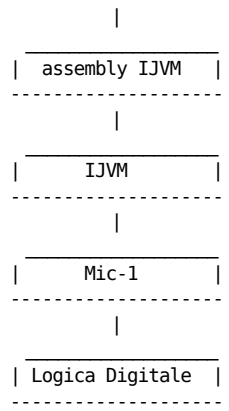
# Note introduttive sul Linguaggio Mic-1

Il linguaggio Mic-1 e' il linguaggio di microprogrammazione utilizzato nel sistema di calcolo a livelli didattico utilizzato nel nostro libro di testo, sistema che possiamo schematizzare come segue:




Il microlinguaggio Mic-1 serve a "guidare" il data path dell'Hardware del nostro sistema.
La macchina IJVM del nostro sistema invece e' realizzata per interpretazione, in particolare firmware, sopra Mic-1.
E' chiaro quindi come la struttura del linguaggio Mic-1 sia fortemente influenzata dal fatto che:

- Mic-1 e' il linguaggio che "guida" l'hardware;
- Mic-1 e' stato sviluppato al fine di realizzare IJVM per interpretazione (in particolare via firmware).

Nel libro di testo infatti (nel Capitolo 4) il Mic-1 e' presentato contemporaneamente sia alla descrizione dell'Hardware del sistema (cioe' insieme alla realizzazione Hw di Mic-1), che insieme al linguaggio IJVM ed alla sua realizzazione per interpretazione su Mic-1.

Allo scopo di affrontare per gradi lo studio del Capitolo 4, ed anche per fissare meglio i concetti di *realizzazione Hw* e *realizzazione interpretativa* di macchine astratte, in queste note introduciamo il linguaggio Mic-1 come linguaggio di programmazione a se stante, non legato in alcun modo cioe' al fatto di essere realizzato in Hw e di dover essere utilizzato per interpretare IJVM.
Certo, visto isolatamente come semplice linguaggio di programmazione, e' ovvio che apparira' un linguaggio un po' bizzarro, ma comunque sempre di linguaggio di programmazione si tratta.

Come tutti i ragionevoli linguaggi, Mic-1 possiede il concetto di **variabile**.
Pero', a differenza di molti linguaggi, Mic-1 possiede un numero fisso di variabili, con dei nomi precisi:
MAR, MDR, PC, MBR, MBRU, SP, LV, CPP, TOS, OPC, H
Nomi magari strani, ma che hanno invece un senso che capiremo solo successivamente (quando useremo Mic-1 per realizzare IJVM).
Per noi ora non debbono essere altro che nomi che identificano delle variabili.

Le variabili di Mic-1 possono contenere valori numerici. In particolare,

numeri interi rappresentabili in complemento a 2 con 32 bit.
Tranne MBR e MBRU che invece possono contenere solo numeri piu' piccoli: MBR numeri interi rappresentabili in complemento a 2 con 8 bit, MBRU numeri naturali rappresentabili con 8 bit.
Inoltre, MBR, MBRU sono variabili in sola lettura (possono apparire solo a destra di un'assegnamento), mentre MAR e H sono in sola scrittura (possono apparire solo a sinistra di un'assegnamento). Tutte le altre sono in lettura/scrittura.

Un **programma** Mic-1 ha una struttura estremamente semplice: e' una sequenza di istruzioni. Quindi, per Mic-1, piu' che di programmi, si puo' parlare di "segmenti di codice", non esistendo in Mic-1 concetti sintattici piu' complessi come *blocco*, *procedura*, *metodo* e, appunto, *programma*.

Un'**istruzione** Mic-1 ha la seguente forma:

[ *<etichetta>* ] [ *<assegnamento>* **;** ] [ *<lettura/scrittura>* **;** ] [ *<salto>* ]

Sono messe tra parentesi quadre le parti che in un'istruzione sono opzionali (possono cioe' non essere presenti). Notare come, essendo opzionali tutte le parti, un'istruzione puo' essere vuota (l'istruzione che non fa nulla).

Vediamo ora in dettaglio come possono essere le varie parti che compongono una singola istruzione.

- *<etichetta>* e' una qualsiasi stringa alfanumerica, utilizzata per identificare l'istruzione nei salti.
- *<assegnamento>* e' della forma $Var_1=Var_2=\ldots=Var_n=$*<expr>* [*<shift>*]
  dove
  - $Var_i$ sono variabili in scrittura (notare come in Mic-1 quindi possiamo avere assegnamenti multipli)
  - *<expr>* e' un'espressione della forma seguente: *B*, `NOT`*B*, *B*`+H`, *B*`+H+1`, *B*`+1`, *B*`-H`, *B*`-1`, `-H`, *B* `AND H`, *B* `OR H`, `0`, `1`, `-1`.
    Dove *B* puo' essere una qualsiasi delle variabili in lettura.
    Notare come in Mic-1 le uniche costanti esistenti ed utilizzabili siano `0`, `1` e `-1`.
  - <shift> puo' essere: `>1` oppure `<<8`
- *<lettura/scrittura>* puo' essere `rd`, `wr` oppure `fetch`
- *<salto>* puo' essere
  `goto`*<etichetta>*, oppure
  `if`*<cond>*`goto` *<etichetta>*`;else goto`*<etichetta>*
  dove *<cond>* puo' essere `(N)` oppure `(Z)`
  **N.B.** Non ci puo' essere un salto condizionato in un'istruzione in cui non sia presente anche un assegnamento.

Quindi un'istruzione Mic-1 potrebbe essere:

```
et1 MAR=CPP= SP+H+1 >1; fetch; if (N) goto et2; else goto et4
```

oppure

```
pippo H=TOS; rd; goto etic
```

oppure solamente

```
etic wr
```

Descritta la sintassi di Mic-1, vediamo ora qual e' il significato, la *semantica*, delle istruzioni di tale linguaggio.
Il **significato dell'assegnamento** e' ovvio: si assegna alla variabile (o alle variabili, se l'assegnamento e' multiplo) a sinistra del simbolo "=" il valore di <*expr*>[<*shift*>].
Il valore di <*expr*> e' ovvio. Tale valore viene pero' shiftato di una posizione a destra prima dell'assegnamento nel caso sia presente

">1", oppure a sinistra di 8 bit nel caso sia presente "<<8".
Il **significato del salto non condizionato** e' semplice: con "goto <*etichetta*>" la prossima istruzione eseguita sara' quella identificata da <*etichetta*>.
Il **significato del salto condizionato** invece e' il seguente: si salta all'etichetta indicata nel ramo then o quello else a seconda che la condizione <*cond*> sia vera o meno.
"(N)" e' vera se il valore di <*expr*> nell'assegnamento e' negativo.
"(Z)" e' vera se il valore di <*expr*> nell'assegnamento e' zero.
Notare che si considera il valore di <*expr*> NON shiftato, anche se ci sono presenti ">1" oppure "<<8".
(In realta' il Mic-1 possiede anche un altro tipo di salto che pero', essendo in apparenza un po' strano, lo studieremo quando si affrontera' sul testo la realizzazione Hw di Mic-1).
Il **significato di** `rd` e': leggi dalla memoria dati il contenuto della cella il cui indirizzo e' in MAR e poni tale valore in MDR.
Il **significato di** `wr` e': scrivi nella memoria dati il contenuto di MDR nella cella il cui indirizzo e' in MAR.
Il **significato di** `fetch` e': leggi dalla memoria dati il contenuto della cella il cui indirizzo e' in PC e poni tale valore in MBR e MBRU.
Dal significato di `rd/wr` e di `fetch` si evince che di memorie dati Mic-1 ne ha due: una memoria dati a 32 bit, su cui si opera con `rd/wr`, ed una memoria dati di sola lettura, a 8 bit, su cui si opera con fetch.
Per le operazioni di `rd` e `fetch` esiste la seguente condizione: il valore letto con rd o fetch sara' disponibile in MDR o MBR (MBRU) solamente DUE istruzioni dopo quella contenente il rd o fetch.

Se, per descrivere la sintassi di Mic-1, volessimo usare (anche se non in modo estremamente preciso) il formalismo utilizzato per l'esercizio 43 sulle Macchine Astratte, avremmo:

*<Programma>* ::= *<Istruzione>*
*<Programma>*

*<Istruzione>* ::= [ *<etichetta>* ] [ *<assegnamento>* ; ] [ *<lettura/scrittura>* ; ] [ *<salto>* ]

*<etichetta>* e' una qualsiasi stringa alfanumerica

*<assegnamento>* ::= $<Var>_1=<Var>_2=\ldots=<Var>_n=<expr>[<shift>]$

$<Var>_i$ ::= `MAR` | `MDR` | `PC` | `SP` | `LV` | `CPP` | `TOS` | `OPC` | `H`

*<expr>* ::= *B* | `NOT`*B* | *B*`+H` | *B*`+H+1` | *B*`+1` | *B*`-H` | *B*`-1` | `-H` | *B* `AND H` | *B* `OR H` | `0` | `1` | `-1`

*<B>* ::= `MDR` | `PC` | `MBR` | `MBRU` | `SP` | `LV` | `CPP` | `TOS` | `OPC`

<shift> ::= `>1` | `<<8`

*<lettura/scrittura>* ::= `rd` | `wr` | `fetch`

*<salto>* ::= `goto` *<etichetta>* | `if` *<cond>*`goto` *<etichetta>*`;else goto`*<etichetta>*

*<cond>* ::= `(N)` | `(Z)`

**N.B.**: In alcuni casi, come variabile a sinistra di un'assegnamento si utilizza N oppure Z. Queste due variabili non esistono, e tale assegnamento serve solo per valutare il valore di <*expr*> quando volessimo utilizzare nell'istruzione un salto condizionato in cui la condizione controlli se il valore di <*expr*> e' negativo, oppure zero. Un esempio di uso di N e Z e' nel segmento di codice che segue.

**Esempio di codice Mic-1**
Il seguente segmento di codice inserisce in SP il valore ottenuto dalla moltiplicazione dei valori contenuti in TOS e OPC, supponendo che il contenuto di OPC sia >= 0

```
        H=0
L1      OPC=OPC-1; if(N) goto Exit; else goto L2
L2      H=TOS+H; goto L1
Exit    SP=H
```

Ora bisogna mettersi a fare esercizi di programmazione in Mic-1, utilizzando anche il simulatore Mic-1 scaricabile dalla pagina del programma.
Una volta presa dimestichezza con Mic-1, potremo affrontare il Capitolo 4 del testo, in cui, tra le altre cose, e' descritta la realizzazione Hw di Mic-1.