# SIGNETICS
# 2650
# MICROPROCESSOR

# CONTENTS

# CHAPTER I

# INTRODUCTION

# INTRODUCING THE 2650 FAMILY

*"5—VOLT SYSTEM REDUCES SYSTEM COSTS"*
*"2650 PUTS THE INTERFACE ON THE CHIP...NOT ON THE CIRCUIT BOARD"*
*"POWERFUL INSTRUCTION SET PROVIDES LOWER COST SYSTEMS"*

The greatly increased sophistication and rising production costs of today's logic systems force the system designer to use every available resource in order to economically produce his system. In keeping with this cost reduction goal, Signetics has developed a powerful general purpose integrated microprocessor called the 2650. The first Signetics microprocessor, in conjunction with Signetics MOS and Bipolar memory and interface product lines, offers the system designer a viable and attractive alternative to the hard-wired approach to system design. For many applications, the system designer can use this general purpose microprocessor and standard memory and interface circuits to implement systems with lower cost than the hard-wired logic approach without sacrificing performance.

By using the 2650 and compatible products, the system designer can obtain two other major benefits of microcomputer systems. These benefits are greatly enhanced system flexibility and minimized design or modification cycles compared with the hard-wired logic approach.

The requirements of the majority of applications for integrated microprocessors (logic replacement and control functions) have defined a general set of processor parameters based on system and device economies, ease of use, and speed requirements.

These characteristics include:

- Single chip
- Fixed instruction set
- Eight bit parallel structure
- TTL compatibility

In addition to these characteristics, the design of the 2650 has been optimized around three generalized objectives:

- Lowest system cost
- Ease of use
- Capable of a wide range of applications

The optimum technology choice for implementing these features is the low threshold ion-implanted N-Channel silicon gate process. This process has matured in the past few years, providing a combination of high density, low threshold voltage, moderate speed and good manufacturing yields. Using this technology, a total of 576 bits of ROM, approximately 250 bits of register and about 900 logic gates are used to implement the processor function on the 2650 chip.

The instruction set consists of 75 instructions, of which about 40% consists of arithmetic instructions. This class contains the Boolean, arithmetic, and compare operations, each of which may be executed using any one of eight addressing modes. Another 30% of the instruction set consists of branch instructions which incorporate six addressing modes. The remaining 30% of the instruction set includes, among others, I/O instructions, instructions for performing operations on the two status registers, a decimal adjust instruction and the HALT instruction.

Utilizing multiple addressing modes greatly increases coding efficiency, allowing functions to be performed using fewer instructions than less powerful machines. The resulting reduction in routine execution time and memory capacity requirements directly translates into improved system performance and reduced memory cost. In this way the powerful instruction set and addressing modes of the 2650 allow a significant reduction in the memory required to perform a given function, resulting in sizeable system cost savings without sacrificing performance.

# FEATURES OF THE 2650 FAMILY

## 2650 FAMILY APPROACH

* Low System Cost
  - Low cost N-Channel products
  - Intrinsic advantages of single 5V supply
  - Uses standard low cost memories
  - Low cost interfacing
* Ease of Use
  - Easy interfacing
  - Conventional instruction set
  - Ease of programming
* Wide Range of Applications
  - General purpose capability
  - Powerful architecture
  - Powerful instruction set
  - Flexible
  - Expanding family of devices

## FEATURES OF THE MICROPROCESSOR

### Basic 2650 Processor Characteristics

* Single chip 8-bit processor
* Signetics low threshold double ion-implanted silicon gate N-Channel technology
* Single +5V power supply
* Low power consumption: 525 mW maximum
* Single phase TTL-compatible clock
* Static operation: no minimum clock frequency
* Clock frequency: 1.25MHz maximum
* Cycle time: 2.4µs minimum
* Standard 40 pin DIP

### 2650 Interfaces

* TTL compatible inputs, outputs — no external resistors required
* Tri-state bus outputs for multiprocessor and direct memory access systems
* Asynchronous (handshaking) memory and I/O interface
* Accepts wide range of memory timing
* Interfaces directly with industry standard memories
* Powerful control interface
* Single-bit direct serial I/O path
* Parallel 8-bit I/O capability

### 2650 Processor Architecture

* 8-bit bidirectional tri-state data bus
* Separate tri-state address bus
* 32,768-byte addressing range
* Internal 8-bit parallel structure
* Seven 8-bit addressable general purpose registers
* Eight-level on-chip subroutine return address stack
* Program status word for flexibility and enhanced processing power
* Single-level hardware vectored interrupt capability
* Interrupt service routines may be located anywhere in addressable memory
* Separate adder for fast address calculation

### 2650 Instruction Set

* General purpose instruction set with substantial capabilities in arithmetic, character manipulation and control and I/O processing
* Fixed instruction set
* 75 instructions
* Up to eight addressing modes
* True indexing with optional auto increment/decrement
* One, two or three byte instructions
* One- and two-byte I/O instructions
* Selective test of individual bits
* Powerful instruction set and addressing modes minimize memory requirements

4

## FEATURES OF COMPATIBLE PRODUCTS

### 2602, 2606, 1K RAMs

* Completely static operation
* N-Channel silicon gate technology
* 1024 X 1 organization (2602)
  256 X 4 organization (2606)
* Single +5V power supply
* 200mW typical power dissipation
* Maximum access time:
  1$\mu$s    : 2602
  750ns  : 2606
  650ns  : 2602-2
  500ns  : 2602-1, 2606-1
* TTL-compatible
* Tri-state outputs
* Data I/O bus (2606 only)
* Standard 16 pin DIP

### 2608 8K ROM

* Completely static operation
* N-Channel silicon gate technology
* 1024 X 8 organization
* Single +5V power supply
* 400mW maximum power dissipation
* 650ns maximum access time
* TTL compatible
* Tri-state outputs
* Standard 24 pin DIP

### 8T26 Quad Transceiver

* Schottky TTL technology
* Four pairs of bus drivers/receivers
* Separate drive and receive enable lines
* Tri-state outputs
* Low current pnp inputs
* High fan out — driver sinks 40mA
* 20ns maximum propagation delay
* Standard 16 pin DIP

### 8T31 8-bit Bidirectional Port

* Schottky TTL technology
* Two independent bidirectional busses
* Eight bit latch register
* Independent read, write controls for each bus
* Bus A overrides if a write conflict occurs
* Register can be addressed as a memory location via Bus B Master Enable
* 30ns maximum propagation delay
* Low input current: 500$\mu$A
* High fan out — sinks 20mA
* Standard 24 pin DIP

### 8T95/6/7/8 Hex Buffers/Inverters

* Schottky TTL technology
* Six buffers or inverters per package
* Non-inverting (8T95, 8T97) or Inverting (8T96, 8T98)
* Buffered control lines
* Tri-state outputs
* Low current pnp inputs
* Standard 16 pin DIP

### 82S115/123/129 PROMs

* Schottky TTL technology
* Single +5V power supply
* 32 X 8 organization (82S123)
* 256 X 4 organization (82S129)
* 512 X 8 organization (82S115)
* Field programmable (Nichrome)
* On-chip storage latches (82S115 only)
* Low current pnp inputs
* Tri-state outputs
* 35ns typical access time
* Standard 24 pin DIP (82S115)
* Standard 16 pin DIP (82S123, 82S129)

(See Appendix for additional products and data sheets.)

# PROCESSOR HARDWARE DESCRIPTION

## ARCHITECTURE

### GENERAL DESCRIPTION

A block diagram of the processor is shown in Figure 1. The first, second, and third bytes of instructions are read into the processor on the data bus and loaded into the Instruction Register, Holding Register, and Data Bus Register, respectively. The instructions are decoded through a combination of ROM and random logic.

The ALU performs arithmetic, Boolean, and combinatorial shifting functions. It operates on eight bits in parallel and utilizes carry-look-ahead logic. A second adder is used to increment the instruction address register and to calculate operand addresses for the indexed and relative addressing modes. This separate address adder allows complex addressing modes to be implemented with no increase in instruction execution time.

The General Purpose Register Stack and the Subroutine Return Address Stack are implemented with static RAM cells. The Register Stack consists of seven 8-bit registers. The Subroutine Stack can contain eight 15-bit addresses, thereby allowing eight levels of subroutine nesting. Placing the Subroutine Stack on the chip allows efficient ROM-only systems to be implemented in some applications. Separate 15-bit Instruction Address and Operand Address Registers and provided. The 2650 is an 8-bit binary processor with BCD capability. See Figure 2 for a diagram of the 2650 registers as seen by the programmer.

### PROGRAM STATUS WORD

The Program Status Word (PSW) is a major feature of the 2650 with greatly increases its flexibility and processing power. The PSW is a special purpose register within the processor that contains status and control bits.

It is divided into two bytes called the Program Status Upper (PSU) and Program Status Lower (PSL). The PSW bits may be tested, loaded, stored, preset, or cleared using the instructions which affect the PSW. The bits are utilized as follows:

| PSU0, 1,2 | — SP | — Pointer for the Return Address Stack. |
| PSU5 | — II | — Used to Inhibit recognition of additional Interrupts. |
| PSU6 | — F | — Flag is a latch directly driving the flag output. |
| PSU7 | — S | — Sense equals the state of the sense input. |
| PSL0 | — C | — Carry stores any carry from the high-order bit of the ALU. |
| PSL1 | — COM | — Compare determines if a logical or arithmetic comparison is to be made. |
| PSL2 | — OVF | — Overflow is set if a two's complement overflow occurs. |
| PSL3 | — WC | — With Carry determines is the carry is used in arithmetic and rotate instructions. |
| PSL4 | — RS | — Register Select identifies which bank of 3 GP registers is being used. |
| PSL5 | — IDC | — Inter Digit Carry stores the bit-3-to-bit-4 carry in arithmetic operations. |
| PSL6, 7 | — CC | — Condition Code is affected by compare, test and arithmetic instructions. |

### INTERRUPT HANDLING CAPABILITY

The 2650 has a single level hardware vectored interrupt capability. When an interrupt occurs, the 2650 finishes the current instruction and sets the

Interrupt Inhibit bit in the PSW. The processor then executes a Branch to Subroutine Relative to location Zero (ZBSR) instruction and sends out Interrupt Acknowledge and Operation Request signals. On receipt of the INTACK signal the interrupting device inputs an 8-bit address, the interrupt vector, on the data bus. The relative and relative indirect addressing modes combined with this 8-bit address allow interrupt service routines to begin at any addressable memory location.
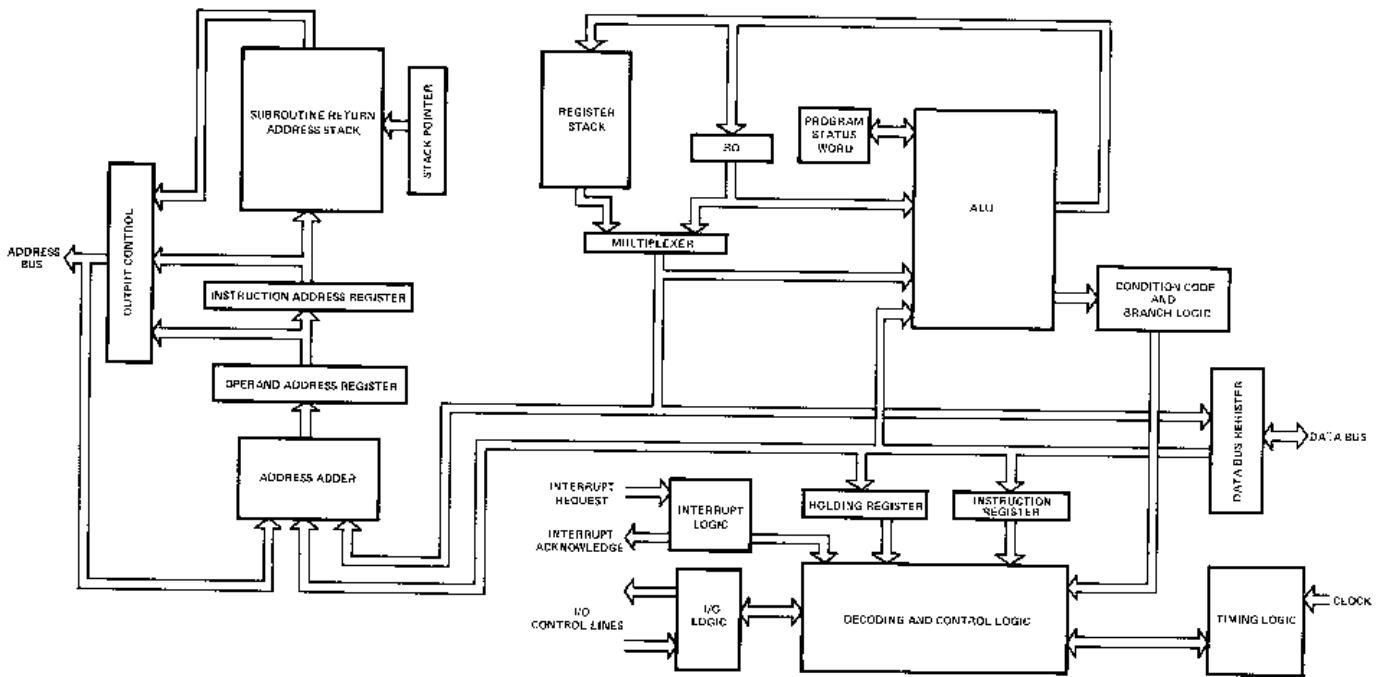


Figure 1. BLOCK DIAGRAM
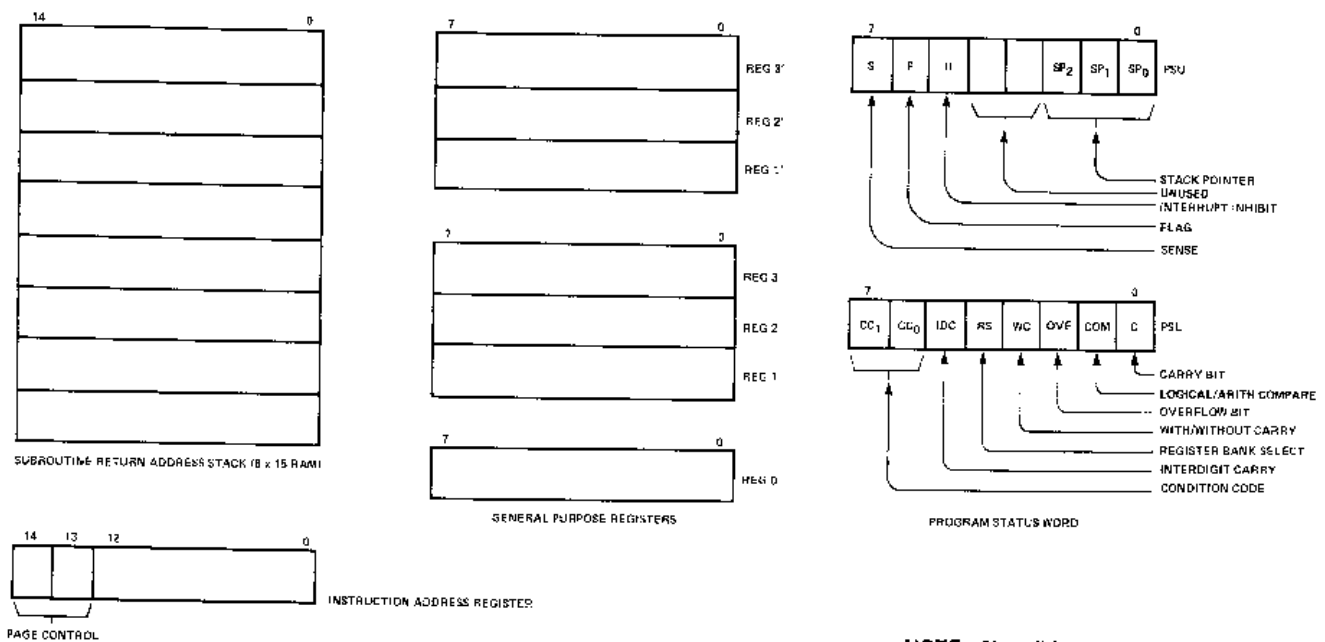


NOTE: Not all internal registers are shown.

Figure 2. MAJOR 2650 REGISTERS

## INTERFACING

### INTRODUCTION TO INTERFACING WITH THE 2650

Five key concepts have been incorporated in the 2650 to make interfacing easy and inexpensive. The extent to which these concepts have been incorporated in the Signetics 2650 provides unique benefits of system density and low cost to the system designer.

### 1. SINGLE 5V POWER SUPPLY

Low threshold double ion-implanted Silicon Gate N-Channel MOS technology is used to allow operation from one +5V power supply with resultant cost savings and improved reliability. This reduces power consumption significantly compared with the multi-power supply approach.

### 2. INTERFACE CIRCUIT COMPATIBILITY

The 2650 inputs and outputs are specified to be compatible with widely available, standard, low cost logic families such as TTL, CMOS and Low-power STTL. This includes the single phase clock input which saves the cost of high level multiphase clock driver circuitry. Bus outputs are tri-state and capable of driving one 7400 TTL load or four 74LS loads. The 2650 is capable of driving several loads of pnp-buffered STTL inputs. Many MSI, Interface and Memory LSI circuits (for example, in Signetics 82S00 and 8T00 series) have these low current pnp inputs and are recommended for use in 2650 microcomputer systems. See Table 1 for DC characteristics of the 2650.

### 3. USE OF STANDARD MEMORIES

One of the major 2650 design achievements is to operate efficiently in a system using industry standard memories, for example 1024 X 1 and 256 X 4 N-channel RAMs and 1024 X 8 N-Channel ROMs. These standard memories are widely available and used in volume with corresponding low cost. Non-standard memories, particularly those produced by only one manufacturer will be less available, run in lower volume and often cost 2 to 3 times as much per bit as industry standard products. The 2650 operates successfully with memories of any access time, due to the completely asynchronous interface that is provided for this purpose. Memories which respond in less than 0.8 microseconds allow the processor to operate at maximum speed.

### 4. NO SPECIAL INTERFACE PRODUCTS

Similarly, another major achievement is to operate efficiently in a system using no special I/O products. This approach avoids the problems of a system requiring high cost specialized components with restricted availability.

### TABLE 1. PRELIMINARY 2650 DC ELECTRICAL CHARACTERISTICS

| SYMBOL | PARAMETER | TEST CONDITIONS | LIMITS MIN | LIMITS MAX | UNIT |
|--------|-----------|-----------------|-----|-----|------|
| $I_{LI}$ | Input Load Current | $V_{IN}$ = 0 to 5.25V | | 10 | $\mu$A |
| $I_{LOH}$ | Output Leakage Current | ADREN, DBUSEN = 2.2V, $V_{OUT}$ = 4V | | 10 | $\mu$A |
| $I_{LOL}$ | Output Leakage Current | ADREN, DBUSEN = 2.2V, $V_{OUT}$ = 0.45V | | 10 | $\mu$A |
| $I_{CC}$ | Power Supply Current | $V_{CC}$ = 5.25V, $T_A$ = 0°C | | 100 | mA |
| $V_{IL}$ | Input Low | | −0.6 | 0.8 | V |
| $V_{IH}$ | Input High | | 2.2 | $V_{CC}$ | V |
| $V_{OL}$ | Output Low | $I_{OL}$ = 1.6 mA | 0.0 | 0.45 | V |
| $V_{OH}$ | Output High | $I_{OH}$ = −100 $\mu$A | 2.4 | $V_{CC}$-0.5 | V |
| $C_{IN}$ | Input Capacitance | $V_{IN}$ = 0V | | 10 | pF |
| $C_{OUT}$ | Output Capacitance | $V_{OUT}$ = 0V | | 10 | pF |

Conditions: $T_A$ = 0°C to 70°C, $V_{CC}$ = 5V ±5%

## 5. POWERFUL MEMORY AND I/O INTERFACE

The following features characterize the memory and I/O interfaces:

- Both memory and input/output may operate in a completely asynchronous fashion. Consequently, devices operating at any speed up to the maximum data transfer rate may be connected without buffering. External latching of data from these interfaces is not required.
- Data paths are driven with tri-state buffers, allowing multiprocessor and Direct Memory Access (DMA) configurations to be designed.
- Eight-bit data paths communicate data in parallel.
- One- and two-byte I/O instructions provide maximum flexibility and efficiency when interfacing with I/O devices.

```
SENSE    →  1          40  →  FLAG
ADR12    ←  2          39  —○ VCC
ADR11    ←  3          38  ←  CLOCK
ADR10    ←  4          37  ←  PAUSE
ADR9     ←  5          36  ←  OPACK
ADR8     ←  6          35  ←  RUN/WAIT
ADR7     ←  7          34  →  INTACK
ADR6     ←  8          33  ↔  DBUS0
ADR5     ←  9   2650   32  ↔  DBUS1
ADR4     ←  10         31  ↔  DBUS2
ADR3     ←  11         30  ↔  DBUS3
ADR2     ←  12         29  ↔  DBUS4
ADR1     ←  13         28  ↔  DBUS5
ADR0     ←  14         27  ↔  DBUS6
ADREN    →  15         26  ↔  DBUS7
RESET    →  16         25  ←  DBUSEN
INTREQ   →  17         24  →  OPREQ
ADR14-D/C ← 18         23  →  R/W
ADR13-E/NE ← 19        22  →  WRP
M/IO     ←  20         21  —○ GND
```
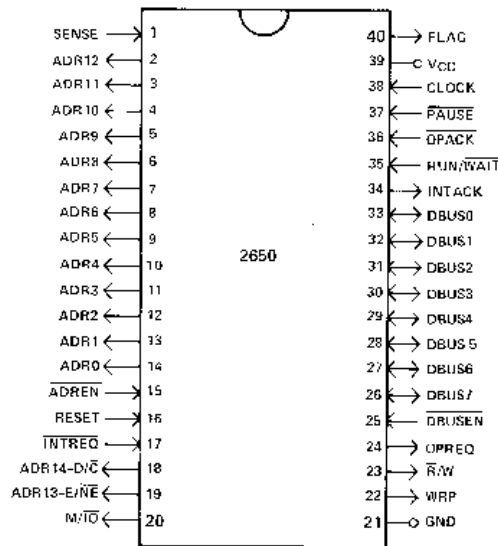
**Figure 3. PIN CONFIGURATION**

### PIN CONFIGURATION AND INTERFACE SIGNAL DEFINITION

Refer to Figure 3 for the 2650 pin configuration. Signals are defined as follows:

ADR0-ADR12 — The low order 13 bits of address for memory access are on these pins. ADR0-ADR7 are also used in two-byte I/O instructions. These outputs are tri-state buffers controlled by ADREN.

ADR13-E/NE — This multiplexed output signal delivers the ADR13 address bit when M/IO is in the M phase or discriminates between Extended and Non-Extended I/O instructions when M/IO is in the I/O phase.

ADR14-D/C — Address 14 or Data/Control is a multiplexed output signal. This pin delivers the ADR14 address bit when M/IO is in the M phase or discriminates between Data and Control I/O instructions when M/IO is in the I/O phase.

ADREN — Address Bus Enable is an input providing the external control for the ADR0-ADR12 tri-state buffer drivers.

DBUS0-DBUS7 — This is the 8-bit, bidirectional tri-state bus over which most data is communicated into or out of the processor.

DBUSEN — Data Bus Enable is an input that controls the tri-state buffer drivers for DBUS0 to DBUS7.

OPREQ — Operation Request is an output signal that informs external devices that the information on other output pins is valid.

9

| | |
|---|---|
| OPACK | — Operation Acknowledge is an input which is used by external devices to end an I/O or memory signaling sequence. |
| M/IO | — Memory/Input-Output. This output informs external devices whether Memory or Input/Output functions are being performed. |
| R/W | — This output signal describes an I/O or memory operation as Read or Write, and defines whether the bidirectional DBUS is transmitting or receiving. |
| WRP | — This Write Pulse is generated during write sequences and may be used to strobe memory or I/O devices. |
| SENSE | —· Is an input, independent of the other I/O signals, that provides a direct input to the processor. |
| FLAG | — This pin provides a direct output signal that is completely independent of the other I/O signals. |
| INTREQ | — Interrupt Request. This input is used by external devices to force the processor into the Interrupt sequence. |
| INTACK | — Interrupt Acknowledge is the signal used by the processor to inform external devices that it has entered an interrupt sequence. |
| PAUSE | — Pause is used to temporarily stop the processor at the end of the current instruction. It may stop processing for an indefinite length of time and is available to use for DMA (Direct Memory Access). |
| RUN/WAIT | —· Informs external circuits as to the Run/Wait status of the 2650 processor. |
| RESET | — Is an input used to cause the 2650 to begin processing from a known state. |
| CLOCK | — This is the only clock input to the processor. It accepts standard TTL levels. |
| VCC | — +5V power. |
| GND | — The logic and power supply ground for the processor. |

### 2650 TIMING

The clock input to the 2650 provides the basic timing information that the processor uses for all its internal and external operations. The clock rate determines the instruction execution time, except to the extent that external memories and devices slow the processor down. The maximum clock rate of the standard 2650 is 1.25 Megacycles (one clock period is 800ns minimum). One unique feature of the 2650 is that the clock frequency may be slowed down to DC, allowing complete timing flexibility for interfacing. This feature permits single stepping the clock which can greatly simplify system checkout. It also provides an easy method to halt the processor. Each 2650 cycle is comprised of three clock periods. Direct instructions require either 2, 3, or 4 processor cycles for execution and, therefore, vary from 4.8 to 9.6$\mu$s in duration.

A timing diagram for a memory read cycle is shown in Figure 4. OPREQ (Operation Request) is the master control signal that coordinates all operations external to the processor. When true, OPREQ indicates that other output signals are valid. During a memory read cycle M/IO is in the M (Memory) state and R/W is in the R (Read) state. The address lines and the control lines become valid before OPREQ rises. The data to be read may be returned anytime after OPREQ becomes valid. An OPACK (Operation Acknowledge) should accompany the read data from the memory. The Data and OPACK signals should remain valid for 50 ns after OPREQ falls.

## INPUT/OUTPUT INTERFACE

The 2650 microprocessor has a set of versatile I/O instructions and can perform I/O operations in a variety of ways. One- and two-byte I/O instructions are provided, as well as a special single-bit I/O facility. The I/O modes provided by the 2650 are designated as Data, Control, and Extended I/O.

Data or Control I/O instructions are one byte long. Any general purpose register can be used as the source or destination. A special control line indicates if either a Data or Control instruction is being executed. Extended I/O is a two-byte read or write instruction. Execution of an extended I/O instruction will cause an 8-bit address, taken from the second byte of the instruction, to be placed on the low order eight address lines. The data, which can originate or terminate with any general purpose register, is placed on the data bus. This type of I/O can be used to simultaneously select a device and send data to it.

Memory reference instructions that address data outside of physical memory may also be used for I/O operations. When an instruction is executed, the address may be decoded by the I/O device rather than memory.

## MEMORY INTERFACE

The memory interface consists of the address bus, the 8-bit data bus and several signals that operate in an interlocked or handshaking mode.

The Write Pulse signal is designed to be used as a memory strobe signal for any memory type. It has been particularly optimized to be used as the Chip Enable or Read/Write signal for the Signetics 2602 and 2606 RAMs.

## INTERFACING – A MINIMAL SYSTEM EXAMPLE

The 2650 has been designed for low cost, easy interfacing, which is dramatically illustrated by a minimal system configuration shown in Figure 5. This system has a Teletype interface, 1024 bytes of ROM, and 256 bytes of RAM, yet requires only seven (7) standard integrated circuit packages. The ROM can contain a bootstrap loader and I/O driver programs for the Teletype. Other programs could reside in ROM or be read into RAM via the Teletype. An alternative to the 2608 N-Channel MOS ROM is the 82S115 Bipolar PROM which offers a 512 X 8 organization. Only one +5-volt power supply is required for this system. The advantages of conceptual simplicity and minimum system costs of the 2650 approach will be obvious to the system designer, particularly when compared to alternative microprocessor products.
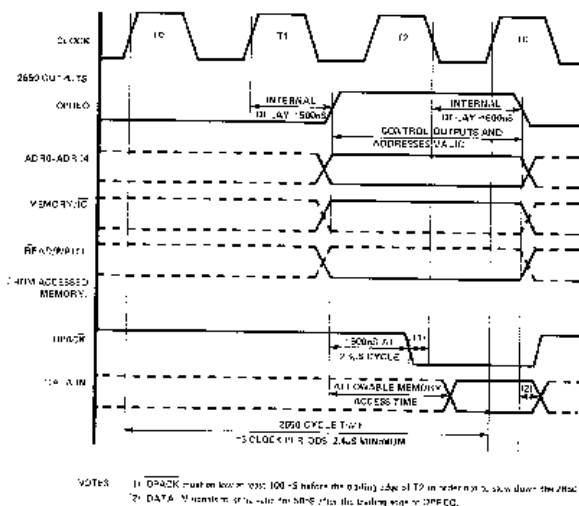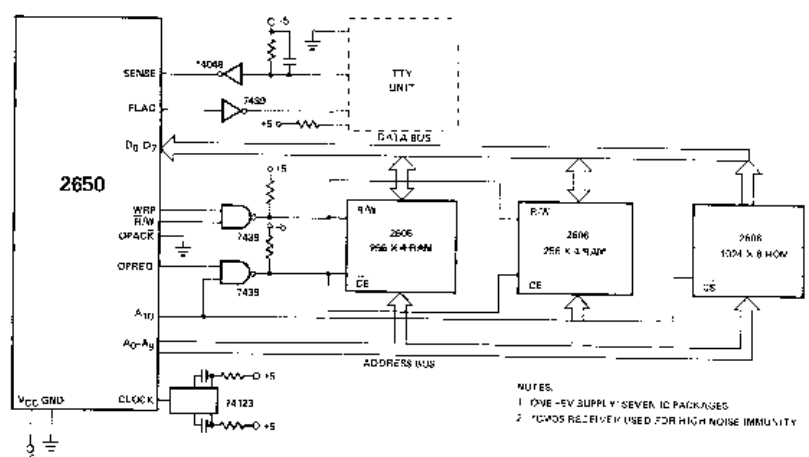


**Figure 4. MEMORY READ CYCLE TIMING**



**Figure 5. SEVEN PACKAGE MINIMAL SYSTEM**

11

# INSTRUCTION SET

It may be seen from examination of the 2650 instruction set that there are many powerful instructions which are all easily understood and are typical of larger computers. There are one-, two-, and three-byte instructions as a result of the multiplicity of addressing modes. See Table 2 for a complete listing and Figure 6 for instruction formats.

Automatic incrementing or decrementing of an index register is available in the arithmetic indexed instructions. All of the branch instructions except indexed branching can be conditional.

Register-to-register instructions are one byte; register-to-storage instructions are two or three bytes long. The two-byte register-to-memory instructions are either immediate or relative addressing types.

## TABLE 2. INSTRUCTION SET

| | MNEMONIC | | OP CODE | FORMAT* | DESCRIPTION OF OPERATION | AFFECTS | CYCLES |
|---|---|---|---|---|---|---|---|
| **LOAD/STORE** | LOD | Z | 000 000 | 1Z | Load Register Zero | CC (Note 1) | 2 |
| | | I | 000 001 | 2I | Load Immediate | CC (Note 1) | 2 |
| | | R | 000 010 | 2R | Load Relative | CC (Note 1) | 3 |
| | | A | 000 011 | 3A | Load Absolute | CC (Note 1) | 4 |
| | STR | Z | 110 000 | 1Z | Store Register Zero (r≠0) | CC (Note 1) | 2 |
| | | R | 110 010 | 2R | Store Relative | — | 3 |
| | | A | 110 011 | 3A | Store Absolute | — | 4 |
| **ARITHMETIC** | ADD | Z | 100 000 | 1Z | Add to Register Zero w/wo Carry | C, CC (Note 1), IDC, OVF | 2 |
| | | I | 100 001 | 2I | Add Immediate w/wo Carry | C, CC (Note 1), IDC, OVF | 2 |
| | | R | 100 010 | 2R | Add Relative w/wo Carry | C, CC (Note 1), IDC, OVF | 3 |
| | | A | 100 011 | 3A | Add Absolute w/wo Carry | C, CC (Note 1), IDC, OVF | 4 |
| | SUB | Z | 101 000 | 1Z | Subtract from Register Zero w/wo Borrow | C, CC (Note 1), IDC, OVF | 2 |
| | | I | 101 001 | 2I | Subtract Immediate w/wo Borrow | C, CC (Note 1), IDC, OVF | 2 |
| | | R | 101 010 | 2R | Subtract Relative w/wo Borrow | C, CC (Note 1), IDC, OVF | 3 |
| | | A | 101 011 | 3A | Subtract Absolute w/wo Borrow | C, CC (Note 1), IDC, OVF | 4 |
| | DAR | | 100 101 | 1Z | Decimal Adjust Register | CC (Note 2) | 3 |
| **LOGICAL** | AND | Z | 010 000 | 1Z | AND to Register Zero (r≠0) | CC (Note 1) | 2 |
| | | I | 010 001 | 2I | AND Immediate | CC (Note 1) | 2 |
| | | R | 010 010 | 2R | AND Relative | CC (Note 1) | 3 |
| | | A | 010 011 | 3A | AND Absolute | CC (Note 1) | 4 |
| | IOR | Z | 011 000 | 1Z | Inclusive OR to Register Zero | CC (Note 1) | 2 |
| | | I | 011 001 | 2I | Inclusive OR Immediate | CC (Note 1) | 2 |
| | | R | 011 010 | 2R | Inclusive OR Relative | CC (Note 1) | 3 |
| | | A | 011 011 | 3A | Inclusive OR Absolute | CC (Note 1) | 4 |
| | EOR | Z | 001 000 | 1Z | Exclusive OR to Register Zero | CC (Note 1) | 2 |
| | | I | 001 001 | 2I | Exclusive OR Immediate | CC (Note 1) | 2 |
| | | R | 001 010 | 2R | Exclusive OR Relative | CC (Note 1) | 3 |
| | | A | 001 011 | 3A | Exclusive OR Absolute | CC (Note 1) | 4 |
| **COMPARE** | COM | Z | 111 000 | 1Z | Compare to Register Zero Arithmetic/Logical | CC (Note 3) | 2 |
| | | I | 111 001 | 2I | Compare Immediate Arithmetic/Logical | CC (Note 4) | 2 |
| | | R | 111 010 | 2R | Compare Relative Arithmetic/Logical | CC (Note 4) | 3 |
| | | A | 111 011 | 3A | Compare Absolute Arithmetic/Logical | CC (Note 4) | 4 |
| **ROTATE** | RRR | | 010 100 | 1Z | Rotate Register Right w/wo Carry | C, CC, IDC, OVF | 2 |
| | RRL | | 110 100 | 1Z | Rotate Register Left w/wo Carry | C, CC, IDC, OVF | 2 |
| **BRANCH** | BCT | R | 000 110 | 2R | Branch On Condition True Relative | — | 3 |
| | | A | 000 111 | 3B | Branch On Condition True Absolute | — | 3 |
| | BCF | R | 100 110 | 2R | Branch On Condition False Relative | — | 3 |
| | | A | 100 111 | 3B | Branch On Condition False Absolute | — | 3 |
| | BRN | R | 010 110 | 2R | Branch On Register Non-Zero Relative | — | 3 |
| | | A | 010 111 | 3B | Branch On Register Non-Zero Absolute | — | 3 |
| | BIR | R | 110 110 | 2R | Branch On Incrementing Register Relative | — | 3 |
| | | A | 110 111 | 3B | Branch On Incrementing Register Absolute | — | 3 |
| | BDR | R | 111 110 | 2R | Branch On Decrementing Register Relative | — | 3 |
| | | A | 111 111 | 3B | Branch On Decrementing Register Absolute | — | 3 |
| | ZBRR | | 100 110 11 | 2ER | Zero Branch Relative, Unconditional | — | 3 |
| | BXA | | 100 111 11 | 3EB | Branch Indexed Absolute, Unconditional (Note 5) | — | 3 |

# TABLE 2. INSTRUCTION SET (CONTINUED)

| | MNEMONIC | | OP CODE | FORMAT* | DESCRIPTION OF OPERATION | AFFECTS | CYCLES |
|---|---|---|---|---|---|---|---|
| **SUBROUTINE BRANCH/RETURN** | BST | R | 001 110 | 2R | Branch To Subroutine On Condition True, Relative | SP | 3 |
| | | A | 001 111 | 3B | Branch To Subroutine On Condition True, Absolute | SP | 3 |
| | BSF | R | 101 110 | 2R | Branch To Subroutine On Condition False, Relative | SP | 3 |
| | | A | 101 111 | 3B | Branch To Subroutine On Condition False, Absolute | SP | 3 |
| | BSN | R | 011 110 | 2R | Branch To Subroutine On Non-Zero Register, Relative | SP | 3 |
| | | A | 011 111 | 3B | Branch To Subroutine On Non-Zero Register, Absolute | SP | 3 |
| | ZBSR | | 101 110 11 | 2ER | Zero Branch To Subroutine Relative, Unconditional | SP | 3 |
| | BSXA | | 101 111 11 | 3EB | Branch To Subroutine, Indexed, Absolute Unconditional (Note 5) | SP | 3 |
| | RET | C | 000 101 | 1Z | Return From Subroutine, Conditional | SP | 3 |
| | | E | 001 101 | 1Z | Return From Subroutine and Enable Interrupt, Conditional | SP, II | 3 |
| **INPUT/OUTPUT** | WRTD | | 111 100 | 1Z | Write Data | — | 2 |
| | REDD | | 011 100 | 1Z | Read Data | CC (Note 1) | 2 |
| | WRTC | | 101 100 | 1Z | Write Control | — | 2 |
| | REDC | | 001 100 | 1Z | Read Control | CC (Note 1) | 2 |
| | WRTE | | 110 101 | 2I | Write Extended | — | 3 |
| | REDE | | 010 101 | 2I | Read Extended | CC (Note 1) | 3 |
| **MISC.** | HALT | | 010 000 00 | 1E | Halt, Enter Wait State | ... | 2 |
| | NOP | | 110 000 00 | 1E | No Operation | — | 2 |
| | TMI | | 111 101 | 2I | Test Under Mask Immediate | CC (Note 6) | 3 |
| **PROGRAM STATUS** | LPS | U | 100 100 10 | 1E | Load Program Status, Upper | F, II, SP | 2 |
| | | L | 100 100 11 | 1E | Load Program Status, Lower | CC, IDC, RS, WC, OVF, COM, C | 2 |
| | SPS | U | 000 100 10 | 1E | Store Program Status, Upper | CC (Note 1) | 2 |
| | | L | 000 100 11 | 1E | Store Program Status, Lower | CC (Note 1) | 2 |
| | CPS | U | 011 101 00 | 2EI | Clear Program Status, Upper, Masked | F, II, SP | 3 |
| | | L | 011 101 01 | 2EI | Clear Program Status, Lower, Masked | CC, IDC, RS, WC, OVF, COM, C | 3 |
| | PPS | U | 011 101 10 | 2EI | Preset Program Status, Upper, Masked | F, II, SP | 3 |
| | | L | 011 101 11 | 2EI | Preset Program Status, Lower, Masked | CC, IDC, RS, WC, OVF, COM, C | 3 |
| | TPS | U | 101 101 00 | 2EI | Test Program Status, Upper, Masked | CC (Note 6) | 3 |
| | | L | 101 101 01 | 2EI | Test Program Status, Lower, Masked | CC (Note 6) | 3 |

*FORMAT CODE: The number indicates the number of bytes. The letter(s) indicate the format type(s). See Fig. 6.

NOTES:
1. Condition code (CC1, CC0): 01 if positive, 00 if zero, 10 if negative.
2. Condition code is set to a meaningless value.
3. Condition code (CC1, CC0): 01 if R0 > r, 00 if R0 = r, 10 if R0 < r.
4. Condition code (CC1, CC0): 01 if r > V, 00 if r = V, 10 if r < V.
5. Index register must be register 3 or 3'.
6. Condition code (CC1, CC0): 00 if all selected bits are 1s, 10 if not all the selected bits are 1s.

## PROGRAM STATUS WORD

PSU

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| S | F | II | Not Used | Not Used | SP2 | SP1 | SP0 |

S  Sense
F  Flag
II  Interrupt Inhibit
SP2  Stack Pointer Two
SP1  Stack Pointer One
SP0  Stack Pointer Zero

PSL

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| CC1 | CC0 | IDC | RS | WC | OVF | COM | C |

CC1  Condition Code One
CC0  Condition Code Zero
IDC  Interdigit Carry
RS  Register Bank Select
WC  With/Without Carry
OVF  Overflow
COM  Logical/Arith. Compare
C  Carry/Borrow

OPERATION CODE  R/V

(Z) REGISTER ADDRESSING

7 6 5 4 3 2 1 0

SYMBOLS:

R – REGISTER NUMBER
V – VALUE OR CONDITION
X – INDEX REGISTER NUMBER
I – INDIRECT BIT

OPERATION CODE  R   DATA MASK OR BINARY VALUE

(I) IMMEDIATE ADDRESSING

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

OPERATION CODE  R/V   RELATIVE DISPLACEMENT
-64≤DISPLACEMENT≤163

(R) RELATIVE ADDRESSING

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

OPERATION CODE  R/X   *INDEX CONTROL  HIGHER ORDER ADDRESS   LOWER ORDER ADDRESS

ABSOLUTE ADDRESSING
(A) (NON-BRANCH INSTRUCTIONS)

23 22 21 20 19 18 17 16   I 15 14 13 12 11 10 9 8   7 6 5 4 3 2 1 0

OPERATION CODE  R/V   HIGHER ORDER ADDRESS   LOWER ORDER ADDRESS
I PAGE

ABSOLUTE ADDRESSING
(B) (BRANCH INSTRUCTIONS)

23 22 21 20 19 18 17 16   15 14 13 12 11 10 9 8   7 6 5 4 3 2 1 0

HIGHER ORDER ADDRESS   LOWER ORDER ADDRESS
UNUSED  PAGE

INDIRECT ADDRESSING

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

OPERATION CODE

(E) MISCELLANEOUS
INSTRUCTIONS

7 6 5 4 3 2 1 0

*INDEX CONTROL:

00 = NON-INDEXED
01 = INDEXED WITH AUTO-INCREMENT
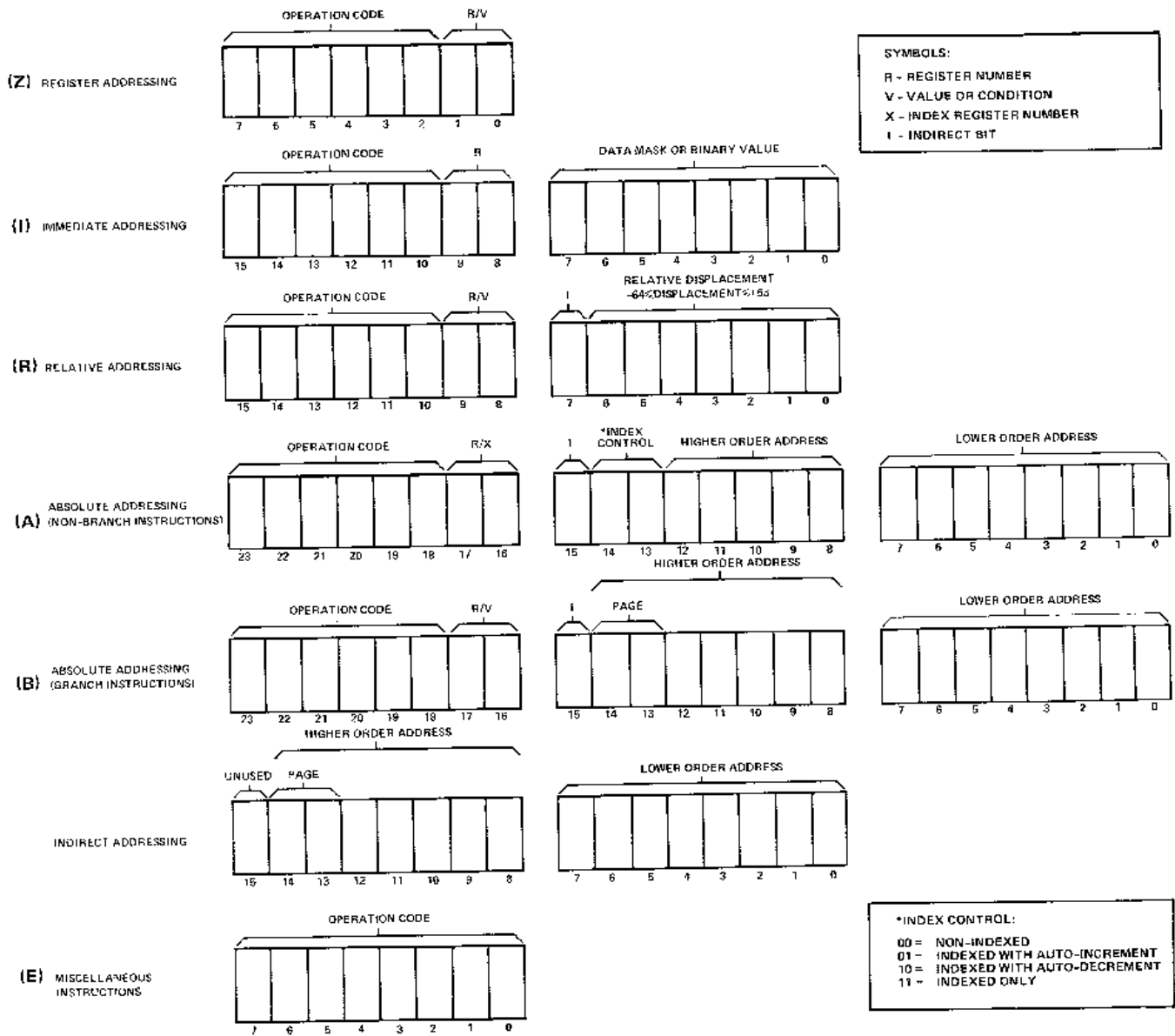10 = INDEXED WITH AUTO-DECREMENT
11 = INDEXED ONLY

Figure 6. INSTRUCTION FORMATS

# SUPPORT

## DOCUMENTATION

The complete manual set is available in a durable 3-ring binder. The binder contains the Hardware Specifications, the Assembler Language Manual, the Software Simulator Manual, and a section called System Application Notes. Our update service provides customers with new application notes and updates to the manual set.

The **Hardware Specification Manual** includes a detailed description of the instruction set, the pin-outs, the AC and DC electrical characteristics, the Input/Output and memory interface signals with timing diagrams, the internal processor organization, and other useful information.

The **Assembler Language Manual** describes how to write programs in the 2650 symbolic assembly language, the pseudo-ops, and how to assemble a 2650 program. Additional information is presented on how to use the assembler program, how to interpret the output listings and how to load object modules.

The **Simulator Manual** describes the nature of the simulation program, how to write simulation commands and how to interpret the simulation output.

**System Application Notes** are included to help the user design with the 2650 processor. These notes present detailed technical information on various subjects of interest and apply to either programming, hardware configuration, or system concepts. This section will continue to grow.

Examples of Application Notes are:

- Serial I/O for the 2650
- Memory Interfaces
- How to use the Decimal Adjust instruction
- I/O Device Selection Methods
- A Minimal System Configuration

## SOFTWARE SUPPORT

Signetics-developed software is available to both the batch processing user and the timesharing user. The Batch Assembler and Batch Simulator are written in standard FORTRAN and may be compiled and executed on most medium to large scale computer systems. Because of the modular design used, it is expected that many minicomputer users will also be able to utilize these programs. The main features of the programs are listed in Tables 3 and 4.

Signetics has also made the Batch Assembler, Batch Simulator and Interactive Simulator available on several international timesharing networks for those customers who wish to run these programs using a timesharing service.

When a customer chooses to follow the timesharing approach, he can also make use of the interactive version of the 2650 Simulator. With the Inter-

### TABLE 3. ASSEMBLER FEATURES

- 2-Pass Assembler
- Diagnostic error messages
- Symbolic addressing including forward references
- Constant generation
- Pseudo-ops to aid programming
- Free format source code

### TABLE 4. SIMULATOR FEATURES

- Cycle Counter for timing estimates
- Instruction fetch break points
- Operand fetch break points
- Trace facilities
- Snapshot dumps
- Patching facility
- Statistical information generated
- Easy-to-use command language
- Optionally selected start and end addresses
- Dynamic changes of simulated registers
- Optionally simulates ROM-RAM environment

active Simulator the software designer can utilize his timesharing terminal to dynamically alter his program and effectively reduce his program development time.

The Signetics 2650 Symbolic Assembly Language has been modeled after other assembly languages; because of this, the assembler is easy to learn and to use.

The Simulator programs are designed to aid the user in testing and correcting his programs. This approach is an alternative to dedicating hardware development tools to one or two programmers or designers for program development. The Simulator allows users to simulate the execution of programs without utilizing a processor. The Simulator utilizes the object module produced by the Assembler as input, and through use of appropriate simulator commands, can display and/or alter the internal registers of the simulated 2650 processor and the simulated memory contents.

The programs are usually delivered delivered on IBM compatible magnetic tape "mini-reels". All programs are in FORTRAN source code as card image records.

A growing Program Library is available to Signetics microprocessor users. We encourage users to submit all non-proprietary programs to Signetics to add to the program library so that we may make them available to other users.

## PROTOTYPING HARDWARE

### PROTOTYPING CARD

In order to develop a product using the Signetics 2650 microprocessor, both hardware and software must be designed. Recognizing that the basic needs of many of our customers for prototyping systems will be similar, Signetics has designed a prototyping card containing a basic microcomputer system. This card provides a starting point for the development of hardware interfaces while simultaneously providing a tool for software checkout.

The first Signetics prototyping card consists of a 2650 processor, ROM memory containing a loader and editor, RAM memory for program storage before committing to PROM or ROM, a TTY interface for easy access, a crystal-controlled clock and two input and output ports (8 bits each).

### SYSTEM COMPATIBLE FAMILIES

The 2650 has been designed to interface directly with industry standard logic and memory families, particularly 7400 and 74LS00 logic families, TTL compatible 5V NMOS memories (Signetics' 2600 series) and bipolar memories (Signetics' 8200 and 82S00 series). Many interface circuits in the 8T00 family are particularly useful for constructing interfaces in 2650 systems.

Other logic families including 8200 TTL, 82S00 STTL and 4000 CMOS are compatible with the 2650. See Table 5.

#### TABLE 5. SYSTEM COMPATIBLE FAMILIES

| Logic | 7400, 8200 | — | TTL |
|---|---|---|---|
| | 74LS00 | — | TTL-LS |
| | 82S00 | — | STTL |
| | 4000 | — | CMOS |
| Memory | 2500 | — | PMOS |
| | 2600 | — | NMOS |
| | 7400, 8200 | — | Bipolar TTL |
| | 82S00 | — | Bipolar STTL |
| Interface | 8T00 | — | TTL, STTL |

# CHAPTER II

# 2650 HARDWARE

GENERAL PURPOSE PROCESSOR
SINGLE CHIP
FIXED INSTRUCTION SET
PARALLEL 8-BIT BINARY OPERATIONS
40 PIN DUAL IN-LINE PACKAGE

N-CHANNEL SILICON GATE MOS TECHNOLOGY
TTL COMPATIBLE INPUTS AND OUTPUTS
SINGLE POWER SUPPLY OF +5 VOLTS
SEVEN GENERAL PURPOSE REGISTERS
RETURN ADDRESS STACK, 8 DEEP, ON CHIP

32K BYTE ADDRESSING RANGE
SEPARATE ADDRESS AND DATA LINES
VARIABLE LENGTH INSTRUCTIONS OF 1, 2, OR 3 BYTES
75 INSTRUCTIONS
MACHINE CYCLE TIME OF $2.4\mu$sec
AT CLOCK FREQUENCY OF 1.25 MHz

DIRECT INSTRUCTIONS TAKE 2, 3 or 4 CYCLES
SINGLE PHASE TTL LEVEL CLOCK INPUT
STATIC LOGIC
TRI-STATE OUTPUT BUSSES
REGISTER, IMMEDIATE, RELATIVE, ABSOLUTE
INDIRECT, AND INDEXED ADDRESSING MODES
VECTOR INTERRUPT FORMAT

# INTRODUCTION

## GENERAL FEATURES

The 2650 processor is a general purpose, single chip, fixed instruction set, parallel 8-bit binary processor. A general purpose processor can perform any data manipulations through execution of a stored sequence of machine instructions. The processor has been designed to closely resemble conventional binary computers, but executes variable length instructions of one to three bytes in length. BCD Arithmetic is made possible through use of a special "DAR" machine instruction.

The 2650 is manufactured using Signetics' N-channel silicon gate MOS technology. N-channel provides high carrier mobility for increased speed and also allows the use of a single 5 volt power supply. Silicon gate provides for better density and speed. Standard 40 pin dual in-line packages are used for the processor.

The 2650 contains a total of seven general purpose registers, each eight bits long. They may be used as source or destination for arithmetic operations, as index registers, and for I/O transfers.

The processor can address up to 32,768 bytes of memory in four pages of 8,192 bytes each. The processor instructions are one, two, or three bytes long, depending on the instruction. Variable length instructions tend to conserve memory space since a one-or two-byte instruction may often be used rather than a three byte instruction. The first byte of each instruction always specifies the operation to be performed and the addressing mode to be used. Most instructions use six of the first eight bits for this purpose, with the remaining two bits forming the register field. Some instructions use the full eight bits as an operation code.

The most complex direct instruction is three bytes long and takes 9.6 microseconds to execute. This figure assumes that the processor is running at its maximum clock rate, and has an associated memory with cycle and access times of one microsecond or less. The fastest instruction executes in 4.8 microseconds.

The clock input to the processor is a single phase pulse train and uses only one interface pin. It requires a normal TTL voltage swing, so no special clock driver is required.

The Data Bus and Address signals are tri-state to provide convenience in system design. Memory and I/O interface signals are asynchronous so that Direct Memory Access (DMA) and multiprocessor operations are easy to implement.

The 2650 has a versatile set of addressing modes used for locating operands for operations. They are described in detail in the INSTRUCTIONS section of this manual.

The interrupt mechanism is implemented as a single level, address vectoring type. Address vectoring means that an interrupting device can force the processor to execute code at a device determined location in memory. The interrupt mechanism is described in detail in the FEATURES section of this manual.

## APPLICATIONS

The ability of the semi-conductor industry to manufacture complete general purpose processors on single chips represents a significant technological advance which should prove to be of great benefit to digital systems manufacturers. In terms of chip size and density of transistors, the processors are simply extensions of the continually evolving MOS technology. But in terms of function provided, a significant threshold has been crossed.

By allowing designers to convert from hardware logic to programmed logic, the integrated processor provides several important advantages.

1. Logic functions may be implemented in memory bits instead of logic gates. The user then has greater access to the advantages of memory circuits. Memories use patterned circuitry and thus provide greater density and therefore greater economy.
2. Random logic implementations of complex functions are highly specialized and cannot be used in other applications. They are not often used in large volume. Programmed logic, on the other hand, relies on general purpose processor and memory circuits that are used in many applications. Thus, economies of volume are available for both the user and the manufacturer.
3. Because the functional specialization resides in the user's program rather than the hardware logic, changes, corrections and additions can be much easier to make and can be accomplished in a much shorter time.
4. With the programmed logic approach it is often possible to add new features and create new products simply by writing new programs.
5. The design cycle of a system using programmed logic can be significantly shorter than a similar system that attempts to use custom random logic. The debugging cycle is also greatly compressed.

A general purpose processor designed to implement programmed logic has many characteristics that allow it to do conventional computer operations as well. Many applications will specialize in programmed logic or in data processing, but some will take advantage of both areas. In a line printer application, for example, a processor can act primarily as a controller handling the housekeeping duties, control sequencing and data interfacing for the printer. It also might buffer the data or do some code conversions, but that is not its primary duty. On the other hand, in a text editing intelligent terminal, the processor is mainly concerned with data manipulation since it handles code translations, display paging, insertions, deletions, line justification, hyphenation, etc.

A point-of-sale type of terminal represents an application that combines both control and data processing activities for the processor. Coordinating the activities of the various devices and displays that make up the terminal is an important part of the job, as are the calculations that are essential to the operation of the machine.

# INTERNAL ORGANIZATION

## INTERNAL REGISTERS

The block diagram for the 2650 shows the major internal components and the data paths that interconnect them. In order for the processor to execute an instruction, it performs the following general steps:

1. The Instruction Address Register provides an address for memory.
2. The first byte of an instruction is fetched from memory and stored in the Instruction Register.
3. The Instruction Register is decoded to determine the type of instruction and the addressing mode.
4. If an operand from memory is required, the operand address is resolved and loaded into the Operand Address Register.
5. The operand is fetched from memory and the operation is executed.
6. The first byte of the next instruction is fetched.

The Instruction Register (IR) holds the first byte of each instruction and directs the subsequent operations required to execute each instruction. The IR contents are decoded and used in conjunction with the timing information to control the activation and sequencing of all the other elements on the chip. The Holding Register (HR) is used in some multiple-byte instructions to contain further instruction information and partial absolute addresses.

The Arithmetic Logic Unit (ALU) is used to perform all of the data manipulation operations, including Load, Store, Add, Subtract, And, Inclusive Or, Exclusive Or, Compare, Rotate, Increment and Decrement. It contains and controls the Carry bit, the Overflow bit, the Interdigit Carry and the Condition Code Register.

The Register Stack contains six registers that are organized into two banks of three registers each. The Register Select bit (RS) picks one of the two banks to be accessed by instructions. In order to accomodate the register-to-register instructions, register zero (RO) is outside the array. Thus, register zero is always available along with one set of three registers.

The Address Adder (AA) is used to increment the instruction address and to calculate relative and indexed addresses.

The Instruction Address Register (IAR) holds the address of the next instruction byte to be accessed. The Operand Address Register (OAR) stores operand addresses and sometimes contains intermediate results during effective address calculations.

The Return Address Stack (RAS) is an eight level, Last In, First Out (LIFO) storage which receives the return address whenever a Branch-to-Subroutine instruction is executed. When a Return instruction is executed, the RAS provides the last return address for the processor's IAR. The stack contains eight levels of storage so that subroutines may be nested up to eight levels deep. The Stack Pointer (SP) is a three bit wraparound counter that indicates the next available level in the stack. It always points to the current address.
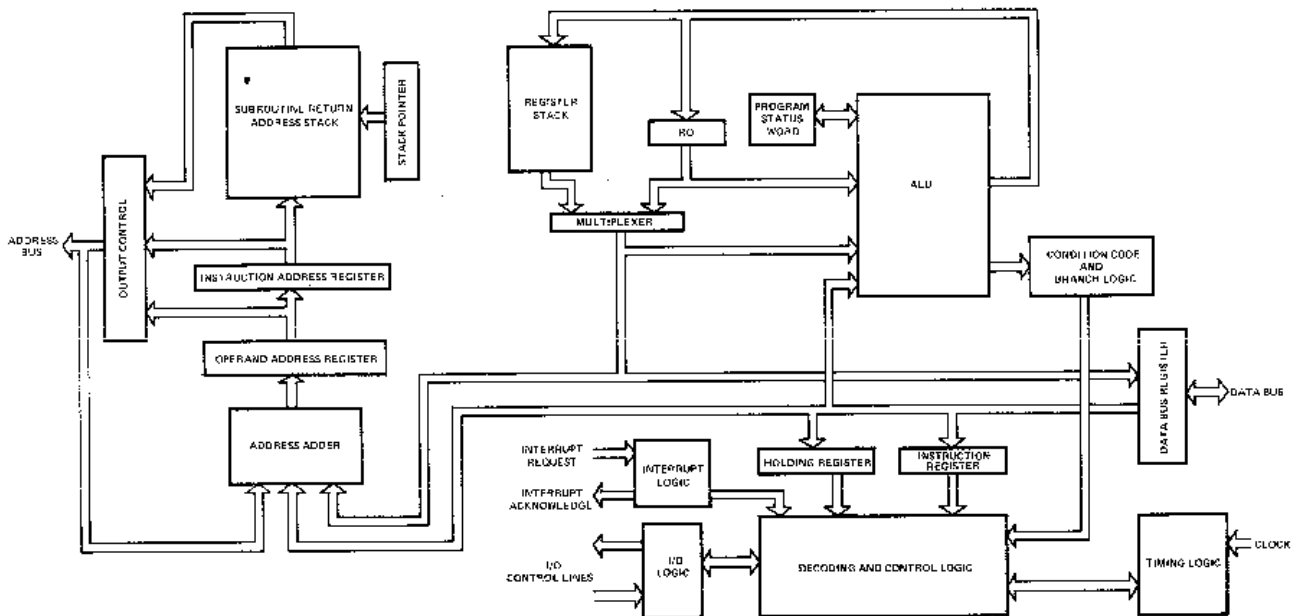
Figure 7. SIGNETICS 2650 BLOCK DIAGRAM

## PROGRAM STATUS WORD

The Program Status Word (PSW) is a special purpose register within the processor that contains status and control bits. It is 16 bits long and is divided into two bytes called the Program Status Upper (PSU) and the Program Status Lower (PSL).

The PSW bits may be tested, loaded, stored, preset or cleared using the instructions which effect the PSW. The sense bit, however, cannot be set or cleared because it is directly connected to pin #1.

| PSU | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | S | F | II | Not Used | Not Used | SP2 | SP1 | SP0 |

- S    Sense
- F    Flag
- II    Interrupt Inhibit
- SP2    Stack Pointer Two
- SP1    Stack Pointer One
- SP0    Stack Pointer Zero

| PSL | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | CC1 | CC0 | IDC | RS | WC | OVF | COM | C |

- CC1    Condition Code One
- CC0    Condition Code Zero
- IDC    Interdigit Carry
- RS    Register Bank Select
- WC    With/Without Carry
- OVF    Overflow
- COM    Logical/Arithmetic Compare
- C    Carry/Borrow

## SENSE (S)

The Sense bit in the PSU reflects the logic state of the sense input to the processor at pin #1. The sense bit is not affected by the LPSU, PPSU, or CPSU instructions. When the PSU is tested (TPSU) or stored into register zero (SPSU), bit #7 reflects the state of the sense pin at the time of the instruction execution.

## FLAG (F)

The Flag bit is a simple latch that drives the Flag output (pin #40) on the processor.

## INTERRUPT INHIBIT (II)

When the Interrupt Inhibit (II) bit is set, the processor will not recognize an incoming interrupt. When interrupts are enabled (II=0), and an interrupt signal occurs, the inhibit bit in the PSU is then automatically set. When a Return-and-Enable instruction is executed, the inhibit bit is automatically cleared.

## STACK POINTER (SP)

The three Stack Pointer bits are used to address locations in the Return Address Stack (RAS). The SP designates the stack level which contains the current return address. The three SP bits are organized as a binary counter which is automatically incremented with execution of Branch-to-Subroutine instructions, and decremented with execution of Return instructions.

## CONDITION CODE (CC)

The Condition Code is a two bit register which is set by the processor whenever a general purpose register is loaded or modified by the execution of an instruction. Additionally, the CC is set to reflect the relative value of two bytes whenever a compare instruction is executed.

The following table indicates the setting of the Condition Code whenever data is set into a general purpose register. The data byte is interpreted as an 8-bit, two's complement number.

| Register Contents | CC1 | CC0 |
|-------------------|-----|-----|
| Positive | 0 | 1 |
| Zero | 0 | 0 |
| Negative | 1 | 0 |

For compare instructions the following table summarizes the setting of the CC. The data is compared as two 8-bit absolute numbers if bit #1, the COM bit, of the Program Status Lower byte is set to indicate "logical" compare (COM=1). If the COM bit indicates "arithmetic" (COM=0), the comparison instructions interpret the data bytes as two 8-bit two's complement binary numbers.

| Register to Storage Compare Instruction | Register to Register Compare Instruction | CC1 | CC0 |
|-----------------------------------------|------------------------------------------|-----|-----|
| Reg X Greater Than Storage | Reg 0 Greater Than Reg X | 0 | 1 |
| Reg X Equal to Storage | Reg 0 Equal to Reg X | 0 | 0 |
| Reg X Less Than Storage | Reg 0 Less Than Reg X | 1 | 0 |

The CC is never set to 11 by normal processor operations, but it may be explicitly set to 11 through LPSL or PPSL instruction execution.

## INTERDIGIT CARRY (DC)

For BCD arithmetic operations it is sometimes essential to know if there was a carry from bit #3 to bit #4 during the execution of an arithmetic instruction.

The IDC reflects the value of the Interdigit Carry from the previous add or subtract instruction. After any add or subtract instruction execution, the IDC contains the carry or borrow out of bit #3.

The IDC is also set upon execution of Rotate instructions when the WC bit in the PSW is set. The IDC will reflect the same information as bit #5 of the operand register after the rotate is executed. See Figure 8.

## REGISTER SELECT (RS)

There are two banks of general purpose registers with three registers in each bank. The register select bit is used to specify which set of three general purpose registers will be currently used. Register zero is common and is always available to the program. An individual instruction may address only four registers, but the bank select feature effectively expands the available on-chip registers to seven. When the Register Select Bit is "0", registers 1, 2, & 3 in register bank #0 will be accessible, and when the bit is "1", registers 1, 2, & 3 in register bank #1 will be accessible.

## WITH/WITHOUT CARRY (WC)

This bit controls the execution of the add, the subtract and the rotate instructions.

Whenever an add or a subtract instruction executes, the following bits are either set or cleared: Carry/Borrow (C), Overflow (OVF), and Interdigit Carry (IDC). These bits are set or reset without regard to the value of the WC bit. However, when WC=1, the final value of the carry bit affects the result of an add or a subtract instruction, i.e., the carry bit is either added (add instruction) or subtracted (subtract instruction) from the ALU.

Whenever a rotate instruction executes with WC=0, only the eight bits of the rotated register are affected. However, when WC=1, the following bits are also affected: Carry/Borrow (C), Overflow (OVF) and Interdigit Carry (IDC). The carry/borrow bit is combined with the 8-bit register to make a nine-bit rotate (see Figure 8). The overflow bit is set whenever the sign bit (bit 7) of the rotated register changes its value, i.e., from a zero (0) to a one (1) or from a one (1) to a zero (0). The interdigit carry bit is set to the new value of bit 5 of the rotated register.

## OVERFLOW (OVF)

The overflow bit is set during add or subtract instruction executions whenever the two initial operands have the same sign but the result has a different sign. Operands with different signs cannot cause overflow. Example: A binary +124 (01111100) added to a binary +64 (01000000) produces a result of (10111100) which is interpreted in two's complement form as a −68. The true answer would be 188, but that answer cannot be contained in the set of 8-bit, two's complement numbers used by the processor, so the OVF bit is set.

Rotate instructions also cause OVF to be set whenever the sign of the rotated byte changes.

ROTATE REGISTER RIGHT WITH CARRY


ROTATE REGISTER RIGHT WITHOUT CARRY


ROTATE REGISTER LEFT WITH CARRY
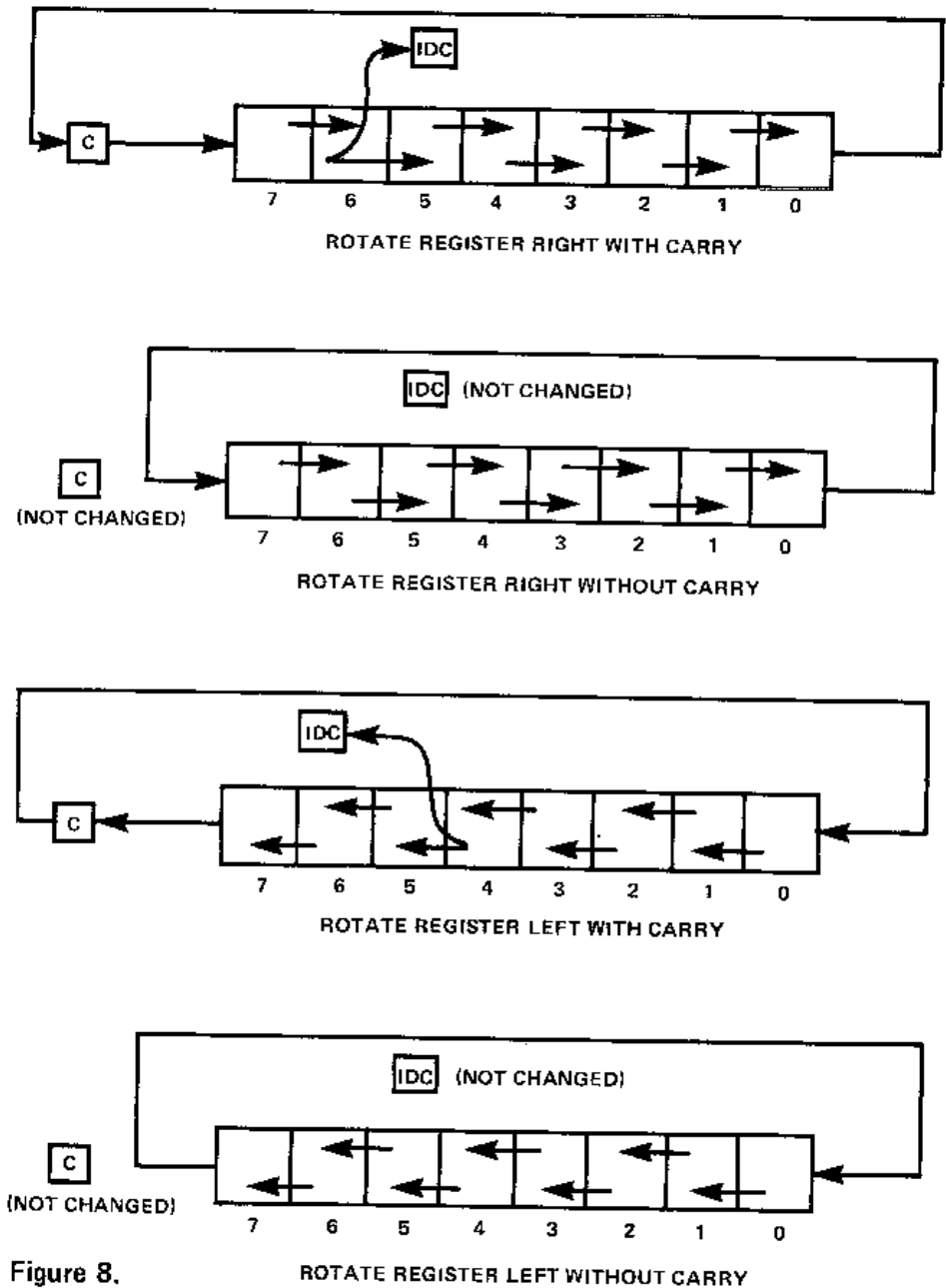
**Figure 8.**

ROTATE REGISTER LEFT WITHOUT CARRY

## COMPARE (COM)

The compare control bit determines the type of comparison that is executed with the Compare instructions. Either logical or arithmetic comparisons may be made. The arithmetic compare assumes that the comparison is between 8-bit, two's complement numbers. The logical compare assumes that the comparison is between 8-bit positive binary numbers. When COM is set to 1, the comparisons will be logical, and when COM is set to 0, the comparisons will be arithmetic. See Condition Code (CC).

## CARRY (C)

The Carry bit is set by the execution of any add or subtract instruction that results in a carry or borrow out of the high order bit of the ALU. The carry bit is set to 1 by an add instruction that generates a carry, and a subtract instruction that does *not* generate a borrow. Inversely, an add that does not generate a carry causes the C bit to be cleared, and a subtract instruction that generates a borrow also clears the carry bit.

Even though a borrow is indicated by a zero in the Carry bit, the processor will correctly interpret the zero during subtract with borrow operations as in the following table.

| Low Order bit Minuend | Low Order bit Subtrahend | Carry bit Borrow bit | Low Order Bit Result |
|:---:|:---:|:---:|:---:|
| 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 0 |

The carry bit may also be set or cleared by rotate instructions as described earlier under "With/Without Carry".

To perform an Add with Carry or a Subtract with Borrow, the WC bit must be set.

## MEMORY ORGANIZATION

The 2650 has a maximum memory addressing capability of $0_{10}$ — $32,767_{10}$ locations. As may be seen in the INSTRUCTIONS section of this manual, most direct addressing instructions have thirteen bits allocated for the direct address. Since thirteen bits can only address locations $0_{10}$ — $8,191_{10}$, a paging system was implemented to accomodate the entire address range.

The memory may be thought of as being divided into four pages of 8,192 bytes each. The addresses in each page range as in the following chart:

| | START ADDRESS | END ADDRESS | |
|---|---|---|---|
| page 0 | 00000000000000 | 00111111111111 | $0_{10}$ — $8191_{10}$ |
| page 1 | 01000000000000 | 01111111111111 | $8192_{10}$ — $16,383_{10}$ |
| page 2 | 10000000000000 | 10111111111111 | $16,384_{10}$ — $24,575_{10}$ |
| page 3 | 11000000000000 | 11111111111111 | $24,576_{10}$ — $32,767_{10}$ |

The low order 13-bits in every page range through the same set of numbers. These 13-bits are the same 13-bits addressed by non-branch instructions and are also the same 13-bits which are brought out of the 2650 on the address lines ADR0 — ADR12.

The high order two bits of the 15-bit address are known as the page bits. The page bits when examined by themselves also represent, in binary, the number of the memory page. Thus, the address $\underline{01}0000001101101$ is known as address location $109_{10}$ in page 1. The page bits, corresponding to ADR13 and ADR14 are brought out of the 2650 on pins 19 & 18. These bits may be used for memory access when more than 8,192 bytes of memory are connected.

There are no instructions to explicitly set the page bits. They may be set through execution of direct or indirect, branch or branch-to-subroutine instructions. It may be seen that these instructions (see INSTRUCTION Section) have 15-bits allocated for address and when such an instruction is executed, the two high order address bits are set into the page bit latches in the 2650 processor and will appear on ADR13 and ADR14 during memory accesses until they are specifically changed.

For memory access from non-branch instructions, the 13-bit direct address will address the corresponding location within the current page only. However, the non-branch memory access instruction may access any byte in any page through indirect addressing which provides the full 15-bit address. In the case of non-branch instructions, the page bits are only temporarily changed to correspond to the high order two bits of the 15-bit indirect address used to fetch the argument byte. Immediately after the memory access, ADR13 & ADR14 will revert to their previous value.

The consequences of this page address system may be summarized by the following statements.

1. The RESET signal clears both page latches, i.e., ADR13 & ADR14 are cleared to zero.
2. All non-branch, direct memory access instructions address memory within the current page.
3. All non-branch, memory access instructions may access any byte of addressable memory through use of indirect addressing which temporarily changes the page bits for the argument access, but which revert back to their previous state immediately following instruction execution.
4. All direct and indirect addressing branch instructions set the page bits to correspond to the high order two bits of the 15 bit address.
5. Programs may *not* flow across page boundaries, they must branch to set the page bits.
6. Interrupts always drive the processor to page zero.

# INTERFACE

### RESET

The RESET signal is used to cause the 2650 to begin processing from a known state. RESET will normally be used to initialize the processor after power-up or to restart a program. RESET clears the Interrupt Inhibit control bit, clears the internal interrupt-waiting signal, and initializes the IAR to zero. RESET is normally low during program execution, and must be driven high to activate the RESET function. The leading and trailing edges may be asynchronous with respect to the clock. The RESET signal must be at least three clock periods long. If RESET alone is used to initiate processing, the first instruction will be fetched from memory location page zero byte zero after the RESET signal is removed. Any instruction may be programmed for this location including a Branch to some program located elsewhere.

Processing can also be initiated by combining an interrupt with a reset. In this case, the first instruction to be executed will be at the interrupt address.

### CLOCK

The clock signal is a positive-going pulse train that determines the instruction execution rate. Three clock periods comprise a processor cycle. Direct instructions are 2, 3, or 4 processor cycles long, depending on the specific type of instruction. Indirect addressing adds two processor cycles to the direct instruction times.

### PAUSE

The PAUSE input provides a means for temporarily stopping the execution of a program. When PAUSE is driven low, the 2650 finishes the instruction in progress and then enters the WAIT state. When PAUSE goes high, program execution continues with the next instruction. If PAUSE is turned on then off again before the last cycle of the current instruction begins, program execution continues without pause. If both PAUSE and INTREQ occur prior to the last cycle of the current instruction, the interrupt will be recognized, and an INTACK will be generated immediately following release of the PAUSE. The next instruction to be executed will be a ZBSR to service the interrupt.

If an INTREQ occurs while the 2650 is in a WAIT state due to a PAUSE, the interrupt will be acknowledged and serviced after the execution of the next normal instruction following release of the PAUSE.

### INTREQ

The Interrupt Request input (normally high) is a means for external devices to change the flow of program execution. When the processor recognizes an INTREQ, i.e., INTREQ is driven low, it finishes the instruction in progress, inserts a ZBSR instruction into the IR, turns on the Interrupt Inhibit bit in the PSU, and then responds with INTACK and OPREQ signals. Upon receipt of INTACK, the interrupting device may raise the INTREQ line and present a data byte to the processor on the DBUS. The required byte takes the same form as the second byte of a ZBSR instruction. Thus, the interrupt initiated Branch-to-Subroutine instruction may have a relative target address anywhere within the first or last 64 bytes of memory page 0. If indirect addressing is specified, a branch to any location in addressable memory is possible.

For devices that do not need the flexibility of the multiple target addresses, a byte of eight zeroes may be presented and will cause a direct subroutine branch to memory location zero in page zero. The relative address presented by the interrupting device is handled with a normal I/O read sequence using the usual interface control signals. The addition of the INTACK signal distinguishes the interrupt address operation from other operations that may take place as part of the execution of the interrupted instruction. At the same time that it acknowledges the $\overline{\text{INTREQ}}$, the processor automatically sets the bit that inhibits recognition of further interrupts. The Interrupt Inhibit bit may be cleared anytime during the interrupt service routine, or a Return-and-Enable instruction may be used to enable interrupts upon leaving the routine. If an $\overline{\text{INTREQ}}$ is waiting when the Interrupt Inhibit bit is cleared, it will be recognized and processed immediately without the execution of an intervening instruction.

### $\overline{\text{OPACK}}$

The Operation Acknowledge signal is a reply from external memory or I/O devices as a response to the Operation Request signal from the processor. OPREQ is used to initiate an external operation. The affected external device indicates to the processor that the operation is complete by turning on the $\overline{\text{OPACK}}$ signal. This procedure allows asynchronous functioning of external devices.

If a Memory operation is initiated by the processor, the memory system will provide an $\overline{\text{OPACK}}$ when the requested memory data is valid on the Data Bus. If an I/O operation is initiated by the processor, the addressed I/0 device may respond with an $\overline{\text{OPACK}}$ as soon as the write data is accepted from the Data Bus, or after the read operation is completed. However, in order to avoid slowing down the processor when using memories or I/0 devices that are just fast enough to keep the processor operating at full speed the $\overline{\text{OPACK}}$ signal must be returned before the external operation is completed. Any $\overline{\text{OPACK}}$ that is returned within 600 nsec. following an OPREQ will not delay the processor. Data from a read operation can return up to 1000 nsec. after an OPREQ is sent and still be accepted by the processor without causing delays. If all devices will always respond within these time limits, the $\overline{\text{OPACK}}$ line may be permanently connected in the ON (low) state. Whenever an $\overline{\text{OPACK}}$ is not available within that time, the processor will delay instruction execution until the first clock following receipt of the $\overline{\text{OPACK}}$. All output line conditions remain unchanged during the delay and the processor does not enter the WAIT state. $\overline{\text{OPACK}}$ is true in the low state and false in the high state.

### SENSE

The SENSE line provides an input line to the 2650 that is independent of the normal I/O Bus structures. The SENSE signal is connected directly to one of the bits in the Program Status Word. It may be stored or tested by an executing program. When a store (SPSU) or test (TPSU) instruction is executed, the SENSE line is sampled during the last cycle of the instruction.

Through proper programming techniques the SENSE signal may be used to implement a direct serial data input channel, or it may be used to present any bit of information that the designer chooses.

The SENSE input and FLAG output facilities provide the simplest method of communicating data in or out of the 2650 Processor as neither address decoding nor synchronization with other processor signals is necessary.

## $\overline{\text{ADREN}}$

The Address Enable signal allows external control of the tri-state address outputs (ADR0-ADR12). When $\overline{\text{ADREN}}$ is driven high, the address lines are switched to their third state and show a high output impedance. This feature allows wired-OR connections with other signals. The ADR13 and ADR14 lines which are multiplexed with other signals are not affected by this signal.

When a system is not designed to utilize the feature, the $\overline{\text{ADREN}}$ input may be connected permanently to a low signal source.

## $\overline{\text{DBUSEN}}$

The Data Bus Enable signal allows external control of the tri-state Data Bus output drivers. When $\overline{\text{DBUSEN}}$ is driven high, the Data Bus will exhibit a high output impedance. This allows wired-OR connection with other signals.

When a system is not designed to utilize this feature, the $\overline{\text{DBUSEN}}$ input may be permanently connected to a low signal source.

## DBUS

The Data Bus signals form an 8-bit bi-directional data path in and out of the processor. Memory and I/O operations use the Data Bus to transfer the write or read data to or from memory.

The direction of the data flow on the Data Bus is indicated by the state of the $\overline{\text{R}}$/W line. For Write operations, the output buffers in the processor output data to the bus for use by memory or by external devices. For Read operations, the buffers are disabled and the data condition of the bus is sensed by the processor. The output buffers may also be disabled by the $\overline{\text{DBUSEN}}$ signal.

The signals on the data bus are true signals, i.e., a one is a high level and a zero is low.

## ADR

The Address signals form a 15 bit path out of the processor, and are used primarily to supply memory addresses during memory operations. The addresses remain valid as long as OPREQ is on so that no external address register is required. For extended I/O operations, the low order eight bits of the ADR lines are used to output the immediate byte of the instruction which typically is interpreted as a device address.

The 13 low order lines of the address are used only for address information. The two high order address lines are multiplexed with I/O control information. During memory operations, the lines serve as memory addresses. During I/O operations they serve as the D/$\overline{\text{C}}$ and E/$\overline{\text{NE}}$ control lines. Demultiplexing is accomplished through use of the Memory/$\overline{\text{IO}}$ Control line.

The line ADR0 carries the low order address bit, and ADR12 carries the high order address bit. The output drivers may be disabled by the $\overline{\text{ADREN}}$ signal.

The signals on the address bus are true, i.e., a one is a high level and a zero is low.

## OPREQ

The Operation Request output is the coordinating signal for all external operations. The M/$\overline{\text{IO}}$, $\overline{\text{R}}$/W, E/$\overline{\text{NE}}$, D/$\overline{\text{C}}$ and INTACK lines are operation control signals that describe the nature of the external operation when the OPREQ line is true. The DBUS and ADR bus also should not be considered

valid except when OPREQ is in the high, or on state.

No output signals from the processor will change as long as OPREQ is on, with the exception of WRP. OPREQ will stay on until the external operation is complete, as indicated by the $\overline{OPACK}$ input. The processor delays all internal activity following an OPREQ until the $\overline{OPACK}$ signal is received.

### INTACK

The Interrupt Acknowledge signal is used by the processor to respond to an external interrupt. When an $\overline{INTREQ}$ is received, the current instruction is completed before the interrupt is serviced. When the processor is ready to accept the interrupt it sets the INTACK to the high, or on, state along with OPREQ. The interrupting device then presents a relative address byte to the DBUS and responds with an $\overline{OPACK}$ signal. $\overline{INTREQ}$ may be turned off anytime following INTACK. INTACK will fall after the processor receives the $\overline{OPACK}$ signal.

### M/$\overline{IO}$

The Memory/$\overline{IO}$ output is one of the operation control signals that defines external operations. M/$\overline{IO}$ indicates whether an operation is memory or I/O and should be used to gate Read or Write signals between memory or I/O devices.

The state of M/$\overline{IO}$ will not change while OPREQ is high.

The high state corresponds to Memory operation, and the low state corresponds to an I/O operation.

### $\overline{R}$/W

The Read/Write output is one of the operation control signals that defines external operations. $\overline{R}$/W indicates whether an operation is *Read* or *Write*. It controls the nature of the external operation and indicates in which direction the DBUS is pointing. $\overline{R}$/W should not be considered valid until OPREQ is on and the state of the $\overline{R}$/W line does not change as long as OPREQ is on.

The high state corresponds to the Write operation, and the low state corresponds to the Read operation.

### D/$\overline{C}$

The Data/Control Output is an I/O signal which is used to discriminate between the execution of the two types of one byte I/O instructions. There are four one byte I/O instructions; WRTC, WRTD, REDC, REDD. When Read Control or Write Control is executed, the D/$\overline{C}$ line takes on the low state which indicates Control ($\overline{C}$). When Read Data or Write Data is executed, the D/$\overline{C}$ line takes on the high state, indicating Data (D).

D/$\overline{C}$ should not be considered valid until (a) OPREQ is on and (b) M/$\overline{IO}$ indicates an I/O operation and (c) E/$\overline{NE}$ indicates a non-extended (one byte) operation. D/$\overline{C}$ is multiplexed with a high order address line. When the M/$\overline{IO}$ line is in the I/O state, the ADR14-D/$\overline{C}$ line should be interpreted as "D/$\overline{C}$". (When the M/$\overline{IO}$ line is in the M state, the ADR14-D/$\overline{C}$ line should be interpreted as memory address line #14.)

## E/$\overline{\text{NE}}$

The Extended/Non-Extended output is the operation control signal that is used to discriminate between two byte and one byte I/O operations. Thus, E/$\overline{\text{NE}}$ indicates the presence or absence of valid information on the eight low order address lines during I/O operations.

E/$\overline{\text{NE}}$ should not be considered valid until (a) OPREQ is on and (b) M/$\overline{\text{IO}}$ indicates an I/O operation. E/$\overline{\text{NE}}$ is multiplexed with a high order address line. When the M/$\overline{\text{IO}}$ line is in the I/O state, the ADR13-E/$\overline{\text{NE}}$ line should be interpreted as "E/$\overline{\text{NE}}$". (When the M/$\overline{\text{IO}}$ line is in the M state, the ADR13-E/$\overline{\text{NE}}$ line should be interpreted as memory address bit #13.)

There are six I/O instructions; REDE, WRTE, REDC, REDD, WRTC, WRTD. When either of the two byte I/O instructions is executed (REDE, WRTE), the E/$\overline{\text{NE}}$ line takes on the high state or "Extended" indication. When any of the one byte I/O instructions is executed, the line takes on the low state or "non-extended" indication.

## RUN/$\overline{\text{WAIT}}$

The RUN/$\overline{\text{WAIT}}$ output signal indicates the Run/Wait Status of the processor. The WAIT state may be entered by executing a HALT instruction or by turning on the $\overline{\text{PAUSE}}$ input. At any other time the processor will be in a RUN state.

When the processor is executing instructions, the line is in the high or RUN state; when in the WAIT state, the line is held low.

The HALT initiated WAIT condition can be changed to RUN by a RESET or an interrupt. The $\overline{\text{PAUSE}}$ initiated WAIT condition can be changed to RUN by removing the $\overline{\text{PAUSE}}$ input.

If a RESET occurs during a $\overline{\text{PAUSE}}$ initiated WAIT state and the $\overline{\text{PAUSE}}$ remains low; the processor will be reset, fetch one instruction from page zero byte zero and return to the WAIT state. When the $\overline{\text{PAUSE}}$ is eventually removed, the previously fetched instruction will be executed.

## FLAG

The FLAG output indicates the state of the Flag bit in the PSW. Any change in the Flag bit is reflected by a change in the FLAG output. A one bit in the Flag will give a high level on the FLAG output pin. The LPSU, PPSU, and CPSU instructions can change the state of the Flag bit. The FLAG output is always a valid indication of the state of the Flag bit without regard for the status of the processor or control signals. Changes in the Flag bit are synchronized with the last cycle of the changing instruction.

## WRP

The Write Pulse output is a timing signal from the processor that provides a positive-going pulse in the middle of each requested write operation (memory or I/O) and a high level during read operations. The WRP is designed to be used with Signetics 2606 R/W memory circuits to provide a timed Chip Enable signal. For use with memory, it may be gated with the M/$\overline{\text{IO}}$ signal to generate a Memory Write Pulse.

Because the WRP pulse occurs during any write operation, it may also be used with I/O write operations where convenient.

## SIGNAL TIMING

The Clock input to the 2650 provides the basic timing information that the processor uses for all its internal and external operations. The clock rate determines the instruction execution rate, except to the extent that external memories and devices slow down the processor. Each internal processor cycle is composed of three clock periods as shown in Figure 9, 2650 TIMING DIAGRAMS.



GENERAL TIMING

INTERRUPT TIMING

TRI-STATE BUS TIMING
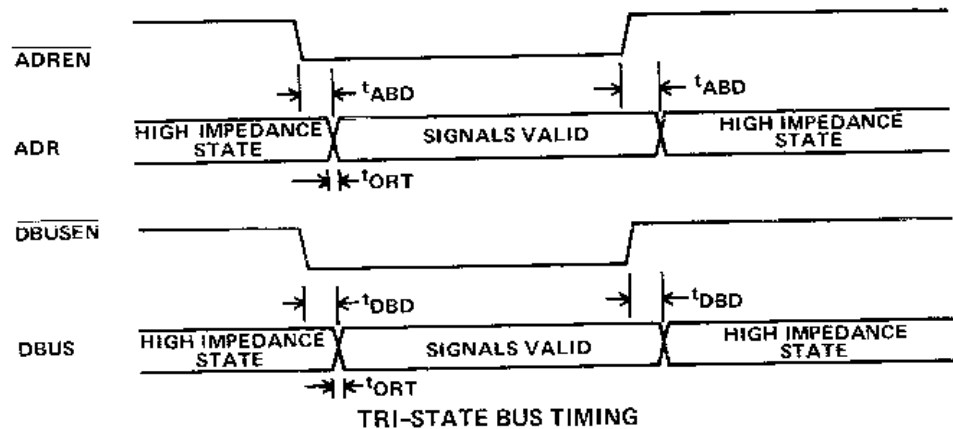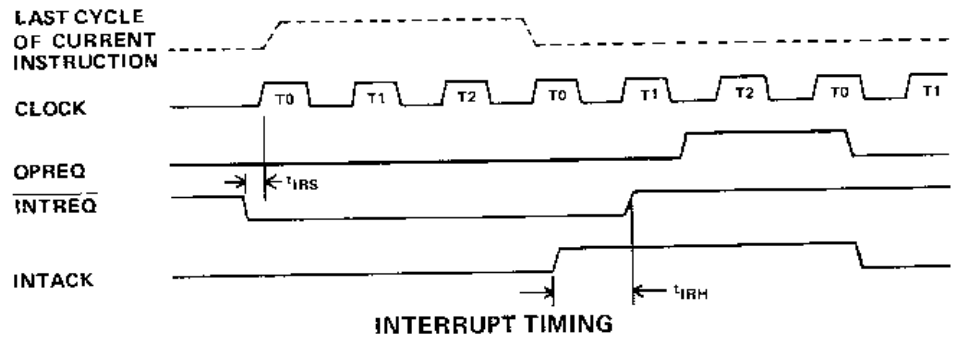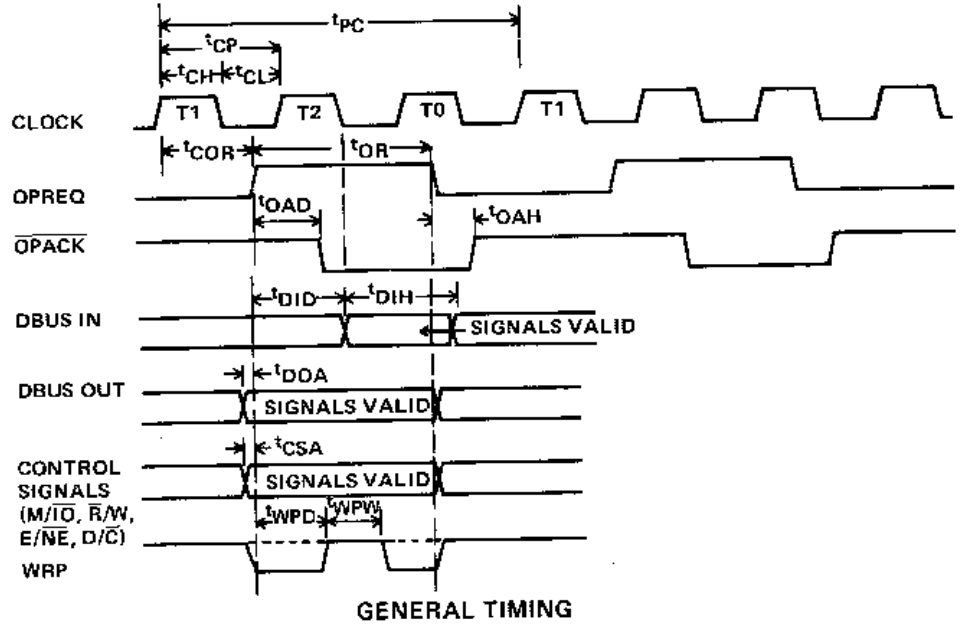
Figure 9. 2650 TIMING DIAGRAMS

OPREQ is the master control signal that coordinates all operations external to the processor. Many of the other signal interactions are related to OPREQ. The timing diagram assumes that the clock periods are constant and that $\overline{OPACK}$ is returned in time to avoid delaying instruction execution. In that case, OPREQ will be high for 1.5 clock periods ($1/2$ of $t_{pc}$) and then will be low for another 1.5 clock periods.

The interface control signals have been designed to implement asynchronous interfaces for both memory and input/output devices. The control signals are relatively simple and provide the following advantages: no external synchronizing is necessary, external devices may run at any data rate up to the processor's maximum I/O data rate, and because data signals are furnished with guard signals the external devices are often relieved of the necessity of latching information such as memory address.

### MEMORY READ TIMING

The following signals are involved in the processor's memory read sequence, as shown in Figure 10.

| | |
|---|---|
| OPREQ | = Operation Request |
| DBUS0-DBUS7* | = Data Bus |
| ADR0-ADR12 | = Address Bus |
| ADR13 | = Address bit 13 |
| ADR14 | = Address bit 14 |
| M/$\overline{IO}$ | = Memory/Input-Output |
| $\overline{R}$/W | = Read/Write |
| $\overline{OPACK}$* | = Operation Acknowledge |

The signals marked with an asterisk are sent from the memory device to the processor. The other signals are developed by the processor.

OPREQ is a guard signal which must be valid (high) for the other signals to have meaning. When reading main memory the 2650 simultaneously switches OPREQ to a high state, M/$\overline{IO}$ to M (memory), $\overline{R}$/W to $\overline{R}$ (Read), and places the memory address on lines ADR0-ADR14. Remember that
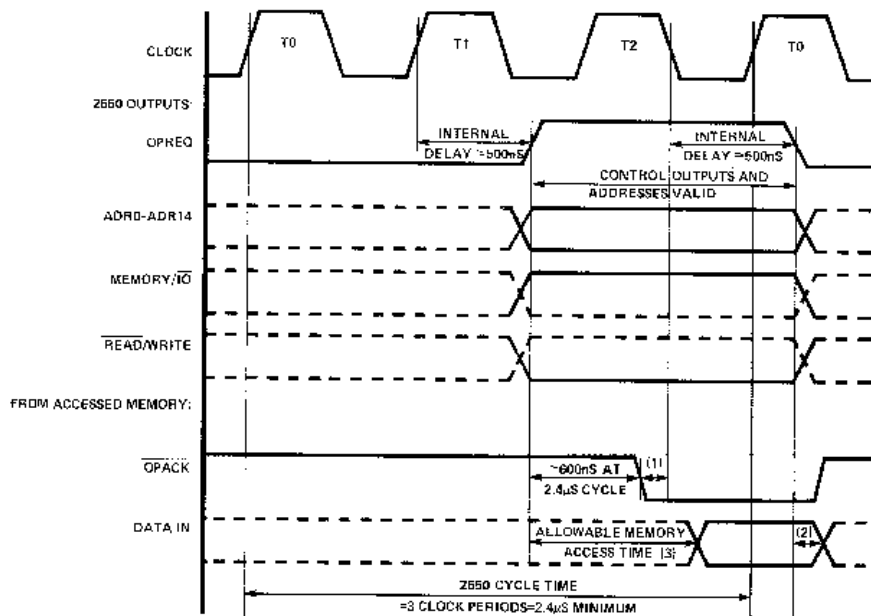


NOTES: [1] OPACK must go low at least 100 nS before the trailing edge of T2 in order not to slow down the 2650.

[2] DATA IN signals must be valid for 50nS after the trailing edge of OPREQ.

[3] Allowable memory access time is 1us with 2.4µs cycle time.

**Figure 10. MEMORY READ SEQUENCE**

ADR13 & ADR14 are multiplexed with other signals and must be logically ANDed with OPREQ and M to be interpreted. Of course, ADR13 & ADR14 may be ignored if only page zero (8,192 bytes) is used.

Once the memory logic has determined the simultaneous existance of the signals mentioned above, it places the true data corresponding to the given address location on the data bus (DBUS0 to DBUS7), and returns an $\overline{\text{OPACK}}$ signal to the processor. The processor, recognizing the $\overline{\text{OPACK}}$, strobes the data into the receiving register and lowers the OPREQ. This completes the memory read sequence.

If the $\overline{\text{OPACK}}$ signal is delayed by the memory device, the processor waits until it is received. OPREQ is lowered only after the receipt of $\overline{\text{OPACK}}$. The memory device should raise $\overline{\text{OPACK}}$ after OPREQ falls.

### MEMORY WRITE TIMING

The signals involved with the processor's memory write sequence are similar to those used in the memory read sequence with the following exceptions: 1) the $\overline{\text{R}}/\text{W}$ signal is in the W state and, 2) the WRP signal provides a positive going pulse during the write sequence which may be used as a chip enable, write pulse, etc.

Figure 11 demonstrates the signals that occur during a memory write.
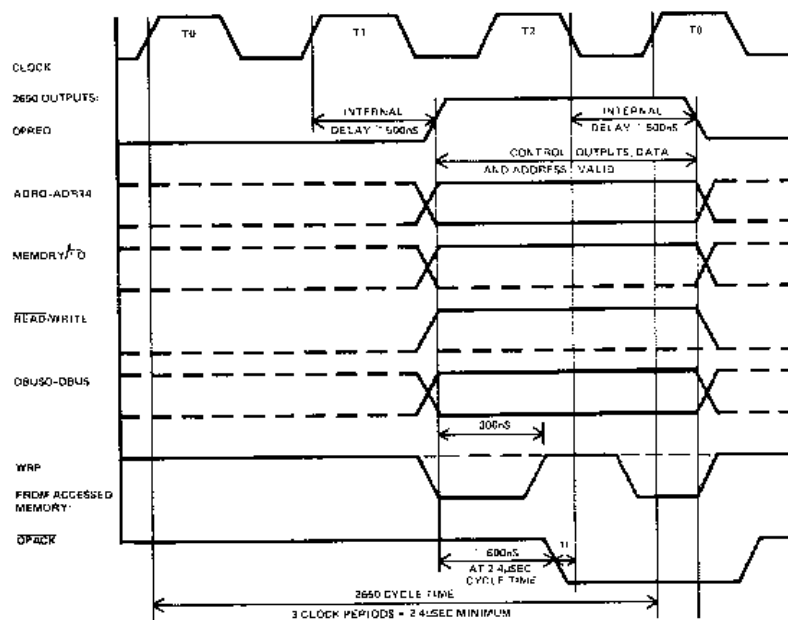


NOTES. (1) $\overline{\text{OPACK}}$ must go low at least 100nS before the trailing edge of T2 in order not to slow down the 2650.

Figure 11. MEMORY WRITE SEQUENCE

### INPUT/OUTPUT TIMING

The signal exchanges for I/O with external devices is very similar to the signaling for memory read/write. See the Features Section, INPUT/OUTPUT FACILITIES.

### CRITICAL TIMES

Figure 9 describes the timing relationship between the various interface signals. The critical times are labeled and defined in the table of AC characteristics.

# ELECTRICAL CHARACTERISTICS

## PRELIMINARY AC CHARACTERISTICS

$T_A = 0°C$ to $70°C$  $V_{CC} = 5V \pm 5\%$ unless otherwise specified, see notes 1,2,3 & 4.

| SYMBOL | PARAMETER | LIMITS | | UNITS |
| --- | --- | --- | --- | --- |
| | | MIN | MAX | |
| $t_{CH}$ | Clock High Phase | 400 | 10,000 | nsec |
| $t_{CL}$ | Clock Low Phase | 400 | $\infty$ | nsec |
| $t_{CP}$ | Clock Period | 800 | $\infty$ | nsec |
| $t_{PC}$[6] | Processor Cycle Time | 2,400 | $\infty$ | nsec |
| $t_{OR}$ | OPREQ Pulse Width | $2t_{CH} + t_{CL} - 100$ | $\infty$ | nsec |
| $t_{COR}$ | Clock to OPREQ Time | 100 | 700 | nsec |
| $t_{OAD}$[7] | $\overline{OPACK}$ Delay Time | 0 | $\infty$ | nsec |
| $t_{OAH}$ | $\overline{OPACK}$ Hold Time | 0 | $\infty$ | nsec |
| $t_{CSA}$ | Control Signal Available | 50 | | nsec |
| $t_{DOA}$ | Data Out Available | 50 | | nsec |
| $t_{DID}$[8] | Data in Delay | 0 | 1000(8) | nsec |
| $t_{DIH}$[9] | Data in Hold | 150 | | nsec |
| $t_{WPD}$ | Write Pulse Delay | $t_{CL} - 100$ | $t_{CL} - 50$ | nsec |
| $t_{WPW}$ | Write Pulse Width | $t_{CL}$ | $t_{CL}$ | nsec |
| $t_{ABD}$ | Address Bus Delay | | 80 | nsec |
| $t_{DBD}$ | Data Bus Delay | | 120 | nsec |
| $t_{IRS}$[10] | $\overline{INTREQ}$ Set up Time | 0 | | nsec |
| $t_{IRH}$[10] | $\overline{INTREQ}$ Hold Time | 0 | | nsec |
| $t_{ORT}$[5] | Output Buffer Rise Time | | 150 | nsec |

NOTES ON AC CHARACTERISTICS
1. See preceding timing diagrams for definition of timing terms.
2. Input levels swing between 0.65 volt and 2.2 volts.
3. Input signal transition times are 20ns.
4. Timing reference level is 1.5 volts.
5. Load is $-100\mu A$ at 20pF.
6. A Processor Cycle time consists of three clock periods.
7. In order to avoid slowing down the processor, $\overline{OPACK}$ must be lowered 100ns before the trailing edge of
   T2 clock, if $\overline{OPACK}$ is delayed past this point, the processor will wait in the T2 state and sample $\overline{OPACK}$
   on each subsequent negative clock edge until $\overline{OPACK}$ is lowered.
8. In order to avoid slowing the processor down, input data must be returned to the processor in $1\mu s$ or
   less time from the OPREQ edge, at a cycle time of $2.4\mu s$.
9. Input data must be held until 50ns after OPREQ falls.
10. In order to interrupt the current instruction, $\overline{INTREQ}$ must fall prior to the first clock of the last cycle
    of the current instruction. $\overline{INTREQ}$ must remain low until INTACK goes high.

## MAXIMUM GUARANTEED RATINGS[1]

| | |
|---|---|
| Operating Ambient Temperature | $0^\circ C$ to $+70^\circ C$ |
| Storage Temperature | $-65^\circ C$ to $+150^\circ C$ |
| All Input, Output, and Supply Voltages with respect to ground pin[3] | $-0.5V$ to $+6V$ |
| Package Power Dissipation[2]=1W Pkg. | 1.6W |

## PRELIMINARY 2650 DC ELECTRICAL CHARACTERISTICS

| SYMBOL | PARAMETER | TEST CONDITIONS | LIMITS | | UNIT |
|---|---|---|---|---|---|
| | | | MIN | MAX | |
| $I_{LI}$ | Input Load Current | $V_{IN} = 0$ to $5.25V$ | | 10 | $\mu A$ |
| $I_{LOH}$ | Output Leakage Current | ADREN, DBUSEN = 2.2V, $V_{OUT} = 4V$ | | 10 | $\mu A$ |
| $I_{LOL}$ | Output Leakage Current | ADREN, DBUSEN = 2.2V, $V_{OUT} = 0.45V$ | | 10 | $\mu A$ |
| $I_{CC}$ | Power Supply Current | $V_{CC} = 5.25V$, $T_A = 0^\circ C$ | | 100 | mA |
| $V_{IL}$ | Input Low | | -0.6 | 0.8 | V |
| $V_{IH}$ | Input High | | 2.2 | $V_{CC}$ | V |
| $V_{OL}$ | Output Low | $I_{OL} = 1.6$ mA | 0.0 | 0.45 | V |
| $V_{OH}$ | Output High | $I_{OH} = -100 \mu A$ | 2.4 | $V_{CC}-0.5$ | V |
| $C_{IN}$ | Input Capacitance | $V_{IN} = 0V$ | | 10 | pF |
| $C_{OUT}$ | Output Capacitance | $V_{OUT} = 0V$ | | 10 | pF |

Conditions: $T_A = 0^\circ C$ to $70^\circ C$, $V_{CC} = 5V \pm 5\%$

NOTES:
1. Stresses above those listed under "Maximum Guaranteed Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or at any other condition above those indicated in the operation sections of this specification is not implied.
2. For operating at elevated temperatures the device must be derated based on a $+150^\circ C$ maximum junction temperature and a thermal resistance of $50^\circ C/W$ junction to ambient (40 pin 1W package).
3. This product includes circuitry specifically designed for the protection of its internal devices from the damaging effects of excessive static charge. Nonetheless, it is suggested that conventional precautions be taken to avoid applying any voltages larger than the rated maxima.
4. Parameter valid over operating temperature range unless otherwise specified.
5. All voltage measurements are referenced to ground.
6. Manufacturer reserves the right to make design and process changes and improvements.
7. Typical values are at $+25^\circ C$, nominal supply voltages, and nominal processing parameters.

## INTERFACE SIGNALS

| TYPE | PINS | ABBREVIATION | FUNCTION | SIGNAL SENSE |
|---|---|---|---|---|
| INPUT | 1 | GND | Ground | GND=0 |
| INPUT | 1 | Vcc | +5 Volts ±5% | Vcc=1 |
| INPUT | 1 | RESET | Chip Reset | RESET=1 (pulse), causes reset |
| INPUT | 1 | CLOCK | Chip Clock | |
| INPUT | 1 | PAUSE | Temp. Halt execution | PAUSE=0, temporarily halts execution |
| INPUT | 1 | INTREQ | Interrupt Request | INTREQ=0, requests interrupt |
| INPUT | 1 | OPACK | Operation Acknowledge | OPACK=0, acknowledges operation |
| INPUT | 1 | SENSE | Sense | SENSE=0 (low) or SENSE=1 (high) |
| INPUT | 1 | ADREN | Address Enable | ADREN=1 drives into third state |
| INPUT | 1 | DBUSEN | Data Bus Enable | DBUSEN=1 drives into third state |
| IN/OUT | 8 | DBUS0-DBUS7 | Data Bus | DBUSn=0 (low), DBUSn=1 (high) |
| OUTPUT | 13 | ADR0-ADR12 | Address 0 through 12 | ADRn=0 (low), ADRn=1 (high) |
| OUTPUT | 1 | ADR13 or E/NE | Address 13 or Extended/Non-Extended | Non-Extended=0, Extended=1 |
| OUTPUT | 1 | ADR14 or D/C | Address 14 or Data Control | Control=0, Data 1 |
| OUTPUT | 1 | OPREQ | Operation Request | OPREQ=1, requests operation |
| OUTPUT | 1 | M/IO | Memory/IO | IO=0, M=1 |
| OUTPUT | 1 | R/W | Read/Write | R=0, W=1 |
| OUTPUT | 1 | FLAG | Flag Output | FLAG=1 (high), FLAG 0 (low) |
| OUTPUT | 1 | INTACK | Interrupt Acknowledge | INTACK=1, acknowledges interrupt |
| OUTPUT | 1 | RUN/WAIT | Run/Wait Indicator | RUN=1, WAIT=0 |
| OUTPUT | 1 | WRP | Write Pulse | WRP=1 (pulse), causes writing |

## PIN CONFIGURATION

| | | | |
|---|---|---|---|
| SENSE | 1 | 40 | FLAG |
| ADR 12 | 2 | 39 | Vcc |
| ADR 11 | 3 | 38 | CLOCK |
| ADR 10 | 4 | 37 | PAUSE |
| ADR 9 | 5 | 36 | OPACK |
| ADR 8 | 6 | 35 | RUN/WAIT |
| ADR 7 | 7 | 34 | INTACK |
| ADR 6 | 8 | 33 | DBUS 0 |
| ADR 5 | 9 | 2650 | 32 | DBUS 1 |
| ADR 4 | 10 | 31 | DBUS 2 |
| ADR3 | 11 | 30 | DBUS 3 |
| ADR 2 | 12 | 29 | DBUS 4 |
| ADR 1 | 13 | 28 | DBUS 5 |
| ADR 0 | 14 | 27 | DBUS 6 |
| ADREN | 15 | 26 | DBUS 7 |
| RESET | 16 | 25 | DBUSEN |
| INTREQ | 17 | 24 | OPREQ |
| ADR 14-D/C | 18 | 23 | R/W |
| ADR 13-E/NE | 19 | 22 | WRP |
| M/IO | 20 | 21 | GND |

TOP VIEW

39

# FEATURES

## INPUT/OUTPUT FACILITIES

The 2650 processor provides several mechanisms for performing input/output functions. They are flag and sense, non-extended I/O instructions, extended I/O instructions, and memory I/O. These four facilities are described below.

### FLAG & SENSE I/O

The 2650 has the ability to directly output one bit of data without additional address decoding or synchronizing signals.

The bit labeled "Flag" in the Program Status Word is connected through a TTL compatible driver to the chip output at pin #40. The Flag output always reflects the value in the Flag bit.

When a program changes the Flag bit through execution of an LPSU, PPSU, or CPSU, the bit will be set or cleared during the last cycle of the instruction that changes it.

The Flag bit may be used conveniently for many different purposes. The following is a list of some possible uses:

1. A serial output channel
2. An additional address bit to increase addressing range.
3. A switch or toggle output to control external logic.
4. The origin of a pulse for polling chains of devices.

The Sense bit performs the complementary function of the Flag and is a single bit direct input to the 2650. The Sense input, pin #1 is connected to a TTL compatible receiver and is then routed directly to a bit position in the Program Status Word. The bit in the PSW always represents the value of the external signal. It may be sampled anytime through use of the TPSU or SPSU instructions.

This simple input to the processor may be used in many ways. The following is a list of some possible uses:

1. A serial input channel
2. A sense switch input
3. A break signal to a processing program
4. An input for yes/no signaling from external devices.

### NON-EXTENDED I/O

There are four one byte I/O instructions; REDC, REDD, WRTC, and WRTD. They are all referred to as non-extended because they can communicate only one byte of data, either into or out of the 2650.

REDC and REDD causes the input transfer of one byte of data. They are identical except for the fact that the $D/\overline{C}$ Signal is in the D state for REDD and in the $\overline{C}$ state for REDC. Similarly, the instructions WRTC and WRTD cause an output transfer of one byte of data. The $D/\overline{C}$ line discriminates between the two pairs of input/output instructions. The $D/\overline{C}$ line can be used as a 1-bit device address in simple systems.

The read and write timing sequences for the one byte I/O instructions are the same as the memory read and write sequences with the following exceptions: the $M/\overline{IO}$ signal is switched to $\overline{IO}$, the $D/\overline{C}$ line becomes valid, $E/\overline{NE}$ is switched to $\overline{NE}$ (non-extended), and the Address bus contains no valid information.

The $\overline{\text{NE}}$ signal informs the devices outside the 2650 that a one byte I/O instruction is being executed. The D/$\overline{\text{C}}$ line indicates which pair of the one byte I/O instructions are being executed; D implies either WRTD or REDD, and $\overline{\text{C}}$ implies either WRTC or REDC. Finally, to determine whether it is a read or a write, examine the $\overline{\text{R}}$/W signal level.

Table 6 illustrates the sense of the interface signals. The "Signal Timing" section should be referenced for the exact timing relationships. It should be remembered that the control signals are not to be considered valid except when the OPREQ signal is valid.

### TABLE 6. I/O INTERFACE SIGNALS

|  | OPREQ | M/$\overline{\text{IO}}$ | $\overline{\text{R}}$/W | ADR13-E/$\overline{\text{NE}}$ | ADR14-D/$\overline{\text{C}}$ |
|---|---|---|---|---|---|
| MEMORY READ | T | M | $\overline{\text{R}}$ | ADR13 | ADR14 |
| MEMORY WRITE | T | M | W | ADR13 | ADR14 |
| 2 BYTE READ | T | $\overline{\text{IO}}$ | $\overline{\text{R}}$ | E | Don't Care |
| 2 BYTE WRITE | T | $\overline{\text{IO}}$ | W | E | Don't Care |
| 1 BYTE CONTROL READ | T | $\overline{\text{IO}}$ | $\overline{\text{R}}$ | $\overline{\text{NE}}$ | $\overline{\text{C}}$ |
| 1 BYTE CONTROL WRITE | T | $\overline{\text{IO}}$ | W | $\overline{\text{NE}}$ | $\overline{\text{C}}$ |
| 1 BYTE DATA READ | T | $\overline{\text{IO}}$ | $\overline{\text{R}}$ | $\overline{\text{NE}}$ | D |
| 1 BYTE DATA READ | T | $\overline{\text{IO}}$ | W | $\overline{\text{NE}}$ | D |

### EXTENDED I/O

There are two, two byte I/O instructions; REDE and WRTE. They are referred to as extended because they can communicate two bytes of data when they are executed. The REDE causes the second byte of the instruction to be output on the low order address lines, ADR0-ADR7, which is intended to be used as a device address while the byte of data then on the Data Bus will be strobed into the register specified in the instruction. The WRTE also presents the second byte of the instruction on the Address Bus, but a byte of data from the register specified in the instruction is simultaneously output on the Data Bus.

The two byte I/O instructions are similar to the one byte I/O instructions except: the D/$\overline{\text{C}}$ line is not considered, and the data from the second byte of the I/O instruction appears on the Address Bus all during the time that OPREQ is valid. The data on the Address Bus is intended to convey a device address, but may be utilized for any purpose.

Table 6 illustrates the sense of the interface signals for extended I/O instructions. Refer to "Signal Timing" section for exact timing relationships.

### MEMORY I/O

The 2650 user may choose to transfer data into or out of the processor using the memory control signals. The advantage to this technique is that the data can be read or written by the program through ordinary instruction execution and data may be directly operated upon with the arithmetic instructions.

To make use of this technique, the designer has to assign memory addresses to devices and design the device interfaces to generate the same signals as memory.

A disadvantage to this method is that it may be necessary to decode more address lines to determine the device address than with other I/O facilities.

# INTERRUPT MECHANISM

The 2650 has been implemented with a conventional, single level, address vectoring interrupt mechanism. There is one interrupt input pin. When an external device generates an interrupt signal ($\overline{\text{INTREQ}}$), the processor is forced to transfer control to any of 128 possible memory locations as determined by an 8-bit vector supplied by the interrupting device.

Of special interest is that the device may return a relative indirect address signal which causes the processor to enter an indirect addressing sequence upon receipt of an interrupt. This enables a device to direct the processor to execute code anywhere within addressable memory.

Upon recognizing the interrupt signal, the processor automatically sets the Interrupt Inhibit bit in the Program Status Word. This inhibits further interrupts from being recognized until the interrupt routine is finished executing and a Return-and-Enable instruction is executed or the inhibit bit is explicitly cleared.

When the inhibit bit in the PSW is set, the processor will not recognize an interrupt input. The Interrupt Inhibit bit may be set under program control (LPSU, PPSU) and is automatically set whenever the processor accepts an interrupt. The inhibit bit may be cleared in three ways:

1. By a RESET operation
2. By execution of an appropriate clear or load PSU instruction; (CPSU, LPSU)
3. By execution of a Return-and-Enable instruction.

The sequence of events for a normal interrupt operation is as follows:

1. An executing program enables interrupts.
2. External device initiates interrupt with the $\overline{\text{INTREQ}}$ line.
3. Processor finishes executing current instruction.
4. Processor sets inhibit bit.
5. Processor inserts the first byte of ZBSR (Zero Branch-to-Subroutine, Relative) instruction into the instruction register instead of what would have been the next sequential instruction.
6. Processor accesses the data bus to fetch the second byte of the ZBSR instruction.
7. Interrupting device responds to the Processor generated INTACK (Interrupt Acknowledge) by supplying the requested second byte.
8. The processor executes the Zero Branch-to-Subroutine instruction, saving the address of the instruction following the interrupted instruction in the RAS, and proceeds to execute the instruction at page 0, byte 0, or the address relative to page 0, byte 0 as given by the interrupting device.
9. When the interrupt routine is complete, a return instruction (RETC, RETE) pulls the address from the RAS and execution of the interrupted program resumes.

Since the interrupting device specifies the interrupt subroutine address in the standard relative address format, it has considerable flexibility with regard to the interrupt procedure. It can point to any location that is within +63 or −64 bytes of page zero, byte zero of memory. (Negative relative addresses wrap around the memory, modulo $8,192_{10}$ bytes.) The interrupting device also may specify whether the subroutine address is direct or indirect by providing a zero or one to DBUS #7 (pin #26). If the external device is not complex enough to exercise these options, it may respond to the INTACK operation with a byte of all zeroes. In such a case, the processor will execute a direct Branch-to-Subroutine to page zero, byte zero of memory.

The timing diagram in Figure 12 will help explain how the interrupt system works in the processor. The execution of the instruction labeled "A" has been proceeding before the start of this diagram. The last cycle of instruction "A" is shown. Notice that, as in all external operations, the OPREQ output eventually causes an $\overline{OPACK}$ input, which in turn allows OPREQ to be turned off. The arrows show this sequence of events. The last cycle of instruction "A" fetches the first byte of instruction "B" from Memory and inserts it into the Instruction Register.

Assume that instruction "B" is a two cycle, two byte instruction with no operand fetch (e.g., ADDI). Since the first byte has already been fetched by instruction "A", the first cycle of instruction "B" is used to fetch the second byte of instruction "B". Had instruction "B" not been interrupted, it would have fetched the first byte of the next sequential instruction during its second (last) cycle. The dotted lines indicate that operation.

Since instruction "B" is interrupted, however, the last cycle of "B" is used to insert the interrupt instruction (ZBSR) into the instruction register. Notice that the $\overline{INTREQ}$ input can arrive at any time. Instruction B is interrupted since INTREQ occured prior to the last (2nd) cycle of execution.

Instead of being the next sequential instruction following "B", instruction "C" is the completion of the interrupt. The first cycle of "C" is used to fetch the second byte of the ZBSR instruction from the DBUS as provided by the interrupting device. This fact is indicated by the presence of the INTACK control signal. The $\overline{INTREQ}$ may then be removed. When the device responds with the requested byte, it uses a standard operation acknowledge procedure $\overline{(OPACK)}$ to so indicate to the processor. During the second cycle of instruction "C" the processor executes the ZBSR instruction, and fetches the first byte of instruction "D" which is located at the subroutine address.
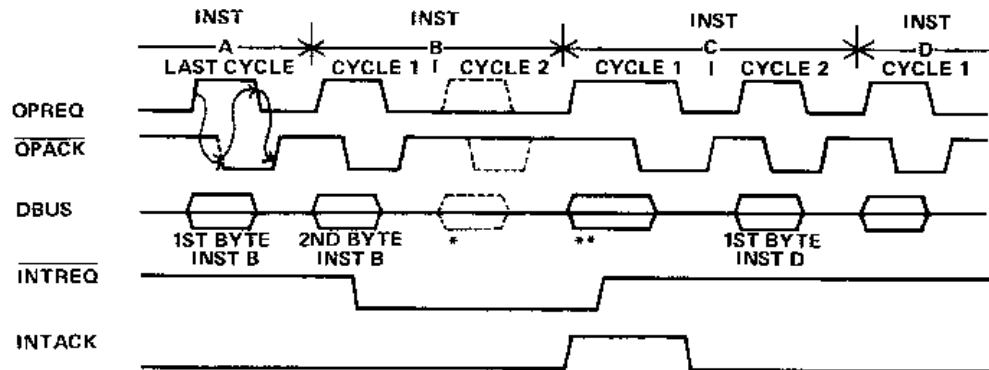


* PROCESSOR INSERTS 1ST BYTE OF ZBSR INSTRUCTION. ADDRESS OF 1ST BYTE OF INSTC IS PUSHED INTO RETURN ADDRESS STACK.

** 2ND BYTE OF ZBSR (INTERRUPT VECTOR)

Figure 12. INTERRUPT TIMING

## SUBROUTINE LINKAGE

The on-chip stack, along with the Branch-to-Subroutine and Return instructions provide the facility to transfer control to a subroutine. The subroutine can return control to the program that branched to it via a Return instruction.

The stack is eight levels deep which means that a routine may branch to a subroutine, which may branch to another subroutine, etc., eight times before any Return instructions are executed.

When designing a system that utilizes interrupts, it should be remembered that the processor jams a ZBSR into the IR and then executes it. This will cause an entry to be pushed into the on-chip stack like any other Branch-to-Subroutine instruction and may limit the stack depth available in certain programs.

When branching to a subroutine, the following sequence of events occurs:
1. The address in the IAR is used to fetch the Branch-to-Subroutine instruction and is then incremented in the Address Adder so that it points to the instruction following the subroutine branch.
2. The Stack Pointer is incremented by one so that it points to the next Return Address Stack location.
3. The contents of the IAR are stored in the stack at the location designated by the Stack Pointer.
4. The operand address contained in the Branch-to-Subroutine instruction (the address of the first instruction of the subroutine) is inserted into the IAR.

When returning from a subroutine, this sequence of events occurs:
1. The address in the IAR is used to fetch the return (RETC, RETE) instruction from memory.
2. When the return instruction is recognized by the processor, the contents of the stack entry pointed to by the Stack Pointer is placed into the IAR.
3. The Stack Pointer is decremented by one.
4. Instruction execution continues at the address now in the IAR.

## CONDITION CODE USAGE

The two-bit register, called the Condition Code, is incorporated in the Program Status Word. It may be seen in the description of the 2650 instructions, that the Condition Code (CC) is specifically set by every instruction that causes data to be transferred into a general purpose register and it is also set by compare instructions.

The reason for this design feature is that after an instruction executes, the CC contains a modest amount of information about the byte of data which has just been manipulated. For example, a program loads register one with a byte of unknown data and the Condition Code setting indicates that the byte is positive, negative or zero. The negative indication implies that bit #7 is set to one.

Consequently, a data manipulation operation when followed by a conditional branch is often sufficient to determine desired information without resorting to a specific test, thus saving instructions and memory space.

In the following example, the Condition Code is used to test the parity of a byte of data which is stored at symbolic memory location CHAR.

| | | | |
|------|--------|-------|------|
| EQ | EQU | 0 | THE EQUAL CONDITION CODE |
| CHAR | DATA | 2 | UNKNOWN DATA BYTE |
| WC | EQU | H'04' | THE WITH CARRY BIT |
| NEG | EQU | 2 | CC MASK |
| | CPSL | WC | CLEAR CARRY BIT |
| | LODI,R2 | –8 | SET UP COUNTER |
| | SUBZ | R0 | CLEAR REG 0 |
| | LODR,R1 | CHAR | GET THE CHARACTER (cc is set) |
| LOOP | BCFR,NEG | G01 | IF NOT SET, DON'T COUNT (cc is tested) |
| | ADDI,R0 | +1 | COUNT THE BIT |
| G01 | RRL,R1 | | MOVE BITS LEFT (cc is set) |
| | BIRR,R2 | LOOP | LOOP TILL DONE |

```
*       FINISHED,TEST IF REG 0 HAS A ONE IN LOW ORDER
*       IF BIT #0 = 1, ODD PARITY. IF BIT #0 = 0, THEN EVEN.
```

| | | |
|------|--------|-------|
| | TMI,R0 | H'01' |
| | BCTR,EQ | ODD |
| EVEN | HALT | |
| ODD | HALT | |

## START-UP PROCEDURE

The 2650 processor, having no internal start-up procedure must be started in an orderly fashion to assure that the internal control logic begins in a known state.

Assuming power is applied to the chip and the clock input is running, the easiest way to start is to apply a Reset signal for at least three clock periods. When the RESET signal is removed the processor will fetch the instruction at page 0, byte 0 and commence ordinary instruction execution.

To start processing at a specific address, a more complex start-up procedure may be employed. If an Interrupt signal is applied initially along with the Reset, processing will commence at the address provided by the interrupting device. Recall that the address provided may include a bit to specify indirect addressing and therefore the first instruction executed may be anywhere within addressable memory. The Reset and Interrupt signal may be applied simultaneously and when the Reset is removed, the processor will execute the usual interrupt signal sequence as described in INTERRUPT MECHANISM. There is an example of a start-up technique in the System Application Notes.

# INSTRUCTIONS

## ADDRESSING MODES

An addressing mode is a method the processor uses for developing argument addresses for machine instructions.

The 2650 processor can develop addresses in eight ways:

- Register addressing
- Immediate addressing
- Relative addressing
- Relative, indirect addressing
- Absolute addressing
- Absolute, indirect addressing
- Absolute, indexed addressing
- Absolute, indirect, indexed addressing

However, of these eight addressing modes, only four of them are basic. The others are variations due to indexing and indirection. The basic addressing mode of each instruction is indicated in parentheses in the first line of each detailed instruction description. The following text describes how effective addresses are developed by the processor.

### REGISTER ADDRESSING

All register-to-register instructions are one byte in length. Instructions utilizing this addressing mode appear in this general format.

Operation Code Register

```
| | | | | | | |
 7 6 5 4 3 2 1 0
```
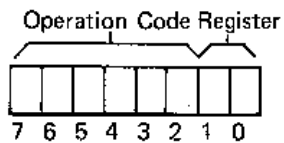
Since there are only two bits designated to specify a register, register zero always contains one of the operands while the other operand is in one of the three registers in the currently selected bank. Register zero may also be specified as the explicit operand giving instructions such as: LODZ    R0.

In one byte register addressing instructions which have just one operand, any of the currently selected general purpose registers or register zero may be specified, e.g., RRL,R0.

### IMMEDIATE ADDRESSING

All immediate addressing instructions are two bytes in length. The first byte contains the operation code and register designation, while the second byte contains data used as the argument during instruction execution.

Operation Code Register       Two's complement binary number
                              or 8-bit logic mask

```
| | | | | | | |    | | | | | | | |
 7 6 5 4 3 2 1 0    7 6 5 4 3 2 1 0
      Byte 0              Byte 1
```
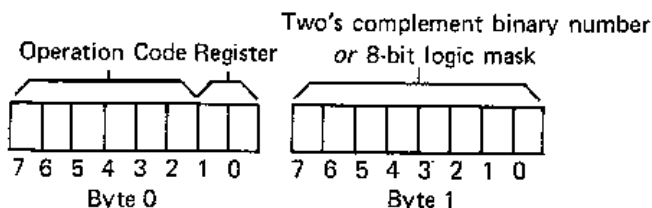
The second byte, the data byte, may contain a binary number or a logic mask depending on the particular instruction being executed. Any register may be designated in the first byte.

## RELATIVE ADDRESSING

Relative addressing instructions are all two bytes in length and are memory reference instructions. One argument of the instruction is a register and the other argument is the contents of a memory location. The format of relative addressing instructions is:
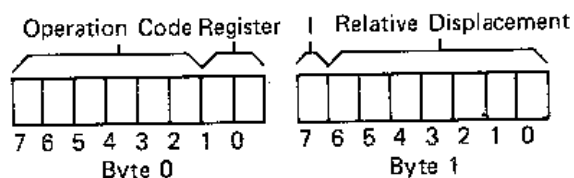


The first byte contains the operation code and register designation, while the second byte contains the relative address. Bits 0—6, byte 1, contain a 7-bit two's complement binary number which can range from −64 to +63. This number is used by the processor to calculate the effective address. The effective address is calculated by adding the address of the first byte following a relative addressing instruction to the relative displacement in the second byte of the instruction.

If bit 7, byte 1 is set to "1", the processor will enter an indirect addressing cycle, where the actual operand address will be accessed from the effective address location. See Indirect Addressing.

Two of the branch instructions (ZBSR, ZBRR) allow addressing relative to page zero, byte 0 of memory. In this case, values up to +63 reference the first 63 bytes of page zero and values up to −64 reference the last 64 bytes of page zero.

## ABSOLUTE ADDRESSING FOR NON-BRANCH INSTRUCTIONS

Absolute addressing instructions are all three bytes in length and are memory reference instructions. One argument of the instruction is a register, designated in bits 1 and 0, byte 0; the other argument is the contents of a memory location. The format of absolute addressing instructions is:
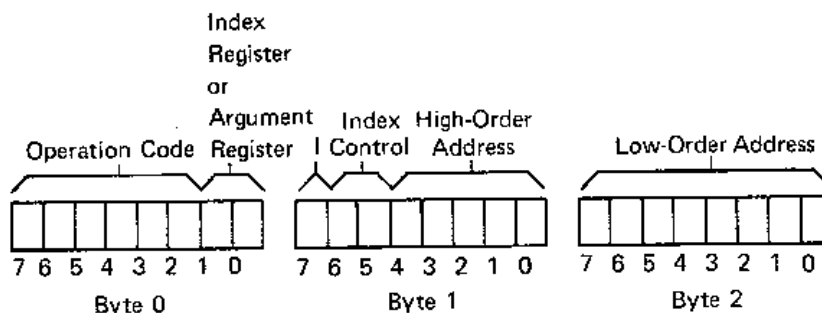


Bits 4—0, byte 1 and 7—0, byte 2 contain the absolute address and can address any byte within the same page that the instruction appears.

The index control bits, bits #6 and #5, byte 1 determine how the effective address will be calculated and possibly which register will be the argument during instruction execution. The index control bits have the following interpretation:

| Index Control | | |
|---|---|---|
| Bit 6 | Bit 5 | Meaning |
| 0 | 0 | Non-indexed address |
| 0 | 1 | Indexed with auto-increment |
| 1 | 0 | Indexed with auto-decrement |
| 1 | 1 | Indexed only |

When the index control bits are 0 & 0, bits #1 and #0 in byte 0 contain the argument register designation and bits 0 to 4, byte 1 and bits 0 to 7, byte 2 contain the effective address. Indirect addressing may be specified by setting bit #7, byte 1 to a one.

When the index control bits are 1 & 1, bits #1 and #0 in byte 0 designate the index register and the argument register implicitly becomes register zero. The effective address is calculated by adding the contents of the index register (8-bit absolute integer) to the address field. If indirect addressing is specified, the indirect address is accessed and then the value in the index register is added to the indirect address. This is commonly called post indexing.

When the index control bits contain 0 & 1, the address is calculated by the processor exactly as when the control bits contain 1 & 1 *except* a binary 1 is added to the contents of the selected index register *before* the calculation of the effective address proceeds. Similarly, when the index control bits contain 1 & 0, a binary 1 is subtracted from the contents of the selected index register *before* the effective address is calculated.

## ABSOLUTE ADDRESSING FOR BRANCH INSTRUCTIONS

The three byte, absolute addressing, branch instructions deviate slightly in format from ordinary absolute addressing instructions as shown below:
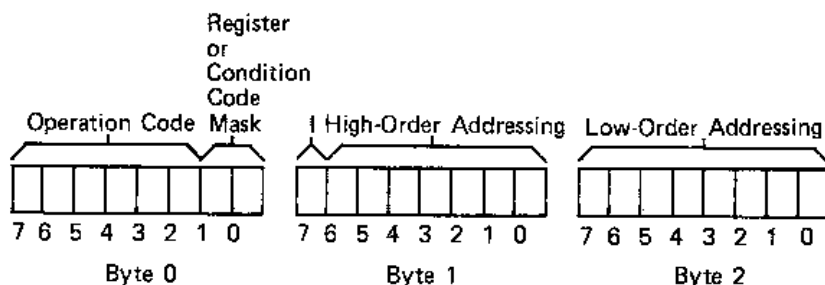


The notable difference is that bits 6 and 5, byte 1, are no longer interpreted as Index Control bits, but instead are interpreted as the high order bits of the address field. This means that there is no indexing allowed on most absolute addressing branch instructions, but indexed branches are possible through use of the BXA and BSXA instructions. The bits #6 and #5, byte 1, are used to set the current page register, thus enabling programs to directly transfer control to another page.

See the MEMORY ORGANIZATION, BXA and BSXA instructions, and INDIRECT ADDRESSING.

## INDIRECT ADDRESSING

Indirect addressing means that the argument address of an instruction is not specified by the instruction itself, but rather the argument address will be found in the two bytes pointed to by the address field or relative address field, of absolute or relative addressing instructions. In the case of absolute addressing, the value of the index register is added to the indirect address *not* to the value in the address field of the instruction. In both cases, the processor will enter the indirect addressing state when the bit designated "I" is set to one. Entering the indirect addressing sequence adds two cycles (6 clock periods) to the execution time of an instruction.

Indirect addresses are 15-bit addresses stored right justified in two contiguous bytes of memory. As such, an indirect address may specify any location in addressable memory (0—32,767). The high order bit of the two byte indirect address is not used by the processor.

Only single level indirect addressing is implemented. The following examples demonstrate indirect addressing.

**Example 1.**

| 0 0 0 0 1 1 1 0 | 1 0 0 0 0 0 0 0 | 0 1 0 1 0 0 0 1 | LODA,R2 | *H'51' |

Address $10_{16}$    $11_{16}$    $12_{16}$

| 0 0 0 0 0 0 0 1 | 0 0 1 0 1 0 0 0 | ACON | H'128' |

Address    $51_{16}$    $52_{16}$

| 0 1 1 0 0 1 1 1 | DATA | H'67' |

Address    $128_{16}$

The LODA instruction in memory locations 10, 11, and 12 specifies indirect addressing (bit 7, byte 1, is set). Therefore, when the instruction is executed, the processor takes the address field value, H'51', and uses it to access the two byte indirect address at 51 and 52. Then using the contents of 51 and 52 as the effective address, the data byte containing H'67' is loaded into register 2.

**Example 2.**

| 0 0 0 0 1 0 1 0 | 1 0 0 0 0 1 0 1 | | LODR,R2 | *H'17' |

Address $10_{16}$    $11_{16}$

| 0 0 0 0 0 0 0 1 | 0 0 1 0 1 0 0 0 | ACON | H'128' |

Address    $17_{16}$    $18_{16}$

| 0 1 1 0 0 1 1 1 | DATA | H'67' |

Address    $128_{16}$

In a fashion similar to the previous example, the relative address is used to access the indirect address which points to the data byte. When the LODR instruction is executed, the data byte contents, H'67', will be loaded into register 2.

## INSTRUCTIONS FORMAT EXCEPTIONS

There are several instructions which are detected by decoding the entire 8 bits of the first byte of the instruction. These instructions are unique and may be noticed in the instruction descriptions. Examples are: HALT, CPSU, CPSL.

Of this type of instruction, two operation codes were taken from otherwise complete sets thus eliminating certain possible operations. The cases are as follows:

(NOT OKAY)  STRZ  0 — Storing register zero into register zero is not implemented, the operation code is used for NOP (no operation).
(OKAY)      NOP

(NOT OKAY)  ANDZ  0 — AND of register zero with register zero is not implemented, the operation code is used for HALT.
(OKAY)      HALT
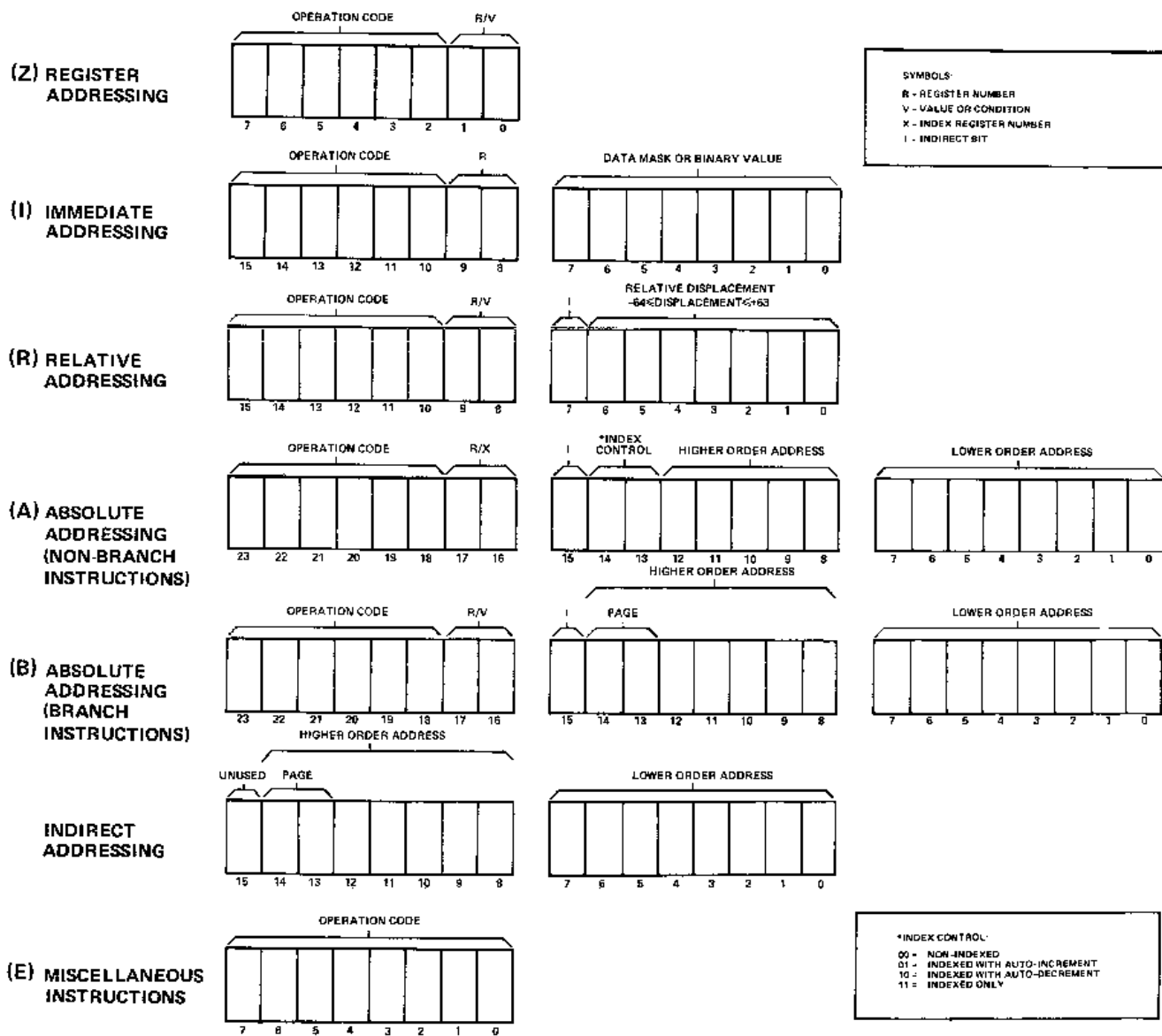


Figure 13. INSTRUCTION FORMATS

## LOAD REGISTER ZERO

(Register Addressing)

**Mnemonic**        LODZ             r

**Binary Coding**
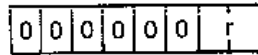
| 0 | 0 | 0 | 0 | 0 | 0 | r |
|---|---|---|---|---|---|---|

7 6 5 4 3 2 1 0

**Execution Time**       2 cycles (6 clock periods)

**Description**

This one-byte instruction transfers the contents of the specified register, r, into register zero. The previous contents of register zero are lost. The contents of register r remain unchanged.

When the specified register, r, equals 0, the operation code is changed to $60_{16}$ by the assembler. The instruction, 00000000, yields indeterminate results.

**Processor Registers Affected**        CC

**Condition Code Setting**

| Register Zero | CC1 | CC0 |
|---|---|---|
| Positive | 0 | 1 |
| Zero | 0 | 0 |
| Negative | 1 | 0 |

## LOAD IMMEDIATE

(Immediate Addressing)

**Mnemonic**        LODI,r            v

**Binary Coding**

| 0 | 0 | 0 | 0 | 0 | 1 | r |   | | | | | v | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

7 6 5 4 3 2 1 0      7 6 5 4 3 2 1 0

**Execution Time**       2 cycles (6 clock periods)

**Description**

This two-byte instruction transfers the second byte of the instruction, v, into the specified register, r. The previous contents of r are lost.

**Processor Registers Affected**        CC

**Condition Code Setting**

| Register r | CC1 | CC0 |
|---|---|---|
| Positive | 0 | 1 |
| Zero | 0 | 0 |
| Negative | 1 | 0 |

## LOAD RELATIVE

**Mnemonic**     LODR,r     (∗)a

**Binary Coding**
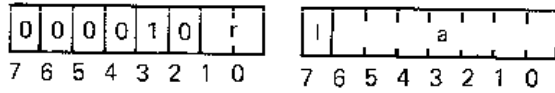
| 0 | 0 | 0 | 0 | 1 | 0 | r |
|---|---|---|---|---|---|---|
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

| I | a |
|---|---|
| 7 | 6 5 4 3 2 1 0 |

**Execution Time**     3 cycles (9 clock periods)

**Description**

This two-byte instruction transfers a byte of data from memory into the specified register, r. The data byte is found at the effective address formed by the addition of the a field and the address of the byte following this instruction. The previous contents of register r are lost. Indirect addressing may be specified.

**Processor Registers Affected**     CC

**Condition Code Setting**

| Register r | CC1 | CC0 |
|---|---|---|
| Positive | 0 | 1 |
| Zero | 0 | 0 |
| Negative | 1 | 0 |


## LOAD ABSOLUTE

**Mnemonic**     LODA,r     (∗)a(,X)

**Binary Coding**
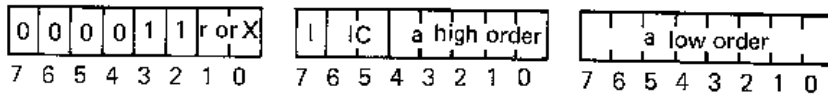
| 0 | 0 | 0 | 0 | 1 | 1 | r or X |
|---|---|---|---|---|---|---|
| 7 | 6 | 5 | 4 | 3 | 2 | 1 0 |

| I | IC | a high order |
|---|---|---|
| 7 | 6 5 | 4 3 2 1 0 |

| a low order |
|---|
| 7 6 5 4 3 2 1 0 |

**Execution Time**     4 cycles (12 clock periods)

**Description**

This three-byte instruction transfers a byte of data from memory into the specified register, r. The data byte is found at the effective address. If indexing is specified, bits 1 and 0, byte 0, indicate the index register and the destination of the operation implicitly becomes register zero. The previous contents of register r are lost.

Indirect addressing and/or indexing may be specified.

**Processor Registers Affected**     CC

**Condition Code Setting**

| Register r | CC1 | CC0 |
|---|---|---|
| Positive | 0 | 1 |
| Zero | 0 | 0 |
| Negative | 1 | 0 |

**Mnemonic**         STRZ                r

**Binary Code**
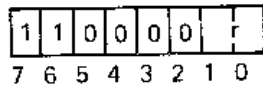
| 1 | 1 | 0 | 0 | 0 | 0 | r |
|---|---|---|---|---|---|---|
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

**Execution Time**      2 cycles (6 clock periods)

**Description**

This one-byte instruction transfers the contents of register zero into the specified register r. The previous contents of register r are lost. The contents of register zero remain unchanged.

**Note:**   Register r may not be specified as zero. This operation code, '11000000', is reserved for NOP.

**Processor Registers Affected**         CC

**Condition Code Setting**

| Register r | CC1 | CC0 |
|---|---|---|
| Positive | 0 | 1 |
| Zero | 0 | 0 |
| Negative | 1 | 0 |

---

**Mnemonic**         STRR,r                (*)a

**Binary Code**
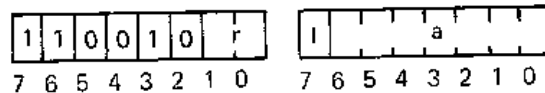
| 1 | 1 | 0 | 0 | 1 | 0 | r |  | I |  |  |  | a |  |  |  |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

**Execution Time**      3 cycles (9 clock periods)

**Description**

This two-byte instruction transfers a byte of data from the specified register, r, into the byte of memory pointed to by the effective address. The contents of register r remain unchanged and the contents of the memory byte are replaced.

Indirect addressing may be specified.

**Processor Registers Affected**         None

**Condition Code Setting**         N/A

## STORE ABSOLUTE

**Mnemonic**      STRA,r          (*)a(,X)

**Binary Code**
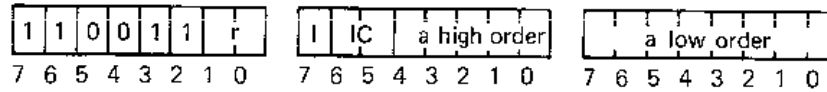
| 1 | 1 | 0 | 0 | 1 | 1 | r | | I | IC | a high order | | a low order |
|---|---|---|---|---|---|---|---|---|----|--------------|---|-------------|

```
7 6 5 4 3 2 1 0      7 6 5 4 3 2 1 0      7 6 5 4 3 2 1 0
```

**Execution Time**      4 cycles (12 clock periods)

**Description**

This three-byte instruction transfers a byte of data from the specified register, r, into the byte of memory pointed to by the effective address. The contents of register r remain unchanged and the contents of the memory byte are replaced.

Indirect addressing and/or indexing may be specified. If indexing is specified, bits 1 and 0, byte 0, indicate the index register and the destination of the operation implicitly becomes register zero.

**Processor Registers Affected**          None

**Condition Code Setting**          N/A

## ADD TO REGISTER ZERO

**Mnemonic**      ADDZ          r

**Binary Code**
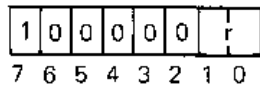
| 1 | 0 | 0 | 0 | 0 | 0 | r |
|---|---|---|---|---|---|---|

```
7 6 5 4 3 2 1 0
```

**Execution Time**      2 cycles (6 clock periods)

**Description**

This one-byte instruction causes the contents of the specified register, r, and the contents of register zero to be added together in a true binary adder. The 8-bit sum of the addition replaces the contents of register zero. The contents of register r remain unchanged.

**Note:**   Add with Carry may be effected. See Carry bit.

**Processor Registers Affected**          C, CC, IDC, OVF

**Condition Code Setting**

| Register Zero | CC1 | CC0 |
|---------------|-----|-----|
| Positive | 0 | 1 |
| Zero | 0 | 0 |
| Negative | 1 | 0 |

## ADD IMMEDIATE

**Mnemonic**        ADDI,r        v

**Binary Coding**
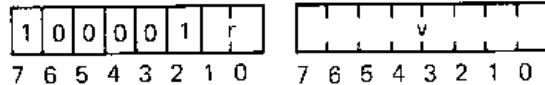
| 1 | 0 | 0 | 0 | 0 | 1 | r | | | | | | v | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 7 | 6 | 5 | 4 | 3 | 2 | 1 0 | | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

**Execution Time**      2 cycles (6 clock periods)

**Description**

This two-byte instruction causes the contents of register r and the contents of the second byte of this instruction to be added together in a true binary adder. The eight-bit sum replaces the contents of register r.

**Note:**    Add with Carry may be effected. See Carry bit.

**Processor Registers Affected**        C, CC, IDC, OVF

**Condition Code Setting**

| Register r | CC1 | CC0 |
|---|---|---|
| Positive | 0 | 1 |
| Zero | 0 | 0 |
| Negative | 1 | 0 |

---

## ADD RELATIVE

**Mnemonic**        ADDR,r        (*)a

**Binary Coding**
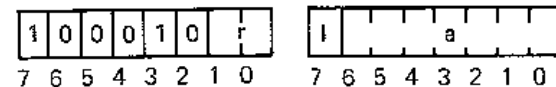
| 1 | 0 | 0 | 0 | 1 | 0 | r | | I | | | | a | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 7 | 6 | 5 | 4 | 3 | 2 | 1 0 | | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

**Execution Time**      3 cycles (9 clock periods)

**Description**

This two-byte instruction causes the contents of register r and the contents of the byte of memory pointed to by the effective address to be added together in a true binary adder. The eight-bit sum replaces the contents of register r.

Indirect addressing may be specified.

**Note:**    Add with Carry may be effected. See Carry bit.

**Processor Registers Affected**        C, CC, IDC, OVF

**Condition Code Setting**

| Register r | CC1 | CC0 |
|---|---|---|
| Positive | 0 | 1 |
| Zero | 0 | 0 |
| Negative | 1 | 0 |

**Mnemonic**          ADDA,r          (*)a(,X)

**Binary Coding**
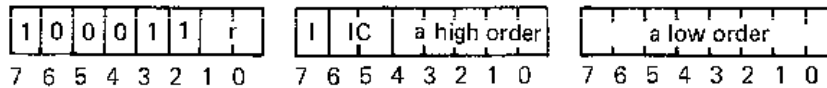
| 1 | 0 | 0 | 0 | 1 | 1 | r | | I | IC | a high order | | a low order |
|---|---|---|---|---|---|---|---|---|---|---|---|---|

```
7  6  5  4  3  2  1  0     7  6  5  4  3  2  1  0     7  6  5  4  3  2  1  0
```

**Execution Time**     4 cycles (12 clock periods)

**Description**

This three-byte instruction causes the contents of register r and the contents of the byte of memory pointed to by the effective address to be added together in a true binary adder. The eight-bit sum replaces the contents of register r.

Indirect addressing and/or indexing may be specified. If indexing is specified, bits 1 and 0, byte 0, indicate the index register and the destination of the operation implicitly becomes register zero.

**Note:**     Add with Carry may be effected. See Carry bit.

**Processor Registers Affected**          C, CC, IDC, OVF

**Condition Code Setting**

| Register r | CC1 | CC0 |
|---|---|---|
| Positive | 0 | 1 |
| Zero | 0 | 0 |
| Negative | 1 | 0 |

---

**SUBTRACT FROM REGISTER ZERO**                    (Register Addressing)

**Mnemonic**          SUBZ          r

**Binary Coding**
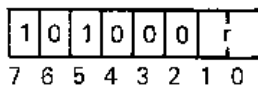
| 1 | 0 | 1 | 0 | 0 | 0 | r |
|---|---|---|---|---|---|---|

```
7  6  5  4  3  2  1  0
```

**Execution Time**     2 cycles (6 clock periods)

**Description**

This one-byte instruction causes the contents of the specified register r to be subtracted from the contents of register zero. The result of the subtraction replaces the contents of register zero.

The subtraction is performed by taking the binary two's complement of the contents of register r and adding that result to the contents of register zero. The contents of register r remain unchanged.

**Note:**     Subtract with Borrow may be effected. See Carry bit.

**Processor Registers Affected**          C, CC, IDC, OVF

**Condition Code Setting**

| Register Zero | CC1 | CC0 |
|---|---|---|
| Positive | 0 | 1 |
| Zero | 0 | 0 |
| Negative | 1 | 0 |

**Mnemonic**          SUBI,r              v

**Binary Code**
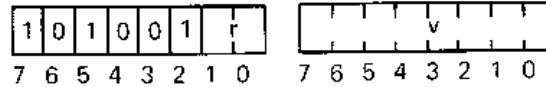
| 1 | 0 | 1 | 0 | 0 | 1 | r |
|---|---|---|---|---|---|---|
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

| | | | | v | | | |
|---|---|---|---|---|---|---|---|
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

**Execution Time**       2 cycles (6 clock periods)

**Description**

This two-byte instruction causes the contents of the second byte of this instruction to be subtracted from the contents of register r. The result of the subtraction replaces the contents of register r.

The subtraction is performed by taking the binary two's complement of the contents of the second instruction byte and adding that result to the contents of register r.

**Note:**   Subtract with Borrow may be effected. See Carry bit.

**Processor Registers Affected**          C, CC, IDC, OVF

**Condition Code Setting**

| Register r | CC1 | CC0 |
|---|---|---|
| Positive | 0 | 1 |
| Zero | 0 | 0 |
| Negative | 1 | 0 |

**Mnemonic**          SUBR,r              (∗)a

**Binary Code**
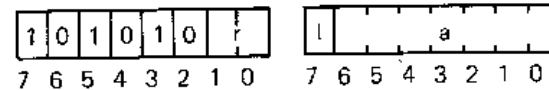
| 1 | 0 | 1 | 0 | 1 | 0 | r |
|---|---|---|---|---|---|---|
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

| I | | | a | | | | |
|---|---|---|---|---|---|---|---|
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

**Execution Time**       3 cycles (9 clock periods)

**Description**

This two-byte instruction causes the contents of the byte of memory pointed to by the effective address to be subtracted from the contents of register r. The result of the subtraction replaces the contents of register r.

The subtraction is performed by taking the binary two's complement of the contents of the byte of memory and adding that result to the contents of register r.

Indirect addressing may be specified.

**Note:**   Subtract with Borrow may be effected. See Carry bit.

**Processor Registers Affected**          C, CC, IDC, OVF

**Condition Code Setting**

| Register r | CC1 | CC0 |
|---|---|---|
| Positive | 0 | 1 |
| Zero | 0 | 0 |
| Negative | 1 | 0 |

## SUBTRACT ABSOLUTE <span style="float:right">(Absolute Addressing)</span>

**Mnemonic**     SUBA,r          (*)a(,X)

**Binary Code**
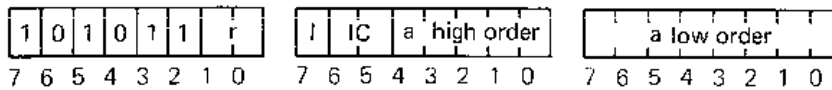
| 1 | 0 | 1 | 0 | 1 | 1 | r | | | I | IC | a high order | | | a low order | |
|---|---|---|---|---|---|---|---|---|---|----|-----|---|---|---|---|

7 6 5 4 3 2 1 0    7 6 5 4 3 2 1 0    7 6 5 4 3 2 1 0

**Execution Time**     4 cycles (12 clock periods)

**Description**

This three-byte instruction causes the contents of the byte of memory pointed to by the effective address to be subtracted from the contents of register r. The result of the subtraction replaces the contents of register r.

The subtraction is performed by taking the binary two's complement of the contents of the memory byte and adding that result to the contents of register r.

Indirect addressing and/or indexing may be specified. If indexing is specified, bits 1 and 0, byte 0, indicate the index register and the destination of the operation implicitly becomes register zero.

**Note:**     Subtract with Borrow may be effected. See Carry bit.

**Processor Registers Affected**     C, CC, IDC, OVF

**Condition Code Setting**

| Register r | CC1 | CC0 |
|------------|-----|-----|
| Positive   | 0   | 1   |
| Zero       | 0   | 0   |
| Negative   | 1   | 0   |

---

## AND TO REGISTER ZERO <span style="float:right">(Register Addressing)</span>

**Mnemonic**     ANDZ          r

**Binary Code**
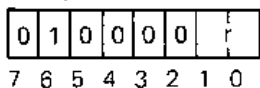
| 0 | 1 | 0 | 0 | 0 | 0 | r |
|---|---|---|---|---|---|---|

7 6 5 4 3 2 1 0

**Execution Time**     2 cycles (6 clock periods)

**Description**

This one-byte instruction causes the contents of the specified register, r, to be logically ANDed with the contents of register zero. The result of the operation replaces the contents of register zero. The contents of register r remain unchanged.

The AND operation treats each bit of the argument bytes as in the truth table below:

| Bit (0-7) | Bit (0-7) | AND Result |
|-----------|-----------|------------|
| 0         | 0         | 0          |
| 0         | 1         | 0          |
| 1         | 1         | 1          |
| 1         | 0         | 0          |

**Note:**     Register r may not be specified as zero. This operation code, '01000000', is reserved for HALT.

**Processor Registers Affected**     CC

**Condition Code Setting**

| Register Zero | CC1 | CC0 |
|---------------|-----|-----|
| Positive      | 0   | 1   |
| Zero          | 0   | 0   |
| Negative      | 1   | 0   |

## AND IMMEDIATE

**Mnemonic**         ANDI,r          v

**Binary Code**

| 0 | 1 | 0 | 0 | 0 | 1 | r | | | | | | | v | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |

**Execution Time**      2 cycles (6 clock periods)

**Description**

This two-byte instruction causes the contents of the specified register r to be logically ANDed with the contents of the second byte of this instruction. The result of this operation replaces the contents of register r.

The AND operation treats each bit of the argument bytes as in the truth table below:

| Bit (0-7) | Bit (0-7) | AND Result |
|-----------|-----------|------------|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 1 | 1 |
| 1 | 0 | 0 |

**Processor Registers Affected**      CC

**Condition Code Setting**

| Register Zero | CC1 | CC0 |
|---------------|-----|-----|
| Positive | 0 | 1 |
| Zero | 0 | 0 |
| Negative | 1 | 0 |

## AND RELATIVE

**Mnemonic**         ANDR,r          (*)a

**Binary Code**

| 0 | 1 | 0 | 0 | 1 | 0 | r | | l | | | | a | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |

**Execution Time**      3 cycles (9 clock periods)

**Description**

This two-byte instruction causes the contents of the specified register r to be logically ANDed with the contents of the memory byte pointed to by the effective address. The result of this operation replaces the contents of register r.

The AND operation treats each bit of the argument bytes as in the truth table below:

| Bit (0-7) | Bit (0-7) | AND Result |
|-----------|-----------|------------|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 1 | 1 |
| 1 | 0 | 0 |

**Processor Registers Affected**      CC

**Condition Code Setting**

| Register Zero | CC1 | CC0 |
|---------------|-----|-----|
| Positive | 0 | 1 |
| Zero | 0 | 0 |
| Negative | 1 | 0 |

**Mnemonic** ANDA,r (*)a(,X)

**Binary Code**
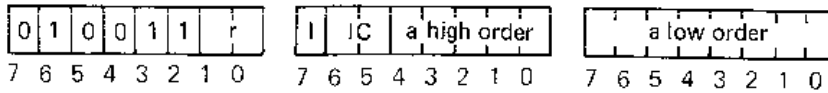
| 0 | 1 | 0 | 0 | 1 | 1 | r | | I | IC | a high order | | a low order | |
|---|---|---|---|---|---|---|---|---|----|--------------|---|-------------|---|

```
7 6 5 4 3 2 1 0    7 6 5 4 3 2 1 0    7 6 5 4 3 2 1 0
```

**Execution Time** 4 cycles (12 clock periods)

**Description**

This three-byte instruction causes the contents of Register r to be logically ANDed with the contents of memory byte pointed to by the effective address. The result of the operation replaces the contents of register r.

The AND operation treats each bit of the argument bytes as in the truth table below:

| Bit (0-7) | Bit (0-7) | AND Result |
|-----------|-----------|------------|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 1 | 1 |
| 1 | 0 | 0 |

Indirect addressing and/or indexing may be specified. If indexing is specified, bits 1 and 0, byte 0, indicate the index register and the destination of the operation implicitly becomes register zero.

**Processor Registers Affected** CC

**Condition Code Setting**

| Register Zero | CC1 | CC0 |
|---------------|-----|-----|
| Positive | 0 | 1 |
| Zero | 0 | 0 |
| Negative | 1 | 0 |

## INCLUSIVE OR TO REGISTER ZERO (Register Addressing)

**Mnemonic** IORZ r

**Binary Code**
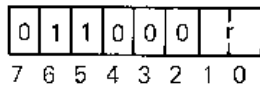
| 0 | 1 | 1 | 0 | 0 | 0 | r |
|---|---|---|---|---|---|---|

```
7 6 5 4 3 2 1 0
```

**Execution Time** 2 cycles (6 clock periods)

**Description**

This one-byte instruction causes the contents of the specified register, r, to be logically Inclusive ORed with the contents of register zero. The result of this operation replaces the contents of register zero. The contents of register r remain unchanged.

The Inclusive OR operation treats each bit of the argument bytes as in the truth table below:

| Bit (0-7) | Bit (0-7) | Inclusive OR Result |
|-----------|-----------|---------------------|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 1 | 1 |
| 1 | 0 | 1 |

**Processor Registers Affected** CC

**Condition Code Setting**

| Register Zero | CC1 | CC0 |
|---------------|-----|-----|
| Positive | 0 | 1 |
| Zero | 0 | 0 |
| Negative | 1 | 0 |

## INCLUSIVE OR IMMEDIATE

<div align="right">(Immediate Addressing)</div>

**Mnemonic**       IORI,r       v

**Binary Code**
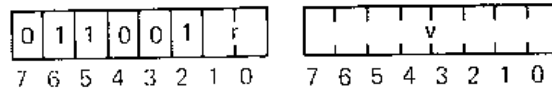
| 0 | 1 | 1 | 0 | 0 | 1 | r |
|---|---|---|---|---|---|---|
| 7 | 6 | 5 | 4 | 3 | 2 | 1 0 |

| | | | v | | | |
|---|---|---|---|---|---|---|
| 7 | 6 | 5 | 4 | 3 | 2 | 1 0 |

**Execution Time**       2 cycles (6 clock periods)

**Description**

This two-byte instruction causes the contents of the specified register r to be logically Inclusive ORed with the contents of the second byte of this instruction. The result of this operation replaces the contents of register r.

The Inclusive OR operation treats each bit of the argument bytes as in the truth table below:

| Bit (0-7) | Bit (0-7) | Inclusive OR Result |
|:---:|:---:|:---:|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 1 | 1 |
| 1 | 0 | 1 |

**Processor Registers Affected**       CC

**Condition Code Setting**

| Register r | CC1 | CC0 |
|---|:---:|:---:|
| Positive | 0 | 1 |
| Zero | 0 | 0 |
| Negative | 1 | 0 |

---

## INCLUSIVE OR RELATIVE

<div align="right">(Relative Addressing)</div>

**Mnemonic**       IORR,r       (*)a

**Binary Code**
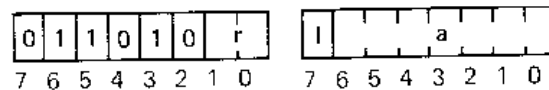
| 0 | 1 | 1 | 0 | 1 | 0 | r |
|---|---|---|---|---|---|---|
| 7 | 6 | 5 | 4 | 3 | 2 | 1 0 |

| I | | | a | | | |
|---|---|---|---|---|---|---|
| 7 | 6 | 5 | 4 | 3 | 2 | 1 0 |

**Execution Time**       3 cycles (9 clock periods)

**Description**

This two-byte instruction causes the contents of the specified register r to be logically Inclusive ORed with the contents of the memory byte pointed to by the effective address. The result of this operation replaces the previous contents of register r.

Indirect addressing may be specified.

The Inclusive OR operation treats each bit of the argument byte as in the truth table below:

| Bit (0-7) | Bit (0-7) | Inclusive OR Result |
|:---:|:---:|:---:|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 1 | 1 |
| 1 | 0 | 1 |

**Processor Registers Affected**       CC

**Condition Code Setting**

| Register r | CC1 | CC0 |
|---|:---:|:---:|
| Positive | 0 | 1 |
| Zero | 0 | 0 |
| Negative | 1 | 0 |

**Mnemonic**      IORA,r      (*)a(,X)

**Binary Code**
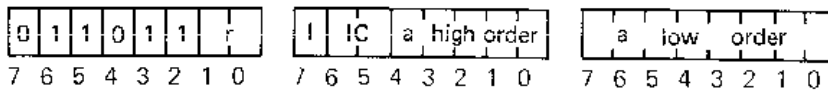
| 0 | 1 | 1 | 0 | 1 | 1 | r |
|---|---|---|---|---|---|---|

7 6 5 4 3 2 1 0

| I | IC | a high order |
|---|----|--------------|

7 6 5 4 3 2 1 0

| a | low order |
|---|-----------|

7 6 5 4 3 2 1 0

**Execution Time**      4 cycles (12 clock periods)

**Description**

This three-byte instruction causes the contents of register r to be logically Inclusive ORed with the contents of the memory byte pointed to by the effective address. The result of the operation replaces the previous contents of register r.

Indirect addressing and/or indexing may be specified. If indexing is specified, bits 1 and 0, byte 0, indicate the index register and the destination of the operation implicitly becomes register zero.

The Inclusive OR operation treats each bit of the argument bytes as in the truth table below:

| Bit (0-7) | Bit (0-7) | | Inclusive OR Result |
|-----------|-----------|---|---------------------|
| 0 | 0 | | 0 |
| 0 | 1 | | 1 |
| 1 | 1 | | 1 |
| 1 | 0 | | 1 |

**Processor Registers Affected**      CC

**Condition Code Setting**

| Register Zero | CC1 | CC0 |
|---------------|-----|-----|
| Positive | 0 | 1 |
| Zero | 0 | 0 |
| Negative | 1 | 0 |

## EXCLUSIVE OR TO REGISTER ZERO
(Register Addressing)

**Mnemonic**      EORZ      r

**Binary Code**
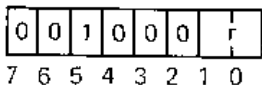
| 0 | 0 | 1 | 0 | 0 | 0 | r |
|---|---|---|---|---|---|---|

7 6 5 4 3 2 1 0

**Execution Time**      2 cycles (6 clock periods)

**Description**

This one-byte instruction causes the contents of the specified register r to be logically Exclusive ORed with the contents of register zero. The result of this operation replaces the contents of register zero. The contents of register r remain unchanged.

The Exclusive OR operation treats each bit of the argument bytes as in the truth table below:

| Bit (0-7) | Bit (0-7) | | Exclusive OR Result |
|-----------|-----------|---|---------------------|
| 0 | 0 | | 0 |
| 0 | 1 | | 1 |
| 1 | 1 | | 0 |
| 1 | 0 | | 1 |

**Processor Registers Affected**      CC

**Condition Code Setting**

| Register Zero | CC1 | CC0 |
|---------------|-----|-----|
| Positive | 0 | 1 |
| Zero | 0 | 0 |
| Negative | 1 | 0 |

## EXCLUSIVE OR IMMEDIATE <span>(Immediate Addressing)</span>

Mnemonic       EORI,r       v

Binary Code
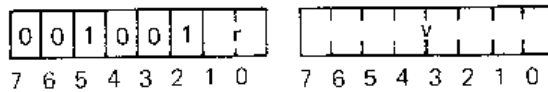
| 0 | 0 | 1 | 0 | 0 | 1 | r | | | | | | v | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Execution Time      2 cycles (6 clock periods)

Description

This two-byte instruction causes the contents of the specified register r to be logically Exclusive ORed with the contents of the second byte of this instruction. The result of this operation replaces the previous contents of register r.

The Exclusive OR operation treats each bit of the argument bytes as in the truth table below:

| Bit (0-7) | Bit (0-7) | Exclusive OR Result |
|-----------|-----------|---------------------|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 1 | 0 |
| 1 | 0 | 1 |

Processor Registers Affected      CC

Condition Code Setting

| Register r | CC1 | CC0 |
|------------|-----|-----|
| Positive | 0 | 1 |
| Zero | 0 | 0 |
| Negative | 1 | 0 |

---

## EXCLUSIVE OR RELATIVE <span>(Relative Addressing)</span>

Mnemonic       EORR,r       (*)a

Binary Code
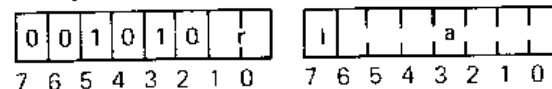
| 0 | 0 | 1 | 0 | 1 | 0 | r | | I | | | | a | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Execution Time      3 cycles (9 clock periods)

Description

This two-byte instruction causes the contents of the specified register r to be logically Exclusive ORed with the contents of the memory byte pointed to by the effective address. The result of this operation replaces the previous contents of register r.

Indirect addressing may be specified.

The Exclusive OR operation treats each bit of the argument bytes as in the truth table below:

| Bit (0-7) | Bit (0-7) | Exclusive OR Result |
|-----------|-----------|---------------------|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 1 | 0 |
| 1 | 0 | 1 |

Processor Registers Affected      CC

Condition Code Setting

| Register r | CC1 | CC0 |
|------------|-----|-----|
| Positive | 0 | 1 |
| Zero | 0 | 0 |
| Negative | 1 | 0 |

## EXCLUSIVE OR ABSOLUTE

**Mnemonic**  EORA,r  (∗)a(,X)

**Binary Code**
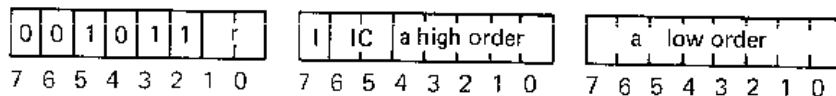
| 0 | 0 | 1 | 0 | 1 | 1 | r |
|---|---|---|---|---|---|---|
| 7 | 6 | 5 | 4 | 3 | 2 | 1 0 |

| I | IC | a high order |
|---|---|---|
| 7 | 6 5 | 4 3 2 1 0 |

| a | low order |
|---|---|
| 7 6 5 | 4 3 2 1 0 |

**Execution Time**  4 cycles (12 clock periods)

**Description**

This three-byte instruction causes the contents of register r to be Exclusive ORed with the contents of the memory byte pointed to by the effective address. The result of the operation replaces the previous contents of register r.

Indirect addressing and/or indexing may be specified. If indexing is specified, bits 1 and 0, byte 0, indicate the index register and the destination of the operation implicitly becomes register zero.

The Exclusive OR operation treats each bit of the argument bytes as in the truth table below:

| Bit (0-7) | Bit (0-7) | Exclusive OR Result |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 1 | 0 |
| 1 | 0 | 1 |

**Processor Registers Affected**  CC

**Condition Code Setting**

| Register r | CC1 | CC0 |
|---|---|---|
| Positive | 0 | 1 |
| Zero | 0 | 0 |
| Negative | 1 | 0 |

## COMPARE TO REGISTER ZERO

**Mnemonic**  COMZ  r

**Binary Code**
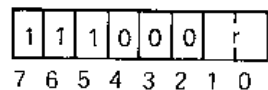
| 1 | 1 | 1 | 0 | 0 | 0 | r |
|---|---|---|---|---|---|---|
| 7 | 6 | 5 | 4 | 3 | 2 | 1 0 |

**Execution Time**  2 cycles (6 clock periods)

**Description**

This one-byte instruction causes the contents of the specified register r to be compared to the contents of register zero. The comparison will be performed in either "arithmetic" or "logical" mode depending on the setting of the COM bit in the Program Status Word.

When COM=1 (logical mode) the values will be interpreted as 8-bit positive binary numbers; when COM=0, the values will be interpreted as 8-bit two's complement numbers.

The execution of this instruction *only* causes the Condition Code to be set as in the following table.

**Processor Registers Affected**  CC

**Condition Code Setting**

|  | CC1 | CC0 |
|---|---|---|
| Register zero greater than Register r | 0 | 1 |
| Register zero equal to Register r | 0 | 0 |
| Register zero less than Register r | 1 | 0 |

## COMPARE IMMEDIATE

**Mnemonic**        COMI,r              v

**Binary Code**
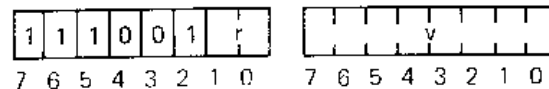
| 1 | 1 | 1 | 0 | 0 | 1 | r |
|---|---|---|---|---|---|---|
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

| | | | | v | | | |
|---|---|---|---|---|---|---|---|
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

**Execution Time**     2 cycles (6 clock periods)

**Description**

This two-byte instruction causes the contents of the specified register r to be compared to the contents of the second byte of this instruction. The comparison will be performed in either the "arithmetic" or "logical" mode depending on the setting of the COM bit in the Program Status Word.

When COM=1 (logical mode), the values will be treated as 8-bit positive binary numbers; when COM=0, the values will be treated as 8-bit two's complement numbers.

The execution of this instruction *only* causes the Condition Code to be set as in the following table.

**Processor Registers Affected**        CC

**Condition Code Setting**

| | CC1 | CC0 |
|---|---|---|
| Register r greater than v | 0 | 1 |
| Register r equal to v | 0 | 0 |
| Register r less than v | 1 | 0 |

---

## COMPARE RELATIVE

**Mnemonic**        COMR,r              (*)a

**Binary Code**
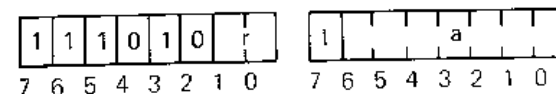
| 1 | 1 | 1 | 0 | 1 | 0 | r |
|---|---|---|---|---|---|---|
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

| 1 | | | | a | | | |
|---|---|---|---|---|---|---|---|
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

**Execution Time**     3 cycles (9 clock periods)

**Description**

This two-byte instruction causes the contents of the specified register r to be compared to the contents of the memory byte pointed to by the effective address. The comparison will be performed in either the "arithmetic" or "logical" mode depending upon the setting of the COM bit in the Program Status Word.

When COM=1 (logical mode), the values will be treated as 8-bit positive binary numbers; when COM=0, the values will be treated as 8-bit, two's complement numbers.

The execution of this instruction *only* causes the Condition Code to be set as in the following table.

**Processor Registers Affected**        CC

**Condition Code Setting**

| | CC1 | CC0 |
|---|---|---|
| Register r greater than memory byte | 0 | 1 |
| Register r equal to memory byte | 0 | 0 |
| Register r less than memory byte | 1 | 0 |

**Mnemonic**      COMA,r        (*)a(,X)
**Binary Code**
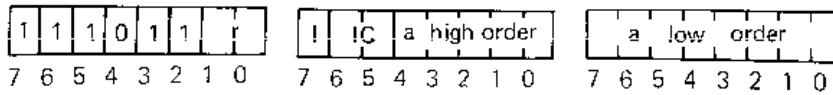
| 1 | 1 | 1 | 0 | 1 | 1 | r | | | ! | IC | a | high order | | | | a | low | order | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | 7 | 6 5 | 4 | 3 2 | 1 | 0 | | 7 6 | 5 4 | 3 2 | 1 0 |

**Execution Time**      4 cycles (12 clock periods)

**Description**

     This three-byte instruction causes the contents of register r to be compared to the contents of the memory byte pointed to by the effective address. The comparison will be performed in either the "arithmetic" or "logical" mode depending on the setting of the COM bit in the Program Status Word.

     Where COM=1 (logical mode), the values will be treated as 8-bit, positive binary numbers; when COM=0 (arithmetic mode), the values will be treated as 8-bit, two's complement numbers.

     Indirect addressing and/or indexing may be specified. If indexing is specified, bits 1 and 0, byte 0, indicate the index register and the destination of the operation implicitly becomes register zero.

     The execution of this instruction *only* causes the Condition Code to be set as in the following table.

**Processor Registers Affected**      CC

**Condition Code Setting**

| | CC1 | CC0 |
|---|---|---|
| Register r greater than memory byte | 0 | 1 |
| Register r equal to memory byte | 0 | 0 |
| Register r less than memory byte | 1 | 0 |

## ROTATE REGISTER LEFT            (Register Addressing)

**Mnemonic**      RRL,r
**Binary Code**
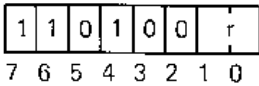
| 1 | 1 | 0 | 1 | 0 | 0 | r | |
|---|---|---|---|---|---|---|---|
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

**Execution Time**      2 cycles (6 clock periods)

**Description**

     This one-byte instruction causes the contents of the specified register r to be shifted left one bit. If the WC bit in the Program Status Word is set to zero, bit #7 of register r flows into bit #0; if WC=1, then bit #7 flows into the Carry bit and the Carry bit flows into bit #0.

     Register bit #4 flows into the IDC if WC=1.
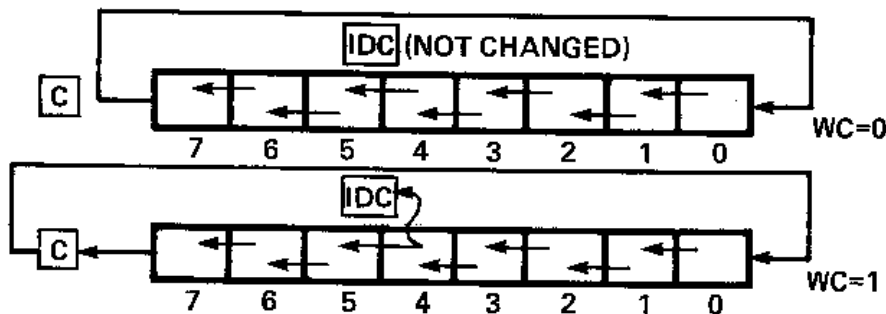


**Note:** Whenever a rotate causes bit #7 of the specified register to change polarity, the OVF bit is set in the PSL.

**Processor Registers Affected**      C, CC, IDC, OVF

**Condition Code Setting**

| Register r | CC1 | CC0 |
|---|---|---|
| Positive | 0 | 1 |
| Zero | 0 | 0 |
| Negative | 1 | 0 |

**Mnemonic**            RRR,r

**Binary Code**
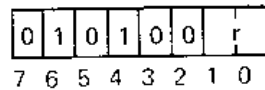
| 0 | 1 | 0 | 1 | 0 | 0 | r |
|---|---|---|---|---|---|---|
| 7 | 6 | 5 | 4 | 3 | 2 | 1 0 |

**Execution Time**      2 cycles (6 clock periods)

**Description**

This one-byte instruction causes the contents of the specified register r to be shifted right one bit. If the WC bit in the Program Status Word is set to zero, bit #0 of the register r flows into bit #7; if WC=1, then bit #0 of the register r flows into the Carry bit and the Carry bit flows into bit #7.

Register bit #6 flows into the IDC if WC=1.
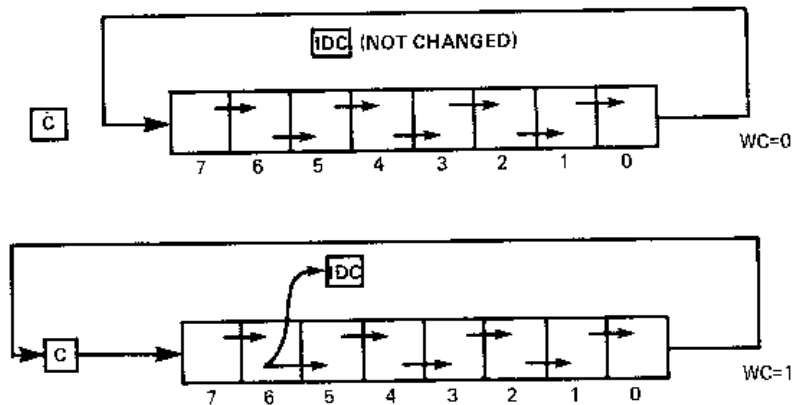


**Note:**  Whenever a rotate causes bit #7 of the specified register to change polarity, the OVF bit is set in the PSL.

**Processor Registers Affected**          C, CC, IDC, OVF

**Condition Code Setting**

| Register r | CC1 | CC0 |
|---|---|---|
| Positive | 0 | 1 |
| Zero | 0 | 0 |
| Negative | 1 | 0 |

---

# LOAD PROGRAM STATUS, UPPER

**Mnemonic**            LPSU

**Binary Code**
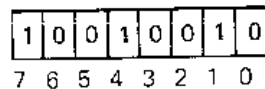
| 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

**Execution Time**      2 cycles (6 clock periods)

**Description**

This one-byte instruction causes the current contents of the Upper Program Status Byte to be replaced with the contents of register zero.

See Program Status Word description for bit assignments. Bits #4 and #3 of the PSU are unassigned and will always be regarded as containing zeroes.

**Processor Registers Affected**          F, II, SP

**Condition Code Setting**                N/A

## LOAD PROGRAM STATUS, LOWER

**Mnemonic**        LPSL

**Binary Code**
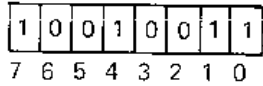
| 1 | 0 | 0 | 1 | 0 | 0 | 1 | 1 |
|---|---|---|---|---|---|---|---|
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

**Execution Time**        2 cycles (6 clock periods)

**Description**

This one-byte instruction causes the current contents of the Lower Program Status Byte to be replaced with the contents of register zero.

See Program Status Word description for bit assignments.

**Processor Registers Affected**        CC, IDC, RS, WC, OVF, COM, C

**Condition Code Setting**

The CC will take on the value in bits #7 and #6 of register zero.


## STORE PROGRAM STATUS, UPPER

**Mnemonic**        SPSU

**Binary Code**
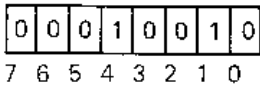
| 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

**Execution Time**        2 cycles (6 clock periods)

**Description**

This one-byte instruction causes the contents of the Upper Program Status Byte to be transferred into register zero.

See Program Status Word description for bit assignments. Bits #4 and #3 which are unassigned will always be stored as zeroes.

**Processor Registers Affected**        CC

**Condition Code Setting**

| Register Zero | CC1 | CC0 |
|---|---|---|
| Positive | 0 | 1 |
| Zero | 0 | 0 |
| Negative | 1 | 0 |

## STORE PROGRAM STATUS, LOWER

**Mnemonic**      SPSL

**Binary Code**
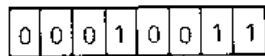
| 0 | 0 | 0 | 1 | 0 | 0 | 1 | 1 |
|---|---|---|---|---|---|---|---|
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

**Execution Time**    2 cycles (6 clock periods)

**Description**

This one-byte instruction causes the contents of the Lower Program Status Byte to be transferred into register zero.

See Program Status Word description for bit assignments.

**Processor Registers Affected**    CC

**Condition Code Setting**

| Register Zero | CC1 | CC0 |
|---|---|---|
| Positive | 0 | 1 |
| Zero | 0 | 0 |
| Negative | 1 | 0 |

## PRESET PROGRAM STATUS UPPER, SELECTIVE    (Immediate Addressing)

**Mnemonic**      PPSU       v

**Binary Code**

| 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | | | | | v | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

**Execution Time**    3 cycles (9 clock periods)

**Description**

This two-byte instruction causes individual bits in the Upper Program Status Byte to be selectively set to binary one. When this instruction is executed, each bit in the v field of the second byte of this instruction is tested for the presence of a one and if a particular bit in the v field contains a one, the corresponding bit in the status byte is set to binary one. Any bits in the status byte which are not selected are not modified.

**Processor Registers Affected**    F, II, SP

**Condition Code Setting**    N/A

## PRESET PROGRAM STATUS LOWER, SELECTIVE   (Immediate Addressing)

**Mnemonic**   PPSL   v

**Binary Code**
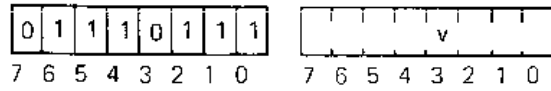
| 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 |   |   |   |   | v |   |   |   |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

**Execution Time**   3 cycles (9 clock periods)

**Description**

This two-byte instruction causes individual bits in the Lower Program Status Byte to be selectively set to binary one. When this instruction is executed, each bit in the v field of the second byte of this instruction is tested for the presence of a one and if a particular bit in the v field contains a one, the corresponding bit in the status byte is set to binary one. Any bits in the status byte which are not selected are not modified.

**Processor Registers Affected**   CC, IDC, RS, WC, OVF, COM, C

**Condition Code Setting**

The CC bits may be set by the execution of this instruction.

## CLEAR PROGRAM STATUS UPPER, SELECTIVE   (Immediate Addressing)

**Mnemonic**   CPSU   v

**Binary Code**
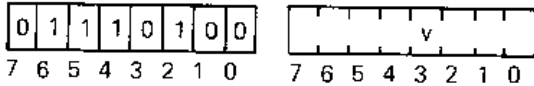
| 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0 |   |   |   |   | v |   |   |   |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

**Execution Time**   3 cycles (9 clock periods)

**Description**

This two-byte instruction causes individual bits in the Upper Program Status Byte to be selectively cleared. When this instruction is executed, each bit in the v field of the second byte of this instruction is tested for the presence of a one and if a particular bit in the v field contains a one, the corresponding bit in the status byte is cleared to zero. Any bits in the status byte which are not selected are not modified.

**Processor Registers Affected**   F, II, SP

**Condition Code Setting**   N/A

## CLEAR PROGRAM STATUS LOWER, SELECTIVE  (Immediate Addressing)

**Mnemonic**      CPSL         v

**Binary Code**
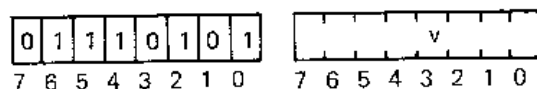
| 0 | 1 | 1 | 1 | 0 | 1 | 0 | 1 |
|---|---|---|---|---|---|---|---|
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

|   |   |   |   | v |   |   |   |
|---|---|---|---|---|---|---|---|
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

**Execution Time**      3 cycles (9 clock periods)

**Description**

This two-byte instruction causes individual bits in the Lower Program Status Byte to be selectively cleared. When this instruction is executed, each bit in the v field of the second byte of this instruction is tested for the presence of a one and if a particular bit in the v field contains a one, the corresponding bit in the status byte is cleared to zero. Any bits in the status byte which are not selected are not modified.

**Processor Registers Affected**      CC, IDC, RS, WC, OVF, COM, C

**Condition Code Setting**

The CC bits may be cleared by the execution of this instruction.

---

## TEST PROGRAM STATUS UPPER, SELECTIVE  (Immediate Addressing)

**Mnemonic**      TPSU         v

**Binary Code**
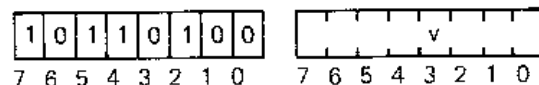
| 1 | 0 | 1 | 1 | 0 | 1 | 0 | 0 |
|---|---|---|---|---|---|---|---|
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

|   |   |   |   | v |   |   |   |
|---|---|---|---|---|---|---|---|
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

**Execution Time**      3 cycles (9 clock periods)

**Description**

This two-byte instruction tests individual bits in the Upper Program Status Byte to determine if they are set to binary one. When this instruction is executed, each bit in the v field of this instruction is tested for the presence of a one, and if a particular bit in the v field contains a one, the corresponding bit in the status byte is tested for a one or zero. The Condition Code is set to reflect the result of this operation.

If a bit in the v field is zero, the corresponding bit in the status byte is not tested.

**Processor Registers Affected**      CC

**Condition Code Setting**

|  | CC1 | CC0 |
|---|---|---|
| All of the selected bits in PSU are 1s | 0 | 0 |
| Not all of the selected bits in PSU are 1s | 1 | 0 |

72

## TEST PROGRAM STATUS LOWER, SELECTIVE  (Immediate Addressing)

**Mnemonic**  TPSL  v

**Binary Code**
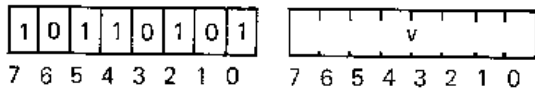
| 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | | | | | | v | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |

**Execution Time**  3 cycles (9 clock periods)

**Description**

This two-byte instruction tests individual bits in the Lower Program Status Byte to determine if they are set to binary one. When this instruction is executed, each bit in the v field of this instruction is tested for a one, and if a particular bit in the v field contains a one, the corresponding bit in the status byte is tested for a one or zero. The Condition Code is set to reflect the result of this operation.

**Processor Registers Affected**  CC

**Condition Code Setting**

| | CC1 | CC0 |
|---|---|---|
| All of the selected bits in PSL are 1s | 0 | 0 |
| Not all of the selected bits in PSL are 1s | 1 | 0 |


## ZERO BRANCH RELATIVE  (Relative Addressing)

**Mnemonic**  ZBRR  (*) a

**Binary Code**

| 1 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | I | | | | a | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
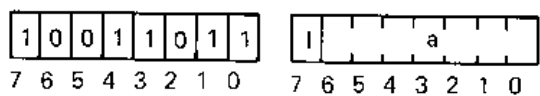
**Execution Time**  3 cycles (9 clock periods)

**Description**

This two-byte unconditional relative branch instruction directs the processor to calculate the effective address differently than the usual calculation for the Relative Addressing mode.

The specified value, a, is interpreted as a relative displacement from page zero, byte zero. Therefore, displacement may be specified from - 64 to +63 bytes. The address calculation is modulo $8192_{10}$, so the negative displacement actually will develop addresses at the end of page zero. For example, ZBRR - 8, will develop an effective address of $8184_{10}$, and a ZBRR +52 will develop an effective address of $52_{10}$.

This instruction causes the processor to clear, address bits 13 and 14, the page address bits; and to replace the contents of the Instruction Address Register with the effective address of the instruction. This instruction may be executed anywhere within addressable memory.

Indirect addressing may be specified.

**Processor Registers Affected**  None

**Condition Code Setting**  N/A

## BRANCH ON CONDITION TRUE, RELATIVE (Relative Addressing)

**Mnemonic**  BCTR,v   (*)a

**Binary Code**

| 0 | 0 | 0 | 1 | 1 | 0 | v |
|---|---|---|---|---|---|---|
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

| I | | | | a | | |
|---|---|---|---|---|---|---|
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

**Execution Time**  3 cycles (9 clock periods)

**Description**

This two-byte conditional branch instruction causes the processor to fetch the next instruction to be executed from the memory location pointed to by the effective address only if the two-bit v field matches the current Condition Code field (CC) in the Program Status Word.

If the v field and CC field do not match, the next instruction is fetched from the location following the second byte of this instruction.

Indirect addressing may be specified.

If the v field is set to $3_{16}$, an unconditional branch is effected.

**Processor Registers Affected**  None

**Condition Code Setting**  N/A

## BRANCH ON CONDITION TRUE, ABSOLUTE (Absolute Addressing)

**Mnemonic**  BCTA,v   (*)a

**Binary Code**
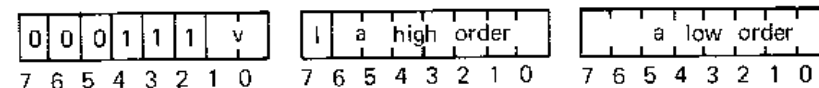
| 0 | 0 | 0 | 1 | 1 | 1 | v |
|---|---|---|---|---|---|---|
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

| I | a | high | order | | | |
|---|---|---|---|---|---|---|
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

| a | low | order | | | | |
|---|---|---|---|---|---|---|
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

**Execution Time**  3 cycles (9 clock periods)

**Description**

This three-byte conditional branch instruction causes the processor to fetch the next instruction to be executed from the memory location pointed to by the effective address only if the two-bit v field matches the two-bit Condition Code field (CC) in the Program Status Word.

If the v field and CC field do not match, the next instruction is fetched from the location following the second byte of this instruction.

Indirect addressing may be specified.

If the v field is set to $3_{16}$, an unconditional branch is effected.

**Processor Registers Affected**  None

**Condition Code Setting**  N/A

## BRANCH ON CONDITION FALSE, RELATIVE  (Relative Addressing)

Mnemonic        BCFR,v        (*)a

Binary Code

| 1 | 0 | 0 | 1 | 1 | 0 | v | | I | | | a | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

**Execution Time**     3 cycles (9 clock periods)

**Description**

This two-byte branch instruction causes the processor to fetch the next instruction to be executed from the memory location pointed to by the effective address only if the two-bit v field does not match the two-bit Condition Code field (CC) in the Program Status Word. If there is no match, the contents of the Instruction Address Register are replaced by the effective address.

If the v field and CC field match, the next instruction is fetched from the location following the second byte of this instruction.

Indirect addressing may be specified.

The v field may not be set to $3_{16}$ as this bit combination is used for the ZBRR operation code.

**Processor Registers Affected**     None

**Condition Code Setting**     N/A

---

## BRANCH ON CONDITION FALSE, ABSOLUTE  (Absolute Addressing)

Mnemonic        BCFA,v        (*)a

Binary Code

| 1 | 0 | 0 | 1 | 1 | 1 | v | | I | a high order | | a low order | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 7 6 5 4 3 2 1 0 | | 7 6 5 4 3 2 1 0 | |

**Execution Time**     3 cycles (9 clock periods)

**Description**

This three-byte instruction causes the processor to fetch the next instruction to be executed from the memory location pointed to by the effective address only if the two-bit v field does not match the two-bit Condition Code field (CC) in the Program Status Word. If there is no match, the contents of the Instruction Address Register are replaced by the effective address.

If the v field and CC field match, the next instruction is fetched from the location following the second byte of this instruction.

Indirect addressing may be specified.

The v field may not be set to $3_{16}$ as this bit combination is used for the BXA operation code.

**Processor Registers Affected**     None

**Condition Code Setting**     N/A

**Mnemonic**        BIRR,r        (*)a

**Binary Code**

| 1 | 1 | 0 | 1 | 1 | 0 | r | | I | | | a | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

**Execution Time**      3 cycles (9 clock periods)

**Description**

This two-byte branch instruction causes the processor to increment the contents of the specified register by one. If the new value in the register is non-zero, the next instruction to be executed is taken from the memory location pointed to by the effective address, i.e., the effective address replaces the previous contents of the Instruction Address Register. If the new value in register r is zero, the next instruction to be executed follows the second byte of this instruction.

Indirect addressing may be specified.

**Processor Registers Affected**      None

**Condition Code Setting**      N/A

---

**Mnemonic**        BIRA,r        (*)a

**Binary Code**
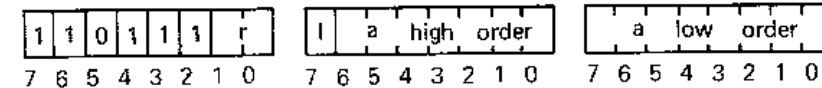
| 1 | 1 | 0 | 1 | 1 | 1 | r | | I | a | high | order | | | | a | low | order | | | | |
|---|---|---|---|---|---|---|---|---|---|------|-------|---|---|---|---|-----|-------|---|---|---|---|
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

**Execution Time**      3 cycles (9 clock periods)

**Description**

This three-byte branch instruction causes the processor to increment the contents of the specified register by one. If the new value in the register is non-zero, the next instruction to be executed is taken from the memory location pointed to by the effective address, i.e., the effective address replaces the previous contents of the Instruction Address Register. If the new value of register r is zero, the next instruction to be executed follows the second byte of this instruction.

Indirect addressing may be specified.

**Processor Registers Affected**      None

**Condition Code Setting**      N/A

## BRANCH ON DECREMENTING REGISTER, RELATIVE (Relative Addressing)

**Mnemonic**    BDRR,r    (*)a

**Binary Code**

| 1 | 1 | 1 | 1 | 1 | 0 | r |   | I |   |   |   | a |   |   |   |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

**Execution Time**    3 cycles (9 clock periods)

**Description**

This two-byte branch instruction causes the processor to decrement the contents of the specified register by one. If the new value in the register is non-zero, the next instruction to be executed is taken from the memory location pointed to by the effective address, i.e., the effective address replaces the previous contents of the Instruction Address Register. If the new value in register r is zero, the next instruction to be executed follows the second byte of this instruction.

Indirect addressing may be specified.

**Processor Registers Affected**    None

**Condition Code Setting**    N/A

## BRANCH ON DECREMENTING REGISTER, ABSOLUTE (Absolute Addressing)

**Mnemonic**    BDRA,r    (*)a

**Binary Code**
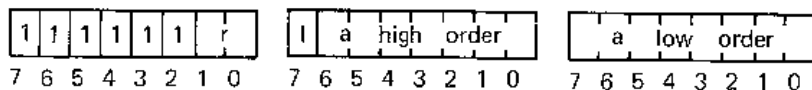
| 1 | 1 | 1 | 1 | 1 | 1 | r |   | I | a | high | order |   | a | low | order |   |
|---|---|---|---|---|---|---|---|---|---|------|-------|---|---|-----|-------|---|
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

**Execution Time**    3 cycles (9 clock periods)

**Description**

This three-byte instruction causes the processor to decrement the contents of the specified register by one. If the new value in the register is non-zero, the next instruction to be executed is taken from the memory location pointed to by the effective address, i.e., the effective address replaces the previous contents of the Instruction Address Register. If the new address in register r is zero, the next instruction to be executed follows the second byte of this instruction.

Indirect addressing may be specified.

**Processor Registers Affected**    None

**Condition Code Setting**    N/A

## BRANCH ON REGISTER NON-ZERO, RELATIVE          (Relative Addressing)

**Mnemonic**          BRNR,r          (*)a

**Binary Code**
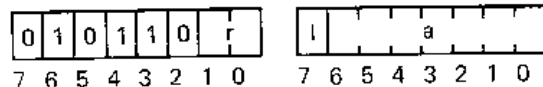
| 0 | 1 | 0 | 1 | 1 | 0 | r |   | I |   |   |   | a |   |   |   |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

**Execution Time**          3 cycles (9 clock periods)

**Description**

This two-byte branch instruction causes the contents of the specified register r to be tested for a non-zero value. If the register contains a non-zero value, the next instruction to be executed is taken from the location pointed to by the effective address, i.e., the effective address replaces the current contents of the Instruction Address Register.

If the specified register contains a zero value, the next instruction is fetched from the location following the second byte of this instruction.

Indirect addressing may be specified.

**Processor Registers Affected**          None

**Condition Code Setting**          N/A


## BRANCH ON REGISTER NON-ZERO, ABSOLUTE          (Absolute Addressing)

**Mnemonic**          BRNA,r          (*)a

**Binary Code**

| 0 | 1 | 0 | 1 | 1 | 1 | r |   | I | a high order |   |   | a low order |   |   |
|---|---|---|---|---|---|---|---|---|--------------|---|---|-------------|---|---|
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 7 | 6 5 4 3 2 1 0 |   |   | 7 6 5 4 3 2 1 0 |   |   |

**Execution Time**          3 cycles (9 clock periods)

**Description**

The three-byte branch instruction causes the contents of the specified register r to be tested for a non-zero value. If the register contains a non-zero value, the next instruction to be executed is taken from the location pointed to by the effective address, i.e., the effective address replaces the contents of the Instruction Address Register.

If the specified register contains a zero value, the next instruction is fetched from the location following the third byte of this instruction.

Indirect addressing may be specified.

**Processor Registers Affected**          None

**Condition Code Setting**          N/A

## BRANCH INDEXED, ABSOLUTE

| Mnemonic | BXA | (*)a,X |
|---|---|---|

**Binary Code**
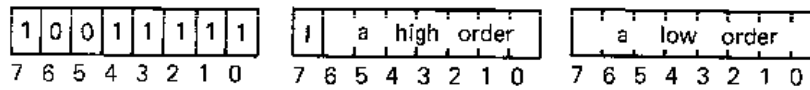
```
| 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 |   | i | a  high  order |   | a  low  order |
  7   6   5   4   3   2   1   0       7  6  5  4  3  2  1  0     7  6  5  4  3  2  1  0
```

**Execution Time**      3 cycles (9 clock periods)

**Description**

This three-byte branch instruction causes the processor to perform an unconditional branch. Indexing is required and register #3 must be specified as the index register because the entire first byte of this instruction is decoded by the processor. When executed, the content of the Instruction Address Register (IAR) is replaced by the effective address.

If indirect addressing is specified, the value in the index register is added to the indirect address to calculate the effective branch address.

| Processor Registers Affected | None |
|---|---|
| Condition Code Setting | N/A |

## ZERO BRANCH TO SUBROUTINE, RELATIVE

| Mnemonic | ZBSR | (*)a |
|---|---|---|

**Binary Code**

```
| 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 |   | I |        a        |
  7   6   5   4   3   2   1   0       7  6  5  4  3  2  1  0
```
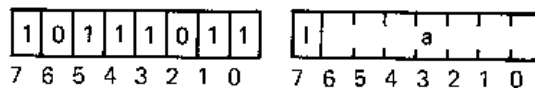
**Execution Time**      3 cycles (9 clock periods)

**Description**

This two-byte unconditional subroutine branch instruction directs the processor to calculate the effective address differently than the usual calculation for the Relative Addressing mode.

The specified value a is interpreted as a relative displacement from page zero, byte zero. Therefore, displacement may be specified from −64 to +63 bytes. The address calculation is modulo $8192_{10}$, so the negative displacement will develop addresses at the end of page zero. For example, ZBSR −10, will develop an effective address of $8182_{10}$, and ZBSR 31 will develop an effective address of $31_{10}$.

This instruction causes the processor to clear the page address bits, address bits 14 and 13, and may be executed anywhere within addressable memory.

Indirect addressing may be specified.

When executed, this instruction causes the Stack Pointer to be incremented by one, the address of the byte following this instruction is pushed into the Return Address Stack (RAS), and control is transferred to the effective address.

| Processor Registers Affected | SP |
|---|---|
| Condition Code Setting | N/A |

## BRANCH TO SUBROUTINE ON CONDITION TRUE, RELATIVE (Relative Addressing)

**Mnemonic**       BSTR,v          (*)a

**Binary Code**

| 0 | 0 | 1 | 1 | 1 | 0 | v |
|---|---|---|---|---|---|---|
| 7 | 6 | 5 | 4 | 3 | 2 | 1 0 |

| I | a |
|---|---|
| 7 | 6 5 4 3 2 1 0 |

**Execution Time**      3 cycles (9 clock periods)

**Description**

This two-byte conditional subroutine branch instruction causes the processor to perform a subroutine branch *only* if the two-bit v field matches the current Condition Code field (CC) in the Program Status Word. If the fields match, the Stack Pointer is incremented by one and the current contents of the Instruction Address Register, which points to the byte following this instruction, is pushed into the Return Address Stack. The effective address replaces the previous contents of the IAR.
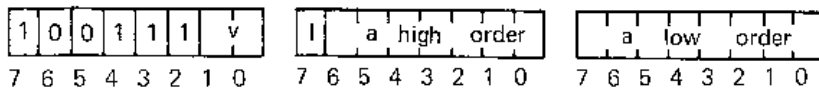
If the v field and CC field do not match, the next instruction is fetched from the location following the second byte of this instruction and the SP is unaffected.

Indirect addressing may be specified.

If v is set to $3_{16}$, the BSTR instruction branches unconditionally.

**Processor Registers Affected**       SP

**Condition Code Setting**             N/A


## BRANCH TO SUBROUTINE ON CONDITION TRUE,ABSOLUTE (Absolute Addressing)

**Mnemonic**       BSTA,v          (*)a

**Binary Code**
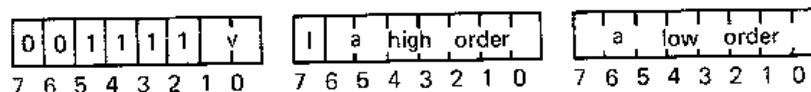
| 0 | 0 | 1 | 1 | 1 | 1 | v |
|---|---|---|---|---|---|---|
| 7 | 6 | 5 | 4 | 3 | 2 | 1 0 |

| I | a high order |
|---|---|
| 7 | 6 5 4 3 2 1 0 |

| a low order |
|---|
| 7 6 5 4 3 2 1 0 |

**Execution Time**      3 cycles (9 clock periods)

**Description**

This three-byte conditional subroutine branch instruction causes the processor to perform a subroutine branch only if the two-bit v field matches the current Condition Code Field (CC) in the Program Status Word. If the fields match, the Stack Pointer is incremented by one and the current contents of the Instruction Address Register, which points to the byte following this instruction is pushed into the Return Address Stack. The effective address replaces the previous contents of the IAR.

If the v field and the CC field do not match, the next instruction is fetched from the location following the third byte of this instruction and the Stack Pointer is unaffected.

Indirect addressing may be specified.

If v is set to $3_{16}$, the BSTA instruction branches unconditionally.

**Processor Registers Affected**       SP

**Condition Code Setting**             N/A

# BRANCH TO SUBROUTINE ON CONDITION FALSE, RELATIVE (Relative Addressing)

Mnemonic         BSFR,v         (*)a

**Binary Code**

| 1 | 0 | 1 | 1 | 1 | 0 | v | | I | | | | a | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Execution Time     3 cycles (9 clock periods)

**Description**

This two-byte conditional subroutine branch instruction causes the processor to perform a subroutine branch *only* if the two-bit v field does *not* match the current Condition Code field (CC) in the Program Status Word. If the fields do not match, the Stack Pointer is incremented by one and the current content of the Instruction Address Register, which points to the location following this instruction, is pushed into the Return Address Stack. The effective address replaces the previous contents of the IAR.

If the v field and the CC match, the next instruction is fetched from the location following this instruction and the SP is unaffected.

Indirect addressing may be specified.

The v field may not be coded as $3_{16}$ because this combination is used for the ZBSR operation code.

Processor Registers Affected         SP

Condition Code Setting         N/A

---

# BRANCH TO SUBROUTINE ON CONDITIONFALSE, ABSOLUTE (Absolute Addressing)

Mnemonic         BSFA,v         (*)a

**Binary Code**
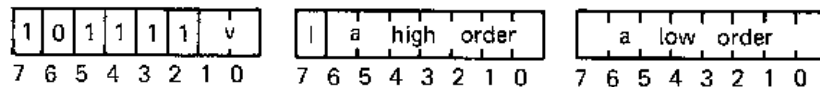
| 1 | 0 | 1 | 1 | 1 | 1 | v | | I | a high order | | | | | | a low order | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Execution Time     3 cycles (9 clock periods)

**Description**

This three-byte conditional subroutine branch instruction causes the processor to perform a subroutine branch *only* if the two-bit v field does *not* match the current Condition Code (CC) in the Program Status Word. If the fields do not match, the Stack Pointer is incremented by one and the current content of the Instruction Address Register, which points to the location following this instruction, is pushed into the Return Address Stack. The effective address replaces the previous contents of the IAR.

If the v field and the CC match, the next instruction is fetched from the location following this instruction and the SP is unaffected.

Indirect addressing may be specified.

The v field may not be coded as $3_{16}$ as this combination is used for the BSXA operation code.

Processor Registers Affected         SP

Condition Code Setting         N/A

## BRANCH TO SUBROUTINE ON NON-ZERO REGISTER, RELATIVE

**Mnemonic** BSNR,r (*)a

**Binary Code**

| 0 | 1 | 1 | 1 | 1 | 0 | r | | | I | | | | a | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
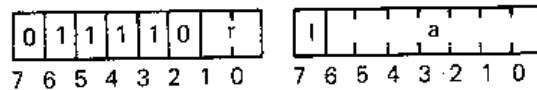
**Execution Time** 3 cycles (9 clock periods)

**Description**

This two-byte subroutine branch instruction causes the contents of the specified register r to be tested for a non-zero value. If the register contains a non-zero value, the next instruction to be executed is taken from the location pointed to by the effective address. Before replacing the contents of the Instruction Address Register with the effective address, the Stack Pointer (SP) is incremented by one and the address of the byte following the instruction is pushed into the Return Address Stack (RAS).

If the specified register contains a zero value, the next instruction is fetched from the location following this instruction.

Indirect addressing may be specified.

**Processor Registers Affected** SP

**Condition Code Setting** N/A

## BRANCH TO SUBROUTINE ON NON-ZERO REGISTER, ABSOLUTE

**Mnemonic** BSNA,r (*)a

**Binary Code**

| 0 | 1 | 1 | 1 | 1 | 1 | r | | I | a | high | order | | a | low | order | |
|---|---|---|---|---|---|---|---|---|---|------|-------|---|---|-----|-------|---|
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
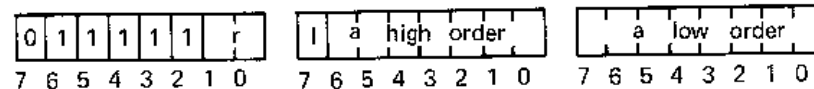
**Execution Time** 3 cycles (9 clock periods)

**Description**

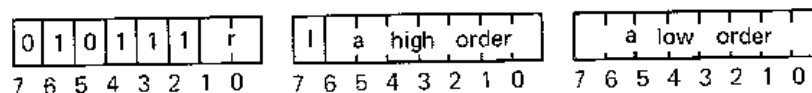This three-byte subroutine branch instruction causes the contents of the specified register r to be tested for a non-zero value. If the register contains a non-zero value, the next instruction to be executed is taken from the location pointed to by the effective address. Before replacing the current contents of the Instruction Address Register (IAR) with the effective address, the Stack Pointer (SP) is incremented by one and the address of the byte following the instruction is pushed into the Return Address Stack (RAS).

If the specified register contains a zero value, the next instruction is fetched from the location following this instruction.

Indirect addressing may be specified.

**Processor Registers Affected** SP

**Condition Code Setting** N/A

## BRANCH TO SUBROUTINE INDEXED, ABSOLUTE, UNCONDITIONAL

(Absolute Addressing)

Mnemonic         BSXA         (*)a,X

Binary Code
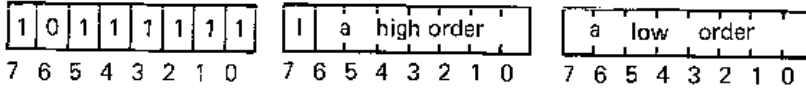
| 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | | I | a | high order | | | a | low | order | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |

(columns: 7 6 5 4 3 2 1 0 for last byte)

Execution Time     3 cycles (9 clock periods)

Description

This three-byte instruction causes the processor to perform an unconditional subroutine branch. Indexing is required and register #3 must be specified as the index register because the entire first byte of this instruction is decoded by the processor.

Execution of this instruction causes the Stack Pointer (SP) to be incremented by one, the address of the byte following this instruction is pushed into the Return Address Stack (RAS), and the effective address replaces the contents of the Instruction Address Register.

If indirect addressing is specified, the value in the index register is added to the indirect address to calculate the effective address.

Processor Registers Affected      SP

Condition Code Setting      N/A


## RETURN FROM SUBROUTINE, CONDITIONAL

Mnemonic         RETC,v

Binary Code
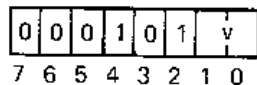
| 0 | 0 | 0 | 1 | 0 | 1 | v | |
|---|---|---|---|---|---|---|---|
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Execution Time     3 cycles (9 clock periods)

Description

This one-byte instruction is used by a subroutine to conditionally effect a return of control to the program which last issued a subroutine branch instruction.

If the two-bit v field in the instruction matches the Condition Code field (CC) in the Program Status Word, the following action is taken: The address contained in the top of the Return Address Stack replaces the previous contents of the Instruction Address Register (IAR), and the Stack Pointer is decremented by one.

If the v field does not match CC, the return is not effected and the next instruction to be executed is taken from the location following this instruction.

If v is specified as $3_{16}$, the return is executed unconditionally.

Processor Registers Affected      SP

Condition Code Setting      N/A

**Mnemonic**          RETE,v

**Binary Code**
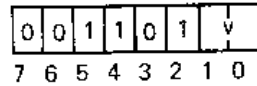
| 0 | 0 | 1 | 1 | 0 | 1 | v | |
|---|---|---|---|---|---|---|---|
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

**Execution Time**      3 cycles (9 clock periods)

**Description**

This one-byte instruction is used by a subroutine to conditionally effect a return of control to the program which last issued a subroutine branch instruction. Additionally, if the return is effected, the Interrupt Inhibit (II) bit in the Program Status Word is cleared to zero, thus enabling interrupts. This instruction is mainly intended to be used by an interrupt handling routine because receipt of an interrupt causes a subroutine branch to be effected and the Interrupt Inhibit bit to be set to 1. The interrupt handling routine must be able to return and enable simultaneously so that the interrupt routine cannot be interrupt unless that is specifically desired.

If the two-bit v field in the instruction matches the Condition Code field (CC) in the Program Status Word, the following action is taken: The address contained in the top of the Return Address Stack (RAS) replaces the previous contents of the Instruction Address Register (IAR), the Stack Pointer is decremented by one and the II bit is cleared to zero.

If the v field does not match CC, the return is not effected and the next instruction to be executed is taken from the location following this instruction.

If v is specified as $3_{16}$, the return is executed unconditionally.

**Processor Registers Affected**          SP , II

**Condition Code Setting**          N/A

---

## READ DATA                                      (Register Addressing)

---

**Mnemonic**          REDD,r

**Binary Code**
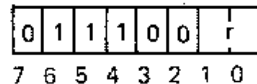
| 0 | 1 | 1 | 1 | 0 | 0 | r | |
|---|---|---|---|---|---|---|---|
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

**Execution Time**      2 cycles (6 clock periods)

**Description**

This one-byte input instruction causes a byte of data to be transferred from the data bus into register r. Signals on the data bus are considered to be true signals, i.e., a high level will be set into the register as a one.

When executing this instruction, the processor raises the Operation Request (OPREQ) line, simultaneously switching the M/$\overline{IO}$ line to $\overline{IO}$ and the $\overline{R}$/W to $\overline{R}$ (Read). Also, during the OPREQ signal, the D/$\overline{C}$ line switches to D (Data) and the E/$\overline{NE}$ switches to $\overline{NE}$ (Non-extended).

See Input/Output section of this manual.

**Processor Registers Affected**          CC

| **Condition Code Setting** | Register r | CC1 | CC0 |
|---|---|---|---|
| | Positive | 0 | 1 |
| | Zero | 0 | 0 |
| | Negative | 1 | 0 |

**Mnemonic**         REDC,r

**Binary Code**
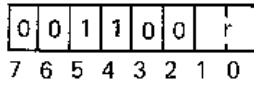
| 0 | 0 | 1 | 1 | 0 | 0 | r |
|---|---|---|---|---|---|---|
| 7 | 6 | 5 | 4 | 3 | 2 | 1 0 |

**Execution Time**      2 cycles (6 clock periods)

**Description**

This one-byte input instruction causes a byte of data to be transferred from the data bus into register r. Signals on the data bus are considered to be true signals, i.e., a high level will be set into the register as a one.

When executing this instruction, the processor raises the Operation Request (OPREQ) line, simultaneously switching the $M/\overline{IO}$ line to $\overline{IO}$, the $\overline{R}/W$ line to $\overline{R}$ (Read), the $D/\overline{C}$ line to $\overline{C}$ (Control), and the $E/\overline{NE}$ line to $\overline{NE}$ (Non-extended).

See Input/Output section of this manual.

**Processor Registers Affected**      CC

**Condition Code Setting**

| Register r | CC1 | CC0 |
|---|---|---|
| Positive | 0 | 1 |
| Zero | 0 | 0 |
| Negative | 1 | 0 |

---

# READ EXTENDED

**Mnemonic**         REDE,r        v

**Binary Code**

| 0 | 1 | 0 | 1 | 0 | 1 | r |   | v |
|---|---|---|---|---|---|---|---|---|
| 7 | 6 | 5 | 4 | 3 | 2 | 1 0 | | 7 6 5 4 3 2 1 0 |
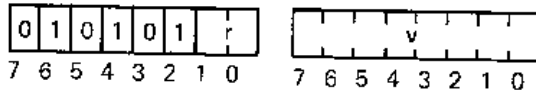
**Execution Time**      3 cycles (9 clock periods)

**Description**

This two-byte input instruction causes a byte of data to be transferred from the data bus into register r. During the execution of this instruction, the content of the second byte of this instruction is made available on the address bus. Signals on the data bus are true signals, i.e., a high level is interpreted as a one.

During execution, the processor raises the Operation Request (OPREQ) line, simultaneously placing the contents of the second byte of the instruction on the address bus. During the OPREQ signal, the $M/\overline{IO}$ line is switched to $\overline{IO}$, the $\overline{R}/W$ line to $\overline{R}$ (Read), line and the $E/\overline{NE}$ line to E (Extended).

See Input/Output section of this manual.

**Processor Registers Affected**      CC

**Condition Code Setting**

| Register r | CC1 | CC0 |
|---|---|---|
| Positive | 0 | 1 |
| Zero | 0 | 0 |
| Negative | 1 | 0 |

**Mnemonic**  WRTD,r

**Binary Code**
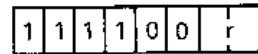
| 1 | 1 | 1 | 1 | 0 | 0 | r |
|---|---|---|---|---|---|---|
| 7 | 6 | 5 | 4 | 3 | 2 | 1 0 |

**Execution Time**  2 cycles (6 clock periods)

**Description**

This one-byte output instruction causes a byte of data to be made available to an external device. The byte to be output is taken from register r and made available on the data bus. Signals on the data bus are true signals, i.e., high levels are ones.

When executing this instruction, the processor raises the Operation Request (OPREQ) line and simultaneously places the data on the Data Bus. Along with the OPREQ, the M/$\overline{IO}$ line is switched to $\overline{IO}$, the $\overline{R}$/W signal is switched to W (Write), and a Write Pulse (WRP) is generated. Also, during the valid OPREQ signals, the D/$\overline{C}$ line is switched to D (Data) and the E/$\overline{NE}$ line is switched to $\overline{NE}$ (Non-extended).

See Input/Output section of this manual.

**Processor Registers Affected**  None

**Condition Code Setting**  N/A

---

**Mnemonic**  WRTC,r

**Binary Code**
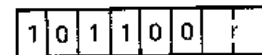
| 1 | 0 | 1 | 1 | 0 | 0 | r |
|---|---|---|---|---|---|---|
| 7 | 6 | 5 | 4 | 3 | 2 | 1 0 |

**Execution Time**  2 cycles (6 clock periods)

**Description**

This one-byte output instruction causes a byte of data to be made available to an external device.

The byte to be output is taken from register r and made available on the data bus. Signals on the data bus are true signals, i.e., high levels are ones

When executing this instruction, the processor raises the Operation Request (OPREQ) line and simultaneously places the data on the Data Bus. Along with the OPREQ signal, the M/$\overline{IO}$ line is switched to $\overline{IO}$, the $\overline{R}$/W signal is switched to W (Write), the D/$\overline{C}$ line is switched to $\overline{C}$ (Control), the E/$\overline{NE}$ is switched to $\overline{NE}$ (Non-extended), and a Write Pulse (WRP) is generated.

See the Input/Output section of this manual.

**Processor Registers Affected**  None

**Condition Code Setting**  N/A

**Mnemonic**    WRTE,r          v

**Binary Code**

| 1 | 1 | 0 | 1 | 0 | 1 | r |
|---|---|---|---|---|---|---|
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

| | | | | v | | | |
|---|---|---|---|---|---|---|---|
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
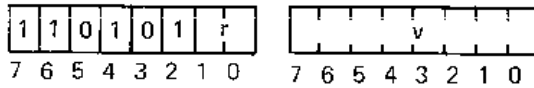
**Execution Time**    3 cycles (9 clock periods)

**Description**

This two-byte output instruction causes a byte of data to be made available to an external device. The byte to be output is taken from register r and is made available on the data bus. Simultaneously, the data in the second byte of this instruction is made available on the address bus. The second byte, v, may be interpreted as a device address.

Signals on the busses are true levels, i.e., high levels are ones.

When executing this instruction, the processor raises the Operation Request (OPREQ) line and simultaneously places the data from register r on the data bus and the data from the second byte of this instruction on the address bus. Along with OPREQ, the $M/\overline{IO}$ line is switched to $\overline{IO}$, the $\overline{R}/W$ line is switched to W (Write), the $E/\overline{NE}$ line is switched to E (Extended), and a Write Pulse (WRP) is generated.

See the Input/Output section of this manual.

**Processor Registers Affected**    None

**Condition Code Setting**    N/A


# NO OPERATION

**Mnemonic**    NOP

**Binary Code**
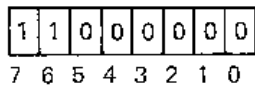
| 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
|---|---|---|---|---|---|---|---|
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

**Execution Time**    2 cycles (6 clock periods)

**Description**

This one-byte instruction causes the processor to take no action upon decoding it. No registers are changed, but fetching and executing a NOP instruction requires two processor cycles.

**Processor Registers Affected**    None

**Condition Code Setting**    N/A

**Mnemonic**  TMI,r  v

**Binary Code**
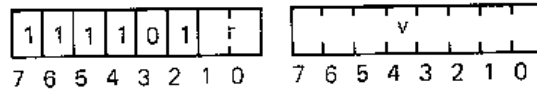
| 1 | 1 | 1 | 1 | 0 | 1 | r | | | | v | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

**Execution Time**  3 cycles (9 clock periods)

**Description**

This two-byte instruction tests individual bits in the specified register r to determine if they are set to binary one. During execution, each bit in the v field of the instruction is tested for a one, and if a particular bit in the v field contains a one, the corresponding bit in register r is tested for a one or zero. The condition code is set to reflect the result of the operation.

If a bit in the v field is zero, the corresponding bit in register r is not tested.

**Processor Registers Affected**  CC

**Condition Code Setting**

| | CC1 | CC0 |
|---|---|---|
| All of the selected bits are 1s | 0 | 0 |
| Not all of the selected bits are 1s | 1 | 0 |

**Mnemonic**         DAR,r

**Binary Code**

| 1 | 0 | 0 | 1 | 0 | 1 | r |
|---|---|---|---|---|---|---|
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
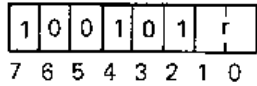
**Execution Time**      3 cycles (9 clock periods)

**Description**

   This one–byte instruction conditionally adds a decimal ten (two's complement negative six in a four-bit binary number system) to either the high order 4 bits and/or the low order 4 bits of the specified register r.

   The truth table below indicates the logical operation performed. The operation proceeds based on the contents of the Carry (C) and Interdigit Carry (IDC) bits in the Program Status Word. The C and IDC remain unchanged by the execution of this instruction.

This instruction allows BCD sign magnitude arithmetic to be performed on packed digits by the following procedure.

BCD Addition:     1.  add $66_{16}$ to augend
                  2.  perform addition of addend and augend
                  3.  perform DAR instruction

BCD Subtraction:  1.  perform subtraction (2's complement of subtrahend is added to the minuend)
                  2.  perform DAR instruction

Since this operation is on sign-magnitude numbers, it is necessary to establish the sign of the result prior to executing in order to properly control the definition of the subtrahend and minuend.

| Carry | Interdigit Carry | | Added to Register r |
|-------|------------------|---|---------------------|
| 0 | 0 | | $AA_{16}$ |
| 0 | 1 | | $A0_{16}$ |
| 1 | 1 | | $00_{16}$ |
| 1 | 0 | | $0A_{16}$ |

**Processor Registers Affected**         CC

**Condition Code Setting**
The Condition Code is set to a meaningless value.

## HALT, ENTER WAIT STATE

**Mnemonic**        HALT

**Binary Code**

| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
|---|---|---|---|---|---|---|---|
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

**Execution Time**     2 cycles (6 clock periods)

**Description**

This one–byte instruction causes the processor to stop executing instructions and enter the WAIT state. The RUN/$\overline{\text{WAIT}}$ line is set to the WAIT state.

The only way to enter the RUN state after a HALT has been executed, is to reset the 2650 or to interrupt the processor.

**Processor Registers Affected**      None

**Condition Code Setting**      N/A

# CHAPTER III

# 2650 ASSEMBLER LANGUAGE

# INTRODUCTION

The assembly language described in this document is a symbolic language designed specifically to facilitate the writing of programs for the Signetics 2650 processor. The 2650 Assembler is a program which accepts symbolic source code as input and produces a listing and/or an object module as output.

The assembler is written in standard FORTRAN IV and is available either through a timesharing service or in batch form directly from Signetics. This is done to assure compatibility and ease of installation on a user's own computer equipment. It is modular and may be executed in an overlay mode should memory restrictions make that necessary. The program is approximately 1,250 FORTRAN card images in length.

An attempt was made in the design of the language to make it similar to other contemporary assembler languages because it was felt that such similarity would reduce the learning time necessary to become proficient in this language. The 2650 assembler features forward references, self-defining constants, free format source code, symbolic addressing, syntax error checking, load module generation, and source statement listing.

In order to understand the 2650 instruction set, architecture, timing, interface requirements and electrical characteristics, the reader is referred to the Signetics 2650 Hardware Specification section.

The assembler is a two pass program that builds a symbol table, issues helpful error messages, produces an easily readable program listing and outputs a computer readable object (load) module.

The assembler features symbolic and relative addressing, forward references, complex expression evaluations and a versatile set of Pseudo-Operations. These features aid the programmer/engineer in producing well-documented, working programs in a minimum of time. Additionally, the assembler is capable of generating data in several number based systems as well as both ASCII and EBCDIC character codes.

## ASSEMBLER LANGUAGE

The assembler language provides a means to create a computer program. The features of the Assembler are designed to meet the following goals:

- Programs should be easy to create
- Programs should be easy to modify
- Programs should be easy to read and understand
- A machine readable, machine language module to be output

This assembler language has been developed with the following features:

- Symbolic machine operation codes (op-codes, mnemonics)
- Symbolic address assignment and references
- Relative addressing
- Data creation statements
- Storage reservation statements
- Assembly listing control statements
- Addresses can be generated as constants
- Character codes may be specified as ASCII or EBCDIC
- Comments and remarks may be encoded for documentation

As Assembly language program is a program written in symbolic machine language. It is comprised of statements. A statement is either a symbolic machine instruction, a pseudo-operation statement, or a comment.

The symbolic machine instruction is a written specification for a particular machine operation expressed by symbolic operation codes and sometimes symbolic addresses or operands. For example:

LOC2        STRR, R0    SAV

*Where:*

LOC2        is a symbol which will represent the memory address of the instruction.

STRR        is a symbolic op-code which represents the bit pattern of the "store relative" instruction.

R0          is a symbol which has been defined as register 0 by the "EQU pseudo-op".

SAV         is a symbol which represents the memory location into which the contents of register 0 are to be stored.

A pseudo-operation statement is a statement which is not translated into a machine instruction, but rather is interpreted as a directive to the assembler program. Example:

SCHD        ACON            REDY

*Where:*

ACON        is a pseudo-op which directs the assembler program to allocate two bytes of memory.

REDY        is a symbol, representing an address. The assembler is directed to place the equivalent memory address into the byte allocated space.

SCHD        is a symbol. The assembler is to assign the memory address of the first byte of the two allocated to this symbol.

**STATEMENTS**

Statements are always written in a particular format. The format is depicted below:

LABEL FIELD  OPERATION FIELD  OPERAND FIELD  COMMENT FIELD

The statement is always assumed to be written on an 80 column data processing card or an 80 column card image.

The Label Field is provided to assign symbolic names to bytes of memory. If present, the Label Field must begin in logical column one.

The Operation Field is provided to specify a symbolic operation code or a pseudo-operation code. If present, the Operation Field must either begin past column one or be separated from logical column one by one or more blanks.

94

The Operand Field is provided to specify arguments for the operation in the Operation Field. The Operand Field, if present, is separated from the Operation Field by one or more blanks.

The Comment Field is provided to enable the assembly language programmer to optionally place an English message stating the purpose or intent of a statement or a group of statements. The Comment Field must be separated from the preceding field by one or more blanks.

## COMMENT STATEMENT

A Comment Statement is a statement that is not processed by the assembler program. It is merely reproduced on the assembly listing. A Comment Statement is indicated by encoding an asterisk in logic column one. Example:

*THIS IS A COMMENT STATEMENT

Logical columns 72-80 are never processed by the assembler, they are always reproduced on the assembly listing without processing. This field is a good place for sequence numbers, if desired.

## SYMBOLIC ADDRESSING

When writing statements in symbolic machine language, i.e., assembler language, the machine operation code is usually expressed symbolically. For example, the machine instruction that stores data from register 0 into a memory location named SAV, may be expressed as:

STRA, R0      SAV

The assembler, when translating this symbolic operation code and its arguments into machine language for the 2650, defines three bytes containing H'CC0020', where '0020' is the value of SAV.

The address of the translated bytes is known because the Assembly Program Counter is always set to the address of the next byte to be assembled.

The user can attach a label to an instruction:

SAVR          STRR,R0      SAV

The assembler, upon seeing a valid symbol in the label field, assigns the equivalent address to the label. In the given example, if the STRR instruction is to be stored in the address H'0127', then the symbol SAVR would be made equivalent to the value H'0127' for the duration of the assembly.

The symbol could then be used anywhere in the source program to refer to the address value or, more typically, it could be used to refer to the instruction location. The important concept is that the address of the instruction need not be known; only the symbol need to be used to refer to the instruction location. Thus, when branching to the STRR instruction, one could write:

BCTA,3        SAVR

When the three byte branch instruction is translated by the assembler,

the address of the STRR instruction is placed in the address field of the branch instruction.

It is also possible to use symbolic addresses which are near other locations to refer to those locations without defining new labels. For example:

|       |         |       |
|-------|---------|-------|
|       | BCTR,3  | BEG   |
|       | BCTR,0  | BEG+4 |
|       | ANDZ    | 3     |
|       | BSTR,3  | S+48  |
| BEG   | LODA,2  | PAL   |
|       | HALT    |       |
|       | SUBI,2  | 3     |

In the above example, the instruction "BCTR,3    BEG" refers to the LODA,2    PAL instruction. The instruction "BCTR,0    BEG+4" refers to the SUBI,2    3 instruction.

BEG+4 means the address BEG plus four bytes. This type of expression is called relative symbolic addressing and given a symbolic address; it can be used as a landmark to express several bytes before or after the symbolic address. Examples:

|       |         |        |
|-------|---------|--------|
|       | BCTR,3  | PAL+23 |
|       | BSTA,0  | STT-18 |

The arguments are evaluated like any other expression and cannot exceed in value the maximum number that can be contained in a FORTRAN integer constant.

### PROGRAM COUNTER

During the assembly process the assembler maintains a FORTRAN Integer cell that always contains the address of the next memory location to be assembled. This cell is called the Program Counter. It is used by the assembler to assign addresses to assembled bytes, but it is also available to the programmer.

The character "$" is the only valid symbol containing a special character that the assembler recognizes without error. "$" is the symbolic name of the Program Counter. It may be used like any other symbol, but it may not appear in the label field.

When using the "$", the programmer may think of it as expressing the idea "$" = "address of myself". For example,

|            |         |     |
|------------|---------|-----|
| $108_{16}$ | BCTR,3  | $   |

This branch instruction is in location $108_{16}$. The instruction directs the microprocessor to "branch to myself". The Program Counter in this example contains the value $108_{16}$.

# LANGUAGE ELEMENTS

Input to the assembler consists of a sequence of characters combined to form assembly language elements. These language elements include symbols, instruction mnemonics, constants and expressions which make up the individual program statements that comprise a source program.

## CHARACTERS

| | |
|---|---|
| Alphabetic: | A through Z |
| Numeric: | 0 through 9 |
| Special characters: | blank |
| | ( left parenthesis |
| | ) right parenthesis |
| | + add or positive value |
| | - subtract or negative value |
| | * asterisk |
| | ' single quote |
| | , comma |
| | / slash |
| | $ dollar sign |
| | < less than sign |
| | > greater than sign |

## SYMBOLS

Symbols are formed from combination of characters. Symbols provide a convenient means of identifying program elements so they can be referenced by other elements.

1. Symbols may consist of 1 to 4 alphanumeric characters: A through Z, 0 through 9.
2. Symbols must begin with an alphabetic character.
3. The character $ is a special symbol which may be used in the argument field of a statement to represent the current value of the Location Counter.
4. The character * is a special symbol which is used as an indirect address indicator.
5. The characters + and - are also used as auto-increment/auto-decrement indicators.

The following are examples of valid symbols:

DOP1        RAV3
AA          TEMZ

The following are examples of invalid symbols:

1LAR        begins with numeric
PA N        imbedded blank

## CONSTANTS

A constant is a self-defining language element. Unlike a symbol, the value of a constant is its own "face" value and is invariant. Internal numbers are represented in 2's complement notation. There are two forms in which constants may be written: the Self-Defining Constant and the General Constant.

### SELF-DEFINING CONSTANT

The self-defining constant is a form of constant which is written directly in an instruction and defines a decimal value. For example:

LODA,R3     BUFF+65

In this example, 65 is a self-defining constant. The maximum value of the integer constant expressed by a self-defining constant is that which, when expressed in binary, will fit within the basic arithmetic unit of the host computer (typically 1 word).

### GENERAL CONSTANT

The general constant is also written directly in an instruction, but the interpretation of its value is dictated by a code character and delimited by quotation marks.

LODA,R3     BUFF+H'3E'

In this example, the code letter H specifies that 3E is a hexadecimal constant equivalent to decimal value 62.

The maximum size of a number generated by a general constant form (B, O, D, H) may be no larger than the size of the FORTRAN integer cell of the host computer. However, the most important concept to understand when using constant forms is that the final value of a resolved expression must fit the constraints of the actual field destined to contain the value. For example:

LODA,R2     PAL+H'3EE2'-H'3EE0'

In this case, the argument, when resolved, must fit into the 13 bits in the actual machine instruction. Even though each of the two hexadecimal constants are larger than can fit into 13 bits, the final value of the expression is containable in 13 bits and therefore the constants are permitted. Similarly, the statement DATA H'3FE' is not allowed, as the DATA statement defines one byte quantities and H'3FE' specifies more than 8 bits. Summarily, the size of the evaluated expressions must be less than or equal to their corresponding data fields. There are 6 types of General Constants:

| Code | Type |
|------|------|
| B | Binary Constant |
| O | Octal Constant |
| D | Decimal Constant |
| H | Hexadecimal Constant |
| E | EBCDIC Character Constant |
| A | ASCII Character Constant |

### B: BINARY CONSTANT

A binary constant consists of an optionally signed binary number of up to 8 bits enclosed in single quotes and preceded by the letter B, e.g., B'1011011'. Binary information is stored right justified.

### O: OCTAL CONSTANT

An octal constant consists of an optionally signed octal number enclosed

by single quotation marks and preceded by the letter O, e.g., O'352'. The value will be right justified.

### D: DECIMAL CONSTANT

A decimal constant consists of an optionally signed decimal number enclosed by single quotation marks and preceded by the letter D, e.g., D'249'. The value will be right justified.

### H: HEXADECIMAL CONSTANT

A hexadecimal constant consists of an optionally signed hexadecimal number enclosed in single quotation marks and preceded by the letter H, e.g., H'3F'. The value will be right justified.

### E: EBCDIC CHARACTER CONSTANT

An EBCDIC character consists of a string of EBCDIC characters enclosed by single quotation marks and preceded by the letter E, e.g., E'ARE YOU THERE?'. Each character will be encoded in 8-bit EBCDIC and stored in successive bytes. The maximum number of characters which may be specified in one character string constant is 16.

### A: ASCII CHARACTER CONSTANT

An ASCII character constant consists of a string of ASCII characters enclosed by quotation marks and preceded by the letter A. For example: A'HELLO THERE'. Each character will be encoded in 7-bit ASCII and stored in successive bytes. The high order bit is always set to zero in each allocated byte. Up to 16 characters may be specified in one statement.

Note: See Appendix C for permissible characters and their equivalent ASCII and EBCDIC codes. To specify a single quotation mark as a character constant it must appear twice in the character string, e.g., A'TYPE' 'HELP' 'NOW' will appear in storage as TYPE'HELP'NOW.

## MULTIPLE CONSTANT SPECIFICATIONS

General constant forms, except A and E, allow multiple specifications within the constant expression. For example: D'52, 21, 208, 27'. A comma separates each byte specification and successive specifications determine successive bytes of storage. Only 16 bytes of information may be specified in any one general constant form and each byte may be optionally signed. For example:

H'03,-F2,+11,-8,33,0'
O'271,133'.

## EXPRESSIONS

An expression is an assembly language element that represents a value. It consists of a single term or a combination of terms separated by arithmetic operators. A term may be a valid symbolic reference, a self-defining constant or a general constant.

It is important to understand that although individual terms in a expression may exceed the number size restriction of the 2650 (one or two bytes), they may not cause the number size of the host computer's integer FORTRAN constant to be exceeded.

Examples of valid expressions:

LOOP            PAL-$
LOOP+5          $-PAL+3
SAM+3 LOOP      BIT 3+H'3H'

Note: The special symbol '$' represents the current value of the location counter.

## SPECIAL OPERATORS

There are two special operators that are recognized by the assembler. They are:

$<$ less than sign
$>$ greater than sign

The assembler interprets these operators in a special way:

$<$ perform a modulo 256 divide (use high order byte)
$>$ perform a divide by 256 (use low order byte)

These operators, when used, must appear as the first character in the argument field. If they are imbedded in an expression, the results are unpredictable.

These special operators are intended to be used to access a two byte address in one byte parts using a minimum of storage. For example, if it is desired to get the high order bits of an address (ADDB) into register 2 and the low order bits into register 1 it could be done as follows:

```
        LODR,R2    APAL
        LODR,R1    APAL+1
        • • •
        • • •
        • • •
APAL    ACON       ADDB
```

or, by utilizing the special operators, it could be done as follows:

```
        LODI,R2    <ADDB
        LODI,R1    >ADDB
```

The first method uses 6 bytes to accomplish what the second method can do in 4 bytes.

The special operators care most often used to facilitate the passing of an address in registers.

# SYNTAX

Assembly language elements may be combined to symbolically express both 2650 instructions and assembler directives. There are specific rules for writing these instructions. This set of rules is known as the Syntax of the symbolic assembler language. The following description assumes a logical input of an 80-column data processing card, but since the host assembler is written in Fortran, the input media may be magnetic tape, magnetic disk, paper tape, etc. Only the format statement for input need be changed to accommodate the various input media.

## FIELDS

A statement prepared for processing by the assembler is logically divided into four fields: the Name Field, the Operation Field, the Argument Field and the Comment Field. Each field is separated by at least one blank character. No continuation cards are allowed, and only logical columns 1 through 72 are scanned by the assembler. Logical columns 73 through 80 inclusive may be used for any desired purpose.

### NAME FIELD

The name (or label) field optionally contains a symbolic name which the assembler assigns to the instruction specified in the remaining part of the line. If a name is specified, it must begin in logical column 1. The assembler assumes that there is no name if logical column 1 is blank. The name field, if present, must contain only a valid symbol.

### OPERATION FIELD

The operation field contains a mnemonic code which represents a 2650 processor operation or an assembly directive. The operation field must be present in every non-comment line. See Appendix A for a list of the valid mnemonic codes. Additionally, depending on the instruction type, the operation field may also specify a general purpose register or a condition code.

### ARGUEMENT FIELD

The argument field contains one or more symbols, constants or expressions separated by commas. The argument field specifies storage locations, constants, register specifications and any other information necessary to completely specify a machine operation or an assembler directive. Embedded blanks are not permitted as they are considered field terminators.

### COMMENT FIELD

The comment field contains any valid characters in any combination. The comment field is not processed by the assembler, but is merely reproduced on the listing next to the accompanying instruction. It is usually used to explain the purpose or intention of a particular instruction or group of instructions.

### COMMENT CARD

An entire 72 column line may be utilized to print comments by coding an asterisk (*) in column 1. This entire card is merely reproduced on the assembly listing without processing by the assembler.

## SYMBOLS

Symbols are used in the name field of a symbolic machine instruction to identify that particular instruction and to represent its address. Symbols may be used for other purposes, such as the symbolic representation of some memory address, the symbolic representation of a constant, the symbolic representation of a register, etc.

No matter how the symbol is used, it must be defined. A symbol is defined when the assembler knows what value the symbol represents. There is only one way to define a symbol. The symbol must at some time appear either in the name field of an instruction or of an assembler directive. The symbol will be assigned the current value of the Location Counter when it appears in the name field of a machine instruction, or it may be assigned some other value through use of the EQU assembler directive. A symbol may not appear in the name field more than once in a program, because this would cause the assembler to try to redefine an already defined label. The assembler will not do this and will flag the second appearance of a particular label as an error.

## SYMBOLIC REFERENCES

Symbols may be used to refer to storage designations, register assignments, constants, etc. For example:

| Address | Name | Operation | Argument |
|---------|------|-----------|----------|
| 101 | MAZE | DATA | H'F5' |
| 102 | | LODA,3 | MAZE |

The symbolic label "MAZE" represents the address 101. It is used in the machine instruction at address 102 to tell the assembler to build an instruction LODA,3    101. The symbolic label, in this case, is a way for the programmer to specify an address without knowing exactly what the address should be when he writes the program. In this example, assume there was a need to modify this sequence of code: a data statement was inserted between the original two statements.

| Address | Name | Operation | Argument |
|---------|------|-----------|----------|
| 99 | MAZE | DATA | H'F5'' |
| 9A,9B | | DATA | H'FE,3A' |
| 9C | | LODA,3 | MAZE |

Even though there was a program change which caused the data at MAZE to be located at address 99, the load instruction referencing the data didn't have to be rewritten because the assembler could provide the proper physical address for the symbolic address MAZE. The instruction at address 9C will be assembled as LODA,3    99.

## SYMBOLIC ADDRESSING

When writing instructions in the symbolic assembler language for the 2650, the addresses may be expressed through symbolic equivalents. The assembler will translate the symbolic address to its numeric equivalent during the assembly process.

It is good programming practice to make all address references symbolic, as this greatly eases the programmer's job in producing a working program. To make the register specification symbolic, one could equate a symbol to the register number:

```
RG3          EQU          3
             • • •
             • • •
             • • •
             • • •
             LODA,RG3     MAZE
```

## FORWARD REFERENCES

A previously defined symbol is one which has appeared in the name field before it is referenced (as above). In contrast, a forward reference is a symbolic reference to a line of code when the symbol has not yet appeared in the name field. For example:

```
             ADDA,2       COEF
             • • •
             • • •
             • • •
COEF         DATA         D'123'
```

Forward references may be used anywhere in a program with the following exceptions:

1. The register/condition field.
2. The symbolic argument fields of EQU, RES, ORG and DATA statements.

## RELATIVE ADDRESSING

The programmer may reference a memory cell either directly or via relative addressing. To refer directly to a memory cell of symbolic address MAIN, one has merely to use the name MAIN in the argument field of the referencing instruction. For example:

```
             BIRA,R2      MAIN
```

It is also possible to express the address of a memory cell symbolically if some nearby cell is symbolically assigned. For example, to load the memory cell which is 5 cells higher in memory than the cell named MAIN, one need only to refer to it as MAIN+5:

```
             LODA,2       MAIN+5
```

This later method is called relative addressing, and the relative count may be given as + or − the maximum value which can be held in one integer variable of the host computer's FORTRAN compiler.

## THE LOCATION COUNTER AND SYMBOL "$"

There is one symbolic name, "$", which is automatically defined by the assembler. This single character name is always symbolically equated to the assembler's Location Counter. Since the Location Counter is used by the assembler during the assembly process and is usually equated to the address

of the next byte to be assembled, it represents the address of the instruction or data currently being specified. For example: BCTR,3    $+5. The branch address will be interpreted by the assembler to be the address of the first byte of the branch instruction plus 5 bytes.

### HARDWARE RELATIVE ADDRESSING

When using instructions which use "hardware relative addressing" (as distinguished from relative addressing discussed earlier in this section), it is important to realize the assembler will not only evaluate the expression which is given as an operand address, but will convert it to a hardware relative address (see the Hardware Specifications manual for a description of the addressing modes). For example:

| Address | Name | Operation | Argument |
|---|---|---|---|
| 100 | SAM | LODA,R2 | PAL |
| 103 | | SUBI,R2 | -3 |
| 105 | | BIRR,R3 | SAM |
| 107 | next instruction | | |

In this code, the BIRR instruction specifies hardware relative addressing. Even though the equivalent value of the symbolic address SAM is 100, the relative addressing instruction requires a displacement relative to the address of the next sequential instruction. Therefore, the operand SAM will be evaluated as $= -$ (current location counter+length of BIRR instruction $-$ SAM) $= -(105+2-100) = -(+7) = -7$. Remember, where the hardware instruction calls for "hardware relative addressing", the expression in the operand field will be evaluated as the displacement from the address of the next sequential instruction. The value of this displacement may range from $-64$ to $+63$.

### INDIRECT ADDRESSING

The symbol "*" is used to specify indirect addressing. For example:

```
        BCTA,3      *SAM
          • • •
          • • •
          • • •
SAM     ACON        SUBR
```

In this code, the BCTA instruction specifies indirect addressing. The assembler will set the indirect bit (byte #1, bit #7) for this instruction.

### AUTO-INCREMENT AND AUTO-DECREMENT

The symbol "+" and "−" are used to specify auto-increment and auto-decrement, respectively. For example:

```
        LODA,R0     BUF,R3,+
```

In this code, which specifies auto-increment, the assembler sets bits #6 and #5 of byte #1 to "01" for this instruction. This option is specified in the instruction set tables as (,X).

# PROCESSOR INSTRUCTIONS

2650 machine instructions may be written in symbolic code. All features provided by the assembler such as symbolic addressing and constant generation may be used. The fields described below are free form and are separated by at least one blank character. The name, however, if present, must begin in logical column 1.

| LABEL | OPERATION | OPERAND | COMMENTS |
|-------|-----------|---------|----------|
| name  | opcode    | operand(s) | |

*Where:*

**LABEL FIELD**      contains an optional label which the assembler will assign as the symbolic address of the first byte of the instruction.

**OPERATION FIELD**      contains any of the 2650 processor mnemonic operation codes as detailed in Appendix A, or any Assembler Directive. This field may include an expression which specifies a register or value as required by the instruction. All symbols used in this field must have been previously defined, i.e., no symbolic forward references are allowed.

**OPERAND FIELD**      contains one or more operand elements such as indirect address indicator, operand expression, index register specification, auto-increment/auto-decrement indicator, constant specification, etc., depending on the requirements of the particular instruction.

**COMMENTS FIELD**      any characters following the argument field will be reproduced in the assembly listing without processing. The Comments Field must be separated from the argument field by at least one blank.

Note:      Refer to Appendix E for a summary of the mnemonic op-codes and see 2650 Hardware Specifications.

# DIRECTIVES TO THE 2650 ASSEMBLER

There are eleven directives which the assembler will recognize. These assembler directives, although written much like processor instructions, are simply commands to the assembler instead of to the processor. They direct the assembler to perform specific tasks during the assembly process, but have no meaning to the 2650 processor. These assembler directives are:

ORG
EQU
ACON
DATA
RES
END
EJE
PRT
SPC
TITL
PCH

## ORG   SET LOCATION COUNTER

The ORG directive sets the assembly Location Counter to the location specified. The assembler assumes an ORG 0 at the beginning of the program if no ORG statement is given.

| LABEL | OPERATION | OPERAND |
|-------|-----------|---------|
| {name} | ORG | expression |

*Where:*

name
optionally provides a symbol whose value will be equated to the specified location.

expression
when evaluated, results in a positive integer value. This value will replace the contents of the location counter, and bytes, subsequently assembled will be assigned sequential memory addresses beginning with this value. Any symbols which appear in the argument must have been previously defined.

*Examples:*

```
LARR        ORG         YORD
STAR        ORG         H'100'
```

## EQU  SPECIFY A SYMBOL EQUIVALENCE

The EQU directive tells the assembler to equate the symbol in the name field with the evaluatable expression in the argument field.

| LABEL | OPERATION | OPERAND |
|-------|-----------|---------|
| name | EQU | expression |

*Where:*

name

is the symbol which is to be assigned some value by the execution of this directive.

expression

may be resolved to zero or some integer value which is containable in the host computer's FORTRAN integer cell. If a symbol is used in the argument, it must have been previously defined.

*Examples:*

```
PAL      EQU      H'10F'
LOP2     EQU      PAL
RAMP     EQU      SLOP-3+PAL
REG1     EQU      1
```

The ACON directive tells the assembler to allocate two successive bytes of storage. The evaluated argument will be stored in the two bytes, the low order 8 bits in the second byte and the high order bits in the first byte. This directive is mainly intended to provide a double byte containing an address for use as the indirect address for any instruction executing in the indirect addressing mode.

| LABEL | OPERATION | OPERAND |
|-------|-----------|---------|
| { name } | ACON | expression |

*Where:*

name is an optional label. If specified, the name becomes the symbolic address of the first byte allocated.

expression is some expression which must resolve to a positive value or zero. If positive, the value should be no larger than that which can be contained in two bytes.

*Example:*

ASUB ACON SUBR

## DATA   DEFINES MEMORY DATA

The DATA directive tells the assembler to allocate the exact number of bytes required to hold the data specified in the argument field of this directive. Up to 16 bytes can be specified with one DATA directive, but the argument field may not extend past logical column 72.

| LABEL | OPERATION | OPERAND |
|-------|-----------|---------|
| {name} | DATA | expression |

*Where:*

name
 is an optional label. If used, the name becomes the symbolic address of the first byte allocated by the directive.

expression
 is a general constant, a self-defining constant or a symbolic address. If a symbol is specified, it must have been previously defined. A multiple constant specification in the argument field will cause a corresponding number of bytes to be allocated. Any other expression that can be resolved to a single value will result in one byte being allocated.

*Examples:*

```
PAL          DATA      LOOP
             DATA      H'03,22,FC,A1'
             DATA      +127
             DATA      D'28'
```

Note: If the expression evaluates to a value between 0 and 255 the result is an eight bit absolute binary number. DATA    +127 results in H'7F'. Also, if the expression evaluates to a value which is less than 0 the result is a 2's complement, binary number. DATA    H'-5' results in H'FB'.

## RES  RESERVE MEMORY STORAGE

The RES directive tells the assembler to reserve contiguous bytes of storage. The number of bytes so reserved is determined by the argument. The reserved bytes are not set to a known value, but rather the effect of this directive is to increment the location counter.

| LABEL | OPERATION | OPERAND |
|-------|-----------|---------|
| {name} | RES | expression |

*Where:*

name
: is an optional label. If used, the name becomes the symbolic address of the first byte allocated.

expression
: is some evaluatable expression which must resolve to some positive integer or zero. The value of this expression may not exceed the maximum positive value containable in a FORTRAN cell of the host computer. If a symbol is specified, it must have been previously defined.

*Example:*

```
LOR       RES       23
MASK      RES       LOR+5
          RES       H'1A'
```

## END END OF ASSEMBLY

The END directive informs the assembler that the last statement to be assembled has been input and the assembler may proceed with the assembly. The END directive causes the assembler to communicate the program start address to the object module.

| LABEL | OPERATION | OPERAND |
|-------|-----------|---------|
|  | END | expression |

*Where:*

expression
: may be resolved to the starting address of the program. If this parameter is not specified, the start address is set to zero.

## EJE   EJECT THE LISTING PAGE

The EJE directive tells the assembler to advance the listing to the top of the next page regardless of the line position on the current listing page.

The directive is used primarily to organize listing for documentation purposes and does not appear in the listing.

| LABEL | OPERATION | OPERAND |
|-------|-----------|---------|
|       | EJE       |         |

## PRT   PRINTER CONTROL

The PRT directive tells the assembler to resume or discontinue printing of the assembled program.

This directive is used primarily to shorten assembly time by listing only that portion of the program which the user needs to see. Only the PRT OFF will appear in the listing.

| LABEL | OPERATION | OPERAND |
|-------|-----------|---------|
|       | PRT       | $\left\{ \begin{array}{l} on \\ off \end{array} \right\}$ |

Note:   PRT is set ON at the beginning of an assembly of the assembler.

## SPC SPACE CONTROL

The SPC directive tells the assembler to skip or space a number of lines.

This directive is used primarily to organize listings for documentation purposes and does not appear in the listing.

| LABEL | OPERATION | OPERAND |
|-------|-----------|---------|
|       | SPC       | expression |

*Where:*

expression      is some evaluatable expression which must resolve to some positive integer. If the value of this expression is equal to, or greater than, the number of lines remaining on the page, the effect is the same as the EJE directive.

*Example:*

SPC             5


## TITL TITLE

The TITL directive tells the assembler to skip to the top of the next page and insert a given title into the main header.

This directive is used primarily for documentation purposes and does not appear in the listing.

| LABEL | OPERATION | OPERAND |
|-------|-----------|---------|
|       | TITL      | expression |

*Where:*

expression      is the title information not to exceed forty character positions.

*Example:*

TITL            MAIN PROGRAM SUBROUTINE

## PCH  PUNCH CONTROL

The PCH directive tells the assembler to selectively resume or discontinue the output of the load module.

This directive is used primarily to shorten assembly time when a load module is not desired or when only a portion of the load module is desired.

| LABEL | OPERATION | OPERAND |
|-------|-----------|---------|
|       | PCH       | $\left\{ \begin{array}{l} \text{on} \\ \text{off} \end{array} \right\}$ |

Note:  PCH is set ON at the beginning of an assembly by the assembler. When PCH OFF is specified, any prior load module data is output.

# THE ASSEMBLY PROCESS

The 2650 assembler translates symbolic source code into machine language instructions. The assembler examines every source statement for syntactic validity and produces the equivalent machine code for the 2650 processor.

This is a two pass assembler, which means, the entire source code is scanned twice by the assembler. On the first pass, all defined labels and their equivalent values are stored in a symbol table, the first byte of every instruction is fully determined, and some errors may be detected. During pass 2, symbolic address references are replaced by their values, errors may be detected, and a listing and load/object module is generated.

## SYMBOL TABLE

The assembler builds and maintains a symbol table during the assembly process. The symbol table contains an entry for each symbol in the assembled program. The entry consists of the symbol itself and its value. Up to 400 symbols may be used in each program assembled. If a symbol, which appears in the argument field of an instruction has never been defined (never appeared in the NAME field), the assembler will generate an error code on the listing because it is unable to resolve an undefined symbol and will place zero as the unresolved value in the object module.

## LOCATION COUNTER

The assembler maintains a memory cell which it uses as a Location Counter. This Location Counter keeps track of the address of the next byte of storage to be allocated by the assembler. During coding, the programmer may think of the Location Counter as containing the address of the first byte of the instruction being written. In this assembler, the Location Counter is also used to provide load information. This means that the addresses displayed on an assembly listing are the actual addresses which are to contain the corresponding information upon loading of the object program.

## ERROR DETECTION

During an assembly, the source program is checked for syntax errors. If errors are found, appropriate notification is given and the assembly proceeds. Although an assembled program containing errors generally will not run properly, it is considered good practice to complete the assembly to locate all errors at one time, rather than terminate it when an error is encountered.

## ERROR CODES

As shown in the listing illustration, there are three columns on the listing in which an error indication may appear. An error displayed, in the first column usually indicates that the error was in the Name Field, the second column corresponds to the Operation Field, and the third corresponds to the Argument Field. Sometimes because an error causes the assembler to view the next field incorrectly, a valid field may be flagged as an error. This is a consequence of the free format source language. A good rule is to fix errors in a particular line of code as they are discovered. In this way, erroneously flagged program errors may then be passed as valid.

The following alphabetic characters are printed in the error indicator columns and imply the corresponding message.

L — Label error. The label contains too many characters, contains invalid characters, has been previously defined, or is an invalid symbol.

O — Op-code error. The op-code mnemonic has not been recognized as a valid mnemonic.

R — Register field error. The register field expression could not be evaluated, or when evaluated, was less than 0 or greater than 3, or the register field was not found.

S — Syntax error. The instruction has violated some syntax rule.

U — Undefined symbol. There is a symbol in the argument field which has not been previously defined.

A — Argument error. The argument has been coded in such a way that it cannot be resolved to a unique value.

P — Paging error. A memory access instruction has attempted to address across a page boundary.

W — Warning. The assembler has detected a syntactically correct but unusual construction. The error will not be counted and will not inhibit the production of the object module.

## USING THE ASSEMBLER

The program is prepared by punching it into cards or otherwise transferring the program statements into a logical card image file. An ORG statement usually occurs early in the program. If no ORG appears, the assembler assumes an ORG 0 to occur before the first assembled statement. An END statement must occur as the last statement. A program written in the 2650 Symbolic Assembler Language should be preceded and possibly followed by control cards for the particular computer system which is being used. Figure 14 shows the control cards for an IBM/370 DOS system. Although the control cards may vary from system to system, the format of the actual 2650 source program will be the same in the system.

The object module produced by the Assembler during pass 2 is directed to the FORTRAN standard device #2, in this instance the card punch. The source program is read by the assembler at standard device #1, the card reader. In some systems the device assignments may be altered if desired, through assign cards. In other systems, however, the assembler must be recompiled with the device numbers desired being set in the main program module.

// JOB SPTP MC01  OLILA MICROPROCESSOR X2464
* OPERATOR - THIS PROG PUNCHES A FEW CARDS: NOTHING X2359
// ASSGN SYS004,SYS001
// DLBL OOUT,PXGO WORK,69/001
// EXTENT SYS004,,,,7505,160
// EXEC CLRUK
// UCL B=(R=U,0=912),X'00',ON,E=(X030)
// END
/*
// ASSGN SYS006,SYS001
// ASSGN SYS007,SYS001
// DLBL INSYS07,PXGO WORK,69/001
// EXTENT SYS007,,,,7505,160
// EXEC PXPTPASM

ASSU SOURCE PROGRAM

**Figure 14**

## OBJECT MODULE

The format of the object module is: The first card or card image is always all 9's.

$$bb999999999999999$$

The second and all subsequent data cards are in the following format. Logical columns (1-5) contain the load address in decimal. Each three columns (6-71) contain the data to be loaded in decimal. Each three columns represent a byte of data; columns (6-8), (9-11), (12-14), etc. Beginning at the address indicated in columns (1-5) each sequential data byte is to be loaded into sequentially ascending addresses in memory. If a '999' appears in a particular data byte position, that byte of information is to be ignored by the loader and the contents of the corresponding location is not modified.

Because there is address and data on every card image, each card image is independent. Therefore, the order of the data cards is unimportant and patch cards may be prepared manually by preparing a data card in the object module format.

The last two card images each serve a special purpose. The next to last card contains a series of '-1' punches. This card is used to signal the end of load information and has no other function.

The last card, which follows the '-1' card, contains either the start address (specified in assembler END statement) or zero in columns (1-5), the remainder of the card contains '-1' punches which have no meaning.

117

## ASSEMBLY LISTING

Figure 15 is a sample of a program listing produced by the 2650 Assembler. The following explanations are keyed to the listing.

1. Page heading — which displays the current version and level of the 2650 Assembler.

2. Line number — every assembled line is assigned a line number for the programmer's convenience.

3. Address column — The numbers in this column are equal to the value of the assembly Location Counter and indicate the address at which the first byte (B1) is to be loaded.

4. Label column — If there is a symbol in the Label Field of a line of code, the value of the label will appear in this column. For example, in line number 17 the value of the label SORT is H'0007'.

5. Data field — This field describes the data bytes which are to be stored sequentially starting at the address in the Address Column.

6. Error columns — These columns may contain the error codes as detailed elsewhere in this chapter.

7. Source code — This area of the listing reproduces the source code as it was read by the assembler.

8. Page number — Every page of the listing is numbered sequentially.

9. Cumulative errors — This field indicates the total of errors detected by the assembler during the assembly process. Warning messages (W) are not included in this total.

Figure 15. SAMPLE PROGRAM LISTING

# CHAPTER IV

# 2650 SIMULATOR

# INTRODUCTION

The 2650 Simulator is a FORTRAN program which allows a user to simulate the execution of his program without utilizing the 2650 processor.

The Simulator executes a 2650 program by maintaining its own internal FORTRAN storage registers to describe the 2650 program itself, the microprocessor registers, the ROM/RAM memory configuration, and the input data to be read dynamically from I/O devices. Multiple simulations of the same program may be executed during a single simulation run. In addition, statistical timing information may be generated.

The Simulator requires as input both the program object module produced by the 2650 Assembler and a deck of user commands. It produces a listing of the user's commands, executes the program and prints ("displays") both static and dynamic information as requested by the user's commands.

# SIMULATOR OPERATION

## GENERAL

Once the Simulator is loaded and started, it performs the following actions:

- Presets each register in simulated memory to a "HALT" instruction. Thus, if the user's program attempts to branch to some undefined area of memory, the current execution of the simulated program is terminated and only relevant data is printed.

- Reads and stores the user's commands. These commands control the performance of the Simulator during program execution. They are stored in a simulator table for reference before, during, and after execution.

- Loads the 2650 object module into simulated memory.

- Starts the simulated program. The simulated program is started at the address specified in the START command. If no START command is submitted, the program is started in the location specified in the END statement of the simulated program (see Assembler manual). If no location is specified in the END statement, the Simulator starts in location 0.

- Oversees the execution of each instruction. Before an instruction is executed, the Simulator checks the address of the instruction and the address of the referenced memory location to see if either of these addresses is referenced by any one of the user's commands. If so, the command is executed. The Simulator then executes the current instruction, updates all affected registers and retrieves the next instruction for execution.

- Terminates the simulated program. The simulation is terminated either by the execution of a "HALT" instruction, or by having executed a preset number of instructions or by having satisfied the conditions of the STOP. command.

- Once the execution of one simulation is complete, the Simulator prints any statistical timing information requested (STAT), and proceeds with the next simulation (TEND) or terminates itself (FEND).

## SIMULATED PROCESSOR STATE

The Simulator maintains a number of FORTRAN integer cells which are used to simulate the microprocessor's state, i.e. the general purpose registers,

the upper and lower program status bytes, the location counter or instruction address register (IAR), the address of the instruction referenced and the contents of the location referenced.

These simulated registers and status bits may be displayed dynamically, (INSTR., REFER., TRACE.) i.e., while the simulated program is executing. Also the general purpose registers and the status bytes may be altered dynamically (SETR., SETP.).

## SIMULATED MEMORY

The Simulator maintains a 2048 cell FORTRAN integer array which is used to simulate read-write random access memory.

It is possible to configure parts of this memory into a ROM-RAM environment by using the SROM Command. If part of the simulated memory is set to Read-Only and an instruction attempts to store data into that memory segment, the Simulator bypasses storing the data, prints a warning message and continues with the next program instruction.

Using Simulator commands, the user may change parts of memory before the program executes (PATCH) and he may display parts of memory dynamically (DUMP.).

The simulated memory is smaller in many cases than the total memory size of the user's physical system. This restriction encourages the construction of modular programs. Because the simulated memory is smaller than a 2650 page, it is not possible to fully test programs which utilize the 2650 paging system, i.e., programs larger than 8192 bytes.

## SIMULATED INPUT/OUTPUT INSTRUCTIONS

The Simulator maintains a 200-byte First In, First Out (FIFO) buffer to store the data read from a simulated input device. This buffer must be preset by the user command, INPUT.

When any 2650 input instruction is simulated (REDE, REDC, REDD), the Simulator accesses the buffer. If there is data in the buffer, the next byte of data is inserted in the simulated register specified by the input instruction. If the buffer contents have been exhausted, a warning message is displayed on the simulator listing.

To simulate the execution of any 2650 output instruction (WRTE, WRTC, WRTD), the Simulator takes the data byte from the register specified in the output instruction and displays it along with the address of the output instruction.

# USER COMMANDS

## GENERAL

The 2650 Simulator accepts commands which specify how the program is to run and what data is to be recorded.

In any one Simulator run, the user may specify that his program be executed any number of times. The user submits a new set of commands for each execution. The final command set is followed by a final end card (FEND), while all prior command sets are terminated with a temporary end card (TEND) (Illust. III-1).
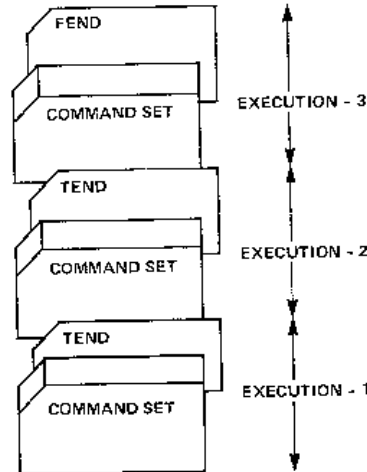


Figure 16. THREE SETS OF COMMANDS

Within any one command set, the user may specify:

- That the program execution start at a specific memory location (START).

- That the execution of the program be complete either when the number of instructions executed equals a specified number (LIMIT) or when the instruction at a specific address executes (STOP.) or when the simulated program itself executes a "HALT" instruction.

- That statistics be displayed at the end of execution (STAT). The Simulator accumulates a count of the total number of instructions executed, the number of each type of instruction executed, and the total number of 2650 machine cycles expended. This information provides a measure of efficiency by indicating how many 1-, 2-, or 3-byte instructions were executed and may be used to calculate program timings.

- That certain areas of simulated memory be designated as Read-Only (SROM) and are therefore inaccessible to any memory write operation.

- That the contents of memory be initialized with specific data (PATCH).

- That a FIFO (First In, First Out) buffer be used to simulate data read from I/O devices (INPUT).

- That the processor state be recorded whenever a specific memory location executes (INSTR.), whenever a specific memory location is referenced (REFER.), or whenever any instruction executes which lies within a specified range of memory addresses (TRACE.). The processor state consists of the location counter, the instruction referenced and its contents, the upper and the lower program status bytes, and the contents of all the general purpose registers.

- That an area of memory be dumped whenever an instruction at a specific memory location executes (DUMP.).

- That certain general purpose registers (SETR.) or the program status bytes (SETP.) be set dynamically, i.e., whenever a specific memory location executes.

- That comments (**) be interspersed between control cards.

Some of these commands execute dynamically, i.e., when an instruction at a specific memory location executes or when that location is referenced. Since the simulator storage capacity limits the total number of locations which may be retained simultaneously (while a program is executing), a total of 30 memory locations may be specified on all the "dynamic" commands submitted for any one execution, i.e., in any one command set. These dynamic commands are identified by a trailing period (.), e.g., "STOP.". This period is treated as a field separator, i.e., it is not treated as part of the command name by the Simulator and is therefore optional. The description for each dynamic command identifies which of its parameters count toward the 30 "dynamic" command limit, i.e., the limit of 30 memory locations.

In addition, the number of DUMP. commands is limited to five (5); the number of SETR. commands is limited to four (4); the number of SETP. commands is limited to two (2); and the number of data read on all INPUT cards in one command set is limited to 200.

All "dynamic commands" are executed *before* the simulated instruction is executed.

For those commands which accept only one set of parameters (LIMIT, SROM, START) only the last set of parameters encountered is used.

## COMMAND FORMATS

Figure 17 contains a list of the commands, their parameters and a brief description of the commands themselves. In addition, the Simulator treats as a comment card, any card with two consecutive asterisks (**) starting in column 1.

The Simulator accepts information in card image form. The entire card is read in FORTRAN "A" format. A command must be complete on one card as continuation cards are not allowed. Comments may appear in any order within a command set.

The command name starts in column 1 and must appear as shown, except for the optional period.

The field of characters which lies between the command name and its parameters or between the parameters themselves is called a field separator. A field separator may contain any number of characters, but none of these characters may be hexadecimal characters (0-9, A-F). For the sake of clarity in all the examples, the following field separators are used to indicate the following functions:

| FIELD SEPARATOR | FUNCTION |
| --- | --- |
| | Identifies a command which counts toward the "dynamic" command limit. |
| blank (s) | Separate a command from its parameters. |
| ( ) | Encloses optional parameters. |
| ; | Separates one set of parameters from another. |
| , | Separates one parameter from another within a set of parameters. |
| ; . . . ; | Indicates that multiple parameters or sets of parameters are legal. If a period flags a command, each of its parameter sets counts toward the "dynamic" command limit. E.g., the following sets of commands are identical: |

|     |       |          |
| --- | ----- | -------- |
| 1.  | INST. | 100      |
|     | INST. | 200      |
| 2.  | INST. | 100; 200 |

The parameters themselves must be hexadecimal numbers (0-9, A-F). The following labels identify parameters in Illustration III-2:

| LOC | Location or address of an instruction which is to be executed or the address of data which is to be referenced. |
| --- | --- |
| NO | A number of data, e.g., the total number of instructions to be executed. |
| FWA | First Word Address of some area of memory. |
| LWA | Last Word Address of some area of memory. |
| VALUE | The value to which some location is to be set. |
| R0, R1 . . . R6 | General Purpose Registers 0-6. |
| PSL | Identifies Lower Program Status Byte. |
| PSU | Identifies Upper Program Status Byte. |

| COMMAND NAME | PARAMETERS | DESCRIPTION |
|---|---|---|
| DUMP. | LOC, FWA-LWA (; . . . ;LOC, FWA-LWA) | Display the area of memory, FWA-LWA, whenever the instruction at LOC executes. |
| FEND | None | Execute the last simulation and terminate the entire run. |
| INPUT | VALUE(; . . . ;VALUE) | Define the data to be read by simulated I/O instructions. |
| INSTR. | LOC(; . . . ;LOC) | Display the processor registers whenever the instruction at LOC executes. |
| LIMIT | NO | Specify the total number of instructions executed. |
| PATCH | LOC,VALUE(; . . . ;LOC,VALUE) | Initialize each memory location, LOC, to VALUE. |
| REFER. | LOC(; . . . ;LOC) | Display the processor register whenever the instruction at LOC is referenced by another instruction. |
| SETP. | LOC(,PSL=VALUE)   (,PSU=VALUE) | Set the program status byte (lower and/or upper) to VALUE whenever the instruction at LOC executes. |
| SETR. | LOC(,R0=VALUE). . .(R6=VALUE) | Set the general purpose registers to VALUE whenever the instruction at LOC executes. |
| SROM | FWA-LWA | Specify the boundaries of Read-Only Memory. |
| START | LOC | Start the simulated program execution at LOC. |
| STAT | None | Display instruction statistics at end of program execution. |
| STOP. | LOC(; . . . ;LOC) | Terminate the program execution when the instruction at LOC executes. |
| TEND | None | Execute the last simulation and prepare to read the User Commands for the next simulation. |
| TRACE. | FWA-LWA(; . . . ;FWA-LWA) | Display the processor registers whenever an instruction executes, which lies within the area of memory, FWA-LWA. |

**Figure 17. COMMAND SUMMARY**

## COMMAND DESCRIPTIONS

The following command descriptions are alphabetized by command name. As previously discussed all parameters are entered in hexadecimal notation (0-9, A-F). All address parameters (LOC, FWA, LWA) are limited to the size of simulated memory.

## DUMP.    DUMP SIMULATED MEMORY

This command causes the Simulator to display selected portions of memory whenever the location counter matches LOC.

Each LOC counts as one "dynamic" command. The total number of "dynamic" commands is limited to thirty (30). The total number of LOC's submitted in DUMP. commands is limited to five (5).

DUMP.    LOC,FWA-LWA(; . . . ;LOC,FWA-LWA)

*Where:*      DUMP. is the command name.

LOC is the address of the 2650 instruction at which the dump occurs.

FWA is the first address of the area to be dumped.

LWA is the last address of the area to be dumped. LWA must be larger than FWA.

*Example:*      DUMP.    5A,0-3FF   100-11A-21A
DUMP.    E0-400-4FF

Note:   More data may be dumped than was specified since the FWA dumped always has a least significant digit of 0, e.g. 30, 100, etc. Similarly, LWA always has a least significant digit of F, e.g. 3F, 10F, etc.

## FEND    FINAL END COMMAND

This command signals the Simulator that the preceding commands complete the directives for the final simulator run. After FEND is read, the Simulator performs the last simulation and comes to its final termination.

FEND

*Where:*    FEND — specifies the command name.

*Example:*    START    1A
TRACE    0, 100
TEND
START    AA
PATCH    11, C2
FEND

## INPUT    DEFINE DATA FOR INPUT

This command loads data into a FIFO storage buffer from which the same data is used to supply I/O instructions with input data. The first data point specified becomes the first one accessed by a 2650 read instruction. The last point specified becomes the last one accessed. Should the buffer become empty during the simulated execution, an error message is printed, the input register remains unchanged and the simulation continues.

Any number of these command cards may be submitted as long as the total number of data specified in one run does not exceed the size of the FIFO storage buffer (200).

INPUT    VALUE(; . . . ;VALUE)

*Where:*    INPUT — specifies the command name.

VALUE — specifies a 2-digit hexadecimal value.

*Example:*    INPUT    0, 1, 2, 3, 10, 1A, FF

## INSTR.    INSTRUCTION TRACE

This command sets a break point at the specified address. When the instruction at this address executes, the Simulator prints out the internal state of the simulated processor. The break point occurs before the instruction is executed.

Each address specified in an INSTR. command counts as one "dynamic" command.

<div align="center">

INSTR.    LOC(; . . . ;LOC)

</div>

*Where:*      INSTR. — specifies the command.

LOC — specifies the address for a break point. The address must be within simulated memory.

*Example:*      INSTR.    1CE, 1A, 22
INSTR.    123-200-5E
INSTR.    74

## LIMIT    LIMIT THE NUMBER OF INSTRUCTIONS EXECUTED

This command determines how many instructions will be executed. If the number given in the LIMIT command is exceeded before the instruction specified by a STOP. command executes or before a 2650 HALT instruction is simulated, the Simulator terminates the current program operation.

Without this command, the Simulator assumes a limit of $1000_{10}$ instructions. The maximum LIMIT which may be specified is determined by the maximum integer constant of the FORTRAN compiler used.

<div align="center">

LIMIT    NO

</div>

*Where:*      LIMIT — specifies the command.

NO — is a number which determines the maximum number of instructions to be executed.

*Example:*      LIMIT    200
LIMIT    2F

## PATCH    PATCH SIMULATED MEMORY

This command alters the contents of memory before a simulation run. It may be used to alter the contents of any byte in memory and overrides load information in the object module for the duration of one simulation run.

Any number of these commands may be given in a simulator command stream.

PATCH    LOC,VALUE(; . . . ;LOC,VALUE)

*Where:*    PATCH — specifies the command.

LOC — specifies the simulated memory address which is to be changed.

VALUE — specifies a 2-digit hexadecimal number to be stored at LOC.

*Example:*    PATCH    0, 1F   1, 0   2. 5E
PATCH    102, EE

## REFER.    MEMORY REFERENCE TRACE

This command causes a break point to occur whenever one of the specified addresses is referenced by a simulated instruction. During the break point, the Simulator prints out the internal state of the simulated processor. The data byte of immediate addressing instructions is handled like an ordinary operand address.

Each address specified in a REFER. command counts as one "dynamic" command.

REFER.    LOC(;LOC. . . ;LOC)

*Where:*    REFER. — specifies the command.

LOC — specifies the effective operand address for a break point. The address must be within simulated memory.

*Example:*    REFER.    3FF/21/18E
REFER.    200
REFER.    5, 50, 22F

## SETP.    SET PROGRAM STATUS BYTE

The SETP. command dynamically alters the upper and/or the lower program status bytes. The specified program status byte is set when the address parameter supplied in the command, LOC, equals the location counter.

A SETP. command must set at least one program status byte. Up to two SETP. commands may be given in a simulator command stream. Each LOC submitted counts as one "dynamic" command.

The PSL and PSU may be entered in any order.

SETP.    LOC(,PSL=VALUE)   (,PSU=VALUE)

*Where:*    SETP. — specifies the command.

LOC — specifies the simulated execution address where the program status byte is to be set.

PSL — specifies that a value is to be entered into PSL.

PSU — specifies that a value is to be entered into PSU.

VALUE — specifies the 2-digit hexadecimal value to be entered into the program status byte.

*Example:*    SETP.    5A PSL=05
             SETP.    10E, PSL=01    PSU=00

## SETR.    SET GENERAL PURPOSE REGISTER

This command dynamically sets the general purpose registers during simulated program execution. Using this command, any or all of the general purpose registers can be set when the location counter value is equal to the address parameter, LOC, supplied in this command.

A SETR. command without parameters is not permitted. Up to four SETR. commands may be given in a simulator command stream. Each LOC counts as one "dynamic" command.

Register identifiers may appear in any order.

SETR.    LOC(,R0=VALUE). . .(,R6=VALUE)

*Where:*    SETR. — specifies the command.

LOC — specifies the simulated execution address where the registers are to be set.

R0 — indicates the general purpose register to be set. R0
R1    always refers to general purpose register 0. R1, R2, and
R2    R3 specify the registers in register bank zero. R4, R5
R3    and R6 specify R1, R2, and R3 in register bank one.
R4
R5
R6

VALUE — specifies the 2-digit hexadecimal value to be stored in the selected register.

*Example:*    SETR.    10A  R1=3F, R2=00, R3=5
.            SETR.    2F3  R0=FF, R5=00

## SROM    DEFINE THE BOUNDARIES OF READ ONLY MEMORY

This command allows the user to simulate a Read Only/Read Write Memory environment. Whenever a 2650 instruction attempts to store data in the area defined as Read Only, a warning message is printed on the simulation listing. The data is not actually stored, but the simulation run continues.

<center>SROM    FWA-LWA</center>

*Where:*      SROM — specifies the command.

FWA — specifies the first address of the simulated ROM area.

LWA — specifies the last address of the simulated ROM area. LWA must be greater in value than the FWA. The addresses specified are inclusive.

*Example:*      SROM    100-FF


## START    START SIMULATION

This command specifies the address at which simulated execution begins. The address specified in the START command supersedes the start address in the load object module. The start address in the load object module is set by an END statement during program assembly and is used by the Simulator if no START command is given (see the 2650 Assembler Language Manual for the END statement).

<center>START    LOC</center>

*Where:*      START — specifies the command.

LOC    — specifies a start address for the program to be simulated.

*Example:*      START    10A
                  START    2

## STAT    DISPLAY INSTRUCTION STATISTICS

This command causes a list of 2650 instructions with the number of times each was executed to be printed out at the end of the simulation run.

STAT

*Where:*            STAT — specifies the command.


## STOP.    STOP SIMULATED EXECUTION

This command terminates the current simulated instruction execution when the location counter matches the command argument, LOC.

Each LOC counts as one "dynamic" command.

STOP.    LOC(; . . . ;LOC)

*Where:*            STOP. — specifies the command.

LOC — specifies the instruction address at which simulated execution ceases.


## TEND    TEMPORARY END COMMAND

This command signals the Simulator that the preceding commands complete the directives for a simulator run. After the TEND is read, the Simulator begins simulated execution of the 2650 program. Because TEND is a temporary end, the Simulator assumes that there is another command stream following it. The last command stream in a simulation run must be terminated with a FEND (final end) command.

TEND

*Where:*            TEND — specifies the command.

*Example:*          PATCH    01, 15    0A, FF
                    TEND
                    START    100
                    PATCH    01, E2    0A, FF
                    FEND

## TRACE.    TRACE PROGRAM FLOW

This command causes break points to occur at each instruction within an area of memory. The user specifies two addresses. If the simulated processor accesses an instruction at an address that falls between the specified addresses, the Simulator prints out the internal state of the simulated processor.

Each set of FWA,LWA counts as one "dynamic" command.

TRACE.    FWA-LWA(; . . . ;FWA-LWA)

*Where:*    TRACE. — specifies the command.

FWA — specifies from what address the trace is in effect.

LWA — specifies to what address the trace is in effect. LWA must be larger in value than FWA. The addresses specified are inclusive.

*Example:*    TRACE.    0-15F, 250-3FF
TRACE.    1-A, 3FF-40A
TRACE.    10-1A    50-5A    60-7A

# SIMULATOR DISPLAY (LISTING)

As the Simulator reads each command set, it prints the card images of the command set and then executes the program. During program execution the following commands result in some form of display:

> DUMP.
> INSTR.
> REFER.
> TRACE.

DUMP. results in the display of an entire area of memory while the last three commands result in some form of trace, i.e., a display of the processor state:

Instruction address register (IAR) or location counter
Instruction executed (INST)
Instruction referenced or effected (EADDR)
Contents of the instruction referenced or effected (EADDR)
Program status byte upper (PSU)
Program status byte lower (PSL)
General purpose registers (R0, R1, R2, R3, R4, R5, R6)

Figures 18 through 21 contain the printout or display output from one Simulator run. Figure 18 shows the first command set, which contains commands to:

- Start at location 0 (START)
- Initialize locations 55-5F, locations 61-6B and location 19 (PATCH)
- Dump locations 55-77 whenever either location 0 or location 3 executes (DUMP)
- Trace locations 14-1A (TRACE)

Figures 18 and 19 show the results of the first command set:

- A dump of locations 55-77. Note that a larger area is dumped than was specified.
- 30 traces
- A final dump of locations 55-77

When the program execution for the first command set is complete, the Simulator reports:

- The number of machine cycles executed
- The number of instructions executed

Figure 20 shows the second command set. It is exactly the same as the first command set except that it initializes locations 12 and 33 instead of location 19.

The output of the second command set is just like the output of the first command set except that it results in 33 traces, not 30.

```
STACT  05
PATCH  55,0 56,1 57,2 56,5 04,3
PATCH  58,5 59,4 60,7 50,2 5E,1
PATCH  5E,0
FAILE  61,5
PATCH  62,0 63,0 64,1 65,1 66,2
BATCH  67,2 66,3 69,8 6A,2 6B,1
DUMP   3,55,77
DUMP   0,55,77
PATCH  19,55
TRACE  19,15
END
```

```
COMMAND DUMP
E050    17 07 15 1B 85 0A 02 22 02 09 00 24 03 02 01 00
0060    80 00 00 00 01 81 03 42 05 6B 02 01 40 40 40 40
0070    40 40 40 50 40 60 00 0B 40 40 40 40 40 40 40 40
```

| TRACE COMMAND | | | | | | |
|---|---|---|---|---|---|---|
| IAR | INST | | EADDR (EADDR) | PSBU PSBL | R0 R1 R2 R3 R4 R5 R6 | |
| 0019 | LCDA,0 | 0061,3,- | 0C6A 0001 | 01 00 | 00 0A 00 0B 00 00 00 | |
| TRACE COMMAND | | | | | | |
| IAR | INST | | EADDR (EADDR) | PSBU PSBL | R0 R1 R2 R3 R4 R5 R6 | |
| 0017 | ACDA,0 | 0055,1 | 0C5F 0000 | 01 40 | 01 0A 00 0A 00 00 00 | |
| TRACE COMMAND | | | | | | |
| IAR | INST | | EADDR (EADDR) | PSBU PSBL | R0 R1 R2 R3 R4 R5 R6 | |
| 001A | CCNI,0 | 0A | 0C1B 000A | 01 40 | C1 0A 00 0A 00 00 00 | |
| TRACE COMMAND | | | | | | |
| IAR | INST | | EADDR (EADDR) | PSBU PSBL | R0 R1 R2 R3 R4 R5 R6 | |
| 0014 | LCDA,0 | 0061,3,- | 0C6A 0002 | 01 80 | 01 09 00 0A 00 00 00 | |
| TRACE COMMAND | | | | | | |
| IAR | INST | | EADDR (EADDR) | PSBU PSBL | R0 R1 R2 R3 R4 R5 R6 | |
| 0017 | ACDA,0 | 0055,1 | 0C5E 0001 | 01 40 | 02 09 00 00 00 00 00 | |
| TRACE COMMAND | | | | | | |
| IAR | INST | | EADDR (EADDR) | PSBU PSBL | R0 R1 R2 R3 R4 R5 R6 | |
| 001A | CCNI,0 | 0A | 0C19 000A | 01 40 | C3 09 00 09 00 00 00 | |
| TRACE COMMAND | | | | | | |
| IAR | INST | | EADDR (EADDR) | PSBU PSBL | R0 R1 R2 R3 R4 R5 R6 | |
| 0014 | LCDA,0 | 0061,3,- | 0C6F 0003 | 01 80 | 03 08 00 09 00 00 00 | |
| TRACE COMMAND | | | | | | |
| IAR | INST | | EADDR (EADDR) | PSBU PSBL | R0 R1 R2 R3 R4 R5 R6 | |
| 0017 | ADDA,0 | 0055,1 | 0C50 0007 | 01 40 | 0B 08 00 08 00 00 00 | |
| TRACE COMMAND | | | | | | |
| IAR | INST | | EADDR (EADDR) | PSBU PSBL | R0 R1 R2 R3 R4 R5 R6 | |
| 001A | CCNI,0 | 0A | 0C1A 000A | 01 40 | 0A 08 00 08 00 00 00 | |
| TRACE COMMAND | | | | | | |
| IAR | INST | | EADDR (EADDR) | PSBU PSBL | R0 R1 R2 R3 R4 R5 R6 | |
| 0014 | LCDA,0 | 0061,3,- | 0C68 0008 | 01 80 | 0C 07 00 08 00 00 00 | |
| TRACE COMMAND | | | | | | |
| IAR | INST | | EADDR (EADDR) | PSBU PSBL | R0 R1 R2 R3 R4 R5 R6 | |
| 0017 | ACDA,0 | 0055,1 | 0C5C 0000 | 01 61 | 09 07 00 07 00 00 00 | |

| TRACE COMMAND | | | | | | |
|---|---|---|---|---|---|---|
| IAR | INST | | EADDR (EADDR) | PSBU PSBL | R0 R1 R2 R3 R4 R5 R6 | |
| 001A | CCNI,0 | 0A | 001F 0004 | 01 40 | 0C 07 00 07 00 00 00 | |
| TRACE COMMAND | | | | | | |
| IAR | INST | | EADDR (EADDR) | PSBU PSBL | R0 R1 R2 R3 R4 R5 R6 | |
| 0014 | LCDA,0 | 0061,3,- | 0067 0002 | 01 61 | C2 06 00 07 00 00 00 | |
| TRACE COMMAND | | | | | | |
| IAR | INST | | EADDR (EADDR) | PSBU PSBL | R0 R1 R2 R3 R4 R5 R6 | |
| 0017 | ACDA,0 | 0055,1 | 0C5B 0004 | 01 61 | 02 06 00 06 00 00 00 | |
| TRACE COMMAND | | | | | | |
| IAR | INST | | EADDR (EADDR) | PSBU PSBL | R0 R1 R2 R3 R4 R5 R6 | |
| 001A | CCNI,0 | 0A | 0C1B 000A | 01 40 | 06 06 00 06 00 00 00 | |
| TRACE COMMAND | | | | | | |
| IAR | INST | | EADDR (EADDR) | PSBU PSBL | R0 R1 R2 R3 R4 R5 R6 | |
| 0014 | LCDA,0 | 0061,3,- | 0C66 0003 | 01 80 | 06 05 00 06 00 00 00 | |
| TRACE COMMAND | | | | | | |
| IAR | INST | | EADDR (EADDR) | PSBU PSBL | R0 R1 R2 R3 R4 R5 R6 | |
| 0017 | ACDA,0 | 0055,1 | 0C54 0005 | 01 40 | 03 05 00 05 00 00 00 | |
| TRACE COMMAND | | | | | | |
| IAR | INST | | EADDR (EADDR) | PSBU PSBL | R0 R1 R2 R3 R4 R5 R6 | |
| 001A | CCNI,0 | 0A | 0C1B 000A | 01 40 | 08 05 00 05 00 00 00 | |
| TRACE COMMAND | | | | | | |
| IAR | INST | | EADDR (EADDR) | PSBU PSBL | R0 R1 R2 R3 R4 R5 R6 | |
| 0014 | LCDA,0 | 0061,3,- | 0065 0001 | 01 80 | 0B 04 00 05 00 00 00 | |
| TRACE COMMAND | | | | | | |
| IAR | INST | | EADDR (EADDR) | PSBU PSBL | R0 R1 R2 R3 R4 R5 R6 | |
| 0017 | ACDA,0 | 0055,1 | 0C5E 0003 | 01 40 | 04 04 00 04 00 00 00 | |
| TRACE COMMAND | | | | | | |
| IAR | INST | | EADDR (EADDR) | PSBU PSBL | R0 R1 R2 R3 R4 R5 R6 | |
| 001A | CCNI,0 | 0A | 0C1B 000A | 01 40 | 04 04 00 04 00 00 00 | |
| TRACE COMMAND | | | | | | |
| IAR | INST | | EADDR (EADDR) | PSBU PSBL | R0 R1 R2 R3 R4 R5 R6 | |
| 0014 | LCDA,0 | 0061,3,- | 0C64 0001 | 01 80 | 04 03 00 04 00 00 00 | |
| TRACE COMMAND | | | | | | |
| IAR | INST | | EADDR (EADDR) | PSBU PSBL | R0 R1 R2 R3 R4 R5 R6 | |
| 0017 | ACDA,0 | 0055,1 | 0C58 0002 | 01 40 | C1 03 00 03 00 00 00 | |
| TRACE COMMAND | | | | | | |
| IAR | INST | | EADDR (EADDR) | PSBU PSBL | R0 R1 R2 R3 R4 R5 R6 | |
| 001A | CCNI,0 | 0A | 0C1B 0004 | 01 40 | 03 03 00 03 00 00 00 | |
| TRACE COMMAND | | | | | | |
| IAR | INST | | EADDR (EADDR) | PSBU PSBL | R0 R1 R2 R3 R4 R5 R6 | |
| 0014 | LCDA,0 | 0061,3,- | 0C63 0000 | 01 80 | 01 02 00 03 00 00 00 | |
| TRACE COMMAND | | | | | | |
| IAR | INST | | EADDR (EADDR) | PSBU PSBL | R0 R1 R2 R3 R4 R5 R6 | |
| 0017 | ACDA,0 | 0055,1 | 0C51 0002 | 01 00 | 00 02 00 02 00 00 00 | |
| TRACE COMMAND | | | | | | |
| IAR | INST | | EADDR (EADDR) | PSBU PSBL | R0 R1 R2 R3 R4 R5 R6 | |
| 001A | CCNI,0 | 0A | 0C1B 000A | 01 40 | 02 02 00 02 00 00 00 | |
| TRACE COMMAND | | | | | | |
| IAR | INST | | EADDR (EADDR) | PSBU PSBL | R0 R1 R2 R3 R4 R5 R6 | |
| 0014 | LCDA,0 | 0061,3,- | 0C62 0000 | 01 80 | 02 01 00 02 00 00 00 | |
| TRACE COMMAND | | | | | | |
| IAR | INST | | EADDR (EADDR) | PSBU PSBL | R0 R1 R2 R3 R4 R5 R6 | |
| 0017 | ACDA,0 | 0055,1 | 0C46 0001 | 01 00 | 00 01 00 01 00 00 00 | |

Figure 18.   FIRST COMMAND SET

```
TRACE COMMAND
IAR         INST
001A    COPY,D     CA              EADCR 1EADCR1    PSBU PSBL    R0 R1 R2 R3 R4 R5 R6
COMMAND DUMP                        CC1B    000A      01   40    01 01 00..01 00 00 00
0050    17 07 15 16 05 03 01 02 02 03 05 04 C3 02 01 00
0060    00 00 01 02 03 04 28 C6 C2 C0 03 01 40 40 40 40
0070    40 40 40 40 40 40 C0 C0 C0 40 40 40 40 40 40 40
```

NO. OF MACHINE CYCLES EXECUTED =      232


NO. OF INSTRUCTIONS EXECUTED =      73


**Figure 19.   FIRST COMMAND SET, Cont.**

```
START 00
PATCH  55,0 56,1 57,2 58,2 59,3
PATCH  5A,5 59,4 5C,3 50,2 5E,1
PATCH  5F,0
PATCH  61,0
PATCH  62,0 63,0 64,1 65,1 66,3
PATCH  67,2 68,9 69,8 6A,2 6F,1
DUMP   3,55,77
DUMP   0,55,77
TRACE  14,14
PATCH  33,0B
PATCH  12,0B
FENC
```

```
COMMAND DUMP
0055   17 07 15 1B 65 00 01 02 07 C3 05 04 C3 02 01 00
0060   00 00 00 00 01 01 C3 D9 0A 02 01 40 40 40 40
007C   40 40 40 40 40 40 40 C0 00 43 40 40 40 40 40
```

```
TRACE COMMAND
IAR        INST              EADDR (EADDR)   PSBU PSBL   R0 R1 R2 R3 R4 R5 R6
0014    LCDA,0    0061,3,-   006B   0001      01   08    00 0B 00 0B 00 00 00
TRACE COMMAND
IAR        INST              EADDR (EADDR)   PSBU PSBL   R0 R1 R2 R3 R4 R5 R6
0017    ADDA,0    0054,1     005F   0000      01   48    01 0B 00 0A 00 00 00
TRACE COMMAND
IAR        INST              EADDR (EADDR)   PSBU PSBL   R0 R1 R2 R3 R4 R5 R6
0014    CCMI,0    0A         001B   0C04      01   48    01 0B 00 04 00 00 00
TRACE COMMAND
IAR        INST              EADDR (EADDR)   PSBU PSBL   R0 R1 R2 R3 R4 R5 R6
0014    LCDA,0    0061,3,-   006A   0002      01   88    01 0A 00 0A 00 00 00
TRACE COMMAND
IAR        INST              EADDR (EADDR)   PSBU PSBL   R0 R1 R2 R3 R4 R5 R6
001F    ADDA,0    0054,1     005F   0001      01   48    02 0A 00 09 00 00 00
TRACE COMMAND
IAR        INST              EADDR (EADDR)   PSBU PSBL   R0 R1 R2 R3 R4 R5 R6
0014    CCMI,0    0A         001B   0004      01   48    03 0A 00 09 00 00 00
TRACE COMMAND
IAR        INST              EADDR (EADDR)   PSBU PSBL   R0 R1 R2 R3 R4 R5 R6
0014    LCDA,0    0061,3,-   0069   0006      01   88    03 09 00 09 00 00 00
TRACE COMMAND
IAR        INST              EADDR (EADDR)   PSBU PSBL   R0 R1 R2 R3 R4 R5 R6
0017    ADDA,0    0054,1     005D   0002      01   48    0B 09 00 08 00 00 00
TRACE COMMAND
IAR        INST              EADDR (EADDR)   PSBU PSBL   R0 R1 R2 R3 R4 R5 R6
0014    CCMI,0    0A         001B   0004      01   48    03 09 00 08 00 00 00
TRACE COMMAND
IAR        INST              EADDR (EADDR)   PSBU PSBL   R0 R1 R2 R3 R4 R5 R6
0014    LCDA,0    0061,3,-   0068   0009      01   29    09 08 00 08 00 00 00
TRACE COMMAND
IAR        INST              EADDR (EADDR)   PSBU PSBL   R0 R1 R2 R3 R4 R5 R6
0017    ADDA,0    0054,1     005C   0003      01   69    03 08 00 07 00 00 00
```

```
TRACE COMMAND
IAR        INST              EADDR (EADDR)   PSBU PSBL   R0 R1 R2 R3 R4 R5 R6
0014    CCMI,0    0A         001B   000A      01   48    0D 08 00 07 00 00 00
TRACE COMMAND
IAR        INST              EADDR (EADDR)   PSBU PSBL   R0 R1 R2 R3 R4 R5 R6
0014    LCDA,0    0061,3,-   0067   0007      01   69    03 07 00 07 00 00 00
TRACE COMMAND
IAR        INST              EADDR (EADDR)   PSBU PSBL   R0 R1 R2 R3 R4 R5 R6
0017    ADDA,0    0054,1     005B   0004      01   69    02 07 00 06 00 00 00
TRACE COMMAND
IAR        INST              EADDR (EADDR)   PSBU PSBL   R0 R1 R2 R3 R4 R5 R6
001A    CCMI,0    0A         001B   0004      01   48    07 07 00 06 00 00 00
TRACE COMMAND
IAR        INST              EADDR (EADDR)   PSBU PSBL   R0 R1 R2 R3 R4 R5 R6
0014    LCDA,0    0061,3,-   0066   0003      01   88    07 06 00 06 00 00 00
TRACE COMMAND
IAR        INST              EADDR (EADDR)   PSBU PSBL   R0 R1 R2 R3 R4 R5 R6
0017    ADDA,0    0054,1     005A   0005      01   48    03 06 00 05 00 00 00
TRACE COMMAND
IAR        INST              EADDR (EADDR)   PSBU PSBL   R0 R1 R2 R3 R4 R5 R6
001A    CCMI,0    0A         001B   000A      01   48    0B 06 00 05 00 00 00
TRACE COMMAND
IAR        INST              EADDR (EADDR)   PSBU PSBL   R0 R1 R2 R3 R4 R5 R6
0014    LCDA,0    0061,3,-   0065   0001      01   88    0B 05 00 05 00 00 00
TRACE COMMAND
IAR        INST              EADDR (EADDR)   PSBU PSBL   R0 R1 R2 R3 R4 R5 R6
0017    ADDA,0    0054,1     0059   0003      01   48    0B 05 00 04 00 00 00
TRACE COMMAND
IAR        INST              EADDR (EADDR)   PSBU PSBL   R0 R1 R2 R3 R4 R5 R6
001A    CCMI,0    0A         001B   0004      01   48    04 05 00 04 00 00 00
TRACE COMMAND
IAR        INST              EADDR (EADDR)   PSBU PSBL   R0 R1 R2 R3 R4 R5 R6
0014    LCDA,0    0061,3,-   0064   0001      01   88    04 04 00 04 00 00 00
TRACE COMMAND
IAR        INST              EADDR (EADDR)   PSBU PSBL   R0 R1 R2 R3 R4 R5 R6
0017    ADDA,0    0054,1     005B   0002      01   48    01 04 00 03 00 00 00
TRACE COMMAND
IAR        INST              EADDR (EADDR)   PSBU PSBL   R0 R1 R2 R3 R4 R5 R6
001A    CCMI,0    0A         001B   0C0A      01   48    03 04 00 03 00 00 00
TRACE COMMAND
IAR        INST              EADDR (EADDR)   PSBU PSBL   R0 R1 R2 R3 R4 R5 R6
0014    LCDA,0    0061,3,-   0063   0000      01   88    03 03 00 03 00 00 00
TRACE COMMAND
IAR        INST              EADDR (EADDR)   PSBU PSBL   R0 R1 R2 R3 R4 R5 R6
0017    ADDA,0    0054,1     0057   0002      01   08    00 03 00 02 00 00 00
TRACE COMMAND
IAR        INST              EADDR (EADDR)   PSBU PSBL   R0 R1 R2 R3 R4 R5 R6
001A    CCMI,0    0A         001B   0006      01   48    02 03 00 02 00 00 00
TRACE COMMAND
IAR        INST              EADDR (EADDR)   PSBU PSBL   R0 R1 R2 R3 R4 R5 R6
0014    LCDA,0    0061,3,-   0062   0000      01   88    02 02 00 02 00 00 00
TRACE COMMAND
IAR        INST              EADDR (EADDR)   PSBU PSBL   R0 R1 R2 R3 R4 R5 R6
0017    ADDA,0    0054,1     0056   0001      01   08    02 02 00 01 00 00 00
```

Figure 20.   SECOND COMMAND SET

142

```
TRACE COMMAND
IAR        INST                    EADDR  (EADDR)      PSBU PSBL      R0 R1 R2 R3 R4 R5 R6
001A       CCMI,0    0A            001B   000A          01   48      01 02 00 01 00 00 00
TRACE COMMAND
IAR        INST                    EADDR  (EADDR)      PSBU PSBL      R0 R1 R2 R3 R4 R5 R6
0014       LCDA,0    0061,3,-      0061   0000          01   68      02 01 00 01 00 00 00
TRACE COMMAND
IAR        INST                    EADDR  (EADDR)      PSBU PSBL      R0 R1 R2 R3 R4 R5 R6
0017       ADDA,0    0054,3        0055   0000          01   08      00 01 00 00 00 00 00
TRACE COMMAND
IAR        INST                    EADDR  (EADDR)      PSBU PSBL      R0 R1 R2 R3 R4 R5 R6
001A       CCMI,0    0A            0013   000A          01   08      00 01 00 00 00 00 00
COMMAND DUMP
0050       17 07 13 18 65 00 01 82 02 03 06 04 03 02 51 00
0060       02 00 01 02 03 04 06 07 03 00 03 01 40 40 40 40
0070       40 40 40 40 40 40 00 00 00 40 40 40 40 40 40 40


NO. OF MACHINE CYCLES EXECUTED -        252


NO. OF INSTRUCTIONS EXECUTED =          79
```

**Figure 21.   SECOND COMMAND SET, Cont.**

# APPENDIXES

# APPENDIX A

## MEMORY INTERFACE

Figure 22 shows a complete interface between the 2650 and a 256 x 8 R/W random access memory. Since the memory chips are MOS they can be driven directly by the address lines and the control lines. The gates shown are assumed to be standard 7400 series TTL so that some signal buffering is assumed to be necessary. If CMOS or 74LS gates are used, some of the buffering inverters may not be necessary. The same is true of the data bus. Depending on the number and nature of the I/O devices being interfaced, it may or may not be necessary to buffer the data bus.

Because the data in and data out signals for the memory chips are bussed together, care must be taken to avoid overlap of drivers on the data bus. In this example, the problem is solved by using the write pulse into the memory as the chip select input instead of using the $\overline{R}/W$ line as is conventionally done. The $\overline{R}/W$ output from the processor is a level and is valid when Operation Request is true. Write Pulse from the processor is gated with the OPREQ and M/$\overline{IO}$ signals to assure proper operation.

For a large memory the next address line (ADR8) could be gated into the chain that generates the chip select signals, with similar write pulse generation for the higher order memory.

The $\overline{OPACK}$ signal is assumed to be false for the duration of all memory operations. This eliminates some gating from that control input. No problems will be encountered with this approach as long as the memories are fast enough for the clock speed being used with the processor. At a cycle time of 2.4μs, data must be returned to the processor by 1μs or less time from the OPREQ leading edge.
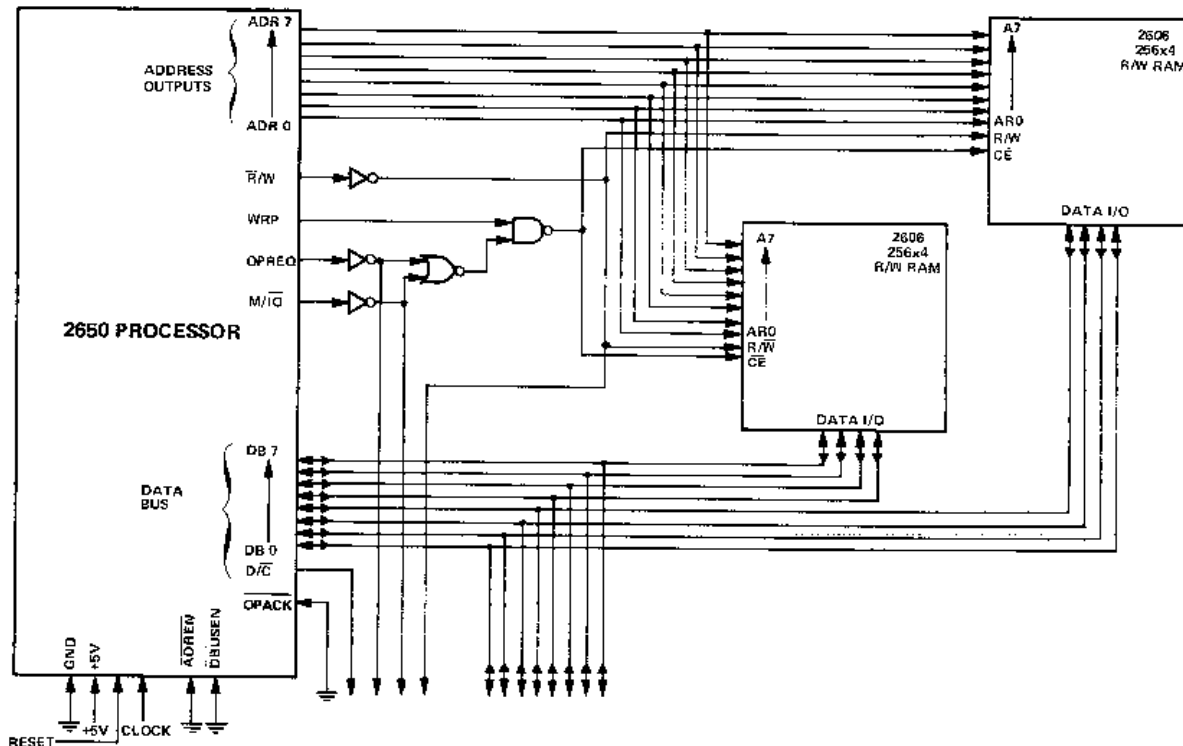


**Figure 22.**

# APPENDIX B

## I/O INTERFACE

Figure 23 shows one of many possible methods for buffering the data bus and interfacing it to several devices. There are advantages to be gained by using the Signetics 8T26. It has a PNP input buffer that keeps its low input level current at $200\mu$A instead of 1.6mA. This lightens the load on the processor bus drivers and allows the processor to interface to several 8T26's if necessary. The 8T26 has four complete driver/receiver pairs in a package, so two packages can fully buffer the 8-bit data bus.

The control signals generated for use with I/O interfaces are very straightforward. Combining M/$\overline{\text{IO}}$ with OPREQ generates a signal that can often be used conveniently at the I/O devices instead of having each device derive the signal individually. In the figure it is gated with the Read/Write information in order to control the bus buffer.

Each I/O device must handle four basic processor interface functions:

(a)  bus interface
(b)  data transfer logic
(c)  device selection logic
(d)  transfer acknowledge logic

Depending on the nature of the complete system and the particular I/O device, these functions can be either extremely simple or fairly complex.
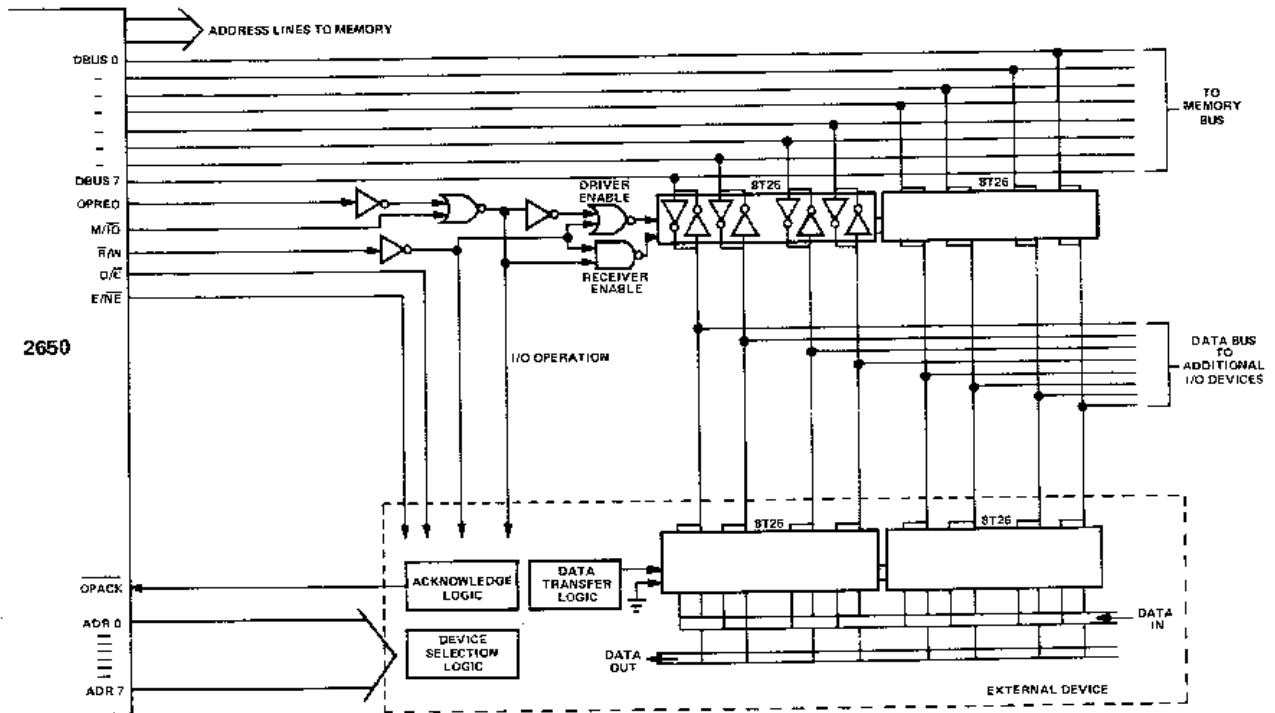


Figure 23.

# APPENDIX C

## INSTRUCTIONS, ADDITIONAL INFORMATION

The 2650 uses variable length instructions that are one, two or three bytes long. The instruction length is determined by the nature of the operation being performed and the addressing mode being used. Thus, the instruction can be expressed in one byte when no memory operand addressing is necessary, as with register-to-register or rotate instructions. On the other hand, for direct addressing instructions, three bytes are allocated. The relative and immediate addressing modes allow two-byte instructions to be implemented.

The 2650 uses explicit operand addressing; that is, each instruction specifies the operand address. The first byte of each 2650 instruction is divided into three fields and specifies the operation to be performed, the addressing mode to be used and, where appropriate, the register or condition code mask to be used.
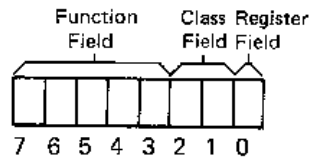


The CLASS field specifies the instruction group, the major address mode and the number of processor cycles required for each instruction. The CLASS field also specifies, with one exception, the number of bytes in the instruction. The following table shows the specifications for each class.

| CLASS FIELD | INSTRUCTION GROUP | ADDRESS REGISTER | BYTE LENGTH | DIRECT CYCLES |
|---|---|---|---|---|
| 0 | Arithmetic | Register | 1 | 2 |
| 1 | Arithmetic | Immediate | 2 | 2 |
| 2 | Arithmetic | Relative | 2 | 3 |
| 3 | Arithmetic | Absolute | 3 | 4 |
| 4 | Control (inc. rotate) | | 1 | 2 |
| 5 | Control | | 1-2 | 3 |
| 6 | Branch | Relative | 2 | 3 |
| 7 | Branch | Absolute | 3 | 3 |

Within the arithmetic groups (classes 0, 1, 2, and 3) the function field specifies one of the eight operations as follows:

| FUNCTION FIELD | ARITHMETIC OPERATION |
|---|---|
| 0 | LOAD |
| 1 | EXCLUSIVE OR |
| 2 | AND |
| 3 | INCLUSIVE OR |
| 4 | ADD |
| 5 | SUBTRACT |
| 6 | STORE |
| 7 | COMPARE |

Within the branch group (classes 6 and 7) the function field specifies one of eight operations as follows:

| FUNCTION FIELD | BRANCH OPERATION |
|---|---|
| 0 | Branch On Condition True |
| 1 | Branch To Subroutine On Condition True |
| 2 | Branch On Register Non-Zero |
| 3 | Branch To Subroutine On Register Non-Zero |
| 4 | Branch On Condition False |
| 5 | Branch To Subroutine On Condition False |
| 6 | Branch On Incrementing Register |
| 7 | Branch On Decrementing Register |

There is very little pattern to the use of the function field within the control group (classes 4 and 5).

The register field is used to specify the index register, to specify the operand source register, to specify the destination register, or a condition code mask. For the register-to-register and the indexed instructions, register zero is implicitly assumed to be the source or the destination of the instruction. For all other instructions that involve a register, the register field allows any of four registers to be specified, except for indexed branch instructions which require that register 3 be specified.

Conditional branch instructions utilize the 2-bit register field as a condition code mask field. A few instructions use the register field as part of the operation code and consequently allow no variation in register usage.

# APPENDIX D

## INSTRUCTION SUMMARY

### SIGNETICS 2650 PROCESSOR

## ALPHABETIC LISTING

| HEX | OP | Pg. | HEX | OP | Pg. | HEX | OP | Pg. |
|-----|------|-----|-----|------|-----|-----|------|-----|
| 8C | ADDA | 57 | 98 | BCFR | 75 | BC | BSFA | 81 |
| 8D | | | 99 | | | BD | | |
| 8E | | | 9A | | | BE | | |
| 8F | | | | | | | | |
| 84 | ADDI | 56 | 1C | BCTA | 74 | B8 | BSFR | 81 |
| 85 | | | 1D | | | B9 | | |
| 86 | | | 1E | | | BA | | |
| 87 | | | 1F | | | | | |
| 88 | ADDR | 56 | 18 | BCTR | 74 | 7C | BSNA | 82 |
| 89 | | | 19 | | | 7D | | |
| 8A | | | 1A | | | 7E | | |
| 8B | | | 1B | | | 7F | | |
| 80 | ADDZ | 55 | FC | BDRA | 77 | 78 | BSNR | 82 |
| 81 | | | FD | | | 79 | | |
| 82 | | | FE | | | 7A | | |
| 83 | | | FF | | | 7B | | |
| 4C | ANDA | 61 | F8 | BDRR | 77 | 3C | BSTA | 80 |
| 4D | | | F9 | | | 3D | | |
| 4E | | | FA | | | 3E | | |
| 4F | | | FB | | | 3F | | |
| 44 | ANDI | 60 | DC | BIRA | 76 | 38 | BSTR | 80 |
| 45 | | | DD | | | 39 | | |
| 46 | | | DE | | | 3A | | |
| 47 | | | DF | | | 3B | | |
| 48 | ANDR | 60 | D8 | BIRR | 76 | BF | BSXA | 83 |
| 49 | | | D9 | | | | | |
| 4A | | | DA | | | | | |
| 4B | | | DB | | | | | |
| 41 | ANDZ | 59 | 5C | BRNA | 78 | 9F | BXA | 79 |
| 42 | | | 5D | | | | | |
| 43 | | | 5E | | | | | |
| | | | 5F | | | | | |
| 9C | BCFA | 75 | 58 | BRNR | 78 | EC | CØMA | 67 |
| 9D | | | 59 | | | ED | | |
| 9E | | | 5A | | | EE | | |
| | | | 5B | | | EF | | |

| HEX | OP | Pg. | HEX | OP | Pg. | HEX | OP | Pg. |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| E4 | CØMI | 66 | 40 | HALT | 90 | 93 | LPSL | 69 |
| E5 |  |  |  |  |  |  |  |  |
| E6 |  |  |  |  |  | 92 | LPSU | 68 |
| E7 |  |  |  |  |  |  |  |  |
| E8 | CØMR | 66 | 6C | IØRA | 63 | C0 | NØP | 87 |
| E9 |  |  | 6D |  |  |  |  |  |
| EA |  |  | 6E |  |  |  |  |  |
| EB |  |  | 6F |  |  |  |  |  |
| E0 | CØMZ | 65 | 64 | IØRI | 62 | 77 | PPSL | 71 |
| E1 |  |  | 65 |  |  |  |  |  |
| E2 |  |  | 66 |  |  | 76 | PPSU | 70 |
| E3 |  |  | 67 |  |  |  |  |  |
| 75 | CPSL | 72 | 68 | IØRR | 62 | 30 | REDC | 85 |
|  |  |  | 69 |  |  | 31 |  |  |
| 74 | CPSU | 71 | 6A |  |  | 32 |  |  |
|  |  |  | 6B |  |  | 33 |  |  |
| 94 | DAR | 89 | 60 | IØRZ | 61 | 70 | REDD | 84 |
| 95 |  |  | 61 |  |  | 71 |  |  |
| 96 |  |  | 62 |  |  | 72 |  |  |
| 97 |  |  | 63 |  |  | 73 |  |  |
| 2C | EØRA | 65 | 0C | LØDA | 53 | 54 | REDE | 85 |
| 2D |  |  | 0D |  |  | 55 |  |  |
| 2E |  |  | 0E |  |  | 56 |  |  |
| 2F |  |  | 0F |  |  | 57 |  |  |
| 24 | EØRI | 64 | 04 | LØDI | 52 | 14 | RETC | 83 |
| 25 |  |  | 05 |  |  | 15 |  |  |
| 26 |  |  | 06 |  |  | 16 |  |  |
| 27 |  |  | 07 |  |  | 17 |  |  |
| 28 | EØRR | 64 | 08 | LØDR | 53 | 34 | RETE | 84 |
| 29 |  |  | 09 |  |  | 35 |  |  |
| 2A |  |  | 0A |  |  | 36 |  |  |
| 2B |  |  | 0B |  |  | 37 |  |  |
| 20 | EØRZ | 63 | 00 | LØDZ | 52 | D0 | RRL | 67 |
| 21 |  |  | 01 |  |  | D1 |  |  |
| 22 |  |  | 02 |  |  | D2 |  |  |
| 23 |  |  | 03 |  |  | D3 |  |  |

| HEX | OP | Pg. |
|-----|-----|-----|
| 50 | RRR | 68 |
| 51 | | |
| 52 | | |
| 53 | | |
| 13 | SPSL | 70 |
| 12 | SPSU | 69 |
| CC | STRA | 55 |
| CD | | |
| CE | | |
| CF | | |
| C8 | STRR | 54 |
| C9 | | |
| CA | | |
| CB | | |
| C1 | STRZ | 54 |
| C2 | | |
| C3 | | |
| AC | SUBA | 59 |
| AD | | |
| AE | | |
| AF | | |
| A4 | SUBI | 58 |
| A5 | | |
| A6 | | |
| A7 | | |
| A8 | SUBR | 58 |
| A9 | | |
| AA | | |
| AB | | |
| A0 | SUBZ | 57 |
| A1 | | |
| A2 | | |
| A3 | | |

| HEX | OP | Pg. |
|-----|-----|-----|
| F4 | TMI | 88 |
| F5 | | |
| F6 | | |
| F7 | | |
| B5 | TPSL | 73 |
| B4 | TPSU | 72 |
| B0 | WRTC | 86 |
| B1 | | |
| B2 | | |
| B3 | | |
| F0 | WRTD | 86 |
| F1 | | |
| F2 | | |
| F3 | | |
| D4 | WRTE | 87 |
| D5 | | |
| D6 | | |
| D7 | | |
| 9B | ZBRR | 73 |
| BB | ZBSR | 79 |

## NUMERIC LISTING

| HEX | OP | Pg. | HEX | OP | Pg. | HEX | OP | Pg. |
|-----|------|-----|-----|------|-----|-----|------|-----|
| 00 | LØDZ | 52 | 24 | EØRI | 64 | 44 | ANDI | 60 |
| 01 | | | 25 | | | 45 | | |
| 02 | | | 26 | | | 46 | | |
| 03 | | | 27 | | | 47 | | |
| 04 | LØDI | 52 | 28 | EØRR | 64 | 48 | ANDR | 60 |
| 05 | | | 29 | | | 49 | | |
| 06 | | | 2A | | | 4A | | |
| 07 | | | 2B | | | 4B | | |
| 08 | LØDR | 53 | 2C | EØRA | 65 | 4C | ANDA | 61 |
| 09 | | | 2D | | | 4D | | |
| 0A | | | 2E | | | 4E | | |
| 0B | | | 2F | | | 4F | | |
| 0C | LØDA | 53 | 30 | REDC | 85 | 50 | RRR | 68 |
| 0D | | | 31 | | | 51 | | |
| 0E | | | 32 | | | 52 | | |
| 0F | | | 33 | | | 53 | | |
| 12 | SPSU | 69 | 34 | RETE | 84 | 54 | REDE | 85 |
| | | | 35 | | | 55 | | |
| 13 | SPSL | 70 | 36 | | | 56 | | |
| | | | 37 | | | 57 | | |
| 14 | RETC | 83 | 38 | BSTR | 80 | 58 | BRNR | 78 |
| 15 | | | 39 | | | 59 | | |
| 16 | | | 3A | | | 5A | | |
| 17 | | | 3B | | | 5B | | |
| 18 | BCTR | 74 | 3C | BSTA | 80 | 5C | BRNA | 78 |
| 19 | | | 3D | | | 5D | | |
| 1A | | | 3E | | | 5E | | |
| 1B | | | 3F | | | 5F | | |
| 1C | BCTA | 74 | 40 | HALT | 90 | 60 | IØRZ | 61 |
| 1D | | | | | | 61 | | |
| 1E | | | | | | 62 | | |
| 1F | | | | | | 63 | | |
| 20 | EØRZ | 63 | 41 | ANDZ | 59 | 64 | IØRI | 62 |
| 21 | | | 42 | | | 65 | | |
| 22 | | | 43 | | | 66 | | |
| 23 | | | | | | 67 | | |

| HEX | OP | Pg. | HEX | OP | Pg. | HEX | OP | Pg. |
|-----|------|-----|-----|------|-----|-----|------|-----|
| 68 | IØRR | 62 | 88 | ADDR | 56 | A4 | SUBI | 58 |
| 69 | | | 89 | | | A5 | | |
| 6A | | | 8A | | | A6 | | |
| 6B | | | 8B | | | A7 | | |
| 6C | IØRA | 63 | 8C | ADDA | 57 | A8 | SUBR | 58 |
| 6D | | | 8D | | | A9 | | |
| 6E | | | 8E | | | AA | | |
| 6F | | | 8F | | | AB | | |
| 70 | REDD | 84 | 92 | LPSU | 68 | AC | SUBA | 59 |
| 71 | | | | | | AD | | |
| 72 | | | 93 | LPSL | 69 | AE | | |
| 73 | | | | | | AF | | |
| 74 | CPSU | 71 | 94 | DAR | 89 | B0 | WRTC | 86 |
| | | | 95 | | | B1 | | |
| 75 | CPSL | 72 | 96 | | | B2 | | |
| | | | 97 | | | B3 | | |
| 76 | PPSU | 70 | 98 | BCFR | 75 | B4 | TPSU | 72 |
| | | | 99 | | | | | |
| 77 | PPSL | 71 | 9A | | | B5 | TPSL | 73 |
| 78 | BSNR | 82 | 9B | ZBRR | 73 | B8 | BSFR | 81 |
| 79 | | | | | | B9 | | |
| 7A | | | | | | BA | | |
| 7B | | | | | | | | |
| 7C | BSNA | 82 | 9C | BCFA | 75 | BB | ZBSR | 79 |
| 7D | | | 9D | | | | | |
| 7E | | | 9E | | | | | |
| 7F | | | | | | | | |
| 80 | ADDZ | 55 | 9F | BXA | 79 | BC | BSFA | 81 |
| 81 | | | | | | BD | | |
| 82 | | | | | | BE | | |
| 83 | | | | | | | | |
| 84 | ADDI | 56 | A0 | SUBZ | 57 | BF | BSXA | 83 |
| 85 | | | A1 | | | | | |
| 86 | | | A2 | | | | | |
| 87 | | | A3 | | | | | |

| HEX | OP | Pg. | HEX | OP | Pg. |
|-----|------|-----|------|------|-----|
| C0 | NØP | 87 | E4 | CØMI | 66 |
| | | | E5 | | |
| | | | E6 | | |
| | | | E7 | | |
| C1 | STRZ | 54 | E8 | CØMR | 66 |
| C2 | | | E9 | | |
| C3 | | | EA | | |
| | | | EB | | |
| C8 | STRR | 54 | EC | CØMA | 67 |
| C9 | | | ED | | |
| CA | | | EE | | |
| CB | | | EF | | |
| CC | STRA | 55 | F0 | WRTD | 86 |
| CD | | | F1 | | |
| CE | | | F2 | | |
| CF | | | F3 | | |
| D0 | RRL | 67 | F4 | TMI | 88 |
| D1 | | | F5 | | |
| D2 | | | F6 | | |
| D3 | | | F7 | | |
| D4 | WRTE | 87 | F8 | BDRR | 77 |
| D5 | | | F9 | | |
| D6 | | | FA | | |
| D7 | | | FB | | |
| D8 | BIRR | 76 | FC | BDRA | 77 |
| D9 | | | FD | | |
| DA | | | FE | | |
| DB | | | FF | | |
| DC | BIRA | 76 | | | |
| DD | | | | | |
| DE | | | | | |
| DF | | | | | |
| E0 | CØMZ | 65 | | | |
| E1 | | | | | |
| E2 | | | | | |
| E3 | | | | | |

## ORGANIZED BY FUNCTION

| LOAD/STORE | | Pg. | ARITHMETIC | | Pg. | ARITHMETIC | | Pg. |
|---|---|---|---|---|---|---|---|---|
| 00 | LØDZ | 52 | 80 | ADDZ | 55 | 68 | IØRR | 62 |
| 01 | | | 81 | | | 69 | | |
| 02 | | | 82 | | | 6A | | |
| 03 | | | 83 | | | 6B | | |
| 04 | LØDI | 52 | 84 | ADDI | 56 | 6C | IØRA | 63 |
| 05 | | | 85 | | | 6D | | |
| 06 | | | 86 | | | 6E | | |
| 07 | | | 87 | | | 6F | | |
| 08 | LØDR | 53 | 88 | ADDR | 56 | 20 | EØRZ | 63 |
| 09 | | | 89 | | | 21 | | |
| 0A | | | 8A | | | 22 | | |
| 0B | | | 8B | | | 23 | | |
| 0C | LØDA | 53 | 8C | ADDA | 57 | 24 | EØRI | 64 |
| 0D | | | 8D | | | 25 | | |
| 0E | | | 8E | | | 26 | | |
| 0F | | | 8F | | | 27 | | |
| C1 | STRZ | 54 | A0 | SUBZ | 57 | 28 | EØRR | 64 |
| C2 | | | A1 | | | 29 | | |
| C3 | | | A2 | | | 2A | | |
| | | | A3 | | | 2B | | |
| C8 | STRR | 54 | A4 | SUBI | 58 | 2C | EØRA | 65 |
| C9 | | | A5 | | | 2D | | |
| CA | | | A6 | | | 2E | | |
| CB | | | A7 | | | 2F | | |
| CC | STRA | 55 | A8 | SUBR | 58 | 41 | ANDZ | 59 |
| CD | | | A9 | | | 42 | | |
| CE | | | AA | | | 43 | | |
| CF | | | AB | | | | | |
| | | | AC | SUBA | 59 | 44 | ANDI | 60 |
| | | | AD | | | 45 | | |
| | | | AE | | | 46 | | |
| | | | AF | | | 47 | | |
| | | | 60 | IØRZ | 61 | 48 | ANDR | 60 |
| | | | 61 | | | 49 | | |
| | | | 62 | | | 4A | | |
| | | | 63 | | | 4B | | |
| | | | 64 | IØRI | 62 | 4C | ANDA | 61 |
| | | | 65 | | | 4D | | |
| | | | 66 | | | 4E | | |
| | | | 67 | | | 4F | | |

| BRANCH | | Pg. | SUBROUTINE BRANCH | | Pg. | COMPARE | | Pg. |
|---|---|---|---|---|---|---|---|---|
| 18 | BCTR | 74 | 38 | BSTR | 80 | E0 | CØMZ | 65 |
| 19 | | | 39 | | | E1 | | |
| 1A | | | 3A | | | E2 | | |
| 1B | | | 3B | | | E3 | | |
| 1C | BCTA | 74 | 3C | BSTA | 80 | E4 | CØMI | 66 |
| 1D | | | 3D | | | E5 | | |
| 1E | | | 3E | | | E6 | | |
| 1F | | | 3F | | | E7 | | |
| 98 | BCFR | 75 | B8 | BSFR | 81 | E8 | CØMR | 66 |
| 99 | | | B9 | | | E9 | | |
| 9A | | | BA | | | EA | | |
| | | | | | | EB | | |
| 9C | BCFA | 75 | BC | BSFA | 81 | EC | CØMA | 67 |
| 9D | | | BD | | | ED | | |
| 9E | | | BE | | | EE | | |
| | | | | | | EF | | |
| 58 | BRNR | 78 | 78 | BSNR | 82 | INPUT/OUTPUT | | |
| 59 | | | 79 | | | 30 | REDC | 85 |
| 5A | | | 7A | | | 31 | | |
| 5B | | | 7B | | | 32 | | |
| 5C | BRNA | 78 | 7C | BSNA | 82 | 33 | | |
| 5D | | | 7D | | | 70 | REDD | 84 |
| 5E | | | 7E | | | 71 | | |
| 5F | | | 7F | | | 72 | | |
| D8 | BIRR | 76 | BF | BSXA | 83 | 73 | | |
| D9 | | | | | | B0 | WRTC | 86 |
| DA | | | | | | B1 | | |
| DB | | | | | | B2 | | |
| DC | BIRA | 76 | BB | ZBSR | 79 | B3 | | |
| DD | | | | | | F0 | WRTD | 86 |
| DE | | | | | | F1 | | |
| DF | | | | | | F2 | | |
| F8 | BDRR | 77 | SUBROUTINE RETURN | | | F3 | | |
| F9 | | | 14 | RETC | 83 | 54 | REDE | 85 |
| FA | | | 15 | | | 55 | | |
| FB | | | 16 | | | 56 | | |
| FC | BDRA | 77 | 17 | | | 57 | | |
| FD | | | 34 | RETE | 84 | D4 | WRTE | 87 |
| FE | | | 35 | | | D5 | | |
| FF | | | 36 | | | D6 | | |
| 9F | BXA | 79 | 37 | | | D7 | | |
| 9B | ZBRR | 73 | | | | | | |

# APPENDIX E

## SUMMARY OF 2650 INSTRUCTION MNEMONICS

In these tables parentheses are used to indicate options. In no case are they coded in any instruction. The following abbreviations are used:

r — register expression, must evaluate to $0 \leqslant r \leqslant 3$.
v — value expression
* — indirect indicator
a — address expression
x — index register expression
X — index register expression with optional auto-increment or auto-decrement

NOTE:
— the use of the indirect indicator is always optional.
— when an index register expression is specified, it can be followed by $',+'$ or $',-'$ which indicates use of auto-increment or auto-decrement of the index register. Example:

$$LODA,0 \qquad DPR,R3,+$$

BXA, BSXA are exceptions and do not permit auto-increment or auto-decrement.
— even though an address expression is specified in a hardware relative addressing instruction, the assembler develops it into a value of $(-64 \leqslant V \leqslant +63)$.
— a memory reference instruction which requires indexing may use only register 0 as the destination of the operation.
— if an index register expression is used with either the BXA or BSXA instructions it must specify index register #3 (either register bank) for indexing. Any other value in the index field will produce an error during assembly. However, it is not necessary to use an index register expression with these instructions; a blank in this field will default to register 3.

## LOAD/STORE INSTRUCTIONS

| | | | Length (bytes) |
|---|---|---|---|
| LODZ | r | Load Register Zero | 1 |
| LODI,r | v | Load Immediate | 2 |
| LODR,r | (*)a | Load Relative | 2 |
| LODA,r | (*)a(,X) | Load Absolute | 3 |
| STRZ | r | Store Register Zero | 1 |
| STRR,r | (*)a | Store Relative | 2 |
| STRA,r | (*)a(,X) | Store Absolute | 3 |

## ARITHMETIC INSTRUCTIONS

| | | | |
|---|---|---|---|
| ADDZ | r | Add to Register Zero | 1 |
| ADDI,r | v | Add Immediate | 2 |
| ADDR,r | (*)a | Add Relative | 2 |
| ADDA,r | (*)a(,X) | Add Absolute | 3 |
| SUBZ | r | Subtract from Register Zero | 1 |
| SUBI,r | v | Subtract Immediate | 2 |
| SUBR,r | (*)a | Subtract Relative | 2 |
| SUBA,r | (*)a(,X) | Subtract Absolute | 3 |

## LOGICAL INSTRUCTIONS

| | | | |
|---|---|---|---|
| ANDZ | r | And to Register Zero | 1 |
| ANDI,r | v | And Immediate | 2 |
| ANDR,r | (*)a | And Relative | 2 |
| ANDA,r | (*)a(,X) | And Absolute | 3 |
| IORZ | r | Inclusive or to Register Zero | 1 |
| IORI,r | v | Inclusive or Immediate | 2 |
| IORR,r | (*)a | Inclusive or Relative | 2 |
| IORA,r | (*)a(,X) | Inclusive or Absolute | 3 |
| EORZ | r | Exclusive or to Register Zero | 1 |
| EORI,r | v | Exclusive or Immediate | 2 |
| EORR,r | (*)a | Exclusive or Relative | 2 |
| EORA,r | (*)a(,X) | Exclusive or Absolute | 3 |

## COMPARISON INSTRUCTIONS

| | | | |
|---|---|---|---|
| COMZ | r | Compare to Register Zero | 1 |
| COMI,r | v | Compare Immediate | 2 |
| COMR,r | (*)a | Compare Relative | 2 |
| COMA,r | (*)a(,X) | Compare Absolute | 3 |

## ROTATE INSTRUCTIONS

| | | Length (bytes) |
|---|---|---|
| RRR,r | Rotate Register Right | 1 |
| RRL,r | Rotate Register Left | 1 |

## BRANCH INSTRUCTIONS

| | | | |
|---|---|---|---|
| BCTR,v | (*)a | Branch on Condition True Relative | 2 |
| BCFR,v | (*)a | Branch on Condition False Relative | 2 |
| BCTA,v | (*)a | Branch on Condition True Absolute | 3 |
| BCFA,v | (*)a | Branch on Condition False Absolute | 3 |
| BRNR,r | (*)a | Branch on Register Non-Zero Relative | 2 |
| BRNA,r | (*)a | Branch on Register Non-Zero Absolute | 3 |
| BIRR,r | (*)a | Branch on Incrementing Register Relative | 2 |

| | | | |
|---|---|---|---|
| BIRA,r | (*)a | Branch on Incrementing Register Absolute | 3 |
| BDRR,r | (*)a | Branch on Decrementing Register Relative | 2 |
| BDRA,r | (*)a | Branch on Decrementing Register Absolute | 3 |
| BXA | (*)a(,x) | Branch Indexed Absolute, Unconditional | 3 |
| ZBRR | (*)a | Zero Branch Relative, Unconditional | 2 |

## SUBROUTINE BRANCH/RETURN INSTRUCTIONS

| | | | |
|---|---|---|---|
| BSTR,v | (*)a | Branch to Subroutine on Condition True, Relative | 2 |
| BSFR,v | (*)a | Branch to Subroutine on Condition False, Relative | 2 |
| BSTA,v | (*)a | Branch to Subroutine on Condition True, Absolute | 3 |
| BSFA,v | (*)a | Branch to Subroutine on Condition False, Absolute | 3 |
| BSNR,r | (*)a | Branch to Subroutine on Non-Zero Register, Relative | 2 |
| BSNA,r | (*)a | Branch to Subroutine on Non-Zero Register, Absolute | 3 |
| BSXA | (*)a(,x) | Branch to Subroutine, Indexed, Unconditional | 3 |
| RETC,v | | Return From Subroutine, Conditional | 1 |
| RETE,v | | Return From Subroutine and Enable Interrupt, Conditional | 1 |
| ZBSR | (*),a | Zero Branch to Subroutine Relative, Unconditional | 2 |

## PROGRAM STATUS INSTRUCTIONS

| | | | |
|---|---|---|---|
| LPSU | | Load Program Status, Upper | 1 |
| LPSL | | Load Program Status, Lower | 1 |
| SPSU | | Store Program Status, Upper | 1 |
| SPSL | | Store Program Status, Lower | 1 |
| CPSU | v | Clear Program Status, Upper, Selective | 2 |
| CPSL | v | Clear Program Status, Lower, Selective | 2 |
| PPSU | v | Preset Program Status, Upper, Selective | 2 |
| PPSL | v | Preset Program Status, Lower, Selective | 2 |
| TPSU | v | Test Program Status, Upper, Selective | 2 |
| TPSL | v | Test Program Status Lower, Selective | 2 |

## INPUT/OUTPUT INSTRUCTIONS

| | | | |
|---|---|---|---|
| WRTD,r | | Write Data | 1 |
| REDD,r | | Read Data | 1 |
| WRTC,r | | Write Control | 1 |
| REDC,r | | Read Control | 1 |
| WRTE,r | v | Write Extended | 2 |
| REDE,r | v | Read Extended | 2 |

## MISCELLANEOUS INSTRUCTIONS

| | | | |
|---|---|---|---|
| HALT | | Halt, Enter Wait State | 1 |
| DAR,r | | Decimal Adjust Register | 1 |
| TMI,r | v | Test Under Mask Immediate | 2 |
| NO | | No Operation | 1 |

# APPENDIX F

## NOTES ABOUT THE 2650 PROCESSOR

1. AUTO-INCREMENT, DECREMENT of index register. This feature is optional on any instruction which uses indexing with the exception of BXA and BSXA. The increment or decrement occurs before the index register is added to the displacement in the instruction.

2. The contents of registers when used for indexing are considered to be unsigned absolute numbers. Consequently, index registers can contain values from 0 to 255. They "wrap-around" so that the number following 255 is 0.

3. Only absolute addressing instructions can be indexed.

4. The Branch on Incrementing Register or Decrementing Register instructions perform the increment or decrement before testing for zero. The only time the branch address is not taken, is when the register contains zero.

5. All hardware relative addressing is implemented as modulo 8K and therefore relative addressing across the top of a page boundary will result in a physical address near the bottom of the page being accessed. For example:

$1FFC_{16}$          LODR,R2          $+16

This instruction results, during execution, in accessing the byte at location 000C in the same page as the instruction. Similarly, negative relative addresses from near the bottom of a page may result in an effective address near the top of the page.

6. Page boundaries cannot be indexed across.

7. Data can always be accessed across a page boundary through use of relative indirect or absolute indirect addressing modes.

8. The only way to transfer control to a program in some other page is to branch absolute or branch indirectly to the new page. Program execution cannot flow across a page boundary.

9. Unconditional branch or branch to subroutine instructions are coded by specifying a value of 3 in the register/value field of BSTA, BSTR, BCTA or BCTR. Example:

```
UN          EQU          3
            • • •
            • • •
            • • •
            BSTA,UN          PAL
            BCTR,3          LOOP
```

Unconditional branches on conditions false (BCFA, BCFR) are not allowed.

# APPENDIX G

## ASC II AND EBCDIC CODES

This table presents the only characters that the assembler will recognize in an A or E type constant and their equivalent codes in hexadecimal.

| VALID CHARACTERS | EBCDIC CODE | ASC II CODE | VALID CHARACTERS | EBCDIC CODE | ASC II CODE |
|---|---|---|---|---|---|
| 0 | F0 | 30 | V | E5 | 56 |
| 1 | F1 | 31 | W | E6 | 57 |
| 2 | F2 | 32 | X | E7 | 58 |
| 3 | F3 | 33 | Y | E8 | 59 |
| 4 | F4 | 34 | Z | E9 | 5A |
| 5 | F5 | 35 | blank | 40 | 20 |
| 6 | F6 | 36 | . | 4B | 2E |
| 7 | F7 | 37 | ( | 4D | 28 |
| 8 | F8 | 38 | + | 4E | 2B |
| 9 | F9 | 39 | ¦ | 4F | 7C |
| A | C1 | 41 | & | 50 | 26 |
| B | C2 | 42 | ! | 5A | 21 |
| C | C3 | 43 | $ | 5B | 24 |
| D | C4 | 44 | * | 5C | 2A |
| E | C5 | 45 | ) | 5D | 29 |
| F | C6 | 46 | ; | 5E | 3B |
| G | C7 | 47 | ¬ or ~ | 5F | 7E* |
| H | C8 | 48 | – | 60 | 2D |
| I | C9 | 49 | / | 61 | 2F |
| J | D1 | 4A | , | 6B | 2C |
| K | D2 | 4B | % | 6C | 25 |
| L | D3 | 4C | — or ← | 6D | 5F* |
| M | D4 | 4D | > | 6E | 3E |
| N | D5 | 4E | ? | 6F | 3F |
| O | D6 | 4F | : | 7A | 3A |
| P | D7 | 50 | # | 7B | 23 |
| Q | D8 | 51 | @ | 7C | 40 |
| R | D9 | 52 | ' | 7D | 27 |
| S | E2 | 53 | = | 7E | 3D |
| T | E3 | 54 | " | 7F | 22 |
| U | E4 | 55 | < | 4C | 3C |

*may have different graphic symbols on different computer systems

# APPENDIX H

## COMPLETE ASCII CHARACTER SET

| | | | (MSB) b₇ | 0 | 0 | 1 | 1 | 1 | 1 |
|---|---|---|---|---|---|---|---|---|---|
| | | b₆ | | 1 | 1 | 0 | 0 | 1 | 1 |
| | | | b₅ | 0 | 1 | 0 | 1 | 0 | 1 |
| b₄ | b₃ | b₂ | b₁ | | | | | | |
| 0 | 0 | 0 | 0 | SP | 0 | @ | P | ` | p |
| 0 | 0 | 0 | 1 | ! | 1 | A | Q | a | q |
| 0 | 0 | 1 | 0 | " | 2 | B | R | b | r |
| 0 | 0 | 1 | 1 | # | 3 | C | S | c | s |
| 0 | 1 | 0 | 0 | $ | 4 | D | T | d | t |
| 0 | 1 | 0 | 1 | % | 5 | E | U | e | u |
| 0 | 1 | 1 | 0 | & | 6 | F | V | f | v |
| 0 | 1 | 1 | 1 | ' | 7 | G | W | g | w |
| 1 | 0 | 0 | 0 | ( | 8 | H | X | h | x |
| 1 | 0 | 0 | 1 | ) | 9 | I | Y | i | y |
| 1 | 0 | 1 | 0 | * | : | J | Z | j | z |
| 1 | 0 | 1 | 1 | + | ; | K | [ | k | { |
| 1 | 1 | 0 | 0 | , | < | L | \ | l | \| |
| 1 | 1 | 0 | 1 | - | = | M | ] | m | } |
| 1 | 1 | 1 | 0 | . | > | N | ↑ | n | ~ |
| 1 | 1 | 1 | 1 | / | ? | O | ← | o | DEL |

# APPENDIX I

## POWERS OF TWO TABLE

| $2^n$ | n | $2^{-n}$ |
|---|---|---|
| 1 | 0 | 1.0 |
| 2 | 1 | 0.5 |
| 4 | 2 | 0.25 |
| 8 | 3 | 0.125 |
| 16 | 4 | 0.062 5 |
| 32 | 5 | 0.031 25 |
| 64 | 6 | 0.015 625 |
| 128 | 7 | 0.007 812 5 |
| 256 | 8 | 0.003 906 25 |
| 512 | 9 | 0.001 953 125 |
| 1 024 | 10 | 0.000 976 562 5 |
| 2 048 | 11 | 0.000 488 281 25 |
| 4 096 | 12 | 0.000 244 140 625 |
| 8 192 | 13 | 0.000 122 070 312 5 |
| 16 384 | 14 | 0.000 061 035 156 25 |
| 32 768 | 15 | 0.000 030 517 578 125 |
| 65 536 | 16 | 0.000 015 258 789 062 5 |
| 131 072 | 17 | 0.000 007 629 394 531 25 |
| 262 144 | 18 | 0.000 003 814 697 265 625 |
| 524 288 | 19 | 0.000 001 907 348 632 812 5 |
| 1 048 576 | 20 | 0.000 000 953 674 316 406 25 |
| 2 097 152 | 21 | 0.000 000 476 837 158 203 125 |
| 4 194 304 | 22 | 0.000 000 238 418 579 101 562 5 |
| 8 388 608 | 23 | 0.000 000 119 209 289 550 781 25 |
| 16 777 216 | 24 | 0.000 000 059 604 644 775 390 625 |
| 33 554 432 | 25 | 0.000 000 029 802 322 387 695 312 5 |
| 67 108 864 | 26 | 0.000 000 014 901 161 193 847 656 25 |
| 134 217 728 | 27 | 0.000 000 007 450 580 596 923 828 125 |
| 268 435 456 | 28 | 0.000 000 003 725 290 298 461 914 062 5 |
| 536 870 912 | 29 | 0.000 000 001 862 645 149 230 957 031 45 |
| 1 073 741 824 | 30 | 0.000 000 000 931 322 574 615 478 515 625 |
| 2 147 483 648 | 31 | 0.000 000 000 465 661 287 307 739 257 812 5 |
| 4 294 967 296 | 32 | 0.000 000 000 232 830 643 653 869 628 906 25 |
| 8 589 934 592 | 33 | 0.000 000 000 116 415 321 826 934 814 453 125 |
| 17 179 869 184 | 34 | 0.000 000 000 058 207 660 913 467 407 226 562 5 |
| 34 359 738 368 | 35 | 0.000 000 000 029 103 830 456 733 703 613 281 25 |
| 68 719 476 736 | 36 | 0.000 000 000 014 551 915 228 366 851 806 640 625 |
| 137 438 953 472 | 37 | 0.000 000 000 007 275 957 614 183 425 903 320 312 5 |
| 274 877 906 944 | 38 | 0.000 000 000 003 637 978 807 091 712 951 660 156 25 |
| 549 755 813 888 | 39 | 0.000 000 000 001 818 989 403 545 856 475 830 078 125 |
| 1 099 511 627 776 | 40 | 0.000 000 000 000 909 494 701 772 928 237 915 039 062 5 |

# APPENDIX J

## HEXADECIMAL-DECIMAL CONVERSION TABLES

*From hex:* locate each hex digit in its corresponding column position and note the decimal equivalents. Add these to obtain the decimal value.

*From decimal:* (1) locate the largest decimal value in the table that will fit into the decimal number to be converted, and (2) note its hex equivalent and hex column position. (3) Find the decimal remainder. Repeat the process on this and subsequent remainders.

*Note:* Decimal, hexadecimal, (and binary) equivalents of all numbers from 0 to 255 are listed on panels 9 – 12.

| HEXADECIMAL COLUMNS | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 6 | | 5 | | 4 | | 3 | | 2 | | 1 | |
| HEX | = DEC | HEX | = DEC | HEX | = DEC | HEX | = DEC | HEX | = DEC | HEX | = DEC |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1,048,576 | 1 | 65,536 | 1 | 4,096 | 1 | 256 | 1 | 16 | 1 | 1 |
| 2 | 2,097,152 | 2 | 131,072 | 2 | 8,192 | 2 | 512 | 2 | 32 | 2 | 2 |
| 3 | 3,145,728 | 3 | 196,608 | 3 | 12,288 | 3 | 768 | 3 | 48 | 3 | 3 |
| 4 | 4,194,304 | 4 | 262,144 | 4 | 16,384 | 4 | 1,024 | 4 | 64 | 4 | 4 |
| 5 | 5,242,880 | 5 | 327,680 | 5 | 20,480 | 5 | 1,280 | 5 | 80 | 5 | 5 |
| 6 | 6,291,456 | 6 | 393,216 | 6 | 24,576 | 6 | 1,536 | 6 | 96 | 6 | 6 |
| 7 | 7,340,032 | 7 | 458,752 | 7 | 28,672 | 7 | 1,792 | 7 | 112 | 7 | 7 |
| 8 | 8,388,608 | 8 | 524,288 | 8 | 32,768 | 8 | 2,048 | 8 | 128 | 8 | 8 |
| 9 | 9,437,184 | 9 | 589,824 | 9 | 36,864 | 9 | 2,304 | 9 | 144 | 9 | 9 |
| A | 10,485,760 | A | 655,360 | A | 40,960 | A | 2,560 | A | 160 | A | 10 |
| B | 11,534,336 | B | 720,896 | B | 45,056 | B | 2,816 | B | 176 | B | 11 |
| C | 12,582,912 | C | 786,432 | C | 49,152 | C | 3,072 | C | 192 | C | 12 |
| D | 13,631,488 | D | 851,968 | D | 53,248 | D | 3,328 | D | 208 | D | 13 |
| E | 14,680,064 | E | 917,504 | E | 57,344 | E | 3,584 | E | 224 | E | 14 |
| F | 15,728,640 | F | 983,040 | F | 61,440 | F | 3,840 | F | 240 | F | 15 |
| 0 1 2 3 | | 4 5 6 7 | | 0 1 2 3 | | 4 5 6 7 | | 0 1 2 3 | | 4 5 6 7 | |
| BYTE | | | | BYTE | | | | BYTE | | | |

The table provides for direct conversion of hexadecimal and decimal numbers in these ranges:

| Hexadecimal | Decimal |
|---|---|
| 000 to FFF | 0000 to 4095 |

In the table, the decimal value appears at the intersection of the row representing the most significant hexadecimal digits ($16^2$ and $16^1$) and the column representing the least significant hexadecimal digit ($16^0$).

*Example:*   $C21_{16}$ = $3105_{10}$

| HEX | 0 | 1 | 2 |
|---|---|---|---|
| C0 | 3072 | 3073 | 3074 |
| C1 | 3088 | 3089 | 3090 |
| C2 | 3104 | 3105 | 3106 |
| C3 | 3120 | 3121 | 3122 |

|    | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|----|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 00 | 0000 | 0001 | 0002 | 0003 | 0004 | 0005 | 0006 | 0007 | 0008 | 0009 | 0010 | 0011 | 0012 | 0013 | 0014 | 0015 |
| 01 | 0016 | 0017 | 0018 | 0019 | 0020 | 0021 | 0022 | 0023 | 0024 | 0025 | 0026 | 0027 | 0028 | 0029 | 0030 | 0031 |
| 02 | 0032 | 0033 | 0034 | 0035 | 0036 | 0037 | 0038 | 0039 | 0040 | 0041 | 0042 | 0043 | 0044 | 0045 | 0046 | 0047 |
| 03 | 0048 | 0049 | 0050 | 0051 | 0052 | 0053 | 0054 | 0055 | 0056 | 0057 | 0058 | 0059 | 0060 | 0061 | 0062 | 0063 |
| 04 | 0064 | 0065 | 0066 | 0067 | 0068 | 0069 | 0070 | 0071 | 0072 | 0073 | 0074 | 0075 | 0076 | 0077 | 0078 | 0079 |
| 05 | 0080 | 0081 | 0082 | 0083 | 0084 | 0085 | 0086 | 0087 | 0088 | 0089 | 0090 | 0091 | 0092 | 0093 | 0094 | 0095 |
| 06 | 0096 | 0097 | 0098 | 0099 | 0100 | 0101 | 0102 | 0103 | 0104 | 0105 | 0106 | 0107 | 0108 | 0109 | 0110 | 0111 |
| 07 | 0112 | 0113 | 0114 | 0115 | 0116 | 0117 | 0118 | 0119 | 0120 | 0121 | 0122 | 0123 | 0124 | 0125 | 0126 | 0127 |
| 08 | 0128 | 0129 | 0130 | 0131 | 0132 | 0133 | 0134 | 0135 | 0136 | 0137 | 0138 | 0139 | 0140 | 0141 | 0142 | 0143 |
| 09 | 0144 | 0145 | 0146 | 0147 | 0148 | 0149 | 0150 | 0151 | 0152 | 0153 | 0154 | 0155 | 0156 | 0157 | 0158 | 0159 |
| 0A | 0160 | 0161 | 0162 | 0163 | 0164 | 0165 | 0166 | 0167 | 0168 | 0169 | 0170 | 0171 | 0172 | 0173 | 0174 | 0175 |
| 0B | 0176 | 0177 | 0178 | 0179 | 0180 | 0181 | 0182 | 0183 | 0184 | 0185 | 0186 | 0187 | 0188 | 0189 | 0190 | 0191 |
| 0C | 0192 | 0193 | 0194 | 0195 | 0196 | 0197 | 0198 | 0199 | 0200 | 0201 | 0202 | 0203 | 0204 | 0205 | 0206 | 0207 |
| 0D | 0208 | 0209 | 0210 | 0211 | 0212 | 0213 | 0214 | 0215 | 0216 | 0217 | 0218 | 0219 | 0220 | 0221 | 0222 | 0223 |
| 0E | 0224 | 0225 | 0226 | 0227 | 0228 | 0229 | 0230 | 0231 | 0232 | 0233 | 0234 | 0235 | 0236 | 0237 | 0238 | 0239 |
| 0F | 0240 | 0241 | 0242 | 0243 | 0244 | 0245 | 0246 | 0247 | 0248 | 0249 | 0250 | 0251 | 0252 | 0253 | 0254 | 0255 |

|    | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|----|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 10 | 0256 | 0257 | 0258 | 0259 | 0260 | 0261 | 0262 | 0263 | 0264 | 0265 | 0266 | 0267 | 0268 | 0269 | 0270 | 0271 |
| 11 | 0272 | 0273 | 0274 | 0275 | 0276 | 0277 | 0278 | 0279 | 0280 | 0281 | 0282 | 0283 | 0284 | 0285 | 0286 | 0287 |
| 12 | 0288 | 0289 | 0290 | 0291 | 0292 | 0293 | 0294 | 0295 | 0296 | 0297 | 0298 | 0299 | 0300 | 0301 | 0302 | 0303 |
| 13 | 0304 | 0305 | 0306 | 0307 | 0308 | 0309 | 0310 | 0311 | 0312 | 0313 | 0314 | 0315 | 0316 | 0317 | 0318 | 0319 |
| 14 | 0320 | 0321 | 0322 | 0323 | 0324 | 0325 | 0326 | 0327 | 0328 | 0329 | 0330 | 0331 | 0332 | 0333 | 0334 | 0335 |
| 15 | 0336 | 0337 | 0338 | 0339 | 0340 | 0341 | 0342 | 0343 | 0344 | 0345 | 0346 | 0347 | 0348 | 0349 | 0350 | 0351 |
| 16 | 0352 | 0353 | 0354 | 0355 | 0356 | 0357 | 0358 | 0359 | 0360 | 0361 | 0362 | 0363 | 0364 | 0365 | 0366 | 0367 |
| 17 | 0368 | 0369 | 0370 | 0371 | 0372 | 0373 | 0374 | 0375 | 0376 | 0377 | 0378 | 0379 | 0380 | 0381 | 0382 | 0383 |
| 18 | 0384 | 0385 | 0386 | 0387 | 0388 | 0389 | 0390 | 0391 | 0392 | 0393 | 0394 | 0395 | 0396 | 0397 | 0398 | 0399 |
| 19 | 0400 | 0401 | 0402 | 0403 | 0404 | 0405 | 0406 | 0407 | 0408 | 0409 | 0410 | 0411 | 0412 | 0413 | 0414 | 0415 |
| 1A | 0416 | 0417 | 0418 | 0419 | 0420 | 0421 | 0422 | 0423 | 0424 | 0425 | 0426 | 0427 | 0428 | 0429 | 0430 | 0431 |
| 1B | 0432 | 0433 | 0434 | 0435 | 0436 | 0437 | 0438 | 0439 | 0440 | 0441 | 0442 | 0443 | 0444 | 0445 | 0446 | 0447 |
| 1C | 0448 | 0449 | 0450 | 0451 | 0452 | 0453 | 0454 | 0455 | 0456 | 0457 | 0458 | 0459 | 0460 | 0461 | 0462 | 0463 |
| 1D | 0464 | 0465 | 0466 | 0467 | 0468 | 0469 | 0470 | 0471 | 0472 | 0473 | 0474 | 0475 | 0476 | 0477 | 0478 | 0479 |
| 1E | 0480 | 0481 | 0482 | 0483 | 0484 | 0485 | 0486 | 0487 | 0488 | 0489 | 0490 | 0491 | 0492 | 0493 | 0494 | 0495 |
| 1F | 0496 | 0497 | 0498 | 0499 | 0500 | 0501 | 0502 | 0503 | 0504 | 0505 | 0506 | 0507 | 0508 | 0509 | 0510 | 0511 |

|    | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|----|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 20 | 0512 | 0513 | 0514 | 0515 | 0516 | 0517 | 0518 | 0519 | 0520 | 0521 | 0522 | 0523 | 0524 | 0525 | 0526 | 0527 |
| 21 | 0528 | 0529 | 0530 | 0531 | 0532 | 0533 | 0534 | 0535 | 0536 | 0537 | 0538 | 0539 | 0540 | 0541 | 0542 | 0543 |
| 22 | 0544 | 0545 | 0546 | 0547 | 0548 | 0549 | 0550 | 0551 | 0552 | 0553 | 0554 | 0555 | 0556 | 0557 | 0558 | 0559 |
| 23 | 0560 | 0561 | 0562 | 0563 | 0564 | 0565 | 0566 | 0567 | 0568 | 0569 | 0570 | 0571 | 0572 | 0573 | 0574 | 0575 |
| 24 | 0576 | 0577 | 0578 | 0579 | 0580 | 0581 | 0582 | 0583 | 0584 | 0585 | 0586 | 0587 | 0588 | 0589 | 0590 | 0591 |
| 25 | 0592 | 0593 | 0594 | 0595 | 0596 | 0597 | 0598 | 0599 | 0600 | 0601 | 0602 | 0603 | 0604 | 0605 | 0606 | 0607 |
| 26 | 0608 | 0609 | 0610 | 0611 | 0612 | 0613 | 0614 | 0615 | 0616 | 0617 | 0618 | 0619 | 0620 | 0621 | 0622 | 0623 |
| 27 | 0624 | 0625 | 0626 | 0627 | 0628 | 0629 | 0630 | 0631 | 0632 | 0633 | 0634 | 0635 | 0636 | 0637 | 0638 | 0639 |
| 28 | 0640 | 0641 | 0642 | 0643 | 0644 | 0645 | 0646 | 0647 | 0648 | 0649 | 0650 | 0651 | 0652 | 0653 | 0654 | 0655 |
| 29 | 0656 | 0657 | 0658 | 0659 | 0660 | 0661 | 0662 | 0663 | 0664 | 0665 | 0666 | 0667 | 0668 | 0669 | 0670 | 0671 |
| 2A | 0672 | 0673 | 0674 | 0675 | 0676 | 0677 | 0678 | 0679 | 0680 | 0681 | 0682 | 0683 | 0684 | 0685 | 0686 | 0687 |
| 2B | 0688 | 0689 | 0690 | 0691 | 0692 | 0693 | 0694 | 0695 | 0696 | 0697 | 0698 | 0699 | 0700 | 0701 | 0702 | 0703 |
| 2C | 0704 | 0705 | 0706 | 0707 | 0708 | 0709 | 0710 | 0711 | 0712 | 0713 | 0714 | 0715 | 0716 | 0717 | 0718 | 0719 |
| 2D | 0720 | 0721 | 0722 | 0723 | 0724 | 0725 | 0726 | 0727 | 0728 | 0729 | 0730 | 0731 | 0732 | 0733 | 0734 | 0735 |
| 2E | 0736 | 0737 | 0738 | 0739 | 0740 | 0741 | 0742 | 0743 | 0744 | 0745 | 0746 | 0747 | 0748 | 0749 | 0750 | 0751 |
| 2F | 0752 | 0753 | 0754 | 0755 | 0756 | 0757 | 0758 | 0759 | 0760 | 0761 | 0762 | 0763 | 0764 | 0765 | 0766 | 0767 |

|    | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|----|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 30 | 0768 | 0769 | 0770 | 0771 | 0772 | 0773 | 0774 | 0775 | 0776 | 0777 | 0778 | 0779 | 0780 | 0781 | 0782 | 0783 |
| 31 | 0784 | 0785 | 0786 | 0787 | 0788 | 0789 | 0790 | 0791 | 0792 | 0793 | 0794 | 0795 | 0796 | 0797 | 0798 | 0799 |
| 32 | 0800 | 0801 | 0802 | 0803 | 0804 | 0805 | 0806 | 0807 | 0808 | 0809 | 0810 | 0811 | 0812 | 0813 | 0814 | 0815 |
| 33 | 0816 | 0817 | 0818 | 0819 | 0820 | 0821 | 0822 | 0823 | 0824 | 0825 | 0826 | 0827 | 0828 | 0829 | 0830 | 0831 |
| 34 | 0832 | 0833 | 0834 | 0835 | 0836 | 0837 | 0838 | 0839 | 0840 | 0841 | 0842 | 0843 | 0844 | 0845 | 0846 | 0847 |
| 35 | 0848 | 0849 | 0850 | 0851 | 0852 | 0853 | 0854 | 0855 | 0856 | 0857 | 0858 | 0859 | 0860 | 0861 | 0862 | 0863 |
| 36 | 0864 | 0865 | 0866 | 0867 | 0868 | 0869 | 0870 | 0871 | 0872 | 0873 | 0874 | 0875 | 0876 | 0877 | 0878 | 0879 |
| 37 | 0880 | 0881 | 0882 | 0883 | 0884 | 0885 | 0886 | 0887 | 0888 | 0889 | 0890 | 0891 | 0892 | 0893 | 0894 | 0895 |
| 38 | 0896 | 0897 | 0898 | 0899 | 0900 | 0901 | 0902 | 0903 | 0904 | 0905 | 0906 | 0907 | 0908 | 0909 | 0910 | 0911 |
| 39 | 0912 | 0913 | 0914 | 0915 | 0916 | 0917 | 0918 | 0919 | 0920 | 0921 | 0922 | 0923 | 0924 | 0925 | 0926 | 0927 |
| 3A | 0928 | 0929 | 0930 | 0931 | 0932 | 0933 | 0934 | 0935 | 0936 | 0937 | 0938 | 0939 | 0940 | 0941 | 0942 | 0943 |
| 3B | 0944 | 0945 | 0946 | 0947 | 0948 | 0949 | 0950 | 0951 | 0952 | 0953 | 0954 | 0955 | 0956 | 0957 | 0958 | 0959 |
| 3C | 0960 | 0961 | 0962 | 0963 | 0964 | 0965 | 0966 | 0967 | 0968 | 0969 | 0970 | 0971 | 0972 | 0973 | 0974 | 0975 |
| 3D | 0976 | 0977 | 0978 | 0979 | 0980 | 0981 | 0982 | 0983 | 0984 | 0985 | 0986 | 0987 | 0988 | 0989 | 0990 | 0991 |
| 3E | 0992 | 0993 | 0994 | 0995 | 0996 | 0997 | 0998 | 0999 | 1000 | 1001 | 1002 | 1003 | 1004 | 1005 | 1006 | 1007 |
| 3F | 1008 | 1009 | 1010 | 1011 | 1012 | 1013 | 1014 | 1015 | 1016 | 1017 | 1018 | 1019 | 1020 | 1021 | 1022 | 1023 |

|     | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|-----|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|
| 40 | 1024 | 1025 | 1026 | 1027 | 1028 | 1029 | 1030 | 1031 | 1032 | 1033 | 1034 | 1035 | 1036 | 1037 | 1038 | 1039 |
| 41 | 1040 | 1041 | 1042 | 1043 | 1044 | 1045 | 1046 | 1047 | 1048 | 1049 | 1050 | 1051 | 1052 | 1053 | 1054 | 1055 |
| 42 | 1056 | 1057 | 1058 | 1059 | 1060 | 1061 | 1062 | 1063 | 1064 | 1065 | 1066 | 1067 | 1068 | 1069 | 1070 | 1071 |
| 43 | 1072 | 1073 | 1074 | 1075 | 1076 | 1077 | 1078 | 1079 | 1080 | 1081 | 1082 | 1083 | 1084 | 1085 | 1086 | 1087 |
| 44 | 1088 | 1089 | 1090 | 1091 | 1092 | 1093 | 1094 | 1095 | 1096 | 1097 | 1098 | 1099 | 1100 | 1101 | 1102 | 1103 |
| 45 | 1104 | 1105 | 1106 | 1107 | 1108 | 1109 | 1110 | 1111 | 1112 | 1113 | 1114 | 1115 | 1116 | 1117 | 1118 | 1119 |
| 46 | 1120 | 1121 | 1122 | 1123 | 1124 | 1125 | 1126 | 1127 | 1128 | 1129 | 1130 | 1131 | 1132 | 1133 | 1134 | 1135 |
| 47 | 1136 | 1137 | 1138 | 1139 | 1140 | 1141 | 1142 | 1143 | 1144 | 1145 | 1146 | 1147 | 1148 | 1149 | 1150 | 1151 |
| 48 | 1152 | 1153 | 1154 | 1155 | 1156 | 1157 | 1158 | 1159 | 1160 | 1161 | 1162 | 1163 | 1164 | 1165 | 1166 | 1167 |
| 49 | 1168 | 1169 | 1170 | 1171 | 1172 | 1173 | 1174 | 1175 | 1176 | 1177 | 1178 | 1179 | 1180 | 1181 | 1182 | 1183 |
| 4A | 1184 | 1185 | 1186 | 1187 | 1188 | 1189 | 1190 | 1191 | 1192 | 1193 | 1194 | 1195 | 1196 | 1197 | 1198 | 1199 |
| 4B | 1200 | 1201 | 1202 | 1203 | 1204 | 1205 | 1206 | 1207 | 1208 | 1209 | 1210 | 1211 | 1212 | 1213 | 1214 | 1215 |
| 4C | 1216 | 1217 | 1218 | 1219 | 1220 | 1221 | 1222 | 1223 | 1224 | 1225 | 1226 | 1227 | 1228 | 1229 | 1230 | 1231 |
| 4D | 1232 | 1233 | 1234 | 1235 | 1236 | 1237 | 1238 | 1239 | 1240 | 1241 | 1242 | 1243 | 1244 | 1245 | 1246 | 1247 |
| 4E | 1248 | 1249 | 1250 | 1251 | 1252 | 1253 | 1254 | 1255 | 1256 | 1257 | 1258 | 1259 | 1260 | 1261 | 1262 | 1263 |
| 4F | 1264 | 1265 | 1266 | 1267 | 1268 | 1269 | 1270 | 1271 | 1272 | 1273 | 1274 | 1275 | 1276 | 1277 | 1278 | 1279 |

|     | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|-----|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|
| 50 | 1280 | 1281 | 1282 | 1283 | 1284 | 1285 | 1286 | 1287 | 1288 | 1289 | 1290 | 1291 | 1292 | 1293 | 1294 | 1295 |
| 51 | 1296 | 1297 | 1298 | 1299 | 1300 | 1301 | 1302 | 1303 | 1304 | 1305 | 1306 | 1307 | 1308 | 1309 | 1310 | 1311 |
| 52 | 1312 | 1313 | 1314 | 1315 | 1316 | 1317 | 1318 | 1319 | 1320 | 1321 | 1322 | 1323 | 1324 | 1325 | 1326 | 1327 |
| 53 | 1328 | 1329 | 1330 | 1331 | 1332 | 1333 | 1334 | 1335 | 1336 | 1337 | 1338 | 1339 | 1340 | 1341 | 1342 | 1343 |
| 54 | 1344 | 1345 | 1346 | 1347 | 1348 | 1349 | 1350 | 1351 | 1352 | 1353 | 1354 | 1355 | 1356 | 1357 | 1358 | 1359 |
| 55 | 1360 | 1361 | 1362 | 1363 | 1364 | 1365 | 1366 | 1367 | 1368 | 1369 | 1370 | 1371 | 1372 | 1373 | 1374 | 1375 |
| 56 | 1376 | 1377 | 1378 | 1379 | 1380 | 1381 | 1382 | 1383 | 1384 | 1385 | 1386 | 1387 | 1388 | 1389 | 1390 | 1391 |
| 57 | 1392 | 1393 | 1394 | 1395 | 1396 | 1397 | 1398 | 1399 | 1400 | 1401 | 1402 | 1403 | 1404 | 1405 | 1406 | 1407 |
| 58 | 1408 | 1409 | 1410 | 1411 | 1412 | 1413 | 1414 | 1415 | 1416 | 1417 | 1418 | 1419 | 1420 | 1421 | 1422 | 1423 |
| 59 | 1424 | 1425 | 1426 | 1427 | 1428 | 1429 | 1430 | 1431 | 1432 | 1433 | 1434 | 1435 | 1436 | 1437 | 1438 | 1439 |
| 5A | 1440 | 1441 | 1442 | 1443 | 1444 | 1445 | 1446 | 1447 | 1448 | 1449 | 1450 | 1451 | 1452 | 1453 | 1454 | 1455 |
| 5B | 1456 | 1457 | 1458 | 1459 | 1460 | 1461 | 1462 | 1463 | 1464 | 1465 | 1466 | 1467 | 1468 | 1469 | 1470 | 1471 |
| 5C | 1472 | 1473 | 1474 | 1475 | 1476 | 1477 | 1478 | 1479 | 1480 | 1481 | 1482 | 1483 | 1484 | 1485 | 1486 | 1487 |
| 5D | 1488 | 1489 | 1490 | 1491 | 1492 | 1493 | 1494 | 1495 | 1496 | 1497 | 1498 | 1499 | 1500 | 1501 | 1502 | 1503 |
| 5E | 1504 | 1505 | 1506 | 1507 | 1508 | 1509 | 1510 | 1511 | 1512 | 1513 | 1514 | 1515 | 1516 | 1517 | 1518 | 1519 |
| 5F | 1520 | 1521 | 1522 | 1523 | 1524 | 1525 | 1526 | 1527 | 1528 | 1529 | 1530 | 1531 | 1532 | 1533 | 1534 | 1535 |

|     | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|-----|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|
| 60 | 1536 | 1537 | 1538 | 1539 | 1540 | 1541 | 1542 | 1543 | 1544 | 1545 | 1546 | 1547 | 1548 | 1549 | 1550 | 1551 |
| 61 | 1552 | 1553 | 1554 | 1555 | 1556 | 1557 | 1558 | 1559 | 1560 | 1561 | 1562 | 1563 | 1564 | 1565 | 1566 | 1567 |
| 62 | 1568 | 1569 | 1570 | 1571 | 1572 | 1573 | 1574 | 1575 | 1576 | 1577 | 1578 | 1579 | 1580 | 1581 | 1582 | 1583 |
| 63 | 1584 | 1585 | 1586 | 1587 | 1588 | 1589 | 1590 | 1591 | 1592 | 1593 | 1594 | 1595 | 1596 | 1597 | 1598 | 1599 |
| 64 | 1600 | 1601 | 1602 | 1603 | 1604 | 1605 | 1606 | 1607 | 1608 | 1609 | 1610 | 1611 | 1612 | 1613 | 1614 | 1615 |
| 65 | 1616 | 1617 | 1618 | 1619 | 1620 | 1621 | 1622 | 1623 | 1624 | 1625 | 1626 | 1627 | 1628 | 1629 | 1630 | 1631 |
| 66 | 1632 | 1633 | 1634 | 1635 | 1636 | 1637 | 1638 | 1639 | 1640 | 1641 | 1642 | 1643 | 1644 | 1645 | 1646 | 1647 |
| 67 | 1648 | 1649 | 1650 | 1651 | 1652 | 1653 | 1654 | 1655 | 1656 | 1657 | 1658 | 1659 | 1660 | 1661 | 1662 | 1663 |
| 68 | 1664 | 1665 | 1666 | 1667 | 1668 | 1669 | 1670 | 1671 | 1672 | 1673 | 1674 | 1675 | 1676 | 1677 | 1678 | 1679 |
| 69 | 1680 | 1681 | 1682 | 1683 | 1684 | 1685 | 1686 | 1687 | 1688 | 1689 | 1690 | 1691 | 1692 | 1693 | 1694 | 1695 |
| 6A | 1696 | 1697 | 1698 | 1699 | 1700 | 1701 | 1702 | 1703 | 1704 | 1705 | 1706 | 1707 | 1708 | 1709 | 1710 | 1711 |
| 6B | 1712 | 1713 | 1714 | 1715 | 1716 | 1717 | 1718 | 1719 | 1720 | 1721 | 1722 | 1723 | 1724 | 1725 | 1726 | 1727 |
| 6C | 1728 | 1729 | 1730 | 1731 | 1732 | 1733 | 1734 | 1735 | 1736 | 1737 | 1738 | 1739 | 1740 | 1741 | 1742 | 1743 |
| 6D | 1744 | 1745 | 1746 | 1747 | 1748 | 1749 | 1750 | 1751 | 1752 | 1753 | 1754 | 1755 | 1756 | 1757 | 1758 | 1759 |
| 6E | 1760 | 1761 | 1762 | 1763 | 1764 | 1765 | 1766 | 1767 | 1768 | 1769 | 1770 | 1771 | 1772 | 1773 | 1774 | 1775 |
| 6F | 1776 | 1777 | 1778 | 1779 | 1780 | 1781 | 1782 | 1783 | 1784 | 1785 | 1786 | 1787 | 1788 | 1789 | 1790 | 1791 |

|     | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|-----|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|
| 70 | 1792 | 1793 | 1794 | 1795 | 1796 | 1797 | 1798 | 1799 | 1800 | 1801 | 1802 | 1803 | 1804 | 1805 | 1806 | 1807 |
| 71 | 1808 | 1809 | 1810 | 1811 | 1812 | 1813 | 1814 | 1815 | 1816 | 1817 | 1818 | 1819 | 1820 | 1821 | 1822 | 1823 |
| 72 | 1824 | 1825 | 1826 | 1827 | 1828 | 1829 | 1830 | 1831 | 1832 | 1833 | 1834 | 1835 | 1836 | 1837 | 1838 | 1839 |
| 73 | 1840 | 1841 | 1842 | 1843 | 1844 | 1845 | 1846 | 1847 | 1848 | 1849 | 1850 | 1851 | 1852 | 1853 | 1854 | 1855 |
| 74 | 1856 | 1857 | 1858 | 1859 | 1860 | 1861 | 1862 | 1863 | 1864 | 1865 | 1866 | 1867 | 1868 | 1869 | 1870 | 1871 |
| 75 | 1872 | 1873 | 1874 | 1875 | 1876 | 1877 | 1878 | 1879 | 1880 | 1881 | 1882 | 1883 | 1884 | 1885 | 1886 | 1887 |
| 76 | 1888 | 1889 | 1890 | 1891 | 1892 | 1893 | 1894 | 1895 | 1896 | 1897 | 1898 | 1899 | 1900 | 1901 | 1902 | 1903 |
| 77 | 1904 | 1905 | 1906 | 1907 | 1908 | 1909 | 1910 | 1911 | 1912 | 1913 | 1914 | 1915 | 1916 | 1917 | 1918 | 1919 |
| 78 | 1920 | 1921 | 1922 | 1923 | 1924 | 1925 | 1926 | 1927 | 1928 | 1929 | 1930 | 1931 | 1932 | 1933 | 1934 | 1935 |
| 79 | 1936 | 1937 | 1938 | 1939 | 1940 | 1941 | 1942 | 1943 | 1944 | 1945 | 1946 | 1947 | 1948 | 1949 | 1950 | 1951 |
| 7A | 1952 | 1953 | 1954 | 1955 | 1956 | 1957 | 1958 | 1959 | 1960 | 1961 | 1962 | 1963 | 1964 | 1965 | 1966 | 1967 |
| 7B | 1968 | 1969 | 1970 | 1971 | 1972 | 1973 | 1974 | 1975 | 1976 | 1977 | 1978 | 1979 | 1980 | 1981 | 1982 | 1983 |
| 7C | 1984 | 1985 | 1986 | 1987 | 1988 | 1989 | 1990 | 1991 | 1992 | 1993 | 1994 | 1995 | 1996 | 1997 | 1998 | 1999 |
| 7D | 2000 | 2001 | 2002 | 2003 | 2004 | 2005 | 2006 | 2007 | 2008 | 2009 | 2010 | 2011 | 2012 | 2013 | 2014 | 2015 |
| 7E | 2016 | 2017 | 2018 | 2019 | 2020 | 2021 | 2022 | 2023 | 2024 | 2025 | 2026 | 2027 | 2028 | 2029 | 2030 | 2031 |
| 7F | 2032 | 2033 | 2034 | 2035 | 2036 | 2037 | 2038 | 2039 | 2040 | 2041 | 2042 | 2043 | 2044 | 2045 | 2046 | 2047 |

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 80 | 2048 | 2049 | 2050 | 2051 | 2052 | 2053 | 2054 | 2055 | 2056 | 2057 | 2058 | 2059 | 2060 | 2061 | 2062 | 2063 |
| 81 | 2064 | 2065 | 2066 | 2067 | 2068 | 2069 | 2070 | 2071 | 2072 | 2073 | 2074 | 2075 | 2076 | 2077 | 2078 | 2079 |
| 82 | 2080 | 2081 | 2082 | 2083 | 2084 | 2085 | 2086 | 2087 | 2088 | 2089 | 2090 | 2091 | 2092 | 2093 | 2094 | 2095 |
| 83 | 2096 | 2097 | 2098 | 2099 | 2100 | 2101 | 2102 | 2103 | 2104 | 2105 | 2106 | 2107 | 2108 | 2109 | 2110 | 2111 |
| 84 | 2112 | 2113 | 2114 | 2115 | 2116 | 2117 | 2118 | 2119 | 2120 | 2121 | 2122 | 2123 | 2124 | 2125 | 2126 | 2127 |
| 85 | 2128 | 2129 | 2130 | 2131 | 2132 | 2133 | 2134 | 2135 | 2136 | 2137 | 2138 | 2139 | 2140 | 2141 | 2142 | 2143 |
| 86 | 2144 | 2145 | 2146 | 2147 | 2148 | 2149 | 2150 | 2151 | 2152 | 2153 | 2154 | 2155 | 2156 | 2157 | 2158 | 2159 |
| 87 | 2160 | 2161 | 2162 | 2163 | 2164 | 2165 | 2166 | 2167 | 2168 | 2169 | 2170 | 2171 | 2172 | 2173 | 2174 | 2175 |
| 88 | 2176 | 2177 | 2178 | 2179 | 2180 | 2181 | 2182 | 2183 | 2184 | 2185 | 2186 | 2187 | 2188 | 2189 | 2190 | 2191 |
| 89 | 2192 | 2193 | 2194 | 2195 | 2196 | 2197 | 2198 | 2199 | 2200 | 2201 | 2202 | 2203 | 2204 | 2205 | 2206 | 2207 |
| 8A | 2208 | 2209 | 2210 | 2211 | 2212 | 2213 | 2214 | 2215 | 2216 | 2217 | 2218 | 2219 | 2220 | 2221 | 2222 | 2223 |
| 8B | 2224 | 2225 | 2226 | 2227 | 2228 | 2229 | 2230 | 2231 | 2232 | 2233 | 2234 | 2235 | 2236 | 2237 | 2238 | 2239 |
| 8C | 2240 | 2241 | 2242 | 2243 | 2244 | 2245 | 2246 | 2247 | 2248 | 2249 | 2250 | 2251 | 2252 | 2253 | 2254 | 2255 |
| 8D | 2256 | 2257 | 2258 | 2259 | 2260 | 2261 | 2262 | 2263 | 2264 | 2265 | 2266 | 2267 | 2268 | 2269 | 2270 | 2271 |
| 8E | 2272 | 2273 | 2274 | 2275 | 2276 | 2277 | 2278 | 2279 | 2280 | 2281 | 2282 | 2283 | 2284 | 2285 | 2286 | 2287 |
| 8F | 2288 | 2289 | 2290 | 2291 | 2292 | 2293 | 2294 | 2295 | 2296 | 2297 | 2298 | 2299 | 2300 | 2301 | 2302 | 2303 |

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 90 | 2304 | 2305 | 2306 | 2307 | 2308 | 2309 | 2310 | 2311 | 2312 | 2313 | 2314 | 2315 | 2316 | 2317 | 2318 | 2319 |
| 91 | 2320 | 2321 | 2322 | 2323 | 2324 | 2325 | 2326 | 2327 | 2328 | 2329 | 2330 | 2331 | 2332 | 2333 | 2334 | 2335 |
| 92 | 2336 | 2337 | 2338 | 2339 | 2340 | 2341 | 2342 | 2343 | 2344 | 2345 | 2346 | 2347 | 2348 | 2349 | 2350 | 2351 |
| 93 | 2352 | 2353 | 2354 | 2355 | 2356 | 2357 | 2358 | 2359 | 2360 | 2361 | 2362 | 2363 | 2364 | 2365 | 2366 | 2367 |
| 94 | 2368 | 2369 | 2370 | 2371 | 2372 | 2373 | 2374 | 2375 | 2376 | 2377 | 2378 | 2379 | 2380 | 2381 | 2382 | 2383 |
| 95 | 2384 | 2385 | 2386 | 2387 | 2388 | 2389 | 2390 | 2391 | 2392 | 2393 | 2394 | 2395 | 2396 | 2397 | 2398 | 2399 |
| 96 | 2400 | 2401 | 2402 | 2403 | 2404 | 2405 | 2406 | 2407 | 2408 | 2409 | 2410 | 2411 | 2412 | 2413 | 2414 | 2415 |
| 97 | 2416 | 2417 | 2418 | 2419 | 2420 | 2421 | 2422 | 2423 | 2424 | 2425 | 2426 | 2427 | 2428 | 2429 | 2430 | 2431 |
| 98 | 2432 | 2433 | 2434 | 2435 | 2436 | 2437 | 2438 | 2439 | 2440 | 2441 | 2442 | 2443 | 2444 | 2445 | 2446 | 2447 |
| 99 | 2448 | 2449 | 2450 | 2451 | 2452 | 2453 | 2454 | 2455 | 2456 | 2457 | 2458 | 2459 | 2460 | 2461 | 2462 | 2463 |
| 9A | 2464 | 2465 | 2466 | 2467 | 2468 | 2469 | 2470 | 2471 | 2472 | 2473 | 2474 | 2475 | 2476 | 2477 | 2478 | 2479 |
| 9B | 2480 | 2481 | 2482 | 2483 | 2484 | 2485 | 2486 | 2487 | 2488 | 2489 | 2490 | 2491 | 2492 | 2493 | 2494 | 2495 |
| 9C | 2496 | 2497 | 2498 | 2499 | 2500 | 2501 | 2502 | 2503 | 2504 | 2505 | 2506 | 2507 | 2508 | 2509 | 2510 | 2511 |
| 9D | 2512 | 2513 | 2514 | 2515 | 2516 | 2517 | 2518 | 2519 | 2520 | 2521 | 2522 | 2523 | 2524 | 2525 | 2526 | 2527 |
| 9E | 2528 | 2529 | 2530 | 2531 | 2532 | 2533 | 2534 | 2535 | 2536 | 2537 | 2538 | 2539 | 2540 | 2541 | 2542 | 2543 |
| 9F | 2544 | 2545 | 2546 | 2547 | 2548 | 2549 | 2550 | 2551 | 2552 | 2553 | 2554 | 2555 | 2556 | 2557 | 2558 | 2559 |

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| A0 | 2560 | 2561 | 2562 | 2563 | 2564 | 2565 | 2566 | 2567 | 2568 | 2569 | 2570 | 2571 | 2572 | 2573 | 2574 | 2575 |
| A1 | 2576 | 2577 | 2578 | 2579 | 2580 | 2581 | 2582 | 2583 | 2584 | 2585 | 2586 | 2587 | 2588 | 2589 | 2590 | 2591 |
| A2 | 2592 | 2593 | 2594 | 2595 | 2596 | 2597 | 2598 | 2599 | 2600 | 2601 | 2602 | 2603 | 2604 | 2605 | 2606 | 2607 |
| A3 | 2608 | 2609 | 2610 | 2611 | 2612 | 2613 | 2614 | 2615 | 2616 | 2617 | 2618 | 2619 | 2620 | 2621 | 2622 | 2623 |
| A4 | 2624 | 2625 | 2626 | 2627 | 2628 | 2629 | 2630 | 2631 | 2632 | 2633 | 2634 | 2635 | 2636 | 2637 | 2638 | 2639 |
| A5 | 2640 | 2641 | 2642 | 2643 | 2644 | 2645 | 2646 | 2647 | 2648 | 2649 | 2650 | 2651 | 2652 | 2653 | 2654 | 2655 |
| A6 | 2656 | 2657 | 2658 | 2659 | 2660 | 2661 | 2662 | 2663 | 2664 | 2665 | 2666 | 2667 | 2668 | 2669 | 2670 | 2671 |
| A7 | 2672 | 2673 | 2674 | 2675 | 2676 | 2677 | 2678 | 2679 | 2680 | 2681 | 2682 | 2683 | 2684 | 2685 | 2686 | 2687 |
| A8 | 2688 | 2689 | 2690 | 2691 | 2692 | 2693 | 2694 | 2695 | 2696 | 2697 | 2698 | 2699 | 2700 | 2701 | 2702 | 2703 |
| A9 | 2704 | 2705 | 2706 | 2707 | 2708 | 2709 | 2710 | 2711 | 2712 | 2713 | 2714 | 2715 | 2716 | 2717 | 2718 | 2719 |
| AA | 2720 | 2721 | 2722 | 2723 | 2724 | 2725 | 2726 | 2727 | 2728 | 2729 | 2730 | 2731 | 2732 | 2733 | 2734 | 2735 |
| AB | 2736 | 2737 | 2738 | 2739 | 2740 | 2741 | 2742 | 2743 | 2744 | 2745 | 2746 | 2747 | 2748 | 2749 | 2750 | 2751 |
| AC0 | 2752 | 2753 | 2754 | 2755 | 2756 | 2757 | 2758 | 2759 | 2760 | 2761 | 2762 | 2763 | 2764 | 2765 | 2766 | 2767 |
| AD0 | 2768 | 2769 | 2770 | 2771 | 2772 | 2773 | 2774 | 2775 | 2776 | 2777 | 2778 | 2779 | 2780 | 2781 | 2782 | 2783 |
| AE0 | 2784 | 2785 | 2786 | 2787 | 2788 | 2789 | 2790 | 2791 | 2792 | 2793 | 2794 | 2795 | 2796 | 2797 | 2798 | 2799 |
| AF0 | 2800 | 2801 | 2802 | 2803 | 2804 | 2805 | 2806 | 2807 | 2808 | 2809 | 2810 | 2811 | 2812 | 2813 | 2814 | 2815 |

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| B0 | 2816 | 2817 | 2818 | 2819 | 2820 | 2821 | 2822 | 2823 | 2824 | 2825 | 2826 | 2827 | 2828 | 2829 | 2830 | 2831 |
| B1 | 2832 | 2833 | 2834 | 2835 | 2836 | 2837 | 2838 | 2839 | 2840 | 2841 | 2842 | 2843 | 2844 | 2845 | 2846 | 2847 |
| B2 | 2848 | 2849 | 2850 | 2851 | 2852 | 2853 | 2854 | 2855 | 2856 | 2857 | 2858 | 2859 | 2860 | 2861 | 2862 | 2863 |
| B3 | 2864 | 2865 | 2866 | 2867 | 2868 | 2869 | 2870 | 2871 | 2872 | 2873 | 2874 | 2875 | 2876 | 2877 | 2878 | 2879 |
| B4 | 2880 | 2881 | 2882 | 2883 | 2884 | 2885 | 2886 | 2887 | 2888 | 2889 | 2890 | 2891 | 2892 | 2893 | 2894 | 2895 |
| B5 | 2896 | 2897 | 2898 | 2899 | 2900 | 2901 | 2902 | 2903 | 2904 | 2905 | 2906 | 2907 | 2908 | 2909 | 2910 | 2911 |
| B6 | 2912 | 2913 | 2914 | 2915 | 2916 | 2917 | 2918 | 2919 | 2920 | 2921 | 2922 | 2923 | 2924 | 2925 | 2926 | 2927 |
| B7 | 2928 | 2929 | 2930 | 2931 | 2932 | 2933 | 2934 | 2935 | 2936 | 2937 | 2938 | 2939 | 2940 | 2941 | 2942 | 2943 |
| B8 | 2944 | 2945 | 2946 | 2947 | 2948 | 2949 | 2950 | 2951 | 2952 | 2953 | 2954 | 2955 | 2956 | 2957 | 2958 | 2959 |
| B9 | 2960 | 2961 | 2962 | 2963 | 2964 | 2965 | 2966 | 2967 | 2968 | 2969 | 2970 | 2971 | 2972 | 2973 | 2974 | 2975 |
| BA | 2976 | 2977 | 2978 | 2979 | 2980 | 2981 | 2982 | 2983 | 2984 | 2985 | 2986 | 2987 | 2988 | 2989 | 2990 | 2991 |
| BB | 2992 | 2993 | 2994 | 2995 | 2996 | 2997 | 2998 | 2999 | 3000 | 3001 | 3002 | 3003 | 3004 | 3005 | 3006 | 3007 |
| BC | 3008 | 3009 | 3010 | 3011 | 3012 | 3013 | 3014 | 3015 | 3016 | 3017 | 3018 | 3019 | 3020 | 3021 | 3022 | 3023 |
| BD | 3024 | 3025 | 3026 | 3027 | 3028 | 3029 | 3030 | 3031 | 3032 | 3033 | 3034 | 3035 | 3036 | 3037 | 3038 | 3039 |
| BE | 3040 | 3041 | 3042 | 3043 | 3044 | 3045 | 3046 | 3047 | 3048 | 3049 | 3050 | 3051 | 3052 | 3053 | 3054 | 3055 |
| BF | 3056 | 3057 | 3058 | 3059 | 3060 | 3061 | 3062 | 3063 | 3064 | 3065 | 3066 | 3067 | 3068 | 3069 | 3070 | 3071 |

|    | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|----|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| C0 | 3072 | 3073 | 3074 | 3075 | 3076 | 3077 | 3078 | 3079 | 3080 | 3081 | 3082 | 3083 | 3084 | 3085 | 3086 | 3087 |
| C1 | 3088 | 3089 | 3090 | 3091 | 3092 | 3093 | 3094 | 3095 | 3096 | 3097 | 3098 | 3099 | 3100 | 3101 | 3102 | 3103 |
| C2 | 3104 | 3105 | 3106 | 3107 | 3108 | 3109 | 3110 | 3111 | 3112 | 3113 | 3114 | 3115 | 3116 | 3117 | 3118 | 3119 |
| C3 | 3120 | 3121 | 3122 | 3123 | 3124 | 3125 | 3126 | 3127 | 3128 | 3129 | 3130 | 3131 | 3132 | 3133 | 3134 | 3135 |
| C4 | 3136 | 3137 | 3138 | 3139 | 3140 | 3141 | 3142 | 3143 | 3144 | 3145 | 3146 | 3147 | 3148 | 3149 | 3150 | 3151 |
| C5 | 3152 | 3153 | 3154 | 3155 | 3156 | 3157 | 3158 | 3159 | 3160 | 3161 | 3162 | 3163 | 3164 | 3165 | 3166 | 3167 |
| C6 | 3168 | 3169 | 3170 | 3171 | 3172 | 3173 | 3174 | 3175 | 3176 | 3177 | 3178 | 3179 | 3180 | 3181 | 3182 | 3183 |
| C7 | 3184 | 3185 | 3186 | 3187 | 3188 | 3189 | 3190 | 3191 | 3192 | 3193 | 3194 | 3195 | 3196 | 3197 | 3198 | 3199 |
| C8 | 3200 | 3201 | 3202 | 3203 | 3204 | 3205 | 3206 | 3207 | 3208 | 3209 | 3210 | 3211 | 3212 | 3213 | 3214 | 3215 |
| C9 | 3216 | 3217 | 3218 | 3219 | 3220 | 3221 | 3222 | 3223 | 3224 | 3225 | 3226 | 3227 | 3228 | 3229 | 3230 | 3231 |
| CA | 3232 | 3233 | 3234 | 3235 | 3236 | 3237 | 3238 | 3239 | 3240 | 3241 | 3242 | 3243 | 3244 | 3245 | 3246 | 3247 |
| CB | 3248 | 3249 | 3250 | 3251 | 3252 | 3253 | 3254 | 3255 | 3256 | 3257 | 3258 | 3259 | 3260 | 3261 | 3262 | 3263 |
| CC | 3264 | 3265 | 3266 | 3267 | 3268 | 3269 | 3270 | 3271 | 3272 | 3273 | 3274 | 3275 | 3276 | 3277 | 3278 | 3279 |
| CD | 3280 | 3281 | 3282 | 3283 | 3284 | 3285 | 3286 | 3287 | 3288 | 3289 | 3290 | 3291 | 3292 | 3293 | 3294 | 3295 |
| CE | 3296 | 3297 | 3298 | 3299 | 3300 | 3301 | 3302 | 3303 | 3304 | 3305 | 3306 | 3307 | 3308 | 3309 | 3310 | 3311 |
| CF | 3312 | 3313 | 3314 | 3315 | 3316 | 3317 | 3318 | 3319 | 3320 | 3321 | 3322 | 3323 | 3324 | 3325 | 3326 | 3327 |

|    | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|----|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| D0 | 3328 | 3329 | 3330 | 3331 | 3332 | 3333 | 3334 | 3335 | 3336 | 3337 | 3338 | 3339 | 3340 | 3341 | 3342 | 3343 |
| D1 | 3344 | 3345 | 3346 | 3347 | 3348 | 3349 | 3350 | 3351 | 3352 | 3353 | 3354 | 3355 | 3356 | 3357 | 3358 | 3359 |
| D2 | 3360 | 3361 | 3362 | 3363 | 3364 | 3365 | 3366 | 3367 | 3368 | 3369 | 3370 | 3371 | 3372 | 3373 | 3374 | 3375 |
| D3 | 3376 | 3377 | 3378 | 3379 | 3380 | 3381 | 3382 | 3383 | 3384 | 3385 | 3386 | 3387 | 3388 | 3389 | 3390 | 3391 |
| D4 | 3392 | 3393 | 3394 | 3395 | 3396 | 3397 | 3398 | 3399 | 3400 | 3401 | 3402 | 3403 | 3404 | 3405 | 3406 | 3407 |
| D5 | 3408 | 3409 | 3410 | 3411 | 3412 | 3413 | 3414 | 3415 | 3416 | 3417 | 3418 | 3419 | 3420 | 3421 | 3422 | 3423 |
| D6 | 3424 | 3425 | 3426 | 3427 | 3428 | 3429 | 3430 | 3431 | 3432 | 3433 | 3434 | 3435 | 3436 | 3437 | 3438 | 3439 |
| D7 | 3440 | 3441 | 3442 | 3443 | 3444 | 3445 | 3446 | 3447 | 3448 | 3449 | 3450 | 3451 | 3452 | 3453 | 3454 | 3455 |
| D8 | 3456 | 3457 | 3458 | 3459 | 3460 | 3461 | 3462 | 3463 | 3464 | 3465 | 3466 | 3467 | 3468 | 3469 | 3470 | 3471 |
| D9 | 3472 | 3473 | 3474 | 3475 | 3476 | 3477 | 3478 | 3479 | 3480 | 3481 | 3482 | 3483 | 3484 | 3485 | 3486 | 3487 |
| DA | 3488 | 3489 | 3490 | 3491 | 3492 | 3493 | 3494 | 3495 | 3496 | 3497 | 3498 | 3499 | 3500 | 3501 | 3502 | 3503 |
| DB | 3504 | 3505 | 3506 | 3507 | 3508 | 3509 | 3510 | 3511 | 3512 | 3513 | 3514 | 3515 | 3516 | 3517 | 3518 | 3519 |
| DC | 3520 | 3521 | 3522 | 3523 | 3524 | 3525 | 3526 | 3527 | 3528 | 3529 | 3530 | 3531 | 3532 | 3533 | 3534 | 3535 |
| DD | 3536 | 3537 | 3538 | 3539 | 3540 | 3541 | 3542 | 3543 | 3544 | 3545 | 3546 | 3547 | 3548 | 3549 | 3550 | 3551 |
| DE | 3552 | 3553 | 3554 | 3555 | 3556 | 3557 | 3558 | 3559 | 3560 | 3561 | 3562 | 3563 | 3564 | 3565 | 3566 | 3567 |
| DF | 3568 | 3569 | 3570 | 3571 | 3572 | 3573 | 3574 | 3575 | 3576 | 3577 | 3578 | 3579 | 3580 | 3581 | 3582 | 3583 |

|    | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|----|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| E0 | 3584 | 3585 | 3586 | 3587 | 3588 | 3589 | 3590 | 3591 | 3592 | 3593 | 3594 | 3595 | 3596 | 3597 | 3598 | 3599 |
| E1 | 3600 | 3601 | 3602 | 3603 | 3604 | 3605 | 3606 | 3607 | 3608 | 3609 | 3610 | 3611 | 3612 | 3613 | 3614 | 3615 |
| E2 | 3616 | 3617 | 3618 | 3619 | 3620 | 3621 | 3622 | 3623 | 3624 | 3625 | 3626 | 3627 | 3628 | 3629 | 3630 | 3631 |
| E3 | 3632 | 3633 | 3634 | 3635 | 3636 | 3637 | 3638 | 3639 | 3640 | 3641 | 3642 | 3643 | 3644 | 3645 | 3646 | 3647 |
| E4 | 3648 | 3649 | 3650 | 3651 | 3652 | 3653 | 3654 | 3655 | 3656 | 3657 | 3658 | 3659 | 3660 | 3661 | 3662 | 3663 |
| E5 | 3664 | 3665 | 3666 | 3667 | 3668 | 3669 | 3670 | 3671 | 3672 | 3673 | 3674 | 3675 | 3676 | 3677 | 3678 | 3679 |
| E6 | 3680 | 3681 | 3682 | 3683 | 3684 | 3685 | 3686 | 3687 | 3688 | 3689 | 3690 | 3691 | 3692 | 3693 | 3694 | 3695 |
| E7 | 3696 | 3697 | 3698 | 3699 | 3700 | 3701 | 3702 | 3703 | 3704 | 3705 | 3706 | 3707 | 3708 | 3709 | 3710 | 3711 |
| E8 | 3712 | 3713 | 3714 | 3715 | 3716 | 3717 | 3718 | 3719 | 3720 | 3721 | 3722 | 3723 | 3724 | 3725 | 3726 | 3727 |
| E9 | 3728 | 3729 | 3730 | 3731 | 3732 | 3733 | 3734 | 3735 | 3736 | 3737 | 3738 | 3739 | 3740 | 3741 | 3742 | 3743 |
| EA | 3744 | 3745 | 3746 | 3747 | 3748 | 3749 | 3750 | 3751 | 3752 | 3753 | 3754 | 3755 | 3756 | 3757 | 3758 | 3759 |
| EB | 3760 | 3761 | 3762 | 3763 | 3764 | 3765 | 3766 | 3767 | 3768 | 3769 | 3770 | 3771 | 3772 | 3773 | 3774 | 3775 |
| EC | 3776 | 3777 | 3778 | 3779 | 3780 | 3781 | 3782 | 3783 | 3784 | 3785 | 3786 | 3787 | 3788 | 3789 | 3790 | 3791 |
| ED | 3792 | 3793 | 3794 | 3795 | 3796 | 3797 | 3798 | 3799 | 3800 | 3801 | 3802 | 3803 | 3804 | 3805 | 3806 | 3807 |
| EE | 3808 | 3809 | 3810 | 3811 | 3812 | 3813 | 3814 | 3815 | 3816 | 3817 | 3818 | 3819 | 3820 | 3821 | 3822 | 3823 |
| EF | 3824 | 3825 | 3826 | 3827 | 3828 | 3829 | 3830 | 3831 | 3832 | 3833 | 3834 | 3835 | 3836 | 3837 | 3838 | 3839 |

|    | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|----|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| F0 | 3840 | 3841 | 3842 | 3843 | 3844 | 3845 | 3846 | 3847 | 3848 | 3849 | 3850 | 3851 | 3852 | 3853 | 3854 | 3855 |
| F1 | 3856 | 3857 | 3858 | 3859 | 3860 | 3861 | 3862 | 3863 | 3864 | 3865 | 3866 | 3867 | 3868 | 3869 | 3870 | 3871 |
| F2 | 3872 | 3873 | 3874 | 3875 | 3876 | 3877 | 3878 | 3879 | 3880 | 3881 | 3882 | 3883 | 3884 | 3885 | 3886 | 3887 |
| F3 | 3888 | 3889 | 3890 | 3891 | 3892 | 3893 | 3894 | 3895 | 3896 | 3897 | 3898 | 3899 | 3900 | 3901 | 3902 | 3903 |
| F4 | 3904 | 3905 | 3906 | 3907 | 3908 | 3909 | 3910 | 3911 | 3912 | 3913 | 3914 | 3915 | 3916 | 3917 | 3918 | 3919 |
| F5 | 3920 | 3921 | 3922 | 3923 | 3924 | 3925 | 3926 | 3927 | 3928 | 3929 | 3930 | 3931 | 3932 | 3933 | 3934 | 3935 |
| F6 | 3936 | 3937 | 3938 | 3939 | 3940 | 3941 | 3942 | 3943 | 3944 | 3945 | 3946 | 3947 | 3948 | 3949 | 3950 | 3951 |
| F7 | 3952 | 3953 | 3954 | 3955 | 3956 | 3957 | 3958 | 3959 | 3960 | 3961 | 3962 | 3963 | 3964 | 3965 | 3966 | 3967 |
| F8 | 3968 | 3969 | 3970 | 3971 | 3972 | 3973 | 3974 | 3975 | 3976 | 3977 | 3978 | 3979 | 3980 | 3981 | 3982 | 3983 |
| F9 | 3984 | 3985 | 3986 | 3987 | 3988 | 3989 | 3990 | 3991 | 3992 | 3993 | 3994 | 3995 | 3996 | 3997 | 3998 | 3999 |
| FA | 4000 | 4001 | 4002 | 4003 | 4004 | 4005 | 4006 | 4007 | 4008 | 4009 | 4010 | 4011 | 4012 | 4013 | 4014 | 4015 |
| FB | 4016 | 4017 | 4018 | 4019 | 4020 | 4021 | 4022 | 4023 | 4024 | 4025 | 4026 | 4027 | 4028 | 4029 | 4030 | 4031 |
| FC | 4032 | 4033 | 4034 | 4035 | 4036 | 4037 | 4038 | 4039 | 4040 | 4041 | 4042 | 4043 | 4044 | 4045 | 4046 | 4047 |
| FD | 4048 | 4049 | 4050 | 4051 | 4052 | 4053 | 4054 | 4055 | 4056 | 4057 | 4058 | 4059 | 4060 | 4061 | 4062 | 4063 |
| FE | 4064 | 4065 | 4066 | 4067 | 4068 | 4069 | 4070 | 4071 | 4072 | 4073 | 4074 | 4075 | 4076 | 4077 | 4078 | 4079 |
| FF | 4080 | 4081 | 4082 | 4083 | 4084 | 4085 | 4086 | 4087 | 4088 | 4089 | 4090 | 4091 | 4092 | 4093 | 4094 | 4095 |

# APPENDIX K

## COMMAND SUMMARY

| COMMAND NAME | PARAMETERS | DESCRIPTION |
|---|---|---|
| DUMP. | LOC, FWA-LWA(; . . . ;LOC, FWA-LWA) | Display the area of memory. FWA-LWA, whenever the instruction at LOC executes. |
| FEND | None | Execute the last simulation and terminate the entire run. |
| INPUT | VALUE(; . . . ;VALUE) | Define the data to be read by simulated I/O instructions. |
| INSTR. | LOC(; . . . ;LOC) | Display the processor registers whenever the instruction at LOC executes. |
| LIMIT | NO | Specify the total number of instructions executed. |
| PATCH | LOC,VALUE(; . . . ;LOC,VALUE) | Initialize each memory location, LOC, to VALUE. |
| REFER. | LOC(; . . . ;LOC) | Display the processor register whenever the instruction at LOC is referenced by another instruction. |
| SETP. | LOC(,PSL=VALUE) (,PSU=VALUE) | Set the program status byte (lower and/or upper) to VALUE whenever the instruction at LOC executes. |
| SETR. | LOC(,R0=VALUE). . .(R6=VALUE) | Set the general purpose registers to VALUE whenever the instruction at LOC executes. |
| SROM | FWA-LWA | Specify the boundaries of Read-Only Memory. |
| START | LOC | Start the simulated program execution at LOC. |
| STAT | None | Display instruction statistics at end of program execution. |
| STOP. | LOC(; . . . ; LOC) | Terminate the program execution when the instruction at LOC executes. |
| TEND | None | Execute the last simulation and prepare to read the User Commands for the next simulation. |
| TRACE. | FWA-LWA(; . . . ;FWA-LWA) | Display the processor registers whenever an instruction executes, which lies within the area of memory, FWA-LWA. |

# APPENDIX L

## ERROR MESSAGES

Whenever the Simulator detects an error in the User Commands, it prints one of the following error messages:

ERROR IN OBJECT MODULE CARD NUMBER
the 2650 object module is incorrectly formatted.

INPUT DATA TABLE OVERFLOW
an INPUT command attempted to expand the simulated data input buffer beyond its limit (200 bytes).

PARAMETER OUT OF RANGE
a User Command either contains an address which is outside the bounds of simulated memory or the command defines a datum which is larger than one byte ($255_{10}$).

SIM MEMORY EXCEEDED
a 2650 object module loads into an area which is outside of simulated memory.

SYNTAX ERROR IN COMMAND
the command parameters are either missing or in error.

TOO MANY COMMANDS
the maximum number of dynamic commands has been exceeded.

TOO MANY DUMP COMMANDS
the maximum number of DUMP commands has been exceeded.

TOO MANY SET REGISTER COMMANDS
the maximum number of SETR. commands has been exceeded.

TOO MANY SET PSB COMMANDS
the maximum number of SETP. commands has been exceeded.

## UNRECOGNIZED COMMAND

a command has been read which is unknown to the Simulator.

## UNEXPECTED END OF FILE

either the object module or the set of User Commands is missing, or one of their respective card decks is incorrectly formatted, or the FEND command is missing.

Whenever the Simulator detects an error while the simulated program is executing it prints one of the following error messages:

## ADDRESS OUT OF RANGE

an instruction attempted to access a location which lies outside of simulated memory.

## INSUFFICIENT INPUT DATA

a I/O instruction attempted to read another datum from the input data buffer (INPUT) after all the data from the buffer had been read. The simulated input register remains unchanged i.e., the instruction is essentially ignored, and program execution continues.

## LC=          ATTEMPT TO STORE INTO ROM

an instruction attempted to store data into the area designated as ROM (SROM).

## LC EXCEEDS MEMORY

the program attempted to execute a memory location which lies outside of simulated memory.

## NO KNOWN OPCODE

the program attempted to execute a memory location which did not contain a valid instruction. Either the program was modified during execution or the program is attempting to execute data.

# APPENDIX M

## SIMULATOR RESTRICTIONS

### SIMULATOR RESTRICTIONS

1. The simulated memory reserved by the Simulator for program storage is limited to 2048 bytes.* Thus, the Simulator will accept only programs or program segments which fit into this area. This implies that the 2650 paging facility (page size = 8192 bytes) cannot be simulated.

2. Some User Commands are limited in the amount of entries they may accept.

| COMMAND | LIMIT |
|---|---|
| DUMP. | 5 LOC's |
| SETR. | 4 LOC's |
| SETP. | 2 LOC's |
| INPUT | 200 VALUE's |
| All "dynamic" commands | 30 LOC's (for TRACE. count 1 for each set of FWA-LWA) |

# APPENDIX N

## SIMULATOR RUN PREPARATION

In order to prepare a program for execution by the Simulator, the programmer:

1. Codes a program in 2650 Assembly Language.
2. Assembles the program until no assembly errors occur.
3. Obtains the object module and listing for the assembled program.
4. Generates command cards using addresses from the listing of the assembled program.
5. Submits the object module and the command cards in that order for a Simulator run.

# signetics

a subsidiary of **U.S. Philips Corporation**