|  |
| --- |
| rochester institute of technology |
| The Smartest Little Computer |
| Hardware and Control Unit Document |
|  |
| **Jenny Zhen & Grant Kurtz** |
| **4/28/2014** |

|  |
| --- |
| The hardware and control unit document describes the basic Register Transfer Language (RTL) for all instructions in the CPU. The control unit describes the control flow of the RTL for managing the overall operation of the CPU. |

Table of Contents

[Introduction 2](#_Toc386393796)

[Architecture 2](#_Toc386393797)

[Diagram 2](#_Toc386393798)

[Branch Prediction Table 2](#_Toc386393799)

[Construction 2](#_Toc386393800)

[Registers 3](#_Toc386393801)

[Operation 3](#_Toc386393802)

[Register Transfer Language 5](#_Toc386393803)

[Fetch Phase 6](#_Toc386393804)

# Introduction

The purpose of this document is to specify the entirety of the simulated computer to test and demonstrate the effectiveness of branch prediction on a pipelined RISC architecture. The instruction set is kept simple and small but with a large enough variability to create a complete testing environment. Instructions are limited to only a single byte in width so as to keep the pipeline itself simple, to minimize the complexity necessary to add branch prediction to the pipeline.

# Architecture

## Diagram

See attached diagram at end of document.

## Branch Prediction Table

### Construction

The branch prediction table (BPT) is responsible for making accurate (albeit slow) predictions for whether an I- or J-Type instruction will branch. If it predicts taken, then the BPT will output the predicted PC to be used instead of the other generated values. The BPT is composed of 32 registers, 16 used for identification of branch statements, and 16 used for storing the new PC. Both sets of registers also contain information about historical pattern data used by the control unit to make a prediction. The algorithm for detecting jump or not jump is a Two-Level Adaptive Predictor.

The first bank of registers, known as the identification bank (I-bank), has the following format.

|  |  |  |  |
| --- | --- | --- | --- |
| Historical Values, 11-10 | 00, A-Predictor, 9-8 | 01, B-Predictor, 7-6 | PC, 5-0 |

The second bank of registers, known as the resolution bank (R-bank), has the following format.

|  |  |  |
| --- | --- | --- |
| 10, C-Predictor, 11-10 | 11, D-Predictor, 9-8 | New Program Counter, 7-0 |

### Registers

The BPT has several status flags and registers for storing various operational data.

|  |  |  |  |
| --- | --- | --- | --- |
| **Register** | **Name** | **Size** | **Notes** |
| R\_PC | Read PC | 8 | Used for querying the BPT for matches. |
| W\_PC | Write PC | 8 | Used for adding or updating entries into the BPT. |
| NPC | New PC | 8 | The computed EA for the branch, for use in later predictions. |
| TAKEN | Taken | 1 | Indicates whether the previous branch was taken or not, updates previous entries. |
| PPC | Predicted PC | 8 | The predicted PC. |
| SVI | Status Flags and Victim Index | 8 | SVI5 is the table full flag, used to indicate not to use the free index register. SVI4 indicates the previous free index is invalid and must be searched again. SVI3-0­ is the previous victim that was purged. This value is incremented before purging the entry. |
| FI | Free Index | 8 | The top four bits are for the instruction that just got added, and the bottom four bits for the instruction that just finished the decode phase. They must be kept in this format to allow for cooperation. |

### Operation

#### Querying for Existing Entries

Making a complete prediction can require multiple steps. The text in bold indicates the BPT has computed a result. After computing a result, no further action is taken.

1. Search for the current PC in the I-bank with the following method
   1. Start searching at 4\* IR1-0 in the I-bank, if the IR7-2 is equal to PC5-0, then the correct entry was found. Go to Step 2, otherwise continue.
   2. Check each entry in the table iteratively until the entire table is exhausted or a blank entry is detected. If the entry is found, then go to Step 2, otherwise continue.
   3. Since our entry does not exist, no prediction can be made. The BPT will predict **branch not taken**, as the PC has not been computed to predict otherwise. If a blank entry was found, then the index of the previous free entry is shifted to the bottom four bits and our new entry is added. If no blank entry was found, set the table full flag to indicate no more free entries.
2. The indexed position in the I-bank will be the corresponding position in the R-bank. Set the top four bits of the free index register to the current index.
3. The historical values bits, I-bank11-10, indicates which saturated predictor to examine. For example, if the history reads 00, then the history was two branches not taken. The corresponding saturated counter, in our case the A-Predictor, will either predict if the following branch is **taken, or not taken**.

#### Adding New Entries

Entries can only be added after the decode phase has finished executing. Only then will the instruction be identified as an R-Type (and therefore not requiring an entry in the table), or an I or J-Type.

1. If the status flag for an invalid free entry slot is set, go to Step 4. If the table full flag is set, then go to Step 5. Otherwise, the bottom four bits of the free entry index indicates where to store our new entry.
2. The history bits are shifted with the new value (taken or not taken), but no changes to the A-D predictors are made (since we have only one data point).
3. If the top four bits of the free entry index are the same as our current one, set a flag indicating the previous entry is invalid, otherwise clear the value. The entry is added, **stop execution**.
4. Our previous free entry index was invalid; iterate through the table searching for a new free position. Our new index is the empty entry, return to Step 2.
5. The table is full; increment the victim index and purge the entry. Our new entry is the current victim index, return to Step 2.

#### Updating Current Entries

Currently existing entries can be updated, but a data hazard can be present in that the index to update was purged by the decode phase in front of us. The process for updating an entry is detailed as follows:

1. The bottom four bits of the free index register contain the index to where our entry is located. If our entry no longer exists, go to Step 4. Otherwise, the current history data indicates which predictor to use.
2. If the branch was taken, increment the respective predictor, otherwise decrement.
3. Shift the current history data left, adding the results of whether the branch was taken.
4. Since our entry was purged, see Section *Adding New Entries*, and then go to Step 1.

# Register Transfer Language

The fetch phase is excluded so as to avoid unnecessary redundancy in the table. Instead, the fetch phase for all instructions is detailed after the following table. It is also assumed that any relevant data is automatically copied down the pipeline.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **OPC** | **DECODE** | **EXECUTE** | **MEMORY** | **WRITEBACK** |
| NOP | - | - | - | - |
| ADD | D/X\_RS ← F/D\_IRrs,  D/X\_RT ← REG[F/D\_IRrt],  D/X\_RD ← REG[F/D\_IRrd] | X/M­\_ALU ← D/X\_RT + D/X\_RS | - | REG[M/W\_RD] ← M/W\_ALU |
| ADDI | D/X\_RS ← F/D\_IRrs,  D/X\_RT ← REG[F/D\_IRrt],  D/X\_IMM ← F/D\_IRimm | X/M­\_ALU ← D/X\_RT + D/X\_IMM | - | REG[M/W\_RS] ← M/W\_ALU |
| AND | D/X\_RS ← F/D\_IRrs,  D/X\_RT ← REG[F/D\_IRrt],  D/X\_RD ← REG[F/D\_IRrd] | X/M­\_ALU ← D/X\_RT & D/X\_RS | - | REG[M/W\_RD] ← M/W\_ALU |
| OR | D/X\_RS ← F/D\_IRrs,  D/X\_RT ← REG[F/D\_IRrt],  D/X\_RD ← REG[F/D\_IRrd] | X/M­\_ALU ← D/X\_RT | D/X\_RS | - | REG[M/W\_RD] ← M/W\_ALU |
| XOR | D/X\_RS ← F/D\_IRrs,  D/X\_RT ← REG[F/D\_IRrt],  D/X\_RD ← REG[F/D\_IRrd] | X/M­\_ALU ← D/X\_RT ⊕ D/X\_RS | - | REG[M/W\_RD] ← M/W\_ALU |
| SLL | D/X\_RS ← F/D\_IRrs,  D/X\_RT ← REG[F/D\_IRrt],  D/X\_RD ← REG[F/D\_IRrd] | X/M­\_ALU ← D/X\_RT << D/X\_RS | - | REG[M/W\_RD] ← M/W\_ALU |
| SRL | D/X\_RS ← F/D\_IRrs,  D/X\_RT ← REG[F/D\_IRrt],  D/X\_RD ← REG[F/D\_IRrd] | X/M­\_ALU ← D/X\_RT >> D/X\_RS | - | REG[M/W\_RD] ← M/W\_ALU |
| SRA | D/X\_RS ← F/D\_IRrs,  D/X\_RT ← REG[F/D\_IRrt],  D/X\_RD ← REG[F/D\_IRrd] | X/M­\_ALU ← D/X\_RT >>a D/X\_RS | - | REG[M/W\_RD] ← M/W\_ALU |
| SB | D/X\_RS ← F/D\_IRrs,  D/X\_RT ← REG[F/D\_IRrt],  D/X\_IMM ← F/D\_IRimm | X/M­\_ALU ← D/X\_RT + D/X\_IMM | MEM[X/M\_ALU] ← X/M\_RS | - |
| LB | D/X\_RS ← F/D\_IRrs,  D/X\_RT ← REG[F/D\_IRrt],  D/X\_IMM ← F/D\_IRimm | X/M­\_ALU ← D/X\_RT + D/X\_IMM | M/W\_MDR ← MEM[X/M\_ALU] | M/W\_RS ← M/W\_MDR |
| SLT | D/X\_RS ← F/D\_IRrs,  D/X\_RT ← REG[F/D\_IRrt],  D/X\_RD ← REG[F/D\_IRrd] | X/M­\_ALU ← D/X\_RD < D/X\_RT | - | REG[M/W\_RS] ← M/W\_ALU |
| BEZ | BPTW­\_PC ← F/D\_PC,  BPTNPC ← EA,  BPTtaken ←RS == 0 | - | - | - |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **OPC** | **DECODE** | **EXECUTE** | **MEMORY** | **WRITEBACK** |
| BNE | BPTW­\_PC ← F/D\_PC,  BPTNPC ← EA,  BPTtaken ←RS != 0 | - | - | - |
| JUMP | BPTW­\_PC ← F/D\_PC,  BPTNPC ← EA,  BPTtaken ← 1 | - | - | - |
| HALT | - | - | - | - |
|  |  |  |  |  |

## Fetch Phase

The fetch phase is where the branch prediction takes over, and consists of two exclusive possibilities:

1. An R-Type instruction will not cause the branch predictor to engage beyond the identification step. Since all R-Type instructions do not modify the PC or cause any branches in executions, the branch predictor will never override the PC.

IRF/D ← MEM[PC]

PC ← PC + 4

1. In the event of successful identification in the BPT, then the following will happen.

BPTR\_PC ← PC

PC ← BPTPPC