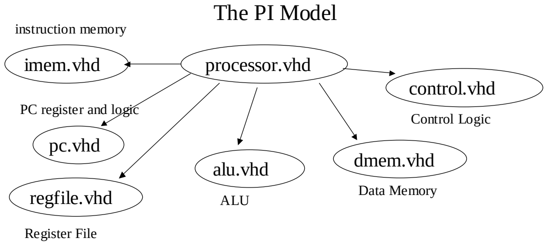
## The PI Model



Auxiliary files:

define.vhd : A VHDL package that contains MIPS opcode definitions. Used by *control*, *ALU*.

mips_components.vhd : A VHDL package that contains component definitions for *imem*, *pc*, *regfile*, *alu*, *dmem*, *control*. Used by processor.vhd.

## Comments on Pipelined Implementation Model

- No branch support at all
- Supports Jump
- Does not have hazard detection or forwarding logic
  - Will run the provided sample 'program.s' correctly as long as three NOPs between every instruction.
- Basically the SCI model with registers between blocks
- Control signals have extensions on them to indicate what stage they are used in. No extension means that this signal is not needed in another stage
  - e.g. memtoreg, memtoreg_ex, memtoreg_mem, memtoreg_wb
  - Data signals that are passed to various stages also use extensions
  - Tried to following signal naming convention in book

## Stalls (Hazard Detection)

- For the case of a LW followed by an ALU instruction that uses the loaded value, need to insert stall in the EX stage
  - *stall_ex* signal placed in code for this purpose, always a '0' right now
  - If stall_ex = '1', then control signals that can modify the processor state (like MemWrite, RegWrite) need to be zeroed.
- For a stall, also need to keep from incrementing the PC again, and prevent a new value from being loaded into the instruction register
  - PCwrite control writing the PC, always a '1' now. Also, instruction register loaded only if PCwrite = '1'.
- There is a process called "dostall" in 'processor.vhd" that needs to be modified to include stalls.

## Forwarding Paths

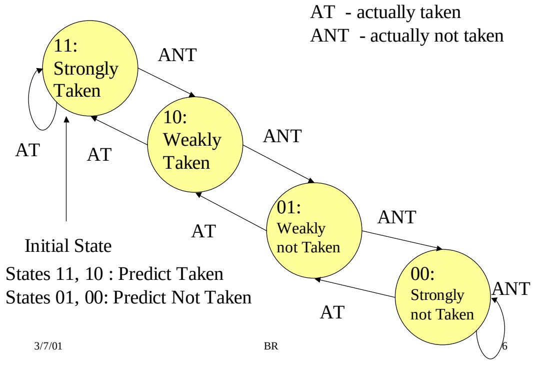
- There are no forwarding paths in the current model
- The process “fwdlogic” must be modified in *processor.vhd* to add forwarding.
  - The forwarding logic needs to alter the ‘data_a’ and ‘data_b’ values that go into the ALU
- The provided “program.s” will exercise many forwarding paths if the *nops* are removed.
  - The forwarding paths exercised are the ones from the MEM, WB stages to the EXE stage.
  - One path that is not exercised the case of a SW instruction needing forwarded value from the WB stage.

## Optional Assignment

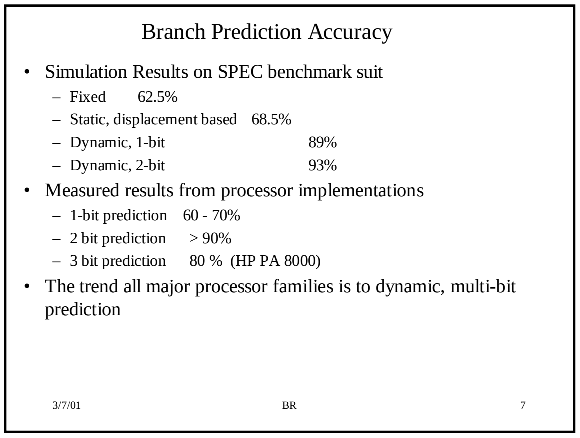
- Modify the processor.vhd to handle a LW stall and forwarding paths from MEM, WB stages to the EXE stages
- Check your work by removing the NOPs from the program.s file and executing the model.
- Hint: Remove NOPS for one instruction at time so that you can incrementally debug your added code.
- The ‘pi.do’ command file does not display all of the signals you may need during debug – feel free to add more signals!
- The modelsim.zip archive has been updated to contain the PI model.

## 2-Bit Prediction

AT - actually taken
ANT - actually not taken

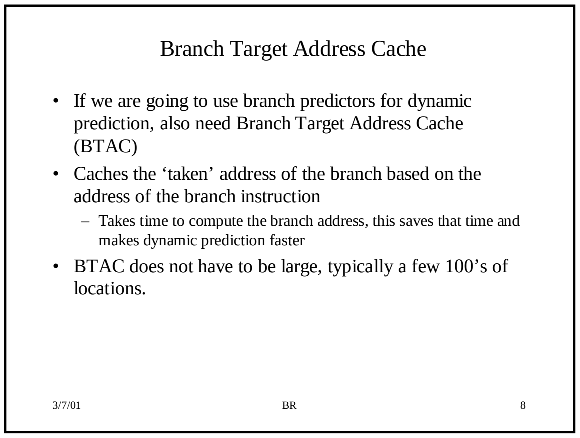


States 11, 10 : Predict Taken
States 01, 00: Predict Not Taken



## Branch Prediction Accuracy

- Simulation Results on SPEC benchmark suit
  - Fixed 62.5%
  - Static, displacement based 68.5%
  - Dynamic, 1-bit 89%
  - Dynamic, 2-bit 93%
- Measured results from processor implementations
  - 1-bit prediction 60 - 70%
  - 2 bit prediction > 90%
  - 3 bit prediction 80 % (HP PA 8000)
- The trend all major processor families is to dynamic, multi-bit prediction



## Branch Target Address Cache

- If we are going to use branch predictors for dynamic prediction, also need Branch Target Address Cache (BTAC)
- Caches the ‘taken’ address of the branch based on the address of the branch instruction
  - Takes time to compute the branch address, this saves that time and makes dynamic prediction faster
- BTAC does not have to be large, typically a few 100’s of locations.