Branch Predication (cont'd)

Example:

```
if (a && b)
        j = j + 1;
else{
        if (c)
                k = k + 1;
        else
                k = k - 1;
        m = k * 5}
i = i + 1;
```

Assumptions:

The values are stored in registers, as follows:
a: R0; *b*: R1; *j*: R2; *c*: R3; *k*: R4; *m*: R5; *i*: R6.

This sequence (for an ordinary processor) would be compiled to:

```
        BZ      R0, L1      branch if a == 0
        BZ      R1, L1      branch if b == 0
        ADI     R2, R2,#1   R2 ← R2 + 1;(integer)
        BR      L4
L1:     BZ      R3, L2      branch if c == 0
        ADI     R4, R4,#1   R4 ← R4 + 1;(integer)
        BR      L3
L2:     SBI     R4, R4,#1   R4 ← R4 - 1;(integer)
L3:     MPI     R5, R4,#5   R5 ← R4 * 5;(integer)
L4:     ADI     R6, R6,#1   R6 ← R6 + 1;(integer)
```

Branch Predication (cont'd)

Let us read it in this way:

```
if not(a == 0) and not(b == 0)                          ADI   R2, R2,#1
if not(not(a == 0) and not(b == 0)) and not(c == 0)     ADI   R4, R4,#1
if not(not(a == 0) and not(b == 0)) and not(not(c == 0))  SBI   R4, R4,#1
if not(not(a == 0) and not(b == 0))                     MPI   R5, R4,#5
                                                        ADI   R6, R6,#1
```

Branch Predication (cont'd)

The same with predicated execution:
(all predicates are initialised as *false*)

```
(1)  P1, P2 = EQ(R0, #0)
(2)  <P2> P1, P3 = EQ(R1, #0)
(3)  <P3> ADI R2, R2,#1
(4)  <P1> P4, P5 = NEQ(R3, #0)
(5)  <P4> ADI R4, R4,#1
(6)  <P5> SBI R4, R4,#1
(7)  <P1> MPI R5, R4,#5
(8)  ADI      R6, R6,#1
```

- The compiler can plan all these instructions to be issued in parallel, except (5) and (6) which are data-dependent.
- Instructions can be started before the particular predicate on which they depend is known. When the predicate will be known, the particular instruction will or will not be committed.

Speculative Loading

You remember when we discussed "delayed loading" (Fö. 5/6):

The load is placed so that memory latency is avoided (the value is already there when it's needed):

```
LOAD   R1,X      loads from address X into R1
ADD    R2,R1     R2 ← R2 + R1
ADD    R4,R3     R4 ← R4 + R3
SUB    R2,R4     R2 ← R2 - R4
```

```
LOAD   R1,X      loads from address X into R1
ADD    R4,R3     R4 ← R4 + R3
ADD    R2,R1     R2 ← R2 + R1
SUB    R2,R4     R2 ← R2 - R4
```