Annexe 14	**ADSP 2100 –Compléments sur l'Accès aux Structures de Données (Variables, Tableau, Tampon Circulaire)**	*2 pages*

Tournez la page S.V.P.

1 PROGRAMMING DATA ACCESSES

The ADSP-2100 Family Development Software supports the declaration and use of a simple data structure: one-dimensional arrays, or buffers. The array may contain a single value (a variable) or multiple values (an array). In addition, the array may be used as a circular buffer. Here is a brief discussion of each instance, with an example of how they are declared and used in assembly language. Complete syntax for all assembler directives is given in the *ADSP-2100 Family Assembler Tools Manual.*

2 Variables & Arrays

Arrays are the basic data structure of the ADSP-21xx. In our literature, the word "array" and the expression "data buffer" (as well as "variable") are used interchangeably. Arrays are declared with assembler directives and can be referenced indirectly and by name, can be initialized from immediate values in a directive or from external data files, and can be linear or circular with automatic wraparound.

An array is declared with a directive such as

```
.VAR/DM coefficients[128];
```

This declares an array of 128 16-bit values located in data memory (DM). The special operators ^ and % reference the address and length, respectively, of the array. It could be referenced as shown below:

```
I0=^coefficients;      {point to address of buffer}
L0=0;                   {set L register to zero}
MX0=DM(I0,M0);          {load MX0 from buffer}
```

These instructions load a value into MX0 from the beginning of the *coefficients* buffer in data memory. With the automatic post-modify of the DAGs, you could execute the second of these instructions in a loop and continuously advance through the buffer.

Alternatively, when you only need to address the first location, you can directly use the buffer name as a label in many circumstances such as

```
MX0=DM(coefficients);
```

The linker substitutes the actual address for the label.

It is also possible to initialize a complete array/buffer from a data file, using the .INIT directive:

```
.INIT coefficients: <filename.dat>;
```

This assembler directive reads the values from the file *filename.dat* into the array at link time. This feature is supported only in the simulator — data cannot be loaded directly into on-chip data memory by the hardware booting sequence.

An array or data buffer with a length of one is a simple single-word variable, and is declared in this way:

```
.VAR/DM coefficient;
```

3 Circular Buffers

A common requirement in DSP is the circular buffer. This is directly implemented by the processors' data address generators (DAGs), using the L (length) registers. First, you must declare the buffer as circular:

```
.VAR/DM/CIRC coefficients[128];
```

This identifies it to the linker for placement on the proper address boundary. Next, you must initialize the L register, typically using the assemblers's % operator (or a constant) and, in the example below, the I register and M register:

```
L0=%coefficients;    {length of circular buffer}
I0=^coefficients;    {point to first address of buffer}
M0=1;                {increment by 1 location each time}
```

Now a statement like

```
MX0=DM(I0,M0);       {load MX0 from buffer}
```

placed in a loop, cycles continuously through *coefficients* and wraps around automatically.

Annexe 15	ADSP 2100 – Jeu d'instructions	10 pages

Tournez la page S.V.P.

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
  - IFC *(write-only)*
  - CNTR
  - OWRCNTR *(write-only)*

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*

Tournez la page S.V.P.

[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.

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.

Tournez la page S.V.P.

Instruction Set Summary

MAC Instructions

[IF cond] | MR | = xop * | yop | | (SS) | ;		*Multiply*
| MF | | xop | | (SU) |		
| (US) |		
| (UU) |		
| (RND) |		
= MR + xop * | yop | | (SS) | ;		*Multiply/Accumulate*
| xop | | (SU) |		
| (US) |		
| (UU) |		
| (RND) |		
= MR − xop * | yop | | (SS) | ;		*Multiply/Subtract*
| xop | | (SU) |		
| (US) |		
| (UU) |		
| (RND) |		
= 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

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

Tournez la page S.V.P.

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

Computation with Memory Write

```
|AX0| = DM ( |I0| , |M0| )    ,    |AY0| = PM ( |I4| , |M4| ) ;
|AX1|          |I1|   |M1|         |AY1|          |I5|   |M5|
|MX0|          |I2|   |M2|         |MY0|          |I6|   |M6|
|MX1|          |I3|   |M3|         |MY1|          |I7|   |M7|
```

Data & Program Memory Read

```
|<ALU>|  ,  |AX0| = DM ( |I0| , |M0| )  ,  |AY0| = PM ( |I4| , |M4| ) ;
|<MAC>|     |AX1|          |I1|   |M1|      |AY1|          |I5|   |M5|
            |MX0|          |I2|   |M2|      |MY0|          |I6|   |M6|
            |MX1|          |I3|   |M3|      |MY1|          |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*

Tournez la page S.V.P.

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| , |M0| );                                        Modify Address Register
         |I1|   |M1|
         |I2|   |M2|
         |I3|   |M3|
         ------------
         |I4|   |M4|
         |I5|   |M5|
         |I6|   |M6|
         |I7|   |M7|

[|PUSH| STS]  [, POP CNTR]  [, POP PC]  [, POP LOOP] ;          Stack Control
 |POP|

|ENA| |SEC_REG   |   [, ...] ;                                 Mode Control
|DIS| |BIT_REV   |
      |AV_LATCH  |
      |AR_SAT    |
      |M_MODE    |
      |TIMER     |
      |G_MODE    |
      |INTS      |

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