Chapter Six

- Forwarding
- Hazards
- Branching

EE3055
Web: www.csc.gatech.edu/copeland/3055

- Use temporary results, don’t wait for them to be written
- register file forwarding to handle read/write to same register
- ALU forwarding

Forwarding

Control

A

B

Register

Set
• Load word can still cause a hazard: (structural hazard)
  – an instruction tries to read a register following a load instruction that writes to the same register.

• Thus, we need a hazard detection unit to "stall" the load instruction.

Can't always forward:

Stalling

• We can stall the pipeline by keeping an instruction in the same stage. This stall may sometimes be avoided by reordering instructions.

Hazard Detection Unit

• Stall by letting an instruction that won't write anything go forward.

Forwarding Unit

Hazard Detection Unit
When we decide to branch, other instructions are in the pipeline! We are predicting "branch not taken" – need to add hardware for flushing instructions if we are wrong.

Branch Hazards

Special ALU (=) makes branch decision in Instr.

Decode stage

M Reg

If Branch Taken

Flushing Instructions

Forwarding Unit

Hazard Detection Unit

Instruction Flush

Control

New mini-ALU to detect <, >, or = so branch condition can be determined without going through the normal ALU loop:

```
mul $t4, $t3, $t2
move $t3, $t4
sgt $s6, $t2,1
addi $t2, $t2, -1
bne $s6, $0, loop
mul $t4, $t3, $t2
la $a0, newline
li $v0, 4
syscall
```
loop:
mul $t4, $t3, $t2
move $t3, $t4
sgt $s6, $t2, 1
addi $t2, $t2, -1
bne $s6, $0, loop
mul $t4, $t3, $t2
la $a0, newline
li $v0, 4
syscall

CPU hardware must contain a table of branch addresses with a bit to show whether branch was taken last time. This bit determines the present assumption about whether the branch will be taken or not. When the prediction is wrong, there must be a "flush" which wastes a clock cycle.

Branch Strategies - Dynamic Branch Prediction.

Improving Performance
• Try and avoid stalls! E.g., reorder these instructions:
  lw $t0, 0($t1)
  lw $t2, 4($t1)
  sw $t2, 0($t1)
  sw $t0, 4($t1)
• Add a "branch delay slot" - the next instruction after a branch is always executed - rely on compiler to "fill" the slot with something useful
• Superscalar: start more than one instruction in the same cycle

Even with forwarding, $t2 will not be ready until one more clock cycle has passed.

Dynamic Scheduling
• The hardware performs the "scheduling" - hardware tries to find instructions to execute - out of order execution is possible - speculative execution and dynamic branch prediction
• All modern processors are very complicated - DEC Alpha 21264: 9 stage pipeline, 6 instruction issue - PowerPC and Pentium: branch history table - Compiler technology important
• "Superscalar" CPU - two or more instructions in parallel - speed increased, but not n-fold - more cases of data dependency and hazards.
Exception Handling in a Pipelined Computer

- The hardware must determine which instruction threw the exception:
  - Illegal instruction -> instruction in ID (instruction decode) stage
  - ALU overflow -> instruction in Execute stage.

- Executing instructions turned into NOP's (flushed).

- I/O interrupts and hardware malfunctions not due to instruction:
  - Some flexibility to finish some instructions, flush later ones.

- Method of flushing various stages:
  - Next Instruction set to Interrupt Handler - hex 4000 0040
  - Instruction Decode - logic turns instruction to all zeroes (NOP)
  - Execute - Ex.Flush signal causes control lines to go to zero when overflow detected in this stage.

- Operating system will kill the program and send an error message for:
  - Illegal instruction, arithmetic overflow, or hardware malfunction.

- Operating system will save the program, store state, and restart for:
  - I/O Interrupt or Operating System Service Call.