In Situ Design of Register Operations

Serge Burckel, Emeric Gioan

In Situ Design of Register Operations

Serge Burckel
INRIA-LORIA
Campus Scientifique BP 239
54506 Vandoeuvre-lès-Nancy Cedex France
Email: serge.burckel@loria.fr

Emeric Gioan
CNRS-LIRMM
161 rue Ada
34392 Montpellier Cedex 5 France
Email: emeric.gioan@lirmm.fr

Abstract

We present methods to design programs or electronic circuits, for performing any operation on k registers of any sizes in a processor, in such a way that one uses no other working memory (such as other registers or external memories). In this way, any operation is performed with at most $4k-3$ assignments of these registers, or $2k-1$ when the operation is linear or bijective.

1 Introduction

The main motivation of this note is the optimization of processor performances. We present constructive mathematical results, described precisely in [1][2][3], that enable any operation on any registers of any sizes to be designed without using other working memory (like other registers or external memories).

The applications of such *in situ computations* (under patent [1]) concern directly Hardware (e.g. for the design of chips) and Software (e.g. for compiler optimizations).

In order to perform an operation Φ on $k > 1$ registers of 32 bits, since a 32 bits processor is able to make successive operations on at most 32 bits, one encounters a classical problem: during the transformation of these registers, their initial values will be modified and it could be impossible to complete the whole operation. For example, in order to exchange the contents of two registers R_1, R_2, if one begins by performing $R_1 := R_2$, half of the operation is done, but since the initial value of R_1 is lost, the second step is not possible anymore. A natural solution is to make a copy of some registers. For instance, this exchange is computed by the sequence $R_3 := R_1; R_1 := R_2; R_2 := R_3$ where R_3 is a third register or an external memory. Nevertheless, this kind of solution can generate some overflows, or use in a dramatic way the registers of the chip, or slowdown computations by external memory access. Globally, this solution by copies decreases the performances.

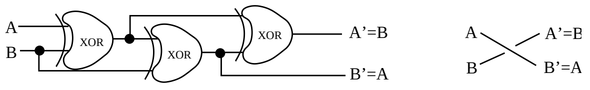

Figure 1. Exchange two bit values

In this note we present other solutions. For example, the previous exchange operation can be designed directly on R_1, R_2 by the following sequence of operations, performed on each bit index $j = 1, 2 \dots 32$, where R_i^j is the j-th bit of register R_i, and where $+$ is addition in $\mathbb{Z}/2\mathbb{Z}$ (i.e. an XOR gate):

$$R_1^j := R_1^j + R_2^j; R_2^j := R_1^j + R_2^j; R_1^j := R_1^j + R_2^j.$$

One obtains circuits like in Figure 1. Observe that this circuit is a *planar* alternative for the crossing of two wires (hence for any permutation of wires) and can be used for the design of planar electronic circuits.

Observe also that each 32 bit assignments of a register can be integrated in a single register assignment. We will say that the total sequence of bit assignments is *integrable* in register assignments. One obtains the sequence of register assignments:

$$R_1 := R_1 \oplus R_2; R_2 := R_1 \oplus R_2; R_1 := R_1 \oplus R_2.$$

where $\oplus$ stands for a bit to bit $+$ operation. The sequences of assignments considered in this note will all have this integrability property. That allows us to state the results in terms of bits and extend them directly in terms of registers. See Section 2.

We show that this way of computing an operation on bits or registers can be obtained for any operation. Moreover the number of assignments is reasonable. In the case of a linear operation involving k bits or registers, we obtain sequences of at most $2k-1$ assignments. We also present an extension to real values instead of binary ones, which has applications like in image processing or floating point operations in arithmetic coprocessors. See Section 3. In the case of a bijective operation on k bits or registers, we obtain sequences of at most $2k-1$ assignments too. Moreover, the inverse operation is computed just by reversing the order of assignments. See Section 4. For a general operation, we obtain sequences of at most $4k-3$ assignments. See Section 5.

2 Bit operations on registers

For generalizations, we will assume that registers can have different sizes. For convenience and in order to state the definitions and results independently of the sizes of registers, we code the value of a sequence of k registers $R_1, R_2, \ldots, R_k$, of respective bit sizes $s(1), ..., s(k)$, by the complete sequence of their bits:

$$(R_1^1, \ldots, R_1^{s(1)}, R_2^1, \ldots, R_2^{s(2)}, \ldots\ldots, R_k^1, \ldots, R_k^{s(k)}).$$

Hence, any operation Φ on these k registers is represented by a mapping $E : \{0,1\}^n \to \{0,1\}^n$ where $n = s(1) + s(2) + \ldots + s(k)$ is the total number of bits. Let us call E the *bit-expansion* of the operation Φ. We naturally call *bit assignment* a mapping $\{0,1\}^n \to \{0,1\}$. We are going to design an *(in situ) integrable circuit (or program)* of E, that performs the transformation $X \mapsto E(X)$ for every $X \in \{0,1\}^n$, by a sequence of bit assignments, in such a way that we operate on successive bits in one way, then in the other way, and so on. This technique enables to obtain an optimal integration in register assignments, as illustrated in Figure 2.

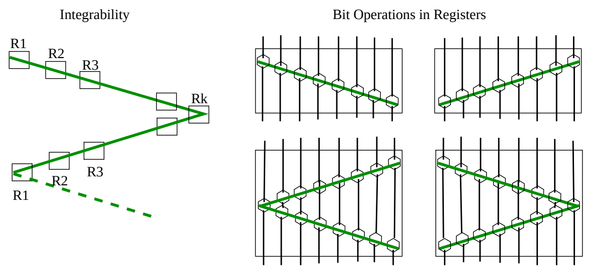

Figure 2. Bit and Register Assignments

For example, the integration of a sequence of bit assignments on 7 bit variables a, b, c, d, e, f, g into assignments on 3 registers $R_1 = (a,b)$, $R_2 = (c,d)$, $R_3 = (e,f,g)$, is obtained in the following way:

$$\underbrace{\alpha_a, \alpha_b}_{\tilde{\alpha}_1}, \underbrace{\alpha_c, \alpha_d}_{\tilde{\alpha}_2}, \underbrace{\alpha_e, \alpha_f, \alpha_g, \alpha'_f, \alpha'_e}_{\tilde{\alpha}_3}, \underbrace{\alpha'_d, \alpha'_c}_{\tilde{\alpha}'_2} \underbrace{\alpha'_b, \alpha'_a, \alpha''_b}_{\tilde{\alpha}'_1}, \ldots$$

where the index on an assignment corresponds to the modified bit or register.

Hence, the main results in the sequel can be equally designed in terms of bit or register operations (with any sizes of registers, a bit being a register of size one). Given a sequence S of bit assignments, we denote $R(S)$ the sequence of register assignments obtained by this integration process.

A sequence of register assignments S is called *reversible* when the assignments are of type $R_i := A(R_i) + F$ and the following transformation of the assignments makes sense:

$$[R_i := A(R_i) + F] \mapsto [R_i := A^{-1}(R_i - F)].$$

Then, the *reversed* sequence S^{-1} performs these assignments in the other way (from the last one to the first one), together with this transformation. In the particular case of bit assignments, the sequence is reversible when the assignments are of type $a := a + f(b, c, d, \ldots)$. Then, the transformation does not change the assignments ($A = A^{-1}$ and $F = -F$). For instance, on four bits (a, b, c, d):

S	S^{-1}
$a := a + bc + 1$	$d := d + abc$
$b := b + a + d$	$c := c + 1$
$c := c + 1$	$b := b + a + d$
$d := d + abc$	$a := a + bc + 1$

3 Linear operations

Back to the example of the introduction, the exchange of registers is a very particular *bit-linear operation*: the final value of each bit by this operation is defined by a linear function on $\mathbb{Z}/2\mathbb{Z}$. In other words, the bit-expansion of a bit-linear operation is a linear mapping on $\{0,1\}^n$. Such bit-linear operations are useful, for instance in image processing.

A *linear bit assignment* is naturally a linear mapping from $\{0,1\}^n$ to $\{0,1\}$. An assignment of a bit (resp. register) X is said *elementary* if it can be omitted like $X := X$ or consists in an addition like $X := X + Y$ where Y is another bit (resp. register).

Theorem 1. *Every bit-linear operation Φ on k registers is designed by a sequence $T = R(S)$ of at most $2k-1$ register assignments, where all the assignments of S are linear (and even elementary for the half of them). Moreover, when Φ is invertible, the sequence S is reversible and the operation Φ^{-1} is designed by the sequence $R(S^{-1})$.*

That is to say, such a bit-linear operation Φ is designed in at most $2k-1$ assignments of the registers where each assignment of a register R_i consists in linear assignments of its bits, like $R_1^3 := R_1^4 + R_3^5 + R_3^8 + R_5^2$.

Notice that bit-linear operations and linear operations on registers are in general distinct notions. For example, the operation on a two bits register $R_1 : (R_1^1, R_1^2) \mapsto (R_1^1, R_1^1 + R_1^2)$ is bit-linear. It maps $(0,0) \mapsto (0,0)$, $(0,1) \mapsto (0,1)$, $(1,0) \mapsto (1,1)$, $(1,1) \mapsto (1,0)$. That corresponds to the operation on the values of $R_1 : 0 \mapsto 0$, $1 \mapsto 1$, $2 \mapsto 3$, $3 \mapsto 2$, which is not linear in $\mathbb{Z}/4\mathbb{Z}$.

However, the previous result can be generalized to linear operations on registers:

Theorem 2. *For every field K, every linear operation Φ on k registers, taking values in K, is designed by a sequence T of at most $2k-1$ linear assignments. Moreover, the last $k-1$ assignments are elementary. If Φ is invertible, the sequence T is reversible and Φ^{-1} is designed by T^{-1}.*

For instance, that property holds for registers taking values in $\mathbb{R}$ (or in floats). This case has been recently applied with success for a company. We improved their programming codes of GPUs (Graphics Processing Units) by performing operations with an optimal number of registers.

We mention that the previous result can be adapted to rings $K = \mathbb{Z}/N\mathbb{Z}$. For example, the linear operation in $\mathbb{Z}/4\mathbb{Z}$ on three registers $(R_1, R_2, R_3) \mapsto (2.R_2 + 3.R_3, R_1 + R_2 + R_3, 3.R_1 + 2.R_2 + R_3)$ is designed by the five assignments:

$$\begin{aligned}
R_1 &:= R_1 + 2.R_3\\
R_2 &:= R_1 + R_2 + 3.R_3\\
R_3 &:= R_1 + 2.R_2 + R_3\\
R_2 &:= R_2\\
R_1 &:= R_1 + R_3
\end{aligned}$$

The fourth elementary assignment $R_2 := R_2$ can be omitted. Moreover, this sequence is reversible and one obtains a design for the inverse operation:

$$\begin{aligned}
R_1 &:= R_1 - R_3\\
R_2 &:= R_2\\
R_3 &:= R_3 - R_1 - 2.R_2\\
R_2 &:= R_2 - R_1 - 3.R_3\\
R_1 &:= R_1 - 2.R_3
\end{aligned}$$

If these registers take their values in $\mathbb{R}$, one obtains:

$$\begin{aligned}
R_1 &:= -3.R_1 + 2.R_3\\
R_2 &:= -R_1/3 + R_2 + 5.R_3/3\\
R_3 &:= -R_1/3 + 2.R_2 - R_3/3\\
R_2 &:= R_2\\
R_1 &:= R_1 + R_3
\end{aligned}$$

and the inverse operation is designed by:

$$\begin{aligned}
R_1 &:= R_1 - R_3\\
R_2 &:= R_2\\
R_3 &:= -3.R_3 + 6.R_2 - R_1\\
R_2 &:= R_2 + R_1/3 - 5.R_3/3 + R_1/3\\
R_1 &:= -R_1/3 + 2.R_3/3
\end{aligned}$$

Observe also that the case of affine mappings (adding some constants) can be reduced to the linear case by introducing a *virtual* register R_0 (just for the construction). For example, the design of the affine operation on three registers on $\mathbb{R}$: $(R_1, R_2, R_3) \mapsto (5 + 2.R_2 + 3.R_3, 2 + R_1 + R_2 + R_3, 9 + 3.R_1 + 2.R_2 + R_3)$ is obtained from the design of the linear operation on four registers $(R_0, R_1, R_2, R_3) \mapsto (R_0, 5.R_0 + 2.R_2 + 3.R_3, 2.R_0 + R_1 + R_2 + R_3, 9.R_0 + 3.R_1 + 2.R_2 + R_3)$ by fixing the value of register R_0 to 1, that is:

$$\begin{aligned}
&\cancel{R_0 := R_0}\\
R_1 &:= 2.R_3 - 3.R_1 - 4.\cancel{R_0}\\
R_2 &:= (5/3).R_3 - (1/3).R_1 + R_2 + (2/3).\cancel{R_0}\\
R_3 &:= 2.R_2 - (1/3).R_3 - (1/3).R_1 + (11/3).\cancel{R_0}\\
R_2 &:= R_2\\
R_1 &:= R_1 + R_3\\
&\cancel{R_0 := R_0}
\end{aligned}$$

4 Bijective operations

The example of register exchange is also a particular *bit-bijective operation* in the sense that its bit-expansion is a bijection on $\{0,1\}^n$. Unlike the case of linear operations, this notion is equivalent to bijections on registers. Observe that we are not only considering the particular $n!$ possible permutations of bits. We consider more general bijections: one-to-one mappings on $\{0,1\}^n$. Hence there are $2^n!$ such bijective operations, and the permutations are very particular cases.

Theorem 3. *Every bijective operation Φ on k registers is designed by a sequence $R(S)$ of at most $2k-1$ assignments. The sequence S is reversible and the sequence $R(S^{-1})$ computes the inverse operation Φ^{-1}.*

The proof is done by induction on the number n of bits and uses some *graph coloring* for the inductive process.

For a short example, consider a bijective operation Φ on two registers of size 2, $R_1 = (a,b)$ and $R_2 = (c,d)$, having bit-expansion $E(a,b,c,d) = (A,B,C,D)$ where :

$$\begin{aligned}
A &= 1 + acd + abc + ab + bc + cd + ad + ac + b + c\\
B &= 1 + abd + cd + bd + ad + ac + b\\
C &= bcd + abd + abc + bc + cd + b + d\\
D &= bcd + ab + bc + cd + bd + a + b + d
\end{aligned}$$

This operation Φ is designed by the 7 bit assignments:

$$\begin{aligned}
a &:= a + c + cd \\
b &:= b + cd + acd + ad \\
c &:= c + bd + ad \\
d &:= d + c + b + ab + a \\
c &:= c + a + d + abd \\
b &:= b + c + d + ad + ac + 1 \\
a &:= a + cd + bd + 1
\end{aligned}$$

Observe that this computation of E is simpler than the definition of E: 22 additions (XOR gates) versus 28, and 14 products (AND gates) versus 29. This observation brings to mind *Strassen-like algorithms* for fast matrices multiplications. The operation Φ^{-1} is designed by the reversed sequence:

$$\begin{aligned}
a &:= a + cd + bd + 1 \\
b &:= b + c + d + ad + ac + 1 \\
c &:= c + a + d + abd \\
d &:= d + c + b + ab + a \\
c &:= c + bd + ad \\
b &:= b + cd + acd + ad \\
a &:= a + c + cd
\end{aligned}$$

Both sequences are integrable in 3 register assignments.

5 General operations

For a general operation Φ, we introduce a decomposition of its bit-expansion $E = F \circ I \circ G$ where G (for *Grouping*) and F (for *Finalize*) are two bijective mappings, and I (for *Identify*) is a particular mapping that can be designed in at most one operation on each register. First, we obtain by this way at most $5k - 4$ register assignments. Second, a technical refinement of this preliminary construction, based upon *arithmetical arguments* for defining G, allows $F \circ I$ to be computed in $2k-1$ assignments. We obtain the following general result:

Theorem 4. *Every operation on k registers is designed by a sequence of at most $4k - 3$ assignments.*

As an example, consider the operation Φ on two registers $R_1 = (a, b)$ and $R_2 = (c)$, having bit-expansion $E(a, b, c) = (A, B, C)$ where :

$$\begin{aligned}
A &= 1 + c + ab + ac \\
B &= c + ac + bc \\
C &= 1 + a + abc + ac + bc
\end{aligned}$$

We obtain a sequence of assignments represented in the following tables (assignments α'_a and α''_a can be condensed into one assignment on a):

$$\overbrace{\alpha_a, \alpha_b, \alpha_c, \quad \alpha'_b, \alpha'_a,}^{G} \ \overbrace{\alpha''_a, \alpha''_b, \alpha''_c, \quad \alpha'''_b, \alpha'''_a}^{F\circ I}$$

$$\rightarrow \qquad \rightarrow \qquad \rightarrow \qquad \rightarrow$$

c	b	a	α_c	α_b	α_a	α_c	α'_b	α'_a	α''_c	α''_b	α''_a	α''_c	α'''_b	α'''_a
0	0	0	0	0	0	0	0	1	1	0	1	1	0	1
0	0	1	0	0	1	0	0	0	0	0	0	0	0	1
0	1	0	0	1	0	0	1	0	1	0	1	1	0	1
0	1	1	1	1	1	1	0	0	0	1	1	0	0	0
1	0	0	1	0	0	1	1	0	1	1	0	1	1	0
1	0	1	0	1	1	0	1	1	1	0	1	1	0	1
1	1	0	1	1	0	1	0	1	0	1	1	0	0	0
1	1	1	1	0	1	1	1	1	1	1	0	1	1	0

The above result gives an upper bound for the number of required assignments. Also, for instance, we are able to obtain an in situ computation made of 4 bit assignments for the previous mapping:

$$\begin{aligned}
c &:= a + b + c \\
a &:= 1 + b + c + ac \\
c &:= c + a + ab \\
b &:= c + ac
\end{aligned}$$

Observe again that this computation of E is simpler than the definition of E: 8 additions (XOR gates) versus 9, and 3 products (AND gates) versus 8.

6 Conclusion

The results presented in this note provide an optimization of register operations in a processor, a chip design or a program, in the sense that a suitable sequence of assignments on registers will perform the operation without using any external memory access. Further optimization concerns the complexity of these assignments, which is closely related to the technical complexity for the design of a chip in terms of logical gates and elementary circuits. Many experimentations (like the ones presented here), as well as technological and mathematical considerations, are strongly encouraging. This is the subject of further work...

References

[1] S. Burckel and E. Gioan. Procédé d'optimisation des ressources mémoires, implémentation de calculs pour processeurs. Reunion Island University - Patent: BREVET INPI FR0705152 (17 juillet 2007).

[2] S. Burckel and E. Gioan. In situ computation of mappings. In preparation.

[3] S. Burckel and M. Morillon. Sequential computation of linear boolean mappings. *Theoretical Computer Science serie A*, 314:287–292, 2004.