Mitsubishi Industrial Robots

ECT 661
Robot Programming using the Melfa Basic language; and the RT ToolBox2 application software.

RT ToolBox2

RT Toolbox is MELFA's programming Interface software. MELFA is:

Mitsubishi **EL**ectric **F**actory **A**utomation

RT ToolBox is used to:

1. Write programming code
2. Save taught positions
3. Communicate with the robot
4. Download & run the program
5. Simulate external Input & Output operations

Reference Materials on BB

MELFA BASIC IV Programming Guide,
MELFA BASIC V Programming Guide, &
(The command s are almost exactly the same)
RT ToolBox2 Software manual & a ‘short form”

These manuals have far more comprehensive coverage of how to use MELFA BASIC commands than are show here. This presentation is just a brief introduction.

There are a number of other commands & options in programming than just what are shown here. Use the hardware & software manuals as your best guide, not the slideshow.

ECT661 Course Overview

The purpose of this course is to teach the structured programming design of industrial robots. You will use 5 or 6 axis Articulated Arm robots manufactured by Mitsubishi (Series A, S. or F). The software is MELFA ToolBox (either Melfa IV, or Melfa V).

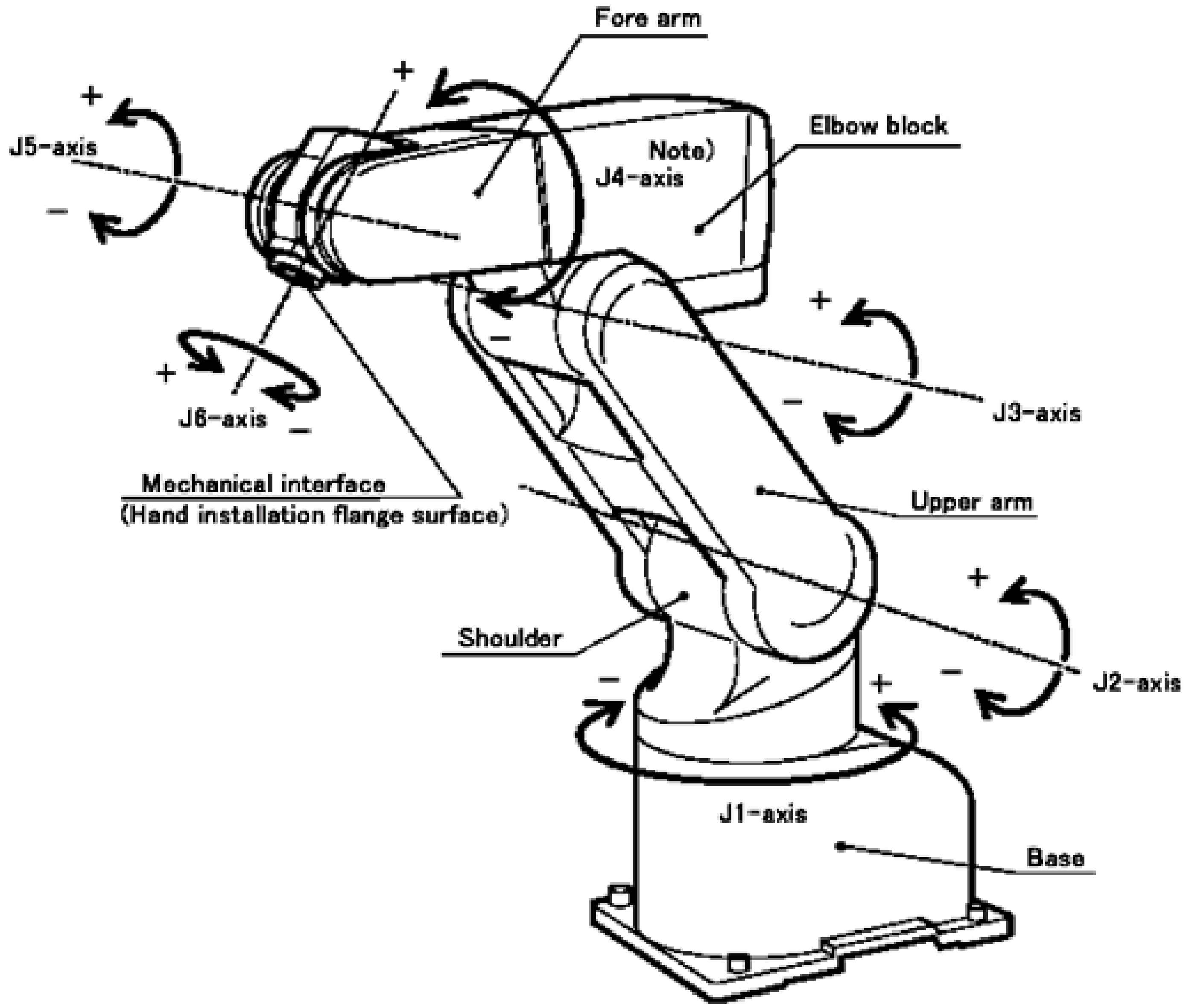
You will complete a series of exercises which increase in complexity, and then develop a final project to demonstrate and document.

To be successful given the amount of time available you will need to work outside of class.

Course Practices

- Students are required to wear facemasks or shields when in the classroom.
- Please clean your robot area when done
- You may work in lab groups of two
- Each group is expected to create a unique solution to each laboratory problem, not copy programming code from someone else.
- All programs must be commented, and demonstrated to the instructor for credit
- This is a stressful time, let’s all be kind and considerate with each other.

Mitsubishi Robot Architecture

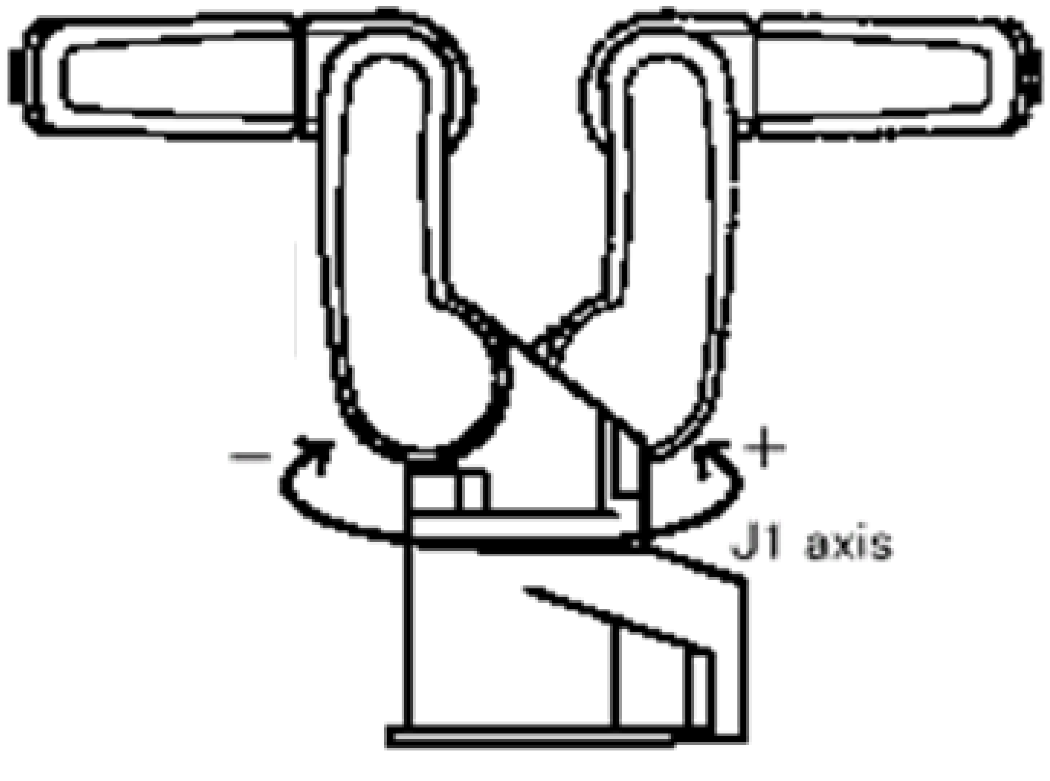


Cartesian Coordinates

- Cartesian coordinate system is a method of measuring special relationships between objects. Cartesian coordinates are measured in three dimensions along the X, Y, and Z axes.
- Movement can be Positive or Negative
- In relation to robotics, Cartesian coordinate movement refers to the movement of the robot's arm along the X, Y, and Z axes.
- JOINT JOG Mode moves each axis individually along a Cartesian Path (only the one Axis moves)

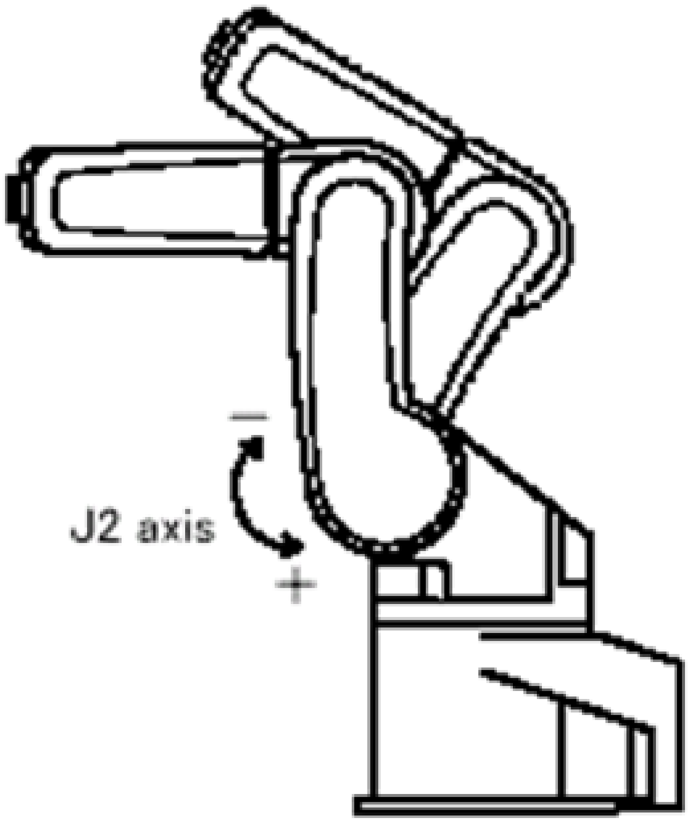
Joint Jog Mode-Waist

- Waist (J1): The most obvious freedom is the rotating motion around the *waist* (J1) of the robot.



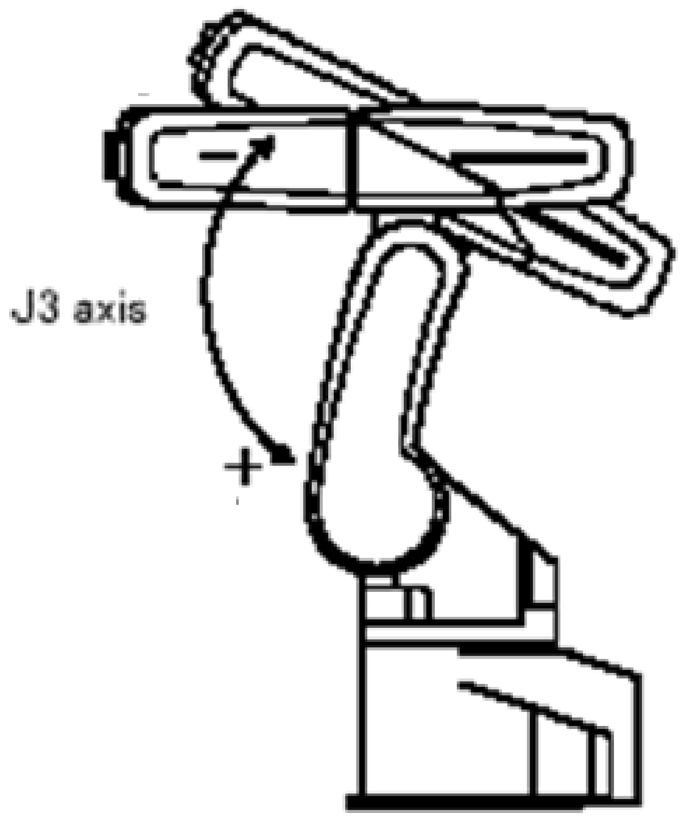
Joint Jog Mode-Shoulder

- Shoulder (J2): Above the waist, is the *shoulder rotation* of the robot. The shoulder connects the waist to the upper arm. This axis allows the arm link to move up and down.



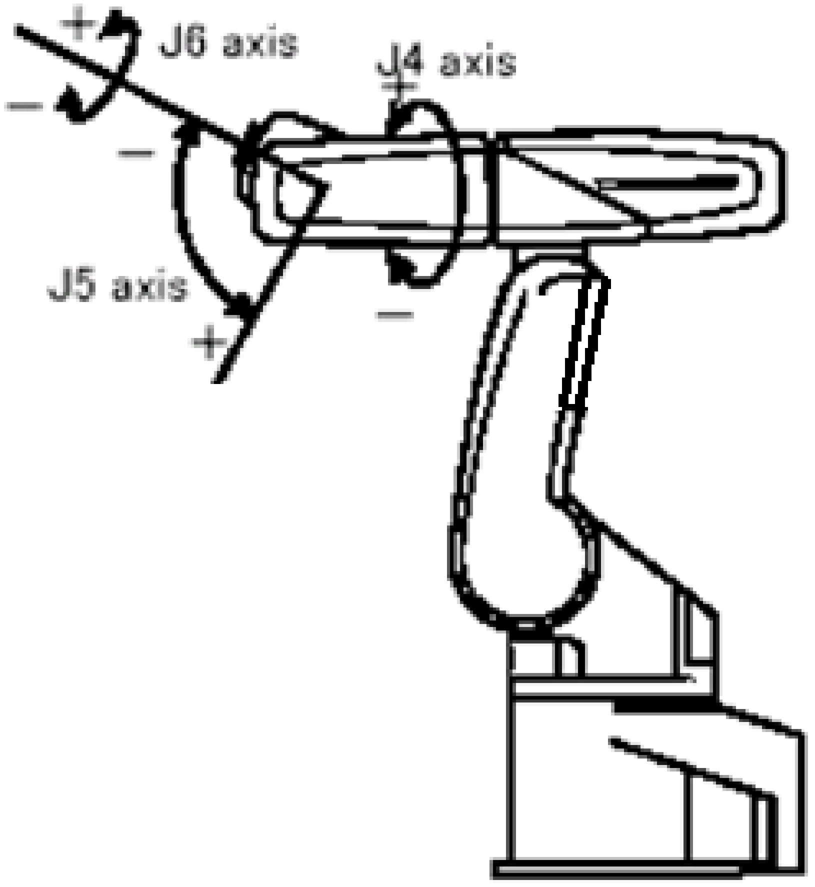
Joint Jog Mode-Elbow

- Elbow (J3): The next link is the *elbow rotation*. This link connects the upper arm and the forearm and allows up and down movement.



Joint Jog Mode- Wrist Axes

- Theta or Yaw (J4), Pitch (J5) and Roll (J6): The Theta (J4) is used only in a 6 axis robot. The two wrist axes permit up and down & rotary motion. The wrist up & down motion is pitch (J5) & the rotating motion is the wrist roll (J6).



XYZ Jog Mode

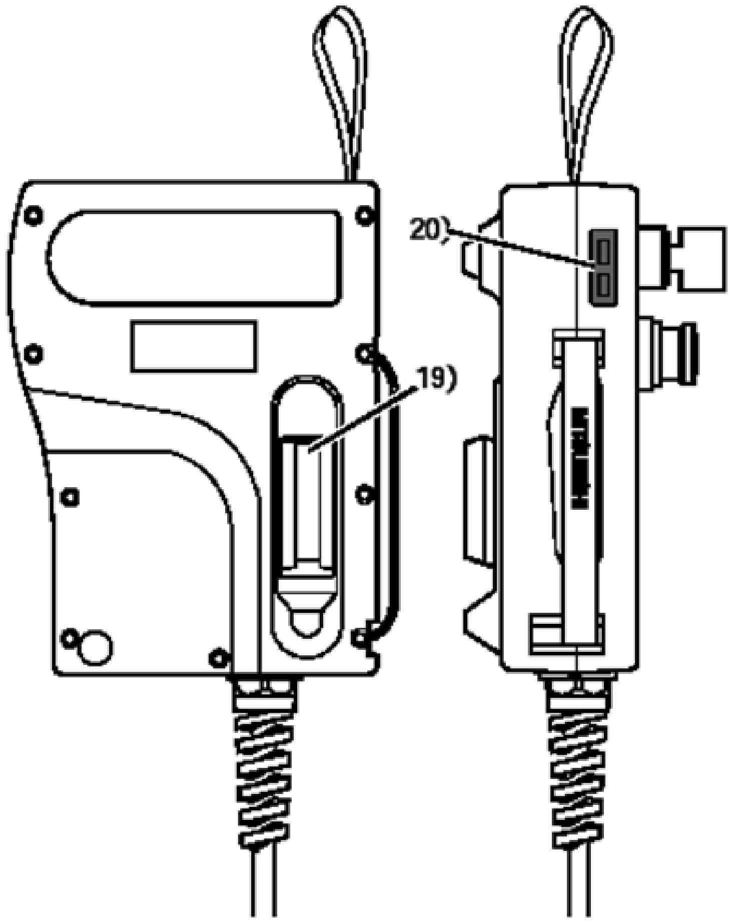
- What is the difference between (Joint Jog) and (XYZ) movement? -- With XYZ movement, the joints to move **together** in an articulated manner where as Joint Jog moves only one joint at a time.
- In XYZ mode the joints cooperate to move the end of the robot arm in a Cartesian path.
- As a general rule: Use Joint mode to move to a rough position, then switch to XYZ mode to fine tune the position before saving.
- Joint mode moves faster than XYZ

Tool Jog Mode

- Tool movement allows you to jog or move the robot along the path of the tool that is inserted in the end effecter.
- This command is useful because of its ability to move in the direction in which the robot’s hand is pointed.

Teach Pendant

- Hand held programming device
- Allows the operator to move the robot & operate Hand
- Can set the speed, & mode
- Has a “deadman” switch
- Cannot be operated when control is given to the controller.
- Size & function vary by maker & Model



Categories of Variable

- Numeric Type *(Starts with M)*
- Character String Type *(Starts with C)*
- Position Type *(Starts with P)*
- External Variables always have a underscore as the 2nd character of the name.

  Example: all I/O Type M_IN( ) or M_OUT( ) use the _ as they are external to memory.

Overview of Variable Types

- System Variable: Predetermined by the variable name and the saved data.
- System Control: Only can be referred to with the program.
  Example: P_CURR is the current position of the robot arm   100 P10=P_CURR
- User Control: This can be referred to and substituted in a program.
  Example: M1 = M_IN(20)
  Variable M1 = 0 or 1 depending upon the state of Input #20 (on or off)

Differences in Robots

- The A series robots use **Melfa IV** language with <u>line numbers</u> and you can reference a jump to a specific line number, OR a label (starts with an *)
- The F series robots use **Melfa V** language which uses <u>steps</u> not line numbers, you can reference a jump to a label.
- Each robot has a prepared startup file for it, use this and rename to your file name and save your file to a USB device.

Lines vs Steps

- The Mitsubishi A series Robots operate using Melfa IV language which requires LINE NUMBERS.
- The Mitsubishi S & F series Robots operate using Melfa V language which uses STEPS.
- Usually this makes no difference in the operation of the program but will affect some advanced commands where it is necessary to do a Jump operation.
- Max number of characters per Line/step =127
- Max number of lines/steps per program =32,767

Robot Command Structure

```
  10    MOV P1  With M_OUT (17) = 1
  (1)    (2)     (3)                 (4)
```

- (1) Line #: Number used to determine the program execution order. (OR the Step number)
- (2) Command Word: This is a command that designates the robot operation or work.
- (3) Data: This is data, such as a variable, a position, or value, required for the command
- (4) Appended or conditional Statement: Used as necessary to modify the action of the statement.

Saving Positions

- Use the teach pendant to physically jog the robot arm to the desired position
- In the software interface select the positions window, and click ADD
- In the popup box create a unique name for the position starting with P (P10, Phome)
- Use the button at the right hand side of the box to get the current axis positions from the robot.
- Save position, confirm it shows in list

Program Sequence

- Most robot programs use line numbers, or Step numbers to determine the order of execution & memory management. A series uses manually inserted lines, S & F use program inserted Steps
- It you are using line numbers it is a good practice to set the start line at 100, and set the line pitch (the gap) to 10.
- You can reorder the numbers later if you need to to make the program look neater. If you are using steps the program automatically reorders the steps as you insert a new line of commands.

Standard Movement “MOV”

- The robot moves to the designated position with Joint Interpolation.
- The Robot chooses the best path from where it currently is to where it is told to go.
- MOV P1   Moves to P1.
- MOV P1 WTH M_OUT (17) = 1 Starts movement toward P1. And simultaneously turns output signal bit 17 ON.
- MOV P1, -50   Moves from P1 to a Z Offset position retracted 50mm.
- If a calculated position exceeds the Work Envelope the robot arm will halt and error (Beep)

Linear Motion “MVS”

- The robot moves to the designated position with Linear interpolation. Move Straight
- MVS is commonly combined with a MOV to act as an “Approach” where you move to a set point in mm above an object using MOV, then use a MVS down to settle over the part.
- MVS P10 Move straight to Position 10
- MVS, -50 Move up 50mm from current
- Positions start with “P” and can either be a name “Phome” “Psafe” or a number P100, P25 or a combination “Psafe2”

Movement: Angular Motion

- MVR P1, P2, P3 Move with radius through P1 ⇨ P2 ⇨ P3
- MVR2 P1, P2, P3 Move with radius through P1 ⇨ P3, P2 is reference
- MVC P1, P2, P3 Move in a circle P1 ⇨ P2 ⇨ P3 ⇨ P1
- MVR is used to make radial curves
- MVC is used to make circles

Gripper Operation

- HOPEN 1 This opens the gripper specified
- HCLOSE 1 This Closes the gripper specified
- The Robots we are using can have up to 4 grippers installed on them. We only have one, so always use the “1” designation for the hand
- Remember that the hand does not open or close instantly, use a time delay command (~0.5 seconds) immediately after a open or close command.
- Hand open/close times may vary by robot

Speed & Override

- Speed (SPD) sets how fast the hand moves in mm per second
  SPD 20   ‘ Move the hand at 20mm per second
- OverRide (OVRD) sets the speed as a percentage of the robot’s maximum speed
  OVRD 50 ‘Move the robot at 50% maximum
- OVRD can be used several times to change the speed as needed in the program.

Timing Delay “DLY”

- When active the robot will pause in place for a specified amount of time
- DLY is commonly used when grasping an object to allow a wait time for the robot arm to stop.
- Dly 1.0   Delay, robot waits 1 sec
- Dly 0.5   Delay, robot waits 500ms
- Delay can be used in combination with other commands as an appended action:
  M_OUT(10)=1 DLY 1.0
  Turn on Output #10 for one second

End of Program “END”

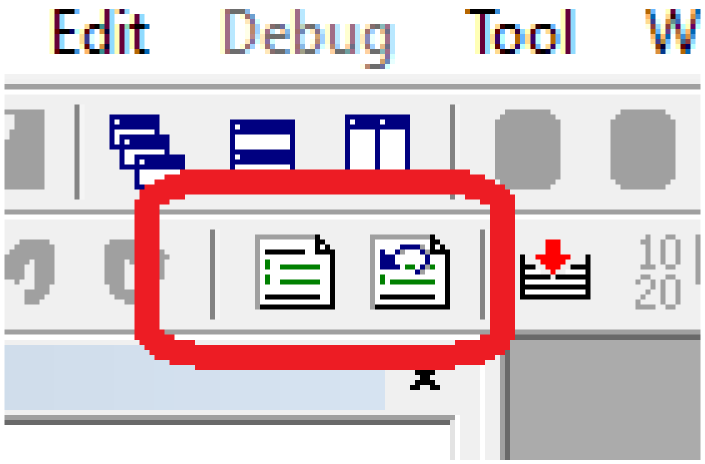
- Designates the last line of the main program
- When END is reached the program will either recycle to the first line and run again, OR if the operator has set the controller to a single cycle the program will stop
- Lines may be added after the END command but these are not run as the program. The program must use a Jump command to go to these rungs and then return to the main program area (this is for subroutines).

Special Characters

- **Underscore** (_): Used for the second character of an identifier (variable name) to identify the variable as an external variable (input or output). M_IN(6), M_OUT(12)
- **Apostrophe** ('): Used at the head of all line comments. When assigned at the first character it is a substitute for a remark line.
- **Comma** (,): Used as a delimiter when there are several parameters or suffixes to a command
- **Asterisk** (*): Placed in front of labels used for the destination of a jump or subroutine.

Program Comments

- Steps/Lines of code can be commented using the ‘ character before the comment
- Comments can be appended to the right of a statement → 10 Mov Phome ‘Move to home
- Whole lines can be a comment
  10 ‘This starts main program loop
- Comments can be inserted or removed using the toolbar icons



Comparison Expressions

• If M1=12 Then *L3	Equal to
• If M1<>2 Then *L3	Not Equal
• If M1< 10 Then *L3	Less Than
• If M1>9 Then *L3	Greater Than
• If M1=<10 Then *L3	Equal to or Less than
• If M1=>11 Then *L3	Equal to or More than

Mathematical Function

• Add	P10=P1+P2	M1=M1+1
• Subtract	P5=P4-P8	M1=M1-1
• Multiply	P1=P10*P3	M1=M1*5
• Divide	P1=P10/P3	M1=M1/2
• Remainder		M1=M1 Mod 3
• Exponent		M1=M1^2
• Negate	P1=-P1	M1=-M1

Substitution (Move) Function

• P1=P5	P1 is given the position of P5
• P5=P_CURR	P5 is given the current position data
• M1=M2	M1 has the value of M2
• M1=65	M1 is given the value 65

Program Control of a Robot

- Robotic program controls are similar to those found in standard programming languages.

- Branching Commands
  Unconditional & Conditional
- Repetition (Loops)
- Subroutines
- Interrupt Calls
- Suspend or End operation
- Input & Output operations

Unconditional Branching: GOTO

- Jumps unconditionally to the designated point in memory (line # or *Name) called a LABEL
- GOTO does not create a return path back to the jump point. You have to use another GOTO.
- Jumped over lines are NOT interpreted.
- Examples:
  10 GOTO 600 -- Jump to Line 600 (V4 only)
  10 GOTO *Here – Jump to label named *Here
- GOTO can jump past the END command.
- GOTO can make the program crash if not used correctly, or cause an endless loop to occur.

Rules for Making Labels

- Begins with the * character
- Can be up to 16 characters MELFA V
  Can be up to 8 characters MELFA IV
- Must be unique (can’t have 2 labels with the same name in different places)
- Cannot use a reserved command word

ON … GOTO

- ON <condition> GOTO Jumps **conditionally** according to the value of the designated variable. The value conditions follow the integer value order.
- ON M1 GOTO *LBL1 --
  IF Variable M1=1, THEN Jump to Label *LBL1

- You can use multiple values with an On…GOTO
- ON M1 GOTO *L1, *L2, *L3

  If M1=1 jumps to *L1, if =2 jumps to step *L2, and if 3 jumps to step *L3. If none of the listed values are true execution proceeds to next step.

Jump to Subroutine & Return

- Subroutines are placed after the END statement. (you jump to the sub and then back to the main)
- UNCONDITIONAL Jump to Subroutine command (10 GOSUB *Label)
- Return from Subroutine = RETURN

-
  ```
  100 GOSUB *Sub1   -- Jump to label *Sub1
  110 MOV P10        ← (RETURN comes to here)
  120 END
  500 *Sub1
  510 DLY 5
  520 RETURN       -- Go to step after Jump to sub
  ```
- RETURN always goes to the line after the jump

ON … GOSUB

- ON <condition> GOSUB Jumps to the designated subroutine **conditionally** according to the value of the designated variable. The value conditions follow the integer value order.
- ON M1 GOSUB *Sub1
  When Variable M1=1, Jump to Subroutine *Sub1.
  You must include a RETURN to end the subroutine
- You can use multiple values with an On…GOSUB
- ON M1 GOSUB *SUB1, *SUB2, *SUB3

  If M1=1 jumps to *SUB1, if =2 jumps to step *SUB2, and if 3 jumps to step *SUB3. If none of the listed values are true execution proceeds to the next step. (Return comes back to the same line)

Nested Subroutines

- A NESTED Subroutine is a subroutine called by another subroutine.
- MELFA BASIC allows you to nest programs 8 levels deep.
- A Nested subroutine when ended returns to the subroutine which called it in the order designated by the Stack Pointer register.

Program Call

- One robot program can CALL another program using the CALLP command. The Called program should have a name longer than 4 characters so that it cannot be accidentally run from the controller.
- 10 CALLP “LAB006”
- Up to 16 Parameters can be passed between programs but you must use the “From Program” Command to store the variables in the called program.

Called Program with Variables

10. CallP “Lab006”, M1, M2, P01
    Call a program named Lab006 and pass M1, M2, and P01 to that program

Lab006 Sub program
10. FPRM M10, M12, Phere
Copy the 3 variable values sent from the calling program to new variables in memory used by the called program

Note: Two <u>different</u> programs can have the same variable names (like M1 or Phome) because they occupy different memory segments.

INTERRUPTS

- Allows the programmer to create a ‘priority” situation that overrides normal execution.
- The Interrupt has to be defined before it is called (what makes the interrupt occur, what action is taken by the interrupt).
- The Interrupt can be turned on or turned off multiple times in a program. This way you can set up points where the interrupt doesn’t take place even if the condition is true.

DEF ACT: Define an Interrupt

- Interrupts allow the normal program sequence to be interrupted to perform a specific task when it occurs, rather than have the robot “wait” for something to occur.

```
20 DEF ACT 1, M_IN(17)=1 GOSUB *SUB_1
         (1)           (2)              (3)
```

(1) Defines the action as ACT 1

(2) If Input #17 goes True (=1) Then

(3) Jump to the Subroutine at Label *SUB_1

Program Interrupts (ACT)

- To use an interrupt it must first be defined (Def ACT) and the Act must be enabled when it occurs,

```
40 ACT 1=1        Enables ACT #1
50 ACT 1=0        Disables ACT #1
```

- Enabling & Disabling manually ONLY allows the ACT to operate IF it is called by an outside condition defined in the Def Act.
- The ACT command only functions as a switch, it does not actually cause the ACT to occur.

Options in an Interrupt

- DEF ACT, M_In(9)=1 GoSub *Sub1
  --If Act #1 is enabled, and Input #9 goes True the robot will decelerate to a stop even in the middle of a instruction, and program execution will go to SUB1
- DEF ACT, M_In(10)=1 GoSub *Sub2**, L**
  -- same as above but when the **L** is appended to the end the statement currently being executed will complete first, then execution will jump to Sub2

Program Interrupt (RETURN)

- Return is used with a call to a subroutine to go back to the main program, but can also be used with an interrupt call if a condition is added.
- Return with no condition is used with a GOSUB
  520 RETURN
- Return with a condition is used with an Interrupt jump to a subroutine:
  520 RETURN 0 --
  Returns to the line of the interrupt Call & runs that line again.
  520 RETURN 1 --
  Returns to the first line after the interrupt call.

IF-THEN-ELSE (one line)

- Program jump only if a specified expression is true.
- Also allows and alternative if the expression is false.
- 100 IF M2<12 THEN *L1 ELSE *L2
  If M2 is less than 12 then jump to step *L1,
  if M2 is not less than 12 then jump to Step *L2.
- The statement should be placed on one line
- If an “ELSE” is not included the program on a false condition will proceed to the next line of code in the program.

```
100 IF M1=10 THEN *L3
110  MOV PHOME
```

IF-THEN-ELSE-ENDIF (Multiple Line statements)

- Acts similar to the IF-THEN-ELSE but allows for a statement set to be included rather than a jump.

-
  ```
  100 IF M1>12 THEN
  110 M2=12
  120 M3=12
  130 ELSE
  140 M1=0
  150 ENDIF
  ```
- Do not use the GOTO within an IF-THEN or a memory error will occur.

SELECT-CASE Statements

- Jumps according to the designated variable and the designated conditions of that value.
- Do not use a GOTO from this command (Error)
- Multiple types of conditions can be designated per command statement.
- Allows you to create a mixed menu of options to choose from.

SELECT-CASE Statements

- Basic format of Select Case statements
-
```
SELECT <condition>
CASE <expression>
<Process>
BREAK
CASE <expression>
<Process>
BREAK
DEFAULT
<Process>
BREAK
END SELECT
```

```
10 SELECT M1         (examine the value of "M1")
20 CASE IS <=10      (If M1 is <=10)
30 MOV P10           (Then do this)
40 BREAK             (Jump to End Select & continue)
50 CASE IS 11        (If M1 is = 11)
60 MOV P2            (Then do this)
70 BREAK             (Jump to End Select)
80 CASE 13 TO 18     (If M1 ranges from 13 TO 18)
90 MOV P4            (Then do this)
100 BREAK            (Jump to End Select)
110 DEFAULT          (If none of the cases are true)
120 MOV PHOME        (Then do this)
130 BREAK            (Jump to End Select)
140 END SELECT       (End of Select Statement)
```

Repetition: (FOR … NEXT)

- Repeatedly executes the program lines between the FOR and the NEXT statements according to the condition specified at the start.
- FOR NEXT is used when you want to loop a <u>specified</u> number of times,
- 20 FOR M1= 1 to 10 Inc Counter by 1
  30 MSUM = MSUM + M1
  40 NEXT M1 Return to Line 20
- FOR NEXT loops can be Nested (loop in a loop)
- You can use a variable value to control the number of repeats → For M1= 1 to M2

Repetition: (WHILE … WEND)

- Looping command that is repeated for an un-specified number of passes until a specified condition is met
-
```
20 WHILE M1<= M2      repeat until M1>M2
30 M1 = M1+1          add 1 to M1 each pass
40 WEND               return to Line 20
```
- It is possible to get into an endless loop using control procedures in a robot (just like in a PLC) but there is no watchdog timer to stop the robot, it just keeps repeating instructions.
- If the initial condition is already TRUE, the While-Wend Loop will not execute (skipped over)

WAIT Statement

- Waits until a specified condition is True
- Used to interface the robot with other devices
- 100 WAIT M_IN(1)=1 (wait till Input 1 is true)
  100 WAIT M_OUT(2)=0 (wait till output 0 is false)
  100 WAIT M_TIMER(2)>30000 (Wait until
  Timer #2 is greater than 30 seconds)
- WAIT can be used interchangeably with the IF-THEN-ELSE command in some cases

Make a light Blink while Waiting

-
  ```
  1  WHILE M_IN(?)=0
  2  M_OUT(?)=1 DLY 0.5
  3  DLY 1.0
  4  WEND
  ```
- Until the indicated Input is made TRUE the while-wend loop will continue with the indicated Output blinking on-off in half second increments.
- Can be used in place of WAIT to alert operator the Robot is running but waiting for something to happen

Halt (HLT)

- This instruction stops the robot and pauses the execution of the program. When the program is started, it is executed from the next step.
- 150 Hlt   Interrupt execution of the program.

- 120 If M_In(20)=1 Then Hlt
  Pauses the program if input signal bit 20 is turned on.

- 210 Mov P1 WthIf M_In(18)=1, Hlt
  Pauses the program if input signal bit 18 is turned on while moving toward P1.

Skip (SKIP)

- This command is appended to cause an instruction to be ignored if a specific condition is TRUE, and continue with the program execution.
- 110 Mov P1 WTHIF M_In(18)=1, SKIP

  Stops the move to P1 IF Input signal bit #18 goes TRUE during the move, and skips forward to the next command. (The robot does not stop)

Appended Statement

- When the robot moves you can add, or append, a co-requisite action using

  1. WTH (with): adds an unconditional action to the movement, or

  2. WTHIF (with-If): adds a conditional action to the movement

  The command is W T H, not WITH.
  (You will get a syntax error)

Conditional Action to a Move

- WTH adds an unconditional action to the move.
- 10 MOV P50 WTH M_OUT(10)=0
  Move to P50 and Turn off output #10
- 30 MOV P1 WTH M_OUT(6)=1, DLY 1.0
  Move to P1, and turn on Output #6 for one second

- WITHIF (With an IF) is a conditional action which must be true for the Move to occur.
- 20 MOV P30 WTHIF M_OUT(10)=1, Skip
  Move to P30, but if Output #10 is True
  Skip this step and go to the next step

WTHIF Conditions

- 10 MOV P1 WTHIF M_IN(12)=1, HLT

  Mov to P1 <u>unless</u> Input #12 is TRUE, if so Stop the robot program

- 10 MOV P1 WTHIF M1>10, M_OUT(4)=1 DLY 1

  Mov to P1, <u>but</u> if M1 is greater than 10 turn on output #4, but only for one second

- Three types of actions are allowed:
  HLT, SKIP, or Substitution (X=Y)

Signal Monitoring

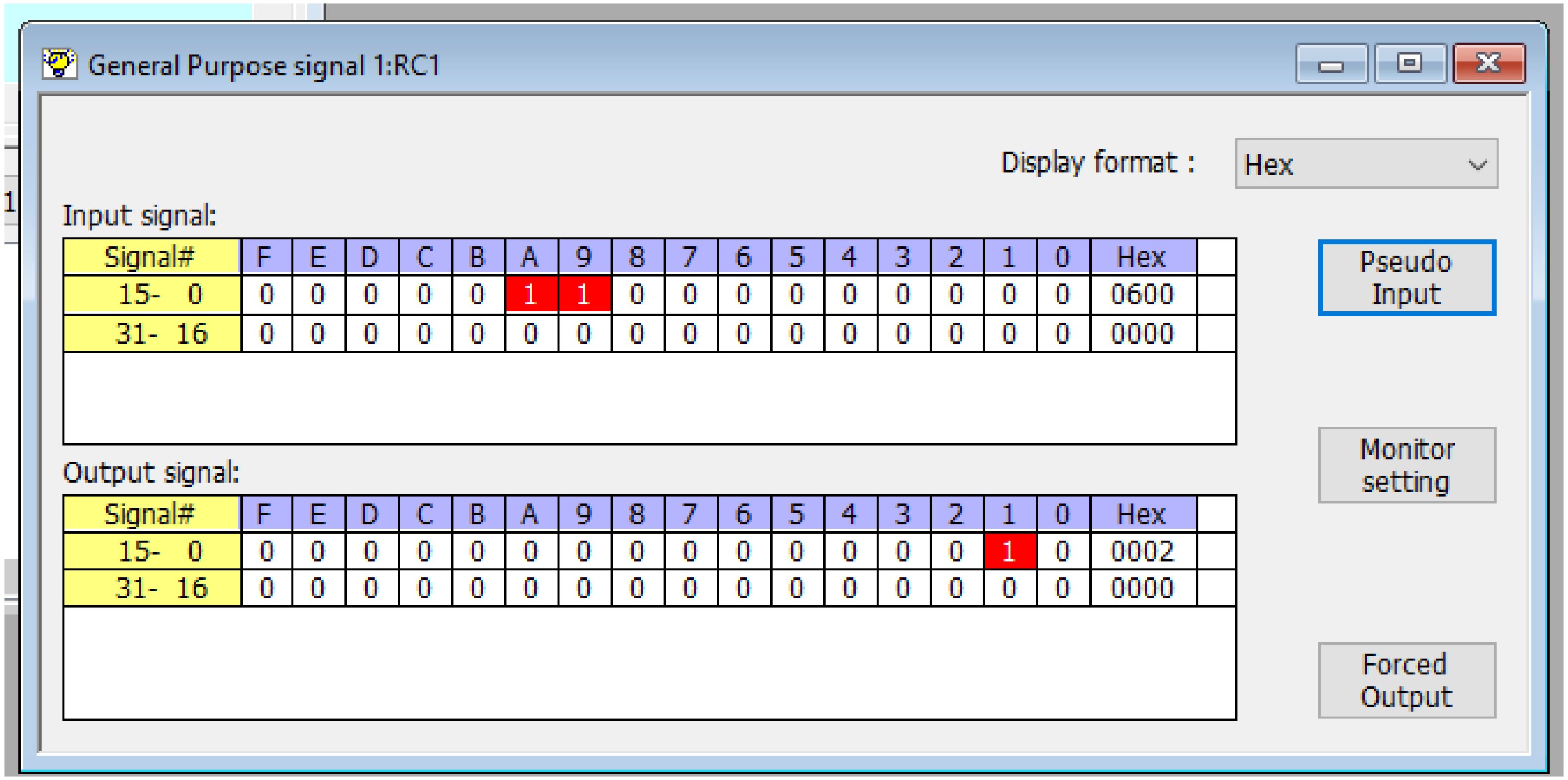
- You can check the status of input and outputs to the robot controller & FORCE I/O on or off.
- From the ToolBook2 project tree, double click the target project [Online] -> [Monitor] -> [Signal Monitor] -> [General Signals].
- The upper level displays the status of input signal, the lower level displays the status of output signals.
- <u>Pseudo-input</u> is used to manually set Input bits HIGH or LOW and enter them into the robot program. (Forcing the I/O)

Inputs and Outputs

- The Robot has a built in PLC which allows the robot to interact with external devices.
- I/O is considered an external variable (M_)
- M_IN(5) → Refers to Input #5
- M_OUT(10) → Refers to Output #10
- Outputs <u>can</u> be turned on or off in the program: M_OUT(10)=0 (Turn off Out #10)
  M_OUT(10)=1 (Turn on Out #10)
- Inputs <u>cannot</u> be operated in the program
  M_IN(5)=1 is not valid for a command

Pseudo (Forced) Input

- Pseudo input only allows Inputs to be forced. Special Purpose Inputs (Inputs 0-5) already have a dedicated function (start, stop, servo ON, Servo OFF). Use the General purpose (6-31)



General Purpose signal 1:RC1

Display format : Hex

Input signal:

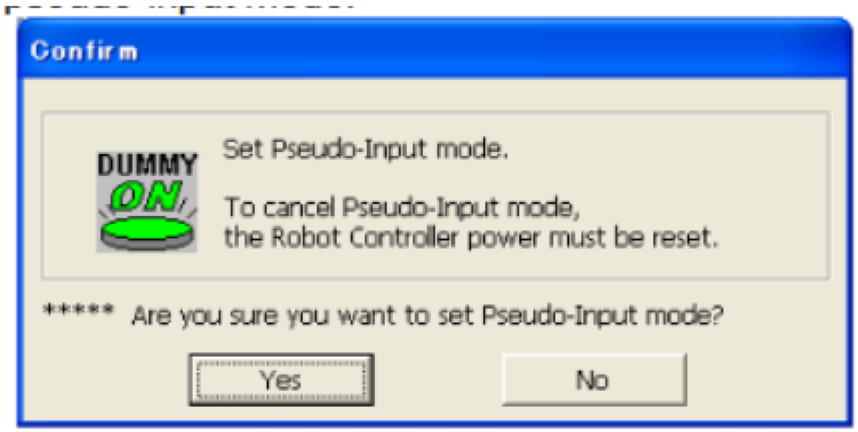
Signal#	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0	Hex
15- 0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0600
31- 16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0000

Output signal:

Signal#	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0	Hex
15- 0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0002
31- 16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0000

Notes on Pseudo Input

- When the robot is in Pseudo Input mode the actual I/O is locked out.
- When you exit Pseudo Input mode you have to cycle the power (reboot the robot)
- The robot will alarm when you come out of Pseudo input mode.


Confirm

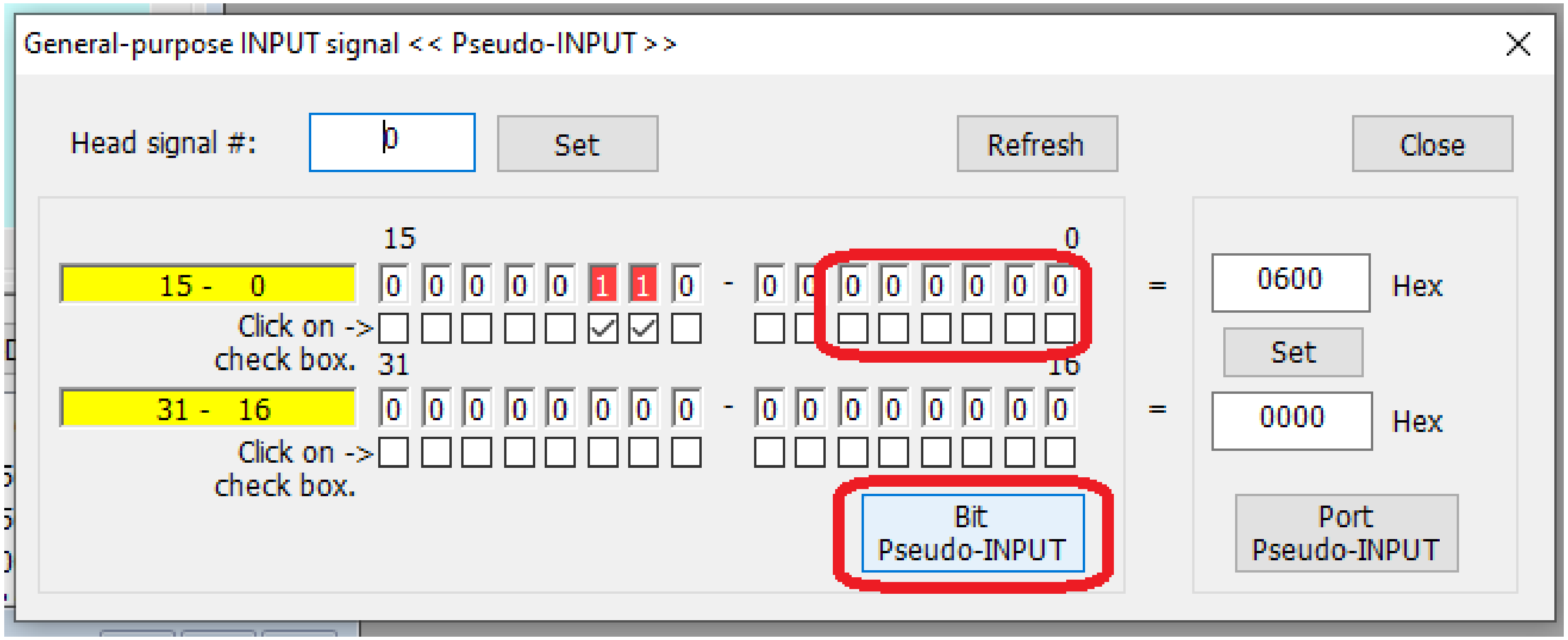
Set Pseudo-Input mode.

To cancel Pseudo-Input mode,
the Robot Controller power must be reset.

***** Are you sure you want to set Pseudo-Input mode?

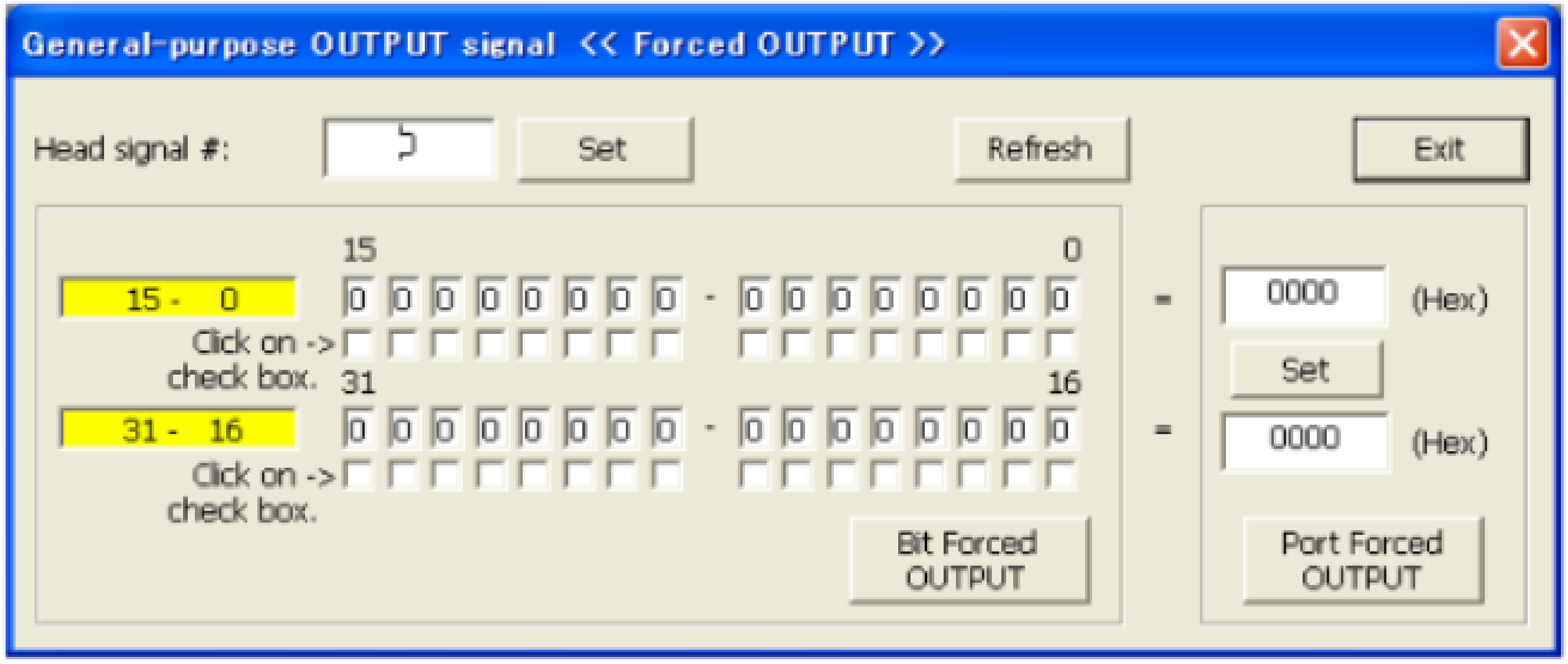
Notes on Pseudo Input

- When you start to run the program first enter a set of inputs to activate the mode.
- Even if it is just all zeros for inputs, then the program will accepts changes to inputs in run time.



Forced Output

- Outputs can be forced on or off. Outputs 0 – 3 are dedicated special purpose outputs.



General-purpose OUTPUT signal << Forced OUTPUT >>

Head signal #: 5 | Set | Refresh | Exit

	15															0			
15 - 0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	=	0000	(Hex)
Click on -> check box.	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐		Set	
	31															16			
31 - 16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	=	0000	(Hex)
Click on -> check box.	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐			

Bit Forced OUTPUT | Port Forced OUTPUT

I/O Variables

- Inputs & outputs are treated as <u>Boolean</u> type variables in a robot program.
- They are either TRUE (=1) or FALSE (=0)
- 10 WAIT M_IN(1)=1
  Wait (suspend running the program) until Input #1 goes TRUE
- 20 IF M_IN(5)=0 Then GOTO *Stack
  If Input #5 is FALSE then Jump to Label *Stack
- 30 MOV P10 WITH M_OUT(17)=1
  Move to P10 and Turn on output #17

Palletizing & Depalletizing Arrays

- Basically this is a sequential loading & unloading operation where the robot calculates the next position in the array for you.
- You set up the array just as you would a spread sheet with X & Y axis values (columns & Rows)
- The only positions you must define are the corners of the pallet (at least 3 corners, or all 4)
- Pallets are two dimensional, but you can stack one pallet on top of another.
- You can only have 8 pallets defined at any one time. (You can redefine a pallet and use it again)

Define Pallet (DEF PLT)

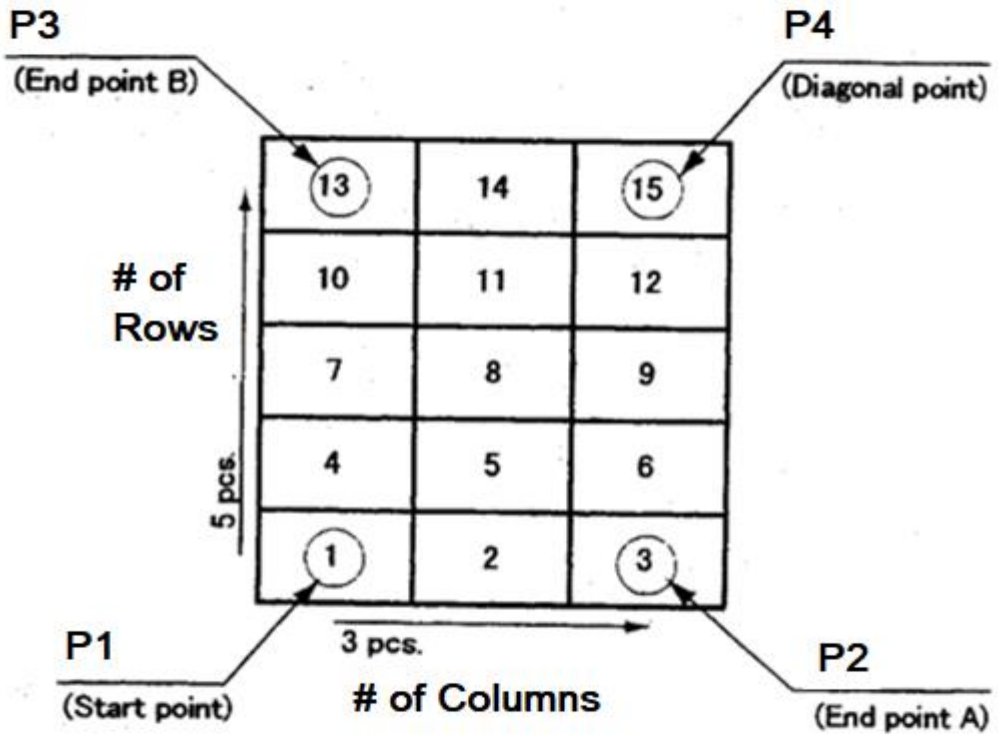
- Before you can load or unload a pallet you have to define the pallet by name, position, columns, rows, and what pattern to follow in the cells
- 50 DEF PLT 1,P1,P2,P3,P4,3,5,1 Where

  P1= Start point

  P2= End point A

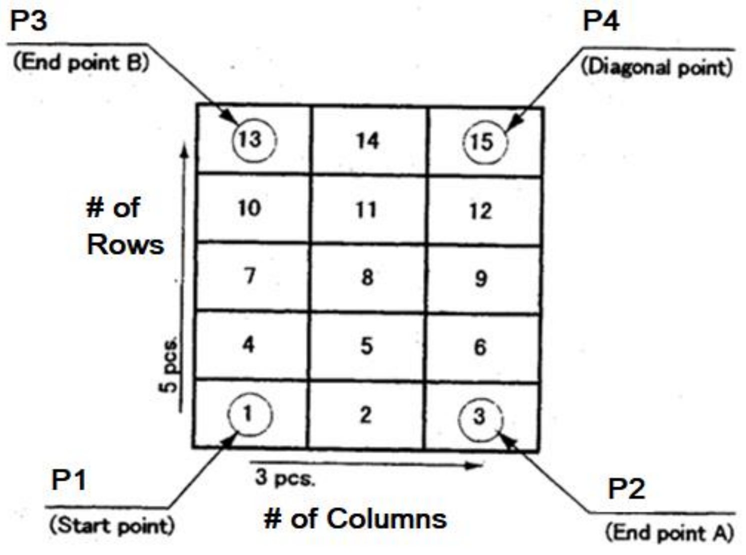
  P3= End point B

  P4= Diagonal point.



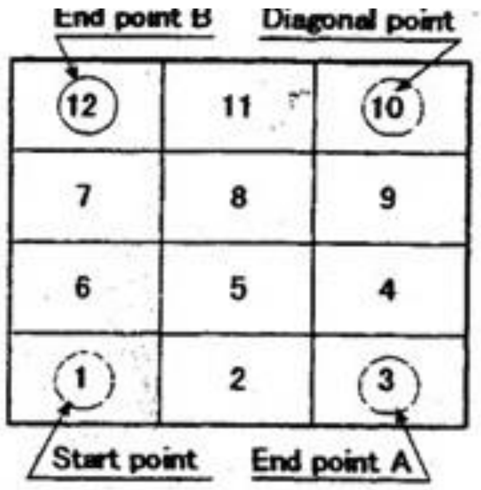
DEF PLT

- EXAMPLE: 50 DEF PLT 1,P2,P3,P4,P5,3,5,1
- DEF PLT 1, -- the name of this pallet is “1”
- The corners are at P2, P3, P4, & P5 are defined by the Programmer and stored in the program.
- 3 & 5 define the number of Cells in the array
  First : # of cells from the starting point to End Point A (Columns)
  Second : # of cells from the Starting Point to End Point B. (Rows)

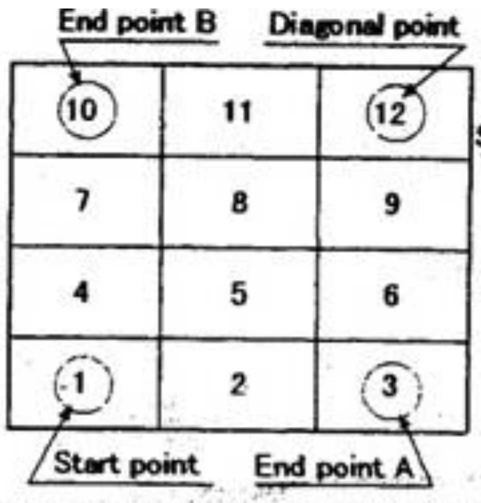


DEF PLT

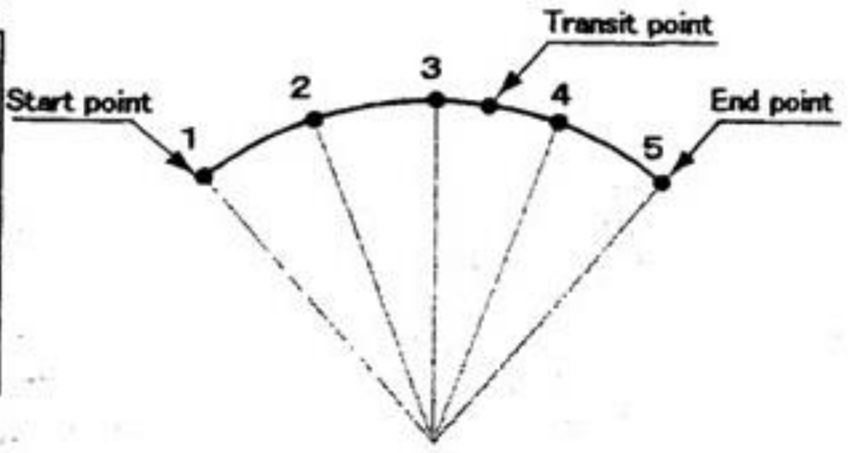
- There are three available patterns of cell movement available:
- (1) Zigzag, (2) same Direction, & (3) Arc pallet
- These determine the order of pick & place



Assignment direction = 1 (zigzag)



Assignment direction = 2 (same direction)



Assignment direction = 3 (arc pallet)

Pallet Command (PLT)

- To use the pallet you create a **position variable** (**Pointer**) then have the robot controller assign the position coordinates to that position.
- EXAMPLE: 40 P10=PLT 1, 1
  Position P10 is located in Pallet #1, and the position number is position number 1.
- EXAMPLE: 70 P20=PLT 2, M1
  Position P20 is located in Pallet #2, the position number is the same as the current value of M1.
- Increment M1 & you advance the position number

Assign a position in a Pallet

```
10 M1=0
20 DEF PLT 7, P10,P11,P12,P13,5,4,1
30 M1=M1+1
40 P1=PLT 7,M1
50 MOV P1,-50
60 MVS P1
```

This creates a Pallet #7 with 5 columns & 4 rows. Position #1 is the cell inside Pallet #7 equal to the current value of M1. So Line 40 assigns P1 as being Cell #1 in Pallet #7 (Column1, Row 1). The Robot then calculates the coordinates of this position and moves there.

Notes about Pallets

- Pallets are two-dimensional but you can choose which two dimensions. 32,767 Cells (max) each
- You can turn the pallet to whatever orientation you desire (P1 doesn’t have to be the lower left corner of the pallet, this allows you to control the order of pick and place as you wish)
- Remember in setting your pallet to allow room for the robot grippers to reach into the pallet.
- Remember to have the grippers move into the pallet from above (use the approach command MOV P10, -50, then MVS straight down to the block, then up again to clear the pallet).

Problems with Pallets

- Nearly all problems come from the DEF PLT command not setup correctly, or incorrect usage of the PLT command.
- If the robot errors when making a move with the PLT command you are often telling it to go to a position that doesn’t exist (such as position #5 in a 2x2 pallet).
- Actually set the blocks in the pattern to be sure you have allowed enough space for all the cells & for the robot grippers.

Timer Operation

- You can use up to 8 timers in a program
- Timers start at the beginning of the run of the program and increment in mS (1 sec = 1000)
- M_Timer(1) = 0 ‘resets a timer to zero
- M1 = M_Timer(2) ‘ save value of a timer
- If M_Timer(3)<= 10000 then *Run_Loop
  ‘If Timer 3 accumulator less than/equal 10 seconds then jump to label Run_Loop
- Robot timers do not have control bits
- You can set up timers in the ToolBox monitor to see current values in the program

Multitasking

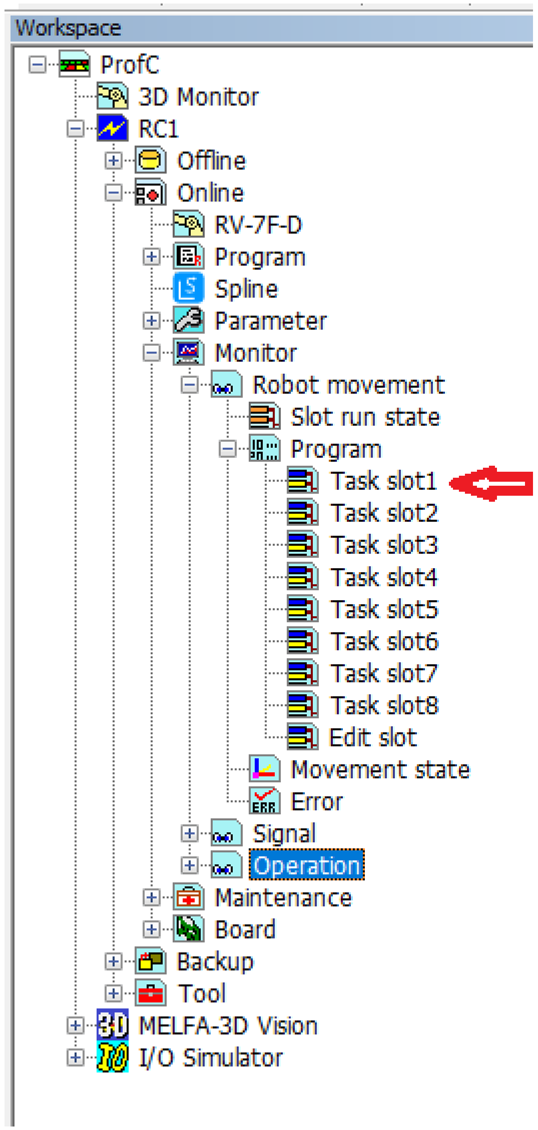
Multitasking involves the ability of the robot controller to run several programs in parallel.

This shortens response time and allows for control of several peripheral devices to interface with the robot controller (such as PLCs, or other controllers)

Multitasking requires segmenting controller memory into partitions called “Slots”.   The Mitsubishi robots can have up to 8 slots created.

The main user program is always loaded into slot #1

Multitasking Commands



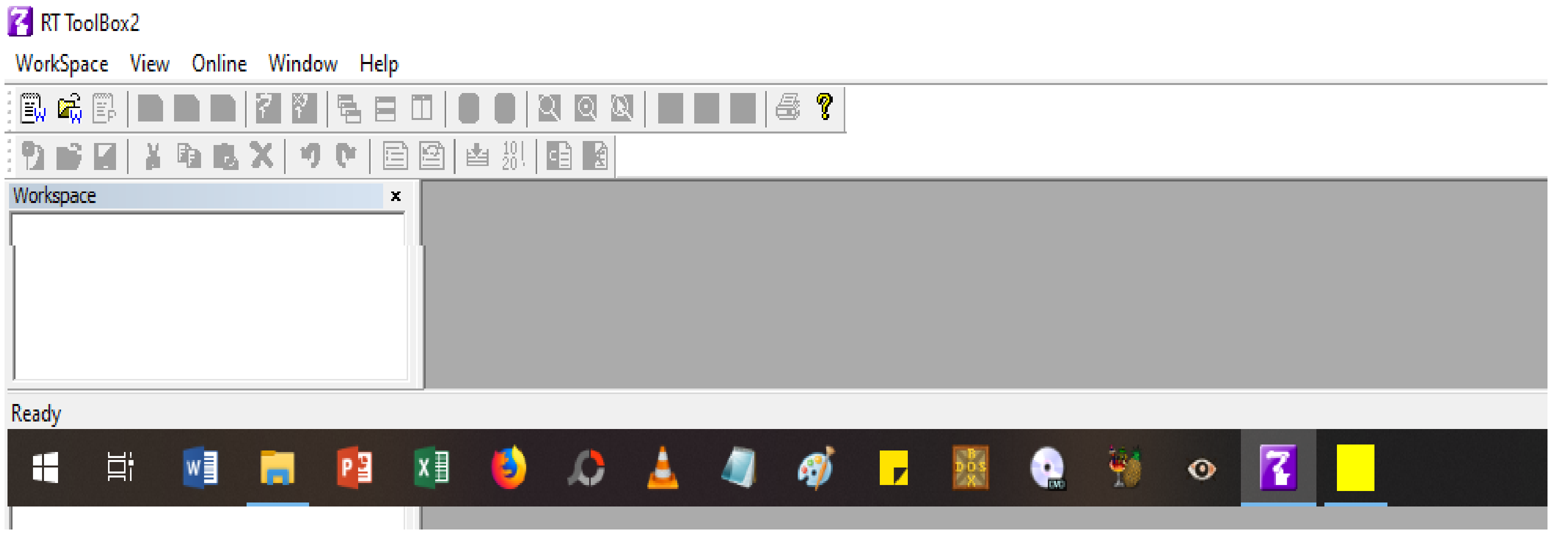
XLOAD: Loads a program from memory into a specified Slot
XLOAD 2, "PRG2"
load file PRG2 into Slot #2

XRUN: Executes (runs) the designated program in the slot
XRUN 2, "PRG2"
Run file PRG2 in Slot #2 concurrently with the user program in Slot #1

XRST (Reset) & XSTP (Stop)

Using RT ToolBox 2 Software

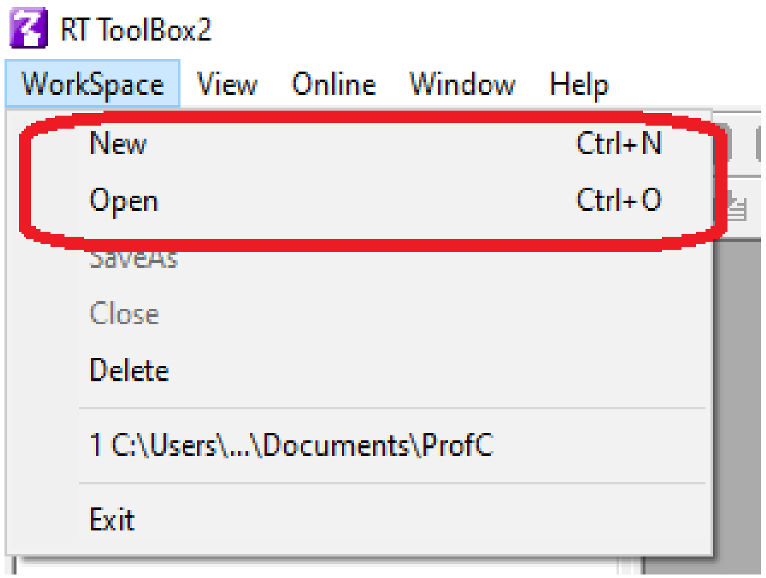
- Open RT ToolBox2 in Windows

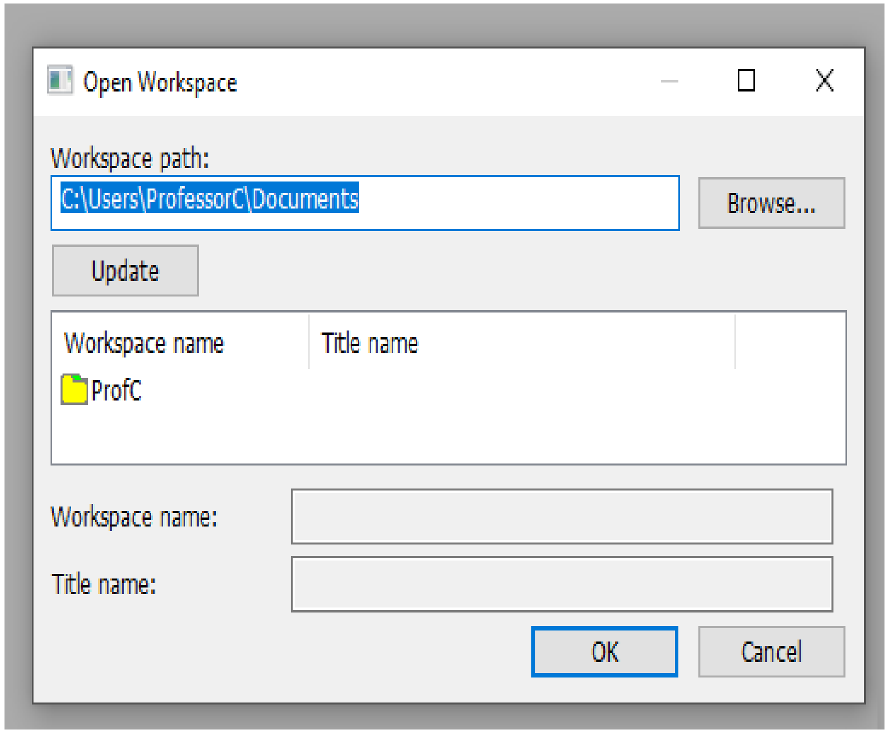


Note that the YELLOW block at the bottom indicates the Robot is OFFLINE

Using RT ToolBox 2 Software

- From the [WORKSPACE] tab → [OPEN] **<u>OR</u>** [NEW]   If Opening a preconfigured existing workspace select, select from menu





Workspace path: C:\Users\ProfessorC\Documents

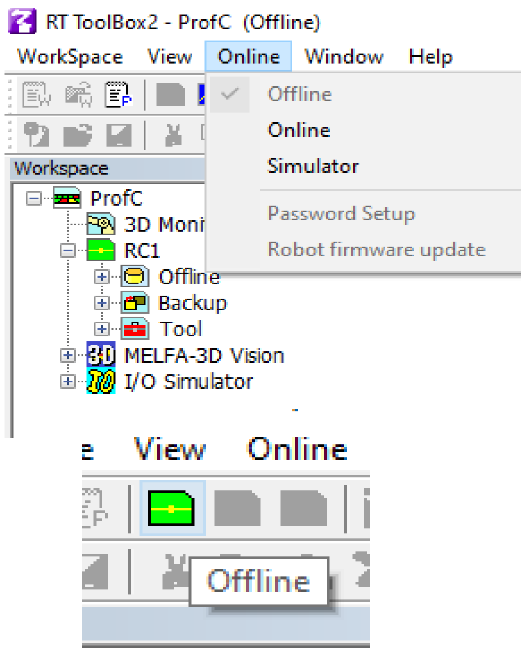
Workspace name	Title name
ProfC	

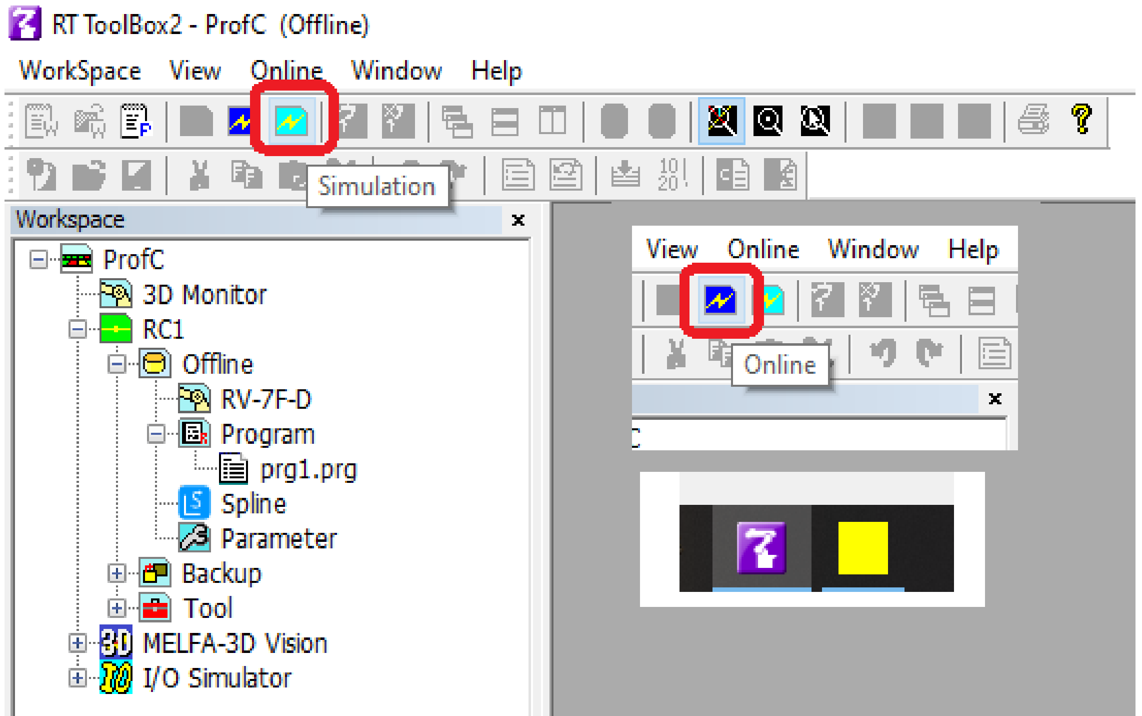
Workspace name:

Title name:

Using RT ToolBox 2 Software

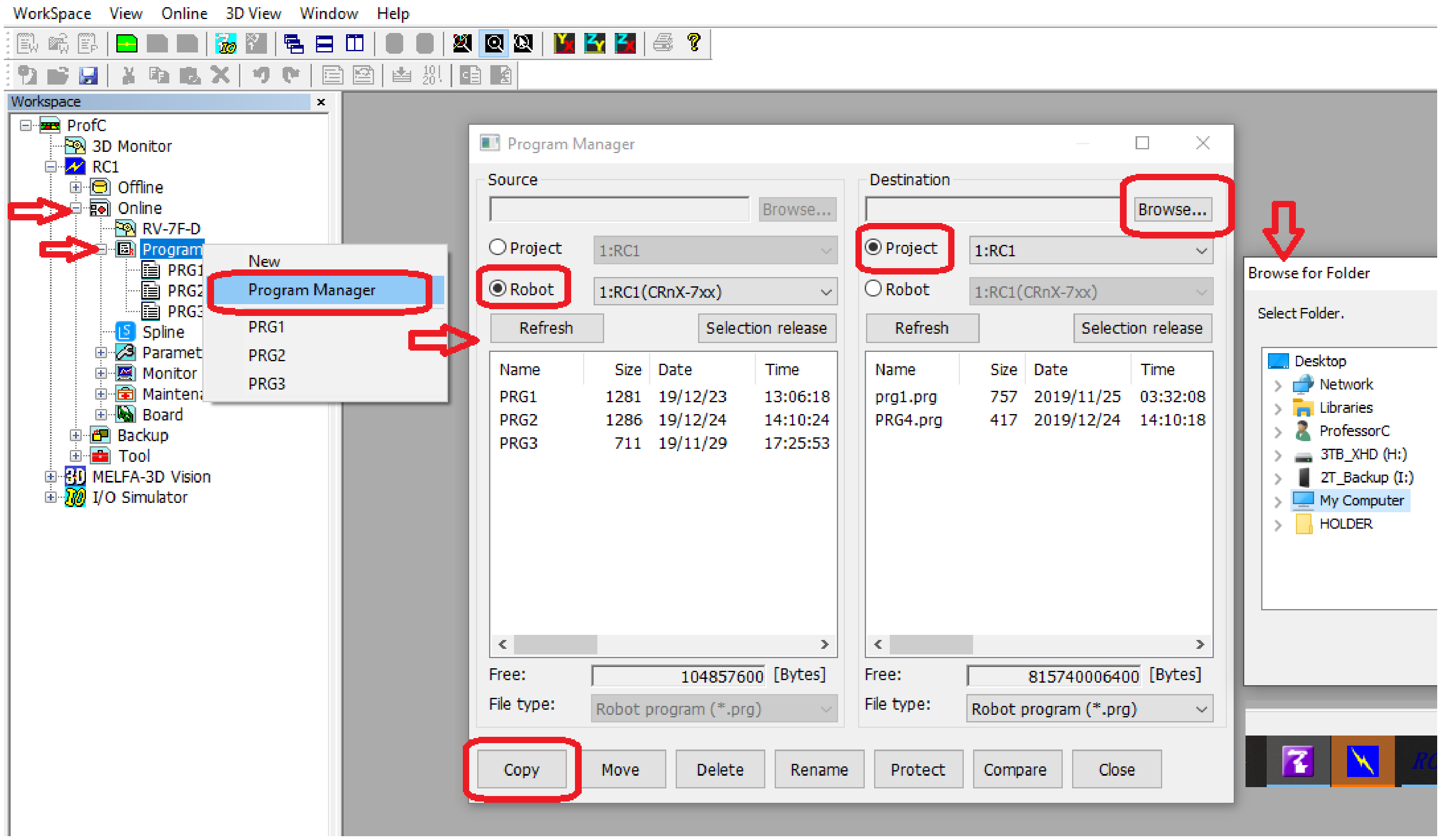
- When the selected or created workspace opens the project tree will be visible. This is the OFFLINE screenshot shown (Yellow block). To either run the program on an actual robot [ONLINE] or to run a simulation within the PC [SIMULATION]. You can either select tab [ONLINE] and from dropdown and choose from there OR you can use the three icons below the [ONLINE] tab.
- A. Light Blue + Lightning Bolt = Run program as simulation in the PC
  B. Dark Blue + Lightning Bolt = Run Online connected to a real robot.
  The Yellow block will turn Green while attempting to communicate, then either go to the Dark Blue or show a Red block with a "Bomb burst" to show failure.
  C. Green + Yellow Flatline = Return to [OFFLINE] mode (only shows if 'online')





Using RT ToolBox 2 Software

- Save program to a USB use the program manager



Program Manager

Source: ○ Project 1:RC1 — [● Robot] 1:RC1(CRnX-7xx)

Refresh | Selection release

Name	Size	Date	Time
PRG1	1281	19/12/23	13:06:18
PRG2	1286	19/12/24	14:10:24
PRG3	711	19/11/29	17:25:53

Free: 104857600 [Bytes]
File type: Robot program (*.prg)

Destination: Browse... — [● Project] 1:RC1 — ○ Robot 1:RC1(CRnX-7xx)

Refresh | Selection release

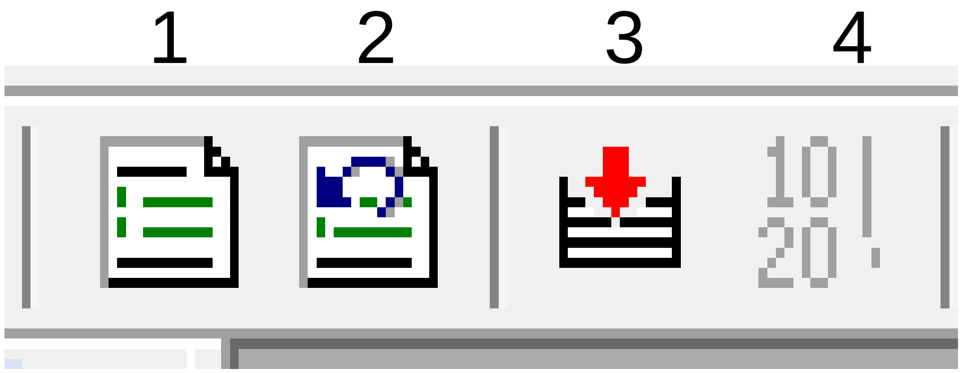
Name	Size	Date	Time
prg1.prg	757	2019/11/25	03:32:08
PRG4.prg	417	2019/12/24	14:10:18

Free: 815740006400 [Bytes]
File type: Robot program (*.prg)

Copy | Move | Delete | Rename | Protect | Compare | Close

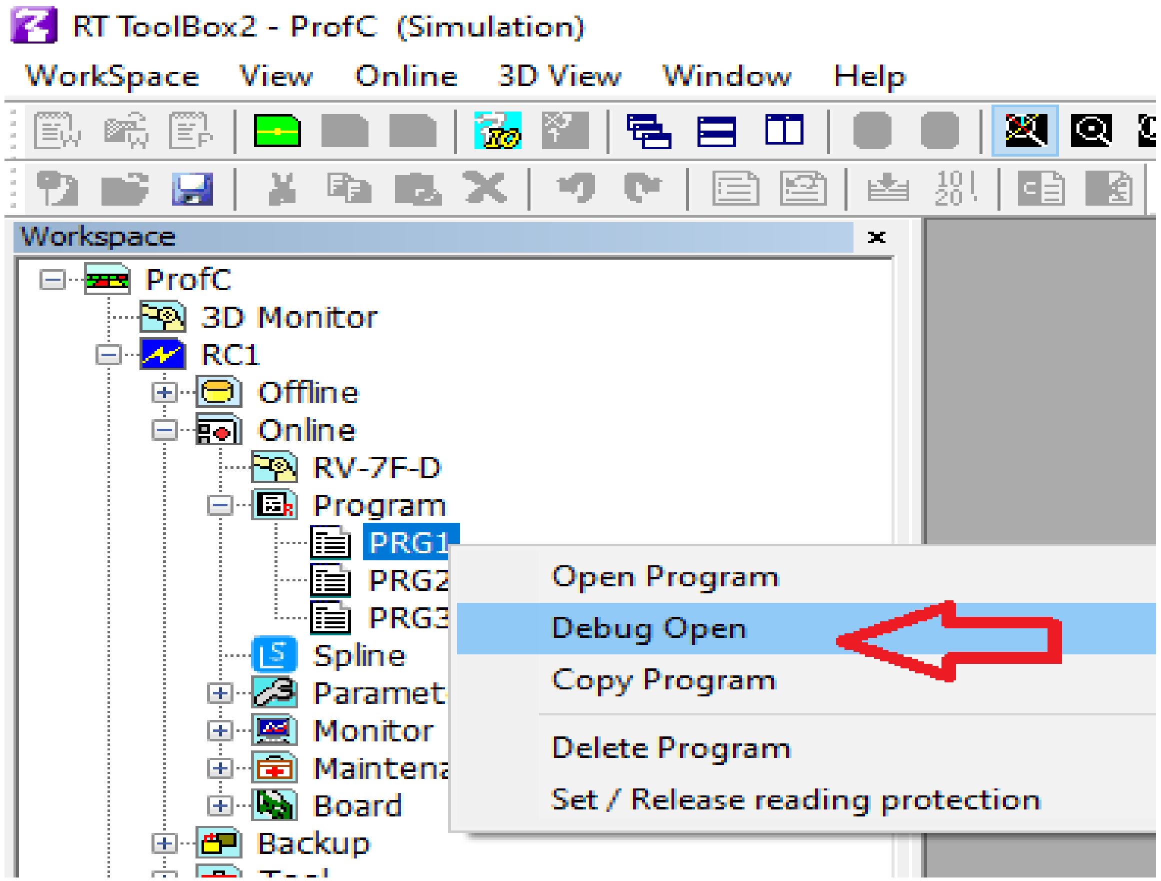
Program Editing Tools

- Program editing tools from the Toolbar. Located on the second row down from main tabs
  1. Turn step/Line into a Comment (puts a ‘)
  2. UN-comment a step/line (removes the ‘)
  3. Syntax check to look for program code errors
  4. Sorting tool

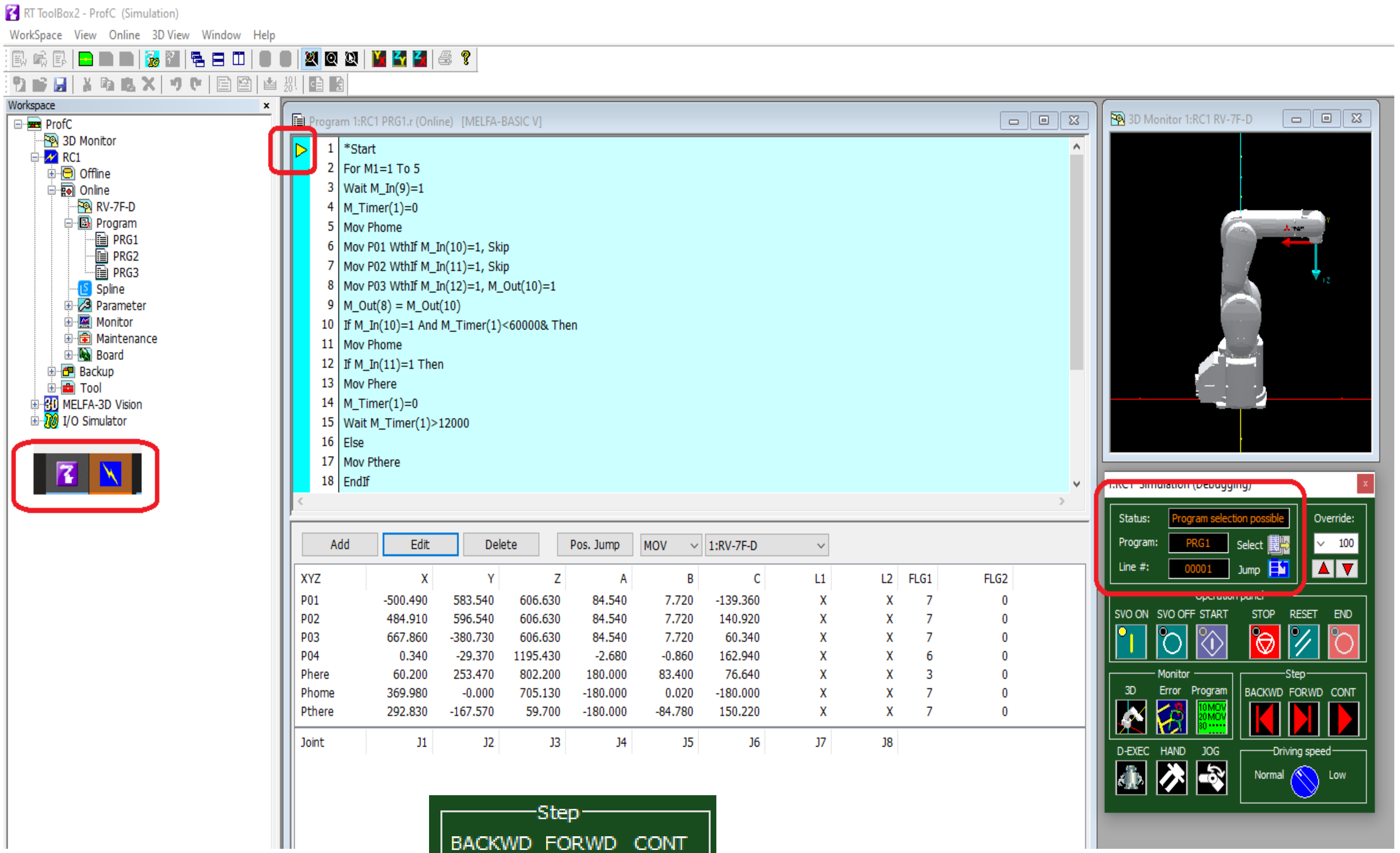


Debugging a program

- From your program right-click to select “Debug”
- When in Debug the screen back ground is blue
- You can step through the program



Debugging a Program

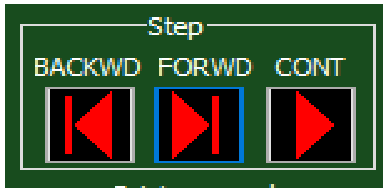


```
*Start
For M1=1 To 5
Wait M_In(9)=1
M_Timer(1)=0
Mov Phome
Mov P01 WthIf M_In(10)=1, Skip
Mov P02 WthIf M_In(11)=1, Skip
Mov P03 WthIf M_In(12)=1, M_Out(10)=1
M_Out(8) = M_Out(10)
If M_In(10)=1 And M_Timer(1)<60000& Then
Mov Phome
If M_In(11)=1 Then
Mov Phere
M_Timer(1)=0
Wait M_Timer(1)>12000
Else
Mov Pthere
EndIf
```

XYZ	X	Y	Z	A	B	C	L1	L2	FLG1	FLG2
P01	-500.490	583.540	606.630	84.540	7.720	-139.360	X	X	7	0
P02	484.910	596.540	606.630	84.540	7.720	140.920	X	X	7	0
P03	667.860	-380.730	606.630	84.540	7.720	60.340	X	X	7	0
P04	0.340	-29.370	1195.430	-2.680	-0.860	162.940	X	X	6	0
Phere	60.200	253.470	802.200	180.000	83.400	76.640	X	X	3	0
Phome	369.980	-0.000	705.130	-180.000	0.020	-180.000	X	X	7	0
Pthere	292.830	-167.570	59.700	-180.000	-84.780	150.220	X	X	7	0

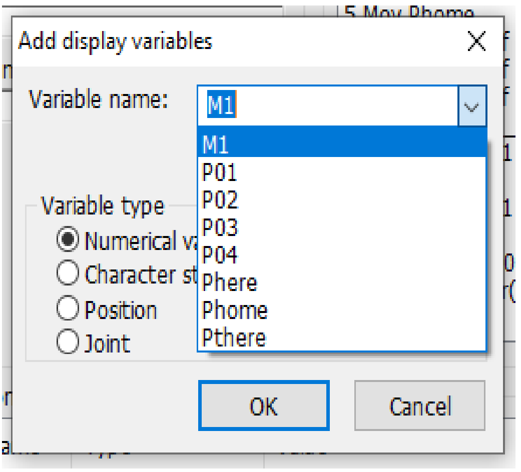
Joint	J1	J2	J3	J4	J5	J6	J7	J8

Status: Program selection possible
Program: PRG1 Select
Line #: 00001 Jump
Override: 100



Setup a Monitor Screen

Allows you to view the value of a variable as you step through the program

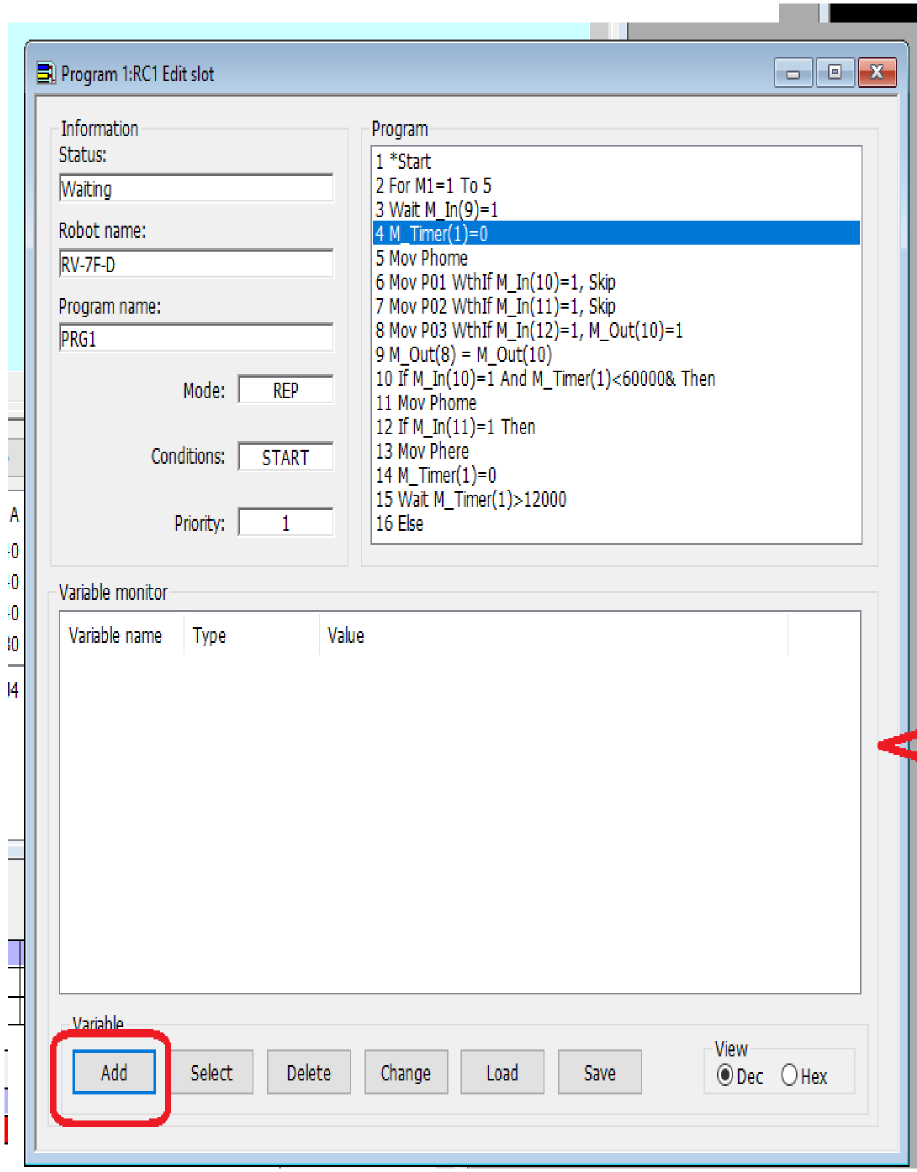


Add display variables

Variable name: M1

- M1
- P01
- P02
- P03
- P04
- Phere
- Phome
- Pthere

Variable type
- [✓ Numerical va]
- Character st
- Position
- Joint



Program 1:RC1 Edit slot

Information

Status: Waiting

Robot name: RV-7F-D

Program name: PRG1

Mode: REP

Conditions: START

Priority: 1

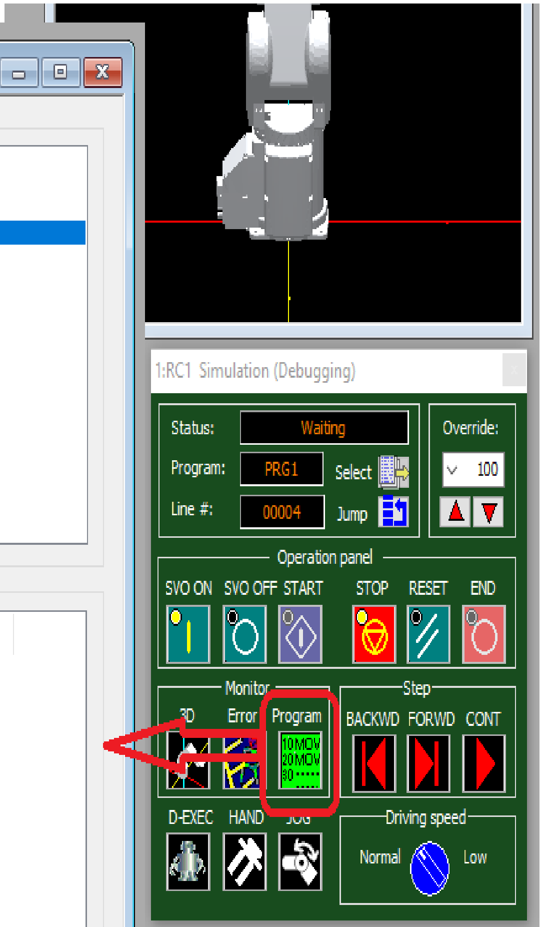
Program

```
1 *Start
2 For M1=1 To 5
3 Wait M_In(9)=1
4 M_Timer(1)=0
5 Mov Phome
6 Mov P01 WthIf M_In(10)=1, Skip
7 Mov P02 WthIf M_In(11)=1, Skip
8 Mov P03 WthIf M_In(12)=1, M_Out(10)=1
9 M_Out(8) = M_Out(10)
10 If M_In(10)=1 And M_Timer(1)<60000& Then
11 Mov Phome
12 If M_In(11)=1 Then
13 Mov Phere
14 M_Timer(1)=0
15 Wait M_Timer(1)>12000
16 Else
```

Variable monitor

Variable name	Type	Value

View: [✓ Dec] Hex



1:RC1 Simulation (Debugging)

Status: Waiting

Program: PRG1

Line #: 00004

Override: 100

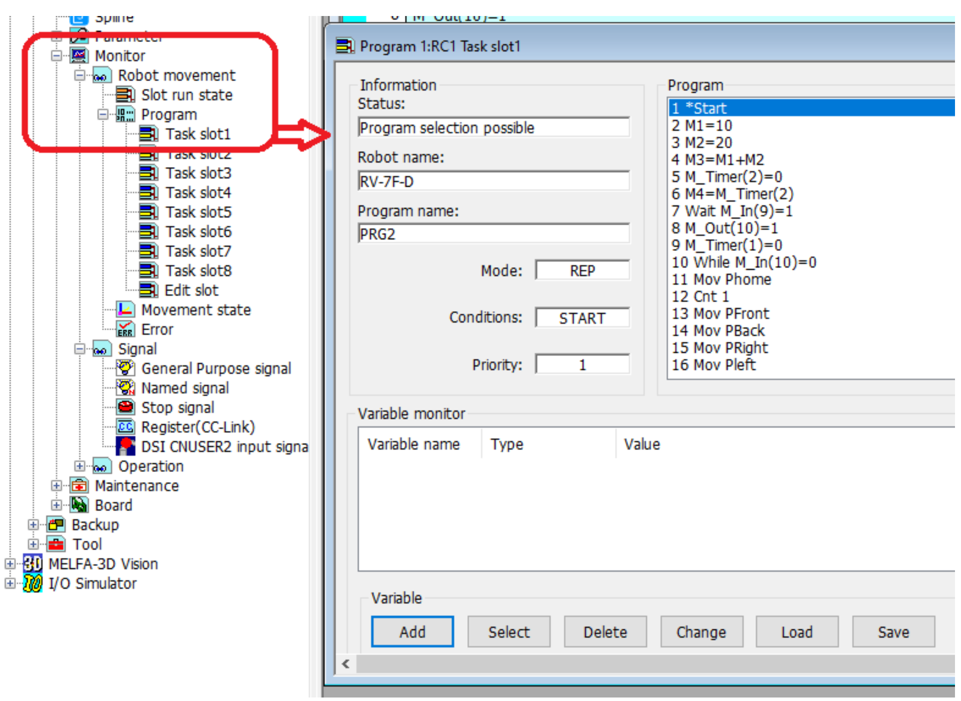
Monitor Variables (2 ways)

(1) Select Monitor → Program → Task Slot 1

(2) Use the Monitor → Program Controller button

Use [ADD] to select the variable to monitor

or a variable can be manually written in.



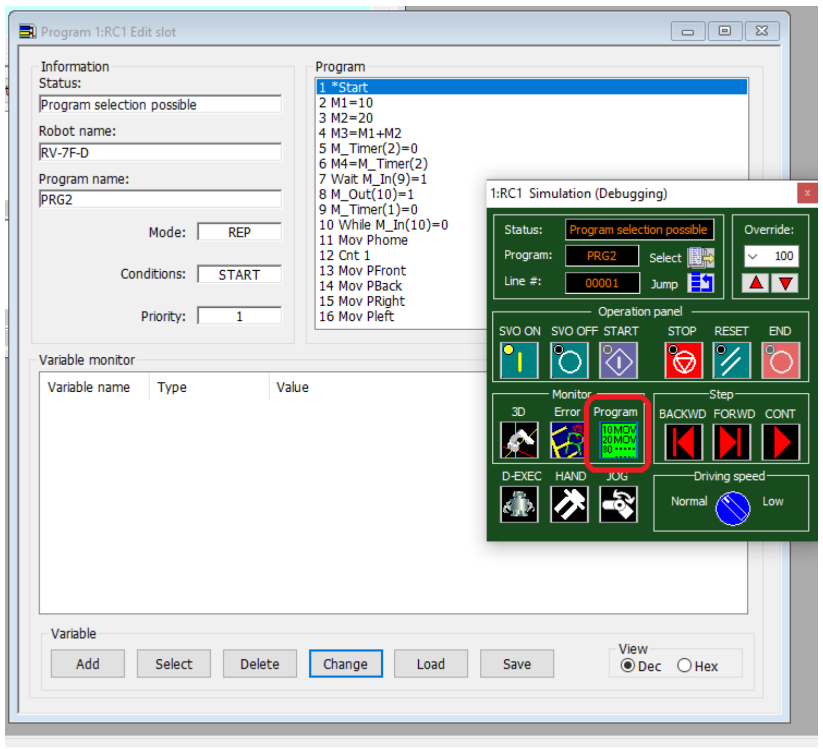
Program 1:RC1 Task slot1

Information
Status: Program selection possible
Robot name: RV-7F-D
Program name: PRG2
Mode: REP
Conditions: START
Priority: 1

Program
```
1 *Start
2 M1=10
3 M2=20
4 M3=M1+M2
5 M_Timer(2)=0
6 M4=M_Timer(2)
7 Wait M_In(9)=1
8 M_Out(10)=1
9 M_Timer(1)=0
10 While M_In(10)=0
11 Mov Phome
12 Cnt 1
13 Mov PFront
14 Mov PBack
15 Mov PRight
16 Mov Pleft
```

Variable monitor

Variable name	Type	Value



Program 1:RC1 Edit slot

Information
Status: Program selection possible
Robot name: RV-7F-D
Program name: PRG2
Mode: REP
Conditions: START
Priority: 1

Program
```
1 *Start
2 M1=10
3 M2=20
4 M3=M1+M2
5 M_Timer(2)=0
6 M4=M_Timer(2)
7 Wait M_In(9)=1
8 M_Out(10)=1
9 M_Timer(1)=0
10 While M_In(10)=0
11 Mov Phome
12 Cnt 1
13 Mov PFront
14 Mov PBack
15 Mov PRight
16 Mov Pleft
```

Variable monitor

Variable name	Type	Value

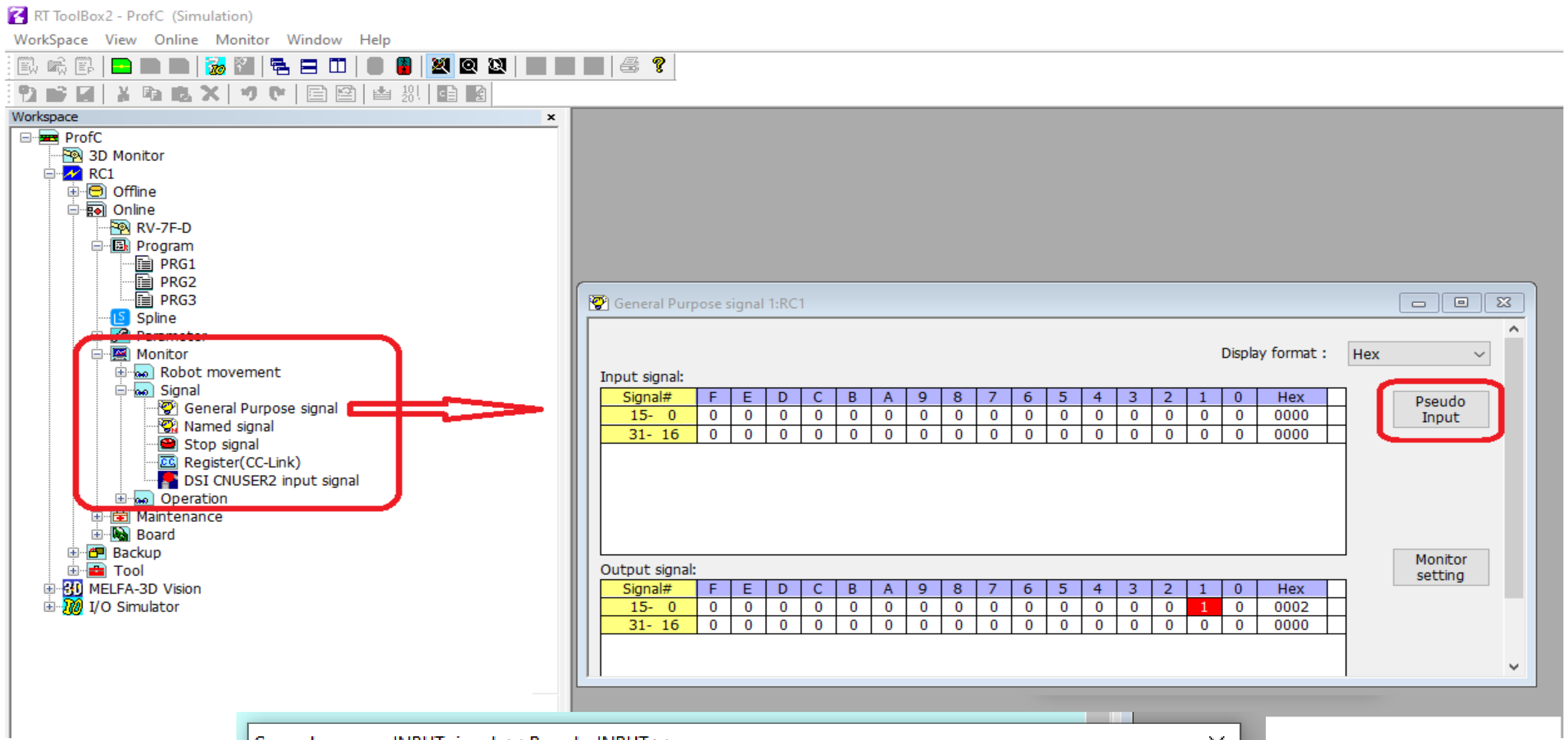
1:RC1 Simulation (Debugging)

Status: Program selection possible
Program: PRG2
Line #: 00001
Override: 100

I/O Simulator

- The robot Controller (RC1) has a built in PLC with 32 inputs and 32 outputs which can interact with the robot through external connections, or be simulated through the software.
- Inputs 0-5 are dedicated in the controller
- Outputs 0-3 are dedicated in the controller
- The remainder are ‘general’ purpose you can use in your program as needed to simulate external actions of switches or sensors.
- Do NOT use the dedicated IO in your programs, you can cause errors to occur or crash the program.

Using I/O Simulator: Pseudo- Input



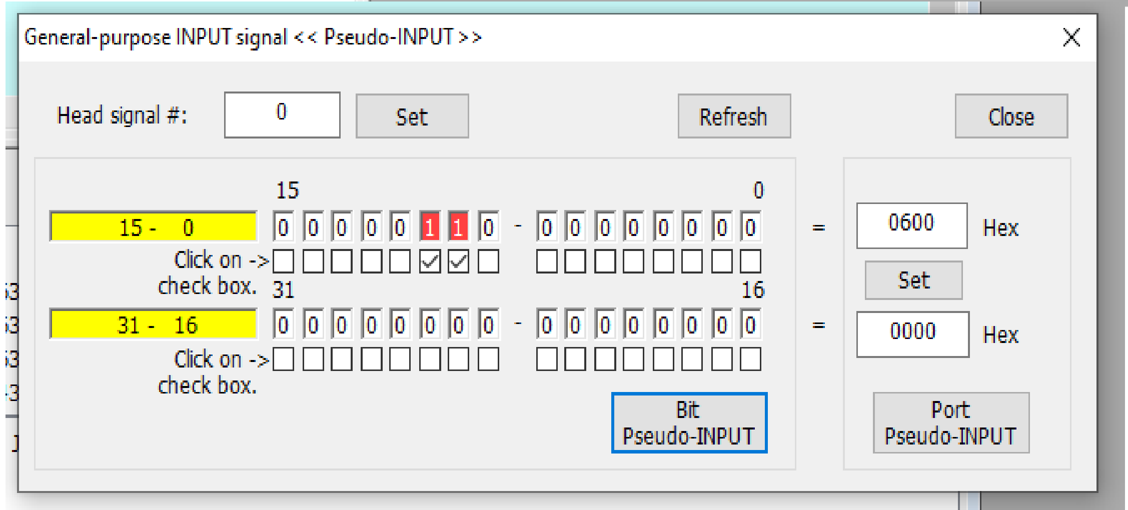
Display format : Hex

Input signal:

Signal#	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0	Hex
15- 0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0000
31- 16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0000

Output signal:

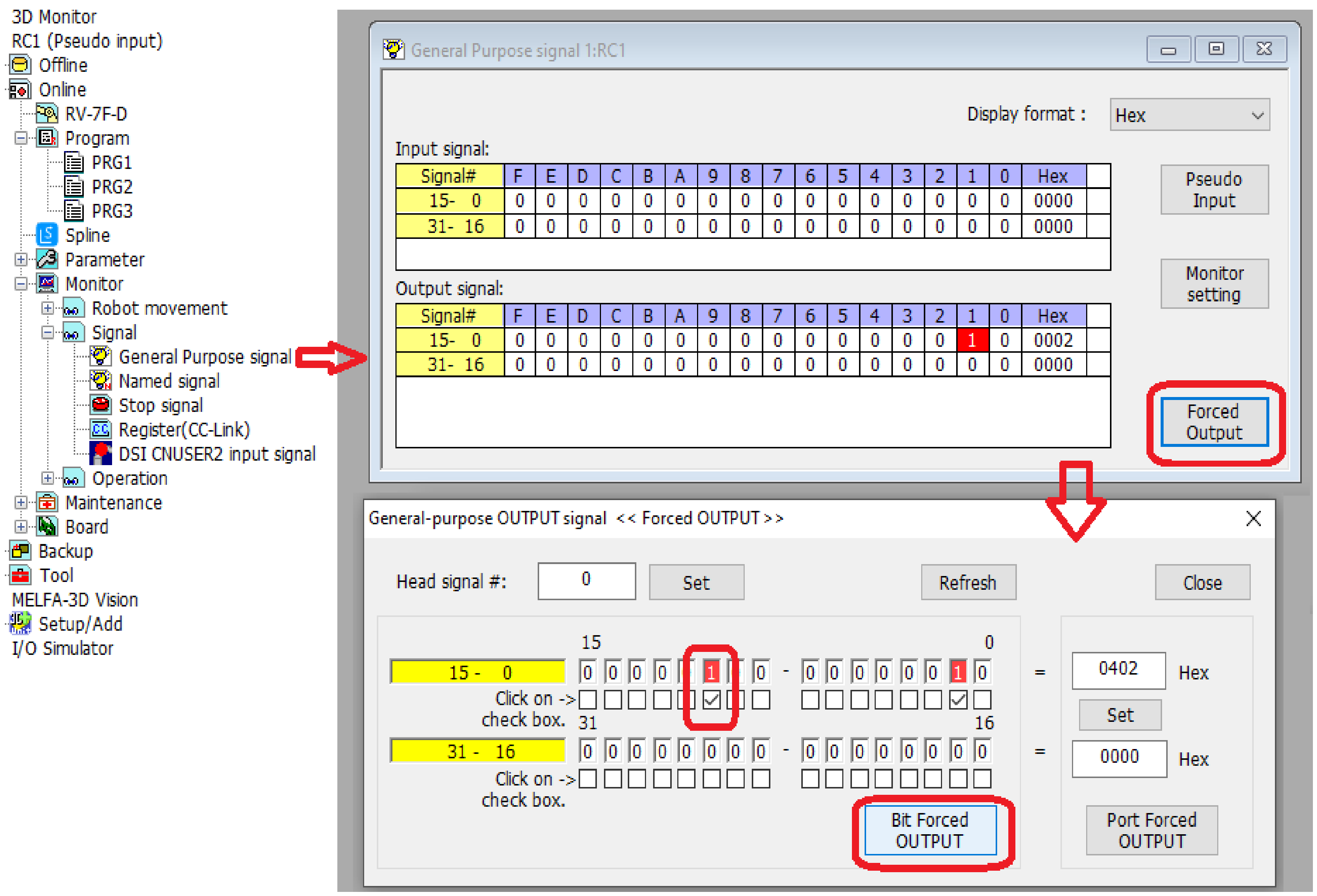
Signal#	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0	Hex
15- 0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0002
31- 16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0000



Head signal #: 0

	15															0		
15 - 0	0	0	0	0	0	1	1	0	-	0	0	0	0	0	0	0	0	= 0600 Hex
Click on -> check box.	☐	☐	☐	☐	☐	✓	✓	☐		☐	☐	☐	☐	☐	☐	☐	☐	
	31															16		
31 - 16	0	0	0	0	0	0	0	0	-	0	0	0	0	0	0	0	0	= 0000 Hex
Click on -> check box.	☐	☐	☐	☐	☐	☐	☐	☐		☐	☐	☐	☐	☐	☐	☐	☐	

Using I/O Simulator: Forced Output



- 3D Monitor
- RC1 (Pseudo input)
  - Offline
  - Online
    - RV-7F-D
    - Program
      - PRG1
      - PRG2
      - PRG3
    - Spline
    - Parameter
    - Monitor
      - Robot movement
      - Signal
        - General Purpose signal
        - Named signal
        - Stop signal
        - Register(CC-Link)
        - DSI CNUSER2 input signal
      - Operation
    - Maintenance
    - Board
  - Backup
  - Tool
- MELFA-3D Vision
  - Setup/Add
- I/O Simulator

General Purpose signal 1:RC1

Display format : Hex

Input signal:

Signal#	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0	Hex
15- 0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0000
31- 16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0000

Output signal:

Signal#	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0	Hex
15- 0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0002
31- 16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0000

General-purpose OUTPUT signal << Forced OUTPUT >>

Head signal #: 0

15 … 0

15 - 0: 0 0 0 0 0 1 0 0 - 0 0 0 0 0 0 1 0 = 0402 Hex

Click on -> check box. ☐ ☐ ☐ ☐ ☐ ☑ ☐ ☐ - ☐ ☐ ☐ ☐ ☐ ☐ ☑ ☐

31 … 16

31 - 16: 0 0 0 0 0 0 0 0 - 0 0 0 0 0 0 0 0 = 0000 Hex

Click on -> check box. ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ - ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐

Using I/O in Robot

If M_IN(10)=1 Then *Go_Here Else *Go_There

Wait M_IN(12)=0

While M_IN(8)=0

MOV P10 WTH M_OUT(6)=1, DLY 3

Input Description	Address		Output Description	Address
Infeed tray in position	8		Infeed tray error	8
Output tray in position	9		Output tray error	9
Cycle Start	10		Ready to cycle start (blink)	10
			Robot in Home position	11
			Part being processed (A → F)	12
			Cycle done (5 seconds)	13

Don’t use IN 0→5 & OUT 0→3

We are Done!

- Remember that there are other commands that can be used and these are located in the manuals posted on BB in the Reference Materials section.
- Next Tuesday we will try some basic motion programming.   The Lab sheet will be on BB.