Lampiran 1. Keluarga Mikrokontroler Basic Stamp

Released Products	Rev.Dx / BS1-IC	BS2-IC	BS2e-IC	BS2sx-IC
Package	PCB w/Proto / 14-pin SIP	24-pin DIP	24-pin DIP	24-pin DIP
Package Size (L x W x H)	2.5" x 1.5" x .5" / 1.4" x .6" x .1"	1.2" x 0.6" x 0.4"	1.2" x 0.6" x 0.4"	1.2" x 0.6" x 0.4"
Environment *	0° - 70° C (32° - 158° F) **	0° - 70° C (32° - 158° F) **	0° - 70° C (32° - 158° F)	0° - 70° C (32° - 158° F)
Microcontroller	Microchip PIC16C56	Microchip PIC16x57	Parallax SX28	Parallax SX28
Processor Speed	4 MHz	20 MHz	20 MHz	50 MHz
Program Execution Speed	~2,000 instructions/sec.	~4,000 instructions/sec.	~4,000 instructions/sec.	~10,000 instructions/sec.
RAM Size	16 Bytes (2 I/O, 14 Variable)	32 Bytes (6 I/O, 26 Variable)	32 Bytes (6 I/O, 26 Variable)	32 Bytes (6 I/O, 26 Variable)
Scratch Pad RAM	N/A	N/A	64 Bytes	64 Bytes
EEPROM (Program) Size	256 Bytes, ~80 instructions	2K Bytes, ~500 instructions	8 x 2K Bytes, ~4,000 inst.	8 x 2K Bytes, ~4,000 inst.
Number of I/O pins	8	16 + 2 Dedicated Serial	16 + 2 Dedicated Serial	16 + 2 Dedicated Serial
Voltage Requirements	5 - 15 vdc	5 - 15 vdc	5 - 12 vdc	5 - 12 vdc
Current Draw @ 5V	1 mA Run / 25 µA Sleep	3 mA Run / 50 µA Sleep	25 mA Run / 200 µA Sleep	60 mA Run / 500 µA Sleep
Source / Sink Current per I/O	20 mA / 25 mA	20 mA / 25 mA	30 mA / 30 mA	30 mA / 30 mA
Source / Sink Current per unit	40 mA / 50 mA	40 mA / 50 mA per 8 I/O pins	60 mA / 60 mA per 8 I/O pins	60 mA / 60 mA per 8 I/O pins
PBASIC Commands***	32	42	45	45
PC Programming Interface	Serial (w/BS1 Serial Adapter)	Serial (9600 baud)	Serial (9600 baud)	Serial (9600 baud)
Windows Text Editor	Stampw.exe (v2.1 and up)	Stampw.exe (v1.04 and up)	Stampw.exe (v1.096 and up)	Stampw.exe (v1.091 and up)

Released Products	BS2p24-IC	BS2p40-IC	BS2pe-IC	BS2px-IC
Package	24-pin DIP	40-pin DIP	24-pin DIP	24-pin DIP
Package Size (L x W x H)	1.2" x 0.6" x 0.4"	2.1" x 0.6" x 0.4"	1.2" x 0.6" x 0.4"	1.2" x 0.6" x 0.4"
Environment *	0° - 70° C (32° - 158° F)	0° - 70° C (32° - 158° F)	0° - 70° C (32° - 158° F)	0° - 70° C (32° - 158° F)
Microcontroller	Parallax SX48	Parallax SX48	Parallax SX48	Parallax SX48
Processor Speed	20 MHz Turbo	20 MHz Turbo	8 MHz Turbo	32 MHz Turbo
Program Execution Speed	~12,000 instructions/sec.	~12,000 instructions/sec.	~6000/sec.	~19,000 instructions/sec.
RAM Size	38 Bytes (12 I/O, 26 Variable)	38 Bytes (12 I/O, 26 Variable)	38 Bytes (12 I/O, 26 Variable)	38 Bytes (12 I/O, 26 Variable)
Scratch Pad RAM	128 Bytes	128 Bytes	128 Bytes	128 Bytes
EEPROM (Program) Size	8 x 2K Bytes, ~4,000 inst.	8 x 2K Bytes, ~4,000 inst.	16 x 2K Bytes (16 K for source)	8 x 2K Bytes, ~4,000 inst.
Number of I/O pins	16 + 2 Dedicated Serial	32 + 2 Dedicated Serial	16 + 2 Dedicated Serial	16 + 2 Dedicated Serial
Voltage Requirements	5 - 12 vdc	5 - 12 vdc	5 - 12 vdc	5 - 12 vdc
Current Draw @ 5V	40 mA Run / 350 µA Sleep	40 mA Run / 350 µA Sleep	15 mA Run / 36 µA Sleep	55 mA Run / 450 µA Sleep
Source / Sink Current per I/O	30 mA / 30 mA	30 mA / 30 mA	30 mA / 30 mA	30 mA / 30 mA
Source / Sink Current per unit	60 mA / 60 mA per 8 I/O pins	60 mA / 60 mA per 8 I/O pins	60 mA / 60 mA per 8 I/O pins	60 mA / 60 mA per 8 I/O pins
PBASIC Commands***	61	61	61	63
PC Programming Interface	Serial (9600 baud)	Serial (9600 baud)	Serial (9600 baud)	Serial (19200 baud)
Windows Text Editor	Stampw.exe (v1.1 and up)	Stampw.exe (v1.1 and up)	Stampw.exe (v1.33 and up)	Stampw.exe (v2.2 and up)

* 70% Non-Condensing Humidity
** Industrial Models Available, -40° - 85° C (-40° - 185° F).
*** Using PBASIC 2.5 for BS2-type models.

Sumber : Parallax, Inc., USA

Lampiran 2. Foto-Foto Robot Pencari Jalur

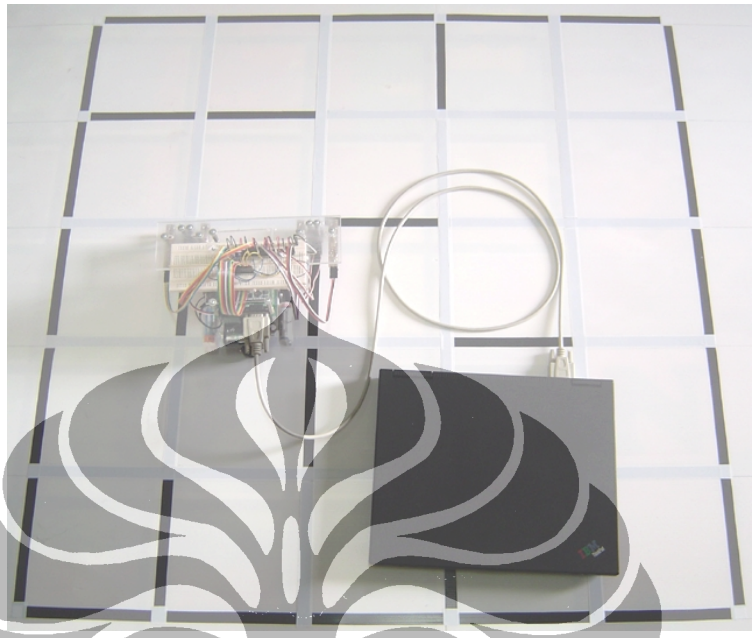


Foto 1 Development Environment:
Download Kecerdasan Buatan pada Robot Pencari Jalur Menggunakan Komunikasi Serial

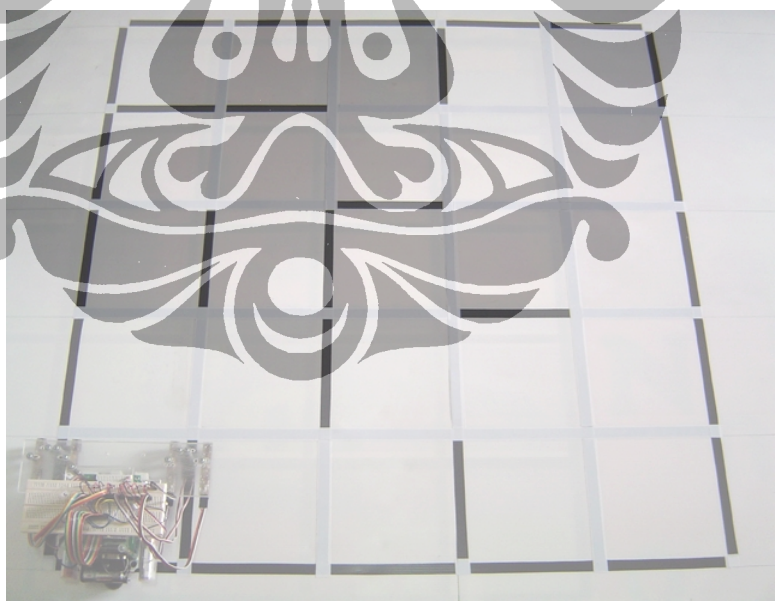


Foto 2. *Cell* Awal Robot Pencari Jalur di Lingkungan Labirin Dua Dimensi

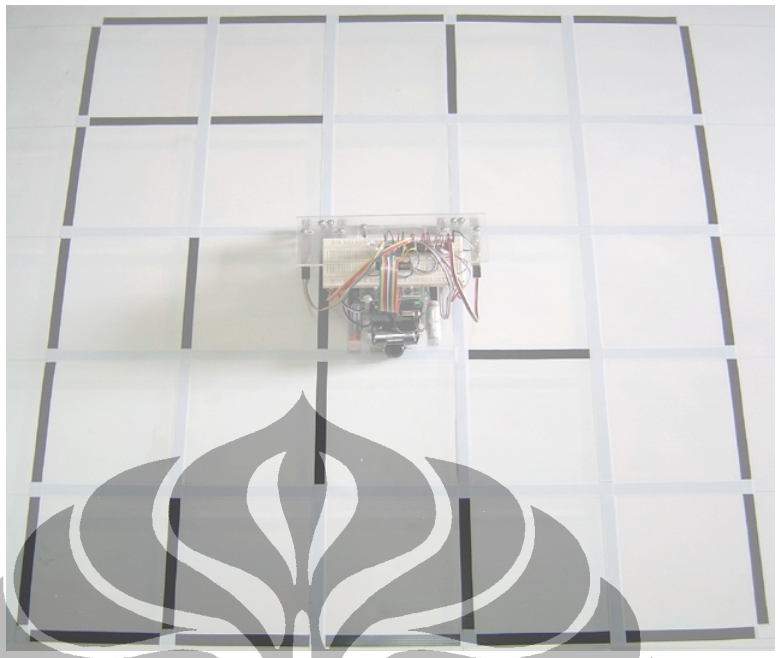
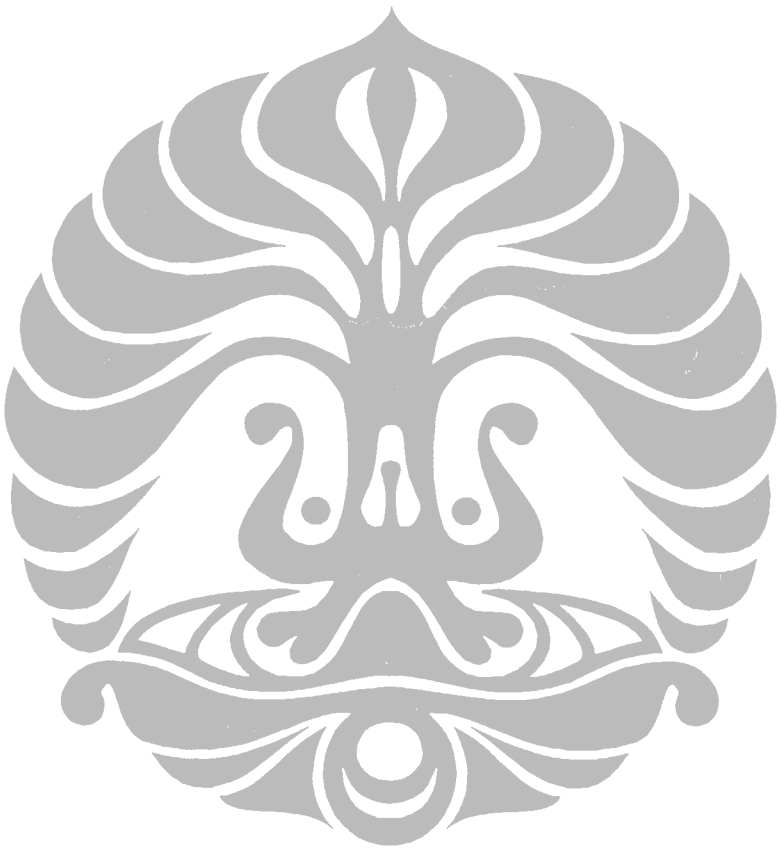


Foto 3. *Cell* Tujuan Robot Pencari Jalur di Lingkungan Labirin Dua Dimensi

Lampiran 2.

Paper pada "The 10th Quality In Research (QIR) International Conference"

University of Indonesia, December 4-6, 2007



Design and Implementation of Artificial Intelligence of Path Searching Robot Based on Microcontroller BASIC Stamp

Harry Sudibyo Soetjokro*, and Gede Indrawan†

* Fac. of Engineering, Universitas Indonesia (UI), Depok, 16424
Tel. 62-021-7270077, fax. 62-021-7270077, email: harisudi@ee.ui.ac.id
†Fac. of Engineering, Universitas Pendidikan Ganesha (Undiksha), Singaraja - Bali
Tel. 62-0362-22570, fax. 62-0362-25735, email: gede.indrawan@gmail.com

Abstract– **Artificial intelligence in robotics, as a smart algorithm programmed into the robot, is needed by the robot to help human to do some work automatically and in autonomous way. In this research, the implanted artificial intelligence is designed for path searching, included in mapping activity, from starting point at corner to destination point at center, in controlled environment like maze. General speaking, this research want to contribute in knowledge development under robotics domain of autonomous position-sensing and navigation.**

Robot design for this artificial intelligence consists of two aspects, i.e. robot prototype itself, and maze environment where path-searching robot will run. Implementation of robot involves three aspects, i.e. input (sensor to capture information from the environment), process (processor and its supporting system as robot brain for data processing), and output (as result of data processing, it can be signal for controlling the motor, etc).

The artificial intelligence in this research is implemented using PBASIC programming language, with BASIC Stamp BS2px from Parallax as targeted microcontroller. Multi bank programming style is used to utilize 16 KB internal EEPROM resource, comprise of eight memory bank with 2 KB capacity respectively, to save program code. This code supports robot function for path searching in maze, by using Flood-Fill algorithm and modified Flood-Fill algorithm as main algorithms. Flexibility and scalability are two concepts of this artificial intelligence to accommodate features addition and to anticipate more complex maze environment.

Keywords– Artificial Intelligence, robot, autonomous position-sensing and navigation, microcontroller, PBASIC, Basic Stamp, multi bank programming, Flood-Fill algorithm¸modified Flood-Fill algorithm

I. INTRODUCTION

Robot, to associate behaviors with a place (localization) requires to know where it is and to be able to navigate point-to-point. Such navigation began with wire-guidance in the 1970s and progressed in the early 2000s to beacon-based triangulation.

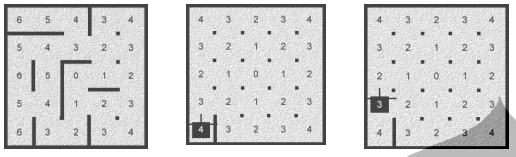
For indoor application, current commercial robots autonomously navigate based on sensing natural features. The first commercial robots to achieve this were Pyxus' HelpMate hospital robot and the CyberMotion guard robot, both designed in the 1980s. These robots originally used manually created CAD floor plans, sonar sensing and wall-following variations to navigate buildings. The next generation, such as MobileRobots' PatrolBot and autonomous wheelchair both introduced in 2004, have the ability to create their own laser-based maps of a building and to navigate open areas as well as corridors. Their control system changes its path on-the-fly if something blocks the way [1].

At the other side, outdoor autonomy is most easily achieved in the air, since obstacles are rare. Cruise missiles are rather dangerous highly autonomous robots. Some of unmanned aerial vehicles (UAVs) are capable of flying their entire mission without any human interaction at all except possibly for the landing where a person intervenes using radio remote control. But some drone aircraft are capable of a safe, automatic landing also.

Outdoor autonomy is the most difficult for ground vehicles, due to: a) 3-dimensional terrain, b) great disparities in surface density, c) weather exigencies, and d) instability of the sensed environment. The Mars rovers MER-A and MER-B can find the position of the sun and navigate their own routes to destinations on-the-fly by: a) mapping the surface with 3-D vision, b) computing safe and unsafe areas on the surface within that field of vision, c) computing optimal paths across the safe area towards the desired destination, d) driving along the calculated route, e) repeating this cycle until either the destination is reached, or there is no known path to the destination

II. BASIC THEORY

The objective of this research is to create artificial intelligence for path searching robot from starting cell at corner to destination cell at center of the maze without breaking 2-dimension walls, represented by black line that exist in different side on each cell (Fig.1a). Fig.1b illustrates how at starting cell, robot has no information about walls except at the current cell where robot is standing on and detect wall using its sensors.



(a) (b) (c)

Fig.1. (a) Maze with wall (b) Maze view by robot at start (c) Robot move to other cell

To solve this path searching problem, Flood-Fill algorithm and modified Flood-Fill algorithm are used as main algorithms for this artificial intelligence.

The Flood-Fill algorithm involves assigning values to each of the cells in the maze where these values represent the distance from any cell on the maze to the destination cell without breaking the wall. The destination cell, therefore, is assigned a value of 0. If the path searching robot is standing in a cell with a value of 1, it is 1 cell away from the goal. If the robot is standing in a cell with a value of 3, it is 3 cells away from the goal. Fig.1b represents initial value for every cell known by robot at start with initial assumption there is no wall at all.

For the maze at Fig.1, we would have 5 rows by 5 columns = 25 cell values. Therefore we would need 25 bytes to store the distance values for a complete maze. Actually, for this research, we have designed other data needed by a cell, as shown by Table 1 below.

Table 1. Cell data design (put in Scratchpad RAM)

CoordinateValue (Address: 0 - 24)	
7 6 5 4 3 2 1 0	x-axis = nibble1 (bit 7 … 4) y-axis = nibble0 (bit 3 … 0)
DistanceValue (Address: 25 - 49)	
7 6 5 4 3 2 1 0	Minimum = 0 (00000000b) Maximum = 255 (11111111b)
WallValue (Address: 50 - 74)	
7 6 5 4 3 2 1 0	Bit 6 = Active path to finish Bit 5 = Side wall check status Bit 4 = Checkpoint status Bit 3 = West (W) wall Bit 2 = South (S) wall Bit 1 = East (E) wall Bit 0 = North (N) wall
OrientationValue (Address: 75 - 99)	
7 6 5 4 3 2 1 0	nibble1 for second W, S, E, N nibble0 for first W, S, E, N

When it comes time to make a move, the robot must examine all adjacent cells which are not separated by walls and choose the one with the lowest distance value. In Fig.1c above, the robot would ignore any cell to the West (left side) because there is a wall, and it would look at the distance values of the cells to the North, East, and South since those are not separated by walls. The cell to the North has a value of 2, the cell to the East has a value of 2 and the cell to the South has a value of 4. The routine sorts the values to determine which cell has the lowest distance value. It turns out that both the North and East cells have a distance value of 2. That means that the robot can go North or East and traverse the same number of cells on its way to the destination cell. Since turning would take time, the robot will choose to go forward to the North cell. Fig.2a gives generic algorithm for that case.

Furthermore, every interior wall is shared by two cells so when robot update the wall value for one cell, robot can update the wall value for its neighbor as well (Fig.2b).

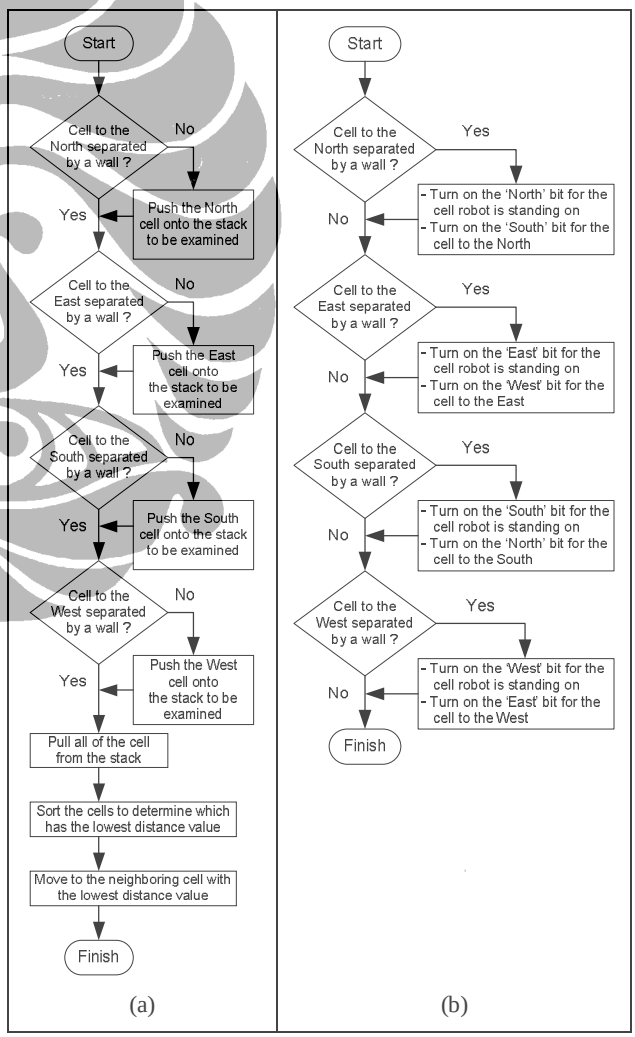


Fig.2. (a) Deciding lowest distance value (b) Updating wall map

The instructions for flooding the maze with distance values could be like algorithm at Fig.3 below.

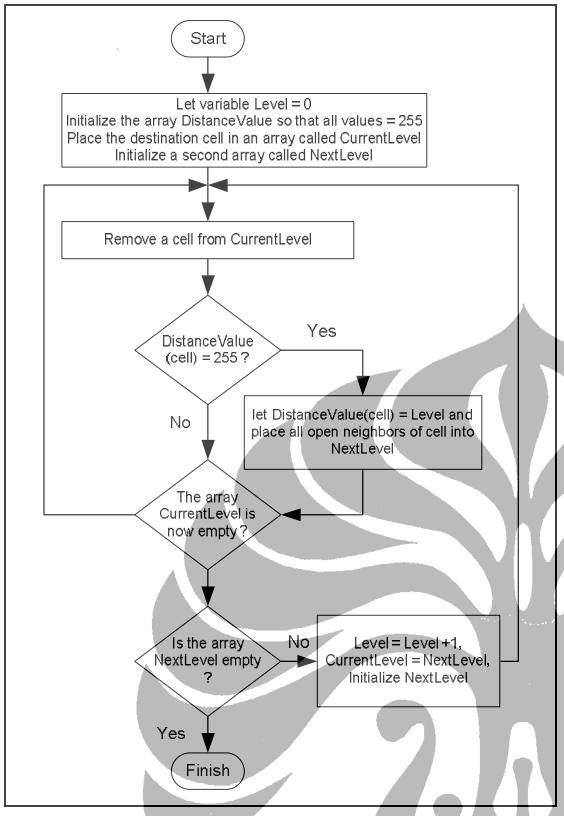


Fig.3. Flooding with distance values for Flood-Fill algorithm

The modified Flood-Fill algorithm at the other side is similar to the regular Flood-Fill algorithm in that the robot uses distance values to move about the maze. The distance values which represent how far the robot is from the destination cell, are followed in descending order until the robot reaches its goal. As the robot finds new walls during its exploration, the distance values need to be updated. Instead of flooding the entire maze with values, as is the case with the regular Flood-Fill, the modified Flood-Fill only changes those values which need to be changed.

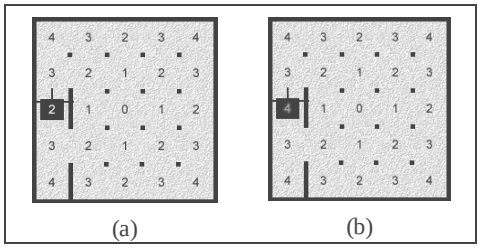


Fig.4. (a) Old distance value (b) New distance value

Fig.4 shows how our robot moves forward one cell and discovers a wall. The robot cannot go West and it cannot go East, it can only travel North or South. But going North or South means going up in distance values. So the cell values need to be updated. When the robot encounters this, it follows this rule: *"If a cell is not the destination cell, its value should be one plus the minimum value of its open neighbors"*. In the case above, the minimum value of its open neighbors is 3. Adding 1 to this value, results in 3 + 1 = 4. The maze now looks like Fig.4b.

There are times when updating a cell's value will cause its neighbors to violate the *"1 + minimum value"* rule and so they must be checked as well. We can see in our example above that the cells to the North and to the South have neighbors whose minimum value is 2. Adding a 1 to this value results in 2 + 1 = 3 therefore the cells to the North and to the South do not violate the rule and the updating routine is done. Now that the cell values have been updated, the robot can once again follow the distance values in descending order.

So our modified Flood-Fill procedure for updating the distance values is looked like Fig.5 below.

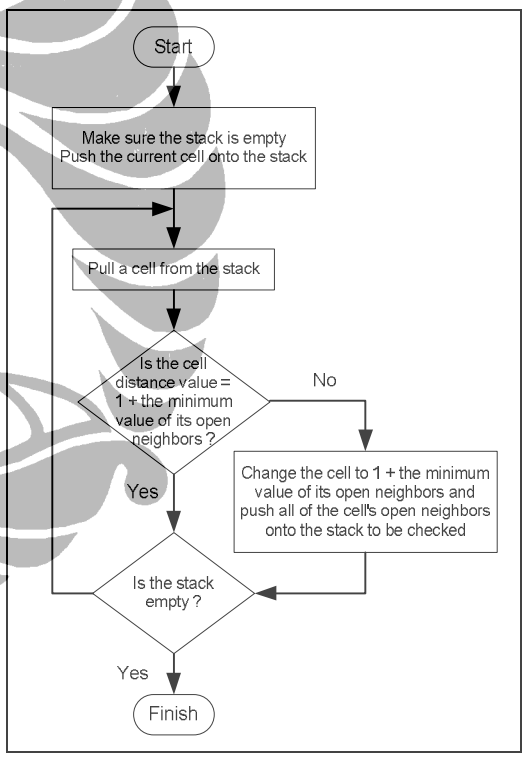


Fig.5. Flooding with distance values (if necessary) for modified Flood-Fill algorithm

As summary for both of algorithm, every time robot arrives in a cell, the Flood-Fill algorithm will perform the following steps [2], i.e.: 1) Update the wall map, 2) Flood the maze with new distance values (if necessary for modified Flood-Fill algorithm), 3) Decide which neighboring cell has the lowest distance value, 4) Move to the neighboring cell with the lowest distance value.

III. EXPERIMENTAL RESULTS

In this research, experiment is conducted to confirm the functionality of our artificial intelligence for path searching robot from starting cell to destination cell in the maze.

The artificial intelligence itself will control the system with simple diagram block (Fig.6a) that always consist of input, process, and output section.

Input section receives wall information from the maze using infra red reflective object sensor Fairchild Semiconductor QRD1114.

Output section for robot maneuver -- in order to avoid breaking the wall, is handled by continuous rotation servo that received signal from Process section to control motor speed and direction.

Process section is handled by microcontroller BASIC Stamp BS2px and its supporting system. Here, the artificial intelligence is implanted with two main algorithms, as shown by Fig.6a. Information of maze map size, with initial distance value for every cell, also initialized at this section. Of course, at initialization, robot does not know about wall map information.

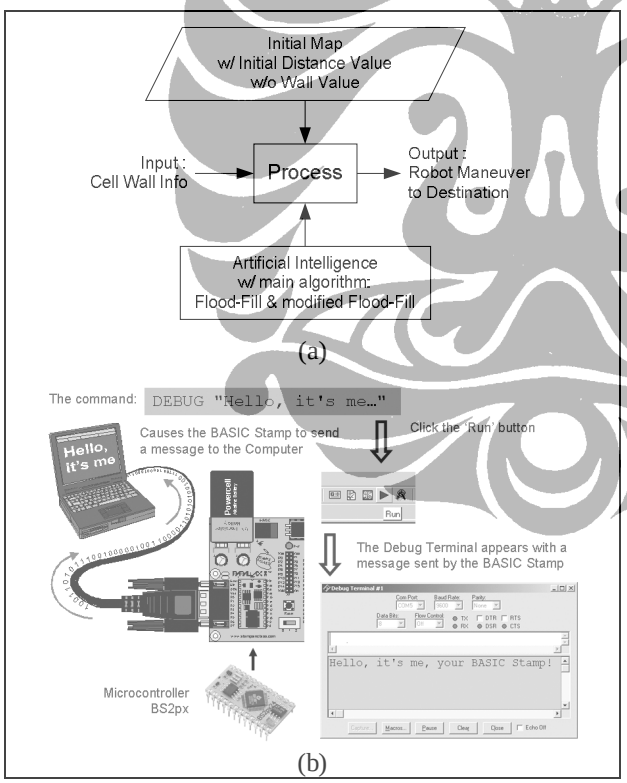


Fig.6. (a) System diagram block (b) Implementation of block process in BS2px development environment [3]

Fig.6b illustrates how the artificial intelligence was developed and implanted to the microcontroller BS2px with specification provided by Table 2.

Table 2. BS2px Specification [4]

Package	24-pin DIP
Package Size (L x W x H)	1.2" x 0.6" x 0.4"
Environment*	0º - 70º C (32º - 158º F)
Microcontroller	Parallax SX48
Processor Speed	32 MHz Turbo
Program Execution Speed	~19,000 instructions/sec.
RAM Size	38 Bytes (12 I/O, 26 Variable)
Scratch Pad RAM	128 Bytes
EEPROM (Program) Size	8 x 2K Bytes, ~4,000 inst.
Number of I/O pins	16 + 2 Dedicated Serial
Voltage Requirements	5 - 12 vdc
Current Draw @ 5V	55 mA Run / 450 µA Sleep
Source / Sink Current per I/O	30 mA / 30 mA
Source / Sink Current per unit	60 mA / 60 mA per 8 I/O pins
PBASIC Commands***	63
PC Programming Interface	Serial (19200 baud)
Windows Text Editor	Stampw.exe (v2.2 and up)

* 70% Non-Condensing Humidity
*** Using PBASIC 2.5 for BS2-type models

Experiment is conducted using some maneuvers as illustrated by Fig.7 and internal data processing is monitored via PC using serial communication, as shown by Fig.8a. Monitoring system itself involves debugging code that was embedded to the artificial intelligence, so it will make life easier to detect and repair any existing bug. Fig.10 shows the monitoring system in detail.

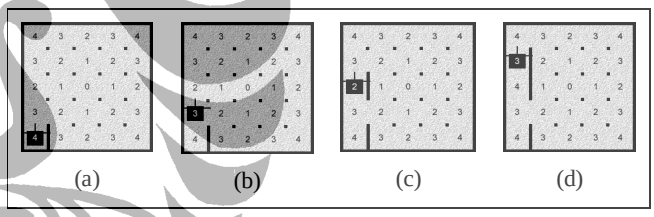


Fig.7. Prototype maneuver

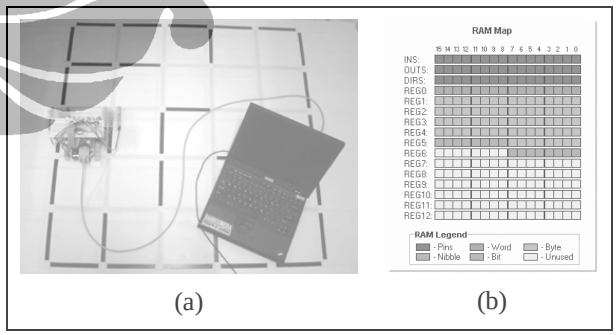
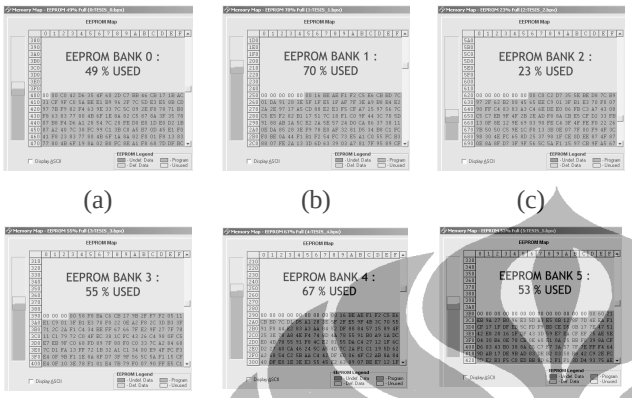


Fig.8. (a) Prototype testing (b) Usage of RAM

Microcontroller BS2px has three types of internal memory resource, i.e. 1) EEPROM for storing program code, 2) RAM for assigned and temporary variables, and 3) Scratchpad RAM for storing cell's data (see Table 1). Fig.8b shows internal RAM resource used by assigned variables that construct this artificial intelligence. RAM itself has total capacity of 13 Words (REG0 - REG12).

Using multi bank programming style, program code for artificial intelligence is stored from EEPROM bank 0 to bank 5 with its own usage percentage shown by Fig.9. Internal EEPROM itself comprised of eight memory banks (bank 0 to bank 7) with capacity 2 KB in each, together form total capacity of 16 KB.



(a) (b) (c)

(d) (e) (f)

Fig.9. Usage of EEPROM Bank

Fig.10 shows the monitoring process for some maneuvers of path searching robot, as illustrated by Fig.7. At robot position like Fig.7c, monitoring result is shown by Fig.10a. At robot position like Fig.7d, monitoring result is shown by Fig.10b.

What monitoring process want to tell us is that modified Flood-Fill algorithm has already work. As summary for that algorithm, every time robot arrives in a cell, the Flood-Fill algorithm will perform the following steps:

1. Update the wall map
Fig.10a gives Wall value = 00011010b at cell with Coordinate value = (0, 2). From Table 1, it means that robot facing North because bit (7-6) = 00b. The cell has already been checked for side wall and front-rear wall (has reach checkpoint) because bit (4) = 1 and bit (5) = 1. Based on that, it is no need again for gathering wall info that has already got, i.e. there are walls at West and East side because bit (3) = 1 and bit (1) = 1.

2. Flood the maze with new distance values (if necessary)
The cell at (0, 2) need to update its distance value because its recent distance value = 2, violates rule: *"If a cell is not the destination cell, its value should be one plus the minimum value of its open neighbors"*. Artificial intelligence updates this value. Fig.10b show updated distance value = 4.

3. Decide which neighboring cell has the lowest distance
Fig.10a shows there are two open neighboring cells with lowest distance = 3, i.e. cell at (0,1) and (0,3). The artificial intelligence choose cell (0,3) to move on because robot just go forward rather than turning that take more time.

4. Move to the neighboring cell with the lowest distance
Fig.10b shows robot move from cell at (0,2) to cell at (0,3). It proves this step.

Fig.10. Monitoring for prototype maneuver

IV. CONCLUSIONS

Path searching from starting cell to destination cell has been accomplished by the artificial intelligence using Flood-Fill algorithm and modified Flood-Fill algorithm.

REFERENCES

[1] Wikipedia, "Autonomous robot", 2007, http://id.wikipedia.org/wiki/Robot

[2] Steve Benkovic, "*Hints, Ideas, Inspiration for Mice Builders*", 2007, http:// micromouseinfo.com/.

[3] Andy Lindsay, "*Robotics with the Boe-Bot, Student Guide*", Parallax, Inc. Press, California, 2004.

[4] Parallax Inc., "*Stamp Specifications*", 2007, http://parallax.com/Portals/0/Downloads/docs/prod/stamps/stampscomparison.pdf.

Lampiran 4.

Kode Program Kecerdasan-Buatan Robot Pencari Jalur

Header (Ada di setiap slot EEPROM yang digunakan)

Kode Program Bank 0 EEPROM

Kode Program Bank 1 EEPROM

Kode Program Bank 2 EEPROM

Kode Program Bank 3 EEPROM

Kode Program Bank 4 EEPROM

Kode Program Bank 5 EEPROM

Kode Program Bank 6 EEPROM

Header (Ada di setiap slot EEPROM yang digunakan)

```
' -----[ Title ]-----------------------------------------------------------------
'
' File..... TESIS.BPX
' Purpose... Micromouse Style Path Searching Robot
' Author.... Gede Indrawan
' E-mail.... gede.indrawan@ui.edu

' {$STAMP BS2px, TESIS_1.bpx, TESIS_2.bpx, TESIS_3.bpx, TESIS_4.bpx, TESIS_5.bpx}
' {$PBASIC 2.5}


' -----[ Program Description ]---------------------------------------------------
'
' This program is used to search the path at maze environment.

' -----[ Revision History ]------------------------------------------------------
'
' DEC. 25TH, 2007 - Version 1.0

' -----[ Conditional Compilation Directive ]--------------------------------------
'
'#DEFINE DebugLow     = 1  'ADD ", TESIS_6.bpx" AT $STAMP DIRECTIVE AT SLOT 0
'#DEFINE DebugNormal  = 1
#DEFINE DebugLow     = 0
#DEFINE DebugNormal  = 0

' -----[ I/O Definitions ]--------------------------------------------------------
'
' PROXIMITY SENSOR ARRAY
' -------- FRONT --------
' 07 06 05 04 03 02 01 00
' L7 L6 L5 L4 R3 R2 R1 R0
' -------- REAR! --------

OuterRightSensor      PIN  0   ' PIN 0-11: ON/OFF SENSOR
FarMiddleRightSensor  PIN  1
NearMiddleRightSensor PIN  2
InnerRightSensor      PIN  3

InnerLeftSensor       PIN  4
NearMiddleLeftSensor  PIN  5
FarMiddleLeftSensor   PIN  6
OuterLeftSensor       PIN  7

OuterRearRightSensor  PIN  8
InnerRearRightSensor  PIN  9

InnerRearLefttSensor  PIN  10
OuterRearLeftSensor   PIN  11

RightMotor            PIN  12  ' Servo motor connections
LeftMotor             PIN  13

SensorInput           PIN  14  ' INPUT FOR ALL SENSORS

UnusedPin             PIN  15  ' Unused I/O pin

' -----[ Constants ]-------------------------------------------------------------
'
Pause4BigLoop         CON  100

EqualTo15             CON  15   ' %1111
EqualTo255            CON  255  ' %11111111

IsLow                 CON  0
IsHigh                CON  1

MatrixSize            CON  5  ' Maximum for Matrix Size = 16
MatrixSizeSq          CON  MatrixSize*MatrixSize
MatrixSizeMinSatu     CON  MatrixSize-1
MatrixSizeSqMinSatu   CON  MatrixSizeSq-1

' DATA DESIGN AT SCRATCHPAD RAM
' HIGHEST LOCATION AT 127 IS READ ONLY (CONTAIN NUMBER OF RUNNING SLOT)
CoordinateValue       CON  MatrixSizeSq*0  ' SPRAM Address =   0 -  24
DistanceValue         CON  MatrixSizeSq*1  ' SPRAM Address =  25 -  49
WallValue             CON  MatrixSizeSq*2  ' SPRAM Address =  50 -  74
OrientValue           CON  MatrixSizeSq*3  ' SPRAM Address =  75 -  99
StackValue            CON  MatrixSizeSq*4  ' SPRAM Address = 100 - 115
StackMaxAddress       CON  115
SensorThres           CON  125  ' Stores black/white threshold = 125 - 126
```

```
GoStraight          CON  0
RotateRight90        CON  1
RotateLeft90        CON  2
RotateRight180       CON  3

GoStraightFactor     CON  30
Loop4GoStraight      CON  60
Pause4GoStraight     CON  20

RotateFactor        CON  30
Loop4Rotate90        CON  90  ' DEBUG: ON = 100, OFF = 90
Pause4Rotate90       CON  20
Loop4Rotate180       CON  180  ' DEBUG: ON = 200, OFF = 180
Pause4Rotate180      CON  20

Loop4PivotLeftScan    CON  70
Loop4PivotRightScan   CON  2*Loop4PivotLeftScan
Pause4Scan           CON  20

GoForwardFactor      CON  30
Loop4GoForward       CON  4
Pause4GoForward      CON  20

PivotFactor          CON  30
Loop4Pivot           CON  2
Pause4Pivot          CON  20
StopMotor            CON  1850  ' stop motor for BS2px (750 for BS2)

TskDoInput           CON  1  ' Program Task at Memory Bank 0

Bank_0               CON  0  ' Memory Bank 0
Bank_1               CON  1  ' Memory Bank 1
Bank_2               CON  2  ' Memory Bank 2
Bank_3               CON  3  ' Memory Bank 3
Bank_4               CON  4  ' Memory Bank 4
Bank_5               CON  5  ' Memory Bank 5
Bank_6               CON  6  ' Memory Bank 6
Bank_7               CON  7  ' Memory Bank 7

North               CON  0
East                CON  1
South               CON  2
West                CON  3

'CodeOverhead          CON  587  ' BS2   -> 220 * 2us = 440us
                       ' BS2px -> 440us / 0.75us = 587
OneOpenNeighbour      CON  1
TwoOpenNeighbours     CON  2
ThreeOpenNeighbours   CON  3

' -----[ Variables ]----------------------------------------------------------
'
wallSensors          VAR  Word
sensor0              VAR  wallSensors.BIT0
sensor1              VAR  wallSensors.BIT1
sensor2              VAR  wallSensors.BIT2
sensor3              VAR  wallSensors.BIT3
sensor4              VAR  wallSensors.BIT4
sensor5              VAR  wallSensors.BIT5
sensor6              VAR  wallSensors.BIT6
sensor7              VAR  wallSensors.BIT7
sensor8              VAR  wallSensors.BIT8
sensor9              VAR  wallSensors.BIT9
sensor10             VAR  wallSensors.BIT10
sensor11             VAR  wallSensors.BIT11
index               VAR  wallSensors.NIB3

'sensorThreshold       VAR  Word  ' Stores black/white threshold

bitparams            VAR  Word
isBrokenRule          VAR  bitparams.BIT0
isCheckpoint          VAR  bitparams.BIT1
isFinish             VAR  bitparams.BIT2
isNoWall             VAR  bitparams.BIT3
isSideWall           VAR  bitparams.BIT4
isSetPath            VAR  bitparams.BIT5
isVisitedCell        VAR  bitparams.BIT6
scanResult           VAR  bitparams.BIT7
task                VAR  bitparams.BIT8
moveType             VAR  bitparams.NIB3

coordinate           VAR  Word
tempCoord            VAR  coordinate.BYTE1
nowCoord             VAR  coordinate.BYTE0
```

```
tempX           VAR  tempCoord.NIB1
tempY           VAR  tempCoord.NIB0
nowX            VAR  nowCoord.NIB1
nowY            VAR  nowCoord.NIB0

adjCoordArr      VAR  Byte(4)

openDistance     VAR  Byte
numOfOpenDist      VAR  openDistance.NIB1
openDist         VAR  openDistance.NIB0

nowWall          VAR  Byte

nowOrient        VAR  Byte

faceIn           VAR  Byte
tempFaceIn         VAR  faceIn.NIB1
nowFaceIn          VAR  faceIn.NIB0

adjDistArr       VAR  Nib(4)

nowDist          VAR  Nib

stackPointer     VAR  Nib

' -----[ EEPROM Data ]----------------------------------------------------------
'
' WallValue
' Bit : 7 6       -> FACING TO
'       0 0       -> NORTH
'       0 1       -> EAST
'       1 0       -> SOUTH
'       1 1       -> WEST
' Bit : 5 4       -> WALL STATUS
'       0 0       -> NO WALL HAVE BEEN CHECKED
'       0 1       -> WEST & EAST WALLS HAVE BEEN CHECKED
'       1 0       -> NORTH & SOUTH WALLS HAVE BEEN CHECKED
'       1 1       -> WEST, EAST, NORTH, SOUTH WALLS HAVE BEEN CHECKED
' Bit : 3 2 1 0    -> WALL EXISTENCE FLAG
'       W S E N     (W = WEST, S = SOUTH, E = EAST, N = NORTH)
'
' DistanceValue    -> NUMBER OF CELL TO BE TRAVELLED FROM CURRENT TO TARGET CELL
'
' CoordinateValue  -> COORDINATE OF CURRENT CELL
' Nibble1 : 7 6 5 4 -> X-AXIS
' Nibble0 : 3 2 1 0 -> Y-AXIS

'               DATA     @CoordinateValue, (MatrixSize*MatrixSize)
'               DATA     @DistanceValue,   (MatrixSize*MatrixSize)
'               DATA     @WallValue,       (MatrixSize*MatrixSize)
'ResetValue     DATA     $FF
```

Kode Program Bank 0 EEPROM

```
' -----[ Initialization ]----------------------------------------------------
'
'Check_Run:
'  READ ResetValue, B25                  ' get reset value
'  B25 = ~B25                           ' invert bits
'  WRITE ResetValue, B25                 ' write inverted bits back
'  IF (B25) THEN Initialize              ' run if inverted > 0

'No_Run:
'  END                                  ' low power mode

' -----[ Main Code ]----------------------------------------------------------
'
  'PAUSE Pause4BigLoop  ' pause for big looping of system

  BRANCH task, [Do_Init, Do_Input]

' ---------------------------------------
Do_Init:

  #IF DebugLow #THEN
    DEBUG CR, "INIT:", CR
  #ENDIF

  GOSUB Init_Map
  GOSUB Init_Stack
  GOSUB Calibrate_Sensors
  GOSUB Init_Start

' ---------------------------------------
Do_Input:

  #IF DebugLow #THEN
    DEBUG CR, "INPUT:", CR
  #ENDIF

  GOSUB Prepare_Cell_Paramaters
  GOSUB Read_Wall_Sensors

' ---------------------------------------
Do_Process_1:
  RUN Bank_1

' -----[ Subroutines ]--------------------------------------------------------
'
' ---------------------------------------
Init_Map:

  #IF DebugLow #THEN
    DEBUG CR, "INIT_MAP:", CR
  #ENDIF

  B24 = MatrixSize
  B23 = 0  ' PLAY SAVE, MAKE IT 0 BEFORE

  ' GENERATE INITIAL MAP FOR MATRIX SIZE WITH ODD VALUE (5, 7, 9, ...)
  IF (B24.BIT0) THEN
    FOR nowX = 0 TO MatrixSizeMinSatu
      FOR nowY = 0 TO MatrixSizeMinSatu
        IF (nowX < (MatrixSizeMinSatu / 2)) THEN
          IF (nowY < (MatrixSizeMinSatu / 2)) THEN
            nowDist = (MatrixSizeMinSatu - nowX) - nowY
          ELSE
            nowDist = nowY - nowX
          ENDIF
        ELSE
          IF (nowY < (MatrixSizeMinSatu / 2)) THEN
            nowDist = nowX - nowY
          ELSE
            nowDist = nowY - (MatrixSizeMinSatu - nowX)
          ENDIF
        ENDIF
        B24 = CoordinateValue + B23
        B25 = nowCoord
        GOSUB Put_Byte
        B24 = DistanceValue + B23
        B25 = nowDist
        GOSUB Put_Byte
        B23 = B23 + 1
      NEXT
    NEXT
```

```
' ' GENERATE INITIAL MAP FOR MATRIX SIZE WITH EVEN VALUE (6, 8, 10, ...)
' ELSE
'   FOR nowX = 0 TO (MatrixSize - 2)
'     FOR nowY = 0 TO (MatrixSize - 2)
'       IF (nowX < ((MatrixSize - 2) / 2)) THEN
'         IF (nowY < ((MatrixSize - 2) / 2)) THEN
'           nowDist = ((MatrixSize - 2) - nowX) - nowY
'         ELSE
'           IF (nowY >= (((MatrixSize - 2) / 2) + 1)) THEN
'             nowDist = (nowY - 1) - nowX
'           ELSE
'             nowDist = nowY - nowX
'           ENDIF
'         ENDIF
'       ELSE
'         IF (nowX >= (((MatrixSize - 2) / 2) + 1)) THEN
'           IF (nowY < ((MatrixSize - 2)/ 2)) THEN
'             nowDist = (nowX - 1) - nowY
'           ELSE
'             IF (nowY >= (((MatrixSize - 2) / 2) + 1)) THEN
'               nowDist = (nowY - 1) - ((MatrixSize - 2) - (nowX - 1))
'             ELSE
'               nowDist = nowY - ((MatrixSize - 2)-(nowX - 1))
'             ENDIF
'           ENDIF
'         ELSE
'           IF (nowY < ((MatrixSize - 2) / 2)) THEN
'             nowDist = nowX - nowY
'           ELSE
'             IF (nowY >= (((MatrixSize - 2) / 2) + 1)) THEN
'               nowDist = (nowY - 1) - nowX
'             ELSE
'               nowDist = nowY - nowX
'             ENDIF
'           ENDIF
'         ENDIF
'       ENDIF
'       B24 = CoordinateValue + B23
'       B25 = nowCoord
'       GOSUB Put_Byte
'       B24 = DistanceValue + B23
'       B25 = nowDist
'       GOSUB Put_Byte
'       B23 = B23 + 1
'     NEXT
'   NEXT

  ENDIF

  '#IF DebugNormal #THEN
  '  GOSUB Maze_Reading
  '#ENDIF

  RETURN

' ----------------------------------------
Init_Stack:

  #IF DebugLow #THEN
    DEBUG CR, "INIT_STACK:", CR
  #ENDIF

  FOR B24 = StackValue TO StackMaxAddress

    B25 = EqualTo255
    GOSUB Put_Byte

    #IF DebugNormal #THEN
      GOSUB Get_Byte
      IF (B24 = 107 OR B24 = 115) THEN
        DEBUG "[", DEC B24, "] = ", IHEX2 B25, CR
      ELSE
        DEBUG "[", DEC B24, "] = ", IHEX2 B25, "; "
      ENDIF
    #ENDIF

  NEXT

  RETURN
```

```
' ----------------------------------------
Calibrate_Sensors:

  #IF DebugLow #THEN
    DEBUG CR, "CALIBRATE_SENSORS:", CR
  #ENDIF

  HIGH FarMiddleLeftSensor              ' Turn SENSOR 6 on
  HIGH SensorInput                    ' Discharge capacitor
  PAUSE 1
  'RCTIME SensorInput, 1, sensorThreshold  ' Measure charge time
  RCTIME SensorInput, 1, W9  ' Measure charge time
  LOW FarMiddleLeftSensor               ' Turn SENSOR 6 off

  'sensorThreshold = sensorThreshold / 3   ' Take 1/4 average
  W9 = W9 / 3   ' Take 1/4 average

  'IF sensorThreshold > CodeOverhead THEN  ' Account for code overhead
  '  sensorThreshold = sensorThreshold - CodeOverhead
  'ELSE
  '  sensorThreshold = 0
  'ENDIF

  B20 = SensorThres
  GOSUB Put_Word:
  'GOSUB Get_Word:

  #IF DebugNormal #THEN
    'DEBUG ? sensorThreshold
    DEBUG "SensorThreshold = ", DEC W9, CR
  #ENDIF

  RETURN

' ----------------------------------------
Init_Start:

  #IF DebugLow #THEN
    DEBUG CR, "INIT_START", CR
  #ENDIF

  nowCoord = 0
  tempCoord = nowCoord
  nowFaceIn = 0
  GOSUB Update_Orientation:

  #IF DebugNormal #THEN
    DEBUG "nowCoord = ", IHEX2 nowCoord, "; ","tempCoord = ", IHEX2 tempCoord, CR
    DEBUG "nowFaceIn = ", IBIN4 nowFaceIn, "; ", "nowOrient = ", BIN4 nowOrient.NIB1, "-", BIN4 nowOrient.NIB0, CR
  #ENDIF

  RETURN

' ----------------------------------------
' PREPARE PARAMETERS FROM PREVIOUS PROCESS BEFORE ENTERING BIG LOOPING PROCESS AGAIN
Prepare_Cell_Paramaters:

  #IF DebugLow #THEN
    DEBUG CR, "PREP_CELL_PARAMS:", CR
  #ENDIF

  #IF DebugNormal #THEN
    DEBUG ? isCheckpoint
  #ENDIF

  IF (isCheckpoint) THEN

    isCheckpoint = isLow

    ' TAKE CARE OF NEW VALUE : COORDINATE VALUE ----------
    nowCoord = tempCoord

    ' CHECK HAS REACHED START CELL AT GO BACK HOME
    GOSUB Get_Start_Status

    ' TAKE CARE OF NEW VALUE : DISTANCE VALUE ------------
    ' CHECK HAS REACHED DESTINATION CELL
    GOSUB Get_Finish_Status

    ' TAKE CARE OF NEW VALUE : WALL VALUE ----------------
    ' CHECK VISIT STATUS
    GOSUB Get_Visit_Status

    ' SET PATH STATUS
    GOSUB Set_Path_Status
```

```
    ' TAKE CARE OF NEW VALUE : ORIENTATION VALUE ---------
    nowFaceIn = tempFaceIn
    GOSUB Update_Orientation

    #IF DebugNormal #THEN
      DEBUG "nowCoord = ", IHEX2 nowCoord, "; ","tempCoord = ", IHEX2 tempCoord, CR
      DEBUG ? nowDist
      DEBUG "nowWall = ", BIN4 nowWall.NIB1, "-", BIN4 nowWall.NIB0, CR
      DEBUG "nowOrient = ", BIN4 nowOrient.NIB1, "-", BIN4 nowOrient.NIB0, CR
      DEBUG "nowFaceIn = ", IBIN4 nowFaceIn, "; ","tempFaceIn = ", IBIN4 tempFaceIn, CR
      DEBUG "isFinish = ", BIN isFinish, "; ", "isSetPath = ", BIN isSetPath, "; ", "isVisitedCell = ", BIN isVisitedCell, CR
    #ENDIF

  ENDIF

  RETURN

' ----------------------------------------
Get_Start_Status:

  IF (nowCoord = 0 AND isFinish = IsHigh) THEN
    isFinish = IsLow
  ENDIF

  RETURN

' ----------------------------------------
Get_Finish_Status:

  B24 = DistanceValue + ((nowX * MatrixSize) + nowY)
  GOSUB Get_Byte
  nowDist = B25

  IF (nowDist = 0) THEN
    isFinish = IsHigh
    isSetPath = IsHigh
  ENDIF

  RETURN

' ----------------------------------------
Get_Visit_Status:

  isVisitedCell = IsLow

  B24 = WallValue + ((nowX * MatrixSize) + nowY)
  GOSUB Get_Byte
  nowWall = B25

  IF (nowWall.BIT5 = IsHigh) AND (nowWall.BIT4 = IsHigh) THEN
    isVisitedCell = IsHigh
  ENDIF

  RETURN

' ----------------------------------------
Set_Path_Status:

  #IF DebugLow #THEN
    DEBUG CR, "SET_PATH:", CR
  #ENDIF

  IF (isSetPath) THEN

    #IF DebugLow #THEN
      DEBUG "Yes", CR
    #ENDIF

    isSetPath = IsLow

    FOR B23 = 0 TO MatrixSizeSqMinSatu

      B24 = OrientValue + B23
      GOSUB Get_Byte
      B22 = B25

      IF (B22 <> 0) THEN

        B24 = WallValue + B23
        GOSUB Get_Byte
        B25.BIT6 = IsHigh
        GOSUB Put_Byte
```

```
      #IF DebugNormal #THEN
        DEBUG "[", DEC B24, "] = ", BIN4 B22.NIB1, "-", BIN4 B22.NIB0, "; ", "[", DEC B24, "] = ", BIN4 B25.NIB1, "-",
BIN4 B25.NIB0, CR
      #ENDIF

    ENDIF

   NEXT

  ENDIF

  RETURN

' ----------------------------------------
Update_Orientation:

  #IF DebugLow #THEN
   DEBUG CR, "UPDATE_ORIENT:", CR
  #ENDIF

  IF (isFinish = IsLow) THEN

   #IF DebugLow #THEN
    DEBUG "Yes", CR
   #ENDIF

   B24 = OrientValue + ((nowX * MatrixSize) + nowY)
   GOSUB Get_Byte
   nowOrient = B25

   SELECT tempFaceIn
    CASE North

     IF (nowOrient.BIT0 = IsLow) THEN
      IF (nowOrient.BIT6) THEN
       nowOrient.BIT6 = IsLow
      ELSEIF (nowOrient.BIT2) THEN
       nowOrient.BIT2 = IsLow
      ELSE
       nowOrient.BIT0 = IsHigh
      ENDIF
     ELSEIF (nowOrient.BIT4 = IsLow) THEN
      IF (nowOrient.BIT6) THEN
       nowOrient.BIT6 = IsLow
      ELSEIF (nowOrient.BIT2) THEN
       nowOrient.BIT2 = IsLow
      ELSE
       nowOrient.BIT4 = IsHigh
      ENDIF
     ENDIF

    CASE East

     IF (nowOrient.BIT1 = IsLow) THEN
      IF (nowOrient.BIT7) THEN
       nowOrient.BIT7 = IsLow
      ELSEIF (nowOrient.BIT3) THEN
       nowOrient.BIT3 = IsLow
      ELSE
       nowOrient.BIT1 = IsHigh
      ENDIF
     ELSEIF (nowOrient.BIT5 = IsLow) THEN
      IF (nowOrient.BIT7) THEN
       nowOrient.BIT7 = IsLow
      ELSEIF (nowOrient.BIT3) THEN
       nowOrient.BIT3 = IsLow
      ELSE
       nowOrient.BIT5 = IsHigh
      ENDIF
     ENDIF

    CASE South

     IF (nowOrient.BIT2 = IsLow) THEN
      IF (nowOrient.BIT4) THEN
       nowOrient.BIT4 = IsLow
      ELSEIF (nowOrient.BIT0) THEN
       nowOrient.BIT0 = IsLow
      ELSE
       nowOrient.BIT2 = IsHigh
      ENDIF
     ELSEIF (nowOrient.BIT6 = IsLow) THEN
      IF (nowOrient.BIT4) THEN
       nowOrient.BIT4 = IsLow
      ELSEIF (nowOrient.BIT0) THEN
```

```
          nowOrient.BIT0 = IsLow
        ELSE
          nowOrient.BIT6 = IsHigh
        ENDIF
      ENDIF

    CASE West

      IF (nowOrient.BIT3 = IsLow) THEN
        IF (nowOrient.BIT5) THEN
          nowOrient.BIT5 = IsLow
        ELSEIF (nowOrient.BIT1) THEN
          nowOrient.BIT1 = IsLow
        ELSE
          nowOrient.BIT3 = IsHigh
        ENDIF
      ELSEIF (nowOrient.BIT7 = IsLow) THEN
        IF (nowOrient.BIT5) THEN
          nowOrient.BIT5 = IsLow
        ELSEIF (nowOrient.BIT1) THEN
          nowOrient.BIT1 = IsLow
        ELSE
          nowOrient.BIT7 = IsHigh
        ENDIF
      ENDIF

  ENDSELECT

  B25 = nowOrient
  GOSUB Put_Byte

ENDIF

#IF DebugNormal #THEN
  DEBUG "isFinish = ", BIN isFinish, CR, CR
#ENDIF

RETURN

' -------------------------------------
Read_Wall_Sensors:

#IF DebugLow #THEN
  DEBUG CR, "READ_WALL_SENSORS:", CR
#ENDIF

B20 = SensorThres
GOSUB Get_Word

'HIGH OuterRightSensor     ' Turn on sensor
'GOSUB Change_IO_Direction
'sensor0 = SensorInput     ' Snapshot of sensor signal states
'LOW OuterRightSensor      ' Turn off sensor

HIGH FarMiddleRightSensor
GOSUB Change_IO_Direction
sensor1 = SensorInput
LOW FarMiddleRightSensor

HIGH NearMiddleRightSensor
GOSUB Change_IO_Direction
sensor2 = SensorInput
LOW NearMiddleRightSensor

HIGH InnerRightSensor
GOSUB Change_IO_Direction
sensor3 = SensorInput
LOW InnerRightSensor

HIGH InnerLeftSensor
GOSUB Change_IO_Direction
sensor4 = SensorInput
LOW InnerLeftSensor

HIGH NearMiddleLeftSensor
GOSUB Change_IO_Direction
sensor5 = SensorInput
LOW NearMiddleLeftSensor

HIGH FarMiddleLeftSensor
GOSUB Change_IO_Direction
sensor6 = SensorInput
LOW FarMiddleLeftSensor
```

```
  'HIGH OuterLeftSensor
  'GOSUB Change_IO_Direction
  'sensor7 = SensorInput
  'LOW OuterLeftSensor

  'HIGH OuterRearRightSensor
  'GOSUB Change_IO_Direction
  'sensor8 = SensorInput
  'LOW OuterRearRightSensor

  'HIGH InnerRearRightSensor
  'GOSUB Change_IO_Direction
  'sensor9 = SensorInput
  'LOW InnerRearRightSensor

  'HIGH InnerRearLefttSensor
  'GOSUB Change_IO_Direction
  'sensor10 = SensorInput
  'LOW InnerRearLefttSensor

  'HIGH OuterRearLeftSensor
  'GOSUB Change_IO_Direction
  'sensor11 = SensorInput
  'LOW OuterRearLeftSensor

  #IF DebugNormal #THEN
    DEBUG "wallSensors = ", BIN4 wallSensors.NIB1, "-", BIN4 wallSensors.NIB0, CR
  #ENDIF

  RETURN

' -----------------------------------------
Change_IO_Direction:

  HIGH SensorInput                ' Push signal voltages to 5 V
  PAUSE 0                         ' Wait 230 us for capacitors
  INPUT SensorInput               ' Start the decays
  'PULSOUT UnusedPin, sensorThreshold  ' Wait for threshold time
  PULSOUT UnusedPin, W9           ' Wait for threshold time

  RETURN

' -----------------------------------------
Put_Byte:
  PUT B24, B25
  RETURN

' -----------------------------------------
Get_Byte:
  GET B24, B25
  RETURN

' -----------------------------------------
Put_Word:
  PUT B20, Word W9
  RETURN

' -----------------------------------------
Get_Word:
  GET B20, Word W9
  RETURN
```

Kode Program Bank 1 EEPROM

```
' -----[ Initialization ]------------------------------------------------------
'
' -----[ Main Code ]-------------------------------------------------------------
'
' ----------------------------------------
Do_Process_1:

  #IF DebugLow #THEN
    DEBUG CR, "PROCESS_1:", CR
  #ENDIF

  #IF DebugNormal #THEN
    DEBUG "nowCoord = ", IHEX2 nowCoord, "; ", "isVisitedCell = ", BIN isVisitedCell, "; ", "isCheckpoint = ", BIN
isCheckpoint, "; ", "isFinish = ", BIN isFinish, CR
  #ENDIF

  IF (isVisitedCell) THEN
    GOSUB Get_Checkpoint
  ELSE
    GOSUB Get_Wall
  ENDIF

  IF (isCheckpoint) THEN
    IF (isFinish) THEN
      GOTO Do_Process_2
    ELSE
      GOTO Do_Process_3
    ENDIF
  ELSE
    GOTO Do_Output_1
  ENDIF

' ----------------------------------------
Do_Process_2:  ' GO BACK HOME AFTER FINISH
  RUN Bank_2

' ----------------------------------------
Do_Process_3:  ' PROCESS DISTANCE VALUE
  RUN Bank_3

' ----------------------------------------
Do_Output_1:  ' MAKE MOVE
  RUN Bank_5

' -----[ Subroutines ]-----------------------------------------------------------
'
' ----------------------------------------
Get_Checkpoint:

  #IF DebugLow #THEN
    DEBUG CR, "GET_CHECKPOINT:", CR
  #ENDIF

  IF (sensor6 = IsLow AND sensor5 = IsLow) AND (sensor2 = IsLow AND sensor1 = IsLow) THEN
    isCheckpoint = IsHigh
  ENDIF

  #IF DebugNormal #THEN
    DEBUG ? isCheckpoint
  #ENDIF

  RETURN

' ----------------------------------------
Get_Wall:

  #IF DebugLow #THEN
    DEBUG CR, "GET_WALL:", CR
  #ENDIF

  #IF DebugNormal #THEN
    DEBUG "nowFaceIn = ", IBIN4 nowFaceIn, "; ", "nowWall[54] = %", BIN nowWall.BIT5, BIN nowWall.BIT4, CR
  #ENDIF

  GOSUB Process_Side_Wall
  GOSUB Process_Checkpoint
  GOSUB Process_Wall

  RETURN
```

```
' ----------------------------------------
Process_Side_Wall:

  '#IF DebugLow #THEN
  '  DEBUG CR, "PROC_SIDE_WALL", CR
  '#ENDIF

  IF (nowFaceIn = North OR nowFaceIn = South) THEN
    IF (nowWall.BIT4 = IsLow) THEN
      GOSUB Get_Side_Wall
    ENDIF

  ELSEIF (nowFaceIn = East OR nowFaceIn = West) THEN
    IF (nowWall.BIT5 = IsLow) THEN
      GOSUB Get_Side_Wall
    ENDIF

  ENDIF

  RETURN

' ----------------------------------------
Get_Side_Wall:

  #IF DebugLow #THEN
    DEBUG CR, "GET_SIDE_WALL:", CR
  #ENDIF

  IF (sensor6 = IsHigh AND sensor5 = IsHigh) OR (sensor2 = IsHigh AND sensor1 = IsHigh) THEN
    isSideWall = IsHigh
  ENDIF

  #IF DebugNormal #THEN
    DEBUG ? isSideWall
  #ENDIF

  RETURN
' ----------------------------------------
Process_Checkpoint:

  '#IF DebugLow #THEN
  '  DEBUG CR, "PROC_CHECKPOINT", CR
  '#ENDIF

  IF (nowFaceIn = North OR nowFaceIn = South) THEN
    IF (nowWall.BIT4 = IsHigh AND nowWall.BIT5 = IsLow) THEN
      GOSUB Get_Checkpoint
    ENDIF

  ELSEIF (nowFaceIn = East OR nowFaceIn = West) THEN
    IF (nowWall.BIT5 = IsHigh AND nowWall.BIT4 = IsLow) THEN
      GOSUB Get_Checkpoint
    ENDIF

  ENDIF

  RETURN

' ----------------------------------------
Process_Wall:

  #IF DebugLow #THEN
    DEBUG CR, "PROC_WALL:", CR
  #ENDIF

  #IF DebugNormal #THEN
    DEBUG "isSideWall = ", BIN isSideWall, "; ", "isCheckpoint = ", BIN isCheckpoint, CR
  #ENDIF

  IF (isSideWall OR isCheckpoint) THEN

    ' ---------- ROBOT FACING NORTH ----------------------------

    IF (nowFaceIn = North) THEN

      ' ---------- PROCESS SIDE WALL ----------------------------

      IF (nowWall.BIT4 = IsLow) THEN

        ' ----------------------------------------
        ' Set WEST WALL  = %00001000
        IF (nowWall.BIT3 = IsLow) THEN

          IF (nowX = 0) THEN
```

```
      nowWall.BIT3 = IsHigh
    ELSEIF (sensor6 OR sensor5) THEN
      nowWall.BIT3 = IsHigh
      ' HERE TO SET ADJACENT CELL'S WALL !!!
      tempX = nowX - 1
      tempY = nowY
      GOSUB Read_Adj_Wall
      IF (B25.BIT4 = IsLow) THEN
        B25.BIT1 = IsHigh  ' NEIGHBORHOOD CELL : EAST WALL  = %00000010
        GOSUB Put_Byte
      ENDIF
    ENDIF

  ENDIF

  ' ----------------------------------------
  ' Set EAST WALL  = %00000010
  IF (nowWall.BIT1 = IsLow) THEN

    IF (nowX = MatrixSizeMinSatu) THEN
      nowWall.BIT1 = IsHigh
    ELSEIF (sensor2 OR sensor1) THEN
      nowWall.BIT1 = IsHigh
      ' HERE TO SET ADJACENT CELL'S WALL !!!
      tempX = nowX + 1
      tempY = nowY
      GOSUB Read_Adj_Wall
      IF (B25.BIT4 = IsLow) THEN
        B25.BIT3 = IsHigh  ' NEIGHBORHOOD CELL : WEST WALL = %00001000
        GOSUB Put_Byte
      ENDIF
    ENDIF

  ENDIF

  ' ----------------------------------------
  nowWall.BIT4 = IsHigh  ' VISITED CELL
  isSideWall = IsLow

  ' ---------- PROCESS FRONT-REAR WALL ----------------------

  ELSEIF (nowWall.BIT4 = IsHigh AND nowWall.BIT5 = IsLow) THEN

    ' ----------------------------------------
    ' Set SOUTH WALL = %00000100
    IF (nowWall.BIT2 = IsLow) THEN

      IF (nowY = 0) THEN
        nowWall.BIT2 = IsHigh
      ENDIF

    ENDIF

    ' ----------------------------------------
    ' Set NORTH WALL = %00000001
    IF (nowWall.BIT0 = IsLow) THEN

      IF (nowY = MatrixSizeMinSatu) THEN
        nowWall.BIT0 = IsHigh
      ELSEIF (sensor4 OR sensor3) THEN
        nowWall.BIT0 = IsHigh
        ' HERE TO SET ADJACENT CELL'S WALL !!!
        tempX = nowX
        tempY = nowY + 1
        GOSUB Read_Adj_Wall
        IF (B25.BIT5 = IsLow) THEN
          B25.BIT2 = IsHigh  ' NEIGHBORHOOD CELL : SOUTH WALL = %00000100
          GOSUB Put_Byte
        ENDIF
      ENDIF

    ENDIF

    ' ----------------------------------------
    nowWall.BIT5 = IsHigh  ' VISITED CELL
    ' isCheckpoint = IsHigh  ' STAY HIGH BECAUSE WILL BE USED FOR CROSS 2 ANOTHER CELL !

  ENDIF

' ---------- ROBOT FACING EAST -----------------------------

ELSEIF (nowFaceIn = East) THEN

  ' ---------- PROCESS SIDE WALL -----------------------------
```

```
IF (nowWall.BIT5 = IsLow) THEN

  ' ----------------------------------------
  ' SET SOUTH WALL  = %00000100
  IF (nowWall.BIT2 = IsLow) THEN

    IF (nowY = 0) THEN
      nowWall.BIT2 = IsHigh
    ELSEIF (sensor2 OR sensor1) THEN
      nowWall.BIT2 = IsHigh
      ' HERE TO SET ADJACENT CELL'S WALL !!!
      tempX = nowX
      tempY = nowY - 1
      GOSUB Read_Adj_Wall
      IF (B25.BIT5 = IsLow) THEN
        B25.BIT0 = IsHigh  ' NEIGHBORHOOD CELL : NORTH WALL = %00000001
        GOSUB Put_Byte
      ENDIF
    ENDIF

  ENDIF

  ' ----------------------------------------
  ' SET NORTH WALL  = %00000001
  IF (nowWall.BIT0 = IsLow) THEN

    IF (nowY = MatrixSizeMinSatu) THEN
      nowWall.BIT0 = IsHigh
    ELSEIF (sensor6 OR sensor5) THEN
      nowWall.BIT0 = IsHigh
      ' HERE TO SET ADJACENT CELL'S WALL !!!
      tempX = nowX
      tempY = nowY + 1
      GOSUB Read_Adj_Wall
      IF (B25.BIT5 = IsLow) THEN
        B25.BIT2 = IsHigh  ' NEIGHBORHOOD CELL : SOUTH WALL = %00000100
        GOSUB Put_Byte
      ENDIF
    ENDIF

  ENDIF

  ' ----------------------------------------
  nowWall.BIT5 = IsHigh  ' VISITED CELL
  isSideWall = IsLow

' ---------- PROCESS FRONT-REAR WALL ----------------------

ELSEIF (nowWall.BIT5 = IsHigh AND nowWall.BIT4 = IsLow) THEN

  ' ----------------------------------------
  ' Set WEST WALL = %00001000
  IF (nowWall.BIT3 = IsLow) THEN

    IF (nowX = 0) THEN
      nowWall.BIT3 = IsHigh
    ENDIF

  ENDIF

  ' ----------------------------------------
  ' Set EAST WALL = %00000010
  IF (nowWall.BIT1 = IsLow) THEN

    IF (nowX = MatrixSizeMinSatu) THEN
      nowWall.BIT1 = IsHigh
    ELSEIF (sensor4 OR sensor3) THEN
      nowWall.BIT1 = IsHigh
      ' HERE TO SET ADJACENT CELL'S WALL !!!
      tempX = nowX + 1
      tempY = nowY
      GOSUB Read_Adj_Wall
      IF (B25.BIT4 = IsLow) THEN
        B25.BIT3 = IsHigh   ' NEIGHBORHOOD CELL : WEST WALL = %00001000
        GOSUB Put_Byte
      ENDIF
    ENDIF

  ENDIF

  ' ----------------------------------------
  nowWall.BIT4 = IsHigh  ' VISITED CELL
  ' isCheckpoint = IsHigh  ' STAY HIGH BECAUSE WILL BE USED FOR CROSS 2 ANOTHER CELL !

ENDIF
```

```
' ---------- ROBOT FACING SOUTH -----------------------------

ELSEIF (nowFaceIn = South) THEN

  ' ---------- PROCESS SIDE WALL -----------------------------

  IF (nowWall.BIT4 = IsLow) THEN

    ' ---------------------------------------
    ' Set WEST WALL  = %00001000
    IF (nowWall.BIT3 = IsLow) THEN

      IF (nowX = 0) THEN
        nowWall.BIT3 = IsHigh
      ELSEIF (sensor2 OR sensor1) THEN
        nowWall.BIT3 = IsHigh
        ' HERE TO SET ADJACENT CELL'S WALL !!!
        tempX = nowX - 1
        tempY = nowY
        GOSUB Read_Adj_Wall
        IF (B25.BIT4 = IsLow) THEN
          B25.BIT1 = IsHigh  ' NEIGHBORHOOD CELL : EAST WALL  = %00000010
          GOSUB Put_Byte
        ENDIF
      ENDIF

    ENDIF

    ' ---------------------------------------
    ' Set EAST WALL  = %00000010
    IF (nowWall.BIT1 = IsLow) THEN

      IF (nowX = MatrixSizeMinSatu) THEN
        nowWall.BIT1 = IsHigh
      ELSEIF (sensor6 OR sensor5) THEN
        nowWall.BIT1 = IsHigh
        ' HERE TO SET ADJACENT CELL'S WALL !!!
        tempX = nowX + 1
        tempY = nowY
        GOSUB Read_Adj_Wall
        IF (B25.BIT4 = IsLow) THEN
          B25.BIT3 = IsHigh  ' NEIGHBORHOOD CELL : WEST WALL = %00001000
          GOSUB Put_Byte
        ENDIF
      ENDIF

    ENDIF

    ' ---------------------------------------
    nowWall.BIT4 = IsHigh  ' VISITED CELL
    isSideWall = IsLow

  ' ---------- PROCESS FRONT-REAR WALL -----------------------

  ELSEIF (nowWall.BIT4 = IsHigh AND nowWall.BIT5 = IsLow) THEN

    ' ---------------------------------------
    ' Set SOUTH WALL = %00000100
    IF (nowWall.BIT2 = IsLow) THEN

      IF (nowY = 0) THEN
        nowWall.BIT2 = IsHigh
      ELSEIF (sensor4 OR sensor3) THEN
        nowWall.BIT2 = IsHigh
        ' HERE TO SET ADJACENT CELL'S WALL !!!
        tempX = nowX
        tempY = nowY - 1
        GOSUB Read_Adj_Wall
        IF (B25.BIT5 = IsLow) THEN
          B25.BIT0 = IsHigh  ' NEIGHBORHOOD CELL : NORTH WALL = %00000001
          GOSUB Put_Byte
        ENDIF
      ENDIF

    ENDIF

    ' ---------------------------------------
    ' Set NORTH WALL = %00000001
    IF (nowWall.BIT0 = IsLow) THEN

      IF (nowY = MatrixSizeMinSatu) THEN
        nowWall.BIT0 = IsHigh
      ENDIF
```

```
    ENDIF

    ' ----------------------------------------
    nowWall.BIT5 = IsHigh  ' VISITED CELL
    ' isCheckpoint = IsHigh  ' STAY HIGH BECAUSE WILL BE USED FOR CROSS 2 ANOTHER CELL !

  ENDIF

' ---------- ROBOT FACING WEST ----------------------------

ELSEIF (nowFaceIn = West) THEN

  ' ---------- PROCESS SIDE WALL ----------------------------

  IF (nowWall.BIT5 = IsLow) THEN

    ' ----------------------------------------
    ' SET SOUTH WALL  = %00000100
    IF (nowWall.BIT2 = IsLow) THEN

      IF (nowY = 0) THEN
        nowWall.BIT2 = IsHigh
      ELSEIF (sensor6 OR sensor5) THEN
        nowWall.BIT2 = IsHigh
        ' HERE TO SET ADJACENT CELL'S WALL !!!
        tempX = nowX
        tempY = nowY - 1
        GOSUB Read_Adj_Wall
        IF (B25.BIT5 = IsLow) THEN
          B25.BIT0 = IsHigh  ' NEIGHBORHOOD CELL : NORTH WALL = %00000001
          GOSUB Put_Byte
        ENDIF
      ENDIF

    ENDIF

    ' ----------------------------------------
    ' SET NORTH WALL  = %00000001
    IF (nowWall.BIT0 = IsLow) THEN

      IF (nowY = MatrixSizeMinSatu) THEN
        nowWall.BIT0 = IsHigh
      ELSEIF (sensor2 OR sensor1) THEN
        nowWall.BIT0 = IsHigh
        ' HERE TO SET ADJACENT CELL'S WALL !!!
        tempX = nowX
        tempY = nowY + 1
        GOSUB Read_Adj_Wall
        IF (B25.BIT5 = IsLow) THEN
          B25.BIT2 = IsHigh  ' NEIGHBORHOOD CELL : SOUTH WALL = %00000100
          GOSUB Put_Byte
        ENDIF
      ENDIF

    ENDIF

    ' ----------------------------------------
    nowWall.BIT5 = IsHigh  ' VISITED CELL
    isSideWall = IsLow

  ' ---------- PROCESS FRONT-REAR WALL ----------------------

  ELSEIF (nowWall.BIT5 = IsHigh AND nowWall.BIT4 = IsLow) THEN

    ' ----------------------------------------
    ' Set WEST WALL = %00001000
    IF (nowWall.BIT3 = IsLow) THEN

      IF (nowX = 0) THEN
        nowWall.BIT3 = IsHigh
      ELSEIF (sensor4 OR sensor3) THEN
        nowWall.BIT3 = IsHigh
        ' HERE TO SET ADJACENT CELL'S WALL !!!
        tempX = nowX - 1
        tempY = nowY
        GOSUB Read_Adj_Wall
        IF (B25.BIT4 = IsLow) THEN
          B25.BIT1 = IsHigh   ' NEIGHBORHOOD CELL : EAST WALL = %00000010
          GOSUB Put_Byte
        ENDIF
      ENDIF

    ENDIF

    ' ----------------------------------------
```

```
      ' Set EAST WALL = %00000010
      IF (nowWall.BIT1 = IsLow) THEN

        IF (nowX = MatrixSizeMinSatu) THEN
          nowWall.BIT1 = IsHigh
        ENDIF

      ENDIF

      ' ----------------------------------------
      nowWall.BIT4 = IsHigh  ' VISITED CELL
      ' isCheckpoint = IsHigh  ' STAY HIGH BECAUSE WILL BE USED FOR CROSS 2 ANOTHER CELL !

    ENDIF

  ENDIF

 ' ----------------------------------------
  GOSUB Write_Wall

 ' ----------------------------------------
  #IF DebugNormal #THEN
    DEBUG "nowWall = ", BIN4 nowWall.NIB1, "-", BIN4 nowWall.NIB0, CR
  #ENDIF

 ENDIF

 RETURN

' ----------------------------------------
Read_Adj_Wall:

 B24 = WallValue + ((tempX * MatrixSize) + tempY)
 GOSUB Get_Byte

 RETURN

' ----------------------------------------
Write_Wall:

 B24 = WallValue + ((nowX * MatrixSize) + nowY)
 B25 = nowWall
 GOSUB Put_Byte

 RETURN

' ----------------------------------------
Put_Byte:
 PUT B24, B25
 RETURN

' ----------------------------------------
Get_Byte:
 GET B24, B25
 RETURN
```

Kode Program Bank 2 EEPROM

```
' -----[ Initialization ]----------------------------------------------------
'
' -----[ Main Code ]-----------------------------------------------------------
'
' ----------------------------------------
Do_Process_2:

  #IF DebugLow #THEN
    DEBUG CR, "PROCESS_2:", CR
  #ENDIF

  #IF DebugNormal #THEN
    DEBUG IHEX2 ? nowCoord
  #ENDIF

  GOSUB Reset_Path_Status
  GOSUB Get_Open_Distance
  GOSUB Get_Visited_Open_Distance
  GOSUB Get_Open_Marked_Path

' ----------------------------------------
Do_Process_4:  ' PREPARE FOR NEXT MOVE
  RUN Bank_4

' -----[ Subroutines ]---------------------------------------------------------
'
' ----------------------------------------
Reset_Path_Status:

  #IF DebugLow #THEN
    DEBUG CR, "RESET_PATH:", CR
  #ENDIF

  IF (nowCoord <> 0) THEN

    B24 = OrientValue + ((nowX * MatrixSize) + nowY)
    B25 = 0
    GOSUB Put_Byte

  ENDIF

  B24 = WallValue + ((nowX * MatrixSize) + nowY)
  GOSUB Get_Byte
  B25.BIT6 = IsLow
  GOSUB Put_Byte

  RETURN

' ----------------------------------------
Get_Open_Distance:

  #IF DebugLow #THEN
    DEBUG CR, "GET_OPEN_DIST:", CR
  #ENDIF

  ' INITIALIZING
  FOR index = West TO North
    adjCoordArr(index) = EqualTo255
    adjDistArr(index) = EqualTo15
  NEXT
  numOfOpenDist = 0
  openDist = 0

  ' PROCESS FOR OPEN NEIGHBOUR AT WEST
  IF (nowX > 0) THEN
    IF (nowWall.BIT3 = IsLow) THEN
      index = West
      tempX = nowX - 1
      tempY = nowY
      GOSUB Save_Open_Distance
    ENDIF
  ENDIF

  ' PROCESS FOR OPEN NEIGHBOUR AT SOUTH
  IF (nowY > 0) THEN
    IF (nowWall.BIT2 = IsLow) THEN
      index = South
      tempX = nowX
      tempY = nowY - 1
      GOSUB Save_Open_Distance
    ENDIF
  ENDIF
```

```
  ' PROCESS FOR OPEN NEIGHBOUR AT EAST
  IF (nowX < MatrixSizeMinSatu) THEN
    IF (nowWall.BIT1 = IsLow) THEN
      index = East
      tempX = nowX + 1
      tempY = nowY
      GOSUB Save_Open_Distance
    ENDIF
  ENDIF

  ' PROCESS FOR OPEN NEIGHBOUR AT NORTH
  IF (nowY < MatrixSizeMinSatu) THEN
    IF (nowWall.BIT0 = IsLow) THEN
      index = North
      tempX = nowX
      tempY = nowY + 1
      GOSUB Save_Open_Distance
    ENDIF
  ENDIF

  ' DEBUG CODE INCLUDED
  GOSUB Get_Number_Of_Open_Distance

  RETURN

' ----------------------------------------
Save_Open_Distance:

  adjCoordArr(index) = tempCoord

  B24 = DistanceValue + ((tempX * MatrixSize) + tempY)
  GOSUB Get_Byte
  adjDistArr(index) = B25

  openDist.LOWBIT(index) = IsHigh

  RETURN

' ----------------------------------------
Get_Number_Of_Open_Distance:

  'INITIALIZING
  numOfOpenDist = 0

  FOR index = West TO North
    IF (openDist.LOWBIT(index)) THEN

      #IF DebugNormal #THEN
        DEBUG "adjCoordArr", "[", DEC index, "] = ", IHEX2 adjCoordArr(index), "; ", "adjDistArr", "[", DEC index, "] = ",
DEC adjDistArr(index), CR
      #ENDIF

      numOfOpenDist = numOfOpenDist + 1

    ENDIF
  NEXT

  #IF DebugNormal #THEN
    DEBUG "numOfOpenDist = ", DEC numOfOpenDist, "; ", "openDist = ", IBIN4 openDist, CR
  #ENDIF

  RETURN

' ----------------------------------------
Get_Visited_Open_Distance:

  #IF DebugLow #THEN
    DEBUG CR, "GET_VIS_OPEN_DIST:", CR
  #ENDIF

  FOR index = West TO North
    IF (openDist.LOWBIT(index)) THEN

      tempCoord = adjCoordArr(index)
      B24 = WallValue + ((tempX * MatrixSize) + tempY)
      GOSUB Get_Byte

      IF (B25.BIT5 = IsLow) OR (B25.BIT4 = IsLow) THEN
        openDist.LOWBIT(index) = IsLow
      ENDIF

    ENDIF
  NEXT
```

```
  ' DEBUG CODE INCLUDED
  GOSUB Get_Number_Of_Open_Distance

  RETURN

' ----------------------------------------
Get_Open_Marked_Path:

  #IF DebugLow #THEN
    DEBUG CR, "GET_OPEN_MARKED_PATH:", CR
  #ENDIF

  FOR index = West TO North
    IF (openDist.LOWBIT(index)) THEN

      tempCoord = adjCoordArr(index)

      B24 = WallValue + ((tempX * MatrixSize) + tempY)
      GOSUB Get_Byte
      IF (B25.BIT6 = IsLow) THEN
        openDist.LOWBIT(index) = IsLow
      ENDIF

    ENDIF
  NEXT

  ' DEBUG CODE INCLUDED
  GOSUB Get_Number_Of_Open_Distance

  RETURN

' ----------------------------------------
Put_Byte:
  PUT B24, B25
  RETURN

' ----------------------------------------
Get_Byte:
  GET B24, B25
  RETURN
```

Kode Program Bank 3 EEPROM

```
' -----[ Initialization ]------------------------------------------------------
'
' -----[ Main Code ]-------------------------------------------------------------
'
' ----------------------------------------
Do_Process_3:

  #IF DebugLow #THEN
    DEBUG CR, "PROCESS_3:", CR
  #ENDIF

  #IF DebugNormal #THEN
    DEBUG IHEX2 ? nowCoord
  #ENDIF

  stackPointer = 0
  tempCoord = nowCoord
  GOSUB Push_Coordinate

' ----------------------------------------
Process_Distance:

  #IF DebugLow #THEN
    DEBUG CR, "PROC_DIST:", CR
  #ENDIF

  GOSUB Pull_Coordinate
  GOSUB Read_Wall

  IF (isNoWall) THEN
    GOTO Check_Stack_Pointer
  ENDIF

  GOSUB Get_Open_Distance
  GOSUB Get_Lowest_Open_Distance
  GOSUB Check_Current_Distance

  IF (isBrokenRule) THEN
    GOSUB Push_Open_Distance

  #IF DebugNormal #THEN
  ELSE
    DEBUG "No push", CR
  #ENDIF

  ENDIF
' ----------------------------------------
Check_Stack_Pointer:

  #IF DebugLow #THEN
    DEBUG CR, "CHECK_SP:", CR
  #ENDIF

  #IF DebugNormal #THEN
    DEBUG ? stackPointer
  #ENDIF

  IF (stackPointer > 0) THEN
    GOTO Process_Distance
  ENDIF

' ----------------------------------------
Current_Coordinate:
  GOSUB Process_Current_Coordinate

' ----------------------------------------
Do_Process_4:  ' PREPARE FOR NEXT MOVE
  RUN Bank_4

' -----[ Subroutines ]-----------------------------------------------------------
'
' ----------------------------------------
Process_Current_Coordinate:

  #IF DebugLow #THEN
    DEBUG CR, "PROC_CUR_COORD:", CR
  #ENDIF

  #IF DebugNormal #THEN
    DEBUG IHEX2 ? nowCoord
  #ENDIF
```

```
  tempCoord = nowCoord
  GOSUB Read_Wall
  GOSUB Get_Open_Distance
  GOSUB Get_Lowest_Open_Distance

  RETURN

' ----------------------------------------
Push_Coordinate:

  #IF DebugLow #THEN
    DEBUG CR, "PUSH_COORD:", CR
  #ENDIF

  IF (stackPointer > 0) THEN
    FOR B23 = stackPointer-1 TO 0

      B24 = StackValue + B23
      GOSUB Get_Byte
      B24 = StackValue + (B23 + 1)
      GOSUB Put_Byte

    NEXT
  ENDIF

  B24 = StackValue
  B25 = tempCoord
  GOSUB Put_Byte

  stackPointer = stackPointer + 1

  #IF DebugNormal #THEN
    DEBUG "Push = ", IHEX2 tempCoord, CR
    GOSUB Show_Stack
  #ENDIF

  RETURN

' ----------------------------------------
Pull_Coordinate:

  #IF DebugLow #THEN
    DEBUG CR, "PULL_COORD:", CR
  #ENDIF

  B24 = StackValue
  GOSUB Get_Byte
  tempCoord = B25

  IF (stackPointer > 0) THEN
    FOR B23 = 0 TO stackPointer-1

      B24 = StackValue + (B23 + 1)
      GOSUB Get_Byte
      B24 = StackValue + B23
      GOSUB Put_Byte

    NEXT
  ENDIF

  stackPointer = stackPointer - 1

  #IF DebugNormal #THEN
    DEBUG "Pull = ", IHEX2 tempCoord, CR
    GOSUB Show_Stack
  #ENDIF

  RETURN

' ----------------------------------------
Show_Stack:

  #IF DebugNormal #THEN
    DEBUG ? stackPointer
    IF (stackPointer > 0) THEN
      FOR B23 = 0 TO stackPointer-1
        B24 = StackValue + B23
        GOSUB Get_Byte
        DEBUG "[", DEC B23, "] = ", IHEX2 B25, CR
      NEXT
    ENDIF
  #ENDIF

  RETURN
```

```
' --------------------------------------
Read_Wall:

  #IF DebugLow #THEN
    DEBUG CR, "READ_WALL:", CR
  #ENDIF

  ' INITIALIZING
  isNoWall = 0

  B24 = WallValue + ((tempX * MatrixSize) + tempY)
  GOSUB Get_Byte
  B23 = B25

  IF (B23.BIT5 = IsLow) OR (B23.BIT4 = IsLow) THEN
    isNoWall = 1
  ENDIF

  #IF DebugNormal #THEN
    DEBUG "Wall = ", BIN4 B23.NIB1, "-", BIN4 B23.NIB0, CR
    DEBUG ? isNoWall
  #ENDIF

  RETURN

' --------------------------------------
Get_Open_Distance:

  #IF DebugLow #THEN
    DEBUG CR, "GET_OPEN_DIST:", CR
  #ENDIF

  ' INITIALIZING
  FOR index = West TO North
    adjCoordArr(index) = EqualTo255
    adjDistArr(index) = EqualTo15
  NEXT
  numOfOpenDist = 0
  openDist = 0

  ' PROCESS FOR OPEN NEIGHBOUR AT WEST
  IF (tempX > 0) THEN
    IF (B23.BIT3 = IsLow) THEN
      index = West
      B22.NIB1 = tempX - 1
      B22.NIB0 = tempY
      GOSUB Save_Open_Distance
    ENDIF
  ENDIF

  ' PROCESS FOR OPEN NEIGHBOUR AT SOUTH
  IF (tempY > 0) THEN
    IF (B23.BIT2 = IsLow) THEN
      index = South
      B22.NIB1 = tempX
      B22.NIB0 = tempY - 1
      GOSUB Save_Open_Distance
    ENDIF
  ENDIF

  ' PROCESS FOR OPEN NEIGHBOUR AT EAST
  IF (tempX < MatrixSizeMinSatu) THEN
    IF (B23.BIT1 = IsLow) THEN
      index = East
      B22.NIB1 = tempX + 1
      B22.NIB0 = tempY
      GOSUB Save_Open_Distance
    ENDIF
  ENDIF

  ' PROCESS FOR OPEN NEIGHBOUR AT NORTH
  IF (tempY < MatrixSizeMinSatu) THEN
    IF (B23.BIT0 = IsLow) THEN
      index = North
      B22.NIB1 = tempX
      B22.NIB0 = tempY + 1
      GOSUB Save_Open_Distance
    ENDIF
  ENDIF

  ' DEBUG CODE INCLUDED
  GOSUB Get_Number_Of_Open_Distance

  RETURN
```

```
' ---------------------------------------
Save_Open_Distance:

  adjCoordArr(index) = B22

  B24 = DistanceValue + ((B22.NIB1 * MatrixSize) + B22.NIB0)
  GOSUB Get_Byte
  adjDistArr(index) = B25

  openDist.LOWBIT(index) = IsHigh

  RETURN

' ---------------------------------------
Get_Number_Of_Open_Distance:

  'INITIALIZING
  numOfOpenDist = 0

  FOR index = West TO North
    IF (openDist.LOWBIT(index)) THEN

      #IF DebugNormal #THEN
        DEBUG "adjCoord", "[", DEC index, "] = ", IHEX2 adjCoordArr(index), "; ", "adjDist", "[", DEC index, "] = ", DEC
adjDistArr(index), CR
      #ENDIF

      numOfOpenDist = numOfOpenDist + 1

    ENDIF
  NEXT

  #IF DebugNormal #THEN
    DEBUG "numOfOpenDist = ", DEC numOfOpenDist, "; ", "openDist = ", IBIN4 openDist, CR
  #ENDIF

  RETURN

' ---------------------------------------
Get_Lowest_Open_Distance:

  #IF DebugLow #THEN
    DEBUG CR, "GET_LOWEST:", CR
  #ENDIF

  IF (numOfOpenDist > OneOpenNeighbour) THEN
    IF (numOfOpenDist = TwoOpenNeighbours) THEN
      GOSUB Sort_2_Distances
    ELSEIF (numOfOpenDist = ThreeOpenNeighbours) THEN
      GOSUB Sort_3_Distances
    ENDIF

    ' DEBUG CODE INCLUDED
    GOSUB Get_Number_Of_Open_Distance

  #IF DebugNormal #THEN
  ELSE
      DEBUG "Only 1", CR
  #ENDIF

  ENDIF

  RETURN

' ---------------------------------------
Sort_2_Distances:

  '#IF DebugLow #THEN
  '  DEBUG CR, "SORT_2:", CR
  '#ENDIF

  SELECT openDist
    CASE %0011
      B25 = East
      B24 = North
    CASE %0101
      B25 = South
      B24 = North
    CASE %1001
      B25 = West
      B24 = North
    CASE %0110
      B25 = South
      B24 = East
```

```
    CASE %1010
      B25 = West
      B24 = East
    CASE %1100
      B25 = West
      B24 = South
  ENDSELECT

  GOSUB Set_Lowest_Of_2_Distances

  RETURN

' ---------------------------------------
Set_Lowest_Of_2_Distances:

  IF adjDistArr(B25) < adjDistArr(B24) THEN
    'NO NEED BECAUSE ALREADY HIGH FROM PREVIOUS SETTING
    'openDist.LOWBIT(B25) = IsHigh
    openDist.LOWBIT(B24) = IsLow

  ELSEIF adjDistArr(B25) > adjDistArr(B24) THEN
    openDist.LOWBIT(B25) = IsLow
    'NO NEED BECAUSE ALREADY HIGH FROM PREVIOUS SETTING
    'openDist.LOWBIT(B24) = IsHigh

  'NO NEED BECAUSE ALREADY HIGH FROM PREVIOUS SETTING
  'ELSE
  '  openDist.LOWBIT(B25) = IsHigh
  '  openDist.LOWBIT(B24) = IsHigh

  ENDIF

  RETURN

' ---------------------------------------
Sort_3_Distances:

  '#IF DebugLow #THEN
  '  DEBUG CR, "SORT_3:", CR
  '#ENDIF

  SELECT openDist
    CASE %0111
      B25 = South
      B24 = East
      B23 = North
    CASE %1011
      B25 = West
      B24 = East
      B23 = North
    CASE %1101
      B25 = West
      B24 = South
      B23 = North
    CASE %1110
      B25 = West
      B24 = South
      B23 = East
  ENDSELECT

  GOSUB Set_Lowest_Of_3_Distances

  RETURN

' ---------------------------------------
Set_Lowest_Of_3_Distances:

  IF adjDistArr(B25) < adjDistArr(B24) THEN

    IF adjDistArr(B25) < adjDistArr(B23) THEN
      'NO NEED BECAUSE ALREADY HIGH FROM PREVIOUS SETTING
      'openDist.LOWBIT(B25) = IsHigh
      openDist.LOWBIT(B24) = IsLow
      openDist.LOWBIT(B23) = IsLow

    ELSEIF adjDistArr(B25) > adjDistArr(B23) THEN
      openDist.LOWBIT(B25) = IsLow
      openDist.LOWBIT(B24) = IsLow
      'NO NEED BECAUSE ALREADY HIGH FROM PREVIOUS SETTING
      'openDist.LOWBIT(B23) = IsHigh

    ELSE
      'NO NEED BECAUSE ALREADY HIGH FROM PREVIOUS SETTING
      'openDist.LOWBIT(B25) = IsHigh
      openDist.LOWBIT(B24) = IsLow
```

```
    'NO NEED BECAUSE ALREADY HIGH FROM PREVIOUS SETTING
    'openDist.LOWBIT(B23) = IsHigh

   ENDIF

  ELSEIF adjDistArr(B25) > adjDistArr(B24) THEN

   IF adjDistArr(B24) < adjDistArr(B23) THEN
    openDist.LOWBIT(B25) = IsLow
    'NO NEED BECAUSE ALREADY HIGH FROM PREVIOUS SETTING
    'openDist.LOWBIT(B24) = IsHigh
    openDist.LOWBIT(B23) = IsLow

   ELSEIF adjDistArr(B24) > adjDistArr(B23) THEN
    openDist.LOWBIT(B25) = IsLow
    openDist.LOWBIT(B24) = IsLow
    'NO NEED BECAUSE ALREADY HIGH FROM PREVIOUS SETTING
    'openDist.LOWBIT(B23) = IsHigh

   ELSE
    openDist.LOWBIT(B25) = IsLow
    'NO NEED BECAUSE ALREADY HIGH FROM PREVIOUS SETTING
    'openDist.LOWBIT(B24) = IsHigh
    'openDist.LOWBIT(B23) = IsHigh

   ENDIF

  ELSE

   IF adjDistArr(B25) < adjDistArr(B23) THEN
    'NO NEED BECAUSE ALREADY HIGH FROM PREVIOUS SETTING
    'openDist.LOWBIT(B25) = IsHigh
    'openDist.LOWBIT(B24) = IsHigh
    openDist.LOWBIT(B23) = IsLow

   ELSEIF adjDistArr(B25) > adjDistArr(B23) THEN
    openDist.LOWBIT(B25) = IsLow
    openDist.LOWBIT(B24) = IsLow
    'NO NEED BECAUSE ALREADY HIGH FROM PREVIOUS SETTING
    'openDist.LOWBIT(B23) = IsHigh

   'NO NEED BECAUSE ALREADY HIGH FROM PREVIOUS SETTING
   'ELSE
   '  openDist.LOWBIT(B25) = IsHigh
   '  openDist.LOWBIT(B24) = IsHigh
   '  openDist.LOWBIT(B23) = IsHigh

   ENDIF

  ENDIF

  RETURN

' ----------------------------------------
Check_Current_Distance:

  #IF DebugLow #THEN
   DEBUG CR, "CHECK_CUR_DIST:", CR
  #ENDIF

  ' INITIALIZING
  isBrokenRule = IsLow

  B24 = DistanceValue + ((tempX * MatrixSize) + tempY)
  GOSUB Get_Byte

  #IF DebugNormal #THEN
   DEBUG "tempDist = ", DEC B25, CR
  #ENDIF

  FOR index = West TO North
   IF (openDist.LOWBIT(index)) THEN
    IF (B25 < adjDistArr(index)) THEN

     isBrokenRule = IsHigh
     B25 = adjDistArr(index) + 1
     'B24 = DistanceValue + ((tempX * MatrixSize) + tempY)
     GOSUB Put_Byte

    ENDIF
    EXIT
   ENDIF
  NEXT

  #IF DebugNormal #THEN
```

```
    DEBUG "isBrokenRule = ", BIN isBrokenRule, "; ", "tempDist = ", DEC B25, CR
  #ENDIF

  RETURN

' ----------------------------------------
Push_Open_Distance:

  #IF DebugLow #THEN
    DEBUG CR, "PUSH_OPEN_DIST:", CR
  #ENDIF

  'IF (isBrokenRule) THEN

    FOR index = West TO North
      IF (adjCoordArr(index) < EqualTo255) THEN

        #IF DebugNormal #THEN
          DEBUG CR, "adjCoord", "[", DEC index, "] = ", IHEX2 adjCoordArr(index), CR
        #ENDIF

        tempCoord = adjCoordArr(index)
        ' THAT'S WHY Push_Coordinate DO NOT USE index (REPLACED BY B23)
        GOSUB Push_Coordinate

      ENDIF
    NEXT

  '#IF DebugNormal #THEN
  'ELSE
  '    DEBUG "No push", CR
  '#ENDIF
  '
  'ENDIF

  RETURN

' ----------------------------------------
Put_Byte:
  PUT B24, B25
  RETURN

' ----------------------------------------
Get_Byte:
  GET B24, B25
  RETURN
```

Kode Program Bank 4 EEPROM

```
' -----[ Initialization ]------------------------------------------------------
'
' -----[ Main Code ]-------------------------------------------------------------
'
' ----------------------------------------
Do_Process_4:

  #IF DebugLow #THEN
    DEBUG CR, "PROCESS_4:", CR
  #ENDIF

  #IF DebugNormal #THEN
    DEBUG IHEX2 ? nowCoord
  #ENDIF

  GOSUB Process_Destination

' ----------------------------------------
Do_Output_1:
  RUN Bank_5

' -----[ Subroutines ]-----------------------------------------------------------
'
' ----------------------------------------
Process_Destination:

  #IF DebugLow #THEN
    DEBUG CR, "PROC_DEST:", CR
  #ENDIF

  IF (nowFaceIn = North) THEN

    SELECT openDist

      CASE %0001, %0011, %0101, %1001, %0111, %1011, %1101
        index = North
        GOSUB Process_New_Coordinate
        GOSUB Process_New_Orientation
        moveType = GoStraight

      CASE %0010, %0110, %1010, %1110
        index = East
        GOSUB Process_New_Coordinate
        GOSUB Process_New_Orientation
        moveType = RotateRight90

      CASE %0100
        index = South
        GOSUB Process_New_Coordinate
        GOSUB Process_New_Orientation
        moveType = RotateRight180

      CASE %1000, %1100
        index = West
        GOSUB Process_New_Coordinate
        GOSUB Process_New_Orientation
        moveType = RotateLeft90

      'CASE %0011 ' GO EAST OR NORTH
      'CASE %0101 ' GO SOUTH OR NORTH
      'CASE %1001 ' GO WEST OR NORTH
      'CASE %0110 ' GO SOUTH OR EAST
      'CASE %1010 ' GO WEST OR EAST *
      'CASE %1100 ' GO WEST OR SOUTH
      'CASE %0111 ' GO SOUTH, EAST, OR NORTH
      'CASE %1011 ' GO WEST, EAST, OR NORTH
      'CASE %1101 ' GO WEST, SOUTH, OR NORTH
      'CASE %1110 ' GO WEST, SOUTH, OR EAST * (ABSOLUTELY NOT EAST)

    ENDSELECT

  ELSEIF (nowFaceIn = East) THEN

    SELECT openDist

      CASE %0001, %1001
        index = North
        GOSUB Process_New_Coordinate
        GOSUB Process_New_Orientation
        moveType = RotateLeft90
```

```
    CASE %0010, %0011, %0110, %1010, %0111, %1011, %1110
      index = East
      GOSUB Process_New_Coordinate
      GOSUB Process_New_Orientation
      moveType = GoStraight

    CASE %0100, %0101, %1100, %1101
      index = South
      GOSUB Process_New_Coordinate
      GOSUB Process_New_Orientation
      moveType = RotateRight90

    CASE %1000
      index = West
      GOSUB Process_New_Coordinate
      GOSUB Process_New_Orientation
      moveType = RotateRight180

    'CASE %0011 ' GO EAST OR NORTH
    'CASE %0101 ' GO SOUTH OR NORTH *
    'CASE %1001 ' GO WEST OR NORTH
    'CASE %0110 ' GO SOUTH OR EAST
    'CASE %1010 ' GO WEST OR EAST
    'CASE %1100 ' GO WEST OR SOUTH
    'CASE %0111 ' GO SOUTH, EAST, OR NORTH
    'CASE %1011 ' GO WEST, EAST, OR NORTH
    'CASE %1101 ' GO WEST, SOUTH, OR NORTH *
    'CASE %1110 ' GO WEST, SOUTH, OR EAST

  ENDSELECT

ELSEIF (nowFaceIn = South) THEN

  SELECT openDist

    CASE %0001
      index = North
      GOSUB Process_New_Coordinate
      GOSUB Process_New_Orientation
      moveType = RotateRight180

    CASE %0010, %0011
      index = East
      GOSUB Process_New_Coordinate
      GOSUB Process_New_Orientation
      moveType = RotateLeft90

    CASE %0100, %0101, %0110, %1100, %0111, %1101, %1110
      index = South
      GOSUB Process_New_Coordinate
      GOSUB Process_New_Orientation
      moveType = GoStraight

    CASE %1000, %1001, %1010, %1011
      index = West
      GOSUB Process_New_Coordinate
      GOSUB Process_New_Orientation
      moveType = RotateRight90

    'CASE %0011 ' GO EAST OR NORTH
    'CASE %0101 ' GO SOUTH OR NORTH
    'CASE %1001 ' GO WEST OR NORTH
    'CASE %0110 ' GO SOUTH OR EAST
    'CASE %1010 ' GO WEST OR EAST *
    'CASE %1100 ' GO WEST OR SOUTH
    'CASE %0111 ' GO SOUTH, EAST, OR NORTH
    'CASE %1011 ' GO WEST, EAST, OR NORTH *
    'CASE %1101 ' GO WEST, SOUTH, OR NORTH
    'CASE %1110 ' GO WEST, SOUTH, OR EAST

  ENDSELECT

ELSEIF (nowFaceIn = West) THEN

  SELECT openDist

    CASE %0001, %0011, %0101, %0111
      index = North
      GOSUB Process_New_Coordinate
      GOSUB Process_New_Orientation
      moveType = RotateRight90

    CASE %0010
      index = East
      GOSUB Process_New_Coordinate
```

```
      GOSUB Process_New_Orientation
      moveType = RotateRight180

    CASE %0100, %0110
      index = South
      GOSUB Process_New_Coordinate
      GOSUB Process_New_Orientation
      moveType = RotateLeft90

    CASE %1000, %1001, %1010, %1100, %1011, %1101, %1110
      index = West
      GOSUB Process_New_Coordinate
      GOSUB Process_New_Orientation
      moveType = GoStraight

    'CASE %0011 ' GO EAST OR NORTH
    'CASE %0101 ' GO SOUTH OR NORTH *
    'CASE %1001 ' GO WEST OR NORTH
    'CASE %0110 ' GO SOUTH OR EAST
    'CASE %1010 ' GO WEST OR EAST
    'CASE %1100 ' GO WEST OR SOUTH
    'CASE %0111 ' GO SOUTH, EAST, OR NORTH *
    'CASE %1011 ' GO WEST, EAST, OR NORTH
    'CASE %1101 ' GO WEST, SOUTH, OR NORTH
    'CASE %1110 ' GO WEST, SOUTH, OR EAST

  ENDSELECT

 ENDIF

 #IF DebugNormal #THEN
  DEBUG CR, "openDist = ", IBIN4 openDist, "; ","index = ", IBIN4 index, "; ","moveType = ", IBIN4 moveType, CR
 #ENDIF

 RETURN

' ---------------------------------------
Process_New_Coordinate:

 #IF DebugLow #THEN
  DEBUG CR, "PROC_NEW_COORD:", CR
 #ENDIF

 SELECT index
  CASE North
   tempX = nowX
   tempY = nowY + 1
  CASE East
   tempX = nowX + 1
   tempY = nowY
  CASE South
   tempX = nowX
   tempY = nowY - 1
  CASE West
   tempX = nowX - 1
   tempY = nowY
 ENDSELECT

 #IF DebugNormal #THEN
  DEBUG "nowCoord = ", IHEX2 nowCoord, "; ","tempCoord = ", IHEX2 tempCoord, CR
 #ENDIF

 RETURN

' ---------------------------------------
Process_New_Orientation:

 #IF DebugLow #THEN
  DEBUG CR, "PROC_NEW_ORIENT:", CR
 #ENDIF

 SELECT index
  CASE North
   tempFaceIn = North
  CASE East
   tempFaceIn = East
  CASE South
   tempFaceIn = South
  CASE West
   tempFaceIn = West
 ENDSELECT

 GOSUB Process_Destination_Orientation

 GOSUB Update_Orientation
```

```
  #IF DebugNormal #THEN
    DEBUG "nowFaceIn = ", IBIN4 nowFaceIn, "; ","tempFaceIn = ", IBIN4 tempFaceIn, CR
  #ENDIF

  RETURN

' ----------------------------------------
Process_Destination_Orientation:

  #IF DebugLow #THEN
    DEBUG CR, "PROC_DES_ORIENT:", CR
  #ENDIF

  IF (isFinish = IsLow) THEN

    B24 = DistanceValue + ((tempX * MatrixSize) + tempY)
    GOSUB Get_Byte

    #IF DebugNormal #THEN
      DEBUG "tempDist = ", DEC B25, CR
    #ENDIF

    IF (B25 = 0) THEN

      #IF DebugLow #THEN
        DEBUG "Yes", CR
      #ENDIF

      ' CAN NOT USE ROUTINE BECAUSE OF STRANGE BEHAVIOUR
      'GOSUB Update_Destination_Orientation

      B23 = 0

      B24 = OrientValue + ((tempX * MatrixSize) + tempY)
      GOSUB Get_Byte
      B23 = B25

      SELECT tempFaceIn
        CASE North, South
          B23.BIT0 = IsHigh
          B23.BIT2 = IsHigh
        CASE East, West
          B23.BIT1 = IsHigh
          B23.BIT3 = IsHigh
      ENDSELECT

      B25 = B23
      GOSUB Put_Byte

    ENDIF

  ENDIF

  RETURN

' ----------------------------------------
Update_Orientation:

  #IF DebugLow #THEN
    DEBUG CR, "UPDATE_ORIENT:", CR
  #ENDIF

  IF (isFinish = IsLow) THEN

    #IF DebugLow #THEN
      DEBUG "Yes", CR
    #ENDIF

    B24 = OrientValue + ((nowX * MatrixSize) + nowY)
    GOSUB Get_Byte
    nowOrient = B25

    SELECT tempFaceIn
      CASE North

        IF (nowOrient.BIT0 = IsLow) THEN
          IF (nowOrient.BIT6) THEN
            nowOrient.BIT6 = IsLow
          ELSEIF (nowOrient.BIT2) THEN
            nowOrient.BIT2 = IsLow
          ELSE
            nowOrient.BIT0 = IsHigh
          ENDIF
        ELSEIF (nowOrient.BIT4 = IsLow) THEN
```

```
      IF (nowOrient.BIT6) THEN
        nowOrient.BIT6 = IsLow
      ELSEIF (nowOrient.BIT2) THEN
        nowOrient.BIT2 = IsLow
      ELSE
        nowOrient.BIT4 = IsHigh
      ENDIF
    ENDIF

  CASE East

    IF (nowOrient.BIT1 = IsLow) THEN
      IF (nowOrient.BIT7) THEN
        nowOrient.BIT7 = IsLow
      ELSEIF (nowOrient.BIT3) THEN
        nowOrient.BIT3 = IsLow
      ELSE
        nowOrient.BIT1 = IsHigh
      ENDIF
    ELSEIF (nowOrient.BIT5 = IsLow) THEN
      IF (nowOrient.BIT7) THEN
        nowOrient.BIT7 = IsLow
      ELSEIF (nowOrient.BIT3) THEN
        nowOrient.BIT3 = IsLow
      ELSE
        nowOrient.BIT5 = IsHigh
      ENDIF
    ENDIF

  CASE South

    IF (nowOrient.BIT2 = IsLow) THEN
      IF (nowOrient.BIT4) THEN
        nowOrient.BIT4 = IsLow
      ELSEIF (nowOrient.BIT0) THEN
        nowOrient.BIT0 = IsLow
      ELSE
        nowOrient.BIT2 = IsHigh
      ENDIF
    ELSEIF (nowOrient.BIT6 = IsLow) THEN
      IF (nowOrient.BIT4) THEN
        nowOrient.BIT4 = IsLow
      ELSEIF (nowOrient.BIT0) THEN
        nowOrient.BIT0 = IsLow
      ELSE
        nowOrient.BIT6 = IsHigh
      ENDIF
    ENDIF

  CASE West

    IF (nowOrient.BIT3 = IsLow) THEN
      IF (nowOrient.BIT5) THEN
        nowOrient.BIT5 = IsLow
      ELSEIF (nowOrient.BIT1) THEN
        nowOrient.BIT1 = IsLow
      ELSE
        nowOrient.BIT3 = IsHigh
      ENDIF
    ELSEIF (nowOrient.BIT7 = IsLow) THEN
      IF (nowOrient.BIT5) THEN
        nowOrient.BIT5 = IsLow
      ELSEIF (nowOrient.BIT1) THEN
        nowOrient.BIT1 = IsLow
      ELSE
        nowOrient.BIT7 = IsHigh
      ENDIF
    ENDIF

  ENDSELECT

  B25 = nowOrient
  GOSUB Put_Byte

ENDIF

#IF DebugNormal #THEN
  DEBUG "isFinish = ", BIN isFinish, CR, CR
#ENDIF

RETURN

' ----------------------------------------
Put_Byte:
  PUT B24, B25
```

```
  RETURN

' ----------------------------------------
Get_Byte:
  GET B24, B25
  RETURN

'' ----------------------------------------
'Put_Word:
'  PUT B20, Word W9
'  RETURN

'' ----------------------------------------
'Get_Word:
'  GET B20, Word W9
'  RETURN

'' ----------------------------------------
'Update_Destination_Orientation:
'
'    ' INITIALIZING
'    B23 = 0
'
'    B24 = OrientValue + ((tempX * MatrixSize) + tempY)
'    GOSUB Get_Byte
'    B23 = B25
'
'    SELECT tempFaceIn
'      CASE North, South
'        B23.BIT0 = IsHigh
'        B23.BIT2 = IsHigh
'      CASE East, West
'        B23.BIT1 = IsHigh
'        B23.BIT3 = IsHigh
'    ENDSELECT
'
'    B25 = B23
'    GOSUB Put_Byte
'
'  RETURN
```

Kode Program Bank 5 EEPROM

```
' -----[ Initialization ]----------------------------------------------------
'
' -----[ Main Code ]-----------------------------------------------------------
'
' ----------------------------------------
Do_Output_1:

  #IF DebugLow #THEN
    DEBUG CR, "OUTPUT_1:", CR
  #ENDIF

  #IF DebugNormal #THEN
    DEBUG IHEX2 ? nowCoord
  #ENDIF

  GOSUB Drive_Motor

' ----------------------------------------
#IF DebugNormal #THEN
Do_Output_2:
  RUN Bank_6
#ENDIF

' ----------------------------------------
Do_Input:
  task = TskDoInput  ' TskDoInput = 1
  RUN Bank_0

' -----[ Subroutines ]---------------------------------------------------------
'
' ----------------------------------------
Drive_Motor:

  #IF DebugLow #THEN
    DEBUG CR, "DRIVE_MOTOR:", CR
  #ENDIF

  #IF DebugNormal #THEN
    DEBUG "isCheckpoint = ", BIN isCheckpoint, "; "
  #ENDIF

  IF (isCheckpoint) THEN

    #IF DebugNormal #THEN
      DEBUG IBIN4 ? moveType
    #ENDIF

    SELECT moveType
      CASE GoStraight
        GOSUB Go_Straight
      CASE RotateRight90
        GOSUB Go_Rotate_Right_90
        GOSUB Go_Straight
      CASE RotateLeft90
        GOSUB Go_Rotate_Left_90
        GOSUB Go_Straight
      CASE RotateRight180
        GOSUB Go_Rotate_Right_180
        GOSUB Go_Straight
    ENDSELECT

    GOSUB Scan_Side_Wall

  ELSE

    #IF DebugNormal #THEN
      DEBUG "wallSensors[65-21] = ", BIN sensor6, BIN sensor5, "-", BIN sensor2, BIN sensor1, CR
    #ENDIF

    IF (sensor6 = isHigh AND sensor5 = isHigh) OR (sensor2 = isHigh AND sensor1 = isHigh) THEN
      GOSUB Go_Forward
    ELSEIF (sensor6 = isLow AND sensor5 = isHigh) OR (sensor2 = isLow AND sensor1 = isHigh) THEN
      GOSUB Pivot_Right
    ELSEIF (sensor6 = isHigh AND sensor5 = isLow) OR (sensor2 = isHigh AND sensor1 = isLow) THEN
      GOSUB Pivot_Left
    ELSE
      GOSUB Stop_Motor
    ENDIF

  ENDIF

  RETURN
```

```
' ----------------------------------------
Go_Straight:
  GOSUB Set_Go_Straight_Duration
  FOR B21 = 1 TO Loop4GoStraight
    GOSUB Pulsout_Motor
    PAUSE Pause4GoStraight
  NEXT
  RETURN

' ----------------------------------------
Set_Go_Straight_Duration:
  W12 = StopMotor + GoStraightFactor
  W11 = StopMotor - GoStraightFactor
  RETURN

' ----------------------------------------
Go_Rotate_Right_90:
  GOSUB Set_Go_Rotate_Right_90_Duration
  FOR B21 = 1 TO Loop4Rotate90
    GOSUB Pulsout_Motor
    PAUSE Pause4Rotate90
  NEXT
  RETURN

' ----------------------------------------
Set_Go_Rotate_Right_90_Duration:
  W12 = StopMotor + RotateFactor
  W11 = StopMotor + RotateFactor
  RETURN

' ----------------------------------------
Go_Rotate_Left_90:
  GOSUB Set_Go_Rotate_Left_90_Duration
  FOR B21 = 1 TO Loop4Rotate90
    GOSUB Pulsout_Motor
    PAUSE Pause4Rotate90
  NEXT
  RETURN

' ----------------------------------------
Set_Go_Rotate_Left_90_Duration:
  W12 = StopMotor - RotateFactor
  W11 = StopMotor - RotateFactor
  RETURN

' ----------------------------------------
Go_Rotate_Right_180:
  GOSUB Set_Go_Rotate_Right_90_Duration
  FOR B21 = 1 TO Loop4Rotate180
    GOSUB Pulsout_Motor
    PAUSE Pause4Rotate180
  NEXT
  RETURN

' ----------------------------------------
Scan_Side_Wall:

  #IF DebugLow #THEN
    DEBUG CR, "SCAN_SIDE_WALL:", CR
  #ENDIF

  ' INITIALIZING
  scanResult = IsLow

  GOSUB Read_Wall_Sensors

  IF NOT (sensor6 OR sensor5 OR sensor2 OR sensor1) THEN

    IF NOT (sensor2 OR sensor1) THEN

      #IF DebugNormal #THEN
        DEBUG "Right wing", CR
        DEBUG ? sensor2
        DEBUG ? sensor1
      #ENDIF

      IF (sensor0 OR sensor8) THEN

        #IF DebugNormal #THEN
          DEBUG "Scan right", CR
          DEBUG ? sensor8
          DEBUG ? sensor0
        #ENDIF

        ' CAN NOT USE ROUTINE BECAUSE OF STRANGE BEHAVIOUR
```

```
    'GOSUB Pivot_Right_Scanning

    GOSUB Set_Pivot_Right_Duration
    FOR B21 = 1 TO Loop4PivotRightScan
      GOSUB Pulsout_Motor
      PAUSE Pause4Scan
      GOSUB Read_Wall_Sensors
      IF (sensor2 OR sensor1) THEN
        scanResult = IsHigh
        EXIT
      ENDIF
    NEXT

  ELSEIF (sensor9) THEN

    #IF DebugNormal #THEN
      DEBUG "Scan left", CR
      DEBUG ? sensor9
    #ENDIF

    ' CAN NOT USE ROUTINE BECAUSE OF STRANGE BEHAVIOUR
    'GOSUB Pivot_Left_Scanning

    GOSUB Set_Pivot_Left_Duration
    FOR B21 = 1 TO Loop4PivotLeftScan
      GOSUB Pulsout_Motor
      PAUSE Pause4Scan
      GOSUB Read_Wall_Sensors
      IF (sensor2 OR sensor1) THEN
        scanResult = IsHigh
        EXIT
      ENDIF
    NEXT

  ENDIF

ENDIF

IF (scanResult = IsLow) AND (NOT (sensor6 OR sensor5)) THEN

  #IF DebugNormal #THEN
    DEBUG "Left wing", CR
    DEBUG ? scanResult
    DEBUG ? sensor6
    DEBUG ? sensor5
  #ENDIF

  IF (sensor7 OR sensor11) THEN

    #IF DebugNormal #THEN
      DEBUG "Scan left", CR
      DEBUG ? sensor7
      DEBUG ? sensor11
    #ENDIF

    ' CAN NOT USE ROUTINE BECAUSE OF STRANGE BEHAVIOUR
    'GOSUB Pivot_Left_Scanning

    GOSUB Set_Pivot_Left_Duration
    FOR B21 = 1 TO Loop4PivotLeftScan
      GOSUB Pulsout_Motor
      PAUSE Pause4Scan
      GOSUB Read_Wall_Sensors
      IF (sensor6 OR sensor5 OR sensor2 OR sensor1) THEN
        scanResult = IsHigh
        EXIT
      ENDIF
    NEXT

  ELSEIF (sensor10) THEN

    #IF DebugNormal #THEN
      DEBUG "Scan right", CR
      DEBUG ? sensor10
    #ENDIF

    ' CAN NOT USE ROUTINE BECAUSE OF STRANGE BEHAVIOUR
    'GOSUB Pivot_Right_Scanning

    GOSUB Set_Pivot_Right_Duration
    FOR B21 = 1 TO Loop4PivotRightScan
      GOSUB Pulsout_Motor
      PAUSE Pause4Scan
      GOSUB Read_Wall_Sensors
      IF (sensor6 OR sensor5) THEN
```

```
          scanResult = IsHigh
          EXIT
        ENDIF
      NEXT

    ENDIF

  ENDIF

  IF (scanResult = IsLow) THEN

    ' CAN NOT USE ROUTINE BECAUSE OF STRANGE BEHAVIOUR
    'GOSUB Pivot_Left_Scanning

    GOSUB Set_Pivot_Left_Duration
    FOR B21 = 1 TO Loop4PivotLeftScan
      GOSUB Pulsout_Motor
      PAUSE Pause4Scan
      GOSUB Read_Wall_Sensors
      IF (sensor6 OR sensor5 OR sensor2 OR sensor1) THEN
        scanResult = IsHigh
        EXIT
      ENDIF
    NEXT

    IF (scanResult = IsLow) THEN

      ' CAN NOT USE ROUTINE BECAUSE OF STRANGE BEHAVIOUR
      'GOSUB Pivot_Right_Scanning

      GOSUB Set_Pivot_Right_Duration
      FOR B21 = 1 TO Loop4PivotRightScan
        GOSUB Pulsout_Motor
        PAUSE Pause4Scan
        GOSUB Read_Wall_Sensors
        IF (sensor6 OR sensor5 OR sensor2 OR sensor1) THEN
          scanResult = IsHigh
          EXIT
        ENDIF
      NEXT

    ENDIF

    ENDIF

  #IF DebugNormal #THEN
  ELSE
    DEBUG "No need", CR
  #ENDIF

  ENDIF

  RETURN

' ----------------------------------------
Read_Wall_Sensors:

  #IF DebugLow #THEN
    DEBUG CR, "READ_WALL_SENSORS:", CR
  #ENDIF

  B20 = SensorThres
  GOSUB Get_Word

  HIGH OuterRightSensor      ' Turn on sensor
  GOSUB Change_IO_Direction
  sensor0 = SensorInput      ' Snapshot of sensor signal states
  LOW OuterRightSensor       ' Turn off sensor

  HIGH FarMiddleRightSensor
  GOSUB Change_IO_Direction
  sensor1 = SensorInput
  LOW FarMiddleRightSensor

  HIGH NearMiddleRightSensor
  GOSUB Change_IO_Direction
  sensor2 = SensorInput
  LOW NearMiddleRightSensor

  'HIGH InnerRightSensor
  'GOSUB Change_IO_Direction
  'sensor3 = SensorInput
  'LOW InnerRightSensor
```

```
  'HIGH InnerLeftSensor
  'GOSUB Change_IO_Direction
  'sensor4 = SensorInput
  'LOW InnerLeftSensor

  HIGH NearMiddleLeftSensor
  GOSUB Change_IO_Direction
  sensor5 = SensorInput
  LOW NearMiddleLeftSensor

  HIGH FarMiddleLeftSensor
  GOSUB Change_IO_Direction
  sensor6 = SensorInput
  LOW FarMiddleLeftSensor

  HIGH OuterLeftSensor
  GOSUB Change_IO_Direction
  sensor7 = SensorInput
  LOW OuterLeftSensor

  HIGH OuterRearRightSensor
  GOSUB Change_IO_Direction
  sensor8 = SensorInput
  LOW OuterRearRightSensor

  HIGH InnerRearRightSensor
  GOSUB Change_IO_Direction
  sensor9 = SensorInput
  LOW InnerRearRightSensor

  HIGH InnerRearLefttSensor
  GOSUB Change_IO_Direction
  sensor10 = SensorInput
  LOW InnerRearLefttSensor

  HIGH OuterRearLeftSensor
  GOSUB Change_IO_Direction
  sensor11 = SensorInput
  LOW OuterRearLeftSensor

  #IF DebugNormal #THEN
   DEBUG "wallSensors = ", BIN4 wallSensors.NIB2, "-", BIN sensor7, BIN sensor6, BIN sensor5, "x", "-", "x", BIN
    sensor2, BIN sensor1, BIN sensor0, CR
  #ENDIF

  RETURN

' ----------------------------------------
Change_IO_Direction:
  HIGH SensorInput              ' Push signal voltages to 5 V
  PAUSE 0                     ' Wait 230 us for capacitors
  INPUT SensorInput             ' Start the decays
  'PULSOUT UnusedPin, sensorThreshold  ' Wait for threshold time
  PULSOUT UnusedPin, W9             ' Wait for threshold time
  RETURN

' ----------------------------------------
Go_Forward:
  GOSUB Set_Go_Forward_Duration
  FOR B21 = 1 TO Loop4GoForward
    GOSUB Pulsout_Motor
    PAUSE Pause4GoForward
  NEXT
  RETURN

' ----------------------------------------
Set_Go_Forward_Duration:
  W12 = StopMotor + GoForwardFactor
  W11 = StopMotor - GoForwardFactor
  RETURN

' ----------------------------------------
Pivot_Right:
  GOSUB Set_Pivot_Right_Duration
  FOR B21 = 1 TO Loop4Pivot
    GOSUB Pulsout_Motor
    PAUSE Pause4Pivot
  NEXT
  RETURN

' ----------------------------------------
Set_Pivot_Right_Duration:
  W12 = StopMotor + PivotFactor
  W11 = StopMotor
  RETURN
```

```
' ----------------------------------------
Pivot_Left:
  GOSUB Set_Pivot_Left_Duration
  FOR B21 = 1 TO Loop4Pivot
    GOSUB Pulsout_Motor
    PAUSE Pause4Pivot
  NEXT
  RETURN

' ----------------------------------------
Set_Pivot_Left_Duration:
  W12 = StopMotor
  W11 = StopMotor - PivotFactor
  RETURN

' ----------------------------------------
Stop_Motor:
  GOSUB Set_Stop_Motor_Duration
  GOSUB Pulsout_Motor
  RETURN

' ----------------------------------------
Set_Stop_Motor_Duration:
  W12 = StopMotor
  W11 = StopMotor
  RETURN

' ----------------------------------------
Pulsout_Motor:
  PULSOUT LeftMotor, W12
  PULSOUT RightMotor, W11
  RETURN

' ----------------------------------------
Get_Word:
  GET B20, Word W9
  RETURN

'' ----------------------------------------
'Pivot_Left_Scanning:
'
'  GOSUB Set_Pivot_Left_Duration
'
'  FOR B20 = 1 TO Loop4PivotLeftScan
'    GOSUB Pulsout_Motor
'    PAUSE Pause4Scan
'    GOSUB Read_Wall_Sensors
'    IF (sensor6 OR sensor5 OR sensor2 OR sensor1) THEN
'      scanResult = IsHigh
'      EXIT
'    ENDIF
'  NEXT
'
'  RETURN

'' ----------------------------------------
'Pivot_Right_Scanning:
'
'  GOSUB Set_Pivot_Right_Duration
'
'  FOR B20 = 1 TO Loop4PivotRightScan
'    GOSUB Pulsout_Motor
'    PAUSE Pause4Scan
'    GOSUB Read_Wall_Sensors
'    IF (sensor6 OR sensor5 OR sensor2 OR sensor1) THEN
'      scanResult = IsHigh
'      EXIT
'    ENDIF
'  NEXT
'
'  RETURN
```

Kode Program Bank 6 EEPROM

```
' -----[ Initialization ]------------------------------------------------------
'
' -----[ Main Code ]-------------------------------------------------------------
'
' ----------------------------------------
Do_Output_2:

  #IF DebugLow #THEN
    DEBUG CR, "OUTPUT_2:", CR
  #ENDIF

  #IF DebugNormal #THEN
    GOSUB Display_Data
  #ENDIF

' ----------------------------------------
Do_Input:
  task = TskDoInput  ' TskDoInput = 1
  RUN Bank_0

' -----[ Subroutines ]-----------------------------------------------------------
'
' ----------------------------------------
Display_Data:
  #IF DebugLow #THEN
    DEBUG CR, "DISPLAY_DATA:", CR
  #ENDIF

  #IF DebugNormal #THEN
    'DEBUG CLS
    GOSUB Parameter_Reading
    GOSUB Maze_Reading
  #ENDIF

  RETURN

' ----------------------------------------
Parameter_Reading:

  #IF DebugNormal #THEN

    DEBUG "[x,y] = [", DEC nowX, ",", DEC nowY, "]", "; ", "nowDist = ", DEC nowDist, "; ", "nowWall = ", BIN4
      nowWall.NIB1, "-", BIN4 nowWall.NIB0, "; ", "nowOrient = ", BIN4 nowOrient.NIB1, "-", BIN4 nowOrient.NIB0, CR
    DEBUG "isVisitedCell = ", BIN isVisitedCell, "; ", "isCheckpoint = ", BIN isCheckpoint, "; ", "isFinish = ", BIN isFinish,
      "; ", "isBrokenRule = ", BIN isBrokenRule, "; ", CR
    'DEBUG "sensorThreshold = ", DEC sensorThreshold, "; ", "numOfOpenDist = ", DEC numOfOpenDist, "; ", "openDist
      = ", IBIN4 openDist, "; ", "moveType = ", IBIN4 moveType, "; ", "scanResult = ", BIN scanResult, CR
    DEBUG "numOfOpenDist = ", DEC numOfOpenDist, "; ", "openDist = ", IBIN4 openDist, "; ", "moveType = ", IBIN4
      moveType, "; ", "scanResult = ", BIN scanResult, CR

  #ENDIF

  RETURN

' ----------------------------------------
Maze_Reading:

  #IF DebugNormal #THEN

    DEBUG CR, "MAZE: COORDINATE + DISTANCE + VISITED-WALL", CR

    FOR B23 = MatrixSizeMinSatu TO 0
      FOR B22 = 0 TO MatrixSizeMinSatu

        B21 = (B22 * MatrixSize) + B23
        B24 = B21 + CoordinateValue
        GOSUB Get_Byte
        DEBUG HEX2 B25, " + "
        B24 = B21 + DistanceValue
        GOSUB Get_Byte
        DEBUG DEC B25, " + "
        B24 = B21 + WallValue
        GOSUB Get_Byte
        DEBUG BIN4 B25.NIB1, "-", BIN4 B25.NIB0, " | "

      NEXT
      DEBUG CR
    NEXT

    DEBUG CR, "MAZE: COORDINATE + DISTANCE + ORIENTATION", CR
```

```
  FOR B23 = MatrixSizeMinSatu TO 0
    FOR B22 = 0 TO MatrixSizeMinSatu

      B21 = (B22 * MatrixSize) + B23
      B24 = B21 + CoordinateValue
      GOSUB Get_Byte
      DEBUG HEX2 B25, " + "
      B24 = B21 + DistanceValue
      GOSUB Get_Byte
      DEBUG DEC B25, " + "
      B24 = B21 + OrientValue
      GOSUB Get_Byte
      DEBUG BIN4 B25.NIB1, "-", BIN4 B25.NIB0, " | "

    NEXT
    DEBUG CR
  NEXT

 #ENDIF

 RETURN

' --------------------------------------
Get_Byte:
 GET B24, B25
 RETURN
```