Ryerson Polytechnic University
Department of Electrical Engineering
COE328 – Digital Systems

Lab 7 - Programmable Processor Module - PPM (2 Weeks)

FORMAL REPORT - 30 Marks **Due Date: Week 13**

1 Objectives:

- To understand the functions of the different units (e.g Control Unit, Memory, Data Path, ALU, PC,..) that are used in the design of the simple processor.
- To implement the instruction sets of the processor by deriving the microcode of the processor.
- To write a simple program using the instruction set of the processor.

2 Introduction

The processor has the following features:

- data word length - 4 bits
- instruction length - 8 bits (4 bits OP Code, 4 bits Operand)
- micro instruction length - 16 bits
- 32 logical and arithmetic functions available
- working storage SRAM (16 by 4 is used)
- program storage: EPROM3
- micro code storage: EPROMs 1 & 2
- programming language and microcode for each instruction
- capable of doing decision branching (conditional jump)

3 Data Path Unit

The data path unit (see Figure refdpunit) is responsible for the manipulation and movement of data. It consists of the following components:

3.1 Data Input Unit

The data is entered into the data path using switches (SW3-0) when the control unit makes the **SM** signal active.

3.2 Output Unit

The register ACCA is used to store the result from the ALU and has been connected to 4 LEDs on the lab module to indicate its value.

3.3 Data Path Unit - Logic Diagram

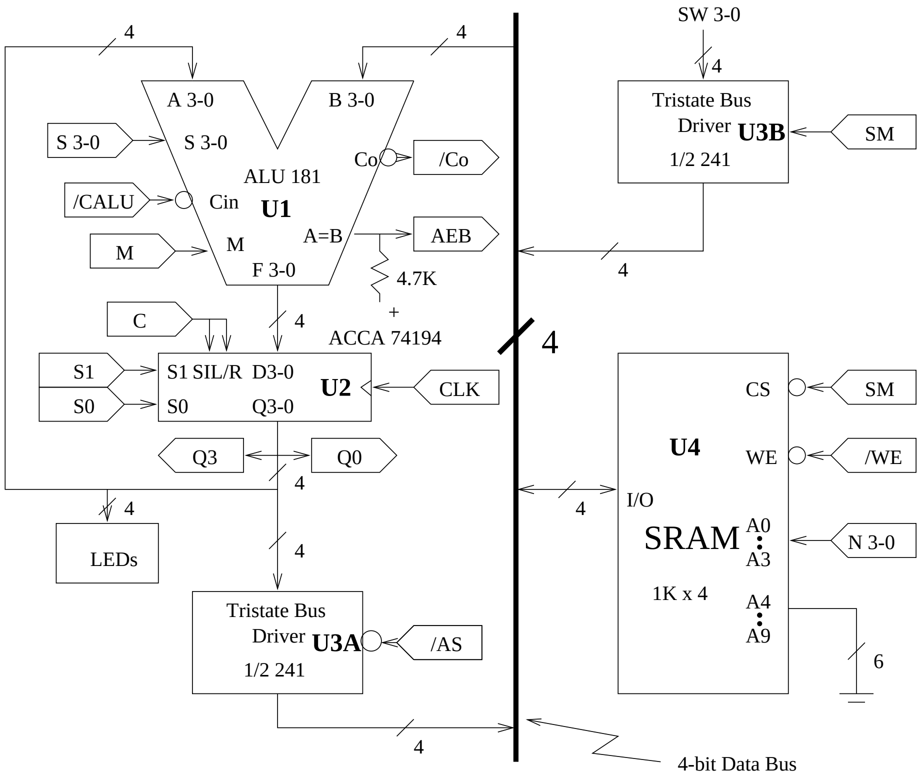


Figure 1: Data Path Unit

3.4 ALU Unit

- The arithmetic and logic operations between the operands A and B take place on the ALU unit.
- The B operand comes from either the input unit (tristate bus driver) or one of the SRAM locations.
- The A Operand comes from the ACCA (74194).
- The Control Unit (EPROM1) sets the value of M and S_{3-0} to select the required ALU function (Refer to ALU data sheet for all operations). The carry bit C is registered and is derived from different sources depending on which instruction is being executed. It comes from operand bit N_0 for instructions CLC and SEC, from the ALU output /Co for the ADDA and SUBA instructions and from the ACCA outputs Q_3 and Q_0 for the ROLA and RORA instructions respectively. OP Code bits $D_7 - D_4$ ($O_3 - O_0$) from EPROM3 are used to determine which instruction is being executed. The carry into the ALU is designated /CALU. It is derived directly from the stored carry bit C when the four add and subtract instructions are executed or from the operand bit N_0 when the ALU is used to increment and decrement the accumulator (INCA, DECA).
- The output from the ALU is stored in the ACCA unit (74194).
- The AEB output goes HIGH when B is subtracted from A and the result is zero. **Note**, however, that this was designed for the active LOW data mode where zero looks like 1111. Since we are using the active HIGH data mode we must subtract an additional ONE to achieve 1111 when the operands are equal. (See the subtract instruction, the carry into the ALU must be zero to achieve this). AEB is also registered as AEBD to avoid a timing race in the A_7 equation. Thus the JEQ N instruction requires two steps to execute, one to establish equality and one to act on it.

3.5 Memory Unit

The SRAM is used to store data (up to 16 addresses, 4 bits each).

- The READ operation is done by Control Unit (EPROM2) as the following:
  - READ address operand is taken from EPROM3, bits $D_3 - D_0$ ($N_3 - N_0$).
  - make the WR signal not active. (WR is a microinstruction bit from EPROM2)
  - make the SM signal = 0 to access memory and disable the input half of the bus driver.
  - the data is supplied to the ALU input B.
- The Write Operation is done as the following:
  - WRITE address operand is taken from EPROM3, bits D3-0 (N3-0).
  - make the SM signal = 0 to access memory and disable the input half of the bus driver.
  - assert the WR signal and the /AS signal (accumulator output is made active).
  - the data stored in ACCA is written to the addressed location.
  - the write operation is controlled by the /WE signal derived from the following equation:

$$/WE = /(WR * /CLK)$$

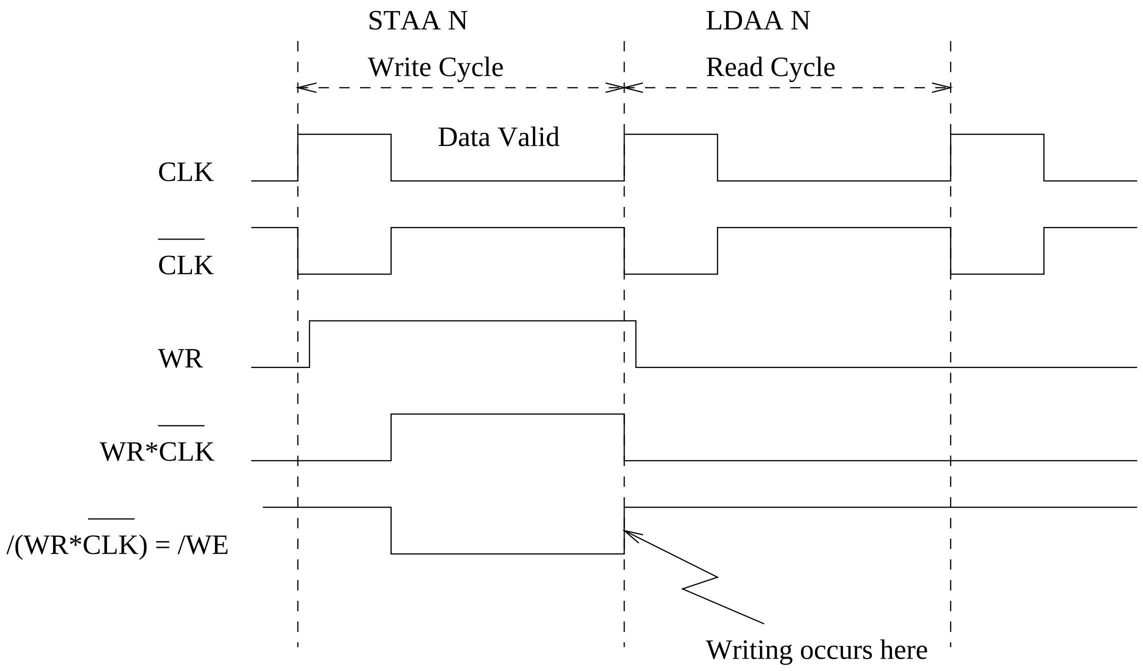
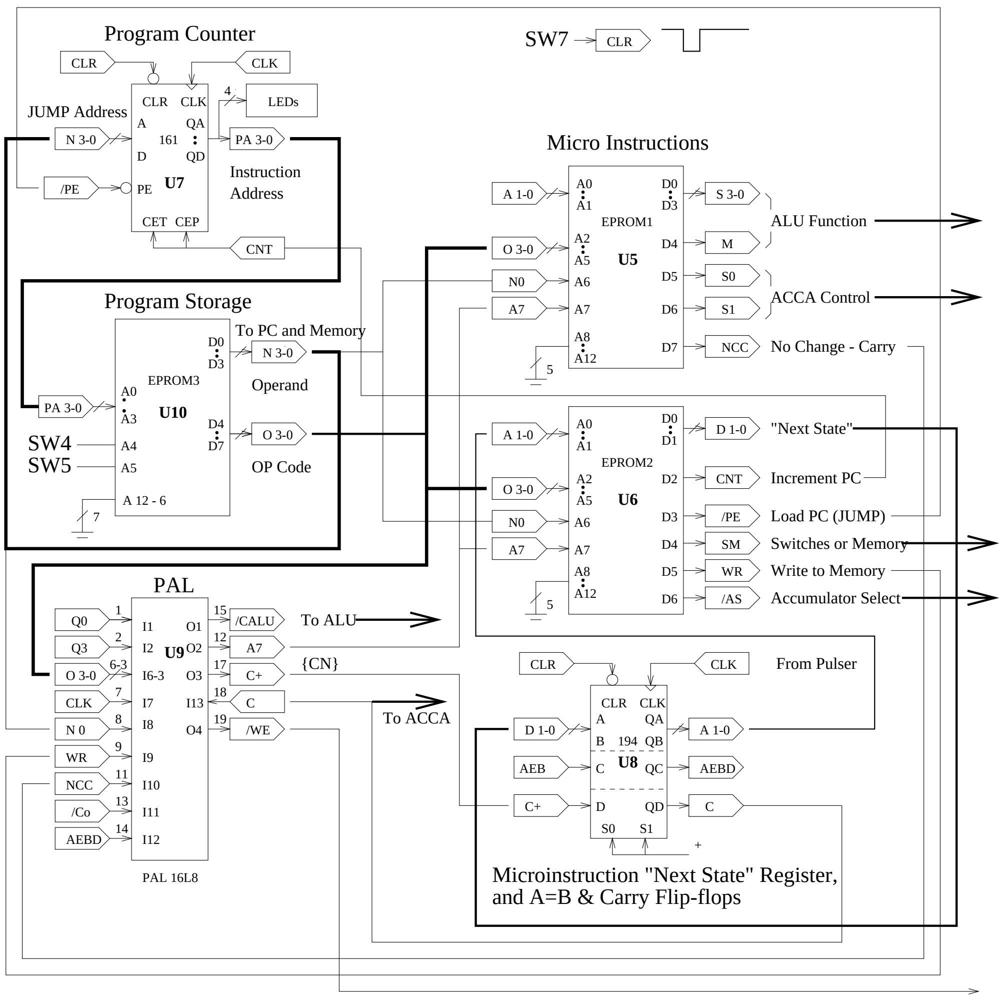
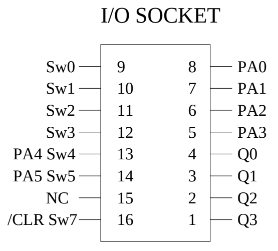


Figure 2: Read/Write Control Waveforms

4 Control Unit

The Control Unit provides all the necessary signals required by the Data Path Unit.





EPROM3 PROGRAM

PA5	PA4	STARTING ADDRESS
0	0	0000
0	1	0010
1	0	0020
1	1	0030

Figure 3: Control Unit

4.1 Program Storage

The program is stored in EPROM3 in a sequence of 8-bit instructions (up to 16 per program allowed).

4.2 Instructions

EPROM3 is used for storing the program instructions. Datalines $D_7 - D_0$ from this chip are treated as a 4-bit OP Code ($O_3 - O_0$) followed by a 4-bit Operand ($N_3 - N_0$). The following are the different instruction codes, both Object and Source, and their description for this processor:

Object Code   Source Code

- **0 0 0 0** $N_3N_2N_1N_0$, ADDA N : ACCA := ACCA + (N) + C.
- **0 0 0 1** $N_3N_2N_1N_0$, SUBA N : ACCA := ACCA - (N) - /C.   **Note:** /C = Borrow Bit
- **0 0 1 0 X X X X**, INPA : load accumulator with input data from the switches, ACCA := Sw
- **0 0 1 1** $N_3N_2N_1N_0$, LDAA N : load accumulator with the contents of memory location N.
- **0 1 0 0** $N_3N_2N_1N_0$, STAA N : store ACCA into memory location N.
- **0 1 0 1** $N_3N_2N_1N_0$, JMP N : jump to program address N.
- **0 1 1 0 X X X 0**, ADDA S : ACCA := ACCA + Sw + C.
- **0 1 1 0 X X X 1**, SUBA S : ACCA := ACCA - Sw - /C.
- **0 1 1 1** $N_3N_2N_1N_0$, ANDA N : AND the contents of N with ACCA and store in ACCA.
- **1 0 0 0 X X X 0**, CLC : clear carry bit (C = 0).
- **1 0 0 0 X X X 1**, SEC : set carry bit (C = 1).
- **1 0 0 1 X X X 0**, DECA : decrement ACCA.
- **1 0 0 1 X X X 1**, INCA : increment ACCA.
- **1 0 1 0 X X X 0**, RORA : rotate ACCA right (Q_3 := C and C := Q_0). (Note: In your circuit Q_3 is on the left whereas in the 74194 specs, Q_3 is considered to be the rightmost bit).
- **1 0 1 0 X X X 1**, ROLA : rotate ACCA left (Q_0 := C and C := Q_3).
- **1 0 1 1** $N_3N_2N_1N_0$, STSW N : (N) := Sw. Uses accumulator as interim storage.
- **1 1 0 0** $N_3N_2N_1N_0$, JCS N : jump to program address N if carry set.
- **1 1 0 1** $N_3N_2N_1N_0$, JEQ N: jump to program address N if A = B.

  Note: ALU must be in subtract mode and CALU clear (borrow = 1) to make F = 1111 (AEB HIGH) when A = B. For these instructions, A = ACCA and B = switches (input).
- **1 1 1 0** $N_3N_2N_1N_0$, JMI N : jump to program address N if ACCA -ve (Q_3 = 1).
- **1 1 1 1 X X X 0**, CLRA : ACCA := 0000.
- **1 1 1 1 X X X 1**, SETA : ACCA := 1111.

Implementation of the Conditional Jump Instructions:

- Every conditional jump instruction has two different sets of microcode to implement it. When A_7 =0 (microcode pages 0 0 or 0 1) when the jump does not occur and when A_7 = 1 (microcode pages 1 0 or 1 1) when the jump conditions are met.

4.3 Program Counter (PC)

The PC uses the 74161 counter. It points to the the next instruction to be executed (stored in EPROM3 via address lines $PA_3 - PA_0$) and is incremented by the CNT signal (D_2 EPROM2). When a Jump is executed, the address of the next instruction (EPROM3, bits $D_3 - D_0$ ($N_3 - N_0$)) is loaded into the PC.

- ($\overline{PE}$, CNT) **Control of 74161 Program Counter:**

CNT = CET and CEP (serial and parallel count enables)

/PE	CNT	
1	0	No Change
1	1	Count
0	X	Load

4.4 Microinstructions

Each program instruction can take up to four lines of microinstructions to implement it. We are, however, only using 1 to 2 of these at the moment, leaving 2 to 3 lines of microinstruction blank, (hex FF).

The microinstructions consist of 15 control signals which are stored in EPROMs 1 & 2. A group of up to 4 microinstructions representing a given program instruction are selected from memory by using the program instruction Op-Code (bits $O_3 - O_0$) as address lines $A_5 - A_2$ to EPROMs 1 & 2. The specific microinstruction (one of the 4 possible) is selected by address lines $A_1 - A_0$ derived from the microinstruction next state register.

In order to increase the number of instructions available to us, we use operand bit N_0 sometimes as a fifth Op-Code bit (see CLC and SEC). To distinguish these instructions from one another we use N_0 as an additional address line (A_6) to EPROMs 1 & 2. This requires microinstructions for Op-Codes not using N_0 in this way to be entered into EPROMs 1 & 2 twice, for $N_0(A_6) = 0$ and for $N_0(A_6) = 1$.

The JUMP instruction microcodes also require additional locations in memory to distinguish those microcodes that cause a jump from those that do not cause a jump. Here we use address line A_7 as described earlier. We have therefore created four pages in microprogram memory as distinguished by the states of A_7 and A_6. These can be known as page 0 0, (CLC, DECA, RORA and CLRA - no jump), page 0 1, (SEC, INCA, ROLA and SETA - no jump), page 1 0, (CLC, DECA, RORA and CLRA - jump), and finally page 1 1, (SEC, INCA, ROLA and SETA - jump).

4.5 Microinstruction Next State Register (State Counter)

The Microinstruction next state register uses a 74194 to hold the address of the next microinstruction. Address lines $A_1 - A_0$ are used to access the current microinstruction out of the group of 4 that are stored in EPROMs 1 & 2. $A_1 - A_0$ are usually equal to 0 0 since most instructions have only one line of microcode. You will notice, however, that in the case of JEQ N which has been given to you on page 9, the first line of microcode leaves A_1+, A_0+ equal to 0 1 so that the second line of microcode for this instruction will be fetched. Here the Control Unit (EPROM2, $D_1 - D_0$) sets the value of the address ($A_1 - A_0$) of the next microinstruction to be fetched (via the 74194 next state register). The CLR signal (SW7) manually forces both the microinstruction program counter and the program counter to go to zero which takes us to the first microinstruction of the first instruction of the program.

This register also has the job of storing the carry bit C and the AEBD bit.

4.6 PAL

The PAL is used to generate additional control signals which are required to operate the system. The output signals are derived from the following PAL Equations:

- Note: The following equations must be put into a suitable format for PALASM.
- WE = WR*/CLK; Develops correct timing for SRAM write. (See waveforms page 3).

- CN $= O_3\overline{O_2}\,\overline{O_1}\,\overline{O_0}N_0$ {CLC, SEC}
  $+ O_3\overline{O_2}O_1\overline{O_0} * (N_0Q_3 + \overline{N_0}Q_0)$ {ROLA, RORA}
  $+ \overline{O_3}\,\overline{O_2}\,\overline{O_1}Co$ {ADDA N/SUBA N}
  $+ \overline{O_3}O_2O_1\overline{O_0}Co$ {ADDA S/SUBA S}
  $+ NCC * C$ {NO CHANGE CARRY}

  CN = C+ = active HIGH next carry bit. (Note: PALASM does not accept a "+" sign as a valid label character). 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 $= O_3\overline{O_2}\,\overline{O_1}O_0N_0$ {DECA, INCA}
  $+ \overline{O_3}\,\overline{O_2}\,\overline{O_1}C$ {ADDA N/SUBA N}
  $+ \overline{O_3}O_2O_1\overline{O_0}C$ {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 $= O_3O_2\overline{O_1}\,\overline{O_0} * C$ {JCS N}
  $+ O_3O_2\overline{O_1}O_0 * AEBD$ {JEQ N}
  $+ O_3O_2O_1\overline{O_0} * Q_3$ {JMI N}

5 Pre-lab preparation:

Each person will be asked to write a program in SOURCE code for a specific task which must be converted to HEX OBJECT code and loaded into EPROM3, also you will be asked to run the three example programs in Appendix A **"Programming the Processor"**.

- Write the microinstructions for the OP Codes that are not completed in Appendix B **"Instruction Microcode"**. **Note:** All instructions with the exception of STSW N and JEQ N require only one line of microcode, **therefore you may ignore any lines you do not require.**

- Convert the microinstructions to data for the EPROMs 1 & 2 and the programs to data for EPROM3.

  > **Note:** Due to the limitation of the 4-bit Program Counter in our current design, the maximum number of program memory (EPROM3) locations directly accessible to a given program is 16, starting at 0000(hex). The provision of external switches (PA5 and PA4) allows us to map manually three other 16-line-pages (starting at 0010, 0020, and 0030) into the same program space (0000-000F). Thus, we can store up to four separate program segments in the same EPROM before erasing. If your EPROM is a 27C256 or 27C512 (instead of 27C64), then the starting addresses of the programs should be C000, C010, C020 and C030 respectively.

  Warning: When adding a program, be sure to read the eprom contents before inputting the new file from the disc. This ensures the unused eprom addresses will be blank for the next addition.

- Write the PAL logic equations assigning quantities to the appropriate pins as required by the given circuit diagram.

6 Procedure:

A formal report is required for this lab.

- Program the 3 EPROMS and the PAL and place all chips in the circuit board supplied.
- Demonstrate the circuit to your instructor executing the required programs with documented examples indicating the expected register contents (PC, ACCA, relevant memory locations and C) after the execution of each instruction.

7 Formal Report Requirements:

- Lab Sheets should be handed in with the formal report. These lab sheets are to be signed and dated at relevant places by the lab instructor during demonstration.
- The four μ Code tables are to be filled out in full.
- The two files of μ Code are to be printed out. Eprom1.asc and Eprom2.asc.
- Your assigned program, source and object code with comments for each line.
- File for EPROM3 (Eprom3.asc) printed out (Includes all 4 programs.)
- Numerical examples for the running of all 4 programs, showing Program Counter, ACCA contents and C and relevant memory contents after each step and for all cycles where applicable.
- Printout of the PAL file pal.pds
- All printouts to have name of student.

A Programming the Processor

To start, we have to understand the architecture of our processor. (This is also the starting point of our microcode design.) The architecture of a processor is defined by both the way the hardware signals are routed, and the specific instruction set available to the user. In our case, the hardware connections are fixed by the existing printed circuit board design, and the instruction set is chosen arbitrarily.

The designs of the hardware and the instruction set are interrelated: We start with the concept of a simple processor and lay down the component interconnections as needed, then propose a set of instructions that can be implemented easily. At that point, we proceed to write down a possible microcode sequence to implement each of the instruction codes we proposed. In the end, we have to modify our initial hardware design in order to accommodate certain features of our chosen instruction set.

Essentially, we have a *single bus* architecture. This is a 4-bit tristated signal bus that interconnects the B-INPUT of the ALU, the I/O lines of the SRAM, the input switches, and the outputs of the accumulator. Tristate drivers are used for both the switch and the accumulator in order to meet the requirement of the signal bus. The inherent nature of the single bus means that we can only send a single source of signal to this bus at a given time. This requirement alone dictates that a seemingly simple task may have to be executed using two or more clock cycles.

The other point worth mentioning is the fact that the output of the ALU alone determines the flow of data into the accumulator (except for ROTATE).

Our proposed set of instructions can be grouped into the following:

Operations	with memory	with switch	other
DATA TRANSFER	LDAA N	INPA	
	STAA N	STSW N	
	STSW N		
DATA PROCESSING			
- alter data			DECA, INCA
			CLRA, SETA
			CLC, SEC
- arithmetic	ADDA N	ADDA S	
	SUBA N	SUBA S	
- logic	ANDA N		
- rotate			RORA, ROLA
PROGRAM CONTROL			JMP N
			JCS N
			JEQ N
			JMI N

Our processor, however primitive in its present form, can be made to execute various combinations of the following basic tasks:

- data input
- memory access
- decision making

Presently this architecture dictates that the program steps must not occupy more than 16 memory addresses, and that the data is limited to 4 bits. There is no hardware support for the detection of any overflow in our present design. Both 4-bit unsigned or signed two's complement data formats can be used with the same hardware setup. **It is the responsibility of the user to define the choice of data format, and to interpret the results accordingly.**

Example 0. 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.

The solution involves the following program sequence

```
Address  ------ Program word -------   ------- Comments ---------
  (Hex)  (Binary)  (Hex)  (Mnemonic)

   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
```

Example 1. Compare 2 numbers X & Y and display the result.

1. If $X = Y$, all LEDs flashing.
   If $X < Y$, LED display rotates 1 bit "left".
   If $X > Y$, LED display rotates 1 bit "right".
2. For display purposes, the "automatic" clock (1 Hz.) is used.

Solution: by RWS

- **Equality Check** $X = Y$
  The ALU function used is $F = A - B - /C_{in}$, and $C_{in} = 0$. This function is incorporated into the instruction code JEQ N. With reference to the PAL equationss, $CALU = C_{in} = 0$ by default, and A_7 = 1101*AEBD.
- **Compare** $(X \geq Y$**,** $X < Y)$
  The ALU function used is $F = A - B - /C_{in}$, and $C_{in} = 1$. This function is implemented by the instruction code SUBA N.
  If $X < Y$, $/C_{out} = 1$, $(/C_o = 1)$
  If $X \geq Y$, $/C_{out} = 0$, $(/C_o = 0)$

This solution involves the following program sequence:

Note: "Y" is entered first and placed in memory. "X" is entered when the program counter displays a count of "2".

Example 1.asc file:

0010 2F 41 DB 2F 8F 11 FE CE 8F AF 59 FF FE 5B AE 5E

```
Address   ------ Program word -------   ------- Comments ---------
  (Hex)   (Binary)  (Hex)  (Mnemonic)

    0     0010 1111  2F     INPA         Input Y from switches
    1     0100 0001  41     STAA 1       Store Y into M1
    2     1101 1011  DB     JEQ B        Look at X; test if Y = X

    3     0010 1111  2F     INPA         Input X from switches
    4     1000 1111  8F     SEC          Required for SUBA N, borrow = 0
    5     0001 0001  11     SUBA 1       Subtract X-Y
    6     1111 1110  FE     CLRA         To ensure a 1-bit display
    7     1100 1110  CE     JCS E        If C = 1, X > Y

    8     1000 1111  8F     SEC          Set C = 1 for a rotating bit
    9     1010 1111  AF     ROLA         Continuous display for X < Y
    A     0101 1001  59     JMP 9

    B     1111 1111  FF     SETA         Display for X = Y
    C     1111 1110  FE     CLR A        Continuous flashing
    D     0101 1011  5B     JMP B

    E     1010 1110  AE     RORA         Display for X > Y; C = 1 and
    F     0101 1110  5E     JMP E        SEC is not required
```

Example 2. Counting the 1's of a given input data.

Solution:

- Clear the contents of M1 (*ans*) initially.
- Load any 4-bit number from switches & store in M2 (*data*).
- Check to see whether bit-3 of *data* is a '1'; if so, increment *ans*.
- Shift (rotate with Carry cleared) Left the *data*.
- Check to see whether *data* no longer contains '1's. (This is accomplished by comparing it to the switches which should have been all set to '0') If so, jump to an endless loop, else repeat from checking bit-3 again.

This solution involves the following program sequence: (introducing the use of conditional jump)

```
Address   ---------- Program word --------------   ------- Description --------
  (Hex)   (Binary)  (Hex)  (Address)  (Mnemonic)
                            Label      Code

    0     1111 1110  FE     start      CLRA
    1     0100 0001  41                STAA ans    Initialize answer to zero
    2     0010 1111  2F                INPA        Input a number from switches
    3     0100 0010  42     test       STAA data   Store into M2 as data
    4     1110 0110  E6                JMI  minus  Jump to 'minus' if bit-3 is a '1'
    5     0101 1010  5A                JMP  next     if not, then jump to 'next'
    6     0011 0001  31     minus      LDAA ans
    7     1001 1111  9F                INCA
```

```
8    0100 0001  41                  STAA ans      increment answer by 1
9    0011 0010  32                  LDAA data
A    1000 1110  8E     next         CLC          Prepare for
B    1010 1111  AF                  ROLA           the next bit position
C    1101 1110  DE                  JEQ  nomore  Jump to ‘nomore’ if there are no more 1’s
D    0101 0011  53                  JMP  test      if not, then jump to ‘test’
E    0011 0001  31     nomore       LDAA ans     Get the final answer
F    0101 1111  5F     loop         JMP  loop    An endless loop
```

Note: This solution introduces the use of *address labels* which makes the source code more readable. Both memory addresses and jump destinations may be represented by these labels. The programmer is responsible for the conversion of these labels to their hex values during the process of “hand” assembly.

You are now required to write and run a program from the following selection:

1. Input two 4-bit *signed two’s complement* numbers. Calculate the sum & convert the answer into a 4-bit *signed magnitude* format. (Ignore any overflow for now.)
2. Input an 8-bit unsigned number (upper 4-bit comes first) and store in two consecutive location (M-high & M-low). Carry out the “divided-by-2” operation (a shift-right operation involving two memory locations as well as the Carry-bit) on this number. Display the final result (the upper 4-bits come first).
3. Input four 4-bit numbers in succession. Compute the *average* of these numbers & store in memory.
4. Input two 4-bit unsigned numbers (A & B) and compute the product (AXB) by adding A B-times in *modulo-16* operation (i.e. ignore the overflow). to an initially cleared location.
5. Variation to #4: Input a single unsigned number and compute the square (AXA) of that number.
6. Simulate the output of a 4-bit twisted ring (Johnson) counter
7. Input a 4-bit number (A) & store in memory. Then count from 0 to A (in steps of 1) then from A to 0 & repeat from start.
8. Input two 4-bit numbers (A & B, $A < B$) & store in memory. Then count from A to B, then back to A and repeat.
9. Input two 4-bit numbers (A & B, $A < B$) & store in memory. Then count from A to B & flash all LEDs when A = B.
10. Load X & Y. If X = Y, all LEDs flashing. If X $\neq$ Y, rotate one bit continuously.
11. Subtract X - Y. If +ve or zero, all LEDs flashing. If -ve, rotate one bit continuously.
12. “Ping-Pong” display. A single one moves back and forth in accumulator display.

B Instruction Microcode

1. Address lines (A7-A2) for EPROMs 1 & 2 are generated by the program instructions (Op codes) stored in EPROM 3.
2. Address lines (A1-A0) for EPROMs 1 & 2 are generated by the microinstruction bits (D1-D0) stored in EPROM 2.
3. The CLR signal (SW7=0) manually resets both the program counter and the "next state" register to "zero". This starts the current program from the 1st microinstruction of the 1st program instruction.
4. To increase the number of possible program instruction codes available, the "N0" bit of some instructions (e.g. INCA,CLRA) is used as part of the instruction code.
5. For instructions not using N0 as a control bit, but as part of the address Operand (eg. ADDA N, JEQ N),the N0 bit could be a "1" or "0". Therefore these instructions must be programmed into EPROMS 1 & 2 twice: for N0=0 and for N0=1.
6. Pin "D7" of EPROM 2 is not used in the current processor design, but is available for future use.
7. In programming the EPROMs, use "1" for the "don't care" (i.e. X) conditions. A "1" can be changed to a "0" by **EDIT** of the EPROM programmer.

B.1 Example 1: ADDA N

Add contents of SRAM (address N3-N0) into the accumulator.

EPROMs 1&2 Address Lines								EPROM1								EPROM2							
	EPROM3							PAL	ACCA		ALU 181						DATA PATH			161		EPROMs	
JP	N0	OP Code				Mic Code		NCC	S1	S0	M	S3	S2	S1	S0		/AS	WR	SM	/PE	CNT	A1+	A0+
A7	A6	A5	A4	A3	A2	A1	A0	D7	D6	D5	D4	D3	D2	D1	D0	D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	0	0	0	0	0	0	1	1	0	1	0	0	1	NC	1	0	0	1	1	0	0
0	0	0	0	0	0	0	1																
0	0	0	0	0	0	1	0																
0	0	0	0	0	0	1	1																

Table 1: ADDA N

EPROM 3 (Mic code 00)

A7	JP=0	No **JUMP** has been initiated
A6	N0=0,1	Both must be included (see note #5 pg.1).
A5–A2	0000	Selects the assigned Op code **ADDA N**.
A1,A0	00	Selects the first of 4 possible *microinstructions* for a particular *program* instruction.

EPROM 1 (Mic code 00)

D7	NCC=0	Allows the carry bit “C” to change, if required.
D6,D5	S1,S0=11	Selects “parallel load” for the **ACCA** shift register.
D4	M=0	Selects the arithmetic ALU function:
D3–D0	1001	F=(A plus B plus Carry).

EPROM 2 (Mic code 00)

D7	NC	No connection (see note #6 pg. 1).
D6	/AS=1	Accumulator output is disconnected from data bus.
D5	WR=0	**SRAM** “read enable”: /WE=/(WR*/CLK)=1.[1]
D4	SM=0	Selects SRAM chip; disconnects SW from data bus.
D3	/PE=1	Disables the **JUMP** address from loading into the program counter.
D2	CNT=1	Increments the program counter for addressing the **next** program instruction stored in EPROM 3. If CNT=0, the program steps to the next microinstruction in the **same** program instruction (i.e. the same OP code).
D1,D0	A1+,A0+	Indicates the next microinstruction.

B.2 Example 2: JEQ N

Jump to program address “N” if A=B. Note: SW3–SW0 data is used for ALU “B” inputs.

EPROMs 1&2 Address Lines								EPROM1								EPROM2							
	EPROM3							PAL	ACCA		ALU 181						DATA PATH			161		EPROMs	
JP	N0	OP Code				Mic Code		NCC	S1	S0	M	S3	S2	S1	S0		/AS	WR	SM	/PE	CNT	A1+	A0+
A7	A6	A5	A4	A3	A2	A1	A0	D7	D6	D5	D4	D3	D2	D1	D0	D7	D6	D5	D4	D3	D2	D1	D0
0	0	1	1	0	1	0	0	1	0	0	0	0	1	1	0	NC	1	0	1	1	0	0	1
0	0	1	1	0	1	0	1	1	0	0	X	X	X	X	X	NC	1	0	X	1	1	0	0
0	0	1	1	0	1	1	0																
0	0	1	1	0	1	1	1																

Table 2: JEQ N (JP=0)

EPROM 3 (Mic code 00)

A7	JP=0	No **JUMP** is indicated.
A6	N0=0,1	Both must be included (see note #5, pg. 1)
A5–A2	1101	1101 is assigned to the Op code **JEQ N**.
A1,A0	00	Selects the first microinstruction.

EPROM 1 (Mic code 00)

D7	NCC=1	Prevents the carry bit “C” from changing.
D6,D5	S1,S0=00	Selects “no change” for the **ACCA** shift register.
D4	M=0	Selects the ALU function: F=A minus B (Cn=0); see
D3–D0	0110	the PAL equation for CALU (no carry, by default).

EPROM 2 (Mic code 00)

D7	NC	No connection.

[1] /CLK represents trailing edge of clock.

D6	/AS=1	ACCA disconnected from data bus.
D5	WR=0	Ensures no “writing” to SRAM (in case of “glitches”).
D4	SM=1	Disables SRAM chip & connects SW to data bus.
D3	/PE=1	Disables the **JUMP** address to the program counter.
D2	CNT=0	Program counter is **not** incremented; the Op code is unchanged.
D1,D0	01	* Selects the 2nd microinstruction, but with 2 possibilities: A7=0 or A7=1 (as determined by the “A7” PAL equation).

* **CASE 1: A7=0** (No JUMP)
The **JUMP** condition is not met (AEB=0) and the program advances, after the 2nd microinstruction, to the **next** program instruction (see table 2 - JEQ N, JP=0).

* **CASE 2: A7=1** (JUMP)
The **JUMP** condition is met (AEB=1) and the 2nd microinstruction (A1,A0=01) is then as shown in table 3 below. The program then **jumps** to the program instruction selected by the JUMP address “N”.

EPROMs 1&2 Address Lines								EPROM1								EPROM2							
	EPROM3							PAL	ACCA		ALU 181						DATA PATH			161		EPROMs	
JP	N0	OP Code				Mic Code		NCC	S1	S0	M	S3	S2	S1	S0		/AS	WR	SM	/PE	CNT	A1+	A0+
A7	A6	A5	A4	A3	A2	A1	A0	D7	D6	D5	D4	D3	D2	D1	D0	D7	D6	D5	D4	D3	D2	D1	D0
1	0	1	1	0	1	0	0	1	0	0	0	0	1	1	0	NC	1	0	1	1	0	0	1
1	0	1	1	0	1	0	1	1	0	0	X	X	X	X	X	NC	1	0	X	0	X	0	0
1	0	1	1	0	1	1	0																
1	0	1	1	0	1	1	1																

Table 3: JEQ N (JP=1)

EPROM 3 (Mic code 01)

A7	JP=1	**JUMP** condition is met (AEB=1).
A6–A2		Signal levels as in table 2 (A7=0 conditions).
A1,A0	01	Selects the 2nd microinstruction; identical to JMP N.

EPROM 1 (Mic code 01)

D7–D0		Signal levels as for A7=0 conditions.

EPROM 2 (Mic code 01)

D3	/PE=0	Enables the **JUMP** address to be loaded into the program counter.
D2	CNT=X	From device specifications: CNT=X when /PE=0.
D1,D0	00	Selects the 1st microinstruction in the program instruction selected by the **JUMP** address “N”.

B.3 Example 3: STSW N

Load **and** store SW data into memory location “N”.

NOTE 1: The STSW N Op code is useful because of the 16 line program limitation, but is considered “unconventional” in terms of conventional microprocessor practice: i.e. 2 **basic** operations are included in the same Op code. Combines the INPA and STAA N Op codes.

NOTE 2: Refer to note 5, page 1.

EPROMs 1&2 Address Lines								EPROM1								EPROM2							
	EPROM3							PAL	ACCA		ALU 181						DATA PATH			161		EPROMs	
JP	N0	OP Code				Mic Code		NCC	S1	S0	M	S3	S2	S1	S0		/AS	WR	SM	/PE	CNT	A1+	A0+
A7	A6	A5	A4	A3	A2	A1	A0	D7	D6	D5	D4	D3	D2	D1	D0	D7	D6	D5	D4	D3	D2	D1	D0
0	0	1	0	1	1	0	0	1	1	1	1	1	0	1	0	NC	1	0	1	1	0	0	1
0	0	1	0	1	1	0	1	1	0	0	X	X	X	X	X	NC	0	1	0	1	1	0	0
0	0	1	0	1	1	1	0																
0	0	1	0	1	1	1	1																

Table 4: STSW N

MIC CODE

A1,A0 = 00 SW data to ACCA. Uses the ALU function F=B.
A1,A0 = 01 ACCA to SRAM (address N).

You are now required to complete the microcode tables on the next few pages.

Instruction Microcode - To Be Completed by Student

PAGE 0 0, (A7 = 0, A6 = 0)

	EPROMs 1&2 Address Lines									EPROM1										EPROM2								
			EPROM3							PAL	ACCA		ALU 181								DATA PATH			PC 161		EPROMs		
	JMP	N0	OP Code				Mic Code			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	0000	0	1	1	0	1	0	0	1	69	X	1	0	0	1	1	0	0	CC
	0	0	0	0	0	0	0	1	0001	X	X	X	X	X	X	X	X	FF	X	X	X	X	X	X	X	X	FF
	0	0	0	0	0	0	1	0	0002	X	X	X	X	X	X	X	X	FF	X	X	X	X	X	X	X	X	FF
	0	0	0	0	0	0	1	1	0003	X	X	X	X	X	X	X	X	FF	X	X	X	X	X	X	X	X	FF
SUBA N	0	0	0	0	0	1	0	0	0004																		
	0	0	0	0	0	1	0	1	0005																		
	0	0	0	0	0	1	1	0	0006																		
	0	0	0	0	0	1	1	1	0007																		
INPA	0	0	0	0	1	0	0	0	0008	1	1	1	1	1	0	1	0	FA	X	1	X	1	1	1	0	0	FC
	0	0	0	0	1	0	0	1	0009	X	X	X	X	X	X	X	X	FF	X	X	X	X	X	X	X	X	FF
	0	0	0	0	1	0	1	0	000A	X	X	X	X	X	X	X	X	FF	X	X	X	X	X	X	X	X	FF
	0	0	0	0	1	0	1	1	000B	X	X	X	X	X	X	X	X	FF	X	X	X	X	X	X	X	X	FF
LDAA N	0	0	0	0	1	1	0	0	000C	1	1	1	1	1	0	1	0	FA	X	1	0	0	1	1	0	0	CC
	0	0	0	0	1	1	0	1		X	X	X	X	X	X	X	X	FF	X	X	X	X	X	X	X	X	FF
	0	0	0	0	1	1	1	0		X	X	X	X	X	X	X	X	FF	X	X	X	X	X	X	X	X	FF
	0	0	0	0	1	1	1	1		X	X	X	X	X	X	X	X	FF	X	X	X	X	X	X	X	X	FF
STAA N	0	0	0	1	0	0	0	0																			
	0	0	0	1	0	0	0	1																			
	0	0	0	1	0	0	1	0																			
	0	0	0	1	0	0	1	1																			
JMP N	0	0	0	1	0	1	0	0		1	0	0	X	X	X	X	X		X	1	0	X	0	0	0	0	
	0	0	0	1	0	1	0	1																			
	0	0	0	1	0	1	1	0																			
	0	0	0	1	0	1	1	1																			
ADDA S	0	0	0	1	1	0	0	0																			
	0	0	0	1	1	0	0	1																			
	0	0	0	1	1	0	1	0																			
	0	0	0	1	1	0	1	1																			
ANDA N	0	0	0	1	1	1	0	0																			
	0	0	0	1	1	1	0	1																			
	0	0	0	1	1	1	1	0																			
	0	0	0	1	1	1	1	1																			
CLC	0	0	1	0	0	0	0	0																			
	0	0	1	0	0	0	0	1																			
	0	0	1	0	0	0	1	0																			
	0	0	1	0	0	0	1	1																			
DECA	0	0	1	0	0	1	0	0																			
	0	0	1	0	0	1	0	1																			
	0	0	1	0	0	1	1	0																			
	0	0	1	0	0	1	1	1																			
RORA	0	0	1	0	1	0	0	0		0	1	0	X	X	X	X	X		X	1	0	X	1	1	0	0	
	0	0	1	0	1	0	0	1																			
	0	0	1	0	1	0	1	0																			
	0	0	1	0	1	0	1	1																			
STSW N	0	0	1	0	1	1	0	0																			
	0	0	1	0	1	1	0	1																			
	0	0	1	0	1	1	1	0																			
	0	0	1	0	1	1	1	1																			
JCS N	0	0	1	1	0	0	0	0		1	0	0	X	X	X	X	X		X	1	0	X	1	1	0	0	
	0	0	1	1	0	0	0	1																			
	0	0	1	1	0	0	1	0																			
	0	0	1	1	0	0	1	1																			
JEQ N	0	0	1	1	0	1	0	0		1	0	0	0	0	1	1	0		X	1	0	1	1	0	0	1	
	0	0	1	1	0	1	0	1		1	0	0	X	X	X	X	X		X	1	0	X	1	1	0	0	
	0	0	1	1	0	1	1	0																			
	0	0	1	1	0	1	1	1																			
JMI N	0	0	1	1	1	0	0	0																			
	0	0	1	1	1	0	0	1																			
	0	0	1	1	1	0	1	0																			
	0	0	1	1	1	0	1	1																			
CLRA	0	0	1	1	1	1	0	0																			
	0	0	1	1	1	1	0	1																			
	0	0	1	1	1	1	1	0																			
	0	0	1	1	1	1	1	1																			

$\left.\begin{matrix}\\ \\ \\ \end{matrix}\right\}$ You may ignore unused lines

Instruction Microcode - To Be Completed by Student

PAGE 0 1, (A7 = 0, A6 = 1)

	EPROMs 1&2 Address Lines									EPROM1										EPROM2								
		EPROM3								PAL	ACCA		ALU 181								DATA PATH			PC 161		EPROMs		
	JMP	N0	OP Code				Mic Code			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	0040	0	1	1	0	1	0	0	1	69	X	1	0	0	1	1	0	0	CC
	0	1	0	0	0	0	0	1																			
	0	1	0	0	0	0	1	0																			
	0	1	0	0	0	0	1	1																			
SUBA N	0	1	0	0	0	1	0	0																			
	0	1	0	0	0	1	0	1																			
	0	1	0	0	0	1	1	0																			
	0	1	0	0	0	1	1	1																			
INPA	0	1	0	0	1	0	0	0	0048	1	1	1	1	1	0	1	0	FA	X	1	X	1	1	1	0	0	FC
	0	1	0	0	1	0	0	1																			
	0	1	0	0	1	0	1	0																			
	0	1	0	0	1	0	1	1																			
LDAA N	0	1	0	0	1	1	0	0																			
	0	1	0	0	1	1	0	1																			
	0	1	0	0	1	1	1	0																			
	0	1	0	0	1	1	1	1																			
STAA N	0	1	0	1	0	0	0	0																			
	0	1	0	1	0	0	0	1																			
	0	1	0	1	0	0	1	0																			
	0	1	0	1	0	0	1	1																			
JMP N	0	1	0	1	0	1	0	0																			
	0	1	0	1	0	1	0	1																			
	0	1	0	1	0	1	1	0																			
	0	1	0	1	0	1	1	1																			
SUBA S	0	1	0	1	1	0	0	0																			
	0	1	0	1	1	0	0	1																			
	0	1	0	1	1	0	1	0																			
	0	1	0	1	1	0	1	1																			
ANDA N	0	1	0	1	1	1	0	0																			
	0	1	0	1	1	1	0	1																			
	0	1	0	1	1	1	1	0																			
	0	1	0	1	1	1	1	1																			
SEC	0	1	1	0	0	0	0	0																			
	0	1	1	0	0	0	0	1																			
	0	1	1	0	0	0	1	0																			
	0	1	1	0	0	0	1	1																			
INCA	0	1	1	0	0	1	0	0																			
	0	1	1	0	0	1	0	1																			
	0	1	1	0	0	1	1	0																			
	0	1	1	0	0	1	1	1																			
ROLA	0	1	1	0	1	0	0	0																			
	0	1	1	0	1	0	0	1																			
	0	1	1	0	1	0	1	0																			
	0	1	1	0	1	0	1	1																			
STSW N	0	1	1	0	1	1	0	0																			
	0	1	1	0	1	1	0	1																			
	0	1	1	0	1	1	1	0																			
	0	1	1	0	1	1	1	1																			
JCS N	0	1	1	1	0	0	0	0																			
	0	1	1	1	0	0	0	1																			
	0	1	1	1	0	0	1	0																			
	0	1	1	1	0	0	1	1																			
JEQ N	0	1	1	1	0	1	0	0																			
	0	1	1	1	0	1	0	1																			
	0	1	1	1	0	1	1	0																			
	0	1	1	1	0	1	1	1																			
JMI N	0	1	1	1	1	0	0	0																			
	0	1	1	1	1	0	0	1																			
	0	1	1	1	1	0	1	0																			
	0	1	1	1	1	0	1	1																			
SETA	0	1	1	1	1	1	0	0																			
	0	1	1	1	1	1	0	1																			
	0	1	1	1	1	1	1	0																			
	0	1	1	1	1	1	1	1																			

Instruction Microcode - To Be Completed by Student

PAGE 1 0, (A7 = 1, A6 = 0)

	EPROMs 1&2 Address Lines									EPROM1									EPROM2								
		EPROM3								PAL	ACCA		ALU 181							DATA PATH			PC 161		EPROMs		
	JMP	N0	OP Code				Mic Code			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
JCS N	1	0	1	1	0	0	0	0	00B0	1	0	0	X	X	X	X	X	9F	X	1	0	X	0	0	0	0	D0
	1	0	1	1	0	0	0	1																			
	1	0	1	1	0	0	1	0																			
	1	0	1	1	0	0	1	1																			
JEQ N	1	0	1	1	0	1	0	0	00B4	1	0	0	0	0	1	1	0	86	X	1	0	1	1	0	0	1	D9
	1	0	1	1	0	1	0	1	00B5	1	0	0	X	X	X	X	X	9F	X	1	0	X	0	0	0	0	D0
	1	0	1	1	0	1	1	0																			
	1	0	1	1	0	1	1	1																			
JMI N	1	0	1	1	1	0	0	0																			
	1	0	1	1	1	0	0	1																			
	1	0	1	1	1	0	1	0																			
	1	0	1	1	1	0	1	1																			

PAGE 1 1, (A7 = 1, A6 = 1)

JCS N	1	1	1	1	0	0	0	0																			
	1	1	1	1	0	0	0	1																			
	1	1	1	1	0	0	1	0																			
	1	1	1	1	0	0	1	1																			
JEQ N	1	1	1	1	0	1	0	0																			
	1	1	1	1	0	1	0	1																			
	1	1	1	1	0	1	1	0																			
	1	1	1	1	0	1	1	1																			
JMI N	1	1	1	1	1	0	0	0																			
	1	1	1	1	1	0	0	1																			
	1	1	1	1	1	0	1	0																			
	1	1	1	1	1	0	1	1																			