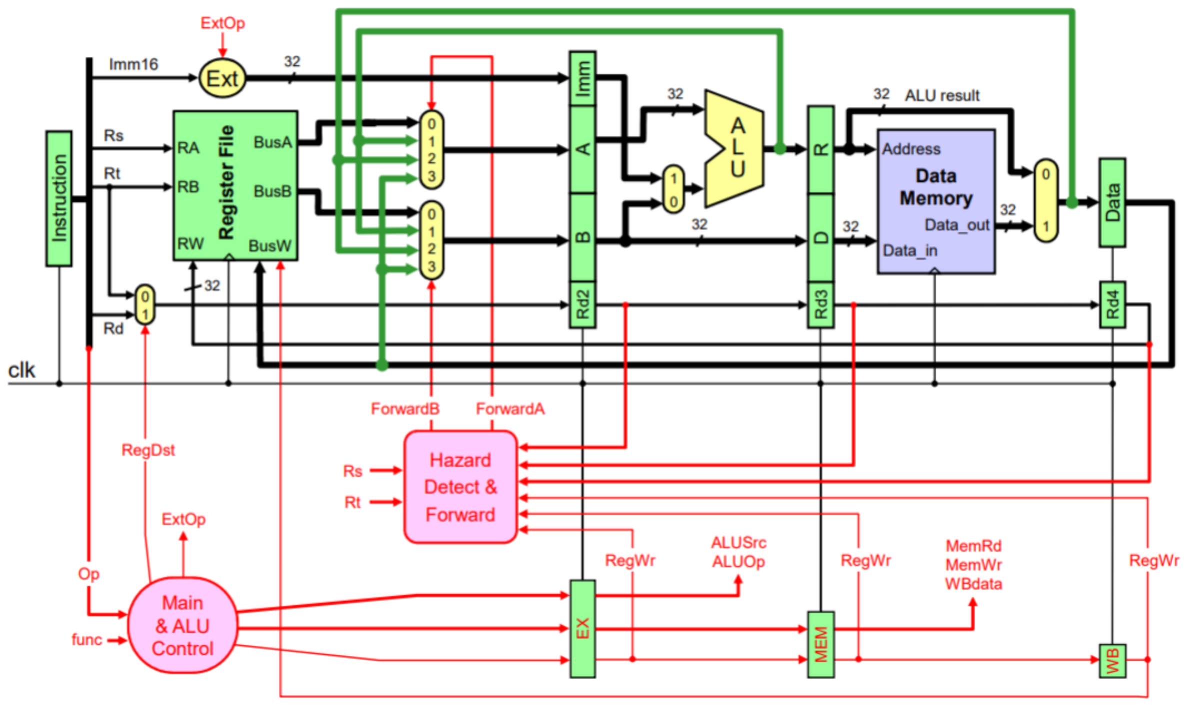
Hazard Detecting and Forwarding Logic



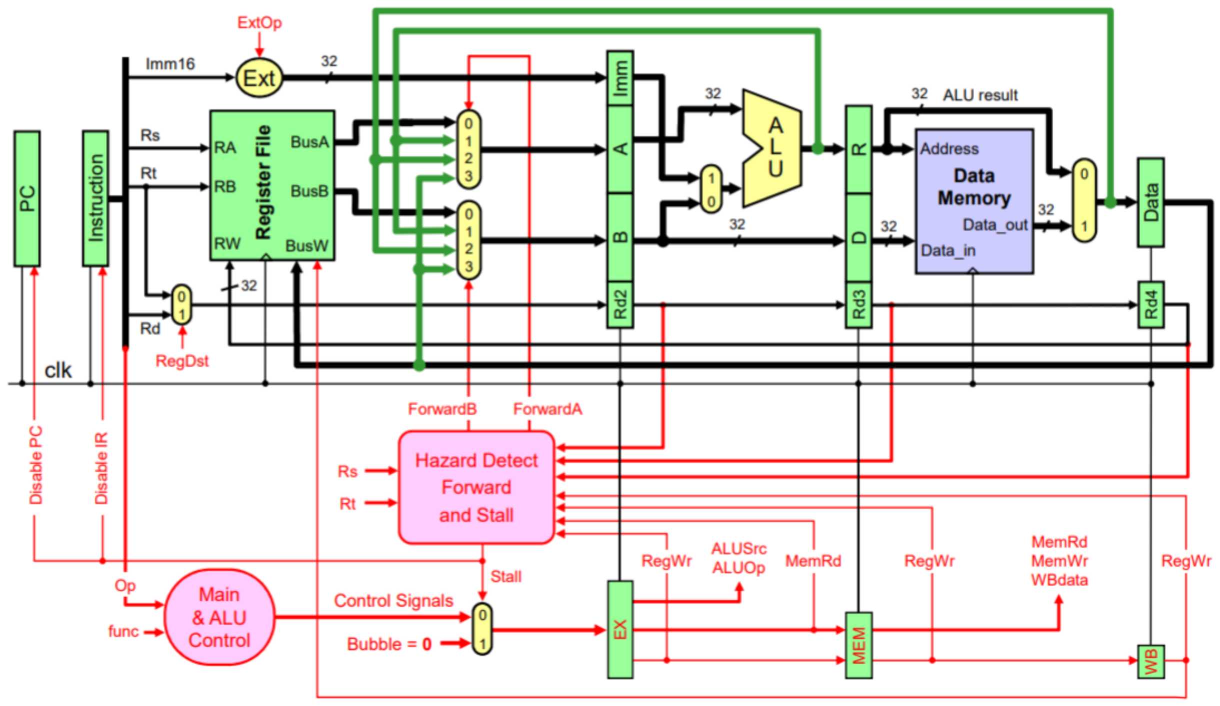
RAW Hazard Detection

- Current instruction is being decoded in the Decode stage
- Previous instruction is in the Execute stage
- Second previous instruction is in the Memory stage
- Third previous instruction is in the Write Back stage

```
If      ((Rs != 0) and (Rs == Rd2) and (EX.RegWr))  ForwardA = 1
Else if ((Rs != 0) and (Rs == Rd3) and (MEM.RegWr)) ForwardA = 2
Else if ((Rs != 0) and (Rs == Rd4) and (WB.RegWr))  ForwardA = 3
Else    ForwardA = 0

If      ((Rt != 0) and (Rt == Rd2) and (EX.RegWr))  ForwardB = 1
Else if ((Rt != 0) and (Rt == Rd3) and (MEM.RegWr)) ForwardB = 2
Else if ((Rt != 0) and (Rt == Rd4) and (WB.RegWr))  ForwardB = 3
Else    ForwardB = 0
```

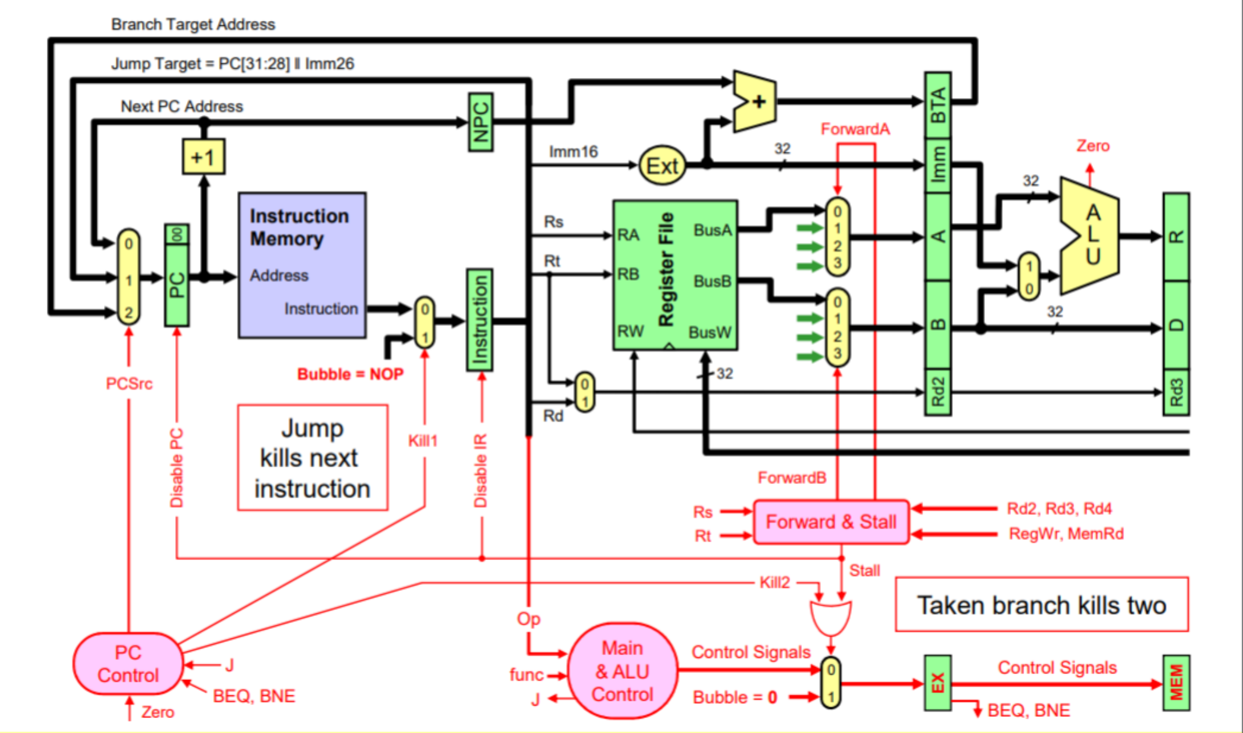
Hazard Detecting and Forwarding Logic



❖ Condition for stalling the pipeline

```
if((EX.MemRd == 1)  // Detect Load in EX stage
and (ForwardA==1 or ForwardB==1)) Stall  // RAW Hazard
```

Pipelined Jump and Branch



Pipelined Processor Design *COE 301 / ICS 233 – Computer Organization* *© Muhamed Mudawar – slide 40*

PC Control for Pipelined Jump and Branch

```
if ((BEQ && Zero) || (BNE && !Zero))
  { Jmp=0; Br=1; Kill1=1; Kill2=1; }
else if (J)
  { Jmp=1; Br=0; Kill1=1; Kill2=0; }
else
  { Jmp=0; Br=0; Kill1=0; Kill2=0; }
```

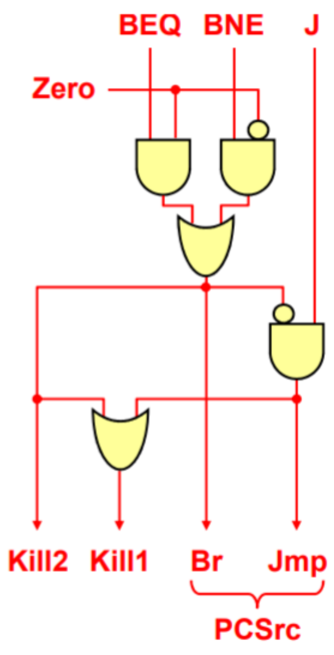
Br $= ((\text{BEQ} \cdot \text{Zero}) + (\text{BNE} \cdot \overline{\text{Zero}}))$

Jmp $= \text{J} \cdot \overline{\text{Br}}$

Kill1 $= \text{J} + \text{Br}$

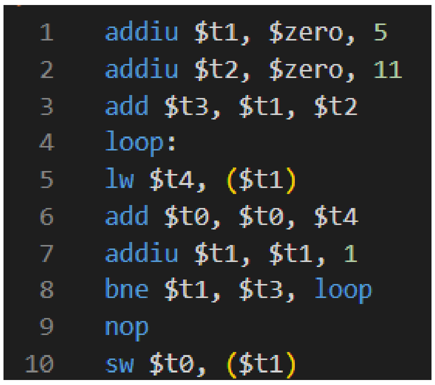
Kill2 $= \text{Br}$

PCSrc $= \{\text{Br}, \text{Jmp}\}$ // 0, 1, or 2



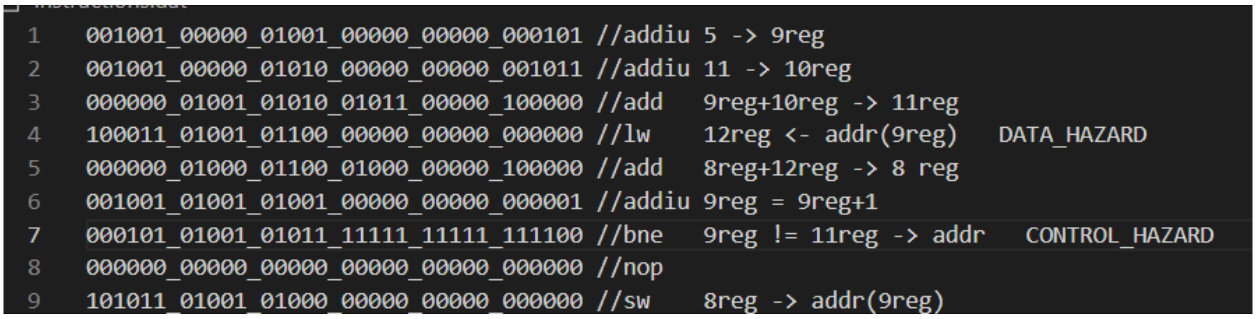
Pipelined Processor Design *COE 301 / ICS 233 – Computer Organization* *© Muhamed Mudawar – slide 41*

Переписав ядро за схемами, які були на слайдах. Виправив обробку файлу інструкцій, тепер можна зразу копіювати інструкції в двійковій системі з марсу та завантажувати їх в ядро. Додав ще одну команду (bne), для реалізації конвеєру додав регістр, для реалізації обробки збоїв додав hazardDetect.

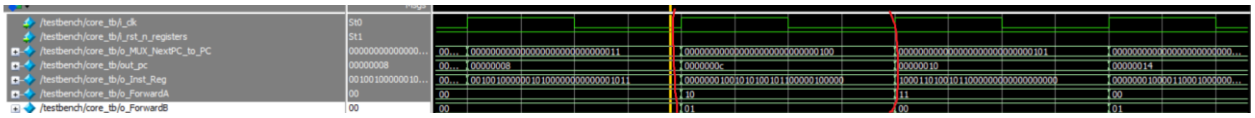


```
addiu $t1, $zero, 5
addiu $t2, $zero, 11
add $t3, $t1, $t2
loop:
lw $t4, ($t1)
add $t0, $t0, $t4
addiu $t1, $t1, 1
bne $t1, $t3, loop
nop
sw $t0, ($t1)
```

Тестовий код

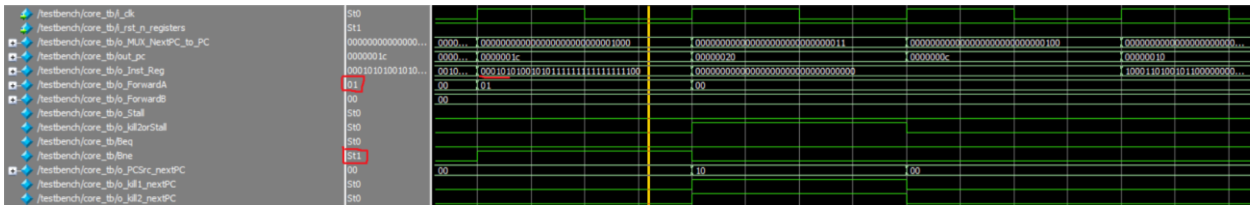


```
001001_00000_01001_00000_00000_000101 //addiu 5 -> 9reg
001001_00000_01010_00000_00000_001011 //addiu 11 -> 10reg
000000_01001_01010_01011_00000_100000 //add   9reg+10reg -> 11reg
100011_01001_01100_00000_00000_000000 //lw    12reg <- addr(9reg)   DATA_HAZARD
000000_01000_01100_01000_00000_100000 //add   8reg+12reg -> 8 reg
001001_01001_01001_00000_00000_000001 //addiu 9reg = 9reg+1
000101_01001_01011_11111_11111_111100 //bne   9reg != 11reg -> addr   CONTROL_HAZARD
000000_00000_00000_00000_00000_000000 //nop
101011_01001_01000_00000_00000_000000 //sw    8reg -> addr(9reg)
```



```
/testbench/core_tb/i_clk                 St0
/testbench/core_tb/i_rst_n_registers     St1
/testbench/core_tb/o_MUX_NextPC_to_PC    00000000000000...
/testbench/core_tb/out_pc                00000008
/testbench/core_tb/o_Inst_Reg            0010010000010...
/testbench/core_tb/o_ForwardA            00
/testbench/core_tb/o_ForwardB            00
```

DATA_HAZARD



```
/testbench/core_tb/i_clk                 St0
/testbench/core_tb/i_rst_n_registers     St1
/testbench/core_tb/o_MUX_NextPC_to_PC    00000000000000...
/testbench/core_tb/out_pc                0000001c
/testbench/core_tb/o_Inst_Reg            0001010100101 0...
/testbench/core_tb/o_ForwardA            01
/testbench/core_tb/o_ForwardB            00
/testbench/core_tb/o_Stall               St0
/testbench/core_tb/o_kill2orStall        St0
/testbench/core_tb/Beq                   St0
/testbench/core_tb/Bne                   St1
/testbench/core_tb/o_PCSrc_nextPC        00
/testbench/core_tb/o_kill1_nextPC        St0
/testbench/core_tb/o_kill2_nextPC        St0
```

CONTROL_HAZARD

Forward заєм