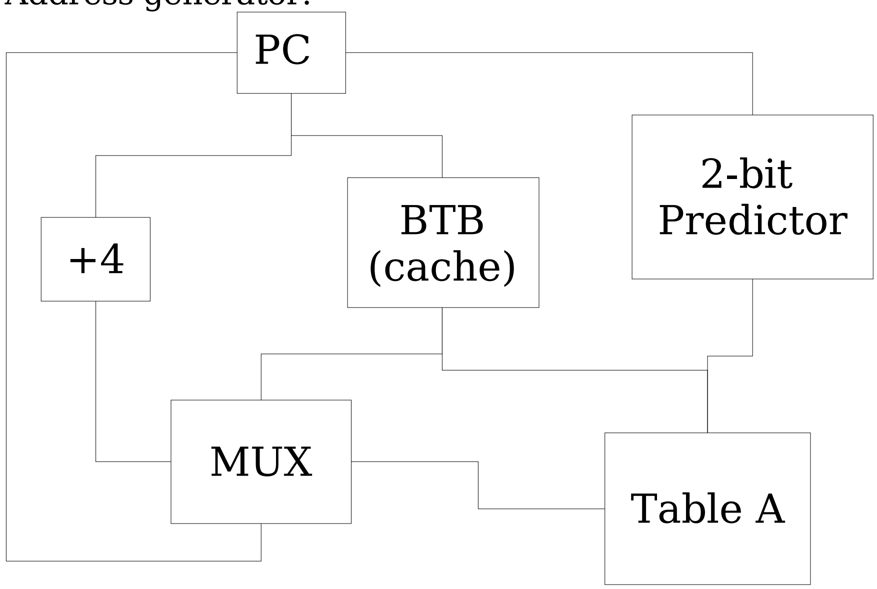
Lecture 6 ECE 511

September 15 2004

Recap

- High bandwidth instruction fetch is very important
- Some branches are very predictable
- Address generator:



Direct Branches

jmp address	unconditional branch to an absolute address
bnz $17, +56	conditional branch to PC relative address
call address	unconditional branch to an absolute address

All information needed to resolve the branch is contained in the instruction.

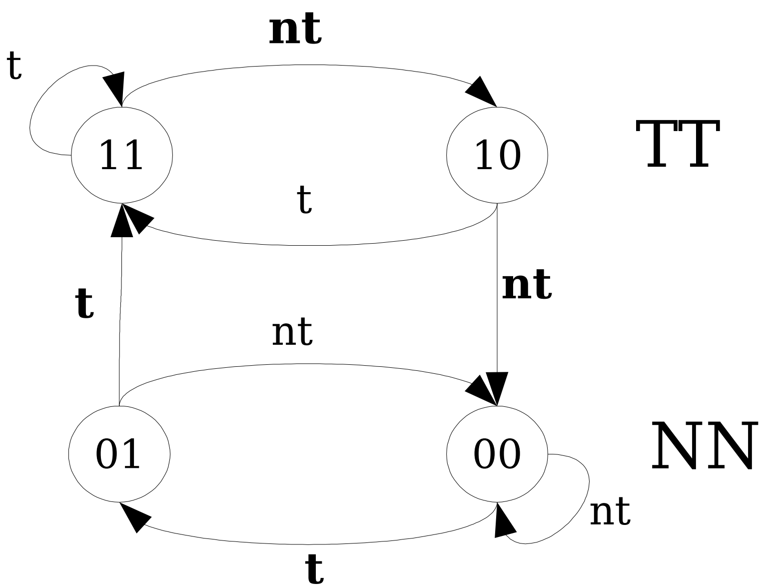
Indirect Branches

ret	Return from a function – all function returns are indirect branches
jr $23	jump register – unconditional branch to an absolute address in register $23
jalr $24	jump and link register – unconditional branch to an absolute address in register $23

- Information needed to resolve branch is contained in the instruction and in data register.

- BTB's are excellent for direct branches, but less for indirect branches

2-bit Smith predictor



Right Bit

- 0 if most recent branch was not taken
- 1 if most recent branch was taken

Left Bit

- 0 if most recent adjacent pair of branches were NOT Taken
- 1 if most recent adjacent pair of branches were Taken

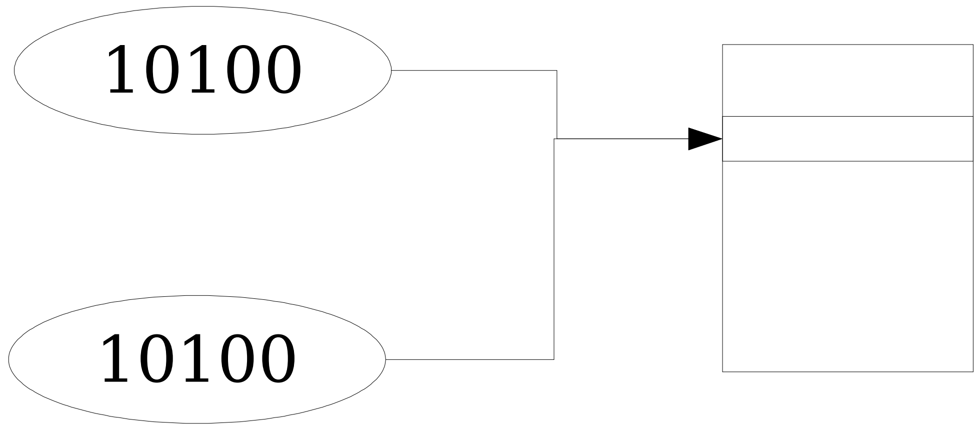
Mis-predicted branches are in **BOLD**.

- The code pattern that produces 100% or thereabouts mispredicts is much less likely that the one for the simple 2 bit saturating predictor. The pattern is:
  Pattern: NNTTNNTTNNTTNNTT...
  Pattern for 2 bit saturating predictor: NTNTNTNTNTNTNTNTNT...
- This 2 bit predictor will not mispredict the second iteration of a loop like a 1bit predictor.

Agreement Predictor

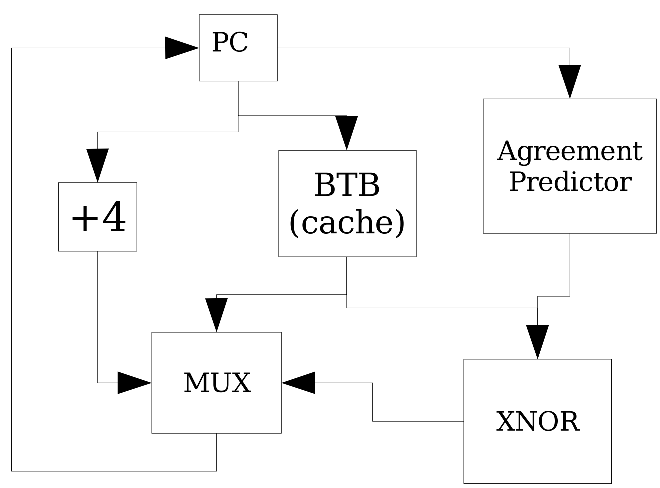
- the BTB returns a hit if a branch was taken recently, else it returns a miss
- the predictor returns an agree or a disagree signal
- For example:
  Consider a predictor table with only 4 entries

➔ the hash function uses bite 3 and 2 of the branch address
➔ so, branch instructions at address 4 [00100] and 20 [10100] both map to predictor entry 01



- The BTB is 80% accurate by itself. So, there is a 64% probability (0.8x0.8) that the BTBcan predict two branches at address 4 and 20. There is only a 4% probability (0.2x0.2) that the BTB will incorrectly predict both branches
- behavior of the branches hashed to the same entry in the predictor

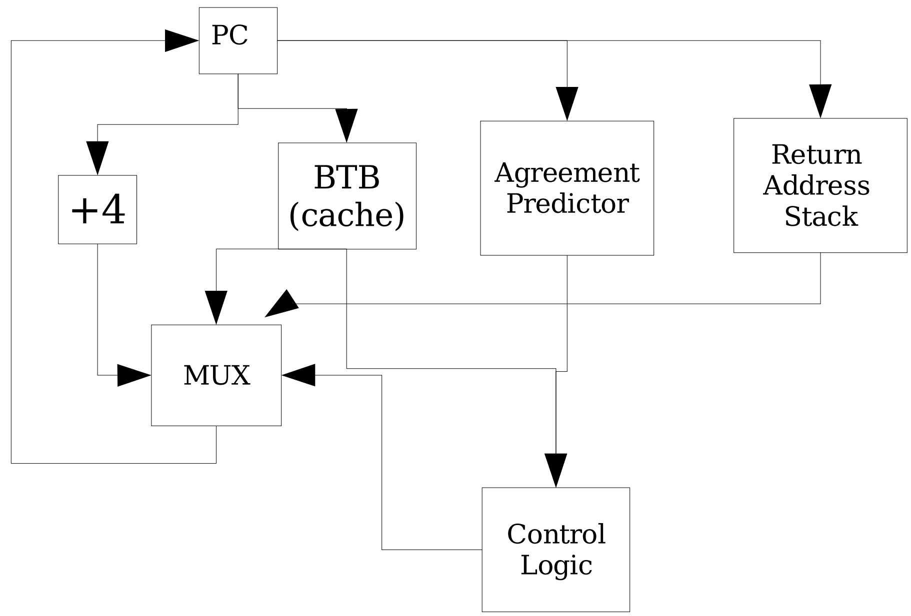
	miss	hit
agree	0	1
disagree	1	0



Return Address Stack

- Return from function call implemented using a return address register, $31
  - ➔ Very convenient implementation: when jumping to a subroutine, the CPU loads the return address register with the address of the instruction following the function call
- To accommodate nested function calls the return address register

must really be a stack. Return addresses are pushed by function calls and popped by function returns.



NOTE: after a context switch the BTB, predictor, and RA stack all contain totally bogus values. So branch prediction is very inefficient immediately after a context switch.

- We need feedback from instruction decode to determine which branches are function calls/ returns, so that we can control pushes and pops to the RA stack. This makes things slow.
- Alternatively, we can store a couple of flags in the BTB to indicate which branches addresses are calls/returns/other. So the BTB can generate the control logic for the stack w/o waiting for feedback from Instruction Decode pipeline stage.

Branch Correlation

- For example
  printf("%d", my_int)
  printf("%e", my_float)
  ➔ the code to convert/print an int is very different from the code to convert/print a float
- Certain calls to printf will invoke a certain set of subroutines: correlated branches. The branch associated with printing an int [printf("%d",...)] is correlated with an internal branch [if (format=="%d")].