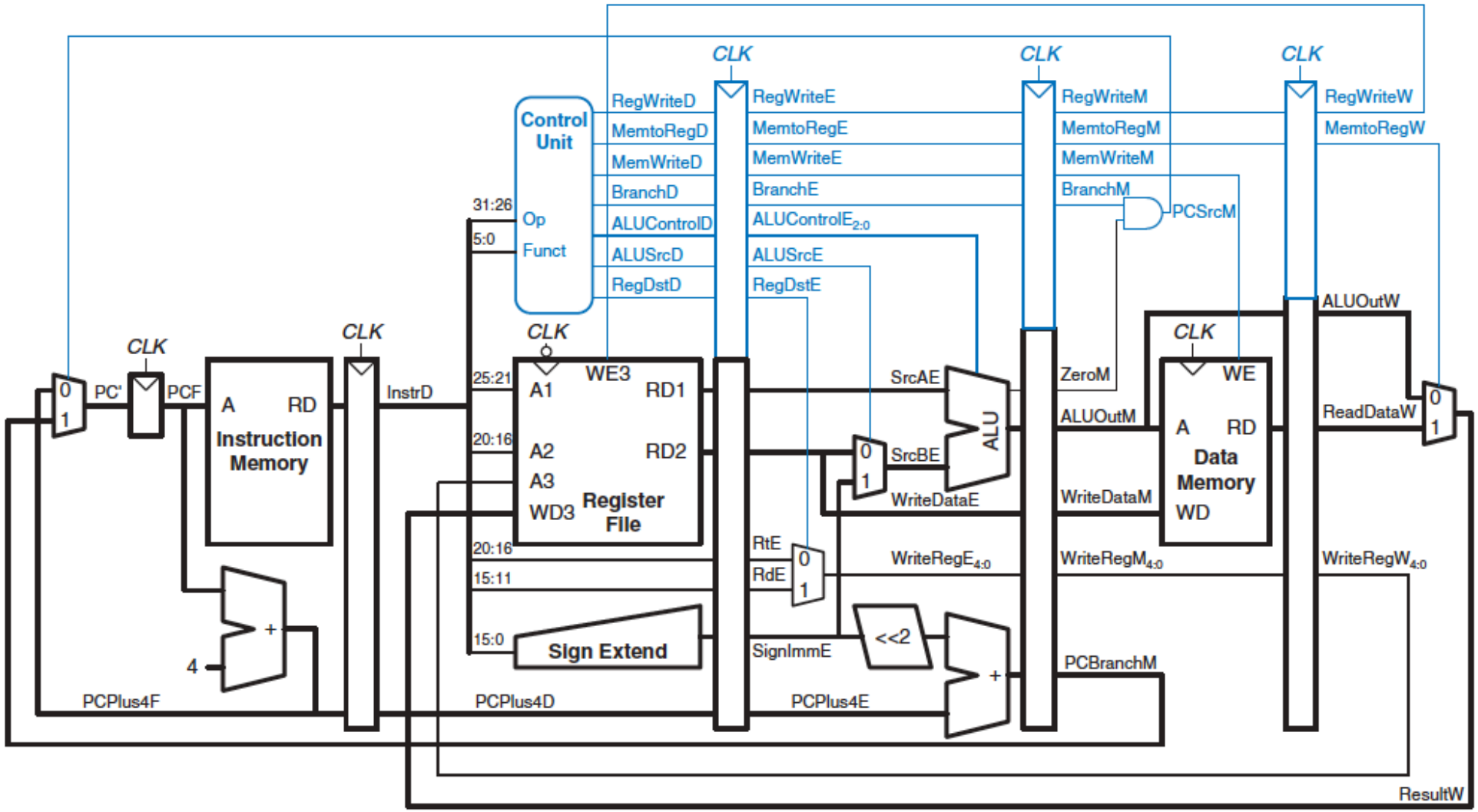
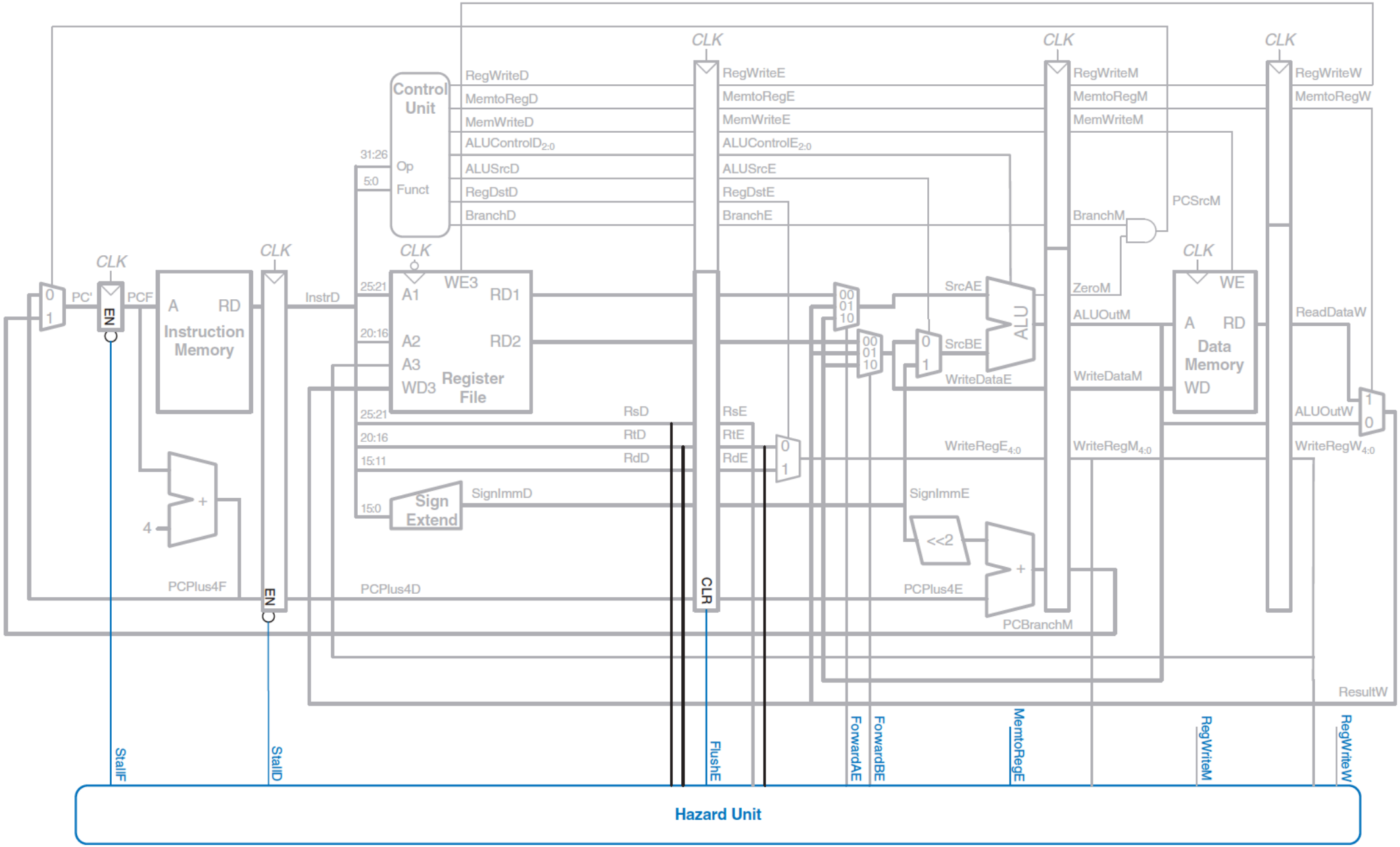
Pipeline Control Hazards


CLK
Control Unit
Op
Funct
31:26
5:0
RegWriteD
MemtoRegD
MemWriteD
BranchD
ALUControlD
ALUSrcD
RegDstD
RegWriteE
MemtoRegE
MemWriteE
BranchE
ALUControlE2:0
ALUSrcE
RegDstE
RegWriteM
MemtoRegM
MemWriteM
BranchM
PCSrcM
RegWriteW
MemtoRegW
ALUOutW
ReadDataW
PC'
PCF
A
RD
Instruction Memory
InstrD
25:21
20:16
15:11
15:0
A1
A2
A3
WD3
WE3
RD1
RD2
Register File
Sign Extend
SrcAE
SrcBE
ALU
ZeroM
ALUOutM
WriteDataE
WriteDataM
RtE
RdE
WriteRegE4:0
WriteRegM4:0
WriteRegW4:0
SignImmE
<<2
PCBranchM
WE
Data Memory
WD
4
PCPlus4F
PCPlus4D
PCPlus4E
ResultW


CLK
Control Unit
Op
Funct
31:26
5:0
RegWriteD
MemtoRegD
MemWriteD
ALUControlD2:0
ALUSrcD
RegDstD
BranchD
RegWriteE
MemtoRegE
MemWriteE
ALUControlE2:0
ALUSrcE
RegDstE
BranchE
RegWriteM
MemtoRegM
MemWriteM
BranchM
PCSrcM
RegWriteW
MemtoRegW
ZeroM
ALUOutM
WriteDataM
ReadDataW
ALUOutW
WriteRegM4:0
WriteRegW4:0
PC'
PCF
EN
A
RD
Instruction Memory
InstrD
25:21
20:16
15:0
A1
A2
A3
WD3
WE3
RD1
RD2
Register File
WE
WD
Data Memory
RsD
RtD
RdD
RsE
RtE
RdE
SignImmD
Sign Extend
SignImmE
<<2
SrcAE
SrcBE
ALU
WriteDataE
WriteRegE4:0
PCPlus4F
PCPlus4D
PCPlus4E
PCBranchM
ResultW
CLR
StallF
StallD
FlushE
ForwardAE
ForwardBE
MemtoRegE
RegWriteM
RegWriteW
Hazard Unit

Hazards can be Data or Control

Data hazard

- Instruction tries to read a register that has not been written back yet by a previous instruction.

Control hazard

- Decision of what instruction to fetch next has not been made by the time the fetch takes place.

Need to enhance our pipelined processor to detect and handle hazards.

Solving Control Hazards

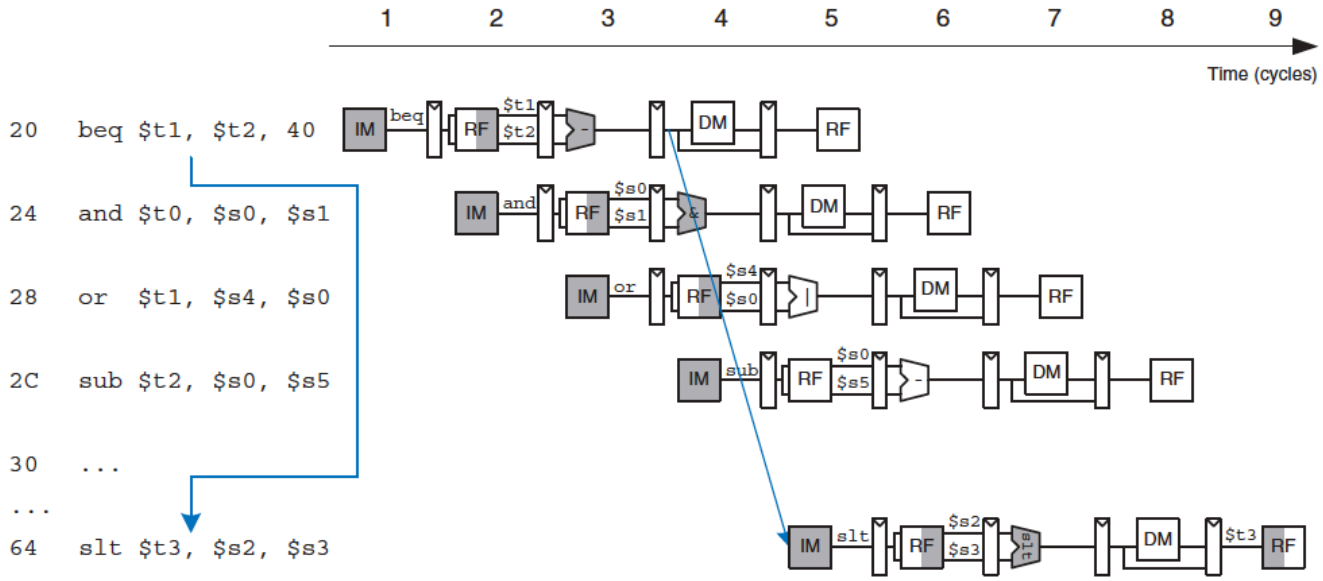
Processor does not know what instruction to fetch next

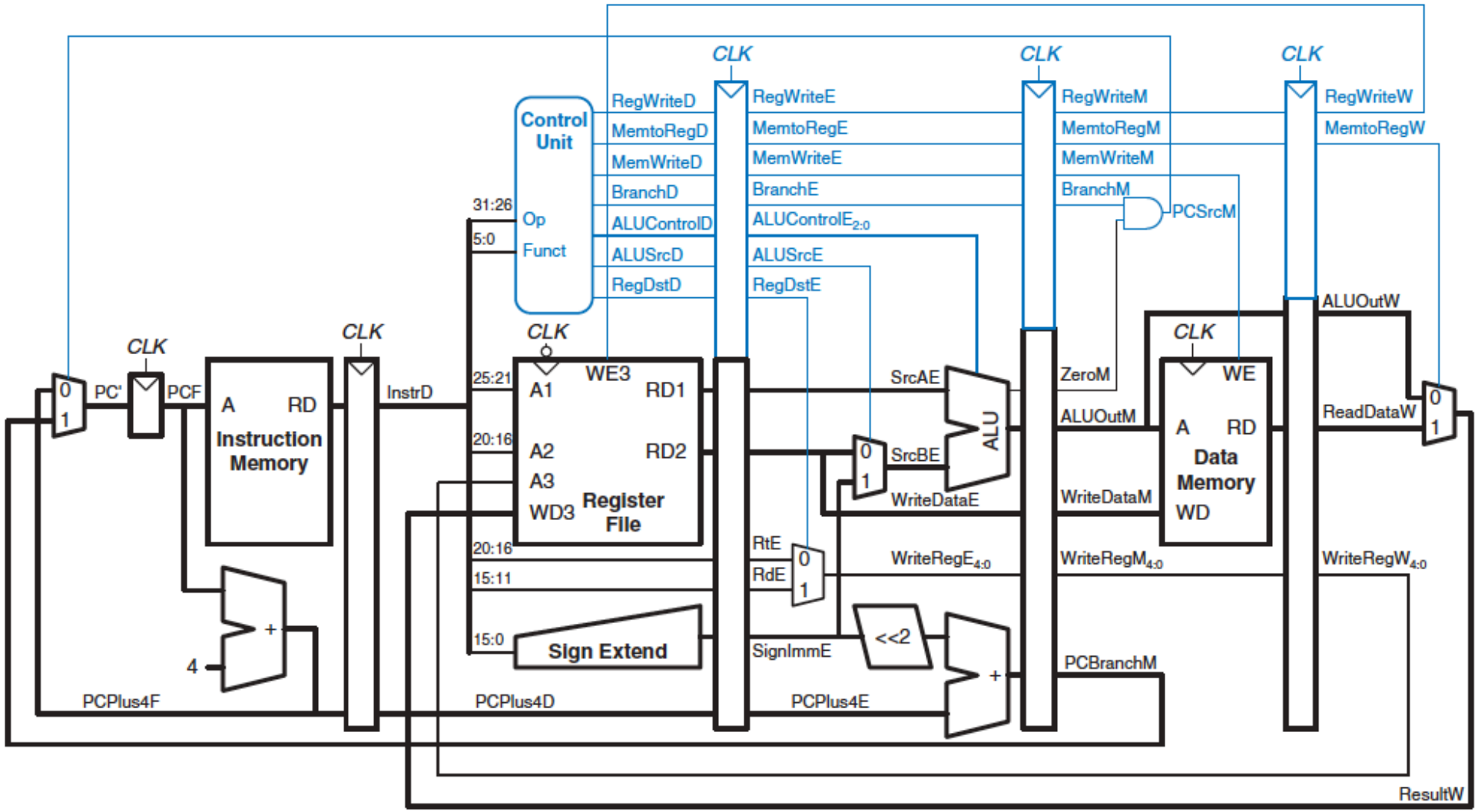
- Branch decision has not been made by the time the next instruction is fetched
- Can stall the pipeline until the branch decision is made

PCSrc computed

- Decision is made in the Memory stage, meaning pipeline would need to stall three cycles at every branch
- Drastically hamper system performance

Can you think of an alternative?




CLK
Control Unit
Op
Funct
31:26
5:0
RegWriteD
MemtoRegD
MemWriteD
BranchD
ALUControlD
ALUSrcD
RegDstD
RegWriteE
MemtoRegE
MemWriteE
BranchE
ALUControlE2:0
ALUSrcE
RegDstE
RegWriteM
MemtoRegM
MemWriteM
BranchM
PCSrcM
RegWriteW
MemtoRegW
ALUOutW
ReadDataW
PC'
PCF
A
RD
Instruction Memory
InstrD
25:21
20:16
15:11
15:0
A1
A2
A3
WD3
WE3
RD1
RD2
Register File
Sign Extend
SrcAE
SrcBE
ALU
ZeroM
ALUOutM
WriteDataE
WriteDataM
RtE
RdE
WriteRegE4:0
WriteRegM4:0
WriteRegW4:0
SignImmE
<<2
PCBranchM
WE
Data Memory
WD
4
PCPlus4F
PCPlus4D
PCPlus4E
ResultW

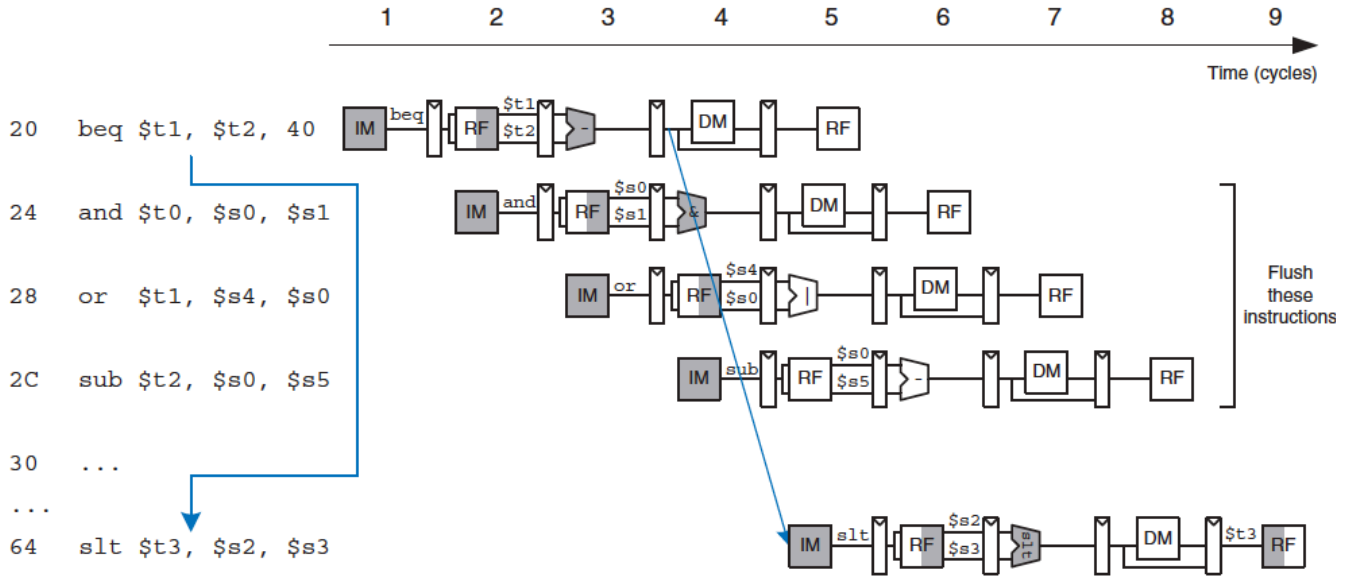
Branch Prediction

Predict whether the branch will be taken.

- Act on a prediction and fill the pipeline accordingly.
- If wrong, throw out the instructions that were loaded.

Flushing

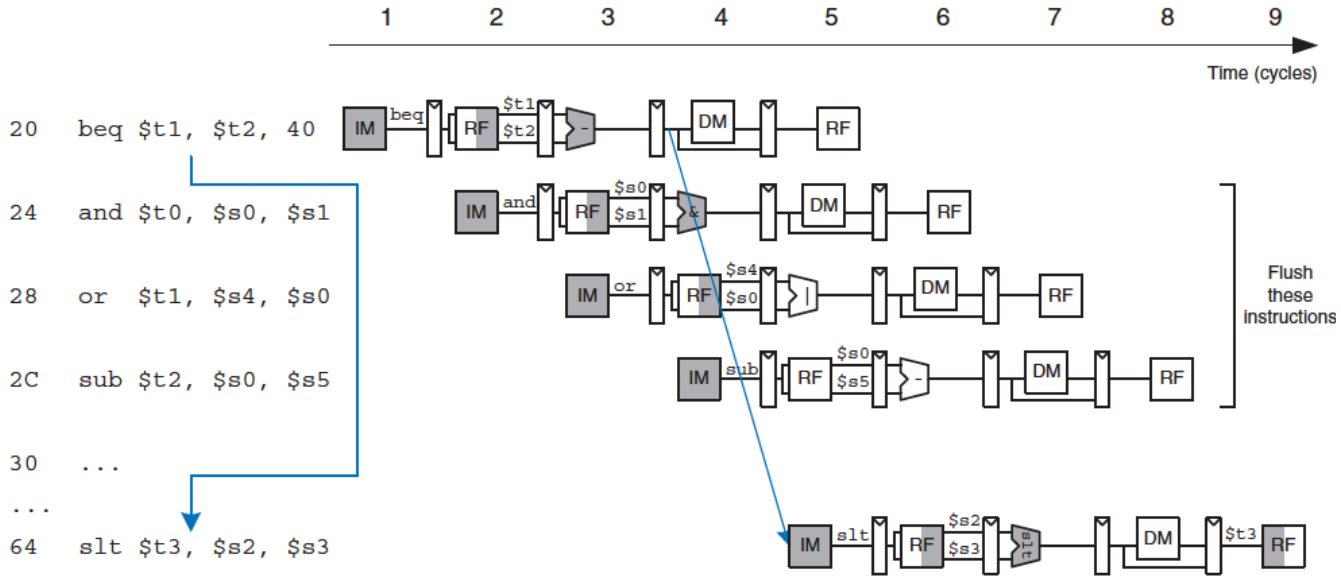
- Wasted instruction cycles are called the branch misprediction penalty.



Branch Prediction

Branch from address 20 to 64 is taken

- Decision not made until cycle 4, and, or and sub have already been fetched
- They must be flushed
- Slt fetched from line 64 on cycle 5

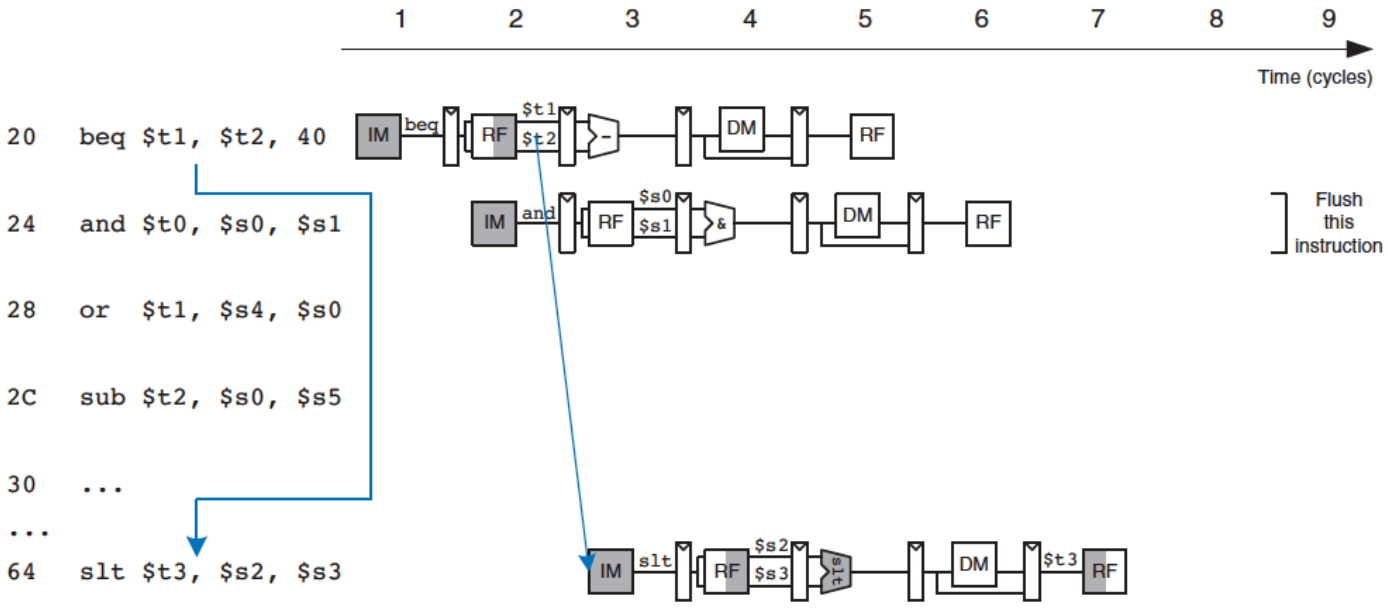


Branch Prediction

Could make our predictions earlier

- Compare the value of two registers with dedicated equality comparator
- Early branch detection

Branch misprediction reduced to one instruction rather than three.



CPU Modifications

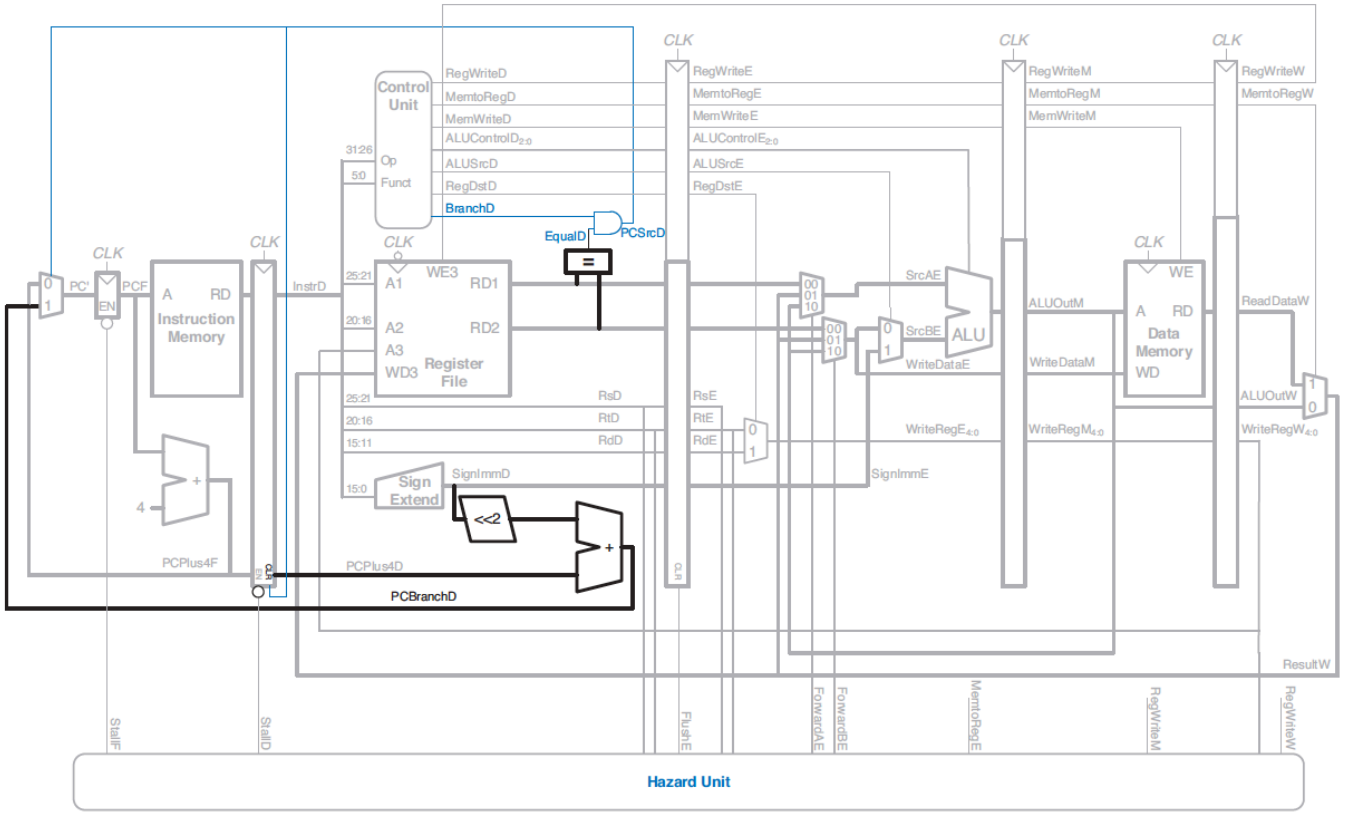
CPU modification with branch decision moved up and control hazards handled.

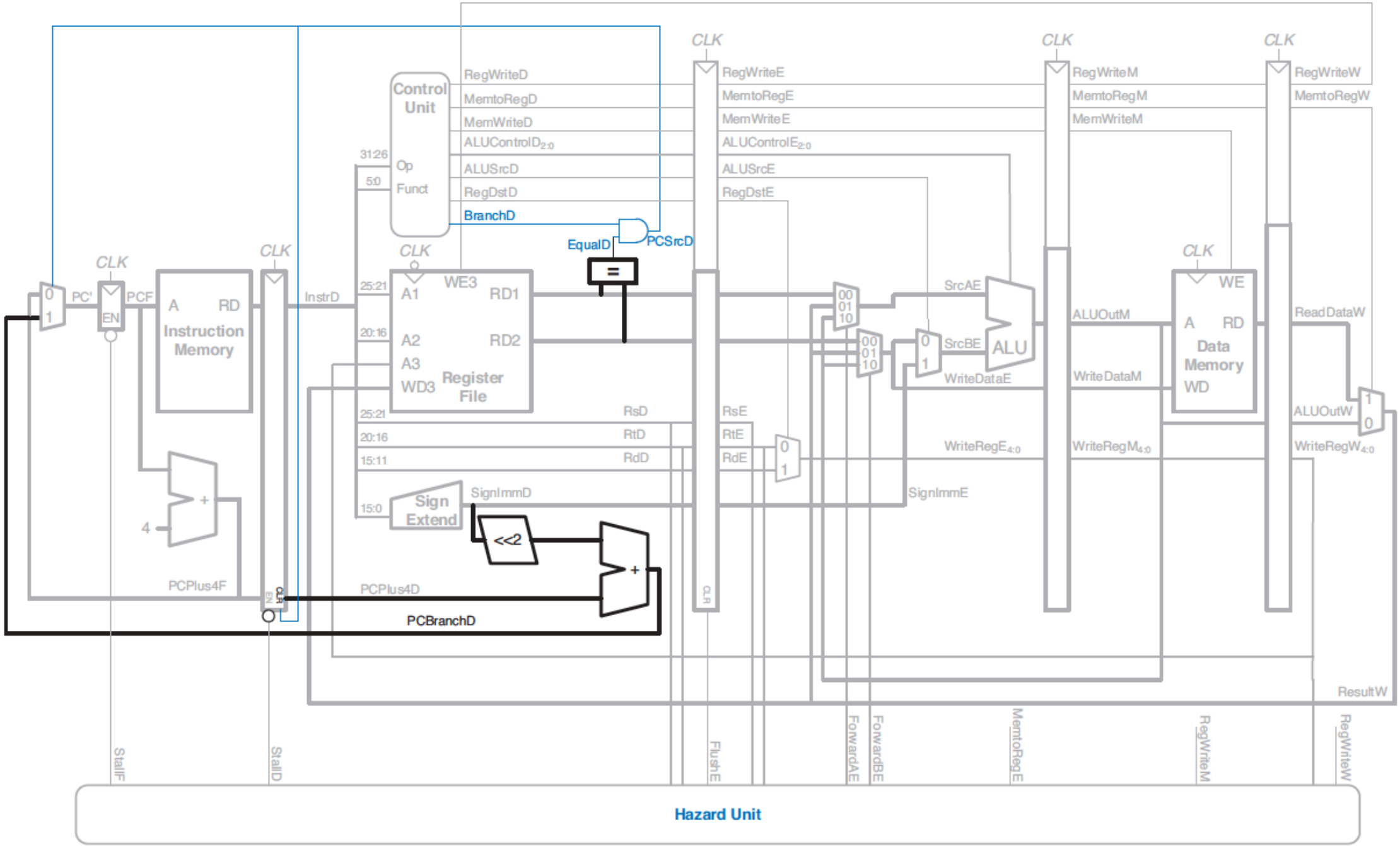
Equality comparator added to the decode stage and PCSrc AND gate is moved earlier

Determine PCSrc in the Decode rather than Memory stage

PCBranch adder must also be moved into the Decode to calculate the destination address in time.

Clear connected to PCSrcD to flush when a branch is taken.




CLK
Control Unit
Op
Funct
31:26
5:0
RegWriteD
MemtoRegD
MemWriteD
ALUControlD2:0
ALUSrcD
RegDstD
BranchD
EqualD
PCSrcD
=
RegWriteE
MemtoRegE
MemWriteE
ALUControlE2:0
ALUSrcE
RegDstE
RegWriteM
MemtoRegM
MemWriteM
RegWriteW
MemtoRegW
PC'
PCF
EN
A
RD
Instruction Memory
InstrD
25:21
20:16
15:11
15:0
A1
A2
A3
WD3
WE3
RD1
RD2
Register File
RsD
RtD
RdD
RsE
RtE
RdE
Sign Extend
SignImmD
SignImmE
<<2
4
+
PCPlus4F
PCPlus4D
PCBranchD
SrcAE
SrcBE
ALU
WriteDataE
ALUOutM
WriteDataM
WE
WD
Data Memory
ReadDataW
ALUOutW
WriteRegE4:0
WriteRegM4:0
WriteRegW4:0
ResultW
CLR
StallF
StallD
FlushE
ForwardAE
ForwardBE
MemtoRegE
RegWriteM
RegWriteW
Hazard Unit

How well can branch prediction work?

- Seems like a 50/50 chance of guessing correctly

- In reality, can do much better than this. Why? (Think about different cases where we branch)

How well can branch prediction work?

```
loop:
        slti $t1, $t0, 500
        beq $t1, $0, done
        … # do work
        addi $t0, $t0, 1
        j loop
done:
        …
```

Summary

1. RAW data hazards occur when an instruction depends on the result of another instruction that has not yet been written to the register file.
2. Data hazards resolved by forwarding if result is computed soon enough.

   Otherwise stall pipeline until the result is available
3. Control hazards occur when the decision of what instruction to fetch has not been made by the time the next instruction must be fetched.

   Solved by predicting what instruction should be fetched

   Flush if prediction turns out wrong.

   Moving the decision earlier reduces how many instructions must be flushed.