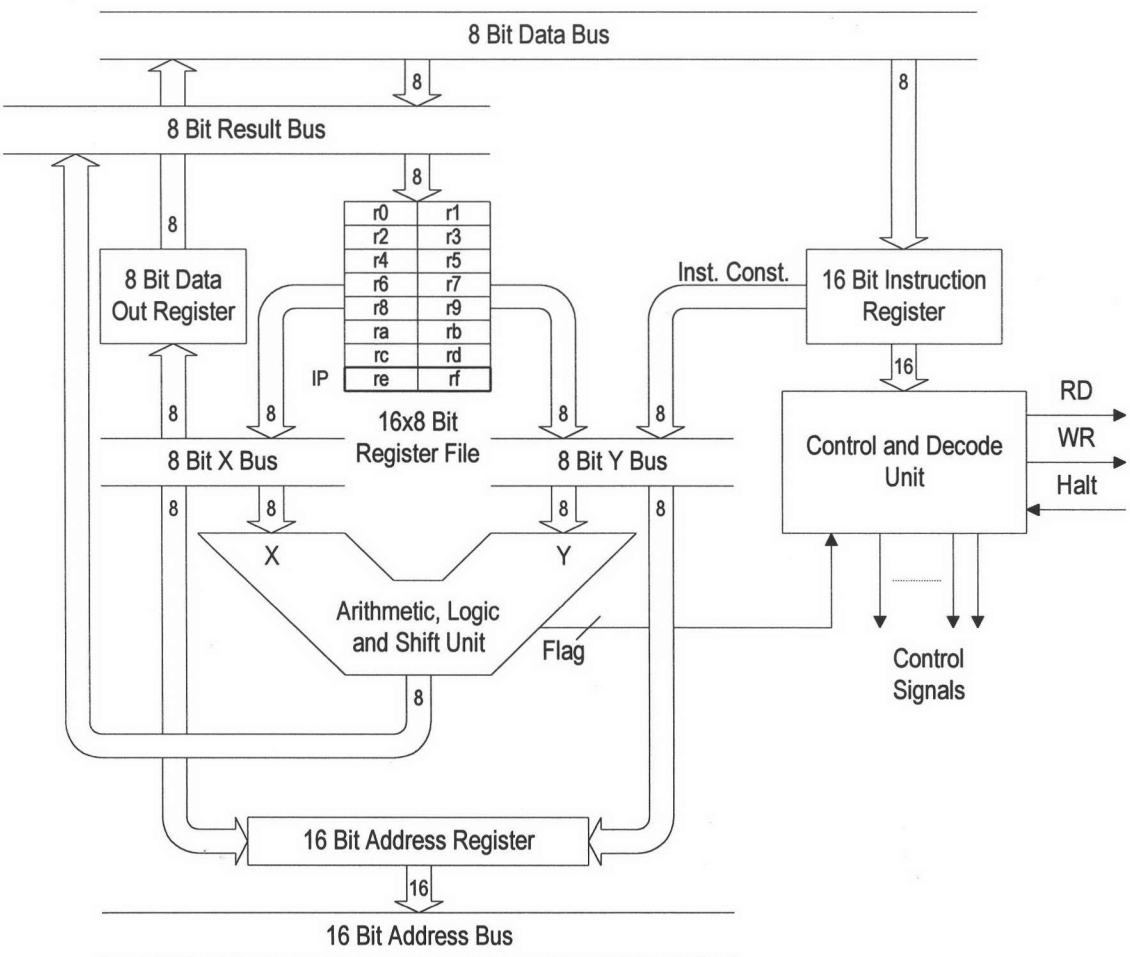
บทที่ 6

หน่วยประมวลผลกลาง

6.1 โครงสร้างภายในของหน่วยประมวลผลกลาง

หน่วยประมวลผลกลาง (Central Processing Unit) เป็นส่วนที่สำคัญที่สุดในตัวประมวลผลคำบรรยายภาพไทย-อังกฤษแบบซ่อนได้ (Thai-English Closed Caption Processor) ทำหน้าที่อ่านข้อมูลคำบรรยายภาพเข้ามาถอดรหัส และเขียนข้อมูลคำบรรยายภาพที่ถอดรหัสแล้วลงในหน่วยความจำเข้าถึงแบบสุ่ม (Random Access Memory) มีโครงสร้างดังรูปที่ 6.1 แต่ละส่วนมีหน้าที่ดังนี้



รูปที่ 6.1 โครงสร้างภายในของหน่วยประมวลผลกลาง

1. *รีจิสเตอร์คำสั่ง* (Instruction Register) เป็นรีจิสเตอร์ขนาด 16 บิตที่ใช้เก็บคำสั่ง (Instruction) ซึ่งอ่านมาจากหน่วยความจำทางบัสข้อมูล (Data Bus) ขนาด 8 บิต เพื่อป้อนให้แก่ หน่วยถอดรหัสและควบคุม (Control and Decode Unit) และมีสัญญาณออกขนาด 8 บิต เป็นค่าคงที่ของคำสั่ง (Instruction Constant) ส่งไปยังบัส Y ใช้เป็นตัวถูกดำเนินการ (Operand) ในกรณีที่คำสั่งนั้นเป็นการดำเนินการ (Operation) ระหว่างรีจิสเตอร์กับค่าคงที่

2. *หน่วยถอดรหัสและควบคุม* (Control and Decode Unit) เป็นส่วนที่ควบคุมการทำงานของส่วนอื่น ๆ ในหน่วยประมวลผลกลาง รับคำสั่ง (Instruction) ขนาด 16 บิต เข้ามาจากรีจิสเตอร์คำสั่ง (Instruction Register) เพื่อถอดรหัส และควบคุมส่วนอื่น ๆ ของหน่วยประมวลผลกลางให้ทำงานตามคำสั่งนั้น มีสัญญาณออก คือ อ่าน (RD) และเขียน (WR) ใช้ควบคุมการอ่าน และเขียนข้อมูลกับอุปกรณ์อื่น ที่อยู่นอกหน่วยประมวลผลกลาง สัญญาณหยุด (Halt) ที่ป้อนเข้ามาจะทำให้หน่วยถอดรหัสและควบคุมนี้หยุดทำงานชั่วขณะ ส่งผลให้หน่วยประมวลผลกลางหยุดทำงานด้วย

3. *แฟ้มรีจิสเตอร์* (Register File) เป็นที่เก็บรีจิสเตอร์ของหน่วยประมวลผลกลางที่นักโปรแกรม (Programmer) สามารถมองเห็น และเรียกใช้งานได้ โดยมีรีจิสเตอร์ทั้งหมด 16 ตัว มีชื่อตั้งแต่ r0 ถึง rf แต่ละตัวมีขนาด 8 บิต รีจิสเตอร์ re กับ rf จะเป็นรีจิสเตอร์พิเศษที่จะนำมาต่อกันเป็น 16 บิต เพื่อใช้เป็นตัวชี้คำสั่ง (Instruction Pointer) โดย re เป็น 8 บิตบน rf เป็น 8 บิตล่าง ทำให้เหลือรีจิสเตอร์ไว้ใช้งาน 14 ตัว แฟ้มรีจิสเตอร์นี้มีพอร์ตเขียน (Write Port) 1 พอร์ต เข้ามาจากบัสผลลัพธ์ (Result Bus) และมีพอร์ตอ่าน (Read Port) 2 พอร์ต ออกไปสู่บัส X และบัส Y

4. *หน่วยคำนวณ, ตรรก และเลื่อน* (Arithmetic, Logic and Shift Unit) เป็นส่วนที่ทำการคำนวณทางคณิตศาสตร์, การดำเนินการทางตรรก และการเลื่อนข้อมูล โดยจะดำเนินการกับข้อมูลขนาด 8 บิต 2 ตัว ที่ป้อนเข้ามาทางบัส X และบัส Y แล้วให้ผลการดำเนินการออกไปที่บัสผลลัพธ์ (Result Bus)

5. *รีจิสเตอร์ตำแหน่ง* (Address Register) เป็นรีจิสเตอร์ขนาด 16 บิต ที่ใช้ส่งตำแหน่งที่อยู่ (Address) ที่หน่วยประมวลผลกลางต้องการเข้าถึง ลงไปยังบัสตำแหน่งที่อยู่ (Address Bus) ตำแหน่งที่อยู่ที่ส่งออกไป เกิดจากการนำข้อมูลจากบัส X และบัส Y ที่มีขนาด 8 บิต มาต่อกันให้มีขนาด 16 บิต แล้วเก็บไว้ โดยข้อมูลจากบัส X จะเป็น 8 บิตบน ส่วนข้อมูลจากบัส Y เป็น 8 บิตล่าง

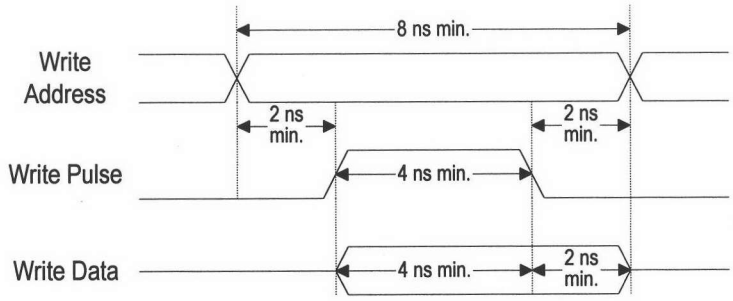
6. *รีจิสเตอร์ข้อมูลออก* (Data Out Register) เป็นรีจิสเตอร์ขนาด 8 บิต ที่ใช้ส่งข้อมูลที่หน่วยประมวลผลกลางเขียนออกไป ลงสู่บัสข้อมูล (Data Bus) โดยข้อมูลที่ส่งออกไป นำจากบัส X เข้ามาเก็บไว้

6.2 การออกแบบแฟ้มรีจิสเตอร์

แฟ้มรีจิสเตอร์ (Register File) ที่อยู่ในหน่วยประมวลผลกลาง มีรีจิสเตอร์อยู่ 16 ตัว มีชื่อตั้งแต่ r0 ถึง rf แต่ละตัวมีขนาด 8 บิต การเข้าถึงรีจิสเตอร์ในแฟ้มรีจิสเตอร์นี้ กระทำผ่านทางพอร์ต 3 พอร์ต คือ พอร์ตเขียน (Write Port) 1 พอร์ต และพอร์ตอ่าน (Read Port) 2 พอร์ต โดยสามารถใช้งานพอร์ตทั้ง 3 ได้พร้อม ๆ กันในแต่ละสัญญาณนาฬิกา

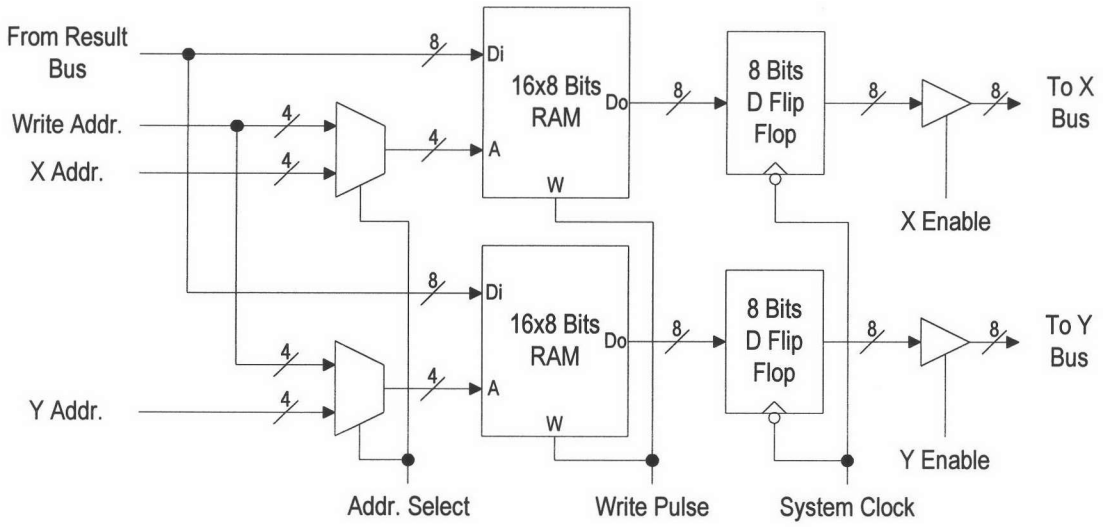
แฟ้มรีจิสเตอร์ที่ออกแบบขึ้นมานี้ ใช้หน่วยความจำขนาด 16x8 บิต (มี 16 ตำแหน่งที่อยู่ ตำแหน่งที่อยู่ละ 8 บิต แต่ละตำแหน่งที่อยู่แทน 1 รีจิสเตอร์) จำนวน 2 ตัว ในชิป FPGA เป็นส่วนประกอบหลัก เหตุที่ต้องใช้หน่วยความจำ 2 ตัว เนื่องจากการอ่านข้อมูลจากหน่วยความจำแต่ละตัวจะอ่านได้เพียงทีละ 1 ตำแหน่งที่อยู่เท่านั้น การใช้หน่วยความจำ 2 ตัว จะทำให้สามารถอ่านข้อมูลจากตำแหน่งที่อยู่ที่ต่างกัน 2 ที่ได้พร้อมกัน แต่มีสิ่งที่ต้องคำนึงในการใช้หน่วยความจำทั้งสองนี้เป็นแฟ้มรีจิสเตอร์ คือ

1. ข้อมูลที่เก็บในหน่วยความจำทั้งสองที่ตำแหน่งที่อยู่ (Address) เดียวกัน (รีจิสเตอร์ตัวเดียวกัน) จะต้องมีค่าตรงกัน ดังนั้นการเขียนข้อมูลลงหน่วยความจำทั้งสองแต่ละครั้ง จะต้องมีตำแหน่งที่อยู่เดียวกัน, ข้อมูลที่เขียนต้องเป็นตัวเดียวกัน และต้องเขียนพร้อมกัน
2. การเขียนข้อมูลลงหน่วยความจำในชิป ต้องระวังไม่ให้จังหวะเวลาของสัญญาณต่าง ๆ ผิดจากข้อกำหนดดังในรูปที่ 6.2



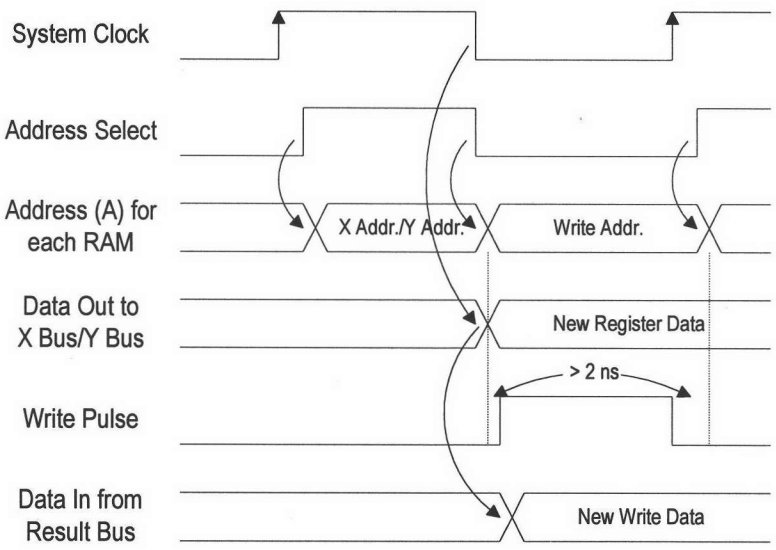
รูปที่ 6.2 ข้อกำหนดทางเวลาในการเขียนข้อมูลลงหน่วยความจำภายในชิป FPGA

จากที่กล่าวไปข้างต้น ทำให้โครงสร้างของแฟ้มรีจิสเตอร์เป็นดังรูปที่ 6.3 คือ ตำแหน่งที่อยู่ที่จะอ่านข้อมูลของหน่วยความจำทั้งสองต่างกัน (X Addr. กับ Y Addr.), ข้อมูลที่จะเขียนลงหน่วยความจำทั้งสองเป็นข้อมูลเดียวกัน (From Result Bus), ตำแหน่งที่อยู่ที่จะเขียนของหน่วยความจำแต่ละตัวเป็นที่เดียวกัน (Write Addr.) และสัญญาณเขียน (Write Pulse) เป็นสัญญาณเดียวกัน



รูปที่ 6.3 โครงสร้างของแฟ้มรีจิสเตอร์

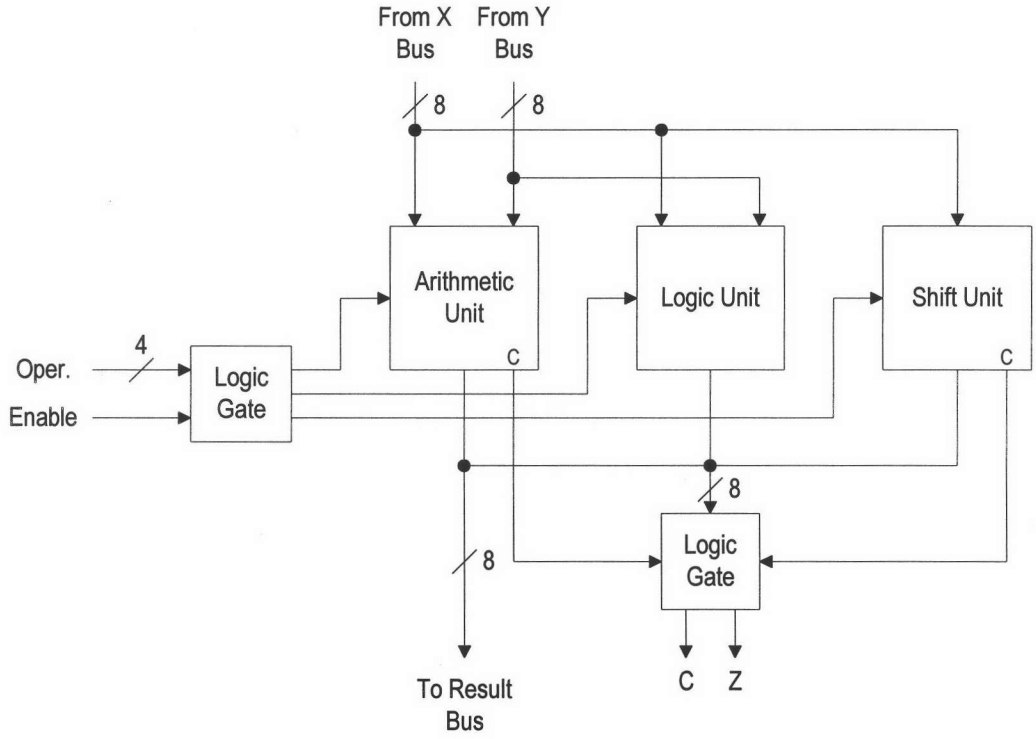
หน่วยความจำที่ใช้ทำเป็นแฟ้มรีจิสเตอร์นี้ จะมีจังหวะการทำงานแบ่งเป็น 2 ช่วง คือ ช่วงครึ่งคาบสัญญาณนาฬิกาแรกใช้อ่านข้อมูล และครึ่งคาบสัญญาณนาฬิกาหลังใช้เขียนข้อมูล โดยจังหวะของสัญญาณต่าง ๆ ที่สำคัญแสดงได้ดังรูปที่ 6.4



รูปที่ 6.4 รูปคลื่นสัญญาณที่สำคัญในแฟ้มรีจิสเตอร์

6.3 การออกแบบหน่วยคำนวณ, ตรรก และเลื่อน

หน่วยคำนวณ, ตรรก และเลื่อน (Arithmetic, Logic and Shift Unit) เป็นส่วนที่ใช้ทำการดำเนินการ (Operation) ทั้งหมดของหน่วยประมวลผลกลาง โดยตัวถูกดำเนินการ (Operand) จะมาจากบัส X และบัส Y ผลที่ได้จะส่งออกไปยังบัสผลลัพธ์ (Result Bus) และแฟล็ก (Flag) จะส่งไปยังหน่วยถอดรหัสและควบคุม (Control and Decode Unit) โครงสร้างภายในแสดงได้ดังรูปที่ 6.5



รูปที่ 6.5 โครงสร้างภายในของหน่วยคำนวณ, ตรรก และเลื่อน

หน่วยคำนวณ, ตรรก และเลื่อนถูกออกแบบโดยแยกเป็น 3 ส่วน คือ

- หน่วยคำนวณ (Arithmetic Unit) ใช้คำนวณทางคณิตศาสตร์ ได้แก่ การบวก และการลบ ออกแบบโดยใช้วงจรบวกลบ (Adder/Subtracter) ขนาด 8 บิตที่อยู่ในไลบรารี (Library) ที่มาพร้อมกับซอฟต์แวร์ของบริษัท Xilinx
- หน่วยตรรก (Logic Unit) ใช้ทำการดำเนินการทางตรรก ได้แก่ AND, OR และ XOR ข้อมูลขนาด 8 บิต
- หน่วยเลื่อน (Shift Unit) ใช้เลื่อนข้อมูลขนาด 8 บิต ไปทางซ้าย หรือทางขวา

สัญญาณควบคุมหน่วยคำนวณ, ตรรก และเลื่อนมี 2 ตัว คือ Enable ใช้สั่งให้ทำงานหรือไม่ทำงาน และ Oper. ใช้บอกว่าให้ทำการดำเนินการใด โดยสัญญาณทั้งสองจะถูกนำมาสร้างเป็นสัญญาณเพื่อควบคุมหน่วยคำนวณ, หน่วยตรรก และหน่วยเลื่อนอีกทีหนึ่ง สัญญาณ Oper. มีความหมายดังรูปที่ 6.6

	Oper [3:2]			
Oper [1:0]	00	01	10	11
00	Y	X Shift Left with '0'	X + Y + 0	Y + 0
01	X AND Y	X Shift Left with '1'	X + Y + 1	Y + 1
10	X OR Y	X Shift Right with '0'	X - Y - 0	Y - 0
11	X XOR Y	X Shift Right with '1'	X - Y - 1	Y - 1

รูปที่ 6.6 ตารางแสดงการดำเนินการของหน่วยคำนวณ, ตรรก และเลื่อน

สำหรับแฟล็ก (Flag) จะมี 2 อย่าง คือ C (Carry) เป็นแฟล็กที่บอกว่ามีการทดจากการบวกหรือการลบ หรือเป็นบิตข้อมูลที่ได้มาจากการเลื่อน กับ Z (Zero) เป็นแฟล็กที่บอกว่าผลการดำเนินการมีค่าเป็น 0 หรือไม่

6.4 ชุดคำสั่งของหน่วยประมวลผลกลาง

หน่วยประมวลผลกลางที่ออกแบบขึ้นมานี้ มีคำสั่งทั้งหมด 28 คำสั่ง โดยคำสั่งทั้งหมดแสดงไว้ในรูปที่ 6.7 ดังนี้

Instruction	Meaning
MOV Reg,#Const.	Move constant to register
MOV Reg,Reg	Move register to register
AND Reg,#Const.	Perform AND operation between register and constant
AND Reg,Reg	Perform AND operation between register and register
OR Reg,#Const.	Perform OR operation between register and constant
OR Reg,Reg	Perform OR operation between register and register
XOR Reg,#Const.	Perform XOR operation between register and constant
XOR Reg,Reg	Perform XOR operation between register and register
ADD Reg,#Const.	Add register with constant
ADD Reg,Reg	Add register with register
ADC Reg,#Const.	Add register with constant and carry flag

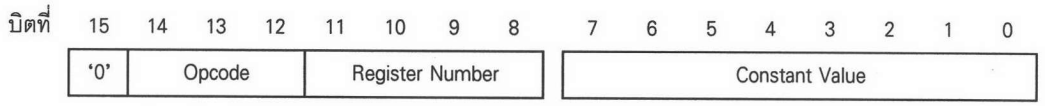
รูปที่ 6.7 ตารางแสดงคำสั่งของหน่วยประมวลผลกลาง

Instruction	Meaning
ADC Reg,Reg	Add register with register and carry flag
SUB Reg,#Const.	Subtract register with constant
SUB Reg,Reg	Subtract register with register
SUC Reg,#Const.	Subtract register with constant and carry flag
SUC Reg,Reg	Subtract register with register and carry flag
SHL Reg	Shift register left
SLC Reg	Shift register left with carry flag
SHR Reg	Shift register right
SRC Reg	Shift register right with carry flag
LOD Reg,@Rx:Ry	Load register from address Rx:Ry
STO Reg,@Rx:Ry	Store register to address Rx:Ry
JMP @Rx:Ry	Jump to absolute address Rx:Ry
JMP Rel 10 Bits	Jump to 10 bits relative address
JNC Rel 10 Bits	Jump when carry flag is cleared to 10 bits relative address
JC Rel 10 Bits	Jump when carry flag is set to 10 bits relative address
JNZ Rel 10 Bits	Jump when zero flag is cleared to 10 bits relative address
JZ Rel 10 Bits	Jump when zero flag is set to 10 bits relative address

รูปที่ 6.7 (ต่อ) ตารางแสดงคำสั่งของหน่วยประมวลผลกลาง

คำสั่ง (Instruction) แต่ละคำสั่งจะแทนด้วยรหัสคำสั่ง (Instruction Code) ขนาด 16 บิต ซึ่งรหัสคำสั่งนี้แบ่งออกเป็น 4 รูปแบบ ได้แก่ รีจิสเตอร์กับค่าคงที่, รีจิสเตอร์อย่างเดียว, Load กับ Store และการกระโดดสัมพัทธ์ แต่ละอย่างมีรายละเอียดดังนี้

1. *รีจิสเตอร์กับค่าคงที่* เป็นรูปแบบของรหัสคำสั่งที่สั่งให้มีการดำเนินการระหว่างรีจิสเตอร์กับค่าคงที่ ผลลัพธ์จากการดำเนินการจะเก็บไว้ที่รีจิสเตอร์ คำสั่งที่ใช้รูปแบบนี้ คือ คำสั่งที่อยู่ในลักษณะ Inst. Reg,#Const. รูปแบบของรหัสคำสั่งแสดงดังรูปที่ 6.8 โดยแต่ละเขต (Field) ของรหัสคำสั่งมีความหมายดังนี้



รูปที่ 6.8 รหัสคำสั่งรูปแบบรีจิสเตอร์กับค่าคงที่

- *Opcode* มีค่าตั้งแต่ '000' ถึง '111' เป็นเขตที่ระบุว่ารหัสคำสั่งนี้แทนคำสั่งใด โดยแต่ละค่าของเขตนี้จะตรงกับคำสั่งใดแสดงไว้ดังรูปที่ 6.9

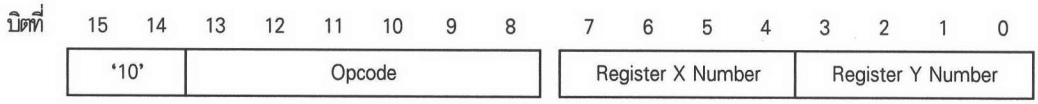
Opcode	Instruction
000	MOV Reg,#Const.
001	AND Reg,#Const.
010	OR Reg,#Const.
011	XOR Reg,#Const.

Opcode	Instruction
100	ADD Reg,#Const.
101	ADC Reg,#Const.
110	SUB Reg,#Const.
111	SUC Reg,#Const.

รูปที่ 6.9 ตารางแสดง Opcode เทียบกับคำสั่งสำหรับรูปแบบรีจิสเตอร์กับค่าคงที่

- *Register Number* เป็นเขตที่บอกหมายเลขรีจิสเตอร์ซึ่งเป็นตัวถูกดำเนินการ (Operand) ของคำสั่งนี้ มีค่าตั้งแต่ '0000' ถึง '1111' สำหรับ r0 ถึง rf ผลการดำเนินการจะเก็บไว้ที่รีจิสเตอร์ตัวนี้
- *Constant Value* เป็นเขตที่เก็บค่าคงที่ที่เป็นตัวถูกดำเนินการอีกตัวหนึ่งมีขนาด 8 บิต

2 . *<u>รีจิสเตอร์อย่างเดียว</u>* เป็นรูปแบบของรหัสคำสั่งที่ตัวถูกดำเนินการ (Operand) มีแต่รีจิสเตอร์เท่านั้น ไม่มีค่าคงที่ ผลลัพธ์จากการดำเนินการจะเก็บไว้ในรีจิสเตอร์ตัวแรก คำสั่งที่ใช้รูปแบบนี้ คือ คำสั่งที่อยู่ในลักษณะ Inst. Reg,Reg หรือ Inst. Reg มีรูปแบบของรหัสคำสั่งดังรูปที่ 6.10 โดยแต่ละเขตของรหัสคำสั่งมีความหมายดังนี้



รูปที่ 6.10 รหัสคำสั่งรูปแบบรีจิสเตอร์อย่างเดียว

- *Opcode* เป็นเขตที่ระบุว่ารหัสคำสั่งนี้แทนคำสั่งใด โดยแต่ละค่าของเขตนี้จะตรงกับคำสั่งใดแสดงไว้ดังรูปที่ 6.11

Opcode	Instruction
0000XX	MOV Reg,Reg
0001XX	AND Reg,Reg
0010XX	OR Reg,Reg
0011XX	XOR Reg,Reg
0100XX	ADD Reg,Reg
0101XX	ADC Reg,Reg
0110XX	SUB Reg,Reg
0111XX	SUC Reg,Reg

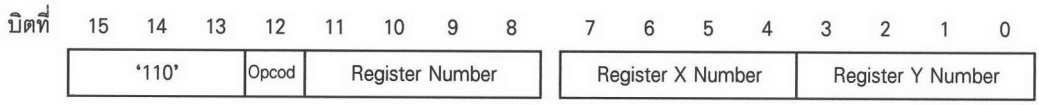
Opcode	Instruction
1000XX	SHL Reg
1001XX	SLC Reg
1010XX	SHR Reg
1011XX	SRC Reg
11XXXX	JMP @Rx:Ry

รูปที่ 6.11 ตารางแสดง Opcode เทียบกับคำสั่งสำหรับรูปแบบรีจิสเตอร์อย่างเดียว

สังเกตว่าคำสั่ง JMP @Rx:Ry เป็นคำสั่งที่อยู่ในรูปแบบนี้เช่นกัน เนื่องจากต้องใช้รีจิสเตอร์ 2 ตัวในการกำหนดตำแหน่งที่จะให้กระโดด แต่ไม่มีการดำเนินการใดเลย ทั้งนี้เป็นข้อยกเว้น

- *Register X Number* เป็นเขตที่บอกหมายเลขของรีจิสเตอร์ ที่เป็นตัวถูกดำเนินการ (Operand) ซึ่งเป็นตัวตั้ง ผลลัพธ์จากการดำเนินการจะเก็บไว้ในรีจิสเตอร์นี้ มีค่าตั้งแต่ '0000' ถึง '1111' สำหรับ r0 ถึง rf
- *Register Y Number* เป็นเขตที่บอกหมายเลขของรีจิสเตอร์ ที่เป็นตัวถูกดำเนินการอีกตัวหนึ่ง มีค่าตั้งแต่ '0000' ถึง '1111' สำหรับ r0 ถึง rf

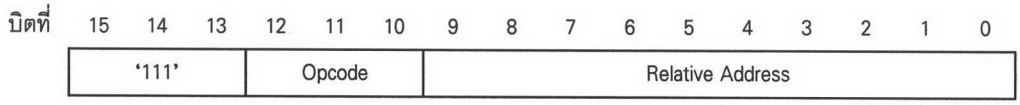
3. *Load กับ Store* เป็นรูปแบบของรหัสคำสั่งที่ใช้กับคำสั่ง LOD และ STO มีรูปแบบของรหัสคำสั่งดังรูปที่ 6.12 โดยแต่ละเขตของรหัสคำสั่งมีความหมายดังนี้



รูปที่ 6.12 รหัสคำสั่งรูปแบบ Load กับ Store

- *Opcode* เป็นเขตที่บอกว่ารหัสคำสั่งนี้ตรงกับคำสั่งใด มีค่า '0' กับ '1' สำหรับคำสั่ง LOD Reg,@Rx:Ry กับ STO Reg,@Rx:Ry ตามลำดับ
- *Register Number* เป็นเขตที่บอกหมายเลขของรีจิสเตอร์ที่จะถูก Load หรือ Store
- *Register X Number* เป็นเขตที่บอกหมายเลขของรีจิสเตอร์ที่ใช้เป็นตำแหน่งที่อยู่ 8 บิตบน ที่จะ Load หรือ Store
- *Register Y Number* เป็นเขตที่บอกหมายเลขของรีจิสเตอร์ที่ใช้เป็นตำแหน่งที่อยู่ 8 บิตล่าง ที่จะ Load หรือ Store

4. *การกระโดดสัมพัทธ์* เป็นรูปแบบของรหัสคำสั่งที่ใช้กับคำสั่งกระโดดแบบสัมพัทธ์ (Relative Jump) ซึ่งหน่วยประมวลผลกลางจะข้ามการทำงานตามคำสั่งถัดไป ไปทำคำสั่งที่ตำแหน่งใหม่ที่ระบุในคำสั่ง ซึ่งเป็นตำแหน่งที่สัมพัทธ์กับตำแหน่งปัจจุบัน มีรูปแบบของรหัสคำสั่งดังรูปที่ 6.13 โดยแต่ละเขตของรหัสคำสั่งมีความหมายดังนี้



รูปที่ 6.13 รหัสคำสั่งรูปแบบการกระโดดสัมพัทธ์

- *Opcode* เป็นเขตที่บอกว่ารหัสคำสั่งนี้แทนคำสั่งใด โดยแต่ละค่าของเขตนี้จะตรงกับคำสั่งใดแสดงไว้ดังรูปที่ 6.14

Opcode	Instruction
000	JNC Rel 10 Bits
001	JC Rel 10 Bits
010	JNZ Rel 10 Bits
011	JZ Rel 10 Bits
1XX	JMP Rel 10 Bits

รูปที่ 6.14 ตารางแสดง Opcode เทียบกับคำสั่งสำหรับรูปแบบการกระโดดสัมพัทธ์

- *Relative Address* เป็นเขตที่เก็บตำแหน่งที่อยู่สัมพัทธ์ (Relative Address) ซึ่งสัมพัทธ์กับตำแหน่งที่อยู่ของคำสั่งถัดไปที่หน่วยประมวลผลกลางจะทำงานตามปกติ ค่าที่เก็บเป็นแบบส่วนเติมเต็ม 2 (Two's Complement) คำสั่งที่กระโดดไปทำจะอยู่ที่ตำแหน่งที่อยู่ของคำสั่งถัดจากคำสั่งกระโดด บวกกับตำแหน่งที่อยู่สัมพัทธ์นี้

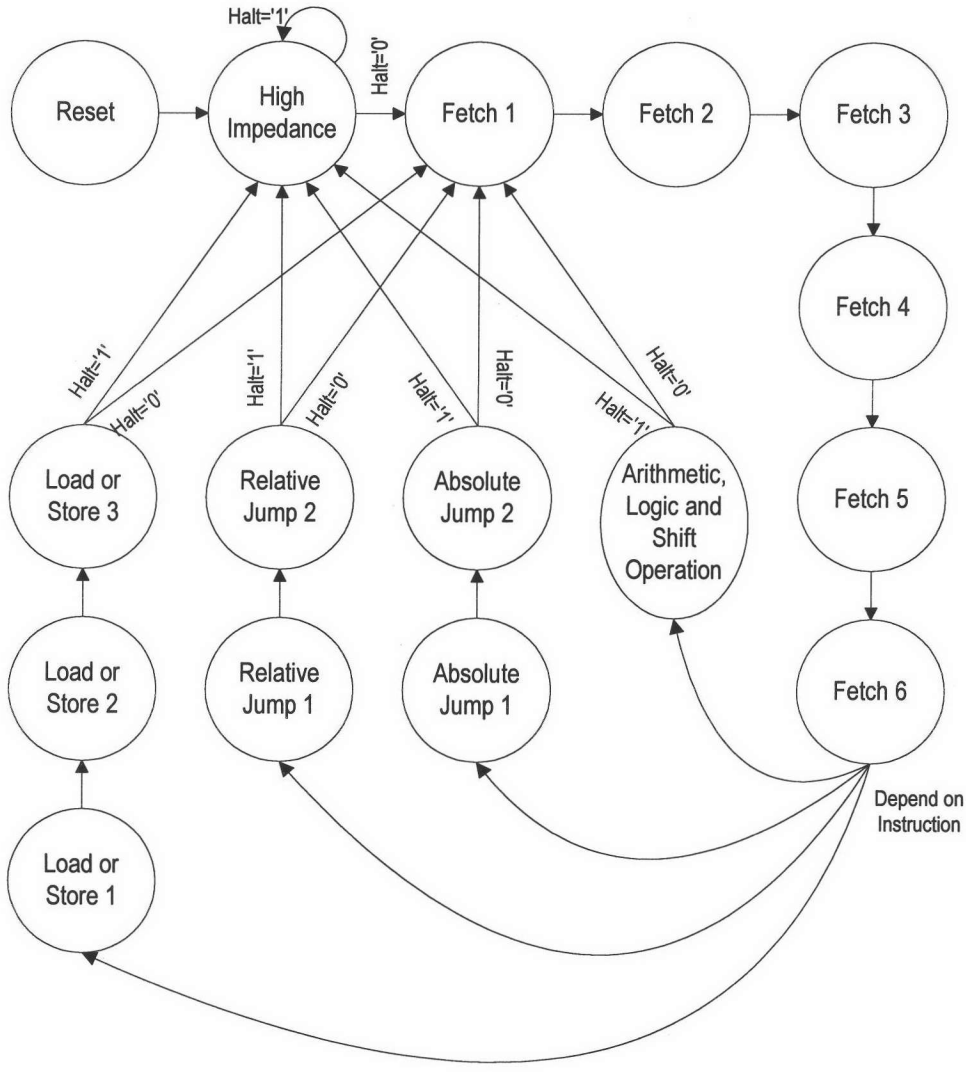
6.5 จังหวะการทำงานของหน่วยประมวลผลกลาง

การทำงานของหน่วยประมวลผลกลางจะขึ้นอยู่กับ หน่วยถอดรหัสและควบคุม (Control and Decode Unit) ซึ่งมีขั้นตอนการทำงานอธิบายได้ด้วยแผนภาพการเปลี่ยนสเตต (State Transition Diagram) ดังรูปที่ 6.15 โดยมีจำนวนสเตต (State) ทั้งหมด 16 สเตต แต่ละสเตตมีรายละเอียดดังนี้

1. *Reset* เป็นสเตตที่หน่วยประมวลผลกลางตั้งค่ารีจิสเตอร์ทุกตัว ในแฟ้มรีจิสเตอร์ (Register File) ให้เท่ากับ 0 เนื่องจากไม่มีการตั้งค่าเริ่มต้นของหน่วยความจำภายในชิป FPGA ที่ใช้ทำแฟ้มรีจิสเตอร์ หลังจากที่จ่ายไฟให้แก่ชิป

2. *High Impedance* เป็นสเตตที่หน่วยประมวลผลกลางหยุดการทำงานชั่วคราว และปล่อยบัสด้วยการให้ขา (Pin) สัญญาณออกที่ต่อไปยังบัสตำแหน่งที่อยู่ (Address Bus) และบัสข้อมูล (Data Bus) มีอิมพีแดนซ์สูง (High Impedance) หน่วยประมวลผลกลางจะค้างอยู่ที่สเตตนี้เมื่อสัญญาณหยุด (Halt) จากตัวกำเนิดการแสดงผลบนหน้าจอ (On Screen Display Generator) ที่กล่าวถึงไปแล้วในบทที่ 5 เป็น '1' จนกระทั่งสัญญาณนี้เป็น '0' จึงทำงานต่อ

3. *Fetch 1* เป็นสเตตที่หน่วยประมวลผลกลางส่งตำแหน่งที่อยู่ (Address) ที่จะอ่านไบต์แรกของคำสั่ง (Instruction) ถัดไป (คือที่ตำแหน่งที่อยู่ re:rf) ลงไปบนบัสตำแหน่งที่อยู่ (Address



รูปที่ 6.15 แผนภาพการเปลี่ยนสเตตของหน่วยประมวลผลกลาง

Bus) โดยรีจิสเตอร์ re และ rf จะถูกอ่านจากแฟ้มรีจิสเตอร์มายังบัส X และบัส Y ตามลำดับ ป้อนให้รีจิสเตอร์ตำแหน่งที่อยู่ (Address Register) เก็บค่าไว้ส่งไปบัสตำแหน่งที่อยู่

4. *Fetch 2* เป็นสเตตที่หน่วยประมวลผลกลางเพิ่มค่า 8 บิตล่างของตัวชี้คำสั่ง (Instruction Pointer) คือ รีจิสเตอร์ rf ขึ้นมา 1 โดยรีจิสเตอร์ rf จะถูกอ่านจากแฟ้มรีจิสเตอร์มายังบัส Y นำมาเพิ่มค่าขึ้น 1 ด้วยหน่วยคำนวณ, ตรรก และเลื่อน แล้วเขียนผลลัพธ์จากบัสผลลัพธ์ (Result Bus) กลับไปที่รีจิสเตอร์ rf ใหม่

5. *Fetch 3* เป็นสเตตที่หน่วยประมวลผลกลางเพิ่มค่า 8 บิตบนของตัวชี้คำสั่ง (Instruction Pointer) คือ รีจิสเตอร์ re ขึ้นมา 1 หากมีการทดมาจากการเพิ่มค่า 8 บิตล่าง (rf) ในสเตตที่

แล้ว และยังเป็นสเตตที่รีจิสเตอร์คำสั่ง (Instruction Register) อ่านคำสั่ง (Instruction) ไบต์แรกขึ้นมาจากบัสข้อมูล (Data Bus)

6. *Fetch 4* เป็นสเตตที่หน่วยประมวลผลกลางส่งตำแหน่งที่อยู่ (Address) ที่จะอ่านไบต์หลังของคำสั่ง (Instruction) ถัดไป (คือที่ตำแหน่งที่อยู่ re:rf) ลงไปบนบัสตำแหน่งที่อยู่ (Address Bus) โดยรีจิสเตอร์ re และ rf จะถูกอ่านจากแฟ้มรีจิสเตอร์มายังบัส X และบัส Y ตามลำดับ ป้อนให้รีจิสเตอร์ตำแหน่งที่อยู่ (Address Register) เก็บค่าไว้ส่งไปบัสตำแหน่งที่อยู่

7. *Fetch 5* เป็นสเตตที่หน่วยประมวลผลกลางเพิ่มค่า 8 บิตล่างของตัวชี้คำสั่ง (Instruction Pointer) คือ รีจิสเตอร์ rf ขึ้นมา 1 โดยรีจิสเตอร์ rf จะถูกอ่านจากแฟ้มรีจิสเตอร์มายังบัส Y นำมาเพิ่มค่าขึ้น 1 ด้วยหน่วยคำนวณ, ตรรก และเลื่อน แล้วเขียนผลลัพธ์จากบัสผลลัพธ์ (Result Bus) กลับไปที่รีจิสเตอร์ rf ใหม่

8. *Fetch 6* เป็นสเตตที่หน่วยประมวลผลกลางเพิ่มค่า 8 บิตบนของตัวชี้คำสั่ง (Instruction Pointer) คือ รีจิสเตอร์ re ขึ้นมา 1 หากมีการทดมาจากการเพิ่มค่า 8 บิตล่าง (rf) ในสเตตที่แล้ว และยังเป็นสเตตที่รีจิสเตอร์คำสั่ง (Instruction Register) อ่านคำสั่ง (Instruction) ไบต์หลังขึ้นมาจากบัสข้อมูล (Data Bus) หลังจากผ่านสเตตนี้แล้วหน่วยประมวลผลกลางจะทำงานสเตตใดต่อไปขึ้นอยู่กับคำสั่งที่อ่านเข้ามาได้

9. *Arithmetic, Logic and Shift Operation* เป็นสเตตที่หน่วยประมวลผลกลางจะทำงานก็ต่อเมื่อคำสั่งที่อ่านเข้ามาเป็นคำสั่ง MOV, AND, OR, XOR, ADD, ADC, SUB, SUC, SHL, SLC, SHR หรือ SRC โดยรีจิสเตอร์ที่เป็นตัวถูกดำเนินการ (Operand) จะถูกอ่านค่าลงบนบัส X และบัส Y เพื่อให้หน่วยหน่วยคำนวณ, ตรรก และเลื่อน ทำการดำเนินการ (Operation) ในกรณีที่ตัวถูกดำเนินการอีกตัวเป็นค่าคงที่ ค่าคงที่ของคำสั่งที่อยู่ในรีจิสเตอร์คำสั่งจะถูกปล่อยลงบนบัส Y แทน

10. *Absolute Jump 1* เป็นสเตตที่หน่วยประมวลผลกลางจะทำงานเมื่อเป็นคำสั่ง JMP @Rx:Ry ในสเตตนี้หน่วยประมวลผลกลางจะเปลี่ยนค่า 8 บิตล่าง ของตัวชี้คำสั่ง (Instruction Pointer) คือ รีจิสเตอร์ rf ให้เท่ากับรีจิสเตอร์ Ry ที่ระบุในคำสั่ง

11. *Absolute Jump 2* เป็นสเตตที่หน่วยประมวลผลกลางเปลี่ยนค่า 8 บิตบน ของตัวชี้คำสั่ง (Instruction Pointer) คือ รีจิสเตอร์ re ให้เท่ากับรีจิสเตอร์ Rx ที่ระบุในคำสั่ง JMP @Rx:Ry

12. *Relative Jump 1* เป็นสเตตที่หน่วยประมวลผลกลาง จะทำงานเมื่อมีการกระโดดแบบสัมพัทธ์ (Relative Jump) ในสเตตนี้หน่วยประมวลผลกลางจะบวกค่า 8 บิตล่าง ของตัวชี้คำสั่ง (Instruction Pointer) ด้วยตำแหน่งที่อยู่สัมพัทธ์ 8 บิตล่างในคำสั่ง โดยอ่านรีจิสเตอร์ rf จากแฟ้มรีจิสเตอร์ลงบนบัส X และอ่านตำแหน่งที่อยู่สัมพัทธ์ 8 บิตล่างจากรีจิสเตอร์คำสั่งลงบนบัส Y นำมาบวกกันด้วยหน่วยคำนวณ, ตรรก และเลื่อน แล้วเขียนผลลัพธ์จากบัสผลลัพธ์ (Result Bus) กลับไปที่รีจิสเตอร์ rf ใหม่

13. *Relative Jump 2* เป็นสเตตที่หน่วยประมวลผลกลางบวกค่า 8 บิตบน ของตัวชี้คำสั่ง (Instruction Pointer) ด้วยตำแหน่งที่อยู่สัมพัทธ์ 2 บิตบนในคำสั่งซึ่งขยายเป็น 8 บิต โดยอ่านรีจิสเตอร์ re จากแฟ้มรีจิสเตอร์ลงบนบัส X และอ่านตำแหน่งที่อยู่สัมพัทธ์ 8 บิตบนจากรีจิสเตอร์คำสั่งลงบนบัส Y นำมาบวกกันด้วยหน่วยคำนวณ, ตรรก และเลื่อน แล้วเขียนผลลัพธ์จากบัสผลลัพธ์ (Result Bus) กลับไปที่รีจิสเตอร์ re ใหม่

14. *Load or Store 1* เป็นสเตตที่หน่วยประมวลผลกลางจะทำงานก็ต่อเมื่อเป็นคำสั่ง LOD หรือ STO Reg,@Rx:Ry ในสเตตนี้หน่วยประมวลผลกลางจะส่งตำแหน่งที่อยู่ (Address) ที่ต้องการอ่าน หรือเขียนข้อมูล (คือที่ตำแหน่งที่อยู่ Rx:Ry) ลงไปบนบัสตำแหน่งที่อยู่ โดยอ่านรีจิสเตอร์ Rx และ Ry จากแฟ้มรีจิสเตอร์มายังบัส X และบัส Y ป้อนให้รีจิสเตอร์ตำแหน่งที่อยู่ (Address Register) เก็บค่าไว้ส่งไปบัสตำแหน่งที่อยู่

15. *Load or Store 2* เป็นสเตตที่หน่วยประมวลผลกลางส่งข้อมูลที่ต้องการเขียนลงไปบนบัสข้อมูล (Data Bus) ในกรณีที่เป็นคำสั่ง STO Reg,@Rx:Ry โดยอ่านค่ารีจิสเตอร์ Reg จากแฟ้มรีจิสเตอร์มายังบัส X ป้อนให้รีจิสเตอร์ข้อมูลออก (Data Out Register) เก็บค่าไว้ส่งไปยังบัสข้อมูล

16. *Load or Store 3* เป็นสเตตที่หน่วยประมวลผลกลางอ่านข้อมูลที่ต้องการจากบัสข้อมูล ในกรณีที่เป็นคำสั่ง LOD Reg,@Rx:Ry เข้ามายังบัสผลลัพธ์ (Result Bus) เขียนลงในรีจิสเตอร์ Reg ที่อยู่ในแฟ้มรีจิสเตอร์