LAMPIRAN A

FOTO ALAT
Foto 1 Rangkaian Regulator

Foto 2 Alat Pencatat Tarif Biaya Angkot
Features
- Compatible with MCS-51™ Products
- 4K Bytes of In-System Reprogrammable Flash Memory
- Endurance: 1,000,000 Write/Erase Cycles
- Fully Static Operation: 0 Hz to 24 MHz
- Three-Level Program Memory Lock
- 128 x 8-bit Internal RAM
- 32 Programmable I/O Lines
- Six Interrupt Sources
- Programmable Serial Channel
- Low Power Idle and Power Down Mode

Description
The AT89C51 is a low-power, high-performance CMOS 8-bit microcomputer with 4K bytes of Flash Programmed and Erasable Read Only Memory (PEROM). This device is manufactured using ATmega's high density nonvolatile memory technology and is compatible with the industry standard MCS-51™ instruction set and pinout. The on-chip Flash allows the program memory to be reprogrammed in-system or by a conventional nonvolatile memory programmer. By combining a versatile 8-bit CPU with Flash on a monolithic chip, the ATmega AT89C51 is a powerful microcomputer which provides a highly reliable and cost-effective solution to many embedded control applications.

Pin Configurations

Block Diagram
The AT89C51 provides the following standard features: 4K bytes of Flash, 128 bytes of RAM, 32 I/O lines, two 16-bit timers/counters, a five vector two-level interrupt architecture, a full duplex serial port, on-chip oscillator and clock circuit. In addition, the AT89C51 is designed with static logic for operation down to zero frequency and supports two software selectable power saving modes. The Idle Mode stops the CPU while allowing the RAM, timers/counters, serial port and interrupt system to continue functioning. The Power Down Mode saves the RAM contents but freezes the oscillator disabling all other chip functions until the next hardware reset.

**Pin Description**

Vcc: Supply voltage

GND: Ground

**Port 0**

Port 0 is an 8-bit open drain bidirectional I/O port. An output port each pin can sink eight TTL inputs. When 1's are written to port 0 pins, the pin can be used as high-impedance inputs. Port 0 may also be configured to be the multiplexed low-order address/data bus during access to external program and data memory. In this mode P0 has internal pull-up.

Port 0 also receives the data bytes during Flash programming, and outputs the code bytes during program verification. External pull-ups are required during program verification.

Port 1

Port 1 is an 8-bit bidirectional I/O port with internal pull-ups. Port 1 output buffers can sink/source four TTL inputs. When 1's are written to Port 1 pins they are pulled high by the internal pull-ups and can be used as inputs. As inputs, Port 1 pins that are externally being pulled low will source current (Io) through the pull-ups.

Port 1 also receives the low-order address bytes during Flash programming and verification.

**Port 2**

Port 2 is an 8-bit bidirectional I/O port with internal pull-ups. Port 2 output buffers can sink/source four TTL inputs. When 1's are written to Port 2 pins they are pulled high by the internal pull-ups and can be used as inputs. As inputs, Port 2 pins that are externally being pulled low will source current (Io) through the pull-ups.

Port 2 also receives the high-order address byte during fetches from external program memory, and during accesses to external data memory that use 16-bit addresses (MOVX @ DDRX).

When the AT89C51 is executing code from external program memory, PSEN is activated twice each machine cycle, except that two PSEN activations are skipped during each access to external data memory.

**EA**

External Access Enable. EA must be strapped to GND in order to enable the device to fetch code from external program memory locations starting at 0000H up to FF00H. Note, however, that it bit 1 is programmed, EA will be internally latched on reset.

**Vcc**

Should be strapped to VCC for internal program execution. This pin also receives the 12-volt programming enable voltage (Vpp) during Flash programming, for pins that require 12-volt Vpp.

**XTAL1**

Input to the inverting oscillator amplifier and input to the internal clock operating circuit.

**XTAL2**

Output from the inverting oscillator amplifier.

**Oscillator Characteristics**

XTAL1 and XTAL2 are the input and output, respectively, of an inverting amplifier which can be configured for use as an on-chip oscillator, as shown in Figure 1. Either a quartz crystal or ceramic resonator may be used. To drive the device from an external clock source, XTAL1 should be left unconnected while XTAL2 is driven as shown in Figure 2. There are no requirements on the duty cycle of the external clock signal, since the input to the internal clock circuitry is through a divide-by-two flip-flop, but minimum and maximum voltage levels and low time specifications must be observed.

**Idle Mode**

In idle mode, the CPU puts itself to sleep while all the on-chip peripherals remain active. The mode is invoked by software. The contents of the on-chip RAM and all the special function registers remain unchanged during this mode. The idle mode can be terminated by any enabled interrupt or by a hardware reset.

**Status of External Pins During Idle and Power Down Modes**

<table>
<thead>
<tr>
<th>Mode</th>
<th>Program Memory</th>
<th>ALE</th>
<th>DSRI</th>
<th>DTR1 DC</th>
<th>DTR2 DC</th>
<th>DOR1 DC</th>
<th>DOR2 DC</th>
<th>DOR3 DC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Idle</td>
<td>External</td>
<td>0</td>
<td>0</td>
<td>Data</td>
<td>Data</td>
<td>Data</td>
<td>Data</td>
<td>Data</td>
</tr>
<tr>
<td>Power Down</td>
<td>Internal</td>
<td>0</td>
<td>0</td>
<td>Data</td>
<td>Data</td>
<td>Data</td>
<td>Data</td>
<td>Data</td>
</tr>
</tbody>
</table>

**Program Store Enable**

Enable the next strobe to external program memory.

It should be noted that when idle is terminated by a hard reset, the device normally resumes program execution from where it left off, up to two machine cycles before the internal reset algorithm takes control. On-chip hardware inhibits access to internal RAM in this event, but access to the port pins is not inhibited. To eliminate the possibility of an uncontrolled write to a port pin when idle is terminated by reset, the instruction following the one that invokes idle should not be one that writes to a port pin or to external memory.

**Figure 1. Oscillator Connections**

**Figure 2. External Clock Drive Configuration**
Power Down Mode
In the power down mode the oscillator is stopped, and the
instruction that invokes power down is the last instruction
executed. The on-chip RAM and Special Function Regis-
ters retain their values until the power down mode is liti-
tated. The only exit from power down is a hardware reset.
Halted states the SFRs but does not change the on-chip RAM.
The reset should not be activated before VCC is restored to its normal operating level and must be held active long enough to allow the oscillator to restart and sta-
bilize.

Lock Bit Protection Modes

<table>
<thead>
<tr>
<th>Program Lock Bits</th>
<th>Lock Bit Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>L01</td>
<td>L02</td>
</tr>
<tr>
<td>U</td>
<td>U</td>
</tr>
<tr>
<td>P</td>
<td>U</td>
</tr>
<tr>
<td>R</td>
<td>U</td>
</tr>
<tr>
<td>P</td>
<td>P</td>
</tr>
<tr>
<td>P</td>
<td>P</td>
</tr>
</tbody>
</table>

Programming the Flash
The AT89C51 is normally shipped with the on-chip Flash memory array initially erased.
On-chip Flash memory data is stored on 15-volt (VCC) program enable signal. The low-voltage programming mode provides a convenient way to program the
AT89C51 for user's system. While the high-voltage programming mode is compatible with conventional third
party flash or EPROM programmers.

AT89C51 is shipped with either the high-voltage or low-voltage programming mode enabled. The respective top-side marking and device signature codes are listed in the following table.

<table>
<thead>
<tr>
<th>Top-Side Mark</th>
<th>Signature</th>
</tr>
</thead>
<tbody>
<tr>
<td>AT89C51</td>
<td>(00H)</td>
</tr>
<tr>
<td>AT89C51</td>
<td>(01H)</td>
</tr>
<tr>
<td>AT89C51</td>
<td>(02H)</td>
</tr>
<tr>
<td>AT89C51</td>
<td>(03H)</td>
</tr>
<tr>
<td>AT89C51</td>
<td>(04H)</td>
</tr>
<tr>
<td>AT89C51</td>
<td>(05H)</td>
</tr>
<tr>
<td>AT89C51</td>
<td>(06H)</td>
</tr>
<tr>
<td>AT89C51</td>
<td>(07H)</td>
</tr>
</tbody>
</table>

The AT89C51 code memory array is programmed byte-by-
byte in either programming mode. In any non-
default byte in the on-chip Flash Memory, the entire memory must be erased using the Chip Erase Mode.

Program Memory Lock Bits
On the chip are three lock bits which can be left unpro-
grammed (U) or can be programmed (P) to obtain the addi-
tional features listed in the table below.

When lock bit 1 is programmed, the logic level at the EA pin
is sampled and latched during reset. If this device is pro-
grammed up without a reset, the latch initializes to a
random value, and holds that value until reset is activated. It is nec-
essary that the initial value of EA be in agreement with the
logic level at that pin in order for the device to function properly.

Program Verify: If lock bits LB1 and LB2 have not been
programmed, the programmed code data can be read back via the address and data lines for verification. The lock bits
cannot be verified directly. Verification of the lock bits is
achieved by observing that their features are enabled.

Chip Erase: The entire Flash array is erased electrically
by using the proper combination of control signals and
by holding ALE/PSEN low for 10 ms. The code array is written
with all "01H" values. The chip erase operation must be executed
before the code memory can be re-programmed.

Reading the Signature Bytes: The signature bytes are
read by the same procedure as a normal verification
of locations 00H, 03H, and 02H, except that P0.6 and P3.7
must be pulled to a logic low. The values returned are as follows.

<table>
<thead>
<tr>
<th>Mode</th>
<th>RST</th>
<th>RST</th>
<th>ALE/PROG</th>
<th>ECE</th>
<th>P2.6</th>
<th>P2.7</th>
<th>P3.6</th>
<th>P3.7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Write Code Data</td>
<td>H</td>
<td>L</td>
<td>L</td>
<td>H12V</td>
<td>L</td>
<td>H</td>
<td>H</td>
<td>H</td>
</tr>
<tr>
<td>Read Code Data</td>
<td>H</td>
<td>L</td>
<td>H</td>
<td>H12V</td>
<td>H</td>
<td>H</td>
<td>L</td>
<td>L</td>
</tr>
<tr>
<td>Write Lock</td>
<td>B1</td>
<td>H</td>
<td>L</td>
<td>H12V</td>
<td>H</td>
<td>H</td>
<td>H</td>
<td>H</td>
</tr>
<tr>
<td>B1</td>
<td>H</td>
<td>L</td>
<td></td>
<td>H12V</td>
<td>H</td>
<td>H</td>
<td>L</td>
<td>L</td>
</tr>
<tr>
<td>B1</td>
<td>H</td>
<td>L</td>
<td></td>
<td>H12V</td>
<td>H</td>
<td>H</td>
<td>L</td>
<td>L</td>
</tr>
<tr>
<td>Chip Erase</td>
<td>M</td>
<td>L</td>
<td>M</td>
<td>H12V</td>
<td>H</td>
<td>H</td>
<td>H</td>
<td>L</td>
</tr>
<tr>
<td>Read Signature Byte</td>
<td>H</td>
<td>L</td>
<td>L</td>
<td>H12V</td>
<td>H</td>
<td>H</td>
<td>H</td>
<td>H</td>
</tr>
</tbody>
</table>

Flash Programming Modes

Programming Algorithm: Before programming the
AT89C51, the address, data, and control signals should be
set up according to the Flash programming mode table and
Figure 3 and 4. To program the AT89C51, take the following
steps.
1. Input the desired memory location on the address
lines.
2. Input the appropriate data byte on the data lines.
3. Activate the correct combination of control signals.
4. Raise ECE to 12V for the high-voltage programming
mode.
5. Pulse ALE/PROG data to program a byte in the Flash
array or the lock bits. The byte write cycle is self-
timed and typically takes no more than 1.5 ms. Repeat steps
1 through 5, changing the address and data for the
entire array or until the end of the object file is reached.

Status Polling: The AT89C51 features status polling to
indicate the end of a write cycle. During a write cycle,
an attempted read of the last byte written will result in the comple-
tion of the write operation on P0.7. Once the entire write cycle has
completed, true data are valid on all outputs, and
the next cycle may begin. Status Polling may begin any time
after a write cycle has been initiated.

Ready/Busy: The progress of byte programming can also
be monitored by the P0.7/BY pin. During programming
P0.7 is pulled low after ALE is high during programming
P0.7 is pulsed high again when programming is done to indicate READY.
Figure 3. Programming the Flash

Figure 4. Verifying the Flash

Flash Programming and Verification Characteristics

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter Description</th>
<th>Min</th>
<th>Max</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>V_{pp}(T1)</td>
<td>Programming Enable Voltage</td>
<td>11.5</td>
<td>12.5</td>
<td>V</td>
</tr>
<tr>
<td>I_{P}(T1)</td>
<td>Programming Enable Current</td>
<td>0.0</td>
<td>1.0</td>
<td>mA</td>
</tr>
<tr>
<td>f_{OCL}</td>
<td>Oscillator Frequency</td>
<td>3</td>
<td>24</td>
<td>MHz</td>
</tr>
<tr>
<td>t_{WOT}</td>
<td>Address Setup to PROG Low</td>
<td>48C</td>
<td>48C</td>
<td>ns</td>
</tr>
<tr>
<td>t_{AH}</td>
<td>Address Hold After PROG</td>
<td>48C</td>
<td>48C</td>
<td>ns</td>
</tr>
<tr>
<td>t_{DTS}</td>
<td>Data Setup to PROG Low</td>
<td>48C</td>
<td>48C</td>
<td>ns</td>
</tr>
<tr>
<td>t_{DH}</td>
<td>Data Hold After PROG</td>
<td>48C</td>
<td>48C</td>
<td>ns</td>
</tr>
<tr>
<td>f_{WH}</td>
<td>P2.7 (ENABLE) High to V_{pp}</td>
<td>48C</td>
<td>48C</td>
<td>ns</td>
</tr>
<tr>
<td>t_{PCL}</td>
<td>V_{pp} Setup to PROG Low</td>
<td>10</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>t_{EH}</td>
<td>V_{pp} Hold After PROG</td>
<td>10</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>t_{OLH}</td>
<td>PROG Width</td>
<td>1</td>
<td>110</td>
<td>ns</td>
</tr>
<tr>
<td>t_{DQ0}</td>
<td>Address to Data Valid</td>
<td>48C</td>
<td>48C</td>
<td>ns</td>
</tr>
<tr>
<td>t_{DQ1}</td>
<td>ENABLE Low to Data Valid</td>
<td>48C</td>
<td>48C</td>
<td>ns</td>
</tr>
<tr>
<td>t_{DF}</td>
<td>Data Float After ENABLE</td>
<td>48C</td>
<td>48C</td>
<td>ns</td>
</tr>
<tr>
<td>t_{DH}</td>
<td>PROG High to BUSY Low</td>
<td>1.0</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>t_{OLC}</td>
<td>Byte Write Cycle Time</td>
<td>2.0</td>
<td></td>
<td>ms</td>
</tr>
</tbody>
</table>

Note: 1. Only used in 12- volt programming mode.
Absolute Maximum Ratings

- Operating Temperature: -50°C to +125°C
- Storage Temperature: -55°C to +150°C
- Voltage on Any Pin with Respect to Ground: ±1.0 V to ±7.0 V
- Maximum Operating Voltage: 6.0 V
- DC Output Current: 15.0 mA

NOTICE: Stresses beyond those listed under “Absolute Maximum Ratings” may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions beyond those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

DC Characteristics

- $T_A = +40°C$ to $+85°C$, $V_{CC} = 5.0 V ± 20%$ (unless otherwise noted)

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Condition</th>
<th>Min</th>
<th>Max</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>$V_{IL}$</td>
<td>Input Low Voltage</td>
<td>(Except $I_{CC}$)</td>
<td>-0.3</td>
<td>3.0</td>
<td>V</td>
</tr>
<tr>
<td>$V_{IH}$</td>
<td>Input High Voltage</td>
<td>(Except $I_{CC}$)</td>
<td>0.2 $V_{CC} - 0.3$</td>
<td>$V_{CC} + 0.5$</td>
<td>V</td>
</tr>
<tr>
<td>$V_{IL}$</td>
<td>Input Low Voltage</td>
<td>(VCC, RST)</td>
<td>0.2 $V_{CC} - 0.3$</td>
<td>$V_{CC} + 0.5$</td>
<td>V</td>
</tr>
<tr>
<td>$V_{OH}$</td>
<td>Output High Voltage</td>
<td>(RST, ALE, $V_{PP}$)</td>
<td>0.7 $V_{CC}$</td>
<td>$V_{CC} + 0.5$</td>
<td>V</td>
</tr>
<tr>
<td>$V_{OL}$</td>
<td>Output Low Voltage</td>
<td>(Port 0, ALE, $V_{PP}$)</td>
<td>$I_{OL} = 1.0 mA$</td>
<td>0.45</td>
<td>V</td>
</tr>
<tr>
<td>$V_{OH}$</td>
<td>Output High Voltage</td>
<td>(Port 0, ALE, $V_{PP}$)</td>
<td>$I_{OL} = 1.0 mA$</td>
<td>0.45</td>
<td>V</td>
</tr>
<tr>
<td>$V_{OL}$</td>
<td>Output Low Voltage</td>
<td>(Port 0, ALE, $V_{PP}$)</td>
<td>$I_{OL} = 1.0 mA$</td>
<td>$V_{PP} + 0.1$</td>
<td>V</td>
</tr>
</tbody>
</table>

AC Characteristics

(Under Operating Conditions: Load Capacitance for Port 0, ALE, PROG, and $PSEN = 100$ pF; Load Capacitance for all other outputs = 50 pF)

External Program and Data Memory Characteristics

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>12 MHz Oscillator</th>
<th>16 to 24 MHz Oscillator</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f_{OSC}$</td>
<td>Oscillator Frequency</td>
<td>Min</td>
<td>Max</td>
<td>Min</td>
</tr>
<tr>
<td>$I_{PP}$</td>
<td>PULSE WIDTH</td>
<td>127</td>
<td>$3_{CC}, 46$</td>
<td>ns</td>
</tr>
<tr>
<td>$I_{NC}$</td>
<td>ADDRESS VALID TO ALE LOW</td>
<td>43</td>
<td>$1_{CC}, 10$</td>
<td>ns</td>
</tr>
<tr>
<td>$I_{NC}$</td>
<td>ADDRESS HOLD AFTER ALE LOW</td>
<td>48</td>
<td>$1_{CC}, 12$</td>
<td>ns</td>
</tr>
<tr>
<td>$I_{OE}$</td>
<td>ALE LOW TO PSEN LOW</td>
<td>233</td>
<td>$4_{CC}, 46$</td>
<td>ns</td>
</tr>
<tr>
<td>$I_{OE}$</td>
<td>PSEN PULSE WIDTH</td>
<td>205</td>
<td>$1_{CC}, 12$</td>
<td>ns</td>
</tr>
<tr>
<td>$I_{OE}$</td>
<td>PSEN LOW TO VALID INSTRUCTION</td>
<td>145</td>
<td>$3_{CC}, 46$</td>
<td>ns</td>
</tr>
<tr>
<td>$I_{OE}$</td>
<td>INPUT INSTRUCTION HLD AFTER PSEN</td>
<td>0</td>
<td>0</td>
<td>ns</td>
</tr>
<tr>
<td>$I_{OE}$</td>
<td>INPUT INSTRUCTION HOLD AFTER PSEN</td>
<td>59</td>
<td>$1_{CC}, 10$</td>
<td>ns</td>
</tr>
<tr>
<td>$I_{OE}$</td>
<td>PSEN ADDRESS TO VALID</td>
<td>76</td>
<td>$1_{CC}, 8$</td>
<td>ns</td>
</tr>
<tr>
<td>$I_{OE}$</td>
<td>ADDRESS VALID TO PSEN LOW</td>
<td>312</td>
<td>$5_{CC}, 65$</td>
<td>ns</td>
</tr>
<tr>
<td>$I_{OE}$</td>
<td>PSEN LOW TO ADDRESS FLOAT</td>
<td>10</td>
<td>10</td>
<td>ns</td>
</tr>
<tr>
<td>$I_{OE}$</td>
<td>PULSE WIDTH</td>
<td>400</td>
<td>$1_{CC}, 100$</td>
<td>ns</td>
</tr>
<tr>
<td>$I_{OE}$</td>
<td>PULSE WIDTH</td>
<td>400</td>
<td>$1_{CC}, 100$</td>
<td>ns</td>
</tr>
<tr>
<td>$I_{OE}$</td>
<td>ADDRESS VALID TO DATA IN</td>
<td>252</td>
<td>$5_{CC}, 65$</td>
<td>ns</td>
</tr>
<tr>
<td>$I_{OE}$</td>
<td>DATA HOLD AFTER RST</td>
<td>0</td>
<td>0</td>
<td>ns</td>
</tr>
<tr>
<td>$I_{OE}$</td>
<td>DATA VALID TO RD</td>
<td>97</td>
<td>$2_{CC}, 26$</td>
<td>ns</td>
</tr>
<tr>
<td>$I_{OE}$</td>
<td>ALE LOW TO VALID DATA IN</td>
<td>517</td>
<td>$2_{CC}, 150$</td>
<td>ns</td>
</tr>
<tr>
<td>$I_{OE}$</td>
<td>ADDRESS VALID TO DATA IN</td>
<td>585</td>
<td>$2_{CC}, 165$</td>
<td>ns</td>
</tr>
<tr>
<td>$I_{OE}$</td>
<td>ALE LOW TO RD OR WR LOW</td>
<td>200</td>
<td>$3_{CC}, 50$</td>
<td>$3_{CC}, 50$</td>
</tr>
<tr>
<td>$I_{OE}$</td>
<td>ADDRESS TO RD OR WR LOW</td>
<td>203</td>
<td>$4_{CC}, 76$</td>
<td>ns</td>
</tr>
<tr>
<td>$I_{OE}$</td>
<td>DATA VALID TO RD TRANSITION</td>
<td>28</td>
<td>$1_{CC}, 20$</td>
<td>ns</td>
</tr>
<tr>
<td>$I_{OE}$</td>
<td>DATA VALID TO WR HIGH</td>
<td>433</td>
<td>$1_{CC}, 20$</td>
<td>ns</td>
</tr>
<tr>
<td>$I_{OE}$</td>
<td>DATA HOLD AFTER WR</td>
<td>33</td>
<td>$1_{CC}, 20$</td>
<td>ns</td>
</tr>
<tr>
<td>$I_{OE}$</td>
<td>LD LOW TO ADDRESS FLOAT</td>
<td>0</td>
<td>0</td>
<td>ns</td>
</tr>
<tr>
<td>$I_{OE}$</td>
<td>WR HIGH TO ALE HIGH</td>
<td>43</td>
<td>$1_{CC}, 20$</td>
<td>$1_{CC}, 25$</td>
</tr>
</tbody>
</table>

Notes:
1. Under steady state (non- teaching) conditions, $I_{CC}$ must be externally limited as follows:
   - Maximum $I_{CC}$ per port pin: 10 mA
   - Maximum $I_{CC}$ per 8-bit port: 40 mA
   - Maximum $I_{CC}$ for all output pins: 71 mA
   - If $I_{CC}$ exceeds the test condition, $V_{CC}$ may exceed the related specification. Pins are not guaranteed to sink current greater than the listed test conditions.
2. Minimum $V_{CC}$ for Power Down is 2V.
Serial Port Timing: Shift Register Mode Test Conditions
(V_{CC} = 5.0 V ± 20%, Load Capacitance = 80 pF)

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>12 MHz Osc</th>
<th>Variable Oscillator</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>t_{DCL}</td>
<td>Serial Port Clock Cycle Time</td>
<td>1.0</td>
<td>125 CL</td>
<td>ns</td>
</tr>
<tr>
<td>t_{DCH}</td>
<td>Output Data Setup to Clock Rising Edge</td>
<td>760</td>
<td>100 CL, 133</td>
<td>ns</td>
</tr>
<tr>
<td>t_{DH}</td>
<td>Output Data Hold After Clock Rising Edge</td>
<td>50</td>
<td>2 CL, 117</td>
<td>ns</td>
</tr>
<tr>
<td>t_{D}</td>
<td>Input Data Hold After Clock Rising Edge</td>
<td>0</td>
<td>0</td>
<td>ns</td>
</tr>
<tr>
<td>t_{R}</td>
<td>Clock Rising Edge to Input Data Valid</td>
<td>760</td>
<td>100 CL, 133</td>
<td>ns</td>
</tr>
</tbody>
</table>

Shift Register Mode Timing Waveforms

AC Testing Input/Output Waveforms

Float Waveforms

Note: 1. AC Inputs during testing are driven at V_{CC} = 0.5V for a logic 0 and 0.45V for a logic 0. Timing measurements are made at V_{TH} = max. for a logic 1 and V_{OL} = max. for a logic 0.

Ordering Information

<table>
<thead>
<tr>
<th>Speed (MHz)</th>
<th>Power Supply</th>
<th>Ordering Code</th>
<th>Package</th>
<th>Operation Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>5V ± 20%</td>
<td>AT99C51-12AC</td>
<td>44A</td>
<td>Commercial (0°C to 70°C)</td>
</tr>
<tr>
<td>16</td>
<td>5V ± 20%</td>
<td>AT99C51-16AC</td>
<td>44A</td>
<td>Commercial (0°C to 70°C)</td>
</tr>
<tr>
<td>20</td>
<td>5V ± 20%</td>
<td>AT99C51-20AC</td>
<td>44A</td>
<td>Commercial (0°C to 70°C)</td>
</tr>
</tbody>
</table>

Note: 1. For timing purposes, a port pin is no longer floating when a 100 mV change from load-voltage occurs. A port pin begins to float when 100 mV change from the loaded V_{OL}/V_{TH} level occurs.
AT89C51

Ordering Information

<table>
<thead>
<tr>
<th>Speed (MHz)</th>
<th>Power Supply</th>
<th>Ordering Code</th>
<th>Package</th>
<th>Operation Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>24</td>
<td>5V ± 20%</td>
<td>AT89C51-24A</td>
<td>44A</td>
<td>Commercial</td>
</tr>
<tr>
<td></td>
<td></td>
<td>AT89C51-24J</td>
<td>44J</td>
<td>0°C to 70°C</td>
</tr>
<tr>
<td></td>
<td></td>
<td>AT89C51-24JC</td>
<td>44P</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>AT89C51-24B</td>
<td>44Q</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>AT89C51-24B4</td>
<td>44A</td>
<td>Industrial</td>
</tr>
<tr>
<td></td>
<td></td>
<td>AT89C51-24B4C</td>
<td>44J</td>
<td>(-40°C to 85°C)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>AT89C51-24B8</td>
<td>44P</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>AT89C51-24B8C</td>
<td>44Q</td>
<td></td>
</tr>
</tbody>
</table>

Microcontroller Instruction Set

For interrupt response time information, refer to the hardware description chapter.

Instructions that Affect Flag Settings

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Flag</th>
<th>Instruction</th>
<th>Flag</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD</td>
<td>X</td>
<td>ADDC</td>
<td>X</td>
</tr>
<tr>
<td>SUBB</td>
<td>X</td>
<td>X</td>
<td>ANL</td>
</tr>
<tr>
<td>ROL</td>
<td>D</td>
<td>O</td>
<td>ANL</td>
</tr>
<tr>
<td>MOV</td>
<td>X</td>
<td>O</td>
<td>ANL</td>
</tr>
<tr>
<td>DA</td>
<td>X</td>
<td>DBL</td>
<td>C</td>
</tr>
<tr>
<td>RNC</td>
<td>X</td>
<td>MOV</td>
<td>X</td>
</tr>
<tr>
<td>RLCP</td>
<td>X</td>
<td>CINE</td>
<td>X</td>
</tr>
</tbody>
</table>

Note: 1. Operations on 01H byte address 200H or bit addresses 208:21F (that is, the PSW or bits in the PSW) also affect flag settings.

The Instruction Set and Addressing Modes

Ri    Register Ri of the currently selected Register Bank.

direct 8-bit internal data location's address. This could be an Internal Data RAM location (0-127) or a SFR (i.e., I/O port, control register, status register, etc. 128-255).

[R] 8-bit internal data RAM location (0-255) addressed indirectly through register Rx (R6).

#data 8-bit constant included in instruction.

#data 16 16-bit constant included in instruction.

addr 16 16-bit destination address. Used by CALL and JMP. A branch can be anywhere within the 64K-byte Program Memory address space.

addr 14 14-bit destination address. Used by ACALL and AJMP. The branch will be within the same 2K-byte page of program memory as the first byte of the following instruction.

rel Signed two's complement 8-bit offset byte. Used by JUMPL and all conditional jumps. Range is -128 to +127 bytes relative to first byte of the following instruction.

bit Direct Addressed bit in Internal Data RAM or Special Function Register.
Instruction Set Summary

<table>
<thead>
<tr>
<th>Instruction</th>
<th>E</th>
<th>D</th>
<th>C</th>
<th>B</th>
<th>A</th>
<th>R</th>
<th>M</th>
</tr>
</thead>
<tbody>
<tr>
<td>NOP</td>
<td>D0</td>
<td>D1</td>
<td>D2</td>
<td>D3</td>
<td>D4</td>
<td>D5</td>
<td>D6</td>
</tr>
<tr>
<td>INC A</td>
<td>E0</td>
<td>E1</td>
<td>E2</td>
<td>E3</td>
<td>E4</td>
<td>E5</td>
<td>E6</td>
</tr>
<tr>
<td>DEC A</td>
<td>E7</td>
<td>E8</td>
<td>E9</td>
<td>EA</td>
<td>EB</td>
<td>EC</td>
<td>ED</td>
</tr>
<tr>
<td>ADD A A</td>
<td>F0</td>
<td>F1</td>
<td>F2</td>
<td>F3</td>
<td>F4</td>
<td>F5</td>
<td>F6</td>
</tr>
<tr>
<td>SUB A A</td>
<td>F7</td>
<td>F8</td>
<td>F9</td>
<td>FA</td>
<td>FB</td>
<td>FC</td>
<td>FD</td>
</tr>
</tbody>
</table>

Note: Key: [D0] = 2 Byte, [D1] = 3 Byte, [D2] = 2 Cycle, [D3] = 4 Cycle, Blank = 1 Byte/1 Cycle

Instruction Set

<table>
<thead>
<tr>
<th>Instruction</th>
<th>E</th>
<th>D</th>
<th>C</th>
<th>B</th>
<th>A</th>
<th>R</th>
<th>M</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOV A,A</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>ADD A A</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SUB A A</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Note: Key: [D0] = 2 Byte, [D1] = 3 Byte, [D2] = 2 Cycle, [D3] = 4 Cycle, Blank = 1 Byte/1 Cycle
**Table 1. AT89 Instruction Set Summary**

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Description</th>
<th>Byte</th>
<th>Oscillator Period</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD A, Rn</td>
<td>Add register to Accumulator</td>
<td>1</td>
<td>12</td>
</tr>
<tr>
<td>ADD A, direct</td>
<td>Add direct byte to Accumulator</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>ADD A, @Ri</td>
<td>Add indirect RAM to Accumulator</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>ADD A, @RiX</td>
<td>Add indirect RAM to Accumulator with Carry</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>ADD A, @data</td>
<td>Add immediate data to Accumulator</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>ADDC A, Rn</td>
<td>Add register to Accumulator with Carry</td>
<td>1</td>
<td>12</td>
</tr>
<tr>
<td>ADDC A, direct</td>
<td>Add direct byte to Accumulator with Carry</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>ADDC A, @Ri</td>
<td>Add indirect RAM to Accumulator with Carry</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>ADDC A, @RiX</td>
<td>Add indirect RAM to Accumulator with Carry</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>ADDC A, @data</td>
<td>Add immediate data to Accumulator with Carry</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>SUBB A, Rn</td>
<td>Subtract Register from Accumulator with borrow</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>SUBB A, direct</td>
<td>Subtract direct byte from Accumulator with borrow</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>SUBB A, @Ri</td>
<td>Subtract indirect RAM from Accumulator with borrow</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>SUBB A, @RiX</td>
<td>Subtract indirect RAM from Accumulator with borrow</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>INC A</td>
<td>Increment Accumulator</td>
<td>1</td>
<td>12</td>
</tr>
<tr>
<td>INC Rn</td>
<td>Increment register</td>
<td>1</td>
<td>12</td>
</tr>
<tr>
<td>INC direct</td>
<td>Increment direct byte</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>DEC A</td>
<td>Decrement Accumulator</td>
<td>1</td>
<td>12</td>
</tr>
<tr>
<td>DEC Rn</td>
<td>Decrement register</td>
<td>1</td>
<td>12</td>
</tr>
<tr>
<td>DEC direct</td>
<td>Decrement direct byte</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>DEC @Ri</td>
<td>Decrement indirect RAM</td>
<td>1</td>
<td>12</td>
</tr>
<tr>
<td>DEC @RiX</td>
<td>Decrement indirect RAM with Carry</td>
<td>1</td>
<td>12</td>
</tr>
<tr>
<td>MUL AB</td>
<td>Multiply A &amp; B</td>
<td>1</td>
<td>12</td>
</tr>
<tr>
<td>SHL AB</td>
<td>Shifl A by B</td>
<td>1</td>
<td>12</td>
</tr>
<tr>
<td>DAA</td>
<td>Decimal Adjust Accumulator</td>
<td>1</td>
<td>12</td>
</tr>
<tr>
<td>RL A</td>
<td>Rotate Accumulator Left</td>
<td>1</td>
<td>12</td>
</tr>
<tr>
<td>RLC A</td>
<td>Rotate Accumulator Left through the Carry</td>
<td>1</td>
<td>12</td>
</tr>
<tr>
<td>LOGICAL OPERATIONS (continued)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Description</th>
<th>Byte</th>
<th>Oscillator Period</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANL A, Rn</td>
<td>AND Register to Accumulator</td>
<td>1</td>
<td>12</td>
</tr>
<tr>
<td>ANL A, direct</td>
<td>AND direct byte to Accumulator</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>ANL A, @Ri</td>
<td>AND indirect RAM to Accumulator</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>ANL A, @RiX</td>
<td>AND indirect RAM to Accumulator with Carry</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>ANL A, @data</td>
<td>AND immediate data to Accumulator</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>ORL A, Rn</td>
<td>OR Register to Accumulator</td>
<td>1</td>
<td>12</td>
</tr>
<tr>
<td>ORL A, direct</td>
<td>OR direct byte to Accumulator</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>ORL A, @Ri</td>
<td>OR indirect RAM to Accumulator</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>ORL A, @RiX</td>
<td>OR indirect RAM to Accumulator with Carry</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>ORL A, @data</td>
<td>OR immediate data to Accumulator</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>XOR A, Rn</td>
<td>XOR Register to Accumulator</td>
<td>1</td>
<td>12</td>
</tr>
<tr>
<td>XOR A, direct</td>
<td>XOR direct byte to Accumulator</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>XOR A, @Ri</td>
<td>XOR indirect RAM to Accumulator</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>XOR A, @RiX</td>
<td>XOR indirect RAM to Accumulator with Carry</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>XOR A, @data</td>
<td>XOR immediate data to Accumulator</td>
<td>2</td>
<td>12</td>
</tr>
</tbody>
</table>

**DATA TRANSFER**

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Description</th>
<th>Byte</th>
<th>Oscillator Period</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOV A, Rn</td>
<td>Move register to accumulator</td>
<td>1</td>
<td>12</td>
</tr>
<tr>
<td>MOV A, direct</td>
<td>Move direct byte to accumulator</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>MOV A, @Ri</td>
<td>Move indirect RAM to accumulator</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>MOV A, @RiX</td>
<td>Move indirect RAM to accumulator with Carry</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>MOV A, @data</td>
<td>Move immediate data to accumulator</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>MOV @Ri, A</td>
<td>Move accumulator to register</td>
<td>1</td>
<td>12</td>
</tr>
<tr>
<td>MOV @RiX, A</td>
<td>Move accumulator to register with Carry</td>
<td>1</td>
<td>12</td>
</tr>
<tr>
<td>MOV @data, A</td>
<td>Move immediate data to register</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>MOV direct, A</td>
<td>Move accumulator to direct byte</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>MOV direct, Rn</td>
<td>Move direct byte to direct byte</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>MOV direct, @Ri</td>
<td>Move direct data to indirect register</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>MOV @Ri, direct</td>
<td>Move indirect RAM to direct byte</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>MOV @RiX, direct</td>
<td>Move indirect RAM to direct byte with Carry</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>MOV @data, direct</td>
<td>Move immediate data to direct byte</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>MOV @Ri, @Ri</td>
<td>Move indirect RAM to indirect RAM</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>MOV @RiX, @Ri</td>
<td>Move indirect RAM to indirect RAM with Carry</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>MOV @data, @Ri</td>
<td>Move immediate data to indirect RAM</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>MOV @Ri, @data</td>
<td>Move indirect RAM to immediate data</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>MOV @RiX, @data</td>
<td>Move indirect RAM to immediate data with Carry</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>MOV @data, @data</td>
<td>Move immediate data to immediate data</td>
<td>2</td>
<td>12</td>
</tr>
</tbody>
</table>

**DOORS CAN VARIABLE MANIPULATION**

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Description</th>
<th>Byte</th>
<th>Oscillator Period</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLR</td>
<td>Clear Carry</td>
<td>1</td>
<td>12</td>
</tr>
<tr>
<td>SR</td>
<td>Set</td>
<td>1</td>
<td>12</td>
</tr>
<tr>
<td>SCL</td>
<td>Set Carry</td>
<td>1</td>
<td>12</td>
</tr>
<tr>
<td>CPL</td>
<td>Complement</td>
<td>1</td>
<td>12</td>
</tr>
<tr>
<td>CLC</td>
<td>Clear Carry</td>
<td>1</td>
<td>12</td>
</tr>
<tr>
<td>ANL</td>
<td>AND indirect RAM to accumulator</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>ORL</td>
<td>OR indirect RAM to accumulator</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>XOR</td>
<td>XOR indirect RAM to accumulator</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>NOT</td>
<td>Complement indirect RAM</td>
<td>1</td>
<td>12</td>
</tr>
</tbody>
</table>

**PROGRAM BRANCHING**

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Description</th>
<th>Byte</th>
<th>Oscillator Period</th>
</tr>
</thead>
<tbody>
<tr>
<td>JBC</td>
<td>Jump if Carry is set</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>JNC</td>
<td>Jump if Carry is not set</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>JB</td>
<td>Jump if Bit is set</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>JNB</td>
<td>Jump if Bit is not set</td>
<td>2</td>
<td>12</td>
</tr>
</tbody>
</table>

Note: All mnemonics copyrighted © Intel Corp., 1980.
Table 2. Instruction Opcodes in Hexadecimal Order

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Number of Bytes</th>
<th>Mnemonic</th>
<th>Operands</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>1</td>
<td>NOP</td>
<td></td>
</tr>
<tr>
<td>01</td>
<td>2</td>
<td>ADD</td>
<td>A, B, D</td>
</tr>
<tr>
<td>02</td>
<td>3</td>
<td>JMP</td>
<td>code addr</td>
</tr>
<tr>
<td>03</td>
<td>1</td>
<td>INC</td>
<td>A</td>
</tr>
<tr>
<td>04</td>
<td>2</td>
<td>INC</td>
<td>data addr</td>
</tr>
<tr>
<td>05</td>
<td>1</td>
<td>INC</td>
<td>R4</td>
</tr>
<tr>
<td>06</td>
<td>1</td>
<td>INC</td>
<td>R2</td>
</tr>
<tr>
<td>07</td>
<td>1</td>
<td>INC</td>
<td>R1</td>
</tr>
<tr>
<td>08</td>
<td>1</td>
<td>INC</td>
<td>R0</td>
</tr>
<tr>
<td>09</td>
<td>1</td>
<td>INC</td>
<td>R1</td>
</tr>
<tr>
<td>0A</td>
<td>1</td>
<td>INC</td>
<td>R2</td>
</tr>
<tr>
<td>0B</td>
<td>1</td>
<td>INC</td>
<td>R3</td>
</tr>
<tr>
<td>0C</td>
<td>1</td>
<td>INC</td>
<td>R4</td>
</tr>
<tr>
<td>0D</td>
<td>1</td>
<td>INC</td>
<td>R5</td>
</tr>
<tr>
<td>0E</td>
<td>1</td>
<td>INC</td>
<td>R6</td>
</tr>
<tr>
<td>0F</td>
<td>1</td>
<td>INC</td>
<td>R7</td>
</tr>
<tr>
<td>10</td>
<td>3</td>
<td>JSR</td>
<td>bit addr, code addr</td>
</tr>
<tr>
<td>11</td>
<td>2</td>
<td>CALL</td>
<td>code addr</td>
</tr>
<tr>
<td>12</td>
<td>3</td>
<td>LCALL</td>
<td>code addr</td>
</tr>
<tr>
<td>13</td>
<td>1</td>
<td>RNC</td>
<td>A</td>
</tr>
<tr>
<td>14</td>
<td>1</td>
<td>DEC</td>
<td>A</td>
</tr>
<tr>
<td>15</td>
<td>1</td>
<td>DEC</td>
<td>R4</td>
</tr>
<tr>
<td>16</td>
<td>1</td>
<td>DEC</td>
<td>R5</td>
</tr>
<tr>
<td>17</td>
<td>1</td>
<td>DEC</td>
<td>R6</td>
</tr>
<tr>
<td>18</td>
<td>1</td>
<td>DEC</td>
<td>R7</td>
</tr>
<tr>
<td>19</td>
<td>1</td>
<td>DEC</td>
<td>R2</td>
</tr>
<tr>
<td>1A</td>
<td>1</td>
<td>DEC</td>
<td>R3</td>
</tr>
<tr>
<td>1B</td>
<td>1</td>
<td>DEC</td>
<td>R4</td>
</tr>
<tr>
<td>1C</td>
<td>1</td>
<td>DEC</td>
<td>R5</td>
</tr>
<tr>
<td>1D</td>
<td>1</td>
<td>DEC</td>
<td>R6</td>
</tr>
<tr>
<td>1E</td>
<td>1</td>
<td>DEC</td>
<td>R7</td>
</tr>
<tr>
<td>1F</td>
<td>1</td>
<td>DEC</td>
<td>R1</td>
</tr>
<tr>
<td>20</td>
<td>2</td>
<td>JE</td>
<td>code addr</td>
</tr>
<tr>
<td>21</td>
<td>2</td>
<td>JMP</td>
<td>code addr</td>
</tr>
<tr>
<td>22</td>
<td>1</td>
<td>RCT</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>1</td>
<td>RL</td>
<td>A</td>
</tr>
<tr>
<td>24</td>
<td>2</td>
<td>ADD</td>
<td>A, addr</td>
</tr>
<tr>
<td>25</td>
<td>2</td>
<td>ADD</td>
<td>A, data</td>
</tr>
</tbody>
</table>

Note: Opcodes 26 to 3F are reserved for future use.
### Instruction Set

<table>
<thead>
<tr>
<th>Exec Code</th>
<th>Number of Bytes</th>
<th>Mnemonic</th>
<th>Operands</th>
</tr>
</thead>
<tbody>
<tr>
<td>40</td>
<td>1</td>
<td>ORL</td>
<td>A, R5</td>
</tr>
<tr>
<td>41</td>
<td>2</td>
<td>ACALL</td>
<td>code addr</td>
</tr>
<tr>
<td>42</td>
<td>2</td>
<td>ORL</td>
<td>A, R4</td>
</tr>
<tr>
<td>43</td>
<td>2</td>
<td>JNP</td>
<td>@A+PC</td>
</tr>
<tr>
<td>44</td>
<td>2</td>
<td>MOV</td>
<td>A, data addr</td>
</tr>
<tr>
<td>45</td>
<td>2</td>
<td>MOV</td>
<td>R3, data addr</td>
</tr>
<tr>
<td>46</td>
<td>2</td>
<td>JNC</td>
<td>code addr</td>
</tr>
<tr>
<td>47</td>
<td>2</td>
<td>ANL</td>
<td>data addr</td>
</tr>
<tr>
<td>48</td>
<td>2</td>
<td>ANL</td>
<td>A, data addr</td>
</tr>
<tr>
<td>49</td>
<td>2</td>
<td>ANL</td>
<td>A, data addr</td>
</tr>
<tr>
<td>4A</td>
<td>2</td>
<td>ANL</td>
<td>A, R3</td>
</tr>
<tr>
<td>4B</td>
<td>2</td>
<td>ANL</td>
<td>A, R2</td>
</tr>
<tr>
<td>4C</td>
<td>2</td>
<td>ANL</td>
<td>A, R1</td>
</tr>
<tr>
<td>4D</td>
<td>2</td>
<td>ANL</td>
<td>A, R0</td>
</tr>
<tr>
<td>4E</td>
<td>2</td>
<td>ANL</td>
<td>A, @R1</td>
</tr>
<tr>
<td>4F</td>
<td>2</td>
<td>ANL</td>
<td>A, @R0</td>
</tr>
<tr>
<td>50</td>
<td>2</td>
<td>JZ</td>
<td>code addr</td>
</tr>
<tr>
<td>51</td>
<td>2</td>
<td>AJMP</td>
<td>code addr</td>
</tr>
<tr>
<td>52</td>
<td>2</td>
<td>XRL</td>
<td>code addr</td>
</tr>
<tr>
<td>53</td>
<td>2</td>
<td>XRL</td>
<td>data addr</td>
</tr>
<tr>
<td>54</td>
<td>2</td>
<td>XRL</td>
<td>data addr</td>
</tr>
<tr>
<td>55</td>
<td>2</td>
<td>XRL</td>
<td>data addr</td>
</tr>
<tr>
<td>56</td>
<td>2</td>
<td>XRL</td>
<td>data addr</td>
</tr>
<tr>
<td>57</td>
<td>2</td>
<td>XRL</td>
<td>data addr</td>
</tr>
<tr>
<td>58</td>
<td>2</td>
<td>XRL</td>
<td>data addr</td>
</tr>
<tr>
<td>59</td>
<td>2</td>
<td>XRL</td>
<td>data addr</td>
</tr>
<tr>
<td>5A</td>
<td>2</td>
<td>XRL</td>
<td>data addr</td>
</tr>
<tr>
<td>5B</td>
<td>2</td>
<td>XRL</td>
<td>data addr</td>
</tr>
<tr>
<td>5C</td>
<td>2</td>
<td>XRL</td>
<td>data addr</td>
</tr>
<tr>
<td>5D</td>
<td>2</td>
<td>XRL</td>
<td>data addr</td>
</tr>
<tr>
<td>5E</td>
<td>2</td>
<td>XRL</td>
<td>data addr</td>
</tr>
<tr>
<td>5F</td>
<td>2</td>
<td>XRL</td>
<td>data addr</td>
</tr>
<tr>
<td>60</td>
<td>2</td>
<td>JZ</td>
<td>code addr</td>
</tr>
<tr>
<td>61</td>
<td>2</td>
<td>JSR</td>
<td>@R0</td>
</tr>
<tr>
<td>62</td>
<td>2</td>
<td>MOV</td>
<td>A, data addr</td>
</tr>
<tr>
<td>63</td>
<td>2</td>
<td>MOV</td>
<td>R3, data addr</td>
</tr>
<tr>
<td>64</td>
<td>2</td>
<td>MOV</td>
<td>R3, data addr</td>
</tr>
<tr>
<td>65</td>
<td>2</td>
<td>MOV</td>
<td>R3, data addr</td>
</tr>
<tr>
<td>66</td>
<td>2</td>
<td>MOV</td>
<td>R3, data addr</td>
</tr>
<tr>
<td>67</td>
<td>2</td>
<td>MOV</td>
<td>R3, data addr</td>
</tr>
<tr>
<td>68</td>
<td>2</td>
<td>MOV</td>
<td>R3, data addr</td>
</tr>
<tr>
<td>69</td>
<td>2</td>
<td>MOV</td>
<td>R3, data addr</td>
</tr>
<tr>
<td>6A</td>
<td>2</td>
<td>MOV</td>
<td>R3, data addr</td>
</tr>
<tr>
<td>6B</td>
<td>2</td>
<td>MOV</td>
<td>R3, data addr</td>
</tr>
<tr>
<td>6C</td>
<td>2</td>
<td>MOV</td>
<td>R3, data addr</td>
</tr>
<tr>
<td>6D</td>
<td>2</td>
<td>MOV</td>
<td>R3, data addr</td>
</tr>
<tr>
<td>6E</td>
<td>2</td>
<td>MOV</td>
<td>R3, data addr</td>
</tr>
<tr>
<td>6F</td>
<td>2</td>
<td>MOV</td>
<td>R3, data addr</td>
</tr>
<tr>
<td>70</td>
<td>2</td>
<td>JNZ</td>
<td>code addr</td>
</tr>
<tr>
<td>71</td>
<td>2</td>
<td>SUBB</td>
<td>A, R1</td>
</tr>
<tr>
<td>72</td>
<td>2</td>
<td>SUBB</td>
<td>A, R0</td>
</tr>
<tr>
<td>73</td>
<td>2</td>
<td>SUBB</td>
<td>A, R1</td>
</tr>
<tr>
<td>74</td>
<td>2</td>
<td>SUBB</td>
<td>A, R0</td>
</tr>
<tr>
<td>75</td>
<td>2</td>
<td>SUBB</td>
<td>A, R1</td>
</tr>
<tr>
<td>76</td>
<td>2</td>
<td>SUBB</td>
<td>A, R0</td>
</tr>
<tr>
<td>77</td>
<td>2</td>
<td>SUBB</td>
<td>A, R1</td>
</tr>
<tr>
<td>78</td>
<td>2</td>
<td>SUBB</td>
<td>A, R0</td>
</tr>
</tbody>
</table>

---

### Instruction Set

<table>
<thead>
<tr>
<th>Exec Code</th>
<th>Number of Bytes</th>
<th>Mnemonic</th>
<th>Operands</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>1</td>
<td>SUBB</td>
<td>A, @R1</td>
</tr>
<tr>
<td>01</td>
<td>1</td>
<td>SUBB</td>
<td>A, @R0</td>
</tr>
<tr>
<td>02</td>
<td>1</td>
<td>SUBB</td>
<td>R3, @R1</td>
</tr>
<tr>
<td>03</td>
<td>1</td>
<td>SUBB</td>
<td>R3, @R0</td>
</tr>
<tr>
<td>04</td>
<td>1</td>
<td>SUBB</td>
<td>R3, @R1</td>
</tr>
<tr>
<td>05</td>
<td>1</td>
<td>SUBB</td>
<td>R3, @R0</td>
</tr>
<tr>
<td>06</td>
<td>1</td>
<td>SUBB</td>
<td>R3, @R1</td>
</tr>
<tr>
<td>07</td>
<td>1</td>
<td>SUBB</td>
<td>R3, @R0</td>
</tr>
</tbody>
</table>

---

### Instruction Set

<table>
<thead>
<tr>
<th>Exec Code</th>
<th>Number of Bytes</th>
<th>Mnemonic</th>
<th>Operands</th>
</tr>
</thead>
<tbody>
<tr>
<td>40</td>
<td>1</td>
<td>ORL</td>
<td>A, R5</td>
</tr>
<tr>
<td>41</td>
<td>2</td>
<td>ACALL</td>
<td>code addr</td>
</tr>
<tr>
<td>42</td>
<td>2</td>
<td>ORL</td>
<td>A, R4</td>
</tr>
<tr>
<td>43</td>
<td>2</td>
<td>JNP</td>
<td>@A+PC</td>
</tr>
<tr>
<td>44</td>
<td>2</td>
<td>MOV</td>
<td>A, data addr</td>
</tr>
<tr>
<td>45</td>
<td>2</td>
<td>MOV</td>
<td>R3, data addr</td>
</tr>
<tr>
<td>46</td>
<td>2</td>
<td>JNC</td>
<td>code addr</td>
</tr>
<tr>
<td>47</td>
<td>2</td>
<td>ANL</td>
<td>data addr</td>
</tr>
<tr>
<td>48</td>
<td>2</td>
<td>ANL</td>
<td>A, data addr</td>
</tr>
<tr>
<td>49</td>
<td>2</td>
<td>ANL</td>
<td>A, data addr</td>
</tr>
<tr>
<td>4A</td>
<td>2</td>
<td>ANL</td>
<td>A, R3</td>
</tr>
<tr>
<td>4B</td>
<td>2</td>
<td>ANL</td>
<td>A, R2</td>
</tr>
<tr>
<td>4C</td>
<td>2</td>
<td>ANL</td>
<td>A, R1</td>
</tr>
<tr>
<td>4D</td>
<td>2</td>
<td>ANL</td>
<td>A, R0</td>
</tr>
<tr>
<td>4E</td>
<td>2</td>
<td>ANL</td>
<td>A, @R1</td>
</tr>
<tr>
<td>4F</td>
<td>2</td>
<td>ANL</td>
<td>A, @R0</td>
</tr>
<tr>
<td>50</td>
<td>2</td>
<td>JZ</td>
<td>code addr</td>
</tr>
<tr>
<td>51</td>
<td>2</td>
<td>AJMP</td>
<td>code addr</td>
</tr>
<tr>
<td>52</td>
<td>2</td>
<td>XRL</td>
<td>code addr</td>
</tr>
<tr>
<td>53</td>
<td>2</td>
<td>XRL</td>
<td>data addr</td>
</tr>
<tr>
<td>54</td>
<td>2</td>
<td>XRL</td>
<td>data addr</td>
</tr>
<tr>
<td>55</td>
<td>2</td>
<td>XRL</td>
<td>data addr</td>
</tr>
<tr>
<td>56</td>
<td>2</td>
<td>XRL</td>
<td>data addr</td>
</tr>
<tr>
<td>57</td>
<td>2</td>
<td>XRL</td>
<td>data addr</td>
</tr>
<tr>
<td>58</td>
<td>2</td>
<td>XRL</td>
<td>data addr</td>
</tr>
<tr>
<td>59</td>
<td>2</td>
<td>XRL</td>
<td>data addr</td>
</tr>
<tr>
<td>5A</td>
<td>2</td>
<td>XRL</td>
<td>data addr</td>
</tr>
<tr>
<td>5B</td>
<td>2</td>
<td>XRL</td>
<td>data addr</td>
</tr>
<tr>
<td>5C</td>
<td>2</td>
<td>XRL</td>
<td>data addr</td>
</tr>
<tr>
<td>5D</td>
<td>2</td>
<td>XRL</td>
<td>data addr</td>
</tr>
<tr>
<td>5E</td>
<td>2</td>
<td>XRL</td>
<td>data addr</td>
</tr>
<tr>
<td>5F</td>
<td>2</td>
<td>XRL</td>
<td>data addr</td>
</tr>
<tr>
<td>60</td>
<td>2</td>
<td>JZ</td>
<td>code addr</td>
</tr>
<tr>
<td>61</td>
<td>2</td>
<td>JSR</td>
<td>@R0</td>
</tr>
<tr>
<td>62</td>
<td>2</td>
<td>MOV</td>
<td>A, data addr</td>
</tr>
<tr>
<td>63</td>
<td>2</td>
<td>MOV</td>
<td>R3, data addr</td>
</tr>
<tr>
<td>64</td>
<td>2</td>
<td>MOV</td>
<td>R3, data addr</td>
</tr>
<tr>
<td>65</td>
<td>2</td>
<td>MOV</td>
<td>R3, data addr</td>
</tr>
<tr>
<td>66</td>
<td>2</td>
<td>MOV</td>
<td>R3, data addr</td>
</tr>
<tr>
<td>67</td>
<td>2</td>
<td>MOV</td>
<td>R3, data addr</td>
</tr>
<tr>
<td>68</td>
<td>2</td>
<td>MOV</td>
<td>R3, data addr</td>
</tr>
<tr>
<td>69</td>
<td>2</td>
<td>MOV</td>
<td>R3, data addr</td>
</tr>
<tr>
<td>6A</td>
<td>2</td>
<td>MOV</td>
<td>R3, data addr</td>
</tr>
<tr>
<td>6B</td>
<td>2</td>
<td>MOV</td>
<td>R3, data addr</td>
</tr>
<tr>
<td>6C</td>
<td>2</td>
<td>MOV</td>
<td>R3, data addr</td>
</tr>
<tr>
<td>6D</td>
<td>2</td>
<td>MOV</td>
<td>R3, data addr</td>
</tr>
<tr>
<td>6E</td>
<td>2</td>
<td>MOV</td>
<td>R3, data addr</td>
</tr>
<tr>
<td>6F</td>
<td>2</td>
<td>MOV</td>
<td>R3, data addr</td>
</tr>
<tr>
<td>70</td>
<td>2</td>
<td>JNZ</td>
<td>code addr</td>
</tr>
<tr>
<td>71</td>
<td>2</td>
<td>SUBB</td>
<td>A, R1</td>
</tr>
<tr>
<td>72</td>
<td>2</td>
<td>SUBB</td>
<td>A, R0</td>
</tr>
<tr>
<td>73</td>
<td>2</td>
<td>SUBB</td>
<td>A, R1</td>
</tr>
<tr>
<td>74</td>
<td>2</td>
<td>SUBB</td>
<td>A, R0</td>
</tr>
<tr>
<td>75</td>
<td>2</td>
<td>SUBB</td>
<td>A, R1</td>
</tr>
<tr>
<td>76</td>
<td>2</td>
<td>SUBB</td>
<td>A, R0</td>
</tr>
<tr>
<td>77</td>
<td>2</td>
<td>SUBB</td>
<td>A, R1</td>
</tr>
<tr>
<td>78</td>
<td>2</td>
<td>SUBB</td>
<td>A, R0</td>
</tr>
</tbody>
</table>
### Instruction Set

#### Instruction Definitions

**ACALL addr11**

**Function:** Absolute Call

**Description:** ACALL unconditionally calls a subroutine located at the indicated address. The instruction increments the PC next to obtain the address of the following instruction, then pushes the 16-bit result onto the stack (low-order byte first) and increments the Stack Pointer twice. The destination address is obtained by successively concatenating the five high-order bits of the incremented PC, opcode byte 7 through 5, and the second byte of the instruction. The subroutine called must therefore start within the same 2 K block of the program memory as the first byte of the instruction following ACALL. No flags are affected.

**Example:** Initially SP equals 08H. The label SUBRND is at program memory location 04H. After executing the following instruction:

ACALL SUBRND

at location 012H, SP contains 08H, internal RAM locations 08H and 09H will contain 28H and 01H, respectively, and the PC contains 024H.

**Bytes:** 2

**Cycles:** 2

**Encoding:** a10 a11 0 0 0 0 1 0 0 0 0 0

**Operation:**

<table>
<thead>
<tr>
<th>OPC</th>
<th>R Cy</th>
<th>R Cy</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACALL</td>
<td>00H</td>
<td>addr11</td>
</tr>
</tbody>
</table>

---

### Table

<table>
<thead>
<tr>
<th>Hex Code</th>
<th>Number of Bytes</th>
<th>Mnemonic</th>
<th>Operands</th>
</tr>
</thead>
<tbody>
<tr>
<td>E3</td>
<td>1</td>
<td>MOVX</td>
<td>0,R1</td>
</tr>
<tr>
<td>E4</td>
<td>1</td>
<td>MOVX</td>
<td>A</td>
</tr>
<tr>
<td>E5</td>
<td>2</td>
<td>MOV</td>
<td>0,data addr</td>
</tr>
<tr>
<td>E6</td>
<td>1</td>
<td>MOV</td>
<td>0,R0</td>
</tr>
<tr>
<td>E7</td>
<td>1</td>
<td>MOV</td>
<td>0,R1</td>
</tr>
<tr>
<td>E8</td>
<td>1</td>
<td>MOV</td>
<td>0,R2</td>
</tr>
<tr>
<td>E9</td>
<td>1</td>
<td>MOV</td>
<td>0,R3</td>
</tr>
<tr>
<td>EA</td>
<td>1</td>
<td>MOV</td>
<td>0,R4</td>
</tr>
<tr>
<td>EB</td>
<td>1</td>
<td>MOV</td>
<td>0,R5</td>
</tr>
<tr>
<td>EC</td>
<td>1</td>
<td>MOV</td>
<td>0,R6</td>
</tr>
<tr>
<td>ED</td>
<td>1</td>
<td>MOV</td>
<td>0,R7</td>
</tr>
<tr>
<td>F0</td>
<td>1</td>
<td>MOVX</td>
<td>@DPTR,A</td>
</tr>
<tr>
<td>F1</td>
<td>2</td>
<td>ACALL</td>
<td>code addr</td>
</tr>
<tr>
<td>F2</td>
<td>1</td>
<td>MOVX</td>
<td>@R0,A</td>
</tr>
<tr>
<td>F3</td>
<td>1</td>
<td>MOVX</td>
<td>@R1,A</td>
</tr>
<tr>
<td>F4</td>
<td>1</td>
<td>CPL</td>
<td>A</td>
</tr>
<tr>
<td>F5</td>
<td>2</td>
<td>MOV</td>
<td>data addr,A</td>
</tr>
<tr>
<td>F6</td>
<td>1</td>
<td>MOV</td>
<td>@R1,A</td>
</tr>
<tr>
<td>F7</td>
<td>1</td>
<td>MOV</td>
<td>@R2,A</td>
</tr>
<tr>
<td>F8</td>
<td>1</td>
<td>MOV</td>
<td>@R3,A</td>
</tr>
<tr>
<td>F9</td>
<td>1</td>
<td>MOV</td>
<td>@R4,A</td>
</tr>
<tr>
<td>FA</td>
<td>1</td>
<td>MOV</td>
<td>@R5,A</td>
</tr>
<tr>
<td>FB</td>
<td>1</td>
<td>MOV</td>
<td>@R6,A</td>
</tr>
<tr>
<td>FD</td>
<td>1</td>
<td>MOV</td>
<td>@R7,A</td>
</tr>
<tr>
<td>FE</td>
<td>1</td>
<td>MOV</td>
<td>R0,A</td>
</tr>
<tr>
<td>FF</td>
<td>1</td>
<td>MOV</td>
<td>R1,A</td>
</tr>
</tbody>
</table>
ADD A,<src-byte>

Function: Add
Description: ADD adds the byte variable indicated to the Accumulator, leaving the result in the Accumulator. The carry and auxiliary carry flags are set respectively if there is a carry-out from bit 7 or bit 3, and cleared otherwise. When adding unsigned integers, the carry flag indicates an overflow occurred.

OV is set if there is a carry-out of bit 6 but not out of bit 7, or a carry-out of bit 7 but not bit 6, otherwise, OV is cleared. When adding signed integers, OV indicates a negative number produced as the sum of two positive operands, or a positive sum from two negative operands.

Example: The Accumulator holds $03H (11000110B)$, and register 0 holds $0AH (10101010B)$. The following instruction:

ADD A,R0

leaves $03H (01101101B)$ in the Accumulator with the AC flag cleared and both the carry flag and OV set to 1.

ADD A,R0

Bytes: 1
Cycles: 1
Encoding: 0 0 1 0 1 0 0
Operation: ADD

$A \leftarrow (A) + (R0)$

ADD A, direct

Bytes: 2
Cycles: 1
Encoding: 0 0 1 0 0 0 1 0
Operation: ADD

$A \leftarrow (A) + (direct)$

ADD A,@R0

Bytes: 1
Cycles: 1
Encoding: 0 0 1 0 0 0 1 1
Operation: ADD

$A \leftarrow (A) + (#R0)$

ADD A,#data

Bytes: 2
Cycles: 1
Encoding: 0 0 1 0 0 0 1 0
Operation: ADD

$A \leftarrow (A) + (#data)$

ADD A, A

Bytes: 1
Cycles: 1
Encoding: 0 0 1 0 1 0 0
Operation: ADD

$A \leftarrow (A) + (A)$

ADD A, R0

Bytes: 1
Cycles: 1
Encoding: 0 0 1 0 1 1 0 0
Operation: ADD

$A \leftarrow (A) + (R0)$

ADD A, direct

Bytes: 2
Cycles: 1
Encoding: 0 0 1 1 0 1 0 1
Operation: ADD

$A \leftarrow (A) + (direct)$

ADD A, #data

Bytes: 2
Cycles: 1
Encoding: 0 0 1 1 0 1 0 0
Operation: ADD

$A \leftarrow (A) + (#data)$

ADD A, R0

Bytes: 1
Cycles: 1
Encoding: 0 0 1 1 0 1 1 0
Operation: ADD

$A \leftarrow (A) + (R0)$

ADD A, direct

Bytes: 2
Cycles: 1
Encoding: 0 0 1 1 0 1 0 0
Operation: ADD

$A \leftarrow (A) + (direct)$

ADD A, #data

Bytes: 2
Cycles: 1
Encoding: 0 0 1 1 0 1 0 0
Operation: ADD

$A \leftarrow (A) + (#data)$
AJMP  addr11

Function: Absolute Jump

Description: AJMP transfers program execution to the indicated address, which is formed at run-time by concatenating the high-order five bits of the PC (after incrementing the PC twos), opepose bits 7 through 5, and the second byte of the instruction. The destination must therefore be within the same 2 K block of program memory as the first byte of the instruction following AJMP.

Example: The label JMPADR is at program memory location 0123H. The following instruction

AJMP

JMPADR

is at location 0345H and loads the PC with 0123H.

Bytes: 2
Cycles: 2
Encoding: a10 a11 a8 0 0 0 0 1

Operation: AJMP

(PS) = (PC) + 2
(PC) = aj + page address

ANL  <dest-byte>,<src-byte>

Function: Logical-AND for byte variables

Description: ANL performs the bitwise logical-AND operation between the variables indicated and stores the results in the destination variable. No flags are affected.

The two operands allow for addressing mode combinations. When the destination is the Accumulator, the source can use register, direct, register-indirect, or immediate addressing, when the destination is a direct address, the source can be the Accumulator or immediate data.

Note: When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, not the input pins.

Example: If the Accumulator holds 0C3H (11000111B), and register R0 holds 52H (01010010B), then the following instruction

ANL A, R0

leaves 41H (01000011B) in the Accumulator.

When the destination is a directly addressed byte, this instruction clears combinations of bits in any RAM location or hardware register. The mask byte determining the pattern of bits to be cleared would either be a constant contained in the instruction or a value computed in the Accumulator at run-time. The following instruction:

ANL P1,00111001B

clears bits 7, 3, and 2 of output port 1.

ANL  A,Rn

Bytes: 1
Cycles: 1
Encoding: 0 0 1 0 0 1 0 1

Operation: ANL

(A) = (A) \& (Rn)
ANL C, <src-byte>

Function: Logical AND for bit variables
Description: If the Boolean value of the source bit is a logical 0, then ANL C clears the carry flag; otherwise, this instruction leaves the carry flag in its current state. A zero (0) preceding the operand in the assembly language indicates that the logical complement of the addressed bit is used as the source value, but the source bit itself is not affected. No other flags are affected.
Example: Set the carry flag, and only if P1.0 = 1, ACC.7 = 1, and OV = 0:
MOV C, P1.0 ; LOAD CARRY WITH INPUT PIN STATE
ANL C, ACC.7 ; AND CARRY WITH ACCUM BIT 7
ANL C, OV ; AND WITH INVERSE OF OVERFLOW FLAG

ANL C, bit
Bytes: 2
Cycles: 2
Encoding: 1 0 0 0 0 0 0 1 0
Operation: ANL (C) ← (C) ∧ (bit)

ANL C, bit
Bytes: 2
Cycles: 2
Encoding: 1 0 1 1 0 0 0 0
Operation: ANL (C) ← (C) ∧ (bit)

Instruction Set

CJNE <dest-byte>, <src-byte>, rel

Function: Compare and Jump if Not Equal
Description: CJNE compares the magnitudes of the first two operands and branches if their values are not equal. The branch destination is computed by adding the signed relative displacement in the last instruction byte to the PC, after incrementing the PC to the start of the next instruction. The carry flag is set if the unsigned integer value of the low byte is less than the unsigned integer value of the high byte. Otherwise, the carry flag is cleared. Neither operand is affected.
Example: The Accumulator contains 34H. Register 7 contains 40H. The first instruction in the sequence:
CJNE R7, #00H, NOT_EQ
sets the carry flag and branches to the instruction at label NOT_EQ. By testing the carry flag, this instruction determines whether R7 is greater or less than 0FH.
If the data being presented to Port 1 is also 34H, then the following instruction,
WAT: CJNE A, P1, WAIT
clears the carry flag and continues with the next instruction in sequence, since the Accumulator does not equal the data read from P1 (if some other value was being input on P1, the program loops at this point until the P1 data changes to 34H.)

CJNE A, direct, rel
Bytes: 3
Cycles: 2
Encoding: 1 0 0 1 1 0 1 0 1
Operation: (PC) ← (PC) + 3
IF (direct) THEN
(PC) ← (PC) + relative offset
ELSE IF (A) < (direct) THEN
(C) ← 1
ELSE
(C) ← 0

Instruction Set
Instruction Set

CJNE A, #data, rel
Bytes: 3
Cycles: 2
Encoding: 1 0 1 1 0 1 0 0
Operation:

(PC) ← (PC) + 3
F (A) < > data
THEN
(PC) ← (PC) + relative offset
F (A) < data
THEN
(C) ← 1
ELSE
(C) ← 0

CJNE Rn, #data, rel
Bytes: 3
Cycles: 2
Encoding: 1 0 1 1 r r r
Operation:

(PC) ← (PC) + 3
F (Rn) < > data
THEN
(PC) ← (PC) + relative offset
F (Rn) < data
THEN
(C) ← 1
ELSE
(C) ← 0

CJNE @Ri, data, rel
Bytes: 3
Cycles: 2
Encoding: 1 0 1 1 0 1 1 i
Operation:

(PC) ← (PC) + 3
F (Ri) < > data
THEN
(PC) ← (PC) + relative offset
F (Ri) < data
THEN
(C) ← 1
ELSE
(C) ← 0

CLR A
Function: Clear Accumulator
Description: CLR A clears the Accumulator (all bits set to 0). No flags are affected.
Example:
The Accumulator contains $28H (01001000B). The following instruction CLR A leaves the Accumulator set to $00H (00000000B).
Bytes: 1
Cycles: 1
Encoding: 1 1 1 0 0 1 0 0
Operation: CLR
(A) ← 0

CLR bit
Function: Clear bit
Description: CLR bit clears the indicated bit (next to 0). No other flags are affected. CLR can operate on the carry flag or any directly addressable bit.
Example:
Port 1 has previously been written with $28H (01010010B). The following instruction CLR P1.2 leaves the port set to $09H (00100101B).
Bytes: 1
Cycles: 1
Encoding: 1 0 0 0 1 1
Operation: CLR
(C) ← 0

CLR C

CLR bit
Bytes: 2
Cycles: 1
Encoding: 1 1 0 0 0 0 1 0
Operation: CLR
(BC) ← 0

CLR bit
Bytes: 2
Cycles: 1
Encoding: 1 1 0 0 0 0 1 0
Operation: CLR
(B) ← 0
**Instruction Set**

### CPL

**Function:** Complement Accumulator

**Description:** CPLA logically complements each bit of the Accumulator (one’s complement). Bits which previously contained a 1 are changed to 0 and vice-versa. No flags are affected.

**Example:** The Accumulator contains 5CH (01011100B). The following instruction, 

\[
\text{CPL A}
\]

leaves the Accumulator set to 53H (10100011B).

**Bytes:** 1

**Cycles:** 1

**Encoding:** 1 1 1 1 0 0 1 0

**Operation:** CPL (A) ← ¬(A)

### CPL bit

**Description:** CPL bit complements the bit variable specified. A bit that had been a 1 is changed to 0 and vice-versa. No other flags are affected. CLR can operate on the carry or any directly addressable bit.

**Example:** Port 1 has previously been written with 08H (01001000B). The following instruction sequence, CPL P1.1, CPL P1.2 leaves the port set to 08H (01001000B).

**Bytes:** 1

**Cycles:** 1

**Encoding:** 1 0 1 1 0 1 0 0

**Operation:** CPL (C) ← ¬(C)

### DA

**Function:** Decimal adjust Accumulator for Addition

**Description:** DA A adjusts the eight-bit value in the Accumulator resulting from the earlier addition of two variables (each in packed BCD format), producing two four-bit digits. Any ADD or ADC instruction may have been used to perform the addition.

- If Accumulator bits 3 through 9 are greater than nine (000000100111), or if the AC flag is one, its is added to the Accumulator producing the proper BCD digit in the low-order nibble. This internal addition sets the carry flag if a carry-out of the low-order four-bit field propagates through all high-order bits, but it does not clear the carry flag otherwise.

- If the carry flag is new set, or if the four high-order bits are not binary encoded BCD digits in the high-order nibble. Again, this sets the carry flag if there is a carry-out of the high-order bits, but does not clear the carry. The carry flag thus indicates if the sum of the original two BCD variables is greater than 100, allowing multiple precision decimal addition. OV is not affected.

All of this occurs during the one instruction cycle. Essentially, this instruction performs the decimal conversion by adding 0AH, 0BH, 0CH, or D0H to the Accumulator, depending on initial Accumulator and PSW conditions.

Note: DA A cannot simply convert a hexadecimal number in the Accumulator to BCD notation, nor does DAA apply to decimal subtraction.

**Example:** The Accumulator holds the value 56H (01010110B), representing the packed BCD digits of the decimal number 56. Register 3 contains the value 34H (00110100B), representing the packed BCD digits of the decimal number 34. The carry flag is set. The following instruction sequence follows:

\[
\text{ADDC A, R3}
\]

\[
\text{DA A}
\]

First perform a standard two's-complement binary addition, resulting in the value 08EH (10101001B) in the Accumulator. This carry and auxiliary carry flags are cleared.

The Decimal Adjust instruction then allows the Accumulator to its value 24H (00010100B), indicating the packed BCD digits of the decimal number 24. The accumulator low-order digits of the decimal sum are 24, and the carry-out. The carry is set by the Decimal Adjust instruction, indicating that a decimal overflow occurred. The true sum of 56, 52, and 1 is 124.

BCD variables can be incremented or decremented by adding 01H or 09H. If the Accumulator initially holds 30H (representing the digits of 30 decimal), then the following instruction sequence is used:

\[
\text{ADD A, #09H}
\]

\[
\text{DA A}
\]

This leaves the carry-set and 2FH in the Accumulator, since 30 + 9 = 129. The low-order byte of the sum can be interpreted to mean 30 - 1 = 29.

**Bytes:** 1

**Cycles:** 1

**Encoding:** 1 1 1 1 0 1 0 1 0

**Operation:** DA (30H) ← ¬(bit)
**DEC**

**Function:** Decrement

**Description:** DEC byte decrements the variable indicated by R. An original value of 00H underflows to 0FFH. No flags are affected. Four operand addressing modes are allowed: accumulator, register, direct, or register indirect.

**Example:**
- Register 0 contains FFH (01111111B), Internal RAM locations 7EH and 7FH contain 00H and 40H, respectively.
- The following instruction sequence:
  - DEC @R0
  - DEC R0
  - DEC @R0
- Leaves register 0 set to 7EH and internal RAM locations 7EH and 7FH set to 0FH and 3FH.

**ENCODING**

<table>
<thead>
<tr>
<th>Type</th>
<th>Encoding</th>
<th>Cycles</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEC</td>
<td>0000011</td>
<td>1</td>
<td>DEC (A) ← (A) - 1</td>
</tr>
<tr>
<td>DEC R0</td>
<td>000011</td>
<td>1</td>
<td>DEC (R0) ← (R0) - 1</td>
</tr>
<tr>
<td>DEC direct</td>
<td>0000101</td>
<td>1</td>
<td>DEC (direct) ← (direct) - 1</td>
</tr>
<tr>
<td>DEC @Ri</td>
<td>0000111</td>
<td>1</td>
<td>DEC (Ri) ← (Ri) - 1</td>
</tr>
</tbody>
</table>

---

**DIV**

**Function:** Divide

**Description:** DIV AB divides the unsigned eight bit integer in the Accumulator by the unsigned eight bit integer in register B. The Accumulator contains the integer part of the quotient, register B contains the integer remainder. The carry and overflow flags are cleared.

**Exception:** If A had originally contained 00H, the values returned in the Accumulator and B register are undefined and the overflow flag is set. The carry flag is cleared in any case.

**Example:**
- The Accumulator contains 251 (0FF8H or 11111110B) and B contains 18 (0010100B). The following instruction:
  - DIV AB
- Leaves 13 in the Accumulator (0D3H or 00001101B) and the value 17 (11110000B) in B, since
  
  \[ 251 \div (13 \times 18) = 13 \]

**ENCODING**

<table>
<thead>
<tr>
<th>Type</th>
<th>Encoding</th>
<th>Cycles</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>DIV</td>
<td>000000100</td>
<td>4</td>
<td>DIV (A) ← (A) / (B)</td>
</tr>
</tbody>
</table>

---

**Instruction Set**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEC</td>
<td>Decrement</td>
<td>DEC @R0</td>
</tr>
<tr>
<td>DIV</td>
<td>Divide</td>
<td>DIV AB</td>
</tr>
</tbody>
</table>
DJNZ <byte>,<rel-addr>

Function: Decrement and Jump if Not Zero
Description: DJNZ decrements the location indicated by <byte> and branches to the new address indicated by the second operand if the resulting value is not zero. An original value of 0xFF underflows to 00H. No flags are affected. The branch destination is computed by adding the signed relative displacement value in the last instruction byte to the PC, after incrementing the PC by the first byte of the following instruction.

Example: Internal RAM locations 40H, 50H, and 60H contain the values 00H, 10H, and 11H, respectively. The following instruction sequence calculates the difference of 10H and 11H:

```
DJNZ 40H LABEL1
DJNZ 50H LABEL2
DJNZ 60H LABEL3
```

causes a jump to the instruction at label LABEL2 with the values 00H, 0FH, and 10H in the three RAM locations. The first jump was not taken because the result was zero.

This instruction provides a simple way to execute a program loop a given number of times or to adjust a moderate time delay (from 250 to 510 machine cycles) with a single instruction. The following instruction sequence:

```
MOV R2, #8
TOGGLE: CPL R1
DJNZ R2,TOGGLE
```

jumps R1 8 times, causing four output pins to appear at bit 7 of output Port 1. Each pulse lasts three machine cycles, but for DJNZ and one to alter the pin.

DJNZ Rrel, rel

Function: Decrement and Jump if Not Zero
Description: DJNZ decrements the location indicated by Rrel (a register offset) and branches to the new address indicated by the second operand if the resulting value is not zero. An original value of 0xFF underflows to 00H. No flags are affected. The branch destination is computed by adding the signed relative displacement value in the last instruction byte to the PC, after incrementing the PC by the first byte of the following instruction.

Example: Internal RAM locations 40H, 50H, and 60H contain the values 00H, 10H, and 11H, respectively. The following instruction sequence calculates the difference of 10H and 11H:

```
DJNZ R2, LABEL1
DJNZ R3, LABEL2
DJNZ R4, LABEL3
```

causes a jump to the instruction at label LABEL2 with the values 00H, 0FH, and 10H in the three RAM locations. The first jump was not taken because the result was zero.

This instruction provides a simple way to execute a program loop a given number of times or to adjust a moderate time delay (from 250 to 510 machine cycles) with a single instruction. The following instruction sequence:

```
MOV R2, #8
TOGGLE: CPL R1
DJNZ R2,TOGGLE
```

jumps R1 8 times, causing four output pins to appear at bit 7 of output Port 1. Each pulse lasts three machine cycles, but for DJNZ and one to alter the pin.

INC <byte>

Function: Increment
Description: INC increments the indicated variable by 1. An original value of 0FFH overflows to 00H. No flags are affected.
Note: When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, not the input pins.

Example: Register 0 contains 7EH (01111110B). Internal RAM locations 7EH and 7FH contain OFFH and 40H, respectively. The following instruction sequence:

```
INC @R0
INC @R0
INC @R0
```

leaves register 0 set to 7FH and internal RAM locations 7EH and 7FH holding 00H and 41H, respectively.

INC A

Bytes: 1
Cycles: 1
Encoding: 00000000 00100
Operation: INC A ← (A) + 1

INC R8

Bytes: 1
Cycles: 1
Encoding: 00000000 01111
Operation: INC (R8) ← (R8) + 1

INC direct

Bytes: 2
Cycles: 1
Encoding: 00000000 10101
direct address
Operation: INC (direct) ← (direct) + 1

INC @Ri

Bytes: 1
Cycles: 1
Encoding: 00000000 01111
Operation: INC (@Ri) ← (@Ri) + 1
INC DPTR

**Description:**
INC DPTR increments the 16-bit data pointer by 1. A 16-bit increment (modulo 2^16) is performed, and an overflows of the low-order byte of the data pointer (DPL) from 0FH to 0RH increments the high-order byte (DPH). No flags are affected.

**Example:**
Registers DPH and DPL contain 12H and 0FCH, respectively. The following instruction sequence:

INC DPTR
INC DPTR
INC DPTR
changes DPH and DPL to 13H and 01H.

<table>
<thead>
<tr>
<th>Bytes</th>
<th>Cycles</th>
<th>Encoding</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>01000111</td>
<td>INC DPTR = (DPTR) + 1</td>
</tr>
</tbody>
</table>

**JB bit.rel**

**Function:**
Jump if bit set

**Description:**
If the indicated bit is a one, JB jump to the address indicated; otherwise, it proceeds with the next instruction. The branch destination is computed by adding the signed relative displacement in the third instruction byte to the PC, after incrementing the PC to the first byte of the next instruction. The bit tested is not modified. No flags are affected.

**Example:**
The data present at input port 1 is 11001010B. The accumulator holds 00 (01000101B). The following instruction sequence.

JB P1, LABEL1
JB ACC, LABEL2
causes program execution to branch to the instruction at label LABEL2.

<table>
<thead>
<tr>
<th>Bytes</th>
<th>Cycles</th>
<th>Encoding</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>2</td>
<td>00100000</td>
<td>JB (PC) = (PC) + 3 IF (bit) = 1 THEN (PC) = (PC) + rel</td>
</tr>
</tbody>
</table>

**JBC bit.rel**

**Function:**
Jump if bit is set and clear bit

**Description:**
If the indicated bit is one, JBC branches to the address indicated; otherwise, it proceeds with the next instruction. The bit will not be cleared if it is already a zero. The branch destination is computed by adding the signed relative displacement in the third instruction byte to the PC, after incrementing the PC to the first byte of the next instruction. No flags are affected.

**Example:**
The accumulator holds 0BH (01101011B). The following instruction sequence.

JBC ACC, LABEL1
JBC ACC, LABEL2
causes program execution to continue at the instruction identified by the label LABEL2, with the accumulator modified to 5DH (01111001B).

<table>
<thead>
<tr>
<th>Bytes</th>
<th>Cycles</th>
<th>Encoding</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>2</td>
<td>00001000</td>
<td>JBC (PC) = (PC) + 3 IF (bit) = 1 THEN (PC) = (PC) + rel</td>
</tr>
</tbody>
</table>

**JC rel**

**Function:**
Jump if carry is set

**Description:**
If the carry flag is set, JC branches to the address indicated; otherwise, it proceeds with the next instruction. The branch destination is computed by adding the signed relative displacement in the second instruction byte to the PC, after incrementing the PC twice. No flags are affected.

**Example:**
The carry flag is cleared. The following instruction sequence.

JC LABEL1
JC LABEL2
sets the carry and causes program execution to continue at the instruction identified by the label LABEL2.

<table>
<thead>
<tr>
<th>Bytes</th>
<th>Cycles</th>
<th>Encoding</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2</td>
<td>01000000</td>
<td>JC (PC) = (PC) + 2 IF (C) = 1 THEN (PC) = (PC) + rel</td>
</tr>
</tbody>
</table>
JMP @A+DPTR

Function: Jump indirect

Description: JMP @A+DPTR adds the eight-bit unsigned contents of the Accumulator with the 16-bit data pointer and loads the resulting sum to the program counter. This is the address for subsequent instruction fetches. Sixteen-bit addition is performed (modulo 2^16), a carry-out from the low-order eight bits propagates through the higher-order bits. Neither the Accumulator nor the Data Pointer is altered. No flags are affected.

Example: An even number from 0 to 8 is in the Accumulator. The following sequence of instructions branches to one of four JMP instructions in a jump table starting at JMP_TBL.

```
MOVC DPTR, A
JMP @A+DPTR
JMP_TBL: AJMP LABEL0
AJMP LABEL1
AJMP LABEL2
AJMP LABEL3
```

If the Accumulator equals 00H when starting this sequence, execution jumps to label LABEL2. Because AJMP is a 2-byte instruction, the jump instructions start at every other address.

Bytes: 1
Cycles: 2
Encoding: 0 1 1 1 0 0 0 1 1
Operation: 

```
JMP
(PC) = (A) + (DPTR)
```

JNB bit,rel

Function: Jump if Bit Not set

Description: If the indicated bit is a 0, JNB branches to the indicated address, otherwise, it proceeds with the next instruction. The branch destination is computed by adding the signed relative displacement to the third instruction byte to the PC, after incrementing the PC by the first byte of the next instruction. The bit tested is not modified. No flags are affected.

Example: The data present at input port 1 is 11001010B. The Accumulator holds 01H (01010101B). The following instruction sequence.

```
JNB P1.3, LABEL1
JNB ACC.3, LABEL2
```

causes program execution to continue at the instruction identified by the label LABEL2.

Bytes: 3
Cycles: 2
Encoding: 0 0 1 1 0 0 0 0 0
Operation: 

```
JNB
(PC) = (PC) + 3
IF (bit) = 0
THEN (PC) = (PC) + rel
```

JNC rel

Function: Jump if Carry not set

Description: If the carry flag is a 0, JNC branches to the address indicated, otherwise, it proceeds with the next instruction. The branch destination is computed by adding the signed relative displacement to the second instruction byte to the PC, after incrementing the PC twice to point to the next instruction. The carry flag is not modified.

Example: The carry flag is set. The following instruction sequence.

```
JNC LABEL1
CPL 0
JNC LABEL2
```

causes the carry and causes program execution to continue at the instruction identified by the label LABEL2.

Bytes: 2
Cycles: 2
Encoding: 0 1 0 1 0 0 0 0 0
Operation: 

```
JNC
(PC) = (PC) + 2
IF (D) = 0
THEN (PC) = (PC) + rel
```
### JNZ rel

**Function:** Jump if Accumulator Not Zero

**Description:** If any bit of the Accumulator is a zero, JNZ branches to the indicated address; otherwise, it proceeds with the next instruction. The branch destination is computed by adding the signed relative displacement in the second instruction byte to the PC, after incrementing the PC twice. The Accumulator is not modified. No flags are affected.

**Example:** The Accumulator originally holds 00H. The following instruction sequence,

```
JNZ LABEL1
INC A
JNZ LABEL2
```

sets the Accumulator to 01H and continues at label LABEL2.

<table>
<thead>
<tr>
<th>Bytes</th>
<th>Cycles</th>
<th>Encoding</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2</td>
<td>0 1 1 1 0 0 0 0</td>
<td>rel address</td>
</tr>
</tbody>
</table>

### JZ rel

**Function:** Jump if Accumulator Zero

**Description:** If all bits of the Accumulator are 0, JZ branches to the address indicated; otherwise, it proceeds with the next instruction. The branch destination is computed by adding the signed relative displacement in the second instruction byte to the PC, after incrementing the PC twice. The Accumulator is not modified. No flags are affected.

**Example:** The Accumulator originally contains 00H. The following instruction sequence,

```
JZ LABEL1
DEC A
JZ LABEL2
```

changes the Accumulator to 03H and causes program execution to continue at the instruction identified by the label LABEL2.

<table>
<thead>
<tr>
<th>Bytes</th>
<th>Cycles</th>
<th>Encoding</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2</td>
<td>0 1 1 0 0 0 0 0</td>
<td>rel address</td>
</tr>
</tbody>
</table>

### Instruction Set

#### LCALL addr16

**Function:** Long call

**Description:** LCALL calls a subroutine located at the indicated address. The instruction adds three to the program counter to generate the address of the next instruction and then pushes the 16-bit result onto the stack (low byte first), incrementing the Stack Pointer by two. The high-order and low-order bytes of the PC are then loaded, respectively, into the second and third bytes of the LCALL instruction. Program execution continues with the instruction at this address. The subroutine may therefore begin anywhere in the full 64K byte program memory address space. No flags are affected.

**Example:** Initially the Stack Pointer equals 07H. The label SUBRIN is assigned to program memory location 1234H. After executing the instruction

```
LCALL SUBRIN
```

at location 0123H, the Stack Pointer will contain 09H, internal RAM locations 09H and 08H will contain 26H and 01H, and the PC will contain 1234H.

<table>
<thead>
<tr>
<th>Bytes</th>
<th>Cycles</th>
<th>Encoding</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>2</td>
<td>0 0 0 1 0 0 1 0</td>
<td>addr15 addr8</td>
</tr>
</tbody>
</table>

#### LIMP addr16

**Function:** Long Jump

**Description:** LIMP causes an unconditional branch to the indicated address, by loading the high-order and low-order bytes of the PC (respectively) with the second and third instruction bytes. The branch destination may therefore lie anywhere in the full 64K program memory address space. No flags are affected.

**Example:** The label JUMP1 is assigned to program memory location 1234H. The instruction

```
LIMP JUMP1
```

at location 0123H will load the program counter with 1234H.

<table>
<thead>
<tr>
<th>Bytes</th>
<th>Cycles</th>
<th>Encoding</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>2</td>
<td>0 0 0 0 0 0 0 0</td>
<td>addr15 addr8</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bytes</th>
<th>Cycles</th>
<th>Encoding</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2</td>
<td>0 1 1 1 0 0 0 0</td>
<td>rel address</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bytes</th>
<th>Cycles</th>
<th>Encoding</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2</td>
<td>0 1 1 1 0 0 0 0</td>
<td>rel address</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bytes</th>
<th>Cycles</th>
<th>Encoding</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2</td>
<td>0 1 1 1 0 0 0 0</td>
<td>rel address</td>
</tr>
</tbody>
</table>
**Instruction Set**

**MOV dest-byte, src-byte**

**Function:** Move byte variable

**Description:** The byte variable indicated by the second operand is copied into the location specified by the first operand. The source byte is not affected. No other register or flag is affected.

**Example:** Internal RAM location 30H holds 40H. The value of RAM location 40H is 10H. The data present at input port 1 is 11001010B (OCAH)

- MOV R0,30H  ; R0 = 30H
- MOV A,R0   ; A = 30H
- MOV R1A    ; R1A = 30H
- MOV B,R1   ; B = 10H
- MOV @R1 P1 ; RAM (40H) = 0CAH
- MOV P2 P1  ; P2 = 0CAH

Leaves the value 30H in register 0, 40H in both the Accumulator and register 1, 10H in register B, and 0CAH (11001010B) both in RAM location 40H and output on port 2.

**MOV A,Rx**

- **Bytes:** 1
- **Cycles:** 1
- **Encoding:** 111011 r r r
- **Operation:** MOV (A) ← (Ry)

**MOV A,direct**

- **Bytes:** 2
- **Cycles:** 1
- **Encoding:** 111001 01 1
- **Operation:** MOV (A) ← (direct)

*MOV A,ACC is not a valid Instruction.

**MOV A,@Rx**

- **Bytes:** 1
- **Cycles:** 1
- **Encoding:** 111011 0111
- **Operation:** MOV (A) ← (Ry)

**Instruction Set**

**MOV A,#data**

- **Bytes:** 2
- **Cycles:** 1
- **Encoding:** 01110101 0 r r
- **Operation:** MOV (A) ← #data

**MOV R0, A**

- **Bytes:** 1
- **Cycles:** 1
- **Encoding:** 111111 r r r
- **Operation:** MOV (R0) ← (A)

**MOV R0,direct**

- **Bytes:** 2
- **Cycles:** 1
- **Encoding:** 101011 r r r
- **Operation:** MOV (R0) ← (direct)

**MOV R0,#data**

- **Bytes:** 1
- **Cycles:** 1
- **Encoding:** 011111 1 r r r
- **Operation:** MOV (R0) ← #data

**MOV direct,A**

- **Bytes:** 2
- **Cycles:** 1
- **Encoding:** 111101 01 1
- **Operation:** MOV (direct) ← (A)

**MOV direct,R0**

- **Bytes:** 2
- **Cycles:** 2
- **Encoding:** 100011 r r r
- **Operation:** MOV (direct) ← (Rr)
MOV direct, direct
Bytes: 3
Cycles: 1
Encoding: 
Operation: MOV (direct) → (direct)

MOV direct, @Ri
Bytes: 2
Cycles: 2
Encoding: 
Operation: MOV (direct) → @data

MOV direct, #data
Bytes: 3
Cycles: 2
Encoding: 
Operation: MOV (direct) → #data

MOV @Ri, A
Bytes: 1
Cycles: 1
Encoding: 
Operation: MOV (@Ri) → A

MOV @Ri, direct
Bytes: 2
Cycles: 2
Encoding: 
Operation: MOV (@Ri) → (direct)

MOV @Ri, #data
Bytes: 2
Cycles: 1
Encoding: 
Operation: MOV (@Ri) → #data

Instruction Set

MOV <dest-bit>, <src-bit>
Function: Move bit data
Description: MOV <dest-bit>, <src-bit> copies the 8-bit value indicated by the second operand into the location specified by the first operand. One of the operands must be the carry flag. The other may be any directly addressable bit. No other register or flag is affected.

Example: The carry flag is originally set. The data present at input Port 3 is 11000101B. The data previously written to output Port 1 is 00100101B.

MOV P3.3
MOV C, P3.3
MOV P1.2, C

Leaves the carry cleared and changes Port 1 to 30H (00110100B).

MOV C, bit
Bytes: 2
Cycles: 1
Encoding: 
Operation: MOV (C) → (bit)

MOV bit, C
Bytes: 2
Cycles: 2
Encoding: 
Operation: MOV (bit) → (C)

MOV DPTR, #data16
Function: Load Data Pointer with a 16-bit constant
Description: MOV DPTR, #data16 loads the Data Pointer with the 16-bit constant indicated. The 16-bit constant is loaded into the second and third bytes of the instruction. The second byte (DPH) is the high-order byte, while the third byte (DPL) holds the lower-order byte. No flags are affected.

Example: The instruction, MOV DPTR, #1234H loads the value 1234H into the Data Pointer. DPH holds 12H, and DPL holds 34H.

Bytes: 3
Cycles: 2
Encoding: 
Operation: MOV (DPTR) → #data16

MOV (DPTR) → #data16
DPL → DPL → #data, R0
**MOVC A,@A+<base-reg>**

**Function:** Move Code byte

**Description:** The MOVC instructions load the Accumulator with a code byte or constant from program memory. The address of the byte fetched is the sum of the original unsigned 8-bit Accumulator contents and the contents of a 16-bit based register, which may be either the Data Pointer or the PC. In the latter case, the PC is incremented to the address of the following instruction before being added with the Accumulator; otherwise the base register is not altered. Sixteen-bit addition is performed so a carry-out from the low-order eight bits may propagate through higher-order bits. No flags are affected.

**Example:** A value between 0 and 3 is in the Accumulator. The following instructions will translate the value in the Accumulator to one of four values defined by the DB (define byte) directive.

```
REL_PC
INC A
MOVC A,@A+PC
RET
DB 00H
DB 7FH
DB 86H
DB 99H
```

If the subroutine is called with the Accumulator equal to 0FH, it returns with 7FH in the Accumulator. The INC A before the MOVC instruction is needed to “get around” the RET instruction above the table. If several bytes of code separate the MOVC from the table, the corresponding number is added to the Accumulator instead.

**MOVC A,@A+DPTR**

**Bytes:** 1

**Cycles:** 2

**Encoding:** 1 0 0 1 0 0 1 1

**Operation:** MOVC (A) ← (A) + (DPTR)

**MOVC A,@A+PC**

**Bytes:** 1

**Cycles:** 2

**Encoding:** 1 0 0 0 0 0 0 1 1

**Operation:** MOVC (PC) ← (PC) + 1
(A) ← (A) + (PC)

---

**Instruction Set**

**MOVX <dest-byte>,<src-byte>**

**Function:** Move External

**Description:** The MOVX instructions transfer data between the Accumulator and a byte of external data memory, which is why “X” is extended to MOVX. There are two types of instructions, differing in whether they provide an 8-bit or 16-bit indirect address to the external data RAM.

- If the first type, the contents of (R0) or (R1) in the current address register point to an 8-bit address multiplexed with data on P0. Eight bits are sufficient for external I/O expansion decoding or for a relatively small RAM array. For somewhat larger arrays, any output port pins can be used to output higher-order address bits. These pins are controlled by an output instruction preceding the MOVX.
- In the second type of MOVX instruction, the Data Pointer generates a 16-bit address, P2 outputs the high-order eight address bits (the contents of DP0), while P0 multiplexes the low-order eight bits (SPLV) with data. The P2 Special Function Register retains its previous contents, while the P0 output buffers emit the contents of DP1. This form of MOVX is faster and more efficient for accessing very large data arrays (up to 64K bytes), since no additional instructions are needed to set up the output ports.

- It is possible to use both MOVX types in some situations. A large RAM array with its high order address lines driven by P2 can be addressed via the Data Pointer, or with code to output high-order address bits to P2, followed by a MOVX instruction using R0 or R1.

**Example:** An external 256-byte RAM using multiplexed address lines is connected to the 80H Port 0. Port 3 provides control lines for the external RAM. Ports 1 and 2 also used for normal I/O. Register 0 and 1 contain 12H and 34H. Location 24H of the external RAM holds the value 56H. The instruction sequence.

```
MOVX A,R1
MOVX R0,A
```

**MOVC A,@R1**

**Bytes:** 1

**Cycles:** 2

**Encoding:** 1 1 1 0 0 0 0 1 1

**Operation:** MOVC (A) ← (R1)

**MOVC A,@DPTR**

**Bytes:** 1

**Cycles:** 2

**Encoding:** 1 1 1 0 0 0 0 0 0

**Operation:** MOVC (A) ← (DPTR)
Instruction Set

MOVX @Ri,A

Function: Multiply
Description: MOVX @Ri,A multiplies the unsigned 8-bit integers in the Accumulator and register Ri. The low-order byte of the 16-bit result is left in the Accumulator, and the high-order byte is in B. If the product is greater than 255 (FFFFH), the overflow flag is set; otherwise it is cleared. The carry flag is always cleared.

Example: Originally the Accumulator holds the value 80 (50H). Register B holds the value 16 (10H). The instruction, MOVX @Ri,A, will give the product 1280 (320H), so B is changed to 32H (00100000B) and the Accumulator is cleared. The overflow flag is set, carry is cleared.

Bytes: 3
Cycles: 4
Encoding: 1 0 1 0 0 1 0 0
Operation: MOVX (DPTR) ← (A)

NOP

Function: No Operation
Description: Execution continues at the following instruction. Other than the PC, no registers or flags are affected.

Example: A low-going output pulse on Bit 7 of Port 2 must last exactly 5 cycles. A simple SETB CLR sequence generates a one-cycle pulse, so four additional cycles must be inserted. This may be done (assuming no interrupts are enabled) with the following instruction sequence.

Bytes: 1
Cycles: 1
Encoding: 0 0 0 0 0 0 0 0
Operation: NOP

ORL <dest-byte> <src-byte>

Function: Logical OR for byte variables
Description: ORL performs the bitwise logical-OR operation between the indicated variables, storing the results in the destination byte. No flags are affected.

Example: If the Accumulator holds 03H (00000011B) and R0 holds 2AH (00101010B) then the following instruction, ORL R0, leaves the Accumulator holding the value 05H (00000101B). When the destination is a directly addressed byte, the instruction can set combinations of bits in any RAM location or hardware register. The pattern of bits to be set is determined by a mask byte, which may be either a constant data value in the instruction or a variable computed in the Accumulator at run-time. The instruction, ORL P1.0, P1.0=1001010B leaves bits 0, 4, and 1 of output Port 1.

Bytes: 1
Cycles: 1
Encoding: 0 1 0 0 1 1 1 1
Operation: ORL (A) ← (A) ∨ (Ri)
### ORL

**A.direct**
- **Bytes:** 2
- **Cycles:** 1
- **Encoding:** 0 1 0 0 0 1 0 1
  - **Operation:** ORL (A) ← (A) V [direct]

**A.@R**
- **Bytes:** 2
- **Cycles:** 1
- **Encoding:** 0 1 0 0 0 1 1 1
  - **Operation:** ORL (A) ← (A) V [Ri]

**A.#data**
- **Bytes:** 2
- **Cycles:** 1
- **Encoding:** 0 1 0 0 0 1 0 0
  - **Operation:** ORL (A) ← (A) V [#data]

**direct,#A**
- **Bytes:** 2
- **Cycles:** 1
- **Encoding:** 0 1 0 0 0 1 0 0
  - **Operation:** ORL (direct) ← (direct) V (A)

**direct,#data**
- **Bytes:** 3
- **Cycles:** 2
- **Encoding:** 0 1 0 0 0 1 0 1
  - **Operation:** ORL (direct) ← (direct) V [#data]

### Instruction Set

#### ORL C.<src-bit>

**Function:** Logical OR for bit variables

**Description:** Set the carry flag if the Boolean value is a logical 1, leave the carry in its current state otherwise. A slash (/) preceding the operand in the assembly language indicates that the logical complement of the addressed bit is used as the source value, but the source bit itself is not affected. No other flags are affected.

**Example:**
- Set the carry flag if only if P1.1 = 1, ACC.7 = 1, or OR = 0
  - MOV C,P1.1 /LOAD CARRY WITH INPUT PIN P10
  - ORL C,ACC.7 /OR CARRY WITH THE ACC. BIT 7
  - ORL C,OV /OR CARRY WITH THE INVERSE OF OV

**ORL C,bit**
- **Bytes:** 2
- **Cycles:** 2
- **Encoding:** 0 1 1 1 0 0 1 0
  - **Operation:** ORL (C) ← (C) V (bit)

**ORL C/bit**
- **Bytes:** 2
- **Cycles:** 2
- **Encoding:** 0 1 1 0 0 0 0 0
  - **Operation:** ORL (C) ← (C) V (bit)

### POP direct

**Function:** Pop from stack.

**Description:** The contents of the internal RAM location addressed by the Stack Pointer is read, and the Stack Pointer is decremented by one. The value read is then transferred to the directly addressed byte indicated. No flags are affected.

**Example:** The Stack Pointer originally contains the value 32H, and internal RAM location 30H through 32H contain the values 20H, 21H, and 31H, respectively. The following instruction sequence,

- POP D0H
- POP #D1H
- POP (SP) leaves the Stack Pointer equal to the value 30H and sets the Data Pointer to 0125H. At this point, the following instruction,

- POP (SP) leaves the Stack Pointer set to 20H. In this special case, the Stack Pointer was decremented to 20H before being loaded with the value popped (30H).

**Bytes:** 2
- **Cycles:** 2
- **Encoding:** 1 1 0 0 0 0 0 0
  - **Operation:** POP (direct) ← (#SP) / (SP) ← (SP) - 1
**PUSH**

Function: Push onto stack

Description: The Stack Pointer is incremented by one. The contents of the indicated variable is then copied into the internal RAM location addressed by the Stack Pointer. Otherwise, no flags are affected.

Example: On entering an interrupt routine, the Stack Pointer contains 08H. The Data Pointer holds the value 0123H. The following instruction sequence:

```
PUSH DX
PUSH DH
```

leaves the Stack Pointer set to 09H and stores 23H and 01H in internal RAM locations 0A6H and 0B6H, respectively.

Bytes: 2
Cycles: 2

Encoding: 

<table>
<thead>
<tr>
<th>1</th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>direct address</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**RET**

Function: Return from subroutine

Description: RET pops the high- and low-order bytes of the PC successively from the stack, decrementing the Stack Pointer by two. Program execution continues at the resulting address, generally the instruction immediately following an ACALL or LCALL. No flags are affected.

Example: The Stack Pointer originally contains the value 08H. Internal RAM locations 0A6H and 0B6H contain the values 23H and 01H, respectively. The following instruction:

```
RET
```

leaves the Stack Pointer equal to 08H. Program execution continues at location 0123H.

Bytes: 1
Cycles: 2

Encoding: 

| 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 |

**RL A**

Function: Rotate Accumulator Left

Description: The contents in the Accumulator are rotated one bit to the left. Bit 7 is rotated into the bit 0 position. No flags are affected.

Example: The Accumulator holds the value 0C8H (11000010B). The following instruction:

```
RL A
```

leaves the Accumulator holding the value 68H (10001010B) with the carry unaffected.

Bytes: 1
Cycles: 1

Encoding: 

| 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 |

**RETI**

Function: Return from interrupt

Description: RETI pops the high- and low-order bytes of the PC successively from the stack and restores the interrupt logic to accept additional interrupts at the same priority level as the one just processed. The Stack Pointer is left decremented by two. No other registers are affected; the PSW is not automatically restored to its pre-interrupt status. Program execution continues at the resulting address, which is generally the instruction immediately after the point at which the interrupt request was detected. If a lower or same level interrupt was pending when the RETI instruction is executed, that one instruction is executed before the pending interrupt is processed.

Example: The Stack Pointer originally contains the value 08H. An interrupt was detected during the instruction ending at location 0123H. Internal RAM locations 0A6H and 0B6H contain the values 23H and 01H, respectively. The following instruction:

```
RETI
```

leaves the Stack Pointer equal to 08H and returns program execution to location 0123H.

Bytes: 1
Cycles: 2

Encoding: 

| 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 |

Operation:

<table>
<thead>
<tr>
<th>(SP)</th>
<th>(SP) - 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>(SP)</td>
<td>(SP) - 1</td>
</tr>
<tr>
<td>(SP)</td>
<td>(SP) - 1</td>
</tr>
</tbody>
</table>

Operation:

<table>
<thead>
<tr>
<th>(PC)</th>
<th>(IP)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(PC)</td>
<td>(IP)</td>
</tr>
<tr>
<td>(SP)</td>
<td>(SP) - 1</td>
</tr>
</tbody>
</table>

Operation:

<table>
<thead>
<tr>
<th>A</th>
<th>A</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A = (A)</th>
<th>C = (C)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A</th>
<th>A</th>
<th>=</th>
<th>A</th>
<th>A</th>
<th></th>
<th>A</th>
<th>A</th>
</tr>
</thead>
</table>
Instruction Set

RLC A
Function: Rotate Accumulator Left through the Carry flag
Description: The eight bits in the Accumulator and the carry flag are together rotated one bit to the left. Bit 7 moves into the carry flag, the original value of the carry flag moves into the bit 0 position. No other flags are affected.
Example: The Accumulator holds the value $0C9H$(11001010B), and the carry is zero. The following instruction,
RLC A
leaves the Accumulator holding the value $89H$(10010101B) with the carry set.
Bytes: 1
Cycles: 1
Encoding: 0 0 1 1 6 0 1 1
Operation:

RR A
Function: Rotate Accumulator Right
Description: The eight bits in the Accumulator are rotated one bit to the right. Bit 0 is rotated into the bit 7 position. No flags are affected.
Example: The Accumulator holds the value $0C9H$(11001010B). The following instruction,
RR A
leaves the Accumulator holding the value $0E2H$(11100101B) with the carry unaffected.
Bytes: 1
Cycles: 1
Encoding: 0 0 0 0 6 0 1 1
Operation:

RRC A
Function: Rotate Accumulator Right through Carry flag
Description: The eight bits in the Accumulator and the carry flag are together rotated one bit to the right. Bit 0 moves into the carry flag, the original value of the carry flag moves into the bit 7 position. No other flags are affected.
Example: The Accumulator holds the value $0C9H$(11001010B), the carry is zero. The following instruction,
RRC A
leaves the Accumulator holding the value $93H$(01100011B) with the carry set.
Bytes: 1
Cycles: 1
Encoding: 0 0 0 1 6 0 1 1
Operation:

SETB <bit>
Function: Set Bit
Description: SETB sets the indicated bit to one. SETB can operate on the carry flag or any directly addressable bit. No other flags are affected.
Example: The carry flag is cleared. Output Port 1 has been written with the value 34H $(00110100B)$. The following instructions,
SETB C
SETB P1.0
sets the carry flag to 1 and changes the data output on Port 1 to 35H $(00110101B)$.
Bytes: 1
Cycles: 1
Encoding: 1 1 0 1 0 0 1 1
Operation:

SETB bit
Function: Set Bit
Description: SETB sets the indicated bit to one. SETB can operate on the carry flag or any directly addressable bit. No other flags are affected.
Example: The carry flag is cleared. Output Port 1 has been written with the value 34H $(00110100B)$. The following instructions,
SETB C
SETB P1.0
sets the carry flag to 1 and changes the data output on Port 1 to 35H $(00110101B)$.
Bytes: 1
Cycles: 1
Encoding: 1 1 0 1 0 0 1 1
Operation:

SJMP rel
Function: Short Jump
Description: Program control branches unconditionally to the address indicated. The branch destination is computed by adding the signed displacement in the second instruction byte to the PC, after incrementing the PC two times. Therefore, the range of destinations allowed is from 128 bytes preceding this instruction 127 bytes following it.
Example: The label RELADR is assigned to an instruction at program memory location 0123H. The following instruction,
SJMP RELADR
assembles into location 0125H. After the instruction is executed, the PC contains the value 0123H. Note: Under the above conditions the instruction following SJMP is at 1024H. Therefore, the displacement byte of the instruction is the relative offset $(0123H - 0102H) = 216H$. Put another way, an SJMP with a displacement of 0F8H is a one-instruction infinite loop.
Bytes: 2
Cycles: 2
Encoding: 1 0 0 0 0 0 0 0
Operation:

SJMP (PC) + 2
Operation:

SJMP (PC) + 1
Operation:
### SUBB A, <src-byte>

**Function:** Subtract with borrow
**Description:** SUBB subtracts the indicated variable and the carry flag together from the Accumulator, leaving the result in the Accumulator. SUBB sets the carry (Borrow) flag if a borrow is needed for bit 7 and clears it otherwise. If C was set before executing a SUBB instruction, this indicates that a borrow was needed for the previous step in a multiple-precision subtraction, so the carry is subtracted from the Accumulator along with the source operand. If the carry flag is set after adding and the carry was cleared, that bit is added. If C was not set, or if a borrow is needed for bit 3 and cleared otherwise, CV is set if a borrow is needed into bit 6, but not into bit 7, or into bit 7, but not bit 6.

When subtracting signed integers, CV indicates a negative number produced when a negative number is subtracted from a positive value, or a positive result when a positive number is subtracted from a negative.

The source operand allows for addressing modes: register, direct, register-indirect, or immediate.

**Example:** The Accumulator holds DCBH (11001001B), register 2 holds 64H (01010100B), and the carry flag is set. The instruction:

```
SUBB A, R2
```

will leave the value 74H (01110100B) in the accumulator, with the carry flag and AC cleared but CV set.

Notice that DCBH minus 64H is 72H. The difference between this and the above result is due to the carry (borrow) flag being set before the operation. If the state of the carry is not known before starting a single or multiple-precision subtraction, it should be explicitly cleared by CRR.C instruction.

<table>
<thead>
<tr>
<th>Bytes</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cycles</td>
<td>1</td>
</tr>
<tr>
<td>Encoding</td>
<td>1 0 0 1 1 r r r r</td>
</tr>
<tr>
<td>Operation</td>
<td>SUBB (A) ← (A) − (C) − (Ri)</td>
</tr>
</tbody>
</table>

### SWAP A

**Function:** Swap nibbles within the Accumulator
**Description:** SWAP A exchanges the low and high-order nibbles (four-bit fields) of the Accumulator bits 5 through 1 and bits 7 through 3. The operation can also be thought of as a 4-bit rotate instruction. No flags are affected.

**Example:** When the Accumulator holds the value 0C58H (110001001000B), the instruction:

```
SWAP A
```

leaves the Accumulator holding the value 85C0H (011010001100B).

<table>
<thead>
<tr>
<th>Bytes</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cycles</td>
<td>1</td>
</tr>
<tr>
<td>Encoding</td>
<td>1 1 0 0 0 1 0 9</td>
</tr>
<tr>
<td>Operation</td>
<td>SWAP (A), D (A), L</td>
</tr>
</tbody>
</table>

### XCH A, <byte>

**Function:** Exchange Accumulator with byte variable
**Description:** XCH loads the Accumulator with the contents of the indicated variable, at the same time writing the original Accumulator contents to the indicated variable. The source/descriptor operand can use register, direct, or register-indirect addressing.

**Example:** When internal RAM location 20H holds the value 75H (01110101B), the following instruction:

```
XCH A, @R0
```

leaves RAM location 20H holding the value 5F (01111111B) and 75H (01110101B) in the accumulator.

<table>
<thead>
<tr>
<th>Bytes</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cycles</td>
<td>1</td>
</tr>
<tr>
<td>Encoding</td>
<td>1 1 0 0 1 r r r</td>
</tr>
<tr>
<td>Operation</td>
<td>XCH (A), D (Ri)</td>
</tr>
</tbody>
</table>

### XCH A, <direct>

**Function:** Exchange Accumulator with direct variable
**Description:** XCH loads the Accumulator with the contents of the indicated variable, at the same time writing the original Accumulator contents to the indicated variable. The source/descriptor operand can use register, direct, or register-indirect addressing.

**Example:** When the Accumulator holds the value 3FH (01111111B), the following instruction:

```
XCH A, @R0
```

leaves the Accumulator holding the value 0F (00111111B) and 3FH (01111111B) in the accumulator.

<table>
<thead>
<tr>
<th>Bytes</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cycles</td>
<td>1</td>
</tr>
<tr>
<td>Encoding</td>
<td>1 1 0 0 1 0 1 1</td>
</tr>
<tr>
<td>Operation</td>
<td>XCH (A), 0 (direct)</td>
</tr>
</tbody>
</table>

### XCH A, <register>

**Function:** Exchange Accumulator with register variable
**Description:** XCH loads the Accumulator with the contents of the indicated variable, at the same time writing the original Accumulator contents to the indicated variable. The source/descriptor operand can use register, direct, or register-indirect addressing.

**Example:** When the Accumulator holds the value 3FH (01111111B), the following instruction:

```
XCH A, @R0
```

leaves the Accumulator holding the value 0F (00111111B) in the accumulator.

<table>
<thead>
<tr>
<th>Bytes</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cycles</td>
<td>1</td>
</tr>
<tr>
<td>Encoding</td>
<td>1 1 0 0 0 1 1</td>
</tr>
<tr>
<td>Operation</td>
<td>XCH (A), 0 (Ri)</td>
</tr>
</tbody>
</table>
XCHD A @ R0

Function: Exchange Digit

Description: XCHD exchanges the low-order nibble of the Accumulator (bits 3 through 0), generally representing a hexadecimal or BCD digit, with that of the external RAM location indirectly addressed by the specified register. The high-order nibbles (bits 7 through 4) of each register are not affected. No flags are affected.

Example: RD contains the address 20H. The Accumulator holds the value 76H (01110110B). Internal RAM location 20H holds the value 72H (01110110B). The following instruction:

XCHD A @ R0

leaves RAM location 20H holding the value 76H (01110110B) and 36H (00110110B) in the Accumulator.

Bytes: 1

Cycles: 1

Encoding: 1 1 1 0 1 0 1 1 1

Operation: XCHD

-byte>, <src-byte>

Function: Logical Exclusive-OR for byte variables

Description: XRL performs the bitwise logical Exclusive-OR operation between the indicated variables, storing the results in the destination. No flags are affected.

The two operands allow six addressing mode combinations. When the destination is the Accumulator, the source can use register, direct, register-indirect, or immediate addressing when the destination is a direct address, the source can be the Accumulator or immediate data.

Example: If the Accumulator holds 0C3H (11000011B) and register 0 holds 00AH (01010101B) then the instruction: XRL A, R0

leaves the Accumulator holding the value 0C1H (11000001B).

When the destination is a directly addressed byte, this instruction can complement combinations of bits in any RAM location or hardware register. The pattern of bits to be complemented is then determined by a mask byte, either a constant contained in the instruction or a variable computed in the Accumulator at run-time. The following instruction:

XRL P1, 00111100B

complements bits 3, 4, and 0 of output Port 1.

XRL A, R0

Bytes: 1

Cycles: 1

Encoding: 1 1 1 0 1 1 1 1 r r

Operation: XRL

(A) ← (A) ⊕ (R0)
# NPN general purpose transistors

**BC546; BC547; BC548**

### FEATURES
- Low current (max. 100 mA)
- Low voltage (max. 65 V)

### APPLICATIONS
- General purpose switching and amplification.

### DESCRIPTION
NPN transistor in a TO-92; SOT54 plastic package.
PNP complements: BC556, BC557 and BC558.

### QUICK REFERENCE DATA

<table>
<thead>
<tr>
<th>SYMBOL</th>
<th>PARAMETER</th>
<th>CONDITIONS</th>
<th>MIN.</th>
<th>MAX.</th>
<th>UNIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>V_{CEO}</td>
<td>collector-base voltage</td>
<td>open emitter</td>
<td>50</td>
<td>60</td>
<td>V</td>
</tr>
<tr>
<td></td>
<td>BC546</td>
<td></td>
<td>50</td>
<td>60</td>
<td>V</td>
</tr>
<tr>
<td></td>
<td>BC547</td>
<td></td>
<td>50</td>
<td>60</td>
<td>V</td>
</tr>
<tr>
<td></td>
<td>BC548</td>
<td></td>
<td>50</td>
<td>60</td>
<td>V</td>
</tr>
<tr>
<td>V_{CEO}</td>
<td>collector-emitter voltage</td>
<td>open base</td>
<td>46</td>
<td>46</td>
<td>V</td>
</tr>
<tr>
<td></td>
<td>BC546</td>
<td></td>
<td>46</td>
<td>46</td>
<td>V</td>
</tr>
<tr>
<td></td>
<td>BC547</td>
<td></td>
<td>46</td>
<td>46</td>
<td>V</td>
</tr>
<tr>
<td></td>
<td>BC548</td>
<td></td>
<td>46</td>
<td>46</td>
<td>V</td>
</tr>
<tr>
<td>I_{CM}</td>
<td>peak collector current</td>
<td></td>
<td>200</td>
<td></td>
<td>mA</td>
</tr>
<tr>
<td>P_{T}</td>
<td>total power dissipation</td>
<td>T_{CASE} ≤ 25°C</td>
<td>600</td>
<td></td>
<td>mW</td>
</tr>
<tr>
<td>R_{EBC}</td>
<td>DC current gain</td>
<td>I_E = 2 mA, V_CE = 5 V</td>
<td>110</td>
<td>460</td>
<td></td>
</tr>
<tr>
<td></td>
<td>BC546</td>
<td></td>
<td>110</td>
<td>460</td>
<td></td>
</tr>
<tr>
<td></td>
<td>BC547</td>
<td></td>
<td>110</td>
<td>460</td>
<td></td>
</tr>
<tr>
<td></td>
<td>BC548</td>
<td></td>
<td>110</td>
<td>460</td>
<td></td>
</tr>
<tr>
<td>f_{T}</td>
<td>transition frequency</td>
<td>I_E = 10 mA, V_CE = 5 V, f = 100 MHz</td>
<td>100</td>
<td></td>
<td>MHz</td>
</tr>
</tbody>
</table>
### LIMITING VALUES

In accordance with the Absolute Maximum Rating System (IEC 134).

<table>
<thead>
<tr>
<th>SYMBOL</th>
<th>PARAMETER</th>
<th>CONDITIONS</th>
<th>MIN.</th>
<th>MAX.</th>
<th>UNIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>V(&quot;b&quot;)</td>
<td>collector-base voltage</td>
<td>open emitter</td>
<td>BC546</td>
<td>60 V</td>
<td>V</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>BC547</td>
<td>60 V</td>
<td>V</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>BC548</td>
<td>60 V</td>
<td>V</td>
</tr>
<tr>
<td>V(&quot;b&quot;)</td>
<td>collector-emitter voltage</td>
<td>open base</td>
<td>BC546</td>
<td>65 V</td>
<td>V</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>BC547</td>
<td>65 V</td>
<td>V</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>BC548</td>
<td>65 V</td>
<td>V</td>
</tr>
<tr>
<td>V(&quot;b&quot;)</td>
<td>emitter-base voltage</td>
<td>open collector</td>
<td>BC546</td>
<td>6 V</td>
<td>V</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>BC547</td>
<td>6 V</td>
<td>V</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>BC548</td>
<td>6 V</td>
<td>V</td>
</tr>
<tr>
<td>IC</td>
<td>collector current (DC)</td>
<td></td>
<td>BC546</td>
<td>100 mA</td>
<td>mA</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>BC547</td>
<td>100 mA</td>
<td>mA</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>BC548</td>
<td>100 mA</td>
<td>mA</td>
</tr>
<tr>
<td>ICM</td>
<td>peak collector current</td>
<td></td>
<td>BC546</td>
<td>200 mA</td>
<td>mA</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>BC547</td>
<td>200 mA</td>
<td>mA</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>BC548</td>
<td>200 mA</td>
<td>mA</td>
</tr>
<tr>
<td>PDM</td>
<td>total power dissipation</td>
<td>T&lt;sub&gt;case&lt;/sub&gt; 25 C; note 1</td>
<td>500 mW</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T&lt;sub&gt;stg&lt;/sub&gt;</td>
<td>storage temperature</td>
<td></td>
<td>85</td>
<td>150 °C</td>
<td></td>
</tr>
<tr>
<td>T&lt;sub&gt;θJA&lt;/sub&gt;</td>
<td>junction temperature</td>
<td></td>
<td>85</td>
<td>150 °C</td>
<td></td>
</tr>
<tr>
<td>T&lt;sub&gt;θJA&lt;/sub&gt;</td>
<td>operating ambient temperature</td>
<td></td>
<td>85</td>
<td>150 °C</td>
<td></td>
</tr>
</tbody>
</table>

**Note:**
1. Transistor mounted on an FR4 printed-circuit board.

### THERMAL CHARACTERISTICS

<table>
<thead>
<tr>
<th>SYMBOL</th>
<th>PARAMETER</th>
<th>CONDITIONS</th>
<th>VALUE</th>
<th>UNIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>R&lt;sub&gt;θJA&lt;/sub&gt;</td>
<td>thermal resistance from junction to ambient</td>
<td>note 1</td>
<td>0.25</td>
<td>°C/W</td>
</tr>
</tbody>
</table>

**Note:**
1. Transistor mounted on an FR4 printed-circuit board.

### CHARACTERISTICS

T<sub>case</sub> = 25 °C unless otherwise specified.

<table>
<thead>
<tr>
<th>SYMBOL</th>
<th>PARAMETER</th>
<th>CONDITIONS</th>
<th>VALUE</th>
<th>TYP.</th>
<th>MAX.</th>
</tr>
</thead>
<tbody>
<tr>
<td>IC</td>
<td>collector cut-off current</td>
<td>V&lt;sub&gt;ce&lt;/sub&gt; = 30 V;</td>
<td>5</td>
<td>μA</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>T&lt;sub&gt;case&lt;/sub&gt; = 150 °C</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IC&lt;sub&gt;em&lt;/sub&gt;</td>
<td>emitter cut-off current</td>
<td>V&lt;sub&gt;ce&lt;/sub&gt; = 30 V;</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>T&lt;sub&gt;case&lt;/sub&gt; = 150 °C</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>V&lt;sub&gt;β&lt;/sub&gt;DC</td>
<td>DC current gain</td>
<td>BC546A, BC547A, BC548A</td>
<td>90</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>BC546B, BC547B, BC548B</td>
<td>100</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>BC546C, BC547C, BC548C</td>
<td>200</td>
<td></td>
<td></td>
</tr>
<tr>
<td>V&lt;sub&gt;β&lt;/sub&gt;AC</td>
<td>AC current gain</td>
<td>BC546A, BC547A, BC548A</td>
<td>110</td>
<td>200 mW</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>BC546B, BC547B, BC548B</td>
<td>120</td>
<td>240 mW</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>BC546C, BC547C, BC548C</td>
<td>130</td>
<td>280 mW</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>BC546, BC547, BC548</td>
<td>110</td>
<td>200 mW</td>
<td></td>
</tr>
<tr>
<td>V&lt;sub&gt;CE&lt;/sub&gt;</td>
<td>collector-emitter saturation voltage</td>
<td>V&lt;sub&gt;ce&lt;/sub&gt; = 5 V;</td>
<td>0.6</td>
<td>mA</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>note 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>V&lt;sub&gt;BE&lt;/sub&gt;</td>
<td>base-emitter saturation voltage</td>
<td>V&lt;sub&gt;ce&lt;/sub&gt; = 0.5 V;</td>
<td>700</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>note 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>V&lt;sub&gt;C&lt;/sub&gt;</td>
<td>base-emitter voltage</td>
<td>V&lt;sub&gt;ce&lt;/sub&gt; = 5 V;</td>
<td>6.4</td>
<td>V</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>note 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C&lt;sub&gt;JC&lt;/sub&gt;</td>
<td>collector capacitance</td>
<td>V&lt;sub&gt;ce&lt;/sub&gt; = 30 V;</td>
<td>11.5</td>
<td>pF</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>f = 1 MHz</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C&lt;sub&gt;BE&lt;/sub&gt;</td>
<td>emitter capacitance</td>
<td>V&lt;sub&gt;ce&lt;/sub&gt; = 30 V;</td>
<td>8.0</td>
<td>pF</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>f = 1 MHz</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>f&lt;sub&gt;TF&lt;/sub&gt;</td>
<td>transition frequency</td>
<td>V&lt;sub&gt;ce&lt;/sub&gt; = 5 V;</td>
<td>100</td>
<td>MHz</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>I&lt;sub&gt;ce&lt;/sub&gt; = 100 mA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>f&lt;sub&gt;GR&lt;/sub&gt;</td>
<td>noise figure</td>
<td>V&lt;sub&gt;ce&lt;/sub&gt; = 5 V;</td>
<td>2</td>
<td>dB</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>R&lt;sub&gt;in&lt;/sub&gt; = 5 kΩ</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Notes:**
1. V<sub>ce</sub> increases by about 0.5 mV/K with increasing temperature.
2. V<sub>be</sub> increases by about 2 mV/K with increasing temperature.
NPN general purpose transistors

Fig.2 DC current gain: typical values.

Fig.3 DC current gain: typical values.

Fig.4 DC current gain: typical values.
NPN general purpose transistors
BC546; BC547; BC548

PACKAGE OUTLINE
Plastic single-ended leaded (through hole) package; 3 leads
SO754

DEFINITIONS

<table>
<thead>
<tr>
<th>Data Sheet Status</th>
<th>This data sheet contains general or goal specifications for product development.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preliminary</td>
<td>This data sheet contains preliminary data. Supplementary data may be published later.</td>
</tr>
<tr>
<td>Product</td>
<td>This data sheet contains final product specifications.</td>
</tr>
</tbody>
</table>

Limiting values

Limiting values given are in accordance with the Absolute Maximum Rating System (IEC 134). Stress above one or more of the limiting values may cause permanent damage to the device. These are stress ratings only and operation of the device at these or any other conditions above those given in the Characteristics sections of the specification is not implied. Exposure to limiting values for extended periods may affect device reliability.

Application information

Where application information is given, it is advisory and does not form part of the specification.

LIFE SUPPORT APPLICATIONS

These products are not designed for use in life support appliances, devices, or systems where malfunction of these products can reasonably be expected to result in personal injury. Philips customers using or selling these products for use in such applications do so at their own risk and agree to fully indemnify Philips for any damages resulting from such improper use or sale.
BC559; BC560
PNP general purpose transistors

FEATURES
• Low current (max. 160 mA)
• Low voltage (max. 45 V)

APPLICATIONS
• General purpose switching and amplification.

DESCRIPTION
PNP transistor in a TO-92, SOT14 package.
NPN complements: BC559 and BC560.

QUICK REFERENCE DATA
<table>
<thead>
<tr>
<th>SYMBOL</th>
<th>PARAMETER</th>
<th>CONDITIONS</th>
<th>MIN</th>
<th>MAX</th>
<th>UNIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>VCEO</td>
<td>collector-emitter voltage</td>
<td>open emitter</td>
<td>BC559</td>
<td>V</td>
<td>-30</td>
</tr>
<tr>
<td></td>
<td></td>
<td>open emitter</td>
<td>BC560</td>
<td>V</td>
<td>-30</td>
</tr>
<tr>
<td>VCEO</td>
<td>collector-emitter voltage</td>
<td>open emitter</td>
<td>BC559</td>
<td>V</td>
<td>-30</td>
</tr>
<tr>
<td></td>
<td></td>
<td>open emitter</td>
<td>BC560</td>
<td>V</td>
<td>-40</td>
</tr>
<tr>
<td>IC</td>
<td>DC current gain</td>
<td>TTh = +25 °C</td>
<td>0.5</td>
<td>800</td>
<td>mA</td>
</tr>
<tr>
<td>fT</td>
<td>transition frequency</td>
<td>Ic = 10 mA, VCE = -6 V</td>
<td>500</td>
<td>800</td>
<td>MHz</td>
</tr>
</tbody>
</table>
PNP general purpose transistors
BC559; BC560

LIMITING VALUES

<table>
<thead>
<tr>
<th>SYMBOL</th>
<th>PARAMETER</th>
<th>CONDITIONS</th>
<th>MIN.</th>
<th>MAX.</th>
<th>UNIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>VCC0</td>
<td>collector-base voltage</td>
<td>open emitter</td>
<td>–</td>
<td>–10</td>
<td>V</td>
</tr>
<tr>
<td>VCC0</td>
<td>collector-base voltage</td>
<td>BC559</td>
<td>–</td>
<td>–10</td>
<td>V</td>
</tr>
<tr>
<td>VCC0</td>
<td>collector-base voltage</td>
<td>BC560</td>
<td>–</td>
<td>–10</td>
<td>V</td>
</tr>
<tr>
<td>VCEO</td>
<td>collector-emitter voltage</td>
<td>open base</td>
<td>–</td>
<td>–30</td>
<td>V</td>
</tr>
<tr>
<td>VCEO</td>
<td>collector-emitter voltage</td>
<td>BC559</td>
<td>–</td>
<td>–30</td>
<td>V</td>
</tr>
<tr>
<td>VCEO</td>
<td>collector-emitter voltage</td>
<td>BC560</td>
<td>–</td>
<td>–45</td>
<td>V</td>
</tr>
<tr>
<td>VBE0</td>
<td>emitter-base voltage</td>
<td>open collector</td>
<td>–</td>
<td>–5</td>
<td>V</td>
</tr>
<tr>
<td>IC</td>
<td>collector current (DC)</td>
<td>–</td>
<td>–100</td>
<td>mA</td>
<td></td>
</tr>
<tr>
<td>ICN</td>
<td>peak collector current (DC)</td>
<td>–</td>
<td>–200</td>
<td>mA</td>
<td></td>
</tr>
<tr>
<td>IMAX</td>
<td>peak-base current</td>
<td>–</td>
<td>–200</td>
<td>mA</td>
<td></td>
</tr>
<tr>
<td>Ptot</td>
<td>total power dissipation</td>
<td>Ta = 25 ºC</td>
<td>500</td>
<td>W</td>
<td></td>
</tr>
<tr>
<td>Ta</td>
<td>storage temperature</td>
<td>–</td>
<td>–65</td>
<td>ºC</td>
<td></td>
</tr>
<tr>
<td>Tj</td>
<td>junction temperature</td>
<td>–</td>
<td>–150</td>
<td>ºC</td>
<td></td>
</tr>
<tr>
<td>TAMB</td>
<td>operating ambient temperature</td>
<td>–65 to +150 ºC</td>
<td>–</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

THERMAL CHARACTERISTICS

<table>
<thead>
<tr>
<th>SYMBOL</th>
<th>PARAMETER</th>
<th>CONDITIONS</th>
<th>VALUE</th>
<th>UNIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>RθJA</td>
<td>thermal resistance from junction to ambient</td>
<td>–</td>
<td>250</td>
<td>K/W</td>
</tr>
</tbody>
</table>

Note
1. Transistor mounted on an FR4 printed-circuit board.

CHARACTERISTICS

<table>
<thead>
<tr>
<th>SYMBOL</th>
<th>PARAMETER</th>
<th>CONDITIONS</th>
<th>MIN.</th>
<th>TYP.</th>
<th>MAX.</th>
<th>UNIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>ICBO</td>
<td>collector cut-off current</td>
<td>Ic = 0; Vcc = –30 V</td>
<td>–</td>
<td>–1</td>
<td>–15</td>
<td>mA</td>
</tr>
<tr>
<td>IB</td>
<td>collector cut-off current</td>
<td>Ic = 0; Vcc = –30 V, Tj = 125ºC</td>
<td>–</td>
<td>–4</td>
<td>–9</td>
<td>mA</td>
</tr>
<tr>
<td>IEQ</td>
<td>emitter cut-off current</td>
<td>Ic = 0; Vcc = –5 V</td>
<td>–</td>
<td>–</td>
<td>–100</td>
<td>mA</td>
</tr>
<tr>
<td>Pre</td>
<td>DC current gain</td>
<td>Ic = –2 mA; Icc = –5 V; see Figs 2, 3 and 4</td>
<td>125</td>
<td>–</td>
<td>600</td>
<td></td>
</tr>
<tr>
<td>Pre</td>
<td>DC current gain</td>
<td>BC559A</td>
<td>115</td>
<td>–</td>
<td>600</td>
<td></td>
</tr>
<tr>
<td></td>
<td>BC559B, BC560B</td>
<td>120</td>
<td>–</td>
<td>700</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>BC559C, BC560C</td>
<td>420</td>
<td>–</td>
<td>800</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VCEO</td>
<td>collector-emitter saturation voltage</td>
<td>Ic = –10 mA, Ie = –0.3 mA</td>
<td>–82</td>
<td>–300</td>
<td>mV</td>
<td></td>
</tr>
<tr>
<td>VCEO</td>
<td>collector-emitter saturation voltage</td>
<td>Ic = –100 mA, Ie = –0.3 mA</td>
<td>–183</td>
<td>–600</td>
<td>mV</td>
<td></td>
</tr>
<tr>
<td>VCEO</td>
<td>base-emitter saturation voltage</td>
<td>Ic = –10 mA, Ie = –0.3 mA, note 1</td>
<td>–750</td>
<td>–1000</td>
<td>mV</td>
<td></td>
</tr>
<tr>
<td>VCEO</td>
<td>base-emitter saturation voltage</td>
<td>Ic = –100 mA, Ie = –0.3 mA, note 1</td>
<td>–990</td>
<td>–1500</td>
<td>mV</td>
<td></td>
</tr>
</tbody>
</table>

Notes
1. VCEO decreases by about ~1.7 mV/K with increasing temperature.
2. VCEO decreases by about ~2 mV/K with increasing temperature.
PNP general purpose transistors

BC559; BC560

DEFINITIONS

<table>
<thead>
<tr>
<th>Data sheet status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Objective specification</td>
<td>This data sheet contains target or goal specifications for product development.</td>
</tr>
<tr>
<td>Preliminary specification</td>
<td>This data sheet contains preliminary data; supplementary data may be published later.</td>
</tr>
<tr>
<td>Product specification</td>
<td>This data sheet contains final product specifications.</td>
</tr>
</tbody>
</table>

Limiting values

Limiting values given are in accordance with the Absolute Maximum Rating System (IEC 734). Values above one or more of the limiting values may cause permanent damage to the device. These are stress ratings only and operation of the device at these or at any other conditions above those given in the Characteristics sections of the specification is not implied. Exposure to limiting values for extended periods may affect device reliability.

Application information

Where application information is given, it is advisory and does not form part of the specification.

LIFE SUPPORT APPLICATIONS

These products are not designed for use in life support appliances, devices, or systems where malfunction of these products can reasonably be expected to result in personal injury. Philips customers using or selling these products for use in such applications do so at their own risk and agree to fully indemnify Philips for any damages resulting from such improper use or sale.

Philips Semiconductors – a worldwide company

For all other countries contact Philips Semiconductors, Marketing & Sales Communications

R&D Center P.O. Box 7777 1400 AB Amstelveen, The Netherlands


All rights reserved. Reproduction, adaptation or translation without prior written permission is prohibited, except as allowed under copyright laws.

The information presented in this document is believed to be accurate and reliable. However, the information is presented without warranty of any kind, either expressed or implied. Philips reserves the right to make changes without further notice to any products herein. Philips assumes no responsibility for its use, nor for any infringements of patents or other rights of third parties, which are incurred by its sale or use. These specifications are subject to change without notice.

Printed in The Netherlands
### SINGLE DIGIT LED DISPLAYS

<table>
<thead>
<tr>
<th>Model</th>
<th>Description</th>
<th>Model</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>D7-30</td>
<td></td>
<td>D7-35</td>
<td></td>
</tr>
<tr>
<td>BF-7444D</td>
<td></td>
<td>BF-7414D</td>
<td></td>
</tr>
<tr>
<td>D7-21</td>
<td></td>
<td>D7-32</td>
<td></td>
</tr>
<tr>
<td>BF-7451C</td>
<td></td>
<td>BF-7451C</td>
<td></td>
</tr>
<tr>
<td>D7-23</td>
<td></td>
<td>D7-34</td>
<td></td>
</tr>
<tr>
<td>BF-7451C</td>
<td></td>
<td>BF-7451C</td>
<td></td>
</tr>
</tbody>
</table>

**NOTES:**
1. All dimensions are in millimeters (mm).
2. Specifications are subject to change without notice.
3. Following in JEDEC standard P1.75.
4. SOP: Pin 10, 50% Nominal.
LAMPIRAN C

PERANGKAT LUNAK
PROGRAM PENGHITUNG TARIF KENDARAAN UMUM

#INCLUDE "8051.H"

#define ORG     .ORG
#define END     .end
#define equ     .equ
#define data    .equ
#define bit     .equ

P0      .equ    080H    ;Port 0 - Not present on the 89C2051
SP      .equ    081H    ;Stack pointer
DPL     .equ    082H    ;Data pointer low, part of 16 bit reg with DPH
DPH     .equ    083H
PCON    .equ    087H    ;Power control, not bit addressable,
TCON    .equ    088H    ;Timer/counter control register, see bit list below
TMOD    .equ    089H    ;Timer/counter mode control register
TL0     .equ    08AH    ;Timer 0 low
DPL     .equ    082H    ;Data pointer low, part of 16 bit reg with DPH
TMOD    .equ    089H    ;Timer/counter mode control register
TL0     .equ    08AH    ;Timer 0 low
TL1     .equ    08BH    ;Timer 1 low
TH0     .equ    08CH    ;Timer 0 high - also reload val in 8bit auto RL mode
TH1     .equ    08DH    ;Timer 1 high - also reload val in 8bit auto RL mode
P1      .equ    090H    ;Port 1
SBUF    .equ    099H    ;Serial buffer - read for Serial Rx, written to Tx
P2      .equ    0A0H    ;Port 2 - Not present on 89C2051
IE      .equ    0A8H    ;Interrupt enable register, see bit list below
IP      .equ    0B0H    ;Port 3
IP      .equ    0B0H    ;Port 3
IP      .equ    0B8H    ;Interrupt priority register, see bit list below
T2CON   .equ    0C8H    ;8052, 80154 only
RCAP2L  .equ    0CAH    ;8052, 80154 only
RCAP2H  .equ    0CBH    ;8052, 80154 only
TL2     .equ    0CCCH   ;8052, 80154 only
TH2     .equ    0CDH    ;8052, 80154 only
PSW     .equ    0D0H    ;Program status word, see bit list below
ACC     .equ    0E0H    ;Accumulator
B       .equ    0F0H    ;Secondary Accumulator, used in Multiply and Divide
IOCON   .equ    0F8H    ;80154 only

PORT 0 BITS
P0.0    .equ    080H    ;Port 0 bit 0
P0.1    .equ    081H    ;Port 0 bit 1
P0.2    .equ    082H    ;Port 0 bit 2
P0.3    .equ    083H    ;Port 0 bit 3
P0.4    .equ    084H    ;Port 0 bit 4
P0.5    .equ    085H    ;Port 0 bit 5
P0.6    .equ    086H    ;Port 0 bit 6
P0.7    .equ    087H    ;Port 0 bit 7

PORT 1 BITS
P1.0    .equ    090H    ;Port 1 bit 0
P1.1    .equ    091H    ;Port 1 bit 1
P1.2    .equ    092H    ;Port 1 bit 2
P1.3    .equ    093H    ;Port 1 bit 3
P1.4    .equ    094H    ;Port 1 bit 4
P1.5    .equ    095H    ;Port 1 bit 5
P1.6    .equ    096H    ;Port 1 bit 6
P1.7    .equ    097H    ;Port 1 bit 7

PORT 2 BITS
P2.0    .equ    0A0H    ;Port 2 bit 0
P2.1    .equ    0A1H    ;Port 2 bit 1
Perangkat Lunak C - 2

0066+ 0000    P2.2 .equ 0A2H ;Port 2 bit 2
0067+ 0000    P2.3 .equ 0A3H ;Port 2 bit 3
0068+ 0000    P2.4 .equ 0A4H ;Port 2 bit 4
0069+ 0000    P2.5 .equ 0A5H ;Port 2 bit 5
0070+ 0000    P2.6 .equ 0A6H ;Port 2 bit 6
0071+ 0000    P2.7 .equ 0A7H ;Port 2 bit 7
0072+ 0000
0073+ 0000    ;PORT 3 BITS
0074+ 0000    P3.0 .equ 0B0H ;Port 3 bit 0
0075+ 0000    P3.1 .equ 0B1H ;Port 3 bit 1
0076+ 0000    P3.2 .equ 0B2H ;Port 3 bit 2
0077+ 0000    P3.3 .equ 0B3H ;Port 3 bit 3
0078+ 0000    P3.4 .equ 0B4H ;Port 3 bit 4
0079+ 0000    P3.5 .equ 0B5H ;Port 3 bit 5
0080+ 0000    P3.6 .equ 0B6H ;Port 3 bit 6
0081+ 0000    P3.7 .equ 0B7H ;Port 3 bit 7
0082+ 0000
0083+ 0000    ;ACCUMULATOR BITS
0084+ 0000    ACC.0 .equ 0E0H ;Acc bit 0
0085+ 0000    ACC.1 .equ 0E1H ;Acc bit 1
0086+ 0000    ACC.2 .equ 0E2H ;Acc bit 2
0087+ 0000    ACC.3 .equ 0E3H ;Acc bit 3
0088+ 0000    ACC.4 .equ 0E4H ;Acc bit 4
0089+ 0000    ACC.5 .equ 0E5H ;Acc bit 5
0090+ 0000    ACC.6 .equ 0E6H ;Acc bit 6
0091+ 0000    ACC.7 .equ 0E7H ;Acc bit 7
0092+ 0000
0093+ 0000    ;B REGISTER BITS
0094+ 0000    B.0 .equ 0F0H ;Breg bit 0
0095+ 0000    B.1 .equ 0F1H ;Breg bit 1
0096+ 0000    B.2 .equ 0F2H ;Breg bit 2
0097+ 0000    B.3 .equ 0F3H ;Breg bit 3
0098+ 0000    B.4 .equ 0F4H ;Breg bit 4
0099+ 0000    B.5 .equ 0F5H ;Breg bit 5
0100+ 0000    B.6 .equ 0F6H ;Breg bit 6
0101+ 0000    B.7 .equ 0F7H ;Breg bit 7
0102+ 0000
0103+ 0000    ;PSW REGISTER BITS
0104+ 0000    P .equ 0D0H ;Parity flag
0105+ 0000    F1 .equ 0D1H ;User flag 1
0106+ 0000    OV .equ 0D2H ;Overflow flag
0107+ 0000    RS0 .equ 0D3H ;Register bank select 1
0108+ 0000    RS1 .equ 0D4H ;Register bank select 0
0109+ 0000    F0 .equ 0D5H ;User flag 0
0110+ 0000    AC .equ 0D6H ;Auxiliary carry flag
0111+ 0000    CY .equ 0D7H ;Carry flag
0112+ 0000
0113+ 0000    ;TCON REGISTER BITS
0114+ 0000    IT0 .equ 088H ;Intr 0 type control
0115+ 0000    IE0 .equ 089H ;Intr 0 edge flag
0116+ 0000    IT1 .equ 08AH ;Intr 1 type control
0117+ 0000    IE1 .equ 08BH ;Intr 1 edge flag
0118+ 0000    TR0 .equ 08CH ;Timer 0 run
0119+ 0000    TF0 .equ 08DH ;Timer 0 overflow
0120+ 0000    TR1 .equ 08EH ;Timer 1 run
0121+ 0000    TF1 .equ 08FH ;Timer 1 overflow
0122+ 0000
0123+ 0000    ;SCON REGISTER BITS
0124+ 0000    RI .equ 098H ;RX Intr flag
0125+ 0000    TI .equ 099H ;TX Intr flag
0126+ 0000    RB8 .equ 09AH ;RX 9th bit
0127+ 0000    TB8 .equ 09BH ;TX 9th bit
0128+ 0000    REN .equ 09CH ;Enable RX flag
0129+ 0000    SM2 .equ 09DH ;8/9 bit select flag
0130+ 0000    SM1 .equ 09EH ;Serial mode bit 1
0131+ 0000    SM0 .equ 09FH ;Serial mode bit 0
0132+ 0000
0133+ 0000    ;IE REGISTER BITS
0134+ 0000    EX0 .equ 0A8H ;External intr 0
0135+ 0000 ET0 .equ 0A9H ;Timer 0 intr
0136+ 0000 EX1 .equ 0AAH ;External intr 1
0137+ 0000 ET1 .equ 0ABH ;Timer 1 intr
0138+ 0000 ES .equ 0ACH ;Serial port intr
0139+ 0000 ET2 .equ 0ADH ;Timer 2 intr
0140+ 0000 ;Reserved 0AEH Reserved
0141+ 0000 EA .equ 0AFH ;Global intr enable
0142+ 0000
0143+ 0000 ;IP REGISTER BITS
0144+ 0000 PX0 .equ 0B8H ;Priority level-External intr 0
0145+ 0000 PT0 .equ 0B9H ;Priority level-Timer 0 intr
0146+ 0000 PX1 .equ 0BAH ;Priority level-External intr 1
0147+ 0000 PT1 .equ 0BBH ;Priority level-Timer 1 intr
0148+ 0000 PS .equ 0BCH ;Priority level-Serial port intr
0149+ 0000 PT2 .equ 0BDH ;Priority level-Timer 2 intr
0150+ 0000 ;Reserved 0BEH Reserved
0151+ 0000 PCT .equ 0BFFH ;Global priority level
0152+ 0000
0153+ 0000 ;IOC REGISTER BITS 80154 ONLY
0154+ 0000 ALF .equ 0BFH ;Power down port condition
0155+ 0000 P1HZ .equ 0F9H ;Port 1 control
0156+ 0000 P2HZ .equ 0FAH ;Port 2 control
0157+ 0000 P3HZ .equ 0FBH ;Port 3 control
0158+ 0000 IZC .equ 0FCH ;Pullup select
0159+ 0000 SERR .equ 0FDH ;Serial reception error
0160+ 0000 T32 .equ 0FEH ;32 bit timer config
0161+ 0000 WDT .equ 0FFH ;Watchdog config
0162+ 0000
0163+ 0000 ;T2CON REGISTER BITS 8052/80154 ONLY
0164+ 0000 CP/RL2 .equ 0C8H ;Timer 2 capture/reload flag
0165+ 0000 C/T2 .equ 0C9H ;Timer 2 timer/counter select
0166+ 0000 TR2 .equ 0CAH ;Timer 2 start/stop
0167+ 0000 EXEN2 .equ 0CBH ;Timer 2 external enable
0168+ 0000 TCLK .equ 0CCH ;TX clock flag
0169+ 0000 RCLK .equ 0CDH ;RX clock flag
0170+ 0000 EXP2 .equ 0CEH ;Timer 2 external flag
0171+ 0000 TF2 .equ 0CFH ;Timer 2 overflow
0172+ 0000
0173+ 0000
0174+ 0000
0175+ 0000
0176+ 0000
0177+ 0000
0178+ 0000
0179+ 0000
0180+ 0000
0181+ 0000
0182+ 0000
0183+ 0000
0184+ 0000
0185+ 0000
0186+ 0000
0187+ 0000
0188+ 0000
0189+ 0000
0190+ 0000
0191+ 0000
0192+ 0000
0193+ 0000
0194+ 0000
0195+ 0000
0196+ 0000
0197+ 0000
0198+ 0000
0199+ 0000
0200+ 0000
0201+ 0000
0202+ 0000
0203+ 0000
0204+ 0000
0205+ 0000
0206+ 0000
0207+ 0000
0208+ 0000
0209+ 0000
0210+ 0000
0211+ 0000
0212+ 0000
0213+ 0000
0214+ 0000
0215+ 0000
0216+ 0000
0217+ 0000
0218+ 0000
0219+ 0000
0220+ 0000
0221+ 0000
0222+ 0000
0223+ 0000
0224+ 0000
0225+ 0000
0226+ 0000
0227+ 0000
0228+ 0000
0229+ 0000
0230+ 0000
0231+ 0000
0232+ 0000
0233+ 0000
0234+ 0000
0235+ 0000
0236+ 0000
0237+ 0000
0238+ 0000
0239+ 0000
0240+ 0000
0241+ 0000
0242+ 0000
0243+ 0000
0244+ 0000
0245+ 0000
0246+ 0000
0247+ 0000
0248+ 0000
0249+ 0000
0250+ 0000
0251+ 0000
0252+ 0000
0253+ 0000
0254+ 0000
0255+ 0000
0256+ 0000
0257+ 0000
0258+ 0000
0259+ 0000
0260+ 0000
0261+ 0000
0262+ 0000
0263+ 0000
0264+ 0000
0265+ 0000
0266+ 0000
0267+ 0000
0268+ 0000
0269+ 0000
0270+ 0000
0271+ 0000
0272+ 0000
0273+ 0000
0274+ 0000
0275+ 0000
0276+ 0000
0277+ 0000
0278+ 0000
0279+ 0000
0280+ 0000
0281+ 0000
0282+ 0000
0283+ 0000
0284+ 0000
0285+ 0000
0286+ 0000
0287+ 0000
0288+ 0000
0289+ 0000
0290+ 0000
0291+ 0000
0292+ 0000
0293+ 0000
0294+ 0000
0295+ 0000
0296+ 0000
0297+ 0000
0298+ 0000
0299+ 0000
0300+ 0000
0301+ 0000
0302+ 0000
0303+ 0000
0304+ 0000
0305+ 0000

Perangkat Lunak C - 3
Perangkat Lunak C - 4

0036 0040   HITUNGPUTARAN .BLOCK 1 ;PENGHITUNG PUTARAN
0037 0041   METER .BLOCK 1
0038 0042   FLAGPENUMPANG1 .BLOCK 1 ;TANDA PENUMPANG 1 - 4
0039 0043
0040 0043
0041 0043
0042 0000   .ORG 00H
0043 0000 02 01 00   LJMP START
0044 0003
0045 0010   .ORG 100H
0046 0010 75 81 20   START: MOV SP,#20H
0047 0013 75 D0 00   MOV PSW,#0
0048 0016 12 03 3D   LCALL PROC_HAPUSSEMUABUFFER
0049 0019 0109 75 40 00   MOV HITUNGPUTARAN,#0
0050 001C 010C 75 42 00   MOV FLAGPENUMPANG1,#0
0051 001F 010F 75 80 FF   MOV DATA7SEG,#0FFH
0052 0022 0112 75 41 00   MOV METER,#0
0053 0025 0115
0054 0028 0115 75 81 20   LOOPING: MOV SP,#20H
0055 002B 0118 12 01 4D   LCALL PROC_NAIK
0056 002E 011B 12 01 E8   LCALL PROC_TURUN
0057 0031 0121 75 80 FF   MOV DATA7SEG,#0FFH
0058 0034 0124 75 41   MOV METER,#0
0059 0037 0127 75 3B BF   MOV BUFSEG4,#0BFH
0060 003A 012A 12 02 E6   LCALL SCANNING
0061 003D 012D 30 B2 FD   JNB SENSORPIRING,LOOPING
0062 0040 0130 20 B2 FD   LLP: JB SENSORPIRING,LLP
0063 0043 0133 E5 40   MOV A,HITUNGPUTARAN
0064 0046 0135 24 01   ADD A,#01
0065 0049 0137 D4   DA A
0066 004C 013E 75 81 20   LOOPING: MOV SP,#20H
0067 004F 0141 02 01 15   LJMP LOOPING
0068 0052
0069 0052 014D 80 9F 90   MOV A,DATAKARTU
0070 0055 014F 54 0F   ANL A,#0FH
0071 0058 0151 B4 00 01   CJNE A,#00H,CEKKARTU_N
0072 005B 0154 22   RET
0073 005F 0155
0074 0060 0155 12 03 55   CEKKARTU_N: LCALL DELAY
0075 0063 0158 12 03 55   LCALL DELAY
0076 0066 015B 01 0D   CJNE A,#01H,KARTU_N1
0077 0069 015F B4 00 01   CINE A,#00H,KARTU_N1
0078 006C 0162 22   RET
0079 006F 0163
0080 0163 0163 B4 01 1E   KARTU_N1: CJNE A,#01H,KARTU_N2
0081 0166 0166 75 30 00   MOV B,PENUMPANG1_1,#0
0082 0169 0169 75 31 00   MOV B,PENUMPANG1_2,#0
0083 016C 016C E5 42   MOV A,FLAGPENUMPANG1
0084 016F 016E 54 FE   ANL A,#0FEH
0085 0172 0170 44 01   ORL A,#01H
0086 0175 0172 30 F3   MOV FLAGPENUMPANG1,A
0087 0178 0174 75 38 BF   MOV BUFSEG1,#0BFH
0088 017B 0177 75 39 BF   MOV BUFSEG2,#0BFH
0089 017E 017A 75 3A BF   MOV BUFSEG3,#0BFH
0090 0181 017D 3B F9   MOV BUFSEG4,#0F9H
Perangkat Lunak  C - 5

0105 0180 12 02 64        LCALL SCANTAMPILAN
0106 0183 22             RET
0107 0184 B4 02 1E  KARTU_N2: CJNE A,#02H,KARTU_N3
0108 0187 75 32 00        MOV B,PENUMPANG2_1,#0
0109 018A 75 33 00        MOV B,PENUMPANG2_2,#0
0110 018D E5 42          MOV A,FLAGPENUMPANG1
0111 018F 54 FD           ANL A,#0FDH
0112 0191 44 02          ORL A,#02H
0113 0193 F5 42          MOV A,FLAGPENUMPANG1
0114 0195 75 38 BF        MOV BUFSEG1,#0BFH
0115 0198 75 39 BF        MOV BUFSEG2,#0BFH
0116 019B 75 3A BF        MOV BUFSEG3,#0BFH
0117 019E 75 3B A4        MOV BUFSEG4,#0AH
0118 01A1 12 02 64        LCALL SCANTAMPILAN
0119 01A4 22             RET
0120 01A5 B4 03 1E  KARTU_N3: CJNE A,#03H,KARTU_N4
0121 01A8 75 34 00        MOV B,PENUMPANG3_1,#0
0122 01AB 75 35 00        MOV B,PENUMPANG3_2,#0
0123 01AE E5 42          MOV A,FLAGPENUMPANG1
0124 01B0 54 FB           ANL A,#0BFH
0125 01B2 44 04          ORL A,#04H
0126 01B4 F5 42          MOV FLAGPENUMPANG1,A
0127 01B7 75 38 BF        MOV BUFSEG1,#0BFH
0128 01BA 75 39 BF        MOV BUFSEG2,#0BFH
0129 01BB 75 3A BF        MOV BUFSEG3,#0BFH
0130 01BE 75 3B B0        MOV BUFSEG4,#0BH
0131 01C1 12 02 64        LCALL SCANTAMPILAN
0132 01C4 22             RET
0133 01C5 B4 04 1E  KARTU_N4: CJNE A,#04H,KARTU_NN
0134 01C8 75 36 00        MOV B,PENUMPANG4_1,#0
0135 01CC 75 37 00        MOV B,PENUMPANG4_2,#0
0136 01CF E5 42          MOV A,FLAGPENUMPANG1
0137 01D1 54 F7           ANL A,#0FH
0138 01D3 44 08          ORL A,#08H
0139 01D5 F5 42          MOV FLAGPENUMPANG1,A
0140 01D8 75 38 BF        MOV BUFSEG1,#0BFH
0141 01DA 75 39 BF        MOV BUFSEG2,#0BFH
0142 01DB 75 3A BF        MOV BUFSEG3,#0BFH
0143 01DC 75 3B 99        MOV BUFSEG4,#09H
0144 01E0 12 02 64        LCALL SCANTAMPILAN
0145 01E3 22             RET
0146 01E6                KARTU_NN:
0147 01E7 22             RET
0148 01E8
0149 01E8 ;---------------------------------------------------------------
0150 01E8 ; PROCEDURE TURUN:
0151 01E8 ;---------------------------------------------------------------
0152 01E8 PROC_TURUN:
0153 01E9 E5 90          MOV A,DATAKARTU
0154 01EA C4            SWAP A
0155 01EB 54 0F        ANL A,#0FH
0156 01ED B4 00 01      CJNE A,#00H,CEKKARTU_T
0157 01F0 22             RET
0158 01F1
0159 01F2 12 03 55      CEKKARTU_T: LCALL DELAY
0160 01F3 12 03 55      LCALL DELAY
0161 01F5 E5 90          MOV A,DATAKARTU
0162 01F7 C4            SWAP A
0163 01FA 54 0F        ANL A,#0FH
0164 01FB B4 00 01      CJNE A,#00H,KARTU_T1
0165 01FC 22             RET
0166 0200
0167 0201 B4 01 0F  KARTU_T1:      CJNE A,#01H,KARTU_T2
0168 0202 85 30 3E      MOV BIAYA1,B,PENUMPANG1_1
0169 0205 85 31 3F      MOV BIAYA2,B,PENUMPANG1_2
0170 0208 E5 42          MOV A,FLAGPENUMPANG1
0171 020B 54 FE        ANL A,#0FEH
0172 020D F5 42          MOV FLAGPENUMPANG1,A
0173 020F 02 02 48       LJMP TAMPIKANBIAYA
Perangkat Lunak C - 6

0174 0212 B4 02 0F KARTU_T2: CJNE A,#02H,KARTU_T3
0175 0215 85 32 3E MOV BIAYA1,B_PENUMPANG2_1
0176 0218 85 33 3F MOV BIAYA2,B_PENUMPANG2_2
0177 021B E5 42 MOV A,FLAGPENUMPANG1
0178 021D 54 FD ANL A,#0FDH
0179 021F F5 42 MOV FLAGPENUMPANG1,A
0180 0221 02 02 48 LJMP TAMPIKANBIAYA
0181 0224 B4 03 0F KARTU_T3: CJNE A,#03H,KARTU_T4
0182 0227 85 34 3E MOV BIAYA1,B_PENUMPANG3_1
0183 022A 85 35 3F MOV BIAYA2,B_PENUMPANG3_2
0184 022D E5 42 MOV A,FLAGPENUMPANG1
0185 022F 54 FB ANL A,#0FBH
0186 0231 F3 42 MOV FLAGPENUMPANG1,A
0187 0233 02 02 48 LJMP TAMPIKANBIAYA
0188 0236 B4 04 32 KARTU_T4: CJNE A,#04H,KARTU_TN
0189 0239 85 36 3E MOV BIAYA1,B_PENUMPANG4_1
0190 023C 85 37 3F MOV BIAYA2,B_PENUMPANG4_2
0191 023F E5 42 MOV A,FLAGPENUMPANG1
0192 0241 54 F7 ANL A,#0F7H
0193 0243 F5 42 MOV FLAGPENUMPANG1,A
0194 0245 02 02 48 LJMP TAMPIKANBIAYA
0195 0248 0248 E5 3E TAMPILKANBIAYA: MOV A,BIAYA1
0196 024A 12 02 72 LCALL ANDF0
0197 024D F5 38 MOV BUFSEG1,A
0198 024F 54 0F ANDF0: ANL A,#0FH
0199 0251 12 03 2D LCALL CONV7SEG
0200 0254 F5 39 MOV BUFSEG2,A
0201 0256 E5 3F MOV A,BIAYA2
0202 0259 54 01 ANL A,#01H
0203 025B E5 3F MOV A,BIAYA2
0204 025E F5 3A MOV BUFSEG3,A
0205 0261 E5 3F MOV A,BIAYA2
0206 0264 52 02 06 LCALL ANDF0
0207 0266 F5 3B MOV BUFSEG4,A
0208 0268 0264 E5 42 PROC_HITUNG: MOV A,FLAGPENUMPANG1
0209 026A 7B 8F MOV R3,#58F
0210 026C TAMPILTERUS: LCALL SCANNING
0211 026E DB FB DJNZ R3,TAMPILTERUS
0212 026F 026B RET
0213 0270 B4 02 0F KARTU_TN: CJNE A,#00H
0214 0272 02 02 48 LJMP TAMPIKANBIAYA
0215 0274 026B 22 RET
0216 0276 026C ;-------------------------------------------
0217 0278 026C 026C 54 0F ANDF0: ANL A,#0FH
0218 027A 12 02 7D LCALL CONV7SEG
0219 027D F5 22 RET
0220 027F 0272 54 F0 ANDF0: ANL A,#0FH
0221 0281 12 02 70 LCALL CONV7SEG
0222 0284 0279 RET
0223 0286 0279 0279 02 0F PROCIHITUNG: MOV A,FLAGPENUMPANG1
0224 0288 54 0F ANL A,#0FF
0225 028A 02 04 00 01 CJNE A,#0,CEKFLAG1
0226 028C B4 02 02 48 RET
0227 028E 0281 E5 42 CEKFLAG1: MOV A,FLAGPENUMPANG1
0228 028F 0283 54 01 ANL A,#01H
0229 0291 B4 01 03 CJNE A,#01H,CEKFLAG2
0230 0293 0288 12 02 AA LCALL HITUNG_P1
0231 0295 F5 42 CEKFLAG2: MOV A,FLAGPENUMPANG1
0232 0297 028D 54 02 ANL A,#02H
0233 0299 028F B4 02 03 CJNE A,#02H,CEKFLAG3
0234 029B 0292 12 02 99 LCALL HITUNG_P2
0235 029D F5 42 CEKFLAG3: MOV A,FLAGPENUMPANG1
0236 029F 0297 54 04 ANL A,#04H
0237 02A1 0299 B4 04 03 CJNE A,#04H,CEKFLAG4
Perangkat Lunak C - 7

0243 029C 12 02 C8 LCALL HITUNG_P3
0244 029F E5 42 CEKFLAG4: MOV A,FLAGPENUMPANG1
0245 02A1 54 08 ANL A,#08H
0246 02A3 B4 08 03 CJNE A,#08H,CEKFLAGERR
0247 02A6 12 02 D7 LCALL HITUNG_P4
0248 02A9 22 CEKFLAGERR: RET
0249 02AA
0250 02AA
0251 02AA HITUNG_P1:
0252 02AA ;-----BIAYA
0253 02AA E5 31 MOV A,B_PENUMPANG1_2
0254 02AC 24 08 ADD A,#BSATUMETER2
0255 02AE D4 DA A
0256 02AF F5 31 MOV B_PENUMPANG1_2,A
0257 02B0 E5 30 MOVC A,B_PENUMPANG1_1
0258 02B3 34 00 ADDR A,#BSATUMETER1
0259 02B5 D4 DA A
0260 02B6 F5 30 MOV B_PENUMPANG1_1,A
0261 02B8 22 RET
0262 02B9
0263 02B9 HITUNG_P2:
0264 02B9 ;-----BIAYA
0265 02B9 E5 33 MOV A,B_PENUMPANG2_2
0266 02BB 24 08 ADD A,#BSATUMETER2
0267 02BD D4 DA A
0268 02BE F5 33 MOV B_PENUMPANG2_2,A
0269 02C0 E5 32 MOVC A,B_PENUMPANG2_1
0270 02C2 34 00 ADDR A,#BSATUMETER1
0271 02C4 D4 DA A
0272 02C5 F5 32 MOV B_PENUMPANG2_1,A
0273 02C7
0274 02C7 22 RET
0275 02C8 HITUNG_P3:
0276 02C8 ;-----BIAYA
0277 02C8 E5 35 MOV A,B_PENUMPANG3_2
0278 02CA 24 08 ADD A,#BSATUMETER2
0279 02CC D4 DA A
0280 02CD F5 35 MOV B_PENUMPANG3_2,A
0281 02CF E5 34 MOVC A,B_PENUMPANG3_1
0282 02D1 34 00 ADDR A,#BSATUMETER1
0283 02D3 D4 DA A
0284 02D4 F5 34 MOV B_PENUMPANG3_1,A
0285 02D6
0286 02D6 22 RET
0287 02D7 HITUNG_P4:
0288 02D7 ;-----BIAYA
0289 02D7 E5 37 MOV A,B_PENUMPANG4_2
0290 02DA 24 08 ADD A,#BSATUMETER2
0291 02DB D4 DA A
0292 02DC F5 37 MOV B_PENUMPANG4_2,A
0293 02DE E5 36 MOVC A,B_PENUMPANG4_1
0294 02E0 34 00 ADDR A,#BSATUMETER1
0295 02E2 D4 DA A
0296 02E3 F5 36 MOV B_PENUMPANG4_1,A
0297 02E5 22 RET
0298 02E6
0299 02E6
0300 02E6 ;---------------------------------------------------------------------
0301 02E6 SCANNING:
0302 02E6 85 38 80 SCAN1: MOV DATA7SEG,BUFSEG1
0303 02E8 12 02 C0 CLR SLCT7SEG1
0304 02E9 85 39 80 SCAN2: MOV DATA7SEG,BUFSEG2
0305 02EA 85 3A 80 SCAN3: MOV DATA7SEG,BUFSEG3
0306 02EB 85 3B 80 SCAN4: MOV DATA7SEG,BUFSEG4
0307 02ED 12 03 1E LCALL CLEARKONTROL
0308 02EE
0309 02EF 85 3C 80 SCAN5: MOV DATA7SEG,BUFSEG5
0310 02F0 79 4D 80 SCAN6: MOV DATA7SEG,BUFSEG6
0311 02F1 79 5E 80 SCAN7: MOV DATA7SEG,BUFSEG7
Perangkat Lunak C - 8

0312 02FB D2 A2   SETB SLCT7SEG3
0313 02FD D2 A3   SETB SLCT7SEG4
0314 02FF 12 03 1E LCALL CLEARKONTROL
0315 0302
0316 0302 85 3A 80 SCAN3: MOV DATA7SEG,BUFSEG3
0317 0305 D2 A0   SETB SLCT7SEG1
0318 0307 D2 A1   SETB SLCT7SEG2
0319 0309 C2 A2   CLR SLCT7SEG3
0320 030B D2 A3   SETB SLCT7SEG4
0321 030D 12 03 1E LCALL CLEARKONTROL
0322 0310
0323 0310 85 3B 80 SCAN4: MOV DATA7SEG,BUFSEG4
0324 0313 D2 A0   SETB SLCT7SEG1
0325 0315 D2 A1   SETB SLCT7SEG2
0326 0317 D2 A2   SETB SLCT7SEG3
0327 0319 C2 A3   CLR SLCT7SEG4
0328 031B 12 03 1E LCALL CLEARKONTROL
0329 031E
0330 031E
0331 031E 12 03 47 CLEARKONTROL: LCALL DELAYDISPON
0332 0321 D2 A0   SETB SLCT7SEG1
0333 0323 D2 A1   SETB SLCT7SEG2
0334 0325 D2 A2   SETB SLCT7SEG3
0335 0327 D2 A3   SETB SLCT7SEG4
0336 0329 12 03 50 LCALL DELAYDISP
0337 032C 22 RET
0338 032D
0339 032D
0340 032D
0341 032D
0342 032D
0343 032D ;------ ROUTINE KONVERSI ANGKA KE DALAM FORMAT 7 SEGMENT ----
0344 032D CONV7SEG:
0345 032D
0346 032D FE MOV R6,A
0347 032E 90 03 5E MOV DPTR,#SEG
0348 0331 E4 CLR A
0349 0332 BE 00 03 CJNE R6,#00,INCDPTR
0350 0335 02 03 3B LJMP LOAD
0351 0338 A3 INC R6
0352 033B 93 LOAD: MOVC A,@A+DPTR
0353 033C 22 RET
0354 033D
0355 033D
0356 033D PROC_HAPUSSEMUABUFFER:
0357 033D 78 30 MOV R0,#B_PENUMPANG1_1
0358 033F 7C 38 MOV R4,#56
0359 0341 HAPUSSEMUABUFFER:
0360 0341 76 00 MOV @R0,#0
0361 0343 08 INC R0
0362 0344 DC FB DJNZ R4,HAPUSSEMUABUFFER
0363 0346 22 RET
0364 0347
0365 0347 DELAYDISPON:
0366 0347 7E 08 MOV R6,#$08
0367 0349 7F FF DELAY0ON: MOV R7,#$FF
0368 034B DF FE DELAY1ON: DJNZ R7,DELAY1ON
0369 034D DE FA DJNZ R6,DELAY0ON
0370 034F 22 RET
0371 0350
0372 0350 DELAYDISP:
0373 0350 ; MOV R6,#$01
0374 0350 7F 2F DELAY0: MOV R7,#$2F
0375 0352 DF FE DELAY1: DJNZ R7,DELAY1
0376 0354 ; DJNZ R6,DELAY0
0377 0354 22 RET
0378 0355
0379 0355 DELAY:
0380 0355 7E FF MOV R6,#$FF
Perangkat Lunak C - 9

0381  0357 7F FF  DELAYL0:  MOV     R7, #FF
0382  0359 DF FE  DELAYL1:  DJNZ    R7, DELAYL1
0383  035B DE FA  DJNZ    R6, DELAYL0
0384  035D 22    RET
0385  035E    ;-------------------------------------------------------
0386  035E    ;  LOOKUP TABLE
0387  035E    ;-------------------------------------------------------
0388  035E C0F9A4B09992 SEG:    .BYTE
0389  0364 82F88009C0BF
0390  036B    .END
	      tasm: Number of errors = 0
LAMPIRAN D

DATA KOMPONEN
SIDANG TUGAS AKHIR

Perancangan Tarif Biaya Angkot dengan Mikrokontroler AT89C51

Nama : I Wayan Sunarto
NRP : 9622102
Pembimbing TA : Marvin Chandra Wijaya,ST.,MM.,MT
Latar Belakang

- Perselisihan sering terjadi antara penumpang dengan supir angkot yang disebabkan oleh biaya.
- Beda penafsiran tarif biaya

Identifikasi Masalah

Bagaimana merancang & merealisasikan suatu alat pencatat tarif biaya angkot berdasarkan jarak tempuh?
Tujuan

Maksud dan tujuan tugas akhir ini adalah merancang dan merealisasikan suatu alat pencatat tarif biaya angkot berdasarkan jarak tempuh.
Pembatasan Masalah dan Spesifikasi Alat

**Pembatasan Masalah**
1. Input maksimal 15 penumpang
2. Tampilan output biaya maksimal 4 digit
3. Sensor kartu dan sensor jarak bekerja baik

**Spesifikasi Alat**
1. Input 15 penumpang
2. Output 4 digit
3. Sensor optocoupler
Blok Diagram
Gambar 3 Tampilan saat kartu penumpang 1 dimasukkan ke rangkaian naik

Gambar 4 Tampilan saat kartu penumpang 2 dimasukkan ke rangkaian naik
Gambar 5 Tampilan saat kartu penumpang 3 dimasukkan ke rangkaian naik

Gambar 6 Tampilan saat kartu penumpang 4 dimasukkan ke rangkaian naik
Gambar 7 Tampilan saat kartu penumpang 1 dimasukkan ke rangkaian turun dengan jarak tempuh 2 kedipan (1 meter)

Gambar 8 Tampilan saat kartu penumpang 2 dimasukkan ke rangkaian turun dengan jarak tempuh 4 kedipan (2 meter)
Gambar 9 Tampilan saat kartu penumpang 3 dimasukkan ke rangkaian turun
dengan jarak tempuh 6 kedipan (3 meter)

Gambar 10 Tampilan saat kartu penumpang 4 dimasukkan ke rangkaian turun
dengan jarak tempuh 8 kedipan (4 meter)
Rangkaian Mikrokontroler
Rangkaian Naik
Rangkaian Turun
Rangkaian Scanning
Rangkaian Seven Segment
<table>
<thead>
<tr>
<th>Mikrokontroler</th>
<th>Seven Segment (Common Anoda)</th>
</tr>
</thead>
<tbody>
<tr>
<td>P0.0</td>
<td>PIN A</td>
</tr>
<tr>
<td>P0.1</td>
<td>PIN B</td>
</tr>
<tr>
<td>P0.2</td>
<td>PIN C</td>
</tr>
<tr>
<td>P0.3</td>
<td>PIN D</td>
</tr>
<tr>
<td>P0.4</td>
<td>PIN E</td>
</tr>
<tr>
<td>P0.5</td>
<td>PIN F</td>
</tr>
<tr>
<td>P0.6</td>
<td>PIN G</td>
</tr>
<tr>
<td>P0.7</td>
<td>PIN PT</td>
</tr>
</tbody>
</table>

![Seven Segment Diagram](image)
Daftar Heksadecimal Tampilan Seven Segment

<table>
<thead>
<tr>
<th>TAMPILAN ANGKA</th>
<th>P0.7</th>
<th>P0.6</th>
<th>P0.5</th>
<th>P0.4</th>
<th>P0.3</th>
<th>P0.2</th>
<th>P0.1</th>
<th>P0.0</th>
<th>HEKSA DESIMAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>C0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>F9</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>A4</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>B0</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>99</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>92</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>82</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>F8</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>80</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>90</td>
</tr>
</tbody>
</table>
Denah Tempat Duduk
Kartu Penumpang dan Kartu Piringan
Konversi Bilangan Kartu Penumpang

<table>
<thead>
<tr>
<th>Desimal</th>
<th>Heksadesimal</th>
<th>Biner</th>
<th>Kartu Penumpang</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>00</td>
<td>0000 0000</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>01</td>
<td>0000 0001</td>
<td>Kartu Penumpang 1</td>
</tr>
<tr>
<td>2</td>
<td>02</td>
<td>0000 0010</td>
<td>Kartu Penumpang 2</td>
</tr>
<tr>
<td>3</td>
<td>03</td>
<td>0000 0011</td>
<td>Kartu Penumpang 3</td>
</tr>
<tr>
<td>4</td>
<td>04</td>
<td>0000 0100</td>
<td>Kartu Penumpang 4</td>
</tr>
<tr>
<td>5</td>
<td>05</td>
<td>0000 0101</td>
<td>Kartu Penumpang 5</td>
</tr>
<tr>
<td>6</td>
<td>06</td>
<td>0000 0110</td>
<td>Kartu Penumpang 6</td>
</tr>
<tr>
<td>7</td>
<td>07</td>
<td>0000 0111</td>
<td>Kartu Penumpang 7</td>
</tr>
<tr>
<td>8</td>
<td>08</td>
<td>0000 1000</td>
<td>Kartu Penumpang 8</td>
</tr>
<tr>
<td>9</td>
<td>09</td>
<td>0000 1001</td>
<td>Kartu Penumpang 9</td>
</tr>
<tr>
<td>10</td>
<td>0A</td>
<td>0000 1010</td>
<td>Kartu Penumpang 10</td>
</tr>
<tr>
<td>11</td>
<td>0B</td>
<td>0000 1011</td>
<td>Kartu Penumpang 11</td>
</tr>
<tr>
<td>12</td>
<td>0C</td>
<td>0000 1100</td>
<td>Kartu Penumpang 12</td>
</tr>
<tr>
<td>13</td>
<td>0D</td>
<td>0000 1101</td>
<td>Kartu Penumpang 13</td>
</tr>
<tr>
<td>14</td>
<td>0E</td>
<td>0000 1110</td>
<td>Kartu Penumpang 14</td>
</tr>
<tr>
<td>15</td>
<td>0F</td>
<td>0000 1111</td>
<td>Kartu Penumpang 15</td>
</tr>
</tbody>
</table>
Diagram Alir Utama
Diagram Alir Cek Kartu Naik
Diagram Alir Cek Kartu Turun
Diagram Alir Cek Kartu Piringan dan Tampilkan Biaya
Diagram Alir Scanning Tampilan
Cara Kerja

- Sensor naik dan turun berfungsi untuk mendeteksi kartu penumpang
- Sensor jarak berfungsi untuk menghitung jarak tempuh penumpang
- Rangkaian scanning berfungsi untuk menghidupkan seven segment secara bergantian
- Seven segment berfungsi sebagai penampil output
Uji Coba Alat

Pengujian Rangkaian Naik, Jarak dan Turun

Pengujian Terhadap Seven Segment

<table>
<thead>
<tr>
<th>Kartu</th>
<th>Rangkaian Naik</th>
<th>Rangkaian Jarak</th>
<th>Rangkaian Turun</th>
</tr>
</thead>
<tbody>
<tr>
<td>No</td>
<td>Sinar</td>
<td>Tamp. 7 Segment</td>
<td>Putaran</td>
</tr>
<tr>
<td>1</td>
<td>0001</td>
<td>1</td>
<td>2(2 kedp)</td>
</tr>
<tr>
<td>2</td>
<td>0010</td>
<td>2</td>
<td>4(4 Kedp)</td>
</tr>
<tr>
<td>3</td>
<td>0011</td>
<td>3</td>
<td>6(6 kedp)</td>
</tr>
<tr>
<td>4</td>
<td>0100</td>
<td>4</td>
<td>8(8 kedp)</td>
</tr>
</tbody>
</table>

Ket: 2 putaran = 1 meter
1 meter = Rp 8

Tampilan Seven Segment

<table>
<thead>
<tr>
<th>Sensor</th>
<th>Seven Segment</th>
<th>Ket</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sensor naik</td>
<td>Tamp. flag penumpang</td>
<td>kartu naik di masukkan pada sensor naik</td>
</tr>
<tr>
<td>Sensor jarak</td>
<td>Tamp. kedp</td>
<td>Setiap satu putaran</td>
</tr>
<tr>
<td>Sensor turun</td>
<td>Tamp. biara</td>
<td>Untuk mengetahui kedalaman kartu</td>
</tr>
</tbody>
</table>
Regulator
Alat Pencatat Tarif Biaya Angkot
Tampilan saat kartu penumpang 1 dimasukan ke rangkaian naik
Tampilan saat kartu penumpang 2 dimasukan ke rangkaian naik
Tampilan saat kartu penumpang 3 dimasukan ke rangkaian naik
Tampilan saat kartu penumpang 4 dimasukan ke rangkaian naik
Tampilan saat kartu penumpang 1 dimasukan ke rangkaian turun dengan jarak tempuh 2 kedipan (1meter)
Tampilan saat kartu penumpang 2 dimasukan ke rangkaian turun dengan jarak tempuh 4 kedipan (2meter)
Tampilan saat kartu penumpang 3 dimasukan ke rangkaian turun dengan jarak tempuh 6 kedipan (3meter)
Tampilan saat kartu penumpang 4 dimasukan ke rangkaian turun dengan jarak tempuh 8 kedipan (4meter)
Kesimpulan dan Saran

Kesimpulan
- Hasil pengujian menunjukkan alat dapat digunakan sesuai dengan yang diharapkan.
- Penggunaan optocoupler masih mengalami kesulitan dalam mendeteksi kartu penumpang

Saran
- Untuk pengembangan lebih lanjut disarankan penggunaan optocoupler diganti dengan jenis yang lebih khusus untuk mempermudah dalam mendeteksi kartu