ปีการศึกษา 2531

การเพิ่มความเร็วในการประมวลผลด้วย TMS32010

โดย

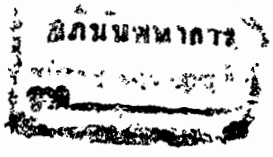
นาย เวธน์ แซ่อึ้ง

นาย สงวน ศาศวัตวิบูลย์

อาจารย์ที่ปรึกษา

รศ. วิพันธ์ ปรีชาพานิช

ศจ. ดร. ไพรัช ธัชยพงษ์



ปริญญานิพนธ์ปีการศึกษา 2531

ภาควิชาระบบควบคุม

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่องการเพิ่มความเร็วในการประมวลผลด้วย TMS32010

ผู้จัดทำ

1	เวธน์	แซ่อึ้ง	281235
2	สงวน	ศาศวัตวิบูลย์	281247

..... อาจารย์ที่ปรึกษา

(รศ. วิพันธ์ ปรีชาพานิช)

..... อาจารย์ที่ปรึกษา

(ศจ. ดร. ไพรัช ธัชยพงษ์)

การเพิ่มความเร็วในการประมวลผลด้วย TMS32010

เวธน์ แซ่อึ้ง
สงวน ศาศวัตวิบูลย์
รศ. วิพันธ์ ปรีชาพานิช อาจารย์ที่ปรึกษา
ศจ.ดร. ไพรัช ธัชยพงษ์ อาจารย์ที่ปรึกษา
ปีการศึกษา 2531

บทคัดย่อ

ปริญญานิพนธ์ฉบับนี้ จัดทำขึ้นจากโครงงานการประมวลผลความเร็วสูงด้วย TMS32010 เพื่อช่วยให้ไมโครคอมพิวเตอร์ไอบีเอ็มพีซีมีประสิทธิภาพในการประมวลสัญญาณเชิงดิจิตัล(digital signal processing) โดยการออกแบบให้ดิจิตัลซิกแนลโปรเซสเซอร์(digital signal processor) TMS32010 ทำงานร่วมกับไมโครโปรเซสเซอร์(microprocessor) 8088 ในไมโครคอมพิวเตอร์ไอบีเอ็มพีซีได้ โครงงานนี้ได้จัดสร้างส่วนฮาร์ดแวร์(hardware) ให้มีการทำงานเป็นอาร์เรย์โปรเซสซิง(array processing) โดยที่ส่วนฮาร์ดแวร์จะประกอบไปด้วย อินพุท-เอาท์พุทพอร์ท (input-output port) หน่วยความจำสำหรับข้อมูลที่จะให้ TMS32010 ประมวลผลและหน่วยความจำข้อมูล(program memory) ของ TMS32010 รวมถึงซอฟท์แวร์ (software) ที่จะสามารถสั่งงานให้การ์ดนี้ทำงานได้ การกำหนดอัลกอริธึม (algorithm) สำหรับการประมวลสัญญาณเชิงดิจิตัล

HIGH SPEED CALCULATING WITH TMS32010

Wathe Sae-Oung

Sanguan Sadsawatvibul

Associate Professor Vipan Preejapanij Advisor

Professor Dr. Pairash Thatchayapong Advisor

1988

Abstract

This thesis is an application of TMS32010 digital signal processor to increase the efficiency of IBM PC microcomputer in computation. By designing interface circuit which can control TMS32010 to be co-processor of IBM PC. The interface circuit provide input-output port, program memory for TMS32010 and control circuit. Afterword it will mention about system software that can operate this system and give an application of this project.

สารบัญ

บทที่ 1

บทนำ

ปริญญานิพนธ์ฉบับนี้ได้รวบรวมทฤษฎีและการทดลองในหัวข้อเรื่องการเพิ่มความเร็วในการประมวลผลด้วย TMS32010(HIGH SPEED CALCULATING WITH TMS32010) ซึ่งเป็นการประยุกต์ใช้ดิจิตอลซิกแนลโปรเซสเซอร์(digital signal processor) ของบริษัทเท็กซัสอินสตรูเมนท์ตระกูล 320 เบอร์ TMS32010 โดยการต่อระบบเป็นการ์ดที่ใช้เสียบลงในสล็อตของไอบีเอ็มพีซีไมโครคอมพิวเตอร์ เพื่อเพิ่มความเร็วในการประมวลสัญญาณเชิงดิจิตอล(digital signal processing) รายละเอียดจะกล่าวถึงทฤษฎีเบื้องต้นของการประมวลสัญญาณเชิงดิจิตอล, การออกแบบฮาร์ดแวร์(hardware)และซอฟท์แวร์(software)ของตัวระบบตลอดจนผลการทดลองที่ได้ นอกจากนี้ยังได้ยกตัวอย่างทฤษฎีและการออกแบบโปรแกรมของงานที่ประยุกต์ใช้กับ TMS32010 อันได้แก่การสร้างสัญญาณคลื่นรูปซายน์(sine-wave generator)

เนื่องจากโครงงานนี้เป็นโครงงานตลอดปีการศึกษา 2531 ดังนั้นในภาคการศึกษาแรกจึงได้ทำการออกแบบและพัฒนาระบบฮาร์ดแวร์ ตลอดจนทำการตรวจสอบการทำงานของระบบฮาร์ดแวร์ที่ออกแบบไว้ด้วย โดยใช้โปรแกรมดีบัก(debug) ตรวจสอบการรับส่งอินพุทเอาท์พอร์ท(input-output port) ทดสอบการย้ายข้อมูลในหน่วยความจำของระบบในละแบงค์(bank) ส่วนในภาคการศึกษาที่สองนั้น ได้ทำการพัฒนาระบบซอฟท์แวร์โดยเขียนเป็นโปรแกรมควบคุมการทำงานของระบบฮาร์ดแวร์นี้ โดยใช้ภาษาแอสเซมบลี(assembly) ลิงค์กับภาษาซี(C language) ซึ่งส่วนที่เกี่ยวข้องกับการติดต่อกับระบบจะเป็นหน้าที่ของภาษาแอสเซมบลี แต่ส่วนที่จัดการเกี่ยวกับจอภาพการแสดงผลออกทางจอก็จะใช้ภาษาซีซึ่งมีความคล่องตัวกว่า โปรแกรมควบคุมการทำงานของระบบนี้ใช้ชื่อว่า ST.EXE มีหน้าที่สั่งการให้ TMS32010 ทำงานและหยุดการทำงานเมื่อ TMS32010 ทำงานเสร็จ หลังจากที่ได้พัฒนาซอฟท์แวร์ในส่วนของโปรแกรมควบคุมการทำงานของระบบเสร็จ จึงได้เริ่มพัฒนาในส่วนของโปรแกรมประยุกต์ใช้งานของ TMS32010 โดยเริ่มจากโปรแกรมให้

TMS32010 ย้ายค่าข้อมูลลงในหน่วยความจำ แล้วทำการตรวจผลการทำงานซึ่งก็ได้ผลถูกต้อง ต่อจากนั้นจึงได้เริ่มพัฒนาโปรแกรมที่ใช้งานจริงๆ คือโปรแกรมสร้างสัญญาณคลื่นรูปซายน์ แล้วทำการทดลองจับเวลาการประมวลผลระหว่าง TMS32010 กับไอบีเอ็มพีซีที่ใช้ไมโครโปรเซสเซอร์(microprocessor) เบอร์ 8088 ของอินเทล และกับเครื่องคอมแพค(COMPAQ-DESKPRO 386/20) โดยที่ผลการทดลองได้แสดงเป็นตารางไว้แล้วในปริญญานิพนธ์ฉบับนี้

เนื้อหาในปริญญานิพนธ์เล่มนี้ได้แบ่งออกเป็น 5 บท และมีภาคผนวกแสดงข้อมูลเฉพาะของดิจิตอลซิกแนลโปรเซสเซอร์, วงจรฮาร์ดแวร์ของระบบ ในบทที่ 2 จะกล่าวถึงทฤษฎีการประมวลสัญญาณเชิงดิจิตอล, ทฤษฎีการคูณประสาน, ทฤษฎีการสร้างสัญญาณคลื่นรูปซายน์, ทฤษฎีการแทนตัวเลขทศนิยมในระบบต่างๆและผลของความผิดพลาดเนื่องจากการแทนตัวเลขทศนิยมจำกัด, การประมวลผลทางคณิตศาสตร์กับระบบเลขทศนิยมรู้จบ ในบทที่ 3 จะกล่าวถึงการออกแบบระบบทั้งด้านฮาร์ดแวร์และซอฟท์แวร์ บทที่ 4 แสดงถึงการทดสอบและผลการจับเวลา และทำการสรุปผลที่ได้ในบทที่ 5

ปริญญานิพนธ์ฉบับนี้ผู้จัดทำหวังว่าจะมีประโยชน์และให้ข้อมูลสำหรับนักศึกษา, ผู้สนใจที่จะทำโครงงานด้านนี้ต่อไปหรือจะนำโครงงานนี้ไปประยุกต์ใช้กับงานในสาขาต่างๆ

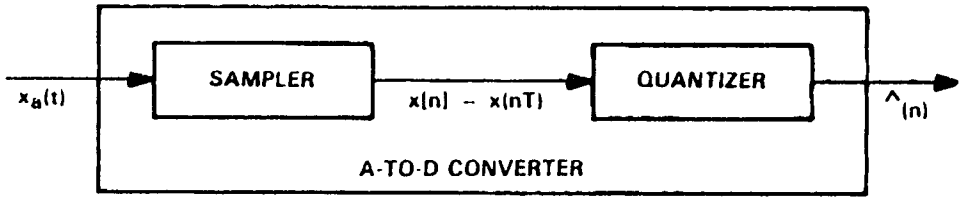
บทที่ 2

ทฤษฎีเบื้องต้นของการประมวลสัญญาณเชิงดิจิตอล

การประมวลสัญญาณเชิงดิจิตอล(digital signal processing DSP) จะเริ่มที่การสุ่มสัญญาณ(sampling) เพื่อทำให้สัญญาณที่เป็นสัญญาณอนาล็อก(analog) หรือสัญญาณต่อเนื่อง(continuous) เปลี่ยนเป็นสัญญาณที่มีลักษณะเป็นช่วงๆ เป็นลำดับหรือสัญญาณดิสครีต(discrete signal)โดยผ่านตัวแปลงสัญญาณอนาล็อกเป็นสัญญาณดิจิตอลหรือเอดีซี (analog-to-digital converter ADC) หลังจากนั้นก็จะใช้ลำดับสัญญาณที่ได้จากการสุ่มนี้ผ่านกระบวนการคำนวณเชิงตัวเลข(numerical computation procedure) เพื่อให้ได้ผลลัพธ์ตามที่ต้องการเป็นเอาท์พุท ต่อจากนั้น ก็จะเปลี่ยนเอาท์พุท กลับไปเป็นสัญญาณอนาล็อกเหมือนเดิม โดยใช้ตัวแปลงสัญญาณดิจิตอลเป็นสัญญาณอนาล็อกหรือดีเอซี (digital-to-analog converter DAC) ขั้นตอนดังกล่างนี้แสดงให้เห็นดังรูปที่ 2.1

รูปที่ 2.1 แสดงผังไดอะแกรมของการประมวลสัญญาณเชิงดิจิตอล

ในขั้นตอนของการแปลงสัญญาณอนาล็อกเป็นสัญญาณดิจิตอลจะสามารถขยายได้ดังรูปที่ 2.2 ซึ่งประกอบด้วยการสุ่มสัญญาณด้วยคาบเวลา T ค่าของสัญญาณแต่ละลำดับจะถูกเก็บไว้ในรูปของเลขไบนารีโดยการควอนไทซ์(quantized) ซึ่งการควอนไทซ์นั้นจะต้องมีการปัดเศษ(rounding) หรือไม่ก็ต้องมีการตัดเศษทิ้งไป(truncated) ผลของการกระทำดังกล่าวทำให้เกิดการแปลงที่ไม่เชิงเส้นขึ้น(nonlinear transformation) อันทำให้เกิดความผิดพลาด(error) ต่างๆขึ้นโดยจะกล่าวในภายหลัง



รูปที่ 2.2 แสดงกระบวนการแปลงสัญญาณอนาล็อกเป็นสัญญาณดิจิตอล

ในระบบของสัญญาณดิสครีตที่มีลักษณะของสัญญาณเป็นลำดับ การประมวลสัญญาณเชิงดิจิตอลจะใช้การคำนวณตามอัลกอริธึม(algorithm) แทนการแปลง(transform)สัญญาณอินพุทเพื่อให้ได้สัญญาณเอาท์พุทตามที่ต้องการ ซึ่งจะแสดงได้โดยสมการ 2.1

$$y(n) = \sum_{k=-\infty}^{\infty} h(k)x(n-k) \qquad -\infty < n < \infty \quad \ldots\ldots 2.1$$

เมื่อ y(n) คือ เอาท์พุทของระบบที่มีลักษณะเป็นลำดับ

x(n) คือ อินพุทของระบบที่เป็นลำดับมาจากการสุ่มสัญญาณและการควอนไทซ์

h(n)คือ ผลตอบสนองต่ออิมพัลส์ของระบบ

สมการที่ 2.1 จะเรียกอีกอย่างว่าผลรวมของการคูณประสาน(convolution sum equation) ในการประมวลสัญญาณดิจิตัลส่วนใหญ่การคำนวณจะต้องอาศัยสมการ 2.1 นี้อยู่มาก

จากรูปที่ 2.1 จะเห็นว่าการประมวลสัญญาณเชิงดิจิตอลจะต้องประกอบไปด้วยวงจรแปลงสัญญาณอนาล็อกเป็นสัญญาณดิจิตอลและวงจรแปลงสัญญาณดิจิตอลเป็นสัญญาณอนาล็อก ซึ่งส่วนใหญ่จะเป็นงานทางด้านดิจิตอลฟิลเตอร์(digital filter) แต่สำหรับโครงงานนี้จะทำเฉพาะส่วนของกระบวนการคำนวณเชิงตัวเลขเท่านั้น ดังนั้นถ้าต้องการใช้งานด้านดิจิตอลฟิลเตอร์ก็เพียงแต่ต่อวงจรแปลงสัญญาณอนาล็อกเป็นสัญญาณดิจิตอล และวงจรแปลงสัญญาณดิจิตอลเป็นอนาล็อกเพิ่มเท่านั้น

2.1 ทฤษฎีการคูณประสาน(DISCRETE-TIME CONVOLUTION)

การคูณประสานเป็นรูปแบบการประมวลผลที่จะให้ผลลัพธ์แสดงในโดเมนเวลา การศึกษาการคูณประสานจะเริ่มจาก สำดับของอิมพัลส์หนึ่งหน่วย(unit impulse) ซึ่งมีคุณสมบัติ

$$d(n) = \begin{cases} 1 & \text{เมื่อ } n = 0 \\ 0 & \text{นอกเหนือจากนี้} \end{cases} \quad \ldots\ldots 2.2$$

ลำดับของอิมพัลส์มีคุณสมบัติของการหน่วง กล่าวคือ

$$d(n-k) = \begin{cases} 1 & \text{เมื่อ } n = k \\ 0 & \text{นอกเหนือจากนี้} \end{cases} \quad \ldots\ldots 2.3$$

คุณสมบัติข้อนี้นับว่าจำเป็นมาก เนื่องจากองค์ประกอบแต่ละตัวของลำดับต่างๆสามารถแยกออกจากลำดับทั้งหมด x(n) ได้ดังนี้

$$x(n) = \sum_{k=-\infty}^{\infty} x(k)d(n-k) \quad \ldots\ldots 2.4$$

d(n-k) จะไม่เป็นศูนย์เมื่อ n=k เพราะฉนั้นเมื่อทำการคูณ d(n-k) กับ x(k) ทีละเทอมจะได้เทอม x(k) ตัวเดียวที่ไม่เป็นศูนย์และผลรวมทุกค่า k ทั้งหมดเป็น x(n) เช่นกัน

ในการประมวลสัญญาณเชิงดิจิตอลก็คือการเปลี่ยนค่าอินพุทที่เข้ามา x(n) ให้กลายเป็นเอาท์พุท y(n) โดยผ่านการแปลงที่จะใช้สัญญลักษณ์ G{ } ดังนั้นลำดับของค่าเอาท์พุท y(n) จึงเป็น

$$\begin{aligned} y(n) &= G\{ x(n) \} \\ &= G\{ x(k)d(n-k) \} \\ &= G\{ \ldots+x(-1)d(n+1)+x(0)d(n)+x(1)d(n-1)\ldots \} \\ &= \ldots+x(-1)G\{d(n+1)\}+x(0)G\{d(n)\}+x(1)G\{d(n-1)\}+\ldots \\ &= \sum_{k=-\infty}^{\infty} x(k)G\{d(n-k)\} \quad \ldots\ldots 2.5 \end{aligned}$$

G{d(n-k)} คือลำดับของเอาท์พุทที่มีอินพุทเป็นลำดับของอิมพัลส์ซึ่งมีค่าที่ไม่เป็น

ศูนย์ ณ n=k ค่าลำดับเอาท์พุทนี้เรียกว่า ผลตอบสนองอิมพัลส์(impulse response) แทนด้วยสัญญลักษณ์ h(n,k)

$$G\{d(n-k)\} = h(n,k) \qquad \text{.....2.6}$$

เพราะฉนั้นค่าเอาท์พุท y(n) สามารถแทนในรูปของผลตอบสนองอิมพัลส์ h(n,K) ได้ดังนี้

$$y(n) = \sum_{k=-\infty}^{\infty} x(k)h(n,k) \qquad \text{.....2.7}$$

รูปแบบนี้เรียกว่า <u>ความสัมพันธ์ของการคูณประสาน</u> และสำหรับระบบที่ไม่แปรผันตามเวลา(time-invariant system) จะได้

$$G\{d(n-k)\} = h(n,k)$$

$$= h(n-k) \qquad \text{.....2.8}$$

แทนค่า h(n,k) ในสมการ 2.7 ได้

$$y(n) = \sum_{k=-\infty}^{\infty} x(k)h(n-k) \qquad \text{....2.9}$$

สมการที่ 2.9 สามารถเขียนแทนด้วยสัญญลักษณ์

$$y(n) = x(n) * h(n)$$

และจากสมการที่ 2.9 แทน m = n-k

$$y(n) = \sum_{m=-\infty}^{\infty} x(n-m)h(m)$$

$$= \sum_{m=-\infty}^{\infty} h(m)x(n-m)$$

$$= h(n) * x(n)$$

นั่นคือ

$$x(n) * h(n) = h(n) * x(n) \qquad \text{..... 2.10}$$

เป็นคุณสมบัติของการสลับที่ของการคูณประสาน

2.2 <u>ทฤษฎีการสร้างสัญญาณคลื่นรูปซายน์(SINE-WAVE GENERATOR)</u>

การสร้างสัญญาแคลื่นรูปซายน์(sine-wave generators) เป็นพื้นฐานที่สำคัญของการประมวลสัญญาณ(signal processing) ซึ่งถูกประยุกต์ให้ใช้งานทางด้านต่างๆ ได้แก่งานทางด้านการสื่อสาร(communication),งานทางด้านการวัดคุม(instrument)

งานทางด้านการควบคุม(control) ในอดีตที่ผ่านมา วิศวกรด้านนี้มักออกแบบตัวกำเนิดความถี่(oscillators) ด้วยวงจรอนาล็อก(analog circuit) แต่ในโครงงานนี้จะใช้ดิจิตอลซิกแนลโปรเซสเซอร์ความเร็วสูง TMS32010 มาออกแบบ ซึ่งมีข้อดีกว่าในหลายๆ ด้าน TMS32010 มีความรวดเร็วและความแม่นยำสูง(accuracy) รวมถึงเสถียรภาพ สัญญาณคลื่นรูปซายน์ที่ได้จะมีความเพี้ยนต่ำและสามารถปรับช่วงของความถี่ได้กว้าง

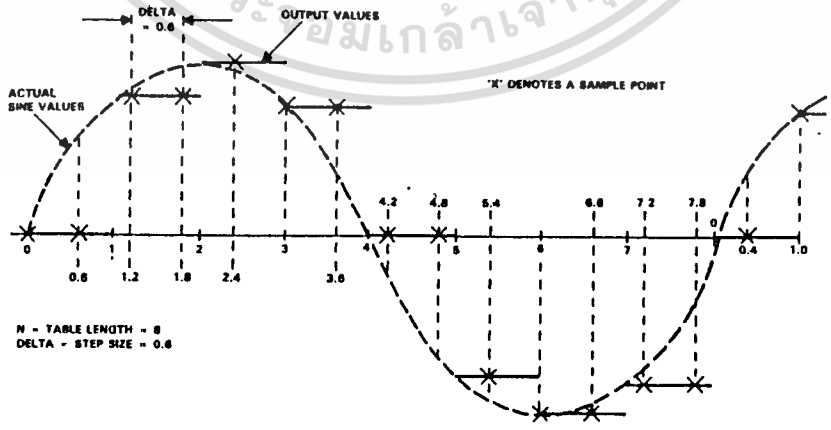
การสร้างสัญญานคลื่นรูปซายน์ทางดิจิตอล โดยใช้ดิจิตอลซิกแนลโปรเซสเซอร์ TMS32010 ทำได้ 2 วิธีด้วยกันคือ

วิธีที่ 1 เป็นวิธีดูค่าจากตารางอย่างรวดเร็ว(fast direct table look up) เหมาะสำหรับระบบที่ต้องการความเร็วในการประมวลผล แต่ไม่ต้องการด้านความเที่ยงตรงแม่นยำมาก

วิธีที่ 2 เป็นวิธีการประมาณค่า โดยอาศัยเทคนิคของการประมาณค่าเชิงเส้น (linear interpolation) เพื่อให้ได้สัญญาณคลื่นรูปซายน์ที่มีความเพี้ยนทางฮาร์โมนิกต่ำ(minimum of harmonic distortion)

2.2.1 วิธีการดูค่าจากตารางโดยตรง(DIRECT TABLE LOOK UP METHOD)

อัลกอริธึมของวิธีนี้จะมีดังนี้คือจะแบ่งค่ามุมของสัญญาณซายน์ออกเป็น N ค่าหรือเป็นการแบ่งวงกลมหนึ่งหน่วยออกเป็น N ส่วนเท่าๆกันดังแสดงในรูปที่ 2.3



รูปที่ 2.3 แสดงการประมาณค่าของสัญญาณซายน์โดยวิธีดูค่าจากตารางโดยตรง

ค่าของสัญญาณซายน์แต่ละค่าจะถูกเก็บไว้ในตาราง(table) โดยมีรูปแบบของการเก็บดังนี้

INDEX	ANGLE	SINE TABLE
0	0x360	S[0] = sin(0 /N)
1	1x360	S[1] = sin(360 /N)
2	2x360	S[2] = sin(720 /N)
.	.	.
.	.	.
N-2	(N-2)x360	S[N-2] = sin[(N-2)x360 /N]
N-1	(N-1)x360	S[N-1] = sin[(N-2)x360 /N]

คลื่นรูปซายน์จะถูกสร้างขึ้น โดยการสเต็ปค่าในตารางด้วยค่าคงที่(ซึ่งผลของมันคือเป็นการเคลื่อนวงกลมหนึ่งหน่วยไปทางทิศทวนเข็มนาฬิกา) และจะมีค่าวนค่าสุดท้ายในตารางเมื่อเกิน 360 องศา โดยการใช้ดัชนีตาราง(table index) เป็นพารามิเตอร์ของมุมและ DELTA เป็นขนาดของการสเต็ป การใช้วิธีดูค่านี้จะสร้างลำดับ

S[mod(K x DELTA , N)] สำหรับ K = 1,2,3,4,.....

โดยที่

mod(a,b) = เศษจากการหารของ a/b โดยที่ผลลัพธ์ของการหารต้องเป็นจำนวนเต็ม เช่น mod(22.34,5) = 2.34

สัญญาณของรูปคลื่นที่สร้างออกมา จะเป็นเพียงค่าประมาณของสัญญาณคลื่นรูปซายน์ โดยทั่วไปแล้ว ยิ่งตารางมีความละเอียดเท่าไร สัญญาณที่สร้างออกมาก็จะมีความถูกต้องมากขึ้นเท่านั้นและมีความใกล้เคียงกับสัญญาณซายน์มากขึ้น

ความถี่ของสัญญาณซายน์ (f) ขึ้นอยู่กับ

1 ช่วงเวลาของการแซมปลิ้ง(sampling interval) t

2 ขนาดของการสเต็ป DELTA

ความถี่สามารถหาได้จาก

f = DELTA / (t x N) เฮิร์ต

เมื่อ t มีหน่วยเป็นวินาที และ DELTA =< N / 2

เช่นในรูปที่ 2.3 ถ้า N = 8 และ DELTA = 0.6 คาบการสุ่มมีค่า 1 มิลลิวินาที ดังนั้น t =0.000125 วินาที

f = 0.6 / (8 x 0.000125)

= 600. เฮิร์ต

2.2.2 การใช้ TMS32010 ในการสร้างสัญญาณคลื่นรูปซายน์

ในส่วนนี้เป็นรูทีนของการสร้างสัญญาณโดยใช้วิธีการอ่านค่าในตารางโดยตรง โดยตารางค่าของซายน์จะมีอยู่ 128 ค่า(ค่าเหล่านี้มาจากการสุ่มสัญญาณ ยิ่งมีความละเอียดมากจำนวนค่าก็จะมีเพิ่มขึ้น) เมื่อใดก็ตามที่รูทีนนี้ถูกเรียกใช้ ค่าถัดไปจะถูกคำนวณทันที

ค่าของซายน์ในตารางจะถูกเปลี่ยนสเกลทั้งหมด และค่าในเลขฐานสิบเช่นค่าของ +1.0 และ -1.0 จะแทนในรูปของเลขทูคอมพลีเมนท์ฐานสิบหก(two's complement hexadecimal) คือ 4000 และ C000 ตามลำดับ สำหรับค่าอื่นๆก็จะถูกเปลี่ยนให้ใกล้เคียงกับเลขฐานสิบหกมากที่สุดโดยการปัดเศษหรือการตัดเศษ แต่ส่วนมากแล้วจะใช้วิธีการปัดเศษมากกว่าการตัดเศษ เพื่อเป็นการหลีกเลี่ยงไม่ให้มีการเพิ่มความเพี้ยนมากขึ้นโดยไม่จำเป็น

ตำแหน่งของหน่วยความจำข้อมูล ALPHA จะใช้การ'mod' เพื่อที่จะสามารถทำการเลือกค่าของซายน์ในตารางได้ ค่าของ ALPHA เป็นจำนวนเต็มและเศษส่วนซึ่งมีรูปแบบดังนี้

Q	Q	Q	Q	Q	Q	Q	Q	.	Q	Q	Q	Q	Q	Q	Q	Q
15	14	13	12	11	10	9	8		7	6	5	4	3	2	1	0

ตำแหน่งของหน่วยความจำข้อมูล DELTA จะเก็บค่าขนาดสเต็ปไว้โดยมีรูปแบบเช่นเดียวกับ ALPHA ทุกๆครั้งที่มีการเรียกใช้รูทีนนี้ค่าที่เก็บไว้ใน ALPHA จะเพิ่มค่าเท่ากับค่าที่เก็บไว้ใน DELTA ส่วนที่เป็นจำนวนเต็มของ ALPHA จะเป็นตัวชี้ในตารางของค่าซายน์ แต่อย่างไรก็ตาม เพราะตารางเริ่มที่ตำแหน่งแอดเดรส SINE ดังนั้นตัวชี้ตัวนี้จะเป็นค่า

ออฟเซ็ทของแอดเดรสก่อนที่จะมีการแอ็กเซส(access)ตารางนี้ ค่า 8 บิทนัยสำคัญสูงของ ALPHA จะถูกมาสก์(mask) เมื่อ ALPHA ถูกอัพเดท(upldate)เพื่อให้แน่ใจว่าค่าจะไม่เกิน 127 รูทีนนี้จะให้ค่าของซายน์ในตำแหน่งของหน่วยความจำข้อมูลที่ SINA

เมื่อมีการกำหนดช่วงของการแซมปลิ้ง t ความถี่ของสัญญาณที่ถูกสร้างขึ้นมาจะหาได้จาก

f = DELTA / t x 128

เนื่องจาก DELTA มีตำแหน่งทศนิยม(precision)ที่มีความละเอียดมากขนาด 8 บิท หรือ 2^8 = 256 ดังนั้นเลขทศนิยมจะมีค่าเป็น 1/256 ค่าความถี่ที่ต้องการมีค่าประมาณที่ผิดพลาดไม่มากกว่า (1/256) / (t x 128) หรือ 1 / (32768 x t)เฮิร์ต

ตัวอย่างเช่น ถ้าความถี่ของการแซมปลิ้งเป็น 8 กิโลเฮิร์ตแล้วความละเอียดของความถี่จะเป็น

8000/32768 = 0.25 เฮิร์ต

2.2.3 <u>การประมาณค่าเชิงเส้น(LINEAR INTERPLATION METHOD)</u>

เพื่อที่จะลดความเพี้ยนทางฮาร์โมนิกสำหรับในแต่ละค่าของตารางค่าซายน์ การอินเทอร์โพเลทจะสามารถทำการคำนวณค่าซายน์ ระหว่างจุดสองจุดได้เที่ยงตรงกว่าแบบแรก วิธีนี้ใช้ค่าในสองจุดเรียงกันในตารางให้เป็นจุดปลายของเส้นตรง จุดตัวอย่างของค่าพารามิเตอร์จะตกอยู่ระหว่างค่าในตารางโดยสมมติว่าให้เป็นเส้นตรงระหว่างจุดสองจุด อัลกอริธึมของวิธีนี้ได้แสดงดังรูปที่ 2.4

อัลกอริธึมนี้มีพื้นฐานมาจากการประมาณค่าเชิงเส้นดังนี้

sin (360 (I+D)/N) = sin (360 I/N) + D * [sin(360 (I+1)/N)
-sin(360 I/N)]

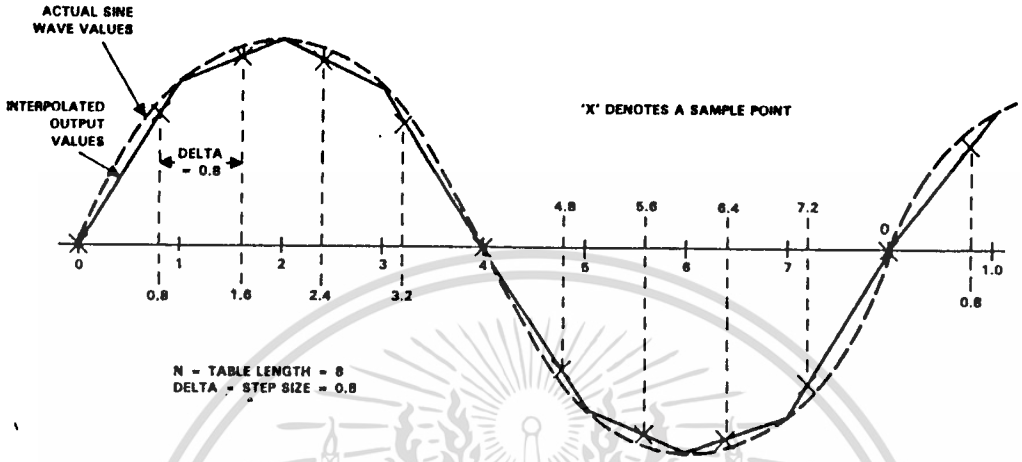
= sin[I] + D*{S[I+1]-S[I]} 2.11

เมื่อ N คือจำนวนของค่าซายน์ในตาราง

I คือจำนวนเต็มโดยที่ 0 =< I =< N-1

D คือจำนวนทศนิยม 0 =< D =< 1.0

ค่าของ S[I+1]-S[I] จะเป็นความชัน(slope) ของเส้นตรงระหว่างจุดตัวอย่างสองจุดซึ่งจะอยู่ภายในวงเล็บ I+D



รูปที่ 2.4 แสดงการประมาณค่าสัญญาณซายน์โดยวิธีประมาณค่าเชิงเส้น

ค่าต่างๆในตารางที่ต้องการใช้ในการอินเทอร์โพเลทจะเก็บไว้ดังนี้

INDEX	ANGLE	SINE TABLE	SLOPE TABLE
0	0x360 /N	S[0]=sin(0 /N)	S[1]-S[0]
1	1x360 /N	S[1]=sin(360 /N)	S[2]-s[1]
2	2x360 /N	S[2]=sin(720 /N)	S[3]-S[2]
.	.	.	.
.	.	.	.
.	.	.	.
N-2	(N-2)x360 /N	S[N-2]=sin(360 (N-2)/N)	S[N-1]-S[N-2]
N-1	(N-1)x360 /N	S[N-1]=sin(360 (N-1)/N)	S[0]-S[N-1]

2.3 ผลของการที่มีทศนิยมจำกัด(FINITE-PRECISION EFFECTS)

เป็นผลมาจากการแทนค่าตัวเลขที่มีรูปแบบที่จำกัดขนาดความละเอียด ซึ่งโดยทั่วไป

จะเก็บไว้ในรูปของเลขไบนารี(binary) โดยเก็บไว้เป็นจำนวนบิท ยิ่งจำนวนบิทมากก็จะทำให้ความละเอียดมาก แต่ก็จะสูญเสียเวลาคำนวณไปมากเช่นกัน ด้วยเหตุนี้ค่าความผิดพลาดที่เกิดขึ้นจากการแทนด้วยเลขจำนวนบิทที่จำกัดนี้จึงหลีกเลี่ยงมิได้ เป็นผลจากการแทนของลำดับอินพุท(input sequence), สัมประสิทธิ์ของสมการ, ค่าตัวกลางระหว่างการประมวลของการบวกหรือการคูณ

2.4 การแทนตัวเลขในคอมพิวเตอร์

โครงสร้างฮาร์ดแวร์ทางดิจิตอล(digital hardware structure) ต่างๆสามารถจะแทนตัวเลขในรูปแบบของผลรวมของจำนวนของไบนารีดิจิต(binary digits) หรือบิทซึ่งมีค่าเป็น 0 หรือ 1 เท่านั้น บิทเหล่านี้สามารถรวมตัวกันได้เป็นรูปแบบต่างๆเช่น

รวมกัน 8 บิท เรียกว่าไบต์(byte)

รวมกัน 16 บิท เรียกว่าเวิร์ด(word)

รวมกัน 32 บิท เรียกว่าดับเบิลเวิร์ด(double word)

รูปแบบที่นิยมใช้สำหรับการแทนค่าตัวเลขของคอมพิวเตอร์นี้มี 2 อย่างคือ

1 ฟิกซ์พอยต์(fixed point)

2 โฟลทติ้งพอยต์(floating point)

2.4.1 ฟิกซ์พอยต์

จะแทนตัวเลขในรูปไบนารีซึ่งแบ่งจำนวนบิททั้งหมดเป็น 3 ส่วนดังนี้

- บิทเครื่องหมาย(Signed bit)
- บิทจำนวนเต็ม(Integer bit)
- บิททศนิยม(Fraction bit)

บิทเครื่องหมายใช้แทนเครื่องหมายของตัวเลขโดยถ้าบิทนี้มีค่าเป็น 0 ตัวเลขเป็นบวก หรือถ้าเป็น 1 ตัวเลขจะมีค่าเป็นลบ

บิทจำนวนเต็มจะเก็บค่าขนาดของเลขจำนวนเต็ม ซึ่งมีค่าเป็นผลรวมของกำลังของเลขฐานสองดังนี้

$$n = b_0 + b_1 2^1 + b_2 2^2 + b_3 2^3 + \ldots\ldots\ldots + b_k 2^k$$

โดยที่ b_i มีค่าเป็น 0 หรือ 1 ; $i = 0,1,2,\ldots,k$

บิททศนิยมจะเก็บค่าขนาดของเลขเศษส่วนไว้ในรูปของผลรวมของกำลังของเลขสองดังนี้คือ

$$m = b_1 2^{-1} + b_2 2^{-2} + b_3 2^{-3} + b_4 2^{-4} + \ldots\ldots\ldots + b_{-k} 2^{-k}$$

b_i คือค่าบิทของบิททศนิยมที่เรียงค่าจากท่างซ้ายไปยังขวามือ

สรุปแล้วค่าเลขฟิกซ์พอยต์นี้จะเป็น

S n.m

โดย S คือ บิทเครื่องหมาย

<u>ระบบเลขทูคอมพลีเมนท์(2's COMPLEMENT NOTATION)</u>

ระบบการแทนเลขฟิกซ์พอยต์ที่นิยมกันมากในปัจจุบันได้แก่ระบบเลขทูคอมพลีเมนท์อันเป็นผลมาจากประสิทธิภาพในการคำนวณของระบบ รูปแบบของบิทของเลขลบจะเริ่มด้วยบิทแรกสุดเป็น 1

ตัวอย่างการแทนด้วยเลขทูคอมพลีเมนท์ จะทำการแปลงจากเลขฐานสิบค่า -6.86_{10}

อันดับแรกแปลงขนาดของตัวเลขนี้ 6.86 ให้เป็นระบบไบนารีโดยใช้วิธีราวอ็อฟ (round-off) ดังนี้

<u>rounf-off</u>	<u>เศษ</u>
6/2 = 3	0
3/2 = 1	1
1/2 = 0	1

เพราะฉนั้น $6_{10} = 0110_2$

ส่วนทศนิยมคือ 0.86 จะใช้วิธีการปัดเศษหรือการตัดเศษก็ได้ โดยมีวิธีดังนี้

	<u>ส่วนจำนวนเต็ม</u>
0.86 x 2 = 1.72	1

0.72 x 2 = 1.44 1

0.44 x 2 = 0.88 0

0.88 x 2 = 1.76 1

0.76 x 2 = 1.52 1 (บิทนัยสำคัญต่ำสุด)

เพราะฉนั้น

$0.86_{10} = 0.1101_2$ ($= 0.8125_{10}$) โดยการตัดเศษ

$= 0.1110_2$ ($= 0.875_{10}$) โดยการปัดเศษ

นำมารวมกันกับส่วนจำนวนเต็ม

$6.86_{10} = 0110.1101_2$ โดยการตัดเศษ

$= 0110.1110_2$ โดยการปัดเศษ

เราเลือกใช้ตัวเลขจากการปัดเศษจะได้

$6.86_{10} = 0110.1110_2$

ทำการคอมพลีเมนท์(complement) ผลลัพธ์นี้ที่ละบิทจะได้

$6.861_{10} = 1001.0001_2$

บวกด้วย 0.0001_2 เข้ากับตัวเลขคอมพลีเมนท์นี้

จะได้

$-6.86_{10} = 1001.0010_2$ ($= -6.875_{10}$)

ตารางข้างล่างนี้จะแสดงการแทนระบบเลขทูคอมพลีเมนท์กับเลขฟิกซ์พอยต์ที่มีจำนวนบิทเป็น 2,3,4

binary number	integer value (binary point at right)	fractions (binary point after sign bit)
2 bit number system		
01	1	0.5
00	0	-0
11	-1	-0.5
10	-2	-1

3 bit number system

011	3	0.75
010	2	0.5
001	1	0.25
000	0	0
111	-1	-0.25
110	-2	-0.5
101	-3	-0.75
100	-4	-1

4 bit number system

0111	7	0.875
0110	6	0.75
0101	5	0.625
0100	4	0.5
0011	3	0.375
0010	2	0.25
0001	1	0
0000	0	0
1111	-1	-0.125
1110	-2	-0.25
1101	-3	-0.375
1100	-4	-0.5
1011	-5	-0.625
1010	-6	-0.75
1001	-7	-0.875
1000	-8	-1

2.4.2 เลขโฟลทพอยต์(FLOAT-POINT NOTATION)

หรือเรียกอีกอย่างหนึ่งว่า รูปแบบของจำนวนจริง ในรูปแบบนี้จำนวน X จะถูกแสดงดังนี้

$$X = m\,2^{e}$$

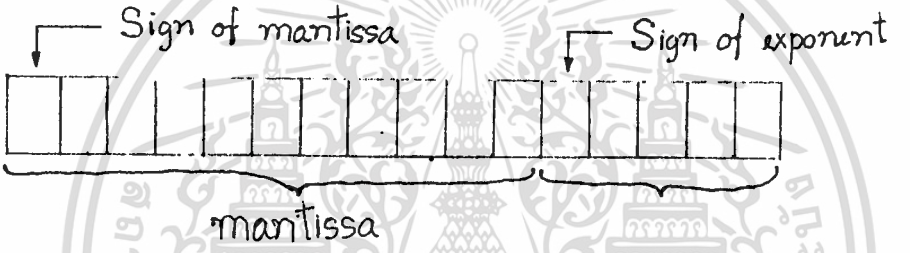
เมื่อ m คือ ค่าของแมนทิสซา(mantissa)

e คือ ค่าของชี้กำลัง (exponent)

ค่าของ m มักเขียนอยู่ในช่วง $1/2 =< m < 1$

ยกตัวอย่างเช่น ค่าของ 1.5 จะมีค่าเท่ากับ 0.75×2^{1}

รูปแบบบิทของเลขโฟลทพอยต์ < กรณี 16 >



2.5 สาเหตุของความผิดพลาดที่เกิดจากการแทนเลขด้วยเลขทศนิยมรู้จบ

เป็นความผิดพลาดของการแทนค่าของเลขทศนิยมไม่รู้จบ(infinite-precision value) ด้วยระบบเลขที่มีทศนิยมจำกัด (finite-precision system)

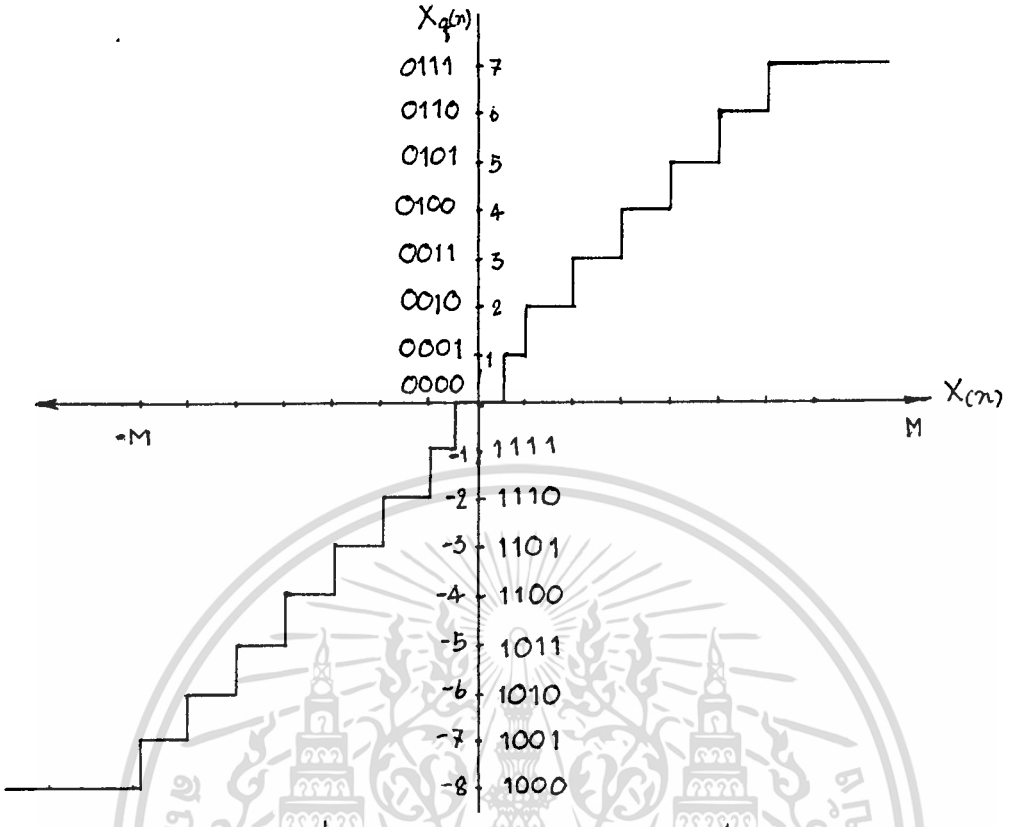
1 ความผิดพลาดจากการควอนไทซ์ค่าของอินพุท

2 ความผิดพลาดจากการควอนไทซ์สัมประสิทธิ์

3 ความผิดพลาดจากการควอนไทซ์ผลลัพธ์

5.1 ความผิดพลาดจากการควอนไทซ์ค่าของอินพุท

เกิดจากการแทนค่าที่ได้จากการสุ่ม (sampling) ของวงจรอนาล็อกทูดิจิตอลคอนเวอร์เตอร์(anolog-to-digital converter ADC) ด้วยจำนวนบิทที่จำกัด ปกติจะใช้เลขฟิกซ์พอยต์แทนด้วย x(n) หลังจากนั้นก็จะแปลงค่า x(n) นี้มาเป็นค่าที่ถูกควอนไทซ์ จากค่า x(n) มาเป็น x_{ϵ} (n) กรรมวิธีการควอนไทซ์จากค่า x(n) มาเป็น x_{ϵ} (n) สามารถแสดงดังรูปที่ 2.5



รูปที่ 2.5 แสดงกรรมวิธีการควอนไทซ์

ช่วงของการควอนไทซ์(quantizer range) นี้จะแสดงในรูปตัวเลขทูคอมพลีเมนท์ซึ่งมีช่วง -M ถึง M-1 จำนวน k สเต็ป แต่ละสเต็ปมีขนาด $= 2M / k$

ถ้าหา ควอนไทเซอร์ประกอบด้วย b บิท จะได้ $k = 2^b$

และ $\triangle = 2M / 2^b$

หากเรากำหนดช่วงของควอนไทซ์ไว้คงที่แล้ว การเพิ่มค่าของจำนวนบิทของตัวควอนไทเซอร์จะทำให้ค่า $\triangle$ ลดลงนั่นคือจะทำให้ $x(n)$ กับ $x_q(n)$ มีความสัมพันธ์เป็นเชิงเส้นกัน

ค่าผิดพลาดที่เกิดจากการควอนไทเซอร์นี้ เกิดจาก 2 กรณี คือ

กรณี $x(n) \leq M$ ให้ความผิดพลาดเป็น $E_g(n)$ เรียกว่า ganular error

$$E_g(n) = x(n) - x_q(n)$$

จะสังเกตได้ว่าขนาดของความผิดพลาดจากกรณีนี้มีค่าน้อยกว่าขนาดของสเต็ป (step size)

กรณี $x(n) > M$ เรียกความผิดพลาดนี้ว่าคลิปปิ้งเออร์เรอร์(clipping

error) แทนด้วย $E_c(n)$ โดยมีค่าเป็น

$$E_c(n) = \begin{cases} x(n) - M - 1 & \text{สำหรับ } x(n) > M-1 \\ x(n) + M & \text{สำหรับ } x(n) < -M \end{cases}$$

ค่าคลิปปิ้งเออร์เรอร์สามารถทำให้มีผลน้อยลง โดยการเพิ่มค่าช่วงของการควอนไทซ์หรือโดยการปรับสเกลของการสุ่มสัญญาณอินพุท x(n) ให้มีค่าน้อยๆพอประมาณซึ่งถ้าทำได้ดังนี้ ก็จะเป็นผลให้ค่าความผิดพลาดของการควอนไทซ์รวม เป็นผลมาจาก granular error มากกว่า

2.5.2 <u>ค่าความผิดพลาดที่เกิดจากการควอนไทซ์สัมประสิทธิ์</u>

เกิดจากการแทนค่าตัวเลขทศนิยมไม่รู้จบ ลงในระบบตัวเลขที่คอมพิวเตอร์สามารถประมวลผลได้ ซึ่งได้แก่เลขฟิกซ์พอยต์หรือโฟลทพอยต์ทำให้ตัวเลขสัมประสิทธิ์นี้ถูกตัดหรือปัดดังในตัวอย่างก่อนหน้านี้ ซึ่งก็ทำให้เกิดความผิดพลาดในการแทนตัวเลขสัมประสิทธิ์นี้ขึ้นได้

2.5.3 <u>ความผิดพลาดจากการควอนไทซ์ผลลัพธ์</u>

เกิดจากการคูณหรือการบวกของขั้นตอนการประมวลผล ทำให้ได้ค่าจำนวนบิทที่มากขึ้น ยกตัวอย่างการคูณกันกับฟิกซ์พอยต์ 2 ค่า ที่แต่ละค่ามีจำนวน 4 บิทผลลัพธ์ที่ได้จะเป็น 8 บิท เราต้องตัด 4 บิททิ้งเพื่อให้คงสภาพของระบบตัวเลขเดิมไว้ ทำให้เกิดความผิดพลาดขึ้น ซึ่งหากเป็นฟิลเตอร์ชนิดนอนรีเคอร์ซีฟ(nonrecursive filter)โครงสร้างจะเหมือนกับการบวกนอยส์(noise)ให้กับเอาท์พุทของแต่ละตัวคูณ แต่ถ้าใช้กับโครงสร้างแบบรีเคอร์ซีฟ(recursive structure) การตัดทิ้งนี้จะทำให้เอาท์พุทเกิดออฟเซ็ท (offset)ที่ไม่เป็นที่ต้องการหรือเกิดการออสซิเลท(oscillate)ได้ เราเรียกว่าลิมิตไซเคิล(limit cycle)

2.6 <u>การประมวลผลทางคณิตศาสตร์กับระบบเลขทศนิยมรู้จบ</u>

การประมวลผลของเลขทศนิยมรู้จบ นี้ประกอบด้วยการคูณการบวกซึ่งจำเป็นสำหรับ

งานแอพลิเคชั่นต่างๆเช่น ดิจิตอลฟิลเตอร์ จะแยกการประมวลผลออกเป็นของฟิกซ์พอยต์และโฟลทพอยต์

2.6.1 คณิตศาสตร์ของเลขฟิกซ์พอยต์

เมื่อมีการแทนค่าตัวแปรที่รับเข้ามาด้วยเลขฟิกซ์พอยต์นี้จะต้องทำการสเกลเพื่อให้อยู่ในขอบเขตที่ต้องการหรือขนาดที่รูปแบบของฟิกซ์พอยต์นั้นจะรับไว้ได้เป็นการป้องกันไม่ให้เกิดการโอเวอร์โฟลว์ขึ้น

การประมวลผลจะแยกพิจารณาออกเป็น การบวก การคูณดังนี้

การบวกเลขฟิกซ์พอยต์

การบวกเลขฟิกซ์พอยต์สองตัวเลขเข้าด้วยกันทำได้ไม่ยาก โดยการบวกกันทีละบิทจากบิทขวามาจนถึงบิททางซ้ายโดยแคร์รี่บิท(carry bit) จะถูกบวกเข้ากับบิทถัดไปทางซ้ายมือด้วย การลบก็ทำได้โดยการทำนิเสธตัวเลขที่จะลบ แล้วนำมาบวกกัน

ตัวอย่างการบวก

1.5_{10} + 1_{10} แทนด้วยเลขไบนารี

001.1

001.0

010.1 = 2.5_{10}

2.5_{10} + (-2_{10})

010.1

110.0

000.1 = 0.5_{10}

จะสังเกตว่าแคร์รี่บิทตัวสุดท้ายจะไม่นำมาคิด

-2.5_{10} + 2_{10}

101.1

010.0

111.1 = -0.5_{10}

3_{10} + 3.5_{10}

011.0

011.1

110.1 = -1.5_{10}

ตัวอย่างหลังสุดนี้จะเกิดการโอเวอร์โฟลว์ขึ้น ทั้งนี้เพราะฟิกซ์พอยต์ขนาด 4 บิทดังในกรณีนี้สามารถแทนได้สูงสุดเพียง 3.5

ปรากฏการณ์เกิดโอเวอร์โฟลว์ขึ้นนี้มีสาเหตุมาจากการบวกเลขขนาด n บิทสองค่าเข้าด้วยกัน ได้ผลลัพธ์ที่ไม่สามารถแทนด้วย b บิทได้ ดังนั้นในการใช้งานจริงจึงต้องมีการป้องกันการเกิดโอเวอร์โฟลว์นี้ขึ้น จึงได้มีการสเกลค่าของตัวแปรที่จะทำการบวกนี้ให้อยู่ภายในขอบเขตที่เลขฟิกซ์พอยต์นี้สามารถจะรับไว้ได้

<u>การคูณกันของเลขฟิกซ์พอยต์</u>

การคูณกันของเลขฟิกซ์พอยต์นี้จะได้ผลลัพธ์ที่มีจำนวนบิทเป็นสองเท่าของตัวตั้งและตัวคูณเช่นคูณเลขฟิกซ์พอยต์ 2 ค่าซึ่งมีจำนวน n บิทในแต่ละค่าผลลัพธ์จะออกมาเป็น 2n บิท สำหรับเลขฟิกซ์พอยต์ แล้วสามารถแยกขั้นตอนการคูณออกเป็น 3 ขั้นตอนคือ

- แยกส่วนของบิทเครื่องหมายออกมาจากส่วนของบิทขนาด
- คูณบิทขนาดเข้าด้วยกัน
- ขนาดเครื่องหมายจะถูกหามา แล้วทำการบวกเข้ากับส่วนที่สองของบิทขนาด

สำหรับในระบบของเลขทศนิยมแล้ว เรามีการแบ่งเป็นส่วนของจำนวนเต็มและทศนิยม โดยบิท b_1 ของจำนวนเต็มและบิท b_f ของเลขทศนิยม ดังนั้นผลรวมของขนาดบิทเป็น

$$b = b_1 + b_f$$

หลังการคูณเลข 2 ชุดนี้เข้าด้วยกันก็จะได้บิทขนาดเป็น $2b_1 + 2b_f$ แต่ผลลัพธ์จะต้องทำการตัดหรือปัดให้มีขนาดเท่ากับบิท b_1 เท่านั้น เพื่อให้เข้ากับระบบตัวเลขเดิม

ตัวอย่างการคูณเลขฟิกซ์พอยต์ที่ขนาด 4 บิทและในรูปแบบเป็น xxx.x ดังนั้นกรณีนี้ $b = 4$, $b_f = 1$, $b_1 = 3$

2.5_{10} x 1.5_{10} ===> 010.1 x 001.1

0101

x

0011

0101

0101

0000

0000

0001111 = 00011.11 (เมื่อแทนจุดทศนิยม)

ทำให้เหลือ 4 บิท ดังระบบตัวเลขเดิม

0011.1 ($= 3.5_{10}$) โดยการตัดทิ้ง

0100.0 ($= 4.0_{10}$) โดยการปัดเศษ

การปัดหรือตัดเศษทิ้งนี้สามารถก่อให้เกิดเออเรอร์ชนิดที่เรียกว่าโปรดักควอนไทเซชั่นเออเรอร์(Product Quantization Error) ดังที่ได้กล่าวมาแล้วข้างต้น

<u>คณิตศาสตร์ของเลขโฟลทติ้งพอยต์</u>

ตัวเลขโฟลทติ้งพอยต์นี้จะสามารถช่วยลดปัญหาการเกิดโอเวอร์โฟลว์ได้มาก จากการที่สามารถรับค่าขนาดตัวเลขไว้ได้มาก การแทนตัวเลขในระบบนี้สามารถลดขั้นตอนการสเกลค่าอินพุทและตัดความกังวลเรื่องโอเวอร์โฟลว์เออเรอร์ขึ้นได้มาก

การบวกเลขโฟลทติ้งพอยต์ จะต้องทำให้ส่วนของเอ็กโปเนนเชี่ยลเท่ากันก่อนจะทำการบวก โดยจะทำให้เลขโฟลทติ้งพอยต์ที่มีส่วนเอ็กโปเนนเชี่ยนน้อยกว่ามีค่าเลขยกกำลังเท่ากัน ทั้งนี้เพื่อป้องกันการเกิดโอเวอร์โฟลว์ขึ้นกับส่วนของแมนทิสซา

การคูณกันของตัวเลขโฟลทติ้งพอยต์ทำได้โดยการคูณค่าของแมนทิสซาเข้าด้วยกัน และบวกค่าของเลขเอ็กโปเนนเชี่ยลเข้าด้วยกัน ต่อจากนั้นก็ทำการปรับค่าของเอ็กโปเนนเชี่ยลใหม่เพื่อให้ได้ส่วนของขนาดของผลคูณแมนทิสซาตกอยู่ในช่วง [1/2 , 1)

หมายเหตุ ในส่วนของการประมวลผลเลขโฟลทติ้งพอยต์จะไม่ได้เน้นมากในโครงงานนี้ เพราะระบบตัวเลขที่ใช้ในโครงงานนี้เป็นระบบตัวเลขฟิกซ์พอยต์ เสียเป็นส่วนใหญ่

บทที่ 3

การออกแบบระบบประมวลผลด้วยความเร็วสูง

การออกอบบระบบประมวลผลด้วยความเร็วสูงได้แยกพัฒนาออกเ เป็นสองส่วนดังนี้

1 การออกแบบและพัฒนาทางด้านฮาร์ดแวร์

2 การออกแบบและพัฒนาทางด้านซอฟท์แวร์

ในการออกแบบระบบจะเริ่มด้วยการออกแบบพัฒนาทางด้านฮาร์ดแวร์ก่อนแล้วจึงตามด้วยการพัฒนา ทางด้านซอฟท์แวร์ รายละเอียดในแต่ละส่วนจะกล่าวโดยละเอียดต่อไปดังนี้

3.1 การออกแบบและพัฒนาทางด้านฮาร์ดแวร์

การจัดสร้างฮาร์ดแวร์จะจัดสร้างให้อยู่บนการ์ด ที่สามารถเสียบใช้งานบนเครื่องไอบีเอ็มพีซีไมโครคอมพิวเตอร์ได้ ลักษณะของวงจรบนการ์ดนี้จะประกอบด้วยไมโครโปรเซสเซอร์(microprocessor) 2 ตัว คือ 8088 ไมโครโปรเซสเซอร์ของอินเทลซึ่งจะขอเรียกชื่อว่า 8088-slave และอีกตัวหนึ่งคือ TMS32010 โครงสร้างของระบบฮาร์ดแวร์ได้แสดงไว้ตามผังไดอะแกรมที่รวมไว้ในภาคผนวกของปริญญานิพนธ์ฉบับนี้

โครงสร้างของระบบฮาร์ดแวร์

จากผังไดอะแกรมรูปที่ 1 แสดงให้เห็นส่วนต่างๆของวงจรซึ่งประกอบด้วย

1 วงจรที่ทำหน้าที่ควบคุมการทำงานของ 8088-slave และ TMS32010 (8088-slave and TMS32010 control logic) วงจรส่วนนี้มีหน้าที่คอยรับคำสั่งการควบคุมการทำงานของไมโครโปรเซสเซอร์ทั้งสองตัวจากไอบีเอ็มพีซีไมโครคอมพิวเตอร์ว่าเมื่อใดจะให้อยู่ในสถานะรีเซ็ท(reset) เมื่อใดจะให้ทำงาน

2 หน่วยความจำ จากผังไดอะแกรมฮาร์ดแวร์ของระบบจะมีหน่วยความจำอยู่ 2 ส่วนคือ

- หน่วยความจำที่เป็นของ 8088-slave จะแบ่งเป็น 64 กิโลไบท์(kbyte)

สำหรับข้อมูลที่ต้องการประมวลผล และ 32 กิโลไบท์สำหรับเป็นโปรแกรมมอนิเตอร์ (monitor program) ของ 8088-slave

- หน่วยความจำที่เป็นของ TMS32010 จะมีขนาด 4 กิโลเวิร์ด(kword) ทำหน้าที่เป็นหน่วยความจำโปรแกรม(program memory) ให้กับ TMS32010

3 วงจรที่ทำหน้าที่สำหรับให้ 8088-slave ควบคุมการทำงานของ TMS32010 ตัว 8088-slave จะมีหน้าที่ติดตามควบคุมการทำงานของ TMS32010 แทนไอบีเอ็มพีซี เพื่อให้ระบบนี้เป็นระบบอาร์เรย์โปรเซสเซอร์(array processor)

สำหรับผังไดอะแกรมรูปที่ 2 และรูปที่ 3 จะแสดงการจัดสัญญาณควบคุมต่างๆ ในส่วนของวงจรอินเตอร์เฟส(interface) ระหว่างการ์ดนี้กับไอบีเอ็มพีซี

<u>การออกแบบระบบฮาร์ดแวร์</u>

การออกแบบในส่วนของฮาร์ดแวร์ ในโครงงานนี้แบ่งวงจรฮาร์ดแวร์บนการ์ดได้เป็น 4 ส่วนดังนี้

- ส่วนของการติดต่อกับไอบีเอ็มพีซี
- ส่วนของการติดต่อกับ 8088-slave
- ส่วนของการติดต่อกับ TMS32010
- ส่วนของหน่วยความจำ

3.1.1 <u>ส่วนของการติดต่อกับไอบีเอ็มพีซี</u>

ในวงจรส่วนนี้จะทำหน้าที่ในการติดต่อระหว่างไอบีเอ็มพีซีกับการ์ดนี้ การติดต่อจะสามารถแยกออกเป็นการติดต่อระหว่างพอร์ทและการติดต่อกับหน่วยความจำ รายละเอียดมีดังต่อไปนี้

3.1.1.1 การติดต่อกับพอร์ทบนการ์ด

สำหรับการ์ดนี้ได้ดีโค้ดพอร์ทอินพุท/เอาท์พุทไว้ที่แอดเดรส 3EEH พอร์ทอินพุททำหน้าที่ตรวจสอบสัญญาณควบคุมการทำงานของ 8088-slave, TMS32010 สัญญาณเลือกแบงค์(bank) ของหน่วยความจำบนการ์ด พอร์ทเอาท์พุททำหน้าที่ควบคุมการทำงานของ

8088-slave และ TMS32010 จากไอบีเอ็มพีซี รายละเอียดของส่วนนี้เริ่มจาก 74LS244-1, 74LS244-4 ทำหน้าที่เป็นบัฟเฟอร์ให้กับสัญญาณควบคุมต่างๆจากไอบีเอ็มพีซี อันได้แก่ $\overline{\text{PC-IOR}}$, $\overline{\text{PC-IOW}}$, $\overline{\text{PC-MEMR}}$, $\overline{\text{PC-MEMW}}$, PC-AEN, PC-RESET, และแอดเดรสบัส PCA0-PCA9 โดยมี 74LS30, 74LS85-2, 74LS155-1 และการจัดสัญญาณด้วยเกทต่างๆเป็นตัวดีโค้ดเดอร์

สัญญาณควบคุมที่ได้จากเอาท์พุทพอร์ทนี้มีดังนี้

BANK1 เป็นสัญญาณเลือกแบงค์ของหน่วยความจำบนการ์ดที่ไอบีเอ็มพีซีจะเลือกแอ็กเซสเอาท์พุทพอร์ทจะแลทช์ค่านี้ไว้โดย 74LS175

WAIT/$\overline{\text{RST}}$ เป็นสัญญาณควบคุมสภาวะรีเซ็ทของ 8088-slave สัญญาณนี้เอาท์พุทพอร์ทจะแลทช์ค่าไว้โดย 74LS175 เช่นกัน

$\overline{\text{GO88}}$ เป็นสัญญาณควบคุมการทำงานของ 8088-slave ปกติสัญญาณนี้จะเป็น 0 ซึ่งจะทำให้ 8088-slave ถูกรีเซ็ทตลอดเวลาจนเมื่อไอบีเอ็มพีซี ทำการโอนข้อมูลที่ต้องการให้ TMS32010 ประมวลผลเรียบร้อยแล้ว จึงสั่งเอาท์ให้สัญญาณนี้เป็น 1 และไม่มีการแลทช์ไว้

$\overline{\text{GODSP1}}$ เป็นสัญญาณควบคุมการทำงานของ TMS32010 มีลักษณะการทำงานเช่นเดียวกับสัญญาณ $\overline{\text{GO88}}$

3.1.1.2 การติดต่อกับหน่วยความจำบนการ์ด

ใช้แอดเดรสบัสจากไอบีเอ็มพีซี PCA16 - PCA19 ทำการดีโค้ดโดยใช้ 74LS85-1 เป็นตัวดีโค้ดเดอร์เพื่อให้ไอบีเอ็มพีซี สามารถติดต่อกับหน่วยความจำบนการ์ดที่แอดเดรส A0000H-AFFFFH ซึ่งมีขนาด 64 กิโลไบท์ผ่านแอดเดรสบัฟเฟอร์ 74LS244-2, 74LS244-3 และดาต้าบัฟเฟอร์ 74LS245

3.1.2 <u>ส่วนของการติดต่อกับ 8088-slave</u>

เพื่อให้การ์ดนี้สามารถทำงานในลักษณะพาราเลล(parallel) กับไอบีเอ็มพีซีได้อย่างสมบูรณ์ จึงจำเป็นต้องใช้ 8088 ไมโครโปรเซสเซอร์อีกหนึ่งตัวไปควบคุมการทำงานของ TMS32010 ในขณะที่ไอบีเอ็มพีซี ตัดการควบคุมจากการ์ดนี้

ในส่วนฮาร์ดแวร์ของ 8088-slave จะประกอบด้วย

3.1.2.1 8088 ไมโครโปรเซสเซอร์

8088-slave นี้จะควบคุมการทำงานได้โดยการควบคุมขา RESET และขา READY สัญญาณที่ควบคุมขา RESET คือสัญญาณ WAIT/RST ส่วนสัญญาณที่ควบคุมขา READY คือสัญญาณ STATUS88 8088 จะถูกต่อในลักษณะมินิมั่มโหมด(minimum mode) เพื่อให้วงจรฮาร์ดแวร์ส่วนนี้ออกแบบได้ง่ายและไม่ซับซ้อน

3.1.2.2 8284 ตัวกำเนิดสัญญาณนาฬิกา

ทำหน้าที่ในการสร้างสัญญาณนาฬิกาเพื่อป้อนให้กับ 8088-slave โดยใช้คริสตอล (crystal) ความถี่ 20 เมกกะเฮิร์ตมาต่อคร่อมที่ขา X_1 และ X_2 ก็จะได้สัญญาณนาฬิกาที่เป็นเอาท์พุทสองสัญญาณคือ สัญญาณ CLK และสัญญาณ CLK20

- สัญญาณ CLK จะมีความถี่เป็น 1/3 เท่าของความถี่คริสตอลนำไปป้อนให้กับ 8088-slave เพื่อใช้เป็นสัญญาณนาฬิกาของระบบ
- สัญญาณ CLK20 เป็นสัญญาณที่นำมาจากขา OSC ของ 8284 มีความถี่เท่ากับความถี่ของคริสตอลคือ 20 เมกกะเฮิร์ต สัญญาณ CLK20 นี้นำไปใช้ป้อนให้เป็นสัญญาณนาฬิกาของ TMS32010 ต่อไป

นอกจากนี้ 8284 ยังมีหน้าที่กำหนดสัญญาณ READY และ RESET ให้กับ 8088-slave โดยใช้สัญญาณ WAIT/$\overline{\text{RST}}$ และ STATUS88 เพื่อให้ไอบีเอ็มพีซีสามารถควบคุมการทำงานของ 8088-slave ได้

สัญญาณ READY ของ 8088-slave จะถูกใช้เป็นสัญญาณ $\overline{\text{FORCE88}}$ ซึ่งเอาไว้ในการควบคุมหน่วยความจำของการ์ดนี้ว่าจะถูกแอ็กเซสโดยไอบีเอ็มพีซี หรือแอ็กเซสโดย 8088-slave ถ้าหากว่าสัญญาณ $\overline{\text{FORCE88}}$ นี้มีสถานะเป็น 0 แสดงว่าขณะนั้น 8088-slave ยังอยู่ในสภาวะรอหรือถูกรีเซ็ทอยู่ดังนั้นไอบีเอ็มพีซี จะเป็นผู้แอ็กเซสหน่วยความจำทั้งหมดบนการ์ดนี้ แต่ถ้าหากว่าสัญญาณ $\overline{\text{FORCE88}}$ มีสถานะเป็น 1 อยู่นั่นคือ 8088-slave จะทำงานและเป็นตัวแอ็กเซสหน่วยความาจำบนการ์ดนี้ ในขณะเดียวกับที่ไอบีเอ็มพีซีถูกตัดขาดการควบคุมการ์ดนี้ นอกจากนี้ก็ได้มีการต่อแอลอีดี(LED) กับสัญญาณนี้ด้วยเพื่อไว้ตรวจสอบการทำงานของ 8088-slave

3.1.2.3 พอร์ท

ในส่วนของ 8088-slave จะมีทั้งอินพุทพอร์ทและเอาท์พุทพอร์ท โดยใช้ 74LS155-2เป็นตัวดีโค้ดเดอร์

- อินพุทพอร์ททำหน้าที่สำหรับรับค่าสัญญาณควบคุม TMS32010 เมื่อ 8088-slave ต้องการสัญญาณควบคุมนี้คือ สัญญาณ $\overline{\text{FORCEDSP}}$, และสัญญาณ STATUSDSP
- เอาท์พุทพอร์ททำหน้าที่ควบคุมสัญญาณ STATUS88ร่วมกับสัญญาณ $\overline{\text{GO88}}$ ที่มาจากไอบีเอ็มพีซี สัญญาณ STATUS88 นี้จะคงสถานะไว้ด้วยอาร์เอสฟลิบฟล็อป(RS FLIP-FLOP) จนกว่าจะมีการสั่งผ่านสัญญาณ $\overline{\text{GO88}}$ หรือ 8088-slave สั่งเอาท์พอร์ทเพื่อให้ตัวเองอยู่ในสถานะ wait เมื่อ TMS32010 ทำงาน

สัญญาณ $\overline{\text{GODSP2}}$ เป็นสัญญาณที่ใช้สั่งให้ TMS32010 ทำงานเช่นเดียวกับสัญญาณ $\overline{\text{GODSP1}}$ ดังนั้น การสั่งให้ TMS32010 ทำงานจึงสามารถสั่งจากไอบีเอ็มพีซี หรือจาก 8088-slave ก็ได้ แต่ต้องหลังจากที่ 8088-slave ทำงานแล้วเท่านั้น ทั้งสัญญาณ $\overline{\text{GODSP1}}$ และ $\overline{\text{GODSP2}}$ จะเอาเข้าไปเป็นอินพุทของอาร์เอสฟลิบฟล็อป (74LS10) เพื่อเป็นตัวแลทช์สัญญาณ FORCEDSP

นอกจากนี้ ก็ยังมีดาต้าบัฟเฟอร์ 74LS245-4, แอดเดรสบัฟเฟอร์ 74LS244-5, 74LS373 ซึ่งทำหน้าที่เป็นตัวปลดส่วนนี้ออกจากระบบเมื่ออยู่ในสภาวะรีเซ็ท

3.1.3 ส่วนของการติดต่อกับ TMS32010

ในส่วนของ TMS32010 นี้มีหน้าที่ในการรับข้อมูลที่ไอบีเอ็มพีซีต้องการให้ประมวลผลจากหน่วยความจำบนการ์ดนี้เข้าสู่หน่วยความจำข้อมูล(data memory) ภายในตัว TMS32010 จากนั้นก็ทำการประมวลผล เมื่อประมวลผลเสร็จแล้วจึงทำการถ่ายข้อมูลนี้กลับไปเก็บไว้ในหน่วยความจำบนการ์ด

ฮาร์ดแวร์ของส่วนนี้ประกอบไปด้วย

3.1.3.1 TMS32010

บนการ์ดนี้จะใช้ TMS32010 ในโหมดของไมโครโปรเซสเซอร์ ดังนั้นขา $\overline{\text{MP}}$/MC จึงต้องต่อลงกราวน์ สัญญาณนาฬิกาสำหรับ TMS32010 ขนาดความถี่ 20 เมกกะเฮิร์ตจะ

นำมาจากสัญญาณ CLK20 ในส่วนของ 8088-slave มาป้อนเข้าที่ขา X2 และพูลอัพ (pull up) กับไฟเลี้ยง

สำหรับขา $\overline{DEN}$, $\overline{BIO}$, $\overline{INT}$ ในการ์ดนี้ไม่มีความจำเป็นต้องใช้งาน ดังนั้นจึงนำไปพูลอัพกับไฟเลี้ยงไว้เช่นกัน

ขาควบคุมที่ต้องใช้ในการติดต่อกับหน่วยความจำโปรแกรมภายนอก กับเอาท์พุทพอร์ทคือขา $\overline{MEN}$ และขา $\overline{WE}$ การดีโค้ดสัญญาณเพื่อให้ได้ความแตกต่างระหว่างสัญญาณเขียนอ่านหน่วยความจำโปรแกรมและสัญญาณเอาท์ข้อมูลของเอาท์พุทพอร์ททำได้โดยการใช้ 74LS139 ดีโค้ดสัญญาณ $\overline{MEN}$, $\overline{WE}$ ร่วมกับ A3-A11 ของ TMS32010 ผ่านลอจิกเกท 74LS00,74LS33 สัญญาณควบคุมที่ดีโค้ดได้มีดังนี้

- สัญญาณ $\overline{DWR}$ เป็นสัญญาณที่ใช้ในการเอนาเบิล(enable) เอาท์พุทพอร์ท
- สัญญาณ $\overline{DSP-WE}$ เป็นสัญญาณสำหรับการอ่านหรือเขียนหน่วยความจำโปรแกรม
- สัญญาณ $\overline{DSP-HE}$, $\overline{DSP-LE}$ เป็นสัญญาณใช้สำหรับเอนาเบิล CS ของหน่วยความจำโปรแกรมถ้า TMS32010 เป็นตัวแอ็กเซสหน่วยความจำโปรแกรมสัญญาณทั้งสองนี้จะเป็นสัญญาณเดียวกัน แต่ถ้าหน่วยความจำโปรแกรมส่วนนี้ถูกแอ็กเซสโดยไอบีเอ็มพีซีหรือ 8088-slave จะต้องแยกการแอ็กเซสเป็น 8-บิท บน($\overline{DSP-HE}$) และ 8-บิท ล่าง ($\overline{DSP-LE}$) เนื่องจาก CPU 8088 แอ็กเซสข้อมูลได้ทีละ 8 บิทเท่านั้น

เนื่องจากในขณะที่ TMS32010 อยู่ในสภาวะถูกปลดออกจากระบบ(ถูกรีเซ็ท) ขาสัญญาณต่างๆไม่อยู่ในสถานะไฮอิมพีแดนซ์(high impedance) ยกเว้นดาต้าบัสซึ่งโดยปกติก็อยู่ในสถานะไฮอิมพีแดนซ์อยู่แล้ว ดังนั้นเพื่อที่จะสามารถทำให้ TMS32010 ถูกปลดออกจากระบบได้อย่างแท้จริงในขณะที่ TMS32010 ถูกรีเซ็ท จึงต้องอาศัย 74LS244 จำนวน 2 ตัวมาเป็นบัฟเฟอร์ก่อนที่จะต่อเข้ากับระบบบัสของการ์ด

3.1.3.2 หน่วยความจำ

เป็นหน่วยความจำโปรแกรมให้กับ TMS32010 บนการ์ดนี้ใช้สแตติกแรม(STATIC RAM) HM6168 HP-55 มีขนาดตัวละ 4kx4 บิท ใช้ทั้งหมด 4 ตัว แรมที่ใช้นี้เป็นแรมความเร็วสูง(high speed RAM) ที่มีแอ็กเซสไทม์(access time) 55 นาโนวินาที เพื่อให้สามารถทำงานทันกับ TMS32010 ซึ่งทำงานที่ความถี่ 20 เมกกะเฮิร์ต

3.1.3.3 เอาท์พุทพอร์ท

ในส่วนของการติดต่อกับ TMS32010 จำเป็นต้องใช้เอาท์พุทพอร์ทสำหรับบอกสถานะการทำงานของ TMS32010 (STATUSDSP) การดีโค้ดเอาท์พุทพอร์ทใช้แอดเดรสบัส A0-A2 ร่วมกับสัญญาณ DWR

การควบคุมการทำงานของ TMS32010 ใช้การรีเซ็ทโดยอาศัยสัญญาณ $\overline{\text{FORCEDSP}}$ ควบคุมการทำงานของ TMS32010

3.1.4 ส่วนของหน่วยความจำบนการ์ด

เนื่องจากบนการ์ดนี้มีการใช้ไมโครโปรเซสเซอร์ถึงสองตัวมาทำงานร่วมกัน ดังนั้นจึงต้องมีการจัดตำแหน่งของหน่วยความจำให้เหมาะสม หน่วยความจำบนการ์ดนี้แบ่งได้เป็น 2 ส่วนคือ ส่วนที่เป็นของ 8088-slave และ ส่วนที่เป็นของ TMS32010 โดยปกติแล้วไอบีเอ็มพีซีจะเป็นตัวแอ็กเซสหน่วยความจำทั้งสองส่วนนี้ เพื่อเป็นการดาวน์โหลด(down load) ข้อมูลและมอนิเตอร์โปรแกรมให้แก่ 8088-slave และ TMS32010 เมื่อ 8088-slave ทำงานแล้วไอบีเอ็มพีซีก็จะถูกตัดออกจากระบบไม่สามารถแอ็กเซสหน่วยความจำบนการ์ดได้อีก แต่ก็ยังสามารถทำการรีเซ็ททั้ง 8088-slave และ TMS32010 เพื่อยกเลิกการทำงานได้ตลอดเวลา การแอ็กเซสหน่วยความจำของไมโครโปรเซสเซอร์บนการ์ดนี้จะแอ็กเซสได้ครั้งละหนึ่งตัวเท่านั้น

3.1.4.1 หน่วยความจำที่เป็นส่วนของ 8088-slave

- ประกอบด้วยสแตติกแรม 65256 (32kx8 bit) ที่ใช้เป็นที่เก็บข้อมูลที่ต้องการจะให้ TMS32010 ประมวลผลและเก็บข้อมูลหลังจากที่ TMS32010 ประมวลผลเสร็จแล้วมีขนาด 64 กิโลไบท์ หน่วยความจำส่วนนี้ไอบีเอ็มพีซีจะแอ็กเซสที่แอดเดรส A0000H-AFFFFH ของแบงค์ 0 ส่วน 8088-slave จะแอ็กเซสที่แอดเดรส 00000H-7FFFFH
- ส่วนที่เป็นมอนิเตอร์โปรแกรม ประกอบด้วยสแตติกแรม 65256 เป็นจำนวน 1 ตัวไอบีเอ็มพีซีจะมองเห็นหน่วยความจำนี้ที่ตำแหน่ง A8000H-AFFFFH ของแบงค์ 0 ส่วน 8088-slave จะแอ็กเซสที่ตำแหน่ง 88000H-8FFFFH

3.1.4.2 หน่วยความจำที่เป็นของ TMS32010

หน่วยความจำนี้จะเป็นหน่วยความจำโปรแกรมสำหรับ TMS32010 มีขนาด 8 กิโลไบท์ ไอบีเอ็มพีซีจะแอ็กเซสที่แอดเดรส A0000H-A1FFFH และ 8088-slave จะแอ็กเซสที่แอดเดรส 80000H-81FFFH

ในการจัดฮาร์ดแวร์ส่วนนี้ สำหรับการแอ็กเซสหน่วยความจำที่เป็นส่วนของ 8088-slave ใช้ 74LS157 ทำการมัลติเพล็กซ์(multiplex) สัญญาณ read,write ระหว่างไอบีเอ็มพีซีกับ 8088-slave แต่ถ้าเป็นการแอ็กเซสหน่วยความจำของ TMS32010 ต้องใช้ 74LS139 เพื่อแยกสัญญาณให้ได้ $\overline{\text{DSP-HE}}$ และ $\overline{\text{DSP-LE}}$

วงจรของฮาร์ดแวร์นี้สามารถดูได้จากภาคผนวกท้ายเล่มของปริญญานิพนธ์ฉบับนี้

3.2 การพัฒนาทางด้านซอฟท์แวร์

หลังจากได้ทำการพัฒนาส่วนของฮาร์ดแวร์แล้ว การเขียนซอฟท์แวร์เพื่อให้ไปควบคุมการทำงานของฮาร์ดแวร์ให้เป็นไปตามเป้าหมายหรือจุดประสงค์ของงานที่จะทำ โดยจะเริ่มจากกการเขียนโปรแกรมให้ไปควบคุมการทำงานของระบบก่อน แล้วถึงจะสั่งให้ระบบทำงานตามที่ผู้ใช้ต้องการอีกที ดังนั้นในส่วนของการพัฒนาซอฟท์แวร์จะแยกออกเป็น 2 ส่วนใหญ่ๆคือ

1 โปรแกรมควบคุมการทำงานของระบบ

2 โปรแกรมประยุกต์ใช้งานของ TMS32010

3.2.1 โปรแกรมควบคุมการทำงานของระบบ

ในส่วนของโปรแกรมควบคุมการทำงานของระบบ จะทำหน้าที่ในการเซ็ทค่าต่างๆของระบบให้เป็นค่าเริ่มต้น อันได้แก่การเคลียร์หน่วยความจำของระบบทั้งหมด(ของ 8088-slave และของ TMS32010) ให้มีค่าเป็นศูนย์เพื่อไม่ให้ไปกวนการทำงานของระบบได้ หลังจากนั้นจะเริ่มทำการดาวน์โหลด(down load) โปรแกรมเริ่มต้นทำงานของ 8088-slave และท้ายสุดก็คือโปรแกรมการทำงานของตัว TMS32010 หลังจากการทำดาวน์โหลดโปรแกรมทุกตัวลงในหน่วยความจำของระบบแล้ว ก็จะทำการสั่งให้ TMS32010

ทำงานอีกทอดหนึ่ง ซึ่ง TMS32010 ก็จะเริ่มทำงานตามที่โปรแกรมไว้ นับจากนี้ไป เมื่อ TMS32010 ทำงานเสร็จก็จะส่งสถานะกลับมาให้ไอบีเอ็มพีซีทราบ ซึ่งไอบีเอ็มพีซีจะอาศัยสถานะนี้เพื่อตัดสินใจในการหยุดการทำงานของตัว TMS32010 หรือ 8088-slave หรือทั้งสองตัว ทั้งหมดนี้เป็นการทำงานอย่างคร่าวๆของระบบประมวลผลด้วยความเร็วสูงในรายละเอียดของโปรแกรมการทำงานของแต่ละส่วนจะแยกกล่าวเป็นหัวข้อย่อยๆดังนี้

- การเคลียร์หน่วยความจำของระบบ
- การดาวน์โหลดโปรแกรมเริ่มต้นการทำงานของ 8088-slave
- การดาวน์โหลดโปรแกรมการทำงานของ 8088-slave
- การดาวน์โหลดโปรแกรมการทำงานของ TMS32010
- การสั่งการให้ระบบเริ่มทำงาน
- การสั่งการให้ระบบหยุดทำงาน

3.2.1.1 <u>การเคลียร์หน่วยความจำของระบบ</u>

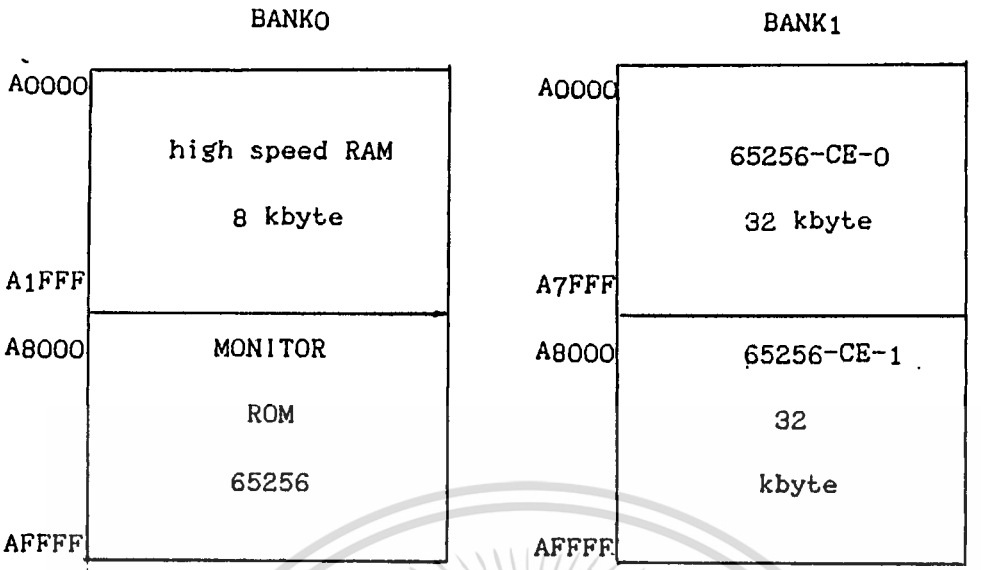
ก่อนที่จะกล่าวถึงวิธีการเคลียร์หน่วยความจำของระบบ จะขอพูดถึงเรื่องหน่วยความจำของระบบก่อน หน่วยความจำของระบบจะแบ่งออกเป็น 2 ส่วน ขึ้นอยู่กับว่าไมโครโปรเซสเซอร์ 8088-slave หรือไอบีเอ็มพีซีมอง

<u>การจัดหน่วยความจำเมื่อไอบีเอ็มพีซีเป็นตัวแอ็กเซส</u>

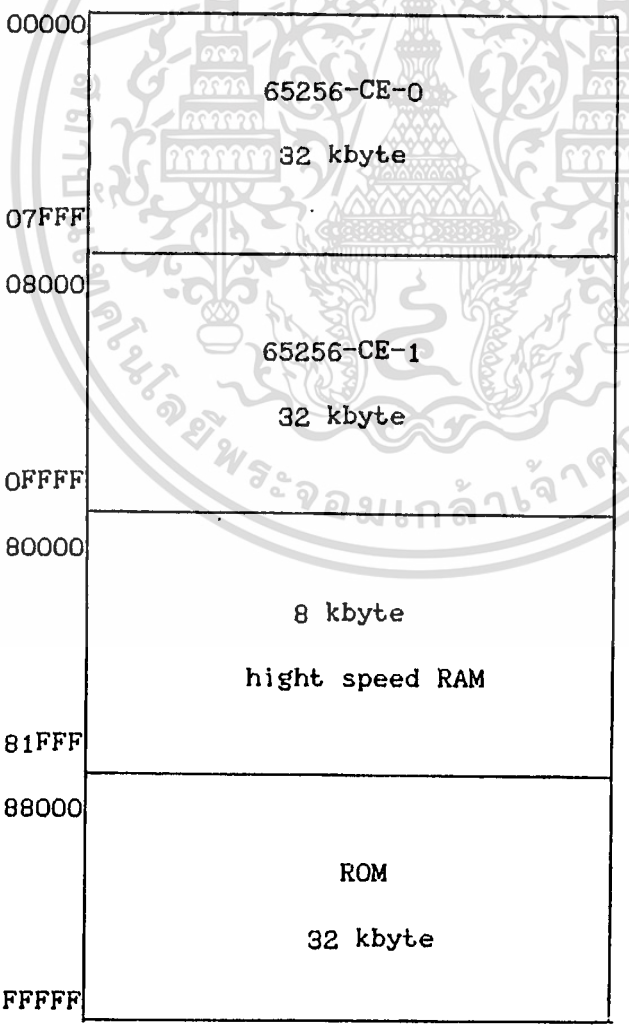
เมื่อไอบีเอ็มพีซีเป็นฝ่ายมองหน่วยความจำของระบบ ไอบีเอ็มพีซีจะจัดหน่วยความจำออกเป็น 2 แบงค์(bank) คือ BANK0 และ BANK1 โดยในแต่ละแบงค์จะมีขนาด 64 เคไบท์ ใช้ 65256 ที่มีขนาด 32 เคไบท์ จำนวน 2 ตัว ซึ่งได้แสดงไว้ในรูปที่ 3.1

<u>การจัดหน่วยความจำเมื่อ 8088-slave เป็นตัวแอ็กเซส</u>

เมื่อ 8088-slave เป็นฝ่ายมองหน่วยความจำของระบบ 8088-slave จะมองหน่วยความจำทั้งหมดเป็นแบงค์เดียวกัน และจะมีเพียง 1 แบงค์เท่านั้น โดยมีรูปแบบของการจัดหน่วยความจำดังที่แสดงไว้ในรูปที่ 3.2



รูปที่ 3.1 แสดงการจัดหน่วยความจำเมื่อไอบีเอ็มพีซีเป็นฝ่ายแอ็กเซส



รูปที่ 3.2 แสดงการจัดหน่วยความจำเมื่อ8088-slave เป็นฝ่ายแอ็กเซส

เมื่อเราทำการเคลียร์หน่วยความจำโดยใช้คำสั่งจากไอบีเอ็มพีซี เป็นตัวเคลียร์หน่วยความจำนี้ เราต้องทำการเคลียร์ทั้ง 2 แบงค์ โดยมีขั้นตอนดังนี้

```
MOV DX,3EE
OUT DX,00
MOV AX,A000
MOV DS,AX
MOV CX,FFFF
MOV AL,00
MOVB
REP
```

จากชุดคำสั่งจะต้องทำการเอาท์ค่าคอนโทรลเวิร์ดออกทางพอร์ทหมายเลข 3EEH ก่อน แล้วจึงทำการย้ายค่า 00 จำนวน 64 เคไบท์ ค่าลงในหน่วยความจำโดยเริ่มจากตำแหน่ง A0000H เป็นต้นไปจนถึง AFFFFH

เมื่อเคลียร์หน่วยความจำใน BANK0 แล้ว ก็ต้องมาเคลียร์หน่วยความจำอีกแบงค์ คือ BANK 1 โดยใช้ชุดคำสั่งดังนี้

```
MOV DX,3EE
OUT DX,80
MOV AX,A000
MOV DS,AX
MOV AL,00
MOVB
REP
```

หลังจากชุดคำสั่งนี้ หน่วยความจำใน BANK 1 คือ 65256 ตัวที่ 1 และ 2 ก็จะถูกเคลียร์ให้เป็น 0 หมด เมื่อทำการเคลียร์หน่วยความจำทั้งหมดของระบบต่อไปก็สามารถดาวน์โหลดโปรแกรมควบคุมการทำงานของระบบในขั้นตอนต่อไป

3.2.1.2 การดาวน์โหลดโปรแกรมเริ่มต้นการทำงานของ 8088-slave

โปรแกรมในส่วนนี้มีชื่อว่า TMSROM1.TXT โปรแกรมการทำงานเริ่มต้นของ 8088-slave นี้คือโปรแกรมที่จะทำการโหลดลงตำแหน่งแรกสุดที่ 8088-slave เริ่มต้นทำงานหลังจากถูกรีเซ็ท ซึ่งก็คือตำแหน่งที่ FFFF0H หรือ FFFF:0000 ส่วนใหญ่แล้วโปรแกรมในส่วนนี้ก็คือคำสั่งกระโดดไปยังตำแหน่ง ที่โปรแกรมควบคุมการทำงานจริงของระบบ ในที่นี้จะใช้ตำแหน่ง 8000:8000 เป็นตำแหน่งที่จะกระโดดไป ดังชุดคำสั่งข้างล่างนี้

```
JMP 8000:8000
```

หลังจากชุดคำสั่งนี้ 8088-slave จะกระโดดหรือย้ายการทำงานไปตำแหน่ง 8800:0000 ซึ่งก็คือเป็นตำแหน่งของมอนิเตอร์รอมขนาด 32 เคไบท์

3.2.1.3 การดาวน์โหลดโปรแกรมการทำงานของ 8088-slave

โปรแกรมในส่วนนี้มีชื่อว่า TMSROM2.TXT ซึ่งจะเป็นโปรแกรมการทำงานจริงของ 8088-slave ซึ่งมีหน้าที่ควบคุมการทำงานของ TMS32010 และคอยติดต่อกับไอบีเอ็มพีซี โปรแกรมส่วนนี้จะถูกดาวน์โหลดลงตำแหน่ง 8800:0000 ก็คือตำแหน่งของมอนิเตอร์รอม โปรแกรมส่วนนี้จะทดสอบการทำงานของ 8088-slave โดยการใส่ค่าในหน่วยความจำใน 65256-0,65256-1 ด้วยค่า 22H ที่แอดเดรส 00000H จำนวน 2 เคไบท์ จากนั้นก็ทำการย้ายข้อมูลนี้จากตำแหน่ง 00000H จำนวน 2 เคไบท์ ไปยังตำแหน่ง 80000H หลังจากนั้นก็จะสั่งให้ TMS32010 ทำงาน รายละเอียดของตัวโปรแกรมเป็นดังนี้

```
MOV AX,0000
MOV ES,AX
MOV DI,0000
MOV CX,1000
MOV AL,22
CLD
REPZ
```

```
        STOSB
        MOV AX,0000
        MOV DS,AX
        MOV AX,0000
        MOV ES,AX
        MOV SI,0000
        MOV DI,0000
        MOV CX,1000
        REPZ
        MOVSB
        MOV AX,300
        MOV DX,300
        OUT DX,AX
        MOV DX,0000
LABLE   NOP
        JMP LABEL
```

3.2.1.4 การดาวน์โหลดโปรแกรมการทำงานของ TMS32010

โปรแกรมการทำงานของ TMS32010 ก็คือโปรแกรมประยุกต์ในงานที่ผู้ใช้ต้องการให้ระบบทำให้ โปรแกรมส่วนนี้มีชื่อว่า TMSROM3.TXT โปรแกรมส่วนนี้เขียนด้วยภาษาแอสเซมบลีของ TMS32010 โดยจะเริ่มการทำงานหลังจากถูกสั่งการให้ทำงานด้วย 8088-slave เมื่อทำงานเสร็จแล้วก็จะทำการส่งสถานะกลับมาให้ไอบีเอ็มพีซีทราบ เพื่อที่ไอบีเอ็มพีซีจะได้ทำการหยุดการทำงานของ TMS32010

โปรแกรมการทำงานของ TMS32010 นี้มีอยู่ด้วยกันหลายโปรแกรมซึ่งในโครงงานนี้ได้แก่โปรแกรมสร้างสัญญาณคลื่นรูปไซน์(sine-wave generator) โปรแกรมการคอนโวลูชั่น(convolution) เป็นต้น รายละเอียดของแต่ละโปรแกรมและทฤษฎีจะได้กล่าวในหัวข้อค่างหากต่อไป ต่อไปจะอธิบายถึงวิธีการเตรียมโปรแกรม TMSROM3.TXT

3.2.1.5 การสั่งให้ระบบเริ่มทำงาน

การสั่งให้ระบบเริ่มทำงานต้องเริ่มด้วยการสั่งงานให้ 8088-slave เริ่มทำงานก่อนแล้วให้ 8088-slave ไปสั่งการให้ TMS32010 ทำงานอีกทอดหนึ่ง ดังนั้นหน้าที่ของไอบีเอ็มพีซีก็คือการสั่งการให้ 8088-slave ทำงาน ซึ่งใช้คำสั่งดังต่อไปนี้

```
MOV DX,3EE
OUT DX,42
```

หลังจากชุดคำสั่งนี้ 8088-slave จะเริ่มทำงาน หน้าที่ต่อไปของไอบีเอ็มพีซีคือคอยเช็คสถานะการทำงานของระบบว่าเสร็จสิ้นจากงานที่ให้ทำแล้วหรือยัง โดยวิธีการเช็คนี้จะใช้การวนลูปรับค่าจากพอร์ทเอาท์พุทหมายเลข 3EEH แล้วเช็คว่าค่าที่รับมาตรงกับค่าสถานะที่เซ็ตไว้หรือไม่

```
          MOV DX,3EE
LOOPCHECK: IN AL,DX
          CMP AL,74
          JNZ LOOPCHECK
```

3.2.1.6 การสั่งการให้ระบบหยุดทำงาน

เมื่อทำการเช็คค่าที่รับเข้ามาจากพอร์ทแล้วตรงกับค่าที่เซ็ตไว้ ไอบีเอ็มพีซีจะหยุดการทำงานของระบบโดยการสั่งเอาท์ค่า 00 ทางเอาท์พุทพอร์ทหมายเลข 3EEH ดังนี้

```
MOV DX,3EE
OUT DX,00
```

3.2.2 การเตรียมโปรแกรมการทำงานของ TMS32010

เริ่มแรกจากการใช้อิดิเตอร์(editor) ต่างๆเช่น Sidekick ,Wordstar ในการสร้างหรือเขียนโปรแกรมภาษาแอสเซมบลีของ TMS32010 เสร็จแล้ว ก็เก็บไฟล์นี้โดยตั้งชื่อไว้ สมมติให้ชื่อว่า TEST.ASM

โดยที่โปรแกรม TEST.ASM ที่โปรแกรมไว้เป็นดังนี้

```
          IDT   'TEST'
          AORG  >0
          B     START
ONE       EQU   1
TWO       EQU   2
START     LDPK  0
          LACK  1
          SACL  ONE
          SACL  TWO
LOOP      OUT   ONE,7
          NOP
          B     LOOP
          END
```

โปรแกรมนี้สั่งการให้ TMS32010 ทำการเก็บค่า 1 ไว้ที่หน่วยความจำข้อมูลตำแหน่งที่ 1 และเก็บค่า 2 ไว้ในหน่วยความจำข้อมูลตำแหน่งที่ 2 เสร็จแล้วให้วนรอบการส่งข้อมูลในหน่วยความจำข้อมูลออกทางพอร์ทหมายเลข 7

เมื่อได้ไฟล์ TEST.ASM นี้แล้ว ขั้นตอนต่อไปก็คือการคอมไพล์ให้ได้เป็นอ็อบเจคต์โค้ด เริ่มจากการออกจากอิดิเตอร์แล้วเข้าสู่พีซีดอส เสร็จแล้วป้อนชื่อตัวคอมไพเลอร์ของ TMS32010 ซึ่งมีชื่อว่า XASM3 ตามด้วยชื่อไฟล์ TEST.ASM

```
A>XASM3
```

เครื่องจะถามซอร์สไฟล์(source file)คือไฟล์ที่มีนามสกุล .ASM

และถามลิสติงไฟล์(listing file)คือไฟล์ที่เกิดตัวโปรแกรมและอ็อปเจคต์โค้ด

และถามอ็อปเจคต์ไฟล์(object file)คือไฟล์ที่เป็นเอาท์พุทจากการคอมไพล์ ซึ่งจะใช้ในการทำงานต่อไป

หลังจากได้ไฟล์ TEST.MPO เสร็จ จะต้องผ่านโปรแกรมแปลงอีกตัวเพื่อให้ได้อ็อป

เจ็คต์โค้ดที่แท้จริงซึ่งสามารถใช้กับระบบได้ โปรแกรมแปลงนี้มีชื่อว่า CONVERT.COM เอาท์พุทหลังจากการคอนเวิร์ทแล้วจะตั้งชื่ออะไรแล้วแต่ผู้ใช้

บทที่ 4

การทดลองและผลการทดลอง

การทดลองได้อาศัยฮาร์ดแวร์ที่ได้ออกแบบและต่อเสร็จใช้การได้ ในการพัฒนาโปรแกรมการทำงานของ TMS32010 โดยแบ่งขั้นตอนในการพัฒนาตัวโปรแกรมที่จะใช้ตรวจจับและเปรียบเทียบการทำงานของ TMS32010 กับไอบีเอ็มพีซีเอ็กซ์ที ลำดับขั้นตอนมีดังนี้

1 สร้างโปรแกรมที่ใช้ควบคุมการทำงานของระบบ โปรแกรมส่วนนี้แบ่งออกเป็น 2 ส่วน

- โปรแกรมในส่วนของไอบีเอ็มพีซี ได้แก่ส่วนที่ทำการเคลียร์หน่วยความจำของระบบแล้วทำการโหลดโปรแกรมของ 8088-slave ที่จะทำการควบคุม TMS32010 อีกทีหนึ่ง

- โปรแกรมในส่วนของ 8088-slave ในส่วนนี้ก็แบ่งเป็น 2 โปรแกรมคือโปรแกรมสั่งการ ณ ตำแหน่งเริ่มทำงานของ 8088-slave ซึ่งก็คือตำแหน่ง FFFF:0000 มีชื่อว่า TMSROM1.TXT และโปรแกรมการทำงานจริงหรือโปรแกรมควบคุมการทำงานของ TMS32010 ซึ่งมีชื่อว่า TMSROM2.TXT

โปรแกรมทั้ง 2 ส่วนนี้สร้างโดยใช้โปรแกรมดีบัก จึงเก็บอยู่ในรูปแบบของไฟล์ .COM สามารถที่จะโหลดเข้าหน่วยความจำได้ 2 ทางคือ โดยตัวโปรแกรมดีบักหรือโดยเขียนโปรแกรมให้ไอบีเอ็มพีซีเป็นตัวโหลดซึ่งจะกล่าวในหัวข้อถัดไป

2 สร้างโปรแกรมการทำงาน TMS32010 ซึ่งจะใช้เป็นตัววัดความเร็วในการทำงานของ TMS32010 โดยปกติ TMS32010 ทำงานโดยอาศัยเลขฐานสิบหก การเตรียมโปรแกรมจึงอาศัยซอฟท์แวร์ ที่ทำหน้าที่เป็นตัวคอมไพล์เลอร์ไว้แปลงโปรแกรม ที่เป็นภาษาแอสเซมบลีของ TMS32010 ให้เป็นอ็อปเจ็คต์ไฟล์ที่จะใช้ป้อนลงในหน่วยความจำ TMS32010 ด้วยวิธีการนี้จะต้องมาป้อนเลขฐานสิบหกทีละไบท์ลงในหน่วยความจำของ TMS32010 ซึ่งทำให้เสียเวลา จึงได้สร้างโปรแกรมแปลงเลขฐานสิบหกเหล่านี้เป็นแอสกีโค้ด(ASCII CODE) ซึ่งสามารถทำการดาวน์โหลดลงในหน่วยความจำ โดยใช้โปรแกรมดาวน์โหลดหรือโปรแกรมดีบักซึ่งทำให้สะดวกและรวดเร็วขึ้น โปรแกรมการแปลงนี้มีชื่อว่า

CONVERT.COM การเรียกใช้โปรแกรมการแปลงเลขฐานสิบหกเป็นระหัสแอสกีจะได้กล่าวถึงในส่วนของคู่มือการใช้โปรแกรม

3 ดาวน์โหลดโปรแกรมในขั้นตอนที่ 1 และ 2 ลงในหน่วยความจำของระบบ ขั้นตอนนี้สามารถทำได้ 2 วิธีคือ

3.1 โดยใช้โปรแกรมดีบัก ด้วยวิธีการนี้จะใช้คำสั่ง l(laod) ของโปรแกรมดีบักในการโหลดโปรแกรมดังกล่าวลงหน่วยความจำจะต้องใช้คำสั่งย้ายข้อมูล ในการย้ายข้อมูลไปยังตำแหน่งที่ต้องการอีกทีหนึ่ง จึงเป็นวิธีที่ค่อนข้างจะยุ่งยาก วิธีที่ดีกว่าจะกล่าวในหัวข้อข้างล่างนี้

3.2 โดยวิธีใช้โปรแกรม ST.EXE โปรแกรม ST.EXE เป็นโปรแกรมที่เขียนขึ้นเองโดยใช้ภาษาซีลิงค์กับภาษาแอสเซมบลี ทำหน้าที่ให้การเคลียร์หน่วยความจำของระบบ, โหลดโปรแกรมต่างๆที่เกี่ยวข้องกับการทำงานของระบบลงในหน่วยความจำ ตลอดจนสั่งการให้ระบบเริ่มทำงานและสั่งหยุดการทำงานของระบบเมื่อตรวจเช็คสถานะที่ระบบทำงานเสร็จ วิธีการใช้โปรแกรม ST.EXE จะได้กล่าวในหัวข้อคู่มือการใช้โปรแกรม

4 ใช้ไอบีเอ็มพีซีสั่งการให้ระบบเริ่มทำงานและทำการประมวลผลตามโปรแกรมการทำงานของ TMS32010 ที่เขียนขึ้นโดยไอบีเอ็มพีซีจะทำการจับเวลาตั้งแต่ระบบเริ่มต้นทำงานถึงสิ้นสุดการทำงานแล้วแสดงค่าเวลาที่ใช้ไปในหน่วยของวินาที การจับเวลานี้จะอยู่ในโปรแกรม ST.EXE

5 โปรแกรมรับค่าจากหน่วยความจำ TMS32010 เข้าสู่หน่วยความจำของไอบีเอ็มพีซี โปรแกรมนี้จะรับผลจากตำแหน่ง A100:0000 ซึ่งเป็นตำแหน่งที่ TMS32010 นำผลลัพธ์จากการคำนวณมาใส่ไว้ โปรแกรมในส่วนนี้มีชื่อว่า TMS.EXE ซึ่งจะอธิบายวิธีการใช้ในส่วนของคู่มือโปรแกรม

6 โปรแกรมแปลงค่าผลลัพธ์ข้อมูลที่ได้จากข้อ 5 ซึ่งจะอยู่ในรูปเลขฟิกซ์พอยต์ให้เป็นเลขโฟลทติ้งพอยต์ โปรแกรมนี้มีชื่อว่า FIXTOFLT.EXE วิธีการใช้โปรแกรมอยู่ในส่วนของคู่มือการใช้โปรแกรม

7 โปรแกรมสร้างกราฟจากข้อมูลที่ได้ทำการแปลงมาในหัวข้อที่ 6 โปรแกรมสร้างกราฟมีชื่อว่า GENGRAPH.EXE เขียนโดยใช้ภาษาซีบนเทอร์โบซีเวอร์ชั่น 2.0

ในส่วนของโปรแกรมการทำงานของ TMS32010 นับว่าเป็นส่วนที่สำคัญที่สุด โดยได้เขียนโปรแกรมการสร้างสัญญาณคลื่นรูปซายน์ ซึ่งรายละเอียดของทฤษฎีการสร้างก็ได้กล่าวมาข้างต้นแล้ว การทดลองเขียนโปรแกรมสร้างสัญญาณคลื่นรูปซายน์ได้ทำการเปรียบเทียบเวลาในการสร้างสัญญาณของตัว TMS32010 กับไอบีเอ็มพีซีและเครื่องคอมแพคเดสก์โปร 386/20 โปรแกรมที่ใช้เป็นตัวเปรียบเทียบเวลาการทำงานกับ TMS32010 มีชื่อว่า TSIN.EXE โปรแกรมนี้เขียนบนเทอร์โบซีเวอร์ชั่น 2.0 ใช้หลักการชี้ค่าจากตาราง (table lookup) เช่นเดียวกับโปรแกรม TMS32010 เพื่อให้มีอัลกอริธึมเหมือนกัน (รายละเอียดเกี่ยวกับการใช้โปรแกรมสามารถดูได้จากคู่มือการใช้โปรแกรม) เพื่อนำเอาโปรแกรมสร้างสัญญาณคลื่นรูปซายน์ทั้ง 2 อย่างนี้มาเปรียบเทียบกันเมื่อทดลองรันโปรแกรม ST.EXE บนเครื่องคอมแพคเดสก์โปร 386/20 กับไอบีเอ็มพีซีเอ็กซ์ที ก็ได้ผลเป็นตารางที่ 4.1 และ 4.2

การเปรียบเทียบนี้ทำโดยนับจำนวนจุดของการสร้างสัญญาณซายน์ โดยคิดเป็นขนาดของ 1 ลูปต่อ 1 จุด ดังนั้น 100x100 ก็เท่ากับ 10000 จุด ซึ่งก็ใช้ผลที่น่าพอใจมาก

นอกจากตารางเปรียบเทียบเวลาแล้ว โปรแกรมสำหรับดึงค่าซายน์ที่สร้างโดย TMS32010 มาแปลงเป็นเลขโฟลทติ้งพอยต์แล้วแสดงออกจอภาพกราฟฟิกก็สามารถแสดงผลออกมาเป็นรูปที่ 4.2

กราฟที่สร้างมานี้เป็นเพียงส่วนหนึ่งของจำนวนจุดที่สร้างโดย TMS32010 ทั้งนี้เพราะจากโปรแกรมสร้างสัญญาณซายน์ของ TMS32010 จะมีเนื้อที่ในการเก็บตัวแปรได้เพียง 2 เคไบท์ ซึ่งจะเก็บได้ 1 เคเวิร์ด หรือ 1 เคจุด(1จุดต่อ1เวิร์ด) แต่ในตารางเปรียบเทียบการทำงานมีขนาดถึง 100x100, 200x200 ,...ซึ่งมากกว่า 1 เคเวิร์ดแน่นอน ดังนั้นรูปกราฟนี้จึงเป็นเพียงการดึงมาแต่ส่วนหน้าของตัวแปรค่าซายน์ที่สร้างโดย TMS32010 ทั้งหมด

4.1 <u>โปรแกรมสร้างสัญญาณคลื่นรูปซายน์</u>

โปรแกรมสร้างสัญญาณคลื่นรูปซายน์นี้มีชื่อไฟล์ว่า GENER1.ASM ทำหน้าที่สร้างสัญญาณซายน์โดยวิธีการดูค่าจากตาราง(DIRECT TABLE LOOK UP METHOD) โปรแกรมนี้อาศัยข้อมูลจากตารางค่าซายน์ซึ่งมีความยาว 128 จุด หรือจุดละ 360/128 = 2.81 อง

ศา ค่าของการสเต็ปหรือเดลต้า(DELTA) เป็น 0.25

ข้อมูลเฉพาะของตัวโปรแกรมมีดังนี้

1 ใช้หน่วยความจำข้อมูลจำนวน 10 เวิร์ด

2 ใช้หน่วยความจำโปรแกรมไว้เก็บตารางค่าซายน์จำนวน 128 ค่า หรือกินเนื้อที่ไป 128 เวิร์ด

3 ใช้หน่วยความจำโปรแกรมไว้เก็บตัวโปรแกรมการทำงานจำนวน 50 เวิร์ด

4 ใช้หน่วยความจำโปรแกรมไว้เก็บค่าตัวแปรอื่นๆอีก 4 เวิร์ด

ขั้นตอนและผังงานการทำงานของโปรแกรมเป็นดังนี้

การทำงานของโปรแกรมนี้จะแสดงให้เห็นตามโฟลว์ชาร์ตในรูปที่ 4.1 ซึ่งจะอธิบายได้ดังนี้

เริ่มแรกจะทำการกำหนดค่าตัวแปรต่างๆที่ใช้ในโปรแกรมมีดังนี้

DELTA คือค่าของการสเต็ปกำหนดให้เป็น 0.25

ALPHA คือค่าของ N x DELTA ; N เป็นจำนวนของการสเต็ป

OFSET คือตำแหน่งของตารางค่าซายน์

LOOIN คือเคาน์เตอร์ลูปในกำหนดเป็น 100

LOOUP คือเคาน์เตอร์ลูปนอกกำหนดให้มีค่าเป็น 100

จำนวนสเต็ปทั้งหมดจะมีค่าเป็น LOOIN x LOOUT ในโปรแกรมนี้จะมีค่าเท่ากับ 10,000 เมื่อกำหนดค่าของลูปเคาน์เตอร์ แล้วก็เรียกใช้รูทีนย่อย ซึ่งมีทำหน้าที่ดังนี้

1 <u>คำนวณค่าอินเด็กซ์(INDEX)</u>

S = mod (N x DELTA , 128)

2 <u>อ่านค่าซายน์จากตาราง</u>

โดยอาศัยค่าอินเด็กซ์(S) ในการชี้ตำแหน่งในตารางค่าซายน์ แล้วอ่านเข้ามาในหน่วยความจำข้อมูลชื่อ SINA

เมื่อเสร็จ 2 ขั้นตอนนี้ก็จะออกจากรูทีนย่อยนี้ แล้วทำการเขียนลงในหน่วยความจำนี้

ตารางที่ 4.1

แสดงผลการเปรียบเทียบเวลาในการสร้างสัญญาณคลื่นรูปซายน์

ระหว่างคอมแพคเดสก์โปร 386/20 กับ TMS32010

TABLE 1 COMPARING SPEED OF GENERATE
SINE WAVE IN SECOND (SEC)

STEP POINT	COMPACT DESKPRO 386/20 INCLUDE 80387	TMS32010	RATIO= COMPACT/TMS32010
100X100	0.94	0.11	8.55
150X150	2.14	0.33	6.40
200X200	3.74	0.35	10.69
250X250	5.87	0.38	15.45
300X300	8.46	0.61	13.87
350X350	11.48	0.72	15.94
400X400	15.05	0.93	16.18
450X450	19.01	1.32	14.40
500X500	23.45	1.49	15.74

ตารางที่ 4.2

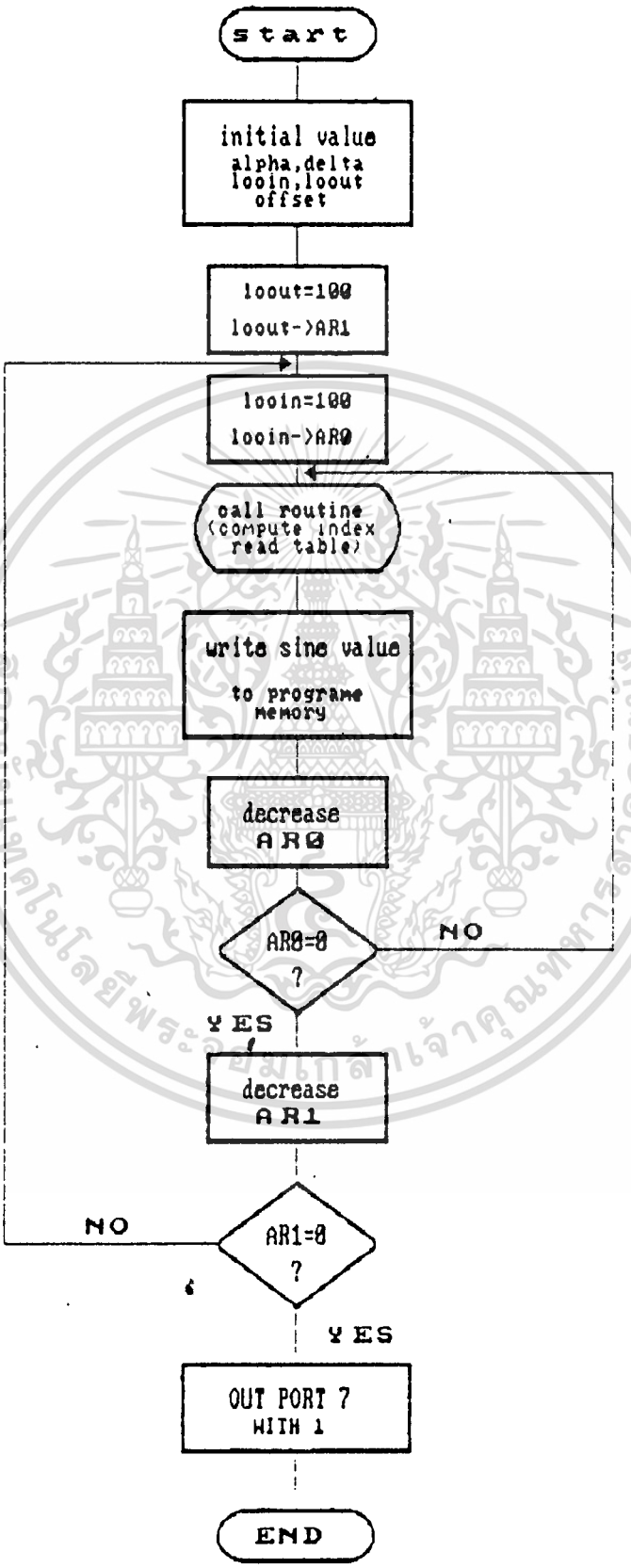
แสดงผลการเปรียบเทียบเวลาในการสร้างสัญญาณคลื่นรูปซายน์

ระหว่างไอบีเอ็มพีซีเอ็กซ์ที กับ TMS32010

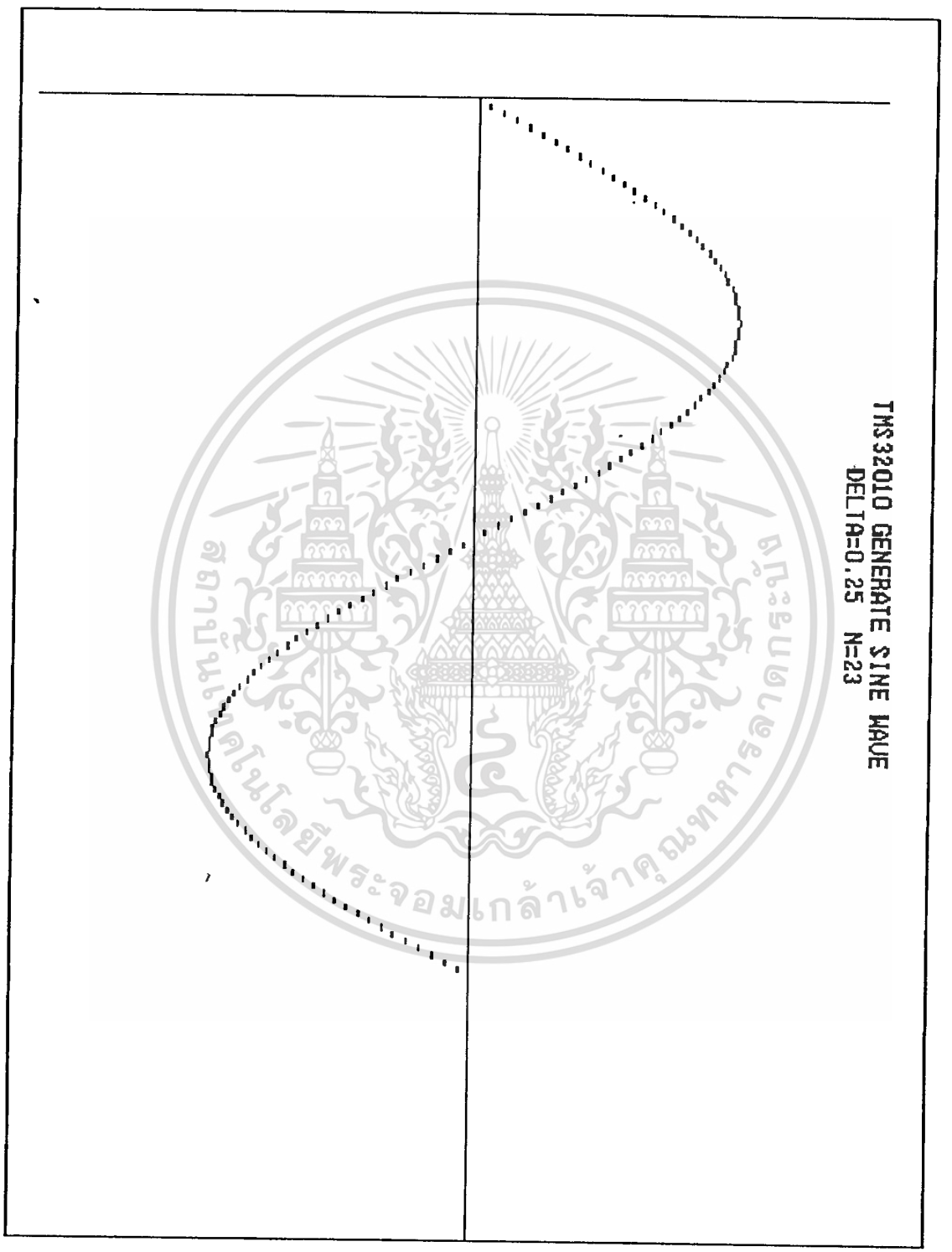
TABLE 2 COMPARING SPEED OF GENERATE SINE WAVE IN SECOND (SEC)

STEP POINT	IBM-PC XT (V-20)	TMS32010	RATIO= IBM-PC XT/ TMS32010
100X100	110.24	0.11	1002.18
150X150	246.29	0.33	746.33
200X200	437.81	0.35	1250.88
250X250	685.20	0.38	1803.16
300X300	982.56	0.61	1610.75
350X350	1334.80	0.72	1853.89
400X400	1763.28	0.93	1896.00
450X450	2231.46	1.32	1690.50
500X500	2755.40	1.49	1849.26

รูปที่ 4.1 แสดงโฟลว์ชาร์ตของการสร้างสัญญาณคลื่นรูปซายน์



รูปที่ 4.2 แสดงสัญญาแคลื่นรูปซายน์ที่ได้จากการสร้างของ TMS32010



4.2 คู่มือการใช้โปรแกรม

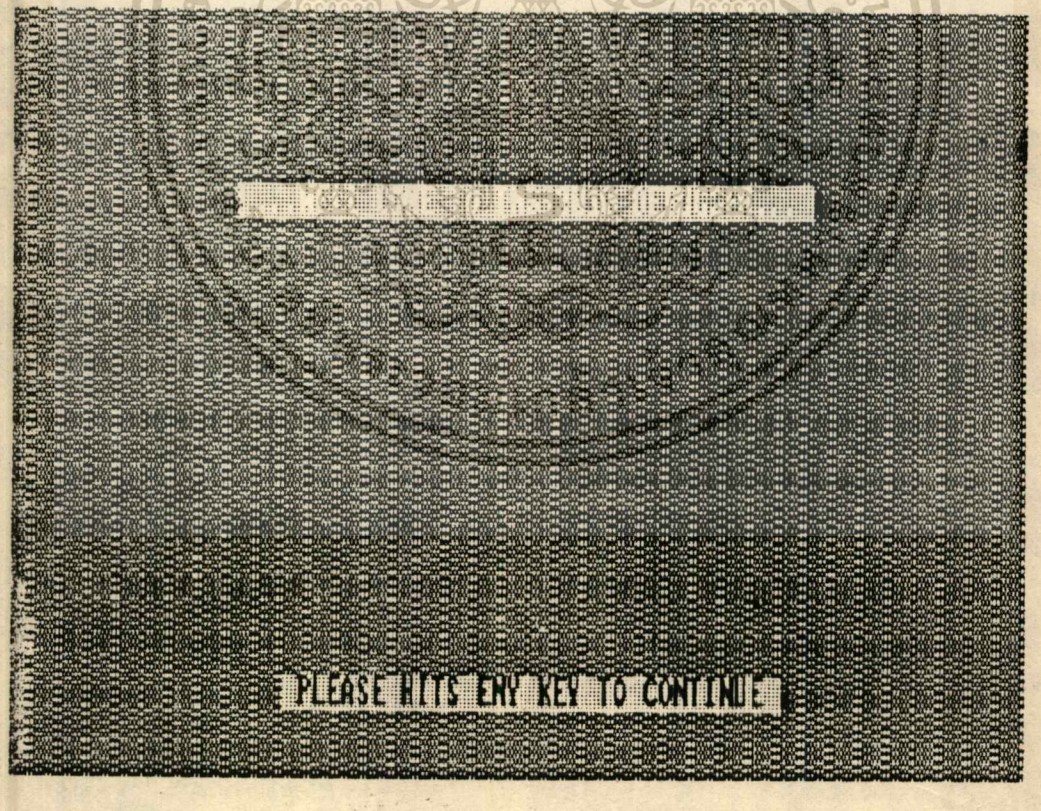
โปรแกรมที่ใช้ในปริญญานิพนธ์นี้มีอยู่ด้วยกันหลายอย่าง มีการเรียกใช้รูทีนย่อยซึ่งใช้ทำหน้าที่ต่างๆตามต้องการ โดยจะแยกกล่าวได้ดังนี้

4.2.1 โปรแกรม ST.EXE

โปรแกรมนี้เป็นโปรแกรมควบคุมการทำงานของระบบ เริ่มตั้งแต่การเปิดใช้หน่วยความจำของระบบด้วยการเอาท์พอร์ทการเลือกแบงค์ การเคลียร์หน่วยความจำ การดาวน์โหลดโปรแกรมของ 8088-slave และของ TMS32010 การสั่งการให้ TMS32010 ทำงาน และการหยุดการทำงานของระบบ โปรแกรม ST.EXE เขียนด้วยภาษาซีลิงค์กับภาษาแอสเซมบลี โดยใช้ภาษาซีเป็นโปรแกรมหลัก แล้วเรียกใช้รูทีนจากภาษาแอสเซมบลีที่คอยจัดการติดต่อกับระบบ ขั้นตอนในการเรียกใช้โปรแกรมมีดังนี้

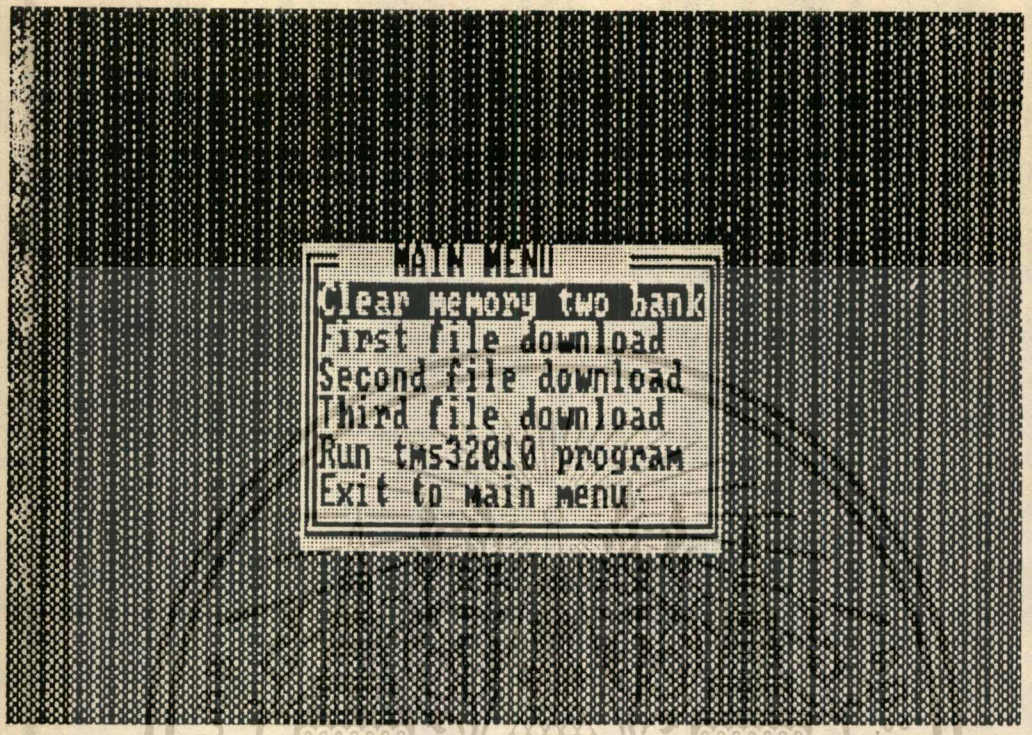
ป้อนชื่อ ST สมมติให้เครื่องหมายพร็อมท์<prompt> ในขณะนี้อยู่ไดรฟ์ A

A>ST แล้วกด RETURN จะปรากฏข้อความในจอดังรูปที่ 4.3



รูปที่ 4.3

ให้กดคีย์อะไรก็ได้ผ่านไปเรื่อยๆจนจอของเมนเมนู ซึ่งจะมีหน้าตาดังรูปที่ 4.4



รูปที่ 4.4

ในเมนเมนูจะมีให้เลือกฟังก์ชั่นการโอเปอร์เรท โดยการเลื่อนเคอร์เซอร์ขึ้นลง แล้วกด RETURN ฟังก์ชั่นแรกคือการเคลียร์หน่วยความจำทั้งสองแบงค์ เพื่อเคลียร์หน่วยความจำของระบบทั้งหมด ฟังก์ชั่นต่อไปคือ FIRST FILE DOWNLOAD ทำหน้าที่โหลดโปรแกรม TMSROM1.TXT เช่นเดียวกับ SECOND FILE DOWNLIAD และ THIRD FILE DOWNLOAD ที่จะโหลดโปรแกรม TMSROM2.TXT และ TMSROM3.TXT ตามลำดับ เมื่อเสร็จการดาวน์โหลดโปรแกรมต่อไปก็ให้เลื่อนเคอร์เซอร์มาที่ RUN TMS32010 PROGRAM แล้วกด RETURN เป็นการสั่งให้ TMS32010 ทำงาน ซึ่งสามารถแสดงให้เห็นดังรูปที่ 4.5

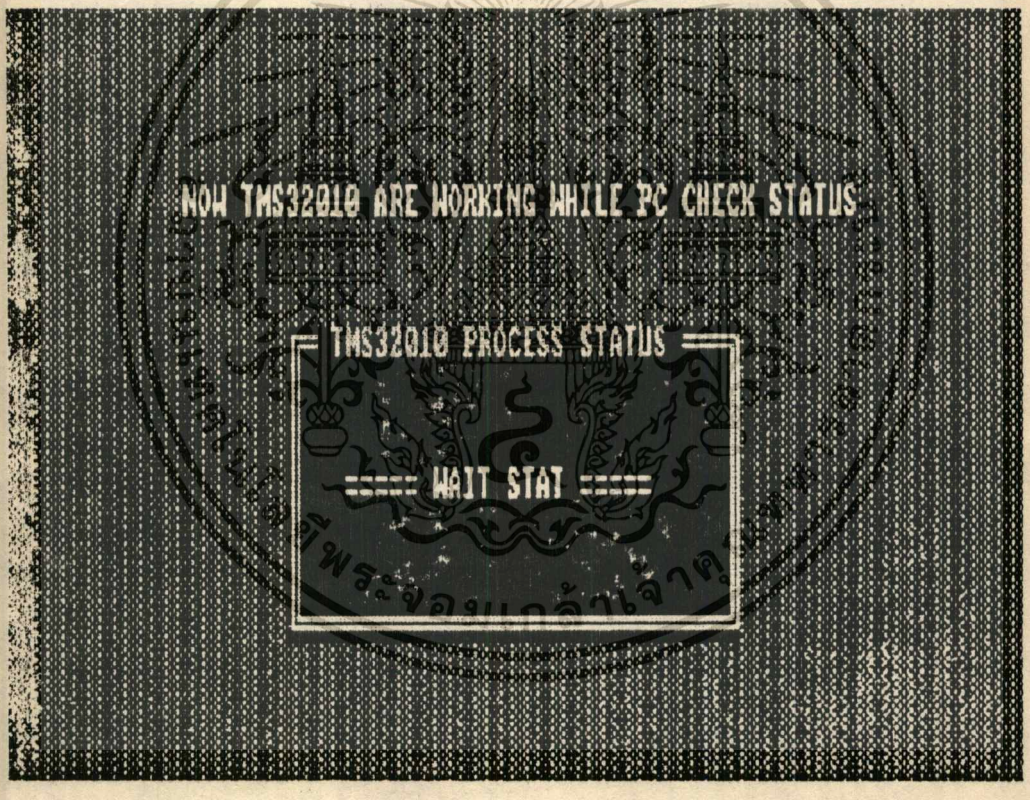
ขณะที่ TMS32010 ทำงานอยู่ข้อความ wait stat จะปรากฏตลอดเวลา เมื่อ TMS32010 ทำงานเสร็จก็จะมีเวลาบอกในวินโดว์(window) TMS32010 PROCESS STATUS ได้แสดงเป็นหน่วยวินาที เมื่อผู้ใช้ต้องการกลับเข้าสู่เมนเมนูก็ทำได้โดยการกดคีย์อะไรก็ได้ โปรแกรมก็จะหลุดมาเข้าเมนเมนูตามปกติ

4.2.2 โปรแกรม FIXTOFLT.EXE

โปรแกรมนี้ทำหน้าที่แปลงเลขฟิกซ์พอยต์ในรูปแบบคิวต่างๆ ในโปรแกรมสร้างสัญญาณคลื่นรูปซายน์ใช้คิว 15 โปรแกรมจะไปรับข้อมูลเลขฟิกซ์พอยต์จากตำแหน่ง A100:0000 แล้วแปลงเป็นเลขโฟลทพอยต์มาเก็บในตัวแปรโฟลท เริ่มเรียกใช้โปรแกรมโดยป้อนชื่อ FIXTOFLT ดังนี้

A>FIXTOFLT แล้วกด RETURN

โปรแกรมจะถามว่าเป็นเลขคิวอะไร ผู้ใช้ป้อนเลขคิวแล้วกด RETURN โปรแกรมจะถามเลขเฮ็กซ์ของคิวฟอร์แมทนั้น ผู้ใช้ป้อนเลขฟิกซ์พอยต์ลงไปแล้วกด RETURN โปรแกรมจะแสดงค่าเลขโฟลทในฐานสิบมาให้ทันที



รูปที่ 4.5

4.2.3 โปรแกรม TSIN.EXE

โปรแกรม TSIN.EXE เป็นโปรแกรมที่เขียนขึ้นด้วยภาษาซี บนเทอร์โบซีเวอร์ชั่น

2.0 ซึ่งใช้สำหรับเปรียบเทียบเวลาการทำงานกับ TMS32010 โดยใช้วัดความเร็วในการประมวลผลของตัว TMS32010 กับไอบีเอ็มพีซีทั่วไป โปรแกรม TSIN.EXE ได้เขียนเลียนแบบโปรแกรม GENER1.ASM ของตัว TMS32010 และใช้หน่วยความจำจำนวนหนึ่งของฮีฟ (heap) ในภาษาซี ไว้เก็บตัวแปรค่าซายน์ ดังนั้นก่อนโปรแกรมเริ่มทำงานจึงต้องทำการเช็คดูว่าเนื้อที่ของฮีฟนี้พอหรือไม่ ขั้นตอนต่างๆในการเรียกใช้โปรแกรมมีดังนี้คือ จากดอส ให้ป้อนชื่อ TSIN ลงไปดังนี้

A>TSIN แล้วกด RETURN

โปรแกรมก็จะแสดงเนื้อที่ของฮีฟที่สามารถใช้งานได้เป็นหน่วยไบท์ดังนี้

Memory free available = nn ; nn คือเวลาในหน่วยวินาที

หลังจากแสดงเนื้อที่ของฮีฟแล้วก็จะถามว่าผู้ใช้ต้องการจะรันในกี่สเต็ปดังนี้

Enter munber of step --------> ป้อนค่าจำนวนสเต็ปลงไป

ค่าจำนวนสเต็ปนี้จะนำไปหาค่าจำนวนจุดของค่าซายน์อีกที โดยจำนวนจุดของค่าซายน์ที่จะสร้างขึ้นเท่ากับจำนวนสเต็ปยกกำลังสอง เช่น ถ้าเราป้อนจำนวนสเต็ปไปเท่ากับ 100 ดังนั้นจำนวนจุดของค่าซายน์ที่จะสร้างขึ้นจะเป็น 10,000 จุด หลังจากป้อนค่าจำนวนสเต็ปลงไปแล้ว โปรแกรมจะเริ่มต้นการสร้างค่าซายน์ตากตารางที่ได้กำหนดไว้และตารางที่ได้ทำไว้ก็ได้ทำเป็นตารางเลขฟิกซ์พอยต์เพื่อเลียนแบบโปรแกรมของ TMS32010 ที่ใช้ตัวเลขแบบฟิกซ์พอยต์ ดังนั้นจึงต้องเรียกรูทีนการแปลงเลขฟิกซ์พอยต์นี้ให้เป็นเลขโฟลทรูทีน FIXTOFLT จะใช้ในการแปลงตัวเลข 2 อย่างนี้ เมื่อทำการแปลงเสร็จก็จะใส่ให้กับจำนวนจุดของค่าซายน์ซึ่งก็เป็นตัวแปรในเนื้อที่ของฮีฟ ด้วยเหตุนี้ หากต้องการสร้างจำนวนจุดที่มากก็ต้องมีฮีฟไว้มากๆ ฮีฟนี้จะแปรผันโดยตรงกับหน่วยความจำที่เหลือของเครื่อง เราจึงควรเอาโปรแกรมแฝงอื่นๆเช่นเทอร์โบซี, SIDEKICK ออกให้หมดก่อนจะรันโปรแกรม TSIN โปรแกรมจะพิมพ์ข้อความว่า

Start time to generate sine wave.....

เพื่อให้ผู้ใช้ทราบว่า ขณะนี้ได้เริ่มต้นสร้างจุดของค่าซายน์แล้ว โปรแกรมจะทำงานไปเรื่อยๆ จนกระทั่งครบจำนวนจุดที่ได้กำหนดไว้ โปรแกรมก็จะแสดงข้อความ

Time use for table lookup = ...(เวลาทั้งหมดที่ใช้ในการสร้างจุด)

ตามด้วยข้อความ

Hit any key to view graph (Esc to exit)

เป็นการบอกให้ผู้ใช้ตัดสินใจ ที่จะดูกราฟของตัวแปรซายน์ที่ได้สร้างขึ้นมาในขั้นตอนก่อนนี้ หรือจะหลุดจากโปรแกรมไปโดยการกดคีย์ Esc

ถ้าผู้ใช้ตัดสินใจที่จะดูกราฟ โปรแกรมก็จะไปเรียกรูทีน GENGRAPH ซึ่งจะทำหน้าที่ในการพล็อตกราฟจากตัวแปรซายน์ที่ได้สร้างขึ้น ให้ผู้ใช้เห็นรูปร่างของสัญญาณซายน์นี้ว่าถูกต้องเพียงใด

4.2.4 โปรแกรม TMS.EXE

โปรแกรม TMS.EXE นี้จะทำการรับค่าตัวแปรในหน่วยความจำของ TMS32010 ที่ได้เก็บค่าซายน์ในรูปฟิกซ์พอยต์เอาไว้เป็นเวิร์ดๆ แล้วนำมาทำการแปลงให้อยู่ในรูปโฟลท ก่อนที่จะทำการพล็อตออกจอดังโปแกรม TSIN.EXE

ขั้นตอนการเรียกใช้โปรแกรม TMS.EXE เริ่มจากป้อนชื่อ TMS ในดอส

A>TMS แล้วตามด้วย RETURN

การป้อนชื่อ TMS นี้จะต้องมั่นใจก่อนว่าได้รันโปรแกรมสร้างสัญญาณซายน์ของตัว TMS32010 มาก่อน พูดอีกนัยหนึ่งก็คือจะต้องมีข้อมูลค่าซายน์ที่สร้างโดย TMS32010 อยู่ในหน่วยความจำของตัว TMS32010 ซึ่งไอบีเอ็มพีซีสามารถไปดึงเอาค่าเหล่านี้ มาทำการพล็อต สมมติว่าขณะนี้ได้รันโปรแกรมการสร้างสัญญาณคลื่นรูปซายน์ของตัว TMS32010 ไปแล้ว เราต้องการเช็คดูว่า TMS32010 ทำงานถูกต้องจริงหรือไม่ โดยการนำเอาข้อมูลเหล่านี้มาพล็อตกราฟดู โปรแกรม TMS จะทำหน้าที่อันนี้โดยหลังจากเรียกใช้โปรแกรมนี้ก็จะแสดงฮีฟที่เหลือไว้เก็บค่าซายน์ในรูปของเลขโฟลทดังนี้

Memory Free available = ... เนื้อที่เหลือไว้ใช้งานได้

แล้วตามด้วยการถามจำนวนสเต็ปที่จะให้พล็อตกราฟออกมาบนจอ

Enter number step -----> ป้อนค่าจำนวนสเต็ปลงไป

ทีนี้โปรแกรมจะทำการเช็คดูว่า จำนวนสเต็ปที่ป้อนเข้าไปนี้จะต้องใช้เนื้อที่ฮีฟเท่าใดเพื่อใช้เก็บ เมื่อเทียบกับฮีฟที่สามารถใช้ได้จริง ถ้าพอก็จะทำในขั้นตอนต่อไป แต่หากไม่พอก็จะแสดงข้อความ

Memory not enough.....

แล้วหลุดออกจากการทำงานทั้งหมดเข้าสู่ดอส

ในขั้นตอนต่อไปของการทำงานของโปรแกรมคือ การไปดึงค่าจากหน่วยความจำของตัว TMS32010 มาทีละเวิร์ด แล้วจึงทำการแปลงแต่ละเวิร์ดนี้ให้เป็นเลขโฟลท โดยในตำแหน่งของข้อมูลค่าซายน์ของตัว TMS32101 จะอยู่ที่ตำแหน่ง A100:0000 และตำแหน่งที่จะเก็บค่าของซายน์ที่ได้ทำการแปลงมาจากค่าฟิกซ์พอยต์นี้ก็จะเก็บ ณ ตำแหน่ง 7000:0000 เป็นต้นไป

เมื่อทำการแปลงได้หมดจากจำนวนสเต็ปที่ป้อนในตอนแรกแล้ว ก็จะนำค่าเหล่านี้ไปพล็อตกราฟอีกทีโดยใช้รูทีน GENGRAPH เช่นเดียวกับของโปรแกรม TSIN.EXE ได้ผลลัพธ์เป็นกราฟรูปซายน์บนจอภาพกราฟฟิก

4.2.5 <u>โปรแกรม FLOAT.EXE</u>

โปรแกรมนี้ได้พัฒนาขึ้นมาเพื่อทำหน้าที่แปลงเลขโฟลทติ้งพอยต์ให้เป็นเลขฟิกซ์พอยต์เอาไว้สร้างตารางเลขฟิกซ์พอยต์ที่จะเรียกใช้โดย TMS32010 ขั้นตอนการใช้งานโปรแกรม FLOAT.EXE มีดังนี้

เริ่มจากการป้อนชื่อ FLOAT นี้ลงในดอส

A>FLOAT แล้วกด RETURN

โปรแกรมจะถามเลขรูปแบบคิวว่าจะให้เป็นคิวชนิดใดตั้งแต่ Q0-Q15 ดังนี้

Q(0-15)-----> ผู้ใช้ป้อนเลขรูปแบบคิวอยู่ระหว่าง 0-15

โปรแกรมจะถามเลขโฟลทที่ต้องการแปลงให้เป็นเลขฟิกซ์ตามฟอร์แมทคิวที่กำหนด

DEC -----> ผู้ใช้ป้อนเลขโฟลท(ฐานสิบ)

เมื่อป้อนเสร็จแล้วกด RETURN โปรแกรมก็จะพิมพ์ค่าเลขฟิกซ์พอยต์มาให้ในรูปเลขฐานสิบหก ก็เป็นอันสิ้นสุดโปรแกรม

บทที่ 5

บทวิจารณ์และสรุป

จากการทดลองเขียนโปรแกรมการสร้างสัญญาณคลื่นรูปซายน์ แล้วทดลองจับเวลาเปรียบเทียบกับเครื่องคอมแพคเดสก์โปรโมเดล 386/20 ซึ่งใช้ซีพียูเบอร์ 80386 และโคโปรเซสเซอร์ 80387 ซึ่งได้แสดงผลในตารางที่ 4.1 จะเห็นได้ว่าการทำงานของตัว TMS32010 ในโปรแกรมนี้ใช้เวลาน้อยกว่าเครื่องคอมแพคเดสก์โปรโมเดล 386/20 โดยเฉลี่ยแล้วประมาณ 15 เท่า ในขณะเดียวกันเมื่อเปรียบเทียบกับไอบีเอ็มพีซีเอ็กซ์ทีซึ่งใช้ซีพียูเบอร์ V-20 จากตารางที่ 4.2 จะพบว่าไอบีเอ็มพีซีเอ็กซ์ทีต้องใช้เวลาในการสร้างสัญญาณซายน์นี้มากกว่า TMS32010 ถึงเป็นจำนวนพันกว่าเท่า จากผลทั้งสองนี้จะเห็นได้ว่าตัว TMS32010 มีประสิทธิภาพในการคำนวณสูงกว่า ซึ่งเป็นไปตามความประสงค์ของโครงงานนี้

สำหรับในการนำระบบของโครงงานนี้ไปใช้งานทางด้านประมวลสัญญาณเชิงดิจิตอลนั้น อาจมีความจำเป็นที่จะต้องมีการเพิ่มเติมวงจรการแปลงสัญญาณอนาล็อกเป็นสัญญาณดิจิตอลและวงจรแปลงสัญญาณดิจิตอลเป็นสัญญาณอนาล็อก ในการนำไปใช้งานด้านนี้จะต้องมีความเข้าใจในทฤษฎีของการประมวลสัญญาณเชิงดิจิตอลอยู่บ้าง เพื่อที่จะได้สามารถเขียนโปรแกรมให้ TMS32010 ทำงานได้อย่างมีประสิทธิภาพ

ในโครงงานนี้ผลงานที่ได้ยังไม่สามารถนำไปใช้งานได้อย่างเต็มที่ อันเนื่องมาจาก

1 ทางด้านฮาร์ดแวร์ จะพบว่าระบบที่ออกแบบมานี้ยังไม่สมบูรณ์พอมีข้อจำกัดบางอย่างอยู่ได้แก่หน่วยความจำข้อมูลของ TMS32010 มีขนาดจำกัดเพียง 2 เคเวิร์ด ถ้าต้องการใช้มากกว่านี้ก็จะต้องมีการดัดแปลงวงจรใหม่

2 ทางด้านซอฟท์แวร์ จะต้องเขียนซอฟท์แวร์สำหรับการควบคุมระบบให้กับตัว 8088-slave ซึ่งโปรแกรมนี้ซับซ้อนพอสมควร จะต้องสามารถติดตามการทำงานของ TMS32010 ได้ และสำหรับโปรแกรมที่เป็นภาษาแอสเซมบลีของ TMS32010 นั้นจะต้องมีอัลกอริธึมที่ดีจึงจะสามารถให้การประมวลผลที่มีประสิทธิภาพได้เต็มที่

ที่อยากจะกล่าวถึงผลการทดลองนี้อีกอย่างก็คือ ในเรื่องของความละเอียดของผลลัพธ์ที่ได้มานั้นจะขึ้นอยู่กับรูปแบบการแทนตัวเลข ในภาษาซีใช้การแทนตัวเลขโฟลทติ้งพอยต์ซึ่งมีขนาด 32 บิท แต่ใน TMS32010 จะใช้การแทนตัวเลขแบบคิว 14 ด้วยขนาด 16 บิท ดังนั้นจึงทำให้ TMS32010 มีความละเอียดน้อยกว่าในภาษาซี แต่ TMS32010 จะประหยัดเนื้อที่ในหน่วยความจำกว่าของภาษาซี

สรุปได้ว่าตัว TMS32010 มีความสามารถในการประมวลผลโดยมีความเร็วในการประมวลสูงเมื่อเทียบกับตัวซีพียูที่ใช้ในเครื่องไอบีเอ็ม ดังนั้นโครงงานนี้เป็นวิธีหนึ่งที่จะทำให้ไมโครคอมพิวเตอร์ไอบีเอ็มประมวลผลได้เร็วขึ้นได้

ภาคผนวก

ข้อมูลและรายละเอียดเกี่ยวกับ TMS32010

TMS32010 เป็นดิจิตอลซิกแนลโปรเซสเซอร์(digital signal processor) ตัวแรกในตระกูลของ TMS 320 ซึ่งเป็นผลิตภัณฑ์ของบริษัทเท็กซัสอินสตรูเมนท์ได้ถูกออกแบบเพื่อการคำนวณที่ต้องการความเร็วสูง โดยที่ตัว TMS32010 นี้สามารถนำมาประยุกต์ใช้ในงานด้านอาร์เรย์โปรเซสเซอร์ซึ่งจะทำให้งานคำนวณมีความรวดเร็วมากขึ้น

ตัวอย่างงานที่สามารถประยุกต์นำตัว TMS32010 ไปใช้มีดังต่อไปนี้

ทางด้านซิกแนลโปรเซสซิง(Signal processing)

-การกรองทางดิจิตอล(Digital filtering)

-คอร์รีเลชั่น(Correlation)

-วินโดว์(Windowing)

-ฟาสต์ฟูเรียร์ทรานสฟอร์ม(Fast Fourier transforms)

-การสร้างคลื่นสัญญาณต่างๆ(Waveform generation)

-เรดาร์และโซนาร์โปรเซสซิง(Radar and Sonar processing)

ทางด้านอินสตรูเมนท์(Instrumentation)

-การวิเคราะห์สเปคตรัม(Spectrum analysis)

-เฟสล็อคลูป(Phase-locked loops)

ทางด้านโทรคมนาคม(Telecommunications)

-อแดปทีฟอีควอไลเซอร์(Adaptive equalizers)

-ไทม์เจนเนอร์เรเตอร์(Time generators)

-โมเด็มความเร็วสูง(High-speed modems)

ทางด้านการคำนวณ(Numeric processing)

-การคูณการหารอย่างรวดเร็ว(Fast multiply/divide

-ดับเบิลพรีซิสชั่น โอเปอร์เรชั่น(Double-precision operation)

-การคำนวณฟังก์ชั่นที่ไม่เชิงเส้น(Non-linear function computation)

ทางด้านอิมเมจโปรเซสซิง(Imagae processing)

-แพทเทิร์นเรค็อคไนชั่น(Patterตn recognition)

-อิมเมจเอนฮานซ์เมนท์(Image enhancement)

-อิมเมจคอมเพรสชั่น(Image compression)

ทางด้านระบบควบคุมความเร็ว(High-speed control)

-เซอร์โวลิงค์(Servo links)

-การควบคุมโพสิชั่นและเรท(Position and rate control)

-การควบคุมมอเตอร์(Motor control)

-จรวดนำวิถี(Missile quidance)

-รีโมทฟีดแบ็คคอนโทรล(Remote feedback control)

-โรโบติกส์(Robotics)

ทางด้านกระบวนการเกี่ยวกับเสียง(Speech processing)

-การวิเคราะห์เสียงพูด(Speech analysis)

-การสังเคราะห์เสียง(Speech synthesis)

-การจดจำเสียง(Speech recognition)

ข้อมูลจำเพาะของ TMS32010

- 200-ns instruction cycle
- 288-byte on-chip data RAM
- Microprocessor version - TMS32010
- Microcomputer version - TMS320M10
  (3k byte on-chip program ROM)
- External program memory expansion to a total of
  8k bytes at full speed
- 16-bit instruction/data word
- 32-bit ALU/accumulator
- 16 x 16-bit multiply in 200 ns
- 0 to 15-bit barrel shifter
- Eight input and eight output channels
- 16-bit bidirectional data bus with
  40-megabits-per second transfer rate
- Interrupt with full context save
- Signed two's complement fixed-point arithmetic
- 2.7-micron NMOS technology
- Single 5-v supply
- 40-pin DIP

หมายเหตุ

TMS32010 และ TMS320M10 จะมีลักษณะเหมือนกันทุกอย่าง ยกเว้นที่แตกต่าง คือ TMS320M10 จะมีรอม(ROM)อยู่ภายในตัว แต่ TMS32010ไม่มี

ตารางที่ 1 แสดงสัญญลักษณ์และชื่อของฮาร์ดแวร์ TMS32010

UNIT	SYMBOL	FUNCTION
Accumulator	ACC	32-bit accumulator
Arithmetic Logic Unit	ALU	Two-port 32-bit arithmetic logic unit
Auxiliary Registers	AR0, AR1	Two 16-bit registers for indirect addressing of data memory and loop counting control. Nine LSBs of each register are configured as bidirectional counters
Auxiliary Register Pointer	ARP	Single-bit register containing address of current auxiliary register
Data Bus	D Bus	16-bit bus routing data from random access memory
Data Memory Page Pointer	DP	Single-bit register containing page address of data RAM (1 page = 128 words)
Data RAM	–	144 X 16 bit word on-chip random access memory containing data
Interrupt Flag Register	INTF	Single-bit flag register that indicates an interrupt request has occurred (is pending)
Interrupt Mode Register	INTM	Single-bit mode register that masks the interrupt flag
Multiplier	–	16 X 16-bit parallel hardware multiplier
Overflow Flag Register	OV	Single-bit flag register that indicates an overflow in arithmetic operations
Overflow Mode Register	OVM	Single-bit mode register that defines a saturated or unsaturated mode in arithmetic operations
P Register	P	32-bit register containing product of multiply operations
Program Bus	P Bus	16-bit bus routing instructions from program memory
Program Counter	PC	12-bit register containing address of program memory
Program ROM	–	1536 X 16-bit word read only memory containing program code (TMS320M10 only)
Shifter	–	Two shifters: one is a variable 0-15-bit left-shift barrel shifter that moves data from the RAM into the ALU. The other shifter acts on the accumulator when it is being stored in data RAM; it can left-shift by 0, 1, or 4 bits.
Stack	–	4 X 12-bit registers for saving program counter contents in subroutine and interrupt calls
T Register	T	16-bit register containing multiplicand during multiply operations

โครงสร้างทางสถาปัตยกรรมของ TMS32010

จุดเด่นของ TMS32010 คือใช้โครงสร้างสถาปัตยกรรมแบบฮาร์วาร์ด ซึ่งจะแยกหน่วยความจำออกเป็นหน่วยความจำโปรแกรม และหน่วยความจำข้อมูล หน่วยความจำทั้งสองส่วนนี้จะถูกแยกออกจากกัน ทำให้สามารถเอ็กซีคิวท์(execute) โปรแกรมไปพร้อมๆกับการเฟทช์(fetch) คำสั่งได้

โครงสร้างทางสถาปัตยกรรมของ TMS32010 ประกอบกันเป็นชิพเดียวดังนี้

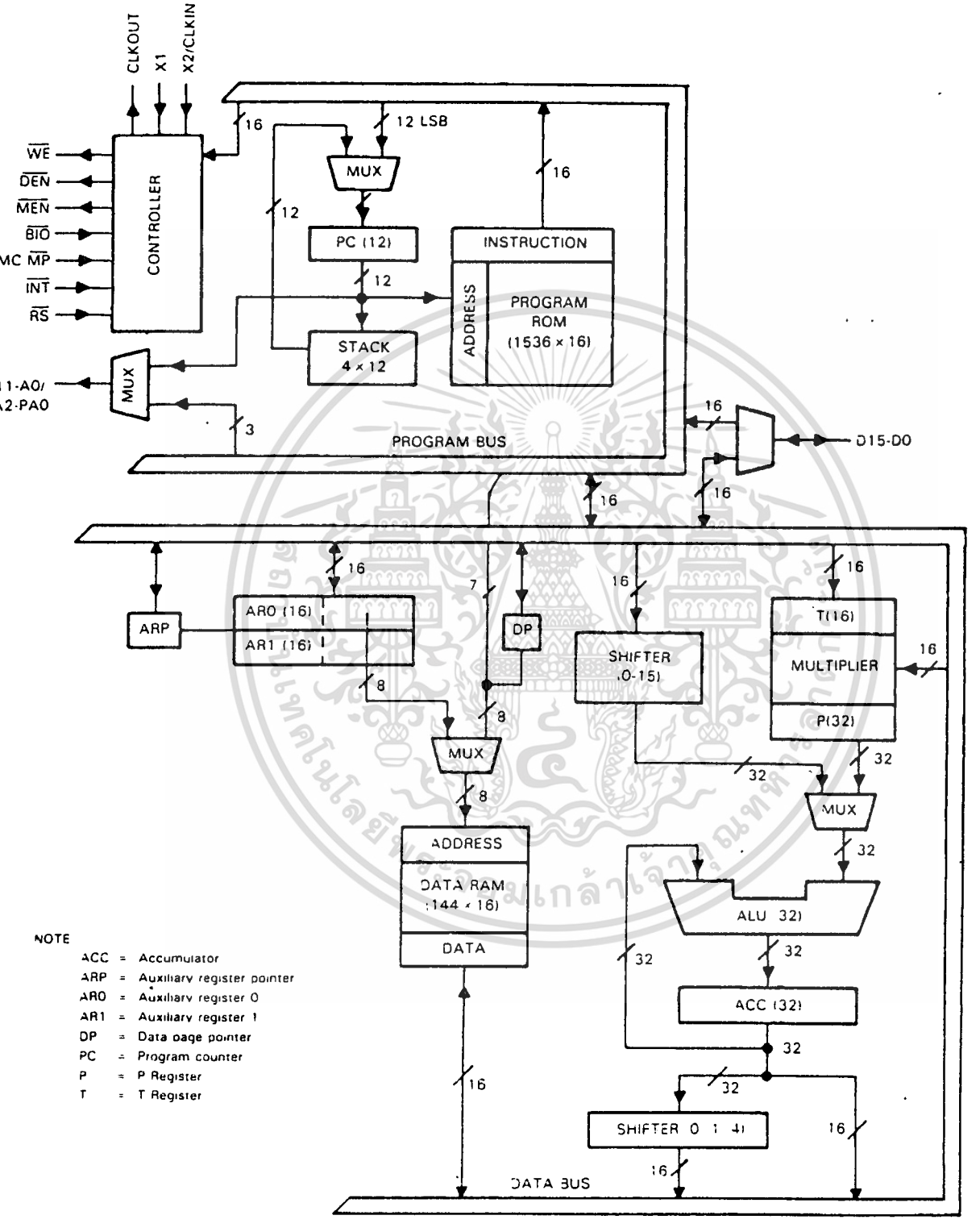
- Volatile 144 x 16-word read/write data memory
- Non-volatile 1536 x 16-word program memory (TMS320M10)
- Double-precision 32-bit ALU/accumulator
- Fast 200-ns multiplier
- Barrel shifter for shifting data memory words into the ALU
- Shifter that shifts the accumulator into the data RAM
- 16-bit data bus for fetching instruction word from off-chip at full speed
- 4 x 12-bit stack that allows context switching
- Autoincrementing/decrementing registers for indirect data addressing and loop counting
- Single-vectored interrupt
- On-chip oscillator

รูปที่ 1 แสดงให้เห็นผังไดอะแกรมของโครงสร้างทางสถาปัตยกรรมของ TMS32010 ซึ่งรายละเอียดจะกล่าวเป็นส่วนๆดังต่อไปนี้

1 หน่วยทางคณิตศาสตร์(ARITHMETIC ELEMENT)

ประกอบด้วยหน่วยพื้นฐาน 4 หน่วยคือ เอแอลยู(ALU), แอคคิวมูเลเตอร์

รูปที่ 1 ผังไดอะแกรมโครงสร้างทางสถาปัตยกรรมของ TMS32010



(accumulator), มัลติพลายเออร์(multiplier), และชิฟเตอร์(shifter)

1.1 เอแอลยู

เป็นเจนเนอร์รัลเพอร์โพสอะริธเมติกยูนิท(general-purpose arithmetic logic unit) หน่วยเอแอลยูนี้จะโอเปอร์เรต(operate) กับข้อมูลขนาด 32 บิท หน่วยนี้จะทำหน้าที่บวก,ลบ,คูณ,หาร และปฏิบัติการแบบลอจิก

1.1.1 โอเวอร์โฟลว์โหมด(Overflow Mode OVM)

รีจิสเตอร์ OVM จะอยู่ภายใต้โปรแกรมควบคุมซึ่งสามารถเซ็ทได้ด้วยคำสั่ง SOVM และรีเซ็ทได้ด้วยคำสั่ง ROVM รีจิสเตอร์นี้สามารถเก็บไว้ในหน่วยความจำข้อมูลในลักษณะเป็นบิทหนึ่งในสเตตัสรีจิสเตอร์(status register)

1.2 แอคคิวมูเลเตอร์(Accumulator)

แอคคิวมูเลเตอร์นี้จะทำหน้าที่เก็บผลลัพธ์ที่ได้จากเอแอลยูและเป็นหน่วยอินพุทให้กับเอแอลยูด้วย แอคคิวมูเลเตอร์นี้โอเปอร์เรตกับข้อมูลขนาด 32 บิท โดยแบ่งเป็นเวิร์ดสูง (high order word (บิทที่ 31-16))และเวิร์ดต่ำ(low order word (บิทที่ 15-0))

1.2.1 แอคคิวมูเลเตอร์สเตตัส(Accumulator Status)

สถานะโอเวอร์โฟลว์ของแอคคิวมูเลเตอร์(Accumulator overflow status) สามารถจะอ่านได้จากแอคคิวมูเลเตอร์โอเวอร์โฟลว์แฟลกรีจิสเตอร์ (overflow flag register OV) รีจิสเตอร์นี้จะถูกเซ็ทถ้าเกิดการโอเวอร์โฟลว์ขึ้นในแอคคิวมูเลเตอร์รีจิสเตอร์ OV นี้เป็นส่วนหนึ่งของสเตตัสรีจิสเตอร์

1.3 มัลติพลายเออร์

หน่วยการคูณ 16x16 บิทแบบขนานนี้ประกอบด้วยทีรีจิสเตอร์(T register),พีรีจิสเตอร์(P register) และมัลติพลายเออร์อาร์เรย์(multiplier array)

-ทีรีจิสเตอร์ มีขนาด 16 บิท ทำหน้าที่เก็บตัวตั้งของการคูณ

-พีรีจิสเตอร์ มีขนาด 32 บิท ทำหน้าที่ในการเก็บผลลัพธ์ของการคูณ

1.4 ชิฟท์เตอร์

ในการชิฟท์ข้อมูลมีอยู่ 2 แบบ คือ

1.4.1 บาร์เรลชิฟท์เตอร์(barrel shifter) สำหรับการชิฟท์ข้อมูลจากหน่วยความจำข้อมูลไปสู่เอแอลยู

1.4.2 พาราเลลชิฟท์เตอร์(parallel shifter) สำหรับการชิฟท์ข้อมูลจากแอคคิวมูเลเตอร์เข้าสู่หน่วยความจำข้อมูล

2 หน่วยความจำข้อมูล(DATA MEMORY)

ในส่วนที่เป็นหน่วยความจำข้อมูลนั้นมีขนาด 144x16 บิท เป็นหน่วยความจำที่อยู่ภายในตัว TMS32010 การเคลื่อนย้ายข้อมูลระหว่างหน่วยความจำโปรแกรมกับหน่วยความจำข้อมูลทำได้โดยการใช้คำสั่ง TBLR และ TBLW

3 การอ้างหน่วยความจำข้อมูล(data memory addressing)

ในการอ้างหน่วยความจำข้อมูลทำได้ 3 วิธีคือ

3.1 การอ้างแอดเดรสทางอ้อม(Indirect Addressing)

การอ้างตำแหน่งของหน่วยความจำข้อมูลโดยวิธีนี้ ทำได้โดยการใช้ 8 บิทล่างของรีจิสเตอร์สำรอง(auxiliary register) เป็นตัวชี้ตำแหน่งแอดเดรสของหน่วยความจำข้อมูล ซึ่งจะสามารถทำการอ้างแอดเดรสได้ถึง 144 ดาต้าเวิร์ด

3.2 การอ้างแอดเดรสโดยตรง(Direct Addressing)

ในการอ้างแอดเดรสแบบนี้ 7 บิทของเวิร์ดคำสั่งจะเป็นตัวชี้ตำแหน่งของแอดเดรสหน่วยความจำข้อมูลร่วมกับดาต้าเพจพอยท์เตอร์ (data page pointer DP) ซึ่งจะมีลักษณะการอ้างดังนี้

DP	ตำแหน่งของหน่วยความจำ
0	0 - 127
1	128 - 144

ส่วนใหญ่แล้วเพจที่สองจะใช้สำหรับตัวแปรของระบบเช่น ถูกใช้สำหรับรูทีนของอินเตอร์รัพท์(interrupt routine) DP เป็นส่วนหนึ่งของสเตตัสรีจิสเตอร์และสามารถนำไปเก็บไว้ในหน่วยความจำข้อมูลได้

3.3 การอ้างแอดเดรสแบบอิมมิเดียท(Immediate Addressing)

ในคำสั่งของ TMs32010มีคำสั่งพิเศษที่สามารถใช้ในการอ้างแอดเดรสได้โดยทันที

4 รีจิสเตอร์

4.1 รีจิสเตอร์สำรอง(Auxiliary Registers)

ภายในตัว TMS32010 มีรีจิสเตอร์ที่เป็นฮาร์ดแวร์และมีขนาด 16 บิทอยู่ 2 ตัว คือ AR0 และ AR1 ทั้งสองตัวนี้ถูกใช้สำหรับเก็บข้อมูลชั่วคราว, การอ้างแอดเดรสทางอ้อมของหน่วยความจำข้อมูล, และการควบคุมลูป

4.2 ตัวชี้รีจิสเตอร์สำรอง(Auxiliary Register Pointer ARP)

เป็นตัวชี้ว่า รีจิสเตอร์ตัวใดถูกใช้งาน มีขนาด 1 บิท โดยเป็นส่วนหนึ่งของสเตตัสรีจิสเตอร์ ลักษณะการใช้งานมีดังนี้

ARP	รีจิสเตอร์ตัวที่ถูกใช้งาน
0	AR0
1	AR1

5 หน่วยความจำโปรแกรม(PROGRAM MEMORY)

หน่วยความจำโปรแกรมมีขนาด 4K x 16 บิท โดยแบ่งเป็น 1536x16 บิท อยู่ภายในชิพ(เฉพาะ TMS320M10 เท่านั้น) สำหรับ TMS32010 จะไม่มีหน่วยความจำโปรแกรมที่อยู่ภายในชิพเลย ดังนั้น TMS32010 จึงมีแต่หน่วยความจำโปรแกรมทั้งหมด 4Kx16 บิทอยู่ภายนอก

5.1 โหมดของการโอเปอร์เรต

การทำงานของ TMS32010 แบ่งได้เป็น 2 โหมดซึ่งขึ้นอยู่กับสถานะของขา MC/MPถ้าขานี้มีสถานะเป็น 1 ก็เป็นโหมดไมโครคอมพิวเตอร์(microcomputer mode) และถ้าขานี้สถานะเป็น 0 ก็จะเป็นโหมดไมโครโปรเซสเซอร์(microprocessor mode)

-โหมดไมโครคอมพิวเตอร์ใช้ได้เฉพาะกับ TMS320M10 เท่านั้น

-โหมดไมโครโปรเซสเซอร์ใช้ได้ทั้ง TMS320M10 และ TMS32010

6 โปรแกรมเคาน์เตอร์(PROGRAM COUNTER PC)

เป็นรีจิสเตอร์ขนาด 12 บิท ใช้สำหรับเก็บค่าแอดเดรสของหน่วยความจำโปรแกรมที่จะเอ็กซิคิวท์ถัดไป PC จะถูกเซ็ทให้เป็น 0 เมื่อ TMS32010 ถูกรีเซ็ท เพื่อที่จะสามารถใช้หน่วยความจำภายนอกได้ เอาท์พุทของ PC ซึ่งจะไปปรากฏที่แอดเดรสบัส A0 ถึง A11 จะต้องทำการต่อผ่านบัฟเฟอร์เสียก่อน สำหรับการอ้างอินพุทเอาท์พุทพอร์ทจะใช้แอดเดรสบัส A0 ถึง A2

7 สแตก(STACK)

เป็นรีจิสเตอร์ขนาด 12 บิท และมีอยู่ 4 ชั้น มีหน้าที่เก็บค่าของโปรแกรมเคาน์เตอร์ เมื่อมีการเรียกใช้ซับรูทีน(subroutine)หรือการอินเตอร์รัพท์ นอกจากนี้เมื่อมีการใช้คำสั่ง PUSH สแตกก็จะทำหน้าที่เก็บค่าของแอคคิวมูเลเตอร์ 12 บิทล่างไว้ที่ส่วนบนสุดของสแตก

8 สแตตัสรีจิสเตอร์(STATUS REGISTER)

ประกอบด้วย 5 สเตตัสบิท โดยที่แต่ละบิทสามารถเปลี่ยนแปลงได้เป็นอิสระ นอกจากนี้สเตตัสรีจิสเตอร์ยังสามารถเก็บไว้ในหน่วยความจำข้อมูลโดยคำสั่ง SST หรือ โหลดข้อมูลจากหน่วยความจำข้อมูลเข้าสู่สเตตัสรีจิสเตอร ได้โดยคำสั่ง LST

สเตตัส รีจิสเตอร์ ประกอบด้วย

-Accumulator Overflow Flag Register (OV) ถ้าบิทนี้เป็น 1 แสดงว่า

แอคคิวมูเลเตอร์เกิดการโอเวอร์โฟลว์เป็น 0 แสดงว่าไม่เกิดการโอเวอร์โฟลว์

-Interrupt Mask Bit (INTM) ถ้าบิทนี้เป็น 0 มีความหมายว่าการอินเตอร์รัพท์ถูกเอนาเบิลโดยคำสั่ง EINT แต่ถ้าเป็น 1 แสดงว่าเป็นการดิสเอเบิลอินเตอร์รัพท์(disable interrupt) ซึ่งสามารเซ็ทด้วยคำสั่ง DINT

-Auxiliary Register Pointer (ARP) บิทนี้มีค่า 0 แสดงว่าเลือกใช้รีจิสเตอร์สำรอง AR0 แต่ถ้าบิทนี้มีค่าเป็น 1 ก็แสดงว่าเลือกใช้รีจิสเตอร์สำรอง AR1 สถานะใน ARP สามารถเปลี่ยนแปลงได้โดยคำสั่ง MAR หรือ LARP

-Data Memory Page Pointer (DP) บิทนี้แสดงการเลือกเพจของหน่วยความจำข้อมูล เพจละ 128 เวิร์ด ถ้าบิทนี้เป็น 0 หมายถึงการเลือกเข้าหน่วยความจำข้อมูล 128 เวิร์ดแรก ถ้าบิทนี้เป็น 1 เป็นการเลือกเข้าหน่วยความจำข้อมูล 16 เวิร์ดหลัง ค่า DP นี้สามารถเปลี่ยนแปลงได้โดยคำสั่ง LDP หรือ LDPK

8 อินพุท-เอาท์พุท(INPUT/OUTPUT)

TMS32010 สามารถทำการติดต่อกับอุปกรณ์ภายนอกได้โดยการอ้างตำแหน่งจากแอดเดรสบัส A0-A2 นั่นคือจะอ้างพอร์ทขนาด 16 บิทได้จำนวน 8 พอร์ท การแยกสัญญาณ IN-OUT อาศัยสัญญาณ DEN และ WE

9 การเขียนและอ่านข้อมูลระหว่างหน่วยความจำข้อมูลและหน่วยความจำโปรแกรม

ด้วยคำสั่ง TBLR และ TBLW ทำให้ผู้ใช้สามารถทำการโอนย้ายข้อมูลระหว่างหน่วยความจำโปรแกรมและหน่วยความจำข้อมูลได้

คำสั่ง TBLR จะทำให้สัญญาณ MEN มีค่าเป็น 0 และทำการอ่านข้อมูลจากหน่วยความจำโปรแกรมเข้าสู่หน่วยความจำข้อมูล

คำสั่ง TBLW ทำให้สัญญาณ WE มีสถานะเป็น 0 และ TMS32010 ก็จะทำการเขียนข้อมูลจากหน่วยความจำข้อมูลเข้าสู่หน่วยความจำโปรแกรม

10 อินเตอร์รัพท์(INTERRUPT)

TMS32010 มีการอินเตอร์รัพท์แบบซิงเกิลเลเวลเวคเตอร์(single level vector) โดยใช้สัญญาณขอบขาลง(negative edge) ที่ขา INT ก็จะเกิดการอินเตอร์รัพท์ ขึ้น และอินเตอร์รัพท์แฟลก(interrupt flag) จะถูกเซ็ท

11 รีเซ็ท(RESET)

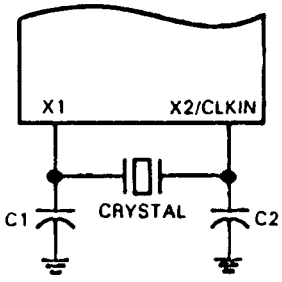
ฟังก์ชั่นของการรีเซ็ท(reset function) จะถูกเอนาเบิลเมื่อมีสัญญาณสถานะ 0 ที่ขา RS อย่างน้อย 5 ไซเคิล(cycle) ของสัญญาณนาฬิกา มีผลให้สัญญาณควบคุม DEN, WE, MEN อยู่ในสถานะ 1, ดาต้าบัสไตรสเตท(tri-state), แอดเดรสบัส(A11ถึงA0) มีสถานะเป็น 0

12 สัญญาณนาฬิกาและตัวกำเนิดสัญญาณนาฬิกา(CLOCK/OSCILLATOR)

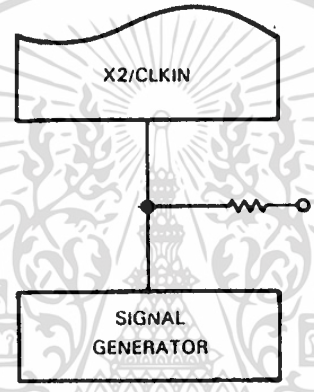
TMS32010 สามารถสร้างสัญญาณนาฬิกาได้ทั้งโดยการใช้ตัวกำเนิดสัญญาณนาฬิกาภายใน(internal oscillator) หรือแหล่งกำเนิดความถี่ภายนอกก็ได้(external frequency source)

ถ้าใช้ตัวกำเนิดสัญญาณนาฬิกาภายในก็นำคริสตอลมาต่อคร่อมขา X1 และ X2 TMS32010 จะสร้างสัญญาณ CLKOUT ที่มีความถี่ขนาด 1/4 เท่าของความถี่ของคริสตอล ดังรูปที่ 2

ถ้าใช้แหล่งกำเนิดความถี่ภายนอก ก็นำแหล่งกำเนิดความถี่มาป้อนเข้าที่ขา X2/CLKIN ตรงๆโดยที่ขา X1 ปล่อยลอยเอาไว้และทำการพูลอัพตัวต้านทานกับไฟเลี้ยงดังรูปที่ 3 เนื่องมาจากสัญญาณที่เข้า CLKIN จะต้องมีระดับโวลเตจ(voltage) อย่างน้อย 2.8 v โดยที่ขนาดของตัวต้านทานนี้ จะขึ้นอยู่กับว่า แหล่งกำเนิดความถี่มีระดับโวลเตจ, กระแส และจำนวนอุปกรณ์อื่นที่แหล่งกำเนิดความถี่นั้นขับ ตัวต้านทานจะต้องให้มีขนาดใหญ่ที่สุดเท่าที่จะใหญ่ได้



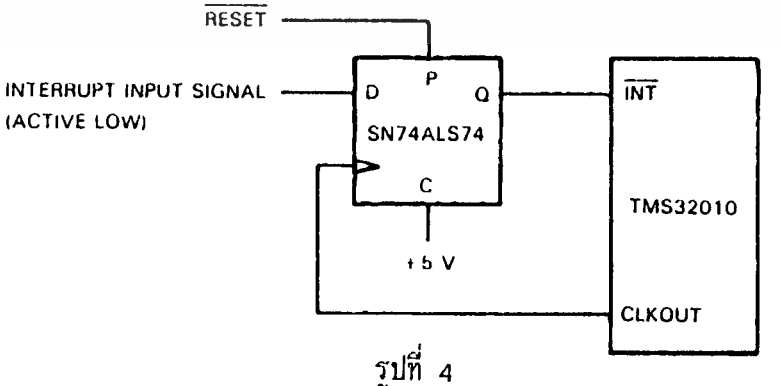
รูปที่ 2



รูปที่ 3

13 <u>การออกแบบระบบการอินเตอร์รัพท์(interrupt sysytem design)</u>

สำหรับระบบที่ใช้อะซิงโครนัสอินเตอร์รัพท์(asynchronous interrupts)บน TMS32010 มีลักษณะดังรูปที่ 4 ในการที่จะให้การอินเตอร์รัพท์ทำงานได้ถูกต้องมีสัญญาณอินพุทที่ซิงโครไนซ์ฮาร์ดแวร์ด้วยสัญญาณขอบขาขึ้นของสัญญาณ CLKOUT บน TMS32010



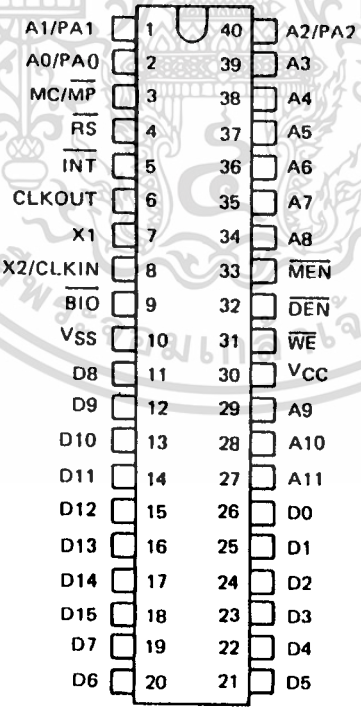
รูปที่ 4

TMS32010 PIN DESCRIPTIONS

SIGNAL	PIN	I/O	DESCRIPTION
			POWER SUPPLIES
V_{CC}	30		Supply voltage (+ 5 V NOM)
V_{SS}	10		Ground reference
			CLOCKS
X2/CLKIN	8	IN	Crystal input pin for internal oscillator (X2). Also input pin for external oscillator (CLKIN).
X1	7	OUT	Crystal input pin for internal oscillator
CLKOUT	6	OUT	Clock output signal. The frequency of CLKOUT is one-fourth of the oscillator input (external oscillator) or crystal frequency (internal oscillator). Duty cycle is 50 percent.
			CONTROL
$\overline{WE}$	31	OUT	Write Enable. When active (low), $\overline{WE}$ indicates that valid output data from the TMS32010 is available on the data bus. $\overline{WE}$ is only active during the first cycle of the OUT instruction and the second cycle of the TBLW instruction (see Section 3.4.3). $\overline{MEN}$ and $\overline{DEN}$ will always be inactive (high) when $\overline{WE}$ is active.
$\overline{DEN}$	32	OUT	Data Enable. When active (low), $\overline{DEN}$ indicates that the TMS32010 is accepting data from the data bus. $\overline{DEN}$ is only active during the first cycle of the IN instruction (see Section 3.4.3). $\overline{MEN}$ and $\overline{WE}$ will always be inactive (high) when $\overline{DEN}$ is active.
$\overline{MEN}$	33	OUT	Memory Enable. $\overline{MEN}$ will be active low on every machine cycle except when $\overline{WE}$ and $\overline{DEN}$ are active. $\overline{MEN}$ is a control signal generated by the TMS32010 to enable instruction fetches from program memory. $\overline{MEN}$ will be active on instructions fetched from both internal and external memory.

SIGNAL	PIN	I/O	DESCRIPTION
			INTERRUPTS
$\overline{RS}$	4	IN	Reset. When an active low is placed on the $\overline{RS}$ pin for a minimum of five clock cycles, $\overline{DEN}$, $\overline{WE}$, and $\overline{MEN}$ are forced high, and the data bus (D15 through D0) is tristated. The program counter (PC) and the address bus (A11 through A0) are then synchronously cleared after the next complete clock cycle from the falling edge of $\overline{RS}$. $\overline{RS}$ also disables the interrupt, clears the interrupt flag register, and leaves the overflow mode register unchanged. The TMS32010 can be held in the reset state indefinitely.
INT	5	IN	Interrupt. The interrupt signal is generated by applying a negative-going edge to the $\overline{INT}$ pin. The edge is used to latch the interrupt flag register (INTF) until an interrupt is granted by the device. An active low level will also be sensed. (See Section 2.10.)
BIO	9	IN	I/O Branch Control. If $\overline{BIO}$ is active (low) upon execution of the BIOZ instruction, the device will branch to the address specified by the instruction (see Section 2.9).
			PROGRAM MEMORY MODES
MC/$\overline{MP}$	3	IN	Microcomputer/Microprocessor Mode. A high on the MC/$\overline{MP}$ pin enables the microcomputer mode. In this mode, the user has available 1524 words of on-chip program memory. (Program memory locations 1524 through 1535 are reserved.) The microcomputer mode also allows an additional 2560 words of program memory to reside off-chip. A low on the MC/$\overline{MP}$ pin enables the microprocessor mode. In this mode, the entire memory space is external, i.e., addresses 0 through 4095. (See Section 2.3.1.)
			BIDIRECTIONAL DATA BUS
D15	18	I/O	D15 (MSB) through D0 (LSB). The data bus is always in the high-impedance state except when $\overline{WE}$ is active (low).
D14	17	I/O	
D13	16	I/O	
D12	15	I/O	
D11	14	I/O	
D10	13	I/O	
D9	12	I/O	
D8	11	I/O	
D7	19	I/O	
D6	20	I/O	
D5	21	I/O	
D4	22	I/O	
D3	23	I/O	
D2	24	I/O	
D1	25	I/O	
D0	26	I/O	

SIGNAL	PIN	I/O	DESCRIPTION
			PROGRAM MEMORY ADDRESS BUS AND PORT ADDRESS BUS
A11	27	OUT	Program memory A11 (MSB) through A0 (LSB) and port addresses PA2 (MSB) through PA0 (LSB). Addresses A11 through A0 are always active and never go to high impedance. During execution of the IN and OUT instructions, pins A2 through A0 carry the port addresses PA2 through PA0.
A10	28	OUT	
A9	29	OUT	
A8	34	OUT	
A7	35	OUT	
A6	36	OUT	
A5	37	OUT	
A4	38	OUT	
A3	39	OUT	
A2/PA2	40	OUT	
A1/PA1	1	OUT	
A0/PA0	2	OUT	



คำสั่งของ TMS32010 (INSTRUCTIONS OF TMS32010)

TMS32010 มีชุดคำสั่งที่ทรงประสิทธิภาพเพื่อรองรับทั้งงานในด้านการคำนวณเชิงตัวเลขและการประมวลสัญญาณรวมทั้งการใช้งานทั่วๆไป ชุดคำสั่งของ TMS32010 มีแสดงในตารางที่ 2 ประกอบด้วยคำสั่งที่มีความยาวหนึ่งเวิร์ด และรอบการทำงานภายในหนึ่งไซเคิล มีอัตราการเอ็กซีคิวต์ห้าล้านคำสั่งต่อหนึ่งวินาที จะมีเพียงคำสั่งเกี่ยวกับการกระโดด (branch) และคำสั่งเกี่ยวกับอินพุทเอาท์พุทเท่านั้นที่จะต้องใช้เวลาหลายไซเคิล

ชุดคำสั่งประกอบไปด้วยชุดคำสั่งของการกระโดด(branch instruction set) โดยทำงานร่วมกับบูลลีนโอเปอร์ชั่น(Boolean operation) และคำสั่งเกี่ยวกับการชิฟท์ ซึ่งคำสั่งเหล่านี้สามารถดำเนินการจัดการเกี่ยวกับบิทและทำการทดสอบบิทได้ มีฮาร์ดแวร์สำหรับการคูณโดยเฉพาะเมื่อใช้คำสั่ง MPY จะใช้เวลาในการเอ็กซีคิวต์เพียงหนึ่งไซเคิลเท่านั้น คำสั่ง SUBC สามารถทำการชิฟท์และการกระโดดอย่างมีเงื่อนไขซึ่งจำเป็นสำหรับการหารอย่างรวดเร็ว

สำหรับคำสั่งพิเศษ TBLR (เทเบิลรีด table read), TBLW (เทเบิลไรท์ table write) มีไว้เพื่อทำการติดต่อข้อมูลระหว่างหน่วยความจำโปรแกรมและหน่วยความจำข้อมูล คำสั่ง TBLR ใช้สำหรับการย้ายข้อมูลขนาด 16 บิทจากหน่วยความจำโปรแกรมเข้าสู่หน่วยความจำข้อมูล ในทางกลับกัน คำสั่ง TBLW ใช้สำหรับการย้ายข้อมูลจากหน่วยความจำข้อมูลเข้าสู่หน่วยความจำโปรแกรม สองคำสั่งนี้เป็นการขยายจำนวนหน่วยความจำที่ใช้สำหรับเก็บข้อมูล

โหมดของการแอดเดรส(ADDRESSING MODE)

ชุดคำสั่งในการเลือกโหมดของการแอดเดรสมีลักษณะดังนี้

1 Direct Addressing Mode

ในโหมดนี้ 7 บิทของเวิร์ดคำสั่งจะอิงกับดาต้าเพจพอยต์เตอร์สำหรับการชี้ตำแหน่งของหน่วยความจำข้อมูล โดยวิธีนี้จะชี้เพจแรกได้ 128 เวิร์ด และเพจที่ 2 ชี้ได้ 16 เวิร์ด

2 Indirect Addressing Mode

ในโหมดนี้จะชี้ตำแหน่งของหน่วยความจำข้อมูลจาก 8 บิทล่างในรีจิสเตอร์สำรอง

ตัวใดตัวหนึ่ง ARO หรือ AR1 ตัวชี้รีจิสเตอร์สำรอง ARP จะเป็นตัวเลือกว่าใช้รีจิสเตอร์ตัวใด รีจิสเตอร์สำรองสามารถที่จะเพิ่มหรือลดโดยอัตโนมัติหลังจากการเอ็กซีคิวต์คำสั่งอินไดเรกท์ ซึ่งจะมีประโยชน์ในการอ้างตำแหน่งข้อมูลได้ในไซเคิลเดียวเท่านั้น

3 Immediate Addressing Mode

ชุดคำสั่งของ TMS32010 มีคำสั่งพิเศษที่เป็นคำสั่งอิมมิเดียท(immediate instruction) คำสั่งเหล่านี้จะใช้ข้อมูลจากเวิร์ดคำสั่งไม่ใช่จากหน่วยความจำโปรแกรม คำสั่งอิมมิเดียทเหล่านี้ได้แก่ คำสั่งคูแแบบอิมมิเดียท(MPYK), คำสั่งโหลดแอคคิวมูเลเตอร์แบบอิมมิเดียท(LACK), และคำสั่งโหลดรีจิสเตอร์สำรองแบบอิมมิเดียท(LARK)

ฟอร์แมตของคำสั่งการแอดเดรสซิง(INSTRUCTION ADDRESSING FORMAT)

1 ฟอร์แมตของการอ้างแอดเดรสแบบไดเรกต์

(Direct Addressing Format)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OPCODE								0	dma						

บิทที่ 7 = 0 เป็นการกำหนดว่าเป็นโหมดของการอ้างแอดเดรสแบบไดเรกต์ ออปโค้ด(opcode)เก็บไว้ที่บิทที่ 8 ถึงบิทที่ 15 ส่วนบิทที่ 0 - 6 จะเก็บตำแหน่งแอดเดรสของหน่วยความจำข้อมูล 7 บิทนี้จะทำการอ้างแอดเดรสของหน่วยความจำได้ 128 เวิร์ด (1เพจ) โดยการใช้ตัวชี้เพจ(DP)จะสามารถชี้ได้ทั้งหมด 144 เวิร์ด

2 ฟอร์แมตของการอ้างแอดเดรสแบบอินไดเรกต์

(Indirect Addressing Format)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OPCODE								1	0	INC	DEC	ARP	0	0	ARP

บิทที่ 7 = 1 เป็นการกำหนดว่าอยู่ในโหมดของการอ้างแอดเดรสแบบอินไดเรกต์ ออ๊ปโค้ดเก็บไว้ที่บิทที่ 8 - 15 และบิทที่ 0 - 3 จะเป็นบิทควบคุมการอ้างแอดเดรสแบบอินไดเรกต์

จากรูป บิทที่ 3 และบิทที่ 0 ควบคุมตัวชี้รีจิสเตอร์ ARP ถ้าบิทที่ 3 = 0 แล้วค่าที่บิท 0 จะถูกโหลดเข้าสู่ ARP หลังจากการเอ็กซีคิวต์ปัจจุบัน ถ้าบิท 3 = 1 แล้วค่าที่อยู่ใน ARP จะไม่เปลี่ยนถ้า ARP = 0 แสดงว่าใช้ค่าใน AR0 เป็นตัวชี้แอดเดรสของหน่วยความจำข้อมูล ถ้า ARP = 1 ก็เป็นการกำหนดว่าให้ AR1 เป็นตัวชี้แอดเดรสของหน่วยความจำข้อมูล

บิท 5 และบิท 4 ทำการควบคุมตัวรีจิสเตอร์สำรอง ถ้าบิท 5 = 1 แล้วตัวรีจิสเตอร์ที่ถูกกำหนดโดย ARP จะเพิ่มค่าครั้งละ 1 ภายหลังจากเอ็กซีคิวต์คำสั่งปัจจุบัน ถ้าบิทที่ 4 = 1 แล้วตัวรีจิสเตอร์สำรองที่ถูกกำหนดโดย ARP จะลดค่าครั้งละ 1 หลังจากเอ็กซีคิวต์คำสั่งปัจจุบัน ถ้าทั้งบิท 4 และ 5 เป็น 0 แล้วตัวรีจิสเตอร์สำรองจะไม่ลดและไม่เพิ่ม ส่วนบิทที่ 6,2,1 จะสงวนไว้และต้องกำหนดเป็น 0 เสมอ

3 ฟอร์แมตของการอ้างแอดเดรสแบบอิมมิเดียท

(Immediate Addressing Format)

รวมอยู่ในชุดคำสั่งของ TMS32010 ใน 5 คำสั่งที่มีอิมมิเดียทโอเปอร์แรนด์ (LDPK,LARK,MPYK,and LARP)

ACCUMULATOR INSTRUCTIONS				
MNEMONIC	DESCRIPTION	NO. CYCLES	NO. WORDS	OPCODE INSTRUCTION REGISTER 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
ABS	Absolute value of accumulator	1	1	0 1 1 1 1 1 1 1 1 0 0 0 1 0 0 0
ADD	Add to accumulator with shift	1	1	0 0 0 0 ← S → I ← D →
ADDH	Add to high-order accumulator bits	1	1	0 1 1 0 0 0 0 0 I ← D →
ADDS	Add to accumulator with no sign extension	1	1	0 1 1 0 0 0 0 1 I ← D →
AND	AND with accumulator	1	1	0 1 1 1 1 0 0 1 I ← D →
LAC	Load accumulator with shift	1	1	0 0 1 0 ← S → I ← D →
LACK	Load accumulator immediate	1	1	0 1 1 1 1 1 1 0 ← K →
OR	OR with accumulator	1	1	0 1 1 1 1 0 1 0 I ← D →
SACH	Store high-order accumulator bits with shift	1	1	0 1 0 1 1 ← X → I ← D →
SACL	Store low-order accumulator bits	1	1	0 1 0 1 0 0 0 0 I ← D →
SUB	Subtract from accumulator with shift	1	1	0 0 0 1 ← S → I ← D →
SUBC	Conditional subtract (for divide)	1	1	0 1 1 0 0 1 0 0 I ← D →
SUBH	Subtract from high-order accumulator bits	1	1	0 1 1 0 0 0 1 0 I ← D →
SUBS	Subtract from accumulator with no sign extension	1	1	0 1 1 0 0 0 1 1 I ← D →
XOR	Exclusive OR with accumulator	1	1	0 1 1 1 1 0 0 0 I ← D →
ZAC	Zero accumulator	1	1	0 1 1 1 1 1 1 1 1 0 0 0 1 0 0 1
ZALH	Zero accumulator and load high-order bits	1	1	0 1 1 0 0 1 0 1 I ← D →
ZALS	Zero accumulator and load low-order bits with no sign extension	1	1	0 1 1 0 0 1 1 0 I ← D →

AUXILIARY REGISTER AND DATA PAGE POINTER INSTRUCTIONS

MNEMONIC	DESCRIPTION	NO CYCLES	NO WORDS	OPCODE INSTRUCTION REGISTER 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
LAR	Load auxiliary register	1	1	0 0 1 1 1 0 0 R 1 ← D →
LARK	Load auxiliary register immediate	1	1	0 1 1 1 0 0 0 R ← K →
LARP	Load auxiliary register pointer immediate	1	1	0 1 1 0 1 0 0 0 1 0 0 0 0 0 0 K
LDP	Load data memory page pointer	1	1	0 1 1 0 1 1 1 1 1 ← D →
LDPK	Load data memory page pointer immediate	1	1	0 1 1 0 1 1 1 0 0 0 0 0 0 0 0 K
MAR	Modify auxiliary register and pointer	1	1	0 1 1 0 1 0 0 0 1 ← D →
SAR	Store auxiliary register	1	1	0 0 1 1 0 0 0 R 1 ← D →

BRANCH INSTRUCTIONS

MNEMONIC	DESCRIPTION	NO. CYCLES	NO. WORDS	OPCODE INSTRUCTION REGISTER 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
B	Branch unconditionally	2	2	1 1 1 1 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 ← BRANCH ADDRESS →
BANZ	Branch on auxiliary register not zero	2	2	1 1 1 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ← BRANCH ADDRESS →
BGEZ	Branch if accumulator ≥ 0	2	2	1 1 1 1 1 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 ← BRANCH ADDRESS →
BGZ	Branch if accumulator > 0	2	2	1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ← BRANCH ADDRESS →
BIOZ	Branch on $\overline{BIO}$ = 0	2	2	1 1 1 1 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 ← BRANCH ADDRESS →
BLEZ	Branch if accumulator ≤ 0	2	2	1 1 1 1 1 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 ← BRANCH ADDRESS →
BLZ	Branch if accumulator < 0	2	2	1 1 1 1 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 ← BRANCH ADDRESS →
BNZ	Branch if accumulator ≠ 0	2	2	1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 ← BRANCH ADDRESS →
BV	Branch on overflow	2	2	1 1 1 1 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 ← BRANCH ADDRESS →
BZ	Branch if accumulator = 0	2	2	1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 ← BRANCH ADDRESS →
CALA	Call subroutine from accumulator	2	1	0 1 1 1 1 1 1 1 1 0 0 0 1 1 0 0
CALL	Call subroutine immediately	2	2	1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ← BRANCH ADDRESS →
RET	Return from subroutine	2	1	0 1 1 1 1 1 1 1 1 0 0 0 1 1 0 1

T REGISTER, P REGISTER, AND MULTIPLY INSTRUCTIONS

MNEMONIC	DESCRIPTION	NO. CYCLES	NO. WORDS	OPCODE INSTRUCTION REGISTER 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
APAC	Add P register to accumulator	1	1	0 1 1 1 1 1 1 1 1 0 0 0 1 1 1 1
LT	Load T register	1	1	0 1 1 0 1 0 1 0 1 ← D →
LTA	LTA combines LT and APAC into one instruction	1	1	0 1 1 0 1 1 0 0 1 ← D →
LTD	LTD combines LT, APAC, and DMOV into one instruction	1	1	0 1 1 0 1 0 1 1 1 ← D →
MPY	Multiply with T register; store product in P register	1	1	0 1 1 0 1 1 0 1 1 ← D →
MPYK	Multiply T register with immediate operand; store product in P register	1	1	1 0 0 ← K →
PAC	Load accumulator from P register	1	1	0 1 1 1 1 1 1 1 1 0 0 0 1 1 1 0
SPAC	Subtract P register from accumulator	1	1	0 1 1 1 1 1 1 1 1 0 0 1 0 0 0 0

CONTROL INSTRUCTIONS

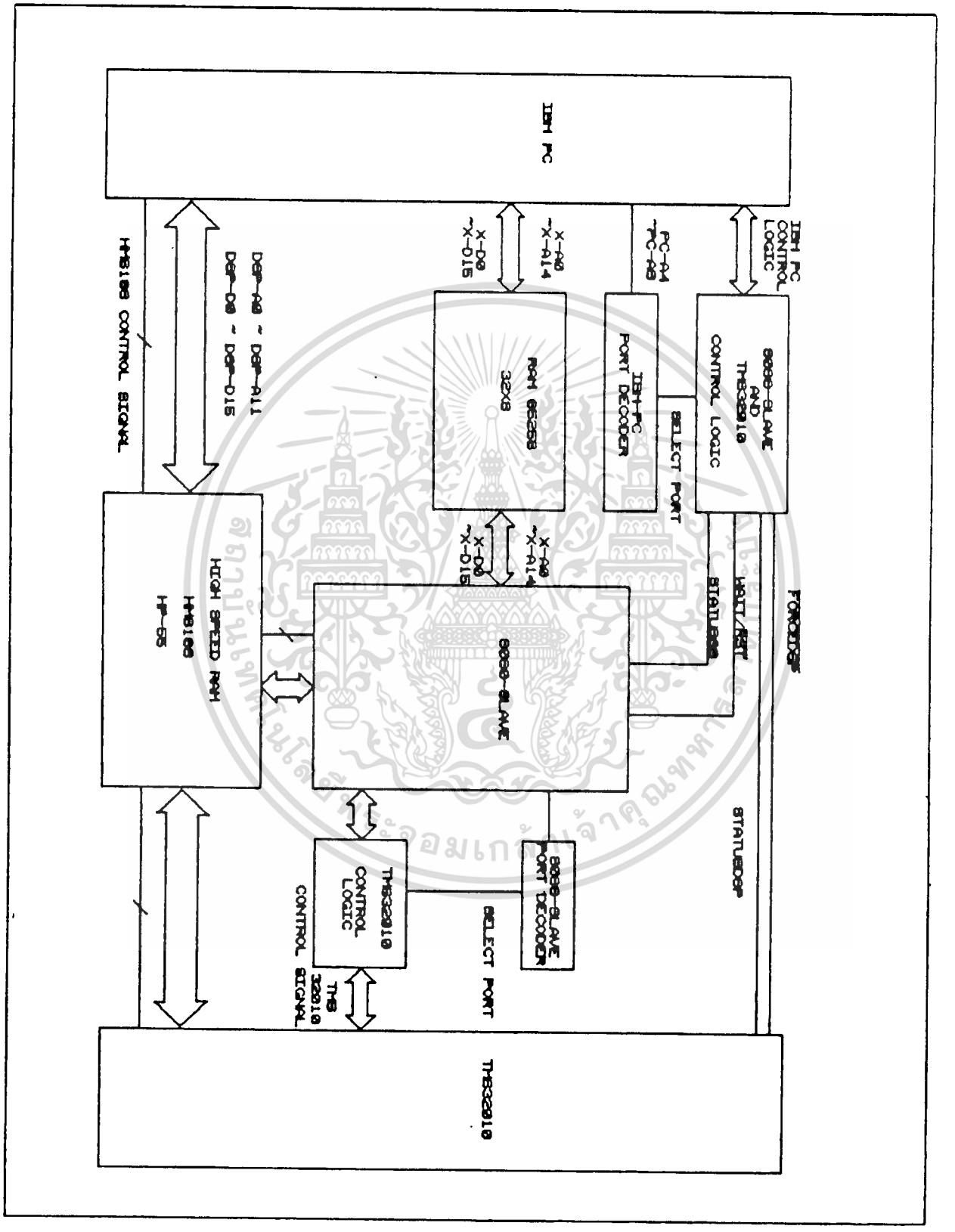
MNEMONIC	DESCRIPTION	NO. CYCLES	NO. WORDS	OPCODE INSTRUCTION REGISTER 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
DINT	Disable interrupt	1	1	0 1 1 1 1 1 1 1 1 0 0 0 0 0 0 1
EINT	Enable interrupt	1	1	0 1 1 1 1 1 1 1 1 0 0 0 0 0 1 0
LST	Load status register	1	1	0 1 1 1 1 0 1 1 1 ← D →
NOP	No operation	1	1	0 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0
POP	Pop stack to accumulator	2	1	0 1 1 1 1 1 1 1 1 0 0 1 1 1 0 1
PUSH	Push stack from accumulator	2	1	0 1 1 1 1 1 1 1 1 0 0 1 1 1 0 0
ROVM	Reset overflow mode	1	1	0 1 1 1 1 1 1 1 1 0 0 0 1 0 1 0
SOVM	Set overflow mode	1	1	0 1 1 1 1 1 1 1 1 0 0 0 1 0 1 1
SST	Store status register	1	1	0 1 1 1 1 1 0 0 1 ← D →

I/O AND DATA MEMORY OPERATIONS

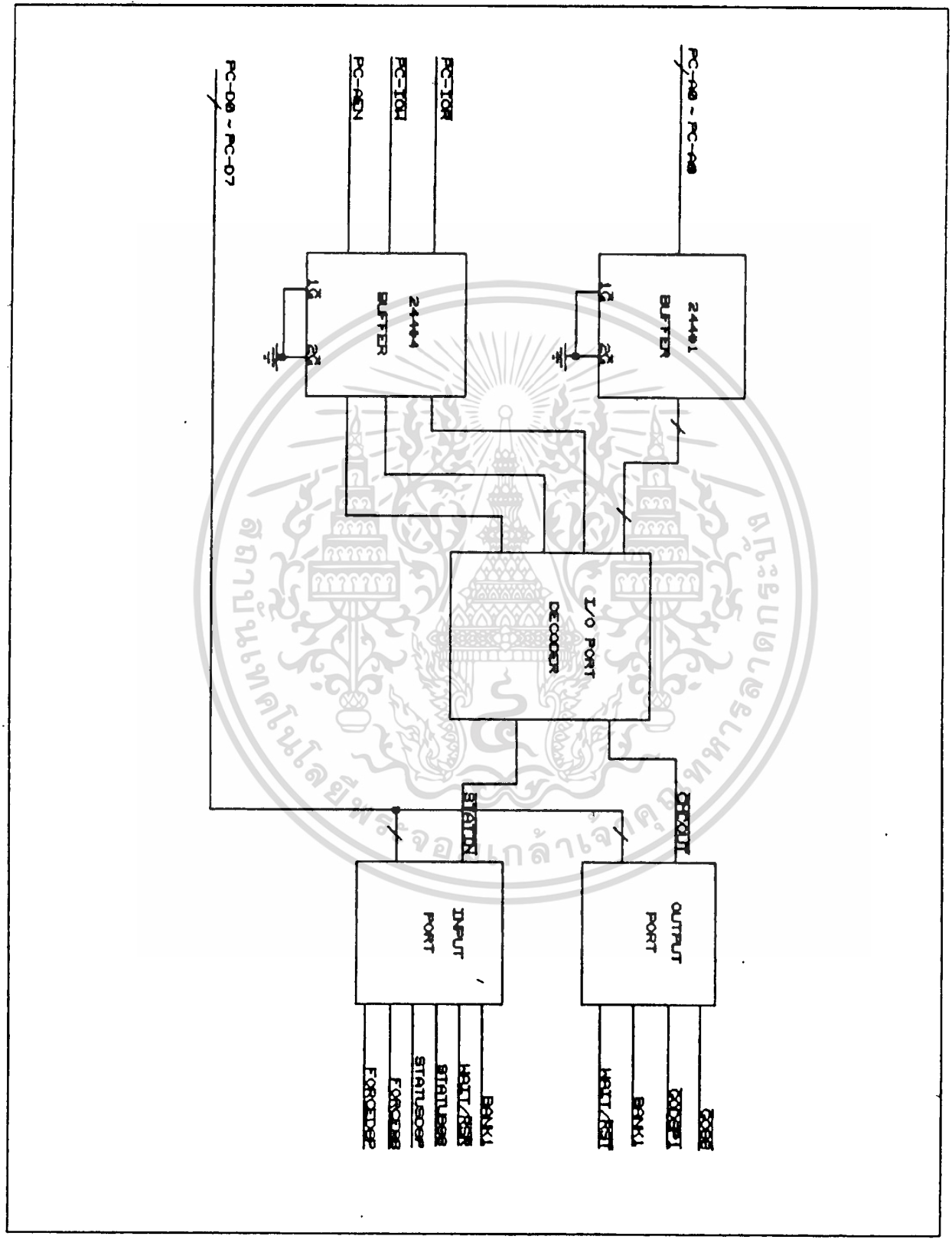
MNEMONIC	DESCRIPTION	NO. CYCLES	NO. WORDS	OPCODE INSTRUCTION REGISTER 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
DMOV	Copy contents of data memory location into next location	1	1	0 1 1 0 1 0 0 1 1 ← D →
IN	Input data from port	2	1	0 1 0 0 0 ←PA→ 1 ← D →
OUT	Output data to port	2	1	0 1 0 0 1 ←PA→ 1 ← D →
TBLR	Table read from program memory to data RAM	3	1	0 1 1 0 0 1 1 1 1 ← D →
TBLW	Table write from data RAM to program memory	3	1	0 1 1 1 1 1 0 1 1 ← D →

วงจรฮาร์ดแวร์ของระบบ

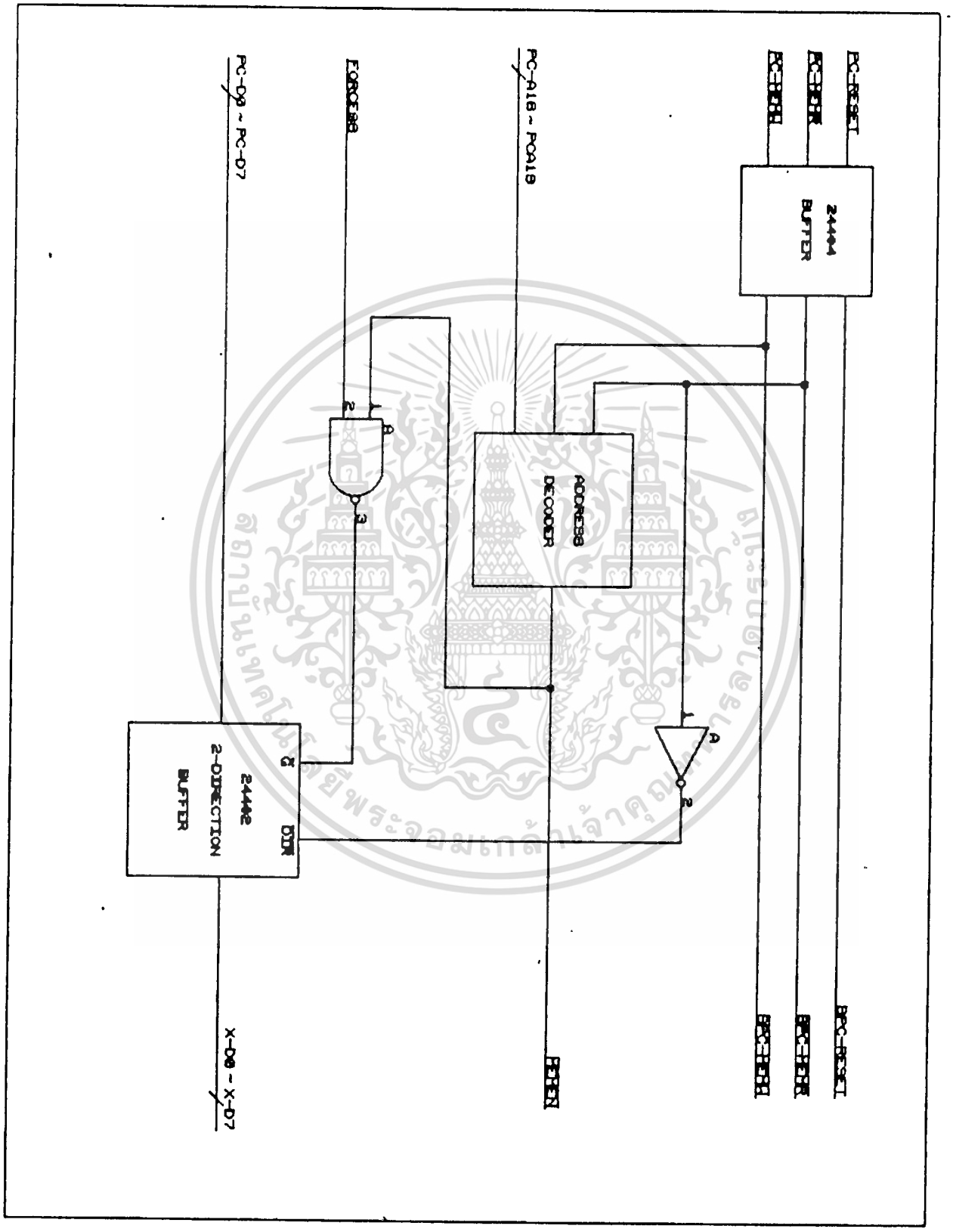
วงจรรูปที่ 1



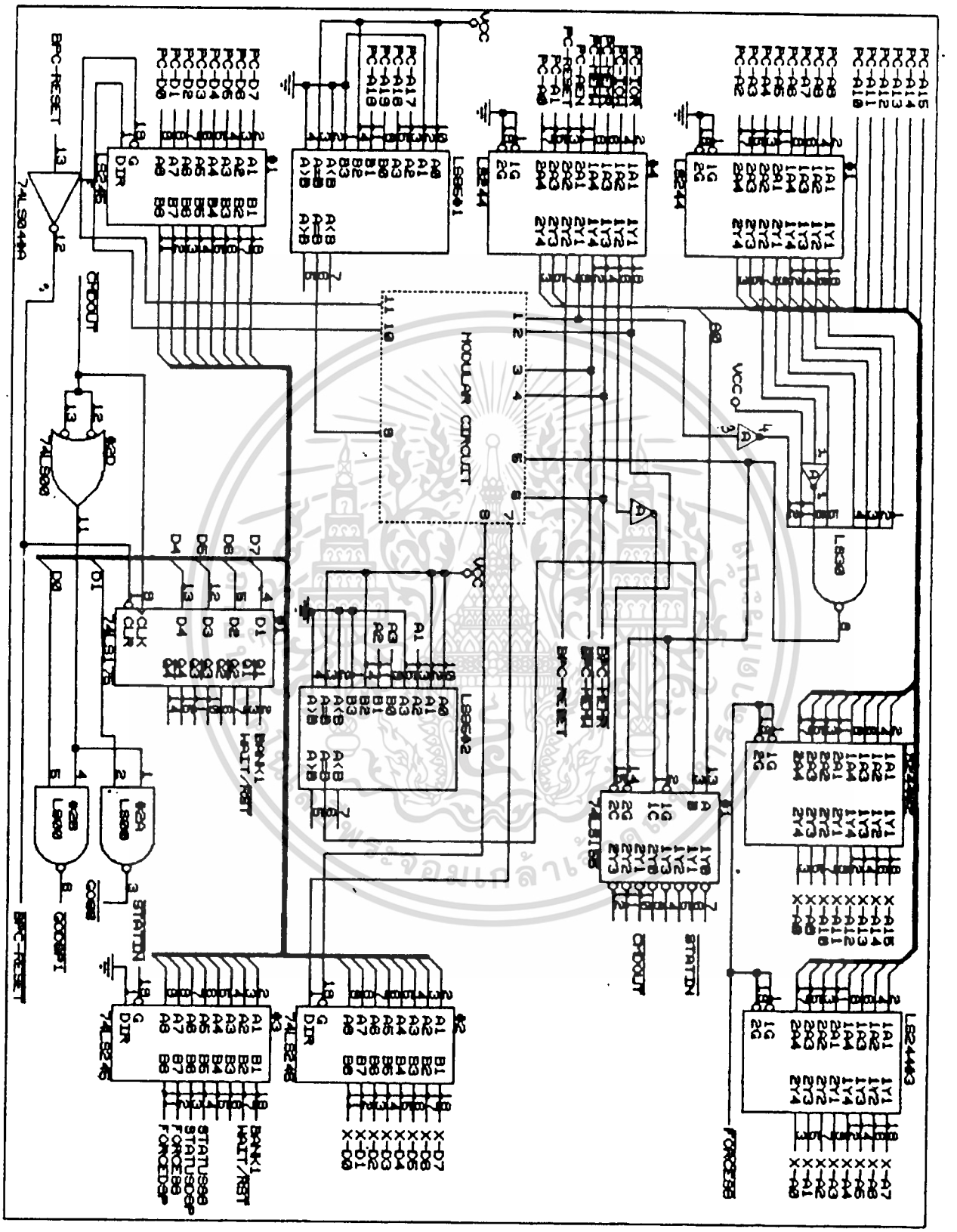
วงจรรูปที่ 2



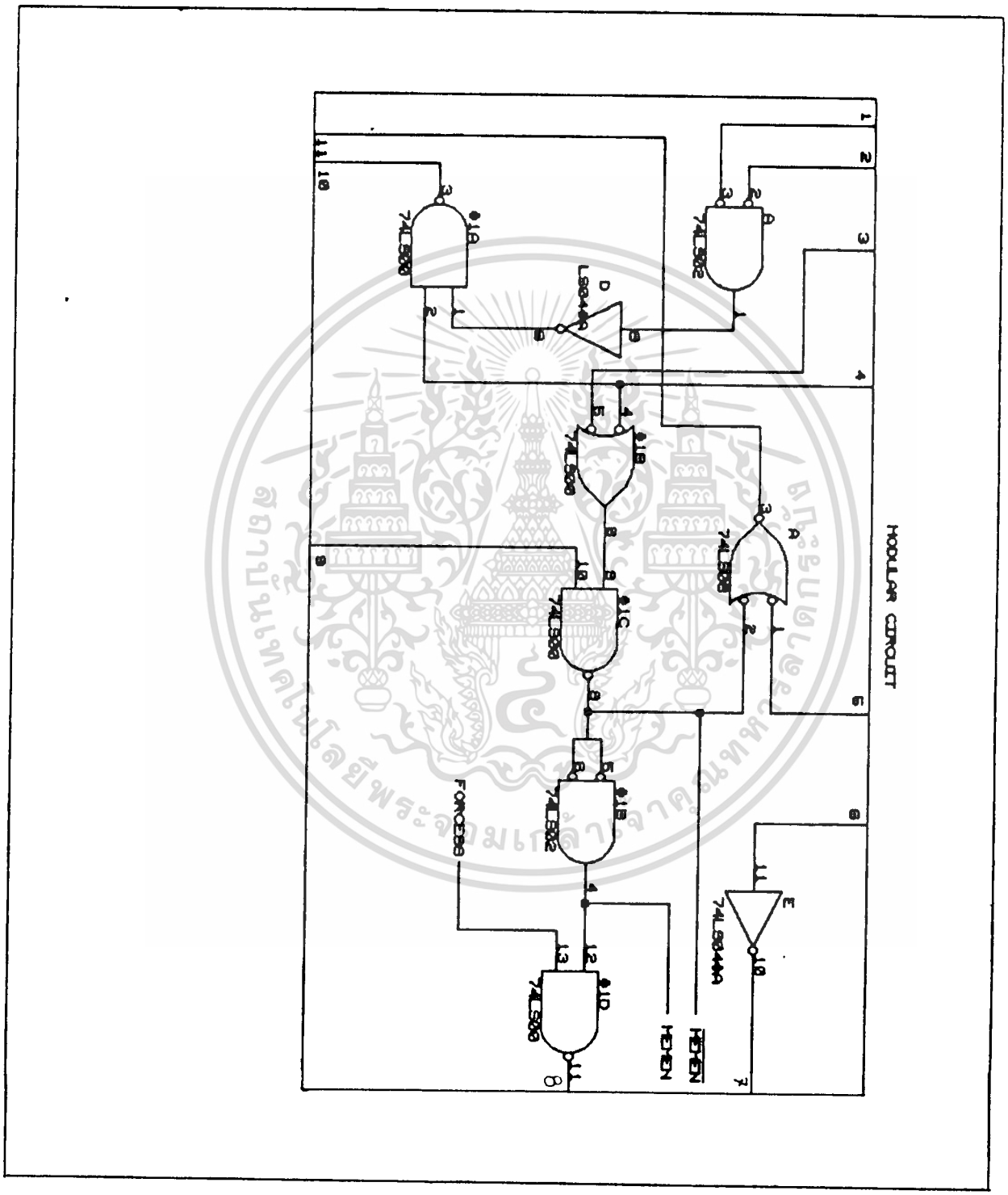
วงจรรูปที่ 3



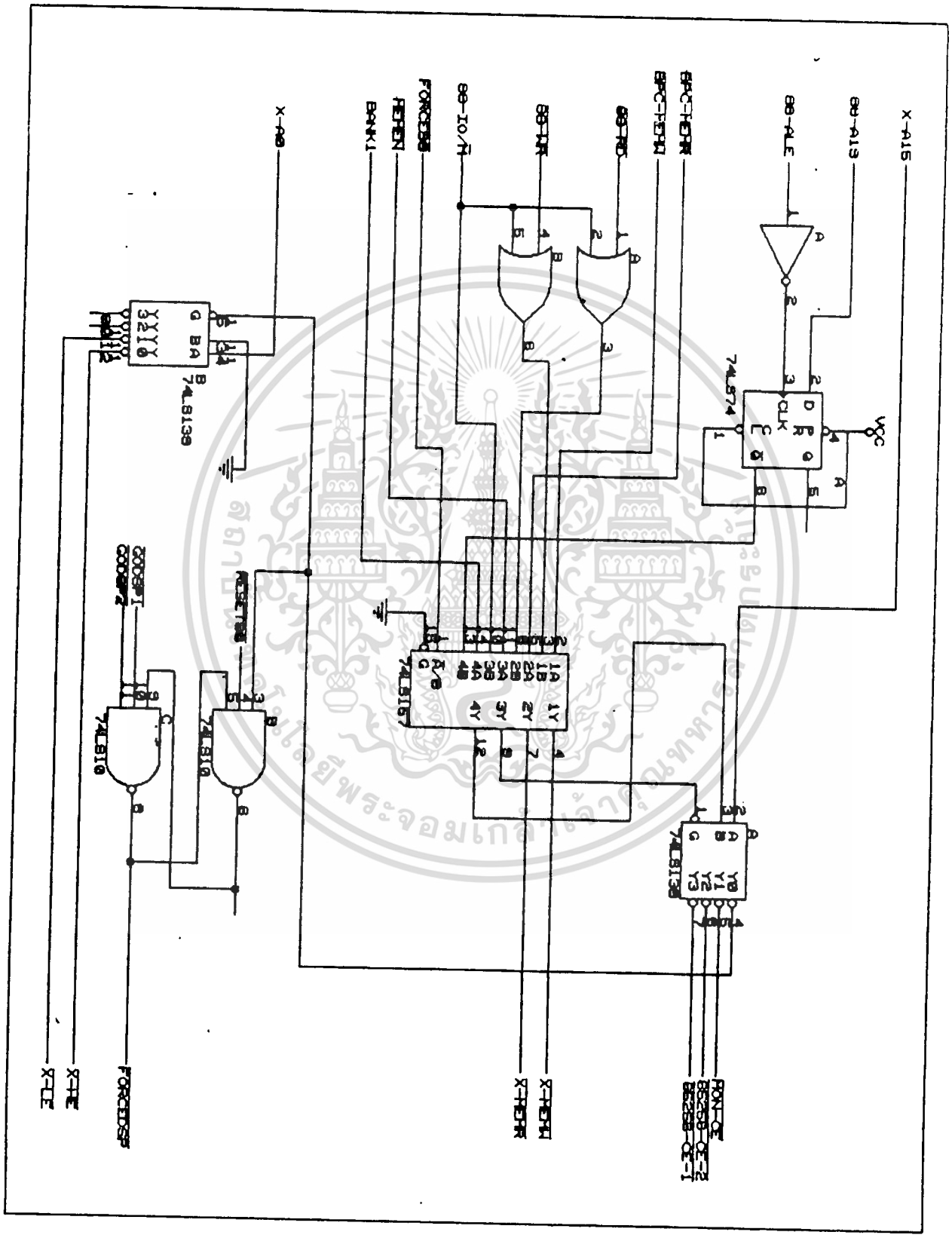
วงจรรูปที่ 4



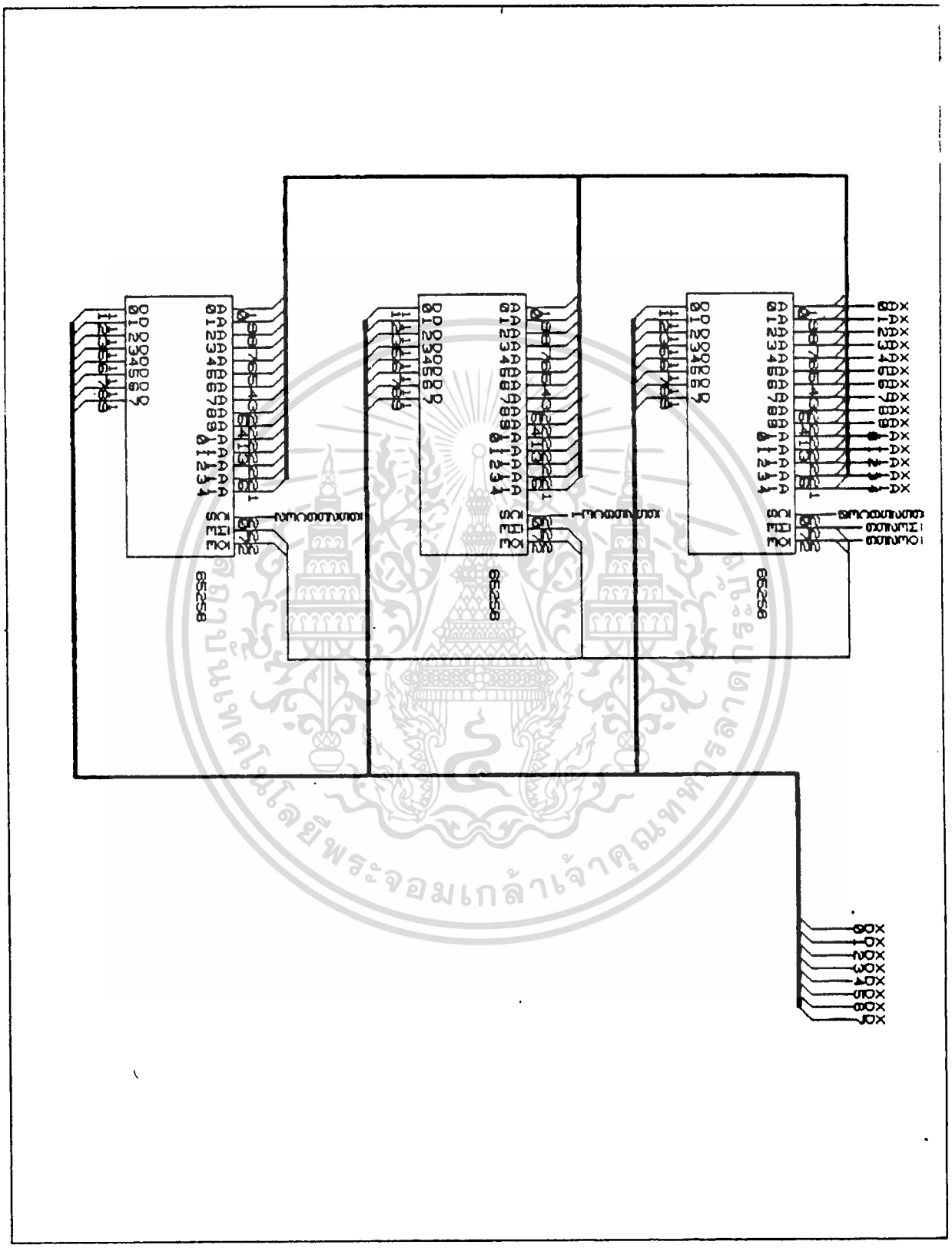
วงจรรูปที่ 5



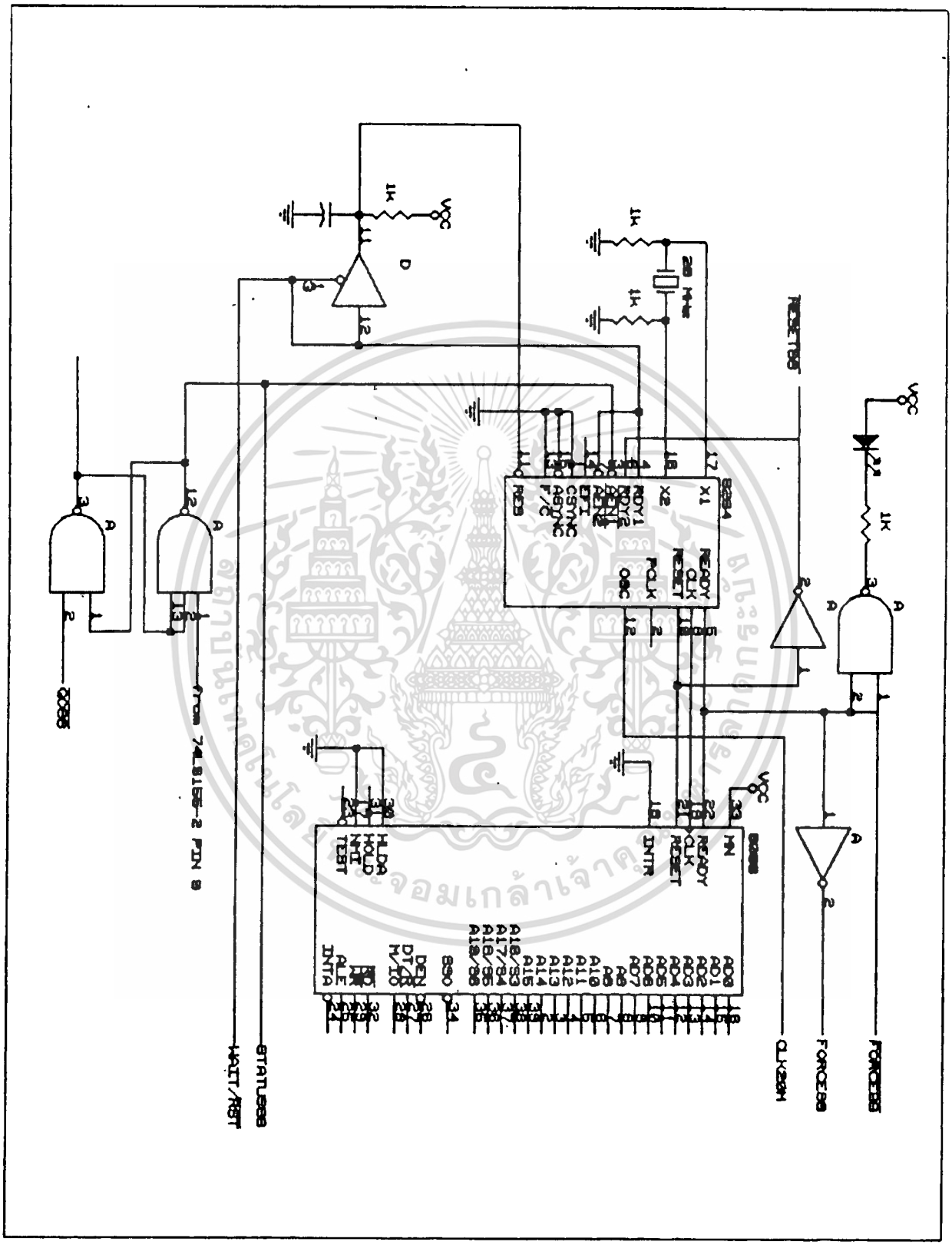
วงจรรูปที่ 6



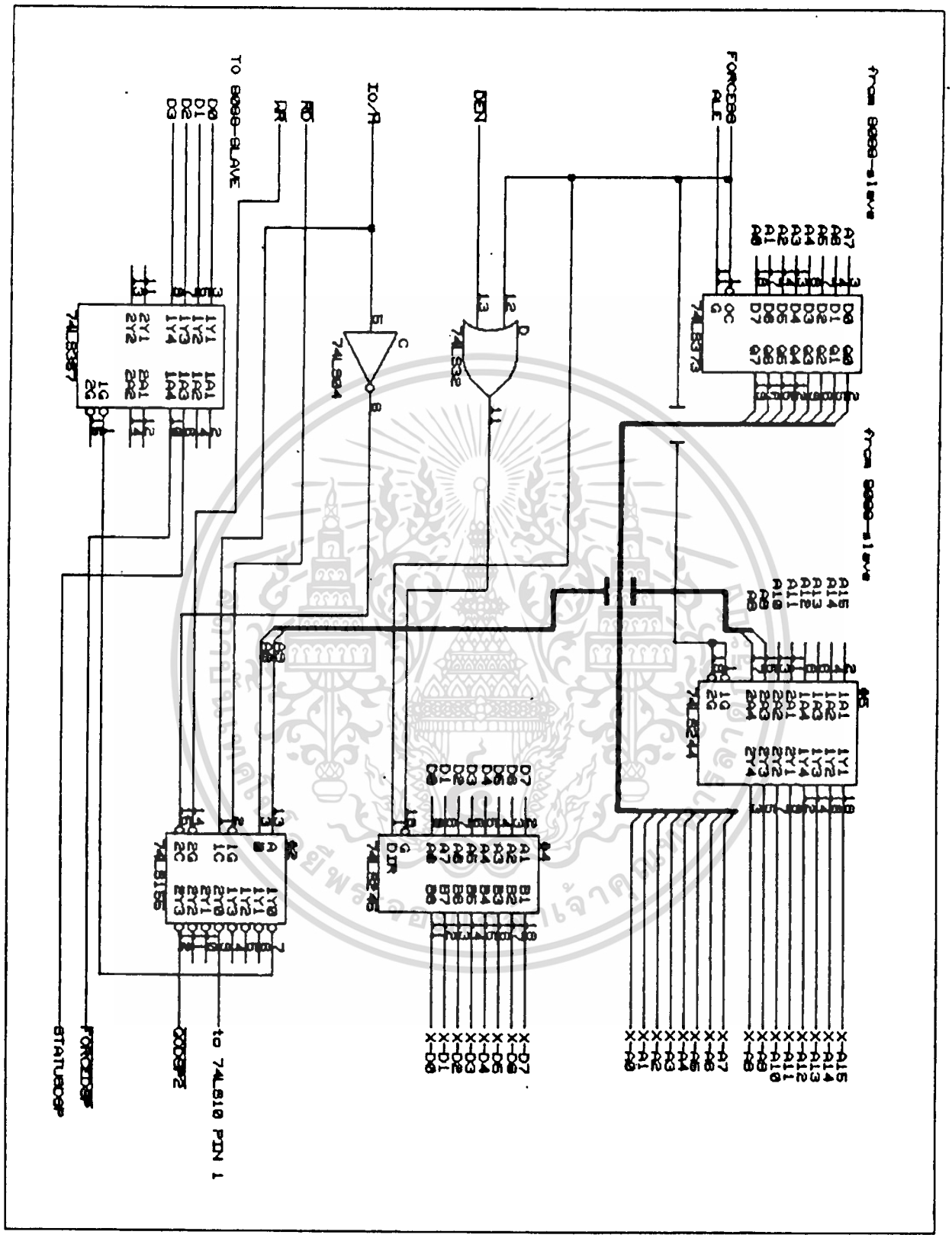
วงจรรูปที่ 7



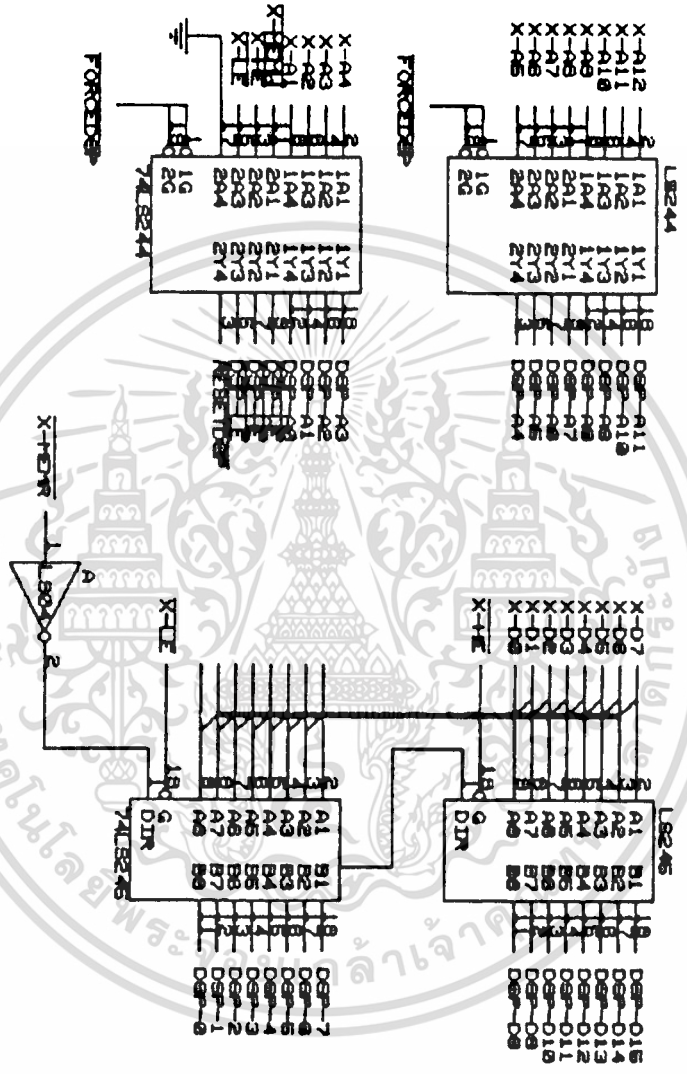
วงจรรูปที่ 8



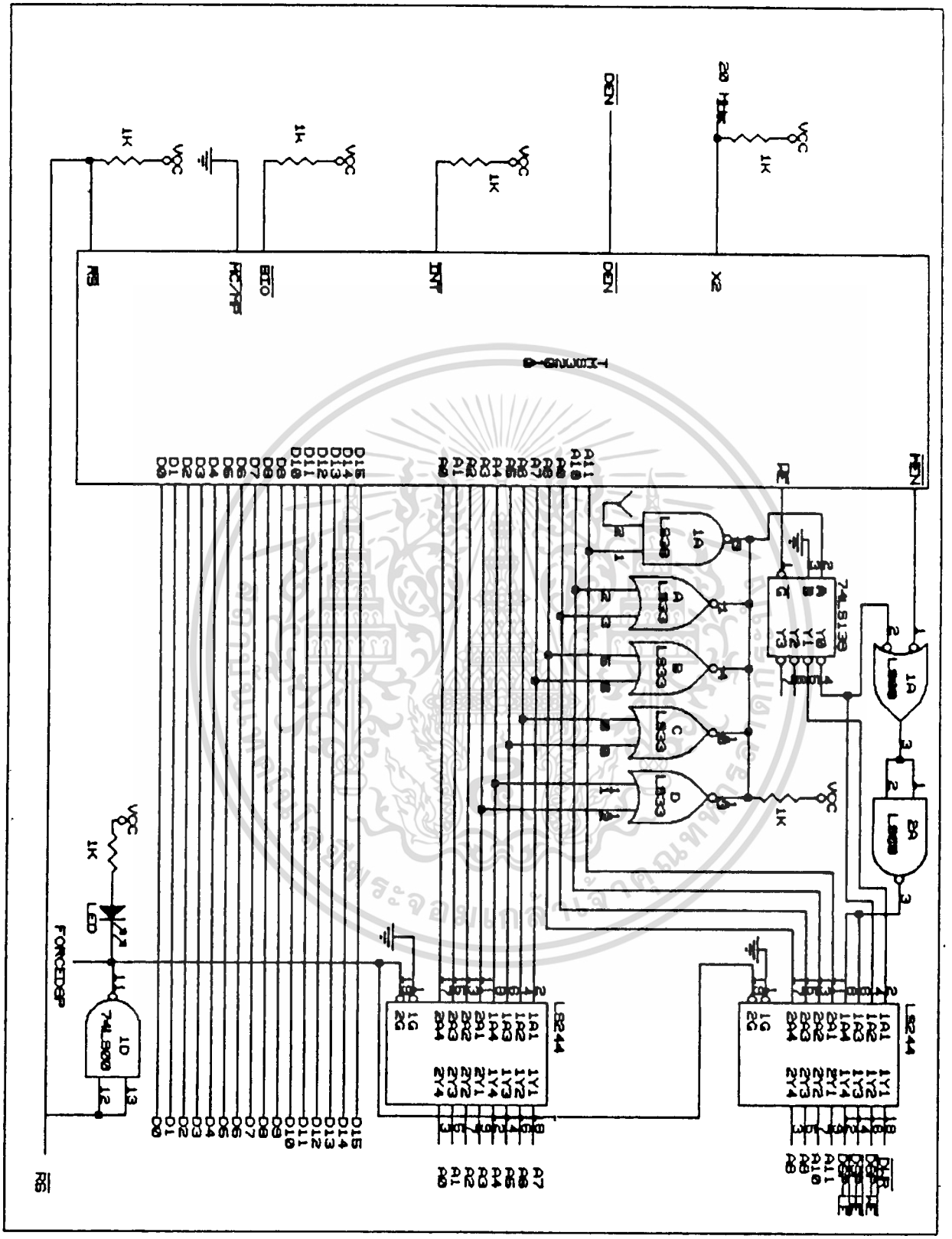
วงจรรูปที่ 9



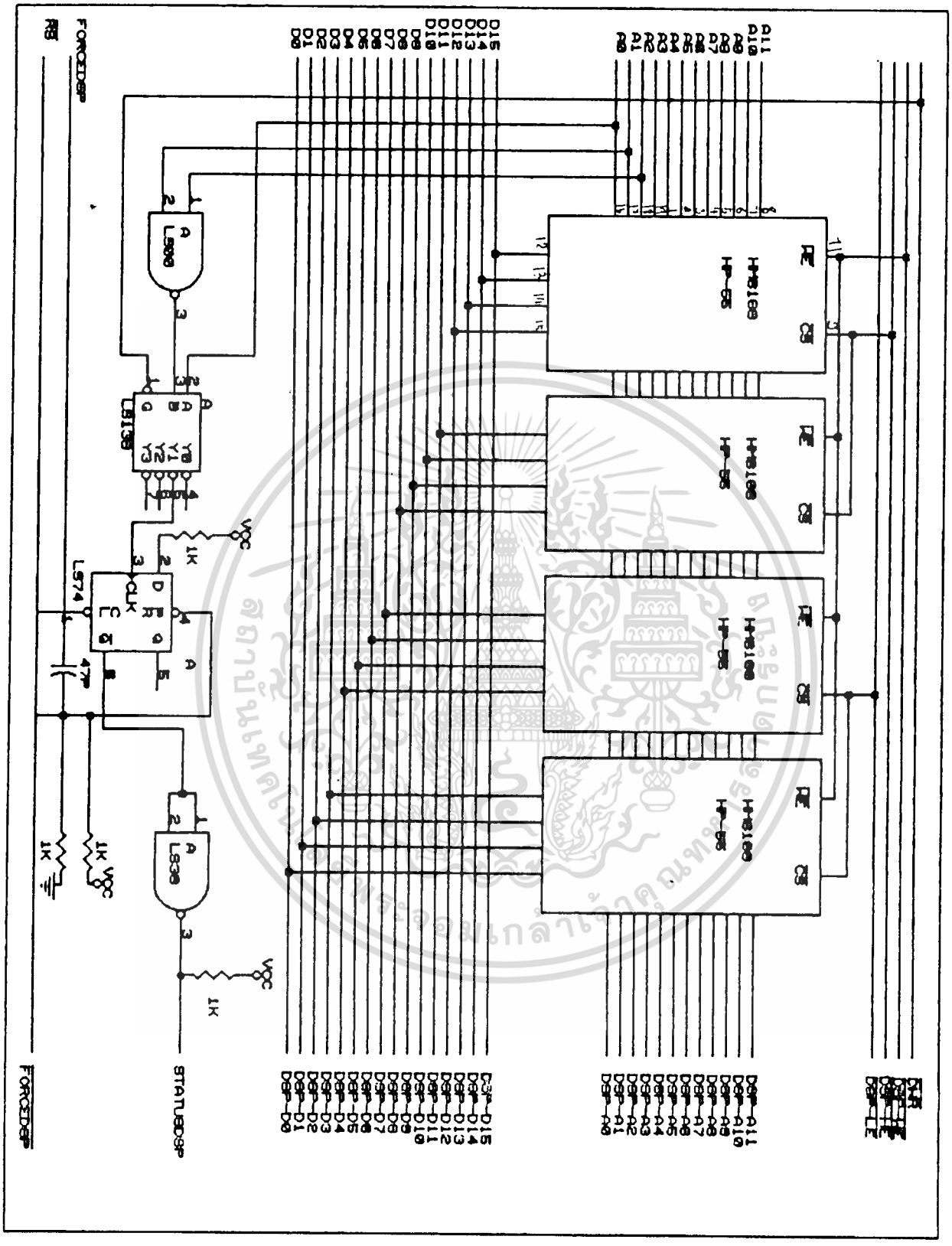
วงจรรูปที่ 10



วงจรรูปที่ 11



วงจรรูปที่ 12



```
;              PROGRAM    STEP3.ASM
;         link  with  program  st.c
;   Last Update is 21/3/32    Time 9:45:40
      TITLE DOSREAD - Read disk records created by DOS
                .model  large
                .stack  100h
        LF      EQU     10
        CR      EQU     13
        NORM    EQU     07H
        REVS    EQU     70H
        SCRM    EQU     0B000H
        SCRC    EQU     0B800H
        STG     EQU     <WELL COME TO TMS32010 DEBUGER>

                include macproc.h

SCREEN          SEGMENT   AT  0B000H
SCREEN          ENDS

MESSAGE  SEGMENT  PARA
 BUFFCLR        DB        2000 DUP(0B1h),'$'
 BUFFSEC        DB        2000 DUP(0B0H),'$'
 BUFFEMP        DB        2000 DUP(000H),'$'
 MSGSHOW        DB        ' **** START STATEMENT ******* ','$'
 STARTMG        DB        ' ...WELL COME TO TMS32010 DEBUGGER... ','$'
 RUNTMSG        DB        ' PLEASE HITS ANY KEY RUN TMS32010 ','$'
 WAITMSG        DB        ' ===== WAIT STAT ===== ','$'
 BANKMSG        DB        '                       ','$'
 MOVECMP        DB        ' ALLREADY  MOVE DATA ','$'
 PASSMSG        DB        ' PLEASE HITS ENY KEY TO CONTINUE ','$'
 CLRMEMS        DB        ' THE NEXT PROCESS IS TO CLEAR ALL MEMORY','
 testb          DB        00
MESSAGE  ENDS
DATAS    SEGMENT  PARA  PUBLIC  'DATA'
      OPENMSG1      DB          '**** OPEN FILE1 ERROR ****$'
      OPENMSG2      DB          '**** OPEN FILE2 ERROR ****$'
      OPENMSG3      DB          '**** OPEN FILE3 ERROR ****$'
      READMSG       DB          '**** READ ERROR ****$'
      DOWNLD1       DB          'TMSROM1 TXT'
      DOWNLD2       DB          'TMSROM2 TXT'
      DOWNLD3       DB          'TMSROM3 TXT'
; Start far fcb data
      FCBREC        LABEL       BYTE            ;FCB for disk file
      FCBDRIV       DB          00              ;Disk Drive DEFULT:
      FCBNAME       DB          8 DUP(?)        ;file Name
      FCBEXT        DB          3 DUP(?)        ;Extension
      FCBBLKTER     DW          0000            ;Current block #
      FCBRCSZ       DW          0000            ;Logical Record Size
                    DD          ?               ; DOS File Size
                    DW          ?               ; DOS date
                    DT          ?               ; DOS reserved
      FCBSQRC       DB          00              ;Current record #
                    DD          ?               ;Relative record #
      RECLEN        EQU         500             ;Record length
      SECTOR        DB          RECLEN DUP (' '),'$'
```

```
        ENDCDE          DB              00

DATAS       ENDS

CODES       SEGMENT     PARA    PUBLIC  'CODE'
            ASSUME      CS:CODES,DS:DATAS,ES:NOTHING,SS:STACK
            public      _dosfile
_dosfile PROC           far
         push           bp
         push           es
         push           ds
         mov            ax,DATAS
         mov            ds,ax
         mov            es,ax
         curtype        14,14
         CALL           FAR PTR _CLRMEM              ;Clear memory all
         CALL           FAR PTR _FMODULE1
         CALL           FAR PTR _FMODULE2
         CALL           FAR PTR _FMODULE3
         CALL           FAR PTR _READYRUN
         curtype        6,7
A90:
         pop            ds
         pop            es                  ;Yes - terminate
         pop            bp
         ret
_dosfile  ENDP

;---------- CALL SCRPASS PROC ----
            PUBLIC  _SCRPASS
_SCRPASS    PROC
            push        ds
            push        es
            writes      BUFFCLR,07H,0,0         ; PAINT BACKGROUND
            writes      STARTMG,7FH,20,8
            writes      PASSMSG,0f0H,23,22
            mov         ah,00
            int         16h
            pop         es
            pop         ds
            ret
_SCRPASS    ENDP
;---------- CALL FMODULE1 PROC -----------
            PUBLIC      _FMODULE1
_FMODULE1   PROC
           push     ds
           push     es
           mov      ax,DATAS
           mov      ds,ax
           mov      es,ax
           CALL     FAR PTR CLRBLOCK
           mov      ax,seg DOWNLD1
           mov      ds,ax
           lea      si,DOWNLD1
           mov      ax,seg FCBNAME
```

```
            mov     es,ax
            lea     di,FCBNAME
            cld
            mov     cx,11
            rep     movsb
A20LOOP:
            CALL    FAR PTR DOWNLOAD
            cmp     ENDCDE,05               ; TEST FOR ERROR READ
            JNZ     $+5
            JMP     TERMINATE
            mov     ax,0a000h
            mov     es,ax
            mov     di,0FFF0h           ; [ES:DI] => a000:0FFF0
;-------------------------------------------------------------------
            call    far ptr transfer
;       Transfer program & data to exp ram
;-------------------------------------------------------------------
            CMP     ENDCDE,03               ;End - File ?
            JNZ     A20LOOP                 ;No - continue
TERMINATE:
            pop     es
            pop     ds
            ret
 FMODULE1    ENDP
;---------- CALL _FMODULE2 PROC ----------
            PUBLIC  _FMODULE2
_FMODULE2   PROC
            push    ds
            push    es
; Second
        CALL        FAR PTR CLRBLOCK
        cld
        mov         ax,seg DOWNLD2
        mov         ds,ax
        lea         si,DOWNLD2
        mov         ax,seg FCBNAME
        mov         es,ax
        lea         di,FCBNAME
        mov         cx,11
        rep         movsb
        CALL        FAR PTR E10OPEN     ; Open File,set DTA
        CMP         ENDCDE,00           ; Valid Open ?
        JZ          $+5                 ; Yes - continue
        JMP         SECEND
        cld
        mov         ax,0a000h
        mov         es,ax
        mov         di,8000h            ; [ES:DI] => a000:8000
SECLOOP:
        CALL        FAR PTR F10READ         ;Read disk record
        CMP         ENDCDE,00           ;Normal read ?
        JZ          SEC2                ;Yes - continue
        CMP         ENDCDE,03           ;End - File,some data ?
        JZ          $+5                 ;No - exit
        JMP         SECEND
```

```
SEC2:
;-----------------------------------------------------------------
        call        far ptr transfer
;     Transfer program & data to expansion ram
;-----------------------------------------------------------------
        CMP         ENDCDE,03        ;End - File ?
        JNZ         SECLOOP          ;No - continue
SECEND:
            pop     es
            pop     ds
_FMODULE2   ENDP
;---------- CALL _FMODULE3 PROC ----------
           PUBLIC _FMODULE3
_FMODULE3    PROC
             push    ds
             push    es
; THIRD
             CALL        FAR PTR CLRBLOCK
             cld
             mov         ax,seg DOWNLD3
             mov         ds,ax
             lea         si,DOWNLD3
             mov         ax,seg FCBNAME
             mov         es,ax
             lea         di,FCBNAME
             mov         cx,11
             rep         movsb

             CALL        FAR PTR E10OPEN          ;Open File,set DTA
             CMP         ENDCDE,00        ;Valid Open ?
             JZ          $+5
             JMP         TRDEND           ;Yes - continue
             cld
             mov         ax,0a000h
             mov         es,ax
             mov         di,0000h          ; [ES:DI] => a000:0000
TRDLOOP:
             CALL        FAR PTR F10READ          ;Read disk record
             CMP         ENDCDE,00       ;Normal read ?
             JZ          TRDNEXT         ;Yes - continue
             CMP         ENDCDE,03       ;End - File,some data ?
             JNZ         TRDEND          ;No - exit
TRDNEXT:
;-----------------------------------------------------------------
             call        far ptr transfer
;        Transfer program & data to expansion ram
;-----------------------------------------------------------------
             CMP         ENDCDE,03        ;End - File ?
             JNZ         TRDLOOP          ;No - continue
TRDEND:
             pop     es
             pop     ds
             ret
_FMODULE3    ENDP
;---------- CALL DOWNLOAD PROC -----------
```

```
            PUBLIC  DOWNLOAD
DOWNLOAD    PROC
            push    es
            push    ds
            CALL    FAR PTR E10OPEN          ;Open File,set DTA
            CMP     ENDCDE,00                ;Valid Open ?
            JZ      $+5                      ;Yes - continue
            JMP     D90
            cld                              ;Clear Direction Flage
D20LOOP:
            CALL    FAR PTR F10READ          ;Read disk record
            CMP     ENDCDE,00                ;Normal read ?
            JZ      D30                      ;Yes - continue
            CMP     ENDCDE,03                ;End - File,some data ?
            JZ      D30                      ;No - exit
D90:        mov     ENDCDE,09
D30:
            pop     ds
            pop     es
            ret
DOWNLOAD    ENDP
;---------- CALL CLEAR BLOCK -------------
            PUBLIC  CLRBLOCK
CLRBLOCK    PROC
            push    es
            push    ds
            push    dx
            cld
            mov     cx,500
            mov     ax,seg FCBREC
            mov     es,ax
            lea     di,FCBREC
            mov     ax,00
            rep     stosw
            pop     dx
            pop     ds
            pop     es
            ret
CLRBLOCK    ENDP
;---------- READY PROC -------------------
            PUBLIC  _READYRUN
_READYRUN   PROC
            push    ds
            push    dx
            writes  RUNTMSG,8FH,23,22
            mov     ah,0
            int     16h
            pop     dx
            pop     ds
            ret
_READYRUN   ENDP
;---------- CALL  _OPERATE ---------------
            PUBLIC _OPERATE
_OPERATE    PROC
            push    ds
```

```
            push    es
;-------------------------------
;         ON  8088-SLAVE
;-------------------------------
            mov         dx,3eeh
            mov         al,40h
            out         dx,al
            mov         al,42h
            out         dx,al
;-------------------------------
;        CHECK 8088-SLAVE STATUS
;-------------------------------
            mov         ax,seg testb
            mov         es,ax
            mov         es:testb,15
            mov         dx,3eeh
L100:       in          ax,dx
            cmp         al,74h
            jnz         $+5
            jmp         F00
            call        far ptr waitstat
            jmp         L100
;-------------------------------
;       OFF 8088-SLAVE
;-------------------------------
F00:        mov         dx,3eeh
            mov         al,00
            out         dx,al
            pop     es
            pop     ds
OPERATE     ENDP

;---------- MAIN  PROGRAM  OFF -----------
            public  waitstat
waitstat    proc
            push    ds
            push    dx
            assume  cs:CODES
            mov     ax,SEG testb
            mov     ds,ax
            writes  WAITMSG,8FH,25,16
            inc     testb
            mov     al,testb
            cmp     al,19
            jz      $+5
            jmp     stopw
            mov     al,14
            mov     testb,al
            dosprint BANKMSG,25,16
;           writes  BANKMSG,07H,25,14
;           writes  BANKMSG,07H,25,15
;           writes  BANKMSG,07H,25,16
;           writes  BANKMSG,07H,25,17
;           writes  BANKMSG,07H,25,18
stopw:      pop     dx
```

```
            pop     ds
            ret
waitstat    endp
;-----------------------------------------------
;  CLEAR MEMORY OF TWO BANK
;-----------------------------------------------
            PUBLIC _CLRMEM
_CLRMEM     PROC
            push    ds
            push    es
            mov     dx,3eeh
            mov     al,80h
            out     dx,al                   ; set bank 1
            mov     ax,0a000h
            mov     ds,ax
            call    far ptr trnmem
            mov     dx,3eeh
            mov     al,00h
            out     dx,al                   ; set bank 0
            call    far ptr trnmem
            writes      MOVECMP,0F0H,25,10
            pop     es
            pop     ds
            ret
_CLRMEM     ENDP

trnmem      proc
            push    ds
            mov     bx,0000h
            mov     cx,0ffffh
  strb10:   mov     al,00h
            mov     [bx],al
            inc     bx
            loop    strb10
            pop     ds
            ret
trnmem      endp
;-----------------------------------------------------------------
;------------        OPEN DISK FILE          ---------------------
;-----------------------------------------------------------------
            public  E10OPEN
E10OPEN     PROC
            PUSH    DS
            MOV     AX,SEG FCBREC
            MOV     DS,AX
            LEA     DX,FCBREC
            MOV     AH,0FH              ;Request Open
            INT     21H
            CMP     AL,00               ;File found ?
            JNZ     E20                 ;No - error
            MOV     FCBRCSZ,RECLEN      ;SET RECORD LENGTH (EQU)
            MOV     AH,1AH
            LEA     DX,SECTOR           ;SET ADDRESS DTA
            INT     21H
            POP     DS
```

```
                RET
E20:
                MOV     ENDCDE,01          ;Error message
                writes  OPENMSG1,70H,25,12
                POP     DS
                RET
E10OPEN         ENDP
;-----------------------------------------------------------------
;---------------      READ  DISK SECTOR      ---------------------
;-----------------------------------------------------------------
                PUBLIC  F10READ
F10READ         PROC
                MOV    AH,14H              ;Request Read
                LEA    DX,FCBREC
                INT    21H
                CMP    AL,00               ;File found ?
                JZ     F90                 ;Yes - exit
                MOV    ENDCDE,AL
                CMP    AL,01               ;End -of- File /
                JZ     F90                 ;Yes - exit
                CMP    AL,03               ;End -of- File /
                JZ     F90                 ;Yes - exit
                writes READMSG,70H,25,14
F90:            RET
F10READ         ENDP
;-----------------------------------------------------------------
;------------------- TRANSFER to RAM -----------------------------
;-----------------------------------------------------------------
              PUBLIC  transfer      ; 8000h
transfer      proc
              mov   dx,3eeh         ; Port 3ee on menory
              mov   al,00h          ; Bank 0(ROM) on
              out   dx,al
              lea   si,SECTOR
              mov   cx,FCBRCSZ      ;counter
 countmem:    mov   al,[si]
              mov   es:[di],al
              inc   si
              inc   di
              loop  countmem
              ret
transfer      endp
;-----------------------------------------------------------------
;---------------  DISK ERROR ROUTINE:     ----------------------
;-----------------------------------------------------------------
              PUBLIC  X10ERR
X10ERR        PROC
              MOV     AH,09         ;DX contains address of message
              INT     21H
              RET
X10ERR        ENDP
;-----------------------------------
CODES         ENDS
              end
```

```
/*************************************************************/
/*                  PROGRAME  ST.EXE                         */
/*   DATE  21/3/89                TIME : 21.30  P.M.        */
/*************************************************************/
#define MAX_MENU  20
#define MAX_FRAME 20
#include <stdio.h>
#include <dos.h>
#include "tmshead.h"

char far *vid_men;
extern void dosfile();
extern void SCRPASS();
xtern void operate();
struct time {
                unsigned char ti_hour;
                unsigned char ti_min;
                unsigned char ti_sec;
                unsigned char ti_hund;
                float         total;
            }now1,now2;
runp()
{
 char ch;
  clearscr(0xb0);
  while(((ch=popup(0))!=-1)&&(ch!=5)) {
   switch(ch) {
    case 0:
            clrmem();
            break;
    case 1:
            fmodule1();
            break;
    case 2:
            fmodule2();
            break;
    case 3:
            fmodule3();
            break;
    case 4:
            fillscreen();
            tmsrunproc();
            break;
    default:
            break;
   }
  }
}
tmsrunproc()
{
 float  timeuse;
  ntime(&now1.ti_hour,&now1.ti_min,&now1.ti_hund,&now1.ti_sec
        ,&now1.total);
  operate();
  ntime(&now2.ti_hour,&now2.ti_min,&now2.ti_hund,&now2.ti_sec
```

```
        ,&now2.total);
  timeuse=(now2.total-now1.total)/100.0;
  setcur(28,19);
  printf("TIME USE = %f",timeuse);
  exitproc();
}
ramaddress()
{
 char vmode;
 vmode=video_mode();
 if((vmode!=2)&&(vmode!=3)&&(vmode!=7)){
   printf("video must be in 80 column text mode");
   exit(1);
 }
 /* set proper address of ram      */
 if(vmode==7) vid_men = (char far *) 0xb0000000;
   else       vid_men = (char far *) 0xb8000000;
}
fillscreen()
{
  clearscr(0xb0);
  write_string(15,8
    ,"NOW TMS32010 ARE WORKING WHILE PC CHECK STATUS",0X8F);
  windowx(2);
}
exitproc()
{
  write_string(15,8,
       " OK TMS32010 ARE COMPLETE WORKING YOUR JOB.. ",0X8F);
  write_string(30,23,"HIT ANY KEY TO EXIT",0x8f);
  getch();
  restore_window(2);
  clearscr(00);
}
ntime(h,m,r,s,b)
int  *h,*m,*s,*r;
float *b;
{
    int wh,wm,ws,wr;
    union  REGS  u;
    u.h.ah=0x2c;
    intdos(&u,&u);
    *h=u.h.ch;     wh=*h;
    *m=u.h.cl;     wm=*m;
    *s=u.h.dh;     ws=*s;
    *r=u.h.dl;     wr=*r;
    *b=(float)wh*360000.0+(float)wm*6000.0+(float)ws*100.0+(float)wr;
}
main()
{
  ramaddress();
  initscreen();
  SCRPASS();
  runp();
}
```

```
*****************************************************
*                PROGRAM  GENER1.ASM                *
*   DATE 21/3/89                  TIME  23:35:40 P.M  *
*****************************************************
        IDT 'GENER1'
        AORG   >0
        B  START
WRTLOC   EQU    >800
*********************************
*    DATA MEMORY LOCATION
*********************************
ZERO    EQU    0
DELTA   EQU    1
ALPHA   EQU    2
SINA    EQU    3
TEMP    EQU    4
MASK    EQU    5
OFSET   EQU    6
ONE     EQU    7
LOOIN   EQU    100
LOOUT   EQU    104
*********************************
*  INITIAL VARIALBLE PROCEDURE
*********************************

START   LDPK   0
        LACK   0
        SACL   ZERO              * DATA0 = 0
        LACK   1
        SACL   ONE               * ONE = 1
        LACK   M1
        TBLR   MASK
        LACK   SINE
        SACL   OFSET
        ZAC
        SACL   ALPHA
        LACK   D1
        TBLR   DELTA
        LACK   LPI
        TBLR   LOOIN
        LACK   LPO
        TBLR   LOOUT
        LDPK   0
        LAR    AR1,LOOUT
*
L0      LAR    AR0,LOOIN
        LT     ONE
        MPYK   WRTLOC
        PAC
L1      CALL   SWAVE1
*********************************
*  PUSH SINE VALUE TO MEMORY    *
*********************************
        TBLW   SINA
        ADD    ONE
```

```
          LARP    AR0
          BANZ    L1
          LARP    AR1
          BANZ    L0
*
          OUT     ONE,7
WAITS     NOP
          B       WAITS
*
SWAVE1    PUSH
          LAC     ALPHA,8
          SACH    TEMP
          LAC     TEMP
          ADD     OFSET
          TBLR    SINA
          LAC     ALPHA
          ADD     DELTA
          AND     MASK
          SACL    ALPHA
          POP
          RET
***************************
* LOCATION VARIALBLE
***************************
M1        DATA    >7FFF
D1        DATA    >40             *  0.25  IN  Q8 FORMAT
LPI       DATA    511             *  512
LPO       DATA    511
***************************
* SINE TABLE  LOOGUP
***************************
SINE      DATA    >0
          DATA    >324
          DATA    >646
          DATA    >964
          DATA    >C7C
          DATA    >F8D
          DATA    >1294
          DATA    >1590
          DATA    >187E
          DATA    >1B5D
          DATA    >1E2B
          DATA    >20E7
          DATA    >238E
          DATA    >2620
          DATA    >289A
          DATA    >2AFB
          DATA    >2D41
          DATA    >2F6C
          DATA    >3179
          DATA    >3368
          DATA    >3537
          DATA    >36E5
          DATA    >3871
          DATA    >39DB
```

```
DATA    >3B21
DATA    >3C42
DATA    >3D3F
DATA    >3E15
DATA    >3EC5
DATA    >3F4F
DATA    >3FB1
DATA    >3FEC
DATA    >4000
DATA    >3FEC
DATA    >3FB1
DATA    >3F4F
DATA    >3EC5
DATA    >3E15
DATA    >3D3F
DATA    >3C42
DATA    >3B21
DATA    >39DB
DATA    >3871
DATA    >36E5
DATA    >3537
DATA    >3368
DATA    >3179
DATA    >2F6C
DATA    >2D41
DATA    >2AFB
DATA    >289A
DATA    >2620
DATA    >238E
DATA    >20E7
DATA    >1E2B
DATA    >1B5D
DATA    >187E
DATA    >1590
DATA    >1294
DATA    >F8D
DATA    >C7C
DATA    >964
DATA    >646
DATA    >324
DATA    >0
DATA    >FCDC
DATA    >F9BA
DATA    >F69C
DATA    >F384
DATA    >F073
DATA    >ED6C
DATA    >EA70
DATA    >E782
DATA    >E4A3
DATA    >E1D5
DATA    >DF19
DATA    >DC72
DATA    >D9E0
DATA    >D766
```

```
DATA    >D505
DATA    >D2BF
DATA    >D094
DATA    >CE87
DATA    >CC98
DATA    >CAC9
DATA    >C91B
DATA    >C78F
DATA    >C625
DATA    >C4DF
DATA    >C3BE
DATA    >C2C1
DATA    >C1EB
DATA    >C13B
DATA    >C0B1
DATA    >C04F
DATA    >C014
DATA    >C000
DATA    >C014
DATA    >C04F
DATA    >C0B1
DATA    >C13B
DATA    >C1EB
DATA    >C2C1
DATA    >C3BE
DATA    >C4DF
DATA    >C625
DATA    >C78F
DATA    >C91B
DATA    >CAC9
DATA    >CC98
DATA    >CE87
DATA    >D094
DATA    >D2BF
DATA    >D505
DATA    >D766
DATA    >D9E0
DATA    >DC72
DATA    >DF19
DATA    >E1D5
DATA    >E4A3
DATA    >E782
DATA    >EA70
DATA    >ED6C
DATA    >F073
DATA    >F384
DATA    >F69C
DATA    >F9BA
DATA    >FCDC

END
```

กิตติกรรมประกาศ

ปริญญานิพนธ์ฉบับนี้เสร็จสมบูรณ์ได้ต้องขอขอบพระคุณ

รศ. วิพันธ์ ปรีชาพานิช และ ศจ. ดร. ไพรัช ธัชยพงษ์ ที่ได้กรุณารับเป็นอาจารย์ที่ปรึกษา

อาจารย์ กวิน แหยมมั่น ที่เอื้อเฟื้อสถานที่และอุปกรณ์ในการจัดทำโครงงานนี้จนลุล่วง

หนังสืออ้างอิง

1. Kun-Shan Lin, "Digital Signal Processing Application with TMS320 Family, Volume I", 269 p., 1987.
2. Roman Kuc, "Introduction to Digital Signal Processing", McGRAW-HILL, 20-31 p., 1988.
3. Russell Rector-George Alexy, "The 8086 Book include the 8088", Osbone/McGRAW-HILL, 7-1 p., 1980.
4. "IBM Technical Reference Personal Computer Hardware Reference Library", 1983.
5. TEXAS INSTRUMENT, "TMS32010 User's Guide", 1983.