Zero Knowledge Ethereum Virtual Machine

Jordi Baylina Héctor Masip-Ardevol Jose L. Muñoz-Tapia

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

Table of Contents

Introduction

μVM Architecture

Main State Machine

- Storage
- Memory
- Binary Operations
- Arithmetic Operations

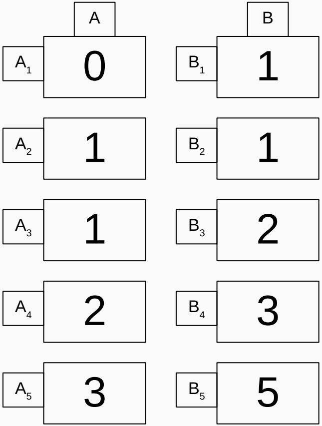
Table of Contents

Introduction

μVM Architecture

Main State Machine

First Example: The Fibonacci Sequence i



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

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

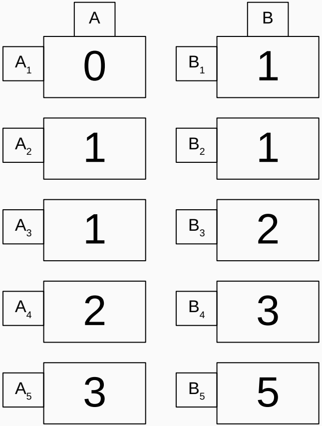
for $i \in [5]$.

- Now, represent these states as polynomials evaluated on the group $H = \{\omega, \omega^2, \omega^3, \omega^4, \omega^5 = 1\}$:

$$
\begin{aligned}
A(\omega^i) = A_i &\quad \Longrightarrow \quad A = [0, 1, 1, 2, 3] \\
B(\omega^i) = B_i &\quad \Longrightarrow \quad B = [1, 1, 2, 3, 5]
\end{aligned}
$$

for $i \in [5]$.

First Example: The Fibonacci Sequence ii



- We can now translate the previous relations to the polynomial setting:

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

- However, this is not completely correct, since when we evaluate in ω^5 we obtain:

$$
\begin{aligned}
A(\omega^6) &= A(\omega) = 0 \neq 5 = B(\omega^5), \\
B(\omega^6) &= B(\omega) = 1 \neq 8 = A(\omega^5) + B(\omega^5).
\end{aligned}
$$

- Let's add an auxiliary registry C to solve this problem.
- To create simple polynomial identities that can can be described as relations between successive points in H, we will make the state machine cyclic, that is, to start again in (0,1).

First Example: The Fibonacci Sequence iii

	A		B		C
A_1	0	B_1	1	C_1	1
A_2	1	B_2	1	C_2	0
A_3	1	B_3	2	C_3	0
A_4	2	B_4	3	C_4	0
A_5	3	B_5	5	C_5	0

- Similarly to A and B, represent the state C as a polynomial evaluated on H:

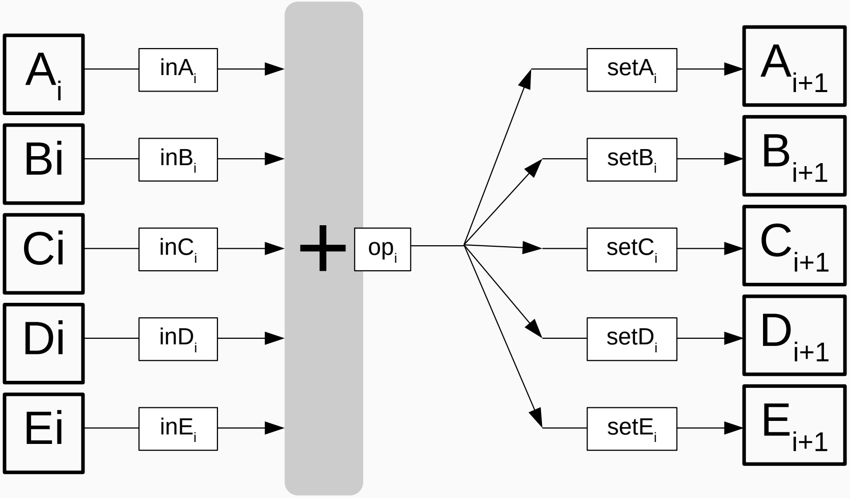
$$C(\omega^i) = C_i \quad \forall i \in [5] \quad \Longrightarrow \quad C = [1, 0, 0, 0, 0].$$

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

$$A(x\omega) = B(x)(1 - C(x\omega)),$$
$$B(x\omega) = (A(x) + B(x))(1 - C(x\omega)) + C(x\omega).$$

Notice that now, when $x = w^5$:
$A(Xw) = A(w^6) \neq B(X)$; $A(Xw) = A(w^6) = 0.$
$B(Xw) = B(w^6) \neq A(X) + B(X)$; $B(Xw) = B(w^6) = 1.$

Starting from the Basics: Move State Machine i



Starting from the Basics: Move State Machine ii

- We have used the following notation:
  a) **inX**: 1 or 0 depending if the state X_i is included in the sum or not.
  b) **op**: The resulting operation between the included states.
  c) **setX**: 1 or 0 depending if one state (or a combination or more) will be moved into X_{i+1}.
- The relations between the states of the registries can be expressed as follows:

$$\begin{aligned}
\mathrm{op}_i &= A_i \cdot \mathrm{in}A_i + B_i \cdot \mathrm{in}B_i + C_i \cdot \mathrm{in}C_i + D_i \cdot \mathrm{in}D_i + E_i \cdot \mathrm{in}E_i,\\
A_{i+1} &= \mathrm{set}A_i \cdot (\mathrm{op}_i - A_i) + A_i,\\
B_{i+1} &= \mathrm{set}B_i \cdot (\mathrm{op}_i - B_i) + B_i,\\
C_{i+1} &= \mathrm{set}C_i \cdot (\mathrm{op}_i - C_i) + C_i,\\
D_{i+1} &= \mathrm{set}D_i \cdot (\mathrm{op}_i - D_i) + D_i,\\
E_{i+1} &= \mathrm{set}E_i \cdot (\mathrm{op}_i - E_i) + E_i.
\end{aligned}$$

How to Encode the Move State Machine i

- Let's assume that we want to perform the following instructions:

$$\textbf{MOV } B, A \quad \textbf{MOV } C, D \quad \textbf{MOV } A, D \quad \textbf{MOV } E, B.$$

Position	in$A$	in$B$	in$C$	in$D$	in$E$	set$A$	set$B$	set$C$	set$D$	set$E$	Inst. Value
0	1	0	0	0	0	0	1	0	0	0	0x41
1	0	0	0	1	0	0	0	1	0	0	0x88
2	0	0	0	1	0	1	0	0	0	0	0x28
3	0	1	0	0	0	0	0	0	0	1	0x202

- We code the instruction value as follows:

$$\text{inst} = \text{in}A + 2\cdot\text{in}B + 2^2\cdot\text{in}C + 2^3\cdot\text{in}D + 2^4\cdot\text{in}E + 2^5\cdot\text{set}A + 2^6\cdot\text{set}B + 2^7\cdot\text{set}C + 2^8\cdot\text{set}D + 2^9\cdot\text{set}E.$$

How to Encode the Move State Machine ii

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

$$\begin{aligned}\mathrm{inst}(x) = {} & \mathrm{in}A(x) + 2 \cdot \mathrm{in}B(x) + 2^2 \cdot \mathrm{in}C(x) + 2^3 \cdot \mathrm{in}D(x) + 2^4 \cdot \mathrm{in}E(x) \\ & + 2^5 \cdot \mathrm{set}A(x) + 2^6 \cdot \mathrm{set}B(x) + 2^7 \cdot \mathrm{set}C(x) + 2^8 \cdot \mathrm{set}D(x) + 2^9 \cdot \mathrm{set}E(x).\end{aligned}$$

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

Position	**Instruction**	**Inst. Value**	**Rom** $=$ **inst** $+ 2^{16} \cdot$ **position**
0	**MOV** B, A	0x0041	0x00041
1	**MOV** C, D	0x0088	0x10088
2	**MOV** A, D	0x0028	0x20028
3	**MOV** E, B	0x0202	0x30202

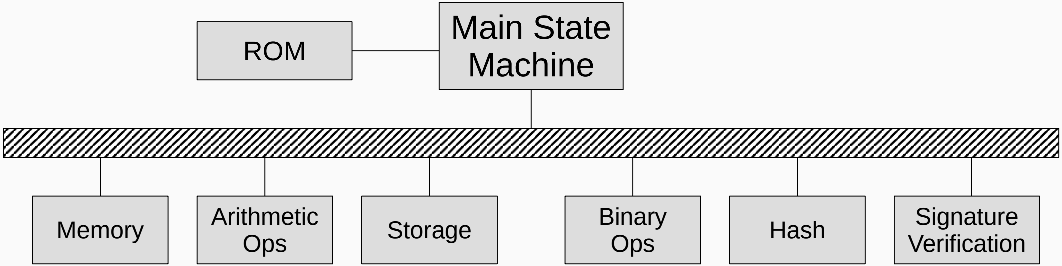
Table of Contents

Introduction

μVM Architecture

Main State Machine

μVM Architecture



All the relations between the different state machines are described as polynomial identities.

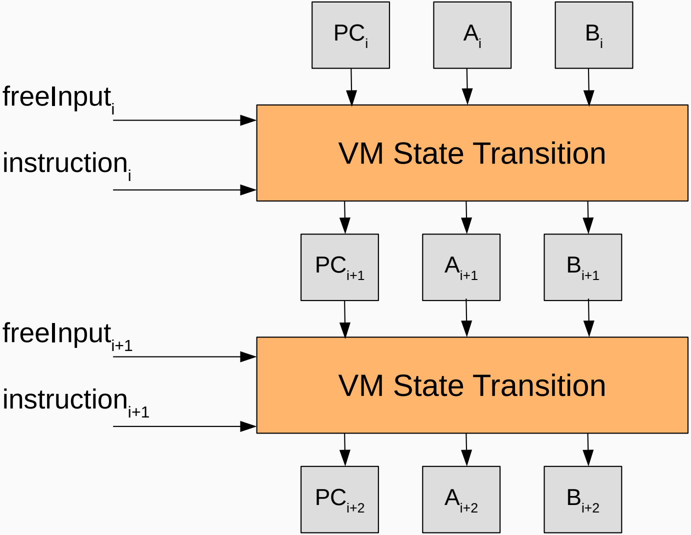
Table of Contents

Introduction

μVM Architecture

Main State Machine

Main State Machine of a Simplified Virtual Machine



Example Program i

- Let's now work with a real program:

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

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

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

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

Example Program iii

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

Position	Instruction		Inst. Value	freeLoad	PC	A	B
0	FREELOAD	A	0x00010000	10	0	0	0
1	MOV	$B, 3$	0x00020003	0	1	10	0
2	JMP (*if* $B = 0$)	6	0x00040006	0	2	10	3
3	MUL	A, A	0x00100000	0	3	10	3
4	DEC	B	0x00200000	0	4	100	3
5	JMP	2	0x00080002	0	5	100	2
6	JMP (*if* $B = 0$)	6	0x00040006	0	2	100	2
7	MUL	A, A	0x00100000	0	3	100	2
8	DEC	B	0x00200000	0	4	1000	2
9	JMP	2	0x00080002	0	5	1000	1
10	JMP (*if* $B = 0$)	6	0x00040006	0	2	1000	1
11	MUL	A, A	0x00100000	0	3	1000	1
12	DEC	B	0x00200000	0	4	10000	1
13	JMP	2	0x00080002	0	5	10000	0
14	JMP (*if* $B = 0$)	6	0x00040006	0	2	10000	0
15	STOP	$\varnothing$	0x00400000	0	6	10000	0

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?

- We can do it with Plookup!

Checking the Correct Program Execution ii

- On the one side:

Position	Instruction		Inst. Value	Rom $=$ inst $+ 2^{32} \cdot$ position
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 iii

- On the other side:

Position	Instruction		Inst. Value	freeLoad	PC	A	B	instTrace $= \text{inst} + 2^{32} \cdot \text{PC}$
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

Checking the Correct Program Execution iv

- 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.

Table of Contents

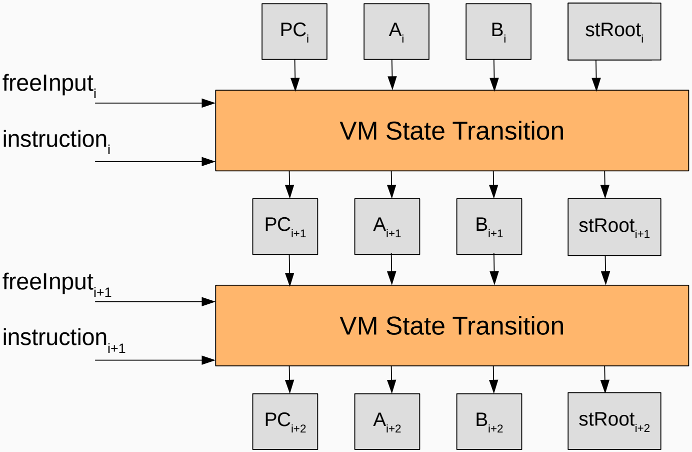
Introduction

μVM Architecture

Main State Machine

- Storage
- Memory
- Binary Operations
- Arithmetic Operations

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

Introduction

μVM Architecture

Main State Machine

- Storage
- Memory
- Binary Operations
- Arithmetic Operations

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

Introduction

μVM Architecture

Main State Machine

Storage

Memory

Binary Operations

Arithmetic Operations

Checking Binary Operations: XOR

Operation

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

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:

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

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

Introduction

μVM Architecture

Main State Machine

- Storage
- Memory
- Binary Operations
- Arithmetic Operations

Checking Arithmetic Operations: Multiplication

			10 Bytes $\mathbf{a_2}$	11 Bytes $\mathbf{a_1}$	11 Bytes $\mathbf{a_0}$
			10 Bytes $\mathbf{b_2}$	11 Bytes $\mathbf{b_1}$	11 Bytes $\mathbf{b_0}$
			10 Bytes $\mathbf{c_2}$	11 Bytes $\mathbf{c_1}$	11 Bytes $\mathbf{c_0}$
11 Bytes $\mathbf{d_5}$	11 Bytes $\mathbf{d_4}$	11 Bytes $\mathbf{d_3}$	11 Bytes $\mathbf{d_2}$	11 Bytes $\mathbf{d_1}$	11 Bytes $\mathbf{d_0}$
10 Bytes $\mathbf{e_2}$	11 Bytes $\mathbf{e_1}$	11 Bytes $\mathbf{e_0}$	10 Bytes $\mathbf{f_2}$	11 Bytes $\mathbf{f_1}$	11 Bytes $\mathbf{f_0}$

$$
\begin{aligned}
A &= a_2 \cdot 256^{22} + a_1 \cdot 256^{11} + a_0, \\
B &= b_2 \cdot 256^{22} + b_1 \cdot 256^{11} + b_0, \\
C &= c_2 \cdot 256^{22} + c_1 \cdot 256^{11} + c_0, \\
D &= d_5 \cdot 256^{55} + d_4 \cdot 256^{44} + d_3 \cdot 256^{33} \\
&\quad + d_2 \cdot 256^{22} + d_1 \cdot 256^{11} + d_0, \\
E &= e_2 \cdot 256^{22} + e_1 \cdot 256^{11} + e_0, \\
F &= f_2 \cdot 256^{22} + f_1 \cdot 256^{11} + f_0,
\end{aligned}
$$

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

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

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

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