บทที่ 4

ตัวประมวลผลสัญญาณดิจิตอล

ตัวประมวลผลสัญญาณดิจิตอลหรือเรียกย่อกันโดยทั่วไปว่าดีเอสพี เป็นวงจรรวมที่ถูกออกแบบขึ้นมาเป็นพิเศษเพื่อให้มีความเหมาะสมในการประมวลผลสัญญาณดิจิตอล ลักษณะพิเศษที่เป็นคุณสมบัติร่วมของดีเอสพีทุกตระกูลก็คือ มีความสามารถในการคำนวณทางคณิตศาสตร์ได้อย่างรวดเร็ว ทั้งนี้ก็เพื่อให้มันสามารถจัดการกับสัญญาณดิจิตอลรับเข้าและสร้างสัญญาณส่งออกได้อย่างทันเวลา ในปัจจุบันมีดีเอสพีที่ใช้กันอย่างกว้างขวางอยู่หลายตระกูลด้วยกันได้แก่ ตระกูล TMS320 ของบริษัทเท็กซัสอินสตรูเมนต์ ตระกูล ADSP2100 ของบริษัทอนาล็อกดีไวส์ ตระกูล 56000 ของบริษัทโมโตโรลา และยังมีดีเอสพีในตระกูลอื่น ๆ ของอีกหลายบริษัท ทุกบริษัทต่างก็มุ่งพัฒนาดีเอสพีเพื่อให้มีความสามารถในการประมวลผลสัญญาณด้วยประสิทธิภาพสูงสุด มีการนำเทคนิคต่าง ๆ เข้ามาใช้เพื่อเร่งความเร็วในการประมวลผล ที่พบเห็นได้เสมอได้แก่ การออกแบบระบบบัสข้อมูล (data bus) แยกจากบัสโปรแกรม (program bus) เพื่อทำให้ดีเอสพีสามารถอ่านเขียนหน่วยความจำข้อมูลและหน่วยความจำโปรแกรมได้พร้อม ๆ กันระบบบัสแบบนี้มีชื่อเรียกว่าสถาปัตยกรรมฮาร์วาร์ด การยุบรวมหน่วยความจำมาไว้ภายในตัวดีเอสพีเพื่อลดเวลาในการอ่านเขียนหน่วยความจำลง การใช้หน่วยความจำความเร็วสูง การใช้เทคนิคการทำงานแบบขนานภายในตัวดีเอสพีเป็นต้น

โดยทั่วไปเราสามารถแบ่งดีเอสพีได้เป็นสองกลุ่มด้วยกันคือ

1. ดีเอสพีชนิดทศนิยมตายตัว (fixed point DSP)
2. ดีเอสพีชนิดทศนิยมลอยตัว (floating point DSP)

ทั้งสองกลุ่มมีความแตกต่างที่เห็นชัดในเรื่องรีจิสเตอร์ที่ใช้ในการเก็บเลขจำนวนจริงที่เป็นทศนิยม ดีเอสพีในกลุ่มทศนิยมตายตัวโดยทั่วไปใช้รีจิสเตอร์ที่มีจำนวนบิตน้อยกว่าในกลุ่มทศนิยมลอยตัว อย่างเช่นตระกูล ADSP2100 ของบริษัทอนาล็อกดีไวส์ซึ่งเป็นดีเอสพีแบบทศนิยมตายตัวใช้รีจิสเตอร์ขนาด 16 บิตและเก็บเลขทศนิยมในลักษณะตำแหน่งจุดทศนิยมตายตัว ในขณะที่ ADSP21060/62 ซึ่งเป็นดีเอสพีในกลุ่มทศนิยมลอยตัวใช้ 32 บิตในการเก็บเลขทศนิยมตามมาตรฐาน IEEE การเขียนโปรแกรมสำหรับดีเอสพีแบบทศนิยมตายตัวจะต้อง

ระมัดระวังไม่ให้ตัวเลขมีค่าใหญ่กว่าที่จะเก็บได้ในรีจิสเตอร์ ส่วนดีเอสพีแบบทศนิยมลอยตัวนั้นสามารถแสดงตัวเลขในแบบเอ็กซ์โพเนนต์ (exponent) กับแมนติสซา (mantissa) หน่วยประมวลผลเชิงคณิตศาสตร์ของมันสามารถคำนวณในแบบทศนิยมลอยตัวได้โดยตรง มันจึงสามารถคำนวณได้อย่างรวดเร็วและไม่ต้องพะวงถึงเรื่องข้อมูลล้นรีจิสเตอร์ด้วยเพราะสามารถเก็บจำนวนค่าสูง ๆ ได้เป็นอย่างดี แต่อย่างไรก็ตามดีเอสพีชนิดทศนิยมลอยตัวโดยทั่วไปมีราคาแพงกว่าดีเอสพีแบบทศนิยมตายตัว ในที่นี้จะกล่าวถึงดีเอสพีที่ใช้ในการวิจัยครั้งนี้นั่นคือ ADSP2101 ซึ่งเป็นดีเอสพีแบบทศนิยมตายตัวของบริษัทอนาล็อกดีไวส์เท่านั้น

ADSP2101 และดีเอสพีตัวอื่น ๆ ในตระกูลนี้เป็นดีเอสพีที่ใช้หลักการทำงานแบบขนาน ทำให้มันสามารถทำการประมวลผลหลาย ๆ อย่างได้ภายในไซเคิลเดียว นั่นคือภายในหนึ่งไซเคิลมันสามารถ

1. สร้างสัญญาณแอดเดรส (address) สำหรับการเฟ็ทช์ (fetch) คำสั่งครั้งถัดไป
2. เฟ็ทช์คำสั่ง
3. เคลื่อนย้ายข้อมูลหนึ่งหรือสองข้อมูล
4. ปรับปรุงข้อมูลในตัวชี้ตำแหน่งหน่วยความจำได้หนึ่งหรือสองตัว
5. ทำการประมวลผลทางเลขหรือตรรกะ

<u>โครงสร้างภายในของ ADSP2101</u>

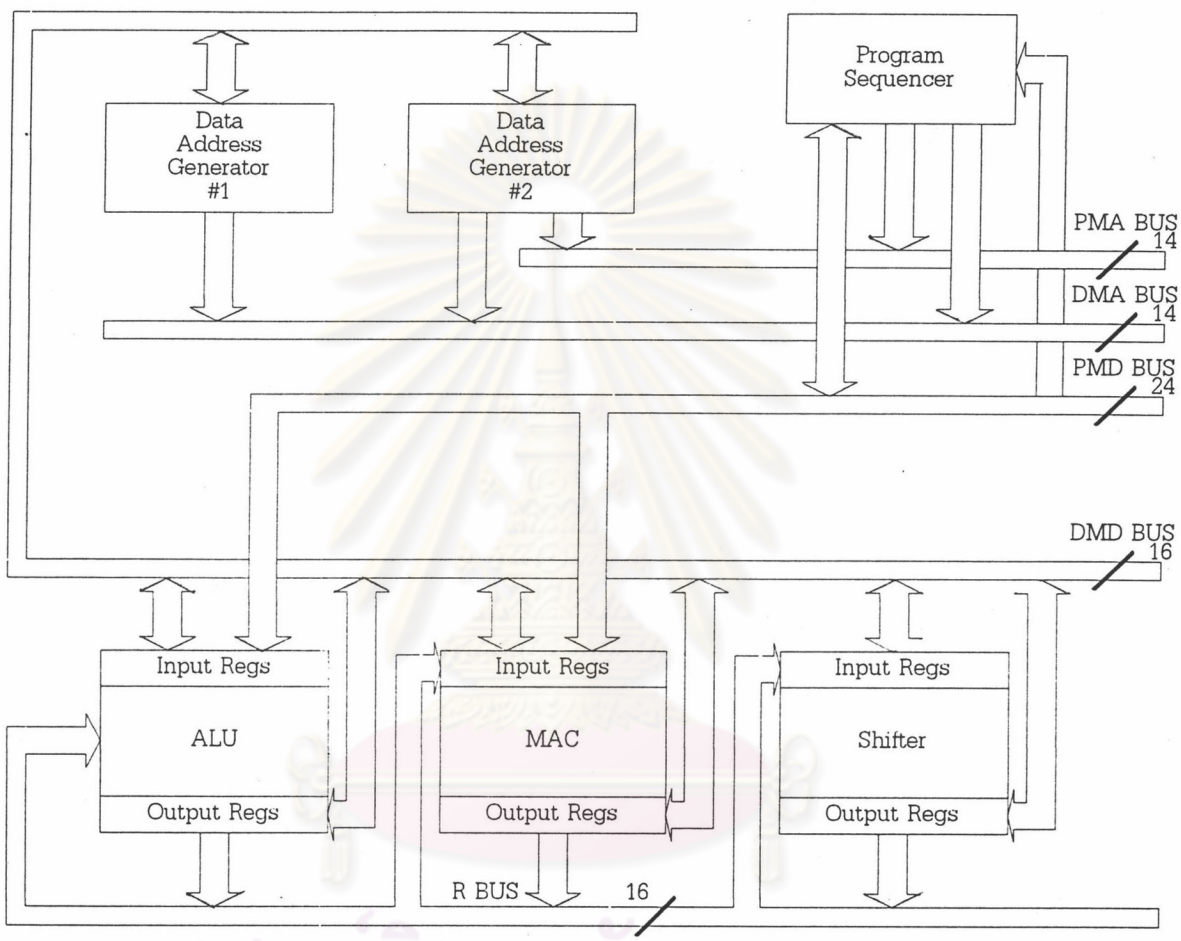
ADSP2101 เป็นดีเอสพีในตระกูล ADSP2100 ดีเอสพีในตระกูลนี้มีโครงสร้างพื้นฐานภายในที่เหมือน ๆ กัน จะต่างกันที่ขนาดของหน่วยความจำภายในซึ่งแต่ละตัวในตระกูลอาจมีขนาดไม่เท่ากัน และในเรื่องของอุปกรณ์รอบข้างภายใน (built-in peripheral) สำหรับ ADSP2101 มีขนาดหน่วยความจำโปรแกรมภายใน 2 กิโลเวิร์ด และมีหน่วยความจำข้อมูล 1 กิโลเวิร์ด มีอุปกรณ์รอบข้างได้แก่ ช่องสื่อสารอนุกรมจำนวน 2 ช่อง มีไทเมอร์ (timer) จำนวนหนึ่งตัว รายละเอียดจะได้กล่าวถึงในลำดับถัดไป

1. <u>โครงสร้างพื้นฐาน</u>

ในรูปที่ 4.1 แสดงโครงสร้างพื้นฐานของ ADSP2101 รายละเอียดของแต่ละส่วนมีดังนี้

1.1 <u>หน่วยคำนวณ</u> หน่วยคำนวณประกอบไปด้วยหน่วยประมวลผลย่อยจำนวน 3 หน่วยด้วยกันนั่นคือ หน่วยเลขคณิต/ตรรก (arithmetic/logic unit - ALU) หน่วย

การคูณ/ตัวสะสม (multiplier/accumulator - MAC) และหน่วยเลื่อนข้อมูลบาเรล (barrel shifter) หน่วยคำนวณสามารถจัดการกับข้อมูลขนาด 16 bit และมีฮาร์ดแวร์ที่จะสนับสนุนการคำนวณตัวเลขที่มีความละเอียด (ตำแหน่งของจุดทศนิยมของจำนวนทศนิยมตายตัว) ต่าง ๆ กันได้



รูปที่ 4.1 แสดงโครงสร้างพื้นฐานของดีเอสพี ADSP2101

ในส่วนของ ALU มีความสามารถทำงานในเชิงตรรกะ การบวกลบคูณ รวมไปถึงมีฮาร์ดแวร์พื้นฐานที่ใช้ในการหารด้วย ในส่วนของ MAC มีความสามารถในการคูณซึ่งสามารถทำเสร็จภายในไซเคิล (cycle) เดียว นอกจากนี้ยังสามารถทำการคูณแล้วต่อด้วยการบวก คูณแล้วต่อด้วยการลบได้ในไซเคิลเดียวเช่นกัน ความสามารถนี้มีประโยชน์มากในงานด้านประมวลผลสัญญาณดิจิตอล ตัวอย่างที่เห็นได้ชัดอันหนึ่งก็คือการคำนวณหาออโตโครีเลชันของสัญญาณดังที่ได้กล่าวมาแล้วในบทก่อน สำหรับตัวเลื่อนข้อมูลมีความสามารถในการเลื่อนบิตของข้อมูลซึ่งทำได้ทั้งแบบตรรกะและแบบเลขคณิต นอกจากนี้ก็ยังมีความสามารถในการนอร์มอลไลซ์ (normalize) และดีนอร์มอลไลซ์ (de-normalize) ข้อมูล ความสามารถนี้มีประโยชน์อย่างมากในการคำนวณแบบทศนิยมตายตัวเพราะทำให้เราสามารถปรับข้อมูลให้มีขนาดอยู่ในระดับที่เหมาะสมต่อการคำนวณและทำให้มั่นใจได้ว่าจะไม่เกิดการล้นของข้อมูลเมื่อมีการคำนวณและ

สะสมผลการคำนวณไว้ในรีจิสเตอร์ หน่วยคำนวณถูกจัดวางในลักษณะที่ผลลัพธ์จากหน่วยประมวลผลหนึ่งสามารถนำเข้าเป็นข้อมูลเข้าของอีกหน่วยประมวลผลหนึ่งได้ภายในไซเคิลการทำงานรอบถัดไปโดยผ่านทางบัสผลลัพธ์ (result bus - R bus)

ภายในตัวประมวลผลแต่ละตัวมีรีจิสเตอร์ขาเข้าซึ่งก็คือ Input Regs ในรูปรีจิสเตอร์ขาเข้าเชื่อมโยงกับบัสข้อมูลภายใน (data memory data bus - DMD bus) ข้อมูลจากรีจิสเตอร์ภายในตัวดีเอสพีจะเข้าสู่หน่วยประมวลผลทั้งสามผ่านทางบัสดังกล่าว เมื่อการประมวลผลเสร็จสิ้นลงผลลัพธ์ที่ได้ก็จะถูกส่งออกไปที่รีจิสเตอร์ขาออกซึ่งก็คือ Output Regs ที่แสดงในรูปก่อนที่ข้อมูลจะถูกส่งกลับไปยังรีจิสเตอร์ภายในตัวดีเอสพีหรือส่งไปเป็นข้อมูลขาเข้าของหน่วยประมวลผลอื่น รีจิสเตอร์ขาเข้าและขาออกทำหน้าที่เป็นเสมือนจุดหยุดพักของข้อมูลก่อนที่จะถูกส่งไปประมวลผลหรือส่งกลับไปยังบัส มีผลเหมือนเป็นไปป์ไลน์หนึ่งระดับ ส่วนบัสผลลัพธ์ นั้นก็ช่วยแก้ปัญหาจากการหน่วงข้อมูลอันเกิดจากตัวไปป์ไลน์เองในกรณีที่ผลลัพธ์จะต้องส่งกลับไปยังหน่วยประมวลต่อทันทีโดยไม่ต้องผ่านรีจิสเตอร์ภายในตัวดีเอสพี

1.2 <u>หน่วยสร้างสัญญาณแอดเดรสและหน่วยลำดับการทำงานของโปรแกรม</u> (address generators and program sequencer) หน่วยสร้างสัญญาณแอดเดรส (address generators - DAG) มีจำนวนสองหน่วยด้วยกันเพื่อให้การใช้งานหน่วยประมวลผลเต็มประสิทธิภาพ DAG จะกำเนิดสัญญาณแอดเดรสเมื่อมีการเอ็กซีคิวต์คำสั่งที่อ้างอิงถึงข้อมูลภายหน่วยความจำ ในขณะทำการเคลื่อนย้ายข้อมูลระหว่างหน่วยความจำกับรีจิสเตอร์ ในแต่ละ DAG มีตัวชี้แอดเดรสจำนวน 4 ตัวด้วยกัน เมื่อมีการใช้คำสั่งอ้างอิงหน่วยความจำแบบอ้อม (indirect addressing mode) ซึ่งใช้ตัวชี้แอดเดรสในการระบุตำแหน่งในหน่วยความจำ เมื่อได้ข้อมูลแล้วตัวชี้แอดเดรสจะถูกเพิ่มหรือลดค่าลง ด้วยค่าในรีจิสเตอร์ที่กำหนดไว้ การปรับค่าของตัวชี้แอดเดรสจะอยู่ในไซเคิลการทำงานเดียวกันกับการเคลื่อนย้ายข้อมูลทำให้การทำงานเป็นไปอย่างรวดเร็ว เหมาะสำหรับการจัดการเกี่ยวกับแอเรย์ของข้อมูล

ตัวชี้แอดเดรสแต่ละตัวจะมีรีจิสเตอร์ L อยู่เป็นคู่กับมัน เราสามารถทำโมดูโล (modulo) แอดเดรสได้โดยอัตโนมัติในขณะเคลื่อนย้ายข้อมูลจากหน่วยความจำในแต่ละครั้ง รีจิสเตอร์ L เป็นตัวกำหนดตัวเลขที่จะนำมาโมดูโล คุณสมบัติข้อนี้เรานำมาใช้ทำแถวคอยแบบวงกลม (circular queue) ซึ่งเป็นฮาร์ดแวร์ สำหรับงานวิจัยนี้ก็ใช้แถวคอยแบบวงกลมโดยอาศัยวิธีดังกล่าวเป็นที่พักตัวอย่างข้อมูลเสียงจากโคเด็ค (codec)

DAG1 สามารถกำเนิดแอดเดรสที่ใช้อ้างเฉพาะหน่วยความจำข้อมูลได้เพียงอย่างเดียว ส่วน DAG2 สามารถสร้างแอดเดรสได้ทั้งหน่วยความจำข้อมูลและหน่วยความจำโปรแกรม

หน่วยลำดับการทำงานของโปรแกรมมีหน้าที่สร้างสัญญาณแอดเดรสสำหรับใช้ในการเฟ็ทช์คำสั่งจากหน่วยความจำโปรแกรม คำสั่งจะถูกเคลื่อนย้ายจากหน่วยความจำมาเก็บไว้ในรีจิสเตอร์คำสั่ง (instruction register) ซึ่งรีจิสเตอร์ตัวนี้ทำหน้าที่เป็นไปป์ไลน์หนึ่งระดับสำหรับการเฟ็ทช์คำสั่ง คำสั่งจะถูกเคลื่อนย้ายมาที่รีจิสเตอร์ที่ไซเคิลหนึ่ง และจะถูกเอ็กซีคิวต์ (execute) ในไซเคิลถัดไปซึ่งก็เป็นเวลาเดียวกันที่คำสั่งถัดไปถูกเคลื่อนย้ายเข้ามาที่รีจิสเตอร์คำสั่งด้วย หน่วยลำดับการทำงานสามารถเอ็กซีคิวต์คำสั่งกระโดดแบบมีเงื่อนไข การเรียกโปรแกรมย่อย และการกลับจากโปรแกรมย่อยได้ภายในไซเคิลเดียว ส่วนการทำคำสั่งลูป (loop) ดีเอสพีก็สามารถทำได้โดยไม่ต้องใช้เวลาเพิ่มเติมในการตรวจสอบว่าทำครบรอบจำนวนครั้งที่ต้องการหรือยัง ทั้งนี้โดยอาศัยรีจิสเตอร์นับลูป (loop count register) และ สแต็กลูป (loop stack)

1.3 บัส (bus) ส่วนประกอบภายในเชื่อมโยงเข้าหากันด้วยบัสภายในจำนวนห้าบัสด้วยกัน บัสแอดเดรสสำหรับหน่วยความจำโปรแกรม (PMA) และบัสแอดเดรสสำหรับหน่วยความจำข้อมูล (DMA) ใช้สำหรับเป็นสัญญาณแอดเดรสในการอ่านเขียนหน่วยความจำโปรแกรมและหน่วยความจำข้อมูลตามลำดับ ส่วนบัสข้อมูลสำหรับหน่วยความจำโปรแกรม (PMD) และบัสข้อมูลสำหรับหน่วยความจำข้อมูล (DMD) ใช้เป็นเส้นทางผ่านของข้อมูลหรือคำสั่งจากหน่วยความจำข้อมูลและหน่วยความจำโปรแกรม บัสทั้งสองคู่ยังถูกมัลติเพล็กซ์ภายในคู่ของมันและออกไปนอกตัวชิพ (chip) เป็นบัสแอดเดรสและบัสข้อมูล การแยกแยะว่าการอ่านเขียนหน่วยความจำทำกับหน่วยความจำชนิดใดก็อาศัยขาสัญญาณ BMS DMS และ PMS เป็นตัวบอก BMS ใช้เป็นสัญญาณในการเลือกหน่วยความจำบูท (boot memory) ใช้เมื่อตอนเริ่มเปิดเครื่อง สัญญาณ PMS ใช้เป็นสัญญาณในการเลือกหน่วยความจำโปรแกรม ส่วน DMS ใช้เป็นสัญญาณในการเลือกหน่วยความจำข้อมูล บัสที่เหลืออีกบัสหนึ่งก็คือ บัสผลลัพธ์ (R bus) ซึ่งใช้เป็นบัสในการเคลื่อนย้ายข้อมูลระหว่างหน่วยคำนวณด้วยกัน

บัส PMA เป็นบัสที่มีความกว้าง 14 บิต ทำให้สามารถอ้างหน่วยความจำโปรแกรมได้ 16 กิโลเวิร์ด ซึ่งข้อมูลที่เก็บไว้ในหน่วยความจำโปรแกรมอาจเป็นโปรแกรมหรือข้อมูลใด ๆ ก็ได้ ส่วนบัส PMD มีความกว้าง 24 บิตซึ่งเป็นขนาดเท่ากับจำนวนบิตของคำสั่ง (instruction)

สำหรับบัส DMA ก็มีขนาด 14 บิตเช่นกันทำให้สามารถอ้างหน่วยความจำข้อมูลได้ 16 กิโสเวิร์ดเช่นเดียวกันแต่ข้อมูลในหน่วยความจำข้อมูลซึ่งใช้บัส DMD ในการเคลื่อนย้ายมีความกว้าง 16 บิตเท่านั้น บัส DMD เป็นบัสที่ใช้ในการเคลื่อนย้ายข้อมูลระหว่างรีจิสเตอร์กับหน่วยความจำและระหว่างรีจิสเตอร์กับรีจิสเตอร์ด้วยกัน การเคลื่อนย้ายสามารถทำเสร็จในหนึ่งไซเคิลเท่านั้น

ในส่วนต่อไปจะกล่าวถึงอุปกรณ์รอบข้างภายในตัว ADSP2101 ซึ่งประกอบด้วยพอร์ตสื่อสารอนุกรมจำนวนสองพอร์ต และไทเมอร์ จำนวนหนึ่งตัว

2.1 พอร์ตสื่อสารอนุกรม (serial port - SPORT) เป็นพอร์ตสื่อสารแบบสมวาร (synchronous) สามารถส่งและรับข้อมูลได้ในเวลาเดียวกัน การควบคุมการรับส่งข้อมูลต้องใช้สัญญาณเฟรมมิง (framing signal) เป็นตัวบอกจุดเริ่มต้นในการรับส่งข้อมูล แต่ละพอร์ตสามารถสร้างสัญญาณนาฬิกาได้ด้วยตัวมันเองหรือจะโปรแกรมให้ใช้สัญญาณนาฬิกาจากภายนอกก็ได้ ในทำนองเดียวกันสัญญาณเฟรมมิงก็อาจกำเนิดจากภายในชิพหรือว่าใช้สัญญาณจากภายนอกก็ได้ ความกว้างของข้อมูลที่รับส่งในหนึ่งเวิร์ดก็กำหนดได้เช่นกันว่าต้องการจำนวนกี่บิตโดยโปรแกรมได้ตั้งแต่ 3 บิตจนถึง 16 บิต ในการวิจัยครั้งนี้ได้เชื่อมต่อพอร์ตสื่อสารอนุกรมพอร์ตหนึ่งเข้ากับโคเด็ค และโปรแกรมจำนวนบิตของข้อมูลเป็น 8 บิต สำหรับสัญญาณนาฬิกาได้จากวงจรภายนอก ส่วนสัญญาณเฟรมมิงสร้างจากวงจรภายภายในชิพ

2.2 ไทเมอร์ เป็นตัวสร้างสัญญาณขัดจังหวะ (interrupt) เพื่อขัดจังหวะการทำงานของดีเอสพีเป็นระยะ ๆ ขึ้นกับการโปรแกรม คาบเวลาในการการขัดจังหวะสามารถโปรแกรมได้โดยการเขียนค่าที่เหมาะสมลงในรีจิสเตอร์ตัวนับ (counter register) ขนาด 16 บิต และรีจิสเตอร์พรีสเกลเลอร์ (pre-scaler) ขนาด 8 บิต รีจิสเตอร์พรีสเกลเลอร์จะทำให้รีจิสเตอร์ตัวนับมีค่าลดลงหนึ่งทุก ๆ ช่วงเวลาหนึ่ง ซึ่งช่วงเวลาดังกล่าวเราสามารถเลือกได้ตั้งแต่ทุกไซเคิลไปจนถึงทุก 256 ไซเคิล เมื่อรีจิสเตอร์ตัวนับได้นับลงจนถึงศูนย์แล้วก็จะเกิดการการขัดจังหวะขึ้น และรีจิสเตอร์ตัวนับก็จะถูกกำหนดให้มีค่าเดิมเหมือนเริ่มโปรแกรมอีกครั้งหนึ่งโดยอัตโนมัติ

คณิตศาสตร์ทศนิยมตายตัวสำหรับ ADSP2101

เลขจำนวนจริงที่เป็นทศนิยมแบ่งได้เป็นสามส่วนด้วยกัน คือ ตัวเลขหน้าจุดทศนิยม จุดทศนิยม และตัวเลขหลังจุดทศนิยม โดยที่ตัวเลขหน้าจุดทศนิยมเป็นเลขจำนวนเต็ม และตัวเลขหลังจุดทศนิยมเป็นเศษส่วน การบรรจุตัวเลขทศนิยมในรีจิสเตอร์หรือหน่วยความจำของคอมพิวเตอร์ทำได้สองรูปแบบคือรูปแบบทศนิยมตายตัวและรูปแบบทศนิยมลอยตัว

– รูปแบบทศนิยมตายตัว ในรูปแบบนี้จุดทศนิยมไม่จำเป็นต้องเก็บลงในรีจิสเตอร์หรือหน่วยความจำจริง ๆ แต่มันจะถูกกำหนดเอาไว้ก่อนว่าอยู่ที่ตำแหน่งระหว่างบิตใดกับบิตใด ซอฟท์แวร์ที่ประมวลผลทางคณิตศาสตร์จะคำนวณโดยถือว่ามีจุดทศนิยมอยู่ที่ตำแหน่งนั้น

การคำนวณบางประเภทอย่างเช่นการคูณจะทำให้จุดทศนิยมนั้นเปลี่ยนตำแหน่งไป เมื่อคำนวณเสร็จแล้วอาจจำเป็นต้องเลื่อนบิตข้อมูลถ้ายังต้องการจุดทศนิยมในตำแหน่งเดิม

— รูปแบบทศนิยมลอยตัว ในรูปแบบนี้การเก็บเลขทศนิยมจะแยกออกเป็นสองจำนวนคือเอ็กซ์โปเนนต์และแมนติสซา เอ็กซ์โปเนนต์เป็นตัวกำหนดตำแหน่งของจุดทศนิยม ส่วนแมนติสซาเป็นจำนวนเต็มที่ถูกนอร์มอลไลซ์แล้ว

เนื่องจากในการวิจัย ใช้ตัวเลขในรูปแบบทศนิยมตายตัวเท่านั้นไม่ได้ใช้รูปแบบทศนิยมลอยตัว ต่อไปจะกล่าวถึงรูปแบบทศนิยมตายตัวและการคำนวณในแบบทศนิยมตายตัวเท่านั้น

รูปแบบของจำนวนทศนิยมตายตัวของ ADSP2101 เป็นแบบทูคอมพลีเนนต์ (two complement) เขียนได้ในรูป I.Q โดยที่ I เป็นจำนวนเต็มที่แสดงจำนวนบิตที่อยู่หน้าจุดทศนิยมโดยนับรวมจำนวนบิตเครื่องหมายด้วย ส่วน Q เป็นจำนวนเต็มที่แสดงจำนวนบิตที่อยู่หลังจุดทศนิยม ตัวอย่างเช่น 1.15 แทนตัวเลขที่มีบิตเครื่องหมายหนึ่งบิตส่วนบิตที่เหลือแทนขนาดของตัวเลขนั้น ตัวอย่างตัวเลขในรูปแบบนี้ได้แก่ 0.101110111111111 รูปแบบในอีกตัวอย่างหนึ่งได้แก่ 16.0 ซึ่งเป็นรูปแบบของจำนวนเต็มไม่มีทศนิยม สำหรับรูปแบบ 1.15 นั้นเป็นรูปแบบที่เหมาะกับการทำงานของ ADSP2101 มากที่สุดเนื่องจากเป็นรูปแบบที่ ADSP2101 ทำงานได้อย่างมีประสิทธิภาพสูงสุด

1. <u>การบวก</u>

ADSP2101 สามารถบวกเลขจำนวน 16 บิตได้โดยตรงและยังคงตำแหน่งของจุดทศนิยมของผลลัพธ์ไว้ที่เดิมด้วย ไวยากรณ์คำสั่งในภาษาแอสเซมบลีของการบวกเป็นดังนี้

```
[IF cond]   AR หรือ AF  = xop+yop;
            หรือ        = xop+C;
            หรือ        = xop+yop+C;
```

สำหรับ IF cond เป็นเงื่อนไขสำหรับทดสอบว่าจะทำการบวกหรือไม่ ซึ่งจะไม่กล่าวถึงรายละเอียดในที่นี้ การทำงานของคำสั่งบวกในกรณีที่เงื่อนไขเป็นจริงจะทำการบวกค่าใน xop เข้ากับค่าใน yop หรือ แครีบิต (C) หรือทั้ง yop และแครี่บิต ผลจากการบวกจะถูกเก็บไว้ในรีจิสเตอร์ AR หรือ AF

xop สามารถเป็นรีจิสเตอร์ต่าง ๆ ได้เฉพาะต่อไปนี้
AX0, AX1, AR, MR2, MR1, MR0, SR1 และ SR0

yop สามารถเป็นรีจิสเตอร์ต่าง ๆ ได้เฉพาะต่อไปนี้
AY0, AY1 และ AF

ตัวอย่างของคำสั่งบวกได้แก่

AR = AX0 + AY0;

นอกจากนี้ ADSP2101 ยังสามารถบวกเลขที่มีจำนวนบิตมากกว่า 16 บิตได้โดยอาศัยบิตทด (carry bit) เป็นตัวช่วย รายละเอียดมือธิบายไว้ในเอกสารอ้างอิง Digital Signal Processing Application Using the ADSP2100 Family, 1994

2. <u>การลบ</u>

ADSP2101 สามารถบวกลบเลขจำนวน 16 บิตได้โดยตรงและยังคงตำแหน่งของจุดทศนิยมของผลลัพธ์ไว้ที่เดิมด้วย ไวยากรณ์คำสั่งในภาษาแอสเซมบลีของการลบเป็นดังนี้

[IF cond] AR หรือ AF = xop-yop;
หรือ = xop+C-1;
หรือ = xopl-yop+C-1;

หรืออาจเขียนในอีกรูปแบบหนึ่งก็ได้นั่นคือ

[IF cond] AR หรือ AF = yop-xop;

หรือ = yop-xop+C-1;

หรือ = -xop+C-1

สำหรับ IF cond เป็นเงื่อนไขสำหรับทดสอบว่าจะทำการลบหรือไม่ ซึ่งจะไม่กล่าวถึงรายละเอียดในที่นี้ การทำงานของคำสั่งลบในกรณีที่เงื่อนไขเป็นจริงจะทำการลบตามรูปแบบไวยากรณ์ที่แสดงไว้ ผลจากการลบจะถูกเก็บไว้ในรีจิสเตอร์ AR หรือ AF

xop สามารถเป็นรีจิสเตอร์ต่าง ๆ ได้เฉพาะต่อไปนี้
AX0, AX1, AR, MR2, MR1, MR0, SR1 และ SR0
yop สามารถเป็นรีจิสเตอร์ต่าง ๆ ได้เฉพาะต่อไปนี้
AY0, AY1 และ AF

ตัวอย่างของคำสั่งลบได้แก่

AR = AX0 - AY0;

นอกจากนี้ ADSP2101 ยังสามารถลบเลขที่มีจำนวนบิตมากกว่า 16 บิตได้โดยอาศัยบิตทดเป็นตัวช่วย รายละเอียดมีอธิบายไว้ในเอกสารอ้างอิง Digital Signal Processing Application Using the ADSP2100 Family, 1994

3. <u>การคูณ</u>

รูปแบบคำสั่งสำหรับการคูณเป็นดังนี้

[IF cond] MR หรือ MF = xop * yop (SS);
หรือ = xop * yop (SU);
หรือ = xop * yop (US);
หรือ = xop * yop (UU);
หรือ = xop * yop (RND);

สำหรับ IF cond เป็นเงื่อนไขว่าจะทำการคูณหรือไม่ รายละเอียดจะไม่กล่าวถึงในที่นี้ การทำงานของการคูณเป็นไปตามรูปแบบที่แสดงไว้ ผลคูณที่ได้จะถูกเก็บไว้ในรีจิสเตอร์ MR หรือ MF รูปแบบของโอเปอแรนด์ทั้งสองถูกระบุไว้ต่อท้ายโอเปอแรนด์ โดยที่ S หมายถึงโอเปอร์แรนด์เป็นจำนวนแบบมีเครื่องหมาย U หมายถึงโอเปอร์แรนด์แบบไม่มีเครื่องหมาย ตัวอักษรตัวแรกเป็นรูปแบบของ xop ส่วนตัวหลังหมายถึงรูปแบบของ yop สำหรับ RND เมื่อทำการคูณเสร็จแล้วผลของการคูณจะถูกปัดเพื่อให้ลงในบิตข้อมูลขนาด 24 บิตได้ หรือในกรณีที่ไม่ได้เกิดการล้นของข้อมูลก็จะทำการปัดเศษข้อมูลในบิต 31-16 ให้เป็นข้อมูลขนาด 16 บิต และเก็บลงในรีจิสเตอร์ปลายทางที่ระบุไว้ในคำสั่ง

xop สามารถเป็นรีจิสเตอร์ต่าง ๆ ได้เฉพาะดังต่อไปนี้
MX0, MX1, MR2, MR1, MR0 AR, SR1 และ SR0
yop สามารถเป็นรีจิสเตอร์ต่าง ๆ ได้เฉพาะดังต่อไปนี้
MY0, MY1 และ MF

ตัวอย่างของคำสั่งคูณได้แก่

MR = MX0 * MY0 (UU);

4. <u>การหาร</u>

รูปแบบของคำสั่งหารเป็นดังนี้

DIVS yop,xop;

DIVQ xop;

รีจิสเตอร์ที่เป็นได้สำหรับ xop ได้แก่ AX0, AX1, AR, MR2, MR1, MR0, SR1 และ SR0

ส่วนรีจิสเตอร์ที่เป็นได้สำหรับ yop ได้แก่ AY1 และ AF

คำสั่งหารนี้จะทำการหาร yop ด้วย xop นั่นคือ yop/xop คำสั่งหารพื้นฐานมีด้วยกันสองคำสั่งคือ DIVS และ DIVQ การหารที่มีตัวตั้งขนาด 32 บิตด้วยตัวหารขนาด 16 บิต และได้ผลหารขนาด 16 บิต ทำเสร็จในเวลา 16 ไซเคิล สำหรับการหารที่ต้องการความละเอียดมากกว่านี้ก็สามารถทำได้เช่นกัน

การหารสามารถทำได้ทั้งการหารจำนวนแบบมีบิตเครื่องหมายและการหารจำนวนที่ไม่มีบิตเครื่องหมาย แต่ตัวตั้งและตัวหารจำเป็นต้องเป็นจำนวนที่มีบิตเครื่องหมายหรือไม่มีบิตเครื่องหมายเหมือน ๆ กัน การตั้งต้นการหารเริ่มจากกำหนดค่า 16 บิตบนของตัวตั้งไว้ในรีจิสเตอร์ yop นั่นคือ AY1 หรือ AF ส่วนตัวหารถูกกำหนดไว้ในรีจิสเตอร์ xop หลังจากนั้นการหารจะถูกกระทำโดยการเอ็กซีคิวต์คำสั่งหาร DIVS และ DIVQ การเอ็กซีคิวต์คำสั่ง DIVQ ซ้ำจะเป็นการกระตุ้นให้เกิดการหารตามขั้นตอนวิธีการหารแบบ บวกลบแบบมีเงื่อนไขไม่คืนค่า (non-restoring conditional add-subtract division algorithm) ผลของการหารในที่สุดจะถูกเก็บไว้ในรีจิสเตอร์ AY0

สำหรับการหารจำนวนแบบคิดเครื่องหมาย ในขั้นแรกจะต้องเอ็กซีคิวต์คำสั่ง DIVS หนึ่งครั้ง ซึ่งจะเป็นการคำนวณเครื่องหมายของผลหาร หลังจากนั้นจะต้องเอ็กซีคิวต์คำสั่ง DIVQ เป็นจำนวนครั้งเท่ากับจำนวนบิตที่ยังเหลืออยู่ในผลหาร ยกตัวอย่างเช่นสำหรับการหารเลข 16 บิตแบบมีเครื่องหมาย เราจะต้องเอ็กซีคิวต์คำสั่ง DIVS หนึ่งครั้งหลังจากนั้นก็จะต้องเอ็กซีคิวต์คำสั่ง DIVQ อีก 15 ครั้ง

สำหรับการหารตัวเลขแบบไม่คิดเครื่องหมายในขั้นแรกจะต้องกำหนดค่า 16 บิตบนของตัวตั้งไว้ใน AF และจะต้องกำหนดบิต AQ ในรีจิสเตอร์ ASTAT (arithmetic status register) ให้เป็นศูนย์ ซึ่งการทำเช่นนี้จะเป็นการระบุว่าเครื่องหมายของผลหารเป็นบวก หลังจากนั้นเราก็จะต้องเอ็กซีคิวต์คำสั่ง DIVQ เป็นจำนวนครั้งเท่ากับจำนวนบิตของผลลัพธ์ ตัวอย่างเช่นสำหรับการหารตัวเลข 16 บิต เราก็จะต้องเอ็กซีคิวต์คำสั่ง DIVQ 16 ครั้ง

บิตของผลหารที่ได้จากการเอ็กซีคิวต์คำสั่ง DIVS และ DIVQ ในแต่ละครั้งก็คือ AQ ซึ่งจะถูกเขียนลงในรีจิสเตอร์ ASTAT ในช่วงท้ายของแต่ละไซเคิล เศษที่เหลือจากการหารด้วยขั้นตอนวิธีนี้ยังเป็นจำนวนที่ไม่ถูกต้องและจำเป็นต้องมีการแก้ไขก่อนนำไปใช้ สำหรับราย

ละเอียดในการแก้ไขมีรายละเอียดอยู่ในหนังสือคู่มือผู้ใช้ของ ADSP2101 (ADSP2101 Family User Manual, 1994)

การควบคุมลำดับคำสั่ง

ลำดับคำสั่งของโปรแกรมสามารถควบคุมได้ด้วยคำสั่งต่อไปนี้

- DO UNTIL ใช้สำหรับการทำงานในลักษณะลูป
- JUMP ใช้สำหรับให้โปรแกรมกระโดดไปทำงานยังที่อื่น
- CALL ใช้สำหรับเรียกโปรแกรมย่อย
- RTS ใช้สำหรับการกลับจากโปรแกรมย่อย
- RTI ใช้สำหรับกลับจากโปรแกรมย่อยที่ให้บริการการขัดจังหวะ (interrupt)
- IF สำหรับให้โปรแกรมกระโดดโดยมีเงื่อนไข
- IDLE หยุดการทำงานของโปรแกรมจนกว่าจะเกิดการขัดจังหวะ