NATIONAL

GODBOUT
BILL GODBOUT ELECTRONICS
BOX 2355, OAKLAND AIRPORT, CA 94614

IPC-16A/500D-1 MOS/LSI single chip 16-bit microprocessor (PACE)˙

general description

PACE (Processing And Control Element) is a single-chip, 16-bit microprocessor packaged in a standard, hermetically sealed, 40-pin ceramic dual-in-line package.

Silicon gate, P-channel enhancement mode standard process technology ensures high performance, high reliability and high producibility.

PACE is intended for use in applications where the convenience and efficiency of 16-bit word length is desired while maintaining the low cost inherent in single chip, fixed instruction microprocessors. The basic economics in conjunction with the users' ability to programmatically specify 8 or 16-bit data operations provides the following applications advantages:

(continued on page 3)

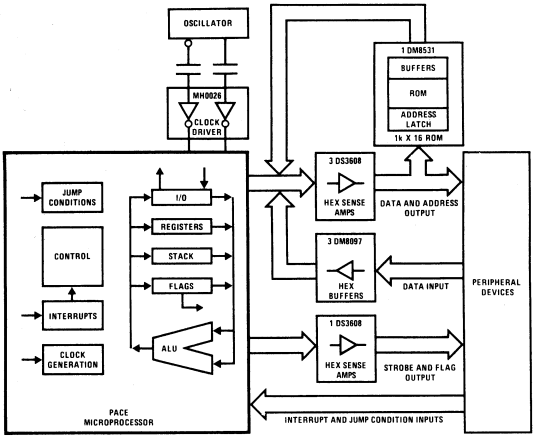
features

■ 16-bit instruction word	Addressing flexibility, speed
■ 8 or 16-bit data word	Wide application
■ Powerful instruction set	Efficient programming
■ Common memory and peripheral addressing	Powerful I/O instructions
■ Shares instructions with National's IMP-16 basic set	Allows software compatibility
■ Four general purpose accumulators	Reduces memory data transfers
■ 10-word stack	Interrupt processing/ data storage
■ Six vectored priority interrupt levels	Simplifies interrupt service and hardware
■ Programmer accessible status register	May be preserved, tested, or modified
■ 2μs microcycle	Fast instruction execution
■ Can utilize DM8531 1k-by-16 ROM	Single memory package
■ Two clock inputs	Minimum external components

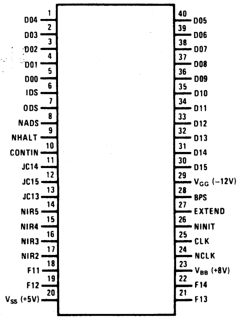
applications

- Test system and instrument control
- Process controllers
- Machine tool control
- Terminal control
- Small business machines
- Traffic controllers
- Word processing systems
- Peripheral device controllers
- Educational systems
- Sophisticated games
- Distributed and multiprocessor systems

block and connection diagrams



Dual-In-Line Package



TOP VIEW

BILL GODBOUT ELECTRONICS
BOX 2355, OAKLAND AIRPORT, CA 94614

PACE SPECIFICATIONS --- IPC-16A/500D-1

ABSOLUTE MAXIMUM RATINGS (NOTE 1)

All Input or Output Voltages with Respect to
Most Positive Supply Voltage (V_{BB}).......... -0.3V to -20V
Operating Temperature Range................ +20°C to +45°C
Storage Temperature Range........ -65°C to +150°C
Lead Temperature (soldering, 10 seconds).. +300°C

ELECTRICAL CHARACTERISTICS (T_A = +20°C to +45°C, V_{SS} = +5.14 ±2%, V_{GG} = -12.3 ±2%, V_{BB} = V_{SS} + 3V)

PARAMETER	CONDITIONS	MIN	TYP	MAX	UNITS
OUTPUT SPECIFICATIONS					
D00-D15, F11-F14, ODS, IDS, NADS (These are open drain outputs which may be used to drive DS3608 sense amps, or may be used with pull down resistors to provide a voltage output).					
Logic "1" Output Current (Note 7)	V_{OUT} = 2.0V	-0.8	-2.0	-10	mA
Logic "0" Output Current	VGG ≤ VOUT ≤ VSS			±10	µA
NHALT, CONTIN					
Logic "1" Output Voltage	IOUT = 200 µa	VSS-1			V
Logic "0" Output Voltage	IOUT = 200 µa			1.0	V
INPUT SPECIFICATIONS					
D00-D15, NIR2-NIR5, EXTEND, JC13-JC15, CONTIN, NINIT, NHALT (These are TTL compatible inputs.) (Note 2)					
Logic "1" Input Voltage		VSS-1		VSS+0.3	V
Logic "0" Input Voltage		VSS-1		VSS-4.5	V
Pullup Transistor "ON" Resistance (D00-D15) (Note 3)	VIN = VSS - 1V		4	8	kOhms
Pullup Transistor "ON" Resistance (except D00-D15)	VIN = VSS - 1V		2	4	kOhms
Logic "0" Input Current (D00-D15)	VIN = .4V		-1.0	-3	mA
Logic "0" Input Current (except D00-D15)	VIN = .4V		-2.0	-5	mA
Input Capacitance	VIN = VSS,		10		pF
CLK, NCLK (These are MOS Clock Inputs.)	f_T = 500 kHz				
Clock "1" Voltage (Note 5)		VSS-1		VSS+0.3	V
Clock "0" Voltage		VGG		VGG+1	V
Input Capacitance (Note 6)		30	80	150	pF
Bias Supply Current	V_{BB} = VSS + 3.0V		30		µA
Average Power Dissipation	t_p=0.5 µs, T_A 25°C		700	1500	mW
TIMING SPECIFICATIONS (SEE ADDITIONAL TIMING INFORMATION FIGURES 7 THROUGH 10)					
CLK, NCLK (Referenced to 10% and 90% Amplitude)					
Rise and Fall Time (t_r, t_f)		10		25	ns
Clock Width (t_{WCLK}, t_{WNCLK})		220			ns
Clock Overlap ($t_{OV\ A}$, $t_{OV\ B}$)				-10	ns
Clock Period (tp)		0.5		.55	µs
EXTEND					
Individual Extend Duration				1.0	µs
Propagation Delay					
F11-F14 (Note 8)	V_{OUT} = 2.0V		100	300	ns
NHALT, CONTIN (Note 9)	C_L = 20 pF		100	200	ns
NADS, IDS, ODS, D00-D15 (Note 8)	V_{OUT} = 2.0V		60	100	ns
D00-D15 Input Setup Time (Note 10)		200	75		ns
NINIT Initialization Pulse Width		8			clock cycles
NIR2-NIR5 Input Pulse Width to Set Latch		1			clock cycles

Note 1. Maximum ratings indicate limits beyond which permanent damage may occur. Continuous operation at these limits is not intended and should be limited to those conditions specified under electrical characteristics.
Note 2. Pullup transistor provided on chip. (See figure 6.)
Note 3. Pullup transistors on JC13, JC14, JC15 are turned on one out of 8 clock intervals. Pullup transistors on D00-D15 are turned on during last clock period of Input Data Strobe (IDS). Other pullup transistors are on continuously when in data input mode.
Note 4. Pin 28 (BPS) is tied to V_{GG}.
Note 5. Clamp diodes and series damping resistors may be required to prevent clock overshoot.
Note 6. Capacitance is not constant and varies with clock voltage and internal state of processor.
Note 7. For $V_{SS} \geq V_{OUT} \geq 2.0$ volts output current is a linear function of V_{OUT}.
Note 8. Delays measured from valid logic level on clock edge initiating change to valid current output level.
Note 9. Delay measured from valid logic level on clock edge initiating change to valid voltage output level.
Note 10. With respect to end of Input Data Strobe (IDS). (See figure 7.)

general description (con't)

PACE is particularly efficient when handling both 8 and 16-bit interfaces within the same microprocessor based system. Requirements for external hardware are minimized without sacrificing coding efficiency.

PACE is extremely cost effective in applications dominated by 8-bit data element interfaces. Coding and address generation efficiencies, as well as operating speeds for double precision operations found only in 16-bit microprocessors are extended to the 8-bit system.

The principal resources featured in PACE to minimize system program and read/write storage while increasing throughput include:

FOUR 16-BIT GENERAL PURPOSE WORKING REGISTERS available to the user reduce the number of memory load and store operations associated with saving temporary and intermediate results in system memory. This results in increased throughput with reduced program and data storage requirements.

AN INDEPENDENT 16-BIT STATUS AND CONTROL FLAG REGISTER automatically and continuously preserves system status. The user may operate on its contents as data, allowing masking, testing and modification of several bit fields simultaneously.

A TEN WORD (16-BIT) LAST-IN, FIRST-OUT (LIFO) STACK automatically preserves return addresses during interrupt servicing and sub-routine execution. The presence of a stack inherently decreases response time to interrupts while eliminating both program and read/write system storage overhead associated with storing stack information outside the microprocessor chip. In some applications the 10-word stack plus on-chip registers can totally eliminate the need for off-chip read/write memory.

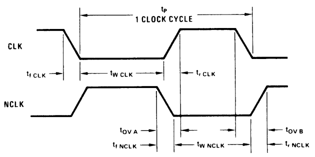
STACK FULL/STACK EMPTY interrupts are provided to facilitate off-chip stack storage in those applications where additional stack capacity is desirable.

A SIX LEVEL, VECTORED PRIORITY INTERRUPT SYSTEM internal to the chip provides automatic interrupt identification, eliminating both program storage overhead and the time normally required to poll peripherals in order to identify the interrupting device. When more than six interrupts are involved, more than one peripheral may be placed on a priority-level by means of a simple open collector connection to the appropriate priority interrupt request line.

FOUR SENSE INPUTS AND FOUR CONTROL FLAG OUTPUTS allow the user to respond directly to specific combinations of status present in the microprocessor based system. This ability to respond directly to system status requires no external hardware and allows appropriate control signal outputs to be generated programmatically, eliminating costly hardware, program overhead and throughput associated with implementing these functions over the system data bus.

Other PACE features which minimize the cost of external support hardware include easily generated clock inputs and I/O cycle extend capability.

The PACE single chip 16-bit microprocessor permits the implementation of a complete microprocessor system with 16,384 bits of read-only program storage and TTL data bus interface in fewer than a dozen standard support packages, as shown in the diagram on the first page.



Note: Clock timing referenced to 10% and 90% amplitude points.

FIGURE 1. Clock Timing

FUNCTIONAL DESCRIPTION

The PACE microprocessor, shown in *Figure 2*, provides 16-bit parallel data processing capability. This word length provides considerable convenience for addressing memory and peripheral devices and provides sufficient accuracy that many applications will not require the use of double precision arithmetic. It also provides increased speed by processing twice as many bits per cycle and reducing time consuming memory accesses. However, for those applications not requiring high accuracy, or for character processing, PACE provides the ability to operate on 8-bit data, while still providing 16-bit instructions and addressing capability.

Data Storage

Seven data registers are provided, four of which are directly available to the programmer (as accumulators AC0 to AC3) for data storage and address formation. AC0 is the principal working register, AC1 is the secondary working register, and AC2 and AC3 are page pointers or auxiliary data registers. The other three registers serve as a program counter and two temporary registers are used by the control section to effect the PACE instruction set.

Additional data storage is provided for up to ten words by a last-in, first-out or push-pull stack. The stack is used primarily for storing the contents of the program counter during subroutine execution and interrupt servicing. The stack may also be used for storing status information or data; in some applications, such as device controllers, the stack plus four accumulators may provide enough storage to eliminate the need for external read-write memory. For applications where the 10-word capacity of the stack is insufficient, external read-write memory may be used as a stack extension. This is facilitated by the provision of stack full and stack empty interrupts, allowing implementation of a simple stack service routine.

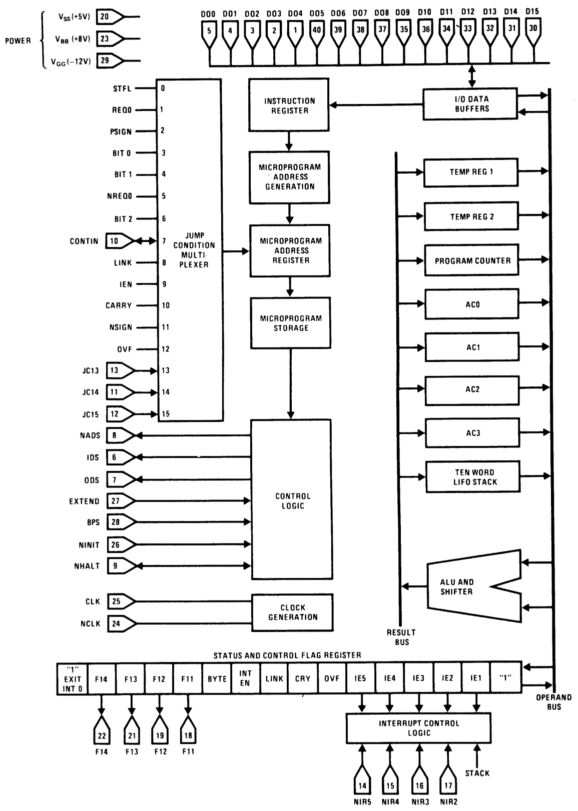


FIGURE 2. PACE Detailed Block Diagram

ALU

The arithmetic and logic unit (ALU) provides the data manipulation capability which is an essential feature of any microprocessor. The operations provided by the ALU include AND, OR, XOR, complement, shift left, shift right, mask byte and sign extend. Both binary and (4-digit per word) binary-coded-decimal (BCD) addition capability are provided, thus eliminating the program storage and execution time required to perform BCD to binary conversion.

A unique feature of the PACE ALU is the ability to operate on either 8 or 16-bit data, as specified by the programmer through the use of a status flag. This feature allows character oriented and other 8-bit applications to be implemented and executed using an 8-bit peripheral data bus and read-write memory, while address formation and instruction storage are implemented in the more effective 16-bit data length.

Status

All status and control bits for PACE are provided in a single status flag register, whose contents may be loaded from or to any accumulator or the stack. This allows convenient testing, masking and storage of status. In addition, a number of status bits may be tested directly by the conditional branch instruction, and any bit may be individually set or reset. The function of each bit in the status flag register is listed in Table I and described briefly below. The carry flag is set to the state of the carry output resulting from binary and BCD arithmetic instructions, and serves as a carry input for some of these instructions. The overflow flag is set true if an arithmetic overflow results from a binary arithmetic instruction.

TABLE I. Status Flag Register Bit Functions

Register Bit	Flag Name	Function
0	"1"	Not used—always logic 1
1	IE1	Interrupt Enable Level 1
2	IE2	Interrupt Enable Level 2
3	IE3	Interrupt Enable Level 3
4	IE4	Interrupt Enable Level 4
5	IE5	Interrupt Enable Level 5
6	OVF	Overflow
7	CRY	Carry
8	LINK	Link
9	IEN	Master Interrupt Enable
10	BYTE	8-bit data length
11	F11	Flag 11
12	F12	Flag 12
13	F13	Flag 13
14	F14	Flag 14
15	"1"	Always logic 1, addressed for Interrupt 0 exit

The link flag serves as a 1-bit extension for certain shift and rotate instructions. The byte flag is used to specify an 8-bit data length for data processing instructions, while arithmetic operations for address formation remain at the 16-bit data length. In the 8-bit data mode, modifications of the carry, overflow and link flag are based on the eight least significant data bits only.

Four flags (bits 10—14) are provided which may be assigned functions by the programmer. These flags drive output pins and may be used to directly control system functions or as software status flags. Bits 0 and 15 of the status register have not been implemented in hardware and always appear as a logic 1. The interrupt enable flags are explained below.

Control

The operation of the PACE microprocessor consists of repeatedly accessing or fetching instructions from the external program store and executing the operations specified by these instructions. These two steps are carried out under the control of a microprogram (the microprocessor is not designed for user microprogramming). The microprogram is similar to a state table specifying the series of states of system control signals necessary to carry out each instruction. Microprogram storage is provided by a programmable logic array, and microprogram routines are implemented to fetch and execute instructions. The fetch routine causes an instruction address to be transferred from the program counter register to the I/O bus and initiates an input data operation. When the instruction is provided on the data bus, the fetch routine causes it to be loaded into the instruction register. The instruction operation code is transformed into the address of the appropriate instruction-execution routine by the address generation logic. As the last step of the fetch routine, this address is loaded into the microprogram address register, causing a branch to the appropriate instruction execution routine. The execution routine consists of one or more microinstructions to implement the functions required by the instruction. For example, the routine for a register ADD instruction would access the two accumulators to be added over the operand bus, cause the ALU to perform an ADD operation, load the carry and overflow flags from the ALU and store the result in the specified accumulator. The control logic interprets the microinstructions to carry out these operations. The final step of the execution routine is a jump back to the fetch routine to access the next instruction. Each microcycle requires 2μs and 4 or 5 microcycles are typically required to fetch and execute a machine instruction. Other routines implemented by the microprogram include interrupt servicing and system initialization. The microprogram controls the operation of a conditional jump multiplexer which is used to specify 16 conditions for the conditional branch instruction. The conditions which may be tested are indicated in Table II and include four signal inputs to the chip, which may be used to test external system conditions.

TABLE II. Branch Conditions

Number	Mnemonic	Condition
0	STFL	Stack full
1	REQ0	(AC0) equal to zero(1)
2	PSIGN	(AC0) has positive sign(2)
3	BIT 0	Bit 0 of AC0 true
4	BIT 1	Bit 1 of AC0 true
5	NREQ0	(AC0) is non-zero(1)
6	BIT 2	Bit 2 of AC0 is true
7	CONTIN	CONTIN (continue) input is true
8	LINK	LINK is true
9	IEN	IEN is true
10	CARRY	CARRY is true
11	NSIGN	(AC0) has negative sign(2)
12	OVF	OVF is true
13	JC13	JC13 input is true
14	JC14	JC14 input is true
15	JC15	JC15 input is true

Note 1: If the selected data length is 8 bits, only bits 0-7 of AC0 are tested.

Note 2: Bit 7 is the sign bit (instead of bit 15) if the selected data length is 8 bits.

The control circuitry may be initialized at any time by use of the NINIT input signal. This will cause the stack addressing circuitry, all flags and the program counter to be set to zero, and the strobes to go false and level zero interrupt enable to go true. This signal should always be used to initialize the processor after applying power. The first instruction after initialization is accessed from location zero.

Interrupts

The PACE microprocessor provides a six level, vectored, priority interrupt structure. This allows automatic identification of an interrupting device's level and allows all devices on an interrupt level to be enabled or disabled as a group, independent of other interrupt levels. An individual interrupt enable is provided in the status register for each level, as shown in *Figure 3*, and a master

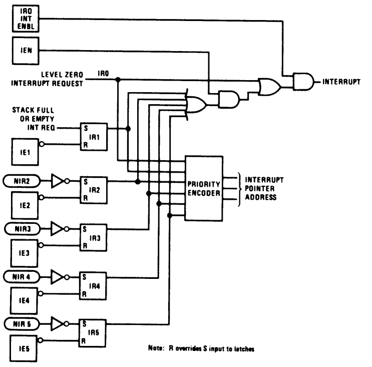

IRO INT ENBL
IEN
LEVEL ZERO INTERRUPT REQUEST
IRO
INTERRUPT
STACK FULL OR EMPTY INT REQ
S
IR1
R
IE1
NIR2
IR2
IE2
PRIORITY ENCODER
INTERRUPT POINTER ADDRESS
NIR3
IR3
IE3
NIR 4
IR4
IE4
NIR 5
IR5
IE5
Note: R overrides S input to latches

FIGURE 3. Interrupt System

interrupt enable (IEN) is provided for all 5 lower priority levels as a group. Negative true interrupt request inputs are provided to allow several interrupts to be "wire-ORed" on each input. When an interrupt request occurs, it will set the interrupt request latch if the corresponding interrupt enable is true. The latch will be set by any pulse exceeding one clock period in duration, which is useful for capturing narrow timing or control pulses. If the master interrupt enable (IEN) is true, then an interrupt will be generated. During the interrupt sequence an address is provided by the output of the priority encoder and is used to access the pointer for the highest-priority interrupt request (IR0 is highest priority, IR5 is lowest priority). The pointers are stored in locations 2–7 (see Table III) for interrupt requests 1–5 and 0, respectively. The pointer specifies the starting address of the interrupt service routine for that particular interrupt level. Before executing the interrupt service routine, the program counter is pushed on the stack and IEN is set false. The interrupt service routine may set IEN true after turning off the interrupt enable for the level currently being serviced (or resetting the interrupt request). (The interrupt enables may be set and reset using the SFLG and PFLG instructions.)

The non-maskable level zero interrupt (IR0) is an exception to this interrupt procedure. It has a program counter storage location pointer (the program counter is not stored on the stack for this particular interrupt in order to preserve the processor state) which is followed by the level zero interrupt service routine. The IR0 interrupt enable is cleared when a level zero interrupt occurs (IEN is unaffected) and may be set true by addressing (non-existent) status flag 15. This allows execution of one more instruction (typically JMP@) to return from the IR0 interrupt routine before another interrupt will be acknowledged. This interrupt level is typically used by the control panel, which then can always interrupt the application program and does not affect system status. The control panel service routine interprets and executes the functions specified by control panel switches and displays selected data on the panel lights. Level zero interrupts are generated by driving the NHALT signal line low.

TABLE III. Interrupt Pointer Table

8	Int 0 Program
7	Int 0 PC Pointer
6	Int 5 Pointer
5	Int 4 Pointer
4	Int 3 Pointer
3	Int 2 Pointer
2	Int 1 Pointer
1	Not Assigned
Loc 0	Initialization Inst

Data Input and Output

All data transfers between PACE and external memories or peripheral devices take place over the 16 data lines (D00–D15) and are synchronized by the 4 control signals (NADS, IDS, ODS, and EXTEND). Data transfers occur during each instruction access and during the data accesses required by memory reference instructions. This class of instructions could perhaps more properly be called the "I/O reference class" in the case of the PACE microprocessor, since all data transfers, whether with memory or peripheral devices or a central processor data bus, occur through the execution of these instructions. This unified bus architecture is in contrast with many other microprocessors and minicomputers that have one instruction type (I/O class) for communication with peripheral devices and another instruction type (memory reference class) for communication with memories. The advantage of the approach used by PACE is that a wider variety of instructions (the entire memory reference class) is available for communication with peripherals. Thus, the DSZ (decrement and skip if zero) instruction can be used to decrement a peripheral device register, or the SKAZ (skip if AND is zero) instruction can be used to test the contents of a peripheral device status register. The LD (load) and ST (store) instructions are used for simple data transfers.

All I/O transactions consist of an address output interval followed by a data transfer interval. The address specifies a memory location or peripheral device. The allocation is entirely up to the user (within the requirements for interrupt pointers). A straightforward allocation would be to assign all addresses from 0000_{16} to $7FFF_{16}$ as memory addresses and all addresses from 8000_{16} to $FFFF_{16}$ as peripheral device addresses. In this case, the most significant address bit specifies whether the transaction is with memory or a peripheral device. A variety of easily decoded address allocation schemes may be used, depending on the amount of ROM, RAM, peripheral devices and the particular application. Both address and data words are transmitted or received as 16-bit parallel data over the data lines (D00–D15). If 8-bit data is being transferred, the unused bits can be treated as "don't care" bits by the hardware and the 8-bit data length selected by the software.

Data transfer operations are synchronized by the NADS (Address Data Strobe), IDS (Input Data Strobe), ODS (Output Data Strobe) and EXTEND signals as shown in *Figure 4.* Address data is provided on the 16 data lines. An NADS is provided in the center of the address data and may be used to strobe the address into an address latch. A number of memory products provide address

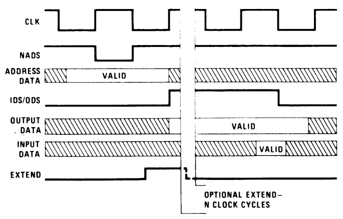


FIGURE 4. PACE I/O Timing

latches on the chip, which avoids the need for implementing this function externally. The input data strobe and output data strobe indicate the type of data transfer and may be used to enable TRI-STATE® I/O buffers and gate data into registers or memories as required by the system design. The EXTEND input allows the I/O cycle time to be extended by multiples of the clock cycle to adapt to a variety of memory and peripheral devices or for DMA bus interfacing.

INSTRUCTIONS

The PACE microprocessor provides a general-purpose mix of 45 instruction types. The memory reference instructions utilize a flexible memory addressing scheme providing three floating memory pages and one fixed page of 256 words each. The register instructions provide convenient data manipulation without requiring a memory access. The data transfer instructions provide a means of moving data among the functional blocks of the microprocessor system.

Addressing Modes

Instructions which use both direct and indirect memory addressing are included in the PACE instruction set. Three modes of direct memory addressing are available: base page, program counter relative, and index register relative. The mode of addressing is specified by the XR field of the instruction as illustrated in *Figure 5*.

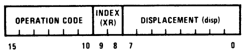


FIGURE 5. Memory Reference Instruction Format

When the XR field is 00, base page (page zero) addressing is used. Two different types of base page addressing are available and may be selected by the base-page-select (BPS) signal input. If BPS = 0, the 16-bit memory address is formed by setting bits 8 through 15 to zero, and using the 8-bit displacement (disp) for bits 0 through 7; this permits addressing of the first 256 words of memory (locations 0–255). If BPS = 1, the 16-bit memory address is formed by setting bits 8 through 15 equal to bit 7 of disp and using disp for bits 0 through 7; this permits addressing the first 128 words (0 through FF_{16}) and the last 128 words ($FF80_{16}$ through $FFFF_{16}$) of memory. The latter technique is useful for splitting the base page between read-write and read-only memories or between memory and peripheral devices, so the convenience of base page addressing is available for accessing data or peripherals.

Addressing relative to the program counter (PC) is specified when the XR field is 01. With this mode, the memory address is formed by adding the contents of the program counter to the value of the displacement field interpreted as a signed two's complement number (that is, the 8-bit disp field is interpreted as a 16-bit value with bits 8 through 15 set equal to bit 7; this allows representation of numbers from −128 through +127). When the address is formed, the program counter has already been incremented and contains a value one greater than the location of the current instruction; thus, memory addresses that may be referenced as 127 locations below through 128 above the address of the current instruction.

With the index register relative mode of addressing, any memory location within the 65,536 word address space may be referenced. The disp field is interpreted as a signed value ranging from −128 through 127 as with PC relative addressing. The memory address is formed by adding disp to the contents of either accumulator AC2 (when XR = 10) or accumulator AC3 (when XR = 11).

This type of addressing is very desirable for microprocessor applications which require address computation at execution time, since the use of read-only-memory for program storage prevents address modification within the program storage memory. A summary of the direct addressing modes is presented in Table IV.

TABLE IV. Summary of Addressing Modes

XR Field	Addressing Mode	Effective Address
00	Base Page	EA = disp
01	Program Counter Relative	EA = disp + (PC)
10	AC2 Relative (indexed)	EA = disp + (AC2)
11	AC3 Relative (indexed)	EA = disp + (AC3)

Note 1: For base page addressing, disp is positive and in the range of 000 to 255 if BPS = 0, and is a signed number in the range of −128 to +127 if BPS = 1.
Note 2: For relative addressing, disp has a range of −128 to +127.

Indirect addressing consists of first establishing an address in the same fashion as with direct addressing [by either the base page, relative to PC, or indexed (relative to AC2 or AC3) mode]. The 16-bit contents of the memory location at this address is then used as the address of the operand, allowing any memory location to be addressed.

As noted previously, the memory addressing modes are also used for peripheral I/O operations. The address space must be divided between read-write memory, read-only memory and I/O devices.

Instruction Summary

The instruction set is divided into eight instruction classes as listed in Table V. The branch instructions provide the means to transfer control anywhere in the 16-bit addressing space. Conditional branches are effected using the BOC instruction, which allows testing any one of 16 conditions, including status flags, the contents of AC0, and user inputs to the chip. Additional testing capability is provided by the skip instructions, which provide memory or peripheral to register comparisons

TABLE V. PACE Instruction Summary

	Mnemonic	Meaning	Operation	Assembler Format		Instruction Format			
1.	**Branch Instructions**								
	BOC	Branch On Condition	(PC) ← (PC) + disp if cc true	BOC	cc,disp	0 1 0 0	cc	disp	
	JMP	Jump	(PC) ← EA	JMP	disp (xr)	0 0 0 1 1 0	xr	disp	
	JMP@	Jump Indirect	(PC) ← (EA)	JMP	@disp (xr)	1 0 0 1 1 0			
	JSR	Jump To Subroutine	(STK) ← (PC), (PC) ← EA	JSR	disp (xr)	0 0 0 1 0 1			
	JSR@	Jump To Subroutine Indirect	(STK) ← (PC), (PC) ← (EA)	JSR	@disp (xr)	1 0 0 1 0 1			
	RTS	Return from Subroutine	(PC) ← (STK) + disp	RTS	disp	1 0 0 0 0 0	0 0	disp	
	RTI	Return from Interrupt	(PC) ← (STK) + disp, IEN = 1	RTI	disp	0 1 1 1 1 1			
2.	**Skip Instructions**								
	SKNE	Skip if Not Equal	If (ACr) ≠ (EA), (PC) ← (PC) + 1	SKNE	r,disp (xr)	1 1 1 1	r	xr	disp
	SKG	Skip if Greater	If (AC0) > (EA), (PC) ← (PC) + 1	SKG	0,disp (xr)	1 0 0 1 1 1			
	SKAZ	Skip if And is Zero	If [(AC0) ∧ (EA)] = 0, (PC) ← (PC) + 1	SKAZ	0,disp (xr)	1 0 1 1 1 0			
	ISZ	Increment and Skip if Zero	(EA) ← (EA) + 1, if (EA) = 0, (PC) ← (PC) + 1	ISZ	disp (xr)	1 0 0 0 1 1			
	DSZ	Decrement and Skip if Zero	(EA) ← (EA) – 1, if (EA) = 0, (PC) ← (PC) + 1	DSZ	disp (xr)	1 0 1 0 1 1			
	AISZ	Add Immediate, Skip if Zero	(ACr) ← (ACr) + disp, if (ACr) = 0, (PC) ← (PC) + 1	AISZ	r,disp	0 1 1 1 1 0	r		
3.	**Memory Data Transfer Instructions**								
	LD	Load	(ACr) ← (EA)	LD	r,disp (xr)	1 1 0 0	r	xr	disp
	LD@	Load Indirect	(AC0) ← ((EA))	LD	0,@disp (xr)	1 0 1 0 0 0			
	ST	Store	(EA) ← (ACr)	ST	r,disp (xr)	1 1 0 1	r		
	ST@	Store Indirect	((EA)) ← (AC0)	ST	0,@disp (xr)	1 0 1 1 0 0			
	LSEX	Load With Sign Extended	(AC0) ← (EA) bit 7 extended	LSEX	0,disp (xr)	1 0 1 1 1 1			
4.	**Memory Data Operate Instructions**								
	AND	And	(AC0) ← (AC0) ∧ (EA)	AND	0,disp (xr)	1 0 1 0 1 0	xr	disp	
	OR	Or	(AC0) ← (AC0) ∨ (EA)	OR	0,disp (xr)	1 0 1 0 0 1			
	ADD	Add	(ACr) ← (ACr) + (EA), OV, CY	ADD	r,disp (xr)	1 1 1 0	r		
	SUBB	Subtract with Borrow	(AC0) ← (AC0) + ~ (EA) + (CY), OV, CY	SUBB	0,disp (xr)	1 0 0 1 0 0			
	DECA	Decimal Add	(AC0) ← (AC0) $+_{10}$ (EA) $+_{10}$ (CY), OV, CY	DECA	0,disp (xr)	1 0 0 0 1 0			
5.	**Register Data Transfer Instructions**								
	LI	Load Immediate	(ACr) ← disp	LI	r,disp	0 1 0 1 0 0	r	disp	
	RCPY	Register Copy	(ACdr) ← (ACsr)	RCPY	sr,dr	0 1 0 1 1 1	dr	sr	not used
	RXCH	Register Exchange	(ACdr) ← (ACsr), (ACsr) ← (ACdr)	RXCH	sr,dr	0 1 1 0 1 1			
	XCHRS	Exchange Register and Stack	(STK) ← (ACr), (ACr) ← (STK)	XCHRS	r	0 0 0 1 1 1	r	not used	
	CFR	Copy Flags Into Register	(ACr) ← (FR)	CFR	r	0 0 0 0 0 1			
	CRF	Copy Register Into Flags	(FR) ← (ACr)	CRF	r	0 0 0 0 1 0			
	PUSH	Push Register Onto Stack	(STK) ← (ACr)	PUSH	r	0 1 1 0 0 0			
	PULL	Pull Stack Into Register	(ACr) ← (STK)	PULL	r	0 1 1 0 0 1			
	PUSHF	Push Flags Onto Stack	(STK) ← (FR)	PUSHF		0 0 0 0 1 1	not used		
	PULLF	Pull Stack Into Flags	(FR) ← (STK)	PULLF		0 0 0 1 0 0			
6.	**Register Data Operate Instructions**								
	RADD	Register Add	(ACdr) ← (ACdr) + (ACsr), OV, CY	RADD	sr,dr	0 1 1 0 1 0	dr	sr	not used
	RADC	Register Add With Carry	(ACdr) ← (ACdr) + (ACsr) + (CY), OV, CY	RADC	sr,dr	0 1 1 1 0 1			
	RAND	Register And	(ACdr) ← (ACdr) ∧ (ACsr)	RAND	sr,dr	0 1 0 1 0 1			
	RXOR	Register Exclusive OR	(ACdr) ← (ACdr) $\veebar$ (ACsr)	RXOR	sr,dr	0 1 0 1 1 0			
	CAI	Complement and Add Immediate	(ACr) ← ~ (ACr) + disp	CAI	r,disp	0 1 1 1 0 0	r	disp	
7.	**Shift And Rotate Instructions**								
	SHL	Shift Left	(ACr) ← (ACr) shifted left n places, w/wo link	SHL	r,n,ℓ	0 0 1 0 1 0	r	n	ℓ
	SHR	Shift Right	(ACr) ← (ACr) shifted right n places, w/wo link	SHR	r,n,ℓ	0 0 1 0 1 1			
	ROL	Rotate Left	(ACr) ← (ACr) rotated left n places, w/wo link	ROL	r,n,ℓ	0 0 1 0 0 0			
	ROR	Rotate Right	(ACr) ← (ACr) rotated right n places, w/wo link	ROR	r,n,ℓ	0 0 1 0 0 1			
8.	**Miscellaneous Instructions**								
	HALT	Halt	Halt	HALT		0 0 0 0 0 0	not used		
	SFLG	Set Flag	$(FR)_{fc}$ ← 1	SFLG	fc	0 0 1 1	fc	1	not used
	PFLG	Pulse Flag	$(FR)_{fc}$ ← 1, $(FR)_{fc}$ ← 0	PFLG	fc	0 0 1 1	fc	0	

without altering data. The memory data transfer instructions provide data transfers between the accumulators and memory or peripheral devices. The load with sign extended is provided to convert 8-bit, two's complement data to 16-bit data, allowing 16-bit address modification when the 8-bit data length has been selected.

The memory data operate instructions provide operations between the principal working register (AC0) and memory or peripheral data. This includes both binary and BCD arithmetic instructions. The register data transfer instructions provide a very complete set of transfer possibilities between the accumulators, flag register and stack, and include the capability to load immediate data. Register data operate instructions provide logical and arithmetic operations between any two accumulators. They may be used for address and data modification and to reduce the number of (time consuming) memory references in a program. The shift and rotate instructions allow 8 different operations which are useful for multiply, divide, bit scanning and serial input-output operations. The miscellaneous instructions include the capability to set or reset (pulse) any of the 16 bits of the status flag register individually. Instruction execution times are shown in Table VI.

A simple example program is provided by the binary multiply routine shown on page 9. This program multiplies the 16-bit value in AC2 by the 16-bit value in AC1 and provides a 32-bit result in AC0 (high order) and AC1 (low order). Worst case execution time is under one millisecond.

Binary Multiply Routine

```
CONST:  .WORD  X'FFFF       ; CONSTANT FOR DOUBLE PREC. ADD
START:  LI     R1, 0        ; CLEAR RESULT REGISTER
        LI     R3, 16       ; LOOP COUNT TO AC3
        CAI    R0, 0        ; COMPLEMENT MULTIPLIER
LOOP:   RADD   R1, R1       ; SHIFT RESULT LEFT INTO CARRY
        RADC   R0,R0        ; SHIFT CARRY INTO MULTIPLIER
                            ; AND MULTIPLIER INTO CARRY
        BOC CARRY, TEST     ; TEST FOR ADD
        RADD   R2, R1       ; ADD MULTIPLICAND TO RESULT
        SUBB   R0, CONST    ; ADD CARRY TO H.O. RESULT
TEST:   AISZ   R3, -1       ; DECREMENT LOOP COUNT
        JMP LOOP            ; REPEAT LOOP
```

TABLE VI. Instruction Execution Times

Mnemonic	Meaning	Execution Time
1. Branch Instructions		
BOC	Branch On Condition	$5M + E_R + 1M$ if branch
JMP	Jump	$4M + E_R$
JMP@	Jump Indirect	$4M + 2E_R$
JSR	Jump To Subroutine	$5M + E_R$
JSR@	Jump To Subroutine Indirect	$5M + 2E_R$
RTS	Return from Subroutine	$5M + E_R$
RTI	Return from Interrupt	$6M + E_R$
2. Skip Instructions		
SKNE	Skip if Not Equal	$5M + 2E_R + 1M$ if skip
SKG	Skip if Greater	$7M + 2E_R + 1M$ if skip
SKAZ	Skip if And is Zero	$5M + 2E_R + 1M$ if skip
ISZ	Increment and Skip if Zero	$7M + 2E_R + E_W + 1M$ if skip
DSZ	Decrement and Skip if Zero	$7M + 2E_R + E_W + 1M$ if skip
AISZ	Add Immediate, Skip if Zero	$5M + E_R + 1M$ if skip
3. Memory Data Transfer Instructions		
LD	Load	$4M + 2E_R$
LD@	Load Indirect	$5M + 3E_R$
ST	Store	$4M + E_R + E_W$
ST@	Store Indirect	$4M + 2E_R + E_W$
LSEX	Load With Sign Extended	$4M + 2E_R$
4. Memory Data Operate Instructions		
AND	And	$4M + 2E_R$
OR	Or	$4M + 2E_R$
ADD	Add	$4M + 2E_R$
SUBB	Subtract With Borrow	$4M + 2E_R$
DECA	Decimal Add	$7M + 2E_R$
5. Register Data Transfer Instructions		
LI	Load Immediate	$4M + E_R$
RCPY	Register Copy	$4M + E_R$
RXCH	Register Exchange	$6M + E_R$
XCHRS	Exchange Register and Stack	$6M + E_R$
CFR	Copy Flags Into Register	$4M + E_R$
CRF	Copy Register Into Flags	$4M + E_R$
PUSH	Push Register Onto Stack	$4M + E_R$
PULL	Pull Stack Into Register	$4M + E_R$
PUSHF	Push Flags Onto Stack	$4M + E_R$
PULLF	Pull Stack Into Flags	$4M + E_R$
6. Register Data Operate Instructions		
RADD	Register Add	$4M + E_R$
RADC	Register Add With Carry	$4M + E_R$
RAND	Register And	$4M + E_R$
RXOR	Register Exclusive Or	$4M + E_R$
CAI	Complement and Add Immediate	$5M + E_R$
7. Shift And Rotate Instructions		
SHL	Shift Left	$(5 + 3n)\ M + E_R,\ n = 1 - 127;$ $6M + E_R,\ n = 0$
SHR	Shift Right	
ROL	Rotate Left	
ROR	Rotate Right	
8. Miscellaneous Instructions		
HALT	Halt	
SFLG	Set Flag	$5M + E_R$
PFLG	Pulse Flag	$6M + E_R$

M = Machine cycle time = 4 clock periods E_R = Extend time for read cycle
n = number of shifts E_W = Extend time for write cycle
Note: External interrupt response time is $7M + E_R$ plus time to finish current instruction.

While the instruction set is compact at 45 instruction types (or 337 individual instructions), it is powerful enough to allow considerably more efficient program coding than most microprocessors and compares favorably with many minicomputers.

I/O DESCRIPTION

Drivers and Receivers

Equivalent circuits for PACE drivers and receivers are shown in *Figure 6*. All inputs have static charge protection circuits consisting of an RC filter and voltage clamp. These devices should still be handled with care, as the protection circuits can be destroyed by excessive static charge. Pullup transistors on several inputs are turned on during one of the eight internal clock phases. In the case of bidirectional signals, the output driver transistors also serve as input pullup transistors.

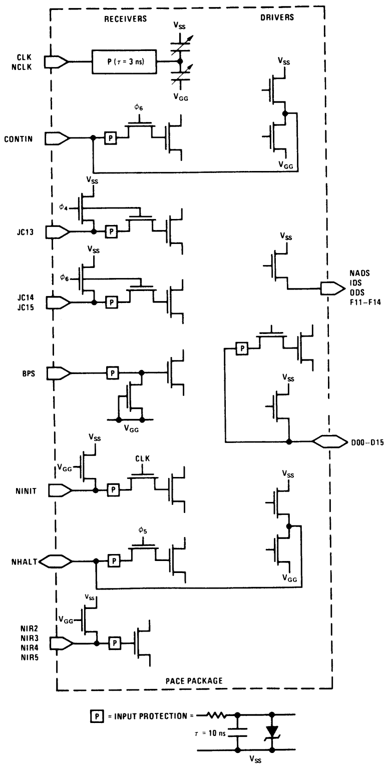


FIGURE 6. PACE Driver and Receiver Equivalent Circuits

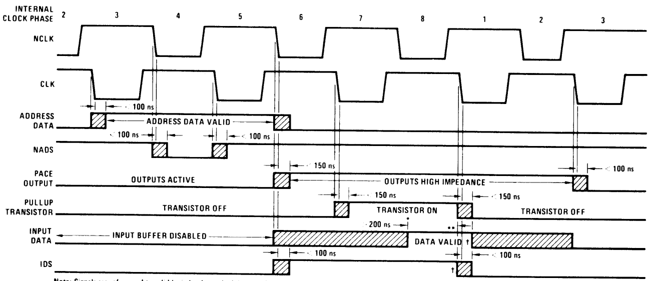
Data I/O Timing

All data transfers between PACE and external memories or peripheral devices take place over the 16 data lines. These transfers are synchronized by the NADS, IDS, ODS and EXTEND signals. Timing for address data output is shown in *Figure 7.* All signal timing is referenced to valid logic "1" or logic "0" clock levels. Cross-hatched areas indicate uncertainty of output transitions or "don't care" (optional) states for data inputs. Address data becomes valid one clock phase prior to the Address Data Strobe and remains valid for one clock phase afterwards. Typically, NADS will be used to strobe the address data into a latch, either internal or external to the memory chips, or to clock decoded peripheral addresses into a flip-flop.

The PACE address output drivers assume a high impedance state during the data input interval as shown in *Figure 7.* The IDS signal may be used to disable the output sense amplifiers and enable TRI-STATE® input buffers. Increased power supply current may occur during the transition period of the TRI-STATE enable signal, when several devices may be simultaneously enabled. Therefore, good power and ground layout and bypass filtering practice should be observed. The data lines must be driven to valid input data logic levels by the end of IDS, and all logic 1 inputs must reach a minimum intermediate level of $V_{SS} - 2.35V$ 200 ns prior to the end of internal clock phase 8. TTL devices will actively drive the input to this minimum intermediate level and the transition will be completed by a combination of the on-chip pullup transistor and the (reduced) TTL output drive current. Typically, this data input timing will allow operation of the microprocessor in a system at maximum speed if the access time of the system memory is less than 700 ns. For memories with longer access times the clock frequency may be reduced or the I/O cycle extend feature may be used, as described below.

Data output timing is shown in *Figure 8.* Output data becomes valid at the leading edge of ODS and remains valid for one clock period following the trailing edge.

The Output Data Strobe is typically used as a read-write signal for memory and an output data latch strobe for peripheral interfaces.



Note: Signals are referenced to valid logic levels on clock inputs. All times in Figures 7 – 10 are typical maximums or minimums. Internal clock phases are shown for reference only, they are not available externally.

*V_{IN} must be > $V_{SS} - 2.35V$ at this time if logic "1" input.

**V_{IN} must be valid level (i.e., $V_{SS} - 1$) at this time (this timing allows for pull-up transistor time constant).

†Data must be valid until trailing edge of IDS (i.e., data hold time = 0 ns).

FIGURE 7. Address Output and Data Input Timing

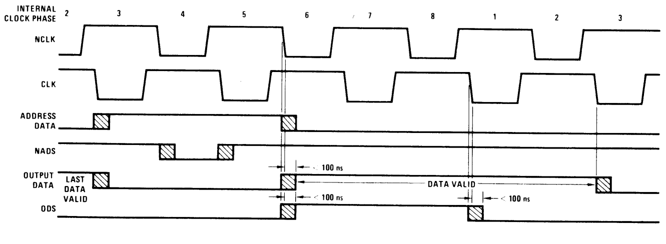


FIGURE 8. Data Output Timing

For systems utilizing memories with access times greater than 700 ns it may be desirable to use the EXTEND input to lengthen the I/O cycle by multiples of the clock period. Timing for this is shown in *Figure 9*. In the case of either input or output operations, the extend should be brought true prior to the end of internal phase 6. The timing shown in *Figure 9* will provide the minimum extend of one clock period. Holding EXTEND true for and additional n clock periods longer will cause an extention of n + 1 clock periods. As indicated in the electrical characteristics, there must be at least 64 non-extend clock cycles every 640 microseconds. This includes the use of EXTEND for both extending and suspending I/O operations.

In DMA or multiprocessor systems it may be desirable to prevent I/O operations by PACE when the bus is in use by another device. This may be done by using the EXTEND signal immediately following an IDS or ODS as shown in *Figure 10*. Alternatively, the extend timing of *Figure 9* may be used, as the extend function occurs independent of whether there is an I/O operation, that is, whenever the internal clock phase 6 occurs.

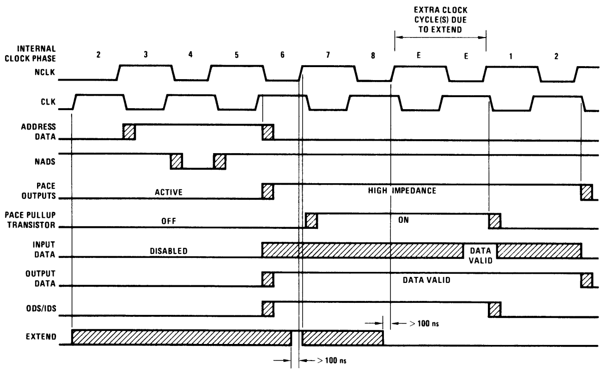


FIGURE 9. Extend I/O Signal Timing

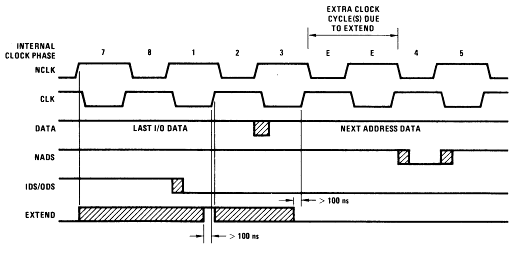
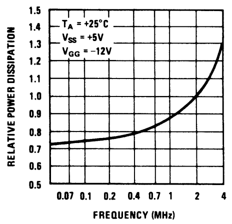


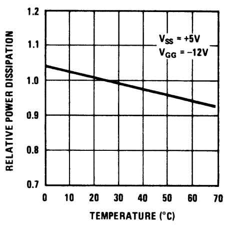
FIGURE 10. Suspend I/O Signal Timing

typical performance characteristics

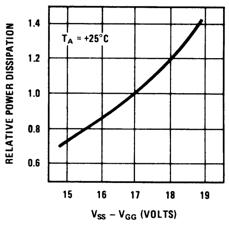
Power Dissipation vs Clock Frequency



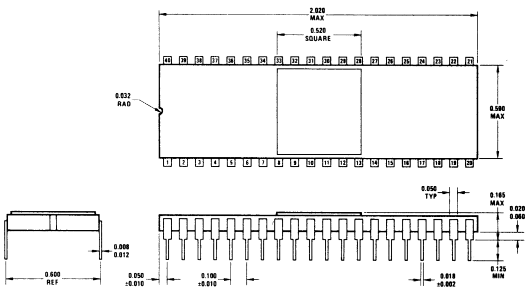
Power Dissipation vs Temperature



Power Dissipation vs Supply Voltage



physical dimensions



Cavity Dual-In-Line Package (D)

National Semiconductor Corporation
2900 Semiconductor Drive, Santa Clara, California 95051, (408) 732-5000/TWX (910) 339-9240
National Semiconductor GmbH
808 Fuerstenfeldbruck, Industriestrasse 10, West Germany, Tele. (08141) 1371/Telex 05-27649
National Semiconductor (UK) Ltd.
Larkfield Industrial Estate, Greenock, Scotland, Tele. (0475) 33251/Telex 778-632

NATIONAL

CHAPTER 1
INTRODUCTION TO PACE

1.1 DESCRIPTION

National Semiconductor's Processing and Control Element, called PACE, is a *single-chip* full-feature Central Processing Unit (CPU). PACE is housed in a 40-pin, ceramic, dual-in-line package. The ultrahigh density and overall layout of the microcircuit are shown in figure 1-1.

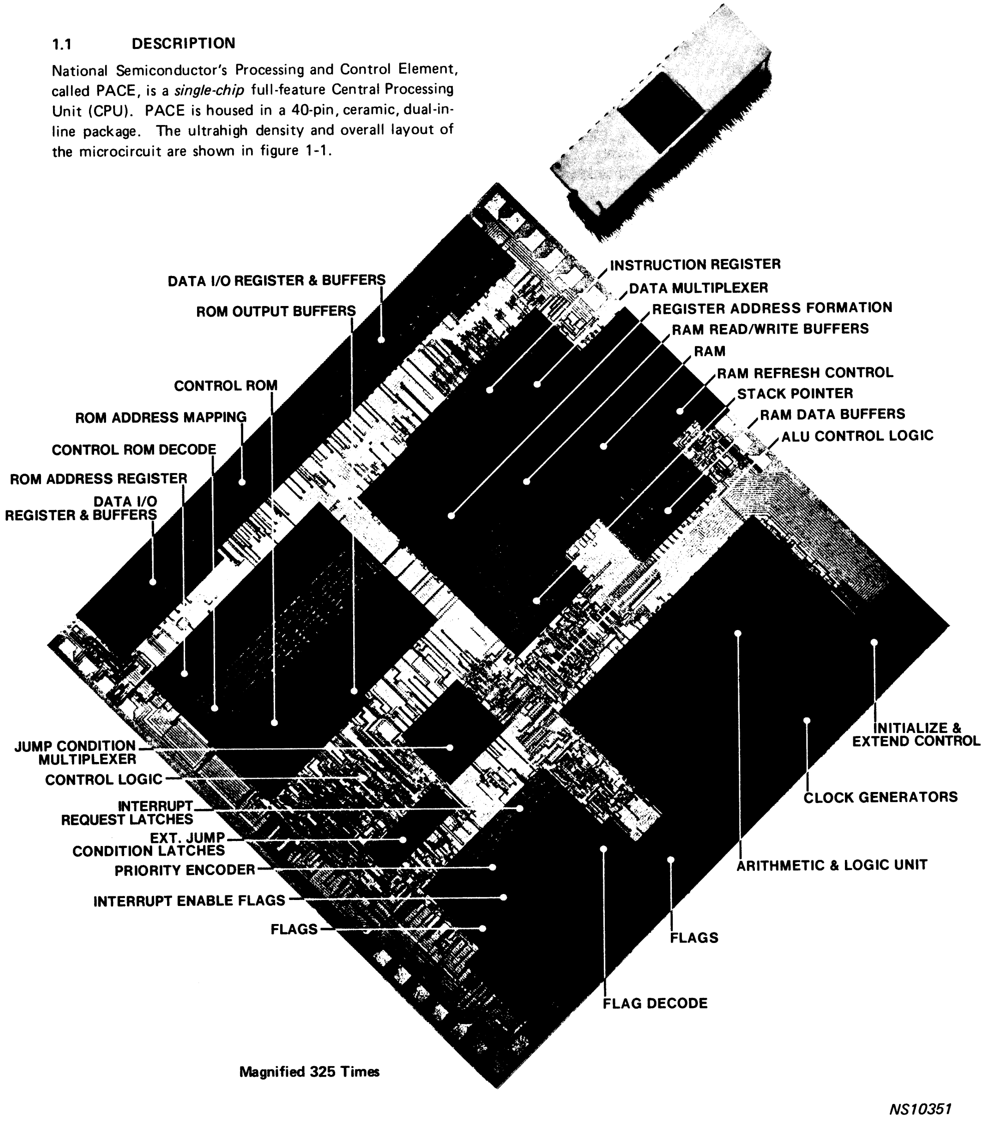


Figure 1-1. PACE Chip and Circuit Layout

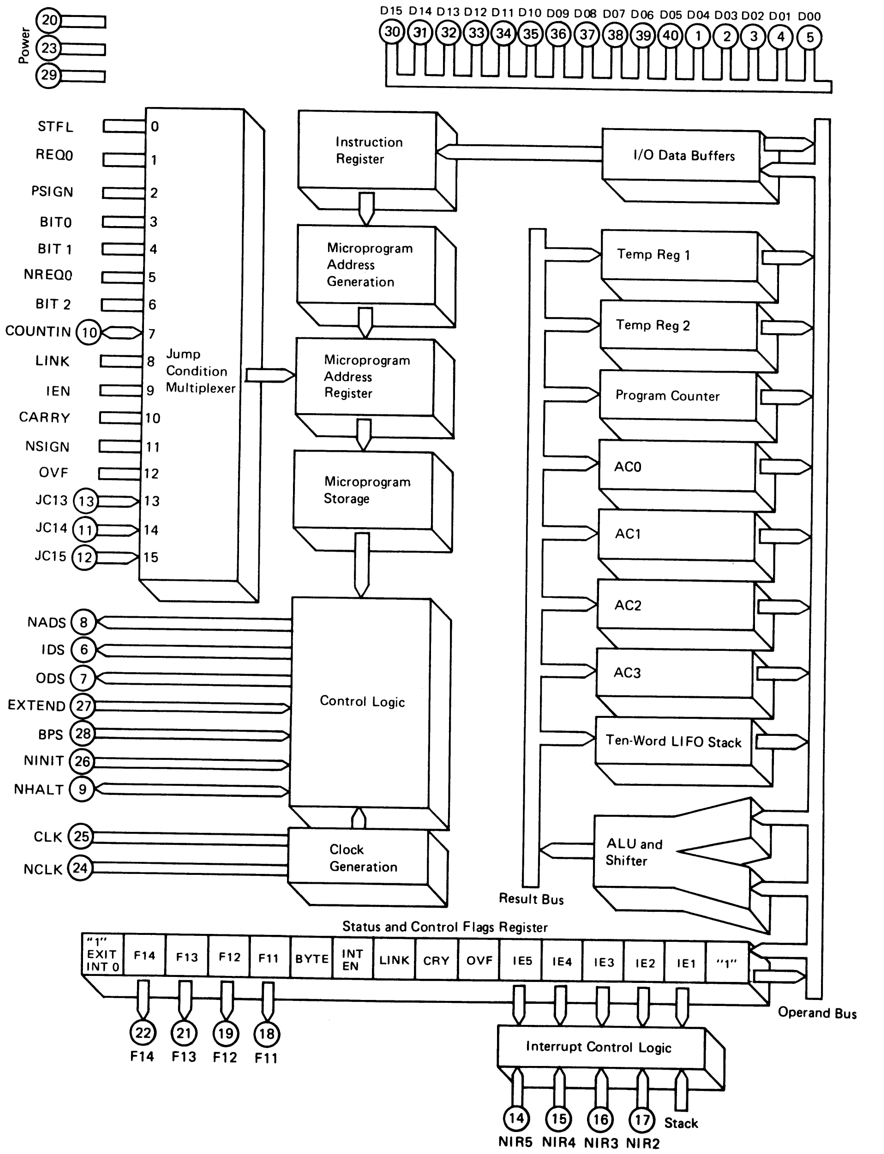


Figure 1-2. PACE Functional Block Diagram

PACE also is called a microprocessor, the prefix *micro* relating to the microscopic size of the physical circuit and components on the chip. An extraordinary amount of data-processing capability is provided in one component by the *single-chip* microconstruction.

Figure 1-2 reveals the CPU architecture and pinouts of the *single-chip* PACE – consisting of registers, control logic, an arithmetic unit, and the data buses. Some of the outstanding operational features of the PACE microprocessor are listed below.

Features

- 16-bit instruction word offers addressing flexibility and speed.
- 8- or 16-bit data word interfaces increase application flexibility.
- 45 instruction types provide efficient programming.
- Common memory and peripheral addressing means powerful I/O instructions.
- Shares instructions with National Semiconductor's IMP-16, allowing software compatibility.
- Four general-purpose accumulators reduce memory data transfers.
- 10-word Stack is utilized for interrupt processing/data storage.
- Six vectored priority-interrupt levels speed interrupt service and simplify hardware.
- Programmer-accessible status register may be preserved, tested, or modified.
- Typical 10-microsecond instruction execution guarantees high throughput.
- 1K-by-16 Read-Only Memory allows single-memory package systems.
- Single-phase true and complement clock minimizes external components.
- +5-volt and -12-volt standard supplies ensure minimum cost.

The PACE MOS/LSI chip is produced using silicon-gate, P-channel enhancement-mode standard-process technology. This means that the following very significant advantages are realized.

- Lower cost per function
- Lower component count
- Simplified design
- Higher reliability
- High noise immunity
- Low threshold voltage

Among some of the benefits of a single-chip device with the above-listed advantages are the following.

- LOWER COMPONENT COUNT --- Generally, this means lower procurement, incoming testing, inventory, handling, rework, and assembly cost – and higher reliability.
- SIMPLIFIED DESIGN --- LSI devices enable engineering design groups to take advantage of prepackaged circuits that are self-contained and perform a unified function. Also, a design group not strongly oriented towards digital design may make use of the latest techniques and devices without requiring expertise related to the design methods of interfacing the circuits internal to LSI devices. In summary, use of LSI devices requires considerably less engineering time to develop a product.
- HIGHER RELIABILITY --- The long history of field maintenance of all types of electronic systems clearly demonstrates the high reliability of LSI devices. System maintenance has shown that the reliability of low-power circuits is inversely proportional to the number of component lead connections in the system. This factor, coupled with the abundant functional capability of LSI, greatly increases the probability that an LSI-based system will function properly over extended periods of time.
- IMPROVED PERFORMANCE --- PACE offers higher throughput because of a powerful instruction set, a proven architecture, and 16-bit address generation and data handling.
- LOWER COST --- The reduction of cost is an aggregate savings resulting from the other advantages already enumerated. In the microprocessor field, the inherent functional superiority of high-density devices is seldom questioned. The superior performance of the single-chip PACE, coupled with reduced engineering and assembly cost, higher reliability, lower operating and maintenance costs, and smaller size of the microprocessor, definitely results in a much better performance-to-price ratio than heretofore possible. It makes the PACE microprocessor the most competitive processor on the market.

1.2 OUTSTANDING FEATURES OF PACE

The outstanding features of PACE are described in detail later. Nevertheless, to provide an overall view of the many favorable facets of PACE, these features are listed and briefly described below.

- 8- OR 16-BIT DATA HANDLING --- PACE is cost effective in applications dominated by 8-bit data interfaces. Efficient coding and address generation found only in 16-bit microprocessors are extended to 8-bit applications.
- INCREASED THROUGHPUT --- PACE minimizes data and program storage requirements, while increasing data-processing throughput.
- USER GROUP --- Membership open to users and others interested in microprocessors. Provides a vehicle of communications between members and with National Semiconductor. Makes programs available from its User Group Software Library.

- SOFTWARE SUPPORT --- Includes Source Statement Editor, Assemblers, Loaders, Debug Routine, Utilities, and Diagnostics. (Also, as previously mentioned, a Source Statement Translator converts IMP-16 software to PACE software.)

1.3 PACE APPLICATIONS

Applications for PACE could very well be in the thousands. The suitability of PACE will, in many cases, be a matter of evaluation by potential users for their particular needs. A few applications are listed.

- Test system and instrument control
- Process controllers
- Machine tool control
- Terminal control
- Small business machines
- Traffic controllers
- Word-processing systems
- Peripheral device controllers
- Educational controllers
- Sophisticated games
- Distributed and multiprocessor systems
- Automotive controller

1.8 SOFTWARE SUPPORT

The importance of National Semiconductor-supplied support software cannot be overemphasized. The microprocessor design process is most efficient when the designer fully appreciates and uses the support software.

1.9 SPECIFYING HARDWARE BEHAVIOR WITH SOFTWARE

The microprocessor approach differs from older, discrete-logic controllers in only one important way, and all differences in approach stem from the following.

- In the random-logic approach, a set of logic is wired to handle each function, and all logic operations proceed in parallel.
- In the microprocessor approach, one central set of logic is provided inside the microprocessor. The central set of logic is rewired in real time, under control of the program, to handle each of the logic functions in serial.

Thus, the discrete-logic designer buys a set of functions and then wires the functions to perform a specific job. On the other hand, the microprocessor user buys a microprocessor and then must tell the microprocessor how to wire itself, from microsecond to microsecond, to perform different jobs.

The purpose of the system software is to aid the user in describing, designing, and debugging a microsecond-to-microsecond description of the microprocessor wiring. Such a description, when rendered in terms the microprocessor can understand, is called a program.

1.10 TYPES OF SOFTWARE

One of the most important steps in the programming process is the translation of the program description from commands that the programmer writes and understands into binary strings that the microprocessor uses to perform operations. Two types of commonly used translator programs are assemblers and compilers.

Utility programs facilitate the preparation of input code (using the Editor Program) and the debugging of the resultant object code (using the Debug Program) and, also, are used to enter the programs (using the Loader Program) into the Microprocessor Development System.

The following paragraphs provide more detailed descriptions of assembler and compiler programs and three types of utility programs (Editor, Debug, and Loaders).

Figure 1-11 gives a birds-eye view of the PACE computer-program breakdown.

NOTE

A software summary table of the PACE software line is located in appendix A, table A-2.

1.10.1 PACE EDITOR

The PACE Editor enables the generation of new source statement text and the modification of existing source text in preparation for program assembly. The normal editing procedure is to input assembly-language source statements and comments, edit the text, and output the edited text, along with a punched paper tape suitable for input to the assembler.

1.10.2 PACE ASSEMBLERS

The user has the alternative of selecting among four PACE assemblers: the PACE Resident Assembler, the PACE IMP-16 Cross Assembler, the PACE Conversational Assembler, and the PACE FORTRAN Cross Assembler. All Assemblers are completely compatible in programs assembled and vary only in operating environments.

The PACE Resident Assembler runs on an IPC-16P. The PACE IMP-16 Cross Assembler runs on an IMP-16P or IMP-16L with a minimum of 4K words of memory and a TTY. The PACE Resident Assembler and PACE IMP-16 Cross Assembler accept *free-format* source statements from either the keyboard, paper tape, or a card reader and produce a Load Module (LM) on paper tape and an object listing on the TTY printer. The Resident and Cross Assemblers require three passes over the source program; however, if either the object listing or the LM is suppressed, only two passes are required.

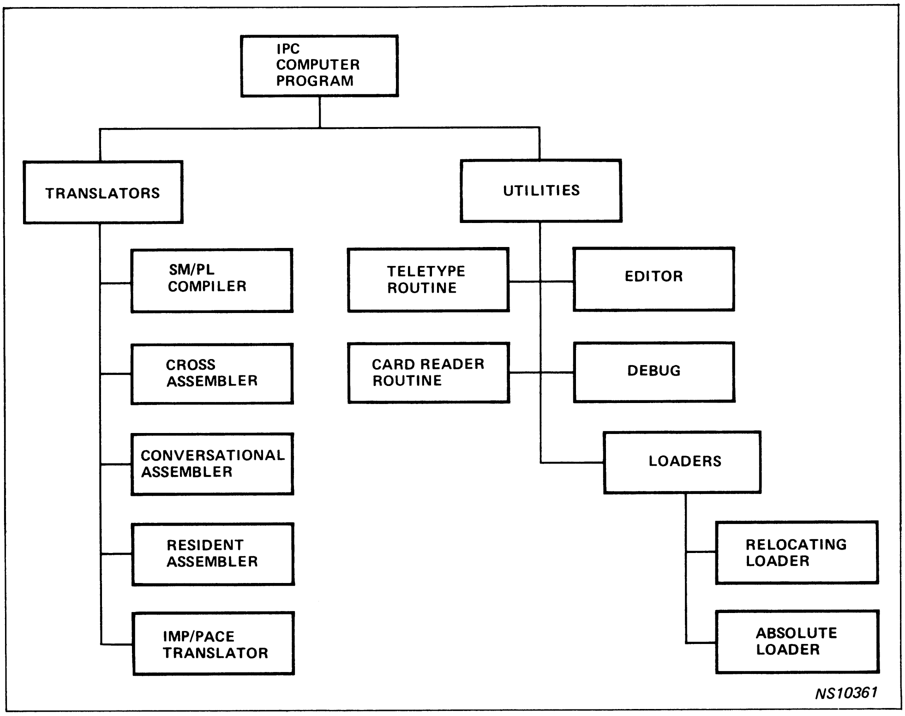


Figure 1-11. PACE Computer Programs

The PACE Conversational Assembler, which runs on an IPC-16P and combines the features of an editor and a resident assembler, simplifies the editing and assembly procedures by eliminating the need for multiple loadings of an editor, a resident assembler, and the user-generated program. The PACE Conversational Assembler requires 8K words of memory for operation.

The PACE FORTRAN Cross Assembler Program generates an object program from a source program on a host computer for subsequent execution by a PACE microprocessor. The assembler may be used on different host processors since the assembler is written in FORTRAN IV (USA Standard Language Subset). The assembler requires 100K bytes of memory and the following minimum hardware complement: processor input unit, scratch unit, list output unit, and binary output unit.

The PACE FORTRAN Cross Assembler accepts *free-format* source statements and, in two passes, produces an LM (object program) and a program listing.

1.10.3 PACE SM/PL COMPILER

The PACE SM/PL Compiler is a high-level computer program written in IPC assembly language. Comparable to high-level-language programming of the large-scale computers and minicomputers, the SM/PL Compiler considerably simplifies microcomputer programming. This results in fewer programming manhours and shortens leadtime – and, hence, reduces programming cost.

The SM/PL Compiler runs on an IPC-16P Microprocessor Development System, with a requirement of at least 12K memory words. The object code thus produced is highly efficient – in many cases comparable to the object code produced by programs written in the IPC assembly language. The object code is in standard Relocatable-Load-Module (RLM) format. All IPC peripherals are supported by the SM/PL Compiler.

A sequence of declarations and statements comprise the language of the SM/PL Compiler. Declarations control allocation of storage, define simple macros, and define procedures. Statements compute results and store them in a location defined by a variable name; statements also provide conditional tests and branching, iteration control, and procedure innovation.

Compiler procedures are in the form of subroutines that are defined by declarations and called by statements. Each subroutine may represent a program module, so a particular program may perform a number of tasks, each task being implemented by a subroutine. These subroutines are available to be used as procedures as part of other similar programs.

Figure 1-12 illustrates the software used to assemble or compile on PACE.

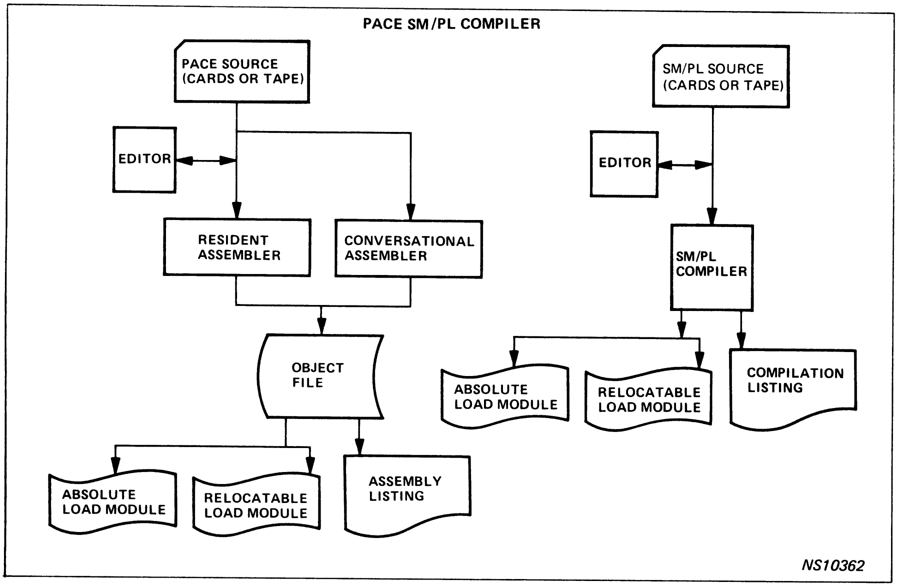


Figure 1-12. Software Used to Assemble or Compile on PACE

1.10.5 PACE LOADERS

The PACE loaders are programs that read and load one or more LMs, produced by a PACE assembler, into the main memory for execution.

The output by the PACE FORTRAN Cross Assembler is reformatted into an LM before loading into the PACE memory for execution. ANSI FORTRAN programs are available to reformat the output from the PACE FORTRAN Cross Assembler into an LM suitable to the loader and loading method employed.

The outputs from the PACE Resident Assembler and the PACE IMP-16 Cross Assembler do not require reformatting. The LMs are output directly from the PACE Resident Assembler and PACE IMP-16 Cross Assembler onto paper tape.

Two methods are available for loading data into the main memory for execution: absolute and relocatable. Each loading method involves tradeoffs among the following considerations: the complexity of the loading process, the amount of work that must be performed by the user, and the flexibility available to the user at load time (versus assembly time).

Several PACE programs are available for loading correctly formatted LMs into the PACE memory for execution: PACE Relocating Loader (PACE General Loader), PACE Absolute Card Reader Loader, and PACE Absolute Paper Tape Loader. The loading methods and the loaders available for each method are described in the following paragraphs.

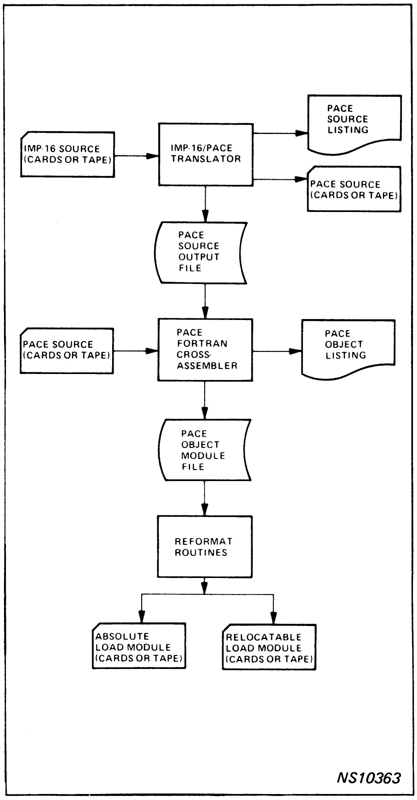

IMP-16 SOURCE (CARDS OR TAPE)
IMP-16/PACE TRANSLATOR
PACE SOURCE LISTING
PACE SOURCE (CARDS OR TAPE)
PACE SOURCE OUTPUT FILE
PACE SOURCE (CARDS OR TAPE)
PACE FORTRAN CROSS-ASSEMBLER
PACE OBJECT LISTING
PACE OBJECT MODULE FILE
REFORMAT ROUTINES
ABSOLUTE LOAD MODULE (CARDS OR TAPE)
RELOCATABLE LOAD MODULE (CARDS OR TAPE)
NS10363

Figure 1-13. PACE Software Implemented on a Host Computer

1.10.5.1 PACE Absolute Loaders

A PACE Absolute Loader, resident in the ROM of the IPC-16P, loads one or more programs into preallocated, fixed areas of memory. The exact memory areas to be occupied by each user-generated program must be determined by the user before assembly. Also, any linking of one program to another or to common, shared data must be accomplished at assembly time by assignment of common labels to fixed, absolute addresses in memory. The advantages of using a PACE Absolute Loader are that a small, simple loader may be used and no commands are required at load time.

1.10.5.2 PACE Relocating Loader

The PACE Relocating Loader (PACE General Loader) is a command-driven PACE program that reads one or more relocatable LMs from either the Card Reader or the Paper Tape Reader, relocates object code, and transfers control to the specified entry point. The PACE Relocating Loader provides the most flexible loading process. The PACE Relocating Loader process allows relocation of programs at LM load time rather than at assembly time. The PACE Relocating Loader follows either an inherent method for allocating programs to available memory or user-generated instructions that designate where each program should be loaded. Figure 1-14 illustrates the PACE loading sequence.

1.10.6 PACE INPUT/OUTPUT ROUTINES

The PACE Input/Output Routines are described in the following paragraphs.

1.10.6.1 PACE Teletype Routines

The PACE Teletype Routines reside in ROM on the TTY/Card Reader Interface Card of the IPC-16P System. The routines are used to send and receive information to and from the TTY or to receive data from the Paper Tape Reader. When both a High-speed Paper Tape Reader and a TTY Paper Tape Reader are used, the program verifies the tape reader that first supplies data and, subsequently, accepts input data from that tape reader.

1.10.6.2 PACE Card Reader Routine

The PACE Card Reader Routine resides in ROM on the TTY/Card Reader Interface Card of the IPC-16P System. The PACE Card Reader Routine accepts an absolute LM in Hollerith-coded card format and loads the data into main memory. There are no restrictions on loadable addresses; any read/write memory location can be used.

1.10.7 PACE DEBUG PROGRAM

The PACE Debug Program supervises the operation of a user program during checkout. This program provides the following facilities for testing computer programs:

- Printing selected areas of memory in hexadecimal or ASCII format
- Modifying the contents of selected areas in memory
- Modifying computer registers and stack
- Inserting instruction breakpoint halts
- Taking memory snapshots during execution of a user program
- Initiating execution at any point in a program
- Searching memory

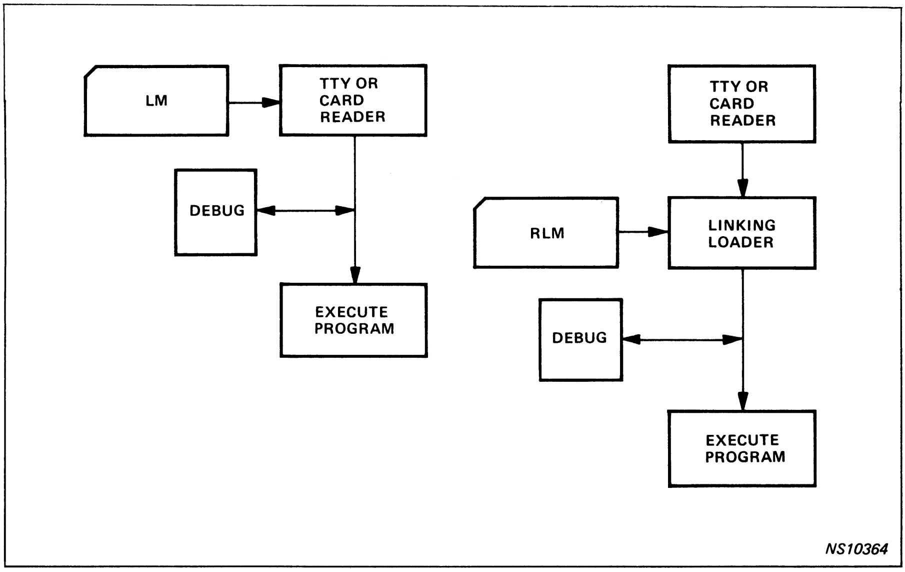


Figure 1-14 PACE Loading Sequence

CHAPTER 2
PACE AND FAMILY OF CHIPS

2.1 INTRODUCTION

The following paragraphs provide additional descriptive information and in-depth application data concerning the PACE microprocessor and family of chips. The PACE instruction set and addressing methods are detailed in appendix B. The instruction set description includes the instruction word bit configuration, assembler format, and instruction execution time formula for each instruction type. Applications data are provided for input/output control techniques, use of jump conditions and flags, interrupts, Cycle Extends, and DMA operation.

NOTE

Since this document was prepared during the final design phase of the PACE product line, some discrepancies may exist in the timing and electrical specifications presented with the following application information. For preliminary design purposes, refer to the latest data sheets to verify the timing and electrical parameters.

2.2 PACE MICROPROCESSOR

The PACE microprocessor provides the control and timing signals required for system or subsystem operation in addition to providing data manipulation and storage capabilities. The following paragraphs provide more information regarding the PACE microprocessor.

2.2.1 GENERAL DESCRIPTION

Data transfers between PACE (see figure 2-1) and memory or peripheral devices are effected over the 16-bit (D00-D15) parallel Input/Output Data Bus. The Input/Output Data Bus interfaces with the Instruction Register and the Operand Bus by way of the I/O Data Buffers. The Operand Bus also interfaces with seven registers (Temporary Registers 1 and 2, Program Counter, and AC0 through AC3) and a 10-word Stack. The seven registers and Stack are provided for data storage. Four of the registers (AC0 through AC3) are available to the programmer as general-purpose accumulators. The Program Counter contains the address of the next instruction. The contents of any selected register or the Stack are routed over the Operand Bus to the Arithmetic and Logic Unit (ALU) and Shifter. Resultant ALU and Shifter output is returned to the selected register or Stack, as appropriate, by way of the Result Bus. The ALU and Shifter, besides performing arithmetic operations, also sets status flags in accordance with the data length (8-bit or 16-bit) selected by the state of the BYTE Status Flag.

All status information is stored in the 16-bit Status and Control Flags Register. The Status and Control Flags Register contents can be loaded onto the Operand Bus for temporary storage on the Stack or in any accumulator for examination or modification of status information.

Instructions under execution by PACE are stored in the Instruction Register and are interpreted and executed by a microprogram stored in an on-chip ROM. Instruction execution time is determined by the instruction under execution, memory access time, and the external clock frequency.

2.2.2 EXTERNAL CLOCK REQUIREMENTS

The external clock signals (see figure 2-2) applied to PACE must consist of single-phase true and complement signals such as those produced by a System Timing Element (STE). PACE uses the external clock signals to generate internal multiphase clock signals that provide control timing for microprocessor operations.

2.2.3 DESCRIPTION OF HARDWIRED SIGNALS AND TIMING

PACE operations are controlled by software which *rewires* the PACE Control Logic in real time (at a speed determined by the external clock) to handle each microprocessor function in serial order. The clock and other signals that require hardwired connection to the PACE microprocessor are described in table 2-1. Pin assignments for the PACE CPU are shown in figure 2-3.

Instructions consist of four machine cycles or more, depending on the operations performed. The timing shown in figures 2-4, 2-5, and 2-6 represent the first machine cycle of the instruction being executed. The number of cycles for the instructions are given in appendix B.

NOTES

1. Positive logic convention is used throughout this manual. A logic '1' or high signal corresponds to a more-positive voltage level. A logic '0' or low signal corresponds to a more-negative voltage level. All signal names beginning with *'N'* or followed by an asterisk (✱) denote complemented signals that are asserted or activated by a logic '0'. Otherwise, signals are asserted by a logic '1'.
2. Bits are numbered from 00 to 15, right to left, with bit 00 representing the least significant bit.
3. The *X'* preceding a value denotes the hexadecimal numbering system.

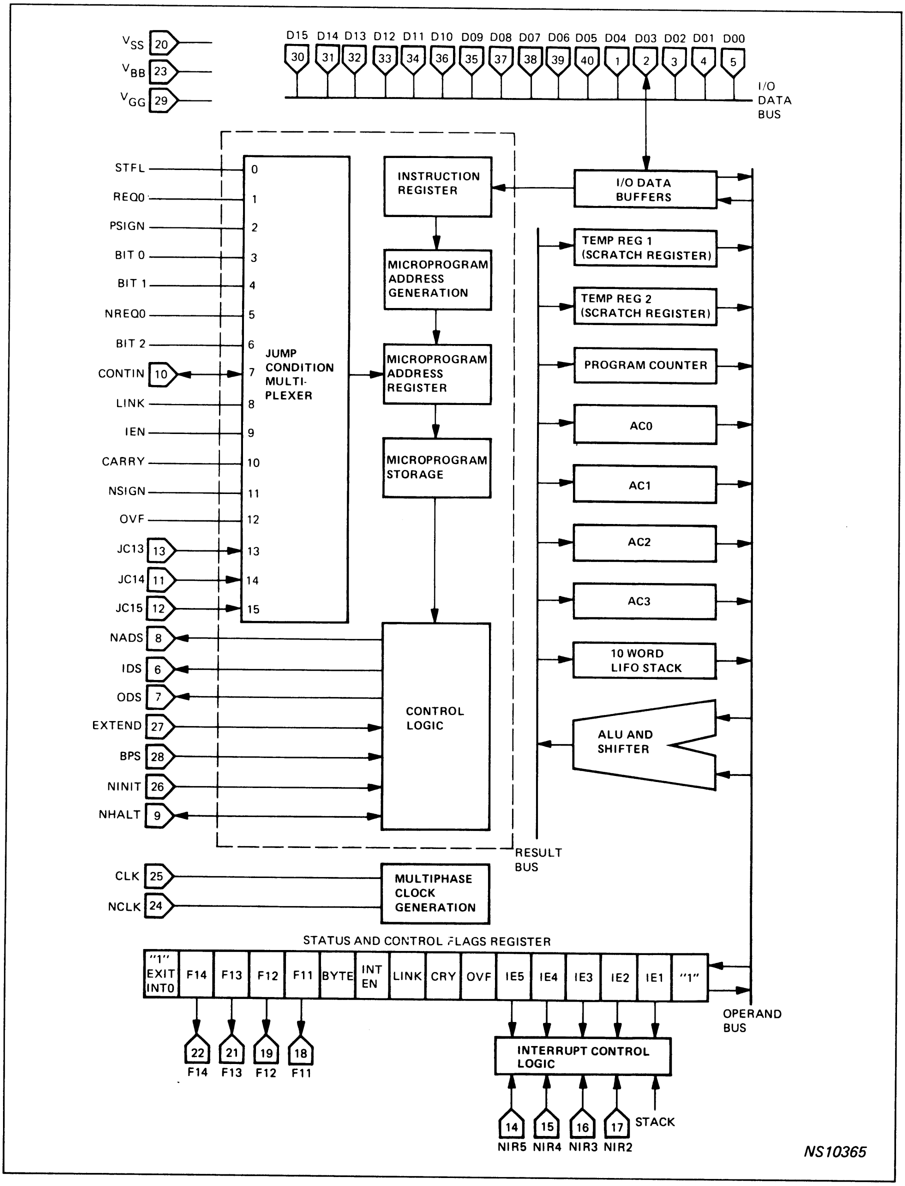
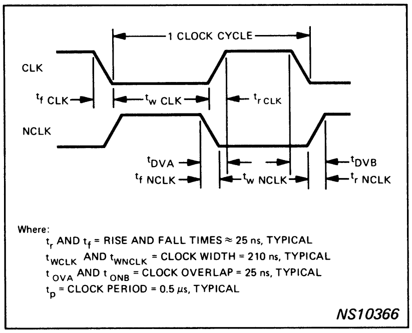


Figure 2-1. PACE Microprocessor Functional Block Diagram



Where:
t_r AND t_f = RISE AND FALL TIMES ≈ 25 ns, TYPICAL
t_{WCLK} AND t_{WNCLK} = CLOCK WIDTH = 210 ns, TYPICAL
t_{OVA} AND t_{ONB} = CLOCK OVERLAP = 25 ns, TYPICAL
t_p = CLOCK PERIOD = 0.5 μs, TYPICAL

Figure 2-2. External Clock Timing Parameters

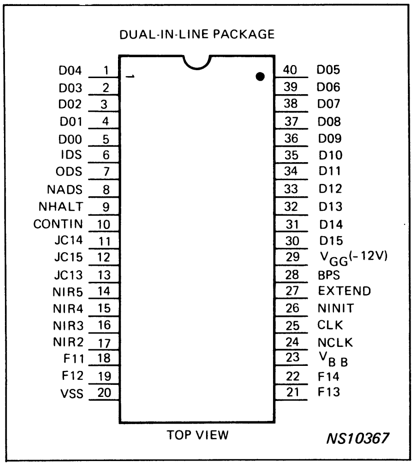


Figure 2-3. PACE Microprocessor Pin Assignments

Table 2-1. Descriptions of PACE Hardwired Signals

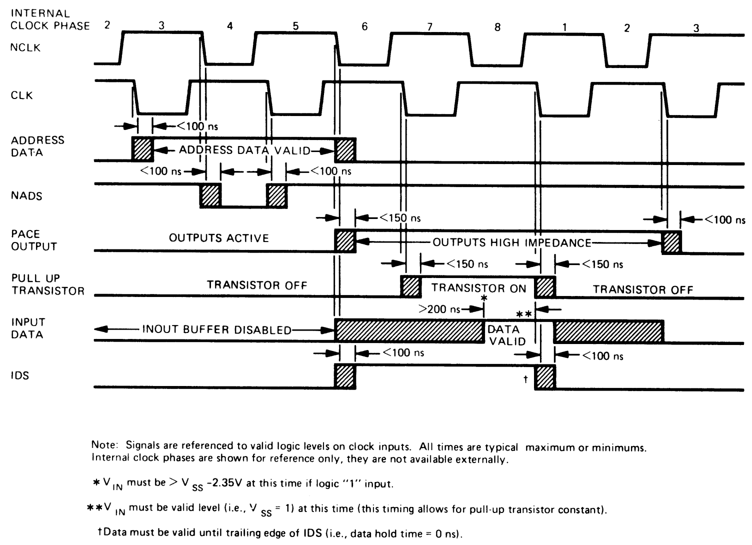
Signal Mnemonic/Name	Description
	NOTES 1. Some of the PACE microprocessor functions and buses referred to in the signal descriptions are illustrated in figure 2-1. 2. Refer to figures 2-4, 2-5, and 2-6 when signal descriptions discuss address output/data input timing, data output timing, and extending I/O signal timing, respectively. 3. Figure 2-7 illustrates user flag timing considerations.
CLK, NCLK/ True and complemented clock	External true and complemented clock inputs to PACE. Used in generation of PACE internal multiphase clock signals that provide timing control for internal PACE functions.
D00-D15/Data Bits 00-15	Input/output MOS Data Bus Lines.
IDS/Input Data Strobe	PACE output signal used to enable external devices so data can be placed on-line to PACE. IDS operation is as follows: 1. Following output of peripheral or memory address information from PACE (see figure 2-4), D00-D15 data line drivers (internal to PACE) assume high-impedance state and PACE Control Logic drives IDS Signal high. 2. IDS remains high for approximately 1.5 CLK periods. 3. Valid input data to PACE must be present on D00-D15 Input/Output Data Bus Lines when IDS is driven low again by Control Logic after approximately 1.5-CLK-period duration.

Table 2-1. Descriptions of PACE Hardwired Signals (Continued)

Signal Mnemonic/Name	Description
ODS/Output Data Strobe	PACE output signal used to enable external devices to accept data output from PACE. ODS operation is as follows: 1. Following output of peripheral or memory address information from PACE (see figure 2-5), data are placed on D00-D15 Input/Output Data Bus Lines by PACE. 2. At approximately the same time that data are placed on Input/Output Data Bus, ODS Signal is driven high by PACE Control Logic to signify that output data from PACE are available to memory or peripherals. 3. ODS remains high for approximately 1.5 CLK periods. 4. Output data remain on Input/Output Data Bus after ODS is driven low again by Control Logic after approximately 1.5-CLK-period duration. Thus, ODS trailing edge can be used to clock PACE output data into External Data Latch (ALE). ODS can also be used as read/write control signal for external RAM memory elements.
NADS/Address Data Strobe	PACE output signal used to clock address information from PACE into ALE. After address information (see figures 2-4 and 2-5) is placed on Input/Output Data Bus by PACE, NADS Signal is driven low for approximately 0.5 CLK period by PACE Control Logic. NADS is active in middle of approximately 1.5 CLK periods that address information is valid. Thus, either edge of NADS can be used to clock address information into ALE.
EXTEND/Extended Data Transfer	PACE input signal used to temporarily increase time duration of data input/output transfers to accommodate accessing of slow memories or peripherals without altering CLK frequency. EXTEND Signal must be driven high at beginning of ODS or IDS Signal (see figure 2-6). If EXTEND is held high as indicated in figure 2-6, data-transfer operation is extended by 1 CLK period. Holding EXTEND high for additional n clock periods increases data-transfer timing by n + 1 clock periods.
NINIT/Initialize	PACE input signal that initializes microprocessor functions. When NINIT is low, PACE operation is suspended and all PACE strobe signals (IDS, ODS, NADS, and so forth) are set to inactive state. After NINIT completes low-to-high transition, the following conditions are effected: 1. PACE Program Counter contents are set to zero. 2. Internal Stack Pointer (indicates last Stack level accessed) is cleared. 3. All flags and interrupt enables are set low except Level-0 Interrupt Enable which is set high. All other registers contain an arbitrary value.
NHALT/Control Panel Halt	PACE Control Logic input/output signal used for nonmaskable Level-0 Interrupt, microprocessor stall, and programmed HALT indicator output. When NHALT is applied as low input, microprocessor operation halts after completing execution of current instruction. When Halt Instruction is executed, NHALT Line is driven low by PACE Control Logic for a 7/8 duty cycle. Microprocessor can be stalled by using external open-collector driver to hold NHALT Line low for desired time duration, thereby overriding NHALT output buffer on PACE chip.

Table 2-1. Descriptions of PACE Hardwired Signals (Continued)

Signal Mnemonic/Name	Description
CONTIN/Continue Jump Condition	PACE Jump Condition Multiplexer input/output signal used to sense external signal through BOC Instruction. Also used to restore microprocessor operation from suspended state or cause subroutine branch to Level-0 Interrupt Service Routine (generally used to implement Control Panel functions). Driving CONTIN Input high for 4 CLK periods, minimum, causes halted microprocessor to resume operation. As output, CONTIN is driven low for approximately 3 clock periods by PACE Jump Condition Multiplexer to acknowledge that microprocessor operation is stalled. CONTIN Line must be pulsed to terminate Halt Instruction.
BPS/Base Page Select	Input signal to PACE Control Logic that enables one of two base-page addressing schemes to be selected. When BPS is low, first 256 words of memory constitute base page (page zero). When BPS is high, first 128 memory words and last 128 memory words constitute base page.
JC13, 14, 15/Jump Conditions 13, 14, and 15	User-specified branch-condition inputs to PACE Jump Condition Multiplexer. Some possible uses are testing system status and receiving serial data. When JC13, 14, or 15 is high, PACE Branch-On Condition Instruction effects program branch if Jump Condition Input is true.
F11, 12, 13, 14/Flags 11, 12, 13, and 14	PACE Status and Control Flags Register general-purpose control flag outputs. F11-14 may be used for direct control of system functions or serial data output. Individual flags may be set by PACE Set Flag Instruction and pulsed or reset by Pulse Flag Instruction (see figure 2-7). Push Flag and Pull Flag Instructions permit contents of Status and Control Flags Register to be saved on Stack during Interrupt Service Routine or subroutine execution, and then restored.
NIR2, 3, 4, 5/Interrupt Requests 2, 3, 4, and 5	Inputs to PACE Interrupt Control Logic. When NIR2, 3, 4, or 5 Input is low for 1 CLK period, minimum, corresponding internal Interrupt Request Latch is set. NOTE *Use of Interrupts* paragraph later in this chapter provides more comprehensive information concerning interrupt servicing.
V_{BB}	PACE input substrate voltage requirement derived from +5-volt and −12-volt supplies by STE.
V_{GG} (-12V)	PACE input power requirement.
V_{SS} (+5V)	PACE input power requirement.



Note: Signals are referenced to valid logic levels on clock inputs. All times are typical maximum or minimums. Internal clock phases are shown for reference only, they are not available externally.

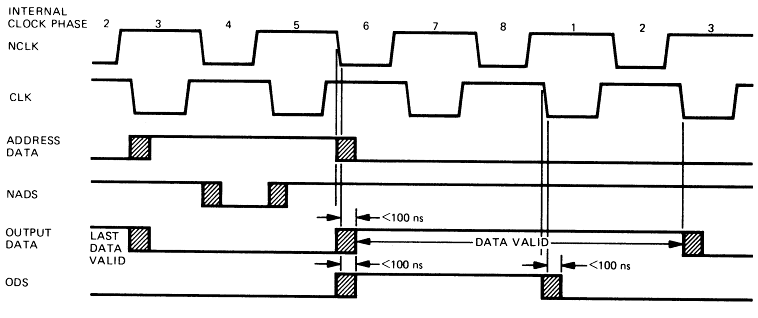
* V_{IN} must be > V_{SS} -2.35V at this time if logic "1" input.

**V_{IN} must be valid level (i.e., V_{SS} = 1) at this time (this timing allows for pull-up transistor constant).

†Data must be valid until trailing edge of IDS (i.e., data hold time = 0 ns).

NS10368

Figure 2-4. Address Output and Data Input Timing Diagram



NS10369

Figure 2-5. Data Output Timing Diagram

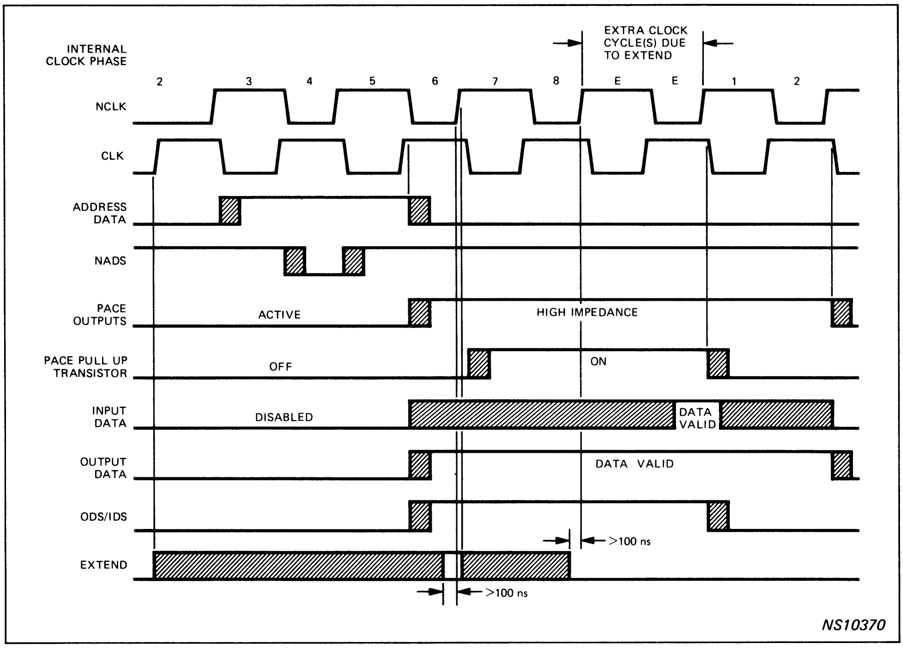


Figure 2-6. Extend I/O Signal Timing Diagram

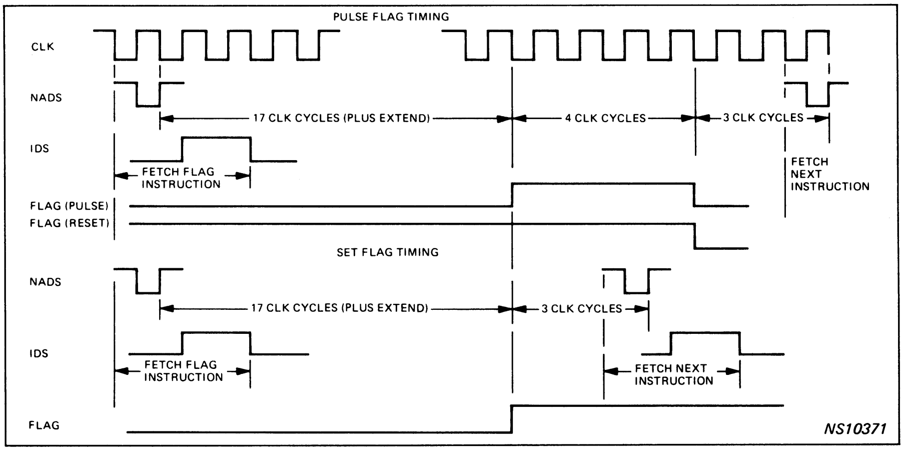


Figure 2-7. Pulse and Set Flag Timing Diagram

Table 2-2. Descriptions of Status and Control Flags

Register Bit	Flag Name	Description	Flag Code (fc)
0	High ('1')	Bit 0 is not used and is always in logic '1' state. Referencing bit 0 with SFLG or PFLG Instruction has no effect. (May be used as NOP Instruction.)	0000
1 2 3 4 5	IE1 IE2 IE3 IE4 IE5	Flags IE1 through IE5 serve as Interrupt Enable Flags for five of six PACE interrupt levels. If Interrupt Enable is high and associated Interrupt Request occurs, microprocessor executes Interrupt Service Routine. If Interrupt Enable is low, associated Interrupt Request is ignored.	0001 0010 0011 0100 0101
6	OVF	Overflow Flag is set to state of twos-complement arithmetic overflow by arithmetic instructions. Overflow Flag is set high if sign bits (most significant bit) of two operands are identical and sign bit of result is different from sign bit of operands. If A, B, and R are sign bits of operands and result, then Overflow Flag is set according to equation $\mathrm{OVF} = (\bar{A} \cdot \bar{B} \cdot R) + (A \cdot B \cdot \bar{R})$ Sign bit is most significant bit for data length selected; thus, if data length is 8 bits, then bit 7 is sign bit; if data length is 16, then bit 15 is sign bit. State of OVF Flag is affected by instructions ADD, DECA, SUBB, RADD, and RADC.	0110
7	CRY	Carry Flag is set to state of binary or decimal carry output of adder by arithmetic instructions. Carry output is derived from most significant bit for data length specified by BYTE Flag. State of CRY Flag is affected by instructions ADD, DECA, SUBB, RADD, and RADC.	0111
8	LINK	Link Flag is included in shift and rotate operations as specified by Shift and Rotate Instructions. Link Flag is unaffected if not selected.	1000
9	IEN	Master Interrupt Enable Flag simultaneously inhibits all five of lowest-priority interrupt levels. No Interrupt Request is serviced unless individual Interrupt Enable Flag for associated Interrupt Request and master Interrupt Enable Flag are high. IEN Flag is set low every time any interrupt (except Level-0) is serviced. IEN Flag is set high by execution of Return To Interrupt Instruction (RTI).	1001
10	BYTE	BYTE Flag selects 8-bit data length when high and 16-bit data length when low.	1010
11 12 13 14	F11 F12 F13 F14	Flags 11 through 14 are general-purpose control flags. Flags 11 through 14 drive PACE output pins and may be used to directly control system functions.	1011 1100 1101 1110
15	High ('1')	Bit 15 is not functional and is always in logic '1' state. Addressing bit 15 with SFLG or PFLG Instruction sets the Level-0 Interrupt Enable high. The Level-0 Interrupt is described in the *Use of Interrupts* paragraph later in this chapter.	1111

Table 2-3. Summary of Direct Addressing Modes

xr Field	Addressing Mode	Effective Address
00	Base-page	EA = disp
01	Program-Counter-relative	EA = disp + (PC)
10	AC2-relative (indexed)	EA = disp + (AC2)
11	AC3-relative (indexed)	EA = disp + (AC3)

NOTES:

1. For base-page addressing, disp is positive and in range of 000 to 255 when BPS is low (0); or disp is signed number in range of -128 to +127 when BPS is high (1).
2. PC contains value one greater than address of current instruction.
3. For relative addressing, disp range is -128 to +127.

2.3.1.2 Indirect Addressing

Indirect addressing consists of first establishing an address in the same manner as direct addressing (by either the base-page, PC-relative, or indexed mode). The contents of the memory location at the selected address then are used as the operand address. Figure 2-14 illustrates indirect addressing.

NOTE

The memory addressing modes also are used for peripheral I/O operations. Address space must be divided between memory and I/O devices.

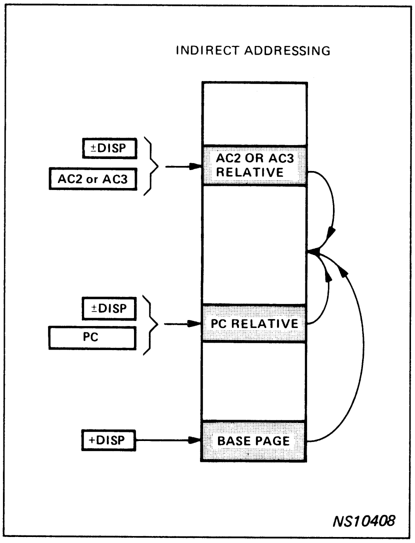


Figure 2-14. Indirect Memory Addressing

For indexed addressing, Accumulators AC2 and AC3 are used as 16-bit memory pointers. If Accumulators AC2 and AC3 are loaded from the 8-bit memory, the high-order 8 bits in the accumulators can be set equal to the sign of the low-order 8 bits by using the Load With Sign Extended Instruction (LSEX). Thus, a 16-bit twos-complement number results.

The Load With Sign Extended Instruction also can be used to set the state of the eight high-order data bits during 8-bit data transfers from peripherals. Alternatively, user-generated software can use Shift Instructions to set the eight high-order data bits to zero. The Shift and Rotate Instruction group (SHL, SHR, ROL, ROR) operates on the low-order 8 bits only and sets the high-order 8 bits to zero when the BYTE Status Flag is set for the 8-bit data-handling mode.

The Immediate Instructions (LI, CAI, AISZ) provide 16-bit, twos-complement data inputs. When working with 8-bit data, the high-order 8 bits usually can be ignored. If required, the high-order 8 bits can be cleared by using a Shift Instruction.

The Branch and Skip Instructions are modified to account for the 8-bit data length. Thus, the REQ0 and NREQ0 conditions are affected only by the low-order 8 bits. The PSIGN and NSIGN Signals indicate the sign of the low-order 8 bits. The Skip Instructions (SKNE, SKG, SKAZ, ISZ, DSZ) test only the low-order 8 bits. Thus, if a Skip Instruction compares 8-bit accumulator data with a 16-bit program memory word, the contents of the high-order 8 bits of both words are ignored. The Add Immediate, Skip if Zero Instruction (AISZ) is the only instruction that tests the entire 16-bit result when 8-bit data handling is selected. Therefore, the AISZ Instruction can be used to increment the index accumulators (AC2, AC3) without skipping every time the low-order 8 bits are zero. Consequently, the sign of 8-bit numbers must be extended by using the Load With Sign Extended Instruction to properly detect zero when using the AISZ Instruction for 8-bit data.

Since the Overflow and Carry Flags are modified by arithmetic instructions, the eight low-order data bits determine the state of the Overflow and Carry Flags when the 8-bit data length is selected. That is, the Carry Flag is set if a carry is generated by the low-order 8 bits and the Overflow Flag is set when an arithmetic overflow occurs in the low-order 8 bits.

The Link Flag is affected by Shift and Rotate Instructions. The Link Flag is set by data shifted out of the low-order 8 bits when the 8-bit data length is selected.

Working with 8-bit data and 16-bit instructions sometimes necessitates performing arithmetic operations by using a 16-bit operand from the program memory and an 8-bit operand from the data memory. If the result is to be treated as 8-bit data, no special considerations are required. However, if the result is to be treated as 16-bit data, the sign of the 8-bit operand first must be extended by using the Load With Sign Extended Instruction. Also, the carry, overflow, and conditional branch signals that are only a function of the low-order 8 bits should not be used. Alternatively, the BYTE Flag temporarily may be set low for 16-bit data handling to accommodate the signals changed by the 8-bit data-handling mode.

The previously mentioned factors make the use of PACE in 8-bit applications convenient while still providing the advantages of a 16-bit instruction set. (Data lengths other than 8 bits or 16 bits also may be used when special external hardware is provided.)

2.5.2.2 16-bit Interfacing

No special considerations are necessary for 16-bit interfacing except to ascertain that the peripheral and memory data bit lines are connected to the appropriate PACE data bit lines (that is, D00 is the least significant bit and D15 is the most significant bit). Also, the BYTE Status Flag must be set low for proper 16-bit data operations.

2.5.2.3 BCD Data

The PACE microprocessor is capable of adding four-digit-per-word BCD data with the Decimal Add Instruction (DECA) or two digits per word if BYTE equals 1. Consequently, no BCD-to-binary conversion is required. In appendix B, table B-4 provides a decimal addition program example that adds two 16-digit BCD strings using the DECA Instruction.

2.5.2.4 Serial Input

Serial interfaces to PACE can be provided by using a single-bit line of the address/data bus for the interface. Another method, which may be preferable for use in systems containing only a few peripherals, is to use a jump condition (JC13, JC14, or JC15) or interrupt input (NIR2 through NIR5) to service serial data. Using a jump condition or interrupt input avoids the need for address decoding and reduces the number of interface lines. The serial input data are applied to the selected interrupt or jump condition input. The state of the jump condition inputs then can be determined by instructions in the user-generated software.

2.5.2.5 Serial Output

As with serial input, a single-bit line of the address/data bus can be used for the interface. However, one of the user flag outputs (F11 through F14) may prove more effective for some system applications. The user flags can be set or cleared by using the Set Flag or Pulse Flag Instructions in the user-generated software. The use of jump condition or interrupt inputs and flag outputs is particularly well suited for asynchronous serial devices, such as a teletypewriter, since only one transmit and one receive line are involved.

2.5.3 USE OF JUMP CONDITIONS AND FLAGS

The PACE microprocessor contains a Jump Condition Multiplexer that samples the 16 jump conditions listed and described in appendix B, table B-3. The Branch-On Condition Instruction (BOC) tests the Jump Condition Multiplexer Output. If the condition for branching (selected by the condition code of the BOC Instruction) is active, a branch occurs; otherwise the next sequential instruction is executed.

The CONTIN Jump Condition is used by the HALT Instruction. If a Halt Instruction is executed, the microprocessor NHALT Output is driven low to indicate that microprocessor activity is suspended until the CONTIN Input is pulsed. While PACE operation is suspended, the NHALT Output Line has a 7/8 duty cycle; that is, every eighth clock phase, the NHALT Output goes high. The NHALT 7/8 duty cycle must be accounted for if the output is used as a logic signal but is of little concern if the output drives only a halt indicator. The NHALT Output goes high after the Halt Instruction is terminated by pulsing the CONTIN Input. The CONTIN Input must go high for four clock cycles, minimum, for PACE operation to resume.

The three unassigned jump condition inputs (JC13, JC14, and JC15) are for user purposes and may be implemented as required by the application.

The 14 status and control flags provided by PACE are listed and described in appendix B, table B-8. As previously described, the user flags (F11 through F14) and user jump conditions can be used for serial data input/output. In some cases, additional flags may be required for control purposes. The additional flags can be obtained conveniently by using a DM9334 8-bit addressable latch. An unused address bit or combination of bits may be used to enable the latch. Three bits can be used to address one of eight flags and another bit can specify set or reset as illustrated in figure 2-35. A Store Instruction may be used to output the address (data output is ignored).

In a similar manner, a multiplexer and latch can be used to expand user jump conditions. The latch is loaded from the address bus, if enabled by an unused address code, and selects a Multiplexer input to one of the user jump conditions.

2.5.4 USE OF INTERRUPTS

The PACE microprocessor provides a six-level priority interrupt structure. Each level is provided with an individual Interrupt Enable as shown in figure 2-36. A master Interrupt Enable (IEN) is provided for all five lower-priority levels at once. The master IEN is an input to the PACE Jump Condition Multiplexer. The state of Interrupt is tested by PACE during the Instruction Fetch Routine (internal to PACE) that is executed after completion of each instruction. Thus, if Interrupt is high, the interrupt is automatically serviced.

Negative-true Interrupt Request Inputs (NIR2 through NIR5) are provided to allow several interrupts to be wire-ORed to each input. When an Interrupt Request occurs, the associated Interrupt Request Latch (IR1 through IR5) is set if the corresponding Interrupt Enable Input is true. Since the Interrupt Request Latch can be set by any pulse exceeding one clock period, narrow timing or control pulses can be captured. If IEN is high, then an interrupt is generated and acknowledged after completing the current instruction.

During the interrupt sequence, an address is provided by the output from the priority encoder. The address is used

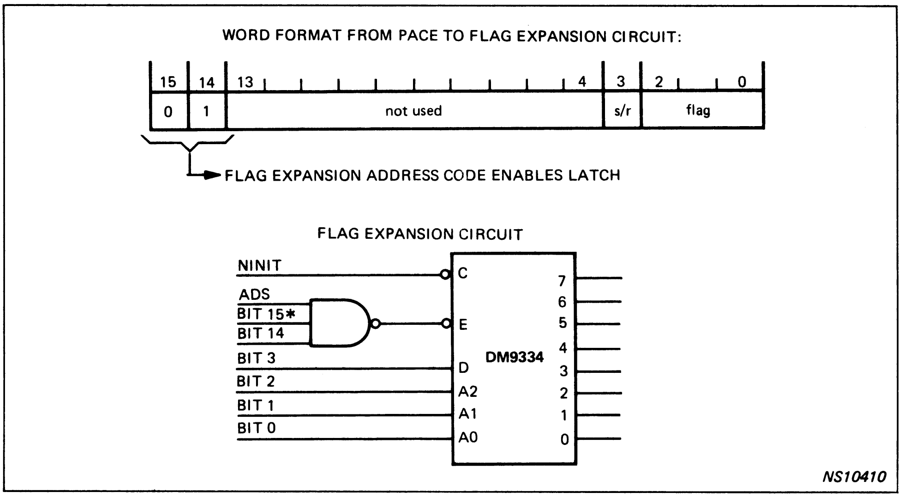


Figure 2-35. One Possible Circuit and Word Format for Obtaining Additional Flags

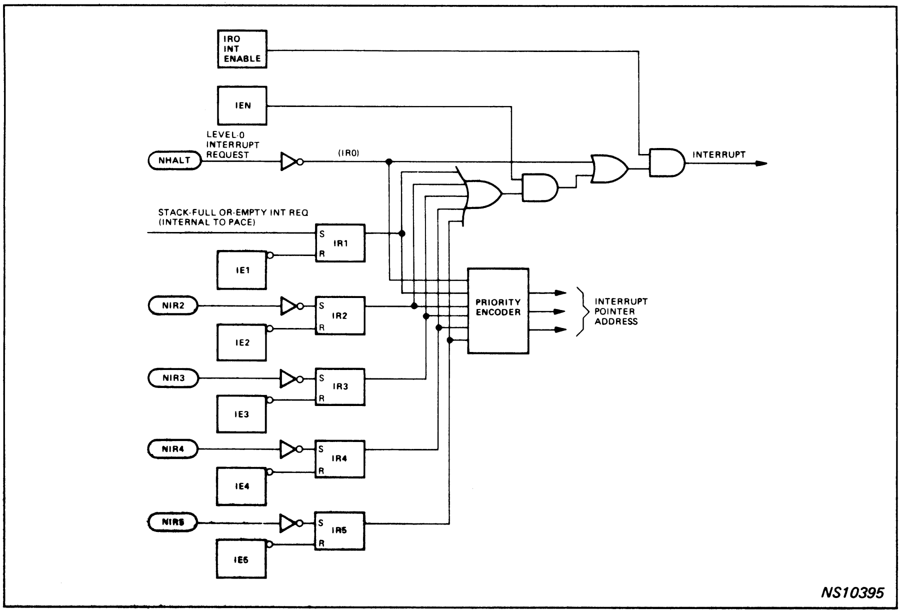


Figure 2-36. PACE Interrupt System

to access the Interrupt Pointer for the highest-priority Interrupt Request (IR0 is highest priority; IR5 is lowest priority). The Interrupt Pointers are stored in memory locations 2 through 8 (see table 2-5) for Interrupt Requests 1 through 5 and 0, respectively. The Interrupt Pointer specifies the starting address of the Interrupt Service Routine for the particular interrupt level, except in the case of the Level-0 Interrupt (IR0), which is used primarily for alarm interrupts and Control Panel implementation.

Table 2-5. Locations of Interrupt Pointers

Interrupt Pointer	Memory Location
Interrupt-0 Program	8
Interrupt-0 PC	7
Interrupt 5	6
Interrupt 4	5
Interrupt 3	4
Interrupt 2	3
Interrupt 1	2
Not Assigned	1
Initialization Instruction	0

Before Interrupt Service Routine execution, the Program Counter contents are pushed onto the Stack and IEN is set low (false). This interrupt handling requires 14 microseconds (28 clock cycles). The Interrupt Service Routine may set IEN high (true) after turning off the Interrupt Enable for the interrupt level currently being serviced (or resetting the Interrupt Request). The Interrupt Enable Flags can be set by the Set Flag (SFLG) and reset by the Pulse Flag (PFLG) Instructions. If an Interrupt Enable Flag is set or reset, one more instruction is executed before the interrupt is enabled or disabled. The Return From Interrupt Instruction (RTI) also may be used to set IEN true. In this case, there is no delay and a pending interrupt takes effect immediately after execution of RTI.

It should be recognized that the function of the individual Interrupt Enables IE1-IE5 is to *arm* or *disarm* the Interrupt Request Latch; whereas, the function of the Master Interrupt Enable (IEN) and Interrupt Enable IR0 is to *enable* or *disable* the latched Interrupt Request Lines.

Three types of external interrupts are likely to occur in PACE applications: short-duration (pulse) interrupts; long-duration resettable interrupts; and nonresettable interrupts. The short-duration interrupt exists for less than the interrupt response time and may be caused by a strobe pulse from a peripheral device or the occurrence of a high-speed transient condition. A short-duration interrupt must be latched to be recognized. Interrupts longer than the clock period are latched by the PACE Interrupt Request Latches. The Interrupt Service Routine must reset the Interrupt Request Latch by turning off the Interrupt Enable for the level being serviced. If the Interrupt Enable is left off, Interrupt Request pulses cannot set the Interrupt Request Latch.

Long-duration resettable interrupts last longer than the interrupt response time and may be reset by the Interrupt Service Routine. An example is a Buffer-full Interrupt by a peripheral device. The Interrupt Service Routine empties the buffer, removing the interrupt. A long-duration interrupt is ignored when Interrupt Enable is low but still generates an interrupt when Interrupt Enable is set true. In servicing long-duration interrupts, the Interrupt Request Latch must be cleared after the interrupt is reset by the Interrupt Service Routine.

Long-duration nonresettable interrupts last longer than the interrupt response time and are not reset by the Interrupt Service Routine. An example of a long-duration nonresettable interrupt is a photoelectric cell that detects the presence of an item on a conveyor. The signal produced by the photoelectric cell (or some other sensor) may last for a significant portion of a second. Setting the Interrupt Request Latch on the edge of the interrupt is desirable and may be accomplished using a simple RC circuit or single-shot to generate a pulse on the edge of the interrupt.

The interrupt response time for PACE is equal to the time to finish the current instruction at the time of the interrupt, plus the time to access the first instruction of the Interrupt Service Routine. Instruction execution times are given in appendix B, table B-2.

An example of an Interrupt Service Routine for Interrupt Level 3 is shown in table 2-6. Memory location 4 contains the address of the first instruction in the routine. When a Level-3 Interrupt occurs, the first instruction preserves the state of the flags on the Stack.

NOTE

IEN is set false by the interrupt prior to being saved on the Stack.

The flag data then are loaded into AC0 and all bits which are to be modified are masked out to zero. The desired bits then are set true by ORing with IESTAT. If the routine is interruptable, then IE3 is set to zero and IEN is set to one. The modified status word then is transferred from AC0 to the status register. The actual servicing of the interrupting device then takes place. At the end of the routine, the flags are restored and a Return Instruction is executed. If the interrupts are to be reenabled, the RTI Instruction must be used since RTI sets IEN true and restores the PC from the Stack.

A Stack Interrupt occurs when the Stack-empty or Stack-full condition exists. The Stack Interrupt consists of a pulse applied to the set input of Interrupt Request Latch 1 (see figure 2-36). The pulse sets the latch if the IEN1 Flag is true; otherwise, the pulse is ignored. The Stack is implemented with a RAM and a Pointer which can access RAM locations 0 to 9. A pulse occurs when the Stack Pointer is at 0 (one entry on Stack), and a Read-Stack Operation occurs to empty the Stack. A pulse also occurs when the Stack Pointer is equal to 7 (eight entries on Stack), and a

Write-Stack Operation occurs to fill the ninth word and leave one word empty to be used by the interrupt. When a Stack Interrupt occurs, the Stack condition can be determined by using the Stack-full Jump Condition (STFL).

With the interrupt scheme described, an interrupt does not occur at initialize but does occur every time the Stack becomes empty. If the Stack is to be extended into memory, a Stack-empty Interrupt is required but may be inhibited by turning off IEN1 in other cases. In order to prevent a Stack Interrupt when both hardware and software Stacks become empty, a dummy word may be pushed on the Stack by the Initialize Routine.

The Level-0 Interrupt is not maskable under program control. A Level-0 Interrupt may be used for alarm conditions such as a power failure or for implementing a software-based Control Panel such as that contained in the IPC-16P Microprocessor Development System. A Level-0 Interrupt can be generated by using the PACE NHALT and CONTIN Signal inputs. Figure 2-37 illustrates the relative timing for Level-0 Interrupt generation. As is shown in figure 2-37, the CONTIN Signal can be used as an interrupt acknowledge signal. For cases where an interrupt acknowledge is not required, or where the CONTIN Signal is used as a sense input to the program, the CONTIN Signal can be held continuously low. While holding the CONTIN Signal continuously low, the NHALT Signal must be driven low for the duration of the longest instruction execution time plus eleven clock cycles to guarantee that a Level-0 Interrupt occurs.

After the NHALT Signal returns to a high state, the Level-0 Interrupt is serviced. Servicing consists of first setting the Level-0 Interrupt Enable (IR0 INT ENABLE in figure 2-36) low to lock out all other possible interrupts. Next, the PACE Program Counter contents are stored in the location specified by memory location 7 (see table 2-5). Then, the instruction at memory location 8 is executed. Storing the Program Counter contents in a memory location instead of on the Stack prevents generation of a Stack-full Interrupt.

To return from a Level-0 Interrupt, the PFLG15 or SFLG15 Instruction is executed to set the Level-0 Interrupt Enable Output high after execution of one additional instruction. The additional instruction is typically a JMP@ through memory location 7, which contains the Program Counter contents. Thus, a proper return to the interrupted program can be effected. During initialization, the Level-0 Interrupt Enable is set high.

In some applications, expansion of the user interrupts by providing several interrupts on a single input may be desirable. Several interrupts can be provided on a single input by using open-collector gates for a wired-OR input and polling all the devices on a given level to discover the origin of the interrupt. However, in some applications, the polling technique may take excessive time. In such cases, use of the DM9318 Priority Encoder is recommended to encode the number of the highest-priority interrupting device. A single instruction in the Interrupt Service Routine then can be used to read the number of the interrupting device over the data bus. The use of the DM9318 Priority Encoder is shown in figure 2-38. The use of a DM8131 Comparator with latched output to detect the appropriate peripheral address also is illustrated in figure 2-38.

NOTE

Status register masking is necessary only when Interrupt Enable status is to be modified to allow higher priority devices to interrupt. Pushing the status register onto the Stack is necessary only if the routine alters the contents of the status register.

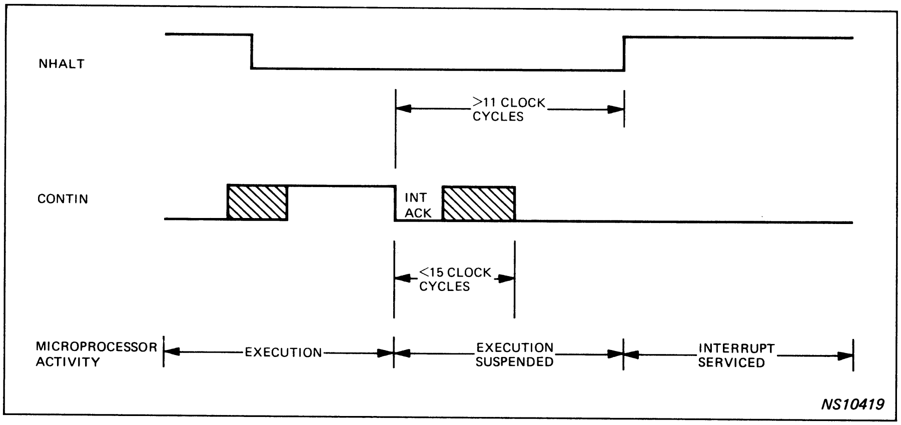


Figure 2-37. Relative Timing for Level-0 Interrupt Generation

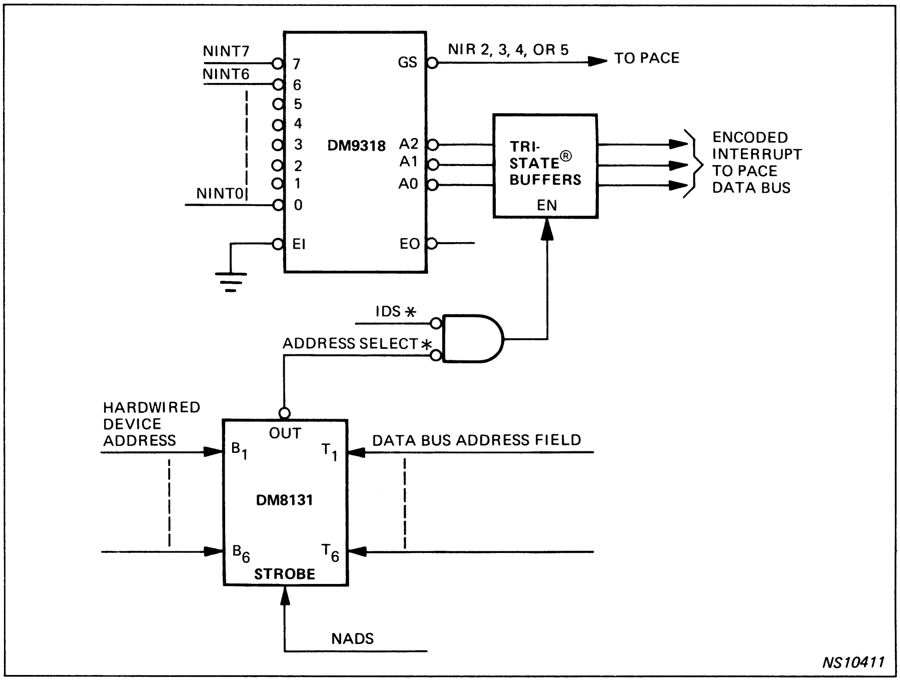


Figure 2-38. Use of DM9318 Priority Encoder and DM8131 Comparator for Interrupt Expansion and Detection

2.5.5 IMPLEMENTATION OF INITIALIZE AND CYCLE EXTEND SIGNALS

The following paragraphs describe the use of Initialize and Cycle Extend Signals. The Initialize Signal is used to initialize the PACE microprocessor and other system elements during the power-up condition or at any other desirable time. The Cycle Extend Signal can be used to increase the I/O cycle time by multiples of the clock period.

2.5.5.1 Initialize

The PACE Initialize Signal (NINIT) input may be used at any time to initialize the microprocessor and should always be used during system power-up. Application of a low NINIT Signal clears the Stack Pointer, sets the flags to zero, sets the Level-0 Interrupt Enable true, and sets the Program Counter contents to zero. The accumulators assume an arbitrary state. The NADS, IDS and ODS data strobes are set false. Thus, if system data are to be preserved during initialization, NINIT should be inhibited during data output cycles.

The PACE data strobes (NADS, ODS, and IDS) are inactive for 16 clock cycles after the trailing edge of the NINIT Signal occurs. After the 16 clock cycles, the first NADS Signal occurs and the first instruction is accessed from memory location X'0000, unless a Level-0 Interrupt (Control Panel Interrupt) is present. All other interrupt levels are disabled. Figure 2-24 shows a circuit that can be used for generating an INIT Signal during system power-up. The resultant output must be inverted to provide NINIT.

2.5.5.2 Cycle Extend

The PACE Extend Input can be used to increase I/O cycle times by multiples of the clock period (see figure 2-6). To extend I/O cycles, a circuit like that illustrated in figure 2-39 can be used. The NADS Signal is used to initiate the Extend pulse while the TTL CLK from the System TTL Timing and Control Bus is used to count out the desired number of clock cycles. The circuit shown in figure 2-39 provides an extend of one clock cycle for data-input operations, as might be required for an MOS ROM.

The Extend Input also can be used for suspending the microprocessor activity to provide a *cycle-steal* for Direct Memory Accessing (DMA). Refer to the *Implementation of DMA* paragraph for more information.

Table 2-6. **Interrupt Service Routine Example**

Assembly Code			Explanation
	. = 4		Set location counter equal to 4.
	.WORD	ISERV3	Pointer to service routine.
	. = 500		Set location counter equal to 500.
ISERV3:	PUSHF		Save flags on Stack.
	PUSH	AC0	Save AC0.
	CFR	AC0	Move flags to AC0.
	AND	AC0, MASK	Mask out old Interrupt Enable status.
	OR	AC0, IESTAT	OR in new Interrupt Enable status.
	CRF	AC0	Store in Flag Register.
	.		.
	.		.
	.		.
	.		Interrupt Service Routine.
	.		.
	.		.
	.		.
	.		.
	.		.
INTXIT:	PULL	AC0	Restore AC0.
	PULLF		Restore flags.
	RTI		Return to interrupted routine.
MASK:	.WORD	(mask data)	
IESTAT:	.WORD	(Interrupt Enable status data)	

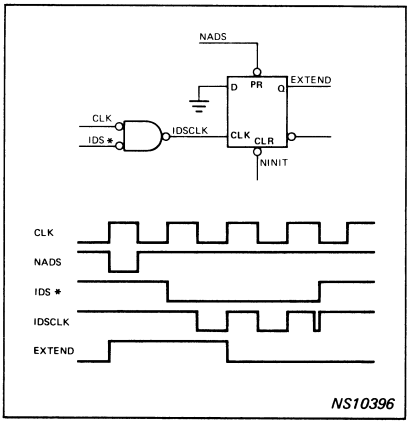


Figure 2-39. Circuit and Timing Diagram for One Clock Cycle Extend

2.5.6 IMPLEMENTATION OF DMA

The PACE ODS and IDS Signals should be tested by the DMA Bus Controller for active states. If ODS and IDS are inactive and a priority bus request is present from a peripheral, then NADS should be tested for 1.5 clock cycles, minimum, while generating an Extend. If NADS is not active within 1.5 clock cycles, then the DMA Bus Controller can generate a Master Bus Grant (MBG). If IDS, ODS, or NADS is active within 1.5 clock cycles, the Extend Signal should be removed until the PACE I/O cycle is complete. When generated, the Extend Signal suspends the PACE Microprocessor operation until the Extend Signal goes low. After the Extend Signal goes high and no NADS occurs for 1.5 clock cycles, the MBG Signal should go high. During the high MBG Signal, BNADS should go low. Then, the BODS/BIDS Signals should go high, as required for output/input operations between peripherals and memory. The cycle should be completed before the Extend and MBG Signals terminate. Approximately 0.5 clock cycle after BODS/BIDS terminates, the bus cycle is ended. Care should be exercised not to extend the bus cycle beyond refresh requirements of the PACE microprocessor (see data sheet).

As an alternate method, microprocessor operation may be suspended by driving the PACE NHALT Input low with an external gate. The external gate output overrides the PACE output buffer. Microprocessor operation then is suspended after execution of the current instruction. The suspension may last for an indefinite period of time without loss of CPU status and may be terminated by use of the PACE CONTIN Input (properly sequenced with removal of the NHALT Input). The timing sequence for the NHALT and CONTIN Signals is shown in figure 2-42. The NHALT and CONTIN method for suspending PACE operation can be useful for DMA block data transfers which require full bus-throughput capacity.

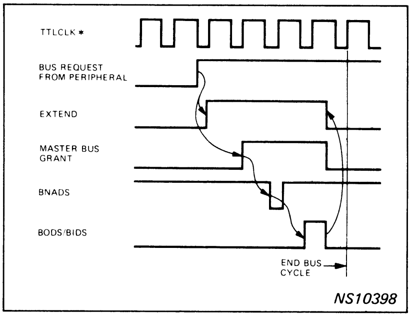


Figure 2-41. Timing Required from DMA Bus Controller

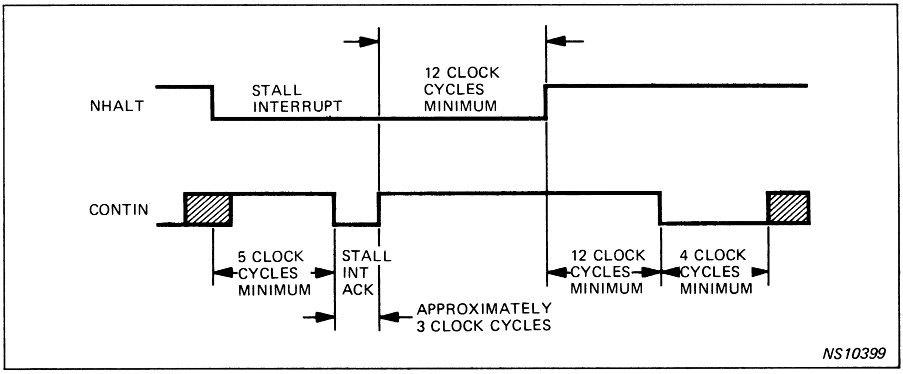


Figure 2-42. Timing Diagram for Externally Applied NHALT and CONTIN Signals

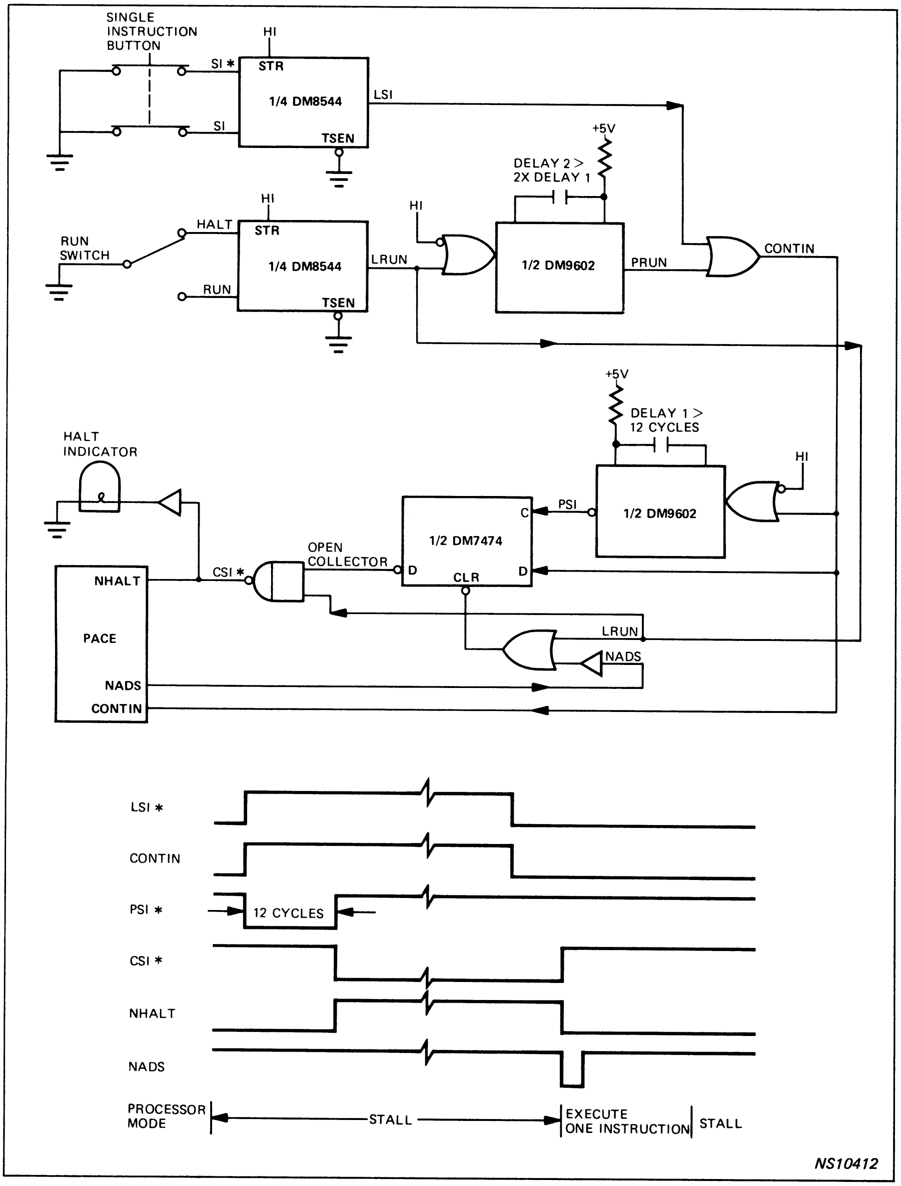


Figure 2-43. Control Panel Logic and Timing (Plus Time Line)

2.5.7 MINIMAL CONTROL PANEL

The previously described programmed halt and processor stall input allow the implementation of a simple Control Panel with a minimum of components. The Control Panel provides HALT, SINGLE INSTRUCTION, and RUN modes and displays data for a selected trap address or data and address for each single instruction executed.

The control logic and the associated timing are shown in figure 2-43. A one-shot is used in conjunction with the SINGLE INST Switch contact closure time (assumed greater than one-shot delay plus 12 clock cycles) to provide the proper single-instruction timing. The time between single-instruction closures is assumed to exceed the longest instruction execution time. The single-instruction sequence is terminated by generating a halt after the first NADS. (The halt must be generated in less than eight clock cycles after NADS.) The RUN mode is entered by generating the single-instruction timing and inhibiting the termination on ADS.

The data display logic is shown in figure 2-44. In the HALT mode, the address and data for each single instruction execution are displayed. In the RUN mode, switch-selected address and data are displayed each time the selected address occurs on the data bus.

While the capabilities of a simple Control Panel are limited and may not be suitable for the program development stage, a simple Control Panel is adequate in certain end applications. A very complete Control Panel facility is provided for program development purposes by the PACE Development System, IPC-16P, described in chapter 4.

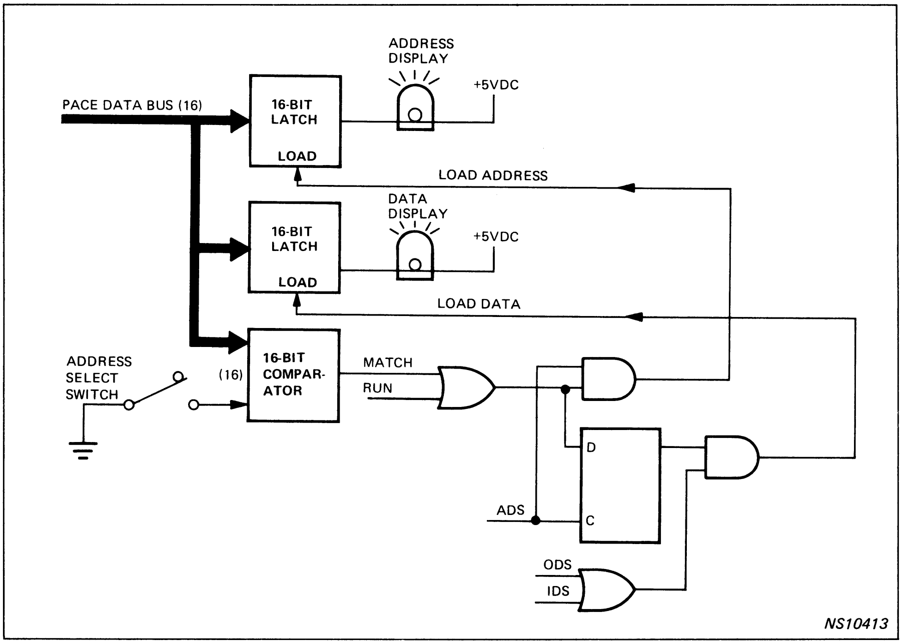


Figure 2-44. Panel Data Display Logic

APPENDIX A

APPLICATION INFORMATION

4.1 INTRODUCTION

In order to successfully apply any MOS/LSI microprocessor in a custom design, close attention must be paid to the electrical characteristics and timing details of the various control, timing, and data signals.

This chapter provides the practical information needed to use the PACE microprocessor as an integrated circuit component. The information presented in this chapter is supplementary to the information provided in previous chapters.

4.2 ELECTRICAL SPECIFICATIONS

Detailed electrical specifications are provided by the IPC-16A/500D PACE data sheet. A copy of the data sheet is contained in appendix A. The data sheet is provided for reference only, and the latest revision, published separately, always should be consulted for up-to-date corrections and supplementary information before finalizing a design.

4.3 CLOCK REQUIREMENTS

Clock inputs to the PACE chip are single-phase true and complement signals with full MOS voltage swings. The definitions of clock timing parameters are provided by figure 4-1.

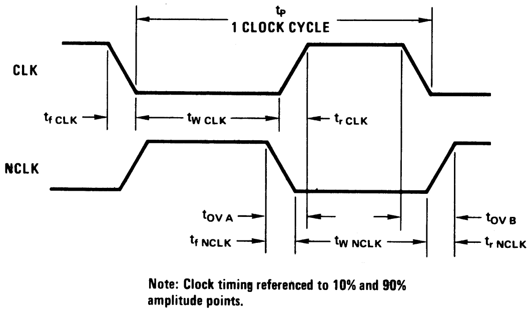


Figure 4-1. Clock Timing Parameters

5.4 USER GROUP

National Semiconductor sponsors COMPUTE (Club of Microprocessor Programmers, Users, and Technical Experts). This user group is dedicated to the world-wide distribution of your ideas and techniques relating to the use of microprocessors. Members of COMPUTE communicate on a regular basis by way of *The Bit-Bucket* newsletter published by National Semiconductor. In the newsletter, you will find everything from soup to nuts — even a user-submitted software library. So get involved with PACE and COMPUTE; they make an excellent partnership. You can meet the former by calling your nearest sales representative and the latter by writing to:

COMPUTE/470
National Semiconductor Corporation
2900 Semiconductor Drive
Santa Clara, California 95051
Telephone: (408) 732-5000, Extension 7183

Don't forget the */470* in the address. That's our mail stop, and your letter will be delayed (or worse yet, lost) without it.

The two clock inputs (CLK and NCLK) are used by the PACE chip to generate eight internal clock signals from every four external clock cycles. The eight internal clock signals are used for sequencing the internal logic circuits.

4.3.1 Clock Generation

Several approaches may be used for clock generation, depending on the system performance and cost objectives. The clock generator functions required are shown in figure 4-2.

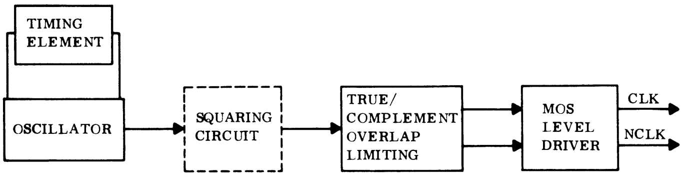


Figure 4-2. Clock Generator Functions Required

The oscillator (see figure 4-2) may be a simple RC-controlled pulse generator, or, if accurate frequency control is desired, a crystal-controlled oscillator can be used. For higher performance systems, a Squaring Circuit may be desirable to provide a 50-percent duty cycle waveform. True and complement waveforms then must be generated, and the amount of overlap must be within specification (see data sheet). The resultant signals then must be translated to the MOS voltage level required by PACE. The final clock generator outputs, after MOS voltage-level translation, are supplied to PACE as the CLK and NCLK Signals.

For many applications, the PACE System Timing Element, IPC-16A/502, provides a cost effective, single-package solution for all the functions indicated in figure 4-2 without sacrificing performance. The components listed in table 4-1 are also useful in configuring a variety of clock generators.

Table 4-1. Components Used For Clock Configurations

Component	Function
DM74LS124 Dual Oscillator	Crystal- or capacitor-controlled TTL oscillator. Second oscillator may be used for system timing, if required.
LM375 Oscillator	Crystal or LC oscillator with TTL output.
MH7803 Two-phase Oscillator/Clock Driver	Complete oscillator and clock driver in single package. Requires operation of PACE chip at reduced clock frequency (approximately 500 kHz).

Table 4-1. Components Used For Clock Configurations (Continued)

Component	Function
MH0026 Two-phase MOS Clock Driver	High-speed dual clock driver in 8-pin mini DIP.
MH0025 Two-phase MOS Clock Driver	Medium-speed, low-cost dual clock driver in 8-pin mini DIP.
MH0009 DC-coupled Two-phase MOS Clock Driver	High-speed dual clock driver. Provides TTL compatible, dc-coupled inputs, thereby eliminating need for input coupling capacitors.

4.3.2 Clock Frequency

The clock frequency may be chosen either to maximize the microprocessor execution speed or to simplify clock generator design/layout and memory interface timing. High-speed clock generation requires more components and increases crosstalk, overshoot, and power dissipation. In addition, high-speed clock generation requires the use of the EXTEND Signal to interface with slow memories. However, careful design and layout prevents problems and achieves higher performance.

Medium- or low-speed clock generation, having slow rise and fall times and a cycle time long enough to access the slowest memory without using the EXTEND Signal, simplifies design and layout but sacrifices performance. If the reduced performance level still is adequate for the application, the only loss is in excess performance for future expansion capability.

4.3.3 Clock Overlap

The clock overlap specifications (listed in the data sheet) are sufficient to allow true and complement signals to be used for the clock inputs. However, care should be exercised in selecting the clock circuit components to minimize delays which increase overlap.

A pair of cross-coupled NOR gates may be used (as shown in figure 4-3) to provide nonoverlapping inputs to the clock drivers. Two-phase nonoverlapping clocks may be used in systems where reduced clock cycle time is acceptable.

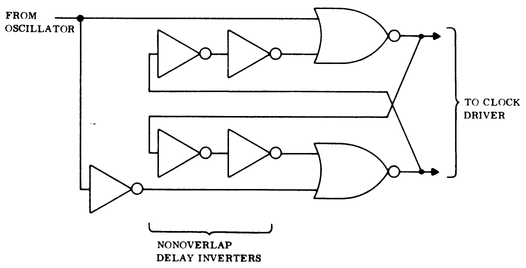
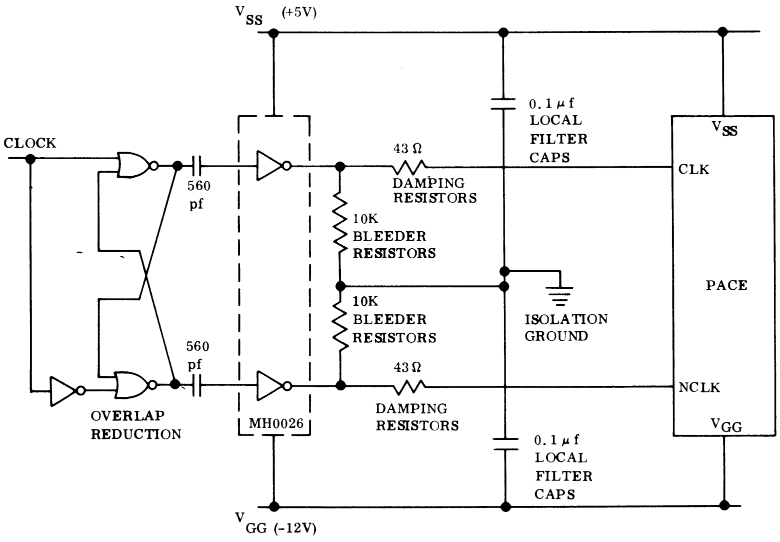


Figure 4-3 Circuit for Nonoverlapping Clock Signals

4.3.4 Clock Overshoot and Crosstalk

High-speed clock circuits tend to overshoot the supply voltage levels due to the inductance of the clock lines and the capacitive load internal to the PACE chip. If the maximum positive level of V_{SS} + 0.3 volts is exceeded, data internal to the PACE chip may be lost. In order to reduce overshoot, minimize the length of the clock lines and do not use fast rise- and fall-time clock drivers that are not required by the application. If high-speed drivers are used, series damping resistors should be provided to prevent overshoot. The damping resistance value should be determined empirically for any given circuit board layout and is likely to be in the range of 30 to 60 ohms. The capacitive load presented by the PACE chip varies with the internal clock phase and control functions. The damping resistance value should be chosen to prevent overshoot with the smallest possible resistance value rather than to excessively reduce rise and fall times with the largest resistance value.

Capacitive crosstalk from one clock line to the other may occur with high-speed clocks. Crosstalk can be reduced by isolating the clock lines with ground lines and by physically placing the series damping resistors close to the drivers (see figure 4-4). High-resistance bleeder resistors (connected from the driver outputs to ground) are desirable for keeping the clock driver output stage in the active (low-impedance) state. Local high-frequency filter capacitors should be provided from the clock driver to ground. Additional information on applying clock drivers is provided in National Semiconductor Application Note 76 "Applying Modern Clock Drivers To MOS Memories."

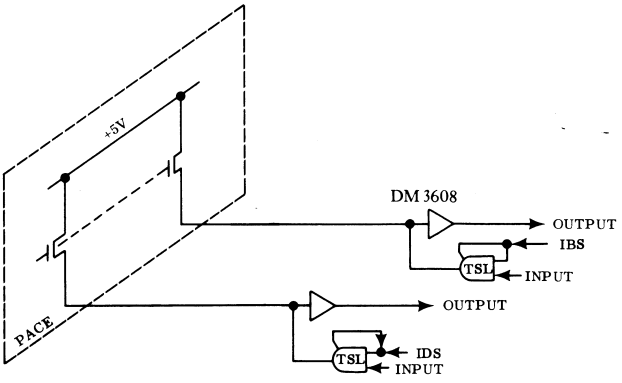


(VALUES ARE TYPICAL)

Figure 4-4. Use of Damping and Bleeder Resistors for Overshoot and Crosstalk Reduction

4.4 SIGNAL BUFFERING

The output buffer on the PACE data lines is an open-drain transistor with the source connected to V_{SS}. The open-drain transistor output buffer is designed to drive a current sense amplifier, as provided on the 8-bit PACE bidirectional transceiver element, IPC-16A/501, or the 6-bit DS3608 sense amplifier. The current sense amplifier must have TRI-STATE ® input capability, since the PACE data pins are bidirectional. A pulldown resistor can be used to allow the PACE chip to drive 74C CMOS or low-power TTL devices (see figure 4-5). For such an application, the pulldown resistor should be switched from -12 volts to +5 volts when IDS goes true to ensure an adequate logic '1' input signal to PACE. The pulldown resistor with switched supply buffering approach requires a slower operating speed than the sense amplifiers, since the data line capacitance must be given time to charge, whereas voltage remains constant when driving current sense amplifiers.



TSEN = TRI-STATE ENABLE

Figure 4-5. Use of Pulldown Resistors with Switched Supply to Drive CMOS or LPTTL

The NHALT and CONTIN PACE outputs are designed to drive low-power TTL or 74C CMOS directly. (The NHALT and CONTIN outputs also can drive sense amplifiers through a series diode.)

All inputs to PACE except BPS, CLK and NCLK are designed to be driven by TTL gates. On-chip pullup resistors are provided to insure the specified logic '1' level. If several devices are tied to a single input or I/O pin, the leakage currents should be kept to a minimum to avoid degrading the logic '1' level and thereby reducing noise margin. DM8097 or DM80L97 hex TRI-STATE ® buffers may be used for data input to bidirectional pins if the IPC-16A/501 bidirectional transceiver element is not used.

The BPS Signal is an MOS-level input with an internal pulldown transistor that causes the input to assume a logic '0' state if not connected.

4.5 DATA BUS STRUCTURES

The multiplexed data pins on PACE allow implementation of a fully multiplexed bus structure to minimize wiring costs. Alternatively, a variety of multiple bus structures may be implemented. Three different possibilities are shown in the diagrams of figure 4-6A, B, and C. Devices which are useful in implementing the illustrated, and other, bus structures include the following:

- DM8097/DM80L97/80C97 Hex TRI-STATE® Buffer
- DS3608 Hex TRI-STATE® Current Sense Amplifier
- DM8542 Quad TRI-STATE® I/O Register
- DM8131 or DM8160 6-bit Comparator
- DM8551 TRI-STATE® Quad D Flip-flop
- DM8833 TRI-STATE® Quad Transceiver

4.6 MEMORY

The PACE processor interfaces with a wide variety of memory devices, allowing an optimized choice for a given application. The EXTEND Signal may be used to extend the memory access time in increments of one clock cycle. In some applications, mapping a portion of the base-page addresses into another address range may be desirable so the base page can be shared among ROM, RAM and peripheral devices.

Table 4-2 lists some memory parts that may be useful in PACE applications.

Table 4-2. Memory Parts for PACE Applications

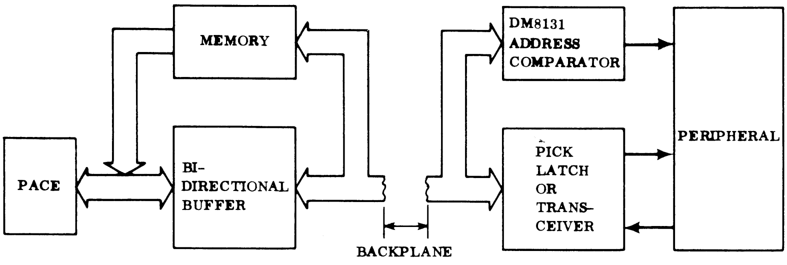
Type	Number	Description
RAM	MM5269	256 x 4-bit static N-channel RAM with address latches
	MM2102	1K x 1-bit static N-channel RAM
	MM1101	256 x 1-bit static P-channel RAM
	MM5262	2048 x 1-bit dynamic P-channel RAM
	MM74C200	256 x 1-bit CMOS RAM
	DM86L89	64 x 4-bit bipolar RAM

Table 4-2. Memory Parts for PACE Applications (Continued)

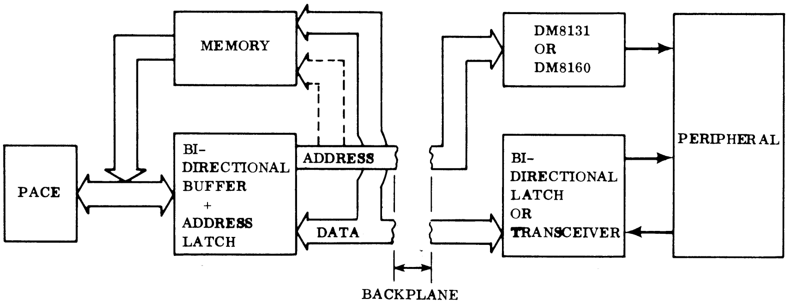
Type	Number	Description
	DM74200	256 x 1-bit bipoler RAM
	DM74204	1024 x 1-bit bipolar RAM
	DM74L89	16 x 4-bit bipolar RAM
PROM	DM8578	32 x 8-bit bipolar PROM
	DM8574	256 x 4-bit bipolar PROM
	MM5203	256 x 8-bit MOS PROM
	MM5204	512 x 8-bit MOS PROM
ROM	DM8598	32 x 8-bit bipolar ROM (compatible with DM8578 PROM)
	DM8597	256 x 4-bit bipolar ROM (compatible with DM8574 PROM)
	DM86L97	256 x 4-bit bipolar ROM
	MM5220/5221	128 x 8-bit MOS ROM
	MM5213	256 x 8-bit MOS ROM (compatible with MM5203 PROM)
	MM5214	512 x 8-bit MOS ROM (compatible with MM5204 PROM)
	DM8796	512 x 8-bit bipolar ROM
	MM5215	1024 x 12-bit MOS ROM
	DM8531	2048 x 8-bit bipolar ROM with address latches
	DM8581	1024 x 16-bit bipolar ROM with address latches

4.7 PERIPHERALS

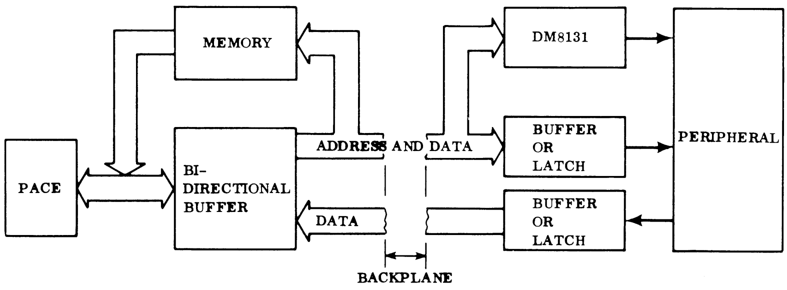
The following paragraphs discuss peripheral addressing and interfacing techniques.



A. FULLY MULTIPLEXED BUS


MEMORY
PACE
BI-DIRECTIONAL BUFFER + ADDRESS LATCH
ADDRESS
DATA
DM8131 OR DM8160
BI-DIRECTIONAL LATCH OR TRANSCEIVER
PERIPHERAL
BACKPLANE

B. SEPARATE ADDRESS MULTIPLEXED DATA



C. SEPARATE OUTPUT AND INPUT

Figure 4-6. Typical Data Bus Structure Configurations

4.7.1 Addressing

Peripheral devices are addressed in the same manner as memory locations. Whenever possible, an easily decoded section of memory address space for peripherals should be selected, even though the entire addressing capability of that addressing space is not utilized. While a wide variety of addressing schemes are possible, the Microprocessor Development System IPC-16P peripheral and memory address assignments should be taken into account in selecting a scheme. On the IPC-16P, addresses X'8000-X'BFFF are assigned to peripherals, and other addresses are available for memory. The IPC-16P system utilizes several of the peripheral addresses for IPC-16P peripheral devices (as listed in chapter 7). Consequently, during the program development phase, the user should avoid conflicts with addresses dedicated to the IPC-16P peripherals.

In some applications, assignment of part of the 16-bit address word as a command or control field may be a desirable way to pass control information to the peripheral as part of the address word, rather than to use a data word. For example, the IPC-16P peripherals use the word assignment shown in figure 4-7, which provides a 3-bit peripheral order field.

15	14	13 11	10 3	2 0
1	0	not used	address	order

Figure 4-7. IPC-16P Word Assignment for Peripheral Addressing

4.7.2 Interfacing 8-bit Peripherals

8-bit peripherals or memories should be interfaced to the eight low-order data lines. The eight high-order data lines are 'don't care' lines and need not be interfaced. The software can mask out the high-order data lines by using the Load with Sign Extended Instruction (LSEX) for loading arithmetic data or by using the shift instructions to set the eight most significant bits to zero. In many cases, the eight most significant bits can be left as random data during processing, since those bits do not affect status conditions. (See paragraph in this chapter titled "8-Bit Data Length.")

4.7.3 Interfacing Serial Devices

Serial interfaces to PACE can be provided over the data bus using a single bit for the data interface. In simple systems with only a few peripherals, use of the jump condition or interrupt inputs and flag outputs may be preferrable to avoid the need for address decoding and reduce the number of interface lines. The use of jump condition or interrupt inputs and flag outputs is particularly for asynchronous serial interfaces, such as a Teletypewriter, since only one transmit and one receive line are involved.

4.8 CONTROL SIGNALS

The NINIT and EXTEND control signals are discussed in the following paragraphs.

4.8.1 Initialize

The NINIT input may be used at any time to initialize the processor and always should be used at power up. The initialize signal causes the stack to initialize, the flags to be set to zero, the Level-zero Interrupt enable to be set true, and the Program Counter to be set to zero. The accumulators initialize to an arbitrary state. NADS, IDS and ODS data strobes are set false if NINIT occurs during an I/O operation, but the signal sequence is not predictable. Thus, if system data is to be preserved during initialization, the NINIT Signal should be inhibited during data I/O cycles.

After the trailing edge of the NINIT Signal, there is no activity on the PACE data strobes for 16 clock cycles. After the 16 clock cycles, the first NADS occurs and the first instruction is accessed from location zero, unless a Level-zero Interrupt is present (all other interrupt levels are disabled).

4.8.2 Extend

The EXTEND input may be utilized to increase I/O cycle times, by multiples of the clock period, or to provide cycle stealing memory access capability, by suspending CPU activity. To extend I/O cycles, the NADS Signal may be used to initiate the EXTEND pulse while a TTL version of CLK may be used to count out the desired number of clock cycles. The circuit illustrated in figure 4-8 provides an extend of one clock cycle (shown in timing diagram of figure 4-8) for data-input operations, as might be required for an MOS ROM.

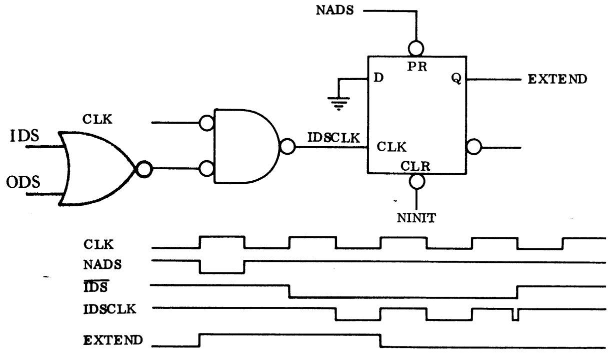


Figure 4-8. Circuit and Timing Diagram for One Clock Cycle Extend

To use the EXTEND input for suspending CPU operation, the EXTEND input is brought true immediately following a data I/O operation. Thus, the data bus then is free for use by peripheral devices to communicate with memory. If waiting for the occurence of a CPU I/O cycle is impractical, the EXTEND input may be brought up any time an I/O cycle is not in progress. If no NADS Signal occurs for 1-1/2 clock cycles, a suspend operation is guaranteed and the peripheral may use the bus. If an NADS does occur, the EXTEND should be removed and then brought high again at the end of the I/O cycle, at which time a suspend is assured. The circuit shown in figure 4-9 provides the functions required for a suspend operation and may be used as the basis of a cycle-stealing DMA system. (Actually this approach will sometimes stall the processor and sometimes steal a cycle.) Maximum response time equals one clock cycle plus one I/O cycle. Note that this approach is only suited to transfers of one or a few words, due to the limited length of an individual extend duration (see data sheet). The Bus Request should be held true until the DMA operation is complete.

Change 1

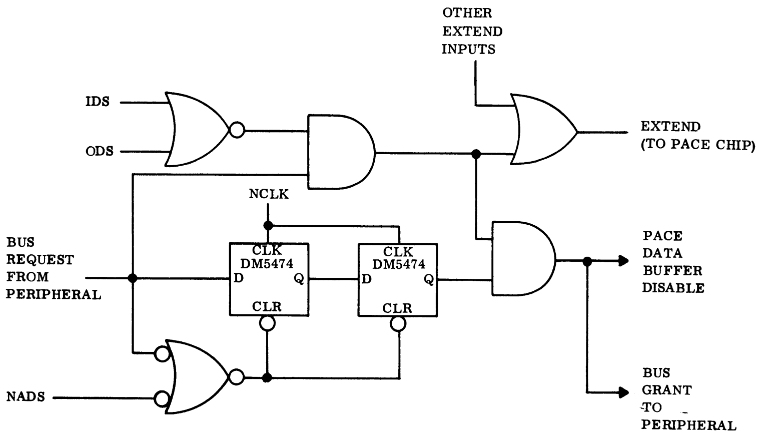


Figure 4-9. Circuit Providing Functions for Suspending CPU Operation

4.9 EXPANSION OF FLAGS AND JUMP CONDITIONS

Four user flags are provided on the PACE CPU. In some cases, additional flags may be required for control purposes. The additional flags can be conveniently obtained by using a DM9334 8-bit addressable latch. An unused address bit or combination of bits may be utilized to enable the latch; 3 bits then can address one of eight flags and another bit can specify set or reset as illustrated in figure 4-10. A Store (ST) instruction may be used to output the address (data output is ignored).

15	14	13 … 4	3	2 … 0
0	1	not used	s/r	flag

Flag expansion address code enables latch

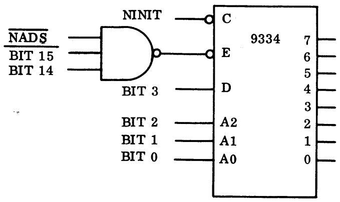


Figure 4-10. Example Circuit and Word Format for Obtaining Additional Flags

In a similar manner, a multiplexer and latch can be utilized to expand the user jump conditions. The latch is loaded from the address bus if enabled by an unused address code and selects a multiplexer input to one of the user jump conditions.

4.10 8-BIT DATA LENGTH

4.10.1 Overall Description

In applications where the principal data length is 8 bits, using an 8-bit data memory and taking advantage of the data-length-selection hardware features may be desirable. The data-length input modifies the operation of shift instructions and status flags to handle 8-bit data. The instruction memory always is 16 bits wide, and proper execution of the 16-bit instructions occurs independently of the data length selected. The use of 16-bit instructions in 8-bit data applications provides higher execution speeds. The hardware design allows the system to be used in the 8-bit mode for variable data, while, at the same time, using all 16 bits of the ALU, registers, and stack to manipulate 16-bit memory addresses. Thus, the 16-bit instruction set is used to manipulate 8-bit data.

NOTE

The use of a status flag to specify data length enables the microprocessor to be switched between 8- and 16-bit modes under program control.

When using the 8-bit data configuration, the 8-bit data is right justified in the 16-bit accumulator. The state of the leftmost 8 bits and the consequent effect on microprocessor operations must be considered. The following items are reviewed in the following paragraphs with respect to the 8-bit data length: data I/O, memory addressing, status flags, conditional branches, shifts and rotates, immediate instructions and mixed data lengths.

4.10.2 Data Input/Output

A system using the 8-bit data configuration usually has a 16-bit instruction memory (typically ROM), an 8-bit data memory, and an 8-bit peripheral device interface. When data is loaded into an accumulator from the 8-bit memory or peripheral device, the unused eight data lines may be driven to a logic '0' by the use of eight open collector gates. Thus, the left byte of the accumulator is zero. The unused eight data lines also may be left open (saving eight gates), in which case the left byte has an undetermined value dependent upon the system noise and previous states. In most cases, a nonzero left byte is of no concern, since status flags, conditional branches, shifts, and rotates ignore the left byte. However, the programmer must be aware of the nonzero value, since the results of instructions such as Copy Register to Flags (CRF) are determined by the left byte as well as the right byte. In cases when the state of the left byte is significant, that byte may be set to zero by using the shift instructions with a count of zero.

4.10.3 Memory Addressing

Both the indexed and base-page addressing modes require some consideration when using the 8-bit data configuration. For base-page addressing, accessing both 16-bit (program words) and 8-bit (data words) data using the base-page mode may be desirable. Since two different memories are used, splitting the base page between the two memories also may be desirable. Base-page splitting is accomplished most easily by using the Base-Page Selection (BPS) input to cause the base-page address to be in the range of -128 to +127, rather than 0 to +255.

For indexed addressing, Accumulators 2 and 3 are used as 16-bit memory pointers. If Accumulators 2 and 3 are loaded from the 8-bit memory, the upper byte may be set equal to the sign of the lower byte by using the LSEX Instruction. Thus, a 16-bit signed twos-complement number results.

4.10.4 Status Flags

The Overflow and Carry Flags are modified by arithmetic instructions. If the 8-bit configuration is selected by the state of the Byte status flag, the Overflow and Carry Flags are set based on the lower 8-bit byte only. That is, the Carry Flag is set if there is a carry out of the lower byte and the Overflow Flag is set based on an arithmetic overflow of the lower byte.

The Link Flag is affected by shift and rotate instructions. The Link Flag is set by the data shifted out of the lower byte when the 8-bit configuration is selected.

4.10.5 Conditional Branches

The branch and skip instructions are modified to account for the 8-bit data length. Thus, the REQ0 and NREQ0 conditions are affected only by the lower byte. The PSIGN and NSIGN Signals indicate the sign of the lower byte. The skip instructions (SKNE, SKG, SKAZ, ISZ and DSZ) test only the lower byte. Thus if 8-bit accumulator data is compared with a 16-bit program memory word, the contents of the upper byte of both words are ignored. The Add Immediate, Skip if Zero Instruction (AISZ) is the only instruction that tests the entire 16-bit result when the 8-bit configuration is selected. Thus, the AISZ Instruction can be used to increment the index accumulators (AC2, AC3) without skipping every time the lower byte is zero. Consequently, the sign of 8-bit numbers must be extended (LSEX Instruction) to properly detect zero when using AISZ with 8-bit data.

4.10.6 Shifts and Rotates

The shift and rotate instruction group (SHL, SHR, ROL, ROR) operates on the lower byte only and sets the upper byte to zero. Shift instructions with a count of zero provide a convenient means of setting the left byte of accumulators to zero when 8-bit data is used.

4.10.7 Immediate Instructions

The immediate instructions (LI, CAI, AISZ) all provide 16-bit, twos-complement data inputs. When working with 8-bit data, the upper byte usually can be ignored. If required, the upper byte can be cleared using a shift instruction.

4.10.8 Mixed Data Lengths

Working with 8-bit data and 16-bit instructions sometimes necessitates performing arithmetic operations by using a 16-bit operand from the program memory and an 8-bit operand from the data memory. If the result is to be treated as 8-bit data, no special considerations are required. If the result is to be treated as 16-bit data, the sign of the 8-bit operand must first be extended by using the LSEX Instruction. Also, signals (carry, overflow, and conditional branches) that are only a function of the lower byte should not be used. Alternatively, the Data Length Flag may temporarily be set to 16 bits, if desired.

4.10.9 Other Data Lengths

The previously mentioned factors make the use of PACE in 8-bit applications very convenient while still providing the advantages of a 16-bit instruction set. However, data lengths other than 8 or 16 bits may also be used, but special hardware conveniences are not provided. In particular, 4-bit applications involving BCD data may be implemented with PACE, since the Decimal Add Instruction (DECA) provides the required arithmetic capability. 4-bit shifts, status information, and special branch conditions are not required for many BCD applications.

4.11 INTERRUPTS

4.11.1 Expanding User Interrupts

Four user interrupt inputs are provided as explained in chapter 2. In some applications, providing several interrupts on a single input may be desirable. Several interrupts can be provided on a single input by using open-collector gates for a wired-or input and polling all the devices on a given level to discover the origin of the interrupt. However, in some applications, the polling technique may take excessive time. In such cases, use of the DM9318 Priority Encoder is recommended to encode the number of the highest priority interrupting device. A single instruction in the interrupt service routine then can be used to read the number of the interrupting device over the data bus. The use of the DM9318 Priority Encoder is shown in figure 4-11. The use of a DM8131 Comparator with latched output to detect the appropriate peripheral address is also illustrated by figure 4-11.

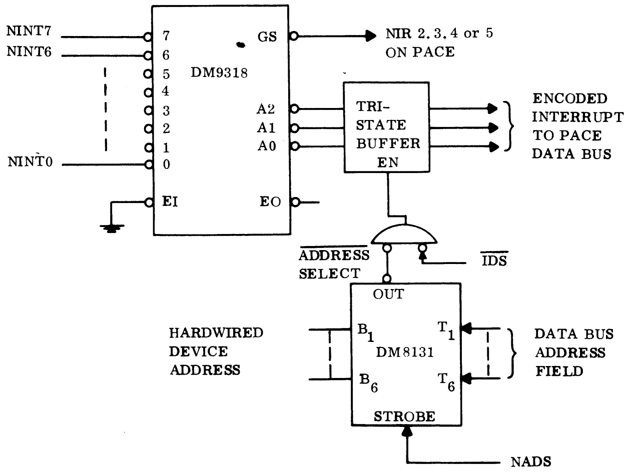


Figure 4-11. Use of DM9318 Priority Encoder and DM8131 Comparator For Interrupt Expansion and Detection

4.11.2 Stack Interrupts

A Stack Interrupt occurs when the stack-empty or stack-full condition exists. The Stack Interrupt consists of a pulse applied to the set input of Interrupt Request Latch 1 (see figure 2-8). The pulse sets the latch if the IEN1 Flag is true; otherwise, the pulse is ignored. The stack is implemented with a RAM and a pointer which can access RAM locations 0 to 9. A pulse occurs when the stack pointer is at 0 (one entry on stack), and a read-stack operation occurs to empty the stack. A pulse also occurs when the stack pointer is equal to 7 (eight entries on stack), and a write-stack operation occurs to fill the ninth word and leave one word empty to be used by the interrupt. When a Stack Interrupt occurs, the stack condition can be determined by using the Stack-full Jump Condition (STFL).

With the interrupt scheme described, an interrupt does not occur at initialize but does occur every time the stack becomes empty. If the stack is to be extended into memory, a Stack-empty Interrupt is required but may be inhibited by turning off IEN1 in other cases. In order to prevent a Stack Interrupt when both hardware and software stacks become empty, a dummy word may be pushed on the stack by the initialize routine. See chapter 3 for an example of a software stack routine.

4.12 NHALT CONTROL LINE

The NHALT control line is used for three different functions: programmed halt indicator output, processor stall input and nonmaskable Level-zero Interrupt input. The programmed halt indication is of interest for many end applications. The other two features, in addition to being used for some end applications, also are used for IPC-16P Control Panel implementation during the program development phase.

4.12.1 Programmed Halt

During normal program execution, the NHALT control line provides a logic '1' output. If a HALT Instruction is executed, the NHALT Line is driven to a logic '0', indicating that processor activity is suspended until the CONTIN input is pulsed. The NHALT output logic '0' signal has a duty cycle of 7/8; that is, every eighth clock phase the output goes to logic '1'. The NHALT 7/8 duty cycle must be accounted for if the output is used as a logic signal but is of little concern if the output drives only a halt indicator on the Control Panel. The NHALT output returns to logic '1' (with 100 percent duty cycle) after the HALT Instruction is terminated by pulsing the CONTIN input. The CONTIN input must go true for a minimum of four clock cycles.

4.12.2 Microprocessor Stall Input

To suspend operation of the PACE microprocessor under external control, the NHALT Line may be driven low by an external gate, overriding the output buffer on the chip as shown in figure 4-12. This feature may be useful for DMA systems requiring the processor to be idled during block data transfers.

The NHALT Signal causes operation of the processor to be suspended after the current instruction completes execution. The suspension may last for an indefinite period of time without loss of CPU status and may be terminated by use of the CONTIN input (properly sequenced with removal of the NHALT input). For cases where it is desirable to know when the CPU completes execution of the current instruction, the CONTIN Signal line is driven with a negative-true interrupt acknowledge signal. The entire timing sequence for the externally applied NHALT and CONTIN Signals is shown in figure 4-13.

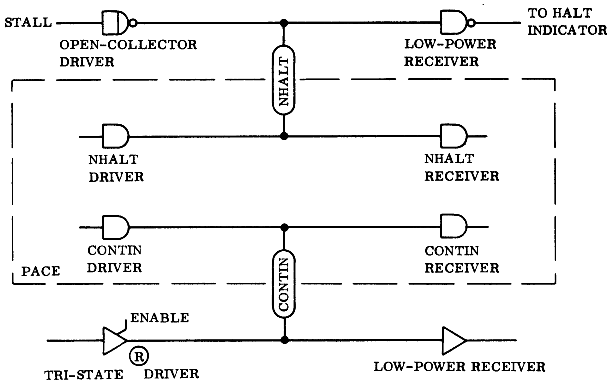


Figure 4-12. Halting and Starting Microprocessor Externally.

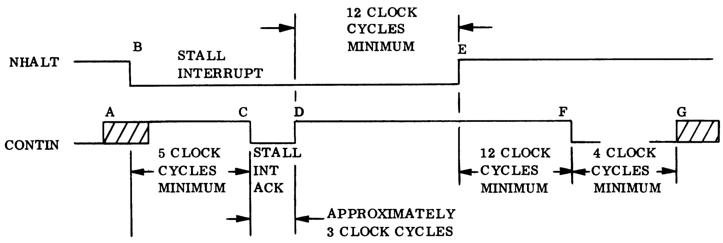


Figure 4-13. Timing Diagram for Externally Applied NHALT and CONTIN Signals

The following actions occur at the referenced points in figure 4-13.

A - The TRI-STATE ® CONTIN Driver is disabled to the high-impedance state. The CONTIN Signal Line then is pulled high by the internal CONTIN Driver Pullup Transistor. Event A may occur prior to event B as long as the program is not testing the CONTIN input using the BOC CONTIN Instruction at the same time (explained in description of event F). Event A also can occur after event B as long as event A is not delayed to the point where event A interferes with receiving the interrupt acknowledge signal.

B - The NHALT Signal Line is driven low by the open-collector driver causing microprocessor operation to be suspended at the end of the current instruction.

C - The internal CONTIN Driver pulls the CONTIN Line low to indicate that the current instruction execution is completed and CPU operation is suspended (this may be desirable for multiprocessor systems). Event C occurs a minimum of five clock cycles after event B and a maximum time equal to the longest instruction execution (long shift).

D - After three clock cycles, the internal driver pulls the CONTIN Line to a high level.

E - The open-collector NHALT Driver turns off to allow the on-chip NHALT Driver Pullup Transistor to pull the line to a logic '1' level. Event E must occur a minimum of 12 clock cycles after the current instruction execution is completed.

F - The TRI-STATE ® CONTIN Driver is re-enabled to the active state (may occur any time after the acknowledge) and drives the CONTIN Line to logic '0'. Event F must occur at least 12 clock cycles after event E. After event F the microprocessor then fetches the next instruction and continues with execution of the instruction.

G - The CONTIN line must stay low for at least 4 clock cycles.

In general, if the microprocessor stall feature is used, the CONTIN input is dedicated to the stall operation. However, multiplexing the CONTIN input is possible, so CONTIN also can be used as a general purpose sense input. For those cases where the BOC Instruction is used to test the CONTIN input, the previous sequence may not be used, since event A may cause the BOC Instruction to branch erroneously. In this case, the interrupt acknowledge may not be used and the sequence shown in figure 4-14 should be followed. The sequence illustrated in figure 4-14 keeps the TRI-STATE ® CONTIN Driver in the active mode all the time and uses a one-shot timer or a counter to provide a delay exceeding the longest instruction execution time. The simpler sequence of figure 4-14 also should be used in other cases where the interrupt acknowledge signal is not required.

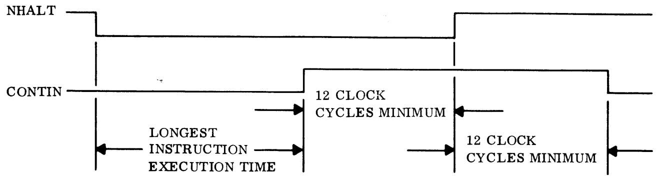


Figure 4-14. TRI-STATE ® CONTIN Driver Always Active

4.12.3 Minimal Control Panel

The previously described programmed halt and processor stall input allow the implementation of a simple Control Panel with a minimum of components. The Control Panel provides HALT, SINGLE INSTRUCTION, and RUN modes and displays data for a selected trap address or data and address for each single instruction executed.

The control logic and the associated timing are shown in figure 4-15. A one-shot is used in conjunction with the SINGLE INST Switch contact closure time (assumed greater than one-shot delay plus 12 clock cycles) to provide the proper single-instruction timing. The time between single-instruction closures is assumed to excede the longest instruction execution time. The single-instruction sequence is terminated by generating a halt after the first NADS. (The halt must be generated in less than eight clock cycles after NADS.) The RUN mode is entered by generating the single-instruction timing and inhibiting the termination on ADS.

The data display logic is shown in figure 4-16. In the HALT mode, the address and data for each single instruction execution are displayed. In the RUN mode, switch-selected address and data are displayed each time the selected address occurs on the data bus.

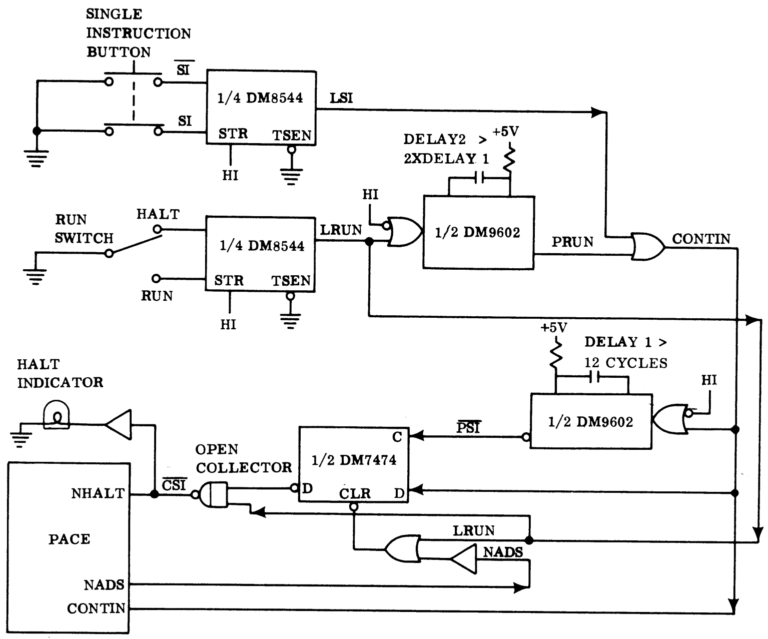
While the capabilities of a simple Control Panel are limited and may not be suitable for the program development stage, a simple Control Panel is adequate in certain end applications.

4.12.4 Level-zero Interrupt

The Level-zero Interrupt is not maskable under program control and is the highest priority interrupt. A Level-zero Interrupt may be used for alarm conditions such as power failure or for implementing a software-based Control Panel. A software-based Control Panel is the type implemented on the IPC-16P Microprocessor Development System. The software-based Control Panel has the capability of examing and altering all registers and memory locations, as well as implementing special functions such as bootstrap loading. A Level-zero Interrupt is generated by using the NHALT and CONTIN inputs as with the processor stall, but by using a different signal sequence as shown in figure 4-17.

Events A, B, C and D (in figure 4-17) occur in exactly the same manner as described in the "Microprocessor Stall Input" paragraph. Events E and F differ between figures 4-13 and 4-17. Event E in figure 4-17 consists of enabling the TRI-STATE ® CONTIN Driver and pulling CONTIN low. A low CONTIN Signal must occur within 15 clock cycles of the start of interrupt acknowledge. Event F completes the interrupt process. Event F consists of turning off the NHALT Driver. Event F must occur more than 11 clock cycles following the interrupt acknowledge.

For those cases where the use of the interrupt acknowledge is not required, or where the CONTIN Line is used as a sense input to the program, the CONTIN Line may be held continuously low, as illustrated in figure 4-18. The NHALT must be held low for the duration of the longest instruction execution time plus 11 clock cycles to guarantee that a Level- zero Interrupt occurs. After the occurence of event F, the Level-zero Interrupt is serviced. Servicing consists of first setting the Level-zero Interrupt enable false to lock out all possible interrupts (see chapter 2). Next, the Program Counter is stored at the location specified by the contents of memory location 7. Then, the instruction at location 8 is executed. Storing the Program Counter in a memory location instead of on the stack prevents generation of a Stack-full Interrupt.



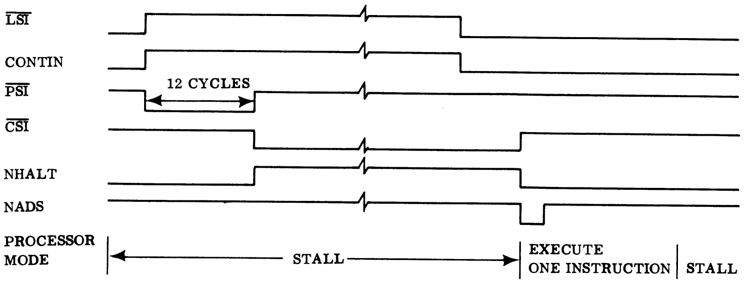


Figure 4-15. Control Panel Logic and Timing (Plus Time Line)

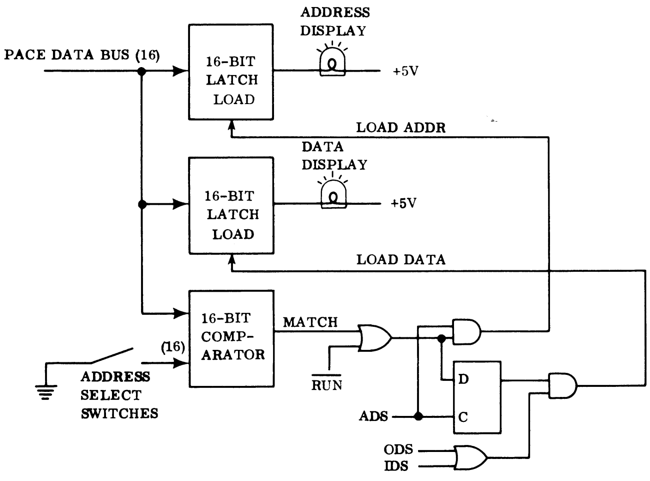


Figure 4-16. Panel Data Display Logic

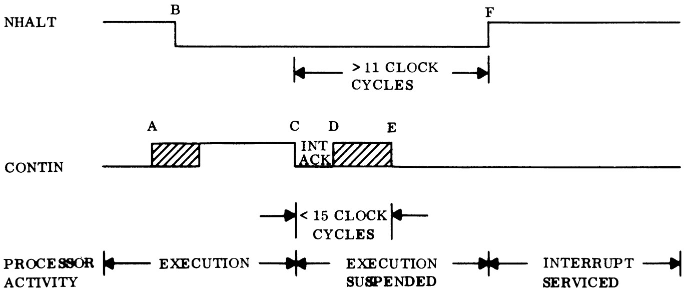


Figure 4-17. Level-zero Interrupt Generation

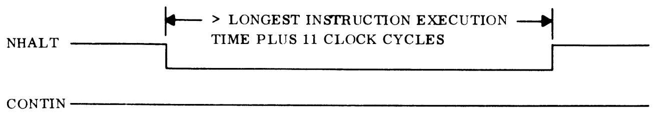


Figure 4-18. CONTIN Line as Sense Input to Program

In order to return from a Level-zero Interrupt, the PFLG 15 Instruction is executed. The PFLG 15 Instruction sets the Level-zero Interrupt enable true again after the execution of one more instruction, which is typically a JMP@ through the location where the PC is stored to cause a proper return to the interrupted program. The Level-zero Interrupt enable is set true during initialization.

Software-driven Control Panels intended for program development monitoring are usually implemented using the Level-zero Interrupt. The Level-zero Interrupt cannot be disabled by the program under development and thus the Control Panel always can control system operation. Also, the Level-zero Interrupt does not use the stack, and, thus, the stack interrupt status is not affected by the Control Panel. Software-driven Control Panels, intended for monitoring the operation of the microprocessor in an end application, often use a lower-priority, maskable interrupt, since monitoring often is secondary to the control operations performed by the microprocessor.

The hardware for a software-driven Control Panel is quite simple, usually consisting of switches for data input and indicators or numeric displays for data output. The switches and displays are implemented as peripheral devices, which can be read and written by the microprocessor to determine the desired control or display actions which the microprocessor must carry out. The functions of the Control Panel then are easily modified or expanded by changing the interrupt service routine.

APPENDIX B

PACE INSTRUCTION RELATED SUMMARIES AND PROGRAM EXAMPLES

B.1 INTRODUCTION

Table B-1 defines the notation and symbols used for the symbolic representation of each instruction contained in table B-2, the instruction summary. The notations in table B-1 are presented in alphabetical order and, then, the symbols are listed. Upper-case mnemonics refer to fields in the instruction word. Lower-case mnemonics refer to the numerical value of the corresponding fields. In cases where both upper-case and lower-case mnemonics are composed of the same letters, only the lower-case mnemonic is given. The use of lower-case notation designates variables.

The formulas in table B-2 (the instruction summary) for computing the execution times of instructions are presented in terms of machine (microinstruction) cycles (M) and input/output data-transfer Cycle Extends (E_R for read and E_W for write). Each machine cycle (M) consists of four clock cycles. The following example shows the method to be employed for computing the execution times of instructions.

EXAMPLE

The formula (listed in table B-2) for the execution time of a RADD Instruction is $4M+E_R$. If the clock cycle (or period) is 500 nanoseconds and the Read Cycle Extend is 500 nanoseconds, then: $M=4(0.5\ \mu sec)=2\ \mu sec$
$E_R=0.5\ \mu sec$
therefore: $4M+E_R=4(2\ \mu sec)+0.5\ \mu sec=8.5\ \mu sec$.
Thus, under the hypothetical clock cycle and Read Cycle Extend times used, the RADD Instruction requires 8.5 microseconds for execution.

Following the instruction summary, the branch conditions for the 16 condition codes used by the Branch-On Condition Instruction are described in table B-3.

Table B-1. Notations/Symbols Used in Instruction Descriptions

Notation/ Symbol	Meaning
ACr	Denotes specific working accumulator (AC0, AC1, AC2, or AC3), where *r* is number of accumulator referenced in instruction.
cc	Denotes 4-bit condition code value for conditional branch instructions.
CRY	Indicates Carry Flag is set if carry exists due to instruction (either addition or subtraction) or reset if no carry exists.
disp	Stands for displacement value and represents operand in non-memory-reference instruction or address field in memory-reference instruction. Disp is 8-bit, signed twos-complement number except when base page is referenced; in latter case, disp is unsigned if BPS=0.
dr	Denotes number of destination working accumulator specified in instruction-word field. Working accumulator is AC0, AC1, AC2, or AC3.
EA	Denotes effective address specified by instructions directly, indirectly, or by indexing. Effective address contents are used during execution of instruction.

Table B-1. Notations/Symbols Used in Instruction Descriptions (Continued)

Notation/ Symbol	Meaning
fc	Denotes number of referenced flag.
FR	Denotes Status and Control Flags Register.
IEN	Denotes Interrupt Enable Flag.
ℓ	Denotes inclusion of 1-bit Link Flag (LINK) in shift operations.
n	Unsigned number indicates number of bit positions to be shifted in Shift and Rotate Instructions.
OVF	Indicates Overflow Flag is set if overflow exists due to instruction (either addition or subtraction) or is reset if no overflow exists. Overflow occurs if signs of operands are alike and sign of result is different from operands.
PC	Denotes Program Counter. During address formation, PC is incremented by 1 to contain address 1 greater than that of instruction being executed.
r	Denotes number of working accumulator specified in instruction-word field. Working accumulator is AC0, AC1, AC2, or AC3.
STK	Denotes top word of 10-word Last-In/First-Out Stack.
sr	Denotes number of source working accumulator specified in instruction-word field. Working accumulator is AC0, AC1, AC2, or AC3.
xr	When not zero, xr value designates number of accumulator to be used in indexed and relative-memory addressing modes. When zero, base-page addressing is indicated.
( )	Denotes contents of item within parentheses. (ACr) is read as *contents of ACr*. (EA) is read as *contents of EA.*
[ ]	Denotes *result of.*
~	Indicates logical complement (ones complement) of value on right-hand side of ~.
→	Means *replaces.*
←	Means *is replaced by.*

Table B-1. Notations/Symbols Used in Instruction Descriptions (Continued)

Notation/ Symbol	Meaning
@	Appearing in operand field of instruction, denotes indirect addressing.
+10	Modulo 10 addition.
$\wedge$	Denotes AND operation.
$\vee$	Denotes OR operation.
$\overline{\vee}$	Denotes EXCLUSIVE OR operation.

Table B-2. PACE Instruction Summary

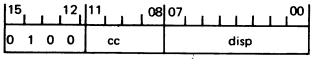
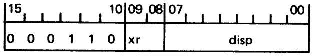
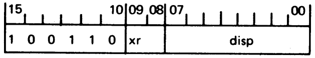
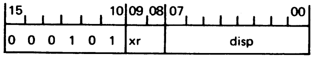
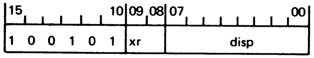
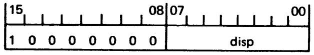
Instruction/Mnemonic	Operation/Description	Assembler Format	Execution Time/Cycles (M)
BRANCH INSTRUCTIONS			
Branch-On Condition BOC 	(PC)←(PC) + disp if cc true 16 possible condition codes (cc) exist. Condition codes are listed in table B-3. If condition for branching designated by cc is true, value of disp (sign extended from bit 7 through bit 15) is added to PC and sum is stored in PC.	BOC cc, disp	5M + E_R + 1M if branch
Jump JMP 	(PC)←EA Effective address EA replaces PC contents. Next instruction is fetched from location designated by new contents of PC.	JMP disp (xr)	4M + E_R
Jump Indirect JMP@ 	(PC)←(EA) Contents of effective address replace PC contents. Next instruction is fetched from location designated by new contents of PC.	JMP @disp (xr)	4M + $2E_R$
Jump to Subroutine JSR 	(STK)←(PC), (PC)←EA Contents of PC are stored on top of Stack. Effective address replaces PC contents. Next instruction is fetched from location designated by new contents of PC.	JSR disp (xr)	5M + E_R
Jump to Subroutine Indirect JSR@ 	(STK)←(PC), (PC)←(EA) Contents of PC are stored on top of Stack. Contents of effective address replace PC contents. Next instruction is fetched from location designated by new contents of PC.	JSR @disp (xr)	5M + $2E_R$
Return from Subroutine RTS 	(PC)←(STK) + disp Contents of PC are replaced by sum of disp added to contents pulled from top of Stack. Program control is transferred to location specified by new contents of PC.	RTS disp	5M + E_R

Table B-2. PACE Instruction Summary (Continued)

Instruction/Mnemonic	Operation/Description	Assembler Format	Execution Time/Cycles (M)
BRANCH INSTRUCTIONS (Continued)			
Return from Interrupt RTI Bits 15–08: 0 1 1 1 1 1 0 0; bits 07–00: disp	$(PC) \leftarrow (STK)$ + disp, IEN = 1 Interrupt Enable Flag (IEN) is set. PC contents are replaced by sum of disp and word pulled from top of Stack. Program control is transferred to location specified by new contents of PC.	RTI disp	$6M + E_R$
SKIP INSTRUCTIONS			
Skip if Not Equal SKNE Bits 15–12: 1 1 1 1; bits 11–10: r; bits 09–08: xr; bits 07–00: disp	If $(ACr) \neq (EA)$, $(PC) \leftarrow (PC) + 1$ ACr contents and contents of effective memory location EA are compared. If contents of ACr and EA are not equal, next instruction in sequence is skipped. Contents of ACr and EA are unaltered. If 8-bit data length is selected, only lower 8 bits are compared.	SKNE r, disp (xr)	$5M + 2E_R$ + 1M if skip
Skip if Greater SKG Bits 15–10: 1 0 0 1 1 1; bits 09–08: xr; bits 07–00: disp	If $(AC0) > (EA)$, $(PC) \leftarrow (PC) + 1$ AC0 contents and contents of effective memory location EA are compared as signed numbers. If contents of AC0 are greater (more positive) than contents of EA, next instruction in sequence is skipped. Contents of AC0 and EA are unaltered. If 8-bit data length is selected, only lower 8 bits are compared.	SKG 0, disp (xr)	$7M + 2E_R$ + 1M if skip
Skip if AND is Zero SKAZ Bits 15–10: 1 0 1 1 1 0; bits 09–08: xr; bits 07–00: disp	If $((AC0) \wedge (EA)) = 0$, $(PC) \leftarrow (PC) + 1$ AC0 contents and contents of effective memory location EA are ANDed. If result equals zero, next instruction in sequence is skipped. Contents of AC0 and EA are unaltered. If 8-bit data length is selected, only lower 8 bits are tested.	SKAZ 0, disp (xr)	$5M + 2E_R$ + 1M if skip

Table B-2. PACE Instruction Summary (Continued)

Instruction/Mnemonic	Operation/Description	Assembler Format	Execution Time/Cycles (M)
SKIP INSTRUCTIONS (Continued)			
Increment and Skip if Zero ISZ 15 … 10 | 09 08 | 07 … 00 1 0 0 0 1 1 | xr | disp	(EA)←(EA) + 1, if (EA) = 0, (PC)←(PC) + 1 EA contents are incremented by 1. If new contents of EA equal zero, next instruction in sequence is skipped. If 8-bit data length is selected, only lower 8 bits are tested.	ISZ disp (xr)	7M + $2E_R$ + E_W + 1M if skip
Decrement and Skip if Zero DSZ 15 … 10 | 09 08 | 07 … 00 1 0 1 0 1 1 | xr | disp	(EA)←(EA) - 1, if (EA) = 0, (PC)←(PC) + 1 EA contents are decremented by 1. If new contents of EA equal zero, next instruction in sequence is skipped. If 8-bit data length is selected, only lower 8 bits are tested.	DSZ disp (xr)	7M + $2E_R$ + E_W + 1M if skip
Add Immediate, Skip if Zero AISZ 15 … 10 | 09 08 | 07 … 00 0 1 1 1 1 0 | r | disp	(ACr)←(ACr) + disp, if (ACr) = 0, (PC)←(PC) + 1 ACr contents are replaced by sum of contents of ACr and disp (sign bit 7 extended through bit 15). Initial contents of ACr are lost. If new contents of ACr equal zero, contents of PC are incremented by 1, thus skipping next instruction. AISZ Instruction always tests full 16-bit result independent of data length selected.	AISZ r, disp	5M + E_R + 1M if skip
MEMORY DATA-TRANSFER INSTRUCTIONS			
Load LD 15 … 12 | 11 10 | 09 08 | 07 … 00 1 1 0 0 | r | xr | disp	(ACr)←(EA) ACr contents are replaced by EA contents. Initial contents of ACr are lost; contents of EA are unaltered.	LD r, disp (xr)	4M + $2E_R$

Table B-2. PACE Instruction Summary (Continued)

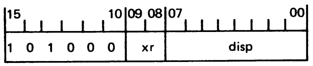
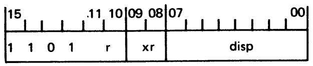
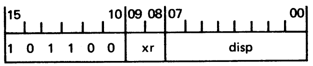
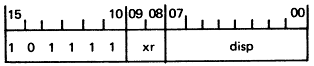
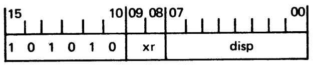
Instruction/Mnemonic	Operation/Description	Assembler Format	Execution Time/Cycles (M)
MEMORY DATA-TRANSFER INSTRUCTIONS (Continued)			
Load Indirect LD@ 	(ACO)←((EA)) ACO contents are replaced indirectly by EA contents. Initial contents of ACO are lost; contents of EA and location designating EA are unaltered.	LD 0, @disp (xr)	5M + 3E_R
Store ST 	(EA)←(ACr) EA contents are replaced by contents of ACr. Initial contents of EA are lost; contents of ACr are unaltered.	ST r, disp (xr)	4M + E_R + E_W
Store Indirect ST@ 	((EA))←(ACO) EA contents are replaced indirectly by ACO contents. Initial contents of EA are lost; contents of ACO and location designating EA are unaltered.	ST 0, @disp (xr)	4M + 2E_R + E_W
Load with Sign Extended LSEX 	(ACO)←(EA) bit 7 extended ACO contents are replaced by EA contents with bit 7 extended through bits 8-15. Initial contents of ACO are lost; contents of EA are unaltered. LSEX permits 8-bit data loading from memory or peripheral to be operated on as 16-bit data.	LSEX 0, disp (xr)	4M + 2E_R
MEMORY DATA-OPERATE INSTRUCTIONS			
AND AND 	(ACO)←(ACO) ∧ (EA) ACO contents and EA contents are ANDed. Result is stored in ACO. Initial contents of ACO are lost; contents of EA are unaltered.	AND 0, disp (xr)	4M + 2E_R

Table B-2. PACE Instruction Summary (Continued)

Instruction/Mnemonic	Operation/Description	Assembler Format	Execution Time/Cycles (M)
MEMORY DATA-OPERATE INSTRUCTIONS (Continued)			
OR — OR 15–10 | 09 08 | 07–00 1 0 1 0 0 1 | xr | disp	(AC0)←(AC0) V (EA) AC0 contents and EA contents are ORed inclusively. Result in stored in AC0. Initial contents of AC0 are lost; contents of EA are unaltered.	OR 0, disp (xr)	4M + $2E_R$
Add — ADD 15–12 | 11 10 | 09 08 | 07–00 1 1 1 0 | r | xr | disp	(ACr)←(ACr) + (EA), OV, CY ACr contents are added algebraically to EA contents. Sum is stored in ACr, and contents of EA are unaltered. Initial contents of ACr are lost. Overflow or Carry Flag is set if overflow or carry occurs, respectively; otherwise Overflow and Carry Flags are cleared.	ADD r, disp (xr)	4M + $2E_R$
Subtract with Borrow — SUBB 15–10 | 09 08 | 07–00 1 0 0 1 0 0 | xr | disp	(AC0)←(AC0) + ~(EA) + (CY), OV, CY AC0 contents are added to complement of EA and carry. Result is stored in AC0 and contents of EA are unaltered. Initial contents of AC0 are lost. Carry and Overflow Flags are set according to result of operation.	SUBB 0, disp (xr)	4M + $2E_R$
Decimal Add — DECA 15–10 | 09 08 | 07–00 1 0 0 0 1 0 | xr | disp	(AC0)←(AC0) $+_{10}$ (EA) $+_{10}$ (CY), OV, CY AC0 contents are treated as 4-digit number and added modulo 10 (for each digit) to contents of EA (treated as 4-digit number) and carry. Initial contents of AC0 are lost; contents of EA are unaltered. Carry Flag is set based on decimal carry output. Overflow Flag is set to arbitrary state.	DECA 0, disp (xr)	7M + $2E_R$

Table B-2. PACE Instruction Summary (Continued)

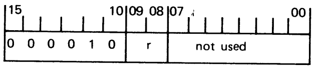
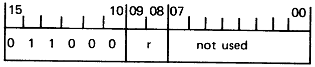
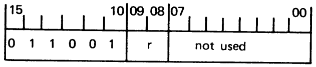
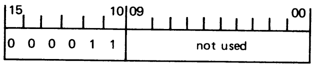
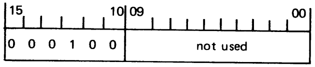
Instruction/Mnemonic	Operation/Description	Assembler Format	Execution Time/Cycles (M)
REGISTER DATA-TRANSFER INSTRUCTIONS (Continued)			
Copy Register into Flags CRF 	(FR)←(ACr) FR contents are replaced by ACr contents. Initial contents of FR are lost; contents of ACr are unaltered.	CRF r	$4M + E_R$
Push Register onto Stack PUSH 	(STK)←(ACr) Stack is pushed by contents of accumulator designated by ACr. Thus, top of Stack holds ACr contents and Stack Pointer is incremented by 1. Initial contents of ACr are unaltered.	PUSH r	$4M + E_R$
Pull Stack into Register PULL 	(ACr)←(STK) Stack is pulled. Contents from top of Stack replace ACr contents. Initial contents of ACr are lost. Contents of Stack Pointer are decremented by 1.	PULL r	$4M + E_R$
Push Flags onto Stack PUSHF 	(STK)←(FR) FR contents are pushed onto Stack. Contents of FR are unchanged.	PUSHF	$4M + E_R$
Pull Stack into Flags PULLF 	(FR)←(STK) FR contents are replaced by contents pulled from top of Stack. Initial contents of FR are lost.	PULLF	$4M + E_R$

Table B-2. PACE Instruction Summary (Continued)

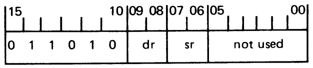
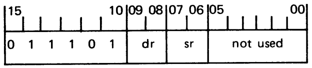
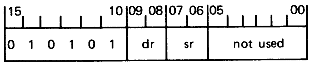
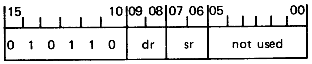
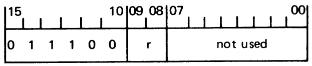
Instruction/Mnemonic	Operation/Description	Assembler Format	Execution Time/Cycles (M)
REGISTER DATA-OPERATE INSTRUCTIONS			
Register Add RADD  15 10 09 08 07 06 05 00 0 1 1 0 1 0 dr sr not used	(ACdr)←(ACdr) + (ACsr), OV, CY ACdr contents are replaced by sum of contents of ACdr and ACsr. Initial contents of ACdr are lost and contents of ACsr are unaltered. Overflow and Carry Flags are modified according to result.	RADD sr, dr	$4M + E_R$
Register Add with Carry RADC  15 10 09 08 07 06 05 00 0 1 1 1 0 1 dr sr not used	(ACdr)←(ACdr) + (ACsr) + (CY), OV, CY ACdr contents are replaced by sum of ACdr and ACsr contents and carry. Initial ACdr contents are lost and ACsr contents are unaltered. Overflow and Carry Flags are modified according to result.	RADC sr, dr	$4M + E_R$
Register AND RAND  15 10 09 08 07 06 05 00 0 1 0 1 0 1 dr sr not used	(ACdr)←(ACdr) $\wedge$ (ACsr) ACdr contents are replaced by result of ANDing ACdr and ACsr contents. Initial contents of ACdr are lost and initial contents of ACsr are unaltered.	RAND sr, dr	$4M + E_R$
Register EXCLUSIVE OR RXOR  15 10 09 08 07 06 05 00 0 1 0 1 1 0 dr sr not used	(ACdr)←(ACdr) $\veebar$ (ACsr) ACdr contents are replaced by result of EXCLUSIVEly ORing ACdr contents and ACsr contents. Initial contents of ACdr are lost and initial contents of ACsr are unaltered.	RXOR sr, dr	$4M + E_R$
Complement and Add Immediate CAI 	(ACr)←~(ACr) + disp ACr contents are replaced by sum of complement of ACr and disp (sign bit 7 extended through bit 15). Initial contents of ACr are lost. Values of 0 and 1 in disp field produce ones and twos, complement, respectively, of (ACr).	CAI r, disp	$5M + E_R$

Table B-2. PACE Instruction Summary (Continued)

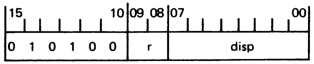
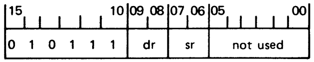
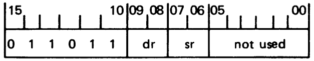
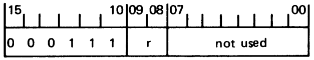
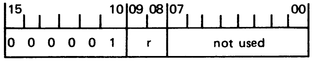
Instruction/Mnemonic	Operation/Description	Assembler Format	Execution Time/Cycles (M)
REGISTER DATA-TRANSFER INSTRUCTIONS			
Load Immediate LI 	(ACr)←disp ACr contents are replaced by disp with sign bit 7 extended through bit 15. Initial contents of ACr are lost.	LI r, disp	4M + E_R
Register Copy RCPY  15 10 09 08 07 06 05 00 0 1 0 1 1 1 dr sr not used	(ACdr)←(ACsr) Destination Register ACdr contents are replaced by contents of Source Register ACsr. Initial contents of ACdr are lost and initial contents of ACsr are unaltered.	RCPY sr, dr	4M + E_R
Register Exchange RXCH  15 10 09 08 07 06 05 00 0 1 1 0 1 1 dr sr not used	(ACdr)←(ACsr), (ACsr)←(ACdr) ACsr contents and ACdr contents are exchanged.	RXCH sr, dr	6M + E_R
Exchange Register and Stack XCHRS 	(STK)←(ACr), (ACr)←(STK) Contents of top of Stack and accumulator designated by ACr are exchanged.	XCHRS r	6M + E_R
Copy Flags into Register CFR 	(ACr)←(FR) ACr contents are replaced by contents of FR. Initial contents of ACr are lost; contents of FR are unaltered.	CFR r	4M + E_R

Table B-2. PACE Instruction Summary (Continued)

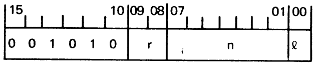
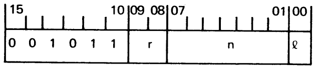
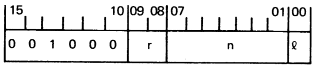
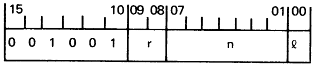
Instruction/Mnemonic	Operation/Description	Assembler Format	Execution Time/Cycles (M)
SHIFT AND ROTATE INSTRUCTIONS			
Shift Left SHL 	(ACr)←(ACr) shifted left n places, w/wo link ACr contents are shifted left n (n = 0-127) bit positions. If selected data length is 8 bits, then bits 8-15 are set to zero. Data shifted out of most significant bit for specified data length are lost if ℓ = 0 and are loaded into LINK if ℓ = 1.	SHL r, n, ℓ	(5 + 3n) M + E_R, n = 1-127; 6M + E_R, n = 0
Shift Right SHR 	(ACr)←(ACr) shifted right n places, w/wo link ACr contents are shifted right n (n = 0-127) bit positions. If selected data length is 8 bits, then bits 8-15 are set to zero. Zeros are shifted into most significant bit for specified data length if ℓ = 0. Contents of LINK are shifted in if ℓ = 1, and contents of LINK are unchanged. Data shifted out of least significant bit are lost.	SHR r, n, ℓ	(5 + 3n) M + E_R, n = 1-127; 6M + E_R, n = 0
Rotate Left ROL 	(ACr)←(ACr) rotated left n places, w/wo link ACr contents are rotated left n (n = 0-127) bit positions. If selected data length is 8 bits, then bits 8-15 are set to zero. Data shifted out of most significant bit position for specified data length are shifted into least significant bit if ℓ = 0, and into LINK if ℓ = 1, in which case least significant bit is loaded from LINK.	ROL r, n, ℓ	(5 + 3n) M + E_R, n = 1-127; 6M + E_R, n = 0
Rotate Right ROR 	(ACr)←(ACr) rotated right n places, w/wo link ACr contents are rotated right n (n = 0-127) bit positions. If selected data length is 8 bits, then bits 8-15 are set to zero. Data shifted out of least significant bit are shifted into most significant bit for specified data length if ℓ = 0, and into LINK if ℓ = 1, in which case most significant bit is loaded from LINK.	ROR r, n, ℓ	(5 + 3n) M + E_R, n = 1-127; 6M + E_R, n = 0

Table B-2. PACE Instruction Summary (Continued)

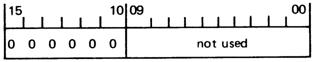
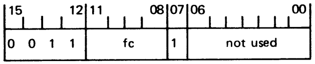
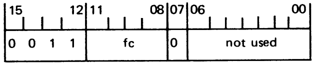
Instruction/Mnemonic	Operation/Description	Assembler Format	Execution Time/Cycles (M)
MISCELLANEOUS INSTRUCTIONS			
Halt HALT 	Halt Microprocessor halts and remains halted until CONTIN Input to Jump Condition Multiplexer makes transition from logic '1' to logic '0'.	HALT	- - - - - -
Set Flag SFLG 	$(FR)_{fc} \leftarrow 1$ Flag, or bit of FR, specified by flag code fc is set true. All other bits of FR are unaltered.	SFLG fc	$5M + E_R$
Pulse Flag PFLG 	$(FR)_{fc} \leftarrow 1$, $(FR)_{fc} \leftarrow 0$ Flag (bit fc of FR) is first set true and then set false (after four clock periods), causing pulsing or resetting of flag, depending on initial state of flag. All other bits of FR are unaffected.	PFLG fc	$6M + E_R$

Table B-3. **Branch Conditions**

Condition Code (cc)	Mnemonic	Condition
0000	STFL	Stack Full (contains nine or more words).
0001	REQ0	(AC0) equal to zero (see note 1).
0010	PSIGN	(AC0) has positive sign (see note 2).
0011	BIT0	Bit 0 of AC0 true.
0100	BIT1	Bit 1 of AC0 true.
0101	NREQ0	(AC0) is nonzero (see note 1).
0110	BIT2	Bit 2 of AC0 is true.
0111	CONTIN	CONTIN (continue) Input is true.
1000	LINK	LINK is true.
1001	IEN	IEN is true.
1010	CARRY	CARRY is true.
1011	NSIGN	(AC0) has negative sign (see note 2).
1100	OVF	OVF is true.
1101	JC13	JC13 Input is true (see note 3).
1110	JC14	JC14 Input is true.
1111	JC15	JC15 Input is true.

NOTES:

1. If selected data length is 8 bits, only bits 0 through 7 of AC0 are tested.
2. Bit 7 is sign bit (instead of bit 15) if selected data length is 8 bits.
3. JC13 is used by PACE Microprocessor Development System and is not accessible during prototyping.

B.2 PROGRAMMING EXAMPLES

The following paragraphs provide typical programming examples.

B.2.1 DECIMAL ADDITION

The decimal addition program (see table B-4) adds two 16-digit BCD strings that are packed 4 digits per word. The two strings to be added are stored in memory starting at locations STR1 and STR2. The resulting digit string is stored in memory starting at location STR2.

B.2.2 TENS COMPLEMENT

Representation of negative decimal numbers in tens-complement form may be desirable for many PACE applications, since the Decimal-Add Instruction then can be used directly for signed number additions. The tens-complement program converts an unsigned BCD number to a tens-complement negative number representation.

The sign of a tens-complement number can be tested by using the BOC Instruction with the PSIGN Jump Condition to test the most significant word of the decimal number.

NOTE

Negative numbers have leading nines while positive numbers have leading zeros.

The tens-complement program presented in table B-5 converts a 16-digit number packed in 4 words of memory beginning at location NUM.

B.2.3 DECIMAL SUBTRACTION

The decimal subtraction program listed in table B-6 performs a decimal subtract by forming the tens complement and using the Decimal-Add Instruction. The 16-digit string, starting at location STR2, is subtracted from the string starting at location STR1.

B.2.4 BINARY MULTIPLICATION

Two binary-multiplication program examples are provided in table B-7. The first program example multiplies the 16-bit value in AC2 by the 16-bit value in AC0 and provides a 32-bit result in AC1 (high order) and AC0 (low order).

NOTE

Positive numbers of 16-bit magnitude are assumed (that is, most significant bit is zero).

The second program multiplies the 16-bit value in AC2 by the 16-bit value in AC0 and provides a 32-bit result in AC0 (high order) and AC1 (low order).

NOTE

16-bit magnitude only is assumed.

Table B-4. Decimal Addition Program Example

```
                         .TITLE  DECADD, ' DECIMAL ADDITION'
                         ;
      0000               .ASECT
      0100               .=X'100
                         ;
      0000     R0        =       0
      0001     R1        =       1
      0002     R2        =       2
      0003     R3        =       3
      0007     CRY       =       7
                         ;
0100  0200  A  STR1:     .WORD   X'200
0101  0250  A  STR2:     .WORD   X'250
0102  0100  A  ADDR1:    .WORD   STR1             ;ADDRESS OF ADDEND STRING
0103  0101  A  ADDR2:    .WORD   STR2             ;ADDRESS OF AUGEND/RESULT ST
                         ;
                         ;
0104  5104  A  START:    LI      R1,4             ;NUMBER DIGITS/4 TO AC1 (LOO
0105  C9FC  A            LD      R2,ADDR1         ;LOAD INDEX REGISTERS WITH
0106  CDFC  A            LD      R3,ADDR2         ;  ARGUMENT ADDRESSES
0107  3700  A            PFLG    CRY              ;CLEAR CARRY FLAG
0108  C200  A  LOOP:     LD      R0,(R2)          ;ADDEND TO AC0
0109  8B00  A            DECA    R0,(R3)          ;DECIMAL ADD WITH AUGEND
010A  D300  A            ST      R0,(R3)          ;STORE RESULT
010B  7A01  A            AISZ    R2,1             ;INCREMENT INDEX
010C  7B01  A            AISZ    R3,1             ;  REGISTERS
010D  79FF  A            AISZ    R1,-1            ;DECREMENT LOOP COUNT
010E  19F9  A            JMP     LOOP             ;ADD NEXT WORD
                         ;
      0104               .END    START
```

Table B-5. Tens Complement Program Example

```
                        .TITLE  TENCOM, ' TENS-COMPLEMENT '
                        ;
     0000               .ASECT
     0100               .=X'100
                        ;
     0000     R0        =       0
     0001     R1        =       1
     0002     R2        =       2
     0007     CRY       =       7
                        ;
0100 00C8 A   NUM:      .WORD   200
0101 0100 A   ADDR:     .WORD   NUM
0102 999A A   CONST:    .WORD   X'999A
                        ;
0103 5104 A   START:    LI      R1,4            ;LOOP COUNT TO AC1
0104 C9FC A             LD      R2,ADDR         ;ADDRESS TO AC2 INDEX REGIST
0105 3780 A             SFLG    CRY             ;SET CARRY FLAG FOR FIRST LO
0106 C1FB A   LOOP:     LD      R0,CONST        ;CONSTANT TO AC0
0107 9200 A             SUBB    R0,(R2)         ;COMPLEMENT AND ADD DECIMAL
                                                ;  NUMBER PLUS CARRY
0108 D200 A             ST      R0,(R2)         ;STORE RESULT
0109 3700 A             PFLG    CRY             ;CLEAR CARRY FOR SUBSEQUENT
010A 7A01 A             AISZ    R2,1            ;INCREMENT POINTER
010B 79FF A             AISZ    R1,-1           ;DECREMENT LOOP COUNT
010C 19F9 A             JMP     LOOP            ;REPEAT LOOP
                        ;
     0103               .END    START
```

Table B-6. Decimal Subtraction Program Example

```
                              .TITLE  DECSUB, ' DECIMAL SUBTRACTION '
                              ;
                              ;         CALLING SEQUENCE
                              ;
                              ;           JSR     DECS
                              ;
      0000       R0           =       0
      000B       SIGN         =       11
      000A       CRY          =       10
      0007       CARRY        =       7
                              ;
 0000 9999 A     H9999:       .WORD   X'9999
 0001 0200 A     OP1:         .WORD   X'200              ;OPERAND 1 IN LOCATION 200
 0002 0201 A     OP2:         .WORD   X'201              ;OPERAND 2 IN LOCATION 201
                              ;
 0003 3700 A     DECS:        PFLG    CARRY              ;CLEAR CARRY FLAG
 0004 3B00 A                  PFLG    SIGN               ;CLEAR SIGN (USER) FLAG
 0005 C1FA T                  LD      R0,H9999           ;TAKE 9'S COMPLEMENT OF OP1
 0006 91FA T                  SUBB    R0,OP1
 0007 89FA T                  DECA    R0,OP2
 0008 4A04 A                  BOC     CRY,CTRUE          ;BRANCH ON CARRY TRUE
                              ;
                              ; CARRY = 0 INDICATES NEGATIVE RESULT
                              ;
 0009 3B80 A                  SFLG    SIGN               ;SET SIGN FLAG
 000A 7001 A                  CAI     R0,1
 000B E1F4 T                  ADD     R0,H9999           ;TAKE 9'S COMPLEMENT
 000C 8000 A                  RTS     0                  ;RETURN
                              ;
                              ; CARRY = 1 INDICATES POSITIVE RESULT
                              ;
 000D 7801 A     CTRUE:       AISZ    R0,1               ;ADD END AROUND CARRY
 000E 8000 A                  RTS     0                  ;RETURN IF RESULT NEQ 0
 000F 8000 A                  RTS     0                  ;RETURN IF RESULT EQ 0
                              ;
      0003                    .END    DECS
```

Table B–7. Binary Multiplication Examples

```
                         .TITLE  BIMULT, ' BINARY MULTIPLICATION '
                         ;
      0000               .ASECT
      0100               .=X'100
                         ;
      0000     R0        =       0
      0001     R1        =       1
      0002     R2        =       2
      0003     R3        =       3
      000A     CARRY     =       10
                         ;
 0100 FFFF A   CONST:    .WORD   X'FFFF           ;CONSTANT FOR DOUBLE-PRECISI
                                                  ;  ADDITION
 0101 5100 A   START:    LI      R1,0             ;CLEAR RESULT REGISTER
 0102 5310 A             LI      R3,16            ;LOOP COUNT TO AC3
 0103 7000 A             CAI     R0,0
 0104 6940 A   LOOP:     RADD    R1,R1            ;SHIFT RESULT LEFT INTO CARR
 0105 7400 A             RADC    R0,R0            ;SHIFT CARRY INTO MULTIPLIER
                                                  ;  AND MULTIPLIER INTO CARRY
 0106 4A02 A             BOC     CARRY,TEST       ;TEST FOR ADD
 0107 6980 A             RADD    R2,R1            ;ADD MULTIPLICAND TO RESULT
 0108 91F7 A             SUBB    R0,CONST         ;ADD CARRY TO HIGH-ORDER RES
 0109 7BFF A   TEST:     AISZ    R3,-1            ;DECREMENT LOOP COUNT
 010A 19F9 A             JMP     LOOP             ;REPEAT LOOP
                         ;
      0101               .END    START
```

```
                         .TITLE  BIMULT,   ' BINARY MULTIPLY'
                         ;
      0000               .ASECT
      0100               .=X'100
                         ;
      0000     R0        =       0
      0001     R1        =       1
      0002     R2        =       2
      0003     R3        =       3
      0003     BIT0      =       3
      0008     LINK      =       8
                         ;
                         ;
 0100 5100 A   START:    LI      R1,0             ;CLEAR RESULT REGISTER
 0101 5310 A             LI      R3,16            ;LOOP COUNT IN AC3
 0102 7000 A             CAI     R0,0             ;COMPLEMENT MULTIPLIER
 0103 4301 A   LOOP:     BOC     BIT0,SHIFT       ;TEST BIT 0
 0104 6980 A             RADD    R2,R1            ;ADD MULTIPLICAND TO RESULT
 0105 3800 A   SHIFT:    PFLG    LINK             ;CLEAR LINK
 0106 2503 A             ROR     R1,1,1           ;SHIFT AC1 INTO LINK
 0107 2C03 A             SHR     R0,1,1           ;SHIFT LINK INTO AC0
 0108 7BFF A             AISZ    R3,-1            ;DECREMENT LOOP COUNT
 0109 19F9 A             JMP     LOOP             ;REPEAT LOOP
                         ;
      0100               .END    START
```

Table B-8. Descriptions of Status and Control Flags

Register Bit	Flag Name	Description	Flag Code (fc)
0	High ('1')	Bit 0 is not used and is always in logic '1' state. Referencing bit 0 with SFLG or PFLG Instruction has no effect. (May be used as NOP Instruction.)	0000
1 2 3 4 5	IE1 IE2 IE3 IE4 IE5	Flags IE1 through IE5 serve as Interrupt Enable Flags for five of six PACE Interrupt levels. If Interrupt Enable is high and associated Interrupt Request occurs, microprocessor executes Interrupt Service Routine. If Interrupt Enable is low, associated Interrupt Request is ignored.	0001 0010 0011 0100 0101
6	OVF	Overflow Flag is set to state of twos-complement arithmetic overflow by arithmetic instructions. Overflow Flag is set high if sign bits (most significant bit) of two operands are identical and sign bit of result is different from sign bit of operands. If A, B, and R are sign bits of operands and result, then Overflow Flag is set according to equation $OVF = (\overline{A}, \overline{B}, R) + (A, B, \overline{R})$ Sign bit is most significant bit for data length selected; thus, if data length is 8 bits, then bit 7 is sign bit; if data length is 16, then bit 15 is sign bit. State of OVF Flag is affected by instructions ADD, DECA, SUBB, RADD, and RADC.	0110
7	CRY	Carry Flag is set to state of binary or decimal carry output of adder by arithmetic instructions. Carry output is derived from most significant bit for data length specified by BYTE Flag. State of CRY Flag is affected by instructions ADD, DECA, SUBB, RADD, and RADC.	0111
8	LINK	Link Flag is included in shift and rotate operations as specified by Shift and Rotate Instructions. Link Flag is unaffected if not selected.	1000
9	IEN	Master Interrupt Enable Flag simultaneously inhibits all five of lowest priority interrupt levels. No Interrupt Request is serviced unless individual Interrupt Enable Flag for associated Interrupt Request and master Interrupt Enable Flag are high. IEN Flag is set low every time any interrupt (except Level 0) is serviced. IEN Flag is set high by execution of Return-To-Interrupt Instruction (RTI).	1001
10	BYTE	BYTE Flag selects 8-bit data length when high and 16-bit data length when low.	1010
11 12 13 14	F11 F12 F13 F14	Flags 11 through 14 are general-purpose control flags. Flags 11 through 14 drive PACE output pins and may be used to directly control system functions.	1011 1100 1101 1110
15	High ('1')	Bit 15 is not functional and is always in logic '1' state. Addressing bit 15 with SFLG or PFLG Instruction sets the Level-0 Interrupt Enable high.	1111