Mehmet Arslan 2374411
Feyza Nur Takıl 2375780

Laboratory Project - Pipelined Processor with Hazard Unit and Branch Predictor

Introduction

In the project, the same pipelined processor defined in lectures is used with modifications.

Datapath Changes

Shifter is added to the datapath immediately before B port of ALU module. To be able to multiply imm24 by 4, the extender is modified. To make Register File write to the registers on the negative edge of the clock, the clock of Register File is inverted. For branch instructions, PC+4 is also carried through the pipeline. For BX instruction, selection for A3 between R14 and carried WA3 signal, WriteSrc signal is carried through the pipeline.

Regarding data processing instructions, the registers Rd and Rm are chosen as A1 and A2, respectively. Rd is then forwarded as the write address. Signals such as RD1, RD2, WA3D, and ExtImm are stored for use in the next cycle. The shifter, ALUSrc, and ALU signals are determined based on the instruction. RD2, ALUResult, and the write address for the Register File are also stored. In the subsequent cycle, during the memory stage, ReadData and ALUOut are captured, but the read data is not selected for the following cycle. During the writeback stage, when the RegWrite signal is high, the carried write address is connected to A3. At the falling edge of the cycle, the value of ALUOut is written to the register.

For memory instructions, Rn is selected as A1, and for STR, Rd is chosen as A2. RD1, RD2, and ExtImm signals are captured at the rising edge of the clock. Then, the ExtImm signal is chosen and added to the value of RD1. No shifting operation is performed for memory instructions. ALUResult and RD2 values are recorded. In the next cycle, the data at the address specified by ALUResult is read and captured for LDR. For STR, when the MemWrite signal is high, the RD2 value is written to the specified position, completing the STR operation. In the subsequent stage, the readData is written to the Rd register for the LDR operation.

In the case of B and BL instructions, R15 is selected for Register File, and the imm24 value is multiplied by 4. Then, in the next cycle, RD1 and ExtImm values are added, and the result is stored. During the memory stage, the memory is read, but the obtained data is not used. Then, in the writeback stage, when the PCSrc signal is high, the PC value will become ResultW in the next cycle. For the BL instruction, with RegWrite set to 1, the PC+4 value is written to the R14 register. The PC+4 value is the value forwarded during instruction decoding. As for BX, the Rm value is chosen as A2, and the RD2 value is stored. In the next cycle, the RD2 value is directly transferred to ALUResult and stored. Similar to data processing instructions, the memory is read, but its value is not used. Finally, in the writeback stage, the ALUResult is written to the PC register.

To implement move immediate, immediate bit is considered to implement data processing instructions. Immediate data processing operations implemented as follows. For the shift amount signal shamt, the rot signal is shifted by 1. ImmSrc signal is 00 to choose imm8 and ALUSrc selects ExtImm signal. Shift operation for shifter is chosen as ROR.

Mehmet Arslan 2374411
Feyza Nur Takıl 2375780

Similar structures mentioned in the lectures are added to the system for forwarding. For mispredictions and LDRstall, flush and stall signals are added. Flush and stall signals are applied not only to datapath registers but also control signals.

To select PC addresses between ALUResultE, BTA, PCPlus4, carriedPCPlus4, MUX before PC register is implemented. When a branch is needed to be taken but not taken during fetch, the PC address should be ALUResultE. On the other hand, if a branch shouldn't be taken but taken during fetch, the PC address should be PC+4 carried through the pipeline. If the branch predictor says that branch will be taken, MUX selects BTA, otherwise selects PC+4. Priority of information comes from execution higher than branch predictor information.

For branch operations, signals related to the branch are not carried through pipeline after the execute stage since there is no operation related to pipeline for memory and writeback stages. Therefore, connection between writeback output and PC register is removed.

Hazard Unit

For mispredictions and LDRstall, flush and stall signals are added. Flush and stall signals are applied not only to datapath registers but also control signals.

Data Hazard Handling

Similar structures mentioned in the lectures are added to the system for forwarding. RA1, RA2 and WA3 signals are carried through the pipeline. When registers in the execute stage match with register names in memory or writeback stage, forwarding takes place. However, for some instructions such as move immediate instruction, RA1, RA2 or WA3 might not be valid and mismatches might take place. So, validity signals are added to the system. Moreover, for flush signals, validity signals are also flushed to prevent undesired matches with zeros of flush and R0 register.

Stall only takes place for LDRstall since 4 cycle stalls for R15 write instruction won't be used. Forwarding signals are the same as in lectures except validity of signals are checked.

Control Hazard Handling

For register branch instructions, PC + 4 is taken and if in the execute stage, it is realized that branch will be taken, decode and execute stages are flushed and the carried PC + 4 signal is selected as new PC value.

For branch instructions, the branch predictor determines whether or not branch should be taken. And in the execute stage, the controller realizes if the prediction is right or wrong. Taken signal of the branch predictor is BranchPredicted and this signal is carried through the pipeline. Flush mechanism can be seen below.

Mehmet Arslan 2374411
Feyza Nur Takıl 2375780

Table 1. Flush Mechanism for branch instructions

BranchTakenE (comes from datapath)	**BranchPredictedE** (comes from predictor)	**Operation**
0	0	Prediction is correct. No flush. Next PC depends on Branch predictor.
0	1	Flush D, E. Next PC value is PC+4 value carried with instruction.
1	0	Flush D, E. Next PC value is output of ALU.
1	1	Prediction is correct. No flush. Next PC depends on Branch predictor.

Branch Predictor

The Branch Predictor is composed of three primary components: the Branch Target Buffer (BTB), Global History Register (GHR), and Pattern History Table (PHT). These components work collaboratively to forecast the outcomes of branch instructions within a program. The Branch Predictor also includes a RESET input, which is utilized to reset the entire branch predictor system. It should be acknowledged that the branch predictor has the potential to make erroneous predictions, causing the computer to branch erroneously. When such situations occur, the Fetch and Decode stages must be cleared, and the program counter (PC) should be incremented by 4 after fetching the branch instruction. Consequently, apart from the three modules mentioned earlier, the branch predictor will also transmit the value of PC + 4 from the branch instructions to the Execute stage for potential utilization.

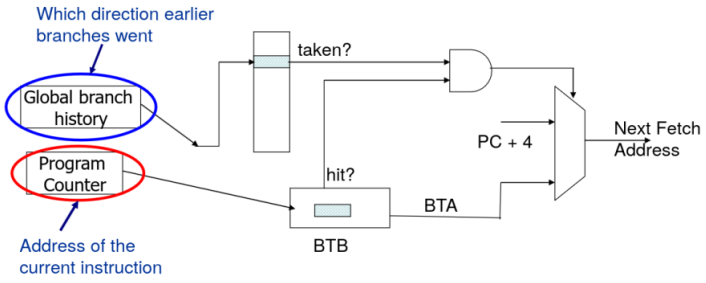


Figure 1: Overall Branch Predictor Layout

Mehmet Arslan 2374411
Feyza Nur Takıl 2375780

Branch Target Buffer

The given code is a module called BTB (Branch Target Buffer) that implements a branch predictor. It takes several inputs, including the clock signal, reset signal, program counter, ALU branch address, and pc + 4. Additionally, it receives branchTakenE and branchPredictedE signals.

The module has two primary outputs: BTA (Branch Target Address) and hit. BTA is a 32-bit output representing the target address for a branch instruction, while hit indicates whether there was a hit in the BTB cache. Inside the module, there are three 40-bit registers: cache0, cache1, and cache2, which store branch instructions and their corresponding target addresses.

The module has an always block triggered by the negative edge of the clock signal. Within this block, the module handles different scenarios based on the inputs. If the reset signal is active (reset == 1), the module resets the cache registers, BTA, and hit. Otherwise, it performs the following operations:

1. It checks if the (pc), obtained by subtracting 4 from (pc + 4), is not present in the cache (cache0, cache1, or cache2) and if the branch is taken (branchTakenE == 1) and not predicted (branchPredictedE == 0). If all these conditions are met, it updates the cache registers with the new branch address, updates BTA and hit to their default values.
2. It checks if the pc matches cache0 (the first 8 bits of cache0). If there is a match, it updates BTA with the corresponding target address from cache0 (the last 32 bits of cache0), sets hit to 1, and preserves the values of cache0, cache1, and cache2.
3. It checks if the pc matches cache1 (the first 8 bits of cache1). If there is a match, it updates BTA with the corresponding target address from cache1(the last 32 bits of cache1), sets hit to 1, and performs swapping operations between cache registers to prioritize cache1 for the next iteration.
4. It checks if the pc matches cache2 (the first 8 bits of cache2). If there is a match, it updates BTA with the corresponding target address from cache2 (the last 32 bits of cache2) sets hit to 1, and performs swapping operations between cache registers to prioritize cache2 for the next iteration.
5. If none of the above conditions are satisfied, it sets BTA to 0, hit to 0, and preserves the values of cache0, cache1, and cache2.

The module provides a mechanism to predict branch targets based on previous branch instructions stored in the cache. It checks the cache for a match with the current program counter (pc) and updates the output BTA accordingly. The hit signal indicates whether there was a successful match in the cache.

Mehmet Arslan 2374411
Feyza Nur Takıl 2375780

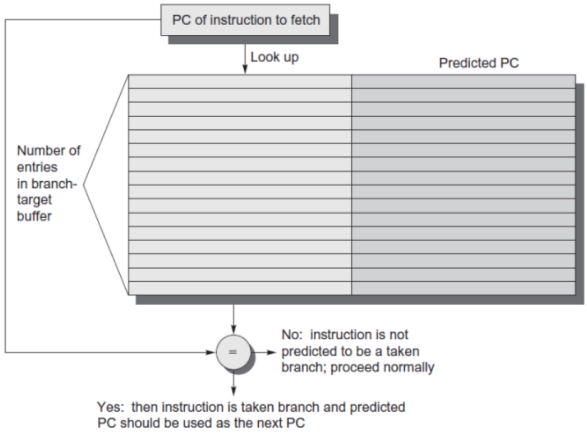


Figure 2: Branch Target Buffer Layout

Global History Register (GHR)

The GHR functions as a basic shift left register that retains a value of 1 when a branch is taken, as triggered by the BranchTakenE signal. In the case of a branch not being taken, it receives a 0 signal from BranchTakenE, based on the result from the execute stage. The GHR transmits its 3-bit branch history output to the PHT synchronously. The GHR operates based on the signals it receives, particularly the BranchE signal, to determine whether the current branch condition should be stored as a "1 (taken)" or "0 (not taken)" value. The following describes the operational principle of the GHR:

Table 2: Operation of the GHR

Reset	Shift (BranchE)	Operation
1	X	DATA ← 0
0	0	DATA ← DATA
0	1	DATA ←{ DATA << 1, In (BranchTakenE) }

Pattern History Table (PHT)

The Pattern History Table (PHT) uses the 3-bit output history of the Global History Register (GHR) to make predictions for branch instructions. Based on past data, the PHT analyzes and predicts whether the branch will be taken or not. With a 3-bit width, the PHT consists of 8 elements, each being 1 bit wide. A value of 1 in an element indicates that the branch was taken according to the corresponding history recorded in the GHR, while a value of 0 indicates that the branch was not taken. Therefore, the PHT entries serve as 1-bit predictors as output asynchronously based on the most recent history from the GHR. If reset signal is given to PHT, all PHT predictions are updated to not-taken (0) state synchronously.

Table3: Pattern History Table Implementation Example

GHR 3-bit Branch History	Branch Prediction
000	0 (NT)
001	0 (NT)
010	0 (NT)
011	1 (T)
100	0 (NT)
101	1 (T)
110	0 (NT)
111	0 (NT)

Initially, all PHT indexes are set to the "not taken" (NT) state, represented by 0, with the PHT table updated whenever a wrong prediction occurs, using two signals to identify such inaccuracies. When the system identifies an instruction as a branch signal during execution, it produces a branchTakenE signal with a value of 1. If the branchPredictedE signal, generated in the fetch stage, is 0, it implies that the instruction is not a branch. These two signals enable us to anticipate a not-taken branch instruction, even though the execution stage confirms it as a branch instruction. In these cases, the PHT index is reversed synchronously due to incorrectly predicting a branch to be not taken when it should have been taken.

A similar problem arises in the opposite scenario. If an instruction is recognized as a not-taken branch signal during execution, a branchTakenE signal with a value of 0 is generated. If the branchPredictedE signal from the fetch stage is present, it indicates that the instruction is a branch. Despite it being a not-taken branch instruction confirmed during the execution stage, these two signals can mispredict it as a taken branch instruction. In these situations, the PHT index is flipped synchronously again because it erroneously predicted a branch to be taken when it should have not been taken.

Table 4: PHT Update Cases

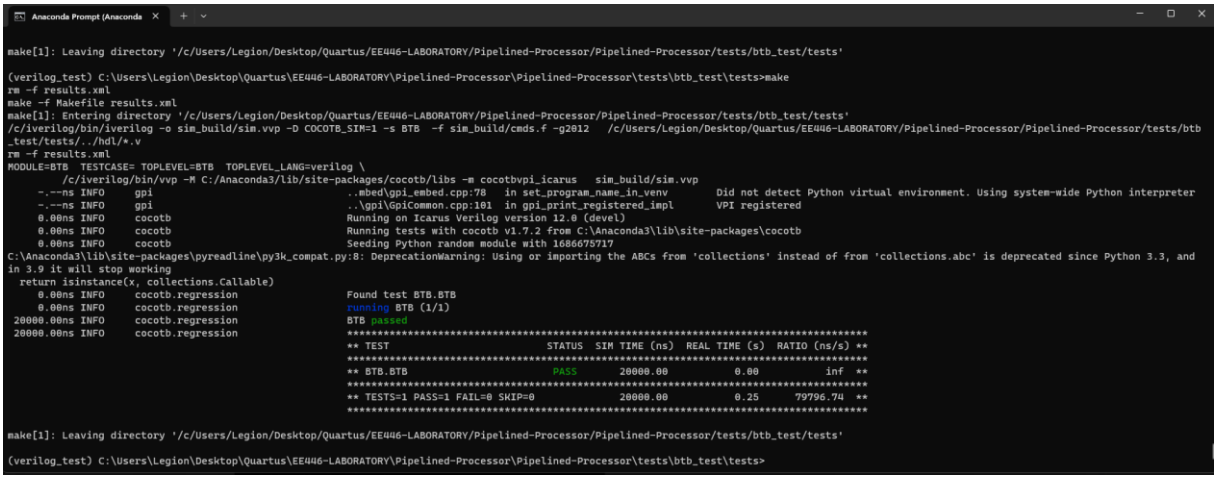
branchTakenE	branchPredictedE	Prediction	Action
0	0	Correct	No Action
0	1	Wrong	Flip PHT Branch Prediction
1	0	Wrong	Flip PHT Branch Prediction
1	1	Correct	No Action

Testbench

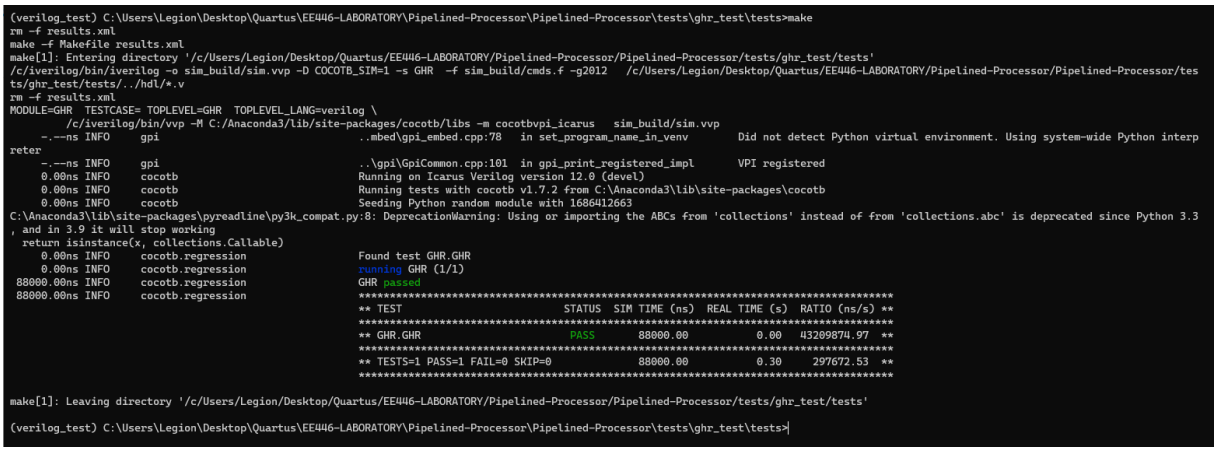
Before integrating hazard unit into the system, we conducted tests of it. Unfortunately, we were unable to test it again after assembling the complete system using cocotb. Nevertheless, we performed waveform tests to extract and observe signals and their changes clearly, ensuring that our hazard unit is fully functional. In addition, we utilized waveform tests to examine the signals in the branch predictor since cocotb does not allow us to observe waves. Nonetheless, we created cocotb tests for all modules within the branch predictor. Subsequently, we thoroughly tested the entire system on an FPGA, confirming the perfect and accurate performance of the branch predictor.

Mehmet Arslan 2374411
Feyza Nur Takıl 2375780

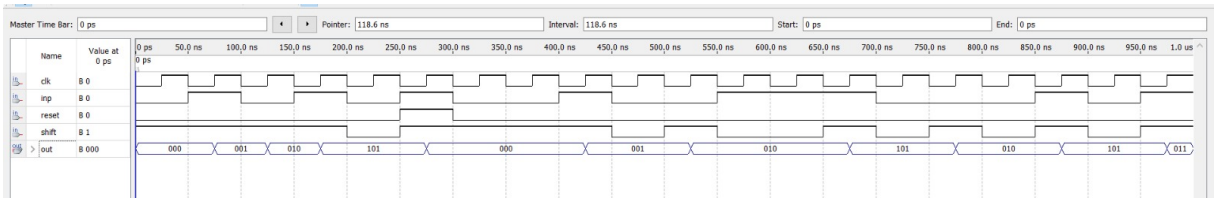
BTB Cocotb Test:



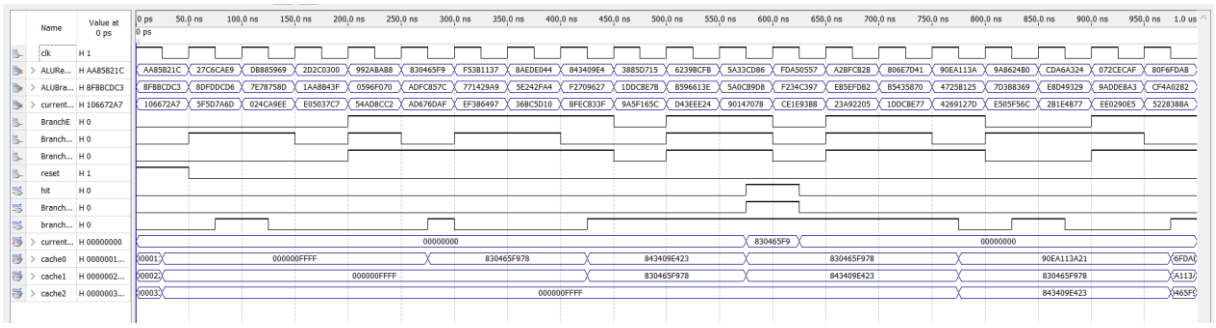
GHR Cocotb Test:



GHR Waveform Test:



Branch Predictor Test:



Datapath Tests:

Mehmet Arslan 2374411
Feyza Nur Takıl 2375780

```
******traverse completed**********
subroutine completed
1936000.00ns INFO     cocotb.regression                 parity_test ←[32mpassed←[49m←[39m
1936000.00ns INFO     cocotb.regression                 ********************************************************************************
                                                        ** TEST                          STATUS  SIM TIME (ns)  REAL TIME (s)  RATIO (ns/s) **
                                                        ********************************************************************************
                                                        ** parity_test.parity_test       ←[32m PASS ←[49m←[39m    1936000.00           0.14   13706181.73  **
                                                        ********************************************************************************
                                                        ** TESTS=1 PASS=1 FAIL=0 SKIP=0           1936000.00           3.44     563045.04  **
                                                        ********************************************************************************
```

```
r0= 0 dut.RD1.value= 0 , r1= 0b11111111111111110000110000001110 ,dut.RD2.value= 0b11111111111111110000110000001110
156000.00ns INFO     cocotb.regression                  complement_test ←[32mpassed←[49m←[39m
156000.00ns INFO     cocotb.regression                  ****************************************************************
**************************                              ** TEST                                STATUS  SIM TIME (ns)  REAL
TIME (s)  RATIO (ns/s) **
                                                        ****************************************************************
**************************                              ** complement_test.complement_test  ←[32m PASS ←[49m←[39m     15
6000.00           0.02    9056338.83  **
                                                        ****************************************************************
**************************                              ** TESTS=1 PASS=1 FAIL=0 SKIP=0                156000.00
   0.35     449037.21  **
                                                        ****************************************************************
**************************
```