Exam for "Systèmes Digitaux" course

Tuesday January 12, 2021

Abstract

This exam is made up of two problems. It is better to answer to some of them in depth than all of them superficially.

The exam duration is 4 hours. The maximum number of pages is 6. You cannot use class material.

1 Barrett modular multiplication

1.1 Modular mathematics

Let $\mathbb{Z}$ be (relative) integers, and $\mathbb{N}$ be the positive integers ($\mathbb{N} = \{0, 1, 2, \ldots\}$).

Let a particular $M \in \mathbb{N}$, $M > 2$. For a given $z \in \mathbb{Z}$, we have the following integer division result:

$$z = qM + r \tag{1}$$

where $q = \lfloor z/M \rfloor$ and $r = z \bmod M$.

Let $\mathbb{R}$ be the real numbers.

For a given $a \in \mathbb{R}$, one writes the floor function $a \in \mathbb{R} \mapsto \lfloor a \rfloor \in \mathbb{Z}$ as:

$$a = \lfloor a \rfloor + \{a\} \tag{2}$$

where $\{a\} \in [0, 1[$.

We define the upper rounding of a real number a as:

$$\lceil a \rceil = \begin{cases} a & \text{if } a \in \mathbb{Z}, \\ 1 + \lfloor a \rfloor & \text{if } a \notin \mathbb{Z}. \end{cases}$$

Lemma 1. *For all $a \in \mathbb{R}$,*

$$\lceil -a \rceil = -\lfloor a \rfloor.$$

Q1.1: Prove this lemma 1

Property 1 (Representation). *Let $i \in \mathbb{N}$. The integer i fits on (exactly) k bits if*

$$2^{k-1} \leq i \leq 2^k - 1. \tag{3}$$

Said differently,

$$k = \lfloor \log_2(i) \rfloor + 1.$$

Notice that the number of bits to write i is not equal to $\lceil \log_2(i) \rceil$, because when i is a power of 2 (e.g., $i = 2^k$), $\lceil \log_2(i) \rceil = k$ but $k+1$ bits are required to write $(100\ldots0)_2$.

1.2 Barrett modular multiplication

1.2.1 Principle

In all this section, we denote as $M \in \mathbb{N}$, $M > 2$, a modulus.

Let $z \in \mathbb{N}$. The residue of z modulo M is:

$$r = z \bmod M = z - \lfloor z/M \rfloor,$$

according to (1).

Let $M \in \mathbb{N}$, $M > 2$, and k the number of bits to write it (namely, $k = \lfloor \log_2(M) \rfloor + 1$). We denote the *integer reciprocal of M* the number μ, defined by:

$$\mu = \lfloor 2^{2k}/M \rfloor.$$

When z satisfies $0 \leq z < M^2$, one can approximate the quotient $q = \lfloor z/M \rfloor$ according to:

$$\begin{aligned}
q &= \lfloor z/M \rfloor \\
&= \left\lfloor \frac{(z/2^{k-1}) \cdot (2^{2k}/M)}{2^{k+1}} \right\rfloor \\
&\geq \left\lfloor \frac{\lfloor z/2^{k-1} \rfloor \cdot \lfloor 2^{2k}/M \rfloor}{2^{k+1}} \right\rfloor \qquad \text{(as per Eqn. (2), } a = \lfloor a \rfloor + \{a\} \geq \lfloor a \rfloor \text{ because } \{a\} \geq 0) \\
&= \left\lfloor \left\lfloor \frac{z}{2^{k-1}} \right\rfloor \frac{\mu}{2^{k+1}} \right\rfloor = \tilde{q}. \qquad (4)
\end{aligned}$$

It appears that $\tilde{q}$ is not a bad approximation of q, and that it allows to avoid computing the division z/M. We detail in Sec. 1.2.2 why the approximation is good, and show that the complexity of computing $\tilde{q}$ is limited in Sec. 1.2.3.

1.2.2 Barrett's result

Barrett's result is that

Proposition 1. *Let q and $\tilde{q}$ defined as per Eqn. (4). One has:*

$$q - \tilde{q} \in \{0, 1, 2\}.$$

Q1.2: Prove this proposition 1

1.2.3 Efficient computation of Barrett's multiplication

Presentation The Barrett multiplication is depicted in Alg. 1.

Algorithm 1: Barrett's multiplication

static parameters:

- A modulus $M > 2$ of length $k = \lfloor \log_2(M) \rfloor + 1$,
- The integer reciprocal $\mu = \lfloor 2^{2k}/M \rfloor$.

input : Multiplier and multiplicand x, y, such that $0 \leq x, y < M$
output : $(x \cdot y) \bmod M$

1 $z \leftarrow x \cdot y$;
2 $\tilde{q} \leftarrow \lfloor \lfloor z/2^{k-1} \rfloor \mu / 2^{k+1} \rfloor$;
3 $r \leftarrow z - \tilde{q} \cdot M$;
4 **while** $r \geq M$ **do** `// Executed 0, 1 or 2 times (see Prop. 1)`
5     $r \leftarrow r - M$; `// Extra-reduction`
6 **return** r ;

Variable data representation For implementation purposes, it is required to know how to representation the values manipulated in Alg. 1, in particular their bitwidth. The list below gathers these values:

- M fits on k bits (exactly),
- μ fits on $k+1$ bits (exactly),
- $\tilde{q}$ fits on k bits (or less),
- r, at line 3 of Alg. 1, fits on $2k$ bits (or less), since z fits on $2k$ bits and $\tilde{q} \cdot M$ also. But actually, we know more: r is positive and it fits on $k+2$ bits.

Q1.3: Prove these four results

Operation We intend to implement the Barrett modular multiplication in combinational logic. The operations in Barrett's multiplication algorithm are turned into hardware functions.

Q1.4: Explain how to implement Line 1 of Alg. 1

Q1.5: Same question with Line 2 of Alg. 1

Q1.6: Same question with Lines 4 and 5 of Alg. 1

In this question, a constant-time solution (combinational) is sought.

2 Constant-time computations

We consider a processor with n-bit registers. The data representation is in "two's complementary", meaning that: The processor is equipped with Boolean and arithmetic operations on data a, b:

- 0 is represented as $(\texttt{00000000})_2$ (when $n = 8$), and
- 255 is represented as $(\texttt{11111111})_2$ (when $n = 8$).
- $a\texttt{+}b$, which returns $a + b \bmod 2^n$;
- $a\texttt{-}b$, which returns $a - b \bmod 2^n$;
- $a\texttt{\&}b$, which returns the bit-wise AND between a and b, seen as vector of n bits;
- $a \texttt{|} b$, which returns the bit-wise OR between a and b, seen as vector of n bits;
- $a\texttt{\^{}}b$, which returns the bit-wise XOR between a and b, seen as vector of n bits;
- $\texttt{\~{}}a$, which returns the one's complement of a, namely $(\neg a_{n-1}\neg a_{n-2} \ldots \neg a_0)_2$ where $a = (a_{n-1}a_{n-2} \ldots a_0)_2$ in binary form.

The processor is also endowed with classical ITE (If-Then-Else) conditional tests. The "Then" branch is taken if the tested value is nonzero, otherwise the "Else" branch is taken.

Condition $a\texttt{==}b$ (resp. $a\texttt{!=}b$) yields 0 or 1, depending whether a and b are equal or not (resp. are different or not).

Algorithm 2: Operation 1

```
input : a, b, s
output: c
if s==0 then
    c ← a;
else
    c ← b;
return c ;
```

Algorithm 3: Operation 2

```
input : a, b, s
output: c
m ← -(s!=0) ;
c ← (a&m)|(b&~m) ;
return c ;
```

Q2.1: Compare the two algorithms 2 and 3, in terms of functionality

Q2.2: Discuss which algorithm is the fastest.

Q2.3: Alg. 3 is said "constant-time", whereas Alg. 2 is not. Could you explain a property of the execution flow which justifies this naming?

Q2.4: Propose a "non constant-time" version of Alg. 4 which is faster, in average (assuming the value of s has the same probability to zero and to be nonzero)

Algorithm 4: Operation 3

input : a, b, s
output: c

```
m ← -(s != 0) ;
t ← m & (a ^ b) ;
c ← a ^ t ;
return c ;
```