Improving System Performance: Pipelining

December 08

CSC201 Section 002

Fall, 2000
Goals

• Speed up the rate at which instructions are executed
  - faster computers!

• Does *not* speed up the time to complete one instruction
Basic Idea

• Assembly lines
  - Stations or stages
  - Progress of one unit through the assembly line
  - Simultaneous (overlapped) processing of multiple units in different stations
Basic Idea (cont)

- Result: an \( n \)-stage pipeline may produce results \( n \) times faster

- So... why not have 1000-stage pipelines?
Stages of Instruction Execution

- IF = instruction fetch and PC increment
- D = instruction decode
- OF = operand fetch
- EX = operation execution
- R = store results
Some Sample Code

- **mov** EAX, var1
- **add** EAX, var2
- **jmp** next1

**Time →**

**MOV**

IF | D | OF | EX | R

**ADD**

IF | D | OF | EX | R

**JMP**

IF | D | OF | EX | R

• Performance improvement?
Problem #1: Data Dependencies

• What if instruction i+1 uses the value produced by instruction i?

• Example

Mov EAX, var1

IF D OF EX R

add EAX, var2

IF D OF EX R

• The OF stage of instruction 2 needs the value produced by the R stage of instruction 1
Problem #1: Data Dependencies

- Solution A: detect dependencies and "stall" the pipeline
  - detection: by hardware, or by compiler

  \[
  \text{Mov EAX, var1} \\
  \text{Mov ECX, 0} \\
  \text{mov EAX, var1} \\
  \text{add EAX, var2}
  \]

- Solution B: rearrange instructions to eliminate stalls
  - again, by compiler, or in hardware. Ex.:

  \[
  \text{mov ESI, 0} \\
  \text{Mov ECX, 0} \\
  \text{mov EAX, var1} \\
  \text{add EAX, var2}
  \]

Replace:

mov ESI, 0
Mov ECX, 0
mov EAX, var1
add EAX, var2

With:

mov EAX, var1
mov ESI, 0
Mov ECX, 0
add EAX, var2
Problem #2: Flow of Control Dependencies

• When instruction i is a conditional jump, which instruction will be i+1?

• Example

```
loop1:
    cmp ECX, 10
    je loopout
    add EAX, EBX
    ...
    inc ECX
    jmp loop1
Loopout:
    mov res, EAX
```
Control Dependency Example

• Cannot start IF stage of instruction i+1 until EX stage of instruction i has finished

Je loopout

```
IF D OF EX R
```

-Or-

```
IF D OF EX R
```

add EAX, EBX
Mov res, EAX
Fixing Control Dependencies

1. Stall pipeline until conditional jump has executed (as shown)

2. *Predict* which way branch will go, fetch instruction from predicted location
   - And “undo” the instruction if the prediction turns out to be wrong

Cmp ECX, 10
Je loopout
Add eax, ebx
(1st instr. After add)
(2nd instr. After add)
Mov res, EAX
Are Branch Predictions Any Good?

- Many schemes for prediction
  - Typical: correct 70-95% of the time
- No penalty if you’re wrong; so why not predict?