

Outro

Martin Schoeberl

Technical University of Denmark
Embedded Systems Engineering

May 4, 2023

Overview

- ▶ Evaluation
- ▶ Who is Alan Turing?
- ▶ Lipsi, a simple processor
- ▶ Show vending machine today
- ▶ Report hand-in at DTU Learn (21 May)

Evaluation

- ▶ In general, it looks like most enjoyed DE2 :-)
- ▶ Some complained it was only Chisel and nothing new
 - ▶ I disagree
 - ▶ e.g., did you already know: tick generation, FSMD, input synchronization, timing, verification, memory, UART,...
- ▶ We can take a quick look

FSMD

- ▶ A finite-state machine with 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 were 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

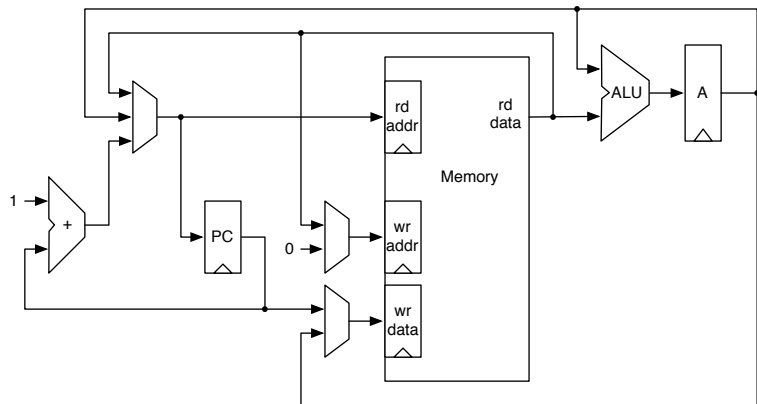
Can We Build Such a Processor?

- ▶ 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 Elements

- ▶ 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	<i>f</i> 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	<i>f</i> i 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	$A = \text{shift}(A)$
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 with 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)
```

An Assembly Program Example

- ▶ Digital hardware and processors only understand 0 and 1
- ▶ But, we do not want to program in 0s and 1s
- ▶ We write in [assembly language](#)

```
ldi 0x12
st r1
ldi 0x34
st r2
ldi 0
add r1
add r2
# now it is 0x46
```

Assembling Instructions

```
for (line <- source.getLines()) {
  if (!pass2) println(line)
  val tokens = line.trim.split(" ")
  val Pattern = "(.*:)"
  val instr = tokens(0) match {
    case "#" => // comment
    case Pattern(1) => if (!pass2) symbols +=
      (1.substring(0, 1.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)

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?
- ▶ Not today, we use something called pipelining
- ▶ You can learn this in:
 - ▶ 02155: Computer Architecture and Engineering

02155: Computer Architecture and Engineering

- ▶ Given by Alberto and me
- ▶ [Course description](#)
- ▶ Learn how a real-world processor work
- ▶ Learn the language of the machine (instructions)
- ▶ Virtual memory and caches
- ▶ We use RISC-V, the free instruction set
- ▶ Project: write a simulator for the RISC-V
 - ▶ In any language, may be in Chisel
 - ▶ May even be a full implementation in an FPGA
- ▶ You can also do a complete RISC-V in an FPGA in a Fagproject or 3 weeks course

Future with Digital Design Education

- ▶ There are many companies in DK doing chip design
- ▶ See [DTU Chip Day](#)
- ▶ FPGAs are available in the cloud
 - ▶ To speedup computing
 - ▶ You can rent them from Amazon
- ▶ FPGAs are also used in embedded systems
- ▶ Digital design is only part of a computer engineering education

Computer Engineering Education at DTU

- ▶ On the interaction between hardware and software
- ▶ Very well payed jobs :-)
- ▶ DTU has now a clear path to a computer engineering education
 - ▶ We start with the Computer Engineering BSc in fall 2023
- ▶ With a 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

Computer Engineering within the EE Bachelor

- ▶ You get well educated in electronics
- ▶ Add the software side to your education
- ▶ Select some of the following courses
 - ▶ 02155: Computer Architecture and Engineering
 - ▶ 02114: Design of a RISC-V Microprocessor (new, 3 weeks)
 - ▶ 02105: Algorithms and Data Structures 1
 - ▶ 02161: Software Engineering 1
 - ▶ 02159: Operating Systems
 - ▶ 02157: Functional Programming
 - ▶ 02141: Computer Science Modelling
 - ▶ 02110: Algorithms and Data Structures 2
- ▶ See also [Prerequisites for the CSE master](#)

Digital Design within a CSE or EE Master

- ▶ Select some of the following courses
 - ▶ 02201: Agile Hardware Design
 - ▶ 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
 - ▶ more to come...

Reading Recommendation

- ▶ [Chip War](#)

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 SystemVerilog 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