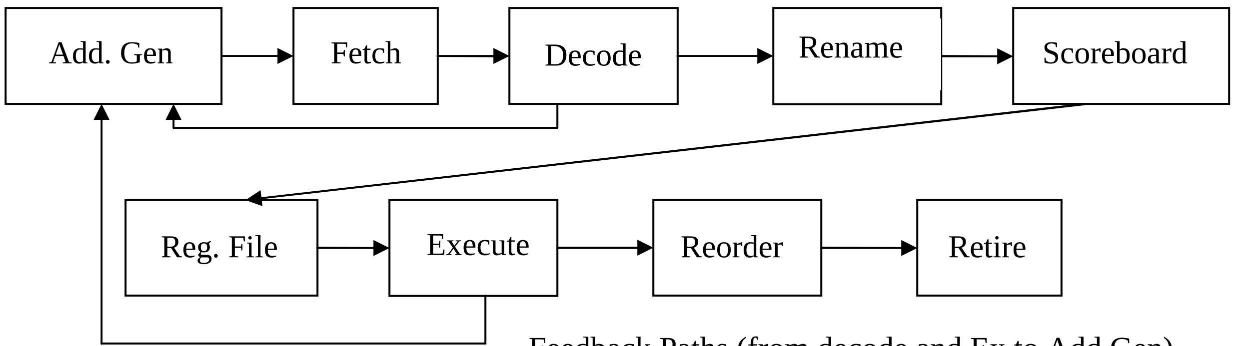
ECE 511: Computer Architecture	Wed, September 8, 2004

Lecture 4

Lecturer: Matthew Frank	***Scribe: Saunvit Pandya***

Pipeline Diagram



Feedback Paths (from decode and Ex to Add Gen)

<u>Fetch Stage</u>

1) Why is it important? – Important module to minimize latency and therefore improve throughput
2) Structure (of Instructions)
   a. Feedback Loops
3) Is Branch Prediction just a Hack? – Essentially, intelligent guess-and-check algorithms

Assume pipeline is on a ***superscalar*** machine – can issue multiple instructions at a time.

<u>Branches</u>

For example:

1004:	OpCode [BGez]	R17	Relative Target [+24]

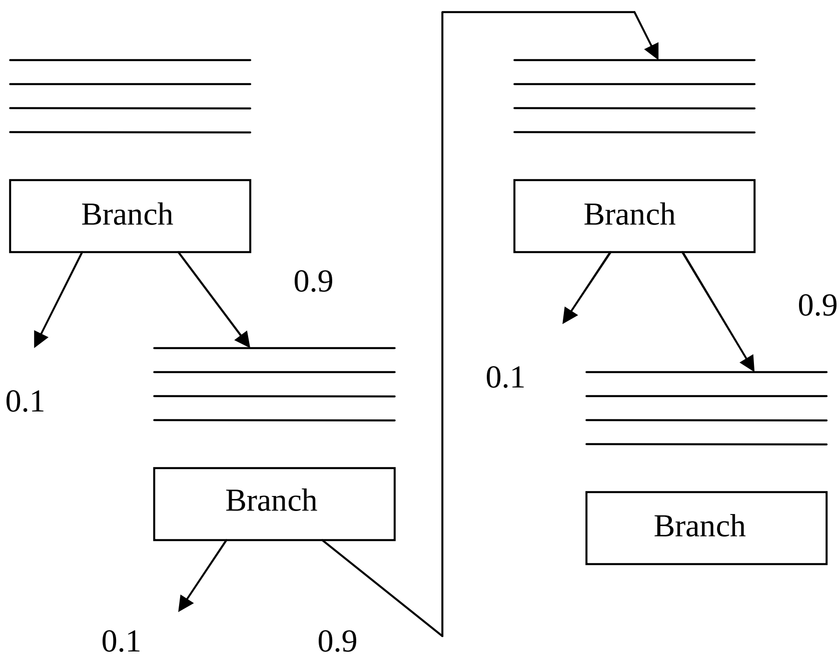
If R17 $\geq$ 0 then PC = PC + Rel. Target = PC+24 = 1028
Else PC = PC + 4 = 1008

996: R17 $\leftarrow$ R15 + R16

By the time you get to 1004, 996 is still waiting to be executed in the scoreboard stage! Therefore, to keep your pipeline busy, you need branch prediction!

Assumption: 1/5 of all instructions are branches, with each branch being biased by 0.9.

Branch Flow Diagram



Therefore we have the following table,

Branch Block	1	2	3	4	i
Probability of Fetch n blocks and then missing	.1	.9 x .1	.9 x .9 x .1	.9^3 * .1	.9^(i-1) * .1

Using the properties of geometric series, we have:

$$\left(\sum_{i=1}^{i=\infty}(.9)^{i-1}\right)*0.1=1$$

This verifies that the sum of all probabilities is 1, and thereby our probability model is accurate. Now to calculate the mean number of basic blocks between mispredictions, we use a similar geometric series:

$$\sum_{i=1}^{\infty} i*(0.1)*(0.9)^{(i-1)} = 0.1*\sum_{i=1}^{\infty} i*(0.9)^{(i-1)} = (.1)\left(\frac{1}{(1-.9)^2}\right) = 10$$

The use of geometric series allows us to extend our analysis on prediction rates v. misprediction blocks over a variety of values:

Prediction Rate (h)	Miss Rate (m)	Average Number of Instructions Between Misses
.8	.2 = 1/5	25
.9	.1 = 1/10	50
.95	.05 = 1/20	100
.96	.04 = 1/25	125
.975	.025 = 1/40	200
.99	.01 = 1/100	500!

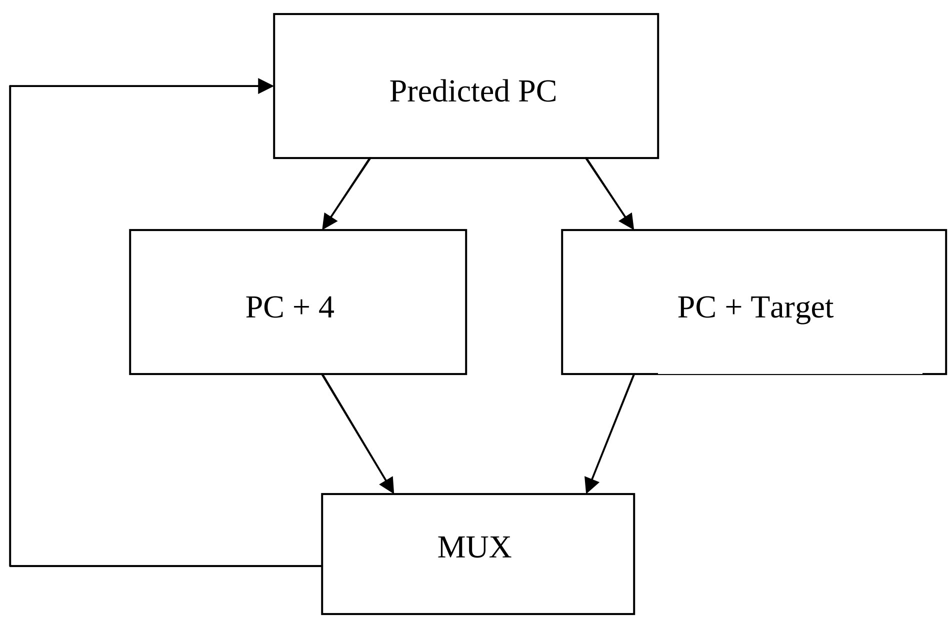
Back to the Pipeline

Problems:

- ➔ Renaming has to happen in order
- ➔ Straddling cache lines
- ➔ Fetch boundaries
- ➔ Resource contention
- ➔ Each box in the pipeline is a like a state machine.

Address Generation State of Pipeline

Simple Branch Decision Diagram



The key concept of ***Feedback****:* From execute (and later) stages, send in correct PC, iff there was a branch misprediction.

<u>Branch Target Buffer</u>

Ex. Direct Mapped BTB

1004	1028
1016	
1064	
2068	
..	..
..	..
TAG (the PC that holds the calling instruction) – used to index the BTB	TARGET (the PC we predict that we have to go to)

Predicted PC -> Hash Function -> BTB Entry

- ➔ Don't look at the last 2 bits (why?: notice the word boundaries)
- ➔ If 2^n cache, use the lower n bits (the next ones up from the last 2).
    - o Ex: If 8-bit BTB cache, use bits 3 to 10.

BTB Details:

- ➔ Use BTB to forecast branches and, most importantly, ***branch targets***.
- ➔ Signals from Execute Stage update BTB.
- ➔ Like other caches
- ➔ Can be set associative and/or fully associative