CS184b: Computer Architecture [Single Threaded Architecture: abstractions, quantification, and optimizations]

Day6: January 23, 2000

Control Hazards

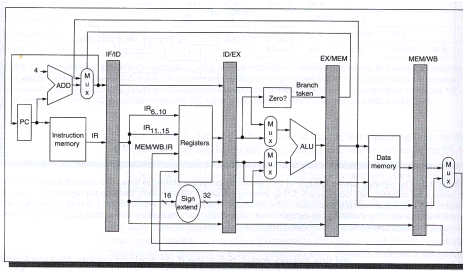
(Pipelined ISA)

Today

- Control Hazards
- Techniques to reduce impact
- Exception
  - what
  - why
  - challenges

Last Time

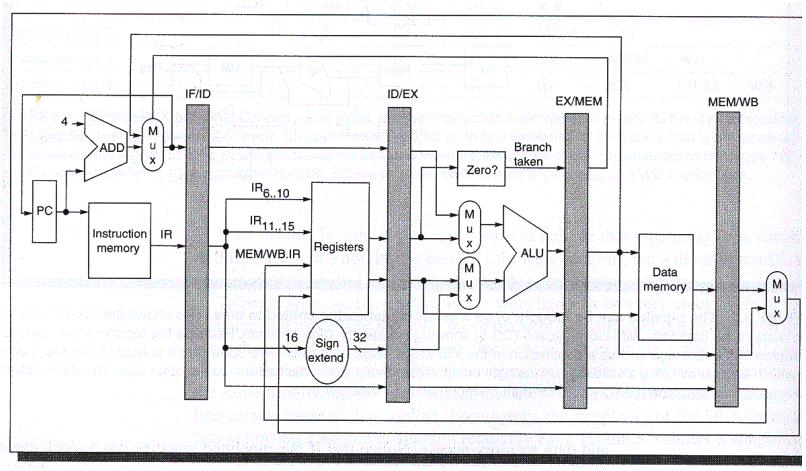
- Pipelining
- Cycles in flow graph
- Avoiding
- or accommodating dynamically



Hazards

- Structural (resource bound)
- Data (value timing)
- Control (knowing where to go next)

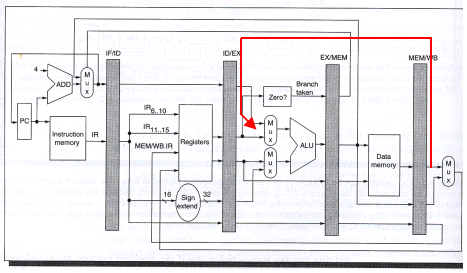
Pipeline DLX



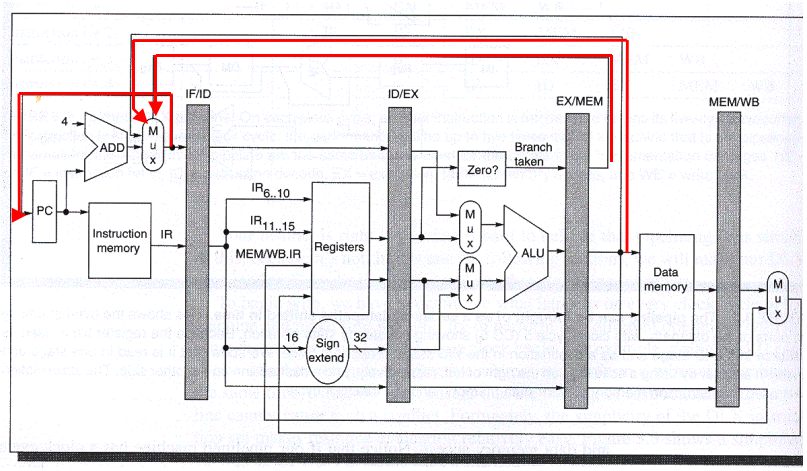
DLX pipelined datapath from H&P (Fig. 3.4)

Data Hazards

- Didn't effect IF or ID
- Worst-case, one cycle stall



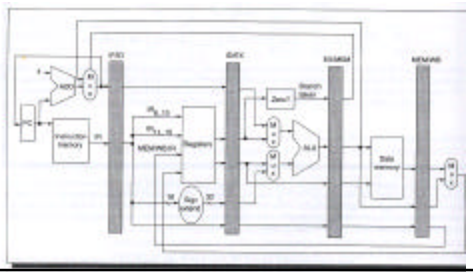
Pipeline DLX



DLX pipelined datapath from H&P (Fig. 3.4)

Control Hazard

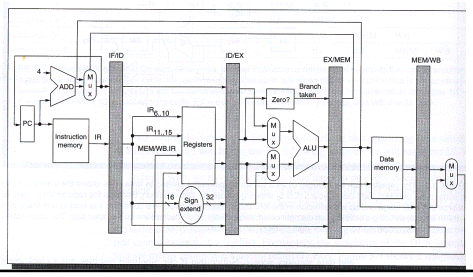
IF ID EX MEM WB
IF ID EX MEM WB
IF ID EX MEM WB
IF ID EX MEM WB
IF ID EX MEM WB



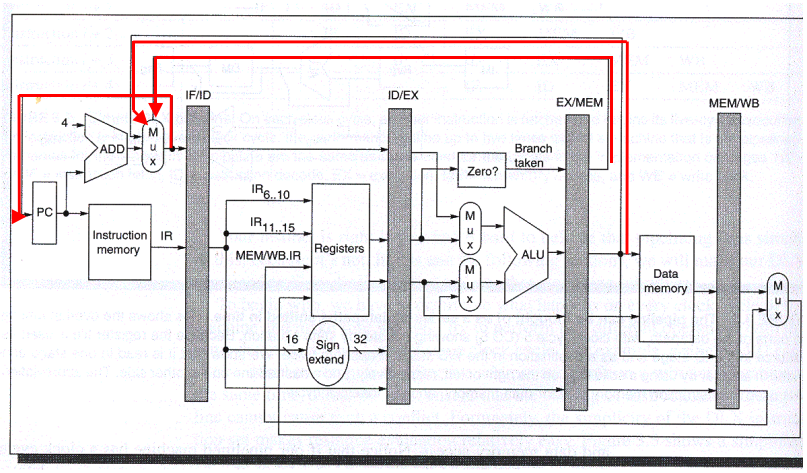
Control Hazard

IF ID EX MEM WB

IF ID EX MEM WB



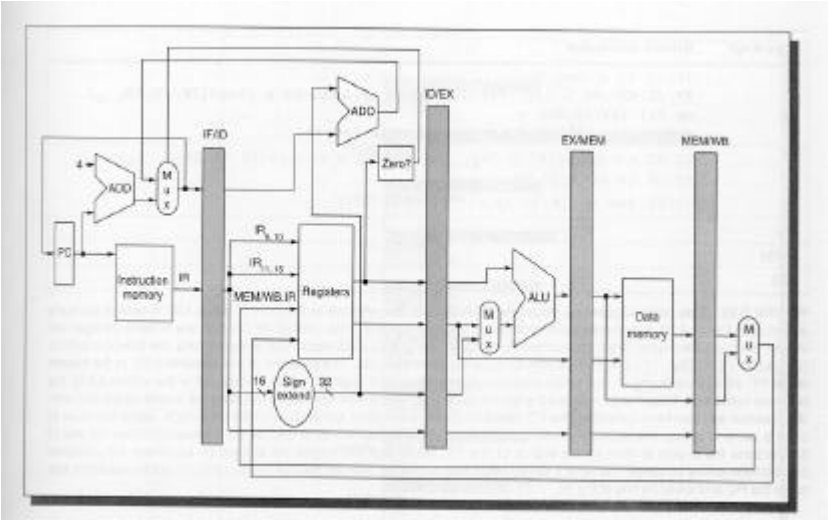
What can we do?



What can we do?

- Move earlier
  - tighten cycle
- "Guess" direction
  - predict (and squash)
- Accepted delayed transfer as part of arch.
  - Branch delay slot

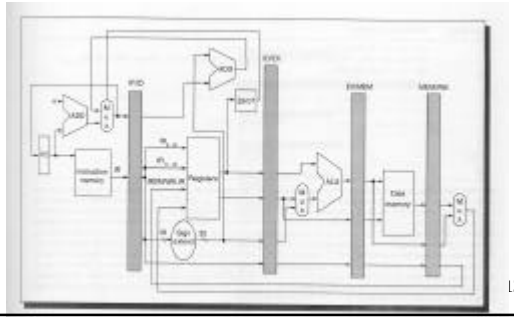
Revised Pipeline



DLX repipelined datapath from H&P (Fig. 3.22)

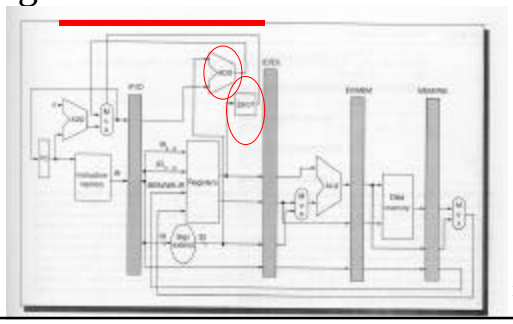
Consequence?

```
IF   ID   EX   MEM  WB
     IF   ID   EX   MEM  WB
          IF   ID   EX   MEM  WB
               IF   ID   EX   MEM  WB
                    IF   ID   EX   MEM  WB
```

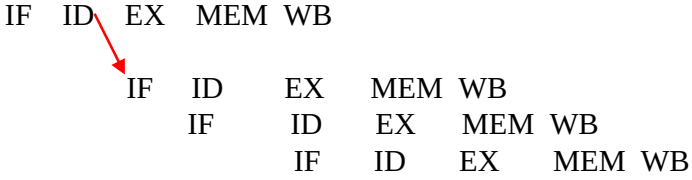


Consequence

- Smaller cycle
- Longer ID stage delay
- Need separate Adder
- Not branch to reg.?

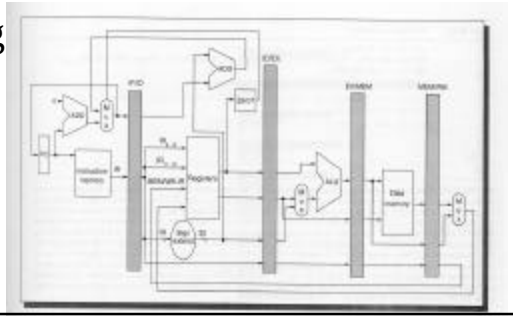


Pipeline



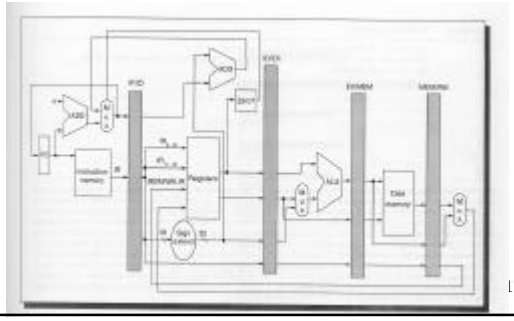
Avoiding Lost Cycles

- Do know where to go in not-taken case
  - just keep incrementing PC
- "Guess" taken
- Begin Executing
- Squash if Wrong



Predict Branch Taken

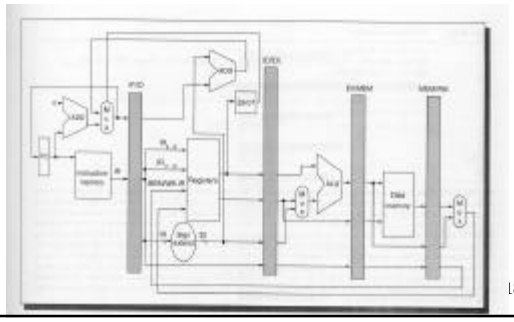
```
Branch:   IF   ID   EX   MEM  WB
Branch+1:      IF   ID   EX   MEM  WB
Branch+2:           IF   ID   EX   MEM  WB
                    IF   ID   EX   MEM  WB
```



Predict Branch Taken (**not**)

```
Branch:   IF   ID   EX   MEM  WB
Branch+1:      IF   ID   --   --   --
Branch+2:           IF   ID   EX   MEM  WB
                    IF   ID   EX   MEM  WB
```

Squash ok:
- no state change,
- no effect of exec op



Avoiding Lost Cycle (2)

- Solve like load latency
  - separate load request
  - from load use
- Separate branch instruction (computation)
- From branch effect
- Architecturally specify
  - branch not take effect until X cycles later

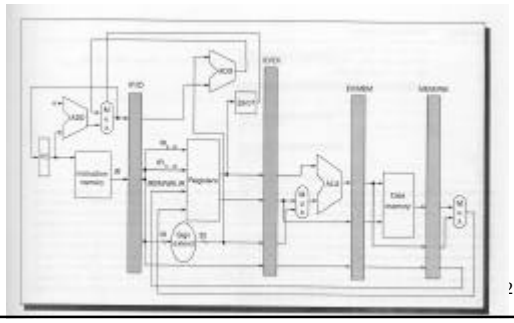
Branch Delay Slot

- SUB R1,R2,R3
- BEQZ R1,exit
- ADD R4,R5,R6 // always executed
- SUB R1,R4,R3

- exit:
- SW R3,4(R11)

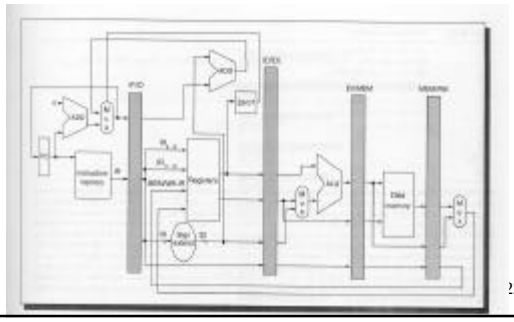
Branch Taken

```
Branch:   IF   ID   EX   MEM  WB
B-Delay:       IF   ID   EX   MEM  WB
Target  :           IF   ID   EX   MEM  WB
                         IF   ID   EX   MEM  WB
```



Branch Not Taken

```
Branch:   IF   ID   EX   MEM  WB
B-Delay:       IF   ID   EX   MEM  WB
Branch+2:         IF   ID   EX   MEM  WB
                       IF   ID   EX   MEM  WB
```



More Control Fun to Come...

- Knowing what to run next can be big limit to exploiting parallelism (deep pipelining)
- ILP need more branch prediction

Exceptions

What?

- Control transfer away from “normal” execution
- Every instruction
  - a conditional, multiway branch?
  - (branch and link)

What (examples)

- Page Fault
- System call
- Interrupt (event)
  - io
  - timer
- Unknown Instruction

Why?

- Cases represented are uncommon
- Instructions explicitly checking for cases add cycles
  - …lots of cycles to check all cases
  - when seldom occur
- Long instructions to describe all ways can branch
  - more compact to “configure” implicit places to go

Properties/Types

- synch/Asynch
- request/coerced
- maskable?
- within/between instructions
- resume/terminate

How make implementation difficult?

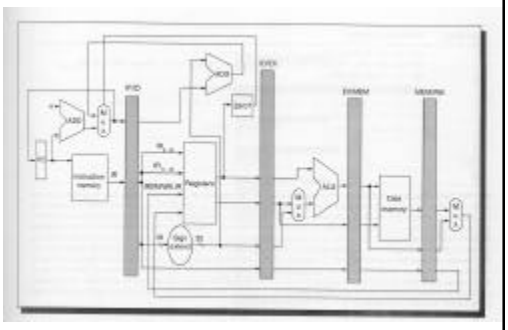
- Need to preserve sequential instruction abstraction
- Creates cause to see what happens between instructions
  - need to provide clean state view
- Instruction fail and need to be "restarted"
  - e.g. page fault

Hard cases

- Synchronous
- within instruction
- restartable

- latencies or parallelism allow out-of-order completion

Hazards



- LW R1,4(R2)
- ADD R3,R4,R3

- Case:
  - DLX
  - Pipeline where WB can occur before MEM
- May be correct to complete ADD
  - no hazards
  - but not restartable when fault on LW address

Restart Hazards

- LW R1,4(R2)
- ADD R3,R4,R3

- Restart/rerun
  - can get wrong answer by executing instruction again

Next Time

- Talk about how to solve
  - (read Smith/Imprecise exception paper)
- …and get started in ILP
  - note: read tomasulo along with HP4.2

Big Ideas

- Maintaining Abstraction
- Challenge of deciding what to do next
  - cyclic dependency
- Minimizing cost thereof
  - pipeline structure (minimize latency)
  - branch delay
  - prediction
- Common Case
  - exceptions
  - control flow