A N E X O

DATOS TECNICOS DE EQUIPO Y ELEMENTOS UTILIZADOS

MICRO-CONTROLADOR 8748 INTEL

The 48-Series microcomputers, are true single chip microcomputers. The devices making up this family of microcomputers contain all of the functions on one chip that are normally performed by multi-chip systems. The various chips that constitute the 48-Series are similar, only de amount of internal RAM and ROM varies. The features contained in the 48-Series are given below. Included within these feateures are several transparent improvements over other similar devices.

* 8 bit CPU
* Built-in RAM and ROM (Externally expandable)
* 1.36 usec or 2.5 usec cycle times.
* Built-in oscillator and clock (Cristal controlled ext)
* 27 I/O lines (Expandable)
* 8 bit Timer/Counter
* Interrupt (Schmitt trigger with hysteresis)
* 96 Instructions (70% single byte)
* Single step
* Binary and BCD Arithmetic
* 8 level Stack
* Programable standby RAM
* Low voltage standby
* On-chip Standby Battery Charging

This extended and improved 48-series of microcomputers offers the user greater flexibility both during and after the development cycle. During development, three devices containing varying amounts of RAM, but no ROM, allow the user to develop an optimum system using as much external EPROM as is necessary.

Once de firmware is finalized, three mask-programmable devices, with varying amounts of RAM and ROM, can be substituted into the final system. The ability to substitute a single 48-series device for multiple devices permits a low-cost, low power alternative to potencially expensive aplications. Additionally, future increased firmware needs are handled by simply upgrading to a device with a greater ROM capacity.

The varying amounts of internal RAM and ROM for the 48-series microcomputers are given in table 1-1

Device	ROM	RAM		Common features
8035	---	64 bytes	!	8 bit CPU
8039	---	128 bytes	!	Reset
8040	---	256 bytes	!	On-board timing/control
8048	1 Kbyte	64 bytes	!	27 I/O lines
8049	2 Kbyte	128 byte	!	Timer/Counter
8050	4 Kbyte	256 byte	!	Interrupt

The instructions set makes the 48-series an efficient controller and arithmetic processor. With 96 instructions, 70% of which are one byte in length, the remainder being two bytes, efficient use is made of the system memory. The I/O lines can be individually set, reset or logically manipulated directly by the software. Additionally, a large set of branch and table look-up instructions provide for efficient logical operations. The internal timer/counter is also directly controlled through dedicated instructions.

48-SERIES ARCHITECTURE

There are nine major functional blocks comprising the 48-series microcomputers. These blocks are in turn composed of various sub-blocks. The various blocks are as follows:

* CPU
  - Instruction Register
  - Arithmetic Logic Unit
  - Accumulator
  - Flag Register

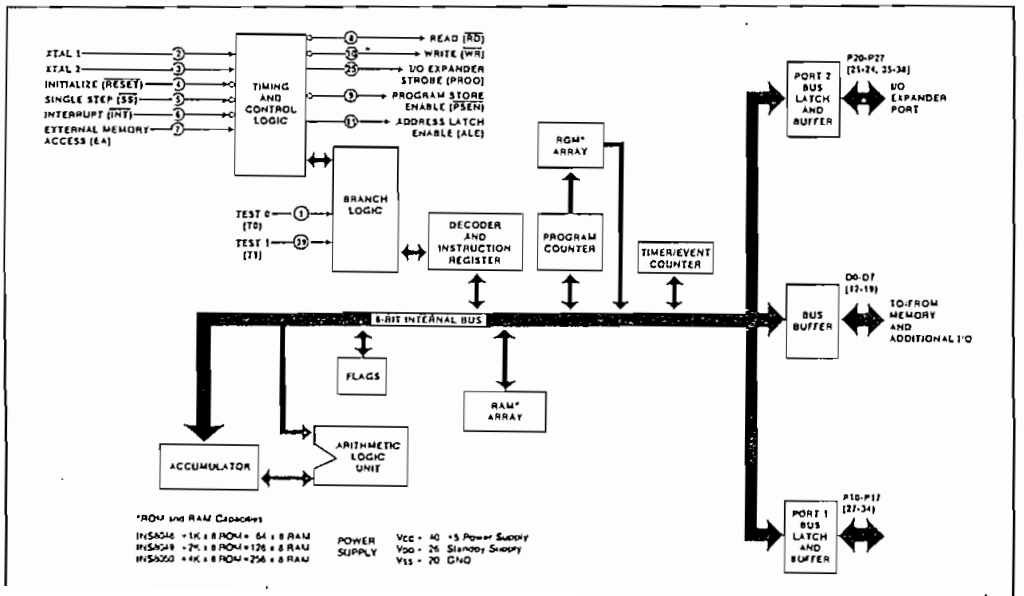
* Branch Logic

* Resident ROM
  - Program Counter

* Resident RAM
  - RAM
  - Registers
  - Stacks

* Input / Output Ports and Bus
* Instruction Register
* Program Status Word
* Internal Timer/Counter
* On-board Timing and Control Logic

A functional block diagram of the 48-series microcomputers is shown in figure 2-1



CPU

The CPU is an 8-bit unit comprised of an instruction register, an arithmetic logic unit, an accumulator, and a flag register. In a typical operation, data placed in the accumulator combines with data from another source on the internal bus. The result of the combination is then stored in either the accumulator or a designated register.

Instruction Register: The instruction register receives the operation code (opcode) portion of each instruction taken from ROM. The opcode then generates specific outputs to control each block of the CPU. The outputs typically control the source and the destination registers, as well as the function to be performed by the arithmetic logic unit.

Arithmetic Logic Unit: The arithmetic Logic Unit ALU operates on the 8-bit data taken from the accumulator and the bus. Operation is controlled by the instruction register and consists of the following operations:

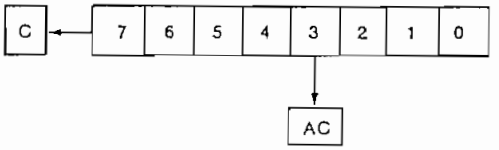
Add with or without carry
AND, OR, EXOR
Decimal Adjust
Increment, Decrement
Complement, Clear
Rotate right and left, with or without carry
Swap Nibbles

Any operation performed in the ALU that results in values greater than eight bits, causes the carry flag to be set.

Accumulator: The accumulator is the main imput port to the ALU. All operations are performed with reference to the accumulator. Most data transfers into the ALU from memory or the I/O port pass through the accumulator.

The accumulator is a register normally used for 8-bit operations. As such the accumulator maintains a carry bit immediatly following bit 7. Additionally 4-bit binary-coded decimal (BCD) operations can also be performed by the accumulator. To support the 4-bit operations, an auxiliary carry bit is maintained immediately following bit 3. Auxiliary carry is affected by the same logical and arithmetic instructions that affect carry. Carry is de only testable bit. Auxiliary carry is used only when converting the accumulator contents from binary to BCD (Using the DAA instruction). The auxiliary carry flag bit can be cleared by moving a zero into bit 6 of the program status word. An illustration of the accumulator is

shown in figure 2-2.



Branch Logic
The branch logic within the 48-series microcomputers permits the user to test various internal or external status conditions. If the selected condition is true, the branch logic forces a jump to the address specified by the program. The various branch condition tests are listed in table 2-2.

Table 2-2 48-Series Branch Conditions

Test	Logic Condition	Instruction
Interrupt	0	JNI
Flag 0	1	JF0
Flag 1	1	JF1
Timer flag	1	JTF
Carry	0 or 1	JC, JNC
Accumulator	0 or Non-0	JZ, JNZ
Accumulator bit test	1	JB0-JB7
Test 0	0 or 1	JT0, JNT0
Test 1	0 or 1	JT1, JNT1
Register	Non-0	DJNZ

Program Status Word
The Program Status Word is an 8-bit word stored in the program status register. The register contains both status information relating to machine operation, and the stack pointer. The contents of the register can be read from, or written to the accumulator. All eight bits must be read or written at the same time.

During subroutine calls or interrupts, the upper four bits of status register are saved on the stack. The contents may be restored to the register upon return, depending upon the return instruction used.

The stack pointer comprises the lower three bits of the program status word, and is an indepedent counter that points to designated spaces in the internal RAM.

The stack occupies RAM locations 8 through (X'17). When reset to zero, the stack pointer actually points to locations 8 and 9 in RAM. An interrupt or subroutine call causes the contents of the program counter and the upper 4 bits of the program status word to be stored in one of the eight stack registers. The stack pointer is then incremented to point to the next two locations.

Up to eight subroutines can be nested at any given time without the stack overflowing. Since the stack pointer is a simple up/down counter, an overflow will cause the deepest address to be lost (the counter overflows from 111 to 000). The pointer also underflows from 000 to 111.

Return from subroutines decrement the stack pointer with the contents of the register pair restored to the program counter and possbly the PSW. Depending upon the return instruction used, status may also be restored.

Note.- When the level of subroutine nesting is less than 8, the unused stack locations may be used as RAM.

An illustration of internal stack composition is shown in figure 2-3

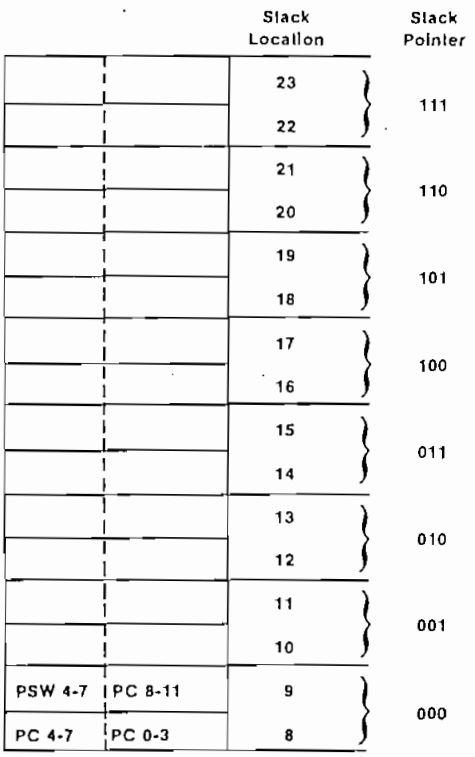


FIGURE 2-3. Internal Stack Composition

The program status word contents are given in table 2-3

Bit position	Contents
0	Stack pointer bit S0
1	Stack pointer bit S1
2	Stack pointer bit S2
3	Not Used
4	Register bank select bit 0=bank 0
5	Flag 0
6	Auxiliary carry
7	Carry

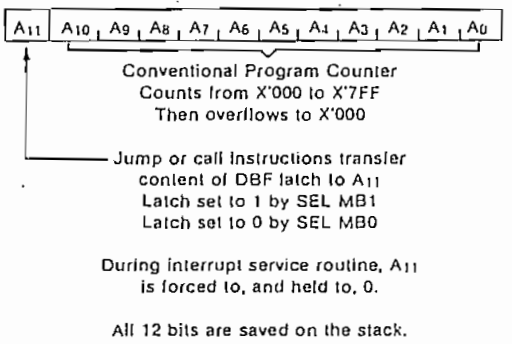
Program Counter

The program counter is an independet 12-bit counter. For normal operation, the counter operates as a sequential up-counter, the output of which generates addresses for ROM. Bit 11 of the program counter is set independently of the normal count sequence by the memory bank select instructions. In this manner, instructiones fetches above or below the 2K memory boundary are affected.

During interrupts or subroutine calls, the contents of the program counter are stored in one of the eight selected stack locations.

During external program fetches, the lower eight bits of the program counter are preset on the Bus port only during ALE, the upper four bits are held on port 2. The thus addressed instruction is taken in on the Bus port when the program store enable (PSEN) signal is active. The program counter is reset to zero (X'000) when the reset input (RESET) goes active.

An illustration of the program counter is shown in the following figure.



Resident ROM

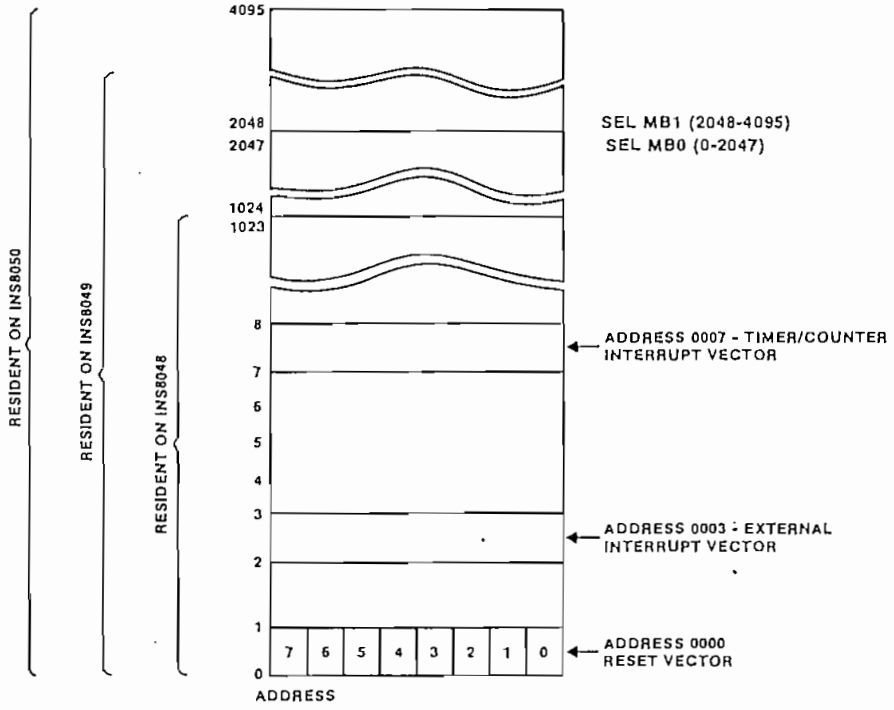
The on-board ROM (for those devices containing ROM) is an 8-bit mask-programmed ROM. Addressing the data or instructions within ROM is done by the program counter. Data or instructions output from ROM are placed onto de the internal bus. ROM addressing, up to a maximum of 4K, is done through the 12-bit program counter. The 8048 and 8049 automatically address external memory when their internal memory boundaries of 1K and 2K respectively are crossed.

There are three dedicated addresses within ROM to provide for system initialization or branching. These three locationes are described in table 2-4. An illustration of the internal ROM organization is shown in figure 2-5.

Table 2-4 Dedicated ROM Addresses

Address	Function
X'000	Reset. The reset input going low, forces the first instruction executed to be fetch from here.
X'003	Interrupt. The interrupt input going low (when interrupt enabled) forced the first instruction of an interrupt service routine to be fetched from here.
X'007	Timer/Counter interrupt. The timer/counter interrupt flag when set (if timer/counter interrupt is enabled) forces the first instruction of a timer/counter service routine to be fetched from here.

Figure 2-5 48-Series ROM Memory Map



Resident RAM

The resident RAM data memory is arranged as 64, 128 or 256 bytes, depending upon the device. There are eight working registers in RAM for each register bank (RB0 and RB1) selected. Register bank 0 occupies 0 through 7, while register bank 1 occupies locations 24 through 31. Indirect addressing to all RAM locations is implemented through the two 8-bit pointer registers, R0 and R1. The pointer registers occupy the first two working register locations: 0 and 1 for register bank 0, or 24 and 25 for register bank 1.

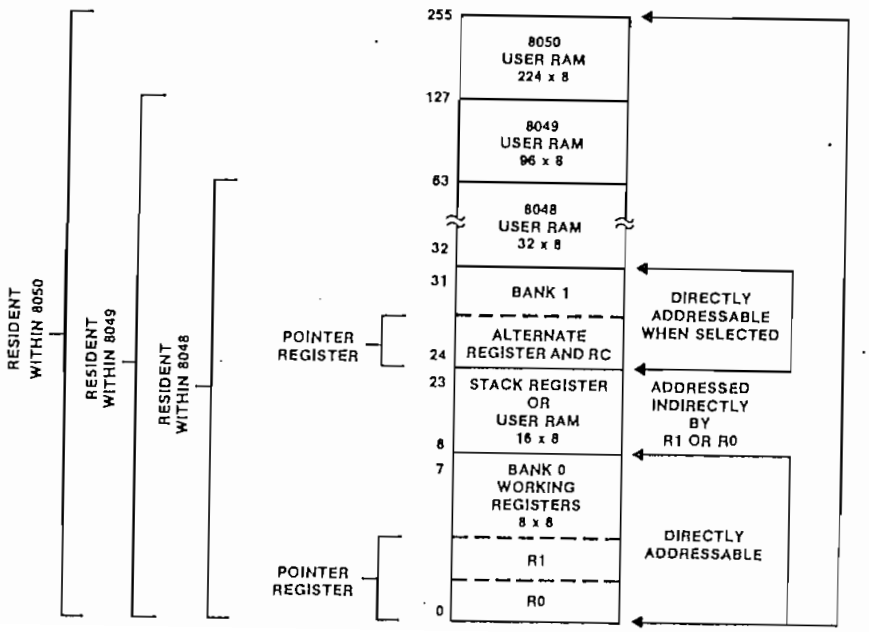
The eight level stack occupies the space between both working registers; locations 8 through 23. Each level of the stack actually accupies two memory locations.

Register bank 1 and any unused stack locations may be used for RAM if the register bank select feature and all of the stack are not used. Register bank 1 working registers may also be used as an extension of the register bank 0 registers, or reserved to service interrupts. The latter feature allows easy "saving" of register bank 0 registers by immediately switching to register bank 1.

Through the use of register bank-switching, the two pointer registers can be expanded to four, thereby allowing easy access up to four working areas. The 48-series RAM memory map is illustrated in figure 2-6.

Note: Internal and external RAM are data memory only. The 48-Series microcomputers cannot execute programs out of internal RAM. The 48-Series microcomputers can execute programs by accessing external memory. If the external access (EA) line is high, external memory may be ROM, PROM or RAM.

Figure 2-6 RAM Data Memory Map.



Internal Timer / Counter

The 48-Series microcomputers contain an internal timer/counter that may be operated in two modes, as a timer or as an event counter.

These operating modes are independent of the CPU, thereby providing the facility of accurate time delays without tying up processor time.

The register for the timer counter is an 8-bit, presettable up-counter. The register can be loaded or read using the MOV instructions to transfer data between the accumulator and the register.

Once started, the counter will count to its maximum value of X'FF, will overflow to X'00 and will continue counting until stopped by either a system reset or a stop instruction.

A system reset does not affect the contents of the register, although it wil stop the counter. The only method of changing the register contents is to load the register using MOV T instruction.

The counter may also be stopped using the STOP TCNT instruction. Once stopped the counter remains stopped until either a start counter. (STRT CNT) or start timer (STRT T) instruction is executed.

As the counter is incremented from X'FF to X'00, the timer overflow flag is set, and an unlatched interrupt request is generated. The flag may be tested by the JTF instruction. The request is honored only if EN TCNTI has been previously executed. The flag may be reset by either the JTF instruction or the Reset input going true.

Timer interrupt request are stored in a latch, the output of which is ORed with the external interrupt. The timer interrupt can be enabled or disabled independently of the external interrupt. If the timer interrupt is enabled, an overflow will cause a subroutine call to location 7 in ROM, where a timer or counter service routine will start. If both an external interrupt and a timer interrupt should happen to occur simultaneously, the external interrupt is recongnized first. Control will shift to location 3 to service the external interrupt. Since the timer interrupt is latched, once the external interrupt service routine is completled, control will return to the main program, at which time the timer interrupt will take effect. This time, control will shift to location 7 to service the timer interrupt. Once a subroutine call to location 7 occurs, the timer interrupt will be

reset. The interrupt can also be reset by the DIS TCNTI instruction.

Timer operation
For the timer/counter to operate as a timer, the STRT T instruction must be executed. Once executed, the instruction first clears and the causes the internal clock to pass through a divide-by-15 prescaler and a divide-by-32 prescaler. The second prescaler output increments the timer. By presetting the timer/counter prior to executing STRT T, accurate time-outs may be achieved.

As an example, assuming an 11 MHz crystal is used, the input is divided by 15. The resulting 733 KHz signal is in turn divided by 32. The final output of 22,917 Hz increments the counter once every 44 microseconds.

By pressenting the counter and detecting overflow, accurate timeouts between 44 microseconds and 11 milliseconds (256 counts) are possible. Timeouts longer than 11 milliseconds are possible by accumulating under software control, multiple overflows in one of the registers.

For times under 44 microseconds, the timer should be used as an event counter, with the external input taking the place of the internal clock source. Dividing the address latch enable signal (ALE) by three or more can serve as an external clock. Using the timer in this mode permits "finetuning" of timing delays through software looping.

Counter Operation
To operate the timer/counter as an event counter the STRT CNT instruction may be used. High-to-low transitions on the T1 input will increment the counter. The counter can not be incremented any more than once per three instruction cycles. Dividing the address latch enable signal (ALE) by three or more provides a convenient source for this timing. The input at T1 must also remain high for 500 nanoseconds after each transition.

A functional block of diagram of the timer/counter is illustrated in figure 2-7.

Timing and Control Logic
The timing and control logic internal to the 48-Series microcomputers permits the following.

- External stimulus to control system operation
- System communication with external memory
- Generate clock signals for internal use.

The various input and output signals that constitute the timing and control logic are listed in table 2-5

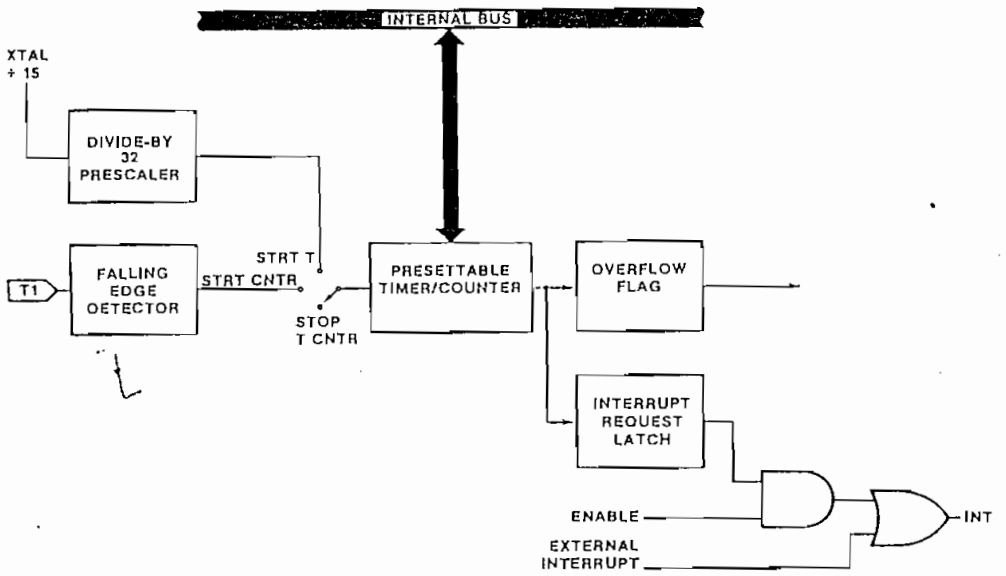


FIGURE 2-7. Timer/Counter Block Diagram

Table 2-5. Timing and Control Logic Signals

INPUTS	OUTPUTS
XTAL1, XTAL2 - Crystal, or Timing Source Inputs	RD - Read strobe
$\overline{\text{RESET}}$ - Initialize	$\overline{\text{WR}}$ - Write strobe
$\overline{\text{SS}}$ - Single Step	PROG - I/O expander strobe
$\overline{\text{INT}}$ - Interrupt	$\overline{\text{PSEN}}$ - Program store enable
EA - External access	ALE - Address latch enable

Internal Clock

The internal clock circuit of the 48-Series microprocessors accepts input from the two pins XTAL 1 and XTAL 2. A crystal or an externally generated clock source can be connected to these two inputs. The XTAL 1 pin (TTL compatible) is the input to a high gain series resonant circuit with a frecuency range of 1 to 6 MHz or 4 to 11 MHz, depending upon the 48-Series part used. The XTAL 2 pin is the output of the circuit providing feedback to the crystal. If accurate frequency references and maximum speed are not required, an inductor can be used in place of the more accurate crystal.

The external clock frequency of the oscillator is divided by three to provide the basic clock cycle for the system. Each clock cycle comprises a single machine state for the system. The basic clock cycle is available as an output at T0. Output is enabled by execution of the ENT0 instruction. Output is disabled whenever the system is reset.

There are five machine cycle times that comprise a single instruction cycle. Each of the five machine cycle time is in turn, comprised of a single clock cycle. The address latch enable signal (ALE) is provided as a continual clock output to enable the 48-Series microcomputers to communicate with external memory. This signal is also derived from the five basic clock cycles that comprise a machine cycle.

A functional block diagram of the clock circuit is shown in figure 2-8. Note that T0 is actually an input when the system is reset. Instruction cycle timing relationships are shown in figure 2-9. An illustration of the basic timing relationships between the clock output from T0 and other system signals is shown in figure 2-10. Instruction execution timing relationships are given in table 2-6.

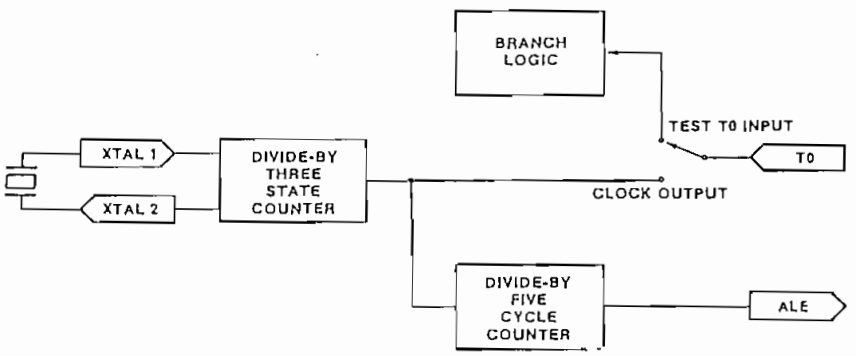


FIGURE 2-8. Internal Clock Block Diagram

Table 2-6 Instruction Execution Timing

Instruction Type	Byte 1 T1	T2	T3	T4	T5
IN A,P	Fetch	Inc PC		Inc Timer	
OUTL P,A	"	"		"	Out to P
ANL P,#NN	"	"		"	Read port
ORL P,#NN	"	"		"	Read port
INS A,Bus	"	"		"	
OUTL Bus,A	"	"		"	Out to port
ANL Bus,#NN	"	"		"	Read port
ORL Bus,#NN	"	"		"	"
MOVX @R ,A	"	"	Out RAM	"	Out to RAM
MOVX A,@R	"	"	Address	"	
MOVD A,P	"	"	Out Opcode	"	
MOVD P,A	"	"	"	"	Out data
ANLD P,A	"	"	"	"	"
ORLD P,A	"	"	"	"	"
J(Condition)	"	"	Sample Cond	"	
STRT T	"	"			Start Timer
STRT CNT	"	"			" Counter
STOP TCNT	"	"			Stop count
ENI	"	"		Enable I	
DIS I	"	"		Disable I	
ENT0 CLK	"	"		Enable Clock	

>>> BYTE 2 <<<

IN A,P	Read Port		
ANL P,#NN	Catch	Inc PC	Out to port
ORL P,#NN	"	"	"
INS A,BUS	Read Port		
ANL BUS,#NN	Catch	Inc PC	Out to port
ORL BUS,#NN	"	"	"
MOVX A,@R	Read Data		
MOVD A,P	Read P		
J(cond..)	Catch	Update PC	

Input/Output signal descriptions

All pins on the 48-Series microprocessors with the exception of the power and clock inputs, are input or output lines. The following sections describe in some detail the operation of these lines.

An illustration of the pin configuration for the 48-Series microcomputers is shown in figure 2-11. A summary of the pin functions is given in table 2-7.

Table 2-7 48-Series Pin Summary

1	T0	Testable input using JT0 and JNT0 instr
2	XTAL 1	Crystal input for internal oscillator
3	XTAL 2	" " " "
4	Reset	Reset input of CPU. Active Low
5	SS	Single step input
6	INT	Interrupt Input
7	EA	External access
8	RD	Read strobe
9	PSEN	Program store enable
10	WR	Write strobe
11	ALE	Address latch enable
12-19	DB0-DB7	Bus port
20	Vss	Circuit ground
21-24	P20-P23	Lower four bits of cuasi-bidirectional Port 2. Outputs upper four address bits during external ROM access and data for 8243 port expander.
25	PROG	Output strobe for 8243 I/O expander
26	Vdd	Provides an input for standby power
27-34	P10-P17	Port 1, quasi-bidirectional
35-38	P24-P27	Upper four bits of port 2
39	T1	Testable using JT1-JNT1, or event counter using STRT CNT
40	Vcc	Main power source (+5V)

Reset

The reset input initializes the processor. The input is a Schmitt trigger that has an optional internal pullup resistor. An external 1-microfarad capacitor tied to this pin assures the reset input will be low as Vcc is brought high. If an external reset is used, the input input must be held at ground for at least 50 milliseconds after the power supplies have stabilized. If the power and oscillator have already stabilized the reset need only be negative true for five machine cycle times. A list of internal functions reset by this signal going low is given below:

- Program counter set to zero.
- Stack pointer set to zero
- Memory bank 0 and register bank 0 selected
- Bus set to tri-state mode (except when EA is true)
- Ports 1 and 2 to input mode
- Interrupts disabled (timer/counter and external)
- Timer stopped
- Timer flag cleared
- F0 and F1 cleared
- Clock output at T0 disabled: T0 becomes testable input

Two typical reset circuits are illustrated in figure 2-12 internal and external reset options.

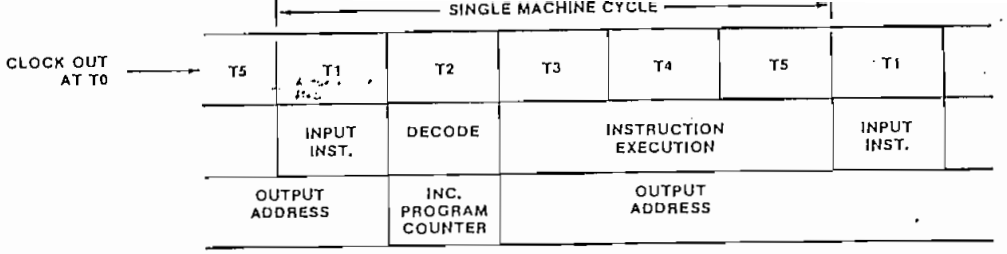


FIGURE 2-9. Instruction Cycle Relationships 87-5

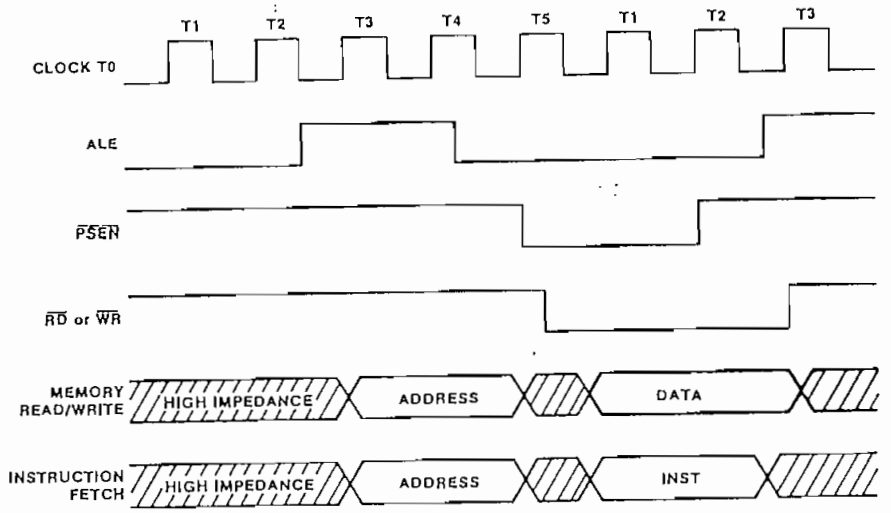


FIGURE 2-10. Timing Relationships 87-6

Single Step

The single step input, in conjunction with the ALE strobe allows the processor to step through ROM, logically performing one instruction at a time. If the instruction is two cycles in length, both cycles are executed and the processor then stops. Instruction execution can be easily followed as the address of the next instruction to be executed is output on both Bus and the lower 4 bits of Port 2. The bus buffer contents are lost while the processor is stopped. If necessary the Bus data can be latched externally on the leading edge of ALE and thus saved.

The single step operation is given in the following steps. Reference should be made to the timing chart of figure 2-13

1.- Single step goes low, requesting the processor to stop.

2.- The processor stops during the instruction fetch of the next command. If a double-byte instruction, both bytes are taken, following which the processor stops. ALE goes high as an acknowledgement.

3.- At this point, the address taken from the program counter is output on both the Bus and Port 2. This state can last indefinitely. Bus and port bit composition is illustrated in figure 2-14

4.- When single step goes high, the processor is allowed to continue.

5.- The processor acknowledges by driving ALE low.

6.- In order to stop the processor at the next instruction, SS must be driven low immediately after ALE goes low. If SS remains High, the processor will continue to run.

Implementation of a single-step circuit for the 48-Series requires a minimum of components. An illustration of such a circuit is shown in figure 2-15 Circuit operation is as follows.

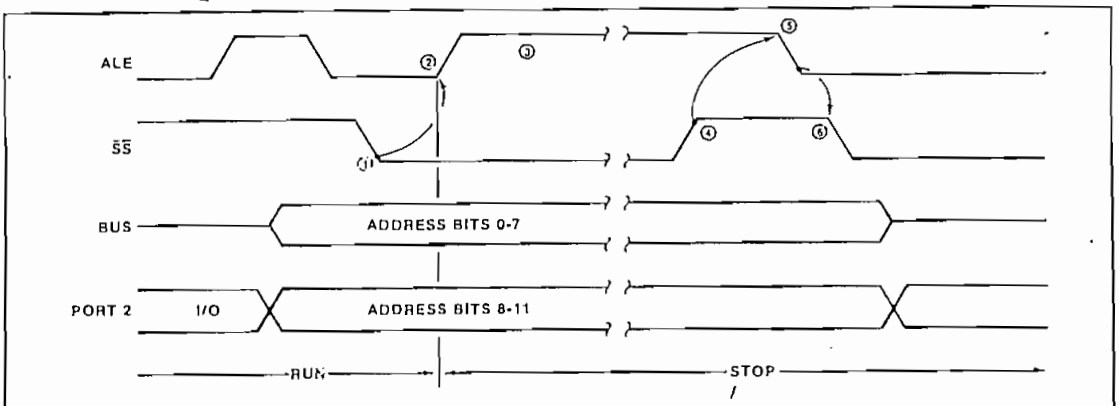


FIGURE 2-13. Single Step Timing

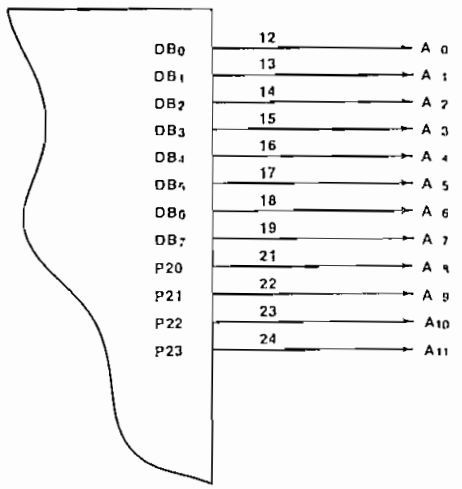
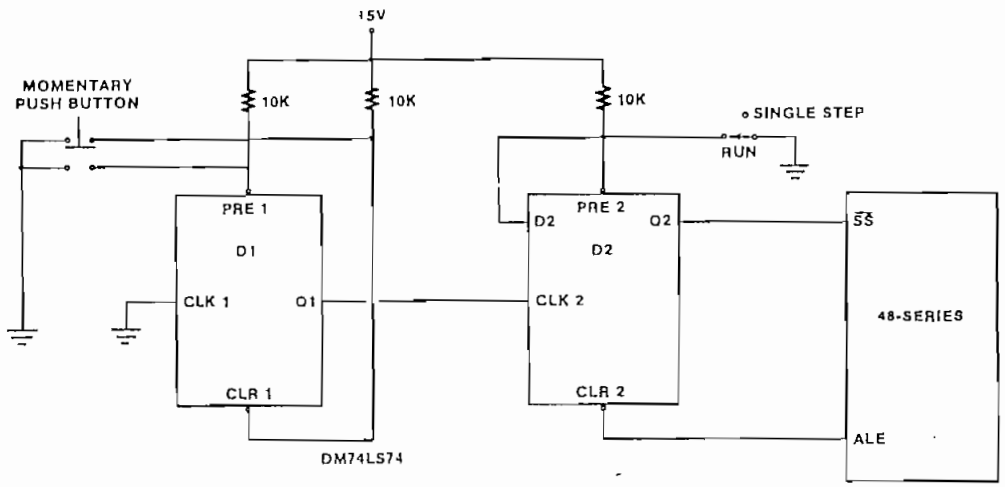


FIGURE 2-14. Address Output Lines



During normal operation the preset input (PRE2) of the DM74LS74 is held low, forcing the Q2 output high, thereby holding SS high. As long as SS remains high, the computer will continue to run. Switching the run/single step switch to the single step position puts a logic 1 on PRE2. As soon as ALE goes low, SS will be forced low, causing the computer to stop. The next instruction is executed by pressing the momentary switch. If ALE is high, the debounce flip flop (D1) clocks a 1 into D2. As soon as SS goes high, the computer fetches another instruction that brings ALE low. When ALE goes low, D2 is cleared, SS goes low and the computer stops. Placing the run/single-step switch back in the run position forces a 1 to SS, allowing the computer to resume execution.

Interrupt (INT)

The interrupt input, when enabled will initiate an interrupt. This input has a Schmitt trigger input, with hysteresis, that is active low. If the interrupts are disabled, the INT line can still be sampled by a conditional jump instruction. When an interrupt is detected, a jump-to-subroutine at location 3 (in internal ROM) occurs as soon as the current instruction completes execution.

During interrupts, the program counter and upper four bits of the program status word are stored on the stack. Control is transferred to location 3 in ROM, which should normally contain a jump to interrupt service routine. Once completed, the interrupt service routine should end with a return-and-restore-status instruction (RETR).

Being a single-level interrupt. INT is disabled while servicing an interrupt, and in re-enabled by the RETR instruction. Internal Timer/Counter interrupts are treated in like manner. If both an external and a timer/counter interrupt occur at the same time, the external interrupt has priority.

A simple programming trick allows the timer/counter to function as a second external interrupt, if desired. By loading X'FF in the timer and putting the timer/counter in the event counter mode, a logic 1 to logic 0 transition at input T1 will cause the counter to increment and overflow creating an interrupt vector to location 7 in ROM.

Interrupts are disabled by either system reset or the disable interrupt instructions: DIS I for an external interrupt, DIS TCNTI for a timer/counter interrupt. The interrupts must be enabled by the program for them to

function. ENI for an external interrupt, EN TCNTI for a timer/counter interrupt. Interrupt sampling occurs each machine cycle during ALE. An interrupt request must be removed before ending the interrupt service routine. If not removed, the processor will immediately re-enter the service routine. A selected output line from the 48-Series microcomputers could be designated an interrupt-acknowledge line. This line can then be activated during the interrupt service routine to reset the requesting interrupt.

The INT input may also be tested by the jump-if-interrupt-is-low instruction (JNI). If INT is left disabled, this input can be tested in the same manner as inputs T0 and T1. An illustration of the internal interrupt structure is shown in Figure 2-16.

External Access (EA)

When the external access input is driven high, the microcomputer performs all memory fetches from external ROM, internal ROM is disabled. Normal 48-Series usage would have the user program stored in internal ROM. As an example, diagnostic routines to test the internal logic of the CPU could be contained in the external ROM. EA should be driven high only while RESET is low.

The internal ROM may also be read, independent of the CPU using the EA input. The sequence for reading internal memory is as follows:

- RESET is driven to 0 volts
- CPU is placed in the read mode by driving EA to +12V
- Address to be read is placed at the BUS and Port 2 lines (P20 through P22)
- RESET goes high, at which time the addresses are latched
- After the addresses are latched, RESET remaining high will cause the contents of the addressed location to be output on the BUS lines.

Timing for reading internal ROM is shown in Figure 2-17

An illustration of a typical read circuit is shown in Figure 2-18.

Test 0 (T0)

Pin T0 is a dual purpose pin, in one state it is a testable input, in another state it is a clock output. Each state is under direct software control.

After power-on reset, the T0 pin is a testable input using either the JNT0 or JT0 instructions. Depending

upon the state of the input, a jump to specified address will occur.

The T0 pin is designated as a clock output by the ENT0 instruction. Once designated a clock output, T0 can only be returned to the testable state by activating RESET.

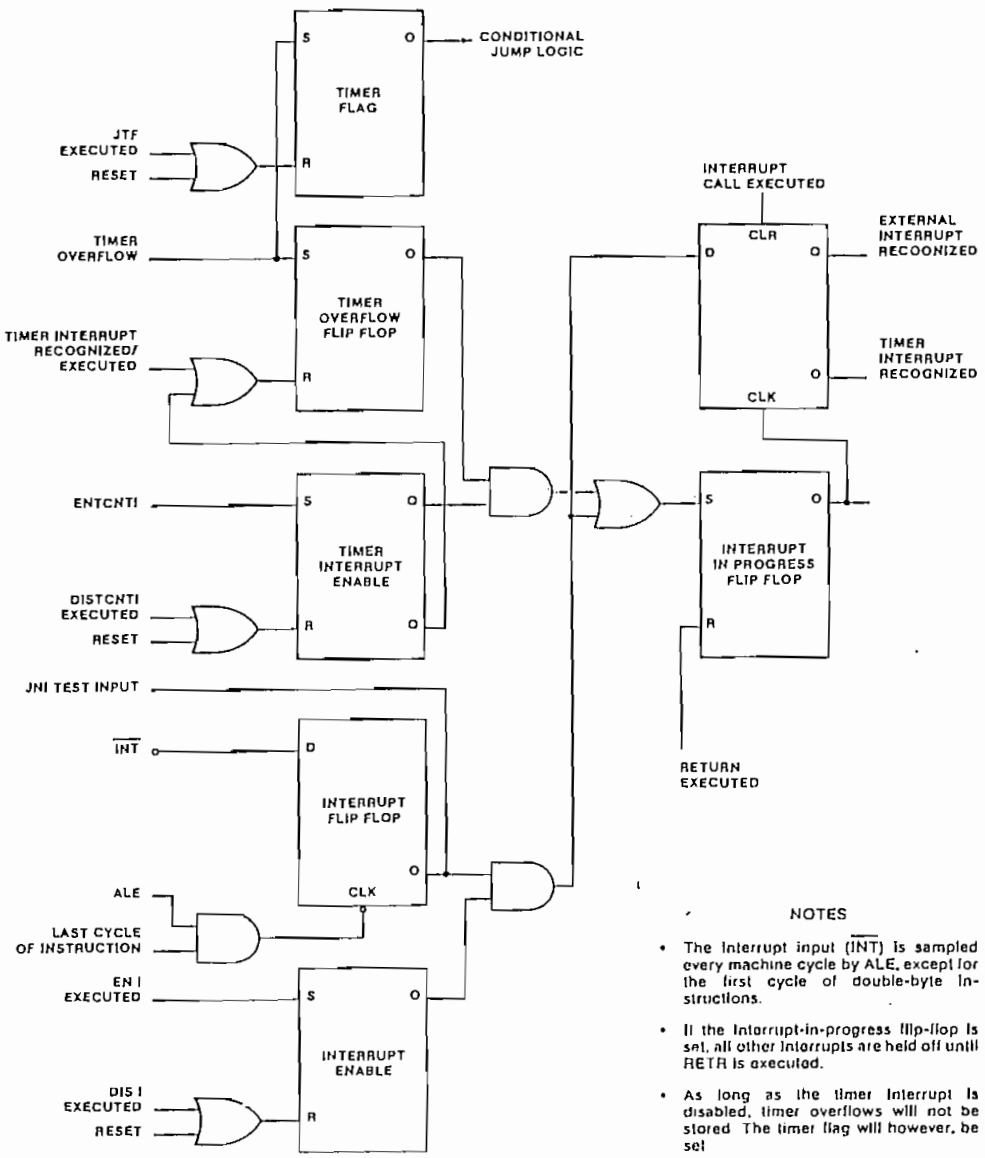


Figure 2-16 Internal Interrupt Structure

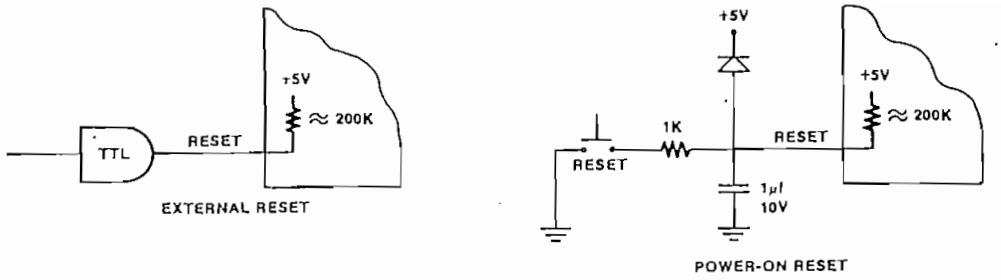


FIGURE 2-12. 48-Series Reset Circuits 87-7

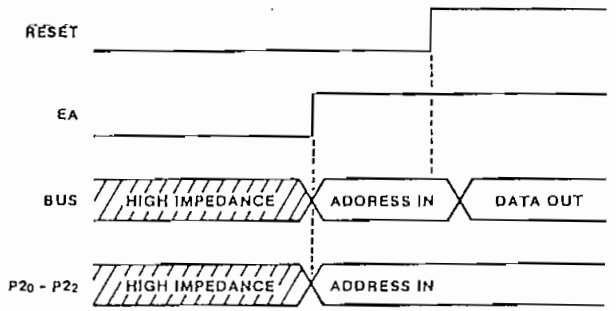


Figure 2-17 Timing for reading internal ROM

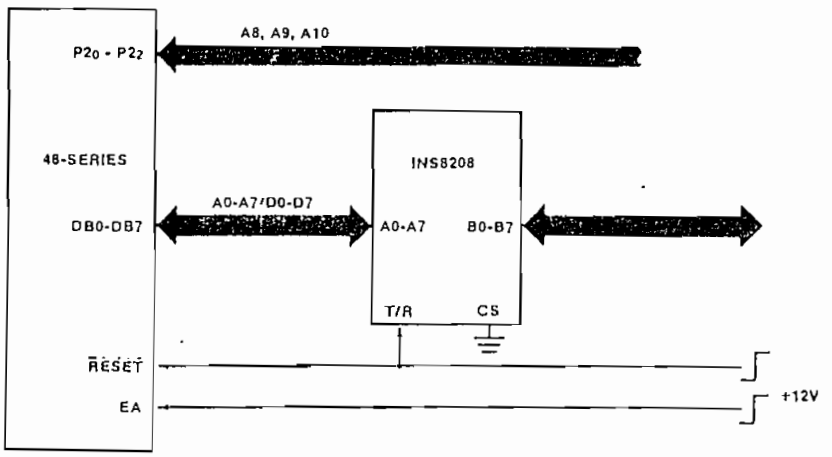


Figure 2-18 Reading Internal ROM

Test 1 (T1)

Pin T1 is also a dual purpose pin, in one state it is a testable input, in another state it is the input to the event counter. For each case, the input function is software controllable. As a testable input, T1 is tested by either the JNT1 or JT1 instructions, with a jump to a specified address occurring if the tested state is true.

The T1 pin is designated as the event counter input by the STRT CNT instruction. Thereafter high-to-low transitions at T1 will increment the counter.

Address Latch Enable (ALE)

The address latch enable strobes permit external latching of the address bits present on the Bus outputs and Port 2, bits 0 through 3. The ALE signal occurs once each machine cycle, with the address valid on the falling edge of ALE. The signal ALE may also be used as a clock signal to enable or disable the single step input. Address latch enable is used in conjunction with the program store enable for external fetches, and the read and write strobes for data accesses from RAM or peripherals.

Write Strobe (WR)

The write strobe is driven low whenever the processor performs an external Bus write operation. Data output occurs on the Bus port with the WR strobe writing the data into external RAM. This signals operates in conjunction with ALE strobe during MOVX @R,A instructions. The WR signal is also active during an OUTL BUS,A instruction where it can be used to notify a peripheral that new Bus port data is available.

Read Strobe (RD)

The read strobe is driven low whenever the processor performs an external Bus read operation. The RD strobe enables data from external RAM or peripherals onto the Bus, at which time it is read into the accumulator. This signal operates in conjunction with the ALE strobe during MOVX @R,A instructions. The RD signal is also active during an INS A,BUS instruction where it can be used as either an interrupt acknowledge or a flag to notify the peripheral that the CPU has read the data.

Program Store Enable (PSEN)

The program store enable signal is used to fetch instructions for the CPU. The PSEN strobe is active only when fetching instructions from external ROM. This signal operates in conjunction with the ALE strobe.

Output Strobe (PROG)

The output strobe PROG is the output strobe for the 8243 I/O port expander. A high to low transition on PROG indicates that an address and instruction is available on Port 2, bits 0 through 3. A low to high transition on PROG indicates that data is present on Port 2 for a write. Data must be valid during the low-to-high transition for a read.

Crystal Input (XTAL 1 , XTAL 2)

These two pins on the 48-Series microcomputers allow either a series-resonant device such as a crystal or inductor; or an independent clock source to connect to a high gain internal amplifier. XTAL1 (pin2) is the input to the amplifier, XTAL2 (pin3) provides feedback. The resonant frequency of the circuit is divided internally to created the basic clock cycle as output at T0.

I/O Ports and Bus

There are 24 input/output lines and three test inputs in the 48-Series microcomputers. The I/O lines are organized as three 8-bit ports. The ports can be either input, output or bidirectional. Port 1 and 2 are especially versatile in that different types of outputs may be intermixed.

Port 1 and 2 differ from Port 0 (the Bus) in that they are quasi-bidirectional while the Bus is a true bidirectional port. This means that they can be used as inputs or outputs while being statically latched. To further explain this, if a 1 is written into any bit of port 1 or 2, that bit can function as an input or a high level output. If a 0 is written into any of this bits, that bit can only function as a low level output. This type of I/O pin is better understood as an open drain output with a large value internal pull-up resistor connected to an input latch. Data is latched in this ports from the CPU and will remain latched until changed. As inputs, these ports are non-latching, and must be read by an input instruction prior to removing the input.

The quasi-bidirectional output structure permits each line to serve as an input, an output or both, even

though the outputs are statically latched.

The lower four bits of port 2 fulfill three distinct functions:

- A quasi bidirectional static port
- A portion of the external ROM address
- An expander port

For all three functions, the output are driven low by an active device or momentarily pulled high by an active device, then held high by a passive device.

This port may contain latched output data yet still be used in another mode without affecting operation of either mode. if the lower four bits of port 2 outputs are the address for an external instruction fetch, the previously latched data will be temporarily removed from the output. If needed the data can be externally latched on the rising edge of ALE. The address for external ROM is output and once the instruction fetch is completed, the latched data is restored. If port 2 is used as a bus to an expander port P20-P23 will contain the value output to the 8243 expander device. After an input from 8243 the port will remain in the input mode. For either case, previously latched data will be destroyed.

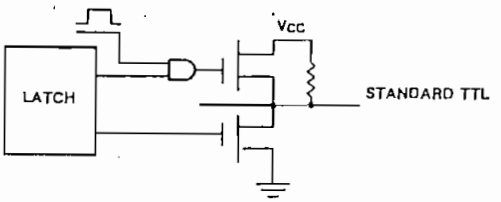
An illustration of the port structure for a standard TTL output is shown in Figure 2-19

Each output is pulled to Vcc through a resistor or moderately high impedance. This pullup provides sufficient current for providing a TTL logic "1", yet can be pulled low by a standard TTL gate output. This ability provides for both input and output on the same pin. For fast '0' transitions, a relatively low impedance device (~5K) is switched on momentarily when a 1 is written to the line. When a 0 is written to the line, a low impedance device (~ 300 ohm) overcomes the 5K pullup and provides TTL current sink capability.

Note: Since the pulldown device is such a low impedance, a "1" must be written to the line prior to any input. A system reset (RESET going true) drives all lines to a high impedance "1".

Port 1 and Port 2 when used with the ANL and ORL instructions, provide an efficient means for handling single-line inputs and outputs.

Figure 2-19 48-Series I/O Port Options



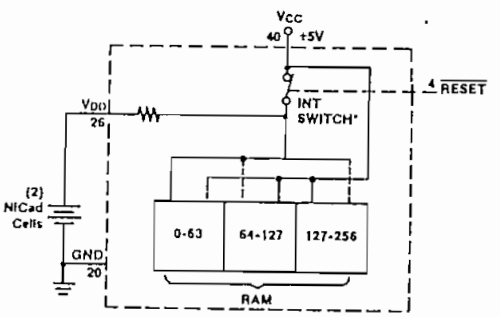


FIGURE 2-20. Internal Charging/Standby Circuit

The Bus port is a true bidirectional port that can be statically latched or used synchronously. When used as a bus, valid input or output must occur during the read or write strobes. During external instruction fetches, the eight low order address bits from the program counter are preset at this port. The fetched instruction must be present at the input when the program strobe (PSEN) goes high. During external RAM operations, addresses and data are output from this port under control of address latch enable (ALE) and read (RD) or write (WR) strobes. When not being written to or read from, the Bus lines are in their high impedance mode (unless previously latched).

As a static output port, data is written and latched using the OUTL instruction. Data is input using the INS instruction. Both instructions generate pulses on the corresponding WR and RD output strobe lines (OUTL generates a WR pulse, INS generates a RD pulse).

The Bus port is not bit-programable as are the other two ports (P1 and P2). Once written to, using an OUTL instruction, the bus will remain an output port until either the device is reset or the bus is used for an external memory access.

SPECIAL FEATURES

All 48-Series microcomputers contain additional features not included in other similar devices. These features are totally transparent to the user if 48-Series microcomputers are substituted for other devices. There is no adverse effect upon system operation.

These added features are:

- Schmitt trigger (with hysteresis) input for interrupt.
- Built-in battery charging circuit.
- Varying amounts of standby RAM (Mask programable optio can alter standby current requirements.
- Mask-programmable pullup resistors on certain inputs.

BATTERY CHARGING CIRCUIT

The internal battery charging circuit is simply as a solid state switch between Vcc an internal resistor, and Vdd. During normal operation RESET is high holding the switch in the closed state, thereby providing power for the internal RAM. Vcc supply also provides the charging current for the external battery.

In the event of a power failure, RESET must go low before Vcc drops below 4.5V. By going low RESET

inhibits any RAM access and open the internal switch. As soon as the switch es opened, current flow reverses and what was originally a charging output for the batteries, become a source of standby power for the internal RAM. Being an XMOS device, the internal RAM only requires a Vdd of 2.2 Volts (2 Ni-Cad cells) to sustain data. The built-in charging circuit, and low stand-by voltage requirements eliminates the need for both an external charging circuit and five Ni-Cad cells. An illustration of the internal charging circuit is shown in figure 2-20

A typical sequence of events leading up to standby mode of operation would be as follows.

- An imminent power supply failure is detected
- An interrupt is generated to vector operation to a standby service routine
- All important data and status is stored in RAM
- RESET goes low, inhibiting any RAM access and places RAM on standby power

Power supply failure detection is the most critical portion of the standby operation. It is important that an imminent failure be detected in time to save all critical data or status.

The simplest and most effective detection circuit is a voltage comparator that monitors the DC supply. The battery back-up can supply the reference voltage while the scaled down unregulated portion of the supply provides the test voltage. Since DC regulators typically fall out of regulation when their input approaches within 2 volt of the output, the comparator can sense this voltage change long before the regulated output drops.

An imminent power failure generates an interrupt causing program operation to branch to a status save routine. The status save routine places all data critical to system operation in the standby RAM locations, thereby assuring continuation of the main program when power is restored.

System operation is such that once an interrupt is generated, the interrupt input is disabled until re-enabled by the enable external interrupt (ENI) instruction. Saving of status of thos while preventing multiple interrupts from a fluctuating power supply.

The reset input should be driven low by the firmware to

prevent undesirable system operation during power down. It is important that the state of the power supply be checked again before issuing a reset to the microprocesor. The jump if negative interrupt (JNI) instruction can recheck the interrupt input without creating an interrupt. If the power supply only fluctuated slightly, this test permits the program to jump around a firmware reset and restore normal program operation. If the power fail signal is still active at the interrupt input, the reset input should be driven low. An effective method is to drive RESET low under firmware control as the final task of the interrupt service routine. If one of the output port bits is used to gate the interrupt into the reset input, the microprocessor itself can drive RESET low.

NOTE:- Since RESET sets all ports high, an active high input should be used to gate INT into RESET, otherwise RESET dissapears immediately following execution of the instruction that caused it. This also means the bit must be initialized to a zero as part of the power-on sequence.

If power is in fact removed, the next power on reset forces the first instruction to be fetched from location 0. Therefore it is the responsability of the initialization sequence to determine whether or not power is being applied for the first time or being reapplied following a power loss. If it is a reapplication of power, program status must be restored and control returned to the interrupt program.

An illustration of typicall standby sequence timing is shown in figure 2-21.

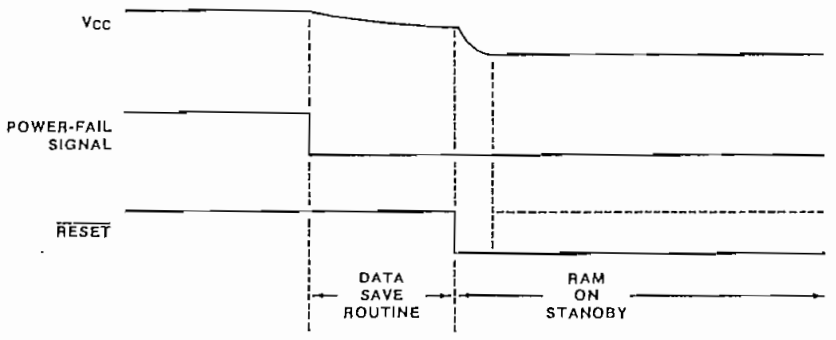


Figure 2-21 Standby Sequence Timing

INPUT / OUTPUT EXPANSION

The 8243 an input/output port expansion device, increases the capability of the 48-Series from 24 lines to 40 lines. The 8243 connects directly to the lower four I/O lines of Port 2 and serves to expand those four I/O lines up to 16 lines. An illustration of the 8243 connected to to 8748 is shown in figure 3-6. The added four ports have their own dedicated instructions and are addressed as ports 4 through 7. The lower three ports are the Bus (Port 0), I/O Port 1 and I/O Port 2.

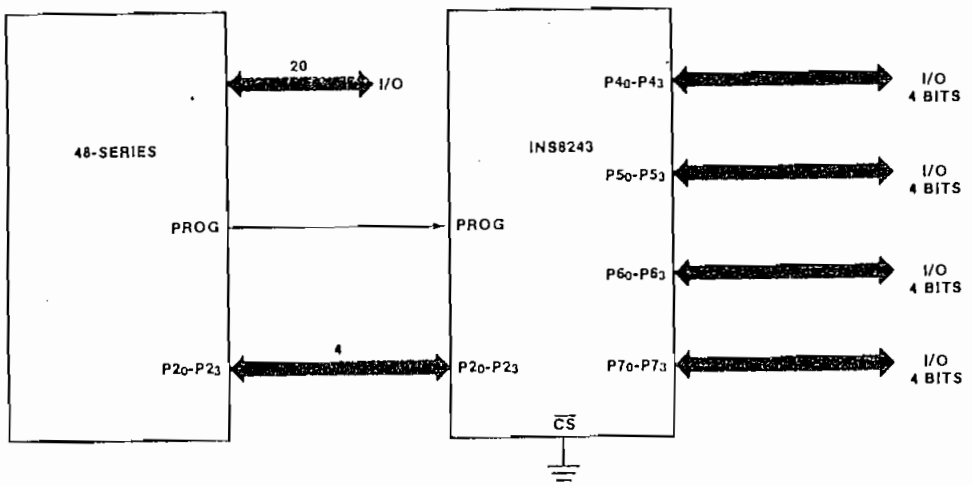


Figure 3-6 I/O Port Expansion

Data transfer is controlled directly by the 48-Series I/O enable signal PROG. As is shown in the output expander timing diagram of Figure 3-7, there are two 4-bit cycles to I/O port selection port control and data. These cycles occur on the leading and trailing edge of the PROG pulse, respectively - during the second cycle of the instruction.

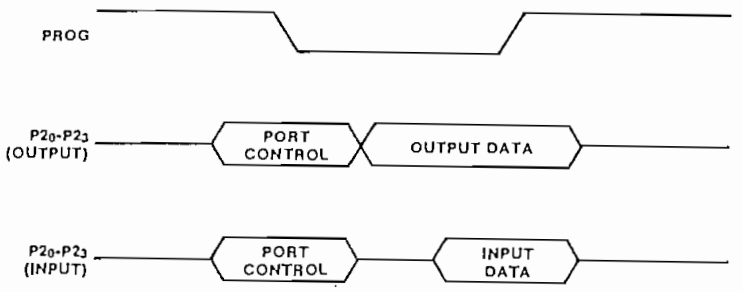


Figure 3-7 Port 2 Expansion Timing

The first or port control cycle is comprised of two 2-bit portions. Bit 0 and 1 select the port while bits 3 and 4 determine the function. The second cycle is comprised of 4-bit data. A detailed breakdown of 8243 port selection is given in figure 3-8

Additional expansion ports can be added to the port expansion 'bus'. By using the upper four bits of Port 2 as chip select signals four 8243s can be added for a total of 64 I/O ports.

I3	I2	Function	P1	P0	Port
0	0	Read	0	0	4
0	1	Write	0	1	5
1	0	OR	1	0	6
1	1	AND	1	1	7

Figure 3-8 8243 Selection Requirements

The 48-Series microcomputers are expandable beyond simple I/O expansion. There are numerous components that may easily attach to the bus. By using the strobes and test inputs, such varied devices as A/D or D/A converters, keyboard and display controllers or floppy

disk controllers can operate with the 48-Series microcomputers.

A list of the various control signals used with I/O components is given in table 3-1. The listed signals are inactive at all times other than for the given instructions.

Signal	Active During
RD	MOVX A,@R or INS BUS
WR	MOVX @R,A or OUTL BUS
PROG	MOVD A,P ; ANLD P,A
	MOVD P,A ; ORLD P,A
PSEN	External instruction fetches MOVP,MOVP3
ALE	Every machine cycle

TABLE 3-1 I/O Control Signals

Signal	Active During
$\overline{RD}$	MOVX A, @R or INS BUS
$\overline{WR}$	MOVX @R,A or OUTL BUS
PROG	MOVD A,P : ANLD P,A ; MOVD P,A ; ORLD P,A
$\overline{PSEN}$	External Instruction fetches; MOVP, MOVP3
ALE	Every machine cycle

THE 48-SERIES INSTRUCTION SET

INTRODUCTION

The assembly language instruction set for the 48-Series microcomputers provides the following types of operations:

- Control
- Data Move
- Timer/Counter
- Accumulator
- Branch
- Input/Output
- Register
- Subroutine
- Flags
- Miscellaneous

The instructions statements when assembled, generate object code. The object code in turn, define the precise operation the 48-Series microcomputer perform.

The 48-Series instruction set contains a total of 96 instructions designed for easy-of-use and to be memory efficient. The instructions are either one or two bytes in lenght, will over 70% of the instructions one byte in lenght. The double-byte instructiones incluide all immediate instructions, certain I/O instructions (excluiding MOV Pp,A) and most jump instructions.

Instruction execution occurs in either one or two machine cycle, with over 60% of the instruction executing in one machine cycle. Typical execution times when using an 11 MHz crystal are 1.36 usec and 2.72 usec for one and two bytes instructions, respectively.

The 48-Series microcomputers are afficient in handling both binary and BCD arithmetic operations. Additionally the 48-Series can easily manipulate single-bits for control operations, as well as supply special instructions to handle loop counters, table look-up and N-way branches.

Control Instructions

The control instructions allow the program to control interrupt, memory bank selection, and internal clock output.

Following initial power-on, the interrupt input (INT) is automatically disabled. The external interrupt input can then be enabled or disabled using two of the control instructions. Additionally INT is disabled while an interrupt is being processed and in re-enabled once the

interrupt routine is completed.

The four bank select instructions designate which bank of internal memory is accessed, two banks for ROM and two for RAM. The working register bank select instructions allow the programmer to substitute another bank of registers for the one presently in use. The bank select instruction provide both an effective 16 working registers and a method for saving register contents on an interrupt or subroutine call. For the latter scheme the user has the option of switching or not switching banks on interrupts. Bank select status as a part of the program status word, is saved on the stack during subroutine calls. Restoration is optional depending upon the return instruction is used. If the switch on interrupt is used, use of the "return and restore" status instruction to complete the interrupt service routine will automatically restore the originally selected bank.

The enable clock output instruction enables the internal clock to pin T0. Before this instruction is used T0 is an input testable by the JT0 and JNT0 instructions. The internal clock frequency is the XTAL input frequency divided by three. The resultant output can then be used as a general purpose clock for the remainder of the system.

Note: After the enable clock output instruction has been executed, the clock output at T0 is disabled only when RESET goes low.

Data Move Instructions

The data move instructions are the primary instructions for moving data back and forth between the accumulator, various registers or system memory. Data transfers from the registers 0-7 is direct (the instruction specifies the source or destination register). Data transfers from internal or external RAM is done indirectly via an address put in either registers R0 or R1 of the active bank. Transfers to or from internal RAM take only one machine cycle. Data stored in the internal ROM can also be loaded directly into the accumulator. Additionallt data transfer can be made directly between the accumulator and either the internal timer/counter or the program status register. The ability to change contents of the program status register provides an alternate means of restoring status after an interrupt or altering the stack pointer if necessary.

There is a data move instruction (XCHD A) that working in conjunction with the SWAP A instructions, permits easy handly of 4-bit quantities, incluiding BCD numbers. This instructions exchanges the lower 4 bits of any

internal RAM location. When used with SWAP A, this instructions makes it easy to handle 4-bit values, incluiding BCD numbers.

Timer/Counter Instructions
The timer/counter instructions enable the on-board 8-bit timer/counter, move data between the accumulator and the timer/counter, and start or stop the timer/counter.

The timer/counter can be used as either a crystal controlled timer using the internal clock as its clock source, or an event counter (or timer) taking its input from the T1 input. Either application can be selected under direct software control. The start instruction used will determine which clock source is used.

The timer/counter can be loaded or read from the accumulator while the counter is either running or stopped. Regardless of the timer/counter operational mode, the stop count instruction will stop the timer/counter. Additionally two instructions enable or disable the timer/counter interrupt flag for timer/counter output.

Accumulator Instructions
The accumulator instructions provides for adding data to (with or without carry) or for performing logical funtions with the accumulator. These operations of ADD, AND, OR, XOR or rotates are performed upon immediate data, data memory of the selected bank of registers. Data is moved between the accumulator and the various registers or memory depending upon the instruction used. A special instruction SWAP A, allows the position of the two 4-bit nibbles in the accumulator to be swapped. This instruction is used in conjunction with the XCHD A instruction.

A decimal adjust instruction has been incluided to facilitate BCD arithmetic. This instruction correct the binary adicion of two 2-bit BCD numbers in the accumulator to produce the correct BCD number.

Additional accumulator operations consist of rotates right and left (with or without carry), complement, increment, decrement or clear. The rotate instructions are performed one bit at a time.

Although there is no subtract instruction in the 48-Series instruction set, substraction can be performed using a simple three-byte set of instructions. The result of the substraction will be retained in the accumulator, the instruction string is:

```
CPL A     ; Complement the accumulator
ADD A     ; Add the value to the accumulator
INC A     ; Add 1 to the accumulator
```

The contents of the program status word can be pulled into the accumulator for modification using the MOV A,PSW instruction. Once modified the new contents can be placed into the PSW using MOV PSW,A

Not only does this permit modification of the PSW flags, but the stack pointer contents may also be modified. By modifying the stack pointer, the contents of various locations within the stack may be used by interrupts, or call and return instructions.

If a program is written that uses the contents of the stack pointer a MOV A,PSW instruction must be used. Following the MOV A,PSW the accumulator contents must be shifted left by one and two substrated (or added depending upon which next location is to looked at) from the contents for the actual memory location within the stack.

Branch Instructions

The jump instructions allow jumps (unconditional and conditional) throughout memory.

The unconditional jump instruction permits jumps to anywhere within the lower 2 K of ROM. To jump to the upper 2 K of ROM, a bank select instruction must precede the jump instruction. Although the bank select must precede the jump, the bank switch does not occur until the jump is executed. Once a bank is selected all jump will be within that bank until the bank select jump sequence is repeated.

The conditional jumps can test the following inputs or machine status. If the inputs or status conditions are true, the jump is permitted.

```
T0 = 1 or 0
T1 = 1 or 0
INT = 0
Accumulator = 0 or Not equal to zero.
Accumulator bit = 1
Carry = 1 or 0
F0 = 1
F1 = 1
Timer Flag = 1
```

The contional jumps allow branching to any address within the current page (256 words) under execution. The test conditions are the instantaneous values present when the conditional jump is executed.

The decrement register and skip is not zero instruction is useful for creating iteration counter. This instruction can be designate any one of the eigth registers in the currently selected bank to be a counter. The counter can effect a branch to any address within the current page.

The indirect jump instruction allows vectoring programs based upon the contents of the accumulator. The accumulator points to a location in the current page of ROM which contains the 8-bit jump address (also within the current page).

Input / Output Instructions

The input/output instructions provide for data transfers between the accumulator and the I/O port. The I/O ports have latched outputs, while the inputs must be read when input data is valid.

Additional instructions permit the ANDing or ORing of immediate data from ROM to either the Bus, Port1 or Port 2, with the result latched at the port. The capability to perform logical operations directly on the ports permits 'masks' stored in ROM to selectively set or reset individual bits of all three ports. Input on any given line via Port 1 and 2 is enabled by first writing a logic 1 to each line where input is desired.

The third I/O Port, the BUS Port, is a true bidirectional port that can be either latched or treated as a fully synchronous bi-directional bus. The BUS port can also have logical operations performed directly to its outputs. Unlike port 1 and 2, all eight bus lines must be treated as either inputs or outputs at any given time. If the BUS is being used as an I/O port, it will be in the input mode from the time the device is reset until an OUTL BUS,A instruction is executed. From then on, the BUS will not be resettable as an input unless the programmer uses it for either an external bus access or the device itself is reset. The BUS port can operate synchronously with external RAM using the MOVX instruction. The I/O instructions generate either a RD or WR pulse depending upon the operation. When either the RD or WR is generated, data must be valid on the trailing edge of the pulse. When the BUS lines are not in use, they are in the TRI-STATE (high impedance) mode.

The 48-Series I/O ports can be expanded from the basic three up to seven. Expansion is via the lower four bits of port 2. Each of the expansion ports 4 through 7 are 4 bits ports that have dedicated instruction types. The expansion port AND and OR instructions combine the accumulator contents with the selected port rather than with immediate data as on Port 0,1 or 2.

The Bus latching instruction, OUTL BUS is for use in single chip operations where BUS is not used as an expansion port. If necessary the OUTL BUS and MOVX instructions can be mixed. Care must be taken though since data previously latched will be destroyed by executing MOVX. The BUS lines will then be left in their TRI-STATE (high impedance) mode.

Note.- The OUTL instruction should never be used in systems with external ROM. If BUS is latched the next instruction, if fetched from external ROM may be fetched improperly.

System expansion can also be done via the BUS port. For expanded systems, additional I/O ports can be memory mapped, using the external RAM address space as addressed by pointer register R0 or R1.

Register Instructions
The register instructions allow the programmer to increment or decrement any of the enabled internal registers. Additionally the contents of a selected RAM location, as selected by the contents of R0 or R1, can be directly incremented.

Subroutine Instructions
The subroutine instructions are used to call or return from a designate subroutine.

Subroutine can be called from one bank to another as long as a bank select instruction precede a subroutine call. Once the subroutine in the other bank is completed execution will return to the bank originally selected.

Note: If the original bank is not reselected following a subroutine call to another bank, the next encontered jump instruction will transfer execution to the bank containing the subroutine.

There are two return from subroutine instructions. One restore status to the upper four bits of the program status word, while the other does not. The return and restore status instruction also signal the end of an interrupt service routine if one has been in progress.

Flag Instructions
The flag instructions allow the programmer to complement or clear three of the four user-accesable flags; Carry, F0 and F1. The functions of the four flags are as follows:

- Carry indicates an ovrflow occurred during a previous accumulator operation
- Auxiliary carry indicates an overflow between BCD digits when adding - Used by the decimal adjust instruction.
- F0 and F1 general purpose flags used for conditional jump test

The carry flags and F0 are accesible as a part of the program status word and are store on the stack during subroutine calls. Restoration is optional depend upon the return instruction used.

No Operation
This instruction does exactly what its name implies, it perform no operation other than take up memory space and execution time. Program execution continues with the next instruction. No operation is a useful tool during debug by saving 'space' to 'patch-in' additional instructions at some later time.

48-SERIES INSTRUCTION SET

The following pages contain detailed information on the 48-Series instruction set. The instructions have been arranged alphabetically, so they may be easily located. An illustration of the instruction set presentation is shown in figure 4-1. A special symbols and notations used with the instructions are shown in table 4-2

Figure 4-1 48-Series Sample Instruction

Mnemonic	MOV A,T
Operation	Move Timer/Counter to accumulator
Op Code	0 1 0 0 0 0 1 0
Symbolic Representation	(A) (T)
Description	The contents of the timer/counter are moved into the accumulator
Special Conditions	Cycles : 1 Bytes : 2

TABLE 4-2 Symbols Used in 48-Series Instructions

Symbol	Description
A	The accumulator
AC	The auxiliary carry flag
Addr	Program memory address (12 bits)
Bd	Bit designator (b=0-7)
BS	The bank switch
BUS	The bus port
C	Carry Flag
CLK	Clock Signal
CNT	Event Counter
D	Nibble designator (4bits)
data	Number or expresion (8 bits)
DBF	Memory bank flip flop
F0,F1	Flags 0,1
I	Interrupt
P	"In page" Operation Designator
Pp	Port designator (1,2 or 4-7)
PSW	Program Status Word
Rr	Register Designator (r=0,1 or 0,7)
SP	Stack Pointer
T	Timer
TF	Timer Flag
T0,T1	Testable Flags 0,1
X	External RAM
#	Prefix for immediate data
@	Prefix for indirect address
S	Program counter's current value
(X)	Contents of external RAM location
((X))	Contents of memory location addressed by the contents of external RAM location.
-	Replace by

ADD A,Rr
Add contents of designated register to accumulator

0 1 1 0 1 r r r

(A) - (A) + (Rr) where r = 0 through 7

The contents of the internal register designated by bits 'r' are added to the accumulator.

Cycles: 1
Bytes: 1
Flags: Carry, Auxiliary Carry

ADD A,@Rr
Add indirect contents of RAM location to the accumulator

0 1 1 0 0 0 0 r

The contents of the internal RAM location as addressed by bits 0 through 5* of register 'r' are added to the accumulator.

* bits 0 through 6 for 8039/8047/8049
* bits 0 through 7 for 8050

Cycles: 1
Bytes: 1
Flags: Carry, Auxiliary Carry

ADD A,#data
Add immediate to specified data to the accumulator

0 0 0 0 0 0 1 1 Byte 1
d7d6d5d4d3d2d1d0 Byte 2

(A) (A) + data

The data contained in byte 2 is added to the data into the accumulator.

Cycles: 2
Bytes: 2
Flags: Carry, Auxiliary Carry

ADDC A,Rr
Add with carry contents of designated register to accumulator.

0 1 1 1 1 r r r

(A) (A) + (C) + (Rr) where r = 0 through 7

The content of the carry bit is added to the accumulator, location 0 as the contents of the register specified by 'r' bits are added to the accumulator.

Cycles: 1
Bytes: 1
Flags: Carry, Auxiliary Carry

ADDC A,@Rr
Add indirect with carry contents of RAM location to accumulator.

0 1 1 1 0 0 0 r

(A) (A) + (C) + ((Rr)) where r = 0 or 1

The content of the carry bit is added to accumulator location 0 while the contents of the internal RAM location addressed by bits 0 through 5* of register 'r' are added to the accumulator.

* bits 0 through 6 for 8039/8048/8049
* bits 0 through 7 for 8050

Cycles: 1
Bytes: 1
Flags: Carry, Auxiliary Carry

ADDC A,#data
Add with carry specified immediate data to accumulator.

0 0 0 1 0 0 1 1 byte 1
d7d6d5d4d3d2d1d0 byte 2

(A) (A) + (C) + data

The content of the carry bit is added to the accumulator location 0 as the data contained in byte 2 is added is added to the data in the accumulator.

Cycles: 2
Bytes: 2
Flags: Carry, Auxiliary Carry

ANL A,Rr
Logical AND contents of designated register with A.

0 1 0 1 1 r r r

(A) (A) AND (Rr) where r = 0 through 7

The contents of the register specified by the 'r' bits are logically ANDed with the data in the accumulator.

Cycles: 1
Bytes: 1

ANL A,@Rr
Logically AND indirect contents of RAM with accumulator.

0 1 0 1 0 0 0 r

(A) (A) AND ((Rr)) where r = 0 or 1

The contents of the internal RAM location as addressed by bits 0 through 5* of register 'r' are logically ANDed with the data in the accumulator.

* bits 0 through 6 for 8039/8048/8049
* bits 0 through 7 for 8050

ANL A,#data
Logical AND immediate specified data with accumulator.

```
0 1 0 1 0 0 1 1          byte 1
d7d6d5d4d3d2d1d0         byte 2
```

(A) (A) AND data

The data contained in byte 2 are logically ANDed with the data in the accumulator and the result are put in the accumulator.

Cycles: 2
Bytes: 2

ANL BUS,#data
Logical AND immediate specified data with contents of BUS.

```
1 0 0 1 1 0 0 0          byte 1
d7d6d5d4d3d2d1d0         byte 2
```

(BUS) (BUS) AND data

The data contained in byte 2 are logically ANDed immediately with the data on the BUS port and the results are sent back to that port. Used of this instruction assumes prior execution of an OUTL BUS,A instruction.

Cycles: 2
Bytes: 2

ANL Pp,#data
Logical AND immediate specified data with designated port (1 or 2).

```
1 0 0 1 1 0 p1 p0        byte 1
d7d6d5d4d3d2d1d0         byte 2
```

(Pp) (Pp) AND data where p = 1 or 2

The data contained in byte 2 are logically ANDed immediately with the data on the port designated by bits 'p' and the result are sent back to that port. Accumulator contents are not affected. Op code bits 'p' designated the following ports:

Port	p1	p0
1	0	1
2	1	0

Cycles: 2
Bytes: 2

ANLD Pp,a
Logical AND contents of accumulator with designated expansion port (4 through 7).

1 0 0 1 1 1 p1 p0

(Pp) (Pp) AND (A0 - 3) where p = 4 through 7

The data in accumulator bits 0 through 3, are logically ANDed with the 4-bit data on the expander port designated by bits 'p' and the results are sent back to that port. Accumulator contents are not affected. Op code bits 'p' designate the following ports:

Port	p1	p0
4	0	0
5	0	1
6	1	0
7	1	1

Cycles: 2
Bytes: 1

CALL addr
Call designated subroutine.

a10 a9 a8 1 0 1 0 0 byte 1
a7 a6 a5 a4 a3 a2 a1 a0 byte 2

((SP)) (PC), (PSW 4-7)
(SP) (SP) + 1
(PC 8-10) addr 8-10
(PC 0-7) addr 0-7
(PC 11) DBF

The contents of both the program counter and program status word bits 4 through 7 are saved on the stack. The stack pointer is incremented by one. The contents of the program counter are replaced by address bits 'a' from bytes 1 and 2. Address bit 11 in the program counter is determined by the most recent bank select instruction (SEL MB) executed.

Note.- Although the stack pointer is only incremented by one, internally it is incremented by two so the PSW and PC can be saved on the stack.

Upon return from the subroutine program execution continues with the instruction immediately following CALL.

Cycles: 2
Bytes: 2

CLR A
Clear contents of accumulator to zero.

0 0 1 0 0 1 1 1

(A) 0

The contents of accumulator are cleared to zero.

Cycles: 1
Bytes: 1

CLR C
Clear content of carry bit to zero.

1 0 0 1 0 1 1 1

(C) 0

The content of the carry bit is cleared to zero. During normla program execution the carry bit may be set to one due to an ADD, ADDC, RLCA, CPLC, RRCA or DAA instructions.

Cycles: 1
Bytes: 1
Flags: Carry

CLR F0
Clear content of flag 0 to zero

1 0 0 0 0 1 0 1

(F0) 0

The content of flag 0 is cleared to zero

Cycles: 1
Bytes: 1
Flags: F0

CLR F1
Clear content of Flag 1 to zero

1 0 1 0 0 1 0 1

(F1) 0

The content of flag 1 is cleared to zero.

Cycles: 1
Bytes: 1
Flags: F1

CPL A
Complement contents of the accumulator

0 0 1 1 0 1 1 1

(A) NOT (A)

The contents of the accumulator are complemented. This is a one complement, with each 1 changing to a 0 and each 0 changing to a 1.

Cycles: 1
Bytes: 1

CPL C
Complement content of carry bit.

1 0 1 0 0 1 1 1

(C) NOT (C)

The content of carry bit is complemented. A content of 1 is changed to 0 and a content of 0 is changed to 1.

Cycles: 1
Bytes: 1
Flags: Carry

CPL F0
Complement content of flag 0

1 0 0 1 0 1 0 1

(F0) NOT (F0)

The content of flag 0 is complemented. A content of 1 is changed to 0, a content of 0 is changed to 1.

Cycles: 1
Bytes: 1
Flags: F0

CPL F1
Complement content of flag 1

1 0 1 1 0 1 0 1

(F1) NOT (F1)

The content of flag 1 is complemented. A content of 1 is changed to 0, a content of 0 is changed to 1.
Cycles: 1
Bytes: 1
Flags: F1

DA A
Decimal adjust contents of accumulator

0 1 0 1 0 1 1 1

The 8-bit contents of the accumulator are adjusted to form two 4-bit BCD digits. Carry is affected if accumulator bits 0 through 3 are greater than nine, or if the auxiliary carry bit is a one, the accumulator is incremented by six. Accumulator bits 4 through 7 are then checked. If they exceed nine, the upper four bits are incremented by six. If an overflow occurs, carry is set to one.

NOTE.- AC is set any time the four LSB's of A>9

Example: Assume accumulator contains 10111010

```
AC  C  7 6 5 4 3 2 1 0
 1  0  1 0 1 1 1 0 1 0     b0-b3 > 9
                0 1 1 0     Add 6
----------------------
 0  0  1 1 0 0 0 0 0 0     b4-b7 > 9
       0 1 1 0             Add 6
----------------------
 0  1  0 0 1 0 0 0 0 1
```

Cycles: 1
Bytes: 1
Flags: Carry

DEC A
Decrement contents of accumulator by one.

0 0 0 0 0 1 1 1

(A) (A) - 1

The contents of the accumulator are decremented by one.

Cycles: 1
Bytes: 1

DEC Rr
Decrement contents of register Rr by one.

1 1 0 0 1 r r r

(Rr) (Rr) - 1 where r = 0 through 7

The contents of the register designated by bits 'r' are decremented by one.
Cycles: 1
Bytes: 1

DIS I
Disable external interrupt input.

0 0 0 1 0 1 0 1

The external interrupt input is disabled. Low going signals at the interrupt input have no effect.

Cycles: 1
Bytes: 1

DIS TCNTI
Disable internal timer/counter interrupt flag.

0 0 1 1 0 1 0 1

The internal timer/counter interrupt flag output is disabled. A pending timer/counter interrupt request is cleared. If the timer is operating and overflows, the timer flag will be set, but no interrupt will occur, and the timer will continue to count.

Cycles: 1
Bytes: 1

DJNZ Rr,addr
Decrement specified register, test contents and jump if not zero.

```
1 1 1 0 1  r  r  r        byte 1
a7a6a5a4a3 a2 a1 a0       byte 2

(Rr)   (Rr) - 1    where  r = 0 through 7
(PC 0-7)   addr   if (Rr) distinct of 0
```

The contents of the register designated by bits 'r' are decremented by one and then tested to see if the contents equal to zero. If the register contents equal to zero, the next sequential instruction is executed. If the register contents do not equal zero, control passes to the instruction at the address designated in byte 2.

The address is eight bits in lenght, limiting jumps to within the current 256-location page. If byte 1 of DJNZ is at location 255 of page 1 and byte 2 is at location 0 of page 2, the jump will be to the specified address within page 2.

Cycles: 2
Bytes: 2

EN I
Enable external interrupt.

0 0 0 0 0 1 0 1

The external interrupt input is enabled. A low-going signal at the interrupt input initiates a vector to location 3 in ROM. If the interrupt input is already low this instruction will execute a call to location 3.

Cycles: 1
Bytes: 1

EN TCNTI
Enable internal timer/counter interrupt.

0 0 1 0 0 1 0 1

The internal timer/counter interrupt is enabled. A timer/counter overflow will set the timer flag and an interrupt vector to location 7 in ROM will occur. If the timer flag is already set indicating a timer/counter overflow, enabling the timer/counter flag output will not cause an interrupt.

Cycles: 1
Bytes: 1

ENT0 CLK
Enable clock output at T0

0 1 1 1 0 1 0 1

Test input T0 is enabled to output the internal system clock. T0 is disabled as a test input. T0 is disabled as a clock output only by a system reset.

Cycles: 1
Bytes: 1

IN A,Pp
Input data to accumulator from designated port (1 or 2).

0 0 0 0 1 0 p1 p0

(A) (Pp) where p = 1 or 2

Data preset at the port designated by bits 'p' is input into the accumulator. Opcode bits 'p' designate the following ports:

Port	p1	p0
1	0	1
2	1	0

INC A
Increment content of accumulator by one.

0 0 0 1 0 1 1 1

(A) (A) + 1

The contents of the accumulator are incremented by one.

Cycles: 1
Bytes: 1

INC Rr
Increment contents of designated register by one.

0 0 0 1 1 r r r

(Rr) (Rr) + 1 where r = 0 through 7

The contents of register designated by bits 'r' are incremented by one.

Cycles: 1
Bytes: 1

INC @Rr
Increment indirect contents of RAM by one.

0 0 0 1 0 0 0 r

((Rr)) ((Rr)) + 1 where r = 0 or 1

The contents of the internal RAM location as addressed by bits 0 through 5* of register 'r' are incremented by one.

* bits 0 through 6 for 8039/8048/8049
* bits 0 through 7 for 8050

Cycles: 1
Bytes: 1

INS A,BUS
Input strobed data to accumulator from BUS.

0 0 0 0 1 0 0 0

(A) (BUS)

Data present at the BUS port is input to the accumulator during the RD strobe.

Cycles: 2
Bytes: 1

JBb addr
Jump to specified address if accumulator bit 'b' is set.

```
b2 b1 b0  1  0  0  1  0        byte 1
a7 a6 a5 a4 a3 a2 a1 a0        byte 2

(PC 0-7)   addr, if Bb = 1
(PC)    (PC) + 2 , if Bb = 0
```

If the accumulator bit designated by bits 'b' is set to a logic one, the contents of the program counter are replaced by address bits 'a' from byte 2. If bit 'b' in the accumulator is a logic zero, the next sequential instruction is executed. Bits b2,b1 and b0 represent a number from 0 to 7 designating which bit in the accumulator is to be tested.

```
Cycles:  2
Bytes:   2
```

JC addr
Jump to specified address if carry flag is set.

```
1  1  1  1  0  1  1  0      byte 1
a7 a6 a5 a4 a3 a2 a1 a0     byte 2

(PC 0-7)    addr   if C = 1
(PC)    (PC) + 2   if C = 0
```

If the carry bit is set to a logic one, the contents of the program counter are replaced by address bits 'a' from byte 2. If carry is a logic zero, the next sequential instruction is executed.

```
Cycles:   2
Bytes:    2
```

JF0 addr
Jump to specified address if flag 0 is set.

```
1  0  1  1  0  1  1  0       byte 1
a7 a6 a5 a4 a3 a2 a1 a0      byte 2

(PC 0-7)    addr,   if F0 = 1
(PC)   (PC) + 2     if F0 = 0
```

If flag 0 is set to a logic one, the contents of the program counter are replaced by address bits 'a' from byte 2. If flag 0 is a logic zero, the next sequential instruction is executed.

```
Cycles:   2
Bytes:    2
```

JF1 addr
Jump to specified address if flag 1 is set.

```
0  1  1  1  0  1  1  0      byte 1
a7 a6 a5 a4 a3 a2 a1 a0     byte 2

(PC 0-7)     addr,  if F1 = 1
(PC)   (PC) + 2     if F1 = 0
```

If flag 1 is set to a logic one, the contents of the program counter are replaced by address bits 'a' from byte 2. If flag 1 is a logic zero, the next sequential instruction is executed.

Cycles: 2
Bytes: 2

JMP addr
Jump direct to specified address within 2K address block

```
a10 a9 a8  0  0  1  0  0        byte 1
a7  a6 a5 a4 a3 a2 a1 a0        byte 2

(PC 8-10)   addr 8-10
(PC 0-7)    addr 0-7
(PC11)   DBF
```

The contents of the program counter are replaced by address bits 'a' in bytes 1 and 2. Address bit 11 in the program counter is determined by the most recent bank select instruction (SEL MB) executed.

Cycles: 2
Bytes: 2

JMPP @A
Jump indirect to specified address within address page.

```
1  0  1  1  0  0  1  1

(PC 0-7)   ((A))
```

The contents of the program counter are replaced by the ROM contents within current page pointed to by the accumulator. For example if the accumulator contains X'20 a jump to the address stored at location 32 (dec) (in the current page) occurs.

Cycles: 2
Bytes: 1

JNC addr
Jump to specified address is carry flag is a logic zero.

```
1   1   1   0   0   1   1   0        byte 1
a7  a6  a5  a4  a3  a2  a1  a0       byte 2

(PC 0-7)    addr    if C = 0
(PC)   (PC) + 2     if C = 1
```

If the carry flag is low, the contents of the program counter are replaced by address bits 'a' from byte 2. If carry is a logic one, the next sequential instruction is executed.

Cycles: 2
Bytes: 2

JNI addr
Jump to specified address if interrupt is a logic zero.

```
1   0   0   0   0   1   1   0        byte 1
a7  a6  a5  a4  a3  a2  a1  a0       byte 2

(PC 0-7)    addr   if  I = 0
(PC)   (PC) + 2    if  I = 1
```

If the interrupt input is at a logic zero, the contents of the program counter are replace by address bit 'a' from byte 2. If the interrupt input is a logic one, the next sequential instruction is executed. This instruction provides a means for testing the condition of the external interrupt pin while it is disabled as an interrupt.

Cycles: 2
Bytes: 2

JNT0 addr
Jump to specified address if test 0 is a logic zero.

```
0   0   1   0   0   1   1   0        byte 1
a7  a6  a5  a4  a3  a2  a1  a0       byte 2

(PC 0-7)    addr     if T0 = 0
(PC)   (PC) + 2      if T0 = 1
```

The input T0 is at a logic zero, the contents of program counter are replaced by address bits 'a' from byte 2. If T0 is a logic one, the next sequential instruction is executed. This instruction should not be executed after an ENT0 CLK instruction.

Cycles: 2
Bytes: 2

JNT1 addr
Jump to specified address if test 1 is a logic zero.

```
0  1  0  0  0  1  1  0        byte 1
a7 a6 a5 a4 a3 a2 a1 a0       byte 2

(PC 0-7)     addr    if T1 = 0
(PC)    (PC) + 2     if T1 = 1
```

If input T1 is a logic zero, the contents of the program counter are replaced by address bits 'a' from byte 2. If T1 is a logic one, the next sequential instruction is executed.

Cycles: 2
Bytes: 2

JNZ addr
Jump to specified address if accumulator is non-zero.

```
1  0  0  1  0  1  1  0        byte 1
a7 a6 a5 a4 a3 a2 a1 a0       byte 2

(PC 0-7)   addr    if  A not equal to zero
(PC)   (PC) + 2    if A equal to zero
```

If the contents of the accumulator are non-zero, contents of the program counter are replaced by address bits 'a' from byte 2. If the accumulator contents are zero, the next sequential instruction is executed.

Cycles: 2
Bytes: 2

JTF addr
Jump to specified address if timer flag is set.

```
0  0  0  1  0  1  1  0        byte 1
a7 a6 a5 a4 a3 a2 a1 a0       byte 2

(PC 0-7)    addr    if TF = 1
(PC)   (PC) + 2     if TF = 0
```

If the internal timer/counter flag is set to a logic one, the contents of the program counter are replaced by address bits 'a' from byte 2. If the timer/counter flag is a logic zero, the next sequential instruction is executed.

Testing the timer/counter flag resets the flag to zero. This instruction provides a means of testing the timer/counter flag if the timer/counter interrupt is disabled. An overflow of the timer/counter will cause an interrupt vector to location 7 in ROM unless the

timer/counter interrupt has be disabled.

Cycles: 2
Bytes: 2

JT0 addr
Jump to specified address if test 0 is a logic one.

```
0  0  1  1  0  1  1  0        byte 1
a7 a6 a5 a4 a3 a2 a1 a0       byte 2

(PC 0-7)   addr  if T0 = 1
(PC)   (PC) + 2  if T0 = 0
```

If input T0 is at a logic one, the contents of the program counter are replaced by address bits 'a' from byte 2. If T0 is a logic zero, the next sequential instruction is executed.

Cycles: 2
Bytes: 2

JT1 addr
Jump to specified address if test 1 is a logic one.

```
0  1  0  1  0  1  1  0        byte 1
a7 a6 a5 a4 a3 a2 a1 a0       byte 2

(PC 0-7)   addr   if T1 = 1
(PC)   (PC) + 2   if T1 = 0
```

If input T1 is at logic one, the contents of the program counter are replaced by address bits 'a' from byte 2. If T1 is a logic zero, the next sequential instruction is executed.

Cycles: 2
Bytes: 2

JZ addr
Jump to specified address if accumulator is zero.

```
1  1  0  0  0  1  1  0        byte 1
a7 a6 a5 a4 a3 a2 a1 a0       byte 2

(PC 0-7)    addr   if A = 0
(PC)   (PC) + 2    if A not = 0
```

If the contents of the accumulator are zero, the contents of the program counter are replaced by address bits 'a' from byte 2. If the accumulator contents are non-zero the next sequential instruction is executed.
Cycles: 2
Bytes: 2

MOV A,#data
Move immediate specified data into accumulator

```
0  0  1  0  0  0  1  1      byte 1
d7 d6 d5 d4 d3 d2 d1 d0     byte 2
```

(A) data

The data contained in byte 2 is moved into accumulator.

Cycles: 2
Bytes: 2

MOV A,PSW
Move contents of program status word into accumulator.

```
1  1  0  0  0  1  1  1
```

(A) (PSW)

The contents of the program status word (PSW) are moved into the accumulator.

Cycles: 1
Bytes: 1

PSW

Carry	Aux carry	Flag 0	Reg Bank Sel.	1	S2	S1	S0
MSB							LSB

MOV A,Rr
Move contents of designated register into accumulator.

```
1  1  1  1  1  r  r  r
```

(A) (Rr) where r = 0 through 7

The contents of the working register designated by bits 'r' are moved into the accumulator.

Cycles: 1
Bytes: 1

MOV A,@Rr
Move indirect contents of RAM into accumulator.

```
1  1  1  1  0  0  0  r
```

The contents of the internal RAM location as addressed by bits 0 through 6 of register 'r' are moved into the accumulator.

MOV A,T
Move contents of timer/counter into accumulator.

0 1 0 0 0 0 1 0

(A) (T)

The contents of the timer/counter are moved into the accumulator.

Cycles: 1
Bytes: 1

MOV PSW,A
Move contents of accumulator into program status word.

1 1 0 1 0 1 1 1

(PSW) (A)

The contents of the accumulator are moved into the program status word (PSW). All condition bits and status codes are affected by this instruction.

Cycles: 1
Bytes: 1
Flags: Carry, Aux Carry, F0, Register Bank select.

C AC F0 RBS 1 S2 S1 S0

MOV Rr,A
Move accumulator contents into designated register.

1 0 1 0 1 r r r

(Rr) (A) where r = 0 through 7

The contents of the accumulator are moved into the working register designated by bits 'r'.

Cycles: 1
Bytes: 1

MOV Rr,#data
Move immediate specified data into designated register.

1 0 1 1 1 r r r byte 1
d7 d6 d5 d4 d3 d2 d1 d0 byte 2

(Rr) data where r = 0 through 7

The data contained in byte 2 is moved into the working register designated by bits 'r' on byte 1.

MOV @Rr,A
Move indirect accumulator contents into RAM.

```
1  0  1  0  0  0  0  r

((Rr))     (A)    where r = 0 or 1
```

The contents of the accumulator are moved to the RAM location as addressed by bits 0 through 6 of register 'r'. Register 'r' contents are unaffected.

Cycles: 1
Bytes: 1

MOV @Rr,#data
Move immediate specified data into RAM.

```
1  0  1  1  0  0  0  r        byte 1
d7 d6 d5 d4 d3 d2 d1 d0       byte 2

((Rr))     data   where r = 0 or 1
```

The data contained in byte 2 is moved to the RAM location as addressed by bits 0 through 6 of register 'r'.

Cycles: 2
Bytes: 2

MOV T,A
Move contents of accumulator into timer/counter.

```
0  1  1  0  0  0  1  0

(T)     (A)
```

The contents of the accumulator are moved into the timer/counter register.

Cycles: 1
Bytes: 1

MOVD A,Pp
Move contents of designated expansion port (4 through 7) into the accumulator.

```
0  0  0  0  1  1  p1  p0

(A0-3)     (Pp)    where p = 4 through 7
(A4-7)     0
```

The four bit data on the expander port, designated by bits 'p', are moved into the accumulator, bits 0 through 3. Bits 4 through 7 in the accumulator are set to zero.

Op code bits 'p' designate the following ports:

Port	p1	p0
4	0	0
5	0	1
6	1	0
7	1	1

Cycles: 2
Bytes: 1

MOVD Pp,A
Move contents of accumulator to designated expansion port (4 through 7).

0 0 1 1 1 1 p1 p0

(Pp) (A0-3) where p = 4 through 7

The data in the accumulator, bits 0 through 3 are moved to the expander port designated by bits 'p'. Accumulator contents are unaffected.

Op code bits 'p' designate the following ports:

Port	p1	p0
4	0	0
5	0	1
6	1	0
7	1	1

Cycles: 2
Bytes: 1

MOVP A,@A
Move data in current page (ROM) into accumulator.

1 0 1 0 0 0 1 1

(PC 0-7) (A)
(A) ((PC))
(PC 0-7) (old PC 0-7) + 1

The contents of the program counter, bits 0 through 7 are replaced by the contents of the accumulator. The contents of the internal ROM location, as addressed by the new contents of the program counter, are moved into the accumulator. The program counter is then sent back to point to the next sequential instruction. Only bits 0 through 7 of the program counter are affected, limiting memory references to the current page. Being a 1-byte, 2-cycle instruction, if MOVP A is at location 255 of a page, the @A addresses a location in the following page.

The program counter is restored after the instruction is executed.

Cycles: 2
Bytes: 1

MOVP3 A,@A
Move data in page 3 (ROM) into accumulator.

1 1 1 0 0 0 1 1

(PC 0-7) (A)
(PC 8-10) 011
(A) ((PC))
(PC) (old PC) + 1

The contents of the program counter, bits 0 through 7 are replaced by the contents of the accumulator. Program counter bits 8 throguh 10 are replaced by 011 respectively. The contents of the internal page 3 ROM, as addressed by the new contents of the program counter, are moved into the accumulator. The program is restored after this instruction is executed.

Cycles: 2
Bytes: 1

MOVX A,@Rr
Move indirect contents of external RAM into accumulator.

1 0 0 0 0 0 0 r

(A) ((Rr)) where r = 0 or 1

The contents of the external RAM location as addressed by register 'r' are moved into the accumulator. Register 'r' contents are unaffected.

Cycles: 2
Bytes: 1

MOVX @Rr,A
Move indirect contents of accumulator into external RAM.

1 0 0 1 0 0 0 r

((Rr)) (A) where r = 0 or 1

The contents of the accumulator are moved into the external RAM location, as addressed by register 'r'. Accumulator and register 'r' contents are unaffected.
Cycles: 2
Bytes: 1

NOP
No operation performed.

0 0 0 0 0 0 0 0

No operation is performed, execution continues with the next sequential instruction.

Cycles: 1
Bytes: 1

ORL A,Rr
Logical OR contents of designated register with accumulator.

0 1 0 0 1 r r r

(A) (A) OR (Rr) where r = 0 through 7

The contents of the register specified by the 'r' bits are logically ORed with the data in the accumulator.

Cycles: 1
Bytes: 1

ORL A,@Rr
Logicall OR indirect contents of RAM with accumulator.

0 1 0 0 0 0 0 r

(A) (A) OR ((Rr)) where r = 0 or 1

The contents of the internal RAM location as addressed by bits 0 through 6 of register 'r' are logically ORed with the data in the accumulator.

Cycles: 1
Bytes: 1

ORL A,#data
Logicall OR specified immediate data with accumulator.

0 1 0 0 0 0 1 1 byte 1
d7 d6 d5 d4 d3 d2 d1 d0 byte 2

(A) (A) OR data

The data contained in byte 2 is logically ORed with the data in the accumulator.

Cycles: 2
Bytes: 2

ORL BUS,#data
Logicall OR immediate specified data with contents of Bus

```
1  0  0  0  1  0  0  0        byte 1
d7 d6 d5 d4 d3 d2 d1 d0       byte 2
```

(BUS) (BUS) OR data

The data contained in byte 2 is logically ORed immediately with the data on the BUS port and the results are sent back to that port. Use of this instruction assumes prior execution of an OUTL BUS,A instruction.

Cycles: 2
Bytes: 2

ORL Pp,#dta
Logical OR immediate specified data with contents of designated Port (1 or 2)

```
1  0  0  0  1  0  p1 p0        byte 1
d7 d6 d5 d4 d3 d2 d1 d0       byte 2
```

(Pp) (Pp) OR data where p = 1 or 2

The data contained in byte 2 is logically ORed with the data on the port designated by bits 'p'. Opcode bits 'p' designate the following ports:

Port	p1	p0
1	0	1
2	1	0

Cycles: 2
Bytes: 2

ORLD Pp,A
Logical OR contents of accumulator with designated expansion port (4-7)

```
1  0  0  0  1  1  p1 p0
```

(Pp) (Pp) OR (A0-3) where p = 4 through 7

The data in accumulator bits 0 through 3 are logically ORed with the data on to the expander port designated by bits 'p'. Opcode bits 'p' designate the following ports:

Port	p1	p0
4	0	0
5	0	1
6	1	0
7	1	1

Cycles: 2
Bytes: 1

OUTL BUS,A
Output contents of accumulator onto Bus.

0 0 0 0 0 0 1 0

(BUS) (A)

The contents of accumulator are placed and latched at the Bus output port. Latched data remains valid until another OUTL BUS instruction is executed, or until another instruction requiring the Bus port (except INS) is executed.

Logiccal operations using Bus data assume prior execution of the OUTL BUS,A instruction.

Data are destroyed any time the Bus port is used as a bus. The Bus port is in input mode after a reset. After execution of OUTL BUS,A the Bus port will remain an output until the device is either reset or this port is used for bus transfers.

Cycles: 2
Bytes: 1

OUTL Pp,A
Output contents of accumulator to designated port (1or2)

0 0 1 1 1 0 p1 p0

(Pp) (A) where p = 1 or 2

The contents of the accumulator are placed and latched at the output port designated by bits 'p'. Opcode bits 'p' designated the following ports:

Port	p1	p0
1	0	1
2	1	0

Cycles: 2
Bytes: 1

RET
Return from subroutine or interrupt without restoring program status word.

1 0 0 0 0 0 1 1

(SP) (SP) - 1
(PC) ((SP))

The contents of the stack pointer are decremented by one. The contents of the stack as pointed to by the new contents of the stack pointer, are then placed in the program counter and the program counter is considered restored.

Note.- Although the stack pointer is only decremented by one, internally it is decremented by two, so the PSW and PC can be pulled on the stack.

Program status word bits 4 through 7 are not restored by this instruction.

Cycles: 2
Bytes: 1

RETR
Return from subroutine or interrupt restoring program status word.

1 0 0 1 0 0 1 1

(SP) (SP) - 1
(PC) ((SP))
(PSW 4-7) ((SP))

The contents of the stack pointer are decremented by one. The contents of the stack as pointed to by the new contents of the stack pointer, are then placed on the program counter and the program counter is considered restored.

Program status word bits 4 through 7 are restored from stack to the program status register.

Note.- Although the stack pointer is only decremented by one, internally it is decremented by two so the PSW and PC can be pulled of the stack.

The RETR instruction should be used to return from interrupt, but should not be used to return from a subroutine within an interrupt. This is because RETR indicates the end of an interrupt routine by re-enabling INT.

Cycles: 2
Bytes: 1
Flags: Carry, Aux Carry, F0, Interrupt Enable Flag (only if register is servicing an interrupt).

RL A
Rotate accumulator left one bit.

1 1 1 0 0 1 1 1

(An + 1) (An) where n = 0 through 6
(A0) (A7)

The contents of the accumulator are rotated left by one bit position. Bit 7 goes directly to bit 0.

Cycles: 1
Bytes: 1

RLC A
Rotate accumulator left one bit through carry.

1 1 1 1 0 1 1 1

(A n+1) (An) where n = 0 through 6
(A0) (C)
(C) (A7)

The contents of the accumulator are rotated left by one bit position. Bit 7 moves to carry and carry moves to bit 0.

Cycles: 1
Bytes: 1
Flags: Carry

RR A
Rotate accumulator right one bit

0 1 1 1 0 1 1 1

(An) (A n+1) where n = 0 through 6
(A7) (A0)

The contents of the accumulator are rotated by one bit position.

Cycles: 1
Bytes: 1

RRC A
Rotate accumulator right one bit through carry.

0 1 1 0 0 1 1 1

(An) (A n+1) where n = 0 through 6
(A7) (C)
(C) (A0)

The contents of the accumulator are rotated right by one bit position. Bit 0 moves to carry and carry moves to bit 7.

Cycles: 1
Bytes: 1
Flags: Carry

SEl MB0
Select bank 0 from ROM (Locations 0 through 2047)

1 1 1 0 0 1 0 1

(DBF) 0

The memory bank flip flop (DBF) is set to zero. For succeeding JMP or CALL instructions program counter bit 11 is set to a logic zero, causing all ROM addresses to fall within locations 0 and 2047.
Cycles: 1
Bytes: 1

SEL MB1
Select bank 1 of ROM (locations 2048 through 4095)

1 1 1 1 0 1 0 1

(DBF) 1

The memory bank flip flop (DBF) is set to one. For succeeding JMP or CALL instructions, program counter bit 11 is set to a logic one causing all ROM addresses to fall within locations 2048 and 4095.

Cycles: 1
Bytes: 1

SEL RB0
Select register bank 0 of RAM (locations 0 through 7)

1 1 0 0 0 1 0 1

(BS) 0

The bank select bit program status word-bit 4, is set to a logic zero. All references to registers 0 through 7 address RAM locations 0 through 7 , respectively. This is the recommended setting for normal program execution.

Cycles: 1
Bytes: 1

SEL RB1
Select register bank 1 of RAM (locations 24 through 31)

1 1 0 1 0 1 0 1

(BS) 1

The bank select bit, program status word bit-4 is set to a logic one. All refenrences to registers 0 through 7 address RAM locations 24 through 31, respectively. This is the recommended setting for interrupt service routines. The bank select bit is saved during interrupts and is restored by RETR when the interrupt service routine is completed.

Cycles: 1
Bytes: 1

STOP TCNT
Stop count for timer/counter

0 1 1 0 0 1 0 1

This instructions stops the internal timer/counter regarless of the mode of operation.

NOTES:
1. If the timer/counter is disabled and one or more falling edges occur at the T1 input, the STRT CNT instruction will cause the counter to increment immediately.

2. There is a mask programmable resistor option to prevent the preceeding from occuring.

Cycles: 1
Bytes: 1

STRT CNT
Start count for event counter.

0 1 0 0 0 1 0 1

Test input T1 is enabled as the input to the timer/counter and the counter is started. The counter is incremented by one for each high to low transition at imput T1.

Cycles: 1
Bytes: 1

STRT T
Start timer

0 1 0 1 0 1 0 1

The internal clock is enabled to the timer/counter and the timer is started. The counter is incremented by one for each 32 instruction cycles. The divide by 32 prescaler is cleared by this instruction.

Cycles: 1
Bytes: 1

SWAP A
Swap 4-bit nibble positions in accumulator.

0 1 0 0 0 1 1 1

(A 4-7) (A 0-3)

Accumulator bits 0 through 3 are swapped with accumulator bits 4 through 7

Cycles: 1
Bytes: 1

XCH A,Rr
Exchanges contents of accumulator and designated register.

0 0 1 0 1 r r r

(A) (Rr) where r = 0 through 7

The contents of the accumulator and the register designated by bits 'r' are exchanged.

Cycles: 1
Bytes: 1

XCH A,@Rr
Exchange indirect contents of accumulator and RAM location.

0 0 1 0 0 0 0 r

(A) ((Rr)) where r = 0 or 1

The contents of the internal RAM location, as addressed by bits 0 through 6 of register 'r' are exchanged with the contents of the accumulator. Register 'r' contents are unaffected.

Cycles: 1
Bytes: 1

XCHD A,@Rr
Exchange indirect lower four bits of accumulator and RAM location.

0 0 1 1 0 0 0 r

(A 0-3) (((Rr)) 0-3) where r = 0 or 1

The lower four bits of internal RAM location as addressed by bits 0 through 6 of register 'r', are exchanged with the lower four bits of the accumulator.

The uper four bits of both RAM and the accumulator are unaffected.

Cycles: 1
Bytes: 1

XRL A,Rr
Logical XOR contents of designated register with accumulator.

1 1 0 1 1 r r r

(A) (A) XOR (Rr) where r = 0 through 7

The contents of the register specified by bits 'r' are logically EXCLUSIVE-ORed with the data in the accumulator. The register contents are unaffected.

Cycles: 1
Bytes: 1

XRL A,@Rr
Logical XOR indirect contents of RAM with accumulator

1 1 0 1 0 0 0 r

(A) (A) XOR ((Rr))

The contents of the internal RAM location as addressed by bits 0 through 6 of register 'r' are logically EXCLUSIVE-ORed with the data in the accumulator. The internal RAM location contents are unaffected.

Cycles: 1
Bytes: 1

XRL A,#data
Logical XOR immediate specified data with accumulator.

1 1 0 1 0 0 1 1 byte 1
d7 d6 d5 d4 d3 d2 d1 d0 byte 2

(A) (A) XOR data

The data contained in byte 2 are logically EXCLUSIVE-ORed with the data in the accumulator.

Cycles: 2
Bytes: 2

LM323K 5 VOLT POSITIVE REGULATOR - 3 AMP -

General Description

The LM323K is a three terminal positive regulator with a preset 5V output and a load driving capability of 3 amp New circuit design and processing techniques are used to provide the high output current without sacrificing the regulation caracteristics of lower current devices.

The 3 amp regulador is virtually blowout proof. Current limiting, power limiting and thermal shutdown provide the same high level of reliability obtained with this techniques in the LM109 1 amp regulator.

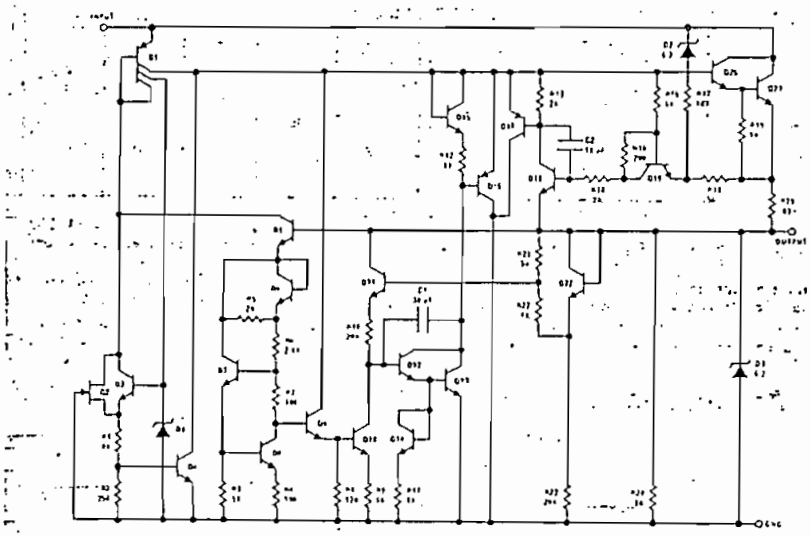
No external components are required for operation of the LM323K. If the device is mora than 4 inches from the filter capacitor, however a 1uF solid tantalum capacitor should be used on the input. A 0.1 uF or larger capacitor may be used on the output to reduce load transient spikes created by fast switching digital logic or to swamp out stray load capacitance.

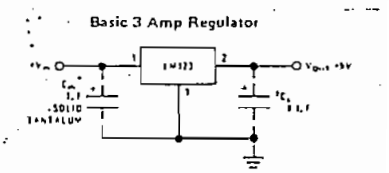
An overall worst case specification for the combined effects of input voltage, load currents, ambient temperature and power dissipation ensure that the LM323K will perform satisfactorily as a system element.

Features

- 3 amp output current
- Internal current and thermal limiting
- 0.01 ohm typical output impedance
- 7.5 minimum input voltage
- 30 W power dissipation

SCHEMATIC DIAGRAM AND CONNECTION





LM7912 3-TERMINAL NEGATIVE REGULATOR

GENERAL DESCRIPTION

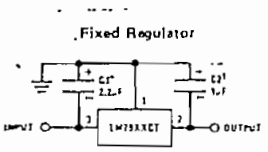
The LM7912 is a -12V fixed regulador output voltage. These device need only one external component -a compensation capacitor at the output. The LM7912 is packaged in the TO-220 package an is capable of supplying 1.5A of output current.

These regulators employ internal current limiting safe area protection and thermal shutdown for protectin against virtually all overload conditions.

Features

- Thermal, short circuit and safe are protection
- High ripple rejection
- 1.5A output current
- 4% preset output voltage.

Typical Application and Connection Diagram



LM339 QUAD COMPARATOR - NTE 834 equivalent

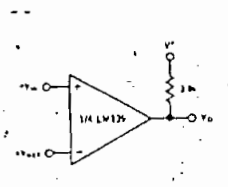
The LM339 series consists of four independient precision voltage comparators with an offset voltage specification as low as 2mV max for all four comparators. These were designed specifically to operate from a single power supply over a wide range of voltages. Operation from split power supplies is also possible and the low power supply current drain is independient of the magnitude of the power supply voltage. This comparators also have a unique caracteristic in that the input common mode voltage range incluides ground, even though operated from a single power supply voltage.

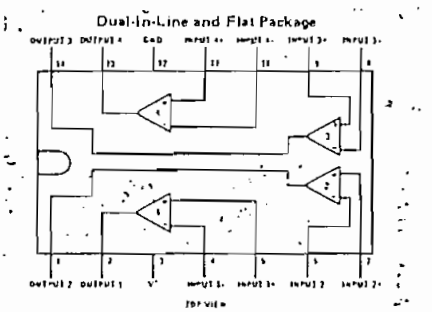
Applications areas incluide limit comparators, simple analog to digital converters; pulse, squarewave and time delay generators; wide range VCO; MOS clock timers; multivibrators and High Voltage digital logic gates. The LM339 series was designed to directly interface with TTL and CMOS. When operated from both plus and minus power supplies, they will directly interface with MOS logic, where the low power drain of the LM339 is a distinct advantage over standard comparators.

Advantages
- High precision comparators
- Reduced Vos drift over temperature
- Eliminates need for dual supplies
- Allows sensing near ground
- Compatible with all forms of logic
- Power drain suitable for battery operation

Features
- Wide supply voltage range or dual supplies
  2 VDC to 36 VDC or +- 1VDC to +- 18 VDC
- Very low supply current drain
- Low input biasing current 25 nA
- Low input offset current +/- 5 nA
- Low input offset voltage +/- 3 mV
- Input common mode voltage range includes ground
- Diferential input voltage range equal to the power supply voltage
- Low output saturation voltage 250 mV at 4 mA
- Output voltage compatible with TTL, DTL, ECL, MOS and CMOS logic systems.

Schematic and Connection Diagrams


Basic Comparator



CD4093B QUAD 2-INPUT NAND Schmitt Trigger

The CD4093B consists of four Schmitt trigger circuits. Each circuits functions as a 2-input NAND gate with schmitt trigger action on both inputs. The gate switches at different points for positive and negative going signals. The difference between the positive V_{T+} and the negative V_{T-} is defined as hysteresis voltage V_H.

All outputs have equal source and sink currents and conform to standard B-series output drive.

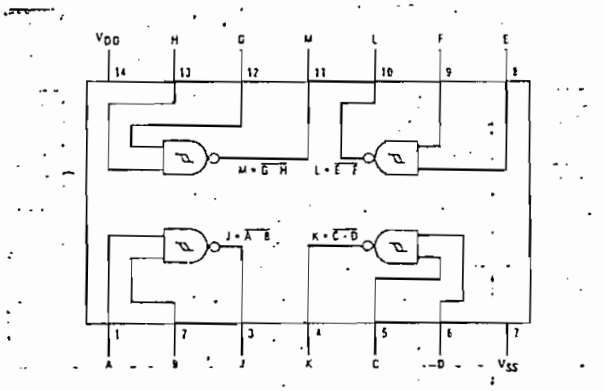
Features

- Wide supply voltage range
- Schmitt trigger on each input
- Noise immunity greater than 50%
- Equal source and sink current
- No limit on input rise and fall rise
- Standard B-series output drive
- Hysteresis voltage at 5V supply - V_H=1.5V

Applications

- Wave and pulse shapers
- High noise enviroment systems
- Monostable multivibrators
- Astable multivibrators
- NAND logic

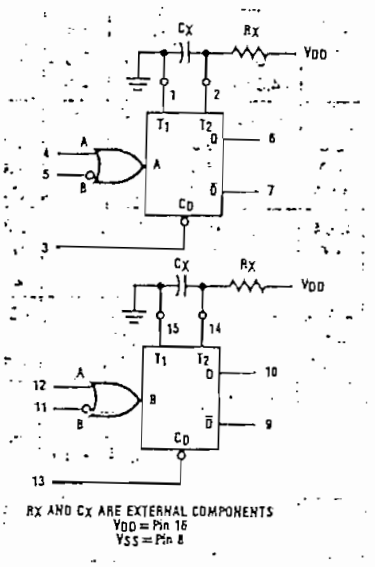
Connection Diagram

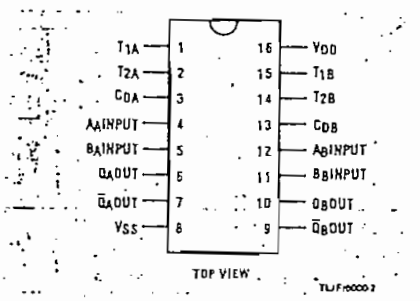


CD4538B DUAL PRECISION MONOSTABLE equiv NTE 4098B

The CD4538B is a dual precision monostable multivibrator with independent trigger and reset controls. The device is retriggerable and resettable and the control inputs are internally latched. Two trigger inputs are provided to allow either rising or falling edge triggering. The reset inputs are active low and prevent triggering while active. Precise control of output pulse-width has been achieved using linear CMOS techniques. The pulse duration and accuracy are determined by externar components Cx and Rx. The device does not allow the timing capacitor to discharge through the timing pin on power down condition. For this reason, no external protection resistor is required in series with the timing pin. Input protection from static discharge is provided on all pins.

Block and Connection Diagrams





Truth Table

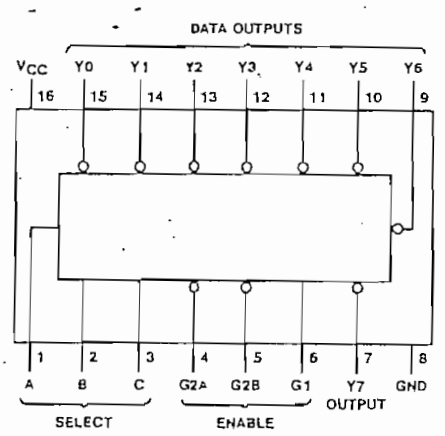
Inputs			Outputs		H=High level
CLR	A	B	Q	Q	L=Low level
L	X	X	L	H	=Transition lo-hi
X	H	X	L	H	=Transition hi-lo
X	X	L	L	H	
H	L				
H		H			

DM74LS138 DECODER DEMULTIPLEXER

These Schottky-clamped circuits are designed to be used in high performance memory decoding or data routing applications, requiring very short propagation delay times. In High performance memory systems these decoders can be used to minimize de effects of system decoding. When used with high speed memories, the delay times of these decoders are usually less than the typical access time of the memory. This means that the effective system delay introduced by the decoder is negligible.

The 74LS138 decodes one-of-eight lines, based upon the conditions at the three binary select inputs and the three enable inputs. Two active-low and one active-high enable inputs reduce the need for external gates or inverters when expanding. A 24-line decoder can be implemented with no external inverters, and a 32-line decoder requires only one inverter. An enable input can be used as a data input for demultiplexing applications.

Connection Diagram



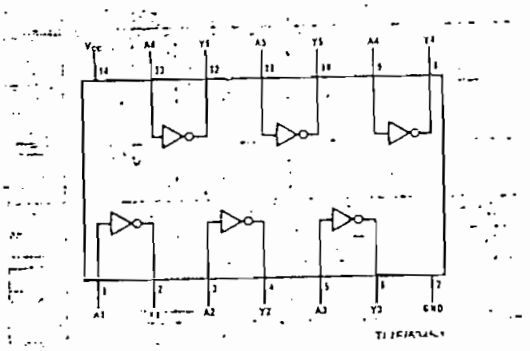
Function Table

Inputs					Outputs							
Enable		Select										
G1	G2	C	B	A	Y0	Y1	Y2	Y3	Y4	Y5	Y6	Y7
X	H	X	X	X	H	H	H	H	H	H	H	H
L	X	X	X	X	H	H	H	H	H	H	H	H
H	L	L	L	L	L	H	H	H	H	H	H	H
H	L	L	L	H	H	L	H	H	H	H	H	H
H	L	L	H	L	H	H	L	H	H	H	H	H
H	L	L	H	H	H	H	H	L	H	H	H	H
H	L	H	L	L	H	H	H	H	L	H	H	H
H	L	H	L	H	H	H	H	H	H	L	H	H
H	L	H	H	L	H	H	H	H	H	H	L	H
H	L	H	H	H	H	H	H	H	H	H	H	L

DM74LS04 HEX INVERTING GATES

This device contains six independent gate each of which performs the logic INVERT function.

Connection Diagram



Function Table

Input A	Output Y	$Y = \overline{A}$
L	H	
H	L	

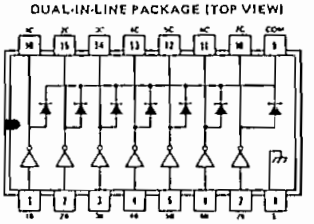
ULN 2003 BUFFER DRIVER equiv NTE 2013

High-voltage high-current Darlington transistor arrays

- 500 mA Rated Collector Current
- High-Voltage Outputs...100 V
- Output Clamp Diodes
- Inputs Compatible with Various Types of Logic
- Relay Driver Applications
- Higher-Voltage Versions of ULN2001A, ULN2002A, ULN2003A, and ULN2004A

INPUT COMPATIBILITY:
SN75466—DTL, TTL, P-MOS, CMOS
SN75467—14 to 25 volt P-MOS
SN75468—TTL or 5 volt CMOS
SN75469—6 to 15 volt CMOS or P-MOS

J OR N
DUAL-IN-LINE PACKAGE (TOP VIEW)



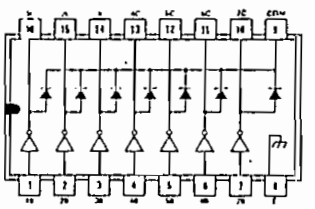
ULN2001A, ULN2002A, ULN2003A, ULN2004A DARLINGTON TRANSISTOR ARRAYS

High-voltage high-current Darlington transistor arrays

- 500 mA Rated Collector Current
- High-Voltage Outputs...50 V
- Output Clamp Diodes
- Inputs Compatible with Various Types of Logic
- Relay Driver Applications
- Designed to be Interchangeable with Sprague ULN2001A Series

INPUT COMPATIBILITY:
ULN2001A—DTL, TTC, P-MOS, CMOS
ULN2002A—14 to 25 volt P-MOS
ULN2003A—TTL or 5 volt CMOS
ULN2004A—6 to 15 volt CMOS or P-MOS

J OR N
DUAL-IN-LINE PACKAGE (TOP VIEW)



DN7447 BCD to 7-Segment DECODER/DRIVER

The DM7447 feature active low outputs designed for driving common anode LEDs or incandescent indicators directly. All of the circuits have full ripple blanking input/output control and a lamp test input. Segment identification and resultant displays are shown in the following figure.

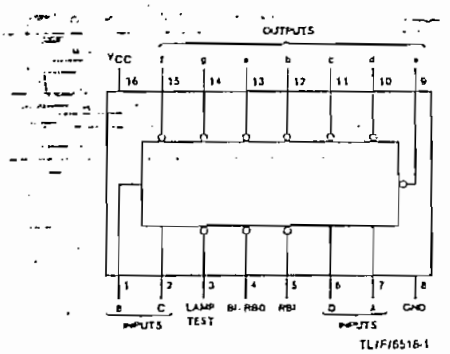
The circuit incorporates automatic leading and/or trailing edge, zero blanking control (RB1 and RB0). Lamp test (LT) of these device may be performed at any time when the BI/RBO node is at high logic level. All types contain an overriding blanking input (BI) which can be used to control the lamp intensity (by pulsing) or to inhibit the outputs.

Features

- Lamp intensity modulation capability
- Open collector output drive indicators directly
- Lamp test provision
- Leading trailing zero suppresion

Active level ----- LOW
Output configuration ---- Open Collector
Sink Current ----- 40 mA
Maximun Voltage ---- 15 V
Typical power dissipation ---- 320 mW

Connection Diagram



5446A (J) 7446A (N)
5447A (J) 7447A (N)

FUNCTION TABLE DM 7447

Decimal	Inputs						BI/RBO	Outputs						
	LT	RBI	D	C	B	A		a	b	c	d	e	f	g
0	H	H	L	L	L	L	H	L	L	L	L	L	L	H
1	H	X	L	L	L	H	H	H	L	L	H	H	H	H
2	H	X	L	L	H	L	H	L	L	H	L	L	H	L
3	H	X	L	L	H	H	H	L	L	L	L	H	H	L
4	H	X	L	H	L	L	H	H	L	L	H	H	L	L
5	H	X	L	H	L	H	H	L	H	L	L	H	L	L
6	H	X	L	H	H	L	H	H	H	L	L	L	L	L
7	H	X	L	H	H	H	H	L	L	L	H	H	H	H
8	H	X	H	L	L	L	H	L	L	L	L	L	L	L
9	H	X	H	L	L	H	H	L	L	L	H	H	L	L
10	H	X	H	L	H	L	H	L	L	L	H	H	H	L
11	H	X	H	L	H	H	H	H	H	L	L	H	H	L
12	H	X	H	H	L	L	H	H	L	H	H	H	L	L
13	H	X	H	H	L	H	H	L	H	H	L	H	L	L
14	H	X	H	H	H	L	H	H	H	H	L	L	L	L
15	H	X	H	H	H	H	H	H	H	H	H	H	H	H
BI	X	X	X	X	X	X	L	H	H	H	H	H	H	H
RBI	H	L	L	L	L	L	L	H	H	H	H	H	H	H
LT	L	X	X	X	X	X	H	L	L	L	L	L	L	L

This program starts by typing 'PROMPROG'. Contains an on line editor for your code once read into the buffer. You can toggle between editing ascii and hexadecimal by toggeling the tab-button. Also this program contains an easy to use database by manufactor and type of device. New devices can be add to this database.

You can now freely choose which program you like the best. Succes !.

This program is provided with the following options:

F1 TYPE

Manufacturer
PROM type
Programming voltage
Vcc when programming
Time for one byte
Repetition factor
Add type to database
Delete type in database

F2 BUFFER

Load file into buffer
Save buffer to file
Read EPROM into buffer
Move buffer data
Clear buffer
Fill buffer
Print buffer
Begin addres
End address

F3 EDIT

View buffer
Goto address
Find byte/word
Search string
Begin address
End address

F4 PROGRAM

Test EPROM empty
Program EPROM
Skip FF
Begin address
End address

F5 VERIFY

Verify EPROM
Begin address
End address

F6 QUIT

Exit to MS/DOS
MS/DOS command

Manufactor DATABASE

AMD	2716	2732	2732A	2764	2764A
	27128	27128A	27256	27512	
EUROTECHNIQUE		ET2716	ETC2716	ET2732	ET2764
FUJITSU	2716	2732	2732A	2764	27128
	27256	27C32A	27C64	27C128	27C256
	27C512				
HITACHI	27128A	27256	27512	27C64G	27C256
	462716	462732	482732AG	462732P	482764
	4827128	4827128P			
INTEL	2716	2732	2732A	2764	2764A
	27128	27128A	27256	27512	27C64
	27C256				
TEXAS INSTRUMMENTS			2732	2732A	2764
			27128	27C128	27C256
TOSHIBA	2732A	2732D	2764	27128	27256
	27512				
NATIONAL SEMICONDUCTOR			2716	2732	27C16
	27C16H	27C32	27C32B	27C32H	27C64
	27C128	27C256	27C512	27CP128B	
NEC	2732	2732A	2764C	27128C	27256D
	26C64C	27256A	27C64	27C256	27C512
OKI	2716	2732	2732A	2764	27128
	27256	27C256	27256A		
WAFERSCALE INTEGRATION			WS27C64	WS27C128	
XICOR	X2864A	X2864AL			

MANUAL DE UTILIZACION DEL EDITOR "EDLIN"

INTRODUCCION

Si usted desea editar un archivo nuevo, unicamente basta teclear el siguiente mensaje: EDLIN NOMBRE.TXT

Y con ello se empieza a editar el file denominado NOMBRE.TXT siendo las tres últimas letras las correspondientes al tipo de file que se desee crear.

Si este archivo no existía en disco, el computador responde con el mensaje "New file" y ya estamos en capacidad de iniciar la creación de este nuevo archivo. Para ello debemos indicar con la letra 'I' para insertarnos en el nuevo archivo.

Una vez que tengamos el texto, podemos terminar la edición con el comando 'E', que indica fin de edición (END). En este momento el archivo es enviado al disco, luego de lo cual estamos en los comandos propios del sistema operativo MS/DOS del IBM PC o compatible.

Además para salir del modo de edición al modo de comandos debemos presionar simultaneamente las teclas CTRL C.

COMANDOS

El programa EDLIN permite una gran variedad de opciones de edición por medio de la utilización de sus comandos como veremos a continuación:

‹línea›
Podemos poner el número de la línea y con ello ingresar únicamente a editar solo dicha línea y luego regresa al modo de comandos.

Tambien podemos situar allí un punto ‹.› y nos va a situar en la última línea modificada.

Finalmente tambien podemos llamar al comando ‹#› y este nos situará al final del texto.

COPY

1,4,20C Indica que copie el bloque que va de la línea 1 a la 4 y la ponga desde la línea 20.

1,4,20,7C Similar al anterior pero el bloque repetido 7 veces.

DELETE

3D Borra la línea 3

5,24D Borra desde la línea 5 hasta la 24

END

E Finaliza la edición y la envía al disco, para luego salir hacia los comandos del DOS.

INSERT

5I Inserte texto inmediatamente antes de la línea 5 anterior.

LIST

2L Provoca un listado del documento desde la línea 2 hasta 23 líneas adelante.

MOVE

20,30,100M Mueve un rango de líneas, desde la 20 hasta la 30 y las sitúa desde la línea 100.

PAGE

P Provoca que se muestre en pantallas suscesivas (de 23 líneas en 23 líneas) todo el documento, conforme se vaya presionando P.

CROSS-16 META-ASSEMBLER

INTRODUCTION

Cross-16 from Universal Cross-Assemblers is a meta-assembler, in that it assembles programs for numerous different target processors. It reads an assembly language source file and corresponding sorted assembler instruction table from disk, and writes an assembled listing and a hexadecimal machina language output file. By using a flexible instruction table structure, Cross-16 is designed to assemble machine language code for most microprocessors and microcontrollers with an address word of 24 or less bits. To further enhance it's flexibility, Cross-16 will produce a hexadecimal output file in the Intel and Motorola 8 and 16 bit formats.

As one proceeds throught this manual, one will pass from the broad generalities needed to use Cross-16, to the detailed specifics necessary to write a processor instruction table.

Universal Truths Concerning Cross-16

The following are broad generalities for those already familiar with compilers and assemblers. Others need not fret, all of the following will be explained in greater detail in the appropiate sections of this manual.

Cross-16 is a two pass cross-assembler with an optional third pass if a phase error is detected.

All input and output files contain ASCII characters with each line terminated by an ASCII carriage-return and line-feed.

The most significant bit of all ASCII characters is reset by Cross-16. (i.e. ASCII_CHAR = ASCII_CHAR AND 7F)

Only one processor assembly instruction or Cross-16 assembler directive is permitted per line.

All label declarations must be terminated with a colon ":", incluiding labels for the EQU and SET directives.

All labels, expressions and operands are internally stored and manipulated using 32 bit signed integers.

Cross-16 makes no distinction between upper and lower case alphabetic characters.

Blank lines in the assembly source code are reproduced in the assembly listing but otherwise ignored.

Overflow errors, undefined labels, and invalid expressions are assigned a value of zero.

There are no restrictions on the character length of labels or processor instructions and all characters are significant. Cross-16 input lines must not exceed 128 characters in length.

Cross-16 does not support macros nor does it include a linker or librarian.

Running Cross-16

The enclosed disk contains the Cross-16 Meta-Assembler (C16.EXE), the Cross-16 Sort Program (C16SORT.EXE) and sixteen processor tables (cpu.TBL) with sixteen corresponding example assembly source file (Tcpu.ASM). Universal Cross-Assemblers recommends that the user back up the entire Cross-16 disk. The copy C16.EXE and the table and example file for the processor that interests you on to a disk or directory that contains your favorite program editor. The following examples use the T8048.ASM and 8048.TBL files.

Cross-16 may then be run using the following command:

```
C16 sourcefile [-L listfile] [-H hexfile]
```

Where square brackets [ ] indicate optional items.

The "-L" tells Cross-16 to produce an assembly list file using the immediately following name. The "-H" tells Cross-16 to produce a hexadecimal output file using the immediately following file name. If these are omitted the corresponding files will not be produced. All error codes will be displayed on the screen, regarless of wheter a list file is created. The format of the hexfile is set using the HOF directive. The processor instruction table used by Cross-16 is also specified in the source file using the CPU directive. The order in which the source, listing and hex files are specified in the command line does not matter. Disk and directory names may be included in the file names.

Some examples are:

```
C16 T8048.ASM              source:  T8048.ASM
                           listing: NONE
                           hex:     NONE

C16 T8048.ASM -L 8048.LST
                           source:  T8048.ASM
                           listing: T8048.LST
                           hex:     NONE
```

```
C16 T8048.ASM -L 8048.LST -H 8048.HEX
                          source   T8048.ASM
                          listing: T8048.LST
                          hex:     T8048.HEX
```

Running Cross-16 produces the following on the screen:

```
A>C16 T8048.ASM

Cross-16 Meta-Assembler    PC/MS-DOS  Version 1.00
Copyright (C) 1986 Universal Cross Assemblers

Starting Pass Number 1

Starting Pass Number 2

End of Assembly --- No Errors

A>
```

Where "A>" is the operating system prompt. Since Cross-16 is a two pass meta-assembler, the source file is actually read twice. The processor instruction table is only read once and kept in RAM for the second pass. Any assembly errors will be displayed on the screen during the second pass, in addition to being placed in the listing if one was specified. The listing and hexadecimal files are written during second pass only.

The Cross-16 Assembly Source File

The source file is the ASCII assembler source code to be assembled by Cross-16. From the source file, the processor instruction table is selected using the CPU directive, and the format of the hexadecimal file is set using the HOF directive.

The following sub-sections describe the fundamental building blocks which make up the assembly source file. Examples are given for each section, but these have been assembled and taken from listing file; therefore the user does not provide the first 16 characters of these examples.

Assembly Line Format

Only one assembly instruction or assembler directive is permitted per line. The assembly line is free format, labels need not start in column 1 for example. Each line may contain some or all of the following sequence of identifiers:

```
Line#   label:   operation    operand(s)  ; comment
```

Line# .- Is an optional decimal integer in the range of 0 to 65535 created by some editors representing the source code line number. If present, Cross-16 will ignore this field except to reproduce it in the listing.

laber.- is a phrase starting with an alphabetic character or an underline "_" and ending with a colon ":" which is assigned the present value of the program counter or other user specified integer value. Characters within the label must be alphanumeric or an underline "_"

Operation.- is a Cross-16 assembler directive, or processor asembly instruction defined in the instruction table. All operations must start with two alphabetic characters.

Operand(s).- are labels, constants, or expressions representing integer values in the range of -2147483648 to 2147483647 and/or character strings. They may be embodied within an assembly instruction.

Comment.- is a statement following a semi-colon ";" usually used to describe the assembly program. The ";" may be placed anywhere on the assembly line and Cross-16 ignores all characters following it on that line.

Labels

A label is an alphanumeric series of characters representing an integer in the range -2147483648 to 2147483647. The label field must start with an alphabetic ASCII character and must end with a colon ":", even when used with the EQU and SET assembler directives. After the first character, all remaining characters may be alphanumeric or an underline "_". Except for the EQU and SET directives, a label is optional and assigned the current value of the program counter. Labels may be of any length except that a Cross-16 input line cannot exceed 128 characters, and all characters are significant. Cross-16 makes no distinction between upper and lower case characters. A label may also stand alone on a line, in which case it will be assigned the current value of the program counter. A dollar sign "$" may be used as an operand representing the current value of the program counter.

The following examples are taken from an actual Cross-16 listing. They use the equate (EQU) directive described further in next sections. Very simply, the equate EQU directive assigns the label on its left, the value of the expression on its right.

```
0200                    ORG 0200H       ; Origin

                ; Valid examples of labels are:
                ;
1234 =          STRT1:      EQU  1234H      ; Label on Directive
1234 =          LD_UP:      EQU  1234H      ; Label with "_"
0200            alone:                      ; Stand alone label

0987 =          LONG_LONG_LONG:  EQU  987H  ; Any length


                ; Invalid examples of labels are:
                ;
S000            1LABEL:     EQU 1234H       ; Starts with number
S000            MISS        EQU 1234H       ; Missing colon ":"
P1234 =         DUP:        EQU 1234H       ; Duplicate Label
P5678 =         DUP:        EQU 5678H       ; Duplicate Label
```

Operands

The following sub-sections describe numeric constans, string constants, and arithmetic and logical operators which in combination with labels may be used to form operands.

Numeric Constants

A numeric constant is an ASCII representation of a 32 bit signed integer in one of several number bases. All numeric constant must begin with a number, and their base is indicated by a trailing alphabetic character. The default base is base 10. The letter "Q" may also be used to specified octal numbers to avoid confusion between "O" and "0". The valid base with their corresponding trailing alphabetic characters and character ranges are as follows:

```
B     Binary      Base 2     0-1
Q     Octal       Base 8     0-7
D     Decimal     Base 10    0-9
H     Hexadecimal Base 16 0-9 , A-F
```

```
                ; Valid examples of numeric constants are
                ;
00AF =          BIN1:       EQU 10101111B   ; Binary
00FF =          OCT1:       EQU 377Q        ; Octal
003C =          OCT2:       EQU 74Q         ; Octal
FFFF =          DEC1:       EQU -1          ; Decimal
000A =          DEC2:       EQU 10D         ; Decimal
00FF =          HEX1:       EQU 0FFH        ; Hexadecimal

                ; Invalid examples of numeric constants are
                ;
V0000 =         HEX2:       EQU 0FFFFFFFFFFH  ; Too large
```

```
E007B =    DEC3:      EQU 123.4    ; Illegal decimal
U0000 =    HEX3:      EQU FFH      ; No leading number
E0025 =    OCT3:      EQU 37 7Q    ; Illegal blank
```

String Constants

String constants consist of a series of ASCII characters between two quotation marks ("). Cross-16 will convert these constants to a hexadecimal representation of their ASCII values. Lower case characters within string constants will be represented as such. A quotation mark (") cannot apear within a character string, for it will be interpreted as being the end of that string. If a quotation mark is needed Universal Cross Assembler recommends that an apostrophe be used ('). The user may also terminate the string, insert the binary value of a quotation mark (22H), and then start another string. A string constant may also be used as an operand where applicable. In a DFB statement, a string constant may be of any length, bearing in mind that an assembly source line must not exceed 128 characters in length. When used as an operand, or in the DWM, DWL, DFL, SET or EQU directives, an error will be flagged if the string constant exceeds the lenght of the operand specified by the instruction. A string constant cannot be extended beyond it's present line without terminating the string with a quotation mark. Although only the first five or seven bytes of the string constant are shown in the listing, it is placed the hex file in its entirety.

The following examples use the define byte (DFB) directive as described in next sections. Very simply, the define byte (DFB) directive places the byte by byte value of the expressions on its right into memory, starting at the present memory location shown on the left.

```
0000                        ORG 0000H

                ; Valid examples of string constants are
                ;
0000 5965612C20             DFB "Yea, yea, yea!"
000E 41426162               DFB "AB","ab"
0012 5468652064             DFB "The dog Said 'Woof Woof'"

              ; Invalid examples of string constants are:
              ;
T002B                       DFB 'Hello'  ; Using apostrophe
A002B 4D69737369            DFB "Missing  ; Missing quote
```

Expressions

Cross-16 will accept numeric expressions made up of

labels, constants and script brackets "()" combined with logical and arithmetic operators. A list of recognized logical and arithmetic operators in there relative precedence follows where x and y represent integer values:

( )	script parentheses
- y	Unary minus of y identical to 0-y or the two's complement of y
x * y	Integer multiplication of x and y
x / y	Integer division of x by y
x MOD y	Integer remainder after division of x/y
x SHL y	Logical shift left of x by y bits and filled from the right with zeros
x SHR y	Logical shift right of x by y bits and filled from the left with zeros
x + y	Integer addition of x and y
x - y	Integer subtraction of y from x
NOT y	Ones complement of y
x AND y	Logical AND of x and y
x OR y	Logical Or of x and y
x XOR y	Exclusive Or of x and y
HIGH y	Binary value of high order byte only
LOW y	Binary value of low order byte only
$	Present value of program counter

```
                ; Valid examples of operators are
                ;
00FF =          EXPRES1:  EQU  2 OR 253  ;Logical OR
0003 =          EXPRES2:  EQU  "3" AND 15 ; AND
FFFA =          EXPRES3:  EQU  NOT 0101B ; NOT
0011 =          EXPRES4:  EQU  10B XOR CR ; XOR
3400 =          EXPRES5:  EQU  0034H SHL 8 ; Shift left
0012 =          EXPRES6:  EQU  1234H SHR 7 ; Shift right
0078 =          EXPRES7:  EQU  LOW 5678H  ; Low byte
0056 =          EXPRES8:  EQU  HIGH 5678H ; High byte
FFFF =          EXPRES9:  EQU  -(3-2)  ; Two's complement
4259 =          EXPRES10: EQU  "AZ"+0FFH  ; Addition
0014 =          EXPRES11: EQU  40-20D  ; Substraction
```

```
4100 =          EXPRES12: EQU  "A" * 256 ; Multiplication
0042 =          EXPRES13: EQU  "BA"/256  ; Division
0001 =          EXPRES14: EQU  10 MOD 3  ; Remainder

                ; Invalid examples of expressions are
                ;
E0000 =         EXPRES15: EQU  2**10D ;Illegal operator
E0000 =         EXPRES16: EQU  23*(-DEC2) ; Wrong parenth
```

Cross-16 Assembler Directives

Cross-16 uses several common assembler directives combined with several of its own as listed below.

```
CPU           CPU Table declaration
DFB           Define Byte
DFS           Define data Storage
DFL           Define long integer
DWM           Define word (High byte first)
DWL           Define word (Low byte first)
END           End of program
EQU           Equate label to specified value
HOF           Hexadecimal output format
IF,ELSE,ENDI  Conditional assembly
INCL          Include source file
ORG           Origin
PAGE          Page control
SET           Set label to dynamic value
TITL          Tittle on Listing
```

CPU --- CPU Table Declaration

The CPU directives tells Cross-16 which processor instruction table is to be loaded during assembly.

The CPU directive has the following syntax:

```
label:    CPU  "cpu_file_name"    ; comment
```

Only the instruction table file name may be specified after with the CPU directive. A disk drive and/or directory may be include in the file specification. A label may be placed before de the CPU directive, wchich will be assigned the current value of the program counter. The instruction table is only read once by Cross-16 during assembly, and all subsequent CPU directives will be ignored. A nonexisting CPU file name will result in a fatal error.

```
          ; Examples of the CPU directive
          ;
0043         CPU "Z8.TBL"    ; CPU Table
0043         CPU "B:8048.TBL" ; CPU table
```

DFB --- Define Byte

The DFB directive allows the user to define the value of storage areas on a byte by byte basis.

The DFB directive has the following syntax:

```
label:     DFB  expr1,expr2,...,expr(n)  ; comment
```

Except for a string constant, the result of each expression must represent an 8 bit value or an error will be flagged. An expression may consist of a numeric constant, a string constant, a label or a formula. There is no limit on the number of bytes that may be defined using a single DFB directive, except that the length of an Cross-16 input line must not exceed 128 characters. Although only the first 5 or 7 bytes of this data will be shown in the listing, it is included in the hex file in its entirety.

```
2000                    ORG 2000H

                    ; Valid examples of the DFB directive are
                    ;
2000  5061746965 NOTE:  DFB  "Patient Lead Off",CR,LF
                                    ; ASCII string
2012  000102            DFB  0,1,2  ;Integers
2015  1548              DFB  LOW($),77Q + 9 ;Expression
2017  3F44              DFB  3FH,44H  ; Hexadecimal

                  ; Invalid examples of the DFB directive
                  ;
V2019 00                DFB 03FFH  ; Too large (10 bits)
V201A 00                DFB -129D  ; Too small
```

DFL --- Define Long Integer

The DFL directive allows the user to define the value of storage areas on a long intege or long word basis (a long integer is 32 bits or four bytes). There is no limit on the number of long integers that may be defined usign a single DFL directive, except that the length of a Cross-16 input line must not exceed 128 characters. Although only the first 5 or 7 bytes of this data will be shown in the listing, it is included in the hex file in its entirety.

The directive syntax is as follows:

```
label:     DFL   expr1,expr2,...,expr(n)  ; comment
```

The value of each expression must be representable using a 32 bit signed integer or an assembly error will be flagged. An expression may consist of a numeric

constant, a string constant, a label or a formula. Although ASCII string constants may be used, an error will be flagged if they exceed 4 characters. The DFL directive stores 32 bit signed integers from the most significant byte to the least significant byte.

```
3000                    ORG 3000H

                ; Valid examples of the DFL directive are
                ;
3000 00000002  CONST1:  DFL 2     ; Numeric constant
3004 00003000           DFL CONST1 ; Label
3008 00000048           DFL 77Q + 9 ; Expression
300C 12345678           DFL 12345678H ; Hexadecimal

                ; An invalid example of the DFL directive is
                ;
V3012 00000000          DFL "ABCDE"    ; Too long
```

DFS --- Define Storage

The define storage directive DFS may be used to reserve a section of memory with unspecified contents during assembly.

The DFS directive has the following syntax:

```
label:   DFS  expression          ; comment
```

The expression can be of any form which represents a 32 bit positive integer value, but only one expression is allowed. The value of the expression is added to the program counter and assembly continues. Except for the changed value of the program counter, no values are written to the hex file.

```
5000                        ORG 5000H

                    ; Valid examples of the DFS directive are
                    ;
5000                STOR_B:   DFS 20 * 1  ; Reserve 20 bytes
5014                STOR_W:   DFS 20 * 2  ; Reserve 20 words
503C                STOR_L    DFS 20 * 4  ; Reserve 20 long

                    ; Invalid examples of the DFS directive
                    ;
V508C               STOR_E    DFS "ABCDE" ; Too long
V508C               STOR_F    DFS - 20    ; Negative value
```

DWL --- Define Word (Least significant word first)

The DWL directive allows the user to define the value of storage areas on a word by word basis (one word is two bytes). There is no limit on the number of words that

may be defined using a single DWL directive, except that the length of a Cross-16 input line must not exceed 128 characters. Although only the first 5 or 7 bytes of this data will be shown in the listing, it is included in the hex file in its entirety.

The DWL directive has the following syntax:

```
Label:   DWL   expr1,expr2,...,expr(n)   ; comment
```

The result of each expression must represent a 16 bit integer value or an error will be flagged. An expression may consist of a numeric constant, a string constant, a label or a formula. Although ASCII string constant may be used, an error will be flagged if they exceed 2 characters in length. The DWL directive will store the least significant byte of the 16 bit value before the most significant byte. By using either the DWM or DWL directives, words may be placed in memory in a format which corresponds to the format used by the target processor.

```
4000                    ORG 4000H

                ; Valid examples of DWL directive are:
                ;
4000  0000010002 CONST2:  DWL 0,1,2    ; Numeric Constant
4006  06004800            DWL LOW($),77Q+9 ;Expressions
400A  3F004400            DWL 3FH,44H ; Hexadecimal
400E  FFFF                DWL 0FFFFH  ; up to 16 bits

                ; Invalid examples of DWL directive are
                ;
V4010 0000                DWL "ABC"   ; Too long
V4012 0000                DWL  $ * 7777Q ; Too large
```

DWM --- Define Word (Most significant byte first).

The DWM directive allows the user to define the value of storage areas on a word by word basis (one word is two bytes). There is no limit on the number of words that may be defined using a single DWL directive, except that the length of a Cross-16 input line must not exceed 128 characters. Although only the first 5 or 7 bytes of data will be shown in the listing, it is included in the hex file in its entirety.

The DWM directive has the following syntax:

```
label:   DWM   expr1,expr2,...,expr(n)   ; comment
```

The result of each expression must represent a 16 bit integer value or an error will be flagged. An expression may consist of a numeric constant, a string

constant, a label or a formula. Although ASCII string constant may be used, an error will be flagged if they exceed 2 characters in length. The DWM directive will store the most significant byte of the 16 bit value before the least significant byte. By using either the DWM or DWL directives, words may be placed in memory in a format that corresponds to the format used by the target processor.

```
3000                    ORG 3000H

                ; Valid examples of the DWM directive are
                ;
3000 0000000100 CONST1:  DWM 0,1,2    ; Numeric constant
3006 3000                DWM CONST1   ; Label
3008 00080048            DWM LOW($),77Q + 9 ; expression
300C 003F0044            DWM 3FH,44H  ; Hexadecimal
3010 FFFF                DWM 0FFFFH ; up to 16 bit

                ; Invalid examples of the DWM directive are
                ;
V3012 0000               DWM "ABC"    ; Too long
V3014 0000               DWM $ * 999D ; Too large
```

END --- End of Program

The End of assembly (END) directive is optional, but when used it will be the last line of the assembly source file assembled (the remainder be ignored)

This directive has the following syntax:

```
Label:    END    expression      ; comment
```

The expression is optional, but may represent any positive 16 or 24 bit integer value, depending on the hexadecimal output format in use. When an expression is given, its value will be included as the address in the final line of the hexadecimal machina language file, representing the starting address of the assembly program. If the END directive or expression are not included, this starting address will default to zero.

```
                ; Valid examples of the END directive
                ;
0000                    END       ; simple format
0000      THE_END       END  RESET ; with starting address
0100                    END 0100H  ; Famous starting address

                ;An invalid example of the END directive is:
                ;
T0001                   END 1,2    ; Multiple operands
```

EQU --- Equate

The Equate (EQU) directive may be used to assign an integer value to a label.

The EQU directive has the following syntax:

```
label:   EQU   expression      ; comment
```

Neither the label or the expression are optional in this directive. The expression may consist of any numeric constant, character string or formula whose value can be represented in 32 bits. Cross-16 will place the value of expression in the label field followed by an equal sign (=) to show that this value is not the current value of the program counter. Defining a label more that once, or placing multiple expressions after the EQU directive, will result in an assembly error.

```
                ; Valid examples of the EQU directive are
                ;
0013 =          CR:     EQU  13H    ; ASCII carriage return
000A =          LF:     EQU  10D    ; ASCII line feed
5012 =          CNTR:   EQU  $      ; program counter
1300 =          EXPR2:  EQU  CR SHL 8 ; Formula

                ; Invalid examples of the EQU directive
                ;
T0021 =         TWICE:  EQU 33,-6   ; Multiple expressions
E0000 =         WHAT:   EQU         ; Missing expression
Y0D05 =                 EQU 3333    ; Missing label
```

HOF --- Hexadecimal Output Format

The hexadecimal output format (HOF) selects the format of the hexadecimal machine language output file.

The HOF directive has the following syntax:

```
Label:   HOF    "format"       ; comment
```

A label may be included with the HOF directive, which will be assigned the current value of the program counter. The HOF directive is optional, and if not incluided Cross-16 will default to the "INT16" format. The HOF directive may appear anywhere in the assembly soruce file. If it appears more than once with different hexadecimal formats specified, the format of the hexadecimal file will change without an error code being generated.

The HOF directive partially controls the format of the assembled listing. If a HOF directive is not used, or

one of the 16 bit hexadecimal formats is declared, Cross-16 will produce a listing with a 32 bit value preceding the EQU and SET directives, 24 bit addresses, and up to 7 bytes of code on each line. If one of the 8 bit hexadecimal formats is declared, Cross-16 will produce a listing with a 16 bit value preceding the EQU and SET directives, 16 bit addresses and up to 5 bytes of code on each line. This allows the listing format to correspond to the address word of the target processor.

```
6500                    ORG 6500H
          ; Valid examples of the HOF directive are:
          ;
006500                  HOF "INT16" ; Intel 16 bit
6500                    HOF "INT8"  ; Intel 8 bit
006500                  HOF "MOT16" ; Motorola 16 bit
6500                    HOF "MOT8"  ; Motorola 8 bit

          ; Invalid examples of the HOF directive are:
          ;
H6500                   HOF "BIN8" ;Illegal hex format
U6500                   HOF INIT   ; Cross-8 syntax
```

INCL --- Include Source file

The INCL directive tells Cross-16 to insert a specified source file into the one specified in the command line, or a previous include file.

The INCL directive has the following syntax:

```
label:   INCL   "source_file_name"  ; comment
```

Include source files may only be nested a maximum of 3 deep before a fatal error occurs. A disk drive and/or directory may be included in the file specification. A label may be placed before the INCL directive, which will be assigned the current value of the program counter. All included files are read once each pass, and are included in there entirety in the listing. Should an END directive be encountered in an included file, assembly of all source files will cease at that point. A nonexistant include file will result in a fatal error.

```
                ; An example of a INCL directive is:
                ;
23A4                INCL  "A:NAME.ASM"
```

ORG --- Origin

The Origin (ORG) directive allows the user to specify

the value of the program counter during assembly.

The ORG directive has the following syntax:

```
label:    ORG  expression     ; comment
```

The ORG directive may be used as ofted as desired, but Cross-16 will not flag areas that may be defined more than once in a single source file. The expresion is not optional in this directive and may consist of any numeric constant, character string or formula whose value can be represented in a 16 or 24 bit positive integer, depending on which hexadecimal format has been declared using the HOF directive. Cross-16 will place the new value of the program counter in the address field. Placing no or multiple expressions after the ORG directive will result in an error being flagged.

```
                ; Valid examples of the ORG directive
                ;
0000            RESET:  ORG  0      ; A common begining
0100                    ORG 0100H   ; A famous start

                ; Invalid examples of the ORG directive
                ;
T8000           TWICE:  ORG 8000H,33 ; Multiple exp
E0000           WHERE:  ORG          ; Missing exp
```

PAGE --- Page Control

The Page (PAGE) directive has two functions. In the following format:

```
label:   PAGE       ; comment
```

Cross-16 will insert a form feed into the listing before the next line of the listing. This causes the printer to continue the listing on a new page.

However, the next format:

```
label:  PAGE   expression      ; comment
```

Will set the page length to the value of the expression. Each time the page length is reached, Cross-16 inserts a form feed into the listing. All positive integer values except 1 and 2 are legal. If the expression has a value of zero, or a pagesize is not specified, form feeds will not be inserted into the listing.

```
A000                    ORG 0A000H

                ; Valid examples of the PAGE directive are:
                ;
A000                    PAGE  56
A000                    PAGE  0    ; No form feed
A000                    PAGE  60
A000                    PAGE     ; Form feed before next line
```

SET --- Set

The Set (SET) directive may be used to assign an integer value to a label. It is similar to the EQU directive except that the value of the label may be redefined using additional SET directives elsewhere in the assembly source file:

The SET directive has the following syntax:

```
label:    SET    expression      ; comment
```

Neither the label or the expression are optional in this directive, the expression may consist of numeric constant, character string or formula whose value can be represented as a 32 bit signed integer. Cross-16 places the value of expression in the address field followed by an equal sign "=" to show that this value is not the current value of the program counter. Placing no or multiple expressions after the SET directive, will result in an assembly error being flagged.

The SET directive is most commonly used for controlling conditional assembly.

```
                ; Valid examples of the SET directive
                ;
0000            CPM:      SET 0     ; CPM is false
FFFF            MS_DOS    SET  -1   ; MS_DOS is true

                ; Invalid examples of the SET directive
                ;
T0021           DOUBLE:  SET  33,-6 ; Multiple exp
V0000           TOO_BIG: SET 1FFFFFFFH ; Value too large
```

TITL --- Title of Listing

The title (TITL) directive will cause Cross-16 to place the character string following the directive at the top of each page of the listing.

The title directive has the following syntax:

```
Label:   TITL   "character string"   ; comment
```

Both the label and the comemnt are optional for this directive. The "charater string" is not, however, and must at very minimum be a null string "". There is no limit to the length of the character string, except that a Cross-16 input line cannot exceed 128 characters. Cross-16 does not require TITL directive to produce correctly formatted listing.

```
A800                ORG 0A800H

          ; Valid examples of the TITL directive are:
          ;
A800                TITL "Cross-16 test file"
A800                TITL "User's choice"
```

THE CROSS-16 LISTING FILE

If requested using the -L directive in the command line, Cross-16 will produce a listing file during the second pass of the assembly source file.

The HOF directive partially controls the format of the assembled listing. If a HOF directive is not used, or one or the 16 bit hexadecimal formats is declared. Cross-16 will produce a listing with a 32 bit value preceding the EQU and SET directives, 24 bit addresses and up to 7 bytes of code on each line. If one of the 8 bit hexadecimal formats is declared, Cross-16 will produce listing with a 16 bit value preceding the EQU and SET directives, 16 bit addresses, and up to 5 bytes of code on each line. This allows the listing format correspond to the address word of the target processor.

Listing Format

The listing is the original assembly source file, with 16 or 24 additional ASCII characters inserted at the begining of each line. Should an assembly error occur, an alphabetic error code will be placed in the first column of the listing, which is normally a " " blank. The next four or six characters represent the hexadecimal value of the assembly instruction or assembler directive. This value will not be displayed after the fifth or seventh byte, but will be placed in the hexadecimal file in its entirety.

The EQU, SET directives are exceptions to these rules.

Assembly Error Codes

When Cross-16 detects an error during assembly, a single character error code is placed in the first column of the assembly listing on the line in which the error is occurred. The error code and assembled line are also shown on the video display, so that the user does not have to search the listing to check for assembly errors. As Cross-16 seaches the instruction table, it will display the first error code generated for the given assembly line, even though other errors may occur. Therefore, certain combinations of instructions and format errors can produce a misleading error code. If the code does not seem to be relevant, look for other possible errors. Errors are included thoughout the directive and instruction examples in this manual, and the alphabetic error codes represent the following:

A ASCII error - Illegal format of an ASCII character string, usually a missing quotation (")

B Bracket error - Unequal numbers of left and right script bracket in an expression []

C Expression error - Unable to evaluate expression as written

H Hex format error - Illegal hexadecimal output format specified in HOF directive

L Line error - Unexpected characters found in assembly line

P Phase error - Value of label changes with succesive assembler passes.

S Syntax error - Cross-16 assembler directive or processor instruction cannot be identified

T Too many errors - Too many arguments are contained in the Cross-16 assembler directive or processor instruction

U Undefined label - Label used in expression or opcode is undefined. Often caused by hex constant which does not start with a number

V Value error - Operand defined in Cross-16 assembler directive or processor instruction is incompatible with given value.

Y Symbol error - Missing or Illegal symbol (Label)