



## Part 2

### Processors

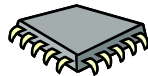


## Computer Processors

### The Brains of the Box

## Computer Processors

- The *Central Processing Unit (CPU)* is the most complex part of a computer
- In fact, it is the computer
- It works far different from a high-level language



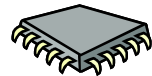
9/18/2017

Sacramento State - Cook - CSc 35 - Fall 2017

3

## Components of a Processor

- *Execution Unit (EU)*
  - performs calculations & logic
  - registers hold data
- *Control Logic Unit (CLU)*
  - reads and decodes instructions
  - talks to other components



9/18/2017

Sacramento State - Cook - CSc 35 - Fall 2017

4

## Execution Unit

- Contains the hardware that *executes* tasks (your programs)
- Different in many processors
- Modern processors often use multiple execution units to execute instructions in parallel to improve performance

9/18/2017

Sacramento State - Cook - CSc 35 - Fall 2017

5

## Execution Unit – The ALU

- The *Arithmetic Logic Unit* performs all calculations and comparisons
- Processor often contains special hardware for integer and floating point



9/18/2017

Sacramento State - Cook - CSc 35 - Fall 2017

6

## Control Logic Unit (CLU)



- Controls the processor
- Determines when instructions can be executed
- Controls internal operations
  - fetch and execute each instruction
  - and store result of each instruction

9/18/2017

Sacramento State - Cook - CSc 35 - Fall 2017

7

## CLU Over Time

- In early processors...
  - CLU was a very small fraction of the hardware
  - EU and the registers took most of the space
- New processors...
  - complex control unit one of the more difficult parts of a processor to design
  - has increased in its percentage of the processor hardware

9/18/2017

Sacramento State - Cook - CSc 35 - Fall 2017

8

## Computer Processors

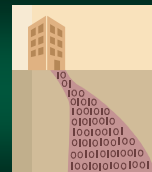
- Over time, thousands of processors were developed
- Examples:
  - Intel x86
  - IBM PowerPC
  - MOS 6502
  - ARM



9/18/2017

Sacramento State - Cook - CSc 35 - Fall 2017

9

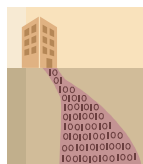


## Registers

Where the work is done

## Registers

- In high level languages, you put active data into variables
- However, it works quite different on processors
- All computations performed are done in *registers*



9/18/2017

Sacramento State - Cook - CSc 35 - Fall 2017

11

## What are registers used for?

- Registers are used to store anything the processor needs to keep track of
- Examples:
  - the result of calculations
  - status information
  - memory location of the running program
  - and much more...

9/18/2017

Sacramento State - Cook - CSc 35 - Fall 2017

12

## What – exactly – is a register?

- A register is a memory location located on the processor itself
- Think of it as a special "variable"
- Designed to be fast
- Some are accessible and usable by a programs, but many are hidden.

9/18/2017

Sacramento State - Cook - CSc 35 - Fall 2017

13

## General Purpose Registers

- *General Purpose Registers (GPR)* don't have a specific purpose
- They are designed to be used by programs – however they are needed
- Often, you must use registers to perform calculations

9/18/2017

Sacramento State - Cook - CSc 35 - Fall 2017

14

## Some Special Registers

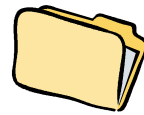
- Program counter
  - controls the current memory location of the running program
  - privileged – only the processor and OS can change it
- Stack pointer
  - tracks the top of the system stack
  - you can modify this ... *with care...*

9/18/2017

Sacramento State - Cook - CSc 35 - Fall 2017

15

## Register Files



- All the related registers are grouped into a *register file*
- Different processors access and use their register files in very different ways
- Some processors support multiple files

9/18/2017

Sacramento State - Cook - CSc 35 - Fall 2017

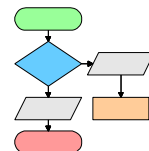
16

Instructions

Your programs are simple

## Instructions

- Processors do not have the constructs you find in high-level languages
- Examples:
  - Blocks
  - If Statements
  - While Statements
  - ... etc



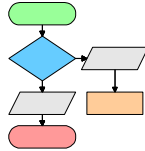
9/18/2017

Sacramento State - Cook - CSc 35 - Fall 2017

18

## Instructions

- Processors can only a series of simple tasks
- These are called *instructions*
- Examples:
  - add two values together
  - move a value
  - jump to a memory location



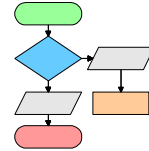
9/18/2017

Sacramento State - Cook - CSc 35 - Fall 2017

19

## Instructions

- These instructions are used to create all logic needed by a program
- We will cover how to do this during the semester

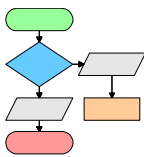


9/18/2017

Sacramento State - Cook - CSc 35 - Fall 2017

20

## Processor Instruction Set



- A processor's *instruction set* defines all the available instructions
- The instructions and their respective formats are very different for each processor

9/18/2017

Sacramento State - Cook - CSc 35 - Fall 2017

21

## What – exactly – is an instruction?

- An instruction is a series of bytes that contain everything the processor needs to know to do something
- An instruction must specify:
  - operation* – what to do
  - operands* – what data is to be used

9/18/2017

Sacramento State - Cook - CSc 35 - Fall 2017

22

## Operation Codes

- Each instruction has an *operation code* (*Opcode*)
- This a unique value that specifies the exact operation to be performed by the processor
- Assemblers use friendly names for called *mnemonics*

9/18/2017

Sacramento State - Cook - CSc 35 - Fall 2017

23

## Instruction Encoding

- Each instruction on a computer is *encoded* into 1's and 0's
- All information that needs to be stored, has to be converted to bits



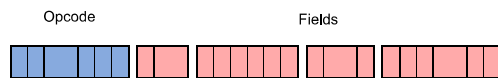
9/18/2017

Sacramento State - Cook - CSc 35 - Fall 2017

24

## Typical Instruction Format

- The *opcode* contains a unique value that indicates the operation to be performed
- It is typically followed by various fields containing register codes, addressing data, etc...



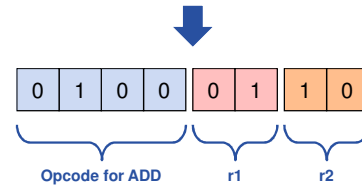
9/18/2017

Sacramento State - Cook - CSc 35 - Fall 2017

25

## Machine Code Example (not x86)

ADD %r1, %r2



9/18/2017

Sacramento State - Cook - CSc 35 - Fall 2017

26

## Types of Operations (Opcodes)

- Data Transfer
- Program Flow Control
- Arithmetic and Logic operations
- Input and Output Instructions

9/18/2017

Sacramento State - Cook - CSc 35 - Fall 2017

27

## Operations: Data Transfer

- One of the most common tasks is moving data to and from registers
- Classified into three categories:
  - *loading* a register with data in memory
  - *storing* data in a register into memory
  - *transferring* data between registers

9/18/2017

Sacramento State - Cook - CSc 35 - Fall 2017

28

## Operations: Control Flow

- Processors do not support blocks, If Statements, etc...
- Instead, you must jump around code you don't want to execute
- This is the same of idea of GoTo Statements



9/18/2017

Sacramento State - Cook - CSc 35 - Fall 2017

29

## Operations: Arithmetic and Logic

- Many operations are used to modify data such as arithmetic, comparisons, and shifting
- Comparison is in this category
  - when two operands are compared – often one is *subtracted* from the other
  - result sets Boolean flags – *more on this later!*



9/18/2017

Sacramento State - Cook - CSc 35 - Fall 2017

30

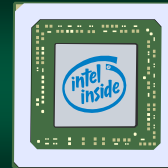
## Operations: Input and Output

- There are also instructions that are designed to talk to ports, hard drives and other components
- However, in modern systems, these are privileged and only usable by the operating system
- You will use interrupts to tell the operating system to input/output data

9/18/2017

Sacramento State - Cook - CSc 35 - Fall 2017

31

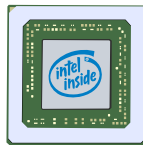


## Original x86 Registers

It was simple at first...

## Original x86 Registers

- 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



9/18/2017

Sacramento State - Cook - CSc 35 - Fall 2017

33

## Intel 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

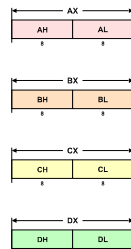
9/18/2017

Sacramento State - Cook - CSc 35 - Fall 2017

34

## 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



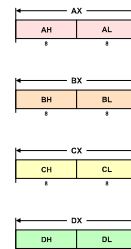
9/18/2017

Sacramento State - Cook - CSc 35 - Fall 2017

35

## 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"



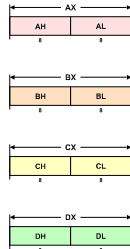
9/18/2017

Sacramento State - Cook - CSc 35 - Fall 2017

36

## 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)

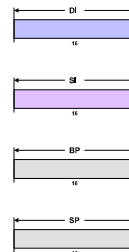


9/18/2017

Sacramento State - Cook - CSc 35 - Fall 2017

37

## Last the 4 Registers



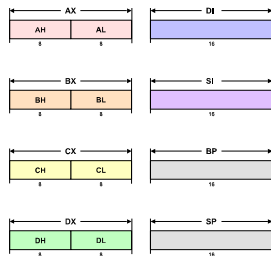
- The remaining 4 registers were not cut in half
- Used for storing indexes (for arrays) and pointers
- Their purpose
  - DI – destination index
  - SI – source index
  - BP – base pointer
  - SP – stack pointer

9/18/2017

Sacramento State - Cook - CSc 35 - Fall 2017

38

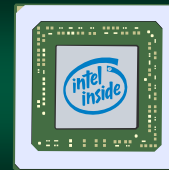
## Original 16-Bit Registers



9/18/2017

Sacramento State - Cook - CSc 35 - Fall 2017

39

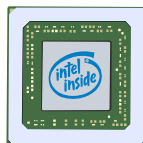


Evolution to  
64 Bit  
Registers

This is going to hurt...

## Original x86 Registers

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



9/18/2017

Sacramento State - Cook - CSc 35 - Fall 2017

41

## Evolution to 32-bit

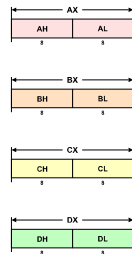
- When the x86 moved into the 32-bit era, Intel expanded the registers to 32-bit
  - the 16-bit ones still exist
  - ... but also have a 32-bit version
  - they have the prefix "e" for *extended*
- New instructions were added (to use them)

9/18/2017

Sacramento State - Cook - CSc 35 - Fall 2017

42

## Original Registers

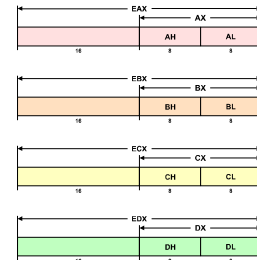


9/18/2017

Sacramento State - Cook - CSc 35 - Fall 2017

43

## Expansion to 32-bit

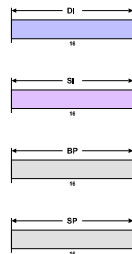


9/18/2017

Sacramento State - Cook - CSc 35 - Fall 2017

44

## Original Registers

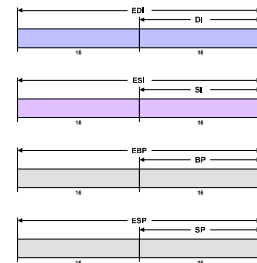


9/18/2017

Sacramento State - Cook - CSc 35 - Fall 2017

45

## Expansion to 32-bit



9/18/2017

Sacramento State - Cook - CSc 35 - Fall 2017

46

## Evolution to 64-bit

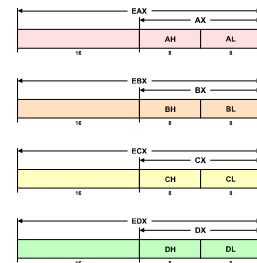
- Once again, the processor evolved – now to 64-bit
- The registers were extended again
  - the 64-bit have the prefix *"r"* for register
  - 8 additional registers were added
  - also, it is now possible to get 8-bit values from all registers (hardware is more consistent!)

9/18/2017

Sacramento State - Cook - CSc 35 - Fall 2017

47

## Expansion to 64-bit



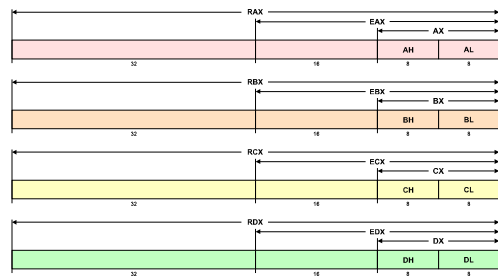
9/18/2017

Sacramento State - Cook - CSc 35 - Fall 2017

48



## Expansion to 64-bit

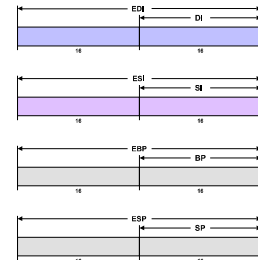


9/18/2017

Sacramento State - Cook - CSc 35 - Fall 2017

49

## Expansion to 64-bit

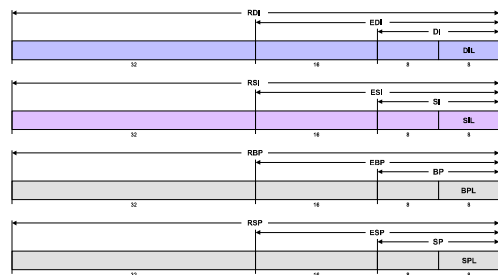


9/18/2017

Sacramento State - Cook - CSc 35 - Fall 2017

50

## Expansion to 64-bit

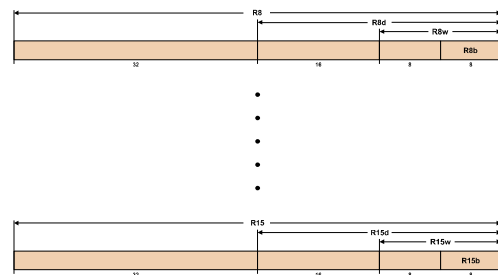


9/18/2017

Sacramento State - Cook - CSc 35 - Fall 2017

51

## New 64-bit Registers: R8...R15



9/18/2017

Sacramento State - Cook - CSc 35 - Fall 2017

52

## 64-Bit Register Table

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

9/18/2017

Sacramento State - Cook - CSc 35 - Fall 2017

53

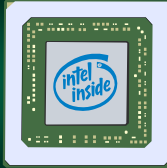
## 64-Bit Register Table

Register	32-bit	16-bit	8-bit High	8-bit Low
<b>r8</b>	<b>r8d</b>	<b>r8w</b>		<b>r8b</b>
<b>r9</b>	<b>r9d</b>	<b>r9w</b>		<b>r9b</b>
<b>r10</b>	<b>r10d</b>	<b>r10w</b>		<b>r10b</b>
<b>r11</b>	<b>r11d</b>	<b>r11w</b>		<b>r11b</b>
<b>r12</b>	<b>r12d</b>	<b>r12w</b>		<b>r12b</b>
<b>r13</b>	<b>r13d</b>	<b>r13w</b>		<b>r13b</b>
<b>r14</b>	<b>r14d</b>	<b>r14w</b>		<b>r14b</b>
<b>r15</b>	<b>r15d</b>	<b>r15w</b>		<b>r15b</b>

9/18/2017

Sacramento State - Cook - CSc 35 - Fall 2017

54

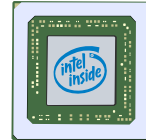


## Basic Intel x86 Instructions

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



9/18/2017

Sacramento State - Cook - CSc 35 - Fall 2017

56

## Types of Operands

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

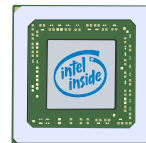
9/18/2017

Sacramento State - Cook - CSc 35 - Fall 2017

57

## Intel x86 Instruction Limits

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



9/18/2017

Sacramento State - Cook - CSc 35 - Fall 2017

58

## 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

9/18/2017

Sacramento State - Cook - CSc 35 - Fall 2017

59

## 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?

9/18/2017

Sacramento State - Cook - CSc 35 - Fall 2017

60

## Instruction: Move

Immediate, Register,  
Memory

**MOV** *source* , *destination*

Register, Memory

9/18/2017

Sacramento State - Cook - CSc 35 - Fall 2017

61

## Example: Move immediate

Source is a Immediate constant

**MOV** \$42, %rax

rax = 42;

Destination is rax

9/18/2017

Sacramento State - Cook - CSc 35 - Fall 2017

62

## Example: "A" Register

# So many options!

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

9/18/2017

Sacramento State - Cook - CSc 35 - Fall 2017

63

## Example: Move register to register

Source is rax

**MOV** %rax, %rbx

rbx = rax;

Destination is rbx

9/18/2017

Sacramento State - Cook - CSc 35 - Fall 2017

64

## Example: Move register to memory

Source is rax

**MOV** %rax, counter

Memory location  
named 'Counter'

9/18/2017

Sacramento State - Cook - CSc 35 - Fall 2017

65

## 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...



9/18/2017

Sacramento State - Cook - CSc 35 - Fall 2017

66

## Instruction: Add

Immediate, Register, Memory

**ADD** *value* , *target*

Register, Memory

9/18/2017

Sacramento State - Cook - CSc 35 - Fall 2017

67

## Example: Move register to memory

Move memory into rax

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

*rax += 2;*

9/18/2017

Sacramento State - Cook - CSc 35 - Fall 2017

68

## 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...



9/18/2017

Sacramento State - Cook - CSc 35 - Fall 2017

69

## Instruction: Logical And

**AND** *value* , *target*

9/18/2017

Sacramento State - Cook - CSc 35 - Fall 2017

70

## Example: Logical Or

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

0011 0000

9/18/2017

Sacramento State - Cook - CSc 35 - Fall 2017

71

## 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

9/18/2017

Sacramento State - Cook - CSc 35 - Fall 2017

72

## Instruction: Call

**CALL** *address*

Usually a label – an constant  
that holds an address

9/18/2017

Sacramento State - Cook - CSC35 - Fall 2017

73

## Example: Print an integer

#This is using the CSC35 library

MOV \$42, %rax

CALL PrintInt

This name is an address

9/18/2017

Sacramento State - Cook - CSC35 - Fall 2017

74