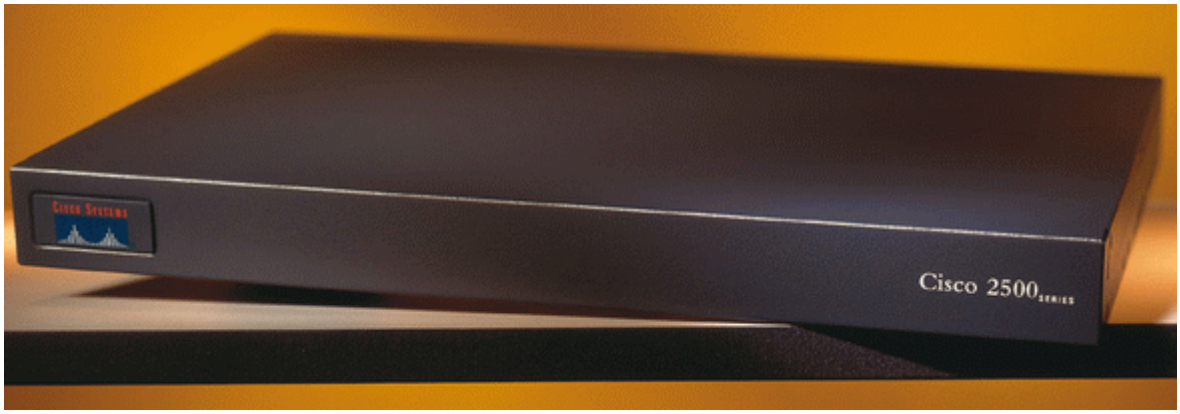


Il primo esemplare di questa famiglia di router è stato commercializzato agli inizi degli anni ’90.
La serie di router Cisco 2500, è costituita da molteplici modelli caratterizzati da una configurazione fissa con almeno due delle seguenti interfacce: Ethernet, TokenRing, seriale sincrona, seriale asincrona e BRI ISDN. Due modelli della serie sono di tipo modulare.
Le caratteristiche dichiarate di questa famiglia di router sono: un processore MC68030 a 20MHz, 8MB di Flash RAM espandibile fino a 16MB e 4MB di Ram espandibili fino a 16MB.

Nome	Porte LAN	Porte WAN
2501	1 porta Ethernet DB-9 AUI	2 porte sync seriali
2502	1 porta Token Ring DB-9	2 porte sync seriali
2503	1 porta Ethernet DB-15 AUI	2 porte sync seriali
		1 porta ISDN BRI (S/T)
2504	1 porta Token Ring DB-9	2 porte sync seriali
		1 porta ISDN BRI (S/T)
2505	8 porte Ethernet hub	2 porte sync seriali
2507	16 porte Ethernet hub	2 porte sync seriali
2509	1 porta Ethernet DB-15 AUI	2 porte sync seriali
		8 porte async
2511	1 porta Ethernet DB-15 AUI	2 porte sync seriali
		16 porte async
2512	1 porta Token Ring DB-9	2 porte sync seriali
2513	1 porta Ethernet DB-15 AUI	2 porte sync seriali
	1 porta Token Ring DB-9	
2514	2 porte Ethernet DB-15 AUI	2 porte sync seriali
2515	2 porte Token Ring DB-9	2 porte sync seriali
2516	14 porte Ethernet hub	2 porte sync seriali
		1 porta ISDN BRI (S/T)
2520	1 porta Ethernet RJ-45	2 porte high-speed sync seriali
		1 porta ISDN BRI (S/T)
		2 porte low-speed sync/async seriali
2521	1 porta Token Ring DB-9	2 porte high-speed sync seriali
		1 porta ISDN BRI (S/T)
		2 porte low-speed sync/async seriali
2522	1 porta Ethernet DB-25 AUI	2 porte sync seriali
		1 porta ISDN BRI (S/T)
		8 porte sync/async seriali
2523	1 porta Token Ring DB-9	2 porte sync seriali
		1 porta ISDN BRI (S/T)
		8 porte sync/async seriali
2524	1 Porta Ethernet RJ45 o DB-25 AUI	3 free slot
2525	1 Porta Token Ring RJ45 o DB-9	3 free slot

Estratto di rom, qui c'è l'inizializzazione di un po' di hardware

```
0100673A  memory_chkdim:  clr.l   d2              Inizializza il contatore di pagine.
0100673C                  clr.l   d1              Inizializza il puntatore alla pagina.
01006740  loop:           movea.l d1,a4
01006742                  move.l  d1,(a4)         Scrivi l'indirizzo ad inizio pagina
01006744                  tst.l   (a0)            L'indirizzo nella pagina0=0?
01006746                  bne.s   exit            Salta se cambia
01006748                  addq.l  #1,d2           Incrementa contatore di pagine
0100674A                  addi.l  #$80000,d1      Passa alla prossima pagina
01006750                  cmpi.l  #$FFFFFF,d1     È l'ultima pagina
01006756                  bls.s   loop            Se no, continua a ciclare
```

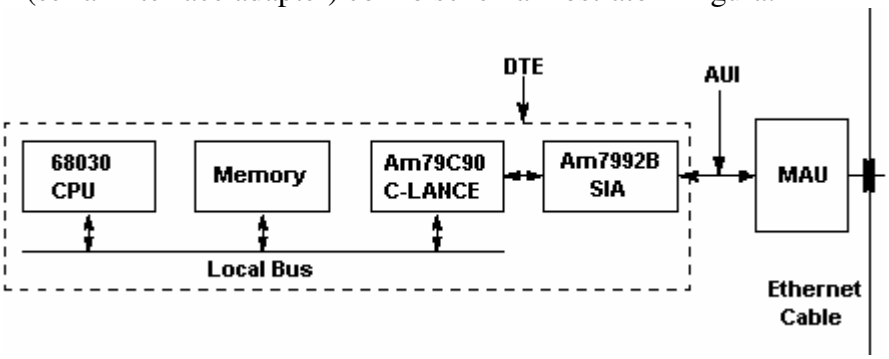
```
01000134  Starts_here:    move.l  #$3E8,d0        Inizializza il contatore a 1000
0100013A  loc_100013A:    subq.l  #1,d0           Countdown
0100013C                  bgt.s   loc_100013A     Ripeti se non finito
0100013E                  movea.l #0,a6           Inizializza il bp dello stack
01000144                  lea     $1000008,a0     Carica in a0 il VB addr della Rom
01000148                  movec   a0,vbr          Setup vb in rom
```

```
0100622E                  cmpi.w #$1315,($2000000).l  Verifica che la nvram è ok
01006236                  bne.s   no_nvram        Salta se nvram non OK
01006238                  move.w  ($2000002).l,d5 Acquisisci il CONFREG
0100623E                  bra.s   exit            Esci
01006240  no_nvram:       move.w  #$101,d5        Assumi il valore di default
01006244  exit:
```

```
01006434  Loop:           move.b  ($2120101).l,d0 Leggi SR del 2681
0100643A                  btst    #2,d0           È disponibile il dispositivo
0100643E                  beq.s   Loop            Se no, attendi
01006440                  move.b  (a0)+,($2120103).l  Manda in output un carattere
01006446                  tst.b   (a0)            La stringa è terminata?
01006448                  bne.s   Loop            Ripeti l'operazione
```

Il sotto sistema ethernet di un Cisco 2500.

Il sotto sistema ethernet di un Cisco 2500 è basato su due devices: AMD Am79C90 (LANCE) ed AMD Am7992B (serial interface adapter) con lo schema mostrato in figura.



Il dispositivo principale del sottosistema è il LANCE device, che è l'unico visibile dalla CPU.
Il dispositivo Am79C90 C-LANCE è programmabile tramite una combinazione di registrie strutture di dati presenti all' interno del dispositivi stesso. Ci sono quattro registri di controllo CSR (control status registers) che sono programmati dalla CPU direttamente. Gli altri registri vengono programmati posizionando nella memoria i rispettivi valori, e lasciando che il controller DMA del dispositivo li carichi nei rispettivi registri.

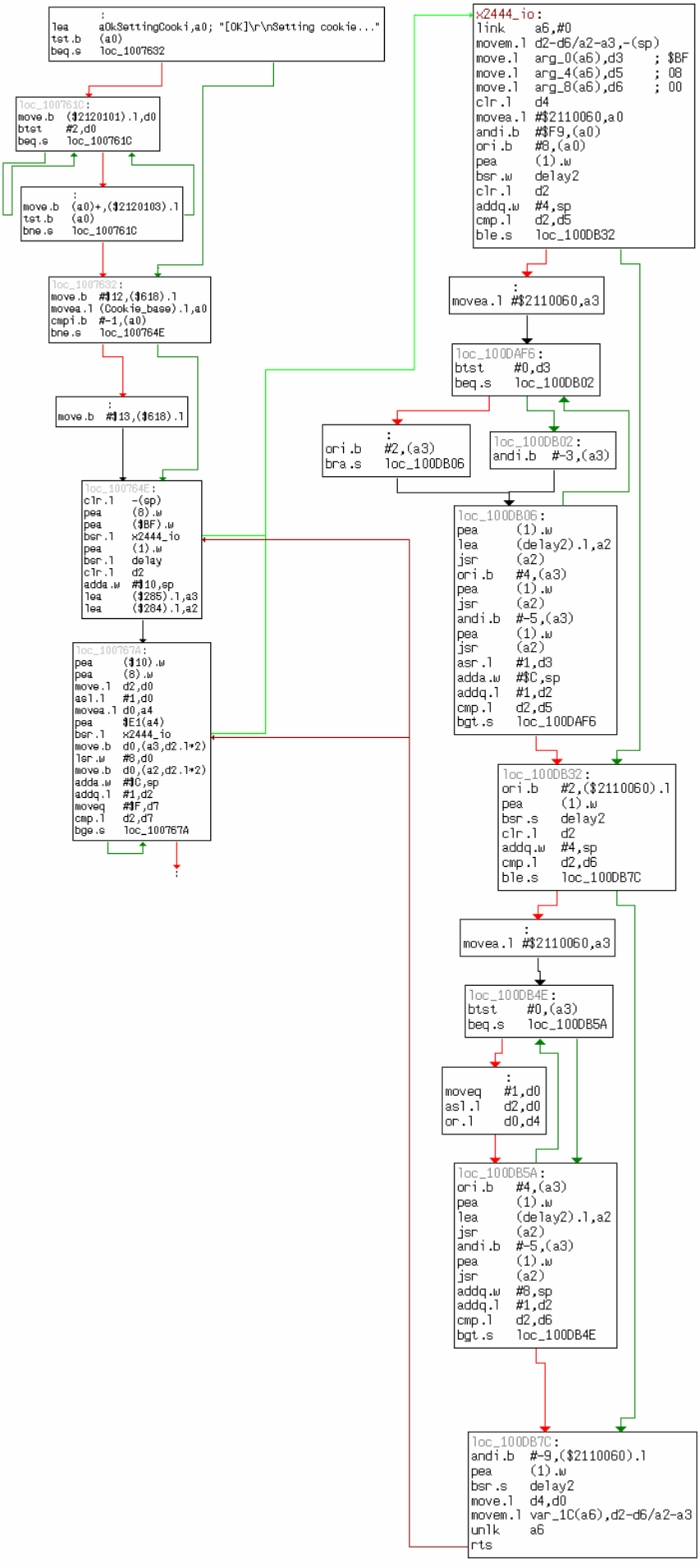
Il dispositivo Am79C90 ha quindi l'abilità di gestire I trasferimenti di memoria senza l'ausilio del processore. Il dispositivo assolve il suo compito di controller ethernet mediante l'ausilio di tre strutture dati nella memoria centrale.

- Un "initializzation block" che consiste in 12 word di memoria centrale contigua in cui sono specificati i dettagli dell' operazione che deve svolgere.
- Un receive ring ed un trasmit ring. Due strutture ad anello lunge 4 word per i dati in ingresso ed i dati in uscita. Ogni elemento della struttura contiene l'indirizzo del buffer la sua lunghezza ed il suo stato.
- Una zona di buffer dove il dispositivo deposita i pacchetti in arrivo e la CPU quelli in partenza

Il cookie è noto, nella letteratura disponibile, come un dispositivo di memoria permanente conte-nente dati specifici della macchina, come ad esempio l'indirizzo MAC della porta Ehternet.
Le note ufficiali riportano il cookie come dispositivo mappato in memoria a partire dall' indirizzo 0x2110040 per 32 byte.
Nella macchina esaminata, che pur comprende un dispositivo a quegli indirizzi, l'analisi dei dati rivela che in a quegli indirizzi non corrispondono dati significativi. L'analisi del codice del resto, sembra prevedere il fatto che possano esserci casi in cui i dati presenti in quegli indirizzi, possano essere non considerati..

```
01006124                movea.l (Cookie_base).l,a0
0100612A                cmpi.b  #-$F,(a0)
0100612E                beq.s   cookie_chk1             ; Salta senza elaborare dati
01006130                cmpi.b  #-1,(a0)
01006134                beq.s   cookie_chk1             ; Salta senza elaborare dati
                        […]
01006152    cookie_chk1:
```

Il codice estratto dalla ROM mostra chiaramente che sono previsti almeno 3 tipi di cookie differenti: quelli in cui il primo byte (cookie_base) ha valore 0xFF, quelli il cui valore è 0xF1, e quelli per cui il primo valore non corrisponde ai precedenti due valori. Nell'apparato esaminato in particolare, il primo byte corrisponde, cosi come tutti gli altri byte, al valore 0xFF. È ragionevole quindi assumere che almeno nel modello dell' apparato esaminato questa zona di memoria non contenga alcun dato significativo, e che se i dati sono effettivamente presenti sono da ricercare altrove.
È da notare comunque che durante la fase di inizializzazione, l'apparato, se il bit 0xF del registro di configurazione è a 1, cioè se è attiva la modalità di boot verbose, stampa il messaggio "Setting cookie..." che suggerisce che anche se l'analisi del cookie così come lo descrive la letteratura ha dato esito negativo, ci debba comunque essere qualcosa che ha a che fare con il cookie in quella zona del codice. Di seguito riportiamo un estratto del codice in quella zona.


:
lea aOkSettingCooki,a0; "[OK]\r\nSetting cookie..."
tst.b (a0)
beq.s loc_1007632
loc_100761C:
move.b ($2120101).l,d0
btst #2,d0
beq.s loc_100761C
:
move.b (a0)+,($2120103).l
tst.b (a0)
bne.s loc_100761C
loc_1007632:
move.b #$12,($1618).l
movea.l (Cookie_base).l,a0
cmpi.b #-1,(a0)
bne.s loc_100764E
:
move.b #$13,($1618).l
loc_100764E:
clr.l -(sp)
pea (8).w
pea ($BF).w
bsr.l x2444_io
pea (1).w
bsr.l delay
clr.l d2
adda.w #$10,sp
lea ($285).l,a3
lea ($284).l,a2
loc_1007674:
pea ($10).w
pea (8).w
move.l d2,d0
asl.l #1,d0
movea.l d0,a4
pea $E1(a4)
bsr.l x2444_io
move.b d0,(a3,d2.l*2)
lsr.w #8,d0
move.b d0,(a2,d2.l*2)
adda.w #$C,sp
addq.l #1,d2
moveq #$F,d7
cmp.l d2,d7
bge.s loc_100767A
:
x2444_io:
link a6,#0
movem.l d2-d6/a2-a3,-(sp)
move.l arg_0(a6),d3 ; $BF
move.l arg_4(a6),d5 ; 08
move.l arg_8(a6),d6 ; 00
clr.l d4
movea.l #$2110060,a0
andi.b #$F9,(a0)
ori.b #8,(a0)
pea (1).w
bsr.w delay2
clr.l d2
addq.w #4,sp
cmp.l d2,d5
ble.s loc_100DB32
:
movea.l #$2110060,a3
loc_100DAF6:
btst #0,d3
beq.s loc_100DB02
:
ori.b #2,(a3)
bra.s loc_100DB06
loc_100DB02:
andi.b #-3,(a3)
loc_100DB06:
pea (1).w
lea (delay2).l,a2
jsr (a2)
ori.b #4,(a3)
pea (1).w
jsr (a2)
andi.b #-5,(a3)
pea (1).w
jsr (a2)
asr.l #1,d3
adda.w #$C,sp
addq.l #1,d2
cmp.l d2,d5
bgt.s loc_100DAF6
loc_100DB32:
ori.b #2,($2110060).l
pea (1).w
bsr.s delay2
clr.l d2
addq.w #4,sp
cmp.l d2,d6
ble.s loc_100DB7C
:
movea.l #$2110060,a3
loc_100DB4E:
btst #0,(a3)
beq.s loc_100DB5A
:
moveq #1,d0
asl.l d2,d0
or.l d0,d4
loc_100DB5A:
ori.b #4,(a3)
pea (1).w
lea (delay2).l,a2
jsr (a2)
andi.b #-5,(a3)
pea (1).w
jsr (a2)
addq.w #8,sp
addq.l #1,d2
cmp.l d2,d6
bgt.s loc_100DB4E
loc_100DB7C:
andi.b #-9,($2110060).l
pea (1).w
bsr.s delay2
move.l d4,d0
movem.l var_1C(a6),d2-d6/a2-a3
unlk a6
rts

L’analisi di questo codice mostra chiaramente l’utilizzo di una procedura che serializza i dati e li manda ad un dispositivo. Il dispositivo in questione non è citato su nessun documento ufficiale, ma è comunque presente sull’ apparato. Un ipotesi coerente con il codice mostrato è che esista un dispositivo NVRAM seriale. Ipotesi confermata dalla presenza di un chip con queste caratteristiche sulla piastra madre dell’ apparato. Il dispositivo in questione è lo Xicor x24c44. Un problema correlato con quello appena affrontato, è quello del reperimento delle informazioni contenute nel chip. Questo problema può essere risolto in due differenti maniere: la prima prevede la simulazione della procedura di accesso al dispositivo, leggendo dal dispositivo le informazioni.
La seconda lascia il compito della lettura dal dispositivo alla ROM del apparato e ne analizza solo i risultati. Per la prima procedura sarebbe conveniente scrivere un programma che piloti il monitor del apparato tramite la porta console e tramite opportuni comandi “e” ed “d” acquisisca i dati necessari. L’altra prevede la lettura della memoria nella locazione dove la procedura di inizializzazione deposita i dati dopo averli letti. In entrambi i casi si sono ottenuti dati nei tra i quali è presente l’ indirizzo MAC del dispositivo Ethernet.

```
00000284    07010000 0C38897E 06000000 00000000
00000294    01459324 00000000 00000000 00000000
```

```
Router>show interface ethernet0
Ethernet0 is administratively down, line protocol is down
  Hardware is Lance, address is 0000.0c38.897e (bia 0000.0c38.897e)
  MTU 1500 bytes, BW 10000 Kbit, DLY 1000 usec, rely 252/255, load 1/255
[…]
```

Avendo deciso la necessità di acquisire in qualche modo il codice contenuto nella ROM del dispositivo, bisogna elaborare una strategia per farlo.
A questo punto esistono fondamentalmente due strategie:

- Estrarre i chip della rom e leggere il loro contenuto con un lettore esterno
- Utilizzare l'hardware del router come un lettore di ROM e leggere il contenuto della ROM come un qualche tipo output del router stesso.

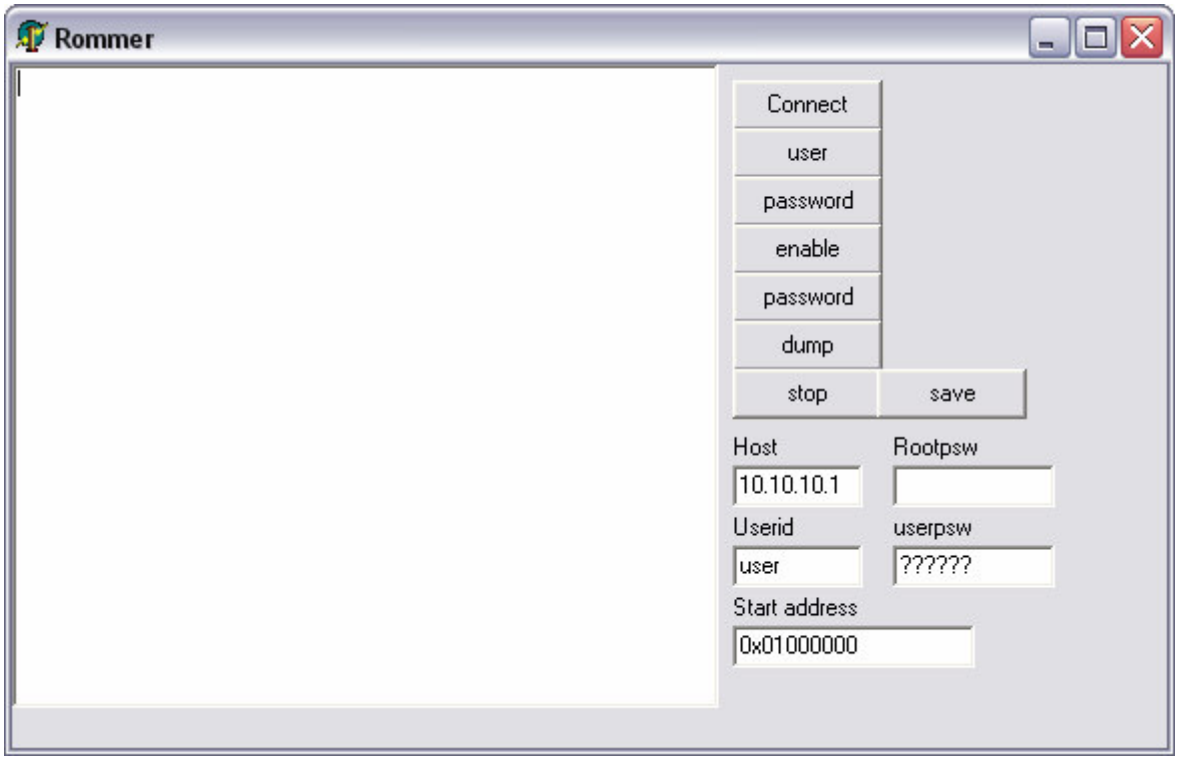
Lasceremo la prima possibilità come ultima, in quanto l'estrazione fisica dei chip della ROM è in qualche modo rischiosa per i chip stessi, e chi dedicheremo alla seconda possibilità.
Non conoscendo le varie funzioni del sistema operativo una idea potrebbe essere quella di scrivere un software da sostituire al sistema operativo, che svolga questa funzione. Questo è un compito difficile per varie ragioni:

- Non si conoscono ancora le caratteristiche dell' hardware del sistema, quindi allo stato di conoscenze attuali non si conoscerebbe nessun modo per fornire in output i dati richiesti.
- Non si conosce come funziona il loader, in altre parole il loader potrebbe verificare alcune caratteristiche del software prima di mandarlo in esecuzione.

Conclusione, la scrittura di un software lettore di ROM, è un compito difficile, anche se praticabile.
Scrivere un software per leggere la ROM non è però l'unico modo per ottenere via software il risultato di un immagine binaria del contenuto della ROM. Il software che gira attualmente sulla piattaforma è un sistema operativo, quindi è possibile che questo già incorpori funzioni simili.
La lettura di documentazione non ufficiale sul sistema operativo in questione, riporta un comando disponibile in modalità supervisore per il dump della RAM, ed un altro comando per la visualizzazione del contenuto della memoria in un punto generico dello spazio di indirizzamento, quindi anche la ROM.
Questo comando è "show memory <indirizzo>".
Un esempio dell' output di questo comando è mostrato in tab###

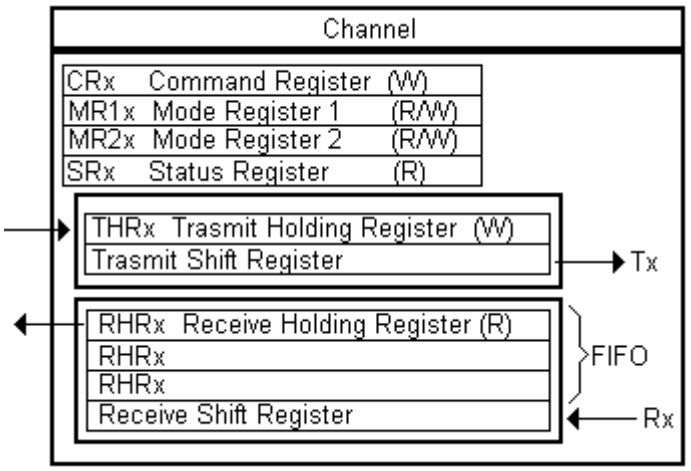
```
cisco1>enable
Password:
cisco1#show memory 0x01000000
01000000: 00001000 01000134 4EF90100 03800024  .......4Ny.....$
01000010: 01008FAA 010001FE 01000294 010002B6  ...*...~.......6
01000020: 010002D0 010002E2 010002F6 0100025C  ...P...b...v...\
01000030: 0100031A 0100033C 01000398 01000212  .......<........
01000040: 01000240 01000398 01000398 01000398  ...@............
```

L'idea è quindi quella di sfruttare questa caratteristica del SO per estrarre il contenuto della ROM.
A questo scopo è stato creato un programma ROMMER che si interfaccia con il router interpretando il ruolo dell' operatore di console, chiedendo via via tutti gli indirizzi di interesse.



Questo programma è stato pensato per funzionare come operatore di console remoto. Collegandosi via Ethernet al router si autentica e richiede le informazioni necessarie. Si sottolinea la scelta di utilizzare l'Ethernet al posto della più comoda interfaccia di console seriale, a causa della velocita di questa, solo 9600 bps.

Il dispositivo SCN2681 (Philips) MC2681 (Motorola) appartiene alla categoria dei Dual Universal Asynchronous Receiver/Transmitter (DUART). Questo dispositivo incorpora due canali indipendenti di comunicazione seriale bidirezionali. Il dispositivo ha l'abilita di invocare le interruzioni, quindi può essere usato sia in interrupt mode che in polling mode. Il dispositivo ha 8 registri per la sua programmazione accessibili sia in lettura, sia in scrittura. I due canali di comunicazione incorporano, ciascuno, un buffer circolare per la ricezione dei dati di 3 posizioni e ciascuno dei due canali ha assegnati dei bit di stato indicanti lo stato del canale di trasmissione.



L' inizio di un progetto di emulazione di un apparato hardware passa sicuramente attraverso una fase in cui si cerca di capire con cosa si ha a che fare. Molti sistemi ci dicono, grosso modo chi sono. Nel nostro caso, il sistema operativo del router ci da informazioni, anche se minimali, sull' hardware del sistema.

Ad esempio il comando "show version"

```
[…]
System image file is "igs-i-l.111-24.bin", booted via flash

cisco 2500 (68030) processor (revision C) with 6144K/2048K bytes of memory.
Processor board ID 01459324, with hardware revision 00000000
Bridging software.
X.25 software, Version 2.0, NET2, BFE and GOSIP compliant.
1 Ethernet/IEEE 802.3 interface.
32K bytes of non-volatile configuration memory.
8192K bytes of processor board System flash (Read ONLY)
[…]
```

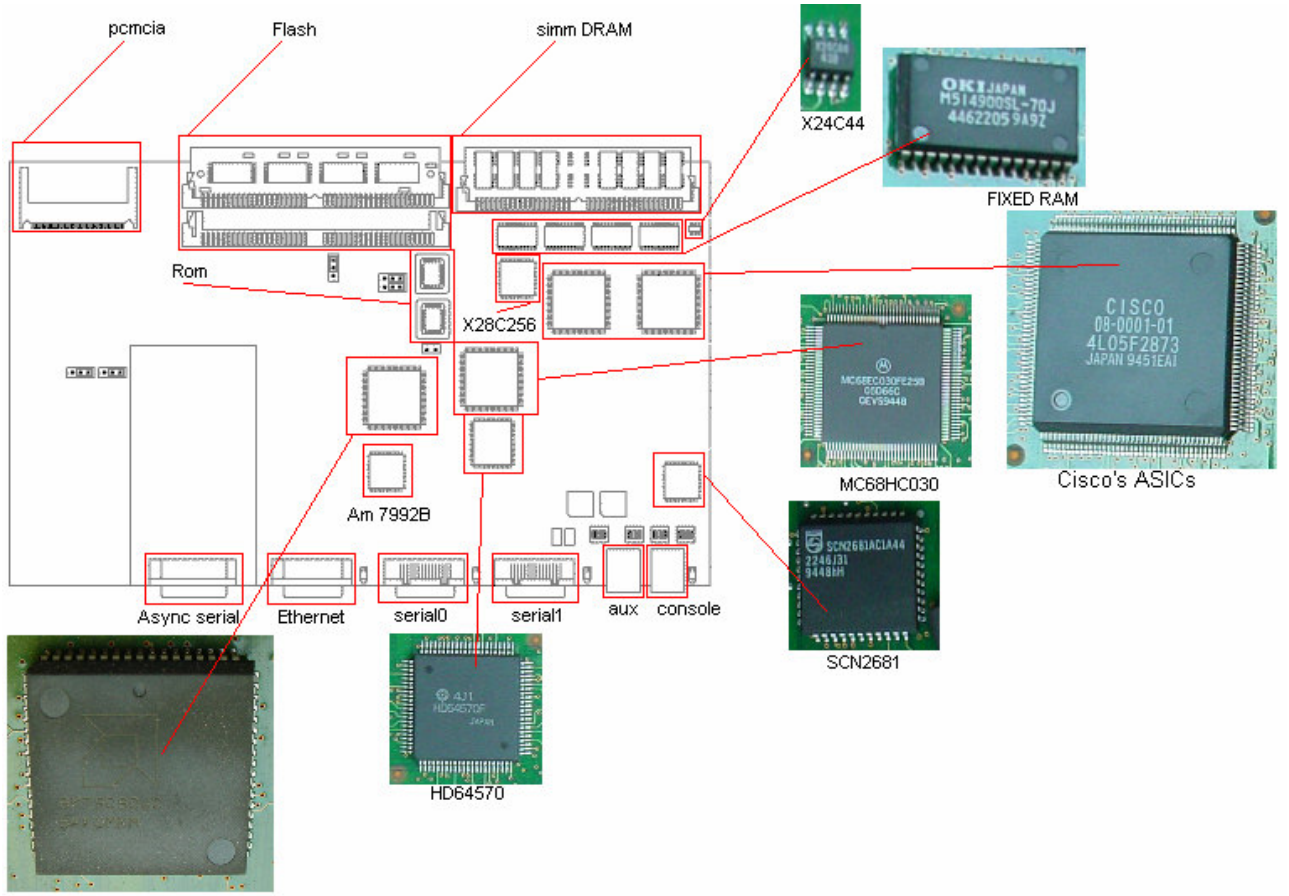
Questo commando mostra chiaramente alcune caratteristiche hardware dell'apparato.
Si vede infatti che il processore è un 68030, che c'è un interfaccia ethernet, che la memoria del dispositivo è 8MB (6144k+2048K), che ci sono 32K si memoria non volatile ed 8MB di memoria flash.
Anche se non estremamente dettagliato, l'output di questo comando mostra già le caratteristiche salienti del dispositivo.
Procedendo su questa linea, è possibile estrarre qualche altra informazione.
Il comando "show interfaces", mostra dettagli sulle interfaccie.

```
Ethernet0 is administratively down, line protocol is down
  Hardware is Lance, address is 0000.0c38.897e (bia 0000.0c38.897e)
  MTU 1500 bytes, BW 10000 Kbit, DLY 1000 usec, rely 255/255, load 1/255
[…]
```

Questo righe ci dicono che l'hardware è un "Lance", e cercando letteratura su "ethernet" e "lance" si giunge al dispositivo AMD 7990 LANCE ethernet controller.

Questo metodo purtroppo, non da informazioni sufficienti sull' hardware del router.

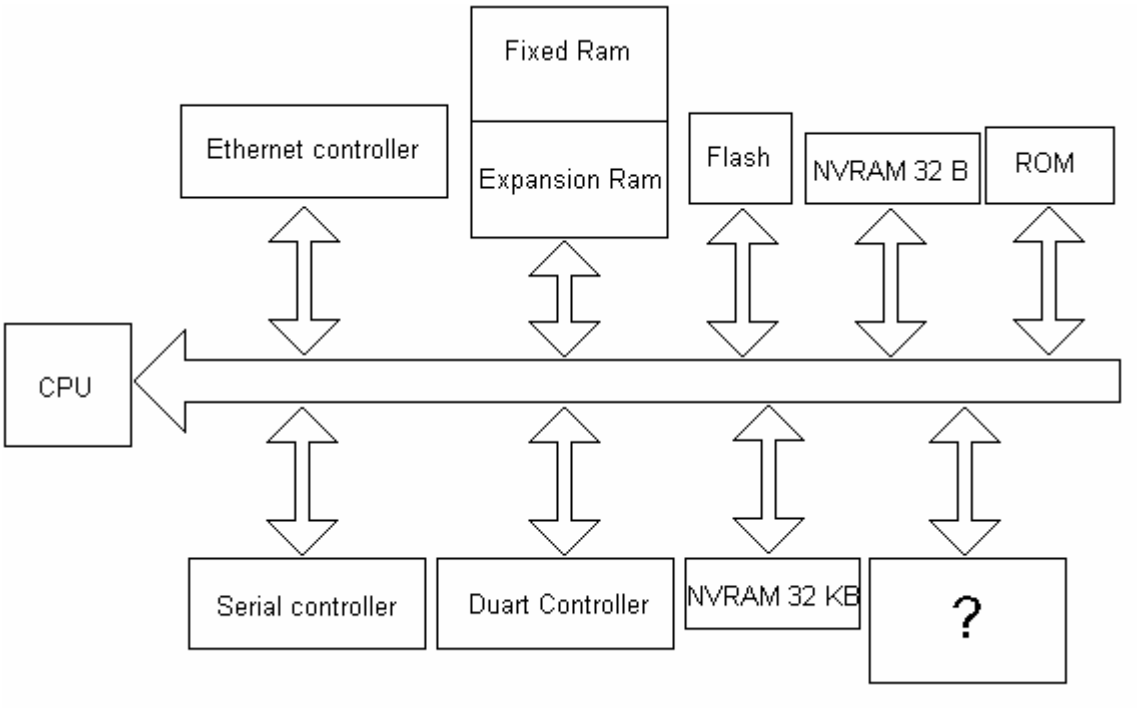
Considerando che Cisco è un produttore di apparecchiature, e non di semiconduttori, è ragionevole credere che le sue apparecchiature siano composte di componenti disponibili sul mercato e quindi componenti di caratteristiche note. Se questo è il caso, su ogni dispositivo del router sarà possibile leggerne il nome.
Osservando la piastra madre del router:



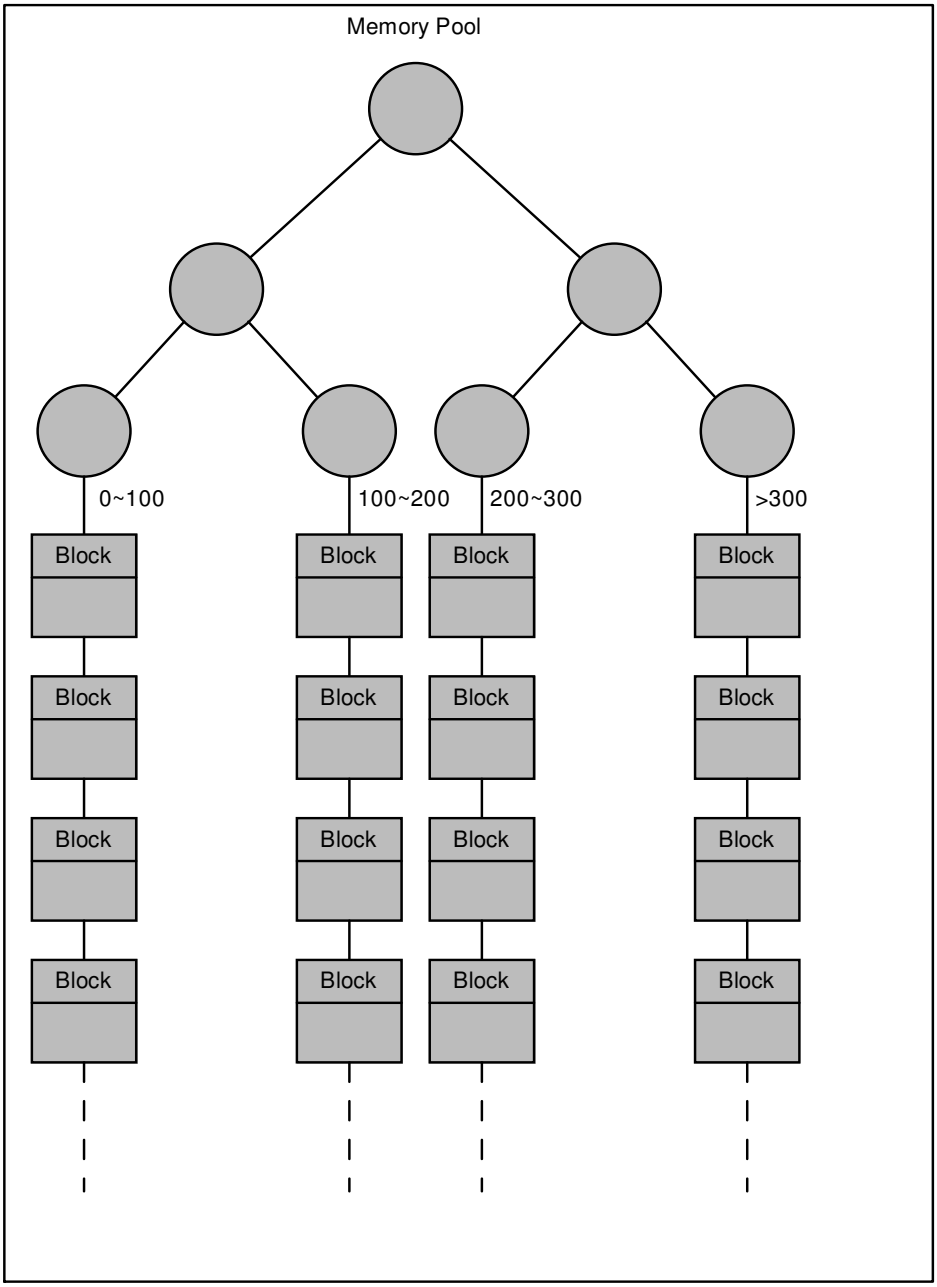
si notano, dopo questa indagine, alcune conferme e precisazioni su quanto era emerso già dall' indagine precedente:
Il processore è un MC68HC030, compatibile con MC68030 ma con una differenza significativa: è privo di MMU.
Il dispositivo ethernet è in effetti un AMD 79C90.
In oltre si viene a conoscenza di alcuni dispositivi di cui si poteva intuire l'esistenza, ma di cui prima di questa indagine non si avevano prove della effettiva presenza.

Un dispositivo DUART per la comunicazione "console" con il router:	SCN2681
Un dispositivo per comunicazioni seriali:	HD64570
Due dispositivi di memoria non volatile:	X24C44 X28C256
Due dispositivi in logica programmabile ASIC:	Cisco 08-0001-01 4L05F2873 Cisco 08-0002-01 4L05F2877
Alloggiamenti per memorie di espansione di tipo	SIMM
Alloggiamenti di espansione per memorie di tipo	FLASH
Presenza di memoria ram saldata sulla piastra madre	OKI M514900SL

Con questo dettaglio di conoscenza si può compilare uno schema che è in prima approssimazione è l'architettura della macchina da emulare.


Fixed Ram
Ethernet controller
Expansion Ram
Flash
NVRAM 32 B
ROM
CPU
Serial controller
Duart Controller
NVRAM 32 KB
?

IOS gestisce la memoria libera dividendola in una serie di memory pools, che sono essenzialmente degli heap e corrispondono ciascuno a zone di memoria contigua.
Ogni memory pool ha associato un insieme di free list, ciascuna contenente blocchi liberi ed in uso di grandezza appartenente ad un certo range. L'ultima free list di ogni pool contiene blocchi di grandezza superiore ad un valore che in genere è dipendente dalla particolare piattaforma che si sta prendendo in considerazione. L' indirizzo della prima componente di ogni free list di uno stesso pool, è memorizzato nelle foglie di un albero binario, in cui il criterio di ordinamento è la dimensione dei blocchi. Durante l'allocazione della memoria, viene cercata la free list di dimensioni appropriate, e quindi in essa cercato il blocco, il quale se dovesse essere più grande, viene ridimensionato e con il rimanente viene fatto un altro blocco.
All'atto della deallocazione della memoria vengono esaminati i blocchi vicini e se liberi, vengono fusi in un unico blocco che eventualmente sarà ricollocato in un'altra free list di dimensioni appropriate.Ogni blocco ha un sistema di sicurezza chiamato red zone. Essenzialmente nella parte terminale di ogni blocco in uso è presente un magic number che viene controllato all' atto della deallocazione salvaguardando così il sistema da un uso improprio della memoria.
Le particolari esigenze di alcune piattaforme, come ad esempio quelle basate su CPU M68k, crea la necessità di routine di allocazione della memoria che generino blocchi di memoria il cui indirizzo risulta "allineato", essenzialmente significa che gli indirizzi sono divisibili per 4.


Memory Pool
0~100
100~200
200~300
>300
Block

Lo scopo di questa fase è acquisire informazioni dell' hardware analizzando il codice che lo pilota. Quindi acquisite le scarse informazioni ufficiali \cite {maps} si passa all' analisi del codice.
Le informazioni ufficiali sulla mappa della memoria del Cisco 2500, sono abbastanza povere, quindi assumeremo come mappa da raffinare la mappa del Cisco 3000 , di cui si ha qualche dettaglio in più.
Il primo passo è l'individuazione dell' entry point, cioè il punto da cui comicia l'esecuzione del codice. Per l'individuazione dei questo indirizzo, assumiamo il comportamento standard dei processori serie Motorola 680x0. Al power up caricano il vettore di reset dalla tavola delle eccezioni, e successivamente caricano a7 con il valore di default per lo stackpoint (preso sempre dalla tavola dei vettori). La tavola dei vettori deve essere un array di longword i cui indirizzi saranno in base 0x01000000. Questa considerazione si basa sul fatto che le routine di interruzione programmate nella rom dovranno essere necessariamente mappate nello spazio di indirizzamento della rom stessa. Questa struttura dati in effetti la si è trovata all' indirizzo 0x01000008, quindi il valore che stami cercando è il secondo elemento di questa struttura 0x01000134.
Questo è l'entry point della rom.
A questo indirizzo troviamo il codice di Tab. \ref {code1}

01000134	Starts_here:	move.l #$3E8,d0	Inizializza il contatore a 1000
0100013A	loc_100013A:	subq.l #1,d0	Countdown
0100013C		bgt.s loc_100013A	Ripeti se non finito
0100013E		movea.l #0,a6	Inizializza il bp dello stack
01000144		lea $1000008,a0	Carica in a0 il VB addr della Rom
01000148		movec a0,vbr	Setup vb in rom

Questo codice conferma la nostra ipotesi di aver trovato la tavola dei vettori.
Un dispositivo facilmente rintracciabile è la DUART console.
Da questo dispositivo infatti devono passare tutti i messaggi verso l'esterno che partono dal router.
Dunque la strategia, è individuare un messaggio qualsiasi e seguire il flusso del programma fino ad all' output alla periferica.
Prendiamo ad esempio il messaggio "[OK]\r\nSizing RAM and ROM..." locato all' indirizzo 0x01005A89. il codice che gestisce l'output di questo messaggio è mostrato in Tab. \ref {code2}

0100642C		lea $1005A89,a0	Carica in a0 il ptr alla stringa
01006430		tst.b (a0)	Verifica che non sia vuota
01006432		beq.s exit	Esce se vuota
01006434	Loop:	move.b ($2120101).l,d0	Leggi SR del 2681
0100643A		btst #2,d0	È disponibile il dispositivo
0100643E		beq.s Loop	Se no, attendi
01006440		move.b (a0)+,($2120103).l	Manda in output un carattere
01006446		tst.b (a0)	La stringa è terminata?
01006448		bne.s Loop	Ripeti l'operazione

La documentazione ufficiale del Cisco 3000 afferma infatti che il dispositivo scn2681 è mappato a partire dall' indirizzo 0x02120100, ed il codice in Tab \ref {code2} lo conferma. Per la funzione dei vari registri si rimanda alla documentazione ufficiale del dispositivo \cite {2681}.

```
0x0100673A                  clr.l   d2
0x0100673C                  clr.l   d1
0x0100673E                  suba.l  a0,a0
0x01006740    loop:         movea.l d1,a4
0x01006742                  move.l  d1,(a4)
0x01006744                  tst.l   (a0)
0x01006746                  bne.s   loc_1006758
0x01006748                  addq.l  #1,d2
0x0100674A                  addi.l  #$80000,d1
0x01006750                  cmpi.l  #$FFFFFF,d1
0x01006756                  bls.s   loop
0x01006758    loc_1006758:
```

Questa routine fa parte dell' inizializzazione dell'hardware del router, in particolare, si capisce dai messaggi precedenti e successivi di output, svolge la funzione di determinare l'ammontare della RAM installata sul sistema. La lettura del codice, senza la conoscenza dell' hardware, può portare ad ipotesi fuorvianti. Nel mio caso c'era una enorme lacuna nella conoscenza dell'hardware, che mi ha portato a considerare un ipotesi, che poi si è rilevata falsa.
Ma vediamo qual è il problema di questa routine.
Esaminando il codice si nota all' indirizzo 0x0100673E l'azzeramento di A0.
L'indirizzo 0x00000, azzerato precedentemente da un'altra funzionalità della routine di inizializzazione, è una locazione che nei 68000 è riservata al PC dopo il reset. Un fatto che servirà ad avvalorare l'ipotesi sbagliata.
La routine prosegue scrivendo valori $\neq 0$ in locazioni multiple di 0x80000 fino ad arrivare a 16MB, controllando che la locazione puntata da A0 (0x0000) rimanga al valore 0. È da notare che nel ciclo non vi sono istruzioni esplicite che modificano il contenuto di questa locazione. Rimane quindi da spiegare come questa routine riesca a determinare l' ammontare della RAM. L'ipotesi fatta è che la scrittura in una locazione non valida attivi una eccezione di "page fault" che comunichi con l'esterno comunicando il codice di errore scrivendolo nella locazione 0x0000.
Vediamo ora un po' di fatti:

Contro	**A favore**
Il processore MC60EC030 è un processore senza MMU.	Un dispositivo MMU può essere anche esterno al processore.
Non sarebbe economicamente vantaggiosa una architettura basata su un MC68EC030 ed una MMU esterna.	L'architettura del Cisco 2500 è ereditata dai suoi predecessori, che in generale potevano avere una MMU esterna.
L'eventuale MMU esterna sarebbe dovuta essere inizializzata in qualche modo dal processore principale prima del codice in questione.	Le mappature della memoria possono essere in generale programmate nel dispositivo, oppure decise da settaggi hardware di tipo jumper. Sulla piastra madre del C2500 esistono una serie di jumper di cui si ignora la funzione.Nel codice sono presenti riferimenti a dispositivi esterni sconosciuti.
Da un esame visivo della piastra madre non si è rilevata la presenza fisica di alcun dispositivo che notoriamente ha funzioni di MMU.	È provata l'esistenza di 2 dispositivi ASICs (application-specific integrated circuit), di cui si ignora la funzione.

Implementare le funzioni di un M68302, sarebbe per i progettisti uno sperpero inutile di risorse. Perché ricostruire un M68302 in un ASIC quando lo si potrebbe avere senza fatica semplicemente comprandolo? E comunque cosa ci sarebbe nell'altro ASIC?
Anche se esiste una serie di registri non identificati, tra cui una serie di 0x400 bytes contigui, questi non sembrano essere sufficienti ad implementare le funzioni di un M68302

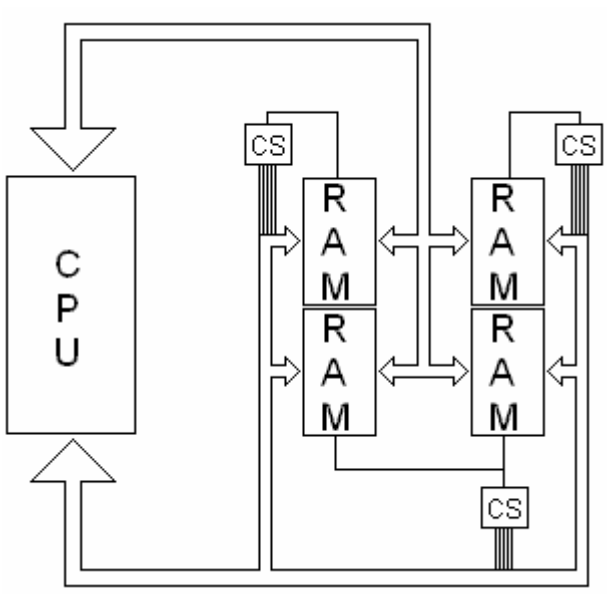
In documenti ufficiali Cisco, è dichiarata la presenza su alcuni C3000 (presunti cloni dei C2500 ASICless), di dispositivi M68302, dispositivi multifunzione con i quali sarebbe eventualmente possibile ottenere delle funzioni equivalenti a quelle di una MMU.
Un M68302 ha un interfaccia basata su gruppi contigui nello spazio di indirizzamento di registri contigui. Considerando che l'eventuale esistenza di un M68320 sarebbe una reimplementazione delle sue funzioni in un ASIC, potrebbe essere verosimile il fatto che l'implementazione non sia completa, e che siano state tralasciate le funzioni di questo dispositivo non indispensabili come ad esempio le UART 16550 e le interfacce parallele.

L'eventuale scelta benché possibile, non risulta conveniente. Si nota infatti all' indirizzo 0x 01000148 il ripristino della tavola degli interrupt a partire da 0x0000. Questo di fatto limita l'eventuale comunicazione in questo modo a soli 64bit utili (power up PC, SP)

È plausibile che la routine di gestione dell'eccezione "page fault", possa comunicare con il SO attraverso la locazione 0x0000. La locazione è infatti utilizzata solo in seguito ad un power up od un reset, ed è notorio che in questi casi i dati in questione non possono trovarsi sulla RAM, ma devono essere in un dispositivo ROM.

Le verifiche sul campo hanno poi dimostrato l'incompatibilità dei dati contenuti nei registri ed i layout dei dati dei registri del dispositivo M68302, e la non presenza di routine di gestione delle eccezioni che modificano il contenuto della locazione 0x0000.

Una spiegazione più semplice al bizzarro funzionamento di questa routine può essere dato dalla presenza di un dozzinale circuito di decodifica degli indirizzi.



Supponiamo ad esempio che le RAM nello schema siano 512KB ciascuna per un totale di 512KBx4 = 2MB.
Supponiamo inoltre che i dispositivi di coincidenza CS utilizzino solo 2 linee del bus indirizzi per la decodifica.
Quindi 19 bits sono quelli necessari ad ogni modulo RAM per decodificare l' indirizzo più altri 2 di chip select si arriva ad 21 bits che in effetti rappresentano i nostri 2 MB. In questo modo, selezionando l' indirizzo 2^{21} si ha un warp round che seleziona di nuovo il banco RAM contenente l' indirizzo 0x0000. Questa potrebbe essere una spiegazione semplice del bizzarro modo di funzionare della routine in questione.

Un indagine basata sull' ispezione visiva dei componenti, benché dia informazioni preziose sui componenti presenti nel sistema, non da assolutamente alcuna informazione su quei componenti sprovvisti di datasheets, come ad esempio i due ASIC marchiati Cisco, e soprattutto non da alcuna informazione su come questi componenti sono mappati in memoria.
Per indagare su questi aspetti c'è la necessità di analizzare il codice del sistema operativo.
La prima domanda alla quale bisogna dare una risposta è se il codice del sistema operativo è autosufficiente, o se invece fa uso di una ROM nella quale sono memorizzate funzioni di libreria.
La prima parte della risposta è in parte suggerita dal test precedente, in effetti sulla piastra madre dell' apparato sono presenti due chip presumibilmente di ROM.
In aggiunta, risono alcuni fatti da considerare:

- il file del sistema operativo è disponibile nella versione binaria per la consultazione
- esiste una modalità di funzionamento, i cui messaggi non sono presenti nel file del sistema operativo
- i primi messaggi che appaiono durante il boot (fig)dell' apparecchiatura non sono presenti nel file del sistema operativo.

```
System Bootstrap, Version 4.14(9.1), SOFTWARE
Copyright (c) 1986-1994 by cisco Systems
2500 processor with 6144 Kbytes of main memory

F3: 3832048+95972+196092 at 0x3000060

          Restricted Rights Legend
[...]
```

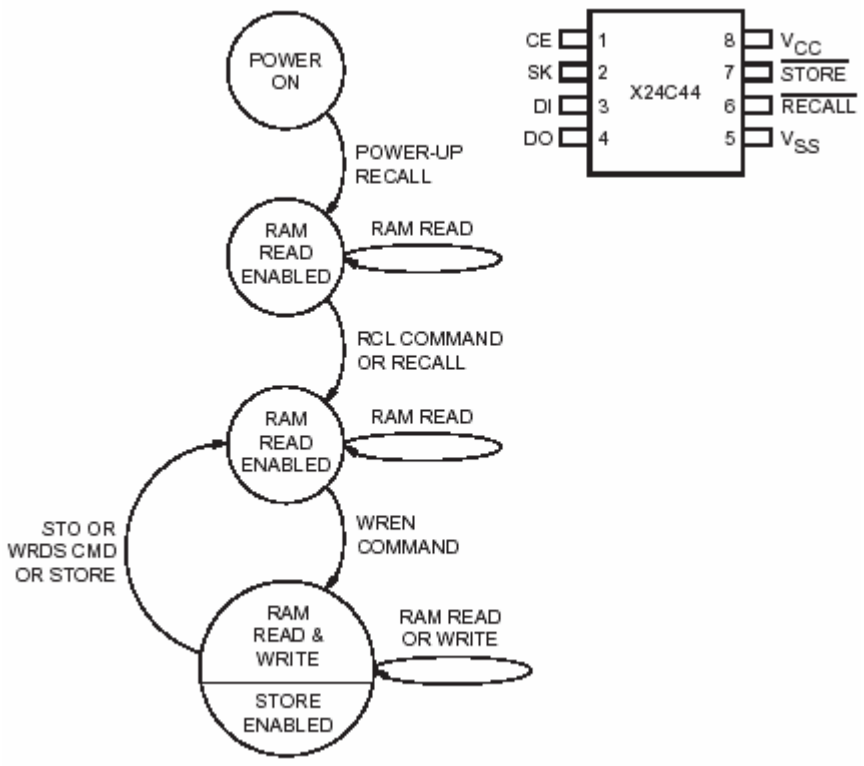
Si conclude quindi che esiste una parte di codice situate nella ROM, della quale non sono disponibili file immagine, che svolge almeno le funzioni di loader.
La prossima domanda alla quale bisogna rispondere è se le funzioni del codice scritto nella ROM si esauriscono con il caricamento del SO oppure se ci sono altre funzioni, ad esempio un supporto di tipo BIOS. La risposta a questa domanda può far cambiare la strategia su come procedere. Ad esempio, Se la risposta fosse ha solo funzioni di loader, si potrebbe pensare di far svolgere queste funzioni al sistema, ed ignorare il codice presente nella ROM, in caso contrari, e cioè nel caso in cui nella ROM ci fossero anche funzioni BIOS, non si potrebbe ignorare questo codice e si dovrebbe necessariamente cercare di estrarre il codice in qualche modo.
Per verificare che funzioni abbia la ROM bisogna guardare come è fatto il codice del sistema operativo, cercando qualche eventuale chiamata al sistema.

```
030D15DC  link    a6,#0
030D15E0  move.l  arg_C(a6),-(sp)
030D15E4  move.l  arg_8(a6),-(sp)
030D15E8  move.l  arg_4(a6),-(sp)
030D15EC  move.l  arg_0(a6),-(sp)
030D15F0  trap    #$F
030D15F0  unlk    a6
030D15F4  rts
```

Questo parte di codice estratta dal codice del sistema operative è una prova che la ROM fornisce oltre al servizio di loader, anche un servizio di libreria di natura ignota.
Queste considerazioni rendono quindi indispensabile estrarre il codice della ROM.

Lo Xicor X24C44 è un dispositivo di memoria non volatile organizzato in una matrice 16x16 bit. Il dispositivo è accessibile da quattro soli piedini, di conseguenza anche l'interfaccia che ha con il sistema è governata da quattro soli segnali binari. I quattro segnali sono nell'ordineun bit di input DI (data input) un bit di output DO (data output), un bit di clock CK (clock) ed un bit di abilitazione CE (chip enable).
La memoria del dispositivo è gestita su due livelli,la ram tampone che serve per scambiate informazioni con l' esterno e la ram permanente che serve per la memorizzazione vera e propria.
Il X24C44 è un dispositivo programmabile con 6 comandi:

- Store: Trasferisce il contenuto della ram volatile in quella permanente.
- Recall: Trasferisce il contenuto della ram permanentein quella volatile.
- Write: Scrive un byte, quindi trasferisce una sequenza di 8 bit da DI nella ram volatile.
- Read: Legge un byte, quindi trasferisce una sequenza di 8 bit dalla ram volatile a DO.
- WREN: Abilita il trasferimento dalla ram volatile alla permanente.
- WRDS: Disabilita il trasferimento dalla ram volatile alla permanente.



Start	Length	Width	Description
0x02000000	0x8000	16	Confreg+ NVRAM
0x02100000	0x400	?	?
0x02110000	0x2	16	System control register 1
0x02110002	0x2	16	System control register 2
0x02110004	0x2	16	Reset control register
0x02110006	0x2	16	Interrupt status register
0x02110008	0x2	16	Control-status register
0x0211000A	0x2	16	?
0x02110010	0xA	8	?
0x02110040	0x40	8	Cookie
0x02110060	0x1	8	x2444 control register
0x02120040	0x1	8	Timer control register
0x02120050	0x2	16	Timer register 0
0x02120060	0x2	16	Timer register 1
0x02120070	0x2	16	Timer register 2
0x02120100	0x40	8	Uart 2681 console/aux interface
0x02130000	0x4	16	Ethernet network interface A registers
0x02131010	0x4	16	Token ring H/W Ch C map register 1
0x02132000	0x100	16	?(serial)
0x02132100	0x7	16	Serial 0 device register
0x02134000	0x2	?	?? Unused