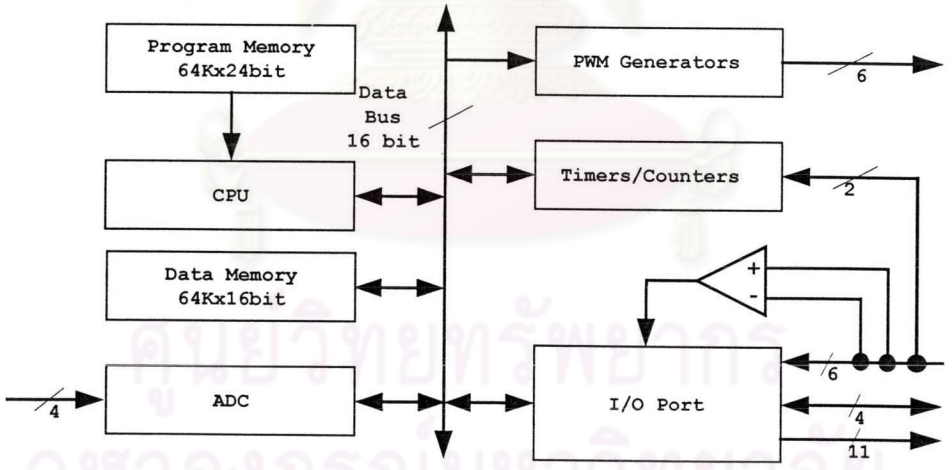
บทที่ 3
สถาปัตกรรมของ Q-Chip

3.1 โครงสร้างของ Q-Chip

Q-Chip เป็นไมโครคอนโทรลเลอร์ที่พัฒนาขึ้นในวิทยานิพนธ์นี้ ประกอบด้วยหน่วยประมวลกลางขนาด 16 บิต ซึ่งเป็นหน่วยประมวลผลกลางแบบฮาร์วาร์ด การทำงานเป็นแบบไปป์ไลน์สองขั้นตอน สามารถประมวลผลได้ 12 MIPS ที่สัญญาณนาฬิกา 12 MHz หน่วยความจำสำหรับโปรแกรมขนาด 24 บิตต่อหนึ่งตำแหน่ง หน่วยความจำสำหรับข้อมูลขนาด 16 บิตต่อหนึ่งตำแหน่ง และอุปกรณ์บริวารต่าง ๆ ดังแสดงในรูปที่ 3.1

หน่วยประมวลผลกลางทำงานโดยการเฟทช์คำสั่งจากหน่วยความจำสำหรับโปรแกรม ซึ่งระบุตำแหน่งด้วยตัวนับโปรแกรมขนาด 16 บิต โดยใช้บัสสำหรับเฟทช์คำสั่งโดยเฉพาะ ส่วนการโอนย้ายข้อมูลระหว่างหน่วยประมวลผลกลาง กับ หน่วยความจำสำหรับข้อมูล และ อุปกรณ์บริวาร ใช้บัสข้อมูลขนาด 16 บิต ชุดเดียวกัน ซึ่งระบุตำแหน่งด้วยบัสตำแหน่งขนาด 16 บิต และอุปกรณ์บริวารถูกกำหนดให้มีตำแหน่งในช่วง 32 ตำแหน่งแรก (0000H-001FH)



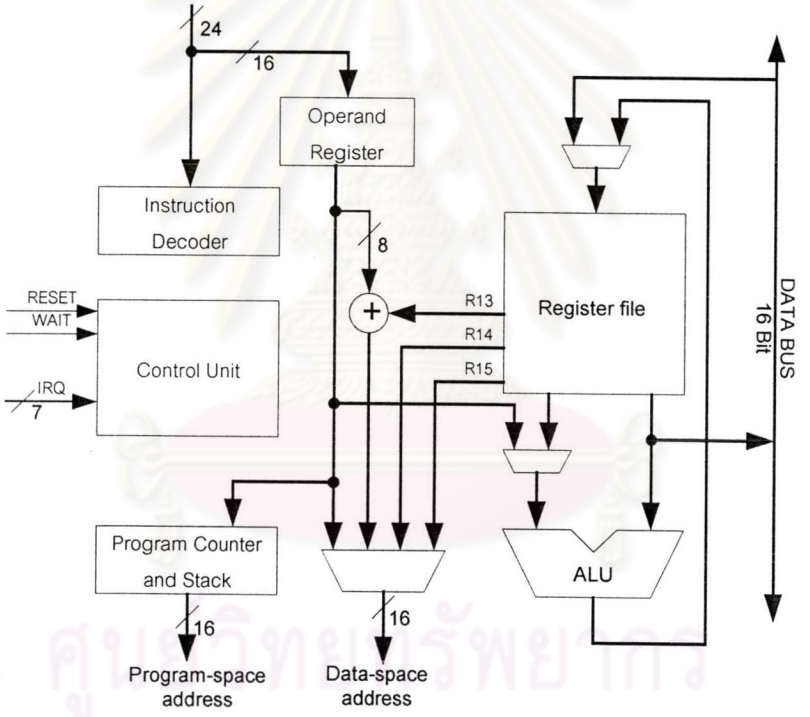
รูปที่ 3.1 โครงสร้างของ Q-Chip

3.2 สถาปัตยกรรมของหน่วยประมวลผลกลาง

สถาปัตยกรรมของหน่วยประมวลผลกลางเป็นแบบฮาร์วาร์ด ซึ่งมีหน่วยความจำสำหรับโปรแกรม กับหน่วยความจำสำหรับข้อมูล แยกจากกัน หน่วยประมวลผลกลางสามารถติดต่อ

หน่วยความจำทั้งสองได้พร้อมกัน ซึ่งสนับสนุนให้หน่วยประมวลผลกลางทำงานแบบไปป์ไลน์ได้ดี โดยหน่วยประมวลผลกลางโครงสร้างเป็นแบบไปป์ไลน์ 2 ขั้นตอน คือ ขั้นตอนเฟทช์คำสั่ง และขั้นตอนดำเนินการ เช่นเดียวกับไมโครคอนโทรลเลอร์ในท้องตลาดที่เป็นแบบริสก์ เช่น AVR [7] และ PIC16XXX [8] การที่มีโครงสร้างแบบไปป์ไลน์ช่วยเพิ่มความเร็วในการทำงานและการแบ่งขั้นตอนของไปป์ไลน์ให้มีหลายขั้นตอนเป็นการเพิ่มความเร็วในการทำงานในแต่ละขั้นตอนมากขึ้น แต่ก็จะเพิ่มความยุ่งยากในการจัดการควบคุมการทำงานมากขึ้นด้วย ดังนั้นการใช้โครงสร้างแบบไปป์ไลน์ 2 ขั้นตอนเป็นการเพิ่มความเร็วในการทำงานแต่วงจรในส่วนควบคุมมีความซับซ้อนไม่มาก

ส่วนประกอบหลักของหน่วยประมวลผลกลางคือ หน่วยคำนวณและตรรกะ แฟ้มรีจิสเตอร์ (Register file) และหน่วยควบคุม (Control Unit) และวงจรอื่น ๆ ดังแสดงในรูปที่ 3.2



รูปที่ 3.2 สถาปัตยกรรมของหน่วยประมวลผลกลาง

หน่วยประมวลกลางสามารถเฟทช์และถอดรหัสคำสั่งภายในช่วงเวลา 1 รอบสัญญาณนาฬิกา แล้วส่งไปทำงานในขั้นตอนดำเนินการโดยใช้เวลาดำเนินการอีก 1 รอบสัญญาณนาฬิกา ดังนั้นเวลาประสิทธิผลที่แต่ละคำสั่งใช้คือ 1 รอบสัญญาณนาฬิกา

จากรูปที่ 3.2 รหัสดำเนินการขนาด 24 บิตถูกส่งให้วงจรถอดรหัสคำสั่ง (Instruction Decoder) ทำการถอดรหัส ซึ่งรหัสดำเนินการ 16 บิตล่างเป็นฟิลด์ของตัวถูกดำเนินการจะถูกเก็บไว้ในรีจิสเตอร์สำหรับตัวถูกดำเนินการ (Operand register) เพื่อใช้สำหรับการประมวลผลในขั้นตอนดำเนินการ ข้อมูลจากรีจิสเตอร์สำหรับถูกดำเนินการใช้เป็นตัวถูกดำเนินการสำหรับหน่วยคำนวณและตรรกะในกรณีแอสเดรสซิ่งโหมดแบบใช้งานทันที (Immediate mode) และใช้เป็นค่า Offset โดยบวกกับค่าในรีจิสเตอร์ R13 เพื่ออ้างถึงข้อมูลในหน่วยความจำที่มีแอสเดรสซิ่งโหมดแบบตัวชี้ (Indexed mode) และระบุตำแหน่งข้อมูลในหน่วยความจำสำหรับข้อมูลในคำสั่งแบบโดยตรง (Direct mode) นอกจากนี้แอสเดรสซิ่งโหมดแบบรีจิสเตอร์โดยอ้อม (Register Indirect mode) โดยเลือกค่าจากรีจิสเตอร์ R14 หรือ R15 ในแฟ้มรีจิสเตอร์ โดยที่รีจิสเตอร์ทั้งสองสามารถเพิ่มค่าได้โดยอัตโนมัติหลังจากใช้งานแล้ว (Post-Increment)

3.2.1 <u>หน่วยคำนวณและตรรกะ</u>

หน่วยคำนวณและตรรกะ ทำหน้าที่ประมวลผลการดำเนินการคำนวณหรือทางตรรกะ โดยตัวถูกดำเนินการเป็นข้อมูลขนาด 16 บิต ตัวถูกดำเนินการสองตัวถูกอ่านมาจากแฟ้มรีจิสเตอร์และผลลัพธ์ที่ได้ถูกเขียนลงรีจิสเตอร์ตัวที่ถูกอ่านมาเป็นตัวถูกดำเนินการตัวแรก ดังในสมการ (1)

กำหนดให้ Rd: Destination Register

Rs: Source Register

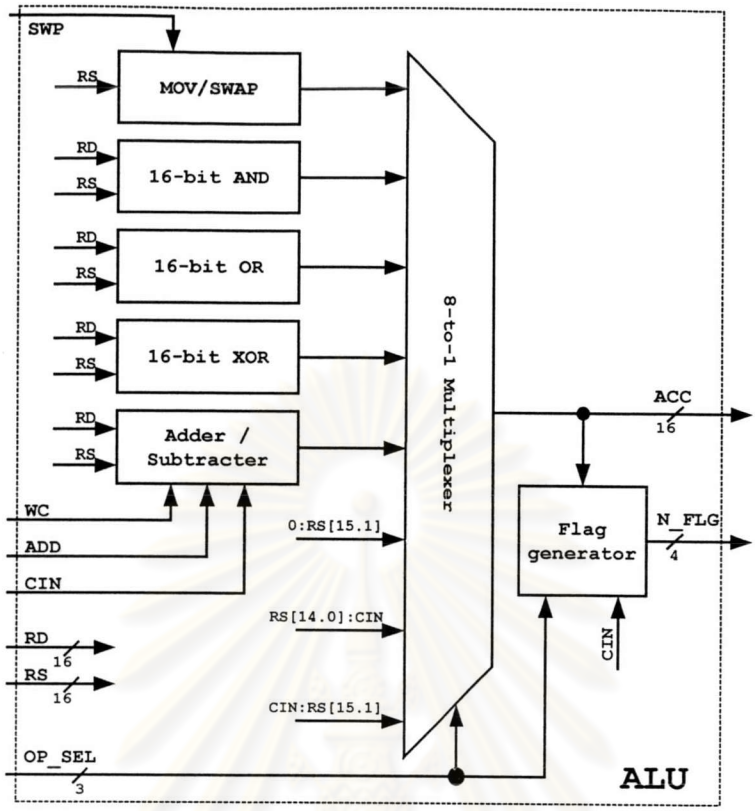
* : Operator

$$Rd = Rd * Rs \quad \ldots\ldots (1)$$

ในกรณีที่ตัวดำเนินการต้องการตัวถูกดำเนินการเพียงแค่ตัวเดียว หน่วยคำนวณและตรรกะจะนำตัวถูกดำเนินการจาก Rs ตามสมการ (2)

$$Rd = * Rs \quad \ldots\ldots (2)$$

ภายในหน่วยคำนวณและตรรกะ มีวงจรบวก/ลบ (16-bit Adder/Subtractor Circuit) โดยมีสัญญาณ ADD และ WC (With Carry) เพื่อเลือกหน้าที่ของวงจรดังในตารางที่ 3.1 ส่วนวงจรที่ดำเนินการทางตรรกะคือ AND OR และ XOR



รูปที่ 3.3 โครงสร้างของหน่วยคำนวณและตรรกะ

วงจร MOV/SWAP ให้ค่าเท่ากับ Rs ในกรณีที่สัญญาณ SWP มีค่าเป็น '0' และให้ค่าสลับไบต์สูงกับไบต์ต่ำของ Rs ในกรณีที่สัญญาณ SWP มีค่าเป็น '1'

เอาต์พุตของวงจรทั้งหมดถูกส่งเข้าวงจร 8-to-1 Multiplexer เพื่อเลือกค่าออกเป็นผลลัพธ์ของการประมวลผลของหน่วยคำนวณและตรรกะ คือสัญญาณ ACC และสัญญาณดังกล่าวส่งเข้าวงจรสร้างสถานะเพื่อตรวจสอบเงื่อนไขของผลลัพธ์แล้วให้ค่าสถานะออกทางสัญญาณ N_FLG ซึ่งจะถูกเก็บเข้าไว้ในรีจิสเตอร์สถานะในรอบสัญญาณนาฬิกาถัดไป สัญญาณ OP_SEL ที่ได้จากวงจรถอดรหัสคำสั่งทำหน้าที่เลือกผลลัพธ์

ตารางที่ 3.1 การเลือกหน้าที่ของวงจรบวก/ลบ

สัญญาณ ADD	สัญญาณ WC	ดำเนินการตามคำสั่ง
1	0	ADD
1	1	ADC
0	0	SUB
0	1	SBC

ตารางที่ 3.2 การเลือกผลลัพธ์ของการประมวลผลของหน่วยคำนวณและตรรกะ

สัญญาณ OP_SEL	สัญญาณ ACC
000	MOV/SWAP
001	AND
010	OR
011	XOR
100	Adder/Subtractor
101	0:RS[15..1] (LSR)
110	RS[14..0]:CIN (RL)
111	CIN:RS[15..1] (RR)

หน่วยคำนวณและตรรกะ สามารถใช้เพื่อคำนวณข้อมูลแบบคิดเครื่องหมาย (Signed number) และไม่คิดเครื่องหมาย (Unsigned number) โดยอาศัยสถานะของรีจิสเตอร์สถานะ และค่าในรีจิสเตอร์สถานะก็ยังใช้ตรวจสอบเงื่อนไขในการกระโดดของโปรแกรม (ดูรายละเอียดในตารางที่ 3.8)

หน่วยคำนวณและตรรกะและชุดคำสั่งกลุ่มคำนวณและตรรกะถูกออกแบบให้มีขนาดเล็ก โดยลดคำสั่งที่สามารถใช้คำสั่งอื่นทดแทนได้เช่น

- LSL(Logical Shift Left) ใช้คำสั่ง ADD แทนโดยบวกตัวถูกดำเนินการที่ต้องการเลื่อนบิตด้วยตัวมันเอง

  LSL R1 ≡ ADD R1,R1

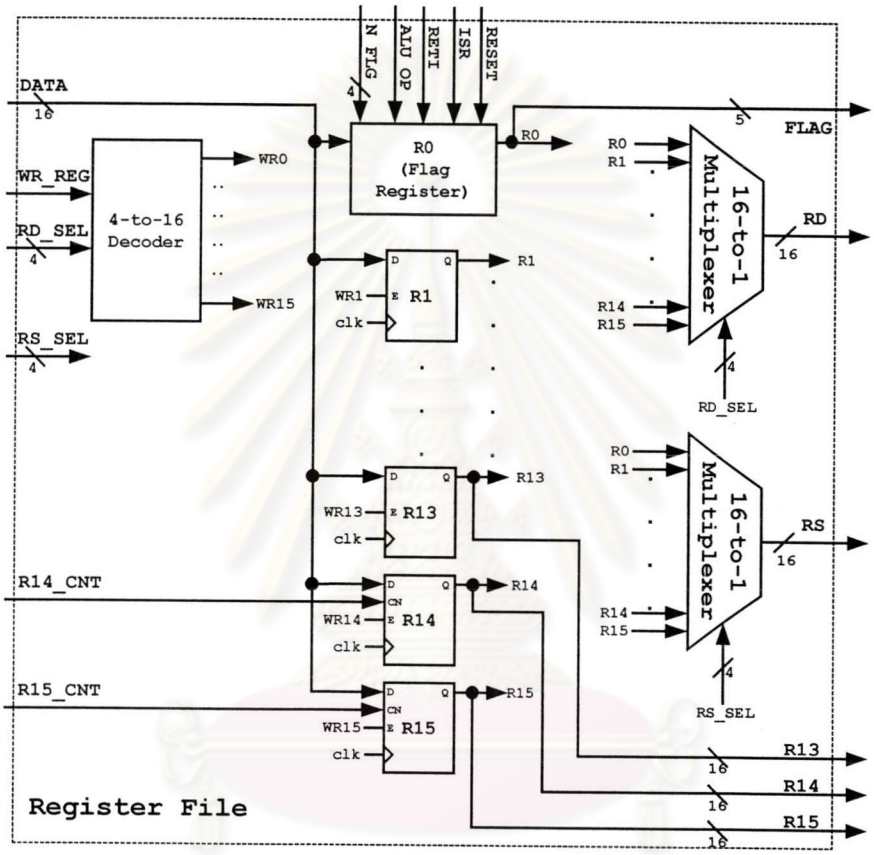
- CLR(Clear) ใช้คำสั่ง XOR แทนโดยดำเนินการ XOR รีจิสเตอร์ที่ต้องการเคลียร์ด้วยตัวมันเอง

  CLR R1 ≡ XOR R1,R1

ลักษณะการออกแบบชุดคำสั่งเช่นนี้ยังคงทำให้ชุดคำสั่งมีความสมบูรณ์ และความไม่ซ้ำซ้อนของคำสั่ง (Orthogonality) มีค่าสูง

3.2.2 แฟ้มรีจิสเตอร์

แฟ้มรีจิสเตอร์ประกอบด้วยรีจิสเตอร์ใช้งานทั่วไปขนาด 16 บิต จำนวน 15 ตัวคือ R1 – R15 สำหรับเก็บข้อมูลที่ใช้ระหว่างการประมวลผล และรีจิสเตอร์สถานะ R0 สำหรับเก็บค่าสถานะการคำนวณและการตอบรับการขัดจังหวะ



รูปที่ 3.4 โครงสร้างของแฟ้มรีจิสเตอร์

จากรูปที่ 3.4 แสดงโครงสร้างของแฟ้มรีจิสเตอร์ให้เห็นว่าสามารถเขียนข้อมูลลงรีจิสเตอร์แต่ละตัวได้โดยการควบคุมของสัญญาณต่อไปนี้

- DATA เป็นข้อมูลขนาด 16 บิตที่ต้องการเขียนลงรีจิสเตอร์
- RD_SEL เป็นสัญญาณควบคุมขนาด 4 บิตสำหรับเลือกรีจิสเตอร์ที่ต้องการเขียนค่า

- WR_REG เป็นสัญญาณเปิดทางให้วงจรถอดรหัสเลือกรีจิสเตอร์ที่ต้องการเขียนค่าลงไป โดยถอดรหัสจากสัญญาณ RD_SEL

ส่วนการอ่านค่าจากแฟ้มรีจิสเตอร์เพื่อนำไปประมวลผลโดยหน่วยคำนวณและตรรกะ หรือเพื่อนำไปเก็บไว้ในหน่วยความจำ สามารถอ่านค่าจากรีจิสเตอร์ได้ครั้งละ 2 ตัว โดยที่ค่าจากรีจิสเตอร์แต่ละตัวจะถูกเลือกว่าวงจรมัลติเพลกเซอร์สองตัวออกทางพอร์ต RD และ RS โดยมีสัญญาณ RD_SEL และ RS_SEL เป็นตัวเลือกตามลำดับ จากโครงสร้างดังกล่าวรีจิสเตอร์ตัวที่ถูกเลือกผ่านพอร์ต RD จะเป็นตัวเดียวกับตัวที่ถูกเลือกเพื่อเขียนค่าลงไป

นอกจากนี้ค่าในรีจิสเตอร์ R13, R14 และ R15 สามารถอ่านผ่านพอร์ต R13 – R15 ได้โดยตรง เพื่อใช้เป็นค่าในรีจิสเตอร์

- R13 เป็นตัวระบุตำแหน่งในหน่วยความจำสำหรับคำสั่งที่มีแอสเดรสซิ่งโหมดแบบตัวชี้ (Indexed mode)
- R14 และ R15 เป็นตัวระบุตำแหน่งในหน่วยความจำสำหรับคำสั่งที่มีแอสเดรสซิ่งโหมดแบบรีจิสเตอร์โดยอ้อม (Register Indirect mode)

รีจิสเตอร์ R14 และ R15 สามารถเพิ่มค่าโดยอัตโนมัติหลังจากการใช้เป็นตัวกำหนดตำแหน่งข้อมูลในหน่วยความจำแล้ว โดยสัญญาณ R14_CNT และ R15_CNT เป็นสัญญาณเปิดทางให้รีจิสเตอร์ R14 และ R15 นับขึ้นตามลำดับ

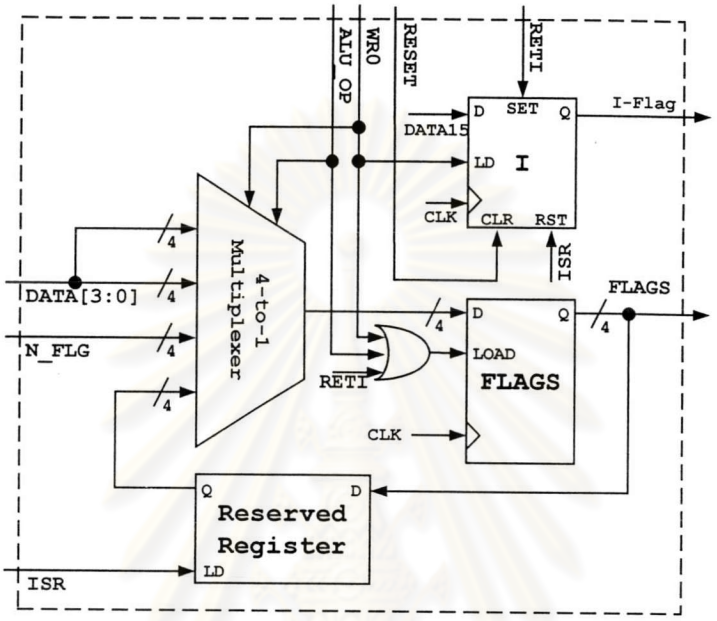
ส่วนรีจิสเตอร์ R0 เป็นรีจิสเตอร์ที่เก็บค่าสถานะการคำนวณของหน่วยคำนวณและตรรกะ รวมทั้งแฟลกควบคุมการขัดจังหวะการทำงาน มีสัญญาณควบคุมรีจิสเตอร์ R0 นอกจากสัญญาณ WR0 ที่ควบคุมการเขียนรีจิสเตอร์ R0 ดังเช่นรีจิสเตอร์ตัวอื่นแล้วยังมีสัญญาณควบคุมที่ต่างไปจากรีจิสเตอร์ตัวอื่น ๆ อีกซึ่งจะกล่าวถึงรายละเอียดในหัวข้อ 3.2.3

3.2.3 <u>รีจิสเตอร์สถานะ</u>

หน้าที่หลักของรีจิสเตอร์ R0 คือเก็บค่าสถานะจากการประมวลผลของหน่วยคำนวณและตรรกะ ซึ่งจะถูกเก็บไว้ในรีจิสเตอร์ R0 ในตำแหน่ง 4 บิตล่างสุด ตามรูปที่ 3.6 โดยสถานะที่หน่วยคำนวณและตรรกะทำการตรวจสอบมี 4 รูปแบบคือ

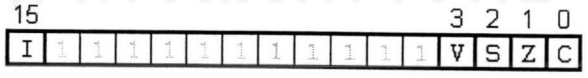
C: Carry Flag	แฟลกตัวทด
Z: Zero Flag	แฟลกศูนย์ มีสถานะเป็น '1' เมื่อผลลัพธ์เป็น 00H
S: Sign Flag	แฟลกเครื่องหมาย มีสถานะเดียวกับบิตสูงสุดของผลลัพธ์

V: Overflow Flag แฟลกเกินช่วง มีสถานะเป็น '1' เมื่อมีการดำเนินการคำสั่งประเภทคำนวณและผลลัพธ์มีค่าช่วงข้อมูล 2's complement 16 บิต



รูปที่ 3.5 โครงสร้างของรีจิสเตอร์สถานะ

สัญญาณ N_FLG เป็นค่าสถานะที่หน่วยคำนวณและตรรกะส่งเข้ามาเก็บไว้ในรีจิสเตอร์ R0 โดยมีสัญญาณ ALU_OP เป็นสัญญาณเปิดทางให้มีการเก็บค่า N_FLG เข้าไว้ในรีจิสเตอร์ R0 ซึ่งสัญญาณดังกล่าวเมื่อหน่วยประมวลผลกลางดำเนินการคำสั่งที่ใช้งานหน่วยคำนวณและตรรกะ นอกจากนี้รีจิสเตอร์ R0 สามารถเขียนค่าข้อมูลลงไปได้โดยใช้คำสั่งประเภทที่ระบุ R0 เป็นรีจิสเตอร์เป้าหมาย ในกรณีดังกล่าวสัญญาณ WR0 จะตื่นตัว ค่าจากสัญญาณ DATA จะถูกเลือกเข้าไปเขียนในรีจิสเตอร์ R0 แทนค่าจากสัญญาณ N_FLG (ดูตารางที่ 3.3)



รูปที่ 3.6 รูปแบบของรีจิสเตอร์สถานะ R0

บิตที่ 15 ของรีจิสเตอร์ R0 ใช้ความคุมการขัดจังหวะการทำงาน สัญญาณที่เกี่ยวข้องกับการขัดจังหวะการทำงานคือ

- ISR (Interrupt Service Routine) จะตื่นตัวเมื่อหน่วยประมวลผลกลางตอบรับการร้องขอขัดจังหวะการทำงาน
- RETI (Return form Interrupt service Routine) ตื่นตัวเมื่อหน่วยประมวลผลกลางดำเนินการคำสั่ง RETI

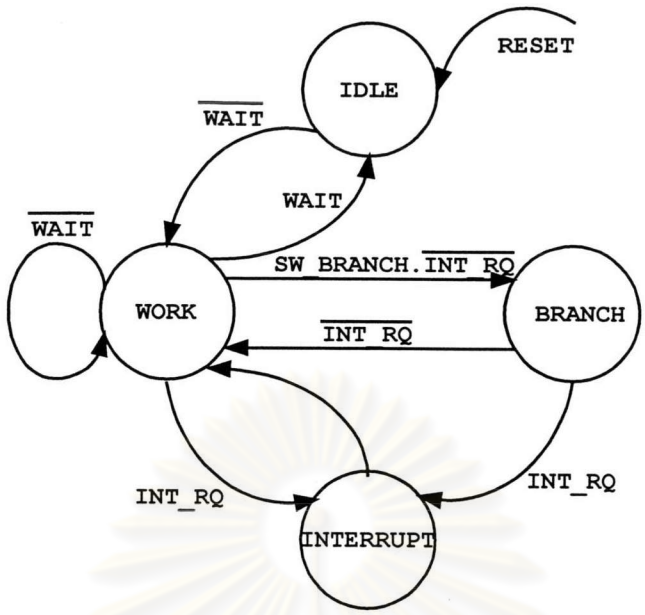
เมื่อสัญญาณ ISR ตื่นตัว แฟลก I จะถูกรีเซตโดยอัตโนมัติเป็นผลให้หน่วยประมวลผลกลางไม่สามารถตอบรับการร้องขอการขัดจังหวะได้อีก และค่าในรีจิสเตอร์ R0 4 บิตล่างสุดซึ่งเป็นค่าสถานะจากการคำนวณจะถูกเก็บไว้ในรีจิสเตอร์สำรอง (Reserved register) และเมื่อหน่วยประมวลผลกลางทำงานตามชุดคำสั่งประจำสำหรับการขัดจังหวะเสร็จพบคำสั่ง RETI ทำให้สัญญาณ RETI ตื่นตัว แฟลก I จะถูกเซตเป็น '1' โดยอัตโนมัติเพื่อให้หน่วยประมวลผลกลางสามารถตอบรับการร้องขอขัดจังหวะในครั้งต่อไปได้ และค่าสถานะที่ถูกเก็บไว้ในรีจิสเตอร์สำรองจะถูกเขียนกลับสู่รีจิสเตอร์ R0 เพื่อให้ค่าสถานะแฟลกเหมือนเดิมก่อนกระโดดกลับไปทำงานตามคำสั่งก่อนมีการขัดจังหวะ

ตารางที่ 3.3 ลำดับความสำคัญการเปลี่ยนแปลงค่าในรีจิสเตอร์ R0

ชนิดการเปลี่ยนแปลง	คำสั่ง	การดำเนินการ
การตอบรับการขัดจังหวะ	-	I <= 0, R0[3:0] => Reserved Reg.
การเขียนค่าลง R0 โดยตรง	MOV, LD โดยระบุ R0 เป็นตัวเก็บค่า	R0 <= DATA
ดำเนินการคำนวณและตรรกะ	MOV และคำสั่งการคำนวณและตรรกะ โดยระบุ R1-R15 เป็นตัวเก็บค่า	R0[3:0] <= N_FLG
กลับจากชุดคำสั่งประจำสำหรับการขัดจังหวะ	RETI	I <= '1' R0[3:0] <= Reserved Reg.

3.2.4 หน่วยควบคุม (Control Unit)

หน่วยควบคุมทำหน้าที่ควบคุมจังหวะการทำงานของหน่วยประมวลผลกลาง เช่น การกระโดดของโปรแกรม การขอขัดจังหวะ และการรีเซตระบบ เป็นต้น มีแผนผังสถานะการทำงานดังรูปที่ 3.7



รูปที่ 3.7 แผนผังสถานะการทำงาน (Operational State Diagram) ของหน่วยควบคุม

สัญญาณที่ควบคุมการเปลี่ยนสถานะของหน่วยควบคุม มีทั้งหมด 4 สัญญาณคือ

- RESET เป็นขาสัญญาณภายนอก มีลำดับความสำคัญสูงสุด เมื่อสัญญาณ RESET ตื่นตัว หน่วยควบคุมจะเข้าสู่สถานะ IDLE ตัวนับโปรแกรมจะถูกรีเซต เป็นการทำให้ทั้งชิปเริ่มต้นทำงานใหม่
- WAIT เป็นขาสัญญาณภายนอก มีลำดับความสำคัญรองลงมาจากสัญญาณ RESET เมื่อสัญญาณนี้ตื่นตัวหน่วยควบคุมจะเข้าสู่สถานะ IDLE จนกว่าสัญญาณ WAIT จะหายไป
- SW_BRANCH เป็นสัญญาณภายในชิปจะตื่นตัวเมื่อเงื่อนไขในการกระโดดของโปรแกรมถูกต้อง และหน่วยประมวลผลกลางต้องกระโดดไปทำงาน ณ ตำแหน่งที่ระบุไว้ในคำสั่ง
- INT_RQ เป็นสัญญาณภายในชิปจะตื่นตัวเมื่อ อุปกรณ์บริวารตัวใดตัวหนึ่งร้องขอขัดจังหวะการทำงาน

สถานะ IDLE เป็นสถานะที่หน่วยประมวลผลกลางไม่ดำเนินการคำสั่งใด ๆ ทั้งสิ้น รวมทั้งไม่มีการตอบสนองการร้องขอขัดจังหวะของอุปกรณ์บริวารต่าง ๆ

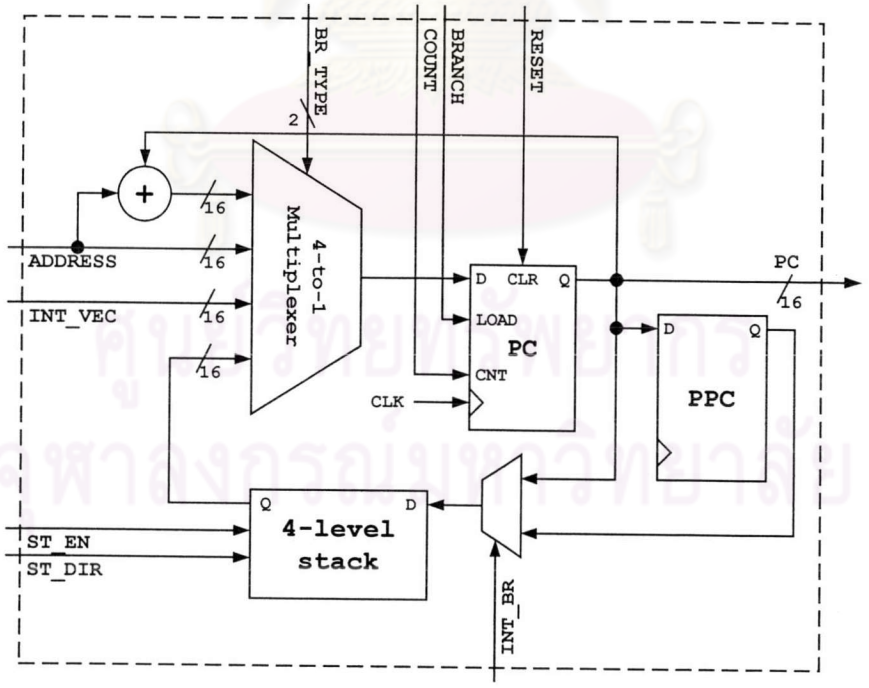
สถานะ WORK เป็นสถานะการทำงานปกติ เมื่อหน่วยประมวลกลางอยู่ในสถานะนี้ คำสั่งที่อยู่ในขั้นตอนดำเนินการจะถูกดำเนินการตามปกติ

สถานะ BRANCH เกิดจากการดำเนินการคำสั่งกระโดดโปรแกรม หน่วยควบคุมจะอยู่ในสถานะนี้เพียง 1 รอบสัญญาณนาฬิกาต่อการกระโดด 1 ครั้ง ในจังหวะที่หน่วยควบคุมอยู่ในสถานะ BRANCH คำสั่งที่อยู่ถัดจากคำสั่งกระโดดจะอยู่ในขั้นตอนดำเนินการพอดี แต่คำสั่งนั้นจะไม่ถูกดำเนินการจึงมีผลเหมือนคำสั่ง NOP

สถานะ INTERRUPT เกิดจากการที่หน่วยประมวลผลกลางตอบรับการขัดจังหวะ สถานะนี้มีลักษณะคล้ายสถานะ BRANCH ซึ่งเป็นการกระโดดของโปรแกรมเหมือนกัน แต่ตำแหน่งของโปรแกรมที่กระโดดไปทำงาน จะต้องสอดคล้องกับหมายเลขการขัดจังหวะ หน่วยประมวลผลกลางจะเข้าสู่สถานะนี้ ในรอบสัญญาณนาฬิกาถัดจากการเกิดสัญญาณร้องขอขัดจังหวะ ไม่ว่าสถานะปัจจุบันจะเป็น WORK หรือ BRANCH ก็ตาม ทำให้การตอบสนองต่อการร้องขอขัดจังหวะเท่ากันทุกครั้ง

คำสั่งที่อยู่ในขั้นตอนดำเนินการจะถูกดำเนินการก็ต่อเมื่อหน่วยควบคุมอยู่ในสถานะ WORK เท่านั้น ส่วนวงจรในขั้นตอนเฟทช์คำสั่งจะดำเนินการทุกรอบสัญญาณนาฬิกาไม่ขึ้นกับสถานะของหน่วยควบคุม

3.2.5 ตัวนับโปรแกรมและสแตก



รูปที่ 3.8 โครงสร้างของตัวนับโปรแกรมและสแตก

ในสถาวะการทำงานปกติตัวนับโปรแกรมขนาด 16 บิตจะนับขึ้นทุกรอบสัญญาณนาฬิกา การเฟทช์คำสั่งเป็นไปอย่างต่อเนื่อง ในกรณีการกระโดดของโปรแกรมตัวนับโปรแกรมถูกเปลี่ยนแปลงค่าอย่างไม่ต่อเนื่องภายใต้การควบคุมของหน่วยควบคุม

สัญญาณ COUNT ตื่นตัวในสถานะที่หน่วยประมวลผลกลางทำงานปกติ สัญญาณ BRANCH ตื่นตัวในกรณีที่มีการกระโดดของโปรแกรม โดยโหลดค่าตัวนับโปรแกรมใหม่โดยใช้สัญญาณ BR_TYPE เป็นตัวเลือกค่า

ตารางที่ 3.4 ค่าตัวนับโปรแกรมในการกระโดดของโปรแกรมแบบต่าง ๆ

สัญญาณ BR_TYPE	ค่าตัวนับโปรแกรม	หมายเหตุ
00	PC <= STACK	คำสั่ง RET หรือ RETI
01	PC <= INT_VEC	ค่า Interrupt vector ตอบรับการขัดจังหวะ
10	PC <= ADDRESS	คำสั่งกระโดดแบบโดยตรง
11	PC <= PC + ADDRESS	คำสั่งกระโดดแบบสัมพัทธ์

การควบคุมวงจรหน่วยความจำสแตกใช้สัญญาณ ST_EN เพื่อเปิดทางให้มีการเลื่อนข้อมูลในสแตก และสัญญาณ ST_DIR ควบคุมทิศทางการเลื่อนข้อมูล หน่วยความจำสแตกประกอบด้วยรีจิสเตอร์ขนาด 16 บิต 4 ชุด สำหรับเก็บค่าตัวนับโปรแกรมเมื่อดำเนินการคำสั่ง CALL และตอบรับขัดจังหวะการทำงาน และส่งค่าคือตัวนับโปรแกรมเมื่อดำเนินการคำสั่ง RET, RETI

ในกรณีที่มีการตอบรับการขัดจังหวะขณะที่ดำเนินการคำสั่งกระโดดโปรแกรมอยู่นั้น สัญญาณ INT_BR จะตื่นตัวทำให้ค่าที่ถูกเขียนลงสแตกคือค่าจากรีจิสเตอร์ PPC (ดูรูปที่ 3.8) ซึ่งจะเก็บค่าตำแหน่งก่อนหน้าตำแหน่งตัวนับโปรแกรมปัจจุบัน นั่นก็คือตำแหน่งของคำสั่งกระโดดโปรแกรมนั่นเอง คำสั่งกระโดดโปรแกรมจะถูกดำเนินการหลังจากทำงานชุดคำสั่งประจำสำหรับการขัดจังหวะเสร็จเรียบร้อยแล้ว ซึ่งเป็นไปตามหลักการของ Q-Chip ที่ให้ความสำคัญแก่การขัดจังหวะมากกว่าการกระโดดของโปรแกรม

3.3 การขัดจังหวะ

หน่วยประมวลผลกลางสามารถตอบรับการขอขัดจังหวะได้ทั้งหมด 7 แหล่งคือสัญญาณ IRQ1 ถึง IRQ7 หากเกิดการร้องขอพร้อมกันสัญญาณที่มีความสำคัญสูงกว่าจะได้การตอบสนอง

ก่อน (IRQ1 มีลำดับความสำคัญสูงสุด) ลำดับความสำคัญและตำแหน่งชุดคำสั่งประจำสำหรับการขัดจังหวะ (Interrupt service routine) แสดงในตารางที่ 3.5

เมื่อมีสัญญาณร้องขอขัดจังหวะเข้ามาขณะที่ I Flag มีสถานะเป็น '1' หน่วยประมวลผลกลางจะตอบรับการร้องขอนั้น และกระโดดไปทำงาน ณ ตำแหน่งที่สอดคล้องกับสัญญาณร้องขอขัดจังหวะ ส่วน I Flag จะถูกรีเซตให้มีสถานะเป็น '0' โดยอัตโนมัติ และข้อมูลในรีจิสเตอร์สถานะ 4 บิตล่างจะถูกเก็บไว้ในรีจิสเตอร์สำรอง ซึ่งทำให้ในขณะที่ดำเนินการในชุดคำสั่งประจำสำหรับการขัดจังหวะอยู่นั้น หน่วยประมวลผลกลางจะไม่ตอบรับการขอขัดจังหวะสัญญาณอื่น ๆ และเมื่อจบชุดคำสั่งประจำสำหรับการขัดจังหวะด้วยคำสั่ง RETI หน่วยประมวลผลกลางจะกระโดดกลับมาทำงาน ณ ตำแหน่งเดิมและ I Flag จะถูกเซตให้มีสถานะเป็น '1' โดยอัตโนมัติ ข้อมูลในรีจิสเตอร์สำรองจะถูกเขียนกลับไปในรีจิสเตอร์สถานะ เป็นการเตรียมพร้อมสำหรับการตอบรับการร้องขอขัดจังหวะครั้งต่อไป

ตารางที่ 3.5 หมายเลข ลำดับความสำคัญ และตำแหน่งชุดคำสั่งประจำสำหรับการขัดจังหวะ

หมาเลข / แหล่งที่มา	ลำดับความสำคัญ	ตำแหน่งชุดคำสั่งประจำสำหรับการขัดจังหวะ
IRQ1 / PA4	1	0002H
IRQ2 / PA5	2	0004H
IRQ3 / TC0 Over flow	3	0006H
IRQ4 / TC1 Over flow	4	0008H
IRQ5 / TC1 Output compare	5	000AH
IRQ6 / TC1 Input capture	6	000CH
IRQ7 / ADC	7	000EH

จากตารางที่ 3.5 จะสังเกตได้ว่าตำแหน่งของชุดคำสั่งประจำสำหรับการขัดจังหวะแต่ละตัวห่างกันเพียง 2 ตำแหน่งเท่านั้น ดังนั้นสองคำสั่งแรกควรเป็นคำสั่งในกลุ่มกระโดด เพื่อกระโดดไปยังตำแหน่งที่เก็บคำสั่งในการทำงานสำหรับการขัดจังหวะนั้น ๆ

ขณะที่เกิดสัญญาณร้องขอขัดจังหวะ คำสั่งที่อยู่ในขั้นตอนดำเนินจะถูกดำเนินการให้เสร็จเรียบร้อย ส่วนคำสั่งที่อยู่ในขั้นตอนเฟทช์จะถูกทิ้งไป และจะถูกเฟทช์เข้ามาทำงานใหม่หลังจากคำสั่ง RETI ดังนั้นคำสั่งแรกของชุดคำสั่งประจำสำหรับการขัดจังหวะจะดำเนินการภายใน 2 รอบสัญญาณนาฬิกาหลัง

เนื่องจากคำสั่งประเภทกระโดดต้องใช้เวลาในการดำเนินการ 2 รอบสัญญาณนาฬิกา หากเกิดสัญญาณร้องขอขัดจังหวะในรอบสัญญาณนาฬิกาที่ 2 ของการดำเนินการคำสั่งกระโดด คำสั่งนั้นจะดำเนินการจนเสร็จเรียบร้อยก่อนเช่นเดียวกับคำสั่งอื่น ๆ แต่ถ้าหากสัญญาณร้องขอการขัดจังหวะเกิดขึ้นในรอบสัญญาณนาฬิกาแรกของการดำเนินการคำสั่งกระโดด คำสั่งนั้นจะไม่ถูกดำเนินการ แต่หน่วยประมวลผลกลางจะดำเนินการคำสั่งกระโดดนั้นหลังจากคำสั่ง RETI ซึ่งลักษณะเช่นนี้ หน่วยประมวลผลกลางจะใช้เวลาในการตอบรับการร้องขอขัดจังหวะเท่ากันทุกกรณี จากรูปที่ 3.7 จะเห็นได้ว่า เมื่อเกิดสัญญาณ INT_RQ หน่วยควบคุมจะเข้าสู่สถานะ INTERRUPT ทำให้ค่าตัวนับโปรแกรมถูกโหลดด้วยค่าตำแหน่งชุดคำสั่งประจำสำหรับการขัดจังหวะ และจะดำเนินการในรอบสัญญาณนาฬิกาถัดไป

3.4 ชุดคำสั่ง

สถาปัตยกรรมชุดคำสั่งของ Q-Chip มีลักษณะเป็นแบบโหลดสตอร์อินสตรักชัน (Load and Store Instructions)

3.4.1 คำสั่ง

คำสั่งสำหรับดำเนินการมีทั้งหมดแบ่งออกเป็น 3 กลุ่มคือ กลุ่มคำนวณและตรรกะ (Arithmetic and Logic Instructions) กลุ่มโอนย้ายข้อมูล (Data Transfer Instructions) และกลุ่มควบคุม (Machine Control Instructions) แสดงในตารางที่ 3.7

ตารางที่ 3.6 สัญลักษณ์ที่ใช้ในตารางที่ 3.7

Symbols	Description	comments
Rd	Destination register in the register file	
Rs	Source register in the register file	
Im	Constant data	16 bit
K	Constant address	16 bit
disp	Displacement for indexed addressing	8 bit
Rel	Relative address	8 -bit 2's complement
X	Register R14 or R15	
cond	Branch condition	see table 3.8
I	Interrupt flag	
PC	Program counter	
STACK	Stack memory	

ตารางที่ 3.2 รายละเอียดชุดคำสั่งโดยสังเขป

Mnemonics	Operands	Description	Operation	Flags V	S	Z	C	Cycle
Arithmatic and Logic Instructions								
ADC	Rd,Rs	Add with Carry	Rd <- Rd + Rs + C	!	!	!	!	1
ADC	R1,Im	Add Immediate with Carry	Rd <- Rd + Im + C	!	!	!	!	1
ADD	Rd,Rs	Add without Carry	Rd <- Rd + Rs	!	!	!	!	1
ADD	R1,Im	Add Immediate	Rd <- Rd + Im	!	!	!	!	1
AND	Rd,Rs	Logical AND	Rd <- Rd AND Rs		!	!		1
AND	R1,Im	Logical AND with Immediate	Rd <- Rd AND Im		!	!		1
LSR	Rd,Rs	Logical Shift Right	C <- Rs[0], Rd[n] <- Rs[n+1], Rd[15] <- 0	!	!	!	!	1
OR	Rd,Rs	Logical OR	Rd <- Rd OR Rs		!	!		1
OR	R1,Im	Logical OR with Immediate	Rd <- Rd OR Im		!	!		1
RL	Rd,Rs	Rotate Left Through Carry	C <- Rs[15], Rd[n] <- Rs[n-1], Rd[0] <- C	!	!	!	!	1
RR	Rd,Rs	Rotate Right Through Carry	C <- Rs[0], Rd[n] <- Rs[n+1], Rd[15] <- C	!	!	!	!	1
SBC	Rd,Rs	Subtract with Carry	Rd <- Rd - Rs - C	!	!	!	!	1
SBC	R1,Im	Subtract Immediate with Carry	Rd <- Rd - Im - C	!	!	!	!	1
SUB	Rd,Rs	Subtract without Carry	Rd <- Rd - Rs	!	!	!	!	1
SUB	R1,Im	Subtract Immediate	Rd <- Rd - Im	!	!	!	!	1
SWP	Rd,Rs	Swap Bytes	Rd[15:8] <- Rs[7:0], Rd[7:0] <- Rs[15:8]		!	!		1
XOR	Rd,Rs	Exclusive OR	Rd <- Rd XOR Rs		!	!		1
XOR	R1,Im	Exclusive OR with Immediate	Rd <- Rd XOR Im		!	!		1
Branch Instructions								
JMP	K	Direct Jump	PC <- K					2
CALL	K	Direct Call Subroutine	STACK <- PC, PC <- K					2
RET		Subroutine Return	PC <- STACK					2
RETI		Interrupt Return	PC <- STACK, I <- 1					2
RET	cond	Conditional Subroutine Return	if condition then PC <- STACK else NOP					2/1*
RETI	cond	Conditional Interrupt Return	if condition then PC <- STACK, I <- 1 else NOP					2/1*

ตารางที่ 3.7(ต่อ) รายละเอียดชุดคำสั่งโดยสังเขป

Mnemonics	Operands	Description	Operation	Flags				Cycle
				V	S	Z	C	
JMP	Rel	Relative Jump	PC <- PC + Rel					2
CALL	Rel	Relative Call Subroutine	STACK <- PC, PC <- PC + Rel					2
JMP	Rel,cond	Conditional Relative-Jump	if condition then PC <- PC + Rel else NOP					2/1*
CALL	Rel,cond	Conditional Relative-Call-Subroutine	if condition then STACK <- PC, PC <- PC + Rel else NOP					2/1*
NOP		No Operation	NOP					1
Data Transfer Instructions								
MOV	Rd,Im	Load Immediate	Rd <- Im		!	!		1
MOV	Rd,Rs	Copy Register	Rd <- Rs		!	!		1
LD	R1,M(K)	Load Direct from Data Space	R1 <- M(K)					1
ST	R1,M(K)	Store Direct to Data Space	R1 -> M(K)					1
LD	Rd,M(R13,disp)	Load Indexed with Displacement	Rd <- M(R13+disp)					1
ST	Rd,M(R13,disp)	Store Indexed with Displacement	Rd -> M(R13+disp)					1
LD	Rd,M(X)	Load Register-Indirect	R1 <- M(X)					1
ST	Rd,M(X)	Store Register-Indirect	R1 -> M(X)					1
LD	Rd,M(X+)	Load Register-Indirect with Post-Increment	R1 <- M(X), X <- X + 1					1
ST	Rd,M(X+)	Store Register-Indirect with Post-Increment	R1 -> M(X), X <- X + 1					1

* ถ้าเงื่อนไขเป็นจริง ใช้เวลาในการกระโดด2 รอบสัญญาณนาฬิกา หากเงื่อนไขไม่เป็นจริงใช้เวลา1 รอบสัญญาณนาฬิกา

! แฟลกมีการเปลี่ยนแปลงสถานะ ตามผลลัพธ์ที่ได้จากหน่วยคำนวณและตรรกะ

ตารางที่ 3.8 เงื่อนไขที่ใช้ในการกระโดดของโปรแกรม

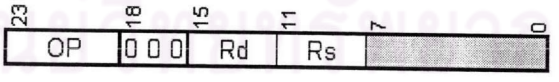
Mnemonic	Boolean	comment
GT	Z(N^V)=0	Signed
GE	(N^V)=0	Signed
EQ, ZE	Z=1	Simple
LE	Z+(N^V)=1	Signed
LT	(N^V)=1	Signed
POS	N = 0	Simple
NEG	N = 1	Simple
UGT	C+Z=0	Unsigned
UGE, NC	C=0	Unsigned
NEQ, NZ	Z=0	Simple
ULE	C+Z=1	Unsigned
ULT, CY	C=1	Unsigned
VF	V=1	Simple
NV	V=0	Simple

3.4.2 <u>รหัสดำเนินการและแอสเดรสซิ่งโหมด</u>

คำสั่งจะถูกแปลเป็นรหัสดำเนินการที่มีความยาว 24 บิต โดยแบ่งออกเป็น 3 ฟิลด์ คือ ฟิลด์คำสั่ง (Instruction field) 5 บิต ฟิลด์แอสเดรสซิ่งโหมด (Addressing mode field) 3 บิต และฟิลด์ตัวถูกดำเนินการ (Operand field) 16 บิต แอสเดรสซิ่งโหมดมีทั้งหมด 7 รูปแบบคือ

1. แอสเดรสซิ่งโหมดแบบรีจิสเตอร์ (Register Addressing Mode)

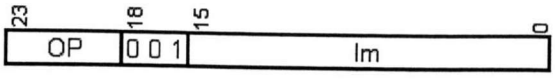
แอสเดรสซิ่งโหมดข้อมูลแบบรีจิสเตอร์ใช้ในคำสั่งที่ใช้งานหน่วยคำนวณและตรรกะ โดยข้อมูลที่อ้างถึง เป็นข้อมูลในแฟ้มรีจิสเตอร์ ที่กำหนดโดยฟิลด์ Rd และ Rs การดำเนินการคำสั่งดังกล่าว จะมีผลต่อสถานะในรีจิสเตอร์สถานะ R0



รูปที่ 3.9 รูปแบบรหัสดำเนินการที่มีแอสเดรสซิ่งโหมดแบบรีจิสเตอร์

2. แอสเดรสซิ่งโหมดแบบใช้ทันที (Immediate Addressing Mode)

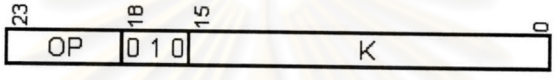
แอสเดรสซิ่งโหมดข้อมูลแบบใช้ทันทีใช้ในคำสั่งที่ใช้งานหน่วยคำนวณและตรรกะ โดยรีจิสเตอร์ที่ถูกนำมาดำเนินการกับค่าคงที่ และเก็บผลลัพธ์จากการคำนวณคือรีจิสเตอร์ R1 การดำเนินการจะมีผลต่อรีจิสเตอร์สถานะ R0 เช่นเดียวกับแอสเดรสซิ่งโหมดแบบรีจิสเตอร์



รูปที่ 3.10 รูปแบบรหัสดำเนินการที่มีแอสเดรสซิ่งโหมดแบบใช้ทันที

3. แอสเดรสซิ่งโหมดแบบโดยตรง(Direct Addressing Mode)

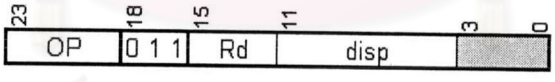
คำสั่งที่ใช้แอสเดรสซิ่งโหมดแบบโดยตรงเป็นคำสั่ง LD หรือ ST เป็นการโอนย้ายค่าระหว่างรีจิสเตอร์ R1 กับหน่วยความจำในตำแหน่งที่ระบุด้วย K ซึ่งสามารถอ้างถึงข้อมูลได้ 65536 ตำแหน่ง ในหน่วยความจำสำหรับข้อมูล และอุปกรณ์บริวาร



รูปที่ 3.11 รูปแบบรหัสดำเนินการที่มีแอสเดรสซิ่งโหมดแบบโดยตรง

4. แอสเดรสซิ่งโหมดแบบตัวชี้ (Indexed Addressing Mode)

คำสั่งที่ใช้แอสเดรสซิ่งโหมดแบบโดยตรงเป็นคำสั่ง LD หรือ ST เป็นการโอนย้ายค่าระหว่างรีจิสเตอร์ในแฟ้มรีจิสเตอร์ กับหน่วยความจำในตำแหน่งที่ระบุด้วย ค่าในรีจิสเตอร์ R13 บวกกับข้อมูลขนาด 8 บิตในฟิลด์ disp

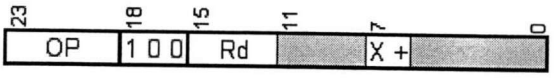


รูปที่ 3.12 รูปแบบรหัสดำเนินการที่มีแอสเดรสซิ่งโหมดแบบตัวชี้

5. แอสเดรสซิ่งโหมดแบบโดยอ้อมรีจิสเตอร์ (Register Indirect Addressing Mode)

คำสั่งที่ใช้แอสเดรสซิ่งโหมดแบบโดยตรงเป็นคำสั่ง LD หรือ ST เป็นการโอนย้ายค่าระหว่างรีจิสเตอร์ในแฟ้มรีจิสเตอร์ กับหน่วยความจำในตำแหน่งที่ระบุด้วย ค่าในรีจิสเตอร์ R14 เมื่อ X = 0

หรือ R15 เมื่อ X = 1 และ ค่าในรีจิสเตอร์ดังกล่าวจะมีค่าเพิ่มขึ้น 1 หลังการดำเนินการถ้าค่าในบิต + มีค่าเป็น 1



รูปที่ 3.13 รูปแบบรหัสดำเนินการที่มีแอสเดรสซิ่งโหมดแบบโดยอ้อมรีจิสเตอร์

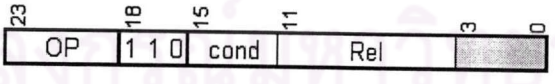
6. แอสเดรสซิ่งโหมดโปรแกรมแบบโดยตรง (Program Direct Addressing Mode)

คำสั่งที่ใช้แอสเดรสซิ่งโหมดโปรแกรมแบบโดยตรง เป็นคำสั่ง JMP หรือ CALL เพื่อกำหนดตำแหน่งโปรแกรมที่กระโดดไปซึ่งระบุโดยฟิลด์ K สามารถอ้างถึงได้ 65536 ตำแหน่ง

รูปที่ 3.14 รูปแบบรหัสดำเนินการที่มีแอสเดรสซิ่งโหมดโปรแกรมแบบโดยตรง

7. แอสเดรสซิ่งโหมดโปรแกรมแบบสัมพัทธ์ (Program Relative Addressing Mode)

คำสั่งที่ใช้แอสเดรสซิ่งโหมดโปรแกรมแบบโดยตรง เป็นคำสั่ง JMP หรือ CALL เพื่อกำหนดตำแหน่งโปรแกรมที่กระโดดไปซึ่งระบุจาก ค่า PC ปัจจุบัน บวกกับข้อมูล 8 บิตแบบ 2's complement ในฟิลด์ Rel โดยเป็นการกระโดดแบบมีเงื่อนไขในฟิลด์ cond ดูรายละเอียดเงื่อนไขในตารางที่ 3.8



รูปที่ 3.15 รูปแบบรหัสดำเนินการที่มีแอสเดรสซิ่งโหมดโปรแกรมาแบบสัมพัทธ์