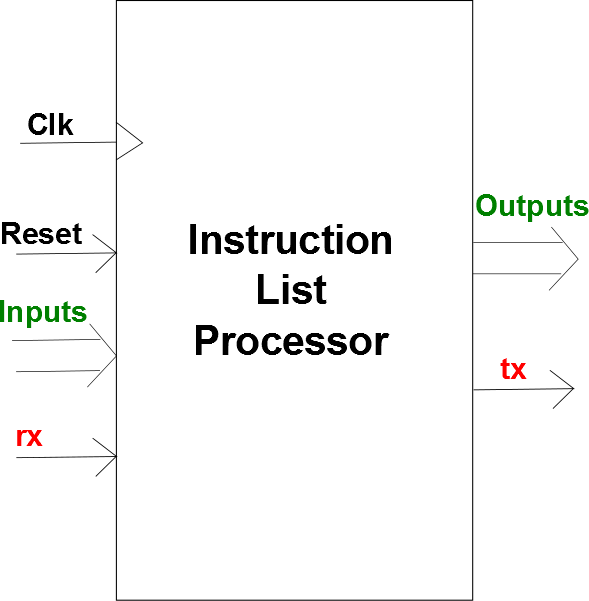
**INSTRUCTION LIST PROCESSOR**

**Introduction:**

The instruction list (IL) processor is designed for industrial automation computing purpose. In contrast to a general purpose processor, IL processor executes only instructions useful in typical PLC programs. The objective behind the design is limited instruction set eliminating any extra hardware not useful in PLC program execution, thereby increasing speed and further speed improvement with implementation of a three-stage pipeline.

This document addresses the detailed design considerations regarding the Instruction List processor. The top view of the core is shown below:



* Top level inputs:

**Clk** – clock signal

**Reset** – master reset signal to reset the processor

**Inputs** – up-to 128 inputs which are continuously updated in input register

**rx** – serial receive signal of UART

* Top level outputs:

**Outputs** – up-to 128 outputs from output register

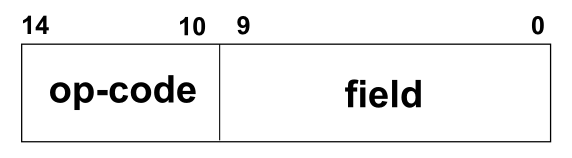
**tx** – serial transmit signal of UART

**Instruction Set:**

The instruction set is defined by referring to IEC 61131-3 standard, which defines what kind of instructions be implemented in the Instruction List language.

1. Branch instructions
   * END – end of IL program, unconditionally jump to start of program. Modifies PC content with start address given in instruction field.
   * JMP – if LSB of accumulator is logic 1, jump to address. Modifies PC in same way.
2. Data transfer instructions
   * LD – load the value, from input-register/output-register/bit-memory/byte-memory, in accumulator.
   * LDi – load the 8-bit value specified in instruction field in accumulator
   * ST – store the value, from accumulator to output-register/bit-memory/byte-memory.
3. ALU instructions
   * ADD, SUB, MUL, DIV – arithmetic operations. (MUL and DIV are not yet implemented)
   * AND, OR, XOR – logical operations, bit-by-bit.
   * GT, GE, EQ, LE, LT – comparison operations, affects LSB of accumulator.
4. Timer and Counter Instructions
   * PRE – loads preset value from accumulator to specified timer or counter
   * ETY – sets type and enables/disables the specified timer or counter
   * RST – reset the timer or counter specified in instruction field
   * LdTC – loads DN (done), TT (timer-timing), CU (counting-up), CD (counting-down) bits for timer/counter specified, to the accumulator.
   * LdACC – loads accumulated value (ACC) of specified timer/counter to the accumulator
5. UART instructions
   * UARTrd – read the received byte from UART
   * UARTwr – write a byte to transmit from UART
   * UARTstat – loads status register of UART to accumulator

**Instruction encoding:**

All the instructions of IL processor are of fixed-length of 15 bits. Any instruction can be seen to have 2 parts viz. op-code and field.

Upper 5-bits are instruction op-code from which each instruction is identified uniquely.

Lower 10-bits are instruction field which contains non-implicit data required to execute that instruction. The instruction field has different formats for various instructions.

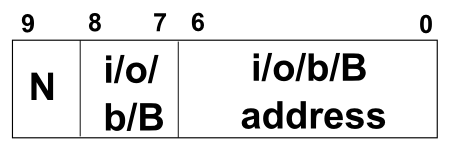
* **Branch instruction:**

For branch instructions viz. END and JMP, all 10-bits of instruction field are used to specify branch address. Thus, total 1 KB of memory can be addressed.

* **LDi instruction:**

Here, 2 MSBs of field are not used while 8 LSBs specify the immediate data to be loaded into accumulator.

* **LD, ST and all ALU instructions:**

For these instructions, the format of field is:

MSB is ‘N’ modifier bit specified in IEC 61131-3 standard. This bit defines whether the operand is to be in original form or negated form.

N = 1 means operand is negated.

e.g. LD N i0 ; load negated input0.

ADD N b2 ; add accumulator with negated bit-memory location -2 content

Next two bits are i/o/b/B.

i/o/b/B = 00 means operand is from input-register

= 01 means operand is from output-register

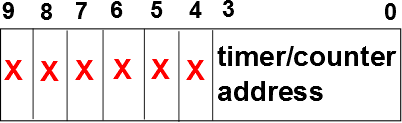
= 10 means operand is from bit-memory

= 11 means operand is from byte-memory

Last 7-bits specify the address of whatever i/o/b/B selected.

Thus, maximum 128 inputs, 128-outputs, 128 bit memory locations and 128 byte memory locations are addressable.

* **Timer/Counter instructions: PRE, RST, LdTC, LdACC**

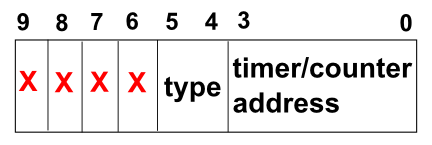


For these three instructions, only last 4-bits of instruction field specify the address of desired timer/counter; rest all MSBs are unused.

Addresses from 0000 to 0111 belong to 8 timers.

Addresses from 1000 to 1111 belong to 8 counters.

* **Timer/Counter instruction: ETY**



type:

|  |  |  |
| --- | --- | --- |
| **type** | **Timer type** | **Counter type** |
| 00 | On-delay timer | invalid |
| 01 | Off-delay timer | Up-counter |
| 10 | Retentive-on-delay timer | Down-counter |
| 11 | invalid | invalid |

* **UART instructions: UARTrd, UARTwr, UARTstat**

None of these instructions require any field details.

UARTwr instruction writes accumulator content to UART transmit FIFO.

**Modules:**

There are 3 functional units in IL processor, separated by 2 pipeline registers to bring simultaneity in their functioning.

Fetch Unit:

* Program Counter
* Instruction ROM

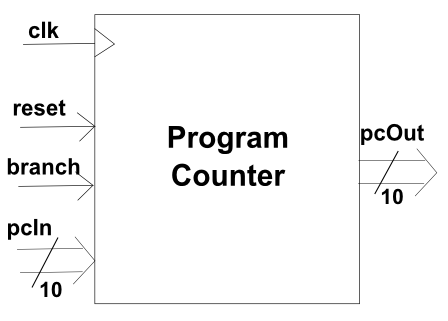
Decode Unit:

* Instruction Decode and Control Unit

Execution Unit:

* ALU
* Accumulator
* Accumulator multiplexer
* Operand2 multiplexer
* Bit and byte negators
* Memories (bit and byte addressable RAMs)
* Input and Output registers
* Timer and Counter peripheral modules
* UART modules

**Program Counter:**

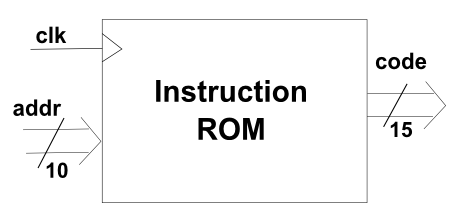


The PC is a loadable up-counter.

Upon reset, it starts from 0 and increments count on every rising edge of clk.

When ‘branch’ signal is asserted, 10-bit value of pcIn is loaded and PC starts counting from there.

**Instruction ROM:**

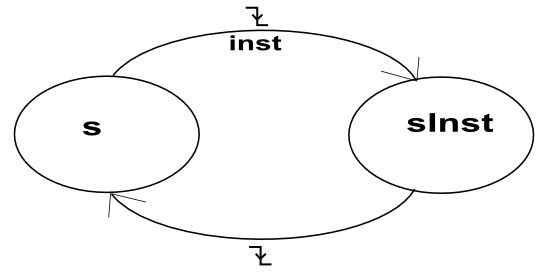
On every rising edge of clk, ROM outs 15-bit code located at memory address given by 10-bit ‘addr’ signal.

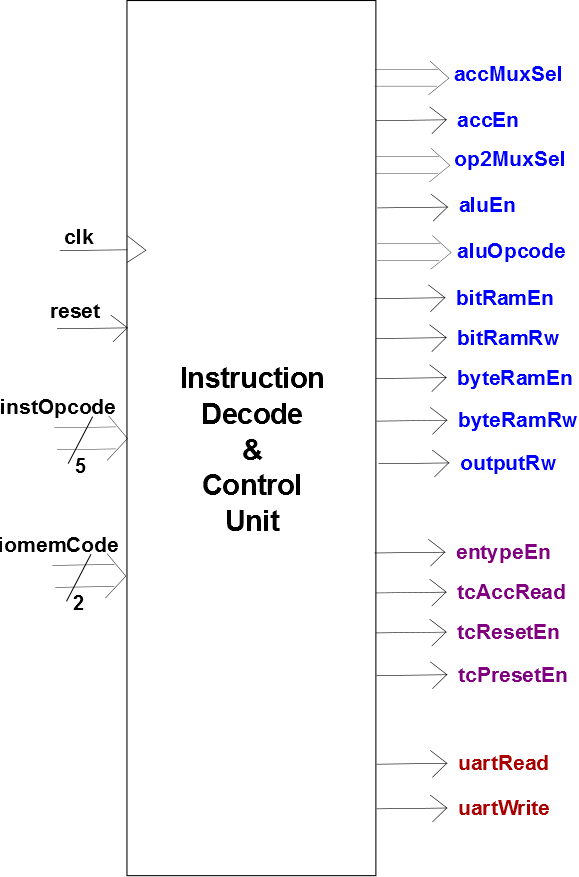
**Instruction Decode and Control Unit:**

This module is implemented as a finite state machine (FSM) which decodes the op-code coming from instruction ROM and generates control signals which drive the modules in execution unit.

The image on following page shows all the control signals.

This module is negative edge triggered.

It operates at double the clock frequency of the clock used for fetch unit i.e. for every single clock of PC, control unit has two clock cycles. This is required for the fact that, for every instruction, control unit needs to be in two states to complete its execution. The state diagram should clarify working of control unit:



When reset signal is asserted, FSM goes into state ‘s’, and stays there as long as reset is HIGH.

While in state ‘s’, the “instOpcode” input is read by the control unit and corresponding control signals are generated. Next state in this state depends upon the opcode.

e.g. for instruction LDi, accEn = 1, and accMuxSel is corresponding select code that selects immediate data from accumulator-multiplexer inputs. Next state assigned is sLd.

On next falling edge, FSM goes to state ‘sInst’.

The ‘sInst’ is not actually a state, but there are 7 states corresponding to ‘sInst’ depending on instruction. In this state, the control signals are rolled back.

e.g. for LDi, accEn is made 0.

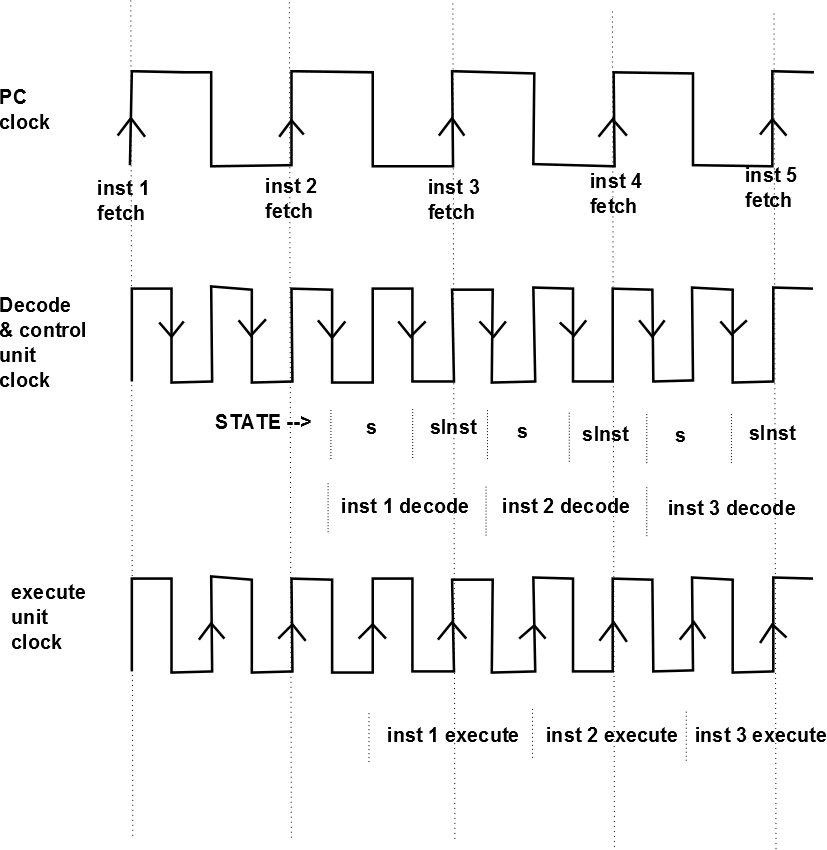
sInst may be one of the sLd, sSt, sAlu, sTc, sUart.

|  |  |
| --- | --- |
| **Instruction** | **sInst** |
| END, JMP | S |
| LDi | sLd |
| ST | sSt |
| LD, ADD, SUB, MUL, DIV, AND, OR, XOR, GT, GE, EQ, LE, LT | sAlu |
| PRE, ETY, RST, LdTC, LdACC | sTc |
| UARTrd, UARTwr, UARTstat | sUart |

For branching instructions, no signal is required from control unit, so its state doesn’t change.

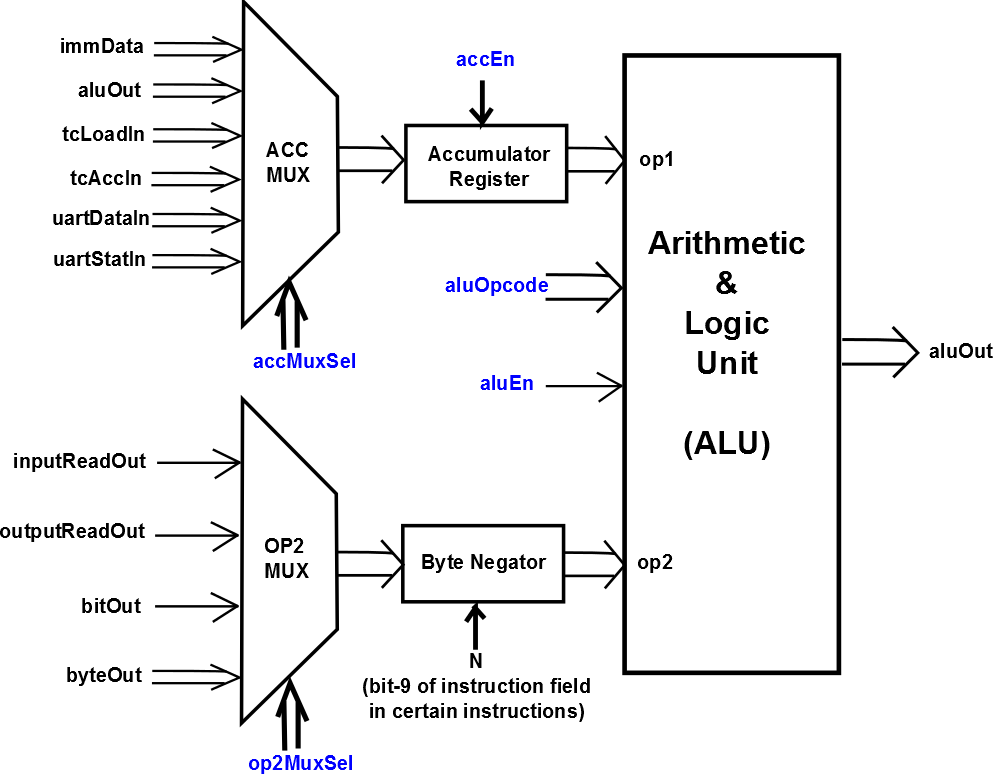
Whichever ‘sInst’ the FSM is in, it returns to state ‘s’ on next falling edge of clock. And again next instruction op-code is decoded.

The timing diagram on following page shows clocks used for three functional units and events with respect to those clock signals.



As seen in waveforms, control signals are generated on falling edge and passed to execution unit on next rising edge of same clock signal. The pipeline register holds those control signals meanwhile.

**Execution Unit modules:**



The data transfer instruction LD and LDi, all arithmetic, logical and comparison instructions are carried by the modules shown above.

Signals highlighted in blue are from instruction decode and control unit.

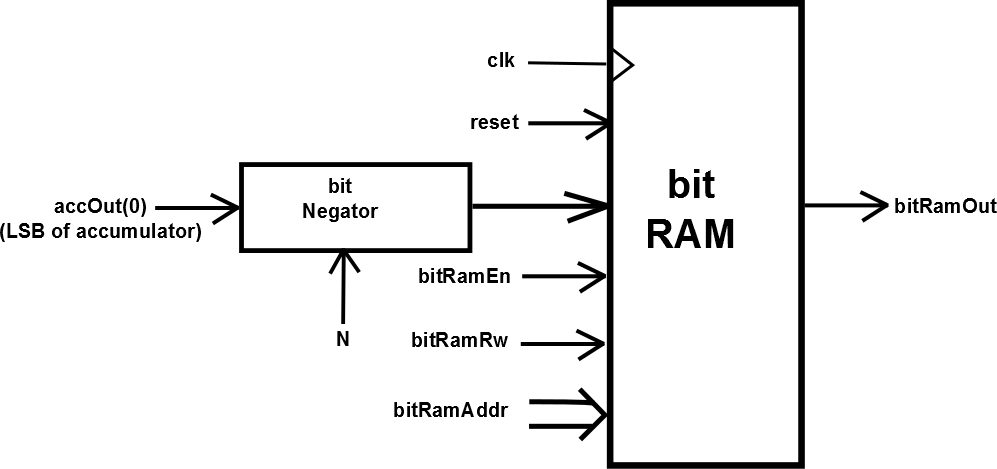
For LD instruction, appropriate ‘op2MuxSel’ is selected by decode unit based on its unit ‘iomemcode’ which are bit-8 and bit-7 in LD instruction used to choose i/o/b/B.

For instruction LDi, all arithmetic, logical and comparison instructions, timer/counter instructions LdTC and LdACC, UART instructions UARTrd and UARTstat, an appropriate value of ‘accMuxSel’ is selected by instruction decode and control unit.

Byte negator inverts its input if N=1 i.e. if operand is to be negated in LD or ALU operations.

**Memories and I/O registers:**

Two memories are used the design, to be used as scratch-pad area.



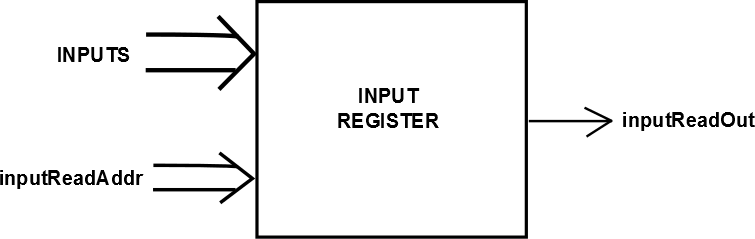
The bit-negator used prior to RAM negates input of RAM if N=1 i.e. when operand is to be negated in ST (store) instruction.

The byte RAM is same, except that byte-negator is used whose input is accOut, and that input and output of RAM are a-byte wide.

Only accumulator register can write to both these memories, using ST instruction.

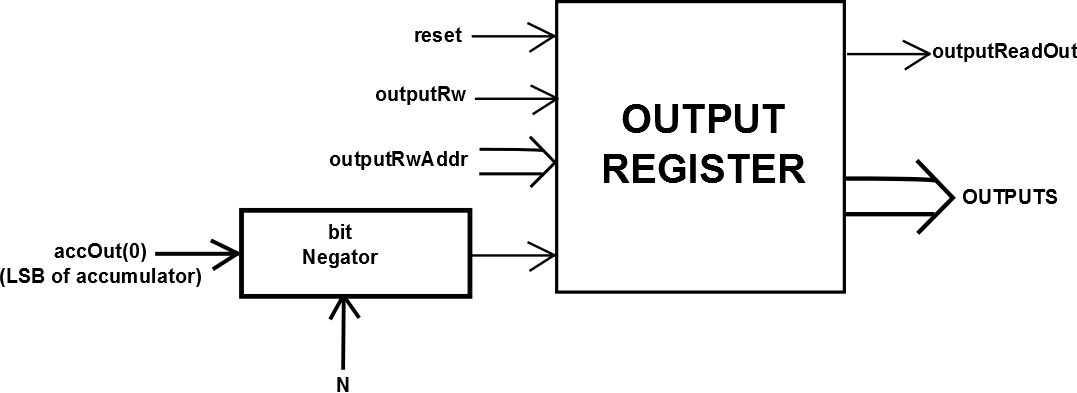
Memories are read using LD instruction.

Input and output registers are important components of the design, as most the PLC instructions involve reading inputs and writing outputs.



‘INPUTS’ is external input to the design, which is continuously updated in the input register. It outputs the input present at address pointed by ‘inputReadAddr’. This output appears at OP2 MUX.

The address ‘inputReadAddr’ is 7-bit and hence, maximum 128 inputs can be addressed.



Output register can be read as well written by LD and ST instructions respectively. Read/Write is controlled by ‘outputRw’ input from instruction decode and control unit.

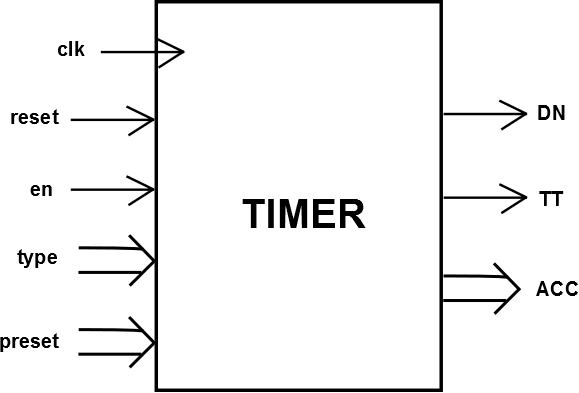
‘outputWriteIn’ is LSB of accumulator.

There are two output ports of this module. ‘outputputReadOut’ outputs a single ‘output’ out of all 128, located at address pointed by ‘outputRwAddr’. ‘OUTPUTS’ is external output of the design which updates external outputs of PLC.

**Timer and Counter module**

Timers and Counters are very important entities required in industrial automation environment. A timer and a counter used in PLC is different than timer and counter implemented in general purpose microcontroller systems.

TIMER:



A timer is of 3 types (or, it works in 3 modes):

* On-delay timer
* Off-delay timer
* Retentive off-delay timer

A timer has

* A ‘PRESET’ value (PRE)
* An ‘ACCUMULATED’ value (ACC)
* A ‘timer-base’

Note: Timer-base functionality is not implemented in the design. It is used to set resolution of timer. I assume that timer works with only one timer-base.

On-delay timer:

* On reset, or when ‘en’ is LOW, the ACC value within timer is reset to 0.
* When ‘en’ is HIGH, on every rising edge of clock, ACC value increments until it reaches PRE.
* When ACC = PRE, the output DN (done) is raised HIGH and ACC value it hold as is.
* The output TT (timer timing) is raised HIGH as long as ‘en’ is HIGH and timing operation is in progress. It goes LOW when DN goes HIGH.
* The output ACC always indicates current value of internal register ACC.
* If at any instant ‘en’ goes LOW, ACC is reset to 0 and timer stops timing. All outputs in this case are 0.

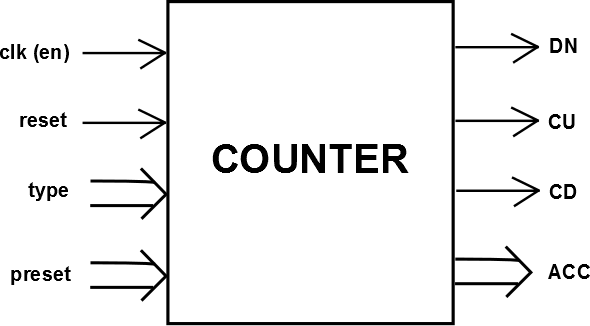
Off-delay timer:

* This timer is activated by rising edge of ‘en’ input, but timing does not start yet.
* The ACC starts incrementing after the ‘en’ goes from HIGH to LOW.

This mode is not implemented in the design correctly. Needs to be re-designed!!

Retentive on-delay timer:

* This works exactly same as on-delay timer, except that value of ACC is retained if ‘en’ goes LOW while timing operation is in progress.
* The only way to reset this timer (to make ACC = 0) is using reset input. RES instruction is used.



COUNTER:

The counter has same functionality as that of timer.

Here, ACC increments/decrements on rising edge of clk (or en) signal.

Once ACC crosses PRE value, it keeps incrementing/decrementing so that events can be measured beyond certain threshold.

For up-counter, ACC starts from 0, while for down-counter, ACC starts from highest possible value.

The preset value used in design is 8-bit and so is ACC. The value is loaded from accumulator register of the processor core (not to be confused with ACC register).

A typical timer program rung looks like:

The notation used to indicate input differs in software tools used for various PLCs. Let’s assume I:1/2 used here corresponds to I2 in our case.

The meaning of rung is, if I2 is active (logic 1), then timer T4 is used as ON-DELAY timer with preset = 5.

IL program could be written as:

LD I2 ; load input2

ETY (0001000111) ; field values are En = 1

; Type = 00

; T/C addr = 0111

LDi #5 ; load immediate value 5 in accumulator

PRE T4 ; set the preset value for timer T4

The DN shown in ladder does not correspond to IL instruction in this rung. It might be used in other rung to turn some output ON or OFF.

The above four lines of IL code should start the timer. To turn it OFF, ETY instruction must be used with En bit 0:

ETY (0000000111)

Similarly, counters can be programmed.

IL code:

LD I0 ; load input0

ETY (0001011000) ; assuming counter0 in up-count mode

LDi #5 ; load immediate data 5

PRE C0 ; set the preset value for counter0

For working of timer and counter, following must be specified:

‘en’ ‘reset’ ‘type’ ‘preset'

This values are given as inputs to each timer and counter implemented. These must be stored somewhere. Following modules are implemented for holding them:

tcEnableAndTypeModule : works for ETY instruction

tcResetModule : works for RES instruction

tcPresetModule : works for PRE instruction

to read outputs of timer/counter

tcAccumModule : works for LdACC instruction

tcLoadModule : works for LdTC instruction

**Complete TIMER & COUNTER module**

