Z90361 OTP
Z90365 ROM

32 KWORD TELEVISION CONTROLLER WITH OSD

PRODUCT SPECIFICATION

PS002500-TVC1099

ZiLOG Worldwide Headquarters
910 E. Hamilton Avenue
Campbell, CA 95008
Telephone: 408.558.8500
Fax: 408.558.8300
www.ZiLOG.com

Document Disclaimer

Table of Contents

List of Figures

List of Tables

Z90365 ROM and Z90361 OTP
32 KWord Television Controller with OSD

1 OVERVIEW

The Z90365 and Z90361 are the ROM and OTP versions of a Television Controller with On-Screen Display (OSD) that contains 32 KWords of program memory and 640 words of RAM.

- The **Z90361** is the one-time programmable (OTP) controller used to develop code and prototypes for specific television applications or initial limited production. Program ROM and Character Generation ROM (CGROM) in the Z90361 are both programmable.

  The Z90361 requires Zilog's Z90369ZEM Emulator with its proprietary Zilog Developmental Studio (ZDS) software for programing. To view code effects, the emulator uses a ZOSD board which connects directly to a television screen. Refer to Figure 1.

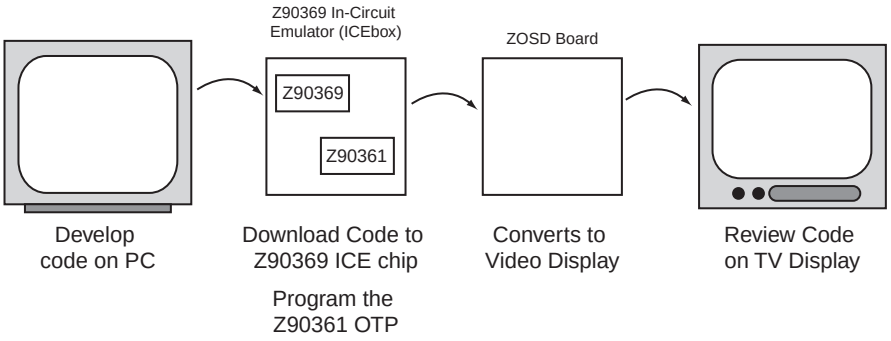


Figure 1 Code Development Environment

- The **Z90365** incorporates the ROM code developed by the customer with the Z90361. Customer code is masked into both program ROM and CGROM.

The Z90365 Television Controller with OSD is based on the ZiLOG's Z89C00 RISC processor core. The Z89C00 is a second generation, 16-bit, fractional, two's complement CMOS Digital Signal Processor (DSP). Most instructions are accomplished in a single clock cycle. The processor features a 24-bit Arithmetic Logic Unit (ALU) and a 24-bit Accumulator. The processor also contains a six-level stack with three vector interrupts. (Note: The multiplier of Z89C00 is disabled in the Z90365 controller.)

The Z90365 contains 32K words of program ROM and 640 words of on-chip data RAM. This device can support up to 512 characters with a16x16, 16x18 and 16x20 programmable matrix. In addition, the Z90365 contains four external register banks with eight registers in each.

The internal 24MHz/2 system clock has a Phase Lock Loop (PLL) controlled by an external 32.768KHz crystal.

A five channel, 4-bit Analog to Digital Converter (ADC) supports

- multiple tuner automatic frequency tuning (AFT)
- analog keypad entry
- audio level input
- Vertical Blank Interval (VBI) data capture.

Seven Pulse Width Modulator (PWM) outputs allow low cost digital to analog conversion (DAC). Five PWMs have 8-bit resolution to control video and audio attributes. Two PWM have 14-bit resolution to control an external Voltage Synthesis Tuner (VST).

A Master/Slave I^2C (Inter Integrated Circuit) bus interface provides serial system interconnect to common peripheral functions.

Twenty programmable I/O pins provide flexibility for other digital input/output functions.

An IR (InfraRed) remote capture register facilitates reliable remote data capture.

On-chip Horizontal Synchronization (H_{SYNC}) and Vertical Synchronization (V_{SYNC}) circuits generate a video timebase (typically used for VCR and set-top applications) in the absence of an available video signal.

Micro-programmable OSD generation logic provides flexibility to tailor OSD features and functions. In addition to normal OSD functions, Closed Captioning is supported in accordance with FCC Report and Order on GEN Docket No. 91-1, dated April 12, 1991. Expanded Data Service (XDS) capability can be implemented as well.

The Z90365 is packaged in a 42-pin SDIP package.

Figure 2 is a block diagram of the internal structure of the chip . Figure 3 illustrates the pin locations and Table 1 describes the function of each pin.

1.1 Block Diagram

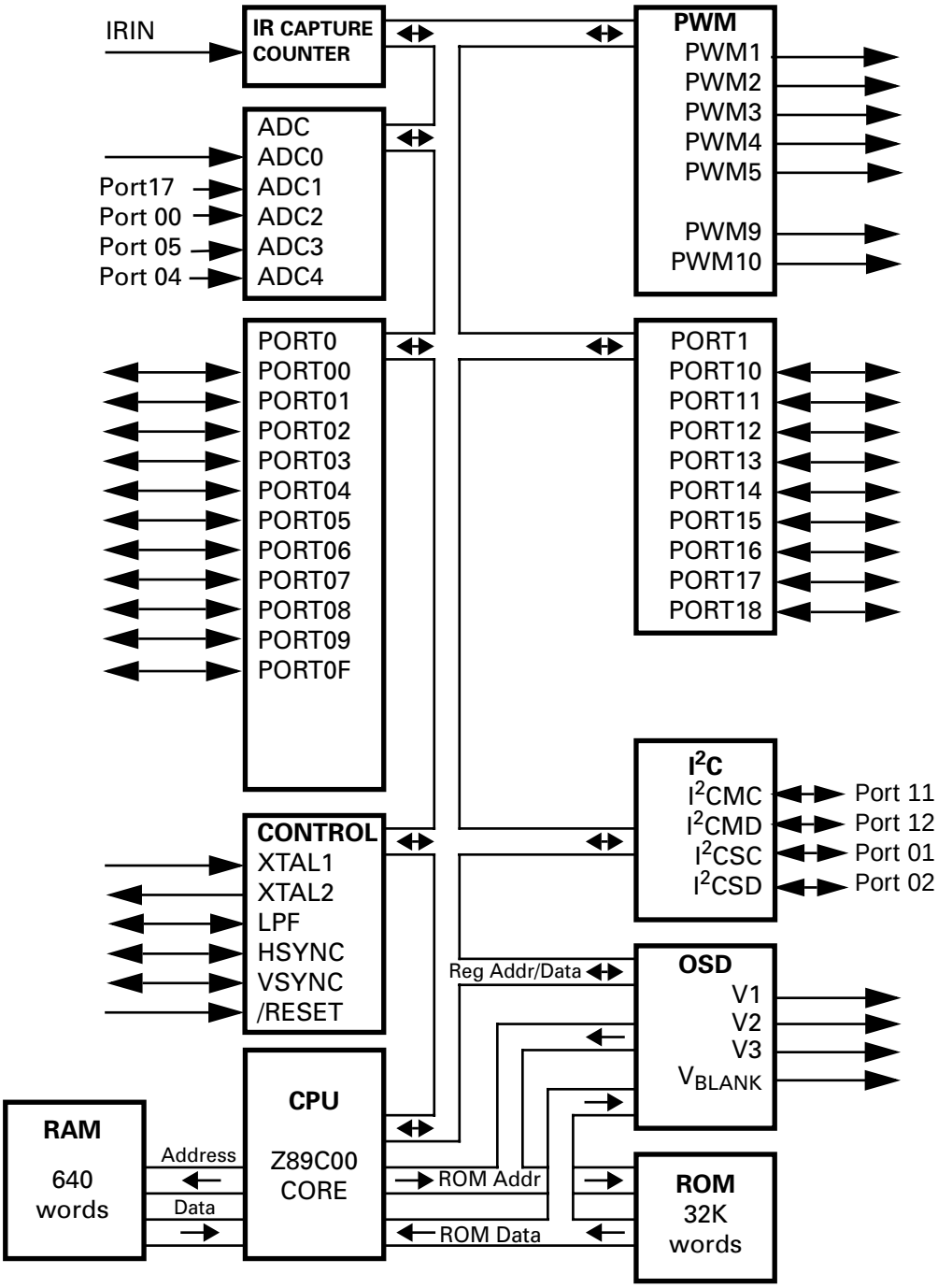


Figure 2 Block Diagram

1.2 Pin Description

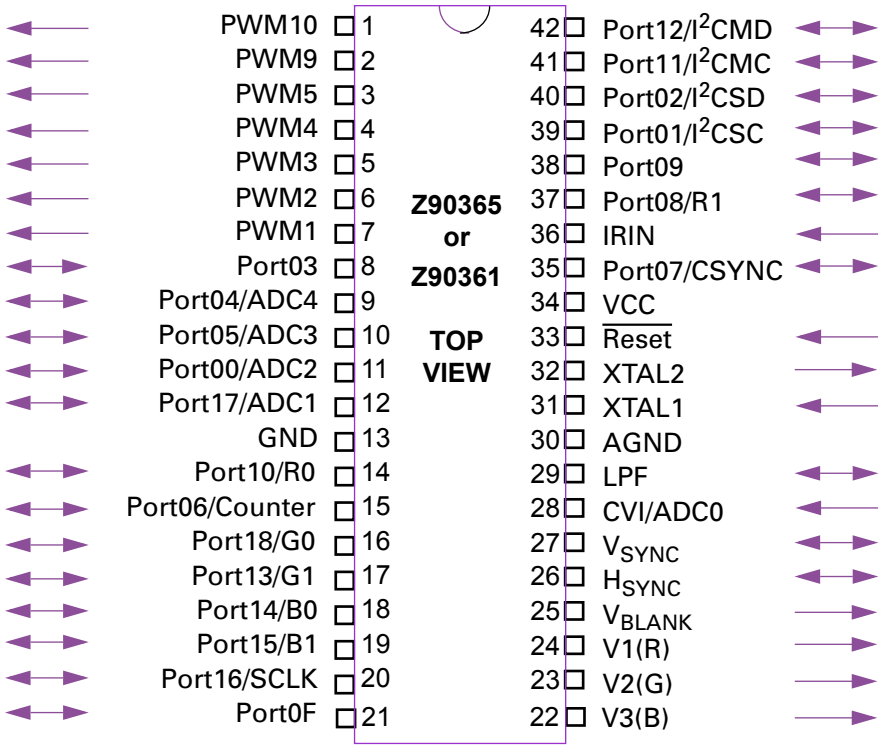


Figure 3 42-Pin SDIP Pinout

Table 1 Z90365 or Z90361 Pin Description

Symbol	Pin #	Function	Direction
VCC	34	+5 Volts	Power
GND	13, 30	0 Volts	Power
IRIN	36	InfraRed remote capture input	Input
ADC[4:0][1]	9, 10, 11, 12, 28	4-Bit Analog to Digital Converter input[2].	Analog Input

Table 1 Z90365 or Z90361 Pin Description

Symbol	Pin #	Function	Direction
PWM10, PWM9	1, 2	14-Bit Pulse Width Modulator output	Output
PWM[5:1]	3, 4, 5, 6, 7	8-Bit Pulse Width Modulator output	Output
PORT0[F;9-0]	21, 38, 37, 35, 15, 10, 9, 8, 40, 39, 11	Bit programmable Input/ Output ports	Input/Output
PORT1[8:0]	16, 12, 20, 19, 18, 17, 42, 41, 14	Bit programmable Input/ Output ports	Input/Output
I^2CMC[3]	41	Master I^2C Clock I/O	Input/Output
I^2CMD[4]	42	Master I^2C Data I/O	Input/Output
I^2CSC[5]	39	Slave I^2C Clock I/O	Input/Output
I^2CSD[6]	40	Slave I^2C Data I/O	Input/Output
XTAL1	31	Crystal oscillator input	Analog Input
XTAL2	32	Crystal oscillator output	Analog Output
LPF	29	LOOP FILTER	Analog Input/Output
H_{SYNC}	26	H_SYNC	Input/Output
V_{SYNC}	27	V_SYNC	Input/Output
$\overline{RESET}$	33	Device Reset	Input
V3, V2, V1	22, 23, 24	OSD video output. Typically drive B, G and R outputs.	Output/Analog Output
V_{BLANK}	25	OSD Blank Output	Output
RGB digital outputs[7]	37, 14, 17, 16, 19, 18	R[1:0], G[1:0] and B[1:0] outputs of the RGB matrix	Output
SCLK[8]	20	internal processor SCLK	Output
CSYNC[9]	35	Composite Sync Output	Output
COUNTER[10]	15	Counter Input/Output	Input/Output

Table 1 Z90365 or Z90361 Pin Description

Symbol	Pin #	Function	Direction
1	ADC1 input pin is shared with Port17, ADC2 input pin is shared with Port00. ADC3 input pin is shared with Port05 and ADC4 input pin is shared with Port04.		
2	ADC0 has a lamp circuit which facilitates Composite Video input.		
3	I^2CMC I/O pin is shared with Port 11		
4	I^2CMD I/O pin is shared with Port12		
5	I^2CSC I/O pin is shared with Port 01		
6	I^2CSD I/O pin is shared with Port02		
7	Digital RGB outputs are shared with Port1 and Port0 pins		
8	Internal processor SCLK is shared with Port16		
9	CSYNC is shared with Port07		
10	Counter is shared with Port06		

2 OPERATION

2.1 CPU Descriptions

The Z89C00 core is a high-performance DSP which has a modified Harvard-type architecture with separate program and data memories. The design has been optimized for processing power and silicon space.

The Z89C00 used in the Z90365 device is modified. The multiplier is disabled and is not accessible. However, the X and Y registers in the multiplier are still available and can be used as a general purpose registers. See the Zilog Z89C00 datasheet.

ALU

The 24-bit ALU has two input ports, one of which is connected to the output of the 24-bit Accumulator. The other input is connected to the 24-bit P-Bus, the upper 16-bits of which are connected to the 16-bit D-Bus.

Instruction Timing

Several instructions are executed in one machine cycle. Long immediate instructions and Jump or Call instructions are executed in two machine cycles. When the program memory is referenced in internal RAM indirect mode, it requires three machine cycles. An additional machine cycle is required if the PC is selected as the destination of a data transfer instruction. This only occurs in the case of a register indirect branch instruction.

Hardware Stack

A six-level hardware stack is connected to the D-Bus to hold subroutine return addresses or data. The CALL instruction pushes PC+2 onto the stack. The RET instruction pops the contents of the stack to the PC.

CPU Registers

The Z90365 has 11 physical internal registers and four external register banks with eight registers in each. In addition, it has nine virtual registers. The 11 internal registers are defined in Table 2 and the status register is defined in Table 3.

Internal Registers

Table 2 Internal Registers

Register	Register Definition
X	X, 16-bit
Y	Y, 16-bit
A	Accumulator, 24-bit
SR	Status Register, 16-bit
Pn:b	Six RAM Address Pointers, 8-Bit each
PC	Program Counter, 16-Bit

X and **Y** are 16-bit general purpose registers.

A is a 24-bit Accumulator. The output of the ALU is sent to this register. When 16-bit data is transferred into this register, it goes into the 16 MSB's and the least significant eight bits are set to zero. Only the upper 16 bits are transferred to the destination register when the Accumulator is selected as a source register in transfer instruction.

SR is the Status Register which contains the ALU status and the control bits listed in Table 3. The status register is always read in its entirety. S15-S12 are set/reset by the hardware and can only be read through software. They are set or reset by the ALU after an operation.

S8-S0 can be written by software. S8, if 0 (reset), allows the hardware to overflow. If S8 is set, the hardware clamps at maximum positive or negative values instead of overflowing. S7 enables interrupts. S6 - S5 are used for "short form direct" addresses which are described below. The definitions of S2-S0 are listed in Table 4.

Table 3 Status Register

Bit/Field	Bit Position	R/W	Description
N	15	R	ALU Negative
OV	14	R	ALU Overflow
Z	13	R	ALU Zero
L	12	R	Carry
Reserved	11	R	Reserved
Reserved	10	R	Reserved
Reserved	9	R	Reserved
OP	8	R/W	Overflow Protection
IE	7	R/W	Interrupt Enable
Register Bank Selector	6,5	R/W	00 Register Bank 0 01 Register Bank 1 10 Register Bank 2 11 Register Bank 3
SFD	4,3	R/W	“Short Form Direct” Bits
RPL	2-0	R/W	RAM Pointer Loop Size

Table 4 RPL Description

S2	S1	S0	Loop Size
0	0	0	256
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	16
1	0	1	32
1	1	0	64
1	1	1	128

Pn:b are the pointer registers for accessing data RAM.
(n= 0, 1, 2 refer to the pointer number)
(b = 0, 1 refers to RAM bank 0 or 1).

Pointer registers can be directly read from or written to, and can point directly to locations in data RAM or indirectly to Program Memory.

PC is the Program Counter. When this register is assigned as a destination register, one NOP machine cycle is automatically added to adjust the pipeline timing.

External Registers

The Z90365 module is capable of directly accessing up to eight external registers using only the three external register address signals that are normally available. In this implementation, two user bits (Status register S6-S5) are combined with the register address signals to provide the ability to address four banks of eight registers each. The registers most critical for speed are located together in Bank 3. In this specification, all external registers are referred to

RX(Y)<Z>

where:

X is a register number within a register bank;
Y is a bank number; and
Z is a bit field number

A register can be selected by setting bits 6 and 5 in the status register to define the bank, then specifying the address of the register on the external register address bus.

External registers reside on the chip and are used to control the operation of all the peripheral modules in the device. By reading or writing to the fields in the external registers, the user can interact with the peripheral devices on the chip.

Virtual Registers

BUS is a read-only register which, when accessed, returns the contents of the D-Bus. It is a virtual register (physical RAM does not exist on the chip).

Dn:b These eight data pointers refer to possible locations in RAM that can be used as pointers to locations in program memory. The programmer decides which location to choose two bits from in the status register and which two bits in the operand. Thus, only the lower 16 possible locations in RAM can be specified. At any one time there are eight usable pointers, four per bank, and the four pointers are in consecutive locations in RAM.

For example, if S3/S4 = 01 in the status register, then D0:0/D1:0/D2:0/D3:0 refers to locations 4/5/6/7 in RAM bank 0. Note that when the data pointers are being written to, a number is actually being loaded to Data RAM, so they can be used as a limited method for writing to RAM.

RAM Addressing

The addresses in RAM can be specified in one of three ways. Refer to Figure 4.

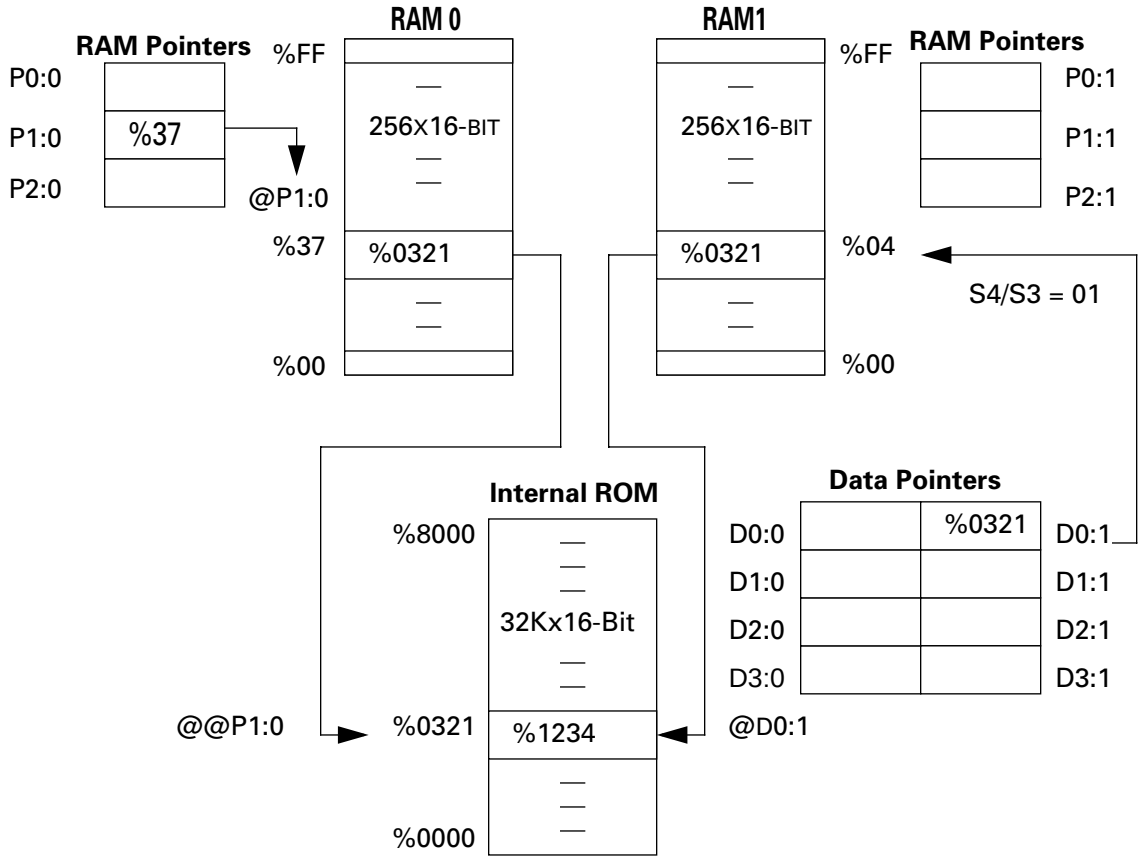


Figure 4 RAM, ROM, and Pointer Architecture

2.2 Memory (ROM and RAM)

The Z90365 has 32K words of Read Only Memory (ROM) and 640 words of Random Access Memory (RAM).

ROM

A 32K word mask ROM provides storage for both Program Memory (PROGROM) and Character Generation ROM (CGROM). The address boundary between these applications is dependent on the storage required for character graphics.

The program ROM section can, in theory, be placed anywhere in the addressable ROM space; however, because CGROM usually starts at location 0000H, PROGROM resides in the higher address locations. The maximum available ROM space for program memory depends on the ROM reserved for CGROM (for an application) and the ROM size of the device selected.

The size of memory used as CGROM depends on the number and resolution of characters stored in memory. An **n** represents the number of characters stored, ranging from 0 to 256. If characters are 16 × 18 pixels, then the upper region of memory starting at the 4K boundary is used for character storage. If not, that region can be used as program memory.

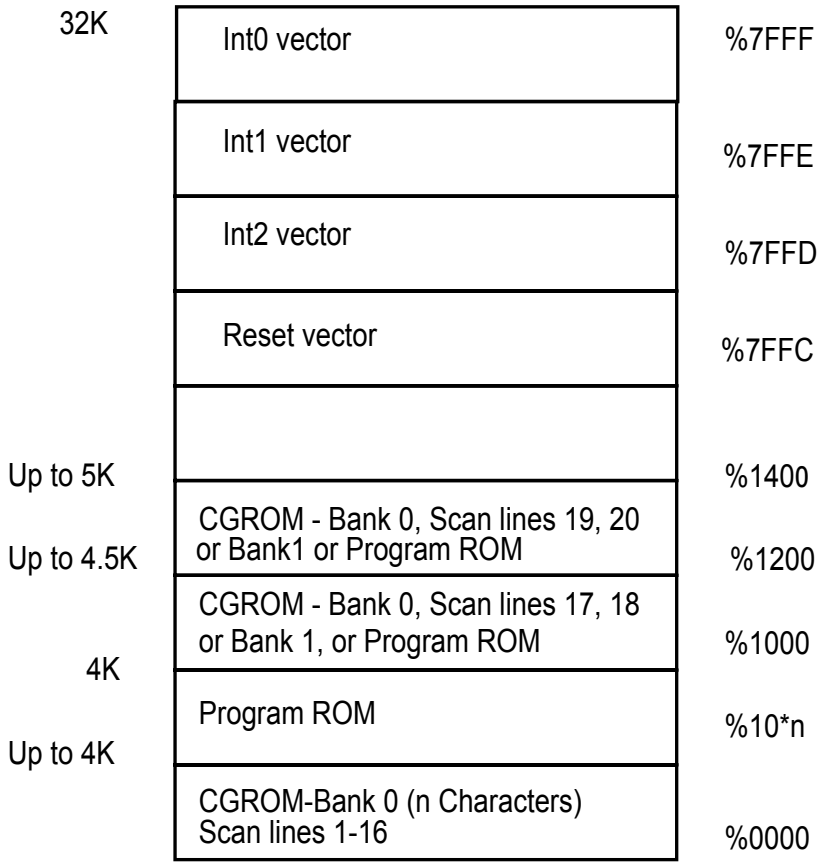


Figure 5 ROM Map

RAM

RAM is organized in banks of 256 words of 16 bits each. Bank 1 is always accessible. Bank0 is mapped to two banks with 256 and 128 words each. Total RAM size is 640 words. See Figure 6.

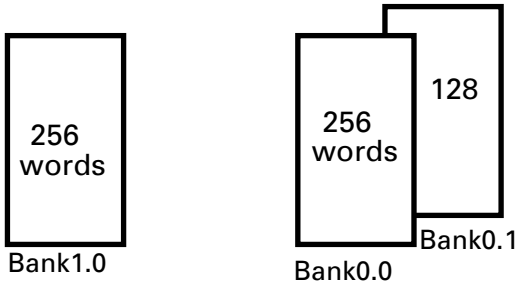


Figure 6 RAM Allocation

2.3 Clock Circuit Description

The processor is able to operate from several clock sources.

- Primary Phase Lock Loop VCO source (PVCO)
- Secondary Phase Lock Loop (SVCO)
- 32.768-kHz oscillator clock (OSC)

In addition, the processor clock can be halted temporarily to select the clock source or access ROM without disrupting normal operation of the processor.

An external crystal controls the internal 32.768-kHz oscillator. The crystal is used as the clock reference for the internal Phase Locked Loop (PLL). The PLL provides the internal PVCO clock for processor operation. SCLK is generated internally by dividing the frequency of an appropriate oscillator (PVCO) by 2. The frequency of the SCLK after POR is 12.058 MHz.

The SCLK signal can be sent to the Port16 output pin under software control by setting bit9 in register R3(10). The SVCO must be used as the system clock when the OSD is generated.

The clock switch control register R6(1) defines the source of the SCLK for the Z90365 core. The block diagram in Figure 7 represents the clock switch circuit.

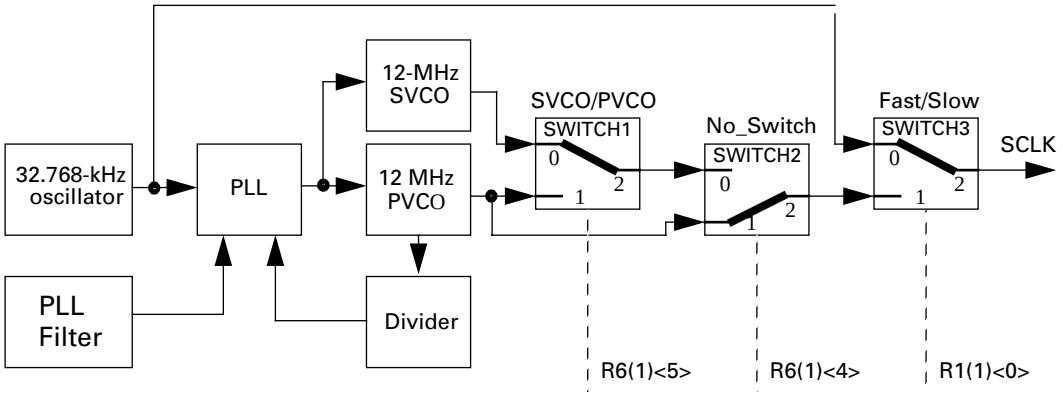


Figure 7 Clock Switching Block Diagram

Input/Drive Circuits

The 32KHz oscillator circuit in Figure 8 is suggested for proper clock operations.

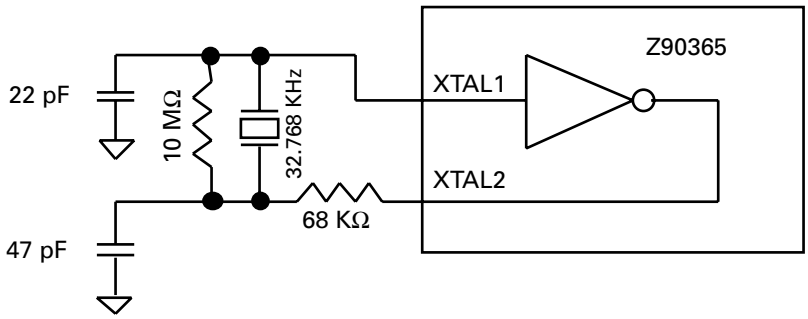


Figure 8 32KHz Oscillator Recommended Circuit

2.4 Reset Conditions

Reset conditions including addresses and registers are listed in Table 5.

Table 5 Reset Conditions

		Reset Condition															
Addr	**Register**	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**	**Comments**
R0(0)	PWM9	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	PWM9
R1(0)	PWM10	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	PWM10
R2(0)	pll_freq	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	PLL frequency control
R3(0)	I^2C_int	0	0	0	0	0	1	x	x	x	x	x	x	x	x	x	x	I^2C interface register
R4(0)	port0	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	11-bit I/O port 0
R5(0)	port1	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	9-bit I/O port 1
R6(0)	dir0	x	x	x	x	x	1	1	1	1	1	1	1	1	1	1	1	11-bit port 0 direction
R7(0)	dir1	x	x	x	x	x	x	x	1	1	1	1	1	1	1	1	1	9-bit port 1 direction
R0(1)	clamp_pos	1	0	0	0	0	0	0	0	0	x	x	x	x	x	x	x	Position of video clamp pulse
R1(1)	sclk_freq	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Stop/sleep/normal mode
R2(1)	WDT/STOP	x	x	x	x	x	x	x	x	x	0	0	0	0	0	0	0	Stop and WDT, 9-bit counter
R3(1)	standard_ctl	0	0	0	0	0	0	0	0	0	0	0	x	x	x	0	0	Output H/VSYNC/ Blink Control
R4(1)	ADC_ctl	0	0	0	0	0	0	0	0	0	0	0	0	x	x	x	x	A/D converter control
R5(1)	cap_1s_ctl	x	x	x	x	0	0	x	x	x	x	x	x	x	x	x	x	Counter timers control
R6(1)	clock_ctl	0	0	0	0	0	0	0	0	0	0	0	1	x	x	x	x	Clock control (switch VCO/DOT)
R7(1)	wdt_smr_ctl	x	x	x	x	x	x	x	x	0	x	x	x	x	x	x	x	SMR and WDT control/interrupt
R0(2)	pwm1_data	0	0	0	0	0	0	0	0	x	x	x	x	x	x	x	x	8-bit PWM 1 data

Table 5 Reset Conditions (Continued)

R1(2)	pwm2_data	0	0	0	0	0	0	0	0	x	x	x	x	x	x	x	x	8-bit PWM 2 data
R2(2)	pwm3_data	0	0	0	0	0	0	0	0	x	x	x	x	x	x	x	x	8-bit PWM 3 data
R3(2)	pwm4_data	0	0	0	0	0	0	0	0	x	x	x	x	x	x	x	x	8-bit PWM 4 data
R4(2)	pwm5_data	0	0	0	0	0	0	0	0	x	x	x	x	x	x	x	x	8-bit PWM 5 data
R5(2)	Reserved	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	Reserved
R6(2)	Reserved	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	Reserved
R7(2)	Reserved	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	Reserved
R0(3)	hi_x2_hi_x3	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	Character multiple/ current data
R1(3)	lo_x2_mid_x3	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	Character multiple/ next or previous data
R2(3)	Ch_x1_lo_x3	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	Character multiple/ character graphics attribute
R3(3)	attr_data	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	Character attribute/ video RAM data
R4(3)	osd_cntl	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	On screen display control
R5(3)	cap_data	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	Capture register data
R6(3)	palette_color	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Display palette color/underline color
R7(3)	output palette	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Output palette

2.5 Power Management

There are two low power operating modes for Z90365: SLEEP mode and STOP mode.

SLEEP Mode

In SLEEP mode, the controller uses the 32.768 KHz clock for the SCLK to reduce power consumption.

STOP Mode

In STOP mode, the processor is suspended and the power consumption is minimized.

2.6 I/O Port Configurations

User control can be monitored either through the keypad scanning port or the 16-bit remote control capture register.

Two input/output port blocks are available for general purpose digital I/O application. Each port bit can be programmed to be either input or output. To conserve the device pin count, some port pins are mapped to provide I/O to the ADC converter block and I^2C interface block.

The 20 configurable I/O pins are general purpose pins for functions such as serial data I/O, LED control, key scanning, power control and monitoring, and I^2C serial data communications.

Port `0` and `1` directions are defined in `R6(0)` and `R7(0)` respectively. `R4(0)` and `R5(0)` are data registers for both Ports `0` and `1`. Figures 9, 10 and 11 show I/O configuration and sharing with other functional units.

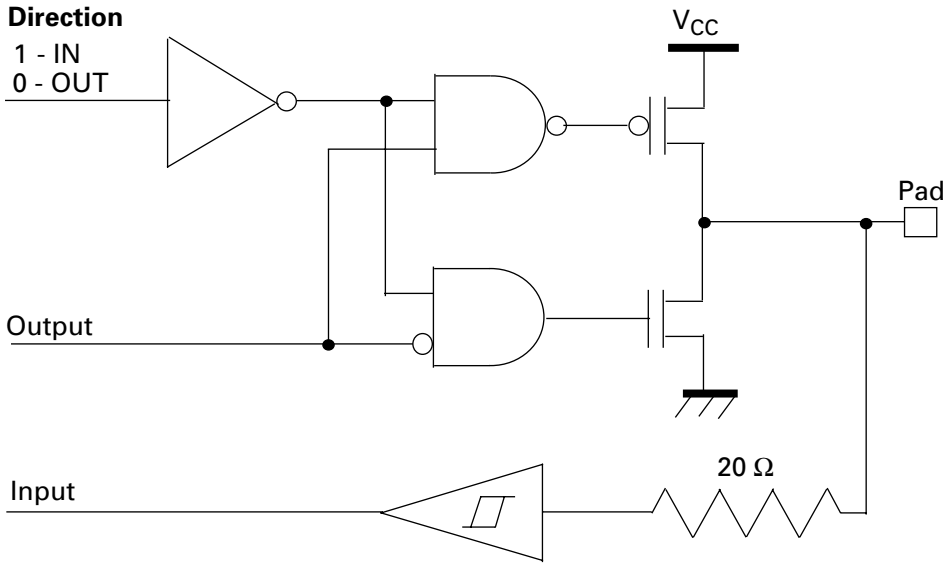


Figure 9 Bidirectional Port Pins

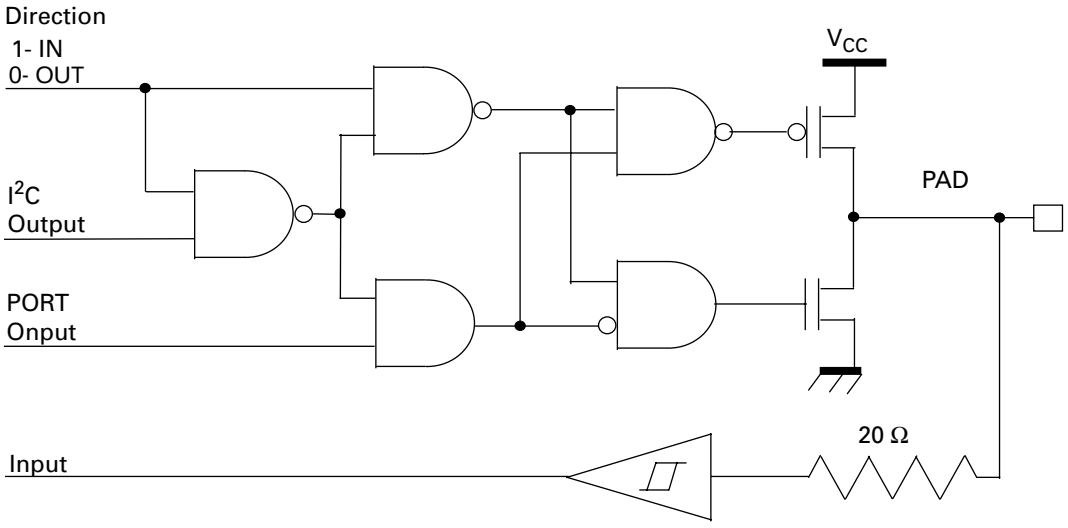


Figure 10 Bidirectional Pins Multiplexed with I^2C Port

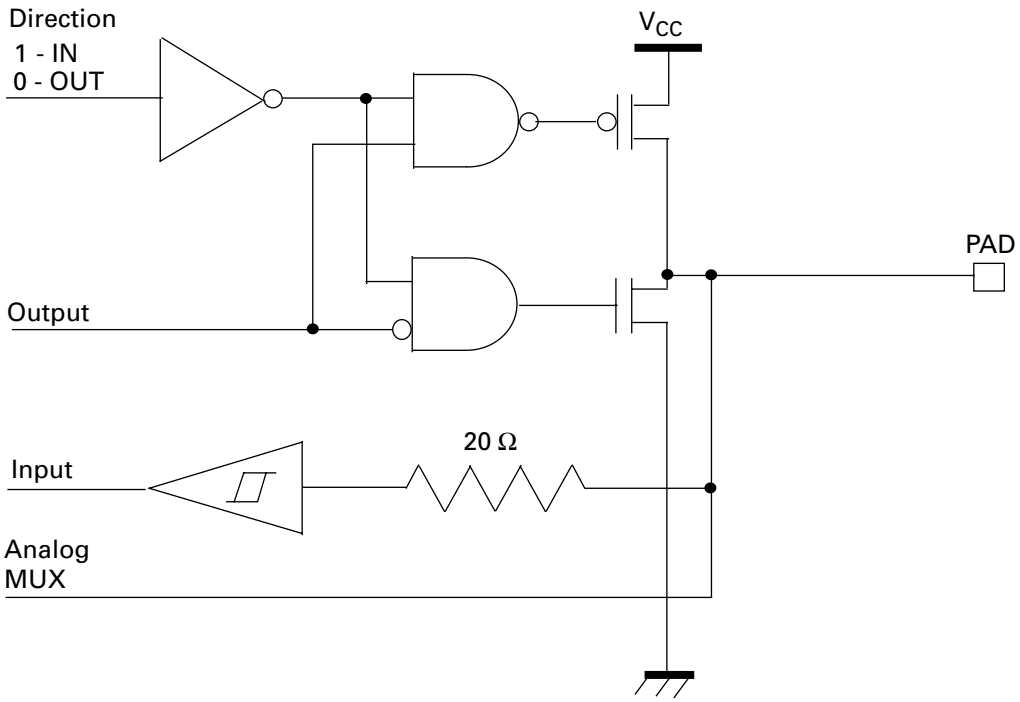


Figure 11 Bidirectional Pins Multiplexed with ADC Inputs

2.7 Interrupts

The Z90365 has three external interrupt signals. There are four interrupt sources.

- Horizontal sync (H_{SYNC})
- Vertical sync (V_{SYNC})
- Capture timer
- External event (Port09).

All interrupts are vectored. The capture timer and Port09 are multiplexed to the same interrupt.

Interrupt priorities are programmable. Each interrupt can be masked by setting fields in the external registers.

When the Z90365 receives an interrupt request from one of the interrupt sources, it executes the interrupt service routine directly for that source.

External register R7(1) controls interrupts.

2.8 Timers

Watch-Dog Timer

The watch-dog timer resets the CPU while it times out.

External register R7(1) controls the watch-dog timer.

Real Time Clock

A clock timer, in real time, generates ticks every 1000, 250, 62.5 or 15.625 ms. External register R5(1) controls the real time clock.

IR Capture Timer

A capture timer measures time between edges of the IR signal. This timer can be programmed to measure timing from rising-to-rising, falling-to-rising, rising-to-falling, or falling-to-falling edges.

IR capture timer is controlled by External register R5(1). Figure 12 is a block diagram of the IR capture register structure.

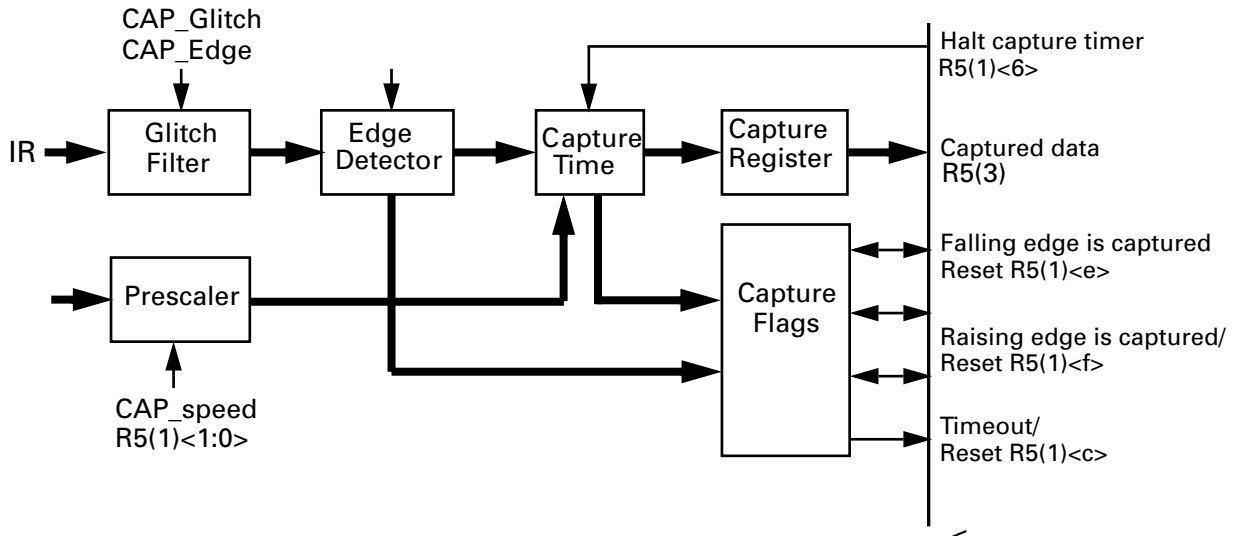


Figure 12 IR Capture Register Block Diagram

2.9 ADC

The Analog to Digital Converter (ADC) is a 4-bit resolution, flash A to D converter. The user program controls the five-to-one analog input multiplexor and conversion start circuits. The 4-bit conversion result can be read by the CPU after each conversion.

The ADC0 input channel is dedicated to VBI (vertical blank interval) data slicing for subsequent digital signal processing. This channel has a special video clamp circuit that provides DC restoration of the composite video input signal. Typical VBI applications include Line 21 Closed Caption, Electronic Data Services, and StarSight Telecast. The range for ADC0 is from 1.5 to 2.0 V.

The four remaining channels, ADC1, ADC2, ADC3, and ADC4, are general purpose ADCs which are normally used to implement tuner automatic frequency control and analog key entry. These four channels have a range from 0 to 5.0 V.

The range for input signals differs according to ADC inputs. Refer to Table 6.

Table 6 ADC Inputs Typical Range

Input	Range (V)	Clamping	Typical application
CVI/ADC0	1.5–2.0	Yes (Ref–)	CCD sampling input
ADC1/P17	0–5.0	No	AFC input
ADC2/P00	0–5.0	No	Key scanning input
ADC3/P05	0–5.0	No	Key scanning input
ADC4/P04	0–5.0	No	Key scanning input

Reference voltages that have been generated internally define the maximum range of the input signal for the ADC.

Nominal values are

Ref+ = 2.0 V

Ref– = 1.5 V @ V_{CC} = 5.0 V

For other V_{CC} values, the reference voltages must be prorated as

Ref+ = 0.4 * V_{CC}

Ref– = 0.3 * V_{CC}

The maximum sampling rate of the ADC converter is 3 MHz. It takes 4 SCLK cycles for valid output data from the ADC to become available. This is especially important if the application uses single shot mode.

The ADC exhibits monotonous conversion characteristics with a nonlinearity of less than 0.5 LSB. ADC0. The ADC has a range of 0.5V (from 1.5V to 2.0V) and is directly multiplexed to the input of the ADC. The remaining ADC inputs (ranging from 0V to 5V) use AGND and AV_{CC} voltage as a reference.

Figure 13 is a block diagram of the ADC inner structure.

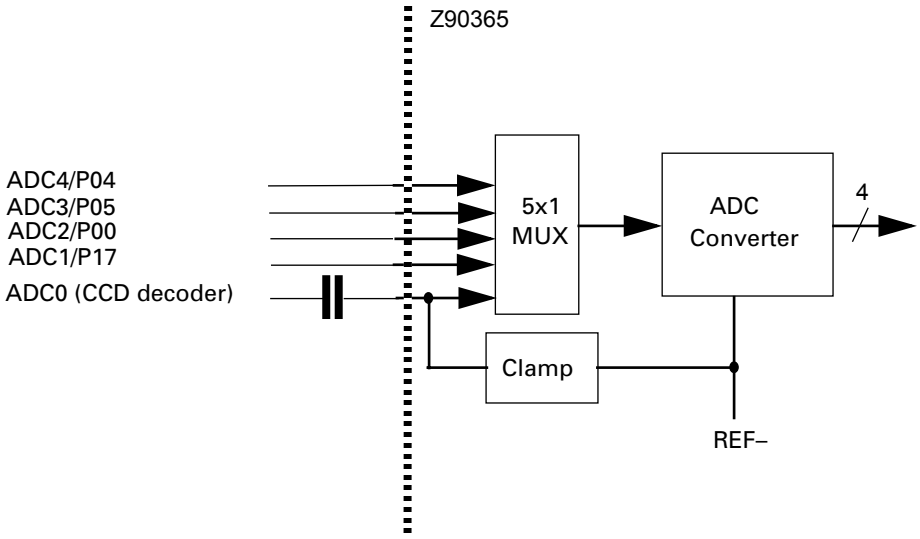


Figure 13 ADC Block Diagram

2.10 Pulse Width Modulation

Pulse Width Modulation is used in conjunction with external low-pass filters to perform digital to analog conversion. Five 8-bit PWMs, (PWM1-PWM5) generate signals to control video and sound attributes. PWM9 and PWM10 are 14-bit PWMs used with an external circuit to generate the control voltage for voltage synthesis tuners. When a chassis has a frequency synthesis tuner, these PWMs can also control video or sound attributes.

Each PWM circuit features a data register with settings under program control. The data in the register determine the ratio of PWM High to PWM Low time. PWM data registers are NOT initialized when reset. To eliminate potential problems with PWM output, initialize PWM data registers BEFORE enabling the VCOs.

External registers R0(2) to R5(2) are data registers for PWM1 to PWM4. External registers R0(0) and R1(0) are data registers for PWM9 and PWM10.

Figure 16 shows PWM circuits.

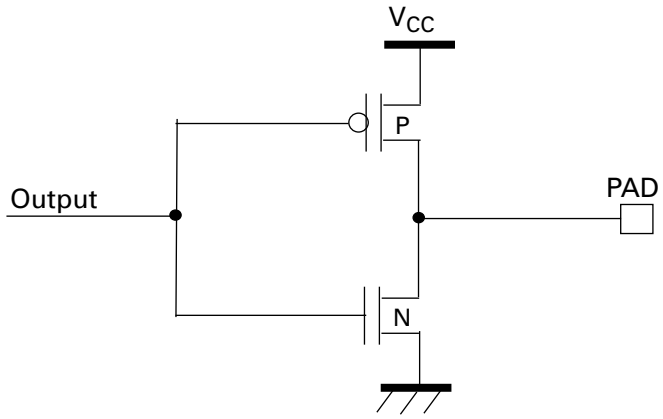


Figure 14 PWM9 and PWM10 Output Circuits

2.11 I²C Interface

There are two hardware modules which support standard I²C bus protocol according to the I²C bus specification published by Phillips in 1992, entitled *I²C Peripherals for Microcontrollers Data Handbook*.

The first module, the Master, can be configured for fast (400 kHz) or slow (100 kHz) bit rates and can be used in applications with a single master.

The second module, the Slave, supports a 7-bit addressing format with both fast and slow bit rates.

Table 7 lists the bit rates for the Master I²C Bus.

Table 7 Master I²C Bus Bit Rates

Mode	I²C mode	Bit Rate	Actual Bit Rate
LO/Fast	Slow	0–100kHz	91kHz
HI/Fast	Fast	0–400kHz	334kHz

To suppress possible problems on both data (SDA) and clock (SCL) lines, digital filters are implemented on all inputs of the I^2C bus interface. These filters have a time constant equal to $3T_{SCLK}$ = 250 ns.

If the master or slave I^2C interface is enabled, corresponding I/Os (Port01 and Port02 for the slave, Port11 and Port12 for the master) must be assigned as outputs.

Master and Slave modules cannot be used simultaneously because of the shared register, (see the Register 3(0) data field). The software activates I^2C modules by writing appropriate commands into the control register. To control the I^2C bus interface, the control register R3(0) toggle bit <c> must point to an appropriate interface (Master or Slave).

M_disable or **S_disable** bits allow either the Master or Slave I^2C interface to be disabled so not to interfere with any activity associated with the Port pins. At Power-on Reset (POR), both I^2C interfaces are enabled. To use the I^2C interface, the corresponding Port pin (multiplexed with the I^2C Data and Clock) must be configured as an output, while M_disable or S_disable bits must be reset to `0`.

External register R3(0) controls the I^2C. Table 8 lists the Master I^2C bus interface commands. Table 9 lists the Slave I^2C bus interface commands. Figures 15 and 16 are flow charts of the Master and Slave modes.

Table 8 Master I^2C Bus Interface Commands

Command	Notes/Function
0 0 0	This command sends a start bit, followed by an address byte specified in the “data” field (bits <7:0>), then fetches an acknowledgment in bit <0>. This command initializes communication, and generates 9 SCL cycles.
0 0 1	This command sends one byte of data specified in the “data” field (bits <7:0>), then fetches an acknowledgment in bit <0>. This command is used in a WRITE frame, and generates 9 SCL cycles.
0 1 0	This command sends bit <7> as an acknowledgment (ACK = 0, NAK = 1), then receives a data byte. This command is used in a READ frame when the next data byte is expected, and generates 9 SCL cycles. Received data appears in the “data” field (bits <7:0>).
0 1 1	This command sends bit <7> as an acknowledgment (ACK = 0, NAK = 1). This command is used in a READ frame to terminate data transfer, and generates one SCL cycle.
1 0 0 1 0 1	A NULL operation. This command must be used with a “RESET” bit <b> and/or a “TOGGLE” bit <c>. Using the “RESET” and/or “TOGGLE” bits with any other command, interferes with the logic of the I^2C interface.
1 1 0	This command receives one data byte. It is used in a READ frame to receive the first data byte after the address byte is transmitted. It generates 8 SCL cycles.
1 1 1	This command sends a stop bit, and generates one SCL cycle.

Table 9 Slave I^2C Bus Interface Commands

Command	Notes/Function
0 0 0	Reserved. Cannot be used.
0 0 1	This command sends bit <7> as an acknowledgment (ACK = 0 only), then receives one data byte. This command is used in a WRITE frame and requires 9 SCL cycles. Received data is read as a “data” field (bits <7:0>).
0 1 0	This command sends one byte of data specified in a “data” field (bits <7:0>), then fetches an acknowledgment in bit <0>. This command is used in a READ frame and requires 9 SCL cycles.
0 1 1	Reserved. Can not be used.
1 0 0 1 0 1	A NULL operation. This command must be used with a “RESET” bit <b> and/or “TOGGLE” bit <c>. Using the “RESET” and/or “TOGGLE” bits with any other command, interferes with the logic of the I^2C interface.
1 1 0	This command sends a bit <7> as a not acknowledgment (NAK = 1 only) in a WRITE or READ frame. This command terminates I^2C communication and requires one SCL cycle. The “Sulfonamide” bit <a> is automatically reset when a “busy” bit <9> goes Low. This command sends a bit <7> as an acknowledgment (ACK = 0 only) in a READ frame and requires one SCL cycle. The Send data command (010) must be executed next. This command acknowledges an address byte in a READ frame.
1 1 1	Reserved. Cannot be used.

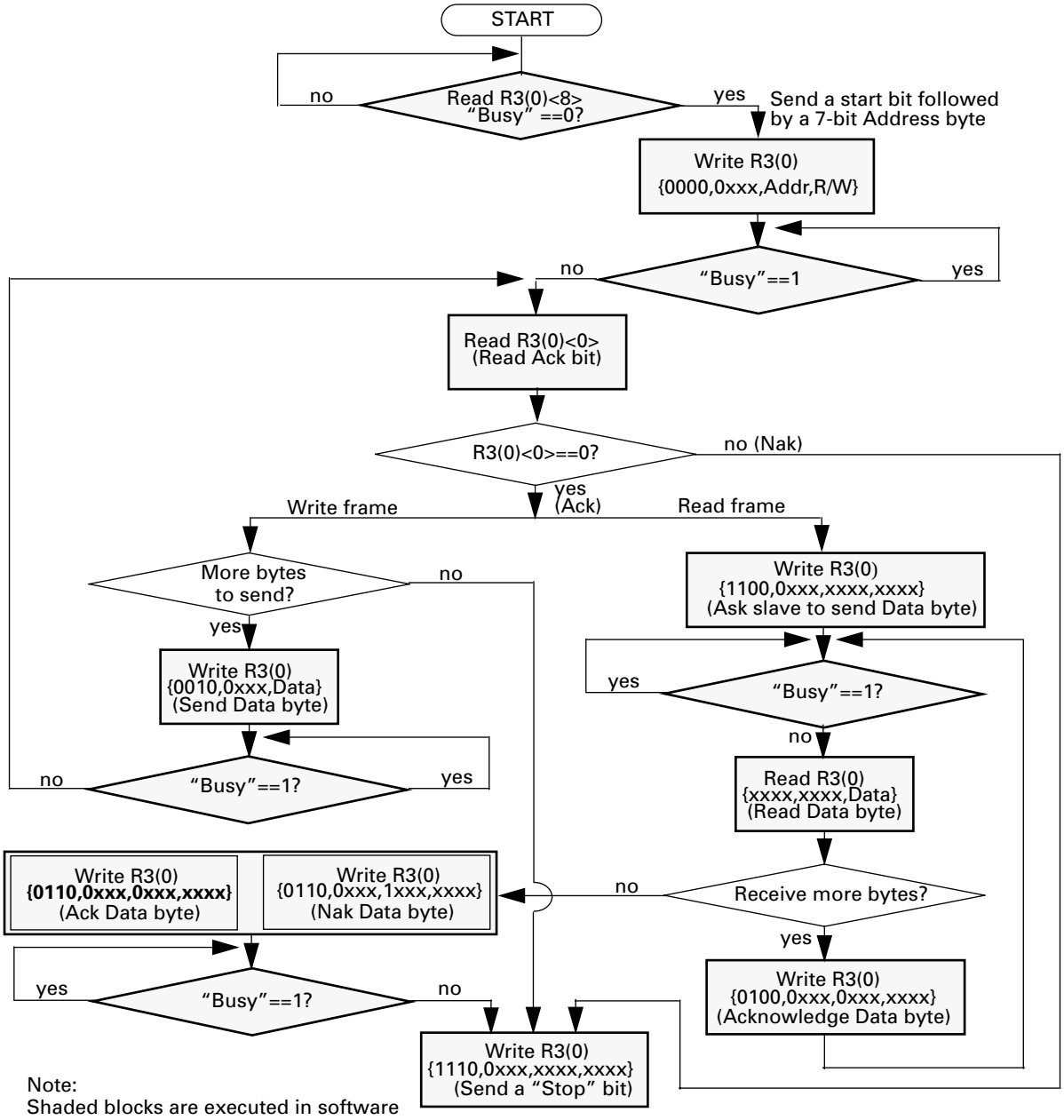
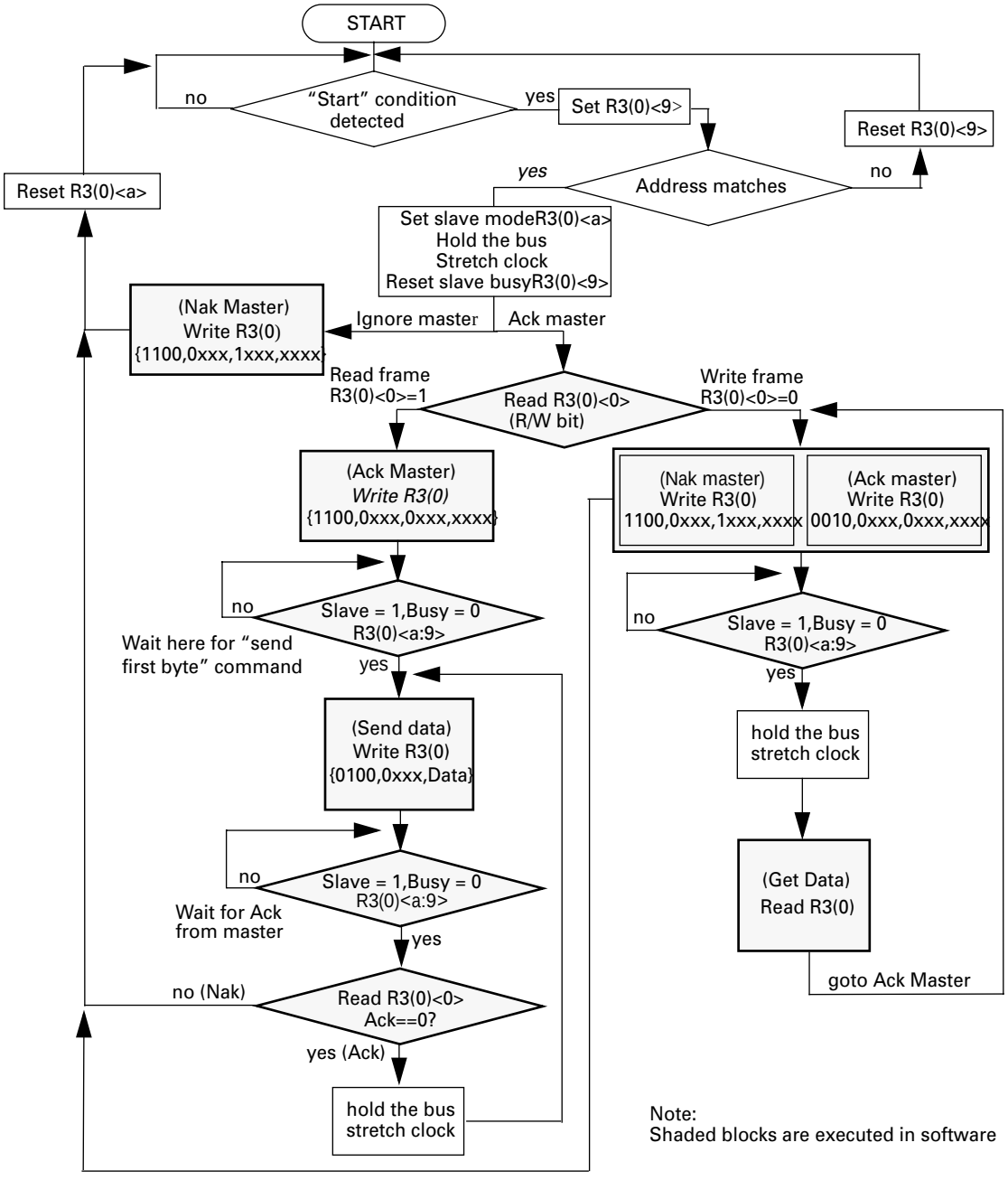


Figure 15 Master Mode



Note: If a "Stop" condition is detected at any point, the hardware resets the "Slave" bit (R3(0)<a>, and releases the I^2C bus.

Figure 16 Slave Mode

2.12 On-Screen Display (OSD)

The Z90365 provides sophisticated on-screen display features. On-Screen Display has two modes.

- OSD Used to generate TV control OSD
- CCD Used to display Closed Caption information

OSD mode provides access to the full set of control attributes including latched and unlatched attributes. Unlatched attributes can be modified on a character-by-character basis. Control characters change latched attributes.

The full 512-character set, formatted in two 256 character banks, can be displayed with many display attributes, including underlining, italics and blinking, eight foreground and background colors, character position offset delay, and background transparency. A 16-bit display character represents foreground color, background color, and underline attributes which can be modified character by character. A character's pixel matrx is stored as 16, 18 or 20 words in Character Generation ROM (CGROM).

Additional hardware provides the capability to display characters at two and three times normal size. The smoothing logic improves the appearance of two and three times normal size characters. Fringing can be activated to improve the visibility of characters by adding a border (one pixel wide) on each side.

The Z90365 provides RGB signals and a video blank signal. RGB signals are available in two modes: digital and analog. In digital mode the output RGB signals correspond to a primary colors palette. Analog mode supports 64 different palettes which can be chosen under software control. In analog mode, each RGB signal is generated by a 2-bit digital to analog converter. The 2-bit digital inputs of the D to A converter can be switched to Port pins (P10, P13, P14, P15, P18 and P08) by setting bit9 in Register3(1) to "1".

Video synchronization is normally obtained from H_FLYBACK and V_FLYBACK, but can be generated by the Z90365 and driven to the external deflection unit via the bidirectional SYNC ports when external video synchronization signals are not present.

OSD is completely software controlled. Hardware supports the optimum generation of the character based OSD, however the CPU can bypass it and generate pixels and attributes directly. The block diagram in Figure 19 shows the OSD data flow.

Figures 17 through 19 show the V1, V2, V3 and Blank output circuits.

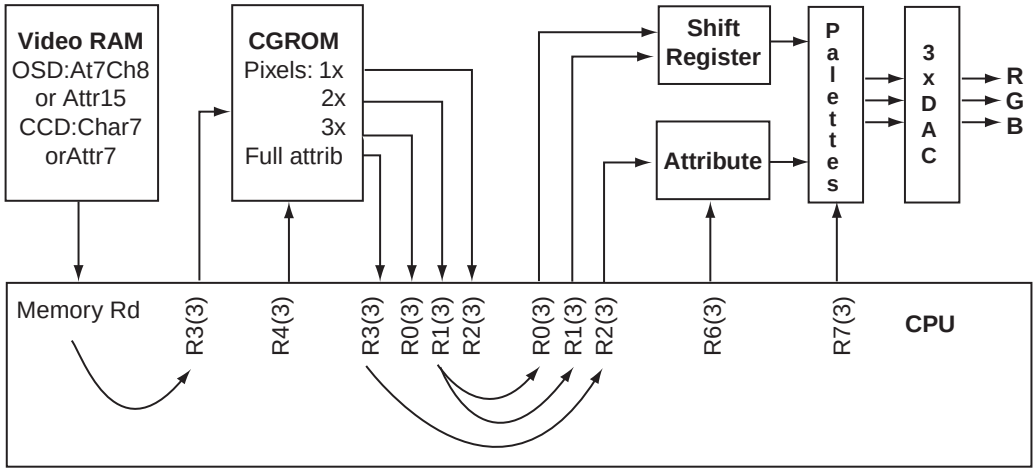


Figure 17 OSD Data Flow

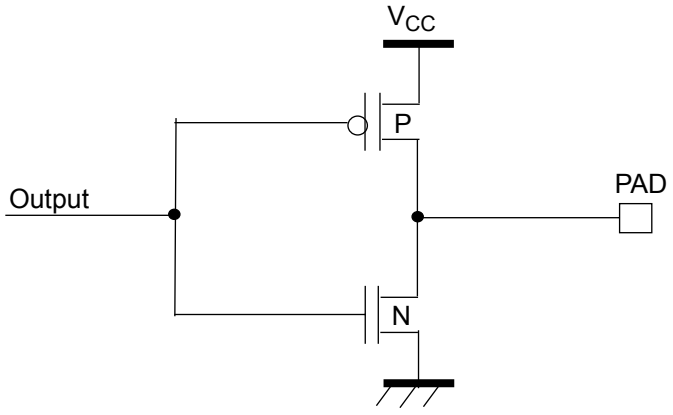


Figure 18 Blank and V1, V2, V3 Outputs in Digital Mode

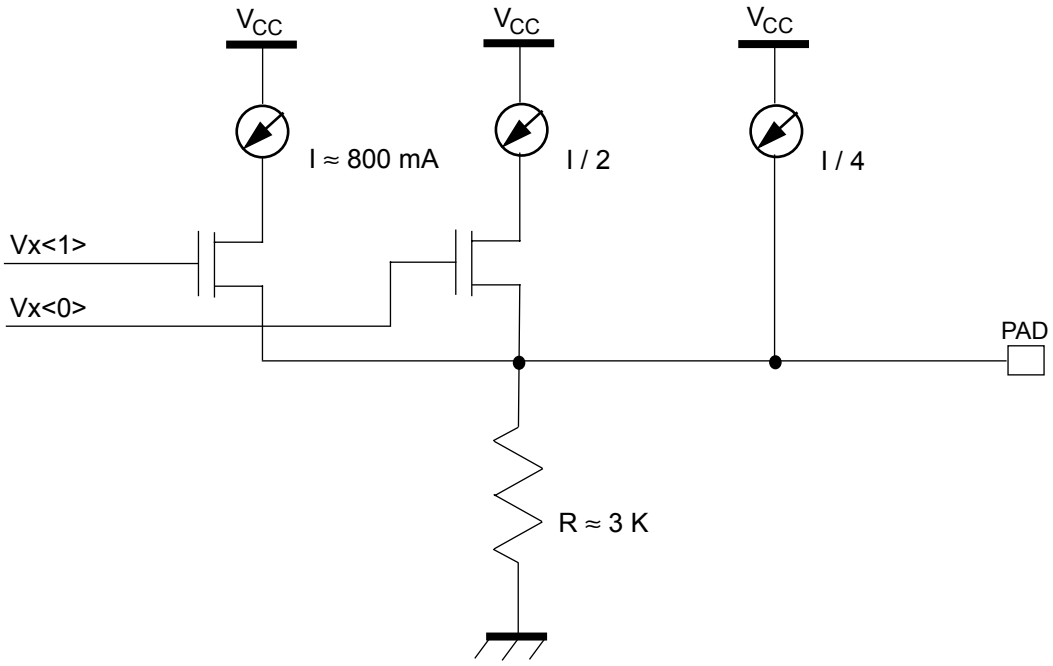


Figure 19 V1, V2 and V3 Outputs in Analog (Palette) Mode

Closed Caption Data Capture

Closed-caption text can be decoded directly from the composite video signal using the processor's digital signal processing capabilities and displayed on the screen. The character representation in this mode provides simple attribute control by inserting control characters. Each word of video RAM specifies two displayed characters.

The 4-bit flash A/D converter, with proper clamping, provides the ability to receive the composite video signal directly and process the closed-caption text embedded in the signal. Signal processing can be applied directly to the signal to improve decoder performance.

CGROM (Character Generation ROM)

The required Character Generation ROM size is dependent on the number of characters that are stored in memory. CGROM always starts at address 0000h. CGROM can be configured as two banks selectable by setting a control bit. Each bank provides up to 256 characters with 16x16, 16x18 or 16x20 pixels matrix. Absolute maximum CGROM size is 9K words (9216 words - 256x16 + 256x20). If both banks are used, the second bank starts from address 1000 hex (refer back to Figure 4).

Each character pixel matrix is 16, 18 or 20 words of ROM. Each word represents 16 pixels. Scan lines 1 to 16 are mapped sequentially to ROM addresses, referenced by the character pointer and a line number offset.

In 16x18 mode, scan lines 17 and 18 are offset by 1000h, referenced to scan line1. In 16 x 20 mode, scan lines 19 and 20 are offset by 1200 hex. That means that if Bank0 is in a 16 x 18 pixel matrix, it overlaps the first 32 characters of Bank1. If Bank0 is a 16 x 20 pixel matrix, it overlaps the first 64 characters of Bank1. The first character in each bank must be a space character. If an application requires Bank0 characters to be in 16 x 18, or 16 x 20, pixel matrix, the first 32 (64 for 16x20) characters of Bank1 must be sacrificed and cannot be used because lines 17 and 18 (for 16 x 18), and lines 17 through 20 (for 16 x 20) of Bank0 characters are mapped in their addresses. The first 8 (16 for 16 x 20) characters of Bank0 cannot have active pixels in lines 17 and 18 (19 and 20 for 16 x 20) to have a blank first (space) character in Bank1.

If only Bank0 of CGROM is used, there is no limitation on character size.

The character scan line from CGROM addressed by the character register is fetched and stored into the CGROM capture register. If a pixel is set to `1`, it displays foreground color. If set to `0`, it displays background color. The scan line can be stretched by the character multiplier to be two or three times normal character size by duplicating each bit in the word.

Controlling Character Expansion

The character size can be stretched to two or three times its size. Hardware fetches data from CGROM and stretches the data to be read from registers R0(3), R1(3), and R2(3). Figure 20 is a block diagram of the structure of the character expansion multiplexor and Table 10 lists bit functions.

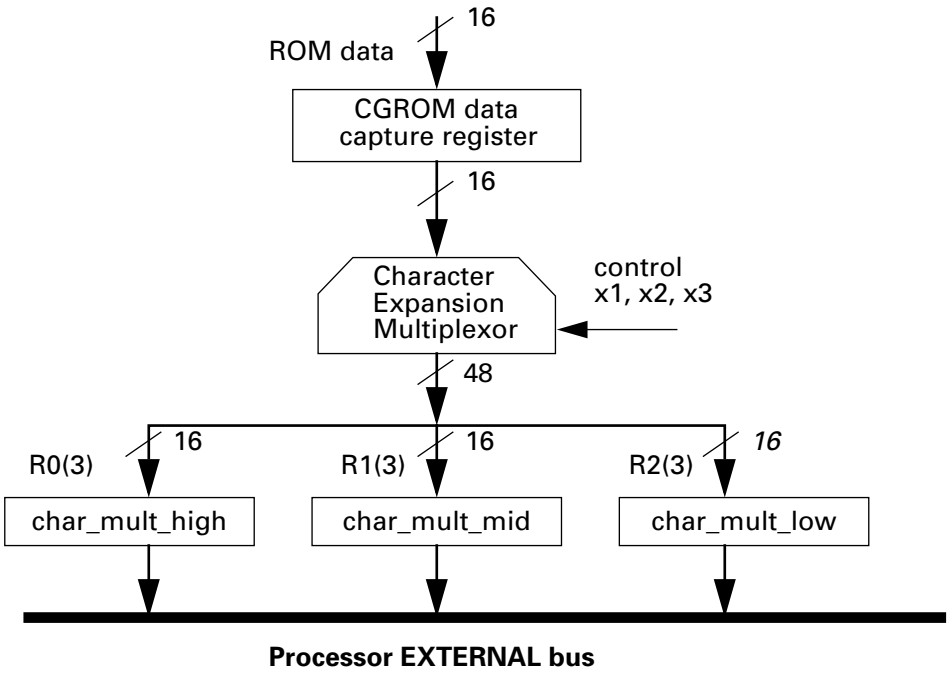


Figure 20 Character Expansion

Table 10 Character Expansion Register

Capture Register Contents	Char_mult_high	Char_mult_mid	Char_mult_low
x1 operation	abcdefghijklmnop		
x2 operation	aabbccddeeffgghh	iijjkkllmmnnoopp	
x3 operation	aaabbbcccdddeeef	ffgggghhhiiijjjkk	klllmmmnnnoooppp

Displayed Data Formats

The Z90365 hardware supports two different data formats.

- **OSD mode, R4(3)<d> = 1** supports a standard OSD with full set of features.
- **CCD mode, R4(3)<d> = 0** supports reduced features which comply with the recommendations of the FCC on Closed Caption support.

In CCD mode, the background color of the characters can NOT be changed and is always preset to BLACK.

OSD Mode

In OSD mode, each character occupies a 16-bit word in VRAM. There are two possible character formats defined: a “display” character and a “control” character. The code stored in “display” character format defines a character code and up to 7 attributes of the character.

The “control” character defines latched attributes and is presented on-screen as a space character. The combination of “display” and “control” characters provides versatile OSD generation.

Smoothing is supported for double-size (x2) and triple-size (x3) characters only.

CCD Mode

In CCD mode, each character occupies 8 bits (one byte) in VRAM. The CCD characters must be mapped into a 16-bit VRAM data field. The hardware supports compressed placement of characters in VRAM. Each word in VRAM is represented by a High byte and a Low byte. A currently active byte is selected by R4(3)<c>. The format and data representation in both bytes is exactly the same.

There are two possible character formats defined: a “display” character and a “control” character. The code stored in “display” character format defines a character code. The “control” character defines five latched attributes (foreground color, italic, underline, blinking, and transparent); it is presented on screen as a space character. The combination of Display and Control characters provides the basis for a specified range of attributes defined by FCC specifications for CCD.

2.13 Other Functions

Video and Sound Attribute Control

Basic receiver functions such as color and volume can be controlled directly by six 6-bit pulse-width modulated ports.

InfraRed Capture Function

The Infrared Remote Control data capture feature uses a capture register to hold the time value from one transition of IR data to the next.

Software periodically checks and reads the capture status and the value if a new capture occurs. Subsequent decoding and command passing of the received IR signal is under software control. Figure 21 illustrates the IR input circuit.

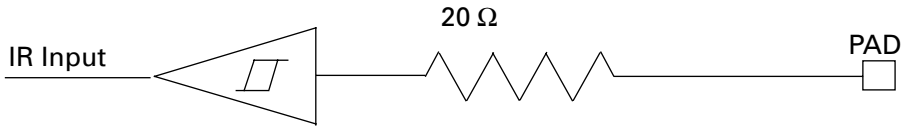


Figure 21 IR Capture Register Input

Loop Filter

The Loop Filter pin configuration is represented in Figure 22.

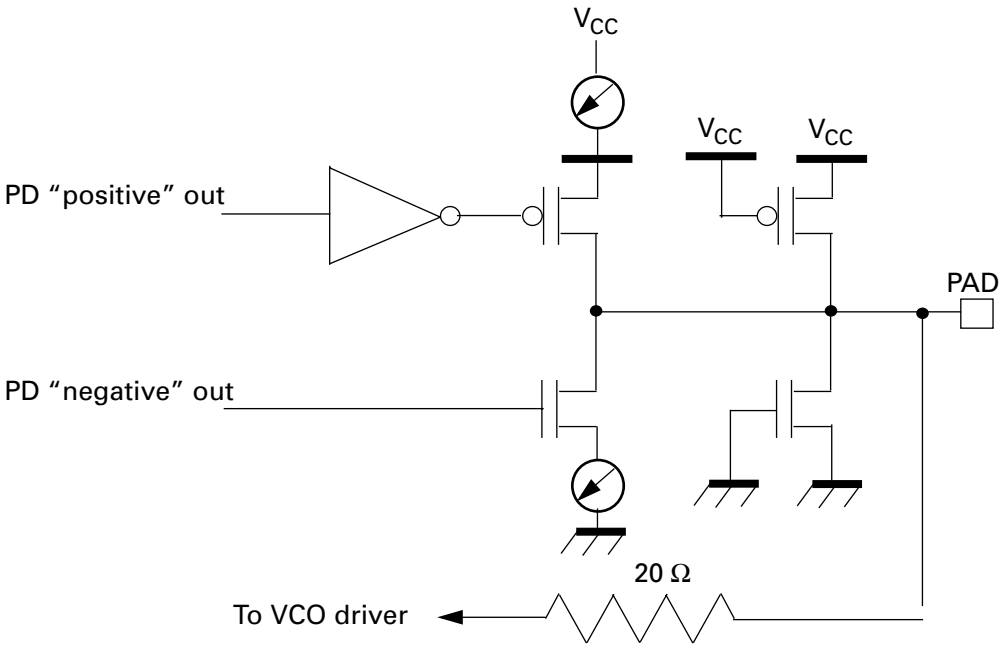


Figure 22 Loop Filter Pin Configuration

3 REGISTER GROUPS

Table 11 provides a summary of the registers in external banks.

Table 11 Register Summary

BANK	BANK Sub Address	READ Register	WRITE Register	Description
Bank0	7	dir1		9-bit I/O port 1 direction control
	6	dir0		11-bit I/O port 0 direction control
	5	port1		9-bit I/O port 1
	4	port0		11-bit I/O port 0
	3	I2C_int		I^2C interface register
	2	pll_freq		PLL frequency control
	1	PWM_data10		14-Bit PWM 10 data
	0	PWM_data9		14-Bit PWM 9 data
Bank1	7	wdt_smr_ctl/Interrupt		SMR and WDT control and interrupt
	6	clock_ctl		Clock control (switch VCO/DOT)
	5	cap_1s_ctl		Counter timers control
	4	ADC_ctl		A/D converter control
	3	standard_ctl		Output H/VSYNC/blnk control
	2	9-bit counter	STOP/WDT	Stop and WDT instructions, 9-bit counter
	1	sclk_freq		Stop/sleep/normal mode
	0	clamp_pos		Defines position of video clamp pulse
Bank2	7	Reserved		Reserved
	6	Reserved		Reserved
	5	Reserved		Reserved
	4	pwm_data5		8-bit PWM 5 data
	3	pwm_data4		8-bit PWM 4 data
	2	pwm_data3		8-bit PWM 3 data
	1	pwm_data2		8-bit PWM 2 data
	0	pwm_data1		8-bit PWM 1 data

Table 11 Register Summary (Continued)

BANK	BANK Sub Address	READ Register	WRITE Register	Description
Bank3	7	output palette		Output palette
	6	palette_color		Display palette color/underline color
	5	capture_data	I^2C slave addr.	Capture register data
	4	osd_control		On screen display control
	3	attribute_data	vram_data	Character attribute/video RAM data
	2	ch_x1_lo_x3	cg_attribute	Character multiple/character graphics attribute
	1	lo_x2_mid_x3	cg_nxt_prv	Character multiple/next or previous data
	0	hi_x2_hi_x3	cg_current	Character multiple/current data

3.1 Register Description

The register file in the Z90365 is organized into four banks which can be selected by writing to bits 5 and 6 (Register Bank Selector bits) in the Status Register of the Z90365 core.

All registers are mapped into an external register space; each bank consists of 8 registers. The Status register is available to read or write at any time. The appropriate bank of registers must be selected before accessing the register. The software must keep track of which register bank is accessible at any time. Refer to Table 12 for register bank assignments.

Table 12 Bank Assignments

Bank	Status Register	Bank Functions
Bank0	xxxx xxxx x00x xxxx b	I/O ports, I^2C interface, PLL frequency, 14-bit PWM
Bank1	xxxx xxxx x01x xxxx b	Control registers
Bank2	xxxx xxxx x10x xxxx b	PWM1–PWM5
Bank3	xxxx xxxx x11x xxxx b	OSD, palette control

3.2 Bank0 (I/O Ports, I²C Interface, PLL Frequency, PWM)

Table 13 defines the bits for Register0, R0(0) PWM9 Data Register.

Table 13 Register0, R0(0) PWM9 Data Register

Bit	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	x	x	x	x	x	x	x	x
Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	x	x	x	x	x	x	x	x

Note: R = Read W = Write X = Indeterminate

Reg Field	**Bit Position**	**R**	**W**	**Value**	**Description**
Port/PWM	`f---------------`	R	W	1 0	Output port mode Output PWM mode (push-pull)
Port_data	`-e--------------`	R	W	x	Output data in Port mode
PWM_data	`--dcba9876543210`	R	W	xxx	Output data in PWM mode

The Port/PWM bit defines the mode of the PWM9 output. When set to `1`, the PWM9 output monitors the data specified by the Port_data field. Otherwise the PWM_data field defines the waveform on the PWM9 pin.

Table 14 defines the bits for Register1 - R1(0) PWM10 Data Register.
Table 15 defines the bits for Register2 - R2(0) PLL Frequency Data Register.

Table 14 Register1, R1(0) PWM10 Data Register

Bit	15	14	13	12	11	10	9	8
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	x	x	x	x	x	x	x	x
Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	x	x	x	x	x	x	x	x

Note: R = Read W = Write X = Indeterminate

Reg Field	Bit Position	R	W	Data	Description
Port/PWM	f---------------	R	W	1 0	Output port mode Output PWM mode (push-pull)
Port_data	-e--------------	R	W	x	Output data in Port mode
PWM_data	--dcba9876543210	R	W	xxx	Output data in PWM mode

Table 15 Register2, R2(0) PLL Frequency Data Register

Bit	15	14	13	12	11	10	9	8
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0
Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	1	1	1	0	0	0	0

Note: R = Read W = Write X = Indeterminate

Reg Field	Bit Position	R	W	Data	Description
M_disable	f---------------	R	W	1 0	I^2C Master interface disabled I^2C Master interface enabled–POR
S_disable	-e--------------	R	W	1 0	I^2C Slave interface disabled I^2C Slave interface enabled–POR
Reserved	--dcba98--------	R W			Return 0 No Effect
Data	--------76543210	R	W	xx	PLL divider =256 + xx

If the master or slave I^2C interface is enabled, the corresponding I/Os (Port01 and Port02 for the slave, Port11 and Port12 for the master) must be assigned as outputs.

The VCO, DOT, and SCLK frequency are defined as:

$$F_{VCO} = F_{DOT} = F_{SCLK} = XTAL * (256 + PLL_{DATA})$$

Therefore, $XTAL = 32.768\ KHz$

At POR the PLL frequency data register is preset to %70, which corresponds to the VCO frequency of 12.058 MHz.

The PLL_data field can be loaded with any value from %00. This value corresponds to an SCLK = 256*XTAL up to %FF, which corresponds to an SCLK = 511*XTAL.

Because the SCLK frequency is proportional to the frequency of the XTAL, it is impossible to specify the maximum value that can be written into the PLL_data field.

Note: It is the **customer's responsibility not to exceed the SCLK frequency of 12.5 MHz**.

For common applications incorporating a 32.768KHz XTAL oscillator, the maximum setting of the PLL_data field in R2(0) is %7D, which corresponds to SCLK = 381*XTAL = 12.485 MHz.

Tables 16 through 20 describe the bits in registers 3 through 7.

Table 16 Register3, R3(0) I²C Interface Register

Bit	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	1	x	x
Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	x	x	x	x	x	x	x	x

Note: R = Read W = Write X = Indeterminate

Reg Field	**Bit Position**	**R**	**W**	**Data**	**Description**
Command	fed----------s	R			Return 0
			W	%D	See full description in Tables 8 and 9.
Toggle	---c----------	R		1	Slave interface
				0	Master interface–POR condition
			W	1	Toggle active I²C interface
				0	No effect
Reset	----b----------	R			Return 0
			W	1	Reset Slave I²C interface if bit <c> = 1 Reset Master I²C interface if bit <c> = 0
				0	No effect
Slave_mode	-----a----------	R		1	Slave mode is active (POR condition)
				0	Slave mode is inactive
			W		No effect
SlaveBusy	------9---------	R		1	Slave I²C interface is busy
				0	Slave I²C interface is idle
			W		No effect
MasterBusy	-------8--------	R		1	Master I²C interface is busy
				0	Master I²C interface is idle
			W		No effect
Data	--------76543210	R		xx	Received data
			W	xx	Data to be sent

Data written to R3(0)<cb> requires 4 cycles before being applied.

Consecutive writings to these bits require at least a 6-cycle delay.

Received data is available for reading only when the busy bit is reset to a 0. At POR, the speed of the I²C interface is set to Low controlled by R(3)1<8>.

Table 17 Register4, R4(0) Port 0 Data Register

Bit	15	14	13	12	11	10	9	8
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	x	x	x	x	x	x	x	x
Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	x	x	x	x	x	x	x	x

Note: R = Read W = Write X = Indeterminate

Reg Field	Bit Position	R	W	Data	Description
Port_data	f-----9876543210	R		xxxx	If a port is configured in Input mode, enter the input data onto the port pins.
			W	xxxx	If a port is configured in Output mode, then the data is written directly to the port data.

Table 18 Register5, R5(0) Port 1 Data Register

Bit	15	14	13	12	11	10	9	8
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	x	x	x	x	x	x	x	x
Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	x	x	x	x	x	x	x	x

Note: R = Read W = Write X = Indeterminate

Reg Field	Bit Position	R	W	Data	Description
Reserved	fedcba9---------	R			Return 0
			W		No Effect
Port_data	-------876543210	R		xxxx	If a port is configured in Input mode, enter the input data onto the port pins.
			W	xxxx	If a port is configured in Output mode, then the data is written directly to the port data.

Table 19 Register6, R6(0) Port 0 Direction Register

Bit	15	14	13	12	11	10	9	8
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	x	x	x	x	x	1	1	1
Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	1	1	1	1	1	1	1	1

Note: R = Read W = Write X = Indeterminate

Reg Field	Bit Position	R	W	Data	Description
Port_direction	fedcba9876543210	R	W	xxxx	1: Input mode for corresponding bit 0: Output mode for corresponding bit

Table 20 Register7, R7(0) Port 1 Direction Register

Bit	15	14	13	12	11	10	9	8
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	x	x	x	x	x	x	x	1
Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	1	1	1	1	1	1	1	1

Note: R = Read W = Write X = Indeterminate

Reg Field	Bit Position	R	W	Data	Description
Reserved	fedcba9---------	R			Return 0
			W		No Effect
Port_direction	-------876543210	R		xxxxx	1: Input mode for corresponding bit
			W		0: Output mode for corresponding bit

3.3 Bank1 (Control Registers)

Tables 21 through 27 provide bit functions for Bank 1 Control registers.

Table 21 Register0, R0(1) Clamp Position Register

Bit	15	14	13	12	11	10	9	8
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	1	0	0	0	0	0	0	0
Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	x	x	x	x	x	x	x

Note: R = Read W = Write X = Indeterminate

Reg Field	Bit Position	R	W	Data	Description
Disable_clamp_1	f---------------	R	W	1	ADC0 Clamp generation is disabled
				0	ADC0 Clamp generation is enabled
Reserved	-edcba987-------	R			Return “0”
			W		No Effect
Position	---------6543210	R	W	xx	Position of clamp pulse (from leading edge of the H-FLYBACK)

At POR the disable_clamp bit is set to 1.

The clamp pulse is generated if Enabled (bit <f>) and the SCLK frequency are switched back to PVCO. The SVCO/PVCO flag in R6(1) must be reset to 0 before the current HSYNC, regardless of whether the SVCO is enabled or disabled.

The clamp position is defined by the Position field. The width of the clamp pulse cannot be modified and is set to 1us. The value that can be assigned to the “Position” field must be >%10 and <%7F. The time interval between the leading edge of the H-FLYBACK and the beginning of the clamp pulse can be calculated from the following equation:

$$T_{DELAY} = Position\left(\frac{1}{TSCLK}\right) = Position \times 82ns$$

Table 22 Register1, R1(1) Speed Control Register

Bit	15	14	13	12	11	10	9	8
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0
Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Note: R = Read W = Write X = Indeterminate

Reg Field	Bit Position	R	W	Data	Description
Reserved	fedcba98765432--				Reserved (Return 0)
Fast_enable	--------------1-	R	W	1 0	PVCO/SVCO enabled PVCO/SVCO disabled–POR
Fast_slow	---------------0	R	W	1 0	SCLK is 12.058 MHz SCLK is 32.768 KHz–POR

When a POR, SMR, or WDT reset occurs, both the Fast_enable and Fast/Slow are reset to 0. This event corresponds to an SCLK frequency of 32.768 kHz.

To switch from a 32.768-kHz SCLK to 12 MHz, use the following procedure.

1. Set the H_Position field R6(1)<3:0> to a nonzero value.
2. Enable the primary and secondary VCOs (set the Fast_enable bit R1(1)<1> to “1”).
3. Wait for one second (1s) for the 12-MHz PLL to stabilize (about 50000 clock cycles). The delay depends on the external PLL filter and CAN vary significantly.
4. Switch the SCLK to a fast clock (set Fast/Slow bit R1(1)<0> to 1).
5. Simultaneously set the H_Position field R6(1)<3:0> to 0FH and the No_Switch field R6(1)<4> to 1 (no clock switch).

To switch from the 12-MHz SCLK to 32.768 kHz, use the following procedure.

1. Switch the SCLK to a 32.768-kHz clock (set Fast/Slow bit R1(1)<0> to 0).
2. Wait for more than R2(0)<7:0> + 256 clock cycles (approximately 32 µS) for the SCLK to be switched.
3. Set the HSYNC_DELAY field R6(1)<3:0> to 0FH.
4. Disable the primary and secondary VCOs (set the Fast_enable bit R1(1)<1> to 0).

Table 23 Register2, R2(1) WDT/STOP (write only) and 9-bit Counter (read only) Control Register

Bit	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	x	x	x	x	x	x	x	x
Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	x	0	0	0	0	0	0	0

Note: R = Read W = Write X = Indeterminate

Reg Field	**Bit Position**	**R**	**W**	**Data**	**Description**
Counter_value	fedcba987-------	R			Counter on Port06 value
			W		No effect
Reserved	---------65432--	R			Return 0
			W		No effect
WDT_instr	--------------1-	R		1	Return 0
				0	WDT enable, WDT reset
			W		No effect
STOP_instr	---------------0	R			Return 0
			W	1	Stop
				0	No effect

When a POR, SMR or a WDT reset occurs, the WDT is disabled. The WDT can be reenabled only after the PVCO and SVCO are enabled, and the part is switched into a Fast mode (SCLK = 12 MHz).

When switching the part into a SLOW mode (SCLK = 32.768 kHz), the WDT halts. To return to Fast mode, the WDT must be initialized again.

Table 24 Register3, R3(1) Standard Control Register

Bit	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0
Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	x	x	x	0	0

Note: R = Read W = Write X = Indeterminate

Reg Field	**Bit Position**	**R**	**W**	**Data**	**Description**
Counter_ reset	f---------------	R		1 0	Return 0 Reset Counter on Port06
			W		No effect
Counter_ON/OFF	-e--------------	R	W	1 0	Counter on Port06 is ON Counter is OFF–POR condition
Mask_HVSYNC	--d-------------	R	W	1 0	Disable HVSYNC output HVSYNC IN/OUT–POR condition
Char_size_16_18/20	---c------------	R	W	1 0	16x20 character matrix 16x16 or 16x18 character matrix–POR
Bank0_128/Bank0_256	----b-----------	R	W	1 0	Extended RAM–128 words Basic Bank–256 words–POR condition
P07/ComSYNC	-----a----------	R	W	1 0	Composite SYNC output P07 I/O–POR condition
RGBC/Port1	------9---------	R	W	1 0	SCLK, R<1:0>, G<1:0>, B<1:0> P16,P08,P10,P13,P18,P15,P14
I^2C_HI/LO_speed	-------8--------	R	W	1 0	HI speed (400 kHz) LO speed (100 kHz)–POR condition
CGROM bank	--------7-------	R	W	1 0	Bank1 is selected (starts @%1000) Bank0 is selected (starts @%0000)
Reserved	---------6------	R		0	Reserved
			W		No effect
OSD_on/off	----------5-----	R	W	1 0	OSD is enabled OSD is disabled–POR

Reg Field	Bit Position	R	W	Data	Description
RGB_polarity	----------4----	R	W	1 0	Negative Positive
Positive/Negative	----------3---	R	W	1 0	Negative HVSYNC in output mode Positive HVSYNC in output mode
SYNC/BLANK	----------2--	R	W	1 0	HVBLANK outputs HVSYNC outputs
25/30_Hz and HV_polarity	----------10	R	W	 10 00 11 01	Internal mode *only* (TV Standard) 50 Hz/625 lines support 60 Hz/525 lines support–POR External mode *only* (HV Polarity) Positive Negative

Two bits define the polarity of the HVSYNC signals. Bit <3> defines polarity of the signals when they are configured as outputs (it does not affect the internal HV–SYNC signals). Bit <1> defines the polarity of the external HV–SYNC signals, affecting the synchronization of the device.

Notes:

1. The composite SYNC is active in internal mode only.
2. When using the internally-generated COMPOSITE SYNC signal, be sure the SCLK is set to 12.09MHz (R2(0)<7:0> = %71). This action helps ensure the best HSYNC frequency approximation.

Table 25 Register 4, R4(1) ADC Control Register

Bit	15	14	13	12	11	10	9	8
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0
Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	x	x	x	x

Note: R = Read W = Write X = Indeterminate

Reg Field	Bit Position	R	W	Data	Description
Reserved	fedcb----------	R		0	Return 0
			W		No Effect
ADC_select	-----a----------	R	W	1	ADC4 select
				0	ADC0, ADC1, ADC2, ADC3 select–POR condition
Reserved	------98--------	R			Return “0”
			W		No effect
ADCspeed	--------76------	R	W	00	Single conversion–POR condition
				01	SCLK/4
				10	SCLK/6
				11	SCLK/8
ADCsource	----------54----	R	W	00	ADC0 (CVI)/ADC4 (P04)–POR
				01	ADC1 (P17)
				10	ADC2 (P00)
				11	ADC3 (P05)
ADCdata	------------3210	R		%D	ADC data
			W		No effect

ADC0 has a signal range from 1.5 to 2.0 V. This field is always connected to the Composite Video Input pin and can be clamped to a Ref– voltage (1.5V).

ADC1, ADC2, ADC3, and ADC4 have a signal range from 0 to 5.0V.

To use the I/O pin as an ADC input, the corresponding port must be set up as an input (refer to R4(0) and R6(0)).

Table 26 Register5, R5(1)Timer Control Register

Bit	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	x	x	x	x	0	0	x	x
Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	x	x	x	x	x	x	x	x

Note: R = Read W = Write X = Indeterminate

Reg Field	Bit Position	R	W	Data	Description
CAPint_r	f---------------	R		1	Rising edge is captured
				0	No rising edge is captured
			W	1	Reset flag
				0	No effect
CAPint_f	-e--------------	R		1	Falling edge is captured
				0	No falling edge is captured
			W	1	Reset flag
				0	No effect
Tout_1s	--d-------------	R		1	Timeout of 1s timer
				0	No timeout of 1s timer
			W	1	Reset flag
				0	No effect
Tout_CAP	---c------------	R		1	Timeout of Capture timer
				0	No timeout of Capture timer
			W	1	Reset flag
				0	No effect
Reserved	----ba----------	R			Return “0”
			W		No effect
Speed_1s	------98--------	R	W	00	1s
				01	250 ms
				10	62.5 ms
				11	15.625 ms
Port09/ CAP_int*	--------7-------	R	W	1	int2 source is Port09
				0	int2 source is Capture timer
CAP_halt*	---------6------	R	W	1	Capture timer is halted
				0	Capture timer is running
CAP_edge*	----------54----	R	W	00	No Capture
				01	Capture on rising edge only
				10	Capture on falling edge only
				11	Capture on both edges
CAP_glitch*	------------32--	R	W	00	Glitch filter is disabled
				01	<8TSCLK is filtered out
				10	<32TSCLK is filtered out
				11	<128TSCLK is filtered out
CAP_speed*	--------------10	R	W	00	SCLK/4
				01	SCLK/8
				10	SCLK/16
				11	SCLK/32

*Resetting a Capture Timer flag does not modify Capture Counter and/or Capture register data. When the glitch filter is enabled, the duration of the pulse is decreased by CAP_glitch value.

Table 27 Register6, R6(1) Clock Switch Control Register

Bit	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0
Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	1	x	x	x	x

Note: R = Read W = Write X = Indeterminate

Reg Field	Bit Position	R	W	Data	Description
Reserved	fedcba9876------	R			Return 0
			W		No effect
SVCO/PVCO	----------5-----	R		1	SCLK = SVCO (flag)
				0	SCLK = PVCO (flag) POR
			W	1	Switch SCLK to PVCO
				0	No effect
No_Switch	----------4----	R	W	1	SCLK = PVCO, no clock switching—POR
				0	Clock switching is enabled
H_Position	------------3210	R	W	%D	Defines delay of H_{SYNC} interrupt by 4X SCLK cycles

H_Position

The H_position field defines the delay between when the Z90365 receives an H-sync and when the H-sync interrupt is executed. Because the On-Screen Display is controlled by software this delay allows fine tuning for On-Screen Display centering. At Power On Reset (POR) this value is in an unknown state. If this value is set to `0x0`, neither the H_{SYNC} interrupt nor clock switching can execute. To receive an H_{SYNC} interrupt, valid delay values are between `0x1` and `0xF`. Valid delay values produce a delay according to the following equation:

```
11 + 4 * (H_position - 1)
```

For example, an H_position setting of 0xF produces a delay of 67 system clock cycles after the trailing edge of H_{SYNC}.

```
11 + 4 * (15 - 1) = 67
```

No_Switch

The No_switch bit determines if the system clock is permanently set to the Primary VCO (PVCO) or allowed to switch between PVCO and the Secondary VCO (SVCO). This bit is set to 1 (NO clock switching) at power up reset.

Caution: After the system has been switched to fast (12 MHz) clock both signals feeding into this switch MUST be PVCO BEFORE the switch setting is changed. Otherwise a short system clock can result which causes the processor to run at a higher frequency than specified. The instruction fetched from memory, at the location with the out-of-spec frequency, can be corrupted!

To ensure safe clocks, the following practices are recommended:

1. Set the No_switch to the required setting before switching from the 32.768 kHz to the fast (12 MHz) clock and leave it there permanently.
2. Use the following procedure when changing from Switching VCO to permanent PVCO while running from the fast clock:
   - Simultaneously set the H_position delay to 0x0, while leaving the No_switch enabled (0), and the SVCO/PVCO left as (0).
   - Wait a minimum of 80 clock cycles to flush any H-sync out of the system.
   - Simultaneously Switch SVCO/PVCO to PVCO (write 1 into R6(1)<5>), while leaving the No_switch enabled (0), and the H_position delay at 0x0.
   - Wait for 3 system clock cycles to be sure that the clock has had time to switch to PVCO.
   - Switch No_switch to No clock switch (write 1 into R6(1)<6>). The H_position can be set to none-zero at this time as well.
3. Use the following procedure when changing from permanent PVCO to Switching VCO while running from the fast clock.
   - Simultaneously set the H_position delay to 0x0, while leaving the No_switch disabled (1), and the SVCO/PVCO setting as don’t care (0).
   - Wait a minimum of 80 clock cycles to flush residual H-sync out of the system.
   - Simultaneously switch SVCO/PVCO to PVCO (write 1 into R6(1)<5>), while leaving the No_switch disabled (1), and the H_position delay at 0x0.
   - Wait 3 system clock cycles to be sure that the clock has had time to switch to PVCO.
   - Switch No_switch to Clock switching is Enabled (write 0 into R6(1)<6>). The H_position can be set to none-zero at this time as well.

SVCO/PVCO

The SVCO/PVCO bit when read back determines the current setting of the system clock. Writing a 1 to this bit switches the system clock from its current setting to PVCO. This switch has a glitch filter that removes random voltage spikes. It must be changed back to PVCO. The Z90365 switches to the SVCO automatically when it receives an H-sync interrupt. This mechanism exists to synchronize the system clock exactly with the H-sync trailing edge. The result is a sharp start of the OSD, jitter free.

An example of a typical SVCO/PVCO switching follows:

1. System clock is set to PVCO.
2. H-sync interrupt occurs. (The system clock has automatically been set to the SVCO). OSD code is executed inside of the H-sync Interrupt Service Routine (ISR).
3. Before leaving the ISR, the user switches the clock back to PVCO.

The clamp pulse (defined in R0(1)) is generated only if the SVCO/PVCO switch is set to PVCO before receiving an H_{SYNC}. The software provides the correct switch setting before every H_{SYNC}.

Table 28 lists the interrupt/WDT for the WST/SMR control register.

Table 28 Register7, R7(1) Interrupts/WDT/SMR Control Register

Bit	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	x	x	x	x	x	x	x	x
Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	x	x	x	x	x	x	x

Note: R = Read W = Write X = Indeterminate

Reg Field	**Bit Position**	**R**	**W**	**Data**	**Description**
Int_priority	fed----------	R	W	x	See Table 29
Int_mask	---cba----------	R	W	1xx 0xx x1x x0x xx1 xx0	int2 is enabled int2 is disabled int1 is enabled int1 is disabled int0 is enabled int0 is disabled

Reg Field	Bit Position	R	W	Data	Description
WDTspeed	------98--------	R	W	00 01 10 11	1.83 ms 7.68 ms 31.12 ms 124.8 ms
SMRflag	--------7-------	R		0 1	No Stop-Mode Recovery–POR Stop-Mode Recovery
			W		No effect
SMR polarity	---------6------	R	W	0 1	OR of all SMR sources NAND of all SMR sources
SMRsource	----------543210	R	W	xx	Bit which corresponds to a “1” in xx binary representation is active
smr5	----------5-----				p09
smr4	-----------4----				p14
smr3	------------3---				p13
smr2	-------------2--				p12
smr1	--------------1-				p11
smr0	---------------0				p10

The final result of the Stop-Mode Recovery (SMR) is RESET. Ports selected for SMR must be assigned as inputs, while the other SMR ports must be assigned as outputs exhibiting a nonactive value. If any SMR source is active, and the Stop Mode is executed, the part resets immediately.

All core interrupts are set to int0 > int1 > int2. These priorities can NOT be changed and are embedded into the core. However, Z90365 architecture provides flexibility to change the priority of the interrupts by switching the interrupt sources between interrupt inputs of the Z90365 core. The correspondence between H_{SYNC}, V_{SYNC} and 1s/CAP interrupts sources, and int0, int1 and int2 interrupts inputs of the Z90365 is listed in Table 29.

Table 29 Interrupt Priority

Int_Priority Field	HSYNC Is Switched To:	VSYNC Is Switched To:	1s/CAP Is Switched To:	Int_Priority Field
0 0 0	int0	int1	int2	0 0 0
0 0 1	int0	int2	int1	0 0 1
0 1 0	int1	int0	int2	0 1 0
0 1 1	int2	int0	int1	0 1 1
1 0 0	int1	int2	int0	1 0 0
1 0 1	int2	int1	int0	1 0 1

3.4 Bank2 (PWM Registers)

Table 30 list the bits for the PWM registers.

Table 30 Register0–Register5, R0(2)–R4(2) PWM 1–5 Registers

Bit	15	14	13	12	11	10	9	8
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0
Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	x	x	x	x	x	x	x	x

Note: R = Read W = Write X = Indeterminate

Reg Field	Bit Position	R	W	Data	Description
Reserved	fedcba98--------	R			Return 0
			W		No effect
PWM_data	--------76543210	R	W	xx	8-bit PWM data

All of the PWMs feature push-pull. Outputs from all PWMs are staged by one PVCO clock. The repetition frequency of the PWM output signals can be calculated from the following equation:

$$F_{PWM} = \frac{F_{PVCO}}{8-256} = \frac{12MHz}{2048} = 6kHz$$

When reset, PWM_data registers are not initialized; however, PWM output is set to 0. Because the PWM is clocked with PVCO, it is better to initialize the PWM_data before enabling PVCO.

3.5 Bank3 (On Screen Display [OSD] registers)

Table 31 lists the R0(3)- R2(3) character multiplier registers (Read operation).
Table 32 lists the R0(3)–R1(3) Shift Registers (Write Operation).
Table 33 lists the R2(3) Attributes Register (Write Operation).

Table 31 Register0–Register2 Read Operation R0(3), R2(3) Character Multiple Registers

Bit	15	14	13	12	11	10	9	8
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	x	x	x	x	x	x	x	x
Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	x	x	x	x	x	x	x	x

Note: R = Read W = Write X = Indeterminate

Reg Name	CGROM Data	Reg Add	Description
cgrom_x2_hi	ffeeddccbbaa9988	R0(3)	High word of double size character R4(3)<6> = 0
cgrom_x3_hi	fffeeedddcccbbba		High word of triple size character R4(3)<6> = 1
cgrom_x2_lo	7766554433221100	R1(3)	Low word of double size character R4(3)<6> = 0
cgrom_x3_mid	aa99988877766655		Middle word of triple size character R4(3)<6> = 1
cgrom_x1	fedcba9876543210	R2(3)	Single size character R4(3)<6> = 0
cgrom_x3_lo	5444333222111000		Low word of triple size character R4(3)<6> = 1

Table 32 Register0–Register1 Write Operation R0(3), R1(3) Shift Registers

Bit	15	14	13	12	11	10	9	8
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	x	x	x	x	x	x	x	x
Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	x	x	x	x	x	x	x	x

Note: R = Read W = Write X = Indeterminate

Reg Name	CGROM Data	Reg Address	Description
current_reg	fedcba9876543210	R0(3)	current line shift register
next/previous_reg	fedcba9876543210	R1(3)	next/previous line shift register

Registers R1(3) and R0(3) must be loaded with video data once every 16 cycles. To support smoothing, register R1(3) must be updated every 16 cycles. The current line register is loaded first, followed by next/previous register during the next cycle. The next/previous register is loaded only if smoothing/fringing attributes are activated for the current character. If neither register is loaded, the space character is displayed. There is no difference between loading 0000 hex into either register or not loading at all.

Table 33 Register2, R2(3) Attributes Register, Write Operation

Bit	15	14	13	12	11	10	9	8
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	x	x	x	x	x	x	x	x
Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	x	x	x	x	x	x	x	x

Note: R = Read W = Write X = Indeterminate

Reg Field	Bit Position	Data	Description
Reserved	f---------------	x	No effect
Background color	-edc----------	000 001 010 011 100 101 110 111	Black Blue Green Cyan Red Magenta Yellow White
Foreground color NOT Palette Mode	----ba9---------	%D	Same as Background mode
Palette selection Palette Mode	----ba----------	00 01 10 11	Palette0 Palette1 Palette2 Palette3

Reg Field	Bit Position	Data	Description
2nd_underline Palette Mode	------9---------	1 0	Second Underline is active Second Underline is NOT active
1st_underline	-------8--------	1 0	First Underline is active First Underline is NOT active
Shift_video	--------7-------	1 0	Video signal is delayed by 8 pixels Standard character positioning
Transparent	---------6------	1 0	Transparent background Background color defined by “background color” field
Blinking	----------5-----	1 0	Blinking character Not blinking character
Italic	-----------4----	1 0	Italic character Not italic character
Color_delay	------------32--	00 01 10 11	Character color changes instantly Color changes with 4 pixels delay Color changes with 8 pixels delay Color changes with 12 pixels delay
Fringing	--------------1-	1 0	Fringing logic is enabled Fringing logic is disabled
Smoothing	---------------0	1 0	Smoothing logic is enabled Smoothing logic is disabled

Fringing is active only if the current background is transparent.

The attributes register must be loaded 8 cycles after the current line register R0(3) is loaded. Loading the attributes register enables the OSD logic during the next 16 cycles. If the attributes register is not loaded, there is no active OSD, even if the current line register R0(3) is loaded.

Table 34 Register3 Read Operation R3(3) Attributes Register

Bit	15	14	13	12	11	10	9	8
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	x	x	x	x	x	x	x	x
Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	x	x	x	x	x	x	x	x

Note: R = Read W = Write X = Indeterminate

Reg Field	Bit Position	Data	Description
Same as R2(3)			

The data read from the attribute register is a combination of attribute fields from the most recently displayed character and control character codes loaded into the attribute_data register. Character codes are fetched from Video RAM and must be loaded into the attribute_data register R3(3). Bit <f> of the attribute_data register (during a read) indicates whether the most recent character was a control or displayed character. The data read from the attribute_data register must be directly loaded into attribute register R2(3). Refer to Table 35.

Table 35 Register3, R3(3) Write Operation Attribute Data Register

Bit	15	14	13	12	11	10	9	8
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	x	x	x	x	x	x	x	x
Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	x	x	x	x	x	x	x	x

Note: R = Read W = Write X = Indeterminate

Reg Field	Bit Position	Data	Description
VRAM_data	fedcba9876543210	xxxx	Character code fetched from VRAM

Loading VRAM data into an attribute_data register initializes a CGROM access cycle. Four clock cycles after the LD instruction, the Z90365 halts for three clock cycles to fetch the data from CGROM and latch it into a CGROM data capture register. After the CGROM data is latched, core operations are resumed. When a control character code is loaded into the attribute_data register, the CGROM data from address `0000` hex is fetched. Therefore, ZiLOG recommends placing a space character at location `0000` hex in CGROM. Refer to Tables 36 through 39 for the various VRAM data formats loaded in R3(3).

Table 36 Display Character Format for Attribute Data Register R3(3) OSD Mode Write Operation

Bit	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	x	x	x	x	x	x	x	x
Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	x	x	x	x	x	x	x	x

Note: R = Read W = Write X = Indeterminate

Reg Field	**Bit Position**	**Data**	**Description**
Control bit	`f---------------`	0	Display character
Background color*	`-edc------------`	000 001 010 011 100 101 110 111	Black Blue Green Cyan Red Magenta Yellow White
Foreground color (Not Palette mode)	`------ba9-------`	%D	Same as Background_color
Foreground palette (Palette mode)	`--------ba------`	00 01 10 11	Palette 0 (defined in R6(3)<8–6> Palette 1 (defined in R6(3)<b–9> Palette 2 (defined in R6(3)<5–3> Palette 3 (defined in R6(3)<2–0>
Second underline (Palette mode)	`-------9--------`	1 0	Second underline attribute is active Second underline attribute is inactive

Reg Field	Bit Position	Data	Description
Attribute8	-------8--------	1 0	Selected attribute (R7(3),<76>) is active Selected attribute (R7(3),<76>) is inactive
Character code	--------76543210	%DD	Defines the character in CGROM

Note: *If both the background and foreground colors of a character are set to be the same, the character's background is displayed as transparent.

Table 37 Control Character Format, OSD Mode Write Operation Attribute Data Register R3(3)

Bit	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	x	x	x	x	x	x	x	x
Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	x	x	x	x	x	x	x	1

Note: R = Read W = Write X = Indeterminate

Reg Field	Bit Position	Data	Description
Control bit	f---------------	1	Control character
Reserved	-edcba98--------	x	Reserved—do not effect OSD
Shift_video	--------7-------	1 0	Video signal is delayed by 8 pixels Standard character positioning
Transparent	---------6------	1 0	Transparent background Background color defined by "background color" field
Blinking	----------5-----	1 0	Blinking character Not blinking character
Italic	-----------4----	1 0	Italic character Not italic character
Color_delay	------------32--	00 01 10 11	Character color changes instantly Color changes with 4 pixels delay Color changes with 8 pixels delay Color changes with 12 pixels delay

Reg Field	Bit Position	Data	Description
Fringing	----------1-	1 0	Fringing logic is enabled Fringing logic is disabled
Smoothing	----------0	1 0	Smoothing logic is enabled Smoothing logic is disabled

Smoothing is supported for double size (x2) and triple size (x3) characters only.

At reset, the background color in OSD mode is black. Foreground color, background color, blinking and italic attributes are delayed by 3/4 character. The smoothing attribute is enabled.

Table 38 Display Character Format, Attribute Data Register R3(3) Write Operation CCD Mode

Bit	15	14	13	12	11	10	9	8
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	x	x	x	x	x	x	x	x
Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	x	x	x	x	x	x	x	x

Note: R = Read W = Write X = Indeterminate

Reg Field	Bit Position	Data	Description
Control bit	7-------	0	Display character
Character code	-6543210	%DD	Defines the character in CGROM.

Table 39 Control Character Format, Attribute Data Register R3(3) CCD Mode, Write Operation

Bit	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	x	x	x	x	x	x	x	x
Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	x	x	x	x	x	x	x	x

Note: R = Read W = Write X = Indeterminate

Reg Field	**Bit Position**	**Data**	**Description**
Control bit	7-------	1	Control character
Transparent	-6------	1 0	Transparent background Background color defined by “background color” field
Blinking	--5-----	1 0	Blinking character Not blinking character
Italic	---4----	1 0	Italic character Not italic character
Foreground color	----321-	000 001 010 011 100 101 110 111	Black Blue Green Cyan Red Magenta Yellow White
First underline	-------0	1 0	Underline attribute is active Underline attribute is inactive

In CCD mode, each character occupies 8 bits (one byte) in VRAM. The CCD characters must be mapped into a 16 bit VRAM data field. The hardware supports compressed character placement in VRAM. Each word in VRAM is represented by HIGH byte and LOW byte. A currently active byte is selected by R4(3)<c>. The format and data representation for both bytes is the same.

There are two possible character formats defined: a “display” character and a “control” character. The code stored in “display” character format defines a character code. The

"control" character defines five latched attributes of the next character and is presented on screen as a space character.

Combining display and control characters generates a CCD or OSD according to FCC specification. Refer to Table 40.

Table 40 Register4 - R4(3) OSD Control Register

Bit	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	x	x	x	x	x	x	x	x
Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	x	x	x	x	x	x	x	x

Note: R = Read W = Write X = Indeterminate

Reg Field	Bit Position	R	W	Data	Description
Underline	fe----------	R	W	1x 0x x1 x0	Second underline is active Second underline is inactive First underline is active First underline is inactive
OSD/CCD	--d----------	R	W	1 0	OSD mode CCD mode
CCD_top/btm	---c----------	R	W	1 0	The upper byte in VRAM is used The lower byte in VRAM is used
Italic_shift	----ba98--------	R	W	x	Defines delay of the character
Blink_off/on	--------7-------	R	W	0 1	Blinking character is displayed Blinking character is NOT displayed (hidden)
MPX_bus	----------65----	R	W	00 01 10 11	x1 character size x2 character size x3 character size Reserved
CGROM scan_line	----------43210	R	W	%D	Defines CGROM addressing

The Underline field must be set by firmware when scan lines that contain underline information are displayed. The underline bits are ANDed with the 2nd and 1st underline

active fields of data loaded into attribute register R2(3), causing the screen character to be underlined.

The Italic shift field defines a delay of video data. It is used to generate italic characters. The firmware decrements by 1 (the value of the Italic_shift field) for each consecutive line. The video signal is delayed only for characters that have the R2(3)<4> (“italic”) bit set to 1.

Table 41 Register5, R5(3) Capture Register, Read Operation

Bit	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	x	x	x	x	x	x	x	x
Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	x	x	x	x	x	x	x	x

Note: R = Read W = Write X = Indeterminate

Reg Field	**Bit Position**	**R**	**W**	**Data**	**Description**
Cap_data	fedcba9876543210	R		%xxxx	16-bit captured data
Reserved	fedcba98-------0	R	W		Return 0 No effect
I2C_saddr	--------7654321-		W	%DD	I^2C Slave interface address

In Read mode, R5(3) returns the 16-bit captured data from the IRIN pin.
In Write mode, the 7-bit I^2C slave interface address must be put in bit 7-1.

Table 42 Register6, R6(3) Palette Control Register

Bit	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0
Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Note: R = Read W = Write X = Indeterminate

Reg Field	Bit Position	R	W	Data	Description
Palette	f---------------	R	W	1 0	Palette mode is active Palette mode is INACTIVE
Underline color	-edc------------	R	W	000 001 010 011 100 101 110 111	Black Blue Green Cyan Red Magenta Yellow White
Palette1	----ba9---------	R	W	%D	Same as Underline color
Palette0	-------876------	R	W	%D	Same as Underline color
Palette3	----------543---	R	W	%D	Same as Underline color
Palette2	-------------210	R	W	%D	Same as Underline color

At POR the palette control register is reset to 0.

Table 43 Register7, R7(3) Output Palette Control Register

Bit	15	14	13	12	11	10	9	8
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0
Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Note: R = Read W = Write X = Indeterminate

Reg Field	Bit Position	R	W	Data	Description
VBLANK_delay	fedc------------	R	W		VBlank delay value—%00–POR condition
Background_on/off	----b-----------	R	W	1 0	Master background is on Master background is off–POR condition
Background_color	-----a98--------	R	W	%D	Defines the color of the Master background (same as the palette)
Reserved	--------7-------	R W			Return 0 No effect

Reg Field	Bit Position	R	W	Data	Description
StarSight palette	---------6------	R	W	0 1	Palette is defined by bits<5..0> See Starlight Palette table
Palette #	----------54----	R	W	00 01 10 11	V1(red) = 2 V1(red) = 2 + 1*B V1(red) = 2 + 1*G V1(red) = 2 + 1*(B or G)
	------------32--	R	W	00 01 10 11	V2(green) = 2 V2(green) = 2 + 1*B V2(green) = 2 + 1*R V2(green) = 2 + 1*(B or R)
	--------------10	R	W	00 01 10 11	V3(blue) = 2 V3(blue) = 2 + 1*G V3(blue) = 2 + 1*R V3(blue) = 2 + 1*(G or R)
Digital_mode	----------543210	R	W	0000 00	Outputs V1, V2, V3 correspond to 100% red, green,& blue outputs.

At POR the Output palette register is set to 0 for digital output.

Note: If bit R7(3)<6> is set to a 1, and bits R7(3)<5:0> are set to 0, the outputs V1, V2 and V3 of Z90365 are still switched into a Digital_ mode. ZiLOG recommends writing the value %65 into bits 6 through 0 to activate the StarSight palette.

Table 44 is the look-up table for the color palette. Table 45 is the look up table for the StarSight palette

Table 44 Color Palette

	Palette #			Color #1			Color #2			Color #3			Color #4			Color #5			Color #6			Color #7		
	R	G	B	R	G	B	R	G	B	R	G	B	R	G	B	R	G	B	R	G	B	R	G	B
	V1	V2	V3	0	0	1	0	1	0	0	1	1	1	0	0	1	0	1	1	1	0	1	1	1
1	0	0	1	0	0	66	0	66	33	0	66	99	66	0	0	66	0	66	66	66	33	66	66	99
2	0	0	2	0	0	66	0	66	0	0	66	66	66	0	33	66	0	99	66	66	33	66	66	99
3	0	0	3	0	0	66	0	66	33	0	66	99	66	0	33	66	0	99	66	66	66	66	66	99
4	0	1	0	0	33	66	0	66	0	0	99	66	66	0	0	66	33	66	66	66	0	66	99	66
5	0	1	1	0	33	66	0	66	33	0	99	99	66	0	0	66	33	66	66	66	33	66	99	99
6	0	1	2	0	33	66	0	66	0	0	99	66	66	0	33	66	33	99	66	66	33	66	99	99
7	0	1	3	0	33	66	0	66	33	0	99	99	66	0	33	66	33	99	66	66	66	66	99	99
8	0	2	0	0	0	66	0	66	0	0	66	66	66	33	0	66	33	66	66	99	0	66	99	66
9	0	2	1	0	0	66	0	66	33	0	66	99	66	33	0	66	33	66	66	99	33	66	99	99
10	0	2	2	0	0	66	0	66	0	0	66	66	66	33	33	66	33	99	66	99	33	66	99	99
11	0	2	3	0	0	66	0	66	33	0	66	99	66	33	33	66	33	99	66	99	66	66	99	99
12	0	3	0	0	33	66	0	66	0	0	99	66	66	33	0	66	33	66	66	99	0	66	99	66
13	0	3	1	0	33	66	0	66	33	0	99	99	66	33	0	66	33	66	66	99	33	66	99	99
14	0	3	2	0	33	66	0	66	0	0	99	66	66	33	33	66	33	99	66	99	33	66	99	99
15	0	3	3	0	33	66	0	66	33	0	99	99	66	33	33	66	33	99	66	99	66	66	99	99
16	1	0	0	33	0	66	0	66	0	33	66	66	66	0	0	99	0	66	66	66	0	99	66	66
17	1	0	1	33	0	66	0	66	33	33	66	99	66	0	0	99	0	66	66	66	33	99	66	99
18	1	0	2	33	0	66	0	66	0	33	66	66	66	0	33	99	0	99	66	66	33	99	66	99
19	1	0	3	33	0	66	0	66	33	33	66	99	66	0	33	99	0	99	66	66	66	99	66	99
20	1	1	0	33	33	66	0	66	0	33	99	66	66	0	0	99	33	66	66	66	0	99	99	66
21	1	1	1	33	33	66	0	66	33	33	99	99	66	0	0	99	33	66	66	66	33	99	99	99
22	1	1	2	33	33	66	0	66	0	33	99	66	66	0	33	99	33	99	66	66	33	99	99	99
23	1	1	3	33	33	66	0	66	33	33	99	99	66	0	33	99	33	99	66	66	66	99	99	99
24	1	2	0	33	0	66	0	66	0	33	66	66	66	33	0	99	33	66	66	99	0	99	99	66
25	1	2	1	33	0	66	0	66	33	33	66	99	66	33	0	99	33	66	66	99	33	99	99	99
26	1	2	2	33	0	66	0	66	0	33	66	66	66	33	33	99	33	99	66	99	33	99	99	99
27	1	2	3	33	0	66	0	66	33	33	66	99	66	33	33	99	33	99	66	99	66	99	99	99

Table 44 Color Palette (Continued)

	Palette #			Color #1			Color #2			Color #3			Color #4			Color #5			Color #6			Color #7		
28	1	3	0	33	33	66	0	66	0	33	99	66	66	33	0	99	33	66	66	99	0	99	99	66
29	1	3	1	33	33	66	0	66	33	33	99	99	66	33	0	99	33	66	66	99	33	99	99	99
30	1	3	2	33	33	66	0	66	0	33	99	66	66	33	33	99	66	99	66	99	33	99	99	99
31	1	3	3	33	33	66	0	66	33	33	99	99	66	33	33	99	66	99	66	99	66	99	99	99
32	2	0	0	0	0	66	33	66	0	33	66	66	66	0	0	66	0	66	99	66	0	99	66	66
33	2	0	1	0	0	66	33	66	33	33	66	99	66	0	0	66	0	66	99	66	33	99	66	99
34	2	0	2	0	0	66	33	66	0	33	66	66	66	0	33	66	0	99	99	66	33	99	66	99
35	2	0	3	0	0	66	33	66	33	33	66	99	66	0	33	66	0	99	99	66	66	99	66	99
36	2	1	0	0	33	66	33	66	0	33	99	66	66	0	0	66	33	66	99	66	0	99	99	66
37	2	1	1	0	33	66	33	66	33	33	99	99	66	0	0	66	33	66	99	66	33	99	99	99
38	2	1	2	0	33	66	33	66	0	33	99	66	66	0	33	66	33	99	99	66	33	99	99	99
39	2	1	3	0	33	66	33	66	33	33	99	99	66	0	33	66	33	99	99	66	66	99	99	99
40	2	2	0	0	0	66	33	66	0	33	66	66	66	33	0	66	33	66	99	99	0	99	99	66
41	2	2	1	0	0	66	33	66	33	33	66	99	66	33	0	66	33	66	99	99	33	99	99	99
42	2	2	2	0	0	66	33	66	0	33	66	66	66	33	33	66	33	99	99	99	33	99	99	99
43	2	2	3	0	0	66	33	66	33	33	66	99	66	33	33	66	33	99	99	99	66	99	99	99
44	2	3	0	0	33	66	33	66	0	33	99	66	66	33	0	66	66	66	99	99	0	99	99	66
45	2	3	1	0	33	66	33	66	33	33	99	99	66	33	0	66	66	66	99	99	33	99	99	99
46	2	3	2	0	33	66	33	66	0	33	99	66	66	33	33	66	66	99	99	99	33	99	99	99
47	2	3	3	0	33	66	33	66	33	33	99	99	66	33	33	66	66	99	99	99	66	99	99	99
48	3	0	0	33	0	66	33	66	0	66	66	66	66	0	0	99	0	66	99	66	0	99	66	66
49	3	0	1	33	0	66	33	66	33	66	66	99	66	0	0	99	0	66	99	66	33	99	66	99
50	3	0	2	33	0	66	33	66	0	66	66	66	66	0	33	99	0	99	99	66	33	99	66	99
51	3	0	3	33	0	66	33	66	33	66	66	99	66	0	33	99	0	99	99	66	66	99	66	99
52	3	1	0	33	33	66	33	66	0	66	99	66	66	0	0	99	33	66	99	66	0	99	99	66
53	3	1	1	33	33	66	33	66	33	66	99	99	66	0	0	99	33	66	99	66	33	99	99	99
54	3	1	2	33	33	66	33	66	0	66	99	66	66	0	33	99	33	99	99	66	33	99	99	99
55	3	1	3	33	33	66	33	66	33	66	99	99	66	0	33	99	33	99	99	66	66	99	99	99
56	3	2	0	33	0	66	33	66	0	66	66	66	66	33	0	99	33	66	99	99	0	99	99	66

Table 44 Color Palette (Continued)

	Palette #			Color #1			Color #2			Color #3			Color #4			Color #5			Color #6			Color #7		
57	3	2	1	33	0	66	33	66	33	66	66	99	66	33	0	99	33	66	99	99	33	99	99	99
58	3	2	2	33	0	66	33	66	0	66	66	66	66	33	33	99	33	99	99	99	33	99	99	99
59	3	2	3	33	0	66	33	66	33	66	66	99	66	33	33	99	33	99	99	99	66	99	99	99
60	3	3	0	33	33	66	33	66	0	66	99	66	66	33	0	99	66	66	99	99	0	99	99	66
61	3	3	1	33	33	66	33	66	33	66	99	99	66	33	0	99	66	66	99	99	33	99	99	99
62	3	3	2	33	33	66	33	66	0	66	99	66	66	33	33	99	66	99	99	99	33	99	99	99
63	3	3	3	33	33	66	33	66	33	66	99	99	66	33	33	99	66	99	99	99	66	99	99	99
65				0	99	99	66	99	66	66	66	66	99	33	33	99	99	66	99	99	0	99	99	99

Table 45 StarSight Palette Color Definitions

Bit Data	Color	V1	V2	V3
000	Black	0	0	0
001	Blue	0	99%	99%
010	Green	66%	99%	66%
011	Grey	66%	66%	66%
100	Red	99%	33%	33%
101	Light Yellow	99%	99%	66%
110	Yellow	99%	99%	0
111	White	99%	99%	99%

4 INSTRUCTION SET

The processor instruction set consists of 30 basic instructions. It has been optimized for high code density and reduced execution time. Single-cycle instruction execution is possible on most instructions.

The format for op codes and addressing modes is provided in the following tables but is normally not required. The assembler removes the burden of hand constructing the instruction format. by translating the mnemonics. System designers can access the instruction format when debugging.

4.1 Instruction Summary

The DSP instruction set can be broken down into the following types of instructions:

- Accumulator Modification
- Arithmetic
- Bit Manipulation
- Load
- Logical
- Program Control
- Rotate and Shift

Instruction format mnemonics are in Table 46. Tables 47 through 53 list other instructions.

Table 46 Instruction Format Mnemonics

Mnemonic	Description
A	Address
am	Accumulator Modification
b	RAM Bank
cc	Condition Code
const exp	Constant Expression
d	Destination Address
dest	Destination Value
fm	Flag Modification
op	Op Code
rp	Register Pointer
s	Source Address
src	Source Value

Table 47 Accumulator Modification Instructions

Mnemonic	Operands	Instruction
ABS	<cc>, A	Absolute Value
CP	A, <src>	Comparison
DEC	<cc>, A	Decrement
INC	<cc>, A	Increment
NEG	<cc>, A	Negate

Table 48 Arithmetic Instructions

Mnemonic	Operands	Instruction
ADD	<cc>, A	Add
CP	A, <src>	Compare
SUB	A, <src>	Subtract

Table 49 Bit Manipulation Instructions

Mnemonic	Operands	Instruction
CCF	None	Clear Carry Flag
CIEF	None	Clear Interrupt Enable Flag
COPF	None	Clear Overflow Protection Flag
SCF	None	Set Carry Flag
SIEF	None	Set Interrupt Enable Flag
SOPF	None	Set Overflow Protection Flag

Table 50 Load Instructions

Mnemonic	Operands	Instruction
LD	<dest>, <src>	Load
POP	<dest>	Pop
PUSH	<src>	Push

Table 51 Logical Instructions

Mnemonic	Operands	Instruction
AND	A, <src>	Logical AND
OR	A, <src>	Logical OR
XOR	A, <src>	Logical Exclusive OR

Table 52 Program Control Instructions

Mnemonic	Operands	Instruction
CALL	A	Call Procedure
JP	A	Jump
RET	None	Return

Table 53 Rotate and Shift Instructions

Mnemonic	Operands	Instruction
RL	<cc>, A	Rotate Left
RR	<cc>, A	Rotate Right
SLL	<cc>, A	Shift Left Logical
SRA	<cc>, A	Shift Right Arithmetic

4.2 Instruction Operands

To access the operands for the DSP, use the Register Pointers, Data Pointers, Hardware Registers, Direct Addressing, Immediate Data and Memory. There are nine distinct types of instruction operands. Tables 54 and 55 describe instruction operands.

Table 54 Instruction Operand Summary

Symbolic Name	Syntax	Description
<pregs>	Pn:b	Register Pointer
<dregs>	Dn:b	Data Pointer
<hwregs>	X, Y, PC, SR, EXTn, A, BUS	Hardware Registers
<accind>	@A	Accumulator Indirect
<direct>	<const exp>	Direct Address Expression
<limm>	#<const exp>	Long (16-Bit) Immediate
<simm>	#<const exp>	Short (8-Bit) Immediate Value
<regind>	@Pn:b @Pn:b+ @Pn:b+Loop @Pn:b-Loop	Indirect Addressing of RAM
<memind> @Dn:b	@Dn:b @@Pn:b @@Pn:b+ @@Pn:b+Loop @@Pn:b-Loop	Indirect Addressing of ROM

Table 55 Instruction Mnemonics/Operands

#	Instruction		Mnemonic/Operand Representation	
1	LD	P2:0, #%F2	LD	<pregs>, <simm>
2	LD	X, @A	LD	<hwreg>, <accind>
3	LD	Y, #%3CF5	LD	<hwreg>, <limm>
4	SUB	A, @@P2:0	SUB	A, <memind>
5	OR	A, @D2:0	OR	A, <memind>
6	AD	D A, %F2	ADD	A, <direct>
7	PUSH	D1:1	PUSH	<dregs>

<pregs> The register pointer mode is used for loading the pointer with the appropriate RAM address. This address references the RAM location that stores the requested data. The pointer can also be used to store 8-bit data when used as a temporary register. The

pointers are connected to the lower 8 bits of the D-bus. Instruction 1 loads Pointer 2, RAM Bank0 with the value F2H.

<regind> The register indirect mode is used for indirect access to RAM. As noted in Instruction 2, the register indirect address method is used to get the operand to multiply it with the accumulator.

<dregs> The data-pointer mode is used as an indirect addressing method similar to @P2:0. The data pointers access the lower 16 bits of each RAM bank. Instruction 8 uses indirect addressing to PUSH information onto the stack.

<memind> Pointer or data registers can be used to access program memory. Both are commonly used to reference program memory. Instructions 5 and 6 display this addressing method. Either pointer is automatically incremented to assist in transferring sequential data.

<accind> Another method of indirect addressing is using the accumulator to store the address. Instruction 3 describes how to use this method.

<direct> The absolute RAM address is used in the direct mode. A range between 0 and 511 (000H to 1FFH) is allowed. The accumulator is used in conjunction with this method as a source or destination operand. Instruction 7 displays the accumulator as the destination.

<limm> This instruction indicates a long immediate load. A 16-bit word can be copied directly from the operand into the specified register or memory. Instruction 4 uses this method.

<simm> This instruction can only be used for immediate transfer of 8-bit data in the operand to the specified RAM pointer.

4.3 Instruction Format

The instruction format that specifies to the processor the action to be taken consists of the op code, destination, source, and other special bits. The assembler makes this operation transparent by providing mnemonics. Occasionally, the instruction format and development code can assist in debugging. Examples to clarify the various instruction formats and explain how specific bit patterns are developed and evaluated are provided below.

Most instructions require one 16-bit word containing the information necessary for the processor to execute the instruction correctly. This process requires one clock cycle for execution. Immediate addressing, immediate operands, JUMP and CALL instructions require two 16-bit words (two clock cycles). Each instruction type has a unique op code and format to differentiate various instructions. Different operations also have unique formats.

The variables `a, op, b, d, s, cc, am, fm, rp` are used in the instruction format to depict bits determined by the active instruction.

ADDITION and INC Formats

The op code and format for an instruction differ to allow the processor to differentiate between the instructions. For example, the ASSITION instruction requires that two operands be defined in the instruction.

```
ADD A, <regind>
```

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	0	0	1	1	0	0	0	0	S	S	S	S
Op Code							RAM Bank	Condition Code				Modification Code			

The `INC` (increment) instruction requires that a condition and modification code be specified.

```
INC A
```

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	1	0	0	0	0	0	1	0	0	1	0	1	0
Op Code								Condition Code				Modification Code			

INC and SLL Formats

The `INC` and `SLL` instructions have the same op code with an accumulator modification format. The last four bits, the modification code, determine the type of operation the accumulator performs.

```
INC A
```

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0
Op Code								Condition Code				Modification Code			

```
SLL A
```

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	1	0	0	0	0	0	0	0	0	0	0	1	1
Op Code								Condition Code				Modification Code			

4.4 Instruction Bit Codes

The values in a series of bits in a register form patterns called `bit codes`. Types of bit codes include

- Condition Codes
- Accumulator Modification Code
- Flag Modification Codes
- Source/Destination Field Designators
- Register Pointer/Data Pointer

The following tables list the options available and their corresponding instructions.

Condition Codes

Condition codes are used in accumulator modification, `CALL`, and `JUMP` instructions.

Table 56 Condition Code Bits

Bit Code	Mnemonic Code Value	Condition Code Value	Condition Code
00000	F		False
00001			Unused
00010	NU0	UI0 is set to 0	Not User Zero
00011	NU1	UI1 is set to 0	Not User One
00100	NC	C is set to 0	No Carry
00101	NZ	Z is set to 0	Not Zero (Not Equal)
00110	NOV	OV is set to 0	No Overflow
00111	PL	N is set to 0	Plus (Not Negative)
01xxx			Unused
10000	T		True
10001			Unused
10010	U0	UI0 is set to 1	User Zero
10011	U1	UI1 is set to 1	User One

Table 56 Condition Code Bits (Continued)

Bit Code	Mnemonic Code Value	Condition Code Value	Condition Code
10100	C	C is set to 1	Carry
10101	Z	Z is set to 1	Zero (Equal)
10110	OV	OV is set to 1	Overflow
10111	MI	N is set to 1	Minus (Negative)
11xxx			Unused

Accumulator Modification Codes

Accumulator modification codes determine the type of modification made to the value in the accumulator. Condition codes are also used with CALL, and JUMP instructions.

Table 57 Accumulator Modification Bits

Bit Code	Mnemonic	Operation
0000	RR	Rotate Right
0001	RL	Rotate Left
0010	SR	Shift Right
0011	SL	Shift Left
0100	INC	Increment
0101	DEC	Decrement
0110	NEG	Negate
0111	ABS	Absolute

Flag Modification Codes

Flag modifications initialize or set/reset bits to accommodate interrupts, overflows, and carrys.

Table 58 Flag Modification Bits

Bit Code	Mnemonic	Operation	Flag	Value
xx10	CCF	Clear Carry	C	0
xx11	SCF	Set Carry	C	1
x1x0	CIEF	Clear Interrupt Enable	IE	0
x1x1	SIEF	Set Interrupt Enable	IE	1
1xx0	COPF	Clear Overflow Protection	OP	0
1xx1	SOPF	Set Overflow Protection	OP	1

Source/Destination Field Designators

Register pointers and data pointers provide convenient access to data. The pointers are a source or destination field in instructions. Specific bit codes are listed below. The register pointer offers optional incrementing or decrementing. This option is specified by thefollowing instruction:

```
LD A, @P2:1+
```

Table 59 Register Pointer/ Data Pointer Bits

Bit Code	Mnemonic
00xx	NOP
01xx	+1
10xx	-1/loop
11xx	+1/loop
xx00	P0:0 or P0:1
xx01	P1:0 or P1:1
xx10	P2:0 or P2:1
0011	D0:0 or D0:1
0111	D1:0 or D1:1
1011	D2:0 or D2:1
1111	D3:0 or D3:1

Data pointers are automatically incremented when accessing program memory (for example, `LD A, @D0:0`) and do not require an incrementing option. Code in `xx11` format is designated for a data pointer when source or destination format is used.

Additional source or destination designators include the other hardware registers provided by the processor. To determine if a data pointer, register pointer or a register is used as a source or destination is discussed in the next section.

Table 59 lists the bit codes for mnemonic resister names.

Table 60 Register Bits

Bit Code	Mnemonic
0000	Bus
0001	X
0010	Y
0011	A
0100	SR
0101	STACK
0110	PC
0111	Reserved
1000	EXT0
1001	EXTI
1010	EXT2
1011	EXT3
1100	EXT4
1101	EXT5
1110	EXT6
1111	EXT7

4.5 Instruction Format Examples

To Refer to the following examples showing how bit codes are used in an instruction format.

Instruction Format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
op	op	op	op	op	op	op	cc	cc	cc	cc	cc	am	am	am	am
Op Code							Condition Code					ACC Modification			

Accumulator Modification Format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
op	op	op	op	op	op	op	cc	cc	cc	cc	cc	am	am	am	am
Op Code							Condition Code					ACC Modification			

Notes:

1. The Variables `a, op, b, d, s, cc, am, fm, rp` are used in the instruction format to depict bits determined by the instruction.
2. The General Instruction Format requires an op code, RAM bank bit, destination and source addresses. For example, `LD A, @P2:1+`

Load Instruction Format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	1	1	0	0	1	1	0	1	1	0
Op Code							RAM Bank	Destination				Source			

The op code (`0000001`) provides a unique signature for the `LD` command. The processor uses this signature to determine the instruction format. The RAM bank bit is high (equal to 1) because of the instruction definition `b=1 (Pn:b)`. The destination bit code is `0011` which corresponds to the accumulator. The source `0110` corresponds to the `+1` option and `P2:0` or `P2:1`. The RAM bank bit shows that the processor loaded the accumulator with the operand designated by Pointer 2 Bank1 (`P2:1`).

Source and destination fields can be accessed from the register pointers, data pointers, or registers. The op code specifies the type of source and destination. An op code of

0000101 specifies that the source is an indirect address to program memory (@@P0.0 or @D0:0) and the destination is a register.

Instruction Format Listing

Instruction formats and applicable instructions are listed in the following tables.

Notes:

1. Several instructions provide various addressing modes to obtain operands; therefore, the same instruction can have several different formats depending on the addressing mode.
2. The variables a, op, b, d, s, ce, am, fm, rp are used in the instruction format to depict bits determined by the specific instruction used.

General Instruction Format

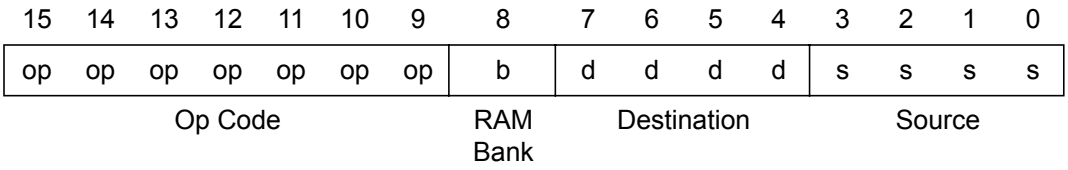


Table 61 General Instruction Format

Mnemonic	Operands	Bit Code	Hex
ADD	A, @P0:0	1000001 0 0000 0000	8200
AND	A, D11	1100110 1 0000 0111	AB07
CP	A,X	0110000 0 0000 0001	6001
LD	A,P	0000000 0 0011 0111	0037
POP	X	0000000 0 0001 0101	0015
PUSH	D0:0	0000001 0 0101 0011	0253
RET		0000000 0 0110 0101	0065
SUB	A,@D1:1	0010101 1 0000 0111	2B07
XOR	A,P2:0	1111001 0 0000 0010	F202

Accumulator Modification Format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
op	op	op	op	op	op	op	cc	cc	cc	cc	cc	am	am	am	am
Op Code							Condition Code					ACC Modification			

Table 62 Accumulator Modification Format

Mnemonic	Operands	Bit Code	Hex Representation
ABS	Z, A	1001000 10101 0111	9157
DEC	A	1001000 00000 0101	9005
INC	NZ, A	1001000 00101 0100	9054
NEG	A	1001000 00000 0110	9006
RR	NU0, A	1001000 00010 0000	9020
RL	PL, A	1001000 00111 0001	9071
SLL	C, A	1001000 10100 0011	9143
SRA	U1, A	1001000 10011 0010	9132

Flag Modification Format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
op	op	op	op	op	op	op	cc	cc	cc	cc	cc	fm	fm	fm	fm
Op Code							Condition Code					Flag Modification			

Table 63 Flag Modification Format

Mnemonic	Bit Code	Hex Representation
CCF	1001010 00000 0010	9402
CIEF	1001010 00000 0100	9404
COPF	1001010 00000 1000	9408
SCF	1001010 00000 0011	9403

Table 63 Flag Modification Format

Mnemonic	Bit Code	Hex Representation
SIEF	1001010 00000 0101	9405
SOPF	1001010 00000 1001	9409

Direct Internal Addressing Format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
op	op	op	op	op	op	op	a	a	a	a	a	a	a	a	a
Op Code							9-Bit Internal Address								

Table 64 Direct Internal Addressing Format

Mnemonic	Operands	Bit Code	Hex Representation
ADD	A, %FF	1000011 011111111	86FF
AND	A, 255	1010011 011111111	A6FF
CP	A, 255	0110011 011111111	66FF
LD		0000111 000010010	0E12

Short Immediate Addressing Format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
op	op	op	op	op	rp	rp	rp	a	a	a	a	a	a	a	a
Op Code					Register Pointer			8-Bit Immediate Address/Data							

Table 65 Short Immediate Addressing Format

Mnemonic	Operands	Bit Code	Hex Representation
LD	P1:1, #%FA	00011 101 11111010	1DFA

Long Immediate Addressing Format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
op	op	op	op	op	op	op	b	d	d	d	d	s	s	s	s
Op Code							RAM Bank	Destination				Source			
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a
16-Bit Address/Data															

Table 66 Long Immediate Addressing Format

Mnemonic	Operands	Hex Representation
ADD	A, #%1234	8800 1234
AND	A, #% 26A4	A800 26A4
LD	X, #%6FFC	0810 6FFC
PUSH	#%C32C	0850 C32C
SUB	A, #%2444	2800 2444
XOR	A,#%AFC2	E800 AFC2

JUMP, CALL Format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	1	0	cc	cc	cc	cc	cc	s	s	s	s
Op Code							Condition Code					Not Used			
16-Bit Address															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a

16-Bit Address

Table 67 Jump, Call Format

Mnemonic	Operands	Hex Representation
CALL	END	4800 0004
JP	U1, END	4D30 0004

4.6 Instruction Timing

The DSP can be executed with single cycle instruction using the independent data memory and program memory buses offered by the modified Harvard architecture and pipeline instruction. This method provides overlapping of instruction fetch and execution cycles. Figure 23 shows the execution sequence. The first instruction takes two clock cycles to execute; subsequent executions occur in a single cycle. All instruction fetch cycles have the same machine timing regardless of whether external or internal memory is used. Because the DSP contains a two-level pipeline, the JUMP and CALL instructions do not disrupt the execution process

In two-byte instructions, the second byte is being fetched while the first byte is executing. Because the processor knows that the instruction is a JUMP or CALL, the second byte is transferred to the program counter and the correct address is fetched into the pipeline. There is no disruption or pipeline flushing. The pipeline flow is affected when the program counter is the destination for a load. Because the load (LD) instruction is a single word instruction, the next instruction is fetched during load execution. To compensate for the instruction in the pipeline, that instruction is executed as a NOP.

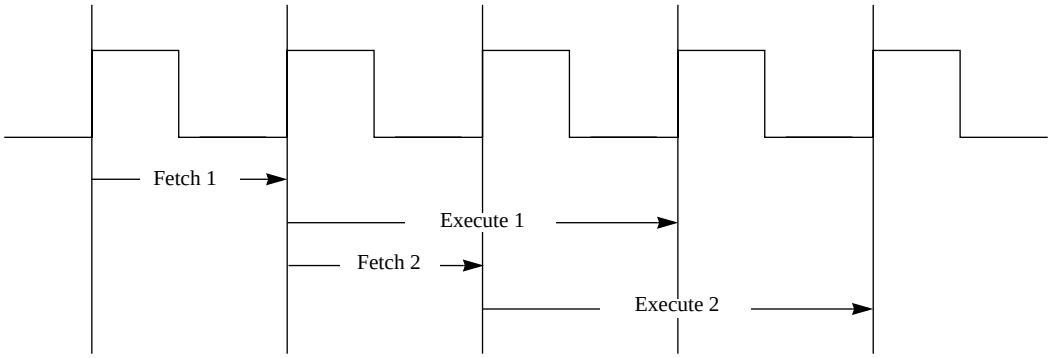


Figure 23 Pipeline Execution

4.7 Instruction Op Codes

Table 68 summarizes essential information about the instruction set.

Table 68 Instruction Op Codes

Inst	Description	Op Code	Synopsis	Operands	Words	Cycles	Examples
ABS	Absolute Value	1001000	ABS[<cc>,] <src>	<cc>,A	1	1	ABS NC, A
		1001000		A	1	1	ABS A
ADD	Addition	1001001	ADD<dest>, <src>	A,<pregs>	1	1	ADD A, P0:0
		1000001		A,<dregs>	1	1	ADD A, D0:0
		1000100		A,<limm>	2	2	ADD A, #%1234
		1000101		A,<memind>	1	3	ADD A, @@P0:0
		1000011		A,<direct>	1	1	ADD A, %F2
		1000001		A<regind>	1	1	ADD A, @P1:1
		1000000		A,<hwregs>	1	1	ADD A, X

Table 68 Instruction Op Codes (Continued)

Inst	Description	Op Code	Synopsis	Operands	Words	Cycles	Examples
AND	Bitwise AND	1011001	AND <dest>, <src>	A,<pregs>	1	1	AND A, P2:0
		1010001		A,<dregs>	1	1	AND A, D0:1
		1010100		A,<limm>	2	2	AND A, #%1234
		1010101		A,<memind>	1	3	AND A, @@P1:0
		1010001		A,<direct>	1	1	AND A, %2C
		1010001		A,<regind>	1	1	AND A, @1:2+LOOP
		1010000		A,<hwregs>	1	1	AND A, EXT3
CALL	Subroutine Call	0010100	CALL <cc>,<address>	<cc>,<direct>	2	2	CALL sub1
		0010100		<direct>	2	2	CALL Z, sub2
CCF	Clear Carry Flag	1001010	CCF	None	1	1	CCF
CIEF	Clear Carry Flag	1001010	CIEF	None	1	1	CIEF
COPF	Clear OP Flag	1001010	COPF	None	1	1	COPF
CP	Comparison		CP<src1>,<src2>				
		0111001		A, <pregs>	1	1	CP A, P0:0
		0110001		A, <dregs>	1	1	CP A, D3:1
		0110101		A, <memind>	1	3	CP A, @@P0:0
		0110011		A, <direct>	1	1	CP A, %FF
		0110001		A, <regind>	1	1	CP A, @P2:1+
		0110000		A, <hwregs>	1	1	CP A, STACK
		0110100		A, <limm>	2	2	CP A, #%FFCF
DEC	Decrement	1001000	DEC [<cc>,] <dest>	<cc>, A	1	1	DEC NZ, A
		1001000		A	1	1	DEC A
INC	Increment	1001000	INC [<cc>,] <dest>	<cc>, A	1	1	INC PL, A
		1001000		A	1	1	INC A
JP	Jump	0100110	JP [<cc>,] <address>	<cc>, <direct>	2	2	JP NIE, Label
		0100110		<direct>	2	2	JP Label

Table 68 Instruction Op Codes (Continued)

Inst	Description	Op Code	Synopsis	Operands	Words	Cycles	Examples
LD	Load Destination with Source	0000000	LD <dest>,<src>	A, <hwregs>	1	1	LD A, X
		0000001		A, <dregs>	1	1	LD A, D0:0
		0001001		A, <pregs>	1	1	LD A, P0:1
		0000001		A, regind>	1	1	LD A, @P1:1
		0000101		A, <memind>, <memind>	1	3	LD A, @D0:0
		0000011		A, <direct>	1	1	LD A, 124
		0000111		<direct>, A	1	1	LD 124, A
		0000100		<dregs>, <hwregs>	1	1	LD D0:0, EXT7
		0001100		<pregs>, <simm>	1	1	LD P1:1, #%FA
		0001010		<pregs>, <hwregs>	1	1	LD P1:1, EXT1
		0000110		<regind>, <limm>	1	1	LD @P1:1, #%1234
		0000010		<regind>, <hwregs>	1	1	LD @PM+, X
		0001001		<hwregs>, <pregs>	1	1	LD Y, P0:0
		0000001		<hwregs>, <dregs>	1	1	LD SR, D0:0
		0000100		<hwregs>, <limm>	2	2	LD PC, #%1234
		0100101		<hwregs>, <accind>	1	3	LD X, @A
		0000101		<hwregs>, <memind>	1	3	LD Y, D0:0
		0000001		<hwregs>, <regind>	1	1	LD A, @P0:0-LOOP
		0000000		<hwregs>, <hwregs>	1	1	LD X, EXT6
				When <dest> is <hwregs>, <dest> cannot be P. When<dest> is <hwregs> and <src> is <hwregs>, <dest> cannot be EXTn if <src> is EXTn, <dest> cannot be X if <src> is X, <dest> cannot be SR if <src> is SR. When <src> is <accind> <dest> cannot be A.			

Table 68 Instruction Op Codes (Continued)

Inst	Description	Op Code	Synopsis	Operands	Words	Cycles	Examples
NEG	Negate	1001000	NEG <cc>, A	<cc>, A	1	1	NEG NZ,A
		1001000		A	1	1	NEG A
NOP	No Operation	0000000	NOP	None	1	1	NOP
OR	Bitwise OR		OR <dest>, <src>				
		1101001		A, <pregs>	1	1	OR A, P0:1
		1100001		A, <dregs>	1	1	OR A, D0:1
		1100100		A, <limm>	2	2	OR A,#%202
		1100101		A, <memind>	1	3	OR A, @@P2:1+
		1100011		A, <direct>	1	1	OR A, %2C
		1100001		A, <regind>	1	1	OR A, @P1:0-LOOP
		1100000		A, <hwregs>	1	1	OR A, EXT6
POP	Pop a Value from the Stack	0001010	POP <dest>	<pregs>	1	1	POP P0:0
		0000100		<regs>	1	1	POP D0:1
		0000010		<regind>	1	1	POP @P0:0
		0000000		<hwregs>	1	1	POP A
PUSH	Push a Value onto the Stack	0001001	PUSH <src>, <pregs>		1	1	PUSH P0:0
		0000001		<dregs>	1	1	PUSH D0:1
		0000001		<regind>	1	1	PUSH @P0:0
		0000000		<hwregs>	1	1	PUSH BU.S
		0000100		<limm>	2	2	PUSH #%2345
		0100101		<accind>	1	3	PUSH @A
		0000101		<memind>	1	3	PUSH @@P0:0
RET	Return from Subroutine	0000000	RET	None	1	2	RET
RL	Rotate Left	1001000	RL <cc>, A	<cc>, A	1	1	RL NZ, A
		1001000		A	1	1	RL A
RR	Rotate Right	1001000	RR <cc>, A	<cc>, A	1	1	RR C, A
		1001000		A	1	1	RR A

Table 68 Instruction Op Codes (Continued)

Inst	Description	Op Code	Synopsis	Operands	Words	Cycles	Examples
SCF	Set C Flag	1001010	SCF	None	1	1	SCF
SIEF	Set IE Flag	1001010	SIEF	None	1	1	SIEF
SLL	Shift Left Logical	1001000	SLL	[<cc>,] A	1	1	SLL NZ, A
		1001000		A	1	1	SLL A
SOPF	Set OP Flag	1001010	SOPF	None	1	1	SOPF
SRA	Shift Right	1001000	SRA <cc>, A	<cc>, A	1	1	SRA NZ, A
	Arithmetic	1001000		A	1	1	SRA A
SUB	Subtract	0011001	SUB <dest>, <src>	A, <pregs>	1	1	SUB A, P1:1
		0010011		A, <dregs>	1	1	SUB A, D0:1
		0010100		A, <limm>	2	2	SUB A, #%2C2C
		0010101		A, <memind>	1	3	SUB A, @D0:1
		0010011		A, <direct>	1	1	SUB A, %15
		0010001		A, <regind>	1	1	SUB A, @P2:0-LOOP
		0010000		A, <hwregs>	1	1	SUB A, STACK
XOR	Bitwise Exclusive OR		XOR <dest>, <src>				
		1111001		A, <pregs>	1	1	XOR A, P2:0
		1110001		A, <dregs>	1	1	XOR A, D0:1
		1110100		A, <limm>	2	2	XOR A, #%3933
		1110001		A, <memind>	1	3	XOR A, @P2:1+
		1110011		A, <direct>	1	1	XOR A, %2F
		1110001		A, <regind>	1	1	XOR A, @P2:0
		1110000		A, <hwregs>	1	1	XOR A, BUS

Instruction Descriptions

The DSP instruction set consists of 30 basic instructions, optimized for high-code density and reduced execution time. Single-cycle instruction execution is possible because of the pipeline and other architectural features. Table 69 contains a description for each instruction.

Table 69 Instruction Descriptions

Mnemonic	Mnemonic Expansion
Instruction Operands	Lists the types of addressing methods for a specific instruction (ABS A or ABS <cc>, A).
Instruction Format	Displays instruction format for register indirect addressing.
Operation	Displays operation sequence.
Affected Flags	Lists flags affected by operation.
Description	Describes the instruction operation.
Examples	A simple example displays the instruction operation and how the registers are affected. The example includes initialization, instruction and result. Also includes cycles and instruction length.

Note: Each Assembly Instruction includes an example for each addressing mode available for the specific instruction.

The mnemonics listed in Table 70 are used in the instruction format.

Table 70 Instruction Format Mnemonics

Mnemonic	Description	Mnemonic	Description
A	Address	dest	Destination Value
am	Accumulator Modification	fm	Flag Modification
b	RAM Bank	op	Op Code
cc	Condition Code	rp	Register Pointer
const exp	Constant Expression	s	Source Address
d	Destination Address	src	Source Value

ABS ABSOLUTE VALUE ABS

Instruction Word:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	1	0	0	0	cc	cc	cc	cc	cc	0	1	1	1
Op Code							Condition Code					ACC Modification			

Syntax

```
ABS <cc>, A
ABS A
```

Operation

```
If ACC < 0 then -(ACC) -> ACC
```

Flags: N: Set if the accumulator has 800000H (see below).

Description

If the contents of the accumulator are determined to be less then 0 (a negative number), the absolute value of the accumulator is calculated (accumulator replaced by its two's complement value). Using the condition code provides an additional method to evaluate a status flag before the absolute value of the accumulator is calculated.

Note: If the accumulator contains 800000H, the ABS A instruction stores the value of the two's complement at address 800000H and sets the Overflow and Negative status bits. There is no overflow protection.

Example:

```
ABS A
```

Initialization: Accumulator contains FFEB00H
SR contains 0000H

Instruction: ABS A

Result: Accumulator contains 001500H
SR contains 1000H

This example uses one word of memory and executes in one machine cycle. Because the value in the accumulator is less then zero, the two's complement is performed and the result is placed in the accumulator ABS(FFEBH)=001500H. The carry bit is set.

Example:

```
ABS <cc>, A
```

Initialization: Accumulator contains `456400H`

Instruction: `ABS MI, A`

Result: Accumulator contains `456400H`

This example uses one word of memory and executes in one machine cycle. The condition code (negative bit) is not set because the accumulator value is positive; therefore, the instruction is not executed.

ADD ADDITION ADD

Instruction Word

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	0	0	1	b	0	0	0	0	s	s	s	s
Op Code							RAM Bank	Destination				Source			

Syntax

```
ADD     A,     <regind>
ADD      A,    <memind>
ADD     A,      <limm>
ADD     A,     <hwregs>
ADD     A,     <direct>
ADD     A,     <pregs>
ADD     A,     <dregs>
```

Operation

```
ACC + <source> -> ACC
```

Flags:
- C: Set if carry from the most significant bit is found.
- N: Set if result in the accumulator is negative.
- Z: Set if result is 0.
- OV: Set if addition exceeds upper (FFFFFH) or lower (800000H) limit of the accumulator.

Description

The addressed data memory operand is added to the accumulator. The result is loaded into the accumulator.

Note: The lower eight bits of the accumulator are unchanged while the add instruction is executed.

Note: Example

```
ADD A, <regind>
```

Initialization: Accumulator contains 123456H
P0:0 contains 4DH
RAM Bank1: 4DH contains 8746H

Instruction: `ADD A, @P0:0`

Result: A contains `997A56H`
@P0:0 contains `746H`

This example uses one word of memory and executes in one machine cycle. The pointer `P0:0` contains the RAM register location (`4DH`). The contents of RAM register `4DH` are added to the accumulator to obtain the sum (`874600H + 123456H = 997A56H`). The sum is contained in the accumulator and the pointer is left unchanged. The direct addressing equivalent would be `ADD A, %4D or ADD A, 77` (`4DH` = 77 decimal).

Example

`ADD A, <memind>`

Initialization: Accumulator contains `123400H`
P0:0 contains `21H`
RAM Bank0: `21H` contains `247AH`
ROM Address: 247AHcontains 0C12H

Instruction: `ADD A, @@P0:0`

Result: A contains `lE4600H`
P0:0 contains `21H`
RAM Bank0: `21H` contains `247BH`

This example uses one word of memory and executes in three machine cycles. The pointer `P0:0` contains the RAM register location (`21H`). The contents of this register have a ROM address. This address refers to the ROM data tplaced in the specified accumulator by an `AND` instruction `123400H + 0C1200H = 1E4600H`. When memory indirect addressing is used, the ROM address is automatically incremented. to provide a convenient method of accessing sequential data. Using `ADD A, @@P0:0+` performs the same operation and also increments the `P0:0` content to `22H`.

Example:

`ADD A, <limm>`

Initialization: Accumulator contains 123400H

Instruction: `ADD A, #%0C12`

Result: A contains `lE4600H`

This example uses two words of memory and executes in two machine cycles. The immediate operand 0C12H is added to the accumulator to obtain the sum 123400H + 0C1200H = 1E4600H.

Example

```
ADD A,<hwregs>
```

Initialization: Accumulator contains 23400H
Register X contains 0C12H

Instruction: ADD A, X

Result: A contains 1E4600H

This example uses one word of memory and executes in one machine cycle. The contents of register X are added to the accumulator to obtain the sum. 123400H + 0C1200H = 1E4600H. All hardware registers can transfer from <hwregs>.

Example

```
ADD A,<direct>
```

Initialization: Accumulator contains 123400H
RAM Bank0: F3H contains 0C12H

Instruction: ADD A,%F3

Result: A contains 1E4600H

This example uses one word of memory and executes in one machine cycle. Register F3H is added to the accumulator to obtain the sum. 123400H + 0C1200H = 1E4600H. An equivalent instruction is ADD A, 243 (F3H = 243 decimal).

Example

```
ADD A, <pregs>
```

Initialization: Accumulator contains 123400H
P0:0 contains 56H

Instruction: ADD A, P0:0

Result: Accumulator contains 128A00H

This example uses one word of memory and executes in one machine cycle. The contents of the pointer register P0:0 is added to the accumulator. 123400H + 005600H = 128A00H. The Pointer Register is connected to the lower 8 bits of the D-bus. The D-bus

is connected to the upper 16-bits of the P-bus. This causes the pointer register operand to become 005600H before being added to the accumulator.

Example

```
ADD A,<dregs>
```

Initialization: Accumulator contains 123400H
D0: 1 contains 8746H

Instruction: ADDA,D0:1

Result: A contains 997A00H
D0: 1 contains 8746H

This example uses one word of memory and executes in one machine cycle. The contents of the data pointer D0:1 are added to the accumulator. The sum is contained in the accumulator and the pointer is left unchanged. The data pointer contains 8746H.

AND BITWISE AND AND

Instruction Word

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	0	0	b	0	0	0	0	s	s	s	s
Op Code							RAM Bank	Destination				Source			

Syntax

```
AND A, <regind>
AND A, <memind>
AND A, <limm>
AND A, <hwregs>
AND A, <direct>
AND A, <pregs>
AND A, <dregs>
AND A, <simm>
```

Operation

```
<accumulator>. AND.<source> –> <accumulator>
```

Flags: N: Set if accumulator result is less than 0.
Z: Set If accumulator result is 0.

Description

Data is stored in the specified accumulator by an AND instruction. The lower eight bits of the accumulator are cleared when this instruction is executed.

Example

```
AND A, <regind>
```

Initialization: Accumulator contains 123456H
P0A contains 45H
RAM Bank1: 45H contains 8746H

Instruction: `AND A, @P0:1`

Result: Accumulator contains 020400H

This example uses one word of memory and executes in one machine cycle. The data in RAM Bank1, referenced by RAM pointer 0, is stored in the specified accumulator using an AND instruction 123456H.AND.874600H = 020400H.

Example

```
AND A, <memind>
```

Initialization: Accumulator contains 123400H
P0:0 contains45H
RAM Bank0: 45H contains 047AH
ROM Address: 047AH contains 8746H

Instruction: AND A, @@P0:0

Result: A contains 020400H
P0:0 contains 45H
RAM Bank0: 45H contains 247BH

This example uses one word of memory and executes in three machine cycles. The pointer P0:0 contains the RAM register location (45H). The contents of this register has a ROM address. This address refers to the ROM data that is placed in the specified accumulator by an AND instruction 123400H.AND.874600H = 020400H. With memory-indirect addressing, the ROM address is automatically incremented. This provides a convenient method to access sequential data. Using AND A, @@P0:0+ performs the same operation and also increments the P0:0 content to 46H.

Example

```
AND A, <limm>
```

Initialization: Accumulator contains 362400H

Instruction: AND A, #%1234

Result: Accumulator contains 122400H

This example uses two words of memory and executes in two machine cycles. The immediate operand 1234H and an accumulator address are processed with an AND instruction to produce the result, 362400H.AND.123400H = 122400H.

Example

```
AND A, <simm>
```

Initialization: Accumulator contains 123456H

Instruction: `AND A, #%1F`

Result: Accumulator contains `001400H`

This example uses one word of memory and executes in one machine cycle. The data in the immediate field and the contents of the accumulator are processed with an AND instruction. `123456H.AND.001F00H = 001400H`.

Example

```
AND A, <hwregs>
```

Initialization: Accumulator contains `123400H`
Register X contains `OC12H`

Instruction: `AND A, X`

Result: A contains `001000H`

This example uses one word of memory and executes in one machine cycle. Use an `AND` instruction to send the contents of Register `X` to the accumulator to obtain the result `123400H.AND.0C1200H = 001000H`. All hardware registercan tsransfer from <`hwregs`>.

Example

```
AND A, <direct>
```

Initialization: Accumulator contains `123400H`
RAM Bank0: F3H contains `0C12H`

Instruction: `AND A, %F3`

Result: A contains `001000H`

This example uses one word of memory and executes in one machine cycle. Use an `AND` instruction to send Register `F3H` to the accumulator to obtain the result `123400H AND OC1200H = 001000H`. An equivalent instruction is `AND A, 243` (`F3H` = 243 decimal).

Example

```
AND A, <pregs>
```

Initialization: Accumulator contains `123400H`
P0:0 contains `56H`

Instruction: `AND A, P0:0`

Result: Accumulator contains 001400H

This example uses one word of memory and executes in one machine cycle. Use an AND instruction to send the contents of the pointer register P0:0 to the accumulator 123400H.AND.005600H = 001400H. The Pointer Register is connected to the lower 8 bits of the D-bus. The D-bus is connected to the upper 16 bits of the P-bus. This action changes the pointer register operand to 005600H before being added to the accumulator.

Example

```
AND A, <dregs>
```

Initialization: Accumulator contains 123400H
D0:1 contains 2645H

Instruction: AND A, 00:0

Result: Accumulator contains 020400H

This example uses one word of memory and executes in one machine cycle. The data register, D0:0, references the operand 2645H. Use an AND instruction to send this data register to the accumulator to produce the result 123400H.AND.2645H = 020400H.

CALL SUBROUTINE CALL CALL

Instruction Word

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	b	cc	cc	cc	cc	s	s	s	s
Op Code								Condition Code				Not Used			

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a
16-Bit Address															

Syntax

```
CALL <direct>
```

Operation

```
PC + 2 –> STACK
16-Bit Address –> PC
```

Flags: None

Description

The current Program Counter (PC) register content is incremented by two and placed on the stack. The address of the specified label in the CALL instruction is then placed in the PC register. The jump is made to the appropriate subroutine via the PC. The condition code option is used if CALL is executed.

Example

```
CALL <direct>                    Cycles: 2
                                          Words: 2
```

Initialization: PC contains 1FFBH
FFT2 subroutine address contains F234H
Stack Level 0 contains 0025H

Instruction: CALL FFT2

Result: PC contains F234H
Stack Level 0 contains 1FFDH
Stack Level 1 contains 0025H

This example uses two words of memory and executes in two machine cycles. The call to the subroutine FFT2 places PC+2 (1 FFDH) on the stack. All information currently on the stack is pushed up the stack. The subroutine address is then placed in the PC register. The processor executes the next instruction addressed by the PC, the FFT2 subroutine.

Example

```
CALL <direct>
```

Initialization: PC contains 1FFBH
FFT2 subroutine address contains F234H
Stack Level 0 contains 0025H

Instruction: CALL FFT2

Result: PC containsF234H
Stack Level 0 contains 1FFDH
Stack Level 1 contains 0025H

This example uses two words of memory and executes in two machine cycles. The call to the subroutine FFT2 places PC+2 (1 FFDH) on the stack. All information currently on the stack is pushed up the stack. The subroutine address is then placed in the PC register. The processor executes the next instruction addressed by the PC, the FFT2 subroutine.

CCF CLEAR CARRY FLAG CCF

Instruction Word

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	1	0	1	0	0	0	0	0	0	0	0	1	0
Op Code							Condition Code					Flag Modification			

Syntax

```
CCF
```

Operation

```
Zero –> Carry Bit
```

Flags: C: Set to 0.

Description

The Clear Carry Flag instruction resets the carry flag with a 0.

Example

```
CCF
```

Initialization: SR contains 3000H

Instruction: CCF

Result: SR contains 2000H
C contains 0

This example uses one word of memory and executes in one machine cycle.

CIEF CLEAR INTERRUPT ENABLE FLAG CIEF

Instruction Word

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	1	0	1	0	0	0	0	0	0	0	1	0	0
Op Code							Condition Code					Flag Modification			

Syntax:

```
CIEF
```

Operation

```
Zero –> IE bit
```

Flags: `IE`: Set to `0`.

Description

The Clear Interrupt Enable Flag instruction sets the IE flag to `0`.

Example

```
CIEF
```

Initialization: SR contains 3080H

Instruction: `CIEF`

Result: SR contains `3000H`
`IE contains 0`

This example uses one word of memory and executes in one machine cycle.

COPFCLEAR OVERFLOW PROTECTION FLAG COPCLEAR

Instruction Word

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	1	0	1	0	0	0	0	0	0	1	0	0	0
Op Code							Condition Code					Flag Modification			

Syntax

```
COPF
```

Operation

```
Zero –> OP bit
```

Flags: P: Set to 0.

The Clear Overflow Protection Flag instruction resets the OP flag to 0.

Example:

```
COPF
```

Initialization: SR contains 0100H

Instruction: COPF

Result: SR contains 0000H
OP contains 0

This example uses one word of memory and executes in one machine cycle.

CP COMPARISON CP

Instruction Word

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	0	1	0	b	0	0	0	0	s	s	s	s
Op Code							RAM Bank	Destination				Source			

Syntax

```
CP          A,        <regind>
CP          A,        <memind>
CP          A,        <limm>
CP          A,        <hwregs>
CP          A,        <direct>
CP          A,        <pregs>
CP          A,        <dregs>
CP          A,        <simm>
```

Operation

```
A - <source> –> set appropriate status bits
```

Flags:
- C: Set if carry is required for operation.
- Z: Set if operands are equal.
- OV: Set if operation exceeds the low (800000H) or high limit (7FFFFFH) of accumulator.
- N: Set if result is negative.

Description

The contents of the register specified in the instruction are compared to the contents of the accumulator in 16-bit mode. The register specified is subtracted from the accumulator and the appropriate flags are set. Because the registers are 16-bit, a comparison with the 24-bit accumulator requires that the lower eight bits of the accumulator be filled with zeros for accurate comparisons. The instruction does not affect the contents of the accumulator except when the overflow protection bit is set and an overflow occurs after the compare is

executed. The accumulator updates with the appropriate low (800000H) or high (7FFFFFH) limit.

Example

```
CP A, <regind>
```

Initialization: A contains 7A2500H
P2:1 contains A4H
RAM Bank1: A4H contains 5463H

Instruction: CP A, @P2:1

Result: A contains 7A2500H
SR contains 1000H

This example uses one word of memory and executes in one machine cycle. The operand referenced by P2:1 is subtracted from the accumulator. 7A2500H - 546300H = 25C200H. Because the comparison does not yield a 0, Bit 0 is not set. The content of the P2:1 register is appended with eight additional 0 bits. For consistent comparisons, the accumulator must contain zeros in the lower eight bits. Also note that the accumulator is unaffected by the operation.

Example

```
CP A, <memind>
```

Initialization: A contains 7A2500H
P2:1 contains A4H
RAM Bank1: A4H contains 5463H
ROM Address: 5463H contains OC12H

Instruction: CP A, @@P2:1

Result: A contains 7A2500H
P2:1 contains A4H
RAM Bank1: A4H contains 5464H
SR contains 1000H

This example uses one word of memory and executes in three machine cycles.The pointer P2:1 contains the RAM register location (A4H). The contents of this register have a ROM address. This address refers to the ROM data compared to the accumulator 7A2500H - OC1200H = 6E1300H. Because the comparison does not yield a 0, Bit 0 is not set. With memory indirect addressing, the ROM address is automatically incremented. Using CP A,@@P2:1 + performs the same operation and also increments P2:1 content to A5H.

Example

```
CP A, <limm>
```

Initialization: A contains 7A2500H

Instruction: `CP A, #%7A25`

Result: A contains 7A2500H
SR contains 3000H

This example uses two words of memory and executes in two machine cycles. The immediate operand is compared to the accumulator. Because they are equal, the Zero Flag is set.

Example

```
CP A, <hwregs>
```

Initialization: A contains 7A2500H
SR contains 0000H
BUS contains 7A25H

Instruction: `CP A, BUS`

Result: A contains 7A2500H
SR contains 3000H

This example uses one word of memory and executes in one machine cycle. The <hwreg> operand is subtracted from the accumulator. Because the two operands are equal, the zero-status bit is set High. <hwregs> can be compared from all hardware registers.

Example

```
CP A, <direct>
```

Initialization: Accumulator contains 7A2500H
RAM Bank0 F3H contains 5463H

Instruction: `CP A, %F3`

Result: A contains 7A2500H
SR contains 1000H

This example uses one word of memory and executes in one machine cycle. Register F3H is compared to the accumulator. 7A2500H - 546300H = 25C200H. An equivalent instruction is CP A,243 (173H = 243 decimal).

Example

```
CP A, <pregs>
```

Initialization: Accumulator contains 123400H
P0:0 contains 56H

Instruction: CP A, P0:0

Result: Accumulator contains 123400H
SR contains 1000H

This example uses one word of memory and executes in one machine cycle. The contents of the pointer register P0:0 are compared to the accumulator. 123400H - 005600H = 11DE00H The Pointer Register is connected to the lower 8 bits of the D-bus. The D-bus is connected to the upper 16 bits of the P-bus. This action changes the pointer register operand to 005600H before being compared to the accumulator.

Example

```
CP A, <dregs>
```

Initialization: A contains 7A2500H
D2:1 contains 5463H

Instruction: CP A, D2:1

Result A contains 7A2500H
SR contains 1000H

This example uses one word of memory and executes in one machine cycle. The contents of the data pointer D2:1 are compared to the accumulator. 7A2500H - 546300H = 25C200H.

DEC DECREMENT DEC

Instruction Word

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	1	0	0	0	cc	cc	cc	cc	cc	0	1	0	1
Op Code							Condition Code					ACC Modification			

Syntax

```
DEC A
DEC <cc>, A
```

Operation

```
ACC - 1 –> ACC
```

Flags:
- C: Set if carry is required for operation
- Z: Set if result is 0.
- N: Set if decrement results in a value less then 0.
- OV: Set if upper (7FFFFFH) or lower (800000H) limits are exceeded.

Description

The accumulator decrements by 1. A condition code can be used to test for a specific condition before decrementing.

Example

```
DEC A
```

Initialization: A contains 7A2500H

Instruction: DEC A

Result: A contains 7A24FFH

This example uses one word of memory and executes in one machine cycle. The value in the accumulator decrements by 1.

Example

```
DEC <cc>, A
```

Initialization: A contains `7A2500H`

Instruction: `DEC MI, A`

Result: A contains `7A2500H`

This example uses one word of memory and executes in one machine cycle. Because the accumulator is not negative, the decrement instruction is not executed.

INC INCREMENT INC

Instruction Word

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	1	0	0	0	cc	cc	cc	cc	cc	0	1	0	0
Op Code							Condition Code					ACC Modification			

Syntax

```
INC <cc>, A
INC A
```

Operation

```
ACC + 1 –> ACC
```

Flags:
- C: Set if carry is required for operation.
- Z: Set if result is 0.
- N: Set if results in a value less then 0.
- OV: Set if upper (7FFFFFH) or lower (800000H) limits are exceeded.

Description

The Increment instruction adds one to the accumulator. A condition code can be used to test for a specific condition for an increment to occur.

Example

```
INC <cc>, A
```

Initialization: A contains 7A2500H

Instruction: INC MI, A

Result: A contains 7A2500H

This example uses one word of memory and executes in one machine cycle. Because the accumulator is not negative, the increment instruction is not executed.

Example

```
INC A
```

Initialization: A contains `7A2500H`

Instruction: `INC A`

Result: A contains `7A2501H`

This example uses one word of memory and executes in one machine cycle. The value in the accumulator is incremented by 1.

JP JUMP JP

Instruction Word

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	1	0	cc	cc	cc	cc	cc	0	0	0	0
Op Code							Condition Code					Not Used			

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a
16-Bit Address															

Syntax

```
JP          <cc>,       <direct>
JP          <direct>
```

Operation

```
16-Bit address —> PC
```

Flags: None

Description

The instruction places the address of the referenced ROM location in the Program Counter (PC). Because the processor obtains its next instruction address from the PC, the processor jumps to the appropriate location. A condition code can be used to test for a specific condition for a JUMP to occur.

Example

```
JP <cc>, <direct>
```

Initialization: Routine 1 address contains 1455H
PC contains 1343H
User 0 input contains 1

Instruction: `JP NUO, Routine 1`

Result: PC contains 1343H

This example uses two words of memory and executes in two machine cycles. Because the User 0 input is set High, the condition code is not met. Therefore, the JUMP instruction does not execute. The User 0 input is used to control the flow of the software.

Example

```
JP <direct>
```

Initialization: Routine 1 address contains 1455H
PC contains 1343H

Instruction: JP Routine 1

Result: PC contains 1455H

This example uses two words of memory and executes in two machine cycles. The value in the program counter is replaced by the Routine 1 address (1455H).

LD LOAD LD

Instruction Word

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	1	b	0	0	1	1	s	s	s	s
Op Code							RAM Bank	Destination				Source			

Syntax

```
LD      A,              <pregs>
LD      <direct>,       A
LD      <hwregs>,       <dregs>
LD      A,              <dregs>
LD      <dregs>,        <hwregs>
LD      <hwregs>,       <limm>
LD      A,              <memind>
LD      <pregs>,        <simm>
LD      <hwregs>,       <accind>
LD      A,              <direct>
LD      <pregs>,        <hwreg>
LD      <hwregs>,       <memind>
LD      A,              <regind>
LD      <regind>,       <limm>
LD      <hwregs>,       <regind>
LD      A,              <hwregs>
LD      <hwregs>,       < pregs>
LD      <hwregs>,       <hwregs>
```

Operation

```
<source> –> <destination>
```

Flags: None

Description

The LOAD command provides the ability to transfer data to several different locations in the processor including hardware registers, accumulator, stack, pointers and memory. All transfers across the various internal buses are transparent to the user.

Notes:

1. A load using the X or Y register provides an automatic multiply operation. This means the operand can be obtained from any register location.
2. The P register is a read-only register, therefore the destination of the load cannot be the P register.
3. LD EXTN, EXTN is not allowed.
4. A LOAD instruction to the accumulator clears the lower eight bits of the 24-bit accumulator.

Example1

```
LD A, <regind>
```

Initialization: A contains 7A2500H
P2:1 contains A4H
RAM Bank1: A4H contains 5463H

Instruction: `LD A, @P2:1`

Result: A contains 546300H

This example uses one word of memory and executes in one machine cycle. Indirect addressing through the pointer registers provides access to RAM data. The data in RAM Bank 1, register A4 is transferred to the accumulator. The contents of the P2:1 register are appended with eight additional 0 bits. This is added to verify a correct arithmetic comparison.

Example2

```
LD A, <memind>
```

Initialization: Accumulator contains 123400H
P0:0 contains 21H

RAM Bank0: 21H contains 247AH
ROM Address: 247AHequals OC1 2H

Instruction: LD A, @@P0:0

Result: A contains OC1 200H
P0:0 contains 21H
RAM Bank0: 21H contains 247BH

This example uses one word of memory and executes in three machine cycles. The pointer P0:0 contains the RAM register location (21H). The contents of this register have a ROM address. This address refers to the ROM data that loads to the accumulator. When memory indirect addressing is used, the ROM address is automatically incremented. Using LD A, @@P0:0+ has the same result, also incrementing the P0:0 content to 22H.

Example3

```
LD A, <limm>                    Cycles: 2
                                Words: 2
```

Initialization: Accumulator contains 123400H

Instruction: LD A, #%2474

Result: A contains 247400H

This example uses two words of memory and executes in two machine cycles. The immediate operand 2474H loads to the accumulator.

Example4

```
LD A, <hwregs>
```

Initialization: Accumulator contains 123400H
Register X contains OC12H

Instruction: LD A, X

Result: A contains OC1200H

This example uses one word of memory and executes in one machine cycle. The contents of Register X are loaded to the accumulator. All hardware registers can transfer from <hwregs>.

Example5

```
LD A, <direct>
```

Initialization: Accumulator contains 123400H
RAM Bank0 F3H contains OC12H

Instruction: LD A, %F3

Result: A contains OC1200H

This example uses one word of memory and executes in one machine cycle. Register F3H is loaded to the accumulator. An equivalent instruction is LD A,243 (F3H = 243 decimal).

Example6

```
LD A, <dregs>
```

Initialization: Accumulator contains123400H
D0: 1 contains 8746H

Instruction: LD A, D0:1

Result: A contains 874600H
D0: 1 contains 8746H

This example uses one word of memory and executes in one machine cycle. The contents of the data pointer D0:1 load to the accumulator.

Example7

```
LD A, <pregs>
```

Initialization: Accumulator contains 123400H
P0:0 contains 56H

Instruction: LD A, P0:0

Result: Accumulator contains 005600H

This example uses one word of memory and executes in one machine cycle. The contents of the pointer register P0:0 are loaded to the accumulator. The Pointer Register is connected to the lower 8 bits of the D-bus. The D-bus is connected to the upper 16 bits of the P-bus. This operation causes the pointer register operand to become 005600H before being loaded into the accumulator.

Example8

```
LD <direct>, A
```

Initialization: Accumulator contains 123400H
RAM Bank0: 3CH contains 5678H

Instruction: `LD %3C, A`

Result: Accumulator contains `123400H`
RAM Bank0: 3CH contains `1234H`

This example uses one word of memory and executes in one machine cycle. The current value in the accumulator is loaded to the register addressed by the instruction (`3CH`). An equivalent instruction is `LD 60, A`. (`3CH` = 60 decimal).

Example9

```
LD <pregs>, <simm>
```

Initialization: P2:0 contains `2FH`
RAM Bank0: 3FH contains `254645H`

Instruction: `LD P2:0, #%3F`

Result: P2:0 contains `3FH`
RAM Bank0: `3FH` contains `254645H`

This example uses one word of memory and executes in one machine cycle. The immediate data (`3FH`) is loaded into the pointer register `P2:0`. This action provides a convenient method for initializing pointer registers.

Example10

```
LD <pregs>, <hwregs>
```

Initialization: P0: 1 contains `2FH`
Y contains `2376H`

Instruction: `LD P0:1, Y`

Result: P0: 1 contains `76H`
Y contains `2376H`

This example uses one word of memory and executes in one machine cycle. The lower 8-bits of the `Y` register are transferred to the pointer register `P0:1`. All hardware registers can transfer from `<hwregs>`.

Example11

```
LD <regind>, <limm>
```

Initialization: P0:1 contains `2FH`
RAM Bank0: 2FH contains `2376H`

Instruction: `LD @P0:1, #%35B8`

Result: P0:1 contains 2FH
RAM Bank1: 2FH contains 35B8H

This example uses two words of memory and executes in two machine cycles. The immediate operand 35B8H is transferred to the Register 2FH of RAM Bank1.

Example12

```
LD <hwregs>, <pregs>
```

Initialization: P2:0 contains 2FH
X Register contains 8B87H

Instruction: `LD X,P2:0`

Result: P2:0 contains 2FH
X Register contains 002FH

This example uses one word of memory and executes in one machine cycle. The contents of the P2:0 pointer (2FH) are loaded into the X register. Transfer to <hwreg> is possible to all hardware registers except the read-only P register.

Example13

```
LD <hwregs>, <dregs>
```

Initialization: D2:0 contains 3C87H
Accumulator contains 8BB722H

Instruction: `LD A,D2:0`

Result: D2:0 contains 3C87H
Accumulator contains 3C8700H

This example uses one word of memory and executes in one machine cycle. The contents of the D2:0 pointer (3C87H) are loaded into the accumulator. Transfer to <hwregs> is possible to all hardware registers except the read-only P register.

Example14

```
LD <hwregs>, <limm>
```

Initialization: Stack0 contains 8B2FH
Stack1 contains 0000H

Instruction: `LD Stack, #%35B8`

Result: Stack0 contains 35B8H
Stack1 contains 8B2FH

This example uses two words of memory and executes in two machine cycles. The immediate data is pushed onto the stack as previous stack data is pushed up the stack. Transfer to <hwregs> is possible to all hardware registers except the read-only P register.

Example15

```
LD <hwregs>, <accind>
```

Initialization: EXT7 Register contains 8B87H
Accumulator contains 77B6H
ROM 77B6H contains 387DH

Instruction: LD EXT7, @A

Result: EXT7 Register contains 387DH

Accumulator contains 77B6H

This example uses one word of memory and executes in one machine cycle. The contents of the ROM Register 77B6H (387DH) are loaded into External Register 7. Transfer to <hwregs> is possible to all hardware registers except the read-only P register and the accumulator register.

Example16

```
LD <hwregs>, <memind>
```

Initialization: Y Register contains 1234H
P0:0 contains 21H
RAM Bank0: 21H contains 247AH
ROM Address: 247AH contains 0C12H

Instruction: LID Y, @@P0:0

Result: Y Register contains 0C12H
P0:0 contains 21H
RAM Bank0: 21H contains 247BH

This example uses one word of memory and executes in three machine cycles. The pointer P0:0 contains the RAM register location (21H). The contents of this register have a ROM addresswhich refers to the ROM data that loads to the Y register. Transfer to <hwregs> is possible to all hardware registers except the read-only P register. When memory indirect addressing is used the ROM address is automatically incremented. Using LD A,@@P0:0+ performs the same operation and also increments the P0:0 content to 22H.

Example17

```
LD <hwregs>, <regind>
```

Initialization: X Register contains 7A25H
P21 contains A4H
RAM Bank1: A4H contains 5463H

Instruction: `LD X, @P2:1`

Result: X Register contains 5463H

This example uses one word of memory and executes in one machine cycle. Indirect addressing through the pointer registers provides access to RAM data. The data in RAM bank 1, register A4 is transferred to the X register. Transfer to <hwreg> is possible to all hardware registers except the read-only P register.

Example18

```
LD <hwregs>, <hwregs>
```

Initialization: X Registe contains 7A25H
EXT5 Register contains 789AH

Instruction: `LD X, EXT5`

Result: X Register contains 789AH

EXT5 Register contains 789AH

This example uses one word of memory and executes in one machine cycle. The EXT5 Register contents are transferred to the X register. Transfer to <hwregs> is possible to all hardware registers except the read-only P register.

NEG NEGATE NEG

Instruction Word

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	1	0	0	0	cc	cc	cc	cc	cc	0	1	1	0
Op Code							Condition Code					ACC Modification			

Syntax

```
NEG A
NEG <cc>, A
```

Operation

```
-ACC —> ACC
```

Flags: `N` Set if result is a negative number.

Two special cases are:

1. If ACC contains `000000` after execution, then `N` and `0V` are cleared, and `Z` and `C` are set
2. If ACC contains `800000` after execution, then `N` and `0V` are set; and `Z` and `C` are cleared.

   The accumulator is replaced with a negative of the current value. To achieve this state, the two's complement is performed.

Example1

```
NEG A
```

Initialization: A contains `003654H`

Instruction: `NEG A`

Result: A contains `FFC9ACH`

This example uses one word of memory and executes in one machine cycle.

Example2

```
NEG <cc>, A
```

Initialization: A contains `000111H`
Carry bit contains `1`

Instruction: `NEG C, A`

Result: A contains `FFFEEFH`

This example uses one word of memory and executes in one machine cycle.

NOP NO OPERATION NOP

Instruction Word

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Syntax

```
NOP
```

Operation

```
PC+ 1—> PC
```

Flags: None

Description

The NOP instruction causes the processor to continue operation for one cycle without affecting previous registers and I/0.

OR BITWISE OR OR

Instruction Word

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	0	0	1	b	0	0	0	0	s	s	s	s
Op Code							RAM Bank	Destination				Source			

Syntax

```
OR A, <regind>
OR A, <memind>
OR A, <limm>
OR A, <hwregs>
OR A, <direct>
OR A, <pregs>
OR A, <dregs>
OR A, <simm>
```

Operation

```
ACC .OR. source –> ACC
```

Flags: N Set If result in accumulator is negative.
Z Set If result is 0.

Description

The accumulator performs an OR instruction on the contents of the specified register. The upper 16 bits of the accumulator are used. The result is placed in the accumulator. The OR instruction is frequently used to compare specific bits to assist in program control.

Note: The lower eight bits of the accumulator are unchanged after execution of the OR instruction.

Example

```
OR A, <regind>
```

Initialization: Accumulator contains 3264A0H
P0:0 contains E2H
RAM Bank0: E2H contains 1126H

Instruction: OR A, @P0:0

Result: A contains 336600H

This example uses one word of memory and executes in one machine cycle. Use an OR instruction to reference the operand P0:0 with the upper 16 bits of the accumulator. The result is stored in the accumulator. 3264A0H OR. 1126A0H = 3366A0H.

Example2

```
OR A,<memind>
```

Initialization: A contains 3264A0H
P2:11 contains A4H
RAM Bank 1: A4H contains 5463H
ROM Address: 5463H contains 1126H

Instruction: OR A, @@P2:1

Result A contains 3366A0H
P2:1 contains A4H
RAM Bank0: A4H contains 5464H
SR contains 0000H

This example uses one word of memory and executes in three machine cycles. The pointer P2:1 contains the RAM register location (A4H). The contents of this register have a ROM address. This address refers to the ROM data that is compared to the accumulator. 3264A0H.OR.112600H = 3366A0H. With memory indirect addressing , the ROM address is automatically incremented. Using ORA,@@P0:0+ performs the same operation and also increments the P0:0 content to A5H.

Example3

```
OR A, <limm>
```

Initialization: A contains 3264A0H

Instruction: OR A, #%1126

Result: A contains 3366A0H
SR contains 0000H

This example uses two words of memory and executes in two machine cycles. The accumulator performs an OR instruction on the immediate data.

POP POP VALUE FROM STACK POP

Instruction Word

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
op	op	op	op	op	op	op	b	d	d	d	d	s	s	s	s
Op Code							RAM Bank	Destination				Source			

Syntax:

```
POP <pregs>
POP <dregs>
POP <regind>
POP <hwregs>
```

Operation

```
STACK 0 –> <destination>
Stack n –> Stack N-1
```

Flags: None

Description

The current value of the stack is copied to the specified register. Because the stack is a last-in, first-out (LIFO) hard-wired architecture, the copy and shift of data remaining in the stack are all performed in a single cycle.

The POP instruction provides the ability to control information sent to the stack, making it possible to expand the stack using software.

Example1

```
POP <pregs>
```

Initialization: Stack 1 contains 0426H
Stack 0 contains 0C06H
P0:0 contains 24H

Instruction: `POP P0:0`

Result Stack 0 contains 0426H
P0:0 contains 06H

This example uses one word of memory and executes in one machine cycle. The destination of Stack 0 (item on top of stack) is P0:0. The 8-LSBs of the data in stack 0 are loaded into P0:0. At transfer, Stack 1 is automatically moved to Stack 0.

Example2

```
POP <dregs>
```

Initialization: Stack 1 contains 0426H
Stack 0 contains 0C06H
D0:0 contains 7676H

Instruction: POP D0:0

Result: Stack 0 contains 0426H
D0:0 contains 0C06H

This example uses one word of memory and executes in one machine cycle. The destination of the Stack 0 (item on top of stack) is given by D0:0. Upon transfer, Stack 1 is automatically moved to Stack 0.

Example3

```
POP <regind>
```

Initialization: Stack 1 contains 0426H
Stack 0 contains 0C06H
P0:0 contains 24H
RAM Bank0: 24H contains 42A4H

Instruction: POP @P0:0

Result Stack 0 contains 0426H

RAM Bank0: 24H contains 0C06H

This example uses one word of memory and executes in one machine cycle. The destination of the Stack 0 (item on top of stack) is given by P0:0. 24H is the register location in RAM Bank0 to which the stack item is transferred. At transfer, Stack 1 is automatically moved to Stack 0.

Example4

```
POP <hwregs>
```

Initialization: Stack 1 contains `0426H`
Stack 0 contains `0C06H`
X Register contains 089CH

Instruction: `POP X`

Result: Stack 0 contains `0426H`
X Register contains `0C06H`

This example uses one word of memory and executes in one machine cycle. The destination of the Stack 0 (item on top of stack) is given by the X Register. At transfer, Stack 1 is automatically moved to Stack 0. Transfer to <`hwregs`> is possible to all hardware registers except the read-only P register.

PUSH PUSH VALUE ONTO STACK PUSH

Instruction Word

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	0	0	1	b	0	0	0	0	s	s	s	s
Op Code							RAM Bank	Destination				Source			

Syntax

```
PUSH <pregs>
PUSH <dregs>
PUSH <memind>
PUSH <accind>
PUSH <regind>
PUSH <hwregs>
PUSH <direct>
PUSH <limm>
```

Operation

```
<source> –> Stack
Stack n –> Stack n+ 1
```

Flags: None

Description

The contents of the specified register are placed on the stack. Because the stack is a last-in, first-out (LIFO) hard-wired architecture, the placement and shifting of current stack data is performed in a single cycle.

The PUSH instruction provides the ability to control information sent to the stack, making it possible to expand the stack through software.

Example1

```
PUSH <pregs>
```

Initialization: Stack 0 contains 0C06H
P1:1 contains A4H

Instruction: PUSH P1:1

Result Stack 1 contains 0C06H
Stack 0 contains 00A4H

This example uses one word of memory and executes in one machine cycle. The pointer P1:1 contains the 8-bit value A4H. The 16-bit value, 00A4H, is pushed onto the stack. At transfer, Stack 0 is automatically moved to Stack 1.

Example2

```
PUSH <dregs>
```

Initialization: Stack 1 contains 0426H
Stack 0 contains 0C06H
D1:1 contains 42A4H

Instruction: PUSH D1:1

Result Stack 1 contains 0C06H
Stack 0 contains 42A4H

This example uses one word of memory and executes in one machine cycle. The pointer D1:1 is pushed onto the stack. At transfer, Stack 0 is automatically moved to Stack 1.

Example3

```
PUSH <memind>
```

Initialization: Stack 1 contains 0426H
Stack 0 contains 0C06H
RAM Bank0: 24H contains 42A4H
P1:1 contains A4H
RAM Bank1: A4H contains 5463H
ROM Address 5463H contains 1126H

Instruction: PUSH @@P1:1

Result: Stack 1 contains 0C06H
Stack 0 contains 1126H
RAM Bank1: A4H contains 5464H

This example uses one word of memory and executes in three machine cycles. When memory indirect addressing is used, the ROM address is automatically incremented.

Using OR A,@@P0:0+ performs the same operation and also increments the P0:0 content to A5H.

Example4

```
PUSH <accind>
```

Initialization: Stack 1 contains 0426H
Stack 0 contains 0C06H
Accumulator contains 42A4H
ROM address 42A4H contains 4C45H

Instruction: PUSH @A

Result Stack 1 contains 0C06H
Stack 0 contains 4C45H

This example uses one word of memory and executes in one machine cycle. Indirect addressing with the accumulator points to the ROM memory (42A4H) The data in this location is pushed onto the stack. At transfer, Stack 0 is automatically moved to Stack 1.

Example5

```
PUSH <regind>
```

Initialization: Stack 1 contains 0426H
Stack 0 contains 0C06H
P1:1 contains A4H
RAM Bank1: A4H contains 42A4H

Instruction: PUSH @P1:1

Result: Stack 1 contains 0C06H
Stack 0 contains 42A4H
RAM Bank1: A4H contains 42A4H

This example uses one word of memory and executes in one machine cycle. The pointer P1:1 contains the RAM register location (A4H). The data at this location is pushed onto the stack. At transfer, Stack 0 is automatically moved to Stack 1.

Example6

```
PUSH <hwregs>
```

Initialization: Stack 1 contains 0426H
Stack 0 contains 0C06H
X Register contains 42A4H

Instruction: `PUSH X`

Result: Stack 1 contains `0C06H`
Stack 0 contains `42A4H`

This example uses one word of memory and executes in one machine cycle. The data in the `X` register is pushed onto the stack. At transfer, `Stack 0` is automatically moved to `Stack 1`. Transfer from <`hwregs`> is possible from all hardware registers.

Example7

```
PUSH <direct>
```

Initialization: Stack 1 contains `0426H`
Stack 0 contains `0C06H`
RAM Bank0: `24H` contains `42A4H`

Instruction: `PUSH %24`

Result: Stack 1 contains `0C06H`
Stack 0 contains `42A4H`

This example uses one word of memory and executes in one machine cycle. The instruction (`24H`) provides the direct register address. The value contained in this register is pushed onto the stack (`42A4H`). At transfer, Stack 0 is automatically moved to Stack 1.

Example8

```
PUSH <limm>
```

Initialization: Stack 1 contains `0426H`
Stack 0 contains `0C06H`

Instruction: `PUSH #%5757`

Result: Stack 1 contains `0C06H`
Stack 0 contains `5757H`

This example uses two words of memory and executes in two machine cycles. The immediate operand `5757H` is pushed onto the stack. At transfer, Stack 0 is automatically moved to Stack `1`.

RET RETURN FROM SUBROUTINE RET

Instruction Word

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	1	1	0	0	1	0	1
Op Code							RAM Bank	Destination				Source			

Syntax

RET

Operation

```
Stack 0  –> PC
Stack n –>  Stack n-1
```

Flags: None

Description

The current stack information is popped from the stack and placed in the Program Counter (PC) register. The jump is made from the subroutine via the PC.

Example

RET

Initialization: Stack 1 contains 0624
Stack 0 contains 0401
PC contains 06DF

Instruction: RET

Result: Stack 0 contains 0624
PC contains 0401H

This example uses one word of memory and executes in one machine cycle.

RL ROTATE LEFT RL

Instruction Word

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	1	0	0	0	cc	cc	cc	cc	cc	0	1	1	1
Op Code							Condition Code					ACC Modification			

Syntax

```
RL A
RL <cc>,A
```

Operation

```
C <= 23 ------------ < ---------- 8 <= C
```

Flags: N Set if result of accumulator is negative
Z Set if result is zero.
C Set if MSB is set before rotate.

Description

The upper 16 bits of the accumulator are rotated left through the carry bit. The lower 8 bits remain unchanged while the resultant LSB, bit 0, is placed with the value 0 (see the accumulator section).

Example1

```
RL A
```

Initialization: A contains 226A84H
Carry bit contains 1

Instruction: RL A

Result: A contains 44D584H
Carry bit contains 0

This example uses one word of memory and executes in one machine cycle. The upper 16 bits (226AH) are shifted left through the carry bit to produce 44D5H. The lower 8 bits (84H) remain unchanged.

Example2

```
RL <cc>, A
```

Initialization: A contains 226A84H
Carry bit contains 0, Z=0

Instruction: RL Z, A

Result: A contains 226A84H

This example uses one word of memory and executes in one machine cycle. The condition code 0 is not set; therefore, the instruction is not executed.

RR ROTATE RIGHT RR

Instruction Word

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	1	0	0	0	cc	cc	cc	cc	cc	0	0	0	0
Op Code							Condition Code					ACC Modification			

Syntax

```
RR A
RR <cc>, A
```

Operation

```
C =>23 ---- > 8 = => 7 - -> -- 0 –> discarded
```

Flags: N Set if result of accumulator is negative.
Z Set if result is 0.
C Set if LSB is set before rotate.

Description

The upper 16 bits of ACC are rotated right through the carry bit. The MSB of the lower 8 bits also obtains the data shifted from the LSB of upper 16 bits. The lower 8 bits are shifted right with the LSB being discarded.

Example1

```
RR A
```

Initialization: A contains 226A84H
Carry bit contains 0

Instruction: RR A

Result: A contains 113542H

This example uses one word of memory and executes in one machine cycle. The upper 16 bits (226AH) are shifted right through the carry bit to produce 1135H. The lower 8 bits (84H) are shifted right to provide 42H.

Example2

`RR <cc>, A`

Initialization: A contains `226A84H`
Carry bit contains `0, Z=0`

Instruction: `RR Z, A`

Result: A contains `226A84H`

This example uses one word of memory and executes in one machine cycle. The condition code `0` is not set; therefore, the instruction is not executed.

SCF SET CARRY FLAG SCF

Instruction Word

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
op	op	op	op	op	op	op	cc	cc	cc	cc	cc	fm	fm	fm	fm
Op Code							Condition Code					Flag Modification			

Syntax

```
SCF
```

Operation

```
1 –> Carry Bit
```

Flags: `C` Set to `1.`

Description

The Set Carry Flag instruction places a one in the carry bit (bit 12 of the Status Register).

Example

```
SCF
```

Initialization: SR contains `2000H`

Instruction: `SCF`

Result: SR contains `3000H`
C contains `1`

This example uses one word of memory and executes in one machine cycle.

SIEF SET INTERRUPT ENABLE FLAG SIEF

Instruction Word

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	1	0	1	0	cc	cc	cc	cc	cc	0	1	0	1
Op Code							Condition Code					Flag Modification			

Syntax

```
SIEF
```

Operation

```
1 –> IE bit
```

Flags: None

Description

The instruction places a `1` in bit 7 of the status register and is used to enable interrupts.

Example

```
SIEF
```

Initialization: SR contains `3000H`

Instruction: `SIEF`

Result: SR contains `3080H`
IE contains `1`

This example uses one word of memory and executes in one machine cycle.

SLL SHIFT LEFT LOGICAL SLL

Instruction Word

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	1	0	0	0	cc	cc	cc	cc	cc	0	0	1	1
Op Code							Condition Code					ACC Modification			

Syntax

```
SLL A
SLL <cc>, A
```

Operation

```
discarded <— C <= 23 - - - - < - - - -0 <= 0
```

Flags: `N` Set if result of accumulator is negative (bit 23 set to `1`).
`Z` Set if result is 0.
`C` Set if MSB is set before shift.

Description

All 24 bits of the accumulator are shifted left through the carry bit. The MSB, bit 23, passes through the carry bit before being discarded. The LSB, bit `0`, is filled with a zero. Subsequent shifts cause additional zeroes to be shifted in.

Example1

```
SLL A
```

Initialization: A contains `226A84H`
Carry bit contains `0`

Instruction: `SLL A`

Result: A contains `226A84H`

This example uses one word of memory and executes in one machine cycle. All 24 bits of the accumulator are shifted left through the carry bit, producing the result `44D508H`.

Example2

```
SLL <cc>, A
```

Initialization: A contains 226A84H
Carry bit contains 0

Instruction: SLL MI, A

Result: A contains 226A84H

This example uses one word of memory and executes in one machine cycle. The condition code N is not set, and the instruction is not executed.

SOPF SET OVERFLOW PROTECTION FLAG SOPF

Instruction Word

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	1	0	1	0	0	0	0	0	0	1	0	0	1
Op Code							Condition Code					Flag Modification			

Syntax

```
SOPF
```

Operation

```
1 –> OP bit
```

Flags: None

Description

The Set Overflow Protection Flag instruction places a one in bit 8 of the status register. If an ALU operation exceeds the limits of the processor, the overflow protection sets the overflow flag (`OV`) and holds the limit value in the accumulator.

Example

```
SOPF
```

Initialization: SR contains `0000H`

Instruction: `SOPF`

Result: SR contains `0100H`
OP contains `1`

This example uses one word of memory and executes in one machine cycle.

SRA SHIFT RIGHT ARITHMETIC SRA

Instruction Word

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	1	0	0	0	cc	cc	cc	cc	cc	0	0	0	0
Op Code							Condition Code					ACC Modification			

Syntax

```
SRA A
SRA        <cc>, A
```

Operation

```
23 => 23 --- > ---- 0 => discarded
```

Flags: N Set if result of accumulator is negative.
Z Set if result is 0.
C Set if LSB is set before the shift.

Description

All 24 bits of the accumulator are shifted right with sign extension through the carry bit. The MSB, bit 23, is replicated in vacated bits. The LSB, bit 0, is passed through the carry before it is discarded.

Example1

```
SRA A
```

Initialization: A contains 226A84H
Carry bit contains 0

Instruction: SRA A

Result: A contains 113542H
Carry bit contains 0

This example uses one word of memory and executes in one machine cycle. All 24 bits of the accumulator are shifted right. The MSB, bit 23, is copied into bit 22. The LSB, bit 0, is discarded.

Example2

```
SRA          <cc>, A
```

Initialization: A contains 226A84H
Carry bit contains 0, N=0

Instruction: SRA A

Result: A contains 113542H
Carry bit contains 0

This example uses one word of memory and executes in one machine cycle. The condition code is set; therefore, the instruction is executed. The initialization of the accumulator sets the PL (NN) condition code.

SUB SUBTRACT SUB

Instruction Word

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	0	0	0	1	b	0	0	0	0	s	s	s	s
Op Code							RAM Bank	Destination				Source			

Syntax

```
SUB A, <regind>
SUB A, <memind>
SUB A, <limm>
SUB A, <hwregs>
SUB A, <direct>
SUB A, <pregs>
SUB A, <dregs>
SUB A, <simm>
```

Operation

```
ACC – (Source) –> ACC
```

Flags:
- C — Set if a carry from the most significant bit is performed.
- N — Set if the result in the accumulator is negative.
- Z — Set if the result is 0.
- OV — Set if the addition exceeds the upper (7FFFFFH) or lower (800000H) limit of the accumulator.

Description

The addressed data memory operand is subtracted from the accumulator. The result is loaded into the accumulator.

The lower eight bits of the accumulator are cleared by the execution of the subtract instruction.

Example1

```
SUB A, <regind>
```

Initialization: Accumulator contains 874600H
P0: 1 contains 45H
RAM Bank1: 45H contains 1234H

Instruction: SUB A, @P0:1

Result: A contains 751200H
@P0:1 contains 1234H

This example uses one word of memory and executes in one machine cycle. The contents of the register pointed by P0:1 are subtracted from the accumulator. The difference is contained in the accumulator and the pointer is left unchanged. The register pointer contains 45H. Because the pointer references RAM Bank1, the absolute register is 145H (325). Therefore, the contents of register 145H are subtracted from the accumulator. 874600H - 123400H = 751200H. The direct addressing equivalent is SUB A,%145 or SUB A,325.

Example2

```
SUB A, <memind>
```

Initialization: Accumulator contains 874600H
P0:0 contains 21H
RAM Bank0: 21H contains 247AH
ROM Address: 247AHcontains 1234H

Instruction: SUB A, @@P0:0+

Result: A contains 751200H
P0:0 contains 22H
RAM Bank0: 21 H contains 247BH

This example uses one word of memory and executes in three machine cycles. The pointer is used for memory indirect addressing. The pointer contains the address of the RAM (address 247AH). The RAM contains the address of the requested ROM data (data 247AH). This operand is subtracted from the accumulator. 874600H - 123400H = 751200H.

Example3

```
SUB A, <limm>
```

Initialization: Accumulator contains 874600H

Instruction: `SUB A, #%1234`

Result: Accumulator contains `751200H`

This example uses two words of memory and executes in two machine cycles. The immediate operand `8746H` is subtracted from the accumulator. `874600H - 123400H = 751200H.`

Example4

```
SUB A, <hwregs>
```

Initialization: Accumulator contains `874600H`
Register X contains `1234H`

Instruction: `SUB A, X`

Result: A contains `751200H`

This example uses one word of memory and executes in one machine cycle. The contents of Register `X` are subtracted from the accumulator. `874600H - 123400H = 751200H`. Transfer from <`hwregs`> is possible from all hardware registers.

Example5

```
SUB A, <direct>
```

Initialization: Accumulator contains `874600H`
RAM Bank0: `F3H` contains `1234H`

Instruction: `SUB A, %F3`

Result: A contains `751200H`

This example uses one word of memory and executes in one machine cycle. The contents of register `F3H` are subtracted from the accumulator. `874600H - 123400H = 751200H`. An equivalent instruction is `SUB A, 243` (`F3H` = 243 decimal).

Example5

```
SUB A, <dregs>
```

Initialization: Accumulator contains `874600H`
D0: 1 contains `1234H`

Instruction: `SUB A, D0:1`

Result: A contains `751200H`
D0:1 contains `1234H`

This example uses one word of memory and executes in one machine cycle. The contents of the data pointer `D0:1` are subtracted from the accumulator. The difference is contained in the accumulator and the pointer is left unchanged.

Example6

```
SUB A, <pregs>
```

Initialization: Accumulator contains `874600H`
P0:0 contains `56H`

Instruction: `SUB A, P0:0`

Result: Accumulator contains `86F000H`

This example uses one word of memory and executes in one machine cycle. The contents of pointer register `P0:0` are subtracted from the accumulator. `874600H - 005600H = 86F000H.` The Pointer Register is connected to the lower 8 bits of the D-bus. The D-bus is connected to the upper 16 bits of the P-bus. This causes the pointer register operand to become `005600H` before it is subtracted from the accumulator.

XOR BITWISE EXCLUSIVE OR XOR

Instruction Word

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	0	0	1	b	0	0	0	0	s	s	s	s
Op Code							RAM Bank	Destination				Source			

Syntax

```
XOR A, <regind>
XOR A, <memind>
XOR A, <limm>
XOR A, <hwregs>
XOR A, <direct>
XOR A, <pregs>
XOR A, <dregs>
XOR A, <simm>
```

Operation

```
A.XOR.<operand> –> A
```

Flags:
- C — Set if carry from the most significant bit is performed.
- N — Set if result in the accumulator is negative.
- Z — Set if result is 0.
- OV — Set if operation exceeds upper (7FFFFFH) or lower (800000H) limit of accumulator.

Description

With the accumulator, perform an XOR instruction on the addressed data memory operand. The result loads into the accumulator.

The lower eight bits of the accumulator are cleared by the XOR instruction.

Example1

```
XOR A, <regind>
```

Initialization: Accumulator contains 005600H
P0:0 contains 00H
RAM Bank0: 00H contains 1234H

Instruction: XOR A, @P0:0

Result: A contains 126200H

This example uses one word of memory and executes in one machine cycle. The pointer is used for memory indirect addressing. The pointer contains the address of the RAM (address 00H). Location 00H has operand 1234H. With the accumulator, 005600H, perform an XOR instruction to obtain the result 126200H.

Example2

```
XOR A, <memind>
```

Initialization: A contains 3264A0H

P21 contains A4H

RAM Bank1: A4H contains 5463H

ROM Address: 5463H contains 1126H

Instruction: XOR A, @@P2:1

Result A contains 2342A0H

P2:1 contains A41H

RAM Bank1: A4H contains 5464H

SR contains 0000H

This example uses one word of memory and executes in three machine cycles. The pointer P2:1 contains the RAM register location(A4H). The contents of this register have a ROM address. This address refers to the ROM data compared to the accumulator. 3264A0H.XOR.112600H = 2342A0H. When indirect memory addressing is used, the ROM address is automatically incremented. Using XOR A, @@P2:1+ performs the same operation and also increments the P21 content to A5H.

Example3

```
XOR A, <limm>
```

Initialization: A contains 3264A0H

Instruction: XOR A, #%1126

Result: A contains 2342A0H
SR contains 0000H

This example uses two words of memory and executes in two machine cycles. Perform an XOR instruction on the immediate data.

Example4

```
XOR A, <hwreg>
```

Initialization: A contains 3264A0H
SR contains 0000H
BUS contains 1126H

Instruction: XOR A, BUS

Result: A contains 2342A0H
SR contains

This example uses one word of memory and executes in one machine cycle. With the accumulator, perform an XOR instruction on the <hwreg> operand.

Example5

```
XOR A, <direct>
```

Initialization: Accumulator contains 3264A0H
RAM Bank0: F3H contains 1126H

Instruction: XOR A, %F3

Result: A contains 2342A0H
SR contains 0000H

This example uses one word of memory and executes in one machine cycle. Register F3H is compared to the accumulator. 3264A0H.XOR.112600H = 2342A0H. An equivalent instruction is XOR A, 243 (F3H = 243 decimal).

5 ELECTRICAL CHARACTERISTICS

5.1 Absolute Maximum and Minimum Ratings

Sym	Parameter	Min	Max	Units	Conditions
V_{CC}	Power supply voltage	0	7	V	
V_{ID}	Input voltage	–0.3	V_{CC}+0.3	V	Digital inputs
V_{IA}	Input voltage	–0.3	V_{CC}+0.3	V	Analog inputs (A/D0–A/D4)
V_O	Output voltage	–0.3	V_{CC}+0.3	V	All push-pull digital outputs
V_O	Output voltage	–0.3	V_{CC}+0.3	V	PWM outputs
I_{OH}	Output current high		–10	mA	one pin
I_{OH}	Output current high		–100	mA	all pins
I_{OL}	Output current low		20	mA	one pin
I_{OL}	Output current low		200	mA	all pins
T_A	Operating temperature	0	70	^oC	
T_S	Storage temperature	–65	150	^oC	

5.2 DC Characteristics

Symbol	Parameter	Min	Typ	Max	Units	Conditions
V_{CC}	Power supply voltage	4.75	5.00	5.25	V	
V_{IL}	Input voltage low	0	0.4	0.2 V_{CC}	V	
V_{IH}	Input voltage high	0.7V_{CC}	3.6	V_{CC}	V	
V_{IHR}	Input voltage High on Reset pin	0.75 V_{CC}	4.2	V_{CC}	V	
V_{PU}	Maximum pull-up voltage			V_{CC}	V	For PWM
V_{OL}	Output voltage low		0.16	0.4	V	@ I_{OL} = 1 mA
V_{OH}	Output voltage high	V_{CC}– 0.4	4.75		V	@ I_{OH} = –0.75 mA
I_{OL}	Output current low	7.2	12		mA	@ V_{OL} = 0.4v
I_{OL1}	Output current low	12	20		mA	@ V_{OL} = 0.8v
I_{OH}	Output current high	4.0	7.0		mA	@ V_{OH} = V_{CC}–0.4
I_{OH1}	Output current high	16	20		mA	@ V_{OH} = 2.4v
I_{OL}	Output current Low	0.35	0.65		mA	@ V_{OL} = 0.4v
I_{OL1}	Output current Low	0.65	1.1		mA	@ V_{OL} = 0.8v
I_{OH}	Output current High	0.35	0.65		mA	@ V_{OH} = V_{CC}–0.4
I_{OH1}	Output current High)	1.1	2.5		mA	@ V_{OH} = 2.4V
V_{XL}	Input voltage XTAL1 low			0.3 V_{CC}	V	External clock generator driven
V_{XH}	Input voltage XTAL1 high	0.6 V_{CC}			V	
I_{IR}	Reset input current	40	90	150	µA	V_{RL} = 0V
I_{IL}	Input leakage	–3.0	0.01	3.0	µA	@ 0V and V_{CC}
I_{CC}	Supply current		60	100	mA	All ports are inputs, RGB is in digital mode
I_{CC}	Supply current		30	50	mA	All ports are inputs, RGB is in analog mode
I_{CC1}	Supply current		5	10	mA	Sleep mode @ 32.768 kHz
I_{CC2}	Supply current		50	100	µA	Stop mode all PWM outputs are @ V_{IN} = 0V

5.3 DC Peripherals

Table 71 V1, V2, and V3 (R,G,B) Analog Output

	Output Voltage (30KΩ Load)			Settling Time
	@ V_{CC} = 4.75V	@ V_{CC} = 5.00V	@ V_{CC} = 5.25V	70% of DC Level, 10pF Load
data = 00	0.00V to 0.65V	0.00V to 0.70V	0.00V to 0.75V	<50ns
data = 01	2.40V ± 0.20V	2.50V ± 0.20V	2.60V ± 0.20V	
data = 10	3.50V ± 0.30V	3.60V ± 0.30V	3.70V ± 0.30V	
data = 11	4.10V ± 0.30V	4.20V ± 0.30V	4.30V ± 0.30V	

Table 72 ADC0/Small Range* (1.5–2.0V; T_A = 0ºC to 70ºC; V_{CC} = 4.75V to 5.25V)

Sym	Parameter	Min	Typ	Max	Units
U_{R-}	Clamping voltage at ADC0	1.0	1.5	2.0	V
ADC_0	Input voltage for level 0	$U_{R-}-(U_{R+}+U_{R-})/15$	U_{R-}	$U_{R-}+(U_{R+}-U_{R-})/15$	V
ADC_1	Input voltage for level 1	U_{R-}	$U_{R-}+(U_{R+}-U_{R-})/15$	$U_{R-}+2(U_{R+}-U_{R-})/15$	V
ADC_2	Input voltage for level 2	$U_{R-}+(U_{R+}-U_{R-})/15$	$U_{R-}+2(U_{R+}-U_{R-})/15$	$U_{R-}+3(U_{R+}-U_{R-})/15$	V
ADC_3	Input voltage for level 3	$U_{R-}+2(U_{R+}-U_{R-})/15$	$U_{R-}+3(U_{R+}-U_{R-})/15$	$U_{R-}+4(U_{R+}-U_{R-})/15$	V
ADC_4	Input voltage for level 4	$U_{R-}+3(U_{R+}-U_{R-})/15$	$U_{R-}+4(U_{R+}-U_{R-})/15$	$U_{R-}+5(U_{R+}-U_{R-})/15$	V
ADC_5	Input voltage for level 5	$U_{R-}+4(U_{R+}-U_{R-})/15$	$U_{R-}+5(U_{R+}-U_{R-})/15$	$U_{R-}+6(U_{R+}-U_{R-})/15$	V
ADC_6	Input voltage for level 6	$U_{R-}+5(U_{R+}-U_{R-})/15$	$U_{R-}+6(U_{R+}-U_{R-})/15$	$U_{R-}+7(U_{R+}-U_{R-})/15$	V
ADC_7	Input voltage for level 7	$U_{R-}+6(U_{R+}-U_{R-})/15$	$U_{R-}+7(U_{R+}-U_{R-})/15$	$U_{R-}+8(U_{R+}-U_{R-})/15$	V
ADC_8	Input voltage for level 8	$U_{R-}+7(U_{R+}-U_{R-})/15$	$U_{R-}+8(U_{R+}-U_{R-})/15$	$U_{R-}+9(U_{R+}-U_{R-})/15$	V
ADC_9	Input voltage for level 9	$U_{R-}+8(U_{R+}-U_{R-})/15$	$U_{R-}+9(U_{R+}-U_{R-})/15$	$U_{R-}+10(U_{R+}-U_{R-})/15$	V
ADC_A	Input voltage for level A	$U_{R-}+9(U_{R+}-U_{R-})/15$	$U_{R-}+10(U_{R+}-U_{R-})/15$	$U_{R-}+11(U_{R+}-U_{R-})/15$	V
ADC_B	Input voltage for level B	$U_{R-}+10(U_{R+}-U_{R-})/15$	$U_{R-}+11(U_{R+}-U_{R-})/15$	$U_{R-}+12(U_{R+}-U_{R-})/15$	V
ADC_C	Input voltage for level C	$U_{R-}+11(U_{R+}-U_{R-})/15$	$U_{R-}+12(U_{R+}-U_{R-})/15$	$U_{R-}+13(U_{R+}-U_{R-})/15$	V
ADC_D	Input voltage for level D	$U_{R-}+12(U_{R+}-U_{R-})/15$	$U_{R-}+13(U_{R+}-U_{R-})/15$	$U_{R-}+14(U_{R+}-U_{R-})/15$	V
ADC_E	Input voltage for level E	$U_{R-}+13(U_{R+}-U_{R-})/15$	$U_{R-}+14(U_{R+}-U_{R-})/15$	$U_{R-}+15(U_{R+}-U_{R-})/15$	V
ADC_F	Input voltage for level F	$U_{R-}+14(U_{R+}-U_{R-})/15$	U_{R+}	$U_{R-}+16(U_{R+}-U_{R-})/15$	V
R_{IN}	Input impedance	1.0			MΩ

NOTE: *The Input voltage level indicated in the table is a switch point from one ADC level to another. To be in the middle of the level half, LSB (($U_{R+}-U_{R-}$)/(15*2)) must be added. The Input voltage must be prorated accordingly if V_{CC} is changed

Table 73 ADC1-ADC4/Full range (0-5.0V) (Vcc = 5.0V, Temp=0-70C)

Data from ADC	Input voltage (volts)	Data from ADC	Input voltage (volts)	Data from ADC	Input voltage (volts)	Data from ADC	Input voltage (volts)
0000	0.000 ± 0.15	0100	1.250 ± 0.15	1000	2.500 ± 0.15	1100	3.750 ± 0.15
0001	0.312 ± 0.15	0101	1.563 ± 0.15	1001	2.813 ± 0.15	1101	4.063 ± 0.15
0010	0.625 ± 0.15	0110	1.875 ± 0.15	1010	3.125 ± 0.15	1110	4.375 ± 0.15
0011	0.934 ± 0.15	0111	2.188 ± 0.15	1011	3.438 ± 0.15	1111	4.688 ± 0.15
Input impedance		> 1.0 MΩ					

5.4 AC Characteristics

Table 74 AC Characteristics (T_A = 0˚C to 70˚C;V_{CC}=4.75V to 5.25V;F_{OSC}=32,768Hz)

Sym	Parameter	Min	Typ	Max	Units
T_{PC}	Input clock period	16	32	100	µS
T_{RC}, T_{FC}	Clock input Rise and Fall		12		ns
TD_{POR}	Power on reset delay	0.8	1.2		S
TW_{RES}	Power on reset minimum width			5 TPC	µS
TD_{HS}	HSYNC incoming signal width	1	10	15	µS
TD_{VS}	VSYNC incoming signal width	1	200	10000	µS
TD_{ES}	Time delay between leading edge of VSYNC and HSYNC in EVEN field	–12	0	+12	µS
TD_{OS}	Time delay between leading edge of VSYNC and HSYNC in ODD field	20	32	44	µS
TW_{HVS}	HSYNC/VSYNC edge width		0.5	2.0	µS

Note: *All timing of the I^2C bus interface are defined by related specifications of the I^2C bus interface.

5.5 Analog RGB

The RGB outputs in analog mode are implemented as controlled current sources with an internal load. These outputs display gamma-corrected, V_{CC} prorated characteristics.

Table 75 RGB Voltage Specification

Parameter	Min	Typical	Max	Units
Supply voltage	4.5	5.0	5.5	V
Full scale voltage	2.5V	2.8V	3.1V	V
2/3 scale voltage	1.7V	2.0V	2.3V	V
1/3 scale voltage	1.0V	1.3V	1.6V	V
Zero scale voltage	—	0.00V	+0.1 V	V
Note: *Measured with 1.5 KΩ load.				

Table 76 RGB Time Specification

Parameter	Min	Typical	Max	Units
Output rise time		50	65	ns
Output fall time		50	65	ns
Note: *Measured with 1.5 KΩ resistor in parallelwith 30 pF capacitor load.				

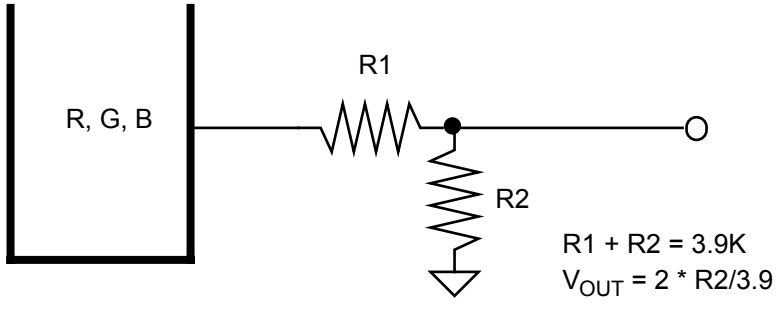


Figure 24 Recommended Application Schematics

6 SYSTEM DESIGN CONSIDERATIONS

The Z90365 provides the ability to

- decode closed-caption transmissions
- display characters on the screen
- manipulate analog and digital control circuits
- monitor keypad and infrared signals directly
- generate OSD if the Z90365 receives vertical and horizontal synchronization signals

In a typical system, normal transmission is received and demodulated. The signals received from the color decoder and deflection unit control the CRT display. To display characters generated by the Z90365 requires a video multiplexor which enables the CRT display's RGB signals and synchronization to be controlled by the video outputs from the processor. When the controller has to display a character on the screen, the multiplexor is switched, and the processor's video signals appear on the display.

The band-limited, A/C-coupled composite video signal is clamped internally to the negative reference voltage (REF–) during the back porch interval. It is then passed to the analog-to-digital converter through a 6:1 multiplexor. The digital signal is then decoded to extract the closed-caption text embedded in the video signal. The characters received are generated as video signals and are then passed to the display.

When a detectable composite video signal is received, the SYNC separator extracts the horizontal and vertical synchronization signals and passes them to the deflection module of the television. The FLYBACK signals from the deflection coils are fed back to the Z90365. The controller uses these signals to align its video signals with those of the normal display. If the composite video signal is not present, video synchronization is provided by the controller. In this case, the SYNC signal pins are set to be outputs. The pins then feed to the deflection unit which controls the display. The SYNC generators can be configured to provide either HSYNC and VSYNC, or H-FLYBACK and V-FLYBACK.

Analog functions such as volume and color controls can be controlled by pulse width modulated outputs from the Z90365. Additional digital controls like channel fine tuning are controlled via the serial I^2C bus.

An infrared remote control receiver can be directly decoded through the capture register, and keypad input can be scanned by directly controlling I/O pins as keyscan ports.

The processor clock is available by referencing an internal phase locked loop to an external 32.768 KHz crystal oscillator. The oscillator minimizes EMI emissions from the clock circuitry. The internal system clock frequency can be selected as 12.058 MHz in normal operation or 32.768 KHz in low power consumption SLEEP mode (usually used if

there is a general system power failure). The Z90365's STOP mode suspends processor clocking for a power-down.

Program, display, and character graphics memory are on the chip, eliminating the meed for external memory components. Characters can be displayed as two or three times normal size. Smoothing and fringing circuits enhance display appearance.

Figure 27 diagrams a typical application of the Z90365 Television Controller as an embedded controller in a television.

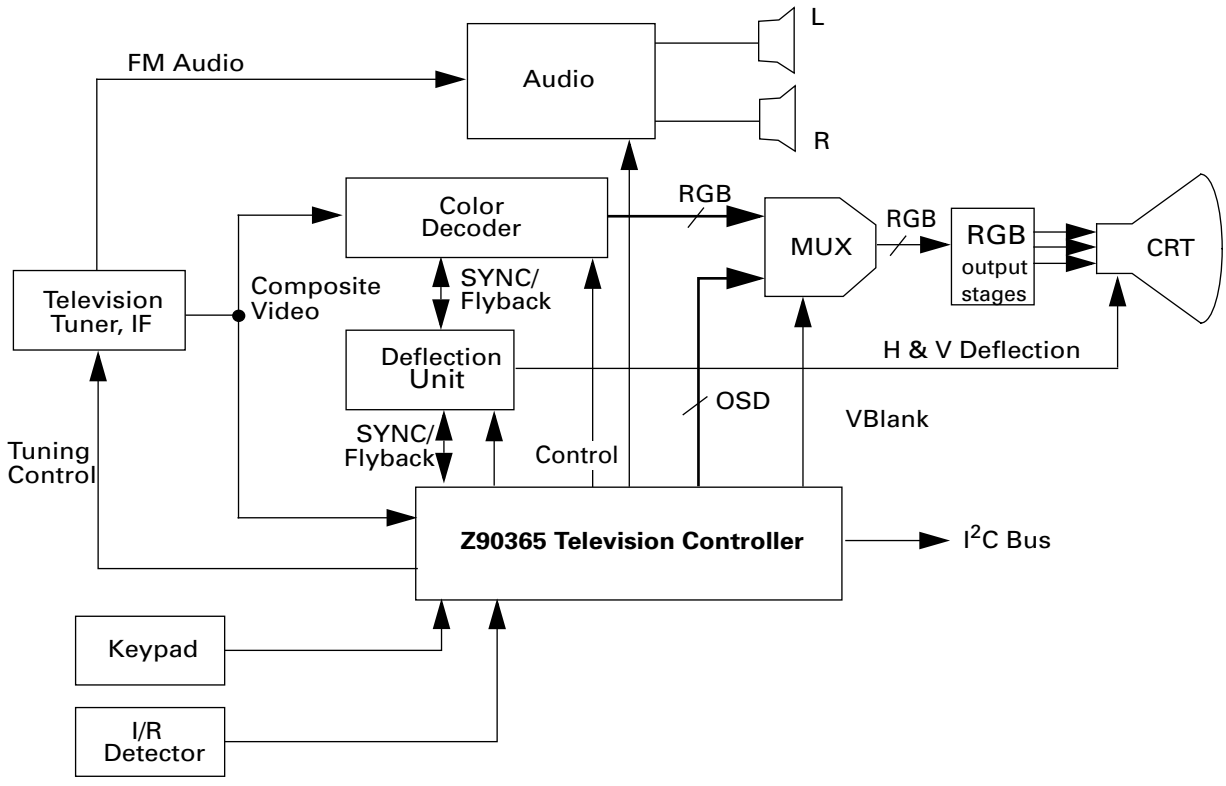
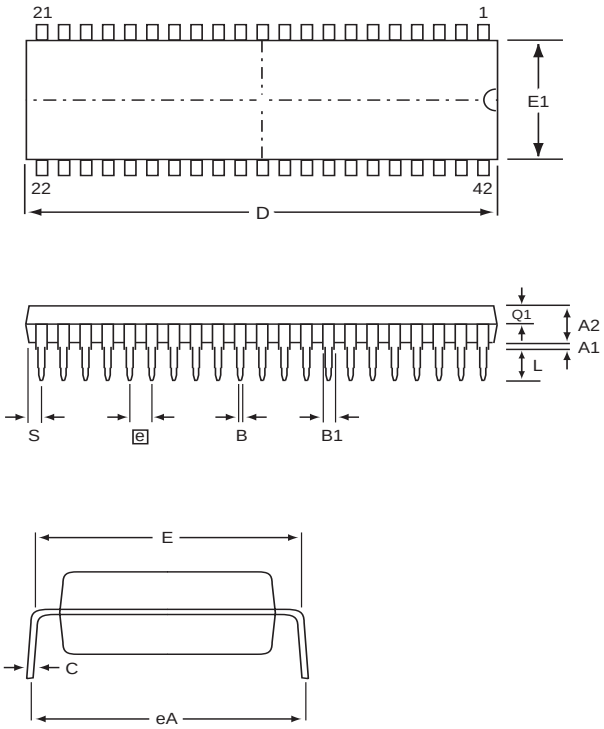


Figure 25 System Block Diagram

7 PACKAGING



Controlling dimensions: inch

Symbol	Millimeter Min	Millimeter Max	Inch Min	Inch Max
A1	0.51		.020	
A2		4.32		.170
B	0.38	0.56	.015	.022
B1	0.76	1.27	.030	.050
C	0.20	0.30	.008	.012
D	36.70	36.96	1.445	1.445
E	15.24	15.88	.600	.625
E1	13.72	14.22	.540	.560
e	1.78 TYP		.070 TYP	
eA	15.49	16.76	.610	.660
L	3.05	3.43	.120	.135
Q1	1.65	1.91	.065	.075
S	0.51	0.76	.020	.030

Customer Feedback Form

Z90365 Product Specification

If there are any problems while operating this product, or any inaccuracies in the specification, please copy and complete this form, then mail or fax it to ZiLOG. Suggestions welcome!

Customer Information

Name	Country
Company	Phone
Address	Fax
City/State/Zip	E-Mail

Product Information

Serial # or Board Fab #/Rev. #
Software Version
Document Number
Host Computer Description/Type

Return Information

ZiLOG
System Test/Customer Support
910 E. Hamilton Avenue, Suite 110, MS 4–3
Campbell, CA 95008
Fax: (408) 558-8536 Email: tools@zilog.com

Problem Description or Suggestion

Provide a complete description of the problem or suggestion. For specific problems, include all steps leading up to the occurrence of the problem. Attach additional pages as necessary.