บทที่ 3
วิธีดำเนินการวิจัย

การวิจัยนี้แบ่งการทำงานออกเป็นสองส่วน ส่วนแรกคือ การติดตั้งเครื่องกัดและระบบควบคุมเข้ากับเครื่องคอมพิวเตอร์ส่วนบุคคล ทำการเขียนโปรแกรมควบคุม รับคำสั่ง ประมวลผล แสดงสถานะ และสร้างสัญญาณดิจิตอลไปควบคุมแผงวงจรควบคุมเสต็ปปิ้งมอเตอร์ในแต่และแกน โดยสื่อสารผ่านทางพอร์ทขนาน ซึ่งในการวิจัยส่วนนี้ จะทำให้ผู้ที่สนใจหรือผู้ที่กำลังศึกษาได้นำไปใช้ในการพัฒนาระบบควบคุมเครื่องกัดซีเอ็นซีบนคอมพิวเตอร์ส่วนบุคคลได้ง่ายและชัดเจน โดยระบบควบคุมสร้างจากกระบวนการในภาษาปาสคาลที่โปรแกรมภาษาสามารถดาวน์โหลดมาจากผู้ผลิตได้อย่างถูกกฎหมาย ส่วนที่สองคือ การนำเอาระบบควบคุมเครื่องกัดซีเอ็นซีไปติดตั้งบนเครื่องคอมพิวเตอร์แบบแผงวงจรเดี่ยว ซึ่งจะมีความแตกต่างทางด้านกายภาพ การรับข้อมูลหรือโปรแกรม NC ซึ่งต้องรับข้อมูลผ่านทางพอร์ทอีเทอร์เน็ต และการติดต่อสื่อสารข้อมูลกับแผงวงจรควบคุมเสต็ปปิ้งมอเตอร์ผ่านทางพอร์ทเอนกประสงค์ 16 บิต ซึ่งทำให้ต้องปรับปรุงโปรแกรมควบคุมให้มีรูปแบบการสั่งงานที่เหมาะสมกับเครื่องคอมพิวเตอร์แบบแผงวงจรเดี่ยว ซึ่งองค์ประกอบและข้อมูลที่สำคัญของเครื่องจะแสดงไว้ในภาคผนวก ข

3.1 เครื่องมือและอุปกรณ์

3.3.1 เครื่องกัดแนวดิ่งขนาดเล็ก SHERLINE รุ่น 5410 ดังรูป 3.1

3.3.2 เครื่องคอมพิวเตอร์ส่วนบุคคล ติดตั้งระบบปฏิบัติการ MS DOS

3.3.3 เครื่องคอมพิวเตอร์แบบแผงวงจรเดี่ยวชนิด PC /104 รุ่น Mity-Mite ดังรูป 3.2 และ VGA Module

3.3.4 แผงวงจรควบคุมเสต็ปปิ้งมอเตอร์ของ SILA Research รุ่น EX-STEPM พร้อมแผงวงจรควบคุมแรงดัน จำนวน 3 ชุด ดังรูป 3.3

3.3.5 แหล่งจ่ายไฟขนาด 300 วัตต์ ขนาดแรงดัน 5 โวล์ท และ 12 โวล์ท

3.3.6 ปากกาจับงานบนโต๊ะงานขนาด 3 นิ้ว

3.3.7 เกจนาฬิกาขนาดความละเอียด 0.01 มิลลิเมตร พร้อมฐานแม่เหล็ก

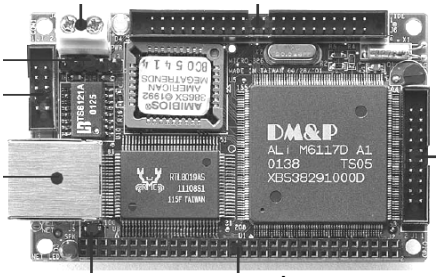
3.3.8 ไมโครมิเตอร์ขนาดความละเอียด 0.001 มิลลิเมตร

3.3.9 มีดกัดแบบ End Mill ขนาดเส้นผ่านศูนย์กลาง 5 มิลลิเมตร

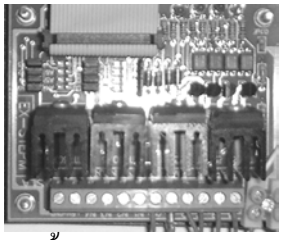
3.3.10 มีดกัดแบบ Ball Mill ขนาดเส้นผ่านศูนย์กลาง 3 มิลลิเมตร

3.3.11 วัสดุที่ใช้ทดสอบการทำงาน ได้แก่ พาราฟิน และ อะคีลิค

รูป 3.1 เครื่องกัดแนวดิ่งขนาดเล็ก SHERLINE รุ่น 5410



รูป 3.2 เครื่องคอมพิวเตอร์แบบแผงวงจรเดี่ยวชนิด PC /104 รุ่น Mity-Mite



รูป 3.3 แผงวงจรควบคุมเสต็ปปิ้งมอเตอร์ของ SILA Research รุ่น EX-STEPM

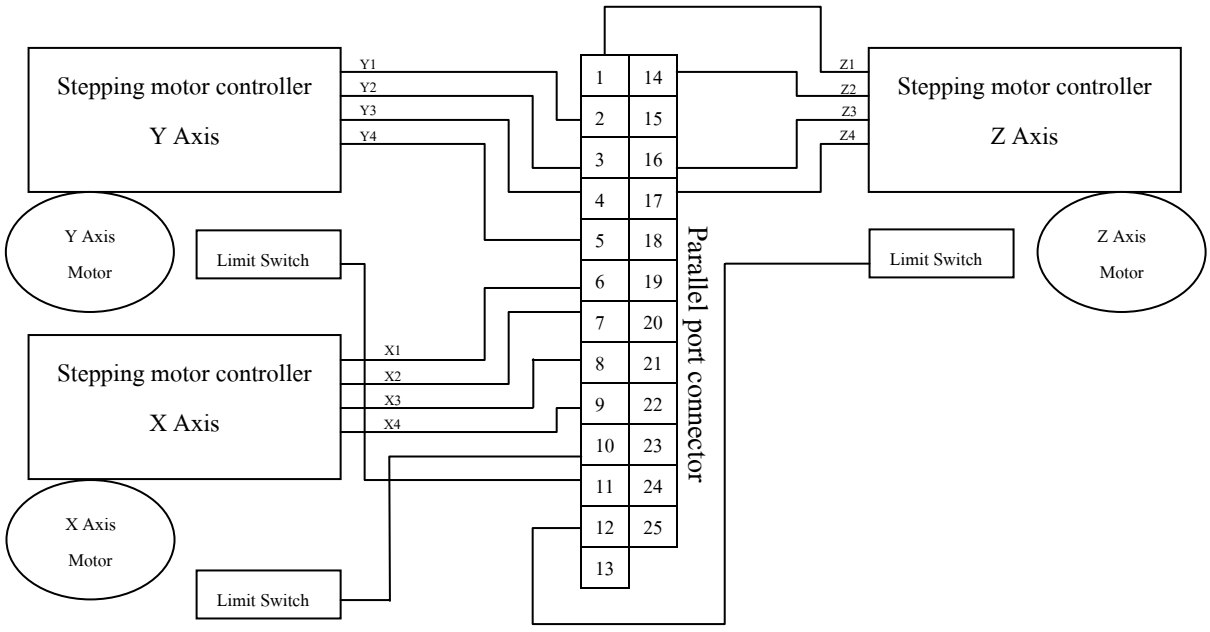
3.2 วิธีดำเนินการวิจัย

3.2.1 การติดตั้งระบบโดยใช้เครื่องคอมพิวเตอร์ส่วนบุคคลเป็นหน่วยประมวลผล

เมื่อใช้เครื่องคอมพิวเตอร์ส่วนบุคคลเป็นหน่วยประมวลผล ระบบจึงต้องมีการติดต่อสื่อสารกันระหว่างอุปกรณ์ภายนอกกับคอมพิวเตอร์ผ่านทางพอร์ทขนาน (LPT1) และจะต้องเขียนโปรแกรมควบคุมโดยกำหนดแอดเดรสของพอร์ทขนานอีกด้วย

3.2.1.1 การติดตั้งระบบควบคุมเสต็ปปิ้งมอเตอร์เข้ากับพอร์ทขนาน

จะเห็นได้ว่าพอร์ทขนาน 1 ชุด สามารถควบคุมเสต็ปปิ้งมอเตอร์ได้ถึง 3 ตัว เนื่องจากพอร์ทขนาน 1 ชุดมีบิตที่ใช้งานได้ถึง 16 บิต แต่มอเตอร์ 1 ตัวต้องการใช้บิตควบคุมเพียง 4 บิตเท่านั้น ดังนั้นพอร์ทเอาท์พุต (Data) 8 บิต จึงสามารถใช้ควบคุมมอเตอร์ได้ 2 ตัว และพอร์ทอินพุต / เอาท์พุต (Control) 4 บิต จะสามารถควบคุมมอเตอร์ได้อีก 1 ตัว ส่วนพอร์ทอินพุต (Status) นั้นใช้รับสัญญาณจาก Limit Switch ก็ได้ หากเครื่องคอมพิวเตอร์ส่วนบุคคลมีพอร์ทขนาน 2 ชุดจะสามารถควบคุมมอเตอร์ได้ถึง 6 ตัว สัญญาณที่ใช้ควบคุมเสต็ปปิ้งมอเตอร์จากพอร์ทขนานจะถูกส่งไปยังแผงวงจรควบคุมเสต็ปปิ้งมอเตอร์ เพื่อแยกสัญญาณออกจากกันโดยถ่ายทอดข้อมูลผ่านทางโฟโต้ทรานซิสเตอร์ ทำให้สามารถป้องกันกระแสไฟฟ้าย้อนกลับไปสร้างความเสียหายยังเครื่องคอมพิวเตอร์ส่วนบุคคลได้ การต่อวงจรเพื่อควบคุมเสต็ปปิ้งมอเตอร์ทั้ง 3 ตัวแสดงดังรูป 3.4



รูป 3.4 การต่อวงจรควบคุมเสต็ปปิ้งมอเตอร์สำหรับมอเตอร์ 3 ชุดกับพอร์ทขนาน

จากรูป 3.4 แผงวงจรควบคุมเสต็ปปิ้งมอเตอร์ของแกน Y และ X จะใช้พอร์ทขนานมาตรฐาน (Standard Parallel Port : SPP) ในช่องสัญญาณ Data 0-3 (Pin 2-5) และ Data 4-7 (Pin 6-9) ตามลำดับ ดังแสดงในตาราง 3.1 และแผงวงจรควบคุมเสต็ปปิ้งมอเตอร์ของแกน Z จะใช้ SPP ในช่องสัญญาณ Control 0-3 คือ nStrobe (Pin1), nAuto-Linefeed (Pin14), nInitialize (Pin16) และ nSelect-Printer/nSelect-In (Pin17) ดังตารางที่ 3.3

ส่วน Limit switch ของมอเตอร์แกน Y, X และ Z จะใช้ SPP ในช่องสัญญาณ Status คือ Busy (Pin11), Act (Pin10) และ Paper-Out/Paper-End (Pin 12) ตามลำดับ ดังแสดงในตาราง 3.2

ตาราง 3.1 Standard Parallel Port (SPP) ของพอร์ท Data

Offset	Name	Read/Write	Bit No.	Properties	Connect
Base+0	Data Port	Write	Bit 7	Data 7	Pin 9 to Motor X 4
			Bit 6	Data 6	Pin 8 to Motor X 3
			Bit 5	Data 5	Pin 7 to Motor X 2
			Bit 4	Data 4	Pin 6 to Motor X 1
			Bit 3	Data 3	Pin 5 to Motor Y 4
			Bit 2	Data 2	Pin 4 to Motor Y 3
			Bit 1	Data 1	Pin 3 to Motor Y 2
			Bit 0	Data 0	Pin 2 to Motor Y 1

ตาราง 3.2 Standard Parallel Port (SPP) ของพอร์ท Status

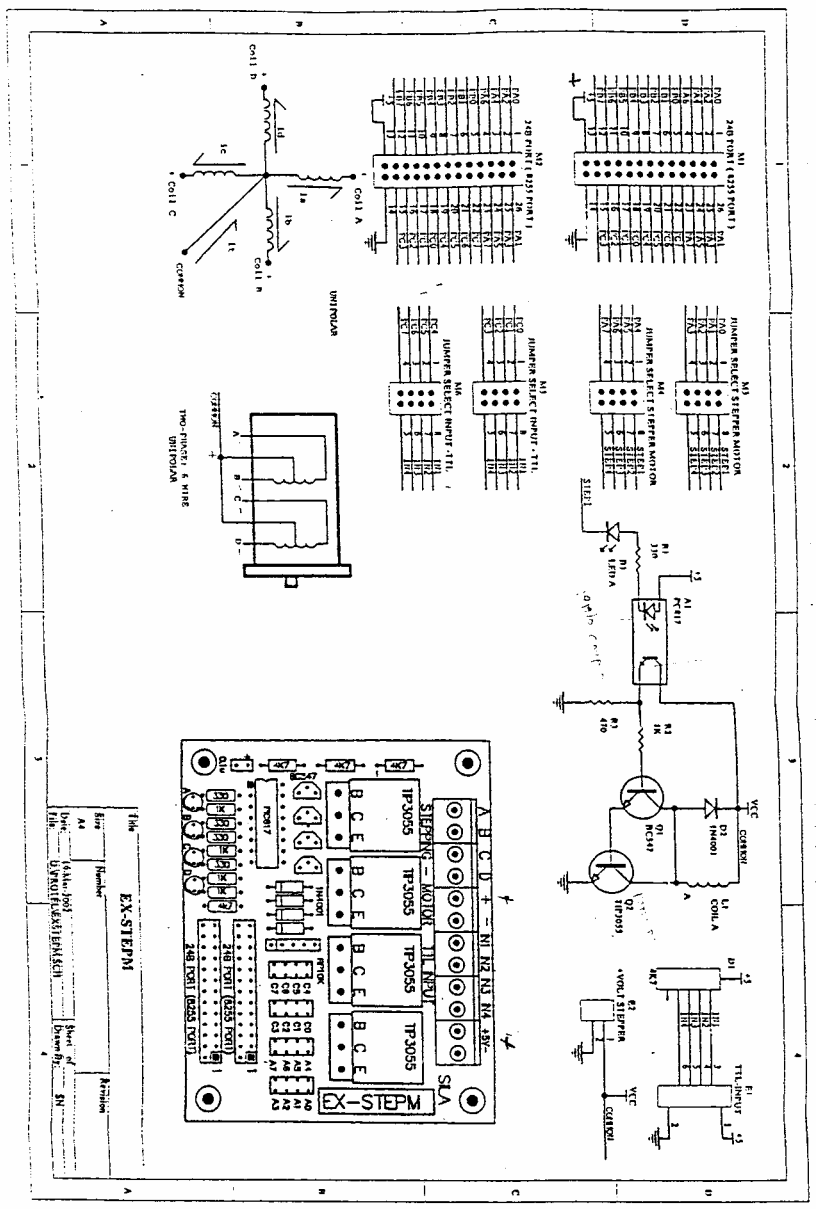
Offset	Name	Read/Write	Bit No.	Properties	Connect
Base+1	Status Port	Read Only	Bit 7	Busy	Pin 11 to Limit switch Y
			Bit 6	nAck	Pin 10 to Limit switch X
			Bit 5	Paper Out	Pin 12 to Limit switch Z
			Bit 4	nSelect In	
			Bit 3	Error	
			Bit 2	IRQ (Not)	
			Bit 1	Reserved	
			Bit 0	Reserved	

ตาราง 3.3 Standard Parallel Port (SPP) ของพอร์ท Control

Offset	Name	Read/Write	Bit No.	Properties	Connect
Base+2	Control Port	Read/Write	Bit 7	Unused	
			Bit 6	Unused	
			Bit 5	Enable Bi-Direction Port	
			Bit 4	Enable IRQ Via Ack Line	
			Bit 3	nSelect Printer	Pin 17 to Motor Z 4
			Bit 2	nInitialize	Pin 16 to Motor Z 3
			Bit 1	nAuto-Linefeed	Pin 14 to Motor Z 2
			Bit 0	nStrobe	Pin 1 to Motor Z 1

3.2.1.2 การติดตั้งแผงวงจรควบคุมเสต็ปปิ้งมอเตอร์และแผงวงจรควบคุมแรงดันไฟฟ้า

แผงวงจรควบคุมเสต็ปปิ้งมอเตอร์ได้เลือกใช้แผงวงจรสำเร็จรูปของ SILA-Research รุ่น EXSTEPM ซึ่งมีวงจรในแต่ละช่องสัญญาณดังรูป 3.5 มาใช้ควบคุมเสต็ปปิ้งมอเตอร์ในแต่ละแกน รวม 3 ชุด โดยทำการติดตั้งตัวต้านทานขนาด 0.5 โอห์ม 5 วัตต์กับเซนแตอร์แทบทั้งสองเส้นของมอเตอร์ เพื่อป้องกันแรงดันย้อนกลับและป้องกันแรงบิดเบรกเนื่องจากการไหลวนของกระแส การติดตั้งสำหรับระบบควบคุมแกน X ทำได้โดยเชื่อมสายสัญญาณจาก LPT1 ขาที่ 6, 7, 8, และ 9 กับ ขาที่ 1, 26, 2, 25 ของแผงวงจรควบคุมเสต็ปปิ้งมอเตอร์ในแกน X ตามลำดับ ระบบควบคุมแกน Y ทำได้โดยเชื่อมสายสัญญาณจาก LPT1 ขาที่ 2, 3, 4,และ 5 กับ ขาที่ 1, 26, 2, 25 ของแผงวงจรควบคุมเสต็ปปิ้งมอเตอร์ในแกน Y ตามลำดับ และระบบควบคุมแกน Z ทำได้โดยเชื่อมสายสัญญาณจาก LPT1 ขาที่ 1, 14, 16,และ 17 กับ ขาที่ 1, 26, 2, 25 ของแผงวงจรควบคุมเสต็ปปิ้งมอเตอร์ในแกน Z ตามลำดับ รวมทั้งเชื่อสายสัญญาณนำเข้าจากลิมิตสวิทซ์แกน X, Y และ Z กับช่อง IN1, IN2 และ IN3 ของวงจรควบคุมเสต็ปปิ้งมอเตอร์ในแกน Y ตามลำดับ โดยที่เชื่อมสายสัญญาณจากขาที่ 18, 17 และ 16 ของวงจรควบคุมเสต็ปปิ้งมอเตอร์ในแกน Y กับขาที่ 10, 11 และ 12 ของพอร์ท LPT1 ตามลำดับดังแสดงในตาราง 3.4



รูป 3.5 วงจรควบคุมเสต็ปปิ้งมอเตอร์

ตาราง 3.4 การเชื่อมสายสัญญาณต่าง ๆ ของแผงวงจรควบคุมเสต็ปปิ้งมอเตอร์

Controller	Stepping motor controller board		Connect to	
	Pin No.	Name	Port	Identify
X	1	PA0	LPT1	Pin 2
	2	PA2	LPT1	Pin 4
	13	+5V	Power Supply	+5V
	14	Ground	LPT1	Pin 18
	25	PA3	LPT1	Pin 5
	26	PA1	LPT1	Pin 3
	A	STEP1	Motor X	Brown
	B	STEP1	Motor X	Red
	C	STEP1	Motor X	Orange
	D	STEP1	Motor X	Yellow
Y	1	PA0	LPT1	Pin 2
	2	PA2	LPT1	Pin 4
	13	+5V	Power Supply	+5V
	14	Ground	LPT1	Pin 19
	16	PC2	LPT1	Pin 12
	17	PC1	LPT1	Pin 11
	18	PC0	LPT1	Pin 10
	25	PA3	LPT1	Pin 5
	26	PA1	LPT1	Pin 3
	A	STEP1	Motor Y	Brown
	B	STEP1	Motor Y	Red
	C	STEP1	Motor Y	Orange
	D	STEP1	Motor Y	Yellow
	IN1	IN1	Limit Switch	X
	IN2	IN2	Limit Switch	Y
	IN3	IN3	Limit Switch	Z
	+5V	+5V	Limit Switch	X, Y and Z
Z	1	PA0	LPT1	Pin 1
	2	PA2	LPT1	Pin 16
	13	+5V	Power Supply	+5V
	14	Ground	LPT1	Pin 20
	25	PA3	LPT1	Pin 17
	26	PA1	LPT1	Pin 14
	A	STEP1	Motor Z	Brown
	B	STEP1	Motor Z	Red
	C	STEP1	Motor Z	Orange
	D	STEP1	Motor Z	Yellow

3.2.1.3 การเขียนโปรแกรมควบคุมการทำงาน

การเขียนโปรแกรมควบคุมการทำงานของระบบ ได้เลือกใช้โปรแกรมภาษา Turbo Pascal 4.0 ในการสร้างโปรแกรมเพื่อรับคำสั่ง ประมวลผลการทำงาน สร้างสัญญาณดิจิตอล และแสดงสถานการทำงานของระบบ โปรแกรมแบ่งออกเป็น 3 ส่วนแรกคือ ส่วนหัว และการประกาศตัวแปร ซึ่งเป็นส่วนที่กำหนดค่าและคุณสมบัติของตัวแปรต่าง ๆ ของโปรแกรม รวมทั้งกำหนดค่าตัวแปรสถานะของการรับสัญญาณจากลิมิตสวิทซ์ โดยกำหนด Switch1 เป็นตัวแปรแบบตรรกะ ที่รับค่าสถานการทำงานจากลิมิตสวิทซ์ของมอเตอร์แกน X และ Switch2, Switch3 สำหรับมอเตอร์แกน Y, Z ตามลำดับ ดังรูป 3.7

```
Program MOTOR;
{Program uses Turbo Pascal V 4.0 Copmpiler}
{Modification Date 27 JAN 2004}
uses crt;
Var Phase1,Phase2,Phase3,K,M,I,T,Na,Nb,Nc,checkNb : Integer;
   StepCountNow,StepCountNeed,StepDiff : Array [1..3] of LongInt;
   compos,coorpos, Hz,Position          : Array [1..3] of Real;
   coorposstep,cirrun      : Array [1..2] of integer;
   coordiff                : Array [1..4] of real;
   Order,Pointer           : Array [1..3] of integer;
   CheckPo, Radius,e       : real;
   DataPort                : Word;
   Niden,Giden,sp          : Char;
   Code,Ano,Gmode,DrX,DrY,DrZ    : Integer;
   CDR,cdr1,cdr2,x,y,xst,free    : Integer;
   MforX,MforY,MforZ,step,Dryst  : Integer;
   radiusstep,cngpnt       : Integer;
   Axis,coor               : Array[1..3] of char;
   Filename                : Array[1..12] of char;
   a,b,c,d,g               : Real;
   f,Buff,Quad,Ncode       : Integer;
   ch                      : Array[1..3] of Boolean;
   Millimeter,AbsPos,LoopCheck: Boolean;
   Auto,check,circheck     : Boolean;
   Readfile                : String[12];
   NcFile                  : text;
Const HzMax : Real = 300;  HzStart : Real = 200;
```

```
Function Switch1 : Boolean;
 Begin
  If (Port[DataPort+1] and $80 > 0) then Switch1 := True
    Else Switch1 := False;
 End;
Function Switch2 : Boolean;
 Begin
  If (Port[DataPort+1] and $20 = 0) then Switch2 := True
    Else Switch2 := False;
 End;
Function Switch3 : Boolean;
 Begin
  If (Port[DataPort+1] and $10 = 0) then Switch3 := True
    Else Switch3 := False;
 End;
```

รูป 3.7 ตัวอย่างส่วนของโปรแกรมที่กำหนดค่าตัวแปรสถานะของการรับสัญญาณจากลิมิตสวิทซ์

ส่วนที่สอง คือการกำหนดโพรซิเยอร์สำหรับการทำงานต่าง ๆ ได้แก่ FindPrinterPort, SetTimer, WaitTimer, DriveStep, GetReady, UnloadMotor, FindZero1, FindZero2, FindZero3, Display, DisPos, Mwait, LinearInput และ CirInput โดยที่จะได้อธิบายแนวคิดการทำงานแต่ละขั้นตอนทีละโพรซิเยอร์ ดังนี้

- FindPrinterPort

เป็นโพรซิเยอร์ที่ใช้สำหรับให้เครื่องคอมพิวเตอร์ได้กำหนดแอดเดรสของพอร์ทเครื่องพิมพ์จาก Start Address 0000:0408 ที่มี Base Address ของ LPT1 ไว้ยังตัวแปร DataPort เพื่อให้สามารถเรียกใช้งานได้อย่างถูกต้อง โดยที่มีรูปแบบของโปรแกรมดังรูป3.8

```
Procedure FindPrinterPort;
 Begin
  DataPort := MemW[$0000:$0408]; {LPT1 addres in BIOS}
  If DataPort = 0 then
   Begin
    WriteLn('Printer port address not found in BIOS');
    Exit;
   End;
 End;
```

รูป 3.8 โพรซิเยอร์ FindPrinterPort

- SetTimer

เป็นโพรซิเยอร์ที่ใช้กำหนดความเร็วในการทำงานจากการกำหนดความถี่จากตัวแปร Rate โดยบรรทัดแรกของโปรแกรมจะแปลงค่าความถี่ที่ต้องการให้เป็นตัวเลขไม่มีเศษ เป็นความยาวช่วงเวลาที่เหมาะสมและตรวจสอบว่าช่วงเวลาที่จะให้ตั้งนั้นนานเกินไปหรือไม่ (ค่าเกิน 65535 หรือ 2 Byte) ถ้าเกินจะปรับค่าให้เป็นค่าสูงสุดที่วงจรจับเวลาจะสามารถรับได้ จากนั้นจึงแบ่งค่าที่ใช้จับเวลาเป็น 2 Byte เพื่อเตรียมส่งไปให้วงจรจับเวลา ถ้าหากช่วงเวลาสั้นไปกว่าที่โปรแกรมจะทำงานได้ทัน ก็จะมีการปรับช่วงเวลาให้เหมาะสมอีกครั้ง จากนั้นโปรแกรมจะสั่งให้วงจรจับเวลา 2 ช่องรับสัญญาณนาฬิกาที่ป้อนเข้ามา แล้วจึงโปรแกรมวงจรจับเวลาช่องสองให้สามารถรับข้อมูลทีละ 2 Byte โดยทำงานเป็นแบบ Rate Generator (Mode 2) และการนับเวลาเป็นแบบ Binary แล้วจึงสั่งค่าตั้งเวลาที่เหมาะสมไปยังวงจรจับเวลาโดยส่ง Least Significant Byte (LSB) ก่อนแล้วจึงส่ง Most Significant Byte (MSB) ตามลำดับไปยังวงจรรวมหมายเลข 8253 ซึ่งมีติดตั้งในคอมพิวเตอร์ทุกเครื่อง ซึ่งแม้ว่าจะติดตั้งโปรแกรมควบคุมลงบนระบบปฏิบัติการใดหรือหน่วยประมวลผลความเร็วใดก็ตาม โปรแกรมจะสามารถควบคุมให้ระบบทำงานด้วยความเร็วเดิมได้ทุกครั้ง โดยโปรแกรมมีรูปแบบดังรูป 3.9

```
Procedure SetTimer(Rate : Real); {Set internal timer Rate in Hz}
 Var TimerMSB,TimerLSB : Byte;
    TimerWord : Word;
    TimerValue : Longint;
 Begin
  TimerValue := Round(1.19318E06/Rate);
If TimerValue > 65535 then TimerWord := 65535
    else TimerWord := TimerValue;
  TimerMSB := TimerWord div 256;
  TimerLSB := TimerWord mod 256;
  If TimerWord < 380 then
   Begin
    TimerMSB := 1;
    TimerLSB := 125;
   End;
  Port[$61] := Port[$61] or 1; {Enable Clock}
  Port[$43] := $B4; {10110100 = Channal 2, 2 byte, Mode 2, Binary}
  Port[$42] := TimerLSB;
  Port[$42] := TimerMSB;
 End;
```

รูป 3.9 โพรซิเยอร์ SetTimer

- WaitTimer

เป็นโพรซิเยอร์ที่กำหนดเวลาให้เครื่องคอมพิวเตอร์หยุดรอเป็นวาลาที่กำหนดโดยโพรซิเยอร์ SetTimer ซึ่งจะทำให้เครื่องกัดสามารถทำงานตามเวลาจริง (Realtime) ซึ่งโปรแกรมจะอ่านค่า Latch ค่าออกมาที่รีจิสเตอร์ของชุดจับเวลาโดยไม่หยุดการทำงานของวงจรแล้วตรวจสอบว่า MSB เป็น 0 หรือยัง เมื่อเป็น 0 แล้วจึงตรวจสอบ LSB ว่าข้าม 0 หรือไม่ โปรแกรมจะไม่ตรวจสอบ LSB เพื่อหาค่า 0 โดยตรงเพราะการทำงานของโปรแกรมภาษาปาสคาลจะไม่เร็วพอที่จะตรวจเช็คได้ โดยมีรูปแบบโปรแกรมดังรูป 3.10

```
Procedure WaitTimer; {Wait for timer count to cross zero}
 Var Dummy : Byte;
 Begin
  Repeat
   Port[$43] := $80;   {Latch register value}
   Dummy := Port[$42]; {Read LSB}
  Until Port[$42] = 0;  {Check MSB}
  Repeat
   Port[$43] := $80;
   Dummy := Port[$42];
  Until Port[$42] > 0;
 End;
```

รูป 3.10 โพรซิเยอร์ WaitTimer

- DriveStep

เป็นโพรซิเยอร์ที่สร้างสัญญาณดิจิตอลเพื่อส่งไปยัง LPT1 ให้ขับเสต็ปปิ้งมอเตอร์ในแกนต่าง ๆ โดยการขับมอเตอร์จะเป็นแบบ 2 เฟส เพื่อให้ได้แรงบิดที่สูงขึ้น โปรแกรมจะเก็บอาเรย์ข้อมูลเพื่อสร้างสัญญาณดิจิตัลสำหรับมอเตอร์แกนต่าง ๆ ทั้ง 3 แกนโดยที่มีรูปแบบแตกต่างกันไป โดยเฉพาะในแกน Z ซึ่งบิตที่ 3, 1 และ 0 จะต้องเป็นข้อมูลแบบอินเวอร์ส เพราะพอร์ทที่สั่งออกไปนั้นเป็นแบบอินเวอร์สด้วย เมื่อโปรแกรมรับค่าหมายเลขมอเตอร์และทิศทางการหมุนแล้ว จะทำการเลือกชุดข้อมูลและป้อนสัญญาณดิจิตอลไปยังพอร์ท LPT รวมทั้งเก็บค่าตำแหน่งที่มอเตอร์เคลื่อนไป โดยมีรูปแบบโปรแกรมดังรูป 3.11

```
Procedure DriveStep(Nba,Direction : Integer);
 Const Sphase1 : Array[1..4] of byte = ($0C,$06,$03,$09);
       Sphase2 : Array[1..4] of byte = ($C0,$60,$30,$90);
       Sphase3 : Array[1..4] of byte = ($07,$0D,$08,$02);
```

```
{Stepper drive low active drive pattern 1100 0110 0011 1001}
{Bits 3 1 and 0 of motor 3 hardware inverted in PC}
Begin
 Case Nba of
  1 : If Direction > 0 then
        Begin
         Inc(Phase2); If Phase2 > 4 then Phase2 := 1;
         Port[DataPort] := Port[DataPort] and $0F or Sphase2[Phase2];
        End
       Else
         Begin
          Dec(Phase2); If Phase2 < 1 then Phase2 := 4;
          Port[DataPort] := Port[DataPort] and $0F or Sphase2[Phase2];
         End;
  2  : If Direction > 0 then
        Begin
         Inc(Phase1); If Phase1 > 4 then Phase1 := 1;
         Port[DataPort] := Port[DataPort] and $F0 or Sphase1[Phase1];
        End
       Else
         Begin
          Dec(Phase1); If Phase1 < 1 then Phase1 := 4;
          Port[DataPort] := Port[DataPort] and $F0 or Sphase1[Phase1];
         End;
  3 : If Direction > 0 then
        Begin
         Inc(Phase3); If Phase3 > 4 then Phase3 := 1;
         Port[DataPort+2] := Port[DataPort+2] and $F0 or Sphase3[Phase3];
        End
       Else
        Begin
         Dec(Phase3); If Phase3 < 1 then Phase3 := 4;
         Port[DataPort+2] := Port[DataPort+2] and $F0 or Sphase3[Phase3];

         End;
 End; {Case}
End;
```

รูป 3.11 โพรซิเยอร์ DriveStep

- GetReady

เป็นโพรซิเยอร์ที่โปรแกรมหลักจะเรียกทำงานในช่วงเริ่มทำงานของโปรแกรมเพียงครั้งเดียว เพื่อกำหนดค่าเริ่มต้นการทำงาน การเรียกใช้โพรซิเยอร์ FindPrinterPort สร้างสัญญาณให้แผงวงจรควบคุมเสต็ปปิ้งมอเตอร์เพื่อให้พร้อมกับการรับคำสั่งการขับมอเตอร์ และกำหนดค่าความเร็วในการทำงานด้วยค่าความเร็วเริ่มต้น มีรูปแบบโปรแกรมดังรูป 3.12

```
Procedure GetReady;
 Begin
  Phase1 := 1; Phase2 := 1; Phase3 := 1;
  FindPrinterPort;
  Port[DataPort] := $CC;
  Port[DataPort+2] := Port[DataPort+2] and $F0 or $07;
  SetTimer(HzStart);
 End;
```

รูป 3.12 โพรซิเยอร์ GetReady

- UnloadMotor

เป็นโพรซิเยอร์ที่ใช้เมื่อสิ้นสุดการทำงานของโปรแกรม โดยจะส่งข้อมูลเพื่อตัดสัญญาณไฟฟ้าที่จะส่งยังวงจรควบคุมเสต็ปปิ้งมอเตอร์ทั้ง 3 แกน มีรูปแบบโปรแกรมดังรูปที่ 3.13

```
Procedure UnloadMotor;
 Begin
  Port[DataPort] := $FF;
  Port[DataPort+2] := Port[DataPort+2] and $F0 or $04;
 End;
```

รูป 3.13 โพรซิเยอร์ UnloadMotor

- FindZero1, FindZero2, FindZero3

ทั้ง 3 โพรซิเยอร์มีรูปแบบที่คล้ายคลึงกัน ใช้ในการควบคุมให้เสต็ปปิ้งมอเตอร์เลื่อนแท่นงานไปยังจุดเริ่มต้น โดยที่เครื่องจะขับมอเตอร์ทั้ง 3 ตัว ไปจนกระทั่งแท่นงานไปชนกับลิมิตสวิทซ์ เมื่อลิมิตสวิทซ์สร้างสัญญาณป้อนกลับให้กับระบบแล้ว ระบบจะสั่งให้มอเตอร์หมุนเดินหน้าไปยังตำแหน่งออฟเซทที่ตั้งไว้ แล้วกำหนดให้ตัวแปรที่เก็บค่าตำแหน่งของแต่ละมอเตอร์เป็น 0 มีรูปแบบโปรแกรมดังรูป 3.14

```
Procedure FindZero1;
 Var I : Integer;
 Const X0offset : Integer = 160;
 Begin
  Gotoxy(25,3); WriteLn ('Finding Motor X zero position.');
  While not  Switch1 do
   Begin
    DriveStep(1,0);
    WaitTimer;
   End;
  For I := 1 to 10 do WaitTimer;
  While Switch1 do
   Begin
    DriveStep(1,1);
    WaitTimer;
   End;
  For I := 1 to X0offset do
   Begin
    DriveStep(1,1);
    WaitTimer;
   End;
  StepCountNow[1] := 0;
  While Switch2 do
   Begin
    DriveStep(2,1);
    WaitTimer;
   End;
  For I := 1 to Y0offset do
  gotoxy(60,8);
Write(stepcountnow[1]);
 End;

Procedure FindZero2;
 Var I : Integer;
 Const Y0offset : Integer = 160 ;
 Begin
  Gotoxy(25,3) ;WriteLn ('Finding Motor Y zero position.');
        While not Switch2 do
   Begin
    DriveStep(2,0);
    WaitTimer;
   End;
  For I := 1 to 10 do WaitTimer;
Begin
    DriveStep(2,1);
    WaitTimer;
   End;
 StepCountNow[2] := 0;
 gotoxy(60,11); Write(stepcountnow[2]);
End;

Procedure FindZero3;
Var I : Integer;
Const Z0offset : Integer = 160;
Begin
 Gotoxy(25,3); WriteLn ('Finding Motor Z zero position.');
 While not  Switch3 do
  Begin
   DriveStep(3,0);
   WaitTimer;
  End;
 For I := 1 to 10 do WaitTimer;
 While Switch3 do
  Begin
   DriveStep(3,1);
   WaitTimer;
  End;
 For I := 1 to Z0offset do
  Begin
   DriveStep(3,1);
   WaitTimer;
  End;
 StepCountNow[3] := 0;
 gotoxy(60,14); Writeln(stepcountnow[3]);
End;
```

รูป 3.14 โพรซิเยอร์ FindZero1, FindZero2 และ FindZero3

- Display

เป็นโพรซิเยอร์ที่ใช้กำหนดรูปแบบการแสดงผลบนหน้าจอเพื่อแสดงตำแน่ง สถานะ และรับข้อมูล ซึ่งประกอบไปด้วยส่วนแสดงสถานะการทำงาน ส่วนแสดงการอ้างอิงตำแหน่ง ความเร็วการทำงาน หน่วยวัด ตำแหน่งของมอเตอร์ในแกนต่าง ๆ ส่วนแสดงตำแหน่งละค่าประกอบในการทำงานแบบเคลื่อนที่เป็นวงกลม โหมดการทำงาน ชื่อโปรแกรม NC และส่วนรับคำสั่ง

- DisPos

เป็นโพรซิเยอร์ที่ใช้ช่วยเหลือในการเขียนโปรแกรมเพื่อแสดงตำแหน่งของมอเตอร์ต่าง ๆ รวมทั้งการบอกตำแหน่งโดยมีหน่วยวัดแบบนิ้วหรือมิลลิเมตร

- Mwait

เป็นโพรซิเยอร์ที่ใช้ช่วยเหลือในการเขียนโปรแกรมเพื่อช่วยในการรอรับค่าหรือคำสั่งของการทำงาน

- LinearInput

เป็นโพซิเยอรที่ใช้ในการอ่านค่าคำสั่ง และพารามิเตอร์ในคำสั่งการเคลื่อนที่แบบเส้นตรง ทั้งการเคลื่อนที่แบบเร็วและการเคลื่อนที่เชิงเส้นตรง (คำสั่ง G00 และ G01) เมื่อโปรแกรมหลักรับค่าคำสั่งและตรวจสอบแล้วว่าเป็นการเคลื่อนที่แบบเส้นตรง โปรแกรมจะอ่านค่าพารามิเตอร์ที่มีอยู่หลังคำสั่งนั้นเป็นตำแหน่งที่ต้องการให้มอเตอร์แต่ละตัวเดินไป แล้วกำหนดไว้ในตัวแปร เพื่อให้โปรแกรมหลักสามารถตรวจสอบและเทียบตำแหน่งปัจจุบันกับตัวแปรนี้ หากไม่เท่ากันเครื่องจะเดินไปยังตำแหน่งที่ตัวแปรกำหนดจนกว่าจะมีค่าเท่ากันตามวิธีการทำงานแต่ละอย่าง โพรซิเยอร์นี้ออกแบบให้สามารถรับค่าพารามิเตอร์ได้ทั้งตัวอักษรพิมพ์ใหญ่หรือเล็ก และไม่จำเป็นต้องลำดับมอเตอร์ หรือไม่ต้องกำหนดตำแหน่งมอเตอร์ให้ครบทั้ง 3 ตัว รวมทั้งยังสามารถอ่านค่าบรรทัดของโปรแกรม NC ในการทำงานโหมดอัตโนมัติได้ด้วย มีรูปแบบของโปรแกรมดังรูป 3.15

```
Procedure LinearInput;
 Begin
   Ano:=1; Free:=1;
   While free=1 do
     Begin
      if Auto=true then
      begin
       case Ano of
       1 : Read(NcFile,sp,Axis[Ano],Compos[Ano]);
       2 : Read(NcFile,Axis[Ano],Compos[Ano]);
       3 : Readln(NcFile,Axis[Ano],Compos[Ano])
```

```
 end; end
 else Begin
      case Ano of
       1 : Read(sp,Axis[Ano],Compos[Ano]);
       2 : Read(Axis[Ano],Compos[Ano]);
       3 : Readln(Axis[Ano],Compos[Ano])
       end;
      end;
case Upcase(axis[Ano]) of
'X' : Begin
      Nb:=1;
      If Millimeter=False then Position[Nb] := compos[Ano]*25.4 else Position[nb]:=compos[Ano];
      If AbsPos=False then StepCountNeed[Nb] := Round(Position[Nb]*200) + StepCountNow[Nb]
        else StepCountNeed[Nb] := Round(Position[Nb] * 200);
      Hz[Nb] := HzStart; SetTimer(Hz[Nb]);
    end;
'Y' : Begin
      Nb:=2;
      If Millimeter=False then Position[Nb] := compos[Ano]*25.4 else Position[nb]:=compos[Ano];
      If AbsPos=False then StepCountNeed[Nb] := Round(Position[Nb]*200) + StepCountNow[Nb]
        else StepCountNeed[Nb] := Round(Position[Nb] * 200);
      Hz[Nb] := HzStart; SetTimer(Hz[Nb]);
    end;
'Z' : Begin
      Nb:=3;
      If Millimeter=False then Position[Nb] := compos[Ano]*25.4 else Position[nb]:=compos[Ano];
      If AbsPos=False then StepCountNeed[Nb] := Round(Position[Nb]*200) + StepCountNow[Nb]
        else StepCountNeed[Nb] := Round(Position[Nb] * 200);
      Hz[Nb] := HzStart; SetTimer(Hz[Nb]);
    end;
else
   Free:=0;
end; {end Case Axis}
if Ano <>3 then
   Begin
      if auto=true then read(Ncfile,Sp) else Read (sp);
      if sp=#13 then
        Begin
          Free:=0;
          writeln('');
```

```
            if auto=true then read(Ncfile,Sp) else Read (sp);
          end;
        end
        else Free:=0;
      Inc(Ano);
      end;
end;
```

รูป 3.15 โพรซิเยอร์ LinearInput

- CirInput

เป็นโพซิเยอร์ที่มีการทำงานคล้ายคลึงกับการทำงานของโพรซิเยอร์ LinearInput โดยจะรับค่าพารามิเตอร์กำหนดตำแหน่งต่าง ๆ ของการเคลื่อนที่แบบเส้นโค้งแบบ 2 มิติ (คำสั่ง G02 และ G03) โดยมีพารามิเตอร์เพิ่มเติมคือตำแหน่งของจุดศูนย์กลาง แล้วนำมาคำนวณหารัศมี และจุดเปลี่ยนแกนหลักในการเคลื่อนที่ แล้วโปรแกรมหลักจะใช้ตัวแปรเหล่านี้ในการเคลื่อนที่ตามรูปแบบต่าง ๆ โดยมีรูปแบบของโปรแกรมดังรูป 3.16

```
Procedure CirInput;
 Begin
   if auto=true then
     begin
      read(NcFile,sp,Axis[1],Compos[1]);
      read(NcFile,sp,Axis[2],compos[2]);
      readln(NcFile,sp,coor[1],coorpos[1],sp,coor[2],coorpos[2]);
     end else
     readln(sp,Axis[1],Compos[1],sp,Axis[2],compos[2],sp,coor[1],coorpos[1],sp,coor[2],coorpos[2]);
   for Ano := 1 to 2 do
    Begin
      case Upcase(axis[Ano]) of
      'X' : Begin
            Nb:=1;
            If Millimeter=False then Position[Nb] := compos[Ano]*25.4 else Position[nb]:=compos[Ano];
            If AbsPos=False then StepCountNeed[Nb] := Round(Position[Nb]*200) + StepCountNow[Nb]
              else StepCountNeed[Nb] := Round(Position[Nb] * 200);
            Hz[Nb] := HzStart; SetTimer(Hz[Nb]);
           end;
       'Y' : Begin
            Nb:=2;
```

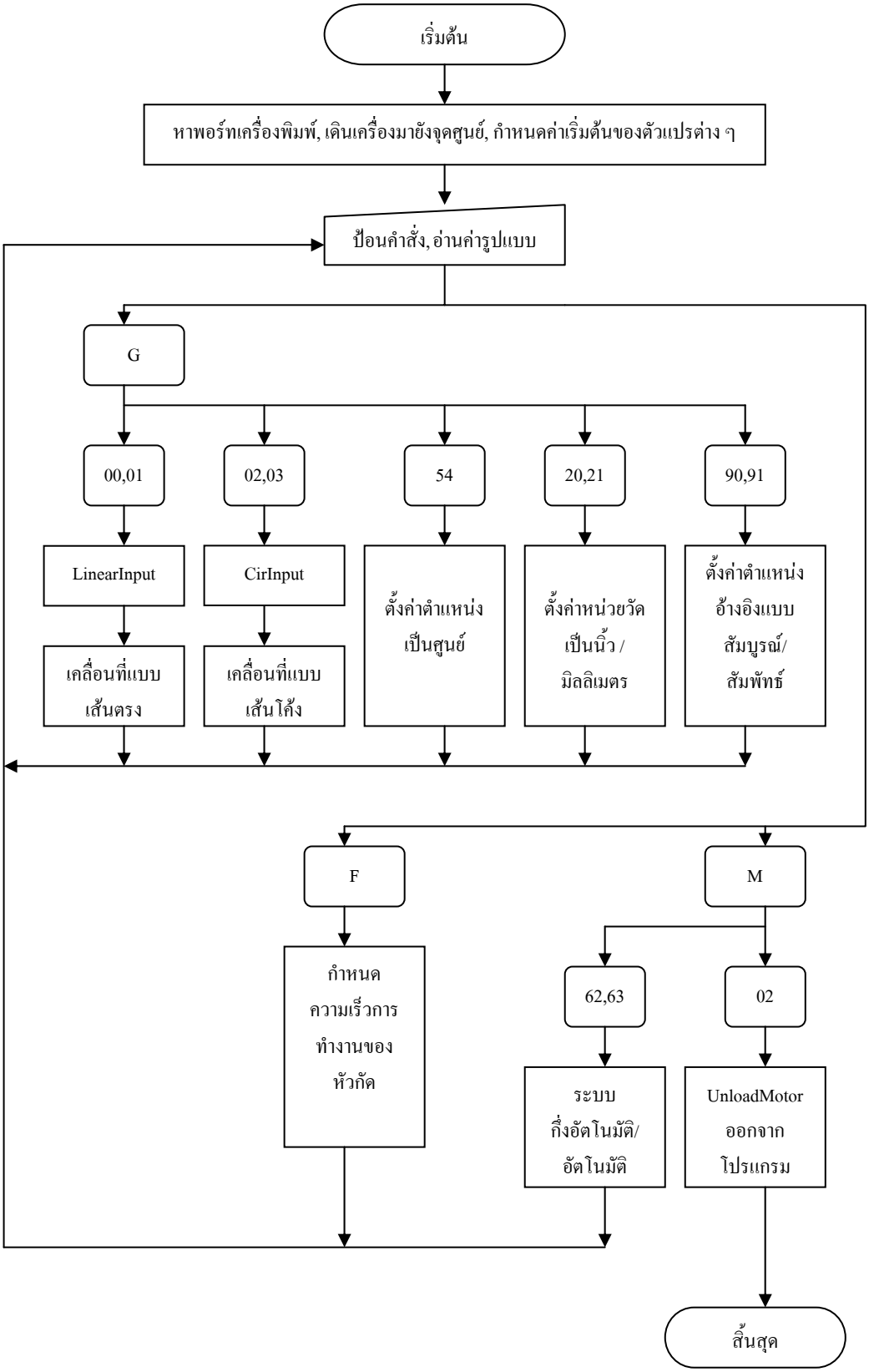
```
            If Millimeter=False then Position[Nb] := compos[Ano]*25.4 else Position[nb]:=compos[Ano];
            If AbsPos=False then StepCountNeed[Nb] := Round(Position[Nb]*200) + StepCountNow[Nb]
              else StepCountNeed[Nb] := Round(Position[Nb] * 200);
            Hz[Nb] := HzStart; SetTimer(Hz[Nb]);
          end;
      else
          writeln ('Other Motor');
      end; {end Case Axis}
      If Millimeter=False then Coorpos[Ano] := Coorpos[Ano]*25.4;
    end;{end loop ano}
    Gotoxy(3,7); write('Circular Coorporation Center');
    Gotoxy(5,8); If Millimeter=False then write('I= ',(coorpos[1]/25.4):3:1) else write('I= ',(coorpos[1]):3:1);
    Gotoxy(5,9); If Millimeter=False then write('J= ',(coorpos[2]/25.4):3:1) else write('J= ',(coorpos[2]):3:1);
    coorposstep[1]:=round(coorpos[1]*200); coorposstep[2]:=round(coorpos[2]*200);
    If AbsPos=False then
      Begin
       coordiff[1]:=abs(coorposstep[1])/200;
       coordiff[2]:=abs(coorposstep[2])/200;
      end else
        Begin
          coordiff[1]:=abs(Stepcountnow[1]-coorposstep[1])/200;
          coordiff[2]:=abs(Stepcountnow[2]-coorposstep[2])/200;
      end;
    coordiff[3]:=(coordiff[1]*coordiff[1]);
    coordiff[4]:=(coordiff[2]*coordiff[2]);
    Radius := sqrt(coordiff[3]+coordiff[4]);
    Radiusstep:=round(radius*200);
    Gotoxy(3,11); write('Radius= ',radius:3:2);
  End;
```

รูป 3.16 โพรซิเยอร์ CirInput

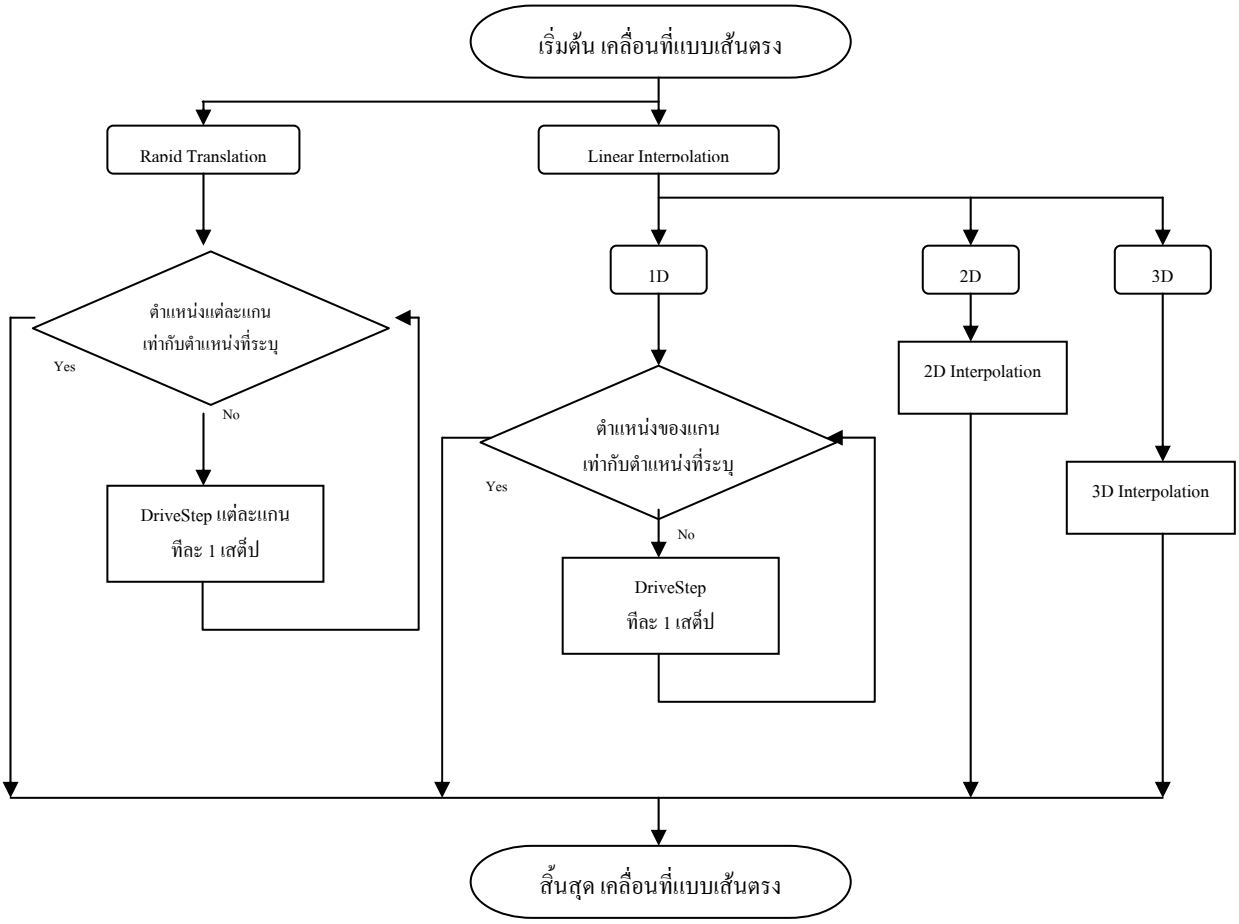
ส่วนสุดท้ายที่เป็นส่วนหลักของโปรแกรม คือ โปรแกรมหลัก โดยมีรูปแบบการทำงานเริ่มจากการรับคำสั่งจากแป้นพิมพ์ทีละคำสั่งหรือรับคำสั่งจากโปรแกรม NC แล้วจำแนกคำสั่งเพื่อรับค่าพารามิเตอร์ประกอบคำสั่งหรือตั้งค่าการทำงานของโปรแกรม การทำงานหลักของโปรแกรมคือการเดินเครื่องกัดในแนวเส้นตรงและการเดินในแนวเส้นโค้ง หลังจากทำงานในแต่ละคำสั่งเรียบร้อยแล้ว ก็จะหมุนวนกลับมารอรับคำสั่งใหม่ จนกว่าจะมีคำสั่งในการเลิกใช้ โปรแกรมต้นฉบับโปรแกรม (Motor.PAS) ที่เขียนขึ้นจาก Turbo Pascal สำหรับการทำงานโดยใช้

คอมพิวเตอร์ส่วนบุคคลเป็นตัวประมวลผลได้บรรจุไว้ในแผ่นซีดีรอม ซึ่งในส่วนนี้จะได้อธิบายวิธีการทำงานของโปรแกรมหลักในส่วนต่าง ๆ ด้วยแผนผังความคิด ในส่วนประกาศตัวแปรและส่วนรับข้อมูล โปรแกรมจะค้นหาพอร์ทเครื่องพิมพ์ กำหนดความเร็ว เดินแท่นงานมายังจุดศูนย์และตั้งค่าเริ่มต้นของตัวแปรต่าง ๆ หลังจากนั้นจะรอรับคำสั่ง เริ่มแรกโปรแกรมได้กำหนดให้มีโหมดการทำงานในระบบกึ่งอัตโนมัติ มีหน่วยวัดเป็นมิลลิเมตร มีการอ้างอิงตำแหน่งแบบสัมบูรณ์ มีความเร็วการทำงานเป็น 60 มิลลิเมตร/นาที เมื่อทำการป้อนคำสั่งแบบ G-code แล้ว โปรแกรมจะเก็บค่าตัวแปรตัวแรกเพื่อแยกว่าเป็นคำสั่งแบบ G, M หรือ F แล้วเลื่อนไปเก็บค่าหมายเลขกำกับคำสั่งสำหรับคำสั่งแบบ G และ M หรือเก็บค่าความเร็วหลังคำสั่ง F สำหรับคำสั่ง G หากหมายเลขกำกับคำสั่งเป็น 00 หรือ 01 โปรแกรมจะเรียกใช้โพรซีเยอร์ LinearInput หรือหากหมายเลขเป็น 02 หรือ 03 โปรแกรมจะเรียกใช้โพรซีเยอร์ CirInput เพื่อเก็บค่าพารามิเตอร์ในการเคลื่อนที่ที่จำเป็นสำหรับรูปแบบคำสั่งต่าง ๆ หลังจากนั้นจะทำการคำนวณเพื่อสั่งให้มอเตอร์เคลื่อนที่ในรูปแบบต่าง ๆ ตามรูปแบบของคำสั่ง โดยโปรแกรมหลักส่วนแรกและส่วนรับข้อมูลจะมีแผนผังความคิดและรูปแบบการทำงานดังรูป 3.17

ส่วนที่สำคัญส่วนที่สองของโปรแกรมหลัก คือ การเดินเครื่องกัดในแนวเส้นตรง ซึ่งโปรแกรมจะทำงานหลังจากที่รับค่าพารามิเตอร์ของตำแหน่งที่ระบุให้โปรแกรมเดินแท่นงานไป เมื่อรับค่าตำแหน่งจากโพรซิเยอร์ LinearInput แล้ว โปรแกรมจะแยกว่าต้องการเดินแท่นงานแบบรวดเร็วหรือเดินให้สัมพันธ์กันเชิงเส้นจากคำสั่ง G00 หรือ G01 ตามลำดับ หากโปรแกรมพบว่ารูปแบบคำสั่งเป็นการสั่งให้เดินแท่นงานแบบรวดเร็ว โปรแกรมจะคำนวณหาจำนวนความยาวที่แตกต่างกัน แล้วคำนวณออกมาเป็นจำนวนเสต็ปสำหรับมอเตอร์แต่ละตัว หากพบว่าจำนวนเสต็ปที่ต้องเดินในแต่ละแกนมากกว่า 0 โปรแกรมจะขับมอเตอร์ทีละ 1 เสต็ปและลดจำนวนสเต็ป แล้วเปลี่ยนไปทำงานเช่นเดิมสำหรับมอเตอร์ตัวอื่น หากมอเตอร์ตัวใดมีจำนวนเสต็ปที่ต้องทำงานลดลงเหลือ 0 แล้ว โปรแกรมจะข้ามไม่ทำการขับมอเตอร์ตัวนั้น ดังนั้นการทำงานแบบรวดเร็วนี้ การขับมอเตอร์จะไม่คำนึงถึงความสัมพันธ์ของตำแหน่งแท่นงานในแต่ละแกน แต่จะทำงานด้วยความเร็วมากที่สุดที่สามารถทำได้เพื่อเดินแท่นงานในแต่ละแกนไปยังตำแหน่งที่ระบุให้เร็วที่สุด การทำงานเช่นนี้มีประโยชน์ในการเคลื่อนแท่นงานไปยังตำแหน่งที่ระบุด้วยความเร็วโดยยังไม่ทำการกัดชิ้นงาน หรือเพื่อเลื่อนหัวกัดขึ้น-ลง ไปยังตำแหน่งที่เตรียมทำการกัดชิ้นงาน การทำงานเช่นนี้มีแผนผังความคิดการทำงานของโปรแกรมดังรูป 3.18 หากโปรแกรมพบว่ารูปแบบคำสั่งเป็นการสั่งให้เดินแท่นงานแบบมีความสัมพันธ์เชิงเส้น โปรแกรมตรวจสอบว่าต้องทำงานพร้อมกันกี่แกน หากพบว่าทำงานเพียงแกนเดียว ก็จะสั่งให้ขับมอเตอร์ด้วยจำนวนเสต็ปที่แตกต่างกันกับกำแหน่งปัจจุบันจนกระทั่งมีตำแหน่งตามที่ผู้ใช้ระบุ ดังรูป 3.18 หากพบว่าต้องทำงาน 2 แกน โปรแกรมจะตรวจสอบว่าแกนใดที่ต้องขับมอเตอร์เป็นจำนวนเสต็ปน้อยกว่า



รูป 3.17 แผนผังความคิดของการทำงานของโปรแกรมโดยรวม

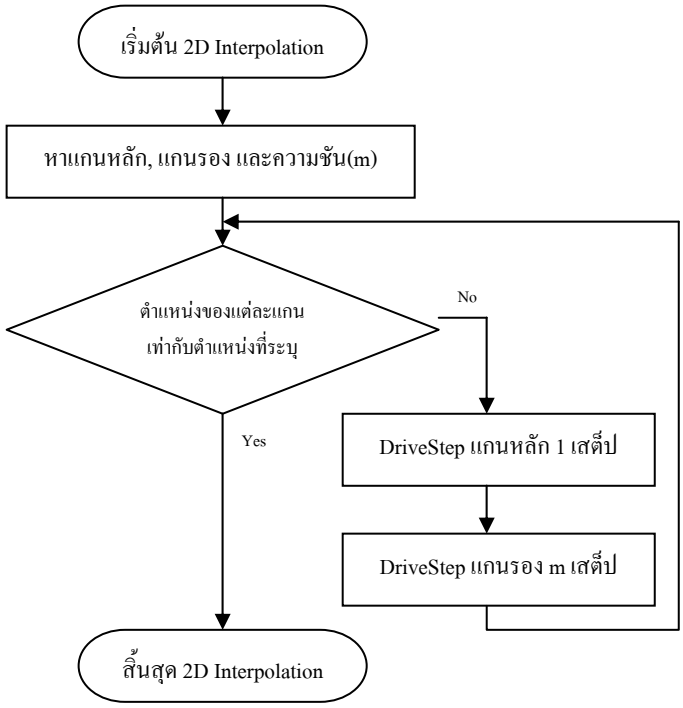


รูป 3.18 แผนผังความคิดของการทำงานของการเดินเครื่องกัดแบบรวดเร็วและแบบเชิงเส้น 1 มิติ

ซึ่งในที่นี้จะเรียกว่าเป็น “แกนหลัก” แล้วหาค่าความชันด้วยการหารจำนวนเสต็ปในแกนที่ต้องขับมากกว่า ซึ่งจะเรียกว่าเป็น “แกนรอง” ด้วยจำนวนเสต็ปในแกนหลัก เพื่อที่จะทราบว่า หากขับมอเตอร์ในแกนหลักไป 1 เสต็ป แล้วจะต้องเดินมอเตอร์ในแกนรองไปกี่เสต็ป เพื่อให้การเคลื่อนที่มีความสัมพันธ์กันในแนวเส้นตรง หลังจากนั้นจะเดินแกนหลัก 1 เสต็ปและต่อด้วยแกนรองเท่าจำนวนความชันที่คำนวณหาได้ (หากเมื่อหารแล้วได้ค่าเป็นจุดทศนิยม ให้ทำการปัดเศษให้เป็นจำนวนเต็ม) โปรแกรมจะทำเป็นวงรอบเช่นนี้ไปจนกว่าตำแหน่งมอเตอร์ในแกนหลักและแกนรองจะเท่ากับตำแหน่งที่ผู้ใช้ระบุ ดังรูป 3.19

รูปแบบสุดท้ายคือการทำงานเชิงเส้นตรงแบบ 3 มิติ มีการขับมอเตอร์ทั้ง 3 แกน โปรแกรมจะคำนวณหาจำนวนเสต็ปในแต่ละแกนที่ต้องขับมอเตอร์ไป แล้วนำมาหาความชันหลัก ด้วยการหารจำนวนเสต็ปในแกนหลักด้วยจำนวนเสต็ปในแกนที่มากเป็นอันดับ 2 หรือในที่นี้เรียกว่า “แกนรองที่ 1” และหาความชันรองด้วยการหารจำนวนเสต็ปในแกนรองที่ 1 ด้วยจำนวนเสต็ปในแกนที่มาก

ที่สุด ซึ่งในที่นี้จะเรียกว่า “แกนรองที่ 2” หลังจากนั้นจะขับมอเตอร์ในแกนหลัก 1 เสต็ป และขับมอเตอร์มอเตอร์ในแกนรองที่ 1 จำนวน 1 เสต็ป แล้วทำการเดินมอเตอร์ในแกนรองที่ 2 เป็นจำนวน

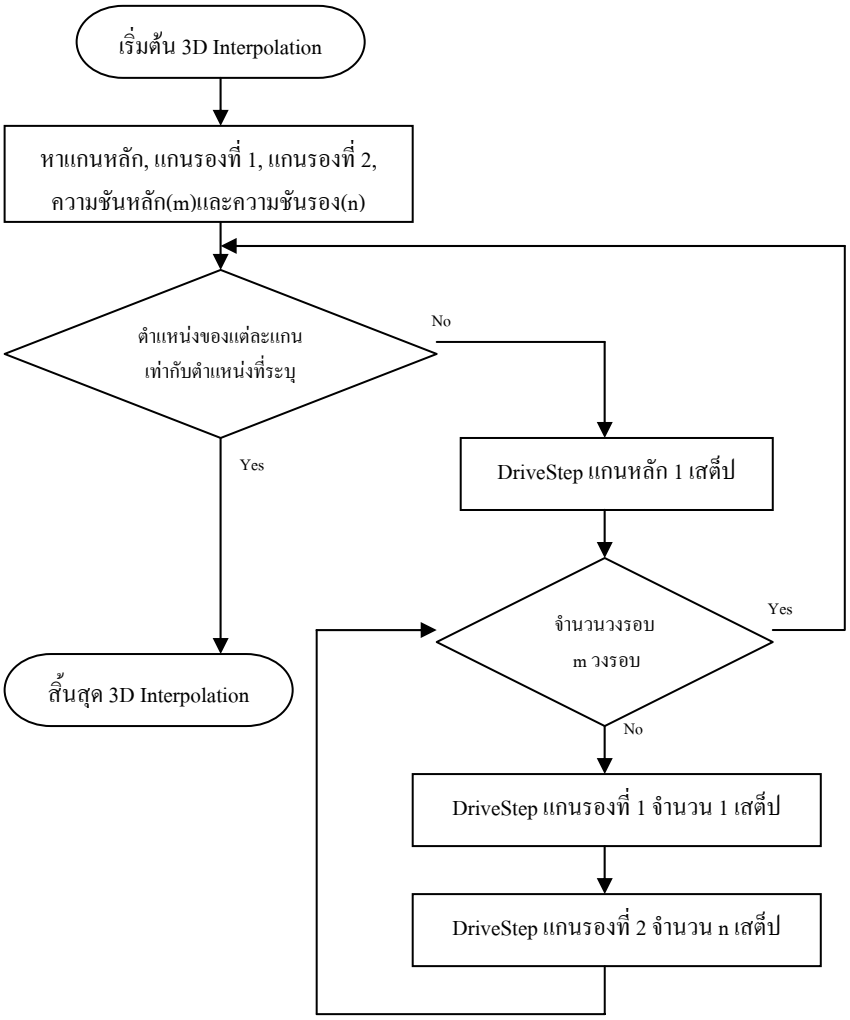


รูป 3.19 แผนผังความคิดของการทำงานของการเดินเครื่องกัดแบบเชิงเส้น 2 มิติ

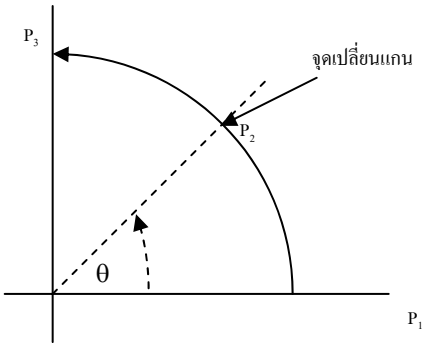
เท่ากับความชันรอง แล้วกลับมาขับมอเตอร์ในแกนรองที่ 1 และ 2 อีกเช่นเดิมเป็นวงรอบจนมีจำนวนวงรอบเท่าความชันหลัก แล้วจึงกลับมาขับมอเตอร์ในแกนหลักอีก 1 เสต็ป แล้วไปขับวงรอบของความชันรองอีกครั้ง ทำเป็นวงรอบเช่นนี้จนกระทั่งตำแหน่งของมอเตอร์ในแกนต่าง ๆ มีค่าเท่ากับตำแหน่งที่ผู้ใช้ระบุ ทำให้สามารถเดินหัวกัดเป็นแบบ 3 มิติได้ แผนผังความคิดสำหรับการทำงานโปรแกรมในการเคลื่อนแท่นงานในแนวเส้นตรงแสดงดังรูป 3.20

ส่วนสำคัญส่วนสุดท้ายของโปรแกรมหลักคือ การเดินเครื่องกันในแนวเส้นโค้ง ที่สามารถเดินเครื่องกัดแนวเส้นโค้งตามเข็มหรือทวนเข็มนาฬิกา ตามรูปแบบคำสั่ง G02 และ G03 ตามลำดับ การทำงานในแนวเส้นโค้งนี้จะสามารถทำงานได้เฉพาะใน 2 มิติ คือ ทำงานครั้งละ 2 แกนมอเตอร์เท่านั้น เมื่อโปรแกรมรับคำสั่งการทำงานแบบเส้นโค้งแล้ว จะเรียนใช้โพรซีเยอร์ CirInput เพื่อเก็บค่าพารามิเตอร์ตำแหน่งของจุดสุดท้ายของเส้นทางเดิน รวมทั้งจุดศูนย์กลางของความโค้ง แล้วนำมาหาค่ารัศมีและจุดเปลี่ยนแกน ดังรูป 3.21 จุดเปลี่ยนแกนนี้มีประโยชน์เมื่อหากคิดว่าส่วนของเส้นโค้งนั้นประกอบขึ้นจากเส้นตรงที่มีความชันต่าง ๆ จากเส้นที่มีความชันเป็นอนันต์ที่จุด P_1 และค่อย ๆ ลดลงมาจนกระทั่งมีความชันเป็น 1 ที่จุด P_2 ซึ่งมีมุม θ เปลี่ยนไป 45 องศา

แล้วจึงมีความชันเป็นเศษส่วน และลดลงเรื่อย ๆ จนกระทั่งมีความชันเท่ากับ 0 ที่จุด P_3 จุดเปลี่ยนแกนคือจุด P_2 นั่นเอง เพราะโปรแกรมจะเปลี่ยนแกนหลักในการขับมอเตอร์ทั้งสองที่จุดนี้



รูป 3.20 แผนผังความคิดของการทำงานของการเดินเครื่องกัดแบบเชิงเส้น 3 มิติ



รูป 3.21 จุดเปลี่ยนแกนของการเคลื่อนที่แบบเส้นโค้ง

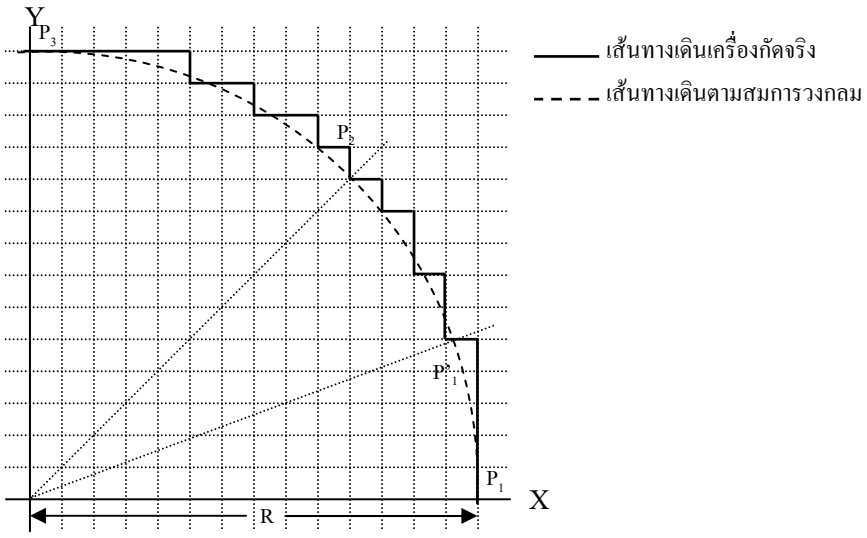
การเคลื่อนที่ของแท่นงานแบบเส้นโค้ง ไม่ว่าจะเป็นการเคลื่อนที่ตามเข็มนาฬิกาหรือทวนเข็มนาฬิกา ล้วนมีพื้นฐานมาจากการเคลื่อนที่แบบเส้นตรงในระยะสั้น ๆ ที่มีความชันเปลี่ยนแปลงอยู่ตลอดเวลา ทำให้ต้องมีการคำนวณหาความชันทุกครั้งที่มีการเคลื่อนที่ของแกนหลัก โดยมีรูปแบบเส้นทางการขับมอเตอร์ดังรูป 3.22 เมื่อต้องเดินแท่นงานจากจุด P_1 ไปยังจุด P_3 ในช่วงแรก แกนหลักในการขับมอเตอร์คือแกน X โปรแกรมจะคำนวณตำแหน่งของจุด P'_1 จากสมการ3.1

$$x^2 + y^2 = R^2 \qquad \text{.....(3.1)}$$

โดยที่ x คือ ตำแหน่งของจุดตามแนวแกน X

y คือ ตำแหน่งของจุดตามแนวแกน Y

R คือ รัศมี



รูป 3.22 รูปแบบเส้นทางการขับมอเตอร์ในการเคลื่อนที่แบบเส้นโค้ง

โดยที่จุด P'_1 จะมีองค์ประกอบความยาวในแนวแกนหลักลดลง 1 เสต็ป แล้วจะทำการเดินแบบเส้นตรงไปยังจุด P'_1 ด้วยการคำนวณหาความชัน m โดยทำการเดินแกนรองเป็นจำนวน m เสต็ปก่อน แล้วจึงเดินแกนหลัก 1 เสต็ปตาม หลังจากนั้นจะคำนวณหาจุดต่อ ๆ ไปและขับแท่นงานไปจนกระทั่งเป็นจุดเดียวกันกับจุดเปลี่ยนแกน P_2 จึงเปลี่ยนแกนหลักเป็นแกน Y แล้วกระทำด้วยกระบวนการเดียวกันจนกระทั่งไปเสร็จสิ้นที่จุด P_3 การเคลื่อนที่เป็นเส้นโค้งในส่วนของวงกลมอื่น ๆ สามารถทำได้โดยการเลือกแกนหลักและทิศทางการขับมอเตอร์ที่ถูกต้อง จนกระทั่งเดินแท่นงานเป็นรูปวงกลม

การขับแท่นงานเป็นเส้นโค้งที่มีจุดเริ่มต้นที่ตำแหน่งใด ๆ เมื่อโปรแกรมได้รับค่าตำแหน่งของจุดศูนย์กลางและรัศมีแล้ว โปรแกรมจะทำการคำนวณหาตำแหน่งทุกตำแหน่งที่เครื่องกัดจะสามารถเดินไปบนวงกลมตามทิศทางที่กำหนด เมื่อค่าที่คำนวณได้มีค่าเท่ากับตำแหน่งปัจจุบันของหัวกัด โปรแกรมจะสั่งให้มอเตอร์ขับแท่นงานตามตำแหน่งบนส่วนของวงกลมนั้นจนกระทั่งตำแหน่งที่แท่นงานเดินไปมีค่าเท่ากับตำแหน่งที่ผู้ใช้ระบุ มอเตอร์จึงจะหยุดทำงาน การทำงานเช่นนี้ ค่าตำแหน่งปัจจุบันและค่าตำแหน่งสิ้นสุดที่ผู้ใช้ระบุต้องมีค่าที่ถูกต้องตามที่โปรแกรมได้คำนวณจากตำแหน่งของจุดศูนย์กลางและรัศมีที่กำหนดเท่านั้น

3.2.2 การติดตั้งระบบโดยใช้เครื่องคอมพิวเตอร์แบบแผงวงจรเดี่ยวเป็นหน่วยประมวลผล

เมื่อทำการติดตั้งและทดสอบการทำงานของระบบที่มีคอมพิวเตอร์แบบแผงวงจรเดี่ยวเป็นหน่วยประมวลแล้ว จึงได้ทำการเปลี่ยนหน่วยประมวลผลเป็นคอมพิวเตอร์แบบแผงวงจรเดี่ยวชนิด PC/104 Mity-Myte module แต่เนื่องจากคอมพิวเตอร์ชนิดนี้ไม่มีพอร์ทเครื่องพิมพ์เช่นเดียวกับคอมพิวเตอร์ส่วนบุคคล แต่ได้จัดให้มีพอร์ทอเนกประสงค์ 16 บิตเพื่อใช้รับเข้า / ส่งออกข้อมูลหรือสัญญาณดิจิตอล รูปแบบของการทำงานของพอร์ทอเนกประสงค์จะมีความแตกต่างกับพอร์ทเครื่องพิมพ์ จึงต้องมีการปรับปรุงการติดตั้งฮาร์ดแวร์ให้เข้ากับการทำงานเดิมของแผงวงจรควบคุมเสต็ปปิ้งมอเตอร์ รวมทั้งซอฟแวร์เพื่อกำหนดรูปแบบการรับเข้า / ส่งออกข้อมูลและสัญญาณดิจิตอล

3.2.2.1 การติดตั้งระบบควบคุมเสต็ปปิ้งมอเตอร์เข้ากับพอร์ทอเนกประสงค์

เนื่องจากผู้ผลิตคอมพิวเตอร์แบบแผงวงจรเดี่ยวได้จัดให้มีพอร์ทอเนกประสงค์เพื่อรับเข้า / ส่งออกข้อมูลและสัญญาณดิจิตอล จึงต้องทำการติดตั้งและเชื่อต่อสายสัญญาณให้เข้ากับแผงวงจรควบคุมเสต็ปปิ้งมอเตอร์และสายสัญญาณเดิมที่เป็นแบบเดียวกับพอร์ทเครื่องพิมพ์ โดยมีรูปแบบการเชื่อมต่อดังตาราง 3.5

ตาราง 3.5 การเชื่อมพอร์ทอเนกประสงค์กับสายสัญญาณแบบพอร์ทเครื่องพิมพ์

พอร์ทอเนกประสงค์		พอร์ทเครื่องพิมพ์
Pin No.	Description	Pin No.
1	GND	18
2	Vcc	
3	GP0	2
4	GP8	1
5	GP1	3
6	GP9	14
7	GP2	4

ตาราง 3.5 (ต่อ) การเชื่อมพอร์ทอเนกประสงค์กับสายสัญญาณแบบพอร์ทเครื่องพิมพ์

พอร์ทอเนกประสงค์		พอร์ทเครื่องพิมพ์
Pin No.	Description	Pin No.
8	GP10	16
9	GP3	5
10	GP11	17
11	GP4	6
12	GP12	11
13	GP5	7
14	GP13	12
15	GP6	8
16	GP14	13
17	GP7	9
18	GP15	10
19	Vcc	
20	GND	19

เพื่อความสะดวกในการเชื่อมต่อ ได้เลือกใช้ขั้วสัญญาณชนิด 25 ขา (ตัวเมีย) แบบเดียวกับพอร์ทเครื่องพิมพ์ที่ติดกับตัวเครื่องคอมพิวเตอร์ส่วนบุคคล ทำให้สามารถใช้สายสัญญาณเส้นเดิมที่ใช้เชื่อมต่อแผงวงจรควบคุมเสต็ปปิ้งมอเตอร์กับพอร์ทเครื่องพิมพ์ของคอมพิวเตอร์ส่วนบุคคลได้

3.2.2.2 การปรับปรุงโปรแกรมการทำงาน

เนื่องจากคอมพิวเตอร์แบบแผงวงจรเดี่ยวมีรูปแบบการรับเข้า / ส่งออกข้อมูลและสัญญาณดิจิตอลแตกต่างจากคอมพิวเตอร์ส่วนบุคคล แต่มีแนวคิดและกระบวนการการทำงานที่เหมือนกัน โปรแกรมที่ใช้ควบคุมการทำงานของเครื่องกัดสำหรับเครื่องคอมพิวเตอร์แบบแผงวงจรเดี่ยวจึงสามารถนำโปรแกรมสำหรับเครื่องคอมพิวเตอร์ส่วนบุคคลมาใช้ปรับปรุงได้ ซึ่งในที่นี้จะได้กล่าวถึงเฉพาะส่วนที่มีการปรับปรุงเท่านั้น ต้นฉบับโปรแกรม (MotorSBC.PAS) ที่เขียนขึ้นจาก Turbo Pascal สำหรับการทำงานโดยใช้คอมพิวเตอร์แบบแผงวงจรเดี่ยวเป็นตัวประมวลผลได้บรรจุไว้ในแผ่นซีดีรอม และรูปแบบและวิธีการใช้พอร์ทอเนกประสงค์สำหรับคอมพิวเตอร์แบบแผงวงจรเดี่ยว PC/104 Mity-Mite Module แสดงดังในภาคผนวก ค

- ส่วนหัวและส่วนประกาศตัวแปร

ส่วนหัวและส่วนประกาศตัวแปรของโปรแกรมจะมีรูปแบบเดียวกันกับโปรแกรมของคอมพิวเตอร์ส่วนบุคคล แต่จะเพิ่มโพรซีเยอร์ SetPort เพื่อเป็นส่วนช่วยในการกำหนดหมายเลขของพอร์ทและค่าที่ต้องการส่งไปยังพอร์ทนั้น รวมทั้งทำการปรับปรุงส่วนของการกำหนดค่าตรรกะ

สำหรับตัวแปรที่รับค่าจากลิมิตสวิทซ์ ซึ่งข้อมูลที่รับเข้าจากลิมิตสวิทซ์นั้น เมื่อวงจรของลิมิตสวิทซ์ปิด ค่าสัญญาณดิจิตอลที่บิตนั้นจะมีค่าเป็น 0 จึงต้องเปรียบเทียบข้อมูลทั้ง 8 บิต จากจำนวน 239 (Binary 11101111) สำหรับลิมิตสวิทซ์ของมอเตอร์แกน X ซึ่งติดตั้งอยู่บนพอร์ทอเนกประสงค์หมายเลข GP12, จำนวน 223 (Binary 11011111) สำหรับลิมิตสวิทซ์ของมอเตอร์แกน Y ซึ่งติดตั้งอยู่บนพอร์ทอเนกประสงค์หมายเลข GP13 และจำนวน 191 (Binary 10111111) สำหรับลิมิตสวิทซ์ของมอเตอร์แกน Z ซึ่งติดตั้งอยู่บนพอร์ทอเนกประสงค์หมายเลข GP14 ซึ่งมีรูปแบบโปรแกรมดังรูป 3.23

```
Program MOTORSBC;
{Program uses Turbo Pascal V 4.0 Copmpiler}
{Modification Date 29 APR 2004}
{Modified for Single Board Computer}
uses crt;
Var Phase1,Phase2,Phase3,K,M,I,T,Na,Nb,Nc,checkNb : Integer;
   StepCountNow,StepCountNeed,StepDiff      : Array [1..3] of LongInt;
   compos,coorpos, Hz,Position              : Array [1..3] of Real;
   coorposstep,cirrun                       : Array [1..2] of integer;
   coordiff                                 : Array [1..4] of real;
   Order,Pointer,PortData                   : Array [1..3] of integer;
   CheckPo                                  : real;
   DataPort                                 : Array [1..2] of Integer;
   Niden,Giden,sp                           : Char;
   Code,Ano,Gmode,DrX,DrY,DrZ               : Integer;
   CDR,cdr1,cdr2,x,y,xst,free               : Integer;
   MforX,MforY,MforZ,step,Dryst             : Integer;
   radiusstep,cngpnt                        : Integer;
   Radius,e                                 : real;
   Axis,coor                                : Array[1..3] of char;
   Filename                                 : Array[1..12] of char;
   a,b,c,d,g                                : Real;
   f,Buff,Quad,Ncode                        : Integer;
   ch                                       : Array[1..3] of Boolean;
   Millimeter,AbsPos,LoopCheck: Boolean;
   Auto,check,circheck                      : Boolean;
   Readfile                                 : String[12];
   NcFile                                   : text;

Const HzMax : Real = 300;
   HzStart : Real = 200;
```

```
Procedure Setport(index,data:integer);
     Begin
    Port[$22] := index;
    Port[$23] := data;
 end;

Function Switch1 : Boolean;
 Begin
   Setport($4F,$00);
   Port[$22] := $4C;
   portdata[1]:= port[$23];
   If (portdata[1] =239) then Switch1 := True
     Else Switch1 := False;
   SetPort($4F,$FF);
 End;

Function Switch2 : Boolean;
 Begin
   Setport($4F,$00);
   Port[$22] := $4C;
   portdata[2]:= port[$23];
   If (portdata[2] =223) then Switch2 := True
     Else Switch2 := False;
   SetPort($4F,$FF);
 End;

Function Switch3 : Boolean;
 Begin
   Setport($4F,$00);
   Port[$22] := $4C;
   portdata[3]:= port[$23];
   If (portdata[3] =191) then Switch3 := True
     Else Switch3 := False;
   SetPort($4F,$FF);
 End;
```

รูป 3.23 รูปแบบของส่วนหัวและส่วนประกาศตัวแปร

สำหรับโพรซีเยอร์ SetPort นั้น จะรับเอาเลขจำนวนเต็ม 2 จำนวน จำนวนแรกคือหมายเลขของพอร์ทที่จะรับเข้า/ส่งออกข้อมูลโดยจะไปกำหนดในพอร์ทหมายเลข 22h ส่วนตัวเลขที่ 2 คือ

ค่าที่กำหนดลักษณะการทำงานของพอร์ทนั้น ๆ โดยจะไปเก็บค่านั้นไว้ในพอร์ทหมายเลข 23h การใช้งานพอร์ทอเนกประสงค์นี้ เมื่อต้องการส่งข้อมูลหรืออ่านค่าข้อมูลจากพอร์ท ต้องทำการเปิดพอร์ทก่อน และทำการปิดพอร์ทเมื่อเลิกใช้งาน โดยการส่งค่าไปยังพอร์ทหมายเลข 13h และส่งค่า 00h เพื่อเปิดพอร์ท และค่า C5h เพื่อปิดพอร์ท

การกำหนดให้มีการรับเข้าหรือส่งออกข้อมูลสำหรับพอร์ทอเนกประสงค์กระทำได้ครั้งละ 8 บิต หรือ 1 ไบต์ เมื่อต้องการกำหนดการทำงานของพอร์ทอเนกประสงค์หมายเลข GP 7-0 ต้องทำการส่งค่าไปยังพอร์ทหมายเลข 4Eh โดยส่งค่า 00h สำหรับกำหนดให้พอร์ทรับเข้าข้อมูล และค่า FFh สำหรับกำหนดให้พอร์ทส่งออกข้อมูล และเมื่อต้องการกำหนดการทำงานของพอร์ทอเนกประสงค์หมายเลข GP 15-8 ต้องทำการส่งค่าไปยังพอร์ทหมายเลข 4Fh โดยส่งค่า 00h สำหรับกำหนดให้พอร์ทรับเข้าข้อมูล และค่า FFh สำหรับกำหนดให้พอร์ทส่งออกข้อมูลเช่นกัน

- DriveStep

เนื่องจากมีการเปลี่ยนรูปแบบการสั่งงานการรับเข้า/ส่งออกข้อมูล และรูปแบบการทำงานของพอร์ท โพรซีเยอร์ DriveStep จึงต้องทำการปรับปรุงข้อมูลที่ใช้ในการสั่งงานมอเตอร์แต่ละตัว ดังรูป 3.24

```
Procedure DriveStep(Nba,Direction : Integer);
 Const Sphase1 : Array[1..4] of byte = ($0C,$06,$03,$09);
       Sphase2 : Array[1..4] of byte = ($C0,$60,$30,$90);
       Sphase3 : Array[1..4] of byte = ($0C,$06,$03,$09);
 {Stepper drive low active drive pattern 1100 0110 0011 1001}
 Begin
  Case Nba of
  1 : If Direction > 0 then
        Begin
         Inc(Phase2); If Phase2 > 4 then Phase2 := 1;
         DataPort[1] := DataPort[1] and $0F or Sphase2[Phase2];
         Setport ($47,Dataport[1]);
        End
      Else
        Begin
          Dec(Phase2); If Phase2 < 1 then Phase2 := 4;
          DataPort[1] := DataPort[1] and $0F or Sphase2[Phase2];
          Setport ($47,Dataport[1]);
        End;
  2  : If Direction > 0 then
        Begin
         Inc(Phase1); If Phase1 > 4 then Phase1 := 1;
```

```
        DataPort[1] := DataPort[1] and $F0 or Sphase1[Phase1];
        Setport ($47,Dataport[1]);
       End
      Else
        Begin
         Dec(Phase1); If Phase1 < 1 then Phase1 := 4;
         DataPort[1] := DataPort[1] and $F0 or Sphase1[Phase1];
         Setport ($47,Dataport[1]);
        End;
  3 : If Direction > 0 then
       Begin
        Dec(Phase3); If Phase3 < 1 then Phase3 := 4;
        DataPort[2] := DataPort[2] and $F0 or Sphase3[Phase3];
        Setport ($4D,Dataport[2]);
       End
      Else
       Begin
        Inc(Phase3); If Phase3 > 4 then Phase3 := 1;
        DataPort[2] := DataPort[2] and $F0 or Sphase3[Phase3];
        Setport ($4D,Dataport[2]);
       End;
      End; {Case}
End;
```

รูป 3.24 โพรซีเยอร์ DriveStep

จะเห็นได้ว่า อาเรย์ข้อมูลสำหรับมอเตอร์หมายเลข 3 (Z) จะมีความแตกต่างกันกับโปรแกรมสำหรับคอมพิวเตอร์ส่วนบุคคล เนื่องจากในคอมพิวเตอร์ส่วนบุคคล ข้อมูลบางบิตจะมีอินเวอร์ส แต่สำหรับคอมพิวเตอร์แบบแผงวงจรเดี่ยวนั้น ไม่มีการอินเวอร์สข้อมูลสำหรับทุกบิต จึงสามารถสั่งงานโดยตรงได้ สำหรับมอเตอร์ตัวที่ 1 (Y) จะใช้พอร์ทอเนกประสงค์หมายเลข GP3-0 มอเตอร์ตัวที่ 2 (X) จะใช้พอร์ทอเนกประสงค์หมายเลข GP7-4 และมอเตอร์ตัวที่ 3 (Z) จะใช้พอร์ทอเนกประสงค์หมายเลข GP11-8

การส่งข้อมูลไปยังพอร์ทอเนกประสงค์หมายเลข GP7-0 จะต้องส่งค่า 47h ไปยังพอร์ทหมายเลข 22h และข้อมูล 1 ไบต์ไปยังพอร์ทหมายเลข 23h หากต้องการรับค่าข้อมูลจากพอร์ทอเนกประสงค์หมายเลข GP7-0 ทำได้โดยการส่งค่า 46h ไปยังพอร์ทหมายเลข 22h ก่อนเพื่อกำหนดให้คอมพิวเตอร์อ่านค่านั้นแล้วไปเก็บค่าไว้ในพอร์ทหมายเลข 23h และอ่านค่าจากพอร์ทนั้น

การส่งข้อมูลไปยังพอร์ทอเนกประสงค์หมายเลข GP15-8 จะต้องส่งค่า 4Dh ไปยังพอร์ทหมายเลข 22h และข้อมูล 1 ไบต์ไปยังพอร์ทหมายเลข 23h หากต้องการรับค่าข้อมูลจากพอร์ทอเนกประสงค์หมายเลข GP15-8 ทำได้โดยการส่งค่า 4Ch ไปยังพอร์ทหมายเลข 22h ก่อนเพื่อกำหนดให้คอมพิวเตอร์อ่านค่านั้นแล้วไปเก็บค่าไว้ในพอร์ทหมายเลข 23h และอ่านค่าจากพอร์ทนั้น

- GetReady

เพื่อจะให้เครื่องคอมพิวเตอร์พร้อมที่จะทำงาน และมีการเตรียมการเช่นเดียวกันกับโปรแกรมจากเครื่องคอมพิวเตอร์ส่วนบุคคล จึงทำการปรับปรุงโปรแกรมดังรูป 3.25

```
Procedure GetReady;
 Begin
   Setport($13,$C5);
   Phase1 := 1; Phase2 := 1; Phase3 := 1;
   SetPort($4E,$FF);
   SetPort($4F,$FF);
   SetPort($47,$CC);
   SetPort($4D,$0C);
   DataPort[1] := $CC;
   Dataport[2] := $FC;
   SetTimer(HzStart);
 End;
```

รูป 3.25 โพรซีเยอร์ GetReady

ในบรรทัดแรกจะเป็นการสั่งเปิดพอร์ท แล้วในบรรทัดต่อไปโปรแกรมจะตั้งค่าตัวแปรต่าง ๆ เพื่อให้พร้อมกับการเริ่มทำงาน กำหนดให้พอร์ทอเนกประสงค์หมายเลข GP7-0 ทำงานแบบรับเข้าข้อมูล กำหนดให้พอร์ทอเนกประสงค์หมายเลข G15-8 ทำงานแบบรับเข้าข้อมูล และตั้งค่ารูปแบบสัญญาณให้กับวงจรควบคุมเสต็ปปิ้งมอเตอร์ทั้ง 3 แกนในรูปแบบการเริ่มการทำงานเช่นเดียวกับโปรแกรมจากเครื่องคอมพิวเตอร์ส่วนบุคคล

- UnloadMotor

ในการสั่งให้โปรแกรมสิ้นสุดการทำงาน ต้องส่งสัญญาณดิจิตอลให้กับแผงวงจรควบคุมเสต็ปปิ้งมอเตอร์เพื่อตัดสัญญาณไฟฟ้า และสั่งให้ปิดพอร์ท โดยมีรูปแบบโปรแกรมดังรูป3.26

```
Procedure UnloadMotor;
 Begin
   Setport($47,$FF);
```

```
Setport($4D,$0F);
Setport($13,$0);
End;
```

รูป 3.26 โพรซีเยอร์ UnloadMotor

สำหรับโพรซีเยอร์อื่น ๆ รวมทั้งโปรแกรมหลัก การรับข้อมูล การขับแท่นงานแบบเส้นตรงและแบบเส้นโค้ง มีรูปแบบเหมือนกับโปรแกรมจากคอมพิวเตอร์ส่วนบุคคลทุกประการ

3.2.2.3 การติดตั้งระบบเครือข่ายและการนำเข้าโปรแกรม NC

เครื่องคอมพิวเตอร์แบบแผงวงจรที่เลือกใช้นี้ ไม่มีเครื่องอ่านแผ่นดิสก์ขนาด 3.5 นิ้ว จึงไม่สามารถทำการคัดลอกโปรแกรม NC ด้วยแผ่นดิสก์ได้ วิธีที่สามารถทำการคัดลอกแฟ้มโปรแกรม NC ได้คือ การคัดลอกผ่านทางพอร์ทอีเทอร์เน็ต

คอมพิวเตอร์แบบแผงวงจรเดี่ยวมีพอร์ทอีเทอร์เน็ตติดตั้งมาเรียบร้อยแล้ว โดยเชื่อมต่อกับระบบเครือข่ายผ่านสายสัญญาณ ระบบปฏิบัติการต้องทำการติดตั้งโปรโตคอล NetBEUI เพื่อใช้สื่อสารกับคอมพิวเตอร์เครื่องอื่นที่ติดตั้งบนเครือข่าย ซึ่งการติดตั้งโปรโตคอล NetBEUI นี้สามารถทำได้โดยการดาวน์โหลดจากเว็บไซท์ http://www.wown.info/j_helmig/dosclnt.htm เพื่อติดตั้งบนระบบปฏิบัติการ DOS ซึ่งโปรโตคอลนี้จะสามารถสื่อสารกับคอมพิวเตอร์เครื่องอื่นที่ติดตั้งบนระบบเครือข่าย ซึ่งคอมพิวเตอร์เหล่านั้นจะต้องไม่ติดตั้งระบบปฏิบัติการในตระกูล Windows NT40 (เช่นโปรแกรม Windows XP หรือ Windows 2000 Professional) เพราะระบบดังกล่าวไม่อนุญาตให้โปรโตคอล NetBEUI ทำการสื่อสาร การติดตั้งโปรโตคอลสำหรับคอมพิวเตอร์แบบแผงวงจรเดี่ยวมีโปรแกรมไดร์เวอร์มาให้จากผู้ผลิต

เมื่อทำการติดตั้งโปรโตคอลเพื่อการสื่อสารและตั้งค่าเรียบร้อยแล้ว จึงสามารถติดต่อกับคอมพิวเตอร์เครื่องอื่น และคัดลอกแฟ้มโปรแกรม NC มายังโฟลเดอร์เดียวกันกับโปรแกรมควบคุมเครื่องกัด และใช้โปรแกรม NC นี้เพื่อเป็นแฟ้มคำสั่งในการสั่งกัดชิ้นงานได้

3.2.3 การสร้างโปรแกรม NC

ผู้ใช้งานสามารถออกแบบชิ้นงานที่ต้องการด้วยโปรแกรมที่ใช้ในการสร้างรูป 3 มิติ เช่นโปรแกรม Solidwork 2003 หรือโปรแกรม AutoCAD เป็นต้น แล้วใช้โปรแกรม HyperMILL เพื่อช่วยในการสร้างโปรแกรม NC โดยต้องกำหนดวิธีการกัดชิ้นงาน รูปแบบและขนาดของหัวกัด รวมทั้งรูปแบบของโปรแกรม NC ซึ่งโปรแกรมควบคุมเครื่องกัดนี้สามารถทำงานได้โดยใช้

โปรแกรม NC ที่มีรูปแบบเดียวกันกับเครื่องกัดยี่ห้อ MAZAK ซึ่งผู้ใช้ต้องทำการตั้งค่าผู้ผลิตนี้ก่อนจะแปลงรูปเป็นโปรแกรม NC

3.3 การทดสอบการทำงาน

3.3.1 การทดสอบการทำงานจริง

เมื่อทำการติดตั้งระบบเป็นที่เรียบร้อยแล้ว จึงทดสอบการทำงานแบบกึ่งอัตโนมัติ โดยการป้อนคำสั่งทีละคำสั่งทุกรูปแบบการทำงาน ทั้งการเดินแท่นงานเปล่าโดยไม่ทำการกัดชิ้นงาน และมีการกัดชิ้นงาน เพื่อดูลักษณะการทำงานและหาความผิดพลาดที่สามารถเกิดขึ้นได้ หลังจากนั้นจึงทำการสร้างโปรแกรม NC เพื่อทดสอบการทำงานในระบบอัตโนมัติ ทั้งการทำงานแบบเดินเปล่าและมีการกัดชิ้นงานเช่นกัน หลังจากนั้นนำชิ้นงานที่ได้จากการกัดในระบบอัตโนมัติมาวัดขนาดและเปรียบเทียบกับค่าที่ได้ออกแบบไว้ว่ามีความผิดพลาดเพียงใด

3.3.2 การทดสอบเพื่อหาค่าความผิดพลาด (Error)

การทดสอบเพื่อหาค่าความผิดพลาด (Error) สามารถทดสอบได้โดยให้ระบบทำการควบคุมเครื่องกัดซีเอ็นซีด้วยโปรแกรม NC ที่เขียนขึ้น ทดลองกัดชิ้นงานจริงออกมา แล้วทำการวัดขนาดของชิ้นงานด้วยไมโครมิเตอร์ขนาดความละเอียด 0.001 มิลลิเมตร เปรียบเทียบกับแบบงานที่เป็นข้อมูลในการเขียน G-code โดยทำการกัดชิ้นงานเดียวกันเป็นจำนวน 10 ชิ้น แล้วนำมาหาค่าความผิดพลาดเฉลี่ย ค่าความผิดพลาดมากที่สุด รวมทั้งสาเหตุที่ทำให้เกิดความผิดพลาด

3.3.3 การทดสอบเพื่อหาค่าความละเอียด (Resolution)

การทดสอบเพื่อหาค่าความละเอียด (Resolution) สามารถทำได้โดย

3.3.3.1 การหาค่าความละเอียดในการเคลื่อนที่จากการคำนวณ สามารถหาได้จากความสามารถในการรับค่าสัญญาณน้อยที่สุดที่สามารถทำให้มอเตอร์หมุนได้ และความละเอียดของจำนวนสเต็ปใน 1 รอบของมอเตอร์

3.3.3.2 การหาค่าความละเอียดในการเคลื่อนที่โดยการทดลองจากเครื่องกัดซีเอ็นซี โดยจะใช้วิธีป้อนค่าให้กับโปรแกรมโดยทำการป้อนระยะทางต่าง ๆ ที่กำหนดโดยมีอัตราการเพิ่มที่เท่ากัน จำนวน 100 ครั้ง แล้วนำค่าระยะทางที่วัดได้มาหารด้วย 100 จะได้ค่าความละเอียดในการเคลื่อนที่แต่ละครั้ง

3.3.4 การทดสอบเพื่อหาค่าความสามารถในการทวนซ้ำ (Repeatability)

การทดสอบเพื่อหาค่าความสามารถในการทวนซ้ำ (Repeatability) ทำได้โดยให้โปรแกรมทำการเคลื่อนที่แท่นงานเป็นรูปสี่เหลี่ยม โดยไม่ทำการกัดชิ้นงาน ทำการบันทึกค่าตำแหน่งหัวกัดในจุดเริ่มต้น แล้วเดินแท่นงานตามแนวแกน X เป็นระยะทางที่กำหนด แล้วเดินทาง

ในแนวแกน Y ทำซ้ำจนวนเป็นรูปสี่เหลี่ยม จนกลับมาสู่ตำแหน่งเดิมเป็นรูปสี่เหลี่ยม ทำซ้ำจำนวน 30 รอบ จึงทำการวัดตำแหน่งหัวกัดอีกครั้ง ทำเช่นนี้จำนวน 10 รอบ เพื่อหาความผิดพลาดที่เกิดขึ้นบนแนวแกน X และ Y หลังจากนั้นวัดตำแหน่งเริ่มต้นของหัวกัด เดินหัวกัดขึ้นและลงตามแนวแกน Z เป็นระยะทางที่กำหนดจำนวน 30 รอบ แล้ววัดตำแหน่งหัวกัดอีกครั้ง ทำเช่นนี้ 10 รอบการทำงาน เพื่อหาความผิดพลาดที่เกิดขึ้นในแนวแกน Z