การประยุกต์ใช้งานเครื่องคอมพิวเตอร์เสมือนบนไมโครคอนโทรลเลอร์ โดยอาศัยหลักการไบท์โค้ด

IMPLEMENTATION VIRTUAL MACHINE FOR MICROCONTROLLER BASED ON BYTECODE CONCEPT

นรากร จีนจันทร์
NARAKORN JEENJUN

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรมหาบัณฑิต
สาขาวิชาวิศวกรรมอิเล็กทรอนิกส์
คณะวิศวกรรมศาสตร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
พ.ศ. 2552

การประยุกต์ใช้งานเครื่องคอมพิวเตอร์เสมือนบนไมโครคอนโทรลเลอร์ โดยอาศัยหลักการไบท์โค้ด

IMPLEMENTATION VIRTUAL MACHINE FOR MICROCONTROLLER BASED ON BYTECODE CONCEPT

นรากร จีนจันทร์

NARAKORN JEENJUN

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรมหาบัณฑิต

สาขาวิชาวิศวกรรมอิเล็กทรอนิกส์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

พ.ศ. 2552

KMITL-2009-EN-M-040-066

IMPLEMENTATION VIRTUAL MACHINE
FOR MICROCONTROLLER BASED ON BYTECODE CONCEPT

NARAKORN JEENJUN

A THESIS SUBMITTED IN PARTIAL FULFILLMENT

OF THE REQUIREMENT FOR THE DEGREE OF

MASTER OF ENGINEERING IN ELECTRONICS ENGINEERING

FACULTY OF ENGINEERING

KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG

2009

KMITL-2009-EN-M-040-066

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ใบรับรองวิทยานิพนธ์

หัวข้อวิทยานิพนธ์ การประยุกต์ใช้งานเครื่องคอมพิวเตอร์เสมือนบนไมโครคอนโทรลเลอร์โดยอาศัยหลักการไบท์โค้ด

Thesis Title Implementation Virtual Machine for Microcontroller Based on BYTECODE Concept

นักศึกษา นายนรากร จีนจันทร์

รหัสประจำตัว 47060404

ปริญญา วิศวกรรมศาสตรมหาบัณฑิต

สาขาวิชา วิศวกรรมอิเล็กทรอนิกส์

อาจารย์ที่ปรึกษาวิทยานิพนธ์ รศ.สุชาติ คุณทวีเทพ

หมายเลขวิทยานิพนธ์ KMITL-2009-EN-M-040-066

คณะกรรมการสอบวิทยานิพนธ์		ลายมือชื่อ
รศ.ดร.สมศักดิ์	มิตะถา	
ดร.เรวัต	ศิริโภคาภิรมย์	
รศ.ดร.สุริภณ	สมควรพาณิชย์	
รศ.สุชาติ	คุณทวีเทพ	

วัน / เดือน/ ปี ที่สอบ วันศุกร์ที่ 22 พฤษภาคม พ.ศ. 2552 เวลา 09.30-11.30 น.

สถานที่สอบ ณ อาคาร A ชั้น 3 ห้องประชุม 1

คณะวิศวกรรมศาสตร์ รับรองแล้ว

(รองศาสตราจารย์ ดร.กอบชัย เดชหาญ)

คณบดี คณะวิศวกรรมศาสตร์

วันที่ 22 พฤษภาคม พ.ศ. 2552

หัวข้อวิทยานิพนธ์	การประยุกต์ใช้งานเครื่องคอมพิวเตอร์เสมือน บนไมโครคอนโทรลเลอร์โดยอาศัยหลักการไบท์โค้ด
นักศึกษา	นายนรากร จีนจันทร์
รหัสประจำตัว	47060404
ปริญญา	วิศวกรรมศาสตรมหาบัณฑิต
สาขาวิชา	วิศวกรรมอิเล็กทรอนิกส์
พ.ศ.	2552
อาจารย์ที่ปรึกษาวิทยานิพนธ์	รศ.สุชาติ คูณทวีเทพ

บทคัดย่อ

วิทยานิพนธ์ฉบับนี้เสนอ การออกแบบชุดคำสั่ง Byte code และเครื่องคอมพิวเตอร์เสมือนบนไมโครคอนโทรลเลอร์ โดยอาศัยหลักการทำงานพื้นฐานของภาษาจาวาซึ่งไม่ยึดติดกับระบบปฏิบัติการ ในการวิจัยเลือกใช้ AT89C5131 และ PIC18F4620 ซึ่งการทำงานของระบบตัวเลขและการตรวจสอบเงื่อนไขการทำงานของ Byte code จะทำงานบน Stack Based จากผลการทดลองระบบสามารถรองรับการทำงานของชุดคำสั่งเงื่อนไข If, Else, Repeat, Goto, Waitntil ชุดคำสั่งทางคณิตศาสตร์และลอจิก รวมถึงชุดคำสั่งเพื่อใช้ในการติดต่อกับอุปกรณ์รอบข้างภายนอก เช่น Input ports, Output ports, LCD, RS-232 และ I^2C ได้อย่างถูกต้อง

Thesis Title	Implementation Virtual Machine for Microcontroller Based on BYTECODE Concept
Student	Mr. Narakorn Jeenjun
Student ID	47060404
Degree	Master of Engineering
Program	Electronics Engineering
Year	2009
Thesis Advisor	Asst. Prof. Suchart Khuntaweetep

ABSTRACT

This paper presents the design of Byte code Instruction set and Virtual machine on Microcontroller. The design is based on Java language, it is independent of the operating system. The paper selects AT89C5131 and PIC18F4620 which the number and conditional commands operate with Stack Based. From testing result, the system can support conditional command statement If, Else, Repeat, Goto and Waituntil, including Arithmetic and Logical command statement and Peripheral Interface, Input ports, Output ports, LCD, RS-232 and I^2C port.

กิตติกรรมประกาศ

วิทยานิพนธ์ฉบับนี้ประสบความสำเร็จได้ด้วยการสนับสนุน ตลอดจนการให้คำปรึกษา จาก รองศาสตราจารย์ สุชาติ คุณทวีเทพ อาจารย์ที่ปรึกษา ซึ่งให้แนวความคิด คำแนะนำ ดูแลเอาใจใส่ และผลักดันให้มีกำลังใจในการทำงานตลอดมา ผู้วิจัยขอขอบพระคุณในความอนุเคราะห์ที่ได้รับเป็นอย่างสูง

ขอขอบคุณอาจารย์ทุกๆ ท่าน ที่อบรมสั่งสอนและให้ความรู้ที่มีค่ายิ่งต่อการทำงาน

ขอขอบคุณบิดา มารดา ตลอดจนกำลังใจจาก พี่ น้อง เพื่อนๆ ที่คอยให้กำลังใจและการสนับสนุน

ประโยชน์อันใดที่พึงได้รับจากวิทยานิพนธ์ฉบับนี้ ผู้วิจัยของมอบแด่ผู้มีพระคุณทุกท่าน

นรากร จีนจันทร์

สารบัญ

หน้า

สารบัญ(ต่อ)

หน้า

สารบัญตาราง

ตารางที่ หน้า

สารบัญรูปภาพ

สารบัญรูปภาพ(ต่อ)

สารบัญรูปภาพ(ต่อ)

สารบัญรูปภาพ(ต่อ)

บทที่ 1
บทนำ

1.1 ความเป็นมาและความสำคัญของปัญหา

ในปัจจุบันระบบควบคุมแบบอัตโนมัติถูกนำมาใช้อย่างแพร่หลายในวงการอุตสาหกรรม อุปกรณ์เครื่องใช้ในบ้านทั่วไป ไม่เว้นแม้กระทั่งของเล่น เนื่องจากมีความคล่องตัวในการใช้งานสูง และระบบมีขนาดเล็กทำให้ใช้งานได้ง่ายกว่าไมโครคอมพิวเตอร์ ที่มีขนาดใหญ่ไม่เหมาะในการใช้ควบคุมการทำงาน

ระบบดังกล่าว ใช้ไมโครคอนโทรลเลอร์ขนาดเล็ก และมีราคาถูกมาใช้ในการทำงานทดแทน ภายในบรรจุอุปกรณ์การควบคุมพื้นฐานที่สำคัญ เช่น พอร์ต ไทม์เมอร์ หน่วยความจำโปรแกรม และหน่วยความจำข้อมูล เป็นต้น ปัจจุบันมีบริษัทผู้ผลิตไมโครคอนโทลเลอร์มากมาย ได้ทำการพัฒนาโปรแกรมเพื่อใช้งานกับไมโครคอนโทรลเลอร์ที่มีความหลากหลาย การออกแบบโครงสร้างแต่ละบริษัทจึงแตกต่างกัน ดังนั้นถ้าผู้พัฒนาต้องการเปลี่ยนไปใช้ไมโครคอนโทรลเลอร์แบบอื่น หรือของบริษัทผู้ผลิตอื่น จำเป็นอย่างยิ่งที่ต้องทำความเข้าใจโครงสร้างภายใน หรือแม้กระทั่งภาษาที่ใช้ในการควบคุมการทำงานของไมโครคอนโทรลเลอร์ ทำให้เสียเวลาอย่างมากในการพัฒนาระบบโดยรวม

ตัวอย่าง Virtual Machine (VM) บนไมโครคอนโทรลเลอร์ คือ Meta Cricket [1] ซึ่งพัฒนาโดยสถาบัน MIT โดยออกแบบเพื่อควบคุมการทำงานของหุ่นยนต์ขนาดเล็ก ภาษาที่ใช้ติดต่อเป็นภาษาเฉพาะเรียกว่า Cricket LOGO เป็นภาษาแบบบรรทัด และภาษา LOGO เป็นภาษาแบบรูปภาพ ต่อมาได้มีผู้พัฒนาต่อคือ GOGO BOARD [2] ซึ่งพัฒนาโดยสถาบัน MIT เช่นเดียวกัน ต่อมาได้พัฒนาเป็นต้นแบบของหุ่นยนต์ LEGO [3] โดยใช้งานร่วมกับ ภาษา LEGO เป็นภาษาเฉพาะเช่นกัน และมีการประยุกต์ใช้งานอื่นๆอีกที่ยังไม่ได้กล่าวถึง [4,5]

การใช้งานตัวแปลภาษาที่เห็นได้ชัด คือ ภาษาจาวา โดยจะใช้ JAVA Virtual Machine หรือ JVM ซึ่งจะทำหน้าที่ถอดรหัสไบท์โค้ด [6] แล้วแปลเป็นภาษาเครื่องที่เหมาะสม กับเครื่องไมโครคอมพิวเตอร์ โทรศัพท์มือถือ หรือ คอมพิวเตอร์ขนาดพกพา ซึ่งข้อดีของการออกแบบลักษณะนี้ จะช่วยให้ผู้พัฒนาออกแบบหรือพัฒนาโปรแกรมเพียงครั้งเดียว แต่สามารถใช้โปรแกรมนี้ได้กับอุปกรณ์ทุกประเภทที่มีการติดตั้ง JVM การทำงานของ Meta Cricket ก็ใช้หลักการทำงานของภาษาจาวาเช่นเดียวกัน

วิทยานิพนธ์ฉบับนี้ นำเสนอแนวคิดในการออกแบบตัวแปลภาษาที่ใช้ควบคุมการทำงานของไมโครคอนโทรลเลอร์ สามารถนำโปรแกรมต้นแบบที่เขียนนี้ไปใช้กับไมโครคอนโทรลเลอร์ที่ตระกูลแตกต่างกันได้

1.2 วัตถุประสงค์ของงานวิจัย

- ออกแบบตัวแปลภาษาบนไมโครคอนโทรลเลอร์
- ออกแบบไบท์โค้ดเพื่อใช้ติดต่อกับตัวแปลภาษาบนไมโครคอนโทรลเลอร์

1.3 แนวคิดที่ใช้ในงานวิจัย

หลักการทำงานของภาษาจาวา ได้นำมาประยุกต์ใช้งานกับวิทยานิพนธ์ฉบับนี้ โดยการนำการทำงานของภาษาจาวาที่มีการจำการทำงานบน Java Virtual Machine [1] หรือ เรียกสั้นๆว่า JVM แทนการทำงานกับระบบจริง หน้าที่หลักของ JVM คือเป็นตัวถอดรหัสคำสั่งไบท์โค้ด และทำการติดต่อกับระบบปฏิบัติการจริง ผู้พัฒนาโปรแกรมจึงไม่จำเป็นต้องมีความเข้าใจเกี่ยวกับระบบปฏิบัติการจริงที่ต้องการใช้อยู่ เพราะโปรแกรมจะติดต่อกับ JVM แทน

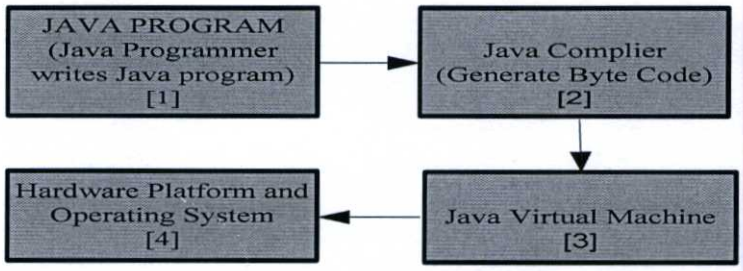
เมื่อผู้พัฒนาโปรแกรมต้องการนำโปรแกรมของตนเองไปใช้งานกับ ระบบปฏิบัติการอื่นๆไม่จำเป็นต้องเขียนโปรแกรมขึ้นใหม่ เพียงทำการติดตั้ง JVM บนระบบปฏิบัติการใหม่ที่ต้องการใช้งาน โปรแกรมก็สามารถทำงานได้ ทำให้ไม่เสียเวลาในการพัฒนาโปรแกรมใหม่เลย เราสามารถอธิบายการทำงานของภาษาจาวาดังรูปที่ 1.1 ได้ดังนี้

[1] เป็นส่วนที่ผู้พัฒนาทำการเขียนโปรแกรมขึ้นมาด้วยภาษาจาวา

[2] โปรแกรมที่ได้คอมไพล์ด้วยจาวาคอมไพล์เลอร์และเปลี่ยนเป็นไบท์โค้ด

[3] การรันโปรแกรมต้องทำการติดตั้ง JVM เพื่อทำการถอดรหัสชุดคำสั่งไบท์โค้ด

[4] JVM ทำหน้าที่ติดต่อกับระบบจริงแทนผู้พัฒนา

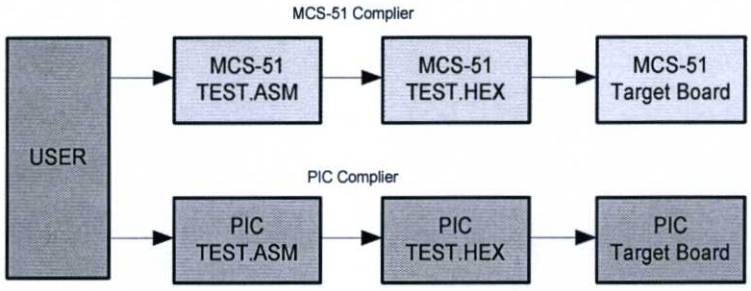


รูปที่ 1.1 Java Operation

1.4 เปรียบเทียบวิธีการที่นำเสนอกับวิธีการแบบพื้นฐาน

1.4.1 การพัฒนาโปรแกรมไมโครคอนโทรลเลอร์แบบเดิม

การพัฒนาโปรแกรมบนไมโครคอนโทรลเลอร์ ต้องอาศัยการเขียนโปรแกรมเพื่อควบคุมการทำงานของรีจิสเตอร์ต่างๆ ที่ออกแบบโดยผู้ออกแบบโดยตรง ผู้ใช้งานจำเป็นอย่างยิ่งที่ต้องมีความรู้พื้นฐานการเขียนโปรแกรม และรู้โครงสร้างภายในของไมโครคอนโทรลเลอร์ที่ต้องการใช้งานแต่ละชนิด ทำให้เสียเวลาในการออกแบบระบบควบคุมเป็นอย่างมาก รูปที่ 1.2 แสดงการออกแบบโปรแกรมของไมโครคอนโทรลเลอร์ 2 ตระกูลที่แตกต่างกัน

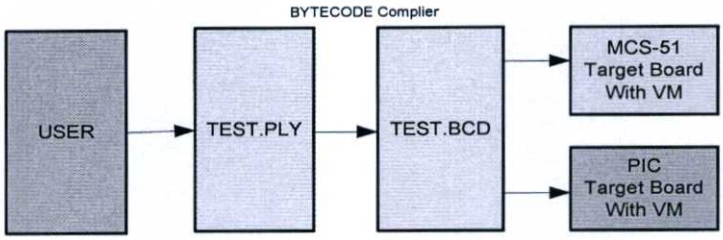


รูปที่ 1.2 Programming for Microcontroller

รูปที่ 1.2 เป็นการพัฒนาโปรแกรมไมโครคอนโทรลเลอร์ตระกูลที่แตกต่างกัน เมื่อผู้พัฒนาต้องการเปลี่ยนใช้ไมโครคอนโทรลเลอร์ตระกูลอื่น ต้องทำการเขียนโปรแกรมขึ้นมาใหม่และทำให้เสียเวลาในการศึกษาโครงสร้างใหม่ด้วย

1.4.2 การพัฒนาโปรแกรมไมโครคอนโทรลเลอร์แบบใหม่

ภาษาที่ออกแบบใหม่เป็นภาษาระดับระดับสูงช่วยให้ผู้พัฒนาทำความเข้าใจได้ง่าย เพราะโปรแกมที่เขียนขึ้นจากผู้พัฒนา จะติดต่อกับระบบเสมือนบนไมโครคอนโทรลเลอร์แทนการติดต่อกับระบบจริง แสดงดังรูปที่ 1.3



รูปที่ 1.3 Microcontroller on Virtual Machine Concept

1.5 ขอบเขตของงานวิจัยและโครงร่างวิทยานิพนธ์

ทำการออกแบบไบท์โค้ดและเครื่องคอมพิวเตอร์เสมือนบนไมโครคอนโทรลเลอร์ โดยใช้ ไมโครคอนโทรลเลอร์ MCS-51 ของ ATMEL เบอร์ AT89C5131 และ PIC ของ MICROCHIP เบอร์ PIC18F4620 เป็นต้นแบบในการออกแบบ โดยโปรแกรมจากผู้พัฒนาสามารถทำการดาวน์โหลดผ่านทางพอร์ตอนุกรมของไมโครคอมพิวเตอร์ด้วยโปรแกรม HyperTerminal

ชุดคำสั่งที่ใช้งานมีอยู่ 3 ส่วนหลัก คือ

1. ชุดคำสั่งเงื่อนไข ประกอบด้วย IF, ELSE, REPEAT, GOTO และ WAITUNTIL
2. ชุดคำสั่งไม่มีเงื่อนไข เช่น LCD, TX232, I2CWRITE, DIGITALSWA เป็นต้น
3. ชุดคำสั่งการทำงานทางคณิตศาสตร์ ลอจิกและการเปรียบเทียบ

โปรแกรมที่เขียนขึ้นเพื่อใช้เป็นเครื่องคอมพิวเตอร์เสมือนบนไมโครคอนโทรลเลอร์ จะใช้ภาษาซีในการออกแบบโดยใช้ KEIL uVision 3 [7] สำหรับ MCS-51 และ MICROCHIP IDE+ CCS [8] สำหรับ PIC

1.6 ขั้นตอนของการศึกษา

วิทยานิพนธ์ฉบับนี้ แบ่งเนื้อหาออกเป็น 5 บท คือ

บทที่ 1 ความเป็นมาของงานวิจัย ความมุ่งหมายและวัตถุประสงค์ สมมติฐาน ทฤษฎี ขอบเขตของการวิจัย และขั้นตอนการศึกษา

บทที่ 2 โครงสร้างของไมโครคอนโทรลเลอร์ตระกูล MCS-51 และ PIC รวมถึงการทำงานของภาษาจาวา และรูปแบบของไบท์โค้ด

บทที่ 3 การออกแบบเครื่องคอมพิวเตอร์เสมือนและไบท์โค้ด

บทที่ 4 ผลการทดลองที่ได้จากการทำงานของระบบ

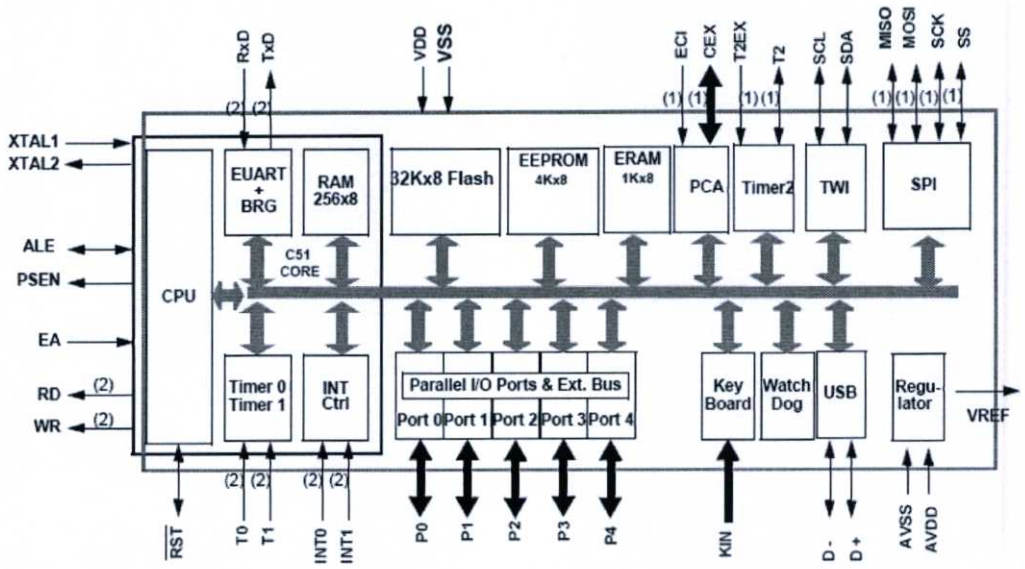
บทที่ 5 บทสรุปผลการวิจัยและข้อเสนอแนะ

บทที่ 2
ทฤษฎีที่เกี่ยวข้องกับวิทยานิพนธ์

2.1 โครงสร้างและสถาปัตยกรรมของ MCS-51

วิทยานิพนธ์ฉบับนี้ ใช้ไมโครคอนโครลเลอร์ตระกูล MCS-51 เบอร์ AT89C5131 ของบริษัท ATMEL มีคุณสมบัติดังนี้

- 80C52X2 สามารถทำงานได้ 6 สัญญาณนาฬิกาต่อคำสั่ง (และสามารถทำได้ 12 สัญญาณนาฬิกาต่อคำสั่ง)
- รองรับสัญญาณนาฬิกาได้สูงสุด 48MHX ที่ X1 และ 24MHZ ที่ X2
- DPTR 2 ตัว
- รองรับการทำงานรับส่งข้อมูลแบบอนุกรม RS-232
- 16 bit Timer Counter
- 256Bytes RAM
- 32Kbytes หน่วยความจำโปรแกรมแฟลชรองรับการโปรแกรมโดยใช้ USB และ พอร์ตอนุกรม
- 4Kbytes EEPROM เป็นหน่วยความจำสำหรับการดาวน์โหลดโปรแกรม 3Kbytes และการใช้งานทั่วไป 1Kbytes
- 1024Bytes ERAM สามารถใช้งานทั่วไปได้
- PCA สามารถโปรแกรมได้ 5 ช่องโดยใช้งานร่วมกับไทม์เมอร์ มีเอาต์พุตเป็น PWM
- การอินเทอร์รัพท์ของคีย์บอร์ดทำที่ P1
- TWI 400Kbit/s (I^2C)
- SPI (Master/Slave)
- 34 I/O
- 4 บิต เอาต์พุตพอร์ต สามารถขับ LED ได้สูงสุด 10mA
- 4 ระดับการจัดลำดับความสำคัญของการอินเทอร์รัพท์จาก 11 แหล่ง
- มีโหมดประหยัดพลังงาน
- สามารถโปรแกรมได้ 2 โหมด คือ In System Program และ In Application Program
- โครงสร้างภายในของ AT89C5131 แสดงดังรูปที่ 2.1



รูปที่ 2.1 AT89C5131 Block Diagram

2.1.1 รายละเอียดของขาสัญญาณ

- P0 : 8 บิต สามารถทำงานเป็น I/O และเป็นแอดเดรสด้านต่ำและดาต้าบัสเมื่อต่อกับหน่วยความจำภายนอก
- P1 : 8 บิต สามารถทำงานเป็น I/O
- P2 : 8 บิต สามารถทำงานเป็น I/O และเป็นแอดเดรสด้านสูงเมื่อต่อกับหน่วยความจำภายนอก
- P3 : 8 บิต สามารถทำงานเป็น I/O และเป็นสัญญาณควบคุมอื่นๆ เช่น พอร์ตอนุกรม, WR, RD, INT0, INT1 เป็นต้น
- P4 : 2 บิต สามารถทำงานเป็น I/O และเป็นสัญญาณควบคุม I^2C
- RST : สั่งงานรีเซ็ตไมโครคอนโทรลเลอร์
- D+ และ D- : เป็นขาของ USB เพื่อใช้ในการโปรแกรมเพียงอย่างเดียวสำหรับวิทยานิพนธ์ฉบับนี้
- ALE : ควบคุมการแยกแอดเดรสและดาต้าของ P0 เมื่อต่อกับหน่วยความจำภายนอก
- PSEN : ควบคุมการทำงานของ OE กรณีที่ใช้หน่วยความจำโปรแกรมภายนอก
- EA : เป็น VCC เมื่อต้องการใช้เป็นหน่วยความจำโปรแกรมภายใน และเป็น VSS เมื่อต้องการใช้เป็นหน่วยความจำโปรแกรมภายนอก
- XTAL1 และ XTAL2 ต่อกับสัญญาณนาฬิกาภายนอก
- VCC : เป็นไฟเลี้ยง 5VDC และ VSS เป็น GND

2.1.2 การเขียนโปรแกรมควบคุมการทำงานของ AT89C5131

ชุดคำสั่งของโมโครคอนโทรลเลอร์ AT89C5131 สามารถใช้งานได้เหมือนกับชุดคำสั่งของ MCS-51 ทั่วไป ภาษาที่ใช้ในการเขียนโปรแกรมควบคุมการทำงานนั้น ส่วนมากใช้ภาษาแอสเซมบลี และภาษาซี ผู้พัฒนาต้องทำความเข้าใจกับโครงสร้างและริจิสเตอร์ภายในโดยละเอียดจึงสามารถเขียนโปรแกรมได้

2.1.3 การเขียนโปรแกรมควบคุมการทำงานของ AT89C5131 ด้วยภาษาแอสเซมบลี

การเขียนโปแกรมด้วยภาษาแอสเซมบลีมีความยุ่งยากมากเนื่องจากผู้พัฒนาต้องเข้าใจการทำงานของภาษาเครื่อง และด้วยตัวภาษาเองเป็นภาษาที่ไม่มีโครงสร้าง จึงยากต่อการพัฒนาและทำการแก้ไขในอนาคต ตัวอย่าง การเขียนโปรแกรมด้วยภาษาแอสเซมบลี แสดงดังรูปที่ 2.2

เราสามารถอธิบายการทำงานได้ ดังต่อไปนี้

- บรรทัดที่ (1) ระบุให้คอมไพล์เลอร์ใช้งาน ชุดคำสั่ง 8051
- บรรทัดที่ (2) ระบุให้คอมไพล์เลอร์รู้ว่าเป็นไฟล์ INTEL 8 บิต
- บรรทัดที่ (3) ส่วนเพิ่มเติมการระบุค่าตำแหน่งของริจิสเตอร์ภายใน
- บรรทัดที่ (4) ตำแหน่งหน่วยความจำโปรแกรมเมื่อไมโครคอนโทรลเลอร์ถูกรีเซ็ต
- บรรทัดที่ (5)-(17) เป็นส่วนโปรแกรมผู้พัฒนา การทำงานของโปรแกรมโดยการส่งค่าออกทาง P1 ซึ่งเป็นพอร์ต จากนั้นสั่งให้หน่วงเวลาด้วยค่าที่กำหนด ด้วยโปรแกรมย่อยชื่อว่า DELAY ต่อจากนั้นทำการหมุนข้อมูลภายในรีจิสเตอร์ A ไปทางขวาและนำค่าที่ได้ออกพอร์ต A อีกครั้ง ทำอย่างนี้ไปเรื่อยๆ จนกว่าไมโครคอนโทรลเลอร์จะถูกตัดไฟเลี้ยง

```
            CPU       "8051.TBL"                  (1)
            HOF       "INT8"                      (2)
            INCL      "AT89C5131.SFR"             (3)

            ORG       0000H                       (4)
            ANL       CKCON0,#11111110B           (5)
START:      MOV       A,#FEH                      (6)
LOOP:       RR        A                           (6)
            MOV       P1,A                        (7)
            ACALL     DELAY                       (8)
            SJMP      LOOP

;*************************************
;           DELAY
;*************************************

DELAY:      MOV   R2,#100                         (10)
L1:         MOV   R3,#50H                         (11)
L2:         MOV   R4,#5H                          (12)
            DJNZ  R4,$                            (13)
            DJNZ  R3,I2                           (14)
            DJNZ  R2,L1                           (15)
            RET                                   (16)
            END                                   (17)
```

รูปที่ 2.2 Assembly Language for AT89C5131

2.1.4 การเขียนโปรแกรมควบคุมการทำงานของ AT89C5131 ด้วยภาษาซี

การเขียนโปรแกรมด้วยภาษาซี มีความยืดหยุ่นมากกว่าภาษาแอสเซมบลี เนื่องจากเป็นภาษาระดับสูง แต่ทำให้ความเร็วของระบบช้าลง การแก้ไขโปรแกรมทำได้ง่ายกว่าเนื่องจากเป็นภาษาที่เป็นโครงสร้าง แสดงดังรูปที่ 2.3

เราสามารถอธิบายการทำงาน ดังต่อไปนี้

- บรรทัดที่ (1) บอกคอมไพล์เลอร์ทำการรวมไฟล์ที่ระบุไว้
- บรรทัดที่ (2) เป็นส่วนเริ่มต้นการทำงาน โดยปกติเริ่มที่ 0000H ของหน่วยความจำโปรแกรม
- บรรทัดที่ (3) ประกาศค่าของตัวแปรขนาด 8 บิต
- บรรทัดที่ (4) คำสั่งวนรอบแบบอนันต์
- บรรทัดที่ (5) คำสั่งวนรอบโดยมีเงื่อนไขกำหนดการทำรอบ
- บรรทัดที่ (6) ส่งค่าออก P1
- บรรทัดที่ (7) เรียกโปรแกรมย่อย delay
- บรรทัดที่ (8) – (10) โปรแกรมย่อยการหน่วงเวลา

```
#include "reg_c51.h"                          (1)


void main(void)                               (2)
{
        unsigned char data,i;                 (3)
        while(1)                              (4)
        {
        for(i=0;i<9;i++)                      (5)
          {
          P1=data<<i;                         (6)
          delay();                            (7)
          }
        }
}

void delay(void)                              (8)
{
        unsigned int k;                       (9)
        for (k=0;k<2000;<k++);                (10)
}
```

รูปที่ 2.3 C Language for AT89C5131

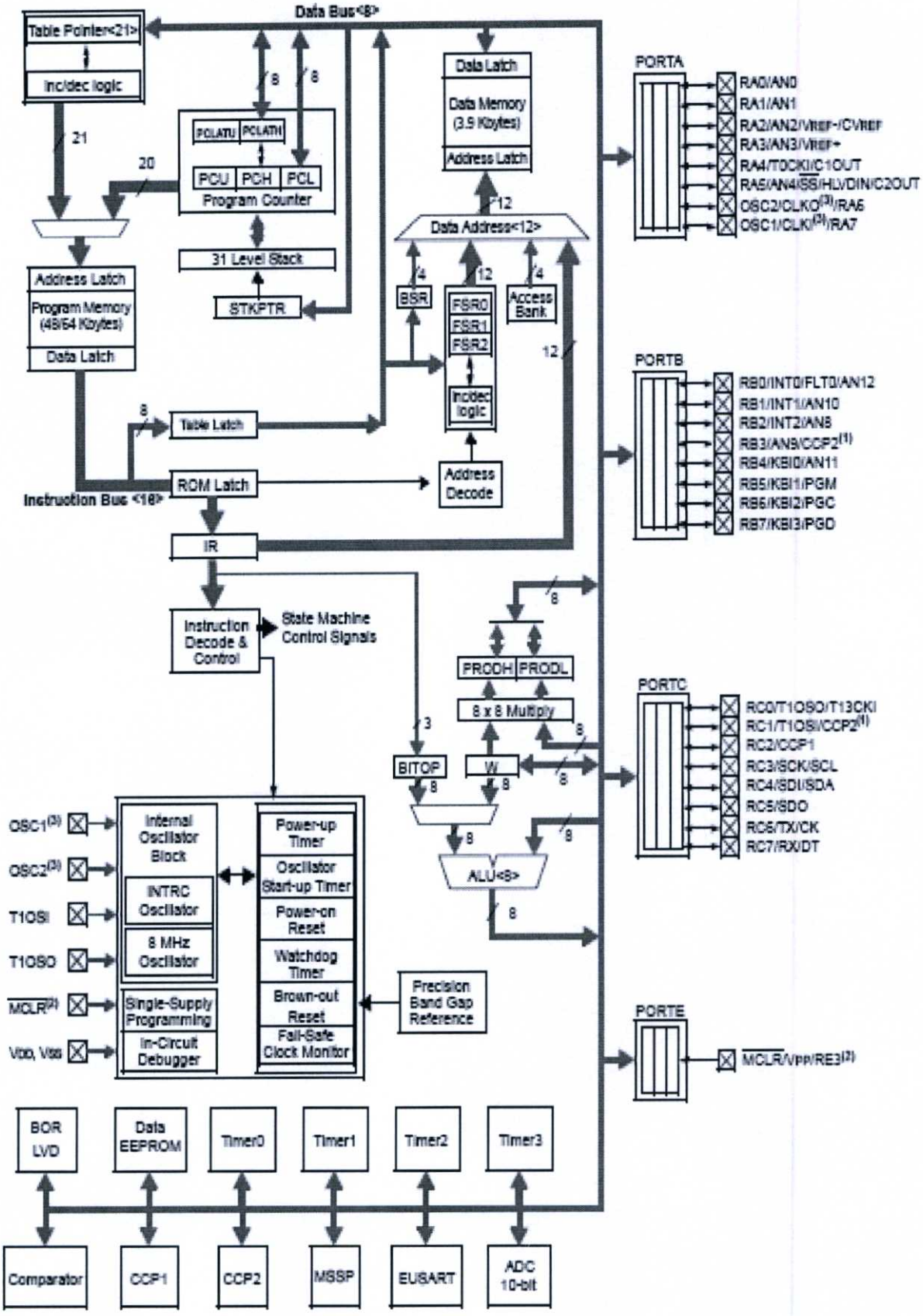
2.2 โครงสร้างและสถาปัตยกรรมของ PIC

วิทยานิพนธ์ฉบับนี้ ใช้ไมโครคอนโครลเลอร์ตระกูล PIC เบอร์ PIC18F4620 ของบริษัท MICROCHIP มีคุณสมบัติดังนี้

- RISC CPU with Nano Watt Technology
- สามารถเลือกสัญญาณนาฬิกาได้ 4 แบบ ที่ความถี่สูงสุด 40MHz
- พอร์ตสามารถรับและจ่ายกระแสได้ 25mA
- มีอินเทอร์รัพท์ภายนอก 3 ตัว
- PCA 2 ตัว
- มีโหมดการรับ-ส่งแบบ SPI, พอร์ตอนุกรม และ I^2C
- 10 bits ADC จาก 13 ช่องอินพุต
- มีตัวเปรียบเทียบอนาล็อก 2 ตัว
- ความสามารถในการลบและเขียนหน่วยความจำแฟลชขนาด 64Kbtyes ได้ 100,000 ครั้ง
- ความสามารถในลบและเขียนหน่วยความจำ EEPROM ขนาด 1Kbtyes ได้ 1,000,000 ครั้ง
- Flash/EEPROM สามารถเก็บข้อมูลได้ 100ปี
- มีหน่วยความจำข้อมูล RAM ขนาด 3986Bytes
- มี 36 I/O

- มีไทม์เมอร์และเคาน์เตอร์ ขนาด 16 บิต 3 ตัว และ ขนาด 8 บิต 1 ตัว
- มีการจัดลำดับการทำงานของอินเทอร์รัพท์
- มี Watch Dog ที่สามารถโปรแกรมการทำงานได้ 4mS-131S
- แหล่งจ่ายไฟเลี้ยงเดี่ยว 5VDC
- สามารถโปรแกรมได้ที่ 5VDC ด้วยการควบคุม 2 สายสัญญาณ
- สามารถทำการตรวจสอบโปรแกรมด้วย ICD
- ทำงานได้ในช่วงแรงดันกว้าง 2.0V – 5.5V
- สามารถโปรแกรมได้ 2 โหมด คือ In System Program และ In Application Program

โครงสร้างภายในของ PIC18F4620 แสดงดังรูปที่ 2.4



รูปที่ 2.4 PIC18F4620 Block Diagram

2.2.1 รายละเอียดของขาสัญญาณต่างๆ

- PA : เป็นพอร์ตขนาด 6 บิต มีโหมดการทำงานหลายอย่าง เช่น เป็น I/O, อนาล็อกอินพุตในกรณีที่ทำงานเป็น ADC และสามารถรับสัญญาณภายนอกในกรณีทำงานเป็นเคาน์เตอร์
- PB : เป็นพอร์ตขนาด 8 บิต มีโหมดการทำงานหลายอย่าง เช่น เป็น I/O, เป็นขาสัญญาณอินเทอร์รัพท์ภายนอก, ขาสัญญาณควบคุมการทำงานขณะโปรแกรมข้อมูลลงหน่วยความจำแฟลช
- PC : เป็นพอร์ตขนาด 8 บิต มีโหมดการทำงานหลายอย่าง เช่น เป็น I/O, เป็นขาสัญญาณ I^2C หรือสามารถใช้งานเป็น พอร์ตอนุกรม ได้
- PD : เป็นพอร์ตขนาด 8 บิต และพอร์ตขนานแบบ PSP(Parallel Slave Port) ได้
- PE : เป็นพอร์ตขนาด 3 บิต และสามารถเป็นสัญญาณการควบคุมการอ่าน หรือเขียนหน่วยความจำภายนอก
- MCLR : เป็นขาควบคุมการรีเซ็ตไมโครคอนโทรลเลอร์
- CLKIN, OSC2/CLK0 : ใช้ร่วมกัน 2 ขา เมื่อต้องการใช้สัญญาณนาฬิกาภายนอก
- VCC และ VSS : เป็นขาที่ใช้ต่อกับไฟเลี้ยงที่ 5VDC และ GND ตามลำดับ

2.2.2 การเขียนโปรแกรมควบคุมการทำงานของ PIC

ชุดคำสั่งของโมโครคอนโทรลเลอร์ PIC18F4620 สามารถใช้งานได้เหมือนกับชุดคำสั่งของ PIC16 ทั่วไป ภาษาที่ใช้เขียนโปรแกรมส่วนมาก คือ ภาษาแอสเซมบลี และภาษาซี เหมือนกับตระกูล MCS-51 ผู้พัฒนาต้องทำความเข้าใจกับโครงสร้างและริจิสเตอร์ภายในอย่างละเอียดจึงสามารถเขียนโปรแกรมได้

2.2.3 การเขียนโปรแกรมควบคุมการทำงานของ PIC18F4620 ด้วยภาษา แอสเซมบลี

ตัวอย่าง การเขียนโปรแกรมการทำงานด้วยภาษาแอสเซมบลี แสดงดังรูปที่ 2.5 เราสามารถอธิบายการทำงาน ดังต่อไปนี้

- บรรทัดที่ (1)-(2) บอกคอมไพล์เลอร์รู้ว่าให้ใช้งาน PIC18F4620
- บรรทัดที่ (3)-(5) กำหนดตำแหน่งการทำงานของตัวแปรบนหน่วยความจำข้อมูล
- บรรทัดที่ (6) กำหนดค่าเริ่มต้นการทำงานของโปรแกรมเมื่อไมโครคอนโทรลเลอร์ถูกรีเซ็ต
- บรรทัดที่ (7)-(10) กำหนดค่าการทำงานเริ่มต้นของ PA ให้เป็นเอาต์พุต

- บรรทัดที่ (11) โหลดค่า 55H ไปเก็บที่ รีจิสเตอร์ W
- บรรทัดที่ (12) โหลดค่า รีจิสเตอร์ W ไปเก็บที่ PA ทำให้ค่าที่ออกจาก PA เท่ากับ 55H
- บรรทัดที่ (13) เรียกโปรแกรมย่อย delay
- บรรทัดที่ (14) โหลดค่า AAH ไปเก็บที่ รีจิสเตอร์ W
- บรรทัดที่ (15) โหลดค่า รีจิสเตอร์ W ไปเก็บที่ PA ทำให้ค่าที่ออกจาก PA เท่ากับ AAH
- บรรทัดที่ (16) เรียกโปรแกรมย่อย delay
- บรรทัดที่ (17) กระโดดไปที่ LABEL loop
- บรรทัดที่ (18)-(28) ส่วนควบคุมการทำงานเพื่อหน่วงเวลาไมโครคอนโทรลเลอร์
- บรรทัดที่ (29) จบการทำงานของโปรแกรม

```
                LIST P=18f4620                  (1)
                include <p18f4620.inc>          (2)

                dt1     EQU     0x23            (3)
                dt2     EQU     0x24            (4)
                dt3     EQU     0x25            (5)

                ORG     0x0000                  (6)

                clrf    TRISA                   (7)
                movlw   0x07                    (8)
                movwf   ADCON1                  (9)
                movwf   CMCON                   (10)
loop            movlw   0x55                    (11)
                movwf   PORTA                   (12)
                call    delay                   (13)
                movlw   0xAA                    (14)
                movwf   PORTA                   (15)
                call    delay                   (16)
                goto    loop                    (17)

delay           movlw   5                       (18)
                movwf   dt1                     (19)
sd3             clrf    dt2                     (20)
sd2             clrf    dt3                     (21)
sd1             decfsz  dt3                     (22)
                goto    sd1                     (23)
                decfsz  dt2                     (24)
                goto    sd2                     (25)
                decfsz  dt1                     (26)
                goto    sd3                     (27)
                return                          (28)
End                                             (29)
```

รูปที่ 2.5 Assembly Language for PIC18F4620

2.2.4 การเขียนโปรแกรมควบคุมการทำงานของ PIC18F4620 ด้วยภาษาซี

ตัวอย่าง การเขียนโปรแกรมการทำงานด้วยภาษาซี แสดงดังรูปที่ 2.6
เราสามารถอธิบายการทำงาน ดังต่อไปนี้

- บอกคอมไพล์เลอร์รู้ว่าให้ใช้งาน PIC18F4620
- จุดเริ่มต้นการทำงานของไมโครคอนโทรลเลอร์โดยกำหนดไว้ที่ 0000H

- บรรทัดที่ (3)-(5) กำหนดค่าเริ่มต้นการทำงานของ PA ให้เป็นเอาต์พุต
- บรรทัดที่ (6) กำหนดการวนรอบเป็นแบบอนันต์
- บรรทัดที่ (7) เขียนค่า 55H ไปที่ PA
- บรรทัดที่ (8) เรียกโปรแกรมย่อยการหน่วงเวลา delay
- บรรทัดที่ (9) เขียนค่า AAH ไปที่ PA
- บรรทัดที่ (10) เรียกโปรแกรมย่อยการหน่วงเวลา delay
- บรรทัดที่ (11) โปรแกรมย่อยใช้ในการหน่วงเวลา
- บรรทัดที่ (12) กำหนดค่าของตัวแปร k ขนาด 16 บิต
- บรรทัดที่ (13) วนรอบค่าของ k ตามที่กำหนดไว้

```
#include <18F4620.h>                    (1)

void main(void)                         (2)
{
        TRIA=0;                         (3)
        ADCON1=7;                       (4)
        CMCON=7;                        (5)

        while(1)                        (6)
        {
          PORTA=0x55;                   (7)
          delay();                      (8)
          PORTA=0xAA;                   (9)
          delay();                      (10)
        }

}

void delay(void)                        (11)
{
        unsigned int k;                 (12)
        for (k=0;k<2000;<k++);          (13)
}
```

รูปที่ 2.6 C Language for PIC18F4620

2.3 การทำงานของภาษาจาวา

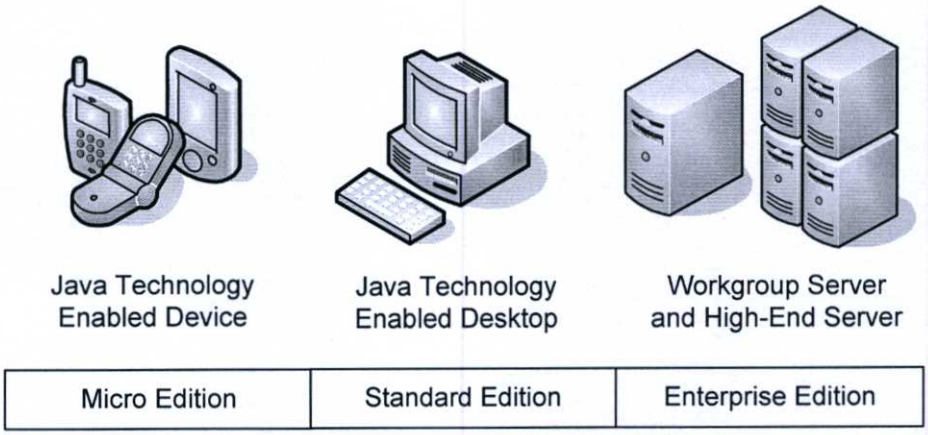
2.3.1 เทคโนโลยีจาวาและประวัติความเป็นมาของภาษาจาวา

ในปี ค.ศ.1991 ทีมวิศวกรของ Sun Microsystems [9] นำทีมโดย James Gosling ได้พัฒนาโปรเจค ชื่อว่า "Green Project" เพื่อทำการวิจัยพัฒนาและสร้างอุปกรณ์อิเล็คทรอนิกส์ที่ชาญฉลาด สามารถทำงานได้ด้วยรีโมทคอนโทรล

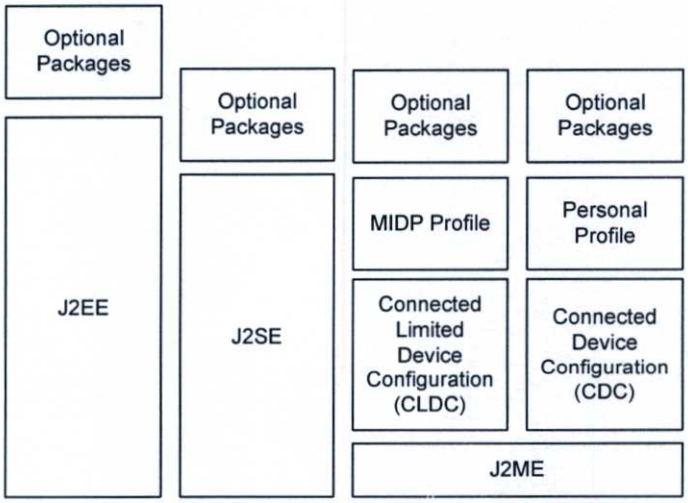
Gosling มีหน้าที่ในการสร้างภาษาสำหรับติดตั้งให้กับอุปกรณ์ เพื่อควบคุมให้อุปกรณ์สามารถทำงานได้อย่างมีประสิทธิภาพ โดยภาษาที่สร้างขึ้นมาต้องมีขนาดเล็ก เพื่ออุปกรณ์เหล่านี้ไม่ต้องใช้หน่วยประมวลผลที่สูงมากในการประมวลผล

ปัจจุบันนี้ภาษาจาวาได้พัฒนาออกเป็น 3 เอดิชัน (edition) ดังรูปที่ 2.7 ประกอบด้วย

- **Java 2 Platform, Standard Edition (J2SE)** เป็นเทคโนโลยีจาวา ที่ออกแบบเพื่อนำมาใช้พัฒนางานบนเครื่องคอมพิวเตอร์ส่วนบุคคล (Personal Computer, PC)
- **Java 2 Platform, Enterprise Edition (J2EE)** เป็นเทคโนโลยีจาวาที่ออกแบบมาให้ใช้งานบนเครื่องเซิร์ฟเวอร์
- **Java 2 Platform, Micro Edition (J2ME)** เป็นเทคโนโลยีที่พัฒนาขึ้นเพื่อเป้าหมายสำหรับพัฒนาแอพลิเคชันที่ทำงานบนอุปกรณ์เครื่องใช้ที่มีทรัพยากรจำกัดไม่ว่าจะเป็นเรื่องขนาดของหน่วยความจำ หรือความสามารถในการประมวลผล เช่น อุปกรณ์พวกโทรศัพท์มือถือ พีดีเอ เป็นต้น



รูปที่ 2.7 Java 2 Platform, Edition



รูปที่ 2.8 สถาปัตยกรรมของ Java Platform Edition

ภายใน J2ME ได้มีการแบ่งออกเป็นเอดิชันย่อยเพื่อความเหมาะสมกับการเลือกไปพัฒนาแอพลิเคชันตามที่แสดงด้วยรูปที่ 2.8 ตามขนาดและลักษณะของงาน อาจแบ่งออกเป็นกลุ่มๆ เหมือนกับเป็นการจัดแบ่งกลุ่มของคลาสและแพ็กเกจของแต่ละเอดิชัน

2.3.2 สถาปัตยกรรมของ J2ME

ปัจจุบันนี้ J2ME เสนอมาตรฐานการติดตั้ง คือ แบบ Connected Limited Device Configuration (CLDC) ส่วนใหญ่ใช้งานกับ โทรศัพท์มือถือ พีดีเอ เป็นต้น ส่วนมาตรฐานการติดตั้งแบบ Connected Device Configuration (CDC) ใช้กันในอุปกรณ์ โทรศัพท์อินเตอร์เน็ต เป็นต้น ในรายงานโครงงานเล่มนี้ขอเสนอสถาปัตยกรรมของ J2ME ที่มีมาตรฐานการติดตั้งแบบ CLDC เนื่องจากเป็นมาตรฐานที่ใช้กันบนโทรศัพท์มือถือ มีโครงสร้างแสดงดัง รูปที่ 2.9

รูปที่ 2.9 สถาปัตยกรรม J2ME ที่มีมาตรฐานการติดตั้งแบบ CLDC

จากรูปที่ 2.9 สามารถอธิบายส่วนประกอบต่างๆ ได้ดังนี้

- **Host Operating System** คือ ระบบปฏิบัติการที่ขึ้นอยู่กับอุปกรณ์ เช่น โทรศัพท์มือถือ Nokia 7650 มี Symbian Operating System เป็นระบบปฏิบัติการ เครื่อง Palm มี Palm Operating System เป็นระบบปฏิบัติการของระบบ เป็นต้น
- **Java Virtual Machines** เป็นส่วนของระบบจัดการควบคุมการทำงานร่วมกันระหว่างตัว Java กับ Host Operating System ให้สามารถทำงานร่วมกันได้
- **Connected Limited Device Configuration (CLDC)** อุปกรณ์ต่างชนิดกัน มีตัวโปรแกรมขับ(Driver) ต่างรูปแบบ ต่างฟังก์ชันกัน ด้วยเหตุผลนี้ J2ME จึงจัดทำตัวติดตั้งขึ้นเพื่อให้ API สามารถทำงานได้บนอุปกรณ์ทุกประเภท CLDC เป็นตัวติดตั้ง

ประเภทหนึ่งโดย CLDC นี้เหมาะกับอุปกรณ์พลังงานแบตเตอรี่ต่ำ เช่น โทรศัพท์มือถือ เป็นต้น

- **Profile** ในการทำงาน J2ME ได้จัด Profile ไว้หลายประเภท Profile ที่ใช้กับ CLDC คือ Mobile Information Device Profile (MIDP) เป็นกลุ่มของ Java API ที่มีการจัดการพวก User Interface การเก็บหน่วยความจำ และ การเชื่อมต่อ โดยตัว Profile นี้เป็นตัวกลางระหว่างแอพลิเคชัน กับ ตัว J2ME

2.3.3 Java Virtual Machine และ Java BYTECODE

Java BYECODE เป็นรูปแบบคำสั่งที่ทำงานร่วมกับ Java Virtual Machine หรือ JVM แต่ละ BYTECODE มีขนาด 1 ไบท์ มีความหมายตรงตัวกับชื่อ โดยมีค่าตั้งแต่ 0-255 หรือ 00H-FFH

BYTECODE เป็นตัวแทนของโปรแกรมจาวาที่สร้างขึ้นหลังจากการคอมไพล์ การทำงานของ BYTECODE จะทำงานร่วมกับ JVM เปรียบเสมือนเป็นตัวแปลภาษาให้เป็นภาษาเครื่องที่ต้องการใช้งานจริง

ระบบที่ต้องการใช้งาน BYTECODE ต้องทำการติดตั้ง JVM ที่เหมาะสมกับระบบนั้นๆ ซึ่งไม่เหมือนกับภาษาโปรแกรมคอมพิวเตอร์แบบอื่นๆ ที่ตัวโปรแกรมหลังจากถูกคอมไพล์แล้วได้เป็นภาษาเครื่องโดยตรงสามารถทำงานได้ทันที

ตัวอย่างโปรแกรมและไบท์โค้ดที่ได้จากการคอมไพล์แสดงดังรูปที่ 2.10

```
Compiled from Employee.java
class Employee extends java.lang.Object {
public Employee(java.lang.String,int);
public java.lang.String employeeName();
public int employeeNumber();
}

Method Employee(java.lang.String,int)
0 aload_0
1 invokespecial #3 <Method java.lang.Object()>
4 aload_0
5 aload_1
6 putfield #5 <Field java.lang.String name>
9 aload_0
10 iload_2
11 putfield #4 <Field int idNumber>
14 aload_0
15 aload_1
16 iload_2
17 invokespecial #6 <Method void storeData(java.lang.String, int)>
20 return

Method java.lang.String employeeName()
0 aload_0
1 getfield #5 <Field java.lang.String name>
4 areturn

Method int employeeNumber()
0 aload_0
1 getfield #4 <Field int idNumber>
4 ireturn

Method void storeData(java.lang.String, int)
0 return
```

รูปที่ 2.10 ตัวอย่างโปรแกรมจาวาและไบท์โค้ด

บทที่ 3
ชุดคำสั่งไบท์โค้ด
และเครื่องคอมพิวเตอร์เสมือนบนไมโครคอนโทรลเลอร์

3.1 BYTECODE Format

ชุดคำสั่งที่ได้ทำการออกแบบเป็นคำสั่งแบบบรรทัด แสดงดังรูปที่ 3.1 ประกอบด้วยส่วนสำคัญ คือ

- AddrH , AddL : ตำแหน่งบรรทัดของโปรแกรม อ้างอิงได้สูงสุด 65536 บรรทัด
- SUM : ผลรวมของไบท์ที่ใช้ไปในบรรทัดนั้นๆ โดยไม่รวม AddrH และ AddrL
- Condition1 : เป็นตัวกำหนดเงื่อนไขหลักของโปรแกรม
- Condition2 : เป็นตัวกำหนดเงื่อนไขรองของโปรแกรม
- CondListH : ใช้เป็นตัวกำหนด Master หรือ Slave ในกรณีที่ค่าของ Condition1 ไม่เท่ากับ 0
    - Slave ค่าของ CondListH จะเป็น 0 เสมอ
    - Master ค่าของ CondListH ใช้เป็นตัวกำหนดจำนวนของ Slave แทน
- CondListL ในกรณีที่ค่าของ Condition1 ไม่เท่ากับ 0 CondListL จะทำหน้าที่อย่างใดอย่างหนึ่งคือ
    - Slave ค่าของ CondListL เป็นตัวกำหนดตำแหน่งบรรทัดย่อย
    - Master ค่าของ CondListL มีค่าเท่ากับ 0 เสมอ

- BYTECODE มีค่า 8 บิต

AddrH, AddrL, SUM, Condition1, Condition2, CondListH, ConListL, BYTECODE

รูปที่ 3.1 BYTECODE Format

3.2 BYTECODE Instruction Set

ชุดคำสั่ง BYTECODE ที่ออกแบบมีทั้งหมด 70 คำสั่ง ประกอบด้วย คำสั่งเงื่อนไข ชุดคำสั่งทางคณิตศาสตร์ลอจิก และการเปรียบเทียบ ส่วนสุดท้ายเป็นชุดคำสั่งสำหรับติดต่อกับอุปกรณ์รอบข้าง แสดงดังตารางที่ 3.1 ระบบที่ทำการทดสอบคือ AT89C5131 สัญญาณนาฬิกา 24Mhz

Command	Condition Code (DEC)	BYTECODE (DEC)	Argument1	Argument2	Return	Byte	Time (uS)	Detail
IF	01-08					1	817	
ELSE	09-16					1	200	
REPEAT	21-28					1	540	
GOTO	100					1	166	
PUSHDATA		01	EXE Stack[x]			2	45	Push 8 bits data to current EXE Stack[x]
Comparison		10	EXE Stack[x]	EXE Stack[x-1]	EXE Stack[x]	1	357	Compare data in EXE Stack[x] and EXE Stack[x-1]
WAITUNTIL		101	DIGITALSWTICHA-D		-	2	448	Wait for switched pressed
WAITTIME		16	EXE Stack[x]			2		Delay with specified time multiply by 100mS
PUSHDATA		37	EXE Stack[x]	EXE Stack[x-1]		3	903	Push data with specified locaton to Internal RAM
COPYDATA		38	EXE Stack[x]	EXE Stack[x-1]		3	913	Copy data from source to destination
GETDATA		39	Address(0-63)		USERTEMP	2	757	Read data from specified address
USERTEMP CLEAR		05				1	300	Reset USERTEMP to be 0
USERTEMP ++		03				1	300	Increment USERTEMP by 1
USERTEMP --		12				1	300	Decrease USERTEMP by 1
PUSH USERTEMP		04	USERTEMP			1	315	Push Data in USERTEMP to current EXE Stack[x]
READ_CUR_Stack		06			USERTEMP	1	317	Read current data in EXE Stack[x] to USERTEMP
MOTORA		46				1	324	Select DC Motor A
MOTORB		47				1	327	Select DC Motor B
MOTORAB		48				1	330	Select DC Motor AB
FORWARD		52				1	695	Forward selected Motor
BACKWARD		53				1	710	Backward selected Motor
STOP		51				1	684	Stop selected Motor
PORTOUT		76	EXE Stack[x]			2	880	Write 8 bits data to port
INPORT		77			USERTEMP	1	939	Read 8 bits data from port
PORTOUTINV		80	EXE Stack[x]			2	1135	Write inverted 8 bits data to port
TX232		19	EXE Stack[x]			2	10100	Write 8 bits data to UART
RX232		20			USERTEMP	1	10150	Read 8 bits data form UART
I2CSTART		78				1	152	Write start signal to I2C
I2CSTOP		79				1	152	Write stop signal to I2C
I2CWRITE		74	EXE Stack[x]				2470	Write 8 bits data to I2C
I2CREAD		75			USERTEMP	1	1547	Read 8 bits data form I2C
DIGITALSWA		57			USERTEMP	1	417	Read status from SWA
DIGITALSWB		58			USERTEMP	1	422	Read status from SWB
DIGITALSWC		59			USERTEMP	1	433	Read status from SWC
DIGITALSWD		60			USERTEMP	1	443	Read status from SWD
LCD		160	EXE Stack[x]			2	2054	Write 8 bits data to LCD
LCDHEX		161	EXE Stack[x]	EXE Stack[x-1]		3	3047	Write 2 ascii to LCD
LCDINIT		162				1	10578	Initialize LCD
LCD_LINE_AUTO		163				1	1118	Automatic set line
LCD_SET_LINE1		164				1	1128	Set LCD to be line 1
LCD_SET_LINE2		165				1	1138	Set LCD to be line 2
LCD_CLEAR		166				1	1148	Clear LCD display

ตารางที่ 3.1 BYTECODE Instruction Set

Command	Condition Code (DEC)	BYTECODE (DEC)	Argument1	Argument2	Return	Byte	Time (uS)	Detail
LCD_SET_CHAR1		176				1	1158	Set LCD character position 1
LCD_SET_CHAR2		177				1	1168	Set LCD character position 2
LCD_SET_CHAR3		178				1	1178	Set LCD character position 3
LCD_SET_CHAR4		179				1	1188	Set LCD character position 4
LCD_SET_CHAR5		180				1	1198	Set LCD character position 5
LCD_SET_CHAR6		181				1	1208	Set LCD character position 6
LCD_SET_CHAR7		182				1	1218	Set LCD character position 7
LCD_SET_CHAR8		183				1	1228	Set LCD character position 8
LCD_SET_CHAR9		184				1	1238	Set LCD character position 9
LCD_SET_CHAR10		185				1	1248	Set LCD character position 10
LCD_SET_CHAR11		186				1	1258	Set LCD character position 11
LCD_SET_CHAR12		187				1	1268	Set LCD character position 12
LCD_SET_CHAR13		188				1	1278	Set LCD character position 13
LCD_SET_CHAR14		189				1	1288	Set LCD character position 14
LCD_SET_CHAR15		190				1	1298	Set LCD character position 15
LCD_SET_CHAR16		191				1	1308	Set LCD character position 16
+		23	EXE Stack[x]	EXE Stack[x-1]	USERTEMP	3	396	USERTEMP=Argument1+Agument2
-		24	EXE Stack[x]	EXE Stack[x-1]	USERTEMP	3	405	USERTEMP=Argument1-Agument2
*		25	EXE Stack[x]	EXE Stack[x-1]	USERTEMP	3	415	USERTEMP=Argument1*Agument2
/		26	EXE Stack[x]	EXE Stack[x-1]	USERTEMP	3	427	USERTEMP=Argument1/Agument2
%		27	EXE Stack[x]	EXE Stack[x-1]	USERTEMP	3	437	USERTEMP=Argument1%Agument2
and		31	EXE Stack[x]	EXE Stack[x-1]	USERTEMP	3	479	Logical AND
or		32	EXE Stack[x]	EXE Stack[x-1]	USERTEMP	3	485	Logical OR
xor		33	EXE Stack[x]	EXE Stack[x-1]	USERTEMP	3	495	Logical XOR
not		34	EXE Stack[x]		USERTEMP	2	500	Logical NOT
=		28	EXE Stack[x]	EXE Stack[x-1]	USERTEMP	3	250	Compare Argument1=Argument2 ?
>		29	EXE Stack[x]	EXE Stack[x-1]	USERTEMP	3	260	Compare Argument1>Argument2 ?
<		30	EXE Stack[x]	EXE Stack[x-1]	USERTEMP	3	270	Compare Argument1<Argument2 ?

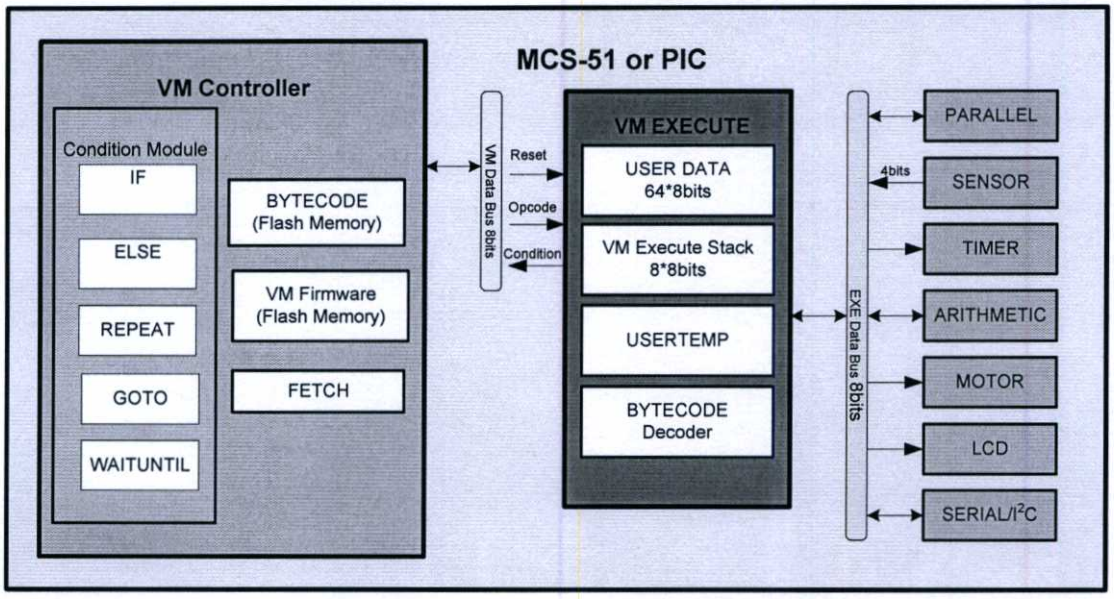
ตารางที่ 3.1 (ต่อ) BYTECODE Instruction Set

3.3 System Overview

ภาพรวมของระบบ แสดงดังรูปที่ 3.2 ประกอบด้วย ส่วนสำคัญหลัก 3 ส่วน คือ

- Virtual Machine Controller (VM Controller)
- Virtual Machine Execute (VM Execute)
- Peripheral Interface

ระบบต้นแบบใช้ไมโครคอนโทรลเลอร์ ตระกูล MCS-51 เบอร์ AT89C5131 ของบริษัท ATMEL หลังจากนั้นใช้ PIC เบอร์ PIC18F4620 ของบริษัท MICROCHIP เพื่อทดสอบการทำงานของ VM



รูปที่ 3.2 System Overview

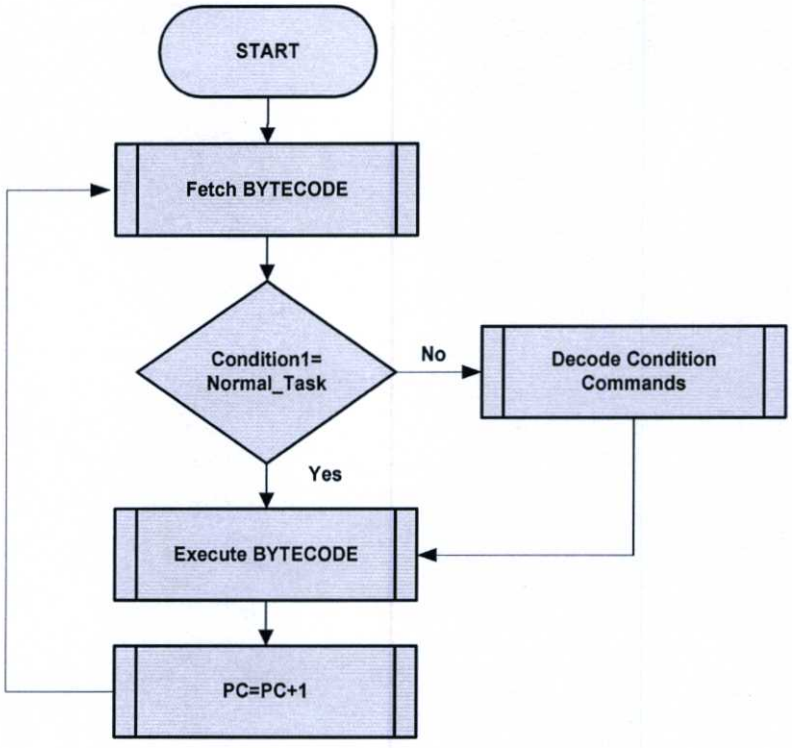
3.4 ส่วนควบคุมการทำงาน VM Controller

การทำงานของส่วนนี้ มีหน้าที่ในการควบคุมระบบโดยรวม ให้ทำงานตามเงื่อนไขตามโปรแกรมของผู้ใช้งานกำหนด ซึ่งมีหน้าที่ทั้งหมดดังต่อไปนี้

- ควบคุมการทำงานของคำสั่งเงื่อนไข IF, ELSE, REPEAT และ WAITUNTIL
- ควบคุมการทำงานของคำสั่ง GOTO/LABEL
- ควบคุมการทำงานของการเฟ็ตไบท์โค้ด (BYTECODE fetching)
- ควบคุมการทำงานของ PC(Program Counter)
- กำหนดค่าเริ่มต้นการทำงานของระบบรวม
- ควบคุมการดาวน์โหลดโปรแกรมของผู้ใช้งาน

การทำงานของ VM แสดงดังรูปที่ 3.3 เริ่มต้นจากการ Fetch BYTECODE จากหน่วยความจำภายในหรือภายนอกไมโครคอนโทรลเลอร์ จากนั้น VM จะตรวจสอบค่า Condition1 ว่าเป็นบรรทัดโปรแกรมที่มีคำสั่งเงื่อนไขหรือไม่ ถ้าใช่ จะทำการตรวจสอบเงื่อนไขก่อน Execute

BYTECODE หลังจากนั้นจะเพิ่ม PC(Program Counter) ขึ้นไป 1 เพื่อ Fetch BYTECODE บรรทัดถัดไป



รูปที่ 3.3 VM Flow chart

3.4.1 โปรแกรมกรณีไม่มีคำสั่งเงื่อนไข

ในกรณีที่ไม่มีคำสั่งเงื่อนไขที่บรรทัดของโปรแกรม ค่า BYTECODE ของ Condition1 จะมีค่าเท่ากับ 0 เสมอ แสดงดังรูปที่ 3.4

```
User Code                          BYTECODE

MOTORA, FORWARD            ;       00,00,03,00,2E,34,
MOTORB, BACKWARDFOR 20     ;       00,01,06,00,2F,35,1,14,10,
TX232('T')                 ;       00,02,04,00,01,54,13,
RX232                      ;       00,03,02,00,14,
MOTORA, STOP               ;       00,04,03,00,2E,3D,
DISPLAY('Hello')           ;       00,05,10,00,01,48,A0,01,65,A0,01,6C,A0,01,6C,A0,01,6F,A0,
```

รูปที่ 3.4 ตัวอย่างโปรแกรมกรณีที่ไม่มีคำสั่งเงื่อนไขควบคุม

3.4.2 โปรแกรมกรณีที่มีคำสั่ง IF และ ELSE

ค่าไบท์โค้ด IF + End IF มีค่า 01-08H และ 81-88H ส่วน ELSE + End ELSE มีค่า 09-10H และ 89-90H ตามลำดับ บรรทัดใดที่มีคำสั่ง IF หรือ ELSE วางอยู่ด้านหน้าของโปรแกรม จะกำหนดให้เป็น Master ส่วนบรรทัดใดที่อยู่ภายในเครื่องหมาย { } จะกำหนดให้เป็น Slave ดังตารางที่ 3.2 แสดงค่าของ Master และ Slave

Type	Condition1	Condition2	CondListH	CondListL
Master	01-08H	*Always 00H*	Number of Slave	*Always 00H*
Slave	01-08H	*Always 00H*	*Always 00H*	Slave Number

ตารางที่ 3.2 Master and Slave Value Table

รูปที่ 3.5 แสดงตัวอย่างโปรแกรมที่มี IF และ ELSE 3 ระดับชั้น โดยแต่ละระดับจะมีค่า BYTECODE แตกต่างกันออกไป

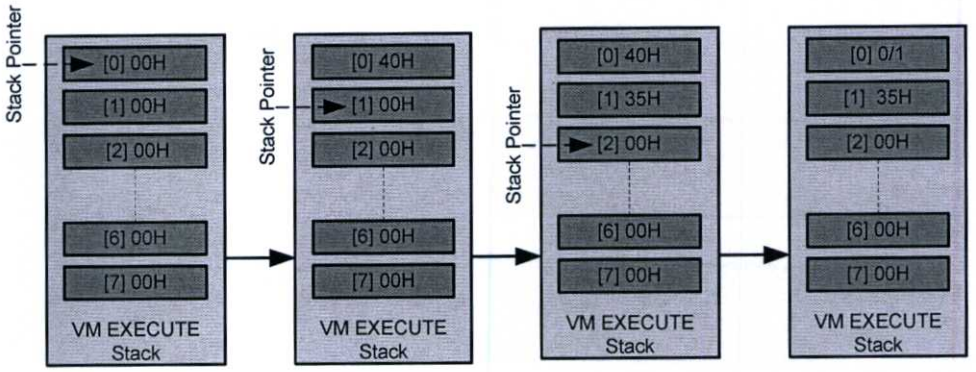
```
IF ()                                   ;Condition = 01H (IF Level 1)
{
        IF()                            ;Condition = 02H (IF Level 2)
        {
                IF()                    ;Condition = 03H (IF Level 3)
                {
                }                       ;Condition = 83H (Close IF Level 3)
                ELSE()                  ;Condition = 0BH (ELSE Level 3)
                {
                }                       ;Condition = 8BH (Close ELSE Level 3)
        }                               ;Condition = 82H (Close IF Level 2)
        ELSE()                          ;Condition = 0AH
        {
        }                               ;Condition = 8AH (Close ELSE Level 2)
}                                       ;Condition = 81H (Close IF Level 1)
ELSE ()                                 ;Condition = 09H (ELSE Level 1)
{
}                                       ;Condition = 89H (Close ELSE Level 1)
```

รูปที่ 3.5 IF และ ELSE 3 ระดับชั้น

การเปรียบเทียบชุดคำสั่งเงื่อนไข IF และ ELSE จะทำบน VM Execute Stack การทำงานแสดงดังรูปที่ 3.6

- เริ่มต้นการทำงานค่าของ Stack กำหนดค่าเป็น 00H เสมอ
- VM จะนำค่าที่ต้องการเปรียบเทียบทั้ง 2 ค่า Push ลง VM Execute Stack
- เมื่อ VM ตรวจสอบพบคำสั่ง Comparison, VM จะนำค่าใน VM Execute Stack เพื่อทำการเปรียบเทียบ

- ผลลัพธ์การเปรียบเทียบจะ Push ลง VM Execute Stack ตำแหน่งแรกเสมอโดยเป็นค่า True และ False



รูปที่ 3.6 VM Execute Stack

3.4.3 โปรแกรมกรณีที่มีคำสั่ง REPEAT

การทำงานของคำสั่งวนรอบ REPEAT สามารถทำงานซ้อนกันได้สูงสุด 8 ระดับ โดยจะใช้ตัวแปร 3 ตัว บน VM แสดงดังตารางที่ 3.3 โดยค่า n = 1 ถึง 8 โดยเป็นตัวระบุระดับชั้นของคำสั่ง

Variable	Bits
iloopn	8
icntn	8
iaddrn	16

ตารางที่ 3.3 Variables for REPEAT Command

การทำงานของคำสั่ง REPEAT แสดงดังรูปที่ 3.7

- เริ่มต้นการทำงาน iloop ถูกโหลดด้วย ConListH โดยในกรณีนี้จะเป็นค่าการวนรอบ
- iaddr จะถูกโหลดด้วย PC ปัจจุบัน
- icounter เท่ากับ 0
- Execute BYTECODE ตามปรกติ
- เพิ่มค่า icounter ด้วย 1
- ตรวจสอบค่าการนับของ icounter กับค่าที่เก็บใน iloop

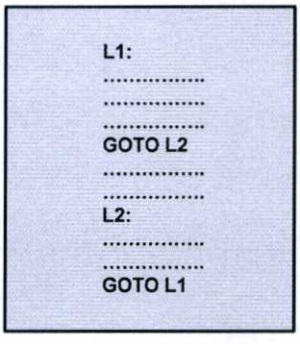
```
User Code                          BYTECODE

LABEL1:                            ;00,00,06,64,00,00,05,00,01,
RX232                              ;00,01,02,00,14,
IF (USERTEMP='1')                  ;00,02,09,01,00,06,00,04,01,31,1C,0A,
{
        REPEAT 8                   ;00,03,04,15,01,08,00,
        {
        OUTPORT(0x01)              ;00,04,07,01,00,00,01,01,01,4C,
        WAITTIME(4)                ;00,05,07,01,00,00,02,01,04,10,
        OUTPORT(0x02)              ;00,06,07,01,00,00,03,01,02,4C,
        WAITTIME(4)                ;00,07,07,01,00,00,04,01,04,10,
        OUTPORT(0x04)              ;00,08,07,01,00,00,05,01,04,4C,
        WAITTIME(4)                ;00,09,07,01,00,00,06,01,04,10,
        }                          ;00,0A,04,C9,01,00,00,
}                                  ;00,0B,04,81,00,00,00,
GOTO LABEL1                        ;00,0C,05,65,00,00,05,00,
```

รูปที่ 3.8 ตัวอย่างโปรแกรมกับคำสั่ง REPEAT

3.4.4 การทำงานของ GOTO และ LABEL

คำสั่ง GOTO จำเป็นต้องทำงานร่วมกับ LABEL ที่ต้องการจะกระโดดไปเสมอ โดยเงื่อนไขการกระโดดมี 2 เงื่อนไข คือ GO UP และ GO DOWN อธิบายได้ดังรูปที่ 3.9

จากตัวอย่าง เห็นได้ว่า โปรแกรมมี 2 LABEL คือ L1 และ L2 ตัวกำหนด GO UP และ GO DOWN กำหนดด้วยคำสั่ง GOTO จากโปรแกรม GOTO L2 เป็น GO DOWN และ GOTO L1 เป็น GO UP



รูปที่ 3.9 GO UP and GO DOWN

การทำงานของ VM เริ่มต้นจากการตรวจสอบเงื่อนไขในการกระโดด (ถูกกำหนดจากคอมไพล์เลอร์ซึ่งยังไม่ได้ออกแบบไว้ในวิทยานิพนธ์ฉบับนี้) จากนั้น VM จะโหลดค่าตำแหน่งของ LABEL เมื่อ VM ทำงานถึงบรรทัดคำสั่ง GOTO จะรีโหลดค่าตำแหน่งจาก LABEL ไปยัง PC เพื่อกระโดดไปยังบรรทัดที่ต้องการ ขนาดของ LABEL มีขนาด 16 บิต ความลึก 64 ตำแหน่ง โดยสามารถอ้างอิงตำแหน่งบรรทัดได้สูงสุด 65535

3.4.5 การทำงานของคำสั่ง WAITUNTIL

การทำงานของคำสั่ง WAITUNTIL ทำงานร่วมกันระหว่าง VM Controller และ VM Execute การทำงานจะวนรอบจนกระทั่งเงื่อนไขที่กำหนดเป็นจริง ถ้าเงื่อนไขไม่เป็นจริงจะไม่ไปบรรทัดต่อไป คำสั่ง WAITUNTIL จะทำงานร่วมกับคำสั่ง DIGITALSWA, DIGITALSWB, DIGITALSWC และ DIGITALSWD เท่านั้น

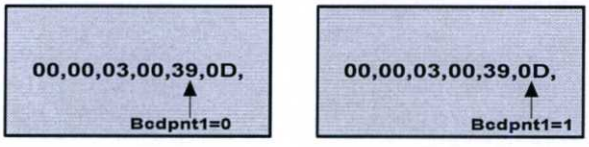
ตัวอย่าง เมื่อต้องการรอรับค่าของ DIGITALSWA โปรแกรมเทียมที่ได้จะเป็น ***WAITUNTIL DIGITALSWA*** โดยมีค่าเท่ากับ ***00, 00, 03, 00, 39,0D*** อธิบายการทำงานได้ดังนี้

- 00,00 : AddrH และ AddrL ตามลำดับ
- 03 : SUM
- 00 : Condition1 (no condition control)
- 39 : อ่านค่า DIGITALSWA
- 0D : รอจนกว่า DIGITALSWA เปลี่ยนรหัสสัญญาณจาก High เป็น Low

เพื่อให้เข้าใจการทำงานของคำสั่ง WAITUNTIL จะต้องกล่าวถึง PC ของ VM ด้วย

- bcdpnt : ทำหน้าที่เป็น PC ควบคุมการชี้ตำแหน่งของโปรแกรมหลัก
- bcdpnt1 : ทำหน้าที่เป็นบอกตำแหน่ง BYTECODE ภายในบรรทัดนั้นที่ทำงานอยู่ การทำงานโดยปรกติ จะเริ่มบอกตำแหน่งถัดจาก Condition1 ในกรณีที่ไม่มีคำสั่งเงื่อนไข และเริ่มบอกตำแหน่งถัดจาก CondListL ในกรณีที่มีคำสั่งเงื่อนไข

จากรูปที่ 3.10 bcdpnt1 จะระบุตำแหน่งข้อมูลแรก คือ 39H ซึ่งเป็นการอ่านค่าการกดของ DIGITALSWA ต่อไปทำการเพิ่ม bcdpnt1+1 ซึ่งจะเป็น 0DH ถ้าเงื่อนไขเป็นจริง ก็จะจบการทำงานของโปรแกรมบรรทัดนี้ แต่ถ้าเงื่อนไขไม่เป็นจริง และจะได้ bcdpnt-1 เพื่อกลับไปอ่านค่า DIGITALSWA ใหม่



รูปที่ 3.10 การทำงานของคำสั่ง WAITUNTIL

3.5 การทำงานของ VM Execute

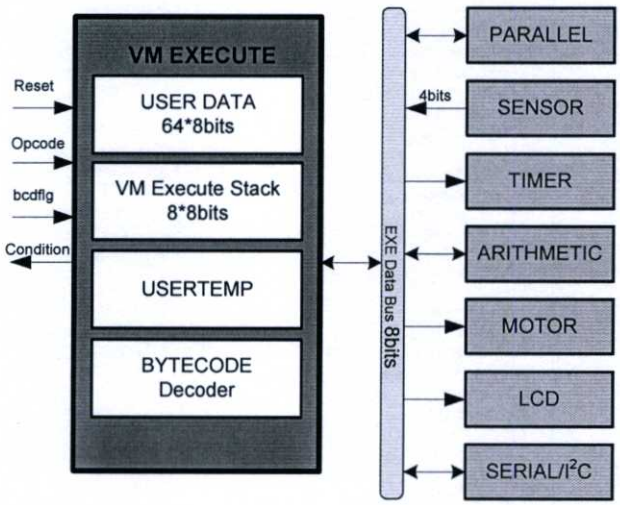
การทำงานของ VM Execute มีหน้าที่ ดังต่อไปนี้

- ถอดรหัสคำสั่ง BYTECODE

- ส่งค่าผลลัพธ์การทำงานให้กับ VM Controller มี 4 แบบคือ if_true, if_false, if_waituntil และ if_pushdata
- ควบคุมการทำงานของ VM Execute Stack
- ควบคุมการทำงานของอุปกรณ์รอบข้าง
- ควบคุมการทำงานของ Arithmetic, Logic และ Comparison

การทำงานของ VM Execute แสดงดังรูปที่ 3.8 โดยสัญญาณทางด้านซ้ายจะทำหน้าที่ติดต่อกับ VM Controller โดยมีอินพุต 3 ค่า และ ส่งค่ากลับ1 ค่า ดังต่อไปนี้

- *bcdcode* : เป็นค่าของ BYTECODE ขนาด 8 บิต
- *bcdflg* : ทำหน้าที่แยก BYTECODE โดยที่ 00H คือ Command และ 01H คือ Data
- *Reset* : ทำหน้าที่ควบคุมการเคลียร์ค่าของ VM Execute Stack
- *Condition* : เป็นค่าที่ใช้ส่งกลับไปให้ VM Controller



รูปที่ 3.11 VM Execute

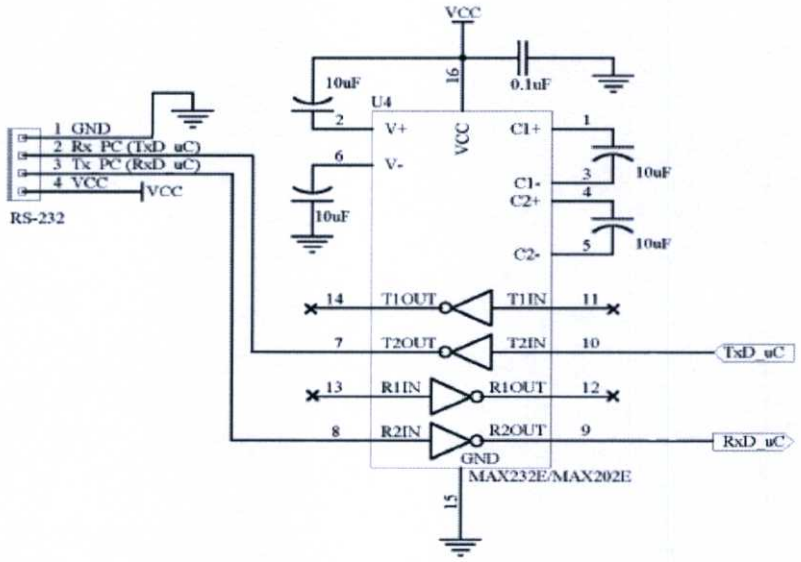
3.6 Peripheral Interface

อุปกรณ์รอบข้างที่ใช้ติดต่อกับระบบมีทั้งหมด 6 ส่วนที่สำคัญ ประกอบด้วย พอร์ตอนุกรม, I^2C พอร์ต, พอร์ตขนาน, LCD, สวิทช์, มอเตอร์ และ สวิทช์ควบคุมการดาวน์โหลดโปรแกรม ซึ่งสามารถเพิ่มเติมได้ภายหลัง

3.6.1 พอร์ตอนุกรม

- AT89C5131 : Txd เชื่อมต่อกับขา P3.1 และ Rxd เชื่อมต่อกับขา P3.0
- PIC18F4620 : Txd เชื่อมต่อกับขา RC6 และ Rxd เชื่อมต่อกับขา RC7

การเชื่อมต่อพอร์ตอนุกรมกับไมโครคอมพิวเตอร์ แสดงดังรูปที่ 3.12

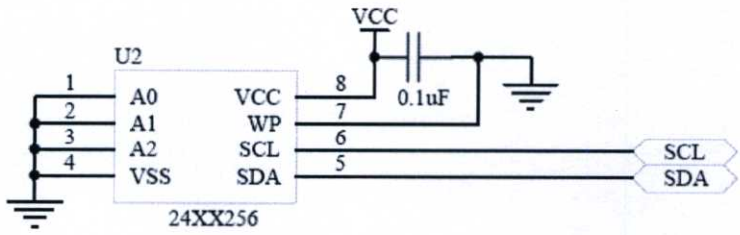


รูปที่ 3.12 วงจรแปลงสัญญาณอนุกรมของ AT89C5131 และ PIC18F4620

3.6.2 พอร์ต I^2C

- AT89C5131 : SCL เชื่อมต่อกับขา P4.0 และ SDA เชื่อมต่อกับขา P4.1
- PIC18F4620 : SCL เชื่อมต่อกับขา RE0 และ SDA เชื่อมต่อกับขา RE1

การเชื่อมต่อ EEPROM กับ AT89C5131 และ PIC18F4620 แสดงดังรูปที่ 3.13



รูปที่ 3.13 การเชื่อมต่อ EEPROM กับ AT89C5131 และ PIC18F4620

3.6.3 LCD

การเชื่อมต่อกับ LCD นั้น เลือกใช้งานในโหมด 4 บิตเพื่อประหยัดการใช้งานพอร์ตของไมโครคอนโทรลเลอร์ โดยสัญญาณที่ใช้ควบคุมทั้งหมดมี 7 เส้นแบ่งเป็น ดาต้า 4 เส้น, RS ,RW และ E ตามลำดับ

- AT89C5131 : RS = P2.0, RW = P2.1, E = P2.2 และ DATA = P2.4-P2.7
- PIC18F4620 : RS = RB0, RW = RB1, E = RB2 และ DATA = RB4-RB7

3.6.4 พอร์ตขนาน 8 บิต และดิจิตอลอินพุต 4 บิต

- AT89C5131 : OUTPUT = P0.0-P0.7 และ INPUT = P3.4-P3.7 (ทำงานที่ลอจิกต่ำ)
- PIC18F4620 : OUTPUT = RC0-RC5, RB6-RB7 และ INPUT = RB1-RB4 (ทำงานที่ลอจิกต่ำ)

3.6.5 มอเตอร์

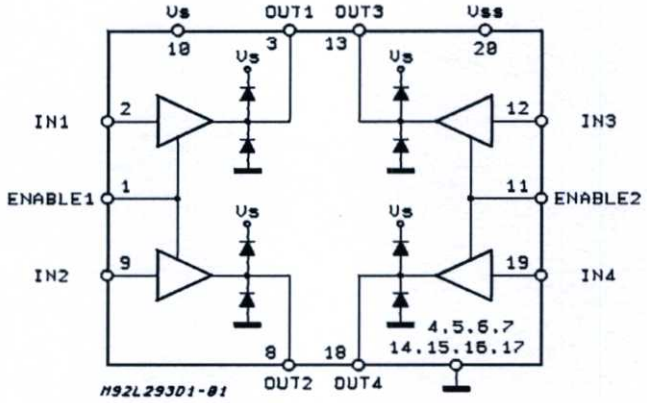
การควบคุมการทำการของมอเตอร์กระแสตรง เลือกใช้ไอซีเบอร์ L293D ของบริษัท ST ซึ่งสามารถต่อมอเตอร์ได้ 2 ตัว ผ่านทาง OUT1-OUT4 โดยที่ OUT1-OUT2 ทำหน้าที่เป็น MOTORA และ OUT3-OUT4 ทำหน้าที่เป็น MOTORB

การควบคุมทิศทางการหมุนซ้ายและขวาจะถูกควบคุมด้วย IN1-IN4 และ ENABLE1-ENABLE2 โดยถ้าขา ENABLE1-2 เป็นลอจิกต่ำ จะทำให้เอาต์พุตมีสถานะความต้านทานสูงซึ่งส่งผลให้ OUTPUT1-4 มีสถานะความต้านทานสูงด้วย

การเชื่อมต่อกับพอร์ตของไมโครคอนโทรลเลอร์เป็นดังนี้

- AT89C5131 : MOTORA ใช้ P1.0-P1.1, MOTORB ใช้ P1.2-P1.3
- PIC18F4620 : MOTORA ใช้ RA0-RA1, MOTORB ใช้ RA2-RA3

วงจรภายในของ L293D แสดงดังรูปที่ 3.14 ไม่จำเป็นต้องมีอุปกรณ์ภายนอกต่อเพิ่มเติม



รูปที่ 3.14 วงจรภายในของ L293D

3.6.6 สวิทช์ควบคุมการดาวน์โหลดโปรแกม

สวิทช์นี้ออกแบบเพื่อควบคุมการทำงานการดาวน์โหลดโปรแกรมเป็นขาอินเทอร์รัพท์ภายนอกซึ่งจะตรวจจับขอบขาลงของสัญญาณ

- AT89C5131 : P3.2
- PIC18F4620 : RB0

3.7 ขนาดของหน่วยความจำในการติดตั้ง VM

ขนาดของหน่วยความจำโปรแกรม, ขนาดของหน่วยความจำข้อมูลและ คอมไพล์เลอร์ที่ใช้ แสดงตารางที่ 3.4

CPU	IDE	C Complier	Assembly Compiler	Code Usage	Data Usage
AT89C5131	KEIL uVision3	C51 V8.06	A51 V8.06	10.97KB	434B
PIC18F4620	MPLAB V7.5	CCS 4.023	MPLAB V7.5	14.96KB	450B

ตารางที่ 3.4 พื้นที่หน่วยความจำที่ใช้สำหรับ AT89C5131 และ PIC18F4620

Part	Vender	Code	Data
AT89C5115	ATMEL	16KB	512B
AT89C51AC2	ATMEL	32KB	1280B
AT89C51AC3	ATMEL	64KB	2304B
AT89C51ED2	ATMEL	64KB	2048B
AT89C51IC2	ATMEL	32KB	1280B
AT89C51ID2	ATMEL	64KB	2048B
AT89C51RB2	ATMEL	16KB	1280B
AT89C51RC2	ATMEL	32KB	1280B
AT89C51RD2	ATMEL	64KB	2048B
AT89C51RE2	ATMEL	128KB	8448B
P89V660FA	NXP	16KB	512B
P89V660FB	NXP	32KB	512B
P89V660FC	NXP	64KB	512B
P89C51RD2	NXP	64KB	1024B

ตารางที่ 3.5 ตัวอย่างของ MCS-51 ที่สามารถใช้งานได้

Part	Vender	Code	Data
PIC18F2410	MICROCHIP	16KB	768B
PIC18F2420	MICROCHIP	16KB	768B
PIC18F2423	MICROCHIP	16KB	768B
PIC18F2431	MICROCHIP	16KB	768B
PIC18F2450	MICROCHIP	16KB	768B
PIC18F2480	MICROCHIP	16KB	768B
PIC18F4420	MICROCHIP	16KB	768B
PIC18F4423	MICROCHIP	16KB	768B
PIC18F2455	MICROCHIP	24KB	2048B
PIC18F2510	MICROCHIP	32KB	1536B
PIC18F2515	MICROCHIP	48KB	3968B
PIC18F2525	MICROCHIP	48KB	3968B
PIC18F2610	MICROCHIP	64KB	3968B
PIC18F2620	MICROCHIP	64KB	3968B

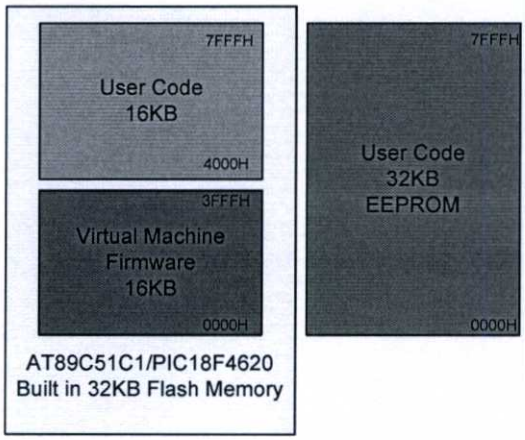
ตารางที่ 3.6 ตัวอย่างของ PIC ที่สามารถใช้งานได้

3.8 การจัดสรรหน่วยความจำของ VM

การจัดสรรหน่วยความจำ แสดงดังรูปที่ 3.12 มีส่วนประกอบ 2 ส่วน คือ

- VM : Built-in Flash ตำแหน่ง 0000-3FFFH 16KB
- User Code : มี 2 ประเภทแล้วแต่การกำหนดการทำงานว่าต้องการใช้แบบใด
  - Internal User Code : Built-in Flash ตำแหน่ง 4000-7FFFH 16KB

- External User Code : EEPROM I^2C Interface ตำแหน่ง 0000-7FFFH 32KB



รูปที่ 3.15 Memory Map

3.9 การดาวน์โหลดโปรแกรม

การทำงานของส่วนนี้ VM Controller จะทำการตรวจจับการทำงานของอินเทอร์รัพท์ภายนอก ซึ่งต่ออยู่กับไมโครสวิทซ์ว่ามีการกดหรือไม่ หลังจากนั้นทำการติดต่อกับไมโครคอมพิวเตอร์ ผ่านทางพอร์ตอนุกรมที่ความเร็ว 9600 bps ด้วยโปรแกรม HyperTerminal ซึ่งสามารถใช้ได้กับ WINDOW95/98/ME/2000/XP

บทที่ 4

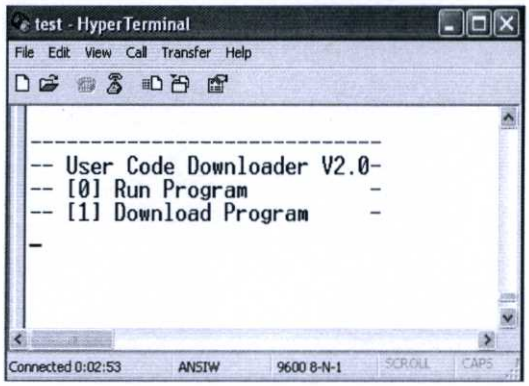
ผลการทดลอง

4.1 การดาวน์โหลดโปรแกรม

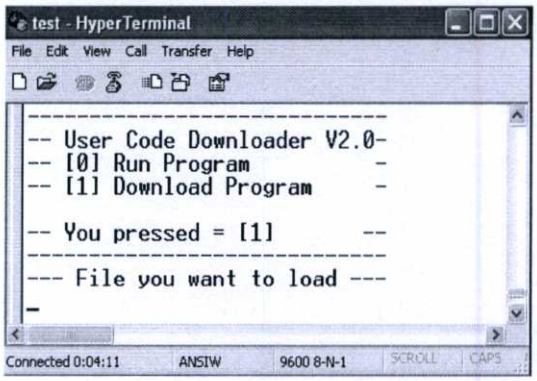
การทำงานของระบบมี 2 โหมดการทำงาน คือ โหมดรัน และโหมดดาวน์โหลดโปรแกรม ทำงานร่วมกับ HyperTerminal บนไมโครคอมพิวเตอร์ โดยมี สวิทซ์ ต่อกับ P3.2 ที่เป็นขาอินเทอร์รัพท์ภายนอกของไมโครคอนโทรลเลอร์ AT89C5131 และขา RB0 เป็นขาอินเทอร์รัพท์ภายนอกของไมโครคอนโทรลเลอร์ PIC18F4620 โดยจะตรวจจับการทำงาน ที่ขอบขาลงของสัญญาณ

การใช้งานโปรแกรมดาวน์โหลดสามารถอธิบายได้ดังนี้

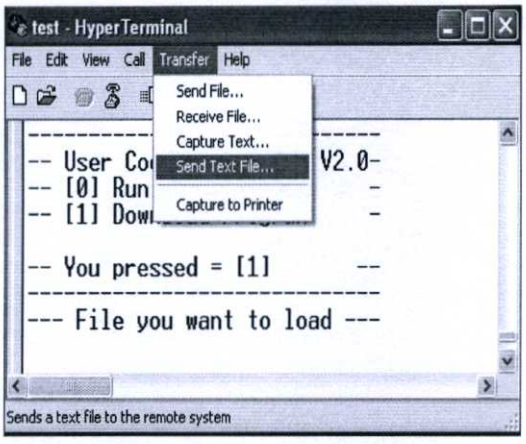
- เปิดโปรแกรม HyperTerminal บนไมโครคอมพิวเตอร์ เมื่อกดสวิทซ์ดาวน์โหลดของไมโครคอนโทรลเลอร์ จะแสดงข้อความบน HyperTerminal ดังรูปที่ 4.1 ซึ่งสามารถเลือกใช้งานได้ [0] คือ รันโปรแกรมปัจจุบันที่อยู่บนแฟลชและ EEPROM ส่วน [1] จะเข้าโหมดการทำงานเพื่อดาวน์โหลดโปรแกรม
- เมื่อผู้ใช้งานเลือก [1] จะเข้าโหมดดาวน์โหลด แสดงดังรูปที่ 4.2 โดยตัวโปรแกรมจะบอกให้เลือกไฟล์ที่ต้องการดาวน์โหลด
- หลังจากนั้นให้ไปที่ เมนู Transfer -> Send Text File .... แสดงดังรูปที่ 4.3
- เมื่อดาวน์โหลดโปรแกรมบนแฟลชและ EEPROM เสร็จแล้ว จะมีข้อความให้กดรีเซ็ตที่ ไมโครคอนโทรลเลอร์เพื่อเข้าสู่โหมดการทำงานบน VM แสดงดังรูปที่ 4.4



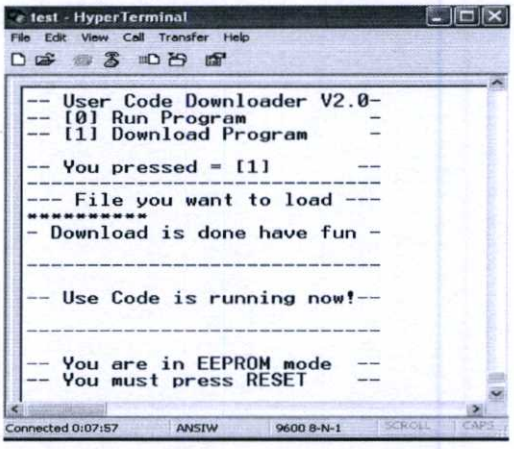
รูปที่ 4.1 เมื่อกดสวิทซ์ดาวน์โหลด



รูปที่ 4.2 เมื่อเลือก [1] เริ่มเข้าสู่โหมดดาวน์โหลด



รูปที่ 4.3 เลือกไฟล์ที่ต้องการโปรแกรม



รูปที่ 4.4 โปรแกรมดาวน์โหลดเสร็จ

4.2 ตัวอย่างโปรแกรมและสัญญาณเอาต์พุต

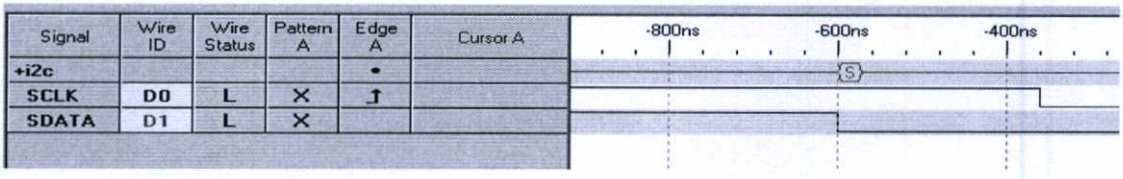
ผลลัพธ์จากการทดลองจะส่งค่าออกพอร์ตขนาด 8 บิต, พอร์ตอนุกรม และ LCD สัญญาณที่ได้ใช้ Logic Analyzer ทำงานเชื่อมต่อกับไมโครคอมพิวเตอร์ด้วยสาย USB ที่มีตัวแปลภาษาช่วยให้สามารถถอดรหัสสัญญาณของ I^2C และ พอร์ตอนุกรม ได้

4.2.1 สัญญาณเอาต์พุตของคำสั่งควบคุม I^2C

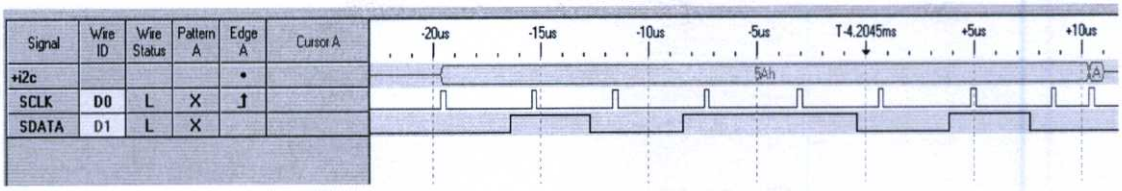
คำสั่งเพื่อใช้งาน I^2C มี 4 คำสั่ง คือ I2CSTART, I2CSTOP, I2CWRITE และ I2CREAD ตัวอย่างรูปที่ 4.5 แสดงการเขียนค่า 5AH ไปที่ I^2C บัส

```
User Code                          BYTECODE

I2CSTART                    ;      00,00,02,00,4E,
I2CWRITE(0x5A)              ;      00,01,04,00,01,5A,4A,
I2CSTOP                     ;      00,02,02,00,4F,
```

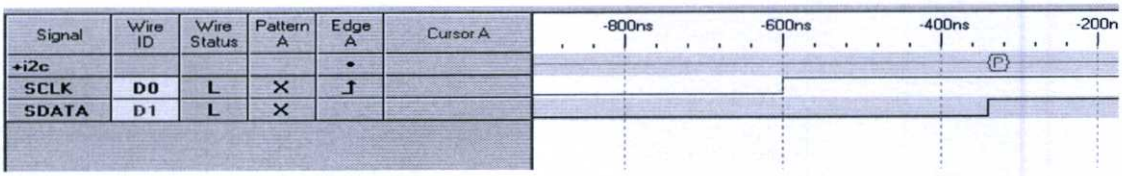
รูปที่ 4.5 ตัวอย่างโปรแกรมเขียนค่า 1 ไบท์กับ I^2C บัส



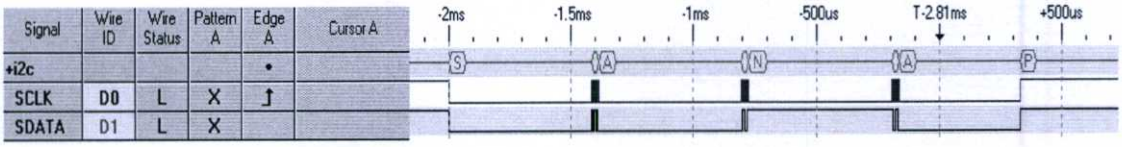
รูปที่ 4.6 สัญญาณ START ของ I^2C



รูปที่ 4.7 เขียนข้อมูล 5AH ลงบน I^2C บัส



รูปที่ 4.8 สัญญาณ STOP ของ I^2C



รูปที่ 4.9 การเขียนข้อมูล 3 ไบท์บน I^2C บัส

4.2.2 สัญญาณเอาต์พุตของคำสั่งควบคุมพอร์ตอนุกรม

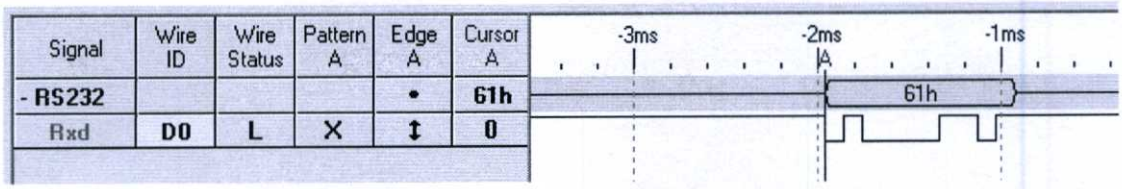
คำสั่งเพื่อใช้งานพอร์ตอนุกรมมี 2 คำสั่ง คือ TX232(d) และ RX232

ตัวอย่างรูปที่ 4.10 เป็นการรับค่าจากพอร์ตอนุกรม 1 ไบท์ค่าที่ได้จะเก็บไว้ที่ USERTEMP จากนั้นนำค่า USERTEMP ส่งออกทางพอร์ตอนุกรมที่ความเร็วเท่ากัน คือ 9600bps การทำงานจะวนรอบอย่างนี้ไปเรื่อยๆ

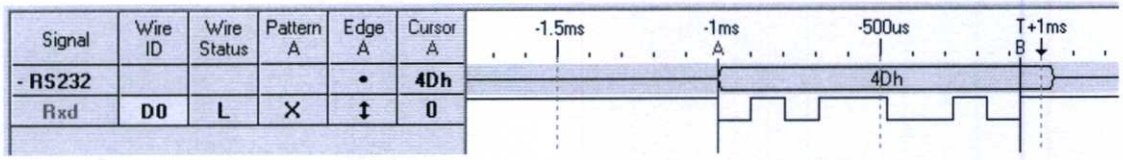
รูปที่ 4.11 แสดงการรับค่า 61H (ตัวอักษร "a") และรูปที่ 4.12 แสดงการรับค่า 4DH (ตัวอักษร "M")

User Code	BYTECODE
LABEL:	;00,00,06,64,00,00,05,00,01,
RX232	;00,01,02,00,14,
TX232(USERTEMP)	;00,02,03,00,04,13,
GOTO LABEL	;00,03,05,65,00,00,05,00,

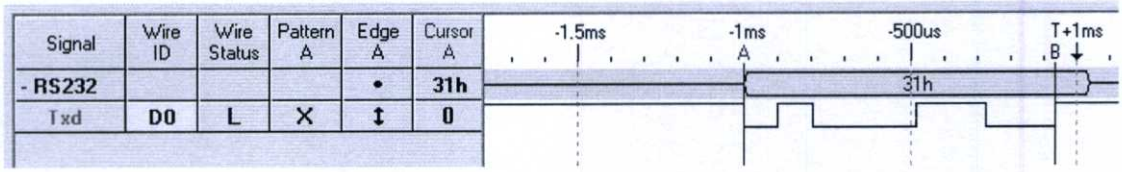
รูปที่ 4.10 ตัวอย่างการใช้งานพอร์ตอนุกรมที่ 9600bps



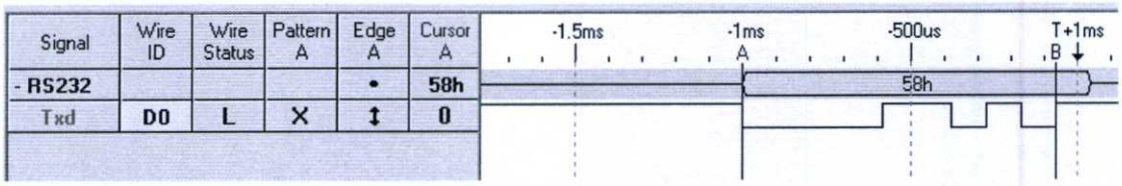
รูปที่ 4.11 รับค่า 61H จากพอร์ตอนุกรม



รูปที่ 4.12 รับค่า 4DH จากพอร์ตอนุกรม



รูปที่ 4.13 ส่งค่า 31H ออกจากพอร์ตอนุกรม



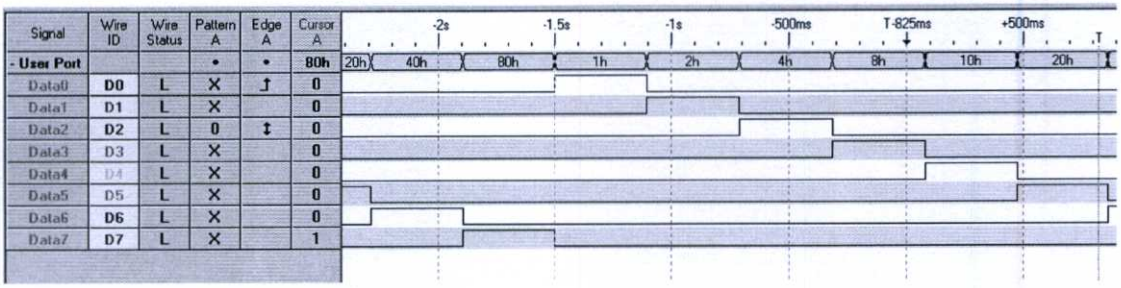
รูปที่ 4.14 ส่งค่า 58H ออกจากพอร์ตอนุกรม

4.2.3 สัญญาณเอาต์พุตของคำสั่งควบคุมพอร์ตขนาน 8 บิต

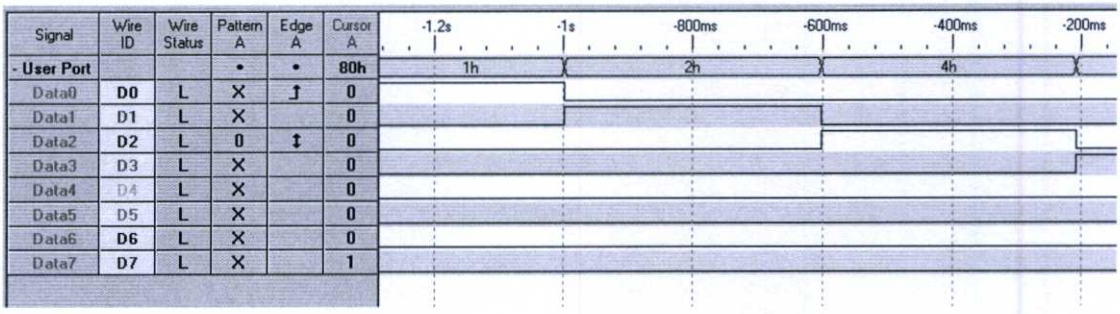
พอร์ตขนานได้ถูกกำหนดให้ใช้งานเป็น เอาต์พุตทั้งหมด 8 บิต ตัวอย่างโปรแกรมรูปที่ 4.15 เป็นการส่งค่าที่กำหนดออกพอร์ต ทีละ 1 บิตโดยเริ่มจาก บิต 0 ถึง บิต 7 โดยมีการใช้งานร่วมกับ คำสั่ง WAITIME ซึ่งกำหนดค่าไว้เท่ากับ 400mS สัญญาณที่ได้ แสดงดังรูปที่ 4.16 และ รูปที่ 4.17 ตามลำดับ

User Code	Byte Code
LABEL:	;00,00,06,64,00,00,05,00,01,
OUTPORT(0x01)	;00,01,04,00,01,01,4C,
WAITTIME(4)	;00,02,04,00,01,04,10,
OUTPORT(0x02)	;00,03,04,00,01,02,4C,
WAITTIME(4)	;00,04,04,00,01,04,10,
OUTPORT(0x04)	;00,05,04,00,01,04,4C,
WAITTIME(4)	;00,06,04,00,01,04,10,
OUTPORT(0x08)	;00,07,04,00,01,08,4C,
WAITTIME(4)	;00,08,04,00,01,04,10,
OUTPORT(0x10)	;00,09,04,00,01,10,4C,
WAITTIME(4)	;00,0A,04,00,01,04,10,
OUTPORT(0x20)	;00,0B,04,00,01,20,4C,
WAITTIME(4)	;00,0C,04,00,01,04,10,
OUTPORT(0x40)	;00,0D,04,00,01,40,4C,
WAITTIME(4)	;00,0E,04,00,01,04,10,
OUTPORT(0x80)	;00,0F,04,00,01,80,4C,
WAITTIME(4)	;00,10,04,00,01,04,10,
GOTO LABEL	;00,11,05,65,00,00,05,00,

รูปที่ 4.15 โปรแกรมการส่งค่าออก USERPORT บิต 0-7



รูปที่ 4.16 สัญญาณการส่งค่าออก USERPORT บิต 0-7



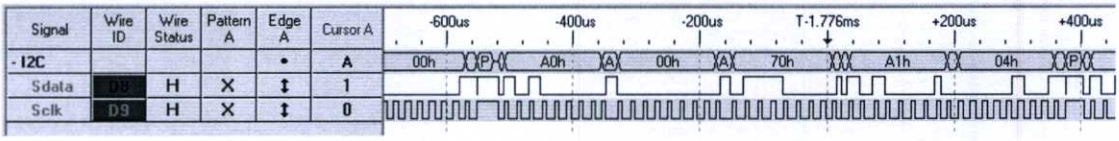
รูปที่ 4.17 WAITTIME มีค่าเท่ากับ 400mS

4.2.4 การ Fetch BYTECODE จากหน่วยความจำภายนอก EEPROM

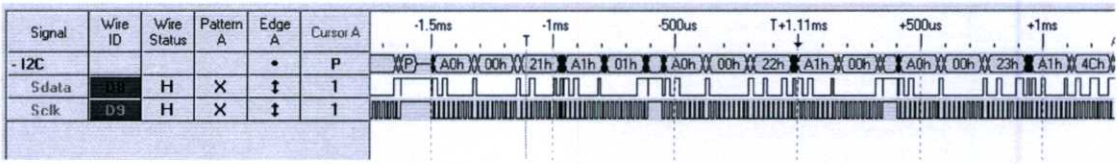
รูปที่ 4.18 แสดงการ Fetch BYTECODE 1 ตำแหน่ง และ รูปที่ 4.19 แสดงการ Fetch BYTECODE 1 บรรทัด จากหน่วยความจำภายนอก รายละเอียดของสัญญาณอธิบายได้ดังนี้

- A0H : เป็นตำแหน่งของ EEPROM
- 00H : เป็นตำแหน่งหน่วยความจำที่ต้องการอ่าน ซึ่งเป็น 8 บิตสูง

- 21H : เป็นตำแหน่งหน่วยความจำที่ต้องการอ่านซึ่งเป็น 8 บิตต่ำ
- A1H : เป็นคำสั่งในการอ่านค่า EEPROM
- 01H : เป็นข้อมูลที่อ่านได้จากตำแหน่ง 0021H
- A0H : เป็นคำสั่งในการเขียนค่าไปที่ EEPROM
- 00H : เป็นตำแหน่งหน่วยความจำที่ต้องการอ่านซึ่งเป็น 8 บิตสูง
- 22H : เป็นตำแหน่งหน่วยความจำที่ต้องการอ่านซึ่งเป็น 8 บิตต่ำ
- A1H : เป็นคำสั่งในการอ่านค่า EEPROM
- 00H : เป็นข้อมูลที่อ่านได้จากตำแหน่ง 0022H
- A0H : เป็นคำสั่งในการเขียนค่าไปที่ EEPROM
- 00H : เป็นตำแหน่งหน่วยความจำที่ต้องการอ่านซึ่งเป็น 8 บิตสูง
- 23H : เป็นตำแหน่งหน่วยความจำที่ต้องการอ่านซึ่งเป็น 8 บิตต่ำ
- A1H : เป็นคำสั่งในการอ่านค่า EEPROM
- 4CH : เป็นข้อมูลที่อ่านได้จากตำแหน่ง 0023H



รูปที่ 4.18 การFetch คำสั่งที่แอดเดรส 70H ไบท์โค้ด คือ 04H



รูปที่ 4.19 การFetch คำสั่ง OUTPUT (00)

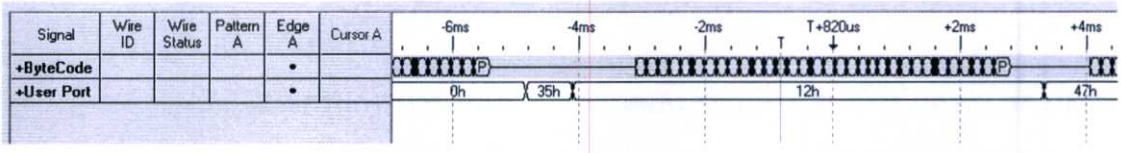
4.2.5 การใช้งานคำสั่งทางคณิตศาสตร์และลอจิก ผลลัพธ์ที่ได้แสดงออก USERPORT

- ตัวอย่างโปรแกรมการบวกเลข 2 จำนวน แสดงดังรูปที่ 4.20 ซึ่งเป็นการบวกค่า 35H ด้วย 12H ผลลัพธ์ที่ได้จะเก็บไว้ที่ USERTEMP หลังจากนั้นจะส่งค่าออกทาง USERPORT แสดงดังรูปที่ 4.21

- ตัวอย่างโปรแกรมการลบเลข 2 จำนวน แสดงดังรูปที่ 4.22 ซึ่งเป็นการลบค่า 35H ด้วย 12H ผลลัพธ์ที่ได้จะเก็บไว้ที่ USERTEMP หลังจากนั้นทำการส่งค่าออกทาง USERPORT แสดงดังรูปที่ 4.23

User Code	BYTECODE
LABEL:	;00,00,06,64,00,00,05,00,01,
0x35+0x12	;00,01,08,00,01,35,4C,01,12,4C,17,
OUTPORT(USERTEMP)	;00,02,03,00,04,4C,
GOTO LABEL	;00,03,05,65,00,00,05,00,

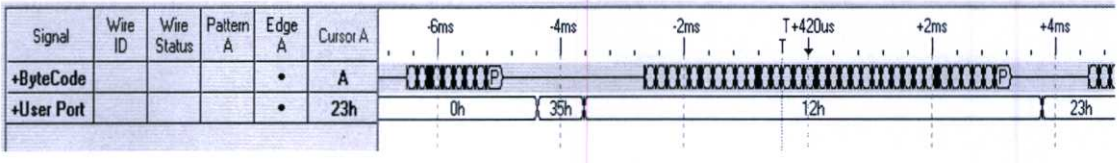
รูปที่ 4.20 โปรแกรมการบวกเลข 2 จำนวน



รูปที่ 4.21 ตัวตั้ง ตัวบวก และผลลัพธ์แสดงค่าออกทาง USERPORT

User Code	BYTECODE
LABEL:	;00,00,06,64,00,00,05,00,01,
0x35-0x12	;00,01,08,00,01,35,4C,01,12,4C,18,
OUTPORT(USERTEMP)	;00,02,03,00,04,4C,
GOTO LABEL	;00,03,05,65,00,00,05,00,

รูปที่ 4.22 โปรแกรมการลบเลข 2 จำนวน

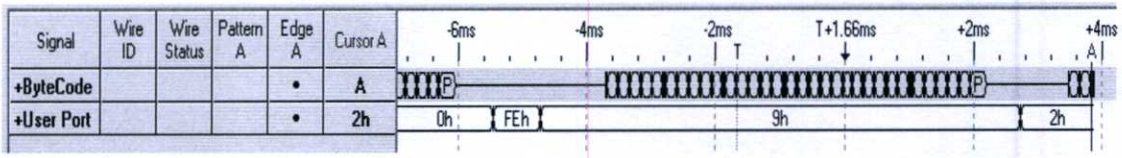


รูปที่ 4.23 ตัวตั้ง ตัวลบ และผลลัพธ์แสดงค่าออกทาง USERPORT

- ตัวอย่างโปรแกรมการคูณเลข 2 จำนวน แสดงดังรูปที่ 4.24 ซึ่งเป็นการคูณค่า 07H ด้วย 05H ผลลัพธ์ที่ได้จะเก็บไว้ที่ USERTEMP หลังจากนั้นจะส่งค่าออกทาง USERPORT แสดงดังรูปที่ 4.25

User Code	BYTECODE
LABEL: 0xFE%0x09 OUTPORT(USERTEMP) GOTO LABEL	;00,00,06,64,00,00,05,00,01, ;00,01,08,00,01,FE,4C,01,09,4C,1B, ;00,02,03,00,04,4C, ;00,03,05,65,00,00,05,00,

รูปที่ 4.28 โปรแกรมการหารเอาเศษเลข 2 จำนวน

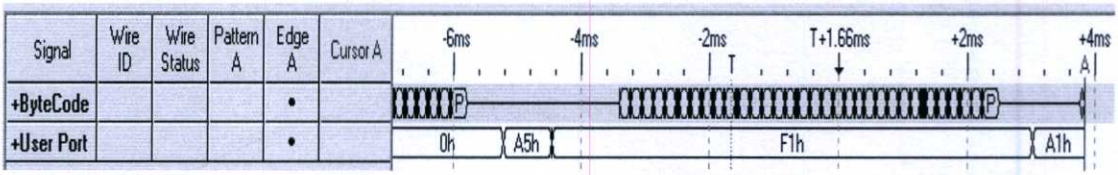


รูปที่ 4.29 ตัวตั้ง ตัวหาร และผลลัพธ์แสดงค่าออกทาง USERPORT

- ตัวอย่างโปรแกรมของลอจิก AND เลข 2 จำนวน แสดงดังรูปที่ 4.30 ซึ่งเป็นการ AND ค่า A5H ด้วย F1H ผลลัพธ์ที่ได้จะเก็บไว้ที่ USERTEMP หลังจากนั้นจะส่งค่าออกทาง USERPORT แสดงดังรูปที่ 4.31

User Code	BYTECODE
LABEL: 0xA5 and 0xF1 OUTPORT(USERTEMP) GOTO LABEL	;00,00,06,64,00,00,05,00,01, ;00,01,08,00,01,A5,4C,01,F1,4C,21, ;00,02,03,00,04,4C, ;00,03,05,65,00,00,05,00,

รูปที่ 4.30 โปรแกรมการ AND เลข 2 จำนวน

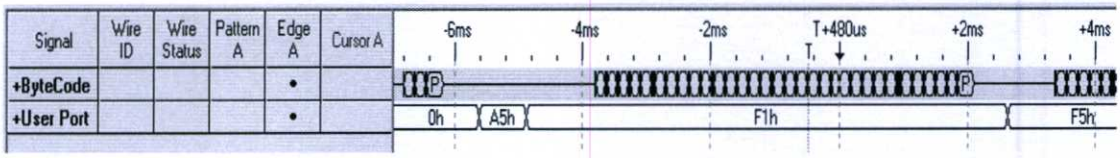


รูปที่ 4.31 ตัวตั้ง ตัวกระทำ และผลลัพธ์แสดงค่าออกทาง USERPORT

- ตัวอย่างโปรแกรมของลอจิก OR เลข 2 จำนวน แสดงดังรูปที่ 4.32 ซึ่งเป็นการ OR ค่า A5H ด้วย F1H ผลลัพธ์ที่ได้จะเก็บไว้ที่ USERTEMP หลังจากนั้นจะส่งค่าออกทาง USERPORT แสดงดังรูปที่ 4.33

User Code	BYTECODE
LABEL:	;00,00,06,64,00,00,05,00,01,
0xA5 or 0xF1	;00,01,08,00,01,A5,4C,01,F1,4C,22,
OUTPORT(USERTEMP)	;00,02,03,00,04,4C,
GOTO LABEL	;00,03,05,65,00,00,05,00,

รูปที่ 4.32 โปรแกรมการ OR เลข 2 จำนวน

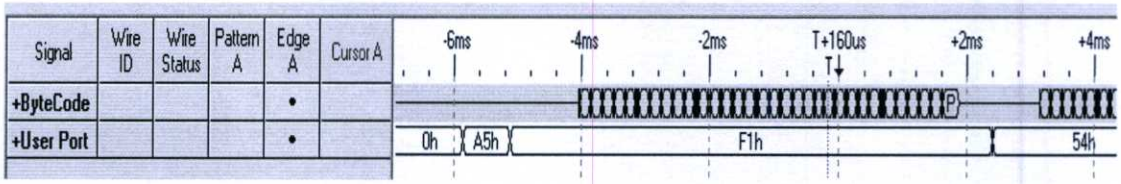


รูปที่ 4.33 ตัวตั้ง ตัวกระทำ และผลลัพธ์แสดงค่าออกทาง USERPORT

- ตัวอย่างโปรแกรมของลอจิก XOR เลข 2 จำนวน แสดงดังรูปที่ 4.34 ซึ่งเป็นการ XOR ค่า A5H ด้วย F1H ผลลัพธ์ที่ได้จะเก็บไว้ที่ USERTEMP หลังจากนั้นจะส่งค่าออกทาง USERPORT แสดงดังรูปที่ 4.35

User Code	BYTECODE
LABEL:	;00,00,06,64,00,00,05,00,01,
0xA5 xor 0xF1	;00,01,08,00,01,A5,4C,01,F1,4C,23,
OUTPORT(USERTEMP)	;00,02,03,00,04,4C,
GOTO LABEL	;00,03,05,65,00,00,05,00,

รูปที่ 4.34 โปรแกรมการ XOR เลข 2 จำนวน

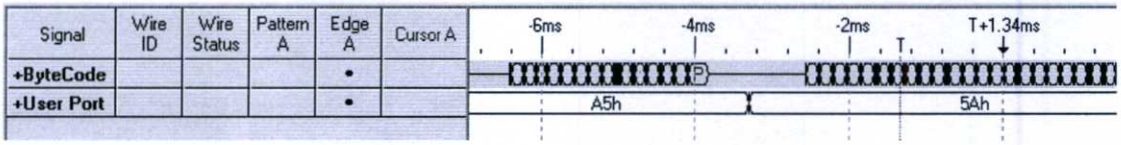


รูปที่ 4.35 ตัวตั้ง ตัวกระทำ และผลลัพธ์แสดงค่าออกทาง USERPORT

- ตัวอย่างโปรแกรมของลอจิก NOT 1 จำนวน แสดงดังรูปที่ 4.36 ซึ่งเป็นการ NOT ค่า A5H ผลลัพธ์ที่ได้จะเก็บไว้ที่ USERTEMP หลังจากนั้นจะส่งค่าออกทาง USERPORT แสดงดังรูปที่ 4.37

User Code	BYTECODE
LABEL:	;00,00,06,64,00,00,05,00,01,
Not 0xA5	;00,01,05,00,01,A5,4C,24,
OUTPORT(USERTEMP)	;00,02,03,00,04,4C,
GOTO LABEL	;00,03,05,65,00,00,05,00,

รูปที่ 4.36 โปรแกรมการ NOT เลข 1 จำนวน



รูปที่ 4.37 ตัวตั้ง และผลลัพธ์แสดงค่าออกทาง USERPORT

4.2.6 การทดสอบคำสั่ง IF 4 ระดับชั้น

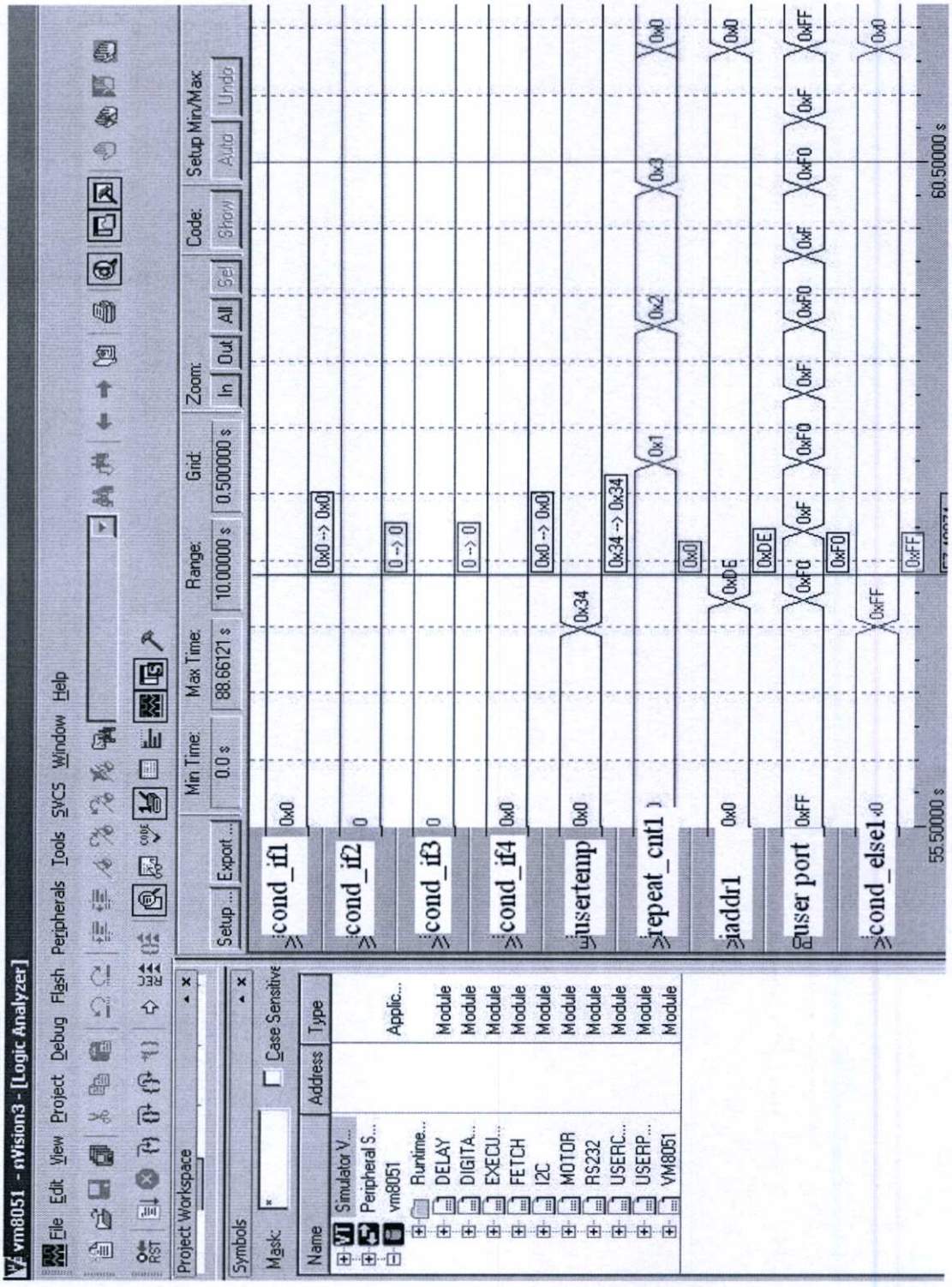
การทำงานของโปรแกรม ดังรูปที่ 4.38 ตัวโปรแกรมทำการตรวจสอบเงื่อนไขการรับค่าจาก พอร์ตอนุกรม 4 ค่า คือ ค่าของตัวอักษร 1, 2, 3, 4 ตามลำดับ โดยถ้าเงื่อนไขแรกไม่ใช่ 1 จะทำการส่งค่าออกพอร์ตขนาน 0FH และ F0H สลับกันวนรอบ 4 ครั้ง

```
[0] loop:
[1] rx232
[2] tx232
[3] if (usertemp=='1')
    {
[4]       output(1)
[5]       rx232
[6]       tx232
[7]       if (usertemp=='2')
          {
[8]                 output(2)
[9]                 rx232
[10]                tx232
[11]                if (usertemp=='3')
                    {
[12]                          output(4)
[13]                          rx232
[14]                          tx232
[15]                          if (usertemp=='4')
                              {
[16]                                    output(AA)
[17]                                    delay(10)
[18]                                    output(00)
[19]                              }

[20]                }

[21]      }
[22] }
[23] else
 {
[24] repeat 4
    {
[25]      delay(10)
[26]      outport(0f)
[27]      delay(10)
[28]      outport(f0)
[29] }
[30] outport(00)
[31] }
[32] goto loop
```

```
[ 0] 00,00,06,64,00,00,05,00,01,
[ 1] 00,01,02,00,14,
[ 2] 00,02,03,00,04,13,
[ 3] 00,03,09,01,00,03,00,04,01,31,1C,0A,
[ 4] 00,04,07,01,00,00,01,01,01,50,
[ 5] 00,05,05,01,00,00,02,14,
[ 6] 00,06,06,01,00,00,03,04,13,
[ 7] 00,03,09,02,00,03,00,04,01,32,1C,0A,
[ 8] 00,04,07,02,00,00,01,01,02,50,
[ 9] 00,05,05,02,00,00,02,14,
[10] 00,06,06,02,00,00,03,04,13,
[11] 00,03,09,03,00,03,00,04,01,33,1C,0A,
[12] 00,04,07,03,00,00,01,01,04,50,
[13] 00,05,05,03,00,00,02,14,
[14] 00,06,06,03,00,00,03,04,13,
[15] 00,0F,09,04,00,03,00,04,01,34,1C,0A,
[16] 00,10,07,04,00,00,01,01,AA,50,
[17] 00,11,07,04,00,00,02,01,10,10,
[18] 00,12,07,04,00,00,03,01,00,50,
[19] 00,13,04,84,00,00,00,
[20] 00,14,04,83,00,00,00,
[21] 00,15,04,82,00,00,00,
[22] 00,16,04,81,00,00,00,
[23] 00,17,05,09,00,04,00,00,
[24] 00,18,04,15,09,04,00,
[25] 00,19,07,09,00,00,01,01,05,10,
[26] 00,1A,07,09,00,00,02,01,0F,50,
[27] 00,1B,07,09,00,00,03,01,05,10,
[28] 00,1C,07,09,00,00,04,01,F0,50,
[29] 00,1D,04,C9,00,00,00,
[30] 00,1E,04,89,00,00,00,
[31] 00,1F,04,00,01,00,50,
[32] 00,20,05,65,00,00,05,00
```

รูปที่4.38 โปรแกรมและไบท์โค้ด IF 4 ระดับชั้น

แต่เนื่องจากการทำงานของคำสั่งเงื่อนไข เป็นการกระทำภายใน VM ซึ่งไม่สามารถมองเห็นผลลัพธ์ภายในได้ จึงต้องจำลองการทำงานเพื่อดูผลการทำงานภายในแทน แสดงดังรูปที่ 4.39



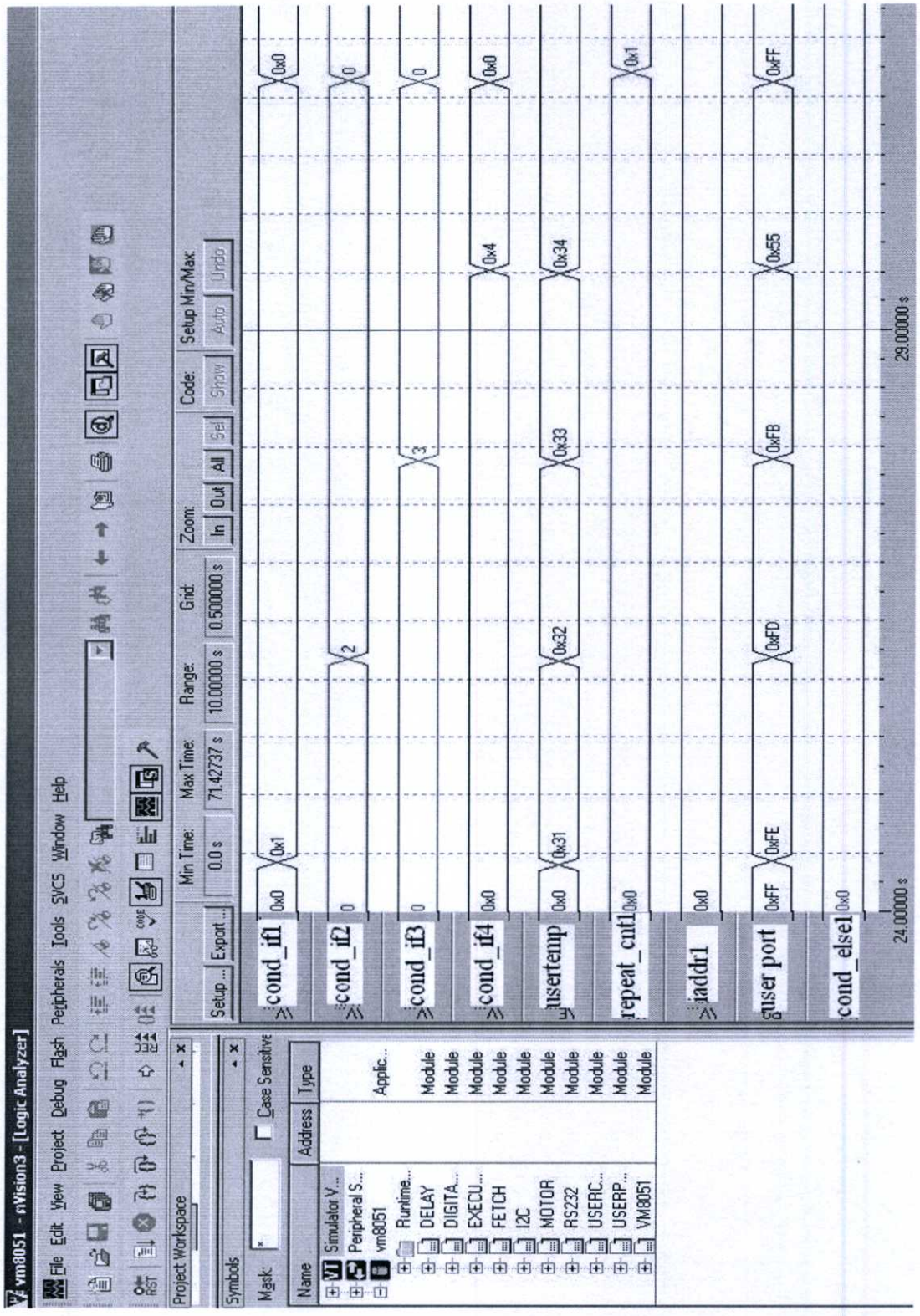
รูปที่ 4.39 การจำลองการทำงานของ IF 4 ระดับชั้น ในกรณี เงื่อนไข ELSE เป็นจริง

รูปที่ 4.39 แสดงผลจำลองการทำงานเมื่อคำสั่ง ELSE เป็นจริง โดยให้ผลลัพธ์การทำงานเป็นไปตามโปรแกรมที่เขียนขึ้นดังนี้

- *cond_else1* มีค่าเท่ากับ FFH แสดงว่าเงื่อนไขเป็นจริง
- *cond_if1, cond_if2, cond_if3* และ *cond_if4* จะเป็น 0 แสดงว่าเงื่อนไขเป็นเท็จ
- ค่าของ *repeat_cnt1* จะเพิ่มค่าจาก 0-3 เนื่องจาก โปรแกรมที่เขียนขึ้น กำหนดให้วนรอบการทำงานเท่ากับ 4
- ค่าของ *iaddr1* เท่ากับ DEH ใช้รีโหลดให้กับ PC เมื่อจบการวนรอบของคำสั่ง *REPEAT*

รูปที่ 4.40 แสดงผลจำลองการทำงานเมื่อคำสั่ง IF เป็นจริง โดยให้ผลลัพธ์การทำงานเป็นไปตามโปรแกรมที่เขียนขึ้นดังนี้

- เริ่มต้นค่าที่รับเข้ามาจากพอร์ตอนุกรมคือ '1' ซึ่งจะมีค่าเท่ากับ 31H โดยค่าจะเก็บไว้ที่ *USERTEMP* ถัดมาเป็นตัวแปร *cond_if1* กำหนดให้เป็น 1 จากนั้นทำการเขียนค่า 01H ไปที่พอร์ตขนาน ในที่นี้มีค่าเท่ากับ FEH เนื่องจาก เอาต์พุตได้ต่อ LED แบบพูลอัพกับไฟเลี้ยงไว้
- ถัดมาค่าที่รับเข้ามาจากพอร์ตอนุกรมคือ '2' ซึ่งจะมีค่าเท่ากับ 32H โดยค่าจะเก็บไว้ที่ *USERTEMP* ถัดมาเป็นตัวแปร *cond_if2* กำหนดให้เป็น 2 จากนั้นทำการเขียนค่า 02H ไปที่พอร์ตขนาน ในที่นี้มีค่าเท่ากับ FDH เนื่องจาก เอาต์พุตได้ต่อ LED แบบพูลอัพกับไฟเลี้ยงไว้
- ถัดมาค่าที่รับเข้ามาจากพอร์ตอนุกรมคือ '3' ซึ่งจะมีค่าเท่ากับ 33H โดยค่าจะเก็บไว้ที่ *USERTEMP* ถัดมาเป็นตัวแปร *cond_if3* กำหนดให้เป็น 3 จากนั้นทำการเขียนค่า 03H ไปที่พอร์ตขนาน ในที่นี้มีค่าเท่ากับ FBH เนื่องจาก เอาต์พุตได้ต่อ LED แบบพูลอัพกับไฟเลี้ยงไว้
- ถัดมาค่าที่รับเข้ามาจากพอร์ตอนุกรมคือ '4' ซึ่งจะมีค่าเท่ากับ 34H โดยค่าจะเก็บไว้ที่ *USERTEMP* ถัดมาเป็นตัวแปร *cond_if4* กำหนดให้เป็น 4 จากนั้นทำการเขียนค่า AAH ไปที่พอร์ตขนาน ในที่นี้มีค่าเท่ากับ 55H แล้วทำการหน่วงเวลา 1 วินาที ก่อนที่จะเขียนค่าไปที่พอร์ตขนานเท่ากับ 00H
- *cond_else1* มีค่าเท่ากับ 0 โดยเงื่อนไขที่ทำการตรวจสอบ ไม่เป็นจริง
- ค่าของ *iddr1* เท่ากับ 0 เนื่องจากการทำงานของ *REPEAT* จะทำงานก็ต่อเมื่อ ELSE เป็นจริงเท่านั้น



รูปที่ 4.40 การจำลองการทำงานของ IF 4 ระดับชั้น ในกรณี เงื่อนไข IF เป็นจริง

4.2.7 การทดสอบคำสั่ง REPEAT 3 ระดับชั้น

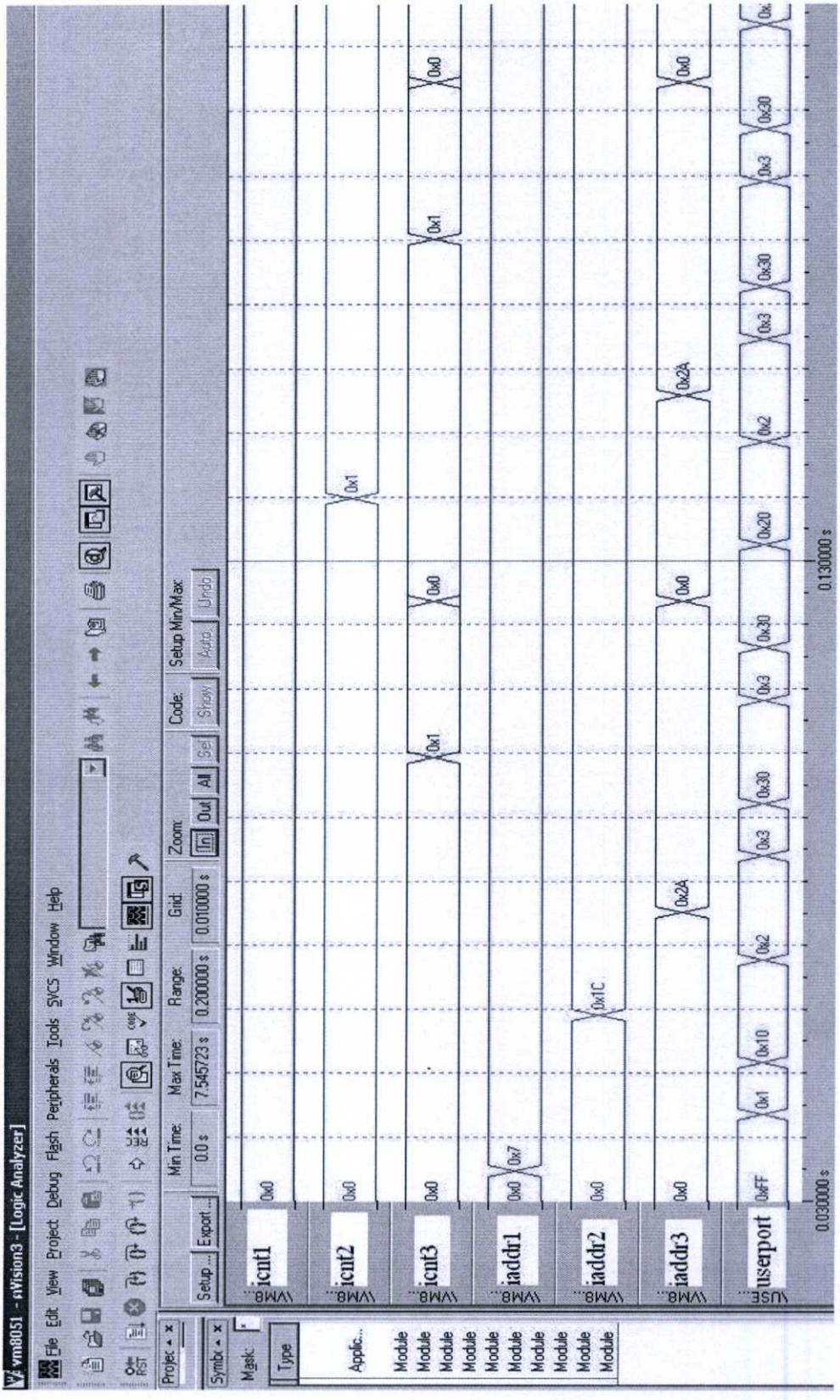
ตัวอย่างการทำงานของคำสั่ง REPEAT แสดงดังรูปที่ 4.41 ผลการจำลองการทำงานของโปรแกรมแสดงดังรูปที่ 4.42 และ รูปที่ 4.43 สามารถอธิบายการทำงานได้ดังนี้

- *icnt1*, *icnt2* และ *icnt3* เป็น Counter ของคำสั่ง *REPEAT*
- *iaddr1*, *iaddr2* และ *iaddr3* มีหน้าที่เก็บค่าของ PC ที่ต้องการทำสั่งวนรอบ ในที่นี้มีค่า 07H, 1CH และ 2AH ตามลำดับ
- *USERPORT* เป็นตัวแสดงผลการทำงานในแต่ละวงรอบของการทำ *REPEAT*

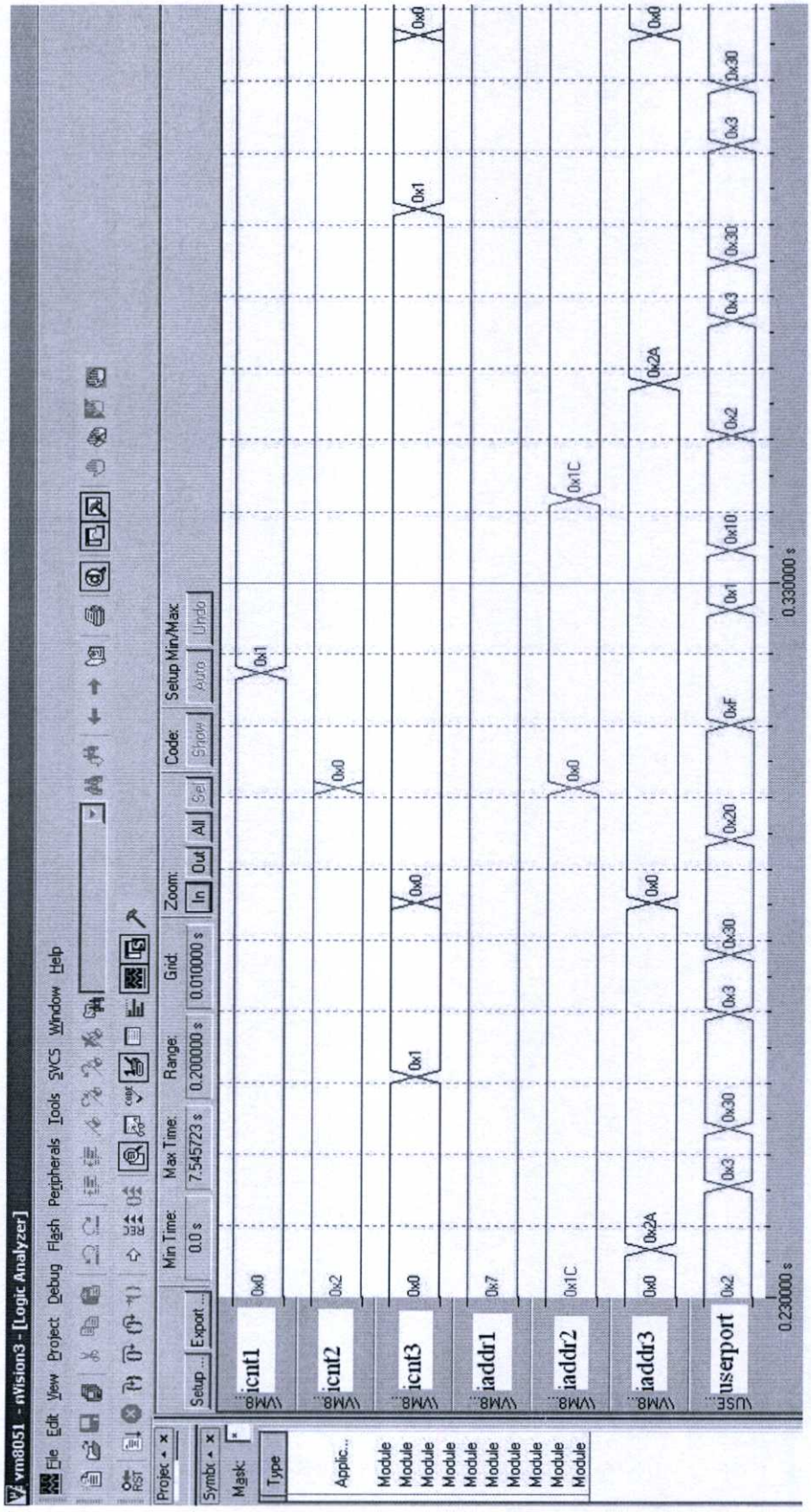
```
[0] repeat 2
        {
[1]     outport(01)
[2]     outport(10)

[3]     repeat 3
                {
[4]             outport(02)
[5]                     repeat 2
                        {
[6]                             outport(03)
[7]                             outport(30)
[8]                     }
[9]             outport(20)
[10]            }
[11]            outport(0F)
[12]    }
```

```
[0] 00,00,04,15,00,02,00,
[1] 00,01,04,00,01,01,4C,
[2] 00,02,04,00,01,10,4C,
[3] 00,03,04,16,00,03,00,
[4] 00,04,04,00,01,02,4C,
[5] 00,05,04,17,00,02,00,
[6] 00,06,04,00,01,03,4C,
[7] 00,07,04,00,01,30,4C,
[8] 00,08,04,CB,00,00,00,
[9] 00,09,04,00,01,20,4C,
[10] 00,0A,04,CA,00,00,00,
[11] 00,0B,04,00,01,0F,4C,
[12] 00,0C,04,C9,00,00,00,
```

รูปที่ 4.41 โปรแกรมและไบท์โค้ดของ REPEAT 3 ระดับชั้น



รูปที่ 4.42 การจำลองการทำงานของ REPEAT 3 ระดับชั้น



รูปที่ 4.43 การจำลองการทำงานของ REPEAT 3 ระดับชั้น (ต่อ)

4.2.8 การทดสอบ LCD

ตัวอย่างการทำงานของ LCD แสดงดังรูปที่ 4.44 ผลการทำงานของโปรแกรม แสดงดังรูปที่ 4.45

```
[0]   lcdinit
[1-2] lcd('12345678abcdefgh')
[3]   loop:
[4]   goto loop;

[0] 00,00,02,00,A2,
[1] 00,01,19,00,01,31,A0,01,32,A0,01,33,A0,01,34,A0,01,35,A0,01,36,A0,01,37,A0,01,38,A0,
[2] 00,01,1A,00,A5,01,41,A0,01,62,A0,01,63,A0,01,64,A0,01,65,A0,01,66,A0,01,67,A0,01,68,A0,
[3] 00,03,06,64,00,00,05,00,01,
[4] 00,04,05,65,00,00,05,00
```

รูปที่ 4.44 โปรแกรมทดสอบ LCD

รูปที่ 4.45 ผลการทดสอบ LCD

4.3 เปรียบเทียบขนาด Code Size และความเร็วในการทำงานระหว่าง Byte code กับภาษาซี

หัวข้อนี้เป็นการเปรียบเทียบผลการทำงานของระหว่างโปรแกรมที่เขียนขึ้นจาก Byte code และโปรแกรมที่เขียนขึ้นจากภาษาซี (None Byte code) โดยใช้ AT89C5131 ทดสอบการทำงาน ส่วนคอมไพล์เลอร์ของภาษาซีที่ใช้คือ KEIL

ตัวอย่างโปรแกรมรูปที่ 4.46 ซึ่งเขียนโดย Byte code โปรแกรมรูปที่ 4.47 และโปรแกรมรูปที่ 4.48 เขียนโดยภาษาซี ทำการเปรียบเทียบขนาดของหน่วยความจำที่ใช้ในการโปรแกรมได้ดังนี้

- โปรแกรมโดย Byte code ใช้พื้นที่ 28 Bytes
- โปรแกรมโดยภาษาซี AT89C5131ใช้พื้นที่ 110 Bytes แสดงดังรูปที่ 4.49
- โปรแกรมโดยภาษาซี PIC18F4620ใช้พื้นที่ 96 Bytes แสดงดังรูปที่ 4.50

สัญญาณรูปที่ 4.51 ได้จากโปรแกรมโดย Byte code สัญญาณรูปที่ 4.52 และสัญญาณรูปที่ 4.53 เป็นสัญญาณจากโปรแกรมโดย ภาษาซีเวลาที่ใช้ในการรันโปรแกรมได้ดังนี้

- โปรแกรมโดย Byte code ใช้เวลาในการทำงานประมาณ 8.4mS
- โปรแกรมโดยภาษาซี AT89C5131 ใช้เวลาในการทำงานประมาณ 2.1mS
- โปรแกรมโดยภาษาซี PIC18F4620 ใช้เวลาในการทำงานประมาณ 2.05mS

```
User Code                       BYTECODE

LABEL:                          ;00,00,06,64,00,00,05,00,01,
RX232                           ;00,01,02,00,14,
TX232(USERTEMP)                 ;00,02,03,00,04,13,
GOTO LABEL                      ;00,03,05,65,00,00,05,00,
```

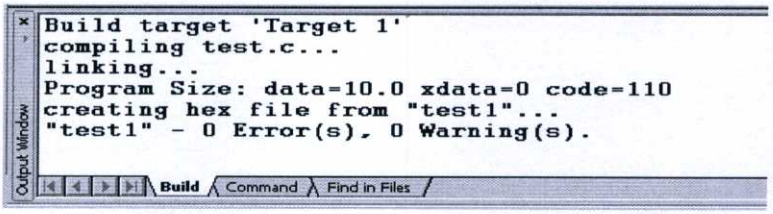
รูปที่ 4.46 โปรแกรมรับส่งข้อมูล 1 byte จาก RS-232 โดยใช้ Byte code

```
#include "reg_C51.h"
#include <intrins.h>
#include <stdio.h>

void initsystem(void)
{
        EA=1;EX0=1;IT0=1;
        SCON=0x50;
        TMOD = 0x21 ; /* Timer 1 in mode 2 and Timer0 mode1*/
        TH1 = 0xF3; /* 9600 Bds at 24MHz */
        TL1 = 0xF3; /* 9600 Bds at 24MHz */
        PCON = 0x80;
        TR1 = 1; /* Timer 1 run */
        TI=1;

}

tx232(unsigned char txdata)
{

putchar(txdata);

}

unsigned char rx232()
{

        return(_getkey());
}

void main (void)
{
        char mychar;

        initsystem();

        while(1)
        {
                mychar=rx232();
                tx232(mychar);
        }
}
```

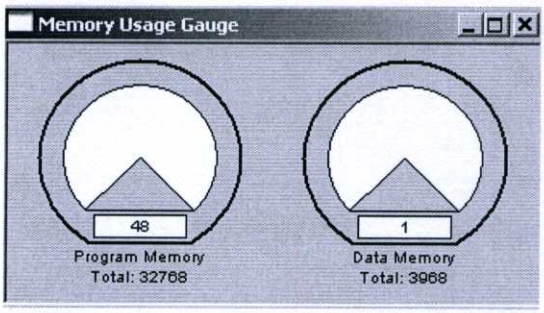
รูปที่ 4.47 โปรแกรมรับส่งข้อมูล 1 byte จาก RS-232 โดยใช้ภาษาซี AT89C5131

```
#if defined(__PCM__)
#include <16F877.h>
#fuses HS,NOWDT,NOPROTECT,NOLVP
#use delay(clock=20000000)
#use rs232(baud=9600, xmit=PIN_C6, rcv=PIN_C7)

#elif defined(__PCH__)
#include <18F4620.h>
#fuses HS,NOWDT,NOPROTECT,LVP
#use delay(clock=10000000)
#use rs232(baud=9600, xmit=PIN_C6, rcv=PIN_C7)
#endif
#include <stdio.h>

void tx232(int8 txdata)
{
        printf("%c",txdata);
}

int8 rx232(void)
{
        return(getchar());
}


void main(void)
{
        int mychar;
        while(1)
        {
                mychar=rx232();
                tx232(mychar);
        }
}
```

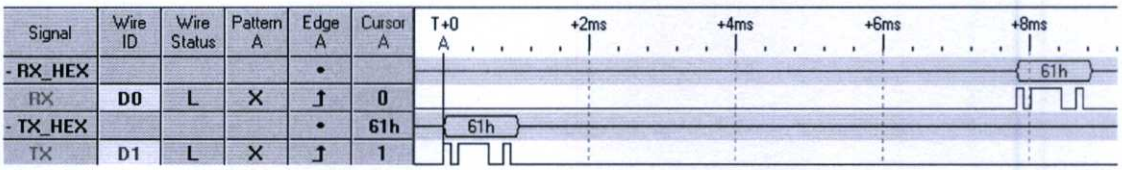
รูปที่ 4.48 โปรแกรมรับส่งข้อมูล 1 byte จาก RS-232 โดยใช้ภาษาซี PIC18F4620



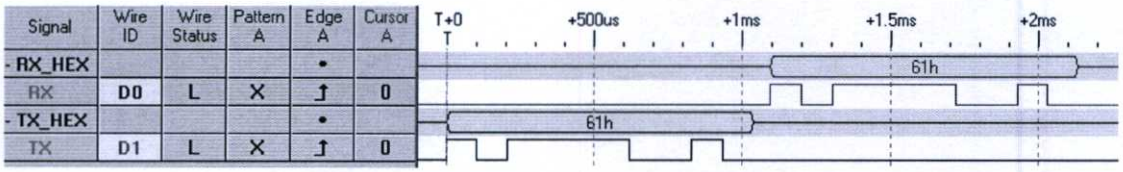
รูปที่ 4.49 ขนาดของ Code size ที่ได้จากการคอมไพล์ด้วย C-Compiler AT89C5131



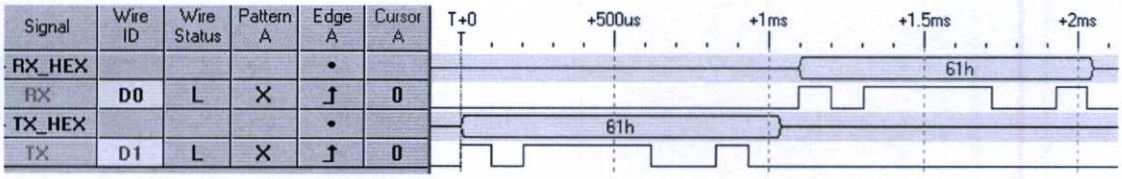
รูปที่ 4.50 ขนาดของ Code size ที่ได้จากการคอมไพล์ด้วย C-Compiler PIC18F4620



รูปที่ 4.51 สัญญาณรับส่งข้อมูล 1 byte จาก RS-232 โดยใช้ Byte code



รูปที่ 4.52 สัญญาณรับส่งข้อมูล 1 byte จาก RS-232 AT89C5131โดยใช้ภาษาซี



รูปที่ 4.53 สัญญาณรับส่งข้อมูล 1 byte จาก RS-232 PIC18F4620โดยใช้ภาษาซี

ตัวอย่างโปรแกรมรูปที่ 4.54 ซึ่งเขียนโดย Byte code และโปรแกรมรูปที่ 4.55 เขียนโดยภาษาซี ทำการเปรียบเทียบขนาดของหน่วยความจำที่ใช้ในการโปรแกรมได้ดังนี้

- โปรแกรมโดย Byte code ใช้พื้นที่ 73 Bytes
- โปรแกรมโดยภาษาซีใช้พื้นที่ 97 Bytes แสดงดังรูปที่ 4.56

สัญญาณรูปที่ 4.57 ได้จากโปรแกรมโดย Byte code และรูปที่ 4.58 เป็นสัญญาณจากโปรแกรมโดย ภาษาซีเวลาที่ใช้ในการรันโปรแกรมได้ดังนี้

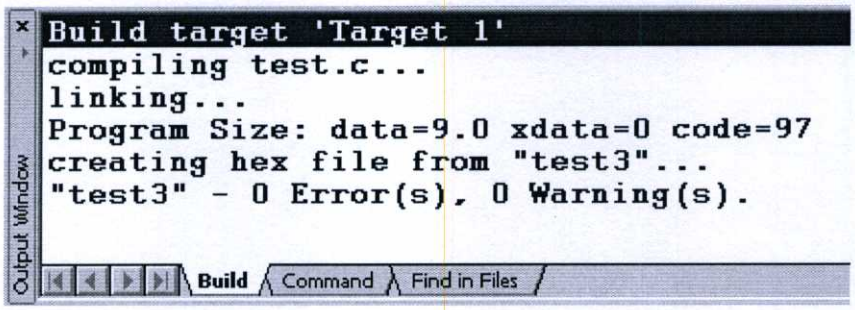
- โปรแกรมโดย Byte code ใช้เวลาในการทำงานประมาณ 460mS
- โปรแกรมโดยภาษาซี ใช้เวลาในการทำงานประมาณ 450mS

```
User Code                          Byte Code

LABEL:                             ;00,00,06,64,00,00,05,00,01,
OUTPORT(0x01)                      ;00,01,04,00,01,01,4C,
WAITTIME(1)                        ;00,02,04,00,01,01,10,
OUTPORT(0x02)                      ;00,03,04,00,01,02,4C,
WAITTIME(1)                        ;00,04,04,00,01,01,10,
OUTPORT(0x04)                      ;00,05,04,00,01,04,4C,
WAITTIME(1)                        ;00,06,04,00,01,01,10,
OUTPORT(0x08)                      ;00,07,04,00,01,08,4C,
WAITTIME(1)                        ;00,08,04,00,01,01,10,
GOTO LABEL                         ;00,11,05,65,00,00,05,00,
```

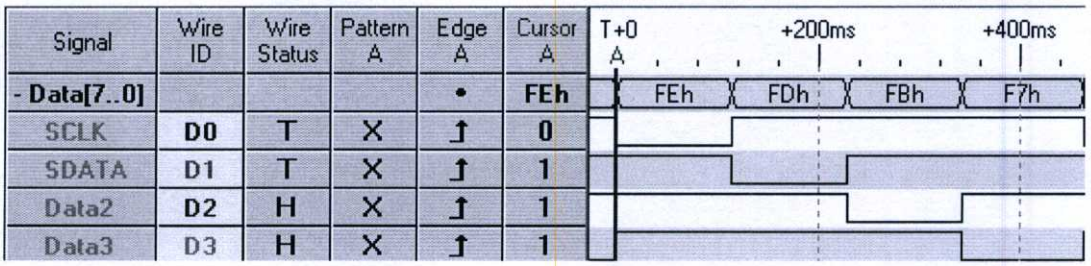
รูปที่ 4.54 โปรแกรมการส่งค่าออก USERPORT บิต 0-3 โดย Byte code

```
#include "reg_C51.h"
#include <intrins.h>
#include <stdio.h>

void delay(int delay_time)
{
        int i,loop;

        for(loop=0;loop<delay_time;loop++)
        {
                for(i=0;i<28;i++)
                {
                        TH0=0;
                        TL0=0;
                        TR0=1;                  //Start Timer
                        TF0=0;

                        while(1)
                        {
                          if (TF0==1) break;
                        }

                        TR0=0;                  //Stop Timer
                }
        }
}


void main (void)
{

        while(1)
        {

         P0=~0x01;
         delay(1);

         P0=~0x02;
         delay(1);

         P0=~0x04;
         delay(1);

         P0=~0x08;
         delay(1);

        }
}
```

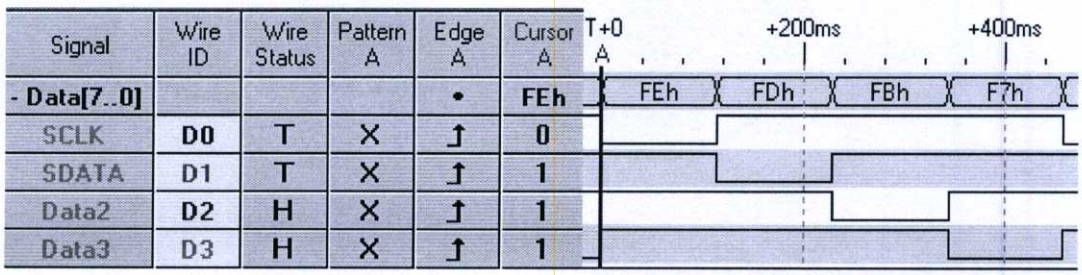
รูปที่ 4.55 โปรแกรมการส่งค่าออก USERPORT บิต 0-3 โดยภาษาซี



รูปที่ 4.56 ขนาดของ Code size ที่ได้จากการคอมไพล์ด้วย C-Compiler



รูปที่ 4.57 สัญญาณส่งค่าออก USERPORT บิต 0-3 โดย Byte code



รูปที่ 4.58 สัญญาณส่งค่าออก USERPORT บิต 0-3 โดยภาษาซี

ตัวอย่างโปรแกรมรูปที่ 4.59 ซึ่งเขียนโดย Byte code และ โปรแกรมรูปที่ 4.60 เขียนโดยภาษาซี ทำการเปรียบเทียบขนาดของหน่วยความจำที่ใช้ในการโปรแกรมได้ดังนี้

- โปรแกรมโดย Byte code ใช้พื้นที่ 23 Bytes
- โปรแกรมโดยภาษาซีใช้พื้นที่ 118 Bytes แสดงดังรูปที่ 4.61

สัญญาณรูปที่ 4.62 ได้จากโปรแกรมโดย Byte code และรูปที่ 4.63 ได้จากโปรแกรมโดยภาษาซี โดยเวลาที่ใช้ในการรันโปรแกรมได้ดังนี้

- โปรแกรมโดย Byte code ใช้เวลาในการทำงานประมาณ 2400uS
- โปรแกรมโดยภาษาซี ใช้เวลาในการทำงานประมาณ 580uS

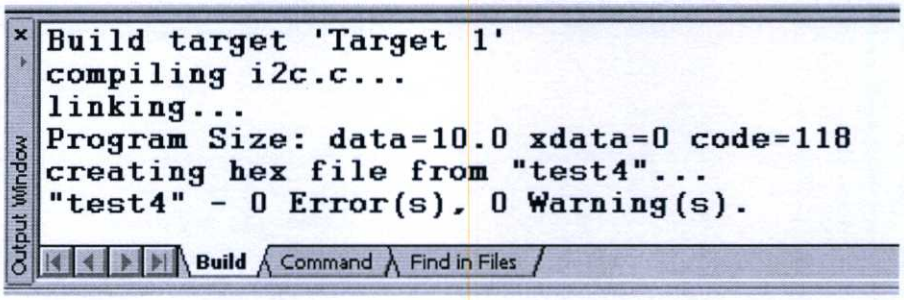
```
User Code                              BYTECODE

I2CSTART                        ;      00,00,02,00,4E,
I2CWRITE(0x5A,0x5B,0x5C)        ;      00,01,0A,00,01,5A,4A,01,5A,4A,01,5A,4A,
I2CSTOP                         ;      00,02,02,00,4F,
```

รูปที่ 4.59 โปรแกรมเขียนข้อมูล 3 ไบท์บน I^2C บัสโดย Byte code

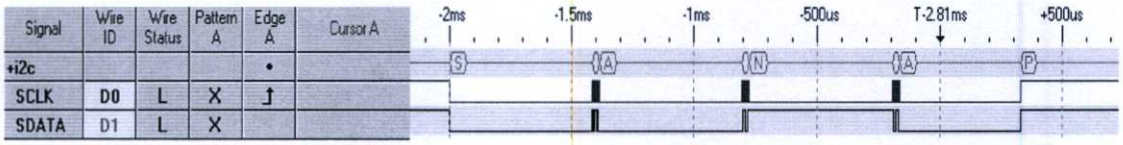
```
#include "vm8051config.h"

void i2c_start (void);

void i2c_stop (void);

void i2c_delay (void);

void i2c_write (unsigned char input_data);

void i2c_delay(void)
{
  _nop_ ();
  _nop_ ();
  _nop_ ();
  _nop_ ();
  _nop_ ();
  _nop_ ();
  _nop_ ();
  _nop_ ();

}

void i2cwrite(unsigned char i2c_data)
{
        i2c_write(i2c_data);

}

void i2c_start (void)
{
        SDATA = HIGH;

        SCLK = HIGH;

        SDATA = LOW;

        SCLK = LOW;

}
```

```
void i2c_stop (void)
{
        unsigned char input_var;

        SCLK = LOW;

        SDATA = LOW;

        SCLK = HIGH;

        SDATA = HIGH;

        input_var = SDATA;

}

void i2c_write (unsigned char output_data)
{
        unsigned char index;

        for(index = 0; index < 8; index++)
        {

                SDATA = ((output_data & 0x80) ?
1 : 0);
        output_data  <<= 1;
                SCLK = HIGH;

                i2c_delay();
                SCLK = LOW;
                i2c_delay();

        }

        index = SDATA;


        SDATA=1;

        SCLK = HIGH;

        i2c_delay();
        SCLK = LOW;
        i2c_delay();
}

void main()
{
        i2c_start();
        i2c_write(0x5a);
        i2c_write(0x5b);
        i2c_write(0x5c);
        i2c_stop();
        while(1);
}
```

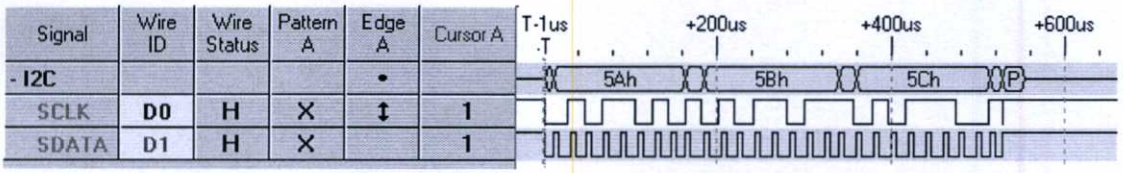
รูปที่ 4.60 โปรแกรมเขียนข้อมูล 3 ไบท์บน I^2C บัสโดยภาษาซี



รูปที่ 4.61 ขนาดของ Code size ที่ได้จากการคอมไพล์ด้วย C-Compiler



รูปที่ 4.62 สัญญาณเขียนข้อมูล 3 ไบท์บน I^2C บัสโดย Byte code



รูปที่ 4.63 สัญญาณเขียนข้อมูล 3 ไบท์บน I^2C บัสโดยภาษาซี

บทที่ 5
สรุปผลการวิจัยและข้อเสนอแนะ

5.1 สรุปผลการวิจัย

การประยุกต์ใช้งานเครื่องคอมพิวเตอร์เสมือนบนไมโครคอนโทรลเลอร์ เป็นอีกทางเลือกหนึ่ง ซึ่งช่วยให้การพัฒนาแอพพลิเคชั่นเป็นไปอย่างรวดเร็ว เนื่องจากระบบถูกออกแบบให้สามารถติดต่อกับอุปกรณ์รอบข้างพื้นฐานภายนอกได้เช่น RS-232, I^2C, LCD, Input ports และ Output ports รวมทั้งความสามารถในการควบคุมการทำงานของโปรแกรม ให้เป็นไปตามข้อกำหนดของผู้ใช้งาน ด้วยชุดคำสั่งเงื่อนไขที่มีเช่น If, Else, Repeat, Goto และ Waituntil รวมถึงผู้ใช้งานสามารถนำโปรแกรมของตนเอง ไปใช้กับไมโครคอนโทรลเลอร์ตระกูลอื่น ที่ทำการติดตั้งเครื่องคอมพิวเตอร์เสมือนได้โดยไม่เสียเวลาในการเขียนโปรแกรมใหม่

การทำงานของชุดคำสั่งทางคณิตศาสตร์ ลอจิก และการเปรียบเทียบ สามารถทำงานได้อย่างถูกต้อง โดยเครื่องคอมพิวเตอร์เสมือนอาศัยพื้นฐานการออกแบบด้วย Stack Based ที่มีความลึก 8 ระดับชั้น ประโยชน์ของการออกแบบด้วย Stack Based ทำให้ง่ายต่อการออกแบบชุดคำสั่ง Byte code เนื่องจากระบบสามารถกำหนดตำแหน่งของข้อมูลที่ต้องการเข้าถึงได้

ขนาดของ Byte code ที่ได้ทำการทดสอบ เทียบกับการเขียนโปรแกรมด้วยภาษาซี มีขนาดเล็กกว่าประมาณ 2-3 เท่า ทำให้ประหยัดพื้นที่ใช้งานของหน่วยความจำ (สามารถเขียนโปรแกรมได้มากกว่าเมื่อใช้หน่วยความจำที่มีขนาดเท่ากัน จากรูปที่ 4.46, 4.49 และ 4.50)

ความเร็วในการทำงานเฉลี่ยโดยรวมของโปรแกรม Byte code เทียบกับโปรแกรมภาษาซีช้ากว่าประมาณ 3-4 เท่า เนื่องจากเครื่องคอมพิวเตอร์เสมือนต้องทำการถอดรหัสชุดคำสั่ง Byte code ก่อน หลังจากนั้นจึงทำการติดต่อกับระบบจริงภายในของไมโครคอนเลอร์ ซึ่งแตกต่างจากภาษาซีที่สามารถติดต่อกับระบบจริงได้โดยตรง (จากรูปที่ 4.51, 4.52 และ 4.53)

5.2 ข้อเสนอแนะ

- ระบบรองรับการทำงานขนาด 8 Bits เพื่อรองรับการทำงานของไมโครคอนโทรลเลอร์ขนาด 16-32 Bits ได้ในอนาคต ผู้พัฒนาต่อต้องทำการปรับปรุงระบบเพิ่ม
- ชุดคำสั่ง Byte code บางคำสั่งสามารถตัดออกได้เพื่อลดการใช้พื้นที่ของหน่วยความจำโปรแกรม เช่น ในกรณีที่ไม่มีคำสั่งเงื่อนไขในบรรทัดนั้นเราสามารถตัดคำสั่ง Byte code ออกได้ 1 Byte
- เพื่อให้การใช้งานสะดวกขึ้น การพัฒนาคอมไพล์เลอร์ จึงมีความจำเป็นเพื่อใช้ในการตรวจสอบรูปแบบของชุดคำสั่ง Byte code

เอกสารอ้างอิง

[1] F.Matin, B.Mikhak, B.Silverman. "MetaCricket : A designer's kit for making computational devices" **IBM Systems Journal**, VOL 39, NOS 3&4, 2000

[2] Arnan (Roger) Sipitakiat. "**GOGO Board**." [Online]. Available : http://padthai.media.mit.edu:8080/cocoon/gogosite/home.xsp?lang=en.

[3] Pekka Nikander. "An Operating System in JAVA for the Lego Mindstorms RCX Microcontroller." **Proceedings of FREENIX Track : 2000 USENIX Annual Technical Conference**, June 13-23, 2000

[4] Lange S., Kebschull U. "Virtual hardware byte code as a design platform for reconfigurable embedded systems." **IEEE**, 2003

[5] นรากร จีนจันทร์. "การออกแบบไบท์โค้ดเฟิร์มแวร์บนไมโครคอนโทรลเลอร์" วิศวกรรมสาร มข. ปีที่ 34 ฉบับที่ 5 (535-546) กันยายน-ตุลาคม 2550

[6] Haqqar, Peter. "**Java bytecode**." [Online]. Available : http://www-128.ibm.com/developerworks/ibm/library/it-haggar_bytecode. 2001

[7] http://www.keil.com

[8] "**CCS**" [Online]. Available : http://www.ccsinfo.com. Customer Computer Service Inc. 2007.

[9] อรพิน ประวัติบริสุทธิ์. **คู่มือเขียนโปรแกรมด้วยภาษา JAVA**. Provision Co., Ltd., 2004

[10] Herbert Schildt. **C++: The Complete Reference.** 3rd Edition, 1998

[11]Dan Gookin. **C for Dummies.** 2nd Edition, 2004

ภาคผนวก ก.

รายละเอียดอุปกรณ์ AT89C5131

Features

- 80C52X2 Core (6 Clocks per Instruction)
  - Maximum Core Frequency 48 MHz in X1 Mode, 24MHz in X2 Mode
  - Dual Data Pointer
  - Full-duplex Enhanced UART (EUART)
  - Three 16-bit Timer/Counters: T0, T1 and T2
  - 256 Bytes of Scratchpad RAM
- 32-Kbyte On-chip Flash In-System Programming through USB or UART
- 4-Kbyte EEPROM for Boot (3-Kbyte) and Data (1-Kbyte)
- On-chip Expanded RAM (ERAM): 1024 Bytes
- USB 1.1 and 2.0 Full Speed Compliant Module with Interrupt on Transfer Completion
  - Endpoint 0 for Control Transfers: 32-byte FIFO
  - 6 Programmable Endpoints with In or Out Directions and with Bulk, Interrupt or Isochronous Transfers
    - Endpoint 1, 2, 3: 32-byte FIFO
    - Endpoint 4, 5: 2 x 64-byte FIFO with Double Buffering (Ping-pong Mode)
    - Endpoint 6: 2 x 512-byte FIFO with Double Buffering (Ping-pong Mode)
  - Suspend/Resume Interrupts
  - Power-on Reset and USB Bus Reset
  - 48 MHz DPLL for Full-speed Bus Operation
  - USB Bus Disconnection on Microcontroller Request
- 5 Channels Programmable Counter Array (PCA) with 16-bit Counter, High-speed Output, Compare/Capture, PWM and Watchdog Timer Capabilities
- Programmable Hardware Watchdog Timer (One-time Enabled with Reset-out): 50 ms to 6s at 4 MHz
- Keyboard Interrupt Interface on Port P1 (8 Bits)
- TWI (Two Wire Interface) 400Kbit/s
- SPI Interface (Master/Slave Mode)
- 34 I/O Pins
- 4 Direct-drive LED Outputs with Programmable Current Sources: 2-6-10 mA Typical
- 4-level Priority Interrupt System (11 sources)
- Idle and Power-down Modes
- 0 to 32 MHz On-chip Oscillator with Analog PLL for 48 MHz Synthesis
- Low Power Voltage Range
  - 3.0V to 3.6V
  - 30 mA Max Operating Current (at 40 MHz)
  - 100 µA Max Power-down Current
- Industrial Temperature Range
- Packages: PLCC52, VQFP64, MLF48, SO28

ATMEL®

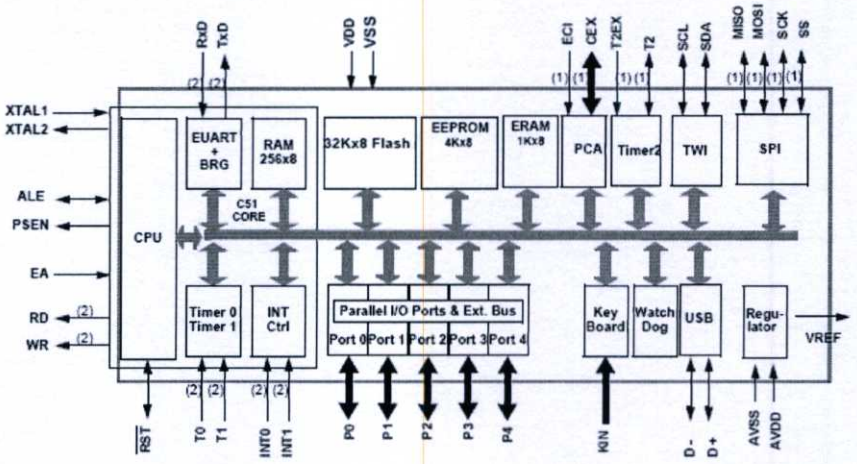
8-bit Flash Microcontroller with Full Speed USB Device

AT89C5131

Rev. 4136B–USB–09/03

ATMEL

Block Diagram



Notes: 1. Alternate function of Port 1
2. Alternate function of Port 3
3. Alternate function of Port 4

รายละเอียดอุปกรณ์ PIC18F4620

MICROCHIP **PIC18F2525/2620/4525/4620**

28/40/44-Pin Enhanced Flash Microcontrollers with 10-Bit A/D and nanoWatt Technology

Power Managed Modes:

- Run: CPU on, peripherals on
- Idle: CPU off, peripherals on
- Sleep: CPU off, peripherals off
- Idle mode currents down to 2.5 µA typical
- Sleep mode current down to 100 nA typical
- Timer1 Oscillator: 1.8 µA, 32 kHz, 2V
- Watchdog Timer: 1.4 µA, 2V typical
- Two-Speed Oscillator Start-up

Flexible Oscillator Structure:

- Four Crystal modes, up to 40 MHz
- 4x Phase Lock Loop (PLL) – available for crystal and internal oscillators)
- Two External RC modes, up to 4 MHz
- Two External Clock modes, up to 40 MHz
- Internal oscillator block:
  - 8 user selectable frequencies, from 31 kHz to 8 MHz
  - Provides a complete range of clock speeds from 31 kHz to 32 MHz when used with PLL
  - User tunable to compensate for frequency drift
- Secondary oscillator using Timer1 @ 32 kHz
- Fail-Safe Clock Monitor
  - Allows for safe shutdown if peripheral clock stops

Peripheral Highlights:

- High-current sink/source 25 mA/25 mA
- Three programmable external interrupts
- Four input change interrupts
- Up to 2 Capture/Compare/PWM (CCP) modules, one with Auto-Shutdown (28-pin devices)
- Enhanced Capture/Compare/PWM (ECCP) module (40/44-pin devices only):
  - One, two or four PWM outputs
  - Selectable polarity
  - Programmable dead time
  - Auto-Shutdown and Auto-Restart

Peripheral Highlights (Continued):

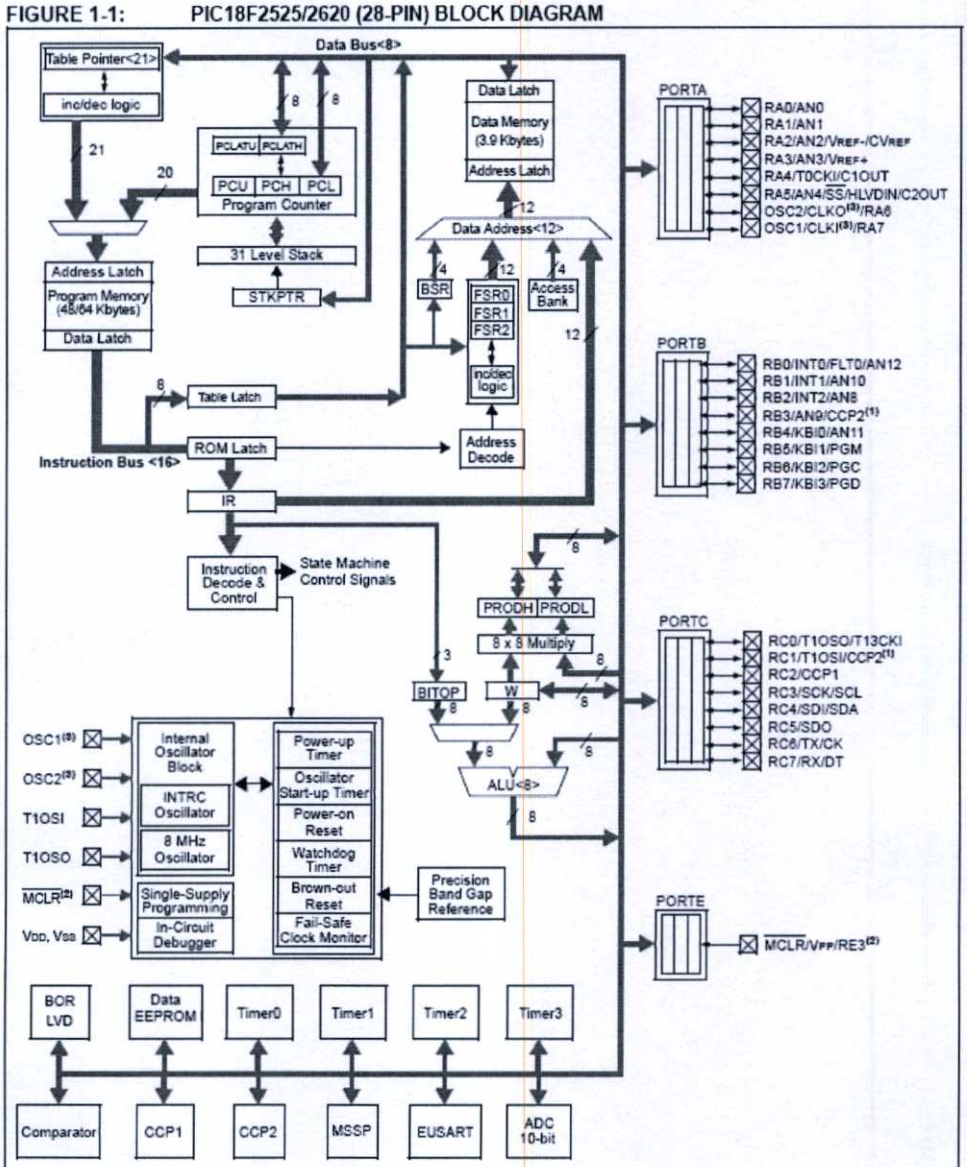
- Master Synchronous Serial Port (MSSP) module supporting 3-wire SPI™ (all 4 modes) and I²C™ Master and Slave modes
- Enhanced Addressable USART module:
  - Supports RS-485, RS-232 and LIN 1.2
  - RS-232 operation using internal oscillator block (no external crystal required)
  - Auto-Wake-up on Start bit
  - Auto-Baud Detect
- 10-bit, up to 13-channel Analog-to-Digital Converter module (A/D):
  - Auto-acquisition capability
  - Conversion available during Sleep
- Dual analog comparators with input multiplexing
- Programmable 16-level High/Low-Voltage Detection (HLVD) module:
  - Supports interrupt on High/Low-Voltage Detection

Special Microcontroller Features:

- C compiler optimized architecture:
  - Optional extended instruction set designed to optimize re-entrant code
- 100,000 erase/write cycle Enhanced Flash program memory typical
- 1,000,000 erase/write cycle Data EEPROM memory typical
- Flash/Data EEPROM Retention: 100 years typical
- Self-programmable under software control
- Priority levels for interrupts
- 8 x 8 Single Cycle Hardware Multiplier
- Extended Watchdog Timer (WDT):
  - Programmable period from 4 ms to 131s
- Single-supply 5V In-Circuit Serial Programming™ (ICSP™) via two pins
- In-Circuit Debug (ICD) via two pins
- Wide operating voltage range: 2.0V to 5.5V
- Programmable Brown-out Reset (BOR) with software enable option

Device	Program Memory		Data Memory		I/O	10-bit A/D (ch)	CCP/ ECCP (PWM)	MSSP		EUSART	Comp.	Timers 8/16-bit
	Flash (bytes)	# Single-Word Instructions	SRAM (bytes)	EEPROM (bytes)				SPI™	Master I²C™			
PIC18F2525	48K	24576	3986	1024	25	10	2/0	Y	Y	1	2	1/3
PIC18F2620	64K	32768	3986	1024	25	10	2/0	Y	Y	1	2	1/3
PIC18F4525	48K	24576	3986	1024	36	13	1/1	Y	Y	1	2	1/3
PIC18F4620	64K	32768	3986	1024	36	13	1/1	Y	Y	1	2	1/3

FIGURE 1-1: PIC18F2525/2620 (28-PIN) BLOCK DIAGRAM



Note 1: CCP2 is multiplexed with RC1 when configuration bit CCP2MX is set, or RB3 when CCP2MX is not set.

2: RE3 is only available when $\overline{\text{MCLR}}$ functionality is disabled.

3: OSC1/CLKI and OSC2/CLKO are only available in select oscillator modes and when these pins are not being used as digital I/O. Refer to **Section 2.0 "Oscillator Configurations"** for additional information.

ภาคผนวก ข.

ผลงานวิจัยที่ได้รับตีพิมพ์

1. นรากร จีนจันทร์, *"การออกแบบไบท์โค้ดเฟิร์มแวร์บนไมโครคอนโทรลเลอร์"*, วิศวกรรมสาร มข. ปีที่ 34 ฉบับที่ 5 (535-546) กันยายน-ตุลาคม 2550
2. Suchart Khuntaweetep and Narakorn Jeenjun , "*Byte Code Interpreter for 8051 Microcontroller*", วารสารวิชาการพระนครเหนือ ปีที่ 19 ฉบับที่ 3 เดือนกันยายน-ธันวาคม 2552

ประวัติผู้เขียน

ชื่อ	นรากร จีนจันทร์
เกิดวันที่	5 มกราคม 2523
การศึกษา	ปีการศึกษา 2538-2541
	ระดับประกาศนียบัตรวิชาชีพ วิทยาลัยเทคนิคลำปาง
	ปีการศึกษา 2541-2544
	ระดับปริญญาตรีวิศวกรรมศาสตร์บัณฑิต สาขาวิศวกรรมอิเล็กทรอนิกส์
	(เกียรตินิยมอันดับ 2) มหาวิทยาลัยเทคโนโลยีมหานคร
ความชำนาญเฉพาะด้าน	การออกแบบซอฟต์แวร์และฮาร์ดแวร์สำหรับไมโครคอนโทรลเลอร์
ปัจจุบัน	วิศวกรอาวุโส บริษัทสแปนชั่น ไทยแลนด์ จำกัด