Comp. Arch Unit 2

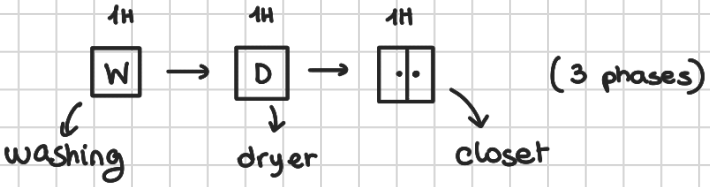
DATE: 13/04/2022

The architecture we've seen so far doesn't have a good performance.

You can get performance by:

- parallelism
  ↓
  PIPELINING (basic form of parallelism)

EXAMPLE: Suppose you have to do the laundry. You can split this in several phases.



Pipelining is useful when you have several instances of a process. If it's only one instance, then there's no point in pipelining.

So if you have "n" loads of clothes,

it will take "n+2" hours to finish
↳ you need 2 more hours to "empty" the dryer and all

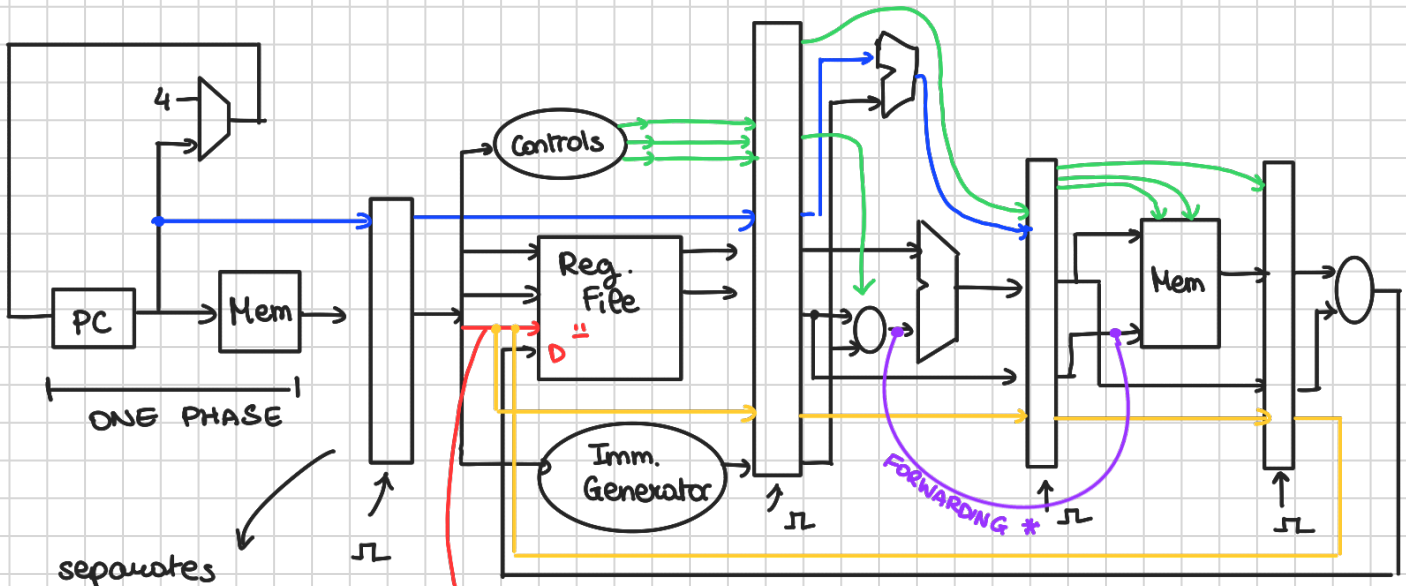
If you do it without pipelining → 3n

Usually written as ⇒ $\frac{3n}{n+2}$

SPEED-UP → how much it sped up by using a parallel solution

THREE REQUIREMENTS FOR PIPELINING

1. You must have a process that can be divided into phases.
2. Each phase has to use different hardware
3. Each phase needs to have approx. same time length.



separates one phase from another (PIPELINE REG.)

wrong because it has the rd of an instruction but the actual "D" (dest. register) needs the dest. reg. of another instruction (previous).

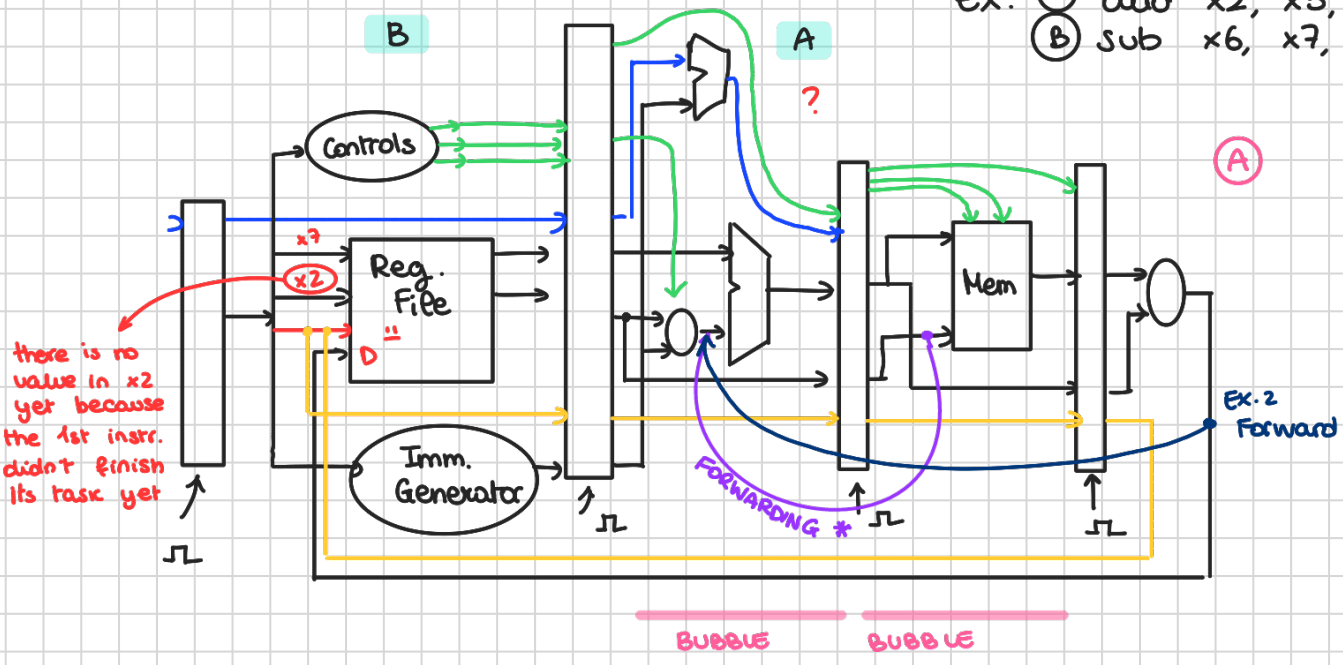
- we take this content along so that it can't be lost
- same thing with PC, we carry it along
- same thing with the controls (NOTE: not all controls are used in every stage)

(A) ADD x2, x3, x5
(B) SUB x6, x7, x8
(C) OR x9, x10, x11
(D) AND x12, x13, x14
(E) XOR x15, x16, x17

This is 5x faster than our previous architecture.

One probleem is that we might execute two instructions where the second one needs the result of the first one which won't be ready in time

EX: (A) add x2, x3, x5
(B) sub x6, x7, x2



There are 2 solutions to overcome this:

- by using BUBBLES. (clock cycle instr. B and A put in between the)

  We need to detect this situation and stall until the prev. instruction is done.
  We do this by putting bubbles in between

  For the example's sake, we will use 2 bubbles

  (refer to the pink lines on the image above.
  Try to eliminate the purple lines in your head)

- by using FORWARDING TECHNIQUE * (no need of bubbles)

  ↓

  basically, we replace wrong value with correct one by means of a MUX

We call these problems "HAZARD"

EX1
① add x5, x6, x7
② sub x8, x5, x11

or

EX2
① add x2, x3, x5
sub x6, x7, x9
and x11, x12, x2

PROBLEM: With "beq" instruction, when we jump, the instuctions next to beq will be discarted. What to do in this case? How can we reduce this loss of clocks/performance?

① LOOP UNROLLING

reduce number of branches.
Done by compiler

② Build a new hardware.
We recognize that there is a branch instr., we duplicate the architecture and we execute both of them (branch, no-branch)
After seeing the result, we discard the duplicate architecture

INTEL PENTIUM → has a lot of pipelining components
(2GH)