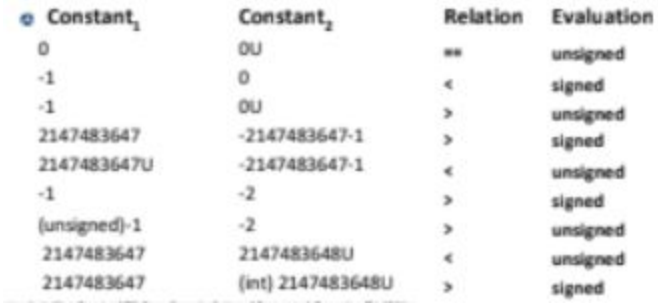
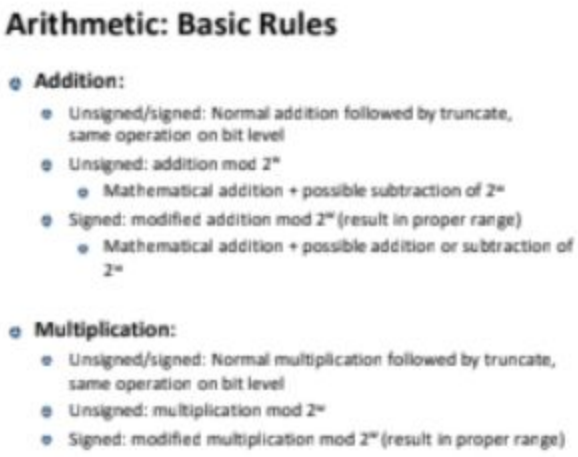
Topics:

- Bit Manipulation
  - Bits are 1, 0s
  - Ways to store data
  - 8 bits in a byte
  - |, &, ^, ~
  - Shifts → represent multiplication and division by power of two
    - Left shifts - always drag in zeroes
    - Right shifts
      - Arithmetic → leading ones if MSB is 1
      - Logical → always zeroes
- Integers (signed v. unsigned)
  - Signed v. Unsigned
    - Unsigned - 2^n numbers to represent
    - Signed - 2^{n-1} negatives and 2^{n-1} - 1 positives
      - Asymmetry with positives and negatives
      - Results in irregularities in arithmetic based on overflow errors
      - Positive to negative is negation of the bit string + 1
  - Casting
    - Casts from signed to unsigned for comparisons
  - Edge Cases
    - T_{min}

        - T_{max}
        - 0
        - -1
        - U_{max}
    - x < 0 → ((x*2) < 0)
    - ux >= 0
    - x & 7 == 7 → (x<<30) < 0
    - ux > -1
        - -1 is casted to U_{max}
    - x > y → -x < -y
    - x * x >= 0
    - x > 0 && y > 0 → x + y > 0
    - x >= 0 → -x <= 0
    - x <= 0 → -x >= 0
    - (x|-x)>>31 == -1
    - ux >> 3 == ux/8
    - x >> 3 == x/8
    - x & (x-1) != 0
    - 

Constant$_1$	Constant$_2$	Relation	Evaluation
0	0U	==	unsigned
-1	0	<	signed
-1	0U	>	unsigned
2147483647	-2147483647-1	>	signed
2147483647U	-2147483647-1	<	unsigned
-1	-2	>	signed
(unsigned)-1	-2	>	unsigned
2147483647	2147483648U	<	unsigned
2147483647	(int) 2147483648U	>	signed

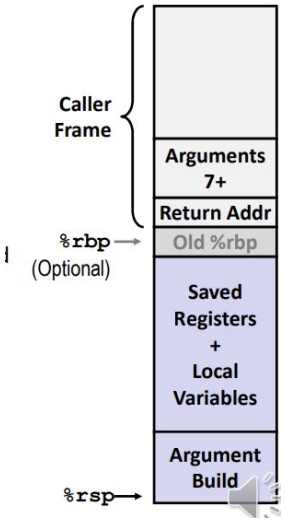
- Operands
    - Truncation → “Discard higher order bits”
    - Multiplication

    - Division
        - Rounding signed negatives round down instead of towards 0
    - Addition
        - Unsigned: Overflow goes back to zero
    - 

    - Strength reduction maybe exists
        - https://en.wikipedia.org/wiki/Strength_reduction
- Word Size
    - x86 used to have 16 bit word size
    - Moved up to 32 and then 64 bits
- Endian
    - Little endian - Store least significant bit in least valued memory address
    - Big endian - Store least significant bit in most valued memory address
    - Endians only apply to integral data types, not arrays or structs

- Sign Extension
    - To add bits but keep the number, extend the most significant bit leftward
- Unsigned extension
    - Add zeroes
- Structure of x86-64
    - CPU → Contains not memory
    - Memory → Stores stack, heap, text file, etc.
    - Registers → Faster access than memory
        - 16 registers on x86-64
        - Some are specialized (%rsp, %rip)
    - Condition codes → Store status info
    - Program counter → Same as %rip
- Compilation Process
    - C code is compiled into text assembly, which is then turned into binary, combined with a linker which provides static libraries to create an executable
- Registers
    - %rax is the return value
    - Parameters passed in %rdi, %rsi, %rdx, %rcx, %r8, %r9
        - Any other parameters go on the stack
    - Registers are 64-bit in x86-64, but have lower sections (%eax, %ax, %ah, %al, etc.)
- movq and leaq

    - Command that copies data from src to destination
        - Movq src, destination
    - Leaq
        - Load the address of the subject
        - No dereferencing
    - Cannot move memory into memory
- Memory Addressing Modes
    - [Mem] D(reg1, reg2, scale) =
        - reg1 + scale * reg2 + D
    - Scale is always a number like 4, 8, 16 etc.
- Condition Codes
    - CF → carry flag (unsigned overflow)
    - ZF → set if result is zero
    - SF → sign flag → set if f < 0
    - OF → overflow flag → set if signed overflow
    - Compare sets the flags
        - test - & operation
        - cmp - subtracts second input from first
- Conditionals (jumps and moves)
    - If-else
        - Regular jumps
    - Switch
        - Jump tables - indirect jump, addressing with scaling of 8

        - A set of addresses that can jump to various locations in code, destination selected based on conditionals
- Loops
    - Do-while
    - While
    - For
        - Can be generalized into similar structures
- Memory Structure
    - Stack starts at the highest address and grows "downwards"
    - 0xFFF… → 0x000...
    - %rsp is the top of the stack
    - How the stack works is called the "stack discipline"
    - Stack frames are created upon every function call
        - 

        - The space is all allocated beforehand

    - Heap
        - Dynamically allocated
    - Data
        - Stores static data like global variables and static variables
    - Text
        - Instructions, read-only
- Function Calls
    - Differentiation between callee-saved and caller-saved registers
- Array
    - Contiguous memory spaces, no endian
    - Differentiation in assembly between static-dimension, dynamic-dimension and multi-dimensional arrays
        - Dynamic-dimension arrays require imul instructions due to uncertainty of shifting
        - Multi-dimensional arrays use multiple levels of mov/lea to index
- Structs
    - Needs padding based on data type
- Unions
    - Needs to be the size of the largest data type