

Laboratorio di Architetture degli elaboratori

Corso B – Turno 2

Docente: Claudio Schifanella

Lezione 9

Progettazione di nuove istruzioni IJVM e estensione
del micro-codice

Estensione Mic1

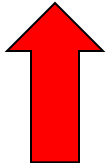
Come progettare nuove istruzioni IJVM

- Definizione nuova macro IJVM
- Definizione delle micro-istruzioni (in MAL) che realizzano la nuova macro
- Creazione programmi .jas che usano la nuova istruzione progettate
- Uso simulatore per verificare la correttezza

Definizione nuova macro IJVM

- Copiare il file ./lib/ijvm.conf e rinominarlo
- Aggiungere la nuova macro-istruzione
 - OPCODE in esadecimale (occhio a mettere un valore che non sia uguale a quelli già utilizzati)
 - Nome simbolico
 - Commenti (dopo "//")

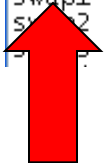
```
0xFF    HALT    // halt the simulator
0xFE    ERR     // print ERROR and halt
0xFD    OUT     // Pop a word from the stack and use the low order 8-bits as an ASCII character to
0xFC    IN      // Push a character from standard input and put it in the low order 8-bits of
0x40    PIPPO    // Delete two words on top of stack
```



Definizione micro-istruzioni ..(I)

- Copiare il file ./examples/MAL/mic1ijvm.mal e rinominarlo
- Aggiungere le micro-istruzioni relative alla nuova macro-istruzione IJVM
 - Etichetta
 - Operazioni
 - Commenti (dopo //)

```
pop2      // Wait for new TOS to be read from memory
pop3      TOS = MDR; goto Main1 // Copy new word to TOS
pippo1    SP = SP - 1; goto pop1 // Decrement the SP and goes to pop1
swap1     MAR = SP - 1; rd      // Set MAR to SP - 1; read 2nd word from stack
swap2     MAR = SP             // Set MAR to top word
pop       H = MDR; wr          // Save TOS in H; write 2nd word to top of stack
```

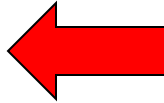


Definizione micro-istruzioni ..(II)

- Definire l'indirizzo nella control store relativo alla prima micro-istruzione

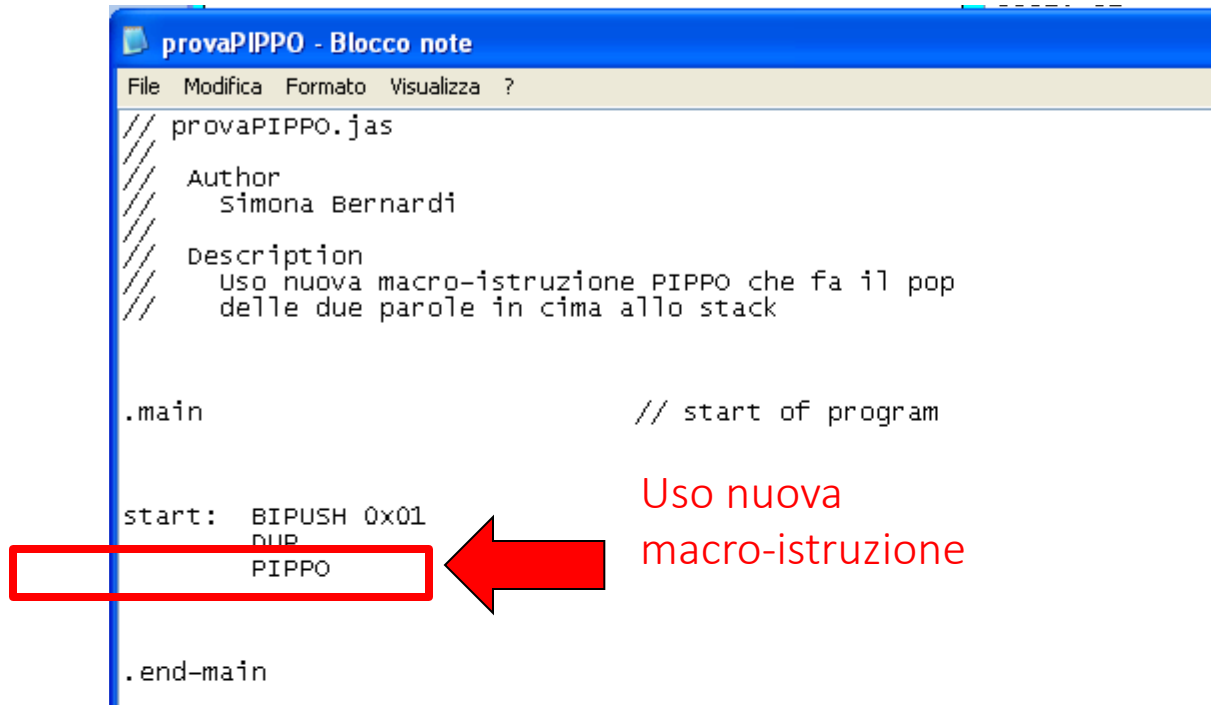
```
//  
// labeled statements are "anchored" at the specified control store address  
.label nop1 0x00  
.label bipush1 0x10  
.label ldc_w1 0x13  
.label iload1 0x15  
.label wide_ildoad1 0x115  
.label istore1 0x36  
.label wide_istore1 0x136  
.label pop1 0x57  
.label pippo1 0x40  
.label dup1 0x39  
.label swap1 0x5F  
.label iadd1 0x60  
.label isub1 0x64  
.label iand1 0x7E  
.label iinc1 0x84  
.label ifeq1 0x99  
.label iflt1 0x9B  
.label if_icmpeq1 0x9F  
.label goto1 0xA7  
.label ireturn1 0xAC  
.label ior1 0xB0  
.label invokevirtual1 0xB6  
.label wide1 0xC4  
.label halt1 0xFF  
.label err1 0xFE
```

L'indirizzo deve essere
uguale all'OPCODE
della nuova
macro-istruzione!!



Uso nuova macro-istruzione

- Creare un nuovo file .jas e scrivere il programma (in JAL) usando la nuova macro-istruzione



```
// provaPIPP0.jas
//
// Author
//   Simona Bernardi
//
// Description
//   Uso nuova macro-istruzione PIPPO che fa il pop
//   delle due parole in cima allo stack

.main                                // start of program

start: BIPUSH 0x01
      DUP
      PIPPO

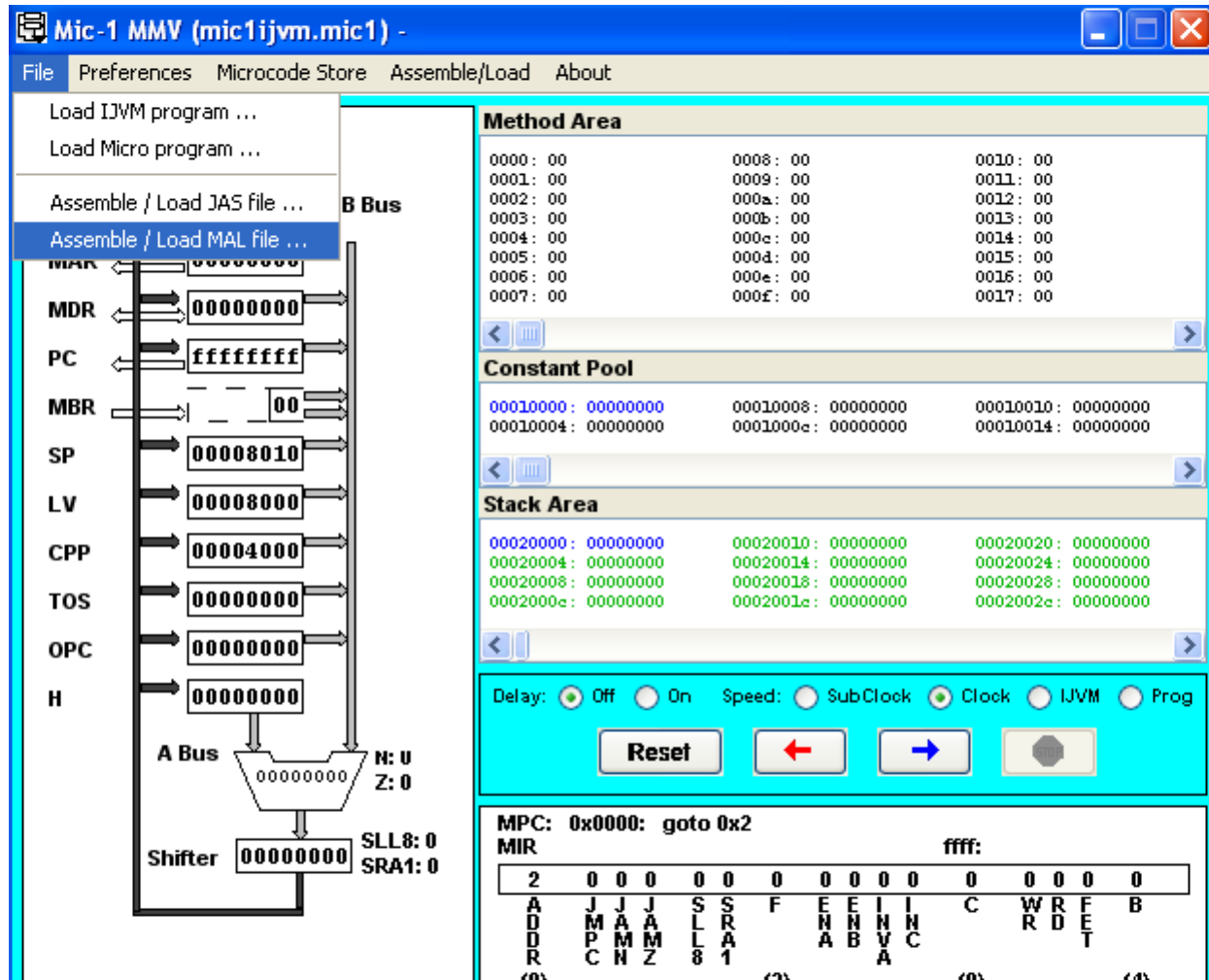
.end-main
```

Uso nuova
macro-istruzione

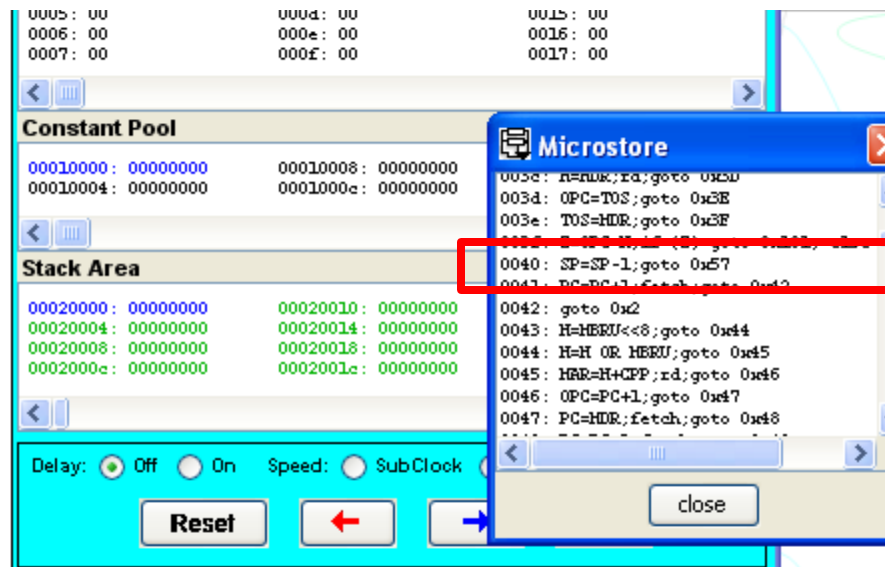
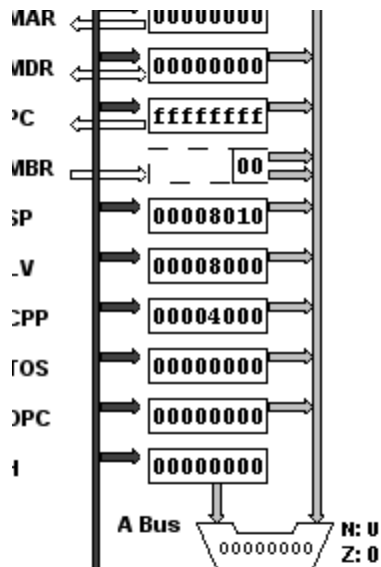
Uso Mic1MMV

- Creare una nuova cartella in esempi:
 - ./examples/myExamples
- Copiare nella nuova cartella
 - Il simulatore Mic1MMV.jar
 - Il nuovo file di configurazione ijvm2.conf
 - Il nuovo file del microinterprete mic1ijvm2.mal
 - Il file .jas del programma che si vuole simulare
- Lanciare il simulatore

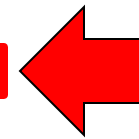
Assemblare e caricare il nuovo file .mal (I)



Assemblare e caricare il nuovo file .mal (II)



La nuova
micro-istruzione



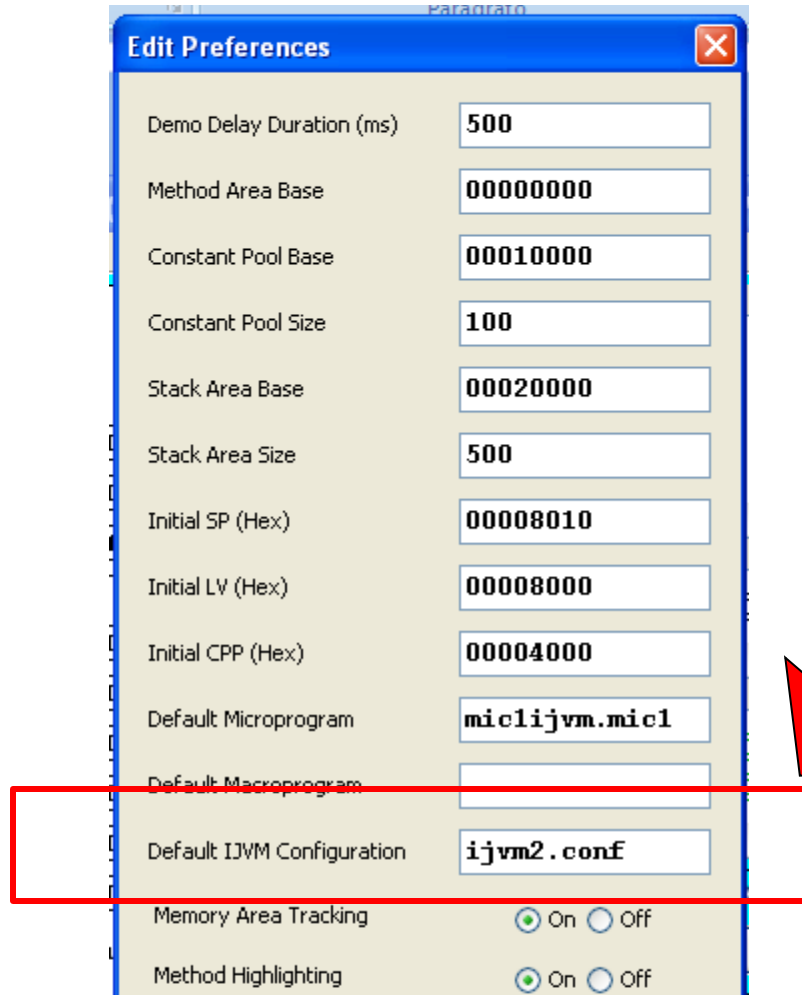
Caricare il nuovo file di configurazione IJVM .conf (I)

The screenshot shows the Mic-1 MMV emulator window. The title bar reads "Mic-1 MMV (mymicro.mic1) - MAL: C:\Programmi\Mic1MMV\examples\MAL\mym...". The menu bar includes "File", "Preferences", "Microcode Store", "Assemble/Load", and "About". A red arrow points to the "File" menu, which is open, showing options: "Edit Preferences ...", "Load Preferences ...", and "Save Preferences ...".

The main window is divided into several sections:

- Hardware State:** A vertical list of registers and buses on the left. The values are: MAR (00000000), MDR (00000000), PC (ffffff), MBR (00), SP (0008010), LV (0008000), CPP (0004000), TOS (00000000), OPC (00000000), and H (00000000). Below these are the A Bus and B Bus, and a status section showing N: 0 and Z: 0.
- Method Area:** A table of memory addresses and values. The values are mostly 00.
- Constant Pool:** A table of memory addresses and values. The values are mostly 00.
- Stack Area:** A table of memory addresses and values. The values are mostly 00.
- Controls:** A section at the bottom with "Delay" (Off/On), "Speed" (SubClock/Clock/IJVM/Pre), a "Reset" button, and navigation arrows.
- Bottom Status:** Shows "MPC: 0x0000: goto 0x2".

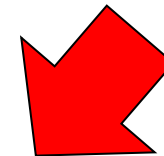
Caricare il nuovo file di configurazione IJVM .conf (II)



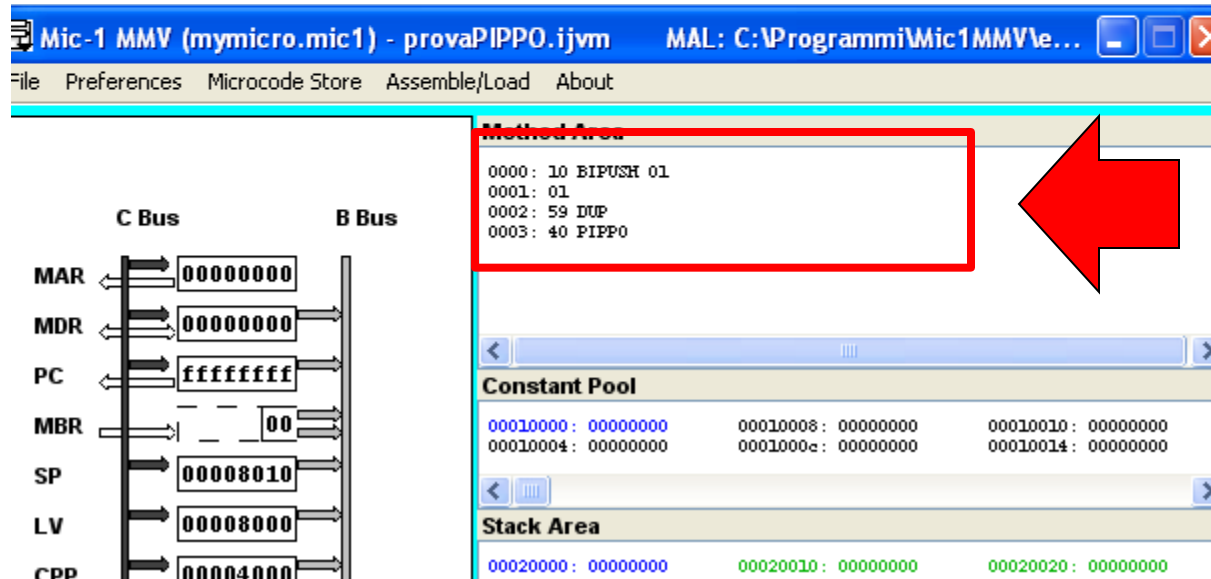
Edit Preferences

Demo Delay Duration (ms)	500
Method Area Base	00000000
Constant Pool Base	00010000
Constant Pool Size	100
Stack Area Base	00020000
Stack Area Size	500
Initial SP (Hex)	00008010
Initial LV (Hex)	00008000
Initial CPP (Hex)	00004000
Default Microprogram	mic1ijvm.mic1
Default Macroprogram	
Default IJVM Configuration	ijvm2.conf
Memory Area Tracking	<input checked="" type="radio"/> On <input type="radio"/> Off
Method Highlighting	<input checked="" type="radio"/> On <input type="radio"/> Off

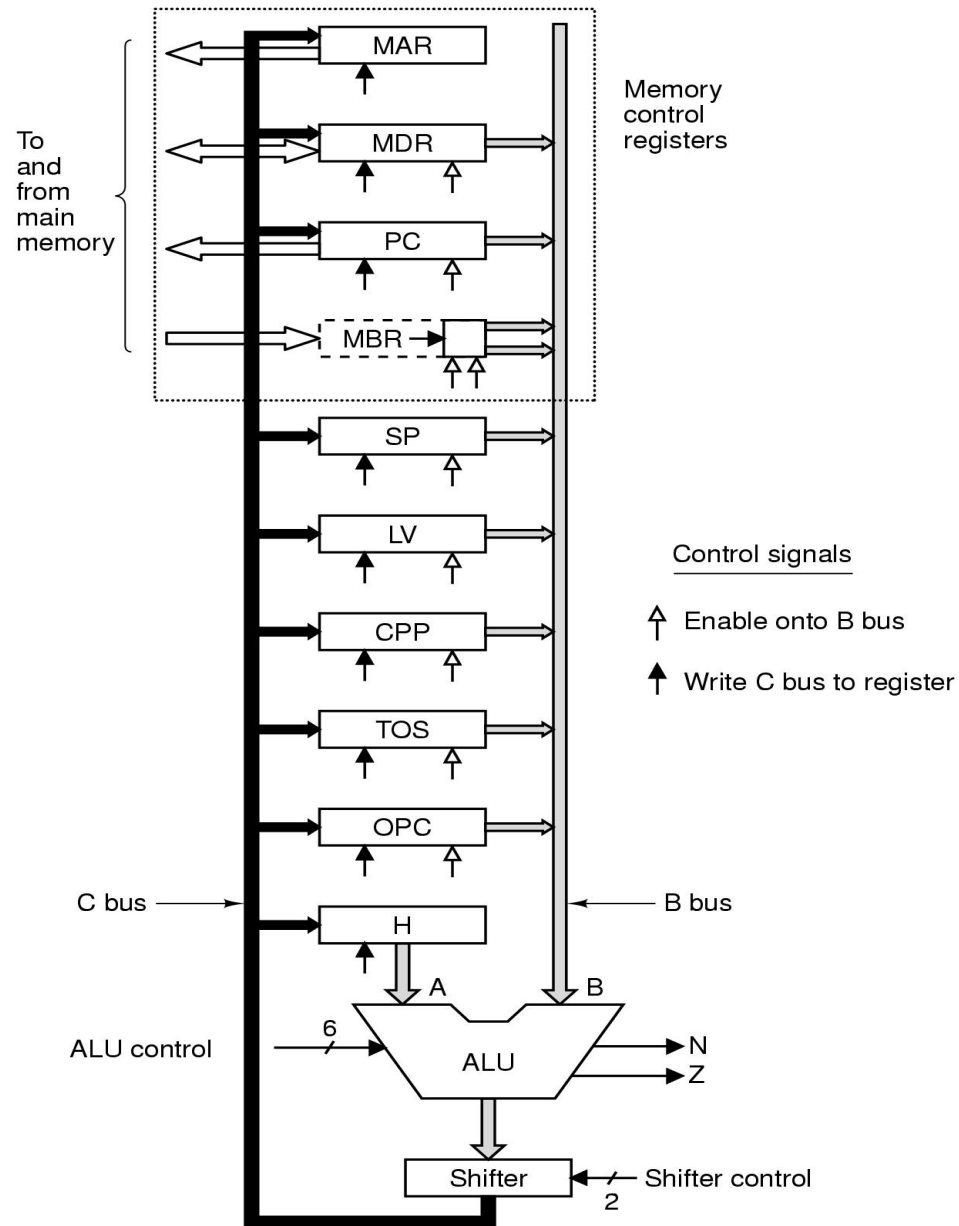
Aggiornarlo
con il nuovo
file .conf

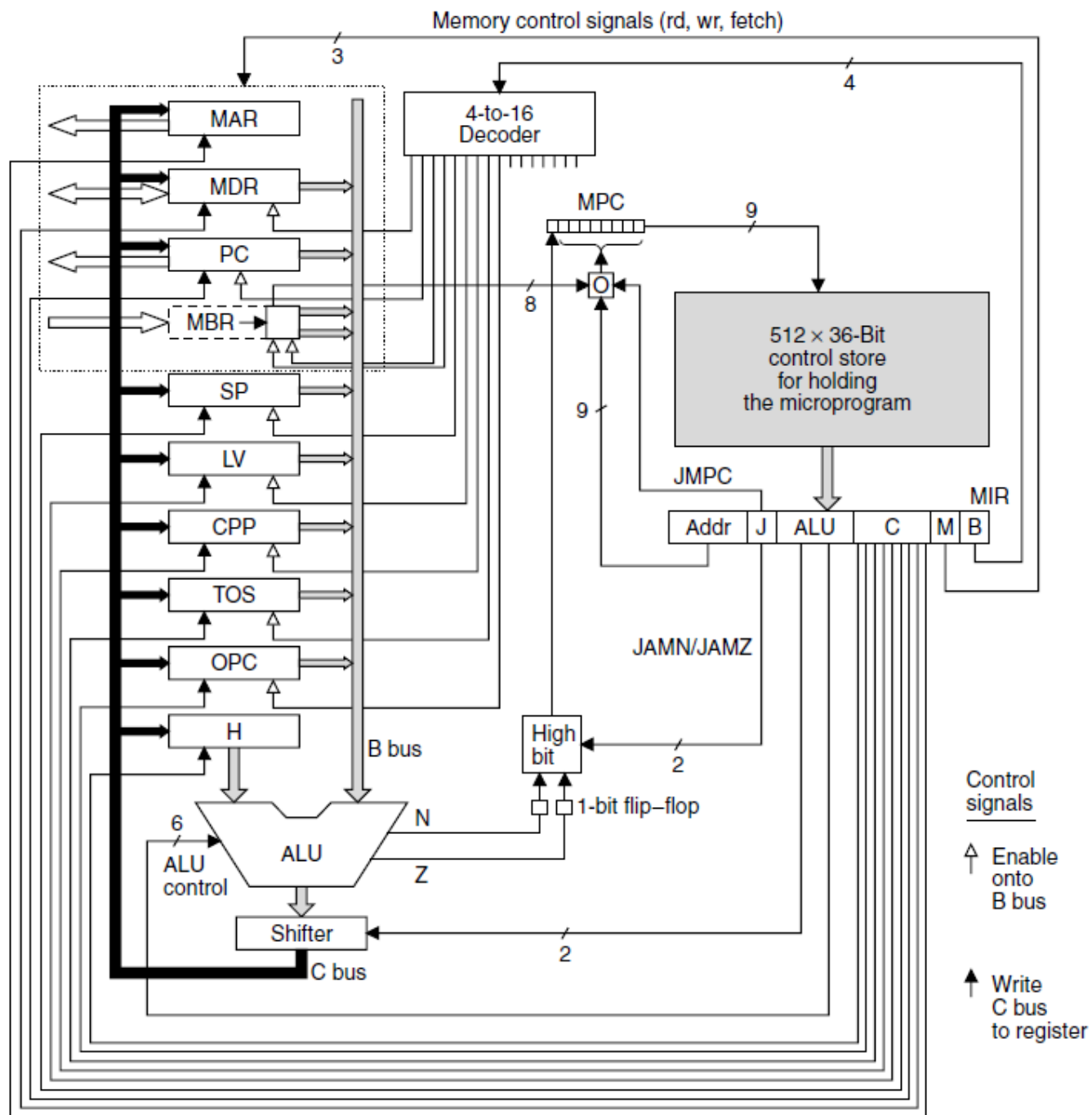


Assemblare e caricare il nuovo file .jas

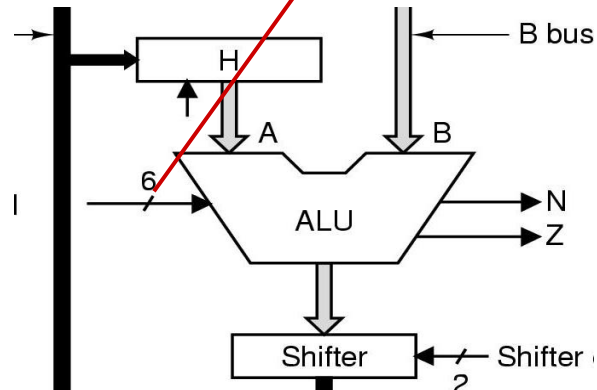


Caricato
nell'area dei
metodi



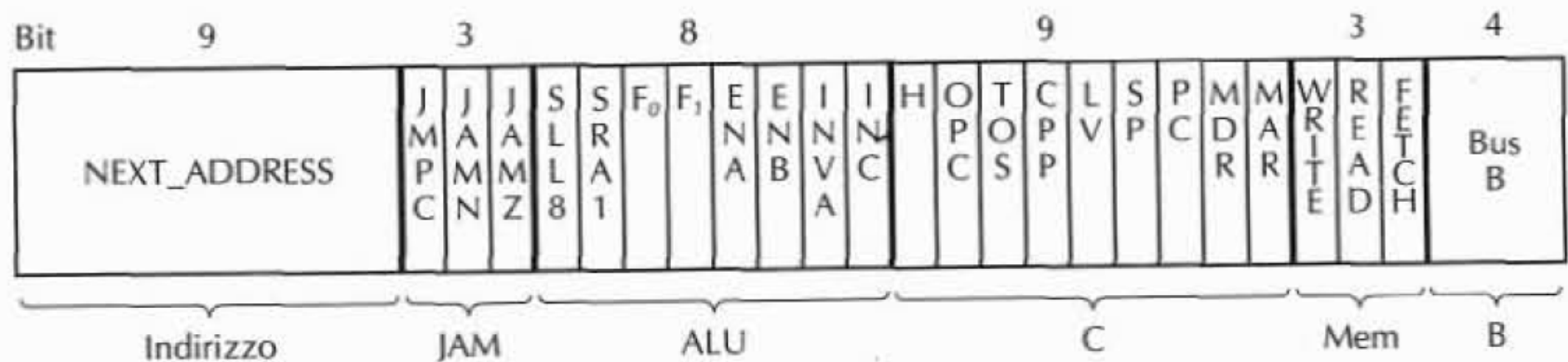


Bit di controllo dell'ALU



F_0	F_1	ENA	ENB	INVA	INC	Function
0	1	1	0	0	0	A
0	1	0	1	0	0	B
0	1	1	0	1	0	\overline{A}
1	0	1	1	0	0	\overline{B}
1	1	1	1	0	0	A + B
1	1	1	1	0	1	A + B + 1
1	1	1	0	0	1	A + 1
1	1	0	1	0	1	B + 1
1	1	1	1	1	1	B - A
1	1	0	1	1	0	B - 1
1	1	1	0	1	1	-A
0	0	1	1	0	0	A AND B
0	1	1	1	0	0	A OR B
0	1	0	0	0	0	0
1	1	0	0	0	1	1
1	1	0	0	1	0	-1

Formato microistruzioni Mic-1



Registri del bus B

0 = MDR	5 = LV
1 = PC	6 = CPP
2 = MBR	7 = TOS
3 = MBRU	8 = OPC
4 = SP	9-15 nessuno

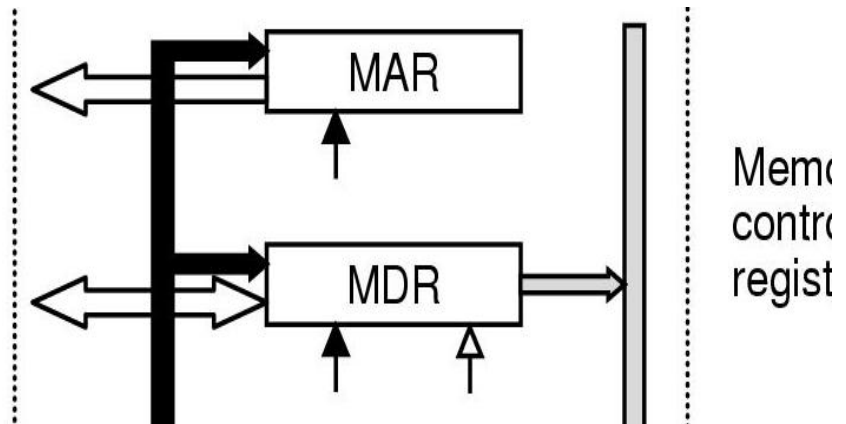
Funzionamento della memoria

La CPU ha due modi per comunicare con la memoria:

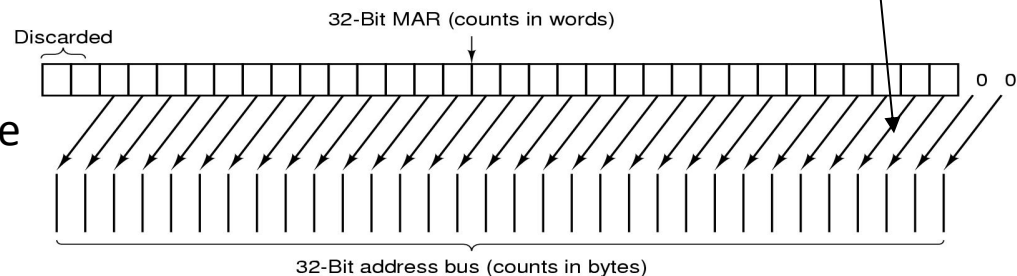
- Una porta di memoria da **32 bit** indirizzabile a parola controllata dai registri **MAR** (Memory address Register) e **MDR** (Memory Data Register)
- Una porta di memoria da **8 bit** indirizzabile a byte controllata dal registro **PC** che legge i byte e li memorizza negli 8 bit meno significativi di MBR. Questa porta può solo leggere in memoria e non può scrivere.

I registri MAR e MDR

- Porta di memoria a 32 bit
- MAR = Memory Address Register
- MDR = Memory Data Register
- Scrittura e lettura a livello ISA
- MAR ha un solo segnale di controllo (solo input dal bus C ma non output verso il bus B)
- Particolare mappatura tra valore in MAR e indirizzo in memoria:
 - MAR indirizza le parole
 - La memoria fisica conta in byte

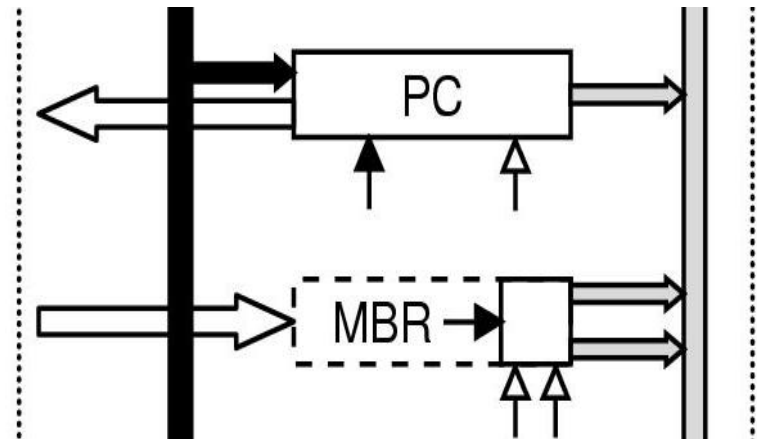


Moltiplica per 4!



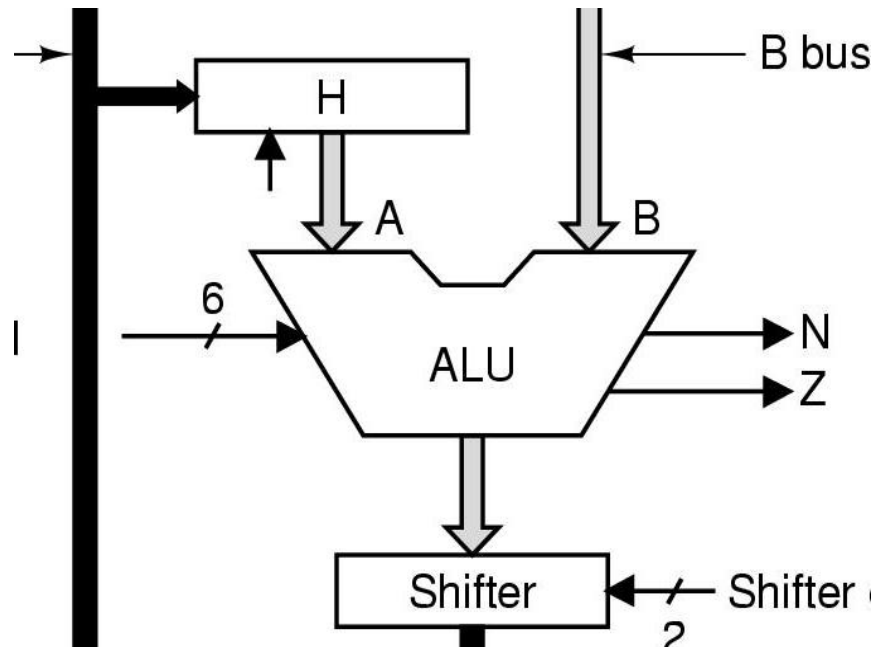
I registri MBR e PC

- Porta di memoria a **8 bit**
- MBR = Memory Buffer Register
- PC = Program Counter
- Lettura del *programma del livello ISA* da eseguire
- MBR: due bit di controllo per l'output su B: ***signed*** e ***unsigned***
- Valgono anche per MBR le considerazioni sui tempi di accesso alla memoria fatti per MDR
- Serve **1** segnale per avviare il fetch da memoria su MBR



Registro H

- Rappresenta l'input di sinistra dell'ALU
- Un solo bit di controllo per l'abilitazione dell'input dal bus C, l'output verso l'ALU è sempre attivo



Esercizio 1

- Si scriva il micro-codice che implementa per Mic-1 l'istruzione IJVM ICONST_0, versione specializzata della BIPUSH che non ha operandi e scrive la costante 0 in cima allo stack

Esercizio 1

- Si scriva il micro-codice che implementa per Mic-1 l'istruzione IJVM ICONST_0, versione specializzata della BIPUSH che non ha operandi e scrive la costante 0 in cima allo stack

```
iconst_01 MAR = SP = SP + 1 // indirizzo della nuova cima dello stack
iconst_02 TOS = MDR = 0; wr; goto Main1 // scrive la costante 0, aggiorna TOS,
//salta a Main1
```

Esercizio 2

- Si scriva il micro-codice che implementa per Mic-1 l'istruzione IJVM ILOAD_1, versione specializzata della ILOAD x che non ha operandi e scrive la variabile a scostamento 1 da LV in cima allo stack

Esercizio 2

- Si scriva il micro-codice che implementa per Mic-1 l'istruzione IJVM ILOAD_1, versione specializzata della ILOAD x che non ha operandi e scrive la variabile a scostamento 1 da LV in cima allo stack

```
iload_11 MAR = LV + 1; rd // imposta indirizzo della variabile considerata,  
        // inizia lettura  
iload_12 MAR = SP = SP + 1 // imposta indirizzo della nuova cima dello stack  
iload_13 TOS = MDR; wr; goto Main1 // scrive cima dello stack, aggiorna TOS,  
        // Main1
```


Esercizio 3

- Si scriva il micro-codice che implementa per Mic-1 l'istruzione IJVM IDOUBLE (che estrae la parola in cima allo stack, ne calcola il doppio e scrive il risultato in cima allo stack)

Esercizio 3

- Si scriva il micro-codice che implementa per Mic-1 l'istruzione IJVM IDOUBLE (che estrae la parola in cima allo stack, ne calcola il doppio e scrive il risultato in cima allo stack)

```
idouble1 H = TOS // copia in H cima dello stack  
idouble2 TOS = MDR = H + TOS // calcola il doppio, lo assegna a TOS e a MDR  
idouble3 MAR = SP; wr; goto Main1 // scrive nuova cima dello stack, va a Main1
```

Esercizio 5

- Si scriva il micro-codice che implementa per Mic-1 l'istruzione IJVM POPTWO (che rimuove due parole dalla cima dello stack).

Esercizio 5

- Si scriva il micro-codice che implementa per Mic-1 l'istruzione IJVM POPTWO (che rimuove due parole dalla cima dello stack).

```
poptwo1 SP = SP - 1 // Decrementa SP
poptwo2 MAR = SP = SP - 1; rd // Leggi la seconda parola in cima allo stack
poptwo3      // Aspetta un ciclo per avere in MDR la seconda parola
poptwo4 TOS = MDR; goto Main1 // Copia la nuova parola in TOS
```

Esercizio 5

- Si scriva il micro-codice che implementa per Mic-1 l'istruzione IJVM POPTWO (che rimuove due parole dalla cima dello stack).

```
poptwo1 SP = SP - 1; goto pop1 // Decrementa SP e vai ad eseguire la pop1
```

Esercizio 6

- Progettare una nuova istruzione JVM IXOR che sostituisce le due parole che si trovano in cima allo stack con il loro XOR (operazione logica bit a bit).

Esercizio 6

- Progettare una nuova istruzione IJVM IXOR che sostituisce le due parole che si trovano in cima allo stack con il loro XOR operazione logica bit a bit).
- In linguaggio MAL (micro-assembly language) sono definite le seguenti operazioni booleane: AND, OR, NOT. D'altra parte, l'operazione XOR è definita in forma normale disgiuntiva come:

$$A \text{ XOR } B = (\text{NOT}(A) \text{ AND } B) \text{ OR } (A \text{ AND } \text{NOT}(B)) \quad (1)$$

Quindi usiamo l'espressione di destra dell'equazione (1) e la realizziamo con il micro-codice

Esercizio 6

- Progettare una nuova istruzione IJVM IXOR che sostituisce le due parole che si trovano in cima allo stack con il loro XOR operazione logica bit a bit).

```
ixor1 MAR = SP = SP - 1; rd // Leggi la seconda parola sullo stack: a
ixor2 H = NOT TOS // assegna ad H il valore di NOT TOS: not(b)
ixor3 OPC = H AND MDR // OPC: not(b)*a
ixor4 H = NOT MDR // NOT H: not(a)
ixor5 H = H AND TOS // H: not(a)*b
ixor6 MDR = TOS = H OR OPC; wr; goto Main1 // MDR: not(a)*b+not(b)*a
      // scrivilo sul top dello stack, aggiorna TOS
```


Esercizio 7

- Si scriva il micro-codice che implementa per Mic-1 l'istruzione IJVM: IDIVVAR varnum, che divide per due (parte intera) il valore della variabile locale di indice varnum (un byte), aggiornando la variabile stessa

Esercizio 7

- Si scriva il micro-codice che implementa per Mic-1 l'istruzione IJVM: IDIVVAR varnum, che divide per due (parte intera) il valore della variabile locale di indice varnum (un byte), aggiornando la variabile stessa

[illegible]