

Structuri iterative pentru adunare

Oprițoiu Flavius
flavius.opritoiu@cs.upt.ro

18 septembrie 2023

Introduction

Objective:

- ▶ Construirea unor structuri pentru adunare multi-operand

Implementările hardware ale *algoritmilor iterativi* implică două faze distincte:

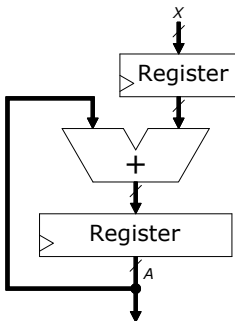
- construirea părții de stare, care stochează starea algoritmului din iterația curentă în iterația următoare
- implemetează partea de prelucrare a datelor, care actualizează starea algoritmului de la o iterație la alta

Adunarea multi-operand secvențială

În fiecare ciclu de tact, un nou operand este livrat în registrul X .
Următorul algoritm calculează suma operandilor și o stochează în acumulatorul A :

```
1:  $A \leftarrow 0$   
2: loop  
3:    $A \leftarrow A + X$   
4: end loop
```

având următoarea implementare hardware:



Problemă rezolvată

Calea de date a unei aplicații criptografice

Exercițiu: Să se construiască calea de date pentru o arhitectură a algoritmului Secure Hash Algorithm 2 (SHA-2) pe 256 de biți (vezi [FIPS15], secțiunea 5.1.1).

Soluție: Unitatea primește la intrare blocuri de 512-biți, pe care le prelucreză secvențial, în vederea determinării rezultatului hash asociat mesajului recepționat. Rezultatul hash este furnizat ca un vector binar pe 256-biți.

Procesarea unui bloc implică următoarele operații:

- ▶ *Message schedule:* extinde cele 16 cuvinte ale blocului recepționat la 64 de cuvinte
- ▶ *Funcția de compresie:* preia un cuvânt furnizat de *message schedule* și actualizează, timp de 64 de iterații, variabilele a , b , c , d , e , f , g și h
- ▶ *Actualizare hash:* adună la valoarea curentă a hash-ului, segmentat în 8 cuvinte, valorile variabilelor de la a la h

Problemă rezolvată (contin.)

Message schedule

Blocul de 512-biți este segmentat în 16 cuvinte a câte 32-biți: M_0 , M_1 , ..., M_{15} , cu M_0 reprezentând cei mai semnificativi 32-biți ai blocului iar M_{15} cei mai puțin semnificativi.

Timp de 64 de iterații, în fiecare ciclu de tact, *message schedule* va construi un nou cuvânt în poziția cea mai puțin semnificativă (primul cuvânt construit îl va succeda pe M_{15} , iar acest prim cuvânt construit va fi succedat de următorul cuvânt construit șamd.). Cuvântul furnizat la ieșire, în fiecare iterație, este M_0 .

La orice moment de timp doar 16 cuvinte sunt necesare pentru a construi următorul cuvânt. În consecință, noul cuvânt va ocupa cea mai puțin semnificativă poziție (M_{15}) toate celelalte cuvinte deplasându-se pe poziția imediat mai semnificativă (M_{15} va ocupa poziția lui M_{14} , M_{14} pe a lui M_{13} , ..., M_1 pe a lui M_0).

Problemă rezolvată (contin.)

Message schedule

Message schedule este descrisă formal în algoritmul de mai jos:

Intrare: Blocul *BLK* ▷ *BLK* poate fi segmentat în 16 cuvinte de 32-biți

Ieșire: Cuvântul M_0 pe 32-biți ▷ Furnizează M_0 în fiecare iterație

```
1: procedure MESSAGE_SCHEDULE(BLK)
2:    $M_0 \leftarrow BLK[511 : 480]$                       ▷ Inițializarea celor 16 cuvinte  $M_i$ 
3:    $M_1 \leftarrow BLK[479 : 448]$ 
4:   ...
5:    $M_{14} \leftarrow BLK[63 : 32]$ 
6:    $M_{15} \leftarrow BLK[31 : 0]$ 
7:   for  $i = 0$  to 63 do                      ▷ Construire cuvânt nou și actualizarea celor 16 cuvinte
8:      $NEW\_WORD \leftarrow \sigma_1(M_{14}) + M_9 + \sigma_0(M_1) + M_0$ 
9:      $M_0 \leftarrow M_1$ 
10:     $M_1 \leftarrow M_2$ 
11:    ...
12:     $M_{14} \leftarrow M_{15}$ 
13:     $M_{15} \leftarrow NEW\_WORD$ 
14:   end for
15: end procedure
```

Operatorul de adunare, $+$, din acest slide și din următoarele, se efectuează (mod 2^{32}).

Problemă rezolvată (contin.)

Message schedule

Funcții $\sigma_0(\alpha)$ și $\sigma_1(\beta)$ sunt definite astfel:

$$\sigma_0(\alpha) = \text{RotireDr}(\alpha, 7) \oplus \text{RotireDr}(\alpha, 18) \oplus \text{DeplasareDr}(\alpha, 3)$$

$$\sigma_1(\beta) = \text{RotireDr}(\beta, 17) \oplus \text{RotireDr}(\beta, 19) \oplus \text{DeplasareDr}(\beta, 10)$$

unde: $\text{RotireDr}(x, p)$ rotește cuvântul x la dreapta cu p biți;

$\text{DeplasareDr}(x, p)$ deplasează cuvântul x la dreapta cu p biți

(adaugă biți de 0 în msb); iar \oplus denotă operatorul SAU-EXCLUSIV

Pentru construirea acestor operatori se pot utiliza funcții Verilog:

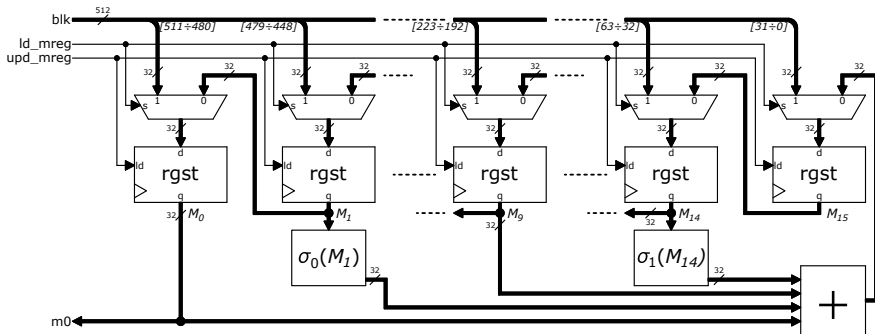
```
1 function [31:0] RotireDr (input [31:0] x, input [4:0] p);  
2     reg [63:0] tmp;  
3     begin  
4         tmp = {x, x} >> p;  
5         RotireDr = tmp[31:0];  
6     end  
7 endfunction
```

Funcția de mai sus se apelează prin: $\text{RotireDr}(\alpha, 7)$

Problemă rezolvată (contin.)

Message schedule

Componenta căii de date care implementează *message schedule* este reprezentată în figura de mai jos:



Notă: modulul *rgst* este disponibil [► aici](#)

Problemă rezolvată (contin.)

Funcția de compresie și actualizarea rezultatului hash

Rezultatului hash, de 256-biți, este format din 8 cuvinte pe 32-biți: H_0 , H_1 , H_2 , H_3 , H_4 , H_5 , H_6 și H_7 , cu H_0 fiind cel mai semnificativ iar H_7 cel mai puțin semnificativ.

Funcția de compresie utilizează 8 variabile pe 32-biți: a , b , c , d , e , f , g și h . Cele 8 variabile sunt inițializate la valoarea cuvintelor H_0 , ..., H_7 ale rezultatului hash curent. Ulterior, pe durata a 64 de iterații, variabilele a până la h sunt actualizate pe baza valorilor lor curente, a cuvântului M_0 furnizat de *message schedule* și a unei constante de rundă, $K(i)$.

La finalul celor 64 de iterații, rezultatul hash este *actualizat*, adunând la fiecare din cele 8 cuvinte H_0 până la H_7 valorile variabilelor a până la h .

Pentru fiecare bloc recepționat se reia operația de compresie urmată de actualizarea rezultatului hash.

Problemă rezolvată (contin.)

Funcția de compresie și actualizarea rezultatului hash

Intrare: Sunt recepționate blocuri mesaj

Ieșire: Cuvinte rezultat hash $H_0, H_1, H_2, H_3, H_4, H_5, H_6, H_7$

1: **procedure** SHA256

2: InițializareCuvinteRezultatHash()

3: **do**

4: $a \leftarrow H_0$

5: $b \leftarrow H_1$

6: ...

7: $h \leftarrow H_7$

8: **for** $i = 0$ **to** 63 **do**

9: $T_1 \leftarrow h + \Sigma_1(e) + Ch(e, f, g) + K(i) + M_0$ ▷ M_0 din expresia lui T_1

10: $T_2 \leftarrow \Sigma_0(a) + Maj(a, b, c)$ ▷ este furnizat de *message*

11: $h \leftarrow g; g \leftarrow f; f \leftarrow e$ ▷ *scheduler* care, întrucât

12: $e \leftarrow d + T_1$ ▷ execută același număr

13: $d \leftarrow c; c \leftarrow b; b \leftarrow a$ ▷ de 64 de iterații, poate rula

14: $a \leftarrow T_1 + T_2$ ▷ în paralel cu această buclă

15: **end for**

16: $H_0 \leftarrow H_0 + a$

17: $H_1 \leftarrow H_1 + b$

18: ...

19: $H_7 \leftarrow H_7 + h$

20: **while not** *last block*

21: **end procedure**

Problemă rezolvată (contin.)

Funcția de compresie și actualizarea rezultatului hash

Algoritmul SHA-256 utilizează următoarele funcții:

$$\Sigma_0(x) = \text{RotireDr}(x, 2) \oplus \text{RotireDr}(x, 13) \oplus \text{RotireDr}(x, 22)$$

$$\Sigma_1(x) = \text{RotireDr}(x, 6) \oplus \text{RotireDr}(x, 11) \oplus \text{RotireDr}(x, 25)$$

$$\text{Ch}(x, y, z) = (x \text{ and } y) \oplus ((\text{not } x) \text{ and } z)$$

$$\text{Maj}(x, y, z) = (x \text{ and } y) \oplus (x \text{ and } z) \oplus (y \text{ and } z)$$

Operatorii and și not de mai sus sunt de tip bit-wise (operează asupra unui vector binar, la nivelul individual al bitului).

Constantele $K(i)$, indexate de iterația curentă, i , sunt specificate de standard ([FIPS15], secțiunea 4.2.2):

i	$K(i)$
0	32'h428a2f98
1	32'h71374491
2	32'hb5c0fbcf
...
63	32'hc67178f2

Problemă rezolvată (contin.)

Funcția de compresie și actualizarea rezultatului hash

Cele 8 cuvinte ale rezultatului hash, $H_0, H_1, H_2, H_3, H_4, H_5, H_6, H_7$, sunt inițializate cu valorile specificate de standard ([FIPS15], secțiunea 5.3.3):

leșire: Inițializare cuvinte rezultat hash $H_0, H_1, H_2, H_3, H_4, H_5, H_6, H_7$

1: **procedure** INIȚIALIZARECUVINTEREZULTATHASH

2: $H_0 \leftarrow 32'h6a09e667$

3: $H_1 \leftarrow 32'hbb67ae85$

4: $H_2 \leftarrow 32'h3c6ef372$

5: $H_3 \leftarrow 32'ha54ff53a$

6: $H_4 \leftarrow 32'h510e527f$

7: $H_5 \leftarrow 32'h9b05688c$

8: $H_6 \leftarrow 32'h1f83d9ab$

9: $H_7 \leftarrow 32'h5be0cd19$

10: **end procedure**

Referințe bibliografice

[FIPS15] National Institute of Standards and Technology, "FIPS PUB 180-4: Secure Hash Standard," Gaithersburg, MD 20899-8900, USA, Tech. Rep., Aug. 2015. [Online]. Available: <http://dx.doi.org/10.6028/NIST.FIPS.180-4> (Last accessed 06/04/2016).