Khaled Ajaj
EE126
Dr. Hempstead
12/11/2022
Using Three late tokens

Lab 6: Overcoming control hazards by resolving conditional/unconditional branches in ID and using flushing

Introduction:

The purpose of this lab was to rework the ARM LEGv8 5-stage pipelined processor designed in previous labs to implement static branch prediction. The goal was to implement control hazards and instruction flushing to accomplish branch prediction.

Experimental Setup:

All of the components used in this lab have already been designed and tested in previous labs, however some components were moved and additional logic was added to support flushing and branch prediction (figure 1).

The static hazard detection in this processor was intended to predict not-taken for every branch. The branch detection hardware such as the branch address adder, shift left, and zero flag were moved to be in the decode stage (ID). This way, branch prediction could occur fully in the ID stage, hence only needing to flush the IF/ID register in the case of a branch misprediciton. The zero flag was re-implemented by using a comparator on the data output of the second register.

In order to flush the IF/ID register, it was modified to have another input (flush enable bit). The new input flushed the register by resetting all the bits in the register to zeros.

The logic for the flush signal was quite simple. Since the processor predicted that the branch was not taken every time, then the register only needed to flush when the branch was indeed taken, therefore a multiplexer was used to pull the flush signal to high only when a branch was taken and the write enable signal of the program counter was high, and pulled to low otherwise.

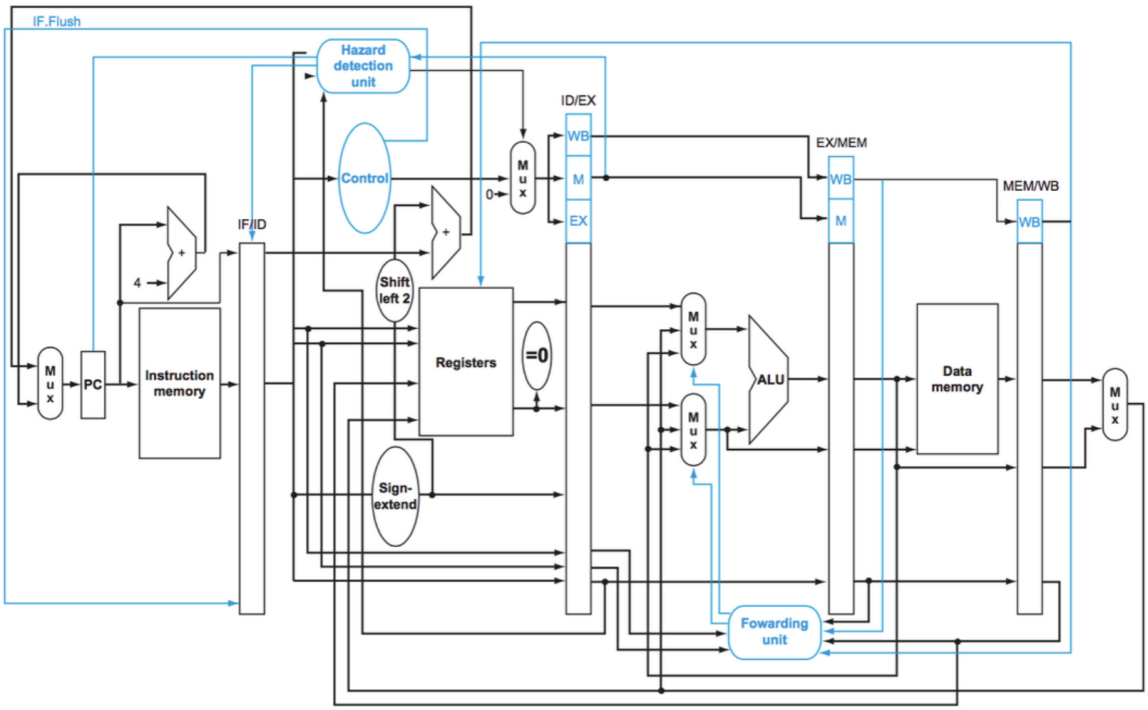

Figure 1. 5-stage pipelined processor design with hazard detection and data forwarding.

Testing and Simulation:

The complete processor was tested by running the provided test assembly code. The test program was run to ensure the functionality of the processor (figure 3). The program run with the intialized register and memory values shown in figure 2. The expected stages of each instruction at each clock cycle (Table 1) were examined against the waveform of the program to ensure functionality, and the program appeared to be working as expected (figures 4-5). In figure 4, it can be seen that the CBZ instruction entered its ID stage in the third cycle, but since the first instruction had not entered its writeback stage yet, and there was no forwarding added to the ID stage, then the value of X23 appeared as zero. Hence the flush signal was pulled high, and the branch was taken from PC = 8 to PC = 8+5<<2 = 28. The first instruction entered its writeback stage in the 5th clock cycle, and X23 was updated to 0x1. The B 2 instruction was fetched in the 6th cycle, therefore the branch was resolved and taken in the ID stage (cycle 7), therefore flushing the ADD X19, X19, X19 instruction. The branch was taken from PC = 32 to PC = 32+2<<2 = 40.

Register Values

```
$X9  = 0h1     $X19 = 0h1
$X10 = 0h2     $X20 = 0h2
$X11 = 0h4
$X21 = 0x000000008BADF00D
$X12 = 0h8
$X22 = 0x000000008BADF00D
$X23 = 0x0000000000000000
$X24 = 0x0000000000000000
```

DMEM Contents

```
DMEM(0x0)  = 0x00 00 00 09
DMEM(0x8)  = 0x00 00 00 08
DMEM(0x10)  = 0x00 00 00 07
DMEM(0x18) = 0x00 00 00 06
```

Figure 2. Initialized memory and register values.

Test Program (IMEM contents): partial machine code | raw assembly code

```
SUB X23, X20, X19    11001011000100110000001010010111
CBZ X23, 5           10110100000000000000000010110111
ADD X9,  X9, X9      10001011000010010000000100101001
SUB X24, X22, X21    11001011000101010000001011011000
CBZ X24, 3           10110100000000000000000001111000
ADD X10, X10, X10    10001011000010100000000101001010
ADD X11, X11, X11    10001011000010110000000101101011
ADD X12, X12, X12    10001011000011000000000110001100
B 2                  00010100000000000000000000000010
ADD X19, X19, X19    10001011000100110000001001110011
ADD X20, X20, X20    10001011000101000000001010010100
nop
nop
nop
nop
```

Figure 3. Test program instructions

Clock Cycles														
	1	2	3	4	5	6	7	8	9	10	11	12	13	14
SUB X23, X20, X19	IF	ID	EX	MEM	WB									
CBZ X23, 5		IF	ID	EX	MEM	WB								
ADD X9, X9, X9			IF	Flush										
SUB X24, X22, X21														
CBZ X24, 3														
ADD X10, X10, X10														
ADD X11, X11, X11				IF	ID	EX	MEM	WB						
ADD X12, X12, X12					IF	ID	EX	MEM	WB					
B 2						IF	ID	EX	MEM	WB				
ADD X19, X19, X19							IF	Flush						
ADD X20, X20, X20								IF	ID	EX	MEM	WB		

Table 1. Processor program execution stages

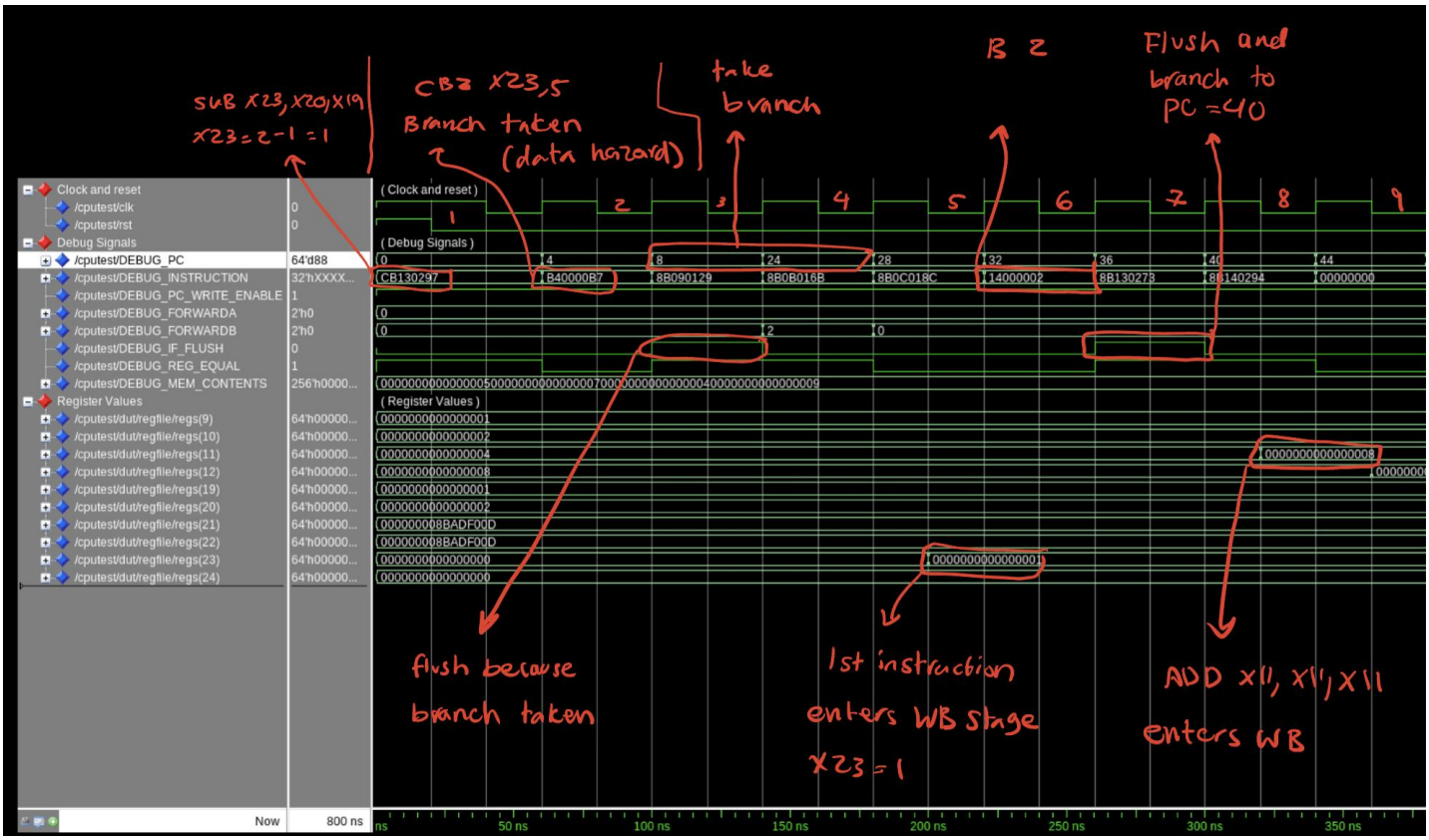

Figure 4. First part of program execution

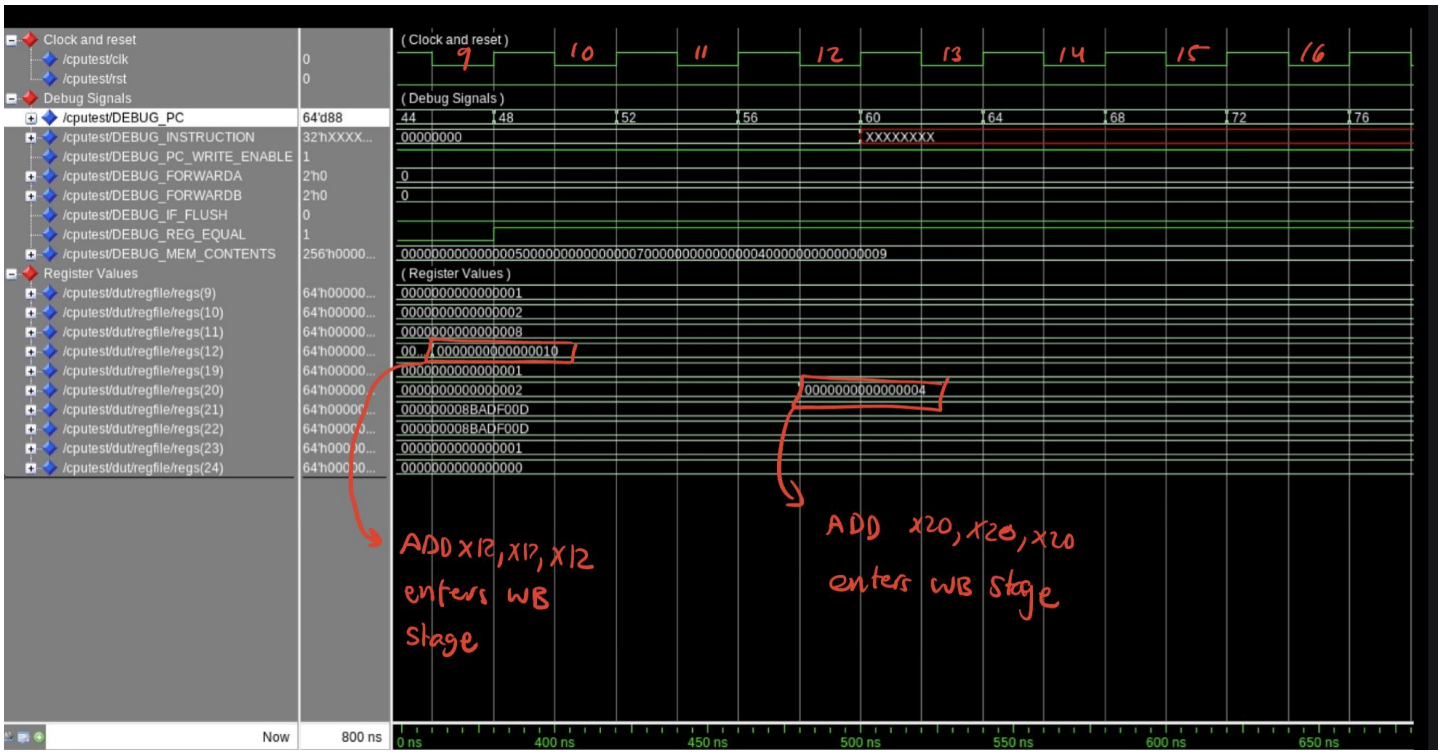

Figure 5. Second part of program execution

From figure 5, it is apparent that ADD X12, X12, X12 entered its writeback stage in cycle 9, which updated X12 to have a value of 0x10. Lastly, in cycle 12, the final instruction entered its witeback stage, updating the value of X20 to 0x4. The final values of the registers and memory were examined (figure 6), and they appeared to match expected theoretical values (taking into account the lack of forwarding into the ID stage). The memory data never changed during program execution as no memory instructions were used in this program.

/cputest/DEBUG_MEM_CONTENTS	256'h0000000000000005000000000000007000000000000000400000000000000009
Register Values	
/cputest/dut/regfile/regs(9)	64'h0000000000000001
/cputest/dut/regfile/regs(10)	64'h0000000000000002
/cputest/dut/regfile/regs(11)	64'h0000000000000008
/cputest/dut/regfile/regs(12)	64'h0000000000000010
/cputest/dut/regfile/regs(19)	64'h0000000000000001
/cputest/dut/regfile/regs(20)	64'h0000000000000004
/cputest/dut/regfile/regs(21)	64'h000000008BADF00D
/cputest/dut/regfile/regs(22)	64'h000000008BADF00D
/cputest/dut/regfile/regs(23)	64'h0000000000000001
/cputest/dut/regfile/regs(24)	64'h0000000000000000

Figure 6. Final register and memory values after program execution.

Conclusion:

The goal of this lab was to implement data control hazard detection capability to a 5-stage pipelined processor. The additional functionality was correctly implemented with the processor predicting branches to be not taken every time, and flushing the instructions in the pipeline when the branch was indeed taken.