# Project Goal

The goal of the project was to design a basic MIPS-like 5-stage pipelined processor with branch prediction unit.

# Instruction Set Architecture (ISA)

The three instruction types used are:

R-type : Register type

--------------------------------------------------------------------

B31-26 | B25-21 | B20-16 | B15-11 | B10-6 | B5-0

--------------------------------------------------------------------

opcode |rs |rt |rd |shift |funct

--------------------------------------------------------------------

I-type : Immediate type

--------------------------------------------------------------------

B31-26 | B25-21 | B20-16 | B15-0

--------------------------------------------------------------------

opcode | rs | rt | imm/offset

--------------------------------------------------------------------

J-type : Jump type

--------------------------------------------------------------------

B31-26 | B25-0

--------------------------------------------------------------------

opcode | target

--------------------------------------------------------------------

The instructions provided by the designed CPU are the followings:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Class | Type | Name | Opcode | Syntax | Semantics | Addressing Mode | Remarks |
| Data transfer | I-type | ld | 110000 | lw rt, offset(rs) | REG[rt] <- MEM[REG[rs] + offset] | Displacement | 32-bit load from memory (16-bit signed offset) |
| I-type | sd | 110001 | sw rt, offset(rs) | MEM[REG[rs] + offset] <- REG[rt] | Displacement | 32-bit store to memory (16-bit signed offset) |
| Arithmetic |  | nop | 000000 | nop |  |  |  |
| R-type | ddd | 000001 | add rd, rs, rt | REG[rd] <- REG[rs] + REG[rt] | Register direct | 32-bit add (no overflow detection) |
| I-type | dddi | 000010 | addi rt, rs, imm | REG[rt] <- REG[rs] + #imm | Register immediate | 16-bit immediate add to a 32-bit register (no overflow detection) |
| R-type | mul | 000011 | mul rd, rs, rt | REG[rd] <- REG[rs] + REG[rt] | Register direct | 16-bit numbers' multiplication |
| Control | I-type | beq | 010010 | beq rs, rt, offset | if condition is true, then PC <- PC + (offset<<2) | Register immediate | Branch if equal to zero (16-bit signed offset) |
| R-type | slt | 010100 | slt rd, rs, rt | if REG[rs] < REG[rt] then REG[rd] <- 1 | Register direct | set less than |

# Branch Prediction Unit

Branch prediction is used to enhance the performance of pipelining. The branch predictor pre-fetches a limited form of data and attempt to predict the result of branch instructions. The processor will speculatively execute instruction base on the predicted result from the branch predictor. A processor can have a better overall performance especially when the prediction rates are high enough to offset miss prediction penalties. Without branch prediction, a processor must stall whenever there are unresolved branch instructions.

A branch predictor is aimed to integrate into the pipeline processor to predict both a branch dynamically based on the information stored in the branch Target Buffer (BTB). The BTB is a small cache memory inside the branch predictor that used to record and update previous information of different branch instruction.

## Branch Target Buffer (BTB)

A BTB is a small piece of memory use to gather and store the information related to branch instruction. The BTB behave like a look-up table for the branch predictor to look for information of previous branch instruction that having same tag with the current address to perform branch prediction for the current instruction. If the branch instruction was not found in the BTB, a new entry will be created inside the BTB to store the information of the branch instruction. The information stored included valid  
bit, tag bit, branch target address and prediction state as per the following.

--------------------------------------------------------------------

B54 | B53-34 | B33-2 | B15-11 | B1-0

--------------------------------------------------------------------

validity |tag |Branch target address |prediction state

--------------------------------------------------------------------

PC input to the BTB is used as per the following:

--------------------------------------------------------------------

B31-12 | B11-2 | B1-0

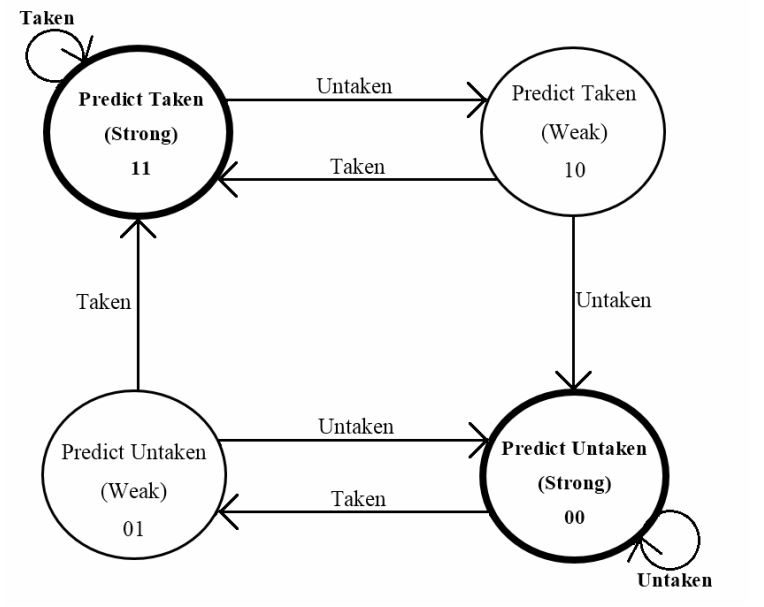
--------------------------------------------------------------------

tag |index |byte offset

--------------------------------------------------------------------

### State Transition of BTB

The following figure shows the FSM of the 2-bit BTB used in this project. A strong state will either remain at the same state when the evaluated condition (taken or untaken) is same at its state, or go to weak state of the same prediction when the evaluated condition is different. A weak state will not remain at its own state and there is no transition between two weak states. A weak state will go to strong state with same prediction when evaluated condition is same or directly go to a strong state of opposite prediction when the evaluate condition is different with the current state, instead of go to a weak state of opposite prediction.



**FSM of BTB**

**Current Implementation**

In this project work, the basic pipeline with above mentioned instructions are implemented. All instructions occupy one clock cycle. Separate data memory and instruction memory are used in order to avoid any possible structural hazards.

Though it was initially planned to implement the 2-bit branch prediction unit. But due to timing constraints, it has not been incorporated into the main design so far.

Testing of the design was carried out by hard-coding the instructions into the instruction memory and registers in register memory.