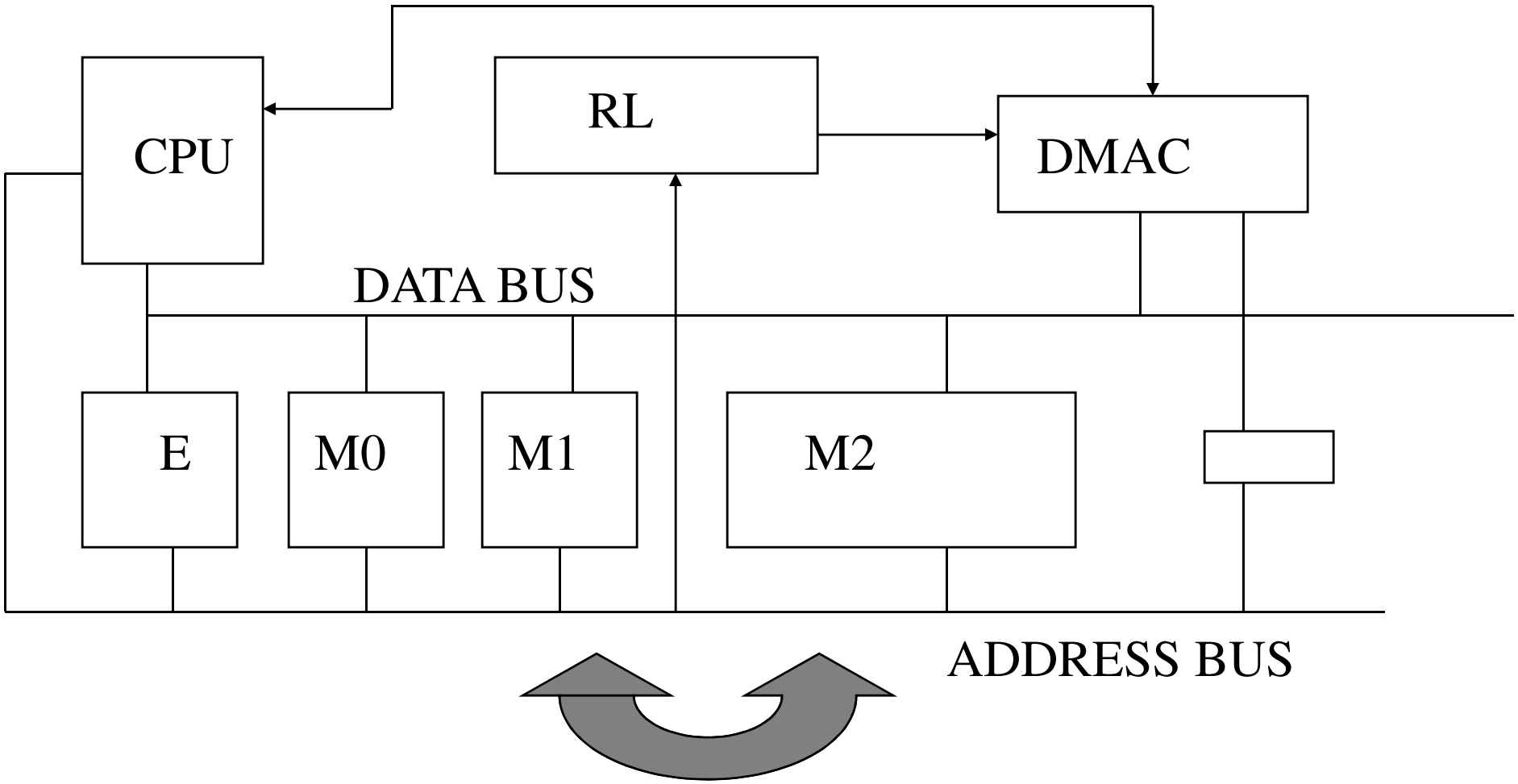
Sistemi di Elaborazione: esame del 24 giugno 1996

Un sistema di elaborazione e' basato su PIC18F8720 a 25MHz con 32k di RAM (banco M0) e 64k di EPROM (banco EPROM), entrambi a 50 nsec di Tacc, agli indirizzi bassi ed alti rispettivamente. Il sistema e' inoltre dotato di un banco da 32k di RAM (M1) a Tacc=100 nsec., posto nello spazio di indirizzamento adiacente a quello di M0. Infine il sistema e' dotato di un a memoria molto lenta da 256 kbyte, chiamata M2 (fisicamente divisa in dispositivi da 64k) a Tacc=800 nsec, posta nello spazio di indirizzamento adiacente a quello di M1. Per Tacc si intende il max. ritardo a partire dalla selezione della cella di memoria (mentre Toe =Tacc/2). Tutte le memorie sono a parallelismo 8 bit.
Si vuole realizzare un sistema di **memoria gerarchica con paginazione** in pagine da 32k. In particolare i dati (non di sistema) sono memorizzati tutti in M2, divisi in pagine da 32k che possono essere temporaneamente rilocate (in modo fisico) nella più veloce M1.
Si progetti il meccanismo hardware di paginazione che deve seguire la seguente politica:
a)Inizialmente M1 e' vuota e al primo accesso a M2 la pagina che contiene il dato richiesto viene copiata in M1.
b)Ad ogni accesso virtualmente in M2 viene confrontato in modo hardware con il registro contenente l'identificatore di pagina (da progettare nel modo più efficiente possibile). Se il dato e' nella pagina contenuta in M1 (caso di *hit*) il dato viene letto e scritto su M1. In caso di *miss* (pagina non contenuta in M1), la pagina deve essere rimpiazzata dalla pagina richiesta con una operazione di scrittura della pagina da M1 a M2 e successivo rimpiazzamento da M2 a M1.
c)Il trasferimento deve avvenire via DMA in modo tale che la CPU acceda sempre e soltanto a M1 (oltre che a M0 e ad EPROM) e solo virtualmente a M2.

1)Si progetti il sistema di elaborazione comprendente il microprocessore, la corretta interfaccia alle memorie ed il DMA e qualsiasi altro dispositivo risulti necessario.
2)Si calcolino i cicli di wait richiesti per le varie memorie e si progetti il sistema di generazione dei ready, il più efficiente possibile.
3)Si valuti in linea di massima quale e' il vantaggio che si ottiene con la paginazione e con che condizioni di località si può sfruttare tale vantaggio

Agenda

- Schema a blocchi del sistema
- Mappatura delle memorie nello spazio di indirizzamento
- Descrizione funzionale del sistema e modalità di interruzione della CPU
- Generazione del segnale di HIT/MISS#
- Schemi collegamenti CPU e DMAC
- Reti logiche per la generazione dei segnali HIT/MISS#, DREQ1 (trasferimento M1→M2) e DREQ2 (trasferimento M2→M1)
- Calcolo degli stati di wait per la CPU in caso di hit
- Gestione degli stati di wait per la CPU in caso di miss
- Calcolo degli stati di wait per il DMAC per trasferimento M1→M2
- Calcolo degli stati di wait per il DMAC per trasferimento M2→M1
- Esempio di generatore di uno stato di wait per il DMAC



Trasferimenti solo via DMAC

RL= Rete Logica di paginazione coordina i trasferimenti in DMAC (in caso di miss), in questo caso la CPU non deve avere l'accesso al bus. Come bloccare la CPU?

ALCUNE NOTE:

CPU, DMA e bus di sistema

CPU a 25 MHz in Byte Select Mode: Progetto dell'interfaccia con le memorie Segnale di ready da progettare per l'accesso a M1 e M2 quando il DMA deve accedere a tali memorie come Master; per il micro il calcolo deve essere per EPROM, M0, M1 ed M2.

Bus di sistema gestito dalla CPU:

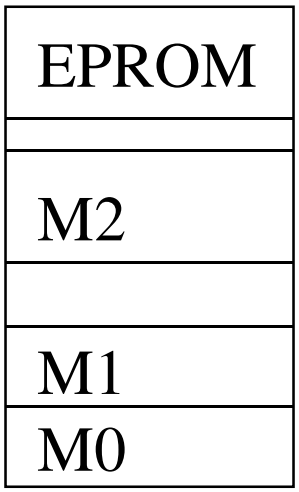
bus degli indirizzi disabilitato in caso di DMA (segnale HOLDA)
bus di dati unico per memoria ed I/O (1 sola periferica, il DMAC) da mappare in spazio di indirizzamento del micro
bus di dati disabilitato in caso di DMA (ENDATA= /OE * /HOLDA e lo stesso per /WR)
bus di controllo disabilitato con HOLDA (su AEN del DMA 8237)

Sistema di memorie

memorie M0,M1, EPROM collegate al bus dati ad 8 bit e al bus di indirizzi
accessibili da parte della CPU (M0, M1 ed EPROM) o dal DMAC (solo M1 ed M2)

4 banchi di M2 a 64 K visibili solo dal DMAC; decodifica da progettare anche in funzione della paginazione. Trasferimento M1-M2 e M2-M1 dal DMAC in fly-by per essere efficienti (solo se trasferimento "allineato") quindi con i segnali di M2 collegati ai segnali di I/O del DMAC.

Definizione dello spazio di indirizzamento



A19.....A0	BA0		
1 1111 1111 1111 1111 1111	A	1FFFFF	64K
1 1111 0000 0000 0000 0000	DA	1F0000	
0 0111 1111 1111 1111 1111	A	07FFFF	256K
0 0100 0000 0000 0000 0000	DA	040000	
0 0000 1111 1111 1111 1111	A	00FFFF	32K
0 0000 1000 0000 0000 0000	DA	008000	
0 0000 0111 1111 1111 1111	A	007FFF	32K
0 0000 0000 0000 0000 0000	DA	000000	

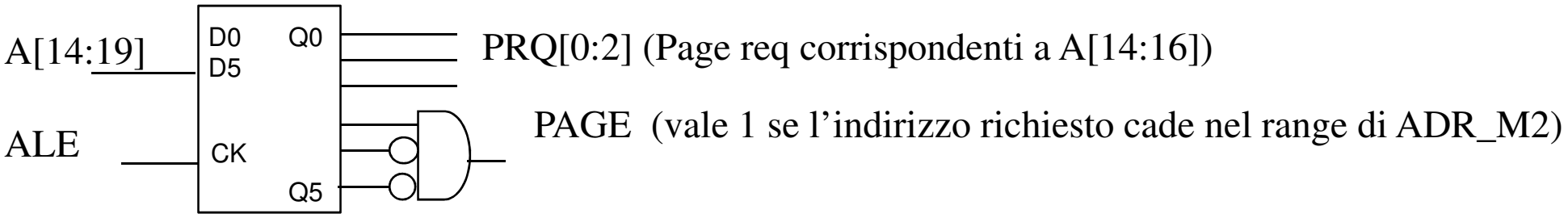
M2 può essere posto " a piacere" nello spazio di indirizzamento rimasto, scelta che semplifica la decodifica

M0, M1, EPROM visibili dalla CPU;
gli accessi della CPU ad M2 vengono rediretti su M1, se la pagina e' in M1 (hit=successo), altrimenti (miss=insuccesso) scatta una richiesta di DMA che esegue l'operazione di write-back della pagina di M1 e di rimpiazzamento della pagina richiesta.

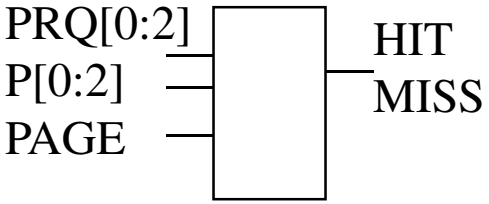
ADR_M0= /A17 * /A14
ADR_M1= /A17 *A14
ADR_M2=/A19*A17 (ma il micro non deve accedere fisicamente a M2 ma alla sua immagine su M1)
ADR_EPROM=A19

Gestione della paginazione

M2 di 256Kbyte contiene 8 pagine da 32k selezionabili con gli indirizzi A[14:16] Perciò si può prevedere una rete che indica la pagina richiesta campionando tali indirizzi quando sono validi.



Se si suppone la presenza di un registro (373) che fornisca l'indirizzo della pagina presente in M1 (Page_id register), con il uscita il bus P[0:2], un comparatore combinatorio indica *hit o miss*




```
if ((PRQ == P) && (PAGE==1))
{           HIT = 1; MISS = 0;    }
else
{           HIT = 0; MISS = 1;    }
```

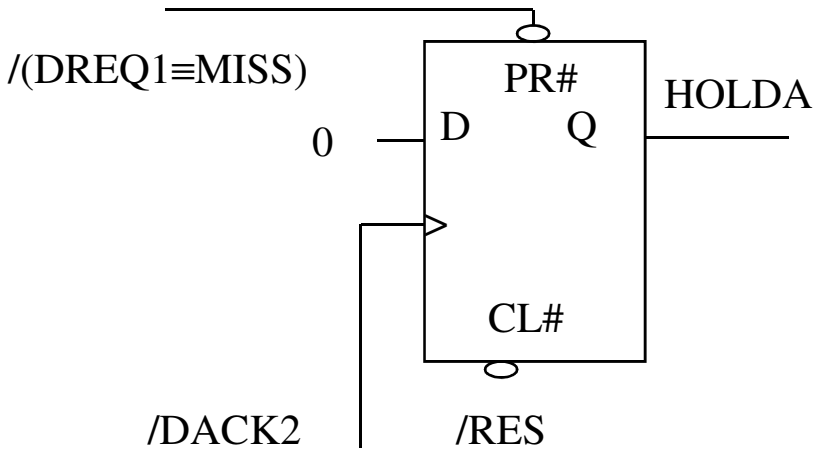
In caso di *hit* l'accesso avviene regolarmente in M1 (e non in M2!).

In caso di *MISS:*

1) come deve essere interrotta la CPU?
(wait, HOLD, intr...) converrebbe in questo caso bloccare il ciclo con un segnale di wait per non perdere il ciclo in corso che non può essere eseguito. In questo modo la paginazione risulta totalmente trasparente alla CPU: non è possibile con questo micro, il primo dato a cui accede ad ogni cambio di area deve essere un dummy data che via software viene buttato.
2) Trasferimenti: prima la pagina presente in M1 viene ricopiata in M2 nell'area di memoria da cui proveniva (potrebbe contenere dei dati modificati che così non vanno persi)
Come gestire il trasferimento?
Via DMAC in block mode; in fly-by è possibile se M2 è collegata come se fosse una porta di I/O (e se gli indirizzi sono identici).
3) poi la pagina di M2 richiesta dall'indirizzo emesso dalla CPU deve essere copiata in M1.
Come sopra, ma nella direzione opposta (è un altro programma di canale nel DMAC). Il Page_id register deve essere aggiornato.

Il segnale di HOLDA non è presente spontaneamente, deve essere costruito .

In un'interfaccia ordinaria il segnale HOLDA è generato dal micro a fronte di un HOLD del DMAC, al termine dell'istruzione corrente (e quindi implicitamente anche al termine del ciclo di bus eventualmente generato dall'istruzione per accedere a operandi).
In questo esercizio, invece, il ciclo di DMA si inserisce *durante* l'istruzione corrente → il segnale di HOLDA deve essere generato artificialmente. Una rete logica che può svolgere questo compito è la seguente (indicata schematicamente come RL1 nella slide con il DMAC):



Si assuma che il trasferimento M1→M2 sia assegnato al canale 1 del DMAC e quello M2→M1 al canale 2.

•All'arrivo del segnale DREQ1 (cioè MISS, che indica la richiesta di DMA), il flip-flop **attiva** l'HOLDA (tramite l'ingresso di Preset asincrono, attivo in logica negata), quindi simula l'azione della CPU.

•Fino a quando deve persistere HOLDA?

Fino al completamento del secondo ciclo di DMA, quello che copia la pagina di M2 su M1. Un segnale di terminazione adatto è il fronte di discesa (trailing edge) del segnale DACK2. Verifichiamo questa temporizzazione: nei manuali del DMAC, HRQ viene disattivato TDQ dopo il fronte di salita del clock in S4 (120 ns a 4 MHz, manuali Intel); In questo micro tutto deve essere costruito con segnali di interrupt per cui non abbiamo problemi di temporizzazioni (a parte l'HOLDA che deve essere compatibile con le tempistiche del DMA, l'HOLD viene recepito dal micro solo per buttare via il primo dato e viene portato su un pin di interrupt.

Il tempo per cui deve persistere HOLDA dopo il fronte di salita del clock in S4 è, insomma, implicito e non precisamente definito (vedi manuali Intersil), perché si assume comunque un minimo di ritardo da parte del processore.

DACK2 si disattiva TAK (220 ns max, minimo non specificato) dopo il fronte di **discesa** del clock in S4 (quindi almeno TCH = 100ns a 4 MHz dopo quello di salita) ; in questo momento, l'ultimo trasferimento è terminato, e non dovrebbe essere critico **resettare** HOLDA contestualmente.

Quindi in caso di *miss*: *write back + page fill*

1) la CPU non può terminare il ciclo e deve flushare il primo dato quando vede un cambio di area di lettura dei dati su memoria M2 esterna (lo si può verificare anche con l'info ATTESA=1*)* ;
Il bus di sistema deve essere disabilitato agendo sull'/OE dei 245 e 373
in questo caso si può disabilitare il bus con il segnale di HOLDA che comunque sarà presente durante il controllo del bus da parte del DMAC

2) richiesta di trasferimento via DMA (DREQ1) da M1 a M2 (*operazione di write back*);
DREQ1 viene generato dalla presenza di MISS (equivalente a /HIT).
il DMAC genera HOLD collegato direttamente ad HOLDA, trasferisce in fly by da M1 a M2 un blocco di 32 Kbyte allineato (guidando direttamente gli indirizzi BA[0:14])
Il CSM1# e' abilitato da HOLDA
il chip select del banco M2 selezionato e ADR14_M2 (sono banchi da 64 K)
derivano dalla decodifica di P[0:2] che indicano quale pagina di M2 era in M1.

3) termine del trasferimento (con EOP) si richiede il trasferimento da M2 a M1 via DMAC (DREQ2) (*operazione di page-fill*); il registro P[0:2] campiona quale pagina ora sarà presente in M1 (ossia campiona il valore di PRQ[0:2]). Il trasferimento avviene dal banco di M2 selezionato da P[0:2] a M1 abilitato da HOLDA.

4) termine del trasferimento: la presenza della pagina giusta in M1 riabilita il segnale di HIT (Page_id register contiene ora il valore richiesto e HIT vale 1).
La CPU si "sveglia" dallo stato di wait (ATTESA=0)

ATTESA= (/HIT)+ HOLDA

Segnali combinatori e decodifica degli indirizzi:

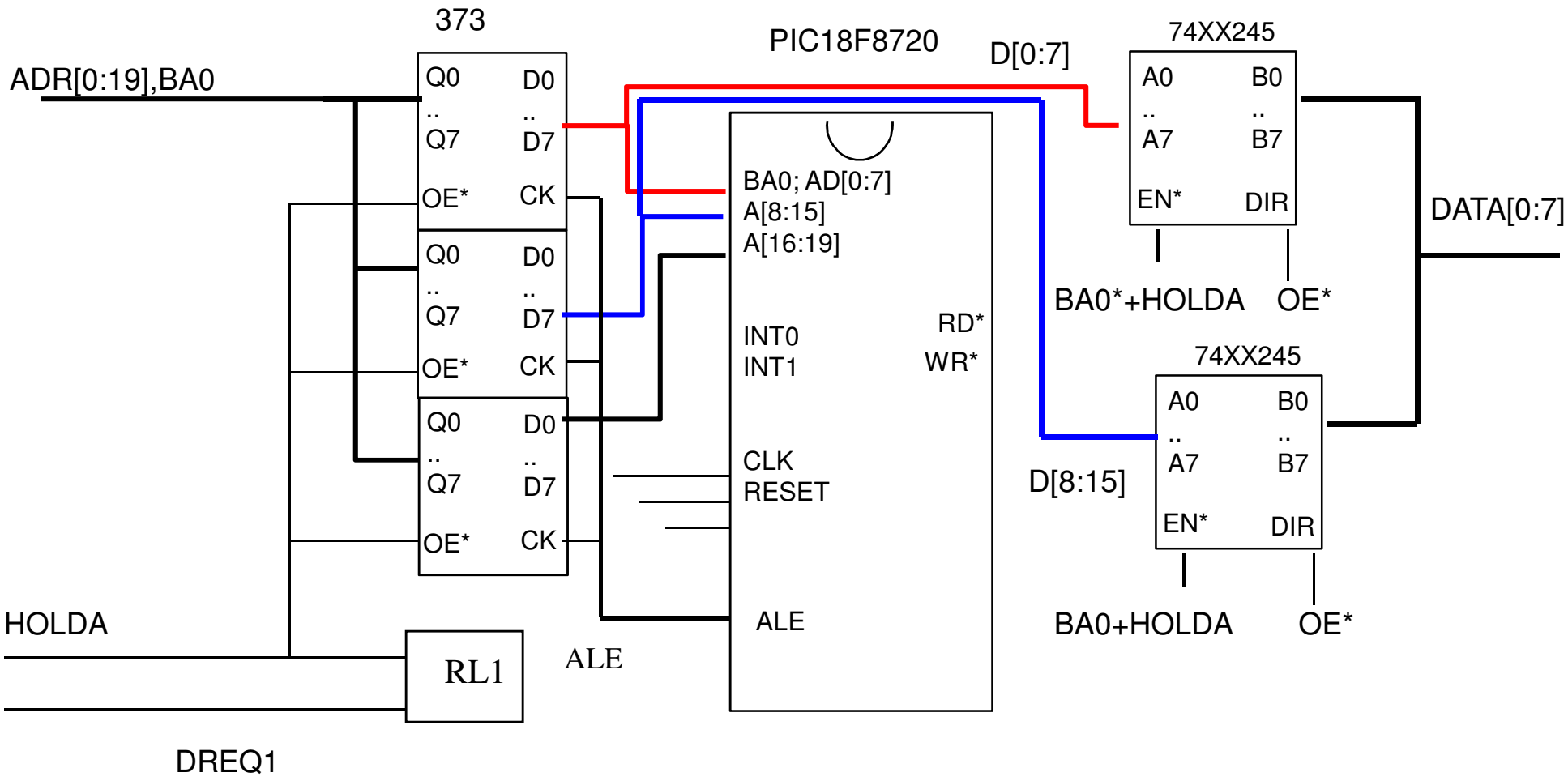
Nota bene: nei cicli di cui è master la CPU sono usati i segnali di stato bufferati con 373 come per gli indirizzi, o con 244 come per i segnali di controllo o con 245 come per i dati

CSE# = / (ADR_EPROM*(/HOLDA))
(solo Code Access e Read Memory sotto CPU)
CSM0# = /(ADR_M0 */HOLDA)
CSM1# = /((ADR_M1+ADR_M2*HIT) * /HOLDA + HOLDA)

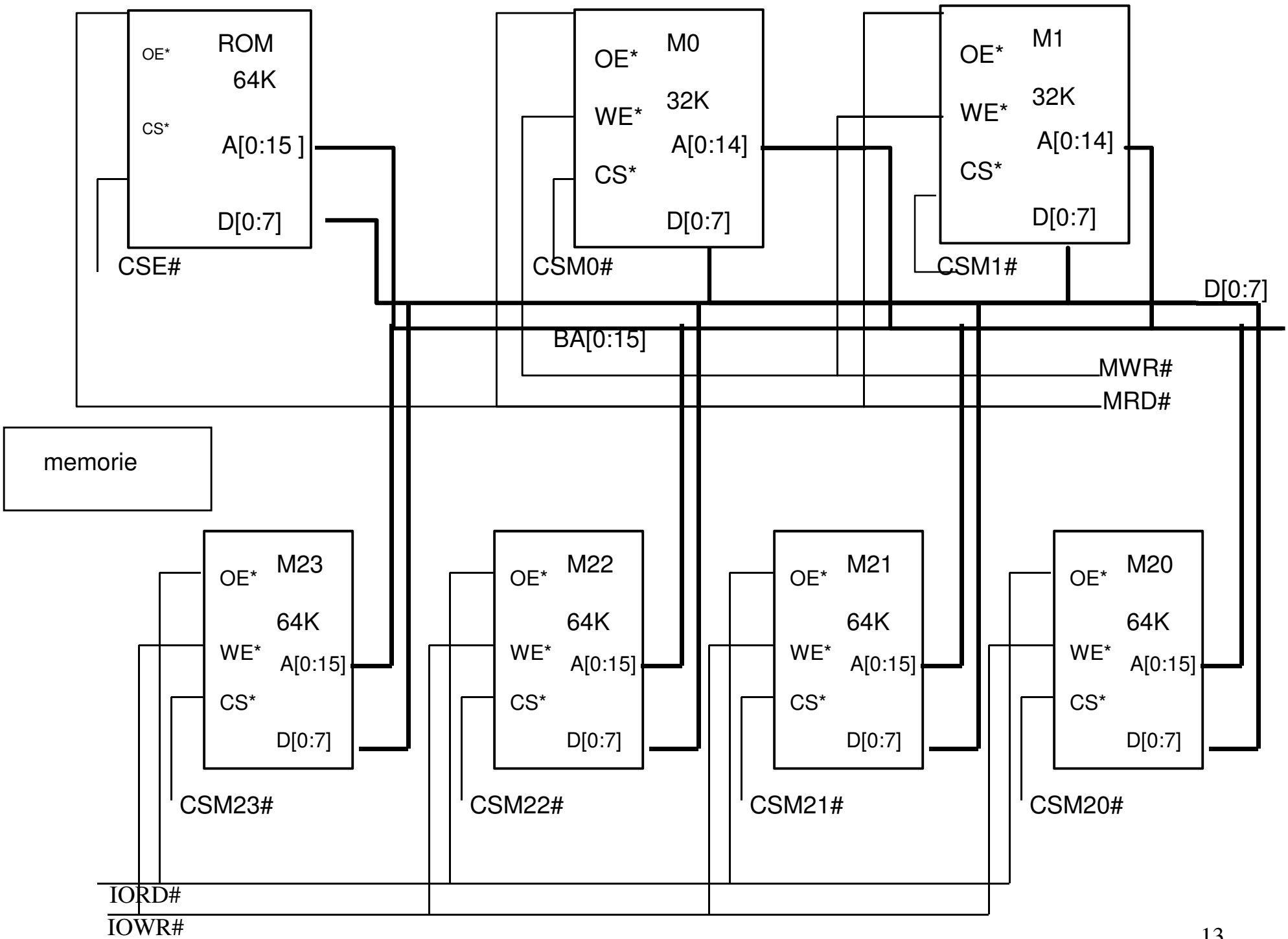
CSM20#=/(/P2 * /P1 *HOLDA) (bit P2 e P1 del registro P[0:2] addr 15 e 16 identificano blocchi
CSM21#=/(/P2 * P1 *HOLDA) da 64K)
CSM22#=/(P2 * /P1 *HOLDA)
CSM23#=/(P2 * P1 *HOLDA)
ADR14_M2 =P0 per l'area dei singoli chip di M2 da abilitare per il trasferimento (non generato da DMA)

CSDMA#= /(/BA19*BA18* /HOLDA) (Addr_base=80000h)
(stato INTA ignorato)

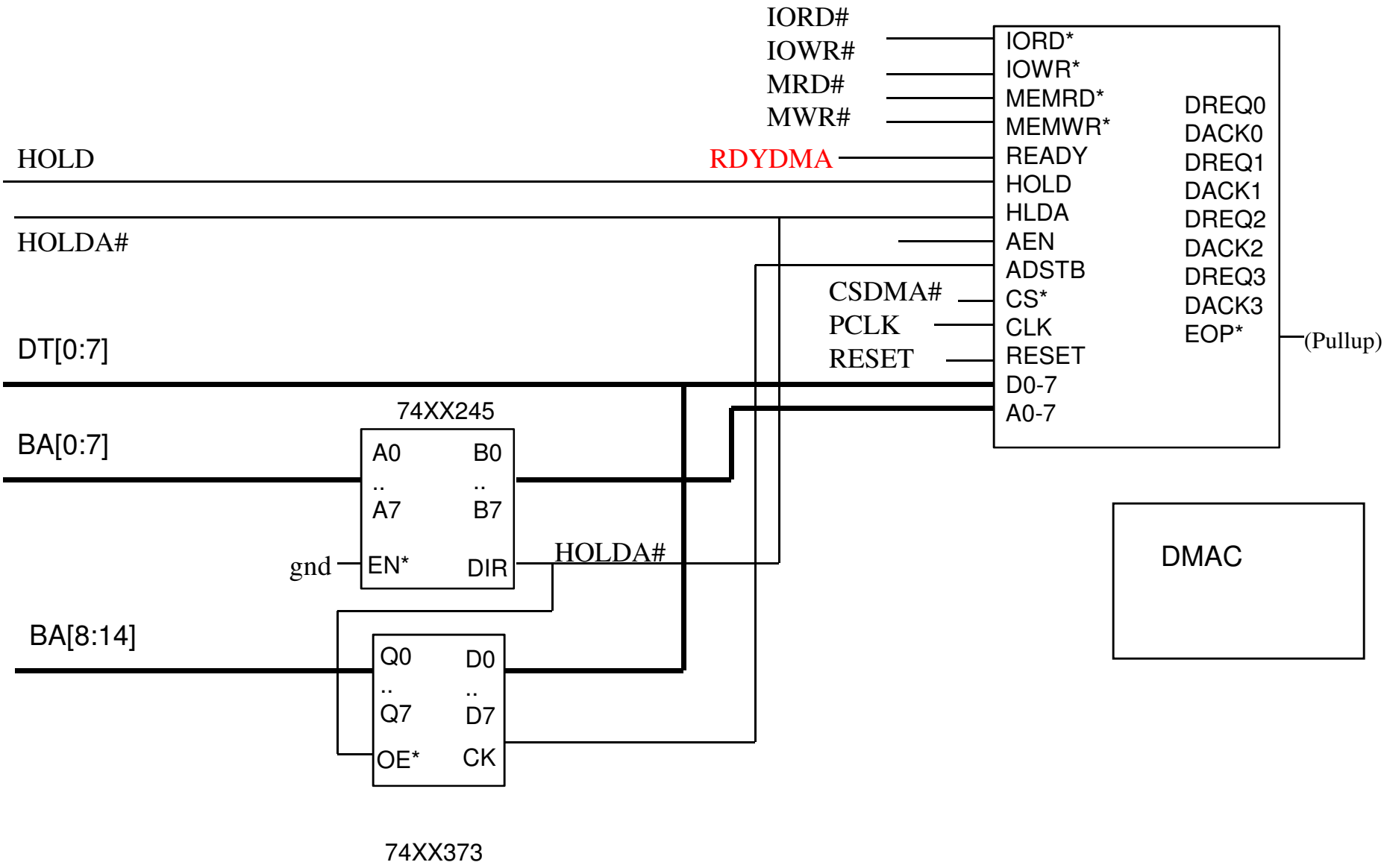
ATTESA= (/HIT)+ HOLDA



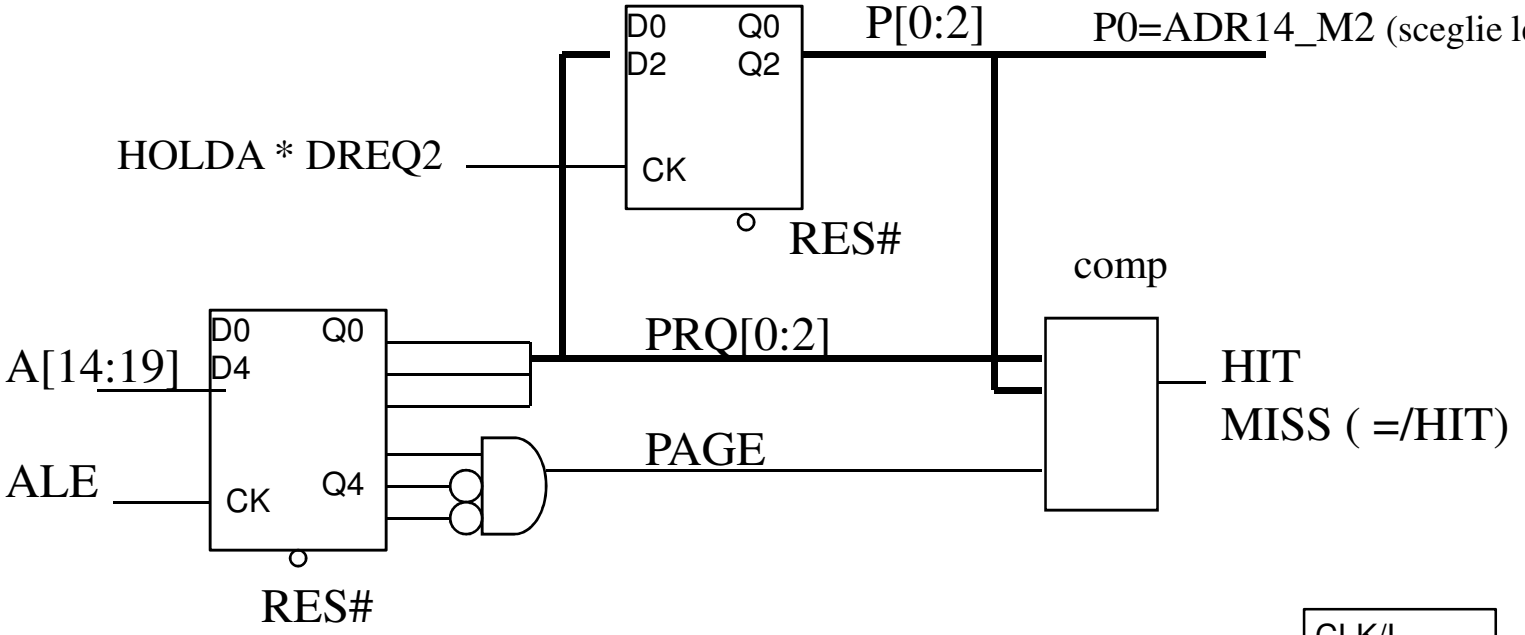
processore
e bus

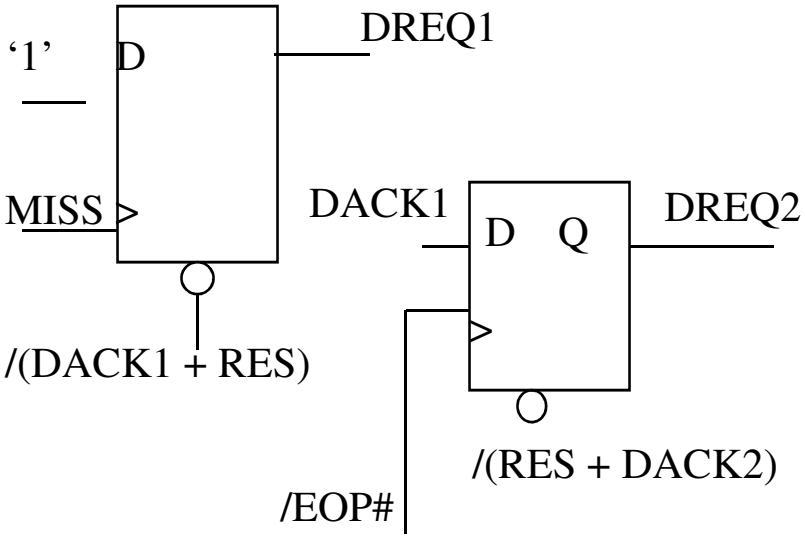


N.B. M2 mappata come IO

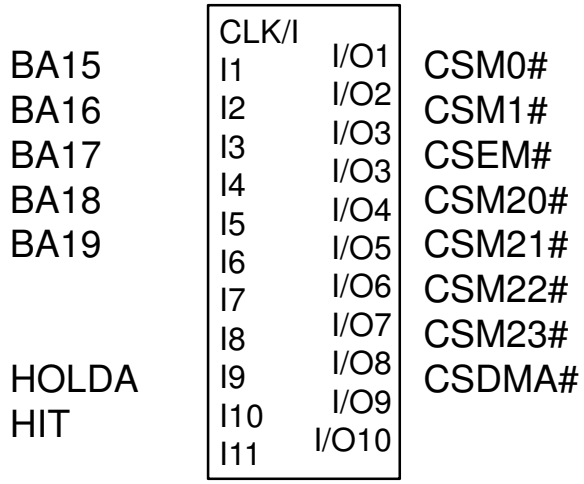

IORD#
IOWR#
MRD#
MWR#
RDYDMA
HOLD
HOLDA#
DT[0:7]
BA[0:7]
BA[8:14]
CSDMA#
PCLK
RESET
IORD*
IOWR*
MEMRD*
MEMWR*
READY
HOLD
HLDA
AEN
ADSTB
CS*
CLK
RESET
D0-7
A0-7
DREQ0
DACK0
DREQ1
DACK1
DREQ2
DACK2
DREQ3
DACK3
EOP*
(Pullup)
DMAC
74XX245
A0
..
A7
B0
..
B7
gnd
EN*
DIR
HOLDA#
Q0
..
Q7
D0
..
D7
OE*
CK
74XX373

Page_id register (aggiornamento alla fine del write back)





(campiona DACK1 sul fronte di discesa di EOP#, sufficiente per generare DREQ2 senza rilasciare il bus dopo il primo trasferimento)



Reti logiche specifiche

Gestione degli stati di wait

Facendo riferimento al manuale del micro a 25MHz, si può calcolare che M0 ed EPROM da 50 ns (e i controllori di I/O) non necessitano di stati di wait.
Calcoliamo per M1 (100 ns Tacc, TOE=Tacc/2) nel caso più stringente di una lettura:

$$\begin{aligned}\text{TDATAMEM} = \max\{ & (60\ (t_{\text{address valid da inizio ciclo}}) + 100\ (t_{aa/acc}) + 10\ (t_{373}) + 10\ (t_{245})), \\ & (40\ (t_{\text{ce da inizio ciclo}}) + 100\ (t_{ce/acc}) + 10\ (t_{245})), \\ & (80\ (t_{\text{oe da inizio ciclo}}) + 50\ (t_{oe}) + 10\ (t_{245}))\} \\ & = \text{MAX}\{180,150,140\} = 180\text{ns}\end{aligned}$$

(rispettivamente per la propagazione degli indirizzi, del chip select, del comando di read, calcolati a partire dall'inizio del ciclo di bus, nella ipotesi che le reti semplici (373, 245, DEC) siano tutte da 10 ns)
Senza Twait, a disposizione si hanno 4 TCK = 160 ns < 180 ns.
Quindi M1 ha bisogno di 1 wait cycle nel caso in cui si presenti una *hit*.
Ovvero tutte le volte in cui debba lavorare su M1 sia in lettura che in scrittura. (infatti il TW si definisce una sola volta per tutti……..

Ancora stati di wait

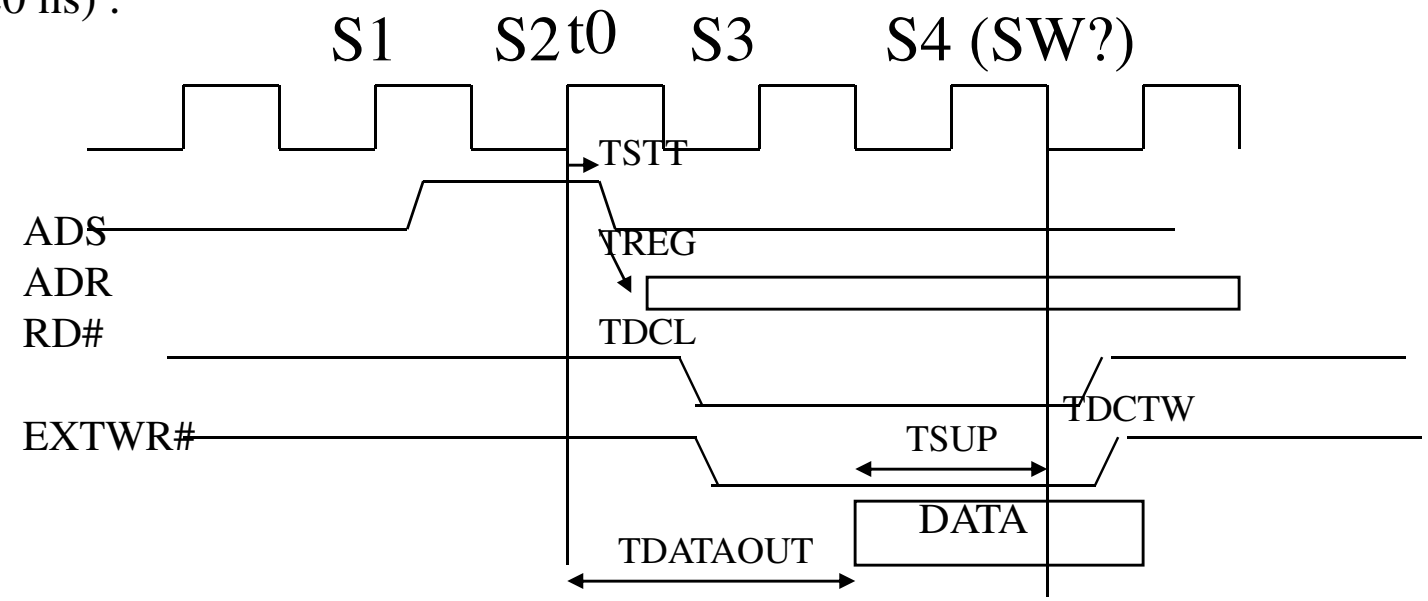
in caso di *MISS*: il micro dovrà "fare altre cose" senza accedere al bus.
La rete che genera ATTESA (pag. 12) prende in ingresso gli indirizzi ed è composta da tre blocchi semplici in cascata.

1ª osservazione: dato che si ha sempre uno stato di wait nel caso di accesso ad M1, le temporizzazioni per il calcolo di ATTESA che è il segnale che può essere usato per avvertire il micro di flushare il dato acquisito, sarebbero molto più rilassate (ci sono 4 clock in più per generarlo).

2ª osservazione: quando il DMAC restituisce il controllo alla CPU resettando ATTESA, la CPU riceve un interrupt, e può accedere al bus;

WAIT DEL DMA:

Il DMAC (ipotesi 8237A-4) lavora a 4Mhz (250 ns ciclo) e campiona il segnale di ready (RDYDMA) al fronte di discesa di S3 e necessita di un tempo di setup TRS di 60 ns (e di un hold TRH di 20 ns) .



Ipotesi (restrittiva): consideriamo gli indirizzi validi dal fronte di discesa del segnale ADSTB.
Con la temporizzazione a ciclo di 250 ns TSTT= ~110 ns, TDCL=200 ns

M1 a M2: da T0 i dati sono sul bus dati dopo un tempo max{(TSTT+TREG+Tacc1),(TDCL+TOE1)} (n.b.: il CS# delle memorie è sempre attivo e la sua temporizzazione è quindi meno stringente degli indirizzi), cioè max{110+10+100, 200+100}, cioè TDATAOUT=300 ns nel caso peggiore.
A questo punto i dati sono disponibili sul bus per essere incamerati da M2; senza Twait,
è permesso un Tsetup = 2 TCK + TCH - TDATAOUT = 500+100-300= 300 ns.
se M2 ha 800 ns di Tacc, prendiamo come Tsetup2 (o TWR2) la metà, cioè 400 ns.
Necessita di 1 stato di wait: (300 + 250) > 400!

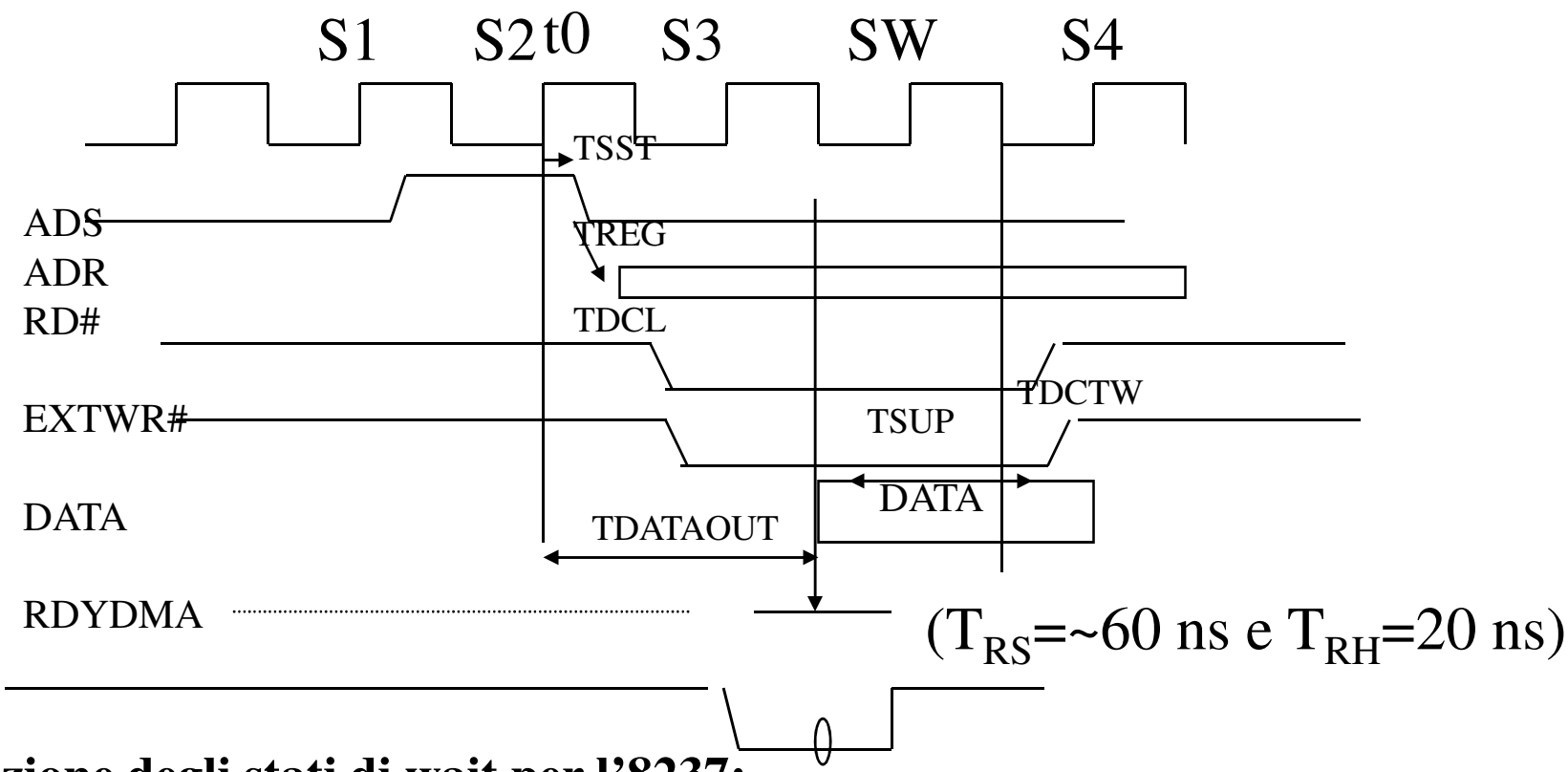
WAIT DEL DMA:

Quindi, abbiamo stimato che i dati siano incamerati da M2 in un tempo totale pari a:
TSTT+TREG+Tacc1+ TWR2 = 110 + 10 + 100 + 400 = 620 ns
In realtà, non si è verificato se non è stata rispettata la temporizzazione sugli indirizzi per M2:
TSTT+TREG+Tacc2 = 110 + 10 + 800 = 920 ns; anche questa temporizzazione va rispettata per M2, ed è evidentemente la più stringente:
TSTT+TREG+Tacc2 < 2 TCK + TCH + N TCK → 920 < 600 + N 250 → N = 2!

Sono necessari due stati di wait.

M2 a M1: da T0 i dati sono sul bus dati dopo un tempo max{(TSTT+TREG+Tacc2),(TDCL+TOE2)} cioè {(110+10+800),(200+400)} = 920 ns. A questo punto i dati sono pronti per essere incamerati da M1 in un tempo pari a TW1 = 100 ns. Il tempo totale è quindi:
TSTT+TREG+Tacc2+ TWR1 = 1020 ns, che deve essere < 600 + N 250 → N = 2

Anche qui va verificato se la temporizzazione sugli indirizzi per M1 è rispettata:
TSTT+TREG+Tacc1 = 110 + 10 + 100 = 220, largamente meno stringente della precedente. Quindi il calcolo degli stati di wait è corretto.



Generazione degli stati di wait per l'8237:

RD e Extended WR sono pronti dopo al max TDCL (200 ns) dal fronte di salita di S2.
Ciò significa che (a 4 MHz) sono attivi almeno 50 ns prima del fronte di salita di S3
Perciò, generando il ready con il fronte di salita di S3, abilitato dalla presenza
del READ o del WRITE (e ovviamente di HOLDA), TRS e TRH sono rispettati.

Esempio di generatore di uno stato di wait:

(/IORD#+/MEMRD#)HOLDA* PCLK