

ESP

An Educational Computer

Peter Hiscocks

Department of Electrical and Computer Engineering
Ryerson Polytechnic University

phiscock@ee.ryerson.ca

November 13, 2001

Contents

1	Introduction	1
2	The Extremely Simple Processor	2
3	Machine Overview	4
4	The Datapath	4
5	The Control Unit	7
6	Schematic Diagrams	10
7	Microcode	13
8	Acknowledgements	16
	Bibliography and References	16

1 Introduction

This paper describes the *ESP: Extremely Simple Computer*, a small computer used to teach the basic concepts and organization of a microprogrammed computer¹.

¹Also known at Ryerson as the *PPM: Programmable Processor Module*.

Students in the Ryerson Electrical and Computer Engineering program third semester Digital Electronics course, study the architecture of the computer and write both microcode and program code as a 3-week final exercise in the laboratory for the course. The exercise is challenging for students, but provides them with a tangible and non-trivial indication of their capabilities at the conclusion of the course. As well, it introduces concepts of computer design that are expanded upon in a subsequent course in Computer Architecture.

It is hoped that the information provided in this paper is useful and sufficiently complete that others may design and build similar small computers for teaching and demonstration purposes.

2 The Extremely Simple Processor

A closeup view the Extremely Simple Processor is shown in figure 1.

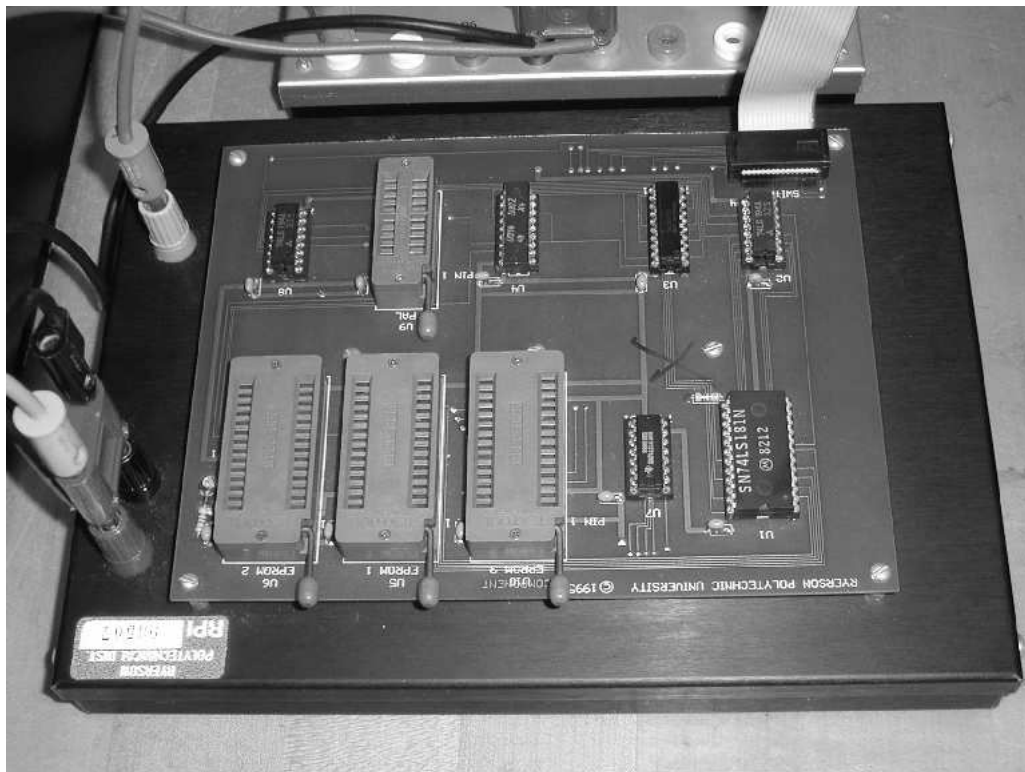


Figure 1: The Extremely Simple Processor: Closeup

The ESP uses a total of 10 readily available integrated circuits. One EPROM, the Program Memory, contains a student program. Two more EPROMs contain the Control Microcode for the computer. The three EPROMs are placed in the large ZIF sockets in the foreground of figure 1. The smaller ZIF socket at the rear holds a 16V8 GAL device which is programmed with miscellaneous combinational logic for the computer.

A general view of the ESP with two other items ancillary equipment is shown in figure 2. The black box at the

top of the photograph provides power to the ESP. The silver box to the right of the ESP provides LED readouts, 4 toggle switches for data input, two clock sources (manual and oscillator) and a reset button.

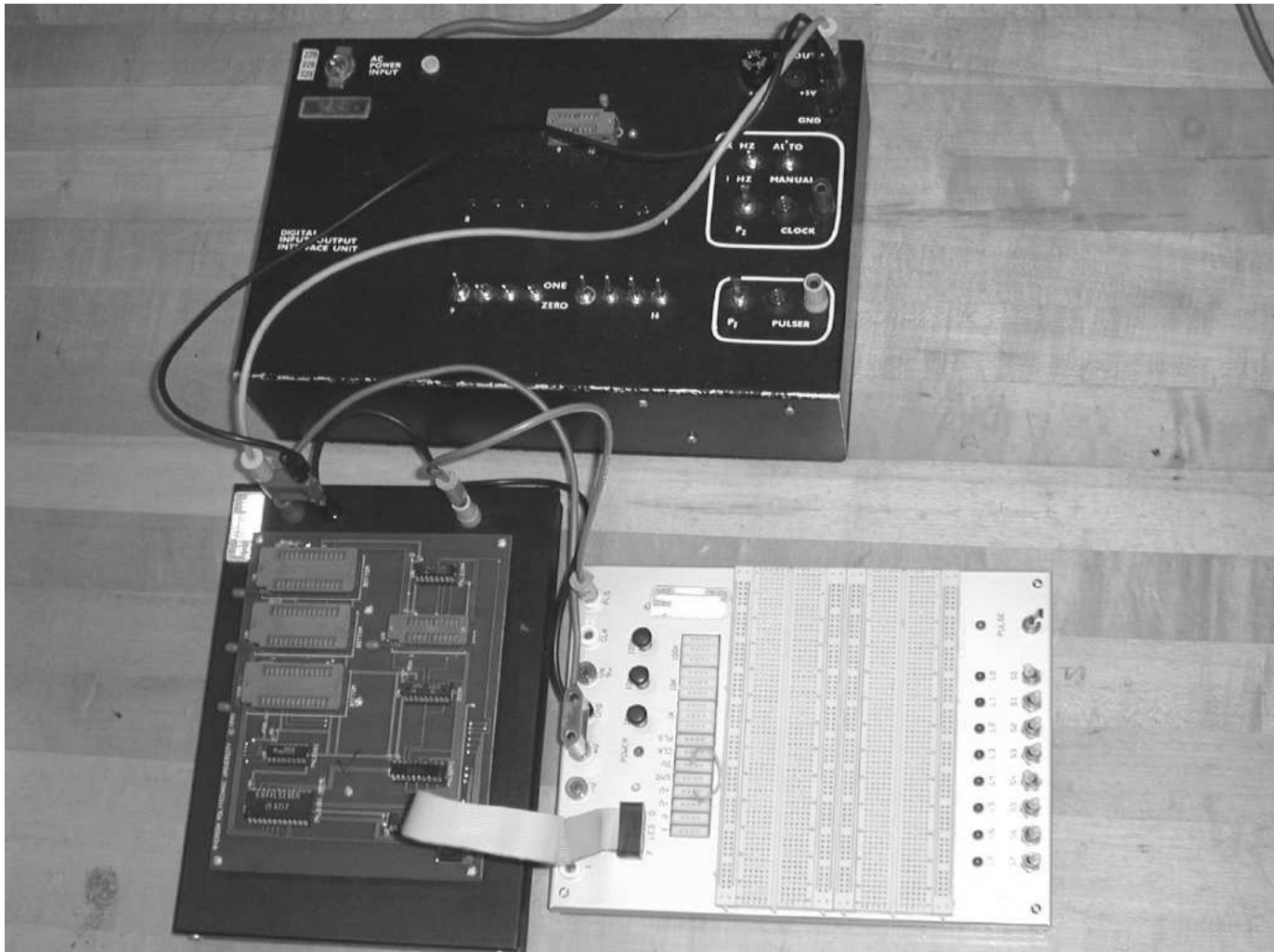


Figure 2: The Extremely Simple Processor: Overview

Some microcode is provided to students and they create the remainder as a lab exercise. When that is working, they write and demonstrate a complete computer program.

Some of the programs that have been assigned to students are as follows:

- Input two 4-bit numbers from the data switches. Calculate and display the sum and difference.
- Count the number of 1's in the 4-bit input data word.

- Input two 2's complement 4-bit numbers. Calculate the sum and convert into signed-magnitude format.
- Input four 4-bit numbers in sequence, compute and display their average.
- Input two 4-bit numbers, compute and display their product.
- Input a 4-bit number **A** into memory. Count from 0 to **A** in sequence, then stop.
- Input two 4-bit numbers **A** and **B**. If $A = B$, then flash the display forever.
- Rotate a 1 bit through the accumulator.
- Program the accumulator to simulate a Johnson (twisted-ring) counter.

Students program the EPROMs using one of several of several lab PCs that are equipped with EPROM programmers².

3 Machine Overview

Specifications for the ESP are shown in figure 3. (Since the ESP is clocked manually, clock speed is not rated.)

Data Word Length	4 bits
Instruction Length	8 bits: 4 bit opcode, 4 bit operand
Microinstruction Length	16 bits
Total ALU functions	32
Number of instructions	16
Conditional Jump?	Yes, three different tests
Program Storage	16 instructions maximum
Working storage	16 × 4 SRAM
Number of ICs	10

Figure 3: ESP Specifications

The machine performs *load and store* operations between a small read-write memory and a single accumulator. It is a Harvard architecture machine – the program and data are in distinct memory spaces. Program text is stored as 8 bit words in an EPROM. Data is moved to and from a 4 bit SRAM.

The complete instruction set is shown in figure 4.

The students use the mnemonics from this table to create an 'assembly language listing' of their computer program. They then hand-translate this assembly language listing into a hexadecimal listing and burn it into the Program EPROM. Since the programs are small – 16 lines maximum – this is quite feasible to do by hand.

An example program is shown in figure 5.

4 The Datapath

The ESP datapath which implements the Load and Store instruction set, is shown in figure 6.

²EPROM Data may be entered directly into the programmer edit buffer. Alternatively, it is possible to create an assembly language source file consisting entirely of DEPOSIT BYTE declarations, and then assemble this to an s-record file. The s-record file may then be loaded into the EPROM programmer.

Name	Op Code	Operand	Mnemonic	Description
Add to Accumulator	0 0 0 0	$N_3N_2N_1N_0$	ADDA N	ACCA := ACCA + (N) + C.
Subtract from Accumulator	0 0 0 1	$N_3N_2N_1N_0$	SUBA N	ACCA := ACCA - (N) - /C. (/C = Borrow Bit)
Input Switches	0 0 1 0	X X X X	INPA	ACCA := Sw
Load Accumulator	0 0 1 1	$N_3N_2N_1N_0$	LDAA N	ACCA := (N)
Store Accumulator	0 1 0 0	$N_3N_2N_1N_0$	STAA N	(N) := ACCA
Jump	0 1 0 1	$N_3N_2N_1N_0$	JMP N	PC := N
Add Switches	0 1 1 0	X X X 0	ADDA S	ACCA := ACCA + Sw + C.
Subtract Switches	0 1 1 0	X X X 1	SUBA S	ACCA := ACCA - Sw - /C.
Logical AND	0 1 1 1	$N_3N_2N_1N_0$	ANDA N	ACCA := (N)&&ACCA
Clear carry	1 0 0 0	X X X 0	CLC	C := 0
Set carry	1 0 0 0	X X X 1	SEC	C := 1
Decrement AC	1 0 0 1	X X X 0	DECA	ACCA := ACCA - 1
Increment AC	1 0 0 1	X X X 1	INCA	ACCA := ACCA + 1
Rotate right	1 0 1 0	X X X 0	RORA	Rotate ACCA right ($Q_3 := C$ and $C := Q_0$).
Rotate left	1 0 1 0	X X X 1	ROLA	Rotate ACCA left ($Q_0 := C$ and $C := Q_3$).
Store switches	1 0 1 1	$N_3N_2N_1N_0$	STSW N	(N) := Sw.
Jump if Carry Set	1 1 0 0	$N_3N_2N_1N_0$	JCS N	(PC) := N if C == 1
Jump if AC equals SW	1 1 0 1	$N_3N_2N_1N_0$	JEQ N	(PC) := N if ACCA == SW
Jump if AC negative	1 1 1 0	$N_3N_2N_1N_0$	JMI N	(PC) := N if ACCA[3]==1
Clear accumulator	1 1 1 1	X X X 0	CLRA	ACCA := 0000
Set accumulator	1 1 1 1	X X X 1	SETA	ACCA := 1111

Term	Description
$N_3N_2N_1N_0$	Memory address, 0 to 15
N	Memory address, 0 to 15
ACCA	Accumulator Register
C	Carry Bit
Sw	Switch register
(N)	Contents of memory address N
:=	is replaced by
Q_0	Least significant bit of ACCA
Q_3	Most significant bit of ACCA
&&	Logical AND
X	don't care bit

Figure 4: ESP Instruction Set

The machine transfers data over a 4 bit Data Bus. Some examples of transfer sequences are:

Load AC Transfer from a memory location, via the B input of the ALU, into the accumulator.

Add to AC Replace the contents of the accumulator with the current contents plus the contents of some memory location. The contents of the accumulator enter the A input of the ALU via the left datapath loop, while data from RAM enters the B input of the ALU. The result is stored back into the accumulator.

Input Switches The contents of the switch register are gated onto the data bus and transferred via the B input of

Input two numbers & calculate both the sum and the difference.

Solution:

- Load a 4-bit number from the switches & store in M1 (memory location #1).
- Load another number from the switches & store in M2.
- Calculate the value of “M1 + M2” & store in M3.
- Calculate the value of “M1 - M2” & store in M4.
- Repeat from start.

Address (Hex)	----- (Binary)	Program word (Hex)	----- (Mnemonic)	----- Comments -----
0	0010 1111	2F	INPA	Input data from switches
1	0100 0001	41	STAA 1	Store data into M1
2	0010 1111	2F	INPA	Input second data from switches
3	0100 0010	42	STAA 2	Store second data into M2
4	1000 1110	8E	CLC	Clear Carry to prepare for ADD
5	0000 0001	01	ADDA 1	Add M1 into Accumulator
6	0100 0011	43	STAA 3	Store "M1 + M2" into M3
7	0011 0001	31	LDAA 1	Load data from M1
8	1000 1111	8F	SEC	Set Carry to prepare for SUB, borrow = /C = 0
9	0001 0010	12	SUBA 2	Subtract M2 from Accumulator
A	0100 0100	44	STAA 4	Store "M1 - M2" into M4
B	0101 0000	50	JMP 0	Jump back to start of program

Figure 5: ESP Program Example

the ALU to the accumulator.

Decrement the accumulator Contents of the accumulator are circulated back into the A input of the ALU, manipulated, and then stored back into the accumulator.

The 74LS181 Arithmetic Logic Unit is uses 5 *function select* lines – four control lines $S_3 \dots S_0$ and a mode control line M – which select one of 32 different operations on two, 4-bit operands.

The output of the ALU, for operands A and B, are summarized in figure 7. Some of the operations are not useful. These table entries are left blank.

The accumulator register is a 74LS194 *4-Bit Bidirectional Universal Shift Register* that can perform 4 different operations based on 2 control signals S_0 and S_1 , as shown in figure 8.

There is also (not shown on the diagram) a 1-bit memory flip-flop, the CARRY flip-flop, that retains any CARRY OUT bit so that the next instruction can test the carry or use it for CARRY IN to the next calculation. Similarly, another 1-bit memory retains the A=B signal for a subsequent conditional jump instruction. The CARRY flip-flop and A=B flip-flop could be regarded as a 2-bit Condition Code Register.

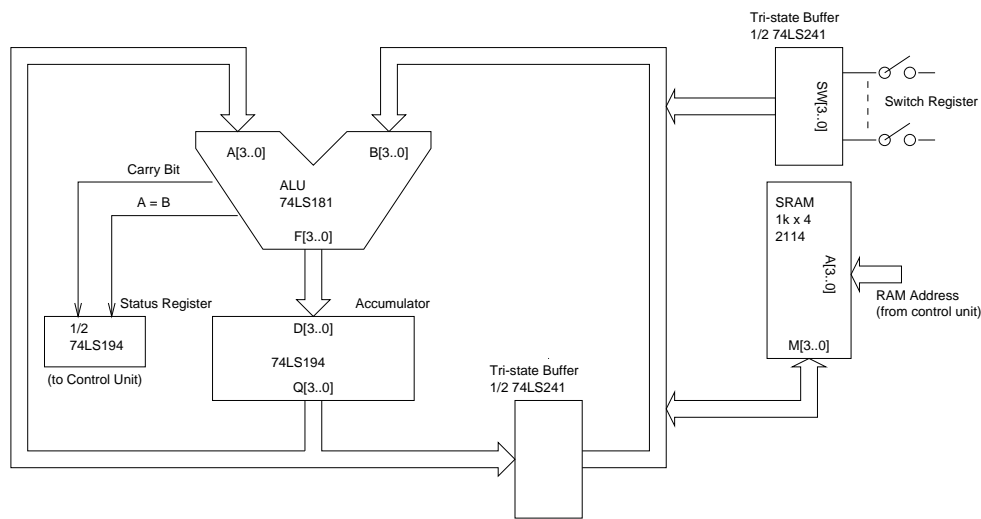


Figure 6: ESP Data Path

5 The Control Unit

The ESP control unit orchestrates the control signals of the various elements in the data path so that the operations specified by the program instructions take place correctly. Figure 9 shows the main features of the ESP control circuits.

The control unit starts at the left edge with the Program Counter (PC). The PC increments through the program instructions and can be parallel loaded to execute a JMP instruction. The program counter receives the system clock but is incremented only when a control signal allows. Up to four clock pulses may occur each time the program counter increments to the next instruction.

The output of the Program Counter connects to 4 address lines of the Program EPROM, which contains the actual computer program. A total of 4 different computer programs can be burned into the EPROM and one of them selected by two Program Select switches.

The Microprogram Memory is contained in two, 8-bit wide EPROMs. This section of the control unit is a state machine in which the current state is latched into the 2-bit State Register. The current state is fed back as address lines A0 and A1 into the state memory and they select a memory location that contains the next state of the machine. Since two bits used to specify the current and next state, the state machine can cycle through a total of four possible states. These four states are the states of the machine or *state sequence* as it executes of some instruction (such as Load the Accumulator) of the computer program³.

The 4-bit operation code of the currently executing program instruction selects one of 16 possible starting addresses in the microprogram EPROM, each of which contains a four-state sequence for that instruction.

As the control system cycles through the 4-state sequence for some instruction, it is accessing locations in microcode memory that contain some 13 bits that control other parts of the machine. Some examples of these signals are:

³In practice, many of the state sequences are only one microinstruction long, and the three other microinstruction states are not used.

Function Select S[3..0]	Logical Operation M=0	Arithmetic Operation M=1
0	A	\overline{A}
1		$A + B$
2		
3		1111
4		
5	\overline{B}	
6	$A \oplus B$	
7		
8		
9		A plus B
A	B	
B	$A * B$	
C		
D		
E	$A + B$	
F	A	A minus 1

Figure 7: '181 ALU Operations

Function Select S[1,0]	Operation
0	Hold (no op)
1	Shift Left
2	Shift Right
3	Parallel Load

Figure 8: '194 Accumulator Operations

ALU Function 5 bits that specify the function of the Arithmetic Logic Unit according to figure 7.

Accumulator Function 2 bits that specify the function of the Accumulator Register as shown in figure 9.

Read-Write 1 bit that specifies a read or write cycle for the SRAM.

Bus Enables 2 bits that enable and disable the tri-state buffers in the datapath.

Program Counter 2 bits that control the Program Counter, choosing between HOLD, INCREMENT and PARALLEL LOAD operations.

As the machine cycles through its 4-state sequence, it generates signals at 13 outputs from the microprogram EPROMs that actuate different sections of the machine.

What we have described so far conveys the basic operation of the control unit, but there are some additional details, not shown in figure 9.

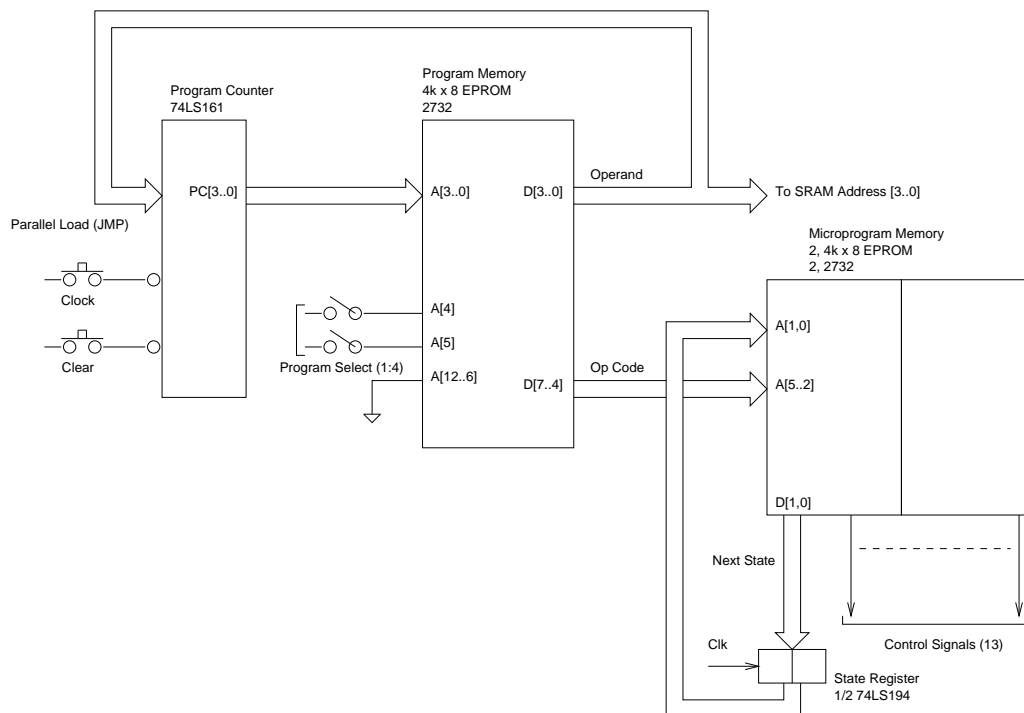


Figure 9: ESP Control Circuit

Extended Decoding

The previous description assumed that instructions could be distinguished by their 4-bit operation code, in bits [7..4] of the instruction. This is true for some instructions, but others, such as `ADDA S` and `SUBA S` must be distinguished by an additional bit in position 0 of the instruction word. This bit must participate in the selection of a state sequence for these instructions, so it is connected to A6 of the microcode memory.

Conditional Branching

When a conditional jump instruction (`JCS N`, `JEQ N` or `JMI N`) is being executed, the machine must execute one of two possible sequences of states depending on the condition input. Consequently, the condition must also be used to select a state sequence and so is used as A7 of the microcode memory.

Combinational Logic

A certain amount of combinational logic is required to implement some signals. For example, the logic equation for the A7 input into microcode memory (true when a branch is to take place, false otherwise) requires combinational logic like:

$$A7 = (JCS_OP * C) + (JEQ_OP * A=B) + (JMI_OP * Q3)$$

where JCS_OP is the op code for the JCS instruction, and so on.

In words, the A7 signal is true whenever the JCS opcode and carry bit are asserted, or the JEQ opcode and equal bit, or the JMP opcode and most significant bit of the accumulator. This and other similar equations are implemented in a 16V8 GAL device.

6 Schematic Diagrams

The complete schematic of the ESP datapath is shown in figure 10. The schematic of the control unit is shown in figure 11.

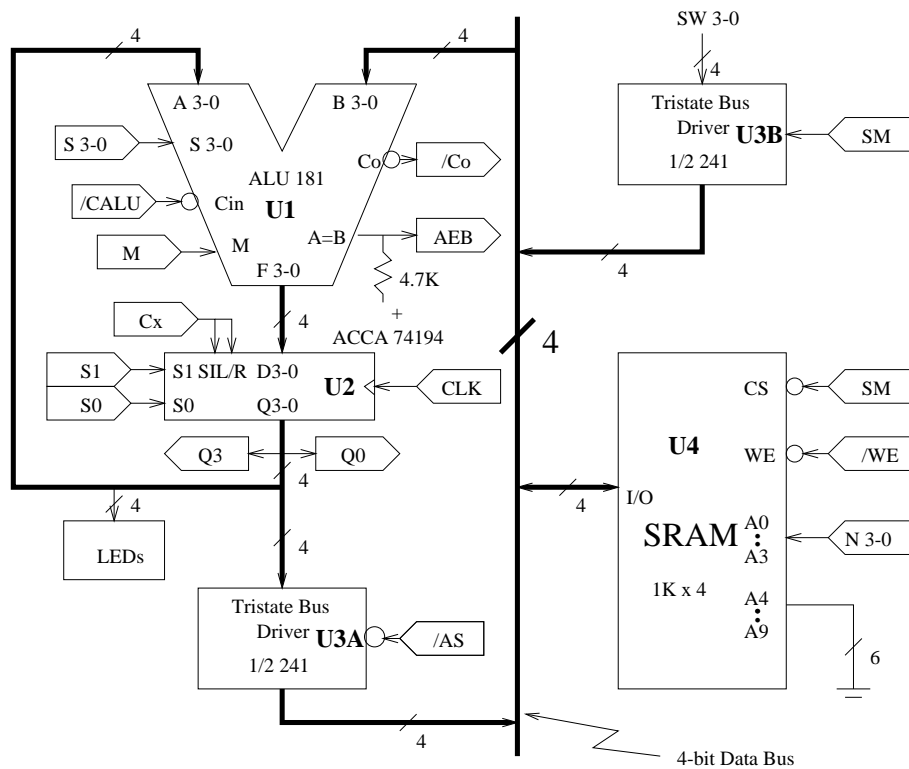
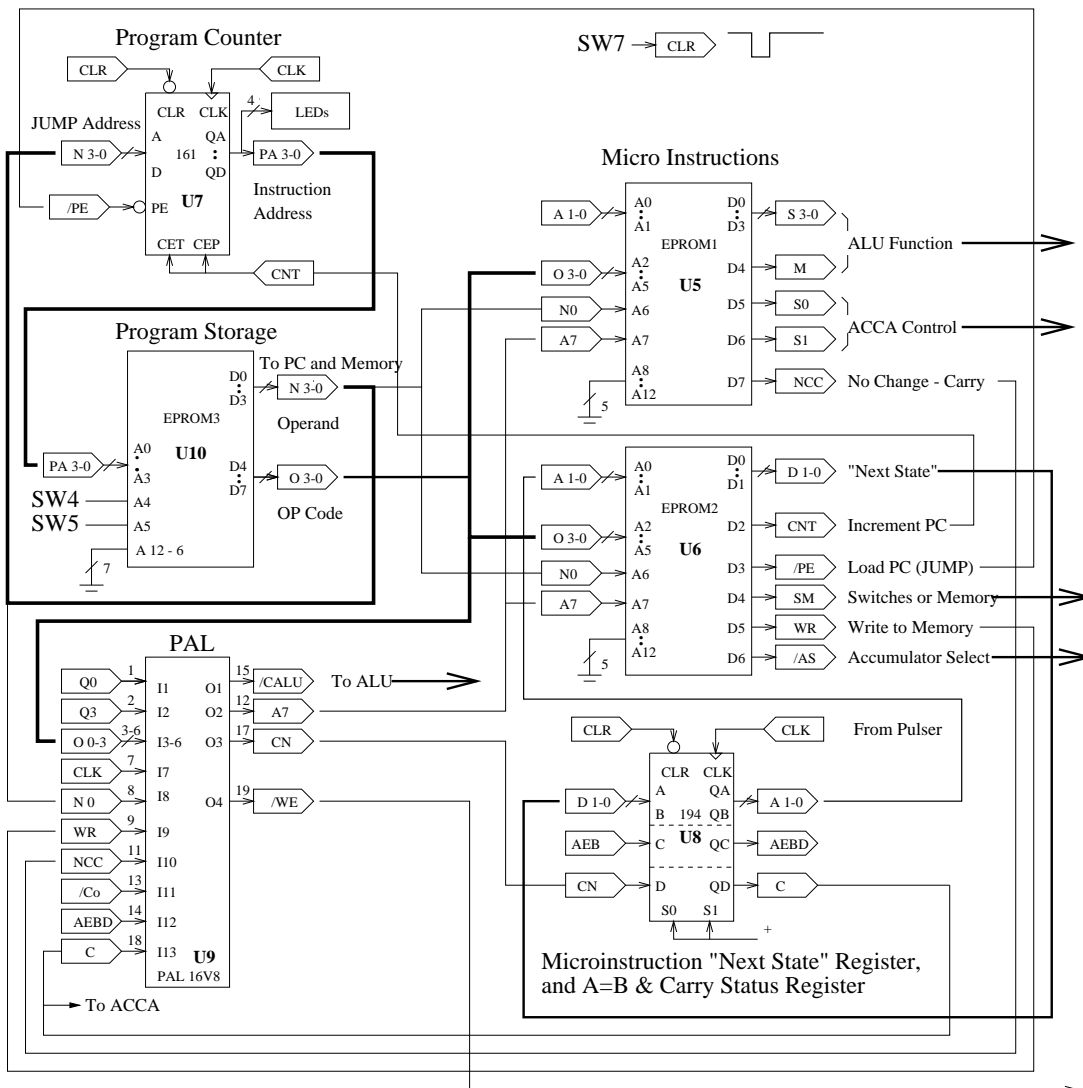


Figure 10: ESP Data Path: Schematic



I/O SOCKET

Sw0	9	8	PA0
Sw1	10	7	PA1
Sw2	11	6	PA2
Sw3	12	5	PA3
PA4 Sw4	13	4	Q0
PA5 Sw5	14	3	Q1
CLK	15	2	Q2
/CLR Sw7	16	1	Q3

PA5 PA4 EPROM3 PROGRAM STARTING ADDRESS

0	0	0000
0	1	0010
1	0	0020
1	1	0030

Figure 11: ESP Control Unit: Schematic

The IO SOCKET shown in figure 11 connects switch and LED signals from the Input-Output box seen in the photograph of figure 2.

Signal Names

A table of the signal names, in alphabetical order, is shown in figure 12.

Signal	Function
/AS	Accumulator Select: connect Accumulator to main bus
/CALU	Carry In bit to the ALU
/Co	ALU Carry Out
/PE	Program counter, Parallel load enable
/WE	Write Enable
A7	Microinstruction address line 7 select
AEB	A equals B
AEBD	A equals B, Delayed (A=B status register output)
A[1..0]	Microinstruction State Select lines
C	Carry signal from status register
CLK	System clock
CLR	Clear the program counter
CN	Intermediate CARRY signal
CNT	Program counter, Count Enable
Cx	Carry In bit to the accumulator
M	ALU Function control line
N0	LSBit of Instruction Operand, used to select microcode address
NCC	No Change Carry: hold contents of carry bit register
N[3..0]	Instruction operand field
N[3..0]	RAM address lines
O[3..0]	Instruction operation code field
PA[3..0]	Program Address lines
Q0	LSBit output from the accumulator
Q3	MSBit output from the accumulator
S1,S0	Accumulator register function select lines
SM	Select Switches or Memory
SW4,SW5	Program select switch lines
S[3..0]	ALU Function control lines
WR	Write to memory

Figure 12: ESP Signals

GAL Equations

The GAL (or PAL) device implements the following combinational equations:

- WE = WR*/CLK; Develops correct timing for SRAM write.
- $CN = O_3 \overline{O_2} \overline{O_1} \overline{O_0} N_0$ {CLC, SEC}
 $+ O_3 \overline{O_2} O_1 \overline{O_0} * (N_0 Q_3 + \overline{N_0} Q_0)$ {ROLA, RORA}
 $+ \overline{O_3} \overline{O_2} \overline{O_1} C_0$ {ADDA N/SUBA N}
 $+ \overline{O_3} O_2 O_1 \overline{O_0} C_0$ {ADDA S/SUBA S}
 $+ NCC * C$ {NO CHANGE CARRY}

CN = C+ = active HIGH next carry bit. The carry bit is inhibited from changing by the NCC bit from EPROM 1 (D_7) unless the instruction requires it. Similarly ACCA should not be changed unless directed to.

- C := CN (C+) and AEBD := AEB to be implemented using the 74194 due to pin-out limitations of the PAL.
- $CALU = O_3 \overline{O_2} \overline{O_1} O_0 N_0$ {DECA, INCA}
 $+ \overline{O_3} \overline{O_2} \overline{O_1} C$ {ADDA N/SUBA N}
 $+ \overline{O_3} O_2 O_1 \overline{O_0} C$ {ADDA S/SUBA S}

Note: CALU will be zero for JEQ N, LDAA N and INPA by default.

PAL output pin defined as /CALU to achieve desired active LOW.

- $A7 = O_3 O_2 \overline{O_1} \overline{O_0} * C$ {JCS N}
 $+ O_3 O_2 \overline{O_1} O_0 * AEBD$ {JEQ N}
 $+ O_3 O_2 O_1 \overline{O_0} * Q_3$ {JMI N}

7 Microcode

The complete microcode listing is shown in figure 13 on page 14 and figure 14 on page 15. Most machine instructions require only one line of microcode, but some (STSW N, JEQ N) require two lines.

The microcode memory uses control lines JMP and N0 as address input A7 and A6 into the microcode memory. Consequently, if there is some possibility of these lines becoming asserted during a given machine instruction, the microcode must be duplicated at several addresses in memory. For example, it is possible for all the instructions that N0 may be come active, so all the instruction microcode entries must appear at two addresses in microcode memory, corresponding to N0=0 and N0=1. Consequently, the complete microcode listing requires the two tables of figures 13 and 14.

This use of condition lines to select microcode addresses rapidly becomes unwieldy when many conditions affect the execution of the microcode sequence, because the microcode must be duplicated at so many locations in the microcode memory. In those cases, it is better to select the desired condition using additional control outputs from the microcode memory, and have that one, selected condition signal actuate one microcode address line.

PPM INSTRUCTION MICROCODE (PG.1 OF 2)

JMP=0, N0=0 *Indicates a 5-bit instruction.

	EPROM 3 OUTPUT						EPROM 1 OUTPUT						EPROM 2 OUTPUT														
	EPROMs 1&2 Address Lines						PAL	ACCA		ALU181				NC	Data Path			PC 161		EPROM1&2							
	JMP N0	OP Code			M.C.		NCC	S1	S0	M	S3	S2	S1		S0	/AS	WR	SM	/PE	CNT	A1+	A0+					
	A7	A6	A5	A4	A3	A2	A1	A0	HEX	D7	D6	D5	D4	D3	D2	D1	D0	HEX	D7	D6	D5	D4	D3	D2	D1	D0	HEX
ADDA N	0	0	0	0	0	0	0	0	00	0	1	1	0	1	0	0	1	69	X	1	0	0	1	1	0	0	CC
SUBA N	0	0	0	0	0	1	0	0	04	0	1	1	0	0	1	1	0	66	X	1	0	0	1	1	0	0	CC
INPA	0	0	0	0	1	0	0	0	08	1	1	1	1	1	0	1	0	FA	X	1	X	1	1	1	0	0	FC
LDAA N	0	0	0	0	1	1	0	0	0C	1	1	1	1	1	0	1	0	FA	X	1	0	0	1	1	0	0	CC
STAA N	0	0	0	1	0	0	0	0	10	1	0	0	X	X	X	X	X	9F	X	0	1	0	1	1	0	0	AC
JMP N	0	0	0	1	0	1	0	0	14	1	0	0	X	X	X	X	X	9F	X	1	0	X	0	0	0	0	D0
ADDA S	0	0	0	1	1	0	0	0	18	0	1	1	0	1	0	0	1	69	X	1	X	1	1	1	0	0	FC
ANDA N	0	0	0	1	1	1	0	0	1C	1	1	1	1	1	0	1	1	FB	X	1	0	0	1	1	0	0	CC
CLC	0	0	1	0	0	0	0	0	20	0	0	0	1	1	1	1	1	1F	X	1	0	X	1	1	0	0	DC
DECA	0	0	1	0	0	1	0	0	24	1	1	1	0	1	1	1	1	EF	X	1	0	X	1	1	0	0	DC
RORA	0	0	1	0	1	0	0	0	28	0	1	0	X	X	X	X	X	5F	X	1	0	X	1	1	0	0	DC
STSW N	0	0	1	0	1	1	0	0	2C	1	1	1	1	1	0	1	0	FA	X	1	X	1	1	1	0	0	F9
	0	0	1	0	1	1	0	1	2D	1	0	0	X	X	X	X	X	9F	X	0	1	0	1	1	0	0	AC
JCS N	0	0	1	1	0	0	0	0	30	1	0	0	X	X	X	X	X	9F	X	1	0	X	1	1	0	0	DC
JEQ N	0	0	1	1	0	1	0	0	34	1	0	0	0	0	1	1	0	86	X	1	X	1	1	0	0	1	F9
	0	0	1	1	0	1	0	1	35	1	0	0	X	X	X	X	X	9F	X	1	0	X	1	1	0	0	DC
JMI N	0	0	1	1	1	0	0	0	38	1	0	0	1	1	1	1	1	9F	X	1	0	X	1	1	0	0	DC
CLRA	0	0	1	1	1	1	0	0	3C	1	1	1	1	0	0	1	1	F3	X	1	0	X	1	1	0	0	DC
JMP=1, N0=0																											
JCS N	1	0	1	1	0	0	0	0	B0	1	0	0	X	X	X	X	X	9F	X	1	0	X	0	0	0	0	D0
JEQ N	1	0	1	1	0	1	0	0	B4	1	0	0	0	0	1	1	0	86	X	1	X	1	1	0	0	1	F9
	1	0	1	1	0	1	0	1	B5	1	0	0	X	X	X	X	X	9F	X	1	0	X	0	0	0	0	D0
JMI N	1	0	1	1	1	0	0	0	B8	1	0	0	X	X	X	X	X	9F	X	1	0	X	0	0	0	0	D0

Figure 13: ESP Microcode: 1 of 2

PPM INSTRUCTION MICROCODE (PG.2 OF 2)

JMP=0, N0=1 *Indicates a 5-bit instruction.

	EPROM 3 OUTPUT							EPROM 1 OUTPUT							EPROM 2 OUTPUT												
	EPROMs 1&2 Address Lines							PAL	ACCA			ALU181				NC	Data Path			PC 161	EPROM1&2						
	JMP	N0	OP Code				M.C.	NCC	S1	S0	M	S3	S2	S1	S0	/AS	WR	SM	/PE	CNT	A1+	A0+					
	A7	A6	A5	A4	A3	A2	A1	A0	HEX	D7	D6	D5	D4	D3	D2	D1	D0	HEX	D7	D6	D5	D4	D3	D2	D1	D0	HEX
ADDA N	0	1	0	0	0	0	0	0	40	0	1	1	0	1	0	0	1	69	X	1	0	0	1	1	0	0	CC
SUBA N	0	1	0	0	0	1	0	0	44	0	1	1	0	0	1	1	0	66	X	1	0	0	1	1	0	0	CC
INPA	0	1	0	0	1	0	0	0	48	1	1	1	1	1	0	1	0	FA	X	1	X	1	1	1	0	0	FC
LDAA N	0	1	0	0	1	1	0	0	4C	1	1	1	1	1	0	1	0	FA	X	1	0	0	1	1	0	0	CC
STAA N	0	1	0	1	0	0	0	0	50	1	0	0	X	X	X	X	X	9F	X	0	1	0	1	1	0	0	AC
JMP N	0	1	0	1	0	1	0	0	54	1	0	0	X	X	X	X	X	9F	X	1	0	X	0	0	0	0	D0
*SUBA S	0	1	0	1	1	0	0	0	58	0	1	1	0	0	1	1	0	66	X	1	X	1	1	1	0	0	FC
ANDA N	0	1	0	1	1	1	0	0	5C	1	1	1	1	1	0	1	1	FB	X	1	0	0	1	1	0	0	CC
*SEC	0	1	1	0	0	0	0	0	60	0	0	0	1	1	1	1	1	1F	X	1	0	X	1	1	0	0	DC
*INCA	0	1	1	0	0	1	0	0	64	1	1	1	0	0	0	0	0	E0	X	1	0	X	1	1	0	0	DC
*ROLA	0	1	1	0	1	0	0	0	68	0	1	0	X	X	X	X	X	3F	X	1	0	X	1	1	0	0	DC
STSW N	0	1	1	0	1	1	0	0	6C	1	1	1	1	1	0	1	0	FA	X	1	X	1	1	1	0	0	F9
	0	1	1	0	1	1	0	1	6D	1	0	0	X	X	X	X	X	9F	X	0	1	0	1	1	0	0	AC
JCS N	0	1	1	1	0	0	0	0	70	1	0	0	X	X	X	X	X	9F	X	1	0	X	1	1	0	0	DC
JEQ N	0	1	1	1	0	1	0	0	74	1	0	0	0	0	1	1	0	86	X	1	X	1	1	0	0	1	F9
	0	1	1	1	0	1	0	1	75	1	0	0	X	X	X	X	X	9F	X	1	0	X	1	1	0	0	DC
JMI N	0	1	1	1	1	0	0	0	78	1	0	0	1	1	1	1	1	9F	X	1	0	X	1	1	0	0	DC
*SETA	0	1	1	1	1	1	0	0	7C	1	1	1	1	1	1	0	0	FC	X	1	0	X	1	1	0	0	DC
JMP=1, N0=1																											
JCS N	1	1	1	1	0	0	0	0	F0	1	0	0	X	X	X	X	X	9F	X	1	0	X	0	0	0	0	D0
JEQ N	1	1	1	1	0	1	0	0	F4	1	0	0	0	0	1	1	0	86	X	1	X	1	1	0	0	1	F9
	1	1	1	1	0	1	0	1	F5	1	0	0	X	X	X	X	X	9F	X	1	0	X	0	0	0	0	D0
JMI N	1	0	1	1	1	0	0	0	F8	1	0	0	X	X	X	X	X	9F	X	1	0	X	0	0	0	0	D0

Figure 14: ESP Microcode: 2 of 2

8 Acknowledgements

Doug Hawkes and Augustine Lee filled in many details of the design, developed lab exercises and created documentation for the project which has been used in this paper. Jim Koch and the technical staff of the EE Department constructed the units and maintain them.

References

- [1] Building a Computer from Scratch
Hilary D. Jones
Byte, November 1977
pp 80-92
- [2] Able to leap μ P's in a single bound – the ABDAT Processor
Arnis G. Butulis and Dan Torbeck
EDN, February 20, 1975
pp 45-49
- [3] Two-bit microcomputer for educational use
Ryo-II Kang and Katsufusa Shono
Microprocessors and Microsystems, Vol 15, No. 6, July/August 1991
pp 299-304
- [4] Designing microprocessors with standard-logic devices
Robert Jaeger
Electronics Magazine
Part 1: (issue), pp 93-98
Part 2: (issue), pp 99-104
- [5] A One-Instruction Computer: Architecture, Implementation, Experiments
Christopher R. Carroll
1992 ASEE Annual Conference Proceedings
pp 1683-1686
- [6] Design of Microprogrammable Systems
Signetics Memory Systems
Technical Note SMS0052AN
December, 1970
- [7] 5701/6701 4-Bit Expandable Bipolar Microcontroller
Clive Ghest
Monolithic Memories Incorporated
July 1976