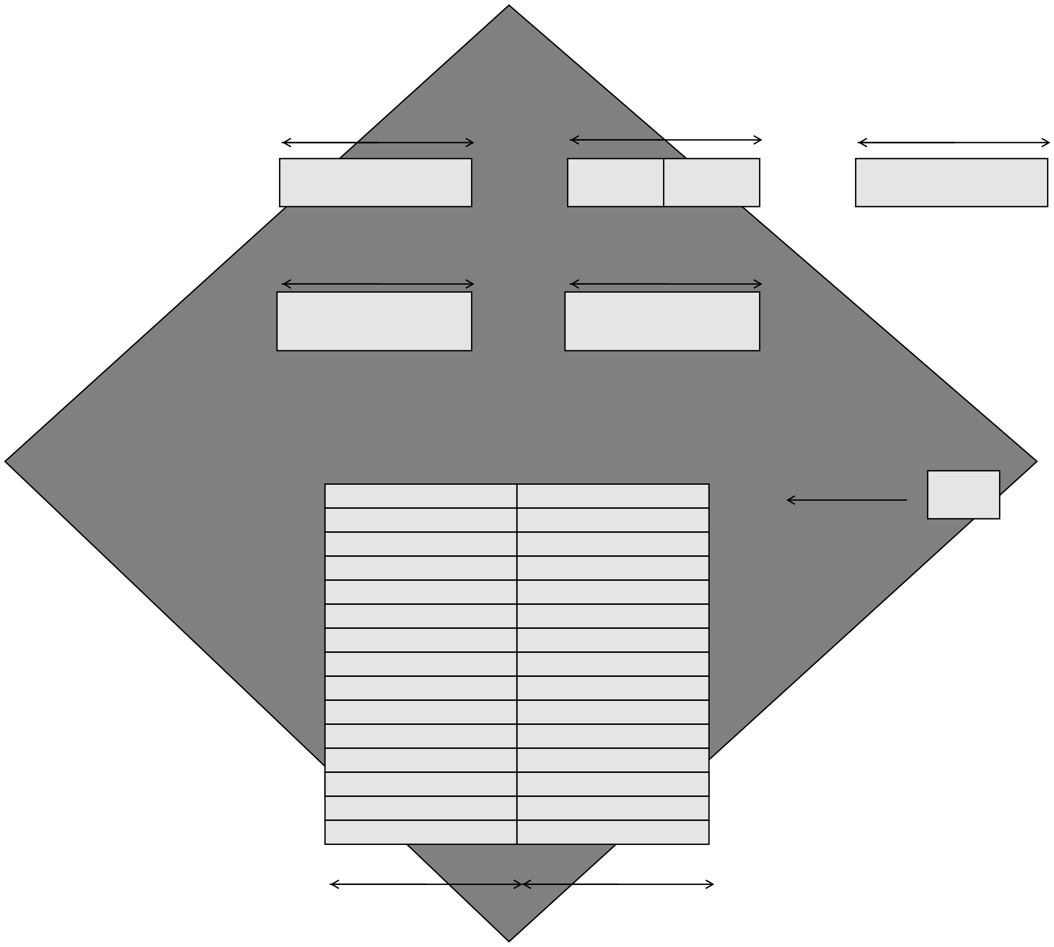
SECTION 5
PROGRAM CONTROL UNIT



SECTION CONTENTS

5.1 PROGRAM CONTROL UNIT

This section describes the hardware of the program control unit (PCU) and concludes with a description of the programming model. The instruction pipeline description is also included since understanding the pipeline is particularly important in understanding the DSP56K family of processors.

5.2 OVERVIEW

The program control unit is one of the three execution units in the central processing module (see Figure 5-2). It performs program address generation (instruction prefetch), instruction decoding, hardware DO loop control, and exception (interrupt) processing. The programmer sees the program control unit as six registers and a hardware system stack (SS) as shown in Figure 5-1. In addition to the standard program flow-control resources, such as a program counter (PC), complete status register (SR), and SS, the program control unit features registers (loop address (LA) and loop counter (LC)) dedicated to supporting the hardware DO loop instruction.

The SS is a 15-level by 32-bit separate internal memory which stores the PC and SR for subroutine calls, long interrupts, and program looping. The SS also stores the LC and LA registers. Each location in the SS is addressable as a 16-bit register, system stack high (SSH) and system stack low (SSL). The stack pointer (SP) points to the SS locations.

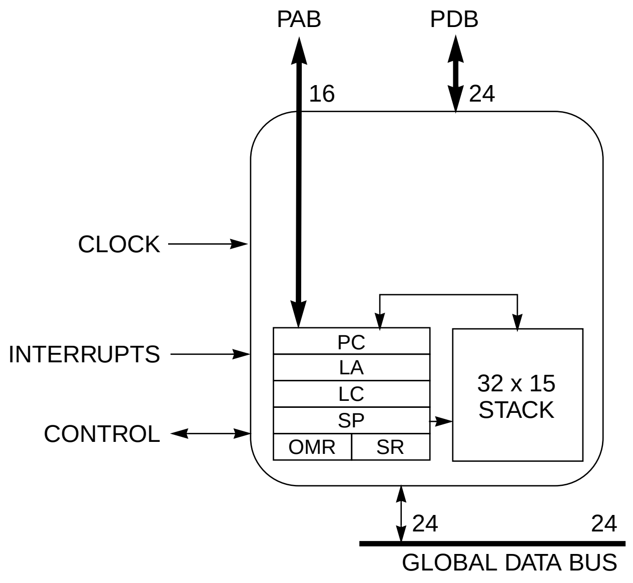


Figure 5-1 Program Address Generator

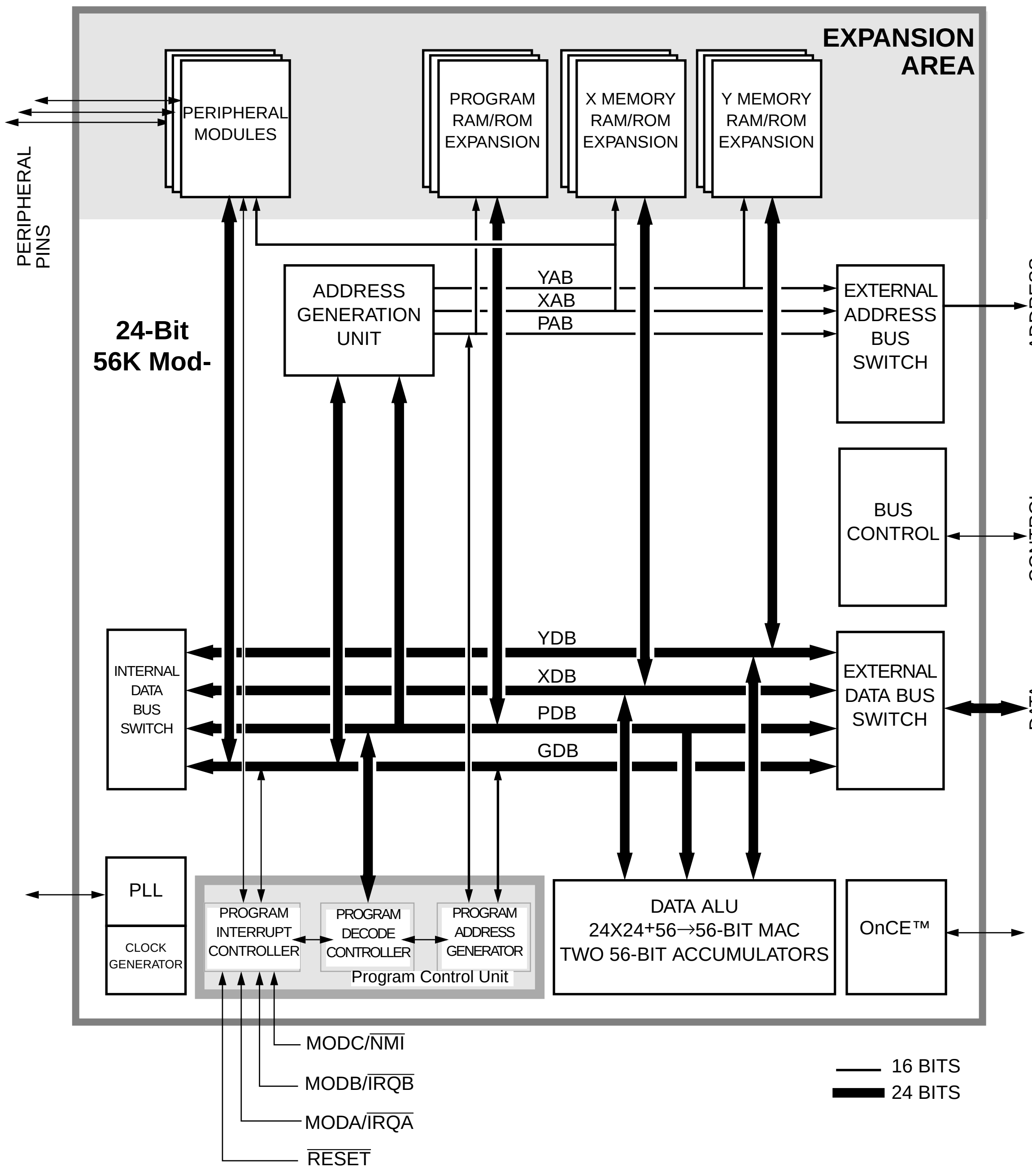


Figure 5-2 DSP56K Block Diagram

All of the PCU registers are read/write to facilitate system debugging. Although none of the registers are 24 bits, they are read or written over 24-bit buses. When they are read, the least significant bits (LSBs) are significant, and the most significant bits (MSBs) are zeroed as appropriate. When they are written, only the appropriate LSBs are significant, and the MSBs are written as don’t care.

The program control unit implements a three-stage (prefetch, decode, execute) pipeline and controls the five processing states of the DSP: normal, exception, reset, wait, and stop.

5.3 PROGRAM CONTROL UNIT (PCU) ARCHITECTURE

The PCU consists of three hardware blocks: the program decode controller (PDC), the program address generator (PAG), and the program interrupt controller (PIC).

5.3.1 Program Decode Controller

The PDC contains the program logic array decoders, the register address bus generator, the loop state machine, the repeat state machine, the condition code generator, the interrupt state machine, the instruction latch, and the backup instruction latch. The PDC decodes the 24-bit instruction loaded into the instruction latch and generates all signals necessary for pipeline control. The backup instruction latch stores a duplicate of the prefetched instruction to optimize execution of the repeat (REP) and jump (JMP) instructions.

5.3.2 Program Address Generator (PAG)

The PAG contains the PC, the SP, the SS, the operating mode register (OMR), the SR, the LC register, and the LA register (see Figure 5-1).

The PAG provides hardware dedicated to support loops, which are frequent constructs in DSP algorithms. A DO instruction loads the LC register with the number of times the loop should be executed, loads the LA register with the address of the last instruction word in the loop (fetched during one loop pass), and asserts the loop flag in the SR. The DO instruction also supports nested loops by stacking the contents of the LA, LC, and SR prior to the execution of the instruction. Under control of the PAG, the address of the first instruction in the loop is also stacked so the loop can be repeated with no overhead. While the loop flag in the SR is asserted, the loop state machine (in the PDC) will compare the PC contents to the contents of the LA to determine if the last instruction word in the loop was fetched. If the last word was fetched, the LC contents are tested for one. If LC is not equal to one, then it is decremented, and the SS is read to update the PC with the address of the first instruction in the loop, effectively executing an automatic branch. If the LC is equal to one, then the LC, LA, and the loop flag in the SR are restored with the stack contents, while instruction fetches continue at the incremented PC value (LA + 1). More information about the LA and LC appears in Section 5.3.4 Instruction Pipeline Format.

The repeat (REP) instruction loads the LC with the number of times the next instruction is to be repeated. The instruction to be repeated is only fetched once, so throughput is increased by reducing external bus contention. However, REP instructions are not interruptible since they are fetched only once. A single-instruction DO loop can be used in place of a REP instruction if interrupts must be allowed.

5.3.3 Program Interrupt Controller

The PIC receives all interrupt requests, arbitrates among them, and generates the interrupt vector address.

Interrupts have a flexible priority structure with levels that can range from zero to three. Levels 0 (lowest level), 1, and 2 are maskable. Level 3 is the highest interrupt priority level (IPL) and is not maskable. Two interrupt mask bits in the SR reflect the current IPL and indicate the level needed for an interrupt source to interrupt the processor. Interrupts cause the DSP to enter the exception processing state which is discussed fully in SECTION 7 – PROCESSING STATES.

The four external interrupt sources include three external interrupt request inputs ($\overline{\mathrm{IRQA}}$, $\overline{\mathrm{IRQB}}$, and $\overline{\mathrm{NMI}}$) and the $\overline{\mathrm{RESET}}$ pin. $\overline{\mathrm{IRQA}}$ and $\overline{\mathrm{IRQB}}$ can be either level sensitive or negative edge triggered. The nonmaskable interrupt ($\overline{\mathrm{NMI}}$) is edge sensitive and is a level 3 interrupt. MODA/$\overline{\mathrm{IRQA}}$, MODB/$\overline{\mathrm{IRQB}}$, and MODC/$\overline{\mathrm{NMI}}$ pins are sampled when $\overline{\mathrm{RESET}}$ is deasserted. The sampled values are stored in the operating mode register (OMR) bits MA, MB, and MC, respectively (see Section 5.4.3 for information on the OMR). Only the fourth external interrupt, $\overline{\mathrm{RESET}}$, and Illegal Instruction have higher priority than $\overline{\mathrm{NMI}}$.

The PIC also arbitrates between the different I/O peripherals. The currently selected peripheral supplies the correct vector address to the PIC.

5.3.4 Instruction Pipeline Format

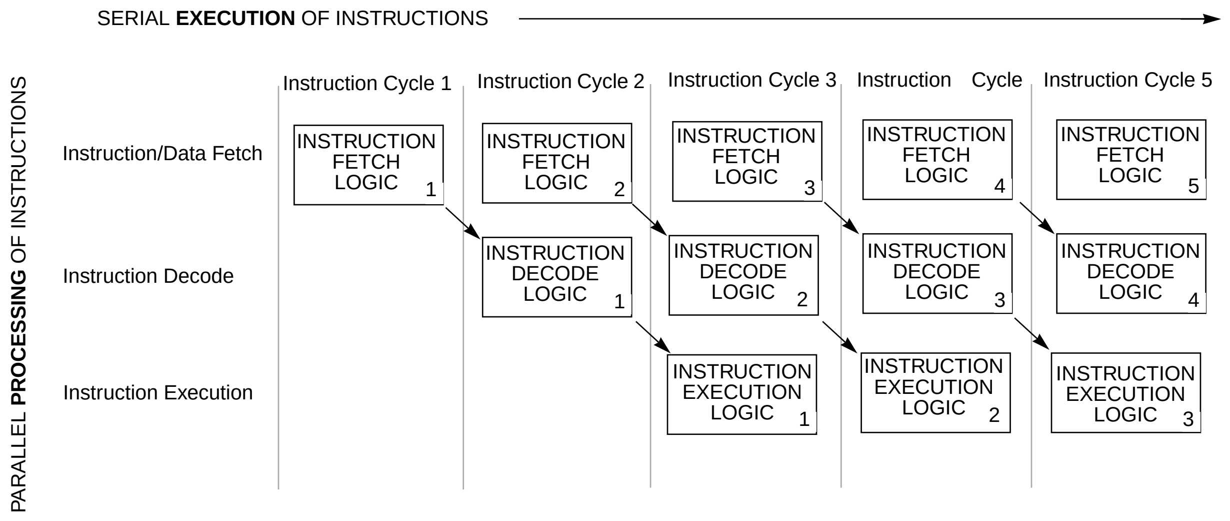
The program control unit uses a three-level pipelined architecture in which concurrent instruction fetch, decode, and execution occur. This pipelined operation remains essentially hidden from the user and makes programming straightforward. The pipeline is illustrated in Figure 5-3, which shows the operations of each of the execution units and all initial conditions necessary to follow the execution of the instruction sequence shown in the figure. The pipeline is described in more detail in Section 7.2.1 Instruction Pipeline.

The first instruction, I1, should be interpreted as follows: multiply the contents of X0 by the contents of Y0, add the product to the contents already in accumulator A, round the result to the “nearest even,” store the result back in accumulator A, move the contents in X data memory (pointed to by R0) into X0 and postincrement R0, and move the contents in Y data memory (pointed to by R4) into Y1 and postincrement R4. The second instruction, I2, should be interpreted as follows: clear accumulator A, move the contents in X0 into the location in X data memory pointed to by R0 and postincrement R0. Before the clear oper-

EXAMPLE PROGRAM SEGMENT

```
Instruction 1   MACR    X0,Y1,A    X:(R0)+,X0    Y:(R4)+,Y1
Instruction 2   CLR     A          X0,X:(R0)+    A,Y:(R4)-
Instruction 3   MAC     X0,Y1,A    X:(R0)+,X0    Y:(R4)+,Y1
```

SEQUENCE OF OPERATIONS


SERIAL **EXECUTION** OF INSTRUCTIONS
PARALLEL **PROCESSING** OF INSTRUCTIONS
Instruction/Data Fetch
Instruction Decode
Instruction Execution
Instruction Cycle 1
Instruction Cycle 2
Instruction Cycle 3
Instruction Cycle
Instruction Cycle 5
INSTRUCTION FETCH LOGIC 1
INSTRUCTION FETCH LOGIC 2
INSTRUCTION FETCH LOGIC 3
INSTRUCTION FETCH LOGIC 4
INSTRUCTION FETCH LOGIC 5
INSTRUCTION DECODE LOGIC 1
INSTRUCTION DECODE LOGIC 2
INSTRUCTION DECODE LOGIC 3
INSTRUCTION DECODE LOGIC 4
INSTRUCTION EXECUTION LOGIC 1
INSTRUCTION EXECUTION LOGIC 2
INSTRUCTION EXECUTION LOGIC 3

EXECUTION OF EXAMPLE PROGRAM

		Instruction Cycle 1	Instruction Cycle 2	Instruction Cycle 3	Instruction Cycle 4	Instruction Cycle 5
INSTRUCTION FETCH → INSTRUCTION DECODE → INSTRUCTION EXECUTION →		I1	I2 I1	I3 I2 I1	I4 I3 I2	I5 I4 I3
PARALLEL OPERATIONS	INITIAL CONDITIONS					
ADDRESS UPDATE (AGU)	R0=\$0005 R4=\$0008	→	→	R0=5+1 R4=8+1	R0=6+1 R4=9–1	R0=7+1 R4=8+1
INSTRUCTION EXECUTION (DATA ALU)	A: A2=\$00 A1=\$000066 A0=\$000000 X0=\$400000 Y1=\$000077	→	→	A: A2=\$00 A1=\$0000A2 A0=\$000000 X0=\$000005 Y1=\$000008	A: A2=\$00 A1=\$000000 A0=\$000000 X0=\$000005 Y1=\$000008	A: A2=\$00 A1=\$000000 A0=\$000050 X0=\$000007 Y1=\$000008
X MEMORY AT ADDRESS \$0005 \$0006 \$0007	DATA \$000005 \$000006 \$000007	→	→	\$000005 \$000006 \$000007	\$000005 \$000005 \$000007	\$000005 \$000005 \$000007
Y MEMORY AT ADDRESS \$0008 \$0009	DATA \$000008 \$000009	→	→	\$000008 \$000009	\$000008 \$0000A2	\$000008 \$0000A2

Figure 5-3 Three-Stage Pipeline

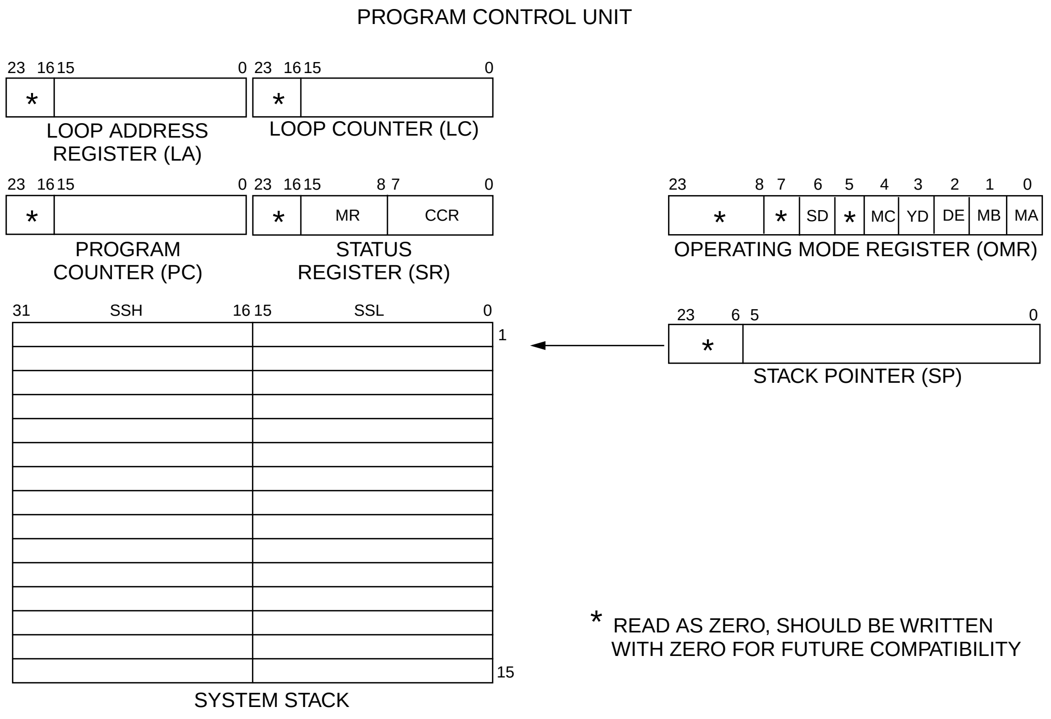


Figure 5-4 Program Control Unit Programming Model

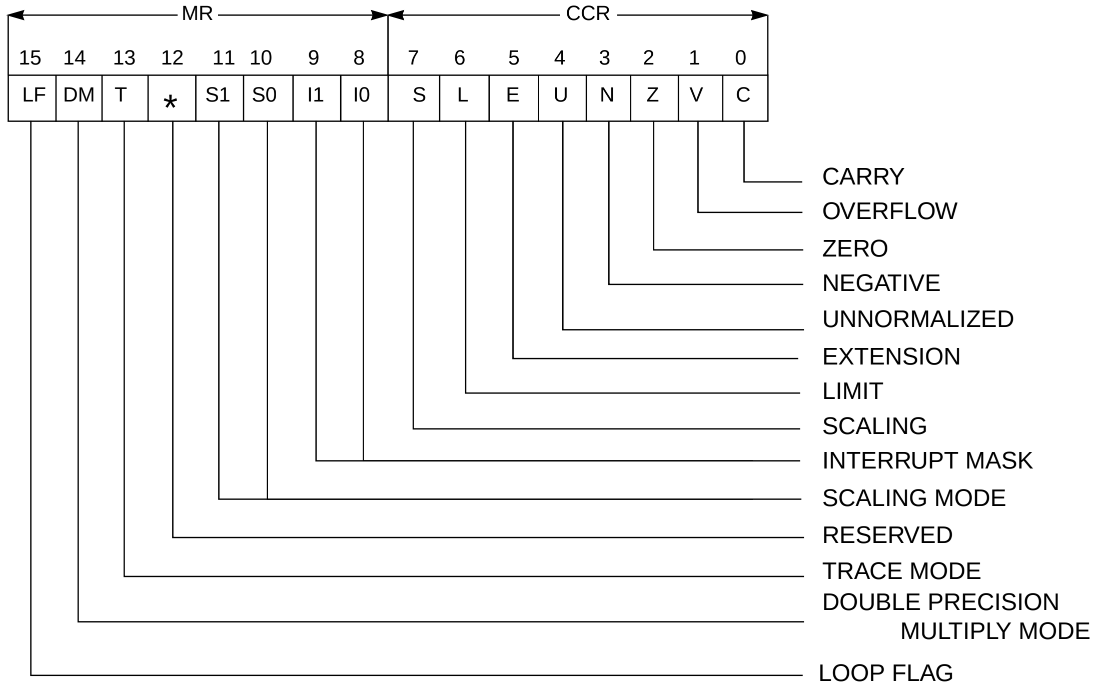
ation, move the contents in accumulator A into the location in Y data memory pointed to by R4 and postdecrement R4. The third instruction, I3, is the same as I1, except the rounding operation is not performed.

5.4 PROGRAMMING MODEL

The program control unit features LA and LC registers which support the DO loop instruction and the standard program flow-control resources, such as a PC, complete SR, and SS. With the exception of the PC, all registers are read/write to facilitate system debugging. Figure 5-4 shows the program control unit programming model with the six registers and SS. The following paragraphs give a detailed description of each register.

5.4.1 Program Counter

This 16-bit register contains the address of the next location to be fetched from program memory space. The PC can point to instructions, data operands, or addresses of operands. References to this register are always inherent and are implied by most instructions.



All bits are cleared after hardware reset except bits 8 and 9 which are set to ones.
Bits 12 and 16 to 23 are reserved, read as zero and should be written with zero for future compatibility

Figure 5-5 Status Register Format

This special-purpose address register is stacked when program looping is initialized, when a JSR is performed, or when interrupts occur (except for no-overhead fast interrupts).

5.4.2 Status Register

The 16-bit SR consists of a mode register (MR) in the high-order eight bits and a condition code register (CCR) in the low-order eight bits, as shown in Figure 5-5. The SR is stacked when program looping is initialized, when a JSR is performed, or when interrupts occur, (except for no-overhead fast interrupts).

The MR is a special purpose control register which defines the current system state of the processor. The MR bits are affected by processor reset, exception processing, the DO, end current DO loop (ENDDO), return from interrupt (RTI), and SWI instructions and by instructions that directly reference the MR register, such as OR immediate to control register (ORI) and AND immediate to control register (ANDI). During processor reset, the interrupt mask bits of the MR will be set. The scaling mode bits, loop flag, and trace bit will be cleared.

The CCR is a special purpose control register that defines the current user state of the processor. The CCR bits are affected by data arithmetic logic unit (ALU) operations, parallel move operations, and by instructions that directly reference the CCR (ORI and ANDI). The CCR bits are not affected by parallel move operations unless data limiting occurs when reading the A or B accumulators. During processor reset, all CCR bits are cleared.

5.4.2.1 Carry (Bit 0)

The carry (C) bit is set if a carry is generated out of the MSB of the result in an addition. This bit is also set if a borrow is generated in a subtraction. The carry or borrow is generated from bit 55 of the result. The carry bit is also affected by bit manipulation, rotate, and shift instructions. Otherwise, this bit is cleared.

5.4.2.2 Overflow (Bit 1)

The overflow (V) bit is set if an arithmetic overflow occurs in the 56-bit result. This bit indicates that the result cannot be represented in the accumulator register; thus, the register has overflowed. Otherwise, this bit is cleared.

5.4.2.3 Zero (Bit 2)

The zero (Z) bit is set if the result equals zero; otherwise, this bit is cleared.

5.4.2.4 Negative (Bit 3)

The negative (N) bit is set if the MSB (bit 55) of the result is set; otherwise, this bit is cleared.

5.4.2.5 Unnormalized (Bit 4)

The unnormalized (U) bit is set if the two MSBs of the most significant product (MSP) portion of the result are identical. Otherwise, this bit is cleared. The MSP portion of the A or B accumulators, which is defined by the scaling mode and the U bit, is computed as follows:

S1	S0	Scaling Mode	U Bit Computation
0	0	No Scaling	$U = \overline{(\text{Bit } 47 \oplus \text{Bit } 46)}$
0	1	Scale Down	$U = \overline{(\text{Bit } 48 \oplus \text{Bit } 47)}$
1	0	Scale Up	$U = \overline{(\text{Bit } 46 \oplus \text{Bit } 45)}$

5.4.2.6 Extension (Bit 5)

The extension (E) bit is cleared if all the bits of the integer portion of the 56-bit result are all ones or all zeros; otherwise, this bit is set. The integer portion, defined by the scaling mode and the E bit, is computed as follows:

S1	S0	Scaling Mode	Integer Portion
0	0	No Scaling	Bits 55,54........48,47
0	1	Scale Down	Bits 55,54........49,48
1	0	Scale Up	Bits 55,54........47,46

If the E bit is cleared, then the low-order fraction portion contains all the significant bits; the high-order integer portion is just sign extension. In this case, the accumulator extension register can be ignored. If the E bit is set, it indicates that the accumulator extension register is in use.

5.4.2.7 Limit (Bit 6)

The limit (L) bit is set if the overflow bit is set. The L bit is also set if the data shifter/limiter circuits perform a limiting operation; otherwise, it is not affected. The L bit is cleared only by a processor reset or by an instruction that specifically clears it, which allows the L bit to be used as a latching overflow bit (i.e., a “sticky” bit). L is affected by data movement operations that read the A or B accumulator registers.

5.4.2.8 Scaling Bit (Bit 7)

The scaling bit (S) is used to detect data growth, which is required in Block Floating Point FFT operation. Typically, the bit is tested after each pass of a radix 2 FFT and, if it is set, the scaling mode should be activated in the next pass. The Block Floating Point FFT algorithm is described in the Motorola application note APR4/D, “Implementation of Fast Fourier Transforms on Motorola’s DSP56000/DSP56001 and DSP96002 Digital Signal Processors.” This bit is computed according to the following logical equations when the result of accumulator A or B is moved to XDB or YDB. It is a “sticky” bit, cleared only by an instruction that specifically clears it.

If	S1=0 and S0=0 (no scaling)
then	S = (A46 XOR A45) OR (B46 XOR B45)
If	S1=0 and S0=1 (scale down)
then	S = (A47 XOR A46) OR (B47 XOR B46)
If	S1=1 and S0=0 (scale up)
then	S = (A45 XOR A44) OR (B45 XOR B44)
If	S1=1 and S0=1 (reserved)
then	the S flag is undefined.

where Ai and Bi means bit i in accumulator A or B.

5.4.2.9 Interrupt Masks (Bits 8 and 9)

The interrupt mask bits, I1 and I0, reflect the current IPL of the processor and indicate the IPL needed for an interrupt source to interrupt the processor. The current IPL of the processor can be changed under software control. The interrupt mask bits are set during hardware reset but not during software reset.

I1	I0	Exceptions Permitted	Exceptions Masked
0	0	IPL 0,1,2,3	None
0	1	IPL 1,2,3	IPL 0
1	0	IPL 2,3	IPL 0,1
1	1	IPL 3	IPL 0,1,2

5.4.2.10 Scaling Mode (Bits 10 and 11)

The scaling mode bits, S1 and S0, specify the scaling to be performed in the data ALU shifter/limiter, and also specify the rounding position in the data ALU multiply-accumulator (MAC). The scaling modes are shown in the following table:

S1	S0	Rounding Bit	Scaling Mode
0	0	23	No Scaling
0	1	24	Scale Down (1-Bit Arithmetic Right Shift)
1	0	22	Scale Up (1-Bit Arithmetic Left Shift)
1	1	—	Reserved for Future Expansion

The scaling mode affects data read from the A or B accumulator registers out to the XDB and YDB. Different scaling modes can occur with the same program code to allow dynamic scaling. Dynamic scaling facilitates block floating-point arithmetic. The scaling mode also affects the MAC rounding position to maintain proper rounding when different portions of the accumulator registers are read out to the XDB and YDB. The scaling mode bits, which are cleared at the start of a long interrupt service routine, are also cleared during a processor reset.

5.4.2.11 Reserved Status (Bit 12)

This bits is reserved for future expansion and will read as zero during DSP read operations.

5.4.2.12 Trace Mode (Bit 13)

The trace mode (T) bit specifies the tracing function of the DSP56000/56001 only. (With other members of the DSP56K family, use the OnCE trace mode described in Section 10.5.) For the DSP56000/56001, if the T bit is set at the beginning of any instruction execution, a trace exception will be generated after the instruction execution is completed. If the T bit is cleared, tracing is disabled and instruction execution proceeds normally. If a long interrupt is executed during a trace exception, the SR with the trace bit set will be stacked, and the trace bit in the SR is cleared (see SECTION 7 – PROCESSING STATES for a complete description of a long interrupt operation). The T bit is also cleared during processor reset.

5.4.2.13 Double Precision Multiply Mode (Bit 14)

The processor is in double precision multiply mode when this bit is set. (See Section 3.4 for detailed information on the double precision multiply mode.) When the DM bit is set, the operations performed by the MPY and MAC instructions change so that a double precision 48-bit by 48-bit double precision multiplication can be performed in six instruc-

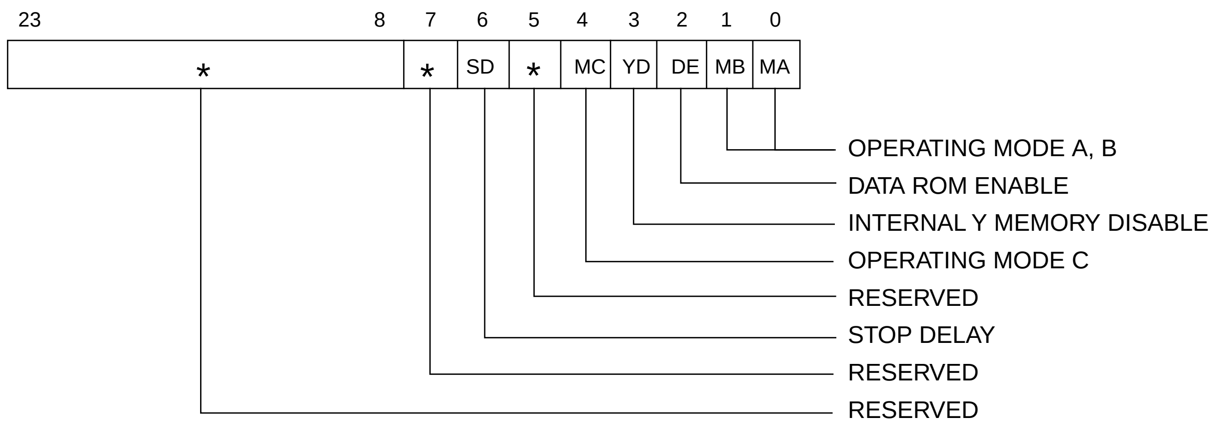


Figure 5-6 OMR Format

tions. The DSP56K software simulator accurately shows how the MPY, MAC, and other Data ALU instructions operate while the processor is in the double precision multiply mode.

5.4.2.14 Loop Flag (Bit 15)

The loop flag (LF) bit is set when a program loop is in progress. It detects the end of a program loop. The LF is the only SR bit that is restored when a program loop is terminated. Stacking and restoring the LF when initiating and exiting a program loop, respectively, allow the nesting of program loops. At the start of a long interrupt service routine, the SR (including the LF) is pushed on the SS and the SR LF is cleared. When returning from the long interrupt with an RTI instruction, the SS is pulled and the LF is restored. During a processor reset, the LF is cleared.

5.4.3 Operating Mode Register

The OMR is a 24-bit register (only six bits are defined) that sets the current operating mode of the processor. Each chip in the DSP56K family of processors has its own set of operating modes which determine the memory maps for program and data memories, and the startup procedure that occurs when the chip leaves the reset state. The OMR bits are only affected by processor reset and by the ANDI, ORI, and MOVEC instructions, which directly reference the OMR.

The OMR format with all of its defined bits is shown in Figure 5-6. For product-specific OMR bit definitions, see the individual chip’s user manual for details on its respective operating modes.

5.4.4 System Stack

The SS is a separate 15X32-bit internal memory divided into two banks, the SSH and the

SSL, each 16 bits wide. The SSH stores the PC contents, and the SSL stores the SR contents for subroutine calls, long interrupts, and program looping. The SS will also store the LA and LC registers. The SS is in stack memory space; its address is always inherent and implied by the current instruction.

The contents of the PC and SR are pushed on the top location of the SS when a subroutine call or long interrupt occurs. When a return from subroutine (RTS) occurs, the contents of the top location in the SS are pulled and put in the PC; the SR is not affected. When an RTI occurs, the contents of the top location in the SS are pulled to both the PC and SR.

The SS is also used to implement no-overhead nested hardware DO loops. When the DO instruction is executed, the LA:LC are pushed on the SS, then the PC:SR are pushed on the SS. Since each SS location can be addressed as separate 16-bit registers (SSH and SSL), software stacks can be created for unlimited nesting.

The SS can accommodate up to 15 long interrupts, seven DO loops, 15 JSRs, or combinations thereof. When the SS limit is exceeded, a nonmaskable stack error interrupt occurs, and the PC is pushed to SS location zero, which is not implemented in hardware. The PC will be lost, and there will be no SP from the stack interrupt routine to the program that was executing when the error occurred.

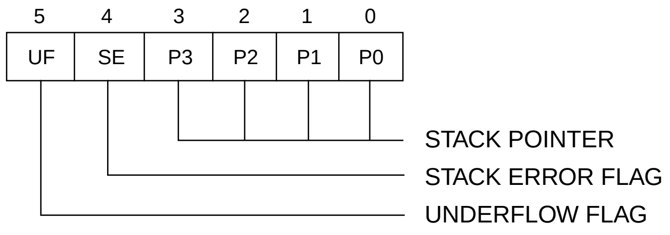


Figure 5-7 Stack Pointer Register Format

5.4.5 Stack Pointer Register

The 6-bit SP register indicates the location of the top of the SS and the status of the SS (underflow, empty, full, and overflow). The SP register is referenced implicitly by some instructions (DO, REP, JSR, RTI, etc.) or directly by the MOVEC instruction. The SP register format is shown in Figure 5-7. The SP register works as a 6-bit counter that addresses (selects) a 15-location stack with its four LSBs. The possible SP values are shown in Figure 5-8 and described in the following paragraphs.

5.4.5.1 Stack Pointer (Bits 0–3)

The SP points to the last location used on the SS. Immediately after hardware reset, these bits are cleared (SP=0), indicating that the SS is empty.

Data is pushed onto the SS by incrementing the SP, then writing data to the location to which the SP points. An item is pulled off the stack by copying it from that location and then by decrementing the SP.

5.4.5.2 Stack Error Flag (Bit 4)

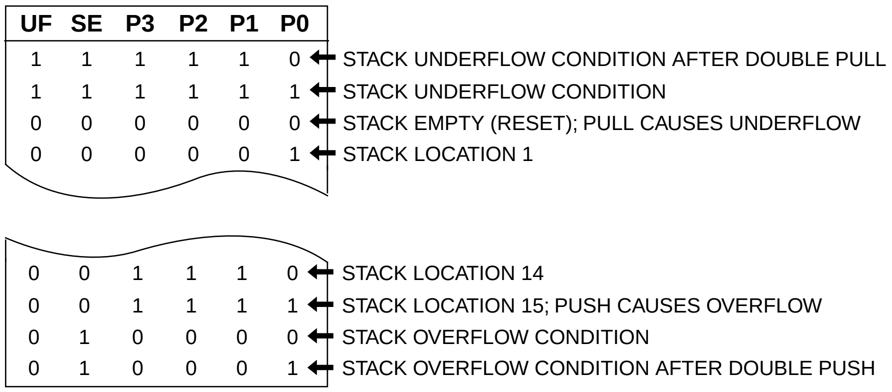
The stack error flag indicates that a stack error has occurred, and the transition of the stack error flag from zero to one causes a priority level-3 stack error exception.

When the stack is completely full, the SP reads 001111, and any operation that pushes data onto the stack will cause a stack error exception to occur. The SR will read 010000 (or 010001 if an implied double push occurs).

Any implied pull operation with SP equal to zero will cause a stack error exception, and the SP will read 111111 (or 111110 if an implied double pull occurs).

The stack error flag is a “sticky bit” which, once set, remains set until cleared by the user. There is a sequence of instructions that can cause a stack overflow and, without the sticky bit, would not be detected because the stack pointer is decremented before the stack error interrupt is taken. The sticky bit keeps the stack error bit set until the user clears it by writing a zero to SP bit 4. It also latches the overflow/underflow bit so that it cannot be changed by stack pointer increments or decrements as long as the stack error is set. The overflow/underflow bit remains latched until the first move to SP is executed.

Note: When SP is zero (stack empty), instructions that read the stack without SP postdecrement and instructions that write to the stack without SP preincrement do not cause a stack error exception (i.e., 1) DO SSL,xxxx 2) REP SSL 3) MOVEC or move peripheral



UF	SE	P3	P2	P1	P0	
1	1	1	1	1	0	STACK UNDERFLOW CONDITION AFTER DOUBLE PULL
1	1	1	1	1	1	STACK UNDERFLOW CONDITION
0	0	0	0	0	0	STACK EMPTY (RESET); PULL CAUSES UNDERFLOW
0	0	0	0	0	1	STACK LOCATION 1
0	0	1	1	1	0	STACK LOCATION 14
0	0	1	1	1	1	STACK LOCATION 15; PUSH CAUSES OVERFLOW
0	1	0	0	0	0	STACK OVERFLOW CONDITION
0	1	0	0	0	1	STACK OVERFLOW CONDITION AFTER DOUBLE PUSH

Figure 5-8 SP Register Values

data (MOVEP) when SSL is specified as a source or destination).

5.4.5.3 Underflow Flag (Bit 5)

The underflow flag is set when a stack underflow occurs. The underflow flag is a “sticky bit” when the stack error flag is set. That is, when the stack error flag is set, the underflow flag will not change state. The combination of “underflow=1” and “stack error=0” is an illegal combination and will not occur unless it is forced by the user. If this condition is forced by the user, the hardware will correct itself based on the result of the next stack operation.

5.4.5.4 Reserved Stack Pointer Registration (Bits 6–23)

SP register bits 6 through 23 are reserved for future expansion and will read as zero during read operations.

5.4.6 Loop Address Register

The LA is a read/write register which is stacked into the SSH by a DO instruction and is unstacked by end-of-loop processing or by an ENDDO instruction. The contents of the LA register indicate the location of the last instruction word in a program loop. When that last instruction is fetched, the processor checks the contents of the LC register (see the following section). If the contents are not one, the processor decrements the LC and takes the next instruction from the top of the SS. If the LC is one, the PC is incremented, the loop flag is restored (pulled from the SS), the SS is purged, the LA and LC registers are pulled from the SS and restored, and instruction execution continues normally.

5.4.7 Loop Counter Register

The LC register is a special 16-bit counter which specifies the number of times a hardware program loop shall be repeated. This register is stacked into the SSL by a DO instruction and unstacked by end-of-loop processing or by execution of an ENDDO instruction. When the end of a hardware program loop is reached, the contents of the LC register are tested for one. If the LC is one, the program loop is terminated, and the LC register is loaded with the previous LC contents stored on the SS. If LC is not one, it is decremented and the program loop is repeated. The LC can be read under program control, which allows the number of times a loop will be executed to be monitored/changed dynamically. The LC is also used in the REP instruction

5.4.8 Programming Model Summary

The complete programming model for the DSP56K central processing module is shown in Figure 5-9. Programming models for the peripherals are shown in the appropriate user manuals.

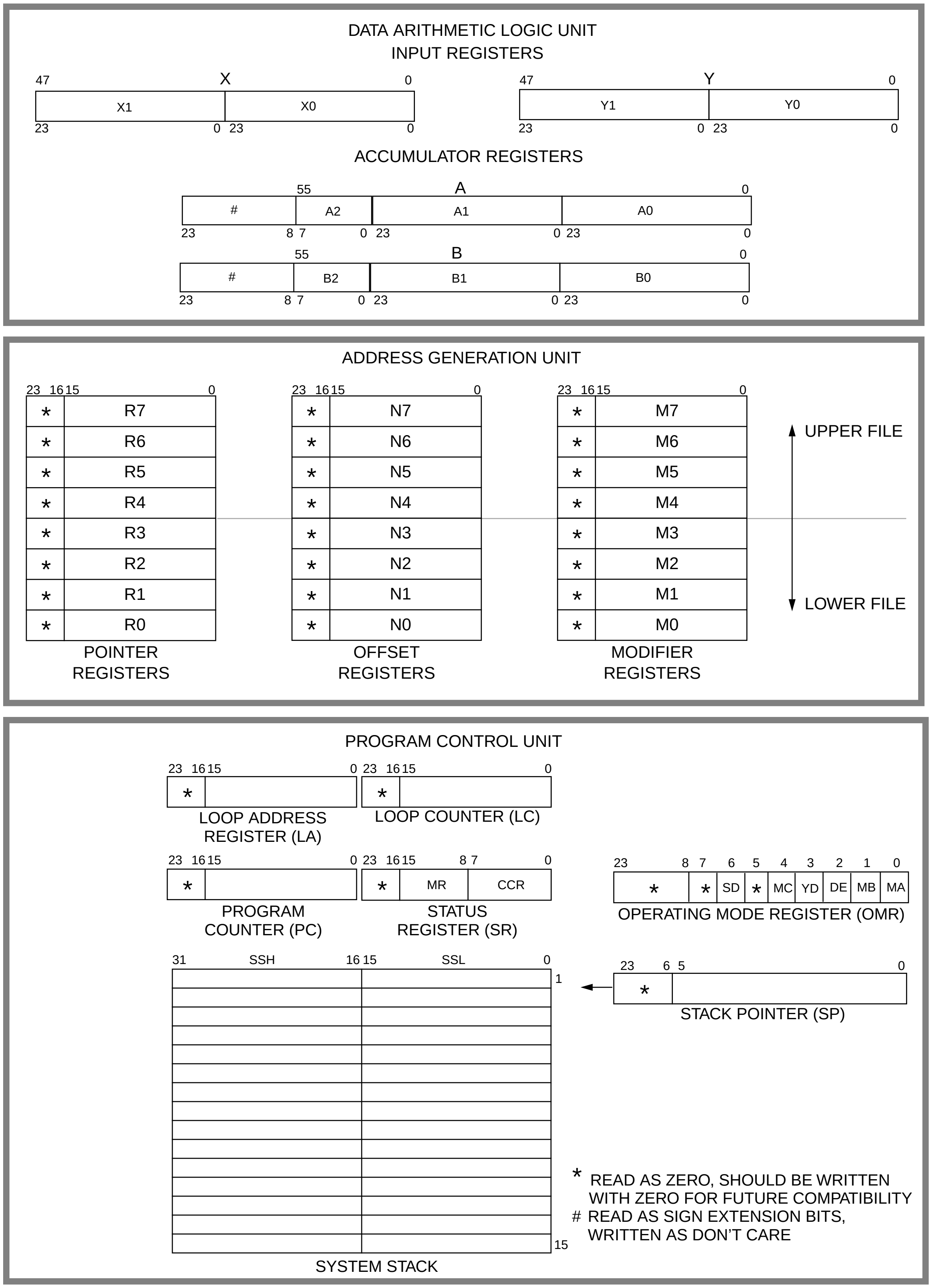


Figure 5-9 DSP56K Central Processing Module Programming Model