REPORT ON

A PROCESS CONTROL PROGRAM

AND

CONTROL PANEL

Submitted To

Dr. D.W. Russell
Real-Time Computing
Pennsylvania State University
Great Valley, Pennsylvania

By

Joseph T. Wunderlich
Engineering Graduate Student

July 27, 1990

TABLE OF CONTENTS

I. INTRODUCTION

This report discusses the design of a process control program and control panel to control a bottling plant.

The design of the control program is based on the physical system shown in [Figure I-1]. The tank sizes, pump ratings, heater rating, and heater temperature setting were determined by analyzing a simulation program prior to the purchase of the tanks, pumps, and heater. The system mixes ingredients and cooks them according to the following recipe:

One Part	Ingredient #1
Two Parts	Ingredient #2
Three Parts	Ingredient #3
Four Parts	Ingredient #4

The 100 gallons (total) of ingredients are put into the reactor and then "cooked" according to the following heat balance equation:

$$Tr=[Uh\ (HTR)\ [Te-Tr]^{2}\ -1/10\ (Tr-Ta)]t$$

Tr = Reactor temperature (C)
Uh = Control signal to turn on heater (0 or 1)
HTR = Heater constant (C/min.)
Te = Heater temperature setting (C)
Ta = Ambient temperature
t = Process time

The 100 gallons in the reactor are heated to (140 C), then cooled to (100 C).

The program is designed to respond to disasters detected by the program. These disasters are:

(i) A pump stuck on or off
(ii) The heater stuck on or off
(iii) The conveyor stuck on or off
(iv) Running out of empty bottles
(v) Running out of ingredients

The program is designed to produce 12 ounce bottles of product. Every bottle must have at least 12 ounces of product, but, the absolute capacity of each bottle is 14 ounces so that possible overfills will not spill fluid onto the conveyor.

The following assumptions are adhered to throughout the control of the process:

(i) No fluid will ever be left in the reactor.

(ii) The ingredients are mixed by the turbulence of pumping them into the reactor. No mechanical mixing is necessary.

(iii) Turbulence will not effect the volume levels.

(iv) The ingredients enter the reactor at ambient temperature.

The original conceptual design for this bottling plant included taking measurements from tank fluid level sensors, a reactor temperature sensor, and an ambient temperature sensor. However, the following additional measurements are taken from the system:

(i) Analog flow rates from flow-meters installed in pipes at the exhaust side of every pump.

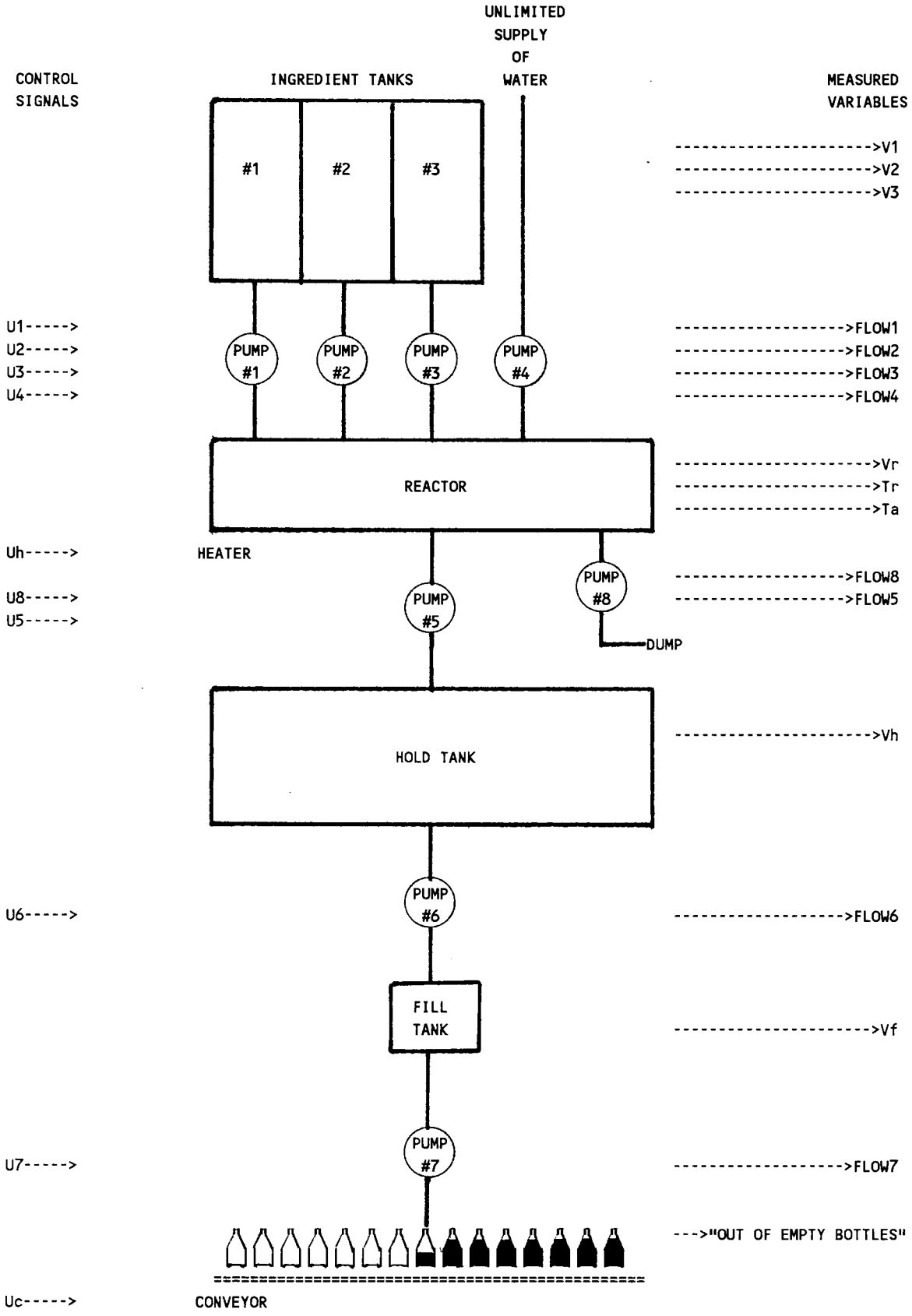
(ii) A digital signal from the heater indicating that it is on.

(iii) A digital signal from the conveyor indicating that it is on.

(iv) A digital signal from the empty bottle supply bin indicating that there are no more bottles.

All of these readings, except (iv), are compared to previously sent control signals to see if the equipment is responding correctly. Therefore, the program detects whether or not equipment is stuck on or stuck off. If one of these disasters does occur, the disaster controls will respond accordingly, and the control panel will indicate exactly what has occurred. The running out of bottles signal (iv), differs only in that it automatically indicates a disaster.

In addition to measured readings from the physical system, feedback from the control panel will indicate if any of the connections to the control lamps has been broken, or if a lamp has "burned-out". If either of these has occurred, the appropriate lamp on the control panel will be turned on. (ie., open circuit indicator lamps). This will be discussed further in [Section (II.) CONTROL PANEL].


UNLIMITED
SUPPLY
OF
WATER
CONTROL
SIGNALS
INGREDIENT TANKS
MEASURED
VARIABLES
#1
#2
#3
----------->V1
----------->V2
----------->V3
U1----->
U2----->
U3----->
U4----->
PUMP
#1
PUMP
#2
PUMP
#3
PUMP
#4
----------->FLOW1
----------->FLOW2
----------->FLOW3
----------->FLOW4
REACTOR
----------->Vr
----------->Tr
----------->Ta
Uh----->
HEATER
U8----->
U5----->
PUMP
#5
PUMP
#8
DUMP
----------->FLOW8
----------->FLOW5
HOLD TANK
----------->Vh
U6----->
PUMP
#6
----------->FLOW6
FILL
TANK
----------->Vf
U7----->
PUMP
#7
----------->FLOW7
--->"OUT OF EMPTY BOTTLES"
Uc----->
CONVEYOR

(Figure I-1)

II. CONTROL PANEL

The control panel for controlling the bottling plant is shown on [Figure II-1]. The top row of indicator lamps indicate the present control signals being sent to the physical system. These lamps are in parallel with the wires going to the physical system. The digital signals to light these lamps are sent to the control panel from (DOUT channel #1-10).

Coming from these control lamps are feedback signals indicating whether or not the control lamp is burnt-out, or the wire going to the control lamp has been broken. These signals are measured as analog voltages and are taken in to the computer through (ADC channel #9-18). An analog voltage of less than one volt is considered as an open circuit. If a lamp is burnt or a wire is broken, the second row of lamps are lit by signals from (DOUT channel #11-20). These signals are created within the control program by comparing the previous control signals sent out with the most recent feedback from the control lamps.

The third and fourth row of indicator lamps will indicate whether or not a specific piece of equipment is stuck on or stuck off. The digital control signals coming from (DOUT channel #21-40), light these lamps. These signals are created within the control program by comparing the last control signals sent out with analog and digital measurements that have been taken from the physical system. The analog measurements are taken from flow meters which have been installed into the pipes at the exhaust-side of every pump.

The digital measurements taken from the physical system consist of a digital signal from the heater indicating that it is on, and a digital signal from the conveyor indicating that it is on.

Located in the first row following the idicator lamps is a row of volt meters indicating the analog measurements of the tank volumes, the reactor temperature, and the ambient temperature. These same signals are taken into the computer through (ADC channel #1-8).

The next row of volt meters measures the flow from the flow meters as previously mentioned. These analog values are also taken into the computer through (ADC channel #19-26). Located on this same row of indicators are two lamps which are lit by digital signals from the heater and conveyor as previously mentioned. These two digital signals are taken into the computer through (DIN channel #1 and 2).

At the bottom lefthand corner of the control panel is a sliding switch potentiometer. This switch is used for choosing a requested volume of product to be produced by the physical system. This switch is calibrated up to 500,000 gallons of product and is sensitive to requested volumes to the exact gallon. Directly below this switch is an L.E.D. which displays the setting of the switch. Adjacent to the L.E.D. is a pushbutton which is part of a circuit that produces a single-shot pulse into (DIN channel #3). The signal that is sent into this channel is used by the control program to produce the desired volume of product.

However, if a new signal is pulsed into the program during the production of a previously input desired volume of product, the program will respond in one of two ways:

1.) If the present volume of product that has already been produced is less than the new requested volume, the program will continue production until the new requested volume is completed.

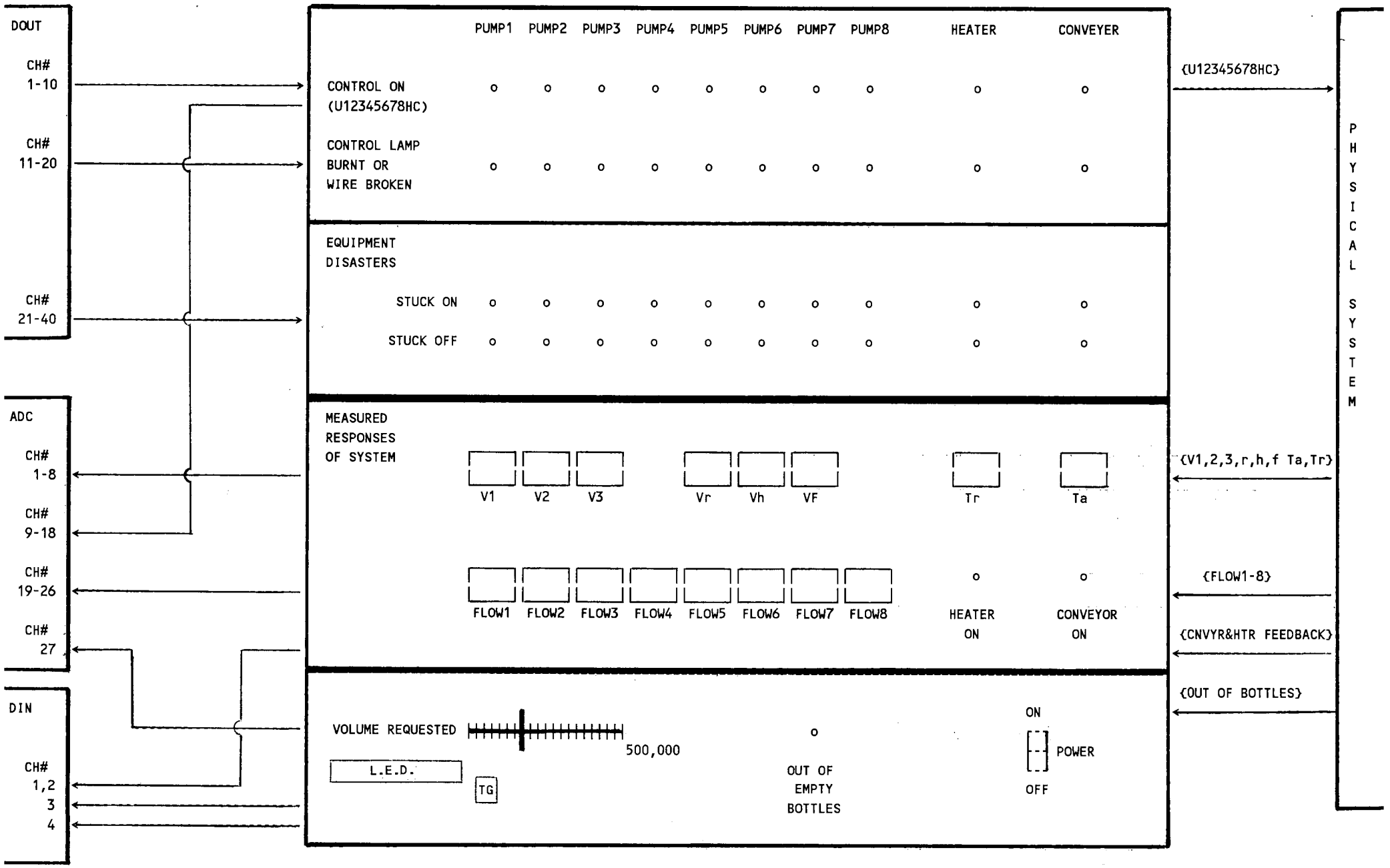
2.) If the new requested volume is less than the volume that has already been produced due to the last requested volume, the system will immediately shut down.

The analog volume request signal is sent from the control panel into the computer through (ADC channel #27).

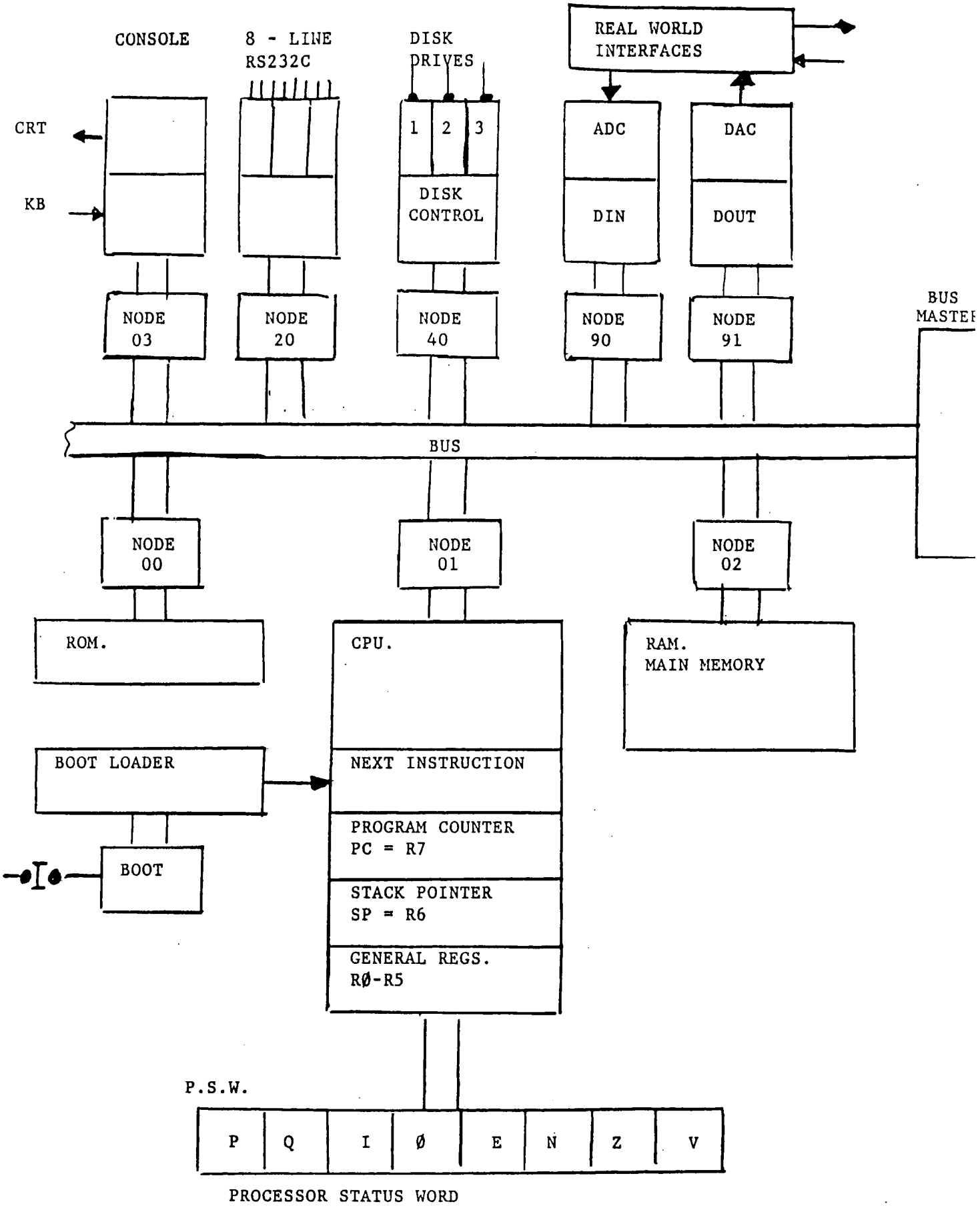
Located on the bottom middle of the control panel is an idicator lamp which indicates if the supply of empty bottles has run out. The digital signal which lights this lamp is sent from the bin where the empty bottles are stored. This digital signal is taken into the computer through (DIN channel #3), and is used by the control program to indicate a type of disaster.

Located on the bottom righthand corner of the control panel is an on/off switch. This creates a digital signal which is taken into the computer through (DIN channel #4). The control program uses this signal to initiate the execution of the control program. If the on/off switch is turned off during the production of a requested volume of product, the control program will treat this as a type of disaster and shut down the physical system accordingly.

The diagram shown in [Figure II-2], represents the organization of the computer system.



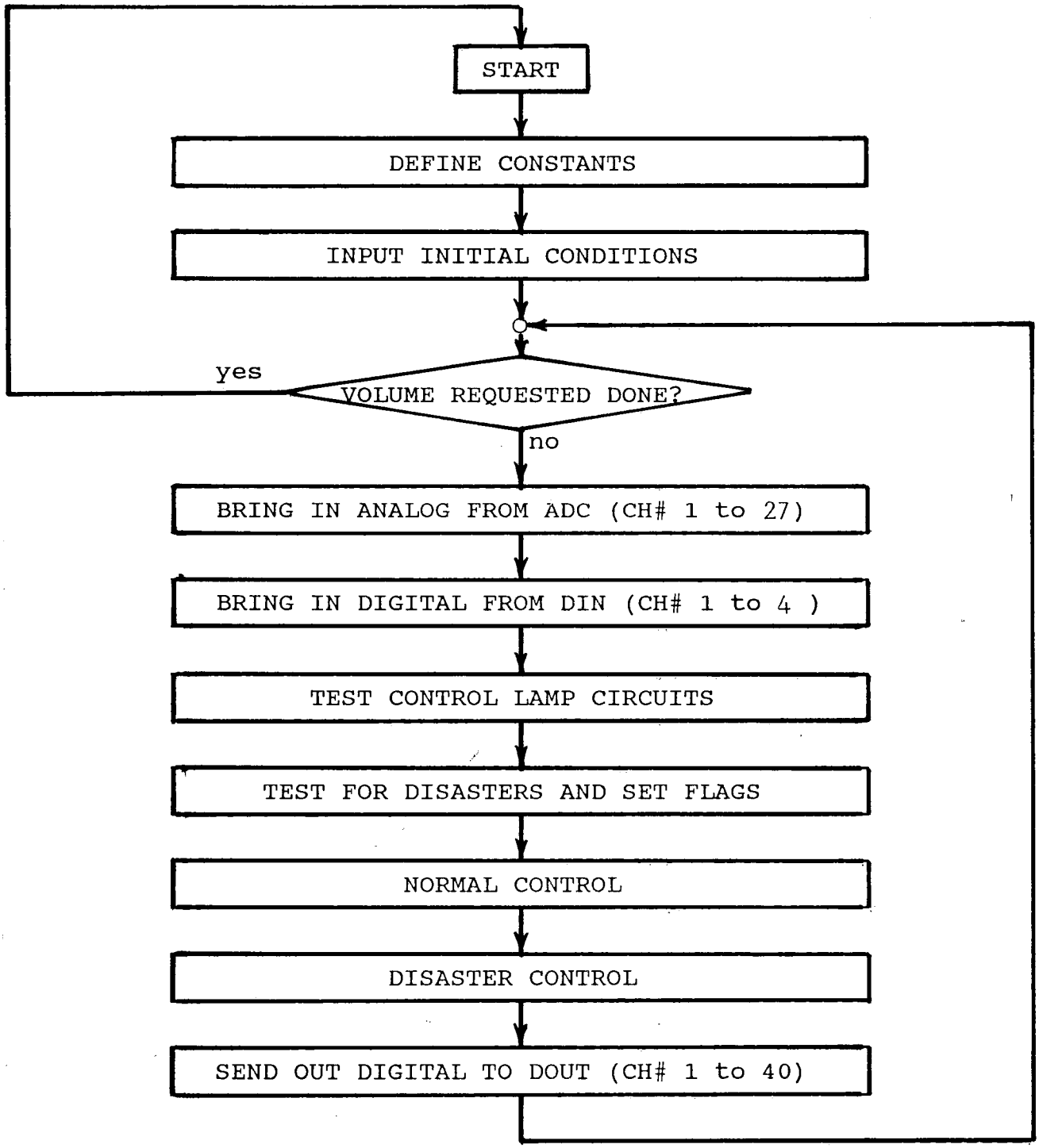
(Figure II-1) CONTROL PANEL


CONSOLE
8 - LINE
RS232C
DISK
DRIVES
REAL WORLD
INTERFACES
CRT
KB
1
2
3
ADC
DAC
DISK
CONTROL
DIN
DOUT
NODE
03
NODE
20
NODE
40
NODE
90
NODE
91
BUS
MASTER
BUS
NODE
00
NODE
01
NODE
02
ROM.
CPU.
RAM.
MAIN MEMORY
BOOT LOADER
NEXT INSTRUCTION
PROGRAM COUNTER
PC = R7
BOOT
STACK POINTER
SP = R6
GENERAL REGS.
RØ-R5
P.S.W.
P
Q
I
Ø
E
N
Z
V
PROCESSOR STATUS WORD

(Figure II-2)

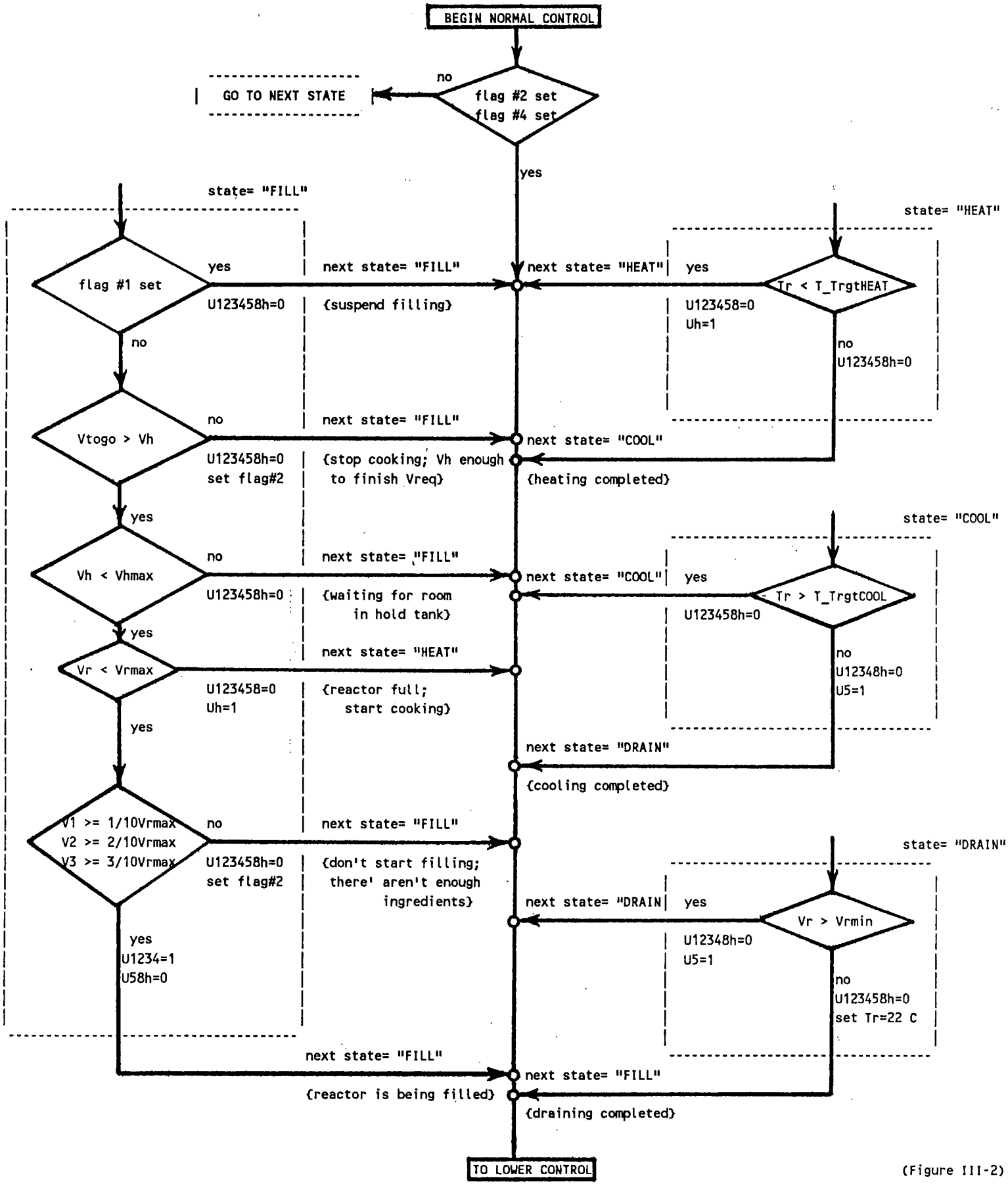
III. CONTROL PROGRAM DESIGN AND CODING

The overall flow of the control program is show in [Figure III-1]. The flow of the normal and disaster control segments of the program are shown in [Figures III-2, III-3, III-4]. The coding of the control program was done in Assembly Language using the instruction set shown in [Figures III-5, and III-6].

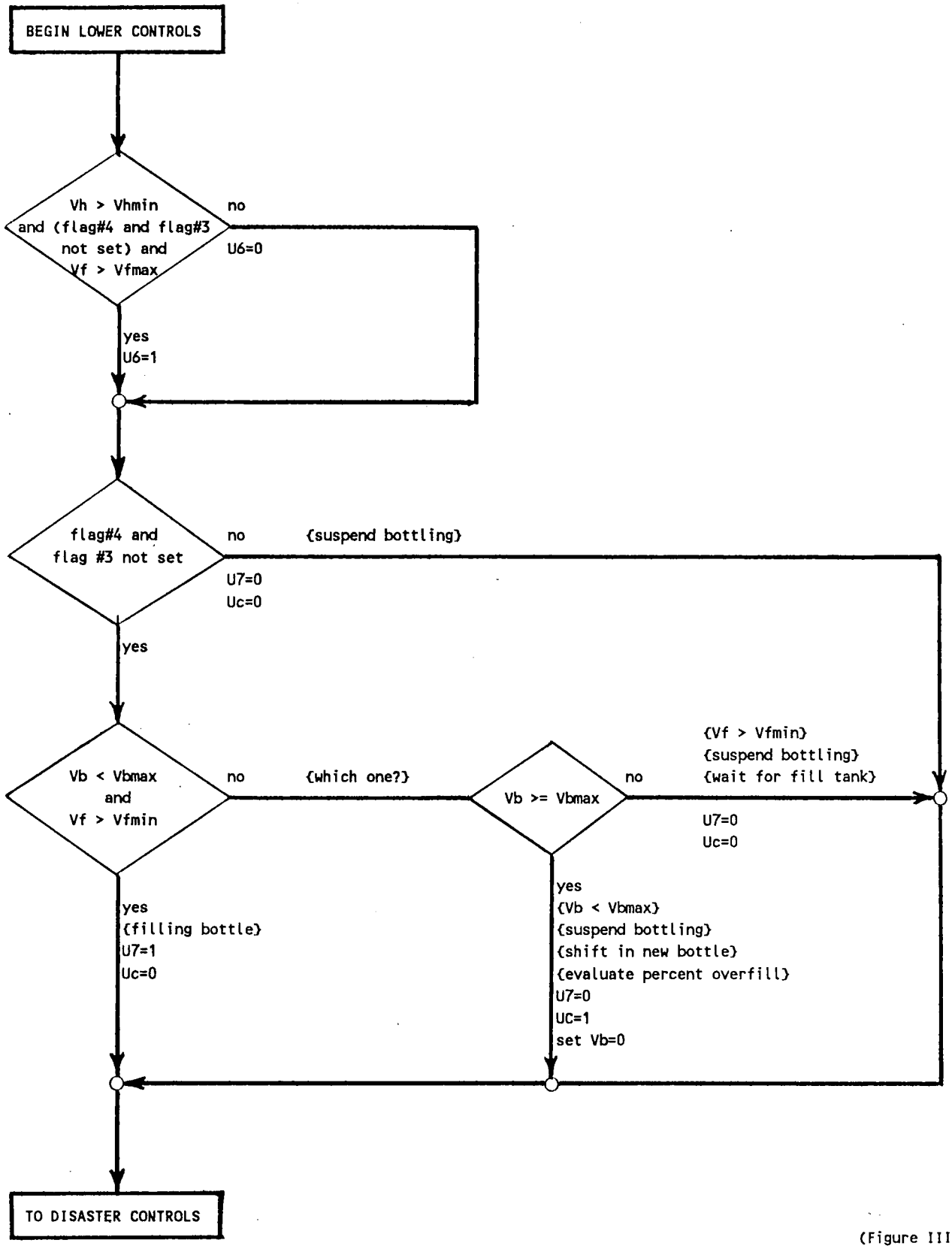


*NOTE: EVEN THOUGH THIS FLOWCHART SHOWS AN ENDLESSLY LOOPING PROGRAM, THERE ARE SIX DIFFERENT SOFTWARE CONSTRUCTS IN THIS PROGRAM TO ALLOW THIS ENDLESSLY LOOPING TASK TO BE EXITED. THESE CONSTRUCTS ARE LISTED AT THE END OF THE CODE.

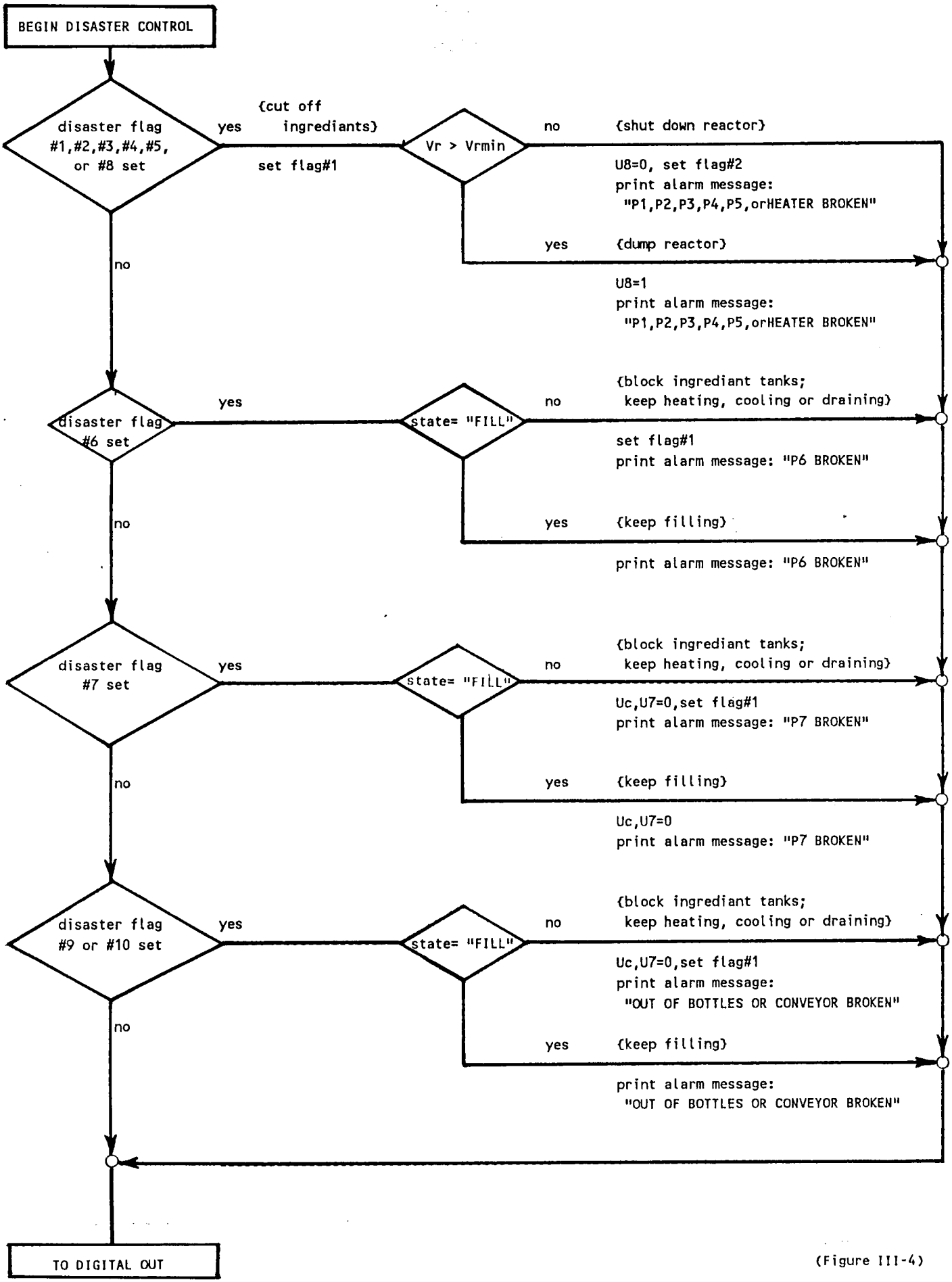
(Figure III-1)


BEGIN NORMAL CONTROL
flag #2 set
flag #4 set
no
GO TO NEXT STATE
yes
state= "FILL"
flag #1 set
yes
next state= "FILL"
U123458h=0
{suspend filling}
no
Vtogo > Vh
no
next state= "FILL"
U123458h=0
set flag#2
{stop cooking; Vh enough to finish Vreq}
yes
Vh < Vhmax
no
next state= "FILL"
U123458h=0
{waiting for room in hold tank}
yes
Vr < Vrmax
next state= "HEAT"
U123458=0
Uh=1
{reactor full; start cooking}
yes
V1 >= 1/10Vrmax
V2 >= 2/10Vrmax
V3 >= 3/10Vrmax
no
next state= "FILL"
U123458h=0
set flag#2
{don't start filling; there' aren't enough ingredients}
yes
U1234=1
U58h=0
next state= "FILL"
{reactor is being filled}
next state= "HEAT"
state= "HEAT"
yes
Tr < T_TrgtHEAT
U123458=0
Uh=1
no
U123458h=0
next state= "COOL"
{heating completed}
next state= "COOL"
state= "COOL"
yes
Tr > T_TrgtCOOL
U123458h=0
no
U12348h=0
U5=1
next state= "DRAIN"
{cooling completed}
next state= "DRAIN"
state= "DRAIN"
yes
Vr > Vrmin
U12348h=0
U5=1
no
U123458h=0
set Tr=22 C
next state= "FILL"
{draining completed}
TO LOWER CONTROL

(Figure III-2)


BEGIN LOWER CONTROLS
Vh > Vhmin and (flag#4 and flag#3 not set) and Vf > Vfmax
no
U6=0
yes
U6=1
flag#4 and flag #3 not set
no
{suspend bottling}
U7=0
Uc=0
yes
Vb < Vbmax and Vf > Vfmin
no
{which one?}
Vb >= Vbmax
no
{Vf > Vfmin}
{suspend bottling}
{wait for fill tank}
U7=0
Uc=0
yes
{filling bottle}
U7=1
Uc=0
yes
{Vb < Vbmax}
{suspend bottling}
{shift in new bottle}
{evaluate percent overfill}
U7=0
UC=1
set Vb=0
TO DISASTER CONTROLS

(Figure III-3)


BEGIN DISASTER CONTROL
disaster flag #1,#2,#3,#4,#5, or #8 set
yes
{cut off ingrediants}
set flag#1
Vr > Vrmin
no
{shut down reactor}
U8=0, set flag#2
print alarm message: "P1,P2,P3,P4,P5,orHEATER BROKEN"
yes
{dump reactor}
no
U8=1
print alarm message: "P1,P2,P3,P4,P5,orHEATER BROKEN"
disaster flag #6 set
yes
state= "FILL"
no
{block ingrediant tanks; keep heating, cooling or draining}
set flag#1
print alarm message: "P6 BROKEN"
yes
{keep filling}
no
print alarm message: "P6 BROKEN"
disaster flag #7 set
yes
state= "FILL"
no
{block ingrediant tanks; keep heating, cooling or draining}
Uc,U7=0,set flag#1
print alarm message: "P7 BROKEN"
yes
{keep filling}
no
Uc,U7=0
print alarm message: "P7 BROKEN"
disaster flag #9 or #10 set
yes
state= "FILL"
no
{block ingrediant tanks; keep heating, cooling or draining}
Uc,U7=0,set flag#1
print alarm message: "OUT OF BOTTLES OR CONVEYOR BROKEN"
yes
{keep filling}
no
print alarm message: "OUT OF BOTTLES OR CONVEYOR BROKEN"
TO DIGITAL OUT

(Figure III-4)

INSTRUCTION SET

[Ri = Register A = Address N =Number ( ) = Contents of]

Op Code	Mnemonic	Operand 1	Operand 2	Notes
00	HLT	-	-	HALT and EXIT PC = 0
01	NOP	-	-	No Operation
02	ION	-	-	Turn Interrupt On
03	IOFF	-	-	Turn Interrupt Off
04	CLI	-	-	Clear Interrupt (I) Bit
05	PAUSE	N	-	Wait 'N' 1/100 Sec.
				[N = Timer #1-255].
10	CLTIM	N	-	Clear Timer N
11	RTIM	N	Ri	Reset Timer 'N' With (Ri).
12	BRTIM	N	A	Branch To ADDR = A When Time UP.
				[N = RS232C Channel #1-8].
20	CLQ	N	-	Clear Queue (Q) Bit
21	TSTQ	N	-	Set Z=1 If Item Waiting In Q.
22	RSQ	N	Ri	Put Value From Q-Buffer Into Ri.
23	WRQ	N	Ri	Put Ri Into Q-Buffer.
30	CLR	Ri	-	Clear Ri To Zero, ZBit=1
31	INC	Ri	-	Increment Ri, Set N & Z & V
32	DEC	Ri	-	Decrement Ri, Set N & Z & V
35	RPSW	Ri	-	Copy PSW Into Ri
36	WPSW	Ri	-	Copy Ri Into PSW
37	PUSH	Ri	-	Copy Ri Onto Stack
38	POP	Ri	-	Copy Stack Into Ri
50	BR	A		Unconditional Branch to Address A
51	BRI	Ri		Branch to A = (Ri).
52	BEQ	A		Branch If Z bit =1 to A.
53	BNE	A		Branch If Z bit=0 to A.
54	BPL	A		Branch If Z = 0 and N = 0 to A.
55	BNG	A		Branch If N = 1
56	BRV	A		Branch If V = 1
57	BINT	A		Branch If INT - bit = 1.
58	BQ	A		Branch If Q - bit = 1.
59	BER	A		Branch If E - bit = 1.
60	ADD	Ri	Rj	Rj = Rj + Ri
61	SUB	Ri	Rj	Rj = Rj - Ri
62	MULT	Ri	Rj	Rj = Rj * Ri
63	DIV	Ri	Rj	Rj = Rj/Ri /V = 1 on overflow
64	CMP	Ri	Rj	Z = 1 If Rj = Ri

(Figure III-5)

INSTRUCTION SET (continued)

[Ri = Register A = Address N = Nuumber ( ) = Contents of]

OP Code	Mnemonic	Operand 1	Operand 2	Notes
70	LDI	Ri	N	Put N - Ri
71	STO	Ri	A	Store (Ri) in A
72	STX	Ri	Rj	Store (Ri) in A = (Rj)
73	LD	Ri	A	Store (A) in Ri
74	LDX	Ri	Rj	Store (A = (Rj) in Ri
80	JSR	A		Call Subroutine at A
81	RTN			Return from Subroutine.
85	CRT	Ri		Display Ri on CRT
86	KB	Ri		Put KB Value in Ri
				[N = Channel #]
90	TADC	N		Start ADC, Z = 1 when done
91	ADC	N	Ri	ADC Value Into Ri
92	TDIN	N		Start DIN, Z = 1 when done
93	DIN	N	Ri	Digital Input (0/1) Into Ri
94	TDAC	N		Z = 1 If DAC Ready
95	DAC	N	Ri	Send Ri to DAC
96	TDOUT	N		Z = 1 If DOUT Ready
97	DOUT	N	Ri	Send Ri to Digital Output.
99	START	A		Start Code At Address = A

(Figure III-6)

```
================================================================================================================================
BOTTLEPLANT         CLR   R0             <PREPARE TO USE REGISTER #0>
                    CTIM      5          <CLEAR TIMER 5>
                    LDI   R0      30000  <LOAD THE VALUE 30000 MILLISECONDS INTO REGISTER #0>
                    RTIM      5 R0       <RESET TIMER #5 WITH 30000 MILLISECONDS>
                    BTIM  END            <IF CONTROL PANEL IS NOT TURNED ON OR CANNEL #4 IS NOT READY WITHIN ...>
                                         <30 SECONDS OF SPAWNING THIS TASK, THIS TASK WILL DIE>
START               CLR   R0             <PREPARE TO USE REGISTER #0>
                    TDIN      4          <TEST CHANNEL>
                    BNE   START          <LOOP UNTIL CHANNEL #4 READY OR TIMER #5 RUNS OUT>
                    DIN       4 R0       <LOAD THE VALUE OF THE ON/OFF SWITCH FROM THE CONTROL PANEL>
                    BEQ   START          <KEEP LOOPING UNTIL CONTROL PANEL TURNED ON OR TIMER #5 RUNS OUT>
                    CTIM      5          <CONTROL PANEL HAS BEEN TURNED ON; TURN OFF 30 SECOND TIMER>
                    <NOTE: ONCE THE MAIN PROGRAM LOOP HAS BEEN ENTERED, THE CONTROL PANEL ON/OFF SWITCH,IF SWITCHED OFF, WILL>
                    <STOP THE PROCESS BECAUSE  DIN CHANNEL #4 IS TAKEN IN AND TESTED EVERY MAIN PROGRAM LOOP (SEE BELOW)>
================================================================================================================================
                    <CONSTANT VALUES ARE TO BE STORED AT AN ADRESS APPROPRIATELY NAMED TO REPRESENT ITS CONTENT>
CONSTANTS           CLR   R0             <PREPARE TO USE REGISTER #0>
                    LDI   R0        140  <CONSTANT ASSIGNED BASED ON GIVEN RECIPE>
                    STO   R0    T_TRGTHT
                    LDI   R0        100  <CONSTANT ASSIGNED BASED ON GIVEN RECIPE>
                    STO   R0    T_TRGTCL
                    LDI   R0        110  <CONSTANT ASSIGNED BASED ON PHYSICAL TANK LIMITS>
                    STO   R0    VRCAP
                    LDI   R0       1000  <CONSTANT ASSIGNED BASED ON PHYSICAL TANK LIMITS>
                    STO   R0    VHCAP
                    LDI   R0          3  <CONSTANT ASSIGNED BASED ON PHYSICAL TANK LIMITS>
                    STO   R0    VFCAP
                    LDI   R0    0.09375  <CONSTANT ASSIGNED BASED ON PHYSICAL TANK LIMITS><12 OUNCES>
                    STO   R0    VBCAP
                    LDI   R0          2  <CONSTANT ASSIGNED TO PREVENT PUMPING BEYOND TANK LIMITS>
                    STO   R0    V1MIN
                    LDI   R0          2  <CONSTANT ASSIGNED TO PREVENT PUMPING BEYOND TANK LIMITS>
                    STO   R0    V2MIN
                    LDI   R0          2  <CONSTANT ASSIGNED TO PREVENT PUMPING BEYOND TANK LIMITS>
                    STO   R0    V3MIN
                    LDI   R0          2  <CONSTANT ASSIGNED TO PREVENT PUMPING BEYOND TANK LIMITS>
                    STO   R0    VRMIN
                    LDI   R0          2  <CONSTANT ASSIGNED TO PREVENT PUMPING BEYOND TANK LIMITS>
                    STO   R0    VHMIN
                    LDI   R0        0.1  <CONSTANT ASSIGNED TO PREVENT PUMPING BEYOND TANK LIMITS>
                    STO   R0    VFMIN
                    LDI   R0    0.28125  <CONSTANT ASSIGNED TO PREVENT PUMPING BEYOND TANK LIMITS>
                    STO   R0    VFLOW
                    LDI   R0        100  <CONSTANT ASSIGNED TO PREVENT PUMPING BEYOND TANK LIMITS>
                    STO   R0    VRMAX
                    LDI   R0        900  <CONSTANT ASSIGNED TO PREVENT PUMPING BEYOND TANK LIMITS>
                    STO   R0    VHMAX
                    LDI   R0        2.9  <CONSTANT ASSIGNED TO PREVENT PUMPING BEYOND TANK LIMITS>
                    STO   R0    VFMAX
                    LDI   R0        125  <CONSTANT ASSIGNED BASED ON RESULTS OF A PREVIOUSLY RUN SIMULATION PROGRAM>
                    STO   R0    P1
                    LDI   R0        250  <CONSTANT ASSIGNED BASED ON RESULTS OF A PREVIOUSLY RUN SIMULATION PROGRAM>
                    STO   R0    P2
                    LDI   R0        375  <CONSTANT ASSIGNED BASED ON RESULTS OF A PREVIOUSLY RUN SIMULATION PROGRAM>
                    STO   R0    P3
                    LDI   R0        500  <CONSTANT ASSIGNED BASED ON RESULTS OF A PREVIOUSLY RUN SIMULATION PROGRAM>
                    STO   R0    P4
                    LDI   R0        500  <CONSTANT ASSIGNED BASED ON RESULTS OF A PREVIOUSLY RUN SIMULATION PROGRAM>
                    STO   R0    P5
```

```
                LDI   R0        50  <CONSTANT ASSIGNED BASED ON RESULTS OF A PREVIOUSLY RUN SIMULATION PROGRAM>
                STO   R0    P6
                LDI   R0        20  <CONSTANT ASSIGNED BASED ON RESULTS OF A PREVIOUSLY RUN SIMULATION PROGRAM>
                STO   R0    P7
                LDI   R0       500  <CONSTANT ASSIGNED BASED ON RESULTS OF A PREVIOUSLY RUN SIMULATION PROGRAM>
                STO   R0    P8
                LDI   R0      0.03  <CONSTANT ASSIGNED BASED ON RESULTS OF A PREVIOUSLY RUN SIMULATION PROGRAM>
                STO   R0    HTR
                LDI   R0       180  <CONSTANT ASSIGNED BASED ON RESULTS OF A PREVIOUSLY RUN SIMULATION PROGRAM>
                STO   R0    TE
================================================================================================================================
INITIAL_CONDITIONS  CLR   R0             <PREPARE TO USE REGISTER #0>
                    CLR   R1             <PREPARE TO USE REGISTER #1>
                    CLR   R2             <PREPARE TO USE REGISTER #2>
                    CLR   R3             <PREPARE TO USE REGISTER #3>
                    CLR   R4             <PREPARE TO USE REGISTER #4>
                    CLR   R5             <PREPARE TO USE REGISTER #5>
                    --------------------
                    <THE FOLLOWING ARE STORED CONSECUTIVELY IN MEMORY: ANALOG-IN VALUES,DIGITAL-IN VALUES,FLAGS,DIGITAL-OUT VALUES>
                    <THIS SIMPLIFIES THE REQUIRED INITIAL ZEROING OF THESE VALUES>
                    <>
      CLEAR_MEM     LDI   R0         95  <REGISTER #0 WILL BE USED AS A COUNTER FOR THE 95 CONSEQUTIVE VALUES>
           CLR1     LDI   R1    FLOW1.ADX<LOAD THE ADRESS OF THE FIRST VALUE FROM TABLE OF 95 VALUES>
                    STX   R2    RI       <STORE A (0) AT THIS ADRESS>
                    DEC   R0             <DECRIMENT COUNTER>
                    BNE   CLR1           <IF ALL 95 NOT CLEARED YET, LOOP BACK AND CONTINUE CLEARING)
                    --------------------
                    STO   R3    BOTNUM   <SET THE NUMBER OF BOTTLES PRODUCED TO ZERO>
                    STO   R3    VB       <SET VOLUME OF THE BOTTLE TO ZERO; THIS IS VALID SINCE..>
                                         <PROGRAM WILL NOT ALLOW A PARTIALLY FULL BOTTLE TO REMAIN WHEN SHUTTING DOWN SYSTEM>
                    --------------------
      SET_STATE     LDI   R5       1001  <REACTOR STATE DEFINED BY CONTENTS OF REGISTER #5: ..>
                                         <FILL"=1001,"HEAT=1002","COOL"=1003,"DRAIN"=1004>
                    STO   R5    STATE    <ALWAYS SET STATE INITIALLY TO "FILL">
                                         <WHEN NOT IN CONTROL PARTS OF PROGRAM, REACTOR STATE STORED AT MEMORY ADRESS NAMED "STATE">
                    --------------------
                    <A REQUESTED VOLUME OF PRODUCT MUST BE PULSED TO ADC CH# 27 FROM THE CONTROL PANEL WITHIN >
                    <5 MINUTES OF TURNING ON THE CONTROL PANEL>
                    <IF THIS IS NOT DONE, THE TASK WILL DIE>
                    <>
                    CTIM     2           <PREPARE TO USE TIMER 2>
                    LDI   R4    300000   <LOAD 300,000 MILLISECONDS (5 MINUTES) INTO R4)
                    RTIM     2 R4        <RESET TIMER 2 TO 5 MINUTES>
      VREQIN        TADC    27           <TEST CHANNEL>
                    BNE   VREQIN         <KEEP LOOPING HERE UNTIL CHANNEL READY OR 5 MINUTES LAPSE>
                    ADC     27 R2        <CHANNEL READY; BRING IN REQUESTED VOLUME FROM CONTROL PANEL>
                    BEQ   VREQIN         <KEEP LOOPING HERE UNTIL A REQUESTED VOLUME (OTHER THAN ZERO) IS BROUGHT IN FROM >
                                         <THE CONTROL PANEL OR 5 MINUTES HAVE LAPSED>
                    STO   R2    VREQ     <A REQUESTED VOLUME (OTHER THAN ZERO) HAS BEEN ENTERED AT THE CONTROL PANEL; STORE IT IN MEM
================================================================================================================================
                    <MAIN LOOP COMES BACK TO HERE>
TOP                 CTIM     3           <PREPARE TO SET PROGRAM SCAN TIME= 10 MILLISECONDS>
                    LDI   R1        10   <PREPARE TO SET PROGRAM SCAN TIME= 10 MILLISECONDS>
                    RTIM     3 R1        <MAIN LOOP SHOULD FINISH IN LESS THAN 10 MILLISECONDS, THEN WAIT FOR TIMER 3 TO RUN OUT..>
                    BEQ   TOP            <...BEFORE STARTING AGAIN AT THE TOP>
                    --------------------
                    <UNLIKE OTHER VALUES BROUGHT IN FROM THE ADC AND DIN, THE REQUESTED VOLUME OF PRODUCT IS NOT STORED AT AN ...>
                    <ADRESS NAMED AS THE VARIABLE IT REPRESENTS (ie.:VREQ) EXCEPT FOR THE FIRST TIME IT IS BROUGHT IN DURING THE>
                    <INITIAL CONDITIONS; THE ADRESS WHICH RECEIVES THE VALUE OF (VREQ) DURING THE REPETITIVE SCANS IS NAMED "ACH27">
                    <VOLUME REQUESTES OF ZERO ARE IGNORED DURING THE MAIN LOOP, HOWEVER,....>
```

```
                <A VOLUME REQUEST PULSED IN DURIRNG THE MAIN LOOP WILL CAUSE THE SYSTEM TO SHUT DOWN IF THE REQUEST IS LESS...>
                <THAN THE AMOUNT OF VOLUME ALREADY PRODUCED TOWARDS THE LAST VOLUME REQUEST>
                <IF THE NEW VOLUME REQUEST IS GREATER THAN THIS, THE SYSTEM WILL CONTINUE PRODUCING UNTIL THE NEW VOLUME...>
                <IS COMPLETED (ie.: THE NEW VREQ AND OLD VREQ ARE NOT ADDITIVE DURING SUCCESIVE SCANS>
                <>
                LD    R0    ACH27     <LOAD THE VOLUME REQUEST VALUE JUST BROUGHT IN ON THE LAST SCAN>
                BEQ   VOLUME_DONE?    <IF IT IS ZERO, IGNORE IT; STAY IN THE MAIN LOOP>
                STO   R0    VREQ      <IF IT IS NOT ZERO, REPLACE THE VALUE OF VREQ WITH THIS NEW VALUE>
                --------------------
VOLUME_DONE?    LD    R0    VREQ      <PREPARE TO DEFINE THE REQUIRED NUMBER OF BOTTLES FROM THE VOLUME REQUESTED>
                LDI   R1    VBMAX     <PREPARE TO DEFINE THE REQUIRED NUMBER OF BOTTLES FROM THE VOLUME REQUESTED>
                DIV   R1    R0        <NUMBER OF BOTTLES REQUIRED= VREQ/VBMAX>
                STO   R0    BOTNUMREQ<STORE IT FOR LATER USE>
                --------------------
                LD    R0    VBMAX     <PREPARE TO DEFINE THE REQUIRED REMAINING VOLUME TO BE PRODUCED>
                LD    R1    BOTNUM    <PREPARE TO DEFINE THE REQUIRED REMAINING VOLUME TO BE PRODUCED>
                MULT  R1    R0        <VBMAX*BOTNUM>
                LD    R1    VREQ      <PREPARE TO DEFINE THE REQUIRED REMAINING VOLUME TO BE PRODUCED>
                SUB   R0    R1        <THE REQUIRED REMAINING VOLUME TO BE PRODUCED= [VREQ -(VBMAX*VBOTNUM)] >
                STO   R1    VTOGO     <STORE IT FOR LATER USE>
                --------------------
                LD    R0    VREQ      <PREPARE TO SEE IF VOLUME REQUESTED IS COMPLETED>
                LD    R1    VTOGO     <PREPARE TO SEE IF VOLUME REQUESTED IS COMPLETED>
                SUB   R1    R0        <IS VOLUME REQUESTED COMPLETED? ie.: IS (VREQ > VTOGO) >
                BEQ   ANALOG_IN       <NO!, BRANCH TO ANALOG_IN>
                BRA   OUT             <YES! , BRANCH OUT OF MAIN LOOP (VOLUME REQUESTED HAS BEEN PRODUCED)>
                                      <THIS DOES NOT KILL THE PROGRAM; THE CONTROL PANEL OPERATOR WILL HAVE 5 MINUTES TO .....>
                                      <ENTER A NEW REQUESTED VOLUME INTO THE CONTROL PANEL>
================================================================================================================================
                <THIS PROGRAM SEGMENT BRINGS IN ANALOG VALUES FROM 27 ADC CHANNELS>
ANALOG_IN       CLR R0                <PURGE REGISTER TO PREVENT ERRONEOUS READINGS>
                CLR R1                <PURGE REGISTER TO PREVENT ERRONEOUS READINGS>
                CLR R2                <PURGE REGISTER TO PREVENT ERRONEOUS READINGS>
                CLR R3                <PURGE REGISTER TO PREVENT ERRONEOUS READINGS>
                CLR R4                <PURGE REGISTER TO PREVENT ERRONEOUS READINGS>
                CLR R5                <PURGE REGISTER TO PREVENT ERRONEOUS READINGS>
                --------------------
                <THIS FOLLOWING SEGMENT SETS A MAXIMUM ALLOWED TIME FOR ALL ANALOG DATA TO BE TAKEN IN>
    AIN_TIM_MSTR CLTIM    1           <PREPARE TO USE THIS TIMER>
                LDI   R0      0.37    <SINCE EACH CHANNEL WILL ONLY BE ALLOWED 0.01 MILLISECONDS>
                RTIM    1 R0          <TOTAL TIME ALLOWED= (27*0.01)+(0.1 MILLISECONDS FOR OTHER INSTRUCTIONS)>
                BTIM1 DIGITAL_IN      <IF TIMER 2 RUNS OUT, ANALOG_IN HAS TAKEN TO LONG; SO JUMP OUT OF IT>
                BRA   ASTART
                ====================
                <THIS SUBROUTINE SETS A MAXIMUM ALLOWED TIME FOR ANY ONE ANALOG READING TO BE TAKEN IN>
                <THIS SUBROUTINE ALSO LOADS THE INCOMING DATA INTO CONSECUTIVE LOCATIONS IN MEMORY>
    ATIM&LD     CTIM    2             <DATA IS IN; CLEAR TIMER TO PREVENT UNWANTED BRANCHING>
                INC   R0              <INCRIMENT THE ADRESS OF WHERE TO STORE THE INCOMING DATA>
                STX   R1    R0        <STORE IT>
                CLR   R1              <PURGE REGISTER TO PREVENT ERRONEOUS READINGS>
    ATIM&LD1    LDI   R4        4     <THE ADRESS OF EACH PROGRAM SEGMENT TO BRING IN A MEASUREMENT ARE SEPERATED BY 4>
                ADD   R4    R3        <R3 NOW CONTAINS THE ADRESS TO JUMP TO IF TIMER 2 RUNS OUT>
                LDI   R2      0.01    <ONLY 0.01 MILLISECONDS TOTAL WILL BE ALLOWED FOR CHANNEL READY AND LOADING VALUE INTO R2>
                RTIM    2 R2          <SET TIMER >
                BTIM  ASKIP           <SKIP CHANNEL WHEN THIS TIMER RUNS OUT>
                BRA   ARETURN         <RETURN FROM SUBROUTINE>
        ASKIP   BRI   R3              <SKIP CHANNEL, TIMER 2 HAS RUN OUT>
        ARETURN RTN                   <RETURN FROM SUBROUTINE>
                ====================
    ASTART      LDI   R0    V1.ADX    <LOAD THE FIRST ADRESS OF THE TABLE WHERE THE INCOMING DATA WILL BE STORED>
```

```
        LD1   R3    AIN1     <LOAD THE ADRESS OF THE FIRST SEGMENT TO BRING IN DATA>
        CTIM      2          <CLEAR TIMER 2>
        JSR   ATIM&LD1
        -------------------
AIN1    TADC      1          <TEST IF CHANNEL READY>
        BNE   AIN1           <IF NOT, KEEP LOOPING> <TIMER 2 WILL STOP THIS LOOPING AFTER 0.01 MILLISECONDS>
        ADC       1 R1       <IF CHANNEL READY, BRING IN DATA>
        JSR   ATIM&LD        <EXECECUTE SUBROUTINE TO LOAD THE DATA IN THE TABLE IN MEMORY AND TO RESET THE SHORT TIMER>
        -------------------
AIN2    TADC      2          <TEST IF CHANNEL READY>
        BNE   AIN2           <IF NOT, KEEP LOOPING> <TIMER 2 WILL STOP THIS LOOPING AFTER 0.01 MILLISECONDS>
        ADC       2 R1       <IF CHANNEL READY, BRING IN DATA>
        JSR   TIM&LD         <EXECECUTE SUBROUTINE TO LOAD THE DATA IN THE TABLE IN MEMORY AND TO RESET THE SHORT TIMER>
        -------------------
AIN3    TADC      3          <TEST IF CHANNEL READY>
        BNE   AIN3           <IF NOT, KEEP LOOPING> <TIMER 2 WILL STOP THIS LOOPING AFTER 0.01 MILLISECONDS>
        ADC       3 R1       <IF CHANNEL READY, BRING IN DATA>
        JSR   TIM&LD         <EXECECUTE SUBROUTINE TO LOAD THE DATA IN THE TABLE IN MEMORY AND TO RESET THE SHORT TIMER>
        -------------------
AIN4    TADC      4          <TEST IF CHANNEL READY>
        BNE   AIN4           <IF NOT, KEEP LOOPING> <TIMER 2 WILL STOP THIS LOOPING AFTER 0.01 MILLISECONDS>
        ADC       4 R1       <IF CHANNEL READY, BRING IN DATA>
        JSR   TIM&LD         <EXECECUTE SUBROUTINE TO LOAD THE DATA IN THE TABLE IN MEMORY AND TO RESET THE SHORT TIMER>
        -------------------
AIN5    TADC      5          <TEST IF CHANNEL READY>
        BNE   AIN5           <IF NOT, KEEP LOOPING> <TIMER 2 WILL STOP THIS LOOPING AFTER 0.01 MILLISECONDS>
        ADC       5 R1       <IF CHANNEL READY, BRING IN DATA>
        JSR   TIM&LD         <EXECECUTE SUBROUTINE TO LOAD THE DATA IN THE TABLE IN MEMORY AND TO RESET THE SHORT TIMER>
        -------------------
AIN6    TADC      6          <TEST IF CHANNEL READY>
        BNE   AIN6           <TEST IF CHANNEL READY>
        ADC       6 R1       <IF NOT, KEEP LOOPING> <TIMER 2 WILL STOP THIS LOOPING AFTER 0.01 MILLISECONDS>
        JSR   TIM&LD         <IF CHANNEL READY, BRING IN DATA>
        -------------------  EXECECUTE SUBROUTINE TO LOAD THE DATA IN THE TABLE IN MEMORY AND TO RESET THE SHORT TIMER>
AIN7    TADC      7          <
        BNE   AIN7           <TEST IF CHANNEL READY>
        ADC       7 R1       <IF NOT, KEEP LOOPING> <TIMER 2 WILL STOP THIS LOOPING AFTER 0.01 MILLISECONDS>
        JSR   TIM&LD         <IF CHANNEL READY, BRING IN DATA>
        -------------------  EXECECUTE SUBROUTINE TO LOAD THE DATA IN THE TABLE IN MEMORY AND TO RESET THE SHORT TIMER>
AIN8    TADC      8          <
        BNE   AIN8           <TEST IF CHANNEL READY>
        ADC       8 R1       <IF NOT, KEEP LOOPING> <TIMER 2 WILL STOP THIS LOOPING AFTER 0.01 MILLISECONDS>
        JSR   TIM&LD         <IF CHANNEL READY, BRING IN DATA>
        -------------------  EXECECUTE SUBROUTINE TO LOAD THE DATA IN THE TABLE IN MEMORY AND TO RESET THE SHORT TIMER>
AIN9    TADC      9          <
        BNE   AIN9           <TEST IF CHANNEL READY>
        ADC       9 R1       <IF NOT, KEEP LOOPING> <TIMER 2 WILL STOP THIS LOOPING AFTER 0.01 MILLISECONDS>
        JSR   TIM&LD         <IF CHANNEL READY, BRING IN DATA>
        -------------------  EXECECUTE SUBROUTINE TO LOAD THE DATA IN THE TABLE IN MEMORY AND TO RESET THE SHORT TIMER>
AIN10   TADC     10          <
        BNE   AIN10          <TEST IF CHANNEL READY>
        ADC      10 R1       <IF CHANNEL READY, BRING IN DATA>
        JSR   TIM&LD         <EXECECUTE SUBROUTINE TO LOAD THE DATA IN THE TABLE IN MEMORY AND TO RESET THE SHORT TIMER>
        -------------------
AIN11   TADC     11          <TEST IF CHANNEL READY>
        BNE   AIN11          <IF NOT, KEEP LOOPING> <TIMER 2 WILL STOP THIS LOOPING AFTER 0.01 MILLISECONDS>
        ADC      11 R1       <IF CHANNEL READY, BRING IN DATA>
        JSR   TIM&LD         <EXECECUTE SUBROUTINE TO LOAD THE DATA IN THE TABLE IN MEMORY AND TO RESET THE SHORT TIMER>
        -------------------
AIN12   TADC     12          <TEST IF CHANNEL READY>
```

```
        BNE   AIN12          <IF NOT, KEEP LOOPING> <TIMER 2 WILL STOP THIS LOOPING AFTER 0.01 MILLISECONDS>
        ADC     12 R1        <IF CHANNEL READY, BRING IN DATA>
        JSR   TIM&LD         <EXECECUTE SUBROUTINE TO LOAD THE DATA IN THE TABLE IN MEMORY AND TO RESET THE SHORT TIMER>
        -------------------
AIN13   TADC    13           <TEST IF CHANNEL READY>
        BNE   AIN13          <IF NOT, KEEP LOOPING> <TIMER 2 WILL STOP THIS LOOPING AFTER 0.01 MILLISECONDS>
        ADC     13 R1        <IF CHANNEL READY, BRING IN DATA>
        JSR   TIM&LD         <EXECECUTE SUBROUTINE TO LOAD THE DATA IN THE TABLE IN MEMORY AND TO RESET THE SHORT TIMER>
        -------------------
AIN14   TADC    14           <TEST IF CHANNEL READY>
        BNE   AIN14          <IF NOT, KEEP LOOPING> <TIMER 2 WILL STOP THIS LOOPING AFTER 0.01 MILLISECONDS>
        ADC     14 R1        <IF CHANNEL READY, BRING IN DATA>
        JSR   TIM&LD         <EXECECUTE SUBROUTINE TO LOAD THE DATA IN THE TABLE IN MEMORY AND TO RESET THE SHORT TIMER>
        -------------------
AIN15   TADC    15           <TEST IF CHANNEL READY>
        BNE   AIN15          <IF NOT, KEEP LOOPING> <TIMER 2 WILL STOP THIS LOOPING AFTER 0.01 MILLISECONDS>
        ADC     15 R1        <IF CHANNEL READY, BRING IN DATA>
        JSR   TIM&LD         <EXECECUTE SUBROUTINE TO LOAD THE DATA IN THE TABLE IN MEMORY AND TO RESET THE SHORT TIMER>
        -------------------
AIN16   TADC    16           <TEST IF CHANNEL READY>
        BNE   AIN16          <TEST IF CHANNEL READY>
        ADC     16 R1        <IF NOT, KEEP LOOPING> <TIMER 2 WILL STOP THIS LOOPING AFTER 0.01 MILLISECONDS>
        JSR   TIM&LD         <IF CHANNEL READY, BRING IN DATA>
        -------------------  EXECECUTE SUBROUTINE TO LOAD THE DATA IN THE TABLE IN MEMORY AND TO RESET THE SHORT TIMER>
AIN17   TADC    17           <
        BNE   AIN17          <TEST IF CHANNEL READY>
        ADC     17 R1        <IF NOT, KEEP LOOPING> <TIMER 2 WILL STOP THIS LOOPING AFTER 0.01 MILLISECONDS>
        JSR   TIM&LD         <IF CHANNEL READY, BRING IN DATA>
        -------------------  EXECECUTE SUBROUTINE TO LOAD THE DATA IN THE TABLE IN MEMORY AND TO RESET THE SHORT TIMER>
AIN18   TADC    18           <
        BNE   AIN18          <TEST IF CHANNEL READY>
        ADC     18 R1        <IF NOT, KEEP LOOPING> <TIMER 2 WILL STOP THIS LOOPING AFTER 0.01 MILLISECONDS>
        JSR   TIM&LD         <IF CHANNEL READY, BRING IN DATA>
        -------------------  EXECECUTE SUBROUTINE TO LOAD THE DATA IN THE TABLE IN MEMORY AND TO RESET THE SHORT TIMER>
AIN19   TADC    19           <
        BNE   AIN19          <TEST IF CHANNEL READY>
        ADC     19 R1        <IF NOT, KEEP LOOPING> <TIMER 2 WILL STOP THIS LOOPING AFTER 0.01 MILLISECONDS>
        JSR   TIM&LD         <IF CHANNEL READY, BRING IN DATA>
        -------------------  EXECECUTE SUBROUTINE TO LOAD THE DATA IN THE TABLE IN MEMORY AND TO RESET THE SHORT TIMER>
AIN20   TADC    20           <
        BNE   AIN20          <TEST IF CHANNEL READY>
        ADC     20 R1        <IF NOT, KEEP LOOPING> <TIMER 2 WILL STOP THIS LOOPING AFTER 0.01 MILLISECONDS>
        JSR   TIM&LD         <IF CHANNEL READY, BRING IN DATA>
        -------------------  EXECECUTE SUBROUTINE TO LOAD THE DATA IN THE TABLE IN MEMORY AND TO RESET THE SHORT TIMER>
AIN21   TADC    21           <
        BNE   AIN21          <TEST IF CHANNEL READY>
        ADC     21 R1        <IF NOT, KEEP LOOPING> <TIMER 2 WILL STOP THIS LOOPING AFTER 0.01 MILLISECONDS>
        JSR   TIM&LD         <IF CHANNEL READY, BRING IN DATA>
        -------------------  EXECECUTE SUBROUTINE TO LOAD THE DATA IN THE TABLE IN MEMORY AND TO RESET THE SHORT TIMER>
AIN22   TADC    22           <
        BNE   AIN22          <TEST IF CHANNEL READY>
        ADC     22 R1        <IF NOT, KEEP LOOPING> <TIMER 2 WILL STOP THIS LOOPING AFTER 0.01 MILLISECONDS>
        JSR   TIM&LD         <IF CHANNEL READY, BRING IN DATA>
        -------------------  EXECECUTE SUBROUTINE TO LOAD THE DATA IN THE TABLE IN MEMORY AND TO RESET THE SHORT TIMER>
AIN23   TADC    23           <
        BNE   AIN23          <TEST IF CHANNEL READY>
        ADC     23 R1        <IF NOT, KEEP LOOPING> <TIMER 2 WILL STOP THIS LOOPING AFTER 0.01 MILLISECONDS>
        JSR   TIM&LD         <IF CHANNEL READY, BRING IN DATA>
        -------------------  EXECECUTE SUBROUTINE TO LOAD THE DATA IN THE TABLE IN MEMORY AND TO RESET THE SHORT TIMER>
AIN24   TADC    24           <
```

```
                BNE   AIN24          <TEST IF CHANNEL READY>
                ADC      24 R1       <TEST IF CHANNEL READY>
                JSR   TIM&LD         <IF NOT, KEEP LOOPING> <TIMER 2 WILL STOP THIS LOOPING AFTER 0.01 MILLISECONDS>
                -------------------  IF CHANNEL READY, BRING IN DATA>
        AIN25   TADC     25          <EXECECUTE SUBROUTINE TO LOAD THE DATA IN THE TABLE IN MEMORY AND TO RESET THE SHORT TIMER>
                BNE   AIN25          <
                ADC      25 R1       <TEST IF CHANNEL READY>
                JSR   TIM&LD         <IF NOT, KEEP LOOPING> <TIMER 2 WILL STOP THIS LOOPING AFTER 0.01 MILLISECONDS>
                -------------------  IF CHANNEL READY, BRING IN DATA>
        AIN26   TADC     26          <EXECECUTE SUBROUTINE TO LOAD THE DATA IN THE TABLE IN MEMORY AND TO RESET THE SHORT TIMER>
                BNE   AIN26          <
                ADC      26 R1       <TEST IF CHANNEL READY>
                JSR   TIM&LD         <IF NOT, KEEP LOOPING> <TIMER 2 WILL STOP THIS LOOPING AFTER 0.01 MILLISECONDS>
                -------------------  IF CHANNEL READY, BRING IN DATA>
        AIN27   TADC     27          <EXECECUTE SUBROUTINE TO LOAD THE DATA IN THE TABLE IN MEMORY AND TO RESET THE SHORT TIMER>
                BNE   AIN27          <
                ADC      27 R1       <IF CHANNEL READY, BRING IN DATA>
                JSR   TIM&LD         <EXECECUTE SUBROUTINE TO LOAD THE DATA IN THE TABLE IN MEMORY AND TO RESET THE SHORT TIMER>
                -------------------
                CTIM     1           <CLEAR TIMER TO PREVENT UNWANTED BRANCHING>
                CTIM     2           <CLEAR TIMER TO PREVENT UNWANTED BRANCHING>
================================================================================================================================================
                <THIS PROGRAM SEGMENT BRINGS IN DIGITAL VALUES FROM 4 DIN CHANNELS>
DIGITAL_IN      CLR RO               <PURGE REGISTER TO PREVENT ERRONEOUS READINGS>
                CLR R1               <PURGE REGISTER TO PREVENT ERRONEOUS READINGS>
                CLR R2               <PURGE REGISTER TO PREVENT ERRONEOUS READINGS>
                CLR R3               <PURGE REGISTER TO PREVENT ERRONEOUS READINGS>
                CLR R4               <PURGE REGISTER TO PREVENT ERRONEOUS READINGS>
                CLR R5               <PURGE REGISTER TO PREVENT ERRONEOUS READINGS>
                -------------------
                <THIS FOLLOWING SEGMENT SETS A MAXIMUM ALLOWED TIME FOR ALL DIGITAL DATA TO BE TAKEN IN>
    DIN_TIM_MSTR CLTIM     1          <PREPARE TO USE THIS TIMER>
                LDI   RO       0.06  <SINCE EACH CHANNEL WILL ONLY BE ALLOWED 0.01 MILLISECONDS>
                RTIM     1 RO        <TOTAL TIME ALLOWED= (4*0.01)+(.02 MILLISECONDS FOR OTHER INSTRUCTIONS)>
                BTIM1 ONOFFCHECK     <IF TIMER 2 RUNS OUT, DIGITAL_IN HAS TAKEN TO LONG; SO JUMP OUT OF IT>
                BRA   DSTART
                ====================
                <THIS SUBROUTINE SETS A MAXIMUM ALLOWED TIME FOR ANY ONE READING TO BE TAKEN IN>
                <THIS SUBROUTINE ALSO LOADS THE INCOMING DATA INTO CONSECUTIVE LOCATIONS IN MEMORY>
    DTIM&LD     CTIM     2           <DATA IS IN; CLEAR TIMER TO PREVENT UNWANTED BRANCHING>
                INC   RO             <INCRIMENT THE ADRESS OF WHERE TO STORE THE INCOMING DATA>
                STX   R1    RO       <STORE IT>
                CLR   R1             <PURGE REGISTER TO PREVENT ERRONEOUS READINGS>
    DTIM&LD1    LDI   R4          4  <THE ADRESS OF EACH DATA SEGMENT TO BRING IN A MEASUREMENT ARE SEPERATED BY 4>
                ADD   R4    R3       <R3 NOW CONTAINS THE ADRESS TO JUMP TO IF TIMER 2 RUNS OUT>
                CLTIM    2           <PREPARE TO USE THIS TIMER>
                LDI   R2       0.01  <ONLY 0.01 MILLISECONDS TOTAL WILL BE ALLOWED FOR CHANNEL READY AND LOADING VALUE INTO R2>
                RTIM     2 R2        <SET TIMER >
                BTIM  DSKIP          <SKIP CHANNEL WHEN THIS TIMER RUNS OUT>
                BRA   DRETURN        <RETURN FROM SUBROUTINE>
        DSKIP   BRI   R3             <SKIP CHANNEL, TIMER HAS RUN OUT>
        DRETURN RTN                  <RETURN FROM SUBROUTINE>
                ====================
    DSTART      LDI   RO    HTRON_ADX<LOAD THE FIRST ADRESS OF THE TABLE WHERE THE INCOMING DATA WILL BE STORED>
                LD1   R3    DIN1     <LOAD THE ADRESS OF THE FIRST SEGMENT TO BRING IN DATA>
                CTIM     2           <CLEAR TIMER 2>
                JSR   ATIM&LD1
                -------------------
        DIN1    TDIN     1           <TEST IF CHANNEL READY>
                BNE   DIN1           <IF NOT, KEEP LOOPING> <TIMER 2 WILL STOP THIS LOOPING AFTER 0.01 MILLISECONDS>
```

```
                ADC       1 R1        <IF CHANNEL READY, BRING IN DATA>
                JSR   DTIM&LD         <EXECECUTE SUBROUTINE TO LOAD THE DATA IN THE TABLE IN MEMORY AND TO RESET THE SHORT TIMER>
                --------------------
        DIN2    TADC      2           <TEST IF CHANNEL READY>
                BNE   DIN2            <IF NOT, KEEP LOOPING> <TIMER 2 WILL STOP THIS LOOPING AFTER 0.01 MILLISECONDS>
                ADC       2 R1        <IF CHANNEL READY, BRING IN DATA>
                JSR   TIM&LD          <EXECECUTE SUBROUTINE TO LOAD THE DATA IN THE TABLE IN MEMORY AND TO RESET THE SHORT TIMER>
                --------------------
        DIN3    TADC      3           <TEST IF CHANNEL READY>
                BNE   DIN3            <IF NOT, KEEP LOOPING> <TIMER 2 WILL STOP THIS LOOPING AFTER 0.01 MILLISECONDS>
                ADC       3 R1        <IF CHANNEL READY, BRING IN DATA>
                JSR   TIM&LD          <EXECECUTE SUBROUTINE TO LOAD THE DATA IN THE TABLE IN MEMORY AND TO RESET THE SHORT TIMER>
                --------------------
        DIN4    TADC      4           <TEST IF CHANNEL READY>
                BNE   DIN4            <IF NOT, KEEP LOOPING> <TIMER 2 WILL STOP THIS LOOPING AFTER 0.01 MILLISECONDS>
                ADC       4 R1        <IF CHANNEL READY, BRING IN DATA>
                JSR   TIM&LD          <EXECECUTE SUBROUTINE TO LOAD THE DATA IN THE TABLE IN MEMORY AND TO RESET THE SHORT TIMER>
                --------------------
                CTIM      1           <CLEAR TIMER TO PREVENT UNWANTED BRANCHING>
                CTIM      2           <CLEAR TIMER TO PREVENT UNWANTED BRANCHING>
                ====================
    ONOFFCHECK  CLR   R0              <PURGE REGISTER: CRITICAL DECISION ABOUT TO BE MADE WITH THIS REGISTER>
                LD    R0    POWERON   <LOAD THE VALUE JUST BROUGHT IN ON CHANNEL# 4; THIS IS THE ON/OFF SWITCH ON THE CONTROL PANE
                BNE   STAYON          <IF SWITCH HAS NOT BEEN TURNED OFF,JUMP TO STAYON>
                CLR   R1              <ELSE, SHUT DOWN SYSTEM>
                INC   R1
                STO   R1    DF11      <SET DF11=1><THIS WILL SHUT DOWN SYSTEM DURING DISASTER CONTROL>
        STAYON  NOP                   <CONTINUE>
================================================================================================================================================
                <THE FOLLOWING PROGRAM SEGMENT TESTS FOR BURNED-OUT CONTROL LAMPS OR A BROKEN WIRE TO CONTROL LAMP>
                <APPROPRIATE LAMPS ON THE CONTROL PANEL WILL BE LIT TO INDICATE WHICH CONTROL CIRCUIT IS FUALTY>
TEST_LAMP_CIRC  LDI   R0    LMPIN_ADX  <LOAD THE ADRESS OF THE FIRST CONTOL LAMP FEEDBACK VALUE FROM A TABLE OF 10 VALUES>
                LDI   R1    U1_ADX     <LOAD THE ADRESS OF THE FIRST CONTOL VALUE FROM A TABLE OF 10 VALUES>
                LDI   R2    OCLMP_ADX  <LOAD THE ADRESS OF THE FIRST OPEN CIRCUIT INDICATOR LAMP VALUE FROM A TABLE OF 10 VALUES>
                LD    R5          10   <USE R5 AS A COUNTER><EACH TABLE HAS 10 CONSEQUTIVE ENTRIES>
      TLC1      LDX   R4    R1         <LOAD CONTENTS OF TABLE ADRESS>
                BEQ   LAMPOK           <CONTOL SIGNAL=0 WAS PREVIOUSLY SENT, THEREFOR MUST ASSUME LAMP CIRCUIT OK>
                LDX   R3    R0         <LAST U=1, CHECK LAMP CIRCUIT ie.:LOAD LAMP FEEDBACK VALUE>
                DEC   R5               <ANALOG READINGS OF LESS THAN (1 volt) ARE CONSIDERED AS ZERO>
                BPL   LAMPOK           <IF LAMP FEEDBACK > 1volt, LAMP CIRCUIT OK>
                CLR   R3               <ELSE, LAMP IS BURNED OUT OR WIRE IS BROKEN>
                INC   R3               <PREPARE TO TURN ON "BURNT OR WIRE BROKEN LAMP"
                STX   R3    R2         <SET VALUE AT OCLMP_ADX TO 1, NEXT DOUT WILL LITE THIS LAMP>
                BRA   TLC2             <PREPARE TO SEE IF ALL LAMPS HAVE BEEN TESTED>
                --------------------
      LAMPOK    CLR   R3               <LAMP AND WIRE TO IT ARE OK>
                STX   R3    R2         <SET VALUE AT OCLMP_ADX TO 0, NEXT DOUT WILL NOT LITE THIS LAMP>
                --------------------
      TLC2      DEC   R5               <DECRIMENT COUNTER>
                BEQ   DISASTER_FLAGS   <IF ALL LAMPS TESTED, JUMP TO NEXT PROGRAM SEGMENT>
                INC   R1               <POINT TO NEXT ADRESS IN TABLE>
                INC   R2               <POINT TO NEXT ADRESS IN TABLE>
                INC   R3               <POINT TO NEXT ADRESS IN TABLE>
                BRA   TLC1             <TEST NEXT CIRCUIT>
================================================================================================================================================
                <THE FOLLOWING PROGRAM SEGMENT TESTS EQUIPMENT STUCK-ON OR STUCK OFF>
                <APPROPRIATE LAMPS ON THE CONTROL PANEL WILL BE LIT TO INDICATE THESE DISASTERS>
                <APPROPRIATE DISASTER FLAGS WILL BE SET TO INDICATE THESE DISASTERS>
DISASTER_FLAGS  LDI   R0    U1_ADX     <LOAD THE ADRESS OF THE FIRST CONTOL VALUE FROM A TABLE OF 10 VALUES>
                LDI   R1    FLOW1.ADX  <LOAD THE ADRESS OF THE FIRST MEASURED FLOW VALUE FROM A TABLE OF 10 VALUES>
```

```
                LDI   R2    1STKON.ADX <LOAD THE ADRESS OF THE FIRST STUCK-ON INDICATOR LAMP VALUE FROM A TABLE OF 10 VALUES>
                LDI   R3    1STKOFF.AX <LOAD THE ADRESS OF THE FIRST STUCK-OFF INDICATOR LAMP VALUE FROM A TABLE OF 10 VALUES>
                LD    R4         10  <USE R4 AS A COUNTER><EACH TABLE HAS 10 CONSEQUTIVE ENTRIES>
DCHECK          LDX   R5    R0       <LOAD CONTENTS OF TABLE ADRESS>
                BEQ   DCHECKLOW      <DID LAST U=0?>  <IF YES, JUMP TO DCHECKLOW>
                LDX   R5    R1       <IF NO, WAS LAST RESPONSE > 0?>
                DEC   R5             <ANALOG READINGS OF LESS THAN (1 volt) ARE CONSIDERED AS ZERO>
                BPL   DCHECKDONE     <IF LAST CONTROL SENT=1 AND LAST RESPONSE > 0, NO DISASTER HERE><GO TO NEXT CHECK>
                CLR   R5             <ELSE, EQUIPMENT IS STUCK-OFF>
                INC   R5             <PREPARE TO TURN ON "STUCK-OFF LAMP"
                STX   R5    R3       <SET VALUE AT 1STKOFF.AX TO 1, NEXT DOUT WILL LITE THIS LAMP>
                BRA   DCHECKDONE     <PREPARE TO SEE IF ALL RESPONSES HAVE BEEN TESTED>
                --------------------
DCHECKLOW       LDX   R5    R1       <LAST U DID (=0),
                DEC   R5             <ANALOG READINGS OF LESS THAN (1 volt) ARE CONSIDERED AS ZERO>
                BEQ   DCHECKDONE     <IF LAST CONTROL SENT=0 AND LAST RESPONSE=  0, NO DISASTER HERE><GO TO NEXT CHECK>
                CLR   R5             <ELSE, EQUIPMENT IS STUCK-ON>
                INC   R5             <PREPARE TO TURN ON "STUCK-ON LAMP"
                STX   R5    R2       <SET VALUE AT 1STKON.ADX TO 1, NEXT DOUT WILL LITE THIS LAMP>
                --------------------
DCHECKDONE      DEC   R4             <DECRIMENT COUNTER>
                BEQ   DFLAGTOP       <IF ALL RESPONSES TESTED, JUMP TO NEXT PROGRAM SEGMENT><ELSE,.......>
                INC   R1             <POINT TO NEXT ADRESS IN TABLE>
                INC   R1             <POINT TO NEXT ADRESS IN TABLE>
                INC   R2             <POINT TO NEXT ADRESS IN TABLE>
                INC   R3             <POINT TO NEXT ADRESS IN TABLE>
                BRA   DCHECK         <TEST NEXT RESPONSE>
                ====================
                <THE DISASTER FLAGS COULD NOT BE SET ABOVE BECAUSE ALL SIX REGISTERS WERE BEING USED>
                <THE VALUES THAT THE DISASTER FLAGS TAKE ON ARE IDENTICAL TO THE VALUES THE DISASTER LAMPS TAKE ON..>
DFLAGTOP        LDI   R0    U1_ADX     <LOAD THE ADRESS OF THE FIRST CONTOL VALUE FROM A TABLE OF 10 VALUES>
                LDI   R1    FLOW1.ADX  <LOAD THE ADRESS OF THE FIRST MEASURED FLOW VALUE FROM A TABLE OF 10 VALUES>
                LDI   R2    DF11.ADX   <LOAD THE ADRESS OF THE ELEVENTH DISASTER FLAG VALUE FROM A TABLE OF 10 VALUES>
                LDI   R3    DF1.ADX    <LOAD THE ADRESS OF THE FIRST DISASTER FLAG VALUE FROM A TABLE OF 10 VALUES>
                LD    R4         10  <USE R4 AS A COUNTER><EACH TABLE HAS 10 CONSEQUTIVE ENTRIES>
DFLAG           LDX   R5    R1       <LOAD CONTENTS OF TABLE ADRESS>
                BEQ   DFLAGLOW       <DID LAST U=0?><IF YES, JUMP TO DFLAGLOW>
                LDX   R5    R1       <IF NO, WAS LAST RESPONSE > 0 ?>
                DEC   R5             <ANALOG READINGS OF LESS THAN (1 volt) ARE CONSIDERED AS ZERO>
                BPL   DCHECKDONE2    <IF LAST CONTROL SENT=1 AND LAST RESPONSE > 0, NO DISASTER HERE><GO TO NEXT CHECK>
                CLR   R5             <ELSE, EQUIPMENT IS STUCK-OFF>
                INC   R5             <PREPARE TO SET DISASTER FLAG
                STX   R5    R3       <SET DF=1
                BRA   DCHECKDONE2    <PREPARE TO SEE IF ALL RESPONSES HAVE BEEN TESTED>
                --------------------
DFLAGLOW        LDX   R5    R1       <LAST U DID (=0),
                BPL   DCHECKDONE2    <IF LAST CONTROL SENT= LAST RESPONSE= 0, NO DISASTER HERE><GO TO NEXT CHECK>
                CLR   R5             <ELSE, EQUIPMENT IS STUCK-ON>
                INC   R5             <PREPARE TO SET DISASTER FLAG
                STX   R5    R2       <SET DF=1
                --------------------
DCHECKDONE2     DEC   R4             <DECRIMENT COUNTER>
                BEQ   NORMAL_CONTOL  <IF ALL RESPONSES TESTED, JUMP TO NEXT PROGRAM SEGMENT><ELSE,.......>
                INC   R1             <POINT TO NEXT ADRESS IN TABLE>
                INC   R1             <POINT TO NEXT ADRESS IN TABLE>
                INC   R2             <POINT TO NEXT ADRESS IN TABLE>
                INC   R3             <POINT TO NEXT ADRESS IN TABLE>
                BRA   DFLAG          <TEST NEXT RESPONSE>
================================================================================================================================
```

```
                    <THIS PROGRAM SEGMENT WILL CALCULATE NEW CONTROL SIGNALS>
NORMAL_CONTROL      CLR   R0              <PREPARE TO USE THIS REGISTER>
                    CLR   R1              <PREPARE TO USE THIS REGISTER>
                    CLR   R2              <PREPARE TO USE THIS REGISTER>
                    CLR   R3              <PREPARE TO USE THIS REGISTER: THIS REGISTER WILL REMAIN CLEARED THROUGHOUT CONTROL>
                    CLR   R4              <PREPARE TO USE THIS REGISTER>
                    ====================
                    <CONTROL OF REACTOR BEGINS HERE> <ASSIGN NEW VALUES FOR U1,2,3,4,5,6,7,8,H,C)
      STATE_PICK    INC   R4              <PREPARE TO USE THIS REGISTER: THIS REGISTER WILL REMAIN (=1) THROUGHOUT CONTROL>
                    LD    R5    STATE     <PREPARE TO SEE WHICH REACTOR STATE IS PRESENTLY INDICATED BY (R5)>
                    LDI   R0      1001    <PREPARE TO SEE WHICH REACTOR STATE IS PRESENTLY INDICATED BY (R5)>
                    CMP   R5    R0        <DOES STATE= "FILL"?>
                    BEQ   FILL            <IF YES, BRANCH THERE; ELSE, CONTINUE>
                    LDI   R0      1002    <PREPARE TO SEE WHICH REACTOR STATE IS PRESENTLY INDICATED BY (R5)>
                    CMP   R5    R0        <DOES STATE= "HEAT"?>
                    BEQ   HEAT            <IF YES, BRANCH THERE; ELSE, CONTINUE>
                    LDI   R0      1003    <PREPARE TO SEE WHICH REACTOR STATE IS PRESENTLY INDICATED BY (R5)>
                    CMP   R5    R0        <DOES STATE= "COOL"?>
                    BEQ   COOL            <IF YES, BRANCH THERE; ELSE, CONTINUE>
                    LDI   R0      1004    <PREPARE TO SEE WHICH REACTOR STATE IS PRESENTLY INDICATED BY (R5)>
                    CMP   R5    R0        <DOES STATE= "DRAIN"?>
                    BEQ   DRAIN           <IF YES, BRANCH THERE; ELSE, CONTINUE>
                    --------------------
                    --------------------
      FILL          LD    R0    FLAG1     <PRESENT REACTOR STATE= "FILL"
                    BEQ   FILL2           <IF (FLAG1)=0 ie.: NOT SET, GO TO NEXT DECISION BLOCK WITHIN FILL STATE BLOCK>
                    LDI   R5      1001    <ELSE, SET NEXT REACTOR STATE FOR NEXT MAIN PROGRAM LOOP= "FILL">
                    JSR   ZEROREACTOR     <AND ZERO ALL REACTER CONTROLS (U12345678H and C) BY EXECUTING SUBROUTINE NAMED ZEROREACTOR>
                    BRA   LOWER_CONTROL   <AND JUMP OUT OF REACTOR CONTROL>
                    --------------------
         FILL2      LD    R0    VTOGO     <NEXT DECISION BLOCK WITHIN FILL STATE BLOCK>
                    LD    R1    VH        <PREPARE TO EVALUATE CONDITIONAL STATEMENT, LOAD VALUE FROM MEM ADX>
                    SUB   R1    R0        <EVALUATE CONDITIONAL STATEMENT: IS (VTOGO > VH)? >
                    BEQ   FILL2A          <NO! ,THEY ARE EQUAL; BRANCH TO FILL2A>
                    BNG   FILL2A          <NO! ,THE REVERSE IS TRUE; BRANCH TO FILL2A>
                    BRA   FILL3           <YES! , GO TO NEXT DECISION BLOCK WITHIN FILL STATE BLOCK>
         FILL2A     JSR   ZEROREACTOR     <ZERO ALL REACTER CONTROLS>
                    STO   R4    FLAG2     <SET FLAG2 (=1)>
                    LDI   R5      1001    <SET NEXT REACTOR STATE FOR NEXT MAIN PROGRAM LOOP= "FILL">
                    BRA   LOWER_CONTROL   <JUMP OUT OF REACTOR CONTROL>
                    --------------------
         FILL3      LD    R0    VHMAX     <NEXT DECISION BLOCK WITHIN FILL STATE BLOCK>
                    LD    R1    VH        <PREPARE TO EVALUATE CONDITIONAL STATEMENT, LOAD VALUE FROM MEM ADX>
                    SUB   R1    R0        <EVALUATE CONDITIONAL STATEMENT: IS (VHMAX > VH)? >
                    BEQ   FILL3A          <NO! ,THEY ARE EQUAL; BRANCH TO FILL3A>
                    BNG   FILL3A          <NO! ,THE REVERSE IS TRUE; BRANCH TO FILL3A>
                    BRA   FILL4           <YES! , GO TO NEXT DECISION BLOCK WITHIN FILL STATE BLOCK>
         FILL3A     JSR   ZEROREACTOR     <ZERO ALL REACTER CONTROLS>
                    LDI   R5      1001    <SET NEXT REACTOR STATE FOR NEXT MAIN PROGRAM LOOP= "FILL">
                    BRA   LOWER_CONTROL   <JUMP OUT OF REACTOR CONTROL>
                    --------------------
         FILL4      LD    R0    VRMAX     <NEXT DECISION BLOCK WITHIN FILL STATE BLOCK>
                    LD    R1    VR        <PREPARE TO EVALUATE CONDITIONAL STATEMENT, LOAD VALUE FROM MEM ADX>
                    SUB   R1    R0        <EVALUATE CONDITIONAL STATEMENT: IS (VRMAX > VR)? >
                    BEQ   FILL4A          <NO! ,THEY ARE EQUAL; BRANCH TO FILL4A>
                    BNG   FILL4A          <NO! ,THE REVERSE IS TRUE; BRANCH TO FILL4A>
                    BRA   FILL5           <YES! , GO TO NEXT DECISION BLOCK WITHIN FILL STATE BLOCK>
         FILL4A     JSR   ZEROREACTOR     <ZERO ALL REACTER CONTROLS>
                    STO   R4    UH        <SET UH= 1>
```

```
            LDI   R5       1002  <SET NEXT REACTOR STATE FOR NEXT MAIN PROGRAM LOOP= "HEAT">
            BRA   LOWER_CONTROL  <JUMP OUT OF REACTOR CONTROL>
            --------------------
     FILL5  LD    R0    V1       <NEXT DECISION BLOCK WITHIN FILL STATE BLOCK>
            LD    R1    VRMAX    <PREPARE TO EVALUATE CONDITIONAL STATEMENT, LOAD VALUE FROM MEM ADX>
            LDI   R2        0.1 <PREPARE TO EVALUATE CONDITIONAL STATEMENT>
            MULT  R2    R1       <PREPARE TO EVALUATE CONDITIONAL STATEMENT>
            SUB   R1    R0       <EVALUATE CONDITIONAL STATEMENT: IS (V1 > .1(VRMAX))? >
            BEQ   FILL5A         <NO! ,THEY ARE EQUAL; BRANCH TO FILL5A>
            BNG   FILL5A         <NO! ,THE REVERSE IS TRUE; BRANCH TO FILL5A>
                                 <YES! , CONTINUE EVALUATING SAME DECISION BLOCK>
            LD    R0    V2       <PREPARE TO EVALUATE CONDITIONAL STATEMENT, LOAD VALUE FROM MEM ADX>
            LD    R1    VRMAX    <PREPARE TO EVALUATE CONDITIONAL STATEMENT, LOAD VALUE FROM MEM ADX>
            LDI   R2        0.2 <PREPARE TO EVALUATE CONDITIONAL STATEMENT>
            MULT  R2    R1       <PREPARE TO EVALUATE CONDITIONAL STATEMENT>
            SUB   R1    R0       <EVALUATE CONDITIONAL STATEMENT: IS (V2 > .2(VRMAX))? >
            BEQ   FILL5A         <NO! ,THEY ARE EQUAL; BRANCH TO FILL5A>
            BNG   FILL5A         <NO! ,THE REVERSE IS TRUE; BRANCH TO FILL5A>
                                 <YES! , CONTINUE EVALUATING SAME DECISION BLOCK>
            LD    R0    V3       <PREPARE TO EVALUATE CONDITIONAL STATEMENT, LOAD VALUE FROM MEM ADX>
            LD    R1    VRMAX    <PREPARE TO EVALUATE CONDITIONAL STATEMENT, LOAD VALUE FROM MEM ADX>
            LDI   R2        0.3 <PREPARE TO EVALUATE CONDITIONAL STATEMENT>
            MULT  R2    R1       <PREPARE TO EVALUATE CONDITIONAL STATEMENT>
            SUB   R1    R0       <EVALUATE CONDITIONAL STATEMENT: IS (V3 > .3(VRMAX))? >
            BEQ   FILL5A         <NO! ,THEY ARE EQUAL; BRANCH TO FILL5A>
            BNG   FILL5A         <NO! ,THE REVERSE IS TRUE; BRANCH TO FILL5A>
            STO   R4    U1       <YES! , SET U1=1>
            STO   R4    U2       <AND, SET U2=1>
            STO   R4    U3       <AND, SET U3=1>
            STO   R4    U4       <AND, SET U4=1>
            STO   R3    U5       <AND, SET U5=0>
            STO   R3    U8       <AND, SET U8=0>
            STO   R3    UH       <AND, SET UH=0>
            LDI   R5       1001  <SET NEXT REACTOR STATE FOR NEXT MAIN PROGRAM LOOP= "FILL">
            BRA   LOWER_CONTROL  <JUMP OUT OF REACTOR CONTROL>
     FILL5A JSR   ZEROREACTOR    <ZERO ALL REACTER CONTROLS>
            STO   R4    FLAG2     SET FLAG2 (=1)>
            LDI   R5       1001  <SET NEXT REACTOR STATE FOR NEXT MAIN PROGRAM LOOP= "FILL">
            BRA   LOWER_CONTROL  <JUMP OUT OF REACTOR CONTROL>
            --------------------
            --------------------
HEAT        LD    R0    T_TRGTHEA<PRESENT REACTOR STATE= "HEAT"
            LD    R1    TR       <PREPARE TO EVALUATE CONDITIONAL STATEMENT, LOAD VALUE FROM MEM ADX>
            SUB   R1    R0       <EVALUATE CONDITIONAL STATEMENT: IS (T_TRGTHEAT > TR)? >
            BEQ   HEATA          <NO! ,THEY ARE EQUAL; BRANCH TO HEATA>
            BNG   HEATA           NO! ,THE REVERSE IS TRUE; BRANCH TO HEATA>
            JSR   ZEROREACTOR    <YES! , ZERO ALL REACTER CONTROLS AND; >
            STO   R4    UH       <SET UH (=1)>
            LDI   R5       1002  <SET NEXT REACTOR STATE FOR NEXT MAIN PROGRAM LOOP= "HEAT">
            BRA   LOWER_CONTROL  <JUMP OUT OF REACTOR CONTROL>
     HEATA  JSR   ZEROREACTOR    <ZERO ALL REACTER CONTROLS AND; >
            LDI   R5       1003  <SET NEXT REACTOR STATE FOR NEXT MAIN PROGRAM LOOP= "COOL">
            BRA   LOWER_CONTROL  <JUMP OUT OF REACTOR CONTROL>
            --------------------
            --------------------
COOL        LD    R0    TR       <PRESENT REACTOR STATE= "COOL"
            LD    R1    T_TRGTCOO<PREPARE TO EVALUATE CONDITIONAL STATEMENT, LOAD VALUE FROM MEM ADX>
            SUB   R1    R0       <EVALUATE CONDITIONAL STATEMENT: IS (TR > T_TRGTCOOL)? >
            BEQ   COOLA          <NO! ,THEY ARE EQUAL; BRANCH TO COOLA>
            BNG   COOLA          <NO! ,THE REVERSE IS TRUE; BRANCH TO COOLA>
```

```
            JSR   ZEROREACTOR    <YES! , ZERO ALL REACTER CONTROLS AND; >
            LDI   R5       1003  <SET NEXT REACTOR STATE FOR NEXT MAIN PROGRAM LOOP= "COOL">
            BRA   LOWER_CONTROL  <JUMP OUT OF REACTOR CONTROL>
     COOLA  JSR   ZEROREACTOR    <ZERO ALL REACTER CONTROLS AND; >
            STO   R4    U5       <SET U5 (=1)>
            LDI   R5       1004  <SET NEXT REACTOR STATE FOR NEXT MAIN PROGRAM LOOP= "DRAIN">
            BRA   LOWER_CONTROL  <JUMP OUT OF REACTOR CONTROL>
            --------------------
            --------------------          ================================================
DRAIN       LD    R0    VR       <PRESENT REACTOR STATE= "HEAT"
            LD    R1    VRMIN    <PREPARE TO EVALUATE CONDITIONAL STATEMENT, LOAD VALUE FROM MEM ADX>
            SUB   R1    R0       <EVALUATE CONDITIONAL STATEMENT: IS (VR > VRMIN)? >
            BEQ   DRAINA         <NO! ,THEY ARE EQUAL; BRANCH TO DRAINA>
            BNG   DRAINA         <NO! ,THE REVERSE IS TRUE; BRANCH TO DRAINA>
            JSR   ZEROREACTOR    <YES! , ZERO ALL REACTER CONTROLS AND; >
            STO   R4    U5       <SET U5=1>
            LDI   R5       1004  <SET NEXT REACTOR STATE FOR NEXT MAIN PROGRAM LOOP= "DRAIN">
            BRA   LOWER_CONTROL  <JUMP OUT OF REACTOR CONTROL>
     DRAINA JSR   ZEROREACTOR    <ZERO ALL REACTER CONTROLS AND; >
            LDI   R2         22  <PREPARE TO SET TR TO AMBIENT TEMPERATURE (TR= 22 C)>
            STO   R2    TR       <SET TR TO AMBIENT TEMPERATURE (TR= 22 C)>
            LDI   R5       1004  <SET NEXT REACTOR STATE FOR NEXT MAIN PROGRAM LOOP= "DRAIN">
            BRA   LOWER_CONTROL  <JUMP OUT OF REACTOR CONTROL>
            --------------------
ZEROREACTOR STO   R3    U1       <SUBROUTINE TO ZERO ALL REACTOR RELATED CONTROLS>
            STO   R3    U2       <SET U2=1>
            STO   R3    U3       <SET U3=1>
            STO   R3    U4       <SET U4=1>
            STO   R3    U5       <SET U5=1>
            STO   R3    U8       <SET U8=1>
            STO   R3    UH       <SET UH=1>
            RTN
            ====================
            <THE FOLLOWING PART OF NORMAL CONTROL DETERMINES NEW VALUES FOR U6,U7, AND UC>
LOWERLOWER1 LD    R0    VH       <PREPARE TO EVALUATE CONDITIONAL STATEMENT, LOAD VALUE FROM MEM ADX>
            LD    R1    VHMIN    <PREPARE TO EVALUATE CONDITIONAL STATEMENT, LOAD VALUE FROM MEM ADX>
            SUB   R1    R0       <EVALUATE CONDITIONAL STATEMENT: IS (VH > VHMIN)? >
            BEQ   LOWER1A        <NO! ,THEY ARE EQUAL; BRANCH TO LOWER1A>
            BNG   LOWER1A        <NO! ,THE REVERSE IS TRUE; BRANCH TO LOWER1A>
                                 <YES! , CONTINUE EVALUATING SAME DECISION BLOCK>
            LD    R0    VF       <PREPARE TO EVALUATE CONDITIONAL STATEMENT, LOAD VALUE FROM MEM ADX>
            LD    R1    VFMAX    <PREPARE TO EVALUATE CONDITIONAL STATEMENT, LOAD VALUE FROM MEM ADX>
            SUB   R1    R0       <EVALUATE CONDITIONAL STATEMENT: IS (VF > VFMAX)? >
            BEQ   LOWER1A        <NO! ,THEY ARE EQUAL; BRANCH TO LOWER1A>
            BNG   LOWER1A        <NO! ,THE REVERSE IS TRUE; BRANCH TO LOWER1A>
                                 <YES! , CONTINUE EVALUATING SAME DECISION BLOCK>
            LD    FLAG4          <PREPARE TO EVALUATE CONDITIONAL STATEMENT, LOAD VALUE FROM MEM ADX>
            BNE   LOWER1A        <EVALUATE CONDITIONAL STATEMENT: IS (FLAG4)=1 ?><IF YES,JUMP TO LOWER1A>
                                 <NO! , CONTINUE EVALUATING SAME DECISION BLOCK>
            LD    FLAG3          <PREPARE TO EVALUATE CONDITIONAL STATEMENT, LOAD VALUE FROM MEM ADX>
            BNE   LOWER1A        <EVALUATE CONDITIONAL STATEMENT: IS (FLAG3)=1 ?><IF YES,JUMP TO LOWER1A>
                                 <NO! , CONTINUE EVALUATING SAME DECISION BLOCK>
            STO   R4    U6       <SET U6=1>
            BRA   LOWER2         <YES! , GO TO NEXT DECISION BLOCK>
    LOWER1A STO         U6       <SET U6=0>
            --------------------
    LOWER2  LD    FLAG4          <PREPARE TO EVALUATE CONDITIONAL STATEMENT, LOAD VALUE FROM MEM ADX>
            BNE   LOWER1A        <EVALUATE CONDITIONAL STATEMENT: IS (FLAG4)=1 ?><IF NO,JUMP TO LOWER2A>
                                 <YES! , CONTINUE EVALUATING SAME DECISION BLOCK>
            LD    FLAG3          <PREPARE TO EVALUATE CONDITIONAL STATEMENT, LOAD VALUE FROM MEM ADX>
```

```
         BNE   LOWER1A        <EVALUATE CONDITIONAL STATEMENT: IS (FLAG4)=1 ?><IF NO,JUMP TO LOWER2A>
                              <YES! , CONTINUE EVALUATING SAME DECISION BLOCK>
         BRA   LOWER3         <GO TO NEXT DECISION BLOCK>
LOWER2A  STO   R3    U7       <SET U7=0>
         STO   R3    UC       <SET UC=0>
         BRA   DISASTER_CONTROLNORMAL CONTROL FINISHED, JUMP TO DISASTER CONTROL TO SEE IF A DISASTER HAS HAPPENED>
         --------------------
         ====================
         <THE FOLLOWING PROGRAM SEGMENT CALCULATES THE VOLUME IN THE BOTTLE(VB)>
         LD    R0    U1OLD    <LOAD THE VALUE OF THE PUMP6 CONTROL SENT OUT AFTER THE PREVIOUS SCAN>
         LD    R1    P6       <LOAD PUMP RATE FOR PUMP#6>
         MULT  R1    R0       <PREPARE TO CALCULATE HOW MUCH WAS PUMPED INTO THE FILL TANK DURING THE PREVIOS SCAN>
         LDI   R1        0.1  <(U1*P6*.01) GALLONS WERE PUMPED INTO THE FILL TANK DURING THE PREVIOS SCAN>
         MULT  R0    R1       <(R1)= (U1*P6*.01) GALLONS PUMPED INTO THE FILL TANK DURING THE PREVIOS SCAN>
         --------------------
         LD    R0    VF       <LOAD THE RECENT MEASUREMENT OF THE VOLUME IN THE FILL TANK>
         LD    R2    VFOLD    <LOAD THE MEASUREMENT OF THE VOLUME IN THE FILL TANK FOR THE PREVIOUS SCAN>
         SUB   R2    R0       <(R0) NOW CONTAINS THE CHANGE OF THE FILL TANK VOLUME DURING THE LAST SCAN>
         --------------------
         SUB   R1    R0       <(VOLUME IN LESS THE VOLUME CHANGE)= VOLUME OUT>
                              <THEREFOR, THE VOLUME FROM THE FILL TANK INTO THE BOTTLE FOR THE LAST SCAN HAS BEEN ...>
                              <CALCULATED AND IS IN R0>
         --------------------
         LD    R1    VBOLD    <LOAD VOLUME IN BOTTLE DURING LAST SCAN>
         ADD   R1    R0       <(R0) NOW CONTAINS THE NEW VOLUME IN THE BOTTLE (VB)>
         STO   R0    VB       <STORE VB FOR USE DURING THIS SCAN>
         STO   R0    VBOLD    <STORE VB FOR USE DURING THE NEXT SCAN>
         --------------------
         LD    R0    VF       <LOAD THE RECENT MEASUREMENT OF THE VOLUME IN THE FILL TANK>
         STO   R0    VFOLD    <STORE VF FOR USE DURING THE NEXT SCAN AS VFOLD>
         LD    R0    U6       <LOAD THE RECENT VALUE OF U6 CALCULATED>
         STO   R0    U6OLD    <STORE U6 FOR USE DURING THE NEXT SCAN AS U6OLD>
         --------------------
         ====================
         BTIM      4 LOWER3   <IF TIMER 4 IS RUN OUT, JUMP TO LOWER3>
         BRA   BOTSHIFT       <IF TIMER 4 IS NOT RUN OUT,JUMP TO BOTSHIFT>  <WAIT FOR BOTTLE TO SHIFT>
         --------------------
LOWER3   LD    R0    VBMAX    <PREPARE TO EVALUATE CONDITIONAL STATEMENT, LOAD VALUE FROM MEM ADX>
         LD    R1    VB       <PREPARE TO EVALUATE CONDITIONAL STATEMENT, LOAD VALUE FROM MEM ADX>
         SUB   R1    R0       <EVALUATE CONDITIONAL STATEMENT: IS (VB > VBMAX)? >
         BEQ   LOWER4         <NO! ,THEY ARE EQUAL; BRANCH TO LOWER4>
         BNG   LOWER4         <NO! ,THE REVERSE IS TRUE; BRANCH TO LOWER4>
                              <YES! , CONTINUE EVALUATING SAME DECISION BLOCK>
         LD    R0    VF       <PREPARE TO EVALUATE CONDITIONAL STATEMENT, LOAD VALUE FROM MEM ADX>
         LD    R1    VFMIN    <PREPARE TO EVALUATE CONDITIONAL STATEMENT, LOAD VALUE FROM MEM ADX>
         SUB   R1    R0       <EVALUATE CONDITIONAL STATEMENT: IS (VF > VMIN)? >
         BEQ   LOWER4         <NO! ,THEY ARE EQUAL; BRANCH TO LOWER4>
         BNG   LOWER4         <NO! ,THE REVERSE IS TRUE; BRANCH TO LOWER4>
         STO   R3    U7       <SET U7=0>
         STO   R3    UC       <SET UC=0>
         BRA   DISASTER_CONTROL  <NORMAL CONTROL FINISHED, JUMP TO DISASTER CONTROL TO SEE IF A DISASTER HAS HAPPENED>
         --------------------
LOWER4   LD    R0    VB       <PREPARE TO EVALUATE CONDITIONAL STATEMENT, LOAD VALUE FROM MEM ADX>
         LD    R1    VBMAX    <PREPARE TO EVALUATE CONDITIONAL STATEMENT, LOAD VALUE FROM MEM ADX>
         SUB   R1    R0       <EVALUATE CONDITIONAL STATEMENT: IS (VBMAX > VB)? >
         BEQ   LOWER4B        <NO! ,THEY ARE EQUAL; BRANCH TO LOWER4A>
         BNG   LOWER4B        <NO! ,THE REVERSE IS TRUE; BRANCH TO LOWER4A>
                              <A BOTTLE HAS BEEN COMPLETED; THERFORE...>
         CTIM      4          <MUST PREPARE TO WAIT FOR BOTTLE TO BE SHIFTED>
         LDI   R0       1000  <BOTTLE WILL TAKE 1 SECOND (1000 MILLISECONDS) TO SHIFT>
```

```
                RTIM      4    1000  <BOTTLE WILL TAKE 1 SECOND (1000 MILLISECONDS) TO SHIFT>
    BOTSHIFT    STO   R3    U7       <SET U7=0>
                STO   R4    U7       <SET UC=1>  <SHIFT IN NEW BOTTLE>
                STO   R3    VB       <SET VB=0>
                LD    R0    BOTNUM   <PREPARE TO INCRIMENT THE BOTTLE COUNT>
                INC   R0             <INCRIMENT THE BOTTLE COUNT>
                STO   R0    BOTNUM   <STORE NEW BOTTLE COUNT>
                BRA   DISASTER_CONTROL  <NORMAL CONTROL FINISHED, JUMP TO DISASTER CONTROL TO SEE IF A DISASTER HAS HAPPENED>
        LOWER4A STO   R3    U7       <SET U7=0>
                STO   R3    UC       <SET UC=0>
                BRA   DISASTER_CONTROL  <NORMAL CONTROL FINISHED, JUMP TO DISASTER CONTROL TO SEE IF A DISASTER HAS HAPPENED>
                LD    DF1            <PREPARE TO EVALUATE CONDITIONAL STATEMENT, LOAD VALUE FROM MEM ADX>
================================================================================================================================
DISASTER_CONTROL    CLR   R0         <PREPARE TO USE THIS REGISTER>
                    CLR   R1         <PREPARE TO USE THIS REGISTER>
                    CLR   R2         <PREPARE TO USE THIS REGISTER>
                    CLR   R3         <PREPARE TO USE THIS REGISTER: THIS REGISTER WILL REMAIN CLEARED THROUGHOUT DISASTER CONTROL
                    CLR   R4         <PREPARE TO USE THIS REGISTER>
                    INC   R4         <PREPARE TO USE THIS REGISTER: THIS REGISTER WILL REMAIN=1 THROUGHOUT DISASTER CONTROL>
                    ====================
                    <THIS PART OF THE PROGRAM WILL OVERIDE CONTROL SIGNALS CREATED IN NORMAL CONTROL>
                    <THIS PROGRAM PRESENTLY CREATES A "GENTLE" SHUTDOWN ONLY FOR STUCK-OFF DISASTERS (DF1-10)>
                    <STUCK-ON DISASTERS (DF11-20) CAUSE AN IMMEDIATE SUT DOWN OF THE SYSTEM AND THE REACTOR IS DUMPED>
    DIS_STUCK_ON    LD    R0   DF11     <IF ANY OF THE DISASTER FLAGS DF11 to DF20 ARE SET JUMP TO DSO1>
                    BPL   DSO1
                    LD    R0   DF12     <IF ANY OF THE DISASTER FLAGS DF11 to DF20 ARE SET JUMP TO DSO1>
                    BPL   DSO1
                    LD    R0   DF13     <IF ANY OF THE DISASTER FLAGS DF11 to DF20 ARE SET JUMP TO DSO1>
                    BPL   DSO1
                    LD    R0   DF14     <IF ANY OF THE DISASTER FLAGS DF11 to DF20 ARE SET JUMP TO DSO1>
                    BPL   DSO1
                    LD    R0   DF15     <IF ANY OF THE DISASTER FLAGS DF11 to DF20 ARE SET JUMP TO DSO1>
                    BPL   DSO1
                    LD    R0   DF16     <IF ANY OF THE DISASTER FLAGS DF11 to DF20 ARE SET JUMP TO DSO1>
                    BPL   DSO1
                    LD    R0   DF17     <IF ANY OF THE DISASTER FLAGS DF11 to DF20 ARE SET JUMP TO DSO1>
                    BPL   DSO1
                    LD    R0   DF18     <IF ANY OF THE DISASTER FLAGS DF11 to DF20 ARE SET JUMP TO DSO1>
                    BPL   DSO1
                    LD    R0   DF19     <IF ANY OF THE DISASTER FLAGS DF11 to DF20 ARE SET JUMP TO DSO1>
                    BPL   DSO1
                    LD    R0   DF20     <NONE OF THE DISASTER FLAGS DF11 to DF20 ARE SET JUMP TO DIS1>
                    BRA   DIS1
                    --------------------
        DSO1        JSR   ZEROREACTOR   <SET U1,2,3,4,5,8,H,=0)
                    STO   R3    U6      <SET U6=0>
                    STO   R3    U7      <SET U7=0>
                    STO   R3    UC      <SET UC=0>
                    STO   R4    U8      <SET U8=1> <DUMP REACTOR>
                    BRA   DIS1
                    BRA   DIS1
                    ====================
        DIS1        BNE   DIS1A         <EVALUATE CONDITIONAL STATEMENT: IS (DF1)=1 ?><IF YES,JUMP TO DIS1A>
                                        <NO! , CONTINUE EVALUATING SAME DECISION BLOCK>
                    LD    R0    DF2     <PREPARE TO EVALUATE CONDITIONAL STATEMENT, LOAD VALUE FROM MEM ADX>
                    BNE   DIS1A         <EVALUATE CONDITIONAL STATEMENT: IS (DF2)=1 ?><IF YES,JUMP TO DIS1A>
                                        <NO! , CONTINUE EVALUATING SAME DECISION BLOCK>
                    LD    R0    DF3     <PREPARE TO EVALUATE CONDITIONAL STATEMENT, LOAD VALUE FROM MEM ADX>
                    BNE   DIS1A         <EVALUATE CONDITIONAL STATEMENT: IS (DF3)=1 ?><IF YES,JUMP TO DIS1A>
                                        <NO! , CONTINUE EVALUATING SAME DECISION BLOCK>
```

```
        LD   R0    DF4      <PREPARE TO EVALUATE CONDITIONAL STATEMENT, LOAD VALUE FROM MEM ADX>
        BNE  DIS1A          <EVALUATE CONDITIONAL STATEMENT: IS (DF4)=1 ?><IF YES,JUMP TO DIS1A>
                            <NO! , CONTINUE EVALUATING SAME DECISION BLOCK>
        LD   R0    DF5      <PREPARE TO EVALUATE CONDITIONAL STATEMENT, LOAD VALUE FROM MEM ADX>
        BNE  DIS1A          <EVALUATE CONDITIONAL STATEMENT: IS (DF5)=1 ?><IF YES,JUMP TO DIS1A>
        BRA  DIS2           <NO!, GO TO NEXT DECISION BLOCK>
        --------------------
DIS1A   LD   R0    VR       <PREPARE TO EVALUATE CONDITIONAL STATEMENT, LOAD VALUE FROM MEM ADX>
        LD   R1    VRMAX    <PREPARE TO EVALUATE CONDITIONAL STATEMENT, LOAD VALUE FROM MEM ADX>
        SUB  R1    R0       <EVALUATE CONDITIONAL STATEMENT: IS (VR > VRMAX)? >
        BEQ  LOWER4B        <NO! ,THEY ARE EQUAL; BRANCH TO DIS1B>
        BNG  LOWER4B        <NO! ,THE REVERSE IS TRUE; BRANCH TO DIS1B>
        STO  R3    U8       <YES!, SET U8=0>
        BRA  CONTROL_OVER   <ALL CONTROL SIGNALS HAVE BEEN CALCULATED, PREPARE TO SEND THEM OUT>
        --------------------
DIS1B   STO  R4    U8       <SET U8=1>
        BRA  CONTROL_OVER   <ALL CONTROL SIGNALS HAVE BEEN CALCULATED, PREPARE TO SEND THEM OUT>
        --------------------
DIS2    LD   DF6            <PREPARE TO EVALUATE CONDITIONAL STATEMENT, LOAD VALUE FROM MEM ADX>
        BNE  DIS2A          <EVALUATE CONDITIONAL STATEMENT: IS (DF6)=1 ?><IF YES,JUMP TO DIS2A>
        BRA  DIS3           <NO!, GO TO NEXT DECISION BLOCK>
        --------------------
DIS2A   LD   R5    STATE    <PREPARE TO EVALUATE CONDITIONAL STATEMENT, LOAD VALUE FROM MEM ADX>
        LD1  R0      1001   <PREPARE TO EVALUATE CONDITIONAL STATEMENT, LOAD VALUE FROM MEM ADX>
        CMP  R0    R5       <PREPARE TO EVALUATE CONDITIONAL STATEMENT, LOAD VALUE FROM MEM ADX>
        BEQ  DIS2B          <EVALUATE CONDITIONAL STATEMENT: DOES STATE= "FILL" ?><IF YES,JUMP TO DIS2B>
        STO  R3    FLAG1    <NO!, SET FLAG#1=1>
        BRA  CONTROL_OVER   <ALL CONTROL SIGNALS HAVE BEEN CALCULATED, PREPARE TO SEND THEM OUT>
        --------------------
DIS2B   BRA  CONTROL_OVER   <ALL CONTROL SIGNALS HAVE BEEN CALCULATED, PREPARE TO SEND THEM OUT>
        --------------------
DIS3    LD   DF7            <PREPARE TO EVALUATE CONDITIONAL STATEMENT, LOAD VALUE FROM MEM ADX>
        BNE  DIS3A          <EVALUATE CONDITIONAL STATEMENT: IS (DF7)=1 ?><IF YES,JUMP TO DIS2A>
        BRA  DIS4           <NO!, GO TO NEXT DECISION BLOCK>
        --------------------
DIS3A   LD   R5    STATE    <PREPARE TO EVALUATE CONDITIONAL STATEMENT, LOAD VALUE FROM MEM ADX>
        LD1  R0      1001   <PREPARE TO EVALUATE CONDITIONAL STATEMENT, LOAD VALUE FROM MEM ADX>
        CMP  R0    R5       <PREPARE TO EVALUATE CONDITIONAL STATEMENT, LOAD VALUE FROM MEM ADX>
        BEQ  DIS3B          <EVALUATE CONDITIONAL STATEMENT: DOES STATE= "FILL" ?><IF YES,JUMP TO DIS3B>
        STO  R3    UC       <NO!, SET UC=0>
        STO  R3    U7       <AND SET U7=0>
        STO  R3    FLAG1    <AND SET FLAG#1=1>
        BRA  CONTROL_OVER   <ALL CONTROL SIGNALS HAVE BEEN CALCULATED, PREPARE TO SEND THEM OUT>
        --------------------
DIS3B   STO  R3    UC       <SET UC=0>
        STO  R3    U7       <SET U7=0>
        BRA  CONTROL_OVER   <ALL CONTROL SIGNALS HAVE BEEN CALCULATED, PREPARE TO SEND THEM OUT>
        --------------------
DIS4    LD   DF9            <PREPARE TO EVALUATE CONDITIONAL STATEMENT, LOAD VALUE FROM MEM ADX>
        BNE  DIS4A          <EVALUATE CONDITIONAL STATEMENT: IS (DF7)=1 ?><IF YES,JUMP TO DIS4A>
        LD   DF10           <PREPARE TO EVALUATE CONDITIONAL STATEMENT, LOAD VALUE FROM MEM ADX>
        BNE  DIS3A          <EVALUATE CONDITIONAL STATEMENT: IS (DF7)=1 ?><IF YES,JUMP TO DIS4A>
        BRA  CONTROL_OVER   <NO!, CONTROL SIGNALS HAVE BEEN CALCULATED, PREPARE TO SEND THEM OUT>
        --------------------
DIS4A   LD   R5    STATE    <PREPARE TO EVALUATE CONDITIONAL STATEMENT, LOAD VALUE FROM MEM ADX>
        LD1  R0      1001   <PREPARE TO EVALUATE CONDITIONAL STATEMENT, LOAD VALUE FROM MEM ADX>
        CMP  R0    R5       <PREPARE TO EVALUATE CONDITIONAL STATEMENT, LOAD VALUE FROM MEM ADX>
        BEQ  DIS3B          <EVALUATE CONDITIONAL STATEMENT: DOES STATE= "FILL" ?><IF YES,JUMP TO DIS4B>
        STO  R3    UC       <NO!, SET UC=0>
        STO  R3    U7       <AND SET U7=0>
```

```
                STO   R3    FLAG1    <AND SET FLAG#1=1>
                BRA   CONTROL_OVER   <ALL CONTROL SIGNALS HAVE BEEN CALCULATED, PREPARE TO SEND THEM OUT>
                --------------------
        DIS4B   BRA   CONTROL_OVER   <ALL CONTROL SIGNALS HAVE BEEN CALCULATED, PREPARE TO SEND THEM OUT>
                --------------------
CONTROL_OVER    STO   R5    STATE    <ALL CONTROL SIGNALS HAVE BEEN CALCULATED, PREPARE TO SEND THEM OUT>


================================================================================================================================================
                <THIS PROGRAM SEGMENT SENDS OUT DIGITAL VALUES TO 40 DOUT CHANNELS>
DIGITAL_OUT     CLR R0               <PURGE REGISTER TO PREVENT ERRONEOUS READINGS>
                CLR R1               <PURGE REGISTER TO PREVENT ERRONEOUS READINGS>
                CLR R2               <PURGE REGISTER TO PREVENT ERRONEOUS READINGS>
                CLR R3               <PURGE REGISTER TO PREVENT ERRONEOUS READINGS>
                CLR R4               <PURGE REGISTER TO PREVENT ERRONEOUS READINGS>
                CLR R5               <PURGE REGISTER TO PREVENT ERRONEOUS READINGS>
                --------------------
                <THIS FOLLOWING SEGMENT SETS A MAXIMUM ALLOWED TIME FOR ALL DIGITAL DATA TO BE SENT OUT>
    DOUT_TIM_MSTR CLTIM     1          <PREPARE TO USE THIS TIMER>
                LDI   R0        0.5  <SINCE EACH CHANNEL WILL ONLY BE ALLOWED 0.01 MILLISECONDS TO SEND OUT,... >
                RTIM      1 R0       <TOTAL TIME ALLOWED= (40*0.01)+(0.1 MILLISECONDS FOR OTHER INSTRUCTION EXECUTIONS>
                BTIM      1 WAIT     <IF TIMER 1 RUNS OUT, TERMINATE THE DIGITAL OUT FOR THIS SCAN>
                BRA   DOSTART
                ====================
                <THIS SUBROUTINE SETS A MAXIMUM ALLOWED TIME FOR ANY ONE DIGITAL SIGNAL TO BE SENT OUT>
                <THIS SUBROUTINE ALSO SENDS OUT THE OUTGOING DATA FROM CONSECUTIVE LOCATIONS IN MEMORY>
    DOTIM&LD    CTIM      2          <DATA IS SENT, CLEAR TIMER 2 TO PREVENT UNWANTED BRANCHING>
                INC   R0             <INCRIMENT THE ADRESS OF WHERE TO GET THE OUTGOING DATA>
                LDX   R1    R0       <LOAD DATA TO BE SENT INTO REGISTER #1>
                CLR   R1             <PURGE REGISTER TO PREVENT ERRONEOUS READINGS>
    DOTIM&LD1   LDI   R4          4  <THE ADRESS OF EACH PROGRAM SEGMENT TO SEND OUT A SIGNAL ARE SEPERATED BY 4>
                ADD   R4    R3       <R3 NOW CONTAINS THE ADRESS TO JUMP TO IF TIMER 2 RUNS OUT>
                LDI   R2       0.01  <ONLY 0.01 MILLISECONDS TOTAL WILL BE ALLOWED FOR CHANNEL READY AND SENDING SIGNAL>
                RTIM      2 R2       <SET TIMER >
                BTIM  DOSKIP         <SKIP CHANNEL WHEN THIS TIMER RUNS OUT>
                BRA   DORETURN       <RETURN FROM SUBROUTINE>
        DOSKIP  BRI   R3             <SKIP CHANNEL, TIMER 2 HAS RUN OUT>
        DORETURN RTN                 <RETURN FROM SUBROUTINE>
                ====================
    DOSTART     LDI   R0    U1_ADX   <LOAD THE FIRST ADRESS OF THE TABLE WHERE THE OUTGOING DATA WILL BE SENT FROM>
                LD1   R3    AIN1     <LOAD THE ADRESS OF THE FIRST SEGMENT TO SEND OUT DATA>
                LDX   R1    R0       <LOAD THE FIRST SIGNAL TO BE SENT>
                CTIM      2          <DATA IS SENT, CLEAR TIMER 2 TO PREVENT UNWANTED BRANCHING>
                JSR   DOTIM&LD1
                --------------------
        DO1     TDOUT     1          <TEST IF CHANNEL READY>
                BNE   DO1            <IF NOT, KEEP LOOPING>  <TIMER 2 WILL STOP THIS LOOPING AFTER 0.01 MILLISECONDS>
                DOUT      1 R1       <IF CHANNEL READY, SEND OUT DATA>
                JSR   DOTIM&LD       <EXECECUTE SUBROUTINE TO GET THE NEXT VALUE FROM THE TABLE IN MEMORY AND RESET TIMER 2>
                --------------------
        DO2     TDOUT     2          <TEST IF CHANNEL READY>
                BNE   DO2            <IF NOT, KEEP LOOPING>  <TIMER 2 WILL STOP THIS LOOPING AFTER 0.01 MILLISECONDS>
                DOUT      2 R1       <IF CHANNEL READY, SEND OUT DATA>
                JSR   DOTIM&LD       <EXECECUTE SUBROUTINE TO GET THE NEXT VALUE FROM THE TABLE IN MEMORY AND RESET TIMER 2>
                --------------------
        DO3     TDOUT     3          <TEST IF CHANNEL READY>
                BNE   DO3            <IF NOT, KEEP LOOPING>  <TIMER 2 WILL STOP THIS LOOPING AFTER 0.01 MILLISECONDS>
                DOUT      3 R1       <IF CHANNEL READY, SEND OUT DATA>
                JSR   DOTIM&LD       <EXECECUTE SUBROUTINE TO GET THE NEXT VALUE FROM THE TABLE IN MEMORY AND RESET TIMER 2>
```

```
        --------------------
DO4     TDOUT     4          <TEST IF CHANNEL READY>
        BNE   DO4            <IF NOT, KEEP LOOPING>  <TIMER 2 WILL STOP THIS LOOPING AFTER 0.01 MILLISECONDS>
        DOUT      4 R1       <IF CHANNEL READY, SEND OUT DATA>
        JSR   DOTIM&LD       <EXECECUTE SUBROUTINE TO GET THE NEXT VALUE FROM THE TABLE IN MEMORY AND RESET TIMER 2>
        --------------------
DO5     TDOUT     5          <TEST IF CHANNEL READY>
        BNE   DO5            <IF NOT, KEEP LOOPING>  <TIMER 2 WILL STOP THIS LOOPING AFTER 0.01 MILLISECONDS>
        DOUT      5 R1       <IF CHANNEL READY, SEND OUT DATA>
        JSR   DOTIM&LD       <EXECECUTE SUBROUTINE TO GET THE NEXT VALUE FROM THE TABLE IN MEMORY AND RESET TIMER 2>
        --------------------
DO6     TDOUT     6          <TEST IF CHANNEL READY>
        BNE   DO6            <IF NOT, KEEP LOOPING>  <TIMER 2 WILL STOP THIS LOOPING AFTER 0.01 MILLISECONDS>
        DOUT      6 R1       <IF CHANNEL READY, SEND OUT DATA>
        JSR   DOTIM&LD       <EXECECUTE SUBROUTINE TO GET THE NEXT VALUE FROM THE TABLE IN MEMORY AND RESET TIMER 2>
        --------------------
DO7     TDOUT     7          <TEST IF CHANNEL READY>
        BNE   DO7            <IF NOT, KEEP LOOPING>  <TIMER 2 WILL STOP THIS LOOPING AFTER 0.01 MILLISECONDS>
        DOUT      7 R1       <IF CHANNEL READY, SEND OUT DATA>
        JSR   DOTIM&LD       <EXECECUTE SUBROUTINE TO GET THE NEXT VALUE FROM THE TABLE IN MEMORY AND RESET TIMER 2>
        --------------------
DO8     TDOUT     8          <TEST IF CHANNEL READY>
        BNE   DO8            <IF NOT, KEEP LOOPING>  <TIMER 2 WILL STOP THIS LOOPING AFTER 0.01 MILLISECONDS>
        DOUT      8 R1       <IF CHANNEL READY, SEND OUT DATA>
        JSR   DOTIM&LD       <EXECECUTE SUBROUTINE TO GET THE NEXT VALUE FROM THE TABLE IN MEMORY AND RESET TIMER 2>
        --------------------
DO9     TDOUT     9          <TEST IF CHANNEL READY>
        BNE   DO9            <IF NOT, KEEP LOOPING>  <TIMER 2 WILL STOP THIS LOOPING AFTER 0.01 MILLISECONDS>
        DOUT      9 R1       <IF CHANNEL READY, SEND OUT DATA>
        JSR   DOTIM&LD       <EXECECUTE SUBROUTINE TO GET THE NEXT VALUE FROM THE TABLE IN MEMORY AND RESET TIMER 2>
        --------------------
DO10    TDOUT    10          <TEST IF CHANNEL READY>
        BNE   DO10           <IF NOT, KEEP LOOPING>  <TIMER 2 WILL STOP THIS LOOPING AFTER 0.01 MILLISECONDS>
        DOUT     10 R1       <IF CHANNEL READY, SEND OUT DATA>
        JSR   DOTIM&LD       <EXECECUTE SUBROUTINE TO GET THE NEXT VALUE FROM THE TABLE IN MEMORY AND RESET TIMER 2>
        --------------------
DO11    TDOUT    11          <TEST IF CHANNEL READY>
        BNE   DO11           <IF NOT, KEEP LOOPING>  <TIMER 2 WILL STOP THIS LOOPING AFTER 0.01 MILLISECONDS>
        DOUT     11 R1       <IF CHANNEL READY, SEND OUT DATA>
        JSR   DOTIM&LD       <EXECECUTE SUBROUTINE TO GET THE NEXT VALUE FROM THE TABLE IN MEMORY AND RESET TIMER 2>
        --------------------
DO12    TDOUT    12          <TEST IF CHANNEL READY>
        BNE   DO12           <IF NOT, KEEP LOOPING>  <TIMER 2 WILL STOP THIS LOOPING AFTER 0.01 MILLISECONDS>
        DOUT     12 R1       <IF CHANNEL READY, SEND OUT DATA>
        JSR   DOTIM&LD       <EXECECUTE SUBROUTINE TO GET THE NEXT VALUE FROM THE TABLE IN MEMORY AND RESET TIMER 2>
        --------------------
DO13    TDOUT    13          <TEST IF CHANNEL READY>
        BNE   DO13           <IF NOT, KEEP LOOPING>  <TIMER 2 WILL STOP THIS LOOPING AFTER 0.01 MILLISECONDS>
        DOUT     13 R1       <IF CHANNEL READY, SEND OUT DATA>
        JSR   DOTIM&LD       <EXECECUTE SUBROUTINE TO GET THE NEXT VALUE FROM THE TABLE IN MEMORY AND RESET TIMER 2>
        --------------------
DO14    TDOUT    14          <TEST IF CHANNEL READY>
        BNE   DO14           <IF NOT, KEEP LOOPING>  <TIMER 2 WILL STOP THIS LOOPING AFTER 0.01 MILLISECONDS>
        DOUT     14 R1       <IF CHANNEL READY, SEND OUT DATA>
        JSR   DOTIM&LD       <EXECECUTE SUBROUTINE TO GET THE NEXT VALUE FROM THE TABLE IN MEMORY AND RESET TIMER 2>
        --------------------
DO15    TDOUT    15          <TEST IF CHANNEL READY>
        BNE   DO15           <IF NOT, KEEP LOOPING>  <TIMER 2 WILL STOP THIS LOOPING AFTER 0.01 MILLISECONDS>
        DOUT     15 R1       <IF CHANNEL READY, SEND OUT DATA>
        JSR   DOTIM&LD       <EXECECUTE SUBROUTINE TO GET THE NEXT VALUE FROM THE TABLE IN MEMORY AND RESET TIMER 2>
```

```
        -------------------
DO16    TDOUT    16          <TEST IF CHANNEL READY>
        BNE   DO16           <IF NOT, KEEP LOOPING>  <TIMER 2 WILL STOP THIS LOOPING AFTER 0.01 MILLISECONDS>
        DOUT     16 R1       <IF CHANNEL READY, SEND OUT DATA>
        JSR   DOTIM&LD       <EXECECUTE SUBROUTINE TO GET THE NEXT VALUE FROM THE TABLE IN MEMORY AND RESET TIMER 2>
        -------------------
DO17    TDOUT    17          <TEST IF CHANNEL READY>
        BNE   DO17           <IF NOT, KEEP LOOPING>  <TIMER 2 WILL STOP THIS LOOPING AFTER 0.01 MILLISECONDS>
        DOUT     17 R1       <IF CHANNEL READY, SEND OUT DATA>
        JSR   DOTIM&LD       <EXECECUTE SUBROUTINE TO GET THE NEXT VALUE FROM THE TABLE IN MEMORY AND RESET TIMER 2>
        -------------------
DO18    TDOUT    18          <TEST IF CHANNEL READY>
        BNE   DO18           <IF NOT, KEEP LOOPING>  <TIMER 2 WILL STOP THIS LOOPING AFTER 0.01 MILLISECONDS>
        DOUT     18 R1       <IF CHANNEL READY, SEND OUT DATA>
        JSR   DOTIM&LD       <EXECECUTE SUBROUTINE TO GET THE NEXT VALUE FROM THE TABLE IN MEMORY AND RESET TIMER 2>
        -------------------
DO19    TDOUT    19          <TEST IF CHANNEL READY>
        BNE   DO19           <IF NOT, KEEP LOOPING>  <TIMER 2 WILL STOP THIS LOOPING AFTER 0.01 MILLISECONDS>
        DOUT     19 R1       <IF CHANNEL READY, SEND OUT DATA>
        JSR   DOTIM&LD       <EXECECUTE SUBROUTINE TO GET THE NEXT VALUE FROM THE TABLE IN MEMORY AND RESET TIMER 2>
        -------------------
DO20    TDOUT    20          <TEST IF CHANNEL READY>
        BNE   DO20           <IF NOT, KEEP LOOPING>  <TIMER 2 WILL STOP THIS LOOPING AFTER 0.01 MILLISECONDS>
        DOUT     20 R1       <IF CHANNEL READY, SEND OUT DATA>
        JSR   DOTIM&LD       <EXECECUTE SUBROUTINE TO GET THE NEXT VALUE FROM THE TABLE IN MEMORY AND RESET TIMER 2>
        -------------------
DO21    TDOUT    21          <TEST IF CHANNEL READY>
        BNE   DO21           <IF NOT, KEEP LOOPING>  <TIMER 2 WILL STOP THIS LOOPING AFTER 0.01 MILLISECONDS>
        DOUT     21 R1       <IF CHANNEL READY, SEND OUT DATA>
        JSR   DOTIM&LD       <EXECECUTE SUBROUTINE TO GET THE NEXT VALUE FROM THE TABLE IN MEMORY AND RESET TIMER 2>
        -------------------
DO22    TDOUT    22          <TEST IF CHANNEL READY>
        BNE   DO22           <IF NOT, KEEP LOOPING>  <TIMER 2 WILL STOP THIS LOOPING AFTER 0.01 MILLISECONDS>
        DOUT     22 R1       <IF CHANNEL READY, SEND OUT DATA>
        JSR   DOTIM&LD       <EXECECUTE SUBROUTINE TO GET THE NEXT VALUE FROM THE TABLE IN MEMORY AND RESET TIMER 2>
        -------------------
DO23    TDOUT    23          <TEST IF CHANNEL READY>
        BNE   DO23           <IF NOT, KEEP LOOPING>  <TIMER 2 WILL STOP THIS LOOPING AFTER 0.01 MILLISECONDS>
        DOUT     23 R1       <IF CHANNEL READY, SEND OUT DATA>
        JSR   DOTIM&LD       <EXECECUTE SUBROUTINE TO GET THE NEXT VALUE FROM THE TABLE IN MEMORY AND RESET TIMER 2>
        -------------------
DO24    TDOUT    24          <TEST IF CHANNEL READY>
        BNE   DO24           <IF NOT, KEEP LOOPING>  <TIMER 2 WILL STOP THIS LOOPING AFTER 0.01 MILLISECONDS>
        DOUT     24 R1       <IF CHANNEL READY, SEND OUT DATA>
        JSR   DOTIM&LD       <EXECECUTE SUBROUTINE TO GET THE NEXT VALUE FROM THE TABLE IN MEMORY AND RESET TIMER 2>
        -------------------
DO25    TDOUT    25          <TEST IF CHANNEL READY>
        BNE   DO25           <IF NOT, KEEP LOOPING>  <TIMER 2 WILL STOP THIS LOOPING AFTER 0.01 MILLISECONDS>
        DOUT     25 R1       <IF CHANNEL READY, SEND OUT DATA>
        JSR   DOTIM&LD       <EXECECUTE SUBROUTINE TO GET THE NEXT VALUE FROM THE TABLE IN MEMORY AND RESET TIMER 2>
        -------------------
DO26    TDOUT    26          <TEST IF CHANNEL READY>
        BNE   DO26           <IF NOT, KEEP LOOPING>  <TIMER 2 WILL STOP THIS LOOPING AFTER 0.01 MILLISECONDS>
        DOUT     26 R1       <IF CHANNEL READY, SEND OUT DATA>
        JSR   DOTIM&LD       <EXECECUTE SUBROUTINE TO GET THE NEXT VALUE FROM THE TABLE IN MEMORY AND RESET TIMER 2>
        -------------------
DO27    TDOUT    27          <TEST IF CHANNEL READY>
        BNE   DO27           <IF NOT, KEEP LOOPING>  <TIMER 2 WILL STOP THIS LOOPING AFTER 0.01 MILLISECONDS>
        DOUT     27 R1       <IF CHANNEL READY, SEND OUT DATA>
        JSR   DOTIM&LD       <EXECECUTE SUBROUTINE TO GET THE NEXT VALUE FROM THE TABLE IN MEMORY AND RESET TIMER 2>
```

```
        --------------------
D028    TDOUT    28          <TEST IF CHANNEL READY>
        BNE   D028           <IF NOT, KEEP LOOPING>  <TIMER 2 WILL STOP THIS LOOPING AFTER 0.01 MILLISECONDS>
        DOUT     28 R1       <IF CHANNEL READY, SEND OUT DATA>
        JSR   DOTIM&LD       <EXECECUTE SUBROUTINE TO GET THE NEXT VALUE FROM THE TABLE IN MEMORY AND RESET TIMER 2>
        --------------------
D029    TDOUT    29          <TEST IF CHANNEL READY>
        BNE   D029           <IF NOT, KEEP LOOPING>  <TIMER 2 WILL STOP THIS LOOPING AFTER 0.01 MILLISECONDS>
        DOUT     29 R1       <IF CHANNEL READY, SEND OUT DATA>
        JSR   DOTIM&LD       <EXECECUTE SUBROUTINE TO GET THE NEXT VALUE FROM THE TABLE IN MEMORY AND RESET TIMER 2>
        --------------------
D030    TDOUT    30          <TEST IF CHANNEL READY>
        BNE   D030           <IF NOT, KEEP LOOPING>  <TIMER 2 WILL STOP THIS LOOPING AFTER 0.01 MILLISECONDS>
        DOUT     30 R1       <IF CHANNEL READY, SEND OUT DATA>
        JSR   DOTIM&LD       <EXECECUTE SUBROUTINE TO GET THE NEXT VALUE FROM THE TABLE IN MEMORY AND RESET TIMER 2>
        --------------------
D031    TDOUT    31          <TEST IF CHANNEL READY>
        BNE   D031           <IF NOT, KEEP LOOPING>  <TIMER 2 WILL STOP THIS LOOPING AFTER 0.01 MILLISECONDS>
        DOUT     31 R1       <IF CHANNEL READY, SEND OUT DATA>
        JSR   DOTIM&LD       <EXECECUTE SUBROUTINE TO GET THE NEXT VALUE FROM THE TABLE IN MEMORY AND RESET TIMER 2>
        --------------------
D032    TDOUT    32          <TEST IF CHANNEL READY>
        BNE   D032           <IF NOT, KEEP LOOPING>  <TIMER 2 WILL STOP THIS LOOPING AFTER 0.01 MILLISECONDS>
        DOUT     32 R1       <IF CHANNEL READY, SEND OUT DATA>
        JSR   DOTIM&LD       <EXECECUTE SUBROUTINE TO GET THE NEXT VALUE FROM THE TABLE IN MEMORY AND RESET TIMER 2>
        --------------------
D033    TDOUT    33          <TEST IF CHANNEL READY>
        BNE   D033           <IF NOT, KEEP LOOPING>  <TIMER 2 WILL STOP THIS LOOPING AFTER 0.01 MILLISECONDS>
        DOUT     33 R1       <IF CHANNEL READY, SEND OUT DATA>
        JSR   DOTIM&LD       <EXECECUTE SUBROUTINE TO GET THE NEXT VALUE FROM THE TABLE IN MEMORY AND RESET TIMER 2>
        --------------------
D034    TDOUT    34          <TEST IF CHANNEL READY>
        BNE   D034           <IF NOT, KEEP LOOPING>  <TIMER 2 WILL STOP THIS LOOPING AFTER 0.01 MILLISECONDS>
        DOUT     34 R1       <IF CHANNEL READY, SEND OUT DATA>
        JSR   DOTIM&LD       <EXECECUTE SUBROUTINE TO GET THE NEXT VALUE FROM THE TABLE IN MEMORY AND RESET TIMER 2>
        --------------------
D035    TDOUT    35          <TEST IF CHANNEL READY>
        BNE   D035           <IF NOT, KEEP LOOPING>  <TIMER 2 WILL STOP THIS LOOPING AFTER 0.01 MILLISECONDS>
        DOUT     35 R1       <IF CHANNEL READY, SEND OUT DATA>
        JSR   DOTIM&LD       <EXECECUTE SUBROUTINE TO GET THE NEXT VALUE FROM THE TABLE IN MEMORY AND RESET TIMER 2>
        --------------------
D036    TDOUT    36          <TEST IF CHANNEL READY>
        BNE   D036           <IF NOT, KEEP LOOPING>  <TIMER 2 WILL STOP THIS LOOPING AFTER 0.01 MILLISECONDS>
        DOUT     36 R1       <IF CHANNEL READY, SEND OUT DATA>
        JSR   DOTIM&LD       <EXECECUTE SUBROUTINE TO GET THE NEXT VALUE FROM THE TABLE IN MEMORY AND RESET TIMER 2>
        --------------------
D037    TDOUT    37          <TEST IF CHANNEL READY>
        BNE   D037           <IF NOT, KEEP LOOPING>  <TIMER 2 WILL STOP THIS LOOPING AFTER 0.01 MILLISECONDS>
        DOUT     37 R1       <IF CHANNEL READY, SEND OUT DATA>
        JSR   DOTIM&LD       <EXECECUTE SUBROUTINE TO GET THE NEXT VALUE FROM THE TABLE IN MEMORY AND RESET TIMER 2>
        --------------------
D038    TDOUT    38          <TEST IF CHANNEL READY>
        BNE   D038           <IF NOT, KEEP LOOPING>  <TIMER 2 WILL STOP THIS LOOPING AFTER 0.01 MILLISECONDS>
        DOUT     38 R1       <IF CHANNEL READY, SEND OUT DATA>
        JSR   DOTIM&LD       <EXECECUTE SUBROUTINE TO GET THE NEXT VALUE FROM THE TABLE IN MEMORY AND RESET TIMER 2>
        --------------------
D039    TDOUT    39          <TEST IF CHANNEL READY>
        BNE   D039           <IF NOT, KEEP LOOPING>  <TIMER 2 WILL STOP THIS LOOPING AFTER 0.01 MILLISECONDS>
        DOUT     39 R1       <IF CHANNEL READY, SEND OUT DATA>
        JSR   DOTIM&LD       <EXECECUTE SUBROUTINE TO GET THE NEXT VALUE FROM THE TABLE IN MEMORY AND RESET TIMER 2>
```

```
                ----------------------
        DO40    TDOUT    40          <TEST IF CHANNEL READY>
                BNE   DO40           <IF NOT, KEEP LOOPING>  <TIMER 2 WILL STOP THIS LOOPING AFTER 0.01 MILLISECONDS>
                DOUT     40 R1       <IF CHANNEL READY, SEND OUT DATA>
                ----------------------  EXECECUTE SUBROUTINE TO GET THE NEXT VALUE FROM THE TABLE IN MEMORY AND RESET TIMER 2>
                CTIM      1          <CLEAR 1 TIMER TO PREVENT UNWANTED BRANCHING>
                CTIM      2          <CLEAR 2 TIMER TO PREVENT UNWANTED BRANCHING>
================================================================================================================================================
WAIT            BTIM      3 TOP      <SINCE MAIN LOOP (SCAN) SHOULD FINISH IN LESS THAN THE 10 MILLISECONDS DICTATED BY TIMER 3,>
                BNE   WAIT           <THE PROGRAM WILL WAIT HERE UNTIL TIMER 3 RUNS OUT>
================================================================================================================================================
OUT             LD    RO    ESCAPE   <"ESCAPE" IS AN ADRESS IN THE SHARED INTERTASK AREA OF RAM WHERE A FLAG CAN BE SET TO..>
                BRA   START          <PREVENT THIS TASK FROM ENDLESSLY LOOPING>
================================================================================================================================================
END             NOP                  <THE PROGRAM CAN GET HERE IN ONE OF SIX WAYS:   >
                                     <  1) IF THE "ESCAPE" FLAG IN THE SHARED INTERTASK AREA OF RAM IS SET >
                                     <  2) IF THE CONTROL PANEL IS NOT TURNED ON WITHIN 30 SECONDS OF THIS TASK BEING SPAWNED...>
                                     <                                                  (ie.: TIMER 5 RUNS OUT)>
                                     <  3)IF A VOLUME REQUEST IS NOT PULSED IN WITHIN 5 MINUTES OF TURNING ON THE CONTROL PANEL>
                                     <  4)IF A VOLUME REQUEST IS NOT PULSED IN WITHIN 5 MINUTES OF FINISHING A PREVIOUS...>
                                     <    REQUESTED VOLUME>
                                     <  5)SOMETHING IS WRONG WITH ADC CHANNEL #27 (REQUESTED VOLUME FROM CONTROL PANEL)>
                                     <  6)SOMETHING IS WRONG WITH DIN CHANNEL #4 (CONTROL PANEL ON/OFF SWITCH)>
```

IV. CONCLUSIONS

The most critical part of designing a control program is timing. This differs from the writing of a simulation program in that no physical process delays are involved in a simulation program unless the simulation program relies on values input by a concurrently running Real-Time program. An explanation of the five timers used in the control program will demonstrate the critical timing aspects of controlling a physical system.

Timer #1

This timer is used to limit the overall time used to bring into and send out analog and digital values from the computer. This is to prevent the program from "hanging" if there are problems with the channels.

Timer #2

This timer is used to limit the time used for any individual channel.

Timer #3

This timer is the most critical part of the program. It dictates the scan-time for program control. The scan-time determines the time-delay between taking in measured values from the system, and sending out controls. The program must wait for the physical system to respond to the previous set of controls sent before it samples physical system measurements again. This sets a maximum for the sampling rate. The sampling rate cannot be too infrequent if the behavior of the system is to be tightly controlled by the control program. This defines the minimum for the sampling rate. The range of sampling rate values between the minimum and maximum defined above, dictates the size of the "Time Window" for sending process control actuation signals. The logic for the process control program that has been written to control the bottling plant is looped through every ten milliseconds. This amount of time should allow the physical system to adaquately respond while still tightly controlling the physical system. The program is forced to wait after looping through its logic until timer #3 runs out. The computer which will run the control program is assumed to have a processing speed of at least one MIP. This would result in an approximate average instruction execution of one microsecond. At this processing rate the logic of the control program should never take longer than two milliseconds to execute. This is true even if the total allowed time for all digital and analog inputting and outputting is used. The input and output values have been mapped into consecutive memory locations to simplify the storing and sending of values. This causes the output to lag behind input by one 10 millisecond scan. However, this 1/100th of a second delay has been judged to be tolerable. The most critical impact of using a 10 millisecond scan-time is the affect that it has on overfilling bottles. Since pump #7 is rated at 20 GPM, the maximum possible amount of overfill during any one scan is:
(20 gal/min)*(1min/60sec)*(0.01 sec.)*(128 oz./gal)= 0.426 ounces.
This has been judged to be a tolerable condition. Also, this condition is the maximum possible overfill and therefore overfill will usually be less.

Timer #4

This timer is used to allow the conveyor to shift in a new bottle. Since the scan-time is 1/100th of a second, the bottle cannot physically be shifted in this small amount of time. Therefore, the segment of program control which activates pump #7 when the bottle volume is calculated to be zero, is skipped for 100 scans. This will give the bottle one second to shift. Therefore timer #4 is set to 1000 milliseconds immediately following the shifting-out of a just-filled bottle.

Timer #5

This timer is used at the very beginning of the program to force the program to end if the control panel has not been turned on within 30 seconds of initiating the control program. This is the only way to exit the endlessly looping control program, except for a software construct that has been created just for this purpose. This construct consists of polling for a value of (1) at a memory address located in the shared intertask area of RAM specifically defined for the purpose of setting a flag to escape out of the endlessly looping control program.

The instruction set used to write this control program had some minor limitations. The most significant being, the inability to compute the channel number for I-O instructions. If the channel number was allowed to be an integer that could be incremented, the amount of lines of code could have greatly been reduced. However, even though the channel number could be calculated, the program would have to loop to calculate and therefore the control program would still take approximately the same amount of time to loop through its logic. Therefore, the critical aspects of timing would not be changed; only the ease of writing the program would result. Also if the channel number could be calculated, problems would arise in maintaining the code as relocatable.

A type of instruction that could be added to the instruction set is the Conditional Bit Test instruction. This type of instruction could be used to evaluate specific bits of a byte of information that was brought into the Q Buffer from a RS232C line. This capability could be used if dedicated computer's were located at a physical system for the sole purpose of collecting data, and communicating it to the main plant computer. Each bit of the byte communicated could be used for digital feedback. However, for the purpose of the bottling plant control program, the present instruction set was sufficient.