2020-1 컴퓨터 조직론

Project 3. Single-cycle CPU design

Juhee Kim

System & software security Lab.
Seoul National University
2020/05/15

Table of Contents

Project Goals

- About this project
  - You will draw the control-path line for the single-cycle CPU.
  - You will implement the control path for the single-cycle CPU.

- Goals
  - Learn how to implement a control and data path in a single cycle CPU.
  - Learn how different RISC-V instructions interact in the control and data path of a single-cycle CPU.

Project Overview

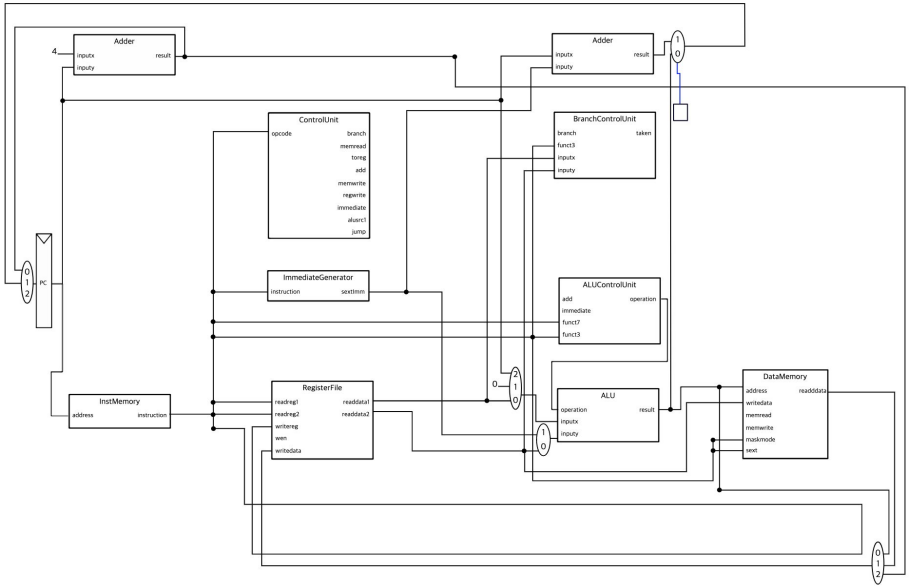
1. Draw the control wires of single-cycle CPU diagram

2. Implement each components step-by-step
    - Control unit overview
    - Part I: Memory instructions
    - Part II: Branch instructions
    - Part III: Full applications

Single cycle CPU design

- Draw the control wires of single-cycle CPU diagram
  - this diagram includes all the necessary data path wires and Muxes
  - only control path wires are missing

  - Wire마다 bit width 표시



Control unit overview

- Control unit
  - opcode format and control signals for R-type instructions

opcode	opcode format	branch	memread	toreg	add	memwrite	immediate	regwrite	alusrc1	jump
-	default	false	false	3	false	false	false	false	0	0
000000	invalid	false	false	0	false	false	false	false	0	0
0110011	R	false	false	0	false	false	false	true	0	0

- You must fill in all other instructions in the table in src/main/scala/components/control.scala.
  - R-types are already implemented. Check with sbt:dinocpu> testOnly dinocpu.SingleCycleRTypeTesterLab3
  - **Important: Do not modify the I/O. Modify only signals table.**

Control unit overview

- Control unit
  - input: 7-bit opcode
  - output:

```
branch :      true if branch or jump and link register (jal). update PC with immediate
memread :     true if we should read from memory
toreg :       0 for writing ALU result, 1 for writing memory data, 2 for writing pc + 4
add :         true if the ALU should add the results
memwrite :    true if writing to the data memory
regwrite :    true if writing to the register file
immediate :   true if using the immediate value
alusrc1 :     0 for read data 1, 1 for the constant zero, 2 for the PC
jump :        0 for no jump, 2 for jump, 3 for jal (jump and link register)
```

Part I: Memory instructions

- I-types

31-20	19-15	14-12	11-7	6-0	Name
imm[11:0]	rs1	funct3	rd	0010011	I-type

```
R[rd] = R[rs1] <op> immediate  // <op> is specified by the funct3
```

  - Hint: Extend the table in control.scala, add 1 extra MUX to the CPU (cpu.scala) and wire the control signals to the MUX
  - Testing

```
sbt:dinocpu> testOnly dinocpu.SingleCycleITypeTesterLab3
```

Part I: Memory instructions

- `lw` (load word)

31-20	19-15	14-12	11-7	6-0	Name
imm[11:0]	rs1	010	rd	000011	lw

```
R[rd] = M[R[rs1] + immediate]
```

  - Testing

```
sbt:dinocpu> testOnly dinocpu.SingleCycleLoadTesterLab3
```

Part I: Memory instructions

- U-types
- lui (load upper immediate)

31-12	11-7	6-0	Name
imm[31:12]	rd	0110111	lui

```
R[rd] = imm << 12
```

Important: The immediate generator will produce the shifted and sign extended value! You do not need to shift the immediate value outside of the immediate generator.

- auipc (add upper immediate to pc)

31-12	11-7	6-0	Name
imm[31:12]	rd	0010111	auipc

```
R[rd] = pc + (imm << 12)
```

```
sbt:dinocpu> testOnly dinocpu.SingleCycleUTypeTesterLab3
```

Part I: Memory instructions

- sw (store word)

31-25	24-20	19-15	14-12	11-7	6-0	Name
imm[11:5]	rs2	rs1	010	imm[4:0]	0100011	sw

```
M[R[rs1] + immediate] = R[rs2]
```

  - Testing

```
sbt:dinocpu> testOnly dinocpu.SingleCycleStoreTesterLab3
```

Part I: Memory instructions

- Others

```
l     : load
s     : store
b     : byte (8 bits)
h     : half word (16 bits)
u     : unsigned (32 bits)
sext  : sign-extend
&     : bit-wise AND
```

31-25	24-20	19-15	14-12	11-7	6-0	Name	Functionality
imm[11:5]	imm[4:0]	rs1	000	rd	0000011	lb	R[rd] = sext(M[R[rs1] + immediate] & 0xff)
imm[11:5]	imm[4:0]	rs1	001	rd	0000011	lh	R[rd] = sext(M[R[rs1] + immediate] & 0xffff)
imm[11:5]	imm[4:0]	rs1	010	rd	0000011	lw	R[rd] = M[R[rs1] + immediate]
imm[11:5]	imm[4:0]	rs1	100	rd	0000011	lbu	R[rd] = M[R[rs1] + immediate] & 0xff
imm[11:5]	imm[4:0]	rs1	101	rd	0000011	lhu	R[rd] = M[R[rs1] + immediate] & 0xffff
imm[11:5]	rs2	rs1	000	imm[4:0]	0100011	sb	M[R[rs1] + immediate] = R[rs2] & 0xff
imm[11:5]	rs2	rs1	001	imm[4:0]	0100011	sh	M[R[rs1] + immediate] = R[rs2] & 0xffff
imm[11:5]	rs2	rs1	010	imm[4:0]	0100011	sw	M[R[rs1] + immediate] = R[rs2]

```
sbt:dinocpu> testOnly dinocpu.SingleCycleLoadStoreTesterLab3
```

Part II: Branch instructions

```
b    : branch
u    : unsigned
eq   : equals
ne   : not equals
lt   : less than
ge   : greater than or equal to
```

- Branch instructions

31-25	24-20	19-15	14-12 (funct3)	11-7	6-0 (opcode)	Name
imm[12, 10:5]	rs2	rs1	000	imm[4:1,11]	1100011	beq
imm[12, 10:5]	rs2	rs1	001	imm[4:1,11]	1100011	bne
imm[12, 10:5]	rs2	rs1	100	imm[4:1,11]	1100011	blt
imm[12, 10:5]	rs2	rs1	101	imm[4:1,11]	1100011	bge
imm[12, 10:5]	rs2	rs1	110	imm[4:1,11]	1100011	bltu
imm[12, 10:5]	rs2	rs1	111	imm[4:1,11]	1100011	bgeu

```
if (R[rs1] <op> R[rs2])
  pc = pc + immediate
else
  pc = pc + 4
```

Part II: Branch instructions

branch: true if we are looking at a branch
funct3: the middle three bits of the instruction (12-14). Specifies the type of branch. See RISC-V spec for details.

inputx: first value (e.g., reg1)
inputy: second value (e.g., reg2)
taken: true if the branch is taken.

- Branch control unit

Instead of using the ALU to compute whether the branch is taken or not (the zero output), we use a dedicated branch control unit.

You should implement the proper control to choose the right type of branch test.
You must also correctly set or reset the taken output if the branch test passes of fails, respectively.

Modify `src/main/scala/components/branch-control.scala`

Hint: use Chisel's switch, `is`, `when`, `elsewhen`, `otherwise`, or `MuxCase`. You can also use normal operators, such as `<`. `>`, `===`, `=/=`, etc.

```
sbt:dinocpu> testOnly dinocpu.SingleCycleBranchTesterLab3
```

Part II: Branch instructions

- J-type instructions: unconditional branches
- jal

31-12	11-7	6-0	Name
imm[20, 10:1, 11, 19:12]	rd	1101111	jal

```
pc = pc + imm
R[rd] = pc + 4
```

- jalr

31-20	19-15	14-12	11-7	6-9	Name
imm[11:0]	rs1	000	rd	1100111	jalr

```
pc = R[rs1] + imm
R[rd] = pc + 4
```

```
sbt:dinocpu> testOnly dinocpu.SingleCycleJALRTesterLab3
```

Part III: Full applications

- fibonacci,
  which computes the nth Fibonacci number. The initial value of t1 contains the Fibonacci number to compute, and after computing, the value is found in t0.
- naturalsum
- multiplier
- divider

To run all the applications at once :

```
sbt:dinocpu> testaOnly dinocpu.SingleCycleApplicationsTesterLab3
```

To run a single application :

```
sbt:dinocpu> testOnly dinocpu.SingleCycleApplicationsTesterLab3 -- -z <binary name>
```

Evaulation

- Rule: **You are not allowed to change the I/Os.**

- Code portion to be written
  - src/main/scala/components/branch-control.scala
  - src/main/scala/components/control.scala
  - src/main/scala/single-cycle/cpu.scala

Evaluation

- Evaluation Criteria
  - Draw the single-cycle CPU diagram (100)
    - Wire와 bit width 모두 올바른 경우: 100
    - Bit width에 오류가 있는 경우: 50
    - Wire 연결에 오류가 있는 경우: 0
  - Single-cycle CPU implementation (200)
    - SingleCycleITypeTesterLab3: 10
    - SingleCycleLoadTesterLab3: 25
    - SingleCycleUTypeTesterLab3: 25
    - SingleCycleStoreTesterLab3: 25
    - SingleCycleLoadStoreTesterLab3: 25
    - SingleCycleBranchTesterLab3: 25
    - SingleCycleJALRTesterLab3: 25
    - SingleCycleApplicationsTesterLab3: 40

Hints

- `printf` debugging
  - Use `printf` when you want to print during the simulation.
  - Note: this will print at the end of the cycle so you'll see the values on the wires after the cycle has passed.
  - Use `printf(p"This is my text with a $var\n")` to print Chisel variables. Notice the "p" before the quote!
  - You can also put any Scala statement in the print statement (e.g., `printf(p"Output: ${io.output}")`).
  - Use `println` to print during compilation in the Chisel code or during test execution in the test code. This is mostly like Java's println.
  - If you want to use Scala variables in the print statement, prepend the statement with an 's'. For example, `println(s"This is my cool variable: $variable")` or `println("Some math: 5 + 5 = ${5+5}")`.

Submission

Deadline: 2020.05.29 (목) 오후 11:55

Delay: TBD

제출 방법: Project2와 동일

1. 제출 홈페이지 접속 (http://kayle.snu.ac.kr:37373/)
2. 지난번에 이메일로 받은 api-key로 홈페이지 접속
3. proj3_hw 폴더에서 ‘zip –r lab3.zip proj3_cpu_diagram.pdf src’ 명령 으로 diagram파일과 src 폴더를 lab3.zip 파일로 압축한 후 홈페이지 에 제출