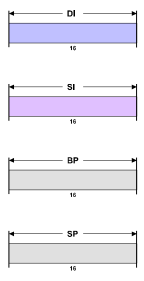


Original General Purpose Registers

- This essentially doubled the number of registers
- So, there are:
  - four 16-bit registers or
  - eight 8-bit registers
  - *(and any combination you can think off)*

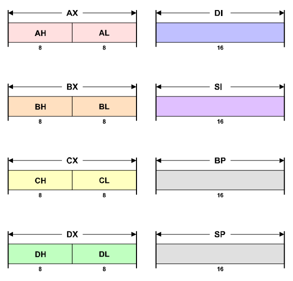


Last the 4 Registers



- The remaining 4 registers were not cut in half
- Used for storing indexes (for arrays) and pointers
- Their purpose
  - DI – destination index
  - SI – source index
  - BP – base pointer
  - SP – stack pointer

Original 16-Bit Registers



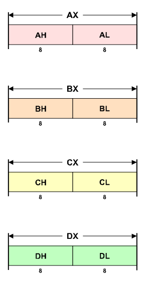
Evolution to 64 Bit Registers

This is going to hurt…

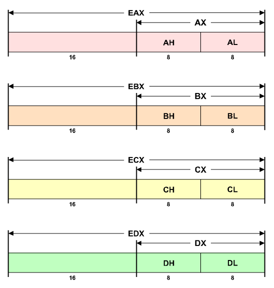
Evolution to 32-bit

- When the x86 moved into the 32-bit era, Intel expanded the registers to 32-bit
  - the 16-bit ones still exist
  - they have the prefix "e" for extended
- New instructions were added (to use them)

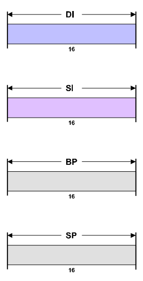
Original Registers



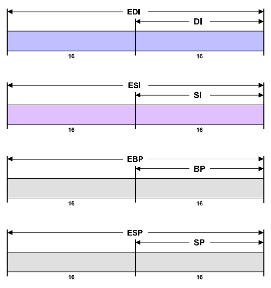
Expansion to 32-bit



Original Registers



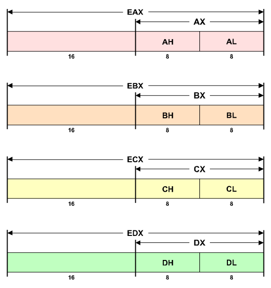
Expansion to 32-bit



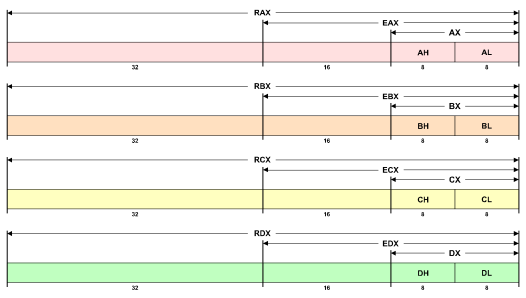
Evolution to 64-bit

- The processor then evolved to 64-bit
- The registers were extended again
  - the 64-bit have the prefix *"r" for register*
  - 8 additional registers were added
  - also, it is now possible to get 8-bit values from all registers (hardware is more consistent!)

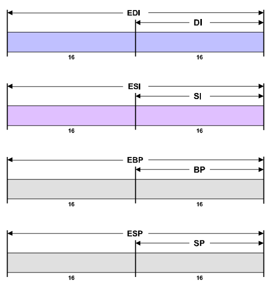
Expansion to 64-bit



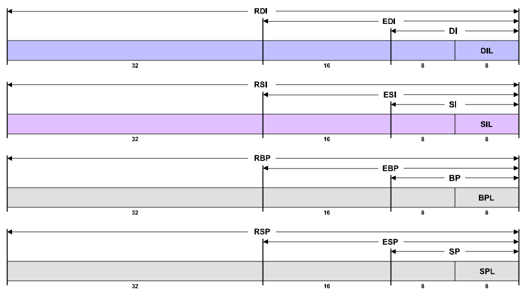
Expansion to 64-bit



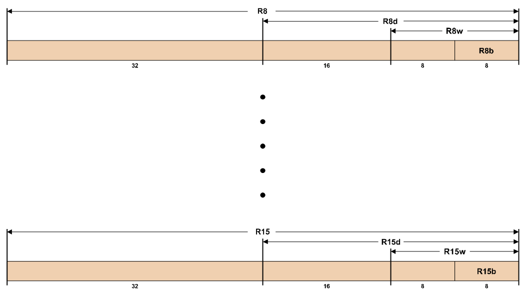
Expansion to 64-bit



Expansion to 64-bit



New 64-bit Registers: R8…R15

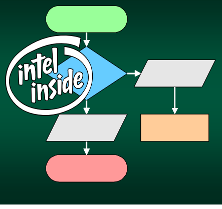


64-Bit Register Table

Register	32-bit	16-bit	8-bit High	8-bit Low
rax	eax	ax	ah	al
rbx	ebx	bx	bh	bl
rcx	ecx	cx	ch	cl
rdx	edx	dx	dh	dl
rsi	esi	si		sil
rdi	edi	di		dil
rbp	ebp	bp		bpl
rsp	esp	sp		spl

64-Bit Register Table

Register	32-bit	16-bit	8-bit High	8-bit Low
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

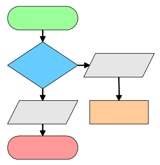


Basic Intel x86 Instructions

Feel the pow-wah of the x86!

Basic Intel x86 Instructions

- Each x86 instruction can have up to 2 operands
- Operands in x86 instructions are very versatile
- Often each argument can be either a memory address, register or an immediate value

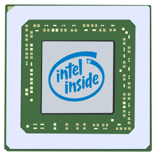


Types of Operands

- Registers
- Memory address
- Register pointing to memory
- A constant stored with the instruction – this is called an *immediate*

Intel x86 Instruction Limits

- There are some limitations…
- Some instructions must use an immediate
- Some instructions require a *specific* register to perform calculations



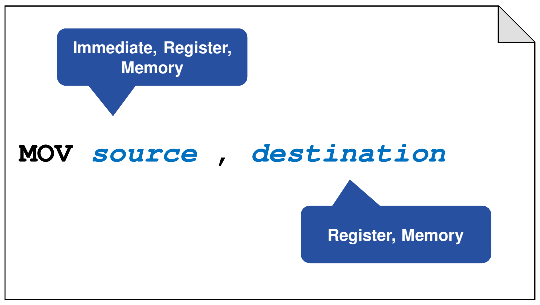
Intel x86 Instruction Limits

- A register must always be involved
  - processors use registers for all activity
  - both operands cannot access memory at the same time
  - *the processor has to have it at some point!*
- Also, obviously, the receiving field cannot be an immediate value

Instruction: Move

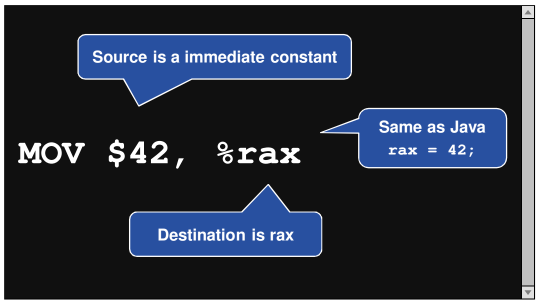
- The x86 Move Instruction combines load, store, and register transfer logic
- It is one of the most common instructions used in programs (true of all processors)
- Remember how often you use the assignment statement in C / Java?

Instruction: Move

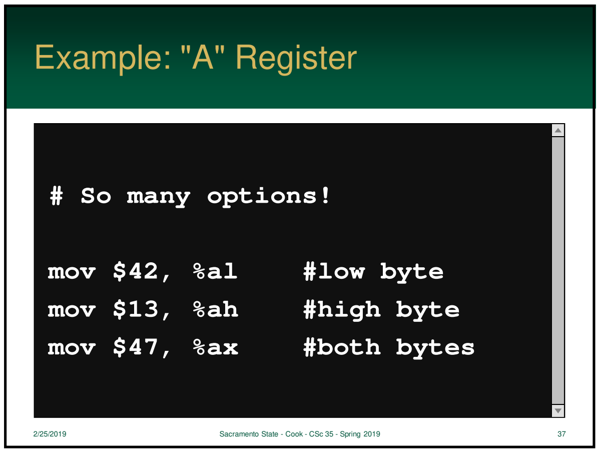


```
MOV source , destination
```

Example: Move immediate



```
MOV $42, %rax
```

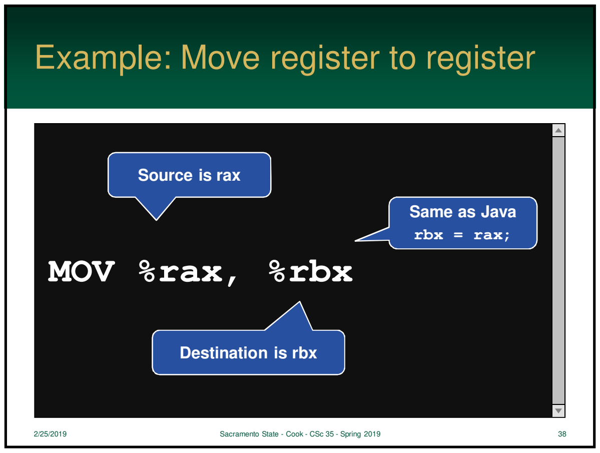


Example: "A" Register

```
# So many options!

mov $42, %al      #low byte
mov $13, %ah      #high byte
mov $47, %ax      #both bytes
```

2/25/2019 Sacramento State - Cook - CSc 35 - Spring 2019 37



Example: Move register to register

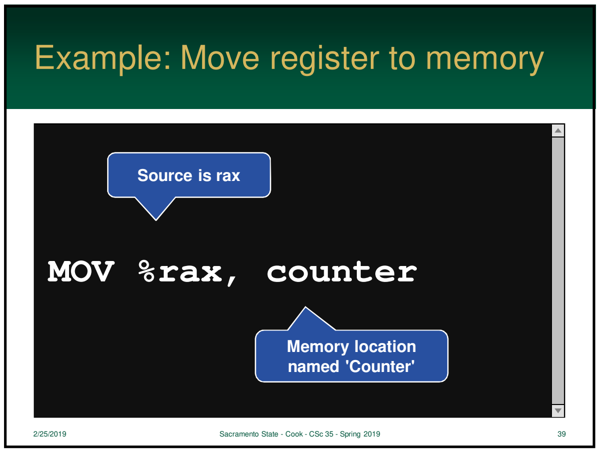
Source is rax

Same as Java `rbx = rax;`

```
MOV %rax, %rbx
```

Destination is rbx

2/25/2019 Sacramento State - Cook - CSc 35 - Spring 2019 38



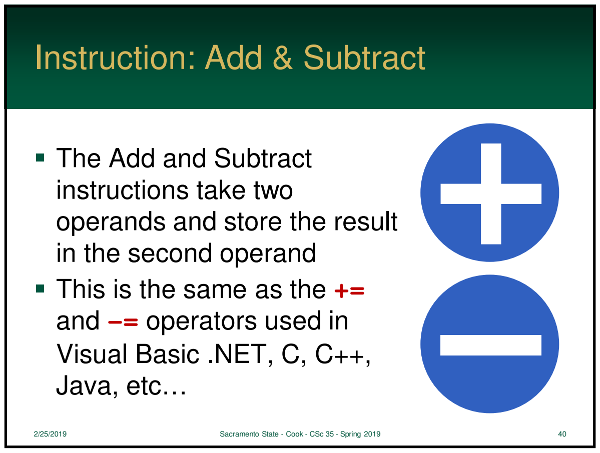
Example: Move register to memory

Source is rax

```
MOV %rax, counter
```

Memory location named 'Counter'

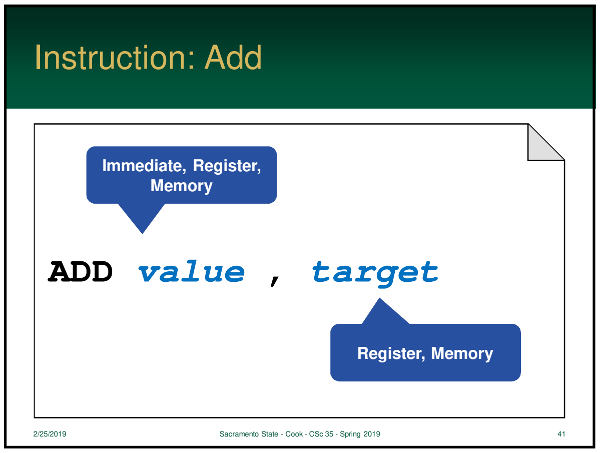
2/25/2019 Sacramento State - Cook - CSc 35 - Spring 2019 39



Instruction: Add & Subtract

- The Add and Subtract instructions take two operands and store the result in the second operand
- This is the same as the **+=** and **−=** operators used in Visual Basic .NET, C, C++, Java, etc…

2/25/2019 Sacramento State - Cook - CSc 35 - Spring 2019 40



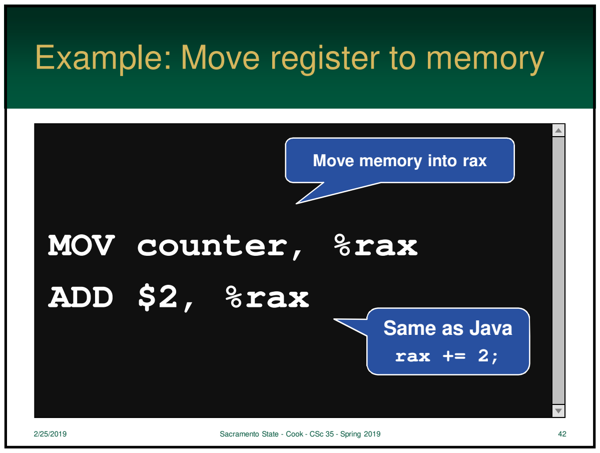
Instruction: Add

Immediate, Register, Memory

```
ADD value , target
```

Register, Memory

2/25/2019 Sacramento State - Cook - CSc 35 - Spring 2019 41



Example: Move register to memory

Move memory into rax

```
MOV counter, %rax
ADD $2, %rax
```

Same as Java `rax += 2;`

2/25/2019 Sacramento State - Cook - CSc 35 - Spring 2019 42

Instruction: And & Or

- The Logical And and Logical Or instructions take two operands and stores the result in the second operand
- This is the same as the ^= and |= operators used in C, C++, Java, etc…

∧

∨

Instruction: Logical And

```
AND value , target
```

Example: Logical Or

```
#Convert 5 to ASCII '5'
MOV   $5, %rax
OR    $0x30, %rax
```

0011 0000

Instruction: Call

- The Call Instruction transfers control to a memory location (a subroutine)
- Subroutines are analogous to the functions you wrote in Java
- Once it completes, execution continues after the call

Instruction: Call

```
CALL  address
```

Usually a label – a constant that holds an address

Example: Print an integer

```
#This is using the CSC35 library

MOV   $42, %rcx
CALL  PrintInt
```

This name is an address