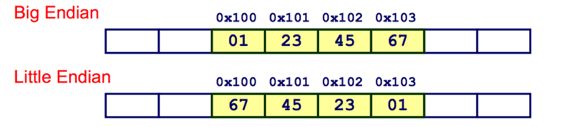
Transistors - implement logic gates.

Logic gates - manipulate binary numbers to implement Boolean functions.

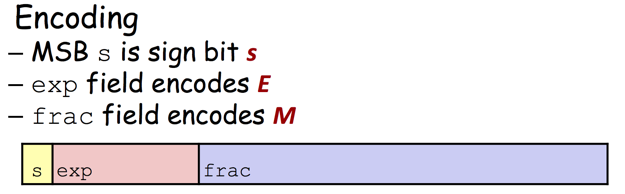
1 byte = 8 bits

0x01234567

int -> unsigned int -> long -> unsigned long -> long long -> unsigned long long -> float -> double -> long double

Registers are used to make our programs work with other programs and library.

Linkers allow us to use libraries and have multiple-file programs

Assembler is machine dependent

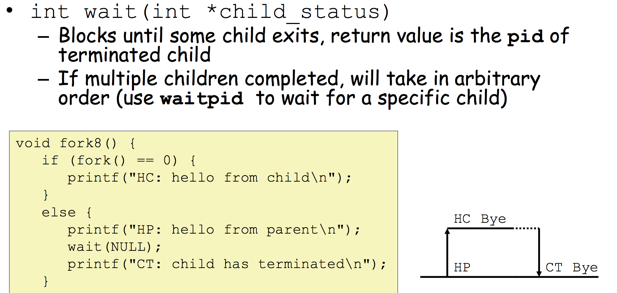
Complier is machine and language dependent

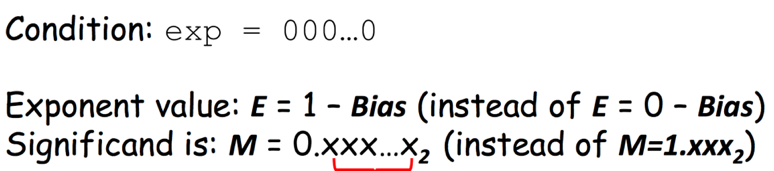
IEEE 754:

Normalized Encoding: Condition: exp ≠ 000…0 and exp ≠ 111…1

Single precision: E = exp – 127

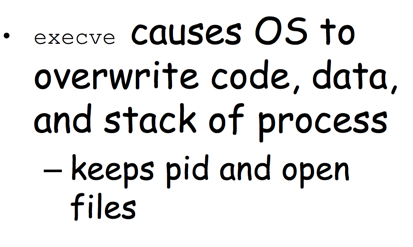
Double precision: E = exp – 1023 Fraction = 2^E \* M

Denormalized Encoding:



Special Values Encoding

Condition: exp = 111…1

Case: exp = 111…1, frac = 000…0

Represents value ∞ (infinity)

Operation that overflows

Case: exp = 111…1, frac ≠ 000…0

Not-a-Number (NaN)

Represents case when no numeric value can be determined

• Transfer data between memory and register – Load data from memory into register – Store register data into memory

• Transfer control – Unconditional jumps to/from procedures – Conditional branches

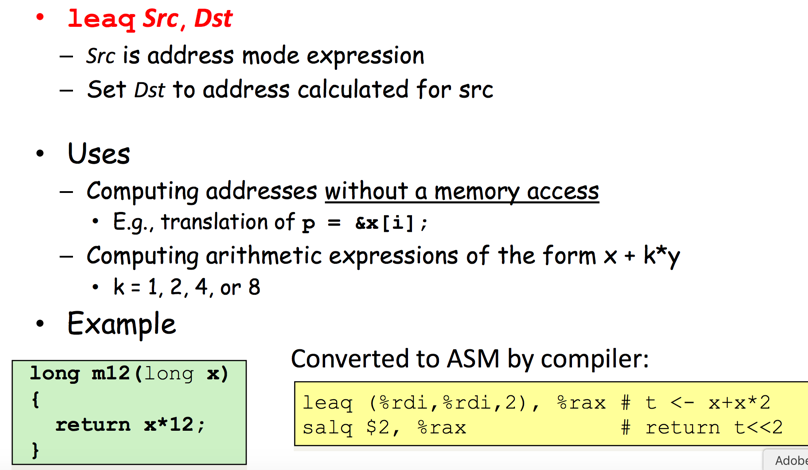
Two scenarios where (some) local variables of a function must be in the stack and not in registers?

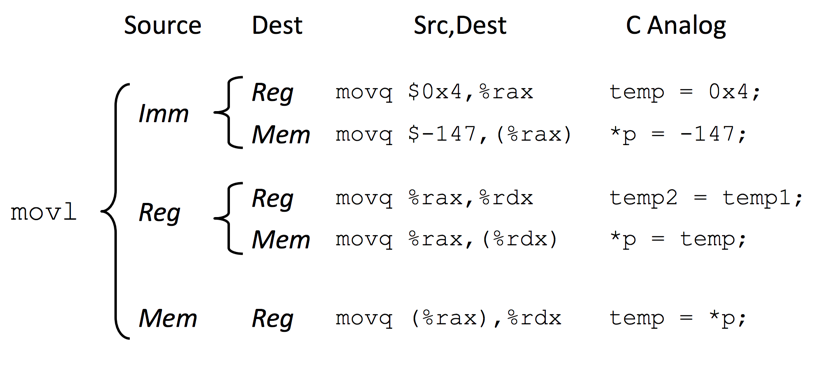
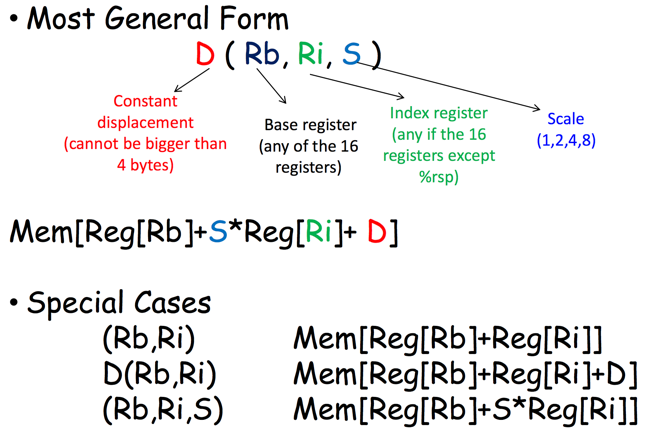
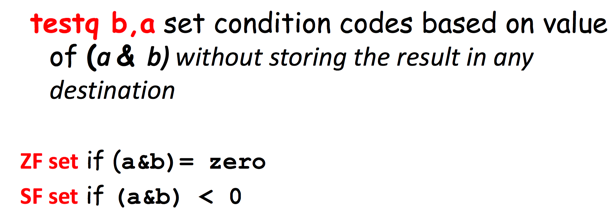
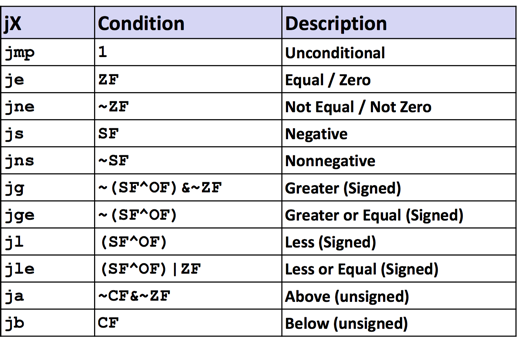
If we don’t have enough registers to hold all local variables needed.

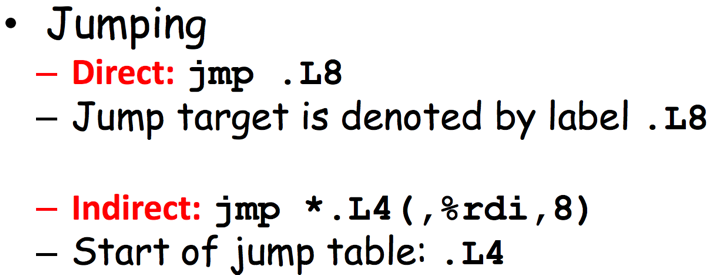
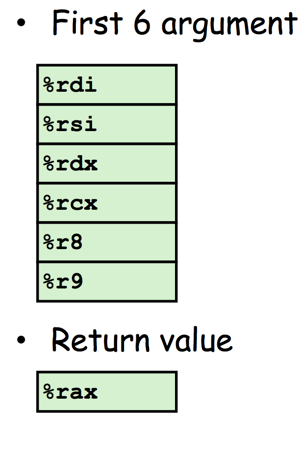
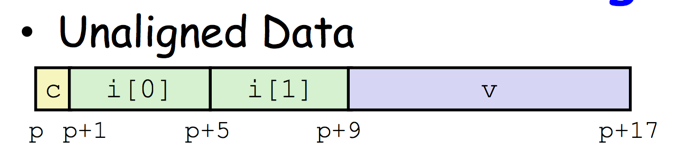
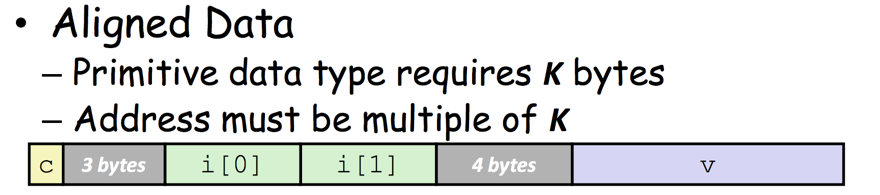
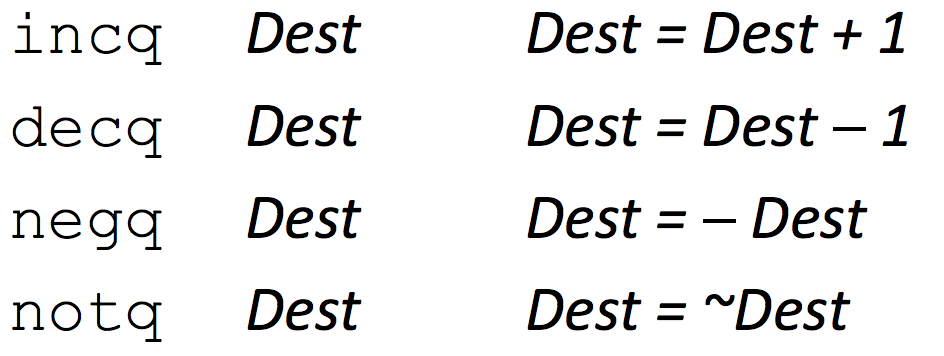
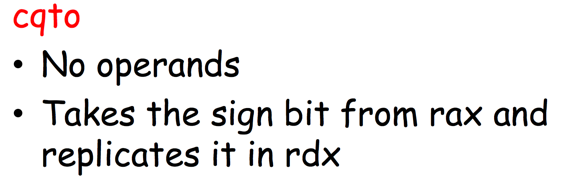
If we want to generate an address for a variable (in case we want to assign it to a pointer for example)

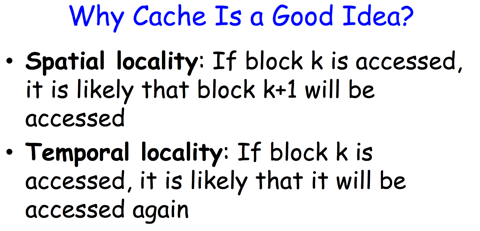
Macro vs function

Size: macro will make the code bigger in size, because each occurrence of the macro name will be substituted by the macro code. This is not the case with a function. • Speed: Calling a function is slower than a macro, because a function requires setting the stack, etc.

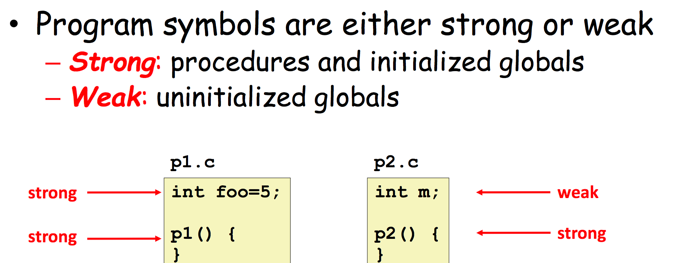
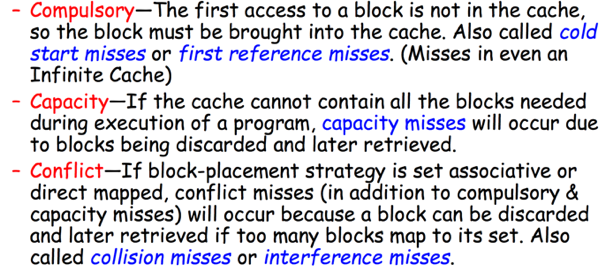




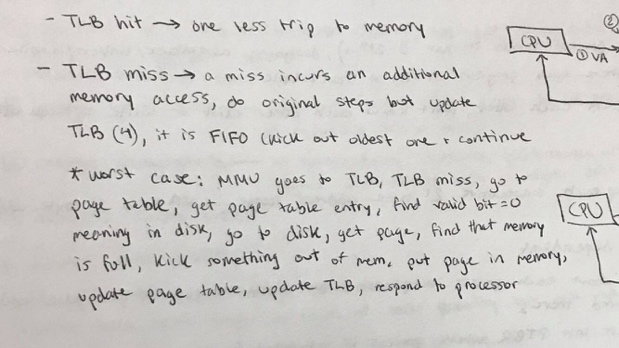
Switch case: Uses jump and indirect jump

Average access time = m + (1-p) M

m = cache acess time M = memory access time



Britannica, “Collectivization”

One fork is one processes

Log2(Block size) =offset

Log2(number of sets) = set

Address = tag +offset + set

TLB access virtual into physical

Associatively = blocks/ number of sets

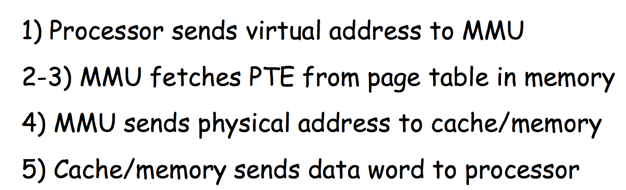
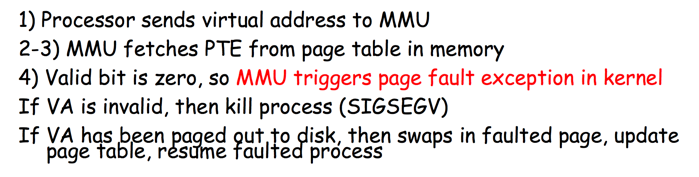
One page table per processes

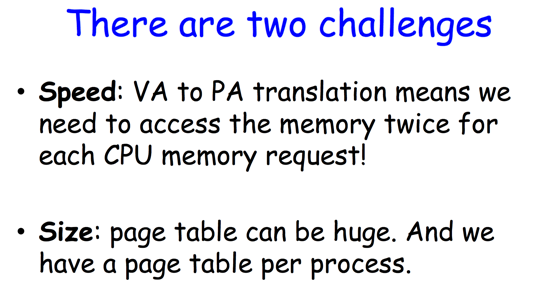
Virtual memory 3 advantages – illusion of more memory than actually have - Each process thinks it has the whole memory for itself – protect the processes from each other

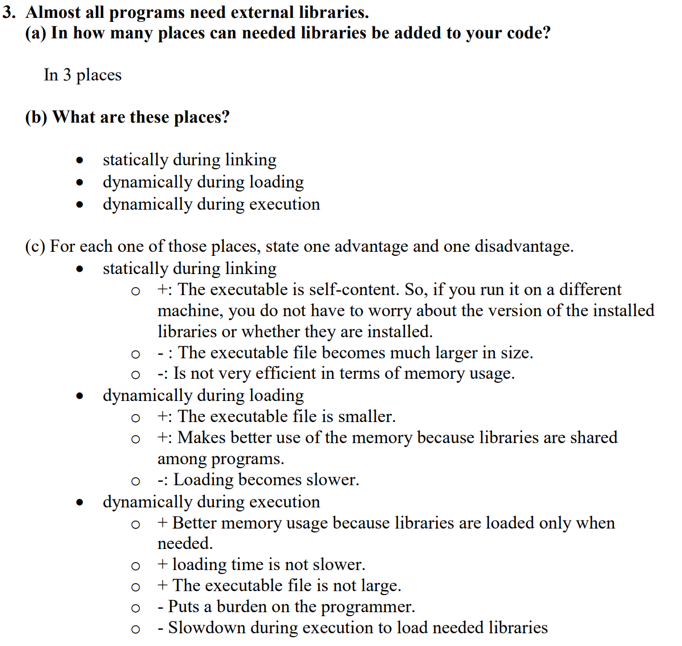
How Can We Reduce Misses?

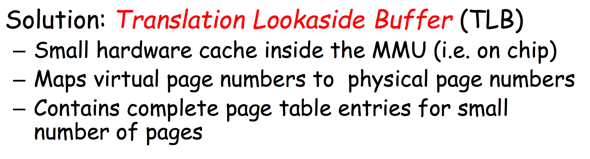
1) Increase Block Size: 2) Change Associativity: 3) Increase Cache Size

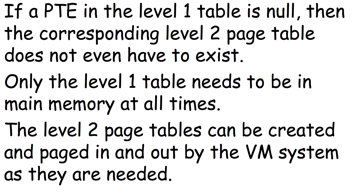
Page hit Page miss







 Speed: A TLB hit eliminates a memory access

Size:

