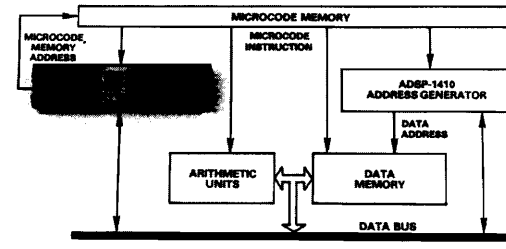
ANALOG DEVICES

Word-Slice Program Sequencer

ADSP-1401

FEATURES

- **16-Bit Microcode Addressing Capability**
- **Look-Ahead™ Pipeline**
- **Extensive Interrupt Processing, With Ten On-Chip Interrupt Vectors**
- **70ns Cycle Time; 25ns Clock-to-Address Delay**
- **64-Word RAM for Storing:**
  - **Subroutine Linkage**
  - **Jump Addresses**
  - **Counters**
  - **Status Register**
- **375mW Maximum Power Dissipation with CMOS Technology**
- **48-Pin Ceramic or Plastic DIP and 52-Lead Plastic Leaded Chip Carrier**



WORD-SLICE® MICROCODED SYSTEM WITH ADSP-1401

3

GENERAL DESCRIPTION

The ADSP-1401 is a high-speed microprogram controller optimized for the demanding sequencing tasks found in digital signal processors and general purpose computers. In addition to high speed (25ns clock-to-address delay) and large addressing range (64K of program memory), this Word-Slice® component has unique features that make it highly versatile:

- on-chip storage and control of ten prioritized and maskable interrupts
- four decrementing event counters
- absolute, relative and indirect addressing capability
- download capability (writeable control store) and
- a dynamically configurable 64-word RAM.

The ADSP-1401 microprogram sequencer's main task is to provide the appropriate microprogram addressing to support programming requirements (e.g., looping, jumping, branching, subroutines, condition testing and interrupts). An internal Look-Ahead pipeline, controlled by both phases of the clock, allows the ADSP-1401 to satisfy these requirements at very high speed.

During each micro-instruction, the ADSP-1401 monitors the conditions and instructions to determine the next microprogram address. This address can come from one of several sources: the stack, the jump address space in the RAM, the data port, the interrupt vectors, or the microprogram counter. An extensive set of conditional instructions are also available, including jumps, branches, subroutines, interrupts, and writeable control store.

The ADSP-1401's internal 64-word RAM is user-configurable into three regions; subroutine stack, register stack and indirect jump address space. The subroutine stack is used for linking interrupts and subroutines and, during their execution, allow storage of system states. The register stack allows association of unique jump addresses with various levels of interrupts and subroutines (both local and global stacks are provided). Indirect jump capability is also supported, addressing for which is provided at the data port.

Interrupts are handled entirely on chip. The ADSP-1401's internal interrupt control logic includes registers for eight external (user) interrupt vectors, a mask register, and a priority decoder. Two additional vectors are reserved for internally-generated interrupts resulting from counter underflow and stack limit violation. A stack limit violation is caused by stack overflow, underflow or collision. A mechanism is provided for recovering from stack violations.

The ADSP-1401's four decrementing 16-bit counters are used to track loops and events. These counters generate a signal when negative. This negative condition is used by several conditional instructions and can also trigger an internal interrupt.

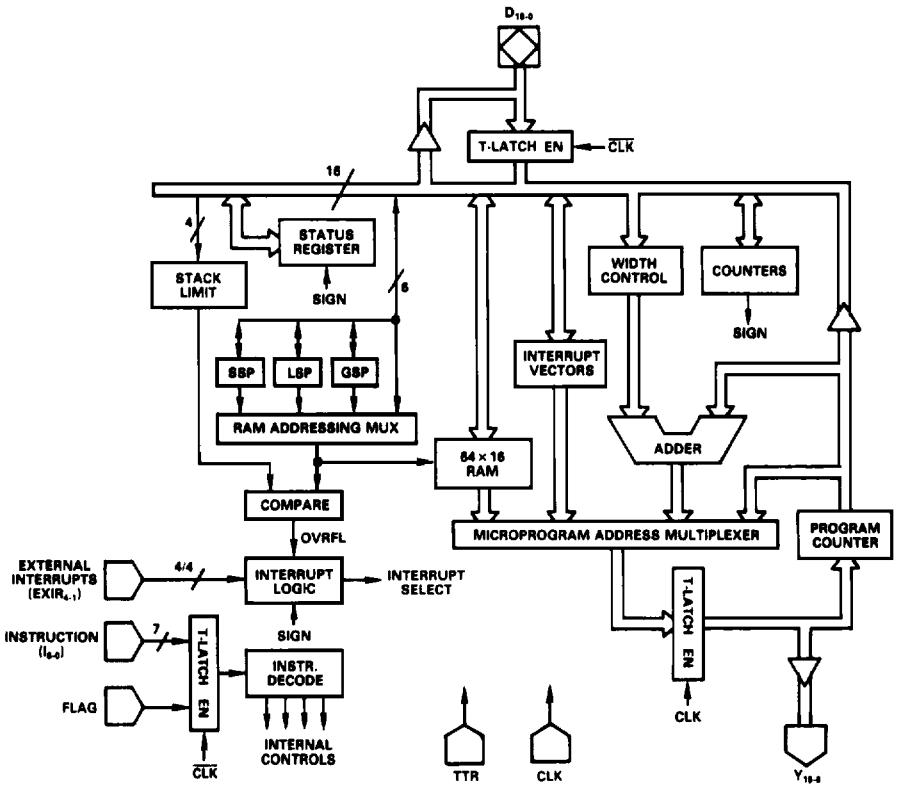


Figure 1. ADSP-1401 Block Diagram

ADDRESSING MODES

Direct: both absolute and relative
Indirect: from internal RAM

HARDWARE FEATURES

Instruction Port
Bidirectional Data Port
Four Input Address Multiplexer
Three Stack Pointers
Four Event Counters
Condition Flag
Eight Prioritized and Maskable User Interrupts
TTR Pin:
- Trap
- Three-State
- Reset

INSTRUCTION TYPES

Jumps and Branches
Stack Operations
Status Register Operations
Counter Operations
Interrupt Control
Relative Address Width Controls
Instruction Hold Control
Writeable Control Store
Dedicated Counter Underflow Interrupt
Dedicated Stack Overflow Interrupt

ADSP-1401 PIN ASSIGNMENTS

Pin Name	Description
I_6-I_0	The 7-bit microinstruction controlling the ADSP-1401.
$Y_{15}-Y_0$	Output bus which provides addresses to the microprogram memory.
$D_{15}-D_0$	Bidirectional Data bus for transferring data to or from the ADSP-1401.
$EXIR_{4-1}$	Four external interrupt request lines. Note that internal circuitry supports 8 interrupts with the aid of an external 2 to 1 multiplexer.
CLK	External clock input
FLAG	An input used for conditional instructions. Its source is usually a condition multiplexer.
TTR	A multi-purpose pin accommodating traps, output disable and reset.
V_{DD}	+5 Volt supply.
GND	Ground.

1.0 ARCHITECTURE

1.1 Look-Ahead Pipeline

Logically, the Look-Ahead pipeline is split into two halves: the first, located at the instruction and data ports; and the second, located at the address port. Each half of the pipeline (input vs. output) has a transparent latch which operates out of phase with the other; the address latch is transparent during the first half of the cycle (clock HI), while the input latches (instruction and data) are transparent during the second half of the cycle (clock LO). This complementary arrangement allows new instructions to be decoded (in preparation for the following cycle) while the program address for the current cycle is held steady.

1.2 Instruction Port

The instruction port receives 7-bit instructions defining the next operation to perform from microcode. The ADSP-1401 has a built-in Look-Ahead pipeline latch, eliminating the need for an external microcode latch to hold instructions. This implementation has the further benefit of allowing instruction "look-ahead"; the sequencer is able to decode the next instruction during execution of the current cycle. During the "look-ahead" period, the sequencer precalculates the next address, allowing its output as early as possible in the next cycle.

External instructions are internally latched during clock HI, and passed directly to the instruction decoder during clock LO (transparent phase); thus, implementing the first half of the Look-Ahead pipeline latch.

The use of the instruction hold mode (see: Instruction Set Description, 2.7; and Instruction Hold Control, appendix 4.1) allows an instruction to be held in the instruction latch for execution over several cycles (freeing microcode for use by other devices).

1.3 Address Port and Multiplexer Sources

The address port provides 16-bit program addresses with three-state drivers designed for driving large microcode memories. Addresses come from a four-to-one microprogram address multiplexer. Between the multiplexer and output port is a transparent latch which is transparent during clock HI and latched during clock LO, permitting addresses to be output as early as possible during phase one (clock HI) while holding the address constant during phase two (clock LO) – implementing the second half of the Look-Ahead pipeline latch.

Inputs to the microprogram address multiplexer are the:

- 16-Bit Program Counter
- 16-Bit Adder
- Interrupt Vector File and
- Internal 64-Word RAM.

Addressing Modes

The ADSP-1401 supports two addressing modes: direct and indirect. The direct addressing mode uses the internal adder to generate either absolute addresses from the data port (without modification) or relative addresses from the program counter (with or without extension: see Status Register, 1.4.4). The indirect addressing mode uses the lower order bits at the data port to access the contents of internal RAM for output.

Output Drivers

The address port output drivers are always active unless placed in the high-impedance state by the IDLE instruction or appropriately asserting the TTR pin (see TTR Pin, 1.7). This allows other devices to supply microcode addresses, which is particularly useful in multi-tasking or context switching applications where several ADSP-1401s may be sharing common microcode memory.

1.3.1 Program Counter

The program counter (PC) consists of a 16-bit incrementing counter. For most instructions, the PC is incremented by the end of the cycle (post-increment) as follows:

$$PC <= \text{output address} + 1.$$

1.3.2 Adder and Width Control

For absolute jumps, data from the data port is passed unchanged through the adder directly to the microprogram address port. For relative jumps, a twos complement offset is supplied from the data port and added with the 16-bit PC. Since the PC normally points to the next instruction, the jump distance is (offset + 1) from the jump instruction. See Status Register (1.4.4) for more details.

The width control block permits microcode width to be reduced in systems not requiring full, 16-bit jump distances. Internal width control logic sign-extends reduced offsets of 8- and 12-bits to full 16-bit precision, accommodating jumps in either direction (positive or negative displacement).

1.3.3 Interrupt Vector File

Ten prioritized interrupt vectors may be stored in the interrupt vector file. The associated interrupts are internally latched and may be individually masked or entirely disabled by the "Disable Interrupts" (DISIR) instruction. The highest priority interrupt vector displaces the usual address on the next cycle following its detection. See Interrupts (1.4.3) for more details.

1.3.4 Internal RAM

Any of the 64 words of RAM may be output on the address port. Four distinct address sources may access the RAM:

- Local Stack Pointer
- Global Stack Pointer
- Subroutine Stack Pointer and
- Lower Order Data Port Bits.

The use of internal RAM and its various address sources are described in section 1.4.2.

1.4 Bidirectional Data Port

The 16-bit bidirectional data port (D_{15-0}) supplies direct or indirect jump addresses and permits loading or dumping of all internal registers. The input data latch freezes incoming data (for counter or register writes executed during that cycle) during the first half-cycle (clock HI) and is transparent for the remainder of the cycle. The output data driver asserts output data *only* during the first half-cycle of a data output instruction and is independent of the address port drivers. This complementary I/O arrangement permits data to be output from the sequencer (as in a read register instruction) during the first half-cycle while accommodating external data setups (for the next cycle) during the second half-cycle.

Direct addressing via the data port may be either relative or absolute. For indirect addressing, the six LS data bits (D_{5-0}) are used to address internal RAM, containing the desired jump address (see Internal RAM, 1.4.2).

1.4.1 Counters

Four independent 16-bit counters are provided for maintaining loops and event tracking. These counters hold twos complement values that may be decremented or preloaded through dedicated instructions. The sign bit associated with the most recently used counter, prior to its decrement, is always saved in the status register (SR_1). Simultaneously, the sign bit is also made available to control various conditional instructions or for asserting the lowest priority interrupt, IR_0, reserved for counter underflow (see: Instruction Set Description, 2.0; and Interrupts, 1.4.3).

Note that interrupt IR_0 is primarily used for ending writeable control store downloads (see Instruction Set Description – WCS, 2.7). Use of IR_0 in the context of a "Decrement Counter and Interrupt on Underflow" operation represents the worst case instruction and flag setup times because of the additional overhead in processing the interrupt after determining whether the counter was underflowed. These setup times are specified two ways:

1. all conditions and
2. IR_0 masked.

The source of SIGN (applied to the condition test) depends upon the type of instruction used (see Instruction Set Description, 2.1). Two possibilities exist:

1. If an *explicit* counter is selected, then the sign applied is that of the counter, prior to the decrement.
2. If *no counter* is selected, then the sign applied is implicitly that of the *status register*, SR_1.

1.4.2 Internal RAM

The ADSP-1401's internal 64-word RAM implements two distinct stacks: a Subroutine Stack (SS) and a Register Stack (RS). The subroutine stack has a dedicated, Subroutine Stack Pointer (SSP), while the register stack shares two pointers: the Local Stack Pointer (LSP) and the Global Stack Pointer (GSP). The three stack pointers are each held in 6-bit, preloadable, up/down counters.

Upon reset, (TTR pin held HI for three cycles, see TTR Pin, 1.7) the SSP is initialized to 0 (top of RAM). The RS pointers (LSP and GSP) are typically configured as shown in Figure 2 using the "Write RSP" instruction (WRRSP). The SSP pushes down while the RS pointers push up. Selection of the active RS pointer (LSP or GSP) is made in the status register.

Stack overflow detection is provided via a stack limit register to protect software integrity and allow stack expansion (see Instruction Set Description – SLRIVP, 2.5).

Each RS pointer may be explicitly initialized by performing the "Write RS Pointer" (WRRSP) instruction. The LSP should be located above the GSP, allowing the local stack to grow upwards as the level of nested subroutines increases. Finally, indirect jump address space (as needed) should be reserved below the global stack.

The sequencer will generate a stack underflow interrupt whenever RAM location zero is popped. This facility may be used in support of stack paging. IV_9 should be masked if not using stack paging, allowing location zero to be used as the first stack location without interrupting. When using paged stacking, location zero must be reserved as an underflow buffer to avoid a subsequent stack POP (which may otherwise occur, depending upon the next instruction) prior to the interrupt routine saving the stack.

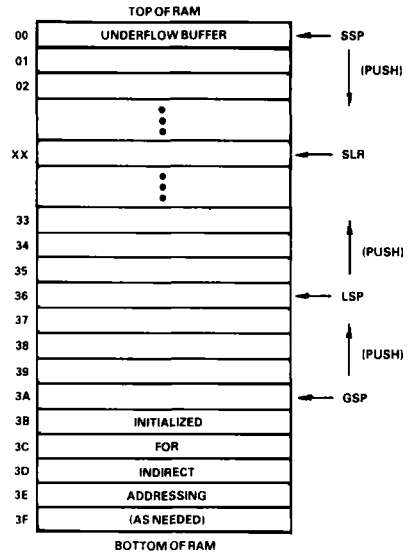


Figure 2. Typical RAM Initialization

Register Stack Pointers (LSP and GSP)

Upon entering a routine, up to four jump addresses may be pushed onto the register stack. A Push onto the register stack first decrements the RS pointer (either LSP or GSP, depending upon the status register) and then writes the appropriate data to RAM. A Pop from the register stack first reads the RAM location and then increments the RS pointer (LSP or GSP).

Four registers are available within context of any routine which are addressed relative to the stack pointer (LSP or GSP) by the two LSBs of the relevant instruction. For example, the instruction:

IF CONDITION, JMP R_2

accesses the location (LSP + 2 or GSP + 2) in RAM as the conditional address source. Prior to exiting a routine, local or global registers can be effectively removed from the RS by the "ADD i TO RSP" (AIRSP) instruction (see Instruction Set Description, 2.2).

Often, the same set of jump addresses are used by several different routines. The GSP is available for addressing these common registers — conserving RAM space and eliminating repeated stack pushes and pops. Global registers can be pushed, popped, and used by conditional instructions in the same way that local registers are handled. In addition, the GSP can itself be pushed and popped to/from the subroutine stack, allowing different routines to access different subsets of the global stack area.

Subroutine Stack Pointer (SSP)

A Push onto the SS (jump subroutine or interrupt) first increments the SSP and then writes the return address to RAM. A pop from the SS first reads the return location and then decrements the SSP, effectively removing the data from the stack (although the data remains in RAM). For interrupts, the return address is the one that would have been output in the cycle when the

interrupt vector was output. For subroutine jumps, the return address is the instruction immediately following the subroutine call. For further information, see: Return from Interrupt with Pending Interrupt, appendix 4.2; and the Instruction Set Description, 2.0.

The subroutine stack can also be used to save key program parameters such as the status register, GSP, or counter values. After entering a new routine, critical parameters from the calling routine are pushed onto the stack, thus freeing the associated hardware for use by the new routine. Prior to the end of the routine, the original parameters are restored with their former values for continued use by the calling routine.

The Stack Usage Example (appendix 4.3) illustrates the state of RAM after three subroutine calls.

Stack Limit Register and Stack Overflow

The preloadable Stack Limit Register (SLR) and associated circuitry warns the user of impending stack overflows, permitting stack overflow recovery. The highest priority interrupt, IR_9, is assigned to stack overflow, although it may be masked. A stack overflow interrupt will occur under any of the following three circumstances:

- a push causing the SSP to increment to the value in the stack limit register
- a pop from SS location 00 (underflow)
- a push causing the RS pointer (LSP or GSP) to decrement to the value in the stack limit register +3.

The three location buffer between the SLR and the RS pointer allows for three extra pushes that may occur (in a worst case) prior to entering the stack overflow service routine. These pushes would be:

1. the push causing the initial overflow
2. a possible push operation while IV_9 is output and
3. the IR_9 return address push.

See: Interrupts, 1.4.3; and Three Stack Pushes on Stack Overflow (appendix 4.2.5) for more details.

The SLR is only 4-bits wide and is compared to the 4 MS bits of the 6-bit RAM address. Therefore, stack limits may only be set at integer multiples of 2^2, i.e., RAM locations 0, 4, 8, 12, . . . , 60. The SLR is right-filled the additional two bits with zeros or ones, depending upon the direction of the push being performed ('00' for SS pushes and '11' for RS pushes, see Instruction Set Description – SLRIVP, 2.5). In the cycle following a stack overflow, the highest priority interrupt vector IRV_9 (also used for trapping; see TTR Pin, 1.7) is output. To determine the cause of this interrupt, both SS and RS pointers must be tested in the first several cycles of the service routine. Prior to returning from the overflow interrupt routine, the SLRIVP instruction must be executed, to clear the calling IR_9 from the interrupt latch.

1.4.3 Interrupts

The ADSP-1401 processes eight external and two internal interrupts. All external interrupts are level sensitive (positive logic: see IR Latch, this section) and are processed by the interrupt logic block. The block elements (see Figure 4) are comprised of an interrupt de-multiplexer followed by an interrupt latch, masking logic and priority decoder for selecting the most urgent interrupt (IR_9 having the highest priority, and IR_0 the lowest), and special one-shot to override the address multiplexer with the interrupt vector (IV_{9-0}) on the cycle following the interrupt request.

The external interrupts (IR_{8-1}) may be used for any purpose, however, unused inputs *must not* be left floating (i.e., tie them to logic LO so as to preclude the associated interrupt). Two additional interrupts which are internal are reserved for stack overflow — IR_9 (see Stack Limit Register and Stack Overflow, 1.4.2) and counter underflow — IR_0 (see Counters, 1.4.1). See Counters (1.4.1) for implications of using IR_0 for other than writable control store downloading.

Interrupt vectors are always output (assuming interrupts are enabled and the associated interrupt is not masked) on the cycle immediately following the acceptance of the interrupt request. Contextual saves (stacking and storing) should be made immediately upon entering the interrupt service routine and restored immediately prior to its exit.

Up to four external interrupts may be connected directly to the external interrupt pins, $EXIR_{4-1}$, and are treated as interrupts IR_{8-5}, respectively. Lower priority interrupts, IR_{4-1}, must be masked out in this case.

Up to eight external interrupts may be accommodated using time-division multiplexing. An external 2:1 multiplexer reduces the eight external interrupts to two groups of four (see Figure 3). An internal de-multiplexer automatically restores the external interrupts back to eight.

The interrupt vector file may be directly read and written via the data bus with the aid of the Interrupt Vector Pointer (see Instruction Set Description, Interrupts, 2.5).

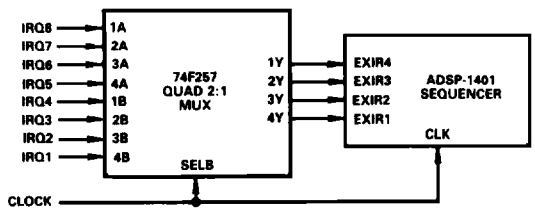


Figure 3. Expanding External Interrupts

IR Latch

Interrupt requests IR_{8-5} are latched during the first half-cycle (clock HI), while IR_{4-1} are latched during the second half-cycle (clock LO). Once latched, external interrupt requests are held until processed, even if the external request signal goes away. This latching technique allows removal of external interrupt sources after they have been recognized by the sequencer.

Latched user interrupt requests (IR_{8-1}) are held until: i) the interrupt is processed and a "Return from Interrupt" (RTNIR) instruction is executed; ii) the interrupt service routine executes a "Clear Current Interrupt" instruction (allowing nested interrupts); or, iii) a "Clear All Interrupts" instruction is executed. Reserved interrupts (IR_9 and IR_0) are cleared from the interrupt latch by utilizing the SLRIVP and CLRS instructions, respectively. See Internal IR Control Logic (1.4.3) for details.

The user may bypass the interrupt latch with the "Select Transparent Interrupts" (STIR) instruction (setting status register bit SR_0). In the transparent mode, the interrupting device must assert the interrupt request until the interrupt service routine resets the request source.

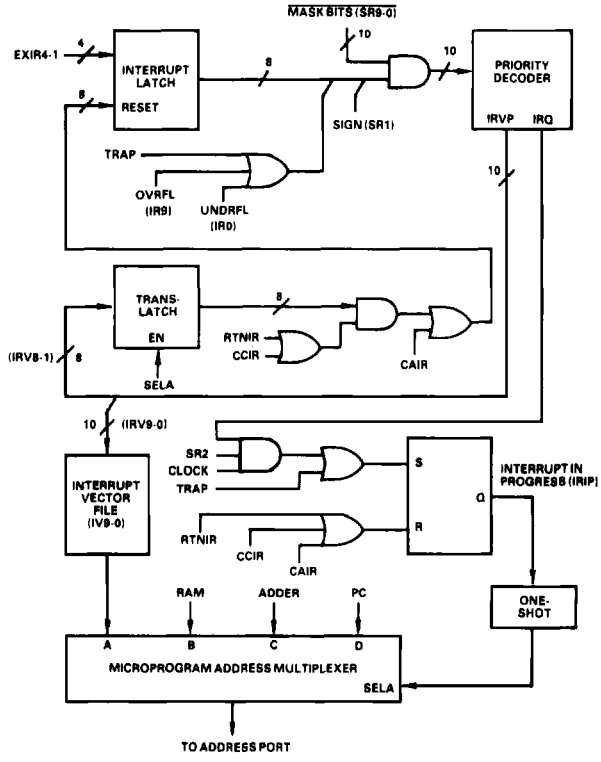


Figure 4. Internal Interrupt Control Logic

IR Mask

All ten interrupts may be independently masked using status register bits SR_{15-6} (corresponding to interrupts IR_{9-0}). Setting a particular mask bit prevents the interrupt from being executed. Note that the status register may be read or written via the Data port, and also pushed and popped to/from the subroutine stack, allowing nesting and servicing of interrupts in any desired order (see: Internal IR Control Logic, 1.4.3; and Status Register, 1.4.4).

Two instructions allow bitwise clearing or setting of the interrupt mask. "IR Mask Bit Clear" (IRMBC) will clear those mask bits for which the corresponding data bits (D_{15-6}, as applied to IR_{9-0}) are set, while "IR Mask Bit Set" (IRMBS) will set those mask bits for which the corresponding data bits are set. In both cases, zeros in the data field will preserve the corresponding mask bit. See Instruction Set Description – Status Register, 2.3.

IR Priority Decoder

Unmasked interrupts are passed to the priority decoder which determines the most urgent, valid interrupt and generates an internal Interrupt Request Signal (IRS). The corresponding vector is then fetched from the interrupt vector file and passed to the address port.

Minimum IR Servicing Requirements

Interrupt vectors are output on the cycle following the acceptance of an interrupt request. Interrupt jumps differ from subroutine jumps in that subroutine jumps push the return address in the *same* cycle as the jump address is output, whereas interrupt return addresses are not pushed until the *following* cycle. This is because the instruction executing while the interrupt vector is output may be utilizing RAM and must complete its execution prior to pushing the interrupt return address. Thus, the PC (interrupt return address) is pushed automatically in the first cycle of the interrupt service routine, i.e., *the cycle following the interrupt request acceptance.*

For this reason, the first instruction of any interrupt service routine is always ignored; it *must* be a no-op (CONT). Note that a minimum interrupt service routine would be a CONT followed by a RTNIR.

Internal IR Control Logic

The interrupt enable bit of the status register, SR_2, *must* be set for interrupt servicing to occur. Interrupt servicing may be inhibited by clearing this bit, although external interrupt requests will continue to be latched.

Only one interrupt is ever active at a time. Additional interrupts are "locked out" by an internal "Interrupt In Progress" signal (IRIP) during interrupt servicing (except for TRAP), although they continue to be latched. The IRIP signal is automatically reset upon the "Return from Interrupt" (RTNIR) instruction which pops the return address from the subroutine stack to the PC.

Normally, multiple interrupts are accumulated in the interrupt latch. Whenever a valid interrupt is pending, the internal signal "Interrupt Request" (IRQ) is asserted. Upon each RTNIR, the highest priority, unmasked, pending interrupt is serviced.

Nested interrupts are supported with two instructions: "Clear Current Interrupt" (CCIR) or "Clear All Interrupts" (CAIR). The CCIR instruction clears the IRIP signal and interrupt latch bit for the interrupt in progress. This action re-enables interrupting, relegating the interrupt in progress to a subroutine status. If an external interrupt is pending, the associated IR vector will be output on the cycle following CCIR. To cancel all pending interrupt requests, the CAIR instruction clears the IRIP signal and the entire interrupt latch.

Normally, it is good practice to convert interrupts to subroutines. This can be done by executing the "Clear Current Interrupt" (CCIR) instruction (resetting IRIP) and should be done as early as possible in the interrupt service routine. There are two reasons for changing the status of an interrupt to that of a subroutine. Firstly, if IRIP is allowed to remain active throughout the interrupt service routine, then the occurrence of either internal interrupt (stack overflow or counter underflow, IR_9 or IR_0, respectively) will remain undetected until the current interrupt concludes; the user will be unaware of these interrupt requests.

When using the TRAP capability (see TTR Pin, 1.7), there is a second reason to clear IRIP. Because TRAP must have the highest priority, interrupt IR_9 (when invoked by a TRAP request) is not locked out by IRIP. This allows TRAP to displace an interrupt in progress, but also means that upon completion of the trap service routine, IRIP will be cleared by the RTNIR instruction; re-enabling interrupting in spite of the incomplete interrupt which TRAP displaced.

Either of these instructions (CCIR or CAIR) require an "extra" cycle before a pending interrupt vector may be output. A typical scenario being an interrupt in progress, IR_n (containing a CCIR instruction), with a interrupt pending, IR_m:

CCIR Example

µCode Location	Instruction Executing	Output Address	Comments
n	IR_n Routine	n+1	IR_m Pending
n+1	CCIR	n+2	Clear IRIP
n+2	IR_n Routine	IV_m	IR_m Recognized
IV_m	IR_m Routine	IV_m+1	. . .

1.4.4 Status Register

The ADSP-1401 has a 16-bit status register for storing various operational modes. The ten MS bits of this register (SR_{15-6}) comprise the interrupt mask for interrupts IR_{9-0}, respectively. The remaining six LS bits (SR_{5-0}) control the operational modes as shown below.

Status Register Bit Assignments

Bit#	Function (HI/LO)
SR_{15}	IR_9 Mask Bit
.	.
.	.
.	.
SR_6	IR_0 Mask Bit
SR_{5-4}	Relative Jump Width Selection: '00' = 16-bit relative address width '01' = 8-bit width '10' = IHC Mode (8-bit width) '11' = 12-bit width
SR_3	Select GSP/LSP
SR_2	Enable/Disable Interrupts
SR_1	Set/Clear Sign Bit
SR_0	Select Transparent/Latched Interrupts

The status register can be directly read and written via the data port and also pushed and popped to/from the subroutine stack. In addition, status register bits SR_{15-6} (the interrupt mask) may be bitwise cleared or set with dedicated instructions. See: Instruction Set Description – Status Register, 2.3; and Interrupts – IR Mask, 1.4.3.

1.5 Clock

The input clock employs both HI and LO levels to control the various transparent latches throughout the device. Generally, the clock should be symmetric; however, in some instances the clock may be stretched during the second half-cycle (LO) to accommodate unusual circumstances such as a cache memory miss (see: TTR Pin – Trap, 1.7).

1.6 External Flag

The external flag input may be used to control conditional instructions. FLAG is latched similarly to instructions (latched during clock HI and transparent during clock LO), but requires less setup time. Two instructions make explicit use of FLAG as their condition (JPCOF and JPCNF), while others employ a condition mode selection (UNCONDITIONAL, NOT FLAG, FLAG, or SIGN; see Instruction Set Description, 2.0) to be specified as part of their opcode.

1.7 TTR Pin (Trap, Three-State and Reset)

The Trap, Three-State and Reset pin (TTR) is a time-multiplexed, three-purpose pin used to

- provide program trap capability
- control the address port output drivers and
- reset the ADSP-1401.

If the TTR pin is held HI for an entire cycle, the RESET sequence begins and TTR must be held HI for at least two more complete cycles (RESET requires three cycles to complete). If trap and three-state control capabilities are also needed, the combination of the 1401's internal circuits and the external circuitry shown in Figure 5 can be used to effectively time-multiplex the TTR pin.

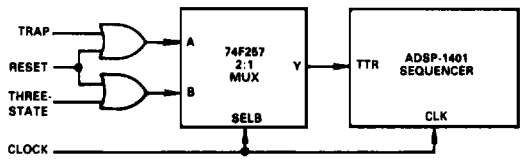


Figure 5. External Logic for TTR Pin

Trap

For a trap to occur, the TTR pin must be asserted during clock LO only. The primary reason to invoke a trap is in support of cache memory systems, or in case of system emergencies. Cache memory systems generally utilize a large microcode memory space, of which only a small area (that currently under execution) is comprised of high-speed RAM (the balance consisting of slower, less costly memory). The high-speed RAM is directly accessible by the sequencer, whereas the bulk of (slow) memory is usually accessible indirectly (via a cache memory controller which controls downloads of code to the cache memory area).

In a cache-based system, microcode is generally executed from the high-speed cache. If an access is attempted to code not resident in the cache area, the cache memory controller must detect the discrepancy and generate an exception to the access (a "cache miss"). Then, the missing code segment must be downloaded to the cache memory area (see: Instruction Set Description – Writeable Control Store, 2.7).

When a cache miss occurs, the cache memory control logic asserts the TTR pin while stretching the system clock LO. Upon detecting the trap request, the sequencer immediately generates the highest priority interrupt, IR_9, replacing the current address (that causing the cache miss). The cache miss address is pushed on the subroutine stack and popped after the interrupt service routine has reloaded the cache area with the missing code segment.

Note: Trap requests which occur on the first cycle of an interrupt service routine are *not* recognized. The ADSP-1401 always executes a CONT instruction in this cycle, and ignores its instruction port and therefore trap requests as well.

The trap interrupt differs from the standard interrupt protocol in three ways:

1. The interrupt vector, IV_9, is output asynchronously, i.e., it occurs t_{TRAD} after asserting the Trap signal and must occur *before* the next cycle! To accomplish this, a clock stretch cycle may be needed to allow enough time to fetch the new instruction.
2. The *current address* is pushed onto the SS for later restoration (after the cache miss is resolved), whereas standard interrupts push the *current address + 1*.
3. Trap interrupts *cannot* be masked or disabled. Note that if IR_9 is also used for stack overflow and underflow, the service routine must discriminate which actually occurred.

Caution: because trapping is asynchronous, spikes on the TTR pin wider than 3ns during clock LO may initiate inadvertent trapping.

Three-State
The address port is placed in a high-impedance state when the TTR pin is HI during clock HI and LO during clock LO. The TTR signal is latched during clock LO and transparent during clock HI. This facilitates full cycle, three-state control. (Note that the IDLE instruction can also place the address port in a high-impedance state.)

Reset
The TTR pin may be used to initialize the ADSP-1401 by asserting it (HI for both clock phases) for at least three full cycles. Use of the reset operation alone does not require the multiplexing described above. However, if the trap and/or three-state controls are also needed, they must not occur in the same cycle (this would be an abnormal situation), as this constitutes a reset. The RESET signal forces a zero output address, places the data port in the high-impedance state, and resets internal registers as follows:

Sequencer Status after RESET Operation

Parameter	Reset Condition
Program Counter	μCode Location 0000_{16}
Subroutine Stack Pointer (SSP)	RAM Location 00_{10}
Local Stack Pointer (LSP)	Undefined
Global Stack Pointer (GSP)	Undefined
Stack Limit Register (SLR)	RAM Location 32_{10}
RAM Data	No Change
Counters	No Change
Interrupt Mask (SR_{15-6})	All Bits to '0' (Unmasked)
Interrupt Vector File	No Change
Interrupt Vector Pointer (IVP)	Undefined
SR_{5-4}	'00' (16-Bit Relative Offsets)
SR_3	'0' (LSP Selected)
SR_2	'0' (Interrupts Disabled)
SR_1	'0' (Sign Bit Cleared)
SR_0	'0' (Latched Interrupt Mode)
Writeable Control Store Mode	Cleared

NOTE:
The first instruction (microcode location 0000_{16}) must be a "CONT".

2.0 INSTRUCTION SET DESCRIPTION

The instruction set is divided into seven categories pertaining to generic operation (see data sheet outline or Mnemonics and Opcodes, 4.5).

Several instructions employ two instruction bits (I_1 and I_0) to specify a counter (C_{3-0}) and/or a local register (R_{3-0}, relative to the RSP) as arguments. Nine of the conditional instructions use another two instruction bits (I_3 and I_2) to select one of the four condition modes:

'00' UNCONDITIONAL
'01' NOT FLAG
'10' FLAG
'11' SIGN

The sign bit of the status register, SR_1, may also be used to (implicitly or explicitly) store an external condition. This is useful if the condition results from an operation performed in the middle of a loop, but is not tested until the end; the loop is exited with an "If Sign: Jump" instruction. Recall that any subsequent counter operations will overwrite SR_1.

2.1 Jump and Branch Instructions

Jump and branch instructions provide flow control of microcode execution, offering three-way branches, jumps, subroutine calls, returns, and addressing mode selection (see Figure 6). These instructions support conditional control, allowing addressing from the register stack, the data port, or the indirect jump address space in the RAM. Generally, they are of the form:

If Condition: Do Operation; Else, Continue.

JPCOF IF FLAG: JUMP PC

The address is not incremented while the flag is at a logic HI, i.e., PC< = PC. If the flag is LO, the next address is (PC + 1).

JPCNF IF NOT FLAG: JUMP PC

The address is not incremented while the flag is at a logic LO, i.e., PC< = PC. If the flag is HI, the next address is (PC + 1).

JTWO IF CONDITION: JUMP PC + 2

If the condition specified is met, this instruction causes the next sequential microprogram address to be skipped. This instruction allows single instruction bypassing or interleaving without need to provide explicit addressing.

JDA IF CONDITION: JUMP DATA, ABSOLUTE

If the specified condition is met, this instruction causes a jump to the absolute address at the data port. If the condition is not met, the next sequential instruction will be executed.

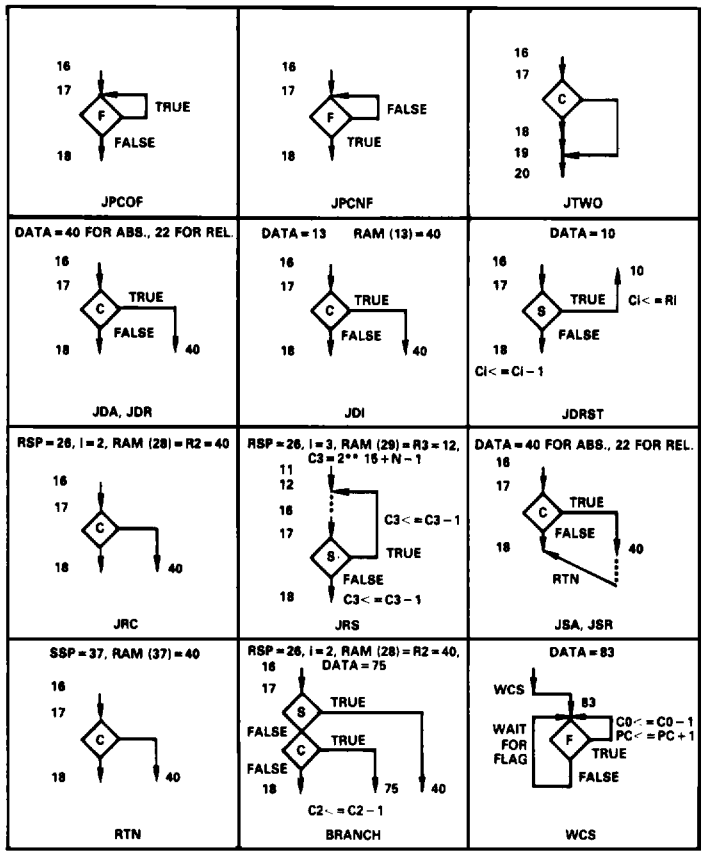


Figure 6. Instruction Flow Charts

JDR IF CONDITION: JUMP DATA, RELATIVE

If the condition specified is met, the address at the data port will be added to the PC and output (jump distance is offset plus one). The offset width is determined by the address width selection (8, 12, or 16-bits). If the condition is not met, the next sequential instruction will be executed.

JDI IF CONDITION: JUMP DATA, INDIRECT

If the condition specified is met, this instruction will output the address stored in the RAM address given by bits D_{5-0} of the data port. If the condition is not met, the next sequential instruction will be executed.

JDRST IF SIGN OF C_i: JUMP DATA, $C_i <= R_i$;
ELSE, $C_i <= C_i - 1$

This instruction first tests the sign of the counter, C_i. If negative, the address at the data port is output and the counter is re-initialized (reset) with the data in the register pointed to by (RSP + i). If the sign is positive, the counter is decremented and the next sequential address is output. The register and counter use the same subscript, i.

JRC IF CONDITION: JUMP R_i. (COND ≠ SIGN)

If the condition specified is met, output the address in RAM at the location (RSP + i), where i is given by I_{1-0} of the instruction. The selected condition may not be SIGN, as this is the JRS instruction. The PC may be pushed on the register stack and referenced as a register thus allowing a "jump to stack" instruction which is useful for looping.

JRS IF SIGN OF C_i: JUMP R_i, $C_i <= C_i - 1$;
ELSE, $C_i <= C_i - 1$

This instruction first tests the sign of counter, C_i. If negative, output the address in RAM at location (RSP + i). If the sign is positive, the next sequential microprogram address is output. The counter is always decremented after the test.

JSA IF CONDITION: JUMP SUBROUTINE, ABSOLUTE

If the condition specified is met, the 16-bit absolute address at the data port is output and the PC will be pushed onto the subroutine stack. If the condition is not met, the next sequential instruction will be executed.

JSR IF CONDITION: JUMP SUBROUTINE, RELATIVE

If the condition specified is met, the address at the data port is added to the PC and output (jump distance is offset plus one) and the PC is pushed onto the subroutine stack. The offset width is determined by the address width selection (8, 12, or 16-bits). If the condition is not met, the next sequential instruction will be executed.

RTN IF CONDITION: RETURN FROM SUBROUTINE

This instruction is used to return from subroutines. If the condition specified is met, the subroutine stack is POPped, which outputs the return address and decrements the SSP. If the condition is not met, the next sequential instruction will be executed.

BRANCH IF SIGN OF C_i: JUMP R_i, $C_i <= C_i - 1$;
ELSE, IF CONDITION:
JUMP DATA, $C_i <= C_i - 1$;
ELSE, $C_i <= C_i - 1$ (COND ≠ SIGN)

This instruction implements a three-way branch with the address source from the data port, register R_i, or the PC. The instruction first tests the sign bit of the counter C_i; if negative, the output address is given by R_i, i.e., RSP + i. If the sign was not true, but the specified condition is true, the address source is the data port. If the sign was not true and the condition is not met, the next sequential instruction is executed.

The counter and the register use the same subscript value i. The counter is *always* decremented. Note that this instruction uses only absolute data addresses; relative addressing is not available with the three-way branch instruction.

2.2 Stack Operations

Subroutine Stack
Subroutine Stack Pointer (SSP) instructions are used for maintaining the subroutine stack. These instructions may also be used to upload or download the entire RAM for examination, stack expansion or context switches.

PSDSS PUSH DATA ONTO SS

Increments the stack pointer and then loads the RAM location specified by the SSP with the data at the data port.

PPSSD POP SS TO DATA PORT

Transfers the contents of the stack location given by the stack pointer to the data port and decrements the stack pointer.

WRSSP WRITE SSP

Loads the SSP with bits D_{5-0} of the data port.

RDSSP READ SSP

Read the 6-bit subroutine stack pointer. This allows the value of the stack pointer to be saved or examined. Bits D_{5-0} of the data port correspond to bits 5-0 of the SSP. The 10 MSB's of the data port (D_{15-6}) are undefined.

DSSP DECREMENT SSP

Decrements the stack pointer without reading.

Register Stack
Register Stack Pointer (RSP) instructions are used to upload and download the entire RAM for initialization, examination, or context switching and to maintain the RAM space allocated to local and global jump registers. As previously discussed, register stack instructions refer to either the Local Stack Pointer (LSP) or the Global Stack Pointer (GSP), depending upon the status register (SR_3). If SR_3 is LO, register stack instructions pertain to the LSP. If SR_3 is HI, register stack instructions pertain to the GSP.

SGSP SELECT GSP

Select the Global Register Stack Pointer. Set Status bit SR_3 (HI).

SLSP SELECT LSP

Select the Local Register Stack Pointer. Clear Status bit SR_3 (LO).

RDRSP READ RSP

Transfers the RSP to the data port bits D_{5-0} for examination or storage. The 10 MSBs (D_{15-6}) of the D port are undefined.

WRRSP WRITE RSP

Preload the selected RSP (LSP or GSP) with bits D_{5-0} of the data port.

PSPC PUSH PC ONTO RS

Decrements the RSP and writes the PC to the register stack. This instruction may be used to set up a JRC loop (IF CONDITION: JUMP R_i = PC).

PSGSP PUSH GSP ONTO SS

Increment the SSP and write the GSP onto the subroutine stack.

PPGSP POP GSP FROM SS

Write the subroutine stack to the GSP and decrement the SSP.

PSDRS PUSH DATA ONTO RS

Decrement the RSP and then write the data at the data port into the location specified by the updated RSP.

PPRSD POP RS TO DATA PORT

Transfers RAM data pointed to by the RSP to the data port and then increments the RSP.

AIRSP ADD i TO RSP

Add i to the register stack pointer. Note that i = 0, 1, 2, or 3 in this instruction corresponds to 4, 1, 2, or 3, respectively. This instruction effectively removes up to four registers from the stack.

S1RSP SUBTRACT ONE FROM RSP

Subtract 1 from the RSP without a write. This instruction is used to modify the RSP without explicitly reloading it.

S4RSP SUBTRACT FOUR FROM RSP

Subtract four from the RSP without a write. This instruction may be used to modify the RSP without explicitly reloading it.

2.3 Status Register Operations

The status register bits, SR_{15-0}, contain ten mask bits, SR_{15-6}, for masking interrupts IR_{9-0}, and six control bits, SR_{5-0} (see

Bidirectional Data Port, 1.4). The entire status register can be read or written via the data port, or pushed or popped to/from the subroutine stack. Upon RESET, the entire status register is initialized to zero.

RDSR READ SR

The entire status register (SR_{15-0}) is output over the data port (D_{15-0}).

WRSR WRITE SR

Write the data port (D_{15-0}) to the status register (SR_{15-0}).

PSSR PUSH SR ONTO SS

Increment the SSP and then write the status register to the subroutine stack.

PPSR POP SR FROM SS

The top of the subroutine stack is written into the status register, and then the SSP is decremented.

2.4 Counter Operations

Counters may be pushed and popped to/from the subroutine stack or loaded directly from the data port. The counters may be read externally by pushing the counters onto the subroutine stack then popping the subroutine stack to the data port. The device has four counters, denoted C_i, which are indexed by the two LSBs of the instruction.

If a jump is required *after* N events (until sign), the counter should be loaded with two less than the number of events desired (N − 2). If a jump is required *for* N events (while sign), the counter is loaded with $2^{15} + N - 2 = 8000_{16} + N - 2$.

Care must be taken when using the counter underflow interrupt (IR_0, see 1.4.3) to clear the sign bit *before* the IR_0 mask bit is cleared.

WRCNTR WRITE C_i

Write to the selected counter, C_i, from the data port.

CLRS CLEAR SIGN BIT

Clear status register bit SR_1.

SETS SET SIGN BIT

Set status register bit SR_1.

PSCNTR PUSH C_i ONTO SS

Increment the SSP and write the specified counter onto the subroutine stack.

PPCNTR POP C_i FROM SS

Transfer the data from the subroutine stack to the counter specified by the instruction, then decrement the SSP.

DCCNTR DECREMENT C_i

Unconditionally decrement counter C_i.

IFCDEC IF CONDITION: DECREMENT C_0

Decrement counter C_0 on condition. If the sign condition is selected, the sign is taken from the status register bit SR_1, rather than from the counter sign (which normally provides the sign condition).

Normally, if the counter underflow interrupt (IR_0) is enabled, it is activated by the counter sign bit going HI. However, if IFCDEC is used to decrement C_0, the IR_0 interrupt is activated by the SR_1 bit, rather than the sign bit of C_0. Since the SR_1 bit goes HI only after C_0 has underflowed, IFCDEC must be executed once more after the C_0 underflow to generate the IR_0 interrupt. Alternatively, the preloaded value of C_0 may be reduced by one.

2.5 Interrupt Control

Detailed interrupt operation is described in the Interrupts section (1.4.3). Here, specific interrupt operations such as interrupt clearing, IRV read/write, interrupt mask manipulation, etc., are described.

CCIR CLEAR CURRENT INTERRUPT

Allows nesting of user interrupts IR_{8-1} on subsequent instructions by clearing both the interrupt latch bit currently being serviced and the interrupt in progress signal (IRIP), re-enabling interrupts. If an external interrupt is pending, the associated IR vector will not be output until the cycle following CCIR. Internal interrupts (IR_9 and IR_0) are *not* cleared by CCIR and must be explicitly cleared through the SLRIVP and CLRS instructions, respectively.

CAIR CLEAR ALL INTERRUPTS

Clears external interrupt latches IR_{8-1}, and re-enables the interrupt interface (IRIP cleared LO). The next sequential instruction will be executed prior to the jump to a pending interrupt. Internal interrupts (IR_9 and IR_0) are *not* cleared by CAIR and must be explicitly cleared through the SLRIVP and CLRS instructions, respectively.

RTNIR RETURN FROM INTERRUPT

Clears the current interrupt latch for IR_{8-1}, re-enables interrupts (IRIP cleared LO), and pops the return address from the subroutine stack. The next sequential instruction will be executed prior to the jump to a pending interrupt routine. Internal interrupts are *not* cleared and the IR_9 and IR_0 interrupt latches must be cleared explicitly through the SLRIVP and CLRS instructions, respectively.

RDIV READ IRV AND INCREMENT IVP

Outputs the interrupt vector currently pointed to by IVP to the data port and then increments the IVP. Interrupts should be disabled when writing or reading interrupt vectors.

WRIV WRITE IRV AND INCREMENT IVP

Writes the interrupt vector currently pointed to by the IVP from the data port and then increments the IVP. Interrupts should be disabled when writing or reading interrupt vectors.

IRMBC IR MASK BITWISE CLEAR

Allows selected IR mask bits to be cleared. Data port bits D_{15-6} are applied to status register bits SR15-6 (corresponding to mask bits for IR_{9-0}). Those data bits which are HI will clear the mask bit, while those data bits which are LO will leave the mask bit intact. Data port bits D_{5-0} are ignored.

IRMBS IR MASK BITWISE SET

Allows selected IR mask bits to be set. Data port bits D_{15-6} are applied to status register bits SR_{15-6} (corresponding to mask bits for IR_{9-0}). Those data bits which are HI will set the mask bit, while those data bits which are LO will leave the mask bit intact. Data port bits D_{5-0} are ignored.

DISIR DISABLE INTERRUPTS

Disables the execution of all further interrupts by clearing the enable interrupt flag (SR_2). External interrupts continue to be latched.

ENAIR ENABLE INTERRUPTS

Enables execution of interrupts by setting the enable interrupt flag (SR_2).

SLIR SELECT LATCHED INTERRUPTS

Places the interrupt request latches in the latched mode for interrupts IR_{8-1} (SR_0 LO). Interrupts are latched if they are valid at the appropriate clock edge. Interrupts IR_{8-5} are latched at the positive going clock edge while IR_{4-1} are latched at the negative going clock edge.

STIR SELECT TRANSPARENT INTERRUPTS

Places the interrupt request latches in the transparent mode (SR_0 HI) for interrupts IR_{8-1}. The interrupt request is only valid while the external interrupt inputs are high. Interrupts are still processed on the next cycle, so long as they meet the minimum interrupt setup specification. Note that selecting transparent interrupting will clear any pending interrupts stored in the interrupt latch.

SLRIVP WRITE SLR WITH D_{5-2}, AND IVP WITH D_{15-12}

Loads the 4-bit stack limit register (SLR) and the 4-bit interrupt vector pointer (IVP) from the data port. This instruction also clears the stack overflow interrupt request IR_9.

For stack overflow detection, the active 6-bit stack pointer (SSP, LSP or GSP) is compared to a 6-bit word comprised of the 4-bit SLR (MSBs) and the two LSBs determined by the instruction type, as follows:

'00' for subroutine stack push (PSDSS); or,
'11' for register stack push (PSDRS).

For example, if a stack limit of 36_{10} and positioning of the IVP at IRV_7 is desired, the value '0111xxxxxx1001xx' is provided at the data port. Note that the SLR and IVP cannot be read.

The interrupt vector pointer (IVP) addresses the vector file for reading or writing interrupt vectors. To write interrupt vectors IRV_{9-0}, the IVP must first be initialized by SLRIVP. The WRIV instruction (see above) is then used to write the interrupt vector pointed to by the IVP, which is then incremented automatically.

2.6 Relative Address Width Controls

The width control instructions allow reduction of microcode when Jump Data Relative and Jump Subroutine Relative instructions need less than the full, 16-bit range. Use these instructions to sign extend the 8, 12 or 16-bit wide jump data presented at the data port. The jump width may be selected by the explicit instructions or by directly setting the status register bits SR_{5-4} as described below. Any of these three instructions will reset the Instruction Hold Control mode (see Misc. Instructions – IHC, 2.7).

Note that selection of 8-bit width can be made with or without IHC. For all relative jumps, the jump distance is the offset +1.

REL16 SELECT 16-BIT RELATIVE JUMPS

Select the 16-bit relative jump. This adds D_{15-0} at the data port to the PC to obtain the jump address. The status bits SR_{5-4} are set to '00'.

REL12 SELECT 12-BIT RELATIVE JUMPS

Selects the jump data from D_{11-0}. The offset is sign-extended allowing relative jumps in the range +2047 to −2048. The status bits SR_{5-4} are set to '11'.

REL8 SELECT 8-BIT RELATIVE JUMPS

Selects the jump data from D_{7-0}. The offset is sign-extended allowing relative jumps in the range +127 to −128. The status bits SR_{5-4} are set to '01'.

2.7 Miscellaneous Instructions

CONT CONTINUE

Increment and output the next location in microcode memory without any other changes. Allows straight line microcode execution.

IDLE DISABLE OUTPUTS AND JUMP PC

Places the address port into the high-impedance state, inhibiting program counter (PC) increments. Useful in applications where multiple sequencers share a common microcode address bus.

This instruction causes the ADSP-1401 to behave as if the clock had stopped. The IDLE instruction may be latched internally by using IHC, freeing microcode for use by another device.

External interrupt requests must be inhibited during IDLE. If interrupts are not inhibited, the ADSP-1401 will attempt to process an interrupt that goes active. However, it will be unable to output an interrupt vector because the IDLE instruction places the address port in the high-impedance state; more importantly, it will set its IRIP flag, which will inhibit further interrupt processing even after the IDLE state is exited.

Interrupts can be inhibited using the interrupt mask or the DISIR instruction. While inhibited, interrupt requests will still be latched in the interrupt latch.

IHC ENABLE INSTRUCTION HOLD CONTROL

Sets SR_{5-4} to '10' and redefines the function of IR_1 to allow a subsequent instruction to be held for repeated execution, regardless of the instruction port. Use of the IHC mode requires that the mask bit for IR_1 be set. See Instruction Hold Control, appendix 4.1 for more details.

While in the IHC mode, asserting IR_1 HI (prior to the second half-cycle of any instruction) will hold that instruction and disable *all* interrupts (although they continue to be latched) until IR_1 is brought LO again (again, prior to the second half-cycle of any instruction).

It is recommended that IR_1 be dedicated to control of the IHC mode (if needed). However, if it must also be used for subsequent interrupting, then the CAIR instruction should be executed before unmasking IR_1 (to clear the interrupt request resulting from use of IR_1 as the IHC control).

Use of IHC is constrained to 8-bit relative addressing (see Relative Address Width Controls, 2.6) and clearing IHC is accomplished by executing any of the relative address width control instructions (changing status register bits SR_{5-4}).

WCS WRITE CONTROL STORE

Provides sequential addressing during microcode downloads to a RAM based microcode store. The instruction may be interpreted as:

JUMP DATA;
IF FLAG: DECREMENT C_0 AND CONTINUE UNTIL INTERRUPTED.

Upon initiation of the WCS instruction, the sequencer outputs the address found at the data port (that of the first instruction to be downloaded). The external flag is then used to gate subsequent sequential addressing for the download and decrementing of counter C_0. This action continues until an interrupt is detected (from either a C_0 underflow, externally or the chip is RESET). Instructions at the instruction port are *ignored* during WCS, until the interrupt or reset occurs.

The external flag allows synchronization of an external memory with the sequencer. FLAG should be asserted HI as each new μcode word is made available for writing to μcode memory.

Notes on Using a Writeable Control Store:

- If a counter interrupt is desired, counter C_0 must be initialized with *two* less than the length of microcode segment to be downloaded.
- If counter interrupting is to be used to exit the WCS mode, IRV_0 should be unmasked and initialized with the address of the instruction to be executed upon WCS completion (see Interrupts, 1.4.3 for timing).
- Since interrupting is used to exit the WCS mode, the last address downloaded is pushed onto the SS stack as an interrupt return address. However, because it is not actually a return address, the SS should be popped immediately by decrementing the SSP (DSSP) to clear it of this last address.
- Since FLAG is used to gate the download, it should not become active until after the WCS instruction is executed.

See application note "Writeable Control Store using the ADSP-1401."

3.0 SPECIFICATIONS

This section describes the ADSP-1401's performance parameters. The Specifications Table lists the device's relevant electrical and switching characteristics, while Figure 7 presents the corresponding timing diagram.

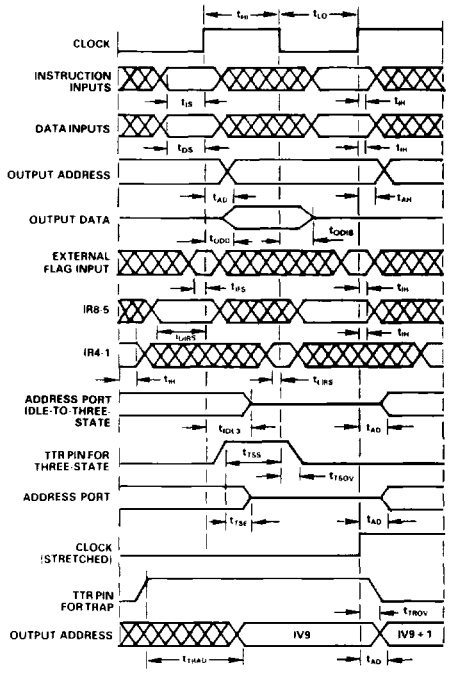


Figure 7. ADSP-1401 Timing Diagram

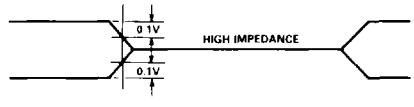


Figure 8. Three-State Reference Levels

ORDERING INFORMATION

Part Number	Temperature Range	Package	Package Outline
ADSP-1401JN	0 to + 70°C	48-Pin Plastic DIP	N-48A
ADSP-1401KN	0 to + 70°C	48-Pin Plastic DIP	N-48A
ADSP-1401JP	0 to + 70°C	52-Lead PLCC	P-52
ADSP-1401KP	0 to + 70°C	52-Lead PLCC	P-52
ADSP-1401JD	0 to + 70°C	48-Pin Ceramic DIP	D-48A
ADSP-1401KD	0 to + 70°C	48-Pin Ceramic DIP	D-48A
ADSP-1401SD	− 55°C to + 125°C	48-Pin Ceramic DIP	D-48A
ADSP-1401TD	− 55°C to + 125°C	48-Pin Ceramic DIP	D-48A
ADSP-1401SD/883B	− 55°C to + 125°C	48-Pin Ceramic DIP	D-48A
ADSP-1401TD/883B	− 55°C to + 125°C	48-Pin Ceramic DIP	D-48A

SPECIFICATIONS[1]

RECOMMENDED OPERATING CONDITIONS

Parameter		J & K Grades Min	J & K Grades Max	S & T Grades[2] Min	S & T Grades[2] Max	Unit
V_{DD}	Supply Voltage	4.75	5.25	4.5	5.5	V
T_{AMB}	Ambient Operating Temp.	0	70	−55	125	°C

ELECTRICAL CHARACTERISTICS

Parameter		Test Conditions	J & K Grades Min	J & K Grades Max	S & T Grades[2] Min	S & T Grades[2] Max	Unit
V_{IH}	Hi-Level Input Voltage	V_{DD} = max	2.0		2.0		V
V_{IHC}	Clock Input Hi-Level Input Voltage	V_{DD} = max	3.0		3.5		V
V_{IL}	Lo-Level Input Voltage	V_{DD} = min		0.8		0.8	V
V_{OH}	Hi-Level Output Voltage	V_{DD} = min, I_{OH} = −1mA	2.4		2.4		V
V_{OL}	Lo-Level Output Voltage	V_{DD} = min, I_{OL} = 3mA		0.6		0.6	V
I_{IH}	Hi-Level Input Current, All Inputs	V_{DD} = max, V_{IN} = 5V		10		10	μA
I_{IL}	Lo-Level Input Current, All Inputs	V_{DD} = max, V_{IN} = 0V		10		10	μA
I_{OZH}	Three-State Leakage Current	V_{DD} = max, V_{IN} = max		50		50	μA
I_{OZL}	Three-State Leakage Current	V_{DD} = max, V_{IN} = 0		50		50	μA
I_{DD}	Supply Current	max clock rate, TTL inputs		90		115	mA
I_{DD}	Quiescent Supply Current	V_{IN} = 2.4V		50		65	mA

ABSOLUTE MAXIMUM RATINGS

Supply Voltage . . . . . −0.3V to 7V
Input Voltage . . . . . −0.3V to V_{DD}
Output Voltage Swing . . . . . −0.3V to V_{DD}
Operating Temperature Range (Ambient) . . −55°C to +125°C
Storage Temperature Range . . . . . −65°C to +150°C
Lead Temperature (10 Seconds) . . . . . 300°C

ESD SENSITIVITY

The ADSP-1401 features proprietary input protection circuitry. Per Method 3015 of MIL-STD-883, the ADSP-1401 has been classified as a Class 1 device.

Proper ESD precautions are strongly recommended to avoid functional damage or performance degradation. Charges as high as 4000 volts readily accumulate on the human body and test equipment and discharge without detection. Unused devices must be stored in conductive foam or shunts, and the foam should be discharged to the destination socket before devices are removed. For further information on ESD precautions, refer to Analog Devices' *ESD Prevention Manual*.

SWITCHING CHARACTERISTICS[3]

Parameter		J Grade Min	J Grade Max	K Grade Min	K Grade Max	S Grade[2] Min	S Grade[2] Max	T Grade[2] Min	T Grade[2] Max	Unit
t_{HI}	Clock HI	50		40		60		50		ns
t_{LO}	Clock LO	40		30		50		40		ns
t_{IS}	Instruction Setup Time	36		30		45		40		ns
t_{DS}	Data Setup Time	10		*		15		15		ns
t_{IH}	Input Signal Hold Time	3		*		*		*		ns
t_{AD}	Address Delay[4] (C = 50pF)		35		25		45		35	ns
t_{AH}	Address Hold Time	3		*		1		1		ns
t_{ODD}	Output Data Delay (C = 30pF)		50		35		60		45	ns
t_{ODIS}	Output Data Disable Time		20		15		25		20	ns
t_{IFSM}	Input Flag Setup Time (IR0 masked)	15		10		20		15		ns
t_{IFSU}	Input Flag Setup Time (no constraints)	30		26		35		30		ns
t_{UIRS}	Upper Interrupts (IR_{8-5}) Setup Time	30		25		35		30		ns
t_{LIRS}	Lower Interrupts (IR_{4-1}) Setup Time	20		15		25		20		ns
t_{TSS}	Three-State (TTR) Setup Time	10		*		15		15		ns
t_{TSOV}	Three-State (TTR) Overlap Time (With Trap)		13		13		5		5	ns
t_{TSE}	Three-State (TTR) Disable Delay		20		15		25		20	ns
t_{IDL3}	IDLE-to-Three-State Disable Delay		20		15		25		20	ns
t_{TROV}	Trap (TTR) Overlap Time (With Three-State)		10		8		10		10	ns
t_{TRAD}	Trap (TTR) to Address Delay		60		45		70		55	ns

NOTES
*Specifications same as J grade.
[1]All specifications are over the recommended operating conditions.
[2]S and T grade parts are available processed and tested in accordance with MIL-STD-883B. The processing and test methods used for S/883B and T/883B versions of the ADSP-1401 can be found in Analog Devices' Military Databook.
[3]Input levels are GND and 3.0V. Rise times are 5ns. Input timing reference levels and output reference levels are 1.5V except for three-state reference levels, which are shown in Figure 8. For capacitive loads greater than 100pF, we recommend the use of external buffers.
[4]Address delays may be derated from the specified 50pF test loading shown in Figure 12 by adding 7ns/50pF for increased capacitive loading.
Specifications subject to change without notice.

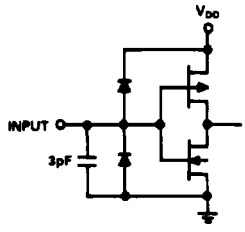


Figure 9. Equivalent Input Circuit

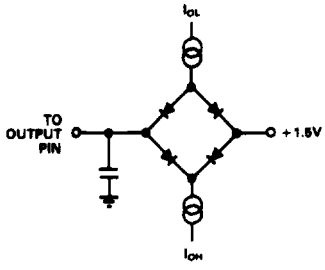


Figure 10. Normal Load for ac Measurements

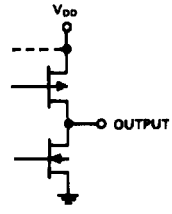


Figure 11. Equivalent Output Circuit

4.0 APPENDICES

4.1 Instruction Hold Control (IHC)

The IHC function allows external microcode width to be reduced by allowing the 1401's microcode field to be shared with another device. This sharing is accomplished by temporarily latching an instruction that is used repetitively within the ADSP-1401 and re-directing its microcode to a different device. Control of the latching is accomplished by the IHC instruction, which re-assigns the function of interrupt signal IR_1, becoming the latch/unlatch control line.

To use this mode, execute the IHC instruction, which sets status register bits SR_{5-4} to '10'. Interrupt line IR_1 now controls the instruction hold mode (not interrupt), so IR_1 must be masked. The shared signal, IR_5 (recall, IR_{8-5} and IR_{4-1} share the same pins), is still used normally, since it is active during clock low.

To initiate an instruction hold, execute the instruction to be repeated, while asserting IR_1 (HI) prior to the clock falling edge of the same cycle. For so long as IR_1 is kept high (on the falling edge of the clock), the instruction will repeat. All interrupts are automatically disabled while the instruction is held.

When IR_1 is needed for interrupts (instead of controlling the instruction hold mode) the IHC mode may be disabled by: executing one of the relative jump width control instructions; or, by changing status register bits SR_{5-4} directly. Prior to unmasking IR_1, execute the CAIR (clear all interrupts) instruction to clear the interrupt latch.

4.2 Programming Examples

The following examples are given to illustrate some fine points of programming the ADSP-1401.

4.2.1 Jump Register (See Figure 13a)

In this example, three jump registers (R_{3-1}) are loaded with external data and one (R_0) is loaded with the program counter, enabling a jump to the top-of-stack.

Current Address	Instruction Executed	Output Address	Comments	RSP
20	PSDRS	21	Push R_3	57
21	PSDRS	22	Push R_2	56
22	PSDRS	23	Push R_1	55
23	PSPC	24	Push PC (R_0 = 24)	54
24	Start of Loop . . .	25		
. . .	. . .	. . .		
30	. . .	. . .		
31	JRC (R_0)	32/24	Cond. Jump to [R_0] = 24	
32/24	. . .			

4.2.2 Return from Interrupt with Pending Interrupt (See Figure 13b)

This example shows the program flow when two interrupts occur in the same cycle or an interrupt is latched while another interrupt is being executed. The "Return from Interrupt" instruction (RTNIR) will execute one instruction of the mainline routine before servicing a pending interrupt since interrupts are not re-enabled until the end of the cycle. Here, IV_7 = 60 and IV_3 = 21.

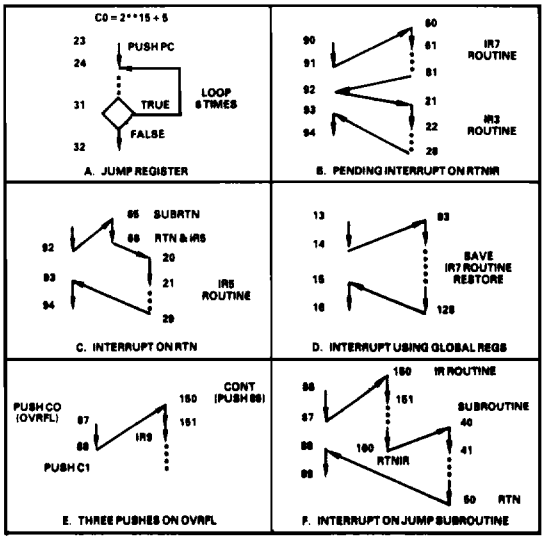


Figure 13. Programming Examples

Current Address	Instruction Executed	Output Address	Comments
89	. . .	90	. . .
90	. . .	91	Interrupts I_7 & I_3 valid.
91	. . .	60	IV_7 output. Instruction 91 still executed.
60	CONT	61	92 is pushed on stack.
61	. . .	62	. . .
. . .	. . .	. . .	
81	RTNIR	92	92 popped and interrupts re-enabled.
92	. . .	21	IV_3 output. Instruction 92 still executed.
21	CONT	22	93 pushed on stack.
22	. . .	23	. . .
. . .	. . .	. . .	. . .
28	RTNIR	93	93 is popped from stack.
93	. . .		

4.2.3 Interrupt on a Return from Subroutine (See Figure 13c)

If an interrupt occurs on a subroutine return, no instructions in the main program are executed prior to servicing the interrupt routine. Here, IV_5 = 20.

Current Address	Instruction Executed	Output Address	Comments
91	. . .	92	. . .
92	JSR	65	Jump to 65. 93 pushed.
65	. . .	66	IR_5 becomes valid.
66	RTN	20	IV_5 address output. 93 popped.
20	CONT	21	93 pushed.
. . .	. . .	. . .	. . .
29	RTNIR	93	93 popped.
93	. . .		

4.2.4 Interrupt Routine using Global Registers (See Figure 13d)

Current Address	Instruction Executed	Output Address	Comments
12	. . .	13	Mainline . . .
13	. . .	14	IR_7 occurs here.
14	CONT	93	Output IV_7.
93	PSSR	94	Push status register.
94	PSCNTR (C_3)	95	Save previous values . . .
95	PSCNTR (C_1)	96	. . .
96	PSGSP	97	. . .
97	WRSR	98	Write new values . . .
98	WRCNTR (C_3)	99	. . .
99	WRCNTR (C_1)	100	. . .
100	WRRSP	101	. . .
101	. . .	102	Begin interrupt servicing . .
. . .	. . .	. . .	. . .
123	. . .	124	End of interrupt service routine.
124	PPGSP	125	Pop in reverse order of pushes . . .
125	PPCNTR (C_1)	126	. . .
126	PPCNTR (C_3)	127	. . .
127	PPSR	128	. . .
128	RTNIR	15	Jump back to mainline.
15	. . .	16	. . .

4.2.5 Three Stack Pushes on Stack Overflow (See Figure 13e)
The four register buffer between the subroutine stack and the register stack will be filled with three values whenever the stack push that caused the overflow is followed by another instruction that causes a stack push. The second stack push occurs since the instruction that is interrupted (the second stack push) must complete internally to preserve the correct state of the ADSP-1401 after the interrupt. The third push occurs to provide the return address to the main program. The sequence is illustrated below. Assume that the address of the stack overflow service routine (IV_9) is at 150.

Current Address	Instruction Executed	Output Address	Comments
86	. . .	87	
87	PSCNTR (C_0)	88	The push causes a stack overflow.
88	PSCNTR (C_1)	150	The interrupted instruction executes.
150	CONT	151	89 is pushed onto the stack.
151	. . .		

4.2.6 Interrupt on Jump Subroutine Instruction (See Figure 13f)

Current Address	Instruction Executed	Output Address	Comments
86	. . .	87	Interrupt occurs to location 150
87	JSA (40)	150	
150	CONT	151	40 Pushed on stack
. . .	. . .	. . .	. . .
. . .	. . .	160	. . .
161	RTNIR	40	Return from interrupt
40	. . .	41	

4.3 Use of RAM by Multiple Subroutines
This diagram (Figure 14) shows the state of RAM after three nested subroutine calls.

Prior to the first subroutine call, the RSP was used to preload the bottom portion of the RAM with indirect jump addresses. Next, global jump registers were preloaded. In the mainline program, only global jump registers are used.

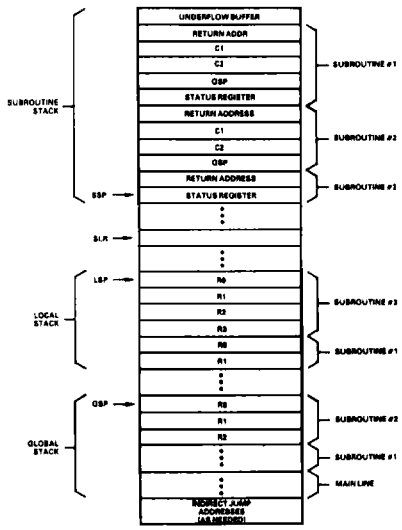


Figure 14. RAM Status after Subroutine Calls

The instruction calling the first subroutine pushes the return address of the main program onto the subroutine stack. The values of counters 1 and 3 are also pushed onto the stack to free counters 1 and 3 for use in subroutine #1. The GSP is saved since different routines will require different GSPs. Similarly, the status register of the main program is saved. As shown, routine #1 uses both global and local jump registers. It selects the GSP or LSP at the appropriate times in the routine by executing SGSP or SLSP instructions.

Routine #2 saves the return address, some counters, and the GSP for routine #1. Since no local registers are used in routine #2, none are loaded.

Routine #3 saves the return address and the status register. Since the GSP and counters are not used in this routine, they are not saved. After the new status register is loaded (selecting the LSP), local registers are pushed onto the stack.

4.4 Bus Drive Considerations with the Word-Slice Family
The various members of Analog Devices' Word-Slice family are designed with high-speed drivers on all output pins. This capability means that large peak currents may pass through the ground and V_{DD} pins when all the bus lines are simultaneously charging their load capacitance from LO to HI, or *vice versa*.

To calculate the peak current for a typical family member (such as the ADSP-1401 Program Sequencer), we assume that all output drivers are switching from a HI to a LO state. From a

fall time and capacitance measurement, we can determine that the peak current in each driver is:

$$I_{peak} = C_{load} \cdot \Delta V/\Delta t,$$

where $\Delta V/\Delta t$ is the initial slew rate.

In the case of the program sequencer, for an external load capacitance of 50pF and a measured slew rate of 0.6V/ns, the peak current will be about 30mA. Since there are 16 such drivers, the total peak current may approach 480mA!

The internal ground and supply lines may undergo a large disturbance during this transition unless the ADSP-1401 is tied to a solid ground plane and good high frequency decoupling is used (0.1μF ceramic between GND and V_{DD} as close as possible to the device). Otherwise, is it possible that internal data in the ADSP-1401 may be lost.

4.5 Mnemonics and Opcodes

Opcode bits "ii" select the relevant register (R_{3-0}) and/or counter (C_{3-0}). Opcode bits "cc" select the condition to be applied:

'00' UNCONDITIONAL
'01' NOT FLAG
'10' FLAG
'11' SIGN

The SIGN condition is precluded from instructions prefixed with "*".

Mnemonic	Opcode (I_{8-0})	Description
Jump and Branch Instructions:		
JPCOF	001 0101	IF FLAG: JUMP PC (*self*)
JPCNF	011 0101	IF NOT FLAG: JUMP PC (*self*)
JTWO	101 cc01	IF COND: JUMP PC+2 (*skip*)
JDA	111 cc11	IF COND: JUMP DATA, ABSOLUTE
JDR	111 cc01	IF COND: JUMP DATA, RELATIVE
JDI	101 cc10	IF COND: JUMP DATA, INDIRECT
JDRST	100 11ii	IF SIGN OF C_i: JUMP DATA, $C_i <= R_i$; ELSE, $C_i <= C_i - 1$
*JRC	110 cc ii	IF COND: JUMP R_i
JRS	110 11ii	IF SIGN OF C_i: JUMP R_i, $C_i <= C_i - 1$
JSA	111 cc00	IF COND: JUMP SUB, ABSOLUTE
JSR	111 cc10	IF COND: JUMP SUB, RELATIVE
RTN	101 cc11	IF COND: RETURN FROM SUB
*BRANCH	100 cc ii	IF SIGN OF C_i: JUMP R_i; ELSE, $C_i <= C_i - 1$, IF COND: JUMP DATA
Stack Operations:		
Subroutine Stack		
PSDSS	001 1110	PUSH DATA ONTO SS
PPSSD	011 1110	POP SS TO DATA PORT
WRSSP	000 1110	WRITE SSP
RDSSP	010 1100	READ SSP
DSSP	000 0010	DECREMENT SSP
Register Stack		
SGSP	000 0111	SELECT GSP
SLSP	000 0110	SELECT LSP
RDRSP	010 1111	READ RSP
WRRSP	000 1100	WRITE RSP
PSPC	010 0011	PUSH PC ONTO RS
PSGSP	000 0101	PUSH GSP ONTO SS
PPGSP	000 0100	POP GSP FROM SS
PSDRS	001 1111	PUSH DATA ONTO RS
PPRSD	011 1111	POP RS TO DATA PORT
AIRSP	010 10ii	ADD i TO RSP
S1RSP	000 1111	SUBTRACT 1 FROM RSP
S4RSP	011 1100	SUBTRACT 4 FROM RSP

Status Register Bit Assignments

Bit#	Function (HI/LO)
SR_{15}	IR_9 Mask Bit
•	•
•	•
•	•
SR_6	IR_0 Mask Bit
SR_{5-4}	Relative Jump Width Selection: '00' = 16-bit relative address width '01' = 8-bit width '10' = IHC Mode (8-bit width) '11' = 12-bit width
SR_3	Select GSP/LSP
SR_2	Enable/Disable Interrupts
SR_1	Set/Clear Sign Bit
SR_0	Select Transparent/Latched Interrupts

Status Register Operations:		
RDSR	010 1110	READ SR
WRSR	001 1100	WRITE SR
PSSR	010 0001	PUSH SR ONTO SS
PPSR	010 0010	POP SR FROM SS
Counter Operations:		
WRCNTR	011 10ii	WRITE C_i
CLRS	001 0100	CLEAR SIGN BIT
SETS	011 0100	SET SIGN BIT
PSCNTR	000 10ii	PUSH C_i ONTO SS
PPCNTR	001 10ii	POP C_i FROM SS
DCCNTR	011 00ii	DECREMENT C_i
IFCDEC	101 cc00	IF COND: DECREMENT C_0
Interrupt Control:		
CCIR	001 0001	CLEAR CURRENT INTERRUPT
CAIR	000 0001	CLEAR ALL INTERRUPTS
RTNIR	000 0011	RETURN FROM INTERRUPT
RDIV	010 1101	READ INTERRUPT VECTOR AND INCREMENT IVP
WRIV	000 1101	WRITE INTERRUPT VECTOR AND INCREMENT IVP
IRMBC	001 0011	IR MASK BITWISE CLEAR
IRMBS	001 0010	IR MASK BITWISE SET
DISIR	001 0110	DISABLE INTERRUPTS
ENAIR	011 0110	ENABLE INTERRUPTS
SLIR	001 0111	SELECT LATCHED INTERRUPTS
STIR	011 0111	SELECT TRANSPARENT INTERRUPTS
SLRIVP	001 1101	WRITE SLR<= D_{5-2} AND IVP<= D_{15-12}
Relative Address Width Controls:		
REL16	010 0100	SELECT 16-BIT RELATIVE ADDRESSING
REL12	010 0111	SELECT 12-BIT RELATIVE ADDRESSING
REL8	010 0110	SELECT 8-BIT RELATIVE ADDRESSING
Miscellaneous Instructions:		
CONT	000 0000	CONTINUE
IDLE	001 0000	IDLE
IHC	010 0101	ENABLE INSTRUCTION HOLD CONTROL
WCS	010 0000	WRITE CONTROL STORE

ADSP-1401 PIN CONFIGURATIONS

DIP
D-48A
N-48A

PIN	FUNCTION	PIN	FUNCTION
1	D7	48	D6
2	D8	47	D5
3	D9	46	D4
4	D10	45	D3
5	D11	44	D2
6	D12	43	D1
7	D13	42	D0
8	D14	41	CLK
9	D15	40	FLAG
10	EXIR1	39	I6
11	EXIR2	38	I5
12	GND	37	V_{DD}
13	EXIR3	36	I4
14	EXIR4	35	I3
15	TTR	34	I2
16	Y15	33	I1
17	Y14	32	I0
18	Y13	31	Y0
19	Y12	30	Y1
20	Y11	29	Y2
21	Y10	28	Y3
22	Y9	27	Y4
23	Y8	26	Y5
24	Y7	25	Y6

PLCC
P-52

PIN	FUNCTION	PIN	FUNCTION
1	D7	52	D6
2	D8	51	D6
3	D9	50	D4
4	D10	49	D3
5	D11	48	D2
6	D12	47	D1
7	GND	46	GND
8	D13	45	D0
9	D14	44	CLK
10	D15	43	FLAG
11	EXIR1	42	I6
12	EXIR2	41	I5
13	GND	40	V_{DD}
14	EXIR3	39	I4
15	EXIR4	38	I3
16	TTR	37	I2
17	Y15	36	I1
18	Y14	35	I0
19	Y13	34	Y0
20	GND	33	GND
21	Y12	32	Y1
22	Y11	31	Y2
23	Y10	30	Y3
24	Y9	29	Y4
25	Y8	28	Y5
26	Y7	27	Y6