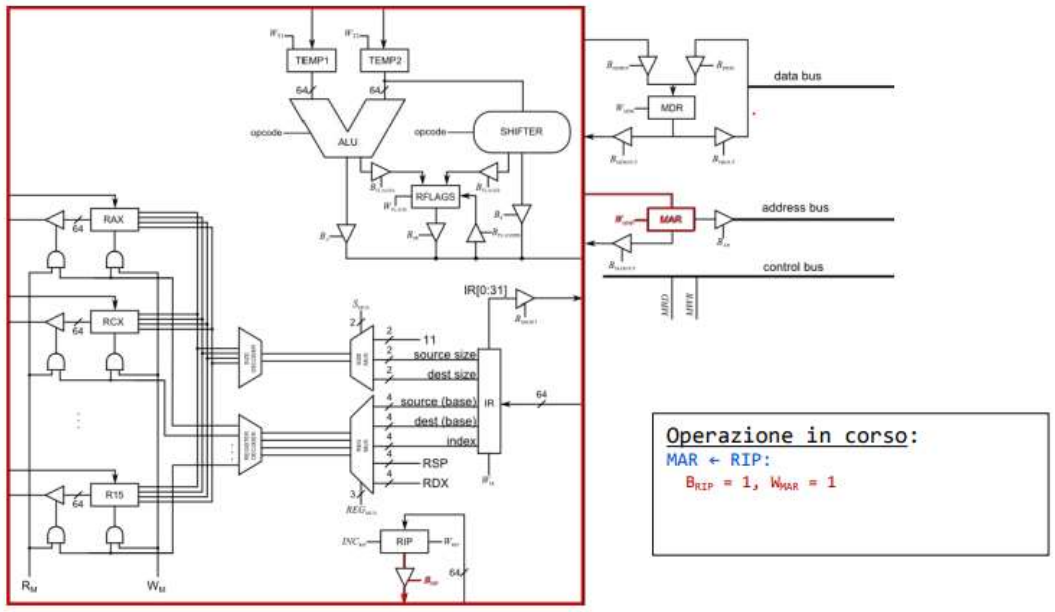
Lezione 21 Z64 7 CU 2

giovedì 16 novembre 2023 17:34

Andiamo a vedere ora in dettaglio le micro-istruzioni : **iniziamo dal fetch** (micro-programma anche esso) :

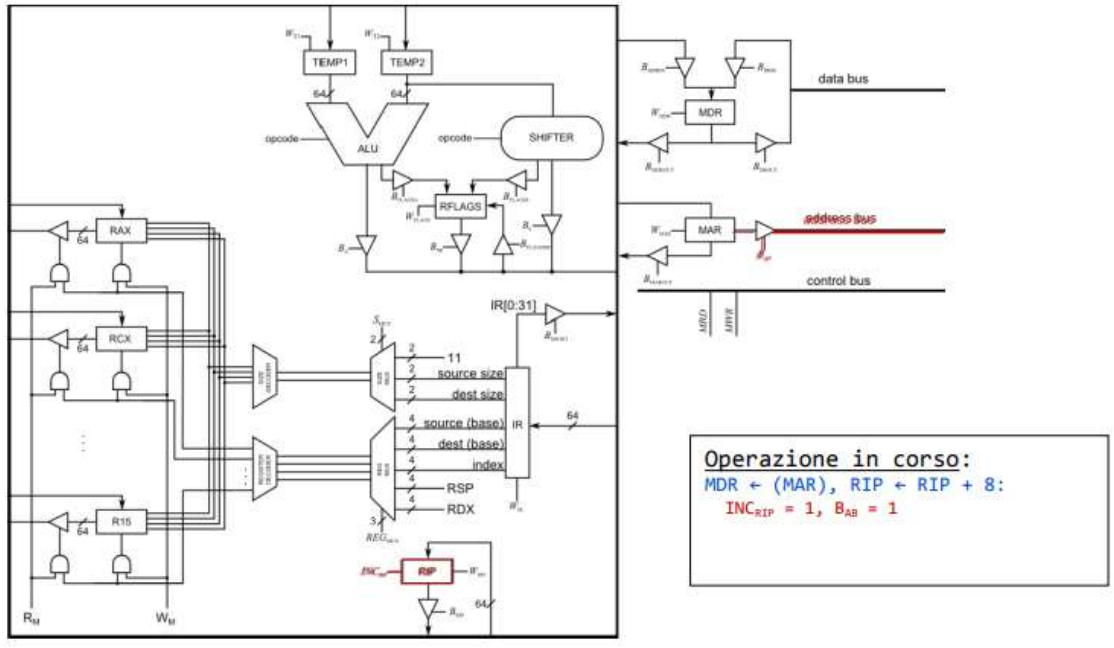
$$MAR \leftarrow RIP$$
$$MDR \leftarrow (MAR);\ RIP \leftarrow RIP + 8$$
$$IR \leftarrow MDR$$

In dettaglio :



In questa prima micro-istruzione dobbiamo spostare RIP in MAR : quindi i segnali che si attivano sono i seguenti : Brip impostato ad 1 per abilitare il buffer-tri-state che permette la scrittura dei dati sull'internal bus data . Questi dati viaggiano fino al registro MAR , il quale ha il segnale di Wmar settato ad 1 per abilitare la scrittura dentro esso.

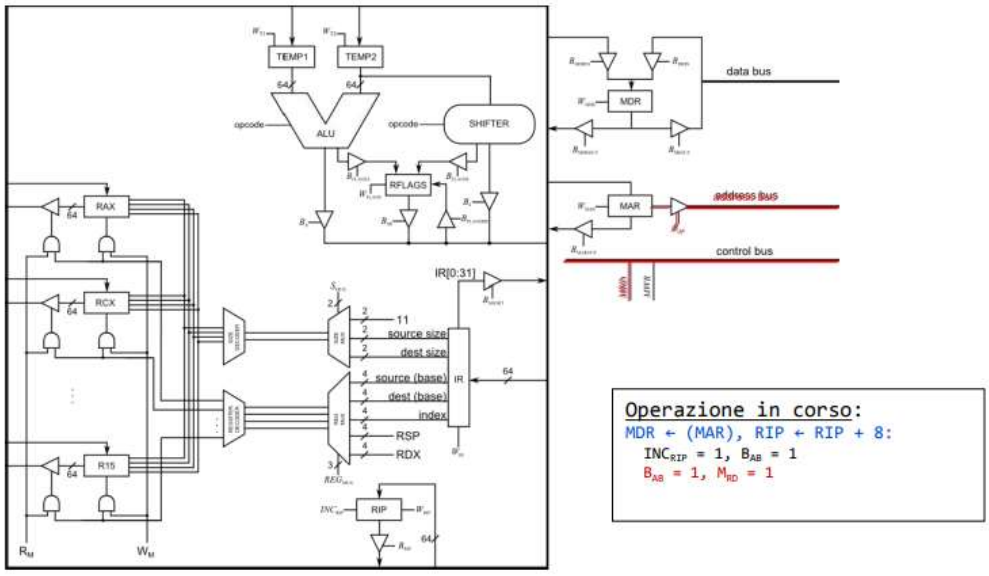
Seconda micro-istruzione:



Qui chiediamo alla memoria di darci il dato ed in parallelo incrementiamo RIP. Notiamo che in questa esecuzione non ci basta un solo colpo di clock , visto che la memoria è più lenta : RAM impiega 3 colpi di clock per eseguire. Per mandare indirizzo su address bus devo abilitare il buffer-

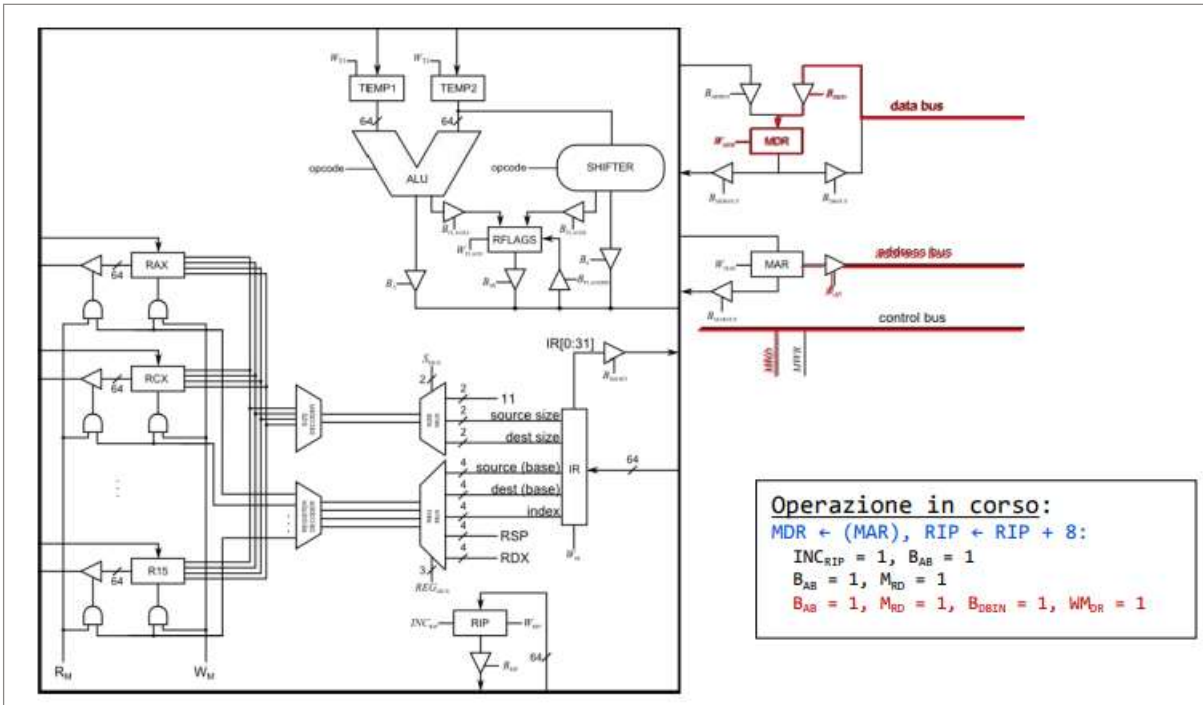
tri-state (Bab =1), mentre nel contempo e vista l'indipendenza / esecuzione parallela istruzioni, incremento anche RIP (IncRip=1).

Vediamo il secondo step :



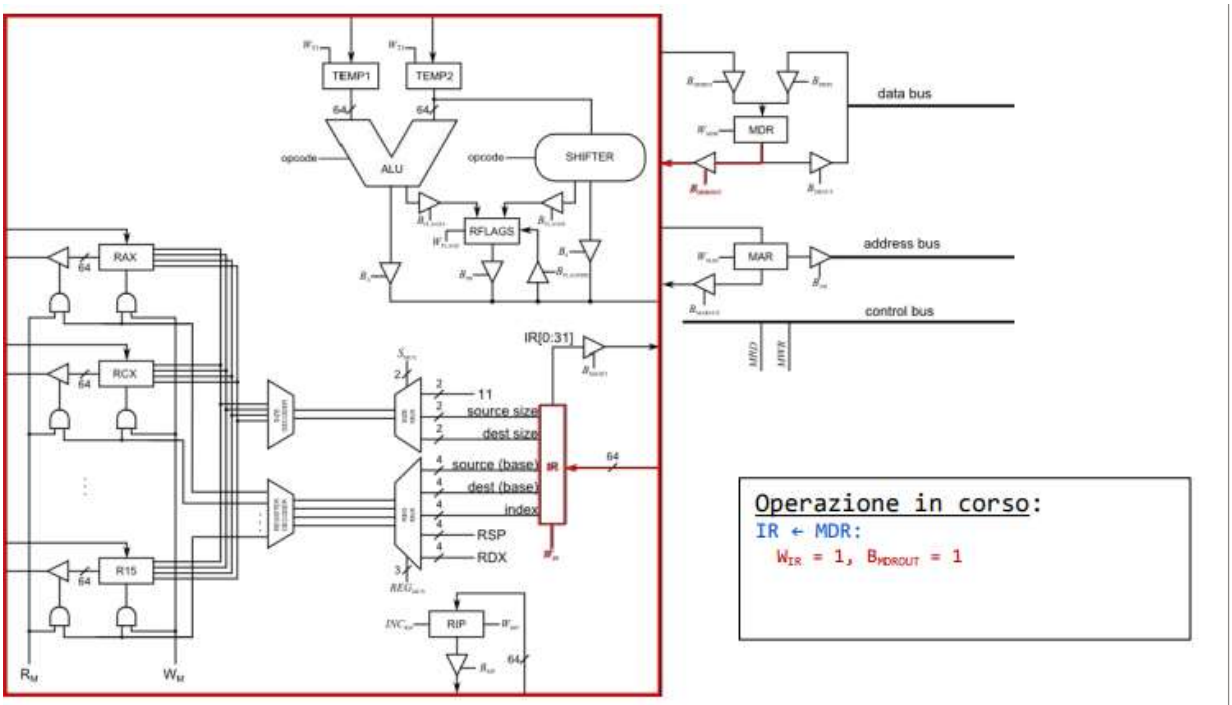
In questo step , devo mantenere il dato costante su address bus, quindi tengo abilitato ancora il buffer-tri-state di MAR , ed inoltre visto che devo scrivere sul control bus, abilito il segnale di scrittura su di esso (Mrd=1).

Ultimo colpo di clock : bufferizzo all'interno del processore i dati che stanno arrivando dal data bus



Per effettuare la scrittura finale su MDR , si mantiene il buffer-tri-state di MAR attivo (Bab=1), cosi i dato escono su address bus, si mantiene il control bus in scrittura (Mrd=1), si abilita il buffer-tri-state di MDR per fare entrare i dati dal data bus (Bdbin=1) e si abilita il segale di scrittura su MDR (Wmdr=1)

Mentre nell'ultima fase della micro-istruzione andiamo a copiare MDR(codifica dell'istruzione) nell'IR :



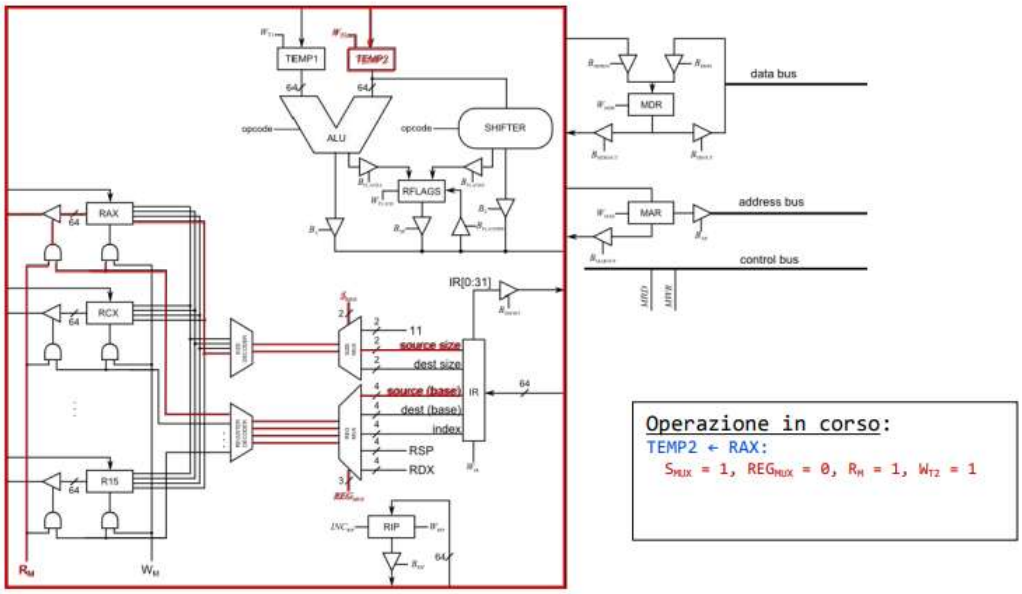
Visto che devo copiare il contenuto di MDR in IR , abilito il buffer-tri-state per permettere scrittura sull'internal bus data (Bmdrout=1), poi i dati viaggiano su internal bus data, fino ad arrivare all'IR : serve il segnale di scrittura abilitato per fare ciò : Wir=1. **Quindi per fare la fase di fetch servono necessariamente 4 cicli di clock.**

Vediamo altri esempi :

1. `movq %rax, %rcx:`
   - MAR ← RIP
   - MDR ← (MAR); RIP ← RIP + 8
   - IR ← MDR
   - TEMP2 ← RAX
   - RCX ← TEMP2

   a. Da notare che per qualunque micro-istruzione esegue la fase di fetch .
   b. Vediamo ora la micro-istruzione TEMP2<-RAX

i.

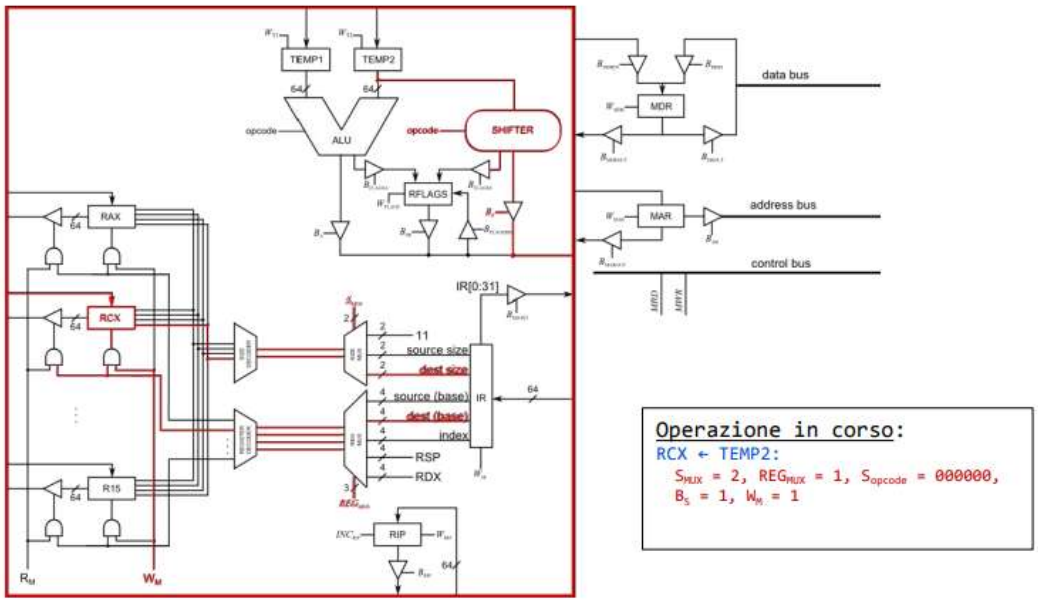


ii. Notiamo che usiamo TEMP2 (registro tampone) , non passo per la ALU, ma passo per lo shifter , il quale in questo caso fa lo shift di 0 posizioni : il dato passa cosi' come è ed inoltre perché posso usare 1 solo registro alla volta. Per attivare la lettura da RAX devo abilitare Rm =1 (da uc) , il quale permette di selezionare un qualunque registro, quindi per selezionare solo quel registro devo prendere gli

input dei decoder : codice del registro e taglia del registro virtuale . Queste informazioni vengono prese dall'IR : codice sorgente e taglia sorgente : piloto i mux : Smux=1 (taglia) e Regmux =0 (sorgente), attivo buffer-tri-state di RAX (permette il viaggio dei dati su internal data bus) e raggiungo TEMP2. Infine per attivare la scrittura su TEMP2 , devo abilitare il segnale di write (Wt2=1).

c. Vediamo ora RCX<-TEMP2

i.



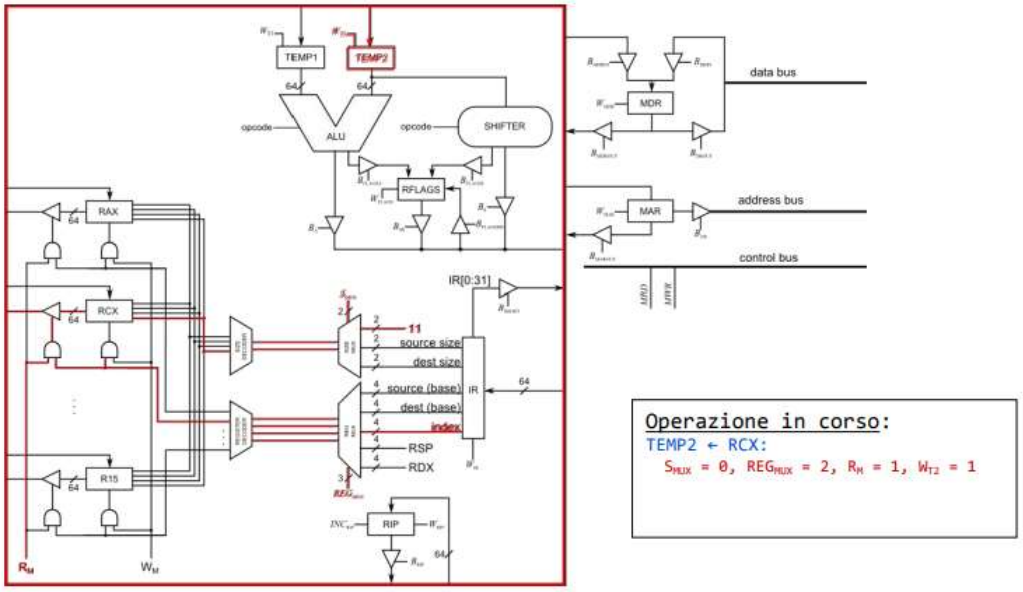
ii. Per abilitare la scrittura di temp2 sull'internal data bus, devo passare per lo shifter ed abilitare il corrispondente buffer-tri-state (Sopcode=000000, Bs=1), quindi i dati arrivano dentro RCX , il quale ha impostato ad 1 il segnale di scrittura (Wm=1). Per quanto riguarda invece i mux : seleziono per quello sorgente il secondo ingresso (Smux=2), mentre per quello della dimensione prendo la seconda linea (Regmax= 1 : registro destinazione).

2. `movq %rax, 0xaaaa(%rax, %rcx, 8):`
   - MAR ← RIP
   - MDR ← (MAR); RIP ← RIP + 8
   - IR ← MDR
   - TEMP2 ← RCX
   - TEMP1 ← Shifter Out[SHL, 000011]
   - TEMP2 ← RAX
   - MAR ← ALU OUT[ADD]
   - TEMP1 ← IR[0:31]
   - TEMP2 ← MAR
   - MAR ← ALU OUT[ADD]
   - MDR ← RAX
   - (MAR) ← MDR

a. Modalità di indirizzamento completa (base , indice , scala e spiazzamento)

b. Partiamo da TEMP2<-RCX

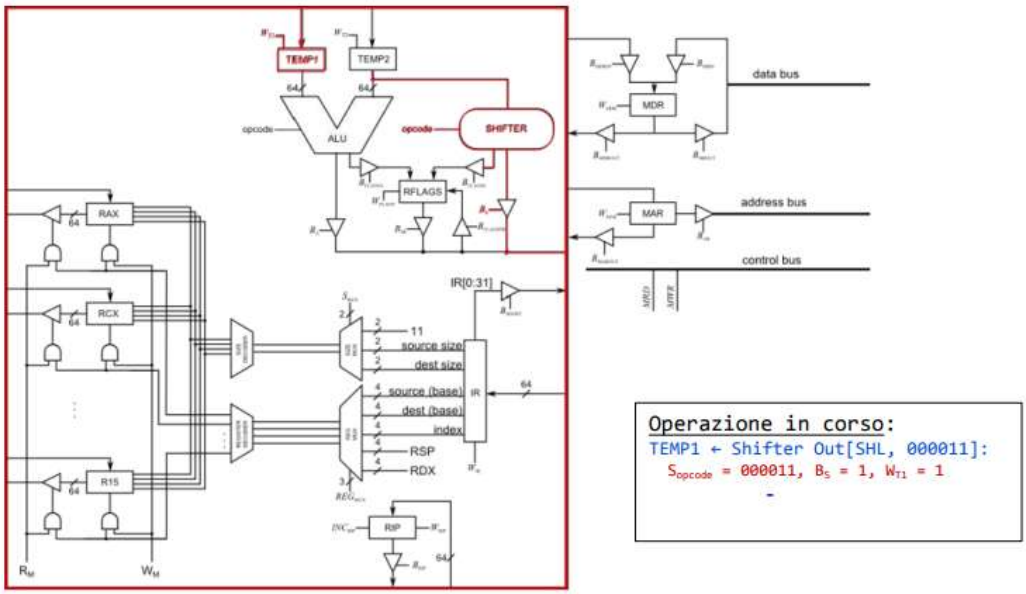
i.



ii. Discorso analogo a prima : riuso componenti, ma con diversa configurazione.

c. TEMP1<-shifter out [shl,000011] : 000011 rappresenta il codice dello shifter (3) il che equivale a shiftare di 8 :

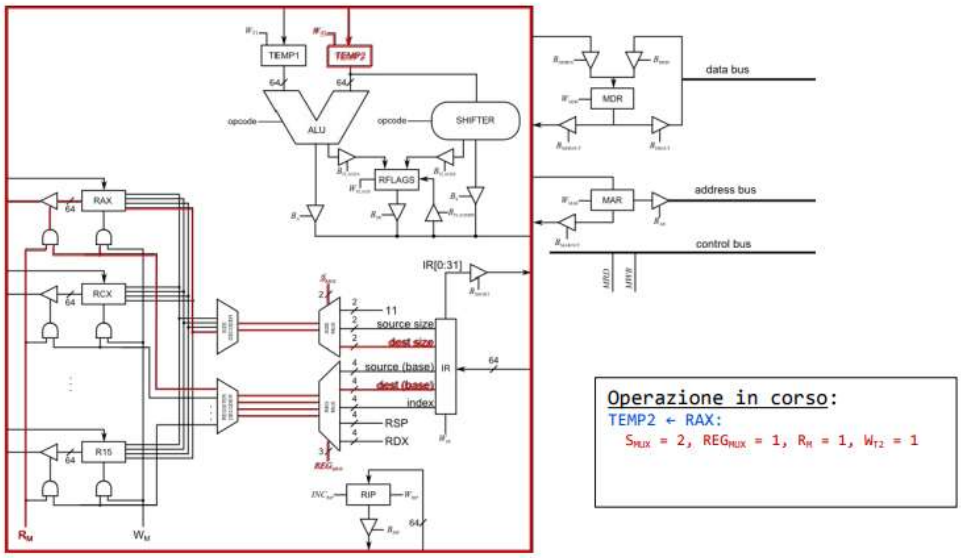
i.



ii. Dopo aver shiftato di 3 posizioni , abilito il buffer-tri-state per mandare i dati su internal data bus, e questi dati li scrivo su TEMP1 , abilitando il segnale opportuno (Wtemo1=1) .

d. TEMP2<-RAX:

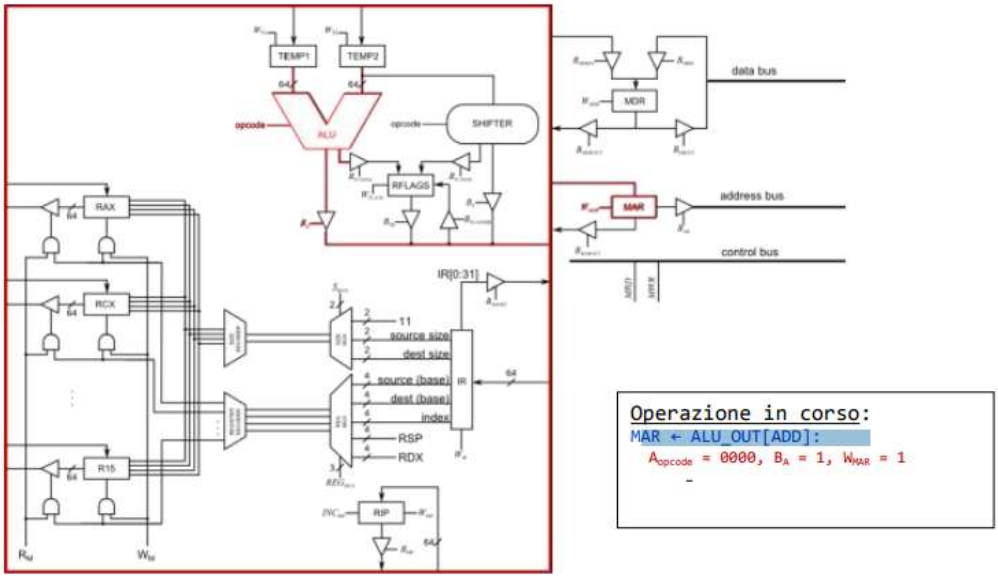
i.



ii. Setto i segnali nel giusto ordine : Smux=2 (taglia della destinazione), Regmux=1 (tipo del registro destinazione) , poi abilito il segnale di scrittura nei registri (Rm=1) ed infine abilito la scrittura su TEMP2 (Wt2=1)

e. MAR ← ALU_OUT[ADD]:

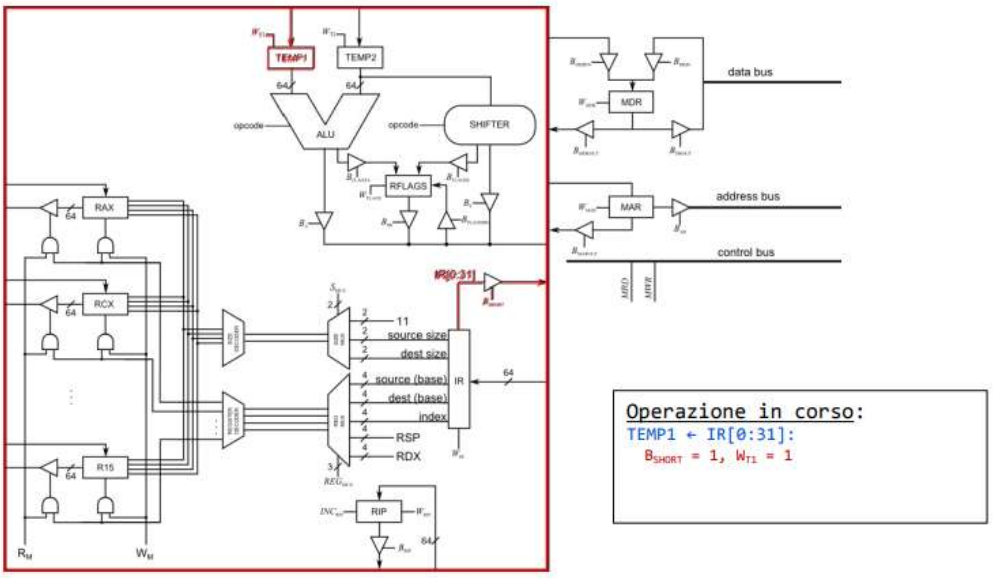
i.



ii. Qui passo per ALU: effettua la somma (Aopcode=0000) , abilito il buffer-tri-state per scrivere sull'internal data bus (Ba=1) ed infine scrivo su MAR , abilitando anche qui il segnale (Wmar=1).

f. TEMP1<-IR[0:31] : cosi calcolo lo spiazzamento

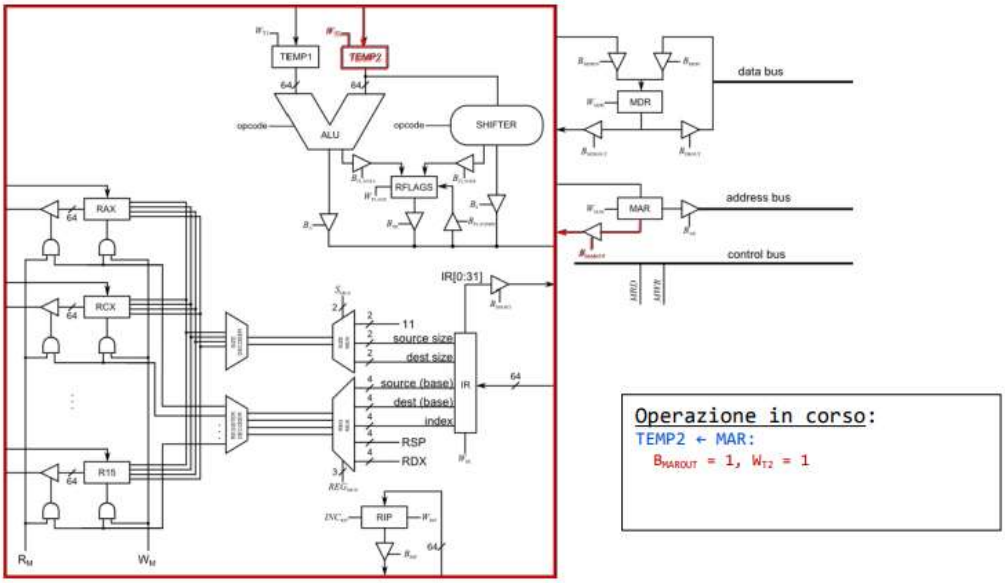
i.



ii. Prendo i 32 bit dello spiazzamento da IR, abilito il suo buffer-tri-state (Bshort=1), cosi i dato viaggiano sull'internal bus data ed infine li scrivo in TEMP1, abilitando il segnale di scrittura (Wt1=1).

g. TEMP2<-MAR:

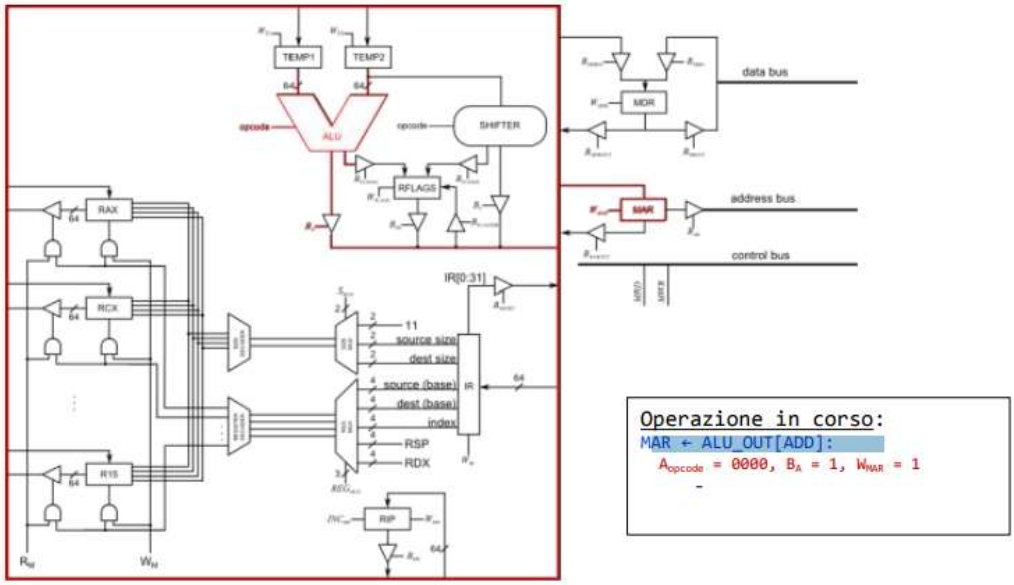
i.



ii. Qui abilito il buffer-tri-state di MAR per scrivere su interna bus data (Bmarout=1), cosi i dati viaggiano sull'internal bus data, arrivando a TEMP2 e vi vengono scritti in quanto il segnale di scrittura è abilitato (Wt2=1).

h. MAR ← ALU_OUT[ADD]:

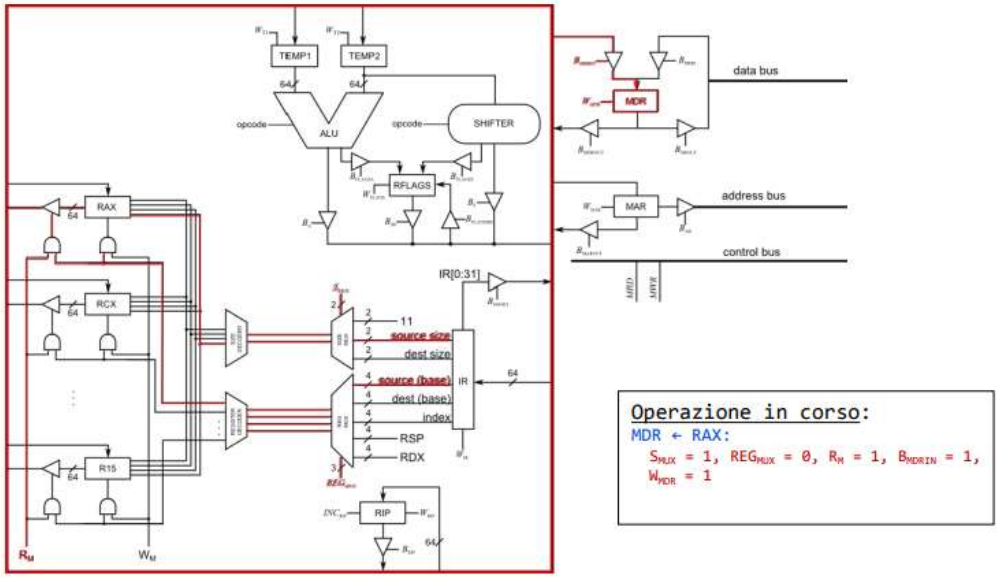
i.



ii. Sposto base*indice+ spiazzamento in MAR: faccio fare all'alu la somma (Aopcode= 0000) , abilito il buffer-tri-state per scrivere su internal bus data (Ba=1) , facendo viaggiare i dati fino a MAR ed abilitando il segnale di scrittura (Wmar=1).

i. MDR<-RAX:

i.

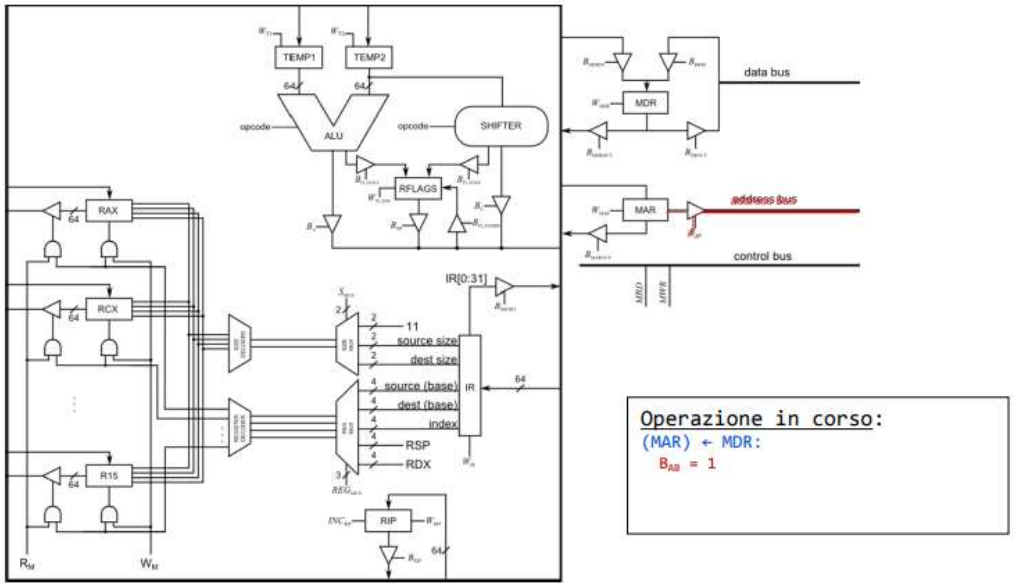


ii. Cominciamo dai mux: Smux=1 in quanto devo lavorare con il registro sorgente (RAX) , RegMUX=0 (registro base) , poi abilito la lettura dai registri (Rm=1), abilito il buffer-tri-state per mandare i dati sull'internal data bus , poi li mando a MDR abilitando il buffer-tri-state in entrata (Bmdrin=1), ed infine scrivo il valore abilitando il segnale di scrittura (Wmdr=1).

j. (MAR)<-MDR:

i. Primo step

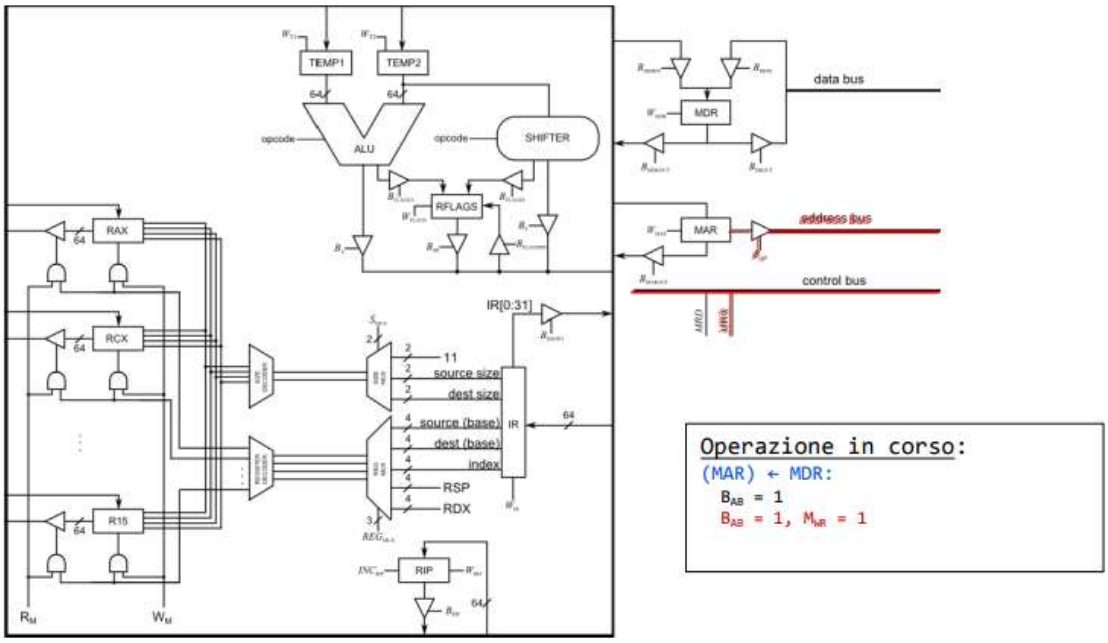
1)



2) Abilito il buffer-tri-state di MAR per scrivere i dati sull'address bus (Bab=1)

ii. Secondo step :

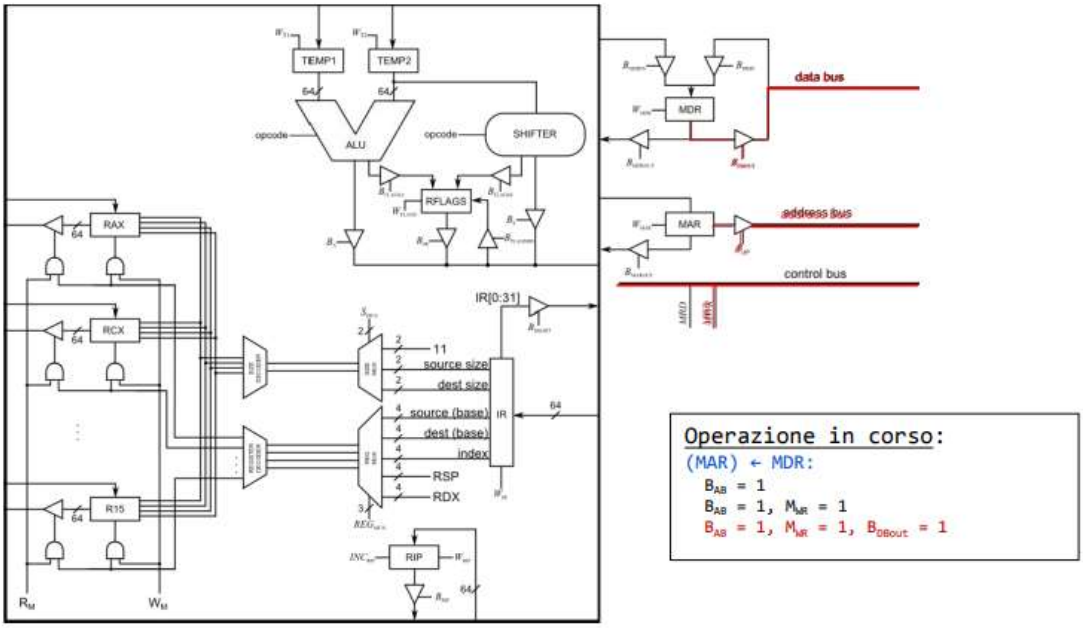
1)



2) Per seconda micro-istruzione , abilito il control bus, ed in particolare la scrittura su ste stesso : (Bab=1 e MWR=1)

iii. Terzo step:

1)



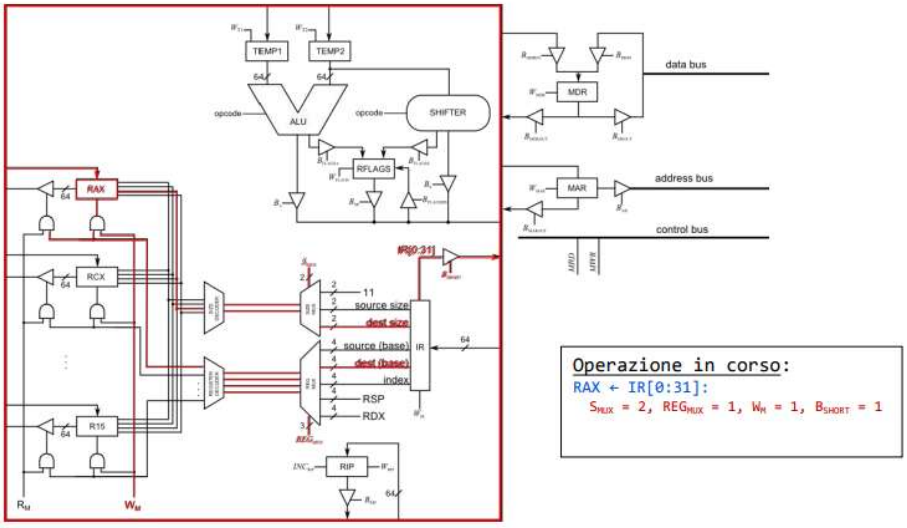
2) Abilito il buffer-tri-state di MDR (Bmdrout=1) cosi i dati viaggiano nel data bus.

3.

`movl $0xaaaa, %eax:`

- MAR ← RIP
- MDR ← (MAR); RIP ← RIP + 8
- IR ← MDR
- EAX ← IR[0:31]

a.



b. Non c'è spiazzamento , e copia i 32 bit meno significativi di IR in RAX : seleziono Smux= 2 , Regmux=1 , abilito il segnale per scrivere sui registri (Wm=1) e abilito anche il buffer-tri-state di IR (Bshort=1).

4.

`movq $0xaaaa, %rax:`

- MAR ← RIP
- MDR ← (MAR); RIP ← RIP + 8
- IR ← MDR
- MAR ← RIP
- MDR ← (MAR); RIP ← RIP + 8
- RAX ← MDR

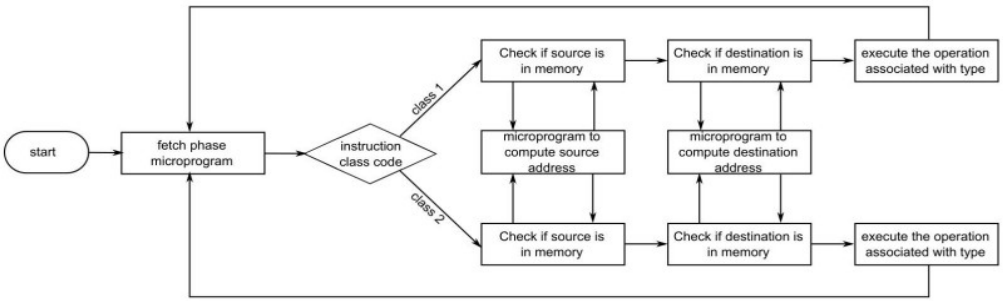
a. Qui stessa costante di prima , ma a 64 bit, quindi abbiamo bisogno di 128 bit .
b. In questo caso IR punta nel mezzo dell'istruzione !!
c. Devo per forza trasferire alla memoria : devo sistemare anche il IR .

Quindi in base al tipo di operandi che si hanno , l'unita di controllo ha interazioni diverse (modalità di indirizzamento) : si cerca di ridurre il peso delle micro-operazioni , magari usando delle micro-

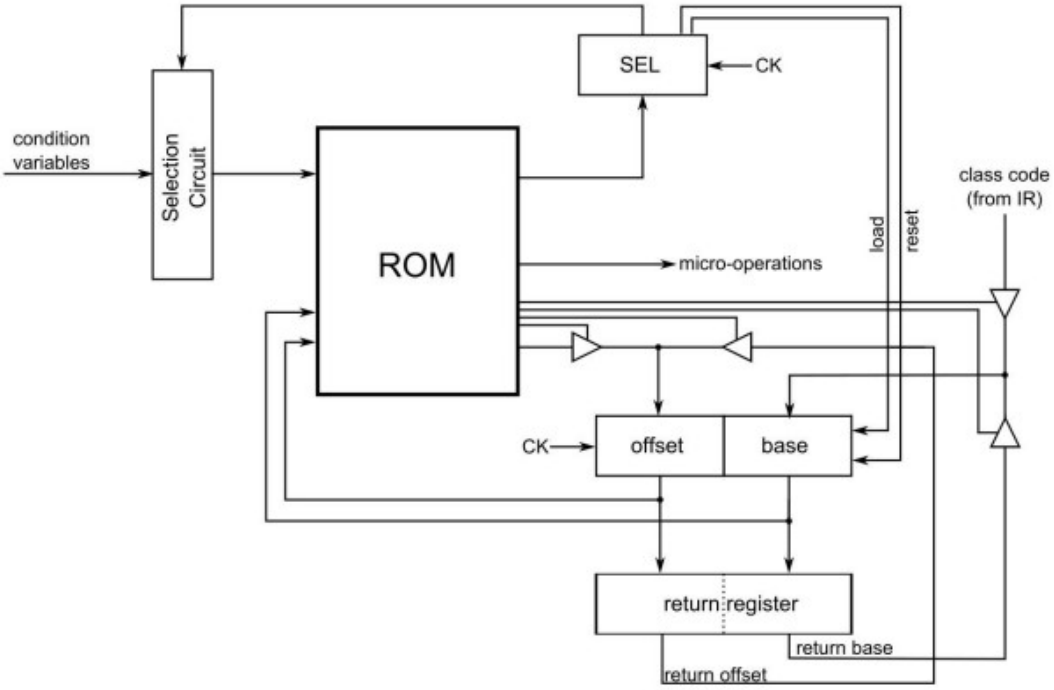
funzioni/micro-programma (salvato nella ROM) :

```
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
```

I microprogrammi vanno a finire nelle pagine vuote della ROM (ve ne sono 8 "Libere"). Quindi di conseguenza devo modificare la mia macchina a stati , che diventa :



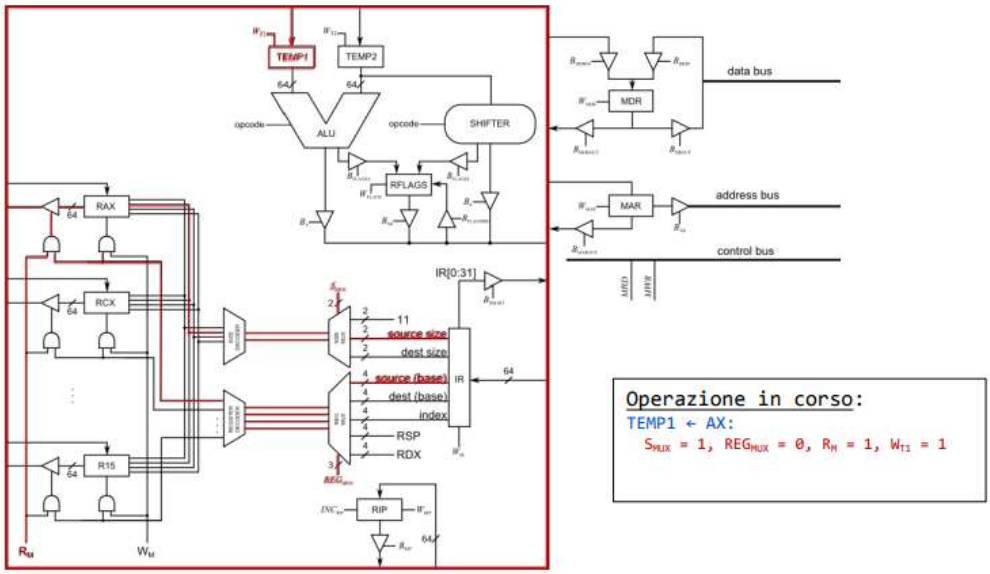
Mentre il circuito diventa cosi :



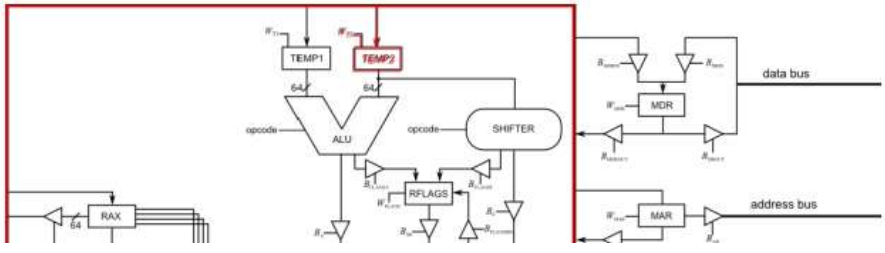
Vediamo ora altri esempi :

1. `addw %ax, %cx:`
   - `MAR ← RIP`
   - `MDR ← (MAR); RIP ← RIP + 8`
   - `IR ← MDR`
   - `TEMP1 ← AX`
   - `TEMP2 ← CX`
   - `CX ← ALU OUT[ADD]`

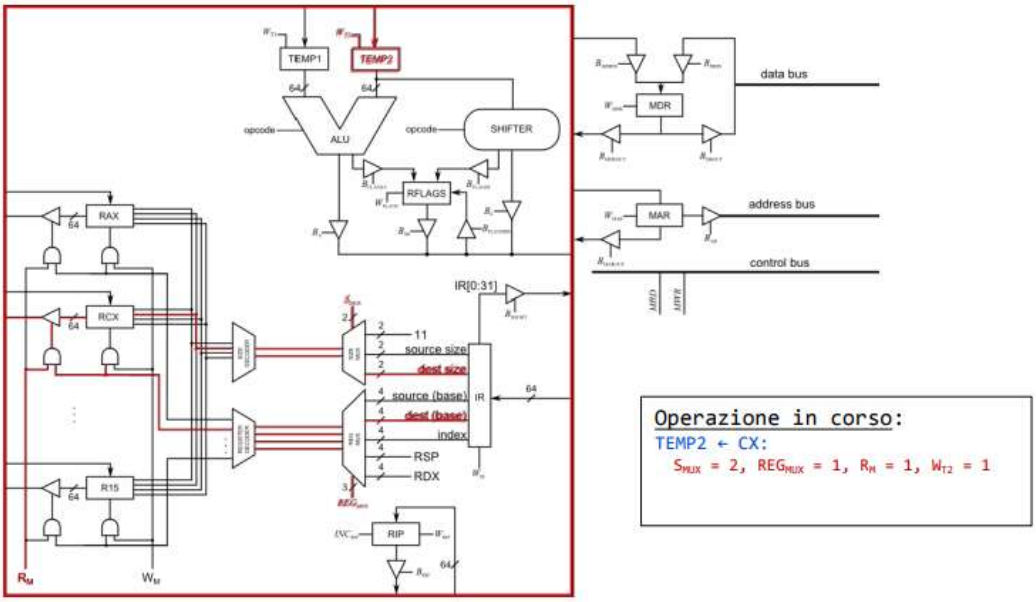
a.



b. Cominciamo con il configurare il mux delle sorgenti a 1 (Sreg=1), quello delle taglie a 0 (Regmux=0), abilitiamo il segnale di lettura del banco registri (Rm=1), abilitiamo il buffer-tri-state del registro, cosi i dati viaggiano sull'internal bus data, arrivando a TEMP1 ed abilitiamo il segnale di scrittura (Wtemp1=1).

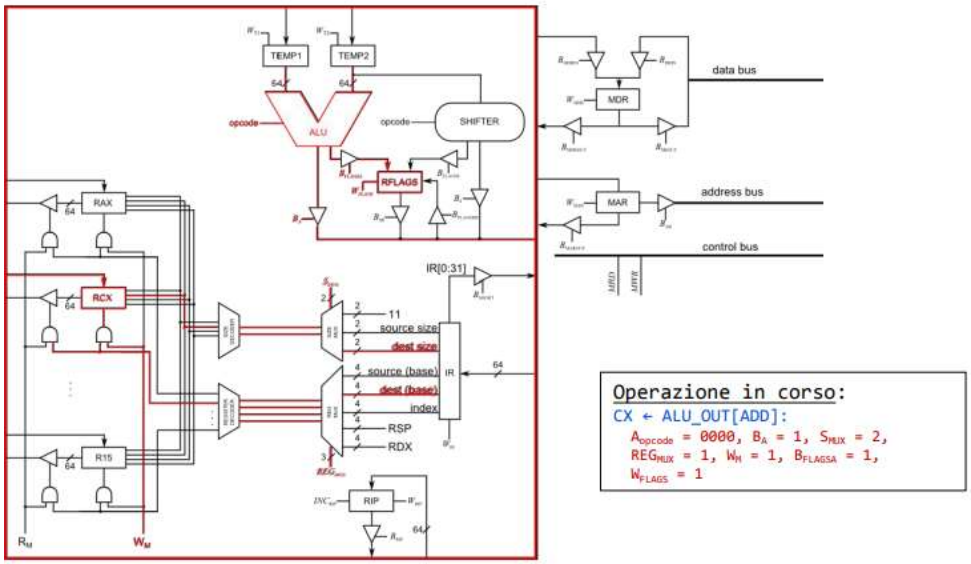


c.



d. visto che in questa micro-istruzione dobbiamo lavorare con un registro , settiamo i mux nei modi corretti : il primo (tipo) lo setto a dest(Smux=1), quello della taglia (secondo) lo setto ad 1 (Regmux=1)->registro base di dest , abilito il buffer-tri-state e lo mando a TEMP2, abilitando il segnale di scrittura su TEMP2(Wtemp2=1)

e.

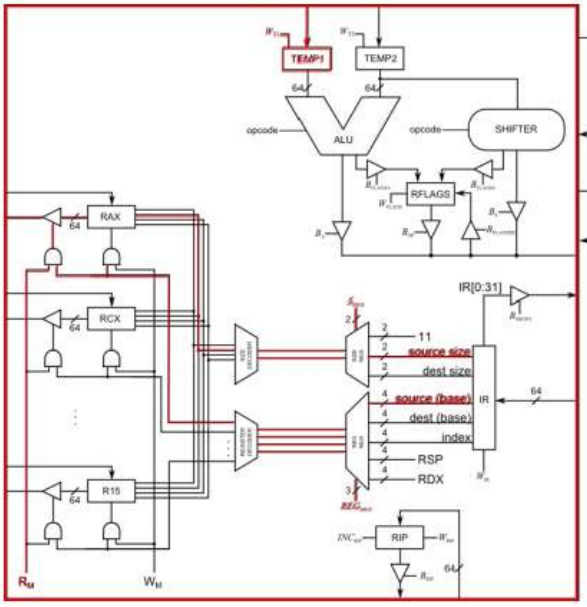


f. Per quanto riguarda il settaggio dei mux è lo stesso di prima ; concentriamoci ora sull'ALU: gli si passa opcode 0000 , si abilita il buffer-tri-state che permette di scrivere sull'internal bus data (Ba=1), abilito il buffer-tri-state per scrivere nel registro flags(Bflagsa=1) ed abilito il segnale di scrittura su Rflags (Wrflags=1); infine per scrivere nel registro devo abilitareil segnale di scrittura nei registri (Wreg=1).

2. andw %ax, %cx:
   - MAR ← RIP
   - MDR ← (MAR); RIP ← RIP + 8
   - IR ← MDR
   - TEMP1 ← AX
   - TEMP2 ← CX
   - CX ← ALU OUT[AND]

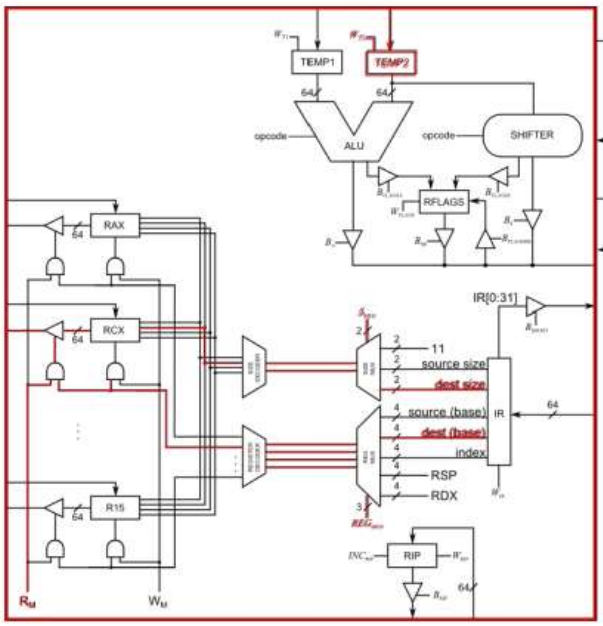
a. Simile a quello di sopra , ma cambia l'opcode della ALU : da 0000 a 0110 :

b.



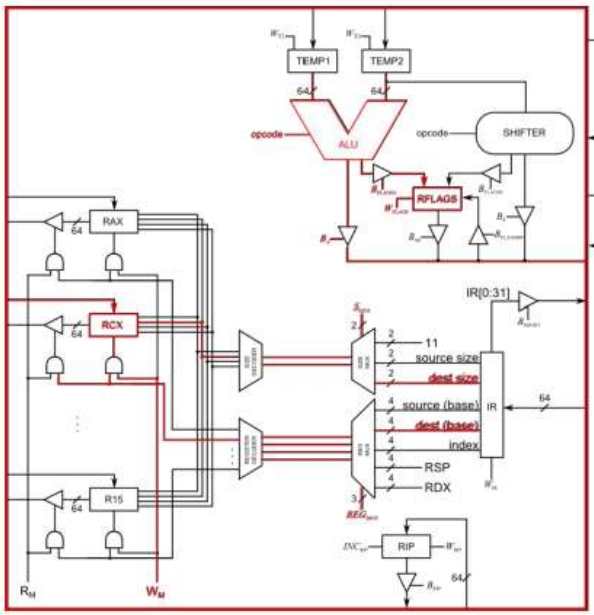
Operazione in corso:
TEMP1 ← AX:
S_{MUX} = 1, REG_{MUX} = 0, R_M = 1, W_{T1} = 1

c.



Operazione in corso:
TEMP2 ← CX:
S_{MUX} = 2, REG_{MUX} = 1, R_M = 1, W_{T2} = 1

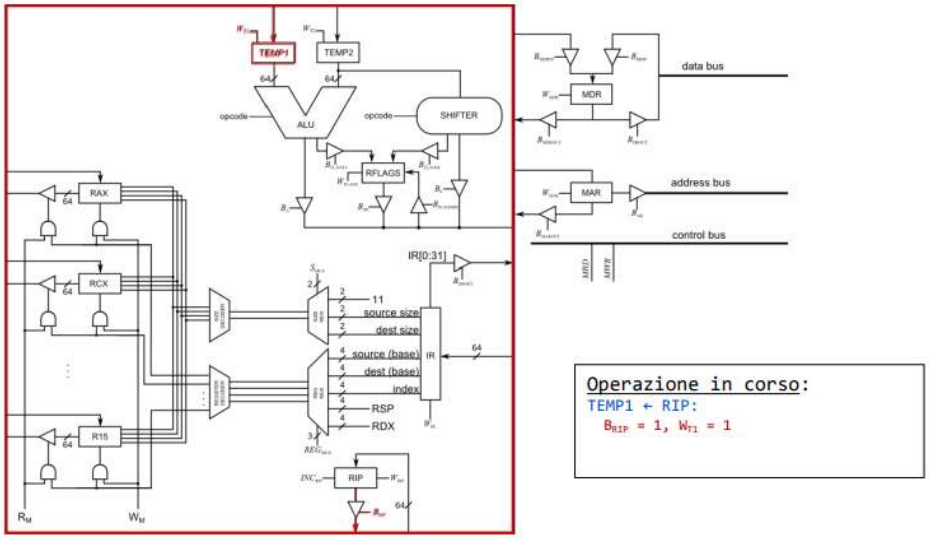
d.



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

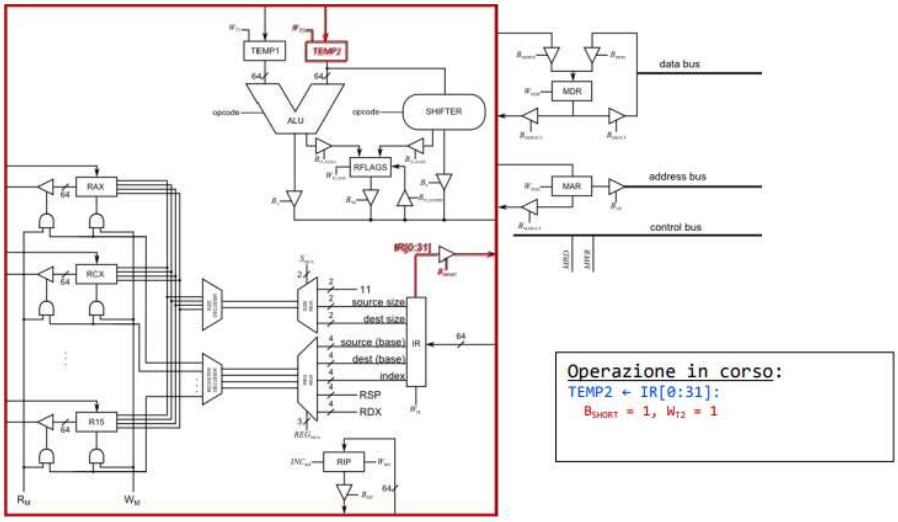
3. `jz displacement:`
   - `MAR ← RIP`
   - `MDR ← (MAR); RIP ← RIP + 8`
   - `IR ← MDR`
   - `IF FLAGS[ZF] == 1 THEN`
   - `    TEMP1 ← RIP`
   - `    TEMP2 ← IR[0:31]`
   - `    RIP ← ALU OUT[ADD]`
   - `ENDIF`

a.



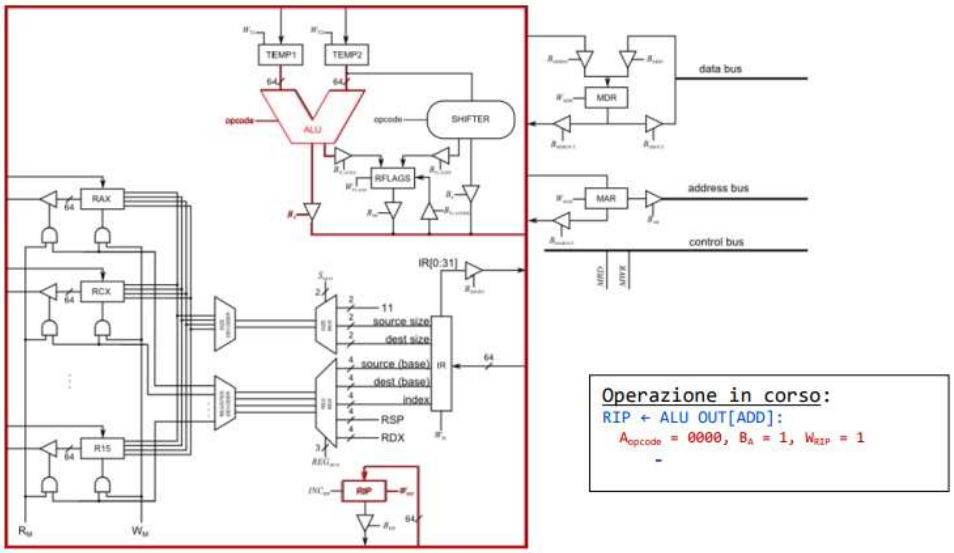
b. Abilito il buffer-tri-state di RIP per scrivere sull'internal bus data, poi i dati arrivano a TEMP1 ed abilito la scrittura s questo registro : (Wtemp1=1)

c.



d. Prendo i 32 bit dello spiazzamento da IR[0:31], abilito il buffer-tri-state per scrivere sull'internal data bus , e lo scrivo su TEMP2 ed abilito il segnale di scrittura (Wtemp2=1).

e.



f. Infine alla ALU dando il codice della somma (Aluop=0000), abilito il buffer-tri-state(Ba=1) per scrivere sull'internal bus data , lo vado a scrivere nel RIP ed abilito la scrittura nel RIP(Wrip=1).

Notiamo che si deve essere sicuri di aggiornare il registro flags se e solo se si hanno istruzioni aritmetiche.