# Why?

Previously you learned how to build a readable and writeable addressable memory. You’ll use this and other components to design your first programmable processor.

# Model 1: The add (register) instruction

In this part, you will build your first programmable processor. It will support one ARM instruction, add, and four 32-bit registers called R0, R1, R2, and R3.

1. Consider the instruction

add R2, R1, R3

What registers will be written and what registers will be read to execute this instruction?

1. Suppose we implement the four registers as an addressable memory. Give the values of the memory parameters.

W =

L =

# read ports =

# write ports =

# Read This!

Such a component that houses all the registers behind the abstraction of an addressable memory is called a ***register file***.

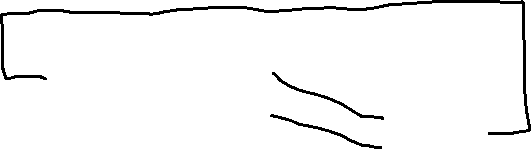
1. Write the RTL[[1]](#footnote-1) for the add instruction, shown in the reference sheet.
2. Since we only support the add instruction, we don't care much for op, funct, and other fields. Therefore, suppose our programs are given to the processor as three inputs RD, RN, and RM. Each instruction executes within 1 clock period.

Translate the following program to the RD, RN, RM inputs.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **clock period** | **program** | **RD** | **RN** | **RM** |
| 1 | add R2, R1, R3 |  |  |  |
| 2 | add R0, R0, R1 |  |  |  |
| 3 | add R1, R2, R2 |  |  |  |
| 4 | add R1, R1, R1 |  |  |  |

1. Design the processor using the given 32-bit adder and the register file.

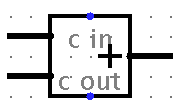


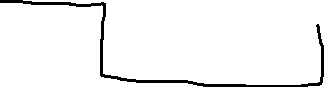
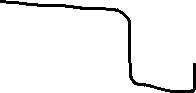


RD

RN

RM





# Model 2: The add (immediate) instruction

Now we can start including more instructions. Let's support both add (register) and add (immediate).

1. Write the RTL for the add (register) instruction. Directly underneath that, write the RTL for the add (immediate) instruction. Highlight their differences.

Add register: R[Rd] <- R[Rn] + R[Rm]

Add immediate: R[Rd] <- R[Rn] + ZeroExtend(imm8)

1. Translate the following program to the RD, RN, RM, and immediate. We'll also need a new input, I. I=0 means the instruction is add (register) and I=1 means the instruction is add (immediate). Put an 'X' for cases where it doesn't matter what that input value is.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **clock period** | **program** | **RD** | **RN** | **RM** | **immediate** | **I** |
| 1 | add R2, R1, #10 | R2 | R1 | / | #10 | 1 |
| 2 | add R0, R0, #7 | R0 | R0 | / | #7 | 1 |
| 3 | add R1, R2, R2 | R1 | R2 | R2 | / | 0 |
| 4 | add R1, R1, R3 | R1 | R3 | R3 | / | 0 |

1. Add circuitry to your processor to include the add (immediate) instruction. Your processor must support both instructions.



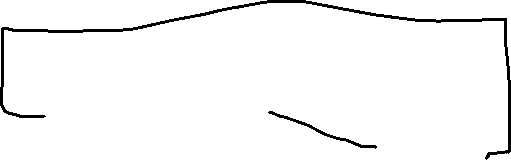
RD

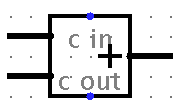
RN

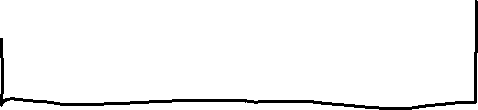
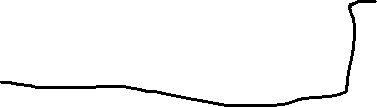
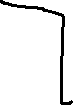
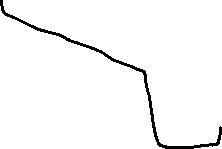
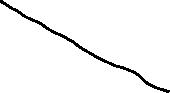
RM

Immediate

I







# Exercises



1. Now add the instruction orr (for both register and immediate). For each one, use the same methodology: i) write the new instruction’s RTL under the existing instructions, ii) annotate the differences and similarities, and iii) modify the circuit.

Orr (Reg): R[Rd] <- R[Rn] V R[Rm]

Orr (imm): R[Rd] <- R[Rn] V ZeroExtend(imm8)

# Extension questions

1. Based on the process of modifying the add processor to support the additional instructions, create a flowchart instructing someone how to implement a new arithmetic instruction (don't assume you know *which* instruction they are implementing).

# Why?

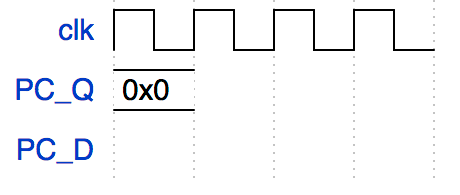
What circuit provides the program to the processor? Where should the inputs RD, RN, RM, immediate come from? Where should the input I come from? And, how should a new input be provided each clock cycle?

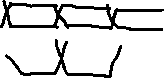
# Model 3: Stored programs

Here is Model 2, but we’ve included each instruction's fields, and the memory address of that instruction.

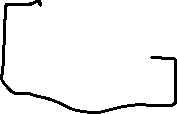
|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Memory address** | **program** | **I** | **RD** | **RN** | **RM** | **immediate** | **op** | **funct** |
| 0x0 | add R2, R1, #10 | 1 | 2 | 1 | - | 10 | 00 | 0100 |
| 0x4 | add R0, R0, #7 | 1 | 0 | 0 | - | 7 | 00 | 0100 |
| 0x8 | add R1, R2, R2 | 0 | 1 | 2 | 2 | - | 00 | 0100 |
| 0xC | add R1, R1, R3 | 0 | 1 | 1 | 3 | - | 00 | 0100 |

1. Let PC be a 32-bit register that stores the memory address of the current line of code. Draw the wave form for the D and Q signals of the PC register. Assume each instruction takes one clock period.





1. Draw a circuit that behaves as the waveform above. Label its output as “Address”. It has no inputs, and it has one output called "Address".



1. We need an addressable memory where we can store our instructions. For now, let's assume the memory is **read only** and already contains the instructions in it. Let's assume that the memory can store programs of up to 128 instructions. What are the parameters of the memory?

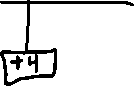
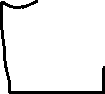
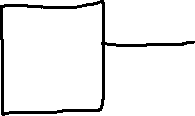
L = 128 bit width of address = 7

W = 32bit

# read ports = 1

# write ports = 1

1. Combine your answers to #13,14 to make a circuit with one output, the 32-bit machine code of the current instruction. Call this output "Instruction".



Refer to your diagram of the add (reg/immediate) processor from Model 2.

1. Complete your processor from Model 2 #6 by connecting it to the circuit you finished above. Make sure you get the register numbers and I out of the instruction. (HINT: need to use a splitter on the Instruction signal; make sure to label the bit indexes).
2. In Model 2, we used a register file of 4 registers. In a full ARM processor, we need 32 registers. Give the parameters of the addressable memory for this register file.

L = bit width of address =

W =

# read ports =

# write ports =

# Extension Questions

1. What changes could you make to add more instructions, like load word/store word, branches, and conditional instructions?

# Appendix

<http://wavedrom.com/editor.html>

{ "signal" : [ { name: "clk", wave:"p..."},

{ name: "PC\_Q", wave: "2", data: "0x0"},

{ name: "PC\_D", wave: ""}

], }

Learning goals

* Build a simple datapath that can execute the add instruction
* Modify the add datapath to enable another instruction

Concepts

* register file

Process skills goals

* Problem solving: addressed by designing a circuit to meet a written specification
* Written communication: addressed by conveying a design as a clear, labeled drawing

Prerequisite skills

* Design an accumulator circuit
* Use an addressable memory with multiple ports
* Interpret a simple MIPS assembly program

1. Depending on the reference sheet, this might also be called Behavior or Operation [↑](#footnote-ref-1)