Report: Lab5

20190533 Hyemin Lee
20190235 Gangtae Park

1. Introduction

이번 Lab5는 Lab3의 Single-cycle CPU와 Lab4의 Multi-cycle CPU의 cycle 수를 더욱 감소시킬 수 있는 Pipeline CPU를 verilog를 이용해서 구현하는 Lab이다. Pipeline CPU를 구현하기 위해서는 IF, ID, EX, MEM, WB 각각의 stage들 사이에 pipeline register를 선언해서 한 cycle 내에 여러 stage들을 동시에 실행하게 하였다. 또한, jump나 branch instruction에서의 효율을 증가시키기 위해서 BTB를 이용해 branch prediction을 구현했고 2-bit saturation counter 방법을 적용시켰다.

2. Design

우리가 구현한 Pipeline CPU는 clock의 positive edge를 만날 때마다 각 stage에 있는 instruction들이 다음 stage로 넘어간다. 각 pipeline register들의 변수 명에 FD(IF/ID pipeline register), DE(ID/EX pipeline register), EM(EX/MEM pipeline register), MW(MEM/WB pipeline register) 태그를 붙여서 어떤 stage에서 필요한 register인지 보기 쉽게 정리하였다.

	REGISTER									
	I_MEM_DI	PC	jumped	RF_RA1	RF_RA2	RF_RD1	RF_RD2	imm	ALU_out	WA1
IF	produce	produce								
	FD_IR	FD_PC								
ID	use			produce	produce	produce	produce	produce		produce
	DE_IR	DE_PC		DE_RA1	DE_RA2	DE_RD1	DE_RD2	DE_imm		DE_WA
EX	(forwarding use)	use	produced	(forwarding use)	(forwarding use)	use	use	use	produce/use	
	EM_IR	EM_PC	EM_jumped		EM_RA2		EM_RD2		EM_ALU_out	EM_WA
MEM	(forwarding use)				(forwarding use)		use		use	
	MW_IR	MW_PC	MW_jumped						MW_ALU_out	MW_WA
WB	use	use	use						use	use

뿐만 아니라, control flow에서는 각 MUX들의 값들을 결정해주는 변수들과 RF_WE, D_MEM_WEN, D_MEM_BE를 전달해준다. 이 값들도 마찬가지로 pipeline register를 통해서 MUX가 실제로 작동하는 stage까지 넘겨주어야 한다.

CONTROL FLOW								
RF_WE	D_MEM_WEN	D_MEM_BE	ALU_cont	m_imm	m_ALU_in1	m_ALU_in2	m_WD	m_target
produce	produce	produce	produce	produce/ use	produce	produce	produce	produce
DE_WE	DE_WEN	DE_BE	DE_ALU_cont		DE_ALU_in1	DE_ALU_in2	DE_WD	DE_target
			use		use	use		use
EM_WE	EM_WEN	EM_BE					EM_WD	
	use	use						
MW_WE							MW_WD	
use							use	

Forwarding unit은 data forwarding이 필요한 곳에서 이용된다. ForwardA와 ForwardB는 EX stage에서의 RF_RD들이 ID stage의 RF_RD를 받을지 혹은 이전 instruction에서 접근하는 register의 값을 필요하기 때문에 그 값을 forwarding할 지 여부를 결정한다. ForwardC와 ForwardD는 ALU의 input을 EX stage의 RF_RD로 할지 혹은 MEM stage의 값을 forwarding 할 지, WB stage의 값을 forwarding 할 지 여부를 결정한다. 마지막으로 ForwardE는 MEM stage의 D_MEM_DOUT을 MEM stage의 RF_RD2로 할지 혹은 WB stage의 값을 forwarding 할 지 여부를 결정한다.

3. Implementation

각 instruction들 마다 각 stage에서 하는 역할이 정해져있다. 이를 표로 정리하면 다음과 같다.

stage	JAL	JALR	Branch	LW	SW	I-type	R-type
IF	PC = next_PC next_PC = PC + 4						
ID	imm 설정 RF_WA1 설정	imm 설정 RF_RA1 설정 RF_WA1 설정	imm 설정 RF_RA1,2 설정 RF_WA1 설정	imm 설정 RF_RA1 설정 RF_WA1 설정	imm 설정 RF_RA1,2 설정 RF_WA1 설정	imm 설정 RF_RA1 설정 RF_WA1 설정	RF_RA1,2 설정 RF_WA1 설정
EX	ALU_in1 = PC ALU_in2 = imm target 설정	ALU_in1 = RF_RD1 ALU_in2 = imm	ALU_in1 = RF_RD1 ALU_in2 = RF_RD2	ALU_in1 = RF_RD1 ALU_in2 = imm			ALU_in1 = RF_RD1 ALU_in2 = RF_RD2

		target 설정	target 설정				
MEM				D_MEM_ADDR = ALU_out	D_MEM_ADDR = ALU_out D_MEM_DOUT = RF_RD2		
WB	RF_WD = PC + 4		RF_WD = ALU_out	RF_WD = D_MEM_DI	RF_WD = ALU_out		

4. Evaluation

ModelSim을 이용해 TB_RISCV_inst, TB_RISCV_forloop, 그리고 TB_RISCV_sort testbench들을 시뮬레이션하였다. 그 결과, TB_RSICV_inst의 경우 모든 testcase(26개)를 통과하였고 총 33cycle의 결과를 얻었다.

```
# Test #   25 has been passed
# Test #   26 has been passed
# Finish:         33 cycle
# Success.
# ** Note: $finish    : E:/OneDrive - kaist.ac
.kr/21 Fall EE312/Lab5-2/testbench/TB_RISCV_in
st.v(243)
#    Time: 445 ns  Iteration: 1  Instance: /TB
_RISCV_inst
```

TB_RISCV_forloop의 경우 모든 testcase(17개)를 통과하였고 총 103cycle의 결과를 얻었다.

```
# Test #   16 has been passed
# Test #   17 has been passed
# Finish:        103 cycle
# Success.
# ** Note: $finish    : E:/OneDrive - kaist.ac
.kr/21 Fall EE312/Lab5-2/testbench/TB_RISCV_fo
rloop.v(232)
#    Time: 1145 ns  Iteration: 1  Instance: /T
B_RISCV_forloop
```

TB_RISCV_sort의 경우 또한 마찬가지로 모든 testcase(40개)를 통과하였고 총 14134cycle의 결과를 얻었다. 모든 testbench에 대해서 testcase를 모두 통과하였음을 알 수 있다.

```
# Test #    39 has been passed
# Test #    40 has been passed
# Finish:       14134 cycle
# Success.
# ** Note: $finish    : E:/OneDrive - kaist.ac
.kr/21 Fall EE312/Lab5-2/testbench/TB_RISCV_so
rt.v(257)
#    Time: 141455 ns  Iteration: 1  Instance:
/TB_RISCV_sort
```

5. Discussion

기본적으로 코딩을 하는데에 시간이 오래걸렸다. 과제를 함에 있어 약 42시간이 소요되었고 중간에 한 번 코드를 새로 작성하기 시작하기도 하였다. 중간중간에 여러가지 어려움이 있었지만 가장 발견하기 어려웠던 오류는 jump와 load delay slot이 동시에 작동하여 문제가 발생하는 것을 알아차리는데 어려움이 있었다. load delay slot이 발생했을 때 instruction이 jump 혹은 branch 이면 PC를 업데이트 하는 것으로 간단하게 해결 할 수 있었지만 wave 형태만 보고는 이 문제점을 알아차리기 힘들었던 것 같다.

6. Conclusion

Lab5는 verilog을 이용해 Pipeline CPU를 구현하는 랩이였다. 이번 랩을 통해서 Multi-cycle에 비해서 Pipeline CPU가 cycle 수를 약 1/3로 줄여주기 때문에 얼마나 더 효율적인지 알 수 있었다. 뿐만 아니라, BTB를 구현하기 전의 sort testbench의 cycle수는 14732 cycle이였고 BTB를 구현한 후에는 14134로, branch prediction을 통해서 cycle 수를 감소시킬 수 있다는 점도 확인 할 수 있었다. Lab5를 통해서 pipeline CPU의 전체적인 흐름을 이해하는 것에 큰 도움이 되었다.