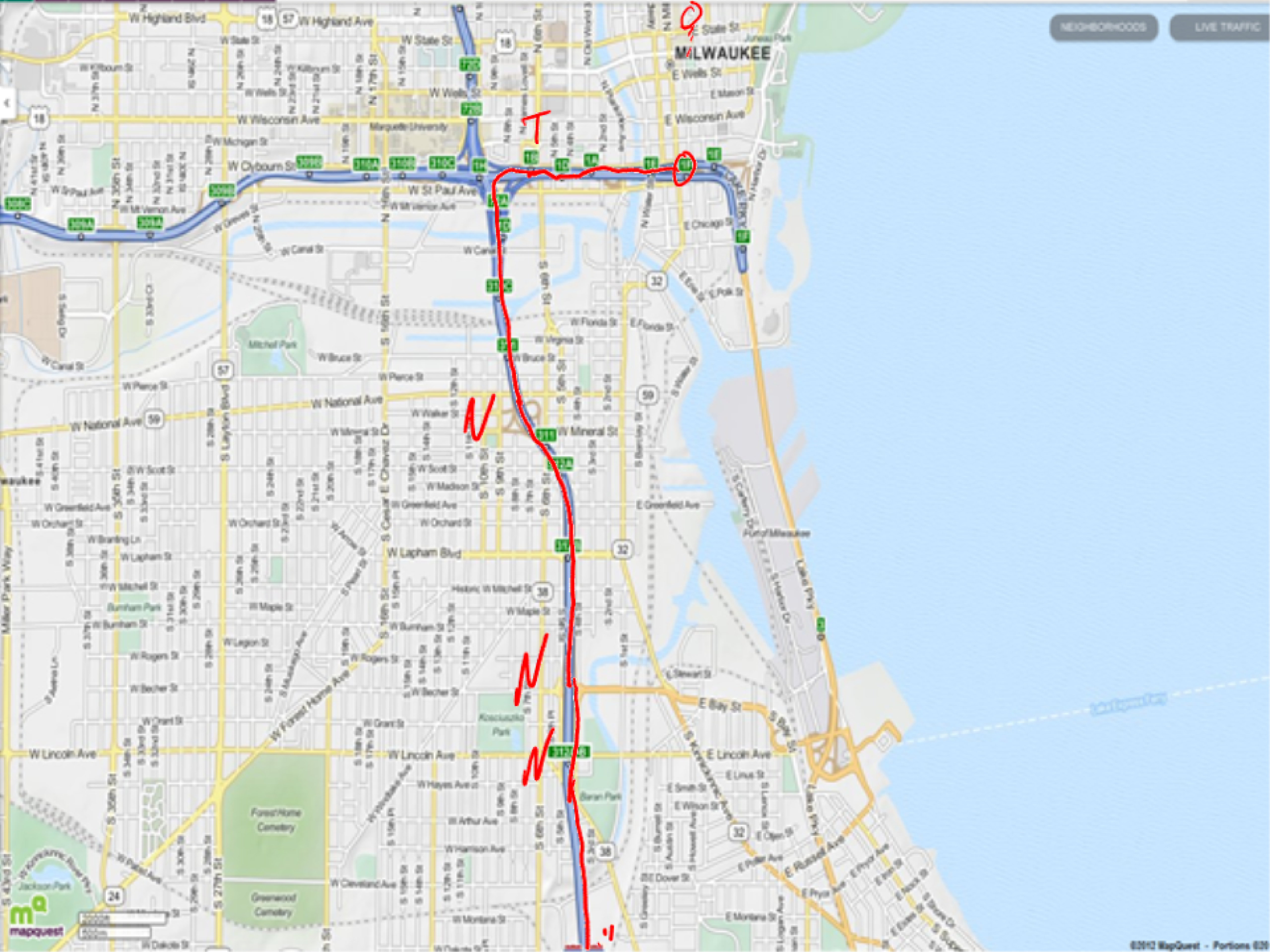
Branch Prediction

Lecture Objectives:

1) Define branch prediction.
2) Draw a state machine for a 2 bit branch prediction scheme
3) Explain the impact on the compiler of branch delay.
4) Define the concept of a vectored interrupt

Analogy

- You are driving down the freeway from Milwaukee to Chicago
- You know that you want to get off of the freeway at Ohio Street
- How do you drive there?

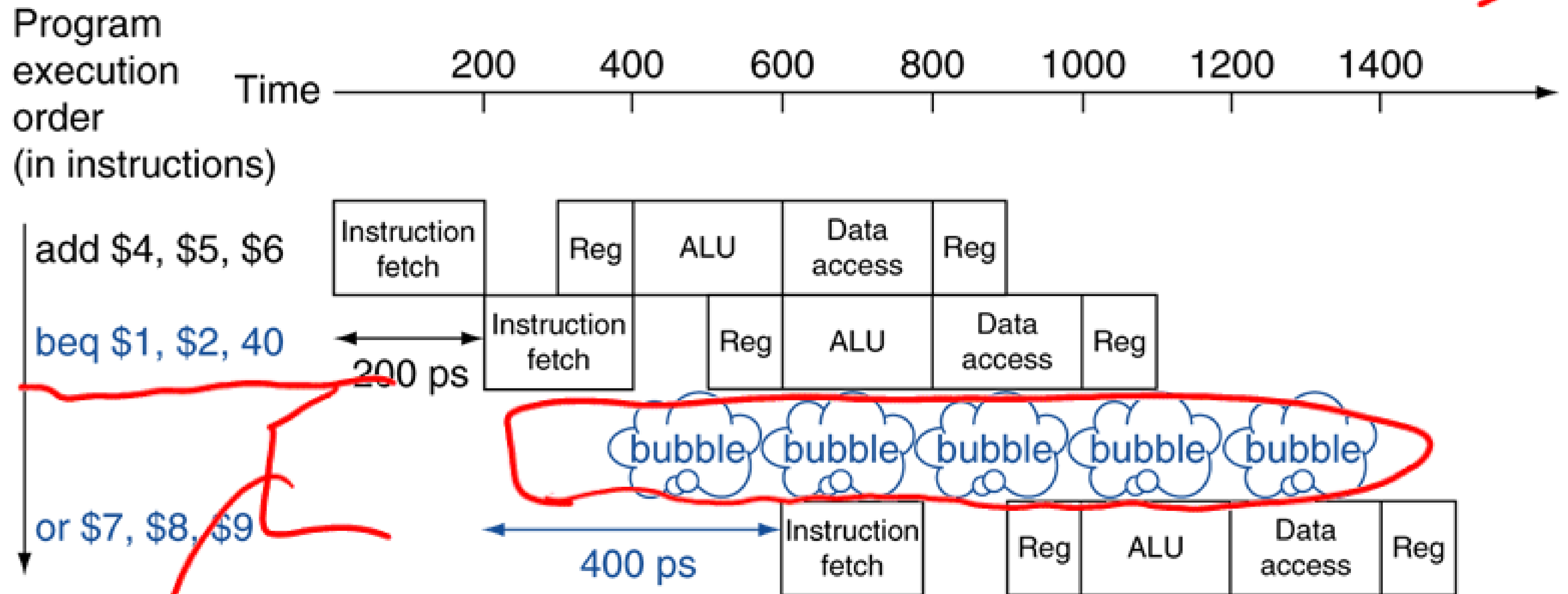

MILWAUKEE
W Highland Blvd
W State St
W Wells St
W Wisconsin Ave
Marquette University
W Clybourn St
W St Paul Ave
W National Ave
W Mineral St
W Greenfield Ave
W Lapham Blvd
W Lincoln Ave
E Bay St
E Lincoln Ave
Port of Milwaukee
Lake Pkwy
Mitchell Park
Kosciuszko Park
Baran Park
Forest Home Cemetery
Greenwood Cemetery
Jackson Park
NEIGHBORHOODS
LIVE TRAFFIC
mapquest
©2012 MapQuest

Definition

- NOP
  - An instruction that does no operation to change state.

Stalling on Branch

- Wait until branch outcome determined before fetching next instruction



Performance needs

- 17% of instructions executed in the SPECint2006 benchmark are branch instructions
  - If we always stalled for 1 clock cycle on the branch, what performance penalty would we have?

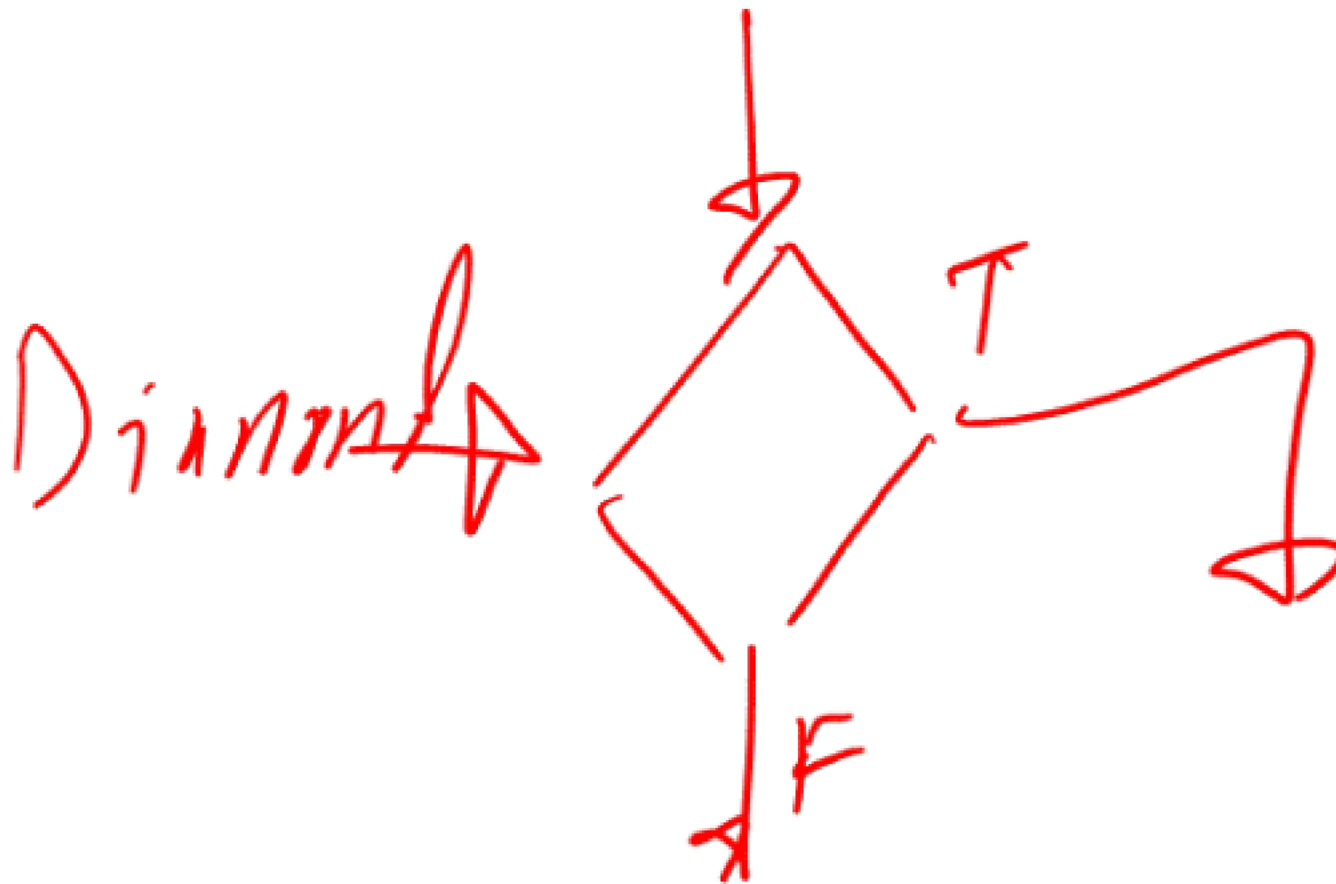
Definition

- Branch Prediction
  - A method of resolving branch hazards that assumes a given outcome for the branch and proceeds from that assumption rather than waiting to ascertain the actual outcome

(Guess!

Control Hazards

- Branch determines flow of control
  - Fetching next instruction depends on branch outcome
  - Pipeline can’t always fetch correct instruction
    - Still working on ID stage of branch



MSOE

Solution #1

- Assume all branches are always taken
  - The processor assumes you will always take the branch
  - If for some reason you should not take the branch, start over again.

↳ Go back from ~~then~~ where we came.

MSOE

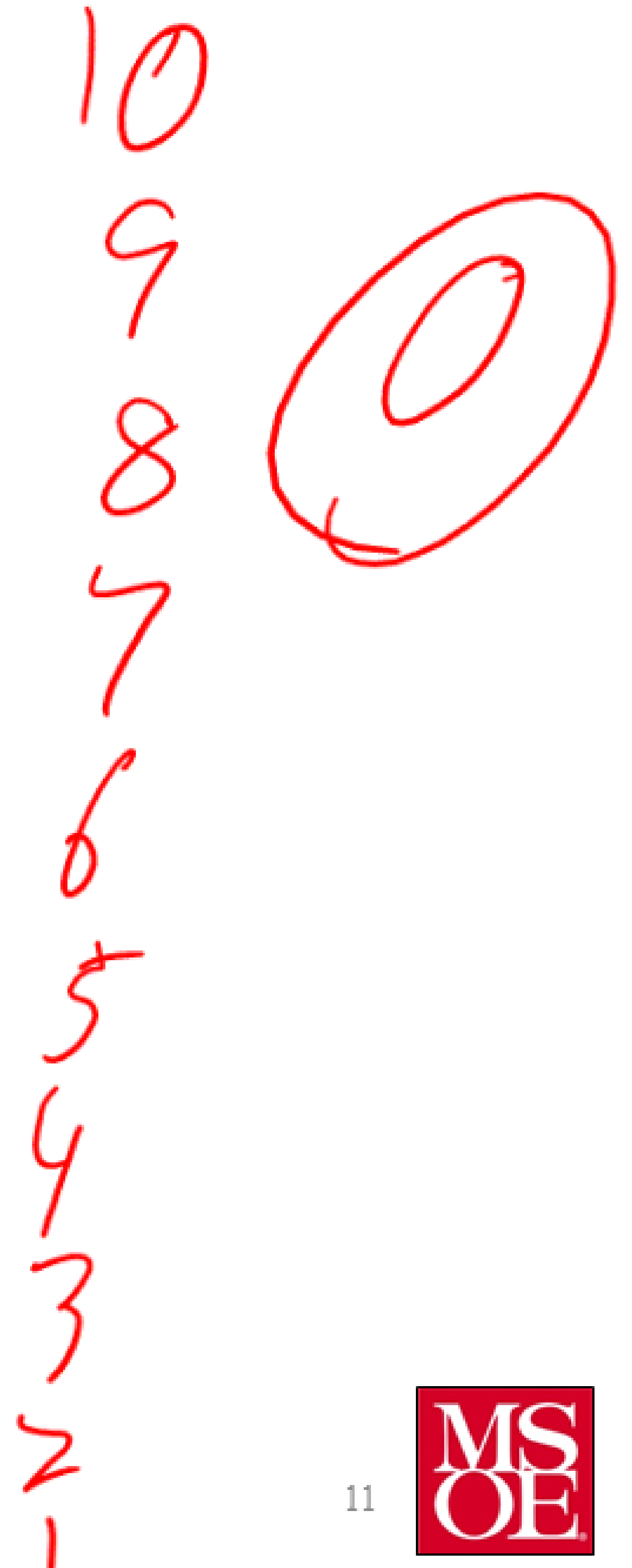
Example assembly code (do – while loop)

```
.text
main:
    li $t0, 10

loop:
    addi $t0, $t0, -1
    add  $t0, $t0, $zero
    bnez $t0, loop

    # Goto main
    j main
```

Example 2 Assembly Code (while loop)

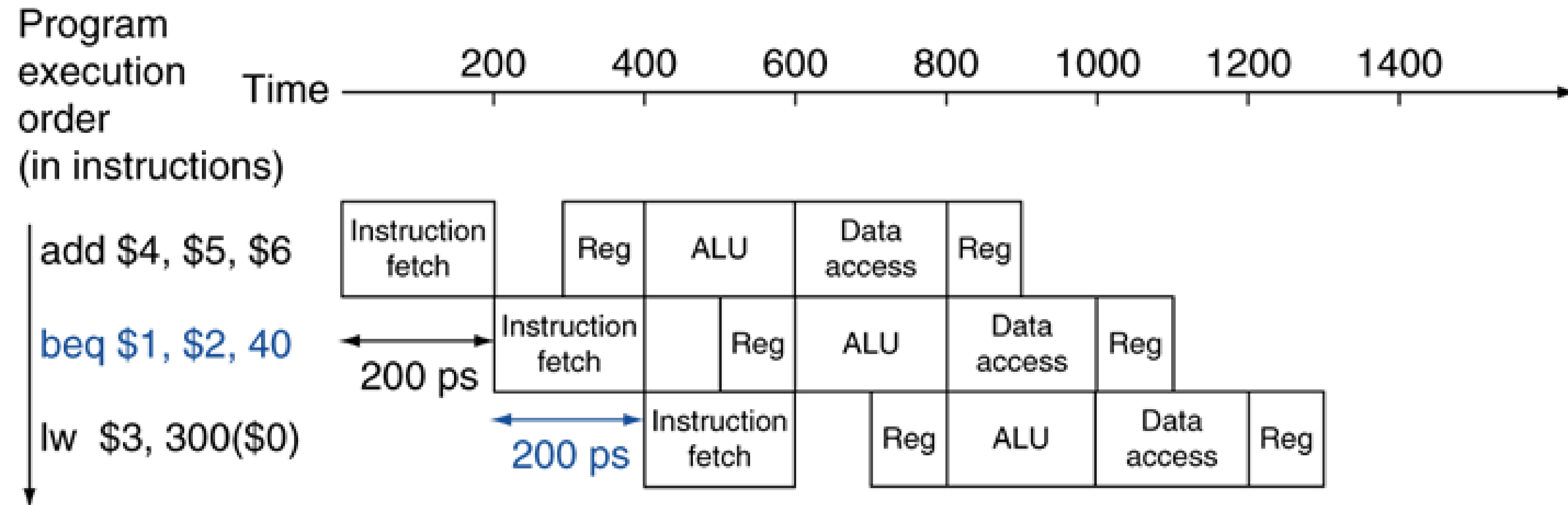
```
.text
main:
    li $t0, 10

loop:   beqz $t0, exitLoop
    addi $t0, $t0, -1
    add $t0, $t0, $zero
    j loop

exitLoop:
    # Goto main
    j main
```

Wrong 10x
Right once

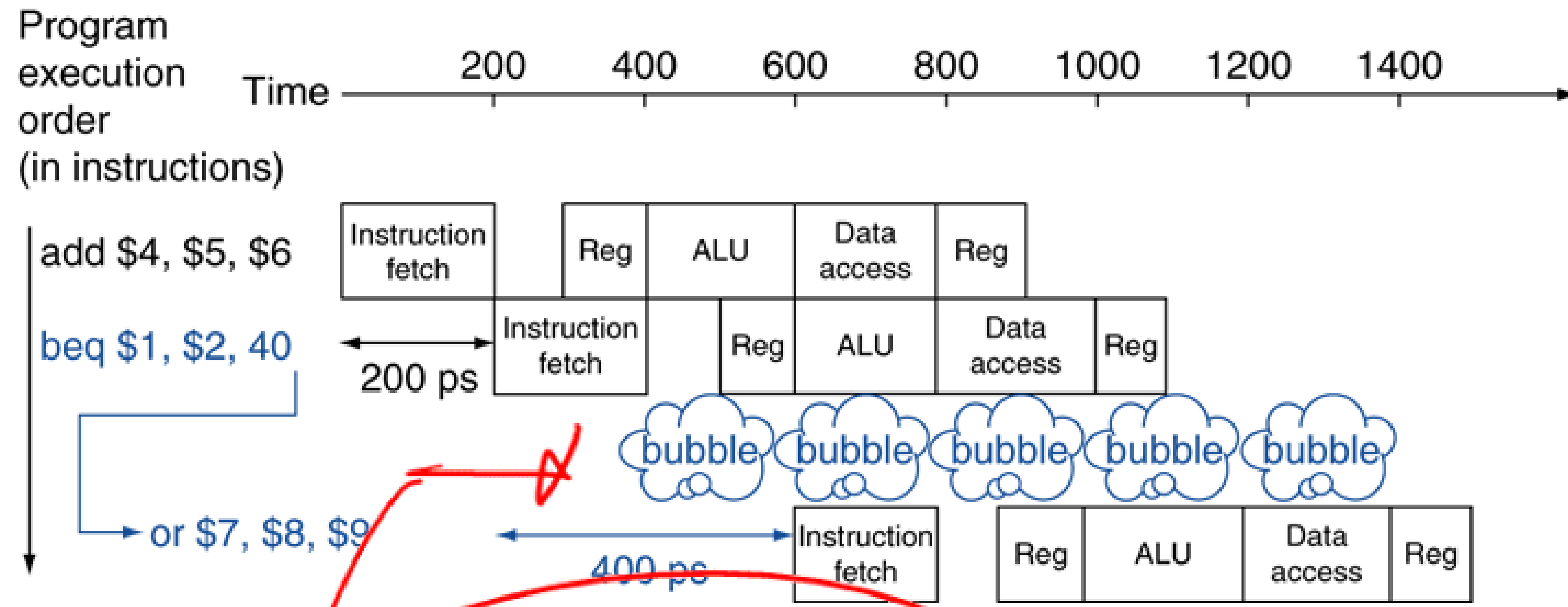


Predict Not Taken

Prediction correct



Prediction incorrect



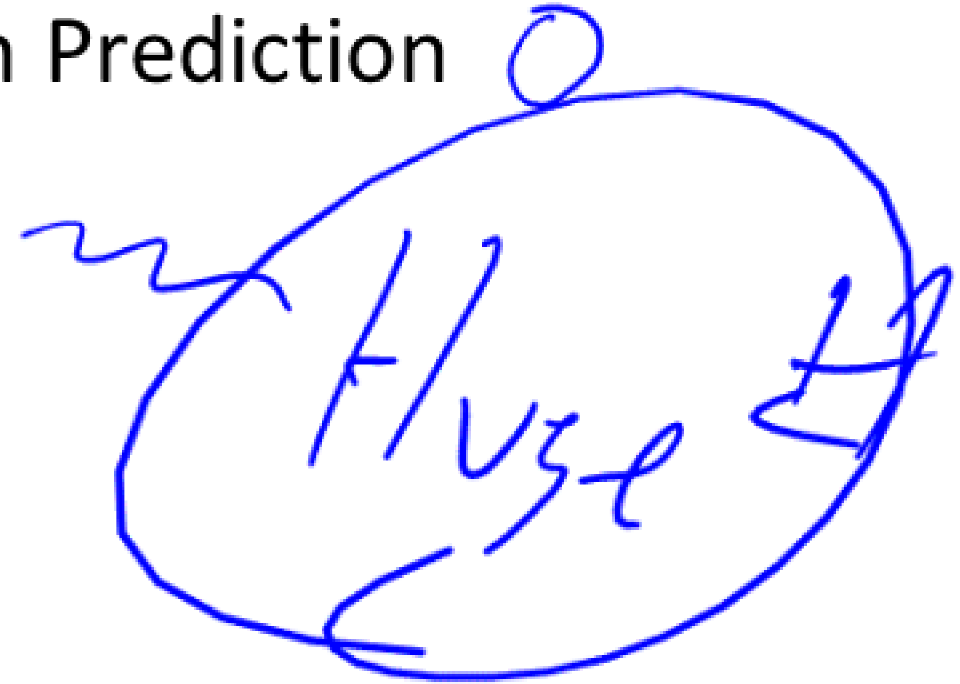
Dynamic Versus static branch prediction

- Static branch prediction
  - Based on typical branch behavior
  - Example: loop and if-statement branches
    - Predict backward branches taken
    - Predict forward branches not taken
- Dynamic branch prediction
  - Hardware measures actual branch behavior
    - e.g., record recent history of each branch
  - Assume future behavior will continue the trend
    - When wrong, stall while re-fetching, and update history

MSOE

Survey

- The branch prediction methods we just discussed were examples of
  A. Static Branch Prediction
  B. Dynamic Branch Prediction
  C. I haven't a clue



Dynamic Branch Prediction

- Branch prediction buffer (aka branch history table)
- Indexed by recent branch instruction addresses
- Stores outcome (taken/not taken)
- To execute a branch
  - Check table, expect the same outcome
  - Start fetching from fall-through or target
  - If wrong, flush pipeline and flip prediction

Loops and Branch Prediction

- Here is a loop of code
  - Which branches should we reliably predict
  - Which ones can we reasonably predict incorrectly?
  - What percentage of branches should we get correct?

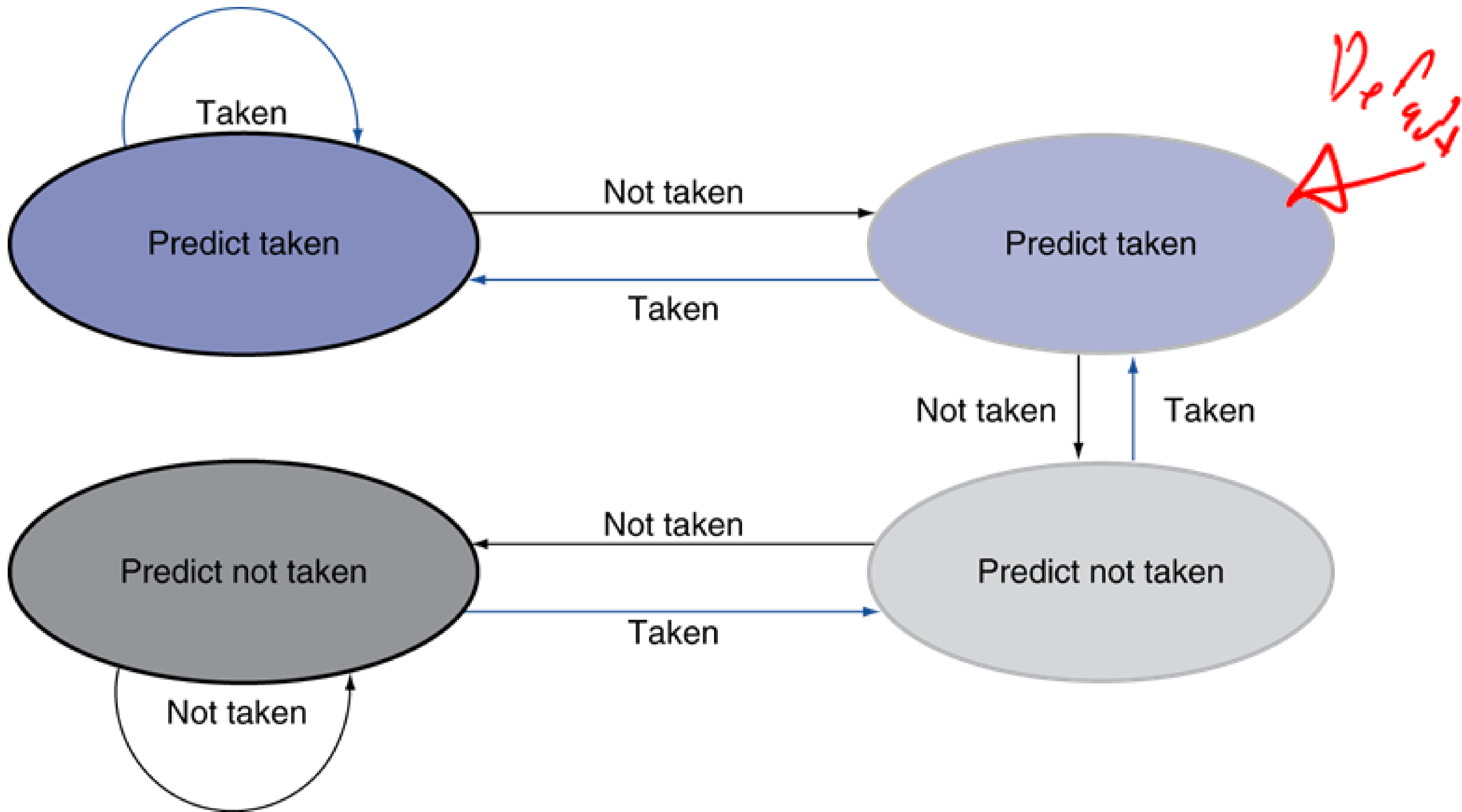
```
.text
main:
    li $t0, 100
loop:
    addi $t0, $t0, -1
    add $t0, $t0, $zero
    bnez $t0, loop
    # Goto main
    j main
```

Loops and Branch Prediction

- Here is a loop of code
  - Which branches should we reliably predict
  - Which ones can we reasonably predict incorrectly?
  - What percentage of branches should we get correct?

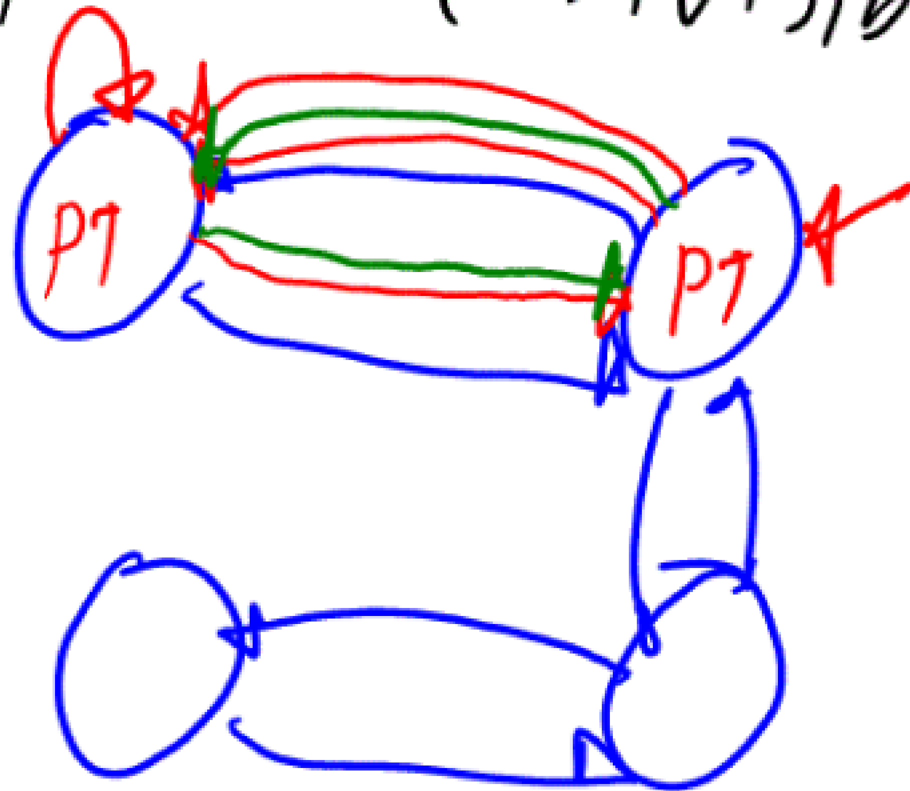
```
.text
main:
    li $t0, 100
loop:
    addi $t0, $t0, -1
    add $t0, $t0, $zero
    bnez $t0, loop
    # Goto main
    j main
```

2 bit branch prediction state machine



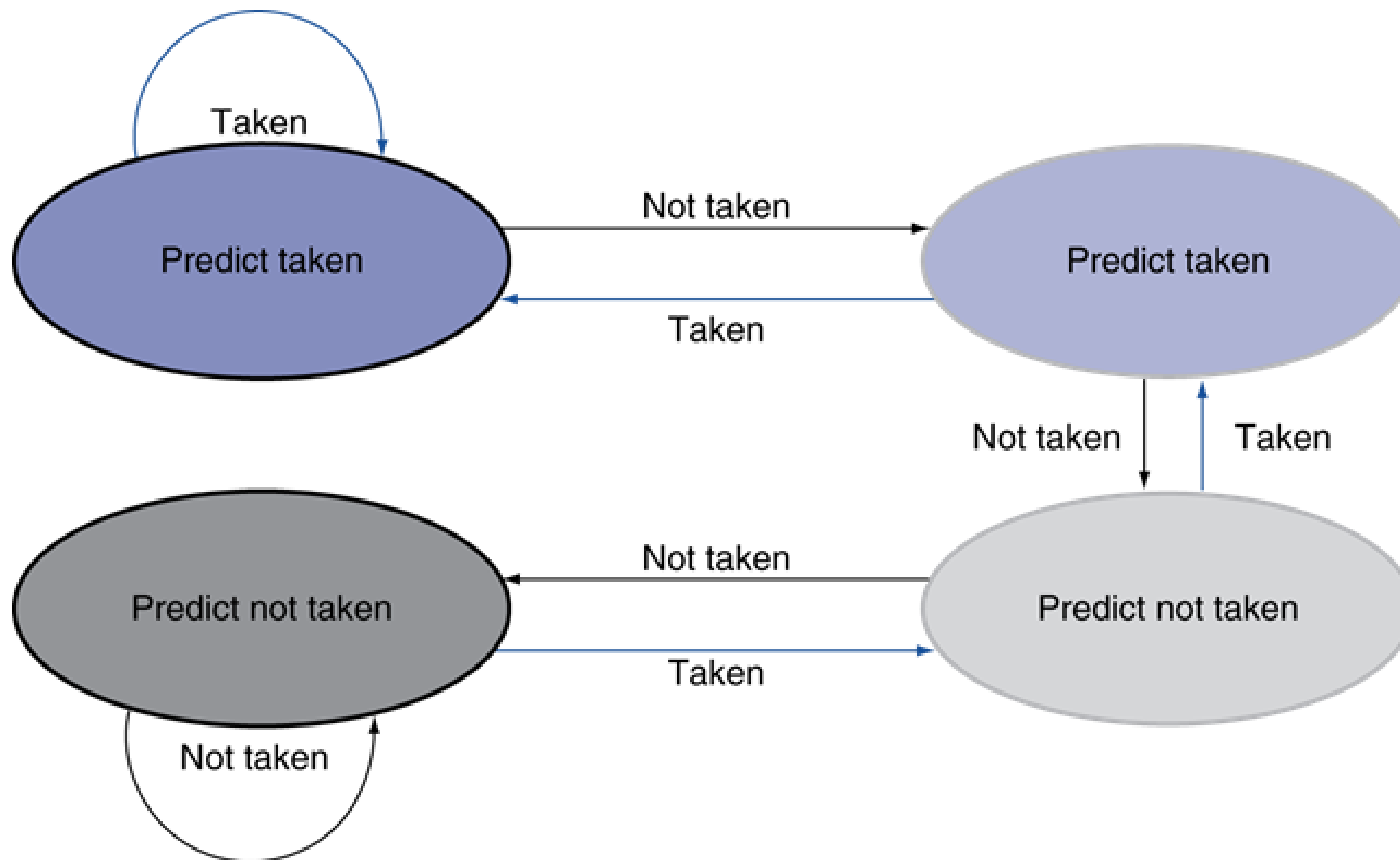
```
For ( x = 0; x < 1000; x++ )
{
    if ( x % 5 != 0 )
    { printf ("Not Divisible by 5");
}
```

Kondi



2-Bit Predictor

- Only change prediction on two successive mispredictions



MSOE

Exceptions and Interrupts

- Unexpected” events requiring change in flow of control
  - Different ISAs use the terms differently
- Exception
  - Arises within the CPU
    - e.g., undefined opcode, overflow, syscall, …
- Interrupt
  - From an external I/O controller
- Dealing with them without sacrificing performance is hard

MSOE

Handling Exceptions

- In MIPS, exceptions managed by a System Control Coprocessor (CP0)
- Save PC of offending (or interrupted) instruction
  - In MIPS: Exception Program Counter (EPC)
- Save indication of the problem
  - In MIPS: Cause register
  - We'll assume 1-bit
    - 0 for undefined opcode, 1 for overflow
- Jump to handler at 8000 00180

MSOE

An Alternate Mechanism

- Vectored Interrupts
  - Handler address determined by the cause
- Example:
  - Undefined opcode: C000 0000
  - Overflow: C000 0020
  - …: C000 0040
- Instructions either
  - Deal with the interrupt, or
  - Jump to real handler

MSOE