Lezione 16: L'architettura LC-3

Laboratorio di Elementi di Architettura e Sistemi Operativi

15 Maggio 2013

Ricorda...

Il ciclo di esecuzione di un'istruzione è composto da sei fasi:

- FETCH
- DECODE
- ADDRESS EVALUATION
- OPERAND FETCH
- EXECUTE
- STORE RESULT

Istruction Set Architecture

Che cos'è l'ISA?
Il termine ISA (Instruction Set Architecture), specifica l'interfaccia tra i comandi software (forniti attraverso un linguaggio di programmazione) e quello che l'hardware è in grado di eseguire (linguaggio macchina).

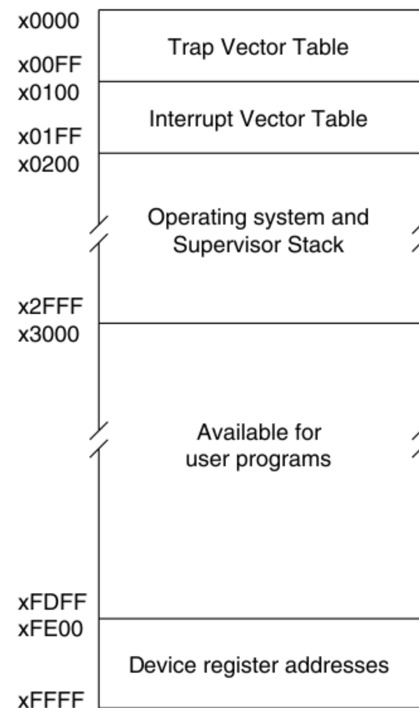
In particolare specifica:

- l'organizzazione della memoria
- l'insieme dei registri
- l'insieme delle istruzioni:
  - opcode
  - tipi di dato
  - metodi di indirizzamento

Vediamole nel dettaglio...

Organizzazione della memoria

- La memoria di LC-3 ha uno spazio di indirizzamento di 2^{16} (= 65536) locazioni.
- Ogni locazione (*word*) contiene 16 bit di dati.
- Gli indirizzi sono a 16 bit.
- Gli indirizzi di memoria sono numerati da 0 (0x0000) a 65535 (0xFFFF)
- Non tutti identificano zone di memoria disponibile per l'utente, come riportato in figura.



Insieme dei registri

- LC-3 mette a disposizione 8 General Purpose Register (GPR);
- i registri sono a 16 bit (word);
- ognuno degli 8 registri (chiamati R0, R1, ..., R7) è identificato da un numero a 3 bit (da R0=000 a R7=111).

Register 0	(R0)	0000000000000001
Register 1	(R1)	0000000000000011
Register 2	(R2)	0000000000000101
Register 3	(R3)	0000000000000111
Register 4	(R4)	1111111111111110
Register 5	(R5)	1111111111111100
Register 6	(R6)	1111111111111010
Register 7	(R7)	1111111111111000

Insieme delle istruzioni

- Un'istruzione è identificata da DUE componenti:
  - **opcode:** specifica cosa l'istruzione chiede di fare alla macchina;
  - **operandi:** specificano i dati su cui la macchina andrà ad operare;
- Ogni istruzione è codificata su 16 bit.

- *Esempio:* vogliamo fare un'ADD fra i registri R0 e R1 e memorizzare il risultato in R2. L'istruzione sarà:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	1
ADD				R2			R0						R1		

- L'ISA di LC-3 definisce 15 istruzioni, ciascuna identificata da un opcode univoco;
- L'**opcode** è specificato dai bit [12:15] di un'istruzione. Poichè vengono usati 4 bit, si possono identificare 16 istruzioni; LC-3 ne specifica 15, lasciandone libera una (codice 1101).
- Ci sono 3 diversi tipi di istruzioni:
  - *operazioni*: processano informazioni;
  - *spostamento di dati*: spostano informazioni tra registri e memoria e viceversa;
  - *controllo*: cambiano il flusso d'esecuzione delle istruzioni.

Tipi di dato

- Per *tipo di dato* si intende un modo per rappresentazione l'informazione che sia gestibile dall'ISA
  - ovvero, per la quale l'ISA mette a disposizione *istruzioni in grado di manipolarla*
- Ci sono molti modi per rappresentare un'informazione...
  - caratteri
  - modulo e segno
  - complemento a uno
  - complemento a due
  - virgola mobile
  - ...
- L'ISA di LC-3 utilizza la forma **interi in complemento a due**.

Metodi di indirizzamento

- Il *metodo di indirizzamento* è il meccanismo per specificare dove si trova un operando;
- LC-3 supporta cinque modi di indirizzamento:
  - all'interno dell'istruzione stessa (literal o immediate)
  - a registro
  - tre a memoria:
    * PC-relative
    * indiretto
    * base + offset

Il linguaggio assembly di LC-3

Abbiamo visto...

- Ogni istruzione è identificata da un insieme di 1 e di 0;
- Ogni locazione di memoria è individuata da un indirizzo a 16 bit.

Tuttavia...
scrivere programmi sottoforma di 1 e 0 non è certo il massimo. Vediamo quindi come rappresentare in modo più comprensibile i nostri programmi.

Il linguaggio assembly di LC-3

- è un linguaggio a basso livello;
- ha lo scopo di rendere la programmazione più semplice rimanendo comunque vicino all'ISA;
- vi è una corrispondenza uno a uno tra istruzioni ASM e istruzioni specifiche dell'ISA;
- mette a disposizione nomi simbolici al posto degli opcode delle istruzioni; ad esempio ADD identifica l'opcode 0001;
- permette inoltre di definire nomi simbolici per le locazioni di memoria (symbolic address).

Istruzioni Assembly

- Il formato generale di un'istruzione assembly è:

```
LABEL   OPCODE   OPERANDS   ; COMMENTS
```

- LABEL: assegna un nome simbolico ad un indirizzo (che può contenere un'istruzione o un indirizzo di memoria). È opzionale;
- OPCODE e OPERANDS: sono specifici per ogni istruzione;
- COMMENTS: identifica un commento (come il comando // del C).

- Una label è un nome simbolico che può essere usato per identificare una zona di memoria;
- viene usata nel programma per fare un riferimento esplicito alla zona di memoria associata;
- in LC-3 una LABEL può essere lunga al massimo 20 caratteri.

- Come già visto, un'istruzione ha un *OPCODE* e un certo numero di *OPERANDI*;
- l'OPCODE:
  - è un nome simbolico che corrisponde ad un'istruzione LC-3;
  - l'idea è che i nomi simbolici (ADD, AND, LDR) sono più facilmente ricordabili rispetto a 0001, 0101, 0110;
- gli OPERANDI: sono specifici per ogni istruzione; ad esempio:
  - BRz AGAIN : se zero, salta all'istruzione AGAIN;

- ADD R1,R1,#3 : somma 3 al contenuto di R1;
- Nota: nelle operazioni che richiedono valori costanti, come ad esempio l'istruzione appena vista, si utilizza *#* per indicare un decimale, *x* per indicare un esadecimale, e *b* per indicare un numero binario.

- Nel linguaggio assembly i commenti sono indicati dal **";"**
- contengono messaggi in linguaggio naturale che non hanno effetti sul processo di esecuzione;
- ciò che segue il *;* viene completamente ignorato.

Le istruzioni di LC-3: operazioni

- Le operazioni sono istruzioni che processano i dati; possono essere di tipo aritmetico (ADD, SUB, MUL, DIV) o di tipo logico (AND, OR, NOT);
- LC-3 definisce tre operazioni:
  - NOT
  - AND
  - ADD
- Combinandole assieme è possibile ottenere tutte le altre operazioni aritmetiche e logiche.

NOT

- L'istruzione **NOT** (opcode = 1001) è l'unica ad utilizzare un solo operando (operazione unaria);
- Utilizza un registro sia per la sorgente che per la destinazione;
- Esegue il complemento bit a bit sui 16 bit del registro sorgente, e memorizza il risultato nel registro destinazione;
- Esempio: si vuole porre nel registro R3 la negazione del registro R5; l'istruzione sarà:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	1	0	1	1	1	0	1	1	1	1	1	1	1
NOT				R3			R5								

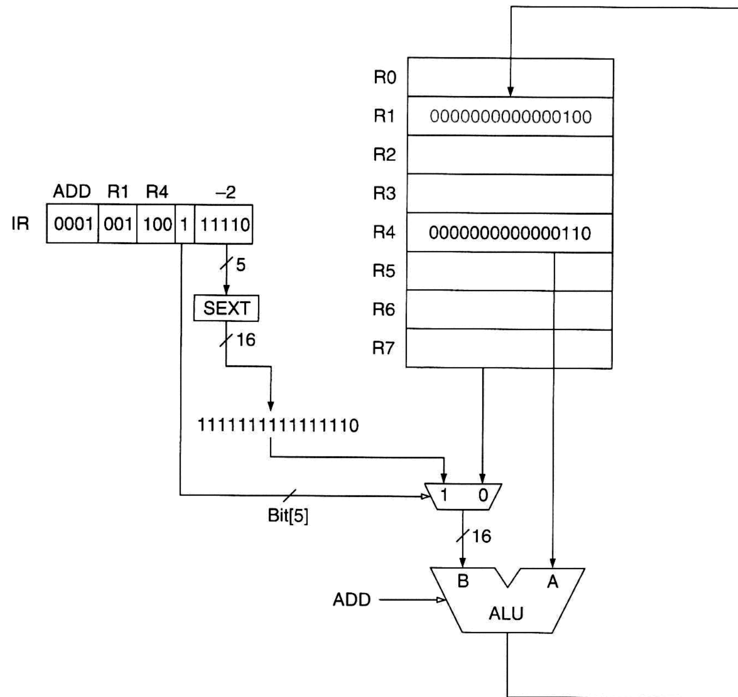
- Nota: i bit [0:5] sono tutti 1.

AND e ADD

- Le istruzioni **AND** (opcode = 0101) e **ADD** (opcode = 0001) operano su due operandi a 16 bit;
- L'ADD esegue l'addizione in complemento a due tra gli operandi;
- L'AND esegue l'and bit a bit tra gli operandi;
- A differenza di NOT, il secondo operando sorgente può essere:
  - un registro,
  - un valore costante.
- Nel primo caso il bit [5] vale 0, così come i bit [3:4]. I bit [0:2] contengono il numero del registro.

- Nel secondo caso invece, il bit [5] è settato a 1, e il valore costante è contenuto nei bit [0:4] (vedi esempio).

Esempio di esecuzione dell'istruzione "ADD R1, R4, #-2":



Le istruzioni di LC-3: controllo del flusso

LC-3 - Controllo

- LC-3 ha 5 operazioni (opcode) riservate per il controllo di flusso:
  - **salto condizionale**
  - **salti non condizionali**
  - chiamata a funzione
  - TRAP
  - return da un'interrupt
- In questo corso vedremo solo i salti condizionali e non.

Nota:
LC-3 ha 3 registri di un bit che sono settati a 1 o a 0 ogni qual volta uno degli otto registri general purpose vengono scritti. Essi sono:

- negative (indicato con N)
- zero (indicato con Z)
- positive (indicato con P)

Salti condizionali

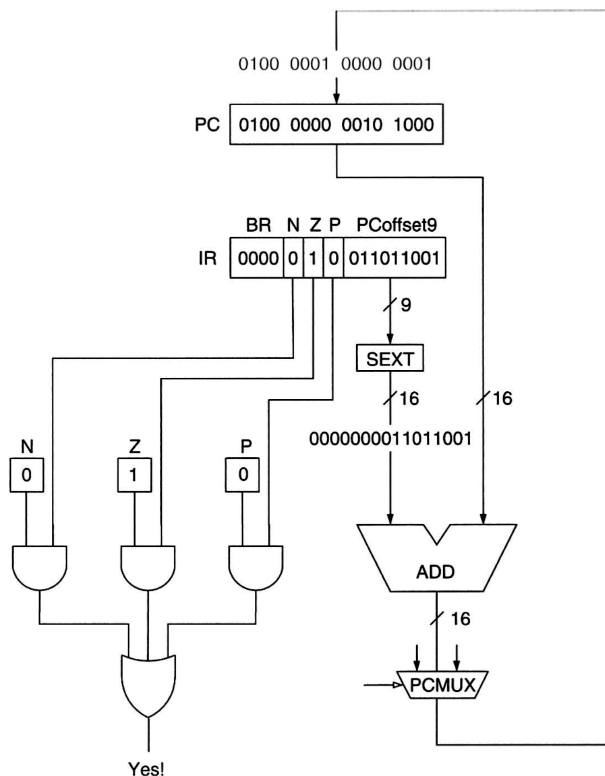
- L'istruzione di branch è **BR** (opcode = 0000);
- Il suo formato è:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	N	Z	P	PCoffset								

- I bit [9:11] corrispondono alle tre condizioni viste; essi sono usati per cambiare il flusso di esecuzione;
- Durante l'ESECUZIONE il processore valuta i bit N, Z, P, e:
  - se il bit [11] = 1 viene esaminata la condizione corrispondente a N;
  - se il bit [10] = 1 viene esaminata la condizione corrispondente a Z;
  - se il bit [9] = 1 viene esaminata la condizione corrispondente a P;
- Il linguaggio assembly mette a disposizione gli opcode **BRn**, **BRz**, **BRp**, **BRnz**, **BRnp**, **BRzp**, **BRnzp**, corrispondenti alle diverse configurazioni dei bit [9:11] dell'instruzione.
- *Solo se nessuna delle condizioni verificate è vera viene caricato nel PC il valore calcolato nella fase di ADDRESS EVALUATION*;
- **Altrimenti...**

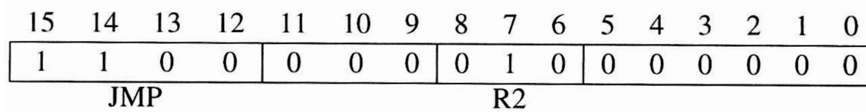
...
Verrà aggiornato il PC con il valore calcolato come somma tra il PC incrementato e il PC offset indicato nell'istruzione.

Esempio: supponiamo che l'istruzione precedente abbia prodotto in un registro general purpose il valore 0; il PC ha valore 0x4027 e l'istruzione corrente è "BRz x0D9". Dopo l'esecuzione di questa istruzione il PC avrà valore 0x4101 e non 0x4028.



Salti non condizionali

- LC-3 mette a disposizione l'istruzione **JMP** (opcode = 1100) per eseguire salti non condizionati;
- JMP carica nel PC il valore contenuto nel registro indicato dai bit [6:8];
- Esempio:



Pseudo direttive assembly

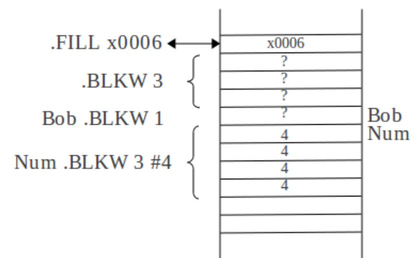
- Per scrivere un programma in assembly sono necessarie alcune *pseudo direttive* (chiamate anche *pseudo-ops*);
- esse non sono vere e proprie istruzioni da eseguire, ma...

servono in fase di traduzione da ASM a linguaggio macchina per far si che il traduttore interpreti correttamente il contenuto del programma.

- si riconoscono perché iniziano con il **"."** e, ovviamente, sono nomi riservati.

Le pseudo-direttive del linguaggio assembly di LC-3 sono:

- **.ORIG**: indica l'indirizzo di partenza del programma LC-3; ad esempio alla riga 05 viene detto che il programma deve iniziare all'indirizzo (esadecimale su 16 bit) x3050;
- **.END**: indica dove finisce il programma. Attenzione: questa istruzione non dice che il programma deve essere terminato, ma l'indirizzo a cui terminano le sue istruzioni!
- **.FILL**: indica che la prossima locazione di memoria contiene il valore indicato dall'operando; ad esempio alla riga 13 viene detto che la nona locazione (a partire dall'inizio del programma) contiene il valore x0006;



- **.BLKW**: indica di riservare una sequenza (contigua), lunga tanto quanto indicato dall'operando, di locazioni di memoria.

  Si possono anche inizializzare;

```
        .ORIG     x3010
HELLO   .STRINGZ  "Hello, World!"

  x3010: x0048
  x3011: x0065
  x3012: x006C
  x3013: x006C
  x3014: x006F
  x3015: x002C
  x3016: x0020
  x3017: x0057
  x3018: x006F
  x3019: x0072
  x301A: x006C
  x301B: x0064
  x301C: x0021
  x301D: x0000
```

- **.STRINGZ**: indica di iniziare una sequenza lunga n+1 di locazioni di memoria. L'argomento è una sequenza di n caratteri compresa fra i doppi apici.

  I caratteri vengono rappresentati su 16bit.

  Il carattere terminatore `'\0'` è sottointeso.

Da codice assembly a codice macchina

Assembly LC-3: esempio di programma

- Consideriamo il seguente programma scritto in C (quindi ad alto livello) che moltiplica per 6 l'intero memorizzato nel registro R1.
- Nota: abbiamo visto la volta scorsa che nell'ISA *non c'è la moltiplicazione*; occorre quindi creare un ciclo (for) che somma NUMBER a se stesso il numero di volte desiderate.

```
int main(){
 int R1;   // da inizializzare col valore desiderato
 int R2=6;  // costante 6
 int R3=0;  // conterra' il risultato
 for( ;R2>0;R2--){
   R3 = R3 + R1;
 }
}
```

- Il suo corrispondente in assembly è:

```
; Programma che moltiplica un intero
; contenuto in R1 per la costante 6
; Il risultato viene memorizzato in R3
;
        .ORIG   x3000
        AND     R2, R2, #0      ; Azzera R2
        ADD     R2, R2, #6      ; Memorizza 6 in R2
        AND     R3, R3, #0      ; Azzera R3. Al termine
                                ; conterrà il risultato
; Ciclo FOR
AGAIN   ADD     R3, R3, R1      ; Aggiungi R1 ad R3
        ADD     R2, R2, #-1     ; R2 è il contatore
        BRp     AGAIN           ; delle iterazioni
;
        HALT
        .END
```

L'assemblatore

- Abbiamo visto che il linguaggio assembly permette di scrivere programmi a più alto livello rispetto le istruzioni macchina;
- tuttavia un'architettura è in grado di eseguire solo istruzioni in linguaggio macchina: *occorre quindi una traduzione*;
- questo processo di traduzione viene fatto dall'**ASSEMBLATORE**.
- In generale, nel linguaggio assembly vi è una corrispondenza 1 a 1 fra le istruzioni ASM e le istruzioni macchina;
- si può quindi pensare di prendere il programma scritto in ASM e, riga per riga, determinare la corrispondente istruzione macchina.
- Esempio:

```
01  .ORIG x3000
02  AND R2,R2,#0 ; reset di R2
...
07 BRp AGAIN
```

```
01  x3000:    0101011011100000
...
```

- *PROBLEMA: cosa succede quando si arriva alla riga 07?*
- ***Alla riga 07 l'assembler NON SA ANCORA A COSA SI RIFERISCE "AGAIN"!!!***

Soluzione:
Occorre un processo di traduzione in due fasi:

1. creare la tabella dei simboli;
2. generare il programma in linguaggio macchina.

Il processo di traduzione

1. Creazione della tabella dei simboli:
    - La tabella dei simboli contiene la corrispondenza fra i nomi simbolici (ad esempio AGAIN) e gli indirizzi di memoria a 16 bit;
    - si scorre l'intero programma fra le istruzioni ".ORIG" e ".END" tenendo traccia dell'indirizzo delle istruzioni, e inserendo nella tabella dei simboli le LABEL ed il relativo indirizzo;
    - per far questo si usa una locazione chiamata Location Counter (LC), inizializzato al valore di ".ORIG".
    - Esempio:

SIMBOLO	INDIRIZZO
AGAIN	x3053

2. Generazione del programma in linguaggio macchina:
    - Si rianalizza riga per riga il programma assembly e con l'ausilio della tabella dei simboli si traduce ogni istruzione nel linguaggio macchina di LC-3;
    - si utilizza ancora una volta LC per determinare gli indirizzi;
    - quando si raggiunge ".END" il processo termina, e il risultato è un file binario corrispondente al programma scritto.

I tool di LC-3

La traduzione va fatta a mano?
No! Esistono tool che ci aiutano a fare questa traduzione. Ad esempio vedremo *lc3as*, il quale a partire da un file ".asm" genera due file: un file oggetto *.obj* contenente il programma in linguaggio macchina, e un file *.sym* contenente la tabella dei simboli generata dopo la prima analisi del file sorgente.

Come testare il nostro programma?
Attualmente LC-3 non esiste, ma esiste un simulatore che simula l'ISA di LC-3 sfruttando l'ISA si un'architettura x86 o x64. Noi useremo la versione per linux chiamata *lc3sim-tk*.

Esercitazione

Esempio:
Vogliamo provare il funzionamento del programma che esegue la moltiplicazione per 6.

- Quello che dobbiamo fare è:
    - creare il nostro programma assembly, che chiameremo "moltiplica.asm";
    - memorizzare l'input in R1;
    - provare il programma con il simulatore LC-3.

Creazione del programma assembly

- Utilizzando un qualsiasi editor di testo, creiamo il file "moltiplica.asm" e scriviamo il nostro programma:

```
; Programma che moltiplica un intero
; contenuto in R1 per la costante 6
; Il risultato viene memorizzato in R3
;
        .ORIG   x3000
        AND     R2, R2, #0      ; Azzera R2
        ADD     R2, R2, #6      ; Memorizza 6 in R2
        AND     R3, R3, #0      ; Azzera R3. Al termine
                                ; conterrà il risultato
; Ciclo FOR
AGAIN   ADD     R3, R3, R1      ; Aggiungi R1 ad R3
        ADD     R2, R2, #-1     ; R2 è il contatore
        BRp     AGAIN           ; delle iterazioni
;
        HALT
        .END
```

Traduzione del programma in linguaggio macchina

- Il prossimo passo consiste nell'invocare l'*assembler* per tradurre il nostro programma assembly in programma macchina.
- Utilizziamo il comando:

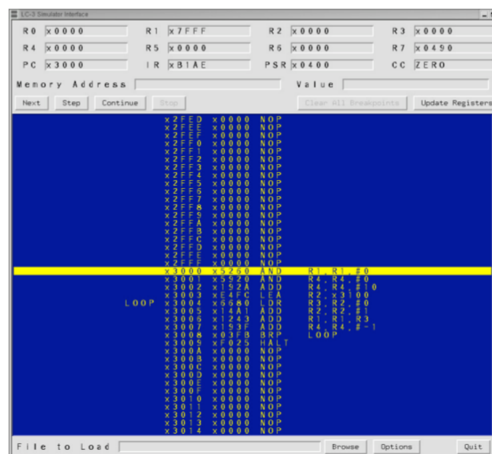
```
lc3as moltiplica.asm
```

- Otteniamo così due file:
  - il nostro programma oggetto: *moltiplica.obj*;
  - la tabella dei simboli: *moltiplica.sym*:

```
// Symbol table
// Scope level 0:
//      Symbol Name       Page Address
//      ----------------  ------------
//      AGAIN            3003
```

Il simulatore

- A questo punto possiamo lanciare il simulatore con il comando ***lc3sim-tk***.



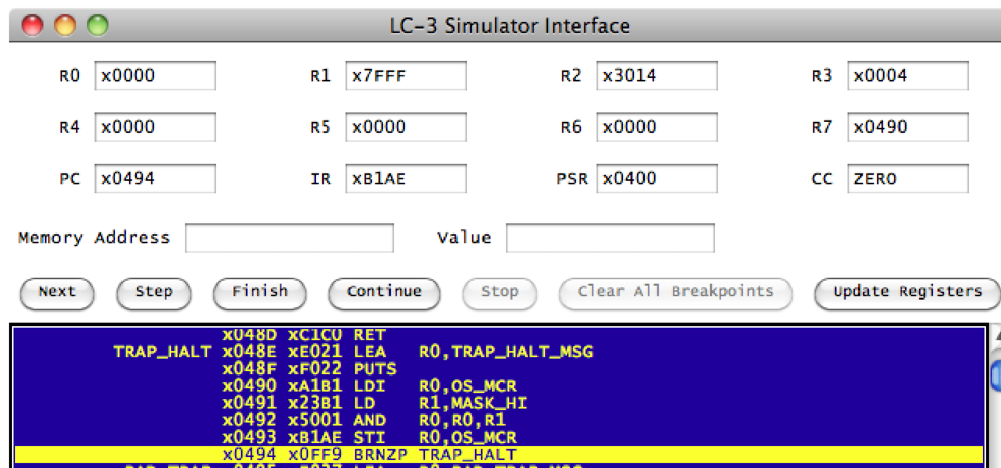
- Il simulatore è composto da 4 zone principali:

- in alto troviamo gli 8 registri general-purpose, il PC, l'IR, e CC (condition codes);
- segue un'area in cui possiamo visualizzare/settare gli indirizzi di memoria;
- l'area blu che visualizza tutti gli indirizzi della macchina simulata. In tutto sono 2^{16}=65536 zone di memoria; per visualizzare un indirizzo specifico basta scriverlo nel campo "Memory Address" e dare invio.
- in basso abbiamo il campo "File to Load" che ci permette di caricare dei file esterni.

- Possiamo quindi caricare il nostro programma cliccando su "Browse", e notiamo che esso viene caricato a partire dall'indirizzo x3000.
- Per visualizzarlo usiamo il campo "Memory Address".
- Inseriamo il valore da moltiplicare (per esempio *x0014*) nel registro R1.
- Dopo esserci assicurati che il PC contenga il valore x3000 possiamo eseguire il nostro programma passo passo cliccando su *"Next"*.
- Prima dell'esecuzione dell'istruzione HALT troveremo il risultato della moltiplicazione nel registro R3.

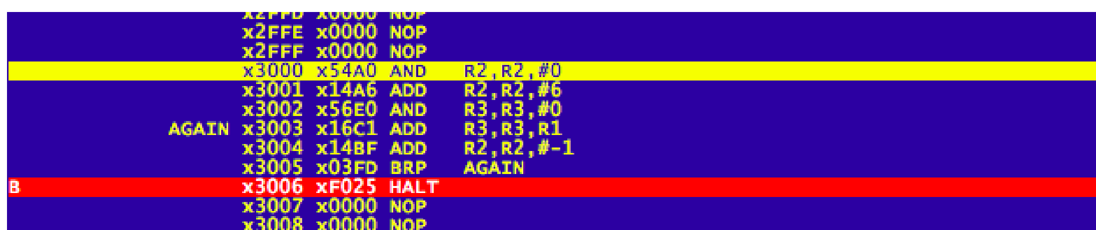
I breakpoint

- Dopo aver resettato il simulatore e ricaricato il valore x0014 in R1, proviamo ad eseguire il programma in una volta sola cliccando su *"Continue"*:

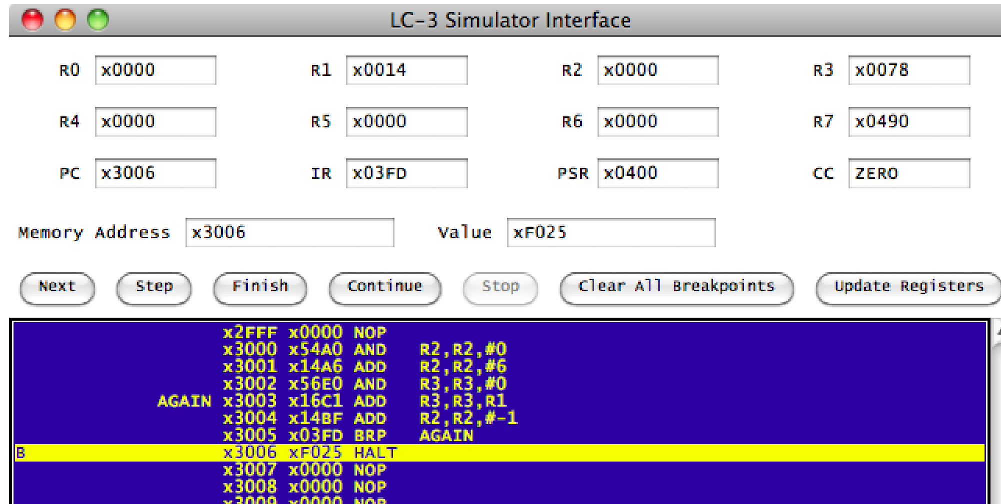


Il risultato non è quello atteso!
L'istruzione HALT fa saltare ad una zona di memoria riservata, e modifica il contenuto dei registri

- Per poter interrompere l'esecuzione del programma *prima dell'esecuzione dell'istruzione HALT* si usano i *breakpoint*.
- Resettiamo nuovamente il simulatore e facciamo un doppio click sulla locazione di memoria che contiene l'istruzione HALT:



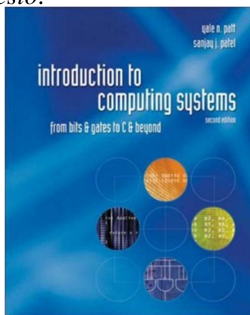
- La B all'inizio della linea identifica il breakpoint:
  - forza il simulatore a fermarsi *prima* dell'esecuzione dell'istruzione HALT
- Ora il comando *"Continue"* ci consente di vedere il risultato del calcolo nel registro R3:



- x0078 è l'esadecimale per 120, che è uguale a 20 (in esadecimale x0014) moltiplicato per 6.
  - controllate usando la calcolatrice in modalità *Programmazione*

Riferimenti

Libro di testo:



Yale N. Patt, Sanjay J. Patel.
Introduction to Computing Systems:
From Bits and Gates to C and Beyond.
Seconda edizione.
McGraw-Hill Higher Education, 2003.

`http://www.mhhe.com/patt2`

Contiene il simulatore, il manuale del simulatore e le appendici con l'elenco completo delle istruzioni e la descrizione dell'ISA