

Lab4-1 - Pipelined CPU(without control flow instructions)

team21: 20200001조은국, 20200431박현빈

2022년 5월 3일
(CSED 311)

Introduction

본 Lab은 pipelined RISC-V CPU를 구현하는 프로그램을 제작하는 과제이다. 모든 instruction은 1 cycle에 1개의 stage를 거치므로, 1 cycle에 서로 다른 5개 instruction의 write-back, memory access, execute, instruction decode and operand fetch, instruction fetch가 이루어지도록 해 throughput을 높인 pipelined cpu를 구현한다. control flow instruction은 고려하지 않는다.

Design

Textbook에 나오는 pipelined cpu에 대한 datapath와 control flow를 참조하고, 이에 필요한 register와 wire, module 등을 verilog로 구현하는 방식으로 design하였다.

PC (control.v)

input: next_pc

output: current_pc

다음 instruction의 address를 cpu의 계산 결과로부터 입력 받아 출력 한다.

clock synchronous

instruction Memory(memory.v)

input: address

output: Memory Data

pc를 address로 받아 그에 해당하는 instruction을 출력한다.

control에 따라 address에 해당하는 data memory의 값을 출력하거나, 해당 address의 memory에 특정 register의 값을 write한다.

* IF 담당

data Memory(memory.v)

input: address, din

output: Memory Data

control에 따라 address에 해당하는 data memory의 값을 출력하거나, memory의 해당 address에 din의 값을 저장한다.

*MEM 담당

Register(RegisterFile.v)

input: register 1,2, register direction, rd_din / write_enable

output: rs1_dout, rs2_dout

입력으로 받은 register에 해당하는 값을 출력하거나, control에 따라 register에 입력으로 받은 값을 write한다.

*instruction decode stage, write back stage 담당

ALU control(operator.v)

input: part_of_inst

output: alu_op

instruction의 일부를 입력으로 받아 alu에서 수행해야 하는 연산에 대한 control을 출력한다.

Clock Asynchronous

ALU(operator.v)

input: alu_op, alu_in_1, alu_in_2

output: alu_result, alu_bcond

control에 따라 입력받은 두 값에 대한 연산을 수행하고, 연산 결과를 출력한다.

*execute stage 담당

MUX(operator.v)

input: control, input1, input2

output: out

control에 따라 입력받은 두 값 중 하나만을 출력한다.

mux2bit(operator.v)

input: control, input1, input2, input3, input4

output: out

control에 따라 입력받은 4가지 값 중 하나만을 출력한다.

Immediate generator(control.v)

input: instruction[31:0]

output: immediate value

instruction에서 immediate부분을 도출하고, sign-extension하여 반환한다.

clock asynchronous

*ID stage 담당

Control Unit(control.v)

input: Opcode

output: 각각의 control signal bit (0 or 1)

opcode에 따라 여러 모듈(MUX, reg, mem 등)에 wire되는 control bit의 값을 출력한다.

clock asynchronous

*instruction decode stage 담당

Hazard Detection Unit(operator.v)

input: IF_ID_inst_out, ID_EX_mem_read_out, ID_EX_rd_out

output: HDU_out

IF_ID의 instruction으로 부터 얻은 rs1 또는 rs2가 ID_EX의 rd와 같고, ID_EX_가 mem_read를 한다면 load instruction이므로 HDU_out이 1이 되어 stall한다.

ecall instruction이 fetch되었을 경우 3 cycle동안 HDU_out이 1이 되어 stall한다.

Forwarding unit(operator.v)

input: ID_EX_rs1, ID_EX_rs2, EX_MEM_rd, MEM_WB_rd, IF_ID_inst_out, EX_MEM_reg_write, MEM_WB_reg_write, alu_src

output: ForwardingA, ForwardingB, ForwardingC, ForwardingD, ForwardingE

MEM stage에서 EX stage로 forward 해야하는지, WB stage에서 EX stage로 forward해야하는지에 대한 control bit를 ForwardingA와 ForwardingB, Forwarding E로

출력 한다.

우리가 구현하는 middle of clock cycle에 register file로 data를 write하지 않기 때문에, WB stage에서 ID로 forwarding을 해야 한다. 이를 위한 control을 ForwardingC, ForwardingD로 출력 한다.

Implementation

Forwarding_unit

- WB stage에서 EX stage로

$(ID_EX_rs1 \neq 0) \&\& (ID_EX_rs1 == MEM_WB_rd) \&\& MEM_WB_reg_write$

EX stage의 rs1과 WB stage의 rd가 같고, WB stage의 reg_write가 1일 때 forwarding 한다.

- MEM stage에서 EX stage로

$(ID_EX_rs1 \neq 0) \&\& (ID_EX_rs1 == EX_MEM_rd) \&\& EX_MEM_reg_write$

EX stage의 rs1과 MEM stage의 rd가 같고, WB stage의 reg_write가 1일 때 forwarding하도록 한다. 1번의 조건의 충족되었어도 2번의 조건이 충족되면 MEM stage에서 EX stage로 forwarding한다.

rs2에 대해서는 data memory의 din과 alu의 input2의 forwarding을 판단할 때 alu_src를 통해 차이를 둔다.

- WB stage에서 ID stage로

$(IF_ID_rs1 \neq 0) \&\& (IF_ID_rs1 == MEM_WB_rd) \&\& MEM_WB_reg_write$

ID stage의 rs1과 WB stage의 rd가 같고, WB stage의 reg_write가 1일 때 forwarding하도록 한다.

IF stage

PC

Current PC를 next_PC로, posedge Clock synchronous하게 update한다.

reset == 1일 경우 0으로 초기화

add_4

next_PC가 될 PC+4를 출력한다.

instruction Memory

input으로 받은 address의 instruction을 output으로 내보낸다.

read: Clock asynchronous

ID stage

Register

input에 해당하는 register 값을 반환하거나, write_enable에 따라 rd(가 0이 아닐 때)에 저장한다.

Read는 Clock asynchronous, Write는 posedge에 따라 Clock synchronous

MUX rs1

is_ecall에 따라 register의 rs1에 일반적인 instruction의 rs1이나 x17으로 assign되도록 한다.

MUX0, MUX1

ForwardC, ForwardD에 따라 WB stage에서 ID stage의 register value에 대해 forwarding을 하거나 하지 않는다.

MUX imm_detection

control에 따라 ALU의 두번째 input으로 register 값 또는 immediate value가 되도록 조절한다.

Control Unit

opcode에 따라 Control bit를 제어 한다.

Hazard detection Unit

IF_ID의 instruction으로 부터 얻은 rs1 또는 rs2가 ID_EX의 rd와 같고, ID_EX_가 mem_read를 한다면 EX stage에서 실행되는 instruction이 load instruction이므로 HDU_out이 1이 되어 stall한다.

ecall instruction이 fetch되었을 경우 3 cycle동안 HDU_out이 1이 되어 stall하고, x17의 값이 write back된 뒤에 x17의 값을 읽도록 한다.

ALU Control Unit

input으로 들어오는 ALUOP과 opcode, funct7, funct3에 따라 알맞은 alu_op을 output으로 출력한다. (clock asynchronous)

Immediate generator

if문으로 Opcode에 따라 immediate value part를 찾는다.

sign extension을 위해, 최고차항의 비트를 반복하여 32 bit를 채운다. 가령 12 bit imm_value의 최고차항이 1인 경우, 앞의 20 bit를 1로 채운다.

Clock asynchronous

EX stage

ALU

alu_op에 따라 and, sub, or, xor, sll, srl을 계산하여 alu_result를 반환한다.

Clock asynchronous

MUX2, 3

Forward A,B에 따라 MEM stage 혹은 write back stage에서 alu의 input에 대해 forwarding을 하거나 하지 않는다.

MUX6

ForwardE에 따라 data memory의 din으로 들어갈 register value에 대해 forwarding을 진행하거나 하지 않는다.

MEM stage

Data Memory

mem_read, mem_write에 따라 input address의 din을 저장하거나, address의 값을 출력한다.

WB stage

Mux4

mem_to_reg에 따라 alu_result 또는 data memory의 output을 출력한다.

Discussion

Compare total cycles between the single cycle and pipelined CPU?

basic_mem.txt를 기준으로, pipelined CPU는 36 cycles, single cycle CPU는 28 cycles가 걸린다. pipelined CPU가 더 많은 cycle이 걸리는 이유는 한 단위 instruction이 5 cycle에 처리되고, 더해 stall이 동작하기 때문이며, cycle의 latency는 pipelined CPU가 Single cycle CPU보다 훨씬 작다.

How to implement Hazard Detection?

IF_ID의 instruction으로부터 얻은 rs1 또는 rs2가 ID_EX의 rd와 같고, ID_EX_가 mem_read를 한다면 load instruction으로 stall한다.

ecall instruction이 fetch되었을 경우 3 cycle동안 HDU_out이 1이 되어 stall하고, x17의 값이 write back된 뒤에 x17의 값을 읽도록 한다.

How to implement data forwarding?

1. MEM stage에서 EX stage로

ID/EX의 rs1, rs2 가 MEM/WB rd와 일치하고, reg_write bit가 1인 경우에 ID/EX의 일치하는 레지스터로 forwarding을 진행한다. 2번의 조건의 충족되었어도 1번의 조건이 충족되면 MEM stage에서 EX stage로 forwarding한다.

2. WB stage에서 EX stage로

EX/MEM의 rs1, rs2 가 MEM/WB rd와 일치하고, reg_write bit가 1인 경우에 EX/MEM의 일치하는 레지스터로 forwarding을 진행한다.

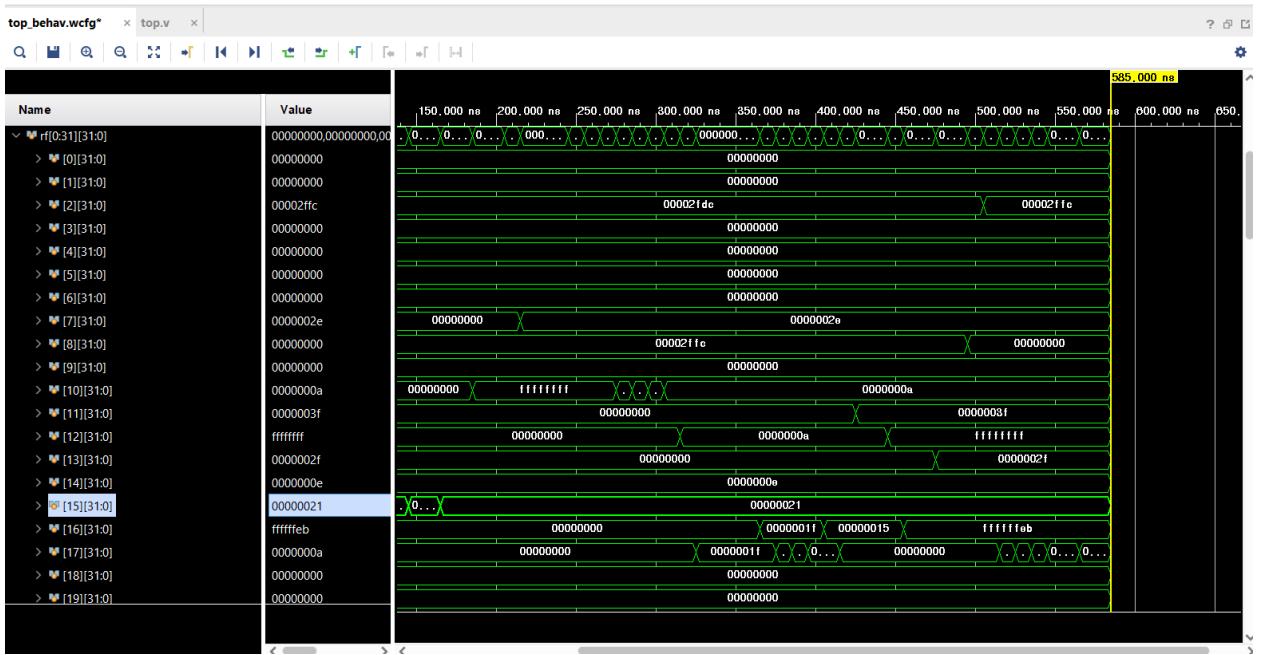
3. WB stage에서 ID stage로

IF/ID의 rs1, rs2 가 MEM/WB rd와 일치하고, reg_write bit가 1인 경우에 IF/ID의 일치하는 레지스터로 forwarding을 진행한다.

Conclusion

1. 실험 결과

여러 testbench code로 ripes의 결과와 비교해 보았을 때, 구현한 CPU가 정상적으로 작동함을 확인할 수 있다.



2. 결론

이번 Lab-Session에서는 한 clock에 한 stage를 나눠 처리하는 pipelined CPU를 구현하였다. Pipeline-Cycle이 stage 별로 register에 정보를 저장하여 동작하는 방식과 이에 따른 memory의 저장, data reading, ALU calculation, PC change 등을 verilog 문법으로 구현하는 방법도 익힐 수 있었다.