ILP: CONTROL FLOW

Mahdi Nazm Bojnordi

Assistant Professor

School of Computing

University of Utah

THE UNIVERSITY OF UTAH

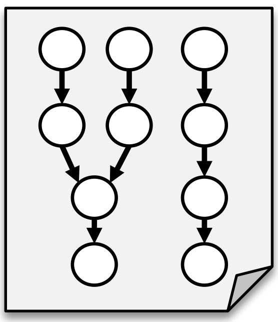
Announcement

- Homework 4
  - Will be uploaded tonight 11:59PM
  - Due date: Sept. 27th, 11:59PM

Big Picture

- **Goal:** improving performance

Software (ILP and IC)

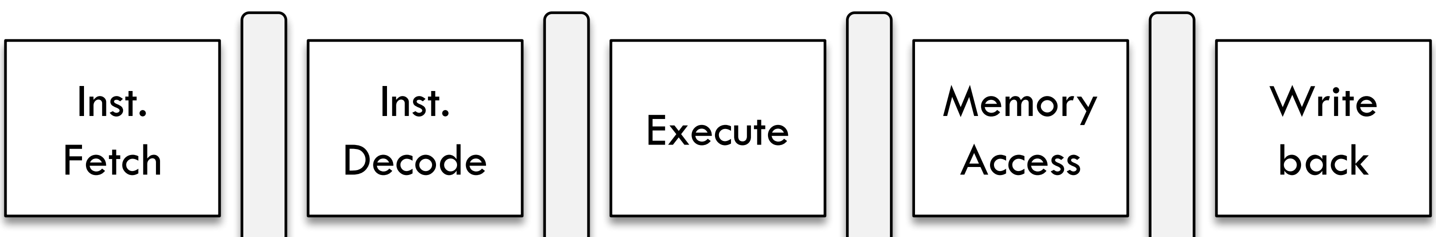


Performance = (IPC x F) / IC

Increasing IPC:
1. Improve ILP → Code gen.
2. Exploit more ILP → Architecture

Increasing F:
1. Deeper pipeline → Architecture
2. Faster technology → Circuit/Device

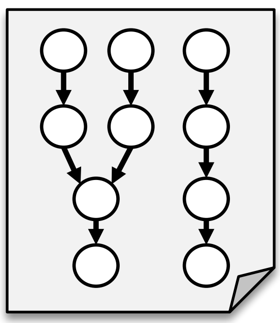
Hardware (IPC)



Big Picture

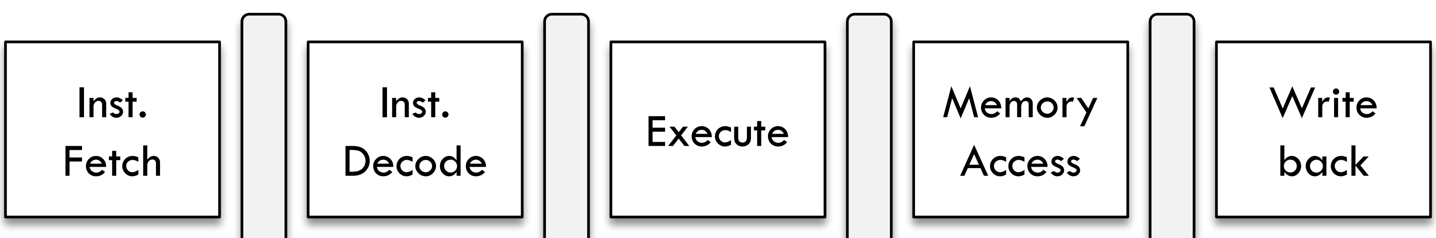
- **Goal:** improving performance

Software (ILP and IC)



Architectural Techniques:
- Deep pipelining
  - Ideal speedup = n times
- Exploiting ILP
  - Dynamic scheduling (HW)
  - Static scheduling (SW)

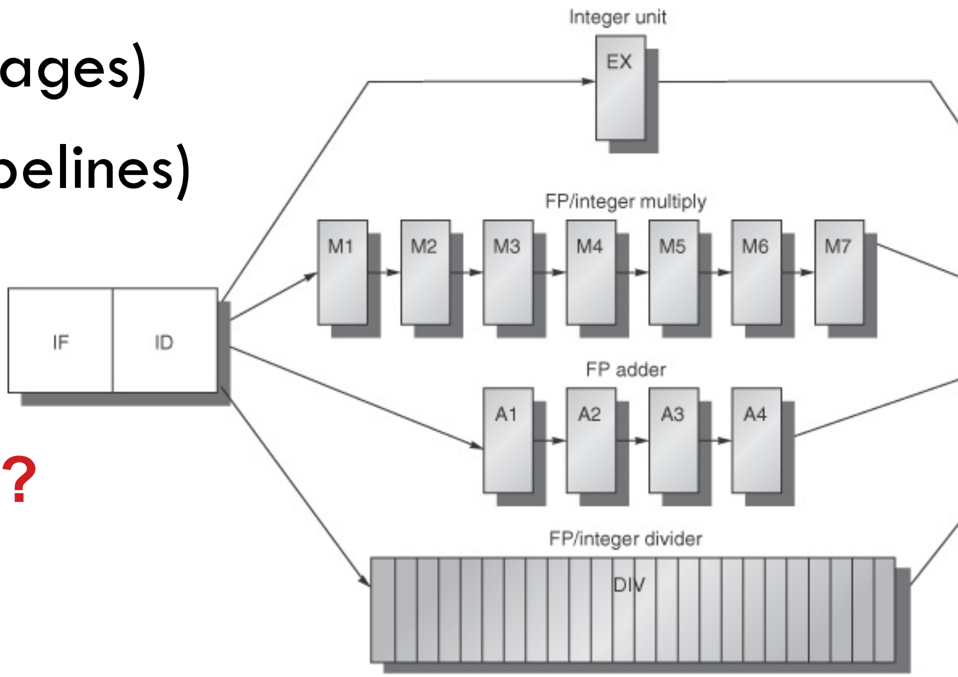
Hardware (IPC)



Recall: Performance Bottleneck

- Key performance limitation
  - Number of instructions fetched per second is limited

- How to increase fetch performance
  - Deeper fetch (multiple stages)
  - Wider fetch (multiple pipelines)



How to handle branches?

Impact of Branches

- Example C code
  - No structural/data hazards
  - What is fetch rate (IPS)?

```
do {
    sum = sum + i;
    i = i – 1;
} while(i > 0);
```

- Five-stage pipeline
  - Cycle time = 10ns

Assembly code:

```
Loop:  ADD   R1, R1, R2
       ADDI  R2, R2, #-1
       BNEQ  R2, R0, Loop
       stall
```

Impact of Branches

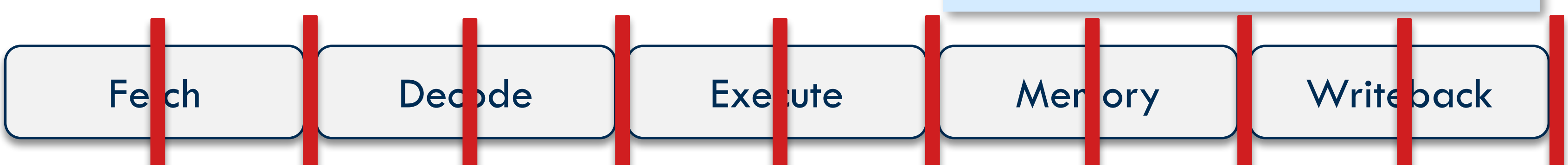
- Example C code
  - No structural/data hazards
  - What is fetch rate (IPS)?

```
do {
    sum = sum + i;
    i = i – 1;
} while(i > 0);
```

- Ten-stage pipeline
  - Cycle time = 5ns

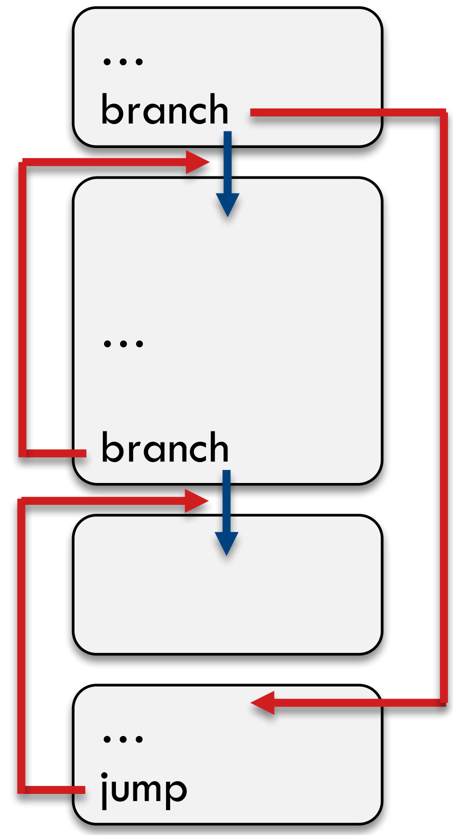
Assembly code:

```
Loop:  ADD   R1, R1, R2
       ADDI  R2, R2, #-1
       BNEQ  R2, R0, Loop
       stall
       stall
       stall
```



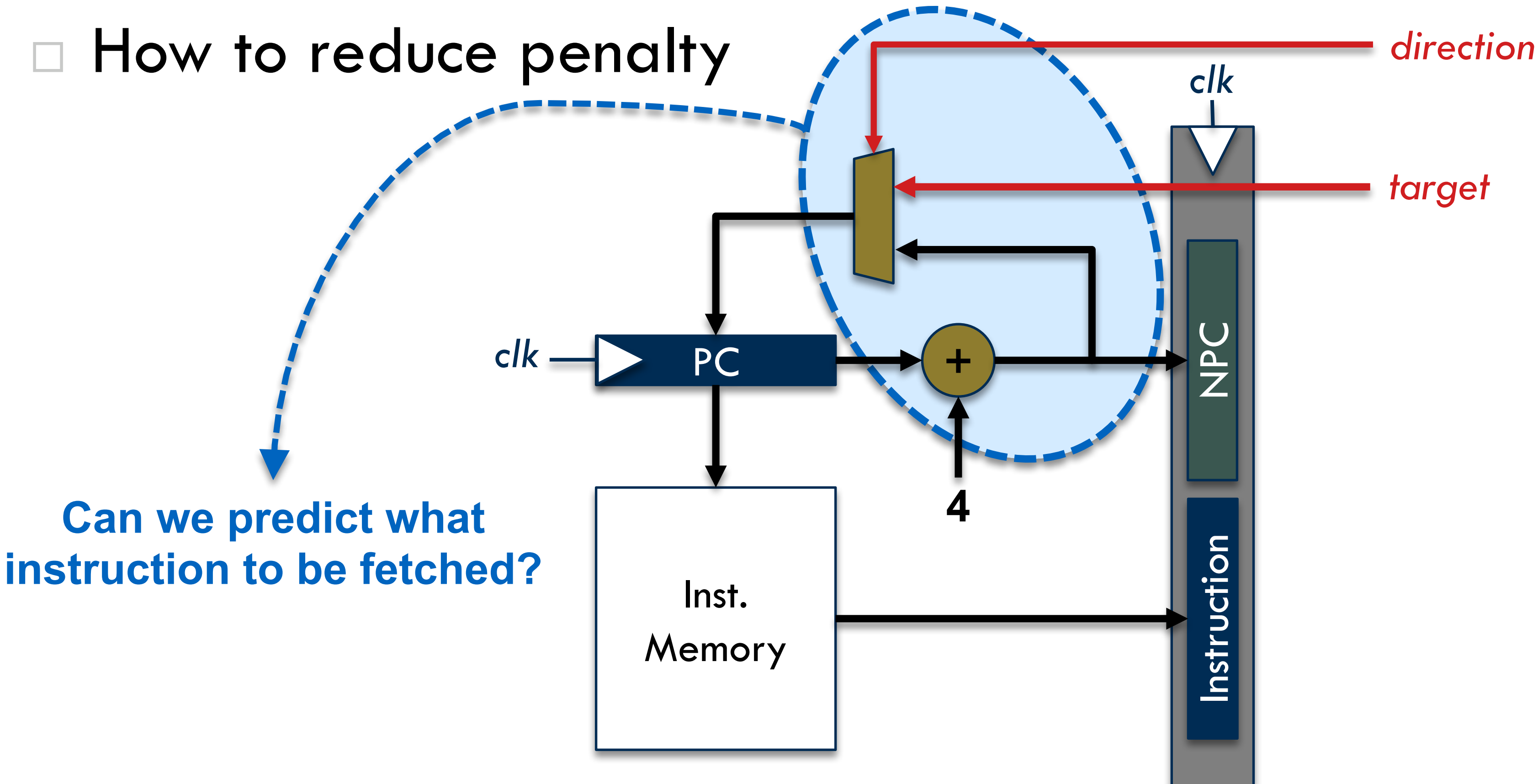
Program Flow

- A program contains basic blocks
  - Only one entry and one exit point per basic block
- Branches
  - Conditional vs. unconditional
    - How to check conditions
    - Jumps, calls, and returns
  - Target address
    - Absolute address
    - Relative to program counter



Branch Instructions

- Branch penalty due to unknown outcome
  - Direction and target
- How to reduce penalty



Branch Prediction

- How to predict the outcome of a branch
  - Profiling the entire program
  - Predict based on common cases

C/C++ code:

```
i = 10000;
do {
    r = i%4;
    if(r == 0) {
        sum = sum + i;
    }
    i = i – 1;
} while(i > 0);
```

	TAKEN	NOT-TAKEN
branch-1		
branch-2		

Branch Prediction

- How to predict the outcome of a branch
  - Profiling the entire program
  - Predict based on common cases

Assembly code:

```
      ADDI  R1, R1, #10000
do:
      ANDI  R2, R1, #3
      BNEQ  R2, R0, skp
      ADD   R3, R3, R1
skp:  ADDI  R1, R1, #-1
      BNEQ  R1, R0, do
```

	TAKEN	NOT-TAKEN
branch-1	2500	7500
branch-2	9999	1

Branch Prediction

- The goal of branch prediction
  - To avoid stall cycles in fetch stage
- Types
  - Static prediction (based on direction or profile)
    - Always not-taken
      - Target = next PC
    - Always taken
      - Target = unknown
  - Dynamic prediction
    - Special hardware using PC

Which ones are influenced

a. Performance

b. Energy

c. Power

Branch Prediction

- Prediction accuracy?
  - A: always not-taken

    0.01

  - B: always taken

    0.99

```
i = 100;
do {
    sum = sum + i;
    i = i – 1;
} while(i > 0);
```

Branch Misprediction Penalty

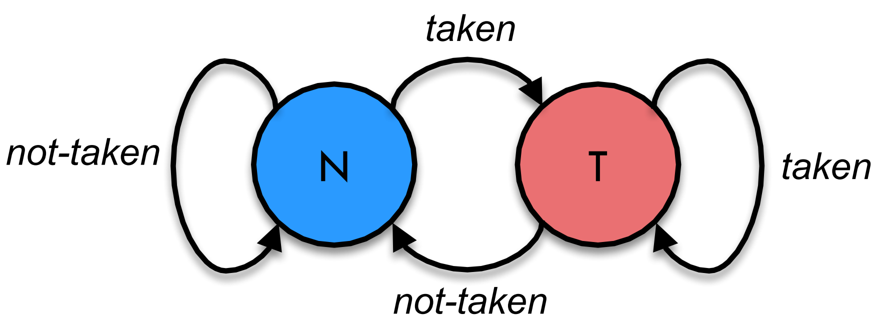
- Example: compute slowdown when there are
  - no data/structural hazards, only control hazards,
  - every 5th instruction is a branch, and
  - 90% branch prediction accuracy
- Slowdown = 1/ (1 + stalls per inst.)
- = 1/(1 + 0.2x0.1x1) = 0.98

Dynamic Branch Prediction

- Hardware unit capable of learning at runtime
  - 1. Prediction logic
    - Direction (taken or not-taken)
    - Target address (where to fetch next)
  - 2. Outcome validation and training
    - Outcome is computed regardless of prediction
  - 3. Recovery from misprediction
    - Nullify the effect of instructions on the wrong path

Simple Dynamic Predictors

- One-bit branch predictor
  - Keep track of and use the outcome of last executed branch



- Prediction accuracy
  - A single predictor shared by multiple branches
  - Two mispredictions for loops (1 entry and 1 exit)

```
while(1) {
    for(i=0; i<10; i++) {      branch-1
    }
    for(j=0; j<20; j++) {      branch-2
    }
}
```