# HY425
# Αρχιτεκτονική Υπολογιστών

## Branch Prediction

Ιάκωβος Μαυροειδής

# Branch Prediction

- Πρόβλεψη είναι πλέον απαραίτητη για να έχουμε καλή απόδοση. Γιατί;

```
MULT F0,F1,F2
DIVD F4,F0,F3
BNEZ F4,Loop
```

- Σημαντική σε out-of-order ή multi-issue επεξεργαστές (Amdahl's Law)

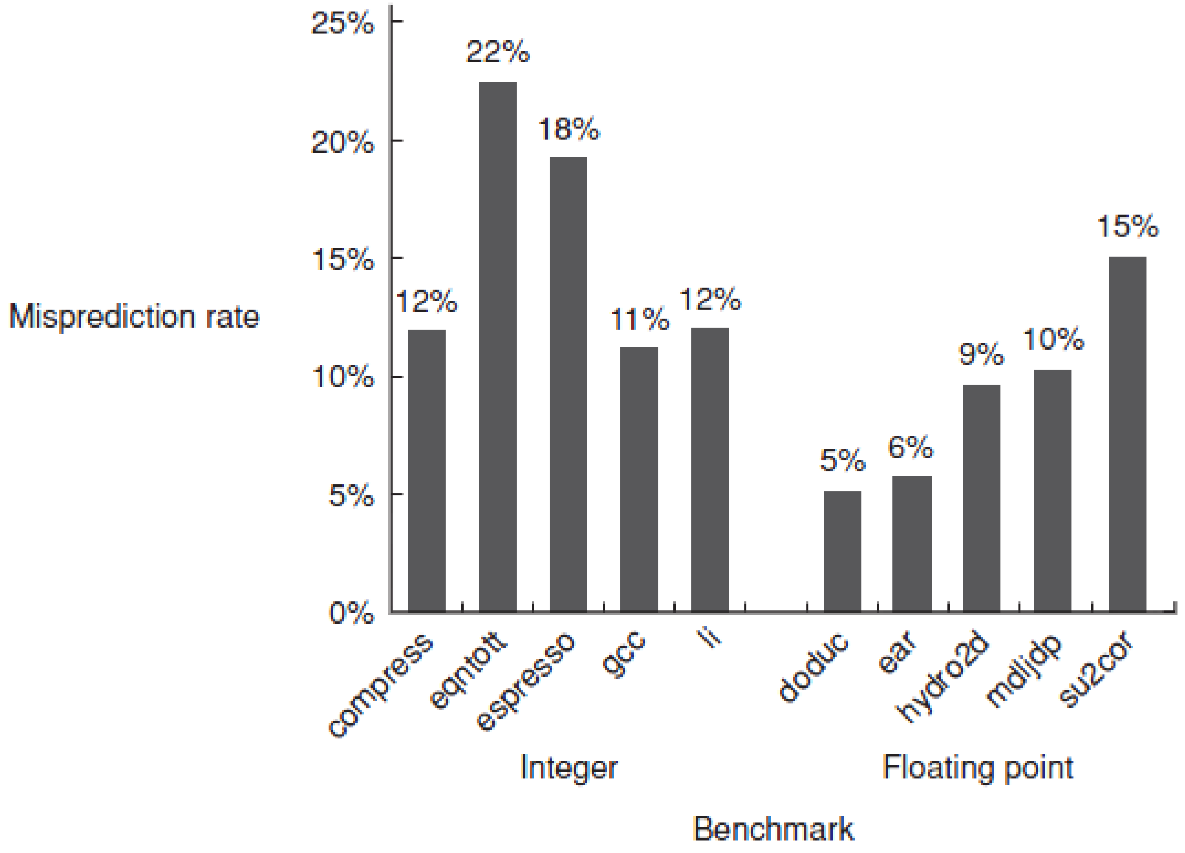
  CPI = Ideal CPI + Structural stalls +
  RAW stalls + WAR stalls + WAW stalls + Control stalls

- The prediction is a hint that is assumed to be correct, and fetching begins in the predicted direction. If the hint turns out to be wrong, the executed (not committed) instructions from the wrong path are cancelled.

# Πρόβλεψη αποτελέσματος

- Γιατί δουλεύει η πρόβλεψη?
  - Underlying algorithm has regularities.
  - Data that is being operated on has regularities.
  - Instruction sequence has redundancies that are artifacts of way that humans/compilers think about problems.
  - Loops are easy to predict their behaviour

# Static Branch Prediction



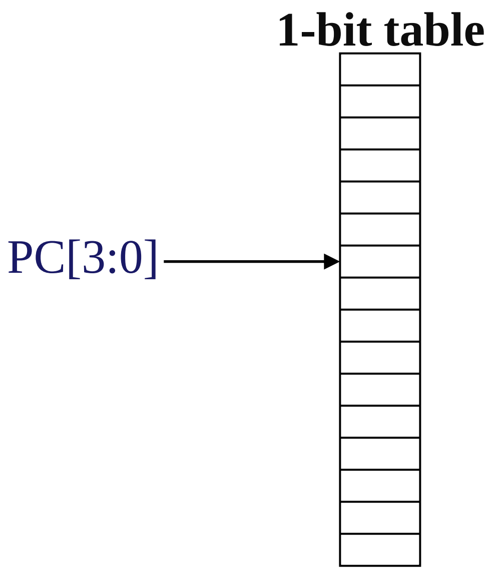
Major limitation : misprediction rate for the integer programs is high

# Dynamic Branch Prediction

- Είναι η δυναμική πρόβλεψη καλύτερη από την στατική;
  - Έτσι φαίνεται.
  - Josh Fisher had good paper on “Predicting Conditional Branch Directions from Previous Runs of a Program.”
    ASPLOS ‘92. In general, good results if allowed to run program for lots of data sets.
    - How would this information be stored for later use?
    - Still some difference between best possible static prediction (using a run to predict itself) and weighted average over many different data sets

# Simplest Dynamic Approach: Branch History Table

- Performance = *f*(accuracy, cost of misprediction)
- ***Branch History Table***: Lower bits of PC index 1-bit table
  - Specifies if branch was taken or not the last time
  - When branch delay is longer than time to compute target PC



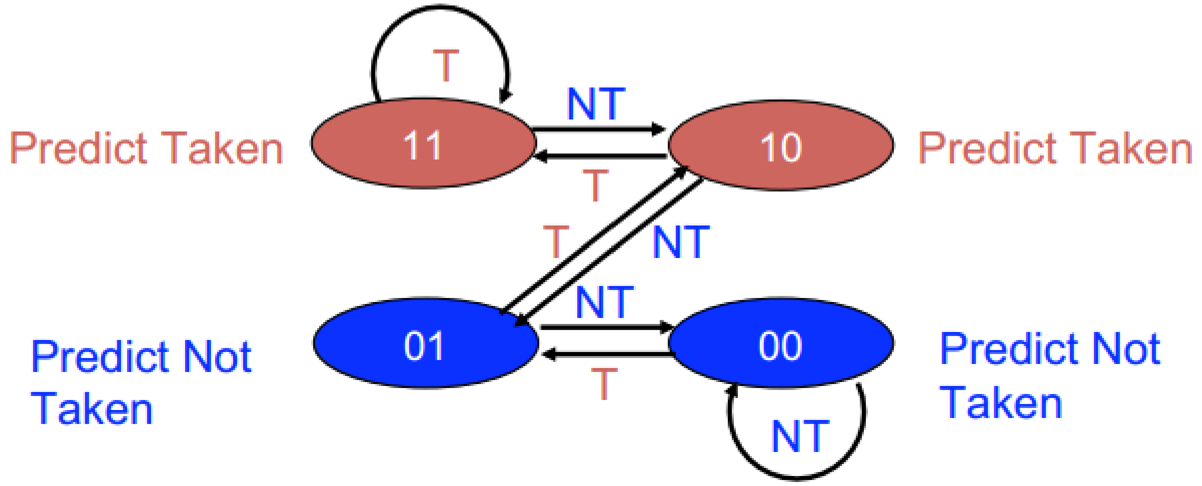
# BHT: Πρόβλημα

- ***Πρόβλημα***☞ σε loop, 1-bit BHT προκαλεί 2 mispredictions
  - End of loop case, when it exits instead of looping as before
  - First time through loop on next time through code, when it predicts exit instead of looping

```
Loop: LD   F0,0(R1) ;F0=vector element
      ADDD F4,F0,F2 ;add scalar from F2
      SD   0(R1),F4 ;store result
      SUBI R1,R1,8  ;decrement pointer 8B (DW)
      BNEZ R1,Loop  ;branch R1!=zero
      NOP           ;delayed branch slot
```

# 2-bit Prediction scheme (Jim Smith, 1981)

- ***Λύση*** ☞ 2-bit scheme αλλάζει πρόβλεψη μόνο αν γίνει λάθος πρόβλεψη 2 φορές:



- Κόκκινο: not taken
- Μπλε: taken
- Προσθέτει *hysteresis* στην πρόβλεψη

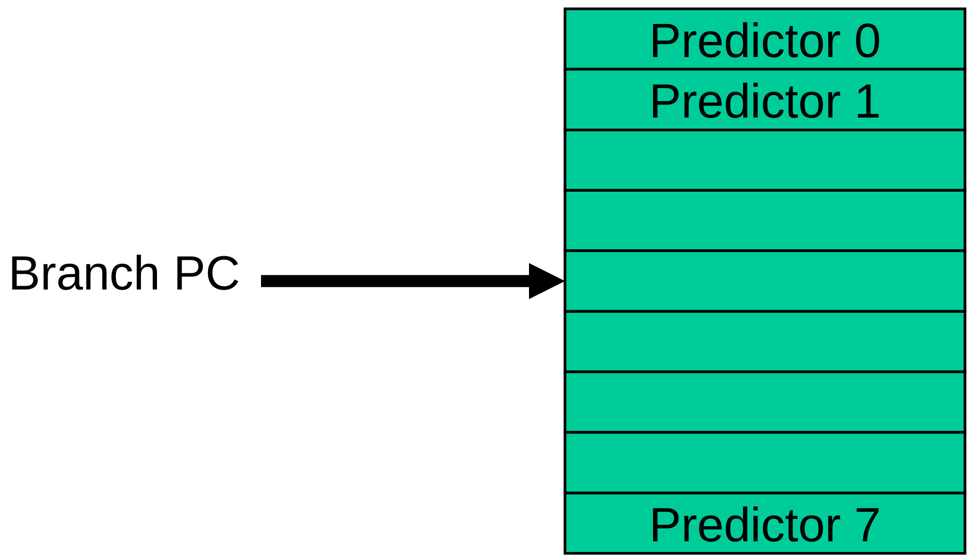
# n-bit Predictors

- n-bit scheme αλλάζει πρόβλεψη μόνο αν γίνει λάθος πρόβλεψη $2^{n-1}$ φορές.

- Χρήση n-bit counter.

- Μελέτες των n-bit predictors έχουν δείξει ότι οι **2-bit predictors αποδίδουν εξίσου**: τα περισσότερα συστήματα έχουν 2-bit predictors.

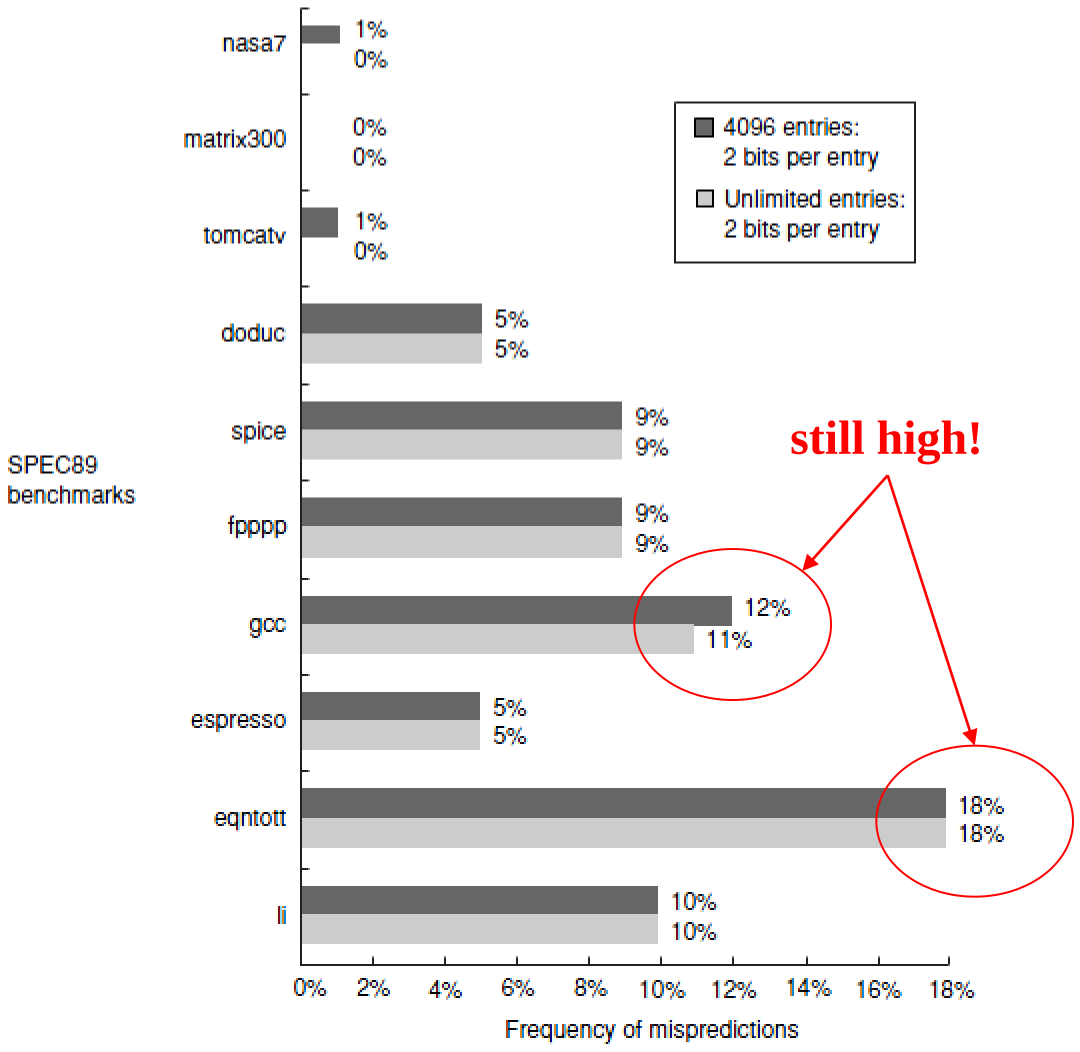
# Use of Branch History Table



- BHT είναι ένας πίνακας από “Predictors”
  - Usually 2-bit, saturating counters
  - Indexed by PC address of Branch
- Στο ID stage του branch κάνουμε access τον πίνακα. Στο ID stage υπολογίζεται και η διεύθυνση του branch.
- Όταν το branch ολοκληρώθει
  - Update corresponding Predictor

# 2-bit BHT Accuracy

- Λάθος πρόβλεψη επειδή:
  - Wrong guess for that branch
  - Got branch history of wrong branch



# Correlating Predictors

- ***Υπόθεση* :** πρόσφατα branches συσχετίζονται. Τα αποτελέσματα πρόσφατων branches επηρεάζουν την πρόβλεψη του τρέχον branch.
- ***Παράδειγμα 1 (από eqntott benchmark):***

## Example in C

```
if (aa==2)
        aa=0;
if (bb==2)
        bb=0;
if (aa != bb) {...}
```

## Example in assembly

```
        DSUBUI        R3,R1,#2
        BNEZ          R3,L1         ;branch b1 (aa!=2)
        DADD          R1,R0,R0      ;aa=0
L1:     DSUBUI        R3,R2,#2
        BNEZ          R3,L2         ;branch b2 (bb!=2)
        DADD          R2,R0,R0      ;bb=0
L2:     DSUBU         R3,R1,R2      ;R3=aa-bb
        BEQZ          R3,L3         ;branch b3 (aa==bb)
```

*If branches b1 and b2 are both not taken (i.e. first two if statements are true) then b3 will be taken*

# Correlating Predictors

- ***Παράδειγμα* 2:**

## Example in C

```
if (d==0)
        d=1;
if (d==1) {...}
```

## Example in asembly

```
        BNEZ          R1,L1         ;branch b1 (d!=0)
        DADDIU        R1,R0,#1      ;d==0, so d=1
L1:     DADDIU        R3,R1,#-1
        BNEZ          R3,L2         ;branch b2 (d!=1)
...
L2:
```

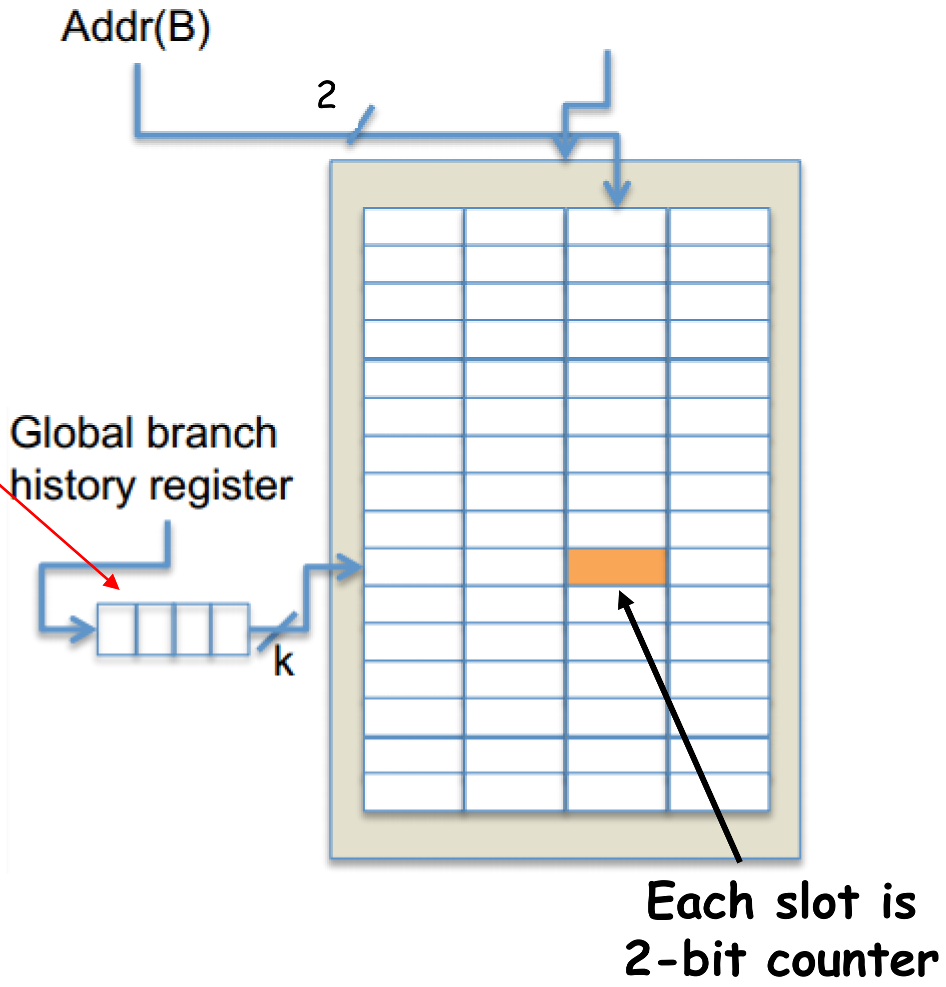
## Possible execution sequences

| Initial value of d | d==0? | b1 | Value of d before b2 | d==1? | b2 |
|---|---|---|---|---|---|
| 0 | yes | not taken | 1 | yes | not taken |
| 1 | no | taken | 1 | yes | not taken |
| 2 | no | taken | 2 | no | taken |

# Example: (4,2) Predictor

**(4,2) GHT (Global History Table) predictor**

- 4 means that we keep four bits of history
- 2 means that we have 2 bit counters in each slot.
- Then behavior of recent branches selects between, say, 16 predictions of next branch, updating just that prediction
- Note also that aliasing is possible here...



**2-bit global branch history register**

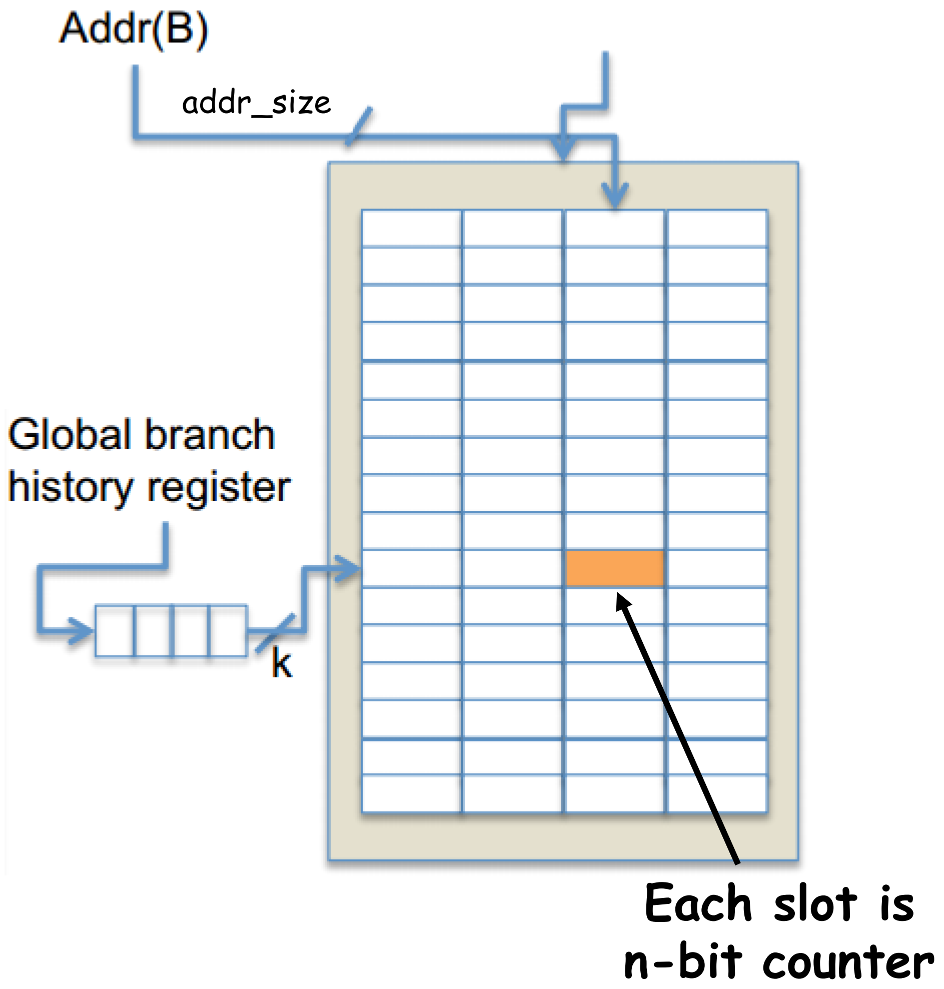
# Correlating Branches

**(k,n) GHT predictor**

- k means that we keep k-bits of history
- n means that we have n-bit counters in each slot.
- Note that the original two-bit counter solution would be a (0,2) GAs predictor
- Συνολική μνήμη :

  $2^{k} * n * 2^{addr\_size} =$

  $2^{k+addr\_size} * n$



2-bit global branch history register

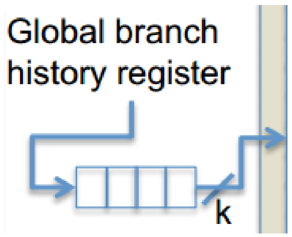
# Yeh and Patt's classification

## Taxonomy of two-level branch predictors

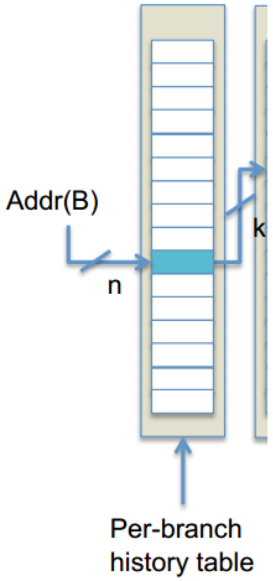
- XAy predictor
- X=G, global history register
- X=P, per-branch history register
- X=S, per-set-of-branches history register
- y=g, global branch history table
- y=p, per-branch history table
- y=s, set-associative per branch history table
  - Set-associative mapping of branch PCs reduces conflicts (aliases)

# Yeh and Patt's classification

- *First Level:*
  - τα k πιο πρόσφατα branches που εκτελέστηκαν οπουδήποτε
    Produces a "**GA***y*" (for "global address") in the Yeh and Patt classification

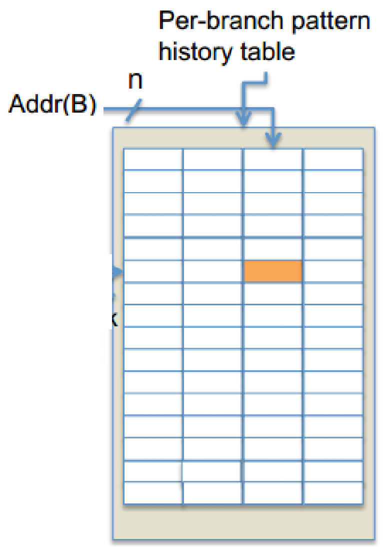


  - τα k πιο πρόσφατα αποτελέσματα του ίδιου branch.
    Produces a "**PA***y*" (for "per address") in same classification (e.g. PAg)

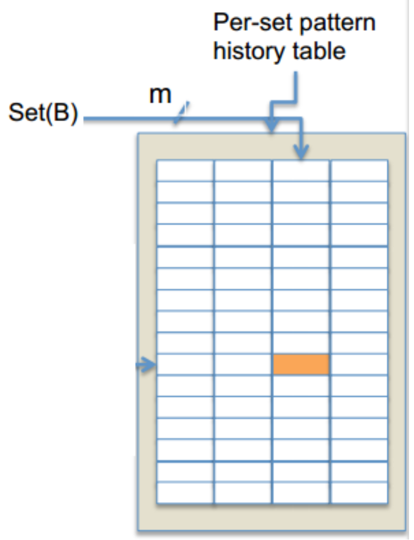


# Yeh and Patt's classification

- *Second Level:*
  - Single entry for any branch "*XA***g**"
  - Per branch history table "*XA***p**"



  - Per set history table "*XA***s**"



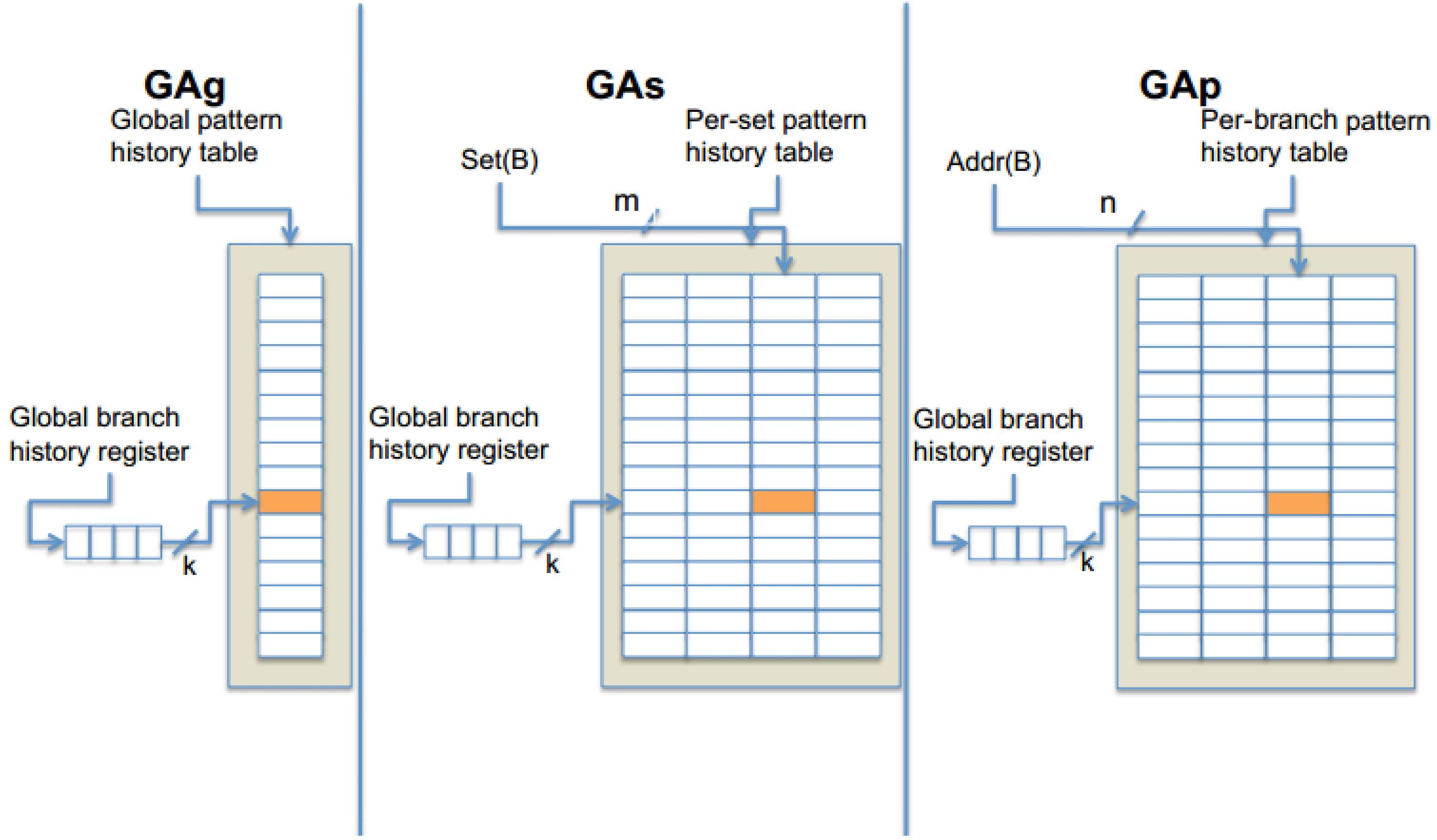
# Yeh and Patt's classification

## Variations of two-level branch predictors

| Variation | Description |
|---|---|
| GAg | Global Adaptive Branch Prediction<br>using one global pattern history table |
| GAs | Global Adaptive Branch Prediction<br>using per-set (of branch PCs) pattern history tables |
| GAp | Global Adaptive Branch Prediction<br>using per-address (of branch PC) pattern history tables |
| PAg | Per-address Adaptive Branch Prediction<br>using global pattern history table |
| SAg | Per-Set Adaptive Branch Prediction<br>using global pattern history table |
| SAs | Per-Set Adaptive Branch Prediction<br>using per-set pattern history tables |
| SAp | Per-Set Adaptive Branch Prediction<br>using per-address pattern history tables |

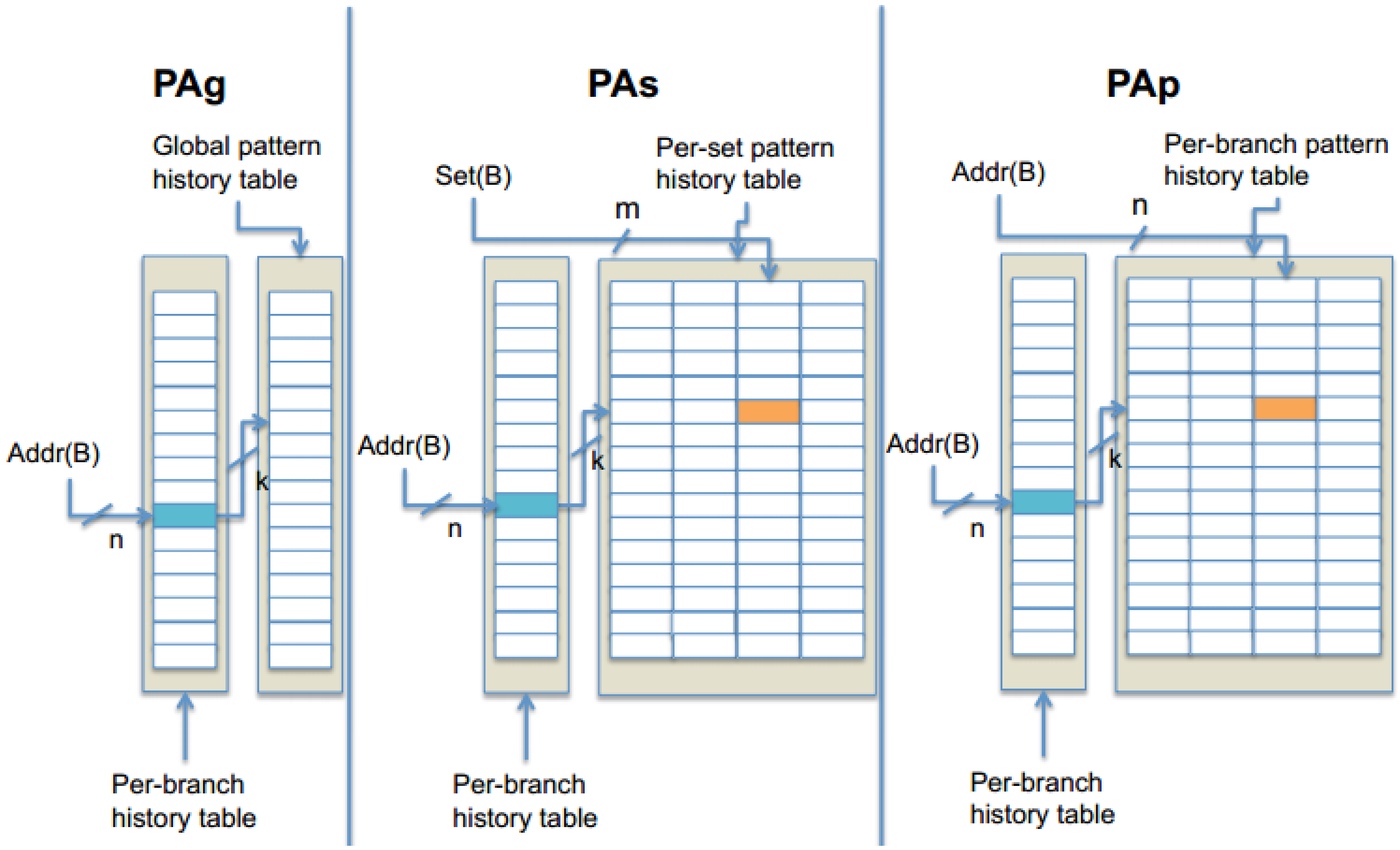
# Yeh and Patt's classification

## Global history predictors


GAg
Global pattern history table
Global branch history register
k
GAs
Per-set pattern history table
Set(B)
m
Global branch history register
k
GAp
Per-branch pattern history table
Addr(B)
n
Global branch history register
k

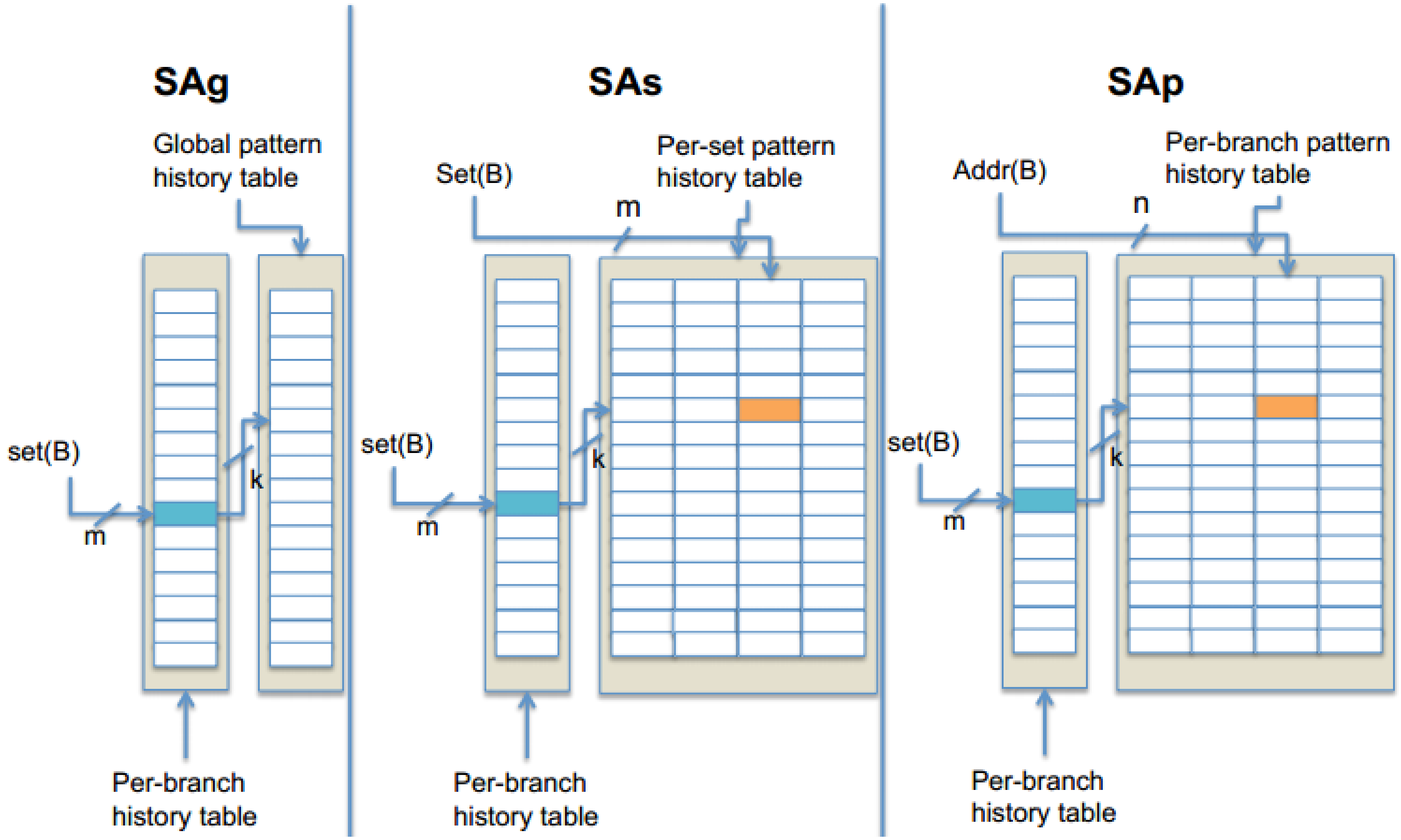
# Yeh and Patt's classification

## Per-branch history predictors

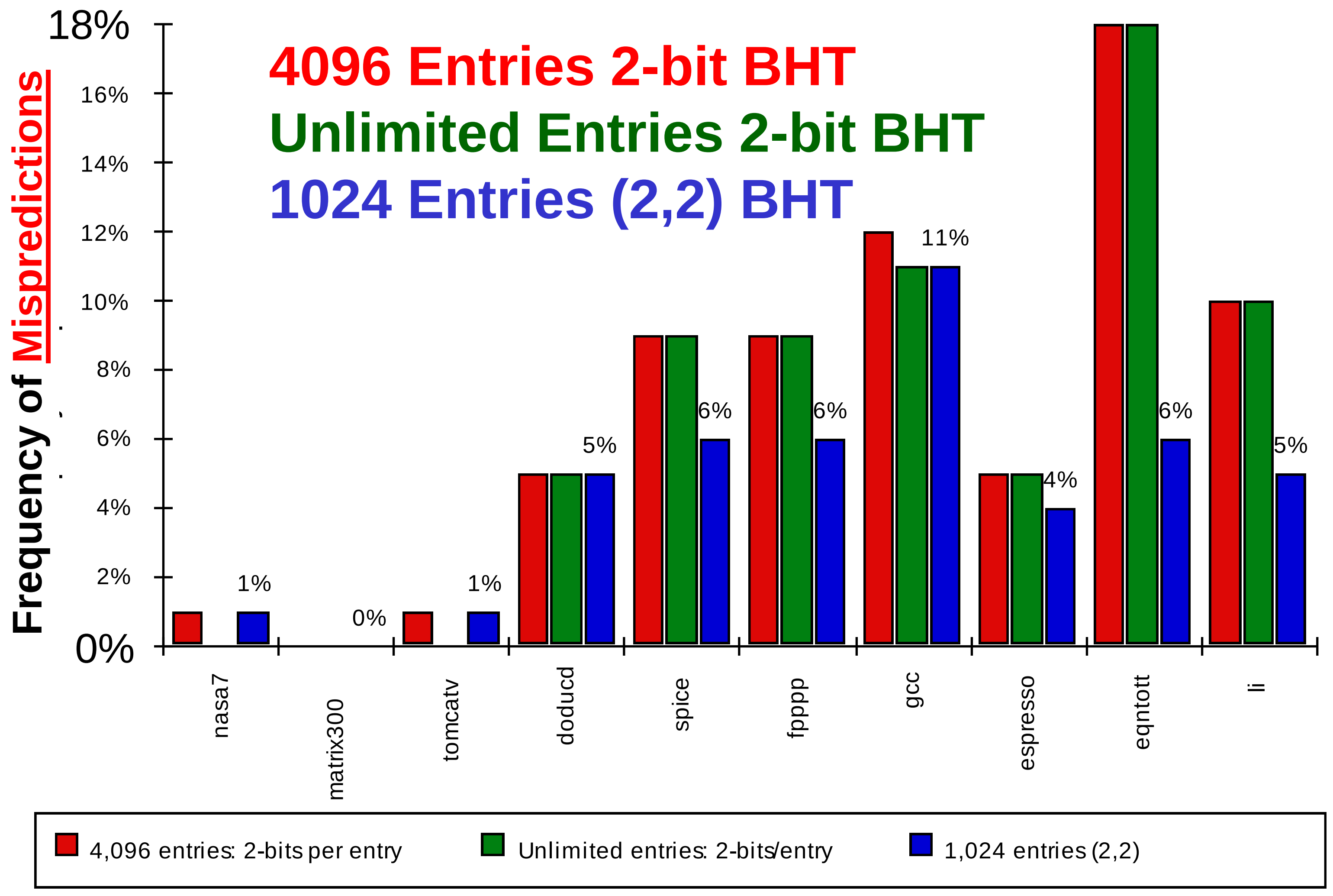


# Yeh and Patt's classification

## Per-set history predictors

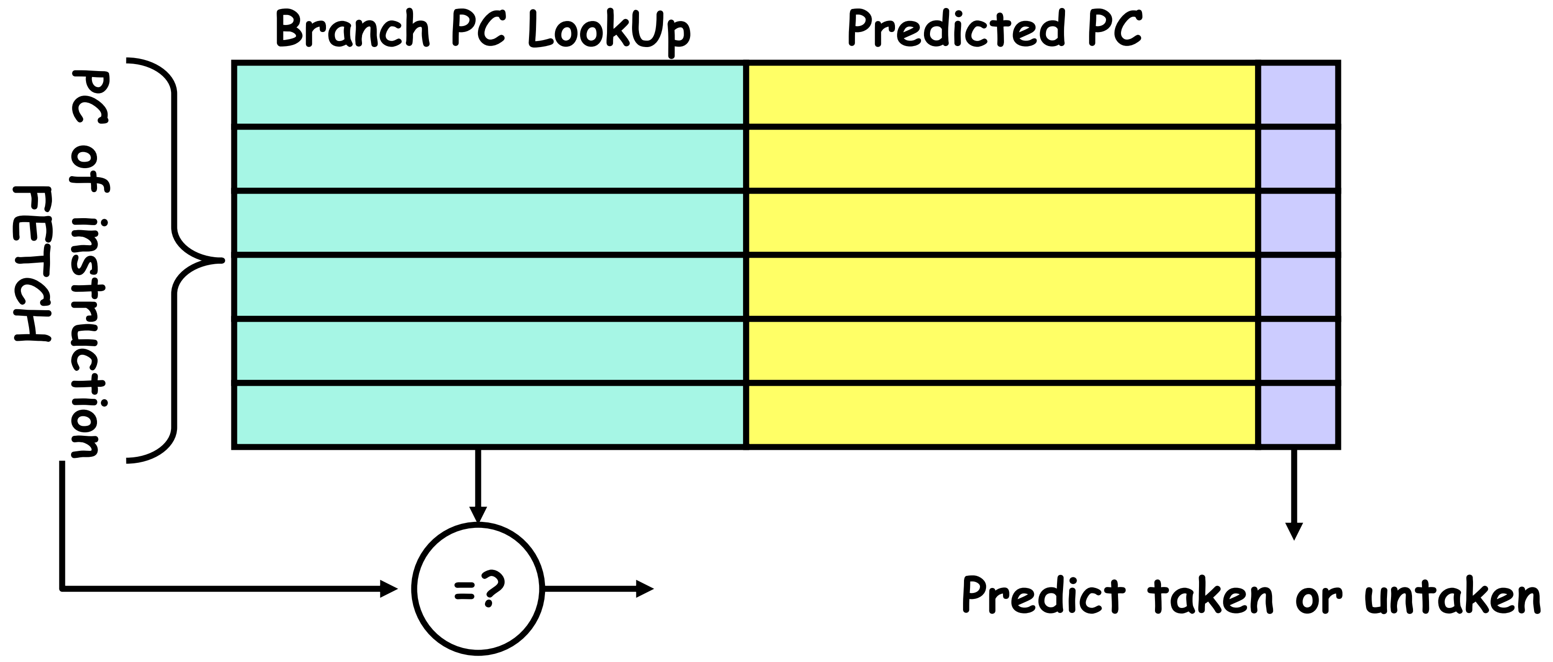


# Accuracy of Different Schemes



# Branch Target Buffer

- Ένα look up table με διευθύνσεις των predicted taken branches. Όταν κάνει match στέλνει το target PC του branch.
- *Optimization*: Store the predicted-taken branches only



# Branch Target Buffer

- To Branch Target Buffer (BTB) γίνεται access στο IF stage: γλυτώνουμε 1 κύκλο.

| IF | ID | EX |
|---|---|---|
| Send PC to memory and BTB. | Send Out predicted PC if entry found in BTB | Update BTB |

## Branch Folding

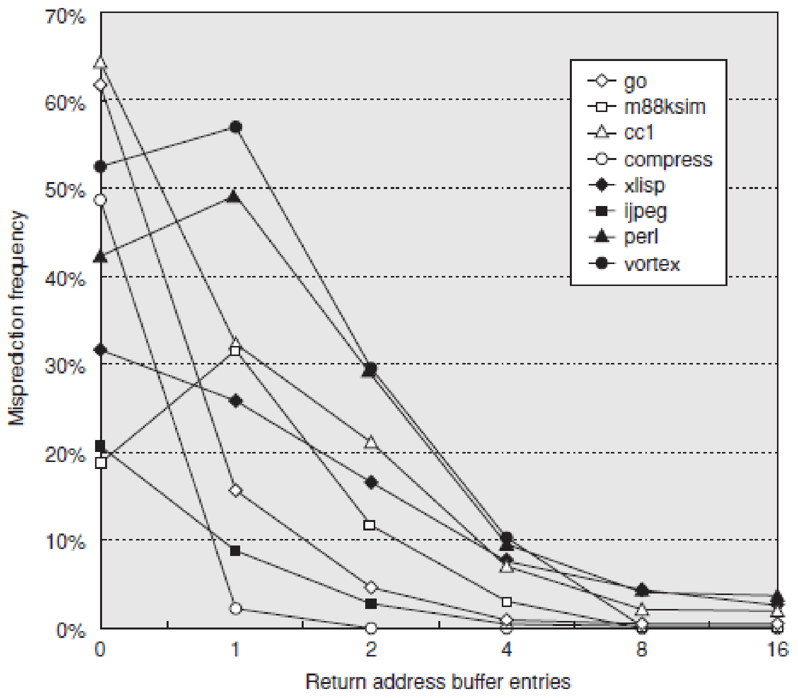
- Αποθήκευσε μία ή περισσότερες target instructions μαζί με target address στο Branch Target Buffer (BTB).

  - Χρόνος πρόσβασης στον BTB μπορεί να μεγαλώσει.

  - zero-cycle branches. Αντικατέστησε την branch εντολή με την target instruction.

# Indirect Jumps

- Indirect procedure calls, select ή case statemenet, gotos. Πλειοψηφία είναι procedure returns.
- ***Πρόβλημα*** ☞ accuracy low when procedure is called from multiple sites.
- ***Λύση*** ☞ small buffer of return addresses operating as a stack. This structure caches the most recent return addresses: pushing a return address on the stack at a call and popping one off at a return.

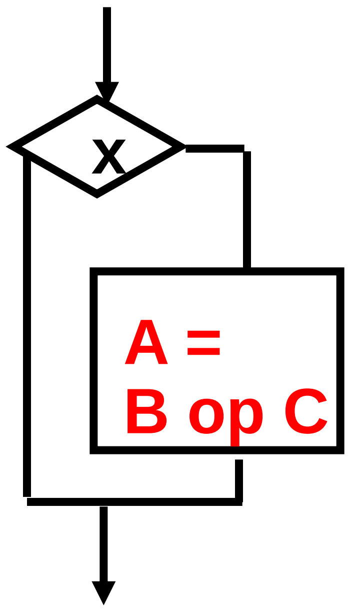


# Predicated Execution

- Απόφυγε branch prediction μετατρέποντας τα branches σε conditionally executed instructions:

  if (x) then A = B op C else NOP

  - If false, then neither store result nor cause exception
  - Expanded ISA of Alpha, MIPS, PowerPC, SPARC have conditional move; PA-RISC can annul any following instr.
  - IA-64: 64 1-bit condition fields selected so conditional execution of any instruction



- Αρνητικά των conditional instructions
  - Still takes a clock even if "annulled"
  - Stall if condition evaluated late
  - Complex conditions reduce effectiveness; condition becomes known late in pipeline

# Dynamic Branch Prediction

- Η πρόβλεψη είναι ιδιάιτερα σημαντική για την απόδοση του συστήματος.
  - Prediction is exploiting “information compressibility” in execution
- **Branch History Table:** 2 bits for loop accuracy
- **Correlation:** Recently executed branches correlated with next branch.
  - Either different branches (GA)
  - Or different executions of same branches (PA).
- **Branch Target Buffer:** include branch address & prediction
- **Predicated Execution** μπορούν να ελαττώσουν των αριθμό των branches, αριθμό των λάθος προβλέψεων.