

Administrative Matters

- **Homework #5**
 - Due Friday 11/16 at 2PM through moodle
- **Midterm #2**
 - Monday, 11/19
 - Cover chapter 5 and 6
 - Cover homework 4 and 5
 - 15% of your grade

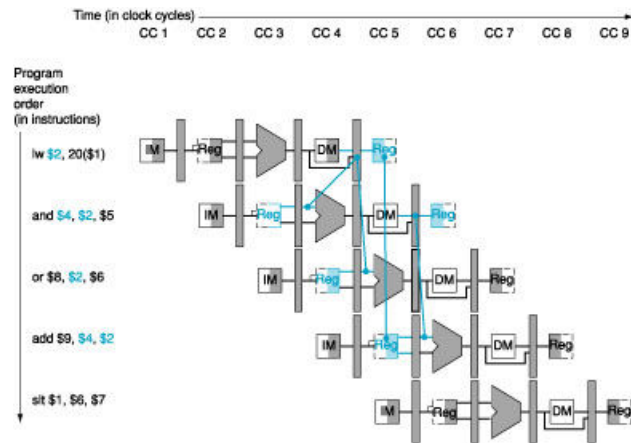


Chapter Six

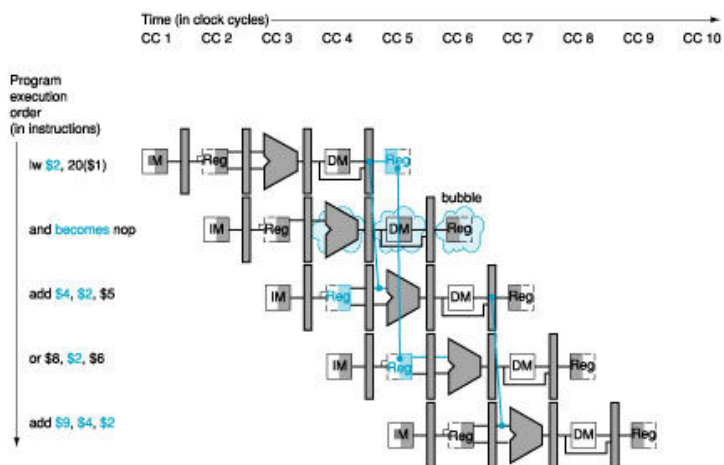
Enhancing Performance with Pipelining

(continue)

Load-used data hazard



Must stall



Precise stalling condition

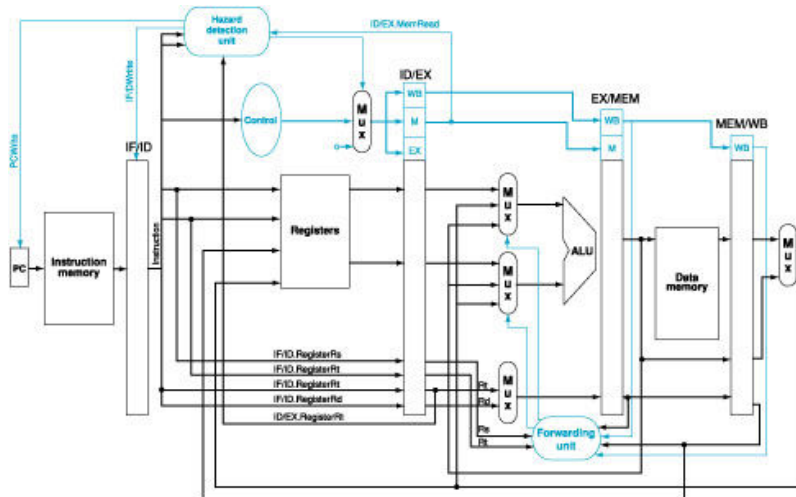
```

if (ID/EX.MemRead and
    ((ID/EX.RegisterRt = IF/ID.RegisterRs) or
     (ID/EX.RegisterRt = IF/ID.RegisterRt)))
    stall the pipeline
    
```

- Already know at decode stage, so can “change” instruction to NOP
- Set the control lines to do “nothing”
 - Don’t write anything
 - Don’t need to read or execute anything
 - Reset the PC also

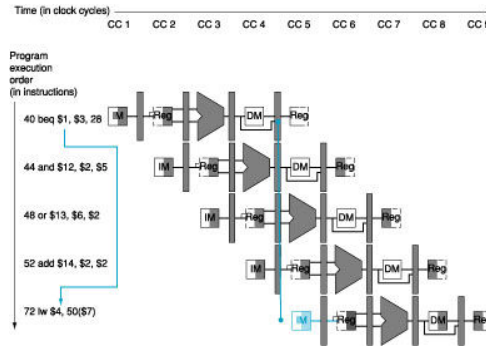
Instruction	Execution/Address Calculation stage control lines				Memory access stage control lines			stage control lines	
	Reg Dst	ALU Op1	ALU Op0	ALU Src	Branch	Mem Read	Mem Write	Reg write	Memto Reg
Rformat	1	1	0	0	0	0	0	1	0
Lw	0	0	0	1	0	1	0	1	1
Sw	X	0	0	1	0	0	1	0	X
beg	X	0	1	0	1	0	0	0	X

Architecture including data hazard stuff



Branch Hazards

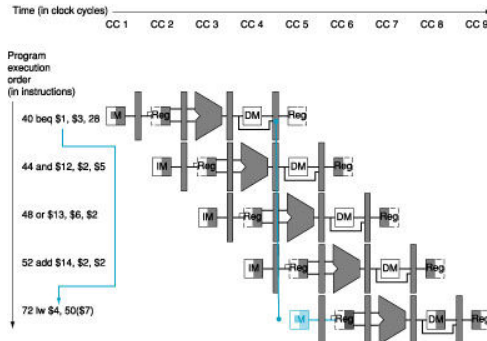
- When we decide to branch, other instructions are in the pipeline!



- What to do?
 - Assume Branch Not taken, Dynamic Branch prediction...

Assume Branch not taken

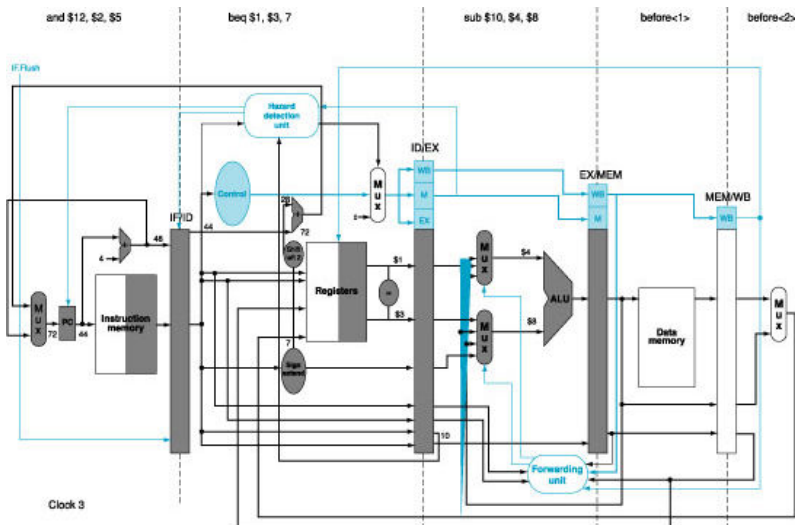
- If we are wrong, the “assumed” instructions must be discarded
 - Up to now, branch result not known until EX/MEM
 - To flush the pipeline
 - Change control to 0, for instructions in IF, ID, EX stage

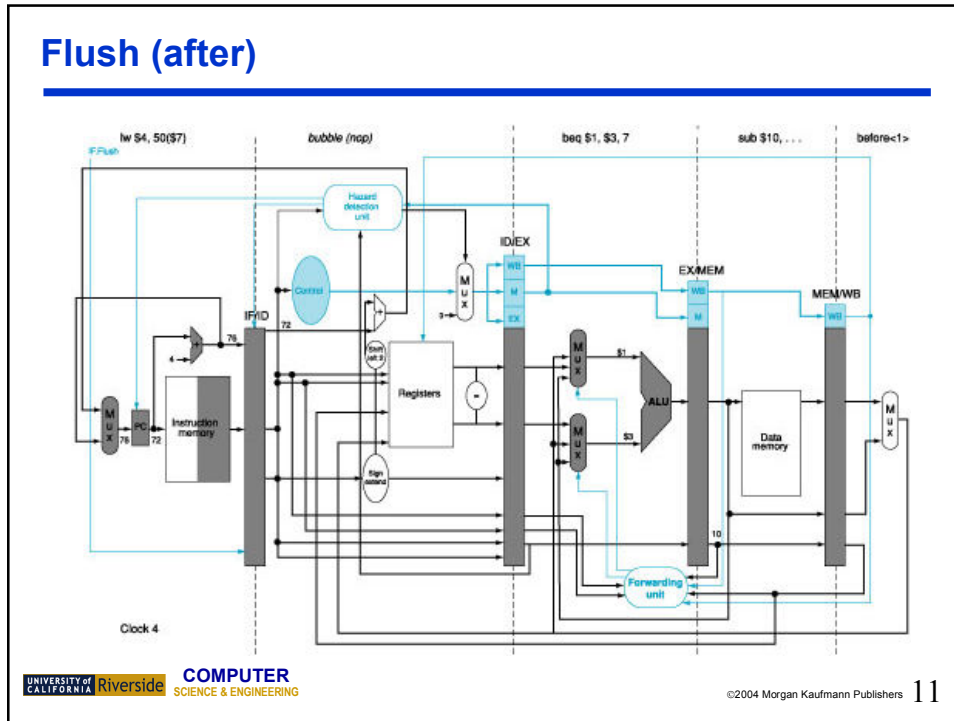


Reducing the delay of taken branches

- Move address calculation to ID
- Move comparison to ID also
 - Add extra hardware
- At the end of ID, already know where to go for next instruction
- Only 1 clock stall
- However
 - Extra forwarding
 - Still stall
- Of course, this is an optimization, we can always flush more...

Flush (before)



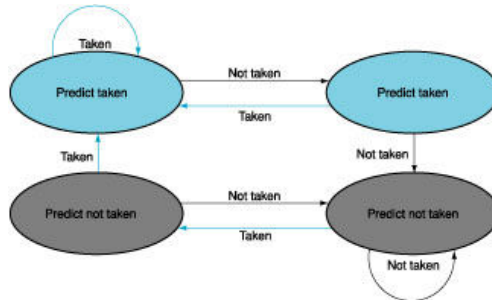


Branches

- If the branch is taken, we have a penalty of one cycle
- For our simple design, this is reasonable
- With deeper pipelines, penalty increases and static branch prediction drastically hurts performance
- Solution: dynamic branch prediction
- **Branch Prediction Buffer**
 - Remember how previously this branch resolves, guess that way..
 - E.g. A loop that loops 9 times (test 10 times) will have
 - 1 initial miss (last time at the branch is also not taken)
 - 1 final miss
 - 80% accuracy

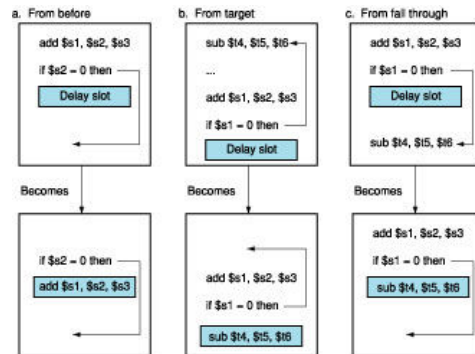
2 bit prediction

- If we use two bits to predict
 - No more initial miss (take two miss to flip)
 - 90% accuracy in prediction



Delayed Branch

- Insert into delay slot instructions that always execute
- Hard to find good instructions
- Hard to find which way to fill
 - Need prediction as to take/not taken (to improve performance)
- Losing popularity...

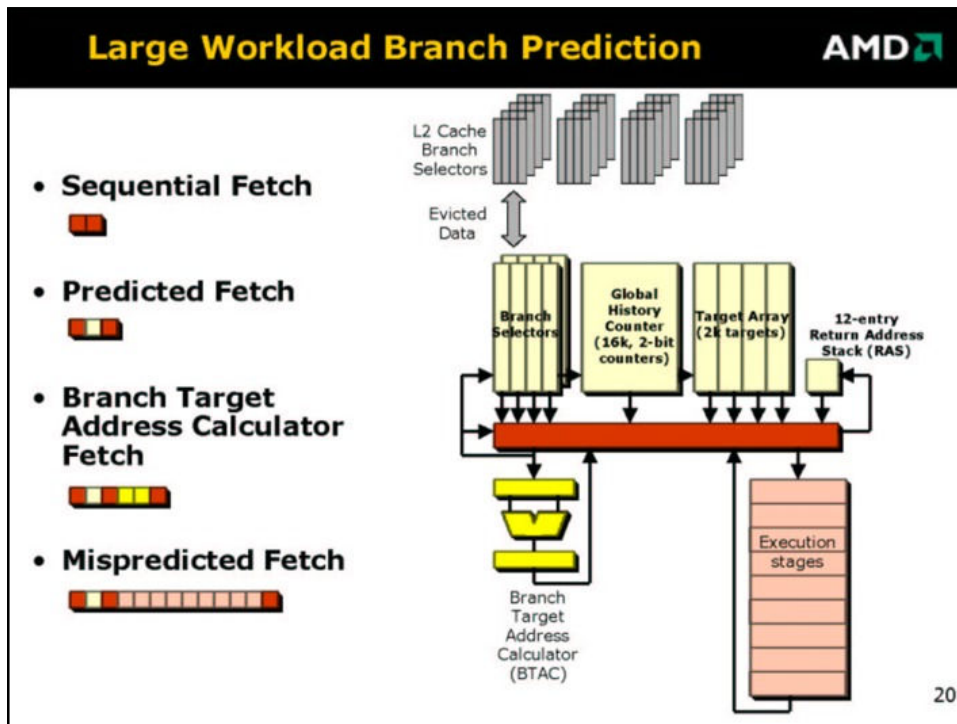


Branch Prediction

- **Sophisticated Techniques:**
 - A “branch target buffer” to help us look up the destination
 - Correlating predictors that base prediction on global behavior and recently executed branches
 - e.g., prediction for a specific branch instruction based on what happened in previous branches
 - Tournament predictors that use different types of prediction strategies
 - and keep track of which one is performing best.
- Branch prediction is especially important because it enables other more advanced pipelining techniques to be effective!
- Modern processors predict correctly 95% of the time!

Quick Question

- **3 branch prediction strategy**
 - Branch not taken
 - Predict taken
 - Dynamic prediction (90% accuracy)
- **0 cycle penalty when they predict correctly and 2 cycles when they are wrong**
- **Which is better if we have:**
 - **A branch that is taken with 5% frequency**
 - $1 * .95 + 3 * .05 = 1.1$
 - $3 * .95 + 1 * .05 = 2.9$
 - $1 * .9 + 3 * .1 = 1.2$
 - **A branch that is taken with 95% frequency**
 - $3 * .95 + 1 * .05 = 2.9$
 - $1 * .95 + 3 * .05 = 1.1$
 - $1 * .9 + 3 * .1 = 1.2$
 - **A branch that is taken with 70% frequency**
 - $1 * .3 + 3 * .7 = 2.4$
 - $3 * .3 + 1 * .7 = 1.6$
 - $1 * .9 + 3 * .1 = 1.2$



Comparing performance

- **Machine 1**
 - Single cycle, 200ps memory, 100ps ALU, 50ps register
 - Clock cycle of $200+50+100+200+50=600$, CPI=1
 - Time per instruction = 600 ps
- **Machine 2**
 - Multicycle, 25% loads, 10% stores, 11% branches, 2% jump, 52% ALU, 5 cycle loads, 4 cycle stores, 4 cycle ALU, 3 cycle Branches, and 3 cycle jumps
 - $CPI = 0.25*5+0.1*4+0.52*4+0.11*3+0.02*3=4.12$
 - Clock = max of stages = 200, Time per instruction = 824 ps
- **Machine 3**
 - Pipelined, half of loads take 2 cycles, 25% of branches are miss predicted hence take 2 cycles, jump always take 2 cycles
 - $CPI = 0.25*1.5+0.1*1+0.52*1+0.11*1.25+0.02*2=1.17$
 - Clock = max of stages = 200, Time per instruction = 234 ps