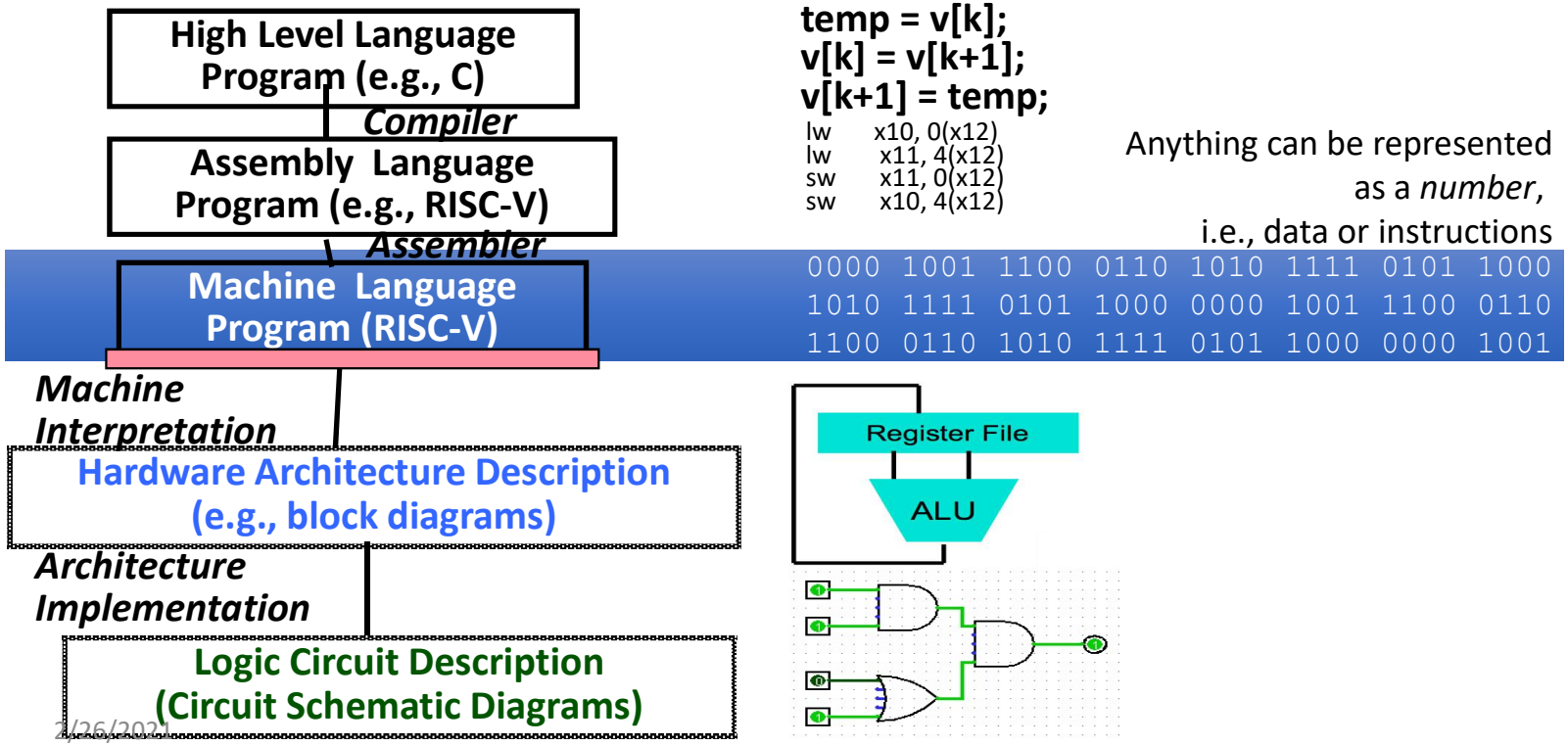
TP 2

Format d’instruccions

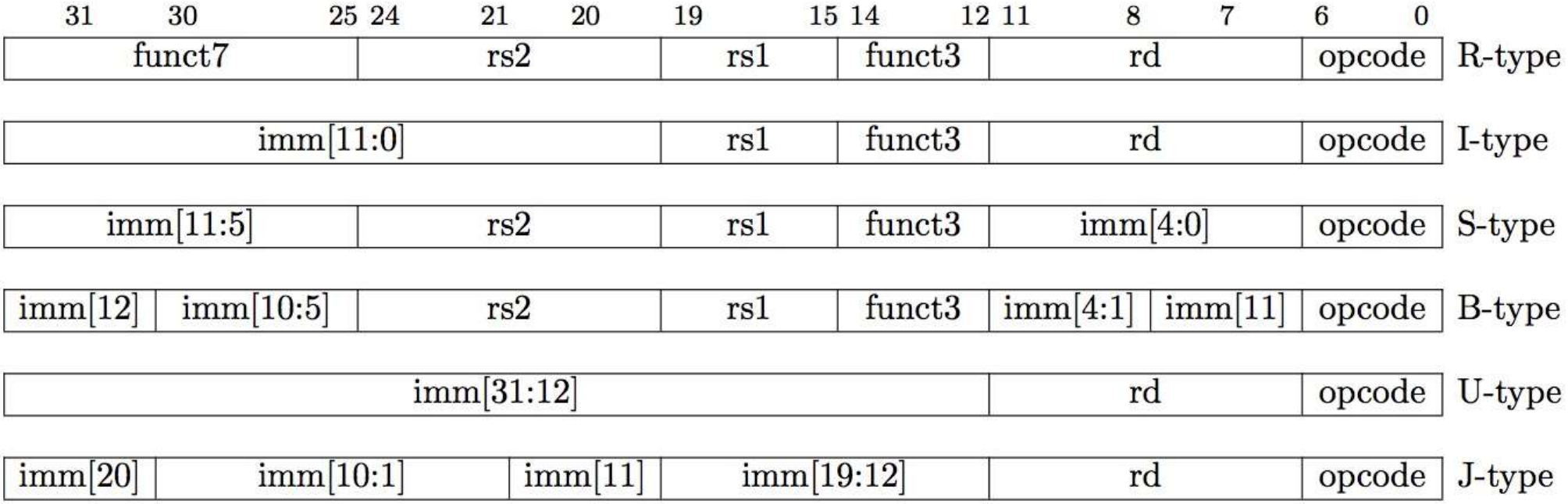
Nivells



Anything can be represented as a *number*, i.e., data or instructions

Format de les instruccions

- formats:
  - R-format for register-register arithmetic operations
  - I-format for register-immediate arithmetic operations and loads
  - S-format for stores
  - B-format for branches (minor variant of S-format, called SB before)
  - U-format for 20-bit upper immediate instructions
  - J-format for jumps (minor variant of U-format, called UJ before)


31 30 25 24 21 20 19 15 14 12 11 8 7 6 0
funct7 rs2 rs1 funct3 rd opcode R-type
imm[11:0] rs1 funct3 rd opcode I-type
imm[11:5] rs2 rs1 funct3 imm[4:0] opcode S-type
imm[12] imm[10:5] rs2 rs1 funct3 imm[4:1] imm[11] opcode B-type
imm[31:12] rd opcode U-type
imm[20] imm[10:1] imm[11] imm[19:12] rd opcode J-type

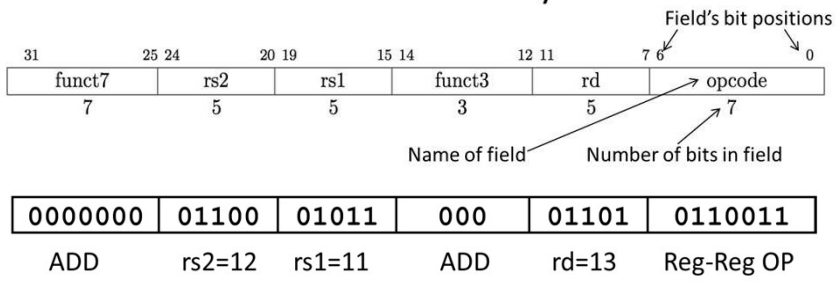
Tipus d'instruccions

- Operacions amb registres:
  - ➢Format: Operació Registre destí, Registre font1, Registre font 2

  Exemple:

  ADD a3, a2, a1

R-Format Instruction Layout



0000000	01100	01011	000	01101	0110011
ADD	rs2=12	rs1=11	ADD	rd=13	Reg-Reg OP

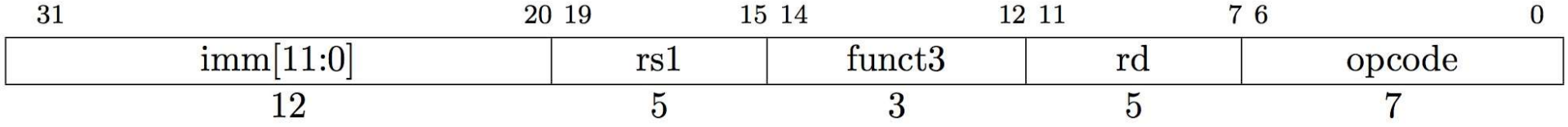
Register	ABI	Use by convention	Preserved?
x0	zero	hardwired to 0, ignores writes	*n/a*
x1	ra	return address for jumps	no
x2	sp	stack pointer	yes
x3	gp	global pointer	*n/a*
x4	tp	thread pointer	*n/a*
x5	t0	temporary register 0	no
x6	t1	temporary register 1	no
x7	t2	temporary register 2	no
x8	s0 *or* fp	saved register 0 *or* frame pointer	yes
x9	s1	saved register 1	yes
x10	a0	return value *or* function argument 0	no
x11	a1	return value *or* function argument 1	no
x12	a2	function argument 2	no
x13	a3	function argument 3	no
x14	a4	function argument 4	no

Register	ABI	Use by convention	Preserved?
x15	a5	function argument 5	no
x16	a6	function argument 6	no
x17	a7	function argument 7	no
x18	s2	saved register 2	yes
x19	s3	saved register 3	yes
x20	s4	saved register 4	yes
x21	s5	saved register 5	yes
x22	s6	saved register 6	yes
x23	s7	saved register 6	yes
x24	s8	saved register 8	yes
x25	s9	saved register 9	yes
x26	s10	saved register 10	yes
x27	s11	saved register 11	yes
x28	t3	temporary register 3	no
x29	t4	temporary register 4	no
x30	t5	temporary register 5	no
x31	t6	temporary register 6	no
pc	*(none)*	program counter	*n/a*

Operacions amb registres

0000000	rs2	rs1	000	rd	0110011	ADD
0100000	rs2	rs1	000	rd	0110011	SUB
0000000	rs2	rs1	001	rd	0110011	SLL
0000000	rs2	rs1	010	rd	0110011	SLT
0000000	rs2	rs1	011	rd	0110011	SLTU
0000000	rs2	rs1	100	rd	0110011	XOR
0000000	rs2	rs1	101	rd	0110011	SRL
0100000	rs2	rs1	101	rd	0110011	SRA
0000000	rs2	rs1	110	rd	0110011	OR
0000000	rs2	rs1	111	rd	0110011	AND

Instruccions amb Immediats



- Only one field is different from R-format, rs2 and funct7 replaced by 12-bit signed immediate, **`imm[11:0]`**
- Remaining fields (rs1, funct3, rd, opcode) same as before
- imm[11:0] can hold values in range [-2048_{ten} , $+2047_{ten}$]
- Immediate is always sign-extended to 32-bits before use in an arithmetic operation
- We'll later see how to handle immediates > 12 bits

Instruccions amb Immediats

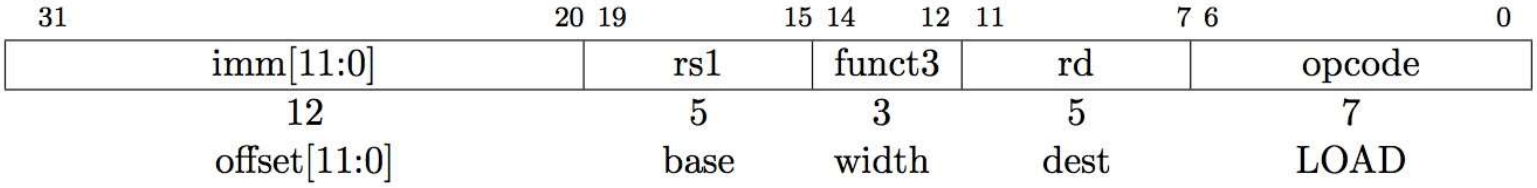
All RV32 I-format Arithmetic Instructions

imm[11:0]		rs1	000	rd	0010011	ADDI
imm[11:0]		rs1	010	rd	0010011	SLTI
imm[11:0]		rs1	011	rd	0010011	SLTIU
imm[11:0]		rs1	100	rd	0010011	XORI
imm[11:0]		rs1	110	rd	0010011	ORI
imm[11:0]		rs1	111	rd	0010011	ANDI
0000000	shamt	rs1	001	rd	0010011	SLLI
0000000	shamt	rs1	101	rd	0010011	SRLI
0100000	shamt	rs1	101	rd	0010011	SRAI

One of the higher-order immediate bits is used to distinguish “shift right logical” (SRLI) from “shift right arithmetic” (SRAI)

“Shift-by-immediate” instructions only use lower 5 bits of the immediate value for shift amount (can only shift by 0-31 bit positions)

Load Instructions are also I-Type

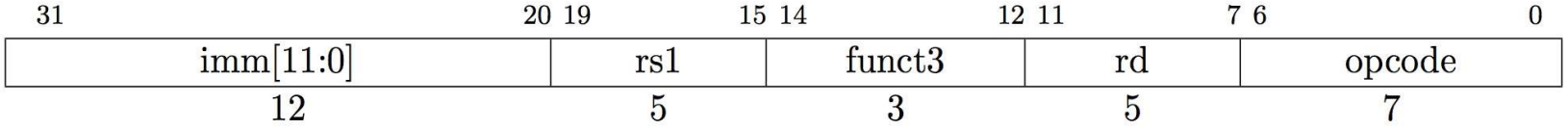


- The 12-bit signed immediate is added to the base address in register rs1 to form the memory address
  - This is very similar to the add-immediate operation but used to create address not to create final result
- The value loaded from memory is stored in register rd

I-Format Load Example

- RISC-V Assembly Instruction:

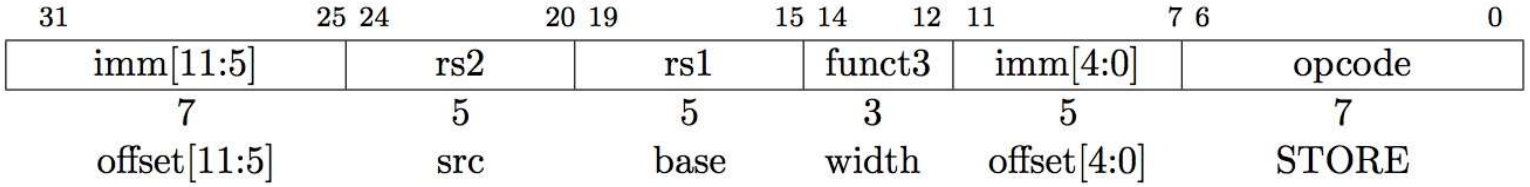
  `lw x14, 8(x2)`



000000001000	00010	010	01110	0000011
imm=+8	rs1=2	LW	rd=14	LOAD

imm[11:0]	rs1	000	rd	0000011	LB
imm[11:0]	rs1	001	rd	0000011	LH
imm[11:0]	rs1	010	rd	0000011	LW
imm[11:0]	rs1	100	rd	0000011	LBU
imm[11:0]	rs1	101	rd	0000011	LHU

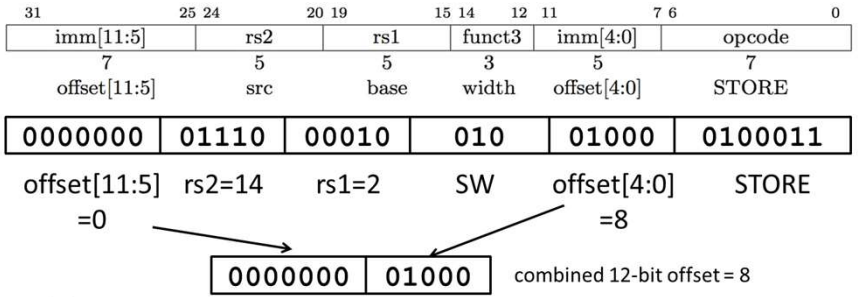
S-Format Used for Stores



- Store needs to read two registers, rs1 for base memory address, and rs2 for data to be stored, as well as need immediate offset!
- Can't have both rs2 and immediate in same place as other instructions!
- Note that stores don't write a value to the register file, ***no rd***!
- RISC-V design decision is move low 5 bits of immediate to where rd field was in other instructions – keep rs1/rs2 fields in same place
  - register names more critical than immediate bits in hardware design

- RISC-V Assembly Instruction:

```
sw x14, 8(x2)
```



imm[11:5]	rs2	rs1	000	imm[4:0]	0100011	SB
imm[11:5]	rs2	rs1	001	imm[4:0]	0100011	SH
imm[11:5]	rs2	rs1	010	imm[4:0]	0100011	SW

RISC-V Conditional Branches

- E.g., **`BEQ x1, x2, Label`**
- Branches read two registers but don't write a register (similar to stores)
- How to encode label, i.e., where to branch to?

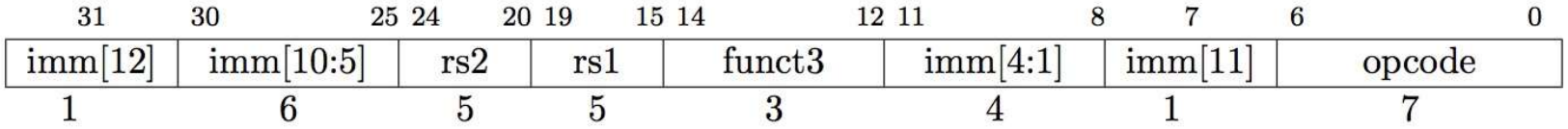
- PC-Relative Addressing: Use the `immediate` field as a two's-complement offset to PC
  - Branches generally change the PC by a small amount
  - Can specify ± 2^{11} addresses from the PC
- Why not use byte address offset from PC?

RISC-V Conditional Branches

- One idea: To improve the reach of a single branch instruction, multiply the offset by four bytes before adding to PC
- This would allow one branch instruction to reach ± 2^{11} × 32-bit instructions either side of PC
  - Four times greater reach than using byte offset

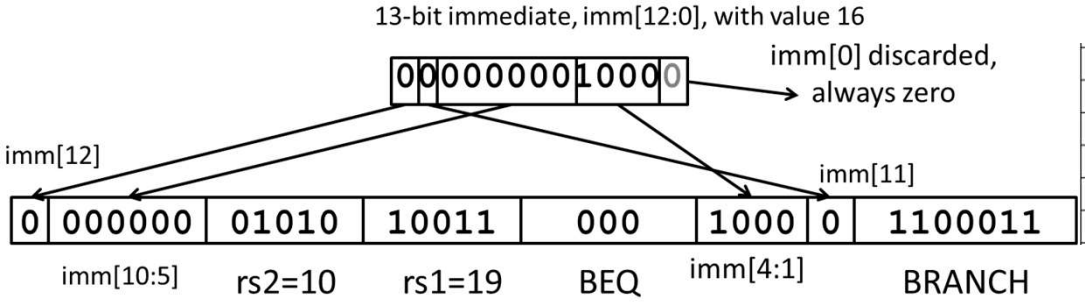
    - If we don't take the branch:
      `PC = PC + 4` (i.e., next instruction)
    - If we do take the branch:
      `PC = PC + immediate*4`
    - **Observations:**
      - `immediate` is number of instructions to jump (remember, specifies words) either forward (+) or backwards (–)

RISC-V Conditional Branches



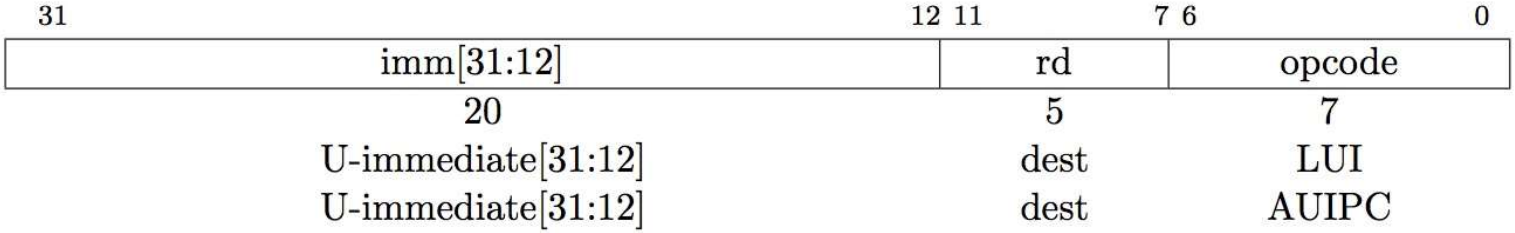
Branch Example, complete encoding

```
beq   x19,x10, offset = 16 bytes
```



imm[12|10:5]	rs2	rs1	000	imm[4:1|11]	1100011	BEQ
imm[12|10:5]	rs2	rs1	001	imm[4:1|11]	1100011	BNE
imm[12|10:5]	rs2	rs1	100	imm[4:1|11]	1100011	BLT
imm[12|10:5]	rs2	rs1	101	imm[4:1|11]	1100011	BGE
imm[12|10:5]	rs2	rs1	110	imm[4:1|11]	1100011	BLTU
imm[12|10:5]	rs2	rs1	111	imm[4:1|11]	1100011	BGEU

U-Format for “Upper Immediate” instructions



- Has 20-bit immediate in upper 20 bits of 32-bit instruction word
- One destination register, rd
- Used for two instructions
  - LUI – Load Upper Immediate
  - AUIPC – Add Upper Immediate to PC

LUI to create long immediates

- LUI writes the upper 20 bits of the destination with the immediate value, and clears the lower 12 bits.
- Together with an ADDI to set low 12 bits, can create any 32-bit value in a register using two instructions (LUI/ADDI).

```
LUI x10, 0x87654       # x10 = 0x87654000
ADDI x10, x10, 0x321# x10 = 0x87654321
```

Exemple

- Feu un programa en assemblador que carregui dos valors de una determinada posició de memòria en dos registres: a1 i a0, faci la suma i ho guardi en a2. Finalment guarda el resultat en memòria.
- Feu un programa en assemblador que carregui dos valors de la memòria en dos registres. Si els valors són iguals, que faci el producte, si un és negatiu que faci la suma, si els dos són positius que faci la resta i si tots dos són negatius, que els passi a positiu i els guardi en memòria.