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.)

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*

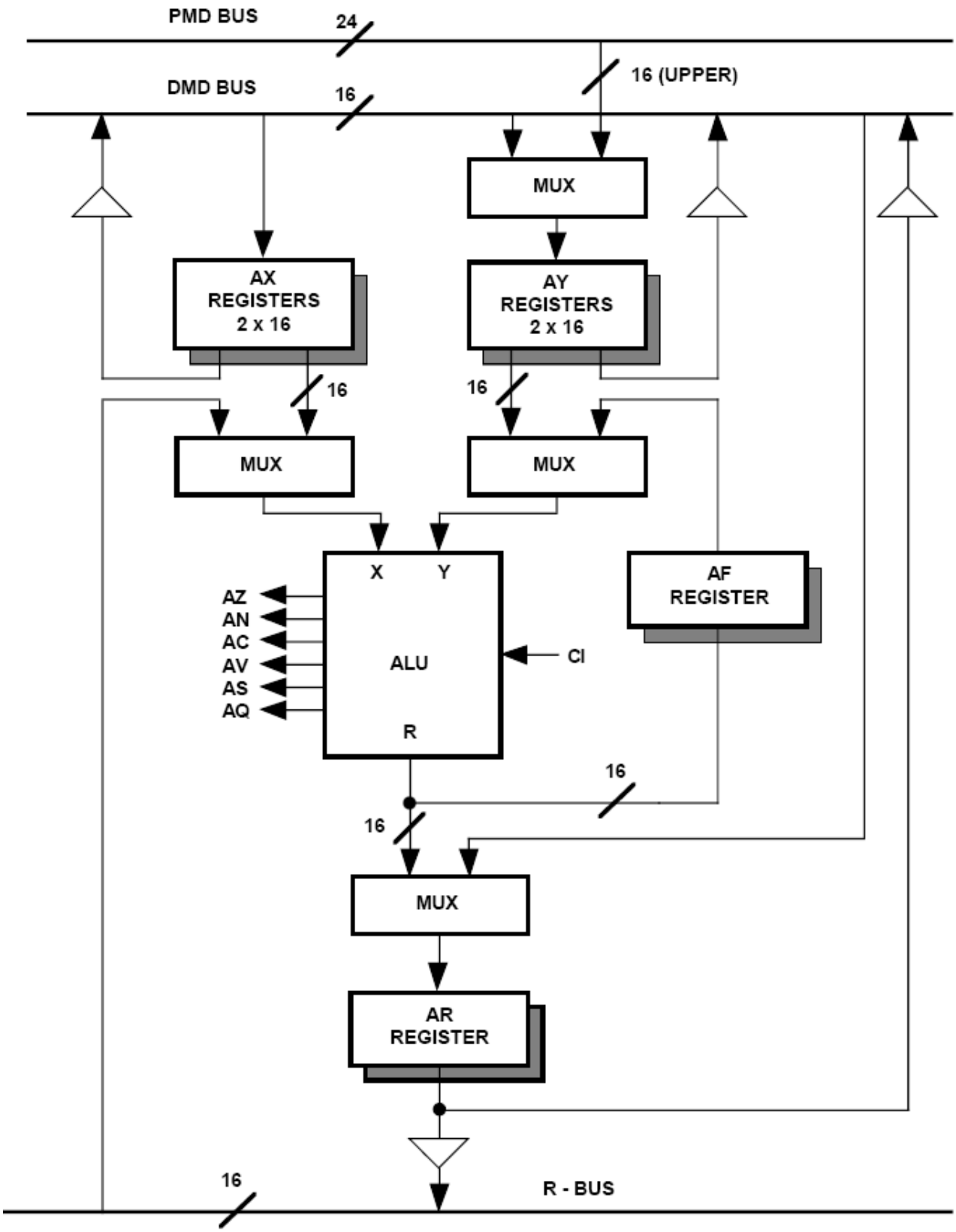
= MR [(RND)] ; *Transfer MR*

= 0 ; *Clear*

IF MV SAT MR ; *Conditional MR Saturation*

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

ALU Block Diagram

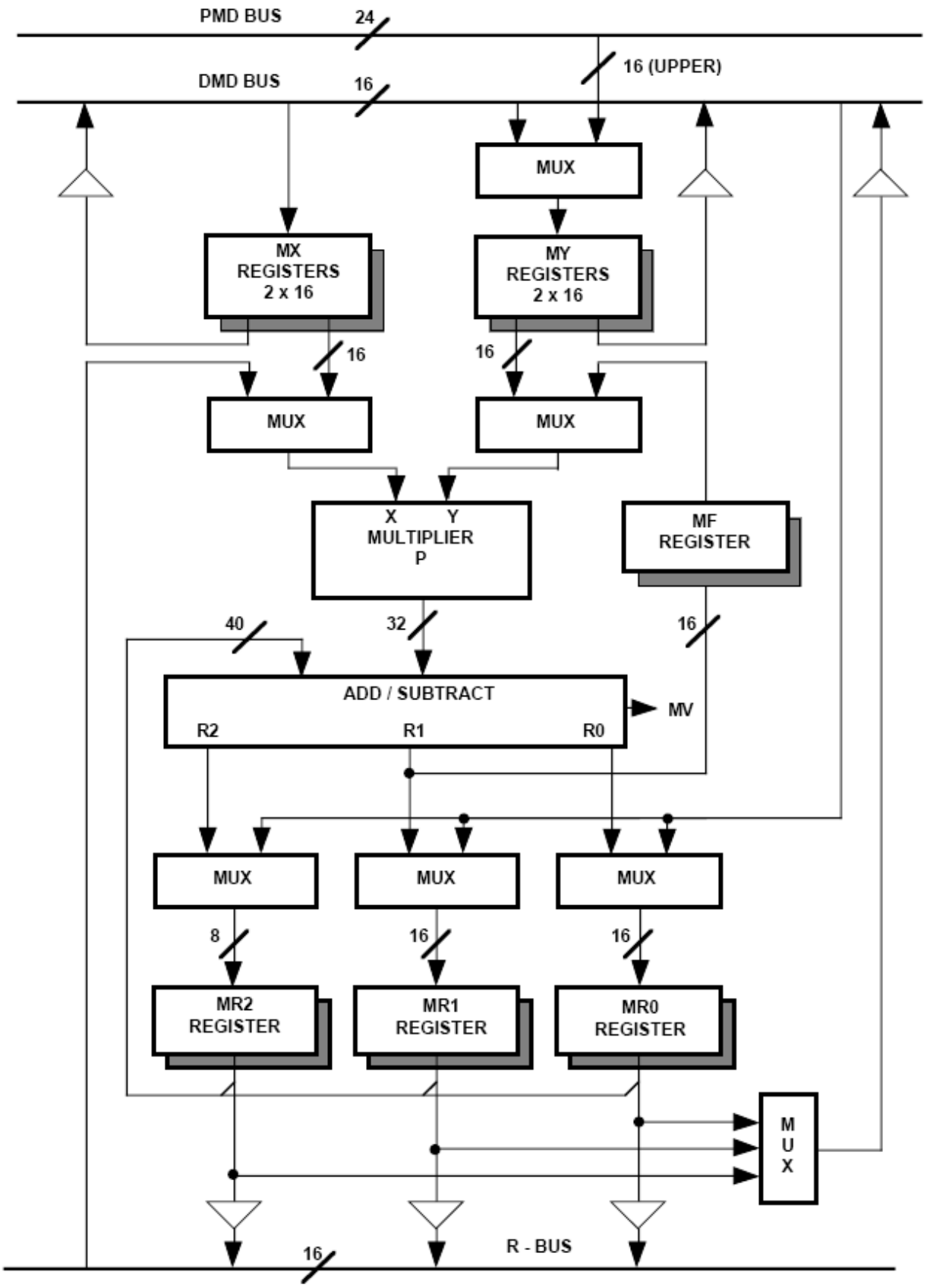


IF Condition Codes

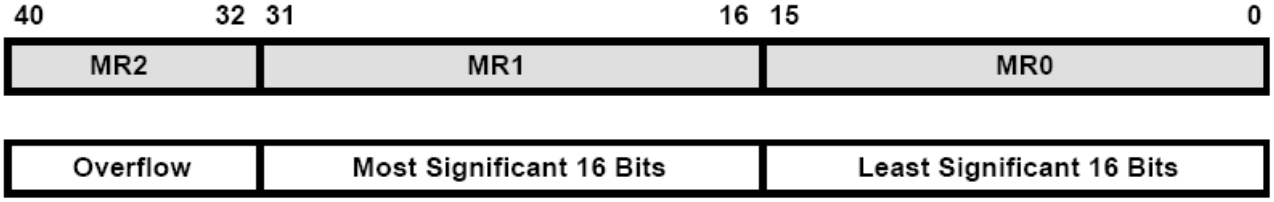
Cond	
EQ	Equal zero
NE	Not equal zero
LT	Less than zero
GE	Greater than or equal zero
LE	Less than or equal zero
GT	Greater than zero
AC	ALU carry
NOT AC	Not ALU carry
AV	ALU overflow
NOT AV	Not ALU overflow
MV	MAC overflow
NOT MV	Not MAC overflow
NEG	Xop input sign negative
POS	Xop input sign positive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

** Only for JUMP, CALL*

MAC Block Diagram



MR Registers



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.

dreg (data registers)

- AX0, AX1
- AY0, AY1
- AR
- MX0, MX1
- MY0, MY1
- MR0, MR1, MR2
- SI, SE, SR0, SR1
- 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)*

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) ;
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> ;
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 Read

<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 & 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

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 , M0 | M1 | M2 | M3) ;
(I4 | I5 | I6 | I7 , 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 ;

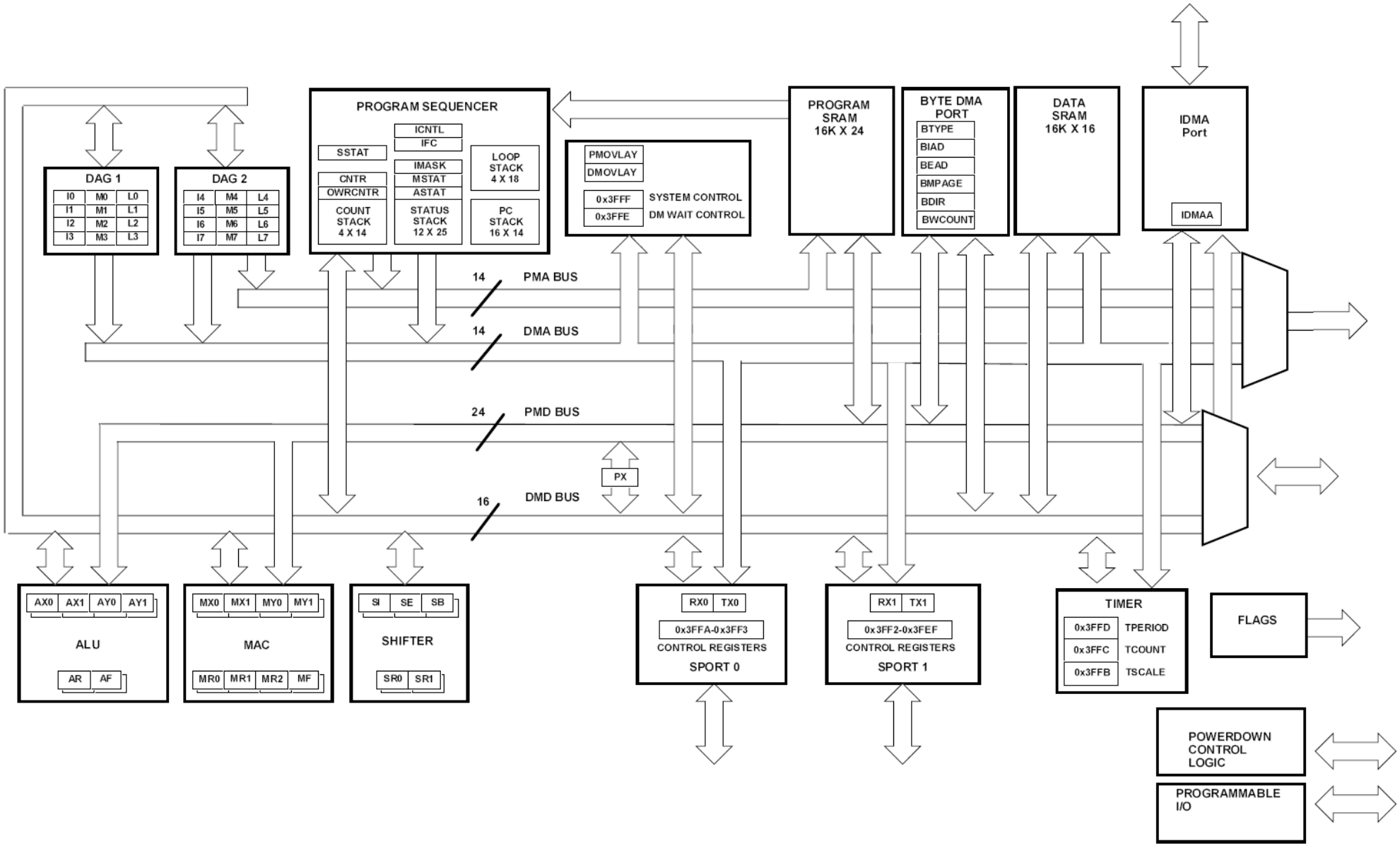
Put Processor In Idle State

IDLE (n) ;

Put Processor In Idle State And Slow Clock By a Factor Of n

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

ADSP-2181 Registers



```
//demonstracja filtraci sygnalow na ADSP2181 = komendzie Matlaba
//y = filter([0.5, 0.375, 0.25, 0.125], 1, [0.875, 0.75, 0.625, -0.875, -0.75, -0.625])

#define taps 4

.section/dm     data16;          //poczatek sekcji o nazwie (data16) w pamieci danych (dm)
.VAR/circ       buf[taps];       //bufor cykliczny na probki sygnalu

.VAR            x_input[] = 1, 2, 3, 4, 5, 6;
.VAR            DA_out;                  //w tym miejscu znajduje sie przetwornik C/A

.section/pm  pm_da;          //poczatek sekcji o nazwie (pm_da) w pamieci programu (pm)
.VAR/circ    coef[taps] = 0.5r, 0.375r, 0.25r, 0.125r;   //bufor cykliczny na wspolczynniki filtru

.section/code interrupts;                /*------Interrupt vector table------*/
   __reset: JUMP start; nop; nop; nop;   /* 0x0000: Reset vector*/

.section/code program;           //sekcja programu
start: I2=buf;                   //poczatek bufora cyrkulacyjnego danych
       L2=length(buf);           //dlugosc bufora cyrkulacyjnego = adresowanie cyrkulacyjne
       m2=1;
       m3=-1;

       I4=coef;                  //poczatek wspolczynnikow
       L4=length(coef);          //dlugosc bufora wspolczynnikow - cyrkulacyjnego
       m4=1;

       I3=x_input;               //dane
       L3=0;                     //adresowanie liniowe

       cntr = length(x_input);             //tyle jest próbek do przefiltrowania
       do next until ce;
            mx0 = dm (I3,M2);              //pobierz nowa probke z bufora sygnalu
            dm(i2,m2)=mx0;                 //umiesc probke w buforze filtru

            cntr=taps-1;                   //ustaw licznik petli na n-1 obrotow
            mr=0, my0=pm(i4,m4);           //wyzeruj rejestr mr, i pobierz pierwsza dana
            do fir1 until ce;
         fir1:   mr=mr+mx0*my0(ss), mx0=dm(i2,m2), my0=pm(i4,m4);        //filtruj

            mr=mr+mx0*my0(rnd);      //w petli bylo n-1 mnozen i dodawan, tu jest n-te
                                     //poniewaz jest to ostatni MAC to zaokraglij wynik
            modify(i2,m3);           // i2 = i2-1
            if mv sat mr;            //jesli wystapilo przepelnienie to ogranicz

next:  dm(DA_out) = mr1;             //wyslij probke przefiltrowana na przetwornik

            idle;                    //zatrzymaj sie
```

When a DO UNTIL instruction is executed, the 14-bit address of the last instruction and a 4-bit termination condition (both contained in the DO UNTIL instruction) are pushed onto the 18-bit by 4-word loop stack. Simultaneously, the PC incrementer output is pushed onto the PC stack. Since the DO UNTIL instruction is located just before the first instruction of the loop, the PC stack then contains the first loop instruction address, and the loop stack contains the last loop instruction address and termination condition. The non-empty state of the loop stack activates the loop comparator which compares the address on top of the loop stack with the address of the next instruction. When these two addresses are equal, the loop comparator notifies the next address source selector that the last instruction in the loop will be executed on the next cycle. At this point, there are three possible results depending on the type of instruction at the end of the loop. Case 1 illustrates the most typical situation. Cases 2 and 3 are also allowed but involve greater program complexity for proper execution.

Case 1

If the last instruction in the loop is not a jump, call, return, or idle, the next address circuit will select the next address based on the termination condition stored on the top of the loop stack. If the condition is false, the top address on the PC stack is selected, causing a fetch of the first instruction of the loop. If the termination condition is true, the PC incrementer is chosen, causing execution to fall out of the loop. The loop stack, PC stack, and counter stack (if being used) are then popped.

The PASS instruction performs the transfer to the AR or AF register and affects the ASTAT status flags (for xop, yop, –1, 0, 1 only). This instruction is different from a register move operation which does not affect any status flags.

1.15 Number	*Decimal Equivalent*
0x0001	0.000031
0x7FFF	0.999969
0xFFFF	–0.000031
0x8000	–1.000000

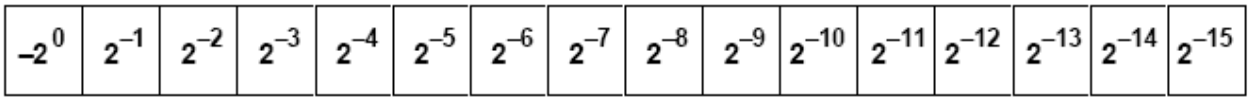


Figure 2.1 Bit Weighting For 1.15 Numbers

2.2.8 ALU Status

The ALU status bits in the ASTAT register are defined below. Complete information about the ASTAT register and specific bit mnemonics and positions is provided in the Program Control chapter.

Flag	*Name*	*Definition*
AZ	Zero	Logical NOR of all the bits in the ALU result register. True if ALU output equals zero.
AN	Negative	Sign bit of the ALU result. True if the ALU output is negative.
AV	Overflow	Exclusive-OR of the carry outputs of the two most significant adder stages. True if the ALU overflows.
AC	Carry	Carry output from the most significant adder stage.
AS	Sign	Sign bit of the ALU X input port. Affected only by the ABS instruction.
AQ	Quotient	Quotient bit generated only by the DIVS and DIVQ instructions.

Syntax	*Status Condition*	*True If:*
EQ	Equal Zero	AZ = 1
NE	Not Equal Zero	AZ = 0
LT	Less Than Zero	AN .XOR. AV = 1
GE	Greater Than or Equal Zero	AN .XOR. AV = 0
LE	Less Than or Equal Zero	(AN .XOR. AV) .OR. AZ = 1
GT	Greater Than Zero	(AN .XOR. AV) .OR. AZ = 0
AC	ALU Carry	AC = 1
NOT AC	Not ALU Carry	AC = 0
AV	ALU Overflow	AV = 1
NOT AV	Not ALU Overflow	AV = 0
MV	MAC Overflow	MV = 1
NOT MV	Not MAC Overflow	MV = 0
NEG	X Input Sign Negative	AS = 1
POS	X Input Sign Positive	AS = 0
CE	Counter Expired	
FOREVER	Always	

Table 3.1 DO UNTIL Termination Condition Logic

When a DO UNTIL instruction is executed, the 14-bit address of the last instruction and a 4-bit termination condition (both contained in the DO UNTIL instruction) are pushed onto the 18-bit by 4-word loop stack. Simultaneously, the PC incrementer output is pushed onto the PC stack. Since the DO UNTIL instruction is located just before the first instruction of the loop, the PC stack then contains the first loop instruction address, and the loop stack contains the last loop instruction address and termination condition. The non-empty state of the loop stack activates the loop comparator which compares the address on top of the loop stack with the address of the next instruction. When these two addresses are equal, the loop comparator notifies the next address source selector that the last instruction in the loop will be executed on the next cycle.