Chapter 4b: Building a Nano RISC-V CPU

Stefan Mangard

October 22, 2019

Computer Organization and Networks
Graz University of Technology

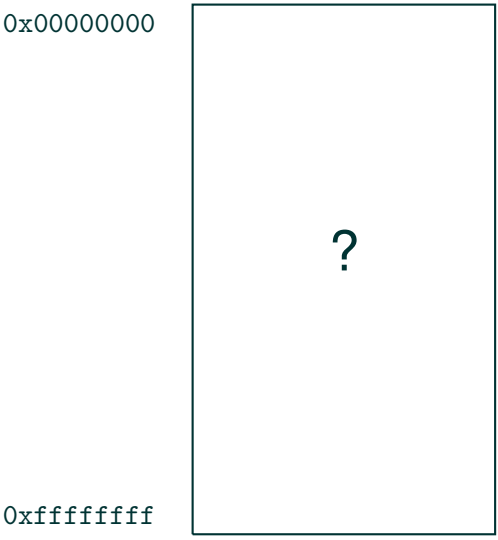
Nano RISC-V

Nano RISC-V is a simple 32-bit RISC-V processor

- 1 Zero Register: `x0`
- 31 General Purpose Registers: `x1` - `x31`
- 2 ALU Instructions (`OP rd, rs1, rs2`)
  - ADD, AND
- Load Instruction (`LW rd, offset(rs1)`)
- Store Instruction (`SW rs2, offset(rs1)`)
- Halt Instruction (`EBREAK`)
- almost 2 KiB of Memory (`0x000` - `0x7fc`)

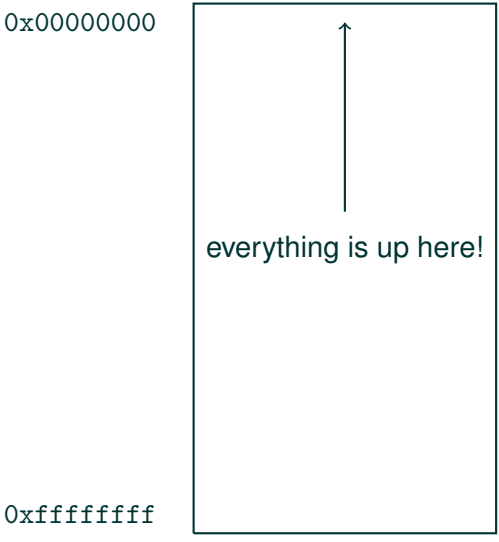
Memory Map (to scale)

- RAM extends from `0x000` to `0x7fc`



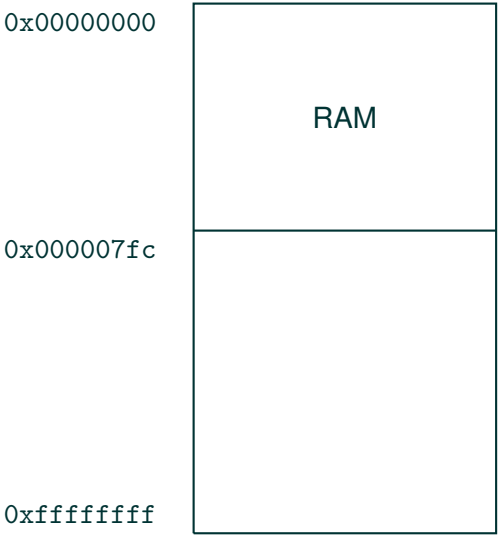
Memory Map (to scale)

- RAM extends from `0x000` to `0x7fc`
  - That's only 0.000047% of the address space!



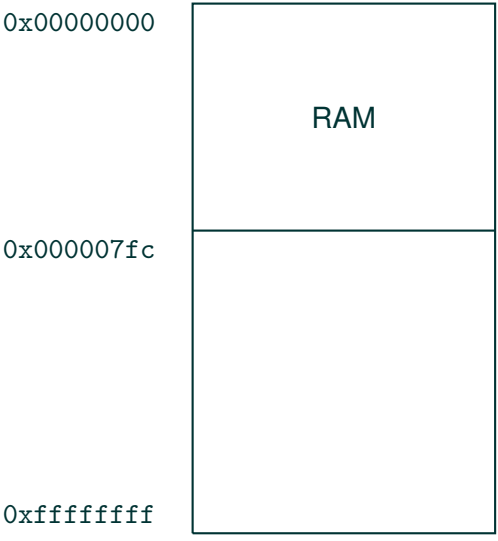
Memory Map (not to scale)

- RAM extends from `0x000` to `0x7fc`
  - That's only 0.000047% of the address space!

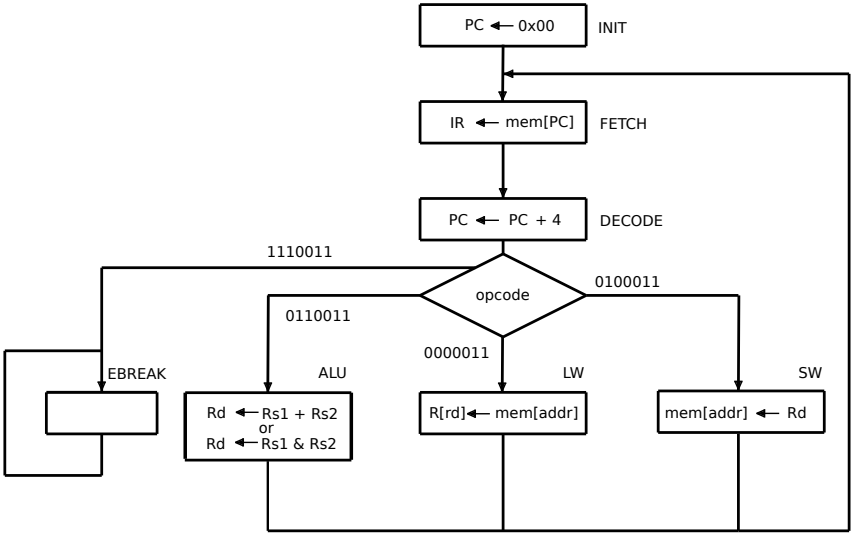


Memory Map (not to scale)

- RAM extends from `0x000` to `0x7fc`
  - That's only 0.000047% of the address space!
- Everything else is invalid memory



Instruction Cycle



Programming Nano RISC-V

Load Instruction

Load a word (32 bits / 4 bytes) from memory into a register:

$$\texttt{rd} \Leftarrow \texttt{MEMORY}[\texttt{rs1} + \textit{offset}]$$

Multiple uses:

- load from a pointer ($\textit{offset} = 0$)
- load from an address ($\texttt{rs1} = \texttt{x0}$)
- load from pointer + offset

Store Instruction

Store a register to a word (32 bits / 4 bytes) in memory:

$$\texttt{MEMORY}[\texttt{rs1} + \textit{offset}] \Leftarrow \texttt{rs1}$$

Multiple uses:

- store to a pointer ($\textit{offset} = 0$)
- store to an address ($\texttt{rs1} = \texttt{x0}$)
- store to pointer + offset

Add and And Instruction

Perform calculations:

$$\texttt{rd} = \texttt{rs1} + \texttt{rs2}$$

$$\texttt{rd} = \texttt{rs1} \ \& \ \texttt{rs2}$$

Notes:

- destination register can be equal to a source register

A Simple Program

- Load values from memory address `0x20`, `0x24` into registers
- Add the registers together
- Store the result back to memory at `0x28`
- Halt the CPU

A Simple Program

Find the right parameters for the instructions:

LW	$\texttt{rd} = \texttt{x1}$	$\texttt{rs1} = \texttt{x0}$	$\mathit{offset} = 0x20$
LW	$\texttt{rd} = \texttt{x2}$	$\texttt{rs1} = \texttt{x0}$	$\mathit{offset} = 0x24$
ADD	$\texttt{rd} = \texttt{x3}$	$\texttt{rs1} = \texttt{x1}$	$\texttt{rs2} = \texttt{x2}$
SW	$\texttt{rs2} = \texttt{x3}$	$\texttt{rs1} = \texttt{x0}$	$\mathit{offset} = 0x28$
EBREAK			

A Simple Program

Use the correct instruction encodings:

Type	`funct7`	`rs2`	`rs1`	`funct3`	`rd`	`opcode`
I-Type	0x20		0	LW	1	LOAD
I-Type	0x24		0	LW	2	LOAD
R-Type	DEFAULT	2	1	ADD	3	ALU
S-Type	hi(0x28)	3	0	SW	lo(0x28)	STORE
I-Type	EBREAK		0	PRIV	0	SYSTEM

A Simple Program

Use the correct instruction encodings:

Type	`funct7`	`rs2`	`rs1`	`funct3`	`rd`	`opcode`
I-Type	0000001	00000	00000	010	00001	0000011
I-Type	0000001	00100	00000	010	00010	0000011
R-Type	0000000	00010	00001	000	00011	0110011
S-Type	0000001	00011	00000	010	01000	0100011
I-Type	0000000	00001	00000	000	00000	1110011

A Simple Program

Put the bits together:

Instruction	Binary	Hexadecimal	Bytes
LW	00000010000000000010000010000011	0x02002083	83 20 00 02
LW	00000010010000000010000100000011	0x02402103	03 21 40 02
ADD	00000000001000001000000110110011	0x002081b3	b3 81 20 00
SW	00000010001100000010010000100011	0x02302423	23 24 30 02
EBREAK	00000000000100000000000001110011	0x00100073	73 00 10 00

A Simple Program

Add some data:

Instruction	Address	Value	Bytes
LW	0x00	0x02002083	83 20 00 02
LW	0x04	0x02402103	03 21 40 02
ADD	0x08	0x002082b3	b3 81 20 00
SW	0x0c	0x02302423	23 24 30 02
EBREAK	0x10	0x00100073	73 00 10 00
	0x14	0	00 00 00 00
	0x18	0	00 00 00 00
	0x1c	0	00 00 00 00
	0x20	42	2a 00 00 00
	0x24	13	0d 00 00 00
	0x28	0	00 00 00 00

A Simple Assembler

- writing Instruction opcodes by hand is tedious
- introduce tool to assemble for us
  - `riscvasm.py`

- usage: `riscvasm.py program.asm -o program.hex`

A Simple Assembler Program

The same program written for the assembler:

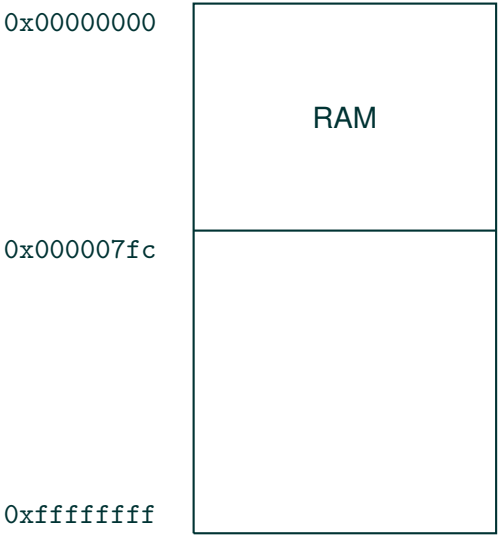
```
.org 0x00 # start program at address 0x00
  LW x1, 0x20(x0)
  LW x2, 0x24(x0)
  ADD x3, x1, x2
  SW x3, 0x28(x0)
  EBREAK

.org 0x20 # place data at address 0x20
  # insert raw data instead of instructions
  .word 42
  .word 13
```

Input and Output

- We need a way to communicate with the outside world

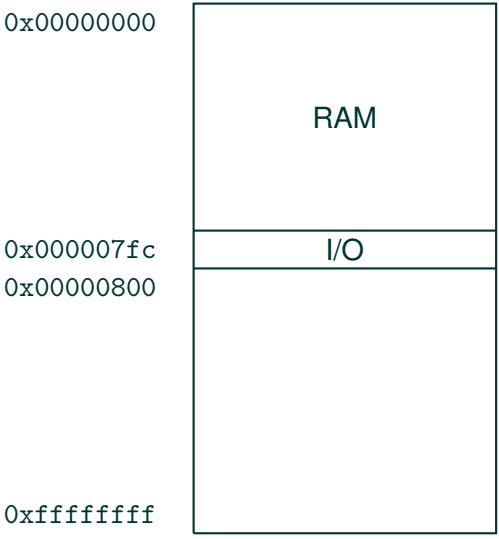
Input and Output

- We need a way to communicate with the outside world



Input and Output

- We need a way to communicate with the outside world
- Solution: add a memory-mapped Input/Output device



Using I/O

Our original program:

```
.org 0x00
  # read from memory
  LW x1, 0x20(x0)
  LW x2, 0x24(x0)
  ADD x3, x1, x2
  # write to memory
  SW x3, 0x24(x0)
  EBREAK
```

Using I/O

The same program with I/O:

```
.org 0x00
  # read from I/O
  LW x1, 0x7fc(x0)
  LW x2, 0x7fc(x0)
  ADD x3, x1, x2
  # write to I/O
  SW x3, 0x7fc(x0)
  EBREAK
```

A Second Program

- Read 10 values from I/O
- Sum them up
- Write the result to I/O
- Halt the CPU

- Note: Nano RISC-V has no way to create loops!

A Second Program

```
.org 0x00
  ADD x1, x0, x0 # clear x1

  LW x2, 0x7fc(x0) # load input
  ADD x1, x1, x2   # x1 += input

  LW x2, 0x7fc(x0) # load input
  ADD x1, x1, x2   # x1 += input

  LW x2, 0x7fc(x0) # load input
  ADD x1, x1, x2   # x1 += input

  LW x2, 0x7fc(x0) # load input
  ADD x1, x1, x2   # x1 += input

  LW x2, 0x7fc(x0) # load input
  ADD x1, x1, x2   # x1 += input

  LW x2, 0x7fc(x0) # load input
  ADD x1, x1, x2   # x1 += input

  LW x2, 0x7fc(x0) # load input
  ADD x1, x1, x2   # x1 += input

  SW x1, 0x7fc(x0) # output sum
  EBREAK
```

Nano RISC-V Summary

Nano RISC-V is a simple 32-bit RISC-V processor

- 1 Zero Register: `x0`
- 31 General Purpose Registers: `x1` - `x31`
- 2 ALU Instructions (`OP rd, rs1, rs2`)
  - ADD, AND
- Load Instruction (`LW rd, offset(rs1)`)
- Store Instruction (`SW rs2, offset(rs1)`)
- Halt Instruction (`EBREAK`)
- Almost 2 KiB of Memory (`0x000` - `0x7fc`)
- Memory-mapped I/O at address `0x7fc`

Chapter 4b: Building a Nano RISC-V CPU

Stefan Mangard

October 22, 2019

Computer Organization and Networks
Graz University of Technology