Shortcoming of 1-bit branch predictors

- Remember the last behavior of the branch

```c
for (i=0; i< 100; i++) {
    A[i] = B[i] * C[i];
    D[i] = E[i] / F[i];
}
```

```
this is a conditional branch
```

- How many prediction hits and misses? Prediction accuracy?

```c
for (i=0; i< 100; i++) {
    for (j=0; j< 5; j++) {
        A[i] = B[i] * C[i];
        D[i] = E[i] / F[i];
    }
}
```

```
this branch is predicted wrong twice every inner loop invocation (every 5 branches)
```

2-bit predictor

- Requires two consecutive miss-predictions to flip direction

```
Predicted | T | T | T | NT | T | T | T | NT | T | T
Actual | T | T | T | NT | T | T | T | NT | T | T
```

Correct prediction
Miss-predicted once
Change prediction
Miss-predicted once
Correct prediction
2-Bit Branch prediction buffer

- A 2-bit branch prediction buffer keeps the state of the predictor.
- The PC (Program Counter) is used to index into the buffer.
- The N bits determine whether the branch is taken or not taken.

Delayed branching

- Another approach to avoid control hazards.
- Change the branch semantics such that the "N" instructions after a branch are always executed before branching takes place.
  - The N instructions are executed regardless of the branch outcome.
  - N is set to be the number of cycles needed to resolve the branch.

- The compiler will try to fill the N slots with useful work.
  - If can't find instructions to fill the slots, use NOPs.

- Simplifies hardware (eliminates hazards by changing the semantics).
- Possibly good performance – if slots are filled with useful instructions.
- Code size will increase??
Finding instructions to fill the delay slots

Handling exceptions (Section 4.9)

- Exceptions (traps or interrupts) are an important part of a processor
- Modern OS and debuggers depend on hardware support for exceptions
- Interrupt mechanism allows efficient use of CPU cycles by allowing the CPU to do useful work while waiting for an event.

- Precise exception
  - Execute all instructions before the faulting (trapped/interrupted) instruction
  - Do not execute the faulting instruction and the following ones
  - Resume from the faulting instruction after the exception is taken care of

- How does MIPS handle exceptions?
  - Record the PC of the next instruction in a special place (the EPC register)
  - Record the cause of the exception (the Cause register)
  - Squash the faulting instruction and the instructions following it
  - Jump to a pre-determined handling routine (PC = 8000 00180)
  - When done, the handling routine will execute a "return from interrupt" to resume from the instruction following the faulting instruction (EPC → PC), or the faulting instruction (EPC-4 → PC) – depending on implementation.
The Handler

- Typically, the exception handler will determine the action(s) to be taken if consequences are not fatal
  - take corrective action
  - use EPC to return to program
- Otherwise
  - Report error using EPC, cause, ...
  - Terminate program

- Handler jumps to an address which is determined by the cause of the exception
  - Addresses are stored in a vector which is indexed by the cause
  - Example:
    
    | Undefined opcode: | C000 0000 |
    | Overflow:          | C000 0020 |
    | ...               | C000 0040 |

For external interrupts: the handler address is stored in a vector which is indexed by the interrupt type. The PC+4 is saved in EPC and the handler’s address replaces the current PC.