สถาปัตยกรรม

RISC

แต่เดิมแนวความคิดที่จะทำให้คอมพิวเตอร์ทำงานได้เร็วขึ้น จะใช้วิธีการเพิ่มขีดความสามารถของคำสั่ง คำสั่งหนึ่งจึงทำงานเพิ่มขึ้นและซับซ้อนขึ้น เช่นคำสั่งค้นหาข้อมูลในสตริงเป็นต้น ด้วยวิธีนี้ทำให้สถาปัตยกรรมเฉพาะของตัวซีพียูต้องสนับสนุนคำสั่งใหม่ ๆ เพิ่มขึ้น ประะจวบกับไซเกิลการทำงานของแต่ละคำสั่งใช้จำนวนไซเกิลไม่เท่ากัน บางคำสิ่ง ทำงานเสร็จภายในไซเกิลเดียว บางคำสั่งใช้หลายไซเกิล ความคิดนี้จึงกลายมาเป็นคอมพิวเตอร์ในกลุ่ม CISC หรือ Complex Instruction Set Compter และความคิดนี้ได้พัฒนาต่อเนื่องมาเป็นลำดับจนถึงปัจจุบัน ซีพียูหลายตัว เช่น 80386,80486 ก็ใช้แนวความคิดนี้ อีกแนวความคิดหนึ่ง ได้เริ่มมาตั้งแต่ปี พ.ศ. 2518 โดยกลุ่มนักวิจัยแห่งมหาวิทยาลัยแคริฟอร์เนีย ได้พัฒนาซีพียูที่ชื่อว่า RISC แนวความคิดนี้คือ ให้ซีพียูทำงานที่ไซเกิลแน่นอน โดยลดจำนวนคำสั่งลง ให้เหลือคำสั่ง พื้นฐานมากที่สุด แล้วใช้หลักการไปป์ไลน์(pipeline) คือนำคำสั่งมาเรียงการทำงานแบบขนานเหลื่อมกันหรือเข้าทำงาน แต่ละสถานี ตามลำดับเรียงกันไป ทุกสถานีงานจะมีงานทำตลอดเวลา การลดจำนวนคำสั่งลงนี้เอง จึงกลายมาเป็นซีพียู ประเภท RISC หรือ Reduce Instructionset Computer คอมพิวเตอร์แบบ RISC จึงทำงานได้เร็วและ เป็นกลไก ที่สามารถเพิ่มขีดความสามารถโดยรวมได้หลาย บริษัทจึงหันมาพัฒนาคอมพิวเตอร์ RISCจนในที่สุดมีซับซีพียู ที่กำหนด ขึ้นมา และบริษัทซันไมโครซิสเต็ม นำมาใช้เป็นซีพียูหลักของตนคือSPARC SPARCเป็นซีพียูที่มีการกำหนด โครงสร้างสถาปัตยกรรมและคำสั่งไว้ชัดเจน โดยมีบริษัทหลายบริษัทนำไปพัฒนาโครงสร้างชิป และผลิตชิปไมโครโปเซสเซอร์ SPARC ได้รับการพัฒนาขีดความสามารถและประยุกต์เข้ากับเครื่องเวิร์กสเตชันและไฟล์เซอร์เวอร์อย่างกว้างขวาง

จุดเริ่มต้นและความเป็นมา

ใน ค.ศ. 1948 ครั้งที่เริ่มมีการพัฒนาดิจิตอลคอมพิวเตอร์ การพัฒนาในครั้งนั้นเริ่มจากเครื่อง MARK1 ซึ่งมีคำสั่ง เพียง 7 คำสั่งแต่ละคำสั่งมีลักษณะการทำงานที่ซับซ้อน เช่น คำสั่งบวก คำสั่งกระโดด คำสั่งในยุคนั้น มีเพียง เท่าที่จำเป็น แต่

เมื่อมีการพัฒนาต่อมาเรื่อย ๆ การสร้างคำสั่งก็เพิ่มมากขึ้น เช่น VAX มีคำสั่งถึงกว่า 300 คำสั่ง

นอกจากนี้คำสั่งก็จะดูซับซ้อนมากขึ้น หากพิจารณาไมโครโปเซสเซอร์ 8080 มีเพียง 78 คำสั่งแต่เมื่อ Z80 ได้รับการพัฒนาขึ้น ชุดคำสั่งก็เพิ่มเป็นซุเปอร์เซต คือมีถึงกว่า 150 คำสั่ง แล้วพัฒนาการต่อมาเป็น 8088, 8086 ก็มีชุดคำสั่งเพิ่มขึ้นอีก การออกแบบแต่ละครั้ง ดูจะ พยายามสร้างชิปเดิม ให้เป็นชิปย่อยและทำงานได้เหมือนเป็นซับเซต ดังนั้นคำสั่งที่พัฒนากันต่อมาก็ยิ่ง สลับซับซ้อนยิ่งขึ้น นั่นคือ กลายมาเป็น CISC นั่นเอง อย่างไรก็ตามการพัฒนา CISC ก็ได้ใช้เทคนิคการเพิ่มประสิทธิภาพโดยรวมหลายอย่าง ที่ทำให้การพัฒนาในยุคหลัง คือ RISC นำมาใช้เช่น มีการทำ Pipelining การพรีเฟตซ์ และการใช้แคช

ทำไมผู้ออกแบบ com จึงเลือก RISC

การพัฒนาการของคอมพิวเตอร์เป็นไปตามแนวทางของ CISC คือทำให้มีชุดคำสั่งที่สลับซับซ้อนมากยิ่งขึ้น และเหตุที่ทำ ให้ผู้พัฒนาเลือกการพัฒนาไปตามแนวนี้คือความต้องการให้ คอมแพติเบิลกับของเดิม เมื่อรากฐานของผู้ใช้มีมาก ผู้ใช้คุ้นเคยกับซีพียูหรือคำสั่งของซีพียูมากแล้ว ครั้นพัฒนาการต้องการทำให้ซีพียูดีขึ้น ก็ต้องทำให้คอมแพติเบิลกับของเดิมและเพิ่มคำสั่งต่าง ๆ เข้าไปอีก การเพิ่ม ขีดความ สามารถบางสิ่งบางอย่างเข้าไปอีกนี้ ทำให้เกิดความซับซ้อนในตัวซีพียูมากขึ้น คำสั่งที่เพิ่มเติมทีหลังจึงเป็นคำสั่งพิเศษโดยที่ไม่ไปกระทบกับซอฟต์แวร์เดิมที่มีอยู่แล้วผู้ออกแบบต้องการลดช่องว่างของซอฟต์แวร์ ระหว่าง ภาษาชั้นสูงกับภาษาเครื่อง ด้วยความพยายามที่จะสร้างคำสั่งภาษาเครื่องให้ใกล้หรือเข้าหาภาษาระดับสูงได้มากที่สุด โปรแกรมเมอร์ที่เขียนภาษาชั้นสูงเมื่อคอมไพล์แปลความมาเป็นภาษาเครื่อง ก็แปลได้โดยตรงหรือง่ายขึ้นดังนั้นคำสั่ง ในภาษาเครื่องจึงต้องสร้างให้ซับซ้อน เพื่อรองรับภาษาระดับสูง การออกแแบโดยใช้ไมโครโค้ด เพื่อให้ฟังก์ชันในระดับสูงของซอร์ฟแวร์มาเป็นไมโครโค้ด หรือจากไมโครโค้ด มาเป็นฮาร์ดแวร์ได้โดยตรง เช่นการมีแมตโคโปรเซสเซอร์ ที่ทำฟังก์ชันเทียบเท่ากับ โปรแกรมย่อยของภาษา
ระดับสูงเช่น การหาค่ลอการิธึมตรีโกณมิติเป็นต้น การสร้างไมโครโค้ดเข้าสู่?าร์ดแวร์จึงทำให้กลไกการ ทำงานสลับซับซ้อนมากขึ้น การมีเครื่องมือ CAD ช่วยออกแบบคอมพิวเตอร์ จึงทำให้ผู้ออกแบบสามารถออกแบบคอมพิวเตอร์ได้ซับซ้อนยิ่งขึ้น และให้

มีลักษณะตามวัตถุประสงค์เฉพาะได้ CAD จึงมีผลทำให้สามารถออกแบบชิป หรือโครงสร้างสถาปัตยกรรมที่ซับซ้อนได้

จากแนวความคิดที่พยายามทำให้คอมพิวเตอร์ทำงานได้เร็วขึ้น โดยพยายามคิดหาสถาปัตยกรรมคอมพิวเตอร์ที่เหมาะสม ในราวปี พ.ศ. 1972 จอร์พ คอด (Cocke) แห่ง ไอบีเอ็ม ได้เฝ้าสังเกตการทำงานของเครื่องคอมพิวเตอร์ของเครื่องไอบีเอ็ม 360 พบว่ามีคำสั่งที่ยุ่งยากและต้องใช้เวลาทำงานแต่ละคำสั่งหลายลูก สัญญาณนาฬิกาแต่ละคำสั่งก็สั้น ยาวต่างกัน เขามีความคิดว่าเป็นไปได้หรือไม่ที่จะทำให้คำสั่งของคอมพิวเตอร์ไม่ต้องวิ่งในระดับไมโครโค้ดที่ต้องใช้สัญญาณ นาฬิกาหลายลูก แต่ทำให้เสร็จในสัญญาณนาฬิกาลูกเดียว ผลของความคิดนี้จึงเกิดการรวมกลุ่มวิจัยระบบ 801 ขึ้นโดยอาศัย พื้นฐานของแนวความคิดดังนี้

หนึ่งคำสั่งใช้เวลาหนึ่งลูกของสัญญาณ ทีมงานต้องการหาโครงสร้างของฮาร์ดแวร์ที่ทำให้การทำงานของคอมพิวเตอร์ ทำงานแต่ละคำสั่งได้เสร็จในเวลาหนึ่งลูกของสัญญาณ นั่นคือถ้าเป็น 10 Mhz ก็ทำได้ 10 ล้านคำสั่งในเวลา 1 วินาที ด้วยความคิดนี้ก็ต้องทำคำสั่งให้ง่ายขึ้น ไม่ยุ่งยากซับซ้อน

โครงสร้างหน่วยความจำแบบลำดับ เพื่อให้ซีพียูไม่เสียเวลาในการอ่านข้อมูลในหน่วยความจำ หรือต้องรอเวลา ดังนั้นโครงสร้างหน่วยความจำต้องมีลำดับคือสามารถเข้าถึงได้ทันทีและรวดเร็ว แนวความคิดนี้เรื่องหน่วยความจำแบบ แคชจึงเกิดขึ้น คอมไพเลอร์เป็นสิ่งที่นักโปรแกรมต้องพึ่ง เมื่อชุดคำสั่งมีเพียงพื้นฐานการเขียนโปรแกรมจึงต้องอาศัยคอมไพเลอร์ที่ดี ผู้เขียนโปรแกรมจึงเขียนโปรแกรมระดับสูงแล่วให้คอมไพเลอร์แปลงไปเป็นคำสั่งระดับภาษาเครื่องให้

ประวัติความเป็นมาของ RISC

ในราวปี ค.ศ. 1975 มหาวิทยาลัยแคลิฟอเนีย ที่เบร์กเล่ย์ ก็สร้างโครงงานพัฒนาคอมพิวเตอร์ โดยใช้ชื่อโครงการว่า RISC โครงงานนี้ สร้างเพื่อให้นักเรียนที่เรียนในระดับบัณฑิตศึกษา ได้ทำการวิทยานิพนธ์เกี่ยวกับโครงสร้างสถาปัตยกรรมของคอมพิวเตอร์เหล่านี้ เพื่อทำให้ประสิทธิภาพดีขึ้น ต่อมาได้เพิ่มโครงงาน SOAR และใช้ CAD ช่วยในการออกแบบและสร้างชิป RISC I เป็นซีพียูที่มีชุดคำสั่งที่ทำงานด้วยเวลาสั้น ๆ ดังนั้นการประมวลผลข้อมูลจะต้องทำกับรีจิสเตอร์ จะมีคำสั่งที่เข้าถึงหน่วยความจำเพียงการโหลดและสตอร์เท่านั้น การทำงานของคำสั่งส่วนใหญ่

เสร็จในหนึ่งไซเกิล คำสั่งที่ใช้มี 31 คำสั่ง ซึ่งสามารถบรรจุเป็นรหัสเพียงขนาด 32 บิต การถอดรหัสคำสั่งสามารถทำได้ทันทีด้วยรูปแบบที่แน่นอน ลักษณะพิเศษของ RISC I คือ มีรีจิสเตอร์ภายในมาก โดยมีรัจิสเตอร์มากกว่าร้อยตัว รีจิสเตอร์เหล่านั้จัดเรียงตัวกันเป็นกลุ่มและทับซ้อนกัน เพื่อให้มีการเรียกข้อมูลมาใช้หรือส่งผ่านข้อมูลถึงกันได้โดยไม่เสีย เวลาไปรับจากหน่วยความจำ RISC I จึงเป็นซีพียูที่ลดการติดต่อระหว่างซีพียูกับหน่วยความจำ จากรายงานของ RISC I และ RISC II กล่าวว่าความเร็วในการทำงานของ RISC เพิ่มขึ้น 2 ถึง 4 เท่าเมื่อเทียบกับการทำงานแบบ CISC เช่น VAX หรือ 68000

ทำไม RISC จึงทำงานได้เร็วขึ้น

ไม่นานนักหลังจากที่ RISC I ได้รับการพัฒนาขึ้นที่มหาวิทยาลัยสแตนฟอร์ด ก็มีการพัฒนาซีพียูที่ใช้สถาปัตยกรรมแบบ RISC เช่นกัน แต่ให้ชื่อว่า MIPS Microprocessor without Interloged Pipe State) ซึ่งเป็นซีพียูที่เน้นในเรื่องการทำงานที่รวดเร็วเช่นกัน เพื่อให้การทำงานได้ตามวัตถุประสงค์ โครงสร้างการทำงานภายใน จึงต้องใช้หลักการแบบขนาน ซีพียู RISC I มีลักษณะการวางรีจิสเตอร์สนับสนุนขบวนการทำงานภายใน 3 กระบวนความ โครงสร้างของ RISC ทั่วไป ก็ต้องทำเช่นว่านี้ โดยให้การทำงานมีลักษณะตามปรัชญาของ RISC ดังนี้
ทำงานได้ในไซเกิลเดียวต่อคำสั่ง ความจริงแล้วการทำงานหนึ่งคำสั่งต้องใช้เวลามากกว่าหนึ่งไซเกิล แต่ที่ทำได้เพราะ ทำให้คำสั่งเหล่านั้นมีลักษณะเป็นไปป์และขนานกัน จึงทำให้เวลาโดยรวมเมื่อเฉลี่ยออกมาแล้วได้คำสั่งละไซเกิลเดียว ออกแบบโหลด/สตอร์ เพื่อลดการทำงานที่ล่าช้า คำสั่งเหล่านี้จะต้องแน่นอนและทำงานได้ในลักษณะ ขนานตามกระบวน การที่กำหนด ใช้การควบคุมด้วยฮาร์ดแวร์ โดยอาศัยวงจรอิเล็กทรอนิคส์ที่ทำให้การทำงานเป็นไปในลักษณะที่ทำคำสั่งได้ในไซเกิลเดียว โดยไม่ต้องใช้ไมโครโค้ด มีจำนวนคำสั่งน้อย และการอ้างแอดเดรสจำกัด เพื่อให้โครงสร้างตายตัวในเรื่องไซเกิลการทำงานจึงต้องลดจำนวนคำสั่งลงให้เหลือเท่าที่จำเป็น เพื่อให้?าร์ดแวร์ควบคุมได้ ฟอร์แมตคำสั่งง่าย เพื่อการแปลความหมายและกระทำความด้วยฮาร์ดแวร์ได้ รูปแบบคำสั่งจึงต้องง่าย ต้องใช้เวลาคอมไพล์ภาษาระะดับสูงมาก เพื่อเปลี่ยนแปลงภาษาระดับสูงมาเป็นระดับต่ำต้องใช้คอมไพเลอร์ที่แปลงมาเป็นรหัสภาษาเครื่องได้ โดยที่คำสั่งห่างไกลทางความหมายจากภาษาระดับสูงมาก ดังนั้นจึงต้องเสียเวลาในการคอมไพล์นานกว่า

การที่ RISC ทำงานได้เร็วกว่าเพราะ

การวางโครงสร้างการทำงานได้กำหนดส่วนของการทำงานไว้ชัดเจนและเป็นแบบขนาน เช่น MIPS โมเดล R2000 ได้วาง โครงสร้างคำสั่งมีการทำงาน 5 ขั้นตอน ดังนั้นจึงวางโครงสร้างการทำงานแบบขนานไว้ 5 ระดับ โดยแบ่งแยกส่วนของการทำงานแต่ละคำสั่งเป็น

IF การเฟตซ์คำสั่ง (Instruction Fettch) คือจังหวะที่เฟตซ์คำสั่งจากหน่วยความจำมาตีความหมาย

RD อ่านโอเปอร์แรนด์จากรีจิสเตอร์ในซีพียู

ALU การคำนวนในหน่วย ALU ตามรหัสของคำสั่งนั้น ๆ

MEM การติดต่อกับหน่วยความจำ

WB การเขียนข้อมูลผลลัพธ์ลงในรีจิสเตอร์

การใช้วิธีการของคำสั่งในการทำงานแบบขนานนี้ เป็นผลทำให้สามารถทำงานได้เสร็จในหนึ่งไซเกิล ทั้ง ๆ ที่แท้จริงแล้วต้องใช้ถึง 5 ไซเกิล แต่ทุกขณะจะมีส่วนของซีพียูทำงานขนานกันทั้ง 5 ส่วนนี้อยู่ ขณะที่เฟตซ์คำสั่งหนึ่งอยู่ภายในก็มีการทำ RD, ALU, MEM, และ WB ของแต่ละคำสั่งที่ผ่านมา ดังนั้นการเฟตซ์จึงเกิดได้ทุกไซเกิล

กลไกการทำงานในลักษณะนี้ใช้กับซีพียู RISC ทุกตัว ตั้งแต่จุดเริ่มต้นของการพัฒนาจนถึงปัจจุบัน โครงสร้างของคำสั่งจึงอยู่ที่ ALU ว่าจะรับข้อมูลได้กว้างเพียงไร เช่น ถ้ารับข้อมูลได้ 64 บิต และให้ ALU เป็นหน่วยคำนวนทางคณิตศาสตร์แบบตัวเลขโฟลตติ้งพอยต์ ก็ยิ่งทำได้เร็วยิ่งขึ้น เช่น ซีพียู I860 ของอินเทลก็ใช้หลักการของ RISC เช่นกัน

ชุดคำสั่งของ RISC

เพื่อให้เห็นโครงสร้างชุดคำสั่งที่ใช้ใน RISC ในที่นี้จะขอยกตัวอย่างชุดคำสั่งของ MIPS กระทำกับรีจิสเตอร์ 3 ตัว คือ scr1,scr2 และ dest กล่าวคือ scr เป็นรีจิสเตอร์ตัวทำงาน dest เป็นรีจิสเตอร์ผลลัพธ์ คำสั่งส่วนใหญ่ เป็นคำสั่งพื้นฐาน ที่เข้าใจได้ง่าย การออกแบบคำสั่งเหล่านี้มุ่งไปที่การใช้รีจิสเตอร์ภายใน ดังนั้นรีจิสเตอร์ภายในมักมีขนาดกว้าง ในที่นี้ใช้ขนาด 32 บิต (อินเทล i860 ใช้ขนาด 64บิต) หากจะพิจารณาคำสั่งที่แสดงจะพบว่าคำสั่งเพียงเท่านี้ ก็เพียงพอต่อการใช้งาน หรือการเขียนโปรแกรมให้ทำงานในสิ่งต่าง ๆ ตามต้องการได้แล้ว การออกแบบ RISCจึงเป็นสิ่งที่ใช้สถาปัตยกรรมที่แตกต่างจาก CISC โดยสิ้นเชิง

ความสามารถอยู่ที่การจัดการหน่วยความจำ เมื่อซีพียู RISC ทำงานด้วยคำสั่งที่ใช้กับรีจิสเตอร์เป็นหลักมีเพียง RD กับ ST เท่านั้น ที่ใช้จากหน่วยความจำ LD กับ ST จึงต้องเกี่ยวกับหน่วยความจำที่ซีพียูต้องติดต่อด้วย อย่างรวดเร็ว การที่ซีพียูต้องติดต่ออย่างรวดเร็วนี้ จึงต้องอาศัยหน่วยความจำแคช ดังนั้นประสิทธิภาพของ RISC จึงขึ้นอยู่กับการจัดโครงสร้างของหน่วยความจำด้วย โดยที่หน่วยความจำแคชจะต้องมีบทบาทที่ทำให้ซีพียูติดต่อแล้วพบข้อมูลเป็นส่วนใหญ่
โครงสร้างของแคชที่ใช้กับ RISC เป็นแบบ direct maped cache ใช้การติดต่อกับหน่วยความจำนี้จะทำให้หน่วยความจำต่อกับแคชในลักษณะที่มีการกำหนดตำแหน่งแน่นอน แคชแบบนี้จำทำให้การเข้าถึงทำได้เร็วและต้องการพื้นที่น้อยกว่า โครงสร้างการอินเตอร์รัปต์ยังคงใช้เหมือนเดิม ในซีพียูแบบ CISC ต้องอาศัยการทำงานในลักษณะโปรแกรมหลายระดับ โดยการใช้สัญญาณอินเตอร์รัปต์เข้ามาเป็นตัวสวิตซ์การทำงานในแต่ละส่วนของโปรแกรมกรณีของ CISC ก็เช่นกัน การอินเตอร์รัปต์ก็มีกลไกการทำงานคล้ายกันจะแตกต่างกันเพียงเล็กน้อยในเรื่องการทำงานภายในซีพียู ใน CISC อาจมีไมโครโค้ดคอยจัดการบางอย่างให้แต่แนวความคิดของการใช้งานยังคงเหมือนกัน

<u>Reduced Instruction Set Architecture</u>

ทำไมถึงต้องเรียกว่า CISC.

มีชุดคำสั่งมากมายที่มีแนวโน้มทำให้เป็นที่รู้จักกัน ซึ่งรวมถึงหัวข้อสำคัญของ คำสั่งและคำสั่งที่ซับซ้อนอีกมากมาย เหตุผลสำคัญ 2 ประการ ที่ทำให้มีแนวโน้มในการเปลี่ยนแปลง คือ

1) ต้องการ Compiler ที่ง่ายต่อการใช้งาน
2) ต้องกาเพิ่มประสิทธิภาพของการทำงานให้มากขึ้น

และทั้ง 2 เหตุผลนี้ ทำให้โปรแกรมเมอร์ได้พัฒนาไปสู่ภาษาระดับสูง และพยายามออกแบบเครื่องซึ่งสามารถสนับสนุนภาษาระดับสูงนี้

มันไม่ใช่จุดมุ่งหมายของเรื่องนี้ที่กล่าวว่าคนออกแบบ CISC ผิดจุดประสงค์แต่เป็นเพราะว่าเทคโนโลยีได้มีการพัฒนามาอย่างต่อเนื่อง และเพราะสถาปัตยกรรมนี้มีอยู่อย่างแพร่หลายมากกว่าใน 2 ขึ้นตอนในข้างต้นที่กล่าวมา โดยส่วนใหญ่จะเน้นตามจุดมุ่งหมายบางอย่างทีมีประสิทธิภาพ ในทางไปสู่ CISC และจัดอยู่ในพวกของ RISC (reduced instruction set computer => คอมพิวเตอร์ลดทอนคำสั่ง , เป็นคอมพิวเตอร์ระบบหนึ่งที่ปฏิบัติการได้เร็วมากเพราะกำหนดให้ Microprocessor แต่ละตัวทำงานจำกัดเฉพาะบางอย่าง)

เข้าใจง่าย

เหตุผลแรกการทำ Compiler ให้ง่าย และเหมือนจะแจ่มแจ้งหน้าที่ของ Compiler จะสร้างคำสั่งภาษาเครื่องตามลำดับ สำหรับประโยคของภาษาระดับสูง ถ้าคำสั่งของภาษาเครื่องคล้ายกับประโยคคำสั่งของภาษาระดับสูง เช่นนี้แล้วเราก็จะทำงานได้ง่ายและสะดวกขึ้น ด้วยเหตุผลนี้ถูกโต้แย้งโดยนักวิจัย RISC พวกเขาพบว่าคำสั่งภาษาเครื่องที่มีความซับซ้อน บ่อยครั้งที่มันไม่มีประโยชน์ เพราะ Compiler จะต้องหากรณีซึ่งระบุโครงสร้างแน่นอน หน้าที่ของมันเห็นได้จากการสร้างโค้ดให้มีขนาดเล็ก , ลดทอนจำนวนครั้งของการประมวลผลแต่ละคำสั่ง ๆ และเพิ่ม Pipelining ให้มันมีความยาวมากขึ้น

เหตุผลสำคัญอื่นที่จะอธิบาย CISC ควรจะสั้นกระทัดรัด และต้องมีความเร็วที่สูง ความคาดหวังทั้ง 2 อย่างนี้จะยืนยัน ว่าโปรแกรมขนาดเล็กควรจะมีความเร็วในการประมวลผลมากมีข้อดี จากโปรแกรมเล็ก ๆ เช่น โปรแกรมเล็กใช้ Memory น้อย ๆ มันจะช่วยประหยัดเนื้อที่ ปัจจุบัน Memory มีราคาแพง ดังนั้นประสิทธิภาพที่ได้เปรียบที่ไม่ต้องใช้เวลาในการแปลคำสั่งนาน ๆ มันจะสำคัญมาก ถ้าโปรแกรมย่อย ควรจะปรับปรุงก็มีได้ 2 ทาง

- ให้เป็นคำสั่งย่อย ๆ คือหน่วย Byte ในคำสั่งย่อย ๆ จะถูกนำมาประมวลผล

- โปรแกรมย่อยจะใช้เวลาน้อยมากในแต่ละ Page รวมทั้งลดความผิดพลาดในแต่ละ page ด้วย

ปัญหาเหล่านี้ไกลจากความเป็นจริง ซึ่งโปรแกรม CISC จะเล็กกว่าโปรแกรม RISC ในหลายสาเหตุโปรแกรม CISC อาจกล่าวได้ว่าเป็น สัญลักษณ์ของภาษาเครื่อง และอาจสั้นมาก ๆ เช่นคำสั่งย่อย ๆ แต่จำนวน Bit ใน Memory ได้ใช้เวลาเร็วมากโดยเราไม่อาจจะสังเกตได้ เคยมีบันทึกไว้ว่าอาจไม่ปลอดภัยนักเมื่อใช้ CISC บน RISC และมันเป็นที่น่าสนใจพอ ๆ กับ VAX(คือ คอมพิวเตอร์ยี่ห้อ หนึ่งผลิตโดยบริษัท Digital Equipment Corp [DEC] เดิมบริษัทนี้ผลิตเฉพาะเครื่อง Mainfram และ Minicomputer)ซึ่ง VEX นี้มีคำสั่งซับซ้อนมากกว่า PDP-11 และเป็นที่ยอมรับว่ามีขนาดเล็กและมีประสิทธิภาพมาก ผลสรุปได้ยืนยันโดยนักวิจัยของ IBM [Radi83] ผู้ซึ่งพบว่า IBM 801 (a RISC) ผลิตโค้ดได้ 0.9 ครั้ง ขนาดของโค้ดนั้นอยู่บนเครื่อง IBM S/370

มีเหตุผลมากมายที่อาจทำให้ได้ผลสรุปที่แปลกใหม่ ปัจจุบันพร้อมแล้วที่ Compiler ของ CISCs มีแนวโน้มที่จะอนุญาต ต่อคำสั่งตัวอย่างซึ่งได้รวบรวม บางคำสั่งที่ไม่ได้พบ บ่อยนักไว้ด้วยเพราะว่ามันเต็มไปด้วยคำสั่วมากมายบน CISC เป็นที่ต้องการของ Opcode ขนาดใหญ่ , ผลิตชุดคำสั่งขนาดยาว และท้ายที่สุด RISCs มีแนงโน้มที่จะให้เป็น มากกว่าหน่วยความจำอ้างอิงและต้องการให้มันเป็น Bit ย่อย ๆ ยกตัวอย่างเช่นผลกระทบล่าสุดของม้นได้มีการถกเถียงมาจนถึงทุกวันนี้

ดังนั้นแนวความคิดของ CISC จะผลิตโปรแกรมขนาดเล็ก ๆให้กับผู้ต้องการความท้าทาย โดยอาจไม่ได้ตระหนักถึง มี 2 แรงกระตุ้นสำหับการเพิ่มคำสั่งที่ซับซ้อนให้คำสั่งถูกประมวลผลได้เร้วขึ้น เครื่องหนึ่ง ๆ คำสั่งอาจมากกว่าคำสั่งเก่า ๆ หลาย ๆ คำสั่งอย่างไรก็ตาม……

เพราะว่า โอนเอียงไปทางการใช้ของsimpler instruction บางทีอาจไม่เป็นอย่างนั้น control unit ทั้งหมดต้องสร้างความซับซ้อนมาก และ/หรือ microprogram control store ต้องสร้างให้ใหญ่ เพื่อความสะดวกของinstruction set แต่ละอันของ execute time ที่เพิ่มขึ้นเป็นเหตุผลของsimple instructions

8	16	16	16
Add	B	C	A

Memory-to-Memory
I = 56, D = 96, M = 152

8	4	16		
Load	rB	B		
Load	rC	B		
Add	rA	rB	rC	
Store	rA	A		

Register-to-Memory

(a) A ← B + C

8	16	16	16
Add	B	C	A
Add	A	C	B
Sub	B	D	D

Memory-to-Memory
I = 168, D = 288, M = 456

8	4	4	4
Add	rA	rB	rC
Add	rB	rA	rC
Sub	rD	rD	rB

Register-to-Memory
I = 104, D = 96, M = 200

(b) A ← B + C; B ← A + C; D ← D − B

I = Size of executed instructions
D = Size of executed data
M = I + D = Total memory traffic

Figure 12.5 Two Comparisons of Register-to-Register and Memory-to-Memory Approaches.

ในความเป็นจริง บางอย่างที่เพิ่มขึ้นที่พบนั้นมีความเร็วเพิ่มขึ้นในการทำงานของ complex functions กำหนดไม่ใช่อย่างเดียวกัน ด้านผลกระทบที่control store มีต่อ instruction cache ในHardware architectในส่วนของpositionพยายามที่จะตัดสินใจเลือกsubroutinesหรือfunctions ที่จะใช้งานบ่อยและส่งไปที่control stroe โดยใช้เป็นเครื่องมือในmicrocode แต่ละผลลัพธ์นี้มีข้อสนับสนุนที่น้อยกว่าในระบบของ S/390เช่นคำสั่งของTranslate and Extended-Precision-Floting-Point-Divideอยู่ในหน่วยเก็บความจำความเร็วสูง ในขณะที่เหตุเกี่ยวข้องในการติดตั้ง ได้ทำการส่งหรือเริ่มติดขัดในความช้าของMain memory

ลักษณะเฉพาะของRISC

-1คำสั่ง/รวม

-Register-to-Register operations

-addressing modes ที่เข้าใจง่าย

-รูปแบบคำสั่งที่เข้าใจง่าย

นี่คือการสรุปย่อของลักษณะเฉพาะเหล่านี้ ตามชนิดของตัวอย่างที่ได้พบภายหลังในบทนี้

ลักษณะเด่นอย่างแรก

คือ one machine instruction per machine ความหมายของmachine cycle คือ เวลาที่ทำให้operant 2ตัวจากregister ทำงานที่ALU operation และเก็บผลลัพธ์ไว้ใน Registerดังนั้น RISC machine instructionจึงไม่ได้ยุ่งยากมากกว่าและยังทำงานได้รวดเร็วด้วยใน microinstructions บนเครื่องCISCจากตัวอย่าง1รอบคำสั่งต้องการmicrocodeน้อยหรือไม่มีเลยการทำงานของคำสั่งนี้ทำงานได้เร็วกว่าในเครื่องแบบอื่นๆเพราะว่าไม่ต้องมีการเข้าถึงmicroprogram control ระหว่าง การทำงานของคำสั่ง

ลักษณะเฉพาะที่2

คือ operations ส่วนมากจะเป็นแบบregister-to –register กับการLOAD แบบง่ายๆและcontrol unit นั้น จากตัวอย่าง RISC instruction set จะประกอบไปด้วยหนึ่งหรือสอง คำสั่งADD VAXมี25 แตกต่างคำสั่งADD ผลดีอื่นๆของสถาปัตยกรรมแบบนี้คือ สนับสนุนส่วนที่ดีที่สุดของregister ดังนั้นบ่อยครั้งที่การเข้าถึงoperandยังค้างอยู่ในหน่วยเก็บความจำความเร็วสูง

emphasis บนregister-to-register operations เป็นทั้งหมดของ RISC design ในเครื่องอื่นๆที่มีคำสั่งเช่นนี้แบบเดียวกัน แต่ประกอบไปด้วย memory-to-memory และการรวมการทำงานของ register/memoryพยายามที่จะเปรียบเทียบ

ลักษณะเฉพาะข้อที่3คือการใช้SAM

ลักษณะเฉพาะตัวคือการใช้SAM เกือบทุกคำสั่งของRISCจะใช้Register addressing เบื้องต้น มีโหมดพิเศษหลายโหมด เช่น การแทนตำแหน่งและยังมีโหมดอื่นที่ซับซ้อนอีกมากได้ประกอบขึ้นในsoftwareจากแบบง่ายๆ

ลักษณะเฉพาะตัวสุดท้าย คือ การใช้ SIF โดยทั่วไปแค่หนึ่งหรือมากกว่านี้เล็กน้อย รูปแบบที่ใช้ ความยาวคำสั่งจะกำหนดและวางแนวทางบนขีดจำกัดของคำ การหาตำแหน่ง กำหนด opcodeโดยเฉพาะ การออกแบบลักษณะเด่นที่มีตัวเลขที่เป็นผลดีในส่วนที่มีการกำหนดไว้ opcode decoding และ การเข้าถึงของregister จะเกิดขึ้นในเวลาเดียวกัน รูปแบบนี้เป็นรูปแบบง่ายๆของCU คำสั่งที่รับมานั้นมีความเหมาะสมเพราะ word –length unitที่รับมา การปรับแนวของขอบเขตของคำนั้นแสดงว่า คำสั่งเดียวจะไม่ทำเครื่องหมายหน้าขอบเขต

ในเวลาเดียวกันลักษณะเฉพาะนี้สามารถที่จะเข้าถึงผลดีที่จะเป็นไปได้ของRISC ประโยชน์นี้แบ่งได้เป็น2ประเภทใหญ่ๆ เกี่ยวเนื่องกับประสิทธิภาพและเกี่ยวข้องกับ VLSI

ในเรื่องของประสิทธิภาพ"circumstantial evidence"เสนอว่า อันดับแรกมีผลมากคือcompilerที่เหมาะสม ต่อการพัฒนา ด้วยคำสั่งที่ง่ายนี้ มีโอกาสมากสำหรับย้ายฟังก์ชันออกจากloop ประสิทธิภาพสำหรับ การใช้registerให้ได้ประโยชน์สูงสุด และอื่นๆ ซ้ำยังเป็นไปได้ในส่วนของการคำนวณ

จุดที่2 จุดเด่นคือทำให้เกิดคำสั่งมากมายโดยตัวแปรโปรแกรมมักจะมีความสัมพันธ์ง่ายๆ มันดูเหมือนว่าเหตุผลที่CUสร้างคำสั่งนี้โดยเฉพาะและแทบจะไม่ได้ใช้หรือไม่สามารถexecute microcode พวกนั้นจะเร็วกว่า เปรียบเทียบกับCISC

จุดที่3 เกี่ยวเนื่องกับการใช้คำสั่ง Piple นักค้นคว้าRISCรู้สึกว่าเทคนิคคำสั่งของpipeสามารถประยุกต์ได้อย่างมีประสิทธิภาพมากด้วยการทำให้ชุดคำสั่งน้อยลง เหล่านี้คือจุดที่ตรวจสอบในรายละเอียดหลายๆอย่างตอนนี้

จุดสุดท้ายเป็นส่วนที่ไม่สำคัญมาก ที่จุดนี้กล่าวถึง โปรแกรมRISCนั้นน่าจะตอบรับอย่างรวดเร็ว ทำให้ขัดจังหวะการทำงาน เพราะการขัดจังหวะจะเกิดขึ้นในระหว่าง การทำงานเบื้องต้น สถาปัตยกรรมกับความยุ่งยาก เช่น การถูกจำกัดทำให้ขาดช่วง ขีดจำกัดของคำสั่ง หรือต้องกำหนดความแตกต่างของจุดที่ขัดจังหวะการทำงานและเครื่องมือสำหรับเริ่มคำสั่งใหม่

ในกรณีของการพัฒนาประสิทธิภาพของ ISA นั้นยังห่างจากความเป็นจริงอยู่อีกมาก จากจำนวนที่ได้ศึกษาไปแล้วนั้นไม่สามารถเปรียบเทียบบนเครื่องที่เปรียบเทียบระหว่างเทคโนโลยีและพลังงานได้ นอกจากนี้สิ่งที่ได้ศึกษาไปทั้งหมดนั้นไม่ได้พยายามที่จะแยกผลกระทบของ ISA และผลกระทบของขนาดของไฟล์ อย่างไรก็ตาม "Circumstatial evidence" กล่าวว่าเป็นแค่ข้อเสนอแนะเท่านั้น

ผลดีอย่างที่สองที่เป็นไปได้คือความชัดเจนที่บอกถึงเครื่องมือของ VLSI เมื่อใช้ VLSI ในการออกแบบและใช้เป็นเครื่องมือของการเปลี่ยนส่วนสำคัญของ processors เช่น IBM S/390 และ VAX ประกอบด้วย ผังวงจร 1 ผัง หรือมากกว่าที่บรรจุ SSI และ MSI จากการเกิดขึ้นของ LSI และ VLSI มันมีความเป็นไปได้ที่จะวาง processors ทั้งหมดลงบนชิพเพียงตัวเดียว สำหรับ single-chip processor มีวิธีกระตุ้น RISC 2 อย่าง คือ

1. มีการปล่อย performance ออกมา on-chip delay นั้นใช้ช่วงเวลาสั้นกว่า interchipdelays เราเคยเห็นตัวอย่างคำสั่ง และการเข้าถึงของ local scalars ในความจริงแล้วทั้งหมดนี้เกิดกระตุ้นปล่อยครั้ง การออกแบบ Berkeley RISC chips ได้พิจารณาไว้ในใจ ด้วยเหตุที่ตัวอย่าง single-chip microprocessor ได้ใช้ไปครึ่ง

หนึ่งของพื้นที่ของ micro control store RISC I chip สนใจกับ 6% ของพื้นที่ของ control unit [sher84]

2. VLSI-related เป็นการออกแบบและimplementation time VLSI processor นั้น ยากแก่การพัฒนา การที่จะเชื่อมั่นในความสามารถของ SSI/MSI ได้ ผู้ออกแบบต้องทำการออกแบบผังวงจร , โครงงาน และลักษณะรูปทรงของอุปกรณ์นั้น แต่ละชิ้นเกี่ยวกับ RISC การที่จะ process นั้นไม่ง่ายเลย โดยดูจากตาราง 12.7 [FIT Z81] ถ้าการเพิ่มประสิทธิภาพของ RISC chip นั้นเท่ากัน เพื่อสามารถเปรียบเทียบ CISC microprocessor ดังนั้น ความสำคัญของ RISC จึงเข้าใกล้ความชัดเจนมากขึ้น

ลักษณะของ CISC กับ RISC

หลังจากที่เริ่มมีการตื่นตัว ของ RISC machnes ได้มีสิ่งที่เกิดขึ้นคือ

1. RISC ถูกออกแบบให้ดีขึ้น โดยรวมเอาลักษณะเด่นบางอย่างของ CLSC
2. CLSC ถูกออกแบบให้ดีขึ้น โดยรวมเอาลักษณะเด่นบางอย่างของ RISC

ผลลัพธ์ของการออกแบบ คือ เกิด Power PC ที่มีความโดดเด่น

ลักษณะเด่นของ Pentium ทู คือรวมเอาลักษณะเฉพาะของ RISC เข้าด้วยกัน ได้มีการสนใจที่จะเปรียบเทียบ [MASH95] เพื่อความเข้าใจ

จากตารางที่ 12.8 จำนวนของการประมวลผลและเปรียบเทียบ จำนวนของตัวเลขนั้นแสดงถึงลักษณะเฉพาะของเครื่องแต่ละเครื่อง สำหรับจุดประสงค์ของการเปรียบเทียบนั้น คือ เมื่อพิจารณา typical ของ RISC

Table 12.8 Characteristics of Some Processors

Processor	Number of instruction sizes	Max instruction size in bytes	Number of addressing modes	Indirect addressing	Load/store combined with arithmetic	Max number of memory operands	Unaligned addressing allowed	Max number of MMU uses	Number of bits for integer register specifier	Number of bits for FP register specifier
AMD29000	1	4	1	no	no	1	no	1	8	3[a]
MIPS R2000	1	4	1	no	no	1	no	1	5	4
SPARC	1	4	2	no	no	1	no	1	5	4
MC88000	1	4	3	no	no	1	no	1	5	4
HP PA	1	4	10[a]	no	no	1	no	1	5	4
IBM RT/PC	2[a]	4	1	no	no	1	no	1	4[a]	3[a]
IBM RS/6000	1	4	4	no	no	1	yes	1	5	5
Intel i860	1	4	4	no	no	1	no	1	5	4
IBM 3090	4	8	2[b]	no[b]	yes	2	yes	4	4	2
Intel 80486	12	12	15	no[b]	yes	2	yes	4	3	3
NSC 32016	21	21	23	yes	yes	2	yes	4	3	3
MC68040	11	22	44	yes	yes	2	yes	8	4	3
VAX	56	56	22	yes	yes	6	yes	24	4	0
Clipper	4[a]	8[a]	9[a]	no	no	1	0	2	4[a]	3[a]
Intel 80960	2[a]	8[a]	9[a]	no	no	1	yes[a]	—	5	3[a]

[a] RISC that does not conform to this characteristic.
[b] CISC that does not conform to this characteristic.

การลดจำนวนคำสั่งของเครื่อง

1.คำสั่งต้องมีขนาดเดียว

2.ขนาดจะต้องเป็นเป็นขนาด4bytes

3.ตัวเลขของdata addressing made ต้องมีขนาดเล็ก โดยปกติจะต้องน้อยกว่า 5 ในตาราง

ทะเบียนและต้นฉบับจะไม่ถูกนับและรูปแบบจะแตกต่างกับรูปแบบเก่า ซึ่งถูกแยกกันนับ

4.การที่Addressing ไม่ตรงซึ่งมันต้องการให้คุณทำให้มันเข้าถึงข้อมูลที่จะไปถึง Address ของตัว

แปรในMemory

5.ไม่มีการทำงานที่ทำทั้ง local/store การทำงานเกี่ยวกับตัวเลข

6.ในหนึ่ง Memory address ตัวแปรต้องไม่มากกว่าคำสั่ง

7.ไม่มีอะไรที่สนับสนุนเหตุผลของการรวมกันของข้อมูลจากการ ทำงานของการ local และก็store

8.จำนวนที่มากที่สุดของการใช้ในการจัดการกับ Memory จากข้อมูลที่อยู่ในคำสั่ง

9.จำนวนบิตของ Integer register ต้องระบุให้มากกว่าหรือเท่ากับ 5 วิธีการนี้จะทำให้ integer

register ohvpmujl6f8nv 32 ซึ่งจะทำให้ในการอ้างถึงนั้นชัดเจน

10.จำนวนบิตของ Flocting-point register ต้องระบุให้มากกว่าหรือเท่ากับ 4 วิธีการนี้จะทำให้
Integer register น้อยที่สุดคือ 16 ซึ่งจะทำให้ในการอ้างถึงนั้นชัดเจน

ข้อ 1 ถึง 3 นั้นเป็นเครื่องชี้บอกว่าคำสั่งจะถูกถอดรหัสด้วยความลำบาก ข้อ 4 ถึง 8 เป็นการแสดงความง่ายหรือยากของ piplining โดยเฉพาะอย่างยิ่งการที่อยู่ในที่นั้นมันต้องการ Virturlmemory ข้อ 9 ถึง 10 ความสามารถในการใช้ข้อได้เปรียบของ compliler

RISC Pipelining

Pipelining and Regular Tnstruction

คำสั่งของ Pipelining ส่วนมากจะใช้ในการเพิ่มประสิทธิ์ภาพ คำสั่งส่วนมากจะเป็นแบบRegister และจำนวนรอบของคำสั่งจะถูกตามด้วย 2 วิธี ดังนี้

1. I : Instruction Fetch
2. E : Execute

การlocal และบันทึกข้อมูลต้องตามด้วย 3 หัวข้อดังนี้

1. I : Instruction Fetch
2. E : Execute
3. D : Memory

รูป 12.6aรูปภาพแสดงเวลาที่Pip ไม่ได้ใช้คำสั่งในการเรียงลำดับ อย่างชัดเจนแสดงว่าเป็นการประมวลผลที่สูญเปล่า ตัวอย่างเหตุการณ์ Pสามารถปรับปรุงประสิทธิ์ภาพให้ดีขึ้น รูป 12.6b แสดงระบบ two-way Pipline ในเวลาเดียวกันจะเกิดความแตกตางของคำสั่ง 2 คำสั่ง
คือ I และ Z ความแตกต่างนี้จะส่งผลกระทบอัตราการเรียงลำดับขั้นตอนเพิ่มขึ้นเป็น 2 เท่า 2ปัญหานี้เราป้องกันความเร็วที่เพิ่มมากขึ้นจากการเรียงข้อมูลตั่งแต่แรก 1.เราสันนิษฐานว่าจะใช้Memory เพียงตัวเดียวในการรับข้อมูลต่อ 1 ขั้นตอนคำสั่งในการแทรกเข้าไปกับการไหลของคำสั่ง โดยcomplier หรือ assembler

Pipelining สามารถปรับปรุงพื้นฐานเพิ่มขึ้น โดยใช้Memory 2 ตัวในการส่งค่าต่อ 1 ขันตอน ผลที่ได้แสดงในรูป 12.6c ขณะเดียวกันคำสั่งเพิ่มเป็น 3 คำสังสามารถทำให้เกิดความซับซ้อนเพิ่มขึ้น และ การปรับปรุงก็ทำให้เกิดปัจจัยอีก 3 ประการ คำสั่งคำสั่ง

อาจจะเป็นสาเหตุที่ทำให้เกิดความเร็วที่มากลดน้อยลง อย่างไรก็ตามจะมีผลกระทบกับข้อมูลอย่างอิสระ ถ้าคำสั่งต้องการตัวแปร นิพจน์ นั้นเป็นการเปลี่ยนแปลงโดยคำสั่งนั้นมาก่อนจำเป็นต้องเสียเวลาอีก แต่จะสามารถแก้ปัญหาโดย NOOP

Piplining อธิบายระยะการทำงานที่ดีที่สุดถ้าขั้นตอน 3 ขั้นเป็นช่วงระยะเวลาที่เท่ากันโดยประมาณ เพราะขั้นตอน E โดยปกติเกี่ยวข้องกับ ALU สาเหตุในนี้ เราสามารถแบ่งเข้าสู่ 2 ขั้นตอน
ย่อย ดังนี้

E1 : Regster file read

E2 : ALU operation and register write

เพราะความเรียบง่ายและความเป็นระเบียบของคำสั่งset การออกแบบขั้นตอนประมาณ 3-4 ขั้นเป็นการออกแบบที่ง่ายและรวดเร็วรูป 12.6d แสดงผล Pipeline four-way ถ้าเราใช้คำสั่ง
4 สั่งก็จะทำให้เกิดปัจจัยเพิ่มขึ้นอีก 4 และทำให้ความเร็วที่ใช้นั้นเกิดผลช้าลง การใช้ NOOPอธิบายเกี่ยวกับข้อมูลและความล่าช้าของส่วนย่อยนั้นได้

จำไว้ว่าตัวเลขเหล่านี้ถูกมีผลกระทบโดยความยาวของpipeline delay และ cpi พื้นฐานที่อย่างเกิดขึ้นอย่างกะทันหัน pipeline delay ที่ความยาวกว่าจะทำให้เพิ่มขึ้นใน penalty และเปอร์เซ็นต์ที่สูงกว่าของเวลาที่เสียไป delay ของ 1 รอบเวลาการทำงานเท่านั้นที่เล็ก คือ the R400 pipeline
รูปที่ 3.35 CPI pernalties สำหรับ 3 แบบแผน branch-predichtion และ pipeline ที่ลึกกว่า

ความแตกค่างระหว่างแบบแผนหลายๆแบบแผนถูกเพิ่มขึ้นกับ delay ที่ยาวกว่าอย่างคงที่ ถ้า Cpi พื้นฐานเป็น 1 และ brach เป็นเพียงแหล่งที่มาของ stall เท่านั้น pipeline ในอุดมคติจะเร็วกว่า pipeline ธรรมดาประมาณ1.56 เท่าที่ซึ่งใช้ในแบบแผน stall-pipeline ส่วนแบบแผน predict-untaken จะดีกว่าแแบบแผน stall-pipeline 1.13 เท่าภายใต้การสันนิษฐานที่เหมือนกัน
Branch Prediction ที่คงที่: การใช้เทคโนโลยี compiler Delayed branches เป็นเทคนิคอย่างหนึ่งที่แสดง pipleline hazard อย่างที่เราได้เห็น ประสิทธิภาพของเทคนิคนี้บางส่วนขึ้นอยู่กับว่าเราเดาทางที่ branch จะไปอย่างถูกต้องที่เวลาในการ compile ก็มีประโยชน์สำหรับทำตารางข้อมูล hazard
จงพิจารณากลุ่ม code ต่อไปนี้ :
……

การขึ้นอยู่กับของ sub และ BEQZบนคำสั่ง LW หมายความว่า stall จะถูกต้องการหลัง LW สมมุติว่าเรารู้ว่า branch นี้เกือบถูกใช้เสมอและค่าของ R7ไม่ถูกต้องการบน fall-throgh path แล้วเราสามารถเพิ่มความเร็วของโปรแกรมโดยเคลื่อนที่คำสั่ง ADD R7, R8, R9 ไปที่ตำแหน่งหลังLW โดยสอดคล้องกัน ถ้าเรารู้ว่า branch ไม่ค่อยจะถูกใช้และค่าของ R4 จะไม่ถูกต้องการบน path ที่ถูกใช้ ดังนั้นเราสามารถพิจารณาการเคลื่อนที่คำสั่ง OR หลังLW อีกอย่างหนึ่งเราสามารถใช้ข้อมูลทำตารางที่ดีกว่า branch delay ด้วยเพราะการเลือกวิธีทำตาราง delay ขึ้นอยู่กับการรู้พฤติกรรมของ branch

ในการแสดง optimization เหล่านี้เราจำเป็นต้องpredict branch อย่างคงที่ เมื่อเรา compile โปรแกรมในบทต่อไป เราจะตรวจสอบการใช้ของ dynamic prediction มีพื้นฐานจากความประพฤติโปรแกรม runtime เราจะมองดูที่วิธี compile-time ที่หลากหลายสำหรับการทำตาราง code เทคนิคเหล่านี้ต้องการ branch prediction ที่คงที่และ ดังนั้นข้อมูลในส่วนนี้เป็นอันตราย

มีวิธี 2 วิธีที่เราสามารถใช้ในการpredict brancher อย่างคงที่โดยการตรวจสอบ program behavior และ โดยการใช้โครงรางข้อมูลที่ถูกเก็บจาการรันของโปรแกรมก่อนๆ เราเห็นในรูป 3.25 ว่า branches ส่วนใหญ่ถูกใช้สำหรับ forward และ backward brancher ดังนั้นแบบแผนที่ง่ายที่สุดคือ predict branch เหมือนที่ถูกใช้

Optimization of Pipelining

เพราะว่าคำสั่ง RISC นั้นง่ายและเป็นระเบียบ ระบบPipeline สามารถใช้ได้อย่างมีประสิทธิ์ภาพการเปลี่ยนแปลงคำสั่งเล็กน้อยในช่วงเวลาและPipeline สามารถปรับปรุงและสั่งคำสั่งนั้นกลับ อย่างไรก็ตามโดยรวมอัตราการexecute ส่วนย่อยน้อยลง

การทดแทนสำหรับตัวแทนสำหรับตัวแปรอิสระ การจัดระเบียบ เทคนิคสามารถพัฒนาขึ้นไปอีก 1.พิจารณาคำสั่งย่อยความล่าช้านั้นPipeline นั้นก็ทำให้มีประสิทธิภาพเพิ่มขึ้นโดยเราจะทำส่วนที่ไม่ส่งผลกระทบจนถึงทั้งภายหลังการทำของคำสั่งตาม(ดังนั้นทำเวลาล่าช้า)คำสั่งย่อยพบตำแหน่งโดยทันทีทันใดเป็นการสั่งslotล่าช้าขั้นตอนใหม่แสดงในตาราง12.9ในคอลัม”normal branch” เราจะเห็นโปรแกรมแสดงคำ

สั่งเกี่ยวกับภาษาเครื่อง ตอมาจะexecute102คำสั่งที่101และ 102 เป็นการแลกเปลี่ยนกันรูป12.7 แสดง

Table 12.9 Normal and Delayed Branch

Address	Normal Branch	Delayed Branch	Optimized Delayed Branch
100	LOAD X,A	LOAD X,A	LOAD X,A
101	ADD 1,A	ADD 1,A	JUMP 105
102	JUMP 105	JUMP 106	ADD 1,A
103	ADD A,B	NOOP	ADD A,B
104	SUB C,B	ADD A,B	SUB C,B
105	STORE A,Z	SUB C,B	STORE A,Z
106		STORE A,Z	

คำสั่ง JUMP จะ fetched ก่อนคำสั่งADDอย่างไรก็ตามคำสั่ง ADD จะสั่ง ADD จะ fetched เสร็จก่อนที่JUMP จะEXECUTEเสร็จ

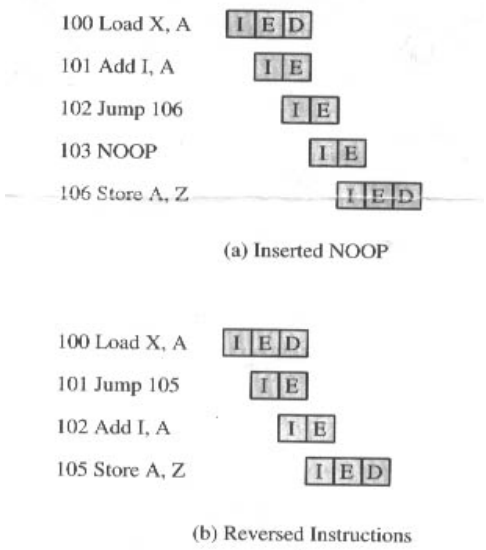

Figure 12.7 Use of the Delayed Branch.

การทำงานของคำสั่งจะสำเร็จเมื่อไม่มี conditional branches การเรียกและการค้นหาเมื่อ conditional branches procedure ไม่สามารถประยุกต์ได้ ถ้าเงื่อนไขโดยทดสอบในแต่ละขั้นซึ่งสามารถทำให้เปลี่ยนแปลง โดยทันที เมื่ออยู่หน้าคำสั่ง compiler สามารถค้นหาการเพิ่มคำสั่งหลังการbranch และจะพบว่าทั้งBenkeley RISC และ IBM 801 ส่วนใหญ่คำสั่ง Conditional branch สามารถoptimized ในรูปแบบนี้ได้ ([patt82a],[radi83])

ความเหมือนกันของระบบนี้คือ สามารถที่จะใช้คำสั่ง loadนี้ โดยที่จะเข้าระบบเป้าหมายในการload ของprocessor เมื่อ processor ทำงาน execution คำสั่งเรื่อยๆ จนกระทั่ง มาถึง

คำสั่งที่ต้องการมันจะหยุดตรงนั้นจนกระทั่งload ถ้า compiler สามารถปรับปรุงคำสั่งเพื่อช่วยในงานload ในpipeline จะทำให้ประสิทธิภาพเพิ่มขึ้น

สุดท้าย ผู้ใช้ควรออกแบบคำสั่ง pipeline ซึ่งไม่ควรที่จะแยกออกจากส่วนอื่นของ

optimzation techniques ประยุกต์กับระบบ ตัวอย่างเช่น [BRAD 916] แสดงรายละเอียดเกี่ยวกับคำสั่งและการจองเนื้อที่ของข้อมูลเพิ่งที่จะต้องตัดสินใจไปพร้อมๆกันให้บรรลุเพื่อประสิทธิภาพสูงสุด

สรุป เกี่ยวกับ RISC

RISC ย่อมาจาก Reduced Instruction Set Computing อ่านออกเสียงว่า “Risk” แตกต่างกับ CISC RISC คือการลดจำนวนคำสั่งลง ชิปของ RISC โดยปกติแล้วการพัฒนาและการผลิตจะถูกกว่า ตัวอย่างเช่น PROCESSOR ของ RISC ประมวลผลได้มากขึ้น75% ซึ่งทั้งสองสัญญาณนาฬิกาเท่ากัน RISC COMPUTER กำเนิดจาก IBM ซึ่งได้วิจัยในปี 1970

สำหรับ RISC นั้น ต้องการเพิ่มความสามารถในการทำงานของแต่ละ Instruction ให้ใช้เวลาใกล้เคียง 1 clocktime ให้ได้มากที่สุด ดังนั้นแต่ละคำสั่งที่อยู่ใน Instruction set จึงไม่ควรซับซ้อน นี่เป็นสาเหตุที่ RISC ตัดคำสั่งที่ซับซ้อนประเภทพวกคูณกับหารเลขจำนวนเต็มออกไป หมายความว่า complier จะต้องแปลคำสั่งที่ซับซ้อนเหล่านั้น ออกเป็นคำสั่งย่อยๆ หลายๆคำสั่งแทน ดังนั้น bytecode จึงมีขนาดไกล้เคียงกันเกือบทุกคำสั่ง แล้วถ้าแตกออกเป็นหลายๆ คำสั่งทำไมถึงเร็วได้ คำตอบคือ RISC ได้ใช้ผู้ช่วยอย่าง PIPELINE SUPER SCALAR และ CACHE เข้าช่วย ข้อดีทำให้การทำ Instruction decoding นั้นจะใช้เวลาที่สั้นมากกว่า cisc และเนื่องจาก bytecode มีขนาดที่ไกล้เคียงกัน จึงทำให้การ FETCH INSTRUCTION นั้นน้อยกว่าตามไป ยิ่งมี cache เข้ามาช่วยจึงเพิ่มความเร็วได้มาก ทำให้ CPU แบบ RISC สามารถเพิ่ม Clock speed ได้สูงกว่าแบบ CISC