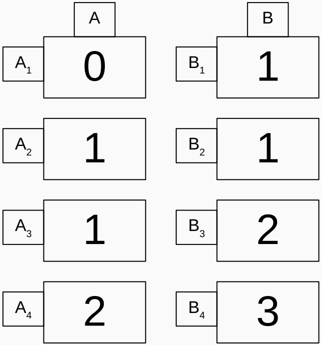
Zero Knowledge Virtual Machine

Jordi Baylina Héctor Masip Jose L. Muñoz-Tapia

Polygon-Hermez
Information Security Group, Universitat Politècnica de Catalunya (UPC)

Table of Contents

Table of Contents



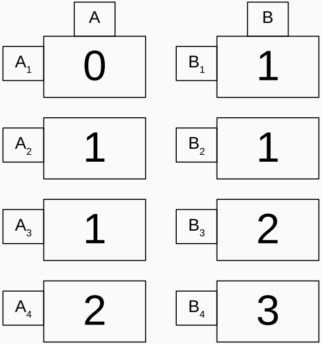
- We can build the Fibonacci state machine with two registries, A and B.
- In the Fibonacci sequence, we have the following relations between the states of these registries:

$$A_{i+1} = B_i,$$
$$B_{i+1} = A_i + B_i.$$

- Let's represent the states of these registries for four steps as polynomials in $\mathbb{Z}_p[x]$ evaluated on the group $H = \{\omega, \omega^2, \omega^3, \omega^4 = 1\}$:

$$A(\omega^i) = A_i \quad \Longrightarrow \quad A = [0, 1, 1, 2]$$
$$B(\omega^i) = B_i \quad \Longrightarrow \quad B = [1, 1, 2, 3]$$

First Example: The Fibonacci State Machine ii



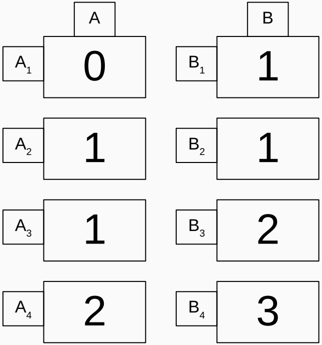
- The relations between the states of registries:

$$
\begin{aligned}
A_{i+1} &= B_i, \\
B_{i+1} &= A_i + B_i,
\end{aligned}
$$

  for $i \in [4]$.

- Are translated into relations (A.K.A identities) in the polynomial setting:

$$
\begin{aligned}
A(x\omega) &= \Big|_H B(x), \\
B(x\omega) &= \Big|_H A(x) + B(x).
\end{aligned}
$$



- So we have:

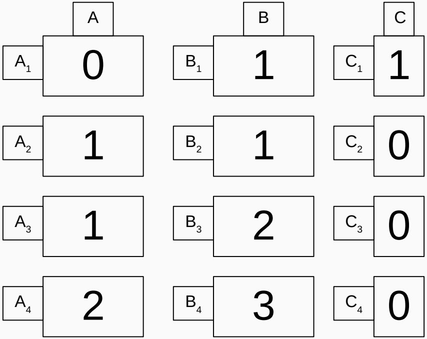
$$A(x\omega) = \Big|_H B(x), \quad B(x\omega) = \Big|_H A(x) + B(x).$$

- However, the previous identities do not correctly and uniquely describe our sequence because:
  1. When we evaluate the identities in ω^4:

$$A(\omega^5) = A(\omega) = 0 \neq 3 = B(\omega^4),$$
$$B(\omega^5) = B(\omega) = 1 \neq 5 = A(\omega^4) + B(\omega^4).$$

     The equations are not cyclic.
  2. Other initial conditions also fulfill the identities, e.g: $(2, 3), (3, 5), (5, 8), (8, 13)$.

First Example: The Fibonacci State Machine iv



- Let's add an auxiliary registry C to solve these problems.
- The corresponding polynomial is:

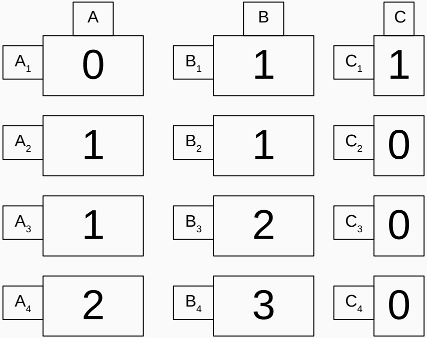
$$C(\omega^i) = C_i \quad \Longrightarrow \quad C = [1, 0, 0, 0].$$

- With this auxiliary registry, we can now fix the polynomial identities as follows:

$$A(x\omega) = \Big|_H B(x)(1 - C(x\omega)),$$

$$B(x\omega) = \Big|_H (A(x) + B(x))(1 - C(x\omega)) + C(x\omega).$$

First Example: The Fibonacci State Machine v



$C(x)$ is publicly known (A.K.A **pre-processed** or **constant**).

- Note that now at $x = w^4$ the identities are satisfied:

$$A(x\omega) = \Big|_{H} B(x)(1 - C(x\omega)),$$

$$B(x\omega) = \Big|_{H} (A(x) + B(x))(1 - C(x\omega)) + C(x\omega).$$

$$A(\omega^4\omega) = A(\omega^5) = A(\omega) = 0,$$

$$B(\omega^4\omega) = B(\omega^5) = B(\omega) = 1.$$

- We can also use other initial conditions (A_0, B_0):

$$A(x\omega) = \Big|_{H} B(x)(1 - C(x\omega)) + A_0 C(x\omega),$$

$$B(x\omega) = \Big|_{H} (A(x) + B(x))(1 - C(x\omega)) + B_0 C(x\omega).$$

Proving our State Machine (High Level) i

$$p_1(x) = A(x\omega) - B(x)(1 - C(x\omega)) - A_0 C(x\omega) = \Big|_H 0,$$

$$p_2(x) = B(x\omega) - (A(x) + B(x))(1 - C(x\omega)) - B_0 C(x\omega) = \Big|_H 0.$$

- We are going to convert these H-ranged identities into an $\mathbb{F}$-ranged identities that is valid for any $x \in \mathbb{F}$.
- To do so, we are going to use the **zero polynomial** $Z_H(x)$.
- $Z_H(x)$ is computed as the polynomial that is zero in H:

$$(\omega, 0), (\omega^2, 0), (\omega^3, 0), (\omega^4, 0) \quad \Longrightarrow \quad Z_H(x) = (x - \omega)(x - \omega^2)(x - \omega^3)(x - \omega^4) = x^4 - 1.$$

Proving our State Machine (High Level) ii

- Notice that $p_1(x)$ and $p_2(x)$ have roots at H.
- That means $Z_H(x)|p_1(x)$ and $Z_H(x)|p_2(x)$ because $(x-\omega)$, $(x-\omega^2)$, etc. are monomials of $p_1(x)$ and $p_2(x)$.
- Now we can compute $d_1(x) = p_1(x)/Z_H(x)$ and $d_2(x) = p_2(x)/Z_H(x)$.
- The identities that need to be checked are $p_1(x) - d_1(x)Z_H(x) = 0$ and $p_2(x) - d_2(x)Z_H(x) = 0$ for any $x \in \mathbb{F}$.

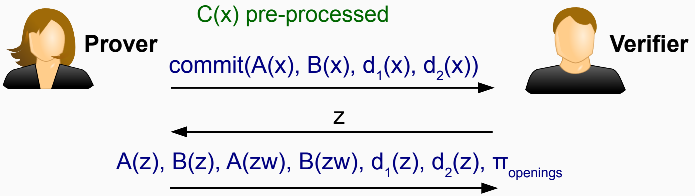
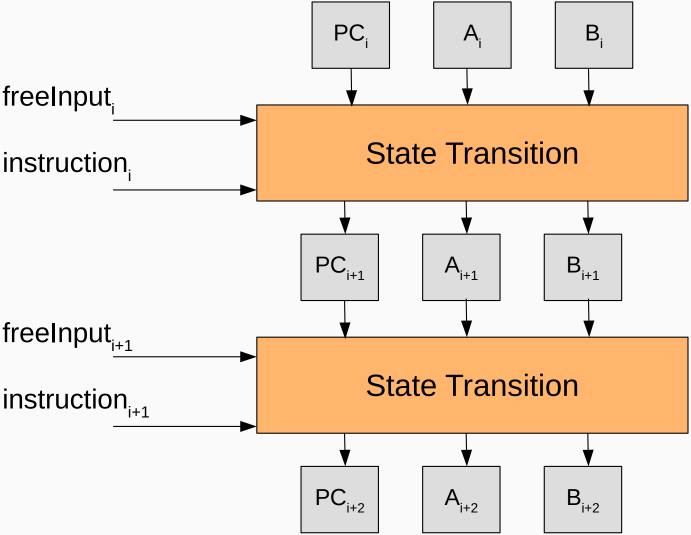


Table of Contents

Simplified State Machine



- Let's assume we have the following program:

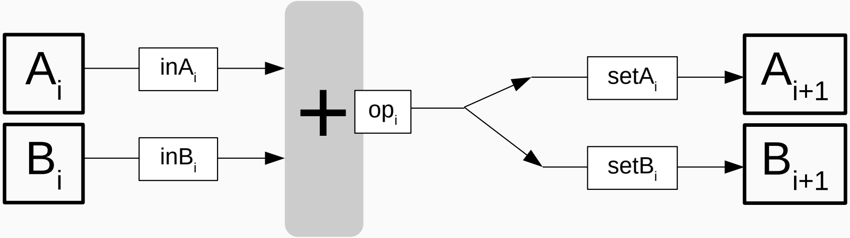
Position	Instruction	
0	MOV	$A, 7$
1	JMP $(if\ A = 0)$	5
2	MOV	$B, 3$
3	MOV	$A, 0$
4	JMP	1
5	STOP	$\varnothing$

Example Program: Moving & Jumping ii

- This program has the following trace:

Position	Instruction		PC_i	A_i	B_i	PC_{i+1}	A_{i+1}	B_{i+1}
0	MOV	$A, 7$	0	0	0	1	7	0
1	JMP ($\mathit{if}\ A = 0$)	5	1	7	0	2	7	0
2	MOV	$B, 3$	2	7	0	3	7	3
3	MOV	$A, 0$	3	7	3	4	0	3
4	JMP	1	4	0	3	1	0	3
5	JMP ($\mathit{if}\ A = 0$)	5	1	0	3	5	0	3
6	STOP	$\varnothing$	5	0	3	5	0	3

Expressing the Relations between the States i



- Here, we use the following notation:
    a) **inX**$_i$: 1 or 0 depending if the state X_i is included in the sum or not.
    b) **op**$_i$: The resulting operation between the included states.
    c) **setX**$_i$: 1 or 0 depending if op_i will be moved into X_{i+1}.

Expressing the Relations between the States ii

- The relations between the states of the registries can be expressed as follows:

$$
\begin{aligned}
\mathrm{op}_i &= A_i \cdot \mathrm{in}A_i + B_i \cdot \mathrm{in}B_i + FREE_i \cdot \mathrm{in}FREE_i, \\
A_{i+1} &= \mathrm{set}A_i \cdot (\mathrm{op}_i - A_i) + A_i, \\
B_{i+1} &= \mathrm{set}B_i \cdot (\mathrm{op}_i - B_i) + B_i, \\
PC_{i+1} &= PC_i + 1 + (isJMP_i + isJMPC_i \cdot isSatisfied_i) \cdot (dest - PC_i - 1).
\end{aligned}
$$

- Here:
    1. *FREE* is the second input passed to the **MOV** instruction.
    2. *dest* is the input passed to the **JMP** or the **JMPC** (conditioned) instructions.

- Let's now explain how to encode the instructions included in the program:

$$\text{MOV } A, 7 \quad \text{JMP } (\textit{if } A = 0)\ 5 \quad \text{MOV } B, 3 \quad \text{MOV } A, 0 \quad \text{JMP } 1 \quad \text{STOP}$$

Instruction		inA	inB	inFREE	setA	setB	isJMP	isJMPC	FREE	dest	Inst. Value
MOV	$A, 7$	1	0	1	1	0	0	0	7	0	0000.0111.0001101
JMP $(\textit{if } A = 0)$	5	0	0	0	0	0	0	1	0	5	0101.0000.1000000
MOV	$B, 3$	0	1	1	1	0	0	0	3	0	0000.0011.0001110
JMP	1	0	0	0	0	0	0	1	0	1	0001.0000.1000000
STOP	$\varnothing$	0	0	0	0	0	0	0	0	0	0000.0000.0000000

- Here, we computed the instruction values as follows:

$$\begin{aligned} \text{inst}_i = {} & \text{in}A_i + 2 \cdot \text{in}B_i + 2^2 \cdot \text{in}FREE_i + 2^3 \cdot \text{set}A_i + 2^4 \cdot \text{set}B_i + 2^5 \cdot isJMP_i + 2^6 \cdot isJMPC_i \\ & + 2^{10} \cdot FREE_i + 2^{14} \cdot dest_i. \end{aligned}$$

How to Encode the Move State Machine ii

- We can write the previous table values as the following polynomial identity:

$$\begin{aligned}\mathrm{inst}(x) = \mathrm{in}A(x) + 2 \cdot \mathrm{in}B(x) + 2^2 \cdot \mathrm{in}FREE(x) + 2^3 \cdot \mathrm{set}A(x) + 2^4 \cdot \mathrm{set}B(x) + 2^5 \cdot isJMP(x) \\ + 2^6 \cdot isJMPC(x) + 2^{10} \cdot FREE(x) + 2^{14} \cdot dest(x).\end{aligned}$$

- Now, to build the program, every instruction will be uniquely identified by its value and the position of the program in which it is executed.
- We define the polynomial $\mathrm{rom}(x)$ which consists on an instruction value concatenated with its position:

$$\mathrm{rom}(x) = \mathrm{inst}(x) + 2^{18} \cdot position(x)$$

- With the support of this encoding, now we can compute the whole trace of the execution of this program:

Position	Instruction		$\mathrm{Rom_i} = \mathrm{inst_i} + 2^{18} \cdot \mathrm{position_i}$
0	MOV	$A, 7$	0000.0000.0111.0001101
1	JMP (*if* $A = 0$)	5	0001.0101.0000.1000000
2	MOV	$B, 3$	0010.0000.0011.0001110
3	MOV	$A, 0$	0011.0000.0000.0001101
4	JMP	1	0100.0001.0000.1000000
5	STOP	$\varnothing$	0101.0000.0000.0000000

Representing the State Machine ii

- We can do the same with the trace of the program:

Position	Instruction		PC_i	A_i	B_i	$\text{instTrace}_i = \text{inst}_i + 2^{18} \cdot PC_i$
0	MOV	$A, 7$	0	0	0	0000.0000.0111.0001101
1	JMP (*if* $A = 0$)	5	1	7	0	0001.0101.0000.1000000
2	MOV	$B, 3$	2	7	0	0010.0000.0011.0001110
3	MOV	$A, 0$	3	7	3	0011.0000.0000.0001101
4	JMP	1	4	0	3	0100.0001.0000.1000000
5	JMP (*if* $A = 0$)	5	1	0	3	0001.0101.0000.1000000
6	STOP	$\varnothing$	5	0	3	0000.0000.0000.0000000

Checking the Correct Program Execution i

- The question that arises now is:

How do we actually verify that we are executing the correct program?

- The solution seems obvious: Check that every row of the trace of the execution coincides with some row of the program.
- Then, the question becomes to:

How do we actually verify that we are executing the correct program in an efficient manner?

Checking the Correct Program Execution ii

- We can do it with the Plookup protocol!
- So, to check that the correct program is being executed, we simply have to use Plookup to determine if:

$$\mathsf{instTrace} \subset \mathsf{Rom}$$

- In simple words, the trace being executed is an execution of the actual program if the instruction trace is contained in the ROM of the program.

Strategy to Follow

- The strategy to follow to check the correct execution of a program is:
    1. Encode each distinct instruction of the program in a deterministic and efficient manner.
    2. Represent each instruction in the program in a unique manner by appending the position in the program to it. We obtain the polynomial $rom(x)$.
    3. Similarly, represent each instruction in the trace of the program by appending the program counter to it.We obtain the polynomial $instTrace(x)$.
    4. Use Plookup to prove that the trace of the program is contained in the rom of the program, i.e., prove that $instTrace \subset rom$.

Non-Naive Example Program i

- Let's now work with a real program:

Position	Instruction	
0	FREELOAD	A
1	MOV	$B, 3$
2	JMP ($\mathit{if}\ B = 0$)	6
3	MUL	A, A
4	DEC	B
5	JMP	2
6	STOP	$\varnothing$

Non-Naive Example Program ii

- This program has the following trace:

Position	Instruction		freeLoad	PC_i	A_i	B_i	PC_{i+1}	A_{i+1}	B_{i+1}
0	FREELOAD	A	10	0	0	0	1	10	0
1	MOV	$B, 3$	0	1	10	0	2	10	3
2	JMP (*if* $B = 0$)	6	0	2	10	3	3	10	3
3	MUL	A, A	0	3	10	3	4	100	3
4	DEC	B	0	4	100	3	5	100	2
5	JMP	2	0	5	100	2	2	100	2
6	JMP (*if* $B = 0$)	6	0	2	100	2	3	100	2
7	MUL	A, A	0	3	100	2	4	1000	2
8	DEC	B	0	4	1000	2	5	1000	1
9	JMP	2	0	5	1000	1	2	1000	1
10	JMP (*if* $B = 0$)	6	0	2	1000	1	3	1000	1
11	MUL	A, A	0	3	1000	1	4	10000	1
12	DEC	B	0	4	10000	1	5	10000	0
13	JMP	2	0	5	10000	0	2	10000	0
14	JMP (*if* $B = 0$)	6	0	2	10000	0	6	10000	0
15	STOP	$\varnothing$	0	6	10000	0	6	10000	0

Non-Naive Example Program iii

- First, we encode each instruction in hexadecimal as follows:

$$\begin{aligned} \text{FREELOAD } A &\to 0x00010000 \\ \text{MOV } B, n &\to 0x00020000 + n \\ \text{JMP } (\textit{if } B = 0)\ n &\to 0x00040000 + n \\ \text{JMP } n &\to 0x00080000 + n \\ \text{MUL } A, A &\to 0x00100000 \\ \text{DEC } B &\to 0x00200000 \\ \text{STOP} &\to 0x00400000 \end{aligned}$$

Position	Instruction		Inst. Value
0	FREELOAD	A	0x00010000
1	MOV	$B, 3$	0x00020003
2	JMP (*if* $B = 0$)	6	0x00040006
3	MUL	A, A	0x00100000
4	DEC	B	0x00200000
5	JMP	2	0x00080002
6	STOP	$\varnothing$	0x00400000

Checking the Correct Program Execution i

- To prove the correct execution of the program, we have to prove that the trace of the trace is contained in the rom of the program, hence:

Position	Instruction		Inst. Value	$\text{Rom}_i = \text{inst}_i + 2^{32} \cdot \text{position}_i$
0	FREELOAD	A	0x00010000	0x0.00010000
1	MOV	$B, 3$	0x00020003	0x1.00020003
2	JMP ($if\ B = 0$)	6	0x00040006	0x2.00040006
3	MUL	A, A	0x00100000	0x3.00100000
4	DEC	B	0x00200000	0x4.00200000
5	JMP	2	0x00080002	0x5.00080002
6	STOP	$\varnothing$	0x00400000	0x6.00400000

Checking the Correct Program Execution ii

- On the other side:

Position	Instruction		Inst. Value	freeLoad	PC	A	B	$\text{instTrace}_i = \text{inst}_i + 2^{32} \cdot \text{PC}_i$
0	FREELOAD	A	0x00010000	10	0	0	0	0x0.00010000
1	MOV	$B, 3$	0x00020003	0	1	10	0	0x1.00020003
2	JMP (*if* $B = 0$)	6	0x00040006	0	2	10	3	0x2.00040006
3	MUL	A, A	0x00100000	0	3	10	3	0x3.00100000
4	DEC	B	0x00200000	0	4	100	3	0x4.00200000
5	JMP	2	0x00080002	0	5	100	2	0x5.00080002
6	JMP (*if* $B = 0$)	6	0x00040006	0	2	100	2	0x2.00040006
7	MUL	A, A	0x00100000	0	3	100	2	0x3.00100000
8	DEC	B	0x00200000	0	4	1000	2	0x4.00200000
9	JMP	2	0x00080002	0	5	1000	1	0x5.00080002
10	JMP (*if* $B = 0$)	6	0x00040006	0	2	1000	1	0x2.00040006
11	MUL	A, A	0x00100000	0	3	1000	1	0x3.00100000
12	DEC	B	0x00200000	0	4	10000	1	0x4.00200000
13	JMP	2	0x00080002	0	5	10000	0	0x5.00080002
14	JMP (*if* $B = 0$)	6	0x00040006	0	2	10000	0	0x2.00040006
15	STOP	$\varnothing$	0x00400000	0	6	10000	0	0x6.00400000

Table of Contents

μVM Architecture

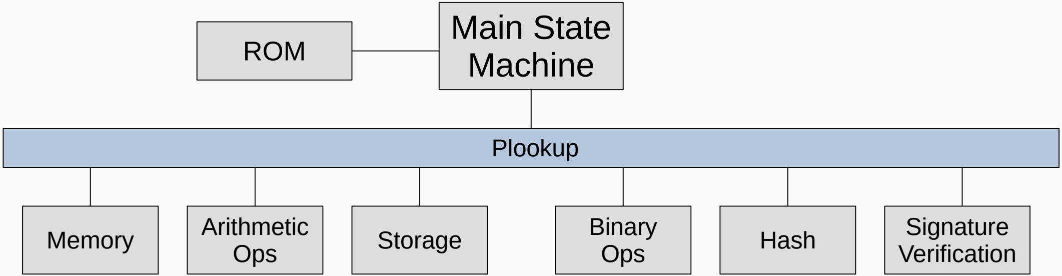


Table of Contents

Table of Contents

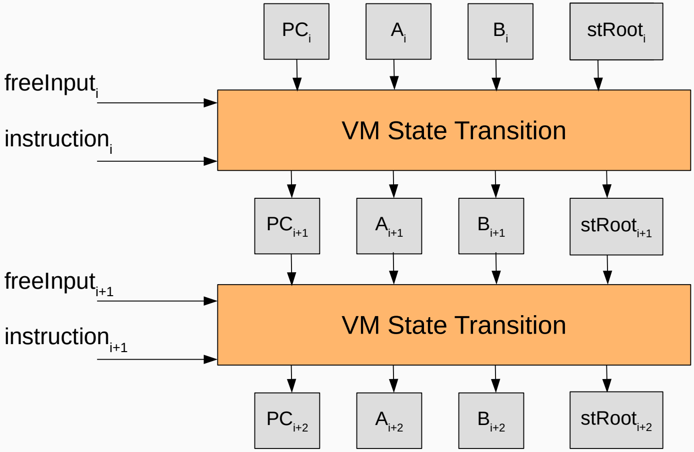
TBD

- Hola

Table of Contents

Table of Contents

Main State Machine of a Virtual Machine



Main State Machine of a Virtual Machine

Position	Instruction		freeLoad	stRoot	A	B	oldStRoot	newStRoot	Key	Value
0				st1			0	0	0	0
1				st1			0	0	0	0
2				st1			0	0	0	0
3	SSTORE	$[A], B$	st2	st1	$0x4C76$	1232	st1	st2	$0x4C76$	1232
4				st2	$0x4C76$	1232	0	0	0	0
5				st2			0	0	0	0
6				st2			0	0	0	0
7	SSTORE	$[A], B$	st3	st2	$0x8E12$	7765	st2	st3	$0x8E12$	7765
8				st3	$0x8E12$	7765	0	0	0	0
9				st3			0	0	0	0
10	SSTORE	$[A], B$	st4	st3	$0xAA23$	9812	st3	st4	$0xAA23$	9812
11				st4	$0xAA23$	9812	0	0	0	0
12				st4			0	0	0	0
13				st4			0	0	0	0
14	SSTORE	$[A], B$	st5	st4	$0x2213$	8610	st4	st5	$0x2213$	8610
15				st5	$0x2213$	8610	0	0	0	0

Table of Contents

Memory in the Main State Machine

Position	Instruction		freeLoad	A	B	mRead	mWrite	Address	Value
0						0	0	0	0
1						0	0	0	0
2						0	0	0	0
3	MWRITE	$[A], B$		$0x4C76$	1232	0	1	$0x4C76$	1232
4				$0x4C76$	1232	0	0	0	0
5	MREAD	$B, [A]$	1232	$0x4C76$	1232	1	0	$0x4C76$	1232
6				$0x4C76$	1232	0	0	0	0
7	MWRITE	$[A], B$		$0x8E12$	7765	0	1	$0x8E12$	7765
8				$0x8E12$	7765	0	0	0	0
9						0	0	0	0
10	MWRITE	$[A], B$		$0x2213$	8610	0	1	$0x2213$	8610
11				$0x2213$	8610	0	0	0	0
12						0	0	0	0
13						0	0	0	0
14	MREAD	$B, [A]$	7765	$0x8E12$	7765	1	0	$0x8E12$	7765
15				$0x8E12$	7765	0	0	0	0

Memory State Machine

Free Inputs					Intermediary State		Results
Position	mRead	mWrite	Address	ValueIn	stOld	stNew	Value
3	0	1	$0x4C76$	1232	0	1232	1232
5	1	0	$0x4C76$		1232	1232	1232
7	0	1	$0x8E12$	7765	1232	7765	7765
14	1	0	$0x8E12$		7765	7765	7765
10	0	1	$0x2213$	8610	7765	8610	8610

- Using Plookup, prove that the polynomial:

  $$main.position(x)+v\cdot main.mRead(x)+v^2\cdot main.mWrite(x)+v^3\cdot main.Address(x)+v^4\cdot main.Value(x),$$

  is included in the polynomial:

  $$mem.position(x)+v\cdot mem.mRead(x)+v^2\cdot mem.mWrite(x)+v^3\cdot mem.Address(x)+v^4\cdot mem.Value(x).$$

Table of Contents

Checking Binary Operations: XOR

<u>Operation</u>

$$f(x) \oplus g(x) = h(x).$$

1. Check byte decomposition:

$$f(x) = f_0(x) + 2^8 f_1(x) + 2^{16} f_2(x) + \dots$$
$$g(x) = g_0(x) + 2^8 g_1(x) + 2^{16} g_2(x) + \dots$$
$$h(x) = h_0(x) + 2^8 h_1(x) + 2^{16} h_2(x) + \dots$$

2. Check byte form elementwise:

$$\begin{array}{ccc} f_0(x) \subset \mathrm{byte(x)} & g_0(x) \subset \mathrm{byte(x)} & h_0(x) \subset \mathrm{byte(x)} \\ f_1(x) \subset \mathrm{byte(x)} & g_1(x) \subset \mathrm{byte(x)} & h_1(x) \subset \mathrm{byte(x)} \\ f_2(x) \subset \mathrm{byte(x)} & g_2(x) \subset \mathrm{byte(x)} & h_2(x) \subset \mathrm{byte(x)} \\ \vdots & \vdots & \vdots \end{array}$$

3. Check XOR operation:

$$f_0(x) + 2^8 g_0(x) + 2^{16} h_0(x) \subset \mathrm{XOR}(x)$$
$$f_1(x) + 2^8 g_1(x) + 2^{16} h_1(x) \subset \mathrm{XOR}(x)$$
$$f_2(x) + 2^8 g_2(x) + 2^{16} h_2(x) \subset \mathrm{XOR}(x)$$
$$\vdots$$

x	byte
ω^0	0x00
ω^1	0x01
$\vdots$	$\vdots$
ω^{123}	0x7B
$\vdots$	$\vdots$
ω^{255}	0xFF

x	XOR
ω^0	0x000000
ω^1	0x010001
$\vdots$	$\vdots$
ω^{5028}	0xB713A4
$\vdots$	$\vdots$
ω^{65535}	0x00FFFF

Table of Contents

Checking Arithmetic Operations: Multiplication

				8 Bytes a_3	8 Bytes a_2	8 Bytes a_1	8 Bytes a_0
				8 Bytes b_3	8 Bytes b_2	8 Bytes b_1	8 Bytes b_0
				8 Bytes c_3	8 Bytes c_2	8 Bytes c_1	8 Bytes c_0
8 Bytes d_7	8 Bytes d_6	8 Bytes d_5	8 Bytes d_4	8 Bytes d_3	8 Bytes d_2	8 Bytes d_1	8 Bytes d_0
8 Bytes e_3	8 Bytes e_2	8 Bytes e_1	8 Bytes e_0	8 Bytes f_3	8 Bytes f_2	8 Bytes f_1	8 Bytes f_0

$$
\begin{aligned}
A &= a_3 \cdot 256^{24} + a_2 \cdot 256^{16} + a_1 \cdot 256^{8} + a_0,\\
B &= b_3 \cdot 256^{24} + b_2 \cdot 256^{16} + b_1 \cdot 256^{8} + b_0,\\
C &= c_3 \cdot 256^{24} + c_2 \cdot 256^{16} + c_1 \cdot 256^{8} + c_0,\\
D &= d_7 \cdot 256^{56} + d_6 \cdot 256^{48} + d_5 \cdot 256^{40} + d_4 \cdot 256^{32}\\
&\quad + d_3 \cdot 256^{24} + d_2 \cdot 256^{16} + d_1 \cdot 256^{8} + d_0,\\
E &= e_3 \cdot 256^{24} + e_2 \cdot 256^{16} + e_1 \cdot 256^{8} + e_0,\\
F &= f_3 \cdot 256^{24} + f_2 \cdot 256^{16} + f_1 \cdot 256^{8} + f_0,
\end{aligned}
$$

$$A \cdot B + C = D = E \cdot 2^{256} + F$$

$$
\begin{aligned}
&d_0 = f_0, d_1 = f_1, d_2 = f_2,\\
&d_2 = f_2 + \text{carry}_1 \cdot 256^{10},\\
&d_3 \cdot 256 + \text{carry}_1 = e_0 + \text{carry}_2 \cdot 256^{11},\\
&d_4 \cdot 256 + \text{carry}_2 = e_1 + \text{carry}_3 \cdot 256^{11},\\
&d_5 \cdot 256 + \text{carry}_3 = e_1.
\end{aligned}
$$

$$\text{carry}_1, \text{carry}_2, \text{carry}_3 \subset \text{byte}.$$

Multiplying

Step	mA	mB	acc_5	acc_4	acc_3	acc_2	acc_1	acc_0
0	a_0	b_0	d_5	d_4	d_3	d_2	d_1	d_0
1	a_0	b_1	d_0	d_5	d_4	d_3	d_2	d_1
2	a_1	b_0	d_0	d_5	d_4	d_3	d_2	d_1
3	a_0	b_2	d_1	d_0	d_5	d_4	d_3	d_2
4	a_1	b_1	d_1	d_0	d_5	d_4	d_3	d_2
5	a_2	b_0	d_1	d_0	d_5	d_4	d_3	d_2
6	a_1	b_2	d_2	d_1	d_0	d_5	d_4	d_3
7	a_2	b_1	d_2	d_1	d_0	d_5	d_4	d_3
8	a_2	b_2	d_3	d_2	d_1	d_0	d_5	d_4

q_{shift}	q_{same}	q_{set}	q_{result}	q_{a_0}	q_{a_1}	q_{a_2}	q_{b_0}	q_{b_1}	q_{b_2}
0	0	1	0	1	0	0	1	0	0
1	0	0	0	1	0	0	0	1	0
0	1	0	0	0	1	0	1	0	0
1	0	0	0	1	0	0	0	0	1
0	1	0	0	0	1	0	0	1	0
0	1	0	0	0	0	1	1	0	0
1	0	0	0	0	1	0	0	0	1
0	1	0	0	0	0	1	0	1	0
1	0	0	1	0	0	1	0	0	1

Table of Contents

Prover Workflow