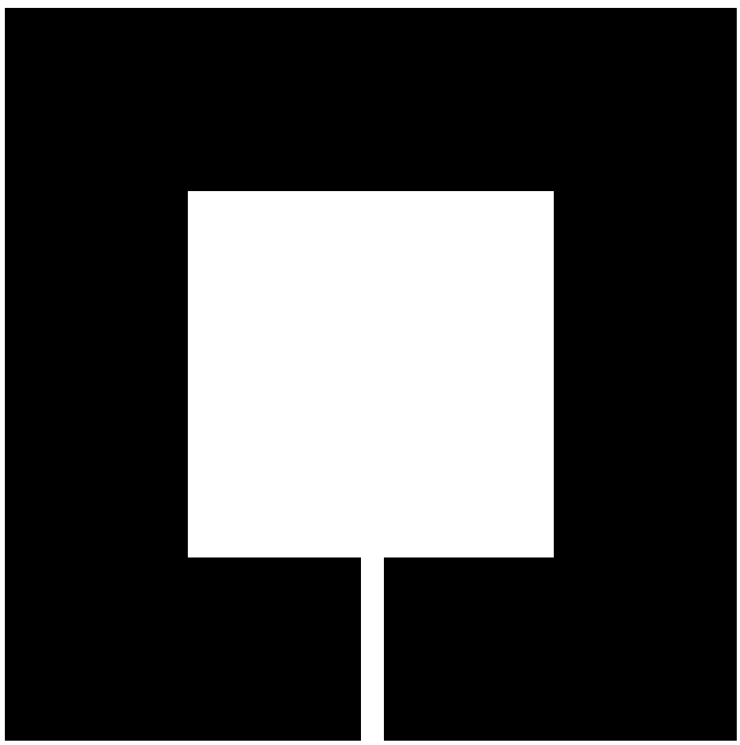


ADSP-2181
EZ-KIT Lite

Programmer's Quick Reference

Development Software Invocation Commands

Assembler `asm21 sourcefile [-switch …] -2181`

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
`-2181`	Use ADSP-2181 specific instructions

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

Switches

`-a archfile`	.ACH architecture file read by linker *(must specify ADSP2181.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
`-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

Simulators `sim2181 [-switch …]`

Switches

`-a archfile`	.ACH architecture file read by simulator *(must specify ADSP2181.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)*

PROM Splitter `spl21 exe_file promfile -pm [-switch …]`
or `spl21 exe_file promfile -dm [-switch …]`
or `spl21 exe_file promfile -loader -2181 [-s] [-i]`

Switches

`-pm`	Extract program memory ROM information
`-dm`	Extract data 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
`-us2`	PROM file format: Motorola S2 record byte stream
`-ui`	PROM file format: Intel Hex record byte stream
`-loader -2181`	Generate BDMA bootstrap loader in PROM file

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*
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
```

Invoking The Simulator

```
> SIM2181  [-a filename] [-c] [-e filename] [-k filename]
           [-w filename] [-o filename] [-help]
```

-a *filename*	Specify .ACH architecture file
-c	Make simulator case-sensitive for assembly code symbols
-e *filename*	Load program (.EXE file)
-k *filename*	Load and run keystroke file
-w *filename*	Start simulator with .WIN windows configuration file
-o *filename*	Open ASCII output file to capture simulator error messages and command output window messages
-help	Display invocation syntax and command line switches; simulator is not invoked

example:

```
> sim2181  -a c:\ADI_DSP\21xx\LIB\ADSP2181
```

General-Purpose Registers	***Program Control Registers***
HOLDER1	PC
HOLDER2	CNTR
HOLDER3	CYCLE
HOLDER4	PM_ADDR
	DM_ADDR

Simulator-Maintained Software Registers

Command	***Definition***
`alias 'newname' 'cmd'`	Create command alias
`alias >'filename'`	Save aliases to file
`alias`	List aliases in command output window
`addsymbol+ 'symbol' addr` `{asym+}`	Add program symbol at address
`assemble addr instr` `{a}`	Assemble instruction at address
`break addr`	Set breakpoint
`break addr, n`	Set multi-breakpoint
`break` `{b}`	List all breaks in command output window
`breakchange expr` `{bc}`	Set break change expression
`breakdelete break#` `{bd}`	Delete break
`breakdelete ALL` `{bd}`	Delete all breaks
`breakexpression expr` `{be}`	Set break expression
`breakrange range` `{br}`	Set break address range
`chipreset` `{cr}`	Chip reset (with program boot)
`connect rfs# tfs#`	Connect RFS of SPORT*x* (*rfs#*) toTFS of SPORT*y* (*tfs#*)
`connect rfs# tfs# OPEN`	Disconnect RFS-to-TFS connection
`connect ALL OPEN`	Disconnect all RFS-to-TFS connections
`connect`	List current RFS-to-TFS connections
`delete 'newname'`	Delete alias
`dump range >'filename'` `{d}`	Dump memory to file
`execute instr` `{exe}`	Execute instruction
`find addr,addr expr` `{f}`	Find numeric value (*expr*) in memory range
`fill addr expr`	Set memory location to value (*expr*)
`fill range expr`	Fill memory range with value (*expr*)
`fill startaddr <'filename'`	Fill memory range from file
`go`	Start program execution
`go addr`	Start program execution, stop at *addr*

{g}

Command	***Definition***
`interrupt` *`irq# min`* [*`max`*][`OFFSET` *`#cycles`*]	Generate periodic interrupt signal
`interrupt` *`sig min`* [*`max`*][`OFFSET` *`#cycles`*]	Generate periodic serial port signal
`interrupt FI` *`period`* [`ONCE|RESET`]	Generate periodic Flag In signal
`interrupt ALL`	List settings for all signals
`interrupt ACTIVE`	List settings for all active signals
{i}	
`keyon` *`'keyfile'`*	Start recording keystroke file
{ko}	
`keyoff`	Stop recording keystrokes, close file
{kf}	
`<'keyfile'`	Playback keystroke file
`load` *`'filename'`*	Load program (.EXE file) into memory
{l}	
`loadrom` *`'filename'`*	Load ROM file (.BNM file) into boot memory
{lr}	
`loadsymbols` *`'filename'`*	Read new symbol table (.SYM file) into simulator
{ls}	
`loopback` *`sport#`*	Connect TFS to RFS and DT to DR of serial port
{lb}	
`loopback` *`sport#`* `OPEN`	Disconnect loopback connection
`loopback ALL OPEN`	Disconnect all loopback connections
`loopback`	List current loopback connections
{lb}	
`openport` *`addr`* [*`<'infile'`*][*`>'outfile'`*]	Open memory-mapped I/O port and assign data files
{op}	
`openport` *`addr  <'infile'`* `CIRC`	Open memory-mapped I/O port with circular input data file
{op}	
`openport` *`addr`*	Close memory-mapped I/O port
{op}	
`opensport` *`sport#`* [*`<'infile'`*][*`>'outfile'`*]	Open SPORT and assign data files
`opensport` *`sport# <'infile'`* `CIRC`	Open SPORT with circular input data file

`opensport` *`sport#`*	Close SPORT
`{os}`	
Command	***Definition***
`plot` *`range decimation`*	Plot memory data
`{pl}`	
`profileadd` *`range# addr,addr`*	Define (or redefine) profile address range
`{pa}`	
`profileclear`	Clear all profile data, delete all address ranges
`{pc}`	
`profiledelete` *`range#`*	Delete profile address range
`{pd}`	
`profilereset`	Clear all profile data, retain address ranges
`{pr}`	
`resetnoboot`	Reset chip, without program boot
`{re}`	
`shell`	Temporarily exit to operating system (type `"exit"` to return to simulator)
`{sh}`	
`state >'`*`file`*`'`	Save simulator state
`state <'`*`file`*`'`	Restore simulator state
`step`	Single step
`step` *`n`*	Multi-step *n* instructions
`{s}`	
`undefine` *`reg`*	Undefine contents of register
`undefine` *`addr`*	Undefine contents of memory location
`undefine` *`range`*	Undefine contents of memory range
`{u}`	
`watchdelete` *`watch#`*	Delete *watch#*
`{wd}`	
`watchexpression` *`expr`*	Set watch expression
`{we}`	
`watchpoint` *`addr`*	Set watchpoint on data variable or buffer
`watchpoint` *`range`*	Set watch address range
`watchpoint`	List all watches in command output window
`{w}`	
`win >'`*`file`*`'`	Save windows configuration (.WIN file)
`win <'`*`file`*`'`	Restore windows configuration (.WIN file)
`?` *`expr`*	Evaluate general expression
`?` *`reg`*	Evaluate contents of register
`?` *`addr`*	Evaluate contents of memory location
`?` *`symbol`*	Locate program symbol
`?+` *`expr`*	Add expression to expressions window
`?-` *`expr#`*	Delete *expr#* from expressions window
`reg = expr`	Set register equal to value of expression

(ex. `AX0=0xFF`)

Short form of command: *command* `chipreset`
{*cmd*} {cr}

To define an address range: *range* = *startaddr* , *endaddr*
range = *startaddr* / *#locations*

Command Window Commands

Ctrl-D Dump memory to file (*in any memory window*)
Ctrl-F Fill memory (*in any memory window*)
Ctrl-G Go to address (*in any memory window*)
Ctrl-L List open windows and choose one to bring to front
Ctrl-M Load memory (*in any memory window*)
Ctrl-R Resize trace length (*in Trace Window*)
Ctrl-T Toggle numeric format in window (decimal ↔ hexadecimal)
Ctrl-O Toggle tracking of data memory and program memory window

Control Key Sequences

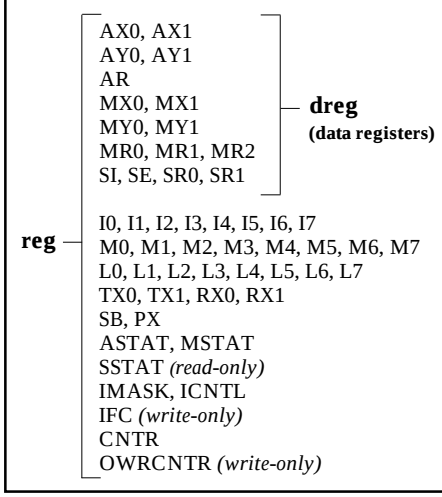
Control Key	^D	^F	^G	^M	^R	^T	^O
Window							
Boot Memory	✔	✔	✔	✔		✔	
Data Memory	✔	✔	✔	✔		✔	✔
Program Memory	✔	✔	✔	✔		✔	✔
Computational Registers						✔	
Control Registers						✔	
DAG Registers						✔	
Program Control Registers						✔	
SPORT Registers						✔	
Command Window						✔	
Expressions						✔	
Profile						✔	
Trace					✔	✔	

Window-to-Control Key Cross Reference

Function Key	*Function*
F1	Help
F2	Goto Next Window
F3	Goto Menu Bar
F4	Run (Go)
F5	Move Window Up ↑
F6	Move Window Down ↓
F7	Move Window Left ←
F8	Move Window Right →
F9	Set / Clear Breakpoint
F10	Single Step
Shift-F5	Enlarge Window Vertically ◊
Shift-F6	Reduce Window Vertically
Shift-F7	Enlarge Window Horizontally ←→
Shift-F8	Reduce Window Horizontally →←
Shift-F9	Set / Clear Multi-Breakpoint
Shift-F10	Multi-Step
Escape	Close window

Keyboard Function Keys (PC only)

Allowed Registers for Data Move & Multifunction Instructions



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:     …
```

Instruction Set Summary

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 Instructions

[IF cond]	AR AF	=	xop	+	yop C yop + C constant	;	*Add/Add with Carry*
		=	xop	–	yop yop + C – 1 constant	;	*Subtract X–Y/Subtract X–Y with Borrow*
		=	yop	–	xop xop + C – 1 constant	;	*Subtract Y–X/Subtract Y–X with Borrow*

Permissible xops
AX0, AX1, AR, MR0, MR1, MR2, SR0, SR1

Permissible yops (base instruction set)
AY0, AY1, AF

Permissible yops and constants (extended instruction set)
AY0, AY1, AF, 0, 1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192, 16384,
32767, -2, -3, -5, -9, -17, -33, -65, -129, -257, -513, -1025, -2049, -4097, -8193, -16385, -32768

[IF cond]	AR AF	=	xop	AND OR XOR	yop	;	*AND, OR, XOR*

[IF cond] | AR / AF | = PASS | xop / yop / constant | ; *Pass, Clear*

Permissible yops (base instruction set)
AY0, AY1, AF

Permissible yops and constants (extended instruction set)
AY0, AY1, AF, 0, 1, 2, 3, 4, 5, 7, 8, 9, 15, 16, 17, 31, 32, 33, 63, 64, 65, 127, 128, 129, 255, 256, 257, 511, 512, 513, 1023, 1024, 1025, 2047, 2048, 2049, 4095, 4096, 4097, 8191, 8192, 8193, 16383, 16384, 16385, 32766, 32767, -1, -2, -3, -4, -5, -6, -8, -9, -10, -16, -17, -18, -32, -33, -34, -64, -65, -66, -128, -129, -130, -256, -257, -258, -512, -513, -514, -1024, -1025, -1026, -2048, -2049, -2050, -4096, -4097, -4098, -8192, -8193, -8194, -16384, -16385, -16386, -32767, -32768

[IF cond] | AR / AF | = – | xop / yop | ; *Negate*

= NOT | xop / yop / 0 | ; *NOT*

= ABS xop ; *Absolute Value*

= yop + 1 ; *Increment*

= yop – 1 ; *Decrement*

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

= | TSTBIT n of xop / SETBIT n of xop / CLBIT n of xop / TGBIT n of xop | ; *Bit Operations*

Permissible xops
AX0, AX1, AR, MR0, MR1, MR2, SR0, SR1

Permissible n Values (0 = LSB)
0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15

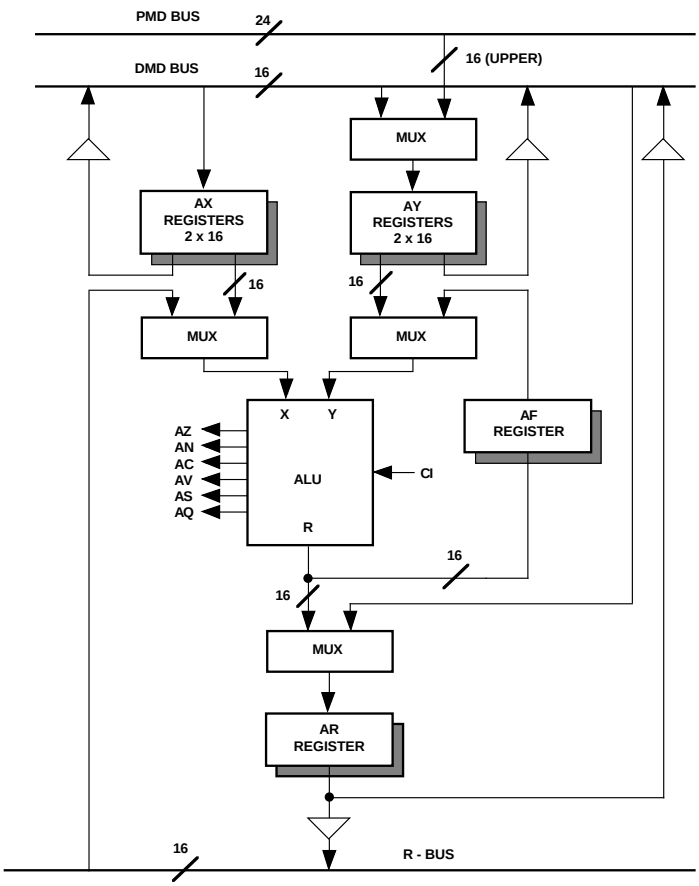
Definitions of Operations
TSTBIT is an AND operation with a 1 in the selected bit
SETBIT is an OR operation with a 1 in the selected bit
CLBIT is an AND operation with a 0 in the selected bit
TGBIT is an XOR operation with a 1 in the selected bit

NONE = <ALU>; *Result Free*

where <ALU> is any unconditional ALU operation of the 21xx base instruction set (except DIVS or DIVQ). (Note that the additional constant ALU operations of the ADSP-2171/2181 extended instruction set are not allowed.)

Description: Perform the designated ALU operation, set the condition flags, then discard the result value. This allows the testing of register values without disturbing the AR or AF register values.

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 / xop	(SS) / (SU) / (US) / (UU) / (RND)	;	*Multiply*
		=	MR + xop *	yop / xop	(SS) / (SU) / (US) / (UU) / (RND)	;	*Multiply/Accumulate*
		=	MR – xop *	yop / xop	(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

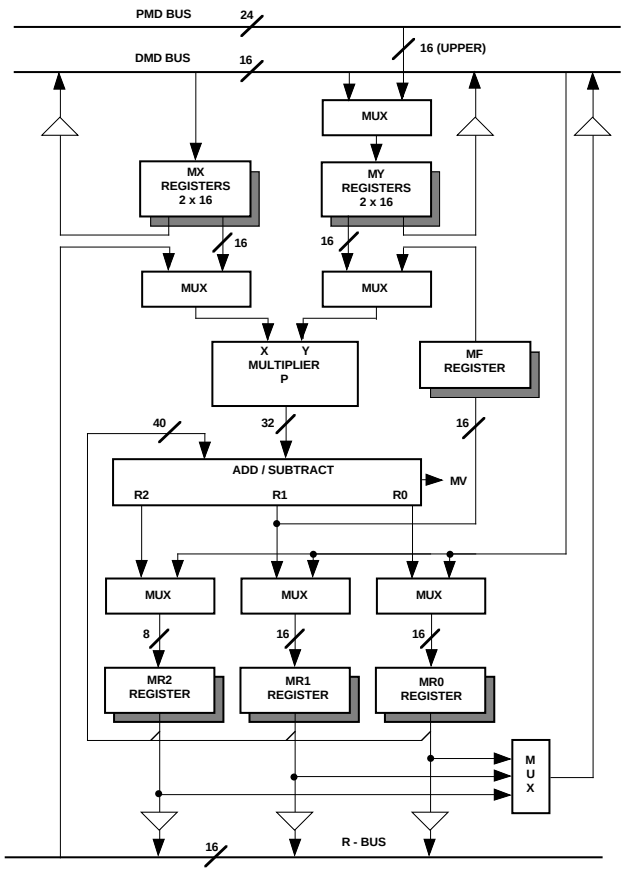
Cond	
EQ	Equal zero
NE	Not equal zero
LT	Less than zero
GE	Greater than or equal zero
LE	Less than or equal zero
GT	Greater than zero
AC	ALU carry
NOT AC	Not ALU carry
AV	ALU overflow
NOT AV	Not ALU overflow
MV	MAC overflow
NOT MV	Not MAC overflow
NEG	Xop input sign negative
POS	Xop input sign positive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

** Only for JUMP, CALL*

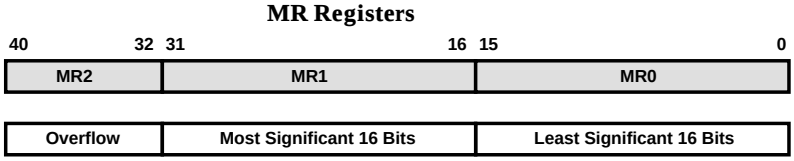
Allowed XOP, YOP Registers for MAC Instructions

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 = EXPADJ 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

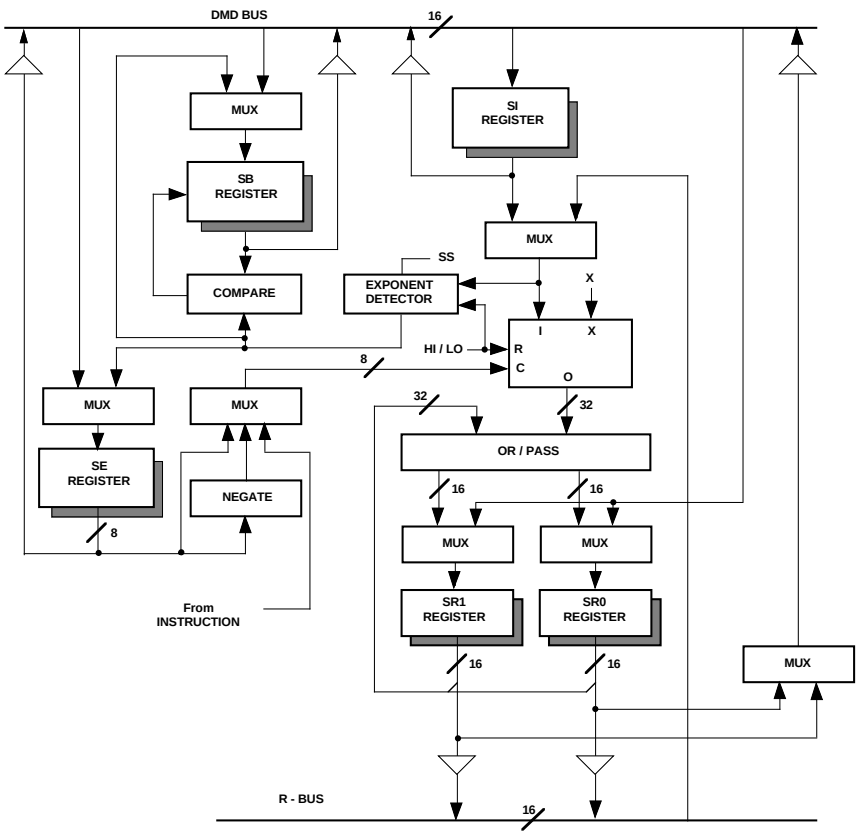
Cond	
EQ	Equal zero
NE	Not equal zero
LT	Less than zero
GE	Greater than or equal zero
LE	Less than or equal zero
GT	Greater than zero
AC	ALU carry
NOT AC	Not ALU carry
AV	ALU overflow
NOT AV	Not ALU overflow
MV	MAC overflow
NOT MV	Not MAC overflow
NEG	Xop input sign negative
POS	Xop input sign positive
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*
dreg = DMOVLAY	*Read Overlay Register*
DMOVLAY = dreg ;	*Write Overlay Register*
reg = DM (<addr>) ;	*Data Memory Read (Direct Address)*
dreg = IO (<addr>); *(See Note 1)*	*I/O Read (Direct Address)*
dreg = DM (I0 , M0) ; I1 M1 I2 M2 I3 M3 I4 M4 I5 M5 I6 M6 I7 M7	*Data Memory Read (Indirect Address)*
dreg = PM (I4 , M4) ; I5 M5 I6 M6 I7 M7	*Program Memory Read (Indirect Address*)
DM (<addr>) = reg ;	*Data Memory Write (Direct Address)*
DM (<addr>) = DMOVLAY ;	*Writes Contents of Overlay Registers to Data Memory*
IO (<addr>) = dreg ; *(See Note 1)*	*I/O Write (Direct Address)*
DM (I0 , M0) = dreg ; I1 M1 I2 M2 I3 M3 I4 M4 I5 M5 I6 M6 I7 M7	*Data Memory Write (Indirect Address)*
PM (I4 , M4) = dreg ; I5 M5 I6 M6 I7 M7	*Program Memory Write (Indirect Address)*

Note 1: <addr> is an address value between 0 and 2048.

Multifunction Instructions

```
|<ALU>  |
|<MAC>  | ,   dreg = dreg ;
|<SHIFT>|
```

Computation with Register-to-Register Move

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

                   PM ( |I4| , |M4| ) ;
                        |I5|   |M5|
                        |I6|   |M6|
                        |I7|   |M7|
```

Computation with Memory Read

```
|DM ( |I0| , |M0| )|
|     |I1|   |M1|  |
|     |I2|   |M2|  |
|     |I3|   |M3|  |
|                  |
|     |I4|   |M4|  |
|     |I5|   |M5|  |
|     |I6|   |M6|  |
|     |I7|   |M7|  |
|                  |
|PM ( |I4| , |M4| )|
|     |I5|   |M5|  |
|     |I6|   |M6|  |
|     |I7|   |M7|  |
```

= dreg , |<ALU> | ;
|<MAC> |
|<SHIFT>|

Computation with Memory Write

```
|AX0 |
|AX1 |
|MX0 | = DM ( |I0| , |M0| )
|MX1 |        |I1|   |M1|
             |I2|   |M2|
             |I3|   |M3|
```

,

```
|AY0 |
|AY1 | = PM ( |I4| , |M4| ) ;
|MY0 |        |I5|   |M5|
|MY1 |        |I6|   |M6|
             |I7|   |M7|
```

Data & Program Memory Read

```
|<ALU>  |
|<MAC>  | ,
```

```
|AX0 |
|AX1 | = DM ( |I0| , |M0| )
|MX0 |        |I1|   |M1|
|MX1 |        |I2|   |M2|
             |I3|   |M3|
```

,

```
|AY0 |
|AY1 | = PM ( |I4| , |M4| ) ;
|MY0 |        |I5|   |M5|
|MY1 |        |I6|   |M6|
             |I7|   |M7|
```

ALU/MAC with Data & Program 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 | INTS [, ...] ; — *Mode Control*

IDLE ; — *Put Processor In Idle State*

IDLE (n) ; — *Put Processor In Idle State And Slow Clock By a Factor Of n*

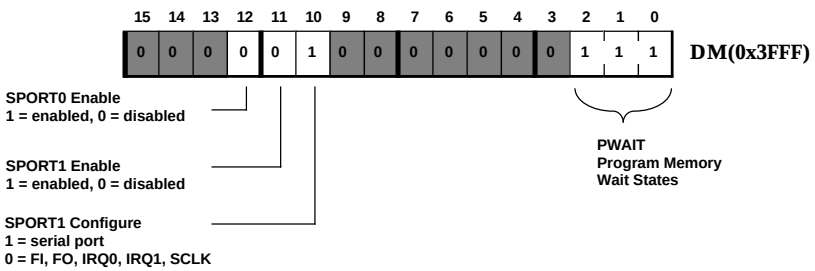
Permissible Values for n: 16, 32, 64, 128

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
INTS	Interrupt enable

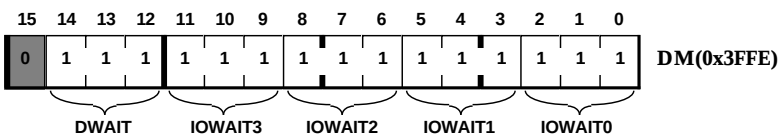
Control/Status Registers

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.

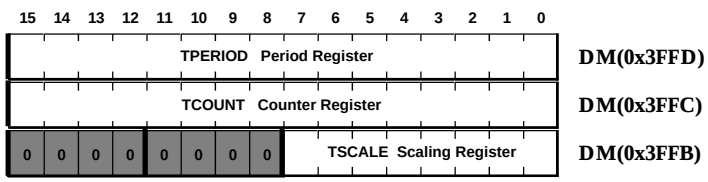
System Control Register



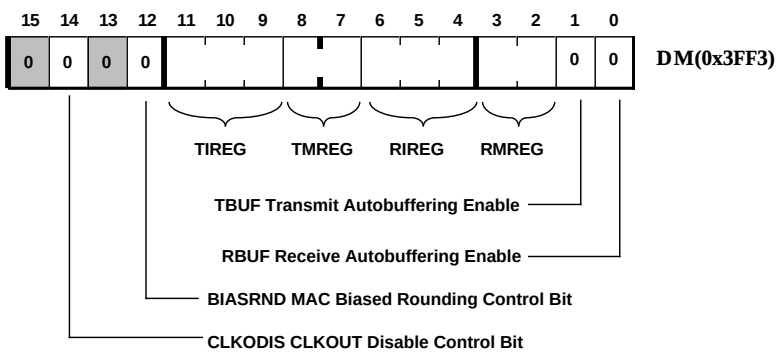
Data Memory Waitstate Register



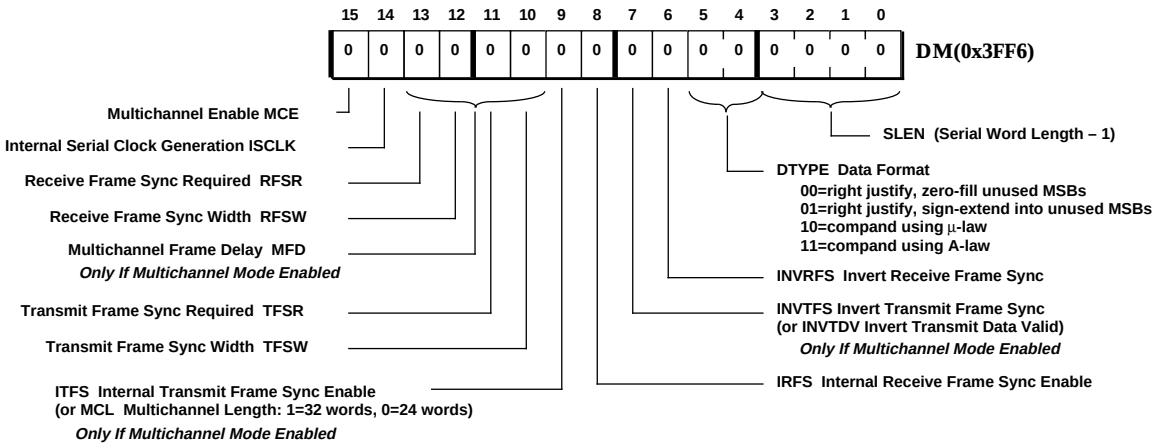
Timer Registers



SPORT0 Autobuffer Control

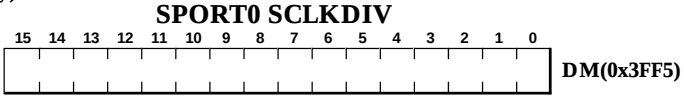


SPORT0 Control Register



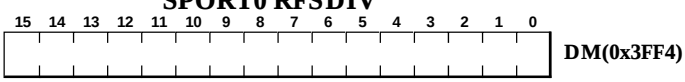
$$\text{SCLKDIV} = \frac{\text{CLKOUT frequency}}{2 * (\text{SCLK frequency})} - 1$$

SPORT0 SCLKDIV

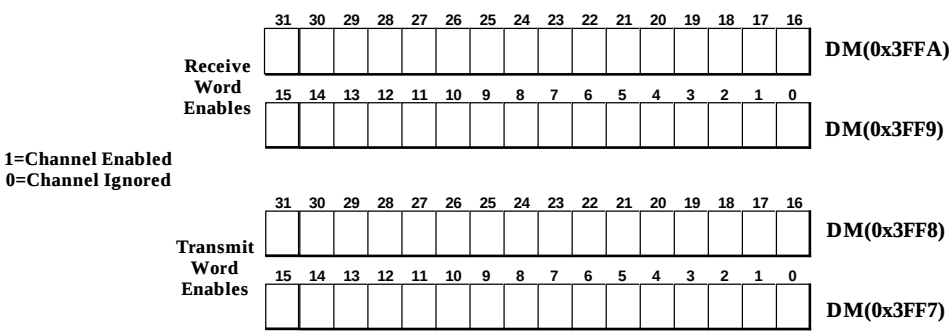


$$\text{RFSDIV} = \frac{\text{SCLK frequency}}{\text{RFS frequency}} - 1$$

SPORT0 RFSDIV

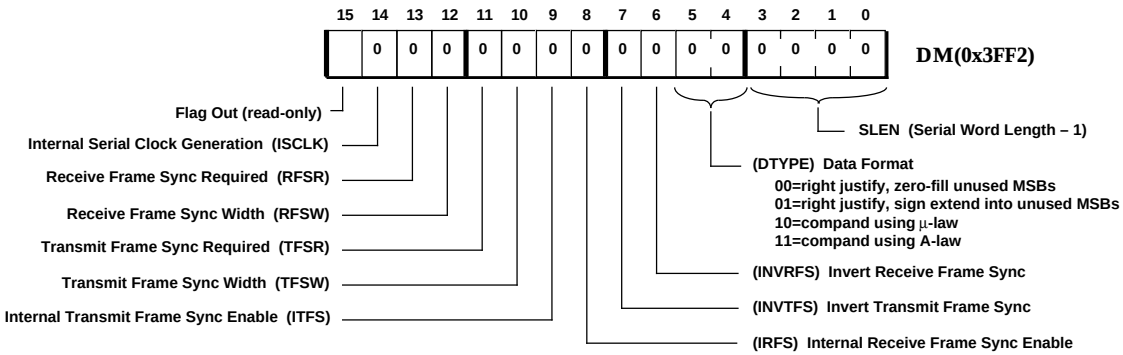


SPORT0 Multichannel Word Enables

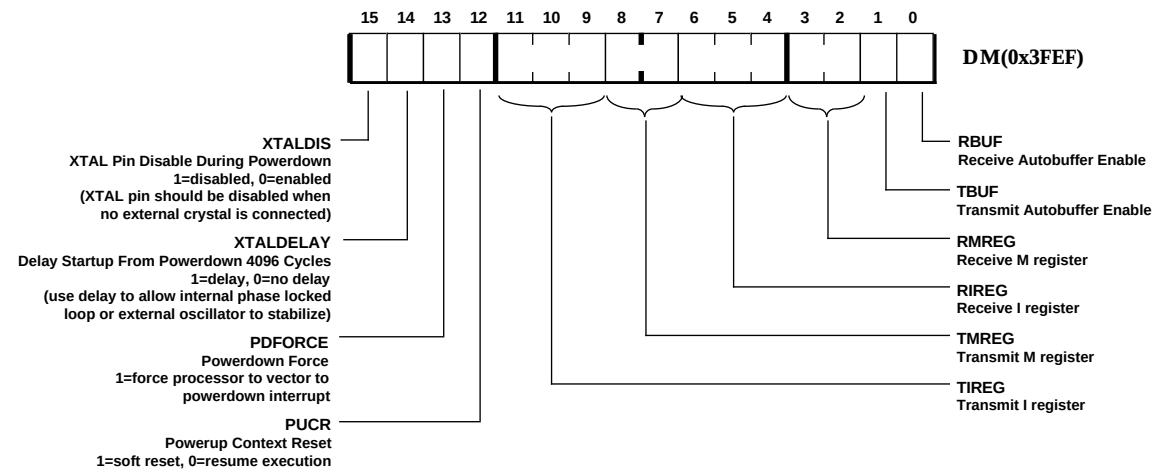


Control/Status Registers

SPORT1 Control Register

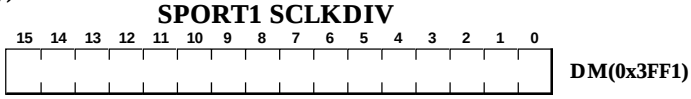


SPORT1 Autobuffer Control



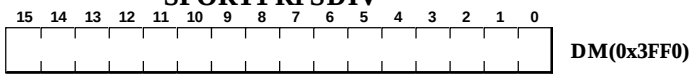
$$\text{SCLKDIV} = \frac{\text{CLKOUT frequency}}{2 * (\text{SCLK frequency})} - 1$$

SPORT1 SCLKDIV

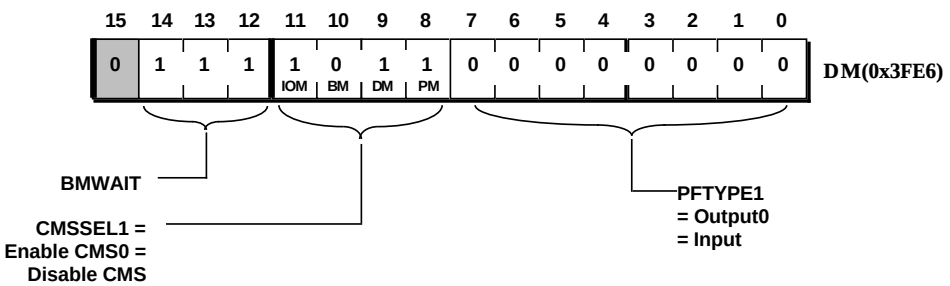


$$\text{RFSDIV} = \frac{\text{SCLK frequency}}{\text{RFS frequency}} - 1$$

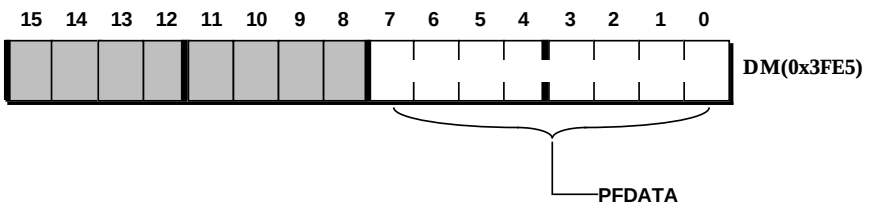
SPORT1 RFSDIV



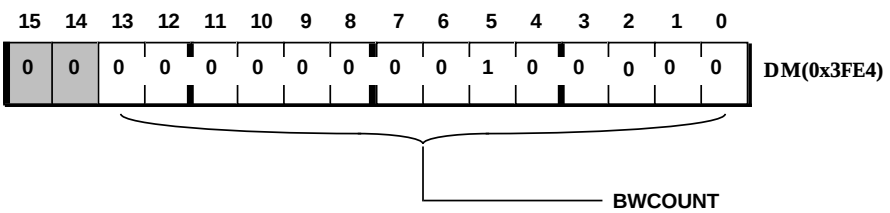
Programmable Flag & Composite Select Control



Programmable Flag Data

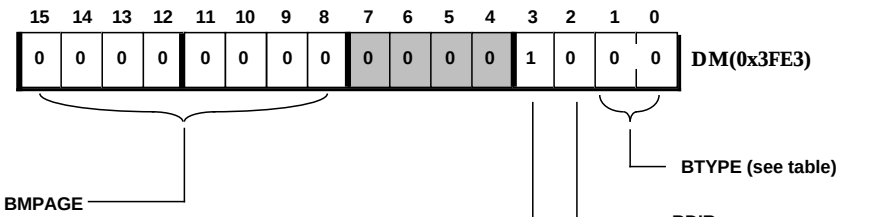


BDMA Word Count



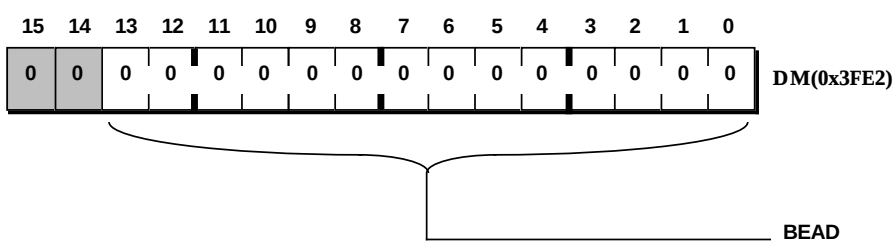
Control/Status Registers

BDMA Control

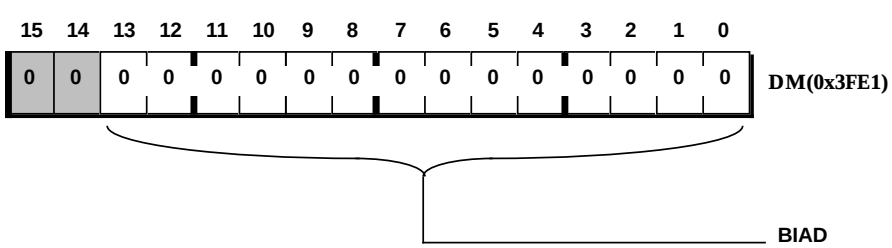


BTYPE	INTERNAL MEMORY SPACE	WORD SIZE	ALIGNMENT
00	PM	24	full word
01	DM	16	full word
10	DM	8	msb
11	DM	8	lsb

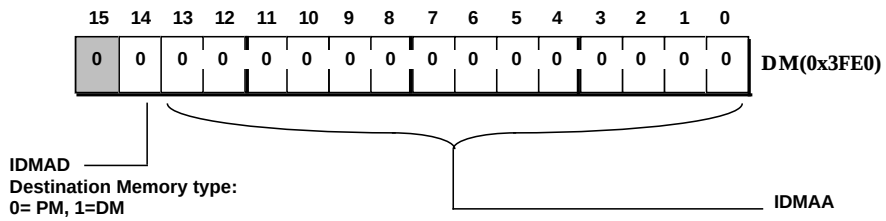
BDMA External Address



BDMA Internal Address



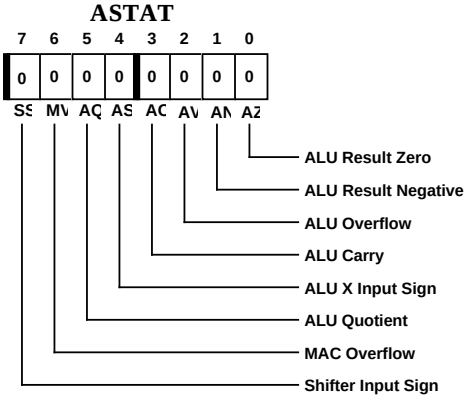
Programmable Flag & Composite Select Control



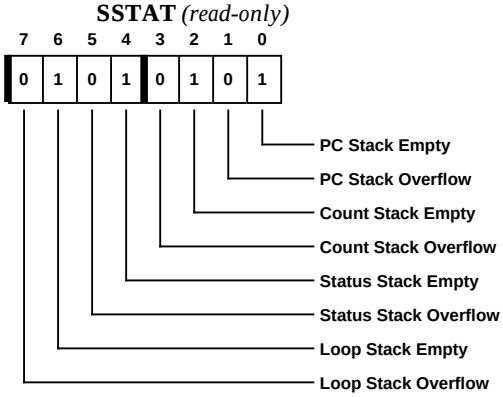
Status Registers

(Non-Memory-Mapped)

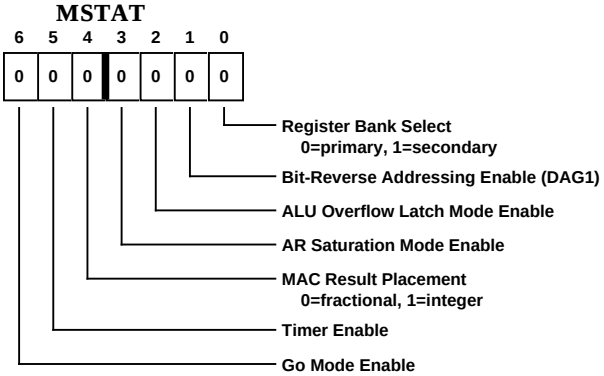
ASTAT



SSTAT *(read-only)*



MSTAT



Mode Names for Mode Control Instruction

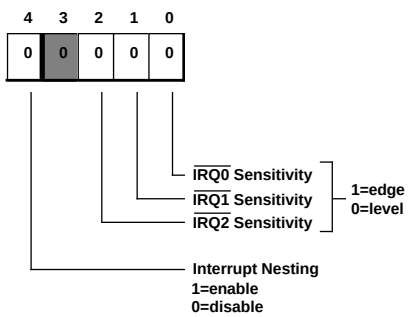
(see Miscellaneous Instructions on page 21)

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
7	INTS	Interrupt enable

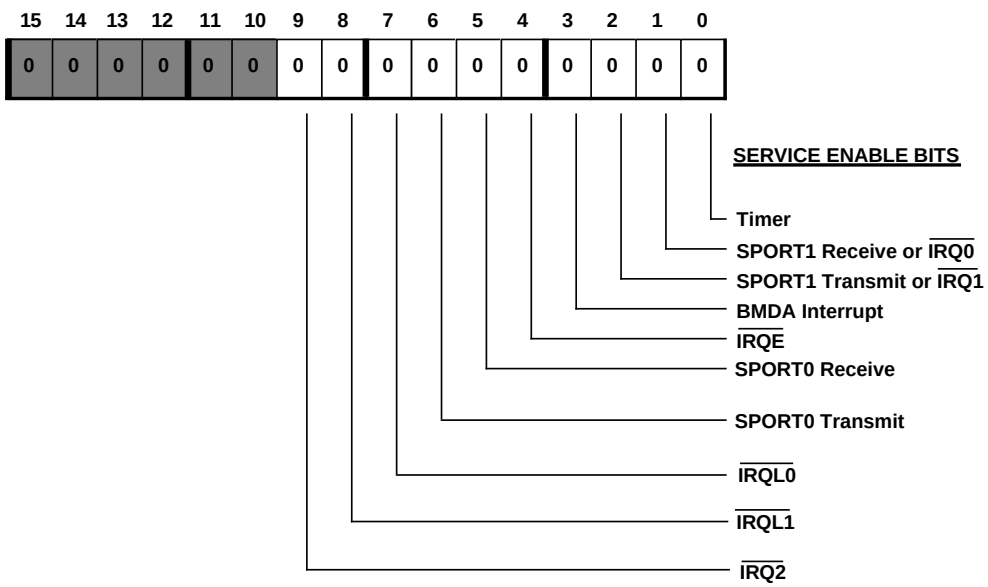
Interrupt Registers

(Non-Memory-Mapped)

ICNTL



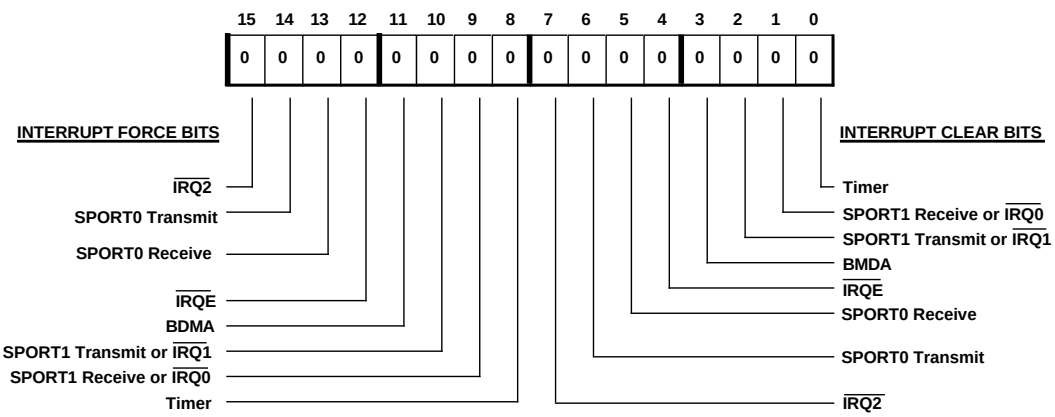
IMASK



Interrupt Registers

(Non-Memory-Mapped)

IFC



Memory Maps

Data Memory

Data Memory	*Address*
32 Memory mapped registers	0x3FFF 0x3FE0
Internal 8160 words	0x3FDF 0x2000
8K Internal (DMOVLAY=0) or External 8K (DMOVLAY=1,2)	0x1FFF 0x0000

Program Memory

Program Memory	*Address*
8K Internal (PMOVLAY = 0, MMAP = 0) or **External 8K** (PMOVLAY = 1 or 2, MMAP = 0)	0x3FFF 0x2000
8K Internal	0x1FFF 0x0000

MMAP = 0

Interrupt Vector Tables

ADSP-2181

Interrupt Source	*Interrupt Vector Address (Hex)*	
Reset (or Power up with PUCR=1)	0000	*(highest priority)*
Power Down (non-maskable)	002C	
IRQ2	0004	
IRQL1	0008	
IRQL0	000C	
SPORT0 Transmit	0010	
SPORT0 Receive	0014	
IRQE	0018	
BDMA Interrupt	001C	
SPORT1 Transmit or IRQ1	0020	
SPORT1 Receive or IRQ0	0024	
Timer	0028	*(lowest priority)*

Control/Status Registers

Symbolic names for the memory-mapped control and status registers are provided in four files included with the development software. The symbols are defined as constants equal to the register addresses, and can be used for direct addressing. To use these symbols, include the appropriate file in the your source code files with the assembler's .INCLUDE directive:

Filename	*Include Directive To Use:*
DEF2181.H	`.INCLUDE <DEF2181.H>;`

Control/Status Register	*Data Memory Address*	*Assembly Code Symbol*
System Control Register	0x3FFF	`Sys_Ctrl_Reg`
Data Memory Wait State Control Register	0x3FFE	`Dm_Wait_Reg`
Timer Period	0x3FFD	`Tperiod_Reg`
Timer Count	0x3FFC	`Tcount_Reg`
Timer Scaling Factor	0x3FFB	`Tscale_Reg`
SPORT0 Multichannel Receive	0x3FFA	`Sport0_Rx_Words1`
Word Enable Register (32-bit)	0x3FF9	`Sport0_Rx_Words0`
SPORT0 Multichannel Transmit	0x3FF8	`Sport0_Tx_Words1`
Word Enable Register (32-bit)	0x3FF7	`Sport0_Tx_Words0`
SPORT0 Control Register	0x3FF6	`Sport0_Ctrl_Reg`
SPORT0 Serial Clock Divide Modulus	0x3FF5	`Sport0_Sclkdiv`
SPORT0 Rcv Frame Sync Divide Modulus	0x3FF4	`Sport0_Rfsdiv`
SPORT0 Autobuffer Control Register	0x3FF3	`Sport0_Autobuf_Ctrl`
SPORT1 Control Register	0x3FF2	`Sport1_Ctrl_Reg`
SPORT1 Serial Clock Divide Modulus	0x3FF1	`Sport1_Sclkdiv`
SPORT1 Rcv Frame Sync Divide Modulus	0x3FF0	`Sport1_Rfsdiv`
SPORT1 Autobuffer Control Register	0x3FEF	`Sport1_Autobuf_Ctrl`
Programmable Flag & Composite Select	0x3FE6	`Prog_Flag_Comp_Sel_Ctrl`
Programmable Flag Data	0x3FE5	`Prog_Flag_Data`
BDMA Word Count	0x3FE4	`BDMA_Word_Count`
BDMA Control	0x3FE3	`BDMA_Control`
BDMA External Address	0x3FE2	`BDMA_External_Adress`
BDMA Internal Address	0x3FE1	`BDMA_Internal_Address`
IDMA Control	0x3FE0	`IDMA_Control`

ADSP-2181 Registers