CS429: Computer Organization and Architecture

Pipeline III

Warren Hunt, Jr. and Bill Young
Department of Computer Sciences
University of Texas at Austin

Last updated: November 4, 2014 at 12:58

How Do We Fix the Pipeline?

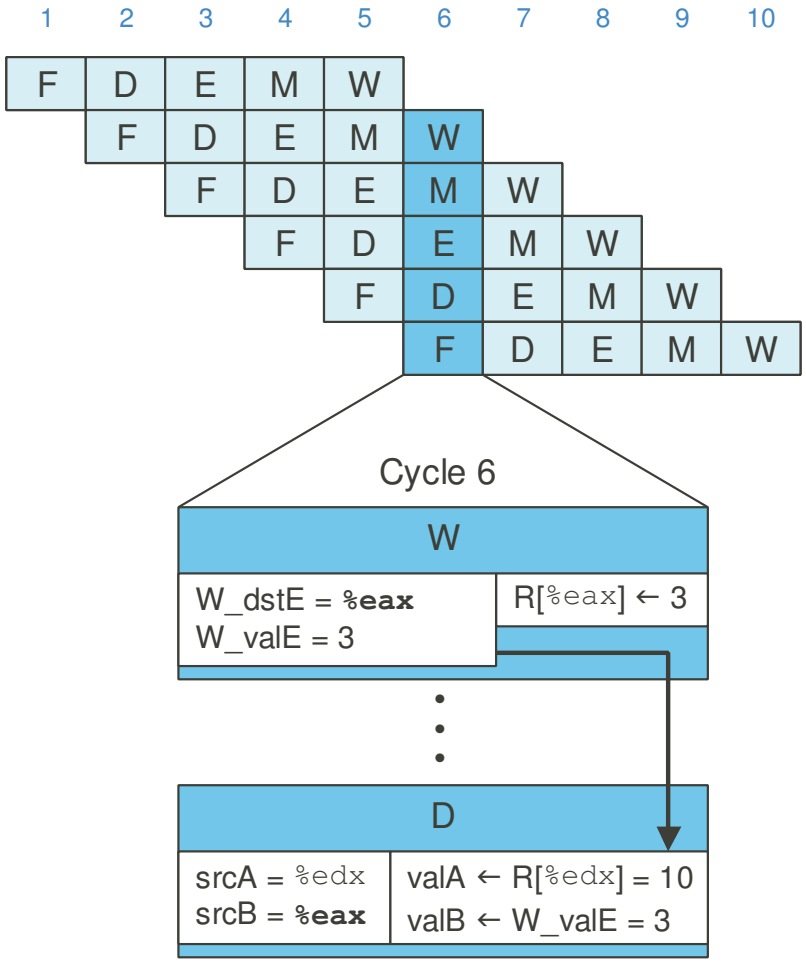
- Pad the program with NOPs: Yuck!
- Stall the pipeline
  - Data hazards:
    - Wait for producing instruction to complete
    - Then proceed with consuming instruction
  - Control hazards:
    - Wait until new PC has been determined
    - Then begin fetching
  - How is this better than inserting NOPs into the program?
- Forward data within the pipeline
  - Grab the result from somewhere in the pipe
    - After it has been computed
    - But before it has been written back
  - This gives an opportunity to avoid performance degradation due to hazards!

Data Forwarding

- **Naive pipeline**
  - Register isn't written until completion of write-back stage.
  - Source operands read from register file in decode stage.
  - Needs to be in register file at start of stage.
- **Observation:** value is generated in execute or memory stage..
- **Trick:**
  - Pass value directly from generating instruction to decode stage.
  - Needs to be available at end of decode stage.

Data Forwarding Example

```
# prog2
0x000: irmovl $10,%edx
0x006: irmovl  $3,%eax
0x00c: nop
0x00d: nop
0x00e: addl %edx,%eax
0x010: halt
```



- `irmovl` in write back stage
- Destination value in W pipeline register
- Forward as valB for decode stage

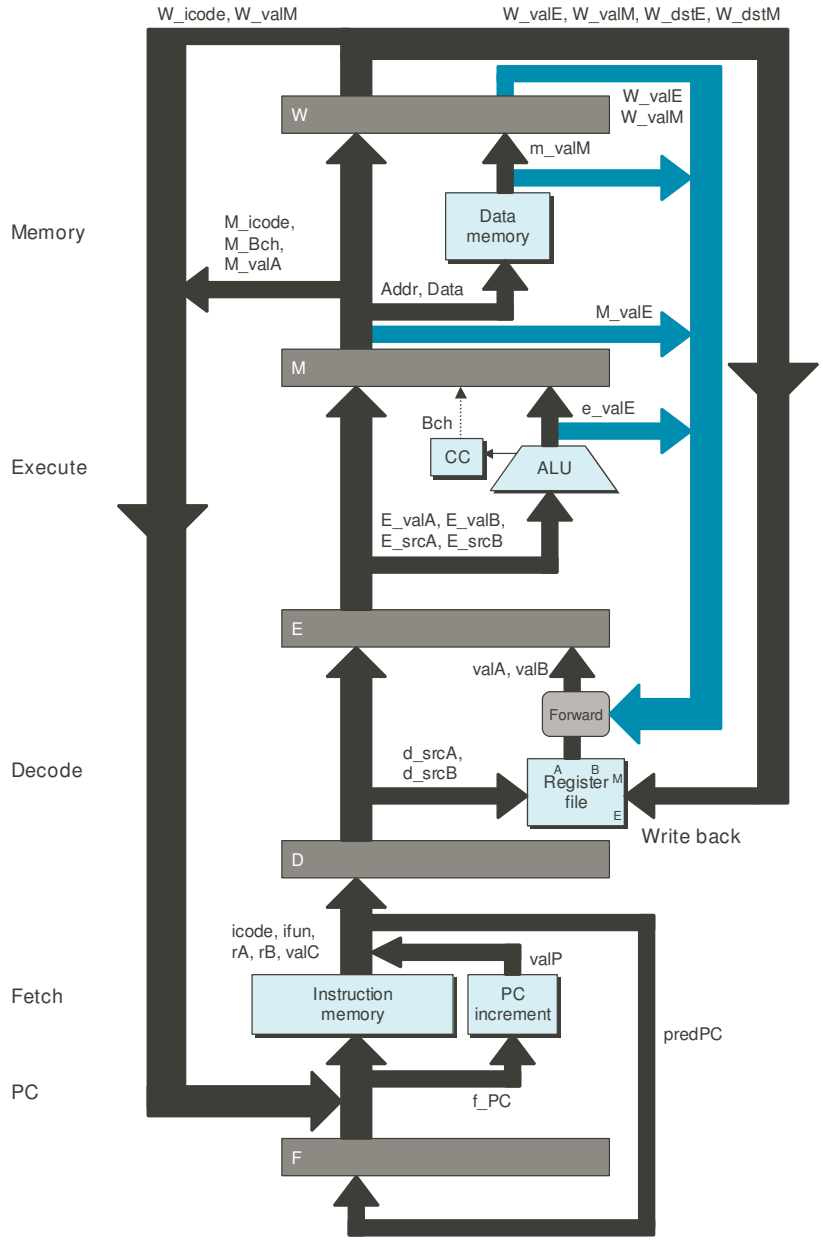
Bypass Paths

Decode Stage:

- Forwarding logic selects valA and valB
- Normally from register file
- Forwarding: get valA or valB from later pipeline stage

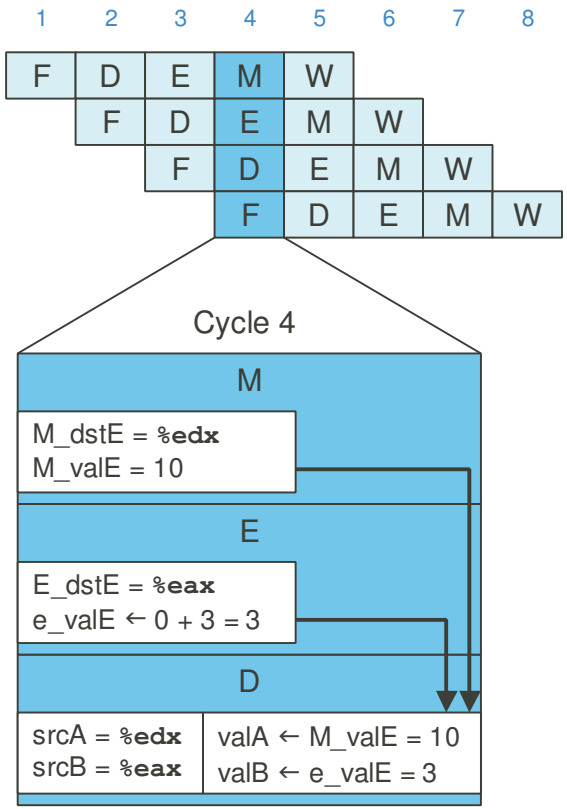
Forwarding Sources:

- Execute: valE
- Memory: valE, valM
- Write back: valE, valM



Data Forwarding Example 2

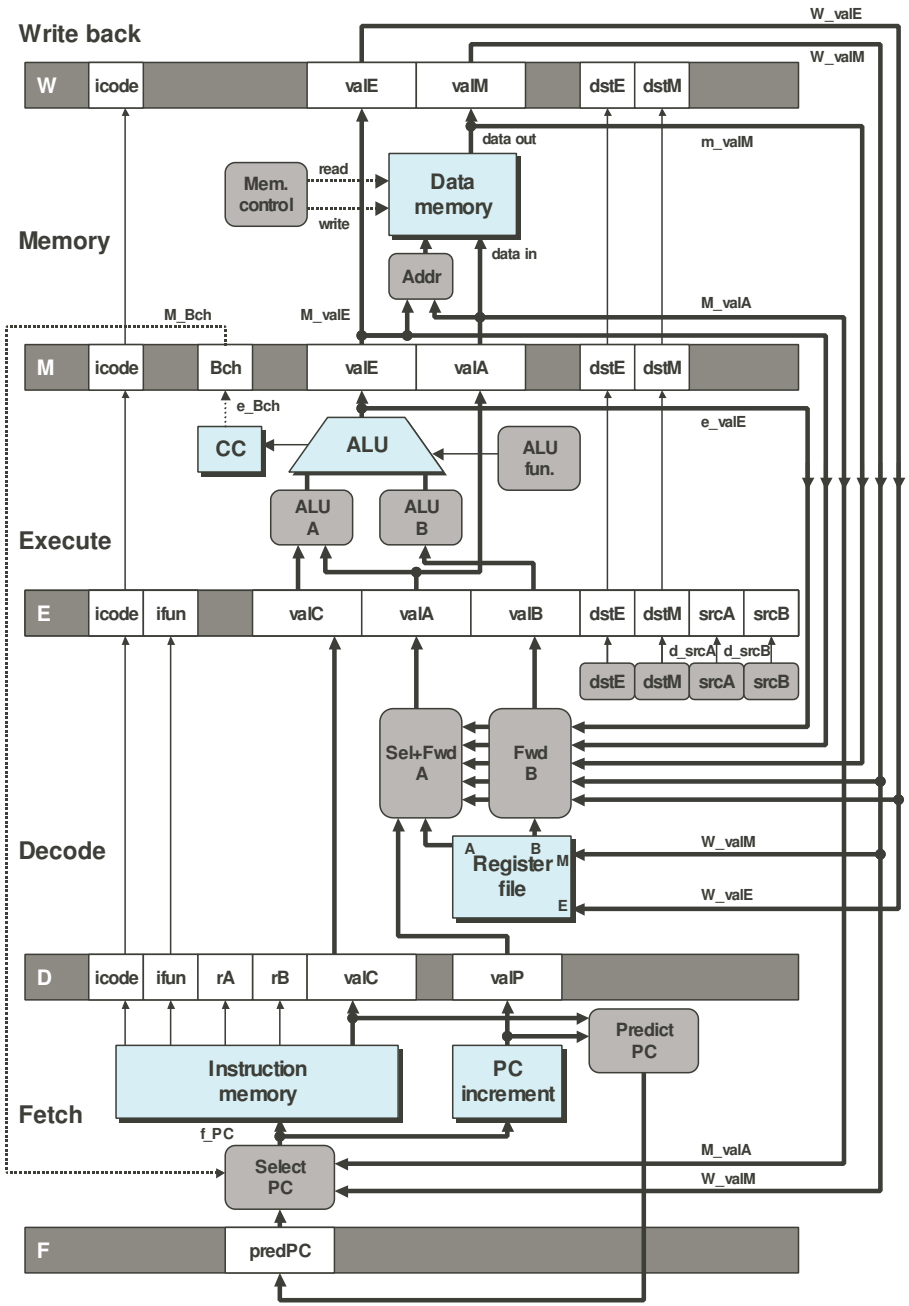
```
# prog4
0x000: irmovl $10,%edx
0x006: irmovl  $3,%eax
0x00c: addl %edx,%eax
0x00e: halt
```



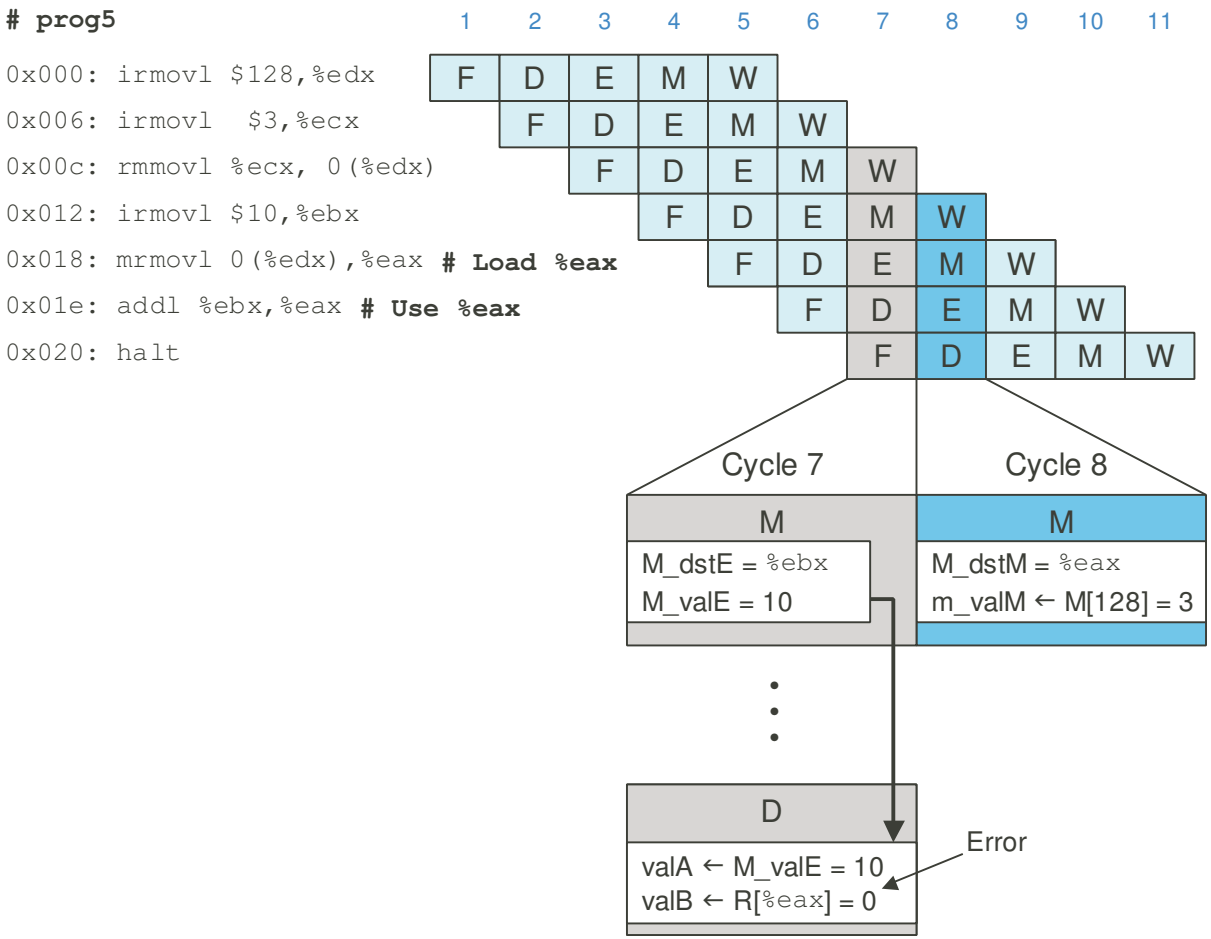
- Register %edx: generated by ALU during previous cycle; forwarded from memory as valA.
- Register %eax: value just generated by ALU; forward from execute as valB.

Implementing Forwarding

- Add new feedback paths from E, M, and W pipeline registers into decode stage.
- Create logic blocks to select from multiple sources for valA and valB in decode stage.


Write back
W
icode
valE
valM
dstE
dstM
W_valE
W_valM
data out
m_valM
Memory
Mem. control
read
write
Data memory
data in
Addr
M_Bch
M_valE
M_valA
M
icode
Bch
valE
valA
dstE
dstM
e_Bch
e_valE
CC
ALU
ALU fun.
Execute
ALU A
ALU B
E
icode
ifun
valC
valA
valB
dstE
dstM
srcA
srcB
d_srcA
d_srcB
Sel+Fwd A
Fwd B
Decode
Register file
A
B
M
E
W_valM
W_valE
D
icode
ifun
rA
rB
valC
valP
Predict PC
Instruction memory
PC increment
Fetch
f_PC
Select PC
M_valA
W_valM
F
predPC

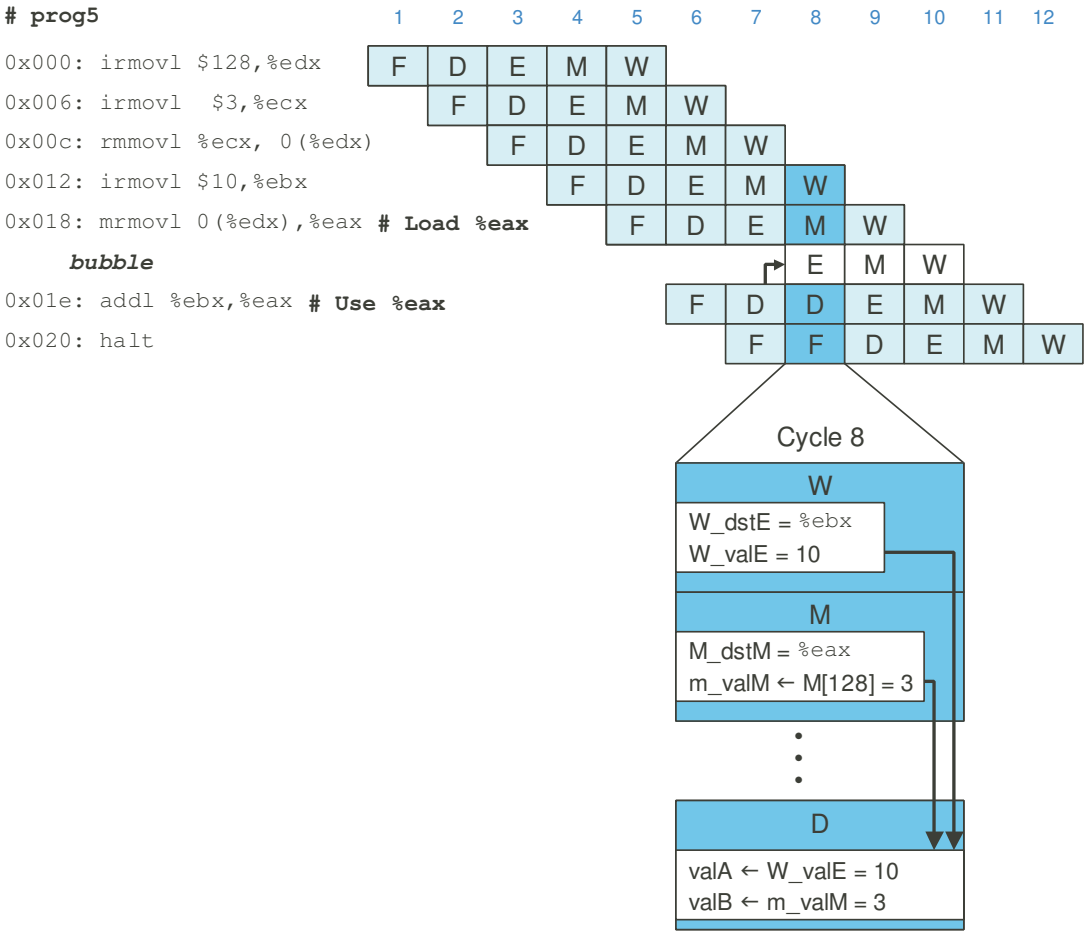
Limitation of Forwarding



Load-use dependency:

- Value needed by end of decode stage in cycle 7.
- Value read from memory in memory stage of cycle 8.

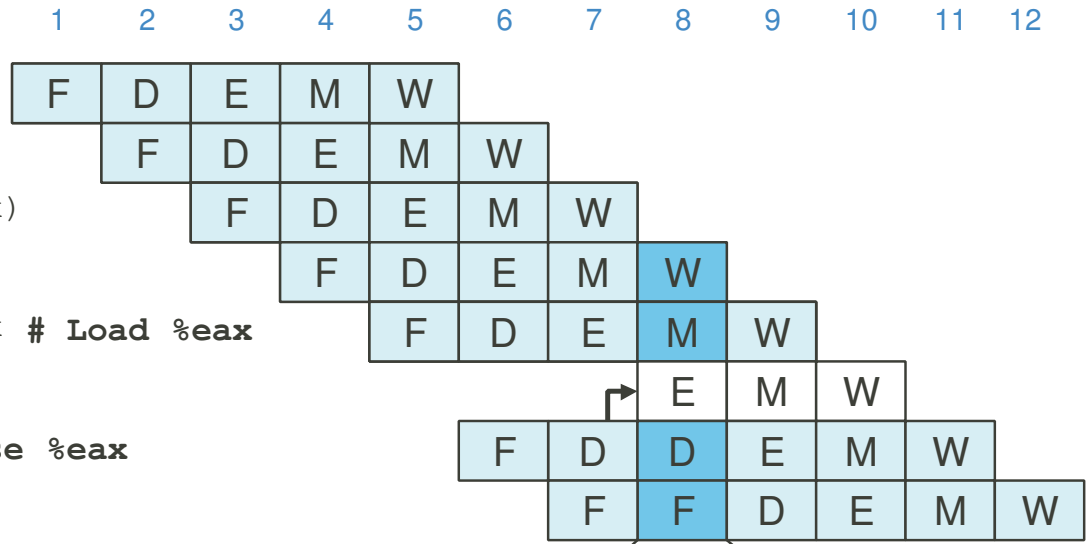
Avoiding Load/Use Hazard



- Stall using instruction for one cycle.
- Can the pickup loaded value by forwarding from memory stage.

Control for Load/Use Hazard

```
# prog5
0x000: irmovl $128,%edx
0x006: irmovl  $3,%ecx
0x00c: rmmovl %ecx, 0(%edx)
0x012: irmovl $10,%ebx
0x018: mrmovl 0(%edx),%eax # Load %eax
     bubble
0x01e: addl %ebx,%eax # Use %eax
0x020: halt
```



- Stall instructions in fetch and decode stages
- Inject bubble into execute stage.

Condition	F	D	E	M	W
Load/Use Hazard	stall	stall	bubble	normal	normal

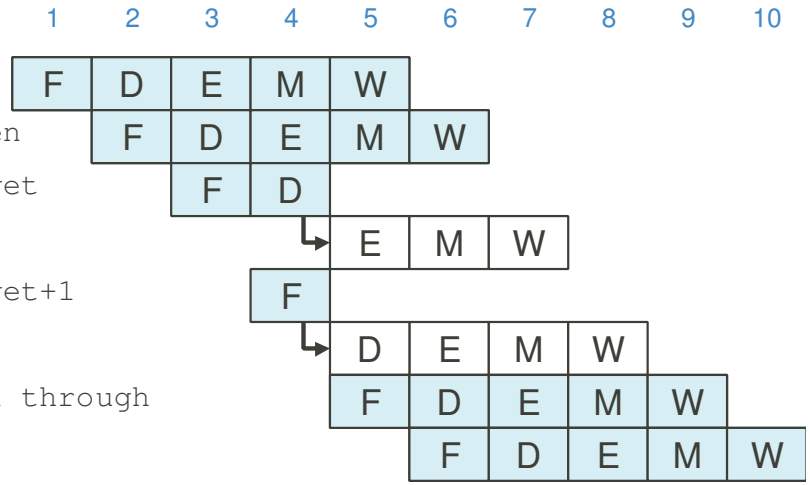
Branch Misprediction Example

```
0x000:      xorl   %eax, %eax
0x002:      jne    t               # Not taken
0x007:      irmovl $1, %eax        # Fall through
0x00d:      nop
0x00e:      nop
0x00f:      nop
0x010:      halt
0x011: t:   irmovl $2, %edx        # Target (should not execute)
0x017:      irmovl $3, %ecx        # Should not execute
0x01d:      irmovl $4, %edx        # Should not execute
```

Should only execute the first 7 instructions.

Handling Misprediction

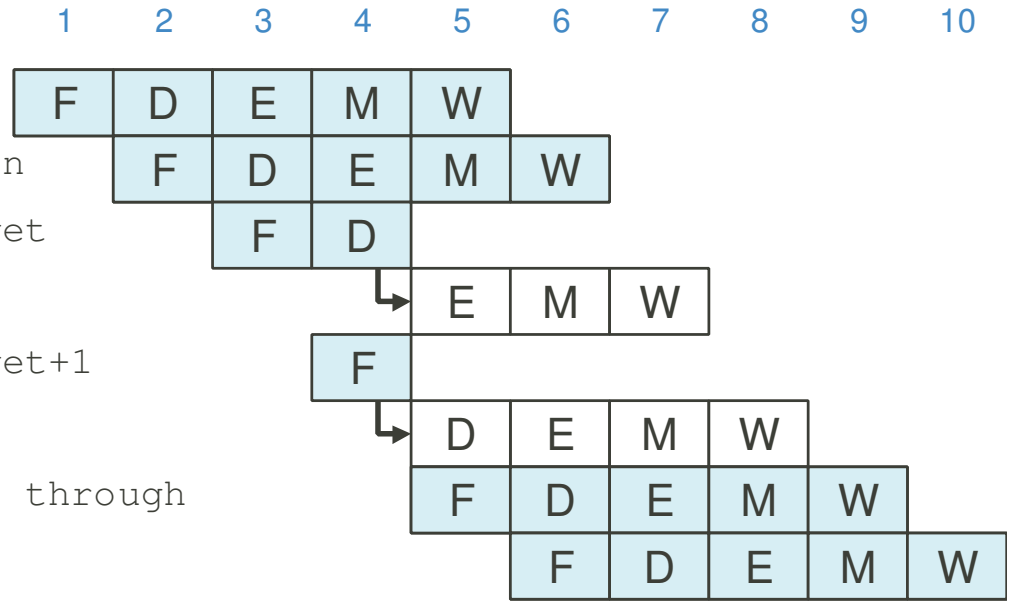
```
# prog8
0x000: xorl %eax,%eax
0x002: jne target # Not taken
0x00e: irmovl $2,%edx # Target
       bubble
0x014: irmovl $3,%ebx # Target+1
       bubble
0x007: irmovl $2,%edx # Fall through
0x00d: halt
```



- **Predict branch as taken**
  - Fetch 2 instructions at target
- **Cancel when mispredicted**
  - Detect branch not taken in execute stage
  - On following cycle, replace instruction in execute and decode stage by bubbles.
  - No side effects have occurred yet.

Control for Misprediction

```
# prog8
0x000: xorl %eax,%eax
0x002: jne target # Not taken
0x00e: irmovl $2,%edx # Target
       bubble
0x014: irmovl $3,%ebx # Target+1
       bubble
0x007: irmovl $2,%edx # Fall through
0x00d: halt
```



Condition	F	D	E	M	W
Mispredicted Branch	normal	bubble	bubble	normal	normal

Return Example

```
0x000:     irmovl  Stack, % esp  # Initialize stack pointer
0x006:     call    p             # Procedure call
0x00b:     irmovl  $5, %esi      # Return point
0x011:     halt
0x020: .pos 0x20
0x020: p: irmovl   $-1, %edi     # procedure
0x026:     ret
0x027:     irmovl  $1, %eax      # should not be executed
0x02d:     irmovl  $2, %ecx      # should not be executed
0x033:     irmovl  $3, %edx      # should not be executed
0x039:     irmovl  $4, %ebx      # should not be executed
0x100: .pos 0x100
0x100: Stack:                    # Stack pointer
```

Previously executed three additional instructions.

Correct Return Example

```
0x026:    ret
          bubble
          bubble
          bubble
0x00b:    irmovl  $5, %esi     # Return
```

- As `ret` passes through pipeline, stall at fetch stage—while in decode, execute, and memory stages.
- Inject bubble into decode stage.
- Release stall when reach write-back stage.

Control for Return

```
0x026:    ret
          bubble
          bubble
          bubble
0x00b:    irmovl  $5, %esi     # Return
```

Condition	F	D	E	M	W
Processing ret	stall	bubble	normal	normal	normal

Special Control Cases

Detection:

Condition	Trigger
Processing ret	`IRET in { D_icode, E_icode, M_icode }`
Load/Use Hazard	`E_icode in {IMRMOVL, IPOPL} &&` `E_dstM in {d_srcA, d_srcB}`
Mispredicted Branch	`E_icode == IJXX & !e_Bch`

Action (on next cycle):

Condition	F	D	E	M	W
Processing ret	stall	bubble	normal	normal	normal
Load/Use Hazard	stall	stall	bubble	normal	normal
Mispredicted Branch	normal	bubble	bubble	normal	normal