UNIVERSITA' DEGLI STUDI DI REGGIO CALABRIA
FACOLTA' DI INGEGNERIA ELETTRONICA

ANNO ACCADEMICO 99-2000

CORSO DI
CALCOLATORI ELETTRONICI

PROGETTO DI UNA COPPIA DI MICROPROCESSORI

STUDENTI :
AZZARA' BENIAMINO
GUARNACCIA GIUSEPPE
MINNITI PIETRO

Presentazione del progetto

Le specifiche previste dalla traccia relative all'ideazione dei microprocessori richiesti consentivano una certa flessibilità su alcune scelte di progetto: si è cercato di trovare la soluzione che si presentasse più semplice e leggibile possibile, cercando di conciliare questo con la funzionalità e l'efficienza. Non sempre ciò è stato possibile e non sempre si è certi di aver trovato la soluzione più efficace. Si è avuta spesso la tendenza a orientarsi su soluzioni che potessero realizzarsi a livello circuitale e non allungare il programma ROM del microprocessore principale, ma questo ha talvolta ridotto la leggibilità. I commenti sono stati introdotti in modo da poter analizzare anche singole parti del progetto, dando a questo, per quanto possibile, la massima modularità.

Talvolta le scelte di progetto si presentano fortemente condizionate da ragioni di carattere economico: ad esempio quando si deve decidere se guadagnare un colpo di clock introducendo un registro in una data funzione o risparmiare il registro; oppure, laddove possibile, se utilizzare un bus a più ingressi o una sola porta nel bus preceduta da un selettore di ingressi. In generale non si è scesi molto in dettaglio in queste analisi temendo di non avere tutti gli elementi necessari per prendere la decisione corretta.

La pagina che segue presenta lo schema generale dove le frecce con un'unica linea indicano il flusso di segnali di controllo, mentre quelle spesse descrivono un flusso di dati o indirizzi.

Tutti i collegamenti tra le parti autonome dello schema generale sono stati qui indicati solo schematicamente con i due tipi di frecce: quelle a linea singola ($\longrightarrow$) indicano un flusso di segnali di controllo, mentre le altre ($\Longrightarrow$) indicano un flusso dati e indirizzi.

Nei disegni che seguono tutte le connessioni tra componenti diverse (M, S, DMA, I/O, RAM, Arbitro) sono evidenziate in modo da poter considerare le singole componenti come scatole chiuse.

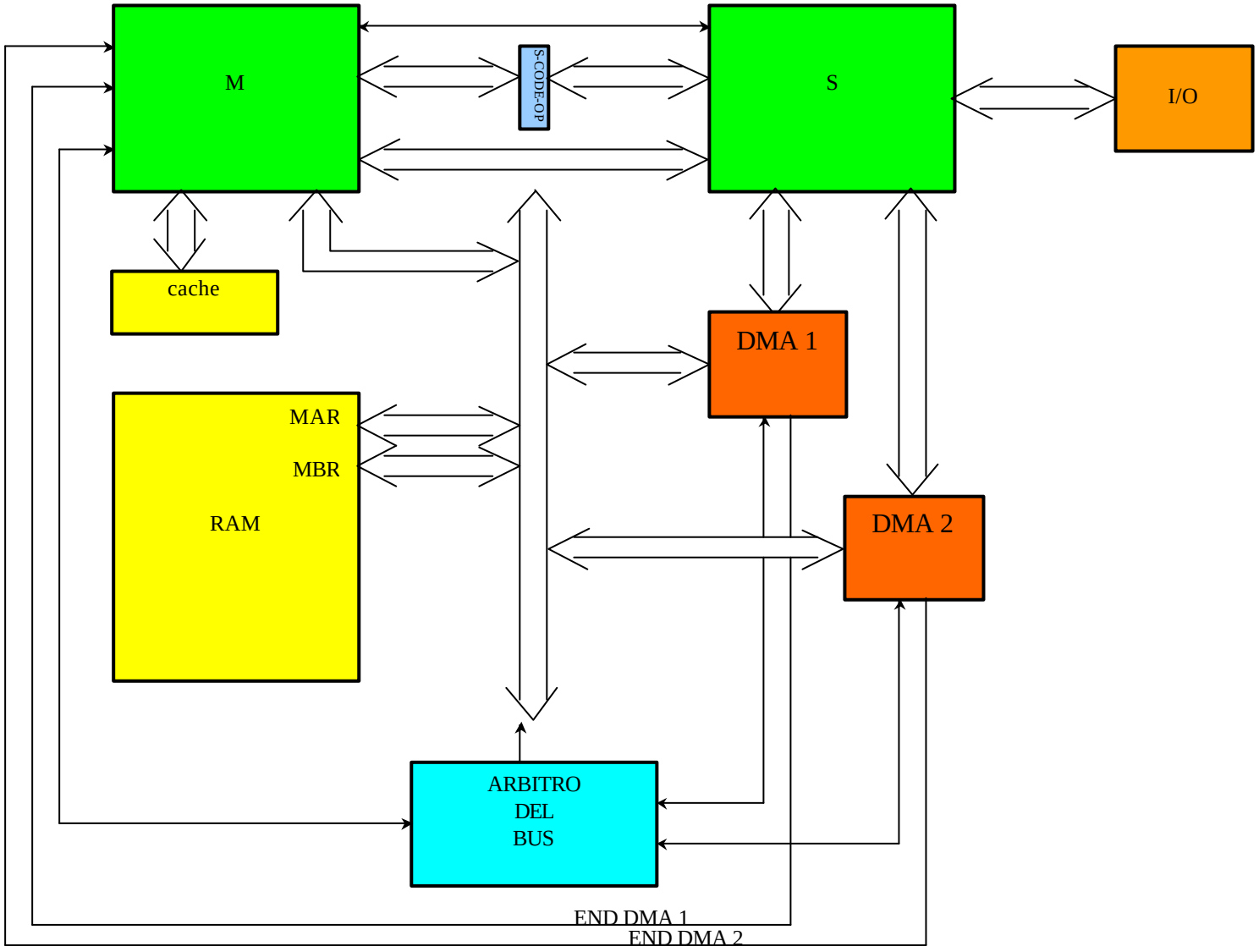
Non sono state invece trattate le parti di controllo.

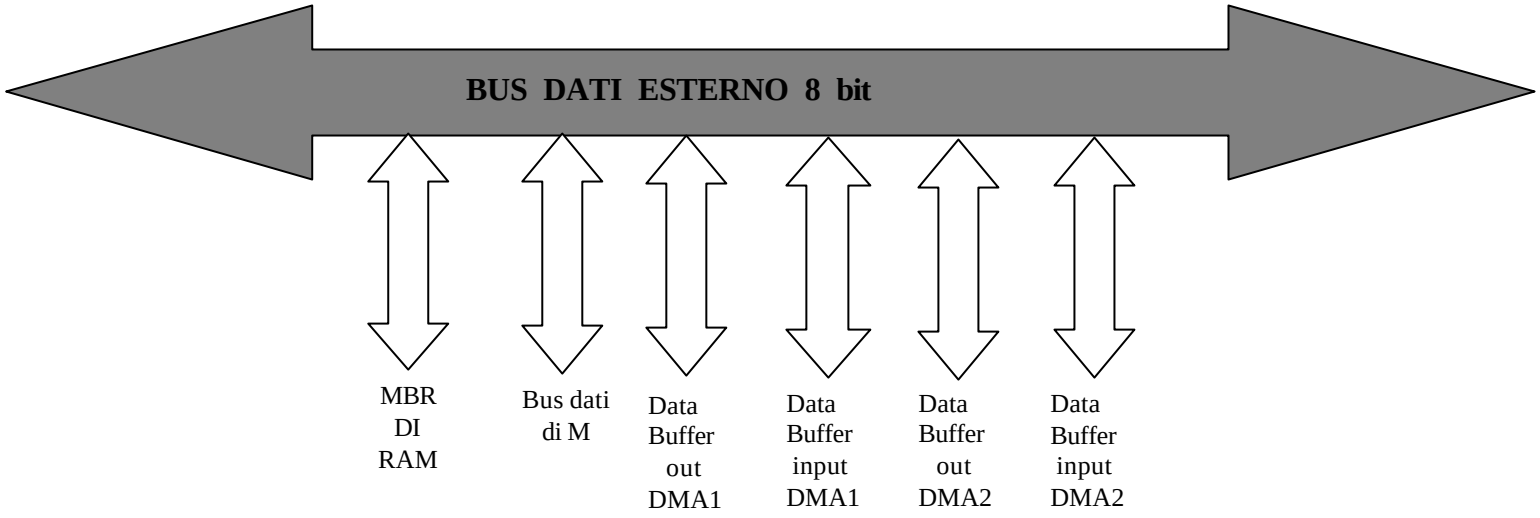
Tutto l' RTL fa riferimento alle parti operative allegate al progetto ed è stato obiettivo fondamentale del lavoro realizzare la totale compatibilità delle μ-istruzioni con le parti operative. Così anche il caricamento iniziale dei registri trova riferimento negli stessi disegni.

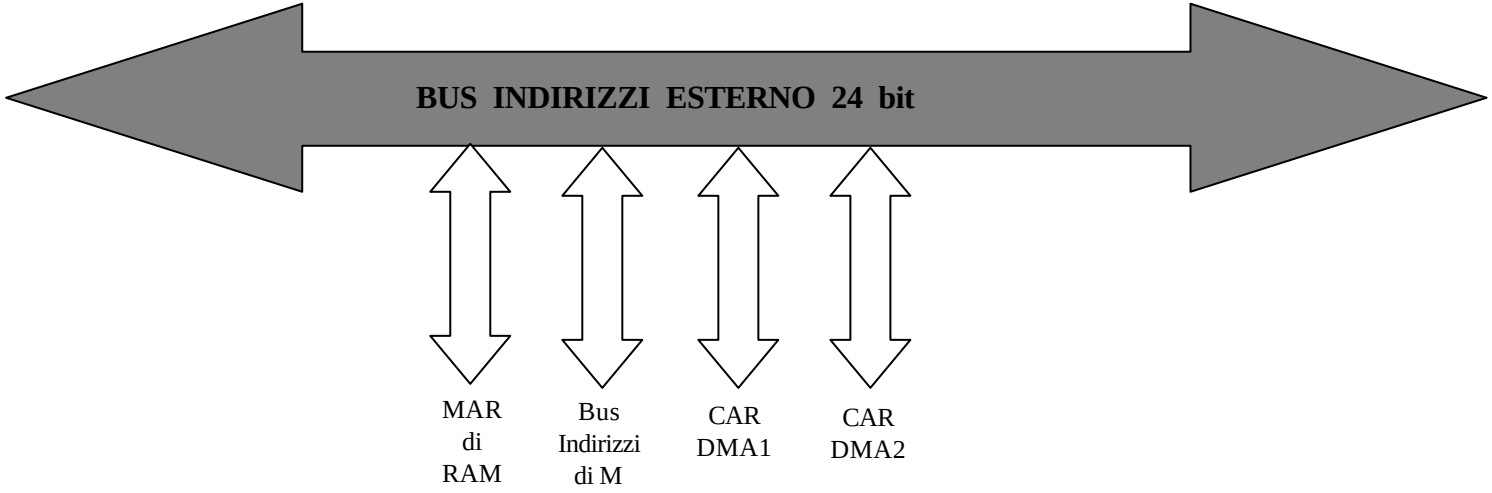
In tutte le connessioni tra registri si specifica la dimensione dei bus e da quali bit dei registri provengano qualora non siano connessi a tutto il registro. Quando da un registro escono più bus, non connessi da un selettore, si intenda il collegamento realizzato "filo per filo".

I flip-flop dei registri vengono sempre numerati da sinistra verso destra e partendo da 0.

SCHEMA GENERALE

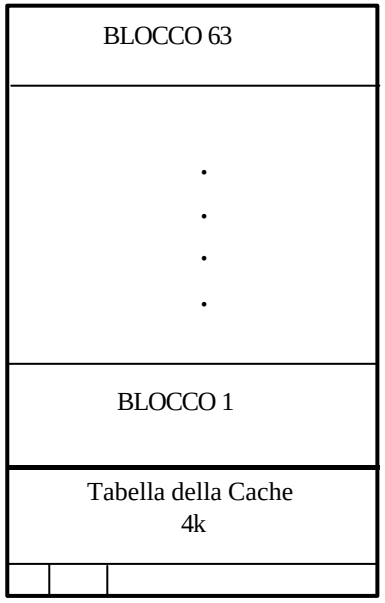



BUS DATI ESTERNO 8 bit
MBR DI RAM
Bus dati di M
Data Buffer out DMA1
Data Buffer input DMA1
Data Buffer out DMA2
Data Buffer input DMA2

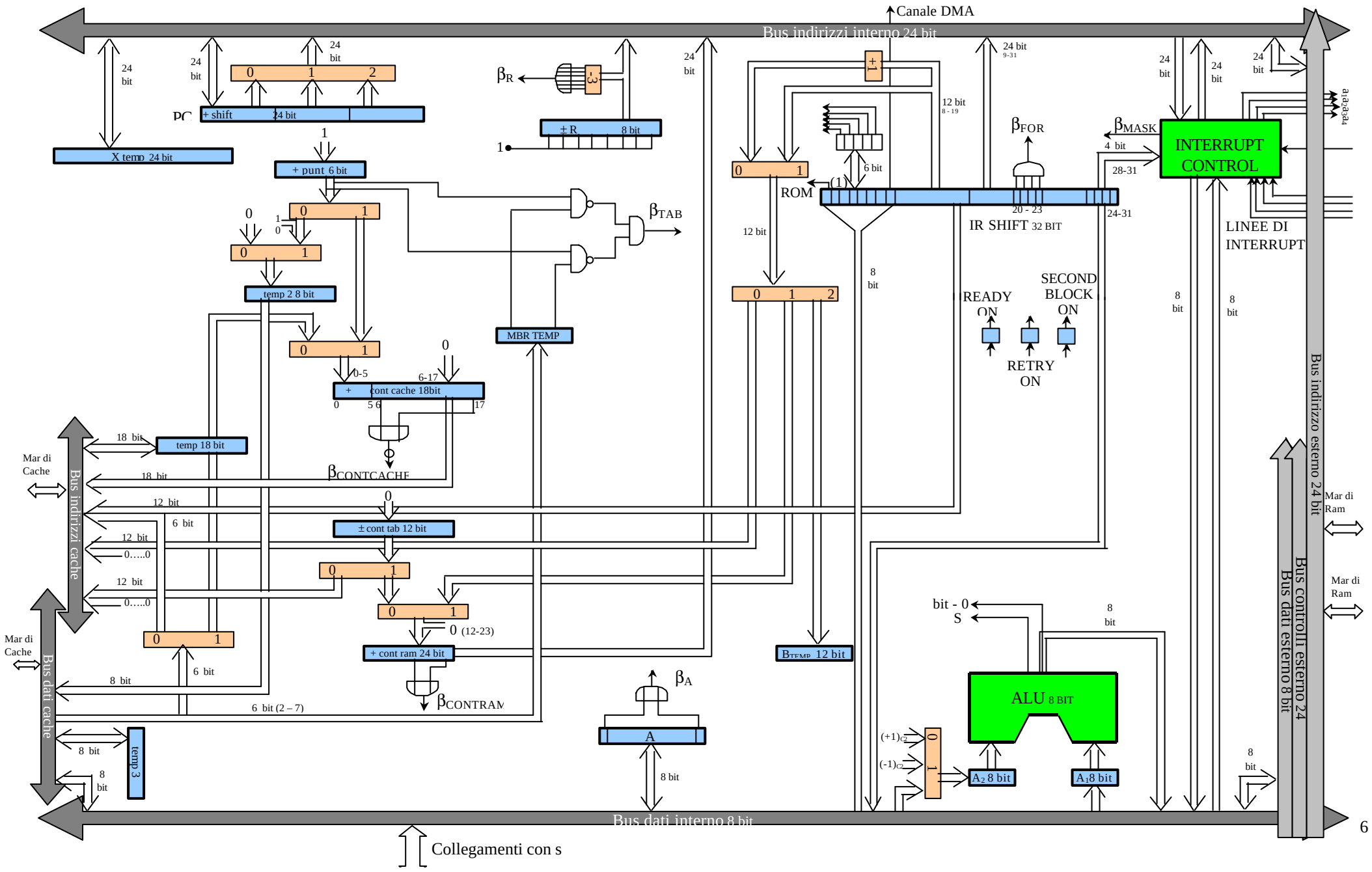

BUS INDIRIZZI ESTERNO 24 bit
MAR di RAM
Bus Indirizzi di M
CAR DMA1
CAR DMA2

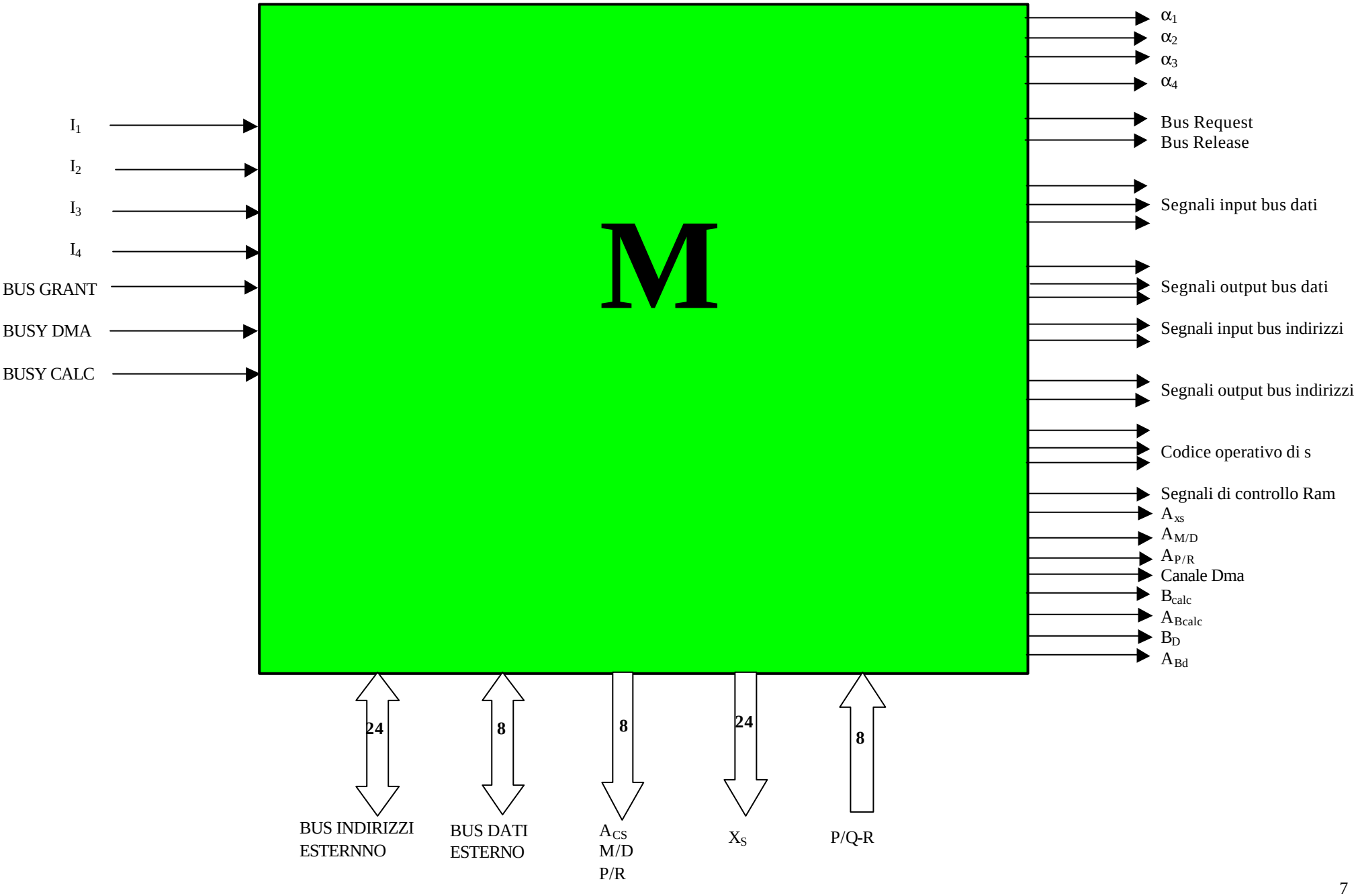
MECCANISMO DI TRASFERIMENTO RAM -CACHE

La cache è partizionata in 63 blocchi di 4K piu' un ulteriore blocco sempre da 4K (4096 righe da 8 bit) che viene utilizzato per rappresentare la tabella .
Si e' scelto la dimensione delle celle a 8 bit perche' si suppone di trattare varie tipologie di dati (a 8 , 16 e 32 bit) .Dalla Ram che risulta essere così partizionata in 4096 blocchi da 4K si trasferiscono così ogni volta blocchi da 4 K .
Ogni rigo della tabella ha 8 bit così suddivisi il primo bit segnala se il blocco è presente in cache , il secondo bit segnala se il blocco è stato modificato
gli altri sei bit danno la posizione del blocco all'interno della cache . Un contatore punt (il cui circuito è stato realizzato ed ingrandito su un altro foglio)indica costantemente il blocco piu' vecchio che dovra' essere trasferito in Ram qualora il blocco sia stato modificato, oppure semplicemente sovrascritto dal nuovo blocco che stiamo inserendo, i blocchi vengono cancellati secondo una politica di tipo F.I.F.O.
La politica utilizzata nella cache è quella WRITE-BACK (questo appesantisce notevolmente i codici della Fetch della Forward e della Retry) .
Il trasferimento tra cache e Ram avviene solo per determinate operazioni che vengono individuate dal segnale β_{or} che viene prodotto tramite una rete combinatoria, costruita nella parte operativa di M (il cui circuito e' stato ingrandito su un altro foglio) . Come prima cosa si cerca se il blocco interessato e' presente in cache : se il blocco e' presente allora non si ha nessun trasferimento ed i dati vengono prelevati dal blocco presente.
Se invece il blocco cercato non e' presente in cache , tramite il contatore punt ,si seleziona il blocco piu' vecchio : se tale blocco e' modificato si trasferisce in Ram e quindi al suo posto dalla Ram verra' trasferito il nuovo blocco , se no il nuovo blocco sovrascriverà semplicemente il blocco piu' vecchio , in entrambi i casi si ha sempre l'aggiornamento della tabella .Per conoscere in quale blocco della Ram e' collocato un dato di indirizzo X e' quindi sufficiente leggere i 12 bit piu' significativi di tale indirizzo . Per rintracciare lo stesso dato eventualmente presente in cache si legge la riga di tabella relativa al blocco della Ram (di indirizzo $X_{0\text{-}11}$) ; si controlla il campo modificato di tale cella e se questo e' 1 si legge nei restanti sei bit della cella la posizione del blocco in cache (da 1 a 63) . Tali sei bit forniscono a questo punto i sei bit piu' significativi dell'indirizzo del dato in Cache .
Essendo, ora, i blocchi in cache un immagine speculare dei blocchi in Ram , l'indirizzo completo del dato in cache si ottiene aggiungendo a quei sei bit più significativi gli stessi dodici bit meno significativi dell'indirizzo . Il dettaglio dei passi su come avviene il trasferimento Ram – Cache è trattato all'interno della fetch



PARTE OPERATIVA DI M



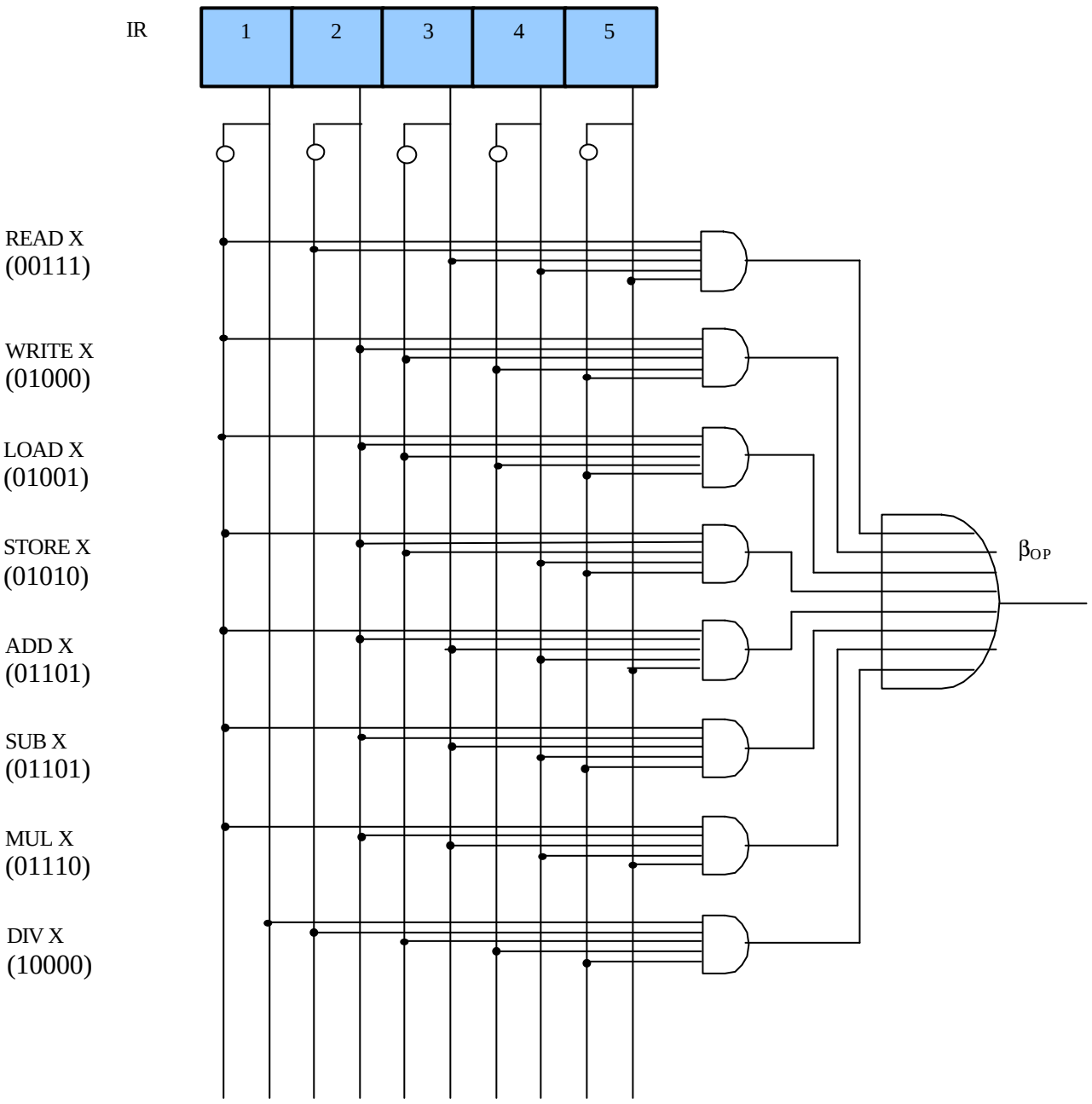

M
I_1
I_2
I_3
I_4
BUS GRANT
BUSY DMA
BUSY CALC
α_1
α_2
α_3
α_4
Bus Request
Bus Release
Segnali input bus dati
Segnali output bus dati
Segnali input bus indirizzi
Segnali output bus indirizzi
Codice operativo di s
Segnali di controllo Ram
A_{xs}
$A_{M/D}$
$A_{P/R}$
Canale Dma
B_{calc}
A_{Bcalc}
B_D
A_{Bd}
24
8
8
24
8
BUS INDIRIZZI ESTERNNO
BUS DATI ESTERNO
A_{CS} M/D P/R
X_S
P/Q-R

(1)VEDI IR PARTE OPERATIVA DI M

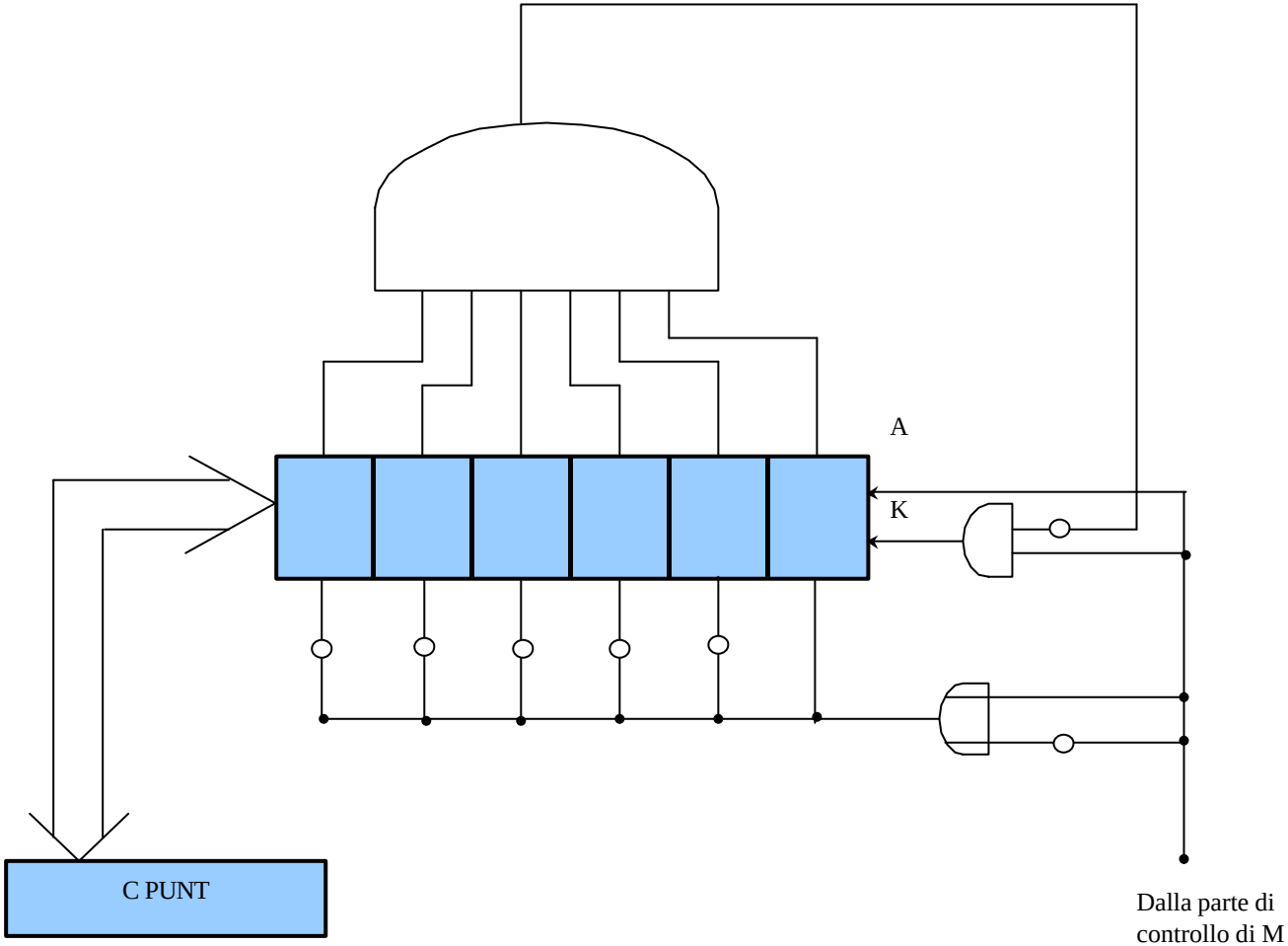
SEGNALE

b_{OP}

QUESTO SEGNALE INDIVIDUA LE OPERAZIONI CHE NECESSITANO DI UN TRASFERIMENTO RAM-CACHE



CONTATORE CPUNT DELLA CACHE



Circuito di gestione degli interrupt

(I due fogli consecutivi costituiscono un unico blocco. La descrizione dei circuiti segue un percorso dal basso verso l'alto.)

Gli interrupt provenienti da S e dai canali DMA arrivano ad M dalle 4 linee $i_1,\ldots,i_4$, abilitano la scrittura dei registri (porta or prima della A dei flip-flop) e, essendo A_k, come vedremo, quasi sempre 0 eccetto un solo colpo di clock (di M, che è il clock più veloce del sistema) per ogni μ-sequenza, memorizzano 1 nel corrispondente flip-flop I_J (la porta and prima della F dei flip-flop lascia passare il valore che proviene dalla linea i_j). A questo punto la richiesta è stata inoltrata e resterà memorizzata nel flip-flop fin quando non verrà servita e cancellata. Non è quindi possibile che una richiesta di interrupt non possa venire accolta.

I quattro flip-flop M_j hanno le A di abilitazione controllate dalla parte di controllo di M con un unico segnale. Le F dei registri di tali flip-flop, invece, sono collegate ai 4 bit meno significativi dell' IR.

Dalle porte and che seguono tali flip-flop uscirà quindi 1 su tutte le richieste di interrupt non mascherate dal programmatore.

A questo punto è bene sottolineare che si è fatta l'assunzione che un interrupt di una data priorità possa interrompere l'esecuzione della routine di servizio di un interrupt di pari priorità. Questo complica leggermente lo schema circuitale perché impone che si controlli che il register pool, dove si salva lo stato di M, sia stato totalmente riempito (pila piena).

Un interruzione va quindi servita se:

1- non si sta servendo alcuna interruzione e si ha una richiesta;
2- si sta servendo un interruzione di priorità inferiore o uguale alla priorità della richiesta in arrivo.

Alla fine di ogni μ-sequenza (μ-istruzioni evidenziate in blu nell'RTL) il programma interroga il flip-flop B_{int} per verificare se c'è un interruzione da servire. Se è presente una interruzione da servire, M deve:

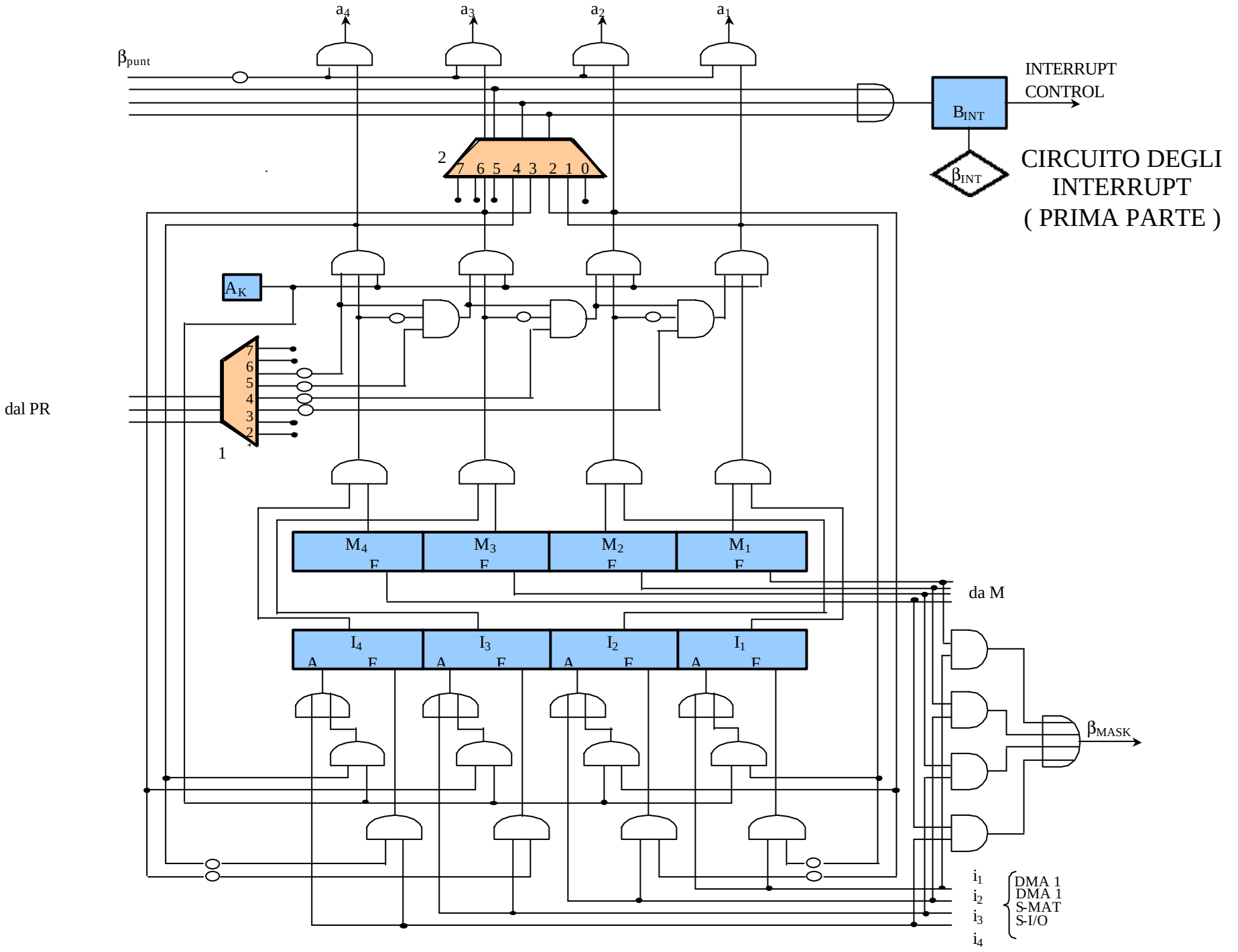
1- informare la periferiche che ha inviato l'interruzione dell'avvenuto servizio;
2- caricare la nuova priorità nel registro PR;
3- resettare il flip-flop della corrispondente interruzione;
4- eseguire alcune operazioni di scrittura in Ram che possano informare il programmatore sull'interruzione accolta.

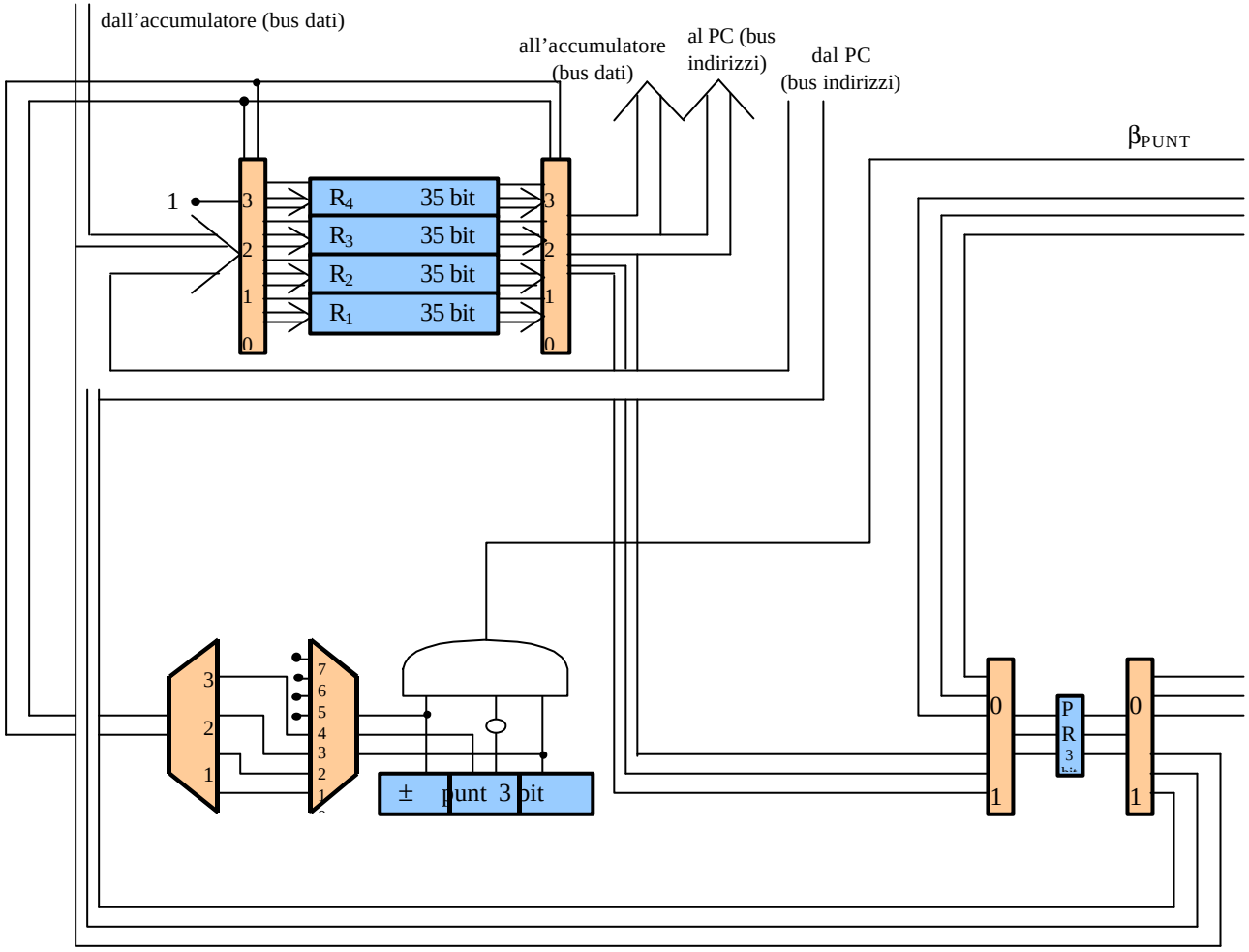
Vediamo come avviene la lettura dell'eventuale presenza e le conseguenti sopraelencate operazioni.

Il registro PR indica la priorità dell'interruzione in corso: dal coder verrà illuminata la linea n che bloccherà, con le porte and a cascata, all'ingresso degli and controllati da A_k, le richieste di priorità j con j<n.

All'arrivo del segnale che mette a 1 il flip-flop A_k, dalle porte and da esso controllate verrà illuminata la sola linea corrispondente all'interruzione da servire; tale linea verrà inoltre utilizzata per informare la periferica dell'avvenuto servizio (segnale a_n,)(il segnale β_{punt} è 0 se la pila non è piena) e la parte di controllo di M, al fine di consentire il caricamento della Ram. Dal decoder uscirà un numero binario diverso da zero e il flip-flop B_{nt} potrà essere caricato. Dovrà essere inoltre azzerato il flip-flop I_n e questo avviene semplicemente abilitandone la lettura (sarà infatti, grazie alle porte and a monte delle A, posta a 1 solo la A del flip-flop I_n) . Il nuovo PR (uscita del decoder) può essere caricato abilitando il canale 0 nel corrispondente selettore.

Quando un'interruzione termina di essere servita, con l'istruzione RETI si ripristina lo stato precedente: il PR, se stavamo servendo un'interruzione senza altre in coda, viene ripristinato a zero.


a4
a3
a2
a1
βpunt
INTERRUPT
CONTROL
BINT
βINT
2
7 6 5 4 3 2 1 0
CIRCUITO DEGLI
INTERRUPT
(PRIMA PARTE)
AK
dal PR
7
6
5
4
3
2
1
1
M4 F
M3 F
M2 F
M1 F
da M
I4 A F
I3 A F
I2 A F
I1 A F
βMASK
i1
i2
i3
i4
DMA 1
DMA 1
S-MAT
S-I/O

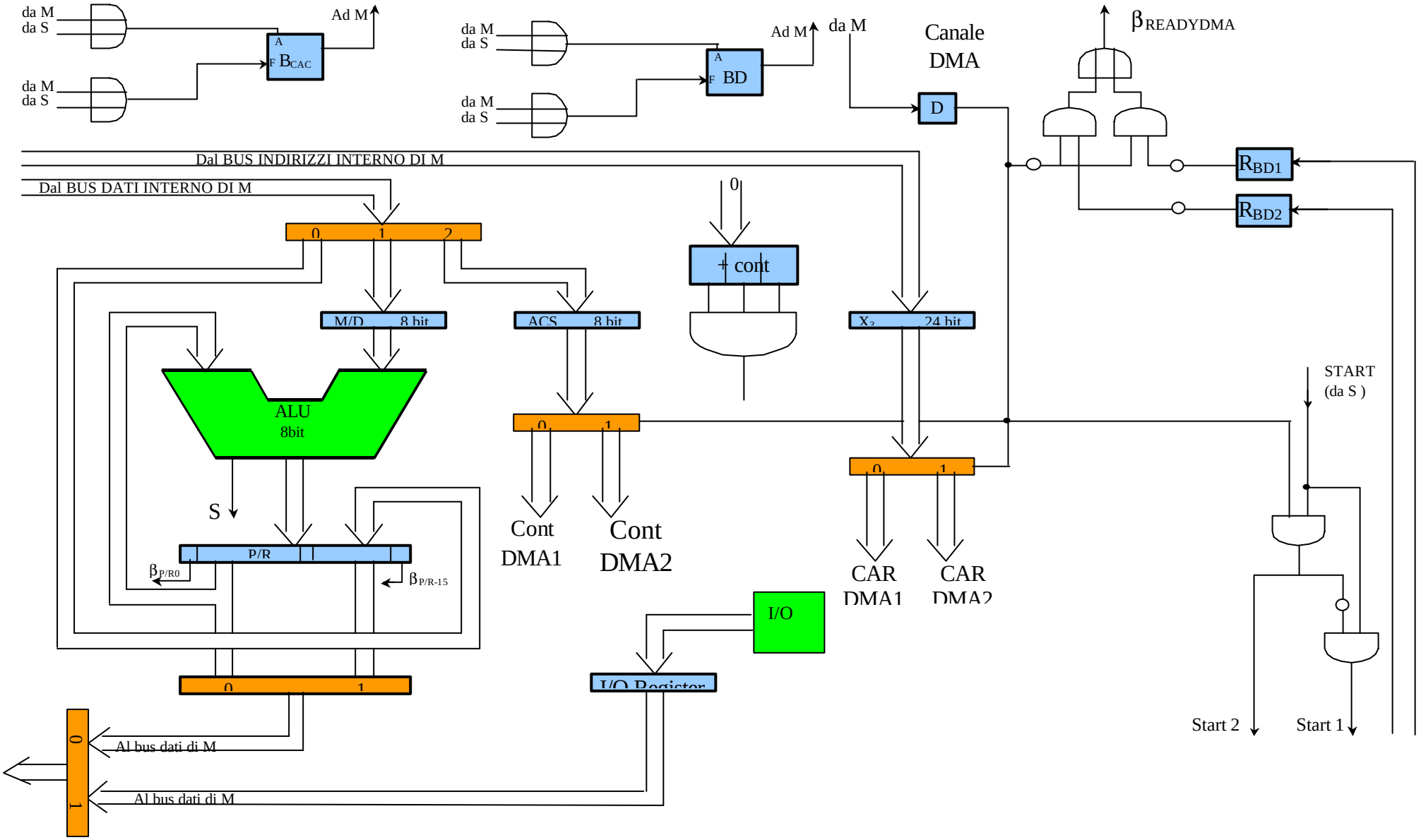


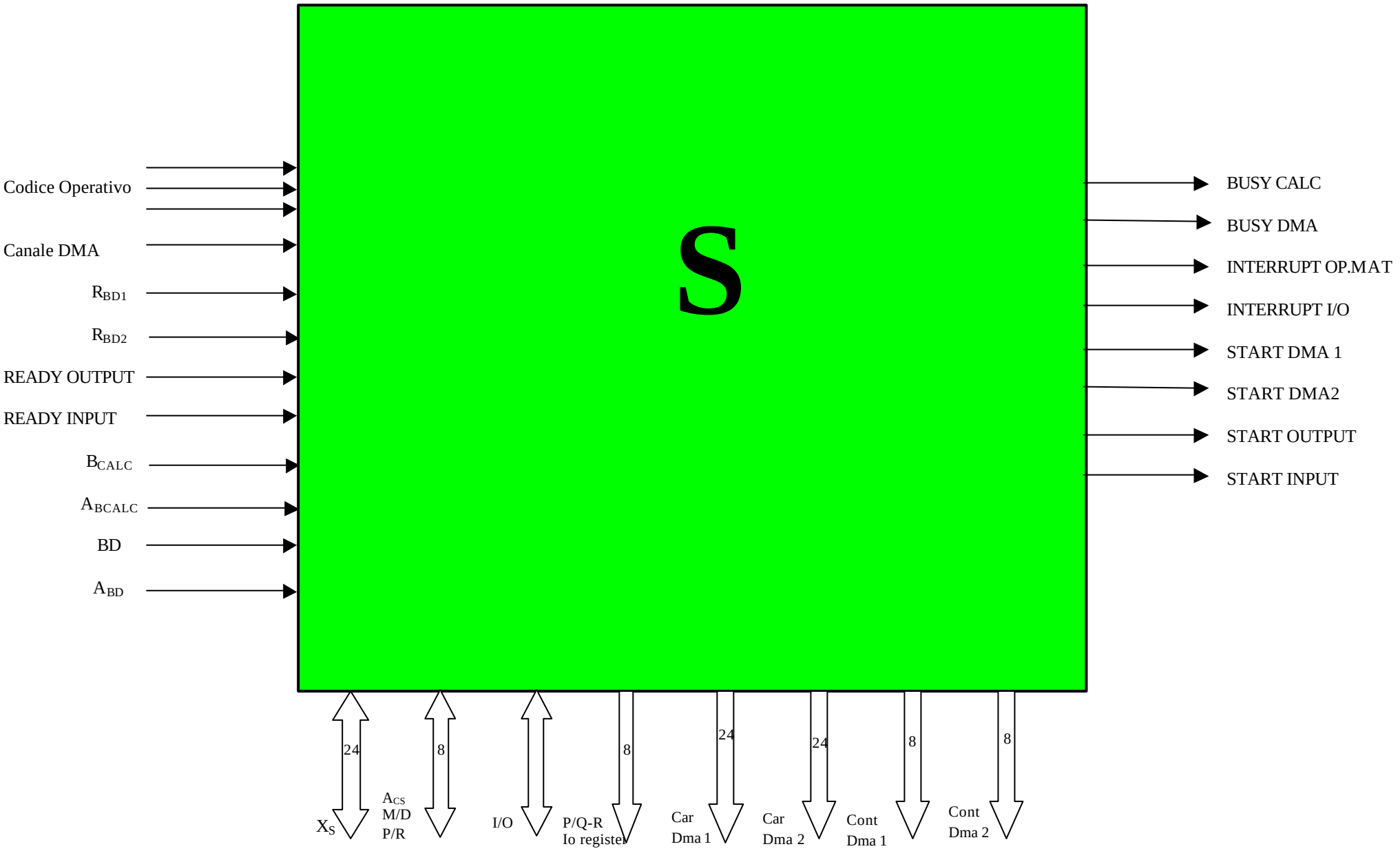
CIRCUITO DEGLI INTERRUPT
(SECONDA PARTE)

PROTOCOLLO DI SCAMBIO M-S

Il processore M richiede le operazioni ad S (MUL-DIV-READ-WRITE- FORWARD – RETRY) inserendo nel registroS-CODE-OP. i tre bit che costituiscono il codice istruzione di S e che sono gli input del processore S, e ponendo ad 1 il relativo flip-flop (BD o B_{calc}) in modo da disabilitare eventuali ulteriori richieste di M , tali flip flop verranno resettati al termine di un calcolo matematico (B_{calc}) o una volta attivato il DMA (BD)
Il codice è stato scritto in maniera tale da consentire ad M il maggior numero di richieste possibili; infatti, come si può verificare, S può accogliere una richiesta di trasferimento anche se il DMA sta lavorando; può accogliere e servire richieste sul canale libero anche se l'altro canale è occupato (naturalmente il canale resta quello specificato da M, tramite il flip flop D che si trova nella parte operativa di S e che viene settato da M), e inoltre può mettere in coda una richiesta anche se entrambi i canali sono occupati .
Qualora S non fosse pero' in condizione di accogliere ulteriori richieste il processore M si porrebbe in busy-waiting .
Terminata l'operazione matematica S spedisce l'interrupt ad M che e' in grado di leggere e scrivere sui registri di S. La lettura del risultato e' inoltre gestibile dal programmatore attraverso l'istruzione WAIT che lo informa della fine del calcolo .Quando M richiede un operazione di I/O ad S , questi inoltra la richiesta alla periferiche ed entra in busy-waiting; ad operazione terminata, S, invia una linea di interrupt ad M che lo informa della fine dell'operazione a questo punto analogamente a quanto avviene per il calcolo M potrà andare a leggere sul registro di S (I/O register).
A questo punto S può ripristinare il flip-flop BD a 0 per accogliere ulteriori richieste.

PARTE OPERATIVA DI S

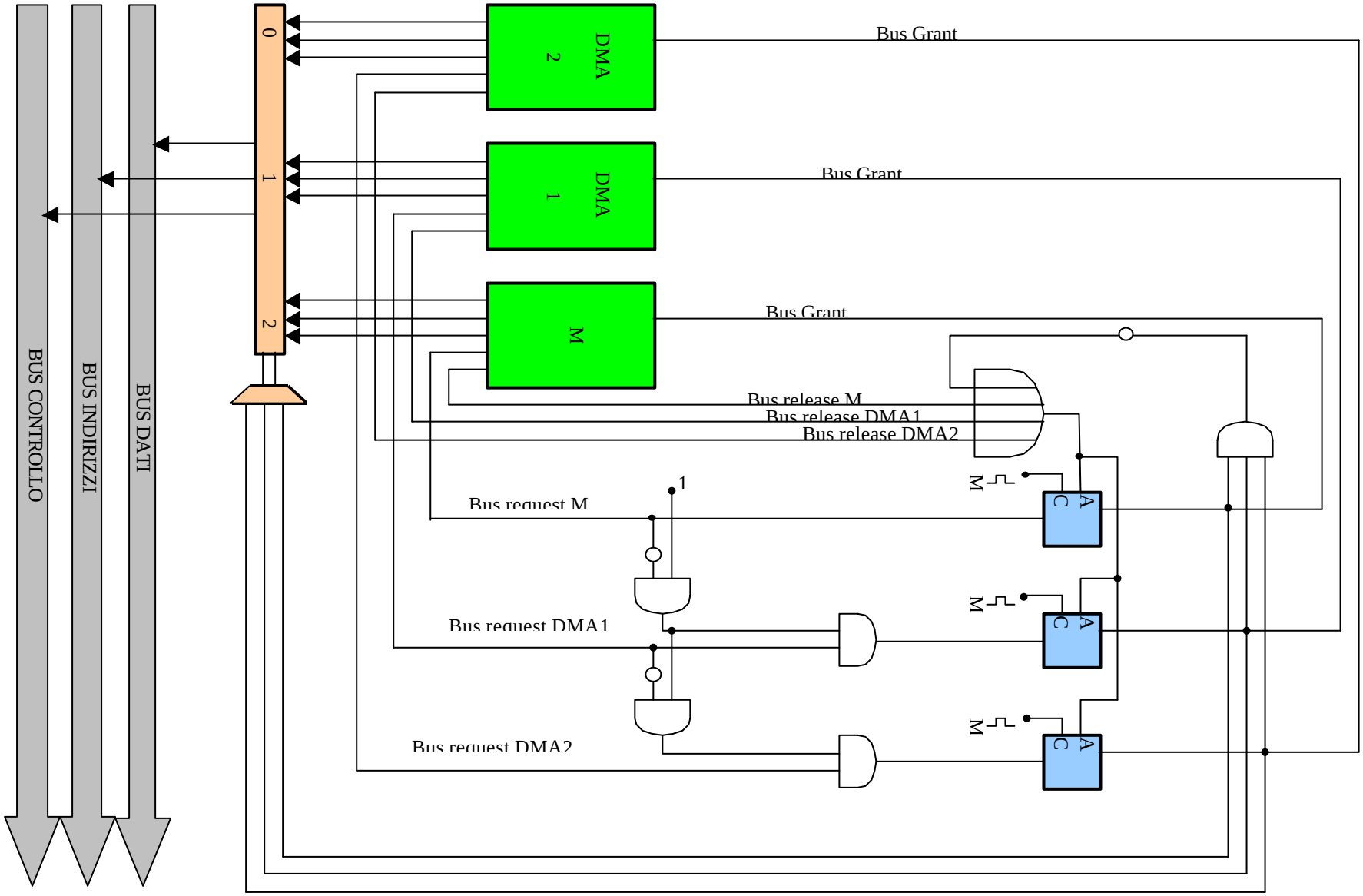


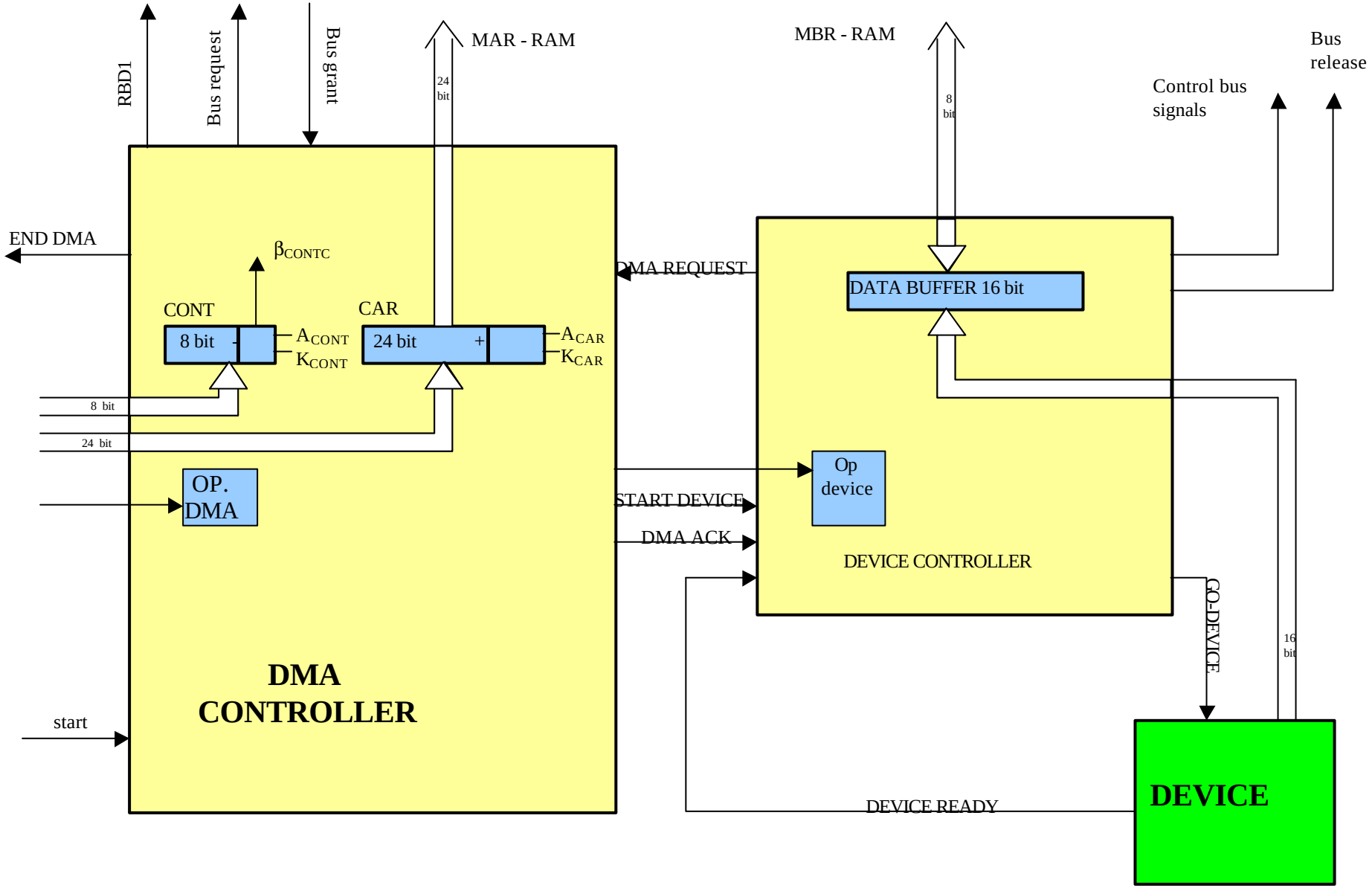

S
Codice Operativo
Canale DMA
R_{BD1}
R_{BD2}
READY OUTPUT
READY INPUT
B_{CALC}
A_{BCALC}
BD
A_{BD}
BUSY CALC
BUSY DMA
INTERRUPT OP.MAT
INTERRUPT I/O
START DMA 1
START DMA2
START OUTPUT
START INPUT
24
8
I/O
8
24
24
8
8
X_S
A_{CS}
M/D
P/R
P/Q-R
Io register
Car
Dma 1
Car
Dma 2
Cont
Dma 1
Cont
Dma 2

Arbitro del Bus

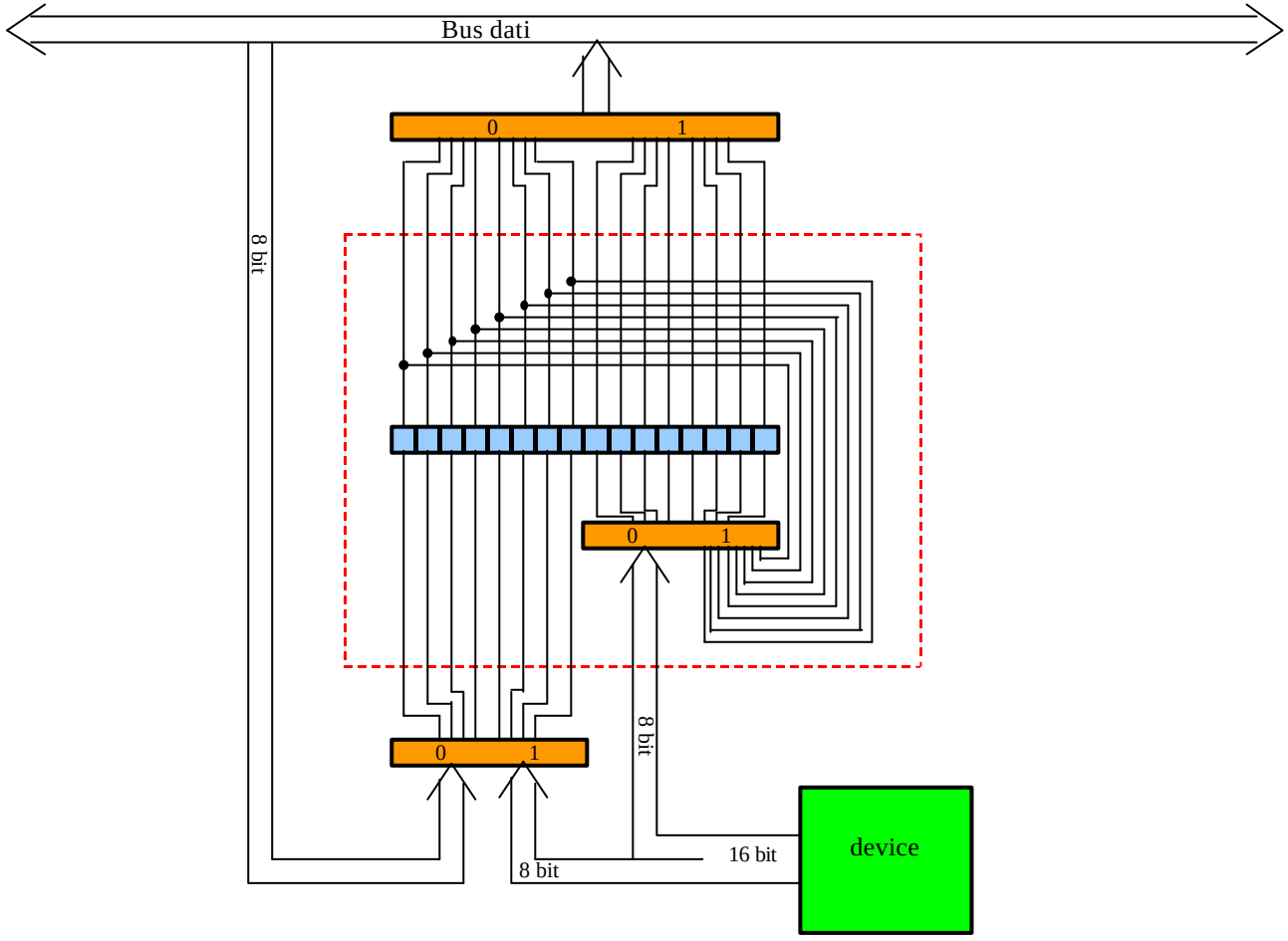
Le tre unita' che fanno richiesta all'arbitro, mantengono ad uno il segnale di bus-request quando inoltrano la richiesta. Se il bus è occupato le A di abilitazione dei flip-flop sincronizzati con il clock più veloce (quello di M) , sono poste a zero .
A monte di tali flip – flop verrà comunque presentata solo la richiesta a maggiore precedenza piu' alta , quando l'unità che ha il bus in uso lo rilascia attiva, per la durata di un colpo di clock , ad 1 il segnale Bus Release che , abilitando i flip-flop, passa il controllo all'unità in attesa che trova ora il segnale Bus Grant ,. contemporaneamente il coder che controlla il selettore abilita il flusso dei segnali di controllo dell'unità autorizzata.
Qualora il bus all'arrivo di una richiesta non sia in uso da alcuna macchina i flip-flop sono comunque abilitati in lettura .

ARBITRO DEL BUS



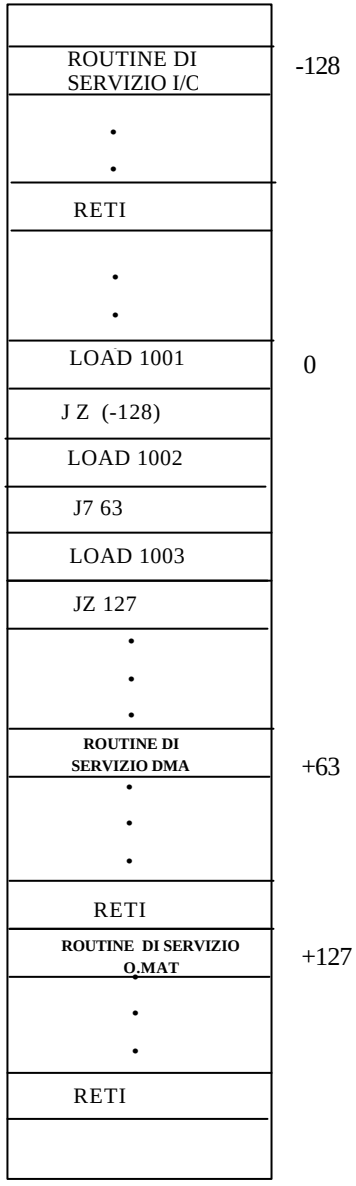

RBD1
Bus request
Bus grant
MAR - RAM
24 bit
MBR - RAM
8 bit
Control bus signals
Bus release
END DMA
βCONTC
CONT
8 bit
ACONT
KCONT
CAR
24 bit
ACAR
KCAR
DMA REQUEST
DATA BUFFER 16 bit
8 bit
24 bit
OP. DMA
Op device
START DEVICE
DMA ACK
DEVICE CONTROLLER
GO-DEVICE
16 bit
DMA CONTROLLER
start
DEVICE READY
DEVICE

REGISTRO DATA BUFFER DEL DMA



ROUTINE DI SERVIZIO

Si assume che il programmatore conosca l'indirizzo di memoria (ad es. 1001) a partire dal quale, in tre celle, la fetch, nell'ambito della gestione degli interrupt, scrive 0 se va servito l'interrupt corrispondente a quella classe.
Ad esempio, nel caso in cui si deve servire un interrupt di classe 2, vi sarà scritto 0 nella cella di indirizzo 1002 e 1 nelle restanti due celle.
La routine di servizio generale può, invece, essere collocata dal programmatore in un area di memoria a sua scelta, a patto che egli ponga in 0 l'indirizzo di partenza di tale routine.
A questo punto, una possibile routine che rimandi alle varie specifiche gestioni può essere quella scritta di seguito (nella tabella). Vengono effettuate tre load successive seguite dalla jz che simulano l'istruzione if non presente nel linguaggio che offriamo al programmatore. Questi può ora scrivere le routine nelle celle a cui fa riferimento la jz.
Una specifica gestione nel nostro caso , ade esempio a fine trasferimento I/O , a seguito di una READ si ridurrebbe alla sola istruzione STORE X sempre ammesso che la fetch abbia preventivamente caricato l'accumulatore e l'IR , il primo attraverso l'I/O register , il secondo dal registro Xtemp.



FETCH (codice operazione 00000)

{Il primo passo della fetch è controllare se ci sono interruzioni: questo è possibile grazie al segnale che arriva dal flip-flop di controllo che è stato preventivamente caricato dalla μ-sequenza precedente. (vedi commenti al circuito degli interrupt)}

If interrupt control=1 then

{ Si e' avuta un interruzione, a questo punto prima di salvare lo stato, verifichiamo se la pila e' piena }

If (not ($ß_{punt}$)) = 1 then { la pila non e' piena:possiamo quindi salvare PC ,A e PR e caricare su PR la nuova priorita' }

PC → $R_{punt\ (0\text{-}23)}$, PR → $R_{punt\ (24\text{-}26)}$, A → $R_{punt\ (27\text{-}34)}$,PUNT + 1 → PUNT , bo,b1,b2 → PR;

{bisogna inoltre caricare 3 specifiche celle della Ram indicando quale delle classi di interrupt stiamo servendo per consentire al programmatore assembler di far partire l' appropriata routine di servizio , (vedi descrizione della routine di servizio) questa porzione di codice qui viene omessa.}

1 → BUSREQUEST; { facciamo richiesta all'arbitro per avere accesso al bus }

0: if BUS GRANT = O then goto 0

else

{ carichiamo in PC l'indirizzo presente nella locazione 0. L'assunzione è che tale indirizzo, scritto dal programmatore, si mantiene tale per tutta l'esecuzione del programma }

{Per leggere nelle prime tre celle della RAM (l'indirizzo è a 24 bit), è necessario un registro a incremento e possiamo utilizzare $CONT_{RAM}$ già presente. Questo, però, volendo lasciare inalterata la parte operativa di M, comporta lo spreco di un colpo di clock per il caricamento che deve passare attraverso $CONT_{TAB}$: spetta alle valutazioni di carattere economico la scelta di introdurre o meno un selettore per velocizzare la gestione degli interrupt.

0 → $CONT_{TAB}$;

0 → $CONT_{RAM}$;

$CONT_{RAM}$ → MAR , $CONT_{RAM}$ + 1 → $CONT_{RAM}$;

M [MAR] → MBR , $CONT_{RAM}$ → MAR , $CONT_{RAM}$ + 1 → $CONT_{RAM}$; } caricamento del PC

{Il registro a shift PC viene caricato in tre tempi}

MBR → $PC_{0\text{-}7}$, $CONT_{RAM}$ → MAR , M [MAR] → MBR ;

MBR $\xrightarrow{SHIFT}$ $PC_{0\text{-}7}$, M [MAR] → MBR ;

MBR $\xrightarrow{SHIFT}$ $PC_{0\text{-}7}$,

A questo punto il controllo passa alla fetch successiva che come sempre esaminerà il flip -flop Bint che va quindi resettato. Questo è possibile semplicemente abilitando la scrittura dato che A_k del circuito degli interrupt è posto a zero.}

1→ A_{Bint} ;

{ rilasciamo il bus }

0 → BUS REQUEST;

1 → BUS RELEASED

fi

else {la pila è piena: non possiamo accogliere altre interruzioni, bisognerà prima servire quelle già presenti nella pila. A questo punto il controllo passa alla fetch successiva che come sempre esaminerà il flip -flop Bint che va quindi resettato. Questo è possibile semplicemente abilitando la scrittura dato che A_k del circuito degli interrupt è posto a zero.}

1→ A_{Bint} ;

else {non sono arrivati interrupt e si procede al caricamento dell'IR}

1 → BUSREQUEST; { facciamo richiesta all'arbitro per avere accesso al bus }

1: if BUS GRANT = O then goto 1
else
PC → MAR, PC+1→ PC;
M[MAR] → MBR, PC+1 → PC, PC → MAR;
MBR → $IR_{TEMP\ 0-7}$,PC→ MAR, PC+1 → PC, M[MAR] → MBR;
MBR → $IR_{TEMP\ 0-7}$, PC→ MAR, PC+1 → PC, M[MAR] → MBR;
MBR → $IR_{TEMP\ 0-7}$, M[MAR] → MBR;
MBR → $IR_{TEMP\ 0-7}$;
{ rilasciamo il bus }
0 → BUS REQUEST;
1 → BUS RELEASED
{ $ß_{op}$ rappresenta il segnale che identifica le operazioni che necessitano del trasferimento Ram – Cache, la rete combinatoria che lo descrive è presente nei fogli allegati }
if $ß_{op}$=1 then
{ verifica della presenza del blocco in cache }
$IR_{TEMP\ 8-19}$ → $MAR_{CACHE\ 6-17}$, 0→ $MAR_{CACHE\ 0-5}$
$M[MAR_{CACHE}]$ → MBR_{CACHE}
If $MBR_{CACHE\ -\ 0}$= 0 then {Il dato non è presente in cache, il blocco corrispondente va trasferito}
{salviamo l'indirizzo della riga di tabella del blocco che stiamo per inserire (non possiamo aggiornare tale riga perché si comp rometterebbe la ricerca del blocco vecchio da sovrascrivere}
MAR_{CACHE} → TEMP;
{avvio della ricerca della riga contenente il blocco da sovrascrivere}
0 → $CONT_{TAB}$;
$CONT_{TAB}$ → MAR_{CACHE} , $CONT_{TAB}$+1→ $CONT_{TAB}$;
$M[MAR_{CACHE}]$ → MBR_{CACHE} , $CONT_{TAB}$ → MAR_{CACHE} , $CONT_{TAB}$+1 → $CONT_{TAB}$;
$MBR_{CACHE\ 2-7}$ → $MBR_{TEMP\ CACHE}$
{ $ß_{TAB}$ rappresenta il segnale di confronto tra CPUNT, il contatore che indica il blocco corrente in cache,e $MBR_{TEMP\ CACHE\ 2-7}$ che indica in quale blocco della cache è presente il dato }
2: if $ß_{TAB}$=0 then {si passa alla riga successiva}
$CONT_{TAB}$ → MAR_{CACHE} , $M[MAR_{CACHE}]$ → MBR_{CACHE} , $CONT_{TAB}$+1→ $CONT_{TAB}$, goto 2 ;
else {il blocco è stato trovato nella tabella alla posizione $CONT_{TAB}$-2}
$CONT_{TAB}$ –1 → $CONT_{TAB}$; $CONT_{TAB}$ -1→ $CONT_{TAB}$;
{avendo effettuato la ricerca del blocco da cancellare, possiamo aggiornare la tabella sulla riga del blocco da inserire}
TEMP → MAR_{CACHE} , CPUNT → $TEMP2_{2-7}$,1→ $TEMP2_{0}$,0 → $TEMP2_{1}$;
TEMP2 → MBR_{CACHE} ;
MBR_{CACHE} → $M[MAR_{CACHE}]$;
{a questo punto possiamo verificare se è necessario il trasferimento cache-ram}
{lettura del campo modificato del blocco da cancellare}
$CONT_{TAB}$ → MAR_{CACHE};
$M[MAR_{CACHE}]$ → MBR_{CACHE};
if $MBR_{CACHE-1}$ =1 then {il blocco è stato modificato}
{aggiornamento della riga della tabella del blocco da sovrascrivere}

0 → TEMP2 ;
TEMP2 → MBR_{CACHE} ;
MBR_{CACHE} → M[MAR_{CACHE}] ;
{trasferimento cache-ram del blocco modificato da sovrascrivere}
$CONT_{TAB}$ → $CONT_{RAM\ 0\text{-}11}$, 0 → $CONT_{RAM\ 12\text{-}23}$, CPUNT → $CONT_{CACHE\ 0\text{-}5}$, 0 → $CONT_{CACHE\ 6\text{-}17}$;
1 → BUSREQUEST; { facciamo richiesta all'arbitro per avere accesso al bus }
3: if BUS GRANT = O then goto 3
else
$CONT_{RAM}$ → MAR_{RAM} , $CONT_{RAM}$ +1 → $CONT_{RAM}$,$CONT_{CACHE}$ → MAR_{CACHE} , $CONT_{CACHE}$+1→$CONT_{CACHE}$;
M[MAR_{CACHE}] → MBR_{CACHE} , $CONT_{CACHE}$ → MAR_{CACHE} , $CONT_{CACHE}$+1→$CONT_{CACHE}$;
MBR_{CACHE} → MBR_{RAM} , M[MAR_{CACHE}] → MBR_{CACHE} , $CONT_{CACHE}$ → MAR_{CACHE} , $CONT_{CACHE}$+1 → $CONT_{CACHE}$;
4 : if NOT($ß_{CONT\text{-}CACHE\ 6\text{-}17}$) = 0 then
$CONT_{CACHE}$ →MAR_{CACHE} ,$CONT_{CACHE}$+1→$CONT_{CACHE}$, M[MAR_{CACHE}] →MBR_{CACHE} , MBR_{CACHE} →MBR_{RAM}
MBR_{RAM} → M[MAR_{RAM}], $CONT_{RAM}$→ MAR_{RAM} , $CONT_{RAM}$ + 1 → $CONT_{RAM}$, goto 4;
else
M[MAR_{CACHE}] → MBR_{CACHE} , MBR_{CACHE} → MBR_{RAM} , MBR_{RAM} → M[MAR_{RAM}] , $CONT_{RAM}$ → MAR_{RAM} ,
$CONT_{RAM}$ + 1 → $CONT_{RAM}$;
MBR_{CACHE} → MBR_{RAM} , MBR_{RAM} → M[MAR_{RAM}] , $CONT_{RAM}$→MAR_{RAM} ;
MBR_{RAM} → M[MAR_{RAM}] ;
0 → BUS REQUEST; { rilasciamo il bus }
1 → BUS RELEASED
fi
else {il blocco non è stato modificato, non viene salvato in ram ma si aggiorna ugualmente la riga della tabella del blocco da sovrascrivere}
0 → TEMP2 ;
TEMP2 → MBR_{CACHE} ;
MBR_{CACHE} → M[MAR_{CACHE}] ;
fi
{finalmente può avvenire il trasferimento ram-cache}
1 → BUSREQUEST; { facciamo richiesta all'arbitro per avere accesso al bus }
5: if BUS GRANT = O then goto 5
else
$IR_{X\ TEMP\ 8\text{-}19}$ → $CONT_{RAM\ 0\text{-}11}$, 0 → $CONT_{RAM\ 12\text{-}23}$, CPUNT → $CONT_{CACHE\ 0\text{-}5}$, 0 → $CONT_{CACHE\ 6\text{-}17}$, CPUNT +1 → CPUNT;
$CONT_{RAM}$ → MAR_{RAM} , $CONT_{RAM}$ + 1 → $CONT_{RAM}$, $CONT_{CACHE}$ → MAR_{CACHE} , $CONT_{CACHE}$ +1 → $CONT_{CACHE}$;
$CONT_{RAM}$ → MAR_{RAM} , $CONT_{RAM}$ +1 → $CONT_{RAM}$, M[MAR_{RAM}] → MBR_{RAM} ;
MBR_{RAM} → MBR_{CACHE} , M[MAR_{RAM}] → MBR_{RAM} , $CONT_{RAM}$ → MAR_{RAM} , $CONT_{RAM}$ +1 → $CONT_{RAM}$;
6:if NOT (OR ($ß_{CONT\ RAM}$)) = 0 then
$CONT_{RAM}$ → MAR_{RAM} , $CONT_{RAM}$ +1 → $CONT_{RAM}$, M[MAR_{RAM}] → MBR_{RAM} , MBR_{RAM} → MBR_{CACHE} ,
MBR_{CACHE} → M[MAR_{CACHE}] , $CONT_{CACHE}$ → M[MAR_{CACHE}] , $CONT_{CACHE}$ +1 → $CONT_{CACHE}$, goto 6 ;
else
M[MAR_{RAM}] → MBR_{RAM} , MBR_{RAM} → MBR_{CACHE} , MBR_{CACHE} → M[MAR_{CACHE}], $CONT_{CACHE}$ → MAR_{CACHE} ,
$CONT_{CACHE}$ +1 → $CONT_{CACHE}$;

$MBR_{RAM} \rightarrow MBR_{CACHE}$, $MBR_{CACHE} \rightarrow M[MAR_{CACHE}]$, $CONT_{CACHE} \rightarrow MAR_{CACHE}$;
$MBR_{CACHE} \rightarrow M[MAR_{CACHE}]$; fi
0 → BUS REQUEST; { rilasciamo il bus }
1 → BUS RELEASED
else ∅ fi {il dato è presente in cache, quindi non si esegue alcun trasferimento}
else ∅ fi {l'operazione non richiede alcun trasferimento}
$IR_{TEMP} \rightarrow IR$;
fi

CALL X (00001)

{ L' assunzione, abbastanza semplificativa, è che le procedure non prevedono il passaggio di parametri: questo, in generale, sarebbe di competenza del programmatore assembler che però dovrebbe disporre di altre operazioni quali la PUSH e la POP.
Proprio per questa ragione, nel context-switching, non si e' ritenuto opportuno, nel salvataggio dello stato, includere l'accumulatore al fine di consentire che alle procedure di utilizzare una variabile globale che potrebbe essere l'accumulatore o una porzione di memoria indirizzata dall'accumulatore . }
{ In presenza di una call l'indirizzo del programma in esecuzione contenuto nel PC viene salvato in una pila che per comodita' è stata realizzata utilizzando le prime 255 celle della ram, eccetto la prime tre che, come previsto dalle specifiche del progetto, contengono l'indirizzo di partenza della routine di servizio . Un puntatore (R) , ad otto bit ad incremento ed a decremento che viene inizializzato a 255, tiene costantemente conto dell'ultima cella occupata .La pila viene caricata dalla cella 255 alla cella 3 : per tenere conto delle prime tre celle (0,1,2) occupate si introduce un transcoder che consente il controllo su R-3 per evitare la sovrascrittura .}

DECR (R) → R ;
OR (R) → β_R ;
{Verifichiamo il segnale β_r che vale zero nel caso di pila piena . In tale situazione viene mandato un segnale di errore alla parte di controllo . }
if β_R = 0 then 1 → ERROR LINE;
else
    1 → BUS REQUEST;
    0 : if BUS GRANT = 0 then goto 0
      else
        {Salviamo PC nella pila che si trova in Ram tale salvataggio viene effettuato in tre tempi grazie al selettore di ingressi .
        Infine abilitiamo con le ultime operazioni la lettura del flip-flop di controllo che indica la presenza di un eventuale interrupt (vedi circuito interrupt) . }
        $PC_{0\text{-}7}$ → MBR_{RAM} , R → MAR_{RAM} , R–1 → R;
        MBR_{RAM} → M[MAR_{RAM}] , $PC_{8\text{-}15}$ → MBR_{RAM} , R → MAR_{RAM} , R-1 → R;
        MBR → M[MAR_{RAM}] , $PC_{16\text{-}24}$ → MBR_{RAM} , R → MAR_{RAM} ;
        MBR_{RAM} → M[MAR_{RAM}];
        0 → BUS REQUEST, 1 → A_K;
        1 → BUS RELEASE, 1 → $A_{BINT,}$ 0 → A_K ;
      fi
    {Carichiamo il PC con il nuovo indirizzo }
    $IR_{8\text{-}31}$ → PC ;
fi

RET (00010)

{ Si deve ripristinare il PC con l'indirizzo salvato al momento dell'ultima call .}

1 → BUS REQUEST;

0 : if BUS GRANT = 0 then goto 0

else

{Carichiamo il PC in tre tempi shiftando i bit ad ogni operazione.

Infine abilitiamo con le ultime operazioni la lettura del flip-flop di controllo che indica la presenza di un eventuale interrupt (vedi circuito interrupt) . }

$R \rightarrow MAR_{RAM}$, $R+1 \rightarrow R$;

$M[MAR_{RAM}] \rightarrow MBR_{RAM}$, $R \rightarrow MAR_{RAM}$, $R+1 \rightarrow R$;

$M[MAR_{RAM}] \rightarrow MBR_{RAM}$, $MBR \xrightarrow{RSHIFT} PC_{0-7}$, $R \rightarrow MAR_{RAM}$, $R+1 \rightarrow R$;

$M[MAR_{RAM}] \rightarrow MBR_{RAM}$, $MBR \xrightarrow{RSHIFT} PC_{0-7}$;

$MBR \xrightarrow{RSHIFT} PC_{0-7}$;

$PC+1 \rightarrow PC$;

$0 \rightarrow$ BUS REQUEST, $1 \rightarrow A_K$;

$1 \rightarrow$ BUS RELEASE, $1 \rightarrow A_{BINT}$, $0 \rightarrow A_K$;

fi

FORWARD (00011)

{ Bisogna trasferire una porzione di Ram su uno dei canali DMA. Tale porzione puo' comprendere al più 255 celle di Ram visto che l'accumulatore e' ad 8 bit . Tale trasferimento quindi puo' coinvolgere al massimo due blocchi di Ram . Questo comporta che se tali blocchi sono stati modificati in cache e' necessario aggiornare la Ram con un trasferimento cache-ram prima di attivare il DMA . Questa procedura poteva essere effettuata nella fetch ma per non appesantire ulteriormente il codice e quindi per una maggiore leggibilita' , si e' preferito fare tale trasferimento direttamente nella Forward ,anche se questo comporta la necessita' di riscrivere una porzione di codice. }

{ Come primo passo si cerca il blocco corrispondente al dato X in cache }

$IR_{X\,8\text{-}19} \rightarrow MAR_{CACHE\,6\text{-}17}$, $0 \rightarrow MAR_{CACHE\,0\text{-}5}$;

$M[MAR_{CACHE}] \rightarrow MBR_{CACHE}$;

{ Si verifica se il blocco è presente e modificato; in alternativa si passa a verificare l'eventuale secondo blocco interessato. }

If ($MBR_{CACHE\,0} = 1$ AND $MBR_{CACHE\,1} = 1$) then

{A questo punto e' necessario aggiornare la ram trasferendo il blocco modificato , la procedura di trasferimento e' la stessa utilizzata nella fetch .}

$MBR_{CACHE\,2-7} \rightarrow CONT_{CACHE\,0\text{-}5}$, $0 \rightarrow CONT_{CACHE\,6\text{-}17}$, $IR_{X\,8\text{-}19} \rightarrow CONT_{RAM\,0\text{-}11}$, $0 \rightarrow CONT_{RAM\,12\text{-}23}$;

$1 \rightarrow$ BUS REQUEST;

0 : if BUS GRANT = 0 then goto 0

else

$CONT_{RAM} \rightarrow MAR_{RAM}$, $CONT_{RAM} + 1 \rightarrow CONT_{RAM}$, $CONT_{CACHE} \rightarrow MAR_{CACHE}$, $CONT_{CACHE} + 1 \rightarrow CONT_{CACHE}$;

$M[MAR_{CACHE}] \rightarrow MBR_{CACHE}$, $CONT_{CACHE} \rightarrow MAR_{CACHE}$, $CONT_{CACHE} + 1 \rightarrow CONT_{CACHE}$;

$MBR_{CACHE} \rightarrow MBR_{RAM}$, $M[MAR_{CACHE}] \rightarrow MBR_{CACHE}$, $CONT_{CACHE} \rightarrow MAR_{CACHE}$, $CONT_{CACHE} + 1 \rightarrow CONT_{CACHE}$;

4 : if NOT($ß_{CONT\text{-}CACHE\,6\text{-}17}$) = 0 then

$CONT_{CACHE} \rightarrow MAR_{CACHE}$, $CONT_{CACHE} + 1 \rightarrow CONT_{CACHE}$, $M[MAR_{CACHE}] \rightarrow MBR_{CACHE}$, $MBR_{CACHE} \rightarrow MBR_{RAM}$

$MBR_{RAM} \rightarrow M[MAR_{RAM}]$, $CONT_{RAM} \rightarrow MAR_{RAM}$, $CONT_{RAM} + 1 \rightarrow CONT_{RAM}$, goto 4;

else

$M[MAR_{CACHE}] \rightarrow MBR_{CACHE}$, $MBR_{CACHE} \rightarrow MBR_{RAM}$, $MBR_{RAM} \rightarrow M[MAR_{RAM}]$, $CONT_{RAM} \rightarrow MAR_{RAM}$,

$CONT_{RAM} + 1 \rightarrow CONT_{RAM}$;

$MBR_{CACHE} \rightarrow MBR_{RAM}$, $MBR_{RAM} \rightarrow M[MAR_{RAM}]$, $CONT_{RAM} \rightarrow MAR_{RAM}$;

$MBR_{RAM} \rightarrow M[MAR_{RAM}]$;

$0 \rightarrow$ BUS REQUEST;

$1 \rightarrow$ BUS RELEASED ;

{ Fine del trasferimento cache – ram }

fi

fi

else ∅ fi

{Per valutare se è coinvolto anche un secondo blocco si controlla se c'e' uno zero nei bit che vanno dal 20 al 23 dell'IR, che rappresentano i bit piu' significativi della metà destra del dato X , se cio' accade (il segnale β_{FOR} sara' uguale a zero) non si ha un secondo blocco interessato . Tale condizione tuttavia e' solo necessaria .}

if $\beta_{FOR/\,RET} = 1$ then

{ Per avere la condizione sufficiente adesso si sommano gli 8 bit meno significativi del dato con l'accumulatore : se tale risultato da un supero siamo certi che un secondo blocco e' stato coinvolto dal trasferimento DMA }

$IR_{24\text{-}31} \rightarrow A_1$, $A \rightarrow A_2$;

if sup ($A_1 + A_2$) = 1 then

{ Come primo passo si cerca il blocco corrispondente al dato X in cache }

$IR_{X\,8\text{-}19} + 1 \rightarrow MAR_{CACHE\,6\text{-}17}$, $0 \rightarrow MAR_{CACHE\,0\text{-}5}$;

$M[MAR_{CACHE}] \rightarrow MBR_{CACHE}$;
{ Si verifica se il blocco è presente e modificato; in alternativa si passa a verificare l'eventuale secondo blocco interessato. }
If $MBR_{CACHE\ 0} = 1$ AND $MBR_{CACHE\ 1} = 1$ then
{A questo punto e' necessario aggiornare la ram trasferendo il blocco modificato ; la procedura di trasferimento e' la stessa utilizzata nella fetch .}
$MBR_{CACHE\ 2-7} \rightarrow CONT_{CACHE\ 0-5}$, $0 \rightarrow CONT_{CACHE\ 6-17}$, $IR_{X\ 8-19} \rightarrow CONT_{RAM\ 0-11}$, $0 \rightarrow CONT_{RAM\ 12-23}$;
$1 \rightarrow$ BUS REQUEST;
0 : if BUS GRANT = 0 then goto 0
else
$CONT_{RAM} \rightarrow MAR_{RAM}$, $CONT_{RAM}+1 \rightarrow CONT_{RAM}$, $CONT_{CACHE} \rightarrow MAR_{CACHE}$, $CONT_{CACHE}+1 \rightarrow CONT_{CACHE}$;
$M[MAR_{CACHE}] \rightarrow MBR_{CACHE}$, $CONT_{CACHE} \rightarrow MAR_{CACHE}$, $CONT_{CACHE}+1 \rightarrow CONT_{CACHE}$;
$MBR_{CACHE} \rightarrow MBR_{RAM}$, $M[MAR_{CACHE}] \rightarrow MBR_{CACHE}$, $CONT_{CACHE} \rightarrow MAR_{CACHE}$, $CONT_{CACHE}+1 \rightarrow CONT_{CACHE}$;
4 : if NOT($ß_{CONT\text{-}CACHE\ 6-17}$) = 0 then
$CONT_{CACHE} \rightarrow MAR_{CACHE}$, $CONT_{CACHE}+1 \rightarrow CONT_{CACHE}$, $M[MAR_{CACHE}] \rightarrow MBR_{CACHE}$, $MBR_{CACHE} \rightarrow MBR_{RAM}$
$MBR_{RAM} \rightarrow M[MAR_{RAM}]$, $CONT_{RAM} \rightarrow MAR_{RAM}$, $CONT_{RAM} + 1 \rightarrow CONT_{RAM}$, goto 4;
else
$M[MAR_{CACHE}] \rightarrow MBR_{CACHE}$, $MBR_{CACHE} \rightarrow MBR_{RAM}$, $MBR_{RAM} \rightarrow M[MAR_{RAM}]$, $CONT_{RAM} \rightarrow MAR_{RAM}$,
$CONT_{RAM} + 1 \rightarrow CONT_{RAM}$;
$MBR_{CACHE} \rightarrow MBR_{RAM}$, $MBR_{RAM} \rightarrow M[MAR_{RAM}]$, $CONT_{RAM} \rightarrow MAR_{RAM}$;
$MBR_{RAM} \rightarrow M[MAR_{RAM}]$;
$0 \rightarrow$ BUS REQUEST;
$1 \rightarrow$ BUS RELEASED ;
{ Fine trasferimento cache –ram }
fi
fi
else ∅
else ∅ ; fi
else ∅ ; fi
{ Prima di caricare i registri di S si verifica che tale coprocessore non sia già occupato da un altra operazione, in tal caso M sta in busy-waiting.}
2: if $B_D = 1$ then goto 2
else
{ Carichiamo i registri di S ed inseriamo il codice operazione della Forward di S.
Con le ultime tre operazioni abilitiamo la lettura del flip-flop di controllo che indica la presenza di un eventuale interrupt (vedi circuito interrupt) .
Con µ-istruzione $1 \rightarrow$ BD si disabilitano ulteriori richieste nelle µ-sequenze successive (finchè S non avrà attivato il DMA e resettato il flip-flop)}
$1 \rightarrow BD, A \rightarrow A_{cs}$, $IR_{X\ 8-31} \rightarrow X_S$, $IR_{X\ 7} \rightarrow D$, <000> → S-CODE-OP., $1 \rightarrow A_K$;
$1 \rightarrow A_{BINT}$, $0 \rightarrow A_K$;
fi

RETRY X (00100)

{ Si deve effettuare il trasferimento da un Device alla Ram. Tale trasferimento puo' modificare uno o due blocchi in Ram : non appena finito il trasferimento sara' quindi necessario aggiornare la cache e tale operazione non potra' che avvenire nella fetch durante la gestione dell'interrupt proveniente dal Dma. All'inizio M pone un flip flop RETRY ON ad 1 che servira' nella fetch (dove verra' posto a zero) a distinguere l'interrupt proveniente dal Dma a seguito di una retry dall'interrupt successivo ad una forward . Infatti la Retry presenta un problema in un caso particolare : mentre il Dma e' interessato al trasferimento su Ram, il processore M potrebbe dover sovrascrivere in cache , e quindi trasferire in Ram, il blocco che il Dma sta modificando con la conseguenza che si potrebbero perdere le informazioni che si stanno inviando dal canale Dma . Questo problema viene risolto nella fetch : prima di effettuare il trasferimento cache-ram viene controllato il flip flop RETRY ON che informa sullo stato del Dma.All'esecuzione della Retry si memorizza in un registro l'indirizzo del blocco interessato dal trasferimento, ed in un flip flop (SECOND-BLOCK-ON) l'eventuale conferma del coinvolgimento del secondo blocco ; tale flip flop verra' posto a zero sempre all'interno della fetch. Se si verifica tale inconveniente si e' assunto che il blocco in cache non debba venire sovrascritto ed il puntatore della coda della cache sara' incrementato al blocco successivo. }

1→ RETRY ON, IR_{8-19}→B_{TEMP}; {Si memorizza l'indirizzo del blocco nel registro Btemp}

{Per valutare se è coinvolto anche un secondo blocco si controlla se c'e' uno zero nei bit che vanno dal 20 al 23 dell'IR, che rappresentano i bit piu' significativi della metà destra del dato X , se cio' accade (il segnale β_{FOR} sara' uguale a zero) non si ha un secondo blocco interessato . Tale condizione tuttavia e' solo necessaria .}

if $\beta_{FOR\,/\,RET}$ = 1 then

{ Per avere la condizione sufficiente adesso si sommano gli 8 bit meno significativi del dato con l'accumulatore : se tale risultato da un supero siamo certi che un secondo blocco e' stato coinvolto dal trasferimento DMA }

IR_{24-31}→ A_1 , A → A_2 ;

if sup (A_1 + A_2) = 1 then

1 → SECOND-BLOCK-ON

else ∅ fi

else ∅ fi

{ Prima di caricare i registri di S si verifica che tale coprocessore non sia già occupato da un altro operazione, in tal caso M sta in busy-waiting.}

0 : if BD = 1 then goto 0

else

{ Carichiamo i registri di S ed inseriamo il codice operazione della Forward di S.

Con µ-istruzione 1→BD si disabilitano ulteriori richieste nelle µ-sequenze successive (finchè S non avrà attivato il DMA e resettato il flip-flop)}

Con le ultime tre operazioni abilitiamo la lettura del flip-flop di controllo che indica la presenza di un eventuale interrupt (vedi interrupt) . }

1→BD,A → Acs, IR_{8-31} → Xs, IR_7 → d , <001> → S-CODE-OP, 1 → A_K;

1 → $A_{BINT,}$ 0 → A_K ;

fi

RETI (00101)

{ Si ripristinano il PC , l'accumulatore (A) e la priorità attraverso il registro PR , (vedi circuito degli interrupt) precedenti all'interruzione.Lo stato , salvato in un register pool, viene ripristinato grazie al puntatore PUNT che indica l'ultimo registro in cui si e' salvato il PC , A ed il PR. Per semplicità si è scelto di mettere in un unico registro il PC , A e il PR. Infine abilitiamo con le ultime operazioni la lettura del flip-flop di controllo che indica la presenza di un eventuale interrupt (vedi circuito interrupt) . }

$R_{PUNT\ 24\text{-}26} \rightarrow PR$, $R_{PUNT\ 0\text{-}23} \rightarrow PC$, $R_{PUNT\ 27\text{-}34} \rightarrow A$;

PC+1 –> PC, PUNT - 1 → PUNT , $1 \rightarrow A_K$;

$1 \rightarrow A_{BINT,}\ 0 \rightarrow A_K$;

MASK C (00110)

{Attraverso la Mask il programmatore deve poter mascherare delle interruzioni che non devono essere servite, quindi pone negli ultimi quattro bit dell'IR la nuova configurazione che sara' caricata nel registro mask, con la convenzione che saranno poste a zero le interruzioni non desiderate. Vista la connessione hardware sempre presente tra gli ultimi 4 bit dell'IR e i 4 flip flop del registro mask e' sufficiente abilitare la scrittura .

Infine abilitiamo la lettura del flip-flop di controllo che indica la presenza di un eventuale interrupt (vedi circuito interrupt) . }

$1 \rightarrow A_{MASK}$, $1 \rightarrow A_K$;

$1 \rightarrow A_{BINT,}\ 0 \rightarrow A_K$;

WAIT C (10001)

{Tramite la Wait il programmatore ha la facolta' di mettere la macchina in busy-waiting che avvenga una delle interruzioni da lui specificate negli ultimi 4 bit dell'istruzione .

Il segnale β_{MASK} restituira' 1 solo al verificarsi di questa condizione.

Infine abilitiamo la lettura del flip-flop di controllo che indica la presenza di un eventuale interrupt (vedi circuito interrupt) . }

0 : if $\beta_{MASK} = 0$ then goto 0
else
$1 \rightarrow A_K$;
$1 \rightarrow A_{BINT,}\ 0 \rightarrow A_K$;
fi

WRITE X (01000)

{ Si chiede ad S il trasferimento del dato presente all'indirizzo X .

Il dato presente in cache , è prelevato con il consueto meccanismo di ricerca in cache tramite tabella (vedi pagina ……) }
$IR_{X\ 8\text{-}19}$ → $MAR_{CACHE\ 6\text{-}17}$, 0 → $MAR_{CACHE\ 0\text{-}5}$;
M[MAR_{CACHE}] → MBR_{CACHE} ;
$MBR_{CACHE\ 2\text{-}7}$ → $MAR_{CACHE\ 0\text{-}5}$, $IR_{X\ 20\text{-}31}$ → $MAR_{CACHE\ 6\text{-}17}$;
M[MAR_{CACHE}] → MBR_{CACHE} ;
MBR_{CACHE} → A;
{ Prima di caricare i registri di S si verifica che tale coprocessore non sia già occupato da un altro operazione, in tal caso M sta in busy-waiting.}
0 : if BD = 1 then goto 0
else
{ Carichiamo i registri di S ed inseriamo il codice operazione della Forward di S.
L'operazione write richiede l'uso in S del registro A_{cs} ; per questa ragione bisogna porre ad 1 il flip-flop di S BD che non consentirà ad altre operazioni che fanno uso dello stesso registro in S di sovrascriverlo e modificare lo stato di S}
Con le ultime tre operazioni abilitiamo la lettura del flip-flop di controllo che indica la presenza di un eventuale interrupt (vedi circuito interrupt) . }
1→BD,A → $A_{CS,}$ <011> → S-CODE-OP, 1 → A_K ;
1 → $A_{BINT,}$ 0 → A_K ;
fi

READ X (00111)
{ Si chiede ad S di prelevare un dato dalle periferiche di I/O che dovrà essere riposto nell'accumulatore al momento della gestione dell'interrupt proveniente da S.
A tal fine e' necessario salvare l'indirizzo di destinazione in un registro temporaneo (Xtemp) , da cui poi M dovrà prelevare l'indirizzo di memoria una volta verificatosi l'interrupt proveniente da S per segnalare la fine della read I/O . Inoltre M setta ad uno il flip flop READ ON (che verra' posto a zero nella fetch) in modo da poter distinguere un interrupt di I/O a seguito di una READ da quello seguente una WRITE.}
$IR_{8\text{-}31}$ → X_{TEMP} , 1→ READ ON ;
{ Prima di caricare i registri di S si verifica che tale coprocessore non sia già occupato da un altro operazione, in tal caso M sta in busy-waiting.}
0 : if BD = 1 then goto 0
else
{ Carichiamo i registri di S ed inseriamo il codice operazione della Forward di S.
Per quanto riguarda l'istruzione 1→BD valgono le stesse considerazioni fatte per la write}
Con le tre operazioni in blu abilitiamo la lettura del flip-flop di controllo che indica la presenza di un eventuale interrupt (vedi circuito interrupt) . }
1→BD,<010> → S-CODE-OP, 1 → A_K ;
1 → $A_{BINT,}$ 0 → A_K ;
fi

LOAD X (01001)

{Il dato collocato all'indirizzo x di Ram, sicuramente presente in cache, viene trasferito nel registro A attraverso la seguente procedura: il numero del blocco Ram corrispondente è dato dai 12 bit più significativi dell' indirizzo X: tali bit forniscono quindi anche la riga della tabella cache dove sono contenute le informazioni sulla collocazione di tale blocco nella partizione della cache. (vedi dettagli nei meccanismi di trasferimento Ram-Cache) . I sei bit meno significativi della cella a tale indirizzo rappresentano quindi i sei bit più significativi dell' indirizzo del dato in cache che può ora essere letto e posto nell'accumulatore.

Con le tre operazioni in blu abilitiamo la lettura del flip-flop di controllo che indica la presenza di un eventuale interrupt (vedi circuito interrupt) . }

$IR_{X\ 8\text{-}19} \rightarrow MAR_{CACHE\ 6\text{-}17}$, $0 \rightarrow MAR_{CACHE\ 0\text{-}5}$;

$M[MAR_{CACHE}] \rightarrow MBR_{CACHE}$;

$MBR_{CACHE\ 2\text{-}7} \rightarrow MAR_{CACHE\ 0\text{-}5}$, $IR_{X\ 20\text{-}31} \rightarrow MAR_{CACHE\ 6\text{-}17}$;

$M[MAR_{CACHE}] \rightarrow MBR_{CACHE}$, $1 \rightarrow A_K$;

$MBR_{CACHE} \rightarrow A$, $1 \rightarrow A_{BINT}$, $0 \rightarrow A_K$;

LOAD #X (01011)

{Il dato da trasferire in A è presente negli otto bit meno significativi dell'IR

Con le tre operazioni in blu abilitiamo la lettura del flip-flop di controllo che indica la presenza di un eventuale interrupt (vedi circuito interrupt) . }

$IR_{X\ 24\text{-}31} \rightarrow A$, $1 \rightarrow A_K$;

$1 \rightarrow A_{BINT}$, $0 \rightarrow A_K$;

STORE X (01010)

{La memorizzazione dell'accumulatore all'indirizzo x di Ram, dato l'uso del protocollo write-back, avviene esclusivamente in cache. Certi del fatto (per come è organizzata la fetch) che il blocco che contiene x è presente in cache, si ricerca in tabella la sua posizione per ricostruirne l'indirizzo esatto e trasferirvi x.

$IR_{X\ 8\text{-}19} \rightarrow MAR_{CACHE\ 6\text{-}17}$, $0 \rightarrow MAR_{CACHE\ 0\text{-}5}$;

$M[MAR_{CACHE}] \rightarrow MBR_{CACHE}$;

{Individuata la riga di tabella, aggiorniamone anche il campo 'modificato' rappresentato dal secondo bit (bit numero 1)

La soluzione che segue, compatibile con la parte operativa di M, non è certo la più efficiente. Una soluzione ottimale dovrebbe invece prevedere tale modifica direttamente sul registro MBR della cache. Ai fini del progetto, però, questo violerebbe l'assunzione iniziale secondo la quale cache e ram sono da considerarsi scatole chiuse con operazioni predefinite.

$MBR_{CACHE} \rightarrow TEMP3$;$1 \rightarrow TEMP3_1$;

$TEMP3 \rightarrow MBR_{CACHE}$

{Scriviamo sul MAR l'indirizzo della cella in cache e trasferiamo il dato}

Con le tre operazioni in blu abilitiamo la lettura del flip-flop di controllo che indica la presenza di un eventuale interrupt (vedi circuito interrupt) . }

$MBR_{CACHE\ 2\text{-}7} \rightarrow MAR_{CACHE\ 0\text{-}5}$, $IR_{X\ 20\text{-}31} \rightarrow MAR_{CACHE\ 6\text{-}17}$;

$A \rightarrow MBR_{CACHE}$, $1 \rightarrow A_K$;

$MBR_{CACHE} \rightarrow M[MAR_{CACHE}]$, $1 \rightarrow A_{BINT}$, $0 \rightarrow A_K$;

JZ X (01100)

{L'assunzione è che il programmatore voglia effettuare un salto di un numero di celle rappresentabile con 8 bit in complemento a 2. In effetti è lecito pensare che un salto relativo può avere senso nell'ambito di piccoli spostamenti, altrimenti sarebbe opportuno, per chi scrive il programma, specificare l'indirizzo assoluto di arrivo. Anche questa assunzione, naturalmente, può considerarsi discutibile, ma altre interpretazioni non cambierebbero in modo considerevole gli aspetti progettuali e le problematiche connesse al salto. Infatti, se il salto fosse consentito a 24 bit, si potrebbe procedere nei due modi seguenti: o introducendo un ALU interna a 24 bit e utilizzandola anche per le somme e le sottrazioni a 8 bit (evitando così gli inconvenienti che l'assunzione fatta, come si vedrà, comporta), oppure continuando a fare le somme a 24 bit con l'ALU a 8 bit, tenendo in conto il riportOcon un meccanismo analogo a quello usato nell' ipotesi adottata.}
{Come primo passo si verifica se A vale 0, altrimenti il programma procede senza variazioni.}
If β_A =0 then
{Si sommano gli 8 bit meno significativi del PC con gli 8 bit meno significativi dell'IR dove è scritto il numero di celle da saltare}
$PC_{16-23} \rightarrow A_1$, $IR_{24-31} \rightarrow A_2$;
$A_1 + A_2 \rightarrow PC_{16-23}$;
{Si distinguono ora gli incrementi negativi dai positivi, seguendo due algoritmi diversi. Nel caso il programmatore richieda un salto in avanti del PC, si controlla il supero della somma $A_1 + A_2$. Se questo è 0 abbiamo finito, altrimenti bisogna sommare ai restanti 16 bit tale riporto; anche questa somma viene fatta in due tempi considerando dapprima gli 8 bit centrali del PC e controllando nuovamente il supero. L'ulteriore incremento +1 poteva essere realizzato sfruttando un registro a incremento senza passare tramite l' ALU. Si è fatta questa scelta per scrivere un codice simile a quello da usare nel caso di JZ con argomento a 24 bit mantenendo l'ALU a 8 bit (dove, al posto di –1, verrà caricato nel registro tampone A_2 il corrispondente blocco di bit di X)}
{controllo sul segno di X}
if IR_{24} =0 then {è positivo o nullo}
if sup ($A_1 + A_2$) = 1 then
$PC_{8-15} \rightarrow A_1$, $1 \rightarrow A_2$;
$A_1 + A_2 \rightarrow PC_{8-15}$;
{Si ripete l'analogo ragionamento per gli 8 bit centrali; si controlla il supero dell'incremento +1: se è nullo, abbiamo finito}
If sup(A_1+A_2) = 1 then {dobbiamo incrementare di uno anche i primi 8 bit}
$PC_{0-7} \rightarrow A_1$;
$A_1 + A_2 \rightarrow PC_{0-7}$;
{Se si ha ancora supero, il programmatore ha commesso un errore}
If SUP = 1 then
1 → ERROR LINE
else ∅ fi
else ∅ fi{abbiamo finito, il controllo può tornare alla fetch}
else ∅ fi{abbiamo finito, il controllo può tornare alla fetch}

else {è negativo}

{in questo caso, se il supero della somma A_1+A_2 è 1 non bisogna fare nulla; altrimenti si controlla il bit più significativo della somma: se vale 0 non bisogna fare nulla, se è 1 si devono decrementare di 1 i restanti bit del PC}

if sup ($A_1 + A_2$) = 0 then

if (bit-0= 1) then

PC_{8-15} → A_1 ,-1→A_2;

$A_1 + A_2$ → PC_{8-15} ;

{adesso bisogna verificare che i bit 8-15 non fossero tutti 0, perché in questo caso bisognerebbe decrementare di uno anche i primi 8 bit; questo può avvenire semplicemente controllando se si è avuto supero nella somma:se non si è avuto supero, i bit erano tutti 0}

If sup (A_1+A_2) = 0 then

PC_{0-7} → A_1 ;

$A_1 + A_2$ → PC_{0-7} ;

{ripetendo il ragionamento, se non si è avuto nuovamente supero, il salto porterebbe a un PC negativo e si avrebbe un errore di programmazione}

If SUP = 1 then

1 → ERROR LINE

else ∅ ; fi

else ∅ ; fi

else ∅ fi {il bit in posizione 0 nullo e l'algoritmo termina}

else ∅ fi {il supero vale 1 e l'algoritmo termina}

else ∅ fi

Con le tre operazioni in blu abilitiamo la lettura del flip-flop di controllo che indica la presenza di un eventuale interrupt (vedi circuito interrupt) . }

1 → A_K;

1 → $A_{BINT,}$ 0 → A_K ;

ADD X (01101)

{L'assunzione è che le operazioni vengano effettuate con operandi a 8 bit.
Si ricerca la cella corrispondente all'indirizzo X con la consueta politica di gestione della cache}

$IR_{X\ 8\text{-}19} \rightarrow MAR_{CACHE\ 6\text{-}17}$, $0 \rightarrow MAR_{CACHE\ 0\text{-}5}$;

$M[MAR_{CACHE}] \rightarrow MBR_{CACHE}$;

$MBR_{CACHE\ 2\text{-}7} \rightarrow MAR_{CACHE\ 0\text{-}5}$, $IR_{X\ 20\text{-}31} \rightarrow MAR_{CACHE\ 6\text{-}17}$;

$M[MAR_{CACHE}] \rightarrow MBR_{CACHE}$;

{il dato è ora disponibile nell' MBR della cache;
trasferiamolo nel registro tampone A_1 dell' ALU, mentre l'accumulatore viene posto nel registro A_2}
Con le tre operazioni in blu abilitiamo la lettura del flip-flop di controllo che indica la presenza di un eventuale interrupt (vedi circuito interrupt) . }

$MBR_{CACHE} \rightarrow A_1$, $A \rightarrow A_2$, $1 \rightarrow A_K$;

{la somma può ora scriversi nel registro A}

$A_1 + A_2 \rightarrow A_C$, $1 \rightarrow A_{BINT}$, $0 \rightarrow A_K$;

SUB X (01110)

{Si deve sottrarre all'accumulatore il dato presente all'indirizzo X (A-M[X]→A)
analogamente a quanto fatto sopra si ha:}

$IR_{X\ 8\text{-}19} \rightarrow MAR_{CACHE\ 6\text{-}17}$, $0 \rightarrow MAR_{CACHE\ 0\text{-}5}$;

$M[MAR_{CACHE}] \rightarrow MBR_{CACHE}$;

$MBR_{CACHE\ 2\text{-}7} \rightarrow MAR_{CACHE\ 0\text{-}5}$, $IR_{X\ 20\text{-}31} \rightarrow MAR_{CACHE\ 6\text{-}17}$;

$M[MAR_{CACHE}] \rightarrow MBR_{CACHE}$;

{il dato, ora disponibile nell' MBR della cache, può essere portato nel registro A_1, mentre l'accumulatore va in A_2 }
Con le tre operazioni in blu abilitiamo la lettura del flip-flop di controllo che indica la presenza di un eventuale interrupt (vedi circuito interrupt) . }

$MBR_{CACHE} \rightarrow A_1$, $A \rightarrow A_2$, $1 \rightarrow A_K$;

{la differenza va quindi in A}

$A_1 - A_2 \rightarrow A_C$, $1 \rightarrow A_{BINT}$, $0 \rightarrow A_K$;

{Per le operazioni di moltiplicazione e di divisione si utilizzano operandi a 8 bit che verranno elaborati seguendo passo per passo la terza versione degli algoritmi trattati rispettivamente a pag.169 e a pag. 181 del Patterson. Tali algoritmi consentono di pervenire al risultato in 8 passi e sfruttano entrambi lo stesso circuito descritto congiuntamente alla parte operativa del coprocessore matematico}

MUL X (01111)

{Il dato è certamente presente in cache e viene spostato nell'MBR}

$IR_{X\ 8\text{-}19} \rightarrow MAR_{CACHE\ 6\text{-}17}$, $0 \rightarrow MAR_{CACHE\ 0\text{-}5}$;

$M[MAR_{CACHE}] \rightarrow MBR_{CACHE}$;

$MBR_{CACHE\ 2\text{-}7} \rightarrow MAR_{CACHE\ 0\text{-}5}$, $IR_{X\ 20\text{-}31} \rightarrow MAR_{CACHE\ 6\text{-}17}$;

$M[MAR_{CACHE}] \rightarrow MBR_{CACHE}$;

{a questo punto può essere inoltrata la richiesta al coprocessore matematico previo accertamento della sua inattività.
Qualora S stesse ancora elaborando una precedente richiesta di calcolo di M, o fosse in attesa di ricevere o inviare dati alle periferiche di I/O, o ancora stesse preparando l'attivazione di un canale DMA, M si troverebbe necessariamente costretto a porsi in stato di busy-waiting}

0 : if B_{calc} =1 then goto 0 {eventuale busy-waiting}

else

{preparazione dei registri e attivazione del coprocessore: osserviamo che il caricamento dei registri di S può essere effettuato nello stesso colpo di clock in cui si modifica il codice operazione di S. Questo è possibile grazie al fatto che l'interfacciamento tra M e S avviene tramite un registro di buffering che assumiamo collegato al clock di M.
Con l'abilitazione del flip-flop B_{calc} in S (che verrà ripristinato a 0 dal coprocessore matematico solo ad operazione conclusa) disabilitiamo la possibilità di richiedere altre operazioni ad S se il coprocessore sta ancora elaborando}

Al solito, con le tre operazioni in blu abilitiamo la lettura del flip-flop di controllo che indica la presenza di un eventuale interrupt (vedi circuito interrupt) . }

$1 \rightarrow B_{calc}$,A $\rightarrow$ M/D , $MBR_{CACHE} \rightarrow P/R_{8\text{-}15}$, <100> $\rightarrow$ S-CODE-OP , $1 \rightarrow A_K$;

$1 \rightarrow A_{BINT}$, $0 \rightarrow A_K$;

fi

DIV X (10000)

{Lettura del dato in cache}

$IR_{X\ 8\text{-}19} \rightarrow MAR_{CACHE\ 6\text{-}17}$, $0 \rightarrow MAR_{CACHE\ 0\text{-}5}$;

$M[MAR_{CACHE}] \rightarrow MBR_{CACHE}$;

$MBR_{CACHE\ 2\text{-}7} \rightarrow MAR_{CACHE\ 0\text{-}5}$, $IR_{X\ 20\text{-}31} \rightarrow MAR_{CACHE\ 6\text{-}17}$;

$M[MAR_{CACHE}] \rightarrow MBR_{CACHE}$;

{verifica dello stato del coprocessore ed eventuale busy-waiting}

0 : if B_{calc} =1 then goto 0

else

{caricamento dei registri di S ed attivazione del codice operativo.
Valgono le analoghe considerazioni fatte nella MUL sulla μ-istruzione $1 \rightarrow B_{calc}$}

Al solito, con le tre operazioni in blu abilitiamo la lettura del flip-flop di controllo che indica la presenza di un eventuale interrupt (vedi circuito interrupt) . }

$1 \rightarrow B_{calc}$,A $\rightarrow$ M/D , $MBR_{CACHE} \rightarrow P/R_{8\text{-}15}$, <101> $\rightarrow$ S-CODE-OP , $1 \rightarrow A_K$;

$1 \rightarrow A_{BINT}$, $0 \rightarrow A_K$;

fi

FORWARD (relativa ad S) (000)

{L'istruzione viene attivata da M dopo aver caricato i registri A_{cs} , X_s e indicato il canale in D. A sua volta, S carica i registri del DMA attraverso una rete che consente di incanalere tali dati verso il controller corretto}

{ S verifica se il DMA indicato da M è occupato ; se ciò accade, S si mette in busy-waiting }

{ Il codice è stato scritto in maniera tale da consentire ad M il maggior numero di richieste possibili; infatti, come si può verificare, S può accogliere una richiesta di trasferimento anche se il DMA sta lavorando; può accogliere e servire richieste sul canale libero anche se l'altro canale è occupato (naturalmente il canale resta quello specificato da M), e inoltre può mettere in coda una richiesta anche se entrambi i canali sono occupati.}

0: if $\beta_{readydma} = 1$ then goto 0
else
{ Vengono caricati da S i registri count e car del DMA e si da' lo start specificando l'operazione. }
$A_{CS} \rightarrow$ CONT , $X_S \rightarrow$ CAR;
$1 \rightarrow$ START , $1 \rightarrow$ OPDMA;
{A questo punto può essere riabilitato il flip-flop che esprime la disponibilità di S a ricevere ulteriori trasferimenti
$0 \rightarrow$ BD;
fi

RETRY X (010)

{Valgono le considerazioni fatte per la forward}

{ S verifica se il DMA indicato da M è occupato , se ciò accade S si mette in busy-waiting }

0: if $\beta_{readydma} = 1$ then goto 0
else
{ Vengono caricati da S i registri count e car del DMA e si da lo start specificando l'operazione. }
$A_{CS} \rightarrow$ CONT , $X_S \rightarrow$ CAR;
$1 \rightarrow$ START , $0 \rightarrow$ OPDMA;
{A questo punto può essere riabilitato il flip-flop che esprime la disponibilità di S a ricevere ulteriori trasferimenti
$0 \rightarrow$ BD;
fi

WRITE X (011) (relativa ad S)

{S gestisce le richieste alle periferiche di I/O col protocollo busy-waiting.
Per semplicità i collegamenti con le periferiche sono stati ridotti alle sole linee di start -output e ready output.
Una volta ricevuto il segnale di OK dall' I/O, S può ripristinare il flip-flop BD a 0 per accogliere ulteriori richieste e inviare l' interrupt al processore M}

```
   1 → START OUTPUT;
0 : if  (READY OUTPUT) = 0 then goto 0
   else
         0→BD ,1→ INTERRUPT I/O ;

   fi
```

READ X (010) (relativa ad S)

{Valgono le stesse considerazioni fatte per la write, con l' unica differenza che S abiliterà la lettura del registro dove l' I/O copia il dato}

```
1 → STAR INPUT,1→ A I/O REGISTER ;
0 :  if (READY INPUT) = 0 then goto 0
    else
         0→ BD,1 → INTERRUPT I/O ;
   fi
```

MUL X (relativa ad S) (100)

{Il registro P/R utilizzato in questa μ-sequenza è descritto assieme alla parte operativa di S, mentre l' algoritmo è quello rappresentato dai diagrammi di flusso del patterson. L'inizializzazione è quella prevista dall'algoritmo e sarà compito di M, una volta che S avrà inviato l'interrupt, prelevare il dato e portarlo sull'accumulatore.

$0 \rightarrow CONT$, $0 \rightarrow P/R_{0\text{-}7}$; { inizializzazione }

0 : if $\beta_{P/R\ 15} = 1$ then

$M/D + P/R_{0\text{-}7}$;

else ∅ ; fi

$P/R \xrightarrow{RSHIFT} P/R$;

if $\beta_{cont} = 0$ then

CONT +1 → CONT , goto 0

else

Una volta terminato il calcolo, S può ripristinare il flip-flop BD a 0 per accogliere ulteriori richieste e inviare l' interrupt al processore M}

$1 \rightarrow$ INTERRUPTs , $0 \rightarrow B_{CALC}$;

fi

DIV X (relativa ad S) (101)

{Valgono le stesse considerazioni fatte per la MUL (vedi diagramma di flusso del patterson)}

$0 \rightarrow CONT$, $0 \rightarrow P/R_{0\text{-}7}$;

0 : $P/R \xrightarrow{LSHIFT} P/R$;

$P/R_{0\text{-}7} - M/D \rightarrow P/R_{0\text{-}7}$;

if $P/R_0 = 0$ then

$P/R \xrightarrow{LSHIFT} P/R, 0 \rightarrow P/R_{0\text{-}15}$;

else

$M/D + P/R_{0\text{-}7} \rightarrow P/R_{0\text{-}7}$;

$P/R \xrightarrow{LSHIFT} P/R, 0 \rightarrow P/R_{0\text{-}15}$;

fi

if $\beta_{cont} = 0$ then

CONT +1 → CONT , goto 0

else

$P/R_{0\text{-}7} \xrightarrow{RSHIFT} P/R_{0\text{-}7}$;

$1 \rightarrow$ INTERRUPTs , $0 \rightarrow B_{CALC}$;

fi