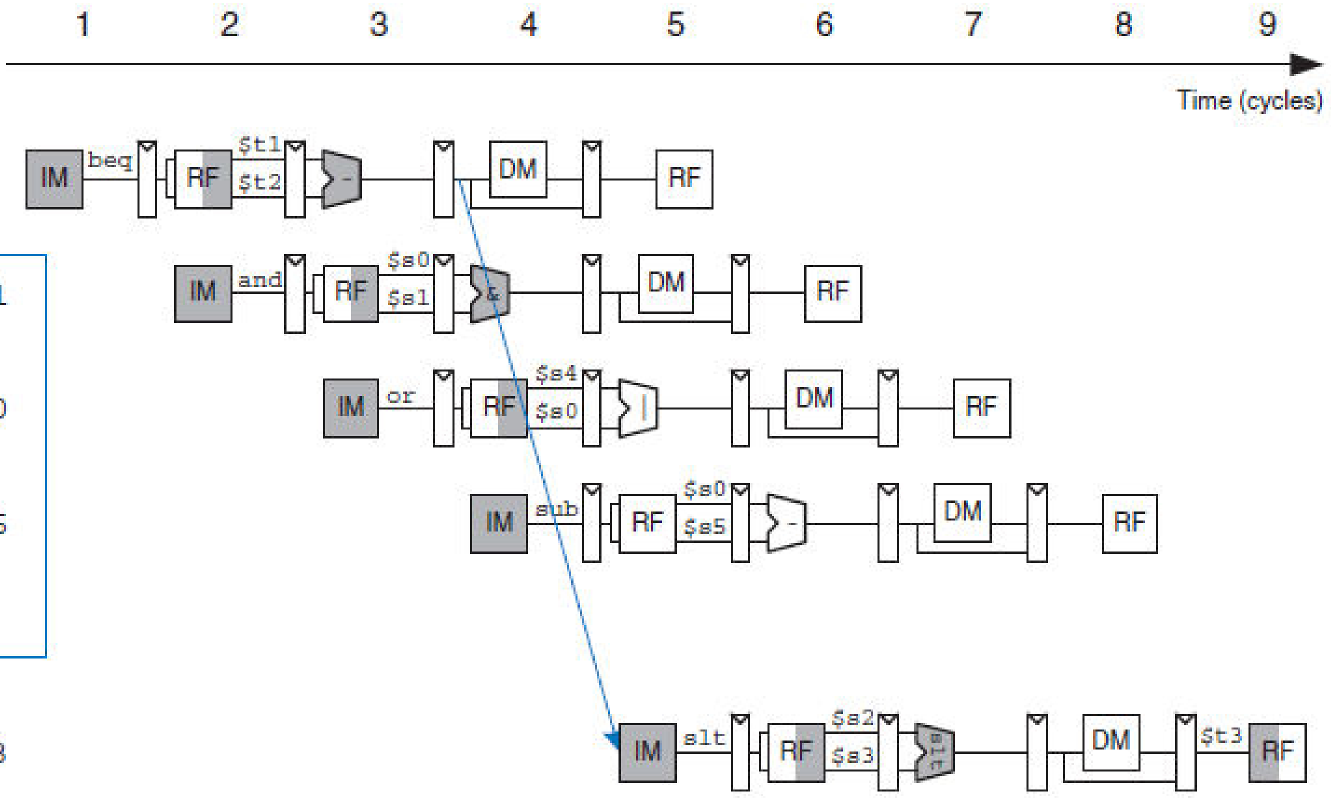
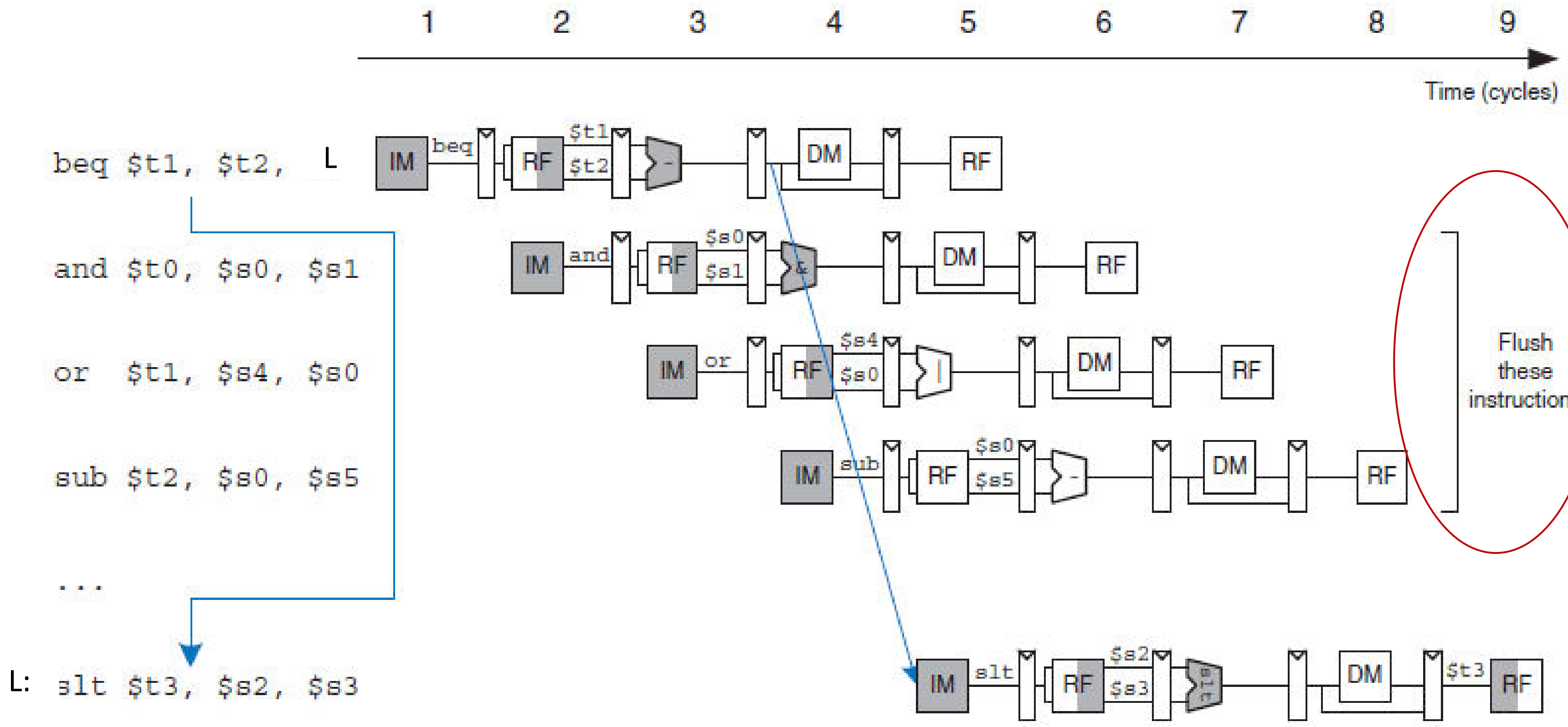
What is this?

```
   beq $t1, $t2,   L

   and $t0, $s0, $s1

   or  $t1, $s4, $s0

   sub $t2, $s0, $s5

   ...

L: slt $t3, $s2, $s3
```



An example of Control Hazards



Control Hazards

- Branch target address is computed only at the end of IE stage
- Cause higher performance penalty as compared to data hazards
- How does one reduce such penalty?

Control Hazards

- What if branch target address is computed at the end of ID stage

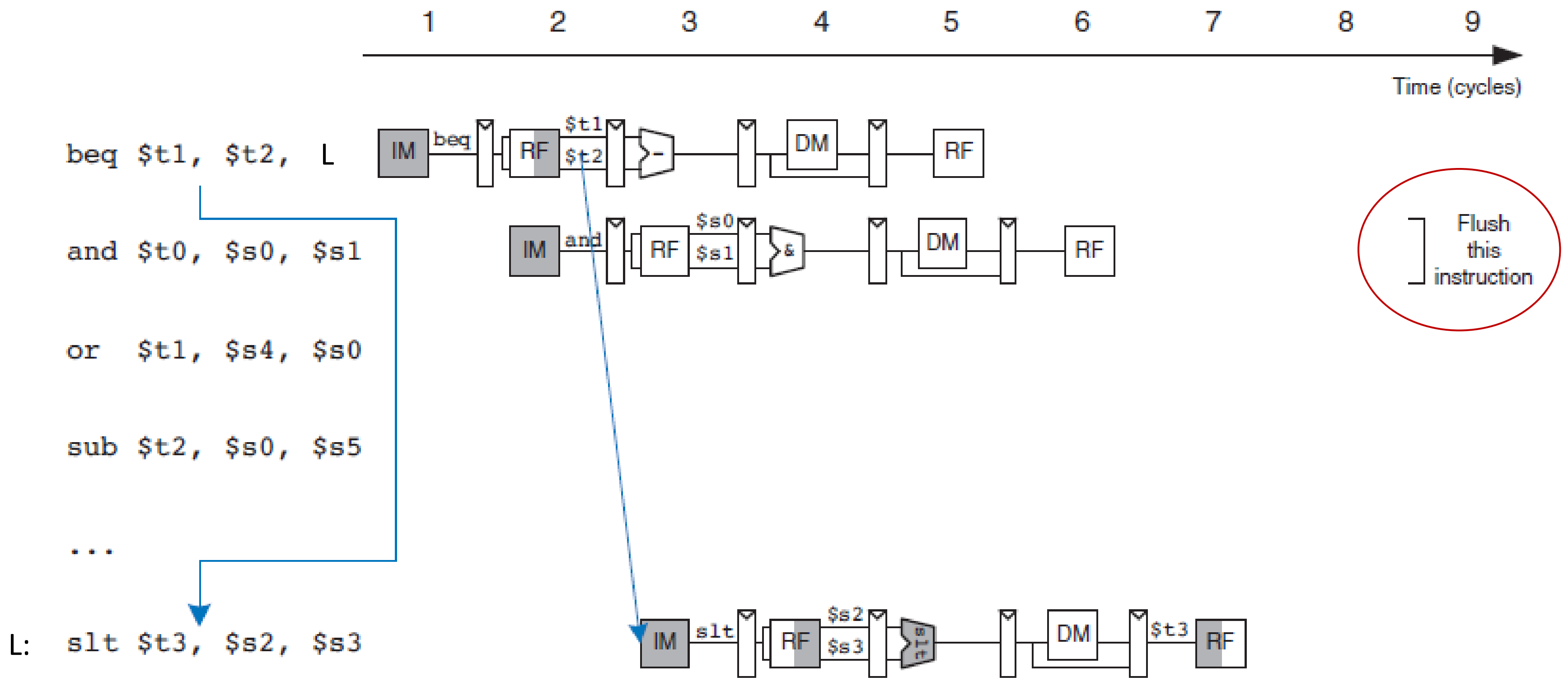
Branch instr.	IF	ID	EXE	MEM	WB		
Branch succ.		**IF**	IF	ID	EXE	MEM	WB
Branch succ + 1				IF	ID	EXE	MEM
Branch succ + 2					IF	ID	EX

$$\text{Speedup} = \frac{CPI\ unpipelined}{1+pipeline\ stall\ cycle\ from\ branchs}$$

$$= \frac{Pipeline\ depth}{1+Branch\ frequency \times Branch\ penalty}$$

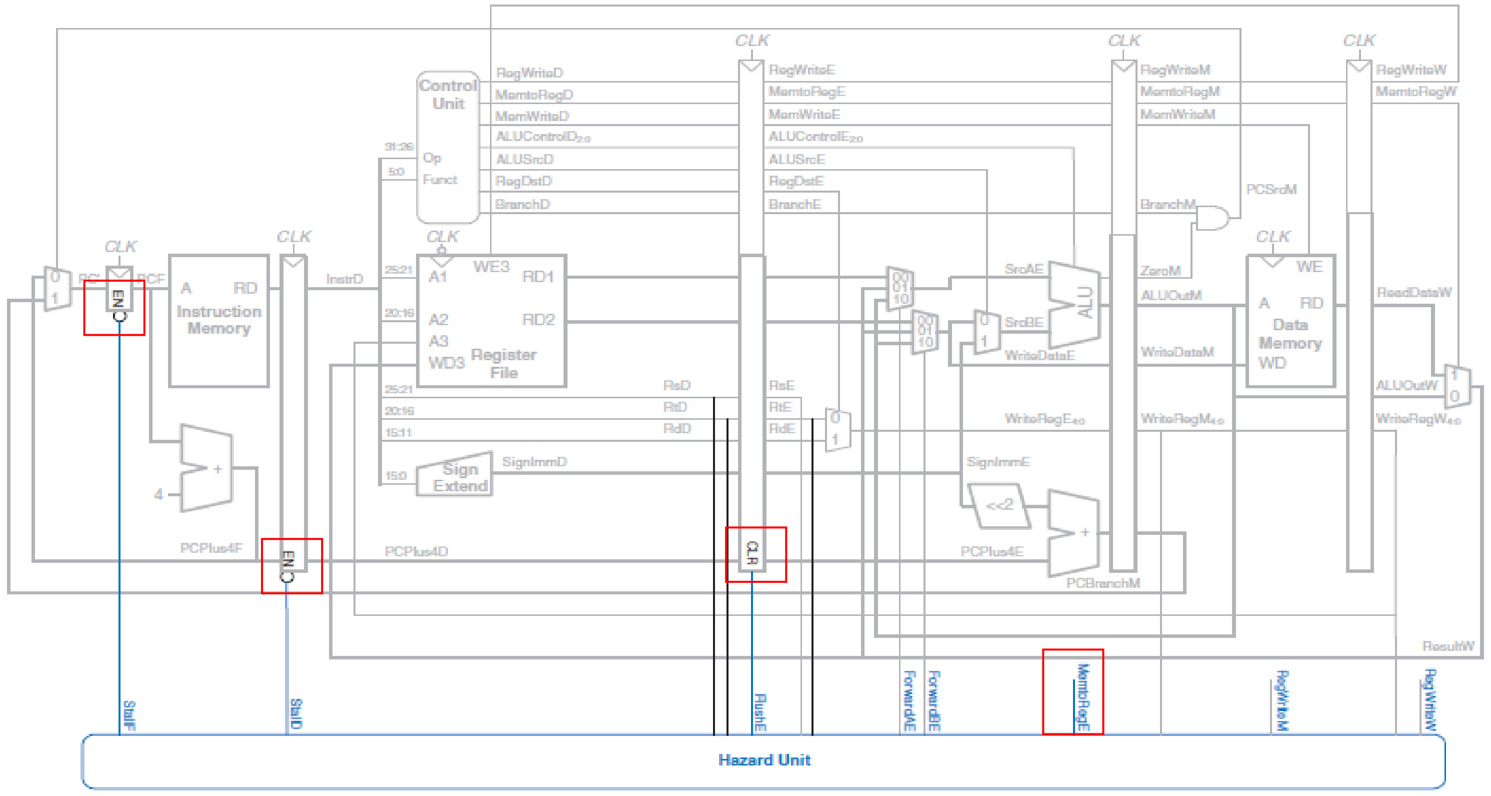
Control Hazards

- Branch target address is computed at the end of ID stage

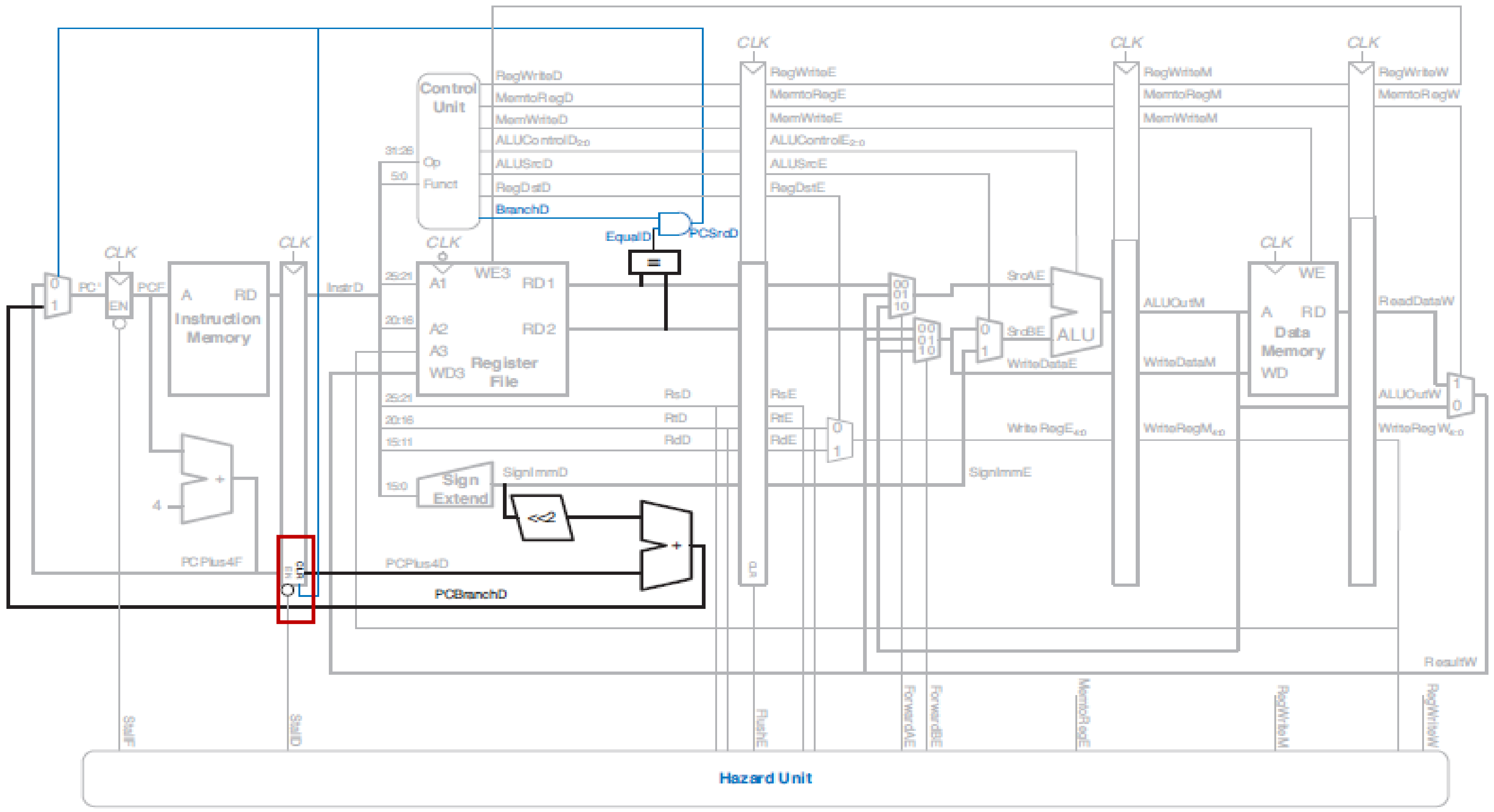


Control Hazards

- Branch target address is computed at the end of ID stage
- What modifications are needed in the pipelined datapath & controlpath?



Control Hazards: Modified Pipeline

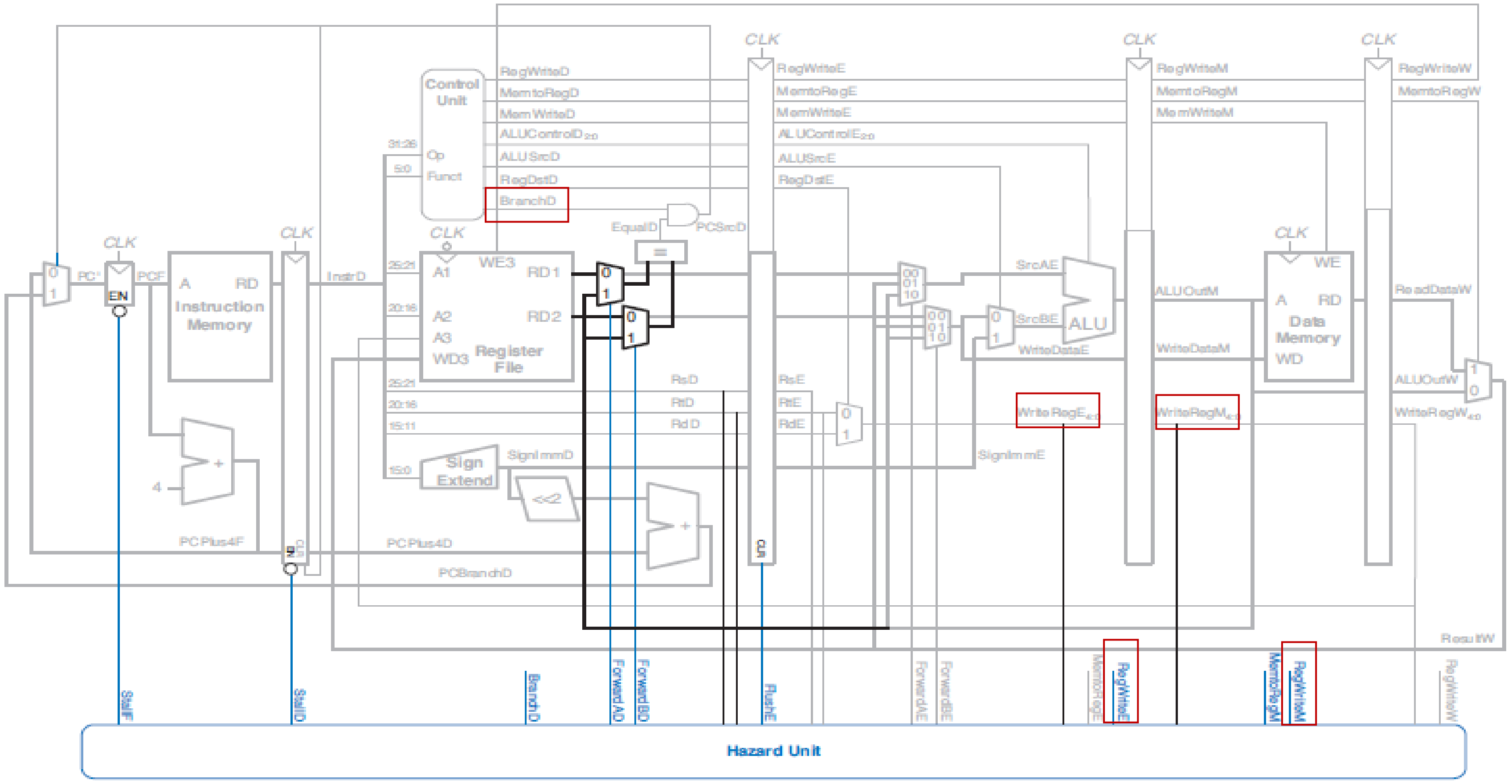


Control Hazards: Modified Pipeline

- What if one source operand of branch instruction was computed by a previous instruction and has not yet been updated into register file?

- One can use forwarding techniques
- Stalling technique can be used (lw-type)

Control Hazards & H/W-based solutions



Control Hazards & H/W-based solutions

- The function of the decode stage & forwarding logic

- FowardAD = (rsD != 0) AND (rsD == WriteRegM) AND RegWriteM
- FowardBD = (rtD != 0) AND (rtD == WriteRegM) AND RegWriteM

Control Hazards & H/W-based solutions

- The function of the stall detection logic for a branch instruction
  - What if one of the source operand is in Execute stage (R-type instr.)
  - What if one of the source operand is in Memory stage (lw-type instr.)
- Branchstall=

BranchD AND RegWriteE AND (WriteRegE == rsD OR WriteRegE == rtD)

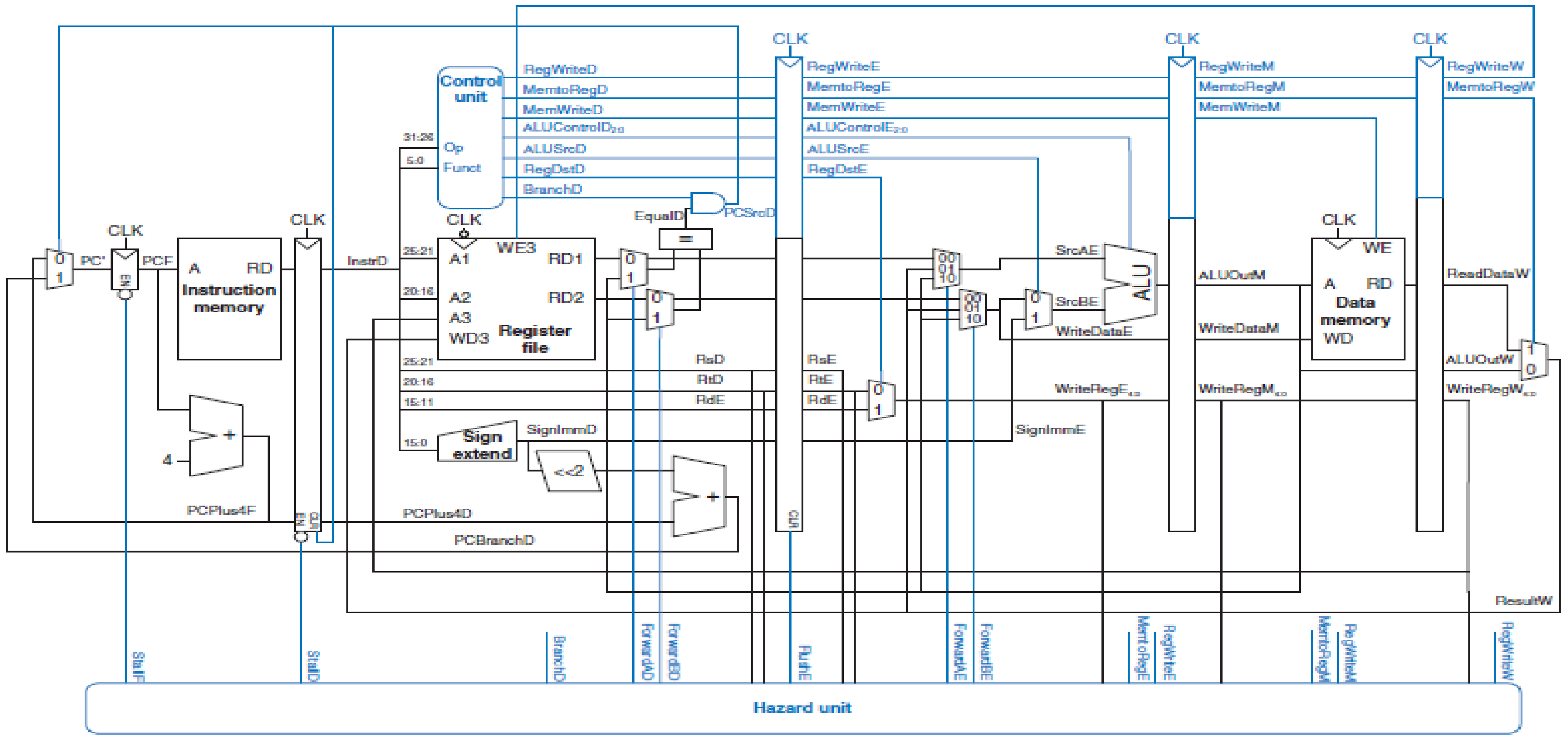
OR

BranchD AND MemtoRegM AND (WriteRegM == rsD OR WriteRegM == rtD)

Control Hazards & H/W-based solutions

- Now the pipelined processor stall due to either a load or a branch hazard
- StallF = StallD = FlushE = lwstall OR branchstall

Pipelined processor with full hazard handling unit



Pipelined processor with full hazard handling unit

- Determine the cycle time
  - consider the critical path
  - five pipeline stages
- Register file is written in the first half of the Writeback cycle and read in the second half of the Decode cycle
- The cycle time of the Decode and Writeback stages is <u>twice</u> the time necessary to do the half-cycle of work

- Cycle period, $T_c = max\begin{pmatrix} t_{pcq} + t_{mem} + t_{setup} \\ 2(t_{RFread} + t_{mux} + t_{eq} + t_{AND} + T_{mux} + t_{setup}) \\ t_{pcq} + t_{mux} + t_{mux} + t_{ALU} + t_{setup} \\ t_{pcq} + t_{memwrite} + t_{setup} \\ 2(t_{pcq} + t_{mux} + t_{RFWrite}) \end{pmatrix}$

Pipelined processor with full hazard handling unit

- XYZ needs to compare the pipelined processor performance to that of the single-cycle and multicycle processors considered in earlier. Most of the logic delays is given in Table. The other element delays are 40 ps for an equality comparator, 15 ps for an AND gate, 100 ps for a register file write, and 220 ps for a memory write. Help XYZ compare the execution time of 100 billion instructions of the program for each processor.

Parameter	Delay (ps)
t_{pcq}	30
t_{mem}	250
t_{RFread}	20
t_{ALU}	200
t_{mux}	25
$t_{RFwrite}$	20
t_{Setup}	20

Pipelined processor with full hazard handling unit

- According to Equation on slide 15, the cycle time of the pipelined processor is

  T_{c3} = max[30 + 250 + 20, 2(150 + 25 + 40 + 15 + 25 + 20), 30 + 25 + 25 + 200 +20, 30 + 220 + 20, 2(30 + 25 + 100)] = 550 ps.

- According to Equation of CPI, the total execution time is $T_3 = (100 \times 10^9$ instructions)(1.15 cycles/instruction) $(550 \times 10^{-12}$ s / cycle) = 63.3 seconds.
- For the single-cycle processor it is 92.5 seconds and 133.9 seconds for the multicycle processor.
- What can be observed?

Control Hazards & Software-based solutions

- Assumption: predict-not-taken or predict-untaken
- Where does this assumption come from?
- Compiler rearranges the code
- Control flow will change only when prediction is wrong
- Example:

Untaken branch instruction	IF	ID	EX	MEM	WB			
Instruction i+1		IF	ID	EX	MEM	WB			
Instruction i+2			IF	ID	EX	MEM	WB		
Instruction i+3				IF	ID	EX	MEM	WB	
Instruction i+4					IF	ID	EX	MEM	WB

Control Hazards & Software-based solutions

- Assumption: predict-not-taken or predict-untaken
- Where does this assumption come from?
- Compiler rearranges the code
- Control flow will change only when prediction is wrong
- Example:

Taken branch instruction	IF	ID	EX	MEM	WB				
Instruction i+1		IF	idle	idle	idle	idle			
Branch target			IF	ID	EX	MEM	WB		
Branch target +1				IF	ID	EX	MEM	WB	
Branch target +2					IF	ID	EX	MEM	WB

Control Hazards & Software-based solutions

- Assumption: predict-taken
- Compiler rearranges the code, for both the assumptions, so that the most frequent path matches the hardware’s choice
- Predict-taken has less advantages than predict-not-taken

Control Hazards & Software-based solutions

- Which assumption is suitable for pipelined MIPS-based architecture?
- Predict-untaken

Control Hazards & Software-based solutions

- How many types of branch are possible?
  - Forward branch
  - Backward branch
- For backward-type branch, predict-taken is suitable
- For forward-type branch, predict-untaken is suitable

Control Hazards & Software-based solutions

- Is there another way to specify predict-taken or predict-untaken in the branch instruction?
- A bit in the branch opcode
  - Bit set means predict-taken
  - Bit not set means predict-untaken
- Who will set or reset such bit?
  - Compiler

Control Hazards & Software-based solutions

- Can we eliminate one cycle delay associated in branch prediction-taken?
- Delayed branch technique

Control Hazards & Delayed branch (S/W-based approach)

- Delayed branch technique
- Examples

- Find useful & independent instruction
- How many delay slots?

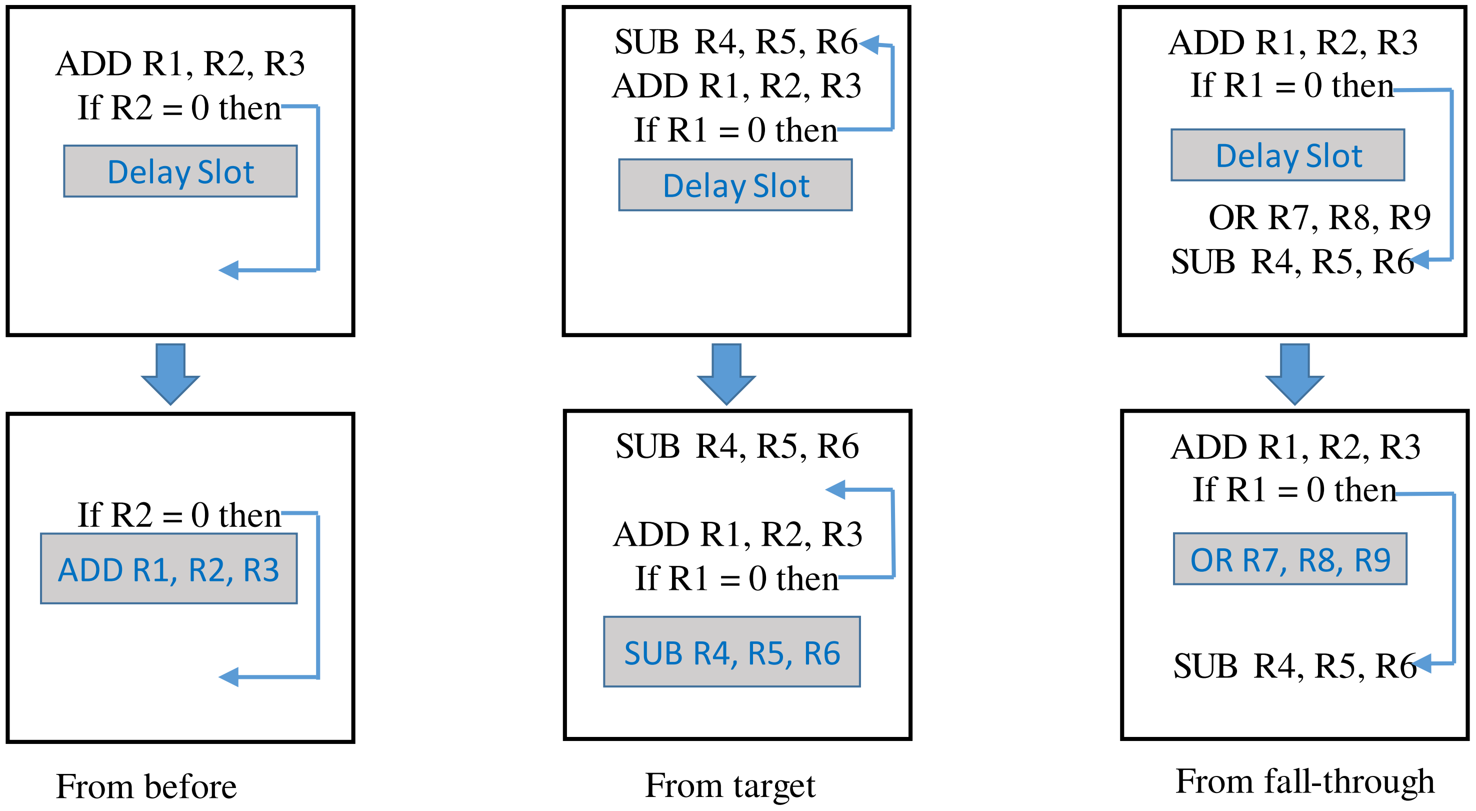
Untaken branch instruction	IF	ID	EX	MEM	WB				
Branch Delay Instruction i+1		IF	ID	EX	MEM	WB			
Instruction i+2			IF	ID	EX	MEM	WB		
Instruction i+3				IF	ID	EX	MEM	WB	
Instruction i+4					IF	ID	EX	MEM	WB
Taken branch instruction	IF	ID	EX	MEM	WB				
Branch Delay Instruction i+1		IF	ID	EX	MEM	WB			
Instruction i+2			IF	ID	EX	MEM	WB		
Instruction i+3				IF	ID	EX	MEM	WB	
Instruction i+4					IF	ID	EX	MEM	WB

Control Hazards & Delayed branch (S/W-based approach)

- How does compiler find useful & independent instruction?
- Can compiler always find such instruction?

Control Hazards & Delayed branch (S/W-based approach)

- How does compiler find useful & independent instruction?
- Can we put a branch instruction in the delay slot?



Control Hazards, static analysis & software-based solution

- Previous software-based approaches are based on static analysis
  - Assumption on Processor Design
    - Predict-not-taken
    - Predict-taken
  - Assumption on Control flow
    - Forward
    - Backward
  - Delayed branch
- Can we take decision during execution of the program?

Control Hazards and Dynamic Branch Prediction

- Previous software-based approaches are based on static analysis
- We give equal priority to each branch instruction, however reality may differ
- Branch prediction change if inputs of the program change
- We need to predict the branch behavior during runtime

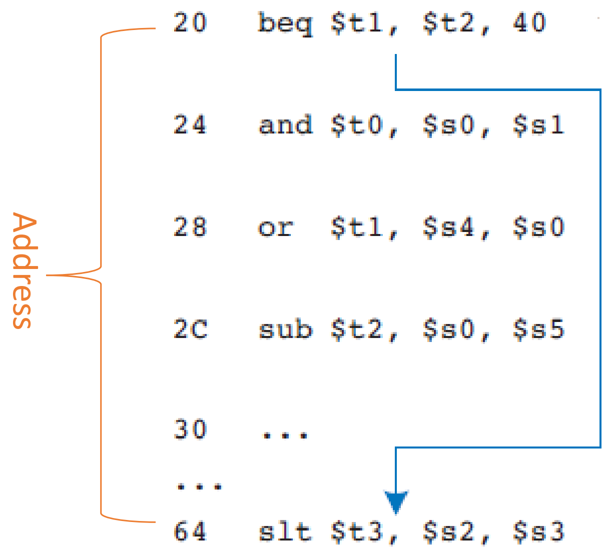
Control Hazards and Dynamic Branch Prediction

- How does one predict the behavior during execution?
- Exploit the previous execution history of the instruction
- What components needed to do such thing?
- Component to make prediction & store target address
- Which stage of the pipeline is suitable for that?
- IF-stage

Control Hazards and Dynamic Branch Prediction

- The *branch-target buffer* (BTB) or *branch-target (address) cache* (BTAC) is a branch-prediction <u>cache</u> that stores the predicted address for the next instruction after a branch

40 represent in 32-bits (16 +16)



```
20   beq $t1, $t2, 40
24   and $t0, $s0, $s1
28   or  $t1, $s4, $s0
2C   sub $t2, $s0, $s5
30   ...
...
64   slt $t3, $s2, $s3
```

Branch address	Target address	Prediction bits
20	64	
...	...	...

Control Hazards and Dynamic Branch Prediction

- The BTB is accessed during the IF stage
- It consists of a table with branch addresses, the corresponding target addresses, and prediction information
- The PC for the next instruction to fetch is compared with the entries in the BTB. If a matching entry is found in the BTB, fetching can start immediately at the target address

Control Hazards and Dynamic Branch Prediction

- What is this prediction bit in the BTB?
- How many bits are needed?

Control Hazards and Dynamic Branch Prediction

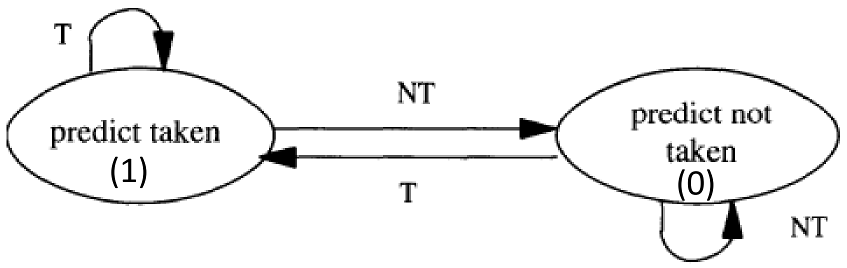
- Example: branch outcome sequence (T – Taken & N – Not Taken)
  - T N T N T N T N T N
- How does one design such a predictor?

Control Hazards and Dynamic Branch Prediction

- One bit predictor
  - If the bit is set, the branch is predicted taken
  - If the bit is not set, the branch is predicted not taken
  - In the case of a misprediction, the bit state is reversed and stored back and so is the prediction direction
- How does one design such predictor?
  - Encode the states
  - Relation among the states
  - Finite State Machine (FSM)

Control Hazards and Dynamic Branch Prediction

- One bit predictor
  - If the bit is set, the branch is predicted taken
  - If the bit is not set, the branch is predicted not taken
  - In the case of a misprediction, the bit state is reversed and stored back and so is the prediction direction



T for Taken
NT for Not Taken

Is there any shortcoming of this approach?

Example: see in the BranchPrediction.xlsx

Control Hazards and Dynamic Branch Prediction

- <u>Shortcoming</u> of one bit predictor
- Predict taken always
- Predict incorrectly twice (why?), rather than once, when it is not taken
- What if we have the nested loops
  - Two misprediction for inner loop
    - Entry in the loop
    - Exit from the loop
  - How does one avoid such double misprediction?

Control Hazards and Dynamic Branch Prediction

Two-bit predictor:

- Two-bits are in each entry in the BHT
- The two bits stand for the prediction states:
  - "predict strongly taken"
  - "predict weakly taken"
  - "predict strongly not taken"
  - "predict weakly not taken"

- For a missprediction in the "strongly" state cases, the <u>prediction direction is not changed</u>, rather the prediction goes into the respective "weakly" state

- A prediction must <u>miss twice, before changing the state</u>

Control Hazards and Dynamic Branch Prediction

- How does one design such 2-bits predictor?

Control Hazards and Dynamic Branch Prediction

- How does one design such 2-bits predictor?
  - Encode the different states
  - Relation among the states
  - Finite State Machine (FSM)

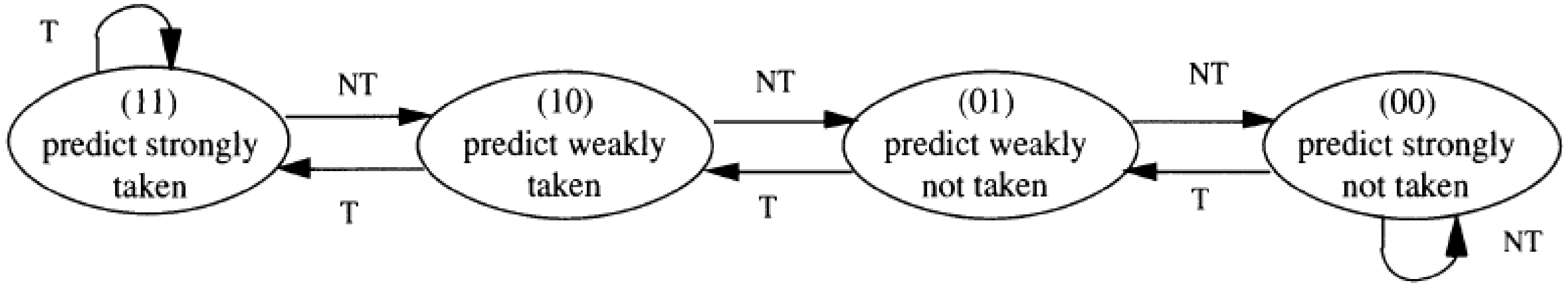
Control Hazards and Dynamic Branch Prediction

- Two kinds of 2-bits prediction methodology
  - The saturation up-down counter
  - Others

Control Hazards and Dynamic Branch Prediction

- The saturation up-down counter
  - Taken branch
    - Increment
  - Not taken
    - Decrement
  - Saturation
  - MSB of a state determine the prediction

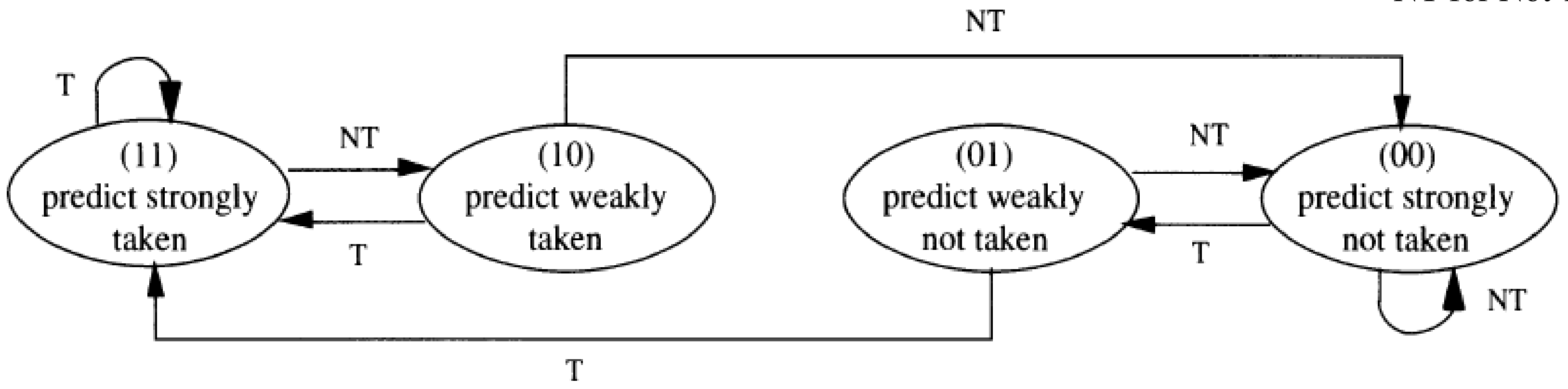
T for Taken
NT for Not Taken



Control Hazards and Dynamic Branch Prediction

- Other methodology
  - It differs from the saturation up-down counter method by changing directly from the "weakly" to the "strongly" states, in case of misprediction
  - Minimize the switching (or power consumption)

T for Taken
NT for Not Taken


NT
T
(11) predict strongly taken
NT
T
(10) predict weakly taken
(01) predict weakly not taken
NT
T
(00) predict strongly not taken
NT
T

Example: see in the BranchPrediction.xlsx

Control Hazards and Dynamic Branch Prediction

- n-bits predictor
  - n-bit counter (0 to 2^n-1)
  - Taken when counter value is one-half of the max. value (2^n-1)
  - Otherwise, Not taken
  - Studies of n-bits predictor have shown that 2-bits predictor do almost well, thus most systems rely on 2-bits predictor

Summary

- Control hazard
- Performance analysis
- Flush the pipeline
- Hardware-based solution technique
  - Take decision at ID stage
- Software-based solution technique
  - Predict-taken
  - Predict-untaken
  - Delayed branch
- Dynamic branch prediction techniques