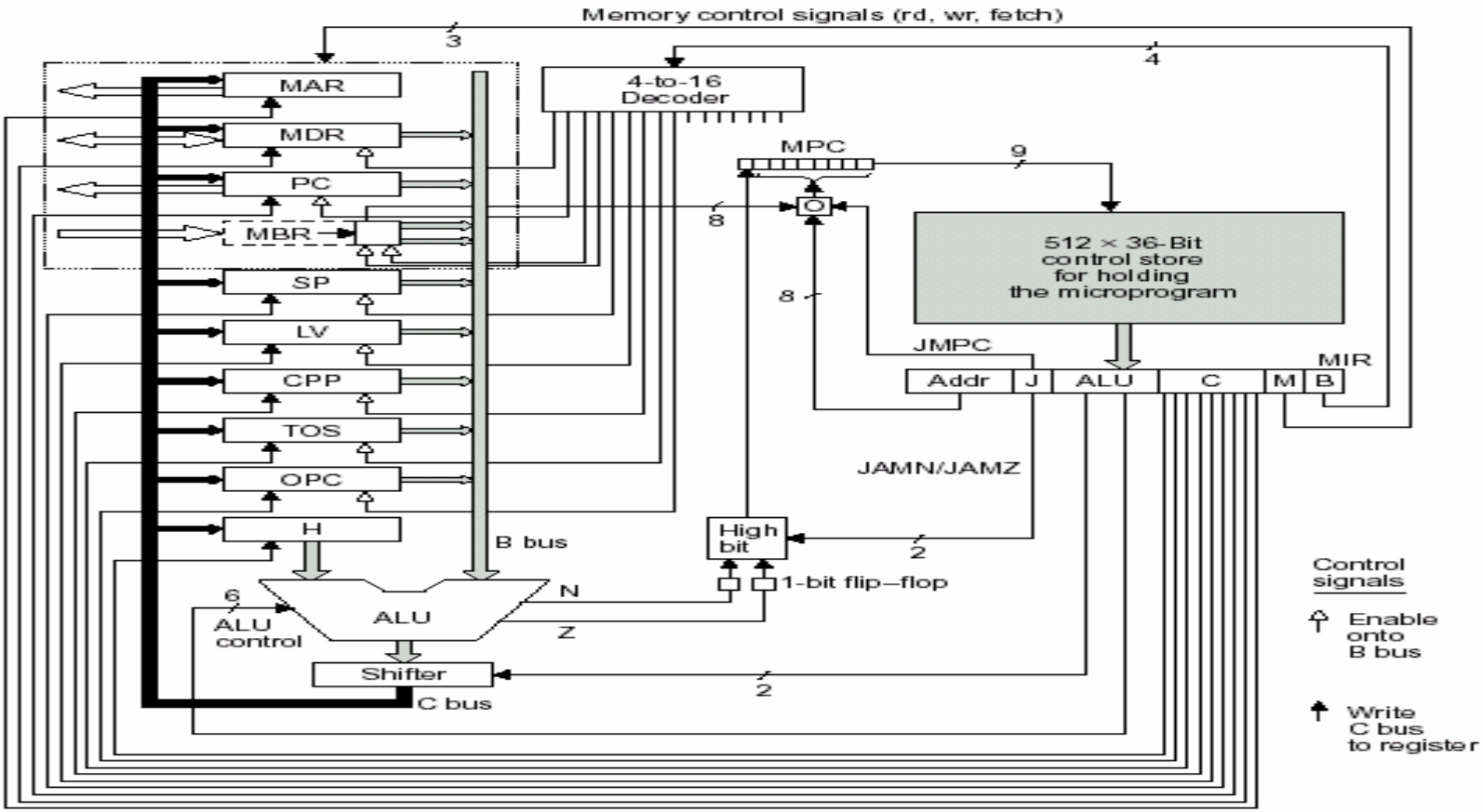
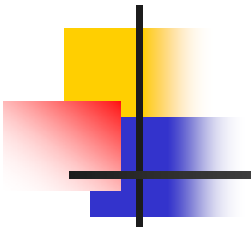
Esempio di ISA: IJVM

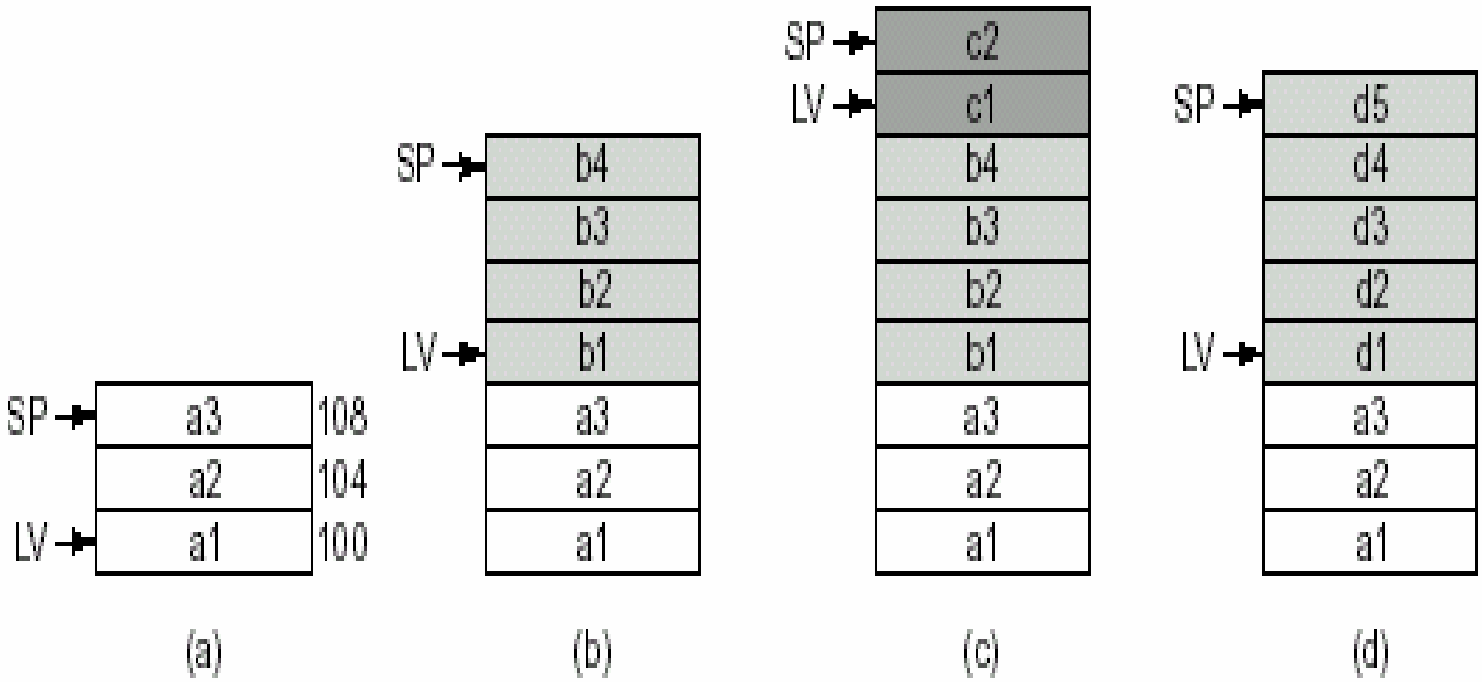




STACK

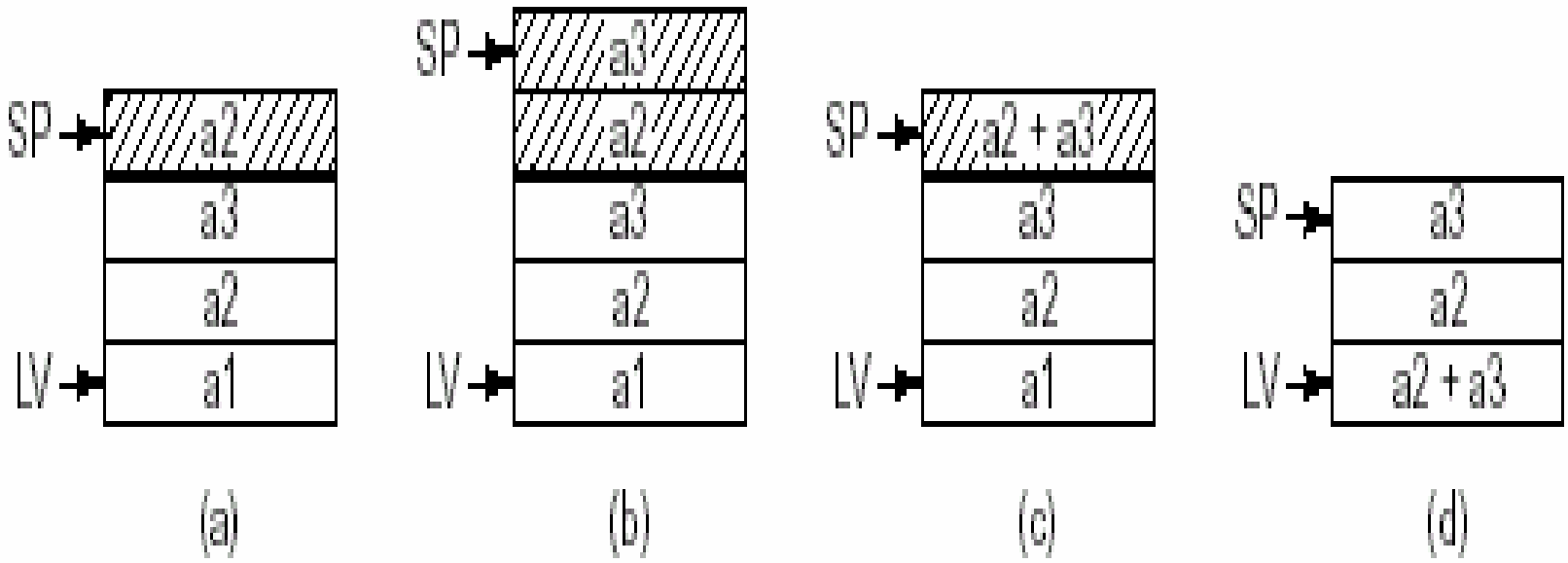
- Per memorizzare le variabili locali di una procedura useremo una parte della memoria (detta **stack**) senza assegnare indirizzi assoluti alle singole variabili.
- Il registro LV punta alla base delle variabili. Un altro registro SP punta in cima allo stack.
- Esempio
- procedura con tre variabili locali a1 ,a2, a3

Rappresentazione frame delle variabili locali

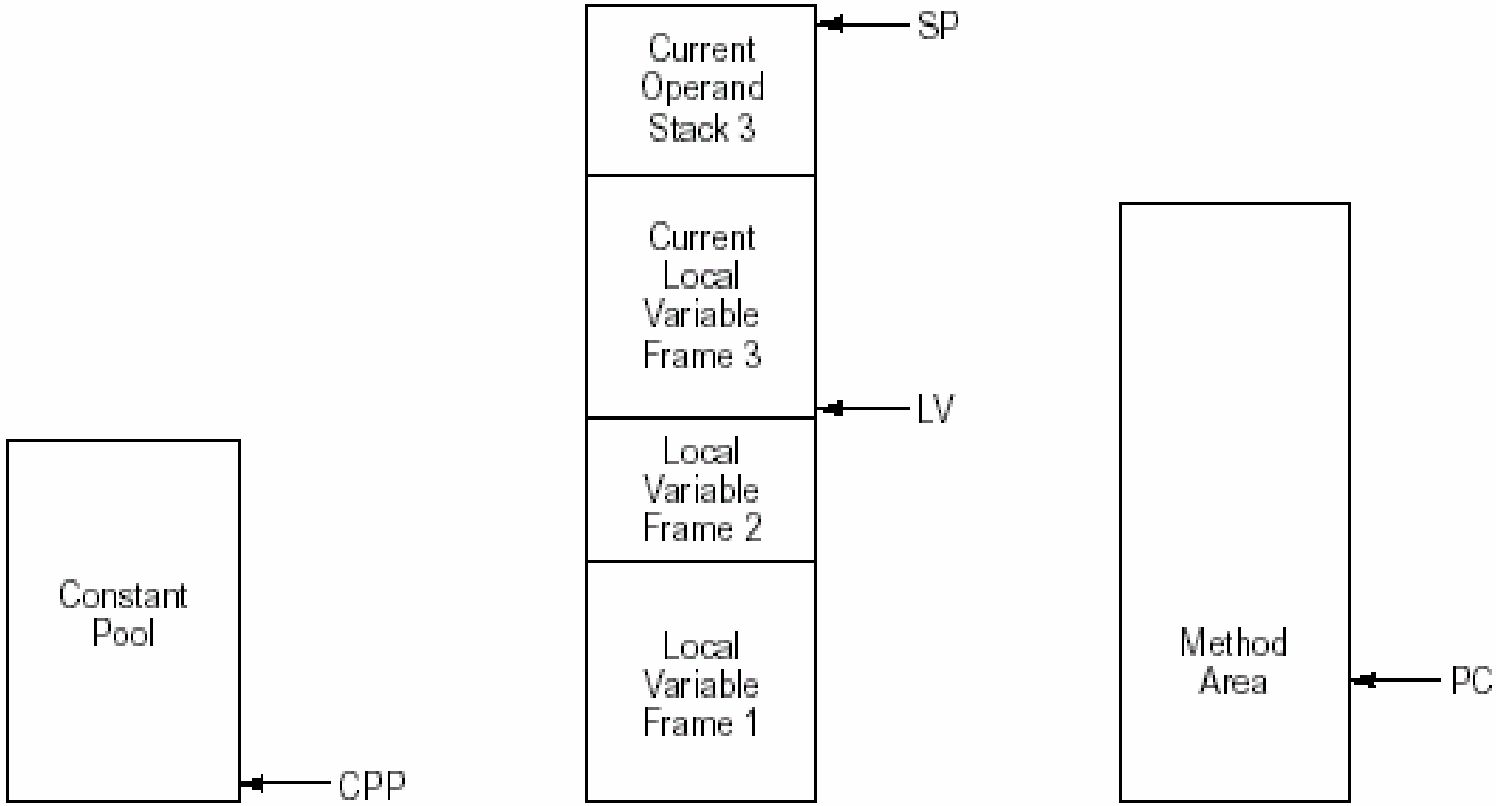


Operandi per il calcolo espressione

a1=a2 + a3



Modello memoria IJVM

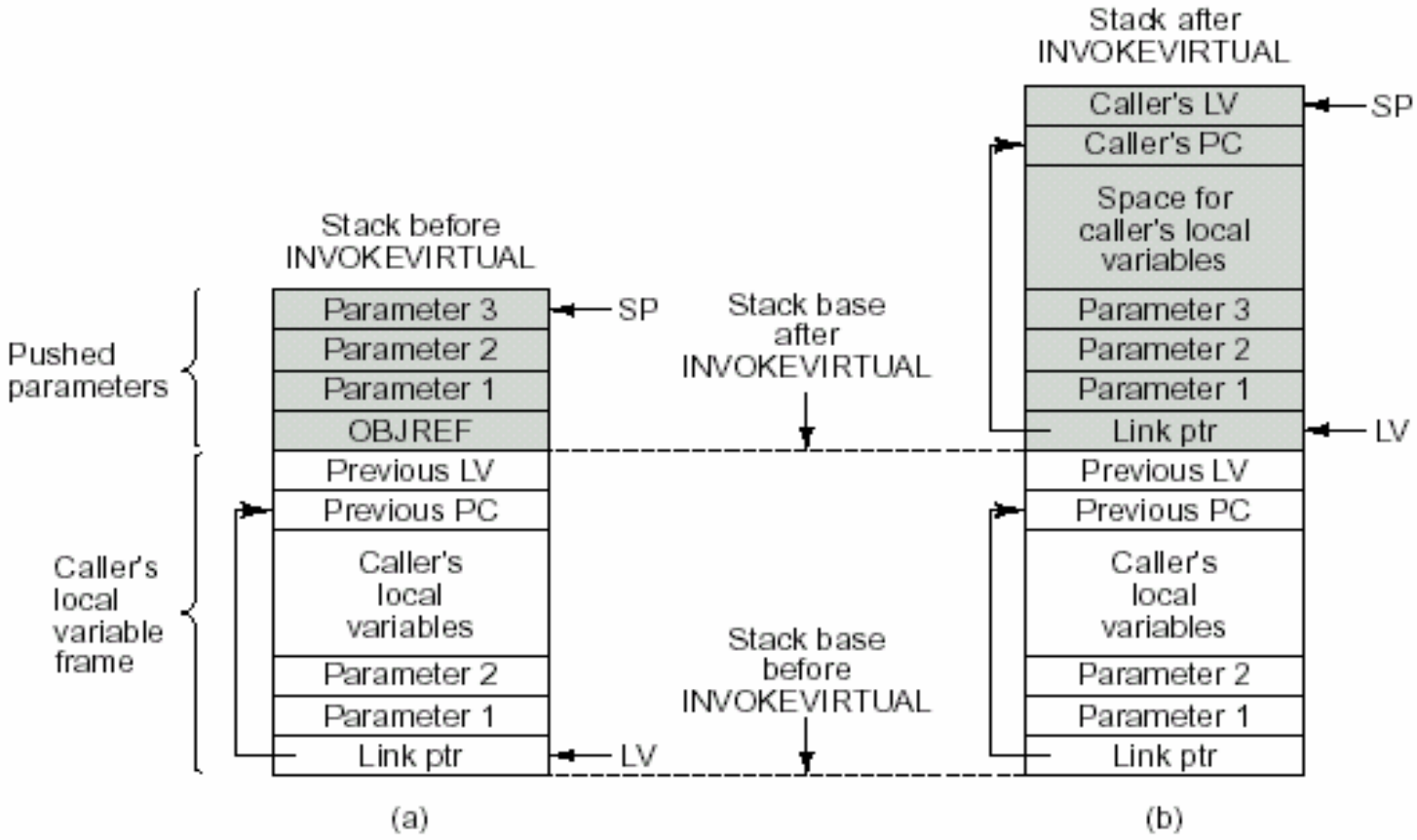


All’inizio del frame delle varabili locali si trovano i parametri con cui è stato chiamato il metodo

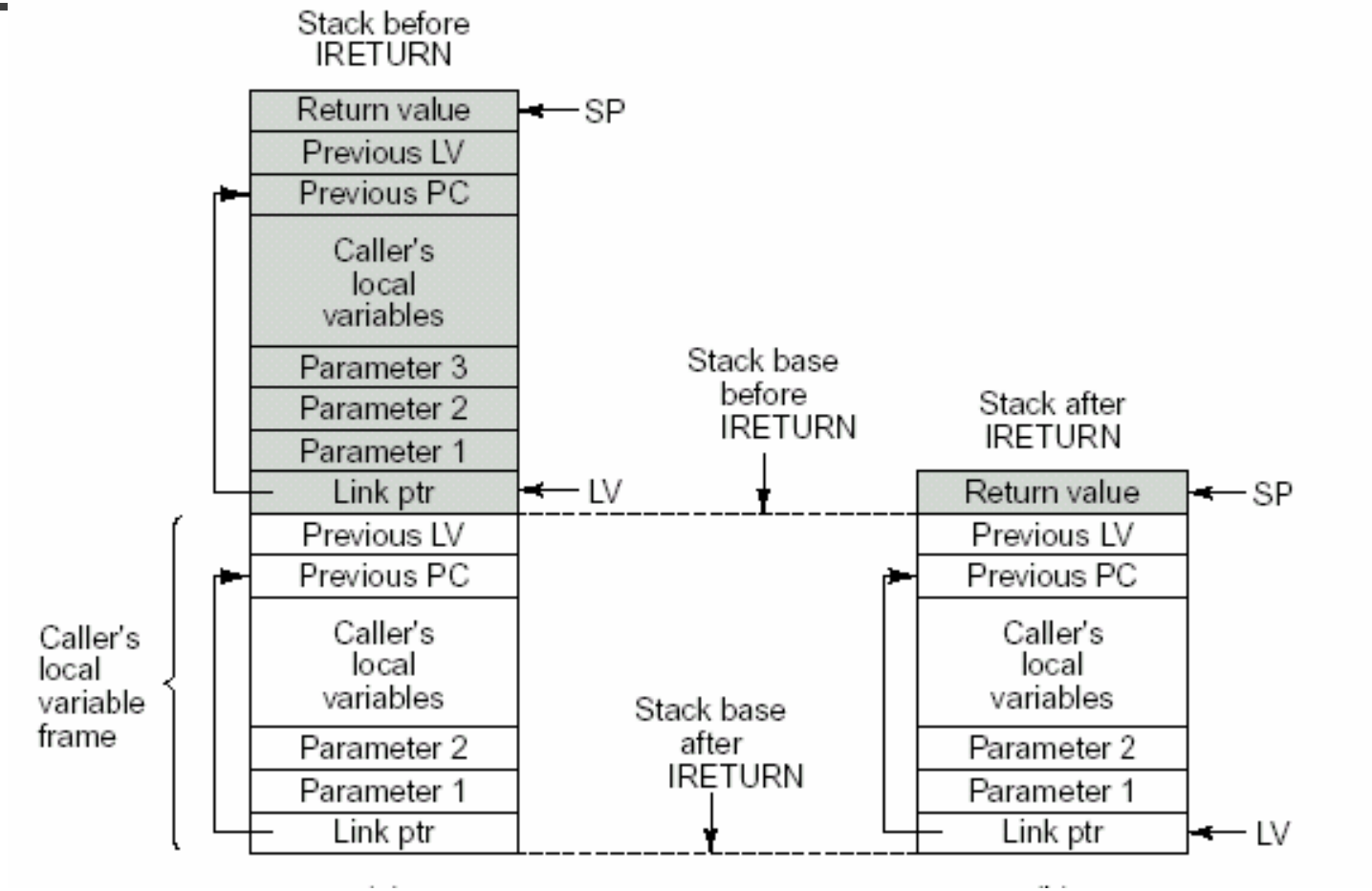
Instruction set di IJVM

Hex	Mnemonic	Meaning
0x10	BIPUSH *byte*	Push byte onto stack
0x59	DUP	Copy top word on stack and push onto stack
0xA7	GOTO *offset*	Unconditional branch
0x60	IADD	Pop two words from stack; push their sum
0x7E	IAND	Pop two words from stack; push Boolean AND
0x99	IFEQ *offset*	Pop word from stack and branch if it is zero
0x9B	IFLT *offset*	Pop word from stack and branch if it is less than zero
0x9F	IF_ICMPEQ *offset*	Pop two words from stack; branch if equal
0x84	IINC *varnum const*	Add a constant to a local variable
0x15	ILOAD *varnum*	Push local variable onto stack
0xB6	INVOKEVIRTUAL *disp*	Invoke a method
0x80	IOR	Pop two words from stack; push Boolean OR
0xAC	IRETURN	Return from method with integer value
0x36	ISTORE *varnum*	Pop word from stack and store in local variable
0x64	ISUB	Pop two words from stack; push their difference
0x13	LDC_W *index*	Push constant from constant pool onto stack
0x00	NOP	Do nothing
0x57	POP	Delete word on top of stack
0x5F	SWAP	Swap the two top words on the stack
0xC4	WIDE	Prefix instruction; next instruction has a 16-bit index

Gestione memoria con Invokevirtual



Gestione memoria con Invokevirtual (2)



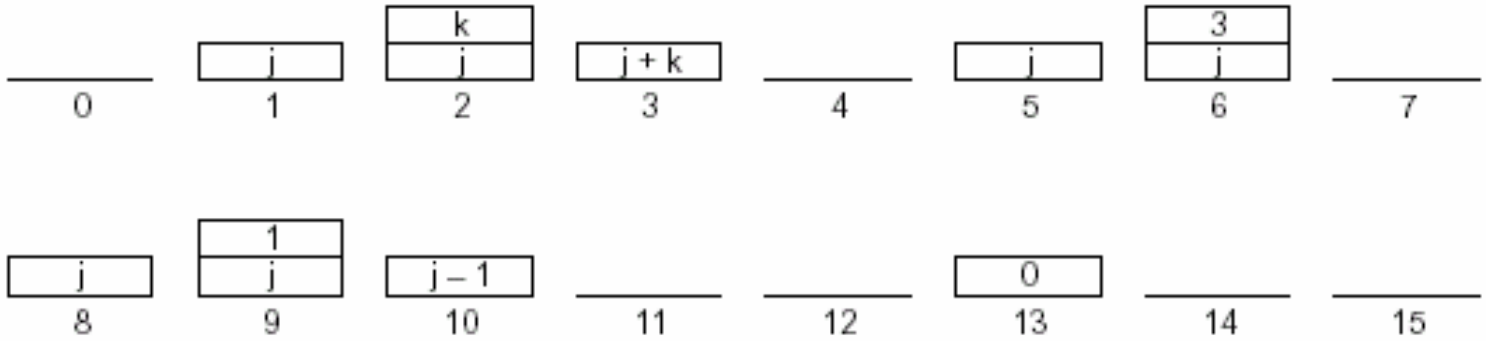
Compilazione Java per IJVM

```
i = j + k;              1    ILOAD j    // i = j + k     0x15 0x02
if (i == 3)             2    ILOAD k                     0x15 0x03
    k = 0;              3    IADD                        0x60
else                    4    ISTORE i                    0x36 0x01
   j = j – 1;           5    ILOAD i    // if (i < 3)    0x15 0x01
                        6    BIPUSH 3                    0x10 0x03
                        7    IF_ICMPEQ L1                0x9F 0x00 0x0D
                        8    ILOAD j    // j = j – 1     0x15 0x02
                        9    BIPUSH 1                    0x10 0x01
                       10    ISUB                        0x64
                       11    ISTORE j                    0x36 0x02
                       12    GOTO L2                     0xA7 0x00 0x07
                       13 L1:          BIPUSH 0          // k = 0 0x10 0x00
                       14    ISTORE k                    0x36 0x03
                       15 L2:
```

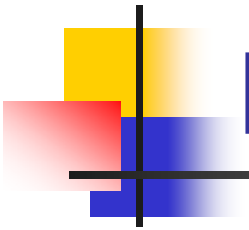
(a) (b) (c)

Figure 4-14. (a) A Java fragment. (b) The corresponding Java assembly language. (c) The IJVM program in hexadecimal.

Gestione Stack dell'esempio



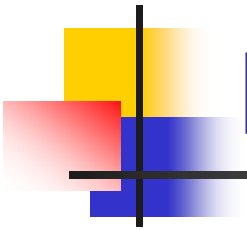
Esempio i=j+k+4



- .method calcola(j,k) // Calcola il valore di i=j+k+4.
- .var
- i
- .end-var

- ILOAD j
- ILOAD k
- IADD
- ISTORE i
- IINC i 4
- // in questo punto finisce la soluzione del problema il resto viene
- // utilizzato per produrre sullo standard I/O il risultato finale

- Iload i
- OUT // questa istruzione è utile soltanto per stampare sullo
- // standard I/O il risultato finale presente sulla variabile "i"



Esempio i=j+k+4 (2)

- Ireturn
- .end-method

- .main
- IN // permette di acquisire dallo Standard I/O il valore di j
- IN // permette di acquisire dallo Standard I/O il valore di k

- Invokevirtual calcola

- Halt
- .end-main

Microistruzioni: il linguaggio MAL

Incrementa SP, inizia lettura, prossima istruzione 122

ReadRegister =SP, ALU = INC, WSp, Readm Nextaddress =122

INVECE

SP = SP +1; eseguita in un ciclo

Durante ogni cliclo si può scrivere su qualsiasi registro

A può essere H, 0, 1 –1

Copia

MDR =SP

MDR = H + SP => SP + H

MAL: Operazioni legali

SOURCE =
MDR,PC,MBR,MBRU,SP,LV,
CPP, TOS, OPC

DEST=

MAR, MDR, PC, SP, LV,
CPP, TOS, OPC, H

DEST = H
DEST = SOURCE
DEST = $\overline{H}$
DEST = $\overline{SOURCE}$
DEST = H + SOURCE
DEST = H + SOURCE + 1
DEST = H + 1
DEST = SOURCE + 1
DEST = SOURCE – H
DEST = SOURCE – 1
DEST = –H
DEST = H AND SOURCE
DEST = H OR SOURCE
DEST = 0
DEST = 1
DEST = –1

MAL: operazioni illegali

MDR = SP + MDR

Non si può eseguire in un ciclo (H deve essere uno degli operandi)

H = H – MDR

Sorgente usabile come sootraendo è H

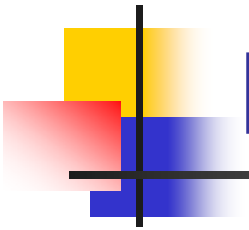
SP = MDR= SP +1

MAR = SP; rd

MDR = H

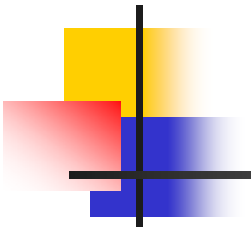
(sia la memoria che l'istruzione assegnano un valore a MDR)

MAL: istruzioni di salto



- Goto label (es. goto MAIN1)
- TOS=TOS (per verificare se è zero)
- Z = TOS
  If (Z) goto L1; else goto L2
- Z=TOS, If (Z) goto L1; else goto L2
- Goto (MBR)

Label	Operations	Comments
Main1	PC = PC + 1; fetch; goto (MBR)	MBR holds opcode; get next byte; dispatch
nop1	goto Main1	Do nothing
iadd1	MAR = SP = SP – 1; rd	Read in next-to-top word on stack
iadd2	H = TOS	H = top of stack
iadd3	MDR = TOS = MDR + H; wr; goto Main1	Add top two words; write to top of stack
isub1	MAR = SP = SP – 1; rd	Read in next-to-top word on stack
isub2	H = TOS	H = top of stack
isub3	MDR = TOS = MDR – H; wr; goto Main1	Do subtraction; write to top of stack
iand1	MAR = SP = SP – 1; rd	Read in next-to-top word on stack
iand2	H = TOS	H = top of stack
iand3	MDR = TOS = MDR AND H; wr; goto Main1	Do AND; write to new top of stack
ior1	MAR = SP = SP – 1; rd	Read in next-to-top word on stack
ior2	H = TOS	H = top of stack
ior3	MDR = TOS = MDR OR H; wr; goto Main1	Do OR; write to new top of stack
dup1	MAR = SP = SP + 1	Increment SP and copy to MAR
dup2	MDR = TOS; wr; goto Main1	Write new stack word
pop1	MAR = SP = SP – 1; rd	Read in next-to-top word on stack
pop2		Wait for new TOS to be read from memory
pop3	TOS = MDR; goto Main1	Copy new word to TOS
swap1	MAR = SP – 1; rd	Set MAR to SP – 1; read 2nd word from stack
swap2	MAR = SP	Set MAR to top word
swap3	H = MDR; wr	Save TOS in H; write 2nd word to top of stack
swap4	MDR = TOS	Copy old TOS to MDR
swap5	MAR = SP – 1; wr	Set MAR to SP – 1; write as 2nd word on stack
swap6	TOS = H; goto Main1	Update TOS
bipush1	SP = MAR = SP + 1	MBR = the byte to push onto stack
bipush2	PC = PC + 1; fetch	Increment PC, fetch next opcode
bipush3	MDR = TOS = MBR; wr; goto Main1	Sign-extend constant and push on stack



iload1	H = LV	MBR contains index; copy LV to H
iload2	MAR = MBRU + H; rd	MAR = address of local variable to push
iload3	MAR = SP = SP + 1	SP points to new top of stack; prepare write
iload4	PC = PC + 1; fetch; wr	Inc PC; get next opcode; write top of stack
iload5	TOS = MDR; goto Main1	Update TOS
istore1	H = LV	MBR contains index; Copy LV to H
istore2	MAR = MBRU + H	MAR = address of local variable to store into
istore3	MDR = TOS; wr	Copy TOS to MDR; write word
istore4	SP = MAR = SP – 1; rd	Read in next-to-top word on stack
istore5	PC = PC + 1; fetch	Increment PC; fetch next opcode
istore6	TOS = MDR; goto Main1	Update TOS

Istruzione POP: Microprogramma

Label	Operations	Comments
pop1	MAR = SP = SP – 1; rd	Read in next-to-top word on stack
pop2		Wait for new TOS to be read from memory
pop3	TOS = MDR; goto Main1	Copy new word to TOS
Main1	PC = PC + 1; fetch; goto (MBR)	MBR holds opcode; get next byte; dispatch

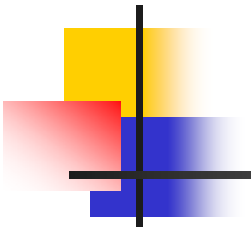
Ottimizzazione POP

Includere il ciclo dell'interprete al termine di ogni sequenza di microcodice

Label	Operations	Comments
pop1	MAR = SP = SP – 1; rd	Read in next-to-top word on stack
Main1.pop	PC = PC + 1; fetch	MBR holds opcode; fetch next byte
pop3	TOS = MDR; goto (MBR)	Copy new word to TOS; dispatch on opcode

Con la stessa tecnica IADD passa da 4 cicli a tre

Incremento del clock da 250 MHz (4) a 333 MHz (3)



Architettura con tre bus

Label	Operations	Comments
iload1	H = LV	MBR contains index; Copy LV to H
iload2	MAR = MBRU + H; rd	MAR = address of local variable to push
iload3	MAR = SP = SP + 1	SP points to new top of stack; prepare write
iload4	PC = PC + 1; fetch; wr	Inc PC; get next opcode; write top of stack
iload5	TOS = MDR; goto Main1	Update TOS
Main1	PC = PC + 1; fetch; goto (MBR)	MBR holds opcode; get next byte; dispatch

Somma fra registri in un ciclo

Iload con architettura a tre bus

Label	Operations	Comments
iload1	MAR = MBRU + LV; rd	MAR = address of local variable to push
iload2	MAR = SP = SP + 1	SP points to new top of stack; prepare write
iload3	PC = PC + 1; fetch; wr	Inc PC; get next opcode; write top of stack
iload4	TOS = MDR	Update TOS
iload5	PC = PC + 1; fetch; goto (MBR)	MBR already holds opcode; fetch index byte

Riduzione da 6 a 5 cicli

Unità per il fetch delle istruzioni

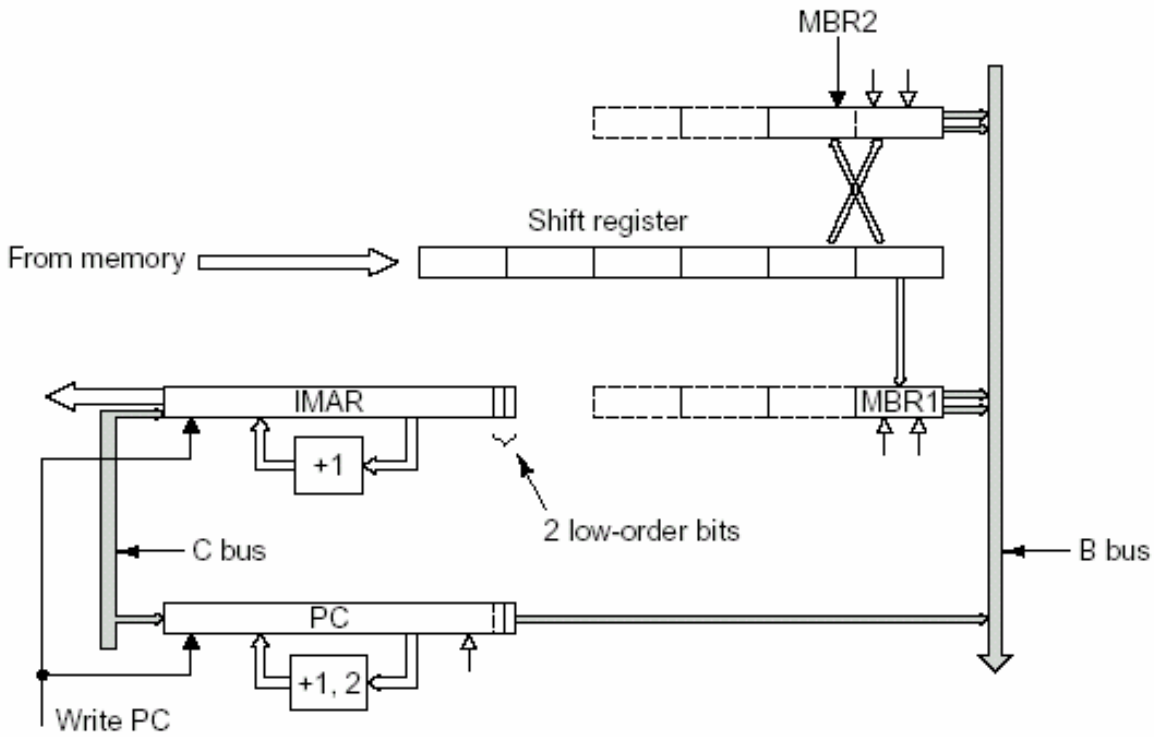
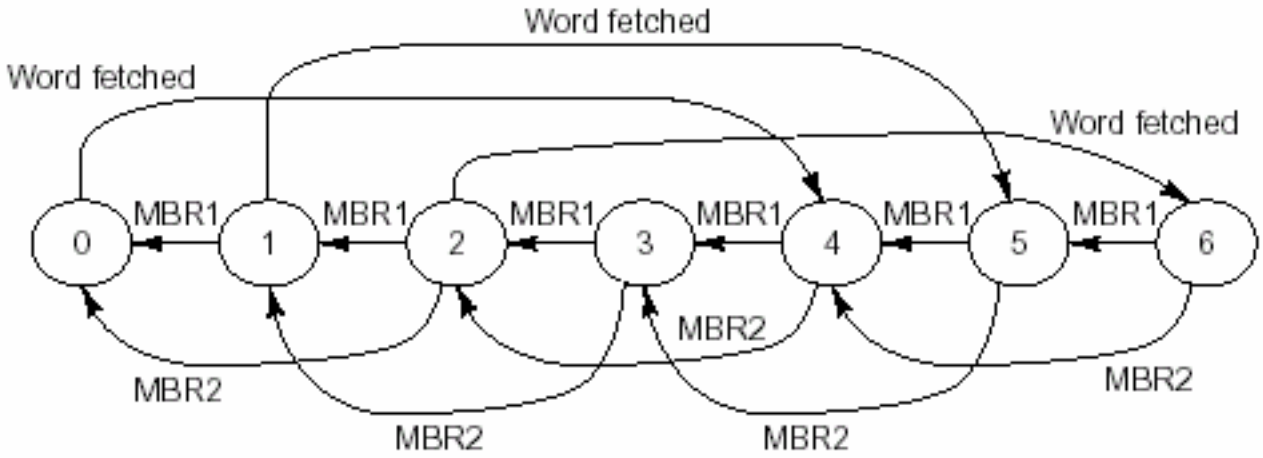


Figure 4-27. A fetch unit for the Mic-1.

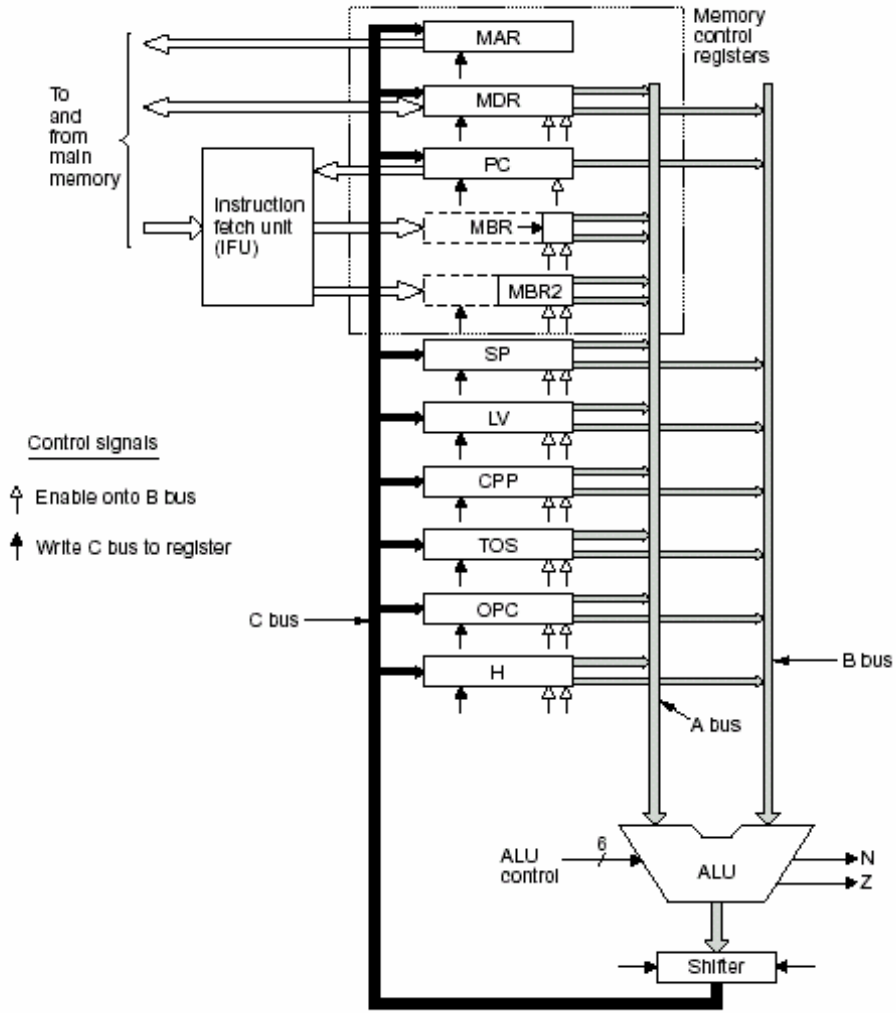
Automa a stati finiti del funzionamento del registro shift



Transitions

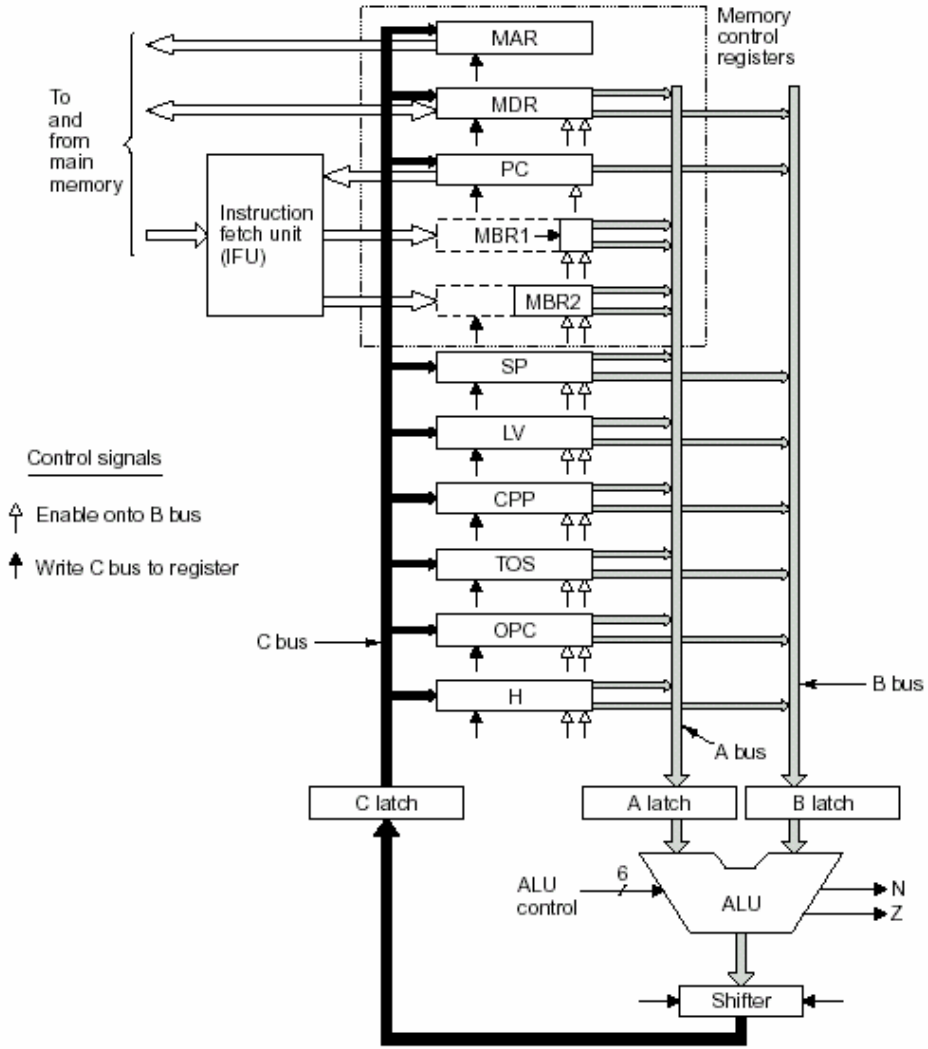
MBR1: Occurs when MBR1 is read
MBR2: Occurs when MBR2 is read
Word fetched: Occurs when a memory word is read and 4 bytes are put into the shift register

Architettura con prefetch e tre bus



Label	Operations	Comments
nop1	goto (MBR)	Branch to next instruction
iadd1	MAR = SP = SP – 1; rd	Read in next-to-top word on stack
iadd2	H = TOS	H = top of stack
iadd3	MDR = TOS = MDR+H; wr; goto (MBR1)	Add top two words; write to new top of stack
isub1	MAR = SP = SP – 1; rd	Read in next-to-top word on stack
isub2	H = TOS	H = top of stack
isub3	MDR = TOS = MDR–H; wr; goto (MBR1)	Subtract TOS from Fetched TOS-1
iand1	MAR = SP = SP – 1; rd	Read in next-to-top word on stack
iand2	H = TOS	H = top of stack
iand3	MDR = TOS = MDR AND H; wr; goto (MBR1)	AND Fetched TOS-1 with TOS
ior1	MAR = SP = SP – 1; rd	Read in next-to-top word on stack
ior2	H = TOS	H = top of stack
ior3	MDR = TOS = MDR OR H; wr; goto (MBR1)	OR Fetched TOS-1 with TOS
dup1	MAR = SP = SP + 1	Increment SP; copy to MAR
dup2	MDR = TOS; wr; goto (MBR1)	Write new stack word
pop1	MAR = SP = SP – 1; rd	Read in next-to-top word on stack
pop2		Wait for read
pop3	TOS = MDR; goto (MBR1)	Copy new word to TOS
swap1	MAR = SP – 1; rd	Read 2nd word from stack; set MAR to SP
swap2	MAR = SP	Prepare to write new 2nd word
swap3	H = MDR; wr	Save new TOS; write 2nd word to stack
swap4	MDR = TOS	Copy old TOS to MDR
swap5	MAR = SP – 1; wr	Write old TOS to 2nd place on stack
swap6	TOS = H; goto (MBR1)	Update TOS
bipush1	SP = MAR = SP + 1	Set up MAR for writing to new top of stack
bipush2	MDR = TOS = MBR1; wr; goto (MBR1)	Update stack in TOS and memory
iload1	MAR = LV + MBR1U; rd	Move LV + index to MAR; read operand
iload2	MAR = SP = SP + 1	Increment SP; Move new SP to MAR
iload3	TOS = MDR; wr; goto (MBR1)	Update stack in TOS and memory

Architettura pipeline



Istruzione SWAP

Label	Operations	Comments
swap1	MAR = SP – 1; rd	Read 2nd word from stack; set MAR to SP
swap2	MAR = SP	Prepare to write new 2nd word
swap3	H = MDR; wr	Save new TOS; write 2nd word to stack
swap4	MDR = TOS	Copy old TOS to MDR
swap5	MAR = SP – 1; wr	Write old TOS to 2nd place on stack
swap6	TOS = H; goto (MBR1)	Update TOS

SWAP con architettura pipeline

	Swap1	Swap2	Swap3	Swap4	Swap5	Swap6
Cy	MAR=SP–1;rd	MAR=SP	H=MDR;wr	MDR=TOS	MAR=SP–1;wr	TOS=H;goto (MBR1)
1	B=SP					
2	C=B–1	B=SP				
3	MAR=C; rd	C=B				
4	MDR=mem	MAR=C				
5			B=MDR			
6			C=B	B=TOS		
7			H=C; wr	C=B	B=SP	
8			Mem=MDR	MDR=C	C=B–1	B=H
9					MAR=C; wr	C=B
10					Mem=MDR	TOS=C
11						goto (MBR1)

Figure 4-33. The implementation of SWAP on the Mic-3.

Funzionamento pipeline