Outro

Martin Schoeberl

Technical University of Denmark
Embedded Systems Engineering

March 2, 2021

Overview

- Exam
- Evaluation
- Chisel from scratch
- Who is Alan Turing?
- Lipsi, a simple processor
- We have well known guest lectures today
  - Kasper, Anthon, and Simon will present today

The Online Exam

- At http://onlineeksamen.dtu.dk/
- On Friday 15/5 10:00–12:00 (12:30)
- It will be two parts:
  - Multiple choice
  - Download and hand in a PDF document (e.g., generated from Word or photo taken)

Evaluation

- Thank you!
- Looks like you enjoyed DE2 and Chisel :-)
- Let us look into it

Write in Chisel from Scratch

- This was a request from the evaluation
- Should not be too hard
- We need just two files
  - `build.sbt` for library dependencies
  - One Scala file
- Let's do it now!

FSMD

- A finite-state machine and a datapath
- Can compute
- Your vending machine is an FSMD
- Can we use this to build a general processor?

What is a *General* Processor?

- A computing machine that can compute all computable problems
- What is computable?
- Mr. Turing thought about this before computers where built (1936)
- The Turing machine can compute all computable problems
- How useful is this?
- What is NOT computable?
- Assumption is infinite resources (memory)
- But even with finite amount of memory it is a VERY useful classification

A Practical Turing-Complete Machine

- Compute with some operations
- Control 1: an FSM to steer the datapath
- Control 2: instructions to steer the FSM
- Storage: memory for the *infinite/large* storage

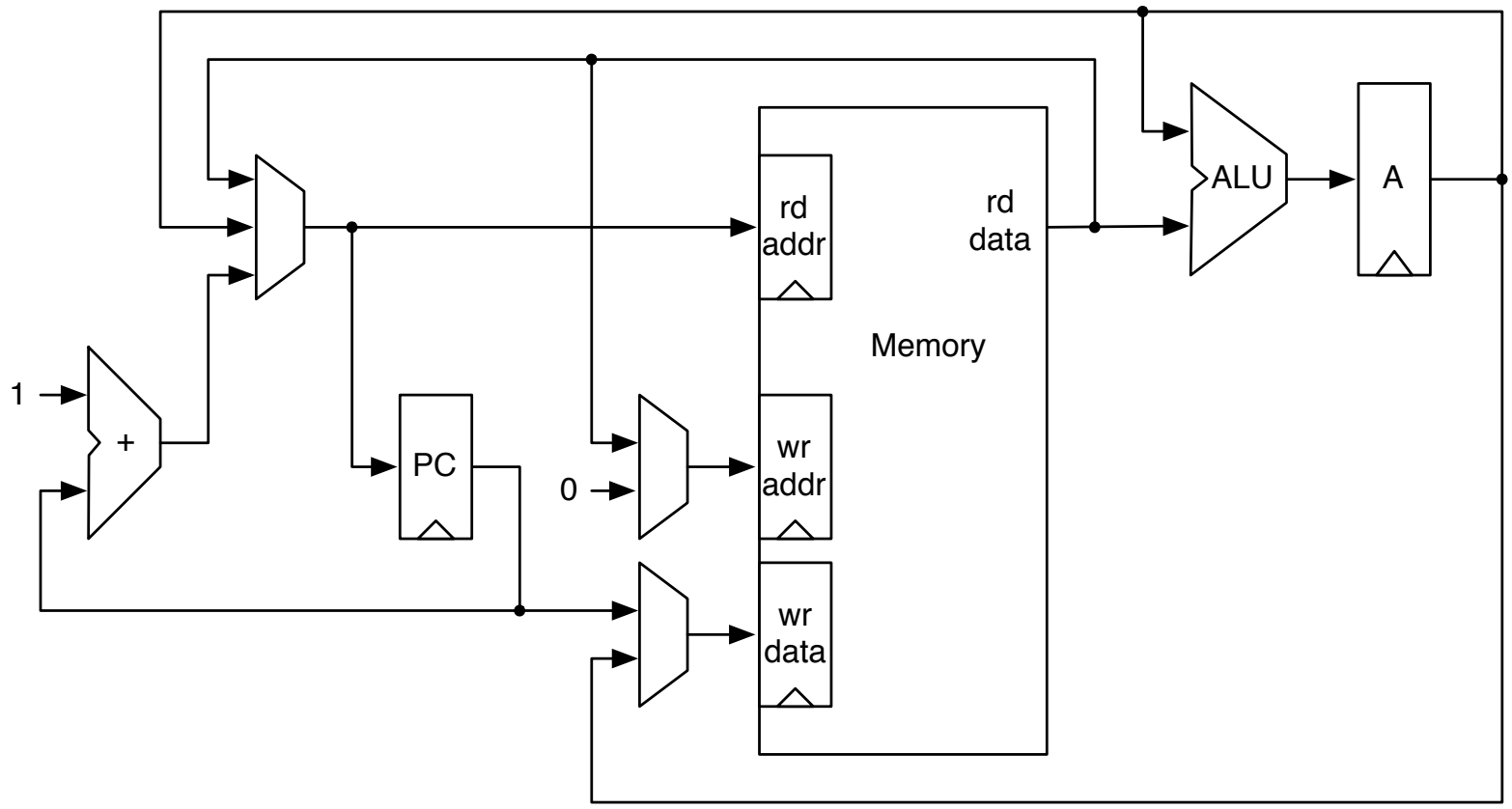
Can We Build Such a Processor?

- Our Chisel and digital design knowledge should be enough
- Let's start with a simple one
- An FSMD plus memory
- As it is small we name it after a small island: Lipsi

Lipsi

- The paper: Lipsi: Probably the Smallest Processor in the World
- Code: The Chisel Code
- A simple Accumulator machine

Lipsi Datapath



Datapath Elemenets

- An arithmetic-logic unit (ALU)
- An accumulator: register A
- Memory for instructions and data
- a program counter (PC)

Commanding the FSM

- We need so-called instructions
- They drive the FSM
- To computer (e.g., +, -, or): ALU operations
- To load from and store into memory
- To (conditionally) branch (implement if/else, loops)

Lipsi Instruction Set

Encoding		Instruction	Meaning	Operation
`0fff rrrr`		*f* rx	ALU register	A = A *f* m[r]
`1000 rrrr`		st rx	store A into register	m[r] = A
`1001 rrrr`		brl rx	branch and link	m[r] = PC, PC = A
`1010 rrrr`		ldind (rx)	load indirect	A = m[m[r]]
`1011 rrrr`		stind (rx)	store indirect	m[m[r]] = A
`1100 -fff`	`nnnn nnnn`	*fi* n	ALU immediate	A = A *f* n
`1101 --00`	`aaaa aaaa`	br	branch	PC = a
`1101 --10`	`aaaa aaaa`	brz	branch if A is zero	PC = a
`1101 --11`	`aaaa aaaa`	brnz	branch if A is not zero	PC = a
`1110 --ff`		sh	ALU shift	A = shift(A)
`1111 aaaa`		io	input and output	IO = A, A = IO
`1111 1111`		exit	exit for the tester	PC = PC

ALU Operations

Encoding	Name	Operation
`000`	add	$A = A + op$
`001`	sub	$A = A - op$
`010`	adc	$A = A + op + c$
`011`	sbb	$A = A - op - c$
`100`	and	$A = A \wedge op$
`101`	or	$A = A \vee op$
`110`	xor	$A = A \oplus op$
`111`	ld	$A = op$

The ALU

```
val add :: sub :: adc :: sbb :: and :: or :: xor
    :: ld :: Nil = Enum(8)
switch(funcReg) {
  is(add) { res := accuReg + op }
  is(sub) { res := accuReg - op }
  is(adc) { res := accuReg + op } // TODO: adc
  is(sbb) { res := accuReg - op } // TODO: sbb
  is(and) { res := accuReg & op }
  is(or) { res := accuReg | op }
  is(xor) { res := accuReg ^ op }
  is(ld) { res := op }
}
```

Some Defaults

```
wrEna := false.B
wrAddr := rdData
rdAddr := Cat(0.U(1.W), nextPC)
updPC := true.B
nextPC := pcReg + 1.U
enaAccuReg := false.B
enaPcReg := false.B
enaIoReg := false.B
```

Conditions for Branches

```
val accuZero = accuReg === 0.U

val doBranch = (rdData(1, 0) === 0.U) ||
  ((rdData(1, 0) === 2.U) && accuZero) ||
  ((rdData(1, 0) === 3.U) && !accuZero)
```

The FSM States and Register

```
val fetch :: execute :: stind :: ldind1 ::
    ldind2 :: exit :: Nil = Enum(6)
val stateReg = RegInit(fetch)
```

A Large State Machine

```
switch(stateReg) {
  is(fetch) {
    stateReg := execute
    funcReg := rdData(6, 4)
    // ALU register
    when(rdData(7) === 0.U) {
      updPC := false.B
      funcReg := rdData(6, 4)
      enaAccuReg := true.B
      rdAddr := Cat(0x10.U, rdData(3, 0))
    }
    // st rx, is just a single cycle
    when(rdData(7, 4) === 0x8.U) {
      wrAddr := Cat(0.U, rdData(3, 0))
      wrEna := true.B
      stateReg := fetch
    }
  ...
```

Memory

- Code memory for instructions
- Data memory for data
- Merge those two
- Instruction memory filled by a program
- That program is an assembler written in Scala

Code and Data Memory

```
val program =
    VecInit(util.Assembler.getProgram(prog).map(_.U))
val instr = program(rdAddrReg(7, 0))

val mem = Mem(256, UInt(8.W))
val data = mem(rdAddrReg(7, 0))
when(io.wrEna) {
  mem(io.wrAddr) := io.wrData
}

// Output MUX
io.rdData := Mux(rdAddrReg(8), data, instr)
```

Assembling Instructions

```
for (line <- source.getLines()) {
  if (!pass2) println(line)
  val tokens = line.trim.split(" ")
  val Pattern = "(.*:)".r
  val instr = tokens(0) match {
    case "#" => // comment
    case Pattern(l) => if (!pass2) symbols +=
        (l.substring(0, l.length - 1) -> pc)
    case "add" => 0x00 + regNumber(tokens(1))
    case "sub" => 0x10 + regNumber(tokens(1))
    case "adc" => 0x20 + regNumber(tokens(1))
    case "sbb" => 0x30 + regNumber(tokens(1))
    case "and" => 0x40 + regNumber(tokens(1))
    case "or" => 0x50 + regNumber(tokens(1))
```

This is done at hardware generation

Co-simulation for Testing

- Write an implementation of Lipsi in Scala
- This is an instruction set simulator, not hardware
- This is your golden model
- Run programs on the simulator and in the Chisel hardware
- Compare the results (the value in the accumulator)

A Processor Summary

- This is a tiny processor as an example
- Chisel is productive: this was all done in 14 hours!
- Kind of useful for small systems
- You could implement your vending machine on it
- Is this the way a general processor is built?
- No, we use something called pipelining
- You can learn this in:
  - 02155: Computer Architecture and Engineering

Future with Digital Design Education

- There are several companies in DK doing chip design
- Also FPGAs are used in embedded systems
- Digital design is only part of a computer engineering education
- But DTU does not have a clear path to a computer engineering education

Digital Design within an EE Master

- Not an obvious choice, as there is no specialization in digital systems
- Select some of the following courses
  - 02155: Computer Architecture and Engineering
  - 02203: Design of Digital Systems
  - 02211: Advanced Computer Architecture
  - 02205: VLSI Design
  - 02217: Design of Arithmetic Processors
  - 02204: Design of Asynchronous Circuits
  - 02209: Test of Digital Systems

Computer Engineering Education at DTU

- On the border between hardware and software
- Very well payed jobs :-)
- Not an easy choice at DTU as well
  - No BSc available
  - Between EE and CS
- Start with Bsc. in EE
- Specialization in Indlejrede systemer og programmering
  - 02155: Computer Architecture and Engineering
  - 02105: Algoritmer og datastrukturer
- Continue as MSc. in Computer Science and Engineering
- Specialization in
  - Digital Systems
  - Embedded and Distributed Systems

Summary

- You now know enough digital design to build any digital system
- You may get better on it with practice
- When you understand the principles you can easily learn Verilog or VHDL in days
- Chisel may be the future for hardware design
- You might apply for a job in silicon valley with your Chisel knowledge ;-)
- Hope to see some of you in upcoming courses