7th Conference on Real Numbers and Computers (RNC 7)

LORIA, Nancy, France, July 10–12, 2006

Pipelined Architecture for Accurate Floating Point Range Reduction

Francisco J. Jaime, Julio Villalba, Javier Hormigo, Emilio L. Zapata

Dept. Computer Architecture

University of Malga

SPAIN

Talk Outline

- Introduction
  - Posing the problem
- Double residue modular range reduction
- Pipelined Architecture
- Radix-4 coding
- Summary and conclusions

Introduction

- Range reduction is required for elementary functions evaluation
- Reductions:
    - Additive
        - Subtracting a multiple of a constant
    - Multiplicative
        - Multiplying by the inverse of a power of a constant

- Our algorithm
    - Additive reduction
        - Trigonometric functions

Introduction

- The reduction can be done in software or in hardware
- Our proposal is oriented toward a hardware implementation for application-specific systems
  - Consider both fixed-point & floating point representation
  - Maximum accuracy for floating point for the specific representation

Introduction

- A poor range reduction may lead to catastrophic accuracy problems in the evaluation of trigonometric functions when

1. the input argument is close to a multiple of the constant (i.e. π)

2. the input argument is large [1]

Introduction

– Steps in the evaluation of trigonometric funcions:

1) Reducction of the input argument to the interval [0,2π)
2) Compute the function inside the interval

Ej.: sin(114 rad)

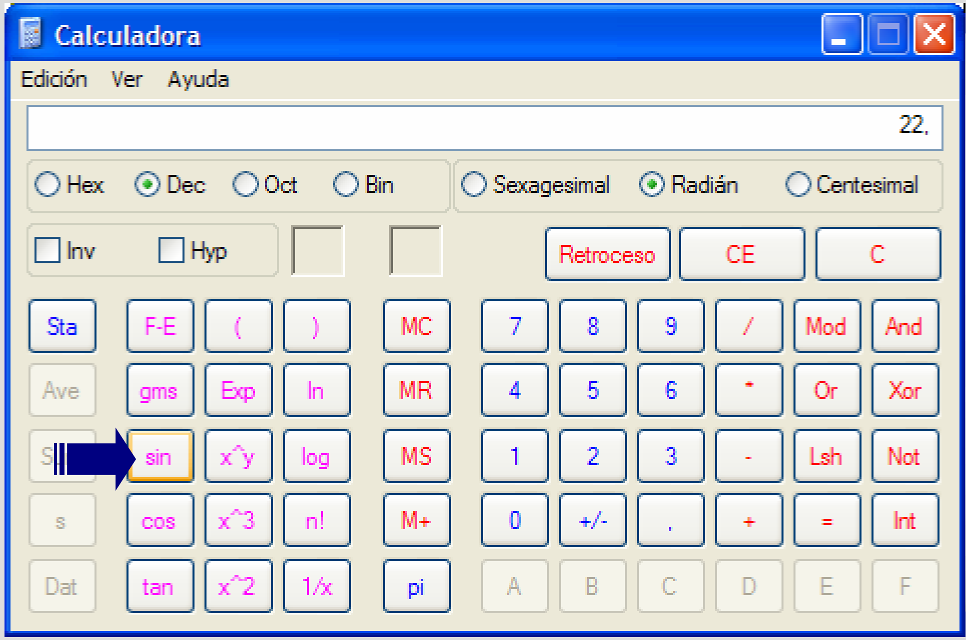
1) Reduction 114 rad -> 0.9 rad

2) Computation sin(0.9 rad)= 0,7850

A not accurate reduction <u>(spep 1) can lead to inaccurate results</u>

Introduction

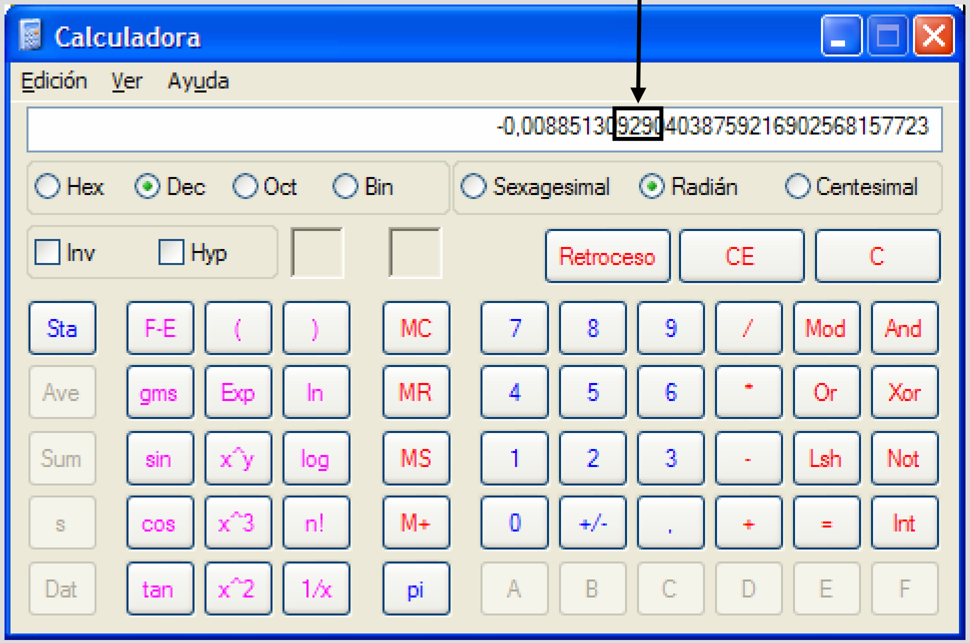
1) Example of input argument close a multiple of π
   - Computation of sin(22 rad)



Introduction

1) Example of an input argument close a multiple of π

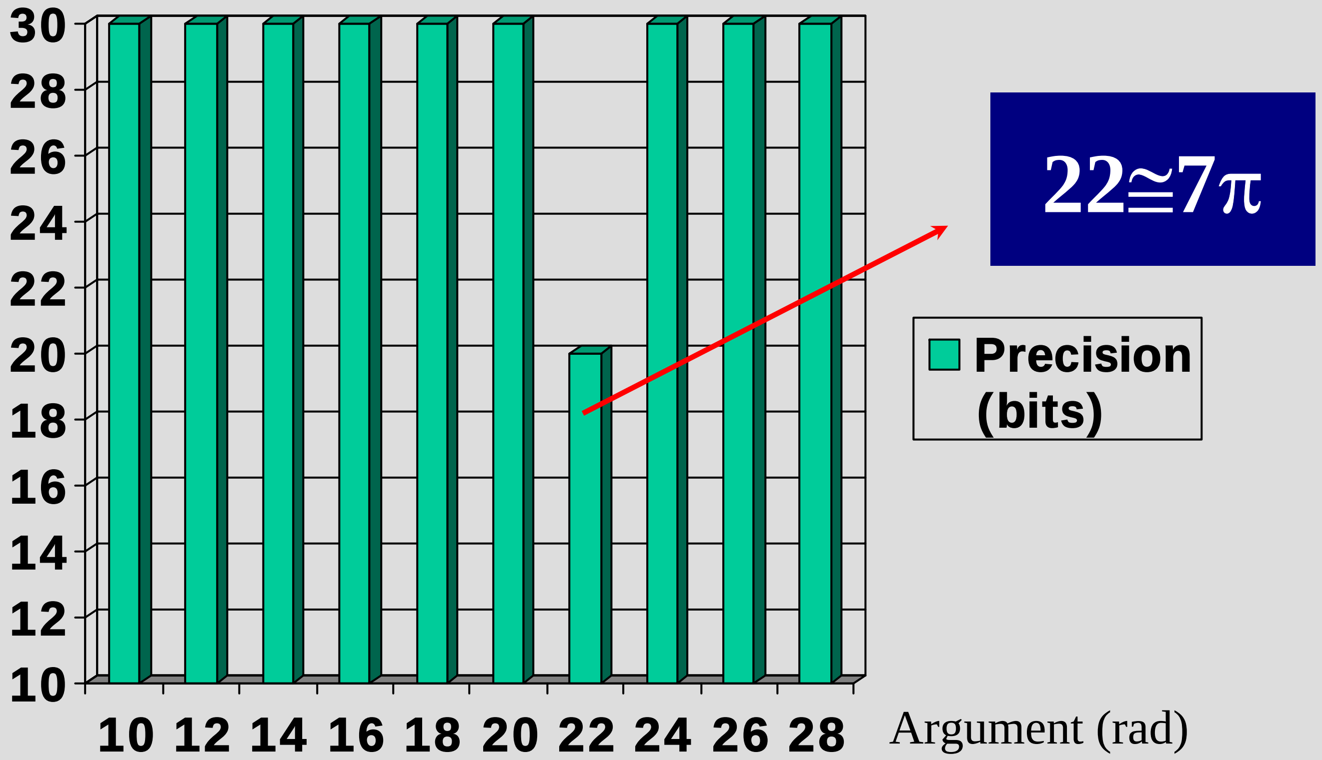
   – Computation of sin(22 rad)



Introduction

1) Arguments close to a multiple of π

– Computation of sin(22 rad)



$$R = 22-7\pi = 0.0000\ 0001\ 0001\ 1101\ 0011\ 0001\ 1001$$

Introduction

2) Large input argument

INSTRUCTION SET REFERENCE

intel®

FSIN—Sine

Opcode	Instruction	Description
D9 FE	FSIN	Replace ST(0) with its sine.

Description

This instruction calculates the sine of the source operand in register ST(0) and stores the result in ST(0). The source operand must be given in radians and must be within the range -2^{63} to $+2^{63}$. The following table shows the results obtained when taking the sine of various classes of numbers, assuming that underflow does not occur.

SRC (ST(0))	DEST (ST(0))
$-\infty$	*

Range of representable numbers for IEEE 754 (single precision): $[-2^{127},2^{127}]$

Introduction

- Solutions
  - Software
  - Hardware

Introduction

- Solutions
  - Software
    - Cody and Waite [3]
      - Software Manual for elementary functions (Prent. Hall 1980)
    - Payne and Hanek algorithm [5]
      - Radian reduction for trigonometric functions (SIGNUN let. 1983)
    - Brisebarre, Defour, Kornerup, Muller, Revol
      - A new range reduction algoritm (Trans. Comp. 2005)

Software solutions: good accuracy, slow

Introduction

- Solutions
  - Hardware
    - Double Residue Modular range reduction [6]
      - Based on a table with positive and negative residues
      - Solves both problems
        - » Input argument close to multiple of a constant
        - » Large argument
      - Architecture: fast, simple hardware and valid for fixed-point & floating-point when the module is a rational or an irrational number.
        - » IEEE 754 full range
      - Handicap for floating point:
        - » word-serial implementation only

[6] Julio Villalba, Tomas Lang, Mario A. González, Double residue modular range reduction for floating-point hardware implementation, IEEE Transactions On Computers, vol. 55, no. 3, march 2006

Introduction

- Our aim:
  - Design a pipelined architecture supporting the double residue modular range reduction
    - preventing the replication of the table of residues

Double residue modular range reduction

Double residue MRR

- Fundamental of doble residue MRR *(fixed-point)*
  - For a number X and a module M, obtain a number R:

$$R = X - k \cdot M \qquad 0 \le R < M$$

- Example: M=7, X=372

$$R = 372 - k \cdot 7 \qquad 0 \le R < 7$$

  - Select k=53, thus R=1
- We call “residue” to R and “mod” to the operation, such that:

$$R = X \bmod M \qquad (372 \bmod 7 = 1)$$

Double residue MRR

- Double residue MRR
  - Define elementary positive residues and elementary negative residues, which are stored in a table:

$$m_i^+ = 2^i \bmod M$$
$$m_i^- = 2^i \bmod M - M$$

Elementary Positive residue

Elementary negative residue

  - Example: M=7, we define

$$m_6^+ = 2^6 \bmod 7 = 64 \bmod 7 = 1$$
$$m_6^- = 2^6 \bmod 7 - 7 = -6$$

$$m_5^+ = 2^5 \bmod 7 = 32 \bmod 7 = 4$$
$$m_5^- = 2^5 \bmod 7 - 7 = -3$$

$$m_4^+ = 2^4 \bmod 7 = 16 \bmod 7 = 2$$
$$m_4^- = 2^4 \bmod 7 - 7 = -5$$

Double residue MRR

- Double residue MRR
  - Algorithm:
    - Define an accumulated residue R(i)

$X = x_{n-1}\dots x_1 x_0 . x_{-1} x_{-2}\dots x_{-f}$

$R(i+1) = R(i) + m_i^* x_i$ with $m_i^* = \begin{cases} m_i^+ & si \quad R(i) < 0 \\ m_i^- & si \quad R(i) \geq 0 \end{cases}$

    - For each power of 2, select the suitable positive or negative residue
      - The accumulated residue is **bounded by M**
        - » R(i) is always inside convergence bound

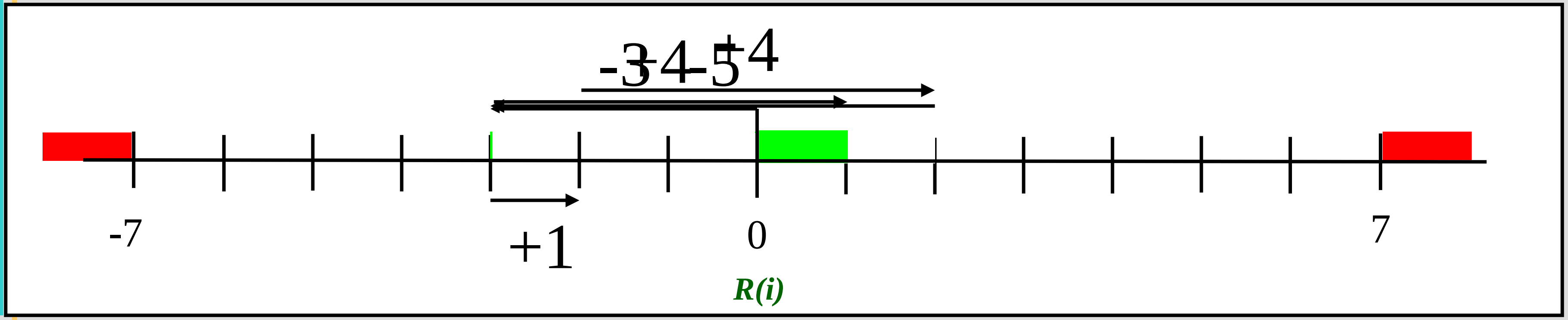
Double residue MRR

- Double residue MRR: an intuitive idea
  - Computation of 372 mod 7 = 1
  - $372 = 101110100 = 2^8+2^6+2^5+2^4+2^2 = 256 + 64 + 32 + 16 + 4$

$(256 \bmod 7 + 64 \bmod 7 + 32 \bmod 7 + 16 \bmod 7 + 4 \bmod 7) =$

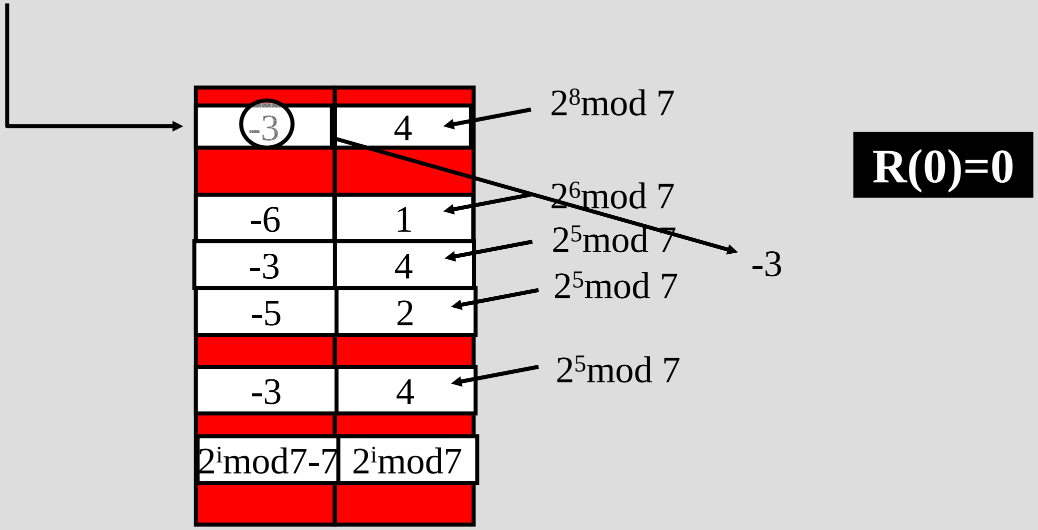
(4,-3) (1,-6) (4,-3) (2,-5) (4,-3)

$= (-3 + 1 + 4 + -5 + 4) = 1$



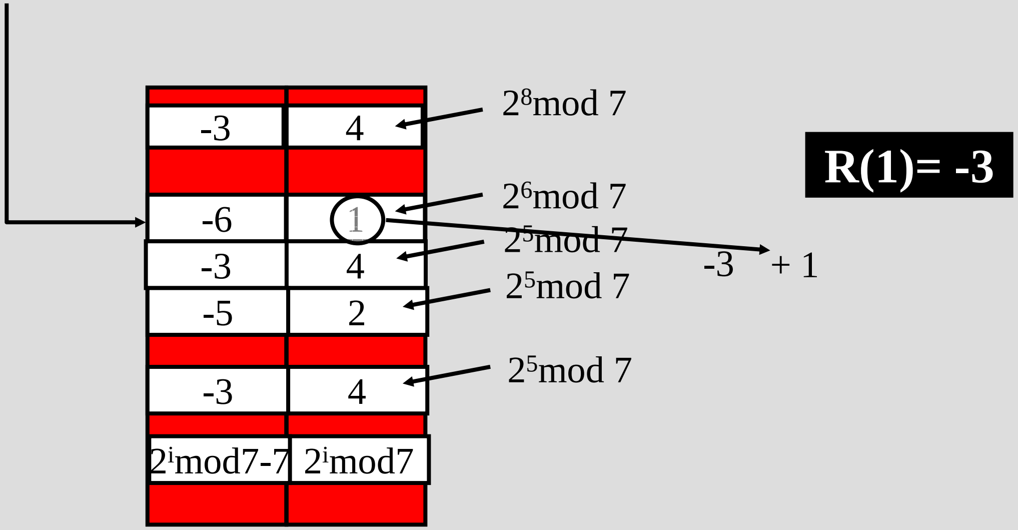
Double residue MRR

- Double residue MRR: an intuitive idea
  - Computation of 372 mod 7 = 1
    - 372 = 101110100 = $2^8+2^6+2^5+2^4+2^2$ =256 + 64 + 32 +16 + 4



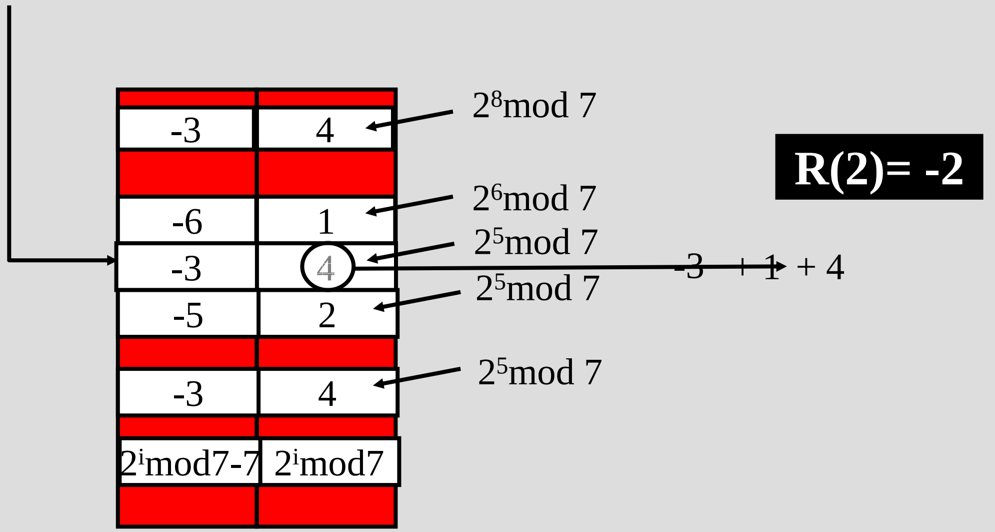
Double residue MRR

- Double residue MRR: an intuitive idea
  - Computation of 372 mod 7 = 1
    - $372 = 101110100 = 2^8+2^6+2^5+2^4+2^2 = 256 + 64 + 32 + 16 + 4$



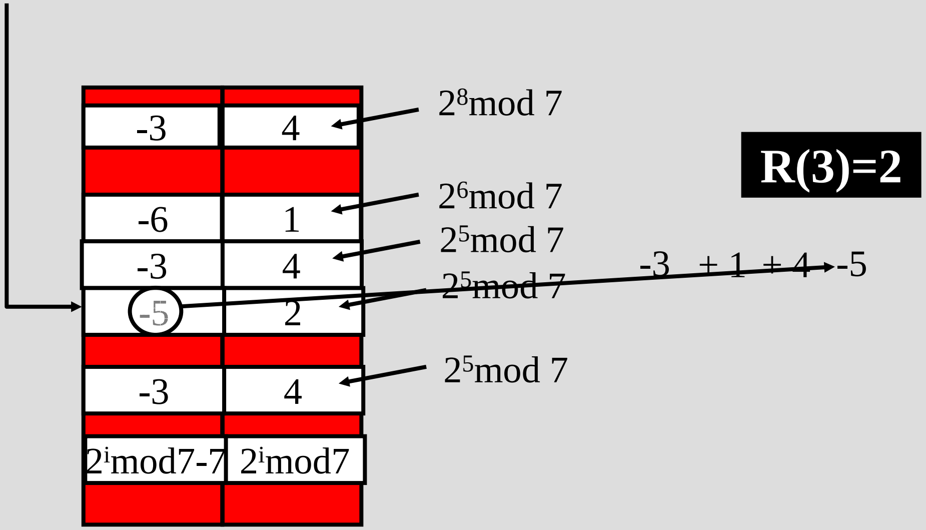
Double residue MRR

- Double residue MRR: an intuitive idea
  - Computation of 372 mod 7 = 1
    - $372 = 101110100 = 2^8+2^6+2^5+2^4+2^2 = 256 + 64 + 32 + 16 + 4$



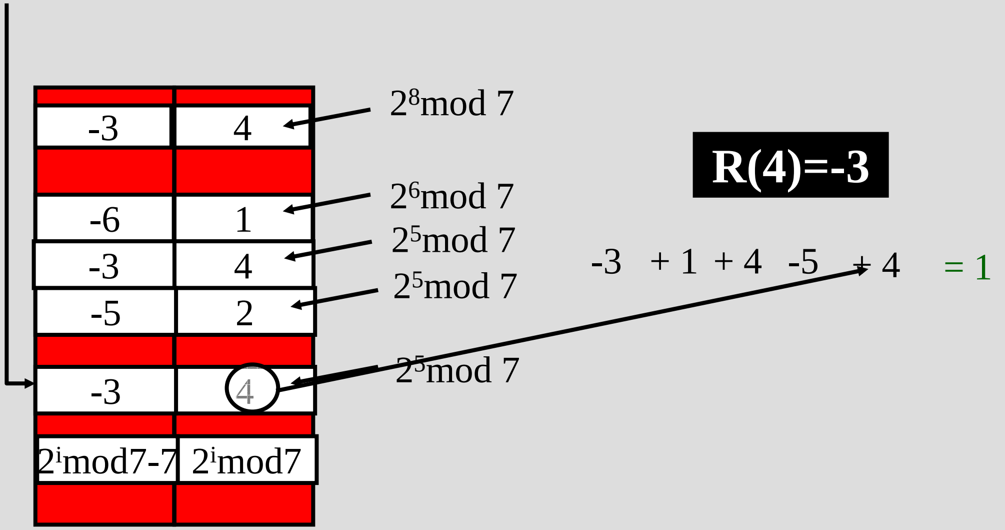
Double residue MRR

- Double residue MRR: an intuitive idea
  - Computation of 372 mod 7 = 1
    - $372 = 101110100 = 2^8+2^6+2^5+2^4+2^2 = 256 + 64 + 32 + 16 + 4$



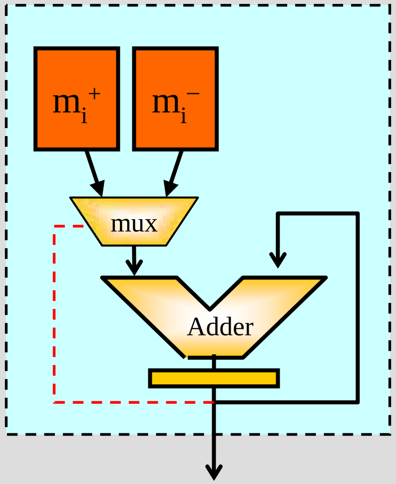
Double residue MRR

- Double residue MRR: an intuitive idea
  - Computation of 372 mod 7 = 1
    - $372 = 101110100 = 2^8+2^6+2^5+2^4+2^2 = 256 + 64 + 32 + 16 + 4$



Double residue MRR

- Word-serial Architecture:



Reduced argument

Double residue MRR

- Implementation in carry-save arithmetic
  - R(i)=S(i)+C(i)
  - ***Estimation*** of the sign of R(i) from the 3- MSBs of R(i) :
    - Define a **new selection function** of the elementary residues:

$$m_i^* = \begin{cases} m_i^+ & if \quad R(i) < 0 \\ m_i^- & if \quad R(i) \geq 0 \end{cases} \Rightarrow m_i^* = \begin{cases} m_i^+ & if \quad \overline{R}(i) < 0 \\ m_i^- & if \quad \overline{R}(i) \geq 0 \end{cases}$$

$\overline{R}(i)$ **Truncated value of R(i) using the 3 MSBs of S(i),C(i)**

Double residue MRR

- Implementation in carry-save arithmetic
  - The iterations are self-correcting
    - The deviation produced by a wrong estimation of the sign of R(i) is corrected by the next iteration
    - A final correction iteration is required to ensure the convergence

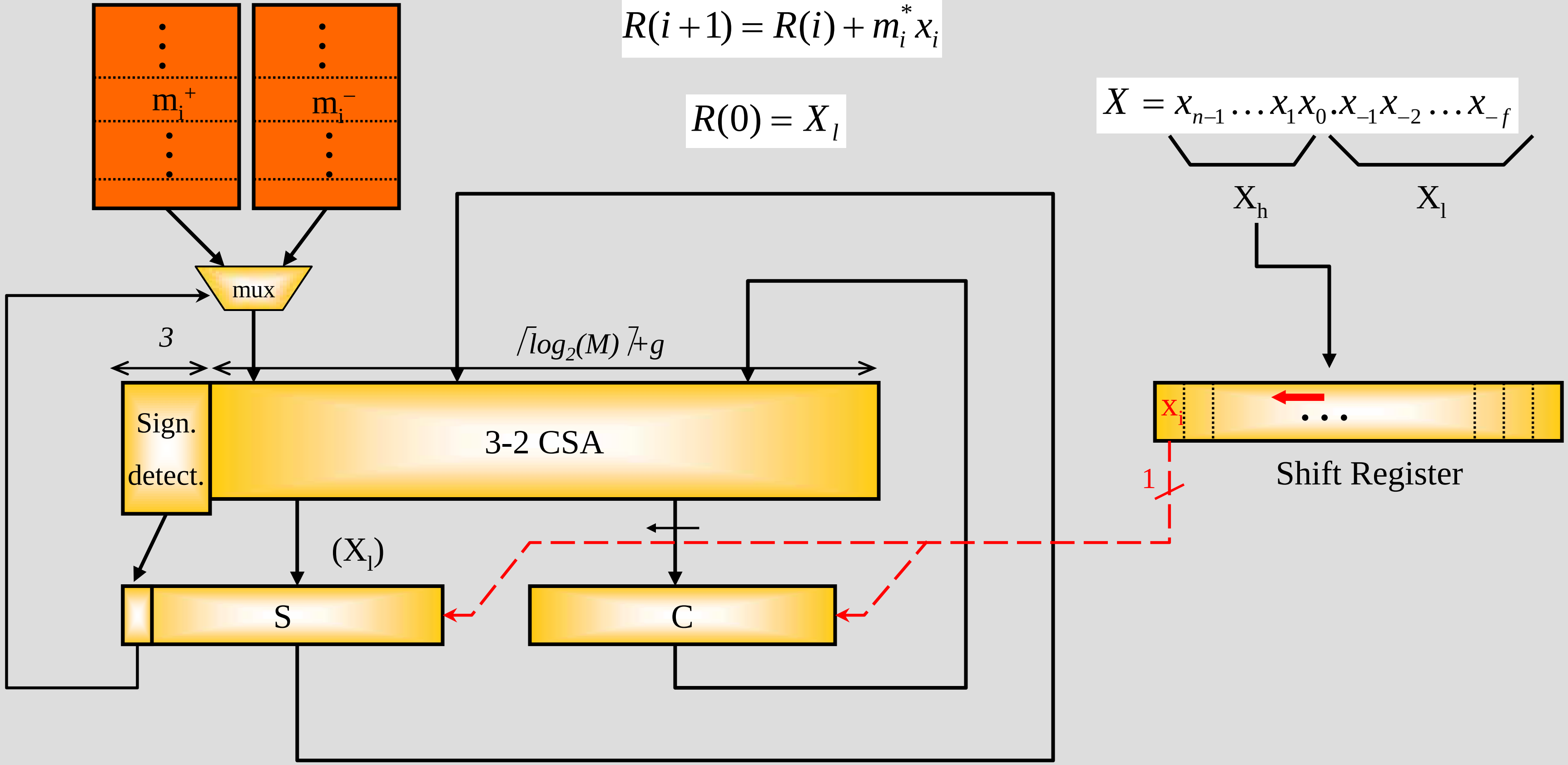
Double residue MRR

– Architecture in carry-save arithmetic (fixed-point)

$$R(i+1) = R(i) + m_i^* x_i$$

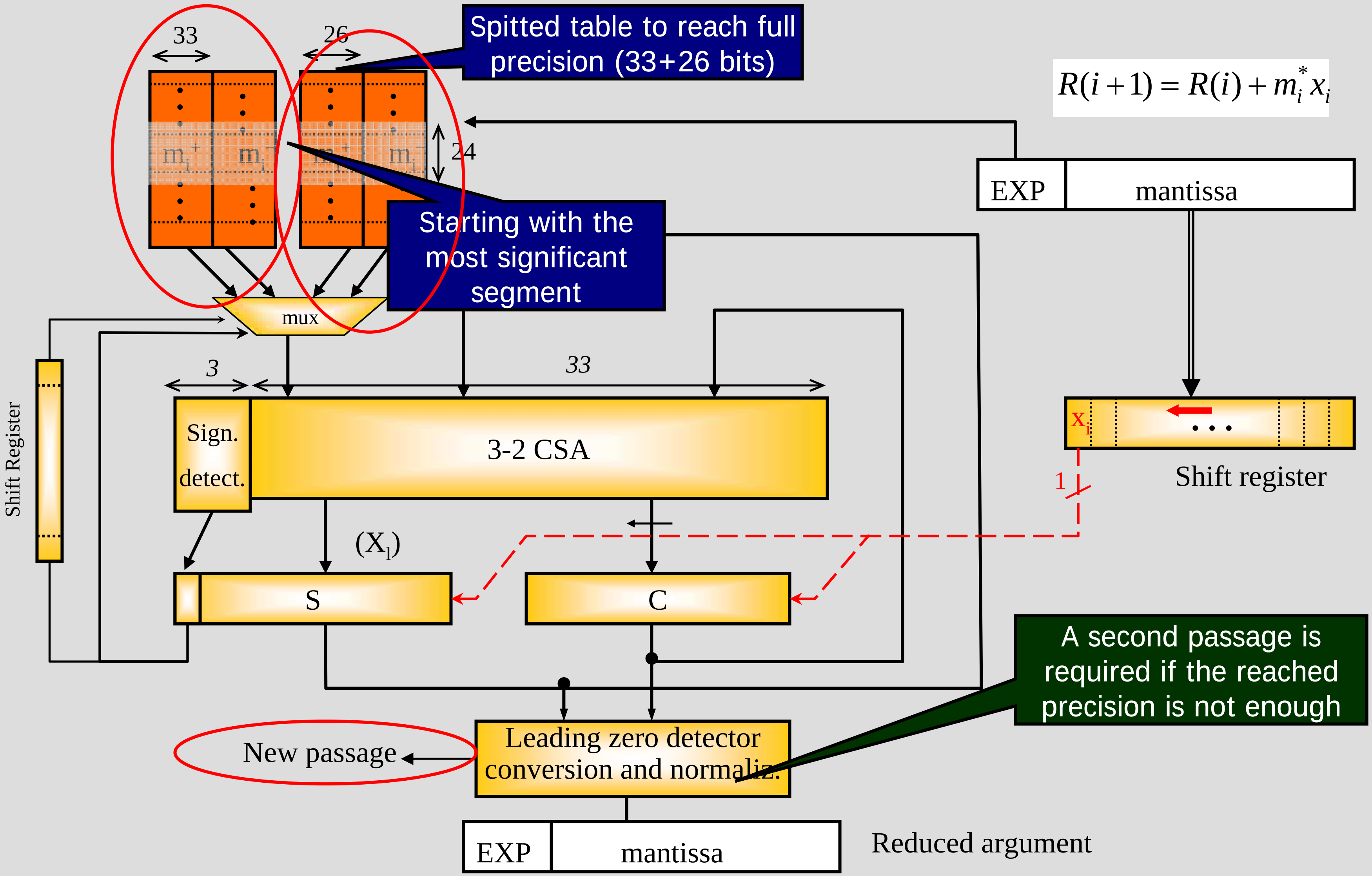
$$R(0) = X_l$$

$$X = x_{n-1} \dots x_1 x_0 . x_{-1} x_{-2} \dots x_{-f}$$



Double residue MRR

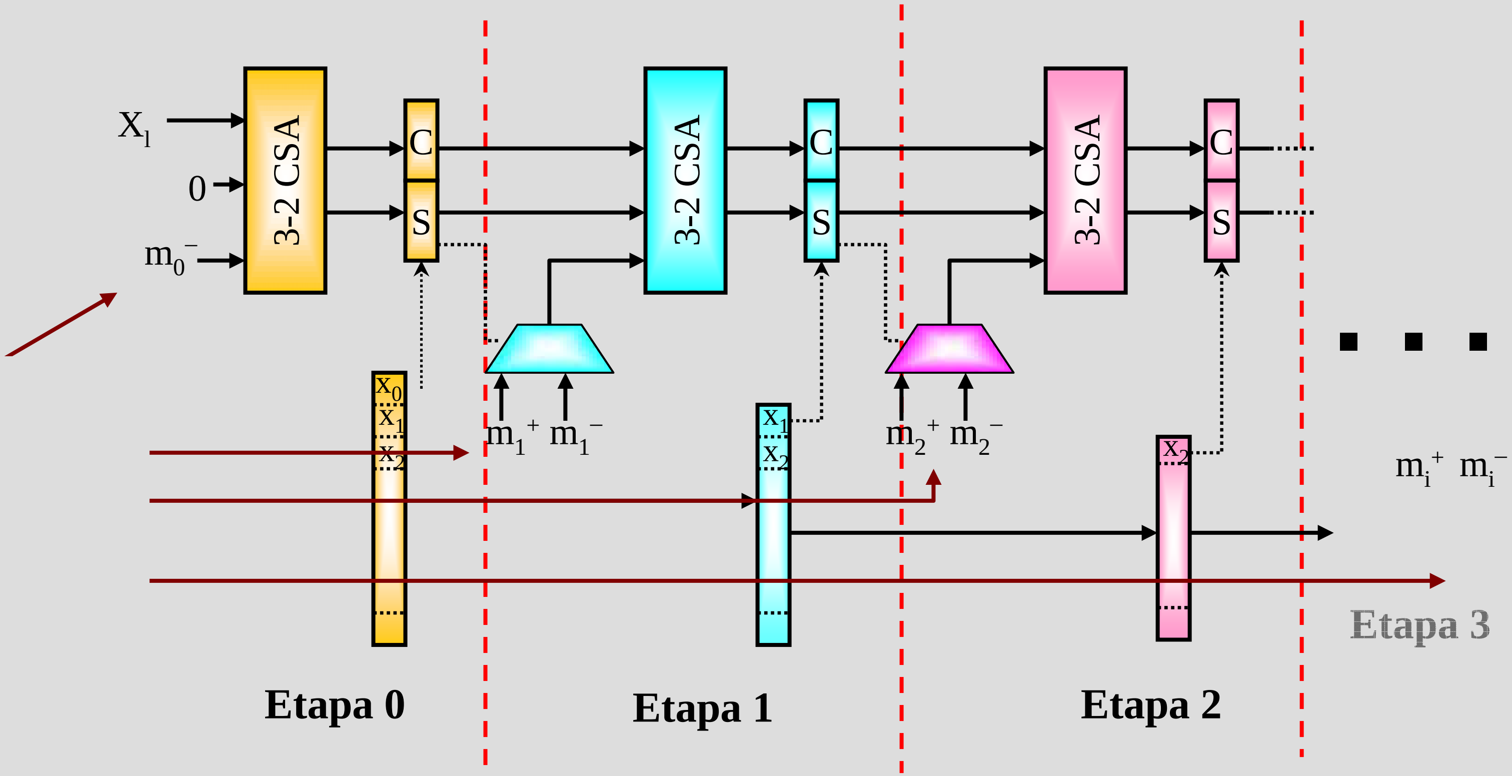
– Architecture in carry-save arithmetic (IEEE 754 floating-point, sing. prec. M=2π)*[6]*



Pipelined architecture

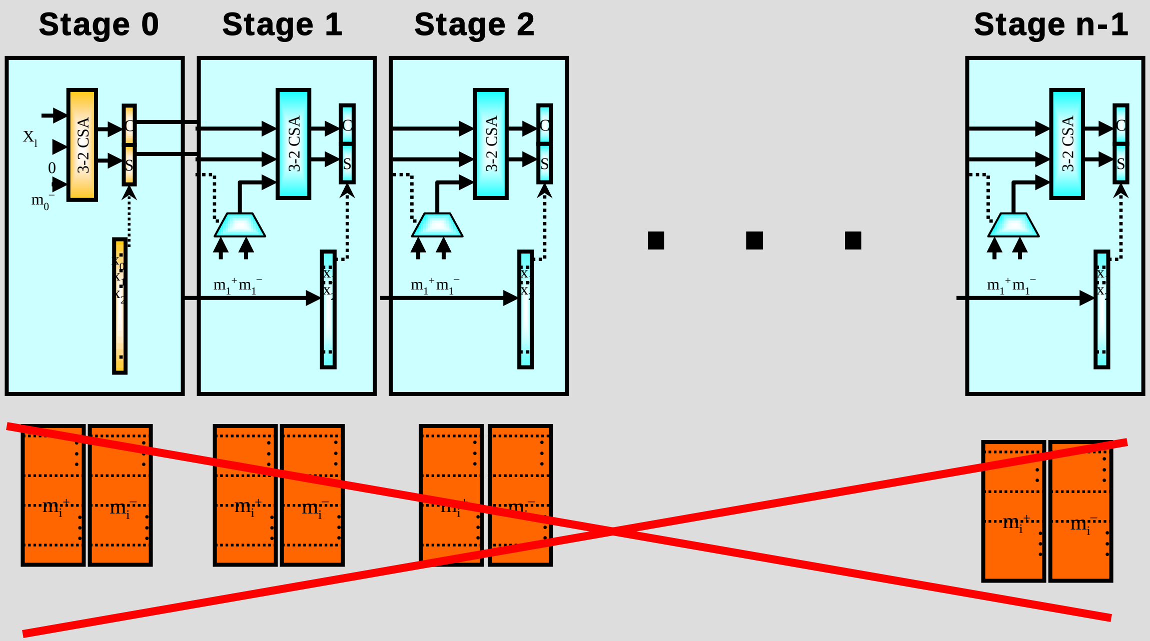
Pipelined architecture

- Fixed-point pipelined architecture for a fixed M
  - Distributed elementary residues m_i (hardwired)



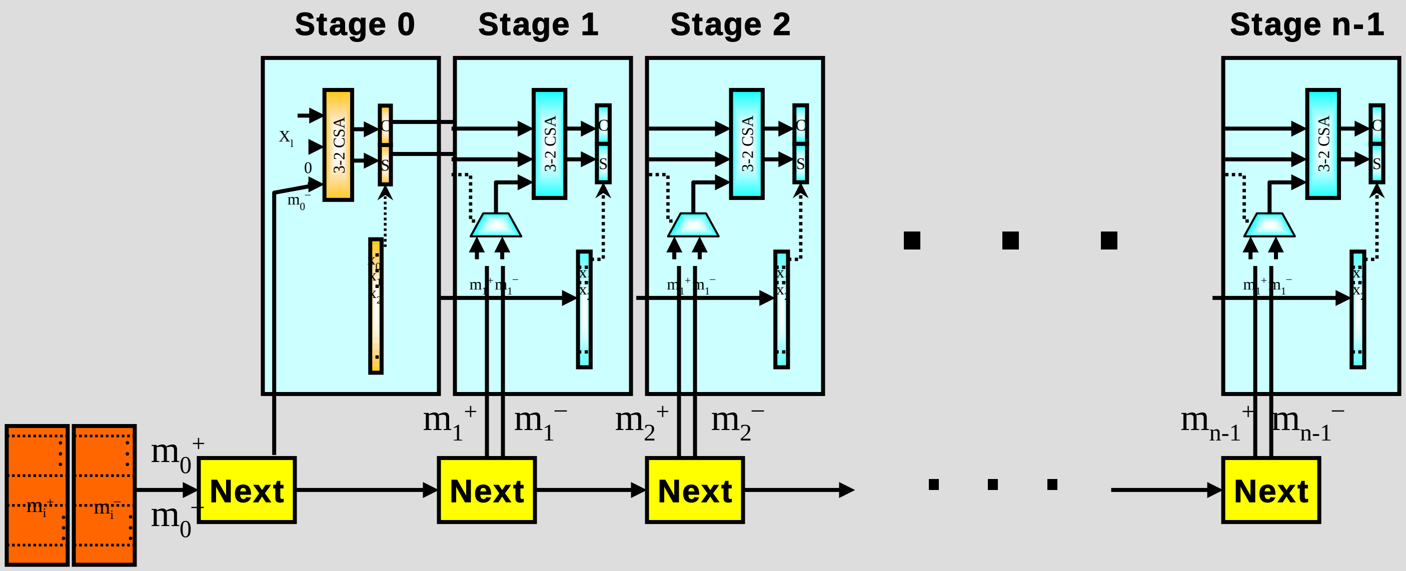
Pipelined architecture

- Floating point pipeliene implementation
  - Problem with the pipeliene implementation for a fixed M:
    - Requires replication of the table (as many tables as stages)



Pipelined architecture

- Our proposal:
  - Prevents the replication of the table by generation of the elementary residue for the next stage



Pipelined architecture

– Preventing the replication:

  • Define a *elementary residue generator as*:

$$m_{i+1}^{+} = \begin{cases} 2m_i^{+} & if \quad 2m_i^{+} < M \\ 2m_i^{+} - M & if \quad 2m_i^{+} \geq M \end{cases}$$

$$m_{i+1}^{-} = \begin{cases} 2m_i^{-} + M & if \quad 2m_i^{+} < M \\ 2m_i^{-} & if \quad 2m_i^{+} \geq M \end{cases}$$

– A single table of elementary residues is required

– The elementary residue for the stage ***i*** is obtained from the elementary residue of the previous stage ***i-1***

Pipelined architecture

– Reducing the hardware cost of the elementary residue generator

$$\text{if} \quad 2m_i^+ < M \quad \text{then} \quad \begin{cases} m_{i+1}^+ = 2m_i^+ \\ m_{i+1}^- = 2m_i^- + M \end{cases} \qquad \textbf{FLAG}_i\textbf{ = 0}$$

$$\text{else} \quad \begin{cases} m_{i+1}^+ = 2m_i^+ - M \\ m_{i+1}^- = 2m_i^- \end{cases} \qquad \textbf{FLAG}_i\textbf{ = 1}$$

– A single comparison is required

– The evolution of the generation of elementary residues can be precomputed and stored (Flags)

» ***Comparison is prevented***

» ***Only one addition is requierd in each stage***

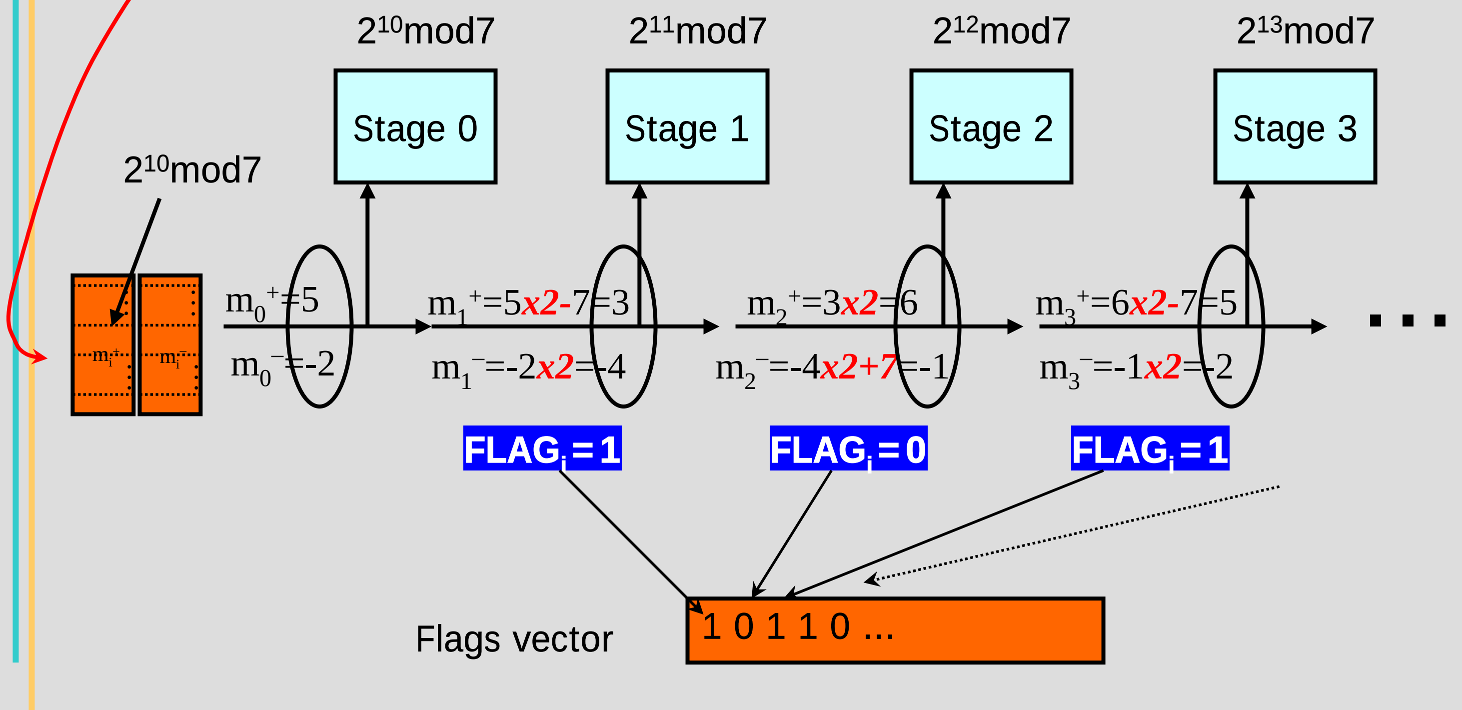
Pipelined architecture

– Example (M=7)

$$
\text{if} \quad 2m_i^+ < M \quad \text{then} \quad \begin{cases} m_{i+1}^+ = 2m_i^+ \\ m_{i+1}^- = 2m_i^- - M \end{cases} \quad \textbf{FLAG}_i\textbf{=0}
$$

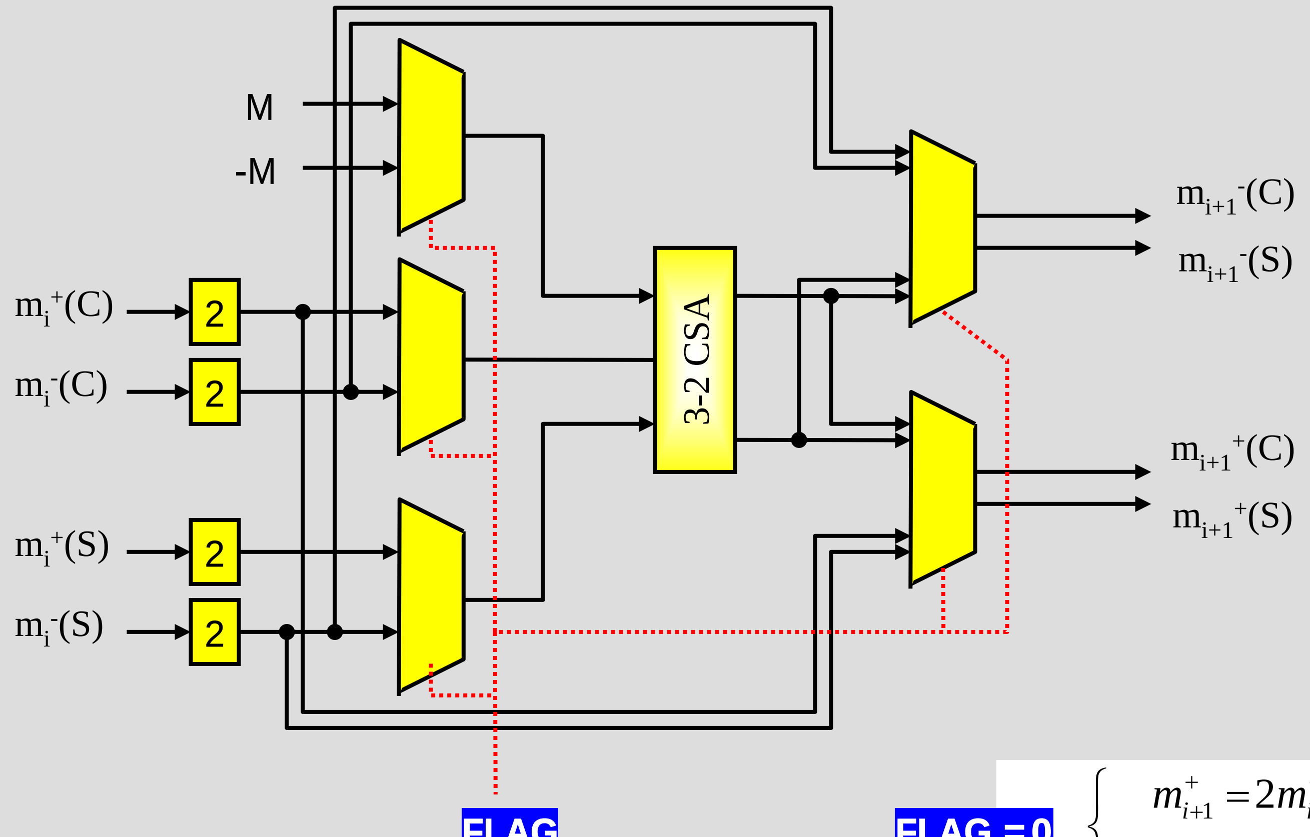
$$
\text{else} \quad \begin{cases} m_{i+1}^+ = 2m_i^+ - M \\ m_{i+1}^- = 2m_i^- \end{cases} \quad \textbf{FLAG}_i\textbf{=1}
$$

X=1.101001.. 2^{10}



Pipelined architecture

– Carry-save elementary residue generator (module **Next**)

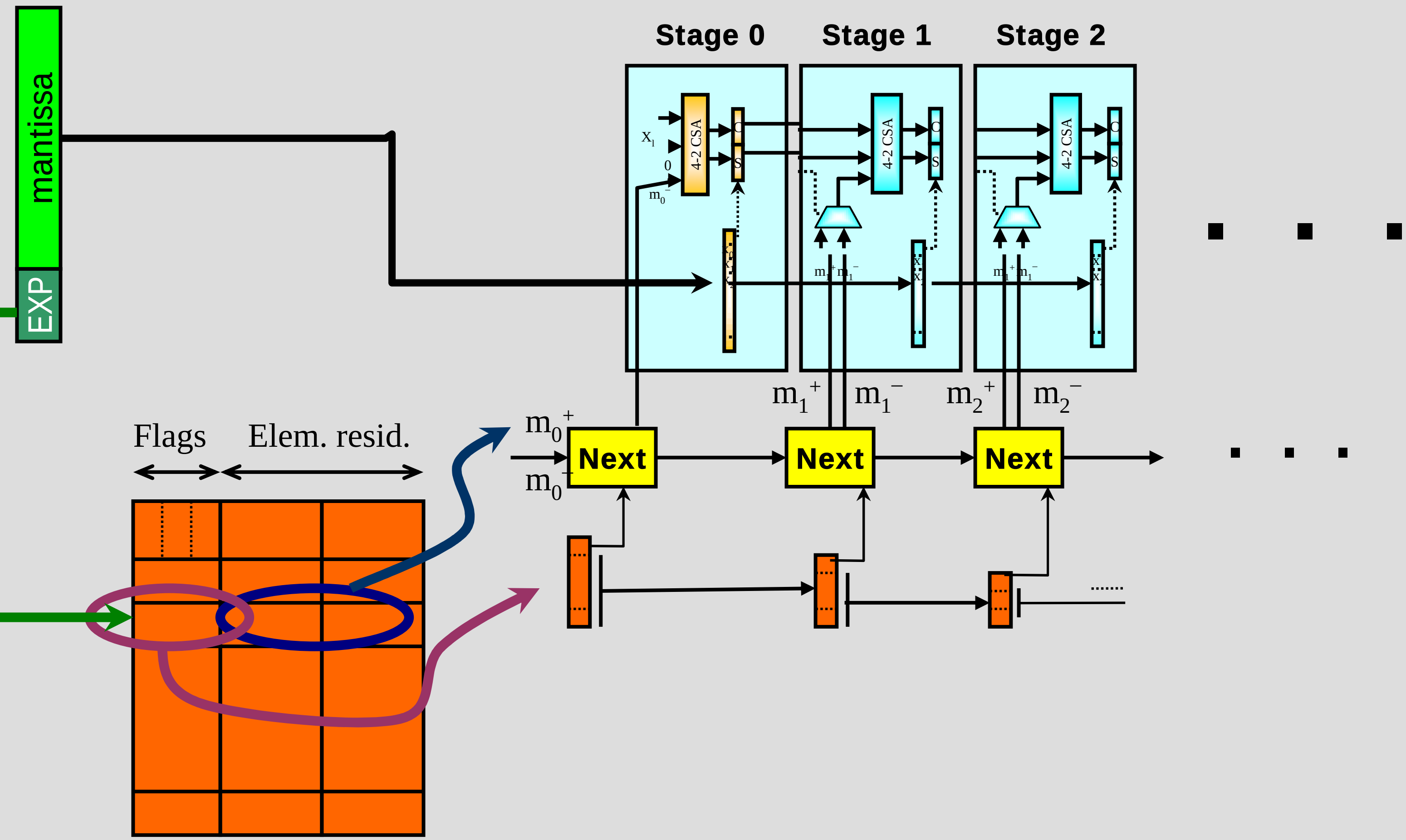


$FLAG_i = 0$
$$\begin{cases} m_{i+1}^+ = 2m_i^+ \\ m_{i+1}^- = 2m_i^- + M \end{cases}$$

$FLAG_i = 1$
$$\begin{cases} m_{i+1}^+ = 2m_i^+ - M \\ m_{i+1}^- = 2m_i^- \end{cases}$$

Pipelined architecture

- Final Architecture



Pipelined architecture

- Error propagation
  - Computing the accumulative addition *R(i)*

$$R(i+1) = R(i) + m_i^* x_i$$

  - Computing elementary residues m_i

$$m_{i+1}^* = 2m_i^* - M$$

Aim: precision 1 ULP using guard bits

Pipelined architecture

- Error propagation
  - Computing the accumulative addition *R(i)*

$$R(i+1) = R(i) + m_i^* x_i$$

- *n iterations:*
  - » *guard bits:* $g = \log_2 \lceil n \rceil$

Pipelined architecture

- Error propagation
  - Computing elementary residues m_i

$$m_{i+1}^* = 2m_i^* - M$$

ULP

g guard bits (accum.)

$$y_1 y_2 y_3 y_4 \ldots y_t \; y_{t+1} y_{t+2} \ldots y_{t+g} \, y_{t+g+1} y_{t+g+2} \ldots y_{t+j}$$

t bits for representation

j total guard bits

Initial error ε: $\varepsilon < 2^{-t-j}$

Pipelined architecture

- Error propagation

  – Computing elementary residues m_i

g guard bits (accum.)

$$y_1 y_2 y_3 y_4 \cdots y_t \, y_{t+1} y_{t+2} \cdots y_{t+g} \, y_{t+g+1} y_{t+g+2} \cdots y_{t+j}$$

t bits for representation

j total guard bits

Initial error ε: $\varepsilon < 2^{-t-j}$

Recurrence:

$$\begin{cases} a_0 = \varepsilon \\ a_k = (2a_{k-1} + 1)\varepsilon \end{cases} \Rightarrow a_k = (2^{k+1} - 1)\varepsilon$$

$$j > g + \log_2(2^{k+1} - 1)$$

Pipelined architecture

- Error propagation conlusion:

  – Computing the accumulative addition *R(i)*

  $$R(i+1) = R(i) + m_i^* x_i$$

  Guard bits: $g = \lceil \log_2(n+1) \rceil$

  – Computing elementary residues m_i

  $$m_{i+1}^* = 2m_i^* - M$$

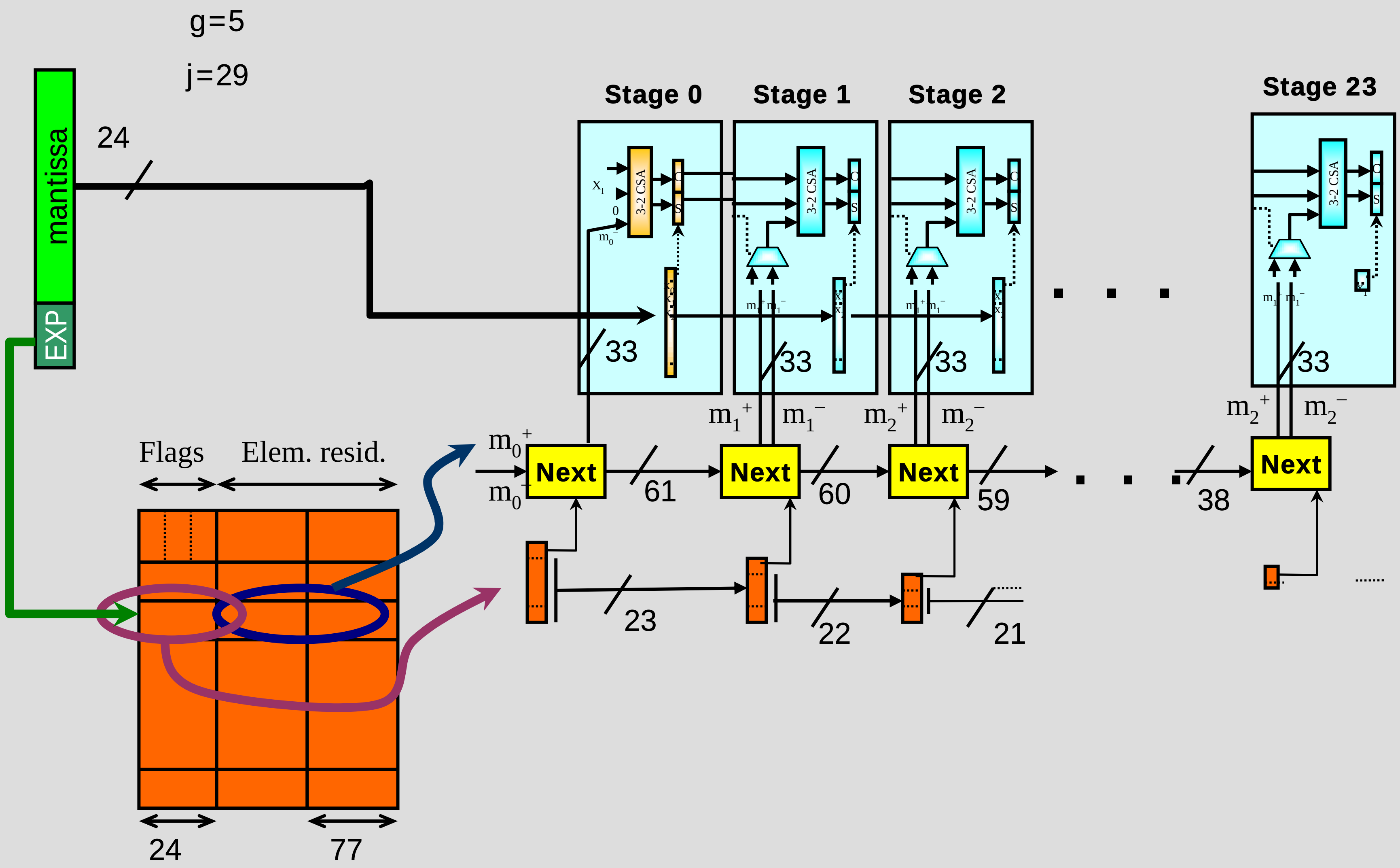
  Guard bits: $j = n + \lceil \log_2(n+1) \rceil$

  – The number of guard bits required for m_i decreases by one in every iteration

    - Hardware reduction in every stage

Pipelined architecture

- Architecture for IEEE-754 single precision ($M=2\pi$, $n=24$)



Future works

- Radix-4 coding $X = x_1 x_2 x_3 \ldots x_i \ldots x_n \quad , x_i \in \{-2,-1,0,1,2\}$
  - Reduces the number of stages by half

$$R(i+1) = R(i) + \sigma_i m_i^*$$

  - Selection function more complex
    - Increases the stage delay

$$\sigma_i = sign(x_i)$$

$$m_i^* = \begin{cases} m_{i+1}^+ & if \quad R(i) < 0 \ \& \ |x_i| = 2 \\ m_i^+ & if \quad R(i) < 0 \ \& \ |x_i| = 1 \\ 0 & if \quad |x_i| = 0 \\ m_i^- & if \quad R(i) \geq 0 \ \& \ |x_i| = 1 \\ m_{i+1}^- & if \quad R(i) \geq 0 \ \& \ |x_i| = 2 \end{cases}$$

Summary and Conclusion

- Novel pipeline architecture for range reduction
  - Carry-save arithmetic
  - Based on double elementary residues
  - Prevent replication of tables of the word-serial version
    - obtaining next elementary residue from the previous one
      - Elementary residue generator
  - Precision 1 ULP
- Valid for floating point reduction
  - IEEE 754 compliant
- Radix-4

  Reduces the number of stages by half

THANK YOU !