Lecţia 6:

Aritmetica pentru calculator - II

G Stefănescu — Universitatea Bucureşti

Arhitectura sistemelor de calcul, Sem.1
Octombrie 2016—Februarie 2017

După: D. Patterson and J. Hennessy, Computer Organisation and Design

Aritmetica pentru calculator

Cuprins:

- Numere (cu si fara semn)
- Adunare si scadere
- Operatii logice
- Unitatea aritmetica si logica
- Adunare rapida
- *Inmultire*
- Impartire
- Operatii cu numere reale
- Concluzii, diverse, etc.

Inmultire

Inmultire:

- Incepem cu metoda uzuală (şcolară) de înmulţire:
- Primul operand este *deânmulţitul D* (`1000`), al doilea *înmulţitorul I* (`1001`), iar rezultatul se numeşte *produs P* (`1001000`).
- Remarcăm că produsul a două numere de m şi n biţi necesită $m+n$ biţi.

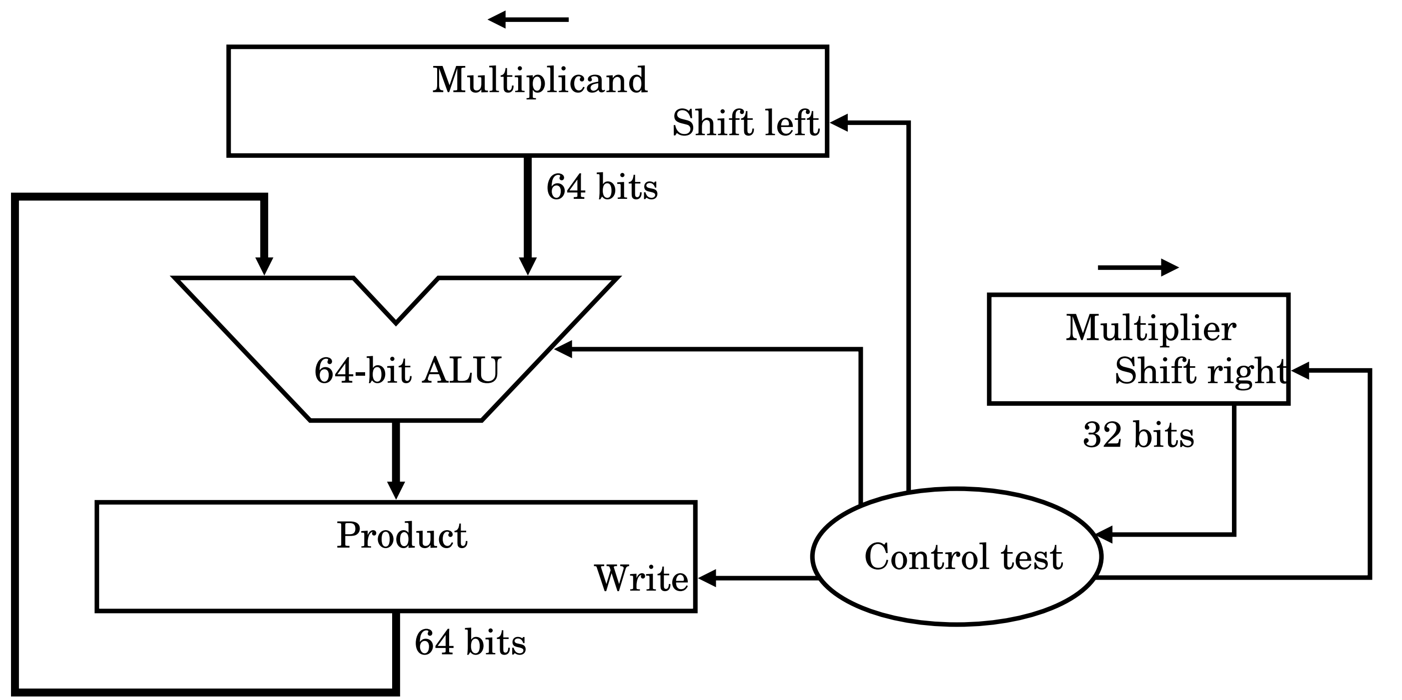
```
      1000
x     1001
   -------
      1000
     0000
    0000
   1000
   -------
   1001000
```

- Simplificat (în prezentul caz binar) algoritmul se reduce la:
  - pentru fiecare poziţie 1 din înmulţitor plasăm în locul respectiv deânmulţitul, altfel punem 0;
  - adunăm numerele.
- Cum vom vedea, în hardware metoda este mult optimizată...

Prima metoda de inmultire (in hardware)

Prima metoda de inmultire:

- Folosim un *ALU pe 64 biţi*.
- Folosim regiştri de 64 biţi pentru *D* şi *P*; folosim un registru de 32 biţi pentru *I*.



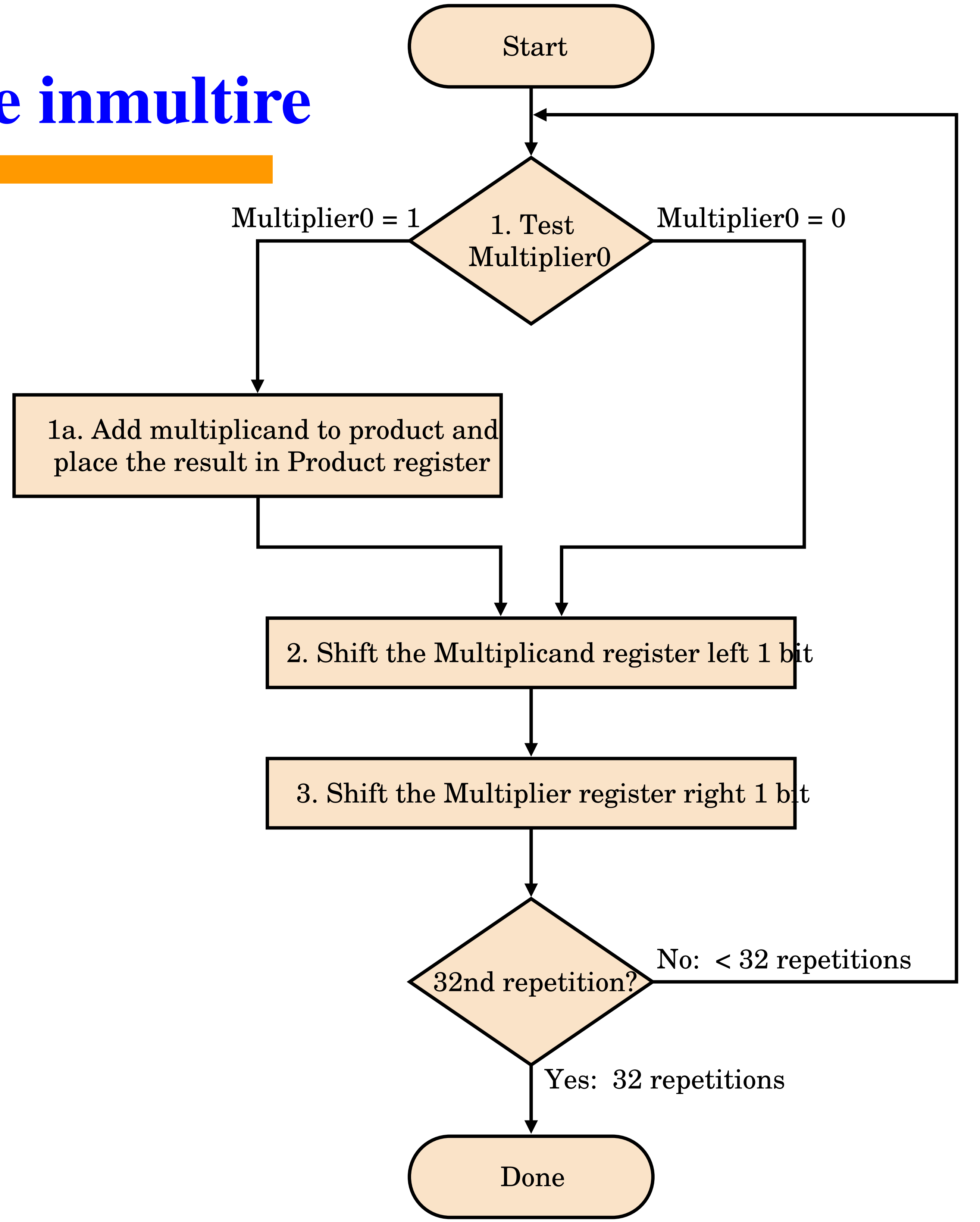
- Când bitul 0 din înmulţitor este 1 se adună produsul (parţial) cu deânmulţitul (iniţial produsul este 0).
- Se shiftează cu 1 bit deânmulţitul la *stînga*, iar înmulţitorul la *dreapta*.
- Se repetă ultimii doi paşi de 32 ori.

..Prima metoda de inmultire

Prima metoda de inmultire (cont.)

Figura conţine schema logică a algoritmului.

`Multiplier0` este bitul 0 din înmulţitor.



..Prima metoda de inmultire

Exemplu (simplificat, pe 4 biţi):

- Fie D, I, P cei 3 regiştri pentru deânmulţit, înmulţitor şi produs.
- Ilustrăm cazul 2×3;
  Iniţial $D = \mathtt{0000\,0010}, I = \mathtt{0011}$, iar $P = \mathtt{0000\,0000}$.
- Execuţie (4 paşi):

  Pas 0: –Bitul 0 din I este 1: $P := P + D = \mathtt{0000\,0010}$;
  –Shift D (stânga), I (dreapta): $D = \mathtt{0000\,0100}$, $I = \mathtt{0001}$;

  Pas 1: –Bitul 0 din I este 1: $P := P + D = \mathtt{0000\,0110}$;
  –Shift D, I: $D = \mathtt{0000\,1000}$, $I = \mathtt{0000}$;

  Pas 2: –Bitul 0 din I este 0: nimic ($P := P$);
  –Shift D, I: $D = \mathtt{0001\,0000}$, $I = \mathtt{0000}$;

  Pas 3: –Bitul 0 din I este 0: nimic ($P := P$);
  –Shift D, I: $D = \mathtt{0010\,0000}$, $I = \mathtt{0000}$.

- Rezultat: $P := \mathtt{0000\,0110}$.

..Prima metoda de inmultire

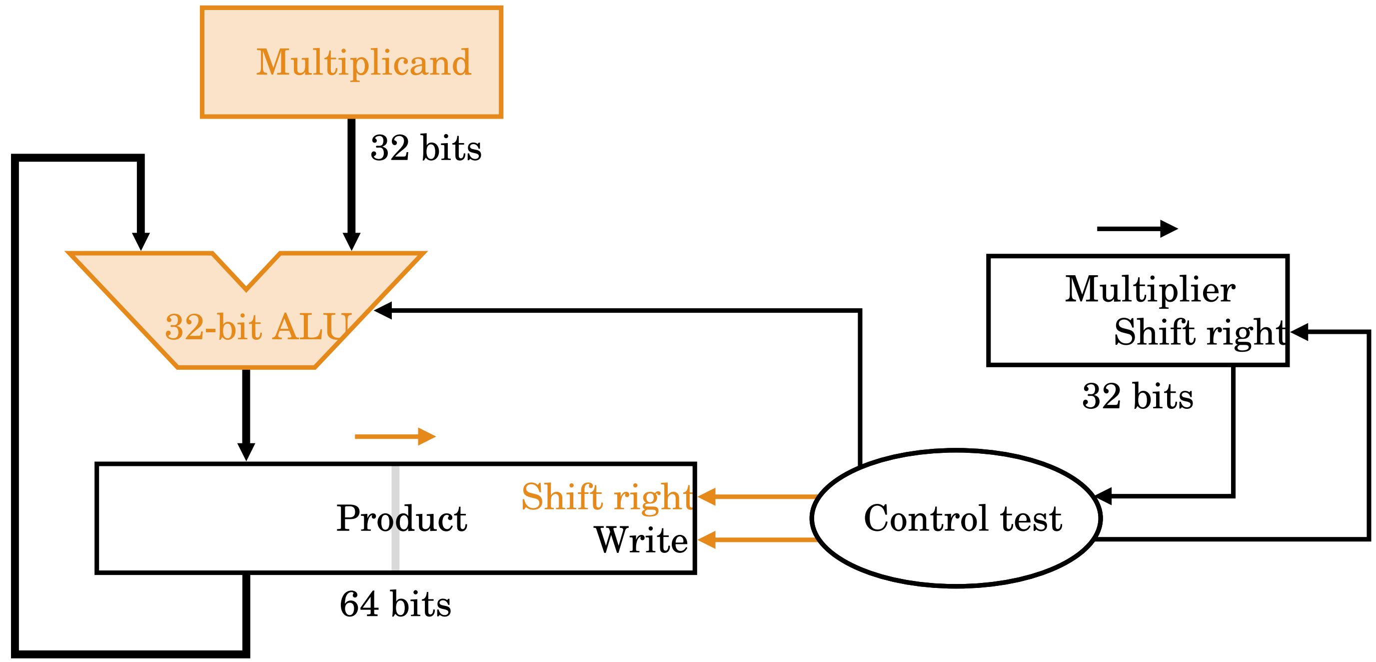
Analiză:

- Cu 3 operaţii pe pas (adunare $P+D$, shift P, shift I) şi 32 paşi, avem aproximativ *100 cicluri* de ceas pentru o *înmulţire*.
- Analize statistice arată că *adunarea şi scăderea* sunt de 5 până la 100 ori *mai frecvente* decât înmulţirea.
- Cu legea *"Execută rapid operaţiile frecvente"*, o implementare cu multiple cicluri de ceas pentru înmulţire este *acceptabilă*.
- Totuşi, dorim implementări ale înmulţirii *mai eficiente*...

A doua metoda de inmultire (in hardware)

A doua metoda:

- Anterior, *jumătate* din cei 64 biţi ai lui *D* erau mereu *zero*.
- Folosim acum un *ALU pe 32 biţi*.



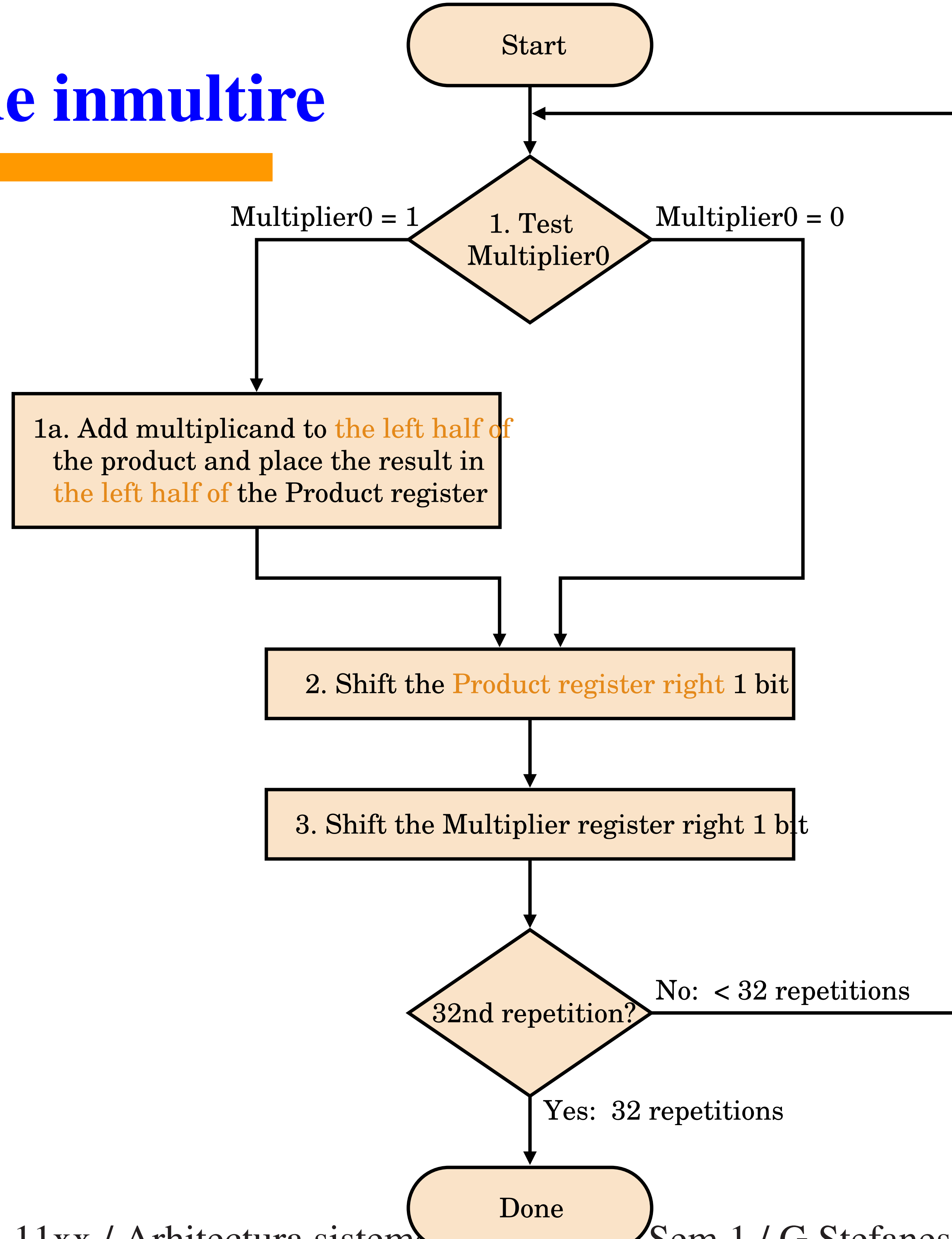
- P foloseşte 64 biţi, din care prima jumătate `P[63-32]` este pentru ieşirea `Result` din ALU. Iniţial $P = 0$.
- Când bitul 0 din înmulţitor este 1 se adună prima jumătate din produs cu deânmulţitul, i.e., `P[63-32] = P[63-32] + D`.
- Se shiftează cu 1 bit *la dreapta* atât înmulţitorul, cât şi produsul (ultimul pe toţi 64 biţi).
- Se repetă ultimii doi paşi de 32 ori.

..A doua metoda de inmultire

A doua metoda de inmultire (cont.)

Figura conţine schema logică a algoritmului.

De notat că registrul `Product` este pe 64 biţi, adică 2 regiştri uzuali.



..A doua metoda de inmultire

Exemplu (simplificat, pe 4 biţi):

- Fie D, I, P ca mai sus.
- Ilustrăm cazul 2×3;
  Iniţial $D = \mathtt{0010}, I = \mathtt{0011}$, iar $P = \mathtt{0000\ 0000}$.
- Execuţie (4 paşi):

  P0: –Bitul 0 din I este 1: $P = \mathtt{0010\ 0000}$ ($P[63\text{-}32] := P[63\text{-}32] + D$);
  –Shift P, I cu 1 bit dreapta: $P = \mathtt{0001\ 0000}$, $I = \mathtt{0001}$;

  P1: –Bitul 0 din I este 1: $P = \mathtt{0011\ 0000}$ ($P[63\text{-}32] := P[63\text{-}32] + D$);
  –Shift P, I: $P = \mathtt{0001\ 1000}$, $I = \mathtt{0000}$;

  P2: –Bitul 0 din I este 0: nimic ($P := P$);
  –Shift P, I: $P = \mathtt{0000\ 1100}$, $I = \mathtt{0000}$;

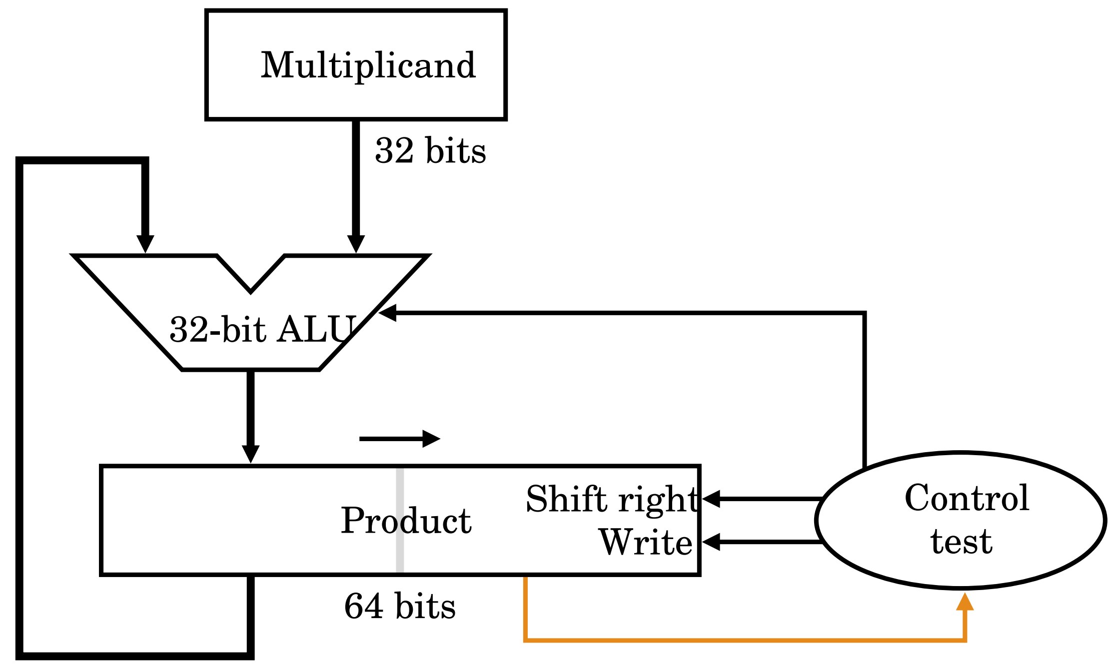
  P3: –Bitul 0 din I este 0: nimic ($P := P$);
  –Shift P, I: $P = \mathtt{0000\ 0110}$, $I = \mathtt{0000}$.

- Rezultat: $P := \mathtt{0000\ 0110}$.

Metoda finala de inmultire (in hardware)

Metoda finala:

- Plasăm *I în jumătatea din dreapta a lui P*, care iniţial era goală.
- Cum ambii regiştri *P, I se shiftează sincron* la dreapta cu 1 bit, *nu se pierde nimic* din *P* ori *I*.



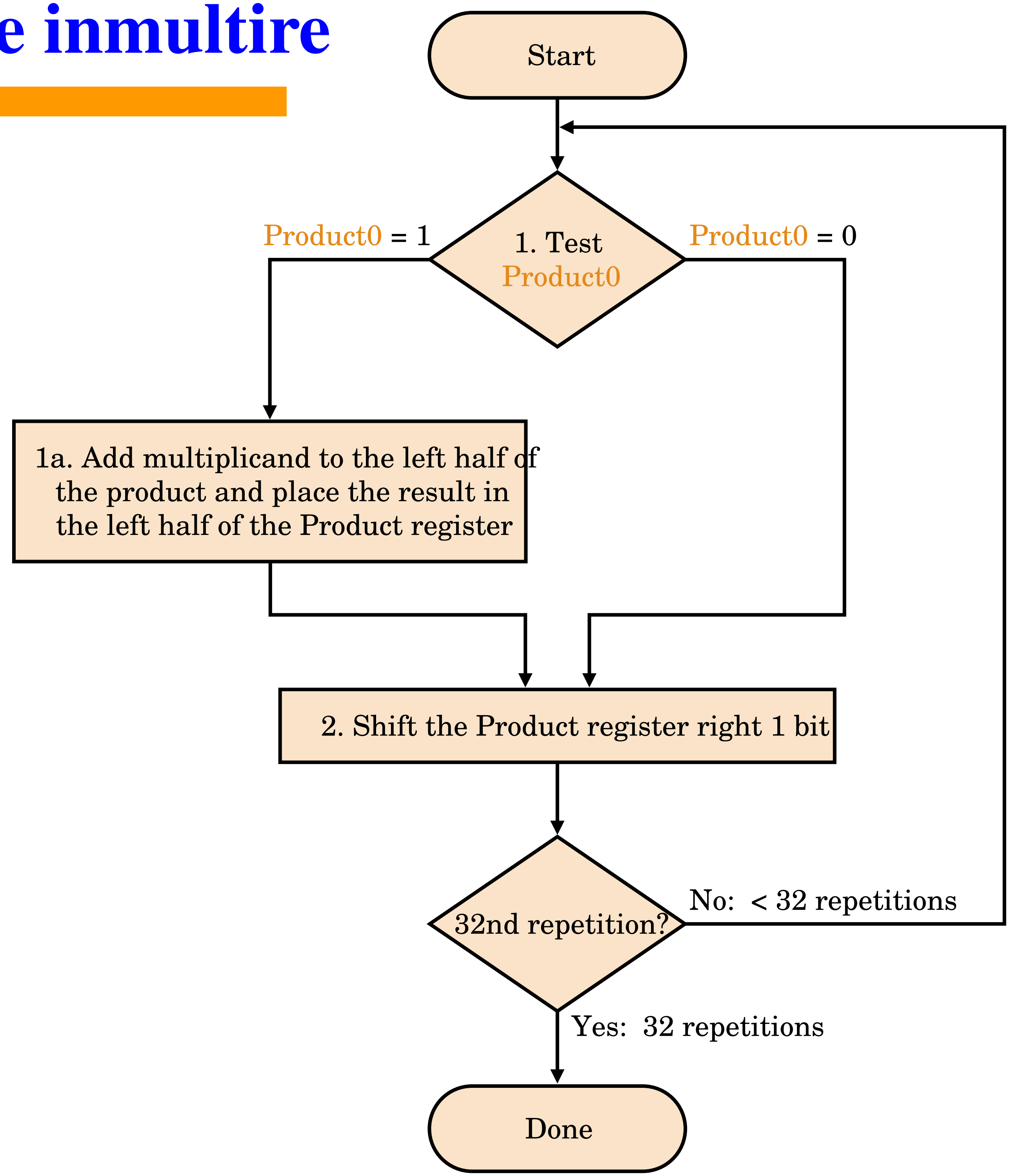
- *P* foloseşte 64 biţi, din care prima jumătate `P[63-32]` este pentru ieşirea `Result` din ALU, iar a doua jumătate `P[31-0]` este pentru *I*. Iniţial `P[63-32] = 0` şi `P[31-0] = I`.
- In rest, algorithmul este similar.

..Metoda finala de inmultire

Metoda finala (cont.)

Figura conţine schema logică a algoritmului.

De notat că registrul `Product` este pe 64 biţi, initial având înmulţitorul în a 2-a jumătate.



..Metoda finala de inmultire

Exemplu (simplificat, pe 4 biţi):

- Fie D, I, P ca mai sus; *I se pune în* $P[31\text{-}0]$.
- Ilustrăm cazul 2×3;
  Iniţial $D = \texttt{0010}$, $P = \texttt{0000 0011}$.
- Execuţie (4 paşi):
  P0: –Bitul 0 din P este 1: $P = \texttt{0010 0011}$ ($P[63\text{-}32] := P[63\text{-}32] + D$);
  –Shift P 1 bit dreapta: $P = \texttt{0001 0001}$;
  P1: –Bitul 0 din P este 1: $P = \texttt{0011 0001}$ ($P[63\text{-}32] := P[63\text{-}32] + D$);
  –Shift P: $P = \texttt{0001 1000}$
  P2: –Bitul 0 din P este 0: nimic ($P := P$);
  –Shift P: $P = \texttt{0000 1100}$;
  P3: –Bitul 0 din P este 0: nimic ($P := P$);
  –Shift P: $P = \texttt{0000 0110}$.
- Rezultat: $P := \texttt{0000 0110}$.

Inmultire cu semn

Inmultire cu semn:

- Extensia la întregi *cu semn* este simplă:
  - se lucrează cu 31 biţi, deci folosim 31 de iterate, neglijând bitul de semn;
  - în final, semnul produsului este `xor` de semnele termenilor (i.e., "$+$" dacă termenii au acelaşi semn şi "$-$" dacă semenle diferă).
- Algorithmul 3 (cel final), funcţionează corect şi pentru numere cu semn.

Algorithmul Booth de inmultire

Algorithmul Booth:

- Inmulţirea cu grupuri de 0 din înmulţitor se reduce la shiftări.
- Observaţia lui Booth a fost că se poate face o *simplificare similară pentru grupurile de 1*:
  - un grup de k de 1, anume $\texttt{11...1}_{\texttt{doi}}$, este egal cu diferenţa $2^k - 1$;
  - o înmulţire cu un astfel de grup se reduce la
    –*o adunare* a deânmulţitului D (pentru bitul 1 din 2^k) şi
    –*o scădere* a lui D (pentru -1);
  - exemplu: $\texttt{0010} \times \texttt{0110}$

    P0: $\texttt{11} = 2^2 - 1$, deci $\texttt{0}\underline{\texttt{11}}\texttt{0} = \texttt{1}\underline{\texttt{00}}\texttt{0} - \texttt{0}\underline{\texttt{01}}\texttt{0}$;
    P1: $\texttt{0010} \times \texttt{0}\underline{\texttt{11}}\texttt{0} = \texttt{0010} \times \texttt{1}\underline{\texttt{00}}\texttt{0} - \texttt{0010} \times \texttt{0}\underline{\texttt{01}}\texttt{0}$
- Este performant (uzual, shiftările sunt mai rapide ca adunările).

..Algorithmul Booth de inmultire

Algorithmul Booth:

- Clasificăm biţii astfel

Bit curent	Bit dreapta	Descriere	Exemplu
1	0	incepe un grup de 1	0000111110000
1	1	mijloc de grup de 1	00001111000
0	1	sfarsit de grup de 1	000011111000
0	0	mijloc de grup de 0	000011111000

- P0: In funcţie de biţii "curent+dreapta" executăm următoarele:
  - 00: mijloc de grup de 0 - nu facem operatii aritmetice;
  - 01: sfârşit de grup de 1 - adunăm D in jumatarea stanga a lui P;
  - 10: inceput de grup de 1 - scădem D in jumatarea stanga a lui P;
  - 11: mijloc de grup de 1 - nu facem operatii aritmetice;
- P1: Ca anterior, shiftăm produsul P cu un bit la dreapta.

..Algorithmul Booth de inmultire

Exemplu (simplificat, pe 4 biţi):

- Fie D, I, P ca mai sus; I se pune în $P[31\text{-}0]$.
- Ilustrăm cazul 2×3; Se adaugă un *bit fictiv la deapta lui P*, iniţial 0; Deci, iniţial $D = \mathtt{0010}$, $P = \mathtt{0000}\ \underline{\mathtt{0011}}\ \mathtt{0}$;
- Execuţie (4 paşi):

  P0: –10, pozitii finale in P: $P = \mathtt{1110\ 0011\ 0}$ ($P[63\text{-}32] := P[63\text{-}32] - D$);
  –Shift P: $P = \mathtt{1111\ 0}\underline{\mathtt{001\ 1}}$;

  P1: –11, finale: nici o operatie
  –Shift P: $P = \mathtt{1111\ 10}\underline{\mathtt{00\ 1}}$;

  P2: –01, finale: $P = \mathtt{0001\ 1000\ 1}$ ($P[63\text{-}32] := P[63\text{-}32] + D$);
  –Shift P: $P = \mathtt{0000\ 110}\underline{\mathtt{0\ 0}}$;

  P3: –00, finale: nici o operatie
  –Shift P cu 1 bit dreapta: $P = \mathtt{0000\ 0110}\ \underline{\mathtt{0}}$.

- Rezultat: $P := \mathtt{0000\ 0110}$.

..Algorithmul Booth de inmultire

Comentarii:

- Shift-ul la dreapta folosit în P este cel *arithmetic*, nu cel logic (anume cel în care se propagă bitul de semn).
- Algorithmul Booth funcţionează corect atât pentru numere pozitive, cât şi pentru *numere negative*.
- Se poate folosi pe *grupuri de biti* pentru a construi înmulţitoare rapide.

Inmultire in MIPS

Inmultire in MIPS:

- Există două instrucţiuni:
  - `mult` pentru înmultire de întregi cu semn şi
  - `multu` pentru înmultire fără semn.
- In MIPS se foloseşte *o pereche de 2 regiştri* pentru produs, numiţi `Hi` şi `Lo`.
- Pentru accesul la regiştri se folosesc instrucţiunile `mflo` (*move from Low*) şi `mfhi` (*move from High*).
- Ambele versiuni *neglijează overflow-ul*, care poate fi detectat de software.

..Inmultire in MIPS

Operatii de inmultire in MIPS:

Tip	Instructiune	Exemple	Semantica	Comentarii
A	multiply	`mult $s2,$s3`	`Hi,Lo=$s2+$s3`	produs cu semn pe 64 biti in Hi,Lo
A	multiply unsigned	`multu $s2,$s3`	`Hi,Lo=$s2+$s3`	produs fara semn pe 64 biti in Hi,Lo
A	move from Hi	`mfhi $s1`	`$s1=Hi`	se obtine o copie a lui Hi
A	move from Lo	`mflo $s1`	`$s1=Lo`	se obtine o copie a lui Lo

(Legenda: A = Instructiune aritmetica)

Aritmetica pentru calculator

Cuprins:

- Numere (cu si fara semn)
- Adunare si scadere
- Operatii logice
- Unitatea aritmetica si logica
- Adunare rapida
- Inmultire
- *Impartire*
- Operatii cu numere reale
- Concluzii, diverse, etc.

Impartire

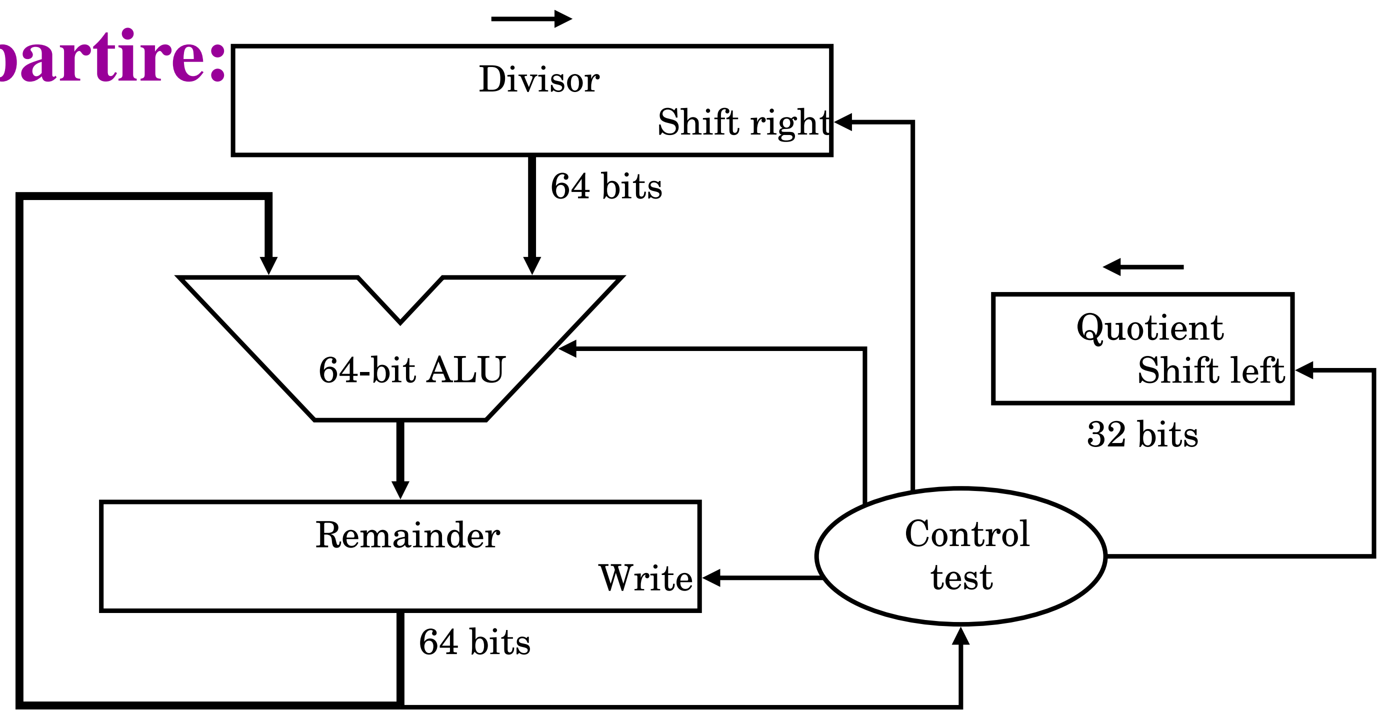
Impartire:

```
                                 1001   Cat
                               -------
Impartitor 1000                1001010   Deimpartit
                               -1000
                               -----
                                   10
                                   101
                                   1010
                                  -1000
                                  -----
                                     10   Rest
```

- Metoda uzuală de împărţire se poate prezenta ca în exemplu.
- Primul operand este *deâmpărţitul* *D* (`1001010`), al doilea *împărţitorul I* (`1000`), iar rezultatul este dat de o pereche *cât C* (`1001`) şi *rest R* (`10`).
- Relaţia dintre mărimile de mai sus este $D = I \times C + R$.
- Metodă: Se află cifrele câtului rând pe rând, scăzând I din D (ca în exemplu) pentru fiecare cifră 1 obţinută în C.

Prima metoda de impartire (in hardware)

Prima metoda de impartire:



- Folosim un *ALU pe 64 biţi*.
- Folosim regiştri de 64 biţi pentru R şi I; folosim un registru de 32 biţi C.
- Iniţial R conţine D, împărţitorul I este în jumătatea *stângă* din registru (`Divisor`), iar $C = 0$.
- Se scade I din R; dacă $R - I \geq 0$, shiftăm C la stânga inserând 1; dacă $R - I < 0$, restaurăm R (adunând I) şi shiftăm C la stânga inserând 0. Apoi shiftăm I cu 1 bit la dreapta.
- Se repetă de 32 ori.

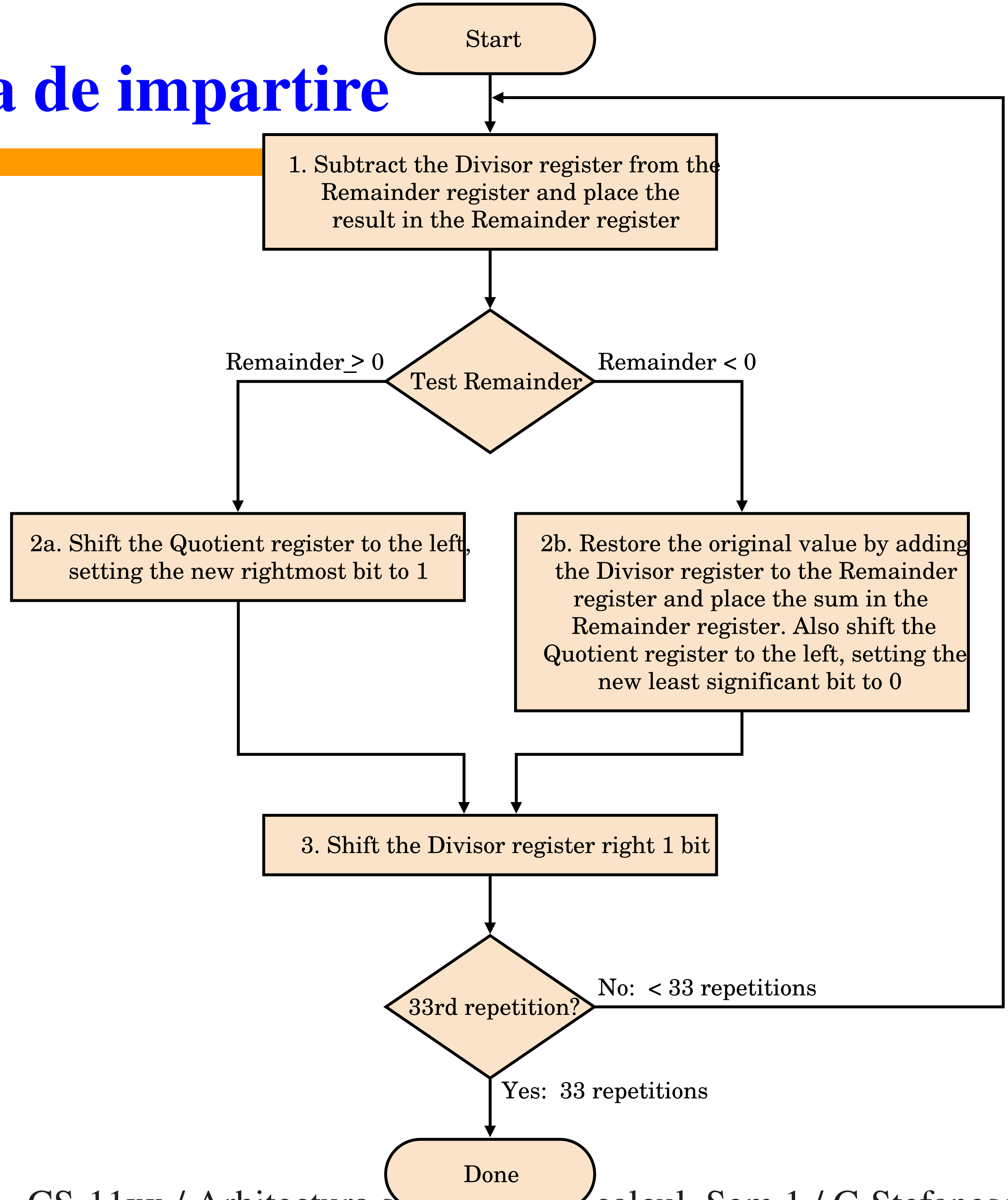
..Prima metoda de impartire

Prima metoda de impartire (cont.)

Figura conţine schema logică a algoritmului.

`Remainder` conţine iniţial deîmpărţitul.

Iniţial, registrul `Divisor` conţine împărţitorul în jumătatea sa stângă.



..Prima metoda de impartire

Exemplu (pe 4 biţi): Impărţim 7 la 2

Pas	Instructiuni	Cat	Impartitor	Rest
0	valori initiale	0000	0010 0000	0000 0111
1:1	R=R-I	0000	0010 0000	1110 0111
1:2b	R<0: R=R+I, sll Q, Q_0=0	0000	0010 0000	0000 0111
1:3	shift I dreapta	0000	0001 0000	0000 0111
2:1	R=R-I	0000	0001 0000	1111 0111
2:2b	R<0: R=R+I, sll Q, Q_0=0	0000	0001 0000	0000 0111
2:3	shift I dreapta	0000	0000 1000	0000 0111
3:1	R=R-I	0000	0000 1000	1111 1111
3:2b	R<0: R=R+I, sll Q, Q_0=0	0000	0000 1000	0000 0111
3:3	shift I dreapta	0000	0000 0100	0000 0111
4:1	R=R-I	0000	0000 0100	0000 0011
4:2a	R>0: sll Q, Q_0=1	0001	0000 0100	0000 0011
4:3	shift I dreapta	0001	0000 0010	0000 0011
5:1	R=R-I	0001	0000 0010	0000 0001
5:2a	R>0: sll Q, Q_0=1	0011	0000 0010	0000 0001
5:3	shift I dreapta	0011	0000 0001	0000 0001

..Prima metoda de impartire

Prima metoda (cont.)

- Calculatorul nu poate “vedea” în avans dacă împarţitorul este mai mic şi trebuie să testeze acest lucru scăzând I din R.

- După scădere, se compară rezultatul cu zero, testând bitul de semn.

- Dacă rezultatul este negativ, se revine la R-ul anterior, adunând la loc I.

- De notat ca avem nevoie de $n+1$ iteraţii pentru a obţine rezultatul corect pentru n biţi.

A 2-a metoda de impartire

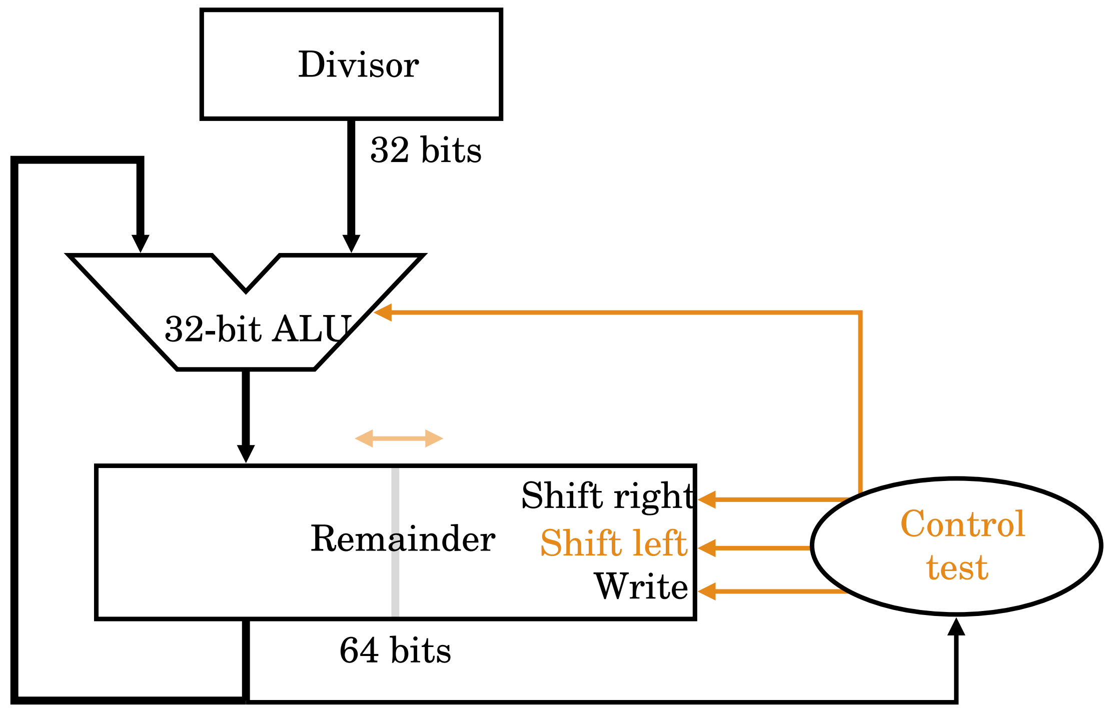
A 2-a metoda:

- Ca anterior, se poate folosi doar un *ALU pe 32 biti*, căci doar jumătate din I conţine informaţie utilă.
- Acum, *R se shiftează la stânga* spre a se alinia cu I.
- Algoritmul nu poate produce un 1 în prima fază, căci rezultatul are fi prea lung pentru C (i.e., sunt $n+1$ iteraţii). Soluţie: Se *permută operaţiile de shift şi scădere*, eliminând o iterată.
- Câtul final este în C, iar restul este *în jumătatea stângă* a lui R.
- Detaliile se omit – se vor incorpora în următoarea versiune (finală).

Metoda finala de impartire (in hardware)

Metoda finala:

- Plasăm *C în jumătatea din dreapta a lui R*.
- Cum ambii regiştri *R, C se shiftează sincron* cu 1 la stânga, *nu se pierde nimic* din *R* ori *C*.



- *R* foloseşte 64 biţi, din care prima jumătate `P[63-32]` este pentru ieşirea `Result` din ALU. In a doua jumătate `P[31-0]`, în final va fi *C*. Iniţial `P[63-0]` conţine *D*.
- Detalii în schema logică care urmează.

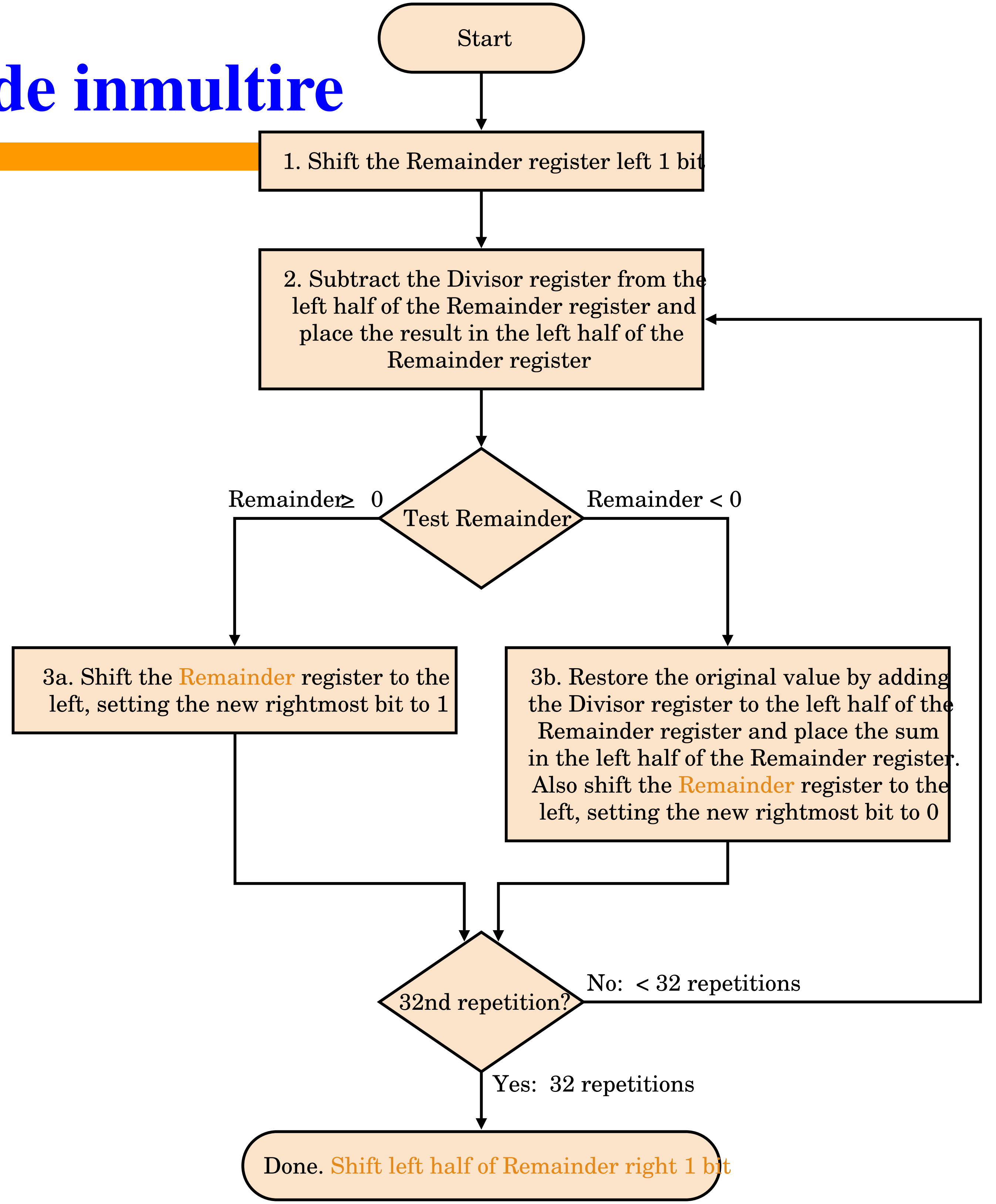
..Metoda finala de inmultire

Metoda finala (cont.)

Figura conţine schema logică a algoritmului.

De notat că registrul R (`Remainder`) este pe 64 biţi.

Initial R conţine deâmpărţitul; în final, $R[63-32]$ conţine restul, iar $R[31-0]$ conţine câtul.



..Metoda finala de inmultire

Exemplu (simplificat, pe 4 biţi): Impărţim 7 la 2

Pas	Instructiuni	Impartitor	Rest
0	valori initiale	0010	0000 0111
0:1	shift R 1 bit stânga	0010	0000 1110
1:2	R=R-I	0010	1110 1110
1:3b	R<0: R=R+I, sll R, R_0=0	0010	0001 1100
2:2	R=R-I	0010	1111 1100
2:3b	R<0: R=R+I, sll R, R_0=0	0010	0011 1000
3:2	R=R-I	0010	0001 1000
3:3a	R>0: sll R, R_0=1	0010	0011 0001
4:2	R=R-I	0010	0001 0001
4:3a	R>0: sll R, R_0=1	0010	0010 0011
	shift R[63-31] 1 bit dreapta	0010	0001 0011

Impartire in MIPS

Impartire in MIPS:

- Există două instrucţiuni:
  - `div` pentru împărţire de întregi cu semn şi
  - `divu` pentru împărţire fără semn.
- Se folosesc aceeaşi 2 regiştri `Hi,Lo` şi pentru împărţire; iniţial, ei conţin deîmparţitul, iar în final rezultatele (cât, rest).
- Ambele versiuni *neglijează overflow-ul*.

..Impartire in MIPS

Operatii de impartire in MIPS:

Tip	Instructiune	Exemple	Semantica	Comentarii

A	divide	`div $s2,$s3`	`Lo=$s2/$s3`
			`Hi=`
			`$s2 mod $s3`
				Lo contine catul;
				Hi contine restul

A	divide unsigned	`div $s2,$s3`	`Lo=$s2/$s3`
			`Hi=`
			`$s2 mod $s3`
				Lo contine catul;
				Hi contine restul

(Legenda: A = Instructiune aritmetica)

Aritmetica pentru calculator

Cuprins:

- Numere (cu si fara semn)
- Adunare si scadere
- Operatii logice
- Unitatea aritmetica si logica
- Adunare rapida
- Inmultire
- Impartire
- *Operatii cu numere reale*
- Concluzii, diverse, etc.

Virgula mobila

Virgula mobila: (engl. *floating-point*, pe scurt `fp`)

- Numerele *reale* pot fi reprezentate cu virgulă (engl. punct), e.g., $\pi = 3.141592\ldots_{\text{zece}}$.
- Pentru numere mari/mici folosim o *reprezentare normalizată*, inmulţind cu puteri ale bazei, e.g. $0.000000001 = 1.0_{\text{zece}} \times 10^{-9}$, $315576000 = 3.15576 \times 10^{9}$.
- Similar, în calculatoare folosim baza 2 şi *reprezentarea în virgulă mobilă*

  $$1.xxxxxxxx_{\text{doi}} \times 2^{yyyy}$$

  Secvenţa *xxxxxxxx* se numeşte *mantisă* (*significand*), iar *yyyy* *exponent*.
- Virgula de sus este *mobilă*, căci, folosind exponenţi, o putem muta standard după prima cifră semnificativă.

Virgula mobila, in calculator

Virgula mobila, in calculator:

- Reprezentarea *MIPS în virgulă mobilă* foloseşte *1 bit de semn, 8 pentru exponent, şi 23 pentru mantisă*

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
s	exponent								mantisa																						

- Numerele reprezentate variază (aproximativ) între $2.0_{zece} \times 10^{-38}$ şi $2.0_{zece} \times 10^{38}$. Dacă se depăşesc limitele, apare *overflow*.
- Reprezentarea de sus este reprezentare de *precizie simplă* (*single precision*).

Virgula mobila, in calculator

Virgula mobila, in calculator (cont.)

- Pentru precizie mai mare, folosim *precizie dublă* (*double precision*), anume reprezentarea pe 2 regiştri cu *1 bit de semn, 11 pentru exponent, şi 52 pentru mantisă*

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
s	exponent											mantisa ...																			

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
... mantisa (continuare)																															

- Numerele reprezentate variază (aproximativ) între $2.0_{zece} \times 10^{-308}$ şi $2.0_{zece} \times 10^{308}$, dar marele câştig nu este intervalul, ci acurateţea spotită (mai multe cifre semnificative).
- Formatele de mai sus sunt standardizate, i.e., parte din *IEEE 754 floating-point standard*.

Standardul IEEE 754

IEEE 754:

- O problemă apare cu reprezentarea exponenţilor pozitivi/negativi. Convenţia este că
  - $00...0_{doi}$ este "cel mai negativ" exponent, iar
  - $11...1_{doi}$ este "cel mai pozitiv" exponent;

  Folosim o *notaţie polarizată* (*biased notation*), anume valoarea reală se obţine adunând un număr de *polarizare* (*bias*).
- Formula generală este

$$(semn, exp, mantisa) \mapsto (-1)^{semn} \times (1 + mantisa) \times 2^{exp - polarizare}$$

- Pentru IEEE 754 de simplă precizie, numărul de polarizare este *127*; pentru dublă precizie *1023*.

..Standardul IEEE 754

Exemple: Reprezentare IEEE 754 pentru -0.75_{zece} în simplă şi dublă precizie:

- In simplă precizie:

$$\begin{aligned} -0.75_{zece} = -3/4_{zece} &= -11/2^2{}_{doi} = -0.11_{doi} \\ &= -1.1 \times 2^{-1} = (-1)^1 \times (1+.1) \times 2^{126-127}, \end{aligned}$$

deci $(semn, exp, mantisa) = (1, 126, .1)$ şi reprezentarea este:

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	8 biti								23 biti																						

..Standardul IEEE 754

Exemple (cont.)

- In dublă precizie,

$$-0.75_{\text{zece}} = (-1)^1 \times (1 + .1) \times 2^{1022-1023},$$

deci $(semn, exp, mantisa) = (1, 1022, .1)$ şi reprezentarea este:

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	1	1	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	11 biti											20 biti																			

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
32 biti																															

Adunare in virgula mobila

Adunare: Ilustrăm algorithmul pe cazul zecimal, cu 4 cifre după virgulă, cu adunarea $9.999 \times 10^1 + 1.610 \times 10^{-1}$.

- Pas1: Se aliniază virgula la numărul cu exponentul mai mic spre a obtine acelaşi exponent
  $1.610 \times 10^{-1} = 0.01610 \times 10^1 = 0.016 \times 10^1$.

- Pas2: Se adună mantisele
  $9.999 + 0.016 = 10.015$.

- Pas3: Se normalizează suma obţinută (pot apare situaţii de overflow ori underflow la exponent)
  $10.015 \times 1^1 = 1.0015 \times 10^2$.

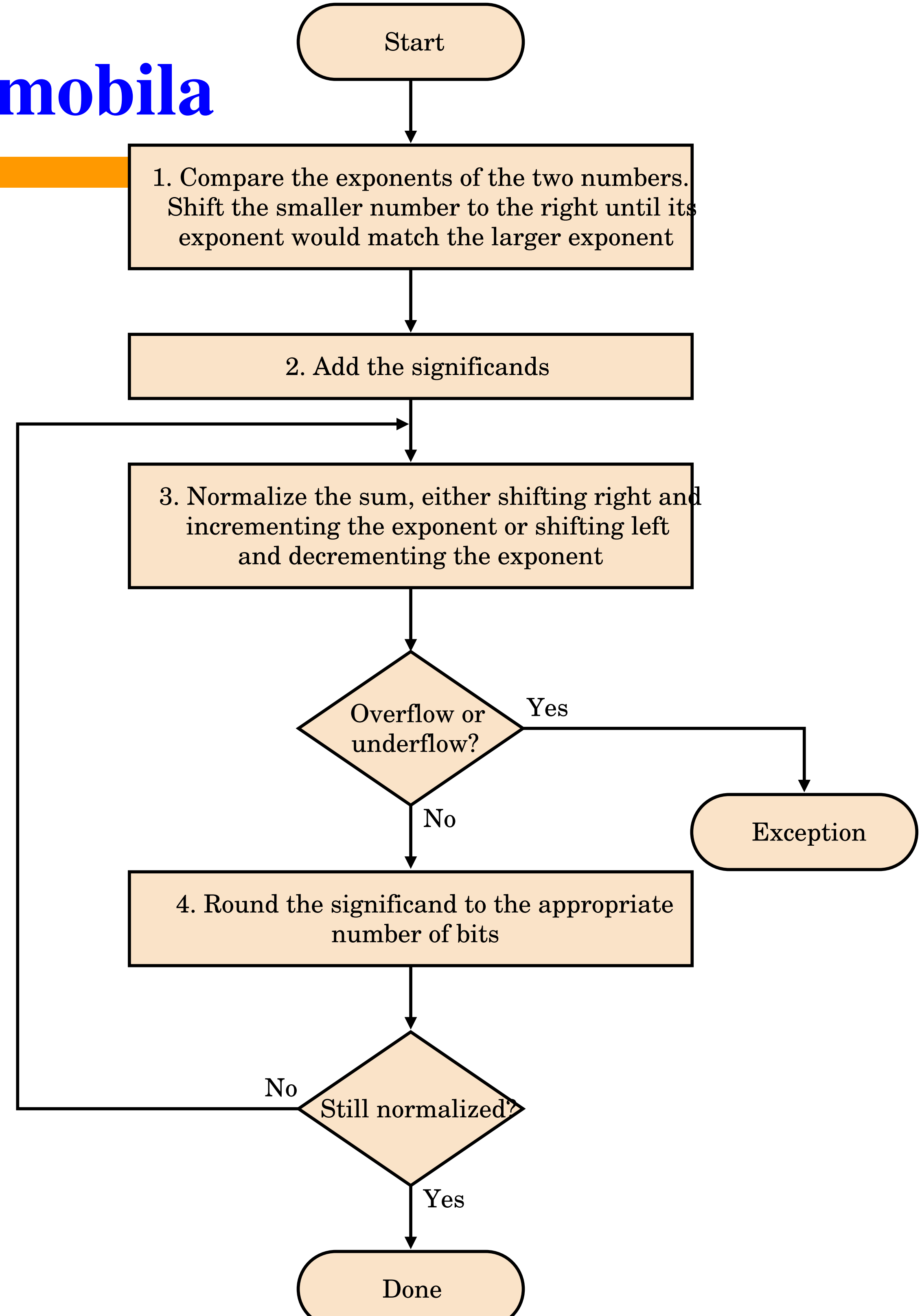
- Pas4: Se rotunjeşte rezultatul
  $1.0015 \times 10^2 = 1.002 \times 10^2$.

..Adunare in virgula mobila

Adunare in virgula mobila (cont.)

Figura conţine schema logică a algoritmului de adunare în virgulă mobilă.

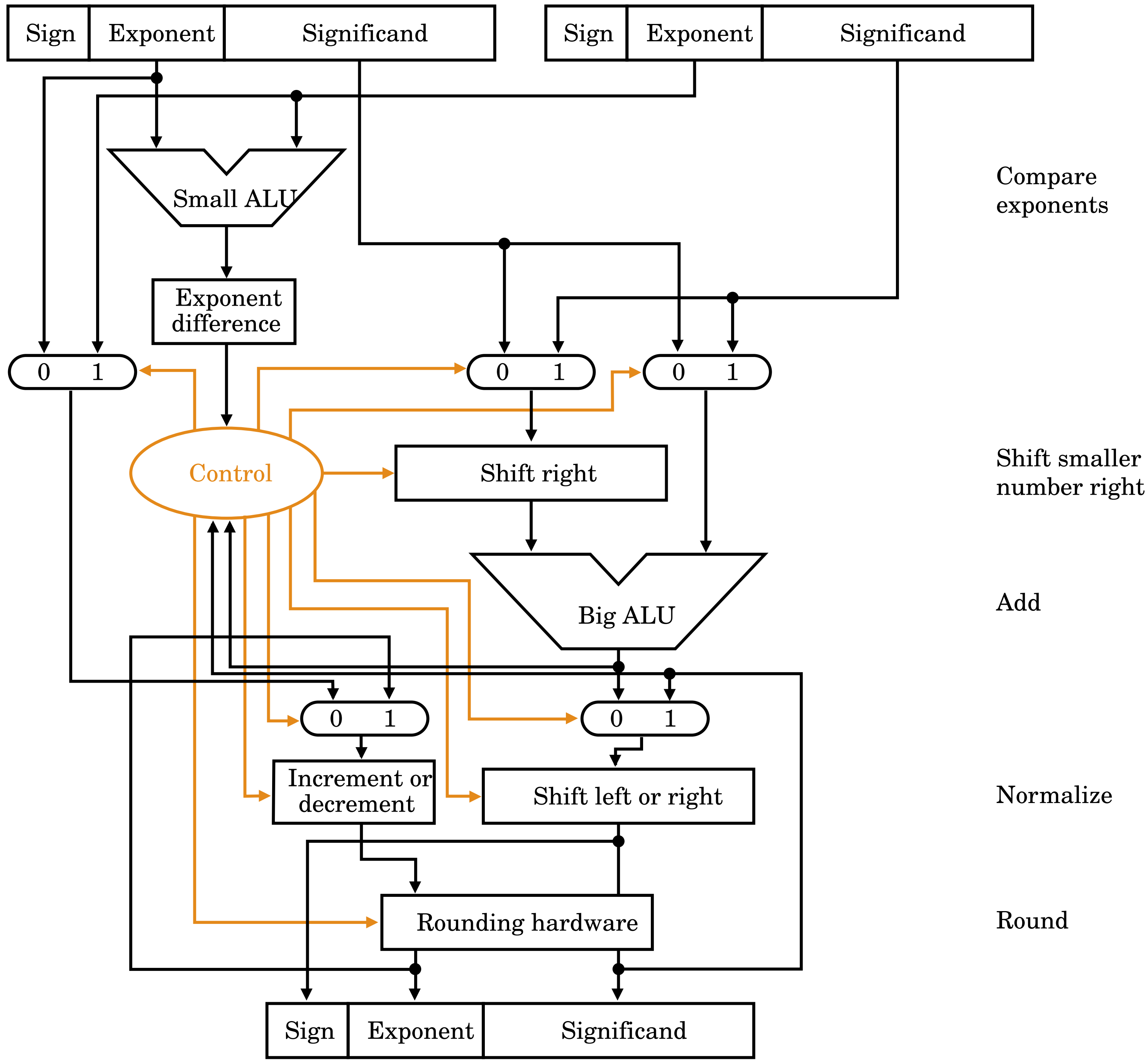
(De notat că rotunjirea poate necesita încă o normalizare.)



..Adunare in virgula mobila

Adunare (cont.)

Figura conţine shema hardware pentru adunare în virgula mobilă.



Inmultire in virgula mobila

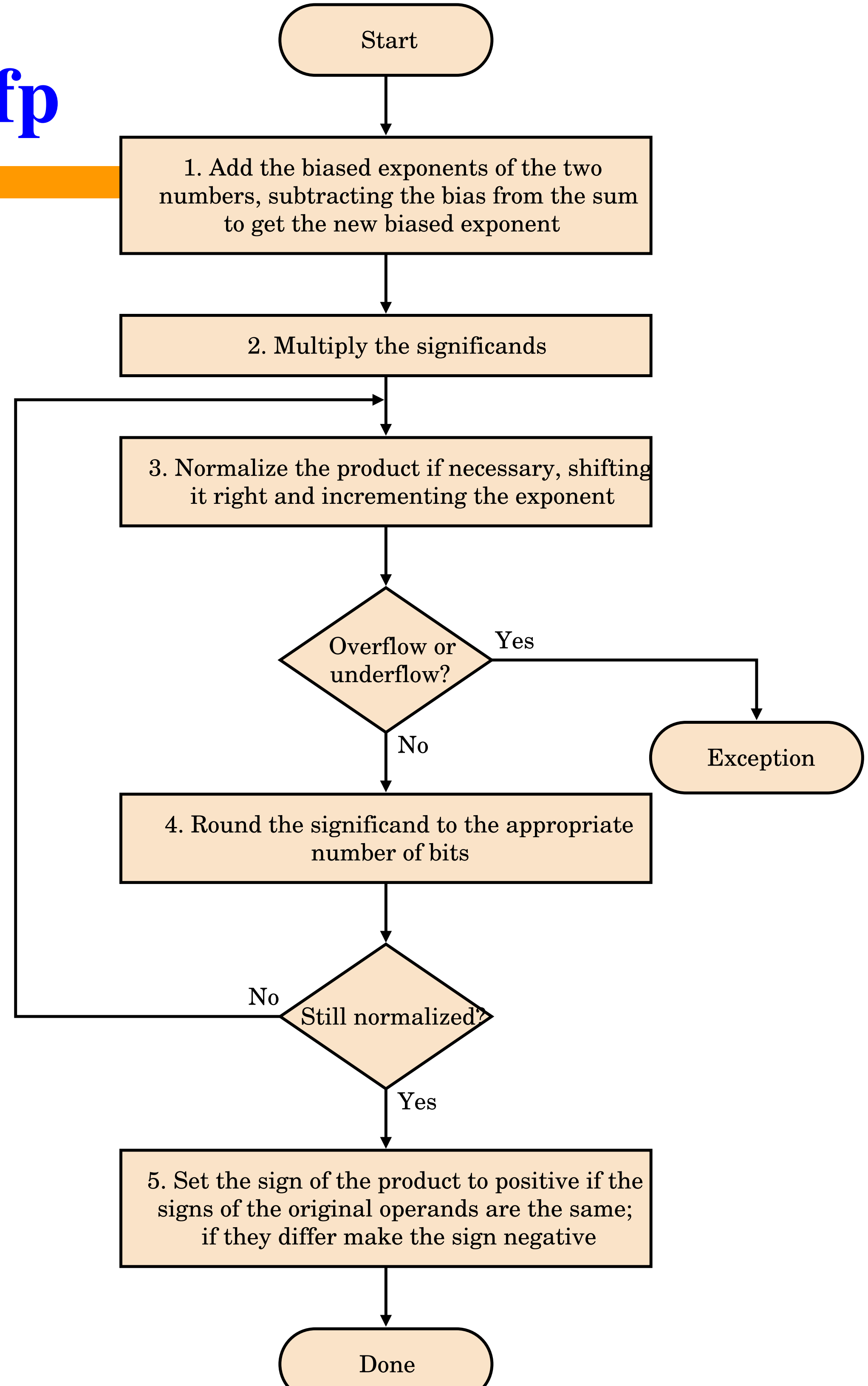
Inmultire: Ilustrăm algorithmul pe cazul zecimal, cu 4 cifre după virgulă, cu inmulţirea $(1.110 \times 10^{10}) \times (9.200 \times 10^{-5})$.

- Pas1: Exponentul rezultatului este suma exponenţilor $10 + (-5) = 5$
  (Cu reprezentarea polarizată, $(10+127)+(-5+127)\underline{-127} = 5+127$!)
- Pas2: Inmulţim mantisele $1.110 \times 9.200 = 10.212000$, deci 10.212×10^5.
- Normalizăm rezultatul $10.212 \times 10^5 = 1.0212 \times 10^6$.
- Rotunjim rezultatul $1.0212 \times 10^6 = 1.021 \times 10^6$.
- Calculăm semnul (produsul semnelor) şi obţinem produsul $+1.021 \times 10^6$.

..Inmultire fp

Inmultire in virgula mobila (cont.)

Figura conţine schema logică a algoritmului de înmulţire în virgulă mobilă.



Suport MIPS pentru virgula mobila

Suport MIPS:

- MPIS are suport pentru reprezentarea IEEE 754, cu operaţii de simplă şi dublă precizie:
  - *adunare:* `add.s` / `add.d`;
  - *scădere:* `sub.s` / `sub.d`;
  - *înmulţire:* `mult.s` / `mult.d`;
  - *împărţire:* `div.s` / `div.d`;
  - *comparaţie:* `c.x.s` / `c.x.d`, unde `x` este: `eq` (egal) / `neq` (ne-egal) / `lt` (mai mic) / `le` (mai mic ori egal) / `gt` (mai mare) / `gt` (mai mare ori egal)
  - *salt condiţionat*: `bclt` (la adevăr) / `bclf` (la fals).

Suport MIPS pentru virgula mobila

Suport MIPS (cont.)

- Există regiştrii în virgulă mobilă speciali, notaţi `$f0,$f1,...,$f31`.
- Există operaţii de incărcare/memorare speciale pentru regiştrii de tip `$f`, anume: `lwcl` (incărcare) / `swcl` (memorare).
- In dublă precizie, se folosesc perechi de regiştri alăturaţi, adresa fiind a celui par.

Aritmetica pentru calculator

Cuprins:

- Numere (cu si fara semn)
- Adunare si scadere
- Operatii logice
- Unitatea aritmetica si logica
- Adunare rapida
- Inmultire
- Impartire
- Operatii cu numere reale
- *Concluzii, diverse, etc.*

Concluzii, diverse, etc.

A se insera...