UNIVERSITAT DE BARCELONA

Pràctica 7 del simulador i8085

Alejandro Guzman

10 Juny 2022

ÍNDEX

1 OBJECTIUS

L'objectiu d'aquesta darrera pràctica és programar en ensamblador diverses aplicacions, demostrant els coneixements adquirits a teoria i al llarg de l'assignatura. En particular, elaborarem un programa capaç de realitzar les següents operacions: suma, resta (operacions aritmètiques), producte lògic i suma lògica (AND i OR respectivament, operacions lògiques).

Aquestes es realitzaran comunicant-se amb l'usuari: a la pantalla de text de l'i8085 mostrarem els caràcters de l'operació (números 0-9 i símbols +,-,&,| i =). Usarem els signes & i | per referirnos a les operacions AND i OR, respectivament.

Cal destacar, tal i com es comenta al guió de pràctiques, que la dificultat de la tasca augmenta en funció del nombre de xifres amb què vulguem treballar. En el nostre cas, veurem una implementació per realitzar les operacions amb nombres de 3 xifres, tot i que veurem que, en alguns casos, seria fàcilment ampliable a xifres arbitràries.

En resum, la pràctica consisteix en introduir una operació per consola, a través del teclat. Un cop entrada aquesta informació, la tractarem, operarem amb ella i, finalment, mostrarem el resultat per pantalla.

2 QÜESTIONS PLANTEJADES AL GUIÓ DE PRÀCTIQUES

1 Suma de dos valors introduïts per consola

En aquest exercici se'ns demana dissenyar una subrutina que, a partir de dos nombres en base 10 introduïts per teclat, faci la suma i mostri el resultat. Llavors l'usuari entrarà els nombres en base decimal i el programa s'encarregarà d'operar de manera correcta aquestes dades, tot i que l'ordinador treballi en base binària (o hexadecimal en el seu defecte, ja que la conversió és directa).

La implementació que veurem no controla els caràcters entrats per l'usuari (permet indistintament de l'estat del programa tots els caràcters tret dels símbols d'operació lògica: 0-9,+,- i =), i només realitza correctament la suma quan s'entren les dades en un format específic. El programa retorna la suma d'un nombre de fins a 3 xifres, seguidament del signe de suma (+)// el segon operand i, finalment, l'igual per indicar al processador que es vol operar. Un exemple d'execució és el següent:

```
.define
  allowed_count 13  // Nombre de caracters permesos
  one         1  // Nombre 1

.data 00h
  allowed:  db 30h,31h,32h,33h,34h,35h,36h,37h,38h,39h,2Bh,2Dh,3Dh  // Caracters permesos
     : 0-9,+,-,=
.data 10h
  counter:  db 00h                  // Comptador de xifres resultat
  resultat: db 00h,00h,00h,00h              // Resultat operacio

.org 200h
// Programa Principal
  lxi H, pila    // Punter de pila apuntant a 100h
  sphl
  mvi H, E0h     // Parell HL apuntant
  mvi L, 00h     // a la memoria de text
  bucle:       // Loop infinit fins que entrem =
  jnz bucle

  call suma    // Subrutina per operar i mostrar el resultat
  hlt      // El programa finalitza

.org 0024h       // Direccio de interrupció TRAP
  call string_in  // Crida a subrutina d'introduccio
  ret        // de dades per consola

.org 100h        // Posicio Pila
  pila:

.org 300h
// Rutina captura i mostra
string_in:
  in 00h     // Port d'entrada
  cpi 00h    // Si no hi ha caracter introduit, surt
  jz no_tecla    // Si hi ha, escriu-lo per pantalla
tecla:
  call check_allowed // Caracter permes? Si no 00h a acumulador
  cpi 00h
  jz no_tecla    // Escriu-lo si esta permes
  call pointers // Actualitzem els punters a memoria
  mov M,A    // Imprimim el caracter per pantalla
```

```
42    inx H
43  no_tecla:
44    cpi 3Dh    // Mirem si hem d'operar (caracter = i és valid)
45    ret
46
47  check_allowed:     // Subrutina control caracters
48    push D
49    push H
50    mvi E, allowed_count
51    lxi H, allowed
52  allowed_loop:      // Comprova si el caracter esta a
53    mov D,M      // la llista de caracters permesos
54    cmp D
55    jz is_allowed
56    inx H
57    dcr E
58    jnz allowed_loop
59    jmp not_allowed
60  is_allowed:      // Esta permes: no modificar
61    mov A,D
62    jmp end_allowed
63  not_allowed:     // No permes: posar 00h a acumulador
64    mvi A,00h
65  end_allowed:
66    pop H
67    pop D
68    ret
69
70  pointers:      // Subrutina per actualitzar punters
71    push PSW     // Evitem perdre el valor de l'acumulador
72    cpi 2Fh    // Carry 0 => [A] > 2Fh, Carry 1 => [A] < 2Fh
73    jnc num_equal // Carry = 0 => Hem entrat numero o =
74    mov A,D    // Mirem si el signe és un menys del 1r operand
75    cpi FFh
76    jz end     // Si? No hem d'actualitzar cap punter
77    mov A,E    // No? Mirem si es signe de l'operacio
78    cpi 00h
79    jnz end    // No? Es el signe del 2n operand, no actualitzem res
80    mov E,L    // Si? Guardem l'offset del signe d'operacio a E!
81    jmp end
82  num_equal:
83    cpi 3Dh    // Mirem si hem introduit =
84    jnz num    // No? Hem entrat un numero
85      mov C,L    // Si? Guardem offset a C!
86    jmp end
87  num:
88    mov A,D    // Mirem si el númeroeés la primera xifra del 1r operand
89    cpi FFh
90    jnz notD     // No? Mirem si hem d'actualitzar algun punter
91    mov D,L    // Si? Guardem l'offset a D!
92    jmp end
93  notD:        // Mirem si el número es la primera xifra del 2n operand
94    mov A,E    // S'ha entrat el signe de l'operacio?
95    cpi 00h
96    jz end     // No? Hem entrat una xifra qualsevol del 1r operand
97    mov A,B    // Si? Actualitzem si no tenim cap xifra del 2n operand
98    cpi 00h
99    jnz end    // Hem entrat alguna? No actualitzem cap punter
100   mov B,L    // No? Guardem offset primer xifra 2n operand a H!
101 end:
102   pop PSW    // Recuperem l'acumulador i finalitzem la subrutina
103   ret
```

```
104
105 suma:        // Subrutina per realitzar la suma i mostrar el resultat
106   push H  // Guardem el punter a on hem d'imprimir per no modificar-lo
107   mov A,D   // Mirem si el primer operand es negatiu
108   cpi 00h
109   jnz sum_op1_neg // Ho es? Mirem el signe del segon operand
110   mov A,E   // No ho es? Es el segon operand negatiu?
111   inr A
112   cmp B
113   jnz sum_op2_neg // Si? Tindrem operacio +num1 + -num2
114   call sumar    // No? Tindrem operacio +num1 + +num2
115   jmp print   // Imprimim el resultat (no negatiu)
116 sum_op1_neg:
117   mov A,E   // Mirem el signe del segon operand
118   inr A
119   cmp B
120   jnz sum_op_neg  // Es negatiu? Tindrem -num1 + -num2
121 min_plus:
122   call compare  // No ho es? Tindrem -num1 + +num2
123   cpi 30h   // Son iguals?
124   jz resul_zero // Si? El resultat sera 0!
125   cpi 00h   // No? Quin es mes gran?
126   jnz resta_neg // Si num1 > num2, restarem num1 - num2
127   call change_ops // Si num2 > num1, restarem num2 - num1
128 resta_pos:
129   call restar   // Restem
130   jmp print   // Imprimim el resultat (positiu)
131 resta_neg:
132   call restar   // Restem
133   jmp neg_print // Imprimim el resultat (negatiu)
134 sum_op2_neg:    // +num1 + -num2!
135   call compare  // Comparem operands
136   cpi 30h   // Son iguals?
137   jz resul_zero // Si? El resultat sera 0!
138   cpi 00h   // No? Quin es mes gran?
139   jnz resta_pos // Si num1 > num2, restarem num1 - num2
140   call change_ops // Si num2 > num1, restarem num2 - num1
141   jmp resta_neg
142 sum_op_neg:
143   call sumar  // Fem la suma com si fossin positius
144   jmp neg_print // Imprimim el resultat amb un menys!
145 resul_zero:
146   call keep_resul // Guardem el 0 i imprimim el resultat
147   jmp print
148 neg_print:
149   pop H     // Recuperem l'adreca a on imprimir el resultat
150   mvi A,2Dh   // Imprimim el signe -
151   mov M,A
152   inx H     // Incrementem per seguir imprimint
153   jmp pos_print // Ara imprimim el nombre com si fos positiu
154 print:
155   pop H
156 pos_print:
157   lxi B,counter
158   ldax B
159   mov D,A   // Guardem al registre D el comptador de xifres
160   mov A,C
161   add D
162   mov C,A   // Apuntem amb BC a la primera xifra del resultat
163 deal_zeros:     // Tractem els possibles 0s a les primeres xifres
164   mov A,D
165   cpi one   // Mirem si nomes falta una xifra a considerar
```

```
166   jz loop    // Si es aixi, la imprimim
167   ldax B     // En cas contrari, mirem si la primera xifra es 0!
168   cpi 30h
169   jnz show     // Si no es zero, mostrem tot el resultat restant
170   dcr D      // Si es zero, no imprimirem aquesta xifra!
171   dcr C
172   jmp deal_zeros  // Seguim buscant possibles zeros a l'esquerra
173 loop:
174   ldax B
175 show:
176   mov M,A    // Imprimim una xifra
177   inx H      // Incrementem per seguir imprimint
178   dcr C
179   dcr D
180   jnz loop     // Seguim imprimint si no hem acabat
181   ret
182
183 sumar:       // Subrutina per sumar dos nombres positius
184 sum_no_carry:    // Cas en que sumem dues xifres sense carry
185   dcr E      // Apuntem a les xifres menys
186   dcr C      // significatives per sumar
187   mov L,E
188   mov A,M    // Carreguem una a l'acumulador
189   mov L,C
190   add M      // Sumem les xifres
191   sui 30h    // Restem el +30h del codi ASCII!
192   cpi 3Ah    // Carry = 0 => Suma xifres >= 10, emportem una
193   jnc sum_deal  // Si? Restem 10 al resultat i n'emportem una
194 sum_no_deal:
195   call keep_resul // Guardem el resultat
196   mov A,E
197   cmp D      // Hem acabat de sumar xifres primer operand?
198   jz nadd_op1    // Si? Mirem que passa amb el segon
199   mov A,C    // No? Mirem igualment que passa amb el segon
200   cmp B
201   jnz sum_no_carry  // Si no hem acabat, seguim sumant (sense carry)
202 add_op:      // Si hem acabat, guardem les xifres restants
203   dcr E
204   mov L,E
205   mov A,M
206 add_resul:
207   call keep_resul // Guardem les xifres restants de l'operand
208   mov A,E
209   cmp D
210   jnz add_op     // Si no hem acabat, seguim guardant
211   jmp nadd     // Si hem acabat, la suma esta completa
212 nadd_op1:
213   mov A,C
214   cmp B      // Hem acabat de sumar xifres segon operand?
215   jz nadd    // Si? Hem acabat de sumar!
216   mov E,C    // No? Passem les dades al parell de registres DE
217   mov D,B    // i usem el bucle add_op ja programat per guardar
218   jmp add_op     // les xifres restants
219 sum_carry:       // Cas en que sumem dues xifres amb carry
220   dcr E      // Apuntem a les xifres menys
221   dcr C      // significatives per sumar
222   mov L,E
223   mov A,M    // Carreguem una a l'acumulador
224   mov L,C
225   add M      // Sumem les xifres
226   sui 2Fh    // Restem el +30h del codi ASCII pero tenim +1 de carry!
227   cpi 3Ah    // Carry = 1 => Suma xifres < 10, no emportem res
```

```
228   jc sum_no_deal  // Seguim la suma sense tractar el carry
229 sum_deal:      // Tractem el carry al sumar xifres
230   sui Ah     // Restem 10 al resultat ja que ens emportem una
231   call keep_resul // Guardem el resultat
232   mov A,E
233   cmp D      // Hem acabat de sumar xifres primer operand?
234   jz c_nadd_op1    // Si? Mirem que passa amb el segon
235   mov A,C    // No? Mirem igualment que passa amb el segon!
236   cmp B
237   jnz sum_carry // Si no hem acabat, seguim sumant (amb carry)
238 c_add_op:      // Si hem acabat, sumem les xifres restants amb el carry
239   dcr E
240   mov L,E
241   mov A,M    // Carreguem la xifra
242   inr A      // Sumem el carry!
243   cpi 3Ah    // Carry = 1 => Suma xifres < 10, no emportem cap
244   jc add_resul  // No tractem més carry? Afegim les xifres restants!
245 deal_c_op:       // Tractem el carry de la suma operand + 1
246   sui Ah     // Restem 10 al resultat ja que ens emportem una!
247   call keep_resul // Guardem el resultat
248   mov A,E
249   cmp D      // Hem acabat de sumar xifres?
250   jz add_carry  // Si? Falta la que ens emportem!
251   jmp c_add_op  // No? Seguim sumant xifres amb carry!
252 c_nadd_op1:
253   mov A,C    // Veiem si ja hem acabat de sumar el segon operand
254   cmp B
255   jz add_carry  // Si hem acabat, falta la que ens emportem!
256   mov E,C    // Si no hem acabat, passem les dades al parell de
257   mov D,B    // registres DE i usem el bucle c_add_op ja programat
258   jmp c_add_op  // per guardar les xifres restants
259 add_carry:
260   mvi A,31h    // Guardem al resultat la que ens emportavem
261   call keep_resul
262 nadd:
263   ret
264
265 restar:      // Subrutina per restar dos nombres positius num1 > num2
266 res_no_carry:    // Cas en que restem dues xifres sense carry
267   dcr E      // Apuntem a les xifres menys
268   dcr C      // significatives per restar
269   mov L,E
270   mov A,M    // Carreguem una a l'acumulador
271   adi 30h    // Sumem el +30h del codi ASCII
272   mov L,C
273   sub M      // Restem les xifres
274   cpi 30h    // Carry = 0: 0 <= Resta xifres < 9, no emportem cap
275   jc res_deal    // Ens emportem? Sumem 10 al resultat i n'emportem una
276 res_no_deal:
277   call keep_resul // Guardem el resultat
278   mov A,C
279   cmp B      // Hem acabat de restar xifres segon operand?
280   jnz res_no_carry  // Si no hem acabat, seguim restant (sense carry)
281   mov A,E    // Si hem acabat, mirem si hem restat totes les xifres del primer
282   cmp D
283   jnz add_op     // Si no hem acabat, afegim les xifres restants!
284   jmp nsub     // Si hem acabat, finalitzem l'operacio
285 res_carry:       // Cas en que restem dues xifres amb carry
286   dcr E      // Apuntem a les xifres menys
287   dcr C      // significatives per restar
288   mov L,E
289   mov A,M    // Carreguem una a l'acumulador
```

```
290   adi 2Fh    // Sumem el +30h del codi ASCII pero tenim -1 de carry
291   mov L,C
292   sub M      // Restem les xifres
293   cpi 30h    // Carry = 0: 0 <= Resta xifres < 9, no emportem cap
294   jnc res_no_deal // No ens emportem? Tornem al cas sense carry
295 res_deal:
296   adi Ah     // Sumem 10 al resultat ja que ens emportem una!
297   call keep_resul // Guardem el resultat
298   mov A,C
299   cmp B      // Hem acabat de restar xifres segon operand?
300   jz sub_carry  // Si tenim carry no podem haver acabat de restar
301   jmp res_carry // No? Seguim restant
302 sub_deal:      // Tractem carry de la resta xifra primer operand - carry
303   adi Ah     // Sumem 10 ja que ens emportem una
304   call keep_resul // Guardem el resultat
305   // Tenint carry no podem haver acabat la resta. Seguim propagant-lo
306 sub_carry:
307   dcr E
308   mov L,E
309   mov A,M    // Carreguem xifra a l'acumulador
310   dcr A      // Restem el carry
311   cpi 30h    // Carry = 0: 0 <= Resta xifres < 9, no emportem cap
312   jc sub_deal    // Ens emportem? Sumem 10 al resultat i n'emportem una
313   call keep_resul // No ens emportem? Guardem i mirem si hem acabat la resta!
314   mov A,E
315   cmp D
316   jnz add_op     // Si no hem acabat, afegim les xifres restants
317 nsub:
318   ret      // Si hem acabat, finalitzem la subrutina
319
320 compare:       // Subrutina per comparar operands (A = 1: num1 > num2,
321   push D     // A = 0: num2 > num1, A = 30h: num1 = num2)
322   push B     // Guardem els offsets abans de fer possibles modificacions
323   mov A,E
324   sub D
325   mov L,A
326   mov A,C
327   sub B
328   cmp L      // Mirem quin nombre te mes xifres!
329   jnz dif_xifres  // Diferents xifres? Un és més gran!
330 cmp_xifra:       // Si tenen les mateixes, hem de comparar-los xifra a xifra
331   mov L,B    // Guardem primera xifra segon operand a acumulador
332   mov A,M
333   mov L,D // Comparem amb la primera xifra primer operand
334   cmp M
335   jnz dif_xifres  // Si les xifres son diferents, un es mes gran!
336   inr D      // Si son iguals, mirem si tenen més xifres!
337   inr B
338   mov A,D    // Com tenen el mateix nombre de xifres, nomes mirem si
339   cmp E      // hem comparat totes les del primer operand
340   jnz cmp_xifra // Si no hem acabat, continuem
341   mvi A,30h    // En cas contrari, els nombres son iguals
342   jmp end_compare
343 dif_xifres:
344   jnc gran_segon  // Si carry = 0 => 2n operand te mes xifres!
345   mvi A,one    // En cas contrari, el primer operand és més gran
346   jmp end_compare
347 gran_segon:
348   mvi A,00h    // El segon operand es mes gran
349 end_compare:
350   pop B      // Recuperem el valor dels offsets i finalitzem la subrutina!
351   pop D
```

```
352   ret
353
354 change_ops:      // Subrutina per permutar els operands
355   push H     // Usem HL com registres auxiliars
356   mov H,D
357   mov L,E
358   mov D,B
359   mov E,C
360   mov B,H
361   mov C,L
362   pop H
363   ret
364
365 keep_resul:      // Subrutina per guardar el resultat d'una xifra a memoria
366   push H
367   push PSW     // Guardem parells de registres a la pila
368   lxi H,counter // Apuntem amb HL al comptador de xifres del resultat
369   inr M      // Sumem 1 a aquest comptador
370   mov A,L    // Fiquem a l'acumulador la part baixa de l'adreÃ§a
371   add M      // A acumulador esta la part baixa adreÃ§a on hem de guardar!
372   mov L,A
373   pop PSW
374   mov M,A    // Guardem el resultat de l'operacio
375   pop H      // Recuperem la informació guardada a la pila i retornem
376   ret
```

Abans de començar la part del codi referent a memòria de programa, definim dues constants i diferents parts de la memòria reservades. Les constants són el nombre de caràcters a permetre en l'entrada per teclat i el nombre 1. D'altra banda, com a dades tenim els caràcters permesos en el seu codi ASCII i, a la posició 10h de memòria, un comptador i aquest mateix resultat (reservant fins a 4 bytes per guardar les 4 possibles xifres d'una suma de nombres de 3 xifres).

És important tenir en compte que guardarem el resultat en ordre invers ja que, per com hem implementat la suma, operarem xifra a xifra. D'aquesta manera, guardarem primer la xifra de les unitats, seguida de les desenes, etc. Tanmateix, el resultat el mostrem correctament (tal i com hem vist als exemples d'execució).

La següent part del programa és el corresponent al *main* en un llenguatge d'alt nivell. Modifiquem el SP per apuntar a la nova direcció de pila (la 100h, tal i com s'especifica sota aquesta part del codi) i donem valors a tots els registres: el parell HL apunta a la memòria on es guarden els caràcters que es mostren a la pantalla de text i tots els altres es reinicien a 00h meny sel registre D, degut a que apuntarà a la primera xifra del primer operand i l'hem d'inicialitzar amb un valor que no pugui prendre a mesura que entrem caràcters per teclat. D'aquesta manera, si carreguem al registre H E0h, substituint la part baixa de la direcció (contingut del registre L) pel d'un d'aquests 4 registres, apuntarem correctament al caràcter en qüestió. Cal destacar que E apuntarà al signe de l'operació i C al signe d'igual.

Aquesta primera part inicialitza els registres i comença a executar un bucle infinit, per tal d'esperar interrupcions produïdes a l'entrar dades per teclat. Cal destacar que aquest bucle finalitzarà amb l'entrada del signe igual (veure final de la subrutina string_in, comprovem si el caràcter entrat té codi ASCII 3Dh, és a dir, si és el signe =), per això és important seguir el format d'entrada, per no provocar errors. Un cop el programa surt del bucle, crida a la subrutina suma, que opera i mostra el resultat per pantalla. Seguidament el programa finalitza.

La direcció de memòria assignada a la interrupció TRAP (0024h) manté les instruccions dels programes de la part guiada: cridem a una subrutina que processarà l'entrada per teclat. A

partir de la direcció de memòria 300h, comença el programa com a tal (les funcionalitats).

La subrutina *string_incaptura* mostra el caràcter entrat, l'única modificació respecte l'exercici 2 guiat és quins caràcters permetem (esmentats anteriorment), amb quin parell de registres apuntem a la memòria de text (HL en comptes de BC) i la crida a la subrutina prèvia a mostrar el caràcter per pantalla (crida a pointers). Aquesta subrutina serveix per seguir l'estratègia d'usar els registres B,C,D,E com offsets per apuntar als operands a memòria. En aquesta subrutina, guardem primer la paraula d'estat de programa a la pila. Aleshores, mirem si hem entrat un número, un signe negatiu o el signe de suma. En funció d'allò entrat, actualitzarem els punters (si hem entrat el primer número d'un operand actualitzarem el registre D o B, si és el signe de suma, el registre E i, finalment, si és el signe igual, el registre C). Un cop actualitzats, recuperem el caràcter entrat de la pila i tornem a la subrutina *string_inper* mostrar aquest caràcter per pantalla. És important tenir en compte que necessitem els punters per saber on es troben les xifres dels operands i poder realitzar la suma amb l'estratègia que veurem més endavant. Només resta veure la subrutina suma, que realitza la suma dels dos operands i la mostra per pantalla. Primer guardem el parell de registres HL a la pila, ja que guarden la direcció de memòria on hem de començar a imprimir el resultat. Un cop fem això hem de determinar de quin tipus de suma es tracta: suma de nombres positius, de dos nombres negatius o d'un positiu i un negatiu. Ho sabem comprovant els *offsets* "punters" B,C,D i E. Si la suma és de dos nombres positius, simplement els sumem i mostrem el resultat per pantalla, si ambdós són negatius, sumem els seus valors absoluts (els nombres com a positius, sense el signe -) i mostrem el resultat per pantalla amb un – davant, indicant que és negatiu (usem que, matemàticament, -A + -B = -(A+B)) i, finalment, si els operands són de signe diferent, determinem quin és més gran (en valor absolut, a través de la subrutina compare). Això ho fem perquè les operacions les fem, tal i com hem esmentat anteriorment, xifra a xifra, seguint els algoritmes de la resta i la suma per nombres decimals. Si el primer operand al fer A – B no és major que el segon, l'operació no serà correcta.

Llavors si sumem dos nombres amb signe diferent, els reordenarem per tal que el primer operand sigui el gran en valor absolut i els restarem. Veiem que la subrutina *suma* simplement processa els operands, crida a diferents subrutines per tractar els nombres i, finalment, els mostra per pantalla el resultat en funció de si és negatiu o positiu. Per imprimir-lo, usem el comptador de xifres de resultat esmentat, per saber quantes hem de mostrar.

Un cop mostrat el resultat, el programa finalitza. Veiem ara les subrutines comparei change_ops: *compare* compara el valor absolut dels dos nombres, seguint l'esquema d'un comparador xifra a xifra. change_ops simplement canvia els *offsets* DE i BC per tal que ara DE apunti al segon operand que a l'haver cridat aquesta subrutina, és més gran en valor absolut, i BC apunti al primer.

Finalment, només resta explicar com fem la suma i la resta(ambdues de dos nombres positius, i en el segon cas amb el primer operand major que el segon)

L'esquema de la suma és senzill, tenim dos blocs ben diferenciats, un que suma dues xifres sense *carry* i un altre que l'afegeix. A mesura que anem sumant xifres, mirem si la suma supera el valor ASCII del 9 (39h), en aquest cas, tenim carry, restem 10 al resultat (Ah)i passem a la part de la suma que operarà amb carry. Cada cop que es determina una xifra, es crida a la subrutina keep_resul,que guarda aquesta xifra ja calculada a memòria per un cop obtingut el resultat, mostrar-lo per pantalla.La resta de la subrutina **sumar** comprova constantment si ja hem acabat d'operar els nombres, en cas contrari, seguim sumant les seves xifres, i si hem aca-

bat, afegim les xifres restants de l'operand que encara en tingui.La subrutina **restar** és essencialment la mateixa però en comptes de restar 10 al tenir *carry* i afegir-lo a la següent xifra, sumarem 10 i restarem una unitat a la següent xifra. També comprovem constantment si hem acabat d'operar, amb la petita diferència que, com el primer operand és més gran que el segon, no podem tenir *carry* al finalitzar l'operació, així que si hem acabat de restar el segon operand i tenim *carry*, el seguim propagant fins finalitzar l'operació. Cal destacar que si no hem acabat de restar però no tenim *carry*, només resta afegir les xifres del primer operand al resultat. Com això ja ho hem implementat a la subrutina sumar(etiqueta add_op), saltem a aquesta posició de memòria quan hem de realitzar aquesta operació.

A continuació passem ara a les qüestions explícites del guió:

Feu servir els adreçaments directe i indirecte i indiqueu al codi on els tenim
Les instruccions que fan servir adreçament directe, només a registre, estan marcades al codi del programa en vermell,i les que fan servir adreçament indirecte a registre en verd (no usem indirecte a memòria). En blau estan marcades aquelles que facin servir ambdós adreçaments (per exemple, la instrucció MOV M,A mou el contingut de l'acumulador, adreçament directe, a la posició de memòria adreçada pel parell de registres HL, adreçament indirecte).

Com gestioneu el problema del signe? I el de l'overflow?
El problema del signe es gestiona tractant amb anterioritat la magnitud dels nombres per determinar abans de realitzar l'operació el signe que tindrà el resultat. D'altra banda, és l'estratègia mateixa en la implementació qui gestiona el problema de l'overflow, com no operem contra acumulador, no ens afecta que només pugui representar fins el nombre 255 (FFh). En altres paraules, tal i com hem vist a les execucions, podem operar amb nombres superiors a 255, i el resultat es mostra correctament, així que no tenim el problemes d'*overflow*.

2 Resta de dos valor introduïts per consola

En aquest exercici se'ns demana implementar la operació resta d'una manera similar a com ho hem fet en l'exercici anterior. Realment a partir de la suma, podem extreure fàcilment la de la resta.

Primera implementació

```
.define
  allowed_count 12  // Nombre de caracters permesos
  one         1   // Nombre 1

.data 00h
  allowed:  db 30h,31h,32h,33h,34h,35h,36h,37h,38h,39h,2Dh,3Dh
.data 10h
  counter:  db 00h                  // Comptador de xifres resultat
  resultat: db 00h,00h,00h,00h              // Resultat operacio

.org 200h
// Programa Principal
  lxi H, pila    // Punter de pila apuntant a 100h
  sphl
  mvi H, E0h     // Parell HL apuntant
  mvi L, 00h     // a la memoria de text
  bucle:       // Loop infinit fins que entrem =
  jnz bucle

  call resta     // Subrutina per operar i mostrar el resultat
```

```
21    hlt      // El programa finalitza
22
23  .org 0024h       // Direccio de interrupció TRAP
24    call string_in   // Crida a subrutina d'introduccio
25    ret        // de dades per consola
26
27  .org 100h        // Posicio Pila
28    pila:
29
30  .org 300h
31  // Rutina captura i mostra
32  string_in:
33    in 00h     // Port d'entrada
34    cpi 00h    // Si no hi ha caracter introduit, surt
35    jz no_tecla    // Si hi ha, escriu-lo per pantalla
36  tecla:
37    call check_allowed // Caracter permes? Si no 00h a acumulador
38    cpi 00h
39    jz no_tecla    // Escriu-lo si esta permes
40    call pointers // Actualitzem els punters a memoria
41    mov M,A    // Imprimim el caracter per pantalla
42    inx H
43  no_tecla:
44    cpi 3Dh    // Mirem si hem d'operar (caracter = i és valid)
45    ret
46
47  check_allowed:     // Subrutina control caracters
48    push D
49    push H
50    mvi E, allowed_count
51    lxi H, allowed
52  allowed_loop:      // Comprova si el caracter esta a
53    mov D,M      // la llista de caracters permesos
54    cmp D
55    jz is_allowed
56    inx H
57    dcr E
58    jnz allowed_loop
59    jmp not_allowed
60  is_allowed:      // Esta permes: no modificar
61    mov A,D
62    jmp end_allowed
63  not_allowed:     // No permes: posar 00h a acumulador
64    mvi A,00h
65  end_allowed:
66    pop H
67    pop D
68    ret
69
70  pointers:      // Subrutina per actualitzar punters
71    push PSW     // Evitem perdre el valor de l'acumulador
72    cpi 2Fh    // Carry 0 => [A] > 2Fh, Carry 1 => [A] < 2Fh
73    jnc num_equal // Carry = 0 => Hem entrat numero o =
74    mov A,D    // Mirem si el signe és un menys del 1r operand
75    cpi FFh
76    jz end     // Si? No hem d'actualitzar cap punter
77    mov A,E    // No? Mirem si es signe de l'operacio
78    cpi 00h
79    jnz end    // No? Es el signe del 2n operand, no actualitzem res
80    mov E,L    // Si? Guardem l'offset del signe d'operacio a E!
81    jmp end
82  num_equal:
```

```
83    cpi 3Dh    // Mirem si hem introduit =
84    jnz num    // No? Hem entrat un numero
85      mov C,L    // Si? Guardem offset a C!
86    jmp end
87  num:
88    mov A,D    // Mirem si el númeroeés la primera xifra del 1r operand
89    cpi FFh
90    jnz notD     // No? Mirem si hem d'actualitzar algun punter
91    mov D,L    // Si? Guardem l'offset a D!
92    jmp end
93  notD:        // Mirem si el número es la primera xifra del 2n operand
94    mov A,E    // S'ha entrat el signe de l'operacio?
95    cpi 00h
96    jz end     // No? Hem entrat una xifra qualsevol del 1r operand
97    mov A,B    // Si? Actualitzem si no tenim cap xifra del 2n operand
98    cpi 00h
99    jnz end    // Hem entrat alguna? No actualitzem cap punter
100   mov B,L    // No? Guardem offset primer xifra 2n operand a H!
101 end:
102   pop PSW    // Recuperem l'acumulador i finalitzem la subrutina
103   ret
104
105 suma:        // Subrutina per realitzar la suma i mostrar el resultat
106   push H   // Guardem el punter a on hem d'imprimir per no modificar-lo
107   mov A,D    // Mirem si el primer operand es negatiu
108   cpi 00h
109   jnz sum_op1_neg // Ho es? Mirem el signe del segon operand
110   mov A,E    // No ho es? Es el segon operand negatiu?
111   inr A
112   cmp B
113   jnz sum_op2_neg // Si? Tindrem operacio +num1 + -num2
114   call sumar     // No? Tindrem operacio +num1 + +num2
115   jmp print    // Imprimim el resultat (no negatiu)
116 sum_op1_neg:
117   mov A,E    // Mirem el signe del segon operand
118   inr A
119   cmp B
120   jnz sum_op_neg   // Es negatiu? Tindrem -num1 + -num2
121 min_plus:
122   call compare   // No ho es? Tindrem -num1 + +num2
123   cpi 30h    // Son iguals?
124   jz resul_zero // Si? El resultat sera 0!
125   cpi 00h    // No? Quin es mes gran?
126   jnz resta_neg // Si num1 > num2, restarem num1 - num2
127   call change_ops // Si num2 > num1, restarem num2 - num1
128 resta_pos:
129   call restar    // Restem
130   jmp print    // Imprimim el resultat (positiu)
131 resta_neg:
132   call restar    // Restem
133   jmp neg_print // Imprimim el resultat (negatiu)
134 sum_op2_neg:     // +num1 + -num2!
135   call compare   // Comparem operands
136   cpi 30h    // Son iguals?
137   jz resul_zero // Si? El resultat sera 0!
138   cpi 00h    // No? Quin es mes gran?
139   jnz resta_pos // Si num1 > num2, restarem num1 - num2
140   call change_ops // Si num2 > num1, restarem num2 - num1
141   jmp resta_neg
142 sum_op_neg:
143   call sumar   // Fem la suma com si fossin positius
144   jmp neg_print // Imprimim el resultat amb un menys!
```

```
145 resta:    // Subrutina per realitzar la resta i mostrar el resultat
146   push H    // Guardem el punter a on hem d'imprimir per no modificar-lo
147   mov A,D   // Mirem si el primer operand és negatiu
148   cpi 00h
149   jnz res_op1_neg // Ho és? Mirem el signe del segon operand
150   mov A,E   // No ho és? Ãs el segon operand negatiu?
151   inr A
152   cmp B
153   jz sum_op2_neg  // No? Tindrem +num1 - +num2 = +num1 + -num2
154   call sumar    // Sí? Tindrem +num1 - -num2 (sumem!)
155   jmp print   // Imprimim el resultat (no negatiu)
156 res_op1_neg:
157   mov A,E   // Mirem el signe del segon operand
158   inr A
159   cmp B
160   jnz min_plus  // Ãs negatiu? Tindrem -num1 - -num2 = -num1 + +num2
161   call sumar    // Ãs positiu? Tindrem -num1 - +num2 = -num1 + -num2
162   jmp neg_print // Sumem i imprimim el resultat amb un menys!
163 resul_zero:
164   call keep_resul // Guardem el 0 i imprimim el resultat
165   jmp print
166 neg_print:
167   pop H     // Recuperem l'adreca a on imprimir el resultat
168   mvi A,2Dh   // Imprimim el signe -
169   mov M,A
170   inx H     // Incrementem per seguir imprimint
171   jmp pos_print // Ara imprimim el nombre com si fos positiu
172 print:
173   pop H
174 pos_print:
175   lxi B,counter
176   ldax B
177   mov D,A   // Guardem al registre D el comptador de xifres
178   mov A,C
179   add D
180   mov C,A   // Apuntem amb BC a la primera xifra del resultat
181 deal_zeros:     // Tractem els possibles 0s a les primeres xifres
182   mov A,D
183   cpi one   // Mirem si nomes falta una xifra a considerar
184   jz loop   // Si es aixi, la imprimim
185   ldax B    // En cas contrari, mirem si la primera xifra es 0!
186   cpi 30h
187   jnz show    // Si no es zero, mostrem tot el resultat restant
188   dcr D     // Si es zero, no imprimirem aquesta xifra!
189   dcr C
190   jmp deal_zeros  // Seguim buscant possibles zeros a l'esquerra
191 loop:
192   ldax B
193 show:
194   mov M,A   // Imprimim una xifra
195   inx H     // Incrementem per seguir imprimint
196   dcr C
197   dcr D
198   jnz loop    // Seguim imprimint si no hem acabat
199   ret
200
201 sumar:      // Subrutina per sumar dos nombres positius
202 sum_no_carry:   // Cas en que sumem dues xifres sense carry
203   dcr E     // Apuntem a les xifres menys
204   dcr C     // significatives per sumar
205   mov L,E
206   mov A,M   // Carreguem una a l'acumulador
```

```
207   mov L,C
208   add M      // Sumem les xifres
209   sui 30h    // Restem el +30h del codi ASCII!
210   cpi 3Ah    // Carry = 0 => Suma xifres >= 10, emportem una
211   jnc sum_deal  // Si? Restem 10 al resultat i n'emportem una
212 sum_no_deal:
213   call keep_resul // Guardem el resultat
214   mov A,E
215   cmp D      // Hem acabat de sumar xifres primer operand?
216   jz nadd_op1    // Si? Mirem que passa amb el segon
217   mov A,C    // No? Mirem igualment que passa amb el segon
218   cmp B
219   jnz sum_no_carry  // Si no hem acabat, seguim sumant (sense carry)
220 add_op:      // Si hem acabat, guardem les xifres restants
221   dcr E
222   mov L,E
223   mov A,M
224 add_resul:
225   call keep_resul // Guardem les xifres restants de l'operand
226   mov A,E
227   cmp D
228   jnz add_op     // Si no hem acabat, seguim guardant
229   jmp nadd     // Si hem acabat, la suma esta completa
230 nadd_op1:
231   mov A,C
232   cmp B      // Hem acabat de sumar xifres segon operand?
233   jz nadd    // Si? Hem acabat de sumar!
234   mov E,C    // No? Passem les dades al parell de registres DE
235   mov D,B    // i usem el bucle add_op ja programat per guardar
236   jmp add_op     // les xifres restants
237 sum_carry:       // Cas en que sumem dues xifres amb carry
238   dcr E      // Apuntem a les xifres menys
239   dcr C      // significatives per sumar
240   mov L,E
241   mov A,M    // Carreguem una a l'acumulador
242   mov L,C
243   add M      // Sumem les xifres
244   sui 2Fh    // Restem el +30h del codi ASCII pero tenim +1 de carry!
245   cpi 3Ah    // Carry = 1 => Suma xifres < 10, no emportem res
246   jc sum_no_deal  // Seguim la suma sense tractar el carry
247 sum_deal:      // Tractem el carry al sumar xifres
248   sui Ah     // Restem 10 al resultat ja que ens emportem una
249   call keep_resul // Guardem el resultat
250   mov A,E
251   cmp D      // Hem acabat de sumar xifres primer operand?
252   jz c_nadd_op1    // Si? Mirem que passa amb el segon
253   mov A,C    // No? Mirem igualment que passa amb el segon!
254   cmp B
255   jnz sum_carry // Si no hem acabat, seguim sumant (amb carry)
256 c_add_op:      // Si hem acabat, sumem les xifres restants amb el carry
257   dcr E
258   mov L,E
259   mov A,M    // Carreguem la xifra
260   inr A      // Sumem el carry!
261   cpi 3Ah    // Carry = 1 => Suma xifres < 10, no emportem cap
262   jc add_resul  // No tractem més carry? Afegim les xifres restants!
263 deal_c_op:       // Tractem el carry de la suma operand + 1
264   sui Ah     // Restem 10 al resultat ja que ens emportem una!
265   call keep_resul // Guardem el resultat
266   mov A,E
267   cmp D      // Hem acabat de sumar xifres?
268   jz add_carry  // Si? Falta la que ens emportem!
```

```
269   jmp c_add_op  // No? Seguim sumant xifres amb carry!
270 c_nadd_op1:
271   mov A,C   // Veiem si ja hem acabat de sumar el segon operand
272   cmp B
273   jz add_carry  // Si hem acabat, falta la que ens emportem!
274   mov E,C   // Si no hem acabat, passem les dades al parell de
275   mov D,B   // registres DE i usem el bucle c_add_op ja programat
276   jmp c_add_op  // per guardar les xifres restants
277 add_carry:
278   mvi A,31h   // Guardem al resultat la que ens emportavem
279   call keep_resul
280 nadd:
281   ret
282
283 restar:     // Subrutina per restar dos nombres positius num1 > num2
284 res_no_carry:   // Cas en que restem dues xifres sense carry
285   dcr E     // Apuntem a les xifres menys
286   dcr C     // significatives per restar
287   mov L,E
288   mov A,M   // Carreguem una a l'acumulador
289   adi 30h   // Sumem el +30h del codi ASCII
290   mov L,C
291   sub M     // Restem les xifres
292   cpi 30h   // Carry = 0: 0 <= Resta xifres < 9, no emportem cap
293   jc res_deal   // Ens emportem? Sumem 10 al resultat i n'emportem una
294 res_no_deal:
295   call keep_resul // Guardem el resultat
296   mov A,C
297   cmp B     // Hem acabat de restar xifres segon operand?
298   jnz res_no_carry  // Si no hem acabat, seguim restant (sense carry)
299   mov A,E   // Si hem acabat, mirem si hem restat totes les xifres del primer
300   cmp D
301   jnz add_op    // Si no hem acabat, afegim les xifres restants!
302   jmp nsub    // Si hem acabat, finalitzem l'operacio
303 res_carry:      // Cas en que restem dues xifres amb carry
304   dcr E     // Apuntem a les xifres menys
305   dcr C     // significatives per restar
306   mov L,E
307   mov A,M   // Carreguem una a l'acumulador
308   adi 2Fh   // Sumem el +30h del codi ASCII pero tenim -1 de carry
309   mov L,C
310   sub M     // Restem les xifres
311   cpi 30h   // Carry = 0: 0 <= Resta xifres < 9, no emportem cap
312   jnc res_no_deal // No ens emportem? Tornem al cas sense carry
313 res_deal:
314   adi Ah    // Sumem 10 al resultat ja que ens emportem una!
315   call keep_resul // Guardem el resultat
316   mov A,C
317   cmp B     // Hem acabat de restar xifres segon operand?
318   jz sub_carry  // Si tenim carry no podem haver acabat de restar
319   jmp res_carry // No? Seguim restant
320 sub_deal:     // Tractem carry de la resta xifra primer operand - carry
321   adi Ah    // Sumem 10 ja que ens emportem una
322   call keep_resul // Guardem el resultat
323   // Tenint carry no podem haver acabat la resta. Seguim propagant-lo
324 sub_carry:
325   dcr E
326   mov L,E
327   mov A,M   // Carreguem xifra a l'acumulador
328   dcr A     // Restem el carry
329   cpi 30h   // Carry = 0: 0 <= Resta xifres < 9, no emportem cap
330   jc sub_deal   // Ens emportem? Sumem 10 al resultat i n'emportem una
```

```
331   call keep_resul // No ens emportem? Guardem i mirem si hem acabat la resta!
332   mov A,E
333   cmp D
334   jnz add_op    // Si no hem acabat, afegim les xifres restants
335 nsub:
336   ret     // Si hem acabat, finalitzem la subrutina
337
338 compare:      // Subrutina per comparar operands (A = 1: num1 > num2,
339   push D    // A = 0: num2 > num1, A = 30h: num1 = num2)
340   push B    // Guardem els offsets abans de fer possibles modificacions
341   mov A,E
342   sub D
343   mov L,A
344   mov A,C
345   sub B
346   cmp L     // Mirem quin nombre te mes xifres!
347   jnz dif_xifres  // Diferents xifres? Un és més gran!
348 cmp_xifra:      // Si tenen les mateixes, hem de comparar-los xifra a xifra
349   mov L,B   // Guardem primera xifra segon operand a acumulador
350   mov A,M
351   mov L,D // Comparem amb la primera xifra primer operand
352   cmp M
353   jnz dif_xifres  // Si les xifres son diferents, un es mes gran!
354   inr D     // Si son iguals, mirem si tenen més xifres!
355   inr B
356   mov A,D   // Com tenen el mateix nombre de xifres, nomes mirem si
357   cmp E     // hem comparat totes les del primer operand
358   jnz cmp_xifra // Si no hem acabat, continuem
359   mvi A,30h   // En cas contrari, els nombres son iguals
360   jmp end_compare
361 dif_xifres:
362   jnc gran_segon  // Si carry = 0 => 2n operand te mes xifres!
363   mvi A,one   // En cas contrari, el primer operand és més gran
364   jmp end_compare
365 gran_segon:
366   mvi A,00h   // El segon operand es mes gran
367 end_compare:
368   pop B     // Recuperem el valor dels offsets i finalitzem la subrutina!
369   pop D
370   ret
371
372 change_ops:     // Subrutina per permutar els operands
373   push H    // Usem HL com registres auxiliars
374   mov H,D
375   mov L,E
376   mov D,B
377   mov E,C
378   mov B,H
379   mov C,L
380   pop H
381   ret
382
383 keep_resul:     // Subrutina per guardar el resultat d'una xifra a memoria
384   push H
385   push PSW    // Guardem parells de registres a la pila
386   lxi H,counter // Apuntem amb HL al comptador de xifres del resultat
387   inr M     // Sumem 1 a aquest comptador
388   mov A,L   // Fiquem a l'acumulador la part baixa de l'adreÃ§a
389   add M     // A acumulador esta la part baixa adreÃ§a on hem de guardar!
390   mov L,A
391   pop PSW
392   mov M,A   // Guardem el resultat de l'operacio
```

```
393   pop H      // Recuperem la informació guardada a la pila i retornem
394   ret
```

Notem que hi ha molt poques modificacions respecte el programa de la suma. Això és degut a la similitud d'aquestes operacions.

La primera diferència es que permetem un caràcter menys, ja que en aquest programa no entrarem el símbol de suma en cap moment. L'única altra modificació és la crida a la subrutina *resta* en comptes de a suma. Llavors hem conservat la subrutina suma perquè en realitat aquest programa consisteix en afegir un tractament addicional a l'entrada per reconvertir la resta en una suma. Tal i com veiem a la subrutina resta, comprovem el signe dels operands i, idènticament a allò realitzat a la tasca anterior, mirem què hem de fer: si els operands tenen signe diferent, podrem simplement sumar-los com si fossin quantitats positives, predeterminant el signe del resultat// si, en canvi, tenen el mateix signe, aleshores determinarem quin és més gran i farem la resta de nombres positius amb el primer operand el més gran,com hem fet abans.

És evident que aquestes funcionalitats ja les hem implementat a la subrutina suma, així que la subrutina resta l'únic que fa és determinar en quin dels 4 casos estem (resta de positius, de negatius, de positiu i negatiu, o viceversa) i salta a diferents parts de la subrutina sumao crida directament a sumar, enfunció de què s'ha de fer.

Passem ara a respondre a les qüestions del guió:

Feu servir els adreçaments directe i indirecte i indiqueu on els tenim
El codi és una simple extensió de l'exercici anterior. Només estan marcades en vermell, verd i blau de la mateixa forma que ho hem fet en l'exercici anterior. Aquelles instruccions que no emprin cap d'aquests adreçaments però s'hagin afegit estaran en taronja.

Com gestioneu el problema del signe? I el problema del carry?
El problema del signe el tractem de la mateixa manera que al programa anterior, amb els punters sabem si el nombre és negatiu o no i predeterminem el signe del resultat en conseqüència. Pel *carry*, en canvi, succeeix quelcom similar que amb el problema de l'overflow, per com hem implementat el programa, no ens és cap molèstia,tret de considerar dos casos a l'hora de sumar/restar xifres, ja que hem de tenir en compte aquesta unitat addicional.

3 Ensamblant el codi

A partir dels codis generats anteriorment, hem de fer un programa capaç de fer sumes, restes, AND's i OR's. Ja hem vist com fer les primeres operacions, així que veurem com implementem les últimes dues i, a més, com les combinem en un sol programa. Cal destacar que en aquest apartat sí controlem l'entrada, no permetem a l'usuari entrar més de 3 xifres, ni escriure dos seguits, etc. Els únics dos signes que poden anar seguits és el de l'operació (+,-,,|) i el signe negatiu del segon operand.

Així doncs, la importància d'aquest programa radica en com és possible treballar amb més de 8 bits (999 > 255 = FFh) quan aquesta és la capacitat màxima dels nostres registres al simulador? Vegem el codi (de la mateixa manera que abans, en taronja apareixen les línies modificades):

```
1 .define
2   op_count    4 // Nombre de signes d'operacio
3   neq_count 14 // Nombre de caracters permesos sense el =
4   num_p_one 11 // Nombre de caracters sumant els numeros + 1
```

```
  num_count 10 // NÂº nombres del 0 al 9
  one        1 // Nombre 1
  error      5 // Nombre lletres paraula ERROR

.data 00h    // Caracters Permesos: Nombres del 0 al 9, +,-,&,|,=
  equal:    db 3Dh              // Signe =
  nums:     db 30h,31h,32h,33h,34h,35h,36h,37h,38h,39h  // Nombres positius
  op_signs: db 2Dh,2Bh,26h,7Ch          // Signes -,+,&,|
  er_chars: db 52h,4Fh,52h,52h          // Paraula RROR (ordre invers)
  fir_char: db 45h              // E, apunta a inici paraula

.data 150h
  counter:  db 00h              // Comptador de xifres resultat
  resultat: db 00h,00h,00h,00h          // Resultat operació
  op1:      db 00h,00h,00h
      op2:      db 00h,00h,00h
  MS1:      db 00h
  MS2:      db 00h
  num_768:  db 37h,36h,38h
  num_512:  db 35h,31h,32h
  num_256:  db 32h,35h,36h
  num_0:    db 30h,30h,30h

.org 200h
// Programa Principal
  lxi H, pila   // Punter de pila apuntant a 100h
  sphl
  mvi H, E0h    // Parell HL apuntant
  mvi L, 00h    // a la memoria de text
  mvi D, FFh    // offset primera xifra 1r operand (evitem errors)
  mvi E, 00h    // offset signe d'operació
  mvi B, 00h    // offset primera xifra 2n operand
  mvi C, 00h    // offset signe igual
  mvi A, 00h    // Reiniciem l'acumulador (no sabem que hi ha!)
  add D     // Reiniciem el flag de zero si estava activat

bucle:      // Loop infinit fins que entrem =
  jnz bucle

  call final    // Subrutina per operar i mostrar el resultat
  hlt     // El programa finalitza

.org 0024h      // Direcció de interrupció TRAP
  call string_in  // Crida a subrutina d'introduccio
  ret       // de dades per consola

.org 100h       // Posicio Pila
  pila:

.org 300h
// Rutina captura i mostra
string_in:
  in 00h    // Port d'entrada
  cpi 00h     // Si no hi ha caracter introduÃ¯t, surt
  jz no_tecla   // Si hi ha, escriu-lo per pantalla
tecla:
  call check_allowed // Caracter permes? Si no 00h a acumulador
  cpi 00h
  jz no_tecla    // Escriu-lo si esta permes
  call pointers  // Actualitzem els punters a memoria
  mov M,A
  inx H
```

```
67 no_tecla:
68   cpi 3Dh    // Mirem si hem d'operar (carÃcter = i és vÃlid)
69   ret
70
71 check_allowed:     // Subrutina control caracters
72   push D
73   push H
74   call get_allowed  // Determinem quins caracters es poden entrar
75 allowed_loop:      // Comprova si el caracter esta a
76   cmp M      // la llista de caracters permesos
77   jz end_allowed  // Esta permes: no modificar
78   inx H
79   dcr E
80   jnz allowed_loop
81   mvi A,00h    // No permes: posar 00h a acumulador
82 end_allowed:
83   pop H
84   pop D
85   ret
86
87 get_allowed:     // Subrutina per saber que podem entrar
88   push PSW     // Guardem l'acumulador i el registre F
89   mov A,L    // S'ha introduÃ¯t algun caracter?
90   cpi 00h
91   jnz op1_state // Sí? Busquem l'estat
92   mvi E,num_p_one    // No? Només podem entrar o número o -
93   lxi H,nums
94   jmp end_state
95 op1_state:
96   mov A,D    // Hem entrat la xifra 1r operand?
97   cpi FFh
98   jnz op1_in_state  // Sí? Busquem l'estat
99   mvi E,num_count    // No? Hem d'entrar algun número!
100  lxi H,nums
101  jmp end_state
102 op1_in_state:
103  mov A,E    // Signe operació introduÃ¯t?
104  cpi 00h
105  jnz sign_in_state // Sí? Busquem l'estat
106  mov A,L    // No? Mirem si ja hem entrat 3 xifres!
107  sub D
108  cpi 03h
109  jz sign_state // Sí? Hem d'entrar el signe de l'operació!
110  mvi E,neq_count    // No? Podem entrar qualsevol caracter menys =
111  lxi H,nums
112  jmp end_state
113 op2_state:
114  mov A,E    // Hem entrat quelcom darrere el signe d'operació?
115  inr A
116  cmp L
117  jnz op2_in_state  // Sí? Hem d'entrar algun número!
118  mvi E,num_p_one // No? Podem entrar o número o -
119  lxi H,nums
120  jmp end_state
121 op2_in_state:
122  mvi E,num_count // Només permetem números 0-9
123  lxi H,nums
124  jmp end_state
125 sign_state:
126  mvi E,op_count  // Només permetem +,-,&,|
127  lxi H,op_signs
128  jmp end_state
```

```
129 sign_in_state:
130   mov A,B    // Hem introduÃ¯t alguna xifra 2n operand?
131   cpi 00h
132   jz op2_state  // No? Hem entrat quelcom darrere el signe d'operació?
133   adi 03h      // Sí? Mirem si ja hem entrat 3 xifres!
134   cmp L
135   jz equal_state  // Sí? Només podem entrar =
136   mvi E,num_p_one // No? Podem entrar alguna xifra o =
137   lxi H,equal
138   jmp end_state
139 equal_state:
140   mvi E,one    // Només permetem =
141   lxi H,equal
142 end_state:
143   pop PSW    // Recuperem paraula estat programa!
144   ret
145
146 pointers:      // Subrutina per actualitzar punters
147   push PSW     // Evitem perdre el valor de l'acumulador
148   cpi 2Fh    // Carry 0 => [A] > 2Fh, Carry 1 => [A] < 2Fh
149   jnc num_equal_or  // Carry = 0 => Hem entrat número, = o |
150   mov A,D    // Mirem si el signe és un menys del 1r operand
151   cpi FFh
152   jz end     // Sí? No hem d'actualitzar cap punter
153   mov A,E    // No? Mirem si és signe de l'operació
154   cpi 00h
155   jnz end    // No? Ãs el signe del 2n operand, no actualitzem res
156   mov E,L    // Sí? Guardem l'offset del signe d'operació a E!
157   jmp end
158 num_equal_or:
159   cpi 3Dh    // Mirem si hem introduÃ¯t =
160   jnz num_or     // No? Hem entrat un número o |
161   mov C,L    // Sí? Guardem offset a C!
162   jmp end
163 num_or:
164   cpi 7Ch    // Mirem si hem introduÃ¯t |
165   jnz num    // No? Hem entrat un número
166   mov E,L    // Sí? Guardem l'offset del signe d'operació a E!
167   jmp end
168 num:
169   mov A,D    // Mirem si el número és la primera xifra del 1r operand
170   cpi FFh
171   jnz notD     // No? Mirem si hem d'actualitzar algun punter
172   mov D,L    // Sí? Guardem l'offset a D!
173   jmp end
174 notD:        // Mirem si el número és la primera xifra del 2n operand
175   mov A,E    // S'ha entrat el signe de l'operació?
176   cpi 00h
177   jz end     // No? Hem entrat una xifra qualsevol del 1r operand
178   mov A,B    // Sí? Actualitzem si no tenim cap xifra del 2n operand
179   cpi 00h
180   jnz end    // Hem entrat alguna? No actualitzem cap punter
181   mov B,L    // No? Guardem offset primer xifra 2n operand a H!
182 end:
183   pop PSW    // Recuperem l'acumulador i finalitzem la subrutina
184   ret
185
186 final:       // Subrutina per realitzar i mostrar l'operació
187   push H     // Guardem el punter a on hem d'imprimir per no modificar-lo
188   mov L,E    // Mirem el signe de l'operació
189   mov A,M
190   cpi 2Bh
```

```
191   jz suma    // Si és +, fem una suma
192   cpi 2Dh
193   jz resta     // Si és -, fem una resta
194   cpi 26h
195   jz and_check  // Si és &, fem l'operació lógica AND
196 or_check:      // En cas contrari, fem l'operació lógica OR
197   call check     // Hem de comprovar si podem fer l'operació!
198   cpi 00h    // Podem fer-la?
199   jnz print_error // No? Mostrem l'error per pantalla!
200   call or    // Realitzem l'operació OR
201   jmp print    // Imprimim el resultat (positiu)^
202 and_check:
203   call check     // Hem de comprovar si podem fer l'operació!
204   cpi 00h    // Podem fer-la?
205   jnz print_error // No? Mostrem l'error per pantalla!
206   call and     // Realitzem l'operació AND
207   jmp print    // Imprimim el resultat (positiu)
208
209 suma:         // Sumem
210   call delete_zeros // Eliminem els possibles 0s a l'entrar el número
211   mov A,D    // Mirem si el primer operand és negatiu
212   cpi 00h
213   jnz sum_op1_neg // Ho és? Mirem el signe del segon operand
214   mov A,E    // No ho és? Ãs el segon operand negatiu?
215   inr A
216   cmp B
217   jnz sum_op2_neg // Sí? Tindrem operació +num1 + -num2
218   call sumar     // No? Tindrem operació +num1 + +num2
219   jmp print    // Imprimim el resultat (no negatiu)
220 sum_op1_neg:
221   mov A,E    // Mirem el signe del segon operand
222   inr A
223   cmp B
224   jnz sum_op_neg  // Ãs negatiu? Tindrem -num1 + -num2
225 min_plus:
226   call compare     // No ho és? Tindrem -num1 + +num2
227   cpi 30h    // Són iguals?
228   jz resul_zero    // Sí? El resultat sera 0!
229   cpi 00h    // No? Quin és més gran?
230   jnz resta_neg    // Si num1 > num2, restarem num1 - num2
231   call change_ops // Si num2 > num1, restarem num2 - num1
232 resta_pos:
233   call restar    // Restem
234   jmp print    // Imprimim el resultat (positiu)
235 resta_neg:
236   call restar    // Restem
237   jmp neg_print // Imprimim el resultat (negatiu)
238 sum_op2_neg:     // +num1 + -num2!
239   call compare  // Comparem operands
240   cpi 30h    // Són iguals?
241   jz resul_zero // Sí? El resultat sera 0!
242   cpi 00h    // No? Quin és més gran?
243   jnz resta_pos // Si num1 > num2, restarem num1 - num2
244   call change_ops // Si num2 > num1, restarem num2 - num1
245   jmp resta_neg
246 sum_op_neg:
247   call sumar     // Fem la suma com si fossin positius
248   jmp neg_print // Imprimim el resultat amb un menys!
249
250 resta:         // Restem
251   mov A,D    // Mirem si el primer operand és negatiu
252   cpi 00h
```

```
253   jnz res_op1_neg // Ho és? Mirem el signe del segon operand
254   mov A,E    // No ho és? Ã s el segon operand negatiu?
255   inr A
256   cmp B
257   jz sum_op2_neg  // No? Tindrem +num1 - +num2 = +num1 + -num2
258   call sumar     // Sí? Tindrem +num1 - -num2 (sumem!)
259   jmp print    // Imprimim el resultat (no negatiu)
260 res_op1_neg:
261   mov A,E    // Mirem el signe del segon operand
262   inr A
263   cmp B
264   jnz min_plus  // Ã s negatiu? Tindrem -num1 - -num2 = -num1 + +num2
265   call sumar     // Ã s positiu? Tindrem -num1 - +num2 = -num1 + -num2
266   jmp neg_print // Sumem i imprimim el resultat amb un menys!
267
268 resul_zero:
269   call keep_resul // Guardem el 0 i imprimim el resultat
270   jmp print
271 print_error:     // Mostrem un missatge d'error per pantalla
272   pop H
273   lxi B,fir_char  // Apuntem a la primera lletra del missatge
274   mvi D,error    // Comptador de 5 lletres (mostrarem ERROR)
275   jmp loop
276 neg_print:
277   pop H      // Recuperem l'adreÃ§a a on imprimir el resultat
278   mvi A,2Dh    // Imprimim el signe -
279   mov M,A
280   inx H      // Incrementem per seguir imprimint
281   jmp pos_print // Ara imprimim el nombre com si fos positiu
282 print:
283   pop H
284 pos_print:
285   lxi B,counter
286   ldax B
287   mov D,A    // Guardem al registre D el comptador de xifres
288   mov A,C
289   add D
290   mov C,A    // Apuntem amb BC a la primera xifra del resultat
291 deal_zeros:      // Tractem els possibles 0s a les primeres xifres
292   mov A,D
293   cpi one    // Mirem si només falta una xifra a considerar
294   jz loop    // Si és així, la imprimim
295   ldax B     // En cas contrari, mirem si la primera xifra és 0!
296   cpi 30h
297   jnz show     // Si no és zero, mostrem tot el resultat restant
298   dcr D      // Si és zero, no imprimirem aquesta xifra!
299   dcr C
300   jmp deal_zeros  // Seguim buscant possibles zeros a l'esquerra
301 loop:
302   ldax B
303 show:
304   mov M,A    // Imprimim una xifra
305   inx H      // Incrementem per seguir imprimint
306   dcr C
307   dcr D
308   jnz loop     // Seguim imprimint si no hem acabat
309   ret
310
311 delete_zeros:      // Subrutina per eliminar els zeros a l'entrada. El
312 delete_op1:      // nombre 001 passara a 1
313   mov A,D
314   inr A
```

```
315   cmp E
316   jz delete_op2    // Si el nombre té només una xifra, no fem res
317   mov L,D
318   mov A,M
319   cpi 30h
320   jnz delete_op2  // Si la primera xifra no és zero, no fem res
321   inr D      // Si la primera xifra és zero, incrementem punter D
322   jmp delete_op1  // Iterem per si tenim dos 0s no significatius
323 delete_op2:
324   mov A,B
325   inr A
326   cmp C
327   jz end_delete // Si el nombre té només una xifra, no fem res
328   mov L,B
329   mov A,M
330   cpi 30h
331   jnz end_delete  // Si la primera xifra no és zero, no fem res
332   inr B      // Si la primera xifra és zero, incrementem punter B
333   jmp delete_op2  // Iterem per si tenim dos 0s no significatius
334 end_delete:
335   ret
336
337 sumar:          // Subrutina per sumar dos nombres positius
338 sum_no_carry:       // Cas en que sumem dues xifres sense carry
339   dcr E      // Apuntem a les xifres menys
340   dcr C      // significatives per sumar
341   mov L,E
342   mov A,M    // Carreguem una a l'acumulador
343   mov L,C
344   add M      // Sumem les xifres
345   sui 30h    // Restem el +30h del codi ASCII!
346   cpi 3Ah    // Carry = 0 => Suma xifres >= 10, emportem una
347   jnc sum_deal  // Si? Restem 10 al resultat i n'emportem una
348 sum_no_deal:
349   call keep_resul // Guardem el resultat
350   mov A,E
351   cmp D      // Hem acabat de sumar xifres primer operand?
352   jz nadd_op1    // Sí? Mirem que passa amb el segon
353   mov A,C    // No? Mirem igualment que passa amb el segon!
354   cmp B
355   jnz sum_no_carry  // Si no hem acabat, seguim sumant (sense carry)
356 add_op:       // Si hem acabat, guardem les xifres restants
357   dcr E
358   mov L,E
359   mov A,M
360 add_resul:
361   call keep_resul // Guardem les xifres restants de l'operand!
362   mov A,E
363   cmp D
364   jnz add_op     // Si no hem acabat, seguim guardant
365   jmp nadd     // Si hem acabat, la suma esta completa
366 nadd_op1:
367   mov A,C
368   cmp B      // Hem acabat de sumar xifres segon operand?
369   jz nadd    // Sí? Hem acabat de sumar!
370   mov E,C    // No? Passem les dades al parell de registres DE
371   mov D,B    // i usem el bucle add_op ja programat per guardar
372   jmp add_op     // les xifres restants
373 sum_carry:        // Cas en que sumem dues xifres amb carry
374   dcr E      // Apuntem a les xifres menys
375   dcr C      // significatives per sumar
376   mov L,E
```

```
377   mov A,M    // Carreguem una a l'acumulador
378   mov L,C
379   add M      // Sumem les xifres
380   sui 2Fh    // Restem el +30h del codi ASCII pero tenim +1 de carry!
381   cpi 3Ah    // Carry = 1 => Suma xifres < 10, no emportem res
382   jc sum_no_deal  // Seguim la suma sense tractar el carry
383 sum_deal:      // Tractem el carry al sumar xifres
384   sui Ah     // Restem 10 al resultat ja que ens emportem una!
385   call keep_resul // Guardem el resultat
386   mov A,E
387   cmp D      // Hem acabat de sumar xifres primer operand?
388   jz c_nadd_op1    // Sí? Mirem que passa amb el segon
389   mov A,C    // No? Mirem igualment que passa amb el segon!
390   cmp B
391   jnz sum_carry // Si no hem acabat, seguim sumant (amb carry)
392 c_add_op:      // Si hem acabat, sumem les xifres restants amb el carry
393   dcr E
394   mov L,E
395   mov A,M    // Carreguem la xifra
396   inr A      // Sumem el carry!
397   cpi 3Ah    // Carry = 1 => Suma xifres < 10, no emportem cap
398   jc add_resul  // No tractem més carry? Afegim les xifres restants!
399 deal_c_op:       // Tractem el carry de la suma operand + 1
400   sui Ah     // Restem 10 al resultat ja que ens emportem una!
401   call keep_resul // Guardem el resultat
402   mov A,E
403   cmp D      // Hem acabat de sumar xifres?
404   jz add_carry  // Sí? Falta la que ens emportem!
405   jmp c_add_op  // No? Seguim sumant xifres amb carry!
406 c_nadd_op1:
407   mov A,C    // Veiem si ja hem acabat de sumar el segon operand
408   cmp B
409   jz add_carry  // Si hem acabat, falta la que ens emportem!
410   mov E,C    // Si no hem acabat, passem les dades al parell de
411   mov D,B    // registres DE i usem el bucle c_add_op ja programat
412   jmp c_add_op  // per guardar les xifres restants
413 add_carry:
414   mvi A,31h    // Guardem al resultat la que ens emportavem
415   call keep_resul
416 nadd:
417   ret
418
419 restar:      // Subrutina per restar dos nombres positius num1 > num2
420 res_no_carry:      // Cas en que restem dues xifres sense carry
421   dcr E      // Apuntem a les xifres menys
422   dcr C      // significatives per restar
423   mov L,E
424   mov A,M    // Carreguem una a l'acumulador
425   adi 30h    // Sumem el +30h del codi ASCII!
426   mov L,C
427   sub M      // Restem les xifres
428   cpi 30h    // Carry = 0: 0 <= Resta xifres < 9, no emportem cap
429   jc res_deal    // Ens emportem? Sumem 10 al resultat i n'emportem una
430 res_no_deal:
431   call keep_resul // Guardem el resultat
432   mov A,C
433   cmp B      // Hem acabat de restar xifres segon operand?
434   jnz res_no_carry  // Si no hem acabat, seguim restant (sense carry)
435   mov A,E    // Si hem acabat, mirem si hem restat totes les xifres del primer
436   cmp D
437   jnz add_op     // Si no hem acabat, afegim les xifres restants!
438   jmp nsub     // Si hem acabat, finalitzem l'operació
```

```
439 res_carry:      // Cas en que restem dues xifres amb carry
440   dcr E     // Apuntem a les xifres menys
441   dcr C     // significatives per restar
442   mov L,E
443   mov A,M   // Carreguem una a l'acumulador
444   adi 2Fh   // Sumem el +30h del codi ASCII pero tenim -1 de carry!
445   mov L,C
446   sub M     // Restem les xifres
447   cpi 30h   // Carry = 0: 0 <= Resta xifres < 9, no emportem cap
448   jnc res_no_deal // No ens emportem? Tornem al cas sense carry!
449 res_deal:
450   adi Ah    // Sumem 10 al resultat ja que ens emportem una!
451   call keep_resul // Guardem el resultat
452   mov A,C
453   cmp B     // Hem acabat de restar xifres segon operand?
454   jz sub_carry  // Sí? Com tenim carry no podem haver acabat de restar!
455   jmp res_carry // No? Seguim restant (amb carry)
456 sub_deal:     // Tractem carry de la resta xifra primer operand - carry
457   adi Ah    // Sumem 10 ja que ens emportem una!
458   call keep_resul // Guardem el resultat
459   // Tenint carry no podem haver acabat la resta! Seguim propagant-lo
460 sub_carry:
461   dcr E
462   mov L,E
463   mov A,M   // Carreguem xifra a l'acumulador
464   dcr A     // Restem el carry
465   cpi 30h   // Carry = 0: 0 <= Resta xifres < 9, no emportem cap
466   jc sub_deal   // Ens emportem? Sumem 10 al resultat i n'emportem una
467   call keep_resul // No ens emportem? Guardem i mirem si hem acabat la resta!
468   mov A,E
469   cmp D
470   jnz add_op    // Si no hem acabat, afegim les xifres restants
471 nsub:
472   ret     // Si hem acabat, finalitzem la subrutina
473
474 compare:      // Subrutina per comparar operands (A = 1: num1 > num2,
475   call delete_zeros // Eliminem els zeros no significatius abans de comparar!
476   push D      // A = 0: num2 > num1, A = 30h: num1 = num2)
477   push B      // Guardem els offsets abans de fer possibles modificacions
478   mov A,E
479   sub D
480   mov L,A
481   mov A,C
482   sub B
483   cmp L     // Mirem quin nombre té més xifres!
484   jnz dif_xifres  // Diferents xifres? Un és més gran!
485 cmp_xifra:      // Si tenen les mateixes, hem de comparar-los xifra a xifra
486   mov L,B   // Guardem primera xifra segon operand a acumulador
487   mov A,M
488   mov L,D   // Comparem amb la primera xifra primer operand
489   cmp M
490   jnz dif_xifres  // Si les xifres són diferents, un és més gran!
491   inr D     // Si són iguals, mirem si tenen més xifres!
492   inr B
493   mov A,D   // Com tenen el mateix nombre de xifres, només mirem si
494   cmp E     // hem comparat totes les del primer operand
495   jnz cmp_xifra // Si no hem acabat, continuem
496   mvi A,30h   // En cas contrari, els nombres són iguals (valor absolut)!
497   jmp end_compare
498 dif_xifres:
499   jnc gran_segon  // Si carry = 0 => 2n operand té més xifres!
500   mvi A,one   // En cas contrari, el primer operand és més gran
```

```
501   jmp end_compare
502 gran_segon:
503   mvi A,00h    // El segon operand és més gran!
504 end_compare:
505   pop B      // Recuperem el valor dels offsets i finalitzem la subrutina!
506   pop D
507   ret
508
509 change_ops:      // Subrutina per permutar els operands
510   push H     // Usem HL com registres auxiliars
511   mov H,D
512   mov L,E
513   mov D,B
514   mov E,C
515   mov B,H
516   mov C,L
517   pop H
518   ret
519
520 keep_resul:      // Subrutina per guardar el resultat d'una xifra a memoria
521   push H
522   push PSW     // Guardem parells de registres a la pila
523   lxi H,counter // Apuntem amb HL al comptador de xifres del resultat
524   inr M      // Sumem 1 a aquest comptador
525   mov A,L    // Fiquem a l'acumulador la part baixa de l'adreÃ§a
526   add M      // A acumulador esta la part baixa adreÃ§a on hem de guardar!
527   mov L,A
528   pop PSW
529   mov M,A    // Guardem el resultat de l'operació
530   pop H      // Recuperem la informació guardada a la pila i retornem
531   ret
532
533 check:       // Subrutina per saber si podem realitzar una operació logica
534   mov A,D    // Acumulador = 0 => Podem, no tenim cap operand negatiu!
535   cpi 00h
536   jnz no_logic  // Si el primer operand és negatiu, no fem l'operació!
537   mov A,E    // Si no ho és, mirem el segon operand
538   inr A
539   cmp B
540   jnz no_logic  // Si és negatiu, no farem l'operació!
541   mvi A,00h    // Farem l'operació!
542 no_logic:
543   ret
544
545 and:         // Subrutina per fer num1 AND num2 amb num1,num2 no negatius
546   call pre_process  // Obtenim els operands, un al registre C i l'altre a ACC
547   ana C      // Realitzem l'operació AND
548   call process_res  // Processem el resultat
549   lxi H,MS1
550   mov A,M    // Carreguem a l'acumulador la part alta del primer operand
551   inr L
552   ana M      // Fem l'operació amb la part alta dels operands
553   call process_MS // Processem la part alta de l'operació
554   ret      // Hem realitzat l'operació AND correctament
555
556 or:        // Subrutina per fer num1 OR num2 amb num1,num2 no negatius
557   call pre_process  // Obtenim els operands, un al registre C i l'altre a ACC
558   ora C      // Realitzem l'operació AND
559   call process_res  // Processem el resultat
560   lxi H,MS1
561   mov A,M    // Carreguem a l'acumulador la part alta del primer operand
562   inr L
```

```
563   ora M      // Fem l'operació amb la part alta dels operands
564   call process_MS // Processem la part alta de l'operació
565   ret      // Hem realitzat l'operació AND correctament
566
567 pre_process:     // Subrutina per processar operands abans de l'operació logica
568   call process_ops  // Obtenim operables amb 8 bits
569   call read_op  // Carreguem el primer operand a l'acumulador
570   mov E,C
571   mov C,A
572   mov D,B    // Guardem a C el primer operand i apuntem amb DE al segon
573   call read_op  // Carreguem el segon operand a l'acumulador
574   ret
575
576 process_ops:
577   push D     // Usem el parell DE per guardar dades temporalment
578   mov D,B
579   mov E,C
580   lxi B,op2    // Apuntem a on volem el segon operand
581   call move_op  // Movem l'operand apuntat per DE a la posició desitjada
582   call process_op // Eliminem la part alta del segon operand!
583   lxi H,MS1
584   mov A,M
585   inr L
586   mov M,A    // Guardem aquesta part com part alta del segon operand
587   lxi B,op2    // Apuntem a la posició desitjada del resultat
588   call move_res // Movem el resultat a la posició especificada!
589   pop D      // Recuperem punters al primer operand
590   lxi B,op1
591   call move_op  // Movem el primer operand a la posició especificada
592   call process_op // Eliminem la part alta del primer operand!
593   lxi B,op1    // Tornem l'operand a la posició especificada!
594   call move_res // Movem l'operand (actualitzat sense els bits superiors)
595   lxi H,op1    // Posicionem punters als operands
596   mov D,L    // Primera xifra primer operand
597   lxi H,op2
598   mov E,L    // Posició després darrera xifra primer operand
599   mov B,L    // Primera xifra segon operand
600   lxi H,MS1
601   mov C,L    // Posició després darrera xifra segon operand
602   ret
603
604 process_op:      // Subrutina per tractar els dos MS bit d'un operand
605   lxi H,counter // Apuntem a la part de dades de la memoria
606   call check_op // Mirem com de gran és l'operand
607   lxi H,MS1    // Guardem els bits més significatius de l'operand
608   mov M,A
609   cpi 03h    // Ã s major o igual a 768?
610   jz sub_op_768 // Sí? Li restem 768!
611   cpi 02h    // No? Ã s major o igual a 512?
612   jz sub_op_512 // Sí? Li restem 512!
613   cpi 01h    // No? Ã s major o igual a 256?
614   jz sub_op_256 // Sí? Li restem 256!
615   lxi H,num_0    // No? Restarem a l'operand 0
616   jmp sub_op
617 sub_op_768:
618   lxi H,num_768 // Restarem a l'operand el nombre 768
619   jmp sub_op
620 sub_op_512:
621   lxi H,num_512 // Restarem a l'operand el nombre 512
622   jmp sub_op
623 sub_op_256:
624   lxi H,num_256 // Restarem a l'operand el nombre 256
```

```
625 sub_op:
626   mov A,L
627   mov B,A
628   adi 03h
629   mov C,A    // Els punters BC estan per fer l'operació amb el nombre!
630   call restar    // Restem a l'operand el nombre especificat
631   ret
632
633 check_op:      // Subrutina per comprovar si un operand es representa amb 9 o 10 bits:
634   mov A,E    // A <= 03h, op >= 768       , A <= 02h, 768 > op >= 512
635   sub D      // A <= 01h, 512 > op >= 256, A <= 00h, 256 > op,8 bits!
636   cpi 03h    // Té l'operand 3 xifres?
637   jnz op_less_256 // No les té? Ã s menor que 256!
638   mov L,D    // Les té? Mirem si és major a 768!
639   mov A,M    // Carreguem primera xifra operand
640   cpi 37h    // Mirem si el nombre és major a 768!
641   jz op_7c     // La xifra de les centenes és 7?
642   jnc op_grea_768 // El nombre és major a 768?
643   cpi 35h    // No? Comparem amb 512!
644   jz op_5c     // La xifra de les centenes és 5?
645   jnc op_grea_512 // El nombre és major a 512?
646   cpi 32h    // No? Comparem amb 256!
647   jz op_2c     // La xifra de les centenes és 2?
648   jnc op_grea_256 // El nombre és major a 256?
649   jmp op_less_256 // No? Podem representar-lo amb 8 bits!
650 op_7c:
651   inr L      // Apuntem a la xifra de les desenes!
652   mov A,M    // Comprovem si la segona xifra és major a 6
653   cpi 36h    // Carry = 0 => Xifra >= 6
654   jz op_6d     // Tenim un 6 a les desenes, és major a 768?
655   jnc op_grea_768 // El nombre és major a 768?
656   jmp op_grea_512 // No?
657 op_6d:
658   inr L      // Apuntem a la xifra de les unitats
659   mov A,M    // Comprovem si la darrera xifra es major a 8
660   cpi 38h    // Carry = 0 => Nombre >= 768 = 512 + 256
661   jc op_grea_512
662 op_grea_768:
663   mvi A,03h    // op >= 768 => Carreguem a acumulador 03h
664   jmp end_check
665 op_5c:
666   inr L      // Apuntem a la xifra de les desenes
667   mov A,M    // Comprovem si la segona xifra es major a 1
668   cpi 31h    // Carry = 0 => Xifra >= 1
669   jz op_6d     // Tenim un 1 a les desenes, es major a 512?
670   jnc op_grea_512 // El nombre es major a 512?
671   jmp op_grea_256
672 op_1d:
673   inr L      // Apuntem a la xifra de les unitats
674   mov A,M    // Comprovem si la darrera xifra és major a 2
675   cpi 32h    // Carry = 0 => Nombre >= 512
676   jc op_grea_256
677 op_grea_512:
678   mvi A,02h    // 768 > op >= 512 => Carreguem a acumulador 02h
679   jmp end_check
680 op_2c:
681   inr L      // Apuntem a la xifra de les desenes!
682   mov A,M    // Comprovem si la segona xifra es major a 5
683   cpi 35h    // Carry = 0 => Xifra >= 5
684   jz op_5d     // Tenim un 5 a les desenes, es major a 256?
685 op_5d:
686   inr L      // Apuntem a la xifra de les unitats!
```

```
687   mov A,M    // Comprovem si la darrera xifra és major a 6
688   cpi 36h    // Carry = 0 => Nombre >= 256!
689   jc op_less_256
690 op_grea_256:
691   mvi A,one    // 512 > op >= 256 => Carreguem a acumulador 01h
692   jmp end_check
693 op_less_256:
694   mvi A,00h    // 256 > op => Carreguem a acumulador 00h
695 end_check:
696   ret
697
698 move_op:       // Subrutina per portar operand apuntat per DC a la direcció
699         // desitjada, apuntada pel parell BC
700   lxi H,E000h    // Apuntem a la pantalla de text
701   mov L,D
702   mov A,E
703   sub D      // Comptador de xifres
704 zero_loop:
705   cpi 03h
706   jz move_op_loop    // Mourem les xifres que tinguem (si no tenim centenes, nomes dues)
707   mov D,A    // En cas contrari incrementem l'acumulador i el punter BC!
708   mvi A,30h
709   stax B     // Guardem un zero a la posicio apuntada per BC
710   inr C
711   mov A,D
712   inr A
713   jmp zero_loop
714 move_op_loop:
715   mov A,M
716   stax B     // Movem una de les xifres
717   inr C
718   inr L
719   mov A,L
720   cmp E      // Si L = E, hem mogut tot l'operand
721   jnz move_op_loop   // Seguim movent xifres
722   mov E,C    // Apuntem a posicio darrere operand amb E
723   mov A,E
724   sui 03h
725   mov D,A    // Apuntem a primera xifra amb D
726   ret
727
728 move_res:      // Subrutina per portar el nou operand, guardat al resultat,
729   lxi H,counter // a la direcció desitjada, apuntada pel parell BC
730   mvi E,00h
731   mov M,E    // Reiniciem el comptador
732   mvi D,03h    // Mourem 3 xifres
733   mov A,L
734   add D
735   mov L,A    // Apuntem amb HL a la primera xifra del nou operand
736 move_res_loop:
737   mov A,M
738   mov M,E    // Reiniciem la posicio de memoria
739   stax B     // Movem una de les xifres
740   inr C
741   dcr L
742   dcr D
743   jnz move_res_loop // Si no hem acabat, seguim movent xifres
744   ret
745
746 read_op:       // Subrutina per passar un nombre de 3 xifres apuntat per DE a ACC
747   mov L,D
748   mvi D,02h    // Iteracions bucle
```

```
749   mov A,M    // Carreguem la primera xifra a ACC
750   sui 30h    // Restem el +30h del codi ASCII
751 read_loop:
752   call mul_10    // x10 acumulador
753   inr L
754   add M      // Sumem xifra a ACC
755   sui 30h    // Restem el +30h del codi ASCII
756   dcr D
757   jnz read_loop
758   ret      // El nombre es troba correctament a l'acumulador
759
760 mul_10:      // Subrutina per multiplicar per 10 el contingut de l'acumulador
761   push D     // Usem els registres DE per operar
762   mvi D,09h    // Comptador de sumes consecutives
763   mov E,A    // Guardem el terme a multiplicar al registre E
764 mul_loop:
765   add E
766   dcr D
767   jnz mul_loop   // Sumem 9 cops més el terme: multipliquem per 10!
768   pop D
769   ret
770
771 process_res:     // Subrutina per passar el resultat d'un nombre menor de
772   lxi H,op1    // 256 a tres xifres decimals, passar-lo de l'acumulador a memoria
773   mov B,L    // Apuntem amb B a la primera xifra del resultat
774   mvi D,30h
775   mvi E,64h    // D és el comptador de xifres, E el nombre a comparar
776   call keep_digit // Guardem les centenes del resultat
777   mvi D,30h
778   mvi E,0Ah
779   call keep_digit // Guardem les desenes del resultat
780   adi 30h    // Convertim les unitats a ASCII
781   mov M,A
782   inr L
783   mov C,L    // Apuntem amb C a la posiciO consecutiva de la darrera xifra
784   ret
785
786 keep_digit:      // Subrutina per comptar quantes centenes/desenes tE un nombre
787 digit_loop:
788   cmp E
789   jc end_digit   // Si nombre < 100, passem a les desenes
790   inr D      // Sumem una centena/desena
791   sub E      // Restem i repetim
792   jmp digit_loop
793 end_digit:
794   mov M,D    // Guardem a memOria i apuntem a la segÃ¼ent xifra
795   inr L
796   ret
797
798 process_MS:      // Subrutina per processar el resultat de la part alta de
799   cpi 03h    // l'operaciO i combinar-lo amb la part baixa
800   jz add_768     // Sumem 768?
801   cpi 02h
802   jz add_512     // Sumem 512?
803   cpi 01h
804   jz add_256     // Sumem 256?
805   lxi H,num_0    // No? Sumarem al resultat 0
806   jmp add_res
807 add_768:
808   lxi H,num_768 // Sumarem al resultat el nombre 768
809   jmp add_res
810 add_512:
```

```
811   lxi H,num_512 // Sumarem al resultat el nombre 512
812   jmp add_res
813 add_256:
814   lxi H,num_256 // Sumarem al resultat el nombre 256
815 add_res:
816   mov A,L
817   mov D,A
818   adi 03h
819   mov E,A    // Els punters DC estan per fer l'operaciO amb el resultat!
820   call sumar     // Sumem al resultat el nombre especificat
821   ret
```

Abans de veure quines noves funcionalitats implementa aquest, hem de tenir molt present com adrecem els operands: el registre H serveix per apuntar als primers dos *bytes* de la memòria (part alta), els registres D i E guarden la part baixa respecte el registre H del primer terme del primer operand i la posició consecutiva al darrer, respectivament. El mateix passa amb B i C. D'aquesta manera, movent el contingut d'aquests registres a L, podem treballar amb els caràcters guardats a memòria.

La primera modificació del codi és la definició de diferents constants a l'inici del programa (ens serviran pel control d'entrada per teclat i per mostrar el missatge d'error en cas que es vulgui realitzar una operació lògica amb nombres negatius). Veiem com, seguidament, guardem els codis ASCII dels caràcters permesos a memòria (en aquest ordre per després usar la subrutina de la part guiada, check_allowed,simplement canviant la quantitat de caràcters permesos i l'inici d'aquests).

Notem que definim una segona part de memòria de dades, la separem perque estem reservant 45 *bytes* de memòria, és a dir, arribaríem fins la posició 002Eh, escrivint a sobre de la zona de memòria reservada per la interrupció TRAP. Tampoc podem guardar-ho aprop de la 100h, ja que és la memòria reservada per la pila. LLavors guardem aquestes segones dades a partir de la direcció 150h (el comptador de xifres del resultat, les xifres en ASCII del resultat, etc.). Estem reservant memòria per guardar els operands (un cop entrats els processarem per tal d'eliminar la part alta d'aquests (els reduirem a un nombre amb el qual es pugui treballar a l'acumulador. D'aquesta manera, agrupant els bits dels operands de 8 en 8 podríem realitzar operacions lògiques d'un nombre arbitrari de bits. Aquests bits addicionals els guardarem a MS1 i MS2 (**Most Significant** bits dels operands 1 i 2). Cal destacar que els nombres 768, 512, 256 i 0 serviran per restar-los als operands i convertir-los a nombres representables amb 8 bits.

Una diferència important són les crides a les subrutines final, que, com les subrutines suma i resta dels anteriors exercicis, operarà i mostrarà el resultat (amb la millora que mirarà quin és el signe de l'operació per tal de realitzar la corresponent) i get_allowed (dins de la subrutina de la part guiada check_allowed), per tal de controlar l'entrada per teclat.

Els canvis a la subrutina final són senzills: per mitjà d'aquest punter al símbol de l'operació, determinem quina hem de fer i, un cop determinada, saltem als blocs que realitzen l'operació. Tenim afegits els blocs or_check i and_check, que comproven que els nombres siguin no negatius.

Anteriorment hem afirmat que aquest programa elimina l'error a l'operar amb nombres amb 0s no significatius com, per exemple, 001. Això era degut a que la subrutina *compare*, que determinava quin nombre era més gran en valor absolut, comptava el nombre de xifres per predeterminar, en cas que es pogués, quin dels dos nombres era més gran. Per tant, per evitar aquest error, abans de comparar-los hem d'eliminar els 0sa l'esquerra. Aquesta és la funció de la subrutina delete_zeros.

Ara veurem les addicions al codi. Començarem amb la subrutina encarregada de determinar si podem realitzar una operació lògica. Com ja hem comentat, simplement comprova si algun dels dos nombres és negatiu. Si ambdós no ho són, permetrà realitzar l'operació, en cas contrari, mostrarem el missatge d'error.

Passem ara a la implementació de les operacions lògiques. És important tenir en compte diversos factors.

Per poder treballar amb l'acumulador, separarem els bits més significatius (9è i 10è bit) dels nombres i els treballarem per separat. Per fer-ho compararem els operands amb 768, 512 i 256, els valors que poden fer que aquests bits siguin 1.
Un cop determinats, guardarem la part alta a memòria, convertirem els operands a nombres representables en 8 bits i, aleshores, els passarem a binari, carregant un operand al registre C i un altre al registre A, per poder fer qualsevol de les dues operacions AND o OR.
Un cop operada la part baixa dels operands, caldrà convertir el resultat novament a decimal, d'això s'ocuparà la subrutina process_res. Finalment, operarem amb els bits més significatius i, mitjançant la subrutina process_MS, afegirem 768, 512, 256 o 0 al resultat temporal obtingut per tenir el definitiu. Com totes les sumes i restes es guarden a la part de memòria etiquetada com resultat, podrem imprimir-ho directament amb els blocs de print que ja tenim.

Amb aquest resum ja hem tractat les subrutines and i or, que fan exactament el mateix canviant les instruccions on es realitzen operacions lògiques (a cada subrutina es realitza l'operació en qüestió). Veiem com "preprocessem" les dades. Aquesta funcionalitat és la que més subrutines involucra: pre_process, process_ops, process_op, check_op, move_op, move_res, read_op i mul_10.La primera realitza subcrides: primer separem els operands en part alta i baixa i després carreguem aquestes parts baixes en binari als registres A i C. La segona és més específica: realitza la conversió a nombres de menys de 9 bits dels nombres que es troben a la pantalla de text, entrats per l'usuari, i guarda les parts alta i baixa, per separat, a memòria. Això ho fa amb una crida a move_op, que copia l'operand de la pantalla de text apuntat per DE a la posició de memòria especificada per BC, ja sigui op1 o op2 (on hi van els operands). Un cop mogut l'operand, el processem: fem comparacions consecutives amb els nombres 768, 512 i 256 (si cal xifra a xifra) i, un cop sabem quins seran els bits de la part alta, els guardem i restem a l'operand la quantitat que cal (per exemple, al nombre 800 se li haurà de restar 41768, donant com a nou operand 32 i com a bits de part alta 11, ja que la descomposició en sumes de potències de 2 del 800 és 512 + 256 + 32). Com al restar,l'operand es troba a la posició de memòria resultat, l'hem de portar a op1 o op2 (en el cas que sigui el primer o el segon operand), això s'aconsegueix mitjançant la subrutina move_res.

Un cop tenim els operands modificats amb la part alta guardada a memòria, ens trobem a la segona instrucció de la subrutina pre_process, ara carreguem l'operand apuntat per DE a l'acumulador (subrutina read_op), el passem a C, carreguem l'altre i ja estarem preparats per realitzar l'operació lògica.Com aconseguim convertir l'operand representable en 8 bits a l'acumulador? La resposta és senzilla: carreguem la xifra de les centenes, multipliquem per 10// li sumem la de les desenes, multipliquem per 10 i, finalment, sumem la de les unitats. L'explicació anterior es basa en el fet que un nombre a b c en base decimal és el nombre $100a + 10b + c = c + (10 \cdot (b + 10a))$.Si hem tingut en compte el +30 del codi ASCII en cada pas, tindrem l'operand perfectament carregat a l'acumulador (veure subrutines read_op i mul_10). Finalment, només queda veure com processem aquest resultat i com el combinem amb l'operació de la part alta. L'estratègia és senzilla, comparem el contingut de l'acumulador amb els nom-

bres 100 = 64h i 10 = Ah iteradament. Si l'acumulador és major o igual que 100, sumem 1 a les centenes i restem 100 al contingut de l'acumulador. D'aquesta manera, si tenim en compte que el 0 en ASCII és 30h, aconseguirem reconvertir el resultat en un nombre decimal representat en ASCII (codi implementat a les subrutines process_res i keep_digit).

Per combinar el resultat amb el de la part alta, seguim allò esmentat abans: determinem què hem de sumar al resultat que ja tenim (768, 512, 256 o 0) a partir dels MS bits acabats d'operar i, finalment, ho mostrem per pantalla.
Hem vist, doncs, una explicació detallada del codi d'aquest apartat, així com el flux d'execució a mesura que avança el programa.

Quina diferència hi ha entre la suma i la OR?
La diferència és que la OR és una operació lògica i la suma és una operació aritmètica.

La instrucció STA 1234h
Aquesta instrucció fa servir adreçament directe, ja que es guarda una còpia del contingut de l'acumulador a la posició de memòria especificada, 1234h.

3 CONCLUSIONS

Gràcies a la realització d'aquesta pràctica he pogut aplicar tots els coneixements relacionats amb l'assignatura d'Introducció als Ordinadors i, més concretament, del simulador i8085, i aplicar-los per a realitzar un programari complet de càlcul d'operacions aritmètiques i lògiques.

En aquesta pràctica s'ha proposat una solució al problema plantejat, que demanava realitzar un programa que realitzés les funcions d'una aplicació calculadora, amb les operacions de suma, resta, AND i OR.

- S'han repassat els coneixements obtinguts a teoria.
- S'ha realitzat els exercicis proposats a la pràctica.
- S'han respost les qüestions plantejades.
- S'ha treballat amb els dispositius d'entrada/sortida.
- S'ha treballat amb el funcionament de les interrupcions