

Issues with one’s complement.  
- Arithmetic operations are cumbersome to implement.  
- E.g. when adding two numbers, if the second operand is negative, ordinary addition may give wrong answer

For two’s complement:

Sign bit contribution: (-22) in this example, and -2N-1 for N-bit integers.

-  There is one zero.

-  Unbalanced: the most negative integer has no positive counterpart.

for numbers in two’s complement, “negate” and “invert” are quite different!

Precedence(high to low): ~, &, |

sum‐of‐ product” canonical form:

Procedure:

* And together all literals (negate if 0) in each row (conjunct)
* Or together rows that have true output
* Repeat for each output bit of the function.

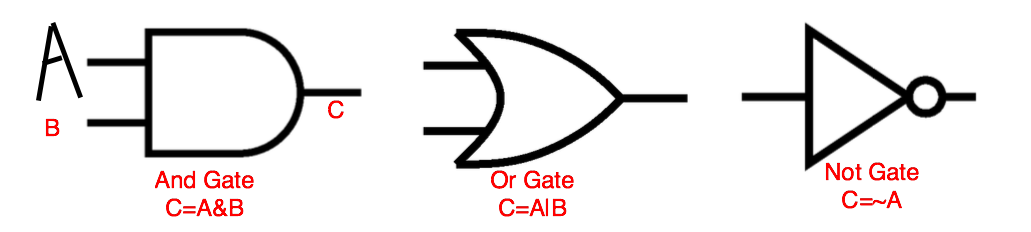
**Universality:** Every Boolean function, no matter how complex, can be expressed using three Boolean operators: AND, OR and NOT.

Universality: Any Boolean expression can be implemented using NAND gates only.

the “product-of- sum” form:

OR together all literals (negate if true) in each row AND together rows that have false output  
Repeat for each output bit of the function.

The number of Boolean functions that can be defined over n Boolean variables is 2^(2^n)



DeMorgan’s Law: ~(A&B)=~A|~B

Summary

• Any function on binary input/output can be implemented in Boolean logic

• Boolean logic can be implemented by physical devices – gates.

• Logic gates, as an abstraction, hide the physical details of the devices.

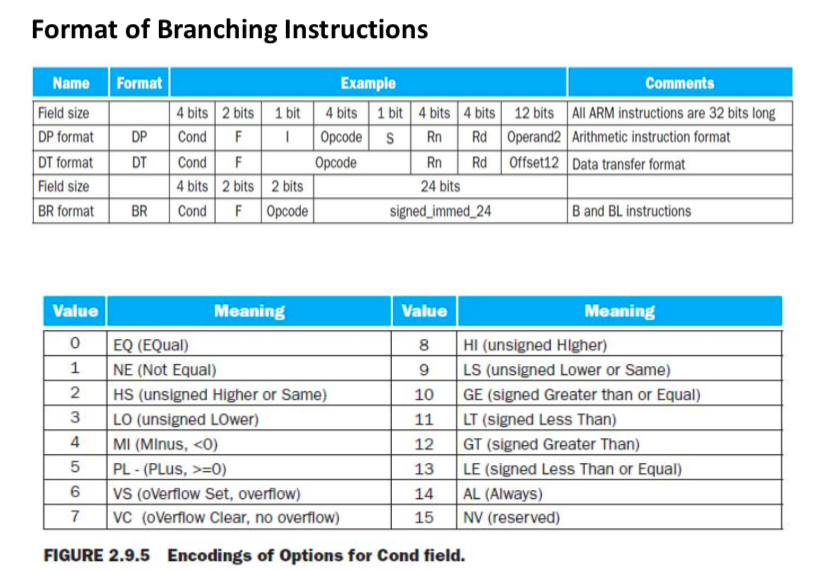
• Only need a small number of primitive gates, actually, only a single gate type NAND2 is enough.

logic devised can be of 2 types: one is combinational

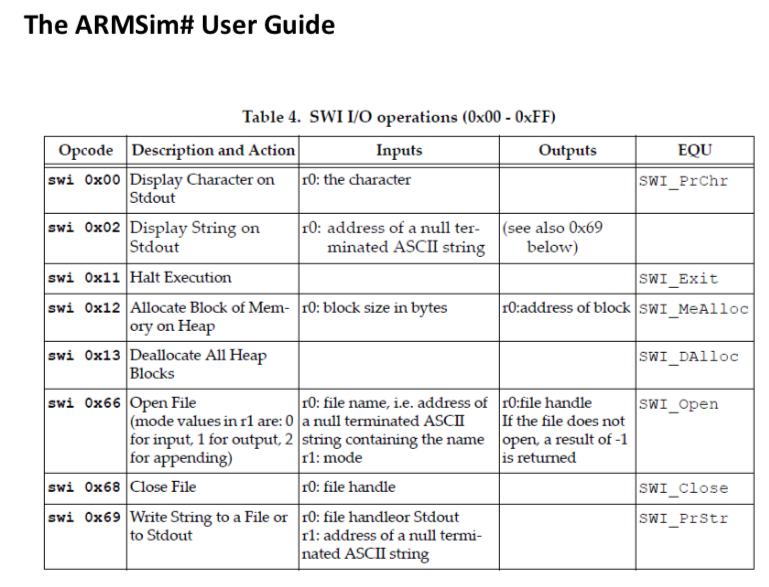
and the other is sequential. While for combinational, the

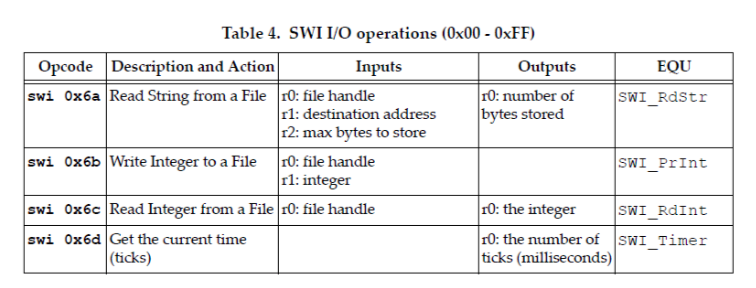
output is determined by the input solely, for sequential,

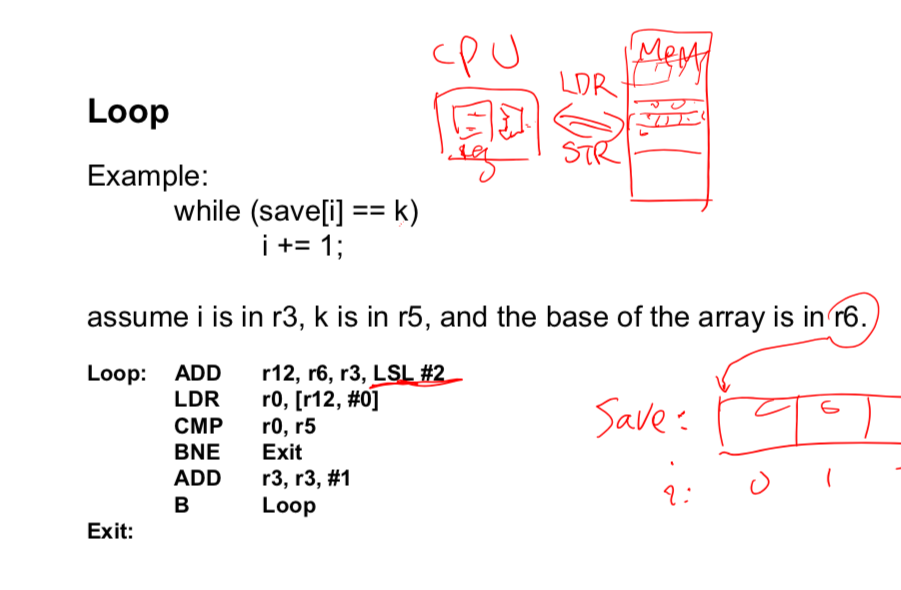
the output is determined by the order/history and the input

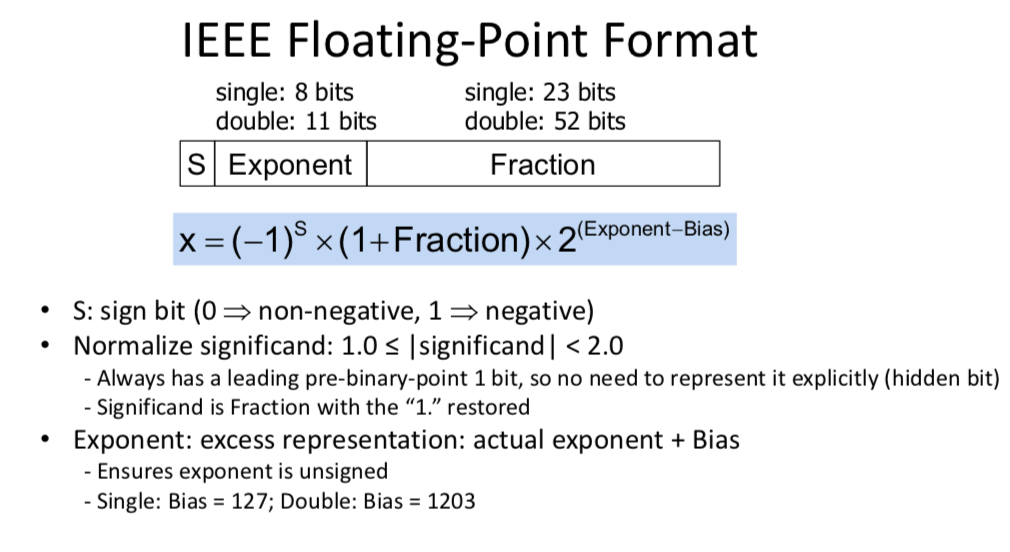


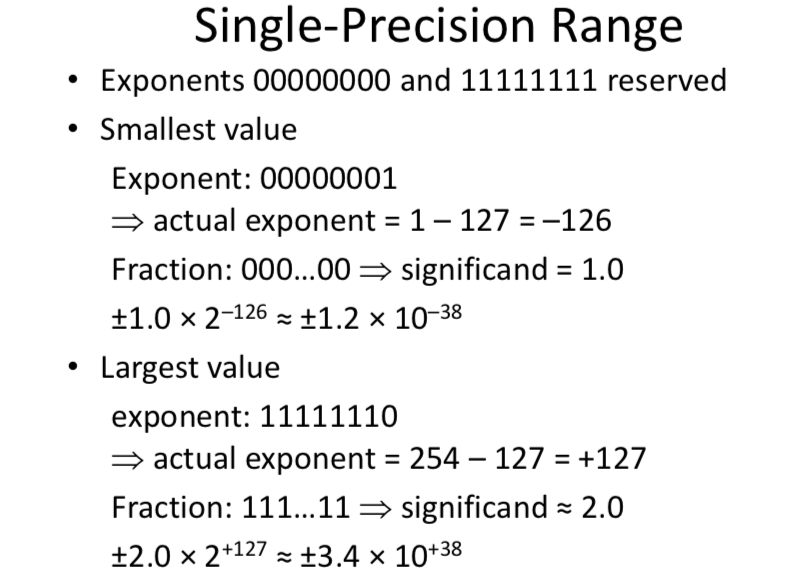


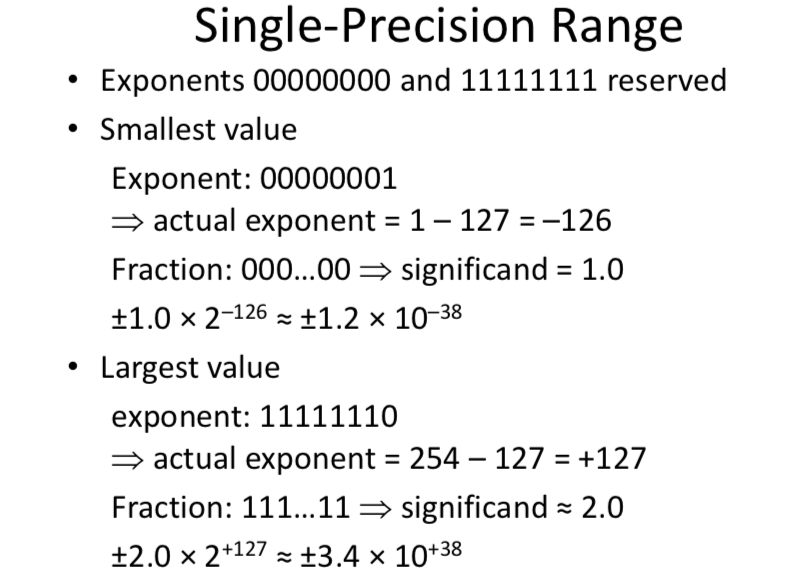




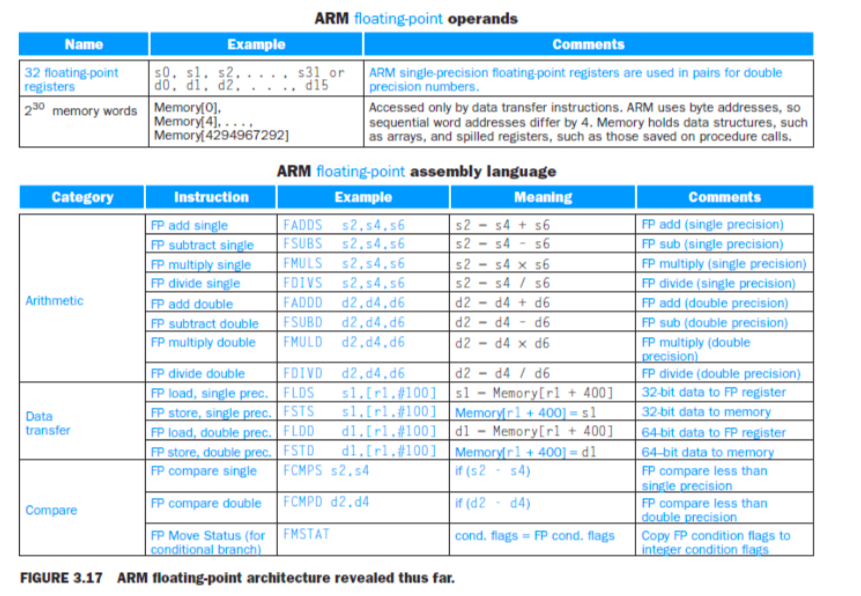


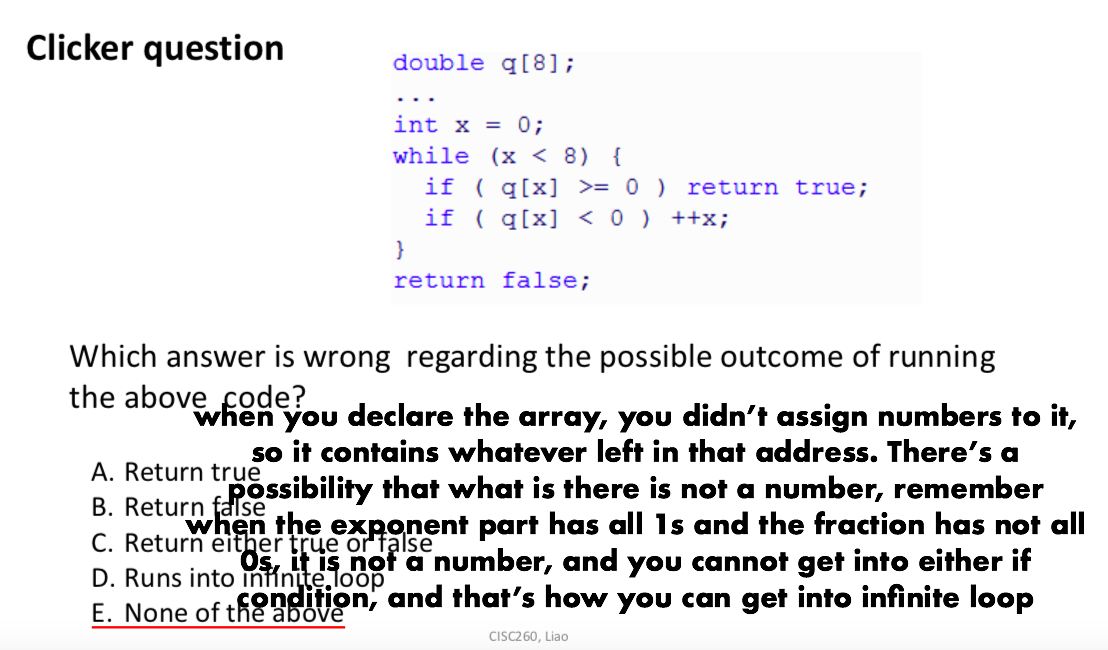


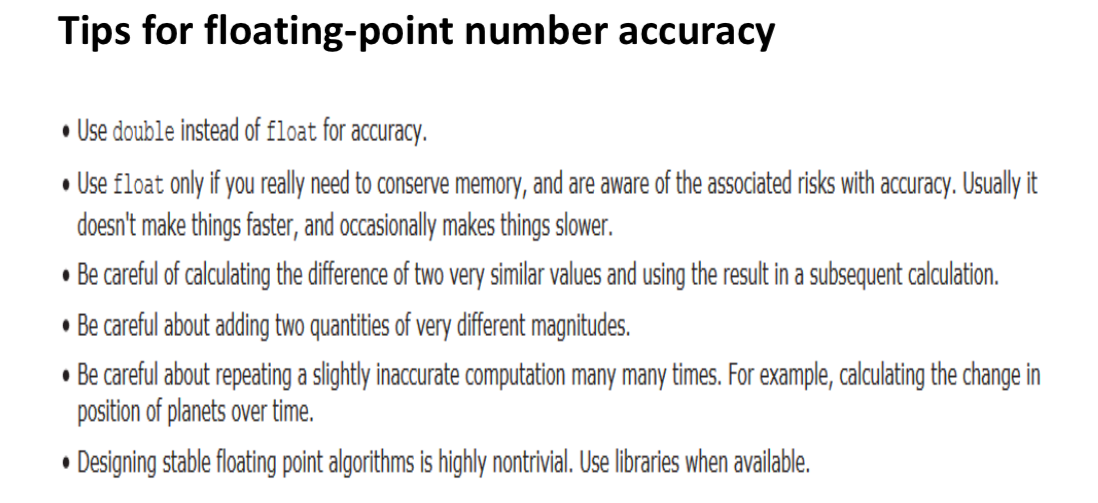


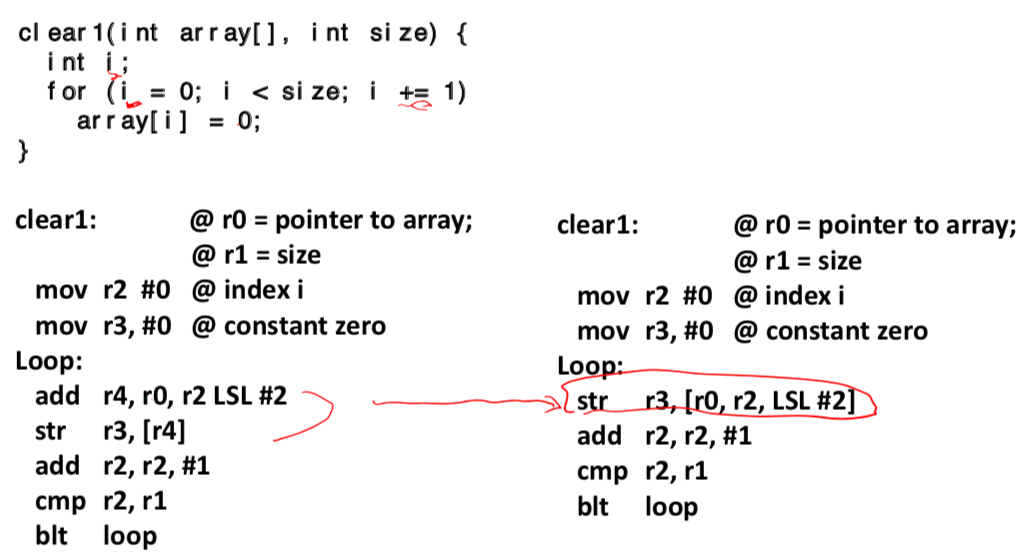




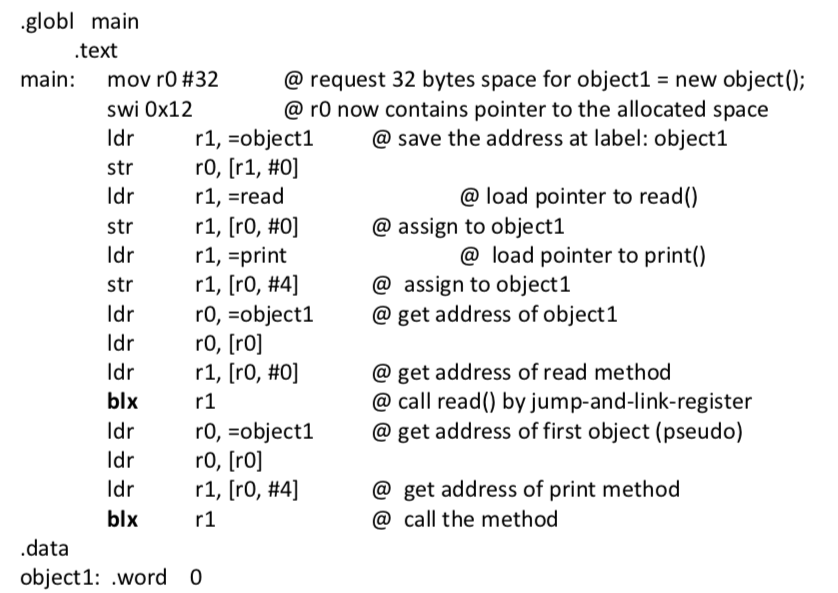


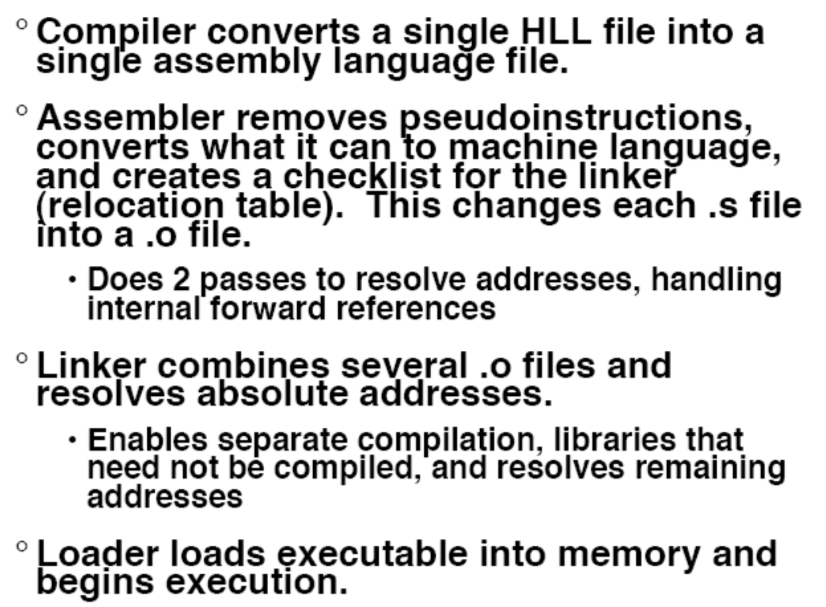


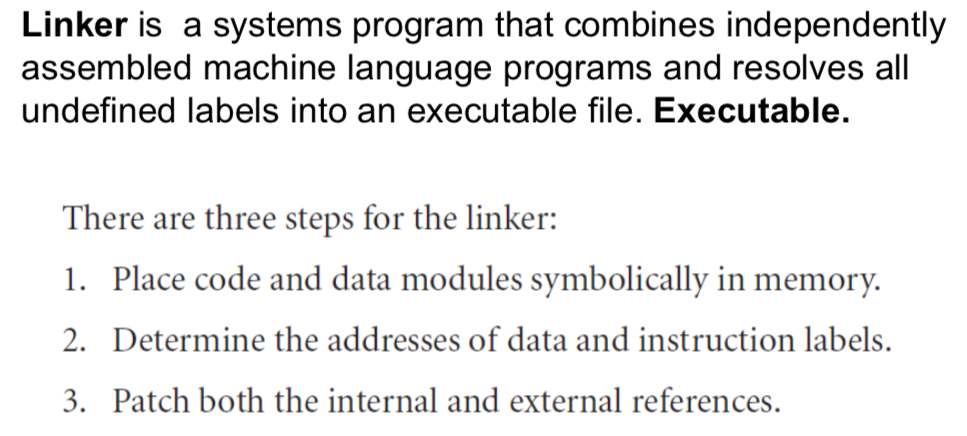


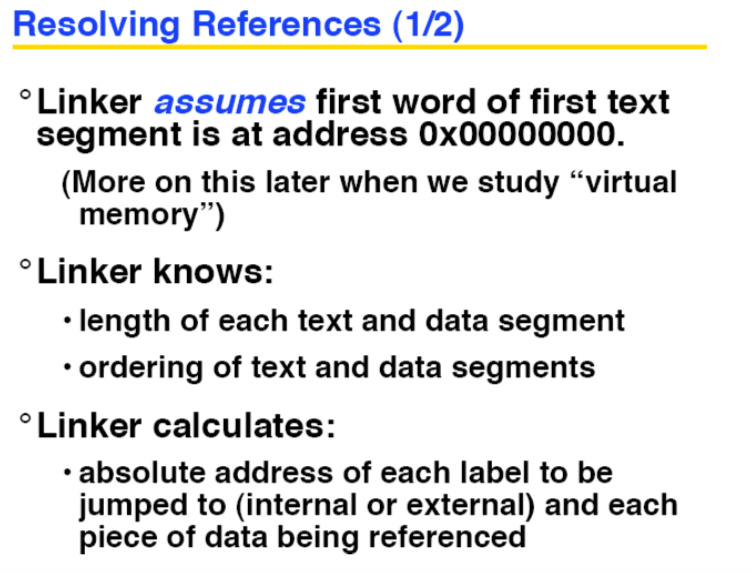


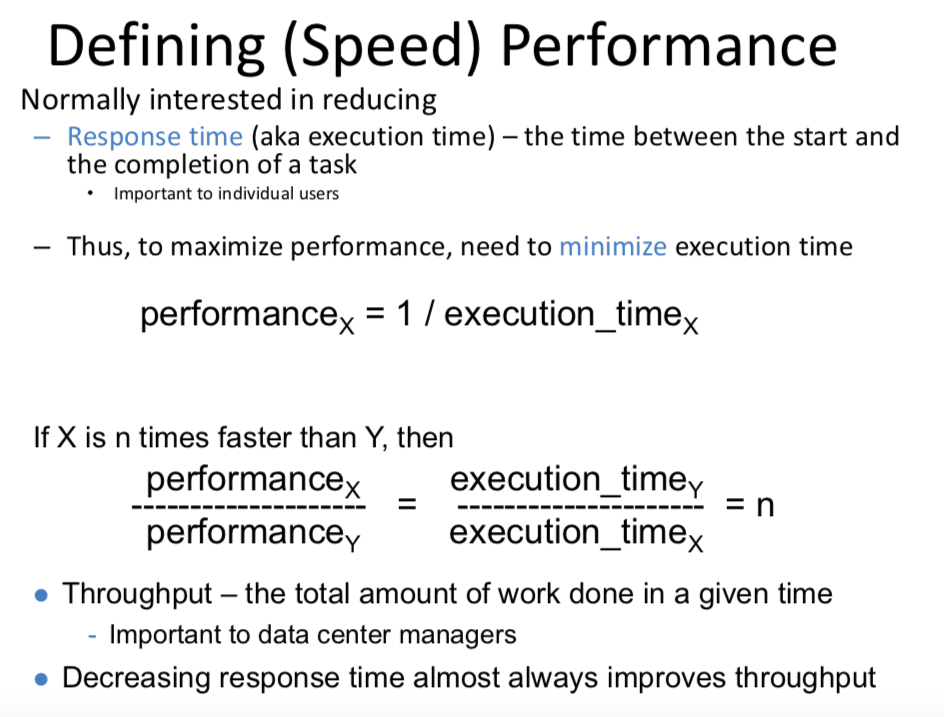


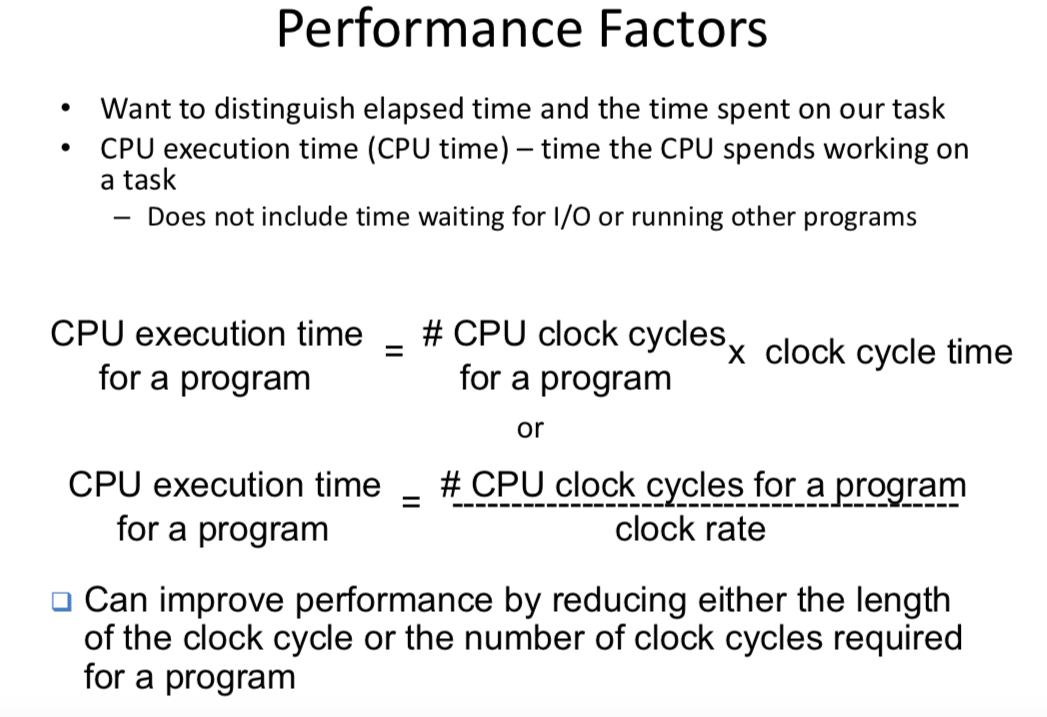


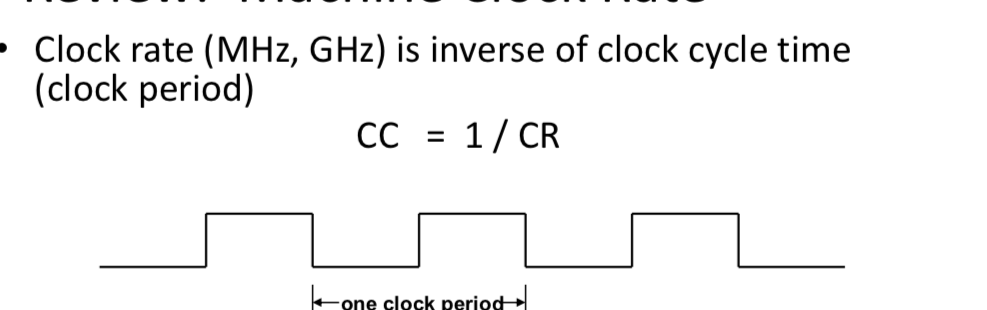




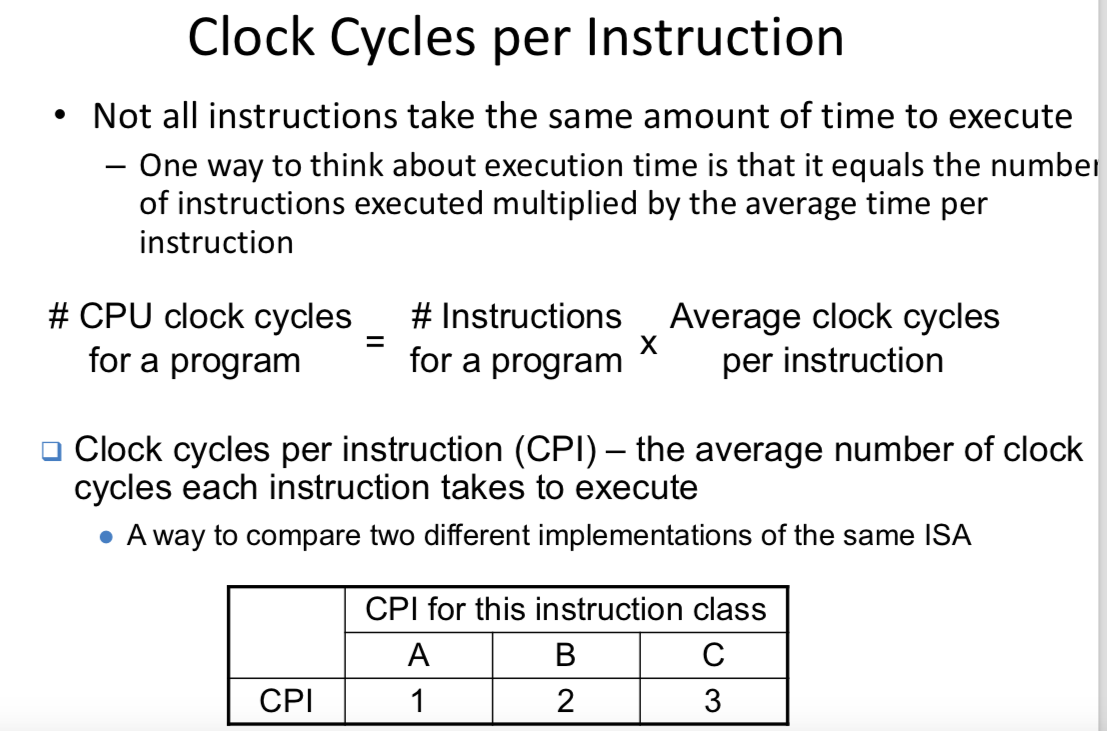








* Multiplication takes more time than addition
* Floating point operations take longer than that of integer
* Accessing memory takes more time than accessing registers
* *Important point: changing the cycle time often changes the number of cycles required for various instructions (more later)*



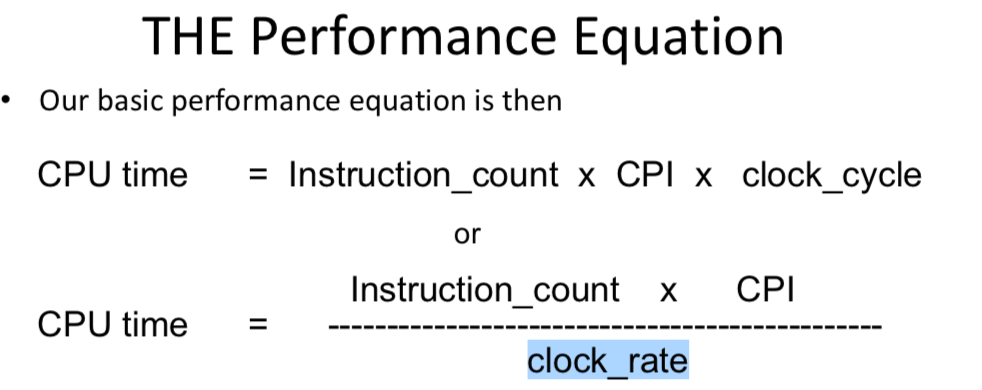
Computing the overall effective CPI is done by looking at the different types of instructions and their individual cycle counts and averaging

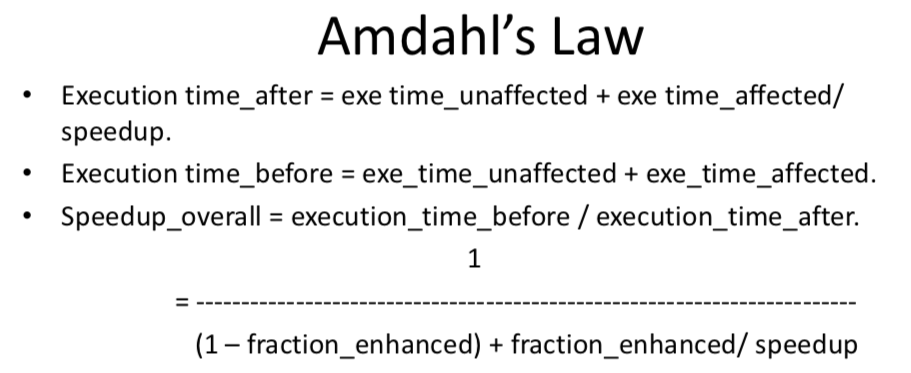
Overall effective CPI = Σ (CPIi x ICi) i=1

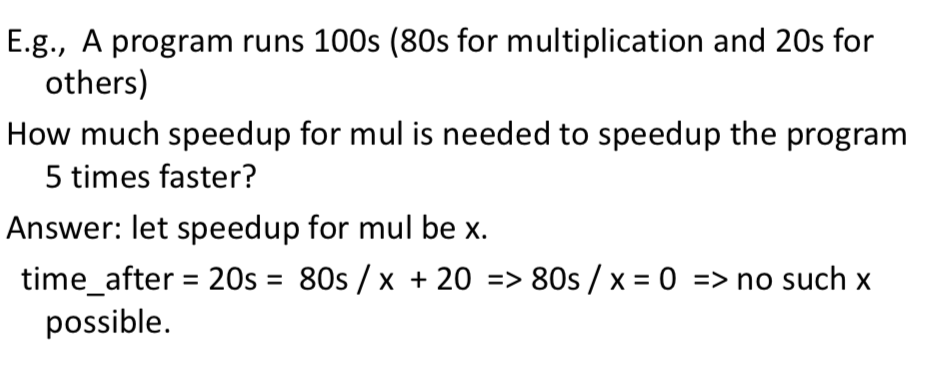
⚫Where ICi is the count (percentage) of the number of instructions of class i executed

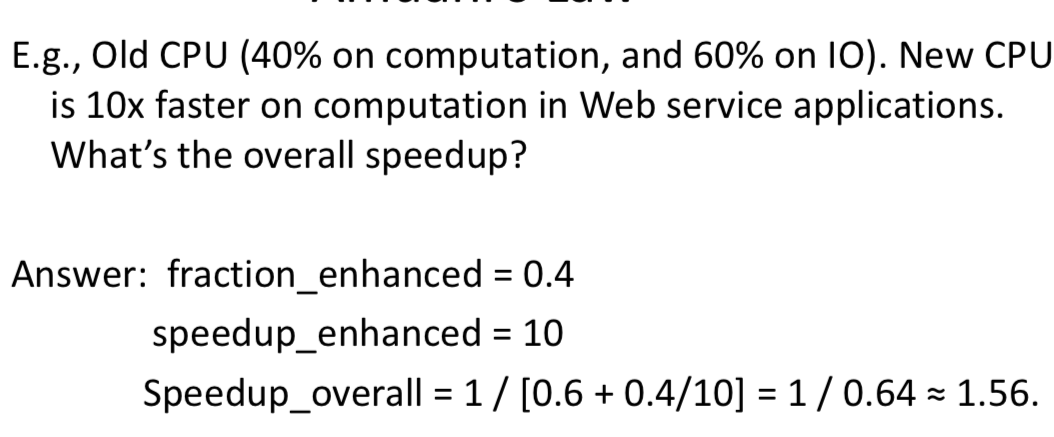
⚫CPIi is the (average) number of clock cycles per instruction for that instruction class

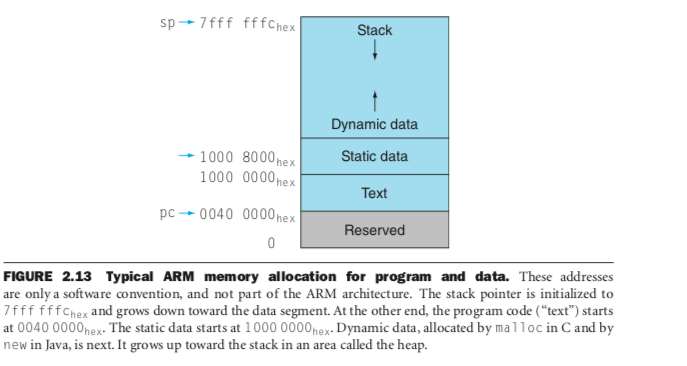
⚫n is the number of instruction classes











**Loop invariant**: instructions whose result does not change from iteration to iteration, and therefore can be moved outside the loop without affecting the semantics of the program.

Design philosophy of RISC v.s. CISC:  
CISC favors hardware solution, i.e, more instructions natively supported by hardware, whereas RISC favors fewer instructions and provides pseudoinstructions supported by the assembler to ease the programming.

