Arhitectura sistemelor de calcul

- Prelegerea 4 -

Reprezentarea numerelor în calculator

Ruxandra F. Olimid

Facultatea de Matematică şi Informatică

Universitatea din Bucureşti

Cuprins

Sistemul binar

- *Întrebare:* Care este reprezentarea lui 21 în binar (bază 2)?
- *Răspuns:* $21 = 10101_{(2)}$

...	**16** (2^4)	**8** (2^3)	**4** (2^2)	**2** (2^1)	**1** (2^0)
21 =	16		+4		+1
	1	**0**	**1**	**0**	**1**

- *Întrebare:* Care este reprezentarea lui $11011_{(2)}$ în zecimal (bază 10)?
- *Răspuns:* $11011_{(2)} = 27$

...	**16** (2^4)	**8** (2^3)	**4** (2^2)	**2** (2^1)	**1** (2^0)
	1	**1**	**0**	**1**	**1**
27 =	16	+8		+2	+1

Reprezentarea în calculator

- *Întrebare:* Care este dimensiunea regiştrilor generali pentru un procesor pe 32 de biţi (ex. MIPS32)?
- *Răspuns:* 32 de biţi
- *Întrebare:* Cum se va reprezenta deci numărul 21 dacă este introdus într-un registru general?
- *Răspuns:* Se completează până la 32 de biţi cu 0 la stânga:

b31	b30	b29	…	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
0	0	0	0…0	0	0	0	0	0	1	0	1	0	1

Reprezentarea în calculator

b31	b30	b29	...	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
0	0	0	0…0	0	0	0	0	0	1	0	1	0	1

- *Bitul cel mai semnificativ* (*msb* = *m*ost *s*ignificant *b*it) este bitul cel mai din stânga; (bitul 31, cu valoarea 0 în exemplul de mai sus)
- *Bitul cel mai puţin semnificativ* (*lsb* = *l*east *s*ignificant *b*it) este bitul cel mai din dreapta; (bitul 0, cu valoarea 1 în exemplul de mai sus)

- *Întrebare:* Care este ponderea msb (în puterile lui 2)?
- *Răspuns:* 2^{31}
- *Întrebare:* Care este ponderea lsb (în puterile lui 2)?
- *Răspuns:* 2^{0}

MIPS32

- *Întrebare:* Ce instrucţiuni / pseudoinstrucţiuni cunoaşteţi pentru a introduce valoarea zecimală 21 în registrul $s1?

- *Răspuns:* Exemple sunt instrucţiunea `ori` (sau logic pe biţi cu valoare imediată) sau pseudoinstrucţiunea `li` (load immediate):

```
[00400010] 34110015  ori $17, $0, 21          ; 8: li $s1,21
[00400014] 34110015  ori $17, $0, 21          ; 9: ori $s1, $0, 21
[00400018] 3402000a  ori $2, $0, 10           ; 10: li $v0,10
[0040001c] 0000000c  syscall                  ; 11: syscall
```

- *Întrebare:* Ce valoare este vizualizată în simulator în registrul $s1 după execuţie? De ce?

- *Răspuns:* Valoarea 0x15, reprezentarea în hexazecimal a lui 21.

Sistemul hexazecimal

- *Întrebare:* Care este reprezentarea lui 65 în hexazecimal (bază 16)?
- *Răspuns:* 65 = 0x41

	…	**256** (16^2)	**16** (16^1)	**1** (16^0)
65 =			+4x16	+1x1
		0	**4**	**1** => **23 = 0x41**

- *Întrebare:* Care este reprezentarea lui 0x65 în zecimal (bază 10)?
- *Răspuns:* 0x65 = 101

	…	**256** (16^2)	**16** (16^1)	**1** (16^0)
			96	**5** => **0x65=101**
0x65 =			+6x16	+5x1

MIPS32

- *Întrebare:* Ce instrucţiuni / pseudoinstrucţiuni cunoaşteţi pentru a introduce valoarea hexazecimală 65 în registrul $s1?

- *Răspuns:* Exemple sunt instrucţiunea `ori` (sau logic pe biţi cu valoare imediată) sau pseudoinstrucţiunea `li` (load immediate):

```
[00400000] 34110065  ori $17, $0, 101          ; 8: li $s1,0x65
[00400004] 34110065  ori $17, $0, 101          ; 9: ori $s1, $0, 0x65
[00400008] 3402000a  ori $2, $0, 10            ; 10: li $v0,10
[0040000c] 0000000c  syscall                   ; 11: syscall
```

- Un număr în baza 16 se reprezintă prin **0x** înaintea valorii

Reprezentarea în calculator

- *Întrebare:* Câte numere se pot scrie pe 32 de biţi?
- *Răspuns:* 2^{32}
- *Întrebare:* Câte numere se pot scrie pe n biţi?
- *Răspuns:* 2^n, demonstrat imediat prin inducţie. Intuitiv:
  - ✓ Pentru $n = 1$, sunt 2 numere (0 şi 1).
  - ✓ Apoi, pentru fiecare bit adăugat, se dublează numărul anterior de numere pentru că acesta poate fi 0 sau 1.

0	<nr. de la pasul anterior>
1	<nr. de la pasul anterior>

Reprezentarea în calculator

- *Întrebare:* Care sunt numerele naturale care se pot scrie pe 32 de biţi (considerăm deci ca numerele sunt toate pozitive)?
- *Răspuns:* $(0; 2^{32}-1)$

- Numerele întregi necesită şi reprezentarea numerelor negative (în *complement faţă de doi*)

- Prin reprezentarea în complement faţă de 2 pe 32 de biţi se pot reprezenta numerele din intervalul $(-2^{31}; 2^{31}-1)$

- Terminologie:
  - ✓ numere *fară semn (unsigned)*
  - ✓ numere *cu semn (signed)*

Tipuri de date în C

- Limitele pentru tipurile de date din C sunt date de utilizarea bitului de semn în reprezentarea în complement faţă de 2:

Tip de date	Bytes	Alte denumiri	Interval de valori
int	4	signed	$(-2^{31}, 2^{31}-1)$ (–2 147 483 648 ; 2147 483 647)
unsigned int	4	unsigned	$(0; 2^{32}-1)$ (0; 4 294 967 295)
short	2	short int, signed short int	$(-2^{15}; 2^{15}-1)$ (–32 768; 32 767)
unsigned short	2	unsigned short int	$(0; 2^{16}-1)$ (0; 65 535)

MIPS32

- În funcție de instrucțiune, numerele sunt considerate:
  - *fară semn* (*unsigned*) – MIPS32: primesc sufixul **u** (ex. `addu, subu`)
  - *cu semn* (*signed*) – MIPS32: nu sunt sufixate în MIPS (ex. `add, sub`)

```
addu $t1,$t2,$t3
subu $t4,$t5,$t6
```

Numerele din regiştrii sunt considerate **fără semn**

```
add $t1,$t2,$t3
sub $t4,$t5,$t6
```

Numerele din regiştrii sunt considerate **cu semn**

Reprezentarea în complement faţă de 2

0111 1111 1111 1111 1111 1111 1111 $1111_{(2)}$ = 2 147 483 647 $(2^{31} - 1)$

0111 1111 1111 1111 1111 1111 1111 $1110_{(2)}$= 2 147 483 646

... ...

0000 0000 0000 0000 0000 0000 0000 $0001_{(2)}$= 1

0000 0000 0000 0000 0000 0000 0000 $0000_{(2)}$= 0

1111 1111 1111 1111 1111 1111 1111 $1111_{(2)}$= -1

1111 1111 1111 1111 1111 1111 1111 $1110_{(2)}$= -2

... ...

1000 0000 0000 0000 0000 0000 0000 $0001_{(2)}$= -2 147 483 647

1000 0000 0000 0000 0000 0000 0000 $0000_{(2)}$= -2 147 483 648 (-2^{31})

- Valoarea numerelor în zecimal se obţine aplicând formula:

$$\mathrm{b31} \cdot (-2^{31}) + \mathrm{b30} \ \cdot 2^{30} + \mathrm{b29} \ \cdot 2^{29} + \ \ldots + \mathrm{b0} \cdot 2^{0}$$

Reprezentarea în complement faţă de 2

0111 1111 1111 1111 1111 1111 1111 $1111_{(2)}$ = 2 147 483 647 $(2^{31} - 1)$

0111 1111 1111 1111 1111 1111 1111 $1110_{(2)}$= 2 147 483 646

... ...

0000 0000 0000 0000 0000 0000 0000 $0001_{(2)}$= 1

0000 0000 0000 0000 0000 0000 0000 $0000_{(2)}$= 0

1111 1111 1111 1111 1111 1111 1111 $1111_{(2)}$= -1

1111 1111 1111 1111 1111 1111 1111 $1110_{(2)}$= -2

... ...

1000 0000 0000 0000 0000 0000 0000 $0001_{(2)}$= -2 147 483 647

1000 0000 0000 0000 0000 0000 0000 $0000_{(2)}$= -2 147 483 648 (-2^{31})

- Observaţi că msb (b31) este:
  - ✓ **0** pentru numerele pozitive
  - ✓ **1** pentru numerele negative

  De aceea msb se numeşte *bit de semn*

Reprezentarea în complement faţă de 2

- *Pas 1:* Se reprezintă numărul (considerat fără semn) pe 31 de biţi şi se setează msb = 0 (dacă numărul este pozitiv) sau msb = 1 (dacă numărul este negativ)

- *Pas 2:* Dacă msb = 1, atunci se complementează toţi biţii cu excepţia bitului de semn

- *Pas 3:* Dacă msb = 1, atunci se adaugă 1

Atenţie! Paşii 2 şi 3 se aplică numai pentru numerele negative

Reprezentarea în complement faţă de 2

Exemplificăm reprezentarea în complement faţă de 2 a lui -1:

- *Pas 1:* Se reprezintă numărul (considerat fără semn) pe 31 de biţi şi se setează msb = 0 (dacă numărul este pozitiv) sau msb = 1 (dacă numărul este negativ)

  1000 0000 0000 0000 0000 0000 0000 0001

- *Pas 2:* Dacă msb = 1, atunci se complementează toţi biţii cu excepţia bitului de semn

  1111 1111 1111 1111 1111 1111 1111 1110

- *Pas 3:* Dacă msb = 1, atunci se adaugă 1

  1111 1111 1111 1111 1111 1111 1111 1111

Reprezentarea în complement faţă de 2

- De ce se foloseşte reprezentarea în complement faţă de 2?

  - ✓ Nu se foloseşte doar bitul de semn ca să se evite reprezentarea duplicată a lui 0 ca +0 şi -0.

  - ✓ Pentru eficienţa la scădere, care se rezumă la adunare cu numărul negativ; exemplu:

    1 – 2 = 1 + (-2)

$$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001_{(2)} = 1$$

$$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1110_{(2)} = -2$$

$$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111_{(2)} = -1$$

Negarea unui număr

Pentru negarea unui număr (i.e. valoarea absolută se păstrează, semnul se schimbă) se execută următorii paşi:

- *Pas 1:* Se complementează toți biții

- *Pas 2:* Se adaugă 1.

Negarea unui număr

Pentru negarea lui 4 (în -4) se execută următorii paşi :

$$4 = \ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0100_{(2)}$$

- *Pas 1:* Se complementează toți biții

$$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1011$$

- *Pas 2:* Se adaugă 1.

$$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1100_{(2)} = -4$$

Utilizare: Adunarea şi scăderea

- Adunarea numerelor se face în mod natural, pe biți, cu transport:

$$
\begin{array}{r}
0111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1011\ + \\
0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001 \\
\hline
0111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1100
\end{array}
$$

- Dacă se foloseşte reprezentarea în complement față de 2, scăderea se reduce la adunare cu numărul negat (-):

$$10 - 4 = 6$$

$$
\begin{array}{r}
0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 1010\ + \\
1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1100 \\
\hline
0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0110
\end{array}
$$

Extensia numărului de biți

Pentru extensia numărului de biți din reprezentare:

- *Pas 1:* Se copiază bitul de semn (msb) pe noile poziții (cei mai semnificativi biți)

Extensia numărului de biți

Pentru extensia lui 4 reprezentat pe 16 biți în reprezentare pe 32 de biți:

$$4 = 0000\ 0000\ 0000\ 0100_{(2)}$$

- *Pas 1:* Se copiază bitul de semn (msb) pe noile poziții (cei mai semnificativi biți); în cazul de față msb = 0, iar biții adăugați sunt marcați cu albastru

$$4 = {\color{blue}0000\ 0000\ 0000\ 0000}\ 0000\ 0000\ 0000\ 0100_{(2)}$$

Extensia numărului de biți

Pentru extensia lui -4 reprezentat pe 16 biți în reprezentare pe 32 de biți:

$$-4 = \ 0000\ 0000\ 0000\ 0100_{(2)}$$

- *Pas 1:* Se copiază bitul de semn (msb) pe noile poziții (cei mai semnificativi biți); în cazul de față msb = 0, iar biții adăugați sunt marcați cu albastru

$$-4 = \ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1100_{(2)}$$

Utilizare: Instrucțiuni cu valoare imediată

- O *valoare imediată* este o valoare care se specifică direct în instrucțiune

- În exemplele de mai jos, 7 şi 0x2a sunt valori imediate reprezentate pe **16 biți** :

[vom analiza mai detaliat cât studiem formatul instrucțiunilor]

```
addi $t1,$t2,7
addi $t1,$t1,0x2a
```

Numerele din regiştrii sunt considerate **cu semn**

```
addiu $t1,$t2,7
addiu $t1,$t1,0x2a
```

Numerele din regiştrii sunt considerate **fără semn**

- Pentru a se putea efectua adunarea, valorile imediate sunt extinse de la 16 la 32 de biți (dimensiunea regiştrilor generali)

MIPS32

- Există posibilitatea ca rezultatul unei adunări sau scăderi să fie prea mare, respectiv prea mic pentru a putea fi reprezentat în registrul destinație
  - *Întrebare:* Care dintre următoarele secvențe de cod presupune depăşirea dimensiunii (*overflow*)?

```
li $t2,0x7fffffff
li $t3,0x00000001
add $t0,$t2,$t3
```

```
li $t2,0x7fffffff
li $t3,0x00000001
addu $t1,$t2,$t3
```

  - *Răspuns:* Prima secvență de instrucțiuni, pentru că suma `0x80000000` este considerată număr negativ într-o instrucțiune de tip `add` (dar număr pozitiv într-o instrucțiune de tip `addu`)

MIPS32

- În caz de overflow:
  - `add,addi,sub` produc excepții
  - `addu,addiu,subu` nu produc excepții, ci trunchiază rezultatul

- O *excepție* este un eveniment neaşteptat care întrerupe execuția programului

- Registrul special **EPC** conține adresa instrucțiunii care a generat excepția

[mai multe info la laborator]

Reprezentarea în virgulă mobilă

- Numerele reale se reprezintă cu virgulă:

$$\pi = 3.141592 \ldots$$

- Acelaşi număr poate fi scris în multiple forme, în funcție de puterile lui 10:

$$\pi = 0.3141592 \ldots \cdot 10 = 0.03141592 \ldots \cdot 10^2 = 31.41592 \ldots \cdot 10^{-1}$$

- De aici denumirea de *virgulă mobilă*

- *Notația ştiințifică (normalizată)* este cea cu un singur digit nenul în stânga virgulei

$$523000 = 5.23 \cdot 10^5 = 5.23e + 5$$
$$10^{1000} = 1 \cdot 10^{1000} = 1e + 1000$$
$$-0.523 = -5.23 \cdot 10^{-1} = -5e - 1$$

Reprezentarea în virgulă mobilă

- Pentru calculator, vom utiliza puterile lui 2, deci un număr în virgulă mobilă va avea forma:

$$1.f \cdot 2^e$$

- Terminologie:
  - ✓ ***f*** se numeşte *mantisă*
  - ✓ ***e*** se numeşte *exponent*

- Reprezentarea în virgulă mobilă pe 32 de biți (*IEEE 754*) foloseşte:
  - ✓ **1** bit pentru semn
  - ✓ **8** biți pentru exponent (e + 127)
  - ✓ **23** de biți pentru mantisă

[http://grouper.ieee.org/groups/75]

b31	b30	b29	...	b23	b22	b21	b20	b19	...	b1	b0
semn	exponent (e + 127)				mantisă						

Reprezentarea în virgulă mobilă

- Pentru reprezentarea în virgulă mobilă avem nevoie de reprezentarea numerelor reale (a mantisei) pe biți
  - ✓ **1** bit pentru semn
  - ✓ **8** biți pentru exponent (e + 127)
  - ✓ **23** de biți pentru mantisă

$$-0.75 = -\frac{3}{4} = -3 \cdot 2^{-2} = -0.11_{(2)} = -1.1 \cdot 2^{-1}$$

b31	b30	...	b23	b22	...	b0
1	01111110			10000000000000000000000		

Reprezentarea în virgulă mobilă în dublă precizie

- Reprezentarea în virgulă mobilă în dublă precizie pe 64 de biți (*IEEE 754*) foloseşte:
  - ✓ **1** bit pentru semn
  - ✓ **11** biți pentru exponent (e + 1023)
  - ✓ **20 +32** de biți pentru mantisă

b31	b30	b29	...	b23	b22	b21	b20	b19	...	b1	b0
semn	exponent (e+1023)							mantisă			

b31	b30	b29	...	b23	b22	b21	b20	b19	...	b1	b0
mantisă											

Reprezentarea în virgulă mobilă în dublă precizie

- Reprezentarea în virgulă mobilă în dublă precizie pe 64 de biți folosește:
  - ✓ **1** bit pentru semn
  - ✓ **11** biți pentru exponent (e + 1023)
  - ✓ **20 +32** de biți pentru mantisă

$$-0.75 = -\frac{3}{4} = -3 \cdot 2^{-2} = -0.11_{(2)} = -1.1 \cdot 2^{-1}$$

b31	b30	...	b20	b19	...	b0
1	01111111110			10000000000000000000		

b31	...	b0
00000000000000000000000000000000		

Reprezentarea numerelor reale

Algoritm: Algoritmul presupune conversiile separată a părţii întregi şi a celei fracţionare, urmate de combinarea rezultatelor.

Partea întreagă se converteşte în baza b conform numerelor întregi. Pentru convertirea părţii fracţionare se foloseşte următoarea observaţie:

Fie $x = 0, y_1 y_2 y_3 \ldots_{(b)} = \frac{y1}{b} + \frac{y2}{b^2} + \frac{y3}{b^3} + \ldots$

Atunci $x \cdot b = y_1 + \frac{y2}{b} + \frac{y3}{b^2} + \ldots$
Deci: $[x \cdot b] = y_1$ şi $\{x \cdot b\} = \frac{y2}{b} + \frac{y3}{b^2} + \ldots$

Fie $x_1 = \frac{y2}{b} + \frac{y3}{b^2} + \ldots$
Atunci $x_1 \cdot b = y_2 + \frac{y3}{b} + \ldots$
Deci: $[x_1 \cdot b] = y_2$ şi $\{x_1 \cdot b\} = \frac{y3}{b} + \ldots$

Se continuă raţionamentul până când partea fracţionară devine 0 sau până când se obţine numărul de zecimale dorit.

Reprezentarea în virgulă mobilă

Ex.: Reprezentarea în simplă precizie a lui $N = 228,15$
$228_{(10)} = 128 + 64 + 32 + 4 = 11100100_{(2)}$.

$0,15_{(10)} =$
$0,15 \cdot 2 = 0,3 \Rightarrow [0,3] = 0$ şi $\{0,3\} = 0,3$;
$0,3 \cdot 2 = 0,6 \Rightarrow [0,6] = 0$ şi $\{0,6\} = 0,6$;
$0,6 \cdot 2 = 1,2 \Rightarrow [1,2] = 1$ şi $\{1,2\} = 0,2$;
$0,2 \cdot 2 = 0,4 \Rightarrow [0,4] = 0$ şi $\{0,4\} = 0,4$;
$0,4 \cdot 2 = 0,8 \Rightarrow [0,8] = 0$ şi $\{0,8\} = 0,8$;
$0,8 \cdot 2 = 1,6 \Rightarrow [1,6] = 1$ şi $\{1,6\} = 0,6$;

STOP fiindcă se repetă partea fracţionară.

Deci, $N = 228,15_{(10)} = 11100100,001001\,1001\,1001\ldots_{(2)}$
$= 1,11001000010011001\ldots \cdot 2^7$

Atunci, $C = 127 + e = 127 + 7 = 134 + (10) = 128 + 4 + 2 = 10000110_{(2)}$

Reprezentarea lui N=228,15 în simplua precizie devine:

0	10000110	11001000010011001...
S	C = e+127	f

Reprezentarea în virgulă mobilă în dublă precizie

Ex.: Reprezentarea în dublă precizie a lui $N = -228,15$

Conform ex. precedent:
$|N| = 11100100,001001\,10011001\ldots_{(2)} = 1,1100100001001\,1001\,1001\ldots \cdot 2^7$

Atunci, $C = 1023{+}e = 1023{+}7 = 1030_{(10)} = 1024{+}4{+}2 = 10000000110_{(2)}$

Reprezentarea lui N=228,15 în dubla precizie devine:

1	10000000110	11001000010011001 1001...
S	C = e+1023	f

Tipuri de date în C

- Limitele pentru tipurile de date din C sunt conforme cu reprezentarea în virgulă mobilă:

Tip de date	Bytes	Ordin de mărime
float	4	$2^{\pm 128} = 3.4e \pm 38$
double	8	$2^{1024} = 1.7e \pm 308$

Reprezentarea în virgulă mobilă

- Se pot întâlni 2 tipuri de excepții:

  - *Overflow* : exponentul este prea mare pentru a fi reprezentat pe 8 biți (când exponentul este pozitiv, deci numărul care se reprezintă este foarte mic)

  - *Underflow* : exponentul este prea mic pentru a fi reprezentat pe 8 biți (când exponentul este negativ, deci numărul care se reprezintă este foarte mic)

Reprezentarea în virgulă mobilă

- Valori speciale:

NaN = Not a Number

QNaN = Quiet NaN

SNaN = Signalling NaN

(semnalizează o excepţie)

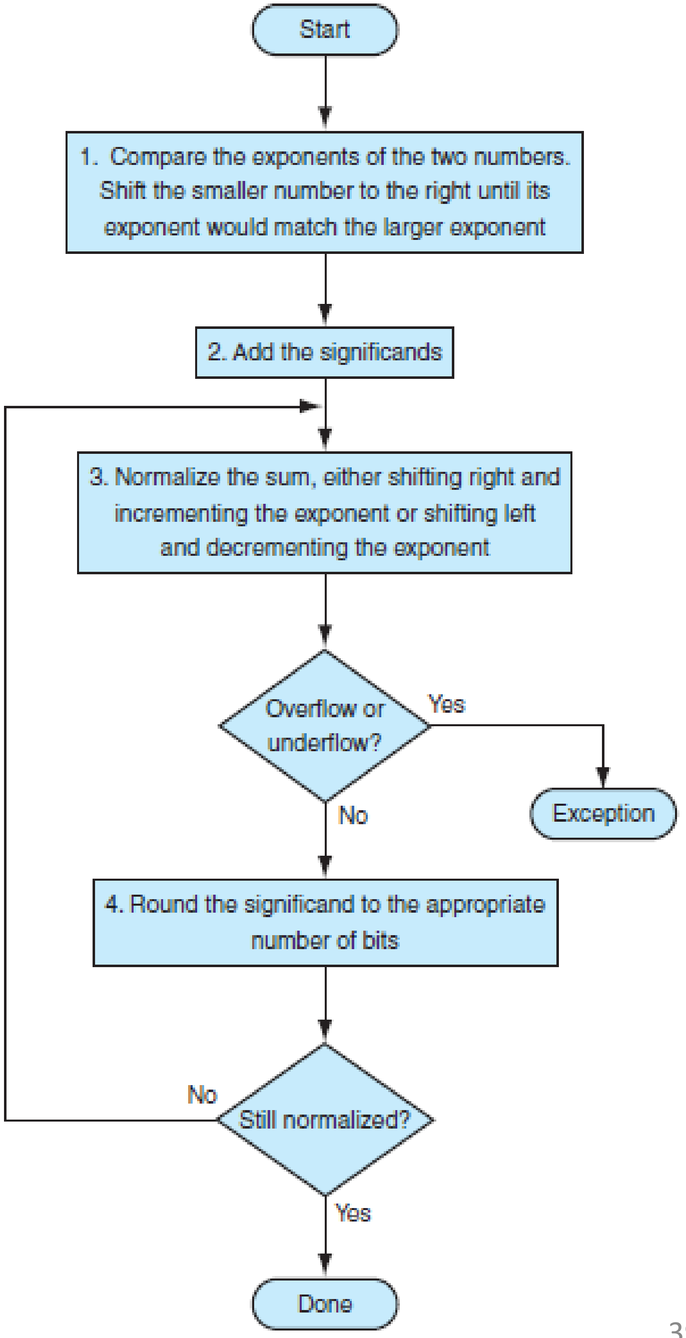
semn	exponent	mantisa	valoare
0	00…00	00…00	+0
1	00…00	00…00	-0
0	11…11	00…00	+Infinit
1	11…11	00…00	-Infinit
0	11⋯11	00⋯01 ⋮ 01⋯11	SNaN
1	11⋯11	00⋯01 ⋮ 01⋯11	SNaN
0	11⋯11	10⋯00 ⋮ 11⋯11	QNaN
1	11⋯11	10⋯00 ⋮ 11.11	QNaN

Reprezentarea în virgulă mobilă

- Operaţii speciale:

Operatie	Rezultat
n ÷ ±Infinit	0
±Infinit × ±Infinit	±Infinit
±nonzero ÷ 0	±Infinit
Infinit + Infinit	Infinit
±0 ÷ ±0	NaN
Infinit - Infinit	NaN
±Infinit ÷ ±Infinit	NaN
±Infinit × 0	NaN

Adunarea în virgulă mobilă



[D. Patterson and J. Hennessy, Computer Organisation and Design]

Adunarea în virgulă mobilă

- *Întrebare:* Adunaţi în virgulă mobilă numerele 0.5 şi -0.4375
- *Răspuns:* Se reprezintă în virgulă mobilă cele 2 numere:

$$0.5 = \frac{1}{2} = 2^{-1} = 2^{-1} = 1 \cdot 2^{-1}{}_{(2)}$$

$$-0.4375 = -\frac{7}{16} = 7 \cdot 2^{-4} = -0.0111_{(2)} = -1.11 \cdot 2^{-2}{}_{(2)}$$

✓ *Pas 1:* Se aduc ambii operanzi la exponentul maxim (-1)

$$1 \cdot 2^{-1}{}_{(2)}$$

$$-1.11 \cdot 2^{-2}{}_{(2)} = -0.111 \cdot 2^{-1}{}_{(2)}$$

✓ *Pas 2:* Se adună mantisele

$$1 \cdot 2^{-1}{}_{(2)} + (-1.11 \cdot 2^{-2})_{(2)} = (1 - 0.111) \cdot 2^{-1}{}_{(2)} = 0.001 \cdot 2^{-1}{}_{(2)}$$

Adunarea în virgulă mobilă

- *Pas 3:* Se normalizează

$$0.001 \cdot 2^{-1}{}_{(2)} = 1 \cdot 2^{-4}{}_{(2)}$$

- *Decizie:* 127+(-4) = 123 se poate reprezenta

b31	b30	...	b23	b22	...	b0
0	01111011			00000000000000000000000		

- *Verificare:*

$$1 \cdot 2^{-4}{}_{(2)} = \frac{1}{16} = 0.0625$$

$$0.5 - 0.4375 = 0.0625$$