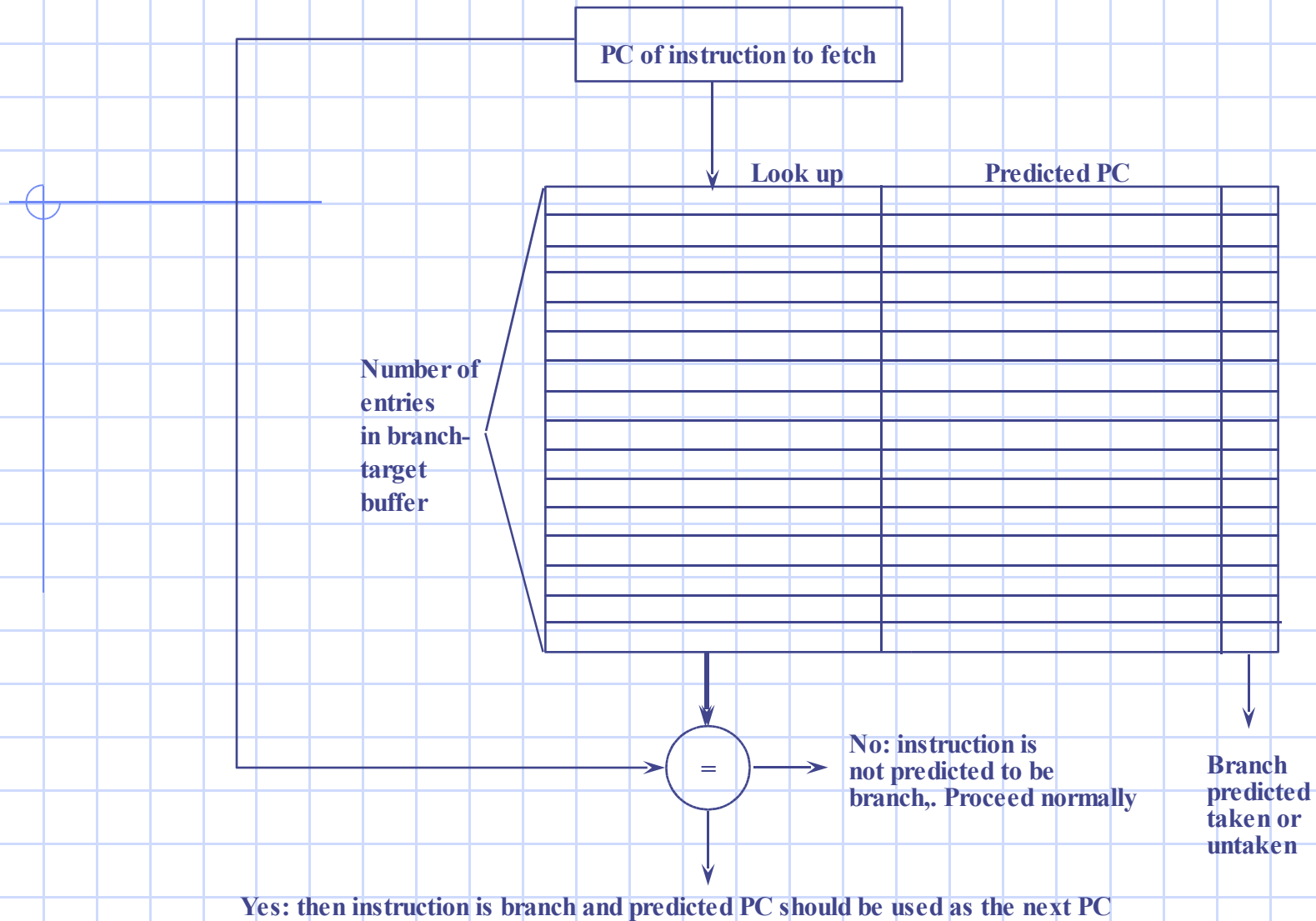


Further Reducing Branch Stalls - Branch-Target Buffer (BTB)

- Consider DLX: how can we know the branch target address earlier at the **IF** stage?
- BTB: a branch-prediction cache that store the predicted address for the next inst after a branch
- Attached to **IF** stage: so if we get a hit, we know the target address at the end of **IF** cycle
- We only need to store taken branches since untaken branch is treated as non-branches.



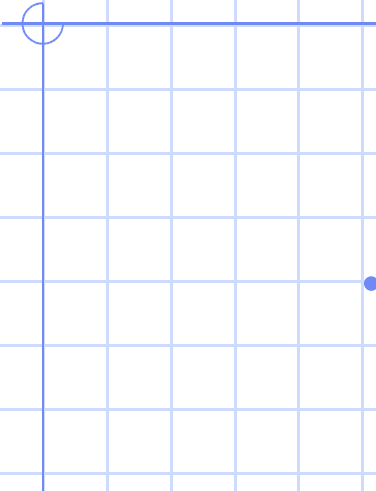
A Branch-Target Buffer

Note:

The PC of the instruction being fetched is matched against a set of instruction addresses stored in the first column; these represent the addresses of known branches. If the PC matches one of these entries, then the instruction being fetched is a taken branch, and the second field, predicted PC, contains the prediction for the next PC after the branch. Fetching begins immediately at that address. The third field, which is optional, may be used for extra prediction state bits.

Branch Target Buffer (BTB)

- A buffer contains the branch target address, as well as the information if it was taken/not-taken
- The buffer is fetches at the same time the instruction I is fetched.
 - so we don't even know I is a branch
- But, if hit occurs, we know this is a branch, and we proceed to process I, if the exec result (taken/not-taken?) matches with the cache entry prediction, we loss no cycle.
- Otherwise, we need to pay a penalty - usually 2 cycles.

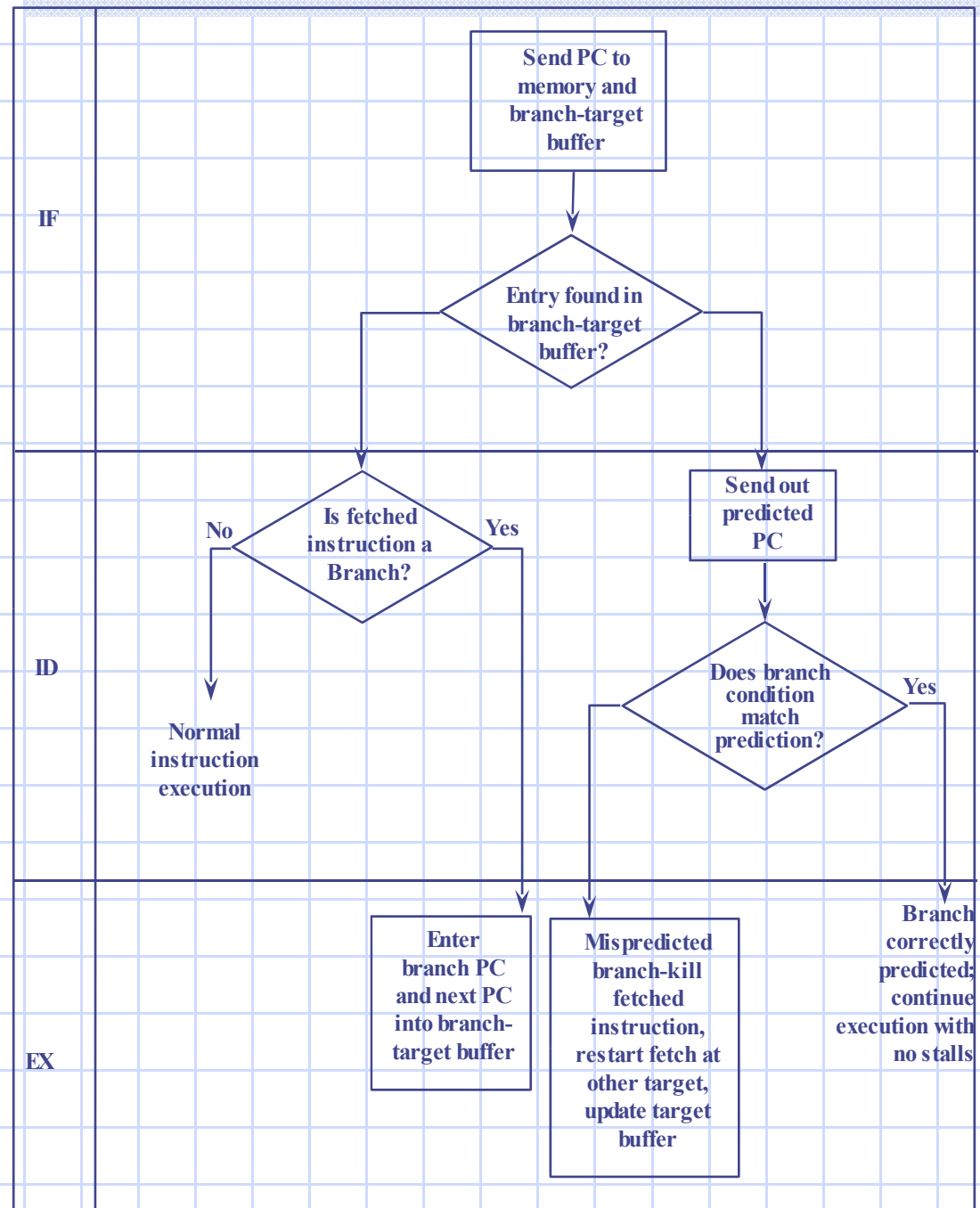
- 
- But, if miss, then we will have to do normal instruction processing, and a cache entry needs to be created.

Note that:

complication when a 2-bit scheme is used: we need to store information on non-taken branches somewhere.

Solution: perhaps a separate BPB in addition to BTB.

The steps involved in handling an instruction with a branch-target buffer.



Note:

If the PC of an instruction is found in the buffer, then the instruction must be a branch, and fetching immediately begins from the predicted PC in ID. If the entry is not found and it subsequently turns out to be a branch, it is entered in the buffer along with the target, which is known at the end of ID. If the instruction is a branch, is found, and is correctly predicted, then execution proceeds with no delays. If the prediction is incorrect, we suffer a one-clock-cycle delay fetching the wrong instruction and restart the fetch one clock cycle later. If the branch is not found in the buffer and the instruction turns out to be a branch, we will have proceeded as if the instruction were a branch and can turn this into an assume-not-taken strategy; the penalty will differ depending on whether the branch is actually taken or not.

Evolution of BTB

- IBM 360/91, 360/195...
 - small 2 entry BTB with actual branch target instructions
 - later: DTH: decode time history
- MU5 (1980)

BTB contain target addresses (called “Jum trace”) and later named “BTB”
- BHC:

combine: small buffer of instruction and DTH to store branch instructions used in AMD 29000

Speculative Execution

- Basic concepts
- Reading suggestions