

PSIM: Processor SIMulator (version 4.2)

by Charles E. Stroud, Professor
Dept. of Electrical & Computer Engineering
Auburn University
July 23, 2003

ABSTRACT

A simulator for a basic stored program computer architecture is described which graphically displays the architecture while showing the detailed operation on a per clock cycle basis. The instruction set consists of sixteen instructions that can be combined to form many capabilities including conditional branching. The simulator includes an assembler for compiling assembly language programs to the machine language code used by the simulator, the ability to display values in decimal and hexadecimal formats, and the ability to display and write to a file the contents of the program memory at any point in the simulation.

INTRODUCTION

Computers consist of a collection of basic digital circuits such as counters, registers, multiplexers, adders, and control logic. Yet the combination of these basic components can be made, via a program, to perform a myriad of complex operations. Understanding computers requires a thorough understanding of the architecture, the instruction set, and the operation of the processor when executing the various instructions of a program. Obtaining this understanding can be difficult when the instruction set is large or the architecture is complicated such that the simplicity of the individual components of the processor is hidden by the complexity of the complete system.

The purpose of this processor simulator (called PSIM) is to provide a very basic ALU-based architecture with a small but versatile instruction set that can be quickly and easily understood by users. This simple processor will then serve as a platform from which more complex commercial processors can be understood and seen to be similar but to support more registers and instructions. The simulator shows graphically the architecture of the microprocessor including the various registers and interconnections. The user is given per clock cycle control of the execution of their program and is shown the data contained in each register, the instruction cycle, and the control signals used to activate the processing of the data by each of the components of the processor. A number of options are provided via the menu driven simulator to allow the user to assemble and load a program into the program memory, view or store (in a file) the contents of the program memory, and select the data representation in terms of decimal or hexadecimal values. The processor architecture implemented in the simulator was initially based on the "simple computer" architecture described in [1] with additional instructions and features added for conditional branching.

This document provides an introduction to and overview of the PSIM architecture and simulator to allow the user to begin writing and simulating programs. While reading this document, the user should access the simulator and assemble and run the example program in order to maximize familiarity with the simulator as well as the architecture and operation of the processor. PSIM should run on any PC since it uses the standard PC graphics character set. The following sections describe accessing simulator, the menu commands, the architecture and operation of the processor, the assembler, and the instruction set.

THE SIMULATOR SCREEN

PSIM is a DOS program and can be accessed by either opening a DOS Prompt window, changing to the directory that contains the PSIM executable (psim.exe), and typing "psim", or by double clicking the PSIM icon in a file manager. The simulator screen has three sections consisting of the command menu (upper right side of screen), the architecture of the processor (center of screen), and the instruction cycle information box (lower right side of screen) as illustrated in Figure 1. The instruction cycle information box informs the user of which of the three instruction cycles (FETCH, DECODE, or EXECUTE) is currently being processed. In addition, the instruction being executed is displayed during the execute cycle(s) and the register transfer level description of each step of the instruction cycle is also displayed in the instruction cycle information box. The control logic outputs are highlighted when active to indicate which register level

operation will take place on the next clock cycle. The value contained in each register is also displayed. The command menu and processor architecture are described in detail in the following sections.

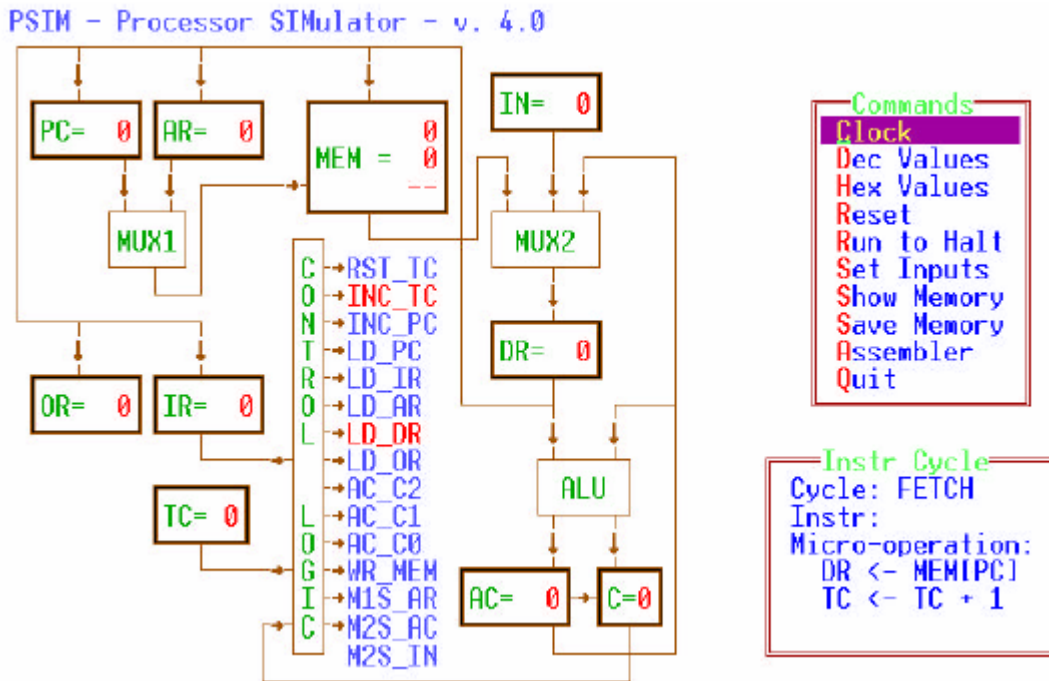


Figure 1. Simulator Screen

THE COMMAND MENU

The command menu appears on the upper right side of the PSIM screen as illustrated in Figure 1. The commands contained in the menu can be accessed by using the up/down arrow keys or by typing the highlighted letter (first letter) of the command. Note that continuing to type the same first letter will cycle through the commands that have the same first letter. Once a command has been selected, it can be executed by pressing the ENTER key or the left mouse button.

The CLOCK command applies one clock cycle to the processor simulation enabling data to be transferred or manipulated in conjunction with the instruction and instruction cycle being processed. Between clock cycles, all data values held in registers represent those values resulting from the previous clock cycle while all highlighted control signals represent actions that will take place at the next clock cycle. Similar to the control signals, the instruction cycle information represents the instruction cycle to be processed during the next clock cycle.

The DECIMAL (DEC) and HEXADECIMAL (HEX) VALUES commands allow the user to choose the representation format of the data contained in the registers as either being in decimal or hexadecimal, respectively. The selected choice of format is also carried over into the display of the program memory as well as the format of program memory data written into an output file. In addition, the decimal/hexadecimal format applies to the addresses associated with the program memory. At the beginning of the simulation, the default data display format is hexadecimal.

The RESET command clears the data held in each register. Resetting of the register values results in the beginning of a FETCH cycle at the first address location in the program memory. The RESET command allows the user to re-start a program at any time during a simulation session. This command may be needed when a new program has been assembled and loaded into the program memory after a previous program has been executed during the same simulation session.

The RUN TO HALT command provides continuous clock cycles to the assembled program execution until a HALT instruction is encountered. This command can be executed at any time after the first DECODE cycle and is primarily intended for checking the proper execution of an assembly language program written for

PSIM. As a result of this capability, the insertion of a HALT instruction at the end of an assembly language program is recommended.

The SET INPUTS command allows the user to specify the values at the Input data bus (IN), or Input Register, in hexadecimal or decimal format, depending on the last value selection chosen (DEC or HEX). The values contained in the Input Register (IN) can be set at any time during the simulation process and will be displayed in hexadecimal or decimal based on the format selected by the DEC or HEX commands.

The SHOW and SAVE MEMORY commands, respectively, display on the screen and store in a file the contents of the program memory (MEM) whenever the commands are executed. This allows the user to inspect the assembled program in terms of the values and locations of the actual machine language. In addition, the user can inspect the contents of memory locations that have been manipulated as a result of the execution of the program. These commands can be executed at any time to examine the actions of a given micro-operation sequence on the contents of the program memory (MEM). The SAVE MEMORY command is useful in storing the assembled machine language at the beginning of program execution or in saving the results after the program has been executed. The format for the program memory (MEM) display is an array of sixteen rows by sixteen columns of data in the selected format (DEC or HEX). The addresses associated with the data begin with address 0 in the upper left hand corner and increases as one moves across the columns then down the rows such that address 255 (or FF in HEX) is located in the lower right hand corner of the screen. The value for the upper 4 bits (most significant bits denoted AH for address high) of the 8-bit address is displayed down the left hand side of the display and the value for the lower 4 bits (least significant bits denoted AL for address low) of the address is displayed across the top of the display.

The ASSEMBLER command is used to assemble a program for the simulator. When this command is executed, the user is asked for the name of the file containing the program to be assembled. Once the file name has been entered, a number of syntactical checks are performed as the program is assembled and loaded into the program memory. Any errors found are reported to the user. Otherwise, the user can return to the simulation screen by pressing the ENTER key or the left mouse button. The assembler is discussed in more detail in a subsequent section.

The final command, QUIT, simply exits the simulator. The default command for the menu is the CLOCK command which will allow the user to continue stepping through the simulation by repeatedly pressing the ENTER key. During the execution of these commands in conjunction with a program, the user can observe the machine language program instructions and data traversing the various registers of the processor. Those registers are introduced and discussed in the following section.

THE ARCHITECTURE

A block diagram of the basic processor can be seen on the simulation display screen where the direction of data flow is indicated by the arrows as illustrated in Figure 1. The program memory (MEM) represents a Random Access Memory (RAM) with an 8-bit address bus (which translates to 256 memory locations) and 8 bits of data per memory location. There are five registers with parallel load capability which include an 8-bit Data Register (DR), an 8-bit Address Register (AR), an Arithmetic-Logic Unit (ALU) in conjunction with an 8-bit Accumulator Register (AC) and a 1-bit Carry register (C), an 8-bit Output Register (OR), and a 4-bit Instruction Register (IR). The 8-bit Input data bus (IN) can also be considered as a register that is parallel loaded by the user. There are two binary counters with increment and reset capabilities which include a 3-bit Timing Counter (TC) and an 8-bit Program Counter (PC) which also has a parallel load capability. The Carry register (C) is a 1-bit register which holds the carry-out data bit or logic operation (NAND, NOR, or OR) bit that can result from ALU operations. All registers and counters have a reset capability activated by the RESET command and all registers and counters will hold their current data when none of their control signals are activated. The contents of all registers are displayed in the specified format (DEC or HEX) on the PSIM display screen. Three memory locations in the program memory (MEM) are also displayed with the center value representing the data stored in the address location pointed to by the current value of the Program Counter (PC) and the upper and lower data values representing the contents of memory locations PC+1 and PC-1, respectively. This allows the user to see the previous and next instruction or operand in the MEM.

In addition to the RAM, registers, and counters, there are two multiplexers (MUX1 and MUX2), an ALU, and a combinational logic network referred to as the Control Logic. MUX1 consists of eight 2-to-1 multiplexers

and MUX2 consists of eight 3-to-1 multiplexers. The Arithmetic-Logic Unit (ALU) is an 8-bit combinational logic function with add, increment, complement, AND, NOR, and XOR capabilities. Finally, the Control Logic is a decoder circuit that takes input from the Instruction Register (IR), Timing Counter (TC), and Carry register (C) to produce control signals to the various registers, counters, multiplexers, and RAM to perform the micro-operation sequences of each instruction cycle. When a control signal is activated as a result of program execution, the net name representing the control signal is highlighted on the PSIM display screen. The control signals are summarized in Table 1. The ALU/AC operations are summarized in Table 2.

The MEM is used to store the program being executed and to store results or temporary data produced by the program. The address for the MEM is supplied by MUX1 which can select addresses from either the PC or the AR. Data written into the MEM is supplied by the DR. Data read from the MEM is loaded into the DR through MUX2. Therefore, the DR serves as a hub for all data entering or exiting the MEM, supplies data to all of the registers and counters except the TC, and can load data from the MEM, IN, AC via MUX2. The IR is used to hold the current instruction being executed and decoded by the Control Logic. The PC is used to address the MEM to obtain the instruction or operands to be processed and is usually incremented to the next MEM location once data has been transferred to the DR. The AR is used to hold addresses to the MEM to access operands or to write data into the MEM. The AC is the primary data processing register in the processor with capabilities to manipulate data within the AC or to load in new data from the DR via the ALU. The C register holds the carry-out bit that can result from an arithmetic operation or the logic value that can result from a logic operation in the ALU. The C register has the distinction of being the only register that drives the Control Logic to facilitate conditional branching during program. Finally, the TC is used to step through the FETCH, DECODE, and EXECUTE cycles with the length of the EXECUTE cycle being determined by the instruction being decoded via the Control Logic.

Table 1. Control Logic Output Signal Operations

Signal	Operation	Notation
RST_TC	Reset Timing Counter	$TC \leftarrow 0$
INC_TC	Increment Timing Counter	$TC \leftarrow TC+1$
INC_PC	Increment Program Counter	$PC \leftarrow PC+1$
LD_PC	Parallel Load Program Counter	$PC \leftarrow DR$
LD_IR	Parallel Load Instruction Register	$IR \leftarrow DR$
LD_AR	Parallel Load Address Register	$AR \leftarrow DR$
LD_DR	Parallel Load Data Register	$DR \leftarrow \text{MUX2 output data}$
LD_OR	Parallel Load Output Register	$OR \leftarrow DR$
AC_C2-0	ALU/Accumulator Control Signals	See Table 2
WR_MEM	Write Program Memory	$\text{MEM}[\text{MUX1 address}] \leftarrow DR$
M1S_AR	MUX 1 selects Address Register	$\text{MUX1} \leftarrow AR$
M2S_AC	MUX 2 selects Accumulator	$\text{MUX2} \leftarrow AC$
M2S_IN	MUX 2 selects Input Register	$\text{MUX2} \leftarrow IN$

Table 2. ALU/Accumulator Control

AC_C2	AC_C1	AC_C0	Operation	Notation
0	0	0	Hold Data	No operation, $AC \leftarrow AC$
0	0	1	Load Data	$AC \leftarrow DR$
0	1	0	Complement	$AC \leftarrow AC', C \leftarrow C'$
0	1	1	Arithmetic Increment	$AC \leftarrow AC+1, C \leftarrow \text{carry-out}$
1	0	0	Arithmetic Add	$AC \leftarrow AC+DR, C \leftarrow \text{carry-out}$
1	0	1	Logical AND	$AC \leftarrow AC \bullet DR, C \leftarrow \text{NAND of result}$
1	1	0	Logical NOR	$AC \leftarrow (AC+DR)', C \leftarrow \text{NOR of result}$
1	1	1	Logical XOR	$AC \leftarrow AC \oplus DR, C \leftarrow \text{OR of result}$

During simulation of a program, the data values in each register can be seen to move through the architecture or be modified as a result of the activation of the various control signals produced by the Control Logic. The activation of the control signals correlates to the various activities taking place during each

micro-operation sequence of the instruction execution. Therefore, the highlighted control signals can be used as an indicator of which registers will be acted upon during the next clock cycle.

THE INSTRUCTION SET

The instruction set for the processor provides the programmer with the ability to manipulate data in and data transfers between the various registers in the processor. The instruction set for this basic processor currently consists of sixteen instructions with two additional instructions to the assembler. Each instruction requires a minimum of three clock cycles to completely execute: one clock cycle for the FETCH cycle to retrieve the instruction from the MEM into the DR (the fetch cycle), a second clock cycle to transfer the instruction into the IR where it is decoded by the Control Logic (the DECODE cycle), and a third (and possibly more) clock cycle(s) to perform the operation specified by the instruction (the EXECUTE cycle).

There are three types of instructions (employing two types of addressing modes) contained in the processor instruction set as summarized in Table 3. These same three types of instructions are found in instruction sets for commercial processors along with additional types. Since the processor works with binary values, each instruction is assigned a code word called an opcode (for operation code). The instruction may be accompanied by additional binary values representing data (operand) or address.

Table 3. Instruction Types Supported by Simulator

Instruction Type	Instruction Fields	
Implied Operand	OPCODE	
Immediate Operand	OPCODE	OPERAND
Direct Address	OPCODE	ADDRESS

The implied operand instruction consists of an opcode only, with the operand being the data already contained in a register of the processor. For example, complementing the AC would require only an opcode since the value to be complemented is assumed to reside in the AC. The opcode (as stored in MEM) for an immediate operand instruction is immediately followed in MEM by the data (operand) that is retrieved from the MEM during the execution of the instruction. For example, loading data into the AC requires the data to be encoded in the instruction (and stored in MEM) immediately following the instruction opcode. The final type of instruction employs direct addressing by providing an address with the opcode where the accompanying address identifies the MEM location where the operand can be obtained. As a result, the address is retrieved from the MEM during the execution of the instruction, the address is then loaded into the AR, and the contents of the AR are then used to address the MEM to retrieve the operand.

The complete instruction set currently supported by the processor is summarized in Table 4. The assembly language format of the instruction is given along with the opcode assigned to each instruction. An operational description is included along with a register transfer level description of each instruction. In the register transfer level description, the first entry represents the register receiving the data (as indicated by the arrow "←") and the entry following the arrow represents the data to be put into the receiving register.

The processing of any instruction begins with the FETCH cycle which retrieves the instruction from the MEM and loads it into the DR. The address used to read the MEM is supplied by the PC via MUX1. The FETCH cycle is defined by the TC being at a count value of zero. The micro-operation sequence of the FETCH cycle is given in the micro-operation sequence chart in Table 5 which includes the value of the TC, a register transfer level description of the operation, and the active control signals produced by the control logic.

The next phase of the instruction cycle common to the processing of all instructions is the DECODE cycle. During the DECODE cycle, the instruction is loaded into the IR where it is decoded by the Control Logic to produce the control signals necessary to execute the specific instruction. At this point, the PC can be incremented to point to the next instruction or the immediate operand or direct address that may accompany the current instruction opcode being decoded. The micro-operation sequence chart for the DECODE cycle is given in Table 5.

The final phase of the instruction cycle is the EXECUTE cycle which may require multiple clock cycles to complete execution. The number of clock cycles depends on the complexity of the instruction being processed. As a result, the micro-operation sequence required for the execution of the sixteen instructions in the instruction set may vary between 1 and 5 clock cycles which translate into TC count values. It is at

this point in the instruction cycle that the instruction to be executed is known and the micro-operation sequences begin to differ to yield the unique instruction execution phase. The micro-operation sequences are given in Table 5 for the execution of each of the sixteen instructions currently supported by the PSIM processor architecture. The micro-operation sequence chart is shown graphically in Figure 2 where each transition from one timing state to the next assumes $TC \leftarrow TC+1$ except where noted. Following Table 5, notes are provided (as indicated next to the instruction name in the table) to give additional information regarding particular instructions.

Table 4. Instruction Set

Instruction	Opcode	Operation Description	Register Transfer
HLT	0000	Halt program execution	$TC \leftarrow TC$
NOP	0001	No operation	$TC \leftarrow 0$
INA	0010	Increment AC	$AC \leftarrow AC+1, C \leftarrow \text{carry-out}$
CMA	0011	Complement AC	$AC \leftarrow AC', C \leftarrow C'$
ISZ	0100	Increment/skip if zero	if $C=0, PC \leftarrow PC+2$
LDI OPRD	0101	Load OPRD in AC	$AC \leftarrow OPRD, C \leftarrow 0$
ADI OPRD	0110	Add OPRD to AC	$AC \leftarrow AC+OPRD, C \leftarrow \text{carry-out}$
BUN ADRS	0111	Branch unconditionally	$PC \leftarrow OPRD$
STA ADRS	1000	Store AC in MEM[ADRS]	$MEM[ADRS] \leftarrow AC$
STI ADRS	1001	Store IN in MEM[ADRS]	$MEM[ADRS] \leftarrow IN$
LDA ADRS	1010	Load MEM[ADRS] in AC	$AC \leftarrow MEM[ADRS], C \leftarrow 0$
LDO ADRS	1011	Load MEM[ADRS] in OR	$OR \leftarrow MEM[ADRS]$
ADA ADRS	1100	Add MEM[ADRS] to AC	$AC \leftarrow AC+MEM[ADRS], C \leftarrow \text{carry-out}$
AND ADRS	1101	Logic AND of MEM[ADRS] and AC	$AC \leftarrow AC \bullet MEM[ADRS], C \leftarrow \text{NAND}(AC)$
NOR ADRS	1110	Logic NOR of MEM[ADRS] and AC	$AC \leftarrow (AC+MEM[ADRS])', C \leftarrow \text{NOR}(AC)$
XOR ADRS	1111	Logic XOR of MEM[ADRS] and AC	$AC \leftarrow AC \oplus MEM[ADRS], C \leftarrow \text{OR}(AC)$

Table 5. Micro-Operation Sequence Chart for Instruction Set

Cycle	Instruction	TC Value	Micro-operations	Active Control Signals	
Fetch	all instructions	0	$DR \leftarrow MEM[PC]$ $TC \leftarrow TC + 1$	LD_DR INC_TC	
Decode	all instructions	1	$IR \leftarrow DR$ $PC \leftarrow PC + 1$ $TC \leftarrow TC + 1$	LD_IR INC_PC INC_TC	
Execute	HLT (note 1)	2	$TC \leftarrow TC$	no signals active	
	NOP	2	$TC \leftarrow 0$	RST_TC	
	INA	2	$AC \leftarrow AC + 1, C \leftarrow \text{carry}$ $TC \leftarrow 0$	AC_C1, AC_C0 RST_TC	
	CMA	2	$AC \leftarrow AC', C \leftarrow C'$ $TC \leftarrow 0$	AC_C1 RST_TC	
	ISZ (note 2)		2	if $C=0, PC \leftarrow PC + 1$ $TC \leftarrow TC + 1$	INC_PC (only if $C=0$) INC_TC
			3	if $C=0, PC \leftarrow PC + 1$ $TC \leftarrow 0$	INC_PC (only if $C=0$) RST_TC
	LDI OPRD (note 3)		2	$DR \leftarrow MEM[PC]$ $TC \leftarrow TC + 1$	LD_DR INC_TC
			3	$AC \leftarrow DR, C \leftarrow 0$ $PC \leftarrow PC + 1$ $TC \leftarrow 0$	AC_C0 INC_PC RST_TC
	ADI OPRD (note 4)		2	$DR \leftarrow MEM[PC]$ $TC \leftarrow TC + 1$	LD_DR INC_TC
			3	$AC \leftarrow AC + DR, C \leftarrow \text{carry}$ $PC \leftarrow PC + 1$ $TC \leftarrow 0$	AC_C2 INC_PC RST_TC

Table 5. Micro-Operation Sequence Chart for Instruction Set (continued)

Cycle	Instruction	TC Value	Micro-operations	Active Control Signals
Execute	BUN ADRS (note 5)	2	DR ← MEM[PC] TC ← TC + 1	LD_DR INC_TC
		3	PC ← DR TC ← 0	LD_PC RST_TC
	STA ADRS (note 6)	2	DR ← MEM[PC] TC ← TC + 1	LD_DR INC_TC
		3	AR ← DR PC ← PC + 1 TC ← TC + 1	LD_AR INC_PC INC_TC
		4	DR ← AC TC ← TC + 1	LD_DR, M2S_AC INC_TC, M1S_AR
		5	MEM[AR] ← DR TC ← TC + 1	WR_MEM, M1S_AR INC_TC
		6	TC ← 0	RST_TC, M1S_AR
	STI ADRS (note 6)	2	DR ← MEM[PC] TC ← TC + 1	LD_DR INC_TC
		3	AR ← DR PC ← PC + 1 TC ← TC + 1	LD_AR INC_PC INC_TC
		4	DR ← IN TC ← TC + 1	LD_DR, M2S_IN INC_TC, M1S_AR
		5	MEM[AR] ← DR TC ← TC + 1	WR_MEM, M1S_AR INC_TC
		6	TC ← 0	RST_TC, M1S_AR
	LDA ADRS (note 7)	2	DR ← MEM[PC] TC ← TC + 1	LD_DR INC_TC
		3	AR ← DR PC ← PC + 1 TC ← TC + 1	LD_AR INC_PC INC_TC
		4	DR ← MEM[AR] TC ← TC + 1	LD_DR & M1S_AR INC_TC
		5	AC ← DR, C ← 0 TC ← 0	AC_C0 RST_TC
	LDO ADRS (note 7)	2	DR ← MEM[PC] TC ← TC + 1	LD_DR INC_TC
		3	AR ← DR PC ← PC + 1 TC ← TC + 1	LD_AR INC_PC INC_TC
		4	DR ← MEM[AR] TC ← TC + 1	LD_DR & M1S_AR INC_TC
		5	OR ← DR TC ← 0	LD_OR RST_TC
ADA ADRS (note 8)	2	DR ← MEM[PC] TC ← TC + 1	LD_DR INC_TC	
	3	AR ← DR PC ← PC + 1 TC ← TC + 1	LD_AR INC_PC INC_TC	
	4	DR ← MEM[AR] TC ← TC + 1	LD_DR, M1S_AR INC_TC	
	5	AC ← AC + DR, C ← carry TC ← 0	AC_C2 RST_TC	

Table 5. Micro-Operation Sequence Chart for Instruction Set (continued)

Cycle	Instruction	TC Value	Micro-operations	Active Control Signals
Execute	AND ADRS (note 8)	2	DR ← MEM[PC] TC ← TC + 1	LD_DR INC_TC
		3	AR ← DR PC ← PC + 1 TC ← TC + 1	LD_AR INC_PC INC_TC
		4	DR ← MEM[AR] TC ← TC + 1	LD_DR, M1S_AR INC_TC
		5	AC ← AC • DR, C ← NAND(AC) TC ← 0	AC_C2, AC_C0 RST_TC
	NOR ADRS (note 8)	2	DR ← MEM[PC] TC ← TC + 1	LD_DR INC_TC
		3	AR ← DR PC ← PC + 1 TC ← TC + 1	LD_AR INC_PC INC_TC
		4	DR ← MEM[AR] TC ← TC + 1	LD_DR, M1S_AR INC_TC
		5	AC ← (AC + DR)', C ← NOR(AC) TC ← 0	AC_C2, AC_C1 RST_TC
	XOR ADRS (note 8)	2	DR ← MEM[PC] TC ← TC + 1	LD_DR INC_TC
		3	AR ← DR PC ← PC + 1 TC ← TC + 1	LD_AR INC_PC INC_TC
		4	DR ← MEM[AR] TC ← TC + 1	LD_DR, M1S_AR INC_TC
		5	AC ← AC ⊕ DR, C ← OR(AC) TC ← 0	AC_C2, AC_C1, AC_C0 RST_TC

Notes:

1. For the HLT instruction, all register values, including the TC are held constant by de-activation of all control signals. As a result, program execution halts.
2. The ISZ instruction is the only instruction with conditional execution dependent on the contents of the single-bit C register. If the content of C is a logic 0, the PC is incremented twice to skip two program memory locations (to skip an immediate operand or a direct address instruction), otherwise the PC is incremented only once.
3. For the LDI OPRD instruction, the operand is read from the MEM during TC=2 and the PC is incremented to point to the next instruction.
4. The ADI OPRD instruction executes identically to that of the LDI OPRD but the select input to MUX2 is activated to select the output of the AC.
5. The BUN ADRS instruction may be viewed as an immediate operand type of instruction where the ADRS is in reality an operand loaded into the PC.
6. For the STA (STI) ADRS instruction, the address of the MEM location, where the accumulator (input register) value is to be stored, is read from the MEM during TC=2 and the PC is incremented to point to the next instruction. The address is then loaded into the AR. The contents of the AC (IN) are loaded into the DR and the MEM is written using the contents of AR as the address and the contents of the DR as the data to be written. These are the only instructions that write data into the MEM during which the WR_MEM control signal is activated. The extra cycle, TC=6, is used to prevent address/data races to the asynchronous MEM during the write cycle.
7. For the LDA (LDO) ADRS instruction, the direct address of the operand is read from the MEM during TC=2 and the PC is incremented to point to the next instruction. The read address is then loaded into the AR and a read of the MEM using the contents of AR as the read address produces the operand to be loaded into the AC (OR).
8. The ADA, AND, NOR, and XOR ADRS instructions execute identically to that of the LDA ADRS instruction but ALU/AC control signals are activated for the appropriate combination of AC and DR. Note that the value of the C register in the case of the AND, NOR, and XOR instructions will be the NAND, NOR, and OR, respectively, of the 8-bit contents of the AC at the end of the instruction; in other words, the logic operation for the C register is performed on the new data entering the AC.

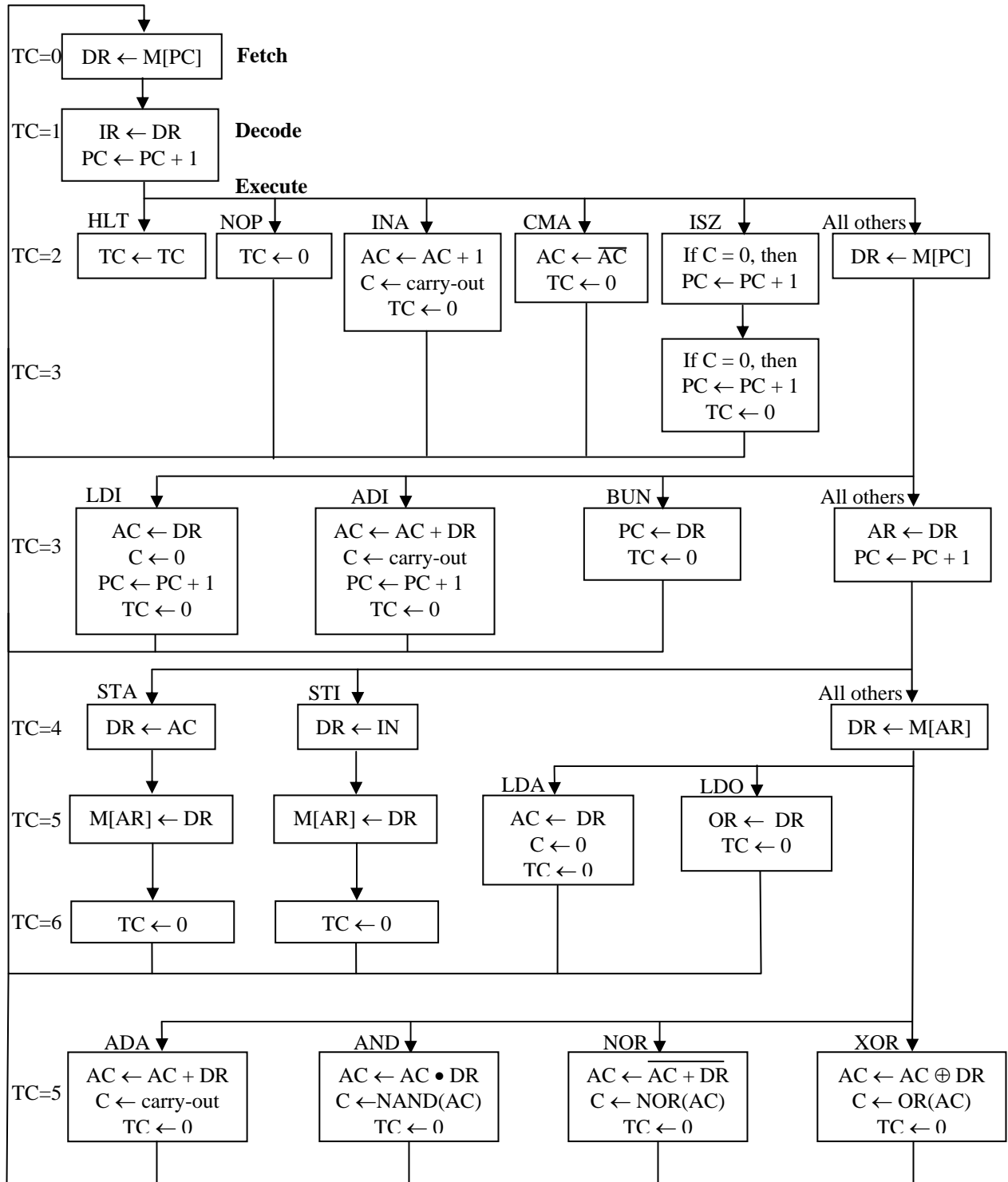


Figure 2. Micro-operation Sequence Chart

There are two additional instructions that are only used by the assembler and do not directly translate to instructions in the program memory. These instructions include:

MEM ADRS OPRD
VAL DEC or HEX

The MEM instruction to the assembler results in the data value specified by the OPRD field being loaded into the program memory at the address location specified by the ADRS field. This assembler instruction can be used to define data values in the program memory that can be accessed by instructions such as LDA and ADA. The VAL instruction to the assembler specifies the data format of subsequent OPRD and/or ADRS fields accompanying assembly language instructions (where "DEC" indicates decimal format and "HEX" indicates hexadecimal format). The default format is hexadecimal format but the data format can be changed at any point in the assembly language program and as often as desired. However, once specified all subsequent data values are assumed to be of the same format until a new VAL instruction is given.

THE ASSEMBLER

The assembler compiles an assembly language program into binary machine language and loads the machine language program into the program memory. The assembler must be accessed prior to simulating a program. The format for the assembly language program is any sequence and combination of the 18 instructions (16 for the processor and 2 for the assembler) along with decimal or hexadecimal values for the appropriate OPRD and ADRS fields. The assembler reads the assembly language program and converts the instructions to the binary opcode placing the first instruction in MEM location 0 followed by the subsequent OPRD or ADRS value in MEM location 1 if the instruction is not an implied operand instruction type. Otherwise, the opcode for the next instruction is loaded into MEM location 1. The process of translation and loading into the next available MEM location continues until the end of the file is reached. The exceptions to this process are the two assembler instructions where the next available MEM location is not loaded. In the case of the MEM instruction, the OPRD data is loaded into the specified ADRS location. In the case of the VAL instruction, the format for translating the OPRD and ADRS arguments in the assembly language program is changed and nothing is written into the MEM.

The assembler provides a number of syntactical checks of the program including invalid instructions and OPRD and ADRS values out of range (less than 0 or greater than 255). Note that the assembler is case sensitive and all instructions must be in capital letters; the only exception is HEX data values. In addition, comment statements can be included in the assembly language program at any point except between an instruction and its argument(s). The syntax for a comment statement requires that the comment begin with a "#" and end with a ";" where the "#" and ";" are separated from the text of the comment and other instructions or arguments of instructions by one or more delimiters which include space, tab, or new line. These same delimiters are used to separate instructions and arguments.

EXAMPLE PROGRAM

To give an example of the simulator options and operation, the program illustrated in Figure 3 can be used. The program performs the mathematical operation $83 - (52 + 24)$ and stores the result in address location 250. The example program contains the correct syntax for the PSIM assembler and can be typed into a file (with any text editor) if it was not included with the PSIM executable (file name "exprog.asm").

```
VAL DEC
LDI 52      # load 52 into the AC ;
ADI 24      # add 24 to the AC ;
CMA        # compliment AC ;
INA        # increment AC ;
ADI 83      # add 83 to the AC ;
VAL HEX
STA fa      # store the contents of AC in M[250] ;
HLT        # stop simulation ;
```

Figure 3. Example Program for PSIM - "exprog.asm"

The first instruction is to the assembler only and specifies that the subsequent operand values will be in decimal format. During execution of the program, the LDI instruction will load the value 52 into the AC and the ADI instruction will add the value 24 to the contents of the AC. The sum 76 will then be converted to a 2's complement representation of -76 by complementing and incrementing the value via the CMA and INA instructions, respectively. The value 83 will then be added to the contents of the AC by the ADI instruction.

The next instruction, the VAL instruction, is to the assembler and specifies that the subsequent values (in this case the value for the address for the STA instruction) will be given in hexadecimal format. During execution of the program, the STA instruction will cause the contents of the AC (a value of 7 at this point) to be stored in MEM location 250 and the HLT instruction will halt the execution of the program.

To execute this example program in PSIM, one must first assemble and load the machine language program in the program memory (MEM). This is accomplished by executing the ASSEMBLER command where the user is asked for the name of the file containing the assembly language program (exprog.asm for this example). Once the program is successfully assembled and loaded into MEM, the user is instructed to press the ENTER key to continue with the simulation. Returning to the simulation screen, the three values displayed in the MEM will be non-zero. The top value represents the operand for the first LDI instruction and will be displayed as a 34 if hexadecimal notation format is selected or as a 52 if the decimal notation format is selected. This value is contained in address location 1 and can also be seen using the SHOW MEMORY command. The center value is displayed as a 5 representing the opcode for the LDI instruction stored in address location 0 in the MEM. The bottom value is displayed as "--" to indicate a non-existing memory location (-1). All register values should be 0 at this point and the processor should be in the FETCH cycle (if not, execute the RESET command).

As the user executes the CLOCK command from the menu, the opcodes for the various instructions can be seen to move into the DR during the FETCH cycle and into the IR during the DECODE cycle. The appropriate control signals will be highlighted to control the movement or modification of data as a result of the decoded instructions in accordance with the micro-operation sequence chart given in Table 5 and Figure 2. The action of writing MEM location 250 with the final result (a value of 7) can be seen by using the SHOW MEMORY command before and after the program memory write operation takes place as illustrated in Figure 4 where 7 has been written into MEM location 250 (the DEC format was used for display).

PSIM - Simulation at PC= 11, TC= 2
CONTENTS OF PROGRAM MEMORY

AH\AL	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	5	52	6	24	3	2	6	83	8	250	0	0	0	0	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
32	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
48	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
64	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
80	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
96	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
112	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
128	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
144	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
160	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
176	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
192	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
208	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
224	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
240	0	0	0	0	0	0	0	0	0	0	7	0	0	0	0	0

Figure 4. Program Memory Contents After PSIM Execution of “exprog.asm”

SUMMARY

The PSIM simulator is intended as an instructional aid for learning the basic operation, architecture, and programming of microprocessors and microcontrollers. The graphic display of the architecture, contents of registers and program memory, instruction cycle information, and activated control signals on a per clock cycle basis enhances the ability of the user to understand the underlying principles on which processors are based. The small instruction set facilitates the ability to begin programming quickly. Yet, the flexibility of the instruction set allows relatively complex programs to be developed. As a result, PSIM forms a platform from which the architecture, operation, and instruction set for more complex processors can be understood.

REFERENCES

- [1] M. Morris Mano, *Computer Engineering Hardware Design*, Prentice Hall, pp. 280-292, 1988.