# Digital Signal Processing
# TMS320C25


ผู้จัดทำ

นาย นัทธพงศ์ อิ่มบำรุง 321141

นาย ลิขิต ธวัชชัยวิรุฒย์ 321269

นาย วสันต์ สนธิเพิ่มพูน 321287


อาจารย์ที่ปรึกษา

อาจารย์ สุรสิทธิ์ วรรณไกรโรจน์

วิทยานิพนธ์สำหรับปริญญาวิศวกรรมศาสตร์

สาขาวิชาอิเล็กทรอนิคส์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหาร ลาดกระบัง

ปีการศึกษา 2533

ปริญญานิพนธ์ ปีการศึกษา 2535

สาขาวิชาอิเล็กทรอนิคส์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหาร ลาดกระบัง

เรื่อง ดิจิตอลซิกแนลโปรเซสซิ่ง

( Digital Signal Processing )


ผู้จัดทำ

นาย นัทธพงศ์ อิ่มบำรุง 321141

นาย ลิขิต ธวัชชัยวิรุตษ์ 321269

นาย วสันต์ สนธิเพิ่มพูน 321287


อ. สุรสิทธิ์ วรรณไกรโรจน์ อาจารย์ที่ปรึกษา

(_____)

# Digital Signal Processing TMS320C25


นายนัทธพงศ์ อิ่มบำรุง 32.1141

นายลิขิต ธวัชชัยวิรุตษ์ 32.1269

นายวสันต์ สนธิเพิ่มพูน 32.1287

อ.สุรสิทธิ์ วรรณไกรโรจน์ อาจารย์ที่ปรึกษา


ปีการศึกษา 2535


**บทคัดย่อ**

ในปัจจุบันนี้ดิจิตอลซิกแนลโปรเซสซิ่งมีการพัฒนาไปอย่างรวดเร็วทำให้สามารถทำงานด้วยความเร็วที่สูงขึ้น และสามารถนำไปประยุกต์ใช้งานในขอบเขตที่กว้างขึ้นซึ่งหากนำระบบดิจิตอลซิกแนลโปรเซสซิ่งมาใช้ในการประมวลผลสัญญาณ แทนวงจรอนาลอกจะทำให้ได้ผลในการโปรเซสที่มีคุณภาพดีกว่าและมีความยืดหยุ่นดีกว่า

ปริญญานิพนธ์นี้จะเป็นการศึกษาทฤษฎีทางดิจิตอลซิกแนลโปรเซสซิ่ง และทำการออกแบบสร้างบอร์ดตัวประมวลผลสัญญาณเชิงเลข โดยใช้ตัวประมวลผลสัญญาณเชิงเลขเบอร์ TMS320C25 ของบริษัทเท็กซัสอินสตรูเมนต์ และในการทำงานจะใช้เสียบลงบนเครื่องไมโครคอมพิวเตอร์ โดยสามารถทำงานเป็น co-processor กับหน่วยประมวลผลกลางของเครื่องไมโครคอมพิวเตอร์ได้

# Digital Signal Processing TMS320C25


| | | |
|---|---|---|
| Mr.Nuttapong | Imbumrung | 32.1141 |
| Mr. Likit | Thawatchaiwirut | 32.1269 |
| Mr. Wasan | Sontipermpoon | 32.1287 |
| Mr. Surasit | Wannakrairoj | Advisor |



**Abtract**

Digital Signal Processing involves the representation, transmittion and manipulation of signal using numerical techniques and digital processor. It has been an exiting and growing technology during the past few years. Its application have also been expaned vigorously to encompass not only the tradition radar signal processing but also today's digital audio processing.

This thesis is about theory in Digital Signal Processing and design of DSP Card using TMS320C25 Digital Signal Processor. This Card will be used in motor control system. It will control Velocity and Position of four AC Servo Motor.

# สารบัญ

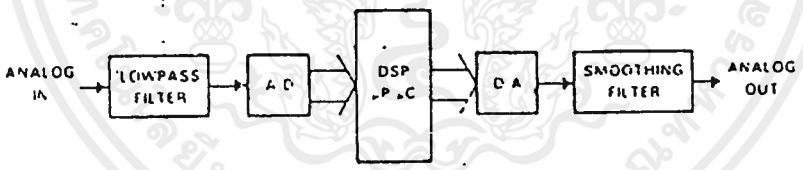
# สารบัญ

# บทที่ 1

# บทนำ

Digital Signal Processing เป็นการเปลี่ยนรูปแบบของสัญญาณโดยใช้การคำนวณทางคณิตศาสตร์กับสัญญาณเชิงเลข (Digital)ของสัญญาณอินพุท(Input)ด้วยดิจิตอลชิกแนลโปรเซสเซอร์ ดังนั้นจึงทำให้เราสามารถได้สัญญาณเอาท์พุท(Output)ของวงจรใด ๆได้ เพียงแค่เรานำสัญญาณอินพุท(input)ผ่านการProcessด้วยสมการของวงจรนั้น ๆ เท่านั้น

การที่ Digital Signal Processing ได้รับความสนใจมากขึ้นเนื่องมาจากเทคโนโลยีในการสร้างวงจรรวม(Integrated Circuit)มีประสิทธิภาพดีขึ้นมาก วงจรขนาดใหญ่สามารถนำมาสร้างเป็นวงจรรวมที่ผลิตบนสารกึ่งตัวนำชิ้นเดียวกันได้และผลิตได้ครั้งละจำนวนมาก ๆ จึงทำให้ราคาถูกลงเป็นอย่างมาก นอกจากนี้การประมวลเชิงเลข (Digital Signal Processing) ยังให้ความแม่นยำ (Accuracy)และความเชื่อถือ (Reliablility) มากกว่าการประมวลสัญญาณเชิงอุปมานมาก โดยทั่วไประบบ DSP จะประกอบด้วยส่วนต่างๆ ดังรูป




รูปที่ 1.1 ระบบ DSP

LOWPASS FILTER มีไว้เพื่อจำกัดช่วงความถี่ของสัญญาณอินพุทจากนั้นสัญญาณอินพุทจะถูกแปลงจาก Analog ให้เป็น Digital โดยวงจร A/D (Analog to Digital Converter) จากนั้นระบบประมวลผลสัญญาณดิจิตอลก็จะนำเอาอัลกอริทึ่ม (Algorithm)ที่มีอยู่มากระทำกับข้อมูลสัญญาณแล้วทำการส่งข้อมูลสัญญาณที่ถูกเปลี่ยนแปลงแล้วสู่ส่วน D/A(Digital-

to Analog Converter) เพื่อเปลี่ยนสัญญาณข้อมูลซึ่งเป็น Digital ให้เป็นสัญญาณ Analog ตามเดิม

ในปัจจุบันมีการนำ DSPไปประยุกต์ในงานต่างๆ มากมายซึ่งสามารถจำแนกออกเป็นประเภทต่าง ๆ ดังนี้

1) การประยุกต์ในงานทั่ว ๆ ไปได้แก่ Digital filter, Adaptive filtering, Wareform generation

2) งานด้านกราฟฟิค (Graphic/Image) ได้แก่ งานด้านการส่งภาพ(image-transimission),การเข้ารหัสภาพ(Image coding หรือCompression),การจำรูปแบบ (Pattern recognition)

3) งานด้านการวัดและเครื่องมือวัด ได้แก่การวิเคราะห์สเปคตรัม(Spectrum-analysis),เครื่องกำเนิดสัญญาณรูปแบบต่างๆ(Function gerneration),การวิเคราะห์แผ่นดินไหว (Seismic Precessing)

4) การประยุกต์ทางด้านเสียงพูด(Voice/Speech)ได้แก่การจำ(recognition) การยืนยัน (Verification) ,การเพิ่มคุณภาพ (Enhancement), การสังเคราะห์(Synthesis)รวมถึงการเปลี่ยนแปลงข้อความ (Text)เป็นเสียงพูด (Speech)

5) ทางด้านระบบควบคุม (Control) ได้แก่ ระบบควบคุมหุ่นยนต์(Robotic), การควบคุมเครื่องจักรกลไฟฟ้าต่าง ๆ ระบบเซอร์โว หรือพวกเฟสล็อคลูพ (PLL)

6) ทางด้านทหาร (Millitary) ได้แก่ การสื่อสารที่เกี่ยวกับความมั่นคง (Secure communication),ระบบเรโซนา,ระบบนำร่อง(Navigation),ระบบนำวิถี (Missile guidance)

7) ทางระบบโทรคมนาคม(Telecommunieation)ได้แก่การกำจัดเสียงสะท้อน (Echo cancellation),ระบบสื่อสารดิจิตอลโมเด็ม (Modem), โทรศัพท์เคลื่อนที่ (Cellular telephones)

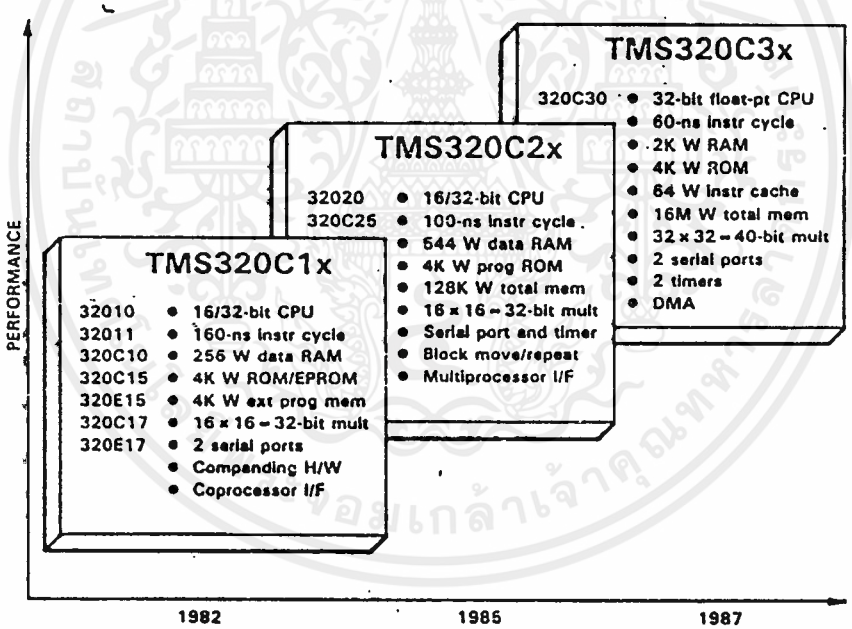
8) ทางด้านการแพทย์ (Medical) ได้แก่ เครื่องช่วยการได้ยิน เครื่องมือวินิจฉัยต่าง ๆ

9) ทางด้านอุตสาหกรรม (Industrial) ได้แก่ เครื่องควบคุมเครื่องยนต์

(Engine control),ระบบเสียงหรือโทรทัศน์ดิจิตอล (Digital audio /TV),เครื่องมือสังเคราะห์เสียงดนตรี (Music Synthesizer) เครื่องเจาะหรือตัดแบบควบคุมด้วยตัวเลข (Numerical control)

ดิจิตอลซิกแนลโปรเซสเซอร์ จำเป็นจะต้องมีความเร็วสูงที่ถูกออกแบบมาโดยเฉพาะให้สามารถทำงานได้อย่างมีประสิทธิภาพโดยใช้ข้อดีของการประมวลผลแบบขนานและการใช้ชุดคำสั่งที่สร้างขึ้นมาเฉพาะทั้งนี้ก็เพื่อการประมวลแบบเวลาจริง(Real Time) ซึ่งหมายถึงเมื่อเราใส่สัญญาณอินพุทเข้าไปก็จะได้สัญญาณเอาท์พุทออกมาเลย โดยจะมีเวลาหน่วง (delay time) น้อยมาก ซึ่งจะมีประโยชน์ในการใช้งานมากกว่าการประมวลผลแบบไม่ใช่เวลาจริง

บริษัทเท็กซัสอินสตรูเมนต์ ได้ผลิตไอซีที่เป็นดิจิตอลซิกแนลโปรเซสเซอร์ ซึ่งเป็นตระกูลที่มีชื่อว่า TMS320 ซึ่งมีวิวัฒนาการดังรูป




รูปที่ 1.2 วิวัฒนาการของไอซีตระกูลTMS320C25

ซึ่งในงานปริญญานิพนธ์นี้ เราจะใช้ TMS320C25 มาสร้างเป็นระบบDSPเพื่อประยุกต์ใช้ในการควบคุมมอเตอร์ เพื่อสร้าง CNC 3มิติ (3dimensional Computerized Numerical Control)

# บทที่ 2

## สถาปัตยกรรมของTMS320C25

TMS320C25 ถูกออกแบบโดย Harvard-Type Architecture ชุดคำสั่งของตัวมันจะมีความ เร็วและความยืดหยุ่น ทำให้สามารถทำการ execute ได้ถึง 10 ล้านคำสั่งใน 1 วินาที

TMS320C25 ผลิตโดยใช้ CMOS Technology สามารถเอ็กซีคิวต์คำสั่งหลายๆคำสั่งภายในเวลา 1ไซเคิล หรือ 100 นาโนวินาที TMS320C25 มีลักษณะภายนอกโดยทั่วไปและคำสั่งต่างๆคล้ายกับของ TMS32020 แต่เบอร์ C25 จะมีคำสั่งเพิ่มขึ้น 22 คำสั่ง(ทั้งหมด 133 คำสั่ง),ออกซิลารีรีจิสเตอร์เพิ่มขึ้น 8 ตัว,มีฮาร์ดแวร์สแตก 8 ระดับ(eight-level hardware stack),มีโปรแกรมรอมบนชิป(on-chip program ROM) และ มี a bit-reversed indexed-address mode

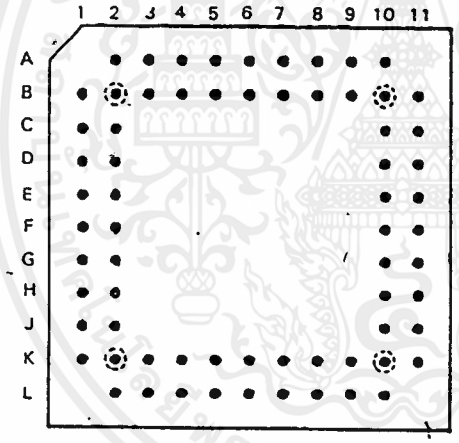
ลักษณะสำคัญ

- มี instruction cycle timing = 200 nsec.
- มี programmable on-chip data RAM = 544 word
- มี on-chip program ROM 4 Kword
- มี data memory และ program memory รวมกันเท่ากับ 128 Kword
- มี 32-bit ALU/Accumulator
- มี 16x16 bit parallel multiplier ซึ่งจะทำให้ได้ผลลัพธ์ 32 bit
- คำสั่งในการคูณและการคำนวณใช้เวลาเพียง 1 cycle
- มี on-chip timer สำหรับการควบคุมการปฏิบัติการ
- มี 16 inputs และ 16 outputs chanel
- มี 16 bit parallel shifter
- มี serial port
- สามารถทำงานแบบ multiprocessor ได้อย่างสอดคล้อง
- สามารถ compat กับ source code ของTMS320C1x ขึ้นไป

- สามารถทำ DMA โดยใช้ประโยชน์จากขา hold
- มี bit-reversed indexed-addressing mode สำหรับ 2-FFTs
- มี on-chip clock generator
- มี package แบบ 64 lead PLCC

<u>รายละเอียดของขาและสัญญาณของTMS320c25</u>

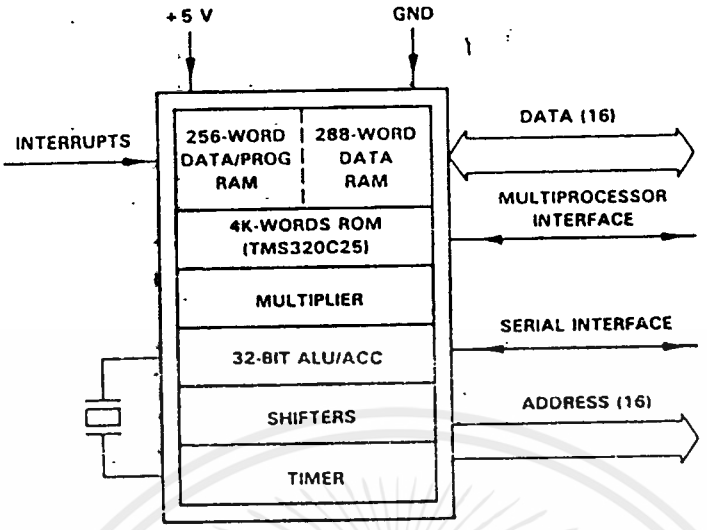
TMS ตระกูล 320 เป็นดิจิตอลซิกแนลโปรเซสเซอร์ ซึ่งมี package แบบ 68-pin grid array (PGA) TMS320C25 มี package แบบ 68-pin plastic -leaded chip carrier (PLCC)ซึ่งจะมีลักษณะ ดังรูป




รูปที่ 3.1 Top View Pin Grid Array

รายละเอียดของขาต่างๆของ TMS320c25

การออกแบบสถาปัตยกรรมของTMS320C25 คำนึงถึงในเรื่องความเร็วของระบบ,ก ความยืดหยุ่นเป็นสำคัญ




รูปที่ 3.2 TMS 320C2x Simplified Block Diagram

TMS320c25 ถูกออกแบบมาเพื่อทำให้เกิดประสิทธิภาพในการใช้โดยการแยก data memory และ program memory อย่างเด็ดขาดโดยระบบบัส เพื่อให้เกิดความเร็วสูงสุดในการExecute On-chip data RAM ขนาดใหญ่ 2 Block ดังรูป ช่วยให้เกิดความยืดหยุ่นในการออกแบบระบบ หนึ่งในสองบล็อกสามารถเลือกได้ว่าจะให้เป็น Program memory หรือ Data memory นอกจากนี้ยังมี data memory ซึ่งเป็น RAM ภายนอกอีก 64K ซึ่งจะทำให้การimplement เป็นไปได้โดยสะดวก 4K-word On-chip ROM ใช้ในการเก็บโปรแกรมที่มีขนาดไม่เกิน 4K-word นอกจากนี้ยังมีหน่วยความจำโปรแกรมภายนอกอีก 64K-wordด้วย โปรแกรมขนาดใหญ่สามารถเอ็กซีคิวต์ด้วยความเร็วสูงสุดโดยสามารถ downloaded จากหน่วยความจำภายนอกซึ่งทำงานช้าเข้าสู่ on-chip RAM เพื่อการดำเนินงานอย่างมีประสิทธิภาพ

TMS320c25 ปฏิบัติการคำนวณแบบ 2's complememt โดยการใช้ 32-bit ALU เป็นหน่วยคำนวณทางคณิตศาสตร์โดยใช้ข้อมูล 16 บิทจาก data RAM หรือจากคำสั่ง หรือจากการใช้ผลลัพธ์ขนาด 32 bitของ multiplier ซึ่งถูกเก็บอยู่ใน product register

ALU สามารถปฏิบัติตาม Boolean Operations โดยความสามารถในการโยกย้ายบิท ต้องการคอนโทรลเลอร์ซึ่งมีความเร็งสูง accumulatorเก็บเอาท์พุทจาก ALU และเป็นอินพุทตัวที่2 ของ ALU

Accumulatorมีความยาว 32-bitและถูกแบ่งเป็น High-orderและLow-order accumulator word ในหน่วยความจำ

Multiplier ทำการคูณ 16x16bit ด้วยวิธี 2's complement ซึ่งจะได้ผลลัพธ์เป็นเลข 32 บิทใน 1 cycle ของคำสั่ง

Multiplier ประกอบด้วย

- T REGISTER
- P REGISTER
- MULTIPLIER ARRAY

T REGISTER มีขนาด 16 บิทใช้สำหรับเก็บตัวตั้งสำหรับการคูณชั่วคราว

P REGISTER ใช้เก็บผลคูณขนาด 32 บิท

ค่าของตัวคูณอาจนำมาจากหน่วยความจำข้อมูล,จากหน่วยความจำโปรแกรมเมื่อใช้คำสั่ง MAC/MACD หรือได้รับโดยทันทีจากคำสั่ง MPYK (Multiply Immediately) Multiplier ซึ่งอยู่บนชิปยอมให้อุปกรณ์ทำ ปฏิบัติการพื้นฐานทางดิจิตอลซิกแนลโปรเซสซิ่งได้ เช่น Convolution,Correlation และ Filtering Scaling ShifterของTMS320c25 มีอินพุท 16 บิทต่อกับ data bus และเอาท์พุท 32 บิทต่อกับ ALU Scaling Shifter สามารถชิพท์อินพุทดาต้าไปทางซ้ายได้ 0-16 บิท ตามคำสั่งซึ่งถูกโปรแกรมไว้ LSBs ของเอาท์พุทจะถูกเติมด้วย 0s และ MSBsอาจจะถูกเติมด้วย 0s หรือเครื่องหมายขยาย ขึ้นอยู่กับสถานะของ Sign-extension mode bitของ Status register ST1 ความสามารถในการชิพท์ทำให้โปรเซสเซอร์สามารถทำการ Numerical scaling,bit extraction,Extenced arithmetic และOverflow prevention ได้

TMS320c25 มี Data bus ซึ่งเป็นบัสแบบขนานขนาด 16 บิท ($D_{15}$-$D_0$) และมีAddress bus($A_{15}$ - $A_0$)พร้อมทั้งมีสัญญาณควบคุมสำหรับเลือก Data memory ,Program memory และ I/O Space(DS,PS,IS)และสัญญาณควบคุมต่างๆไว้สำหรับใช้อินเตอร์เฟสหน่วยความจำภายนอก

สัญญาณ R/~W ใช้ควบคุมทิศทางการเคลื่อนย้ายข้อมูลและสัญญาณ ~STRB จะเตรียมสัญญาณจังหวะในการควบคุมการเคลื่อนย้ายข้อมูลเมื่อใช้ Program Ram หรือ ROM ซึ่งอยู่บนชิปหรือ High speed external memory TMS320c25 จะทำงานด้วยความเร็วเต็มที่โดยปราศจากสภาวะรอ (Wait state) สัญญาณ READY ใช้ในการสร้างสภาวะรอสำหรับการติดต่อกับหน่วยความจำภายนอกที่ช้ากว่า

ฮาร์ดแวร์สแตก 8 ระดับมีไว้เพื่อเก็บค่าโปรแกรมเคาน์เตอร์ระหว่างการเกิดอินเตอร์รัพท์ และการเรียกใช้ Subroutine

การควบคุมของTMS320c25 ได้รับการสนับสนุนจาก On-chip memory -mapped,16 bit timer, Repeat counter, 3 External maskable user-interrupts และInternal interupts ซึ่งเกิดจากการกระทำของSerial port หรือเกิดจาก Timer

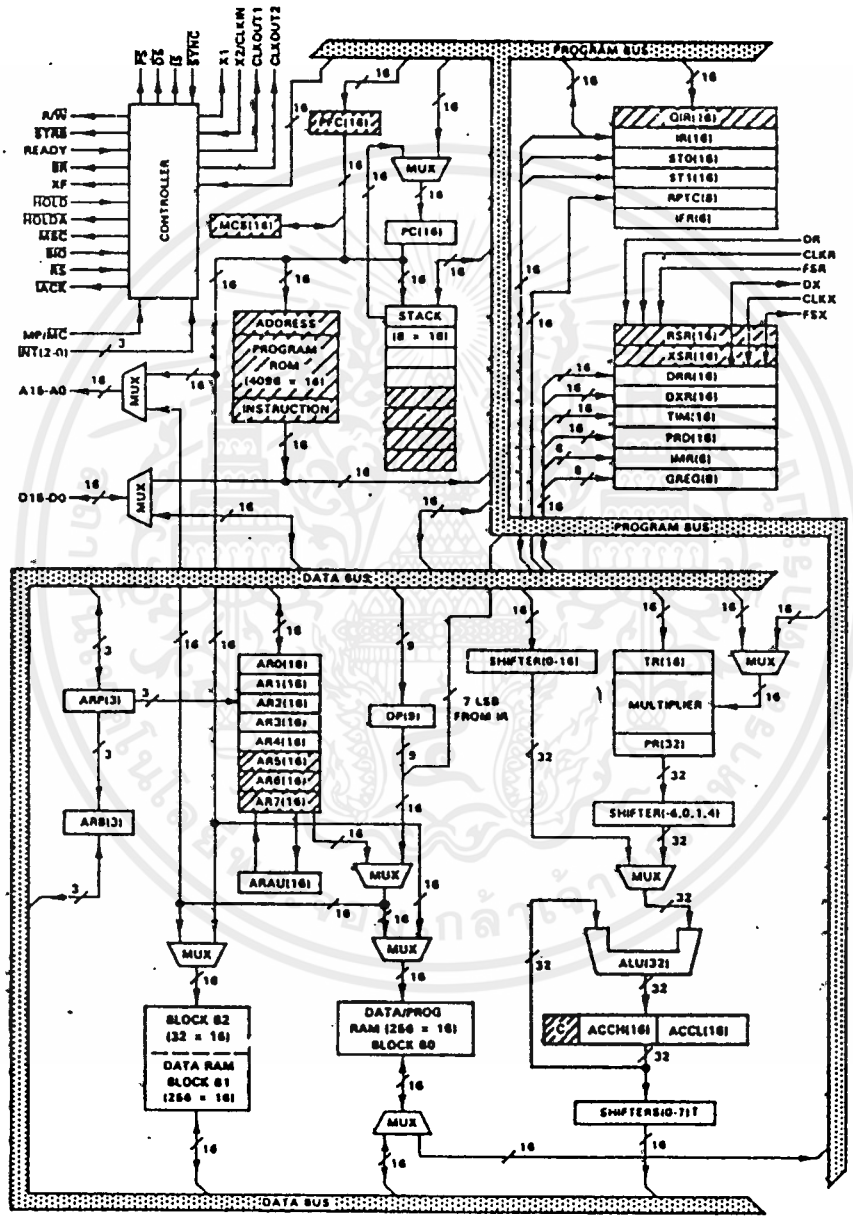
Full-Duplex Serial Portบนชิปใช้สำหรับการติดต่อโดยตรงกับอุปกรณ์แบบอนุกรม เช่น Codec Serial A/D Converter เป็นต้น

2 Serial Port Memory-mapped Registers (Data transmit/ Recieve register)อาจถูกปฏิบัติการในโหมด 8 bit byte หรือ โหมด 16 bit word ก็ได้ รีจิสเตอร์แต่ละตัวมี Clock input จากภายนอก,Framing synchron-ization inputและมีเกี่ยวเนื่องกับ Shift register

การสื่อสารแบบอนุกรมสามารถถูกใช้ระหว่างโปรเซสเซอร์แต่ละตัวในการใช้งานแบบมัลติโปรเซสซิ่ง TMS320c25 มีความสามารถในการกำหนด Global data memory space และการติดต่อกับหน่วยความจำดังกล่าวโดยใช้สัญญาณ ~BR (Bus Request) และ READY เป็นสัญญาณควบคุม

TMS320c25สนับสนุนการทำ DMA(Direct Memory Access) กับหน่วยความจำโปรแกรมและหน่วยความจำข้อมูลภายนอก ด้วยการใช้ขา ~HOLD และ ~HOLDA โปรเซสเซอร์ตัวอื่นสามารถเข้าควบคุม Memory ภายนอกได้อย่างสมบูรณ์ ด้วยการทำ

ให้ขา ~HOLD มีสถานะเป็น Low(0) ทำนี้เนื่องจาก TMS320c25 จะทำให้ Address bus,Data bus และ Control lines เป็น High-impedance การให้สัญญาณระหว่างโปรเซสเซอร์ตัวอื่นกับTMS320c25 สามารถทำได้โดยใช้ interrupts ซึ่งมีอยู่ 2 โหมด




รูปที่ 3.3 Functional Block Diagram

Block Diagram ดังรูปแสดงขาต่างๆที่ใช้ในการอินเตอร์เฟสของTMSด้วย โดยบล็อกที่มีการระบายนั้นจะมีเฉพาะใน TMS320c25

สถาปัตยกรรมของ TMS320c25 ถูกสร้างโดยรอบบัสทั้งสองนั่นคือ Program bus และ Data bus Program bus ทำหน้าที่นำโค๊ดคำสั่ง และ โอเปอร์แรนด์จากโปรแกรมเมมโมรี ส่วน Data bus เป็นตัวเชื่อมการติดต่อกับส่วนต่างๆได้แก่ Central Arithmetic Logic Unit(CALU)และ Auxiliary Register file กับData RAM Program bus และ Data bus สามารถนำข้อมูลจากData RAMบนชิปและหน่วยความจำโปรแกรมภายนอกหรือภายใน ไปยังมัลติพลายเออร์เพื่อทำการคูณหรือการคำนวณได้ภายในไทม์มิง 1 Cycle

TMS320c25 มีความสามารถสูงในการทำงานแบบขนาน เช่น ขณะที่ข้อมูลกำลังถูกปฏิบัติการโดย CALU อยู่ อาจเกิดการคำนวณทางคณิตศาสตร์ใน Auxiliary Register Arithmetic Unit (ARAU) ด้วย ดังนั้น การทำงานแบบขนานจะมีผลทำให้มีความสามารถทางคณิตศาสตร์, ทางตรรกและการโยกย้ายบิท สามารถถูกปฏิบัติในเวลาเพียง 1 Cycleเท่านั้น

MEMORY ORGANIZATION

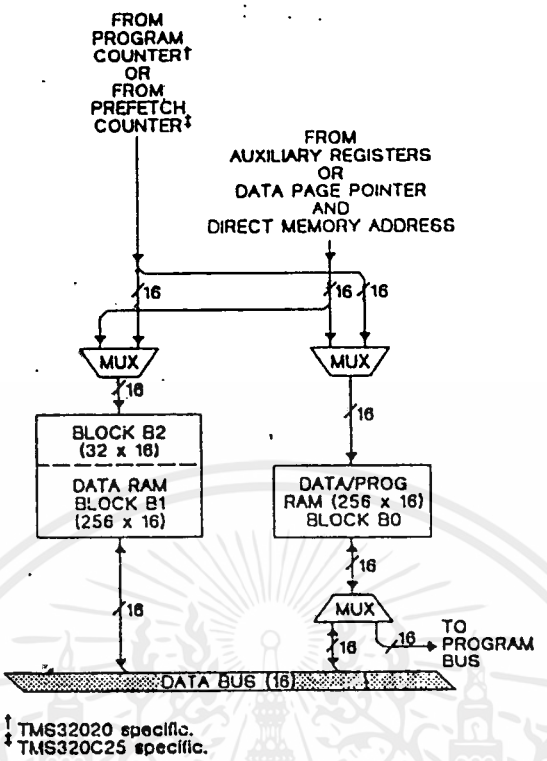
TMS320c25 มี Data RAM บนชิปรวมทั้งหมด 544×16-bit(words)จากจำนวนดังกล่าว 288 Words เป็นหน่วยความจำข้อมูลเสมอส่วนที่เหลืออีก 256 Words สามารถกำหนดให้เป็นหน่วยความจำโปรแกรมหรือหน่วยความจำข้อมูลก็ได้นอกจากนี้ยังมี Maskable program ROM ขนาด 4K อีกด้วย

**1) Data Memory**

544 words ของ data RAM บนชิปถูกแบ่งออกเป็น 3 บล็อก(B0, B1และB2) ดังรูปที่ 3.4

บล็อก B0 256 words สามารถกำหนดให้เป็นหน่วยความจำโปรแกรมหรือหน่วยความจำข้อมูลได้โดยใช้คำสั่งกำหนดตามแต่ละจุดประสงค์การใช้งาน

ส่วนบล็อก B1และB2 288wordsเป็นหน่วยความจำข้อมูลเสมอหน่วยความจำข้อมูลขนาด 544words ยอมให้TMS320c25 จัดการเก็บเรียงข้อมูลได้ขนาด 512Words




รูปที่ 3.4 On-chip Data Memory

(256 Words หาก on-chip RAM ถูกใช้เป็นหน่วยความจำโปรแกรม)ส่วนที่เหลือ 32 Wordsใช้สำหรับการเก็บโดยตรง (Immediate storage)

TMS320c25 สามารถอ้างอิงหน่วยความจำข้อมูลได้ทั้งหมด 64Kwords หากเราใช้หน่วยความจำข้อมูลบนชิปด้วยตำแหน่งของมันจะถูก Map ลงในตำแหน่งที่ต่ำกว่า 1 Kwordsของ Data memory space และหน่วยความจำข้อมูลสามารถขยายเพิ่มขึ้นโดยตรงจนมีขนาด 64 Kwords ขณะที่ยังคงทำงานด้วยความเร็วเต็มที่

2) Program Memory

โปรแกรมแรม,รอมบนชิปหรือหน่วยความจำภายนอกความเร็วสูงสามารถถูกใช้ด้วยความเร็วเต็มที่โดยไม่มีสภาวะรอ(wait state) หรือมิฉะนั้นขา READY ก็สามารถติดต่อกับTMS320c25ให้ช้าลงหากใช้หน่วยความจำภายนอกที่ทำงานช้า

TMS320c25สามารถมีหน่วยความจำโปรแกรมได้ทั้งหมด 64 Kwords โดย RAM ภายใน(บล็อก B0)สามารถถูกกำหนดให้เป็นหน่วยความจำโปรแกรมได้โดยการใช้คำสั่งตามวัตถุประสงค์ การเอ็กซีคิวต์จากบล็อกB0นี้สามารถถูกริเร่มได้หลังจาก Memory space ถูกกำหนดใหม่

หาก TMS320c25 ถูกใช้ร่วมกับ On-chip program ROM ขนาด 4Kwords ซึ่งสามารถถูกโปรแกรมจากโรงงานด้วยโปรแกรมของลูกค้าเองได้ On-chip ROM ทำให้ทำการเอ็กซีคิวต์โปรแกรมได้ด้วยความเร็วเต็มที่ โดยไม่ต้องการหน่วยความจำโปรแกรมภายนอกความเร็วสูง(High Speed External Program Memory) การใช้หน่วยความจำส่วนนี้ยังยอมให้บัสข้อมูลภายนอกเป็นอิสระ เพื่อการแอคเซสหน่วยความจำข้อมูลภายนอก

การ Map ของตำแหน่ง 4Kwords แรก ว่าจะเป็นหน่วยความจำโปรแกรมภายนอกชิปหรือบนชิปสามารถเลือกได้โดยuserจากขา MP/~MC (Microprocessor/Microcomputer) บน TMS320c25 หากที่ขาดังกล่าวมีสถานะเป็น High ตำแหน่ง 4Kwordsแรกเป็นหน่วยความจำนอกชิปถ้ามีสถานะเป็น Low ตำแหน่ง 4Kwordsแรกเป็น On-chip ROM

3)Memory Maps

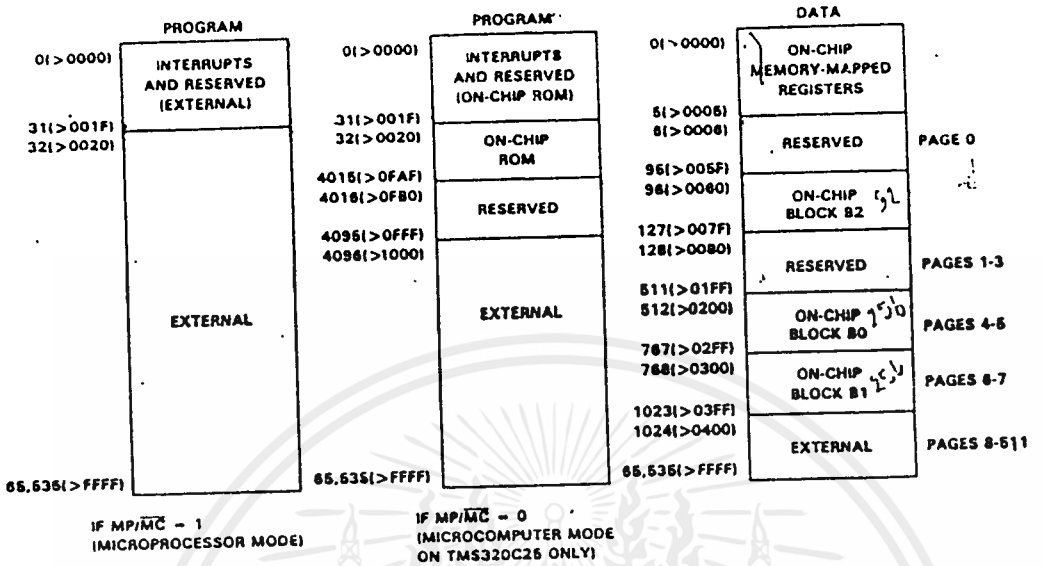
TMS320c25มีแอดเดรสสเปซของหน่วยความจำโปรแกรม,หน่วยความจำข้อมูลและI/O แยกกันซึ่งแสดงดังรูปที่ 3.5

สัญญาณ ~PS,~DS,~IS,~STRB จะแอคทีฟเมื่อมีการอ้างแอดเดรสหน่วยความจำภายนอก ในระหว่าง Internal addressing cycle สัญญาณเหล่านี้จะยังไม่แอคทีฟ(High) เพื่อป้องกันการชัดแย้งในการอ้างแอดเดรสหน่วยความจำ เช่น เมื่อบล็อก B0 ถูกำหนดให้เป็นหน่วยความจำโปรแกรม

หน่วยความจำที่อยู่บนชิปบล็อกB0,B1และB2มีเนื้อที่รวมกันทั้งสิ้น 544Kwords -RAM โปรแกรม/ดาต้าแรม บล็อก B0(256Kwords) อยู่ในpage 4และ5 ของ Data memory mapเมื่อกำหนดให้เป็น data RAM และที่แอดเดรส FF00ถึง FFFFเมื่อกำหนดให้เป็น Program memory

บล็อก B1(เป็นหน่วยความจำข้อมูลเสมอ)อยู่ในpage 6และ 7

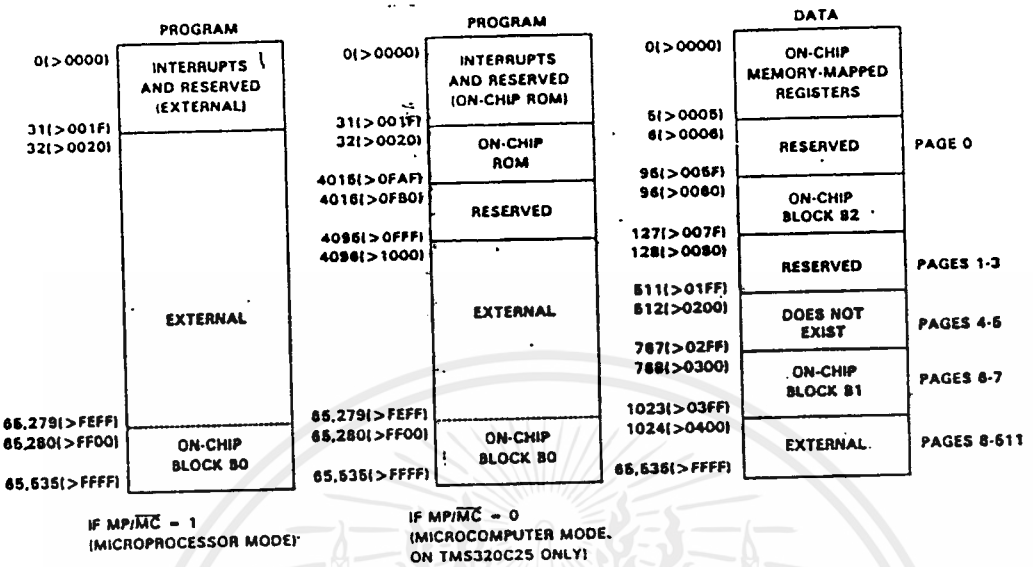
บล็อก B2อยู่ในตำแหน่งที่มากกว่า 32 wordsขึ้นไปของpage 0.




รูปที่3.5(1) Memory Map After CNFD Instruction

Note เนื้อที่ที่เหลือในpage 0 เป็นของ memory-mapped registerและตำแหน่งที่ถูกสงวนไว้ภายใน และpage 1-3ของ data memory map มีตำแหน่งที่ถูกสงวนไว้ภายในด้วยเช่นกัน ซึ่งตำแหน่งดังกล่าวอาจไม่ถูกใช้ในการเก็บ สิ่งที่อยู่ภายในจะเป็นundefinedเมื่อถูกอ่าน

On-chip RAM จะถูก Map เข้าไปใน 64K Word Data memory หรือ Program memory ก็ได้ขึ้นอยู่กับการกำหนดลักษณะของหน่วยความจำ คำสั่ง CNFD/CNFP ถูกใช้ในการกำหนดบล็อก B0 ให้เป็นData/Program memory ตามลำดับคำสั่ง BLKP(Block move from program memory to data memory)อาจถูกใช้ในการดาวน์โหลดโปรแกรมอินฟอร์เมชันไปที่บล็อกB0เมื่อมันถูกกำหนดให้เป็น Data RAM ดังนั้นคำสั่ง CNFP(Configure block as program memory)อาจถูกใช้เพื่อเปลี่ยนบล็อกB0ให้เป็น Program RAM Userอาจยังคงเอ็กซีคิวต์จากหน่วยความจำโปรแกรมภายนอกโดยไม่คำนึงถึงการกำหนดดังกล่าว ขณะที่มีการแอคเซสหน่วยความจำโปรแกรมภายใน




รูปที่ 3.5 (2) Memory Map After a CNFP Instruction

สัญญาณควบคุมภายในยังคงไม่แอคทีฟ

การรีเซ็ทเป็นการกำหนดให้บล็อก B0เป็น Data RAMเนื่องจากการไปป์ไลน์ภายในเมื่อคำสั่ง CNFD/CNFPถูกใช้เพื่อ Mapบล็อกB0ใหม่ มีเวลาหน่วงก่อนที่การกำหนดในครั้งใหม่นี้จะมีผล เวลาหน่วงนี้เท่ากับ 1 เฟทช์ไซเคิลถ้าการเอ็กซีคิวต์มาจาก Program RAM ภายใน TMS320c25 มีเวลาหน่วง 2 เฟทช์ไซเคิลถ้าการเอ็กซีคิวต์มาจาก ROM หรือหน่วยความจำภายนอก สิ่งนี้มีความสำคัญโดยเฉพาะถ้าการเอ็กซีคิวต์โปรแกรมมาจากตำแหน่งบริเวณ FF00 ฉะนั้นคำสั่ง CNFP ต้องอยู่ที่ตำแหน่ง FEFDในหน่วยความจำภายนอกถ้าการเอ็กซีคิวต์ต้องดำเนินต่อมาจากตำแหน่งแรกในบล็อก B0 ถ้า CNFPอยู่ตำแหน่ง FEFD และคำสั่งที่อยู่ตำแหน่งFEFF เป็นคำสั่งที่มีความยาว 2Word เวิร์ดที่สองของคำสั่งจะถูกเฟทช์จากตำแหน่งแรกในบล็อก B0 ถ้าการเอ็กซีคิวต์มาจากตำแหน่งที่สูงกว่าFF00และบล็อก B0 ถูกกำหนดใหม่จึงต้องมีการดูแลเพื่อประกันว่าการเอ็กซีคิวต์กลับเข้าทำงานในจุดที่เหมาะสมในการกำหนด (Configeration) ครั้งใหม่

โปรแกรมรอมบนชิปTMS320c25อยู่ในตำแหน่งที่น้อยกว่า 4Kwordsของหน่วยความจำโปรแกรมเมื่อถูกเลือกโดยการทำให้ขา MP/~MC มีสถานะเป็นศูนย์เมื่อMP/~MC เป็นหนึ่ง ตำแหน่งที่ต่ำกว่า 4Kwordsของหน่วยความจำโปรแกรมจะเป็นของส่วนภายนอก

4) Memory-Mapped Register

ข้อมูลของรีจิสเตอร์ 6 ตัวซึ่งถูก mapให้อยู่บนเนื้อที่ของหน่วยความจำข้อมูลแสดงดังตารางโดยMemory-mapped registerอาจถูกแอดเซสด้วยวิธีเดียวกันกับการแอดเซสตำแหน่งของหน่วยความจำข้อมูลใดๆ ยกเว้นการใช้คำสั่ง BLKD(Block move from data memory to data memory) ไม่สามารถปฏิบัติจาก Memory-mapped registerได้

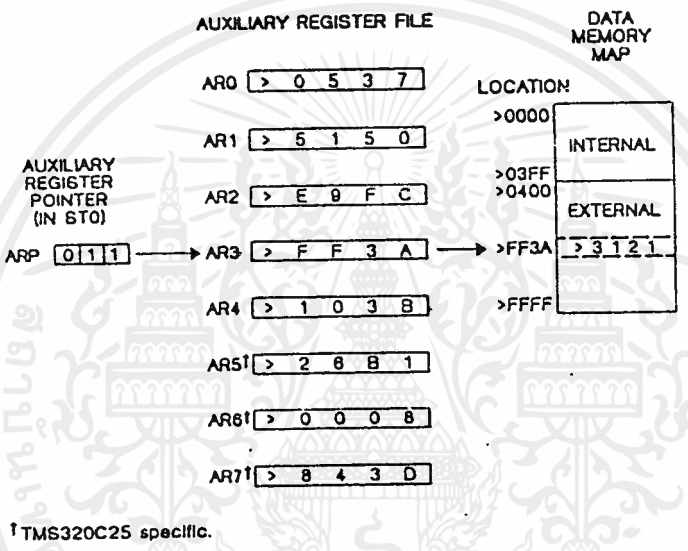
| Register Name | ADDRESS LOCATION | DEFINITION |
|---|---|---|
| DRR(15-0) | 0 | Serial port data recieve register |
| DXR(15-0) | 1 | Serial port data transmit register |
| TIM(15-0) | 2 | Timer register |
| PRD(15-0) | 3 | Period register |
| IMR(5-0) | 4 | Interrupt mask register |
| GREG(7-0) | 5 | Global memory allocation register |

5) Auxiliary Registers

TMS320c25 มีรีจิสเตอร์ไฟล์ที่มีออกซิลิอารีรีจิสเตอร์จำนวน 8 ตัวบรรจุอยู่ (AR0-AR7)

ออกซิลิอารีรีจิสเตอร์อาจถูกใช้สำหรับการอ้างแอดเดรสของหน่วยความจำทางอ้อม หรือใช้เพื่อการเก็บข้อมูลชั่วคราว การอ้างแอดเดรสของออกซิลิอารีรีจิสเตอร์

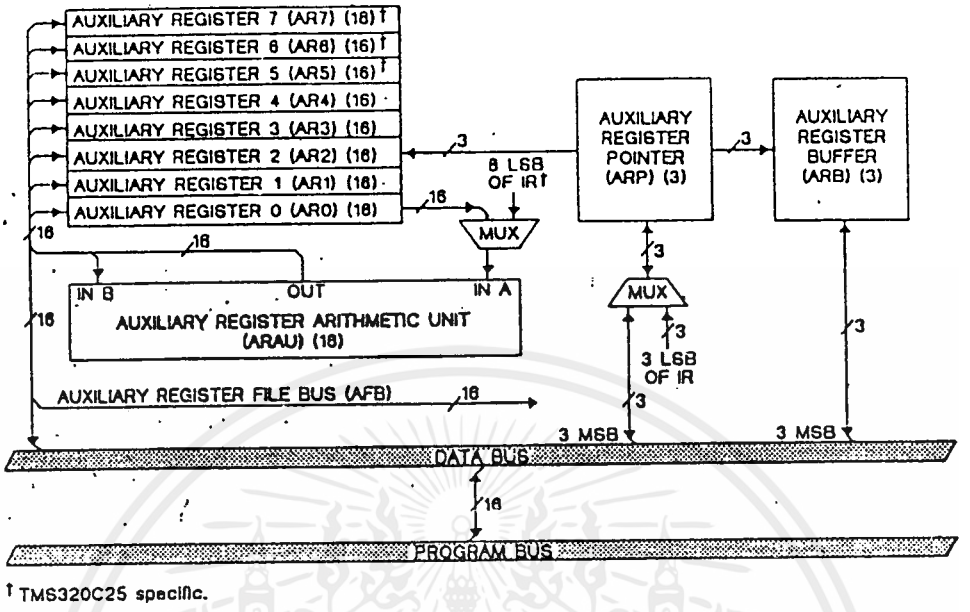
โดยทางอ้อมนั้นยอมให้มีการเก็บค่าแอดเดรสของหน่วยความจำข้อมูลของโอเปอร์แรนด์ของค่าสั่ง ลงในตัวใดตัวหนึ่งใน 8 ตัวได้ รีจิสเตอร์เหล่านี้ถูกชี้โดยค่า Auxiliary register pointer(ARP) ที่เป็นเลขขนาด 3bit มีค่าตั้งแต่ 0ถึง 7 ซึ่งใช้เป็นตัวระบุชี้ AR0 จนถึง AR7 ตามลำดับออกซิลิอารีรีจิสเตอร์และ ARP อาจถูกโหลดมาจากหน่วยความจำข้อมูลหรือจากการกำหนดค่าโอเปอร์แรนด์โดยตรงจากค่าสั่ง




รูปที่ 3.6 Indirect Register Addressing Examples

ออกซิลิอารีรีจิสเตอร์ไฟล์ (AR0-AR7 บน TMS320c25) ถูกต่อกับ Auxiliary Register Arithmetic Unit(ARAU) แสดงได้ดังรูป

จากรูปที่3.6 AR0หรือ 8LSBsของรีจิสเตอร์คำสั่ง(IR) ถูกต่อเข้ากับอินพุทขาหนึ่งของ ARAU ส่วนอินพุทอีกอันหนึ่งมาจาก Current Auxiliary Register (หมายถึงออกซิลิอารีรีจิสเตอร์ที่กำลังถูกชี้)




รูปที่ 3.7 Auxiliary Register File

AR(ARP)หมายถึง เลขที่อยู่ภายในcurrent AR ซึ่งถูกชี้โดยARP

ARAU ปฏิบัติตามฟังก์ชันต่างๆ ดังนี้

| | |
|---|---|
| AR(ARP)+AR0 -> AR(ARP) | Index the current AR by adding a 16-bit integer contained in AR0. |
| AR(ARP)-AR0 -> AR(ARP) | Index the current AR by substracting a 16-bit integer contained in AR0. |
| AR(ARP)+1 -> AR(ARP) | Increment the current AR by one. |
| AR(ARP)-1 -> AR(ARP) | Decremenr the current AR by one. |
| AR(ARP) -> AR(ARP) | AR(ARP) is unchanged. |
| AR(ARP)+IR(7-0)->AR(ARP) | Add 8-bit immediate value to the current AR. |

| | |
|---|---|
| AR(ARP)-IR(7-0)->AR(ARP) | Substract 8-bit immediate value from the current AR. |
| AR(ARP)+rcAR0 -> AR(ARP) | Bit-reversed indexing, add AR0 with reverse-carry(rc) propagation |
| AR(ARP)-rcAR0 -> AR(ARP) | Bit-reversed indexing, substract AR0 with reverse-carry(rc) propagation. |

แม้ว่า ARAU มีประโยชน์ในการถ่ายเทแอดเดรสแบบขนานกับการปฏิบัติการอื่นๆมันอาจบริการเหมือนหน่วยคณิตศาสตร์พิเศษหน่วยหนึ่งเพราะว่าออกซิลิอารีรีจิสเตอร์สามารถสื่อสารกับหน่วยความจำข้อมูลได้โดยตรง ARAUO ทำให้ผลลัพธ์เป็นเลขขนาด16 บิทที่ไม่มีเครื่องหมาย ในขณะที่ CALU ทำให้ผลลัพธ์เป็นเลขขนาด 32 บิทแบบทูคอมพลีเมนต์ (2's complement) คำสั่งที่มีไว้ใช้ในการบรานช์ขึ้นอยู่กับการเปรียบเทียบของออกซิลิอารีรีจิสเตอร์ที่ถูกชี้โดย ARP กับ AR0 คำสั่ง BANZ ยอมให้ออกซิลิอารีรีจิสเตอร์ถูกใช้งานเป็น Loop counter ได้ด้วย

6) Memory Addressing Mode

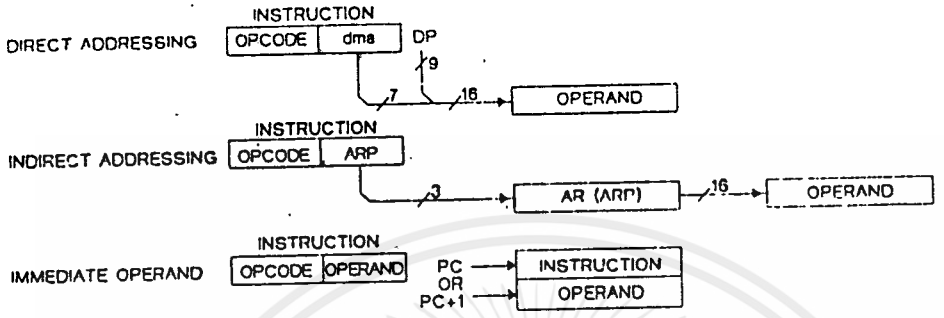
TMS320c25 สามารถอ้างแอดเดรสของหน่วยความจำโปรแกรม และหน่วยความจำข้อมูลได้สูงสุดแต่ละชนิดเท่ากับ 64 Kwords หน่วยความจำข้อมูลบนชิปถูก map อยู่ใน 64Kwords ของเนื้อที่หน่วยความจำข้อมูลส่วน On-chip ROM ใน TMS320c25 ถูกmapอยู่ในเนื้อที่ของหน่วยความจำโปรแกรมเมื่ออยู่ในการทำงานโหมดไมโครคอมพิวเตอร์

Data address bus(DAS) อ้างแอดเดรสของหน่วยความจำข้อมูลด้วยวิธีดังต่อไปนี้

1.Direct Address Bus(DRB) ซึ่งมีการใช้โหมดการอ้างแอดเดรสแบบโดยตรง(direct addressing mode) เช่น ADD>10

2.Auxiliary register File Bus(AFB)ซึ่งมีการใช้โหมดการอ้างแอดเดรสโดยอ้อม (Indirect addressing mode) เช่น ADD*โอเปอร์แรนด์ถูกอ้างแอด

เดรสด้วยค่าของโปรแกรมเคาน์เตอร์ในการอ้างแอดเดรสแบบImmediate addressing mode




รูปที่ 3.7 แสดงแผนภาพการอ้างแอดเดรสโอเปอร์แรนด์แบบ Direct,Indirectและ Immediate addressing mode

ใน Direct addressing mode, Data memory page pointer(DP) ที่มีขนาด 9 บิท เป็นตัวดรรชนีที่หน้าใดหน้าหนึ่งในจำนวน 512 หน้า (page) ในแต่ละหน้ามี 128 words แอดเดรสของหน่วยความจำข้อมูล(data memory address->dma) ใช้ LSBs จำนวน 7 บิทของคำสั่งในการชี้เฉพาะไปยังเวิร์ดที่ต้องการในหน้านั้นๆแอดเดรสบน DRB (Direct Address Bus) ถูกสร้างขึ้นมาโดยการนำ 9-bit DP และ 7-bit ของ DMA มารวมกันเป็นขนาด 16 บิท

ใน Indirect addressing mode, ใช้เลขที่อยู่ใน Current AR ที่ถูกดรรชนีโดย ARP ในการอ้างแอดเดรสหน่วยความจำข้อมูลโดยผ่าน Auxiliary register file bus (AFB) ขณะที่ออกซิลิอารีรีจิสเตอร์ที่ถูกเลือกมีแอดเดรสของหน่วยความจำข้อมูล และข้อมูลกำลังถูกโยกย้ายโดย CALU เลขที่อยู่ในออกซิลิอารีรีจิสเตอร์อาจถูกโยกย้ายผ่าน ARAU ดูรูปที่ 3.5 สำหรับตัวอย่างของ Indirect auxiliary

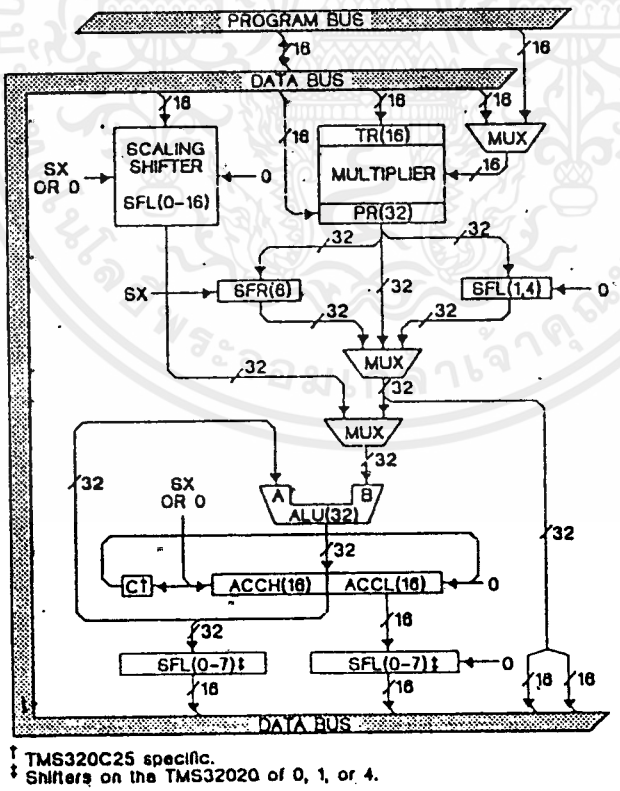
register addressing

เมื่อ Immediate operand ถูกใช้มันจะประกอบอยู่ในเวิร์ดคำสั่งของมันเองหรือหากเป็นกรณีของ 16-bit Immediate operand มันจะประกอบอยู่ในเวิร์ดที่ตามหลังออปโค๊ด

## CENTRAL ARITHMETIC LOGIC UNIT(CALU)

CALU ประกอบด้วย

- 16-bit scaling shifter
- 16x16-bit parallel multiplier
- 32-bit Arithmetic Logic Unit(ALU)
- 32-bit accumulator(ACC)
- shifters ที่เอาท์พุทของ accumulatorและ multiplier




† TMS320C25 specific.
‡ Shifters on the TMS32020 of 0, 1, or 4.

รูปที่ 3.8 แสดงบล็อกไดอะแกรมของส่วนประกอบของ CALU

ซึ่งจากรูป SFL และ SFR หมายถึงชิพท์ไปทางซ้ายหรือขวาตามลำดับ

ในการทำให้เป็นผล (implementation) ของคำสั่งที่เกี่ยวกับ ALU จะมีขั้นตอนดังนี้

1) ข้อมูลถูกเฟทช์จากแรมผ่านดาต้าบัส

2) ข้อมูลถูกส่งผ่านsccaling shifter และ ALU

3) ข้อมูลถูกเคลื่อนย้ายสู่แอคคิวมูเลเตอร์

อินพุทตัวหนึ่งของALU จะได้มาจากแอคคิวมูเลเตอร์เสมอส่วนอีกตัวหนึ่งอาจถูกเคลื่อนย้ายมาจาก PR (Product Register)ของมัลติพลายเออร์หรือจาก Scaling shifter ที่ถูกโหลดมาจากหน่วยความจำข้อมูล

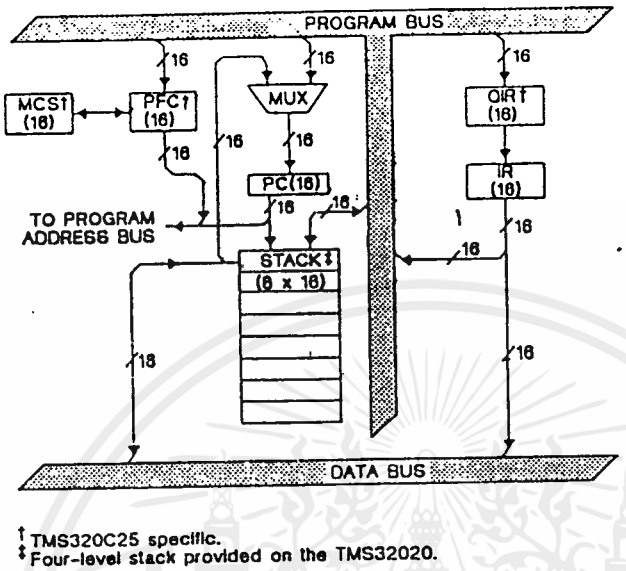
SYSTEM CONTROL

1) Program Counter and Stack

TMS320c25มีโปรแกรมเคาน์เตอร์(PC)ขนาด 16บิทและฮาร์ดแวร์สแตก 8 ตำแหน่งสำหรับการเก็บค่าโปรแกรมเคาน์เตอร์ โปรแกรมเคาน์เตอร์ใช้ในการอ้างแอดเดรสของหน่วยความจำภายในและภายนอกในคำสั่งการเฟทช์สแตกจะถูกใช้ระหว่างการอินเท่อร์รัพท์และเรียกใช้ซับรูทีน

โปรแกรมเคาน์เตอร์อ้างแอดเดรสหน่วยความจำโปรแกรมทั้งที่อยู่บนชิปและนอกชิปโดยผ่าน Program Address Bus(PAB) คำสั่งจะถูกเฟทช์จากหน่วยความจำโปรแกรมและถูกโหลดไปที่ Instruction register(IR) เมื่อIRถูกโหลดโปรแกรมเคาน์เตอร์พร้อมที่จะเริ่มเฟทช์ไซเคิล (fetch cycle)ของคำสั่งต่อไปโปรแกรมเคาน์เตอร์อาจอ้างแอดเดรสของ On-chip RAM บล็อก B0 ได้เมื่อ B0 ถูกกำหนดให้เป็นหน่วยความจำโปรแกรม หรือโปรแกรมเคาน์เตอร์ใช้อ้างแอดเดรสของ on-chip ROM บน TMS320c25 ได้โปรแกรมเคาน์เตอร์อ้างแอดเดรสของหน่วยความจำโปรแกรมนอกชิปได้โดยใช้แอดเดรสบัส ($A_{15}$-$A_0$) และดาต้าบัสภายนอก ($D_{15}$-$D_0$)

หน่วยความจำข้อมูลถูกอ้างแอดเดรสโดยโปรแกรมเคาน์เตอร์ระหว่างคำสั่ง BLKD ซึ่งทำหน้าที่เคลื่อนย้ายบล็อกข้อมูลจากส่วนหนึ่งของหน่วยความจำข้อมูลไปยังส่วนอื่น




รูปที่ 3.9 Program Counter, Stac and Relate Hardware

ค่าในแอคคิวมูเลเตอร์อาจถูกโหลดไปสู่โปรแกรมเคาน์เตอร์เพื่อทำการปฏิบัติการ "Computed GOTO" ให้สำเร็จซึ่งเหตุการณ์ดังกล่าวถูกทำให้เกิดขึ้น เมื่อใช้คำสั่ง BACC(Branch to address in accumulator) หรือ คำสั่ง CALA (Call subroutine indirect) ในการเริ่มเฟทช์ไซเคิลใหม่ โปรแกรมเคาน์เตอร์(PC) จะถูกโหลดด้วยค่า PC+1 หรือด้วยแอดเดรสของการบรานช์ ในกรณีที่เป็นคำสั่งเกี่ยวกับการบรานช์, Calls และอินเทอร์รัพท์

TMS320c25 มีลักษณะซึ่งยอมให้มีการเอ็กซีคิวต์คำสั่งเดียว (single instruction) ที่อยู่ถัดไปเป็นจำนวน N+1 ครั้ง, โดยที่ N หมายถึงค่าที่ได้จากการโหลด Repeat Counter(RPTC) ซึ่งเป็นเคาน์เตอร์ขนาด 8 บิท ถ้าลักษณะการทำซ้ำนี้ถูกใช้

คำสั่งถูกเอ็กซีคิวต์และค่าของ RPTC ถูกลดลงจนกระทั่ง RPTC เป็นศูนย์ ลักษณะเช่นนี้เป็นประโยชน์กับคำสั่งต่างๆมากมาย เช่น NORM (Normalize Contents of Accumulator), MACD (Multiply and Accumulate with data move) และ SUBC (Condition substract) เมื่อใช้กับคำสั่งแบบมัลติไซเคิล (Multicycle instructions) เช่น MACD ลักษณะการทำซ้ำสามารถทำให้คำสั่งเหล่านี้มีการเอ็กซีคิวต์ในไซเคิลเดียว

สแตก (Stack) มีขนาด 16 bit และมี 8 ระดับ PC stack สามารถแอคเซสได้โดยการใช้คำสั่ง POP และ PUSH เมื่อไรก็ตามที่ค่าของโปรแกรมเคาน์เตอร์ถูก Push ลงบนระดับบนสุดของสแตกค่าที่อยู่ในสแตกในแต่ละระดับเมื่อก่อนหน้านี้จะถูกดัน Push ลงไปตามลำดับและจะทำให้ค่าของสแตกที่ระดับสุดท้ายหายไป ดังนั้นจะทำให้ข้อมูลเกิดเสียหายหากเกิดการ push มากกว่า 8 ครั้งก่อนการ Pop 1 ครั้ง ในทางกลับกันหลังจากการ Pop ใดๆ 7 ครั้งติดกันจะทำให้เกิดมีค่าตัวเลขที่สแตกระดับต่ำสุด ซึ่งสแตกทุกระดับจะมีค่าเดียวกันหมด คำสั่ง PSHD และ POPD ใช้ในการ Push ค่าของหน่วยความจำข้อมูลลงบนสแตก หรือ Pop ค่าจากสแตกให้กับหน่วยความจำข้อมูล คำสั่งเหล่านี้ยอมให้สแตกถูกสร้างในหน่วยความจำข้อมูลสำหรับการเก็บของซับรูทีน/อินเทอร์รัพท์ที่เกิน 8 ระดับ

2) Pipeline Operation

การทำไปป์ไลน์คำสั่งมีลำดับของการปฏิบัติงานของบัสภายนอกซึ่งเกิดขึ้นในระหว่างการเอ็กซีคิวต์คำสั่ง ไปป์ไลน์การ Prefetch-Decode-Execute เป็นไปป์ไลน์ที่ User มองไม่เห็นที่จำเป็น ยกเว้นในบางกรณีที่ไปป์ไลน์ต้องถูกแตกออกกจากกัน เช่น คำสั่งการบรานช์ ในการปฏิบัติงานไปป์ไลน์, การ Prefetch,Decode และ Execute จะเป็นอิสระซึ่งยอมให้เกิดการเอ็กซีคิวต์คำสั่งซ้อนกันได้ ดังนั้นระหว่างไซเคิลใดๆ,จะมี 2 หรือ 3 คำสั่งสามารถแอคทีฟได้ แต่ละคำสั่งจะอยู่ที่สเตจต่างกันเมื่อทำงานสมบูรณ์ เป็นผลจากการไปป์ไลน์สามระดับของ TMS320c25

ความแตกต่างของระดับไปป์ไลน์ไม่ทำให้เกิดผลกระทบต่อความเร็วในการเอ็กซีคิวต์คำสั่งเพียงแต่เปลี่ยนลำดับของการเฟทช์/ดีโค้ด คำสั่งทั้งหมดเอ็กซีคิวต์ด้วยจำนวนไซเคิลที่เท่ากันโดยไม่คำนึงว่าคำสั่งเหล่านั้น ถูกเอ็กซีคิวต์จาก RAM, ROMหรือ

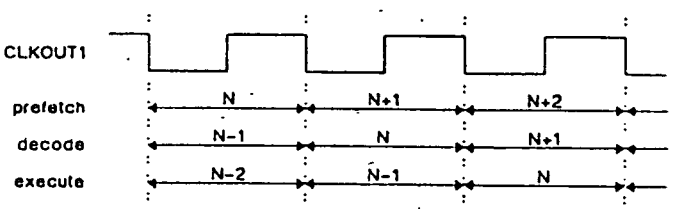
หน่วยความจำข้อมูลภายนอก

จากรูปที่ 3.9 แสดงให้เห็นฮาร์ดแวร์ที่สัมพันธ์กับโปรแกรมเคาน์เตอร์ของ TMS320c25 เพื่อการไปป์ไลน์สามระดับซึ่งมีประสิทธิภาพสูงกว่า ฮาร์ดแวร์เหล่านั้นได้แก่ Prefetch Counter (PFC),16-bit MicroCall Stack(MCS) register,Instruction RegistOer(IR) และ Queue Instruction Register(QIR)

ใน PFC มีแอดเดรสของคำสั่งต่อไปที่จะถูก Prefetch อยู่ขณะที่คำสั่งถูก prefetch คำสั่งจะถูกโหลดเข้าสู่ IR ยกเว้นเมื่อ IR ยังคงมีคำสั่งซึ่งกำลังเอ็กซีคิวต์ในขณะนั้นอยู่ ซึ่งในกรณีนี้คำสั่ง prefetch จะถูกเก็บไว้ใน QIR จากนั้นค่า PFC จะถูกเพิ่มขึ้นและหลังจากเอ็กซีคิวต์คำสั่งในขณะนั้น (Current instruction) สมบูรณ์แล้ว คำสั่งใน QIR ก็จะถูกโหลดเข้าสู่ IR เพื่อนำไปเอ็กซีคิวต์ต่อไป

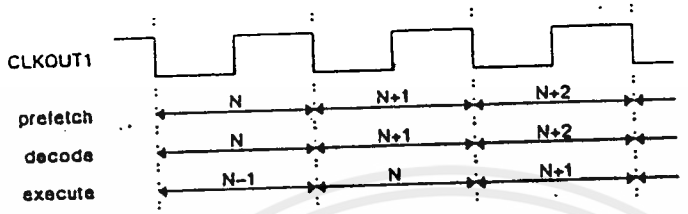
ในโปรแกรมเคาน์เตอร์มีแอดเดรสของคำสั่งต่อไปที่จะถูกเอ็กซีคิวต์อยู่ และจะไม่ถูกใช้โดยตรงในคำสั่งการเฟทช์ เพียงแต่ทำตัวเหมือนกับพอยน์เตอร์อ้างอิงกับตำแหน่งปัจจุบันในโปรแกรม ค่าโปรแกรมเคาน์เตอร์จะถูกเพิ่มขึ้นขณะที่แต่ละคำสั่งถูกเอ็กซีคิวต์ เมื่อเกิดการอินเทอร์รัพท์หรือเรียกใช้ซับรูทีน ค่าที่อยู่ในโปรแกรมเคาน์เตอร์จะถูกส่งไปที่สแตกเพื่อที่จะเก็บค่าเอาไว้เพื่อการlinkกลับเข้าสู่โปรแกรมได้อย่างถูกต้อง

การ Prefetch, Decode และ Execute ของการไปป์ไลน์เป็นอิสระ ดังนั้นจึงมีการยอมให้เกิดการเอ็กซีคิวต์ซ้อนกันได้ ระหว่างไซเคิลใดๆ คำสั่งที่ต่างกัน 3 คำสั่งสามารถแอคทีฟพร้อนกันได้ แต่ละคำสั่งจะอยู่ในสเตจ (Stage)ซึ่งต่างกัน ดังรูปที่3.10 ซึ่งแสดงให้เห็นถึงการไปป์ไลน์ 3 ระดับสำหรับการเอ็กซีคิวต์คำสั่งที่เป็นเวิร์ดเดียวและใช้ไซเคิลเดียว (Single word, Single cycle instructions)จาก ROM หรือหน่วยความจำภายนอกโดยไม่มีสภาวะรอ (Wait state)




รูปที่ 3.10 Three Level Pipeline Operation

การไปป์ไลน์จะถูกลดลงเหลือ 2ระดับเมื่อเกิดการเอ็กซีคิวต์จากหน่วยความจำโปรแกรมภายใน ซึ่งจะทำให้คำสั่งใน internal RAM สามารถเฟทช์และดีโค้ดในไซเคิลเดียวกัน ดังรูปที่ 12




รูปที่ 3.11 Two Level Pipeline Operation

3) Reset

รีเซ็ท (~RS) เป็นการอินเทอร์รัพท์ภายนอกแบบ non-maskable ซึ่งสามารถถูกใช้ในเวลาใดๆ เพื่อทำให้ TMS320c25 อยู่ในสภาวะที่ user รู้รีเซ็ทจึงมักจะถูกใช้หลังจากการเพาว์เวอร์-อัพ การป้อนสัญญาณ low ให้กับขา ~RS จะเป็นผลให้ TMS320c25 ยกเลิกการเอ็กซีคิวต์ และทำให้ค่าโปรแกรมเคาน์เตอร์เป็นศูนย์ ~RSมีผลต่อรีจิสเตอร์และบิทสถานะต่างๆด้วย ขณะเพาเวอร์-อัพ สถานะของโปรเซสเซอร์ไม่แน่นอน สำหรับการปฏิบัติการของระบบที่ถูกต้อง หลังจากเพาเวอร์-อัพสัญญาณรีเซ็ทต้องถูกทำให้เป็นlowอย่างน้อยที่สุด 3 clock cycle เพื่อทำให้เกิดการรีเซ็ทอุปกรณ์อย่างแน่นอน

เมื่อได้รับสัญญาณรีเซ็ท(~RS) จะเกิดผลดังนี้

1.CNF(configuration control) bitใน status register ST1 มีสถานะเป็น 0 เนื่องจากแรมทั้งหมดถูกกำหนดให้เป็นหน่วยความจำข้อมูล (data memory)

2.โปรแกรมเคาน์เตอร์ (PC) ถูกทำให้เป็น 0, และแอดเดรสบัส ($A_{15}$-$A_0$) เป็น 0 ทั้งหมด ขณะที่ขา ~RS เป็น low

3.ดาต้าบัส($D_{15}$-$D_0$)อยู่ในสภาวะไฮ-อิมพีแดนซ์(high impedance state)

4.สัญญาณควบคุมหน่วยความจำและI/Oทั้งหมด(~PS,~DS,~IS,R/~W, ~STRBและ~BR) ถูกทำให้เป็น high ขณะที่ ~RS เป็น low

5.อินเทอร์รัพท์ทั้งหมดถูกดิสอะเบิลโดยการเซ็ทให้INTM(interrupt mode)เป็น 1,อินเทอร์รัพท์แฟลกรีจิสเตอร์ (IFR) ถูกรีเซ็ทให้เป็น 0

6.บิทสถานะ(Status bits)

0 ->OV และ 1 ->XF

1 ->SXM,0 ->PM,1 -> HM,0 ->FO,1 ->C,และ1 ->FSM

(status bits ที่เหลือไม่มีการเปลี่ยนแปลง)

7.Global memory allocation register (GREG) ถูกเคลียร์เพื่อทำmemory local ทั้งหมด

8.รีพีทเคาน์เตอร์(RPTC)ถูกเคลียร์

9.ขา DX (data transmit) อยู่ในสภาวะไฮ-อิมพีแดนซ์ การรับ/ส่งบนserial portถูกยกเลิก,และ TXM (transmit mode) bit ถูกรีเซ็ทให้เป็น low ซึ่งเป็นการกำหนดให้ FSX framing pulse เป็นอินพุทการส่ง/รับอาจถูกเริ่มโดย framing pulsesหลังจากการเอาสัญญาณ ~RS ออก

10.TIM register และ PRD register ถูกเซ็ทให้เป็นค่าสูงสุด (FFFF) โดยTIM register เริ่มมีการลดลงหลังจาก ~RS เป็น high เท่านั้น

11.สัญญาณ ~IACK (interrupt acknowledge) ถูกสร้างขึ้นในลักษณะเดียวกับmaskable interrupt)

12.สถานะของแรมไม่ถูกกำหนด

13.บิท ARB,ARP,DP,IMR,OVM และ TC จะไม่ถูก initialบิทต่างๆ เหล่านี้สามารถ initial ได้โดยใช้ซอฟท์แวร์โดย user หลังจากที่มีการรีเซ็ทแล้ว

การเอ็กซีคิวต์เริ่มต้นจากตำแหน่งที่ 0 ของหน่วยความจำโปรแกรม เมื่อสัญญาณ ~RS เป็น high ในขณะที่อยู่ในโหมด hold ถ้า ~RS เป็น low อยู่การ

รีเซ็ทตามปกติจะเกิดขึ้นภายใน แต่บัสและ control lines ทั้งหมดอยู่ในสภาวะไฮ-อิมพีแดนซ์

4) Status Registers

Status registers 2 อันคือ ST0, ST1 มีสถานะของกรณีและโหมดต่างๆStatus register สามารถถูกเก็บในหน่วยความจำข้อมูลและถูกโหลดจากหน่วยความจำข้อมูลได้ ดังนั้นจึงมีการยอมให้สถานะของ machine ถูกเซฟและรีสโตร์ได้สำหรับอินเทอร์รัพท์และซับรูทีน Status bits ทั้งหมดถูกเขียนลงและอ่านได้โดยการใช้คำสั่ง LST/LST1 และ SST/SST1 ตามลำดับ(โดยมีข้อยกเว้นสำหรับ INTM ซึ่งไม่สามารถโหลดโดยคำสั่ง LSTได้)

DP,ARP และ ARB ถูกแสดงเหมือนเป็นรีจิสเตอร์ที่แยกออกจากกันในโปรเซสเซอร์บล็อกไดอะแกรมของรูปที่ 2 แต่เนื่องจากรีจิสเตอร์เหล่านี้ไม่มีคำสั่งสำหรับการเก็บค่าของรีจิสเตอร์ลงบนRAMของตัวมันเอง ดังนั้นรีจิสเตอร์เหล่านี้จึงถูกรวมกันใน status register

| FIELD | FUNCTION |
|---|---|
| ARB | Auxiliary Register Pointer Buffer.เมื่อไรก็ตามARPถูกโหลด.ค่าARPเก่าจะถูกก็อปปี้ไปที่ ARB ยกเว้นระหว่างคำสั่ง LST เมื่อ ARB ถูก โหลดโดยคำสั่งLST1ค่าเดียวกันนี้จะถูกก็อปปี้ไปยังARPด้วย |
| ARP | Auxiliary Register Pointer.มีขนาด 3บิทเพื่อใช้เลือก AR ที่จะถูกใช้ในการแอดเดรสโดยอ้อม เมื่อ ARP ถูกโหลด,ค่า ARP เก่าถูกก็อปปี้ไปที่ARB ค่าARPอาจจะถูกเปลี่ยนแปลงโดยคำสั่งที่อ้างอิงกับหน่วยความจำขณะที่มีการใช้การอ้างแอดเดรสโดยอ้อม และจากคำสั่ง LARP, MAR,และ LST ARP จะถูกโหลดด้วยค่าเดียวกันกับ ARB เมื่อคำสั่งLST1ถูกเอ็กซีคิวต์ด้วย |

| | |
|---|---|
| C | Carry bit.บิทนี้จะถูกเซ็ทเป็น 1เมื่อผลลัพธ์ของการบวกเกิดตัวทดขึ้น หรือถูกรีเซ็ทเป็น 0 เมื่อผลลัพธ์ของการลบเกิดตัวยืมขึ้น มิฉะนั้นจะเกิดการรีเซ็ทหลังการบวกหรือการเซ็ท หลังจากการลบ ยกเว้นถ้าคำสั่งนั้นคือ ADDHหรือSUBH ADDH สามารถเพียงเซ็ทและ SUBH เพียงรีเซ็ทแครีบิท แต่ไม่สามารถมีผลต่อบิทนี้ในกรณีอื่นๆ คำสั่งการชิพท์และโรเทตมีผลต่อบิทนี้ด้วยเช่นเดียวกับคำสั่งSC,RC,และLST1 คำสั่งการบรานช์สองคำสั่งคือ BC และ BNC มีไว้สำหรับการบรานช์บนแครีบิท บิทนี้ถูกเซ็ทเป็น 1เมื่อมีการรีเซ็ท |
| CNF | On-Chip RAM Configuration Control bit.ถ้าบิทนี้เป็นศูนย์, B0 จะถูกกำหนดให้เป็นหน่วยความจำข้อมูลมิฉะนั้น B0จะถูกกำหนดให้เป็นหน่วยความจำโปรแกรม CNF อาจถูกเปลี่ยนแปลงโดยคำสั่ง CNFD, CNFP, และ LST1 ~RS ทำให้ CNF เป็นศูนย์ |
| DP | Data Memory Page Pointer.DP registerขนาด 9-bitถูกนำไปรวมกับ 7LSBs ของเวิร์ดคำสั่งเพื่อสร้างแอดเดรสโดยตรงขนาด 16-bit DPอาจถูกเปลี่ยนแปลงโดย คำสั่ง LST,LDP,และ LDPK |
| FO | Format bit.เมื่อเซ็ทเป็นศูนย์ serial port registerจะถูกกำหนดให้เป็นรีจิสเตอร์ ขนาด 16-bit เมื่อเซ็ทเป็น 1 Port register ถูกกำหนดให้รับ/ส่งแบบ 8-bit byte FOอาจถูกเปลี่ยนแปลงโดยคำสั่ง FORTและLST1,FOถูกรีเซ็ทเป็นศูนย์ |
| FSM | Frame Synchronization Mode bit.บิทนี้ชี้ให้เห็นว่าพอร์ทอนุกรมปฏิบัติการโดยร่วมกับ Frame sync pulse หรือไม่เมื่อ FSM=1, การปฏิบัติงานของพอร์ทอนุกรมจะเริ่มหลัง จากframe sync pulse |

| | |
|---|---|
| FSM | บนอินพุทของขาFSX/FSR เมื่อFSX=0,อินพุทของขาFSX/FSR จะไม่มีผล และพอร์ทอนุกรมจะปฏิบัติการอย่างต่อเนื่องโดยไม่ต้องการ Frame sync pulse บิทนี้ถูกเซ็ทเป็นหนึ่งโดยการรีเซ็ท |
| HM | Hold Mode bit.เมื่อHM=1,โปรเซสเซอร์หยุดการเอ็กซีคิวต์ภายในเมื่อรับรู้การแอคทีฟของขา~HOLD เมื่อHM=0,โปรเซสเซอร์อาจจะดำเนินการเอ็กซีคิวต์ที่ไม่อยู่ในหน่วยความจำโปรแกรมภายในแต่ทำให้ส่วนอินเทอร์เฟสภายนอกของมันอยู่ในสภาวะไฮ-อิมพี- แดนซ์ บิทนี้ถูกเซ็ทเป็นหนึ่งโดยการรีเซ็ท |
| INTM | Interrupt Mode bit.เมื่อเซ็ทเป็นศูนย์,Unmasked interruptทุกชนิดจะเอ็นาเบิลเมื่อเป็นหนึ่ง,maskable interruptทั้งหมดถูกดิสเอ-เบิล INTMถูกเซ็ทและรีเซ็ทได้โดยใช้คำสั่งDINTและEINTตามลำดับ สัญญาณ~RSและ~IACKจะเซ็ทINTMด้วย INTMไม่มีผลต่อunmaskable ~RS interrupt,INTMไม่มีผลโดยคำสั่งLST |
| OV | Overflow Flag bit.OVถูกเซ็ทเป็นหนึ่งเมื่อเกิด overflowขึ้นที่ ALU ขณะที่เกิดOver- flowขึ้น OVยังคงเซ็ทจนกระทั่งเกิดการรีเซ็ท,คำสั่ง BV ,BNVหรือLSTทำการเคลียร์ OV |
| OVM | Overflow Mode bit.เมื่อเซ็ทเป็นศูนย์ผลลัพธ์ที่เกิด Overflow จะ overflowตามปกติในแอคคิวมูเลเตอร์เมื่อเซ็ทเป็นหนึ่งแอคคิวมูเลเตอร์ถูกเซ็ทเป็นค่าบวกหรือลบสูงสุดของ มันขึ้นอยู่กับการเจอ Overflow คำสั่ง SOVMและROVMใช้เซ็ทและรีเซ็ทบิทนี้ตามลำดับ คำสั่งLSTอาจถูกใช้เพื่อเปลี่ยนแปลงค่าของOVMด้วย |

| | |
|---|---|
| PM | Product Shift Mode.มีขนาด 2-bit,หากมีค่าเป็น00 ผลลัพธ์ขนาด 32-bitของมัลติพลายเออร์ถูกโหลดลงในALU โดยไม่มีการชิพท์ ถ้าPM=01 ,เอาท์พุทของPRจะถูกชิพท์ไปทางซ้ายหนึ่งตำแหน่งและถูกโหลดไปที่ALUโดย มีการเติมLSBsด้วย0s ถ้าPM=10,เอาท์พุทของ PR จะถูกชิพท์ไปทางซ้าย 4-bit และถูกโหลดลงใน ALU โดยมีการเติม LSBs ด้วย 0sถ้าPM=11 ,ทำการชิพท์ไปทางขวา 6-bit,sign-extended ค่าของPRยังคงไม่ เปลี่ยนแปลง การชิพท์เกิดขึ้นเมื่อการเคลื่อนย้ายข้อมูลของPRไปยังALU PMถูกโหลดด้วยคำสั่งSPMและ LST1 บิทPMถูกเคลียร์โดย ~RS |
| SXM | Sign Extension Mode bit.SXM=1,เกิดsign extensionบนข้อมูล ขณะที่ถูกผ่านไปสู่แอคคิวมูเลเตอร์โดยผ่านscaling shifter SXMไ ม่มี ผลต่อนิยามของคำสั่งที่แน่นอนเช่นคำสั่งADDS จะกำจัด Sign Extension โดยไม่คำนึงถึงSXM บิทนี้ถูกเซ็ทและรีเซ็ท ด้วยคำสั่ง SSXM และ RSXM และอาจถูกโหลดโดยLST1ด้วย SXMถูกเซ็ทเป็นหนึ่งด้วย ~RS |
| TC | Test/Control Flag bit.บิทTCถูกกระทบโดยคำสั่ง BIT,BITT,CMPR ,LST1 และ NORMบิทTCถูกเซ็ทเป็นหนึ่ง ถ้าบิทถูกทดสอบโดยBITหรือBITT เป็นหนึ่ง ถ้าเป็นกรณีการเปรียบเทียบโดยถูกทดสอบโดย CMRR ที่มีอยู่ ระหว่าง AR0 และ AR ตัวอื่นๆซึ่งถูกชี้โดย ARP หรือถ้า Exclusive-or fn.ของ 2MSBsของแอคคิวมูเลเตอร์เป็นจริง เมื่อถูกทดสอบด้วยคำสั่ง NORM คำสั่งการบรานช์ 2 คำสั่งคือ BBZ และ BBNZ มีไว้เพื่อทำการ บรานช์บนสถานะของTC |
| TXM | Transmit Mode bit.TXM=1เป็นการกำหนดให้ขาFSXของพอร์ทอนุกรม เป็นเอาท์พุทใน โหมดนี้ พัลส์ถูกสร้างบน FSX เมื่อ DXR ถูกโหลดดังนั้น |

| | |
|---|---|
| TXM | การส่งจึงเริ่มขึ้นที่ขา DX TXM=0เป็นการกำหนดให้ขาFSXเป็นอินพุท TXM ถูกเซ็ทและรีเซ็ทโดยคำสั่ง STXM และ RTXM และอาจถูกโหลดโดยคำสั่ง LST1ด้วย สัญญาณ~RSทำให้TXMเป็นศูนย์ |
| XF | XF pin status bit.บิทสถานะนี้ชี้ถึงสถานะของขา XF XF ถูกเซ็ท และรีเซ็ทโดยคำสั่ง SXF และ RXF หรืออาจถูกโหลดโดยคำสั่งLST1 XF ถูกเซ็ทเป็นหนึ่งด้วย~RS |

5) Repeat Counter

รีพีทเคาน์เตอร์ (RPTC)เป็นเคาน์เตอร์ขนาด 8-bit ซึ่งเมื่อถูกโหลดด้วยค่า N จะทำให้คำสั่งต่อไปถูกเอ็กซีคิวต์ N+1 ครั้ง RPTC สามารถถูกโหลดด้วยค่าตั้งแต่ 0 ถึง 255 โดยการใช้คำสั่ง RPT(repeat) หรือไม่ก็คำสั่ง RPTK(repeat immediate)ซึ่งจะทำให้เกิดการเอ็กซีคิวต์คำสั่งที่ให้มาได้มากที่สุด 256 ครั้ง RPTCถูกเคลียร์โดยการรีเซ็ท

ลักษณะการทำซ้ำสามารถถูกใช้โดยคำสั่ง เช่น multiply accumulates (MAC/MACD),block moves(BLKD/BLKP),I/O transfers(IN/OUT)และtable read/write(TBLR/TBLW) คำสั่งเหล่านี้ซึ่งปกติเป็นคำสั่งแบบมัลติไซเคิล จะถูกไปป์ไลน์เมื่อมีการใช้ลักษณะการทำซ้ำและกลายเป็นคำสั่งแบบไซเคิลเดียว ตัวอย่างคือ คำสั่ง table read อาจใช้ 3 ไซเคิลหรือมากกว่านั้นในการเอ็กซีคิวต์ แต่เมื่อทำซ้ำตำแหน่งของ table สามารถถูกอ่านในทุกๆไซเคิล คำสั่งบางคำสั่งอาจไม่สามารถกระทำซ้ำได้

6) Powerdown Mode

เมื่อปฏิบัติการในเพาเวอร์ดาวน์โหมด TMS320c25 อยู่ในสภาวะที่ไม่แอคทีฟและต้องการเพาเวอร์ประมาณครึ่งหนึ่งของที่ต้องการในการชัพพลายอุปกรณ์เพาเวอร์ดาวน์โหมดถูกเรียกใช้โดยการเอ็กซีคิวต์ คำสั่ง IDLE และด้วยการไดรฟ์ขา ~HOLD ให้เป็น low ขณะที่ status bit HM=1

ขณะที่อยู่ในเพาเวอร์ดาวน์โหมด, สิ่งที่อยู่ภายในทั้งหมดของ TMS320c25

ถูกรักษาไว้เพื่อยอมให้การปฏิบัติการดำเนินต่อไปโดยไม่เปลี่ยนแปลงเมื่อเพาเวอร์ดาวน์โหมดถูกยกเลิก ระหว่างเวลานั้นเส้นดาต้า, แอดเดรส และสัญญาณควบคุม(~PS,~DS, ~IS,~STRB และ R/~W) ถูกรักษาไว้ในสภาวะ high impedance เพาเวอร์ดาวน์โหมดถูกยกเลิกโดยการรับอินเทอร์รัพท์เมื่อค่าสั่ง IDLE กำลังถูกเอ็กซีคิวต์หรือโดยการเคลื่อนย้ายอินพุทของ ~HOLD

## External Memory and I/O Interface

TMS320c25 สนับสนุนความต้องการในการเชื่อมต่อระบบอย่างกว้างขวาง ดาต้าแอดเดรส, โปรแกรมแอดเดรส และ I/O แอดเดรส มีไว้เพื่อการเชื่อมต่อกับหน่วยความจำและ I/O การเชื่อมต่อกับหน่วยความจำใช้สัญญาณต่างๆ ดังนี้

- 16-bit parallel data bus($D_{15}$-$D_0$)
- 16-bit address bus($A_{15}$-$A_0$)
- Dáta,program,and I/O space select(~DS,~PS,and ~IS) signals
- Various system control signals

สัญญาณ R/~W (read/write)ใช้ควบคุมทิศทางของการเคลื่อนย้ายสัญญาณ ~STRB (strobe) ทำให้เกิดสัญญาณเวลาในการควบคุมการเคลื่อนย้าย TMS320c25 มี 16 พอร์ทอินพุท และ 16พอร์ทเอาท์พุท พอร์ทเหล่านี้มี parallel I/O ขนาด 16 บิทสำหรับใช้ในการเชื่อมต่อโดยผ่านทางดาต้าบัสของอุปกรณ์ การปฏิบัติการที่มีอินพุทเดียวเอาท์พุทเดียวโดยใช้ค่าสั่ง IN หรือ OUT ใช้เวลา 2 ไซเคิล อย่างไรก็ตามเมื่อใช้ร่วมกับรีพีทเคาน์เตอร์ การปฏิบัติการใช้ 1 ไซเคิล

การออกแบบ I/O ถูกทำให้ง่ายด้วยการมี I/O ซึ่งมีแนวการปฏิบัติเหมือนกับหน่วยความจำ อุปกรณ์ I/O ถูก map ลงบน I/O address space ด้วยการใช้แอดเดรสบัส ภายนอก และดาต้าบัสภายนอกของโปรเซสเซอร์ในลักษณะเดียวกันกับ memory mapped device เมื่อมีการอ้างแอดเดรสหน่วยความจำภายใน ดาต้าบัสจะอยู่ในสภาวะไฮ-อิมพีแดนซ์ และสัญญาณควบคุมอยู่ในสภาวะที่ไม่แอคทีฟ (high)

การเชื่อมต่อกับหน่วยความจำและI/O,ในการปรับความเร็วถูกทำให้สำเร็จได้โดยการใช้สัญญาณ READY เมื่อมีการสื่อสารกับอุปกรณ์ที่ช้ากว่า TMS320c25 จะรอจนกระทั่งอุปกรณ์นั้นทำหน้าที่ของมันเสร็จ และส่งสัญญาณให้กับโปรเซสเซอร์โดยผ่านเส้น READY จากนั้นทำการเอ็กซีคิวต์ต่อไป

Memory Combination

การเชื่อมต่อหน่วยความจำโปรแกรม และหน่วยความจำข้อมูล สามารถเป็นไปได้ 6 แบบ ดังนี้

1.Program Internal RAM/Data Internal(PI/DI)

2.Program Internal RAM/Data External(PI/DE)

3.Program External/Data Internal(PE/DI)

4.Program External/Data External(PE/DE)

5.Program Internal ROM/Data Internal(PR/DI)

6.Program Internal ROM/Data External(PR/DE)

PI/DI หรือ PR/DI- เมื่อทั้งหน่วยความจำข้อมูลและหน่วยความจำโปรแกรมอยู่บนชิป โปรเซสเซอร์จะ run ด้วยความเร็วเต็มที่โดยไม่มีสภาวะรอ ( Note: คำสั่ง IN และ OUT มีไทม์มิงที่ต่างกันเมื่อโปรแกรมเมมโมรีอยู่ภายในโดย IN ต้องการ 2 ไซเคิลเพื่อใช้เอ็กซีคิวต์ขณะที่ OUT ต้องการเพียง 1 ไซเคิลเท่านั้น

PE/DI- หน่วยความจำโหมดนี้สามารถ run ด้วยความเร็วเต็มที่หากหน่วยความจำโปรแกรมภายนอกมีความเร็วพอเนื่องจากการปฏิบัติการกับข้อมูลสามารถเกิดขึ้นได้ในเวลาเดียวกับการแอคเซสหน่วยความจำภายนอก ถ้าหน่วยความจำโปรแกรมภายนอกมีความเร็วไม่พออาจมี wait state ถูกสร้างเป็นอินพุทของขา READY PI/DE,PE/DE, หรือ PR/DE- การเอ็กซีคิวต์คำสั่งซึ่งอ้างอิงกับหน่วยความจำข้อมูลภายนอกต้องการเวลา(cycle) เพิ่มขึ้นอีก เวลาที่เพิ่มขึ้นอย่างน้อย 2 ไซเคิลถูกต้องการในการเอ็กซีคิวต์คำสั่งอ่านจากหน่วยความจำข้อมูลภายนอก เช่น ADD, LAR เป็นต้น

# บทที่ 3

## การออกแบบวงจร

จากรูปที่ 3.1 ประกอบไปด้วยวงจรส่วนต่าง ๆ เหล่านี้

1. วงจรอินเทอร์เฟส
2. วงจรแอดเดรสซีเล็กเตอร์
3. วงจรดาต้าซีเล็กเตอร์
4. วงจรหน่วยความจำ
5. วงจร TMS320C25 และสัญญาณควบคุม

วงจรต่าง ๆ เหล่านี้จะทำงานร่วมกันอย่างสอดคล้องโดยมีสัญญาณควบคุมคอยควบคุม ให้การทำงานของแต่ละวงจรทำงานร่วมกันอย่างถูกต้อง

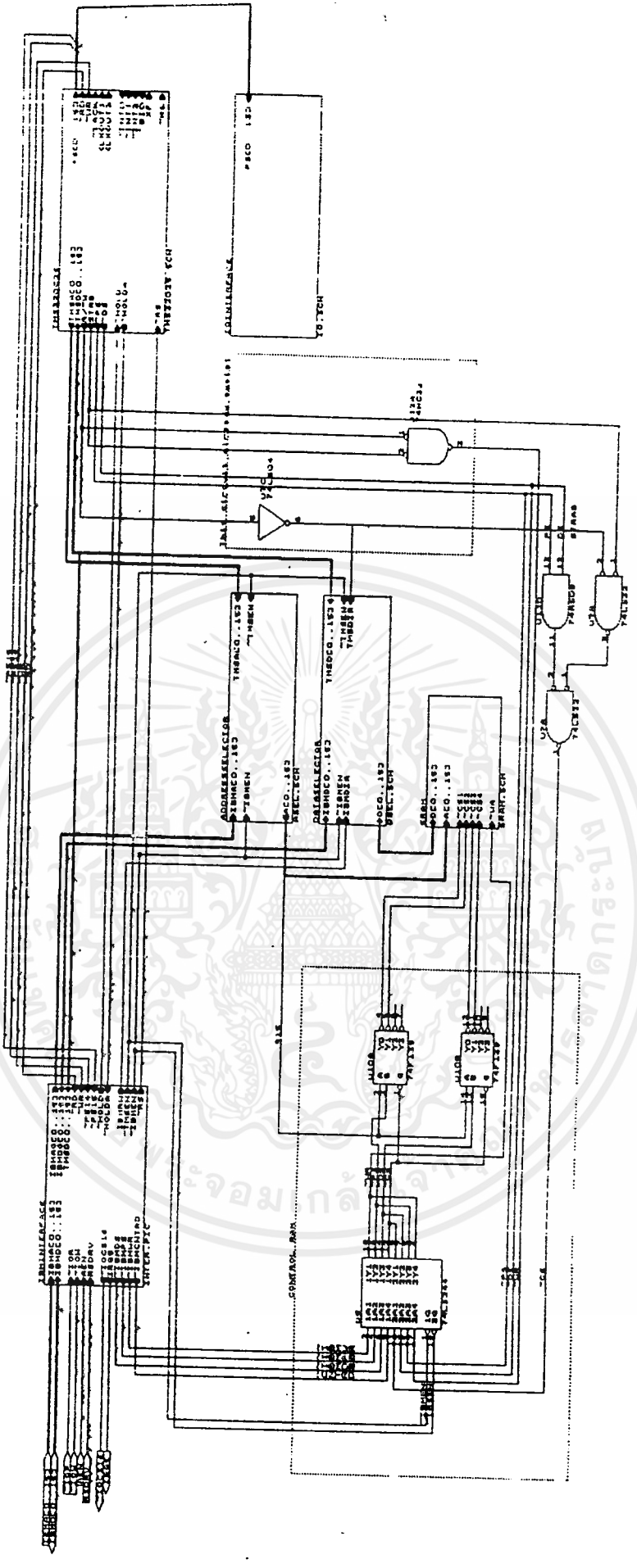
วงจรอินเทอร์เฟส จะทำหน้าที่เป็นตัวอินเทอร์เฟสระหว่างวงจรของ TMS กับ PC ซึ่ง PC จะส่งข้อมูลมาเก็บไว้ในหน่วยความจำภายนอกของ TMS โดยจะส่งข้อมูลที่ต้องการจะส่งผ่านวงจรอินเทอร์เฟสนี้ไปเก็บไว้ที่วงจรหน่วยความจำ

วงจรแอดเดรสซีเล็กเตอร์ จะทำหน้าที่เลือกต่อแอดเดรสบัสของวงจรหน่วยความจำเข้ากับแอดเดรสบัสของ PC หรือแอดเดรสบัสของ TMS ซึ่งขึ้นอยู่กับสัญญาณควบคุมหรืออีกนัยหนึ่งก็คือวงจรแอดเดรสซีเล็กเตอร์เป็นตัวกำหนดให้ PC หรือ TMS เข้าถึงข้อมูลภายในหน่วยความจำ

วงจรดาต้าซีเล็กเตอร์ ทำหน้าที่คล้ายกับวงจรแอดเดรสซีเล็กเตอร์โดยเลือกต่อดาต้าบัสของวงจรหน่วยความจำเข้ากับดาต้าบัสของ PC หรือดาต้าบัสของ TMS ซึ่งขึ้นอยู่กับสัญญาณควบคุมหรืออีกนัยหนึ่งก็คือ วงจรดาต้าซีเล็กเตอร์เป็นตัวกำหนดให้ PC หรือ TMS เข้าถึงข้อมูลภายในหน่วยความจำ

วงจรหน่วยความจำ (External Memory of TMS320C25) วงจรนี้จะเป็นวงจรหน่วยความจำภายนอกของ TMS320C25 วงจรหน่วยความจำนี้ประกอบไปด้วย Static RAM เบอร์ SC2568 8 ตัวซึ่งรวมแล้วมีขนาดทั้งหมด 128 KWord โดยจะแบ่งเป็นหน่วยความจำโปรแกรม 64 KWord และหน่วยความจำข้อมูลอีก 64 KWord

วงจร TMS320C25 และสัญญาณควบคุมจะทำงานโดย TMS จะไปอ่านโปร



**รูปที่ 3.1 แสดงบล็อกของวงจรทั้งหมด**

แกรมการทำงานมาจากหน่วยความจำโปรแกรมในวงจรหน่วยความจำและดาต้าจากหน่วยความจำข้อมูลในวงจรหน่วยความจำ เช่นกัน

เราได้แบ่งวงจรออกเป็น 2 ส่วนโดยส่วนแรกจะเป็นวงจรส่วนอินเทอร์เฟส วงจรแกดเดรสซีเล็กเตอร์ วงจรดาต้าซีเล็กเตอร์ และวงจรหน่วยความจำ

**วงจรอินเทอร์เฟส**

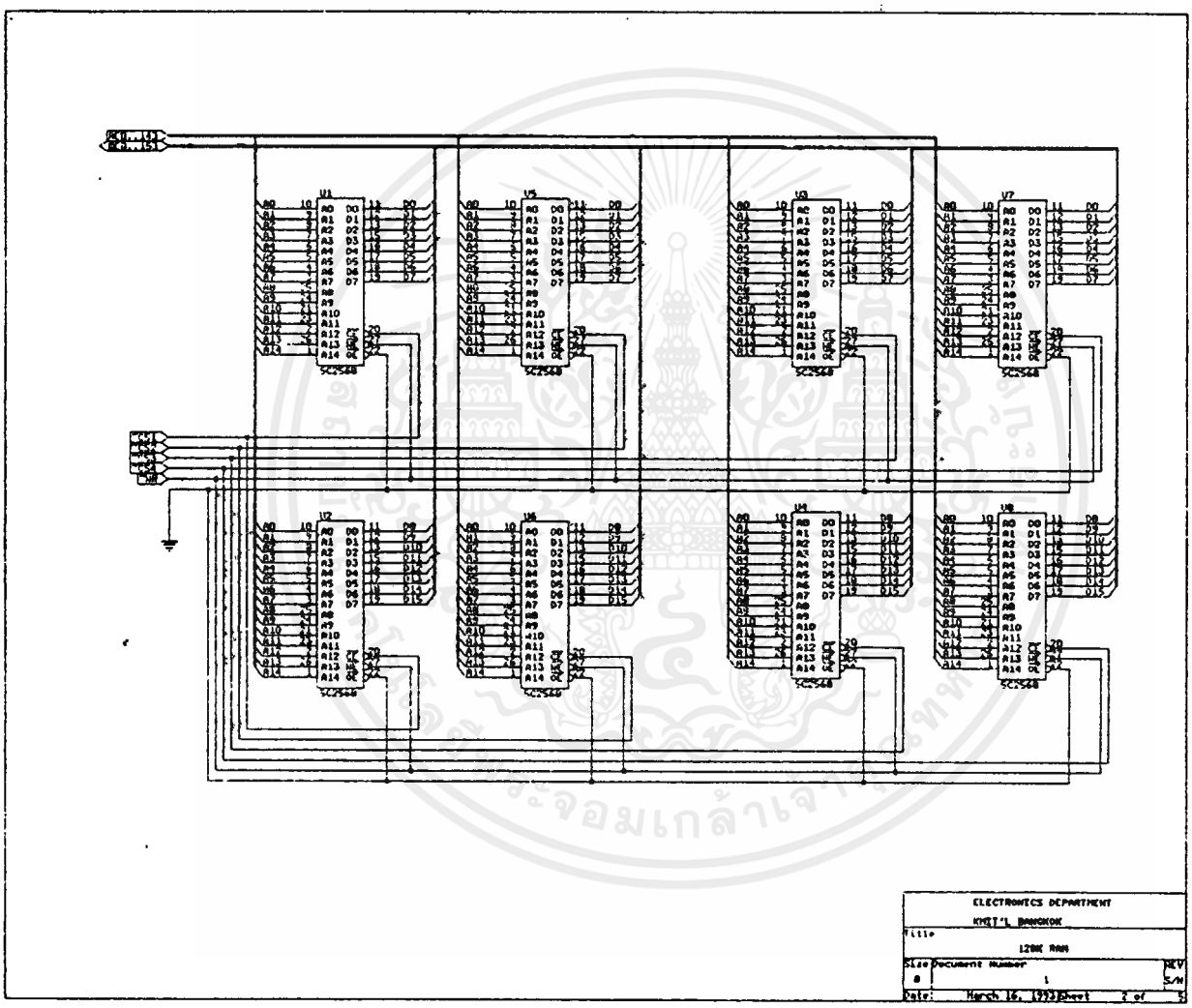
การที่เราให้ PC ติดต่อกับ TMS โดยผ่านทาง I/O Port นั้นก็เพื่อให้การส่งผ่านข้อมูลจาก PC ไป TMS มีความคล่องตัว รวดเร็ว และมีเสถียรภาพ

พอร์ทของ PC ที่เตรียมไว้มีทั้งแบบ XT และ AT ซึ่งแบบ XT นั้นจะเป็น Slotสั้น อยู่บริเวณด้านหลังของเครื่องคอมพิวเตอร์ และในบริเวณเดียวกันนั้นก็จะมี Slot ยาวสำหรับพอร์ท AT อยู่ด้วยสัญญาณต่าง ๆ ที่คยู่ใน XT Bus ประกอบไปด้วยแอดเดรสบัสจำนวน 16 เส้น ( A0-A15 ) ดาต้าบัสจำนวน 8 เส้น ( D0-D7 )และสัญญาณควบคุมที่จำเป็นต่าง ๆ สำหรับสัญญาณของ AT Bus จะเหมือนกับ XT Bus ทุกประการแต่จะมีส่วนที่เพิ่มขึ้นมาอีกคือ ดาต้าบัสจำนวน 8 เส้น (D8-D15) และสัญญาณควบคุมที่เป็นของ AT โดยเฉพาะอีกต่างหากซึ่งรายละเอียดของสัญญาณและตำแหน่งของสัญญาณบน Slot สามารถค้นคว้าได้จากในหนังสือคอมพิวเตอร์ฮาร์ดแวร์ทั่วไป

วงจรนี้ใช้การอินเทอร์เฟสแบบ AT และใช้การเคลื่อนย้ายข้อมูลทีละ 16 บิท จาก PC ไปยังวงจรหน่วยความจำซึ่งช่วยในการประหยัดเวลา การเคลื่อนย้ายข้อมูลมากกว่าใช้การอินเทอร์เฟสแบบ XT

ภายในวงจรอินเทอร์เฟสประกอบไปด้วย 74LS138 2 ตัว, DIP Switch 2 ตัว ใช้สำหรับดีโค้ดแอสเดรสของพอร์ท และ 8255 2 ตัว ซึ่งจะทำหน้าที่เป็นพอร์ทแบบขนานให้กับข้อมูลและแอดเดรสของข้อมูลดังรูปที่ 4.2

จากรูปที่ 3.2 74LS138 ทั้ง 2 ตัวจะทำหน้าที่ดีโค้ดแอดเดรสของพอร์ทร่วมกับ DIP Switch 2 ตัวซึ่งจะดีโค้ดสัญญาณแอดเดรสที่ส่งมาจาก PC ร่วมกับสัญญาณ AEN (Address Enable) สาเหตุที่ต้องนำสัญญาณ AEN มาดีโค้ดด้วยเพื่อป้องกันไม่ให้วงจรนี้ทำงานขณะที่ PC มีการทำ DMA อยู่ เพราะว่าขณะที่มีการทำ DMA PC จะปล่อยทั้งสัญญาณแอดเดรสลงมาที่แอดเดรสบัสพร้อมกับสัญญาณ IOR หรือ IOW ซึ่งบางครั้ง

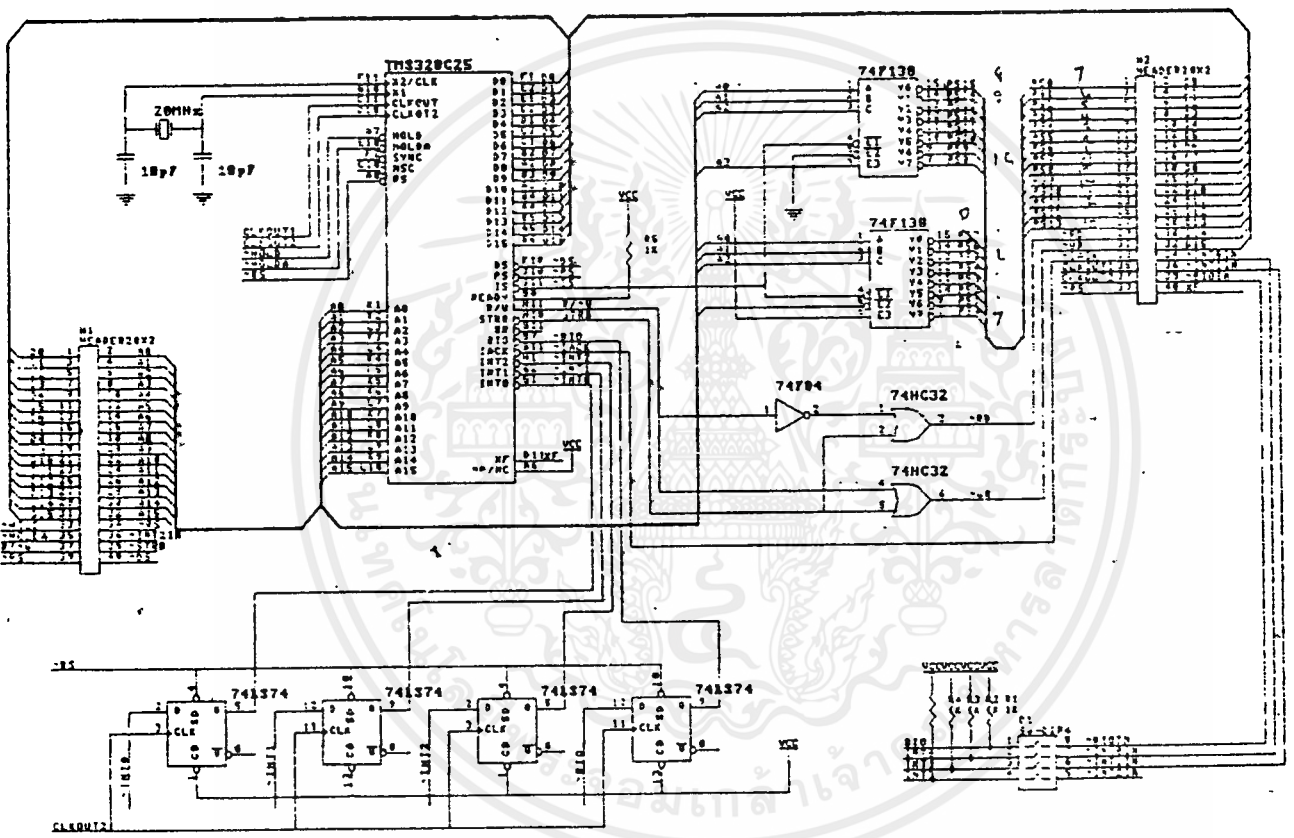



รูปที่ 3.3 วงจรหน่วยความจำ

วงจร TMS320C25 และสัญญาณควบคุม

ภายในวงจรนี้ประกอบไปด้วย TMS320C25 ทำหน้าที่เป็นหน่วยประมวลผลกลางของวงจรนี้ทั้งหมด 74F138 จำนวน 2 ตัว สำหรับดีโค้ดพอร์ตของ TMS320C25วงจรสร้างสัญญาณ Read และ Write และ D Flip-Flop 4 ตัวใช้สำหรับกำหนดลำดับของสัญญาณอินเทอร์รัพท์

สำหรับสัญญาณ CLOCK ของ TMS320C25 ในขั้นนี้เราเลือกใช้เพียง 20 MHz เพราะว่าอยู่ในขั้นทดลองวงจร ซึ่งการประกอบวงจรเราใช้การ ไวร์แลป แทนการออกแบบลายวงจรบนแผ่น PCB เพื่อให้สะดวกในการเปลี่ยนแปลงวงจรได้ตามต้องการ ซึ่งถ้าใช้สัญญาณ CLOCK ด้วยความเร็วสูงสุดที่ TMS ต้องการอาจทำให้เกิดความผิดพลาดอันเนื่องมาจากผลของการไวร์แลปได้




รูปที่ 3.4 วงจร TMS320C25

Processor Initialization

ก่อนที่จะนำเอาอัลกอลิทึมของดิจิตอลซิกแนลโปรเซสซิ่งที่ออกแบบไว้มาทำงานจะต้องมีการอินนิเชี่ยลโปรเซสเซอร์ก่อนทุกครั้งซึ่งโดยทั่วไปแล้วเราจะอินนิเชี่ยลโปรเซสเซอร์หลังจากการรีเซ็ท

เราสามารถรีเซ็ทโปรเซสเซอร์ได้โดย ให้สัญญาณ Low เข้าที่ขา ~RS ของโปรเซสเซอร์ไม่น้อยกว่าสัญญาณนาฬิกา 3 ลูก ซึ่งจะเป็นผลให้ TMS320C25 หยุดการทำงานและค่าโปรแกรมเคาน์เตอร์จะถูกเปลี่ยนให้เป็นศูนย์ ตำแหน่งของหน่วยความจำโปรแกรมจะไปชี้อยู่ที่ตำแหน่งศูนย์ และนอกจากนี้ยังมีการเปลี่ยนแปลงที่ Status bit

หลังจากที่เกิดการรีเซ็ทแล้ว เราควรที่จะอินนิเชี่ยลโปรเซสเซอร์เพื่อให้โปรเซสเซอร์เริ่มทำงานตามที่เราต้องการ ซึ่งสิ่งที่เราควรจะอินนิเชี่ยลหลังจากการรีเซ็ทคือ

- Memory-mapped rigisters
- Interrupt structure
- Mode control (OVM, SXM, FO, TXM, PM; plua HM and FSM on TMS320C25
- Memory control (CNF)
- Auxiliary registers and the auxiliary register pointer (ARP)
- Data memory page pointer (DP)

Example of Processor Initialization (TMS320C25)

```
TUTL    'PROCESSOR INITIALIZARTION'
IDT     'EXAMPLE'
DEF     RESET,INTO,INT1,INT2
DEF     TINT,RINT,XINT,USER
REF     TIME,RCV,XMT,PROC
```

```
PROCESSOR INITIALIZATION FOR THE TMS320C25

          AORG >0000

RESER     B    INIT     ; RS- BEGINS PROCESSING HERE.

INTO      B    ISRO     ; INTO- BEGINS PROCESSING HERE.

INT1      B    ISR1     ; INT1- BEGINS PROCESSING HERE.

INT2      B    ISR2     ; INT2- BEGINS PROCESSING HERE.

          AORG >0018

TINT      B    TIMER    ; TIMER INTERRUPT PROCESSING.

RINT      B    RCV      ; SERIAL PORT RECEIVE PROCESSING.

XINT      B    XMT      ; SERIAL PORT TRANSMIT PROCESSING.

USER      B    PROC     ; TRAP VECTOR PROCESSING BEGINS.


INIT   ROVM     ; DISABJ,E OVERFLOW MODE

       LDPK 0   ; POINT DP REGISTER TO DATA PAGE 0.

       LARP 7   ; POINT TO AUXILIARY REGISTER 7.

       LACK >3F ; LOAD ACCUMALATOR WITH >3F.

       SACL 4   ; ENABLE ALL INTERRUPTS VIA IMR.


INTERNAL DATA MEMORY INITIALIZATION.

ZAC              ; ZERO THE ACCUMULATOR.

LARK AR7,>60     ; POINT TO BLOCK B2.

RPTK 31

SACL *+          ; STORE ZERO IN ALL 32 LOCATIONS.

LRLK AR7,>200    ; PIONT TO BLOCK B0.

RPTK 255

SACL *+          ; ZERO ALL OF PAGER 4 AND 5.

SRLK AR7,>300    ; POINT TO BLOCK B1

RPTK 255
```

```
SACL *=          ; ZERO ALL OF PAGES 6 AMD 7
THE PROCESSOR ID INITIALIZED.
```

เราได้เขียนโปรแกรมที่มีการเซ็ทและรีเซ็ทค่า PRD Register เพื่อทำให้เกิด Timer interrupt โปรแกรมนี้ถูกเขียนขึ้นมาเพื่อใช้ในการตรวจสอบว่าวงจรที่ได้ออกแบบมานั้นทำงานได้ตามต้องการหรือไม่

ภายใน TMS320C25 มี Timer ขนาด 16 bit และใช้ร่วมกับการอินเทอร์รัพท์ในการปฏิบัติการฟังก์ชั่นต่าง ๆ Timer ตัวนี้เป็น Down counter ซึ่งได้รับ clock จากสัญญาณ CLKOUT1 ของ TMS320C25 อย่างต่อเนื่อง และจะนับเป็นจำนวนเท่ากับ (PRD+1) ไซเคิลของ CLKOUT1 เราสามารถทำให้เกิด Timer interrupt (TINT) ได้ทุก ๆ 2-65536 ไซเคิล ของ TMS320C25 โดยการโปรแกรมค่าใน Period register (PRD)

Memory mapped register 2 ตัวถูกใช้ในการปฏิบัติการของ Timer Timer register (TIM) ซึ่งหมายถึงหน่วยความจำข้อมูลตำแหน่งที่ 2 จะมีค่าของ Timer ที่นับในปัจจุบันอยู่ที่ทุก ๆ ไซเคิลของ CLKOUT1 ค่าใน TIM จะถูกลดลงทีละ 1 PRD register จะมีค่าเริ่มต้นในการนับของ Timer อยู่ เมื่อ TIM ถูกลดจนมีค่าเป็นศูนย์จะเกิด Timer interrupt ในไซเคิลต่อมาค่าใน PRD register จะถูกโหลดลงใน TIM ซึ่งในลักษณะนี้ จะทำให้เกิด Timer interrupt ทุก ๆ PRD+1 ไซเคิล

โปรแกรมนี้เป็นแอสเซมบลี้โค้ดในการประยุกต์ใช้ Timer สมมติเราโหลดค่า X ลงใน PRD register จะทำให้เกิดสัญญาณนาฬิกามีความถี่ตามสมการข้างล่าง คือ

CLKOUT1 / (PRD+1) = 2*Frequency of signal

จากวงจรเราใช้ CLOCKIN 20 MHz เพราะฉนั้น CLKOUT1 = 5 MHz

PRD = X

Frequency of signal = 5 MHz/2*(X+1)

ใน Timer interrupt service routine สัญญาณ XF ของ TMS320C2จะถูก Toggle สัญญาณที่ขา XF นี้ถูกใช้เป็นสัญญาณอินพุทของ BZO ด้วย

สัญญาณเอาท์พุทที่ขา XF จะเป็นสัญญาณนาฬิกาที่มี duty clycle 50% ตราบเท่าที่ยังไม่มีการดิสเอเบิลอินเทอร์รัพท์ ซึ่งอินเทอร์รัพท์อาจจะดิสเอเบิลโดยตรง หรือโดยการใช้ DZNT หรือโดยการเคลียร์ค่าสั่งรีทีทาโมดก็ได้

## TMS320C2x Signal Description

| SIGNAL | PIN (PGA/PLCC) | I/O/Z† | DESCRIPTION |
|---|---|---|---|
| SUPPLY/OSCILLATOR SIGNALS | | | |
| CLKOUT1 | C11/58 | O | Master clock output signal (CLKIN frequency/4). In this document (and on the TMS320C25), CLKOUT1 rises at the beginning of quarter-phase 3 (Q3) and falls at the beginning of quarter-phase 1 (Q1). See Appendix C for device phase definitions. |
| CLKOUT2 | D10/57 | O | A second clock output signal. In this document (and on the TMS320C25), CLKOUT2 rises at the beginning of quarter-phase 2 (Q2) and falls at beginning of quarter-phase 4 (Q4). See Appendix C for device phase definitions. |
| $V_{CC}$ | A10/61<br>B10/62<br>H2/23<br>L6/35 | I | Four 5-V supply pins, tied together externally. On the TMS32020, pin A6 is also a supply pin. |
| $V_{SS}$ | B1/10<br>K11/44<br>L2/27 | I | Three ground pins, tied together externally. |
| X1 | G10/51 | O | Output pin from the internal oscillator for the crystal. If a crystal is not used, this pin should be left unconnected. |
| X2/CLKIN | F11/52 | I | Input pin to the internal oscillator from the crystal. If a crystal is not used, a clock may be input to the device on this pin. |
| SERIAL PORT SIGNALS | | | |
| CLKR | B9/64 | I | Receive clock input. External clock signal for clocking data from the DR (data receive) pin into the RSR (serial port receive shift register). Must be present during serial port transfers. |
| CLKX | A9/63 | I | Transmit clock input. External clock signal for clocking data from the XSR (serial port transmit shift register) to the DX (data transmit) pin. Must be present during serial port transfers. |
| DR | J1/24 | I | Serial data receive input. Serial data is received in the RSR (serial port receive shift register) via the DR pin. |
| DX | E11/54 | O/Z | Serial data transmit output. Serial data transmitted from the XSR (serial port transmit shift register) via the DX pin. Placed in high-impedance state when not transmitting. |
| FSR | J2/25 | I | Frame synchronization pulse for receive input. The falling edge of the FSR pulse initiates the data-receive process, beginning the clocking of the RSR. |
| FSX | F10/53 | I/O | Frame synchronization pulse for transmit input/output. The falling edge of the FSX pulse initiates the data-transmit process, beginning the clocking of the XSR. Following reset, the default operating condition of FSX is as an input. This pin may be selected by software to be an output when the TXM bit in the status register is set to 1. |

† Input/Output/High-impedance state

## TMS320C2x Signal Description

| SIGNAL | PIN (PGA/PLCC) | I/O/Z† | DESCRIPTION |
|---|---|---|---|
| MULTIPROCESSING SIGNALS | | | |
| $\overline{BR}$ | G11/50 | O | Bus request signal. Asserted when the TMS320C2x requires access to an external global data memory space. READY is asserted to the device when the bus is available and the global data memory is available for the bus transaction. |
| $\overline{HOLD}$ | A7/67 | I | Hold input. When asserted, the TMS320C2x places the data, address, and control lines in the high-impedance state. |
| $\overline{HOLDA}$ | E10/55 | O | Hold acknowledge signal. Indicates that the TMS320C2x has gone into the hold mode and that an external processor may access the local external memory of the TMS320C2x. |
| $\overline{SYNC}$ | F2/19 | I | Synchronization input. Allows clock synchronization of two or more TMS320C2x's. $\overline{SYNC}$ is an active-low signal and must be asserted on the rising edge of CLKIN. |
| INTERRUPT AND MISCELLANEOUS SIGNALS | | | |
| $\overline{BIO}$ | B7/68 | I | Branch control input. Polled by BIOZ instruction. If low, the TMS320C2x executes a branch. This signal must be active during the BIOZ instruction fetch. |
| $\overline{IACK}$ | B11/60 | O | Interrupt acknowledge signal. Output is only valid while CLKOUT1 is low. Indicates receipt of an interrupt and that the program is branching to the interrupt-vector location indicated by A15-A0. |
| $\overline{INT2}$<br>$\overline{INT1}$<br>$\overline{INT0}$ | H1/22<br>G2/21<br>G1/20 | I | External user interrupt inputs. Prioritized and maskable by the interrupt mask register and the interrupt mode bit. |
| MP/$\overline{MC}$ | A6/1 | I | Microprocessor/microcomputer mode select pin for the TMS320C25 only. When asserted low (microcomputer mode), the pin causes the internal ROM to be mapped into the lower 4K words of the program memory map. In the microprocessor mode, the lower 4K words of program memory are external. On the TMS32020, MP/$\overline{MC}$ must be connected to $V_{CC}$. |
| $\overline{MSC}$ | C10/59 | O | Microstate complete signal. Asserted low and valid only during CLKOUT1 low when the TMS320C2x has just completed a memory operation, such as an instruction fetch or a data memory read/write. $\overline{MSC}$ can be used to generate a one wait-state READY signal for slow memory. |
| $\overline{RS}$ | A8/65 | I | Reset input. Causes the TMS320C2x to terminate execution and forces the program counter to zero. When brought to a high level, execution begins at location zero of program memory. $\overline{RS}$ affects various registers and status bits. |
| XF | D11/56 | O | External flag output (latched software-programmable signal). Used for signalling other processors in multiprocessor configurations or as a general-purpose output pin. |

† Input/Output/High-impedance state

## TMS320C2x Signal Description

| SIGNAL | PIN (PGA/PLCC) | I/O/Z† | DESCRIPTION |
|---|---|---|---|
| | | ADDRESS/DATA BUSES | |
| A15 MSB<br>A14<br>A13<br>A12<br>A11<br>A10<br>A9<br>A8<br>A7<br>A6<br>A5<br>A4<br>A3<br>A2<br>A1<br>A0 LSB | L10/43<br>K9/42<br>L9/41<br>K8/40<br>L8/39<br>K7/38<br>L7/37<br>K6/36<br>K5/34<br>L5/33<br>K4/32<br>L4/31<br>K3/30<br>L3/29<br>K2/28<br>K1/26 | O/Z | Parallel address bus A15 (MSB) through A0 (LSB). Multiplexed to address external data/program memory or I/O. Placed in high-impedance state in the hold mode. |
| D15 MSB<br>D14<br>D13<br>D12<br>D11<br>D10<br>D9<br>D8<br>D7<br>D6<br>D5<br>D4<br>D3<br>D2<br>D1<br>D0 LSB | B6/2<br>A5/3<br>B5/4<br>A4/5<br>B4/6<br>A3/7<br>B3/8<br>A2/9<br>B2/11<br>C1/12<br>C2/13<br>D1/14<br>D2/15<br>E1/16<br>E2/17<br>F1/18 | I/O/Z | Parallel data bus D15 (MSB) through D0 (LSB). Multiplexed to transfer data between the TMS320C2x and external data/program memory or I/O devices. Placed in high-impedance state when not outputting or when $\overline{RS}$ or $\overline{HOLD}$ is asserted. |
| | | INTERFACE CONTROL SIGNALS | |
| $\overline{DS}$<br>$\overline{PS}$<br>$\overline{IS}$ | K10/45<br>J10/47<br>J11/46 | O/Z | Data, program, and I/O space select signals. Always high unless low level asserted for communicating to a particular external space. Placed in high-impedance state in the hold mode. |
| READY | B8/66 | I | Data ready input. Indicates that an external device is prepared for the bus transaction to be completed. If the device is not ready (READY = 0), the TMS320C2x waits one cycle and checks READY again. READY also indicates a bus grant to an external device after a $\overline{BR}$ (bus request) signal. |
| R/$\overline{W}$ | H11/48 | O/Z | Read/write signal. Indicates transfer direction when communicating to an external device. Normally in read mode (high), unless low level asserted for performing a write operation. Placed in high-impedance state in the hold mode. |
| $\overline{STRB}$ | H10/49 | O/Z | Strobe signal. Always high unless asserted low to indicate an external bus cycle. Placed in high-impedance state in the hold mode. |

† Input/Output/High-impedance state

## TMS320C2x Internal Hardware

| UNIT | SYMBOL | FUNCTION |
|---|---|---|
| Accumulator | ACC (31-0)<br>ACCH(31-16)<br>ACCL(15-0) | A 32-bit accumulator split in two halves: ACCH (accumulator high) and ACCL (accumulator low). Used for storage of ALU output. |
| Arithmetic Logic Unit | ALU | A 32-bit two's-complement arithmetic logic unit having two 32-bit input ports and one 32-bit output port feeding the accumulator. |
| Auxiliary Register Arithmetic Unit | ARAU | A 16-bit unsigned arithmetic unit used to perform operations on auxiliary register data. |
| Auxiliary Register File | AR0-AR7<br>(15-0) | A register file containing five/eight 16-bit auxiliary registers (AR0-AR7), used for addressing data memory, temporary storage, or integer arithmetic processing through the ARAU. |
| Auxiliary Register File Bus | AFB(15-0) | A 16-bit bus that carries data from the AR pointed to by the ARP. |
| Auxiliary Register Pointer | ARP(2-0) | A 3-bit register used to select one of five/eight auxiliary registers. |
| Auxiliary Register Pointer Buffer | ARB(2-0) | A 3-bit register used to buffer the ARP. Each time the ARP is loaded, the old value is written to the ARB, except during an LST (load status register) instruction. When the ARB is loaded with an LST1, the same value is also copied into ARP. |
| Central Arithmetic Logic Unit | CALU | The grouping of the ALU, multiplier, accumulator, and scaling shifter. |
| Data Bus | D(15-0) | A 16-bit bus used to route data. |
| Data Memory Address Bus | DAB(15-0) | A 16-bit bus that carries the data memory address. |
| Data Memory Page Pointer | DP(8-0) | A 9-bit register pointing to the address of the current page. Data pages are 128 words each, resulting in 512 pages of addressable data memory space (some locations are reserved). |
| Direct Data Memory Address Bus | DRB(15-0) | A 16-bit bus that carries the 'direct' address for the data memory, which is the concatenation of the DP register with the seven LSBs of the instruction. |
| Global Memory Allocation Register | GREG(7-0) | An 8-bit memory-mapped register for allocating the size of the global memory space. |
| Instruction Register | IR(15-0) | A 16-bit register used to store the currently executing instruction. |
| Interrupt Flag Register | IFR(5-0) | A 6-bit flag register used to latch the active-low external user interrupts $\overline{\text{INT}}$(2-0) and the internal interrupts XINT/RINT (serial port transmit/receive) and TINT (timer) interrupts. The IFR is not accessible through software. |
| Interrupt Mask Register | IMR(5-0) | A 6-bit memory-mapped register used to mask interrupts. |

†TMS320C25 only.

| UNIT | SYMBOL | FUNCTION |
|---|---|---|
| Microcall Stack† | MCS (15-0) | A single-word stack that temporarily stores the contents of the PFC while the PFC is being used to address data memory with the block move (BLKD/BLKP), multiply-accumulate (MAC/MACD), and table read/write (TBLR/TBLW) instructions. |
| Multiplier | MULT | A 16 x 16-bit parallel multiplier. |
| Period Register | PRD (15-0) | A 16-bit memory-mapped register used to reload the timer. |
| Prefetch Counter† | PFC (15-0) | A 16-bit counter used to prefetch program instructions. The PFC contains the address of the instruction currently being prefetched. It is updated when a new prefetch is initiated. The PFC is also used to address data memory when using the block move (BLKD/BLKP), multiply-accumulate (MAC/MACD), and table read/write (TBLR/TBLW) instructions. |
| Product Register | PR(31-0) | A 32-bit product register used to hold the multiplier product. The PR on the TMS320C25 can also be accessed as the most or least significant words using the SPH/SPL (store P register high/low) instructions. |
| Program Bus | P(15-0) | A 16-bit bus used to route instructions (and data for the MAC and MACD instructions). |
| Program Counter | PC (15-0) | A 16-bit program counter used to address program memory. The PC always contains the address of the next instruction to be executed. The PC contents are updated following each instruction decode operation. On the TMS32020, the operations of the TMS320C25 prefetch counter are performed by the program counter. |
| Program Memory Address Bus | PAB(15-0) | A 16-bit bus that carries the program memory address. |
| Queue Instruction Register† | QIR(15-0) | A 16-bit register used to store prefetched instructions. |
| Random Access Memory (data or program) | RAM (B0) | A RAM block with 256 x 16 locations configured either as data or program memory. |
| Random Access Memory (data only) | RAM (B1) | A data RAM block, organized as 256 x 16 locations. |
| Random Access Memory (data only) | RAM (B2) | A data RAM block, organized as 32 x 16 locations. |
| Repeat Counter | RPTC (7-0) | An 8-bit counter to control the repeated execution of a single instruction. |
| Serial Port Data Receive Register | DRR(15-0) | A 16-bit memory-mapped serial port data receive register. Only the eight LSBs are used in the byte mode. |
| Serial Port Data Transmit Register | DXR(15-0) | A 16-bit memory-mapped serial port data transmit register. Only the eight LSBs are used in the byte mode. |
| Serial Port Receive Shift Register† | RSR(15-0) | A 16-bit register used to shift in serial port data from the RX pin. RSR contents are sent to the DRR after a serial transfer is completed. RSR is not directly accessible through software. |

†TMS320C25 only.

# TMS320C2x Internal Hardware

| UNIT | SYMBOL | FUNCTION |
|---|---|---|
| Serial Port Transmit Shift Register† | XSR(15-0) | A 16-bit register used to shift out serial port data onto the DX pin. XSR contents are loaded from DXR at the beginning of a serial port transmit operation. XSR is not directly accessible through software. |
| Shifters | – | Shifters are located at the ALU input, the accumulator output, and the product register output. An in-place shifter is also located within the accumulator. |
| Stack | Stack(15-0) | A 4/8 x 16 hardware stack used to store the PC during interrupts or calls. The ACCL and data memory values may also be pushed onto and popped from the stack. |
| Status Registers | ST0,ST1 (15-0) | Two 16-bit status registers that contain status and control bits. |
| Temporary Register | TR(15-0) | A 16-bit register that holds either an operand for the multiplier or a shift code for the scaling shifter. |
| Timer | TIM (15-0) | A 16-bit memory-mapped timer (counter) for timing control. |

†TMS320C25 only.

# Instruction Set

| AUXILIARY REGISTERS AND DATA PAGE POINTER INSTRUCTIONS | | | | | | | |
|---|---|---|---|---|---|---|---|
| Mnemonic and Description | | Words | 16-Bit Opcode MSB | | | | LSB |
| ADRK‡ | Add to auxiliary register short immediate | 1 | 0111 | 1110 | KKKK | KKKK | |
| CMPR† | Compare auxiliary register with auxiliary register AR0 | 1 | 1100 | 1110 | 0101 | 00KK | |
| LAR | Load auxiliary register | 1 | 0011 | 0RRR | I DDD | DDDD | |
| LARK | Load auxiliary register short immediate | 1 | 1100 | 0RRR | KKKK | KKKK | |
| LARP | Load auxiliary register pointer | 1 | 0101 | 0101 | 1000 | 1RRR | |
| LDP | Load data memory page pointer | 1 | 0101 | 0010 | I DDD | DDDD | |
| LDPK | Load data memory page pointer immediate | 1 | 1100 | 100K | KKKK | KKKK | |
| LRLK† | Load auxiliary register long immediate | 2 | 1101 | 0RRR | 0000 | 0000 | |
| MAR | Modify auxiliary register | 1 | 0101 | 0101 | I DDD | DDDD | |
| SAR | Store auxiliary register | 1 | 0111 | 0RRR | I DDD | DDDD | |
| SBRK‡ | Subtract from auxiliary register short immediate | 1 | 0111 | 1111 | KKKK | KKKK | |
| T REGISTER, P REGISTER, AND MULTIPLY INSTRUCTIONS | | | | | | | |
| Mnemonic and Description | | Words | 16-Bit Opcode MSB | | | | LSB |
| APAC | Add P register to accumulator | 1 | 1100 | 1110 | 0001 | 0101 | |
| LPH† | Load high P register | 1 | 0101 | 0011 | I DDD | DDDD | |
| LT | Load T register | 1 | 0011 | 1100 | I DDD | DDDD | |
| LTA | Load T register and accumulate previous product | 1 | 0011 | 1101 | I DDD | DDDD | |
| LTD | Load T register, accumulate previous product, and move data | 1 | 0011 | 1111 | I DDD | DDDD | |
| LTP† | Load T register and store P register in accumulator | 1 | 0011 | 1110 | I DDD | DDDD | |
| LTS† | Load T register and subtract previous product | 1 | 0101 | 1011 | I DDD | DDDD | |
| MAC† | Multiply and accumulate | 2 | 0101 | 1101 | I DDD | DDDD | |
| MACD† | Multiply and accumulate with data move | 2 | 0101 | 1100 | I DDD | DDDD | |
| MPY | Multiply (with T register, store product in P register) | 1 | 0011 | 1000 | I DDD | DDDD | |
| MPYA‡ | Multiply and accumulate previous product | 1 | 0011 | 1010 | I DDD | DDDD | |
| MPYK | Multiply immediate | 1 | 101K | KKKK | KKKK | KKKK | |
| MPYS‡ | Multiply and subtract previous product | 1 | 0011 | 1011 | I DDD | DDDD | |
| MPYU‡ | Multiply unsigned | 1 | 1100 | 1111 | I DDD | DDDD | |
| PAC | Load accumulator with P register | 1 | 1100 | 1110 | 0001 | 0100 | |
| SPAC | Subtract P register from accumulator | 1 | 1100 | 1110 | 0001 | 0110 | |
| SPH‡ | Store high P register | 1 | 0111 | 1101 | I DDD | DDDD | |
| SPL‡ | Store low P register | 1 | 0111 | 1100 | I DDD | DDDD | |
| SPM† | Set P register output shift mode | 1 | 1100 | 1110 | 0000 | 10KK | |
| SQRA† | Square and accumulate | 1 | 0011 | 1001 | I DDD | DDDD | |
| SQRS† | Square and subtract previous product | 1 | 0101 | 1010 | I DDD | DDDD | |

†These instructions are specific to the TMS320C2x instruction set.
‡These instructions are specific to the TMS320C25 instruction set.

## Instruction Set

| ACCUMULATOR MEMORY REFERENCE INSTRUCTIONS | | | |
|---|---|---|---|
| Mnemonic | Description | Words | 16-Bit Opcode (MSB … LSB) |
| ABS | Absolute value of accumulator | 1 | 1100 1110 0001 1011 |
| ADD | Add to accumulator with shift | 1 | 0000 SSSS I DDD DDDD |
| ADDC‡ | Add to accumulator with carry | 1 | 0100 0011 I DDD DDDD |
| ADDH | Add to high accumulator | 1 | 0100 1000 I DDD DDDD |
| ADDK‡ | Add to accumulator short immediate | 1 | 1100 1100 KKKK KKKK |
| ADDS | Add to low accumulator with sign-extension suppressed | 1 | 0100 1001 I DDD DDDD |
| ADDT† | Add to accumulator with shift specified by T register | 1 | 0100 1010 I DDD DDDD |
| ADLK† | Add to accumulator long immediate with shift | 2 | 1101 SSSS 0000 0010 |
| AND | AND with accumulator | 1 | 0100 1110 I DDD DDDD |
| ANDK† | AND immediate with accumulator with shift | 2 | 1101 SSSS 0000 0100 |
| CMPL† | Complement accumulator | 1 | 1100 1110 0010 0111 |
| LAC | Load accumulator with shift | 1 | 0010 SSSS I DDD DDDD |
| LACK | Load accumulator short immediate | 1 | 1100 1010 KKKK KKKK |
| LACT† | Load accumulator with shift specified by T register | 1 | 0100 0010 I DDD DDDD |
| LALK† | Load accumulator long immediate with shift | 2 | 1101 SSSS 0000 0001 |
| NEG† | Negate accumulator | 1 | 1100 1110 0010 0011 |
| NORM† | Normalize contents of accumulator | 1 | 1100 1110 1010 0010 |
| OR | OR with accumulator | 1 | 0100 1101 I DDD DDDD |
| ORK† | OR immediate with accumulator with shift | 2 | 1101 SSSS 0000 0101 |
| ROL‡ | Rotate accumulator left | 1 | 1100 1110 0011 0100 |
| ROR‡ | Rotate accumulator right | 1 | 1100 1110 0011 0101 |
| SACH | Store high accumulator with shift | 1 | 0110 1XXX I DDD DDDD |
| SACL | Store low accumulator with shift | 1 | 0110 0XXX I DDD DDDD |
| SBLK† | Subtract from accumulator long immediate with shift | 2 | 1101 SSSS 0000 0011 |
| SFL† | Shift accumulator left | 1 | 1100 1110 0001 1000 |
| SFR† | Shift accumulator right | 1 | 1100 1110 0001 1001 |
| SUB | Subtract from accumulator with shift | 1 | 0001 SSSS I DDD DDDD |
| SUBB‡ | Ssubtract from accumulator with borrow | 1 | 0100 1111 I DDD DDDD |
| SUBC | Conditional subtract | 1 | 0100 0111 I DDD DDDD |
| SUBH | Subtract from high accumulator | 1 | 0100 0100 I DDD DDDD |
| SUBK‡ | Ssubtract from accumulator short immediate | 1 | 1100 1101 KKKK KKKK |
| SUBS | Subtract from low accumulator with sign extension suppressed | 1 | 0100 0101 I DDD DDDD |
| SUBT† | Subtract from accumulator with shift specified by T register | 1 | 0100 0110 I DDD DDDD |
| XOR | Exclusive-OR with accumulator | 1 | 0100 1100 I DDD DDDD |
| XORK† | Exclusive-OR immediate with accumulator with shift | 2 | 1101 SSSS 0000 0110 |
| ZAC | Zero accumulator | 1 | 1100 1010 0000 0000 |
| ZALH | Zero low accumulator and load high accumulator | 1 | 0100 0000 I DDD DDDD |
| ZALR‡ | Zero low accumulator and load high accumulator with rounding | 1 | 0111 1011 I DDD DDDD |
| ZALS | Zero accumulator and load low accumulator with sign extension suppressed | 1 | 0100 0001 I DDD DDDD |

†These instructions are specific to the TMS320C2x instruction set.
‡These instructions are specific to the TMS320C25 instruction set.

# Instruction Set

| CONTROL INSTRUCTIONS | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Mnemonic and Description | | Words | 16-Bit Opcode MSB | | | | | LSB |
| BIT† | Test bit | 1 | 1001 | BBBB | I DDD | DDDD | | |
| BITF† | Test bit specified by T register | 1 | 0101 | 0111 | I DDD | DDDD | | |
| CNFD† | Configure block as data memory | 1 | 1100 | 1110 | 0000 | 0100 | | |
| CNFP† | Configure block as program memory | 1 | 1100 | 1110 | 0000 | 0101 | | |
| DINT | Disable interrupt | 1 | 1100 | 1110 | 0000 | 0001 | | |
| EINT | Enable interrupt | 1 | 1100 | 1110 | 0000 | 0000 | | |
| IDLE† | Idle until interrupt | 1 | 1100 | 1110 | 0001 | 1111 | | |
| LST | Load status register ST0 | 1 | 0101 | 0000 | I DDD | DDDD | | |
| LST1† | Load status register ST1 | 1 | 0101 | 0001 | I DDD | DDDD | | |
| NOP | No operation | 1 | 0101 | 0101 | 0000 | 0000 | | |
| POP | Pop top of stack to low accumulator | 1 | 1100 | 1110 | 0001 | 1101 | | |
| POPD† | Pop top of stack to data memory | 1 | 0111 | 1010 | I DDD | DDDD | | |
| PSHD† | Push data memory value onto stack | 1 | 0101 | 0100 | I DDD | DDDD | | |
| PUSH | Push low accumulator onto stack | 1 | 1100 | 1110 | 0001 | 1100 | | |
| RC‡ | Reset carry bit | 1 | 1100 | 1110 | 0011 | 0000 | | |
| RHM‡ | Reset hold mode | 1 | 1100 | 1110 | 0011 | 1000 | | |
| ROVM | Reset overflow mode | 1 | 1100 | 1110 | 0000 | 0010 | | |
| RPT† | Repeat instruction as specified by data memory value | 1 | 0100 | 1011 | I DDD | DDDD | | |
| RPTK† | Repeat instruction as specified by immediate value | 1 | 1100 | 1011 | KKKK | KKKK | | |
| RSXM† | Reset sign-extension mode | 1 | 1100 | 1110 | 0000 | 0110 | | |
| RTC‡ | Reset test/control flag | 1 | 1100 | 1110 | 0011 | 0010 | | |
| SC‡ | Set carry bit | 1 | 1100 | 1110 | 0011 | 0001 | | |
| SHM‡ | Set hold mode | 1 | 1100 | 1110 | 0011 | 1001 | | |
| SOVM | Set overflow mode | 1 | 1100 | 1110 | 0000 | 0011 | | |
| SST | Store status register ST0 | 1 | 0111 | 1000 | I DDD | DDDD | | |
| SST1† | Store status register ST1 | 1 | 0111 | 1001 | I DDD | DDDD | | |
| SSXM† | Set sign-extension mode | 1 | 1100 | 1110 | 0000 | 0111 | | |
| STC‡ | Set test/control flag | 1 | 1100 | 1110 | 0011 | 0011 | | |

†These instructions are specific to the TMS320C2x instruction set.
‡These instructions are specific to the TMS320C25 instruction set.

# Instruction Set

| BRANCH/CALL INSTRUCTIONS | | | | | | | |
|---|---|---|---|---|---|---|---|
| Mnemonic and Description | | Words | 16-Bit Opcode MSB | | | | LSB |
| B | Branch unconditionally | 2 | 1111 | 1111 | 1DDD | DDDD | |
| BACC† | Branch to address specified by accumulator | 1 | 1100 | 1110 | 0010 | 0101 | |
| BANZ | Branch on auxiliary register not zero | 2 | 1111 | 1011 | 1DDD | DDDD | |
| BBNZ† | Branch if TC bit ≠ 0 | 2 | 1111 | 1001 | 1DDD | DDDD | |
| BBZ† | Branch if TC bit = 0 | 2 | 1111 | 1000 | 1DDD | DDDD | |
| BC‡ | Branch on carry | 2 | 0101 | 1110 | 1DDD | DDDD | |
| BGEZ | Branch if accumulator ≥ 0 | 2 | 1111 | 0100 | 1DDD | DDDD | |
| BGZ | Branch if accumulator > 0 | 2 | 1111 | 0001 | 1DDD | DDDD | |
| BIOZ | Branch on I/O status = 0 | 2 | 1111 | 1010 | 1DDD | DDDD | |
| BLEZ | Branch if accumulator ≤ 0 | 2 | 1111 | 0010 | 1DDD | DDDD | |
| BLZ | Branch if accumulator < 0 | 2 | 1111 | 0011 | 1DDD | DDDD | |
| BNC‡ | Branch on no carry | 2 | 0101 | 1111 | 1DDD | DDDD | |
| BNV† | Branch if no overflow | 2 | 1111 | 0111 | 1DDD | DDDD | |
| BNZ | Branch if accumulator ≠ 0 | 2 | 1111 | 0101 | 1DDD | DDDD | |
| BV | Branch on overflow | 2 | 1111 | 0000 | 1DDD | DDDD | |
| BZ | Branch if accumulator = 0 | 2 | 1111 | 0110 | 1DDD | DDDD | |
| CALA | Call subroutine indirect | 1 | 1100 | 1110 | 0010 | 0100 | |
| CALL | Call subroutine | 2 | 1111 | 1110 | 1DDD | DDDD | |
| RET | Return from subroutine | 1 | 1100 | 1110 | 0010 | 0110 | |
| TRAP† | Software interrupt | 1 | 1100 | 1110 | 0001 | 1110 | |
| I/O AND DATA MEMORY OPERATIONS | | | | | | | |
| Mnemonic and Description | | Words | 16-Bit Opcode MSB | | | | LSB |
| BLKD† | Block move from data memory to data memory | 2 | 1111 | 1101 | IDDD | DDDD | |
| BLKP† | Block move from program memory to data memory | 2 | 1111 | 1100 | IDDD | DDDD | |
| DMOV | Data move in data memory | 1 | 0101 | 0110 | IDDD | DDDD | |
| FORT† | Format serial port registers | 1 | 1100 | 1110 | 0000 | 111K | |
| IN | Input data from port | 1 | 1000 | AAAA | IDDD | DDDD | |
| OUT | Output data to port | 1 | 1110 | AAAA | IDDD | DDDD | |
| RFSM‡ | Reset serial port frame synchronization mode | 1 | 1100 | 1110 | 0011 | 0110 | |
| RTXM† | Reset serial port transmit mode | 1 | 1100 | 1110 | 0010 | 0000 | |
| RXF† | Reset external flag | 1 | 1100 | 1110 | 0000 | 1100 | |
| SFSM‡ | Set serial-port frame synchronization mode | 1 | 1100 | 1110 | 0011 | 0111 | |
| STXM† | Set serial port transmit mode | 1 | 1100 | 1110 | 0010 | 0001 | |
| SXF† | Set external flag | 1 | 1100 | 1110 | 0000 | 1101 | |
| TBLR | Table read | 1 | 0101 | 1000 | IDDD | DDDD | |
| TBLW | Table write | 1 | 0101 | 1001 | IDDD | DDDD | |

†These instructions are specific to the TMS320C2x instruction set.
‡These instructions are specific to the TMS320C25 instruction set.

```
        AORG  0              ; ADDR  CODE
        B     INIT           ; 0000  FF80
                                     ; 0001  0020
        B     INT0           ; 0002  FF80
                                     ; 0003  003E
        B     INT1           ; 0004  FF80
                                     ; 0005  003F
        B     INT2           ; 0006  FF80
                                     ; 0007  0040
        AORG  >18
        B     TINT           ; 0018  FF80
                                     ; 0019  004C
        B     RINT           ; 001A  FF80
                                     ; 001B  0055
        B     XINT           ; 001C  FF80
                                     ; 001D  0056
        B     TRAP           ; 001E  FF80
                                     ; 001F  0057
        AORG  >20
INIT    DINT                 ; 0020  CE01
        ROVM                 ; 0021  CE02
        LDPK  0              ; 0022  C800
        LARP  7              ; 0023  558F
        LACK  >08            ; 0024  CA08
        SACL  4              ; 0025  6004
*
* INTERNAL DATA MEMORY INITIALZATION
*
        ZAC                  ; 0026  CA00
        LARK  AR7,>60        ; 0027  C760
        RPTK  31             ; 0028  CD1F
        SACL  *+             ; 0029  60A7
        LRLK  AR7,>200       ; 002A  D700
                                     ; 002B  0200
        RPTK  255            ; 002C  CDFF
        SACL  *+             ; 002D  60A7
        LRLK  AR7,>300       ; 002E  D700
                                     ; 002F  0300
        RPTK  255            ; 0030  CDFF
        SACL  *+             ; 0031  60A7
        LALK  >33            ; 0032  D001
                                     ; 0033  0033
        SACL  DMA3           ; 0034  6003
        LACK  8              ; 0035  CA08
        OR    DMA4           ; 0036  4D04
        SACL  DMA4           ; 0037  6004
        LRLK  AR7,>400       ; 0038  D700
        RPTK  255            ; 0039  CDFF
        SACL  *+             ; 003A  60A7
WAIT    EINT                 ; 003B  CE00
        B     WAIT           ; 003C  FF80
                                     ; 003D  003B
INT0    RET                  ; 003E  CE26
INT1    RET                  ; 003F  CE26
INT2    LDPK  8              ; 0040  C808
        IN    DMA0,PA6       ; 0041  8600
        LAC   DMA0           ; 0042  2000
        ANDK  >00FF          ; 0043  D004
                                     ; 0044  00FF
        SUBK  >88            ; 0045  CD88
        BZ    OUTP           ; 0046  F680
                                     ; 0047  0049
        RET                  ; 0048  CE26
```

```
OUTP    LDPK  0             ; 0049  C800
        OUT   DMA3,PA7      ; 004A  E703
        RET                 ; 004B  CE26
TINT    BIOZ  SET1          ; 004C  FC80
                                    ; 004D  0051
        RXF                 ; 004E  CE0C
        EINT                ; 004F  CE00
        RET                 ; 0050  CE26
SET1    SXF                 ; 0051  CE0D
        OUT   DMA4,PA7      ; 0052  E704
        EINT                ; 0053  CE00
        RET                 ; 0054  CE26
RINT    RET                 ; 0055  CE26
XINT    RET                 ; 0056  CE26
TRAP    RET                 ; 0057  CE26
```

```
/****------------------****/
/* File : C25.C          */
/****------------------****/

  #define NUMPORT  0x200     /* define decoded address */
  #define PORTA    0
  #define PORTB    2
  #define PORTC    4
  #define CNTRL    6
  #define PSREAD1  0xfff0    /* 1st control word to read program memory */
  #define DSREAD1  0xffe8    /* 1st control word to read data memory */
  #define PSREAD2  0xfff4    /* 2nd control word to read program memory */
  #define DSREAD2  0xffec    /* 2nd control word to read data memory */
  #define PSLOAD   0xfff5    /* control word to write program memory */
  #define DSLOAD   0xffee    /* control word to write data memory */
  #define TOTMS    0xffd2    /* control word for TMSs' running */
  #define RS       0x208     /* control word to reset TMS */
  #define RW       0x210     /* control word to write information to memory */
/*  #define IBMIACK  0X218
  #define COMMU1   0X220
  #define COMMU2   0X228*/
  /*-------PC interrupt values----------------*/
  #define IRQ      5                 /* PC interrupt used for c25 */
  #if (IRQ < 8)
  #define PC_INT   (IRQ+8)
  #define PC_MASK  (~(1<<IRQ))       /* for unmask of PC 8259 */
  #define PC_PIC01 0x21              /* 8259 interrupt controller XT */
  #define PC_PIC00 0x20              /* 0-7 interrupts */
  #else
  #define PCINT    (104+IRQ)
  #define PC_MASK  (~(1<<(IRQ-8)))
  #define PC_PIC01 0xA1
  #define PC_PIC00 0xA0
  #endif

  #define PC_EOI   0x20              /* end of intrrupt */

  #include <stdio.h>
  #include <stdlib.h>
  #include <dos.h>
  #include <conio.h>
  #include <bios.h>

  static void (interrupt far *old_comm_int) ();
  static void interrupt far c25_comm ();
  static int count =0;

  static int  old_int_stat,breq_flag,status;

  void init_c25_comm() {
              disable ();
              old_comm_int = getvect(PC_INT);
              setvect(PC_INT,c25_comm);
              /*---setup pc 8259 ---------*/
              old_int_stat = inp(PC_PIC01);
              outp(PC_PIC01,(old_int_stat & PC_MASK));
              enable();
              breq_flag = 0;
  }

  void restore_c25_comm() {
              disable();
              setvect(PC_INT,old_comm_int);
```

```
            outp(PC_PICOl,old_int_stat);
            enable();
  }

  static void interrupt far c25_comm(void) { /* C25 interrupt service routine */
  disable();
/*  status = inp(IBMIACK);*/
  sound(3000); delay(10); nosound();
  count++;
  outp(PC_PICOO,PC_EOI);
  enable();
  (*old_comm_int) ();
  }


  main()
  {
   char key;
   int i;
   int addr=0x0000;
   int data[100],datal[100];

            window(1,1,80,25);textbackground(BLACK);textcolor(WHITE);
            clrscr();

            outport(NUMPORT+CNTRL,0x8080);
            outport(NUMPORT+PORTC,PSLOAD);
            data[0]=0xff80;
            data[1]=0x0020;
            data[2]=0xff80;
            data[3]=0x003b;
            data[4]=0xff80;
            data[5]=0x003c;
            data[6]=0xff80;
            data[7]=0x003d;
            for(i=0,addr=0;i<8;i++,addr++)
            {
              outport(NUMPORT+PORTB,addr);
              outport(NUMPORT+PORTA,data[i]);
              outport(RW,0x00);
            }
            data[8]=0xff80;
            data[9]=0x003e;
            data[10]=0xff80;
            data[11]=0x0046;
            data[12]=0xff80;
            data[13]=0x0047;
            data[14]=0xff80;
            data[15]=0x0048;
            for(i=8,addr=0x0018;i<16;i++,addr++)
            {
              outport(NUMPORT+PORTB,addr);
              outport(NUMPORT+PORTA,data[i]);
              outport(RW,0x00);
            }
            data[16]=0xce01;
            data[17]=0xce02;
            data[18]=0xc800;
            data[19]=0x558f;
            data[20]=0xca08;
            data[21]=0x6004;
            data[22]=0xca00;
            data[23]=0xc760;
```

```
        data[24]=0xcd1f;
        data[25]=0x60a7;
        data[26]=0xd700;
        data[27]=0x0020;
        data[28]=0xcdff;
        data[29]=0x60a7;
        data[30]=0xd700;
        data[31]=0x0300;
        data[32]=0xcdff;
        data[33]=0x60a7;
        data[34]=0xd001;
        data[35]=0x0100;    /* Set pulse-frequency */
        data[36]=0x6003;
        data[37]=0xca08;
        data[38]=0x4d04;
        data[39]=0x6004;
        data[40]=0xce00;
        data[41]=0xff80;
        data[42]=0x0038;
        data[43]=0xce26;
        data[44]=0xce26;
        data[45]=0xce26;
        data[46]=0xfa80;
        data[47]=0x0043;
        data[48]=0xce0c;
        data[49]=0xce00;
        data[50]=0xce26;
        data[51]=0xce0d;
        data[52]=0xce00;
        data[53]=0xce26;
        data[54]=0xce26;
        data[55]=0xce26;
        data[56]=0xce26;
        data[57]=0xce26;
        for(i=16,addr=0x0020;i<59;i++,addr++)
        {
          outport(NUMPORT+PORTB,addr);
          outport(NUMPORT+PORTA,data[i]);
          outport(RW,0x00);
        }
        outport(NUMPORT+CNTRL,0x9090);
        for(i=0,addr=0x0000;i<8;i++,addr++)
        {
          outport(NUMPORT+PORTC,PSREAD1);
          outport(NUMPORT+PORTB,addr);
          data1[i]=inport(NUMPORT);
          outport(NUMPORT+PORTC,PSREAD2);
          if (data[i]!=data1[i])  {
      printf("\n%x",data1[i]);
                    printf("    DATA error");
                    }
        }
        for(i=8,addr=0x0018;i<16;i++,addr++)
        {
          outport(NUMPORT+PORTC,PSREAD1);
          outport(NUMPORT+PORTB,addr);
          data1[i]=inport(NUMPORT);
          outport(NUMPORT+PORTC,PSREAD2);
    if (data[i]!=data1[i])  {
                    printf("\n%x",data1[i]);
                    printf("    DATA error");
                    }
        }
```

```
        for(i=16,addr=0x0020;i<59;i++,addr++)
        {
          outport(NUMPORT+PORTC,PSREAD1);
          outport(NUMPORT+PORTB,addr);
          data1[i]=inport(NUMPORT);
          outport(NUMPORT+PORTC,PSREAD2);

          if (data[i]!=data1[i])  {
       printf("\n%x",data1[i]);
                    printf("    DATA error");
                    }
          }

    printf("\npress any key to  switch to tms");
/*          init_c25_comm();*/
          while( getch() != 'q'){
                              printf("\nTms is running");
                              outport(NUMPORT+PORTC,TOTMS);
                              delay(0);
                              outport(RS,0x0000);
                              /*delay(100);
                              outport(NUMPORT+PORTC,TOTMS);
                              outport(RS,0x0000);*/
          }
/*          restore_c25_comm();*/
          printf("\nc25 intn %d",count);
          getch();
    outport(NUMPORT+CNTRL,0x9090);
          for(i=0,addr=0x0000;i<8;i++,addr++)
          {
          outport(NUMPORT+PORTC,PSREAD1);
          outport(NUMPORT+PORTB,addr);
          data1[i]=inport(NUMPORT);
          outport(NUMPORT+PORTC,PSREAD2);
          printf("\naddr %x %x",addr,data1[i]);
          if (data[i]!=data1[i])  {
                    printf("    DATA error");
                    getch();
                    }
          }
          for(i=8,addr=0x0018;i<16;i++,addr++)
          {
            outport(NUMPORT+PORTC,PSREAD1);
            outport(NUMPORT+PORTB,addr);
            data1[i]=inport(NUMPORT);
            outport(NUMPORT+PORTC,PSREAD2);
            printf("\naddr %x %x",addr,data1[i]);
            if (data[i]!=data1[i])  {
                    printf("    DATA error");
                    getch();
                    }
          }
          for(i=16,addr=0x0020;i<46;i++,addr++)
          {
            outport(NUMPORT+PORTC,PSREAD1);
            outport(NUMPORT+PORTB,addr);
            data1[i]=inport(NUMPORT);
            outport(NUMPORT+PORTC,PSREAD2);
            printf("\naddr %x %x",addr,data1[i]);
            if (data[i]!=data1[i])  {
                    printf("    DATA error");
                    getch();
                    }
```

```
    }
/*  for(i=0,addr=0x0000;i<60;i++,addr++)
    {
     outport(NUMPORT+PORTC,DSREAD1);
     outport(NUMPORT+PORTB,addr);
     data1[i]=inport(NUMPORT);
     outport(NUMPORT+PORTC,DSREAD2);
     printf("\n%x",data1[i]);
     }*/
     getch();
    }
```

# เอกสารอ้างอิง


1.) Texas Instrument,"Second-Generation TMS 320 User's Guide", 1987

2.) Yasuhiko Dote,"Servo Motor and Motion Control Using DSP", Prentice Hall,1990

3.) Burr_Brown Corporation,"BURR-BROWN Integrated Circuit Data Book Supplement", 1987

4.) นายชินภัทร นันทจิวากรชัย และ นายสุชาติ ผโลปกรณ์,"ดิจิตอลซิกแนลโปรเซสซิง",คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง,2534

# กิตติกรรมประกาศ

ปริญญานิพนธ์นี้สำเร็จลงได้ด้วยความช่วยเหลือเป็นอย่างดีจากอาจารย์ สุรสิทธิ์ วรรณไกรโรจน์ และอาจารย์ กวิน สนธิเพิ่มพูน ซึ่งเป็นอาจารย์ที่ปรึกษาตลอดการทำปริญญานิพนธ์นี้ ที่ให้คำปรึกษาอันมีค่าและเป็นประโยชน์อย่างมาก และขอขอบคุณพี่ไก่ ไจดี ที่ให้คำแนะนำเมื่อมีปัญหามาโดยตลอด และขอบคุณเพื่อน ๆ ทุกคนที่ให้หยิบยืมอุปกรณ์และให้ความช่วยเหลืออย่างดีเสมอมา