

## cmp-32.asm – Analisi simulazione

a cura di Daniele Sana

Questo programma confronta due numeri interi a 32 bit in valore assoluto e salva il risultato del confronto in R2. Questo significa che al termine del programma:

- se R2 = -1 allora primo numero < secondo numero
- se R2 = 0 allora primo numero = secondo numero
- se R2 = +1 allora primo numero > secondo numero

Il testo del programma principale è il seguente:

```
.orig x3000
LD    R0,mswnum1
LD    R1,mswnum2
JSR   cmp16
AND   R2,R2,R2
BRP   pgt
BRN   plt
LD    R0,lswnum1
LD    R1,lswnum2
JSR   cmp16
BRP   pgt
BRN   plt
BRZ   equ
pgt   AND    R2,R2,#0
      ADD    R2,R2,#1
      TRAP   x25
equ   AND    R2,R2,#0
      TRAP   x25
plt   AND    R2,R2,#0
      ADD    R2,R2,#-1
      TRAP   x25

mswnum1    .blkw 1
lswnum1    .blkw 1
mswnum2    .blkw 1
lswnum2    .blkw 1

; qui va il codice della routine cmp16 riportato più in basso

.end
```

All'interno del programma principale viene inoltre eseguita una chiamata alla routine **cmp16** che permette di confrontare due numeri a 16 bit in valore assoluto. In tale routine R0 contiene il primo numero, R1 contiene il secondo, mentre il risultato del confronto viene salvato in R2:

- se R2 = -1 allora primo numero < secondo numero
- se R2 = 0 allora primo numero = secondo numero
- se R2 = +1 allora primo numero > secondo numero

Il codice della routine **cmp16** è il seguente:

```

cmp16 AND    R2,R2,#0
      AND    R0,R0,R0
      BRN    pneg
      AND    R1,R1,R1      ; qui primo numero positivo
      BRN    pgts          ; se secondo negativo, primo > secondo
      BRZP   conc          ; salta a esaminare numeri concordi
pneg  AND    R1,R1,R1      ; qui primo numero negativo
      BRZP   plts          ; se secondo positivo, primo < secondo
      BRN    conc          ; salta a esaminare numeri concordi
conc  NOT    R1,R1
      ADD    R0,R0,R1
      NOT    R0,R0
      BRN    pgts
      BRZ    peqs
      BRP    plts
plts  ADD    R2,R2,#-1
      RET
peq   RET
pgts  ADD    R2,R2,#1
      RET

```

Vediamo innanzitutto quali saranno gli effetti delle pseudo-istruzioni *Assembly* presenti nel codice:

- **.orig x3000** segnala all'*Assembler* che il programma deve essere caricato all'indirizzo x3000;
- le quattro pseudo-istruzioni **.blkw #1** riservano lo spazio necessario a contenere quattro parole di memoria, le cui label stanno ad indicare:
  - o mswnum1 → most significant word for num1
  - o lswnum1 → less significant word for num1
  - o mswnum2 → most significant word for num2
  - o lswnum2 → less significant word for num2

A questo punto, dopo aver compilato il programma, passiamo allo strumento *Simulate* ed effettuiamo il *debugging*, ossia eseguiamo il programma un'istruzione alla volta.

Per collaudare a fondo il programma supponiamo che sia num1 = 5 e num2 = 7: ci aspettiamo pertanto che, alla fine dell'esecuzione, il registro R2 contenga il valore -1.

In Simulate la situazione è la seguente:

The screenshot displays the LC2 Simulator interface. At the top is a menu bar with 'File', 'Options', 'Simulate', 'Display', and 'Help'. Below the menu is a toolbar with icons for file operations, simulation control, and navigation. The main window is divided into three sections:

- Registers:** A table showing the state of registers R0 through R7, PC, IR, and CC. R0-R7 contain hexadecimal values (mostly x0000), PC is x3000, IR is x0000, and CC is Z.
- Assembly Code:** A list of instructions from address x3000 to x3032. Instructions include LD, JSR, AND, BRP, BRN, LD, BRZ, AND, ADD, TRAP, BRNOP, and RET. Comments like 'pgt', 'equ', 'plt', 'conc', and 'peq' are present. A red box highlights the memory addresses x0000 through x0007, which correspond to variables 'mswnum1', 'lswnum1', 'mswnum2', and 'lswnum2'.
- Program Status:** At the bottom, it shows 'cmp-32.obj', '0 instructions executed', and 'Idle'.

Dalle ipotesi assunte per simulare il funzionamento del programma, sono state inizializzate tramite il comando *Simulate* → *Set Value* le quattro celle di memoria allocate dalle pseudo-istruzioni **.blkw** (rettangolo rosso):

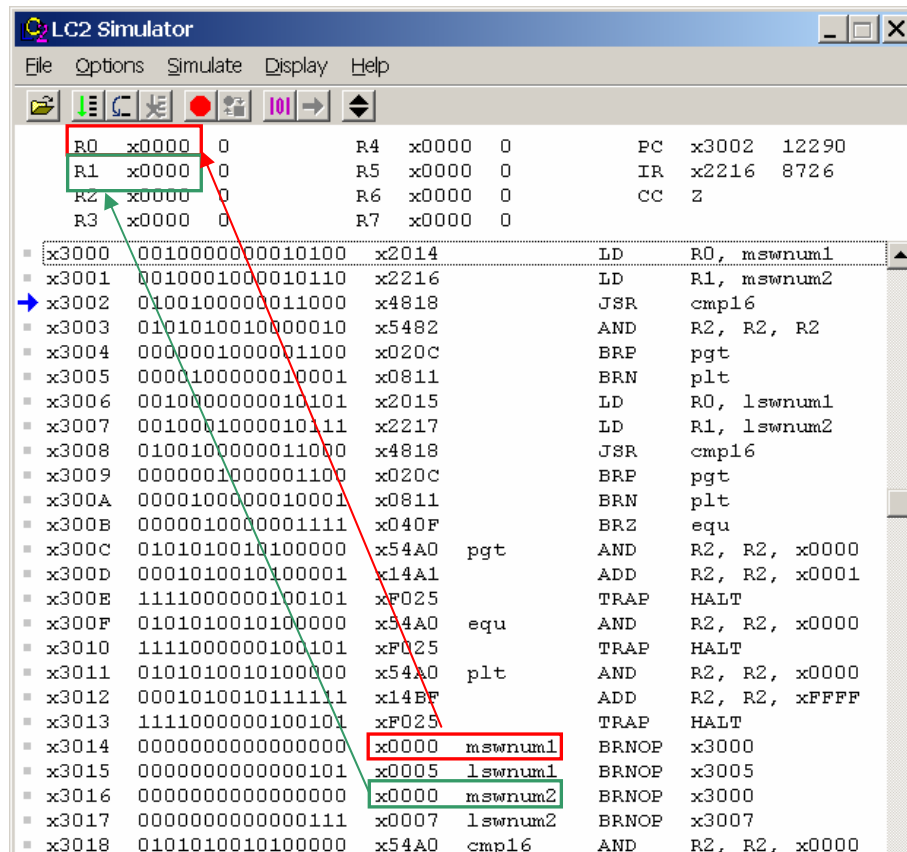
- mswnum1 = x0000
- lswnum1 = x0005
- mswnum2 = x0000
- lswnum2 = x0007

Inoltre, la tabella dei simboli generata dall'*Assembler* è la seguente:

```
// Symbol table
// Scope level 0:
//      Symbol Name      Page Address
//      -----
//      cmp16            3018
//      conc              3021
//      equ               300F
//      lswnum1           3015
//      lswnum2           3017
//      mswnum1           3014
//      mswnum2           3016
//      peqs              3029
//      pgt               300C
//      pgts              302A
//      plt               3011
//      plts              3027
//      pneg              301E
```

Facciamo ora partire il programma con il comando *Simulate* → *Step (F8)* e analizziamo ogni singola istruzione.

La prime due istruzioni **LD R0, mswnum1** e **LD R1, mswnum2** caricano rispettivamente nei registri R0 e R1 i 16 bit più significativi dei numeri da confrontare:



Il passo successivo consiste in una chiamata alla routine **cmp16**: per far questo viene utilizzata l'istruzione **JSR cmp16**, che salva il PC in R7 quindi punta all'indirizzo di memoria corrispondente all'inizio della routine **cmp16**. L'indirizzo salvato in R7 verrà successivamente utilizzato dall'istruzione **RET** per ripristinare il contenuto del PC dopo l'esecuzione della routine.

The screenshot shows the LC2 Simulator window. At the top, there's a menu bar (File, Options, Simulate, Display, Help) and a toolbar. Below, the register file shows R0-R7 with values. R7 is highlighted with a red box. The PC (Program Counter) is highlighted with a green box and shows the value x3018. The instruction stream shows various instructions. A green arrow points from the PC register to the instruction at address x3018, which is highlighted with a green box and shows the instruction 'AND R2, R2, x0000'.

Register	Value
R0	x0000 0
R1	x0000 0
R2	x0000 0
R3	x0000 0
R4	x0000 0
R5	x0000 0
R6	x0000 0
R7	x3003 12291

PC	IR	CC
x3018	x4818	Z

Address	Instruction
x3000	LD R0, mswnum1
x3001	LD R1, mswnum2
x3002	JSR cmp16
x3003	AND R2, R2, R2
x3004	BRP pgt
x3005	BRN plt
x3006	LD R0, lswnum1
x3007	LD R1, lswnum2
x3008	JSR cmp16
x3009	BRP pgt
x300A	BRN plt
x300B	BRZ equ
x300C	AND R2, R2, x0000
x300D	ADD R2, R2, x0001
x300E	TRAP HALT
x300F	AND R2, R2, x0000
x3010	TRAP HALT
x3011	AND R2, R2, x0000
x3012	ADD R2, R2, xFFFF
x3013	TRAP HALT
x3014	BRNOP x3000
x3015	BRNOP x3005
x3016	BRNOP x3000
x3017	BRNOP x3007
x3018	AND R2, R2, x0000
x3019	AND R0, R0, R0
x301A	BRN pneg
x301B	AND R1, R1, R1
x301C	BRN plts
x301D	BRZP conc

Vediamo ora come si comporta in dettaglio la routine **cmp16**:

1. **AND R2, R2, x0000** → viene inizializzato a 0 il registro R2. Ricordiamo che tale registro conterrà il risultato del confronto tra i due numeri a 16 bit;
2. **AND R0, R0, R0** → verifica se il primo numero, contenuto in R0, è positivo, negativo o nullo: il risultato di questa operazione è visibile nei CC (**N** se il numero è negativo, **P** se il numero è positivo, **Z** se il numero è nullo);
3. **BRN pneg** → se il primo numero è negativo ci portiamo all'indirizzo identificato dalla label **pneg** (x301E). In questa situazione se il secondo numero è positivo o nullo, allora sicuramente primo numero < secondo numero. Vediamo infatti cosa succede se finiamo nel "ramo" **pneg**, ricordandoci che se ci entriamo abbiamo la certezza che il primo numero è NEGATIVO:
  - **AND R1, R1, R1** → verifica se anche il secondo numero, contenuto in R1, è negativo: come al solito il risultato di questa operazione è visibile nei CC.
  - **BRZP plts** → se il secondo numero è positivo o nullo, allora salta all'indirizzo identificato dalla label **plts**, descritta al successivo punto 8), che dichiarerà primo numero < secondo numero;

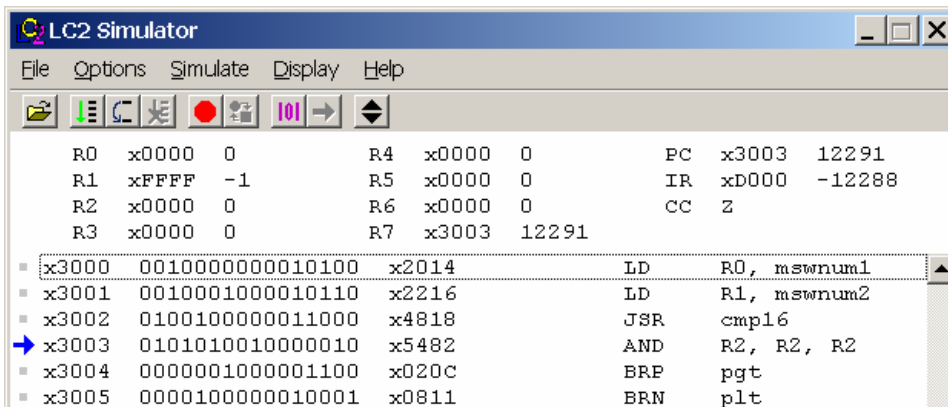
- **BRN conc** → se anche il secondo numero è negativo, allora salta all'indirizzo identificato dalla label **conc**, descritta al successivo punto 7);

Se invece il primo numero è POSITIVO, proseguiamo con il punto 4).

4. **AND R1, R1, R1** → verifica se il secondo numero, contenuto in R1, è positivo, negativo o nullo: il risultato di questa operazione è visibile nei CC (**N** se il numero è negativo, **P** se il numero è positivo, **Z** se il numero è nullo);
5. **BRN pgts** → se il secondo numero è negativo ci portiamo all'indirizzo identificato dalla label **pgts** (descritta al successivo punto 10), che dichiarerà primo numero > secondo numero; in caso contrario andiamo al punto 6);
6. **BRZP conc** → avendo verificato che il secondo numero NON è negativo, questa istruzione salta all'indirizzo identificato dalla label **conc**, descritta al successivo punto 7);
7. vediamo come opera la parte di codice indirizzata dalla label **conc**, supponendo a titolo di esempio che R0 = xFFFA = #-6 e R1=xFFFB=#-5. Deve quindi risultare (alla fine) primo numero < secondo numero:
  - NOT R1, R1 → inverte il secondo numero → R1 = x0004 = #4;
  - ADD R0, R0, R1 → somma il secondo numero al primo e salva il risultato nel primo numero → R0 = xFFFE = #-2;
  - NOT R0, R0 → inverte il primo numero → R1 = 1;
  - BRN pgts → se CC = N allora salta all'indirizzo identificato dalla label pgts (descritta al successivo punto 10), che dichiarerà primo numero > secondo numero;
  - BRZ peqs → se CC = Z allora salta all'indirizzo identificato dalla label peqs (descritta al successivo punto 9), che dichiarerà primo numero = secondo numero;
  - BRP plts → se CC = P allora salta all'indirizzo identificato dalla label plts (descritta al successivo punto 8), che dichiarerà primo numero < secondo numero;
8. vediamo come opera la parte di codice indirizzata dalla label **plts**:
  - ADD R2, R2, xFFFF → R2 = xFFFF = #-1;
  - RET → ritorna al programma principale
9. vediamo come opera la parte di codice indirizzata dalla label **peqs**:
  - RET → ritorna al programma principale (R2 era già stata inizializzata a 0)
10. vediamo come opera la parte di codice indirizzata dalla label **pgts**:
  - **ADD R2, R2, x0001** → R2 = x0001 = #1;
  - **RET** → ritorna al programma principale

Torniamo ora alla descrizione del programma principale supponendo, come detto all'inizio, che il primo numero sia x00000005 (32 bit) e il secondo numero sia x00000007, e riprendiamo

dall'istruzione x3003, ossia dalla prima istruzione che deve essere eseguita dopo la chiamata a **cmp16**. La situazione è la seguente:



The screenshot shows the LC2 Simulator interface. At the top, there's a menu bar with 'File', 'Options', 'Simulate', 'Display', and 'Help'. Below the menu is a toolbar with various icons. The main window displays the state of the processor:

Register	Value (Hex)	Value (Dec)	Register	Value (Hex)	Value (Dec)	Register	Value (Hex)	Value (Dec)
R0	x0000	0	R4	x0000	0	PC	x3003	12291
R1	xFFFF	-1	R5	x0000	0	IR	x0000	-12288
R2	x0000	0	R6	x0000	0	CC	Z	
R3	x0000	0	R7	x3003	12291			

Below the register state, the instruction stream is shown:

Address	Binary	Hex	Instruction
x3000	00100000000010100	x2014	LD R0, mswnum1
x3001	00100010000010110	x2216	LD R1, mswnum2
x3002	01001000000011000	x4818	JSR cmp16
x3003	01010100100000010	x5482	AND R2, R2, R2
x3004	00000010000001100	x020C	BRP pgt
x3005	00001000000010001	x0811	BRN plt

Come ci aspettavamo R2 = 0, dal momento che le parole più significative dei due numeri sono uguali (mswnum1=mswnum2=x0000).

La successiva istruzione **AND R2, R2, R2** verifica il contenuto di R2:

se R2 è positivo, allora primo numero > secondo numero; si esegue quindi un salto all'istruzione identificata dalla label **pgt** (istruzione **BRP pgt**)

se R2 è negativo, allora primo numero < secondo numero; si esegue quindi un salto all'istruzione identificata dalla label **plt** (istruzione **BRN plt**)

Nel nostro caso R2 = 0, pertanto l'elaborazione prosegue:

vengono lette le due parole meno significative dei due numeri (lswnum1 e lswnum2), mediante le istruzioni **LD R0,lswnum1** e **LD R0,lswnum2**;

viene eseguita la routine **cmp16** tramite l'istruzione **JSR cmp16**.



Tornando dalla routine **cmp16** la situazione è la seguente:

The screenshot shows the LC2 Simulator interface. At the top, there's a menu bar with 'File', 'Options', 'Simulate', 'Display', and 'Help'. Below the menu is a toolbar with various icons. The main window displays the state of the processor and the instruction stream.

Register	Value	Register	Value	Register	Value
R0	x0002 2	R4	x0000 0	PC	x3009 12297
R1	xFFFF8 -8	R5	x0000 0	IR	xD000 -12288
R2	xFFFF -1	R6	x0000 0	CC	N
R3	x0000 0	R7	x3009 12297		

Address	Binary	Hex	Label	Instruction
x3000	00100000000010100	x2014		LD R0, mswnum1
x3001	00100010000010110	x2216		LD R1, mswnum2
x3002	01001000000011000	x4818		JSR cmp16
x3003	01010100100000010	x5482		AND R2, R2, R2
x3004	00000010000001100	x020C		BRP pgt
x3005	00001000000010001	x0811		BRN plt
x3006	00100000000010101	x2015		LD R0, lswnum1
x3007	00100010000010111	x2217		LD R1, lswnum2
x3008	01001000000011000	x4818		JSR cmp16
x3009	00000010000001100	x020C		BRP pgt
x300A	00001000000010001	x0811		BRN plt
x300B	00000100000001111	x040F		BRZ equ
x300C	01010100101000000	x54A0	pgt	AND R2, R2, x0000
x300D	00010100101000001	x14A1		ADD R2, R2, x0001
x300E	1111000000100101	xF025		TRAP HALT
x300F	01010100101000000	x54A0	equ	AND R2, R2, x0000

A questo punto:

- se R2 è positivo, allora primo numero > secondo numero; si esegue quindi un salto all'istruzione identificata dalla label **pgt** (istruzione **BRP pgt**)
- se R2 è negativo, allora primo numero < secondo numero; si esegue quindi un salto all'istruzione identificata dalla label **plt** (istruzione **BRN plt**)
- se R2 è nullo, allora primo numero = secondo numero; si esegue quindi un salto all'istruzione identificata dalla label **equ** (istruzione **BRN equ**).

Infine:

- l'istruzione identificata dalla label **pgt**:
  - **AND R2, R2, x0000** → azzerà R2
  - **ADD R2, R2, x0001** → imposta R2 a 1 (valore finale)
  - **TRAP HALT** → termina il programma
- l'istruzione identificata dalla label **plt**:
  - **AND R2, R2, x0000** → azzerà R2
  - **ADD R2, R2, xFFFF** → imposta R2 a -1 (valore finale)
  - **TRAP HALT** → termina il programma
- l'istruzione identificata dalla label **equ**:
  - **AND R2, R2, x0000** → azzerà R2
  - **TRAP HALT** → termina il programma.