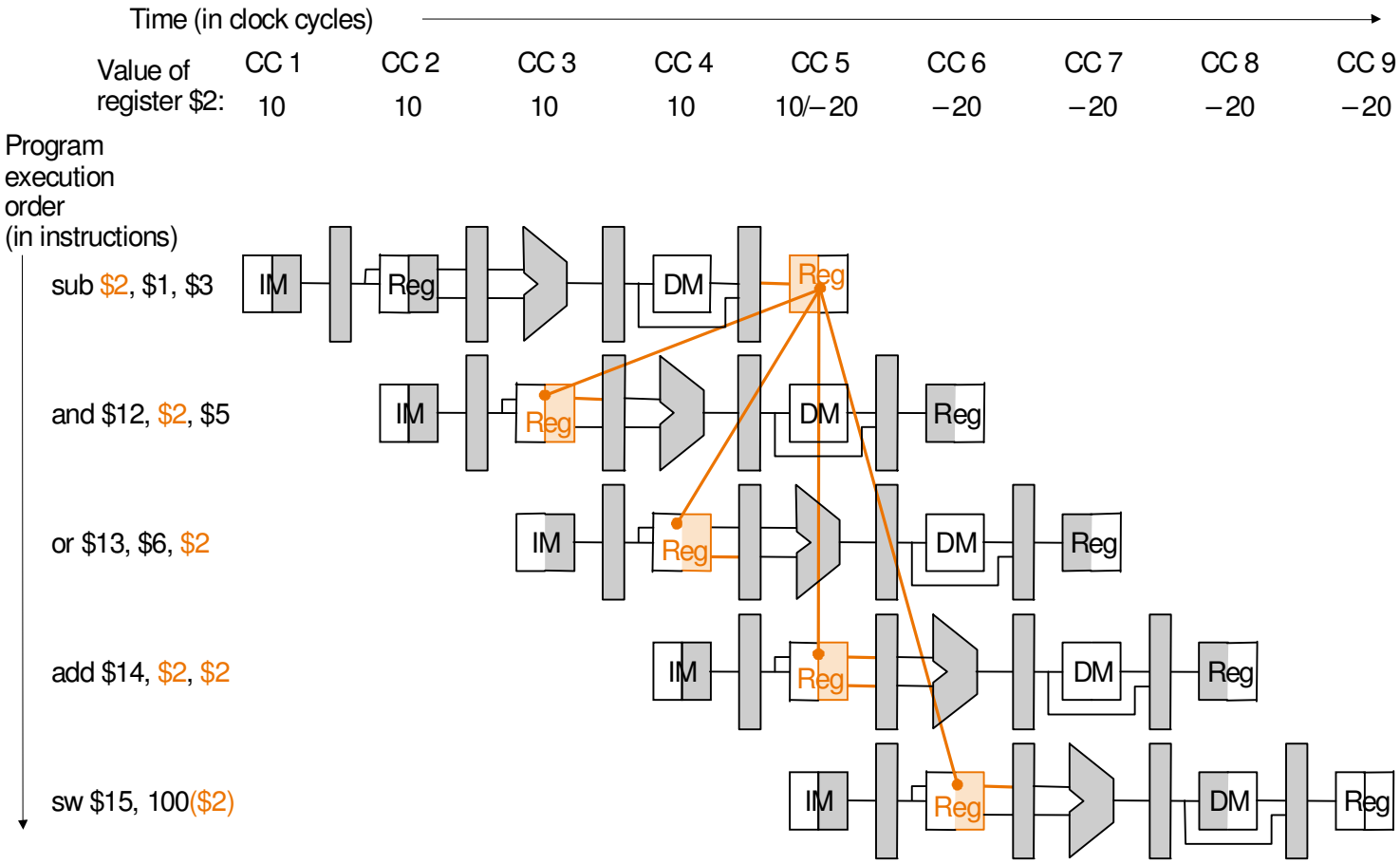
Chapter Six
Pipelining
Harzard

Data Dependencies

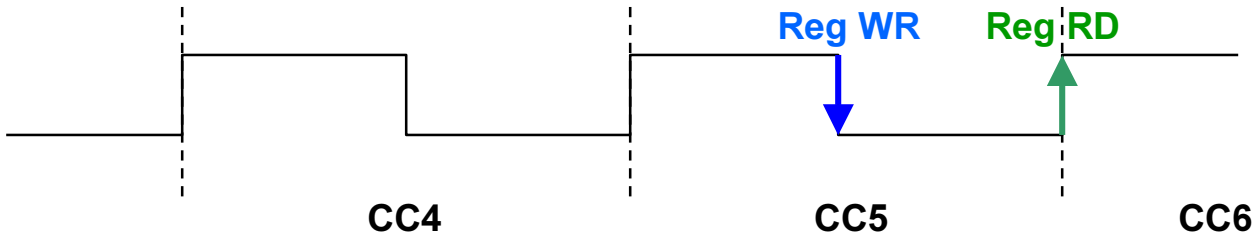
- **Problem with starting next instruction before first is finished**
  - **dependencies that "go backward in time" are data hazards**
  - **qual instrução receberá o valor errado de $2 (velho 10, novo -20)**



Erro devido à dependência de dados

```
sub    $2, $1, $3
and    $12, $2, $5
or     $13, $6, $2
add    $14, $2, $2
sw     $15, 100($2)
```

- and e or lêem resultado errado (velho 10)
- store está claramente à direita (depois no tempo) e lê resultado certo (-20)
- hazard no add pode ser evitado se a escrita no banco de registradores for feita (em CC5) na metade do ciclo (borda de descida)



Software Solution

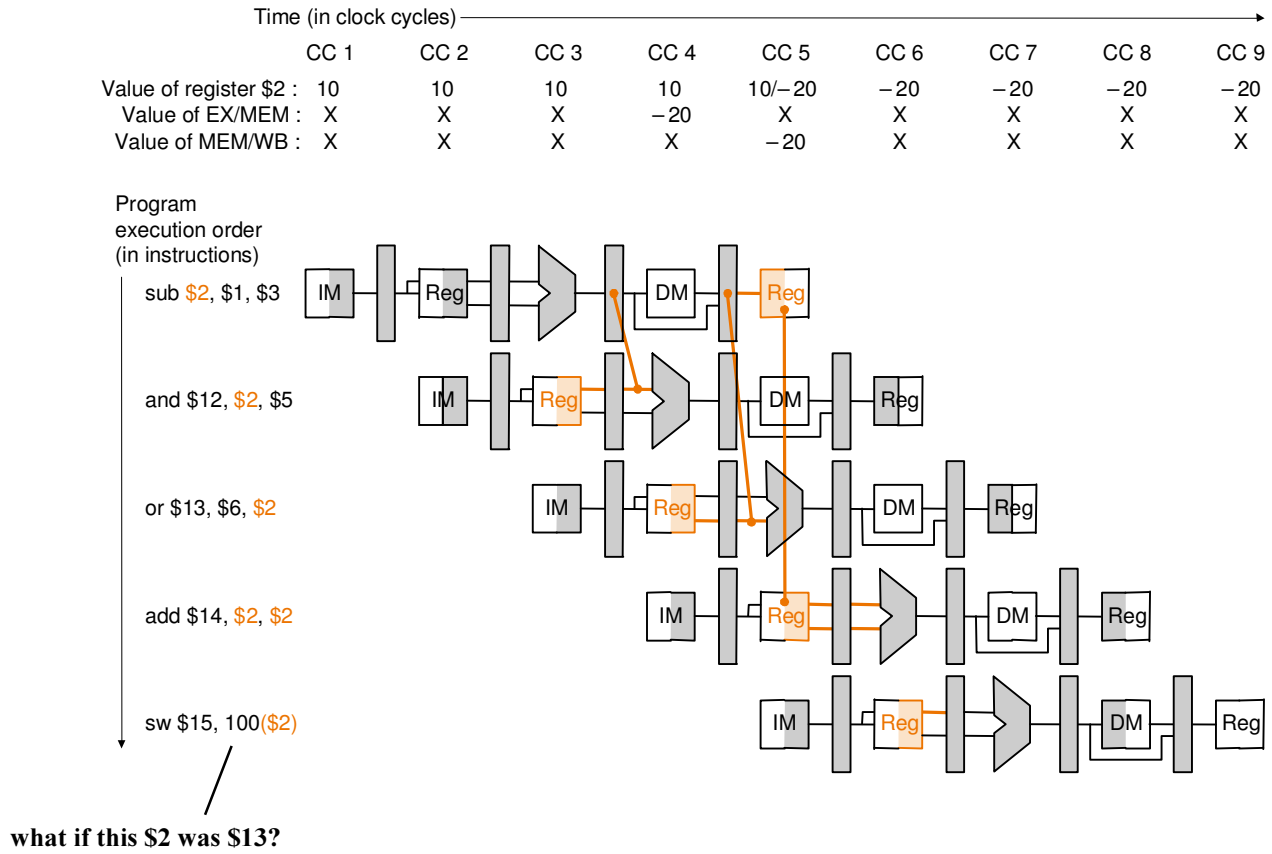
- **Have compiler guarantee no hazards**
- **Where do we insert the “nops” ?**
- **Quantos**

```
sub    $2, $1, $3
and    $12, $2, $5
or     $13, $6, $2
add    $14, $2, $2
sw     $15, 100($2)
```

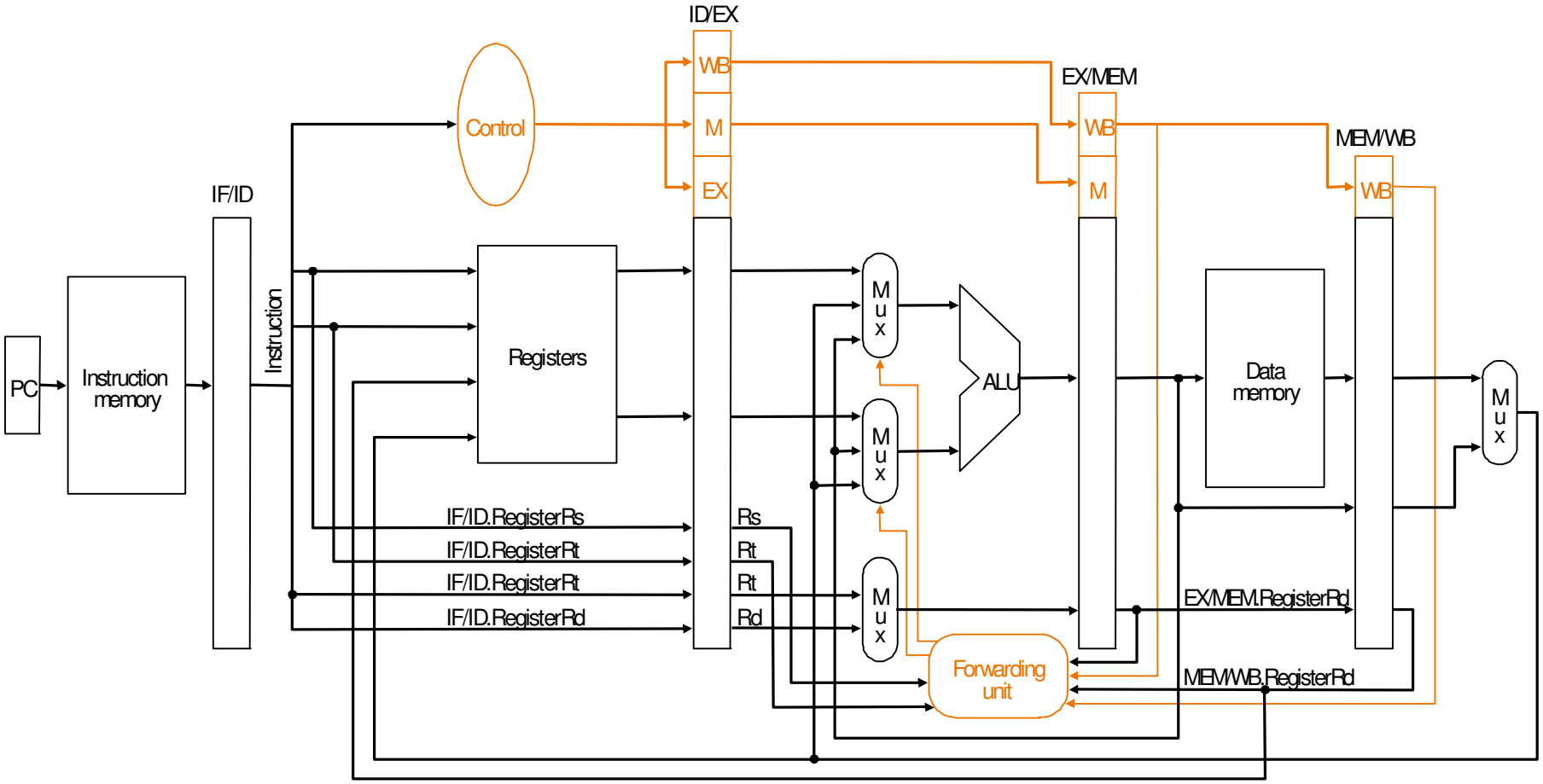
- **Problem: this really slows us down!**

Forwarding

- **Use temporary results, don't wait for them to be written**
  - **register file forwarding to handle read/write to same register**
  - **ALU forwarding**

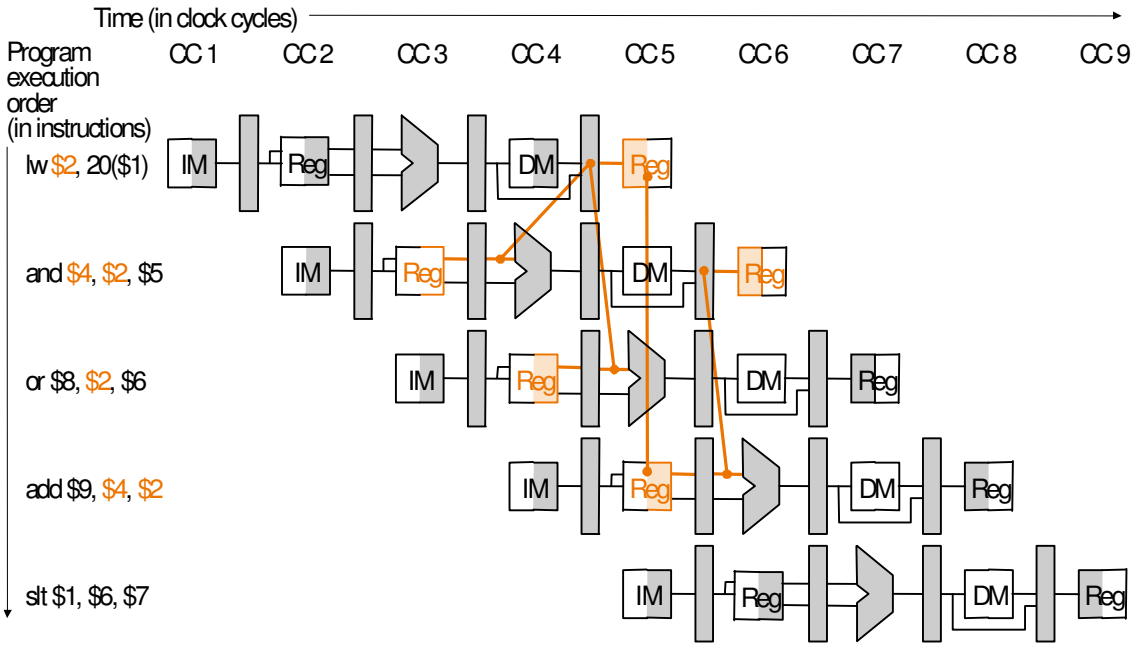


Forwarding



Can't always forward

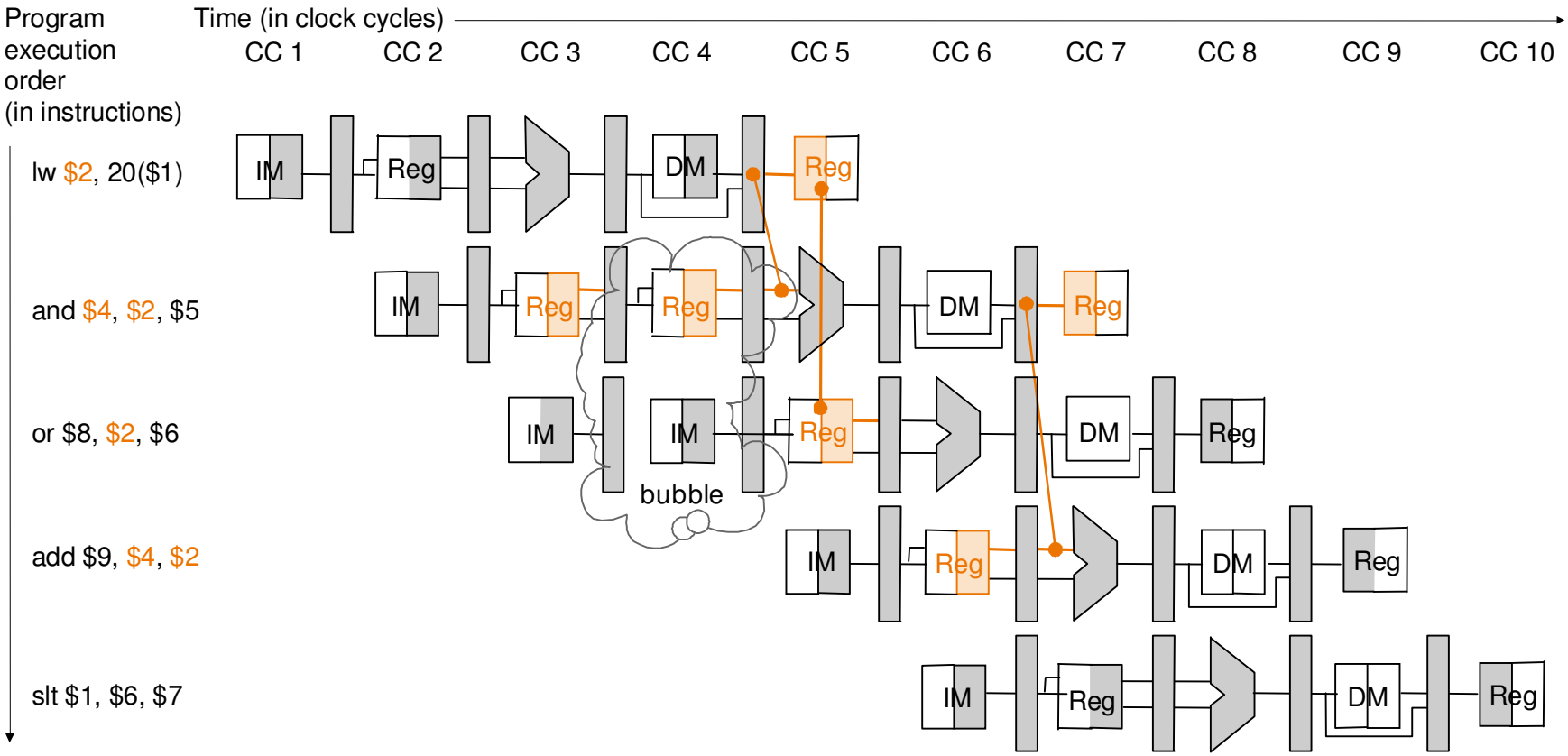
- **Load word can still cause a hazard:**
  - **an instruction tries to read a register following a load instruction that writes to the same register.**



- **Thus, we need a hazard detection unit to "stall" the load instruction**

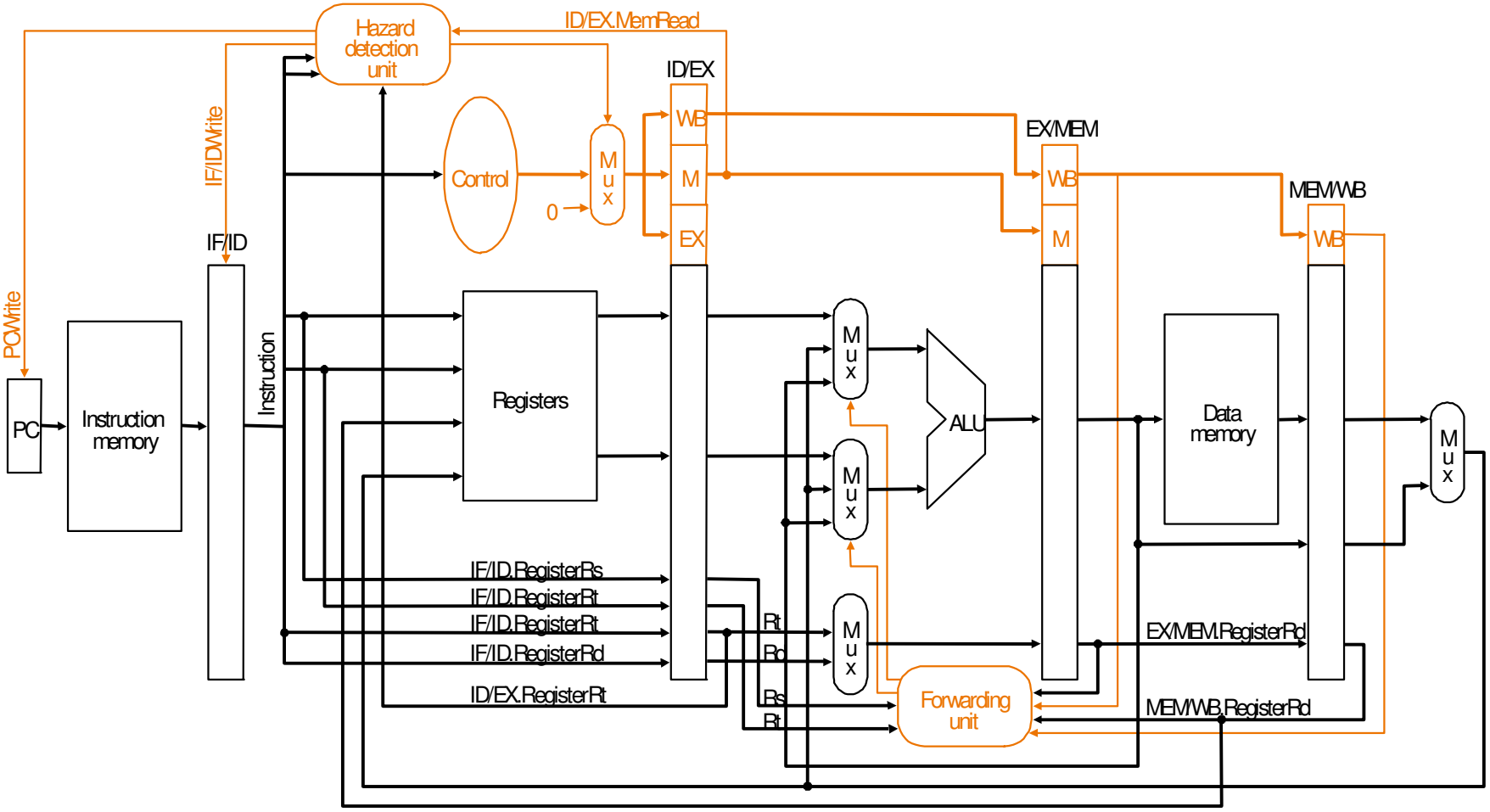
Stalling

- **We can stall the pipeline by keeping an instruction in the same stage**

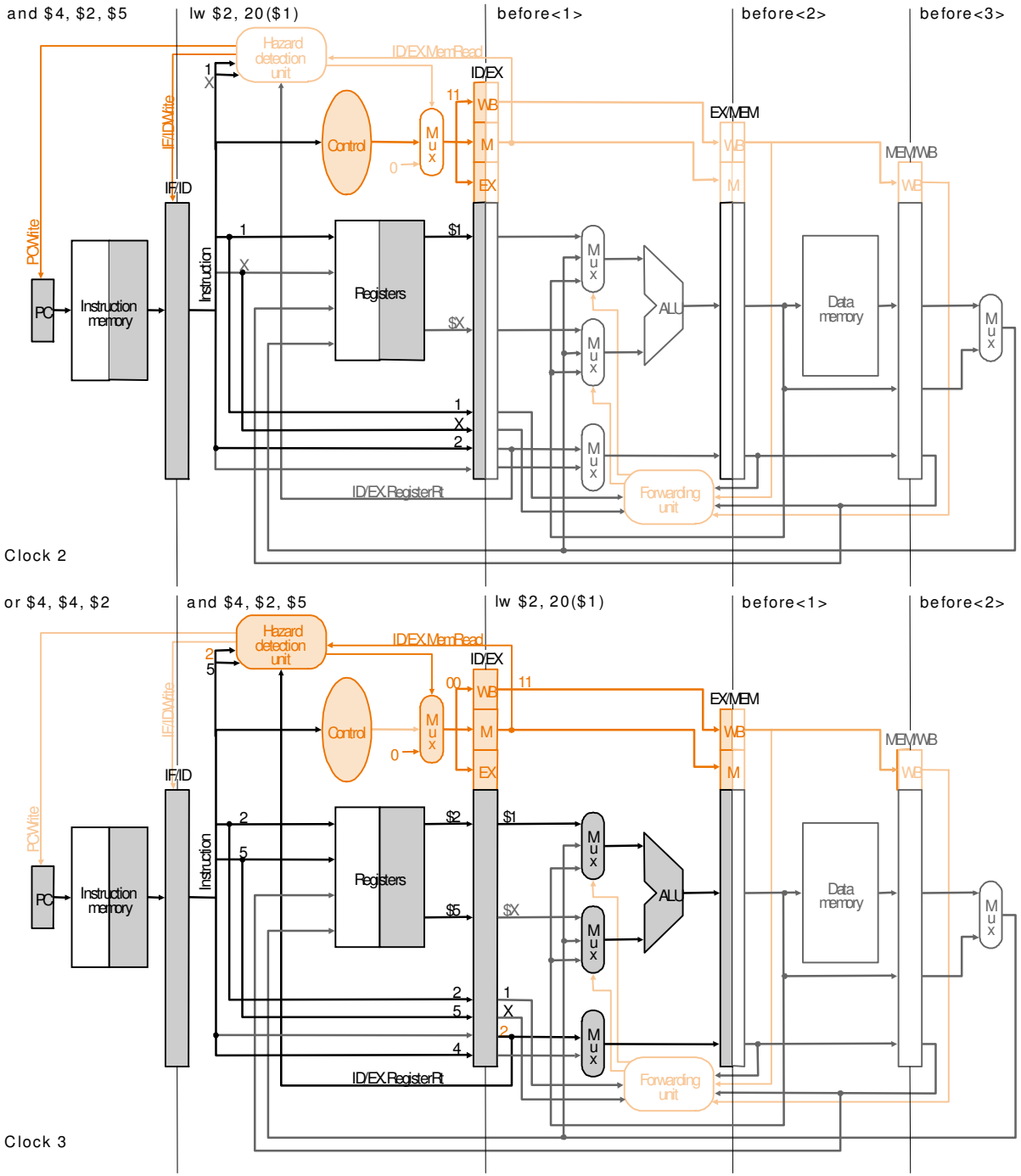


Hazard Detection Unit

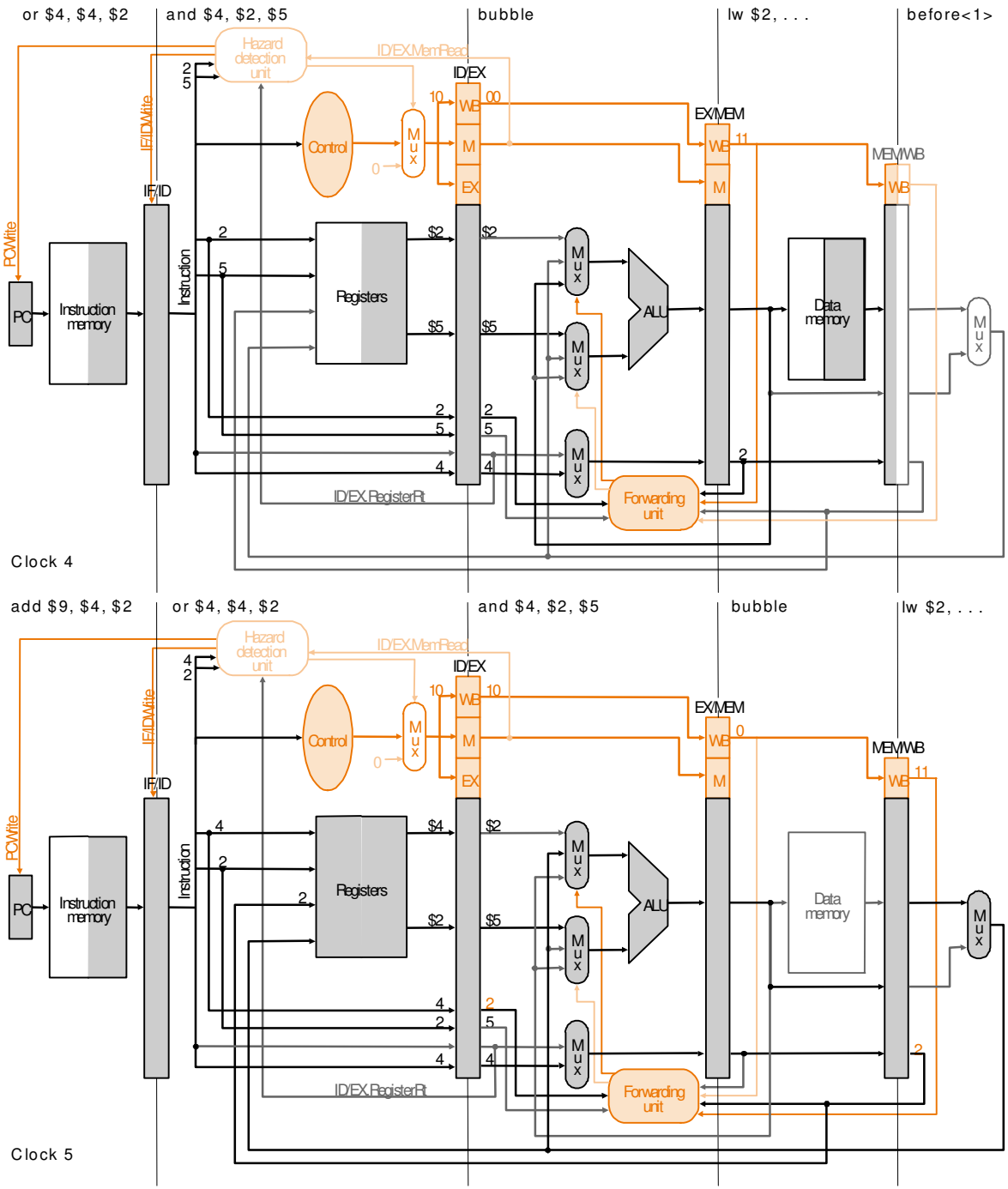
- **Stall by letting an instruction that won't write anything go forward**



Exemplo pag 493

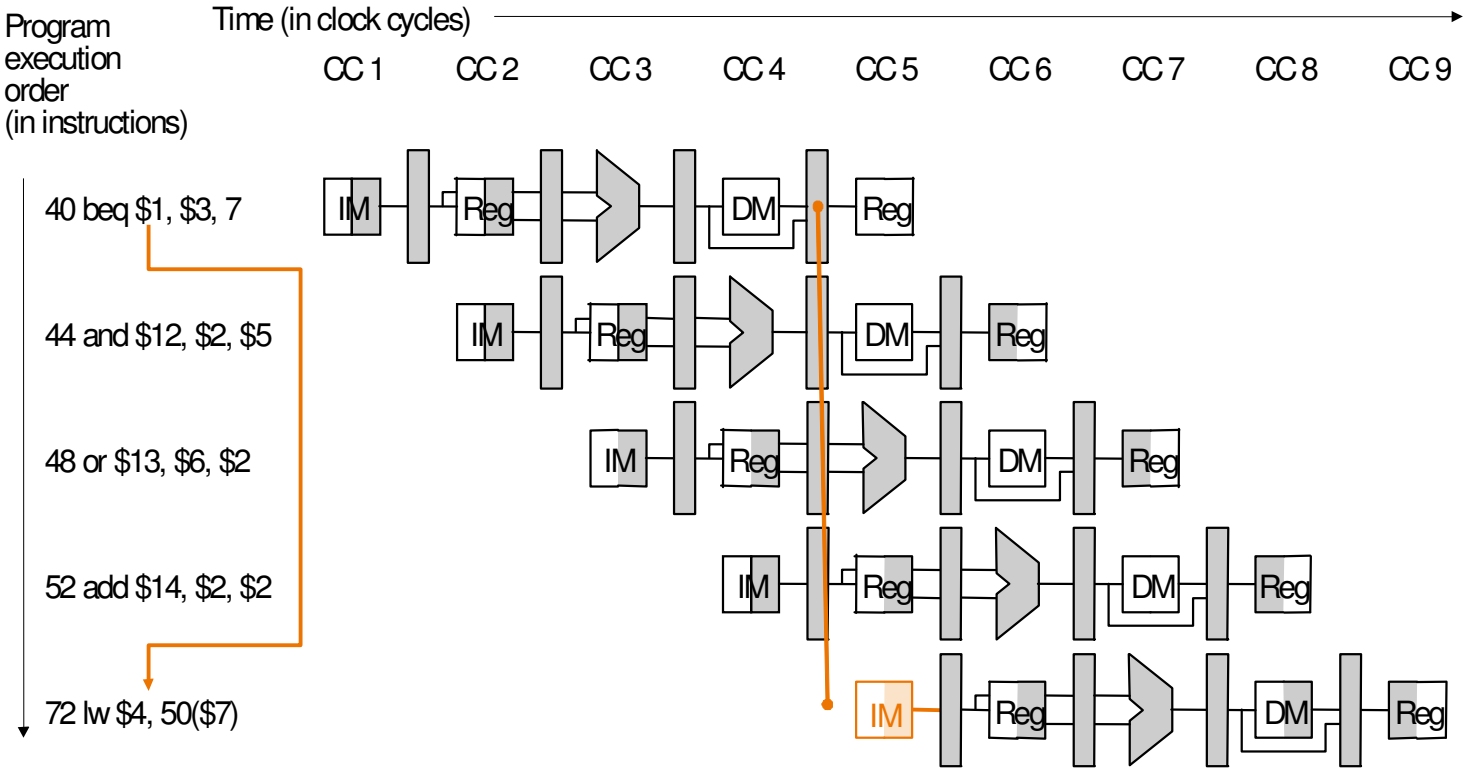


Exemplo pag 493



Branch Hazards

- **When we decide to branch, other instructions are in the pipeline!**

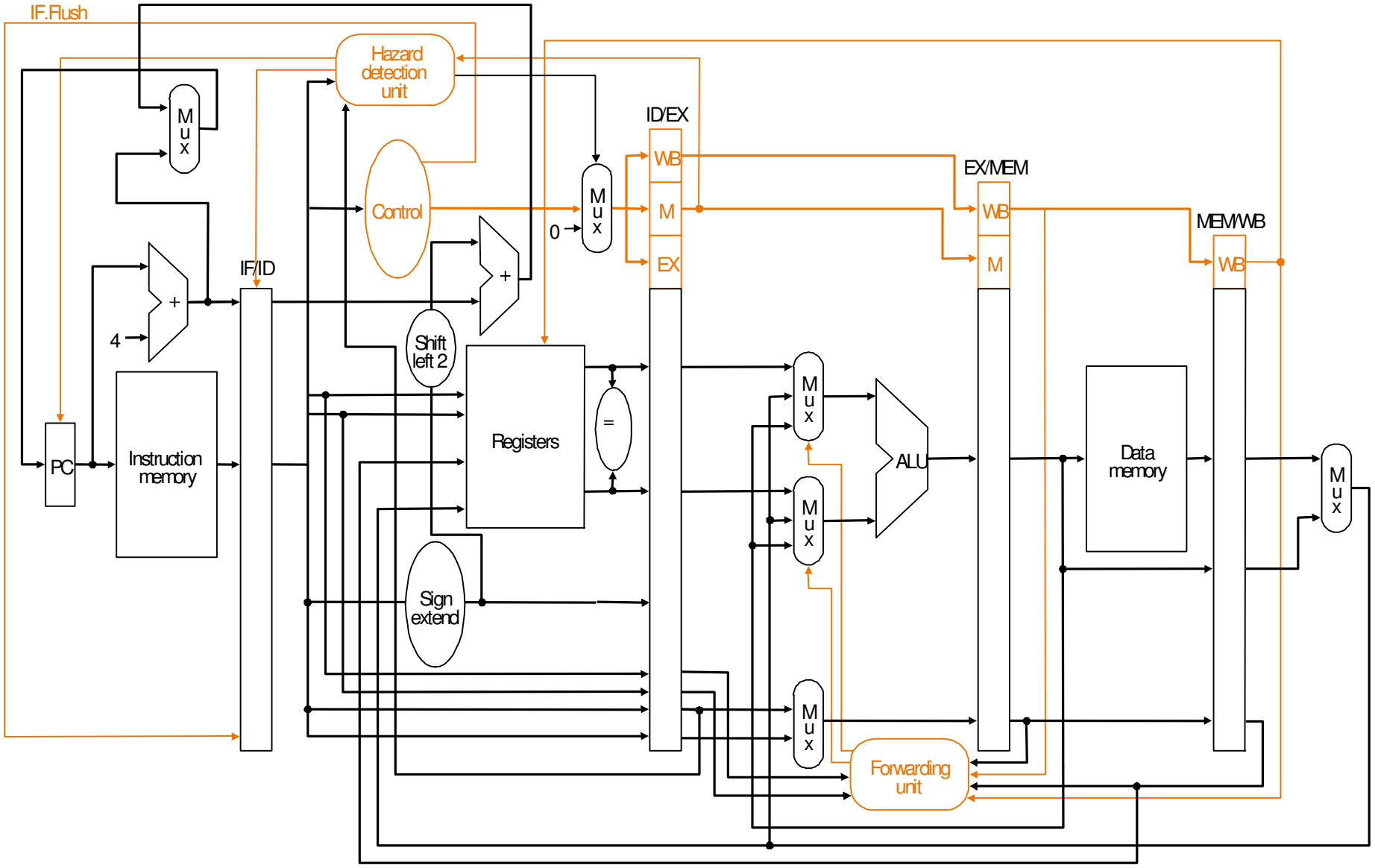


- **We are predicting "branch not taken" (stalling is too slow)**
  - **need to add hardware for flushing instructions if we are wrong**

Diminuindo a penalidade do “branch taken”

- **no esquema anterior**
  - **decisão só é tomada no estágio MEM**
  - **caso “branch taken” é necessário limpar (flush) os estágios IF, ID e EX**
  - **3 clocks perdidos**
- **para diminuir a penalidade:**
  - **antecipar a decisão do estágio MEM para o estágio ID**
  - **economia de dois clocks**
  - **flush somente instrução sendo lida da memória (fetched)**
- **mudanças no hardware:**
  - **cálculo do endereço do desvio (PC + offset<<2)**
  - **comparação dos registradores**
    - 32 XORs com or na saída
    - mais rápido do que a ALU

Flushing Instructions



Improving Performance

- **Try and avoid stalls! E.g., reorder these instructions:**

```
lw $t0, 0($t1)
lw $t2, 4($t1)
sw $t2, 0($t1)
sw $t0, 4($t1)
```

- **Add a "branch delay slot"**
  - **the next instruction after a branch is always executed**
  - **rely on compiler to "fill" the slot with something useful**

- **Branch prediction:**
  - **tentar acertar se o desvio será tomado ou não**
  - **em loops, desvio é tomado a maior parte das vezes**
  - **branch history table**

- **Superscalar: start more than one instruction in the same cycle**

Dynamic Scheduling

- The hardware performs the "scheduling"
  - hardware tries to find instructions to execute
  - out of order execution is possible
  - speculative execution and dynamic branch prediction
- All modern processors are very complicated
  - DEC Alpha 21264: 9 stage pipeline, 6 instruction issue
  - PowerPC and Pentium: branch history table
  - Compiler technology important

- This class has given you the background you need to learn more

- Video: An Overview of Intel's Pentium Processor

  (available from University Video Communications)

Comparação de desempenho (p 504)

- **para monociclo, multiciclo e pipeline**
- **gcc: lw (23%), sw (13%), beq (19%), j (2%), resto (43%)**
- **pipeline:**
  - **2ns (memória e ALU) e 1ns para o registrador (RD ou WR)**
  - **1/2 dos lw seguidas por instruções que usam o resultado**
  - **1/4 dos beq são errados (1 clock perdido)**
  - **jumps perdem um ciclo**
- **pipeline:**
  - **lw: 1 clock sem hazard e 2 clock com hazard; média = 1.5**
  - **beq: 1 clock se OK (3/4) e 2 clocks se não OK (1/4); média = 1.25**
  - **jump: 2 clocks**
  - **demais instruções: 1 clock**
    - CPI = 1.5*0.23+1*0.13+1*0.43+1.25*0.19+2*0.02 = 1.18
    - clock = 2ns; tempo médio de execução = 2*1.18=2.36ns
    - multiciclo; 4.02*2=8.08ns; monociclo = 8ns
    - pipeline é 3.4 vezes mais rápido do que monociclo ou multiciclo

Circuito completo