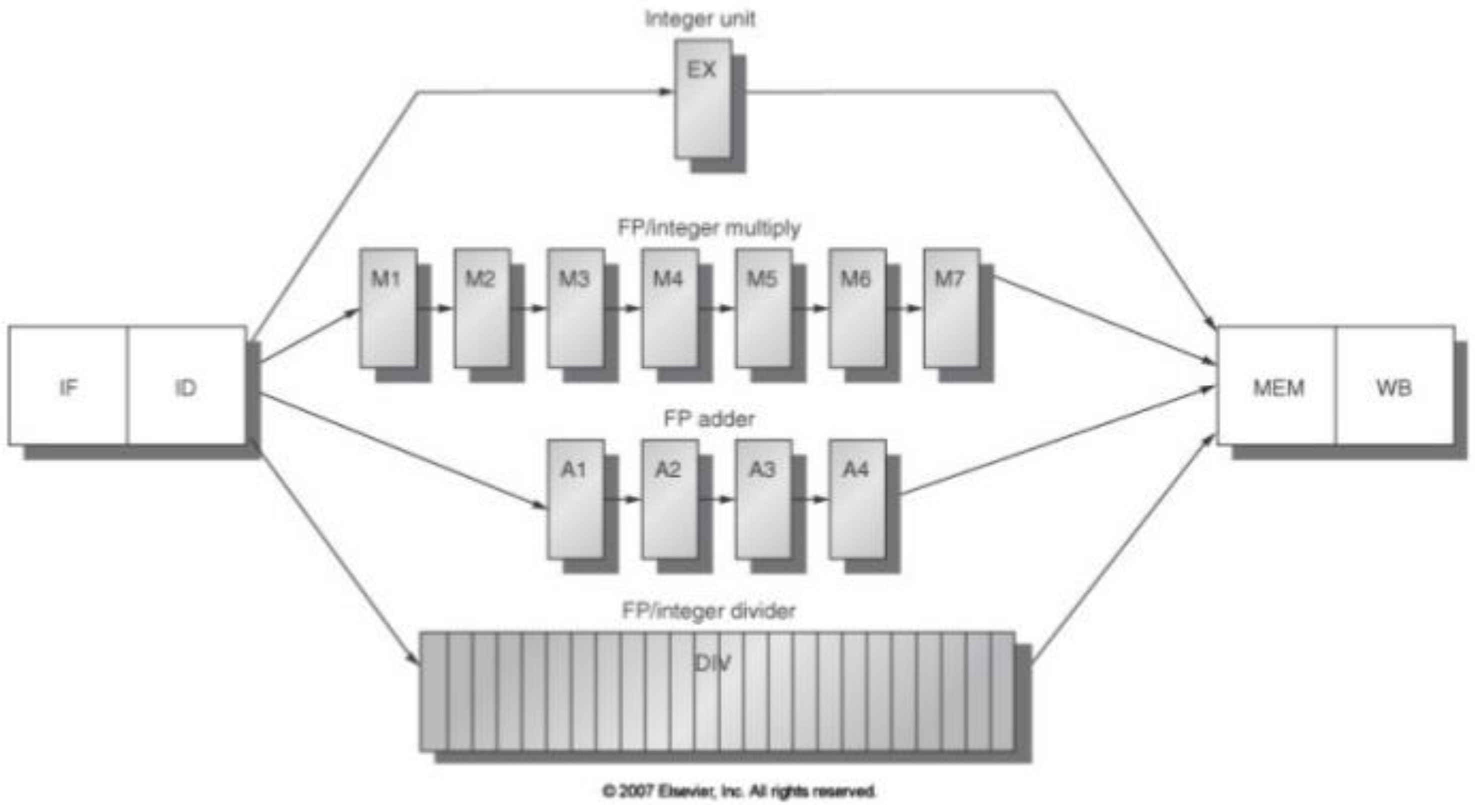
Tutorial 5

Exceptions

Branch Prediction

MIPS FP pipeline



MIPS FP pipeline

<u>Sample program:</u>

```
Loop:   L.D      F1, 0(R1)
        MULT.D   F2, F1, F3
        ADD.D    F4, F1, F2
        DADDUI   R1, R2, #1
        BNEZ     R1, loop
        DSUBI    R6, R6, #1
```

How would those instructions be scheduled in this pipeline?

Exceptions

- Happen when exceptional conditions occur in the CPU
- Synonyms : interrupt, fault (not used consistently)

  e.g.:
  - Integer arithmetic overflow
  - Misaligned memory address
  - Undefined instruction
  - Breakpoint
  - OS service routine
  - ...
- Alter the flow of the program : the instructions (in the code) following the instruction that caused the exception should not proceed immediately. Some other operations might need to be performed before.

Exceptions

- Synchronous (are caused by the program itself its the data) or asynchronous (mostly external causes)
- User requested (program requests it/predictable) or coerced (caused by a hardware event/unpredictable)
- User maskable (the program can ignore the exception) or non-maskable (cannot)
- Within (in the middle of an instruction/mostly caused by what the instruction is doing) or between instructions
- Resume (program continues after exception handled) or terminate (program stops)

See Figure A.27 for examples

Exceptions

- Precise : If the pipeline can be stopped and
  - **Instructions that come after are cancelled, can be restarted from the beginning once the exception has been handled**
  - **Instructions that ran before the faulting instruction complete normally**

  - Why is that useful?
    - Floating point standard
    - Virtual memory
  - Difficulties?
    - **Simultaneous exceptions (e.g.: EX of instr2 and MEM of instr1)**
    - **Out-of-order exceptions (e.g.: IF of instr2 and MEM of instr1)**
    - **Floating point pipeline**
      - **Often complete out-of-order, e.g.: division and multiplication**
      - **Multi-cycle operations mean multiple instructions can be have issued**

Exceptions

- <u>Precise</u> : If the pipeline can be stopped and
  - **Instructions that come after are cancelled, can be restarted from the beginning once the exception has been handled**
  - **Instructions that ran before the faulting instruction complete normally**

- Difficulties:
  - **Simultaneous exceptions (e.g.: EX of instr2 and MEM of instr1)**
  - **Out-of-order exceptions (e.g.: IF of instr2 and MEM of instr1)**

- In MIPS, exception status vector with each instruction in the pipeline. Exception flag cancels writes of faulting instruction and following ones. Status vector checked between MEM/WB -> exceptions handled in order.

Branch prediction

- Main idea: **guess** since we don't know which way a branch will go
  - If we're right, one CC gained compared to flushing
  - If we're wrong, same loss as flushing
- Static schemes: don't depend on historical trends
  - Predict taken, predict non-taken
- Dynamic schemes: depend on past behavior of the branches
  - Local: look at a given branch in isolation
  - Global: look at previous branches in correlation with the current one

1-bit branch predictor

- *Main idea*: we use the history of a branch's past outcomes to predict its future outcomes.
- *1-bit predictor*: when we execute a branch, first check if it was taken when it was <u>last executed</u>: if yes, predict it will be taken this time; if no, predict it will not be taken this time.

- *Example*: `for (i=0; i<10; i++) { … };`
  This loop is iterated 10 times and involves one branch, e.g.,

```
loop: ...
          SLTI R2,R1,#10  ;is i<10?
          BNEZ R2,loop    ;branch if yes
```

  The branch is taken in the first 9 iterations, and not taken on the 10th iteration. What's the result of using 1-bit prediction?

1-bit branch predictor (cont'd)

- (Assume that when the branch is executed for the first time, we predict that it is not taken.)

Iteration #:	1	2	3	4	5	6	7	8	9	10
Predicted branch outcome:	N	T	T	T	T	T	T	T	T	T
Actual branch outcome:	T	T	T	T	T	T	T	T	T	N

- How is 1-bit branch prediction implemented in hardware?
  - Use a branch history table: there is an entry in the table for every branch encountered in the program. (Actually, table is indexed using lower bits of branch instruction's PC. * -> 1 possible)
  - Each table entry contains *one prediction bit* for that branch, e.g., 0 for predict not taken (N), 1 for predict taken (T).

1-bit branch predictor (cont'd)

- The prediction bit is used to predict the branch outcome. It is updated after the branch's *actual* outcome is known.
- The following shows how the prediction bit for the branch in our example changes in each iteration of the loop:

Iter#	Pred. bit	Actual outcome	Update
1	N	T	T
2	T	T	T
3	T	T	T
4	T	T	T
5	T	T	T
6	T	T	T
7	T	T	T
8	T	T	T
9	T	T	T
10	T	N	N

2-bit branch predictor

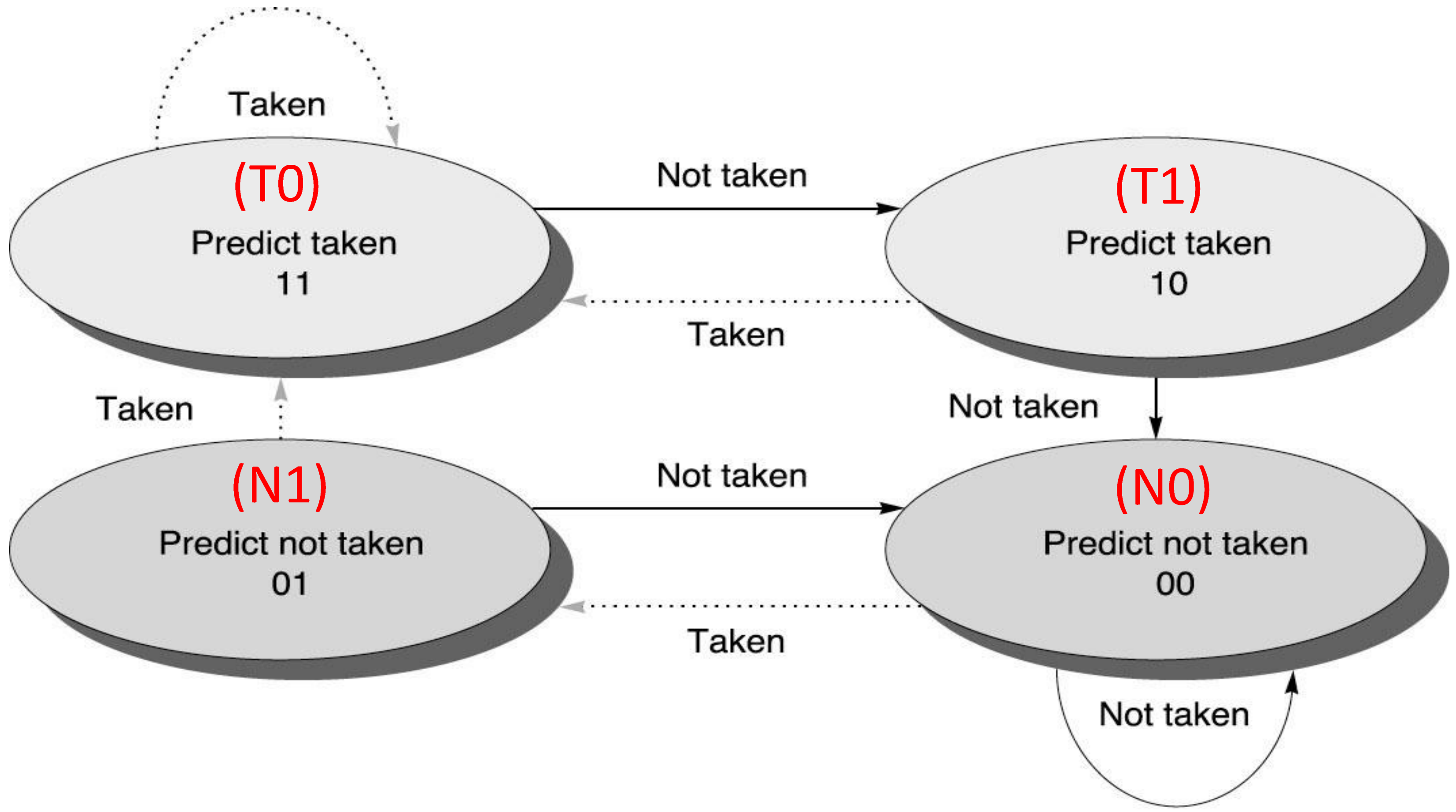
- With 1-bit prediction, if we mispredict once about the branch, we change our mind instantly about the next prediction.
- This might not be a good thing, thus use *2-bit branch prediction* instead: the idea is that we have to mispredict *twice in a row* before changing our mind.
- *Example* (imagine a *for* loop executed again and again):

```
Predicted with 1-bit:  …NTTTTNTTTTNTTTTNTTTT…
Predicted with 2-bit:  …NNTTTTTTTTTTTTTTTTTT…

Actual outcome:        …TTTTNTTTTNTTTTNTTTTN…
```

2-bit branch predictor (cont'd)

- Each entry in the table now needs *2 prediction bits*. They are updated according to the following:



2-bit branch predictor (cont'd)

- Let's look at *one loop* from previous example:

```
Predicted with 2-bit: NNTTTTTTTTTTTTTTTTTT…
Actual outcome:       TTTTNTTTTNTTTTNTTTTN…
```

Iter#	Pred. bit	Predicted outcome	Actual outcome	Update
1	N0	N	T	N1
2	N1	N	T	T0
3	T0	T	T	T0
4	T0	T	T	T0
5	T0	T	N	T1

Correlating branch predictors

- Sometimes the outcome of one branch depends on the outcomes of *other* branches in the code, e.g.,

  ```
  B1:if (a == 0)
         b = 1;
  B2:if (b != 1)
          …
  ```

  Here the outcome of the second branch B2 depends on the outcome of the first branch B1. In other words, the branch outcomes are *correlated*.
- We can exploit this correlation: use the outcomes of <u>previously executed</u> branches to predict the outcome of the current branch.
- We keep track of different predictions for all possible outcomes of the previous branches.

Correlating branch predictors (cont'd)

A note on the the (m, n) notation:

m: number of previous branches correlated

n: number of bit for predictor

- **(1, 1)** correlating branch predictor means
  - Use the outcomes of the previous 1 branch executed
  - And use a 1-bit branch predictor
- **(2, 1)** correlating branch predictor means
  - Use the outcomes of the previous 2 branches executed
  - And use a 1-bit branch predictor
- **(1, 2)** correlating branch predictor means
  - Use the outcomes of the previous 1 branch executed
  - And use a 2-bit branch predictor

Correlating branch predictors (cont'd)

- (1, 1) correlating branch predictor: each branch history table entry has 2 fields of 1 bit each, e.g.,

branch	if prev branch not taken	if prev branch taken
B2	T	N
B3	T	T

B2 will be predicted taken if the previous branch executed was not taken and not taken if the previous branch executed was taken. B3 will be predicted taken in both cases.

- Similarly, for a (1, 2) predictor, we could have:

branch	if prev branch not taken	if prev branch taken
B2	T1	N0
B3	N0	N1

Example

A machine uses a (1, 2) correlating branch predictor. Consider a code segment that has three branches, B1, B2, and B3. At one point in execution, all three branches have been repeatedly taken, and all their prediction bits are set to T0. A sequence of branch outcomes that occurs *right after* this point are given below, listed in the order that the branches are executed. What is the branch misprediction rate for this sequence?

```
Branch   | B1  B2  B3  B2  B1  B3  B1  B3  B1  B2  B1
---------+----------------------------------------------
Outcome  |  N   T   N   N   N   T   N   T   N   T   T
```

Example (cont'd)

Solution: Assume that the two predictors are named predictor 1 and predictor 2, where predictor 1 is used if the previous branch executed was taken, and predictor 2 is used if the previous branch executed was not taken.

Branch	B1	B2	B3	B2	B1	B3	B1	B3	B1	B2	B1
Pred. 1											
Pred. 2											
Predicted Outcome											
Actual Outcome	N	T	N	N	N	T	N	T	N	T	T
Update 1											
Update 2											

Example (cont'd)

Branch	B1	B2	B3	B2	B1	B3	B1	B3	B1	B2	B1
Pred.1 (T)	T0	T1	T1	N0	N0	N0	N0	N0	N0	T0	T0
Pred.2 (N)	T0	T0	T0	T0	T1	N0	N1	N1	T0	T0	T0
Predicted Outcome	T	T	T	T	T	N	N	N	N	T	T
Actual Outcome	N	T	N	N	N	T	N	T	N	T	T
Update 1	**T1**	T1	**N0**	N0	N0	N0	N0	N0	N0	T0	T0
Update 2	T0	T0	T0	**T1**	**N0**	**N1**	N1	**T0**	T0	T0	T0

So the branch misprediction rate is 6/11 = 54.5%.

Exercise: repeat using a (1, 1) correlating branch predictor.