CSc 35- Spring 2018- EXAM 1 Study guide

SAC STATE FOUNDED ON 1947 – CA ESTABLISHED ON 1850

Binary & hex numbers:

- ***Binary numbers***
  - use powers of 2 rather than 10

The number 1010 1001 is ...

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
128	64	32	16	8	4	2	1
1	0	1	0	1	0	0	1

128 + 32 + 8 + 1 = 169

- ***Hexadecimal is base-16***
  - We only have 0...9 to represent digits
  - So, hexadecimal uses A...F to represent 10...15

Hex	Decimal	Binary
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111

Hex	Decimal	Binary
8	8	1000
9	9	1001
A	10	1010
B	11	1011
C	12	1100
D	13	1101
E	14	1110
F	15	1111

Converting Binary to Hex = Easy

- Since 16 = 2^4, a single hex character can represent a total of 4 bits
- Byte can be represented with only 2 hex digits!
- When looking at raw data, editors, called Hex Editors, display data as groups of 2 hex digits

5				C			
0	1	0	1	1	1	0	0

Notation (why important): Hexadecimal and binary notations use the same digits we use for decimal; as a result, some numbers look like valid hex, decimal and binary numbers.

- For example is **101** …
  - binary value 5?
  - decimal value 101?
  - hexadecimal value 257?
- This, obviously, can become problematic

Prefix Notation

- There are also prefix notations that are commonly used.
- Using prefix characters "b" and "h"…
  - **h**101 – hexadecimal
  - **b**101 – binary
  - 101 – just decimal

Subscript Notation

- Commonly, textbooks use a subscript to denote the base
- Examples
  - 101_{16} – hexadecimal, and equal to 257
  - 101_{2} – binary, and equal to 5
  - 101 – decimal
- However, this is not possible to do in common text editors

Postfix Character Notation

- Examples
  - 101**h** – hexadecimal, and equal to 257
  - 101**b** – binary, and equal to 5
  - 101 – just decimal
- Remember to use a lower case "b"
  - "B" is the hex digit for 11
  - someone could read 101B has hex

C-Style Prefix Notation (POPULAR)

- C's notation
  - the prefix "0x" denotes hexadecimal
  - but it lacks a binary notation
  - so, "0b" typically denotes binary
- Examples:
  - 0x101 is hexadecimal
  - 0b101 is binary
  - 101 is decimal

- The C Programming Language's notation is often used
- C is hugely popular and multiple languages are based on its syntax – e.g. Java, C#

ASCII: Created in the 1967

- 7 bits – 128 characters
- uses a full byte, one bit is not used
- ASCII is only good for the United States
- Alphabetic characters (uppercase and lowercase) are 32 ($32 = 2^5$) "code points" apart
  - Uppercase and lowercase letters are just 1 bit different

	Binary	Hex
A	01000001	41
a	01100001	61

Integers:

- Integer data types are stored in simple binary numbers
- The number of bytes used varies: 1, 2, 4, etc….
- Languages often have a unique name for each – short, int, long, etc…

Components of the processor: IS MADE OF TWO PARTS!

- 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

Execution Unit (EU): Different in many processors	**Control Logic Unit (CLU)**
KEY FEATURES 1) performs calculations & logic 2) registers hold data	***KEY FEATURES*** 1) reads and decodes instructions 2) talks to other components
• Contains the hardware that executes tasks • Modern processors often use multiple execution units to execute instructions in parallel to improve performance	• Controls the processor • Determines when instructions can be executed • Controls internal operations fetch and execute each instruction and store result of each instruction
Execution Unit – The ALU	
• The Arithmetic Logic Unit performs all calculations and comparisons • Processor often contains special hardware for integer and floating point	

Privileged mode:

- privileged – only the processor and OS can change it.

Types of operands: (operands – what data is to be used)

- Registers
- Memory address
- Register pointing to memory
- An immediate: A constant stored with the instruction
    - mov $5, %rax

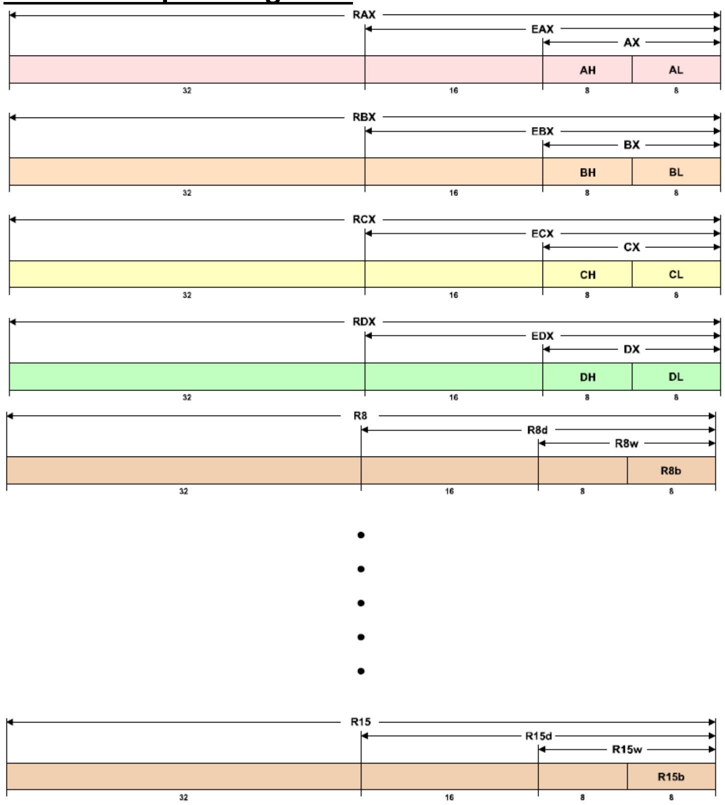
Types of opcodes:

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

x86 Registers: First "x86" was the Intel 8086 released in 1978

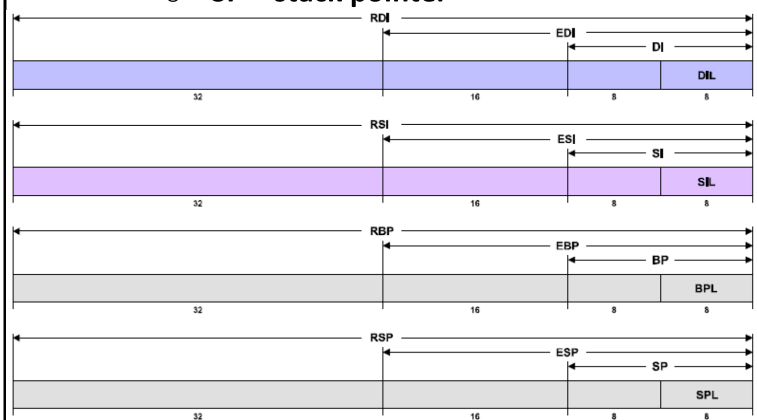
- **8 Registers can be used by your programs**
  - **Four General Purpose: AX, BX, CX, DX**
  - **Four pointer index: SI, DI, BP, SP**
- **The remaining 8 are restricted**
  - **Six segment: CS, DS, ES, FS, GS, SS**
  - **One instruction pointer: IP**
  - **One status register – used in computations**

General Purpose Registers



pointer index

- **Used for storing indexes and pointers**
- **Their purpose**
  - **DI – destination index**
  - **SI – source index**
  - **BP – base pointer**
  - **SP – stack pointer**



Compilers: Convert programs from high-level languages (such as C or C++) into assembly language.

Assemblers: Converts assembly into the binary representation used by the processor.

Linkers: Joins multiple parts (usually object files) into a single file.

What a Linker Does

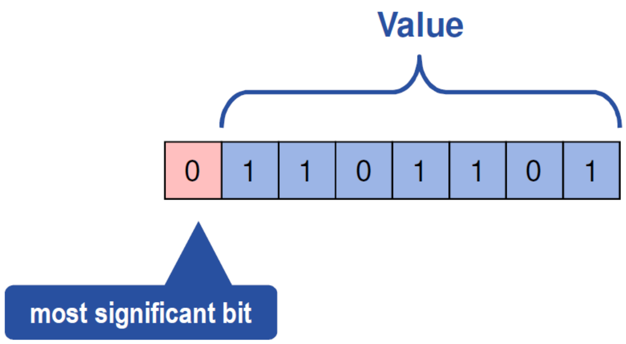
- Connects labels (identifiers) - used in one object - to the object to that defines it.
- So, one object can call another object.
- What you will see: label conflicts and missing labels.

Assembly concepts:

<u>Sign-magnitude</u>: MOST SIGNIFICANT BIT TELLS YOU IF IT IS POSITIVE OR NEGATIVE

<table>
<tr><td rowspan="2">

One approach is to use the most significant bit (msb) to represent the negative sign
If positive, this bit will be a zero
If negative, this bit will be a 1
This gives a byte a range of -127 to 127 rather than 0 to 255



</td><th>Drawbacks</th></tr>
<tr><td>

When two numbers are added, the system needs to check and sign bits and act accordingly
For example:

if both numbers are positive, add values
if one is negative subtract it from the other

There are also rules for subtracting

</td></tr>
</table>

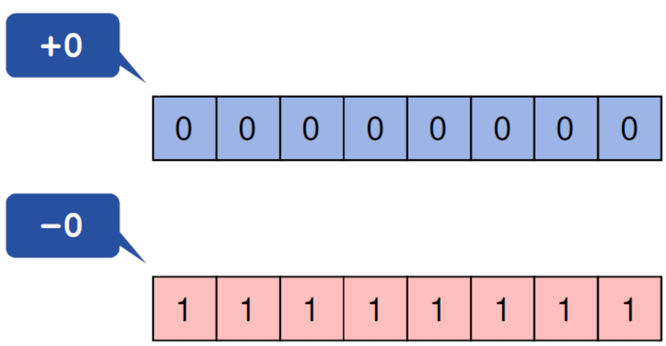
<u>One's complement</u>: FLIP THE BITS

<table>
<tr><td rowspan="2">

Rather than use a sign bit, the value can be made negative by inverting each bit

each 1 becomes a 0
each 0 becomes a 1

Result is a "complement" of the original
This is logically the same as subtracting the number from 0



</td><th>Advantages / Disadvantages</th></tr>
<tr><td>
<u>Advantages over signed magnitude</u>

very simple rules for adding/subtracting
numbers are simply added: 5 - 3 is the same as 5 + -3

<u>Disadvantages</u>

positive and negative zeros still exist
so, it's not a perfect solution

</td></tr>
</table>

<u>Two's complement</u>: JUST ADD ONE TO THE FIRST BIT

<table>
<tr>
<td rowspan="2">

- **Practically all computers nowadays use 2's Complement**
- **Similar to 1's complement, but after the number is inverted, 1 is added to the result**
- **Logically the same as:**
    - **subtracting the number from 2n**
    - **where n is the total number of bits in the integer**

</td>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</table>

<u>Multiplication</u>:

<u>Division</u>:

<u>Sign Extension</u>: