1/2 07. *

A. i_1: LOAD $R_1, 9(R_5)$
i_2: ADD R_6, R_1, R_3.

↳ procesor cu 5 nivele, instr. dependente RAW

→ dependenta de R_1

a) J_1: IF | ID | ALU | MEM | WB |
J_2: | IF | ID | – | – | ALU | MEM | WB |

↑ avem nev. la finele MEM pt. instr. e LOAD.
DS=2

b) J_1: IF | ID | ALU | MEM | WB |
J_2: | IF | ID | – | ALU | MEM | WB |

B: J_1: ADD R_1, R_6, R_4
J_2: LOAD $R_5, 9(R_1)$

a) J_1: IF | ID | ALU | MEM | WB |
J_2: | IF | ID | – | – | ALU | MEM | WB |

b) J_1: IF | ID | ALU | MEM | WB |
J_2: | IF | ID | ALU | MEM | WB |

DS=2

DS=0

Dacă y_2: LD Rg, (R1)

↓

Dacă avem adresare indirectă, val. reg. în AGU nu mă interesează, doar în MEM

a) → DS = 1

var. citită din mem. de la adresa de adr

2.

$K = X[i-4] + 12$

$L = Y[j+5]$ XOR K

M = K AND L

Ri, Rj, RK, RL, RM

Pb 85

Temă

i_1: SUB Ri, Ri, #4

y_2: LOAD RK, (Ri)

y_3: ADD RK, RK, #12

y_4: ADD Rj, Rj, #5

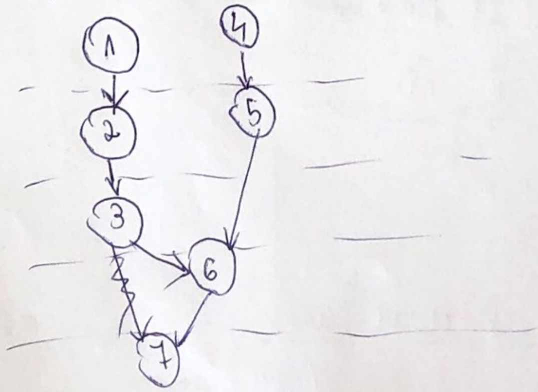
y_5: LOAD RL, (Rj)

y_6: XOR RL, RL, RK

y_7: ~~XOR~~ AND RM, RK, RL

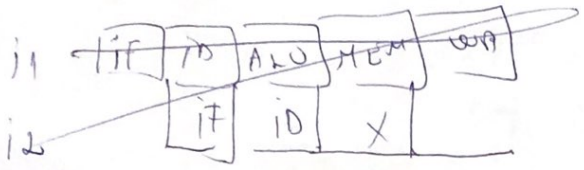
Graf dependențe RAW

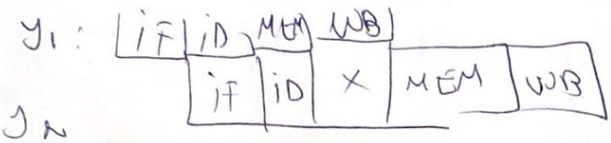
a)



b) 0 instr. = un ciclu

1 – NOP – 2 – NOP – 3 – 4 – NOP – 5 – NOP – 6 – NOP – 7





DS = 1

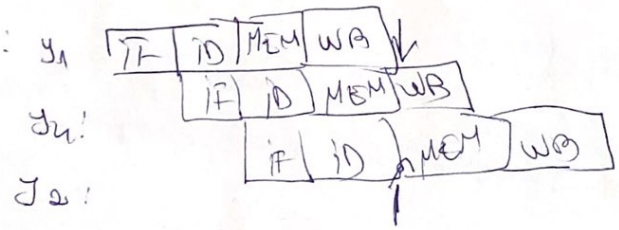
12 cicli

c) Reorganizat

1, 4, 2, 5, 3, 6, 7 → ordine execuție secv. optimizat.

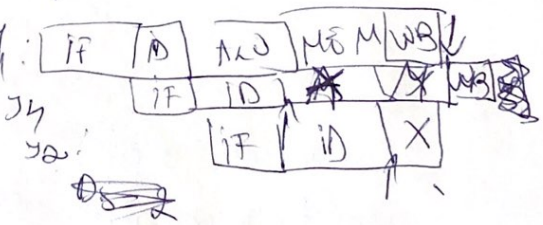
d) fără forwarding: 1 – 4 – 2 – 5 – 3 – NOP – 6 – NOP – 7

9 cicli



2 depinzând de 1

Dacă procesul are suport I1 : IF ID ALU MEM WB



~~DS = 2~~

- cu forwarding
  → DS = 0 ⇒ 7 cicli

Infinitate de regiștri generali → putem înlătura multe dependențe

8. I_1: ADD R1, R2, #15

I_2: ADD R3, R4, #17

I_3: ADD R5, R3, R1

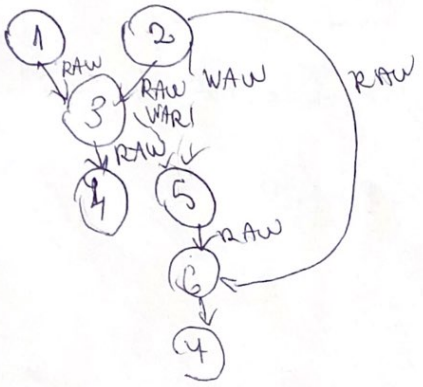
I_4: ADD R6, R5, #12

I_5: ADD R3, R4, #3

I_6: ADD R8, R3, #2

I_7: ADD R9, R8, #14

→ toate dependențele



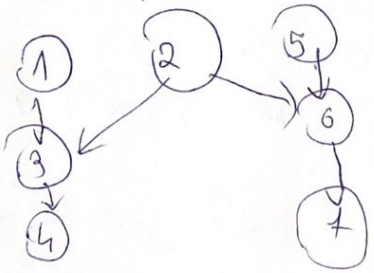
a) DS = 1

1 - 2 - NOP - 3 - NOP - 4 - 5 - 6 - NOP - 7

R: 11 cicluri

Dacă am avea infinite de reg. / pot redenumi registrii și nu mai am dependențe WAW, WAR

⇒



DS = 1

→ 1, 2, 5, 3, 6, 4, 7.