Zilog - Z88C0020VSC00TR Datasheet

What is "Embedded - Microcontrollers"?

"Embedded - Microcontrollers" refer to small, integrated circuits designed to perform specific tasks within larger systems. These microcontrollers are essentially compact computers on a single chip, containing a processor core, memory, and programmable input/output peripherals. They are called "embedded" because they are embedded within electronic devices to control various functions, rather than serving as standalone computers. Microcontrollers are crucial in modern electronics, providing the intelligence and control needed for a wide range of applications.

Applications of "Embedded - Microcontrollers"

Details	
Product Status	Obsolete
Core Processor	Z8
Core Size	8-Bit
Speed	20MHz
Connectivity	UART/USART
Peripherals	DMA
Number of I/O	32
Program Memory Size	-
Program Memory Type	ROMless
EEPROM Size	-
RAM Size	128K x 8
Voltage - Supply (Vcc/Vdd)	4.75V ~ 5.25V
Data Converters	-
Oscillator Type	Internal
Operating Temperature	0°C ~ 70°C (TA)
Mounting Type	Surface Mount
Package / Case	68-LCC (J-Lead)
Supplier Device Package	-
Purchase URL	https://www.e-xfl.com/product-detail/zilog/z88c0020vsc00tr

Table of Contents

FEATURES 1
GENERAL DESCRIPTION 1
Protopack 4
ARCHITECTURE 5
PIN DESCRIPTIONS 6
REGISTERS 7
Working Register Window 8
Register List 9
MODE AND CONTROL REGISTERS 12
I/O PORTS 17
UART 20
Pins 20
ADDRESS SPACE 21
CPU Program Memory 22
ROMless 22
ROM and Protopack 22
CPU Data Memory 22
INSTRUCTION SET 23
Instruction Pointer 23
Flag Register 24
Condition Codes 25
Addressing Modes 25
INSTRUCTION SUMMARY 29
SUPER-8 OPCODE MAP 38
INSTRUCTIONS 39
INTERRUPTS 42
Sources 42
Vectors 42
Levels 42
Enables 43
Service Routines 44
Fast Interrupt Processing 44
Level or Edge Triggered 45
STACK OPERATION 45
User-Defined Stacks 45
COUNTER/TIMERS 46

DMA 46
ABSOLUTE MAXIMUM RATINGS 47
STANDARD TEST CONDITIONS 47
DC CHARACTERISTICS 48
INPUT HANDSHAKE TIMING 48
AC CHARACTERISTICS (20 MHz) 49
Input Handshake 49
OUTPUT HANDSHAKE TIMING 50
AC CHARACTERISTICS (12 MHz, 20 MHz) 50
Output Handshake 50
AC CHARACTERISTICS (12 MHz) 51
Read /Write 51
AC CHARACTERISTICS (20 MHz) 52
Read /Write 52
AC CHARACTERISTICS (20 MHz) 54
EPROM Read Cycle 54
Example: 54
Packaging Information 54

List of Tables

Table 1. Super-8 Registers 10
Table 2. Port Configuration 18
Table 3. Pin Assignments for Ports 2 and 3 19
Table 4. Condition Codes and Meanings 25
Table 5. Instruction Set Notations 27
Table 6. Instruction Summary 29
Table 7. Second Nibble 37
Table 8. Super8 Instructions 39
Table 9. DC Characteristics 48
Table 10. AC Characteristics (20 MHz) Input Handshake 49
Table 11. AC Characteristics (12 MHz, 20 MHz) Output Handshake 50
Table 12. AC Characteristics (12 MHz) Read/Write 51
Table 13. AC Characteristics (20 MHz) Read/Write 52
Table 14. AC Characteristics (20 MHz) EPROM Read Cycle 54

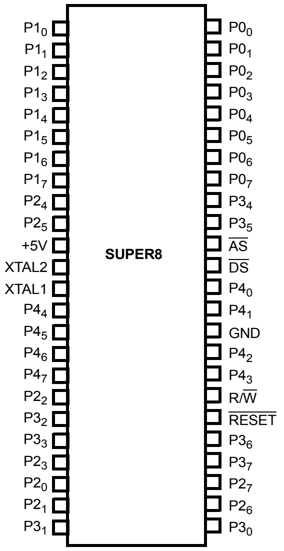


Figure 3.Pin Assignments 48-Pin DIP

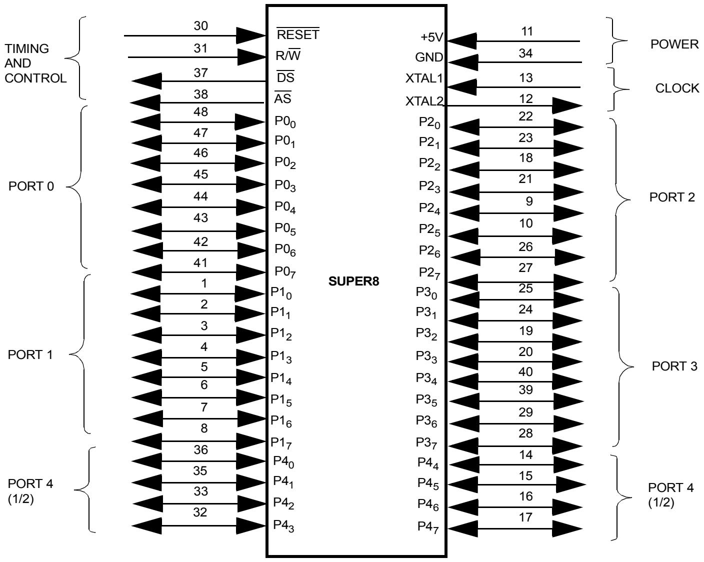


Figure 4.Pin Functions 48-Pin DIP

The 16-bit counters can operate independently or be cascaded to perform 32-bit counting and timing operations. The DMA controller handles transfers to and from the register file or memory. DMA can use the UART or one of two ports with handshake capability.

The architecture appears in the block diagram (Figure 7).

PIN DESCRIPTIONS

The Super8 connects to external devices via the following TTL-compatible pins:

$\overline{\text{AS}}$. *Address Strobe* (output, active Low). $\overline{\text{AS}}$ is pulsed Low once at the beginning of each machine cycle. The rising edge indicates that addresses $R/\overline{W}$ and $\overline{DM}$, when used, are valid.

$\overline{\text{DS}}$. *Data Strobe* (output, active Low). $\overline{\text{DS}}$ provides timing for data movement between the address/data bus and external memory. During write cycles, data output is valid at the leading edge of $\overline{\text{DS}}$. During read cycles, data input must be valid prior to the trailing edge of $\overline{\text{DS}}$.

$P0_0$-$P0_7$, $P1_0$-$P1_7$, $P2_0$-$P2_7$, $P3_0$-$P3_7$, $P4_0$-$P4_7$. *Port I/O Lines* (input/output). These 40 lines are divided into five 8-bit I/O ports that can be configured under program control for I/O or external memory interface.

In the ROMless devices, Port 1 is dedicated as a multiplexed address/data port, and Port 0 pins can be assigned as additional address lines; Port 0 non-address pins may be assigned as I/O. In the ROM and protopack, Port 1 can be assigned as input or output, and Port 0 can be assigned as input or output on a bit by bit basis.

Ports 2 and 3 can be assigned on a bit-for-bit basis as general I/O or interrupt lines. They can also be used as special-purpose I/O lines to support the UART, counter/timers, or handshake channels.

Port 4 is used for general I/O.

During reset, all port pins are configured as inputs (high impedance) except for Port 1 and Port 0 in the ROMless devices. In these, Port 1 is configured as a multiplexed address/data bus, and Port 0 pins PO_0-PO_4 are configured as address out, while pins $P0_5$-PO_7 are configured as inputs.

$\overline{\text{RESET}}$. *Reset* (input, active Low). Reset initializes and starts the Super8. When it is activated, it halts all processing; when it is deactivated, the Super8 begins processing at address 0020H.

ROMless. (input, active High). This input controls the operation mode of a 68-pin Super8. When connected to VCC, the part functions as a ROMless Z8800. When connected to GND, the part functions as a Z8820 ROM part.

Port 4

Port 4 can be configured as I/O only. Each bit can be configured individually as input or output, with either push-pull or open-drain outputs. All Port 4 inputs are Schmitt-triggered.

Port 4 can be placed under handshake control of handshake channel 0. Its register address is R212.

UART

The UART is a full-duplex asynchronous channel. It transmits and receives independently with 5 to 8 bits per character, has options for even or odd bit parity, and a wake-up feature.

Data can be read into or out of the UART via R239, Bank 0. This single address is able to serve a full-duplex channel because it contains two complete 8-bit registers-one for the transmitter and the other for the receiver.

Pins

The UART uses the following Port 2 and 3 pins:

Port/Pin	UART Function
2/0	Receive Clock
3/0	Receive Data
2/1	Transmit Clock
3/1	Transmit Data

Transmitter

When the UART's register address is specified as the destination (dst) of an operation, the data is output on the UART, which automatically adds the start bit, the programmed parity bit, and the programmed number of stop bits. It can also add a wake-up bit if that option is selected.

If the UART is programmed for a 5-, 6-, or 7-bit character, the extra bits in R239 are ignored.

Serial data is transmitted at a rate equal to 1, 1/16, 1/32 or 1/64 of the transmitter clock rate, depending on the programmed data rate. All data is sent out on the falling edge of the clock input.

controlled by the $\overline{DM}$ line (Port $P3_5$), which selects data memory when Low and program memory when High.

Figure 16 on page 23 shows the system memory space.

CPU Program Memory

Program memory occupies addresses 0 to 64K. External program memory, if present, is accessed by configuring Ports 0 and 1 as a memory interface.

The address/data lines are controlled by $\overline{AS}$, $\overline{DS}$ and R/$\overline{W}$.

The first 32 program memory bytes are reserved for interrupt vectors; the lowest address available for user programs is 32 (decimal). This value is automatically loaded into the program counter after a hardware reset.

ROMless

Port 0 can be configured to provide from 0 to 8 additional address lines. Port 1 is always used as an 8-bit multiplexed address/data port.

ROM and Protopack

Port 1 is configured as multiplexed address/data or as I/O. When Port 1 is configured as address/data, Port 0 lines can be used as additional address lines, up to address 15. External program memory is mapped above internal program memory; that is, external program memory can occupy any space beginning at the top of the internal ROM space up to the 64K (16-bit address) limit.

CPU Data Memory

The external CPU data memory space, if separated from program memory by the $\overline{DM}$ optional output, can be mapped anywhere from 0 to 64K (full 16-bit address space). Data memory uses the same address/data bus (Port 1) and additional addresses (chosen from Port 0) as program memory. Data memory is distinguished from program memory by the DM pin ($P3_5$), and by the fact that data memory can begin at address OOOOH. This feature differs from the Z8.

Flag Register

The Flag register (FLAGS) contains eight bits that describe the current status of the Super8. Four of these can be tested and used with conditional jump instructions; two others are used for BCD- arithmetic. FLAGS also contains the Bank Address bit and the Fast Interrupt Status bit.

The flag bits can be set and reset by instructions.

⚠ **Caution:** Do not specify FLAGS as the destination of an instruction that normally affects the flag bits or the result is unspecified.

The following paragraphs describe each flag bit:

Bank Address. This bit is used to select one of the register banks (0 or 1) between (decimal) addresses 224 and 255. It is cleared by the SBO instruction and set by the SB1 instruction.

Fast Interrupt Status. This bit is set during a fast interrupt cycle and reset during the IRET following interrupt servicing. When set, this bit inhibits all interrupts and causes the fast interrupt return to be executed when the IRET instruction is fetched.

Half-Carry. This bit is set to 1 whenever an addition generates a carry out of bit 3, or when a subtraction borrows out of bit 4. This bit is used by the Decimal Adjust (DA) instruction to convert the binary result of a previous addition or subtraction into the correct decimal (BCD) result. This flag, and the Decimal Adjust flag, are not usually accessed by users.

Decimal Adjust. This bit is used to specify what type of instruction was executed last during BCD operations, so a subsequent Decimal Adjust operation can function correctly. This bit is not usually accessible to programmers, and cannot be used as a test condition.

Overflow Flag. This flag is set to 1 when the result of a twos-complement operation was greater than 127 or less than -128. It is also cleared to O during logical operations.

Sign Flag. Following arithmetic, logical, rotate, or shift operations, this bit identifies the state of the MSB of the result. A 0 indicates a positive number and a 1 indicates a negative number.

Zero Flag. For arithmetic and logical operations, this flag is set to 1 if the result of the operation is zero.

For operations that test bits in a register, the zero bit is set to 1 if the result is zero.

For rotate and, shift operations, this bit is set to 1 if the result is zero.

Functional Summary of Commands

Figure 17 shows the formats followed by a quick reference guide to the commands.

Table 19.Instruction Set Notations

Notation	Meaning	Notation	Meaning
cc	Condition code (see Table 4)		DA Direct address (between 0 and 65535)
r	Working register (between 0 and 15)		RA Relative address
rb	Bit of working register	IM	Immediate
r0	Bit 0 of working register	IML	Immediate long
R	Register or working register		dst Destination operand
RR	Register pair or working register pair (Register always start on an even-number boundary)	pairs	src Source operand
		@	Indirect
IA	Indirect address	SP	Stack
Ir	Indirect working register	PC	Program
IR	Indirect register or indirect working register		IP
Irr	Indirect working register pair	FLAGS	Flags
IRR	Indirect register pair or indirect working register	pair	RP
X	Indexed	#	Immediate
XS	Indexed, short offset	%	Hexadecimal
XL	Indexed, long offset	OPC	Opcode

Table 20.Instruction Summary (Continued)

Instruction and Operation	Address Mode		Opcode Byte (Hex)	Flags Affected					
	dst	src		C	Z	S	V	D	H
SP←SP-2	IRR		F4						
@SP←PC,	IA		D4						
PC←dst									
CCF			EF	*	-	-	-	-	-
C = NOT C									
CLR dst	R		B0	-	-	-	-	-	-
dst←0	IR		B1						
COM dst	R		60	-	*	*	0	-	-
dst←NOT dst	IR		61						
CP dst, src	Note[1]		A[ ]	*	*	*	*	-	-
dst - src									
CPIJE	r	Ir	C2	-	-	-	-	-	-
If dst – src = 0, then									
PC←PC + RA									
Ir←Ir + 1									
CPIJNE	r	Ir	D2	-	-	-	-	-	-
If dst – src = 0, then									
PC←PC + RA									
Ir←Ir + 1									
DA dst	R		40	*	*	*	U	-	-
dst←DA dst	IR		41						
DEC dst	R		00	-	*	*	*	-	-
dst←dst -1	IR		01						
DECW dst	RR		80	-	*	*	*	-	-
dst←dst-1	IR		81						
DI			8F	-	-	-	-	-	-

Table 20.Instruction Summary (Continued)

Instruction and Operation	Address Mode		Opcode Byte (Hex)	Flags Affected					
	dst	src		C	Z	S	V	D	H
SMR(0)←0									
DIV dst, src									
dst ÷ src	RR	R	94	*	*	*	*	-	-
dst (Upper)←Quotient	RR	IR	95						
dst (Lower)←Remainder	RR	IM	96						
DJNZ r, dst	RA	r	rA	-	-	-	-	-	-
r←r - 1			(r = 0 to F)						
if r = 0									
PC←PC + dst									
EI			9F	-	-	-	-	-	-
SMR(0)←1									
ENTER			1F	-	-	-	-	-	-
SP←SP - 2									
@SP←IP									
IP←PC									
PC←@IP									
IP←IP + 2									
EXIT			2F	-	-	-	-	-	-
IP←@SP									
SP←SP + 2									
PC←@IP									
IP←IP + 2									
INC dst	r		rE	-	*	*	*	-	-
dst←dst + 1			(r = 0 to F)						
	R		20						
	IR		21						

Table 20.Instruction Summary (Continued)

Instruction	Address Mode		Opcode	Flags Affected					
and Operation	dst	src	Byte (Hex)	C	Z	S	V	D	H
INCW dst	RR		A0	-	*	*	*	-	-
dst←1 + dst	IR		A1						
IRET (Fast)			BF	Restored to before interrupt					
PC↔IP									
FLAG←FLAG‘									
FIS←0									
IRET (Normal)			BF	Restored to before interrupt					
FLAGS←@SP; SP←SP + 1									
PC←@SP; SP←SP + 2; SMR(0)←1									
JP cc, dst	DA		ccD	-	-	-	-	-	-
if cc is true,			(cc = 0 to F)						
PC←dst	IRR		30						
JR cc, dst	RA		ccB	-	-	-	-	-	-
if cc is true,									
PC←PC + d			(cc = 0 to F)						
LD dst, src	r	IM	rC	-	-	-	-	-	-
dst←src	r	R	r8						
	R	r	r9						
			(r = 0 to F)						
	r	IR	C7						
	IR	r	D7						
	R	R	E4						
	R	IR	E5						
	R	IM	E6						
	IR	IM	D6						
	IR	R	F5						
	r	x	87						
	x	r	97						

Table 21.Second Nibble

Addr Mode		Lower Opcode Nibble[1]
dst	src	
r	r	[2]
r	Ir	[3]
R	R	[4]
R	IR	[5]
R	IM	[6]

[1] For example, to use an opcode represented as x[ ] with an "RR" addressing mode, use the opcode "x4."
0= Cleared to Zero
1= Set to One
-= Unaffected
*= Set or reset, depending on result of operation.
U= Undefined

SUPER-8 OPCODE MAP

Lower Nibble (Hex)

Upper Nibble (Hex)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	6 **DEC** R_1	6 **DEC** IR_1	6 **ADD** r_1,r_2	6 **ADD** r_1,Ir_2	10 **ADD** R_2,R_1	10 **ADD** IR_2,R_1	10 **ADD** R_1,IM	10 **BOR*** $r_0\text{-}R_b$	6 **LD** r_1,R_2	6 **LD** r_2,R_1	12/10 **DJNZ** r_1,RA	12/10 **JR** cc,RA	6 **LD** r_1,RM	12/10 **JP** cc,DA	6 **INC** r_1	14 **NEXT**
1	6 **RLC** R_1	6 **RLC** IR_1	6 **ADC** r_1,r_2	6 **ADC** r_1,Ir_2	10 **ADD** R_2,R_1	10 **ADC** IR_2,R_1	10 **ADC** R_1,IM	10 **BCP** r_1,b,R_2	↓	↓	↓	↓	↓	↓	↓	20 **ENTER**
2	6 **INC** R_1	6 **INC** IR_1	6 **SUB** r_1,r_2	6 **SUB** r_1,Ir_2	10 **SUB** R_2,R_1	10 **SUB** IR_2,R_1	10 **SUB** R_1,IM	10 **BXOR*** $r_0\text{-}R_b$	↓	↓	↓	↓	↓	↓	↓	22 **EXIT**
3	10 **JP** IRR_1	**NOTE C**	6 **SBC** r_1,r_2	6 **SBC** r_1,Ir_2	10 **SBC** R_2,R_1	10 **SBC** IR_2,R_1	10 **SBC** R_1,IM	**NOTE A**	↓	↓	↓	↓	↓	↓	↓	6 **WFI**
4	6 **DA** R_1	6 **DA** IR_1	6 **OR** r_1,r_2	6 **OR** r_1,Ir_2	10 **OR** R_2,R_1	10 **OR** IR_2,R_1	10 **OR** R_1,IM	10 **LDB*** $r_0\text{-}Rb$	↓	↓	↓	↓	↓	↓	↓	6 **SBO**
5	10 **POP** R_1	10 **POP** IR_1	6 **AND** r_1,r_2	6 **AND** r_1,Ir_2	10 **AND** R_2,R_1	10 **AND** IR_2,R_1	10 **AND** R_1,IM	10 **BITC** r_1,b	↓	↓	↓	↓	↓	↓	↓	6 **SBI**
6	6 **COM** R_1	6 **COM** IR_1	6 **TCM** r_1,r_2	6 **TCM** r_1,Ir_2	10 **TCM** R_2,R_1	10 **TCM** IR_2,R_1	10 **TCM** R_1,IM	10 **BAND*** $r_0\text{-}Rb$	↓	↓	↓	↓	↓	↓	↓	
7	10/12 **PUSH** R_2	12/14 **PUSH** IR_2	6 **TM** r_1,r_2	6 **TM** r_1,Ir_2	10 **TM** R_2,R_1	10 **TM** IR_2,R_1	10 **TM** R_1,IM	**NOTE B**	↓	↓	↓	↓	↓	↓	↓	
8	10 **DECW** IRR_1	10 **DECW** IR_1	10 **PUSHUD** IR_1,R_2	10 **PUSHUI** IR_2,R_2	24 **MULT** R_2,RR_1	24 **MULT** IR_2,RR_1	24 **MULT** IM,RR_1	10 **LD** r_1,x,r_2	↓	↓	↓	↓	↓	↓	↓	6 **DI**
9	6 **RL** R_1	6 **RL** IR_1	6 **POPUD** IR_2,R_1	6 **POPUI** IR_2,R_1	28/12 **DIV** R_2,RR_1	28/12 **DIV** IR_2,RR_1	28/12 **DIV** IM,RR_1	10 **LD** r_2,x,r_1	↓	↓	↓	↓	↓	↓	↓	6 **EI**
A	10 **INCW** RR_1	10 **INCW** IR_1	6 **CP** r_1,r_2	6 **CP** r_1,Ir_2	10 **CP** R_2,R_1	10 **CP** IR_2,R_1	10 **CP** R_1,IM	**NOTE D**	↓	↓	↓	↓	↓	↓	↓	14 **RET**
B	6 **CLR** R_1	6 **CLR** IR_1	6 **XOR** r_1,r_2	6 **XOR** r_1,Ir_2	10 **XOR** R_2,R_1	10 **XOR** IR_2,R_1	10 **XOR** R_1,IM	**NOTE E**	↓	↓	↓	↓	↓	↓	↓	16/6 **IRET**
C	6 **RRC** IRR_1	6 **RRC** IR_1	16/18 **CPIJE** Ir,r_2,RA	12 **LDC*** r_1,Irr_2	10 **LDW** RR_2,RR_1	10 **LDW** IR_2,RR_1	12 **LDW** RR_1,IML	6 **LD** r_1,Ir_2	↓	↓	↓	↓	↓	↓	↓	6 **RCF**
D	6 **SRA** R_1	6 **SRA** IR_1	16/18 **CPIJNE** Ir_1,r_2,RA	12 **LDC*** r_2,Irr_1	20 **CALL** IA_1		10 **LD** IR_1,IM	6 **LD** $Ir1,r_2$	↓	↓	↓	↓	↓	↓	↓	6 **SCF**
E	6 **RR** R_1	6 **RR** IR_1	16 **LDCD*** r_1,Irr_2	16 **LDCI*** r_1,Irr_2	10 **LD** R_2,R_1	10 **LD** IR_2,R_1	10 **LD** R_1,IM	18 **LDC*** r_1,Irr_2,xs	↓	↓	↓	↓	↓	↓	↓	6 **CCF**
F	8 **SWAP** R_1	8 **SWAP** IR_1	16 **LDCPD*** r_2,Irr_1	16 **LDCPI*** r_2,Irr_1	18 **CALL** IRR_1	18 **CALL** R_2,IR_1	18 **CALL** DA_1	18 **LDC*** r_1,Irr_1,xs	↓	↓	↓	↓	↓	↓	↓	6 **NOP**

Note A

16/18 **BTJRF** r_2,b,RA	16/18 **BTJRT** r_2,b,RA

Note B

8 **BITR** r_1,b	8 **BITS** r_1,b

Note C

6 **SRP** IM	6 **SRP0** IM	6 **SRP1** IM

Note D

20 **LDC*** r_1,Irr_2,xL	20 **LDC*** r_1,DA_2

Note E

20 **LDC*** r_2,Irr_2,xL	20 **LDC*** r_2,DA_1

Legend:
r = 4-bit address
R = 8-bit address
b = bit number
R_1 or r_1 = dsts address
R_2 or r_2 = src address

***Examples:**
BORr_0-R_2
is BORr_1,b,R_2
or BORr_2,b,R_1
LDCr_1,Irr_2
is LDCr_1,Irr_2 = program
or LDEr_1,Irr_2 = data

Sequence:
Opcode, first, second, third operands

NOTE: The blank areas are not defined.

Figure 19.Opcode Map

INSTRUCTIONS

Table 22.Super8 Instructions

Mnemonic	Operands	Instruction
Load Instructions		
CLR	dst	Clear
LD	dst, src	Load
LDB	dst, src	Load bit
LDC	dst, src	Load program memory
LDE	dst, src	Load data memory
LDCD	dst, src	Load program memory and decrement
LDED	dst, src	Load data memory and decrement
LDCI	dst, src	Load program memory and increment
LDEI	dst, src	Load data memory and increment
LDCPD	dst, src	Load program memory with pre-decrement
LDEPD	dst, src	Load data memory with pre-decrement
LDCPI	dst, src	Load program memory with pre-increment
LDEPI	dst, src	Load data memory with pre-increment
LDW	dst, src	Load word
POP	dst	Pop stack
POPUD	dst, src	Pop user stack (decrement)
POPUI	dst, src	Pop user stack (increment)
PUSH	src	Push stack
PUSHUD	dst, src	Push user stack (decrement)
PUSHUI	dst, src	Push user stack (increment)
Arithmetic Instructions		
ADC	dst, src	Add with carry
ADD	dst, src	Add
CP	dst, src	Compare
DA	dst	Decimal adjust
DEC	dst	Decrement

Table 22.Super8 Instructions (Continued)

Mnemonic	Operands	Instruction
Bit Manipulation Instructions		
BAND	dst,src	Bit AND
BCP	dst, src	Bit compare
BITC	dst	Bit complement
BITR	dst	Bit reset
BITS	dst	Bit set
BOR	dst, src	Bit OR
BXOR	dst, src	Bit exclusive OR
TCM	dst, src	Test complement under mask
TM	dst, src	Test under mask
Rotate and Shift Instructions		
RL	dst	Rotate left
RLC	dst	Rotate left through carry
RR	dst	Rotate right
RRC	dst	Rotate right through carry
SRA	dst	Shift right arithmetic
SWAP	dst	Swap nibbles
CPU Control Instructions		
CCF		Complement carry flag
DI		Disable interrupts
EI		Enable interrupts
NOP		Do nothing
RCF		Reset carry flag
SB0		Set bank 0
SB1		Set bank 1
SCF		Set carry flag
SRP	src	Set register pointers
SRP0	src	Set register pointer zero

Service Routines

Before an interrupt request can be granted, a) interrupts must be enabled, b) the level must be enabled, c) it must be the highest priority interrupting level, d) it must be enabled at the interrupting source, and e) it must have the highest priority within the level.

If all this occurs, an interrupt request is granted.

The Super8 then enters an interrupt machine cycle that completes the following sequence:

- It resets the Interrupt Enable bit to disable all subsequent interrupts.
- It saves the Program Counter and status flags on the stack.
- It branches to the address contained within the vector location for the interrupt.
- It passes control to the interrupt servicing routine.

When the interrupt servicing routine has serviced the interrupt, it should issue an interrupt return (IRET) instruction. This restores the Program Counter and status flags and sets the Interrupt Enable bit in the System Mode register.

Fast Interrupt Processing

The Super8 provides a feature called fast interrupt processing, which completes the interrupt servicing in 6 clock periods instead of the usual 22.

Two hardware registers support fast interrupts. The Instruction Pointer (IP) holds the starting address of the service routine, and saves the PC value when a fast interrupt occurs. A dedicated register, FLAG', saves the contents of the FLAGS register when a fast interrupt occurs.

To use this feature, load the. address of the service routine in the Instruction Pointer, load the level number into the Fast Interrupt Select field, and turn on the Fast Interrupt Enable bit in the System Mode register.

When an interrupt occurs in the level selected for fast interrupt processing, the following occurs:

- The contents of the Instruction Pointer and Program Counter are swapped.
- The contents of the Flag register are copied into FLAG'.
- The Fast Interrupt Status Bit in FLAGS is set.
- The interrupt is serviced.
- When IRET is issued after the interrupt service outline is completed, the Instruction Pointer and Program Counter are swapped again.

- The contents of FLAG’ are copied back into the Flag register
- The Fast Interrupt Status bit in FLAGS is cleared.

The interrupt servicing routine selected for fast processing should be written so that the location after the IRET instruction is the entry point the next time the (same) routine is used.

Level or Edge Triggered

Because internal interrupt requests are levels and interrupt requests from the outside are (usually) edges, the hardware for external interrupts uses edge-triggered flip-flops to convert the edges to levels.

The level-activated system requires that interrupt-serving software perform some action to remove the interrupting source. The action involved in serving the interrupt may remove the source, or the software may have to actually reset the flip-flops by writing to the corresponding Interrupt Pending register.

STACK OPERATION

The Super8 architecture supports stack operations in the register file or in data memory. Bit 1 in the external Memory Timing register (R254 bank 0) selects between the two.

Register pair 216-217 forms the Stack Pointer used for all stack operations. R216 is the MSB and R217 is the LSB.

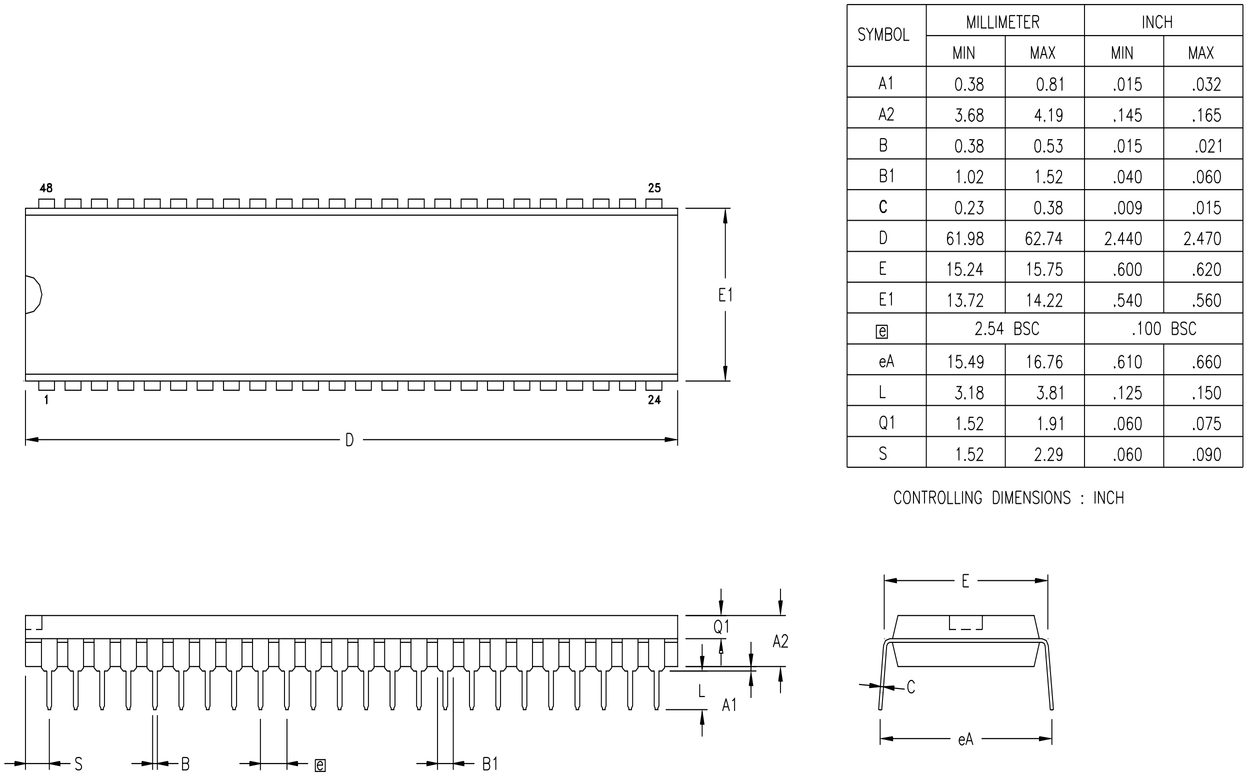
The Stack Pointer always points to data stored on the top of the stack. The address is decremented prior to a PUSH and incremented after a POP.

The stack is also used as a return stack for CALLs and interrupts. During a CALL, the contents of the PC are saved on the stack, to be restored later. Interrupts cause the contents of the PC and FLAGS to be saved on the stack, for recovery by IRET when the interrupt is finished.

When the Super8 is configured for an internal stack (using the register file), R217 contains the Stack Pointer. R216 may be used as a general-purpose register, but its contents are changed if an overflow or underflow, occurs as the result of incrementing or decrementing the stack address during normal stack operations.

User-Defined Stacks

The Super8 provides for user-defined stacks in both the register file and program or data memory. These can be made to increment or decrement on a push by the choice of opcodes. For example, to implement a stack that grows from low



SYMBOL	MILLIMETER		INCH	
	MIN	MAX	MIN	MAX
A1	0.38	0.81	.015	.032
A2	3.68	4.19	.145	.165
B	0.38	0.53	.015	.021
B1	1.02	1.52	.040	.060
C	0.23	0.38	.009	.015
D	61.98	62.74	2.440	2.470
E	15.24	15.75	.600	.620
E1	13.72	14.22	.540	.560
ⓔ	2.54 BSC		.100 BSC	
eA	15.49	16.76	.610	.660
L	3.18	3.81	.125	.150
Q1	1.52	1.91	.060	.075
S	1.52	2.29	.060	.090

CONTROLLING DIMENSIONS : INCH

Figure 29.48-Pin DIP