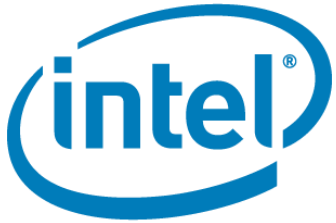


Branch prediction

Grigory Rechistov
With help of Alexander Titov
MDSP

March, 2011

MDSP project, Intel Lab,
Moscow Institute of Physics and Technology

Agenda

- Quick introduction
- Reasons to use
- Static (software) prediction
- Saturating counter
- Two-level adaptive predictor
- Local/global predictors
- Branch target predictor
- Branch predication

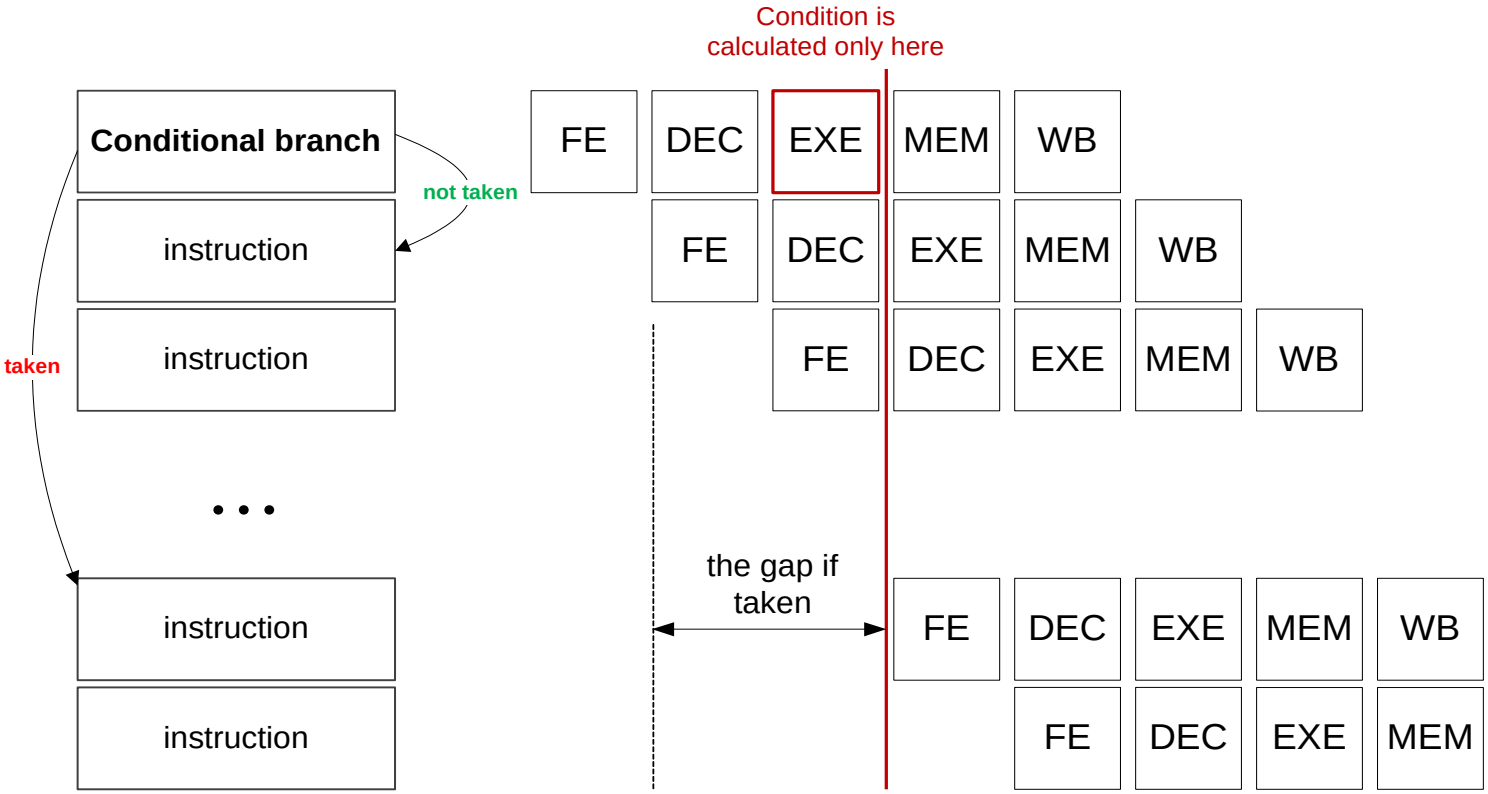
What is a branch predictor

A branch predictor is a digital circuit that tries to guess which way a jump instruction (e.g. an **if-then-else** structure) will go before this is known for sure.

Branch instruction types

	Conditional	Unconditional
Direct	if - then- else for loops (bez, bnez, etc)	procedure calls (jal) goto (j)
Indirect		return (jr) virtual function lookup function pointers (jalr)

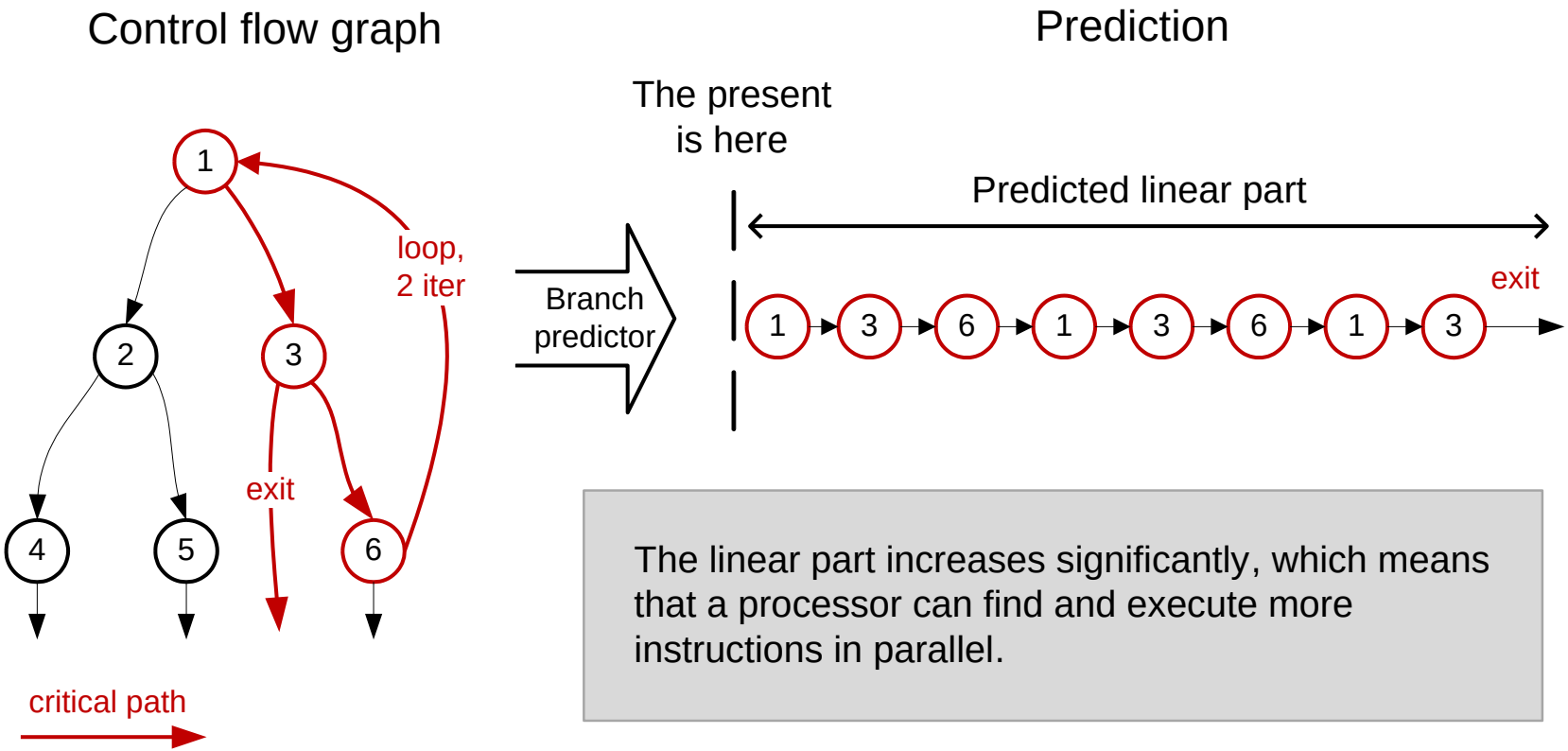
The 1st reason: the gap in pipeline



- The time that is wasted in case of a branch misprediction is equal to the number of stages in the pipeline from the fetch stage to the execute stage. Modern microprocessors tend to have quite long pipelines so that the misprediction delay is between 10 and 20 cycles (if not consider misses in caches).

The 2nd reason: parallel execution

- A processor use special technique to identify instruction can be executed in parallel, but the linear part (instruction between two branches) is to small to select enough instruction (every 5-8th instruction is a branch).
- The branch prediction can increase the linear part:



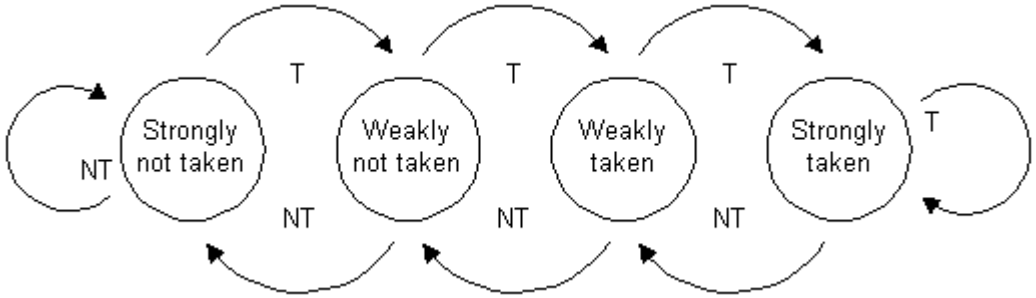
The linear part increases significantly, which means that a processor can find and execute more instructions in parallel.

Static (software) prediction

- There are situations as the branch direction can be statically predicted with a good probability:
  - Backwards branches
  - Exception and error handling
  - Some branches meet with specific heuristics
- The backwards branch is one that has a target address that is lower than its own address. Therefore SBP can help with prediction accuracy of loops, which are usually backward-pointing branches, and are taken more often than not taken.
- Some processors allow branch prediction **hints** to be inserted into the code to tell whether the static prediction should be taken or not taken. In this case it is also possible to use the profile information (the history of execution of a program).

Saturating counter

- A saturating counter or bimodal predictor is a state machine with four states:

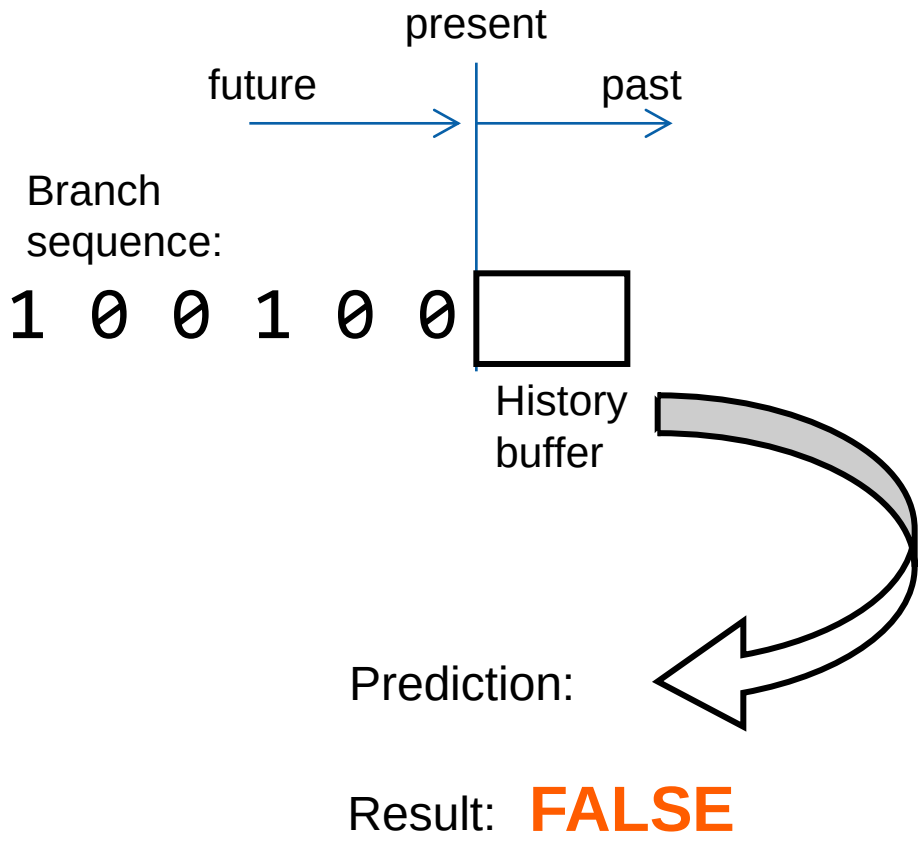


- When a branch is evaluated, the corresponding state machine is updated. Branches evaluated as not taken (NT) decrement the state towards strongly not taken, and branches evaluated as taken (T) increment the state towards strongly taken

- The advantage of the two-bit counter over a one-bit scheme is that a conditional jump has to deviate twice from what it has done most in the past before the prediction changes. For example, a loop-closing conditional jump is mispredicted once rather than twice.

Two level adaptive predictor

- Conditional jumps that are taken every second time or have some other regularly recurring pattern are not predicted well by the saturating counter, but an adaptive predictor can easily solve this problem



Pattern history table

00	strongly NT	weakly NT	weakly T	strongly T
01	strongly NT	weakly NT	weakly T	strongly T
10	strongly NT	weakly NT	weakly T	strongly T
11	strongly NT	weakly NT	weakly T	strongly T

Local predictor

- Local prediction
  - Local branch predictor has a separate history buffer for each conditional jump instruction.
  - Pattern history table may be separate as well or it may be shared between all conditional jumps.
  - Accuracy of the local predictor is equal about 97.1%.

Global predictor

- Global prediction
  - Global branch keeps a shared history of all conditional jumps: in this case any correlation between different conditional jumps is utilized in the prediction making, but when there is no correlation the prediction rate can decrease.
  - This predictor must have a long history buffer in order to work. The size of the pattern history table grows exponentially with the size of the history buffer. Therefore, the big pattern history table is shared between all conditional jumps.
  - Accuracy of the global branch predictor is equal about 96.6%, which is just a little worse than large local predictors.

Real branch predictors are designed as a combination of both types.

Branch target prediction

- The address of a **dynamic** branch isn't known statically in the moment of compilation. It is calculated in the moment of execution of the program.

- A branch target predictor is the part of a processor that predicts the target address of a taken dynamic conditional branch or an dynamic unconditional branch instruction before the target address of the branch instruction is computed by the execution unit of the processor.

And now to something completely different!

Branch predication

- Convenient machine code with jumps:

```
branch if condition to label1
   do_that
   branch to label2
label1:
   do_this
label2:
   ...
```

Branch predication (contd.)

- Predicated code:

```
(condition) do_this
(not condition) do_that
```

- With branch predication, all possible branch paths are executed, the correct path is kept and all others are thrown away.
- **Pros**: Can be faster if conditional blocks are short enough.
- **Cons**: encoding space; unbalanced paths

Architectures with branch predication

- European designs of 1950s: Mailüfterl (1955), the Zuse Z22 (1955), the ZEBRA (1958), and the Electrologica X1 (1958)
- Intel Itanium – almost every instruction is predicated. Non-predicated ones have “always true” predicate
- ARM ISA – almost all instructions are predicated (13 predicates).
- Even Intel IA-32 has some sort of predicated instructions: **`CMOVxx`** (conditional move)

Bibliography

1. http://www.cs.virginia.edu/~skadron/cs654/slides/bpred.ppt
2. Daniel A. Jim´enez. Fast Path-Based Neural Branch Prediction. http://cava.cs.utsa.edu/pdfs/micro03_dist.pdf

Here be Intel logo