Arhitectura Calculatoarelor

Curs 5: Proiectarea Unității Aritmetice-Logice (UAL / ALU) pentru MIPS

E-mail: florin.oniga@cs.utcluj.ro

Web: http://users.utcluj.ro/~onigaf, secțiunea Teaching/AC

Reprezentarea numerelor binare

Semn / Magnitudine	Complement față de 1	Complement față de 2
000 = +0	000 = +0	000 = +0
001 = +1	001 = +1	001 = +1
010 = +2	010 = +2	010 = +2
011 = +3	011 = +3	011 = +3
100 = -0	100 = -3	100 = -4
101 = -1	101 = -2	101 = -3
110 = -2	110 = -1	110 = -2
111 = -3	111 = -0	111 = -1

- Avantajul folosirii complementului față de 2:
  - Scăderea poate folosi aceeași logică binară ca adunarea
  - Bitul de semn poate fi tratat ca un număr normal în adunare
- Dezavantajul complementului față de 1: două reprezentări pentru zero
- Dacă X este negativ, reprezentarea în C2 pe n biți este echivalentă cu reprezentarea binară a numărului 2^n - |X|

Reprezentarea numerelor binare - MIPS

Numere binare întregi fără semn

- Tipic, reprezintă adrese sau alte valori care nu pot fi negative
- Valoarea zecimală a unui număr binar $b_{n-1}b_{n-2}\ldots b_1b_0$ fără semn

$$\text{value} = \sum_{i=0}^{n-1} b_i 2^i$$

- Un număr binar întreg fără semn de n biţi acoperă domeniul de la 0 la $2^n - 1$.

Numere binare întregi cu semn

- Tipic, se folosesc pentru reprezentarea datelor care pot fi pozitive sau negative
- Reprezentarea cea mai comuna → complementul fata de 2
- Valoarea unui număr cu semn (reprezentat in complement față de 2)

$$\text{value} = -b_{n-1}2^{n-1} + \sum_{i=0}^{n-2} b_i 2^i$$

- Un număr binar întreg de n biţi reprezentat în complement față de 2 acoperă domeniul de la -2^{n-1} la $2^{n-1} - 1$.

Reprezentarea numerelor binare - MIPS

Numere binare întregi cu semn, 32 de biți

$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000_{two} = 0_{ten}$

$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001_{two} = +\ 1_{ten}$

$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0010_{two} = +\ 2_{ten}$

...

$0111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1110_{two} = +\ 2{,}147{,}483{,}646_{ten}$

$0111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111_{two} = +\ 2{,}147{,}483{,}647_{ten}$ →maxint

$1000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000_{two} = -\ 2{,}147{,}483{,}648_{ten}$ →minint

$1000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001_{two} = -\ 2{,}147{,}483{,}647_{ten}$

$1000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0010_{two} = -\ 2{,}147{,}483{,}646_{ten}$

...

$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1101_{two} = -\ 3_{ten}$

$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1110_{two} = -\ 2_{ten}$

$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111_{two} = -\ 1_{ten}$

Operații cu numere binare - MIPS

Negativul în complementul față de 2

- Inversează toţi biţii si aduna 1 la numărul negat

  Ex. Cât este negativul pentru: 6 = 0110 => negativul este -6= (not 0110 + 0001) = 1010, negativul lui -6 este (not 1010 + 0001) = 0110 = 6

Adunarea în complementul față de 2

- Se aduna biţii corespunzători ai celor două numere și transportul de la rangul anterior

3 = 0011	-3 = 1101	-3 = 1101	3 = 0011
+ 2 = 0010	+-2 = 1110	+ 2 = 0010	+ -2 = 1110
- - - -	- - - -	- - - -	- - - -

- Adunările numerelor fără semn și a celor reprezentate în complement față de 2 se execută exact la fel, diferă doar modul de detectare a depăşirii

Scădere în complementul față de 2

- Se obține negativul scăzătorul, apoi sunt adunate cele două numere

3-	3 = 0011	3+	3 = 0011	-3 -	-3 = 1101	-3 +	-3 = 1101
2	2 = 0010	(-2)	-2 = 1110	-2	-2 = 1110	2	2 = 0010
?	- - - -	1	1= 0001	?	- - - -	-1	-1 = 1111

Operații cu numere binare - MIPS

Depășirea în complementul față de 2

- Suma sau diferenţa poate depăși domeniul numerelor reprezentabile pe biți disponibili
- Depăşire: rezultatul este prea mare (sau prea mic) pentru a fi reprezentat în mod corect.

  Ex. depășiri pe 4 biți, complement față de 2 (interval reprezentabil -8 .. 7)

5 = 0101	-5 = 1011	+5 = 0101	-5 = 1011
+ 6 = 0110	+-6 = 1010	-- 6 = 1010	- +6 = 0110
-5 =1011	5 = 0101	-5 = 1011	5 = 0101

- Depășirea creează un rezultat eronat care ar trebui detectat
- Depăşirea are loc când valoarea rezultată afectează semnul:
  - 2 numere pozitive și suma este negativă
  - 2 numere negative și suma este pozitivă

Operație	A	B	Rezultatul indică depășire
A + B	>= 0	>= 0	< 0
A + B	< 0	< 0	>= 0
A – B	>= 0	< 0	< 0
A – B	< 0	>= 0	>= 0

- Nu apare depășire la adunare când semnele sunt diferite
- Nu apare depășire la scădere când semnele sunt la fel.

Operații cu numere binare - MIPS

Complementul față de 2 – Detectarea depăşirii

➢ Când se adună numere reprezentate în complement față de 2, depășirea va avea loc numai dacă:

- numerele de adunat au același semn (de ce?)
- semnul rezultatului diferă de semnul operanzilor

$$overflow = a_{n-1} \cdot b_{n-1} \cdot \overline{s_{n-1}} + \overline{a_{n-1}} \cdot \overline{b_{n-1}} \cdot s_{n-1}$$

$a_{n-1}\ a_{n-2} \ldots a_1\ a_0 +$
$b_{n-1}\ b_{n-2} \ldots b_1\ b_0$

$= s_{n-1}\ s_{n-2} \ldots\ s_1\ s_0$

➢ Dacă C_{n-1} si C_n reprezintă transporturile de intrare, respectiv de ieșire ale bitului de semn (din adunarea $A_{n-1}+B_{n-1}+C_{n-1}$ rezultă S_{n-1} și C_n):

Operanzi	Rezultat	C_n	S_{n-1}	A_{n-1}	B_{n-1}	C_{n-1}	Eveniment ?
Pozitivi	Pozitiv	0	0	0	0	0	$C_n=C_{n-1}$ ⇔ fără depășire
	Negativ	0	1	0	0	1	$C_n!=C_{n-1}$⇔ depășire
Negativi	Pozitiv	1	0	1	1	0	$C_n!=C_{n-1}$⇔ depășire
	Negativ	1	1	1	1	1	$C_n=C_{n-1}$ ⇔ fără depășire

➢ Depășirea înseamnă → C_n != C_{n-1}

➢ Detectarea de depăşire → *overflow = CarryOut MSB XOR CarryIn MSB*

$$overflow = c_n \otimes c_{n-1}$$

Operații cu numere binare - MIPS

Aritmetica numerelor fără semn - Detectare de depășire

- Numere fără semn, Depăşire → transport de ieșire din bitul cel mai semnificativ

$$overflow = c_n$$

➢ De exemplu:

1001+	= 9+
1000	= 8
---	---
=0001	= 1

$c_n = 1$

➢ În arhitectura MIPS (completă!)

- Excepţii de depăşire sunt semnalate pentru aritmetica în complement față de 2
  - add, sub, addi
- Excepţii de depăşire nu sunt semnalate pentru aritmetica fără semn
  - addu, subu, addiu

Proiectare ALU

Procesul de proiectare

- Proiectarea începe cu cerințe funcționale și de performanță
- Proiectarea se finalizează prin asamblare
- Proiectarea înțeleasă în termenii de componente și mod de asamblare:
  - Descompunerea Top-Down a funcțiilor complexe (pe bază de comportament) de sus în jos în blocuri primitive
  - Compunerea Bottom-Up a blocurilor primitive de bază de jos în sus în ansambluri mai complexe

Proiectare ALU – MIPS, Cerințe

ALU trebuie sa efectueze un subset de instrucţiuni aritmetice-logice sau să ofere suport pentru alte instrucțiuni (explicit beq, implicit lw, sw):

Tip	opcode	function
addi	001000	xxxxxx
ori	001101	xxxxxx
lw	100011	xxxxxx
sw	101011	xxxxxx
beq	000100	xxxxxx

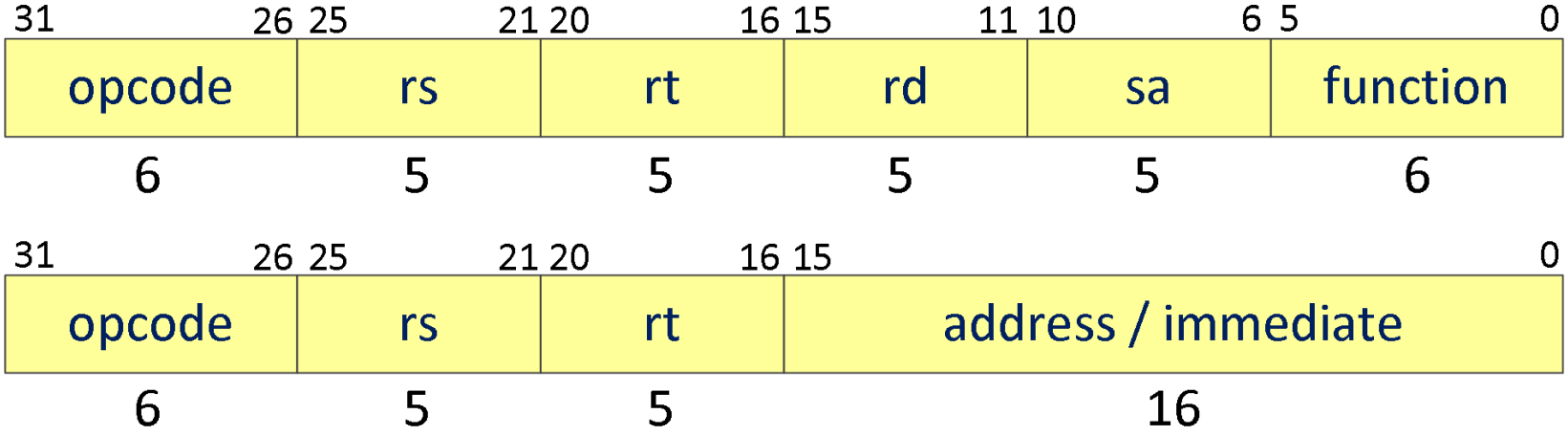
Tip	opcode	function
add	000000	100000
sub	000000	100010
and	000000	100100
or	000000	100101

Proiectare ALU - MIPS, Cerințe

Cerințele ALU MIPS pentru un subset limitat de instrucţiuni

- add, sub, addi, lw, sw → sumator / scăzator cu aritmetică în complement față de 2, cu detecţie de depăşire (Aritmetica cu semn generează depăşire → detecţie, „ignore” la laborator!)
- and, or, ori → operații logice AND, OR
- beq → Detecţie de zero (prin scădere!)
- j → fără

MIPS: Formatul instrucțiunilor aritmetice-logice



Proiectare ALU - MIPS

Abordarea 1: divide et impera, principii

- Descompunerea problemei in sub-probleme simple,
- Găsirea de soluții pentru sub-probleme și asamblarea soluției finale
- Exemplu:
  - Extinderea cu semn/zero a datei imediate înainte de ALU
  - Nu sunt necesare operații specifice în ALU pentru datele imediate; addi se execută la fel ca add …

Cerințe rafinate (Specificații funcționale)

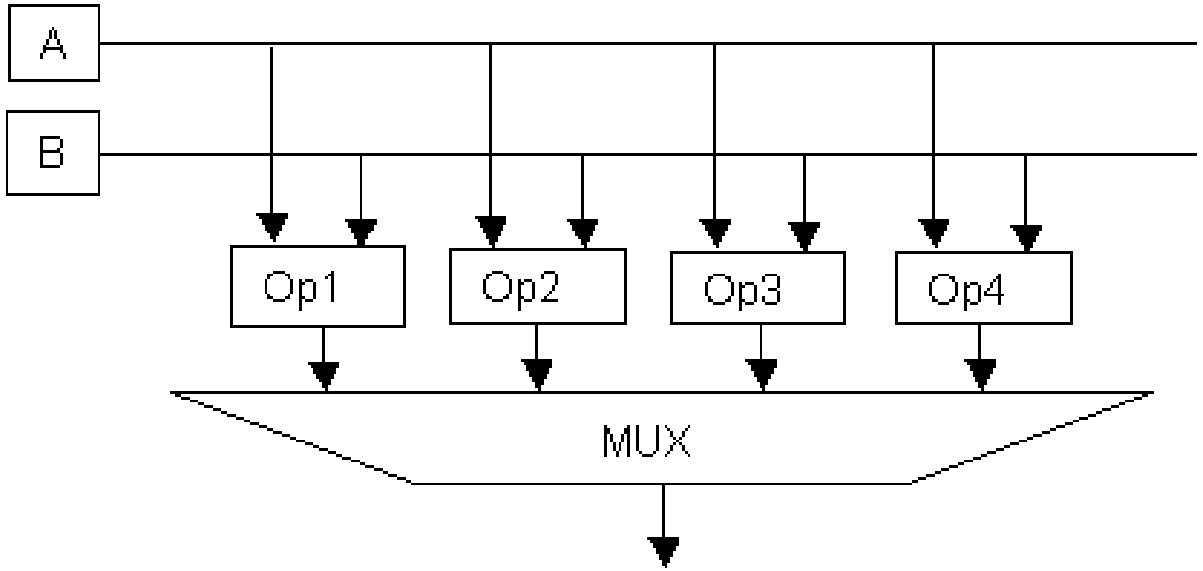
- *Intrări*: 2 operanzi pe 32 biţi, A si B; cod de operaţie de x biţi
- *Ieșiri*: rezultat de 32 biţi; semn, transport, depășire – semnale de stare
- *Operații*: add, sub, and, or – cu posibilități de extindere

Proiectare ALU - MIPS

Abordarea 1: divide et impera

Pentru implementarea operațiilor

- Alegeți componentele logice digitale necesare (SI, SAU, Sumator,..)
- Conectați-le conform specificației și selectați operațiile cerute printr-un MUX



Proiectare ALU - MIPS

Abordarea 2: proiectare incrementală

- Rezolvați o parte a problemei și extindeți-o
- Începeți cu porți simple SI, SAU

…în continuare se începe **proiectarea ALU MIPS, folosind o abordare mixtă**:

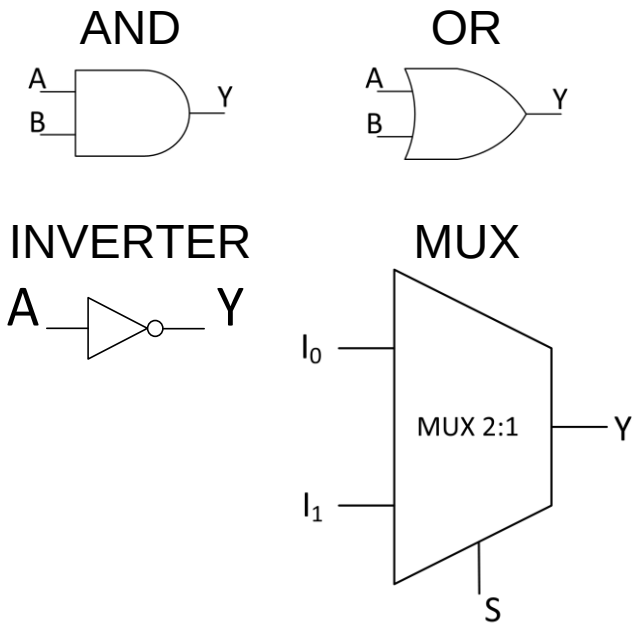
- Se împarte în **probleme mai simple**: construim o ALU pe 1 bit, pe urmă **conectăm 32 de ALU pe 1 bit**
- Construirea ALU pe 1 bit se face **incremental**, pornind de la porți simple

Proiectare ALU 1 bit

Se pleacă de la **blocurile constructive de bază**

- Porți SI, SAU, SAU-EXCLUSIV, etc.
- Inversori
- Multiplexoare

Începem cu **operațiile logice** → există componente hardware corespunzătoare (porți)



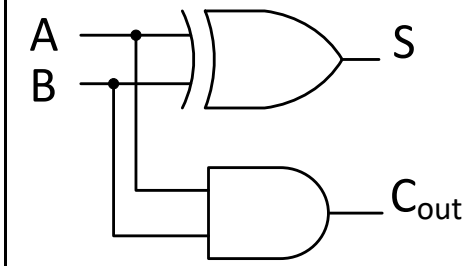
Proiectare ALU 1 bit

Următoarea funcție de inclus este adunarea – 1 bit

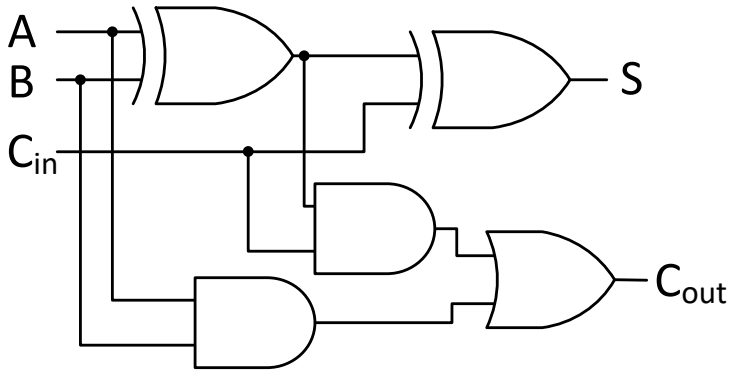
- Semi-sumator, Sumator (complet)

S_h = A xor B

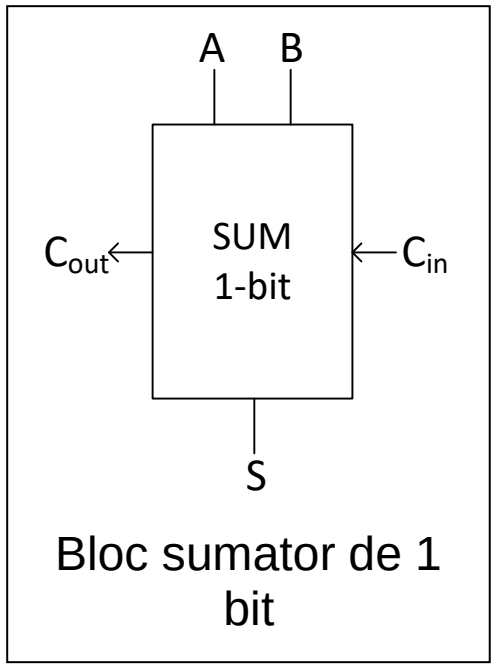
C_{out} = A and B



A	B	C_{out}	S_h
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0



A	B	S_h	C_{in}	C_{out}	S
0	0	0	0	0	0
0	0	0	1	0	1
0	1	1	0	0	1
0	1	1	1	1	0
1	0	1	0	0	1
1	0	1	1	1	0
1	1	0	0	1	0
1	1	0	1	1	1



Bloc sumator de 1 bit

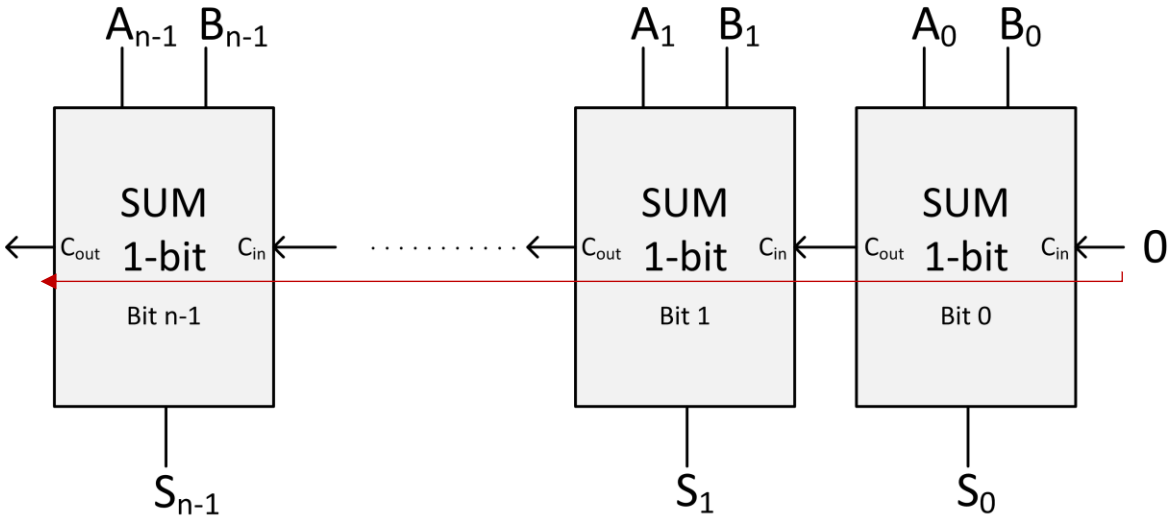
$S = S_h$ xor C_{in} = A xor B xor C_{in};

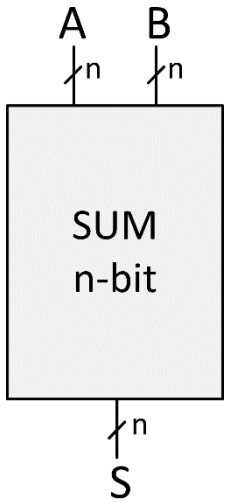
C_{out} = A and B or C_{in} and S_h = A and B or C_{in} and (A xor B)

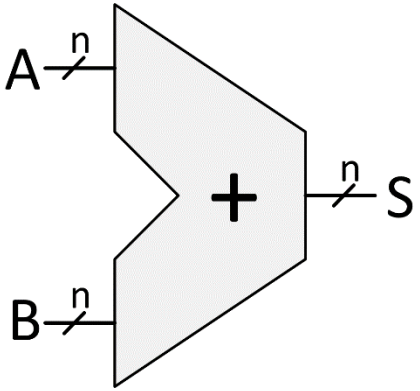
Proiectare ALU 1 bit

O mică deviere: Adunarea pe n biți?

- Înlănțuire (cale critică cu roșu!)
- Numerele de intrare pe *n* biți sunt $A_{n-1}…A_1A_0$ și $B_{n-1}…B_1B_0$, iar suma $S_{n-1}…S_1S_0$

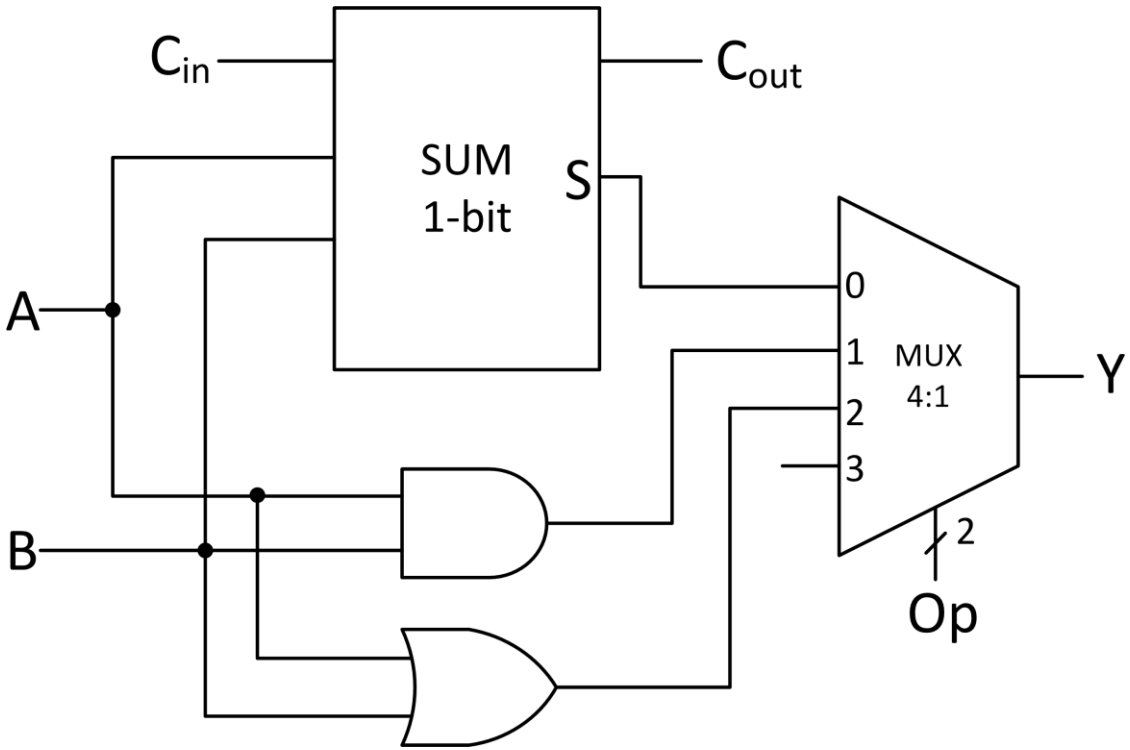






Proiectare ALU 1 bit

- 3 operații: sumator complet pe 1 bit, poarta ȘI și poarta SAU
- un multiplexor, Op va selecta ce rezultat să apară pe ieșirea ALU: 0 +, 1 ȘI, 2 SAU, 3 alte operații

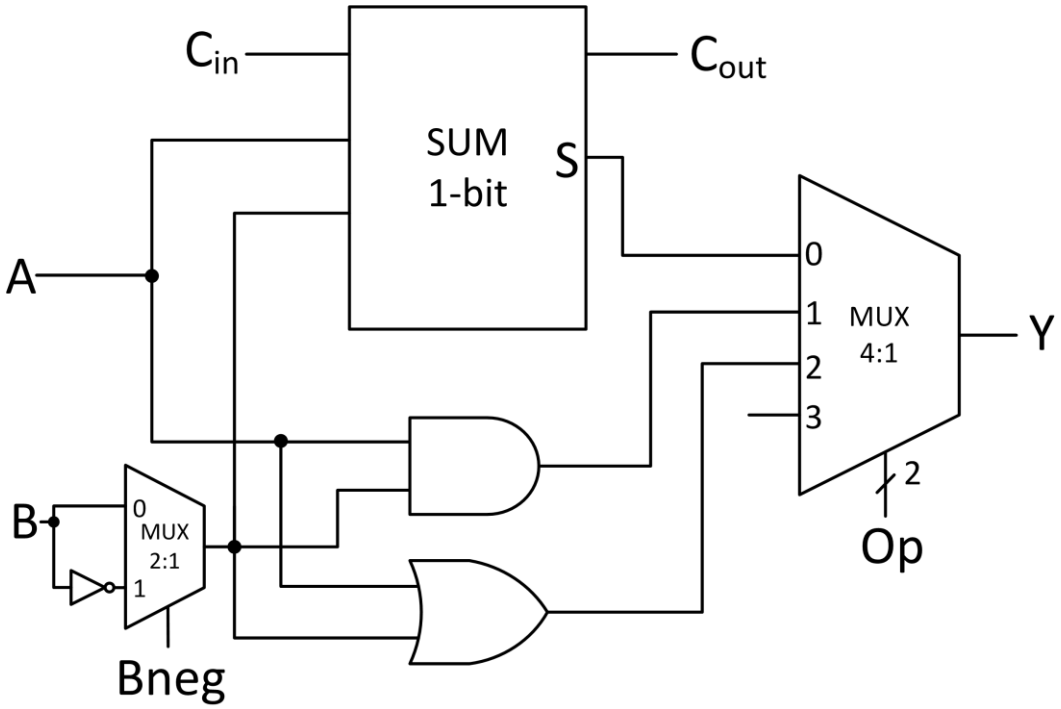


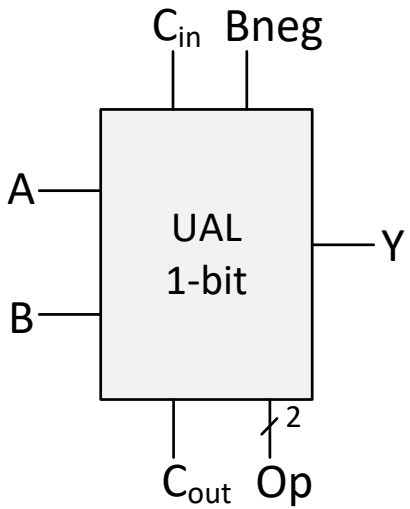
Proiectare ALU 1 bit => scădere

- **Scăderea** – se exploatează proprietățile C_2:

$$A - B = A + (-B) = A + (\bar{B} + 1) = A + \bar{B} + 1$$

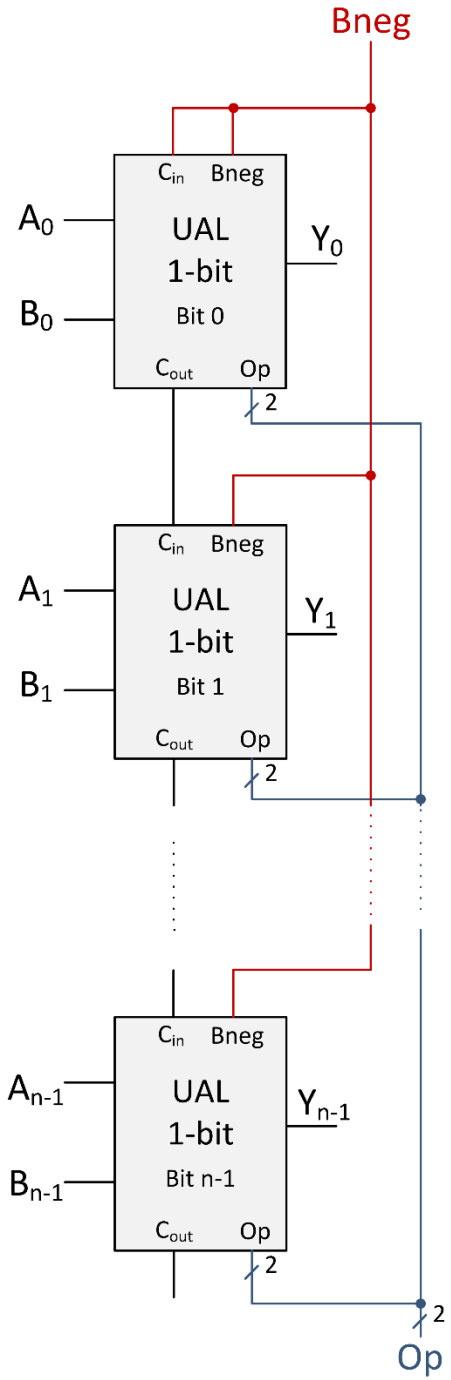
- Se extinde unitatea pe 1 bit cu un inversor + mux la B (selecție B_{neg}):





ALU 1 bit (+, -, or, and) detaliu și diagrama bloc

Proiectare ALU 1 bit => ALU 32 biți



1. Înlănțuire: C_{in} la C_{out}, Op și B_{neg} semnale comune

2. Pentru scădere se forțează "+1" pe bitul 0 (ALU_0)

   - B_{neg} este 1 doar la scădere, se leagă la C_{in} de la ALU_0!

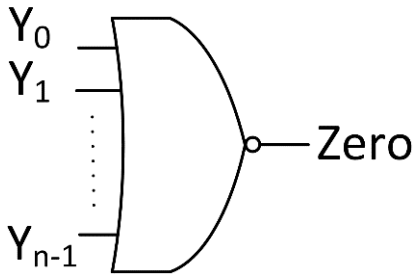
3. $AluCtrl_{2..0}$ este construit concatenând **B_{neg}** cu **Op**:

Operație UAL	$AluCtrl_{2...0}$	Observație
Adunare	0 0 0	B_{neg}=0, Op=0
Scădere	1 0 0	B_{neg}=1, Op=0
ȘI	0 0 1	B_{neg}=0, Op=1
SAU	0 1 0	B_{neg}=0, Op=2

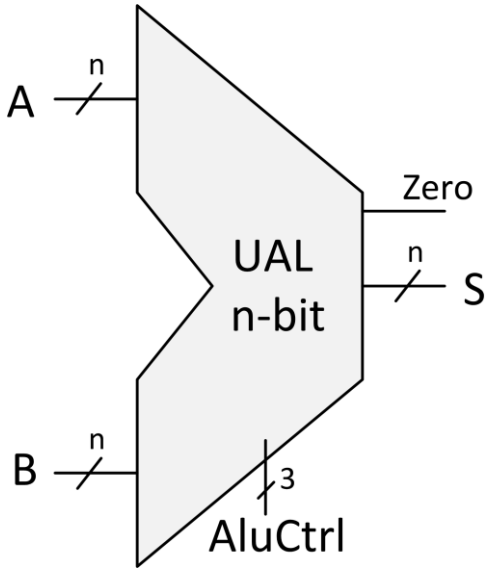
Proiectare ALU 32 biți

Pentru beq:

Detecția apariției simultane de 0 pe biții rezultatului se face cu o poartă NOR (SAU negat) care se leagă la ieșirile ALU (n=32):



Schema bloc ALU pe n biți (n=32), intrări A și B, ieșire (rezultat) S:



Proiectare ALU, alte funcții

Detecție de Depășire → *Overflow = CarryOut MSB xor CarryIn MSB*

➢ Se modifică ALU_{31}, unitatea pentru bitul cel mai semnificativ (MSB) cu integrarea unei porți XOR (nu se mai prezintă schema extinsă în curs)

Proiectare ALU, alte funcții

Comportamentul MIPS la Depășire, două metode:

1. **Ignorare** → MIPS: operații cu numere fără semn
   - Nu se detectează depășirea pentru
     - addu, addiu, subu
     - addiu deși se extinde cu semn!
     - sltu, sltiu pentru comparări fara semn

2. **Recunoaştere** → MIPS: operaţii pentru numere cu semn
   - Se generează o excepţie (întrerupere) care permite programatorului să trateze problema
     - Salt la o adresă definită pentru excepție
     - Adresa de întrerupere este salvată pentru posibilă reluare
     - Instrucțiunile MIPS: add, sub

Proiectare ALU (slide opțional)

Operații adiționale: Instrucțiunea „Detectează mai mic decât” (slt)

- Operaţia slt produce 1, pentru RF[rd], dacă RF[rs] < RF[rt], altfel 0
- slt setează toţi biţii la 0, cu excepţia LSB; LSB este setat conform comparării rs<rt

- Extindem MUX-ul de 3 intrări al ALU, adăugam o intrare noua: Less ← slt result

- Conectam 0 la intrările Less[31:1] = 0.
- Cum se obtine bitul Less[0]?
  - Se scade b din a. Daca diferenţa este negativa, atunci a < b.
  - Bitul 31 al rezultatului adunării/scăderii se notează ca Sign (Semn)
- SLTU (Unsigned); Pentru numere fără semn
  - Less[0] ← Set ← not CF; CF=0 la scădere fără semn ⇔ descăzutul este mai mic.
- SLT (Signed); Pentru numere cu semn
  - Less[0] ← Set ← Sign xor Overflow

Proiectare ALU – alte cerințe adiționale

Înmulțirea

- mult, multU → operații de înmulțire pe 32-biti, cu și fără semn
- Exemplu „hârtie și creion” (fără semn): 1000 (8) * 1001 (9) = 0100 1000 (72)

Deînmulţitul				1	0	0	0	
Înmulţitorul				1	0	0	1	
				1	0	0	0	
			0	0	0	0		
		0	0	0	0			
	1	0	0	0				
Produsul	0	1	0	0	1	0	0	0

m biţi x n biţi → m+n biţi produs

- **În binar este simplu**:
  - Înmulţitor bitul i = 0 → se plasează 0 (0 x deînmulțitul)
  - Înmulţitor bitul i = 1 → se plasează o copie a deînmulțitului (1 x deînmulțitul)

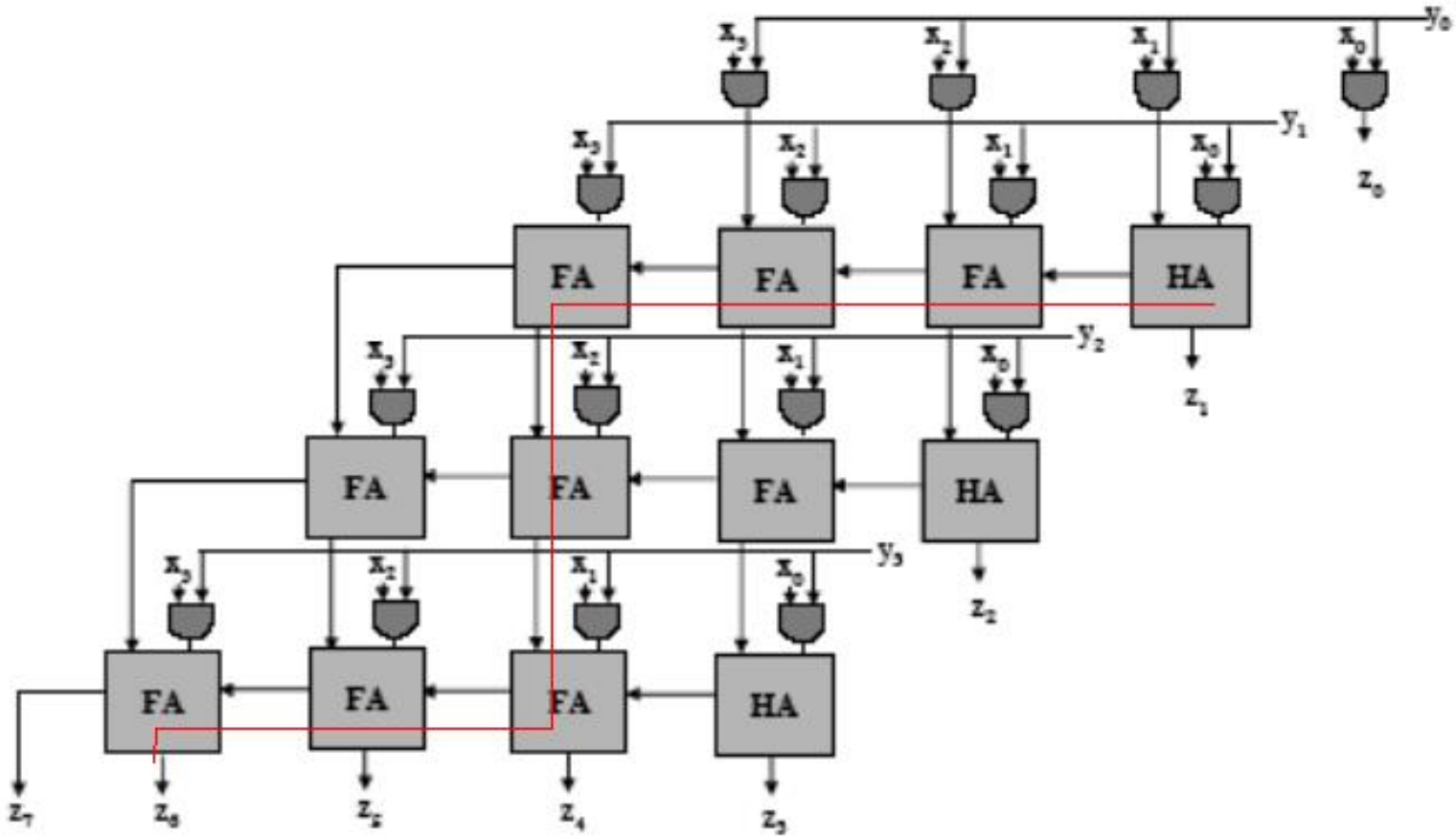
! Înmulțirea constă dintr-o serie de adunări și deplasări

Implementări de înmulţitoare:

- Combinaţional
- Pipelined (de tip banda de asamblare)
- Multi-ciclu, iterativ, cicluri repetate de adunare și deplasare

Proiectare ALU – alte cerințe adiționale

Înmulțitor combinaţional fără semn (Full/Half Adder – FA sau HA)



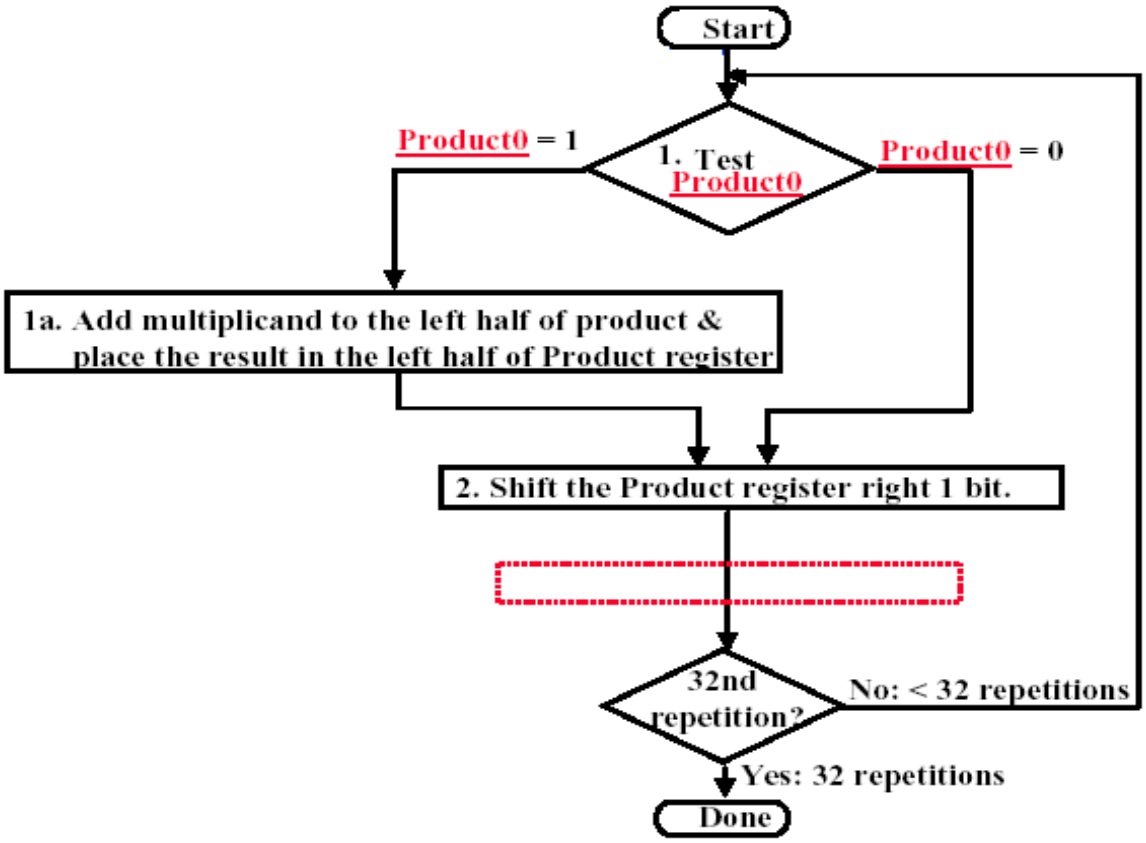
Înmulţitor Matricial cu propagarea transportului (Ripple Carry), $Z_{7..0}=X_{3..0} * Y_{3..0}$

- La fiecare etaj X (deînmulţitul) se deplasează la stânga
- Bitul următor al Y (înmulţitorul) determină dacă deînmulţitul deplasat sau zero trebuie adunat ca produs parțial; se acumulează produsul parțial
- Calea critica marcata cu rosu
- Versiunile de tip pipeline îmbunătățesc productivitatea înmulțitorului

Proiectare ALU – alte cerințe adiționale

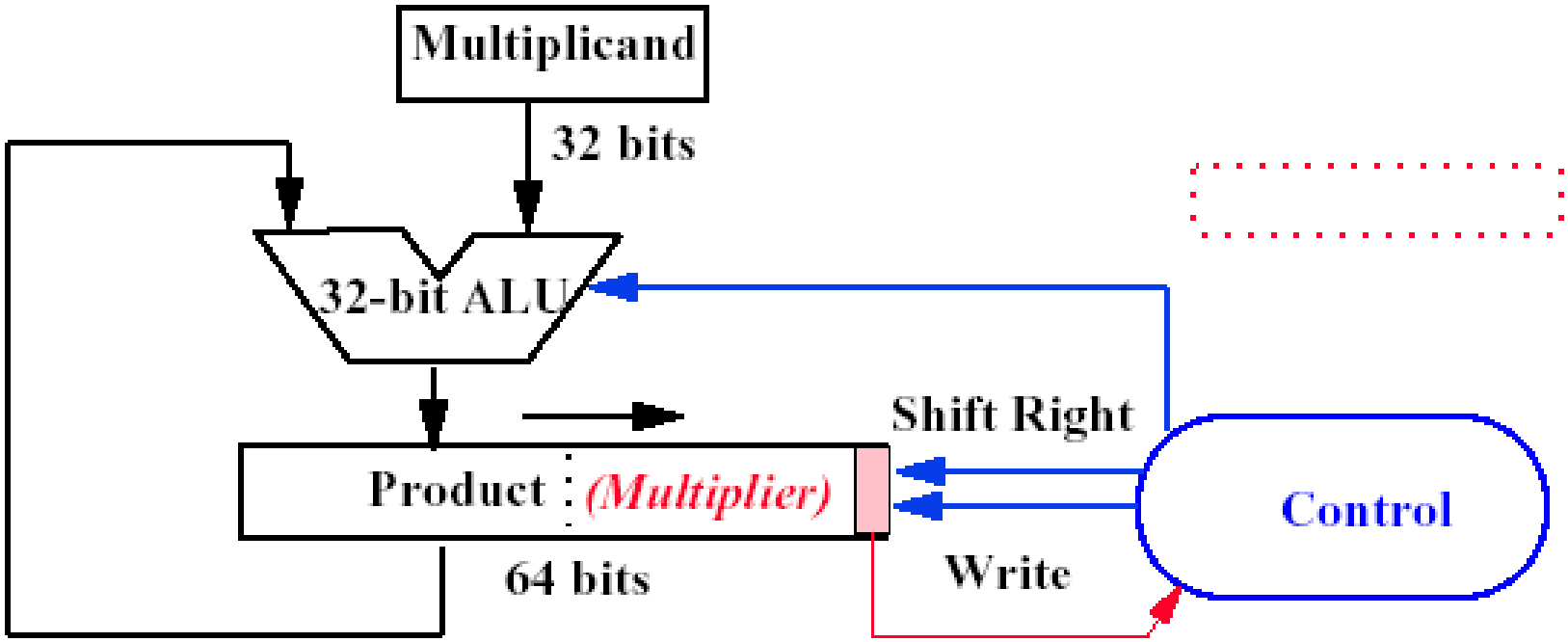
Algoritm de înmulțire prin adunări şi deplasări repetate (multi-ciclu)

- Implementează ad-literam algoritmul de înmulţire pentru numere binare (vezi exemplul „hârtie si creion”)
- Nu există registru separat pentru înmulţitor, se folosește un registru de deplasare pentru a stoca produsul (inclusiv sumele produselor parțiale)
- La inițializare, înmulțitorul se plasează în partea dreapta a registrului de produs de 64 biți, partea stângă se inițializează la 0
- Se repetă de N (nr. de biti) iterații, pentru însumarea produselor parțiale



Algoritmul de înmulţire prin adunări și deplasări repetate

Proiectare ALU – alte cerințe adiționale



Înmulţitor iterativ, căi de date; diagrama bloc

Observaţii asupra algoritmului de înmulţire

- 2 paşi pe bit fiindcă deplasarea se face cu registru de deplasare (Product & Multiplier)
- In MIPS registrele Hi si Lo conţin MSB (stânga) si LSB (dreapta) a produsului
- Instrucţiunile MIPS Mult, MultU: produs in HI și LO

Proiectare ALU – alte cerințe adiționale

Exemplu numeric pe 4 biți: 0010 * 0011

Iterație	Pas	Deînmulțit	Produs	Next
0	Init	0010	0000 0011	Product(0)='1' → add
1	1a	0010	0010 0011	Shift Right
	2		0001 0001	Product(0)='1' → add
2	1a		0011 0001	Shift Right
	2		0001 1000	Product(0)='0' → no add
3	1a		0001 1000	Shift Right
	2		0000 1100	Product(0)='0' → no add
4	1a		0000 1100	Shift Right
	2		0000 0110	Done

Cum efectuăm înmulţiri pentru numerele cu semn?

- Abordare simplă: transformă operanzii în numere pozitive, ține minte dacă produsul final trebuie schimbat ca semn (se renunţa la bitul de semn, iteraţii de 31 paşi)
- Sau: se aplică regulile aritmeticii complementului față de 2. Necesită extinderea semnului pentru „produsele parțiale" și scădere la sfârşit.
- **Versiuni avansate: Algoritmul lui Booth**

Proiectare ALU – alte cerințe adiționale

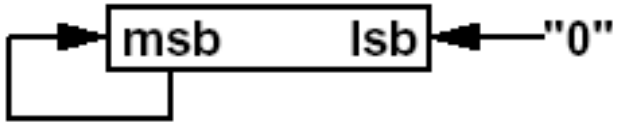
Cerinţe adiţionale: circuite de deplasare

- Sll, Srl, Sra → Necesita deplasare cu 0-31 biţi: logică la stânga/dreapta, aritmetică la dreapta

Două tipuri:

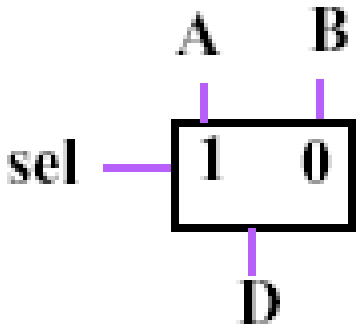
- *Logică* → valoarea care se introduce prin deplasare este întotdeauna "0"

- *Aritmetică* → deplasare la dreapta, se extinde semnul



Notă: Se efectuează o deplasare cu un singur bit pe ciclu. Instrucțiunile pot solicita deplasări multiple de 0 – 32 poziții!

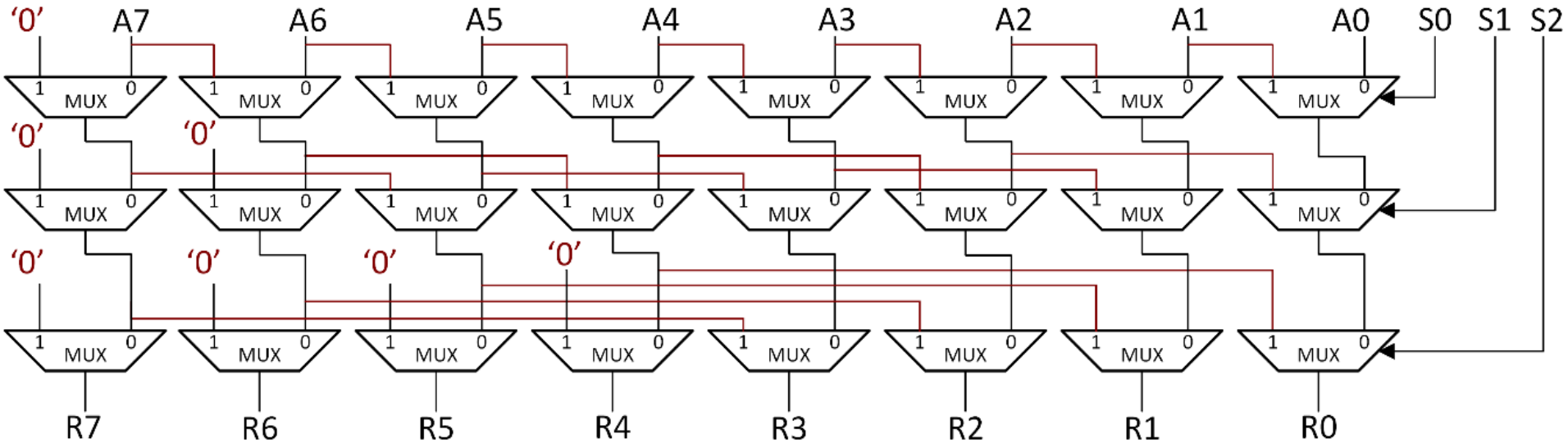
Circuit de deplasare combinaţional implementat cu MUX-uri



Blocul de baza - MUX

Proiectare ALU – alte cerințe adiționale

Circuit de deplasare de 8 biţi spre dreapta, cu MUX-uri



Exerciţii

Cu câte poziţii se face deplasarea la dreapta daca $S_2S_1S_0$=”001”? Dar pentru $S_2S_1S_0$=”011”?

Care este valoarea pentru $S_2S_1S_0$ pentru a efectua o deplasare cu 4 poziţii la dreapta?

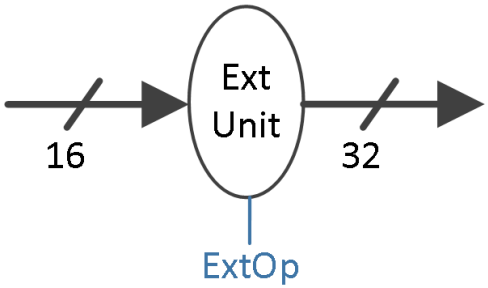
Cate niveluri sunt necesare pentru a efectua o deplasare cu 32 biţi?

Proiectare ALU – alte cerințe adiționale

Circuit pentru Extindere de Semn

Instrucțiunile MIPS folosesc operanzi cu:

- Extindere de Zero (logică) — ori: RF[rt] ← RF[rs] | Z_Ext(imm)
- Extindere de Semn (aritmetică) — lw: RF[rt] ← M[RF[rs] + S_Ext(imm)]



Extindere de Zero

- Circuitul de Extindere extinde (cu zerouri) un număr de 16 biţi (imm [15:0]) de la intrare, la 32 biţi de la ieşire, daca semnalul de comandă ExtOp este 0.
- Z_Ext(imm16) = 0_{31}…..0_{16}||imm_{15}….imm_0

Extindere cu Semn

- Circuitul de Extindere primeşte un număr de 16 biţi, imm [15:0] si îl extinde cu bitul imm_{15}, daca semnalul de comandă ExtOp este 1
  - S_Ext(imm16) = 0_{31}…..0_{16}||imm_{15}….imm_0 dacă imm_{15} = 0
  - S_Ext(imm16) = 1_{31}…..1_{16}||imm_{15}….imm_0 daca imm_{15} = 1

Probleme

1. Proiectati un ALU de 8 biti pentru urmatoarele operatii: A + B, A – B, IncrA, DecrA, PassA and NegateA. Folositi un singur sumator. Prezentati schema cu semnalele de control si un tabel cu valorile semnalelor de control pentru fiecare operatie.
2. Proiectati o unitate pentru Adunare/Scadere care poate lucra cu date de 8, 16 sau 32 de biti. Folositi 32 de sumatoare de 1 bit si circuite auxiliare. Prezentati schema block si semnalele de control pentru operatii de adunare/scadere pentru 4x8, 2x16, 1x32-bit Adunare/Scadere.
3. Prezentati algoritmul de inmultire cu adunări și deplasări repetate. Dati un exemplu numeric.

Bibliografie

1. D. A. Patterson, J. L. Hennessy, “Computer Organization and Design: The Hardware/Software Interface”, 5th edition, ed. Morgan–Kaufmann, 2013.
2. D. A. Patterson and J. L. Hennessy, “Computer Organization and Design: A Quantitative Approach”, 5th edition, ed. Morgan-Kaufmann, 2011.
3. MIPS32™ Architecture for Programmers, Volume I: “Introduction to the MIPS32™ Architecture”.
4. MIPS32™ Architecture for Programmers Volume II: “The MIPS32™ Instruction Set”.
5. Materialul obligatoriu (vezi pagina cu cursul)