Arhitectura calculatoarelor (laborator 10 - S10)

02.12.2024

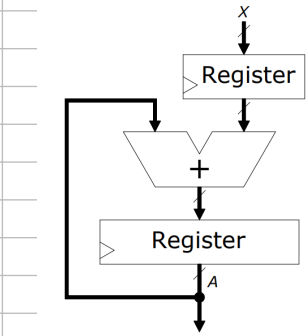
Obiective : Construirea unor structuri pentru adunarea multi-operand

Pentru algoritmii iterativi avem 2 faze:

1. construirea partii de stare, care stocheaza starea algoritmului din iteratia curenta in iteratia urmatoare
2. implemeteaza partea de prelucrare a datelor, care actualizeaza starea algoritmului de la o iteratie la alta

Adunarea multi-operand secvențială

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



A ← 0

loop:

A ← A + X

end loop

→ ex: problema 1 propusă

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 prelucrează 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*

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 senificativi.

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 cuvant 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).

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

procedure MESSAGESCHEDULE(BLK)
  $M_0 \leftarrow BLK[511:480]$ ▷ Inițializarea celor 16 cuvinte M_i
  $M_1 \leftarrow BLK[479:448]$
  ...
  $M_{14} \leftarrow BLK[63:32]$
  $M_{15} \leftarrow BLK[31:0]$
  for $i = 0$ **to** 63 **do** ▷ Construire cuvânt nou și actualizarea celor 16 cuvinte
    $NEW\_WORD \leftarrow \sigma_1(M_{14}) + M_9 + \sigma_0(M_1) + M_0$
    $M_0 \leftarrow M_1$
    $M_1 \leftarrow M_2$
    ...
    $M_{14} \leftarrow M_{15}$
    $M_{15} \leftarrow NEW\_WORD$
  end for
end procedure

Operatorul de adunare, $+$, din acest slide și din următoarele, se efectuează $\pmod{2^{32}}$.

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

$$\sigma_0(\alpha) = RotireDr(\alpha, 7) \oplus RotireDr(\alpha, 18) \oplus DeplasareDr(\alpha, 3)$$
$$\sigma_1(\beta) = RotireDr(\beta, 17) \oplus RotireDr(\beta, 19) \oplus DeplasareDr(\beta, 10)$$

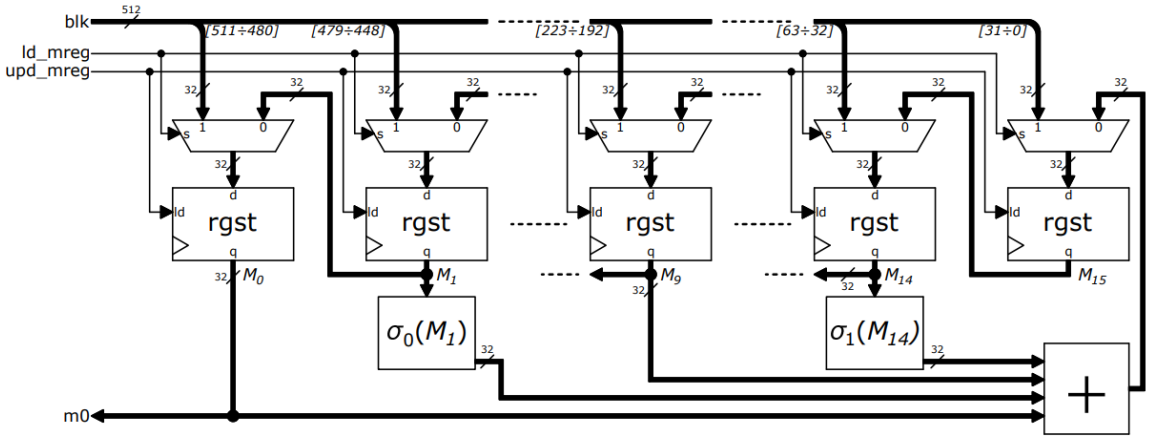
unde: $RotireDr(x, p)$ rotește cuvântul x la dreapta cu p biți; $DeplasareDr(x, p)$ deplasează cuvântul x la dreapta cu p biți (adaugă biți de 0 în msb); iar $\oplus$ denotă operatorul SAU-EXCLUSIV

Pentru construirea acestor operatori se pot utiliza funcții Verilog:

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

Funcția de mai sus se apelează prin: `RotireDr(alpha,7)`

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



Modulul rgst :

```
    module rgst #(
  parameter w=8
)(
  input clk, rst_b, ld, clr, input [w-1:0] d, output reg [w-1:0] q
);
  always @ (posedge clk, negedge rst_b)
    if (!rst_b)          q <= 0;
    else if (clr)        q <= 0;
    else if (ld)         q <= d;
endmodule
```

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.

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 ← H0
5:         b ← H1
6:         ...
7:         h ← H7
8:         for i = 0 to 63 do
9:             T1 ← h + Σ1(e) + Ch(e, f, g) + K(i) + M0      ▷ M0 din expresia lui T1
10:            T2 ← Σ0(a) + Maj(a, b, c)                     ▷ este furnizat de message
11:            h ← g; g ← f; f ← e                           ▷ scheduler care, întrucât
12:            e ← d + T1                                    ▷ execută același număr
13:            d ← c; c ← b; b ← a                           ▷ de 64 de iterații, poate rula
14:            a ← T1 + T2                                   ▷ în paralel cu această buclă
15:        end for
16:        H0 ← H0 + a
17:        H1 ← H1 + b
18:        ...
19:        H7 ← H7 + h
20:    while not last block
21: end procedure
```

Algoritmul SHA-256 utilizează următoarele funcții:

$$\Sigma_0(x) = RotireDr(x,2) \oplus RotireDr(x,13) \oplus RotireDr(x,22)$$
$$\Sigma_1(x) = RotireDr(x,6) \oplus RotireDr(x,11) \oplus RotireDr(x,25)$$
$$Ch(x,y,z) = (x \underline{\ and\ } y) \oplus ((\underline{not}\ x) \underline{\ and\ } z)$$
$$Maj(x,y,z) = (x \underline{\ and\ } y) \oplus (x \underline{\ and\ } z) \oplus (y \underline{\ and\ } z)$$

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

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):

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

```
1: procedure InițializareCuvinteRezultatHash
2:     H0 ← 32'h6a09e667
3:     H1 ← 32'hbb67ae85
4:     H2 ← 32'h3c6ef372
5:     H3 ← 32'ha54ff53a
6:     H4 ← 32'h510e527f
7:     H5 ← 32'h9b05688c
8:     H6 ← 32'h1f83d9ab
9:     H7 ← 32'h5be0cd19
10: end procedure
```