Dalle istruzioni alle microoperazioni

Il funzionamento della CU

SAPIENZA
UNIVERSITÀ DI ROMA

Alessandro Pellegrini

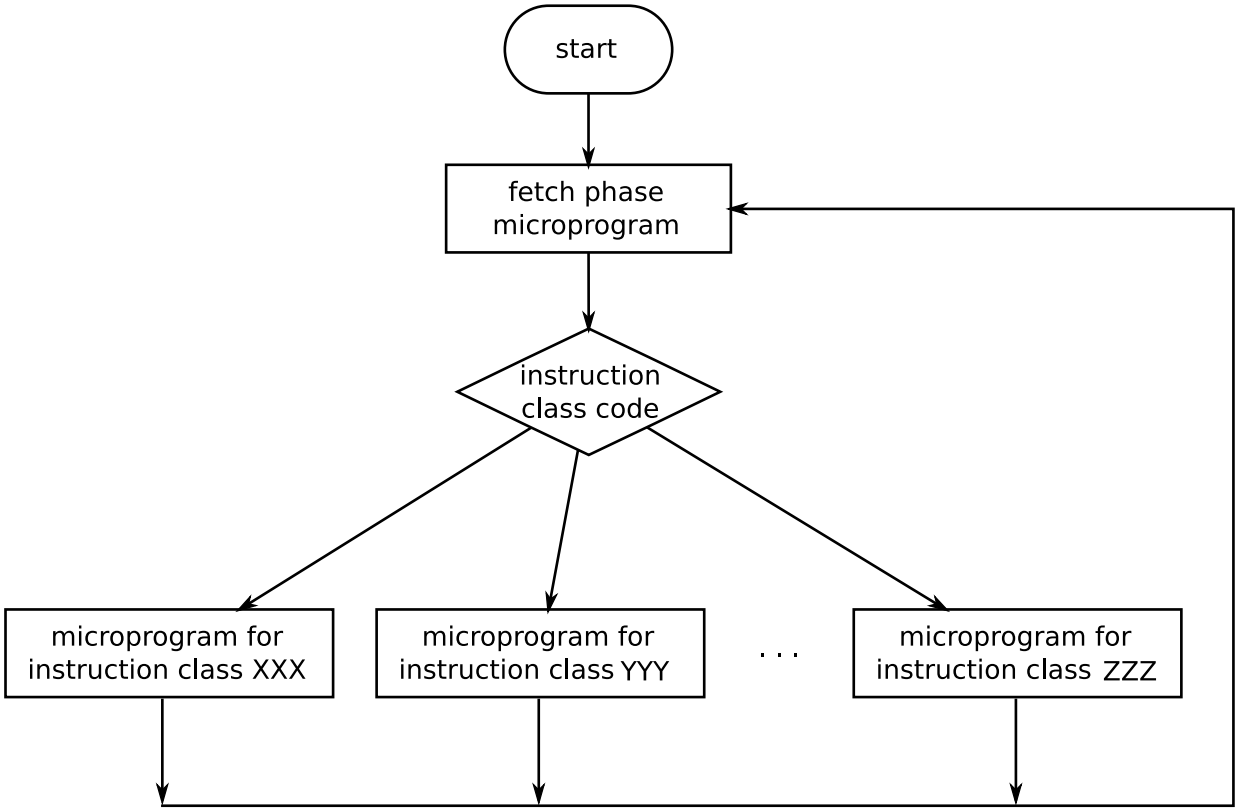
Architettura dei Calcolatori Elettronici
Sapienza, Università di Roma

A.A. 2018/2019

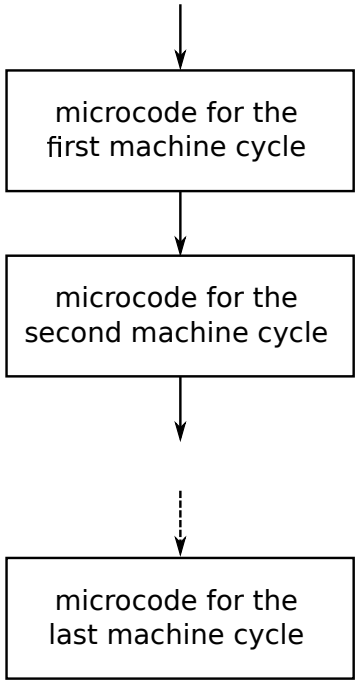
Le microoperazioni

- L'implementazione dello z64 che stiamo studiando è *multiciclo*
- L'esecuzione di un'istruzione è divisa in più fasi (cicli macchina)
- In un singolo ciclo macchina, il processore può non essere in grado di eseguire tutte le operazioni associate
  - Ad esempio, un'istruzione `ADD` richiede il movimento dei dati verso la ALU, il calcolo del risultato e la riscrittura del risultato
- La CU implementa un'istruzione tramite una serie di microoperazioni, ciascuna eseguita in un ciclo macchina

Realizzazione della CU



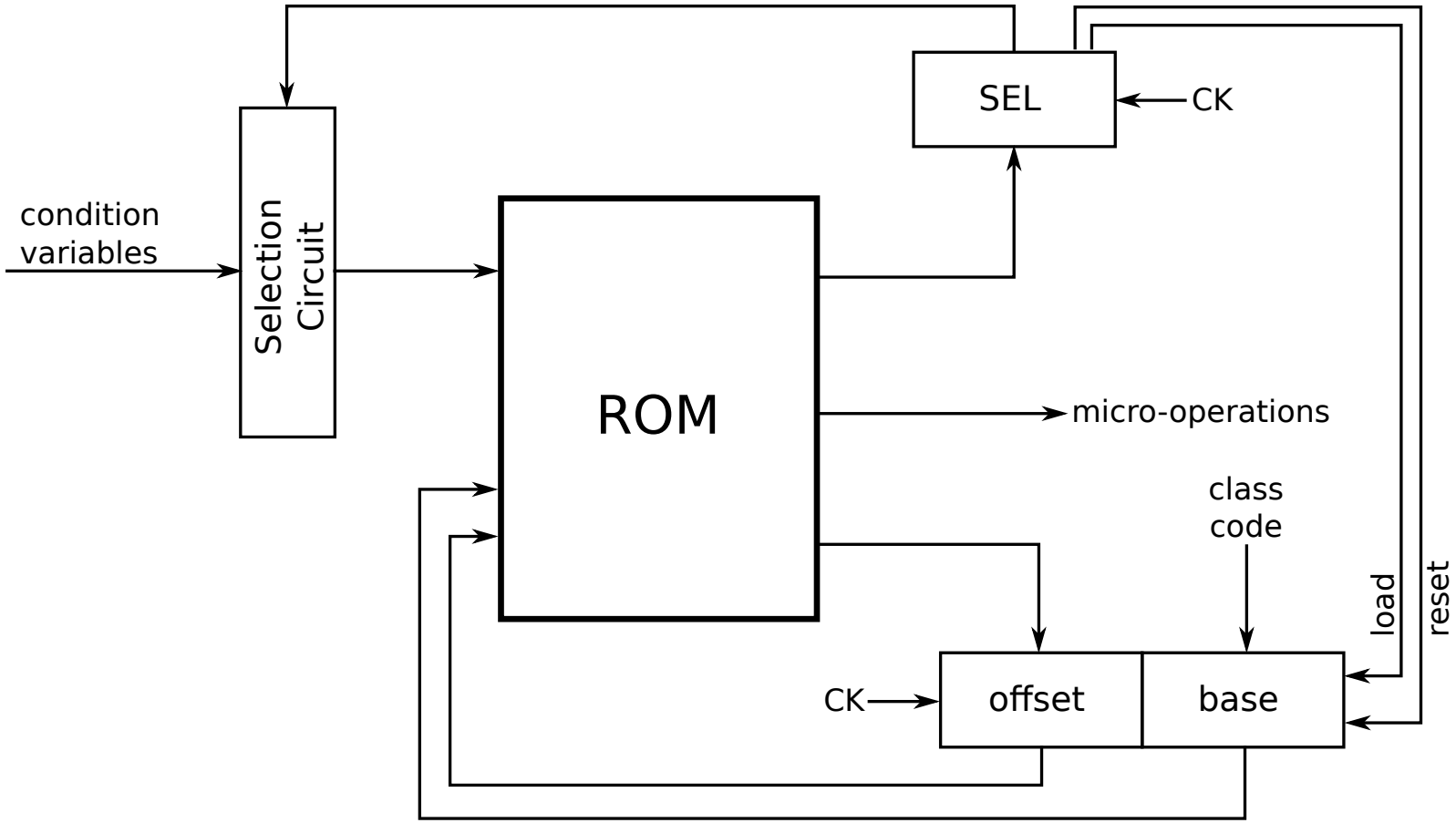
Microcodice per una classe di istruzioni



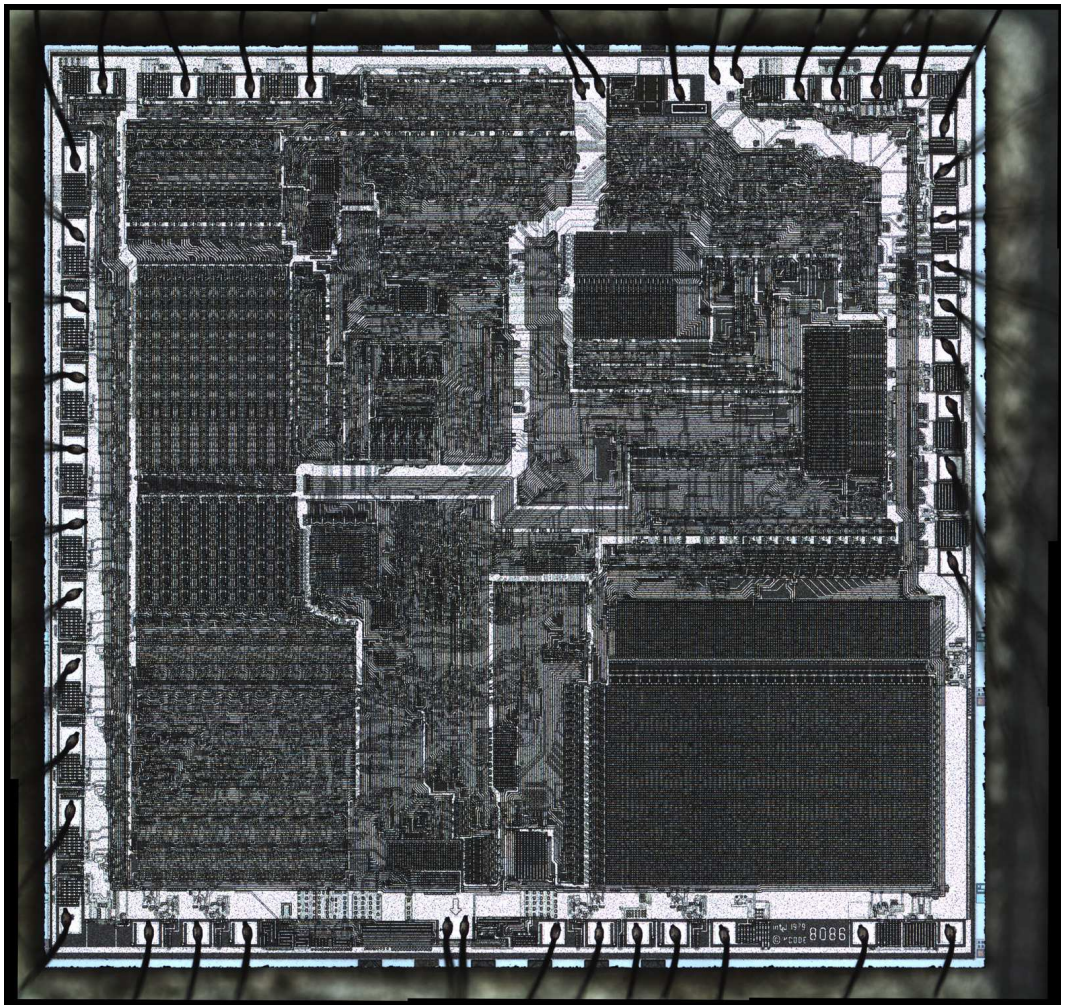
Organizzazione della CU

- Per eseguire il microprogramma di un'istruzione, la CU userà come input:
  - La classe e il tipo dell'istruzione — contenuta in IR
  - Le variabili di condizione che arrivano da fuori la CPU
  - Le variabili di condizione che arrivano da dentro la CPU — ad esempio i bit di $\textsc{Flags}$
  - La modalità di indirizzamento degli operandi dell'istruzione — dedotta da IR
- Il numero di segnali di output della CU dipende dall'implementazione della PU e dei moduli esterni
- L'organizzazione della CU dipende dal costo implmentativo, dalla performance desiderata e dal tipo di macchina a stati finiti scela (Mealy o Moore)

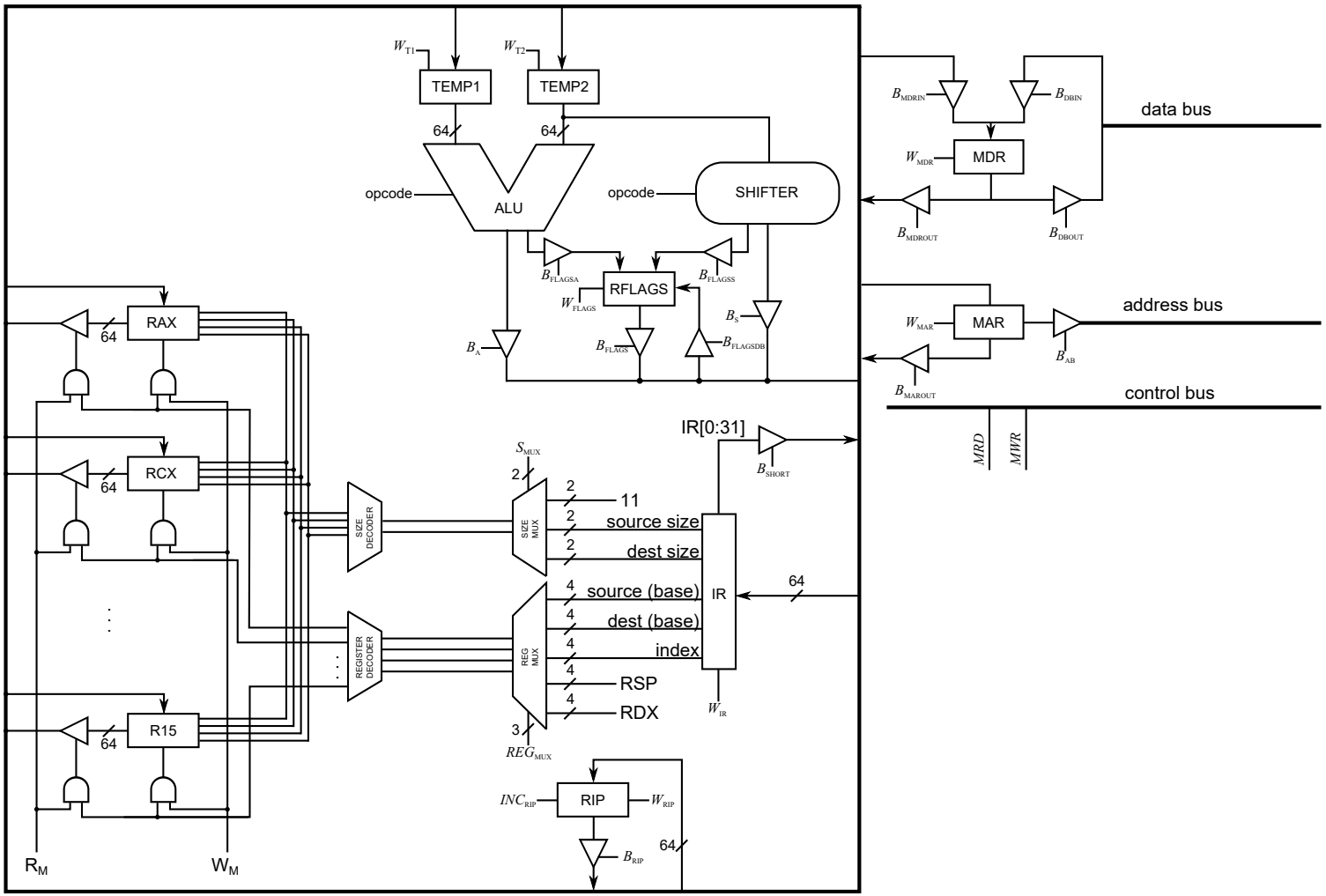
La CU come macchina di Mealy



Qual è il costo del microcodice?



z64: Architettura completa

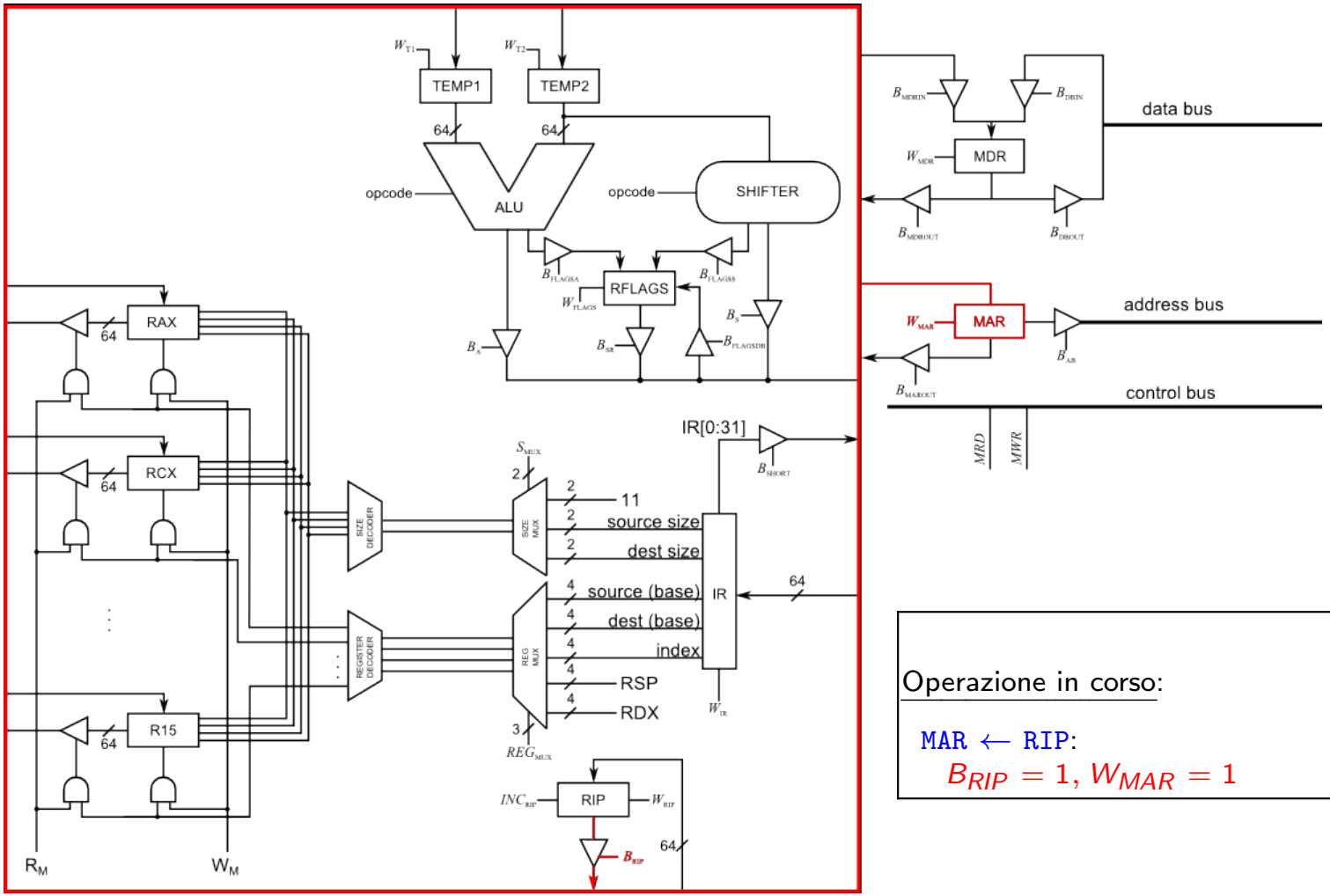


La fase di Fetch

- Ogni operazione incomincia con la fase di fetch
- Le microoperazioni associate alla fase di fetch sono:
  - `MAR` $\leftarrow$ `RIP`
  - `MDR` $\leftarrow$ (`MAR`); `RIP` $\leftarrow$ `RIP + 8`
  - `IR` $\leftarrow$ `MDR`

- In questo modo, l'istruzione successiva viene caricata nel registro `IR` (così da poterla interpretare ed eseguire) e il valore di `RIP` viene incrementato (così da puntare alla prossima istruzione/dato)

La fase di Fetch

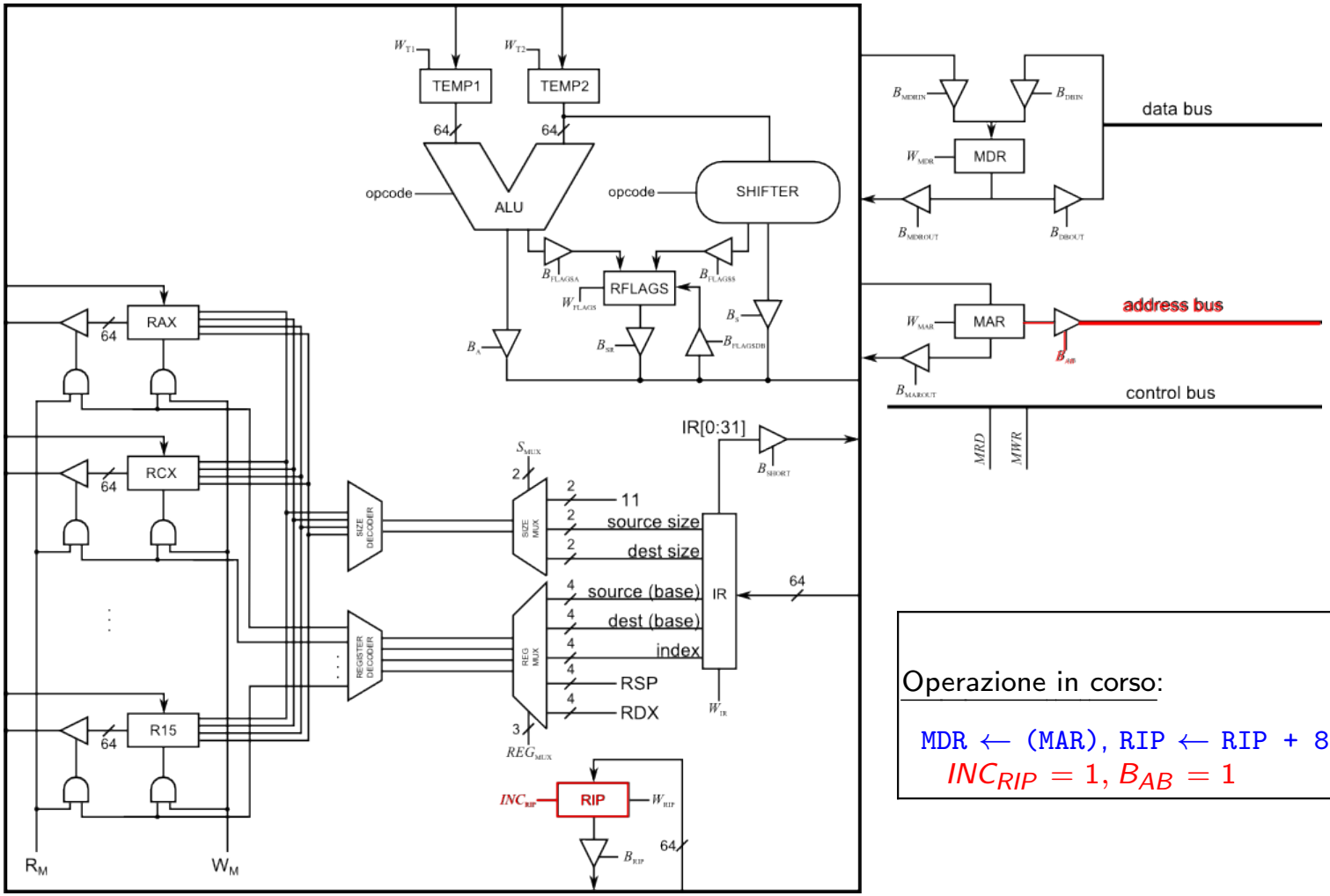


Operazione in corso:

MAR ← RIP:

$B_{RIP} = 1, W_{MAR} = 1$

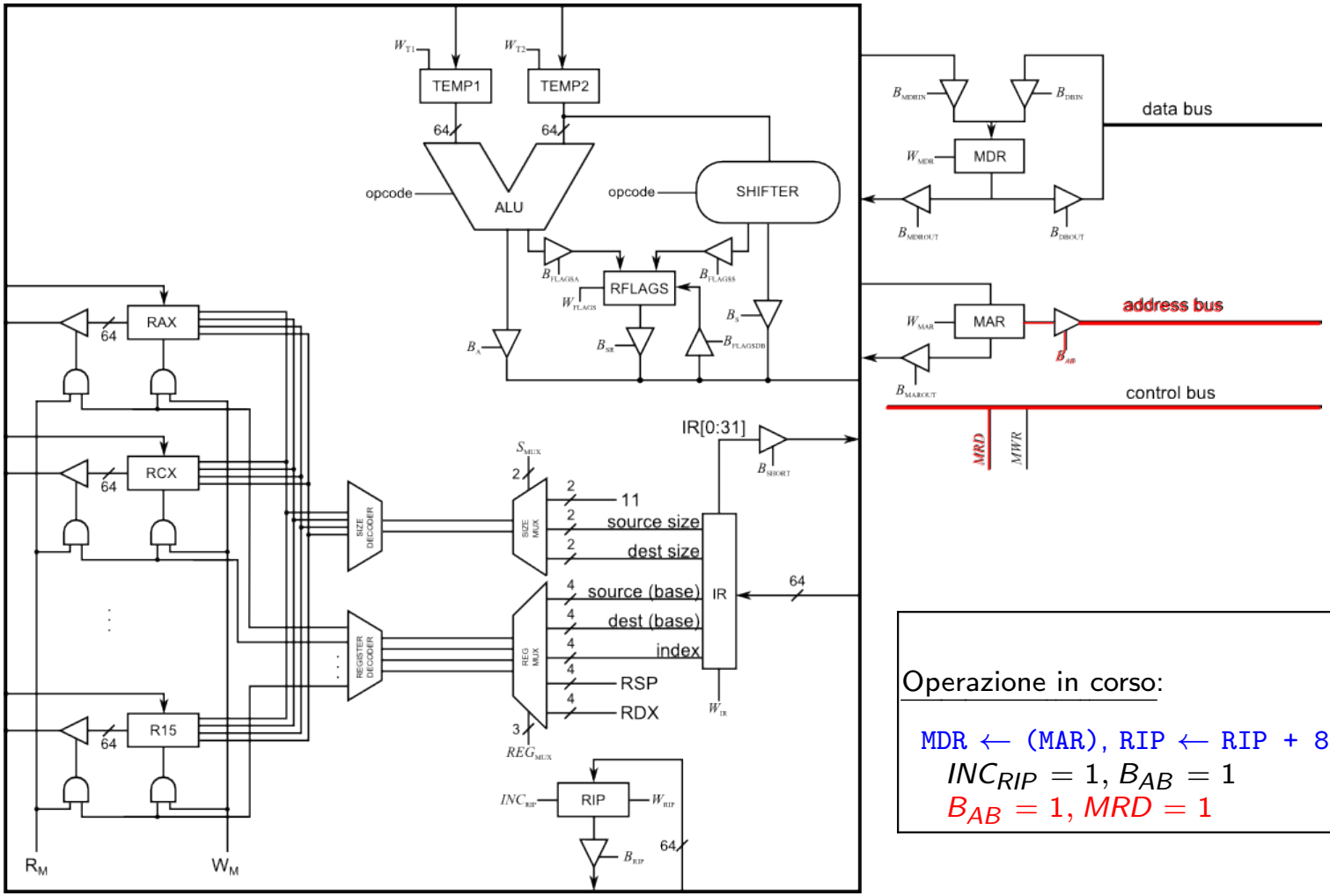
La fase di Fetch



Operazione in corso:

`MDR ← (MAR), RIP ← RIP + 8`:
$INC_{RIP} = 1, B_{AB} = 1$

La fase di Fetch



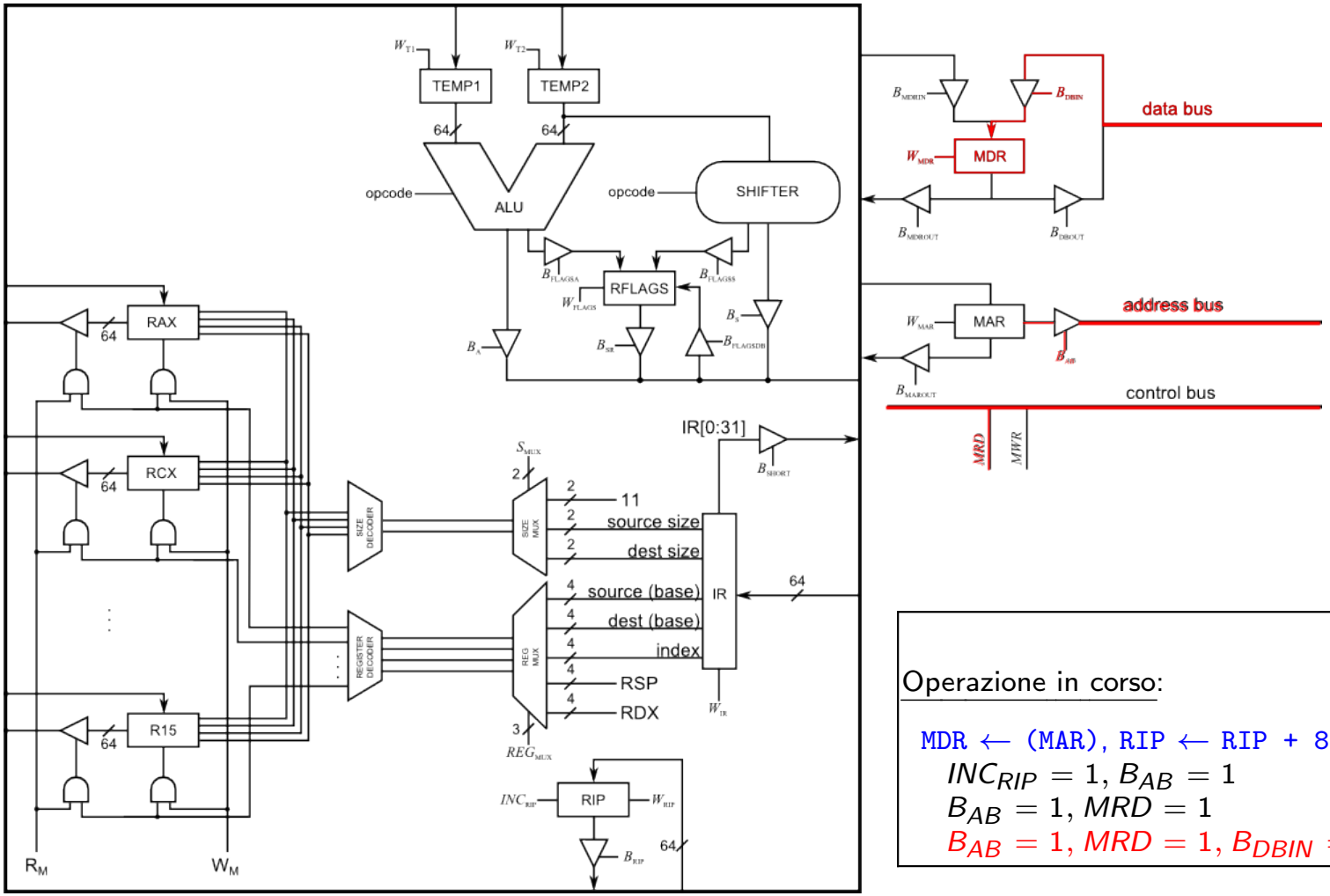
Operazione in corso:

`MDR ← (MAR), RIP ← RIP + 8`:

$INC_{RIP} = 1, B_{AB} = 1$

$B_{AB} = 1, MRD = 1$

La fase di Fetch



Operazione in corso:

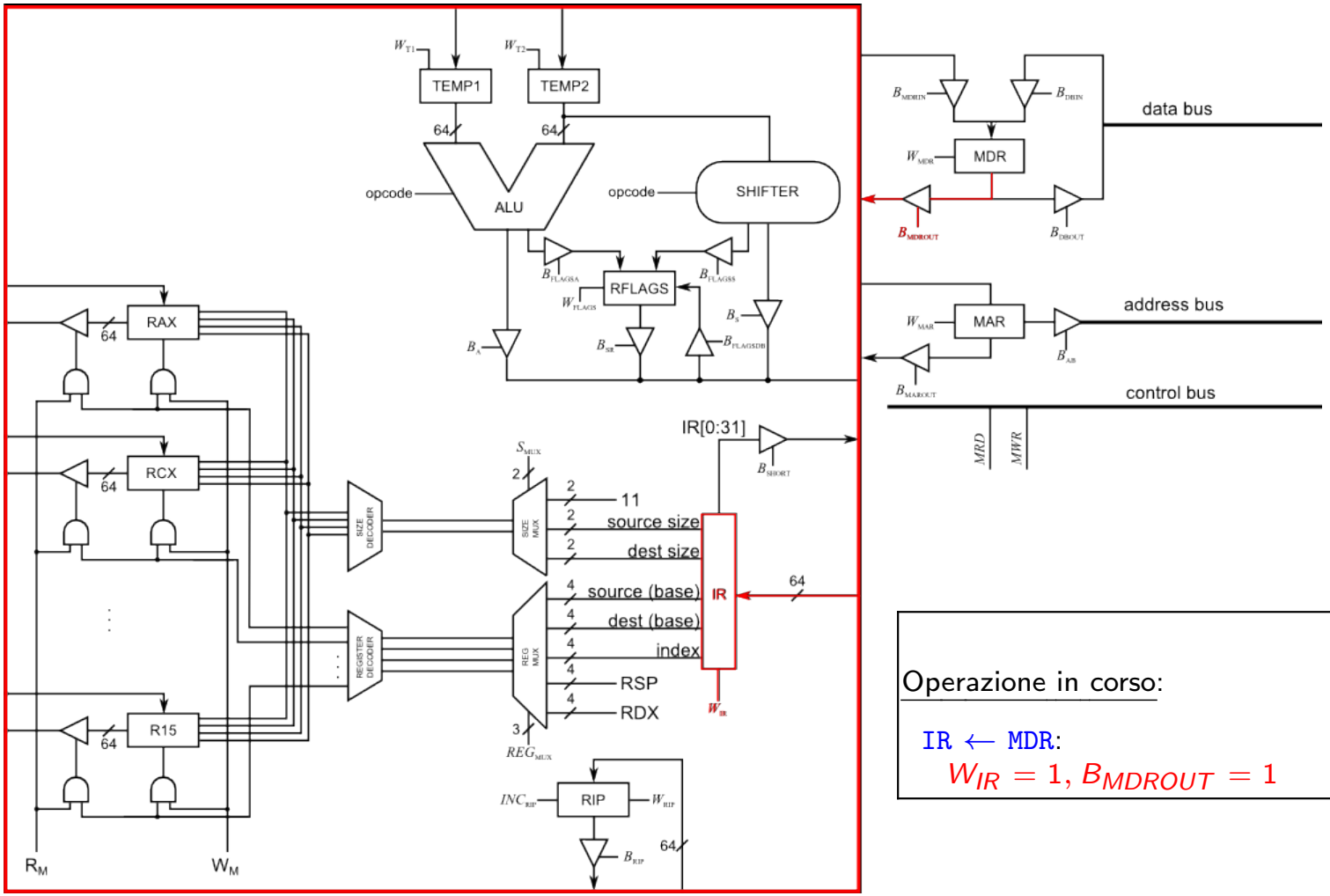
MDR ← (MAR), RIP ← RIP + 8:

$INC_{RIP} = 1, B_{AB} = 1$

$B_{AB} = 1, MRD = 1$

$B_{AB} = 1, MRD = 1, B_{DBIN} = 1, W_{MDR} = 1$

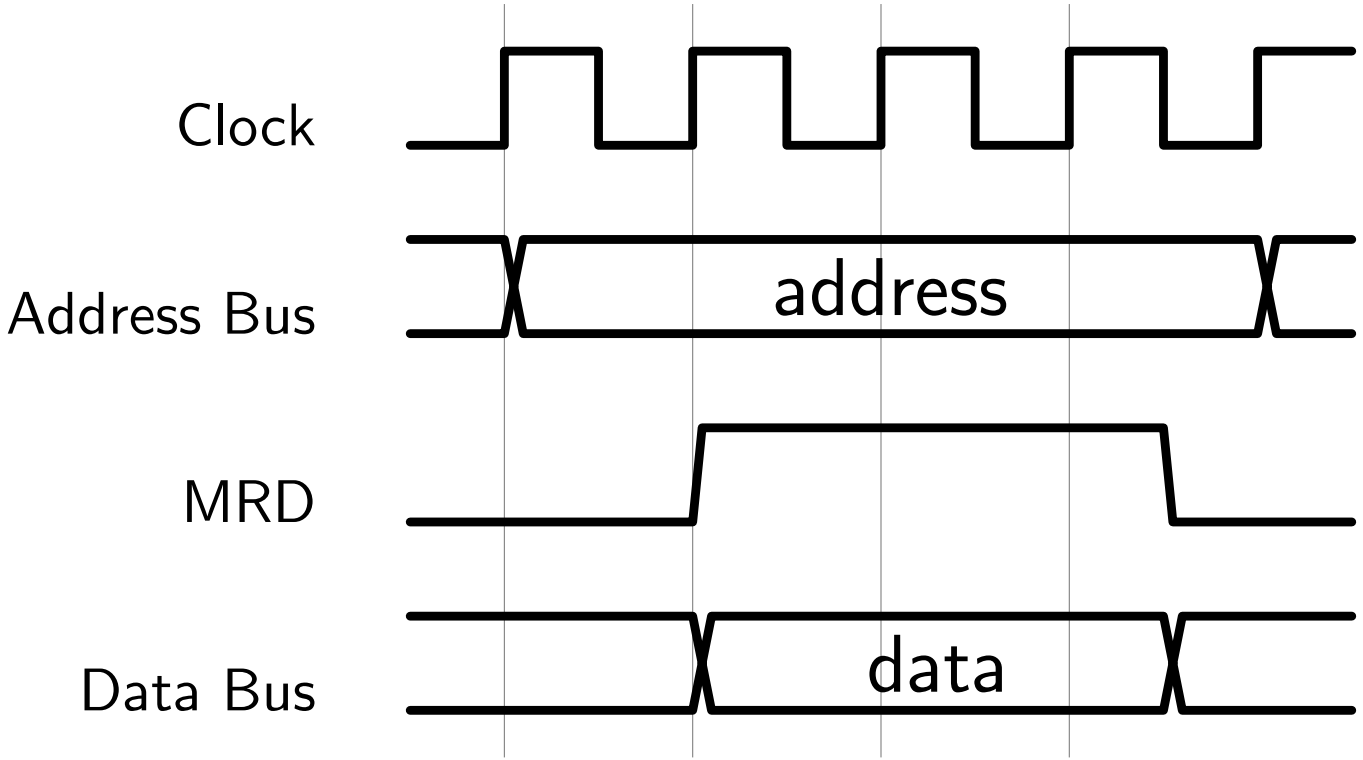
La fase di Fetch



Operazione in corso:

IR ← MDR:
$W_{IR} = 1, B_{MDROUT} = 1$

Interazione con la memoria

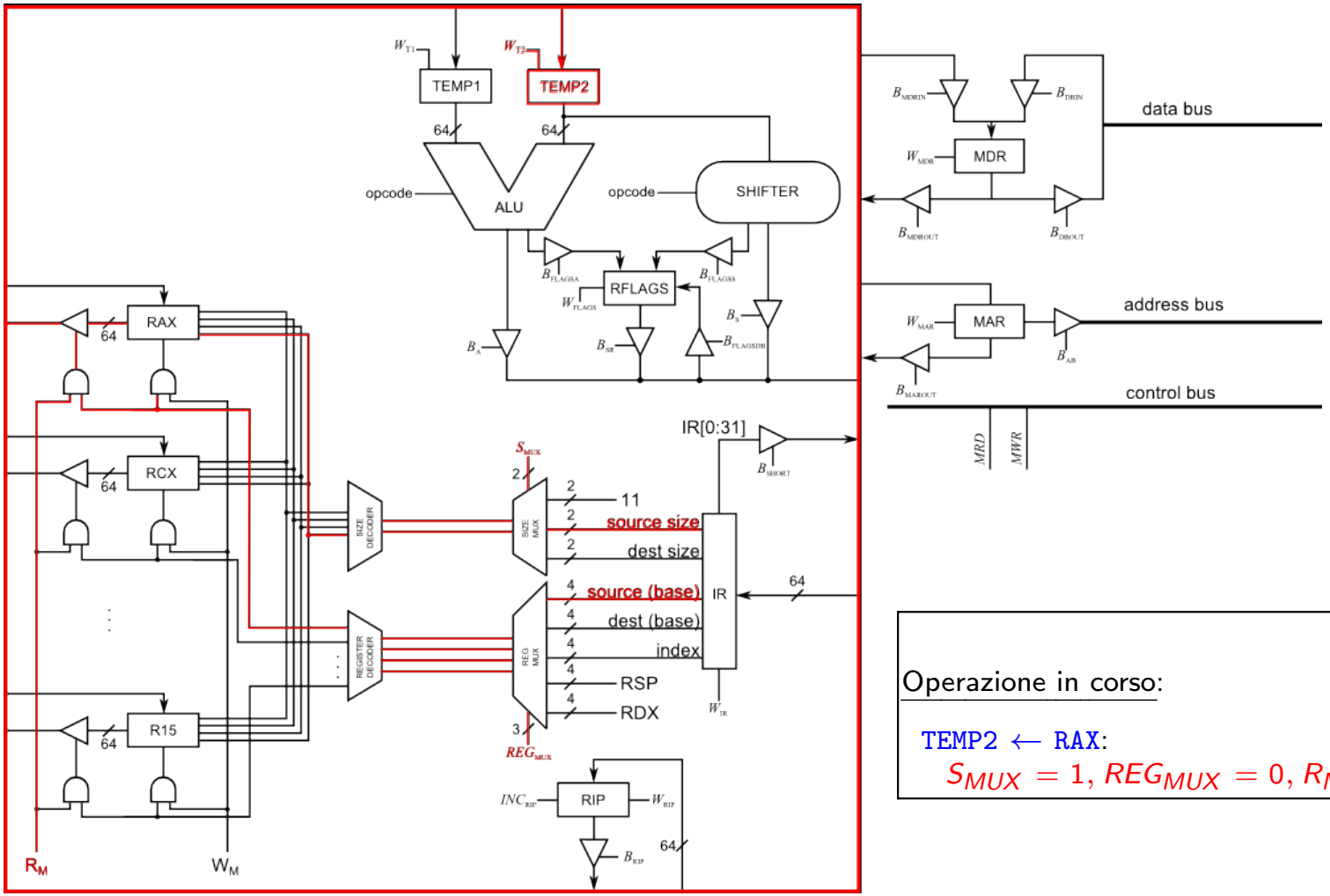


Istruzioni di movimento dati

- Le microoperazioni associate al movimento dati dipendono dalla modalità di indirizzamento utilizzato
- Accedere in memoria utilizzando la modalità di indirizzamento dello z64 è un'attività costosa

- `movq %rax, %rcx`:
  - `MAR` $\leftarrow$ `RIP`
  - `MDR` $\leftarrow$ (`MAR`); `RIP` $\leftarrow$ `RIP + 8`
  - `IR` $\leftarrow$ `MDR`
  - `TEMP2` $\leftarrow$ `RAX`
  - `RCX` $\leftarrow$ `TEMP2`

Istruzioni di movimento dei dati

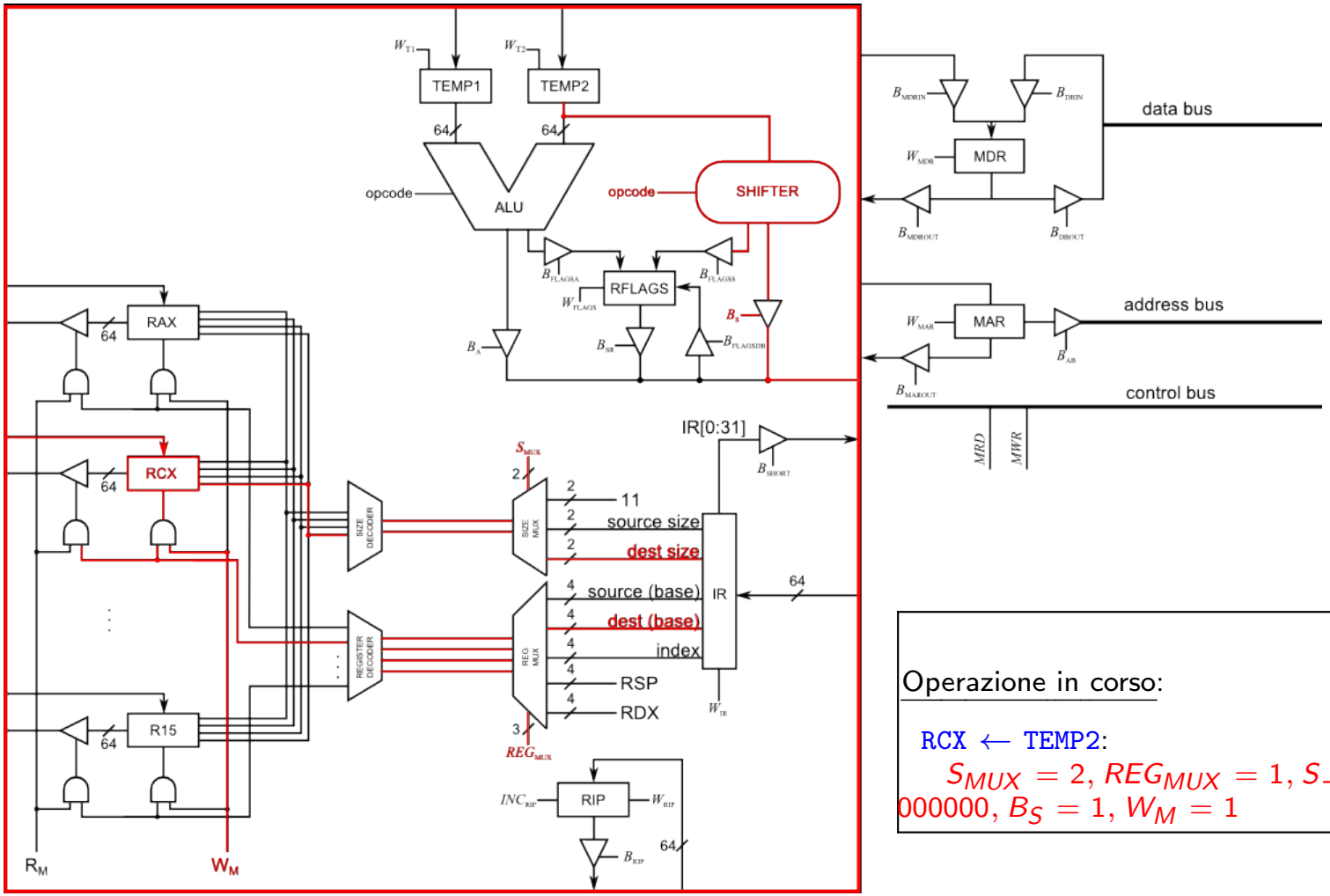


Operazione in corso:

TEMP2 ← RAX:

$S_{MUX} = 1, REG_{MUX} = 0, R_M = 1, W_{T2} = 1$

Istruzioni di movimento dei dati



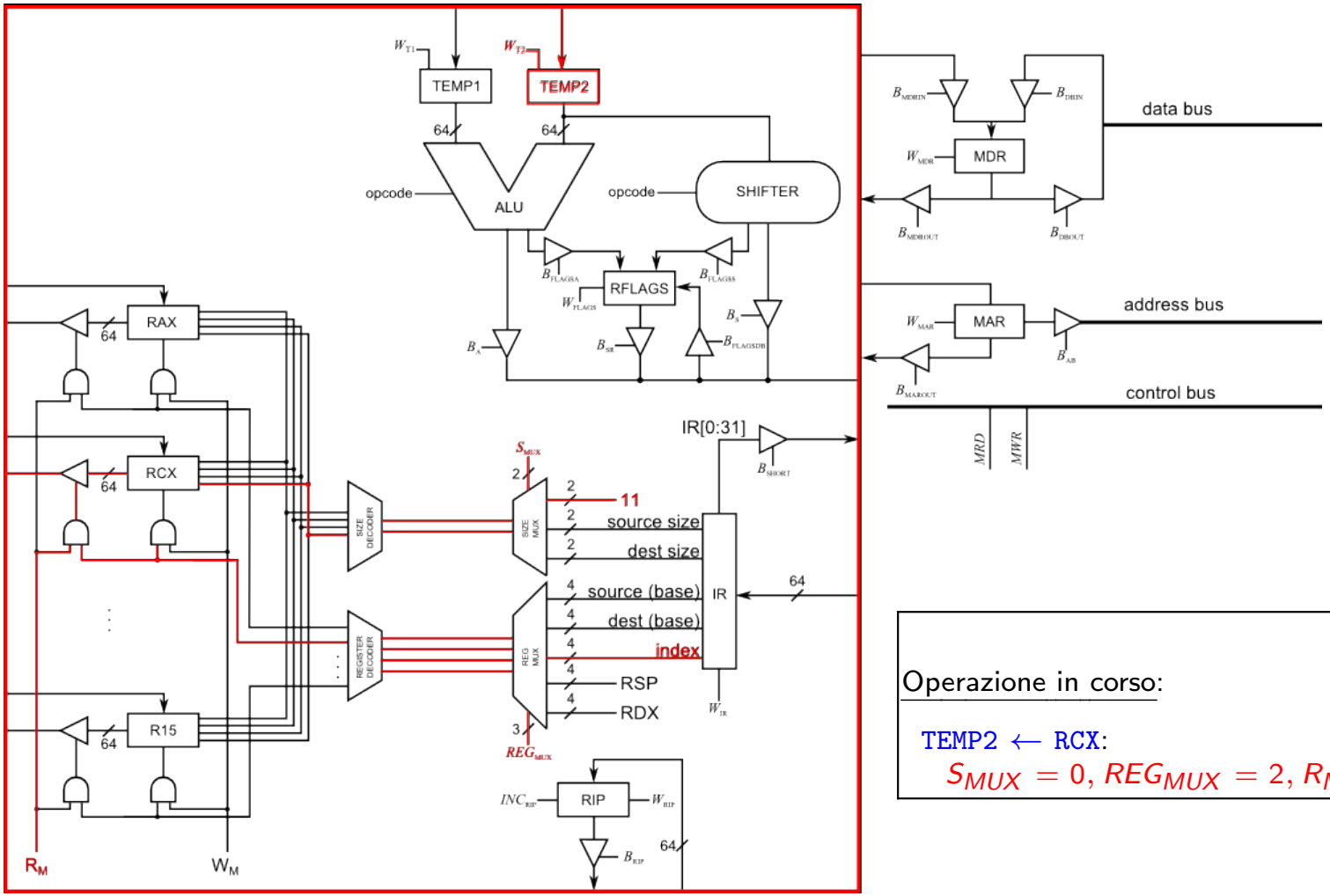
Operazione in corso:

RCX ← TEMP2:
$S_{MUX} = 2, REG_{MUX} = 1, S\_opcode = 000000, B_S = 1, W_M = 1$

Istruzioni di movimento dati: full addressing

- `movq %rax, 0xaaaa(%rax, %rcx, 8)`:
  - MAR $\leftarrow$ RIP
  - MDR $\leftarrow$ (MAR); RIP $\leftarrow$ RIP + 8
  - IR $\leftarrow$ MDR
  - TEMP2 $\leftarrow$ RCX
  - TEMP1 $\leftarrow$ Shifter_Out[SHL, 000011]
  - TEMP2 $\leftarrow$ RAX
  - MAR $\leftarrow$ ALU_OUT[ADD]
  - TEMP1 $\leftarrow$ IR[0:31]
  - TEMP2 $\leftarrow$ MAR
  - MAR $\leftarrow$ ALU_OUT[ADD]
  - MDR $\leftarrow$ RAX
  - (MAR) $\leftarrow$ MDR

Istruzioni di movimento dei dati: full addressing

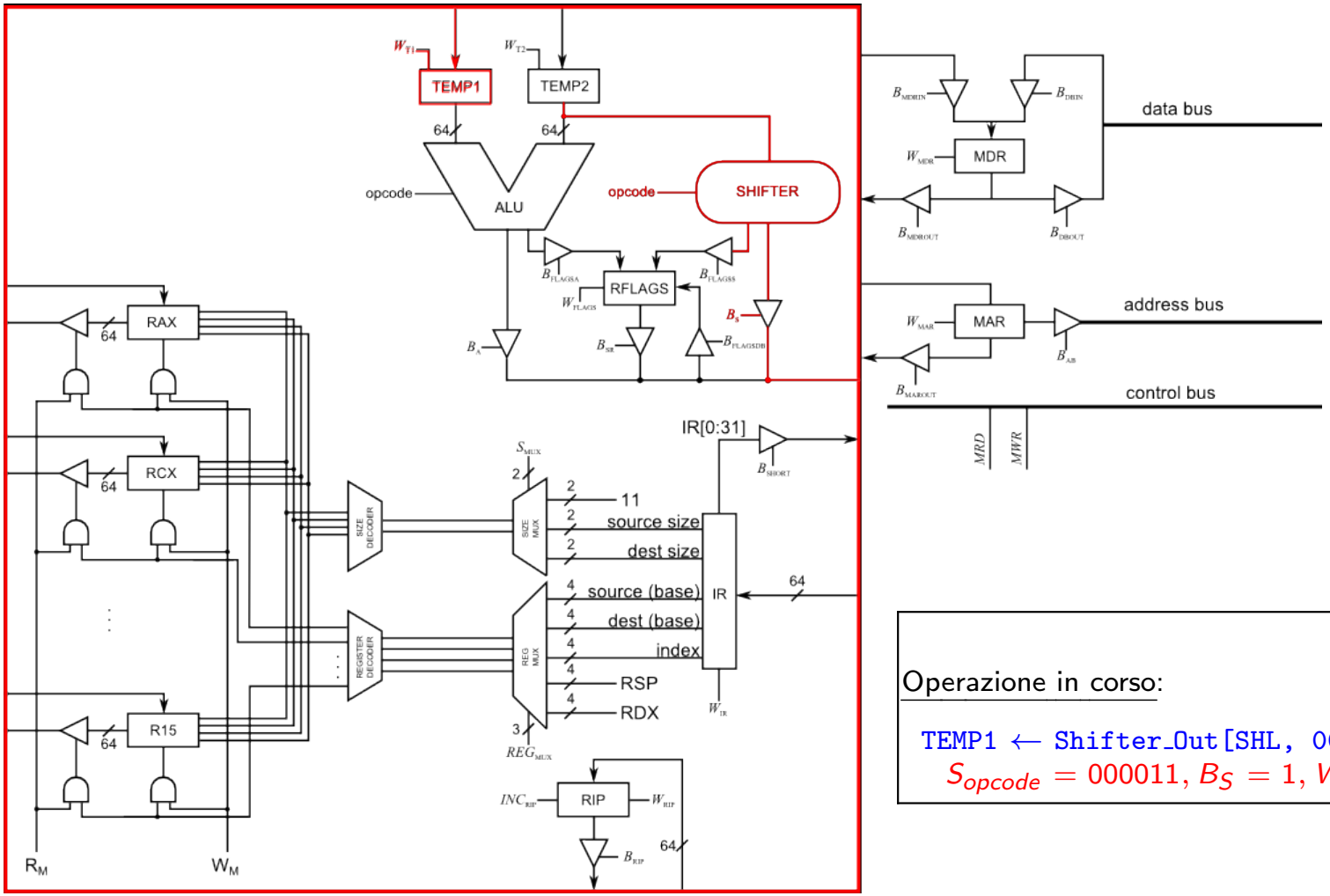


Operazione in corso:

TEMP2 ← RCX:

$S_{MUX} = 0, REG_{MUX} = 2, R_M = 1, W_{T2} = 1$

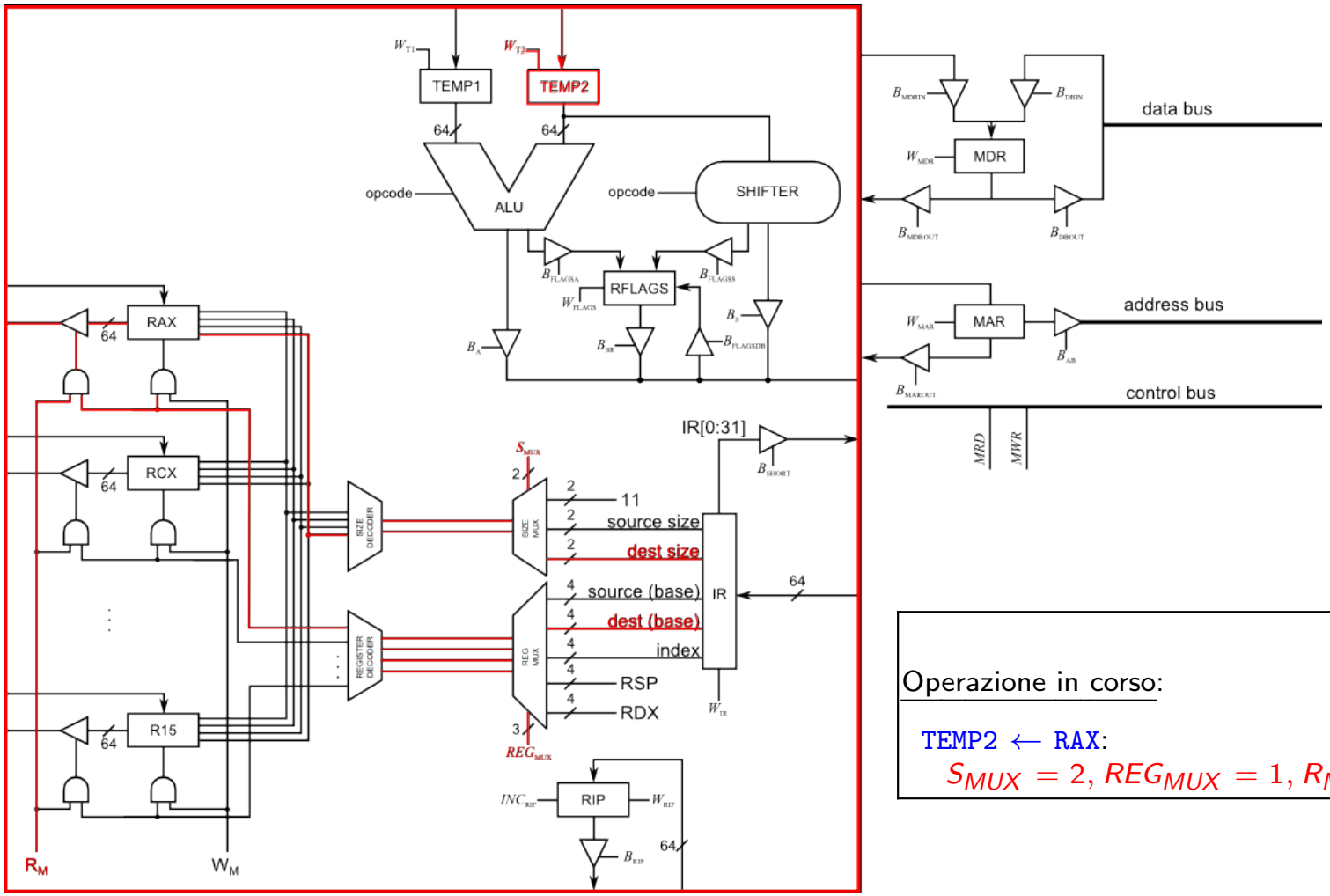
Istruzioni di movimento dei dati: full addressing



Operazione in corso:

TEMP1 ← Shifter_Out[SHL, 000011]:
$S_{opcode} = 000011, B_S = 1, W_{T1} = 1$

Istruzioni di movimento dei dati: full addressing

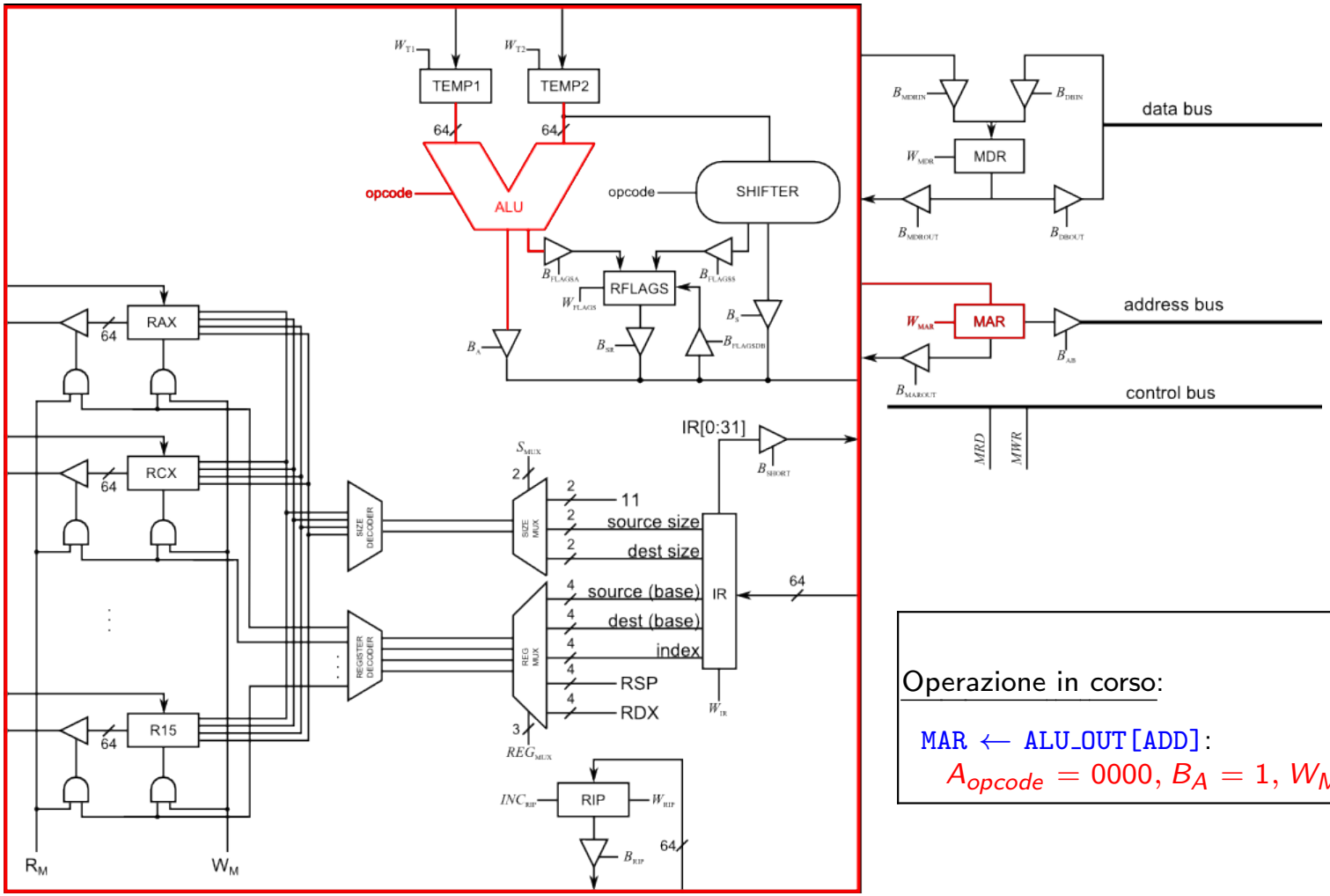


Operazione in corso:

TEMP2 ← RAX:

$S_{MUX} = 2, REG_{MUX} = 1, R_M = 1, W_{T2} = 1$

Istruzioni di movimento dei dati: full addressing

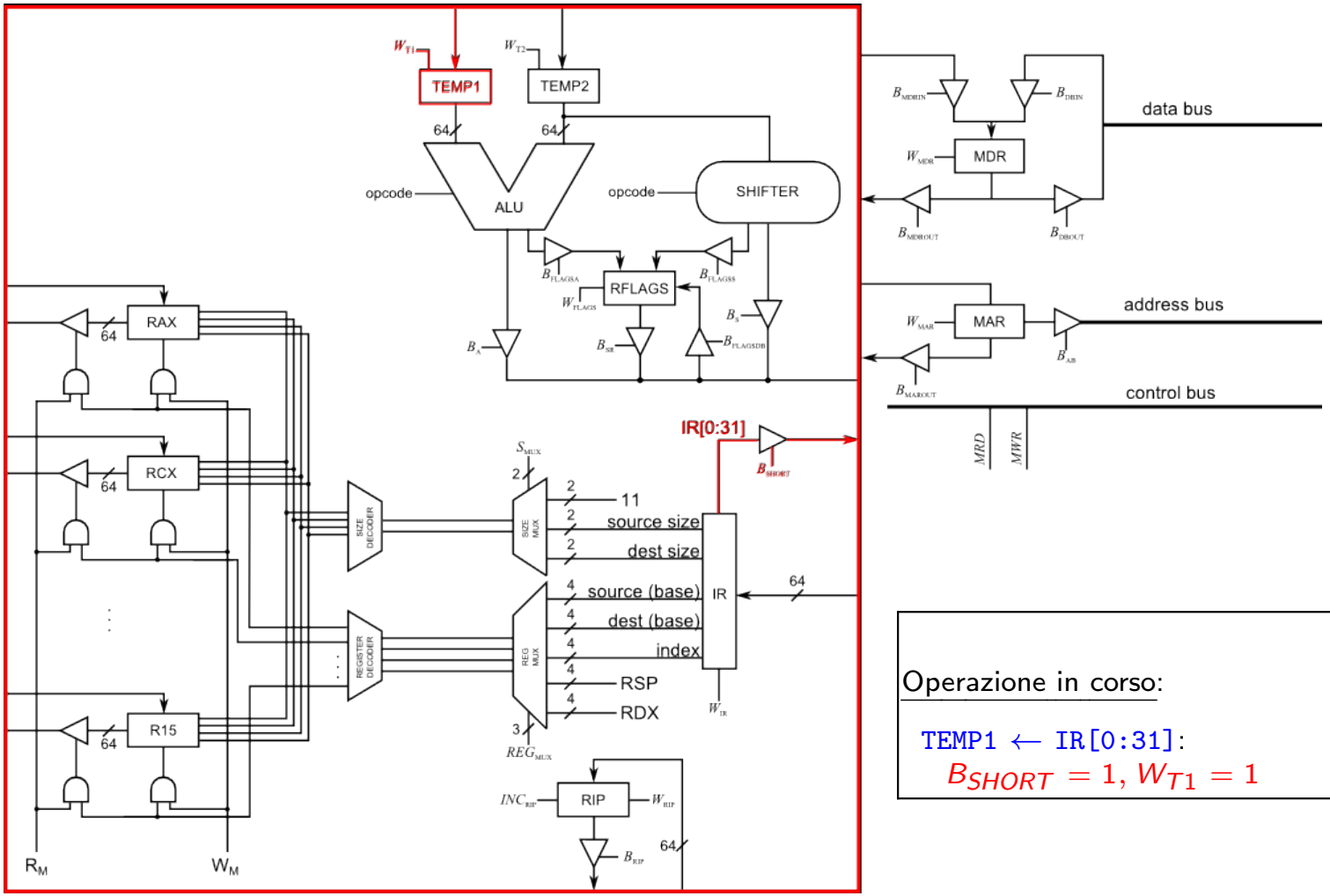


Operazione in corso:

MAR ← ALU_OUT[ADD]:

$A_{opcode} = 0000, B_A = 1, W_{MAR} = 1$

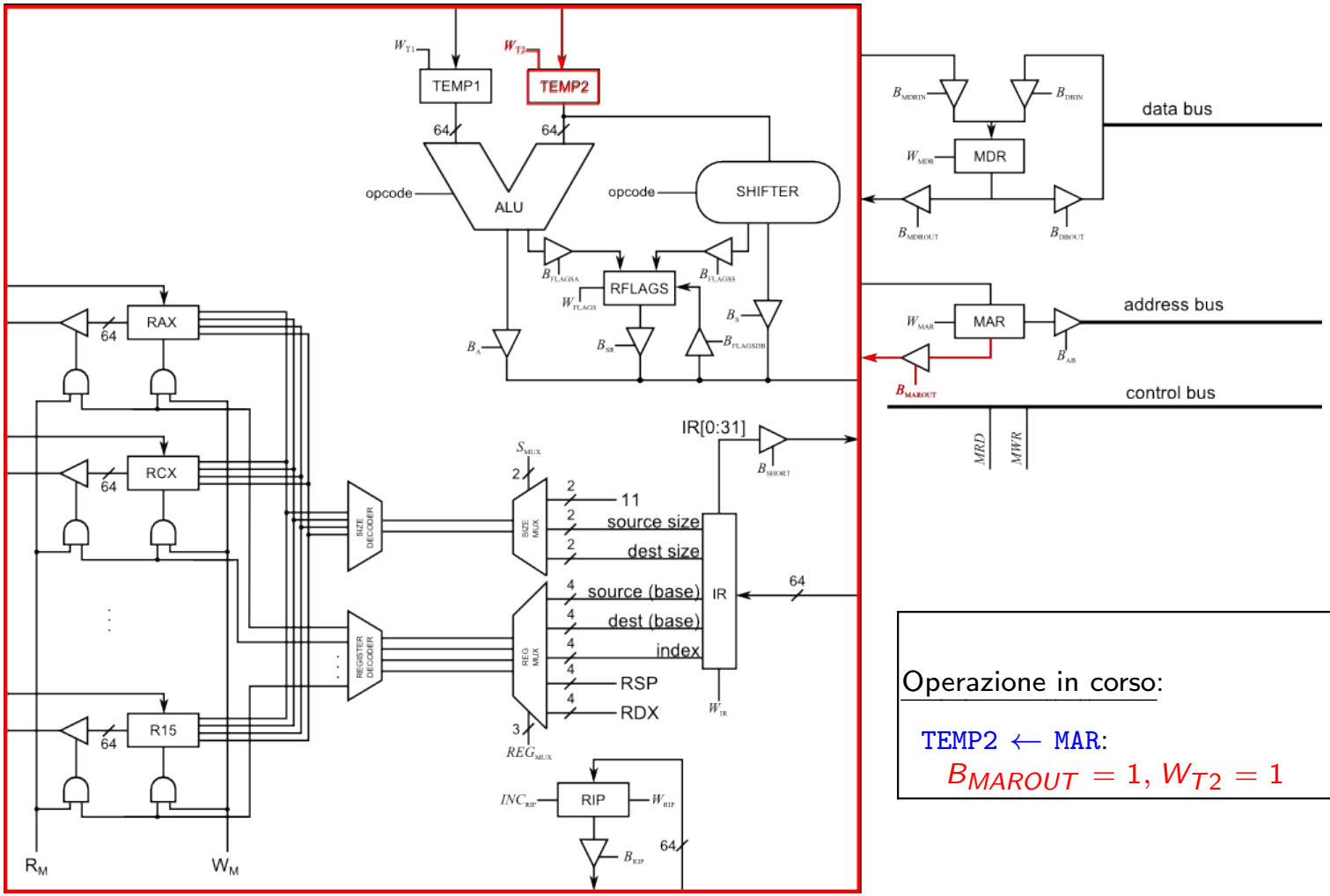
Istruzioni di movimento dei dati: full addressing



Operazione in corso:

TEMP1 ← IR[0:31]:
$B_{SHORT} = 1, W_{T1} = 1$

Istruzioni di movimento dei dati: full addressing

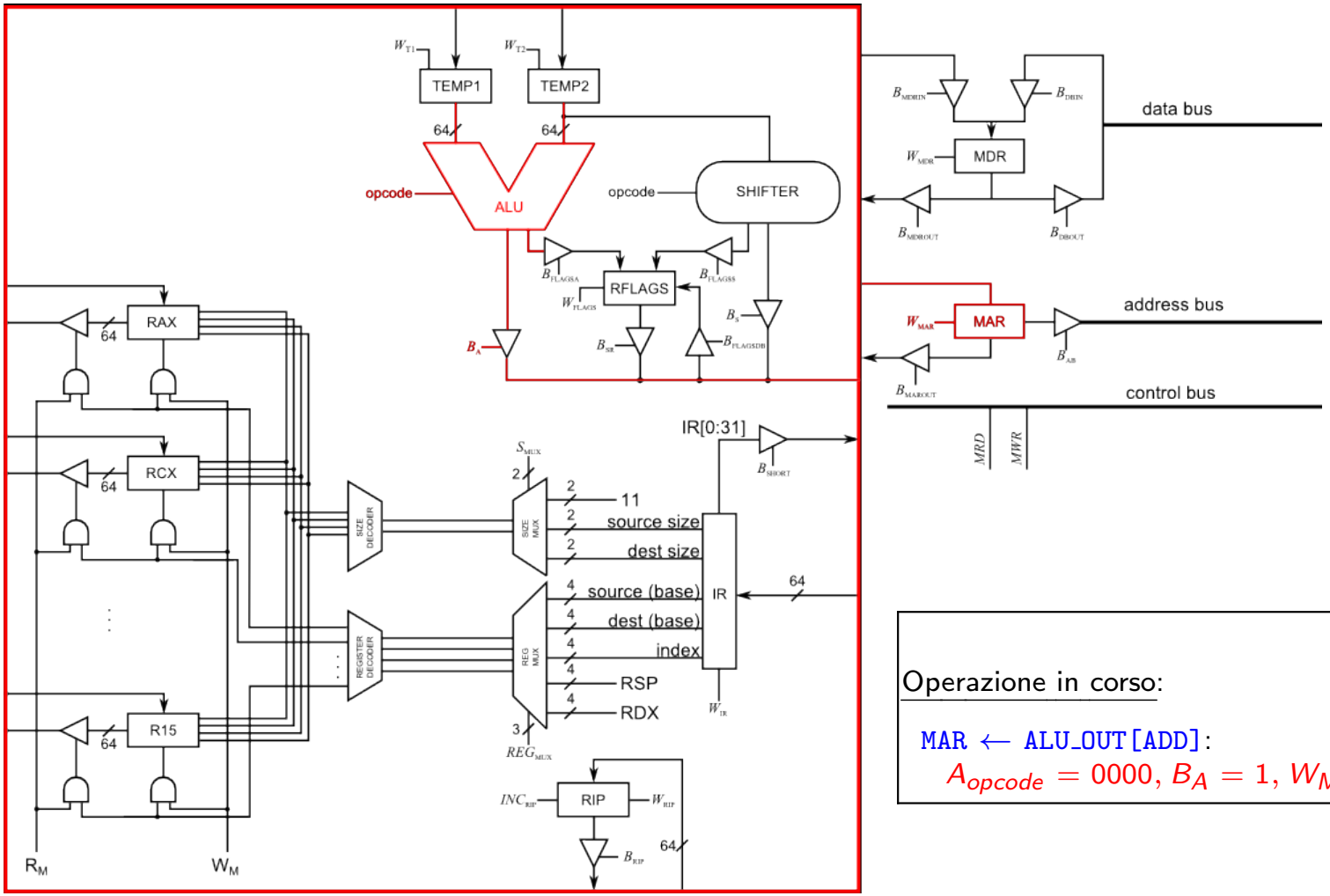


Operazione in corso:

TEMP2 ← MAR:

$B_{MAROUT} = 1, W_{T2} = 1$

Istruzioni di movimento dei dati: full addressing

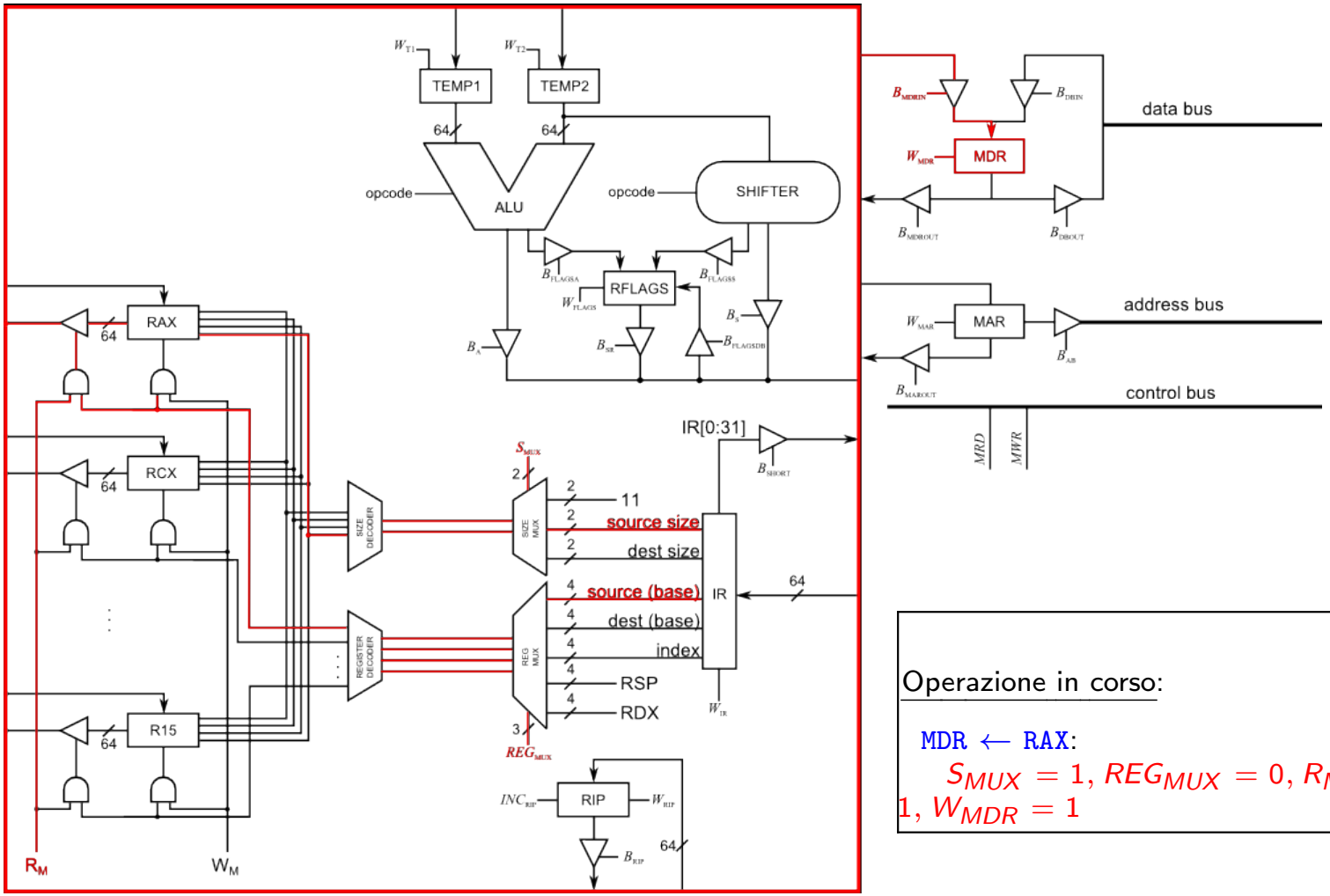


Operazione in corso:

MAR ← ALU_OUT[ADD]:

$A_{opcode} = 0000, B_A = 1, W_{MAR} = 1$

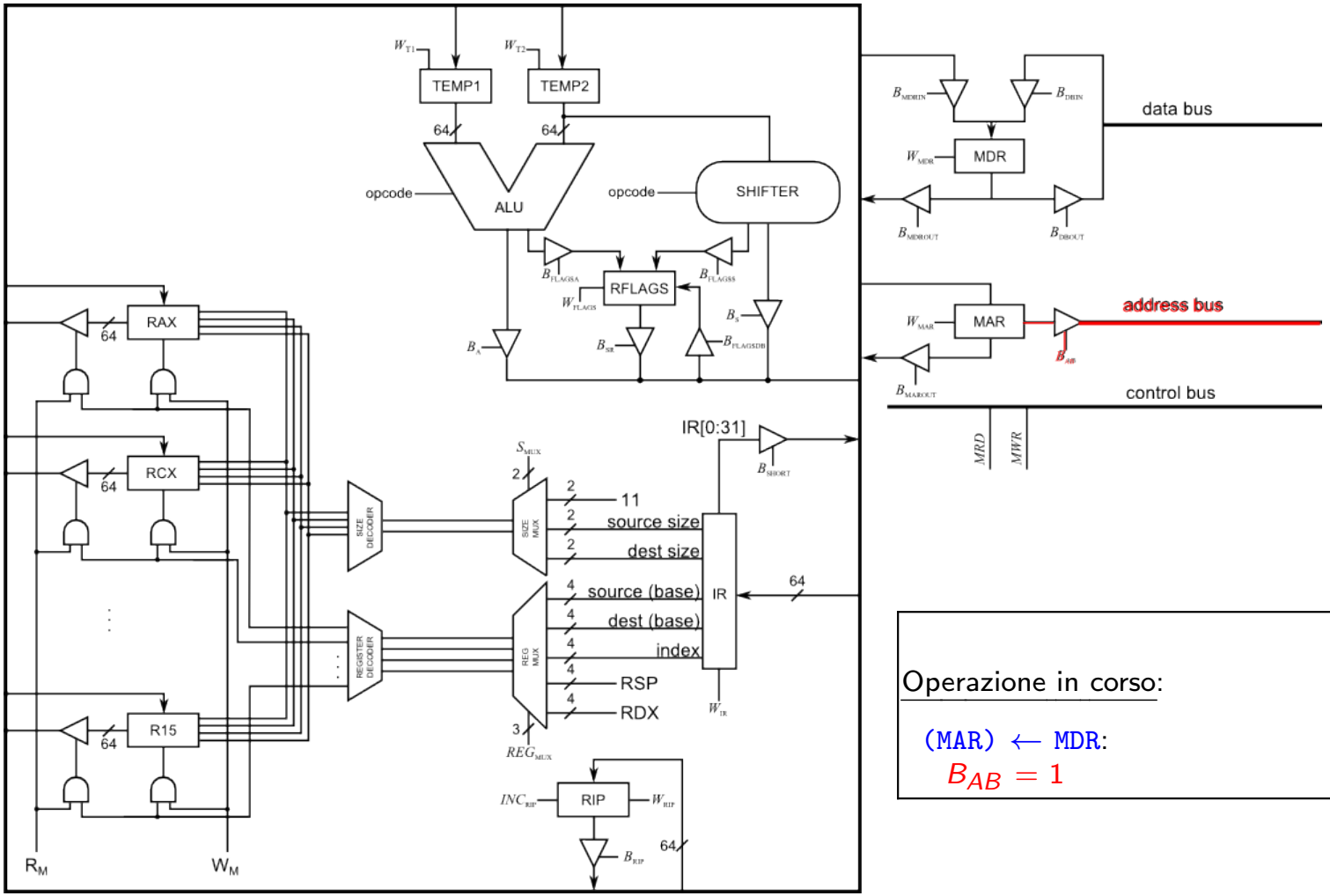
Istruzioni di movimento dei dati: full addressing



Operazione in corso:

MDR ← RAX:
$S_{MUX} = 1, REG_{MUX} = 0, R_M = 1, B_{MDRIN} = 1, W_{MDR} = 1$

Istruzioni di movimento dei dati: full addressing

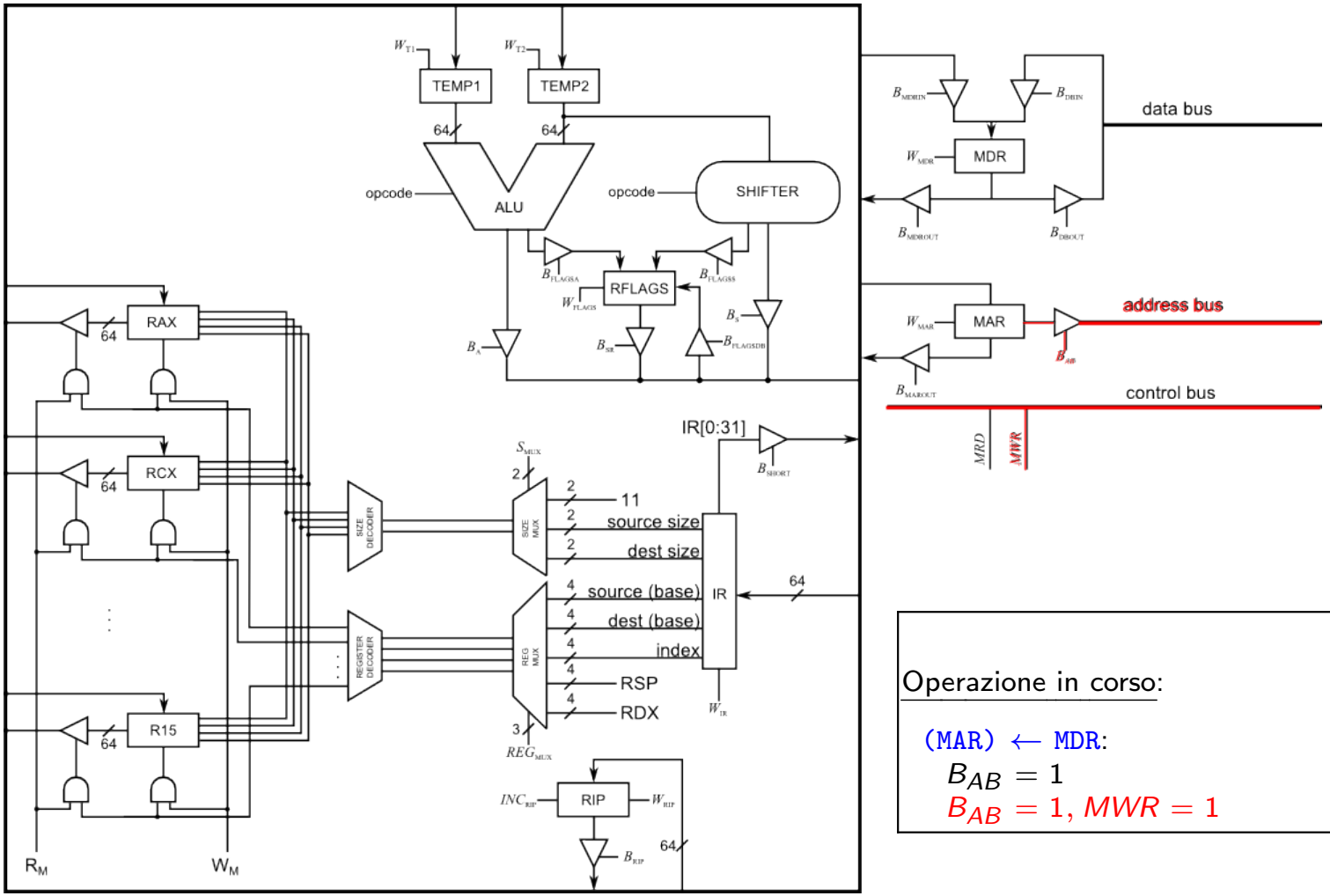


Operazione in corso:

(MAR) ← MDR:

$B_{AB} = 1$

Istruzioni di movimento dei dati: full addressing



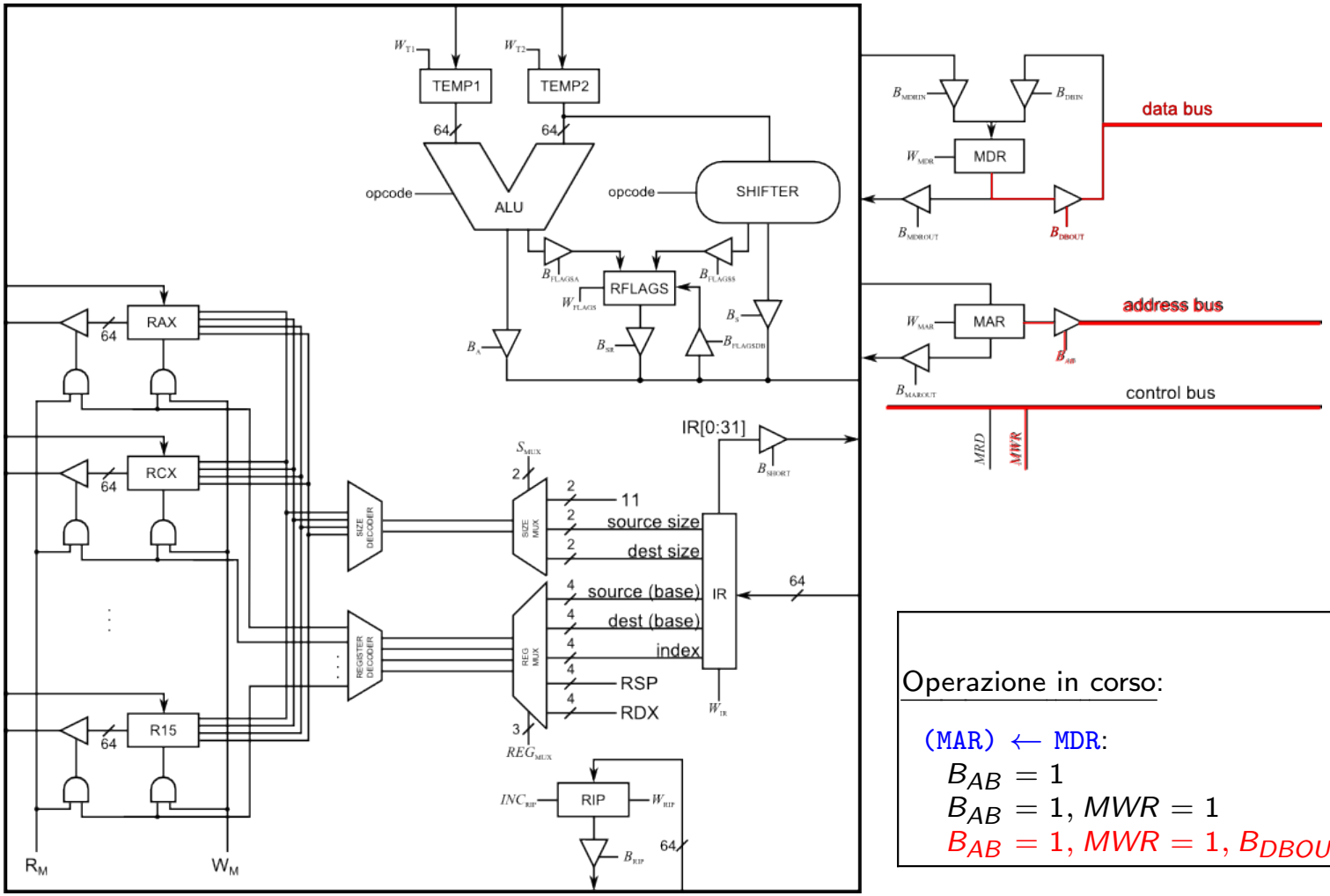
Operazione in corso:

(MAR) $\leftarrow$ MDR:

$B_{AB} = 1$

$B_{AB} = 1, MWR = 1$

Istruzioni di movimento dei dati: full addressing



Operazione in corso:

(MAR) ← MDR:

$B_{AB} = 1$

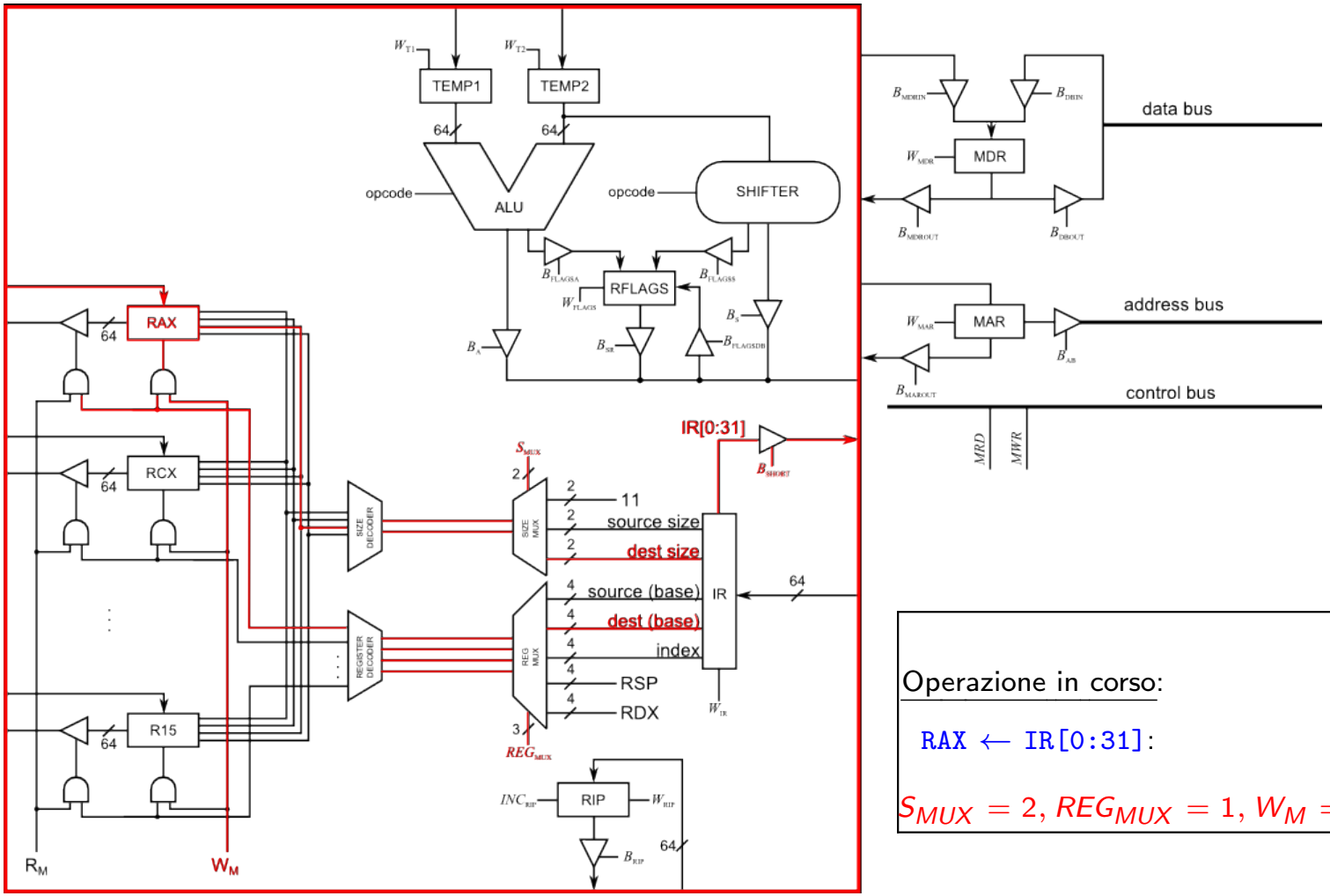
$B_{AB} = 1, MWR = 1$

$B_{AB} = 1, MWR = 1, B_{DBOUT} = 1$

Istruzioni di movimento dati: immediati “piccoli”

- `movl $0xaaaa, %eax`:
  - MAR $\leftarrow$ RIP
  - MDR $\leftarrow$ (MAR); RIP $\leftarrow$ RIP + 8
  - IR $\leftarrow$ MDR
  - EAX $\leftarrow$ IR[0:31]

Istruzioni di movimento dei dati: immediati "piccoli"



Operazione in corso:

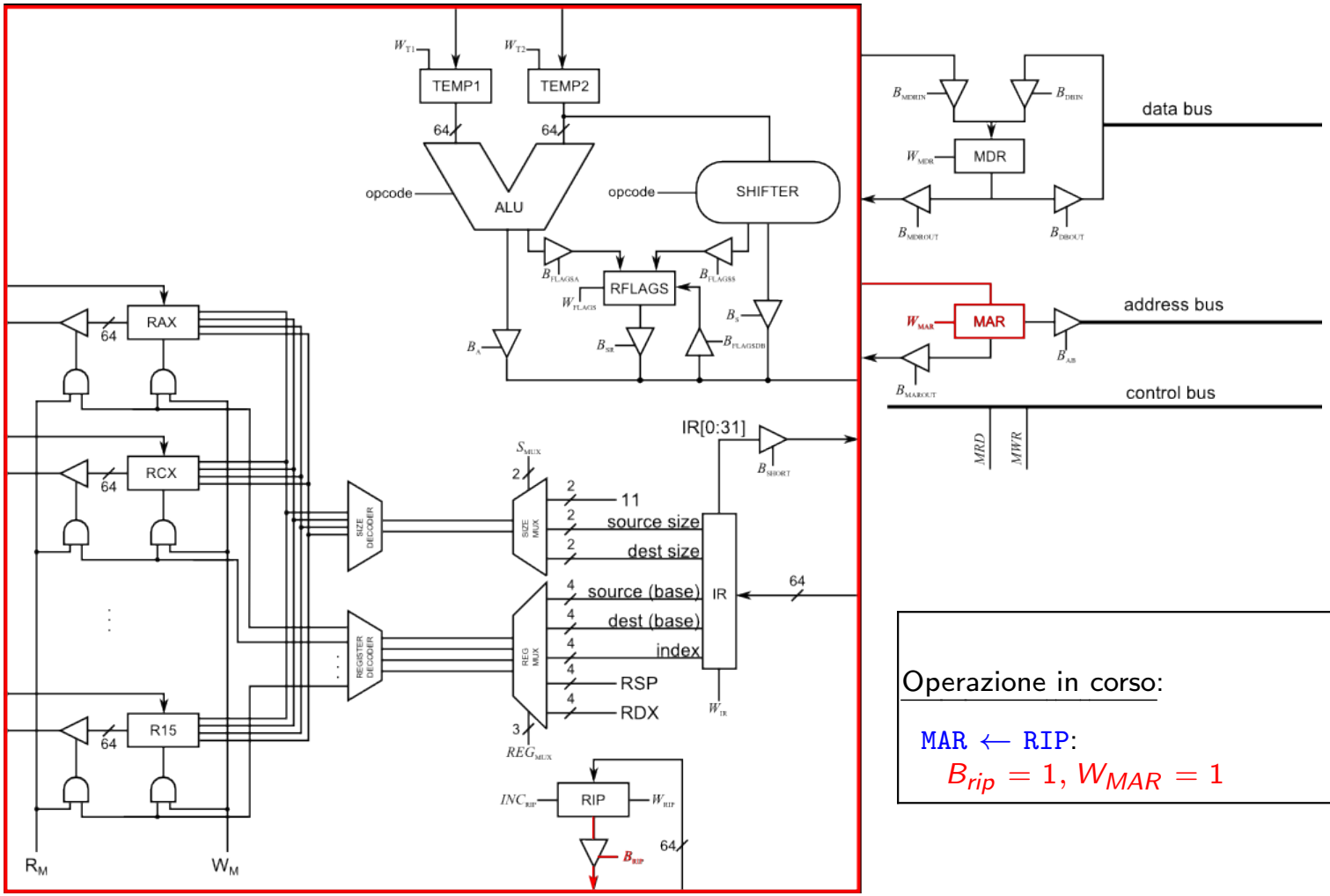
RAX ← IR[0:31]:

$S_{MUX} = 2, REG_{MUX} = 1, W_M = 1, B_{SHORT} = 1$

Istruzioni di movimento dati: immediati “grandi”

- `movq $0xaaaaa, %rax`:
  - MAR $\leftarrow$ RIP
  - MDR $\leftarrow$ (MAR); RIP $\leftarrow$ RIP + 8
  - IR $\leftarrow$ MDR
  - MAR $\leftarrow$ RIP
  - MDR $\leftarrow$ (MAR); RIP $\leftarrow$ RIP + 8
  - RAX $\leftarrow$ MDR

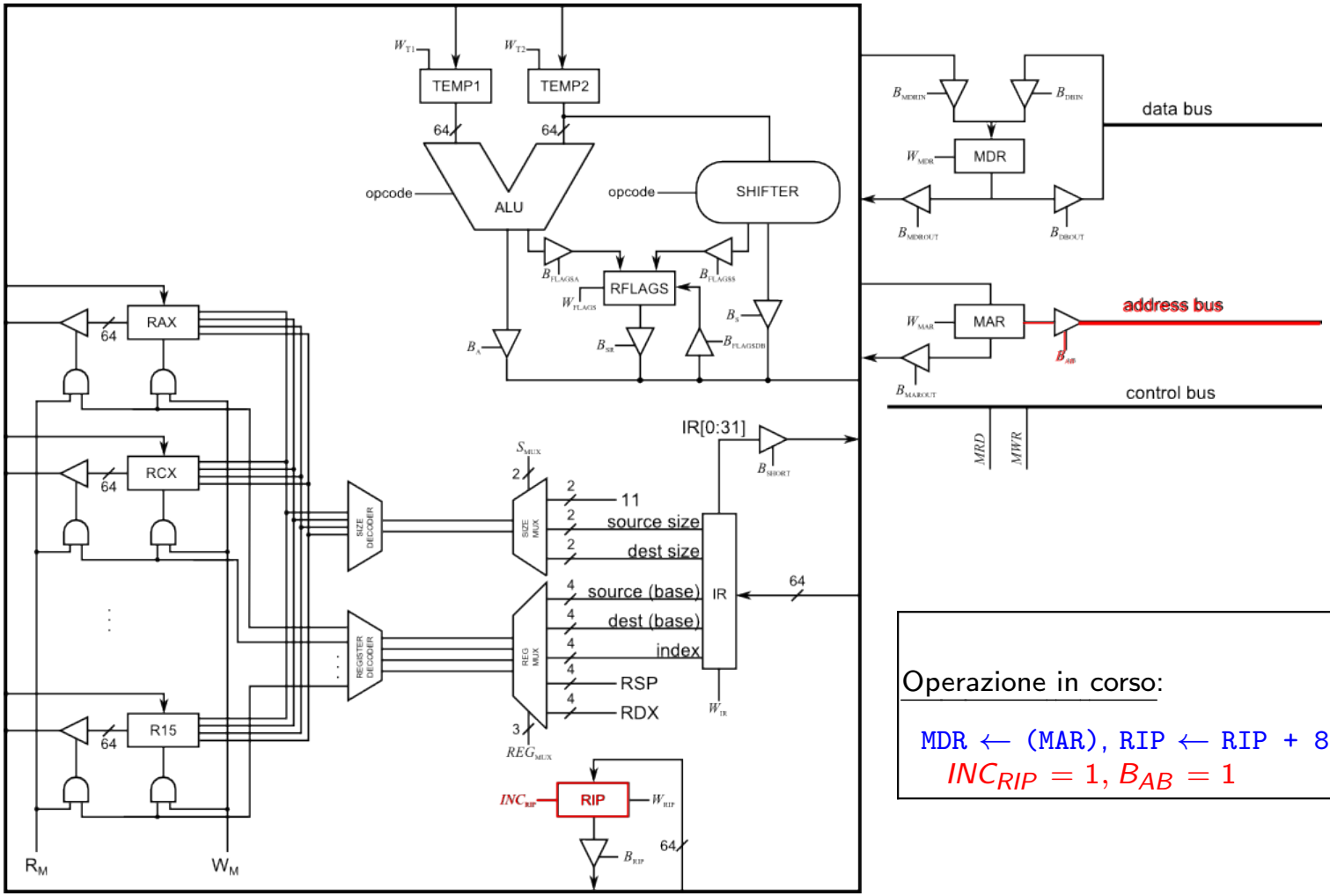
Istruzioni di movimento dati: immediati “grandi”



Operazione in corso:

MAR ← RIP:
$B_{rip} = 1, W_{MAR} = 1$

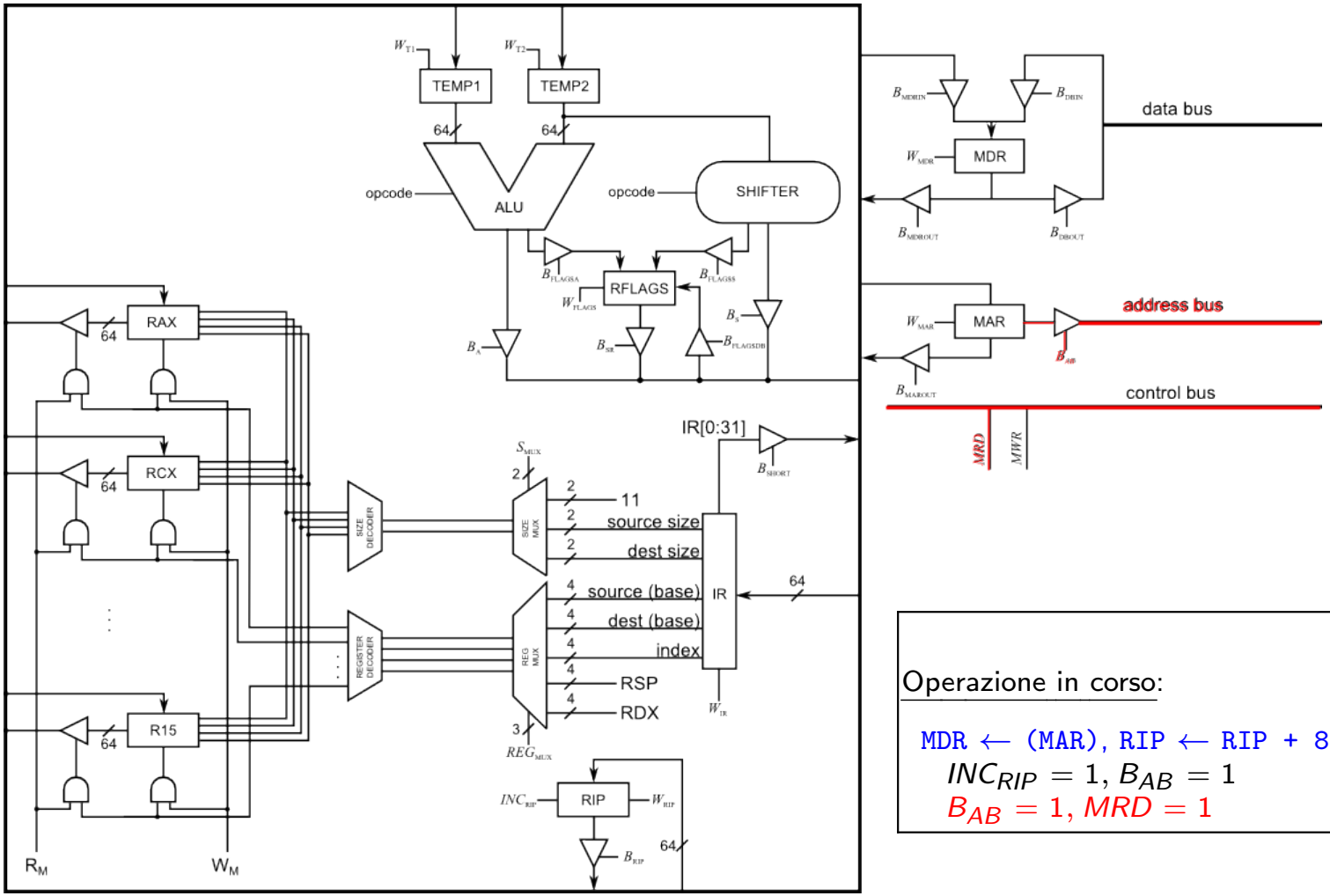
Istruzioni di movimento dati: immediati "grandi"



Operazione in corso:

`MDR ← (MAR), RIP ← RIP + 8`:
$INC_{RIP} = 1, B_{AB} = 1$

Istruzioni di movimento dati: immediati "grandi"



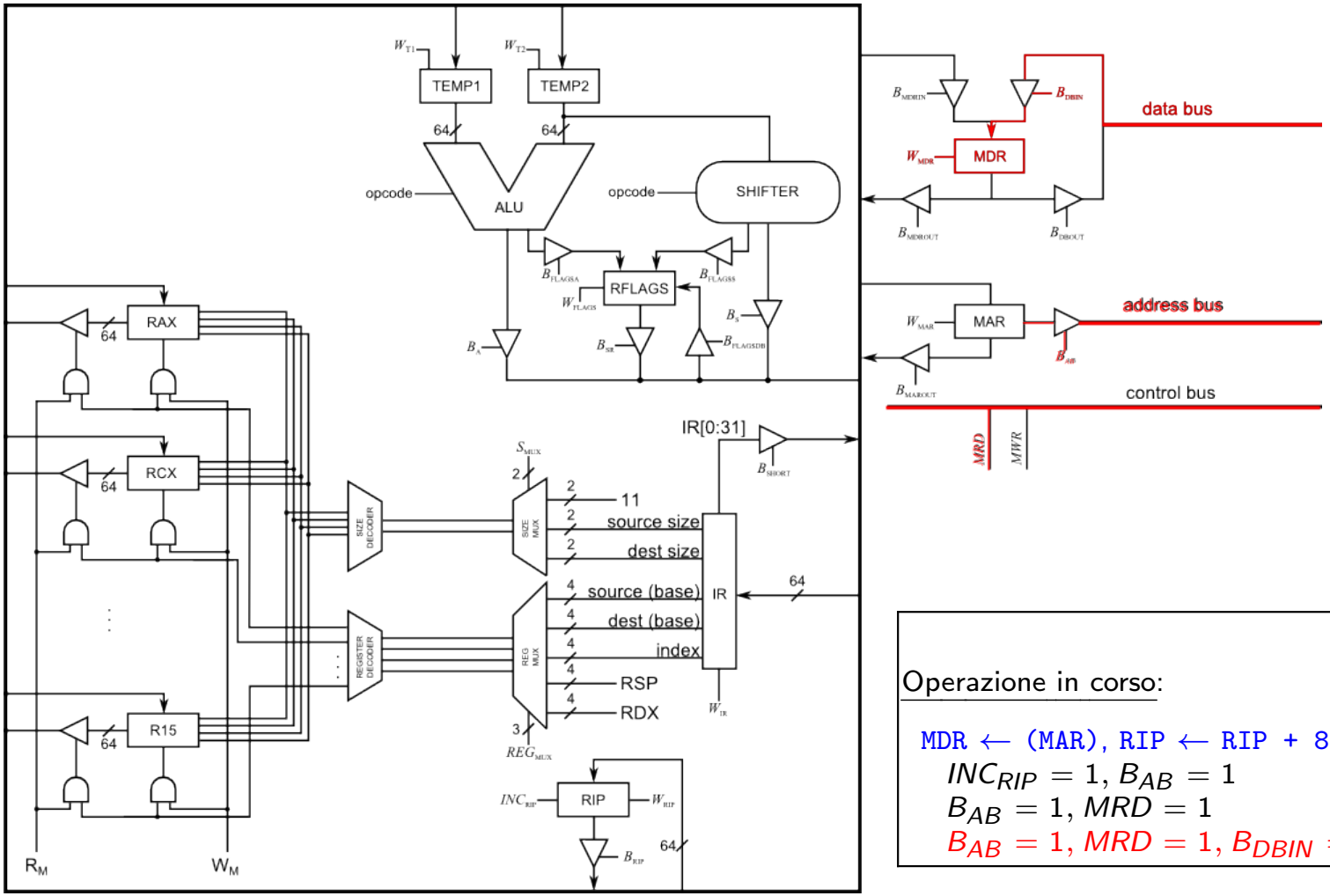
Operazione in corso:

MDR ← (MAR), RIP ← RIP + 8:

$INC_{RIP} = 1, B_{AB} = 1$

$B_{AB} = 1, MRD = 1$

Istruzioni di movimento dati: immediati "grandi"



Operazione in corso:

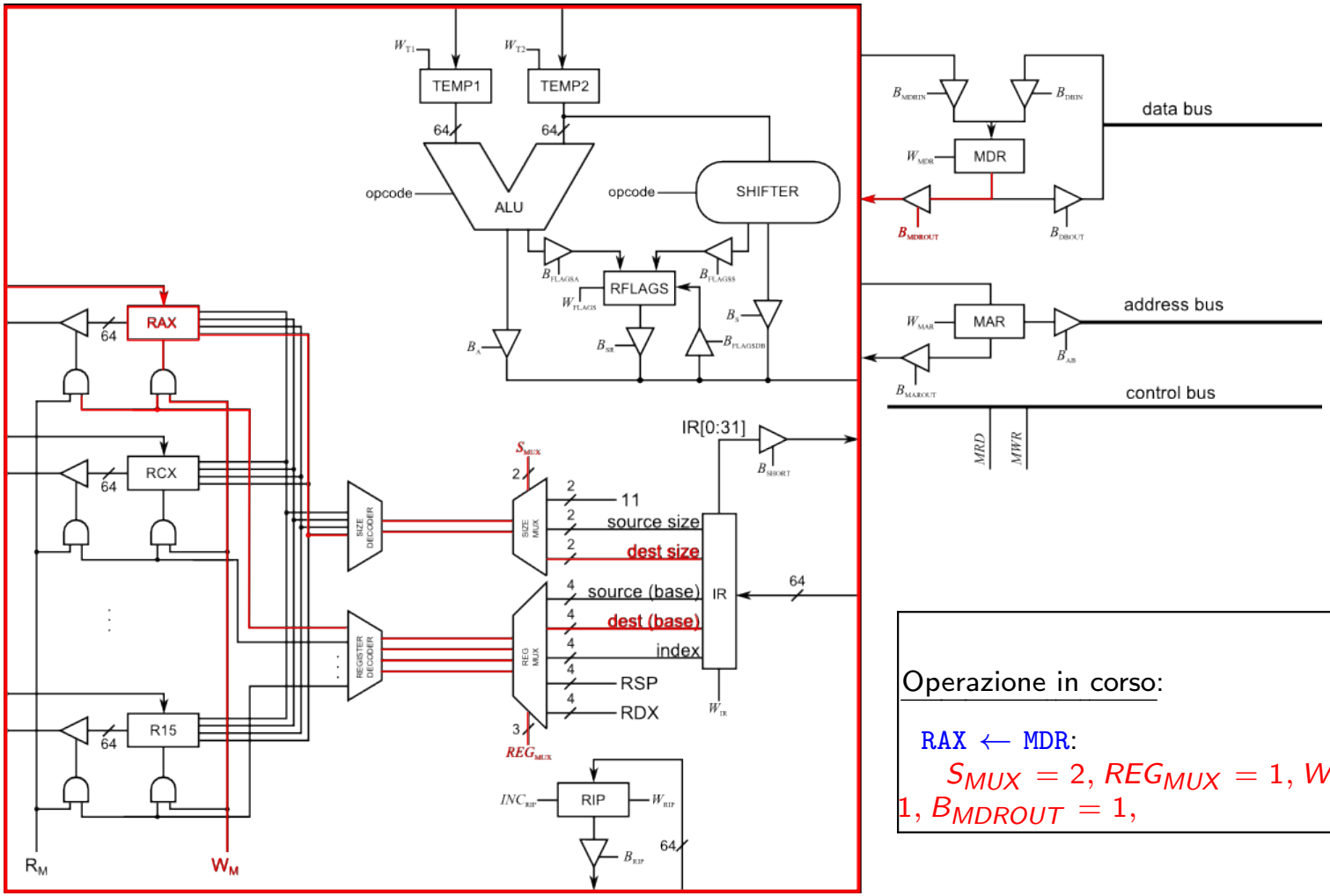
MDR ← (MAR), RIP ← RIP + 8:

$INC_{RIP} = 1, B_{AB} = 1$

$B_{AB} = 1, MRD = 1$

$B_{AB} = 1, MRD = 1, B_{DBIN} = 1, W_{MDR} = 1$

Istruzioni di movimento dati: immediati "grandi"



Operazione in corso:

RAX ← MDR:
$S_{MUX} = 2, REG_{MUX} = 1, W_M = 1, B_{MDROUT} = 1,$

“Unifichiamo” le modalità di indirizzamento

- In funzione degli operandi, un’istruzione di movimento dati deve o meno accedere in memoria
- L’accesso in memoria è parametrico
- ...ma avere tante versioni diverse di microprogramma per tutte le istruzioni che accedono in memoria non è efficiente!

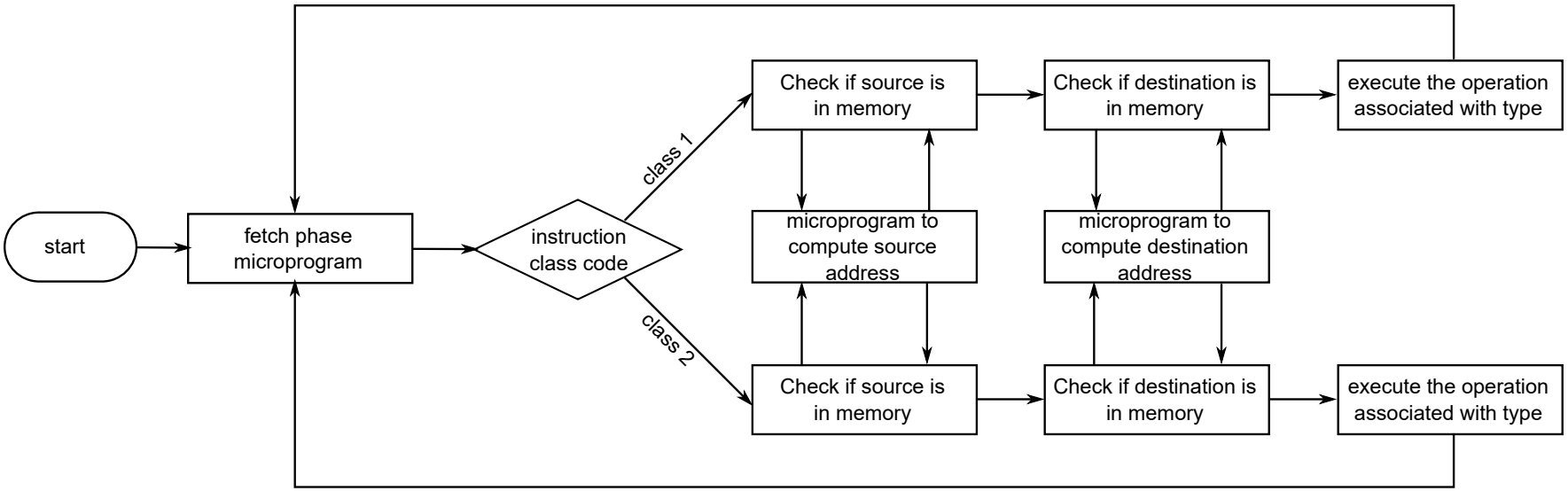
“Unifichiamo” le modalità di indirizzamento

- In funzione degli operandi, un’istruzione di movimento dati deve o meno accedere in memoria
- L’accesso in memoria è parametrico
- ...ma avere tante versioni diverse di microprogramma per tutte le istruzioni che accedono in memoria non è efficiente!
- Una modifica all’organizzazione della CU permette di “chiamare” sottoprgrammi cablati nel microcodice
- Tutte le istruzioni che devono calcolare un indirizzo di memoria possono chiamare quel sottoprogramma

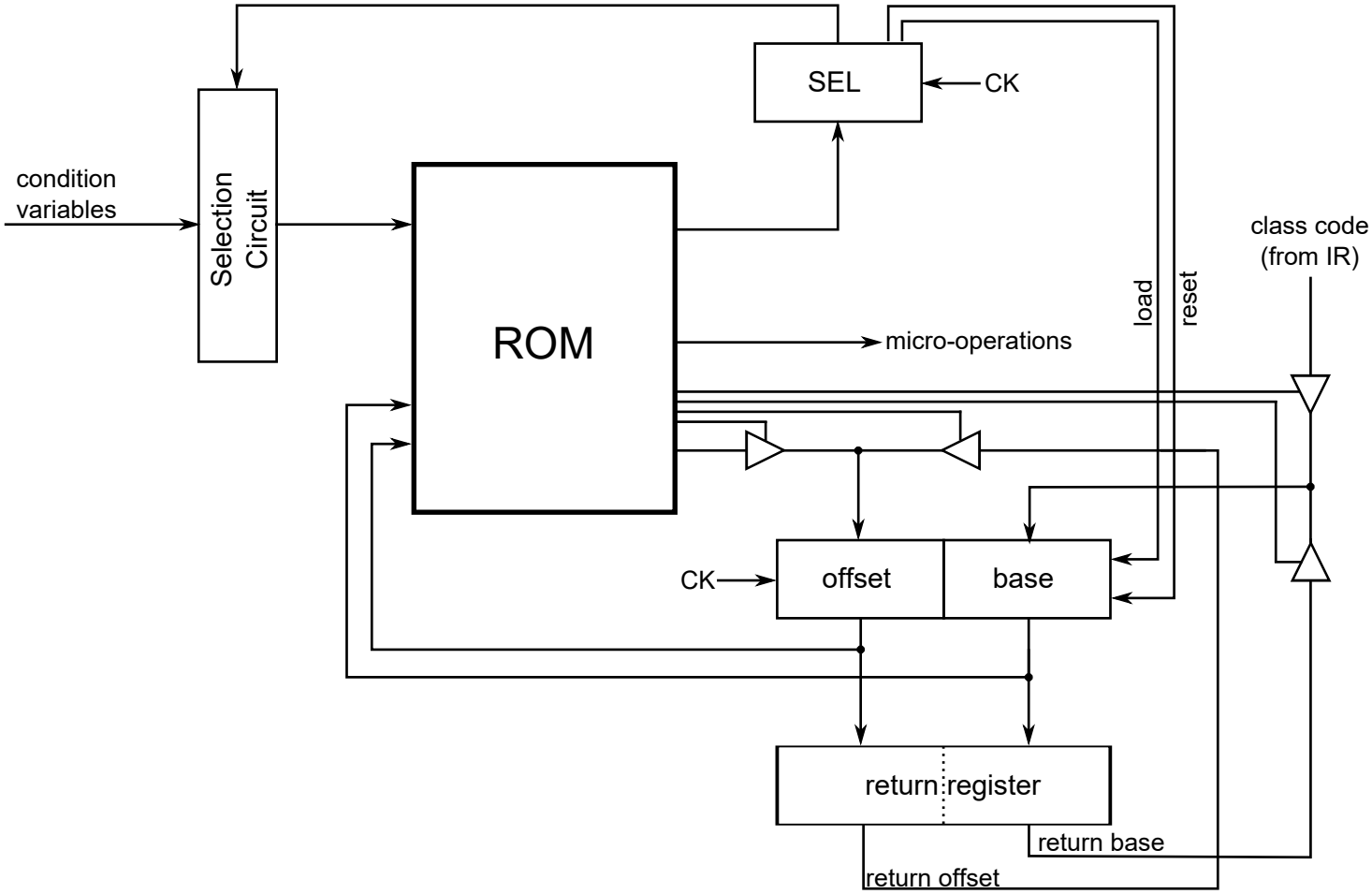
Sottoprogramma per calcolare gli indirizzi

```
1  if D == 1 and B_p == 0 and I_p == 0
2      MAR ← IR[0:31]
3  else if D == 0 and B_p == 1 and I_p == 0
4      MAR ← B
5  else if I_p == 1
6      TEMP2 ← I
7      MAR ← SHIFTER_OUT[SHL, T]
8      TEMP1 ← MAR
9      if D == 1
10         TEMP2 ← IR[0:31]
11         MAR ← ALU_OUT[ADD]
12         TEMP1 ← MAR
13     endif
14     if B_p == 1
15         TEMP2 ← B
16         MAR ← ALU_OUT[ADD]
17     endif
18 endif
```

Chiamata a sottoprogramma



Schema della CU modificata

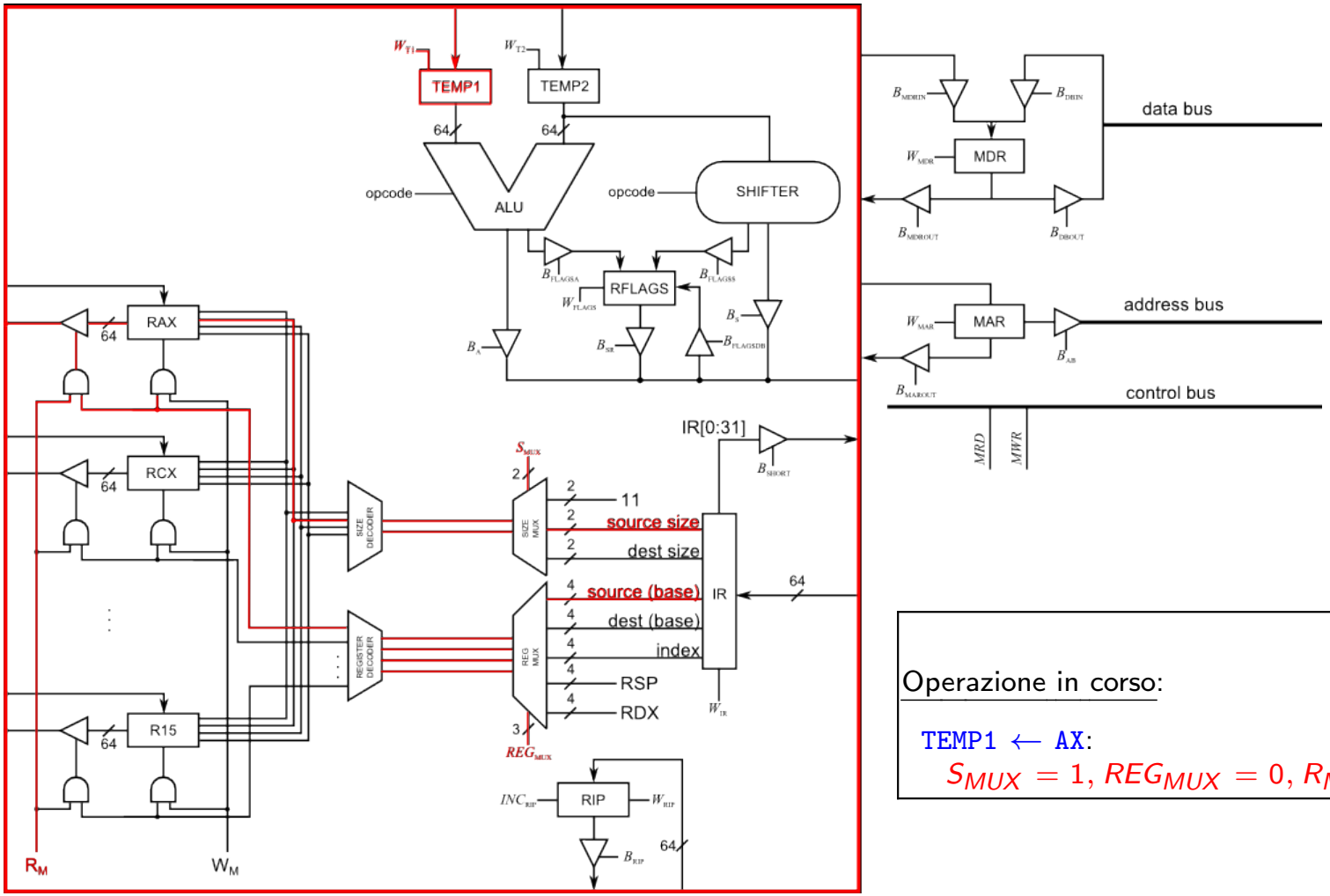


Istruzioni Aritmetiche e Logiche

- L'esecuzione di una determinata operazione aritmetica o logica dipende dall'opcode passato alla ALU

- `addw %ax, %cx`:
  - `MAR` $\leftarrow$ `RIP`
  - `MDR` $\leftarrow$ (`MAR`); `RIP` $\leftarrow$ `RIP + 8`
  - `IR` $\leftarrow$ `MDR`
  - `TEMP1` $\leftarrow$ `AX`
  - `TEMP2` $\leftarrow$ `CX`
  - `CX` $\leftarrow$ `ALU_OUT[ADD]`

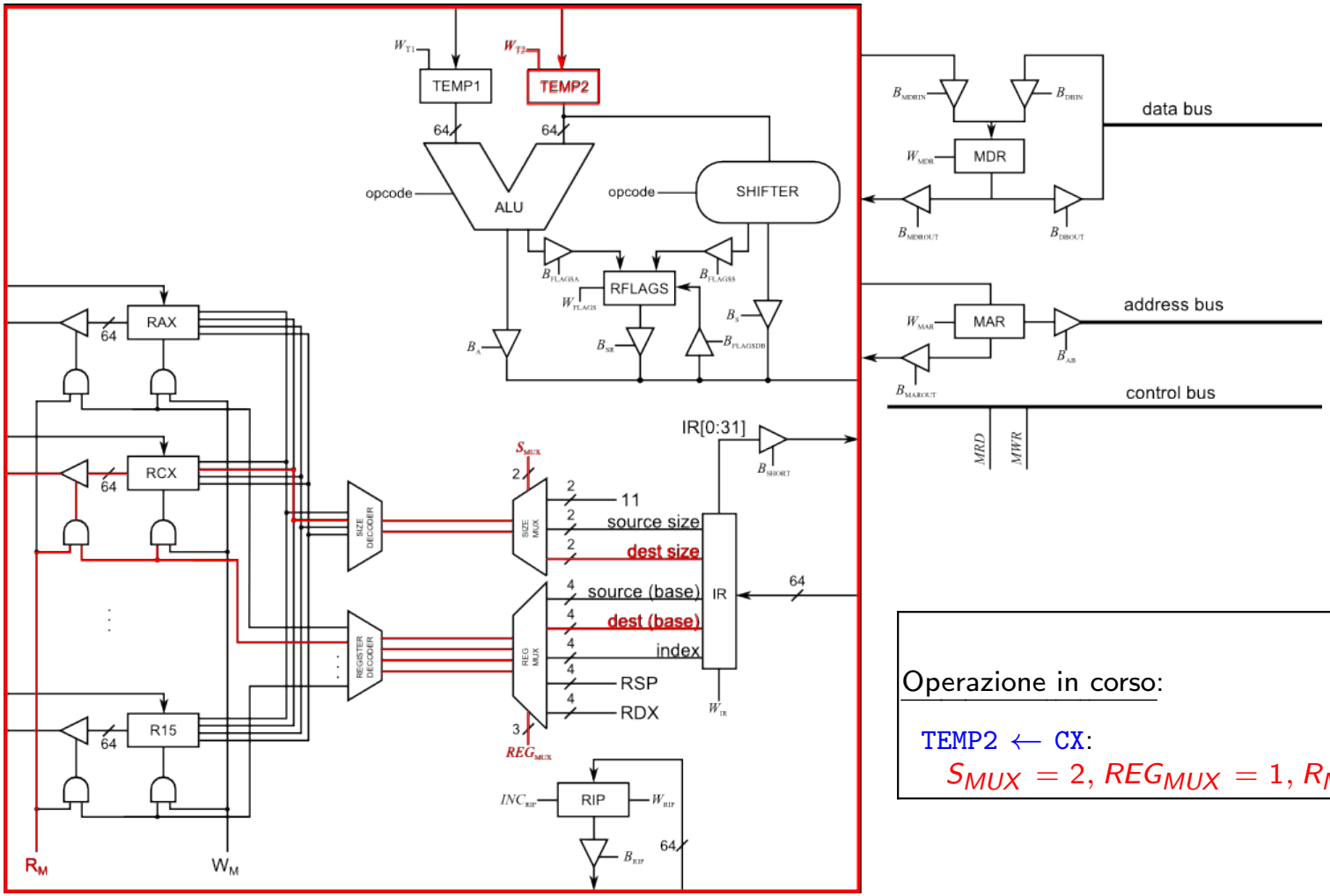
Istruzioni Aritmetiche e Logiche



Operazione in corso:

TEMP1 ← AX:
$S_{MUX} = 1, REG_{MUX} = 0, R_M = 1, W_{T1} = 1$

Istruzioni Aritmetiche e Logiche

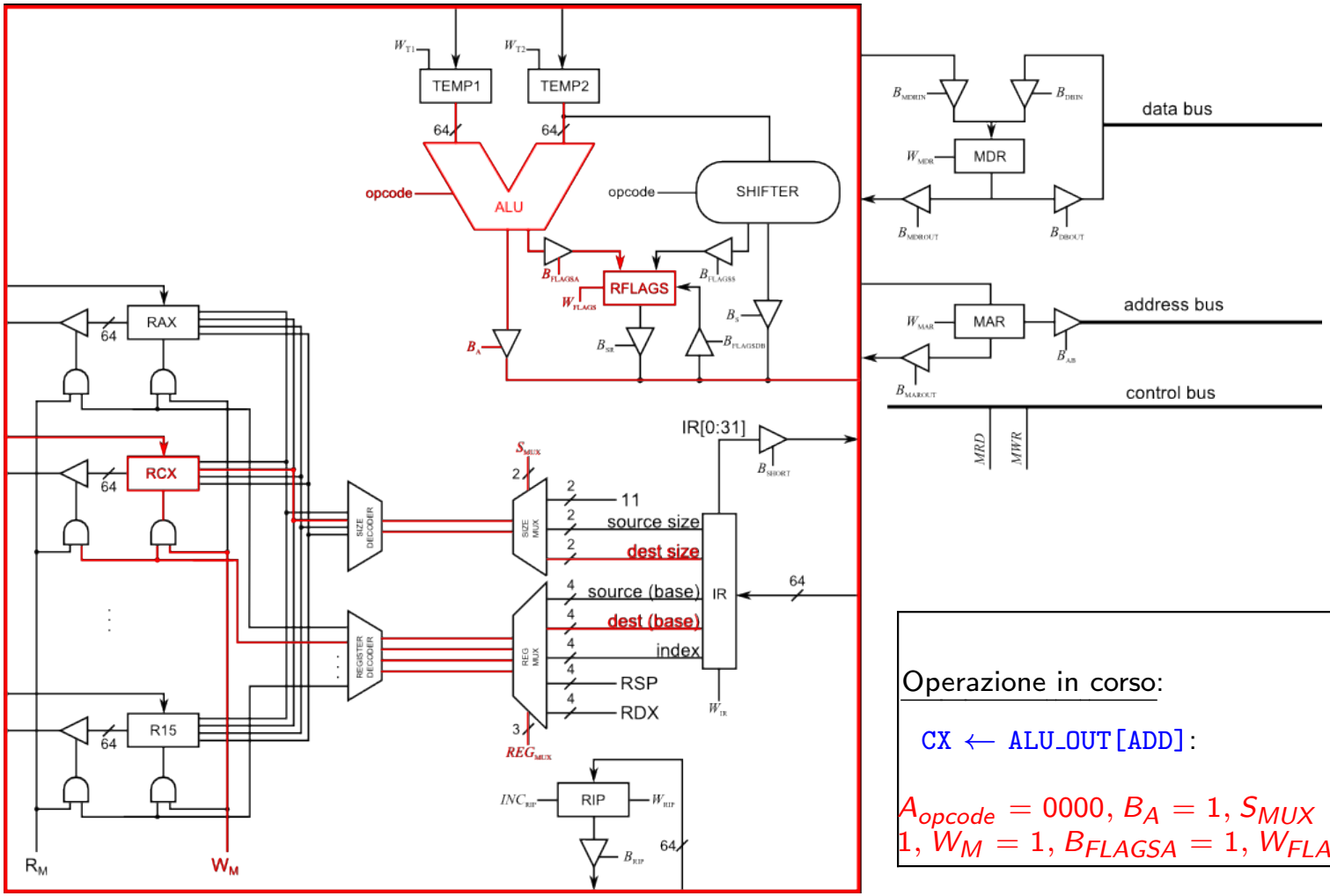


Operazione in corso:

TEMP2 ← CX:

$S_{MUX} = 2, REG_{MUX} = 1, R_M = 1, W_{T2} = 1$

Istruzioni Aritmetiche e Logiche



Operazione in corso:

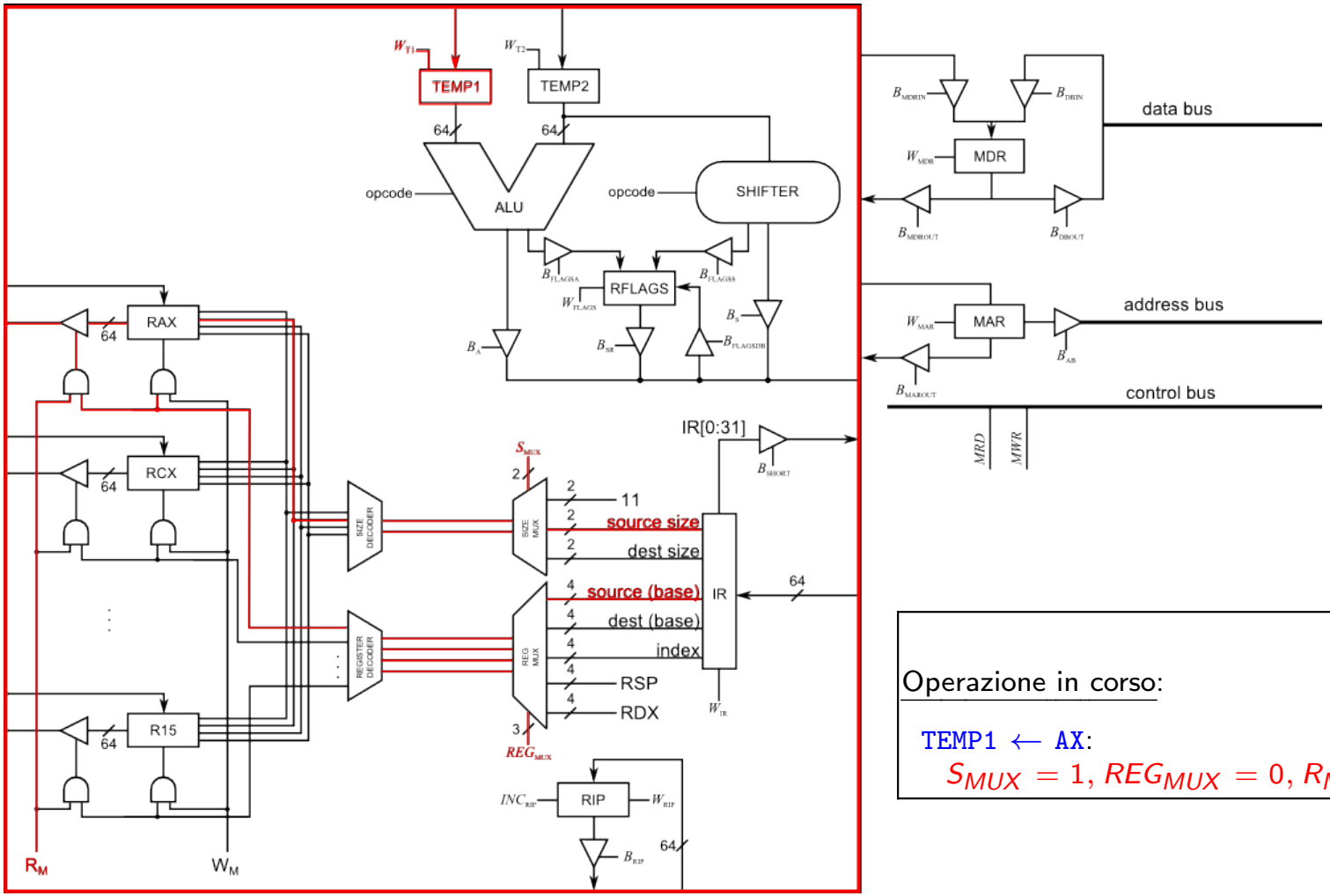
CX ← ALU_OUT[ADD]:

$A_{opcode} = 0000, B_A = 1, S_{MUX} = 2, REG_{MUX} = 1, W_M = 1, B_{FLAGSA} = 1, W_{FLAGS} = 1$

Istruzioni Aritmetiche e Logiche

- `andw %ax, %cx`:
  - `MAR` $\leftarrow$ `RIP`
  - `MDR` $\leftarrow$ `(MAR)`; `RIP` $\leftarrow$ `RIP + 8`
  - `IR` $\leftarrow$ `MDR`
  - `TEMP1` $\leftarrow$ `AX`
  - `TEMP2` $\leftarrow$ `CX`
  - `CX` $\leftarrow$ `ALU_OUT[AND]`

Istruzioni Aritmetiche e Logiche

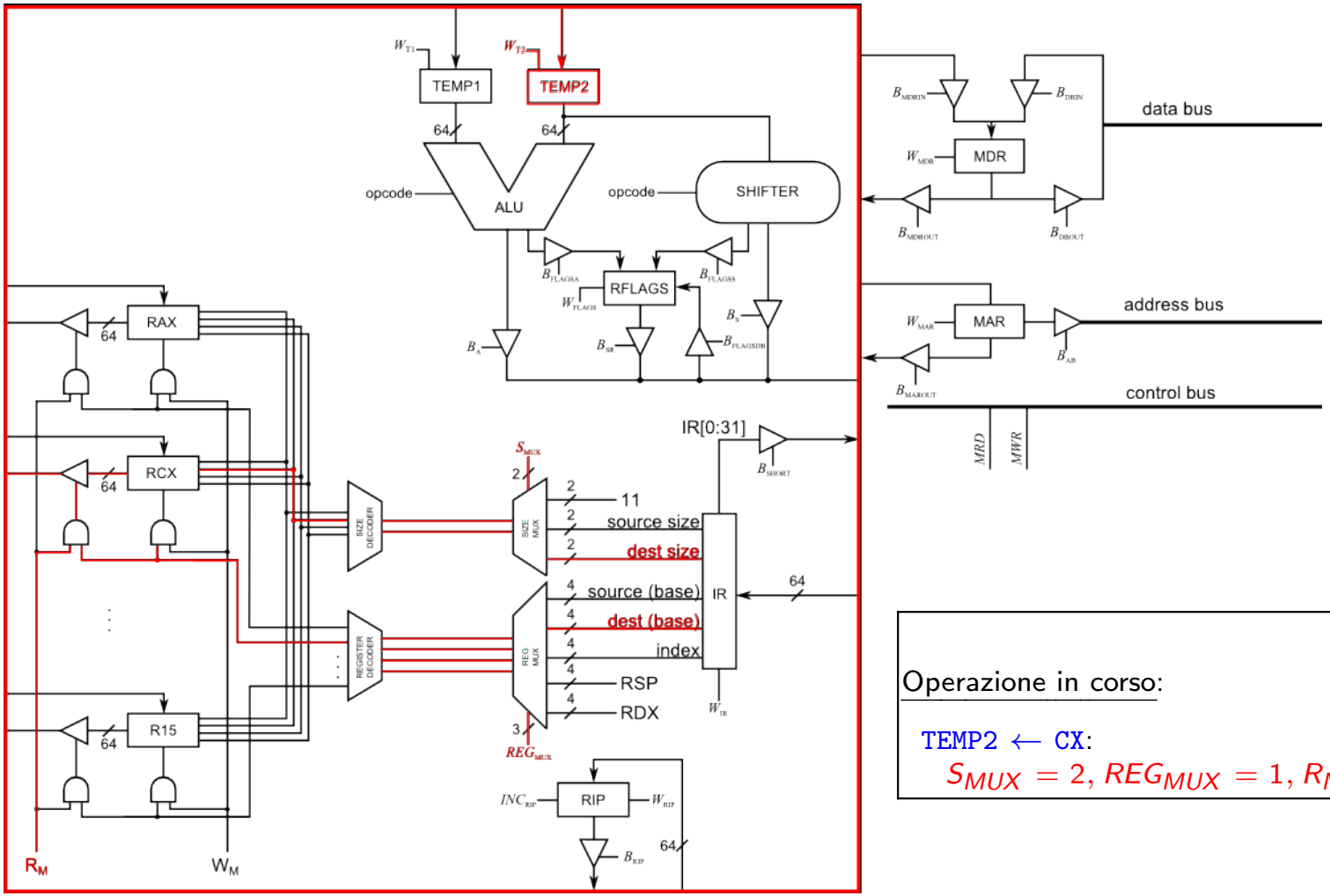


Operazione in corso:

TEMP1 ← AX:

$S_{MUX} = 1, REG_{MUX} = 0, R_M = 1, W_{T1} = 1$

Istruzioni Aritmetiche e Logiche

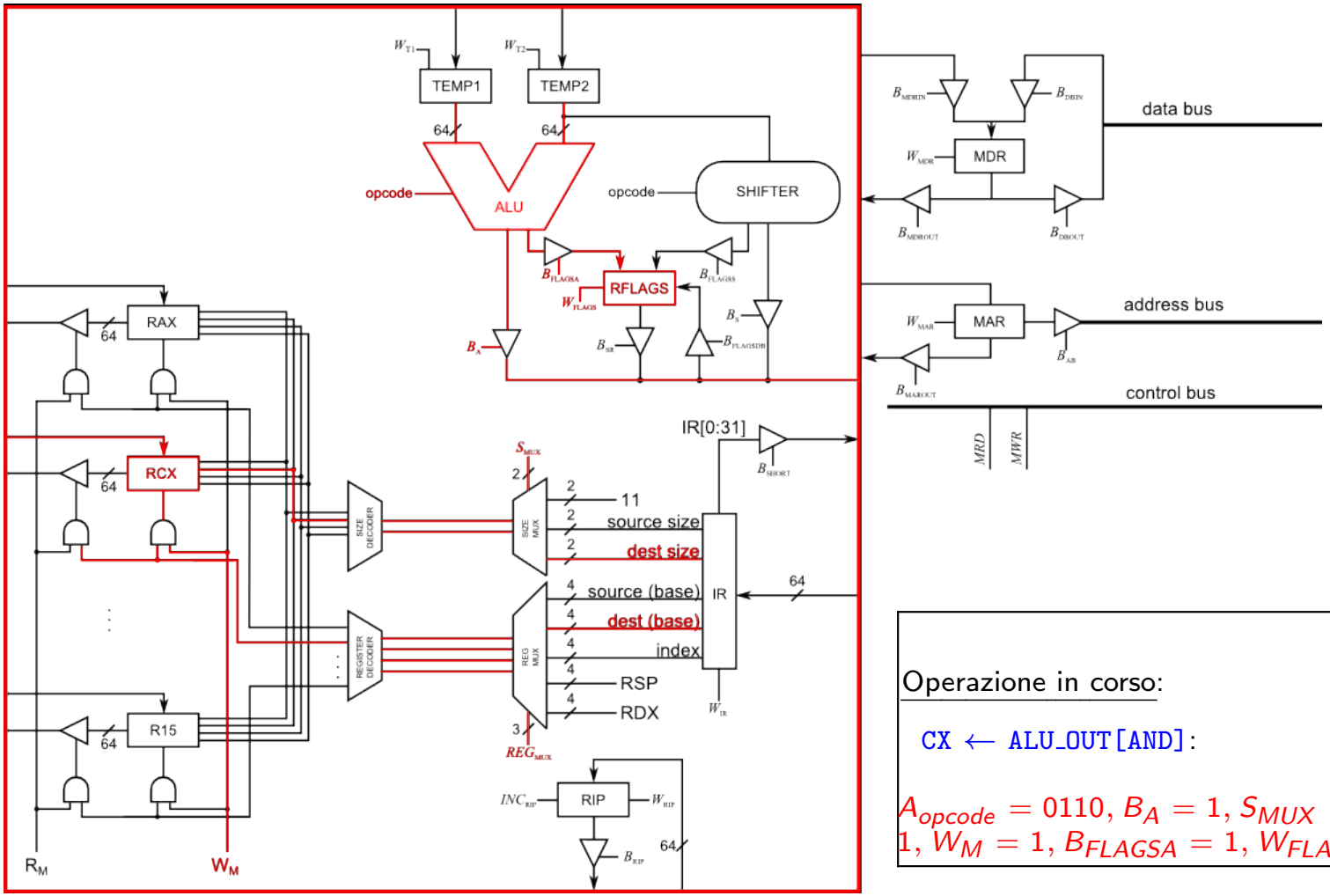


Operazione in corso:

TEMP2 ← CX:

$S_{MUX} = 2, REG_{MUX} = 1, R_M = 1, W_{T2} = 1$

Istruzioni Aritmetiche e Logiche



Operazione in corso:

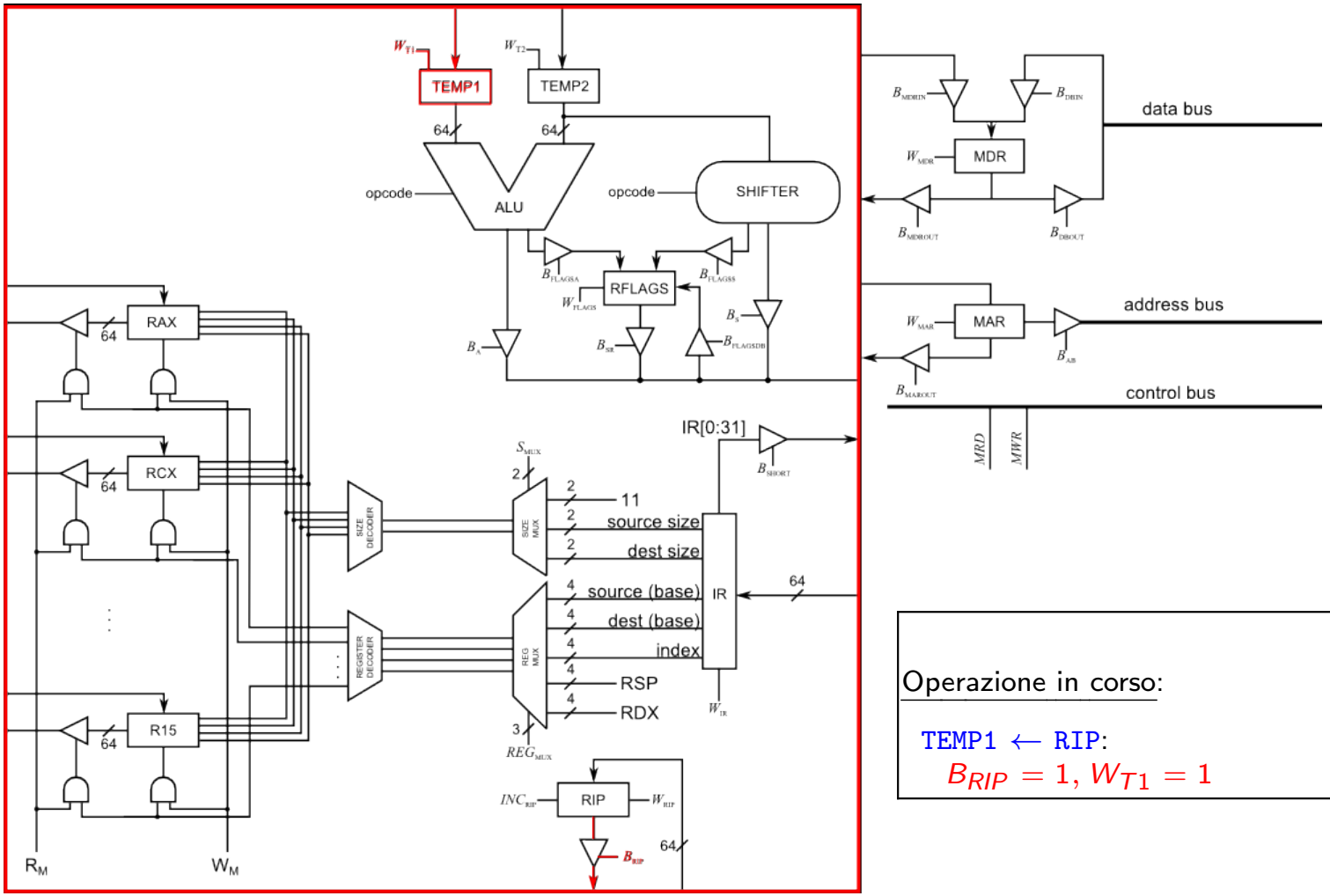
CX ← ALU_OUT[AND]:

$A_{opcode} = 0110, B_A = 1, S_{MUX} = 2, REG_{MUX} = 1, W_M = 1, B_{FLAGSA} = 1, W_{FLAGS} = 1$

Istruzioni di salto condizionale

- `jz displacement`:
  - `MAR` $\leftarrow$ `RIP`
  - `MDR` $\leftarrow$ `(MAR)`; `RIP` $\leftarrow$ `RIP + 8`
  - `IR` $\leftarrow$ `MDR`
  - `IF FLAGS[ZF] == 1 THEN`
  - `TEMP1` $\leftarrow$ `RIP`
  - `TEMP2` $\leftarrow$ `IR[0:31]`
  - `RIP` $\leftarrow$ `ALU_OUT[ADD]`
  - `ENDIF`

Istruzioni di salto condizionale

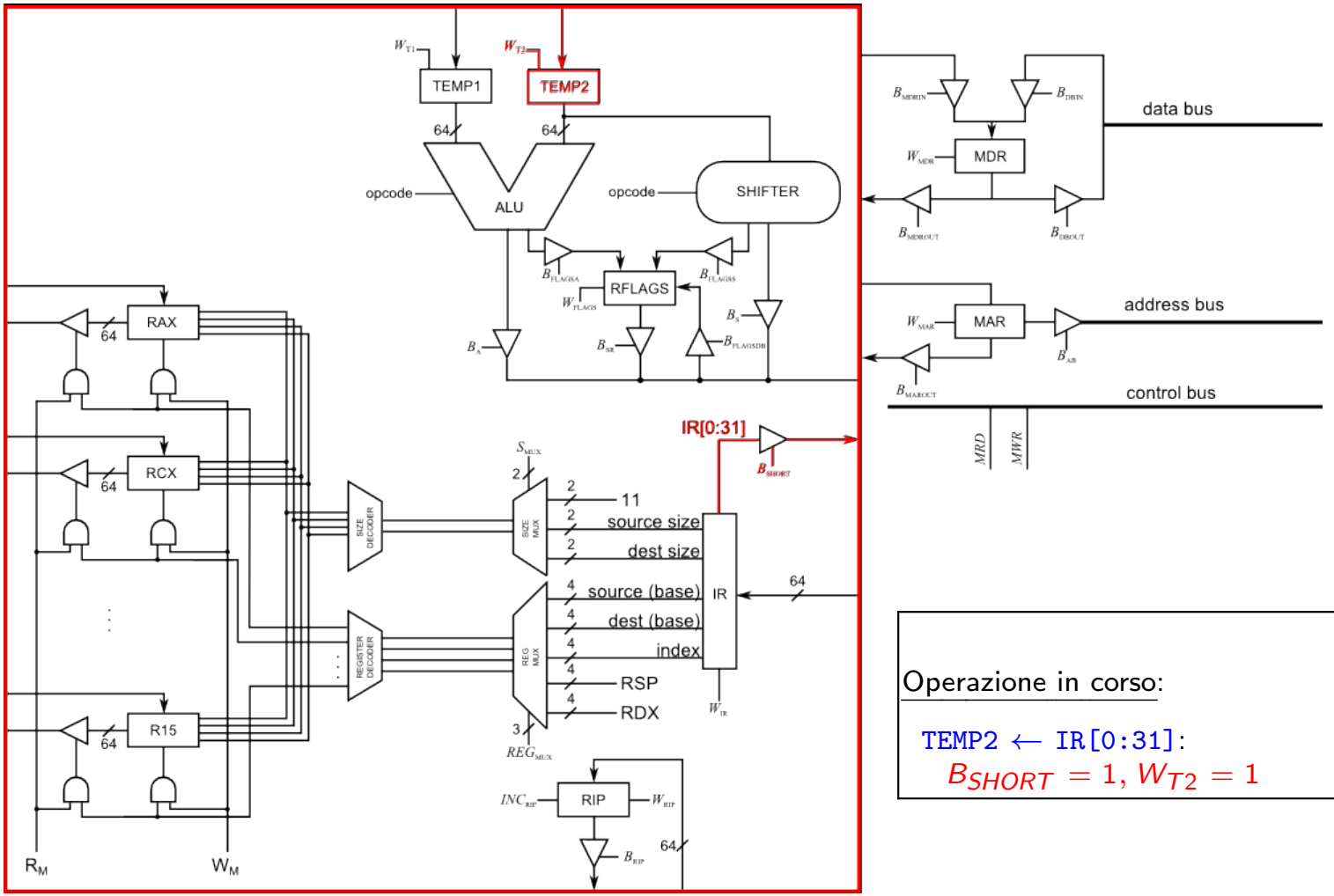


Operazione in corso:

TEMP1 $\leftarrow$ RIP:

$B_{RIP} = 1, W_{T1} = 1$

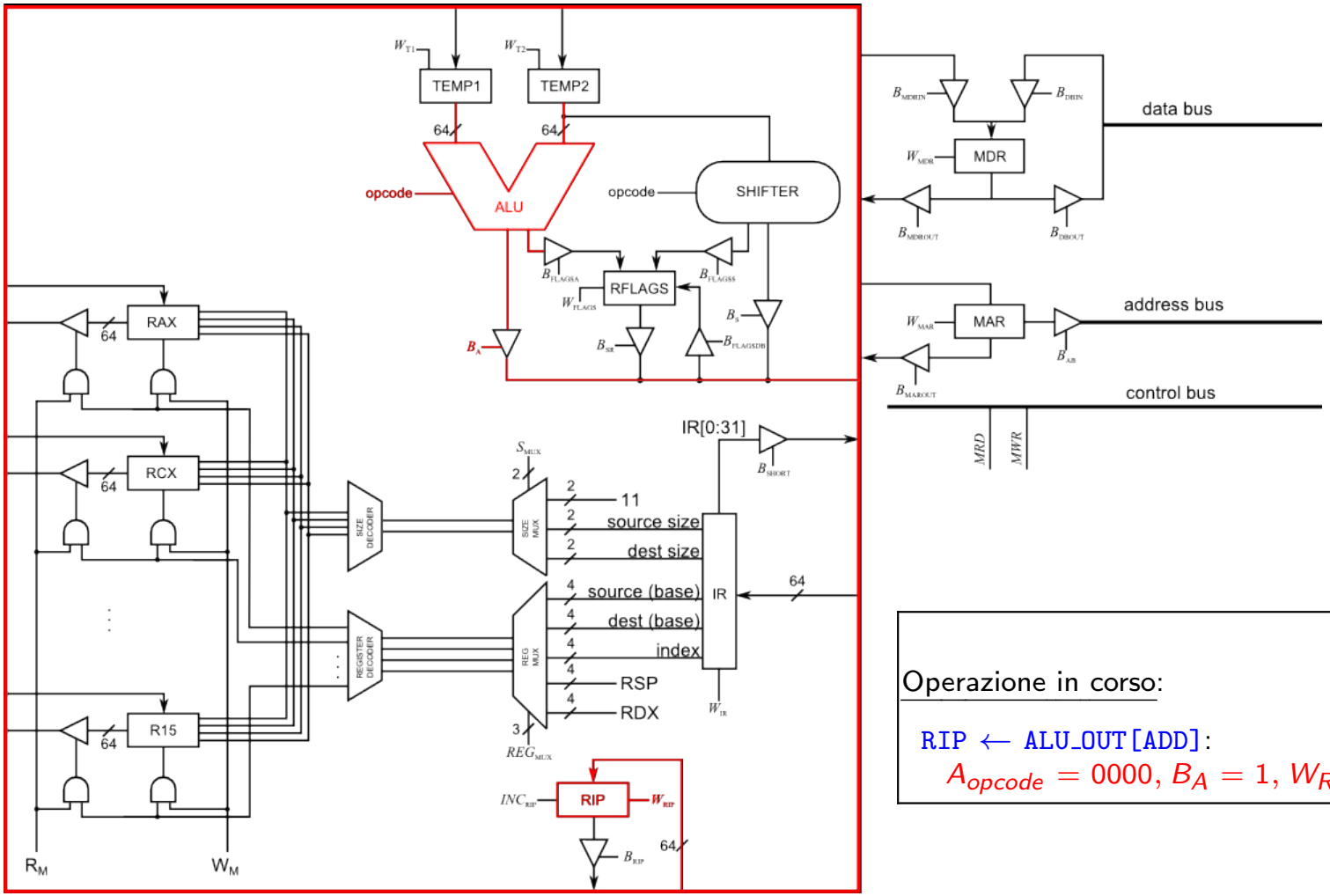
Istruzioni di salto condizionale



Operazione in corso:

TEMP2 ← IR[0:31]:
$B_{SHORT} = 1, W_{T2} = 1$

Istruzioni di salto condizionale



Operazione in corso:

RIP ← ALU_OUT[ADD]:

$A_{opcode} = 0000, B_A = 1, W_{RIP} = 1$