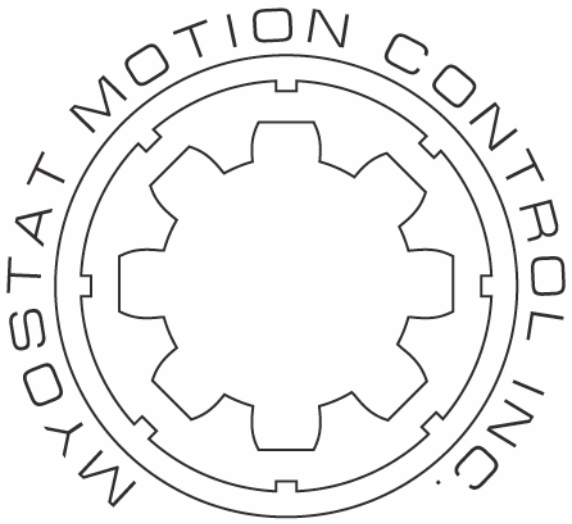
Cool Muscle RT3: CM Banks

1 – General bank Information

2 – Program Banks

3 – Logic Banks



Version 1.0.1 26-07-2007

1. CM Banks – General information

Banks allow for intelligent operation of a Cool Muscle motor. The banks can be programmed and stored in the CM and executed in a number of ways. Banks eliminate the need for an external program executing commands and as such improve performance and timing of the system. The following discusses commands and structure that are relevant to both logic and program banks. The two systems are independent but can offer enormous flexibility when used together. Further we will discuss the advantages of each and when it is best to use either of the bank types.

Both banks use a shared memory resource and the maximum number of steps is 200. The banks and amount of steps can be queried by ?1000.

1. Commands, states and operations

CM banks (logic and program) have access to all CML commands, registers and states. All registers are read/write however only V registers can hold states and can be compared. CML commands not just relevant to banks and registers are fully covered in section XXX.

1. Bank Commands

Command	Example and description	

I
-Input logic branch
-used with I1-I4
	Use in a bank:
	L1.1
	I1.1, C2.1, C3.1
	END.1
	
	Is equivalent to:
	
	V1=1
	L1.1
	I1==V1,C2.1,C3.1
	END.1
		
		If I1=true than C2 else C3
		In this case the ‘,‘ (comma) separates the then, else statement. The equivalent C-syntax would look like:
		if(I1.1) then{
		C2.1;
		}
		else{
		C3.1;
		}

V -Universal global variable -V1-V15	Assign before a call in a bank: V1=”ABCD” V2=1234 V3=”Ux” Use in a bank: P5=V2 V3==V5,?99.1,?96.1	 Use as a string of characters Use as a long type variable Access to internal motor states Set P5 to V2 If V3 equals V5 than query 99 else query 96.
W0 -Wait until the condition is satisfied	Use in a bank: I1.1,W0,C2.1	 While I1 is true do nothing. If I1 is false call bank 2 (C2.1)
T0 -Do nothing. I.e call a 0 second timer	Use in bank: V1==V2,?99,T0	 If V1 equals V2 then query motor status(?99) or else ‘do nothing’
END -end the bank	B1 A1,S1,P1 END	 End of bank program or Ladder Logic bank

: -junction -execute following command immediately after.	Use in bank: V1>V2,W0,?96:?90 Equivalent operation: L1 V1>V2,W0,CL2 END L2 ?96 ?90 END	if V1<=V2 is satisfied execute ?96 then immediately ?90 else wait. This command allows the call of multiple commands on a conditional statement without calling a bank first. As can be seen in the equivalent operation the ':' command replaces the need to call a separate bank to execute multiple commands on a branch.
; -NOTE this function is different in program and logic banks. Logic Bank: -stops the results of an equation from being displayed Program Bank: -merges 2 motion profiles together	Use in logic bank: V1=V2+V3 V1=V2+V3; Use in program bank: r2.1,r2.2,@1.2,@1.1; r1.1,r1.2,@2.2,@1.1	- Displays the solution over comm port - Will not display the solution. Merge the second motion into the first motion. Note: merge motion is to be used with coordinated motion. See the coordinate section for further information.
/ -comment -NOTE: comments cannot be used inside any banks. Inside it will be seen as a divide sign.	Use outside a bank: P1.1=1000 /end point	Use comment outside of banks. Comments are limited to 29 characters.

2. Registers

Register	# of values
P – Position	P1-P25
S – Speed	S1-S15
A – Acceleration	A1-A7
T – Timer	T1-T7
V – Variable	V1-V15
M – Torque	M1-M7
R – Radius	R1-R25
N – Center point	N1-N25

All registers should be set or at least initialized before they are called or changed in a bank. A register is set in CWLite or another terminal type window by setting the register directly equal to the number. E.g P1=100, S10=97353 , V1=-30 or V2="Ux".

If the register is set in the bank you cannot set it in the bank with an actual number. A register should be set before and then called.

CORRECT

```
V1.1=100
B1.1
A1.1=V1.1
END.1
```

INCORRECT

```
B1.1
A1.1=100
END.1
```

A register can then be queried by sending the regsister number not followed by an '=' sign. E.g to query P15.1 simply send P15.1. A response will be returned from the motor in the format P15.1=### where ### is the value of the register.

In general registers are set in logic banks. In program banks they can be set or called. For the calling structure of registers in program banks please see the program banks section. Following this section there is a list of variables, their description and syntax on usage.

Coordinated Motion - N and R are for coordinated motion within banks. There is a separate document covering coordinated motion. It is recommended that program bank operation is fully understood before using coordinated motion.

3. States and constants for V

Variables table:

State	R/W	Syntax	Description
Px	Read	V1="Px"	Current position
Sx	Read	V1="Sx"	Current speed
Ix	Read	V1="Ix"	Current Iq value
Ux	Read	V1="Ux"	Current status
Pe	Read	V1="Pe"	Current tracking error
Pt	Read	V1="Pt"	Current target position
AIN4	Read	V1="Ain4"	Analog in 4 (0~1024)
Dx	Write	V1="Dx"	Write to the target position (must use LM)

St	Write	V1="St"	Current Target Speed
Fx	Read	V1="Fx"	Input 2 frequency
Cx	Read	V1="Cx"	Input 2 counter
	Read	V1="What"	V1 is assigned a string constant (max 4 bytes)
	Read	V1=1976	V1 is assigned a long type integer value

In the above table all syntax examples are given using the variable V1. Any variable from V1-V15 may be used. Variables can also be assigned constants as shown in the last 2 entries on the above table.

4. Branch Operations

All branch operations following the following format:

QUERY,TRUE,FALSE

and would have and equivalent C format

if (QUERY) TRUE
else FALSE

QUERY can take on the following operations:

Operand	Format	Description
&&	V1 = I4.1 && I3.2	"and" operation – TRUE if the result does not equal 0.
||	V1 = I4.1 || I3.2	"or" operation – TRUE if the result does not equal 0.
!!	V1 = !!(I3.2)	"not" operation – TRUE if the result does not equal 0.
>	V1>V2+V3	The result is TRUE when V1>V2 + V3. otherwise the result is FALSE
>=	V1>=V2+V3	The result is TRUE when V1>=V2 + V3. otherwise the result is FALSE
==	S1==S2	The result is TRUE when S1==S2 otherwise the result is FALSE
<	V1<V2	The result is TRUE when V1<V2 otherwise the result is FALSE
<=	V1<=V2	The result is TRUE when V1<=V2 otherwise the result is FALSE
I	I1	If an input is used alone it is not necessary to compare as it is implied that if(I1==1) TRUE, FALSE

Example

```
V1.1=Ix
V2.1=50

L1.1
V1.1>=V2.1,?96.1,T0.1
END.1
```

In the adjacent example if V1 is greater or equal to V2, then query ?96.1 or else do nothing (T0.1). Any register value can be compared and it is not just restricted to V.

5. Math functions inside banks

Both types of banks allow for certain mathematic operations. Below is a table listing all math operations.

Basic math functions table:

Operand	Format	Description
=	P1.1=P2.1+V1.1	"equal" operation
+	P1.1=P2.1+V1.1	"add" operation
-	P1.1=P2.1-V1.1	"subtraction" operation
*	P1.1=P2.1*V1.1	"multiply" operation
/	P1.1=P2.1/V1.1	"division" operation

Advanced math functions table:

Operand	Format	Description
U1 - SIN	B1 S1.1=U1(V1.1) END L1 V1.1=V1.1+V3.1 V2.1=U1(V1.1) END	Set the output from a sine function with U1(f): $y=\sin[f*2\pi]$ The function is scaled in the following manner $y=10000 \sin((f\div10000)2\pi)$ The following relationships exist: V1.1=V1.1 + f * 10 * K87(msec) …(1) U1.1(V1.1) = $10000 \sin((V1.1\div10000)2\pi)$ …(2) Where: f=frequency K87=ladder logic scanning time in msec V1.1=sine angle In the example of the bank program the value S1 is assigned the result of: $S1.1=10000 \sin((V1.1\div10000)2\pi)$. If a sine function with a required frequency is to be generated it is done through the logic banks as given in the example. The logic scanning frequency (K87) and the discrete time increments in U1(V1.1) directly impact the sine frequency as shown in equation (1) above. The frequency is known and K87 is selected. For more information on K87 and its values see the section regarding logic bank K-parameters. With these two known the new V1 is calculated by the incremental addition of V3 on each logic iteration as follows: V1.1=V1.1+V3.1 and V1.1=V1.1 + f * 10 * K87(msec) $\therefore$ **V3.1=f * 10 * K87(msec)**
U2 - COSINE	V2.1=U2(V1.1)	As described above in the sine function.

U3 – SQUARE ROOT	P1.1=P2.1+U3(V1.1)	Square root operation U3(V1.1) = $\sqrt{}$(V1.1) Note: the return value of the operation is an integer value. So for example the $\sqrt{}$7=2. For higher accuracy it is suggested to scale the solution by orders of 10^{2n}. Example: $\sqrt{}(7*10^{2(0)})=2$ $\sqrt{}(7*10^{2(1)})=26$ $\sqrt{}(7*10^{2(2)})=264$ $\sqrt{}(7*10^{2(3)})=2645$
U4 – Lookup Table	P1.1=U4(V1.1)	Search the lookup table for V1.1 and return the solution. The look up value is searched through the N register The return value is the equivalent R register.
U5 – Polynomial using N	N0=4 V1.1=U5(P2.1)	N0 – polynomial order – max value = 25 N1-N25 polynomial coefficients N0=n $U5(x)=(N1)x^{(0)}+(N2)x^{(1)}+(N3)x^{(2)}\ldots+(N)x^{(n-1)}$ $\therefore$ as in the example V1.1=U5(P2.1) $V1.1=(N1)+(N2)(P2.1)+(N3)(P2.1)^2+(N4)(P2.1)^3$
U6 – Polynomial using R	R0=10 V1.1=U6(P2.1)	R0 – polynomial order – max value = 25 R1-R25 polynomial coefficients R0=n $U6(x)=(R1)x^{(0)}+(R2)x^{(1)}+(R3)x^{(3)}\ldots+(R)x^{(n-1)}$ $\therefore$ as in the example V1.1=U6(P2.1) $V1.1=(R1)+(R2)(P2.1)+\ldots+(R8)(P2.1)^8+(R9)(P2.1)^9$

2. Program Banks

CM program banks implement the day to day running of the motor for operations that are stored and saved in the motor. Though under special circumstances the execution of motion may be done through logic banks it is typically good practice to execute them through program banks.

2. Usage – why and when

Program banks will call position (P), speed (S), acceleration (A) and other registers. The registers are only called within the banks but are actually set before a bank is executed. The registers are P, S, A, T, V, M, N and R. A list of these registers and their definitions are given above in the General Bank section.

Unlike logic banks, program banks are run once through unless they have a loop command. Lines are executed one at a time and will wait for the line to finish before executing the next command. This is important if the motor is moving to a position as it will not execute the next line until it has reached position.

Program banks and logic banks can set registers through math commands but only program banks can actually call register that implies motion of the motor. E.g. A1=A2+A3

can be set in both logic and program banks but if A1 is to be called and used in the following command A1,S1,P1 it will be called in a program bank.

3. Program bank commands

Command	Example	Description
B# **END** **(program bank)**	B1.1 A1.1,S1.1,P1.1 END.1	Used when programming a program bank. B# starts the programming of the bank. All following data is considered part of the bank until END.1 is seen. The END.ID is important as the bank must finish with and it's ID.
C# **(Call bank)**	B1.1 C2.1 END.1 B2.1 A1.1,S1.1,P1.1 END.1	Call another program bank. The program will return once the bank is executed. This example will have an equivalent output to the above example.
J# **(Jump bank)**	B1.1 V1>V2,J2,?99.1 END.1 B2.1 A1,S1,P1 END.1	Jump to program bank 2 if V1>V2. Once the bank has jumped it will not return to the previous bank. In this example L2 has nothing in and will therefore do nothing.
[# **(start bank)**	[2	Start a program bank. This example will program bank 2.
] **(pause/stop bank)**	]	If the bank is running, the] command will pause the bank. If the program bank is then started it will continue from that position. If the bank is paused and the] command is sent again the motor and bank will go into a stopped state. I.e. if the bank is started again it will start from the beginning.
?# **(query bank)**	?3	Query a logic bank. This example queries logic bank 3.
B100 **(clear bank)**	B100	Clear all logic banks
?1000 **(query all banks)**	?1000	Query all banks. This will also indicate the amount of steps in program and logic banks. Max amount of steps is 200.

3. Logic Banks

CM Logic is the Cool Muscle implementation of ladder logic. The logic banks run in conjunction with the program banks allowing complete control in monitoring and changing states in the motor. These states include for example position, speed, torque and input status. Math operations and comparator statements allow for further manipulations of states, values and motion parameters.

1. Usage – why and when

Depending on the complexity of the application logic banks may be required. Typically standard motion applications can be accomplished using program banks. When multiple states or inputs are required to be monitored a logic bank may be necessary. A logic bank may also be used when for example a K-parameter does not fully accomplish the objective. For example: if the motor were to overload the default response would be to disable. If the required operation would be to back off, the logic bank could monitor the torque and issue commands to reverse direction on a set torque value.

2. Logic bank commands

The table below describes all commands related to the logic banks only. All commands, branch instructions and math operations in the above section apply.

Command	Example	Description
L# **END**	L1 ?99 END	Used when programming a logic bank. L# starts the programming of the bank. All following data is considered part of the bank until END is seen.
CL#	L1 CL2 END L2 ?99 END	Call another logic bank. The program will return once the logic bank is executed. This example will have an equivalent output to the above example.
JL#	L1.1 V1>V2,JL2,?99.1 END.1 L2.1 END.1	Jump to logic bank 2 if V1>V2. Once the bank has jumped it will not return to the previous bank. In this example L2 has nothing in and will therefore do nothing. A good use of the JL command is when something must just be executed once on power up.
[L#	[L2	Start a logic bank. This example will start logic bank 2.
]L	]L	Stop the current logic bank
[LM#	[LM1	Start a logic motion bank. This example will start logic bank 1 in the logic motion format.
]LM	]LM	Stop the current logic motion bank
?L#	?L3	Query a logic bank. This example queries logic bank 3.
L100	L100	Clear all logic banks
?1000	?1000	Query all banks. This will also indicate the amount of steps in program and logic banks. Max amount of steps is 200.

3. Logic bank syntax

Syntax in the logic banks and program banks are similar. If the syntax is incorrect the bank won't load correctly after that point. If you were to compare the loaded bank to what is expected it will assist in pinpointing where the incorrect syntax is.

Note: When the bank is loaded into a motor through CoolWorks Lite, it can be sent through the script window. In this case the 'send' button should be used and not the 'run' button. The 'send' button is in CWLite version 4.14 and up.

1. L# ... END:

All banks must start with their logic banks number and finish with END.ID. All commands between these two points are considered part of the logic bank.

Example:
L1.1
V1.1=P1.1
END.1

2. Carriage return, separating commands

All commands on the same line are grouped, some may be part of an equation, others a branch. Commands executed consecutively should be placed in the next line separated by a carriage return. The maximum amount of characters between a , (comma) and a ↵ (carriage return) is 40. If it exceeds 40 it will not load the bank correctly.

Example:
L1.1↵
V1.1=P1.1↵
I1.1,C1.1,C2.1↵
END.1

It is good practice to query the banks (?1000) after it has been sent to verify that it is loaded correctly.

4. Timing and execution

Timing of the logic bank is critical to certain applications and understanding how long a bank will take to execute may be important. For example: imagine you want to generate a sine curve with an output timing of 5ms. If a bank is created that takes 7ms the timing will be incorrect and the frequency of the output will not be as expected.

Typically steps take 250µs to execute giving 4 steps per millisecond. 1 step is considered any command other than operands, math symbols and carriage returns. These include the start of the bank.
Example step counts:

1 step	7 steps	4 steps
L1.1 END.1	L1.1 V1==P1,S0,CL2 END.1 L2.1 M0 END.1	L1.1 V1=V2+V3 END.1

In the example with 7 steps depending on the outcome of the branch V1==P1, S0 will be executed and the program will be considered 4 steps and take approximately 1ms to scan. Otherwise, CL2 will be executed and 6 steps taking approximately 1.5ms to scan will be the expected delay.

5. Relevant K-parameters

1. K85 – power-up bank execution

K85 sets the logic bank number to execute when the motor is powered on. Example K85=1 will allow logic bank one to start scanning immediately on power-up.

2. K87 – logic bank scanning frequency

K87 sets the scanning interval in milliseconds.
min = 1
max = 30000

If the scan time is shorter than it actually takes to scan the logic, the banks will be scanned as fast as possible. The entire bank will be scanned and not reset if the scan time is up.

6. Logic motion banks

Logic motion banks are a special form of executing logic banks that deal with a motion giving them a high priority. These banks execute on 1ms intervals and are independent of K87. They are executed with the command [LM1, [LM2, etc. They are programmed as standard logic banks. It is important that the bank scan time is not more than 1ms and as such must be kept to 4 steps. An example of using a motion bank is if the motor where to follow a polynomial or sine curve

Example:
V3 is set to the target position and equals the sine function U2(V1).

```
V3=”Dx”        /target position
V2=1           /incremental counter

L1
V1=V1+V2
V3=U2(V1)
END
```

This example does show 6 steps. However, in this case the timing is less than 1ms as depending on the type of step 4 steps take approximately 1ms.

7. Application Examples

1. Inputs – edges and levels

When the logic from an input is read it reads in the current state of the input. The logic bank needs to determine edges independently. Note: K28-K32 can still be used to detect edges so depending on the application it may be simpler and easier to use the K-parameters.

Depending on the status of the input 3 the system will do the following:

Low level: do nothing
Rising edge: start the motor.
High level: increase the speed of the motor by 1 count per scan
Falling edge: stop the motor and reset the speed.

```
v1=1,v2=2,v3=0,v4="Sx",v5=1

/bank 1 looks at input 3
/If it is equal to the
/previous value it is a level
/and if not an edge is detected.
/finally save the current state
/for the next scan.
L1.1
V3=I3
V3==v13,CL2,CL3
v13=v3
end.1

/bank 2 deals with an edge.
/If we have gone low
/to high start the motor
/otherwise stop the motor.
/when the motor is started
/it is started from the
/current speed.
L2.1
V3==v1,S0=v4:^.1,].1
End.1

/bank 3 checks the level.
/If the level is high increase
/the current speed by v5
/if it is low do nothing.
L3.1
V3==v1,S0=S0+v5,T0
End.1
```

2. Setting the analog input to control speed

If we assume that the resolution of the motor is 1000 pulses (K37=3) we can very simply control the speed by setting the speed (S0) equal to the analog input divided by 4. Since the analog input has a range from 0~1024 if we divide by 4 our speed will never exceed 256 pps (x100) and the resultant RPM is 1536. This is within the specification range of all the motors. The code will look like the following:

set the variables and registers correctly. The bank will scan every 1 ms (K87) and will automatically start on power up (K85). The ^.1 is not necessary to run every time but in this manner it allows the motor to run without any external 'go' command.

```
K37=3
K87=1
K85=1
V1="AIN4"
V2=4
S0.1=0
A0.1=50
P0.1=1000000000

L1.1
S0.1=V1.1/V2.1
^.1
END.1
```

3. Torque Examples

1. Torque – Position Control

Description:
For pure torque-position control where the overall torque of the motor is limited use the M0 command to limit the torque. In this manner the motor will always attempt to 'catch up' to its calculated position.

Uses:
A simple method to reduce the standard torque of the motor. Useful in applications requiring less torque then the motor outputs where the maximum torque could be dangerous or damage expensive equipment.

Additional info:
The user is expected to know some CML to use this example. The P, S, A and M given below are just an example and guideline any values may be used

Code:

```
P0=10000
S0=50
A0=50
M0=30
^
```

2. Bi-directional Torque Limit

Description:
This example shows how the ladder logic can be implemented to monitor the current/torque of the motor in both directions. The motor responds by an over torque limit by changing direction.

Uses:
When an immediate response to a torque limit is required, e.g. when the system collides or jams in a dangerous/expensive situation, the motor will immediately respond by backing off.

Possible changes:
In this code the motor will reverse direction other other options could be to stop the motor, disable the motor, execute a bank in a motor, etc.

Variables:
1.) V1 is the instantaneous Ix (current) of the motor.
2.) V2 and V3 are the current limits CW and CCW respectively. They can be increased or decreased.
3.) S1 and S2 are the speed the motor is to reverse at in opposite directions.
4.) t1 is included as a 250ms timer. This is really for this demo version where somebody would be testing with their hand. The timer could be reduced or removed depending on the application

Additional info:
The user is expected to know some logic bank and CML to use this example. Once the example is loaded the logic needs to be started [L1 and the motor should be run using the dynamic commands. The P,S and A given below are just an example and guideline.

Code:

```
K87=1
K85=1

.1

v1="Ix", v2=30, v3=-30,s1=-20, s2=20, t1=250

p=1000000000
s=0
a=100

L1
v1>v2,s0=s1:^:t1,CL2
END

L2
v1<v3,s0=s2:^:t1,t0
END
```

3. Bi-Directional Torque – Speed Control

Description:
This example shows how the ladder logic can be implemented to monitor the current/torque of the motor in both directions regardless of the running direction. This application of torque control will try to position back to the point of back drive as fast as possible.

Uses:
Limiting torque in any application particularly applications that will be back driven.

Possible changes:
Speed s2 (stop speed) could be changed from 0 to a negative speed. If S2=S1 then it would stop the motor from catching up to it's back drive position. However, you will notice the operation is slightly rougher as the motor oscillates around positive and negative speeds.

Variables:
1.) V1 is the instantaneous Ix (current) of the motor.
2.) V2 is the running direction current limit. It can be increased or decreased.
3.) S1 is the running speed and S2 is the stopping speed (S2=0)
4.) V3 and V4 are variables to hold values.
5.) M0 is the torque value which is set slightly higher than the Ix value. Note: the Ix value is from 0-K4 but M0 is a percentage of K4.

Additional info:
The user is expected to know some logic bank and CML to use this example. Once the example is loaded the logic needs to be started [L1 and the motor should be run using the dynamic commands. The P,S and A given below are just an example and guideline.

Code:

```
S=20,A=50,P=100000

S1.1=20
S2=0

V1="IX", V2=20, V3=0,v4=0,v5=5

B100
L100

L1
M0=V2+V5;
V3=V4-V2;
V1>V2,S0=S2,CL2;
END

L2
V1>V4,S0=S1,T0;
V1<V3,S0=S2,S0=S1;
END
```

4. Uni-directional Torque – Speed Control

Description:
This example shows how the ladder logic can be implemented to monitor the current/torque of the motor in both directions. However, the torque is only applied in the running direction. It essentially acts as a rachet type device where there is limit torque in one direction and cannot be back driven in the opposite direction.

Uses:
Spindle winding applications. Maintining tension in packaging,printing,rolling,etc applications.

Possible changes:
Current could be monitored in the other direction and if they exceed a set value the motor could disable, alarm, set an output, etc.

Speed s2 (stop speed) could be changed from 0 to a positive or even negative speed. A small negative speed would allow the motor to back drive as long as torque/tension is maintained. If S2=S1 then it would be simiar to the bi-directional torque-speed example.

Variables:
1.) V1 is the instantaneous Ix (current) of the motor.
2.) V2 is the running direction current limit. It can be increased or decreased.
3.) S1 is the running speed and S2 is the stopping speed (S2=0)
4.) V3 and V4 are variables to hold values.

Additional info:
The user is expected to know some logic bank and CML to use this example. Once the example is loaded the logic needs to be started [L1 and the motor should be run using the dynamic commands. The P,S and A given below are just an example and guideline.

Code:

```
K87=1
K85=1

S=20,A=50,P=100000
S1.1=20
S2=0

V1="IX", V2=20, V3=0,v4=0
M0=100

B100
L100

L1
V3=V4-V2;
V1>V2,S0=S2,CL2;
END

L2
V1>V4,S0=S1,T0;
V1<V3,S0=S2,S0=S1;
END
```