컴퓨터 조직론
Project 4

Jaewon Hur

System & software security Lab.

Seoul National University

May 29, 2020

목차

프로젝트 목표

- 본 프로젝트 목표
  - Five stage in-order CPU 구현
    1. CPU logic 재구성 및 pipeline register 추가
    2. Forwarding logic 구현
    3. Branching 및 flushing 구현
    4. Hazard detection

프로젝트 개요

- Pipelined CPU
  - Multiple stage를 거쳐 instruction을 수행
  - Single cycle CPU에서 각 stage마다 정보를 저장하기 위해 pipeline register 추가
  - 성능 향상을 위해 forwarding logic, branch predictor, hazard detection 추가

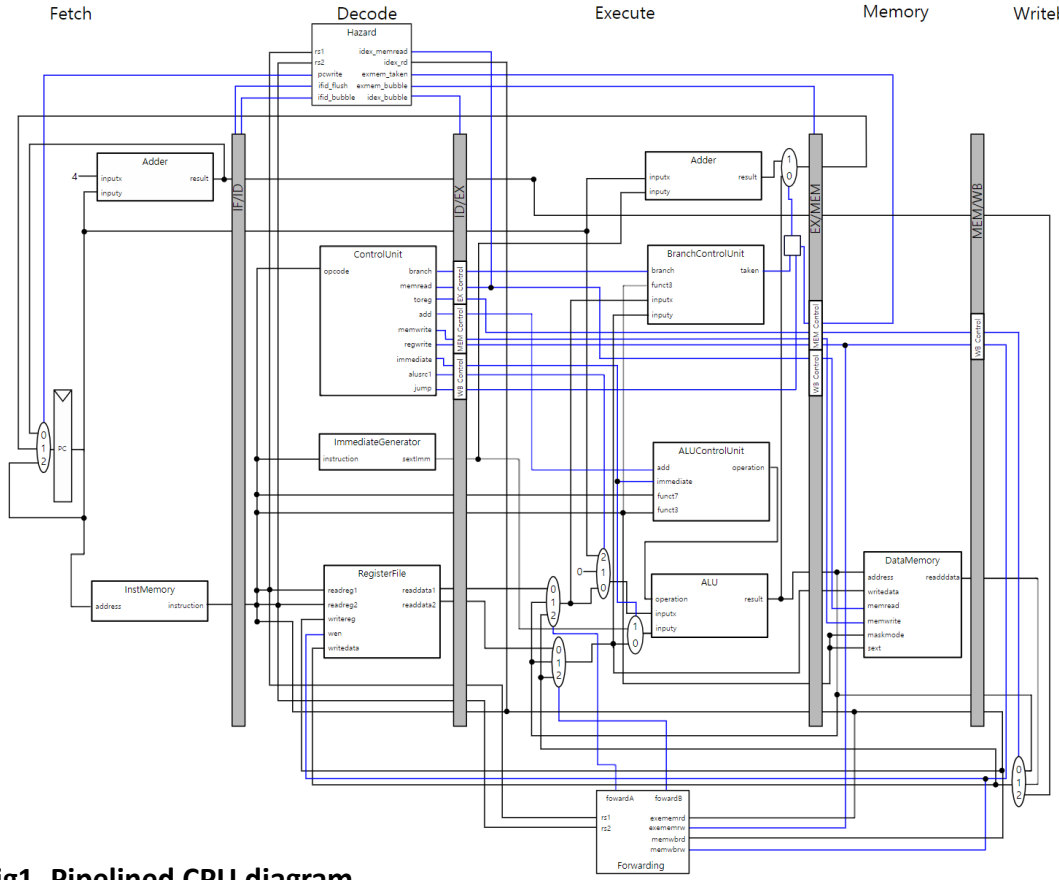

Fig1. Pipelined CPU diagram

프로젝트 구성

1. CPU logic 재구성 및 pipeline register 추가
   - Forwarding, branch prediction, hazard detection을 제외한 pipelined CPU를 구현.
   - IF stage는 이미 구현되어 있으며, 나머지 stage를 따라가며 순차적으로 구현.
     - **src/main/scala/pipelined/cpu.scala**에 기본 template 제공

   - 참고 사항
     - Pipeline register는 *Bundle*로 구현됨, 일반 object처럼 내부 인자에 접근할 수 있음.
     - Control signal (EXControl, Mcontrol, WBControl) 또한 *Bundle*로 구현
     - Module 연결 완료시 DontCare 구문 제거

   - 평가 및 검증

sbt> Lab3 / testOnly dinocpu.RTypeTesterLab3

sbt> Lab3 / testOnly dinocpu.ITypeTesterLab3

sbt> Lab3 / testOnly dinocpu.UTypeTesterLab3

sbt> Lab3 / testOnly dinocpu.MemoryTesterLab3

     - Test cases
       - 14/ 10/ 6/ 10

```
// Everything in the register between IF and ID stages
class IFIDBundle extends Bundle {
  val instruction = UInt(32.W)
  val pc          = UInt(32.W)
  val pcplusfour  = UInt(32.W)
}
```

Fig2. Pipeline register bundle

```
// Control signals used in EX stage
class EXControl extends Bundle {
  val add       = Bool()
  val immediate = Bool()
  val alusrc1   = UInt(2.W)
  val branch    = Bool()
  val jump      = UInt(2.W)
}

// Control signals used in MEM stage
class MControl extends Bundle {

}

// Control signals used in EB stage
class WBControl extends Bundle {

}
```

Fig3. Control signal bundle

```
// Everything in the register between ID and EX stages
class IDEXBundle extends Bundle {

  val excontrol = new EXControl
  val mcontrol  = new MControl
  val wbcontrol = new WBControl
}
```

Fig4. Bundle instantiation

```
val if_id      = RegInit(0.U.asTypeOf(new IFIDBundle))
val id_ex      = RegInit(0.U.asTypeOf(new IDEXBundle))
val ex_mem     = RegInit(0.U.asTypeOf(new EXMEMBundle))
val mem_wb     = RegInit(0.U.asTypeOf(new MEMWBBundle))
```

Fig5. Register instantiation

프로젝트 구성

2. Forwarding logic 구현

- 다음 단계에 따라 forwarding logic 구현
  1) Fig. 6에 따라 forwarding mux 추가 (**src/main/scala/pipelined/cpu.scala**)
  2) Forwarding unit을 processor에 연결
  3) Forwarding unit 내부 forwarding logic 구현 (**src/main/scala/components/forwarding.scala**)

- 평가 및 검증

sbt> Lab3 / testOnly dinocpu.ITypeMultiCycleTesterLab3

sbt> Lab3 / testOnly dinocpu.RTypeMultiCycleTesterLab3

  - Test cases
    - 1/ 8

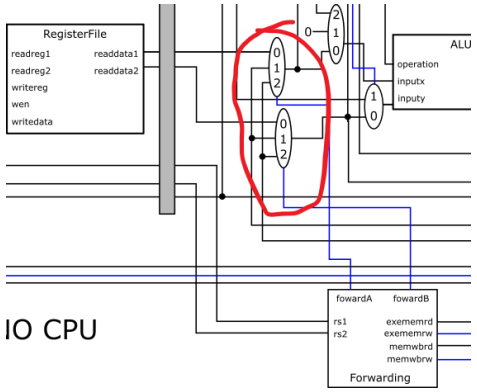

Fig 6. Forwarding mux

프로젝트 구성

3. Branching 및 flushing 구현

- Always not-taken branch predictor (branch condition을 기다리지 말고 계속 pc+4 instruction을 fetch) 가정
- 다음 단계에 따라 branching 및 flushing 구현 (**src/main/scala/pipelined/cpu.scala**)
  1) PC값을 위한 MUX 추가
  2) ID/EX, EX/MEM register에 bubble 주입을 위한 코드 추가
  3) IF/ID register flush 코드 추가
  4) Taken signal을 hazard detection unit과 연결
  5) Branch taken일 때, hazard detection logic 추가 (**src/main/scala/components/hazard.scala**)

- 평가 및 검증

sbt> Lab3 / testOnly dinocpu.BranchTesterLab3

sbt> Lab3 / testOnly dinocpu.JumpTesterLab3

- Test cases
  - 18/ 3

프로젝트 구성

4. Hazard detection

- 특정 조합의 instruction (dependent instructions)을 hazard detection logic이 처리
  - E.x) mem read / use 같은 1 cycle에 operand 값을 얻어올 수 없는 instruction 조합
- 다음 단계에 따라 hazard detection 구현 (**src/main/scala/pipelined/cpu.scala**)
  1) Hazard detection unit의 나머지 부분 연결 (**src/main/scala/components/hazard.scala**)
  2) PC Mux 수정
  3) IF/ID registe에 bubbler 주입 코드 작성

- 평가 및 검증

sbt> Lab3 / testOnly dinocpu.MemoryMultiCycleTester3

sbt> Lab3 / testOnly dinocpu.ApplicationsTesterLab3

  - Test cases
    - 1/ 4

평가 기준

1. CPU logic 재구성 및 pipeline register 추가
   - 성공하는 test case 개수에 따라 차등
2. Forwarding logic 구현
   - 성공하는 test case 개수에 따라 차등
3. Branching 및 flushing 구현
   - 성공하는 test case 개수에 따라 차등
4. Hazard detection
   - 성공하는 test case 개수에 따라 차등

참고

1. Single-step mode test

- Simulation을 한 cycle씩 수행하고 싶을 때
  - **sbt>** Lab3 / runMain dinocpu.singlestep ‘test name’ ‘cpu-version’
  - 예시) **sbt>** Lab3 / runMain dinocpu.singlestep fibonacci pipelined
- Single step interface

```
[info] ApplicationsTesterLab3:
[info] Pipelined CPU
[info] - should run application fibonacci *** FAILED ***
```

Fig 7. Fibonacci test

  - Print registers 명령어
    - print reg <num>: print value in register
    - print regs : print value in all registers
    - print pc: print the address in pc
    - print inst [addr]: print the disassembly for the instruction at addr. (if no addr, use current pc)
  - Print module I/O (wires) 명령어
    - dump all: show all modules and the values of their I/O
    - dump list: List the valid modules to dump
    - dump [module]: Show values of the I/O on a specific module
  - Control 명령어
    - step [num]: move forward this many cycle, default 1
  - Other commands
    - ? : print this help
    - q : quit

Thank you

문의사항은 hurjaewon@snu.ac.kr 로 연락 바랍니다.