Performance effect of Branch Prediction

$p_r[p_t(1+d) + (1- p_t)1] + (1- p_r)[p_t(1+c) + (1- p_t)(1+c)]$

average number of cycles per branch instruction =
$p_r[p_t(1+d) + (1- p_t)1] + (1- p_r)[p_t(1+c) + (1- p_t)(1+c)] =$
$p_r(p_td + 1) + (1- p_r)(1+c)$

Substituted into
$t_{ave} = p_b$(average number of cycles per branch instruction) +
$(1- p_b)$(average number of cycles per non-branch inst.)
with p_b = probability instruction is a branch
$t_{ave} = p_b[p_r(p_td + 1) + (1- p_r)(1+c)] + (1- p_b)(1)$
$= 1 + cp_b(1 – p_r) + dp_bp_rp_t$

Performance effect of Branch Prediction

$t_{ave} = 1 + cp_b(1 – p_r) + dp_bp_rp_t$

For example d = 1 and $p_r = 0.9$
For a 5-stage pipe with c=3
$t_{ave} = 1 + 3*0.2*0.1 + 1*0.2*0.65*0.9 = 1.12$
Efficienncy = $1 / t_{ave} * 100 = 89\%$

Multiple or Target Prefetching

- On first encounter with a branch we initiate a prefetch of its target path code at ID in order to lower delay if the branch is taken.

Delayed Branching

- A compiler/assembler technique is to delay braching for k cycles after the branch instruction , providing time to fetch the target path.
- Usefull instructions or NOPs instructions are inserted

Example

Delay the next k sequential instructions

```
i1:  LOAD  R1, A
i2:  LOAD  R2, B
i3:  BRZR  R2, i7    -- branch to i7 if R2=0
i4:  LOAD  R3, C
i5:  ADD   R4, R2, R3
i6:  MUL   R5, R1, R2
i7:  ADD   R4, R1, R2
```

- Assuming that k=2, the compiler modifies this code by moving the instruction i_1 and NOP instruction after the branch instruction i_3.

```
i2:  LOAD  R2, B
i3:  BRZR  R2, i7
i1:  LOAD  R1, A
     NOP
i4:  LOAD  R3, C
i5:  ADD   R4, R2, R3
i6:  MUL   R5, R1, R2
i7:  ADD   R4, R1, R2
```


Super-scalar vs. Super-pipelines

- Super-scalar and Very Long Instruction Word (VLIW) design exploits the high circuit density of CMOS to provide multiple functional units and hence more parallelism.

Super-scalar vs. Super-pipelines

Example

- For example with two fp-add pipes large vector additions can be distributed.

```
DO I=1 to 10000
    c(i) = a(i) + b(i)
END

DO I=1 to 10000 by 2
    c(i) = a(i) + b(i)        c(i+1) = a(i+1) + b(i+1)
    ADDER_1                   ADDER_2
END
```

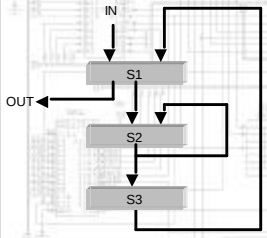

Super-pipelines

- Superpipelines rely on:
  - High speed technology
  - Small pipe stages
  - Hence long pipes to increase the performance.
- Such a design is susceptible to control hazards
  - Needs good branch prediction to perform well

Pipeline Control and Scheduling

Consider a re-circulating pipe

We consider only static pipes



	t_0	t_1	t_2	t_3	t_4
Stage 1	X				X
Stage 2		X	X		
Stage 3				X	

Reservation Table

0, 4
0, 1
0

Forbidden List

Definitions

- Latency:
  - The interval between two initiations in a stage. Latencies between reservations are forbidden latencies.
- Forbitten List:
  - The vector forbidden latencies
    - F = (4, 1, 0)
- Collision Vector:
  - $C = (c_n, \ldots, c_1, c_0)$
  - where $c_i = \{ 1 \; i \in F, 0 \; i \notin F\}$
  - Eg. $C = (c_4, c_3, c_2, c_1, c_0)$
    $C = (1, 0, 0, 1, 1)$

Collision Vector

- When C = (1, 0, 0, 1, 1) it seen that $c_i = 0$ means we may safely insert a new input to the pipe.
  - Hence at 2,3 or 5,6
- If we do insert a new input at e.g. t = 2 , then the forbitten latencies may alter, e.g.

	t_0	t_1	t_2	t_3	t_4	t_5	t_6
Stage 1							
Stage 2							
Stage 3			X			Y	

Reservation Table

0, 2, 4
0, 1, 2
0, 2

Forbidden List

Collision Vector

with altered forbitten latencies may

	t_0	t_1	t_2	t_3	t_4	t_5	t_6
Stage 1					X		Y
Stage 2						Y	
Stage 3			X				

Reservation Table

0, 2, 4
0, 1, 2
0, 2

Forbidden List

- When it seen that $c_i = 0$ means we may safely insert a new input to the pipe.
  - C_2 = (1, 0, 0, 1, 1) OR C = (0, 0, 1, 0, 0)
    = (1, 0, 1, 1, 1)

State Diagram

- Hence we may decribe the forbidden latencies of the pipe with a statediagram, starting at:
  - C = (1, 0, 0, 1, 1)