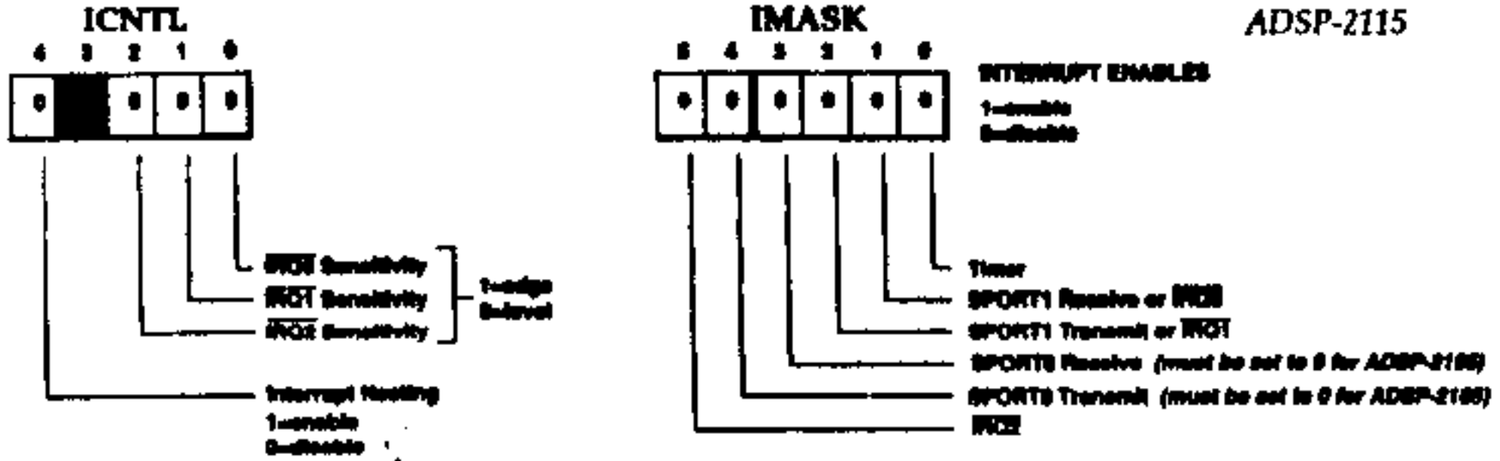
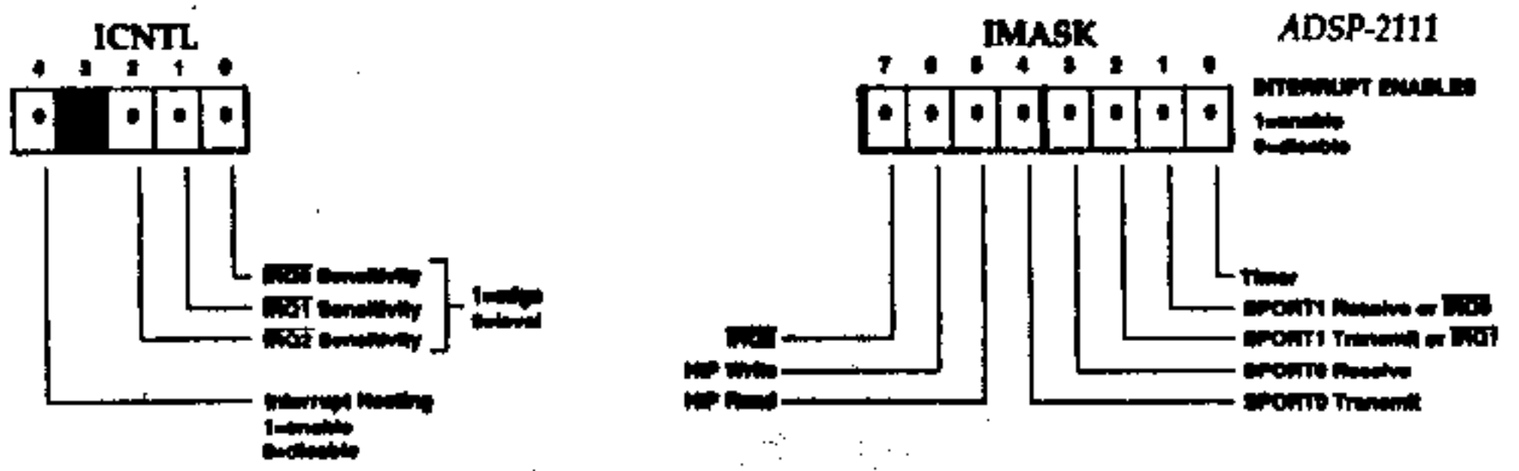
Interrupt Registers
(Non-Memory-Mapped)

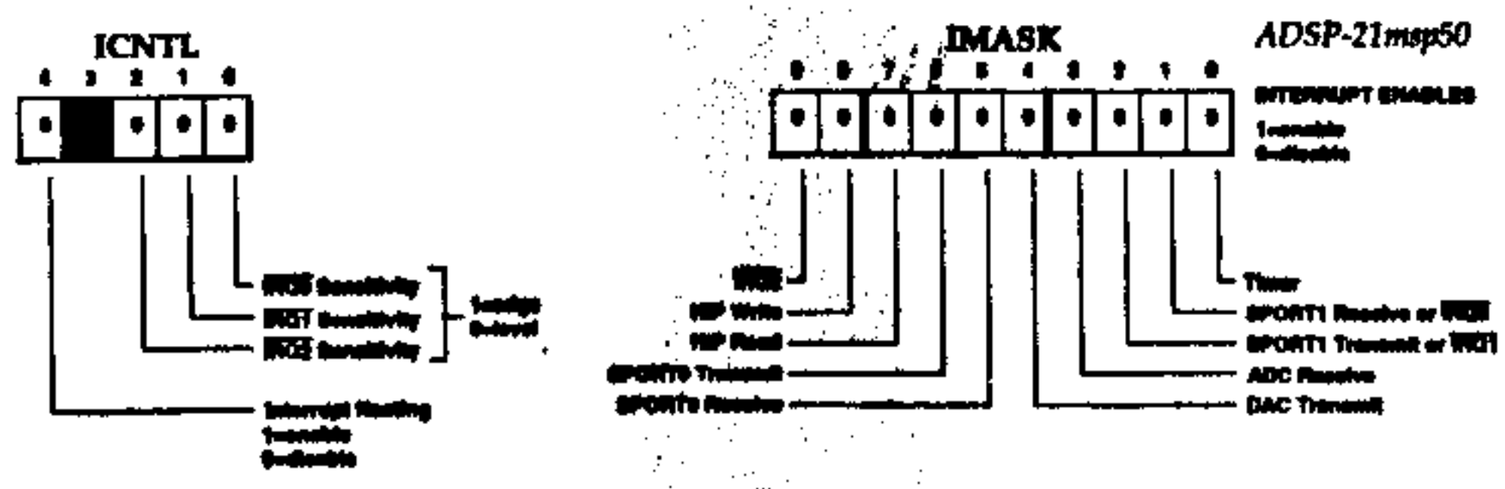
ADSP-2101
ADSP-2105
ADSP-2115



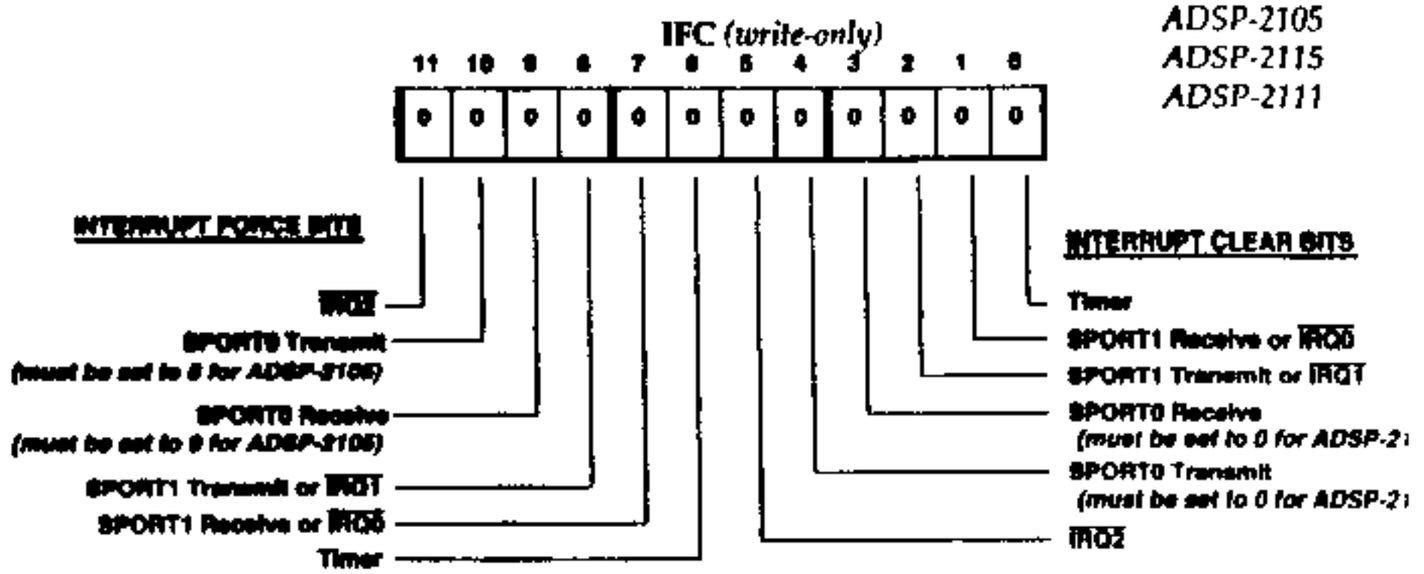
ADSP-2111



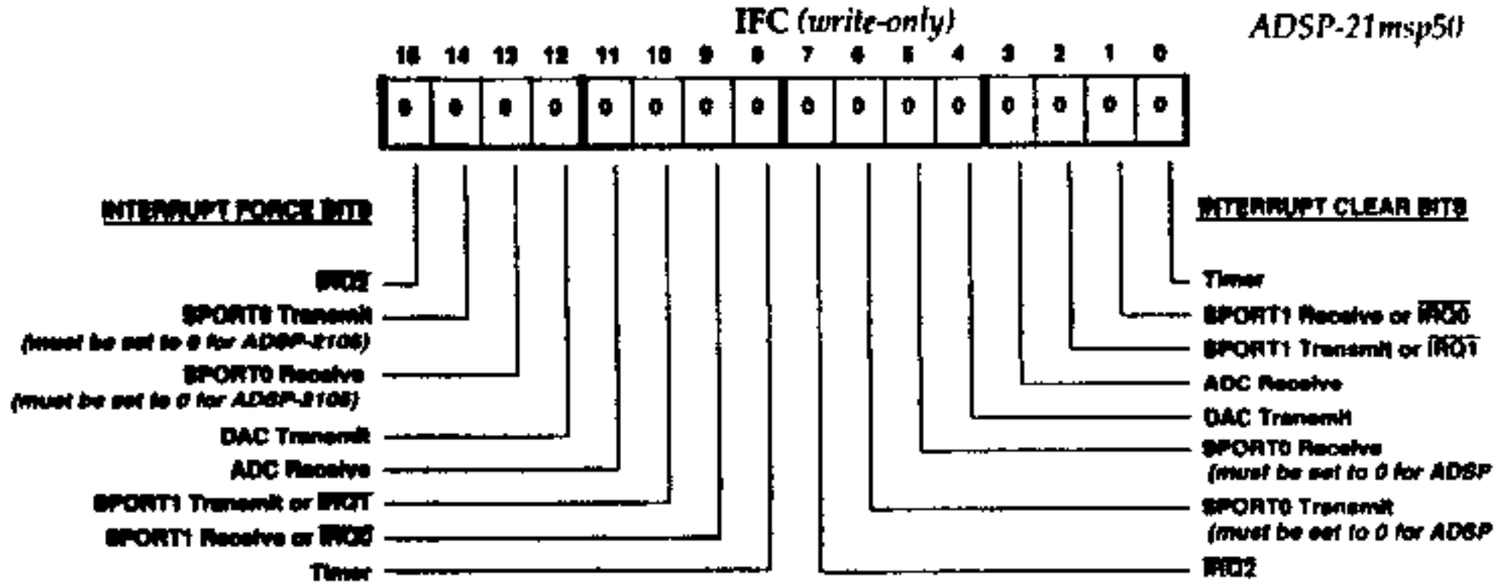
ADSP-21msp50



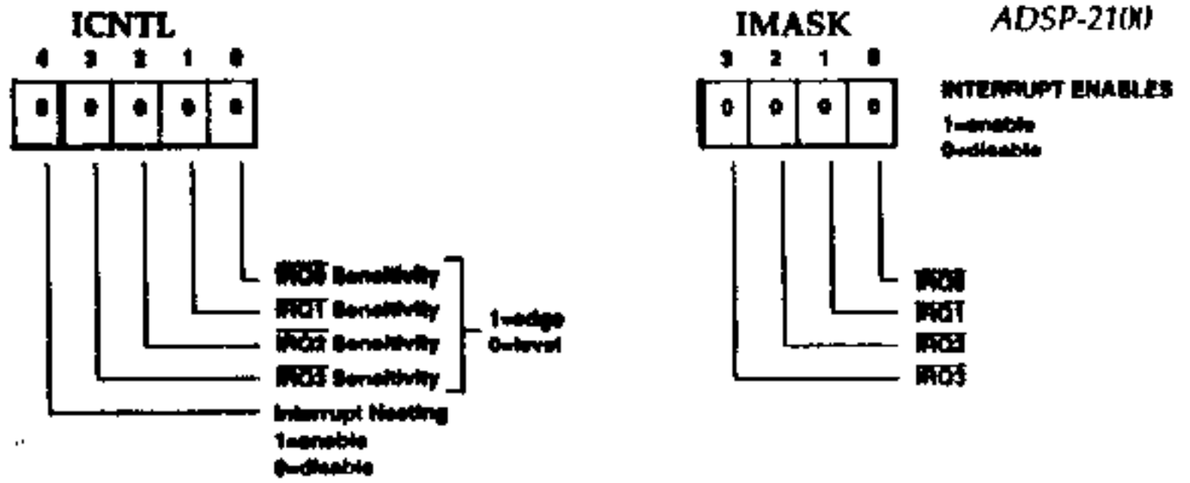
ADSP-2101
ADSP-2105
ADSP-2115
ADSP-2111



ADSP-21msp50



ADSP-2100



Esami di Stato per l'abilitazione all'esercizio della professione di Ingegnere

II Sessione - Novembre 1996

Ingegneria *Elettronica* ed *Informatica*

Tema numero 3

Il candidato svolga l'esercizio approfondendo maggiormente le parti più adatte alla sua preparazione specialistica.

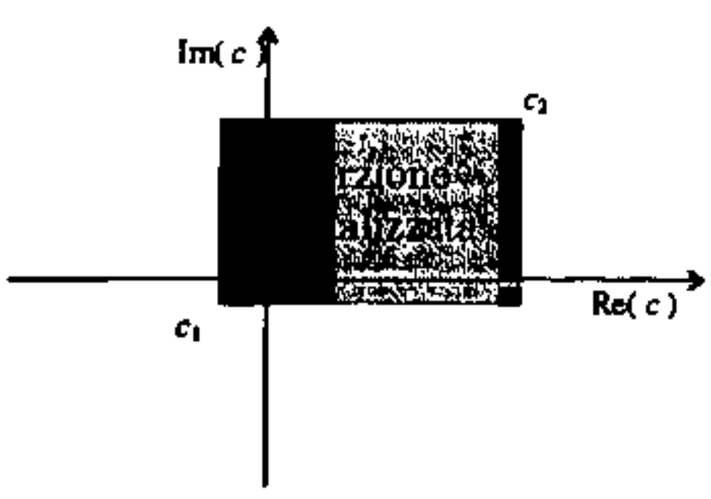
Si progetti un sistema per il calcolo e la visualizzazione ad alta velocità di immagini frattali, quale il famoso *insieme di Mandelbrot*. L'elaborazione si basa sulla seguente equazione:

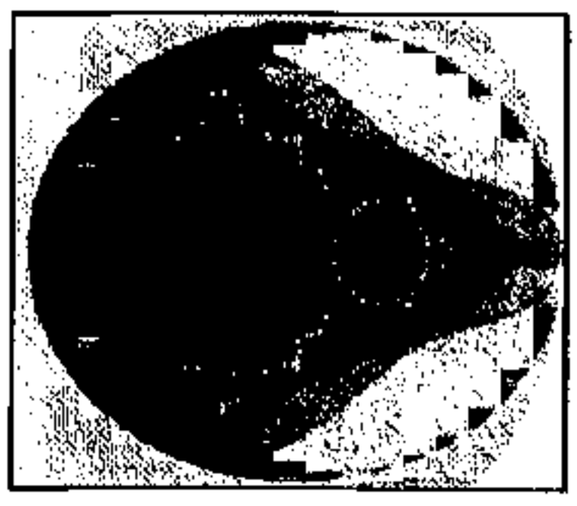
$$\begin{cases} z_0 = 0 \\ z_{k+1} = z_k^2 + c \end{cases} \tag{1}$$

dove sia z che c sono numeri **complessi**, e la parte reale ed immaginaria di c sono entrambe comprese tra -1 e +1. Si dice che un determinato numero c appartiene all'insieme di Mandelbrot se la successione degli z_k non diverge al crescere di k. Dal punto di vista pratico, si fissa un numero massimo di iterazioni N_{max} (pari a 1024) e si definisce un *indice di divergenza* $I(c)$, il quale indica, per ciascun valore c quanto in fretta diverge la successione, secondo la definizione seguente:

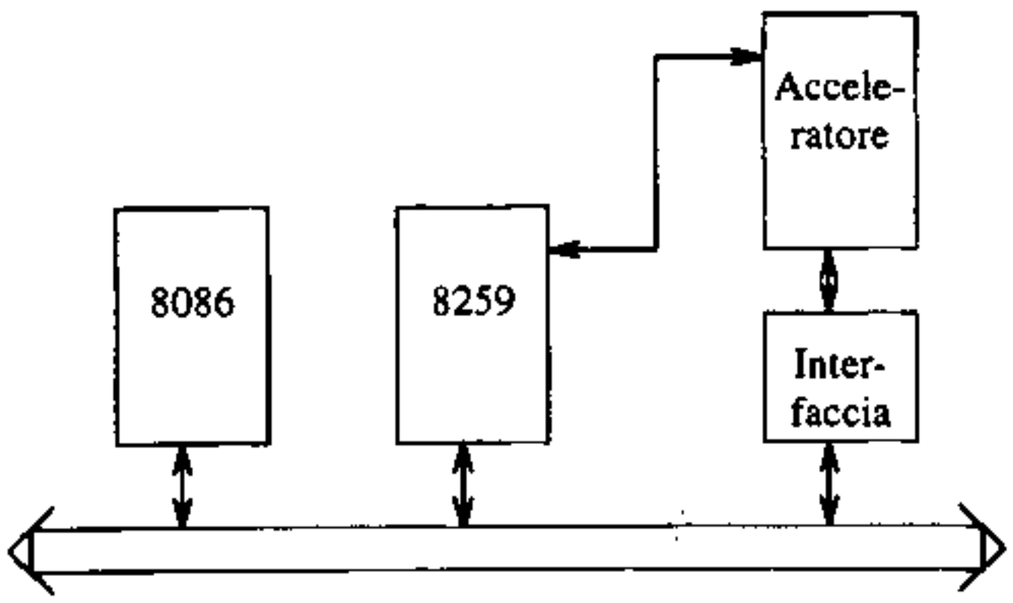
$$\begin{cases} I(c) = k & \text{per il primo } k \text{ per cui } |\mathrm{Re}\, z_k| \geq 2 \text{ o } |\mathrm{Im}\, z_k| \geq 2 \\ I(c) = N_{max} & \text{se la condizione precedente non è mai verificata per } k < N_{max} \end{cases} \tag{2}$$

Il programma dovrà visualizzare un'immagine di risoluzione 1024x768x256 corrispondente ad un intervallo di valori del parametro complesso c. I valori estremi c_1 e c_2 vengono decisi dall'utente, ed il sistema provvede a tracciare l'immagine corrispondente calcolando, per ciascun pixel, il valore di c corrispondente e disegnando il pixel in un colore determinato dalla funzione $I(c)$, opportunamente riscalata sulla scala da 0 a 255.





Il sistema deve essere realizzato su un Personal Computer, in cui le funzioni di calcolo dell'equazione (1) vengano accelerate da un dispositivo hardware dedicato, secondo lo schema a blocchi di seguito riportato.



Elaborazioni aritmetiche

Tutte le operazioni devono svolgersi in virgola fissa con una precisione di almeno 16 bit. Si ipotizzi che i valori di z, parti reale e immaginaria, non superino mai, in valore assoluto, il valore 2. Per la costante c tale limite è pari ad 1.

Si definiscano, in particolare, le ampiezze dei registri contenenti le componenti di z e c in modo da controllare il problema dell'overflow. Si indichino chiaramente la codifica dei dati e la posizione della virgola.

Acceleratore hardware

Ciascun candidato *scelga* la realizzazione con logica dedicata *oppure* quella tramite DSP.

Realizzazione con logica dedicata

Si svolga il progetto utilizzando componenti standard a livello RT. È disponibile un moltiplicatore parallelo 16 bit x 16 bit fixed point signed o unsigned, le cui caratteristiche tecniche sono un'estensione a 16 bit del componente Texas SBP 9708.

Si richiede il diagramma a stati dell'unità di controllo ed un diagramma indicante la connessione dei componenti utilizzati nel data path.

Realizzazione con DSP

Si svolga il progetto utilizzando il componente Analog Devices ADSP-2101 oppure 2111.

Interfaccia

Si supponga disponibile un'interfaccia per la comunicazione sul Bus di sistema. Tale interfaccia *non* deve essere progettata, ma occorre specificare il formato dei dati scambiati attraverso di essa.

L'interfaccia dispone di 4 registri:

Registro	*Ampiezza*	*Direzione*	*Indirizzo*
COMMAND	8 bit	CPU->acceleratore	0x0300
STATUS	8 bit	acceleratore->CPU	0x0301
DATA	16 bit	CPU->acceleratore	0x0302
RESULT	16 bit	acceleratore->CPU	0x0304

Software

Il software del sistema di visualizzazione deve prevedere la visualizzazione dell'immagine sfruttando l'hardware appositamente progettato oppure tramite un'emulazione via software delle stesse funzionalità.

Gestione del dispositivo hardware

Il calcolo del colore di ciascun pixel, legato al valore della funzione $I(c)$, è demandato al dispositivo hardware progettato. I registri dell'interfaccia sono visibili sul bus di sistema agli indirizzi di I/O indicati sopra, ed il dispositivo segnala il termine dell'elaborazione attivando un interrupt IRQ5 (vettorizzato su INT 0Dh).

Emulazione in software del dispositivo

Su richiesta dell'utente, anziché utilizzare il dispositivo, un'apposita procedura software deve calcolare lo stesso risultato, con la stessa precisione. Si utilizzi aritmetica a virgola fissa secondo le indicazioni date in precedenza.

Questa parte dell'esercizio deve essere svolta *obbligatoriamente*, e l'algoritmo usato nell'emulazione software deve rispecchiare il più fedelmente possibile quello implementato dall'acceleratore. Si invitano coloro che hanno scelto una realizzazione con logica dedicata a scrivere l'emulatore in C, e coloro che hanno scelto la realizzazione con DSP a scriverlo in Assembler richiamabile da C.

Interfaccia utente

Il programma visualizza inizialmente l'immagine corrispondente a $c_1 = -1-j$ e $c_2 = 1+j$. Successivamente, l'utente potrà richiedere le seguenti operazioni:

- spostamento dell'immagine nelle 4 direzioni di una quantità pari a metà della dimensione dell'immagine
- ingrandimento di un fattore 2 rispetto al centro dell'immagine visualizzata
- riduzione di un fattore 2 rispetto al centro dell'immagine visualizzata.

Si consideri la possibilità di non ricalcolare la funzione $I(c)$ qualora alcuni pixel della nuova immagine siano comuni a quelli dell'immagine precedente.

Libreria grafica

Si ipotizzi la disponibilità di una libreria grafica elementare, come quella presente nel compilatore Borland C/C++. In particolare possono risultare utili le procedure seguenti:

- `void putpixel(int x, int y, int color)`
- `unsigned getpixel(int x, int y)`
- `void line(int x1, int y1, int x2, int y2)`
- `void bar(int left, int top, int right, int bottom)`
- `void outtextxy(int x, int y, char *textstring)`

Elaborazione dell'immagine

Al termine della visualizzazione di ciascuna immagine, il programma deve:

1. calcolare l'*istogramma* dei colori, ossia disegnare un diagramma a barre indicante il numero di pixel che assumono un colore. Ciascuna barra rappresenta 16 colori diversi, da 16**i* a 16**i*+15.
2. calcolare il numero di *regioni connesse* esistenti, ossia il numero di insiemi di pixel adiacenti tra loro e dello stesso colore.

Analog Devices
DSP Applications
Technical Support Phone Line (617) 461-3672
DSP Bulletin Board (BBS) (617) 461-4258 8 data bits, no parity, 1 stop bit
300/1200/2400/9600 baud

You can also contact the DSP Division in the following ways:

- By contacting your local Analog Devices Sales Office
- For Marketing information, call (617) 461-3881 in Norwood, Massachusetts.
- The Norwood office Fax number is (617) 461-3010
- By writing to:

  Analog Devices
  DSP
  One Technology Way
  P.O. Box 9106
  Norwood, MA 02062-9106
  USA

ADSP-2100 Family Programmer's Quick Reference

PRINTED IN U.S.A.

Development Software Invocation Commands

System Builder `bld21 sys_file [-c]`

Switches

`-c`	Case-sensitivity for architecture file symbols

Assembler `asm21 sourcefile [-switch ...]`

Switches

`-c`	Case-sensitivity for program symbols
`-l`	.LST list file generated
`-m [depth]`	Macros expanded in .LST file
`-i [depth]`	Show contents of INCLUDE files in .LST file
`-o filename`	Rename output files *(default: SOURCEFILE.OBJ)*
`-dident[=literal]`	Define identifier for assembler's C preprocessor
`-s`	No semantics checking on multifunction instructions

Linker `ld21 file1 [file2 ...] [-switch ...]`
or `ld21 -i file_all [-switch ...]`

Switches

`-a archfile`	.ACH architecture file read by linker *(default: 210X.ACH)*
`-i file_all`	Files listed in indirect file *file_all* are linked
`-e executable`	Output files given filename *EXECUTABLE.EXE (default: 210X.EXE)*
`-dryrun`	Quick run to test for link errors *(no .EXE file generated)*
`-g`	.SYM symbol table file generated
`-x`	.MAP memory map file generated
`-lib`	ADSP-21xx Runtime C Library linked
`-user fastlibr`	Search library file generated by librarian
`-dir directory;...`	Specify directories to search for library routines
`-p`	Assign library routines to boot pages where called

Librarian `lib21 lib_name file1 [file2 ...] [-v version]`
or `lib21 lib_name -i file_all [-v version]`

Switches

`-v version`	Embed version number in library file created *(lib_name)*
`-i file_all`	Files listed in indirect file *file_all* are taken as input

Simulators

```
sim2100 [-switch ...]
sim2101 [-switch ...]    use also for 2105, 2115
sim2111 [-switch ...]
sim2150 [-switch ...]
```

Switches

`-a archfile`	.ACH architecture file read by simulator *(default: 210X.ACH)*
`-e exe_file`	.EXE program file loaded by simulator
`-c`	Case-sensitivity for program symbols
`-k keyfile`	Load and execute keystroke file
`-o msgfile`	Generate file containing error messages and Command Output Window messages
`-w winfile`	Simulator starts up with previously saved windows display file *(.WIN)*

C Compiler

```
g21 sourcefile [sourcefile ...]  [-switch ...]
or g21 @file_all  [-switch ...]        | source files listed in file_all |
```

Source (Input) File *Filename Extension*	*File Contents and* *Translation Operation*	*Output File* *Filename Extension*
.c	C source to be preprocessed	.i
.h	Macro or header file to be preprocessed	.i
.i	C source to be compiled	.s
.asm *or* .dsp *or* .s	Assembler source to be preprocessed	.is
.is	Assembler source to be assembled	.obj
.obj	Object file to be linked	.exe
.a	Library file of function definitions	–

Switches

-a *archfile*	.ACH architecture file read by compiler
-c	Compile (and/or assemble) only, without linking
-D*macro*[=*defn*]	Define macro
-g	Generate debugging information for simulator's C source debugger (CBUG)
-I*directory*	Include search directory (append *directory* to search path for include files)
-L*directory*	Library search directory (linker appends *directory* to search path for library files)
-l*library_name*	Link library file (linker searches library file for functions)
-map	Direct linker to generate .map memory map file of symbols
-o *filename*	Select filename for final output files
-runhdr *filename*	Direct linker to use the specified runtime header file *(default: 2105_hdr.obj)*
-save-temps	Save temp files (.i, .s, .is)
-S	Stop after compiling, without assembling (.s file generated)
-U*macro*	Undefine macro
-v	Verbose—compiler prints commands issued to execute each stage
-w	Warnings off—compiler inhibits all warning messages
-W	Warnings extra—compiler issues extra warning messages
-Wall	Warnings all—compiler issues all recommended warning messages

(Note: For a complete list of switch options, refer to the ADSP-2100 Family C Tools Manual)

PROM Splitter

```
sp121 exe_file promfile -pm [-switch ...]
or sp121 exe_file promfile -dm [-switch ...]
or sp121 exe_file promfile -bm [-switch ...]
or sp121 exe_file promfile -loader [-s] [-i]
```

Switches

-pm	Extract program memory ROM information
-dm	Extract data memory ROM information
-bm	Extract boot memory ROM information
-s	PROM file format: Motorola S record *(default)*
-i	PROM file format: Intel Hex record
-us	PROM file format: Motorola S record byte stream *(do not use with –bm)*
-us2	PROM file format: Motorola S2 record byte stream *(do not use with –bm)*
-ui	PROM file format: Intel Hex record byte stream *(do not use with –bm)*
-bs *pagesize*	Select boot page size *(default=2K)*
-bb *boundary*	Select boot page boundaries *(default=2K)*
-loader	Generate boot code with memory loading routines

HIP Splitter

```
hsp121 exe_file  [start_addr] [-i] [-boot]
```

Switches

-i	PROM file format: Intel Hex record *(default: -s)*
-boot	Ordering of 24-bit program memory words: High byte, Low byte, Middle byte *(default: High byte, Middle byte, Low byte)*

System Builder Directives

```
.SYSTEM system_name;
.ADSP21xx;                                   21xx= 2100, 2101, 2105, 2111, 2150,
.MMAPx;      (x=0,1)                               2101P (2101 paged memory system),
.CONST constant_name=expression;                   2101MV (2101 memory variant—use for 2115, 2161)
.SEG/qualifier/qualifier/... seg_name[length];
.PORT/qualifier/qualifier port_name;
.ENDSYS;
```

.SEG qualifiers: PM, DM, BOOT=0,1,2,3,4,5,6,7
RAM, ROM
ABS=*address*
DATA, CODE, DATA/CODE
EMULATOR, TARGET
INTERNAL *(for ADSP2101MV)*

.PORT qualifiers: PM, DM
ABS=*address*

Assembler Directives

```
.MODULE/qualifier/qualifier/... module_name;
.PAGE;
.CONST constant_name=expression;
.VAR/qualifier/qualifier/... buffer_name[length],...;
.INIT buffer_name: init_values;
.GLOBAL buffer_name,...;
.ENTRY program_label,...;
.EXTERNAL external_symbol,...;
.PORT port_name;
.INCLUDE <filename>;
.DMSEG dmseg_name;
.MACRO macro_name (param1, param2, ...);
.ENDMACRO;
.LOCAL macro_label,...;
.NEWPAGE;              Insert pagebreak in .LST file
.PAGELENGTH #lines;    Insert pagebreaks every #lines in .LST file
.LEFTMARGIN #columns;  Set left margin in .LST file
.INDENT #columns;      Indent code #columns in .LST file
.PAGEWIDTH #columns;   Set right margin in .LST file
.ENDMOD;
```

.MODULE qualifiers: RAM, ROM
ABS=*address (do not use with STATIC)*
SEG=*seg_name*
BOOT=0,1,2,3,4,5,6,7
STATIC

.VAR qualifiers: PM, DM,
RAM, ROM
ABS=*address (do not use with STATIC)*
SEG=*seg_name*
CIRC
STATIC

.INIT init_values: constant, constant, ...
<*filename*>
^*other_buffer*
%*other_buffer*

Assembler C Preprocessor Directives

```
#define macro_name (param1, ...) expression   Define a macro
#undef macro_name                             Undefine a macro
#include "filename"                           Include text from another source file

#if expression        Conditionally include and assemble, depending on the value of
                        an expression that evaluates to a constant
#else                 Include and assemble if the previous #if test failed
#endif

#ifdef macro_name     Conditionally include and assemble, if macro_name is defined (with #define or –d switch)
#ifndef macro_name    Conditionally include and assemble, if macro_name is not defined
#else                 Include and assemble if the previous #ifdef or #ifndef test failed
#endif
```

Instruction Set Summary

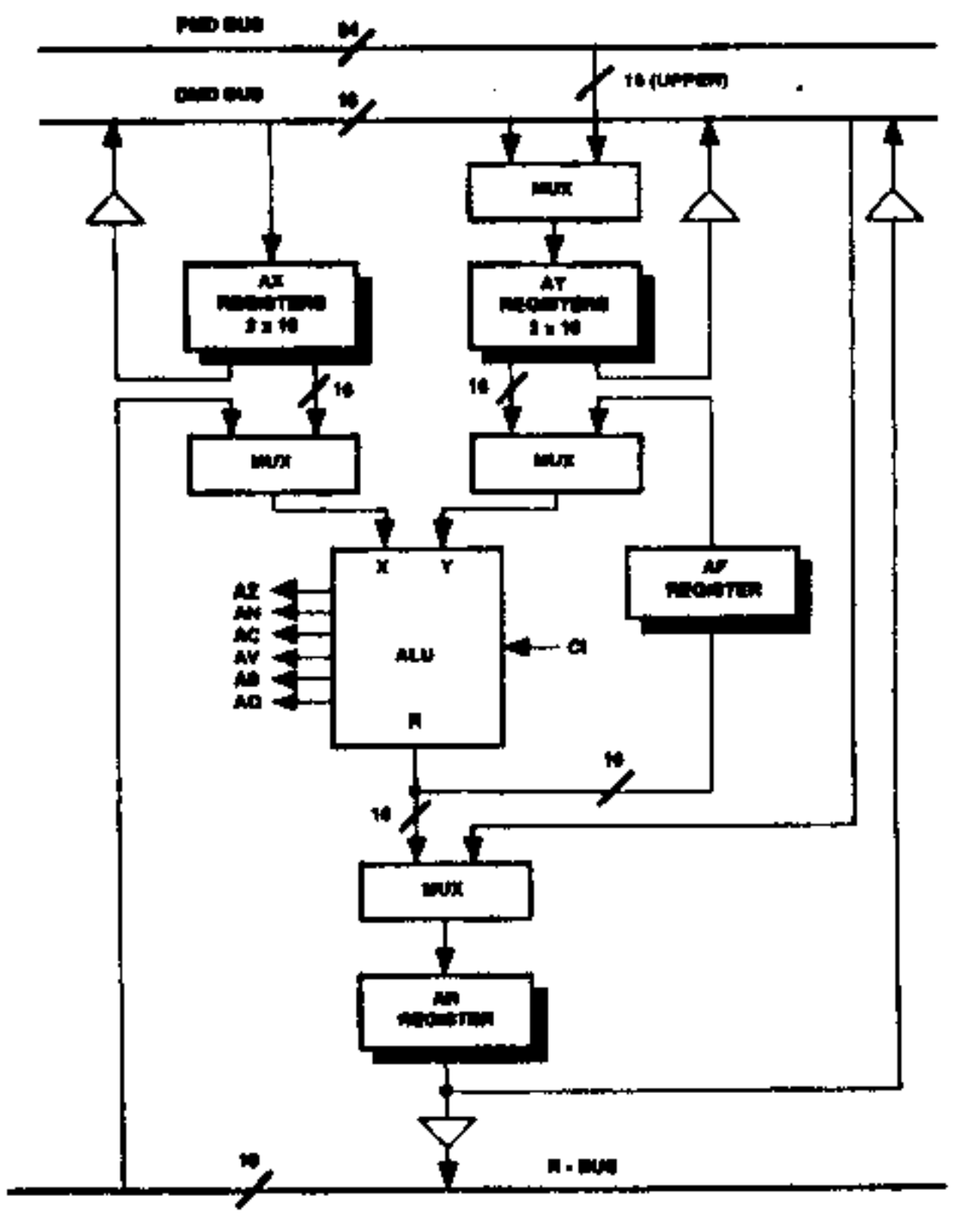
ALU Instructions

```
[IF cond] |AR| = xop + |yop|                ;   Add/Add with Carry
          |AF|      |C  |
                    |yop + C|

          = xop     |-yop|                   ;   Subtract X–Y/Subtract X–Y with Borrow
                    |-yop + C - 1|
                    |+C - 1|

          = |yop  -  |xop|                   ;   Subtract Y–X/Subtract Y–X with Borrow
            |       |xop + C - 1|
            | -xop + C - 1      |

          = xop |AND| yop                     ;   AND, OR, XOR
                |OR |
                |XOR|

          = PASS |xop|                       ;   Pass, Clear
                 |yop|
                 |-1 |
                 | 0 |
                 | 1 |

          = -    |xop|                       ;   Negate
                 |yop|

          = NOT  |xop|                       ;   NOT
                 |yop|
                 | 0 |

          = ABS  xop;                        ;   Absolute Value

          = yop  + 1;                        ;   Increment

          = yop  - 1;                        ;   Decrement

          = DIVS yop, xop;                   ;   Divide
          = DIVQ xop;
```

Notation Conventions

UPPERCASE	Explicit syntax—must be entered exactly as shown (either lowercase or uppercase may be used, however)
I0–I7	Index registers for indirect addressing
M0–M7	Modify registers for indirect addressing
L0–L7	Length registers for circular buffers (set to 0 for non-circular indirect addressing)
<data>	Immediate data value
<addr>	Immediate address value (absolute address or program label)
<exp>	Exponent (shift value) in shift immediate instructions (8-bit signed number)
cond	Condition code for conditional instruction
term	Termination code for DO UNTIL loop
dreg	Data register (of ALU, MAC, or Shifter)
reg	Any register (including dregs)
;	A semicolon terminates the instruction
,	Commas separate multiple operations of a single instruction
[ ]	Brackets enclose optional parts of instruction
[, ...]	Indicates multiple operations of an instruction that may be combined in any order, separated by commas.
|option1| |option2| |option3|	List of options; choose one.

ALU Block Diagram



IF Condition Codes

Cond	
EQ	Equal zero
NE	Not equal zero
LT	Less than zero
GE	Greater than or equal zero
LE	Less than or equal zero
GT	Greater than zero
AC	ALU carry
NOT AC	Not ALU carry
AV	ALU overflow
NOT AV	Not ALU overflow
MV	MAC overflow
NOT MV	Not MAC overflow
NEG	Xop input sign negative
POS	Xop input sign positive
NOT CE	Not counter expired
FLAG_IN *	FI pin=1
NOT FLAG_IN *	FI pin=0

** Only for JUMP, CALL*

Allowed XOP, YOP Registers for ALU Instructions

xop	AX0, AX1 AR MR0, MR1, MR2 SR0, SR1
yop	AY0*, AY1 AF

* DIVS instruction may not use AY0 as YOP operand.

Instruction Set Summary

MAC Instructions

[IF cond]	MR / MF = xop * yop	(SS) (SU) (US) (UU) (RND) ;	*Multiply*
	= MR + xop * yop	(SS) (SU) (US) (UU) (RND) ;	*Multiply/Accumulate*
	= MR − xop * yop	(SS) (SU) (US) (UU) (RND) ;	*Multiply/Subtract*
	= MR [(RND)];		*Transfer MR*
	= 0;		*Clear*
IF MV SAT MR;			*Conditional MR Saturation*

(S) Signed input (xop, yop)
(U) Unsigned input (xop, yop)
(RND) Rounded output

IF Condition Codes

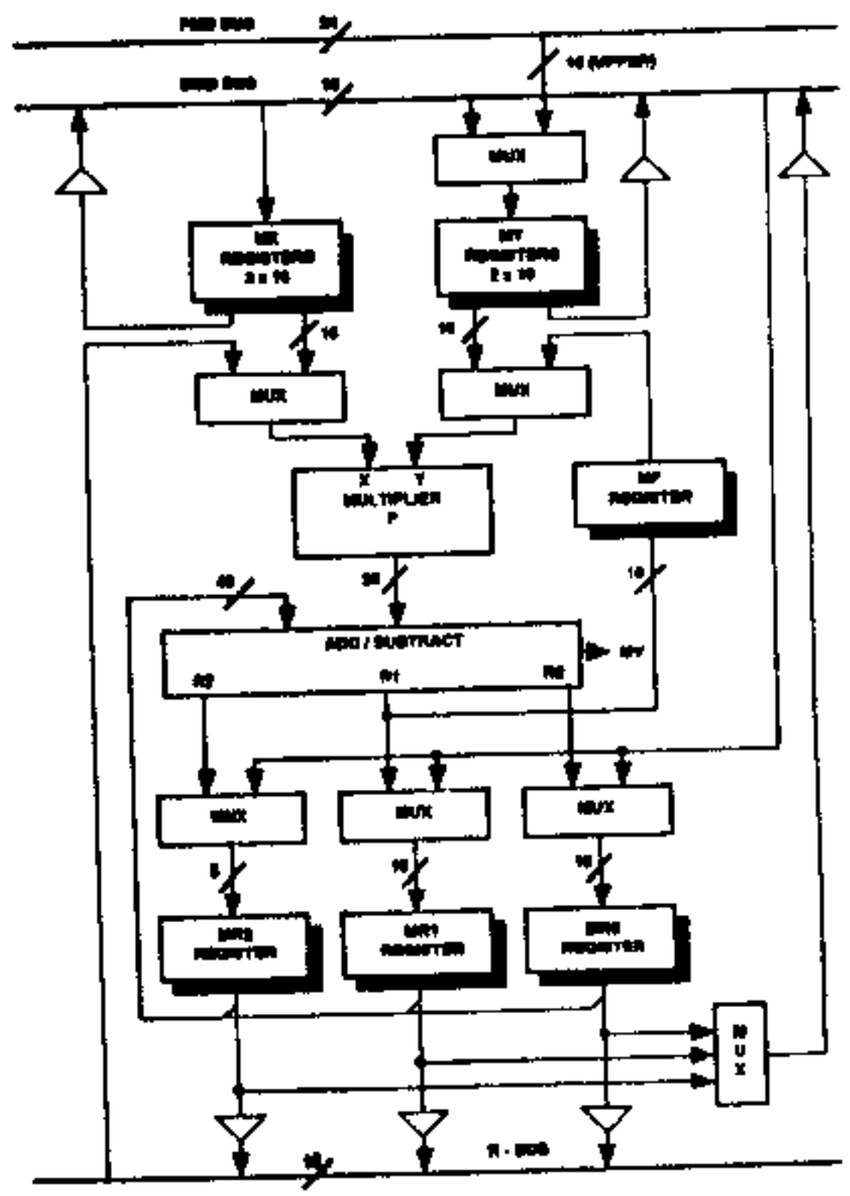
Cond	
EQ	Equal zero
NE	Not equal zero
LT	Less than zero
GE	Greater than or equal zero
LE	Less than or equal zero
GT	Greater than zero
AC	ALU carry
NOT AC	Not ALU carry
AV	ALU overflow
NOT AV	Not ALU overflow
MV	MAC overflow
NOT MV	Not MAC overflow
NEG	Xop input sign negative
POS	Xop input sign positive
NOT CE	Not counter expired
FLAG_IN *	FI pin=1
NOT FLAG_IN *	FI pin=0

** Only for JUMP, CALL*

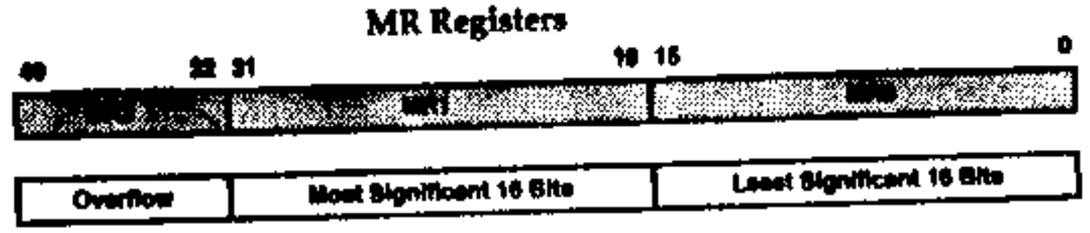
Allowed XOP, YOP Registers for MAC Instructions

xop	MX0, MX1 MR0, MR1, MR2 AR SR0, SR1
yop	MY0, MY1 MF

MAC Block Diagram



MR Registers



Instruction Set Summary

Shifter Instructions

[IF cond] SR = [SR OR] ASHIFT xop (HI)/(LO) ;	*Arithmetic Shift*
[IF cond] SR = [SR OR] LSHIFT xop (HI)/(LO) ;	*Logical Shift*
[IF cond] SR = [SR OR] NORM xop (HI)/(LO) ;	*Normalize*
[IF cond] SE = EXP xop (HI)/(LO)/(HIX) ;	*Derive Exponent*
[IF cond] SB = EXPAD] xop ;	*Block Exponent Adjust*
SR = [SR OR] ASHIFT xop BY <exp> (HI)/(LO) ;	*Arithmetic Shift Immediate*
SR = [SR OR] LSHIFT xop BY <exp> (HI)/(LO) ;	*Logical Shift Immediate*

(HI) Shift is referenced to SR1 (most significant 16 bits)
(LO) Shift is referenced to SR0 (least significant 16 bits)
(HIX) HI extend (AV overflow bit read by exponent detector)

IF Condition Codes

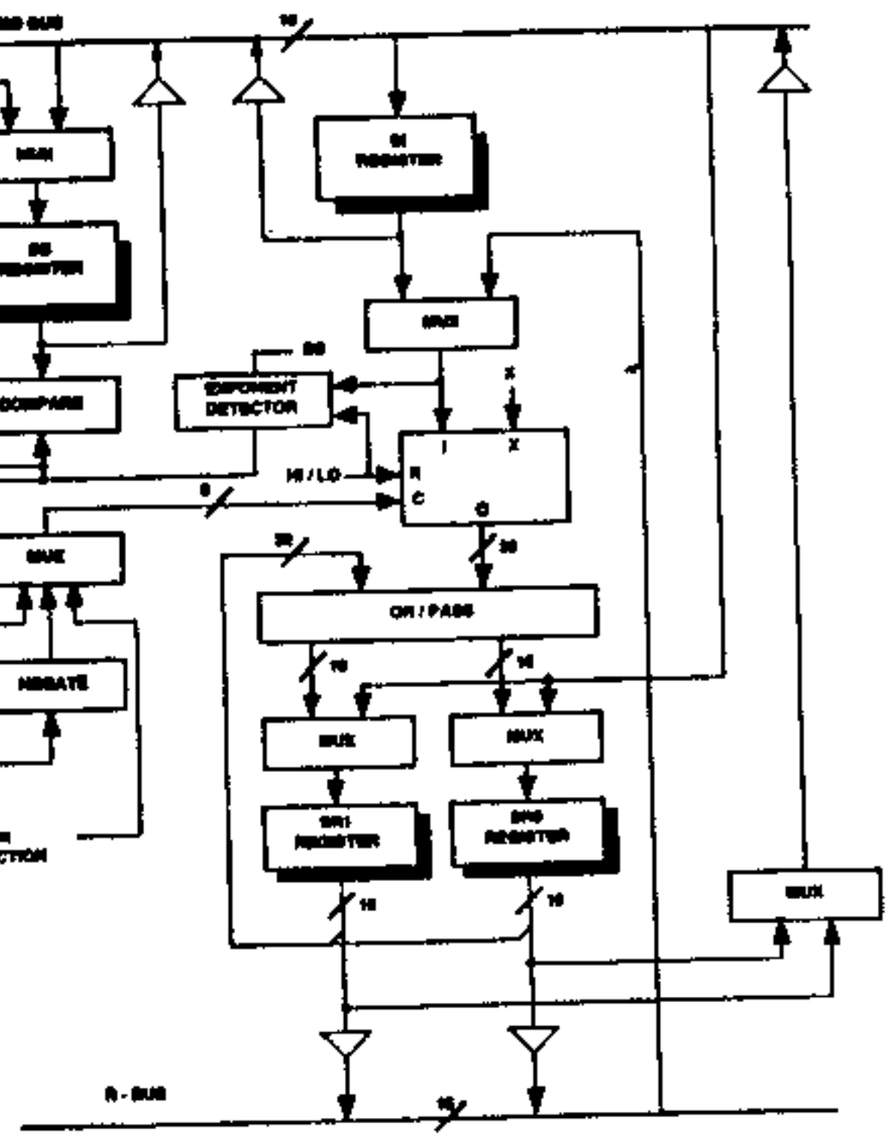
Cond	
EQ	Equal zero
NE	Not equal zero
LT	Less than zero
GE	Greater than or equal zero
LE	Less than or equal zero
GT	Greater than zero
AC	ALU carry
NOT AC	Not ALU carry
AV	ALU overflow
NOT AV	Not ALU overflow
MV	MAC overflow
NOT MV	Not MAC overflow
NEG	Xop input sign negative
POS	Xop input sign positive
NOT CE	Not counter expired
FLAG_IN *	FI pin=1
NOT FLAG_IN *	FI pin=0

** Only for JUMP, CALL*

Allowed XOP Registers for Shifter Instructions

xop	SI, SR0, SR1 AR MR0, MR1, MR2

Shifter Block Diagram



Instruction Set Summary

Data Move Instructions

`reg = reg;`	*Register-to-Register Move*
`reg = <data>;` `dreg = <data>;`	*Load Register Immediate*
`reg = DM (<addr>);`	*Data Memory Read (Direct Address)*
`dreg = DM ( I0 | I1 | I2 | I3 , M0 | M1 | M2 | M3 );` `I4 | I5 | I6 | I7 , M4 | M5 | M6 | M7`	*Data Memory Read (Indirect Address)*
`dreg = PM ( I4 | I5 | I6 | I7 , M4 | M5 | M6 | M7 );`	*Program Memory Read (Indirect Address)*
`DM (<addr>) = reg;`	*Data Memory Write (Direct Address)*
`DM ( I0 | I1 | I2 | I3 , M0 | M1 | M2 | M3 ) = dreg;` `I4 | I5 | I6 | I7 , M4 | M5 | M6 | M7`	*Data Memory Write (Indirect Address)*
`PM ( I4 | I5 | I6 | I7 , M4 | M5 | M6 | M7 ) = dreg;`	*Program Memory Write (Indirect Address)*

Allowed Registers for Data Move & Multifunction Instructions

- **dreg** (data registers):
  - AX0, AX1
  - AY0, AY1
  - AR
  - MX0, MX1
  - MY0, MY1
  - MR0, MR1, MR2
  - SI, SE, SR0, SR1
- **reg**:
  - I0, I1, I2, I3, I4, I5, I6, I7
  - M0, M1, M2, M3, M4, M5, M6, M7
  - L0, L1, L2, L3, L4, L5, L6, L7
  - TX0, TX1, RX0, RX1
  - SB, PX
  - ASTAT, MSTAT
  - SSTAT *(read-only)*
  - IMASK, ICNTL
  - IPC *(write-only)*
  - CNTR
  - OWRCNTR *(write-only)*

Multifunction Instructions

`<ALU> | <MAC> | <SHIFT> , dreg = dreg;`	*Computation with Register-to-Register Move*

`<ALU> | <MAC> | <SHIFT> , dreg = DM ( I0 | I1 | I2 | I3 , M0 | M1 | M2 | M3 );`
`I4 | I5 | I6 | I7 , M4 | M5 | M6 | M7`
`PM ( I4 | I5 | I6 | I7 , M4 | M5 | M6 | M7 );`
	Computation with Memory Read

`DM ( I0 | I1 | I2 | I3 , M0 | M1 | M2 | M3 ) = dreg , <ALU> | <MAC> | <SHIFT>;`
`I4 | I5 | I6 | I7 , M4 | M5 | M6 | M7`
`PM ( I4 | I5 | I6 | I7 , M4 | M5 | M6 | M7 );`
	Computation with Memory Write

`AX0 | AX1 | MX0 | MX1 = DM ( I0 | I1 | I2 | I3 , M0 | M1 | M2 | M3 ) ,` `AY0 | AY1 | MY0 | MY1 = PM ( I4 | I5 | I6 | I7 , M4 | M5 | M6 | M7 );`	*Data & Program Memory R*
`<ALU> | <MAC> , AX0 | AX1 | MX0 | MX1 = DM ( I0 | I1 | I2 | I3 , M0 | M1 | M2 | M3 ) ,` `AY0 | AY1 | MY0 | MY1 = PM ( I4 | I5 | I6 | I7 , M4 | M5 | M6 | M7 );`	*ALU/MAC with Data & P Memory Read*

<ALU>*† Any ALU instruction (except DIVS, DIVQ)
<MAC>*† Any multiply/accumulate instruction
<SHIFT>* Any shifter instruction (except Shift Immediate)

* *May not be conditional instructions*
† *AR, MR result registers must be used—not AF, MF feedback registers*

Instruction Set Summary

Program Flow Instructions

DO <addr> [UNTIL term]; — *Do Until*

[IF cond] JUMP (I4) (I5) (I6) (I7) <addr>; — *Jump*

[IF cond] CALL (I4) (I5) (I6) (I7) <addr>; — *Call Subroutine*

IF FLAG_IN / NOT FLAG_IN JUMP / CALL <addr>; — *Jump/Call on Flag In Pin*

[IF cond] SET / RESET / TOGGLE FLAG_OUT / FL0 / FL1 / FL2 [,...]; — *Modify Flag Out Pin*

[IF cond] RTS; — *Return from Subroutine*

[IF cond] RTI; — *Return from Interrupt Service Routine*

IDLE [(n)]; — *Idle*
n=16, 32, 64, or 128

DO UNTIL Termination Codes

Term	
CE	Counter expired
EQ	Equal zero
NE	Not equal zero
LT	Less than zero
GE	Greater than or equal zero
LE	Less than or equal zero
GT	Greater than zero
AC	ALU carry
NOT AC	Not ALU carry
AV	ALU overflow
NOT AV	Not ALU overflow
MV	MAC overflow
NOT MV	Not MAC overflow
NEG	Xop input sign negative
POS	Xop input sign positive
FOREVER	Always

IF Condition Codes

Cond	
EQ	Equal zero
NE	Not equal zero
LT	Less than zero
GE	Greater than or equal zero
LE	Less than or equal zero
GT	Greater than zero
AC	ALU carry
NOT AC	Not ALU carry
AV	ALU overflow
NOT AV	Not ALU overflow
MV	MAC overflow
NOT MV	Not MAC overflow
NEG	Xop input sign negative
POS	Xop input sign positive
NOT CE	Not counter expired
FLAG_IN *	FI pin=1
NOT FLAG_IN *	FI pin=0

** Only for JUMP, CALL*

Miscellaneous Instructions

NOP; — *NOP*

MODIFY (I0 / I1 / I2 / I3 / I4 / I5 / I6 / I7 , M0 / M1 / M2 / M3 / M4 / M5 / M6 / M7); — *Modify Address Register*

[PUSH / POP STS] [, POP CNTR] [, POP PC] [, POP LOOP]; — *Stack Control*

ENA / DIS SEC_REG / BIT_REV / AV_LATCH / AR_SAT / M_MODE / TIMER / G_MODE [,...]; — *Mode Control*

Modes	
SEC_REG	Secondary register set
BIT_REV	Bit-reverse addressing in DAG1
AV_LATCH	ALU overflow (AV) status latch
AR_SAT	AR register saturation
M_MODE	MAC result placement mode
TIMER	Timer enable
G_MODE	Go mode enable

Circular Buffer Addressing

Next Address = (I + M – B) modulo(L) + B

I=current address M=modify value (signed) M≤L
B=base address L=buffer length

(Set L=0 for standard, non-circular indirect addressing: I+M=modified address)

Buffer Length & Base Address Operators

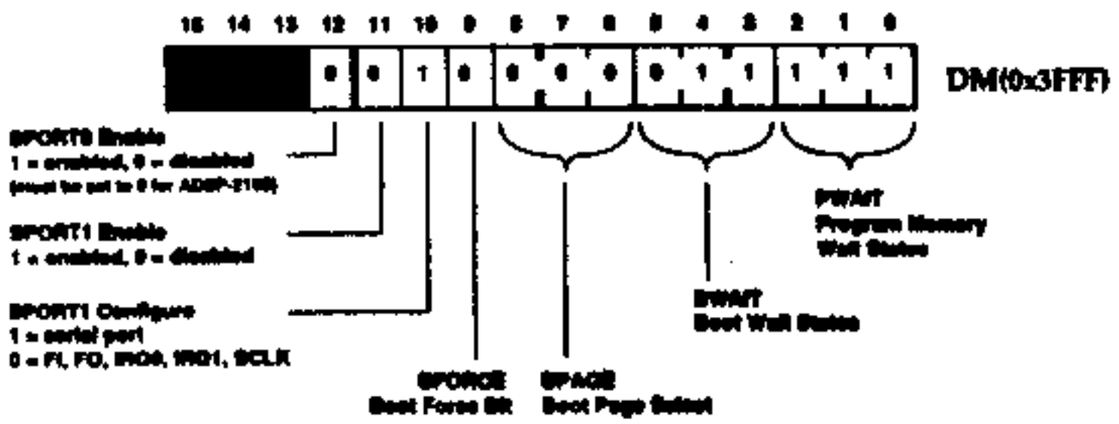
^buffer_name *Base address of buffer_name*
%buffer_name *Length (number of locations) of buffer_name*

Example: Setting Up DAG Registers for Circular Buffer & DO UNTIL Loop

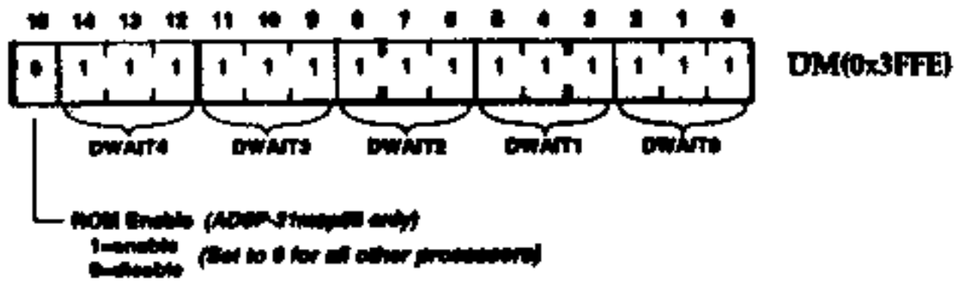
```
      .VAR/DM/RAM/CIRC real_data[n];     {n=number of input samples}
       I5=^real_data;                    {buffer base address}
       L5=%real_data;                    {buffer length}
       M5=1;                             {post-modify I5 by 1}
       CNTR=%real_data;                  {loop counter = buffer length}
       DO loop UNTIL CE;
          AX0=DM(I5,M5);                 {get next sample}
          ...
          {now process sample stored in AX0}
loop:     ...
```

Control/Status Registers

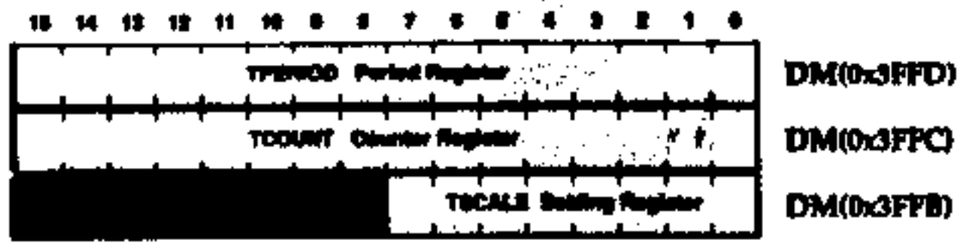
System Control Register



Data Memory Waitstate Register



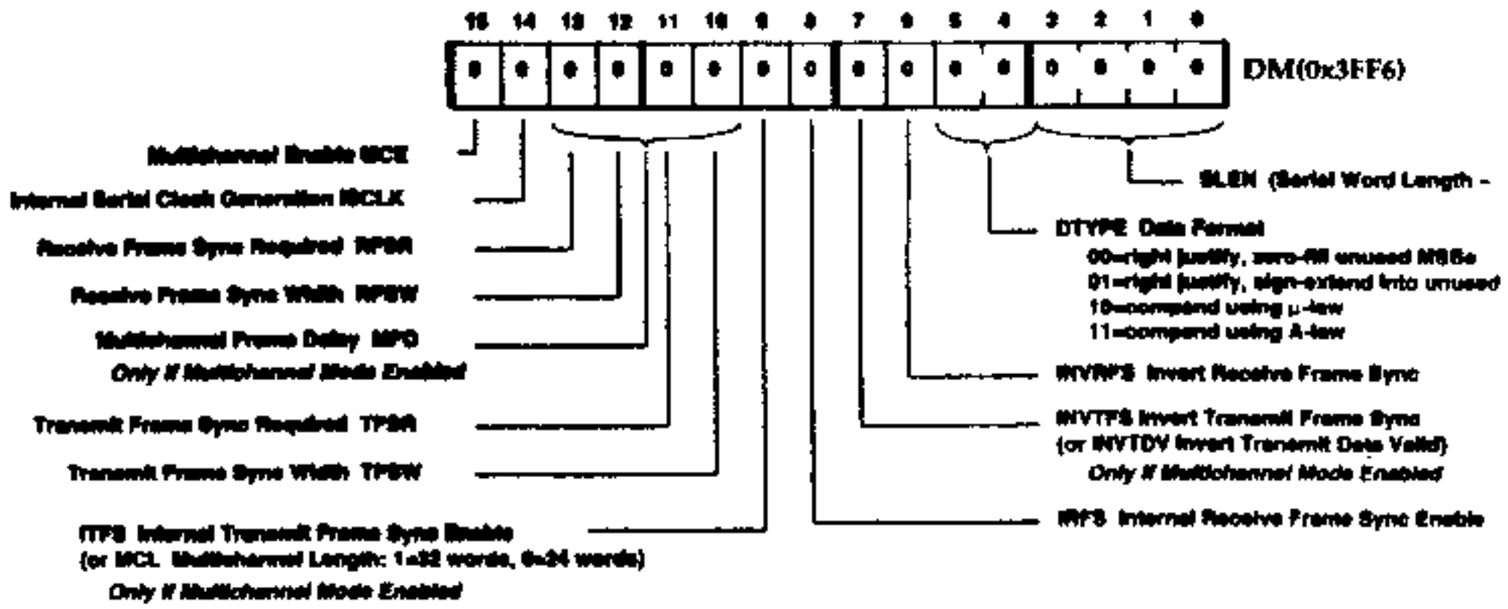
Timer Registers



Control register default bit values at reset are as shown; if no value is shown, the bit is undefined after reset. Reserved bits are shown on a gray field—these bits should always be written with zeros.

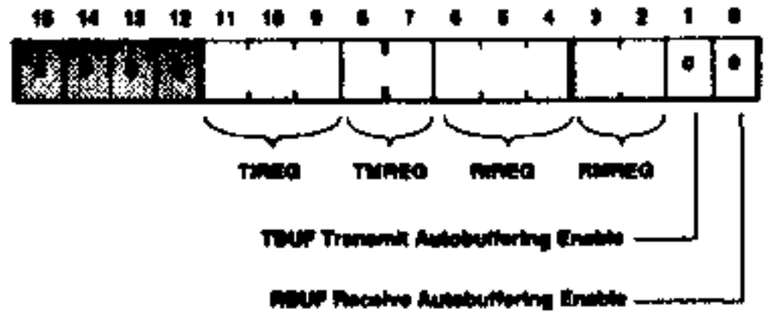
SPORT0 Control Register

Not on ADSP-2105



SPORT0 Autobuffer Control

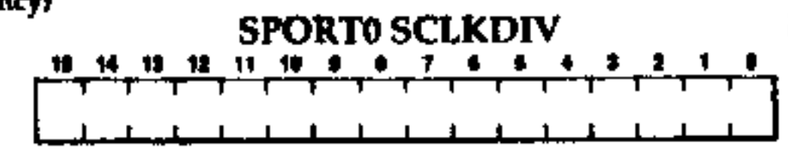
Not on ADSP-2105



$$\text{SCLKDIV} = \frac{\text{CLKOUT frequency}}{2 \times (\text{SCLK frequency})} - 1$$

SPORT0 SCLKDIV

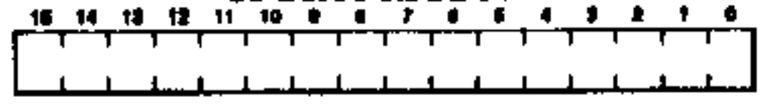
Not on ADSP-2105



$$\text{RFSDIV} = \frac{\text{SCLK frequency}}{\text{RFS frequency}} - 1$$

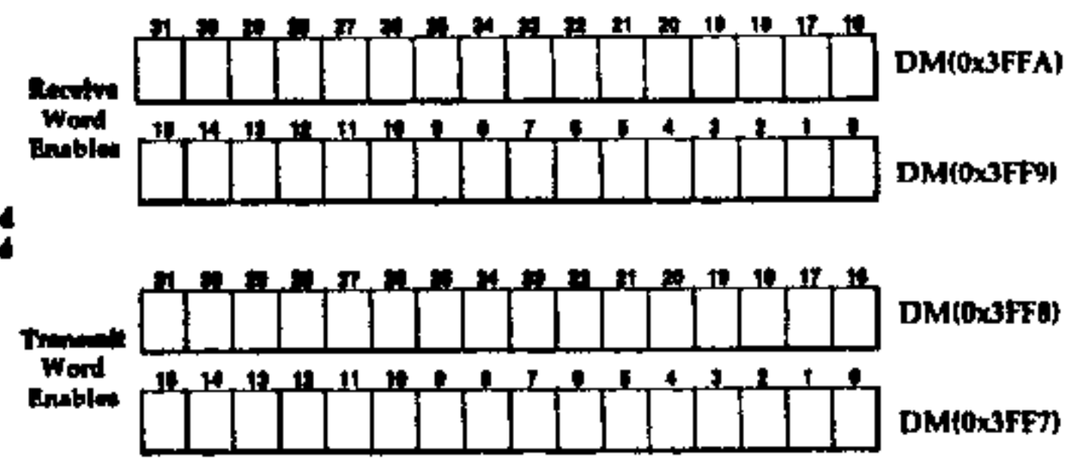
SPORT0 RFSDIV

Not on ADSP-2105

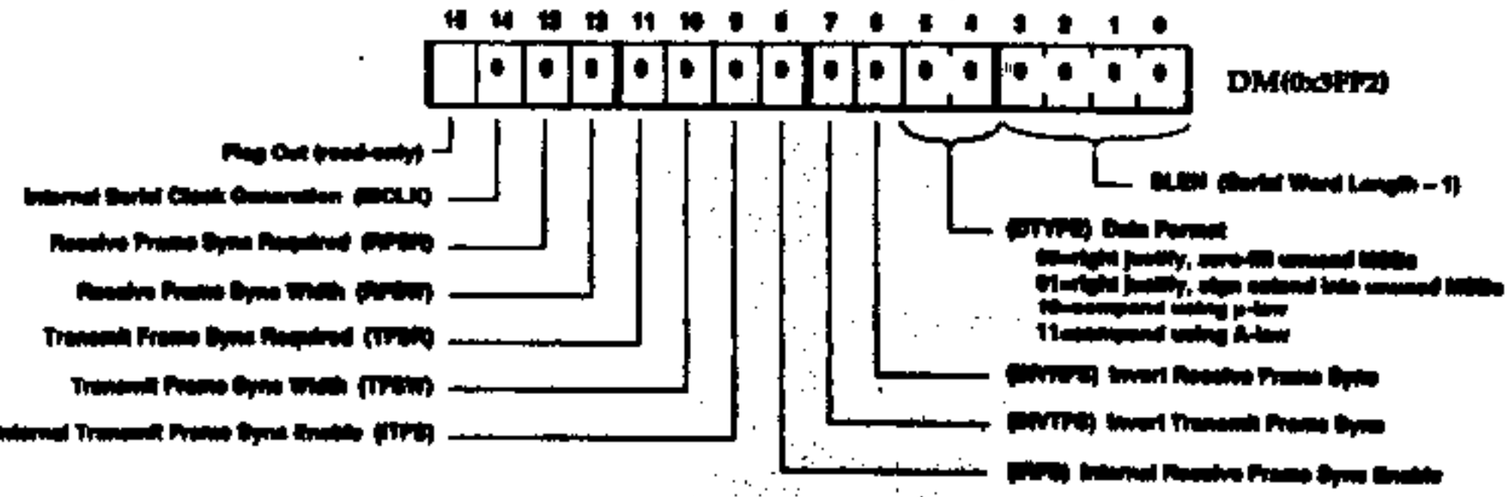


SPORT0 Multichannel Word Enables

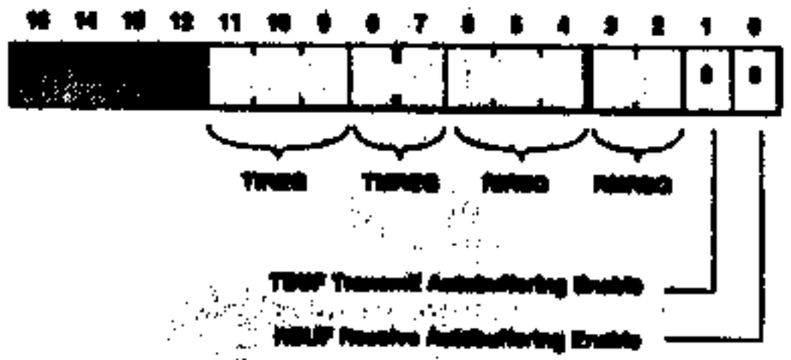
Not on ADSP-2105



SPORT1 Control Register

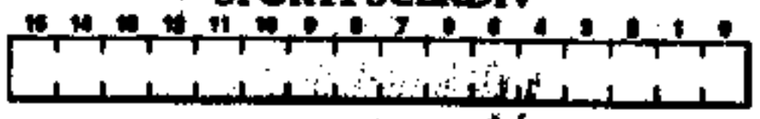


SPORT1 Autobuffer Control *Not on ADSP-21msp50*



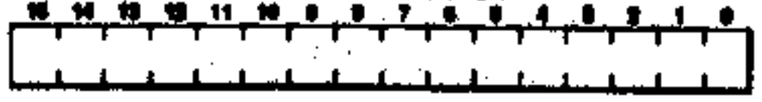
$$\text{SCLKDIV} = \frac{\text{CLKOUT frequency}}{2 \cdot (\text{SCLK frequency})} - 1$$

SPORT1 SCLKDIV

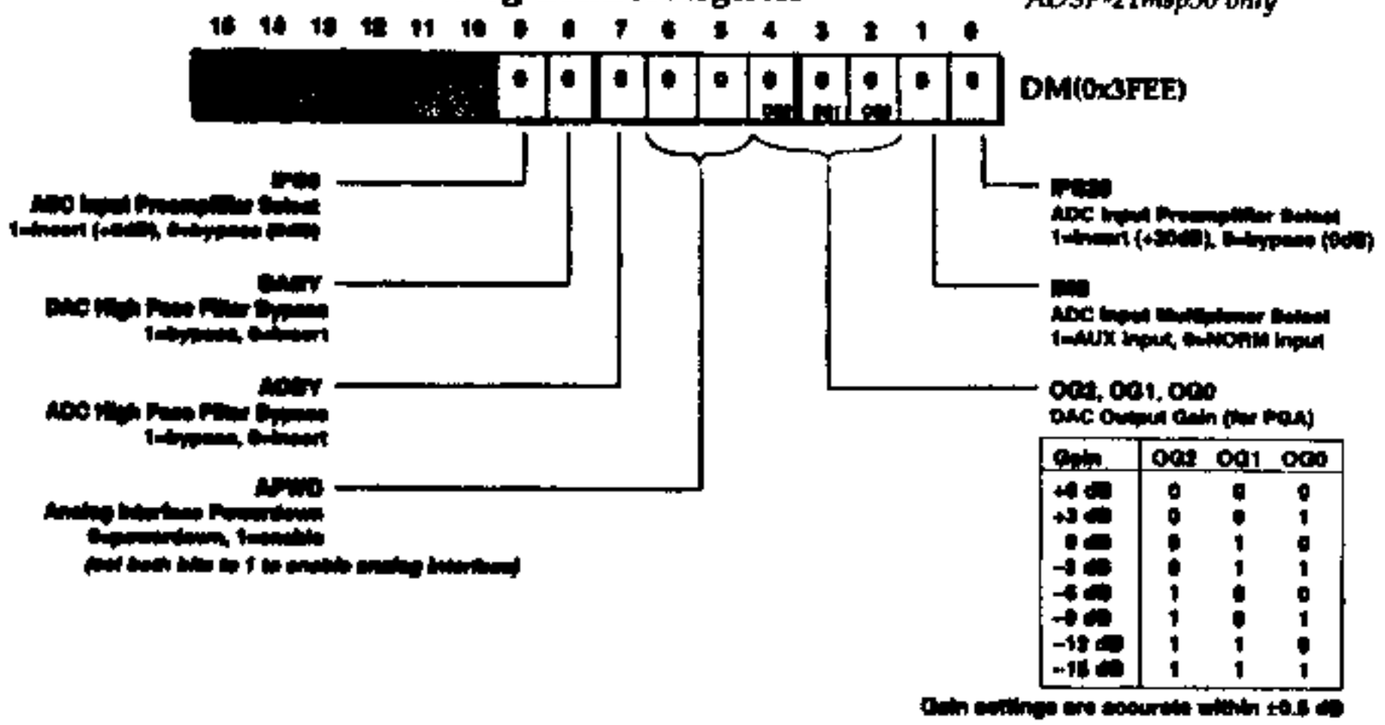


$$\text{RFSDIV} = \frac{\text{SCLK frequency}}{\text{RFS frequency}} - 1$$

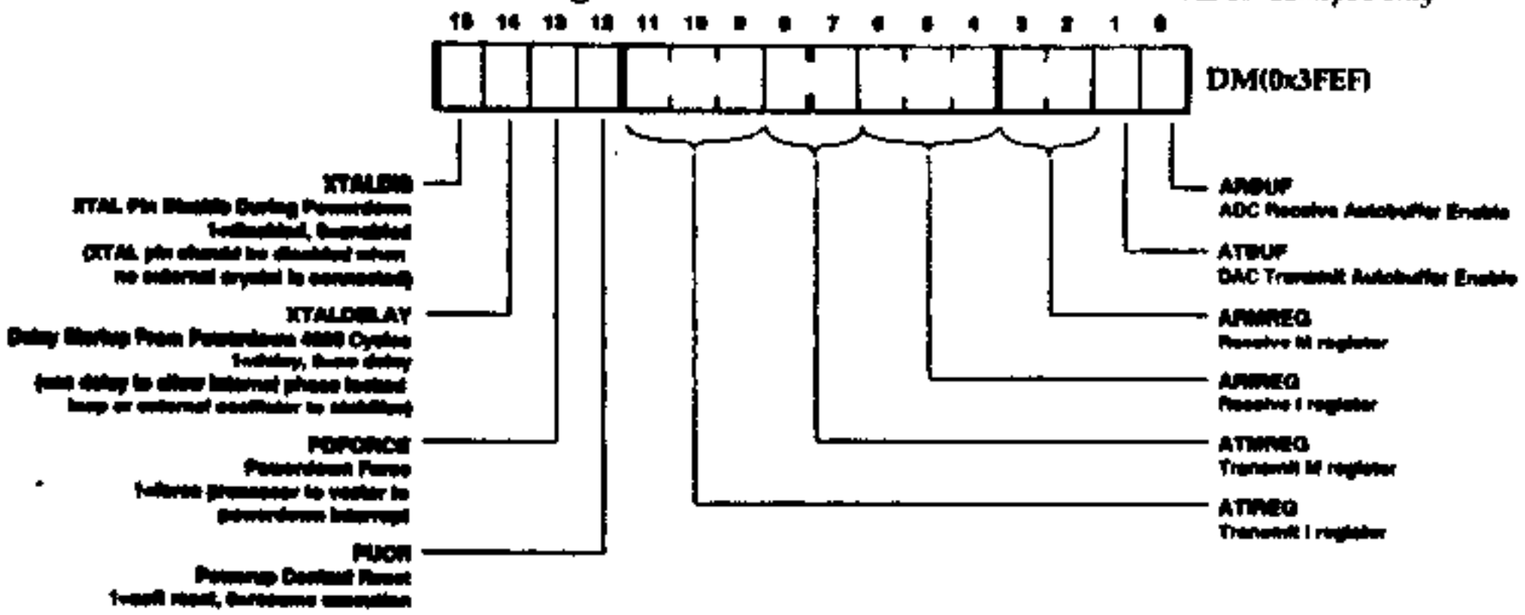
SPORT1 RFSDIV



Analog Control Register *ADSP-21msp50 only*

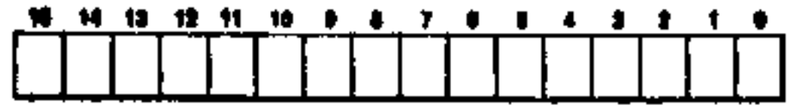


Analog Autobuffer/Powerdown Control *ADSP-21msp50 only*



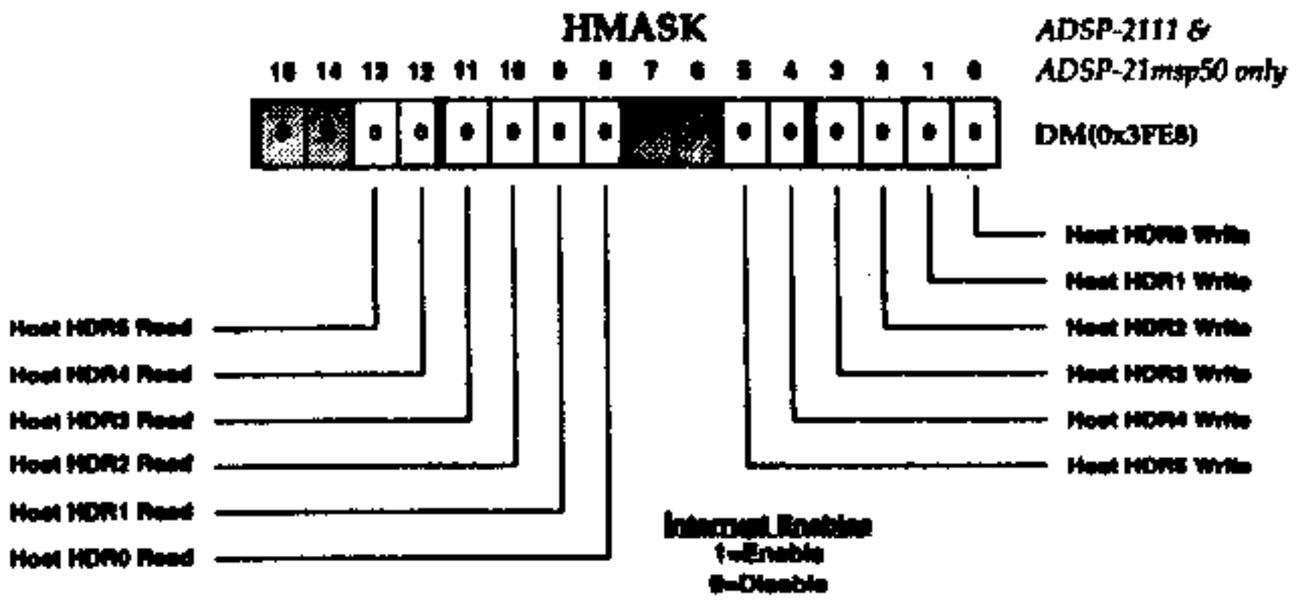
ADC Receive Data *ADSP-21msp50 only*

DAC Transmit Data *ADSP-21msp50 only*

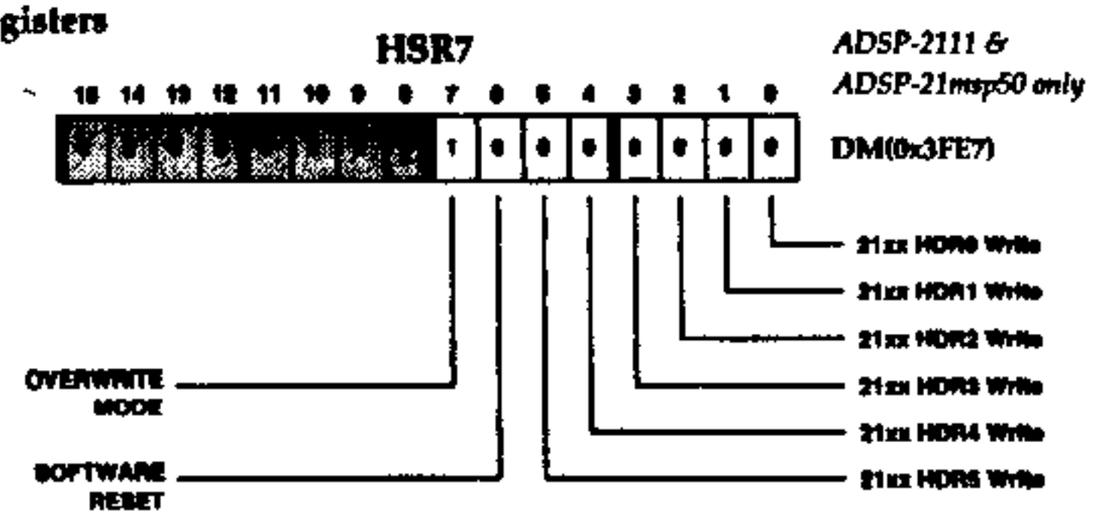


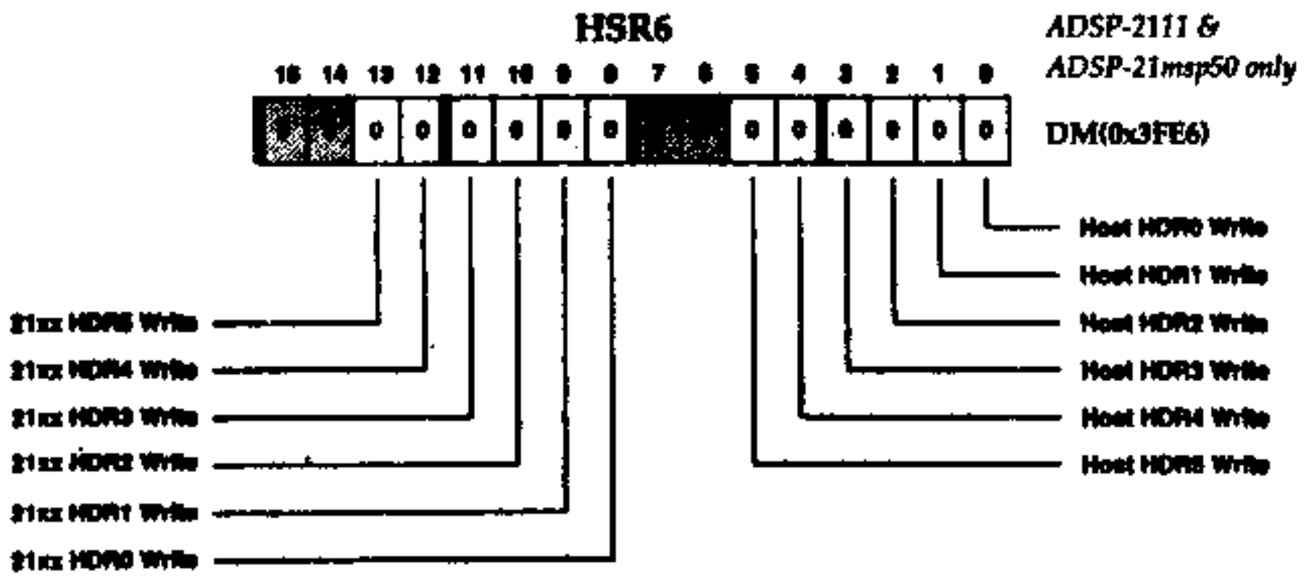
No registers are located between DM(0x3FEC) and DM(0x3FE8)

HIP Interrupt Mask Register

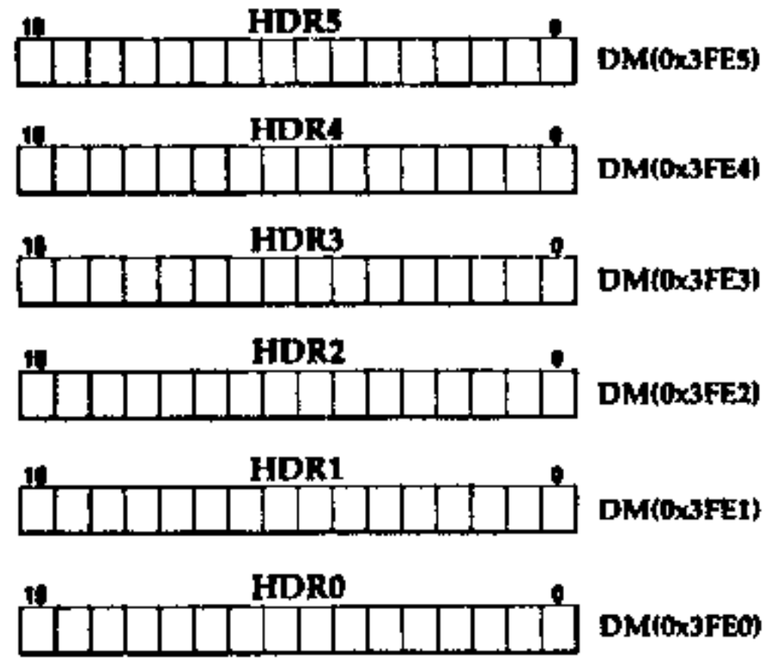


HIP Status Registers



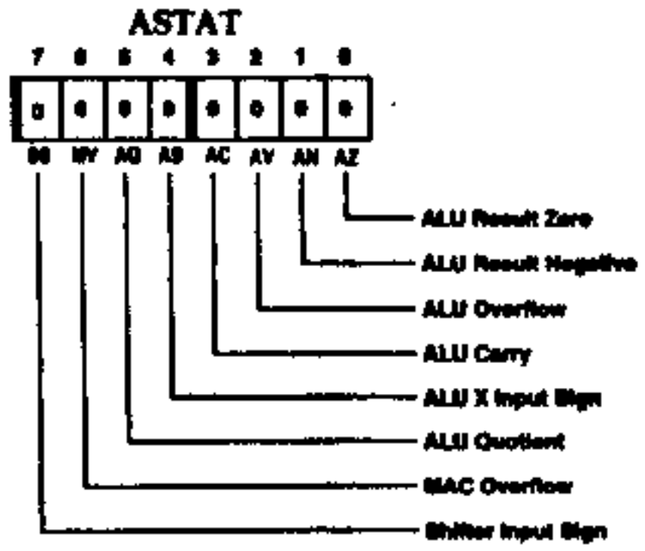


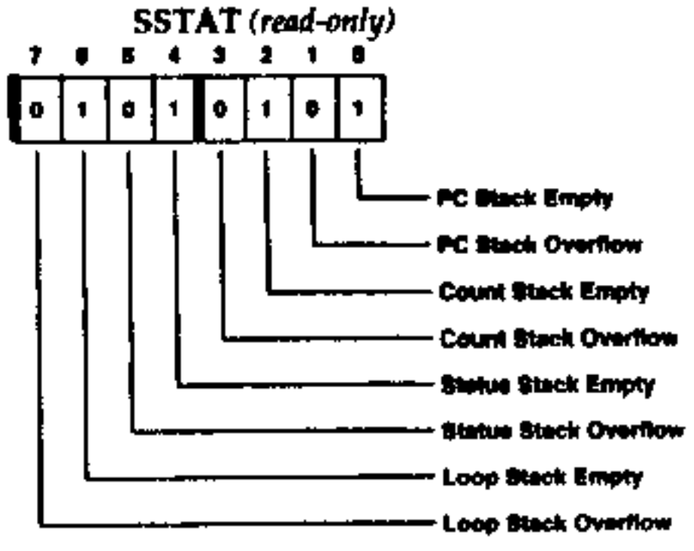
HIP Data Registers

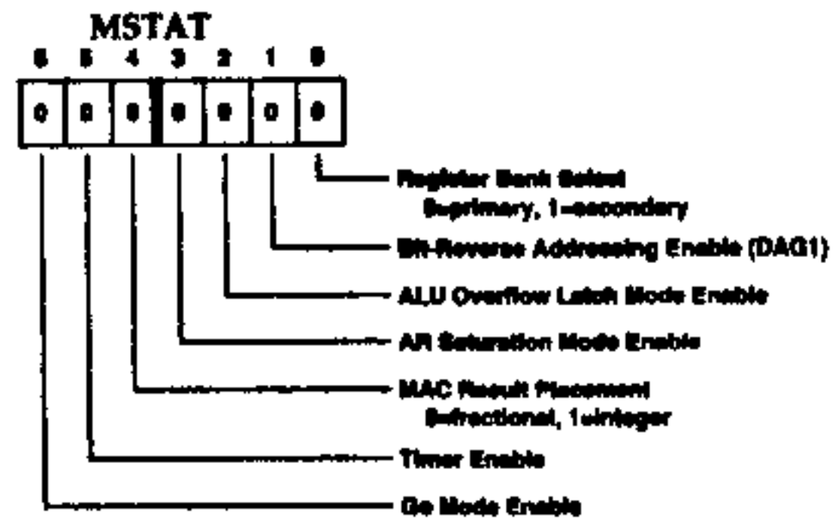


ADSP-2111 &
ADSP-21msp50 onl

Status Registers
(Non-Memory-Mapped)







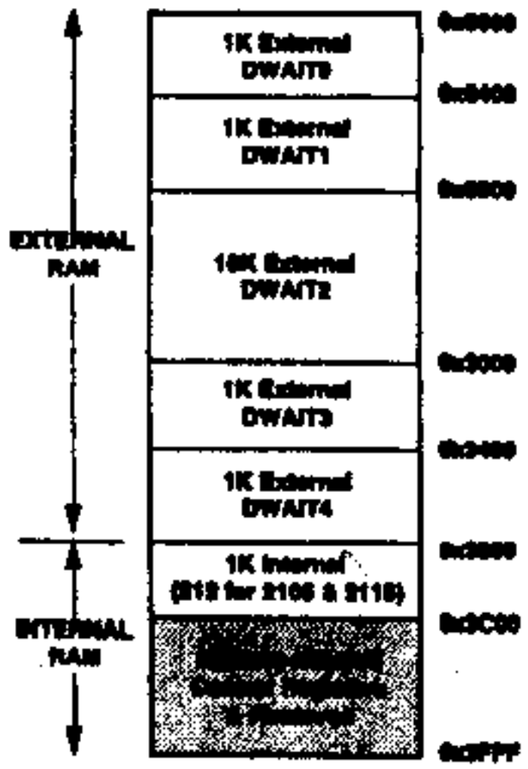
Mode Names for Mode Control Instructi

(see Miscellaneous Instructions on page 13)

Bit	Mode Name	
0	SEC_REG	Secondary register set
1	BIT_REV	Bit-reverse addressing in DAG1
2	AV_LATCH	ALU overflow (AV) status latch
3	AR_SAT	AR register saturation
4	M_MODE	MAC result placement mode
5	TIMER	Timer enable
6	G_MODE	Go mode enable

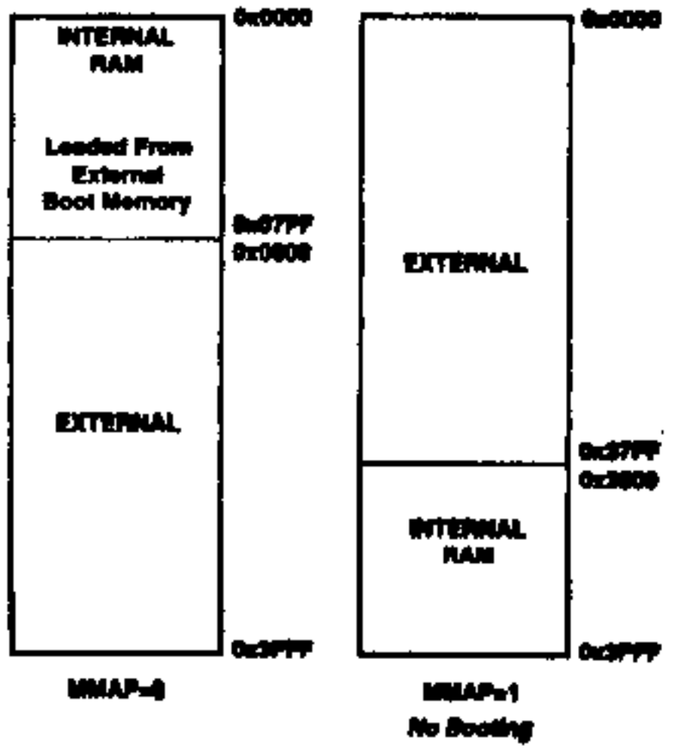
Memory Maps

Data Memory



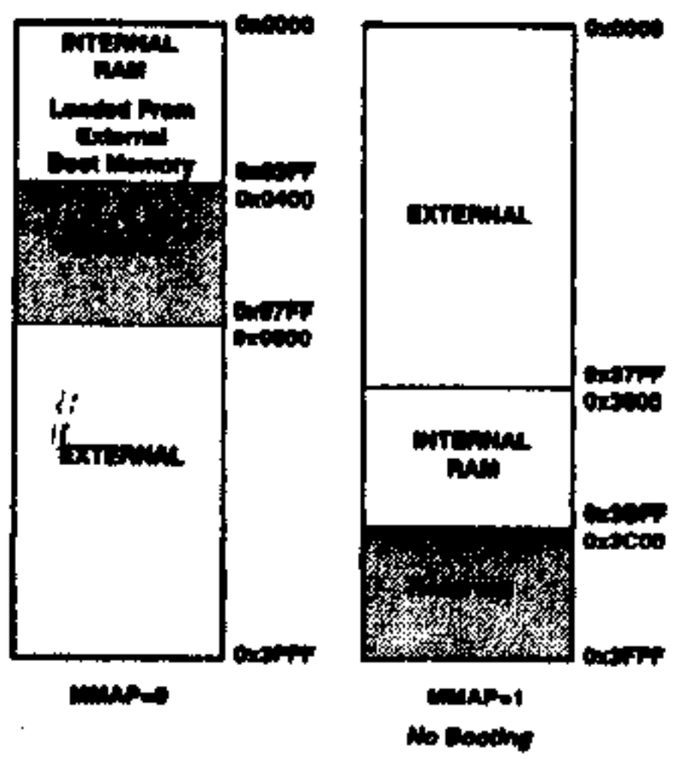
ADSP-2101
ADSP-2111

Program Memory *ADSP-21msp50*



ADSP-2105
ADSP-2115

Program Memory



Interrupt Vector Tables

ADSP-2100

Interrupt Source	*Interrupt Vector Address*
IRQ0	0x0000
IRQ1	0x0001
IRQ2	0x0002
IRQ3	0x0003
RESET startup	0x0004

ADSP-2101/2115

Interrupt Source	*Interrupt Vector Address*
RESET startup	0x0000
IRQ2	0x0004 *(high priority)*
SPORT0 Transmit	0x0008
SPORT0 Receive	0x000C
SPORT1 Transmit/IRQ1	0x0010
SPORT1 Receive/IRQ0	0x0014
Timer	0x0018 *(low priority)*

ADSP-2105

Interrupt Source	*Interrupt Vector Address*
RESET startup	0x0000
IRQ2	0x0004 *(high priority)*
SPORT1 Transmit/IRQ1	0x0010
SPORT1 Receive/IRQ0	0x0014
Timer	0x0018 *(low priority)*

ADSP-2111

Interrupt Source	*Interrupt Vector Address*
RESET startup	0x0000
IRQ2	0x0004 *(high priority)*
HIP Write from Host	0x0008
HIP Read to Host	0x000C
SPORT0 Transmit	0x0010
SPORT0 Receive	0x0014
SPORT1 Transmit/IRQ1	0x0018
SPORT1 Receive/IRQ0	0x001C
Timer	0x0020 *(low priority)*

ADSP-21msp50

Interrupt Source	*Interrupt Vector Address*
RESET *(or powerup w/PUCR=1)*	0x0000
Powerdown *(non-maskable)*	0x002C *(high priority)*
IRQ2	0x0004
HIP Write from Host	0x0008
HIP Read to Host	0x000C
SPORT0 Transmit	0x0010
SPORT0 Receive	0x0014
Analog Transmit (DAC)	0x0018
Analog Receive (ADC)	0x001C
SPORT1 Transmit/IRQ1	0x0020
SPORT1 Receive/IRQ0	0x0024
Timer	0x0028 *(low priority)*

SBP 9708 8-BIT-BY-8-BIT PARALLEL BYTE MULTIPLIER

FEATURES

- Signed or Unsigned 8-Bit Multiplier Produces 16-Bit Product
- On-Chip Latches Simplify System Design
- Transparent or Latched (Pipelined) Operating Modes
- Bus Compatible With Popular 8-Bit Microprocessors
- Supports Systems With Bus Data Rates Up to 4 MHz
- Standard 16-Pin Packaging Maximizes Boards Density
- Compatible With TTL and Low-Threshold MOS
- Choice of Ambient Temperature Performance Ranges:

  SBP 9708M . . . –55°C to 125°C
  SBP 9708E . . . –40°C to 85°C
  SBP 9708C . . . 0°C to 70°C

DESCRIPTION

This 8-bit by 8-bit I^2L parallel multiplier provides a 16-bit signed or unsigned product resulting from two signed 2's complement or unsigned 8-bit bytes. Designed specifically to achieve the cost-effectiveness needed for microprocessor based systems, the SBP 9708 combines the static logic of I^2L with a pin-efficient organization to make available a low-cost byte-oriented peripheral multiplier that can improve CPU throughput rates and reduce software overhead significantly over iterative algorithms.

Sequential byte-parallel operations can be performed in either a transparent or latched (pipelined) multiply mode. In the transparent mode, the multiplier/multiplicand latches accept the two operators, the 16-bit product bypasses the output latches and is available in two of the multiplier array bytes directly from the 8-bit 4-to-1 multiplexer. This mode is best-suited when the need is for fast, isolated multiply occurrences. In the latched mode, during the first write cycle, the multiplier latch accepts new data and the previous array result is strobed into the two product latches. On the next write cycle, the multiplicand is latched. In this mode, designed for multiply-intensive systems, pipelining allows sequential multiplication operations carried out at a rate 50% faster than in the transparent mode.

In addition to single line controls for operating mode (MODE) and sign-bit handling (S/$\overline{U}$), active-low chip-select ($\overline{CS}$) and read-write (R/$\overline{W}$) inputs are configured to make the multiplier addressable from hardware and software as either a memory-mapped, or an I/O-mapped peripheral. Sequential parallel byte loading (write) and result output (reading) are steered to or from the storage latches by a single most-significant byte/least-significant byte (M/$\overline{L}$) control input.

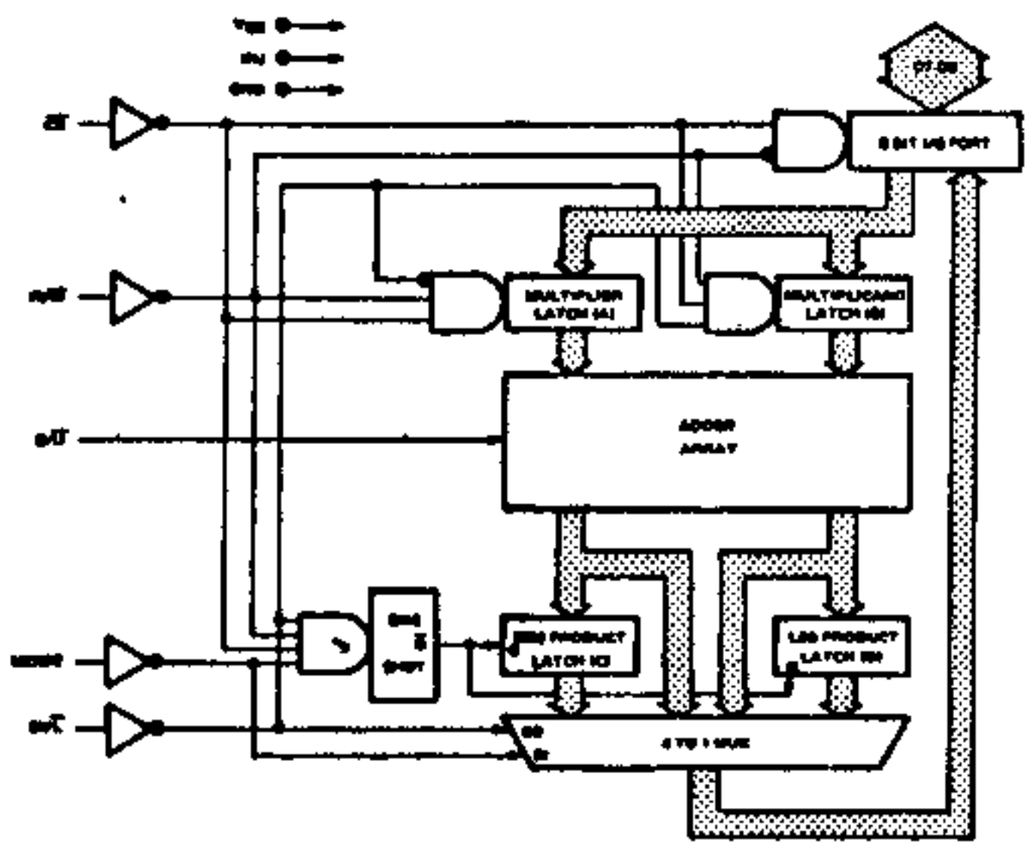
All versions are TTL and low-threshold MOS compatible. Output pullup resistors are provided on chip to minimize external components.

CONTROL FUNCTION TABLE

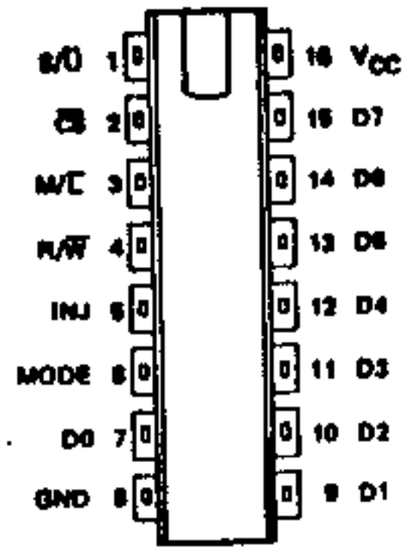
MODE	INPUT				FUNCTION
	$\overline{CS}$	R/$\overline{W}$	M/$\overline{L}$	MODE	
Transparent Multiply	H	X	X	X	I/O Disabled
	L	L	L	H	Load Multiplier Latch
	L	L	H	H	Load Multiplicand Latch
	L	H	L	H	Read LSB Product From Adder Array
	L	H	H	H	Read MSB Product From Adder Array
Latched Multiply	L	L	L	L	Load Multiplier, Latch, Strobe Adder Array Data into Product Latches
	L	L	H	L	Load Multiplicand Latch
	L	H	L	L	Read LSB Product Latch
	L	H	H	L	Read MSB Product Latch

SIGN-BIT SELECTION FUNCTION TABLE

S/$\overline{U}$	I/O SIGNIFICANCE
L	Unsigned Byte Multiply
H	Signed Byte Multiply



16-PIN DIP
SBP 9708C . . . J OR N PACKAGE
SBP 9708 E,M . . . J PACKAGE



PIN DESCRIPTION

SIGNATURE	PIN	DESCRIPTION
V_{CC}	16	Provides +5 V connection to resistive pullups on outputs and voltage dividers on inputs.
GND	8	Ground
INJ *	5	Injector current input
D0-D7	7, 9 thru 15	Data input/output port
$\overline{CS}$, R/$\overline{W}$	2,4	Chip select low causes product latch data to be transferred onto the external data bus if read-write is high or causes multiplier/multiplicand data to be input to one of the input latches from the external bus if R/$\overline{W}$ is low. Chip select high causes D0-D7 to be at a logic high level.
M/$\overline{L}$	3	Most/Least byte select determines which multiplier latch is loaded on input, which byte of the product is read on output. Byte 1 latch and LSB product latch are accessed on M/$\overline{L}$ low.
MODE	6	MODE high causes adder array outputs to be passed directly to the 2 to 1 multiplexer. MODE low causes adder array outputs to be strobed into the product latches as the multiplier latch is loaded. Previous product data is then available after new multipliers are loaded. The multiplier latch must be loaded first.
S/$\overline{U}$	1	Signed multiply when high, unsigned multiply when low.

*Nominal current may be supplied by connection of a 24 Ω±5% (2 watt) resistor from V_{CC} to INJ input.

recommended operating conditions

PARAMETER			MIN	NOM	MAX	UNIT
Supply voltage, V_{CC}			4.5	5	5.5	V
Injector current, I_{INJ}			155	165	175	mA
High-level output voltage, V_{OH}					5.5	V
Low-level output current, I_{OL} (SBP 9708C only*)					8	mA
Width of load pulse (chip select low), $t_{w(\overline{CS})}$			100			ns
Setup time	M/$\overline{L}$ before load pulse, t_{su}		35↓			ns
	Data before end of load pulse, $t_{su(da)}$		60↓			
	R/$\overline{W}$ before load pulse, $t_{su(RW)}$		40↓			
Hold time	M/$\overline{L}$ after load pulse, $t_{h(ML)}$		0↑			ns
	Data after load pulse, $t_{h(da)}$		10↑			
	R/$\overline{W}$ after chip select, $t_{h(RW)}$		0↑			
Operating free-air temperature, T_A		SBP 9708M	−55		125	°C
		SBP 9708E	−40		85	
		SBP 9708C	0		70	

* Values for SBP 9708M and SBP 9708E will be announced at a later date.

electrical characteristics over recommended operating free-air temperature range (unless otherwise noted)

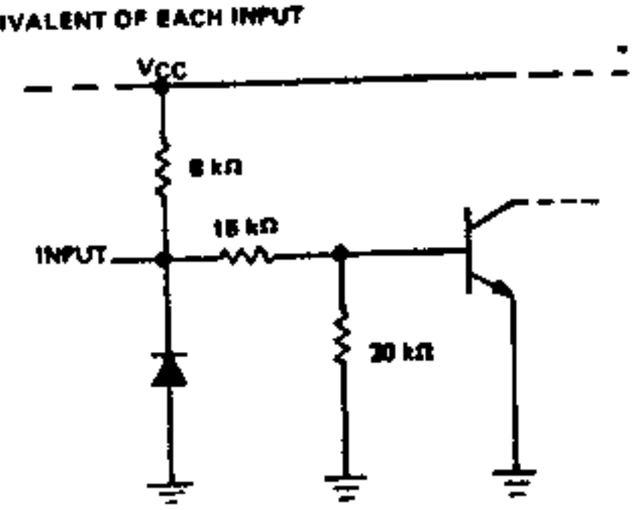
PARAMETER		TEST CONDITIONS†			MIN	TYP*	MAX	UNIT
V_{IH}	High-level input voltage				2			V
V_{IL}	Low-level input voltage						0.8	V
V_{IK}	Input clamp voltage	V_{CC} = MIN,	I_I = −12 mA				−1.5	V
V_{OH}	High-level output voltage	V_{CC} = MIN,	V_{IH} = 2 V, No Load	V_{IL} = 0.8V,	2.8	3.6		V
V_{OL}	Low-level output voltage	V_{CC} = MIN, V_{IL} = 0.8V,	V_{IH} = 2 V, I_{OL} = 8 mA	I_{INJ} = 130 mA,			0.5	V
I_{OS}	Short-circuit output current	V_{CC} = 5.5V			−0.3	−0.8	−1.2	mA
I_I	Input current at maximum input voltage	V_{CC} = MAX,	V_I = 5.5 V			0.4	1	mA
I_{IH}	High-level input current	V_{CC} = MAX,	V_I = 2.4 V			−0.3	−0.7	mA
I_{IL}	Low-level input current	V_{CC} = MAX,	V_I = 0.5 V			−0.7	−1.1	mA
I_{CC}	Supply current	V_{CC} = MAX				8	15	mA

switching characteristics over recommended operating free-air temperature range, V_{CC} = 5V, I_{INJ} = 165 mA

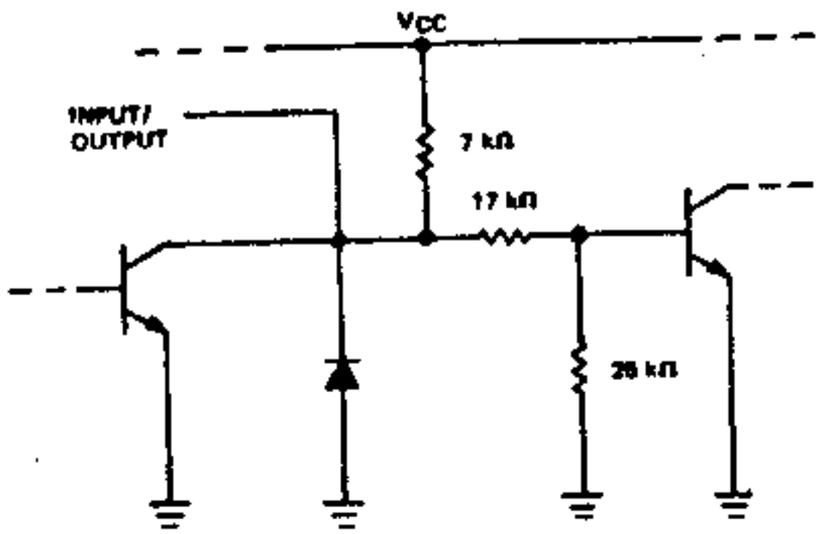
PARAMETER	FROM	TO	TEST CONDITIONS	NO EXTERNAL PULLUP			SEE LOAD CIRCUIT			UNIT
				MIN	TYP*	MAX	MIN	TYP*	MAX	
$t_{PD(CS)}$	$\overline{CS}$	OUTPUT (D7-D0)			240			150	200	ns
$t_{PD(S/U)}$	S/$\overline{U}$	OUTPUT (D7-D0)	C_L = 80 pF		340			250	325	ns
$t_{PD(M/L)}$	M/$\overline{L}$	OUTPUT (D7-D0)			250			160	200	ns
$t_{PD(MPY)}$	$\overline{CS}$ ↑	PROD OUT (D7-D0)	C_L = 80 pF, $V_{I(mode)}$ = 2 V		365			275	400	ns

† For conditions shown as MIN or MAX, use the appropriate value specified under recommended operating conditions for the applicable type.
* All typical values are at V_{CC} = 5V, T_A = 25°C.
↓ ↑ The arrow indicates the transition of the chip-select input used for reference: ↑ for the low-to-high transition, ↓ for the high-to-low transition.

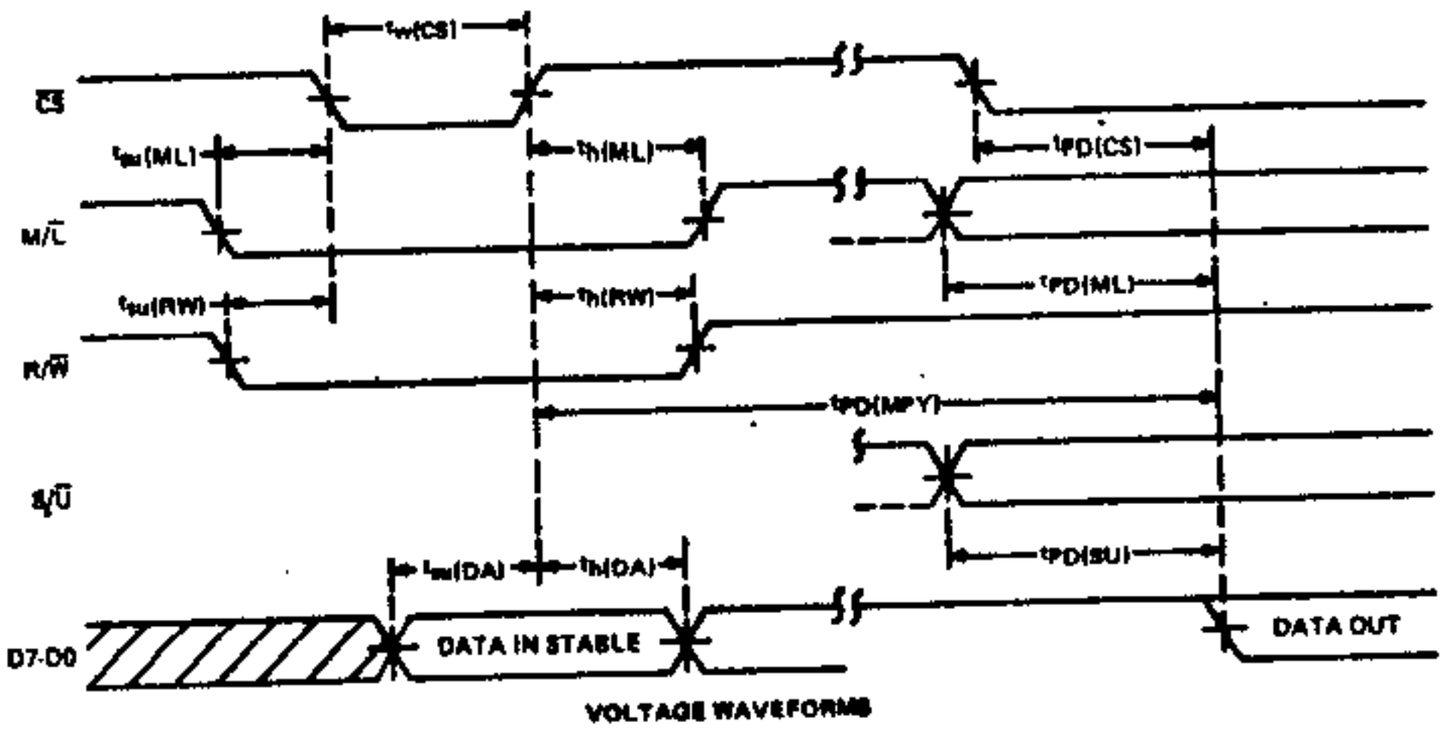
EQUIVALENT OF EACH INPUT



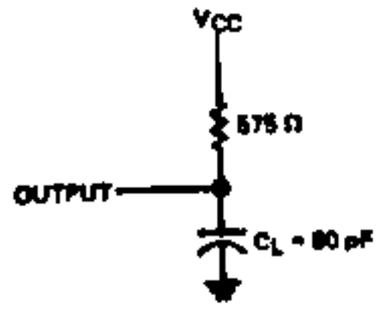
EQUIVALENT OF EACH INPUT/OUTPUT



SCHEMATICS OF INPUTS AND INPUT/OUTPUTS

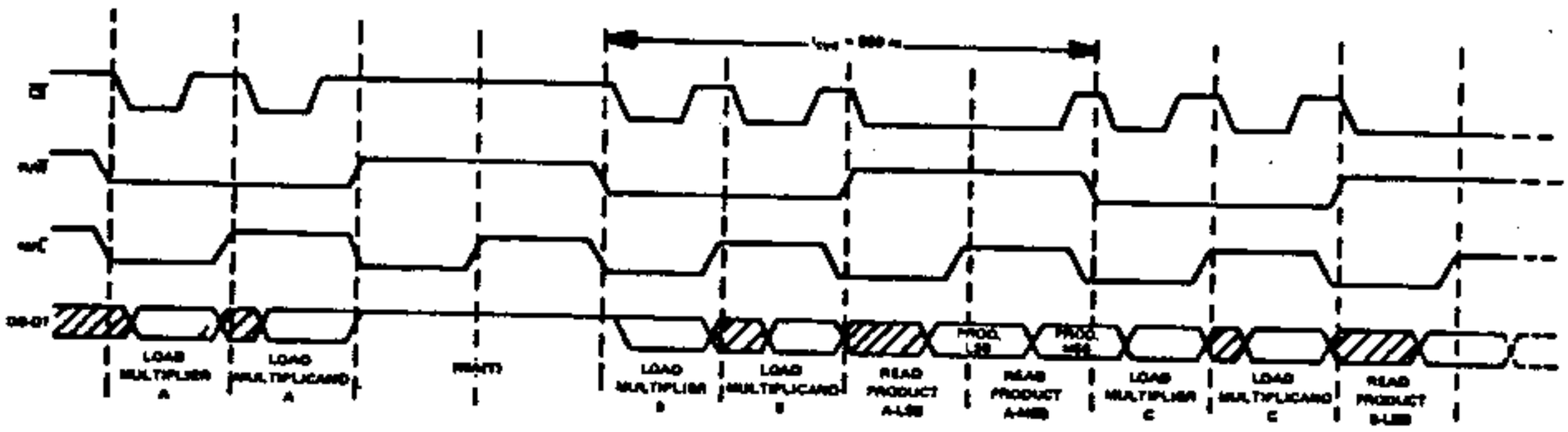


VOLTAGE WAVEFORMS

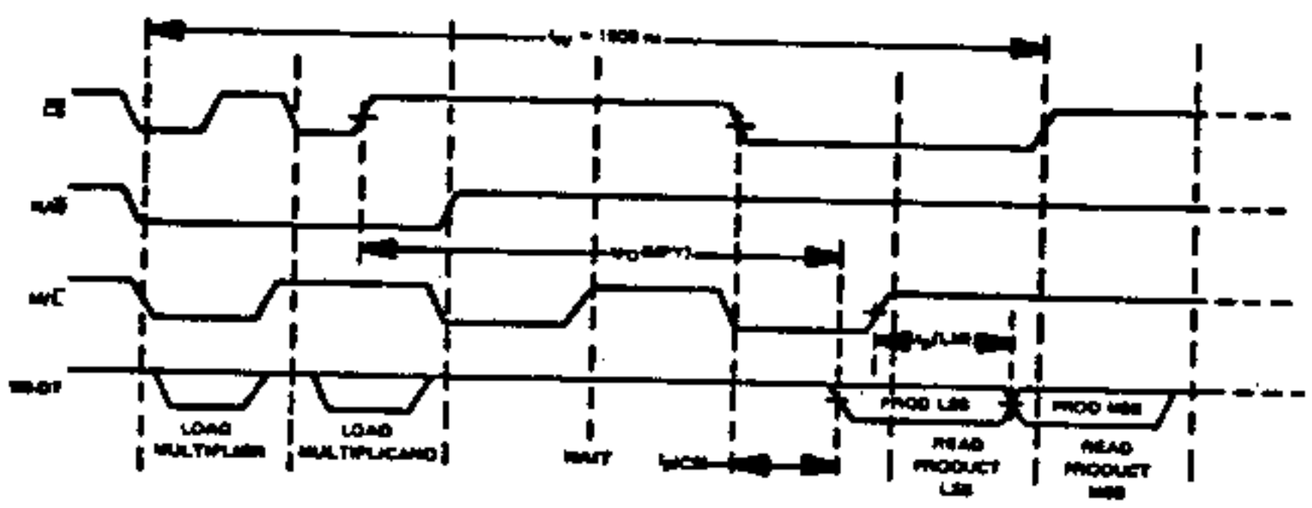


C_L includes probe and jig capacitance.

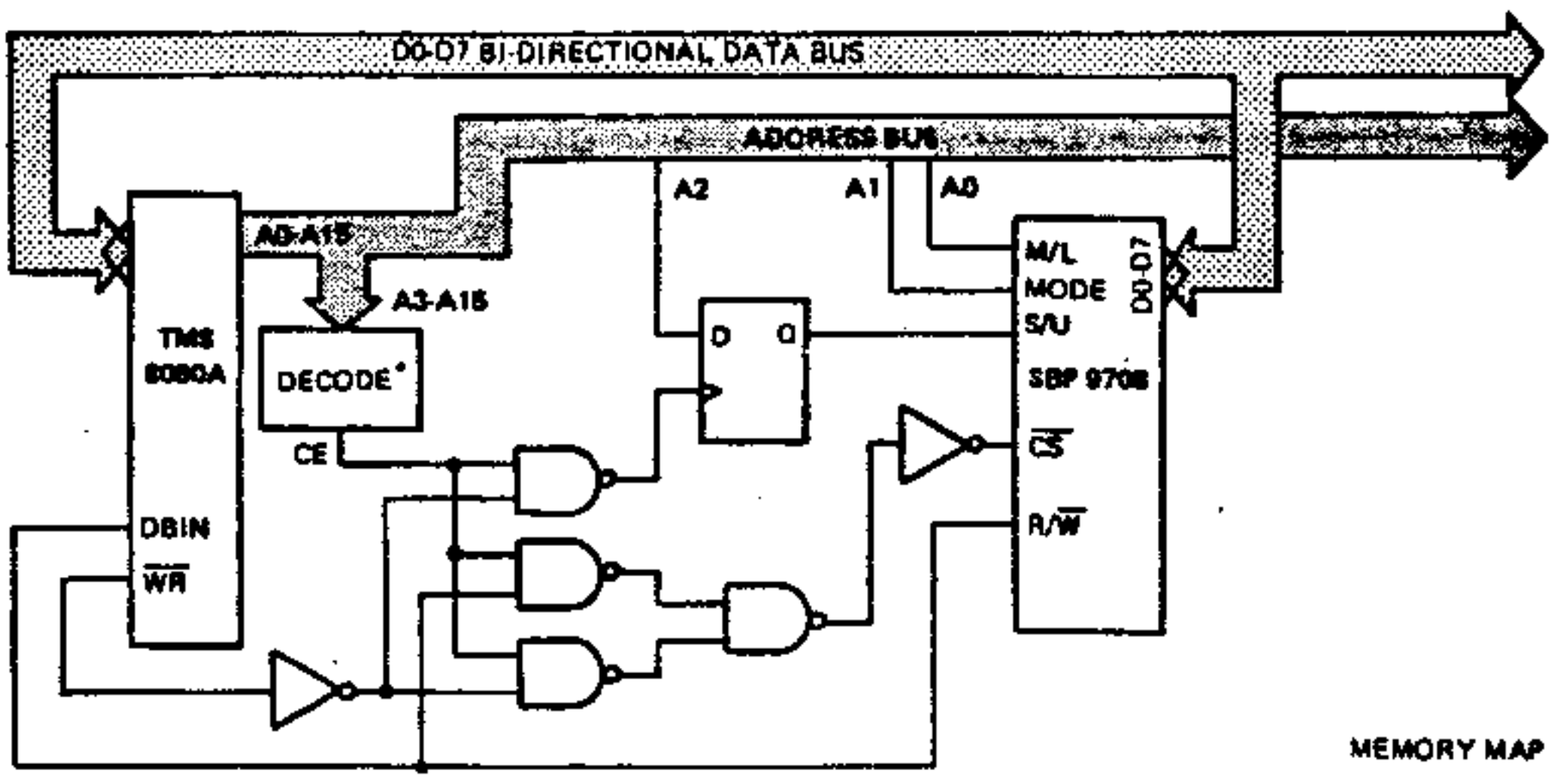
LOAD CIRCUIT



TIMING DIAGRAM—TYPICAL PIPELINED MODE SEQUENCE OF MULTIPLIES



TIMING DIAGRAM—TYPICAL TRANSPARENT MODE CYCLE



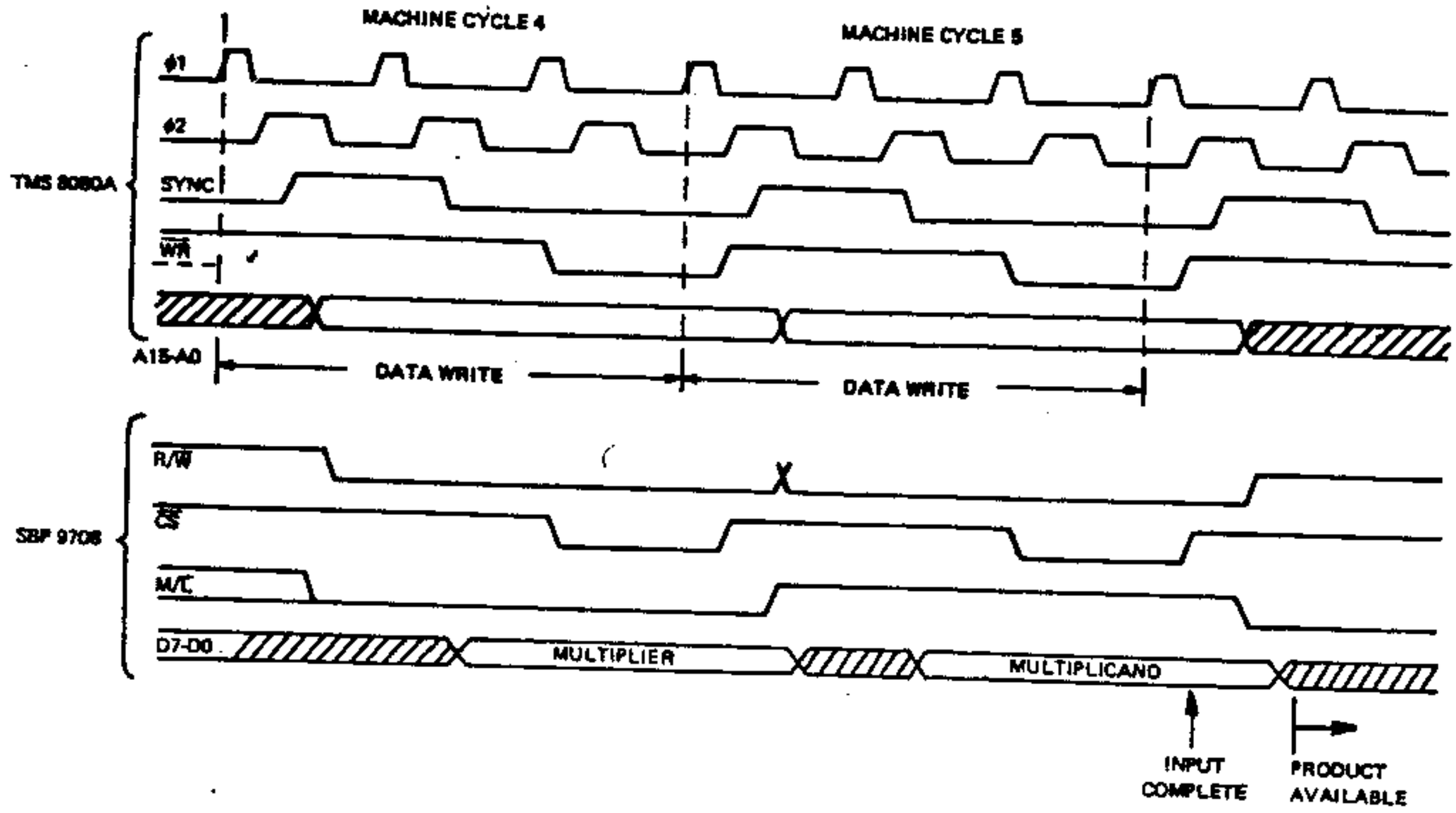
*CE becomes active (high) on a unique decode of address lines A3-A15

MEMORY MAP

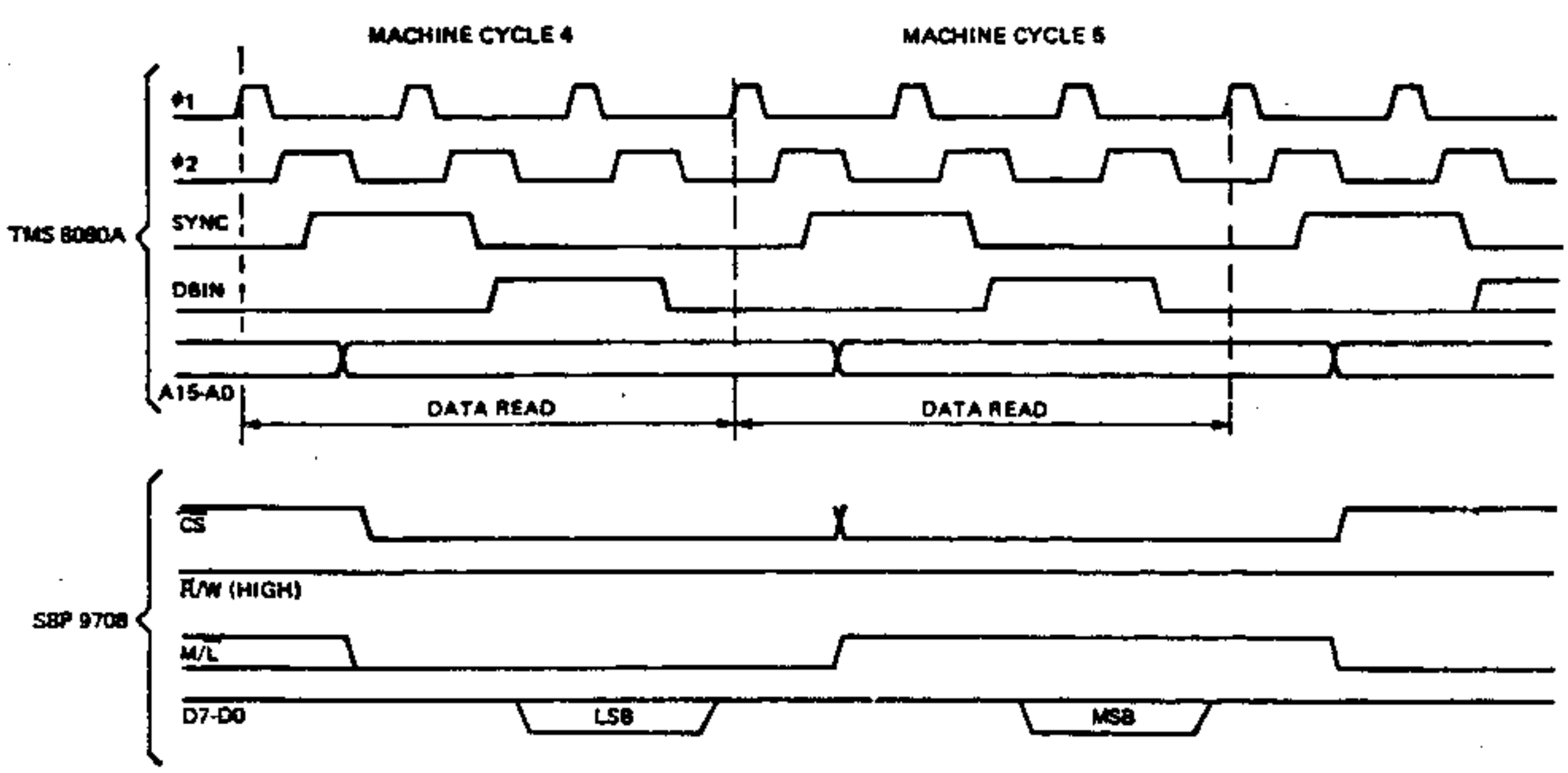
(A_1)	(A_2)	$A_0 = 0$	$A_0 = 1$	
LATCHED, UNSIGNED				ADR
TRANSPARENT, UNSIGNED				ADR + 2
LATCHED, SIGNED				ADR + 4
TRANSPARENT, SIGNED				ADR + 6
		LSB	MSB	

SBP 9708 TYPICAL MEMORY MAPPED APPLICATION (8080A SYSTEM)

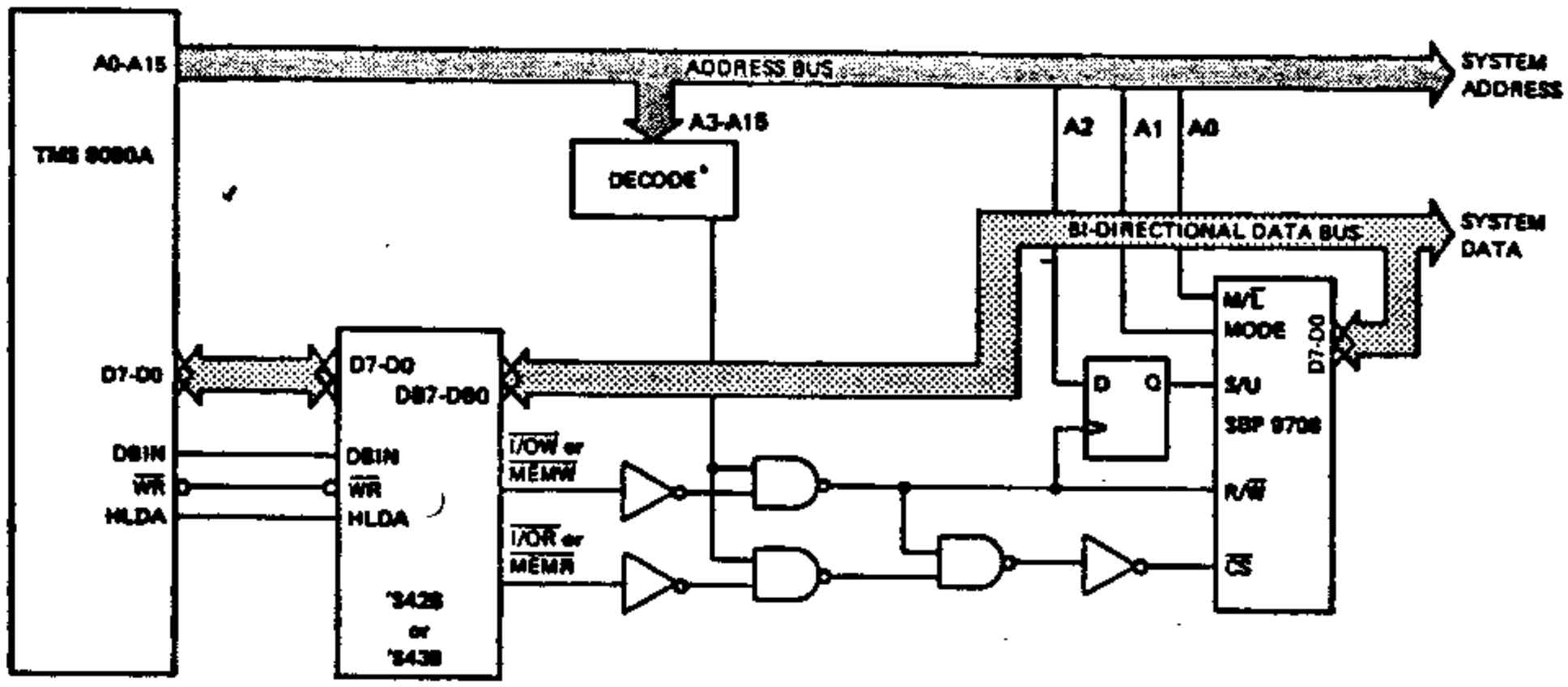
SBP 9708



8080A SYSTEM MULTIPLIER LOAD CYCLE USING SN74LS245 (MEMORY-MAPPED)

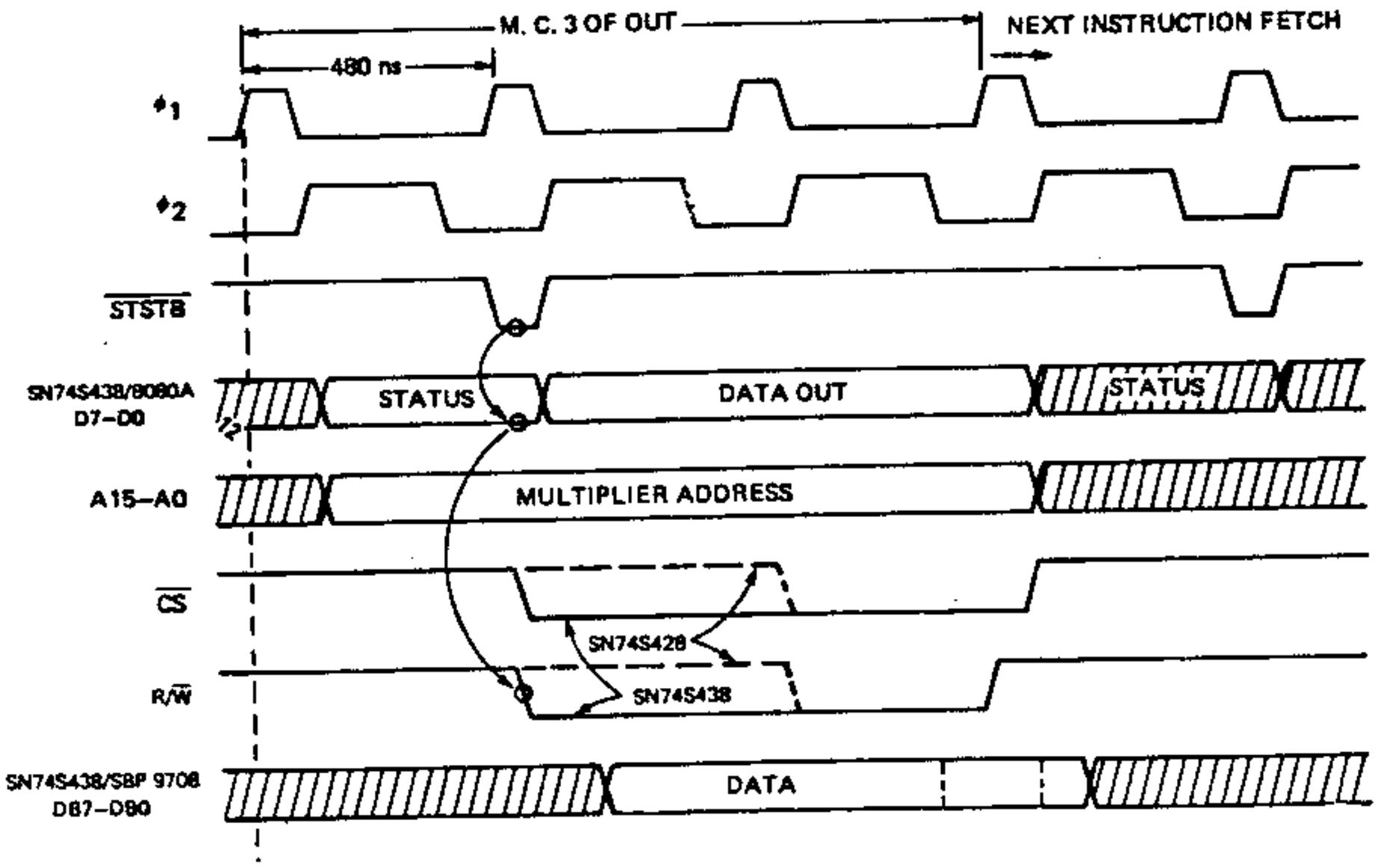


8080A SYSTEM PRODUCT READ CYCLE (MEMORY-MAPPED)

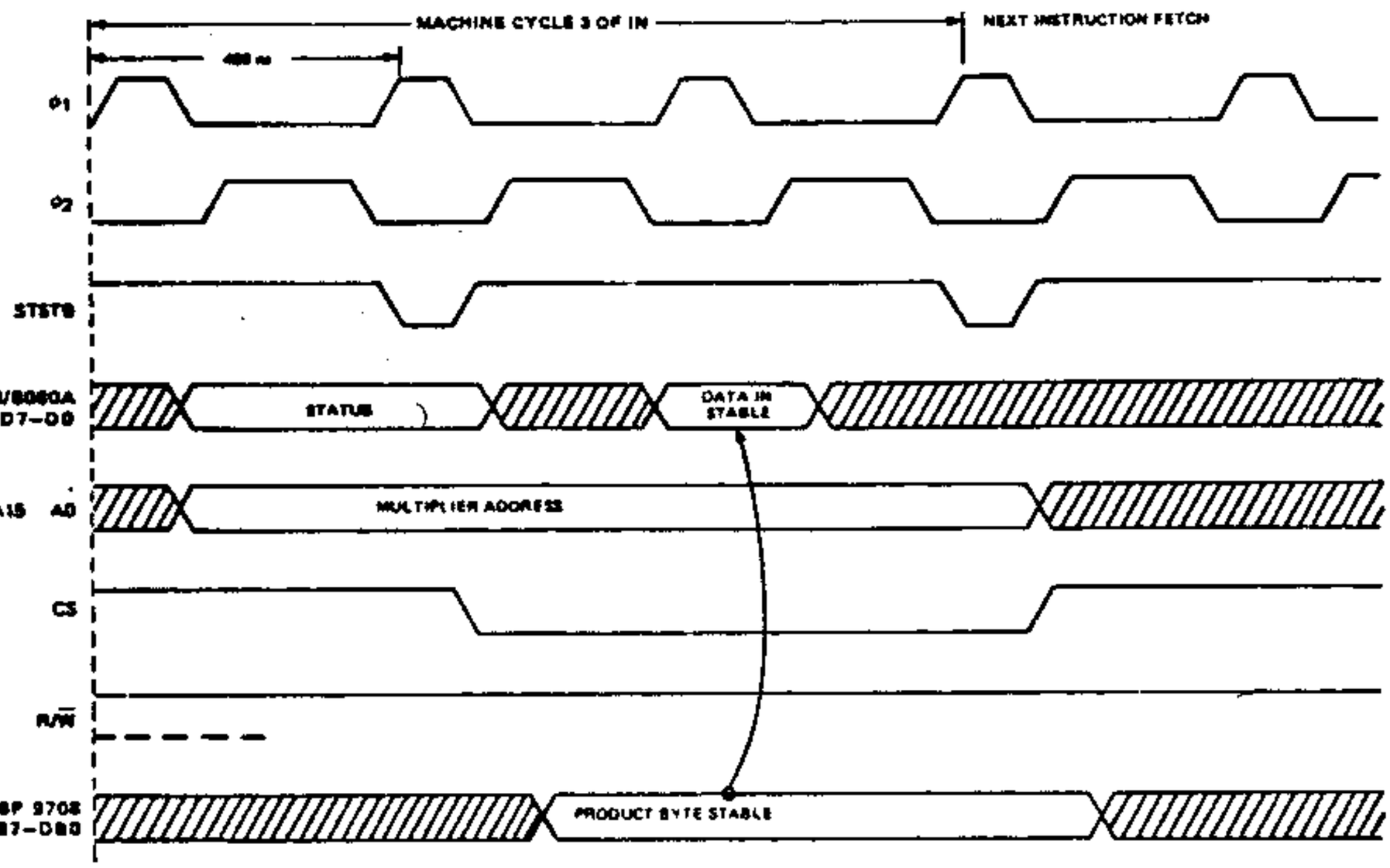


*DECODER OUTPUT GOES ACTIVE (HIGH) ON A UNIQUE DECODE OF ADDRESS LINES A3-A15

SBP 9708 TYPICAL I/O MAPPED APPLICATION (8080A SYSTEM)



8080A SYSTEM MULTIPLIER LOAD CYCLE USING SN74S428 OR SN74S438



8080A SYSTEM MULTIPLIER READ CYCLE USING SN745428 OR SN745438