

# องค์ประกอบคอมพิวเตอร์และภาษาแอสเซมบลี: กรณีศึกษา บอร์ด Raspberry Pi

ผศ.ดร.สุรินทร์ กิตติธรกุล

January 8, 2022

# คำนำ (Preface)

ตำราเล่มนี้ใช้ประกอบการเรียนการสอนวิชา Computer Organization and Assembly Language ตามหลักสูตรวิศวกรรมคอมพิวเตอร์ ภาควิชาวิศวกรรมคอมพิวเตอร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง โดยใช้คอมพิวเตอร์บอร์ดเดียวชื่อ Raspberry Pi3/Pi4 เป็นกรณีศึกษาหลัก ซึ่งภายในชิปประกอบด้วยชิปปีย์ Cortex A53/A72 จากบริษัท ARM เป็นชิปมัลติคอร์ จำนวน 4 แกนประมวลผล ผู้เรียนควรมีพื้นฐานวิชาออกแบบวงจรดิจิทัลและตรรกะ และวิชาการเขียนโปรแกรมคอมพิวเตอร์เบื้องต้นโดยเฉพาะภาษา C เนื้อหาในตำราเล่มนี้สามารถนำไปประยุกต์ใช้กับภาษา C ภาษาไพธอน (Python) หรือ บอร์ด Raspberry Pi4/Pi4 ได้ และสามารถใช้เป็นพื้นฐานของวิชาลำดับถัดไป เช่น สถาปัตยกรรมหรือระบบคอมพิวเตอร์ ระบบปฏิบัติการคอมพิวเตอร์ เป็นต้น ในตำราเล่มนี้ ผู้เขียนจะเน้นที่ขยายกำลังสอง เพื่อให้สอดคล้องกับการทำงานของคอมพิวเตอร์

ตารางที่ 1: ตารางเปรียบเทียบตัวคณด้วยค่าสองยกกำลังและสิบยกกำลัง

សອງកំតាំង	អានវា	គ្រាមានសិប	សិបកំតាំង	អានវា	គ្រាមានសិប
$2^{10}$	គិបិ (kibi)	1024	$10^3$	កិឡូ (kilo)	1000
$2^{20}$	មេបិ (mebi)	$1024^2$	$10^6$	មេភោ (mega)	$1000^2$
$2^{30}$	កិបិ (gibi)	$1024^3$	$10^9$	កិភោ (giga)	$1000^3$

เนื้อหา ครอบคลุม ทฤษฎี และ ปฏิบัติ โดย มี ภาค ผนวก เป็นการ ทดลอง ทั้งหมด 13 การ ทดลอง โดย ใช้ บอร์ด ตระกูล Raspberry Pi แต่ละ บท มี การสรุป/คำตามท้าย บท และ กิจกรรม ท้าย การ ทดลอง เสริม ความ เข้าใจ ผู้อ่าน สามารถ ค้นคว้า ศึกษา เพิ่มเติม เนื้อหา ทฤษฎี และ ปฏิบัติ เพิ่มเติม โดย การ คลิก ลิงก์ (ตัว อักษร สีน้ำเงิน) ไปยัง เว็บไซต์/เว็บเพจ/คลิป/รูปภาพ ใน อินเทอร์เน็ต เลข บท/ภาค ผนวก เลข รูป เลข ตาราง เลข หัวข้อ/นิยาม/ตัวอย่าง/สมการ เอกสาร อ้างอิง ที่ ผู้เขียน อ้างอิง ถึงได้ สะดวก รวดเร็ว และ กดปุ่ม คลิกลับ (Back) เพื่อย้อนกลับมาที่ เนื้อหา ปัจจุบันได้ ทั้งนี้ การใช้งาน ขึ้นอยู่ กับซอฟต์แวร์ ที่ใช้ เปิด อ่านไฟล์ PDF ที่ใช้

# บทคัดย่อ (Abstract)

ผู้เขียนแนะนำเครื่องคอมพิวเตอร์ชนิดต่างๆ ในบทที่ 1 เพื่อเชื่อมโยงกับเครื่องคอมพิวเตอร์ชนิดบอร์ดเดียว (Single Board Computer) ชื่อ Raspberry Pi รุ่นต่างๆ บทที่ 2 วางพื้นฐานด้านข้อมูล/ตัวแปรชนิดต่างๆ และคณิตศาสตร์ของเลขฐาน 2 ในเครื่องคอมพิวเตอร์ เพื่อเชื่อมโยงกับการพัฒนาโปรแกรมด้วยภาษา C/C++ บทที่ 3 อธิบายโครงสร้างด้านฮาร์ดแวร์และซอฟต์แวร์ ด้านฮาร์ดแวร์เน้นที่ซีพียู หน่วยความจำหลัก อินพุต/เอาต์พุต และอุปกรณ์เก็บรักษาข้อมูล ด้านซอฟต์แวร์อธิบายการทำงานของระบบปฏิบัติการของบอร์ด **Raspberry Pi3/Pi4** ในบทที่ 4 ซึ่งจำกัดเฉพาะคำสั่งภาษาแ.osซเมบลีและภาษาเครื่องความยาว 32 บิตและระบบปฏิบัติการลินุกซ์荷ด 32 บิต บทที่ 5 เชื่อมโยงการทำงานของเวอร์ชัลเม莫รีสำหรับบอร์ด Pi3/Pi4 พื้นฐานของแคชชนิด Direct Map และชนิด N-Way Set Associative หน่วยความจำสแตติกแรม และหน่วยความจำ SDRAM บทที่ 6 นำเสนออุปกรณ์และวงจรอินพุต/เอาต์พุตที่สำคัญของบอร์ด Pi3/Pi4 โดยเฉพาะขาซีพียูชนิด General Purpose Input Output (GPIO) เพื่อเป็นตัวอย่างเชื่อมโยงกับหลักการ Memory Mapped I/O การขัดจังหวะหรืออินเทอร์รัปท์ (Interrupt) และการเข้าถึงหน่วยความจำโดยตรง หรือ Direct Memory Access ซึ่งจะทำงานประสานกับระบบไฟล์ของอุปกรณ์เก็บรักษาข้อมูลโดยละเอียดในบทที่ 7 ท้ายที่สุดคือ บทที่ 8 นำหลักการคำนวณแบบขนาดมาประยุกต์ใช้กับบอร์ด Pi3/Pi4 ด้วยภาษา C และไลบรารี **OpenMP** ซึ่งเป็นที่ยอมรับในเครื่องคอมพิวเตอร์แบบขนาดในปัจจุบัน และชูเปอร์คอมพิวเตอร์ ชื่อ **Fugaku** ซึ่งภายใต้เครื่องประกอบด้วยซีพียูของ ARM ขนาด 64 บิต จำนวน 7.6 ล้านแกนประมาณผลลัพธ์

# กิติกรรมประกาศ (Acknowledgments)

ผู้เขียนขอแสดงความนับถือบุคคลเหล่านี้ที่ช่วยให้การเล่นนี้สำเร็จและมีข้อผิดพลาดน้อยลง ผศ.ดร.ชัยวัฒน์ หนูทอง ภาควิชาฯ วิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหาร ลาดกระบัง ดร.รัฐศิลป์ รานอกภานุวัชร์ หลักสูตรวิศวกรรมคอมพิวเตอร์ วิทยาลัยนวัตกรรมด้านเทคโนโลยีและ วิศวกรรมศาสตร์ มหาวิทยาลัยธุรกิจบัณฑิตย์ และ ผศ.ดร.อภิสิทธิ์ รัตนตราธุรักษ์ ภาควิชาฯ วิทยาการคอมพิวเตอร์ และสารสนเทศ คณะวิทยาศาสตร์ประยุกต์ มหาวิทยาลัยเทคโนโลยีพระจอมเกล้าพระนครเหนือ นักศึกษาภาควิชาฯ ที่เคยใช้ทำการเล่นนี้ประกอบการเรียนการสอนทั้งแบบออนไลน์และออนไลน์ ผ่านทาง [ยูทูบชเนนแนล](#)ของผู้เขียน หากทำรายนามมีจุดผิดพลาดหลงเหลืออยู่ ผู้เขียนขอน้อมรับไว้แต่เพียงผู้เดียว

# สารบัญ (Table of Contents)

คำนำ (Preface)	i
บทคัดย่อ (Abstract)	i
กิติกรรมประกาศ (Acknowledgments)	ii
สารบัญตาราง (Table of Tables)	xii
สารบัญรูป (Table of Figures)	xv
<b>1 บทนำ (Introduction)</b>	<b>1</b>
1.1 ชนิดของเครื่องคอมพิวเตอร์ .....	1
1.1.1 คอมพิวเตอร์ตั้งโต๊ะ (Desktop Computer) .....	1
1.1.2 คอมพิวเตอร์พกพา (Portable Computers) .....	1
1.1.3 คอมพิวเตอร์ฝังตัว (Embedded Computer) .....	2
1.1.4 คอมพิวเตอร์เซิร์ฟเวอร์หรือแม่ข่าย (Server Computer) .....	2
1.1.5 ซูเปอร์คอมพิวเตอร์ (Supercomputer) .....	2
1.2 แนวโน้มของจำนวนอุปกรณ์คอมพิวเตอร์ชนิดต่างๆ .....	3
1.3 คอมพิวเตอร์บอร์ดเดียว (Single Board) ตระกูล Raspberry Pi .....	3
1.4 ชิ้นตอนการผลิตชิป (Chip) หรือวงจรรวม (Integrated Circuit) .....	4
1.5 สรุปท้ายบท .....	6
1.6 คำถามท้ายบท .....	6
<b>2 ข้อมูลและคณิตศาสตร์ในคอมพิวเตอร์ (Computer Data and Arithmetic)</b>	<b>8</b>
2.1 ข้อมูลพื้นฐานชนิดต่างในภาษา C/C++ .....	9
2.2 เลขจำนวนเต็มฐานสอง: รูปแบบและค่าฐานสิบ .....	11
2.2.1 ชนิดไม่มีเครื่องหมาย (Unsigned Integer) .....	12
2.2.2 ชนิดมีเครื่องหมาย (Signed Integer) แบบ 2's Complement .....	16
2.2.3 ชนิดมีเครื่องหมาย (Signed Integer) แบบ Sign-Magnitude .....	23
2.3 คณิตศาสตร์เลขจำนวนเต็มฐานสอง .....	25
2.3.1 ชนิดไม่มีเครื่องหมาย .....	26
2.3.2 ชนิดมีเครื่องหมายแบบ 2's Complement .....	32

2.4	เลขทศนิยมฐานสองชนิดจุดตรึง (Fixed Point) . . . . .	34
2.5	เลขทศนิยมฐานสองชนิดจุดลอยตัว (Floating Point) . . . . .	35
2.5.1	การบวกเลขทศนิยมฐานสองชนิดจุดลอยตัว . . . . .	36
2.5.2	การคูณเลขทศนิยมฐานสองชนิดจุดลอยตัว . . . . .	37
2.6	เลขทศนิยมฐานสองชนิดจุดลอยตัวมาตรฐาน IEEE754 . . . . .	39
2.6.1	รูปแบบมาตรฐาน IEEE754 . . . . .	39
2.6.2	ค่าสูงสุด ต่ำสุดและค่าอื่นๆ ของมาตรฐาน IEEE754 . . . . .	43
2.6.3	การบวกเลขทศนิยมฐานสองชนิดจุดลอยตัวตาม IEEE754 . . . . .	44
2.6.4	การคูณเลขทศนิยมฐานสองชนิดจุดลอยตัวตาม IEEE754 . . . . .	48
2.7	ตัวอักษร (Character) และข้อความ (String) . . . . .	50
2.8	สรุปท้ายบท . . . . .	52
2.9	คำถามท้ายบท . . . . .	53
<b>3</b>	<b>ฮาร์ดแวร์และซอฟต์แวร์ของคอมพิวเตอร์ (Computer Hardware and Software)</b> . . . . .	<b>55</b>
3.1	ฮาร์ดแวร์ของเครื่องคอมพิวเตอร์: บอร์ด Pi3/Pi4 . . . . .	56
3.1.1	ชีพีयู (CPU: ARM Cortex A53/A72) . . . . .	58
3.1.2	หน่วยความจำหลัก (Main Memory) . . . . .	59
3.1.3	อุปกรณ์อินพุต/เอาต์พุต (Input/Output Devices) . . . . .	60
3.1.4	อุปกรณ์เก็บรักษาข้อมูล (Data Storage) . . . . .	63
3.2	การพัฒนาซอฟต์แวร์หรือโปรแกรมคอมพิวเตอร์ . . . . .	64
3.2.1	โครงสร้างของชอร์สโค้ดโปรแกรมภาษา C/C++ . . . . .	64
3.2.2	การแปลซอร์สโค้ดภาษา C/C++ ให้กลายเป็นโปรแกรมคอมพิวเตอร์ . . . . .	65
3.2.3	โครงสร้างของชอร์สโค้ดภาษาแอสเซมบลีของ ARM . . . . .	67
3.2.4	การรวมไฟล์อีบเจกต์ (Object) ด้วยลิงก์เกอร์ (Linker) . . . . .	69
3.2.5	ตัวอย่างของคำสั่งภาษาเครื่อง (Machine Code) ของ ARM . . . . .	71
3.3	การทำงานร่วมกันระหว่างฮาร์ดแวร์และซอฟต์แวร์ (Hardware-Software Operation) . . . . .	73
3.3.1	การบูต (Boot) ระบบปฏิบัติการ . . . . .	73
3.3.2	ระบบปฏิบัติการโหลดซอฟต์แวร์ประยุกต์จากไฟล์รูปแบบ ELF . . . . .	76
3.3.3	ชีพีयูเฟชคำสั่งภาษาเครื่องของซอฟต์แวร์ประยุกต์จากหน่วยความจำ . . . . .	78
3.3.4	ชีพียูอ่าน (Load)/เขียน (Store) ข้อมูลในหน่วยความจำหลัก . . . . .	80
3.3.5	ชีพียูใช้งานอินพุตและเอาต์พุตต่างๆ ตามคำสั่งของซอฟต์แวร์ประยุกต์ . . . . .	81
3.3.6	ชีพียูอ่าน/เขียนไฟล์ข้อมูลในอุปกรณ์เก็บรักษาข้อมูล . . . . .	82
3.3.7	ผู้ใช้ชัตดาวน์ (Shut Down) ระบบปฏิบัติการ . . . . .	83
3.4	สรุปท้ายบท . . . . .	84
3.5	คำถามท้ายบท . . . . .	84
<b>4</b>	<b>ภาษาแอสเซมบลีของ ARM ขนาด 32 บิต</b> . . . . .	<b>85</b>
4.1	โครงสร้างของชีพียู ARM Cortex A53/A72 . . . . .	85
4.2	สถาปัตยกรรมชุดคำสั่ง (Instruction Set Architecture) . . . . .	88
4.3	ตัวอย่างคำสั่งภาษาเครื่องในเทิกซ์เช็คเมนต์ . . . . .	89
4.4	คำสั่งประกาศและตั้งค่าเริ่มต้นของตัวแปรในดาต้าเช็คเมนต์ . . . . .	91

4.5	คำสั่งถ่ายโอน (Transfer) ค่าตัวแปรระหว่างดาต้าเซกเมนต์และรีจิสเตอร์ . . . . .	91
4.6	คำสั่งประมวลผลข้อมูล (Data Processing) จากรีจิสเตอร์ . . . . .	94
4.6.1	คำสั่งทางคณิตศาสตร์ (Arithmetic Instructions) . . . . .	95
4.6.2	คำสั่งเลื่อนบิตข้อมูล (Shift Instructions) . . . . .	96
4.6.3	คำสั่งทางคณิตศาสตร์และเลื่อนบิต (Arithmetic and Shift Instructions) . . . . .	97
4.6.4	คำสั่งทางตรรกศาสตร์ (Logical Instructions) . . . . .	98
4.7	คำสั่งควบคุมการทำงาน (Control Instructions) . . . . .	99
4.7.1	การตัดสินใจ IF . . . . .	101
4.7.2	การตัดสินใจ IF-ELSE . . . . .	103
4.7.3	การวนรอบชนิด FOR . . . . .	106
4.7.4	การวนรอบชนิด WHILE . . . . .	108
4.7.5	การวนรอบชนิด DO-WHILE . . . . .	110
4.7.6	การวนรอบชนิด FOR จำนวน 2 ชั้น . . . . .	111
4.8	การเรียกใช้ฟังก์ชัน (Function Call) . . . . .	112
4.8.1	การเรียกใช้ฟังก์ชันในภาษา C/C++ . . . . .	112
4.8.2	คำสั่งเรียกใช้ฟังก์ชันในภาษาแอสเซมบลี . . . . .	113
4.9	อุปกรณ์และคำสั่งภาษาแอสเซมบลีเวอร์ชันอื่นของ ARM . . . . .	116
4.9.1	อุปกรณ์ดิจิทัลที่ใช้ซีพียูที่ออกแบบโดยบริษัท ARM . . . . .	116
4.9.2	คำสั่งภาษาแอสเซมบลีเวอร์ชันอื่นของ ARM . . . . .	118
4.10	สรุปท้ายบท . . . . .	119
4.11	คำถามท้ายบท . . . . .	119
 5	 ลำดับชั้นของหน่วยความจำ (Hierarchy of Memory) . . . . .	121
5.1	ลำดับชั้นของหน่วยความจำของบอร์ด Pi3/Pi4 . . . . .	122
5.2	เวอร์ชวลเม莫รีชินิดเพจ (Paging Virtual Memory) . . . . .	124
5.2.1	หลักการพื้นฐานของเวอร์ชวลเม莫รี . . . . .	124
5.2.2	เวอร์ชวลเม莫รีชินิดเพจ กรณีศึกษาบอร์ด Pi3/Pi4 . . . . .	126
5.3	หน่วยความจำแคช (Cache Memory) . . . . .	130
5.3.1	แคชชนิด DM (Direct Map) . . . . .	132
5.3.2	แคชชนิด N-way SA (Set Associative) . . . . .	135
5.4	หน่วยความจำชนิด静态ติกแรม (Static RAM: SRAM) . . . . .	138
5.4.1	โครงสร้างภายในของชิปหน่วยความจำ static แรม . . . . .	139
5.4.2	การทำงานของ static แรม: ขบวนการอ่านข้อมูล . . . . .	140
5.4.3	การทำงานของ static แรม: ขบวนการเขียนข้อมูล . . . . .	141
5.5	หน่วยความจำหลักชนิด SDRAM . . . . .	142
5.5.1	โครงสร้างภายในของชิป SDRAM . . . . .	143
5.5.2	การทำงานของ SDRAM: อ่านและเขียน . . . . .	145
5.5.3	การรีเฟรช (Refresh) ข้อมูล . . . . .	146
5.6	สรุปท้ายบท . . . . .	147
5.7	คำถามท้ายบท . . . . .	148

<b>6</b>	<b>กลไกอินพุตและเอาต์พุต (Input and Output)</b>	<b>149</b>
6.1	สัญญาณ HDMI สำหรับจอภาพ LCD ขนาดใหญ่ . . . . .	151
6.2	สัญญาณ DSI สำหรับจอภาพ LCD ขนาดเล็ก . . . . .	154
6.3	สัญญาณ CSI สำหรับเชื่อมต่อกล้องขนาดเล็ก . . . . .	156
6.4	สัญญาณ PCM สำหรับข้อมูลเสียงดิจิทัล . . . . .	158
6.5	สัญญาณภาพและเสียงสำหรับจอทีวีอินแล็ป . . . . .	160
6.6	สัญญาณ USB สำหรับอุปกรณ์ต่อพ่วงต่างๆ . . . . .	161
6.7	สัญญาณ Ethernet สำหรับสายเชื่อมต่อกับเครือข่ายอินเทอร์เน็ต . . . . .	163
6.8	สัญญาณ WiFi และ Bluetooth สำหรับการสื่อสารไร้สาย . . . . .	164
6.9	หลักการ Memory Mapped Input/Output . . . . .	166
6.10	หัวเชื่อมต่อ 40 ขา (40-Pin Header) . . . . .	168
6.11	ขา GPIO (General Purpose Input Output) . . . . .	170
6.12	หลักการอินเทอร์รูปท์ (Interrupt) . . . . .	175
6.13	หลักการเข้าถึงหน่วยความจำโดยตรง (Direct Memory Access) . . . . .	179
6.14	แหล่งจ่ายไฟ (Power Supply) . . . . .	181
6.15	สรุปท้ายบท . . . . .	183
6.16	คำถาวรท้ายบท . . . . .	183
<b>7</b>	<b>อุปกรณ์เก็บรักษาข้อมูล (Data Storage Devices)</b>	<b>185</b>
7.1	ระบบไฟล์ (File System) . . . . .	186
7.1.1	การใช้งานไฟล์ (File Operations) . . . . .	186
7.1.2	การแบ่งพาร์ทิชัน (Partition) . . . . .	187
7.1.3	โครงสร้างของระบบไฟล์ยนิกซ์ (Unix) . . . . .	188
7.1.4	ไอโหนด (Inode) . . . . .	189
7.2	ชิปหน่วยความจำแฟลช (Flash Memory) . . . . .	192
7.2.1	การอ่านข้อมูลแบบเพจ (Page Read) . . . . .	194
7.2.2	การเขียนข้อมูล (Program Data) . . . . .	196
7.2.3	หน่วยความจำแฟลชชนิดอื่นๆ . . . . .	197
7.3	การ์ดหน่วยความจำ SD (Secure Digital) . . . . .	198
7.3.1	การอ่านข้อมูล . . . . .	200
7.3.2	การเขียนข้อมูล . . . . .	201
7.4	โซลิดสเตทไดรฟ์ (Solid-State Disk: SSD) . . . . .	202
7.5	ฮาร์ดดิสก์ไดรฟ์ (Hard Disk Drive: HDD) . . . . .	204
7.5.1	โครงสร้างของฮาร์ดดิสก์เชิงกายภาพ . . . . .	204
7.5.2	โครงสร้างของฮาร์ดดิสก์ไดรฟ์เชิงตรรกะ . . . . .	205
7.5.3	ความจุและประสิทธิภาพของฮาร์ดดิสก์ไดรฟ์ . . . . .	206
7.6	สรุปท้ายบท . . . . .	207
7.7	คำถาวรท้ายบท . . . . .	208

<b>8 การคำนวณแบบขนาน (Parallel Computing) ด้วยบอร์ด Pi3/Pi4</b>	<b>209</b>
8.1 เครื่องคอมพิวเตอร์แบบขนานชนิดมัลติคอร์ . . . . .	211
8.1.1 กรณีศึกษา ARM Cortex A72 และ Cortex A53/A72 . . . . .	211
8.1.2 กรณีศึกษา Fujitsu A64FX ในเครื่องซูเปอร์คอมพิวเตอร์ . . . . .	212
8.2 ความขนาน (Parallelism) . . . . .	214
8.2.1 ความขนานของข้อมูล (Data Parallelism) . . . . .	214
8.2.2 ความขนานของงานย่อย (Task Parallelism) . . . . .	215
8.2.3 ความขนานแบบผสม (Hybrid Parallelism) . . . . .	215
8.3 การพัฒนาอัลกอริธึมแบบมัลติเรลดและแบบขนานด้วยไลบรารี OpenMP . . . . .	216
8.3.1 การคูณแมทริกซ์ (Matrix Multiplication) แบบขนาน . . . . .	217
8.3.2 การเรียงข้อมูล (Sorting) แบบขนาน . . . . .	218
8.4 สรุปท้ายบท . . . . .	222
8.5 คำถามท้ายบท . . . . .	222
<b>ภาคผนวก (Appendices)</b>	<b>224</b>
<b>A การทดลองที่ 1 ข้อมูลและคณิตศาสตร์ในคอมพิวเตอร์</b>	<b>224</b>
A.1 การแปลงและคณิตศาสตร์สำหรับเลขจำนวนเต็มฐานสอง . . . . .	225
A.1.1 การทดลองแปลงเลขฐาน . . . . .	225
A.1.2 คณิตศาสตร์เลขจำนวนเต็มฐานสอง . . . . .	229
A.1.3 กิจกรรมท้ายการทดลอง . . . . .	230
A.2 การแปลงและคณิตศาสตร์เลขทศนิยมฐานสองมาตรฐาน IEEE754 . . . . .	232
A.2.1 เลขทศนิยมชนิดจุดลอยตัวมาตรฐาน IEEE754 Single-Precision . . . . .	232
A.2.2 กิจกรรมท้ายการทดลอง . . . . .	236
A.3 รหัสของข้อมูลตัวอักษร . . . . .	238
A.3.1 การทดลอง . . . . .	238
A.3.2 กิจกรรมท้ายการทดลอง . . . . .	239
<b>B การทดลองที่ 2 ตัวอย่างการประกอบและติดตั้งบอร์ด Pi</b>	<b>240</b>
B.1 รายการอุปกรณ์ฮาร์ดแวร์ . . . . .	241
B.2 ตัวอย่างการประกอบบอร์ด Pi และกล่อง . . . . .	242
B.2.1 ประกอบฮีทซิงก์ (Heat Sink) . . . . .	242
B.2.2 ประกอบกล่องกับบอร์ด Pi 3 . . . . .	243
B.3 เครื่องคอมพิวเตอร์ส่วนบุคคลจากบอร์ด Pi 3 โมเดล B . . . . .	245
B.4 กิจกรรมท้ายการทดลอง . . . . .	246
<b>C การทดลองที่ 3 การติดตั้งระบบปฏิบัติการ Raspberry Pi OS</b>	<b>247</b>
C.1 การเตรียมการ์ดหน่วยความจำไมโคร SD . . . . .	247
C.2 การติดตั้ง RaspberryPi OS บนการ์ดหน่วยความจำไมโคร SD . . . . .	248
C.3 การรูปแบบปฏิบัติการ RaspberryPi OS . . . . .	252
C.4 การตั้งค่าบอร์ด Pi เพื่อใช้งาน . . . . .	253

C.4.1	การตั้งค่าต่างๆ . . . . .	253
C.4.2	การตั้งค่า WiFi เพื่อเชื่อมต่อกับอินเทอร์เน็ต . . . . .	254
C.4.3	การรีสตาร์ตและซัตดาวน์ . . . . .	255
C.5	กิจกรรมท้ายการทดลอง . . . . .	256
<b>D</b>	<b>การทดลองที่ 4 การใช้งานระบบปฏิบัติการยูนิกซ์เบื้องต้น</b>	<b>257</b>
D.1	การใช้งานระบบผ่านทาง GUI . . . . .	257
D.1.1	หน้าจอหลัก (Desktop) . . . . .	257
D.1.2	ไฟล์เมเนจเจอร์ (File Manager) . . . . .	258
D.2	การใช้งานระบบผ่านทางโปรแกรม Terminal . . . . .	259
D.2.1	คำสั่งพื้นฐานของระบบยูนิกซ์ . . . . .	261
D.2.2	การซัตดาวน์ (Shutdown) . . . . .	261
D.3	ข้อมูลพื้นฐานของบอร์ด Pi . . . . .	262
D.3.1	ข้อมูลพื้นฐานของซีพียู . . . . .	262
D.3.2	ข้อมูลขั้นสูงของซีพียูและบอร์ด . . . . .	263
D.4	กิจกรรมท้ายการทดลอง . . . . .	264
<b>E</b>	<b>การทดลองที่ 5 การพัฒนาโปรแกรมภาษา C บนลินุกซ์</b>	<b>265</b>
E.1	การพัฒนาโดยใช้ IDE . . . . .	265
E.2	การดีบัก (Debugging) โดยใช้ IDE . . . . .	268
E.3	การพัฒนาโดยใช้ประโยชน์คำสั่งที่ลีฟ์ชั่นตอน . . . . .	269
E.4	โครงสร้างของ Makefile . . . . .	270
E.5	การพัฒนาโดยใช้ Makefile . . . . .	271
E.6	การตรวจจับ Overflow กรณิตศาสตร์เลขจำนวนเต็มฐานสอง . . . . .	272
E.6.1	เลขจำนวนเต็มฐานสองไม่มีเครื่องหมาย . . . . .	272
E.6.2	เลขจำนวนเต็มฐานสองมีเครื่องหมายชนิด 2's Complement . . . . .	273
E.7	กิจกรรมท้ายการทดลอง . . . . .	274
<b>F</b>	<b>การทดลองที่ 6 การพัฒนาโปรแกรมภาษาแอสเซมบลี</b>	<b>275</b>
F.1	การพัฒนาโดยใช้ IDE . . . . .	275
F.2	การดีบักโปรแกรมโดยใช้ IDE . . . . .	278
F.3	การพัฒนาโดยใช้ประโยชน์คำสั่งที่ลีฟ์ชั่นตอน . . . . .	278
F.4	การพัฒนาโดยใช้ Makefile . . . . .	279
F.5	กิจกรรมท้ายการทดลอง . . . . .	280
<b>G</b>	<b>การทดลองที่ 7 การเรียกใช้และสร้างฟังก์ชันในโปรแกรมภาษาแอสเซมบลี</b>	<b>282</b>
G.1	การใช้งานตัวแปรในดาต้าเซกเมนต์ . . . . .	282
G.1.1	การโหลดค่าตัวแปรเดี่ยวจากหน่วยความจำมาพักในรีจิสเตอร์ . . . . .	283
G.1.2	การใช้งานตัวแปรชุดหรืออาร์เรย์ ชนิด word . . . . .	285
G.1.3	การใช้งานตัวแปรอาร์เรย์ชนิด byte . . . . .	286

G.1.4	การเรียกใช้ฟังก์ชันและตัวแปรชนิดประโยครหัส ASCII . . . . .	287
G.2	การสร้างฟังก์ชันเสริมด้วยภาษาแอสเซมบลี . . . . .	289
G.3	กิจกรรมท้ายการทดลอง . . . . .	293
<b>H</b>	<b>การทดลองที่ 8 การพัฒนาโปรแกรมภาษาแอสเซมบลีขั้นสูง</b>	<b>294</b>
H.1	ดีบักเกอร์ GDB . . . . .	294
H.2	การใช้งานสเต็กพอยน์เตอร์ (Stack Pointer) . . . . .	299
H.3	การพัฒนาโปรแกรมภาษาแอสเซมบลีร่วมกับภาษา C . . . . .	301
H.4	กิจกรรมท้ายการทดลอง . . . . .	302
<b>I</b>	<b>การทดลองที่ 9 การศึกษาและปรับแก้อินพุตและเอาต์พุตต่างๆ</b>	<b>303</b>
I.1	จอแสดงผลผ่านพอร์ต HDMI . . . . .	303
I.1.1	การปรับแก้ความละเอียดของจอแสดงผล . . . . .	303
I.1.2	การปรับแก้ขนาดหน่วยความจำของ GPU . . . . .	305
I.2	ระบบเสียงดิจิทัล . . . . .	306
I.2.1	การเลือกช่องสัญญาณเสียงเขื่อมต่อกับลำโพง . . . . .	306
I.2.2	การควบคุมระดับเสียง . . . . .	309
I.2.3	รายชื่ออุปกรณ์ในระบบเสียง . . . . .	309
I.3	พอร์ตเชื่อมต่ออุปกรณ์ USB . . . . .	311
I.3.1	รายชื่ออุปกรณ์กับพอร์ต USB . . . . .	311
I.3.2	รายละเอียดการเชื่อมต่ออุปกรณ์กับพอร์ต USB . . . . .	312
I.4	พอร์ตเชื่อมต่อเครือข่าย WiFi และ Ethernet . . . . .	314
I.4.1	รายชื่ออุปกรณ์เครือข่าย . . . . .	314
I.4.2	การเปิด/ปิดอุปกรณ์เครือข่าย . . . . .	315
I.4.3	การตรวจสอบการเชื่อมต่อกับเครือข่ายเบื้องต้น . . . . .	316
I.5	กิจกรรมท้ายการทดลอง . . . . .	317
<b>J</b>	<b>การทดลองที่ 10 การเชื่อมต่อกับขา GPIO</b>	<b>318</b>
J.1	ไลบรารี wiringPi . . . . .	318
J.2	วงจรไฟ LED กระพริบ . . . . .	320
J.3	โปรแกรมไฟ LED กระพริบภาษา C . . . . .	321
J.4	โปรแกรมไฟ LED กระพริบภาษาแอสเซมบลี . . . . .	323
J.5	กิจกรรมท้ายการทดลอง . . . . .	325
<b>K</b>	<b>การทดลองที่ 11 การเชื่อมต่ออินพุต-เอาต์พุตกับสัญญาณอินเทอร์รัปท์</b>	<b>327</b>
K.1	การจัดการอินเทอร์รัปท์ของ WiringPi . . . . .	327
K.2	วงจรสวิตซ์ปุ่มกดเชื่อมผ่านขา GPIO . . . . .	328
K.3	โปรแกรมภาษา C สำหรับทดลองวงจรอินเทอร์รัปท์ . . . . .	328
K.4	กิจกรรมท้ายการทดลอง . . . . .	330

L การทดลองที่ 12 การศึกษาอุปกรณ์เก็บรักษาข้อมูลและระบบไฟล์	331
L.1 ขนาดของไฟล์และไดเรกทอรี . . . . .	331
L.2 ระบบไฟล์ . . . . .	333
L.3 อุปกรณ์อินพุต/เอาต์พุตในระบบไฟล์ . . . . .	336
L.4 กิจกรรมท้ายการทดลอง . . . . .	338
M การทดลองที่ 13 การพัฒนาอัลกอริธึมแบบขนาดด้วยไลบรารี OpenMP	339
M.1 การวัด CPU Utilization . . . . .	339
M.2 การคุณเมทริกซ์แบบขนาด . . . . .	340
M.3 ความซับซ้อน (Complexity) ของการคำนวณ . . . . .	344
M.4 ประสิทธิภาพ (Performance) ของการคำนวณแบบขนาด . . . . .	345
M.5 กิจกรรมท้ายการทดลอง . . . . .	346
เอกสารอ้างอิง (Bibliography)	347
อภิธานศัพท์ (Glossary)	349
ดัชนี (Index)	350

# สารบัญตาราง (Table of Tables)

1	ตารางเปรียบเทียบตัวคูณด้วยค่าสองยกกำลังและสิบยกกำลัง . . . . .	i
2.1	ชนิด ความหมาย (บิต) ค่าฐานสิบต่ำสุดและสูงสุดของตัวแปรพื้นฐานแต่ละชนิดในภาษา C/C++ หมายเหตุ ค่าต่ำสุดและค่าสูงสุดของ IEEE754 เป็นค่าเลขnoramlize' (Normalize) เพื่อแสดงความสมมาตรของเลขยกกำลัง ที่มา: ntu.edu.sg . . . . .	9
2.2	การแปลงค่าฐานสิบเป็นเลขฐานสองแบบไม่มีเครื่องหมายความหมาย $n=8$ บิต ด้วยการหาร	14
2.3	การแปลงค่าฐานสิบเป็นเลขฐานสองแบบไม่มีเครื่องหมายความหมาย $n=8$ บิต ด้วยวิธีการลบ . . . . .	15
2.4	รูปแบบ (Pattern) ของเลขฐานสองขนาด $n=4$ บิตทั้งหมด $2^4=16$ แบบและการตีความหมายให้เป็นค่าฐานสิบแบบมีเครื่องหมาย 2's Complement และแบบไม่มีเครื่องหมาย	17
2.5	รูปแบบ (Pattern) ของเลขฐานสองขนาด $n=5$ บิตทั้งหมด $2^5=32$ แบบและการตีความหมายให้เป็นค่าฐานสิบแบบมีเครื่องหมาย 2's Complement และแบบไม่มีเครื่องหมาย	18
2.6	รูปแบบ (Pattern) ของเลขฐานสองขนาด $n=8$ บิตทั้งหมด $2^8=256$ แบบและการตีความหมายให้เป็นค่าฐานสิบแบบมีเครื่องหมาย 2's Complement และแบบไม่มีเครื่องหมาย . . . . .	19
2.7	การแปลงค่าฐานสิบแบบมีเครื่องหมายให้เป็นเลขฐานสองความหมาย $n=4$ บิต โดยแปลงให้เป็นค่าฐานสิบโดยใช้สมการที่ (2.32) กรณีที่ $X_{10,s} < 0$ . . . . .	21
2.8	การแปลงค่าฐานสิบแบบมีเครื่องหมายให้เป็นเลขฐานสองความหมาย $n=5$ บิต โดยแปลงให้เป็นค่าฐานสิบโดยใช้สมการที่ (2.32) กรณีที่ $X_{10,s} < 0$ . . . . .	22
2.9	เลขจำนวนเต็มฐานสองความหมาย $n=4$ บิตทั้งหมด $2^4=16$ แบบ สามารถตีความเป็นเลขจำนวนเต็มฐานสิบแบบมีเครื่องหมายชนิด Sign-Magnitude, แบบมีเครื่องหมายชนิด 2's Complement, และแบบไม่มีเครื่องหมาย (Unsigned) . . . . .	24
2.10	ผลคูณเลขขนาด 4 บิต ตัวตั้ง= $1010_2$ ( $10_{10}$ ) ตัวคูณ= $1101_2$ ( $13_{10}$ ) เท่ากับ $130_{10}$ ด้วยวงจรในรูปที่ 2.5 หมายเหตุ Shift คือ การเลื่อนบิต . . . . .	29
2.11	ผลคูณเลขขนาด 4 บิต ตัวตั้ง= $1010_2$ ( $10_{10}$ ) ตัวคูณ= $1101_2$ ( $13_{10}$ ) เท่ากับ $130_{10}$ ด้วยวงจรในรูปที่ 2.6 หมายเหตุ Shift R คือ การเลื่อนบิตไปทางขวา . . . . .	31
2.12	ตารางสรุปความแตกต่างระหว่างเลขศนนิยมฐานสองชนิด จุดลอยตัว ตามมาตรฐาน IEEE754 ชนิด Single Precision โดย $E_{2,IEEE}$ คือ ค่ายกกำลัง และ $Y_2$ คือ ศนนิยมซึ่งเป็นส่วนประกอบของค่านัยสำคัญ (x หมายถึง บิตข้อมูลมีค่าเท่ากับ '0' หรือ '1') . .	43
2.13	ชนิด ความหมาย ข้อมูล และการประยุกต์ใช้งานเลขฐานสองชนิดต่างๆ ในคอมพิวเตอร์ . .	52
3.1	ตารางสรุปข้อมูลด้านฮาร์ดแวร์และซอฟต์แวร์ของบอร์ด Pi3/Pi4 โนเดล B และลิงก์เชื่อมไปยังหัวข้อที่แสดงรายละเอียดในบทต่างๆ . . . . .	57

3.2	ตัวอย่างโปรแกรมภาษาแオスเซมบลีเพื่อประกาศและตั้งค่าเริ่มต้นให้ตัวแปรขนาด 32 บิต จำนวน 4 ตัวแปร และวนรอบวงค่าโดยใช้ตัวแปรพอยน์เตอร์ . . . . .	68
4.1	ตัวอย่างโปรแกรมภาษาแオスเซมบลีเพื่อประกาศและตั้งค่าเริ่มต้นตัวแปรขนาด 32 บิต จำนวน 2 ตัวแปร ในพื้นที่ของดาต้าเซ็กเมนต์ . . . . .	91
4.2	ตัวอย่างโปรแกรมภาษาแオスเซมบลีเพื่ออ่านค่าตัวแปรจากดาต้าเซ็กเมนต์ . . . . .	93
4.3	ตัวอย่างโปรแกรมภาษาแオスเซมบลีเพื่อกำหนดประโยชน์ $x = (a + b) - c$ . . . . .	94
4.4	ตัวอย่างโปรแกรมตามประโยชน์ $x = \text{if}$ . . . . .	102
4.5	ตัวอย่างโปรแกรมตามประโยชน์ $x = \text{if-else}$ . . . . .	104
4.6	ตัวอย่างโปรแกรมประยุกต์ $\text{For}$ ตามสมการที่ (4.1) . . . . .	107
4.7	ตัวอย่างโปรแกรมตามประยุกต์ $\text{While}$ ตามสมการที่ (4.1) . . . . .	109
4.8	ตัวอย่างโปรแกรมตามประยุกต์ $\text{Do-While}$ ตามสมการที่ (4.1) . . . . .	111
4.9	ตัวอย่างโปรแกรมเรียกใช้ฟังก์ชันภาษาแオスเซมบลีที่คล้ายกับตัวอย่างที่ 4.8.1 . . . . .	114
4.10	ชนิดและจำนวนอุปกรณ์ ( $\times 10^6$ ) จำนวนชิป ( $\times 10^6$ ) จำนวนชิปต่ออุปกรณ์ จำนวนชิป TAM (Total Available Market) ( $\times 10^6$ ) จำนวนชิปของ ARM ( $\times 10^6$ ) และเปอร์เซ็นต์ ส่วนแบ่งการตลาด (Share) ของ ARM ในปี ค.ศ. 2010 ที่มา: zdnet.com หมายเหตุ TAM: Total Available Market . . . . .	116
4.11	ชนิดอุปกรณ์ จำนวนชิป TAM (Total Available Market) ปี 2010 ( $\times 10^6$ ) จำนวนชิปของ ARM ปี 2010 ( $\times 10^6$ ) จำนวนอุปกรณ์ TAM ปี 2015 ( $\times 10^6$ ) จำนวนชิปต่อ อุปกรณ์ และจำนวนชิป TAM ปี 2015 ( $\times 10^6$ ) ที่มา: 7boot.com หมายเหตุ TAM: Total Available Market . . . . .	117
5.1	ตัวอย่างโปรแกรมภาษาแオスเซมบลีเพื่อประกอบการอธิบายการทำงานของแคชทั้งสองชนิด . . . . .	132
6.1	หมายเลขอ ชื่อ และวัตถุประสงค์ของสายสัญญาณชนิด HDMI เวอร์ชัน 1.4 ที่มา: wikipedia.org . . . . .	152
6.2	หมายเลขอ และหน้าที่ของสายสัญญาณ DSI สำหรับจอภาพ LCD ขนาดเล็ก ประกอบ ด้วยข้อมูลภาพจำนวน 2 เลน ที่มา: wikipedia.org . . . . .	154
6.3	หมายเลขอ ชื่อ และวัตถุประสงค์ของสายสัญญาณชนิด CSI wikipedia.org . . . . .	157
6.4	ตารางเชื่อมโยงระหว่าง VC/CPU บัสแอดเดรส อุปกรณ์อินพุต/เอาต์พุต (I/O Peripherals) และหมายเลขอหัวข้อของตารางเล่มนี้ เริ่มต้นที่หมายเลข 0x7E00 0000 หมายเหตุ Rx: Receiver, Tx: Transmitter, UART: Universal Asynchronous/Synchronous Receiver Transmitter ที่มา: Broadcom (2012) . . . . .	167
6.5	หมายเลขอ ชื่อ ฯ และตัวเลือก (ชื่อสัญญาณ) ที่ 0-5 ของหัวเข็มต่อสายทั้ง 4 ขา (GND: Ground) ที่มา: Broadcom (2012), ที่มา: Raspberry Pi (Trading) (2019) .	169
6.6	ตารางเชื่อมโยงระหว่าง VC/CPU บัสแอดเดรสกับรีจิสเตอร์ต่างๆ ของวงจร GPIO เพื่อ ตั้งค่าและสั่งงานขา GPIO ทุกขา โดยเริ่มต้นที่หมายเลข 0x7E20 0000 ที่มา: Broadcom (2012) . . . . .	172
6.7	ตารางเลือกค่า $V_{OUT}$ และค่าตัวเหนี่ยวนำ $L_1$ และ $L_2$ ที่มา: Diodes (2012) . . . . .	182

7.1	ตารางเปรียบเทียบเซลล์หน่วยความจำแฟลชชนิด NAND (ซ้าย) และ NOR (ขวา) ตามโครงสร้างและผังการจัดวางของเซลล์หน่วยความจำ ที่มา: Choi (2010) หมายเหตุ F คือ ความกว้างของไฟลต์เกต (Float Gate) มีหน่วยเป็น นาโนเมตร . . . . .	197
7.2	หมายเลขอื่น ข้อ และวัตถุประสงค์ของสัญญาณในหน่วย SD 4 กิกะบิต ที่มา: wikipedia.org	199
7.3	การเปรียบเทียบประสิทธิภาพของอุปกรณ์เก็บรักษาข้อมูลชนิดต่างๆ . . . . .	207
M.1	เวลาและ $\%CPU_{max}$ ของการคูณเมทริซที่ขนาด $N$ และจำนวนเรลดเท่ากับ 1, 2, 4, 8 เเรด . . . . .	343
M.2	อัตราส่วนเวลาการคูณเมทริซที่ขนาด $N$ และเวลาที่ขนาด 200 ที่จำนวนเรลดเท่ากับ 1, 2, 4, 8 เเรด จากสมการที่ (M.2) . . . . .	344
M.3	ผลการคำนวณ $Speedup(n)$ ของการคูณเมทริซที่ขนาด $N$ และจำนวนเรลดเท่ากับ 2, 4, 8 เเรดเทียบกับ 1 เเรดด้วยสมการที่ (M.5) . . . . .	345

# สารบัญรูป (Table of Figures)

1.1	ตัวอย่างเครื่องคอมพิวเตอร์ชนิดเซิร์ฟเวอร์ จากบริษัท Hewlett Packard รุ่น Proliant ที่มา: datacenterknowledge.com . . . . .	2
1.2	ปริมาณยอดขาย (ล้านหน่วย) เครื่องคอมพิวเตอร์ชนิดเดสค์ทอป โน้ตบุ๊ก สมาร์ตโฟน และแท็บเล็ตทั่วโลกในปี ค.ศ. 2010-2016 ที่มา: statista.com . . . . .	3
1.3	ภาพถ่ายซิลิโคนดาย (Silicon Die) ของชิป BCM2835 บนบอร์ด Raspberry Pi ที่มา: raspberrypi.org . . . . .	4
1.4	ขั้นตอนการผลิตชิป (Chip) ภายในประกอบด้วยวงจรรวม ไมโครprocเซอร์และอื่นๆ ที่มา: slideplayer.com . . . . .	5
2.1	เลขจำนวนเต็มและเลขจำนวนจริงบนเส้นจำนวนในคณิตศาสตร์เลขฐานสิบ ที่มา: tu- tortvista.com . . . . .	9
2.2	พื้นที่ในหน่วยความจำซึ่งบรรจุค่าเริ่มต้นของตัวแปรที่ได้ประกาศในตัวอย่างที่ 2.1.1 . . . . .	10
2.3	เส้นจำนวนเปรียบเทียบข้อมูลเลขฐานสองความยาว 4 บิตในรูปแบบของเลขจำนวนเต็ม ชนิด Unsigned ชนิด 2's Complement และชนิด Sign-Magnitude ที่มา: Harris and Harris (2013) . . . . .	11
2.4	สัญลักษณ์ของ ALU (Arithmetic Logic Unit) สำหรับบวก/ลบเลขจำนวนเต็มขนาด $n$ บิตชนิดไม่มีเครื่องหมาย ที่มา: teach-ict.com . . . . .	25
2.5	วงจรคูณเลขขนาด $n=32$ บิต ชนิดที่ 1 โดยใช้ ALU ขนาด $2n = 64$ บิต และรีจิสเตอร์ ตัวตั้งขนาด $2n = 64$ บิต ที่มา: Patterson and Hennessy (2016) . . . . .	28
2.6	วงจรคูณเลขขนาด $n=32$ บิต ชนิดที่ 2 โดยใช้ ALU ขนาด $n=32$ บิต และรีจิสเตอร์ตัว ตั้งขนาด $n=32$ บิต ที่มา: Patterson and Hennessy (2016) . . . . .	30
2.7	โครงสร้างของเลขทศนิยมฐานสองชนิดจุดลอยตัวตามมาตรฐาน IEEE754 Double Precision และ Single Precision ที่มา: pybugs.com . . . . .	39
2.8	เลขทศนิยมโดยตัวชนิดน้อยมัลลิลีอร์ฟตั้งแต่ $\pm N_{min}$ ถึง $\pm N_{max}$ และชนิดดีนอร์ฟตั้ง แต่ $\pm D_{min}$ ถึง $\pm D_{max}$ ที่มา: ntu.edu.sg . . . . .	44
2.9	วงจรบวกเลขชนิดจุดลอยตัวตามมาตรฐาน IEEE754 โดยรับค่าอินพุตจำนวน 2 ตัวด้าน บน และเอาต์พุตผลลัพธ์ด้านล่าง ที่มา: Patterson and Hennessy (2016) . . . . .	45
2.10	วงจรคูณเลขทศนิยมฐานสองชนิดจุดลอยตัวตามมาตรฐาน IEEE754 โดยรับค่าอินพุต จำนวน 2 ตัวด้านบน และเอาต์พุตผลลัพธ์ด้านล่าง ที่มา: Patterson and Hennessy (2016) . . . . .	48

2.11	การเก็บข้อมูลในหน่วยความจำในรูปของรหัสแอสกีด้วยตัวแปรสตริงซึ่ง str ซึ่งเป็นตัวแปรชนิด char[10] str= "Hello!" ที่แสดง adres เริ่มต้น 0x50 หรือ $50_{16}$ ที่มา: Harris and Harris (2013) . . . . .	50
2.12	ตารางรหัส ASCII และ Unicode สำหรับตัวอักษรจำนวน $2^8 = 256$ ตัว ประกอบด้วยรหัสของตัวอักษรภาษาอังกฤษและภาษาไทย ที่มา: ascii-table.com . . . . .	51
3.1	วงรอบการเข้ามายังอินพุต-เอาต์พุต (Input-Output Loop) ระหว่าง ผู้ใช้ ฮาร์ดแวร์ และซอฟต์แวร์ (เกม) ที่มา: wikipedia.org . . . . .	55
3.2	ตำแหน่งของอุปกรณ์ต่างๆ บนบอร์ด Pi3/Pi4 ที่มา: raspberryhome.net . . . . .	56
3.3	บล็อกໄดอะแกรมของชิป AllWinner A64 ที่มีชิปยู ARM Cortex A53/A72 คล้ายกับชิป BCM2837/BCM2711 บนบอร์ด Raspberry Pi3/Pi4 ที่มา: Allwinner Technology (2015a) และ Allwinner Technology (2015b) . . . . .	58
3.4	หน่วยความจำ SDRAM ด้านล่างของบอร์ด Pi3/Pi4 โนเดล B ที่มา: robo-dyne.com . . . . .	59
3.5	ตำแหน่งและรหัสประจำปุ่ม (Scan Code หรือ Position Code) บนคีย์บอร์ดชนิด 106 ปุ่ม ที่มา: ps-2.kev009.com . . . . .	60
3.6	โครงสร้างภายในแม่สัมภាន Optical ประกอบด้วยหลอด LED ส่องแสงกระแทกกับพื้นผิวด้านล่างแล้วสะท้อนกลับมายังตัวรับแสงชนิด CMOS ที่มา: www.pinterest.com . . . . .	61
3.7	โครงสร้างจอแสดงผลชนิด Color LCD ประกอบด้วยแหล่งกำเนิดแสง (Light Source) ปล่อยแสงที่ลุ้นต่างๆ ปรากฏแก่สายตาผู้ใช้ ที่มา: techterms.com . . . . .	62
3.8	โครงสร้างของการ์ดหน่วยความจำ SD ชนิด SDHC ความจุ 16 กิกะไบต์ (GB) ประกอบด้วยชิปหน่วยความจำแฟลช วงจรควบคุม และวงจรเชื่อมต่อ ที่มา: wikipedia.org . . . . .	63
3.9	ตัวอย่างซอฟต์แวร์ภาษา C ฟังก์ชันหลักชื่อ main() เมื่อทำงานเสร็จสิ้นจะรีเทิร์นค่า 0 ที่มา: zentut.com . . . . .	65
3.10	ขั้นตอนการพัฒนาซอฟต์แวร์จากภาษา C/C++ ให้เป็นโปรแกรมประยุกต์ (Application Program) หรือย่อว่า แอป ที่มา: slideplayer.com . . . . .	66
3.11	ตัวอย่างการลิงก์ (Link) หรือรวมไฟล์อีบเจกต์หลัก ไฟล์อีบเจกต์เสริม และไฟล์ไลบรารีเข้าด้วยกันเป็นไฟล์โปรแกรมหรือแอปพลิเคชัน ที่มา: google.com . . . . .	70
3.12	ตัวอย่างการแปลงจากคำสั่งแอสเซมบลีของ ARM ทางด้านขวาเป็นคำสั่งภาษาเครื่อง ทางด้านซ้าย ที่มา: Harris and Harris (2013) . . . . .	71
3.13	โครงสร้างของไดเรกทอรี (Directory) สำคัญ ในระบบปฏิบัติการลินุกซ์ ที่มา: freedompenguin.com . . . . .	74
3.14	โครงสร้างไฟล์ชนิด ELF สำหรับเก็บชุดคำสั่งภาษาเครื่องและข้อมูลเป็นเลขฐานสองอยู่ในอุปกรณ์เก็บรักษาข้อมูล ที่มา: wikipedia.org . . . . .	76
3.15	โครงสร้างของคอมพิวเตอร์ประกอบด้วยฮาร์ดแวร์ชั้นล่างสุด และหน่วยความจำเวอร์ชวลเมโมรี (Virtual memory) ภายใต้ระบบลินุกซ์แบ่งเป็นพื้นที่ เรียกว่า เคอร์เนล สเปซ (ระบบปฏิบัติการ) และ ยูสเซอร์สเปซ (ซอฟต์แวร์ประยุกต์) ที่มา: linux-india.org . . . . .	77
3.16	การจัดวางคำสั่งภาษาเครื่องในเทิกซ์ เชกเมนต์ และ ข้อมูลในดาต้า เชกเมนต์ จากไฟล์โปรแกรมรูปแบบ ELF ไปยังเวอร์ชวลเม莫รี บริเวณยูสเซอร์สเปซ (User Space) ที่มา: wordpress.com . . . . .	79
3.17	(a) ขบวนการอ่าน (Load) ข้อมูลจากหน่วยความจำหลักที่ตำแหน่ง $111_2$ (b) ขบวนการเขียน (Store) ข้อมูลในหน่วยความจำหลักที่ตำแหน่ง $101_2$ . . . . .	80

3.18	โครงสร้างของลิ้นก์เครื่องเนลในเวอร์ชวลเมโมรี ประกอบด้วย I/O Subsystem, Memory Management Subsystem และ Process Management Subsystem ที่มา: wikipedia.org . . . . .	81
3.19	หลักการของ Memory Mapped File ที่มา: rice.edu . . . . .	82
4.1	โครงสร้างภายในแกนประมวลผล 0 ทั้งหมด 4 แกนประมวลผลของ ชิปปี้ Cortex A53/A72 ออกแบบโดยบริษัท ARM ที่มา: tomshardware.com . . . . .	85
4.2	โครงสร้างเชิงตรรกะของแกนประมวลผลประกอบด้วยรีจิสเตอร์ R0-R15 แคชต่างๆ และหน่วยความจำหลัก, (VA: Virtual Address) . . . . .	87
4.3	คำสั่งภาษาเครื่องที่ถูกอ่าน (Load) เข้าสู่หน่วยความจำและแสดงในรูปของภาษาแอสเซมบลีบนโปรแกรมซิมูเลเตอร์ (Simulator) ในบริเวณเท็กซ์เช็คเมนต์ ที่มา: arm.com รีจิสเตอร์สำหรับเก็บสถานะของชิปปี้ ณ ปัจจุบัน (Current Program Status Register: CPSR) พร้อมรายละเอียดทั้ง 32 บิต ที่มา: arm.com . . . . .	90
4.4	คำสั่ง r3 = r4 + (r2 « 2) โดยหากค่า r4 กับค่า r2 หลังจากเลื่อนบิตไปทางซ้ายจำนวน 2 บิตแล้วจึงเก็บผลลัพธ์ใน r3 ที่มา: CodeMachine.com หมายเหตุ ค่าที่ยังเก็บอยู่ในรีจิสเตอร์ r2 จะไม่เปลี่ยนแปลง . . . . .	96
4.5	ไฟล์ชาร์ตการทำงานของประโภค IF . . . . .	97
4.6	ไฟล์ชาร์ตการทำงานของประโภค IF-ELSE . . . . .	101
4.7	ไฟล์ชาร์ตการทำงานของประโภควนรอบชนิด FOR . . . . .	103
4.8	ไฟล์ชาร์ตการทำงานของประโภควนรอบชนิด WHILE . . . . .	106
4.9	ไฟล์ชาร์ตการทำงานของประโภควนรอบชนิด DO-WHILE . . . . .	108
4.10	ไฟล์ชาร์ตการทำงานของประโภคเรียกใช้ฟังก์ชัน sum( a, b ) ในตัวอย่างที่ 4.8.1 . . . . .	110
4.11	ไฟล์ชาร์ตการทำงานของประโภคเรียกใช้ฟังก์ชัน main ในหน่วยความจำล้ายกับตัวอย่างที่ 4.8.2 หมายเหตุ ... หมายถึงประโภคคำสั่งอื่นๆ ที่ไม่เกี่ยวข้องกับตัวอย่าง . . . . .	111
4.12	การทำงานของคำสั่ง BL myFunction เมื่อมีการเรียกใช้ 2 ครั้งจากฟังก์ชัน main ใน . . . . .	113
4.13	การทำงานของคำสั่ง BL myFunction เมื่อมีการเรียกใช้ 2 ครั้งจากฟังก์ชัน main ในหน่วยความจำล้ายกับตัวอย่างที่ 4.8.2 หมายเหตุ ... หมายถึงประโภคคำสั่งอื่นๆ ที่ไม่เกี่ยวข้องกับตัวอย่าง . . . . .	115
4.14	การควบรวมชุดคำสั่งภาษาแอสเซมบลีของ ARM ตั้งแต่เวอร์ชัน 5 ถึงเวอร์ชัน 8A โดยเวอร์ชันใหม่จะรวมคำสั่งของเวอร์ชันเก่าเพื่อรองรับการทำงานซอฟต์แวร์เดิม ที่มา: arm.com . . . . .	118
5.1	ลำดับขั้นของหน่วยความจำชนิดต่างๆ สำหรับชิป BCM2837/BCM2711, (VA: Virtual Address) . . . . .	122
5.2	การแมป (Map) เวอร์ชูลแอดдрес (Virtual Address) ขนาด 3 กิกิไบต์ (GiB) เป็นแอดเดรสกายภาพ (Physical Address) ขนาด 64 เมบิไบต์ (MiB) ตามหลักการเวอร์ชูลเมโมรีชนิดเพจ (Paging Virtual Memory) หมายเหตุ เส้นประ คือ รอยต่อระหว่าง เพจของเวอร์ชูลเมโมรี . . . . .	125
5.3	การแปลงหมายเลขเพจเวอร์ชูล (Virtual Page Number) เป็นหมายเลขเพจกายภาพ (Physical Page Number) ที่มา: slideplayer.com . . . . .	126
5.4	โครงสร้างของเวอร์ชูลเมโมรีชนิดเพจของบอร์ด Pi3/Pi4 โนเบล B ซึ่งใช้ชิปประภูมิ BCM283x, x=5, 6, 7 หมายเหตุ ขนาดของรูปไม่เป็นไปตามพื้นที่ตามความเป็นจริง, MMU: Memory Management Unit ที่มา: Broadcom (2012) . . . . .	127

5.5	โครงสร้างของชิป ARM Cortex A7 ประกอบด้วย TLB และแคชหลาຍระดับเพื่อรับการทำงานของวอร์ชัลเม莫รีชนิดเพจ ที่มา: ARM (2011) และ arm.com . . . . .	128
5.6	โครงสร้างเชิงตรรกะเข้ามายอย่างห่วงรีจิสเตอร์ แคชลำดับที่ 1 และ 2 ภายในชิป BCM2837/BCM2711 และหน่วยความจำหลักภายนอก หมายเหตุ VA: Virtual Address . . . . .	130
5.7	โฟล์ชาร์ตการทำงานของแคชหนึ่งลำดับขั้นสำหรับการเฟทซ์คำสั่งและสำหรับการอ่าน/เขียนข้อมูล . . . . .	131
5.8	การทำงานของแคชคำสั่ง (Instruction Cache) ชนิด Direct Map เมื่อซีพียูเฟทซ์คำสั่งที่หน่วยความจำภายในภาพแอดเดรส (a) แคชคำสั่งยังไม่มีคำสั่งใดๆ เลย (b) PC ออฟเซตเท่ากับ $000_{16}$ , (c) $004_{16}$ , (d) $008_{16}$ . . . . .	133
5.9	การทำงานของแคชคำสั่ง (Instruction Cache) ชนิด Direct Map เมื่อซีพียูเฟทซ์คำสั่งที่ PC ออฟเซตเท่ากับ (e) $030_{16}$ , (f) $034_{16}$ , (g) $00C_{16}$ , (h) $010_{16}$ . . . . .	134
5.10	การทำงานของแคชลำดับที่ 1 แคชคำสั่ง (Instruction Cache) ชนิด 2-Way Set Associative เมื่อ PC เฟทซ์คำสั่งที่หน่วยความจำภายในภาพแอดเดรส (a) แคชคำสั่งยังไม่มีคำสั่งใดๆ เลย (b) PC ออฟเซตเท่ากับ $000_{16}$ , (c) $004_{16}$ , (d) $008_{16}$ . . . . .	135
5.11	การทำงานของแคชลำดับที่ 1 แคชคำสั่ง (Instruction Cache) ชนิด 2-Way Set Associative เมื่อ PC เฟทซ์คำสั่งที่ PC ออฟเซตเท่ากับ (e) $030_{16}$ , (f) $034_{16}$ , (g) $00C_{16}$ , (h) $010_{16}$ . . . . .	136
5.12	ตัวถังแบบ TSOP ของชิปหน่วยความจำชนิดสแตติกแรม Cypress 62256 ที่มา: Cypress Semiconductor (2002) . . . . .	138
5.13	โครงสร้างของหน่วยความจำชนิดสแตติก Cypress 62256 ที่มา: Cypress Semiconductor (2002) . . . . .	139
5.14	ໄດ້ອະແກມเวลา (Timing Diagram) สำหรับการอ่าน (Read) ข้อมูลจากหน่วยความจำชนิดสแตติกแรม ที่มา: Cypress Semiconductor (2002) . . . . .	140
5.15	ໄດ້ອະແກມเวลา (Timing Diagram) สำหรับการเขียน (Write) ข้อมูลในหน่วยความจำชนิดสแตติกแรม ที่มา: Cypress Semiconductor (2002) . . . . .	141
5.16	รูปถ่ายด้านบน (Top View) ด้านล่าง (Bottom View) ด้านข้าง (Side View) และภาพตัดขวาง (Cross Section) ของชิป SDRAM โดยใช้เทคโนโลยี TSV (Through Silicon Via) ผู้ผลิตบริษัท Elpida ที่มา: electroiq.com . . . . .	142
5.17	บล็อกไอອะແກມภายในชิปหน่วยความจำ SDRAM ชนิด DDR2 ประกอบด้วยダイ (Die) 2 ชิ้น แต่ละชิ้นมีบัสข้อมูลขนาด 16 บิตเรียงกันเป็น 32 บิต ที่มา: Micron Technology (2014) . . . . .	143
5.18	ໄດ້ອະແກມเวลา (Timing Diagram) สำหรับการอ่านต่อเนื่องของหน่วยความจำ SDRAM รุ่น Elpida B8132B4PB-8D-F คำสั่ง Burst READ โดย RL (Read Latency) เท่ากับ 5 ไซเกล BL (Burst Length) เท่ากับ 4 ตำแหน่ง หมายเหตุ รูปมีการตัดต่อเพื่อเน้นบริเวณที่สำคัญ, NOP=No Operation ที่มา: Micron Technology (2014) . . . . .	145
5.19	ໄດ້ອະແກມเวลา (Timing Diagram) สำหรับการเขียนต่อเนื่องของหน่วยความจำ Elpida รุ่น B8132B4PB-8D-F ตามคำสั่ง Burst WRITE โดย WL (Write Latency) = 1 ไซเกล, BL (Burst Length) = 4 ตำแหน่ง หมายเหตุ: รูปมีการตัดต่อเพื่อเน้นบริเวณที่สำคัญ ที่มา: Micron Technology (2014) . . . . .	146

6.1	การเชื่อมต่ออุปกรณ์ต่างๆ บนบอร์ด Pi3 โดยมีชิป BCM2837 เป็นศูนย์กลาง ที่มา: xdevs.com . . . . .	150
6.2	ภาพความละเอียด 720 จุด x480 เส้นที่ผู้ใช้มองเห็น แบ่งเป็นการส่งแพ็กเก็ตข้อมูลจุดภาพและแพ็กเก็ตควบคุมทางช่องสัญญาณ TMDS ภายใต้สัญญาณ HDMI ในช่วงเวลาต่างๆ ที่มา: freebsd.org . . . . .	151
6.3	หัวเชื่อมต่อ HDMI ชนิด Female ประกอบด้วยขาสัญญาณทั้งหมด 19 ขา ที่มา: wikipedia.org . . . . .	152
6.4	จอแสดงผลสำหรับเชื่อมต่อระหว่างบอร์ด Pi3/Pi4 ด้วยสัญญาณการแสดงผลแบบอนุกรม (Display Serial Interface) ที่มา: element14.com . . . . .	154
6.5	เบรียบเทียบสัญญาณ DSI ชนิดหนึ่งเลน (Single Lane) และชนิด 4 เลน ที่มา: wikipedia.org . . . . .	155
6.6	การเชื่อมต่อระหว่างบอร์ด Pi3/Pi4 และกล้องขนาดเล็กด้วยสัญญาณกล้องแบบอนุกรม (Camera Serial Interface) ที่มา: element14.com . . . . .	156
6.7	สัญญาณดิจิทัล PCM (Pulse Code Modulation) ความละเอียด 16 ระดับสูม (Sampling) จากสัญญาณอนาล็อกรูปคลื่นไอน์ (Sine Wave) ด้วยความถี่สูง 26 เท่าของความถี่สูงสุด ( $f_s=26f_{max}$ ) ที่มา: wolfcrow.com . . . . .	158
6.8	มาตรฐาน PCM ประกอบด้วย วงจรเชื่อมต่อกับซีพียูผ่านบัส APB, หน่วยความจำบัฟเฟอร์สำหรับส่งและรับข้อมูลเสียงดิจิทัล และวงจรเชื่อมต่อชนิด I <sup>2</sup> S ที่มา: Broadcom (2012) . . . . .	159
6.9	แจ็คขนาด 3.5 มม. (กลาง) ชนิด 4 ขั้วสำหรับเสียบกับบอร์ด Pi3/Pi4 (ขวา) ส่งสัญญาณภาพไปยังแจ็ค RCA (เหลือง) และสัญญาณเสียงไปยังแจ็ค RCA (แดงและขาว) ที่มา: stackexchange.com . . . . .	160
6.10	หัวเชื่อมต่อ USB ชนิด A (บน ฝั่งโฮสท์) และ B (ล่าง ฝั่งอุปกรณ์) ประกอบด้วยสัญญาณ 4 เส้น กราวน์ด (0 โวลต์) (GND) Data+ Data- และไฟกระแสตรง 5 โวลต์ ที่มา: quora.com . . . . .	161
6.11	โครงสร้างของชิป LAN 9514 ภายในประกอบด้วยวงจร USB Hub และวงจร Ethernet ที่มา: Microchip Technology (2009) . . . . .	162
6.12	หัวเชื่อมต่อชนิด RJ45 (ขัยสุด) สำหรับการเชื่อมต่อเครือข่ายท้องถิ่น (Local Area Network) แบบมีสาย Ethernet ที่มา: cytron.io . . . . .	163
6.13	การเข้าปลายสาย RJ45 ทั้งสองด้านสำหรับการเชื่อมต่อระหว่างเครื่องคอมพิวเตอร์และอุปกรณ์สวิตช์ (Switch) ตามมาตรฐาน TIA T568B ที่มา: blogspot.com . . . . .	163
6.14	รูปถ่ายด้านล่างของบอร์ด Pi3 และรูปขยายของชิป BCM43438 สำหรับเชื่อมต่อเครือข่ายไร้สาย WiFi และเครือข่ายไร้สายบลูทูธ (Bluetooth) ที่มา: stackexchange.com . . . . .	164
6.15	บล็อกโดยแกรมภายในชิป BCM43438 ประกอบด้วยขาสัญญาณเชื่อมต่อเสาอากาศ และขาเชื่อมต่อกับไมโครคอนโทรลเลอร์ ที่มา: Cypress Semiconductor (2017) . . . . .	165
6.16	การเชื่อมโยงระหว่าง เวอร์ชวลเอดเดรส (ขวา) แอดเดรสภายนอก (กลาง) และ VC/CPU บัสเอดเดรส (VC/CPU Bus Address) ที่มา: Broadcom (2012) หมายเหตุ VC: VideoCore . . . . .	166
6.17	โครงสร้างภายในขา GPIO สำหรับชิปประภูมิเดียวกับ BCM2835 ที่มา: Broadcom (2012) . . . . .	170

6.18	โดยละเอียดของโปรแกรมที่ต้องการเชื่อมต่อกับอุปกรณ์อินพุต/เอาต์พุตทำให้เกิด อินเทอร์รัปท์เกิดขึ้นเป็นระยะๆ ที่มา: virtualirfan.com . . . . .	175
6.19	โครงสร้างส่วนหนึ่งภายในชิป BCM283x แสดงการเชื่อมต่อแกนประมวลผลต่างๆ กับ วงจร Generic Interrupt Controller (GIC) ผ่าน ARM Advanced eXtensible Inter- face (AXI) หมายเหตุ APB: ARM Peripheral Bus ที่มา: arm.com . . . . .	177
6.20	โดยละเอียดการทำงานของ Generic Interrupt Controller (GIC) เพื่อตอบสนอง ต่อสัญญาณร้องขออินเทอร์รัปท์ที่มีความสำคัญไม่เท่ากัน ที่มา: Ltd. (2017) . . . . .	178
6.21	ลำดับการทำงานของ DMA Controller (DMAC) เพื่อควบคุมการอ่านข้อมูลจากการด หน่วยความจำ SD ไปบันทึกในหน่วยความจำหลัก (Main Memory) . . . . .	179
6.22	โครงสร้างส่วนหนึ่งภายในชิป BCM283x/BCM2711 แสดงการเชื่อมต่อแกนประมวล ผลต่างๆ กับวงจร DMAC (DMA Controller) ผ่าน ARM Advanced eXtensible Interface (AXI) และ AXI-APB Bridge ที่มา: arm.com . . . . .	180
6.23	การแปลงไฟกระแทก 5 โวลต์เป็นไฟกระแทกด้วยชิป PAM 2306 DC-DC Cover- tor คู่ กำหนดค่าเอาต์พุตด้วยค่าตัวเหนี่ยวนำ L <sub>1</sub> และ L <sub>2</sub> หน่วยเป็น ไมโครเอนรี ที่มา: Diodes (2012) . . . . .	181
7.1	โครงสร้างของระบบไฟล์บนอุปกรณ์เก็บรักษาข้อมูลในระบบปฏิบัติการตระกูลยูนิกซ์ ที่มา: Demblon and Spitzner (2004) . . . . .	187
7.2	โครงสร้างของไอโหนดหนึ่งตัวและการเชื่อมโยงกับบล็อกข้อมูลของไฟล์หนึ่งไฟล์ ที่มา: Anderson and Dahlin (2012) . . . . .	189
7.3	ชิปหน่วยความจำแฟลช NAND ผลิตโดยบริษัท Micron Technology โดยใช้ตัวถังชนิด TSOP (Thin Small Outline Package) จำนวน 48 ขา ที่มา: Micron Technology (2004) . . . . .	192
7.4	โครงสร้างภายในชิปหน่วยความจำแฟลช NAND ที่มา: Micron Technology (2004) .	193
7.5	โดยละเอียดเวลาของการอ่านข้อมูลแบบเพจ (Page Read) ของหน่วยความจำแฟลช NAND ที่มา: Micron Technology (2004) . . . . .	195
7.6	โดยละเอียดเวลาของการเขียนข้อมูล (Program Data) และอ่านสถานะ (Read Status) ของหน่วยความจำแฟลช NAND ที่มา: Micron Technology (2004) . . . . .	196
7.7	โครงสร้างภายในการ์ดหน่วยความจำ SD ที่มา: SanDisk Corporation (2003) . . .	198
7.8	โดยละเอียดเวลาของการอ่านข้อมูลจำนวนหลายบล็อกจากการ์ดหน่วยความจำ SD ที่มา: SanDisk Corporation (2003) . . . . .	200
7.9	โดยละเอียดเวลาของการเขียนข้อมูลจำนวนหลายบล็อกจากการ์ดหน่วยความจำ SD ที่มา: SanDisk Corporation (2003) . . . . .	201
7.10	แผ่นวงจรพิมพ์ภายในอุปกรณ์ SSD ชนิด SATA III ประกอบด้วยชิพหน่วยความจำแฟลช ไอนามิกแรม คอนโทรลเลอร์สำหรับควบคุม ที่มา: thatoldnews.site . . . . .	202
7.11	บล็อก โดยละเอียดภายใน SSD ประกอบด้วย ส่วน เชื่อม ต่อ กับ เครื่อง ไมโคร คอนโทรลเลอร์ บัฟเฟอร์ และหน่วยความจำแฟลช ที่มา: codecapsule.com . . .	203
7.12	โครงสร้างและองค์ประกอบของอุปกรณ์ฮาร์ดดิสก์ไดรฟ์ (HDD) ชนิด Serial ATA ที่มา: blogspot.com . . . . .	204
7.13	โครงสร้างของฮาร์ดดิสก์ไดรฟ์แบ่งเป็นไซลินเดอร์ แทร็ก และเซกเตอร์ ที่มา: com- putableminds.com . . . . .	205

8.1	ประสิทธิภาพของการประมวลผลด้วยชิปซีพียูชนิดมัลติคอร์ ที่มา: royalsocietypublishing.org, John (2020) . . . . .	209
8.2	โครงสร้างภายในของชิป ARM ตามเทคโนโลยี big.LITTLE เชื่อมระหว่างแกนประมวลผลด้วย Cache Coherent Interconnect (CCI) ที่มา: arm.com . . . . .	212
8.3	โครงสร้างของชูเปอร์คอมพิวเตอร์ Fugaku ประกอบด้วยตู้เร็กจำนวน 414 ตู้ๆ ละ 384 โนนด แต่ละโนนดมี Fujitsu A64FX จำนวน 1 ชิปซึ่งภายในประกอบด้วยซีพียู ARM จำนวน 48 แกนประมวลผล ที่มา: pcmag.com . . . . .	213
8.4	ภาพถ่ายด้วยและบล็อกไโดอะแกรมของชิป Fujitsu A64FX ประกอบด้วยซีพียู ARM จำนวน 48 แกนประมวลผล ที่มา: fujitsu.com . . . . .	213
8.5	การคำนวณแบบขนานโดยอาศัยความขนาดของข้อมูล (Data Parallelism) และความขนาดของงานย่อย (Task Parallelism) ที่มา: manning.com . . . . .	215
8.6	ตัวอย่างซอฟต์แวร์ที่แสดงการทำงานแบบขนาน และเรตประมวลผล T0, T1 และ T2 พิมพ์ข้อความ Hello World ตามด้วยหมายเลขเรตประจำตัว โดยใช้ไลบรารี OpenMP แก้ไขจากที่มา: slideplayer.com . . . . .	216
8.7	การทำงานของอัลกอริธึม MergeSort ตามหลักการแบ่งและยืดครองสามารถนำมาตัดแปลงเป็นแบบอนุกรมและแบบขนาดได้ในตัวอย่างที่ 8.3.2 ที่มา: slideshare.net . . . . .	219
A.1	หน้าเว็บสำหรับแปลงเลขจำนวนเต็มฐานสองเป็นฐานสิบเป็นฐานสิบเป็นฐานสองหลายนิด 225	
A.2	กรอกเลข -123 ลงในกล่องข้อความ Decimal เพื่อให้โปรแกรมแปลงเลขจำนวนเต็ม -123 เป็นเลขจำนวนเต็มฐานสองมีเครื่องหมายชนิด 2's Complement . . . . .	226
A.3	ผลลัพธ์การแปลงเลข -123 เป็นเลขจำนวนเต็มฐานสองมีเครื่องหมายชนิด 2's Complement . . . . .	226
A.4	การแปลงเลขฐานสองมีเครื่องหมายชนิด 2's Complement 11111111 หรือเท่ากับฐานสิบหก 0xFF . . . . .	227
A.5	ผลลัพธ์การแปลงเลขฐานสองมีเครื่องหมายชนิด 2's Complement 11111111 หรือเท่ากับฐานสิบหก 0xFF . . . . .	227
A.6	ผลลัพธ์จากการแปลงเลข 123.0 ให้เป็นเลขทศนิยมฐานสองชนิด Single Precision . . . . .	232
A.7	ผลลัพธ์จากการแปลงเลข -123.0 ให้เป็นเลขทศนิยมฐานสองตามมาตรฐาน IEEE754 ชนิด Single Precision . . . . .	233
A.8	เมนูด้านล่างสุดของหน้าเว็บ เพื่อเลือกเลขทศนิยมฐานสองชนิด IEEE754 Single Precision (Binary32) และ Double Precision (Binary64) . . . . .	234
A.9	ผลลัพธ์จากการบวกเลข -123.0+123.0 ให้เป็นเลขทศนิยมฐานสองชนิด IEEE754 Single Precision . . . . .	234
A.10	ผลลัพธ์จากการคูณเลข -123.0 × 123.0 ให้เป็นเลขทศนิยมฐานสองชนิด IEEE754 Single Precision . . . . .	235
A.11	เว็บสำหรับการตอบคำถามเพื่อสร้างเลขหรือแปลงเลขฐานสิบด้วยมาตรฐาน IEEE754 Single Precision การกดเลือกคือทำให้บิตนั้นเท่ากับ '1' . . . . .	236
A.12	ผลลัพธ์จากการกรอกและแปลงตัวอักษร ໄ ທ ຍ ก ຂ ຄ a b c เป็นรหัสต่างๆ . . . . .	239
B.1	รูปแสดงรายการอุปกรณ์สำหรับประกอบบอร์ด ที่มา: rs-online.com . . . . .	241
B.2	บอร์ด Pi 3 เมื่อติดอีทซิงก์เรียบร้อยแล้ว . . . . .	242

B.3	แผ่นพลาสติก 5 ชิ้น สำหรับประกอบเป็นกล่องของบอร์ด Pi3 . . . . .	243
B.4	บอร์ด Pi ที่มีกล่องประกอบเรียบร้อยแล้ว . . . . .	244
B.5	บอร์ด Pi เมื่อประกอบขาเชื่อมขยาย 2x20 . . . . .	244
B.6	การเชื่อมต่อคีย์บอร์ดและมาสเตอร์กับช่องเสียบสาย USB บนบอร์ด Pi ให้เรียบร้อย . . . . .	245
B.7	การเชื่อมต่อไฟเลี้ยงจากอแดปเตอร์ทางหัวไมโคร USB กับบอร์ด Pi . . . . .	245
B.8	ภาพสีรุ้งบนจออันเกิดจากบอร์ด Pi ทำงานเป็นปกติ . . . . .	246
C.1	หน้าต่างดาวน์โหลดไฟล์โปรแกรม Raspberry Pi Imager สำหรับติดตั้งระบบปฏิบัติการ Raspberry Pi OS ในการ์ดหน่วยความจำไมโคร SD . . . . .	248
C.2	หน้าต่างของโปรแกรม Raspberry Pi Imager . . . . .	248
C.3	เมนูตัวเลือกใต้เมนู CHOOSE OS . . . . .	249
C.4	เมนูตัวเลือกอื่นๆ ใต้เมนู Raspberry Pi OS (other) . . . . .	249
C.5	เมนูตัวเลือกอื่นๆ ใต้เมนู Other general purpose OS . . . . .	250
C.6	หน้าต่าง Imager . . . . .	250
C.7	ปุ่ม WRITE ในหน้าต่าง Raspberry Pi Imager . . . . .	250
C.8	หน้าต่างเตือนผู้ใช้ที่ต้องการเขียนทับการ์ดหน่วยความจำ SD . . . . .	251
C.9	หน้าต่าง Raspberry Pi Imager ทยอยเขียนข้อมูลภายในการ์ด . . . . .	251
C.10	หน้าต่าง Raspberry Pi Imager วนสอบ (Verify) ข้อมูลภายในการ์ด . . . . .	251
C.11	ปุ่ม CONTINUE ในหน้าต่าง Raspberry Pi Imager เมื่อติดตั้งสำเร็จ . . . . .	252
C.12	ผลการ์ดเข้าไปในสล็อตบันบอร์ด Pi โดยหมายบอร์ดขึ้นมา โปรดสังเกตการ์ดหน่วยความจำจะต้องมีลักษณะดังรูป . . . . .	252
C.13	หน้าต่าง Welcome ของระบบปฏิบัติการ Raspberry Pi OS . . . . .	253
C.14	หน้าต่างตั้งค่าประเทศ โชนเวลา และภาษาในการใช้งานเมนูเป็นภาษาอังกฤษ . . . . .	253
C.15	หน้าต่างสำหรับการเปลี่ยนรหัสผ่านใหม่ของ username ชื่อ pi โดยจะต้องกรอกรหัสผ่านใหม่ซ้ำจำนวน 2 ครั้ง . . . . .	254
C.16	ตัวอย่างรายชื่อสัญญาณ WiFi รอบๆ ที่บอร์ด Pi มองเห็น ซึ่งจะแตกต่างกับของผู้อ่าน . . . . .	254
C.17	หน้าต่างสำหรับกรอกพาสเวิร์ดของสัญญาณ WiFi ที่ต้องการเชื่อมต่อ . . . . .	255
C.18	เว็บเพจเริ่มต้นของเว็บไซต์ www.raspberrypi.org . . . . .	255
C.19	หน้าต่างสำหรับเมนู Shutdown เพื่อให้ผู้ใช้ ขัตดาวน์ รีบูต (Reboot) หรือล็อกเอาท์ (Logout) . . . . .	256
D.1	หน้าต่างของไฟล์เมเนจเจอร์ (File Manager) ขณะที่เปิดได้เรียบร้อยชื่อ /usr . . . . .	258
D.2	รูปไอคอนของโปรแกรม Terminal . . . . .	259
D.3	หน้าต่างของโปรแกรม Terminal ซึ่งสามารถปรับแต่งสีพื้นและสีของตัวอักษรได้ . . . . .	259
D.4	หน้าต่างปรับแต่งสีพื้น (Background) . . . . .	260
D.5	หน้าต่างปรับแต่งสีตัวอักษร (Foreground) . . . . .	260
E.1	หน้าต่างเลือกชนิดโปรแกรมที่จะพัฒนาเป็นชนิด "Console application" . . . . .	266
E.2	หน้าต่างเลือกภาษา C หรือ C++ สำหรับโปรแกรมที่จะพัฒนา . . . . .	267
E.3	การเลือกค่าฟิกกูเรชัน (Configuration) Debug สำหรับคอมไพล์เตอร์ GNU GCC ในโปรแกรม Lab5 . . . . .	267
E.4	การเปิดอ่านไฟล์ main.c ภายใต้โปรแกรม Lab5 ที่สร้างขึ้น . . . . .	268

F.1	การย้ายไฟล์ main.c ออกจากโปรเจคท์ . . . . .	276
F.2	การเพิ่มไฟล์ใหม่ลงในโปรเจคท์ . . . . .	276
F.3	หน้าต่างกดปุ่ม "Yes" เพื่อยืนยัน . . . . .	276
F.4	หน้าต่าง Save File ชื่อไฟล์ว่า main.s . . . . .	277
I.1	หน้าต่าง Screen Layout Editor สำหรับกำหนดค่าต่างๆ กับพอร์ตแสดงผล HDMI . . . . .	304
I.2	หน้าต่าง Set Resolution สำหรับกำหนดความละเอียดหน้าจอแสดงผลที่ต้องการ . . . . .	304
I.3	หน้าต่าง Reboot needed กดปุ่ม Yes เมื่อต้องการรีบูต ณ เวลานั้น . . . . .	304
I.4	หน้าต่าง Raspberry Pi Configuration . . . . .	305
I.5	แท็บ Performance หน้าต่าง Raspberry Pi Configuration . . . . .	305
I.6	หน้าต่างกำหนดขนาดหน่วยความจำสำหรับจีพียูที่ 16 MiB . . . . .	306
I.7	หน้าต่างโปรแกรม raspi-config สำหรับบอร์ด Pi . . . . .	307
I.8	เมนู System Options ในหน้าต่างโปรแกรม raspi-config สำหรับบอร์ด Pi . . . . .	307
I.9	เมนู Audio ภายในเมนู System Options ในหน้าต่างโปรแกรม raspi-config สำหรับบอร์ด Pi . . . . .	308
I.10	ผลลัพธ์ในหน้าต่างโปรแกรม speaker-test สำหรับบอร์ด Pi . . . . .	308
I.11	โปรแกรม ALSA Mixer สำหรับควบคุมระดับเสียงทั้งด้านอินพุต (Capture: F4) และเอ้าต์พุต (Playback: F3) บนบอร์ด Pi . . . . .	309
J.1	วงจรเชื่อมต่อหลอด LED กับบอร์ด Pi ในการทดลองที่ 10 เพื่อทดสอบว่าหลอด LED ทำงาน ที่มา: fritzing.org . . . . .	321
K.1	วงจรสวิตซ์ปุ่มกดสำหรับทดลองการเขียนโปรแกรมอินเทอร์รัฟต์ในการทดลองที่ 11 ที่มา: fritzing.org . . . . .	328
M.1	กราฟแสดงการใช้งานซีพียู (CPU Usage Monitor) ย้อนหลังและค่าสรุป ณ เวลาปัจจุบัน ที่มา: abload.de . . . . .	339

# บทที่ 1

## บทนำ (Introduction)

คอมพิวเตอร์ ประกอบด้วย ฮาร์ดแวร์ และซอฟต์แวร์ ต่างๆ มีประโยชน์ต่อเศรษฐกิจ การประกอบธุรกิจ อุตสาหกรรมการผลิต การศึกษา รวมถึงชีวิตประจำวันมากขึ้น สามารถใช้ประกอบการเรียนการสอน วิชาองค์ประกอบคอมพิวเตอร์ (Computer Organization) และ วิชาสถาปัตยกรรมคอมพิวเตอร์ (Computer Architecture) เป็นต้น โดยอาศัยบอร์ด Raspberry Pi3/Pi4 และ ARM Cortex A53/A72 เป็นกรณีศึกษา อีกทั้งใช้เป็นเอกสารประกอบการพัฒนาโปรแกรมเบื้องต้นโดยมีฮาร์ดแวร์เสริมความเข้าใจ

### 1.1 ชนิดของเครื่องคอมพิวเตอร์

ระบบคอมพิวเตอร์ทั่วไป สามารถแบ่งออกเป็น คอมพิวเตอร์ตั้งโต๊ะ (Desktop Computer) คอมพิวเตอร์ เครื่องฟาร์ม หรือ แม่ข่าย (Server Computer) ซูเปอร์คอมพิวเตอร์ (Supercomputer) คอมพิวเตอร์พกพา (Portable Computer) และคอมพิวเตอร์ฝังตัว (Embedded Computer) หรือในชื่อ Internet of Things (IoT)

#### 1.1.1 คอมพิวเตอร์ตั้งโต๊ะ (Desktop Computer)

คอมพิวเตอร์ตั้งโต๊ะสามารถใช้งานได้หลากหลายชีวันอยู่กับซอฟต์แวร์ที่นำมาติดตั้ง โดยซอฟต์แวร์ประยุกต์เหล่านี้ขึ้นตรงกับซอฟต์แวร์ระบบปฏิบัติการเป็นหลัก เช่น ระบบปฏิบัติการไมโครซอฟต์วินโดวส์ (Microsoft Windows) เวอร์ชันต่างๆ ระบบปฏิบัติการ MAC OS เวอร์ชัน 10.x ระบบปฏิบัติการลินุกซ์ (Linux) ซึ่งมีดิสทริบิวชัน (Distribution) เช่น Ubuntu SuSe RedHat เป็นต้น ตำราเล่มนี้จะอ้างอิงกับระบบปฏิบัติการ Raspberry Pi OS ซึ่งเป็นเวอร์ชันหนึ่งของระบบปฏิบัติการลินุกซ์สำหรับบอร์ด Raspberry Pi

#### 1.1.2 คอมพิวเตอร์พกพา (Portable Computers)

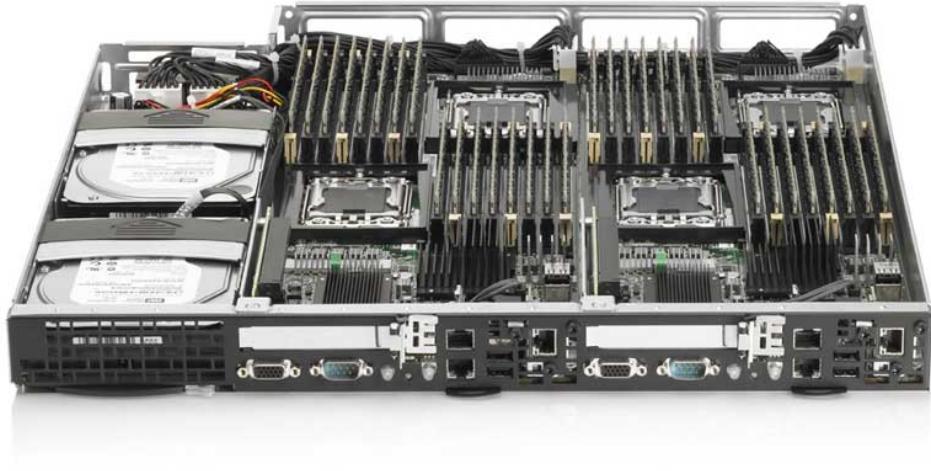
คอมพิวเตอร์พกพา มีขนาดเล็กและพกพาง่าย ใช้กำลังไฟจากแบตเตอรี่เป็นหลัก ได้แก่ คอมพิวเตอร์โน้ตบุ๊ค แท็บเล็ตคอมพิวเตอร์ และโทรศัพท์เคลื่อนที่สมาร์ตโฟน เป็นต้น ระบบปฏิบัติการที่ได้รับความนิยมสำหรับเครื่องเหล่านี้ ได้แก่ ระบบแอนดรอยด์ (Android) และระบบ Apple iOS ทั้งสองระบบนี้มีใช้บนโทรศัพท์สมาร์ตโฟน และแท็บเล็ต มากกว่า 90% ทั่วโลก ผู้ใช้สามารถติดตั้งแอนดรอยด์ (Android) ซึ่งพัฒนาต่อจากระบบปฏิบัติการลินุกซ์บนบอร์ด Pi3/Pi4 ได้เช่นกัน รายละเอียดเพิ่มเติมสามารถศึกษาได้จาก [howtoraspberrypi.com](http://howtoraspberrypi.com)

### 1.1.3 คอมพิวเตอร์ฝังตัว (Embedded Computer)

คอมพิวเตอร์ฝังตัวเป็นสมองกลที่ซ่อนอยู่ในระบบ คอมพิวเตอร์ชนิดนี้สามารถทำงานอยู่ใต้เงื่อนไขของกำลังไฟ/สมรรถนะ/ราคา ปัจจุบัน คอมพิวเตอร์ฝังตัวได้รับความนิยมเพื่อทำหน้าที่เป็นเซ็นเซอร์ (Sensor) และแขนกล (Actuator) สำหรับงานประเภทต่างๆ มากขึ้นและเชื่อมต่อกับเครือข่ายอินเทอร์เน็ตได้อย่างแพร่หลาย ทำให้มีสีอีกเรียกว่า Internet of Things หรือ IoT เช่น เครื่องบินไร้คนขับ (Unmanned Aerial Vehicle) โดรน (Drone) เป็นต้น

### 1.1.4 คอมพิวเตอร์เซิร์ฟเวอร์หรือแม่ข่าย (Server Computer)

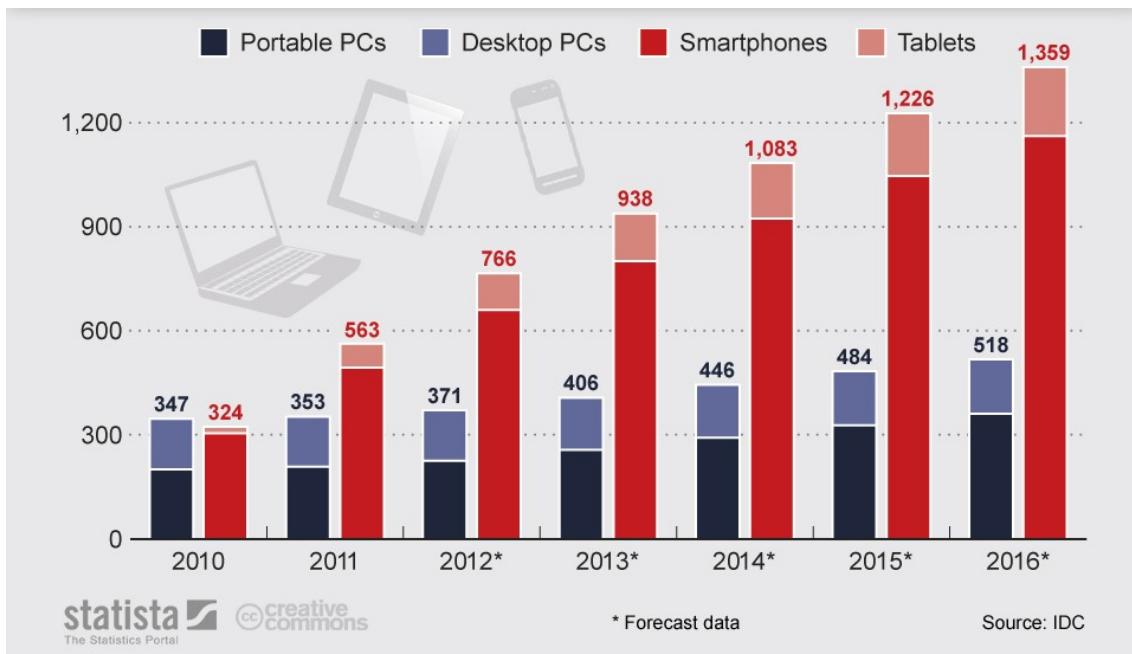
คอมพิวเตอร์เซิร์ฟเวอร์หรือเครื่องแม่ข่าย จะต้องติดตั้งระบบปฏิบัติการ เช่นเดียวกับเครื่องคอมพิวเตอร์ตั้งโต๊ะ ยิ่งไปกว่านั้นคอมพิวเตอร์เซิร์ฟเวอร์เหล่านี้จะต้องให้เปิดให้บริการเครื่องคอมพิวเตอร์ลูกข่าย (Client Computer) ผ่านทางระบบเครือข่ายอินเทอร์เน็ตตลอดเวลา ดังนั้น คอมพิวเตอร์แม่ข่ายจะมีความจุ สมรรถนะ และความเชื่อมั่นสูงเพื่อรับรองการใช้งาน จากผู้ใช้เครื่องคอมพิวเตอร์จำนวนมากซึ่งกระจายอยู่ทั่วโลก ระบบปฏิบัติการที่ได้รับความนิยมสำหรับเครื่องเหล่านี้ ได้แก่ ลินุกซ์ (Linux) และไมโครซอฟต์วินโดวส์ (Microsoft Windows)



รูปที่ 1.1: ตัวอย่างเครื่องคอมพิวเตอร์ชนิดเซิร์ฟเวอร์ จากบริษัท Hewlett Packard รุ่น ProLiant ที่มา: [datacenterknowledge.com](http://datacenterknowledge.com)

### 1.1.5 ซูเปอร์คอมพิวเตอร์ (Supercomputer)

คอมพิวเตอร์ที่มีสมรรถนะในการคำนวณสูงมาก ใช้งานด้านการประมวลผลข้อมูลขนาดใหญ่มาก (Big Data) ที่ได้จากการบันทึกของเครื่องเซิร์ฟเวอร์ และจากข้อมูลดิบจากการรวมของเครื่องคอมพิวเตอร์ฝังตัว เพื่องานประยุกต์สำคัญๆ เช่น การจำลองการทำงานของยา วัสดุ ไวน์ การพยากรณ์อากาศ การใช้ปัญญาประดิษฐ์ (Artificial Intelligence) การเรียนรู้ของเครื่องจักร (Machine Learning) เป็นต้น



รูปที่ 1.2: ปริมาณยอดขาย (ล้านหน่วย) เครื่องคอมพิวเตอร์ชนิดเดสก์ทอป โน้ตบุ๊ค สมาร์ตโฟน และแท็บเล็ตทั่วโลกในปี ค.ศ. 2010-2016 ที่มา: [statista.com](http://statista.com)

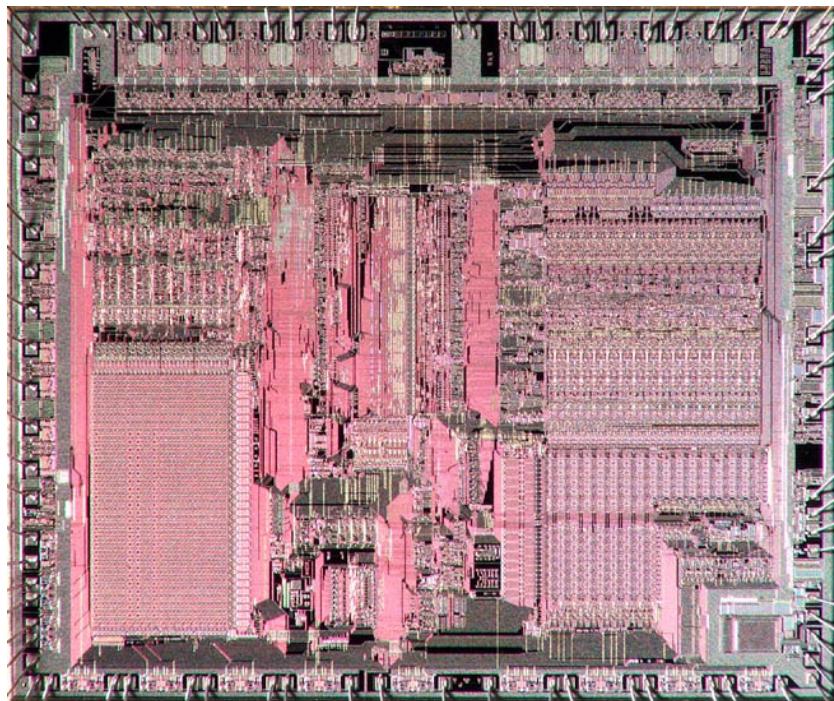
## 1.2 แนวโน้มของจำนวนอุปกรณ์คอมพิวเตอร์ชนิดต่างๆ

แนวโน้มของจำนวนอุปกรณ์คอมพิวเตอร์ กลุ่มคอมพิวเตอร์ส่วนบุคคลตั้งตือะ และ กลุ่มคอมพิวเตอร์พกพา ประเภทสมาร์ตโฟน และแท็บเล็ต ตั้งแต่ปี ค.ศ. 2005-2015 ตามกราฟในรูปที่ 1.2 ผู้อ่านจะเห็นได้ว่า กลุ่ม สมาร์ตโฟนและแท็บเล็ต มีจำนวนเพิ่มมากขึ้นแบบก้าวกระโดด ในขณะที่ กลุ่มพีซี (PC: Personal Computer) หรือคอมพิวเตอร์ตั้งตือะมีแนวโน้มลดลงไปเรื่อยๆ โดยสมาร์ตโฟนขนาดใหญ่ (Large Smartphone) จะได้รับ ความนิยมเพิ่มมากขึ้นเรื่อยๆ ในขณะที่ความนิยมสมาร์ตโฟนขนาดเล็ก (Small Smartphone) มีแนวโน้มลดลง แท็บเล็ตขนาดใหญ่ (Large Tablet) และแท็บเล็ตขนาดเล็ก (Small Tablet) มีแนวโน้มค่อนข้างคงที่

## 1.3 คอมพิวเตอร์บอร์ดเดียว (Single Board) ตระกูล Raspberry Pi

คอมพิวเตอร์บอร์ดเดียว (Single Board) ตระกูล Raspberry Pi ถูกออกแบบให้มีต้นทุนต่ำ ทำให้ราคาถูก เหมาะกับการศึกษาและงานที่มีความซับซ้อนน้อย โดยบอร์ดตัวแรกอาศัยชิป BCM2835 เป็นชิปหลักตั้งแต่ปี 2012 ในปี 2015 บอร์ด Raspberry Pi2 ใช้ชิป BCM2836 ปี 2016 บอร์ด Raspberry Pi3/Pi4 ใช้ชิป BCM2837 ล่าสุดปี 2019 บอร์ด Raspberry Pi4 ใช้ชิป BCM2711 โดยบริษัท Broadcom ประเทศสหรัฐอเมริกา เป็นผู้ผลิต ชิปทั้งหมดนี้ รูปที่ 1.3 คือ ภาพถ่ายダイ (Die) หรือแผ่นวงจรซิลิคอนของ BCM2835 ขนาดประมาณ 50-100 เท่าของชิปจริง โดยปราศจากแพ็คเกจพลาสติกสีดำห่อหุ้ม ชิป BCM2835 มีคุณลักษณะเฉพาะ (Specification) ดังนี้

- ARM1176JZF-S จำนวน 1 แกนประมวลผล (Core) ทำงานที่สัญญาณคล็อก (Clock) ความถี่ 700 เมกะเฮิรตซ์ (MHz) แคชลำดับที่ 1 (Level 1) ขนาด 16 KiB (KibiByte) แคชลำดับที่ 2 (Level 2) ย่อว่า L2 ขนาด 128 KiB (kibibyte)



รูปที่ 1.3: ภาพถ่ายชิลกอนดาย (Silicon Die) ของชิป BCM2835 บนบอร์ด Raspberry Pi ที่มา: [raspberrypi.org](http://raspberrypi.org)

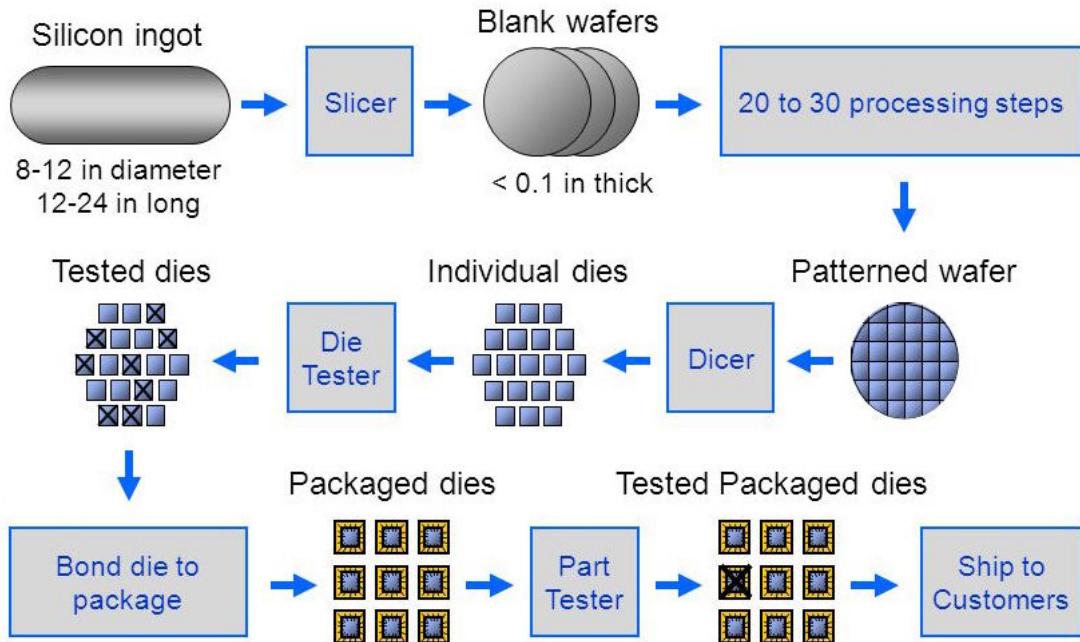
- หน่วยประมวลผลด้านกราฟิก หรือจีพียู (GPU) ชื่อ VideoCore IV ภายในชิปตัวเดียวกัน รายละเอียดเพิ่มเติม: [wikipedia](https://en.wikipedia.org/wiki/Raspberry_Pi)

บอร์ด Pi3/Pi4 ใช้เทคโนโลยีที่สูงขึ้น สมรรถนะและประสิทธิภาพเพิ่มขึ้น โดยมีจำนวน 4 แกนประมวลผล (Core) รายละเอียดเพิ่มเติมในบทที่ 3 บอร์ด Pi4 ใช้ซีพียูจำนวน 4 แกนประมวลผลเท่าเดิม แต่มีสมรรถนะและประสิทธิภาพสูงขึ้น ผู้อ่านสามารถค้นคว้ารายละเอียดทั้งหมดของบอร์ดตระกูลนี้ได้ที่ [github.com](https://github.com) อย่างเป็นทางการ

## 1.4 ขั้นตอนการผลิตชิป (Chip) หรือวงจรรวม (Integrated Circuit)

ภายในเครื่องคอมพิวเตอร์ชนิดต่างๆ มีชิป หรือไมโครชิป (Microchip) ซึ่งในอดีตอาจเรียกว่า วงจรรวม (Integrated Circuit ย่อว่า IC) หรือ ไอซี เมื่อผู้ออกแบบบرمาร์กจะต้องทำซึ่งประกอบด้วยวงจรโลจิคเกตและวงจรทรานซิสเตอร์เข้ามาเพิ่มเป็นจำนวนมากขึ้น คำว่า IC จึงเปลี่ยนเป็น VLSI (Very Large System Integration) และ ULSI (Ultra Large System Integration) ตามลำดับ ภายในชิปจึงมีวงจรดิจิทัลวงจรโลจิคเกตต่าๆ จำนวนมาก ทำให้มีจำนวนทรานซิสเตอร์มากตามและความสามารถที่หลากหลาย เช่น ชิปสามารถเชื่อมต่อกับจอแสดงผล อุปกรณ์และปุ่มต่างๆ ให้กล้ายเป็นเครื่องคอมพิวเตอร์ขนาดเล็กลงเรื่อยๆ จนปัจจุบันมีชื่อเรียกใหม่ว่า SoC (System on Chip) ผู้อ่านควรทำความรู้จักขั้นตอนการผลิตชิปเหล่านี้เบื้องต้น เพราะให้เข้าใจวิธีการของ SoC ต่อไปในอนาคต

ขั้นตอนการผลิตชิป (Chip) ตามรูปที่ 1.4 มีลักษณะคล้ายการทำแผ่นวงจรพิมพ์แต่มีความซับซ้อนและเทคโนโลยีในการผลิตสูง โดยเริ่มต้นจากการนำแท่งผลึกซิลิโคน (Silicon ingot) บริสุทธิ์มาทำความสะอาดแล้ว ขัดกลึงให้เป็นรูปทรงกระบอกขนาดเส้นผ่าศูนย์กลาง 10-12 นิว แล้วจึงแบ่งหรือสไลซ์ (Slice) แท่งผลึกนี้ให้เป็นแผ่นวงกลมที่บางมาก เรียกว่า เวเฟอร์เปล่า (Blank Wafer) เมื่อนำแผ่นเวเฟอร์เปล่าเหล่านี้ไปผ่านขั้นตอนการ



รูปที่ 1.4: ขั้นตอนการผลิตชิป (Chip) ภายในประกอบด้วยวงจรรวม ไมโครโปรดเซสเซอร์และอื่นๆ ที่มา: [slideplayer.com](http://slideplayer.com)

ปลูกถ่ายสาร (Doping) และอื่นๆ จำนวน 20-40 ขั้นตอน จนกลายเป็นแผ่นเวเฟอร์ที่มีลวดลาย (Patterned wafers) โดยเวเฟอร์หนึ่งแผ่นประกอบด้วย **ダイ** (Die) จำนวนหนึ่งกระจาดในลักษณะตารางดังรูปที่ 1.4 หลังจากทดสอบด้วยแต่ละตัวด้วยตัวทดสอบเวเฟอร์ (Wafer Tester) ด้วยตัวที่ไม่ผ่านการทดสอบจะถูก加以淘汰เพื่อทำเครื่องหมายแล้ว แผ่นเวเฟอร์จะถูกตัดด้วยไดเซอร์ (Dicer) ให้ด้วยแยกตัวออกจากกันเป็นสี่เหลี่ยม ด้วยตัวที่ผ่านการทดสอบก็จะถูกบีด (Bond) เข้ากับตัวถัง (Package) เชื่อมต่อขา กับตัวถังด้วยลวดทองคำ ให้ตรงตามลักษณะตัวถังที่ต้องการ หลังจากการทดสอบด้วย Part Tester เพื่อทดสอบหาชิปที่ไม่ผ่าน เมื่อพับแล้วเครื่องจะทำเครื่องหมายและไม่ถูกส่ง (Ship) ไปยังลูกค้า สรุปคือ **ชิป** (Chip) คือ ด้วยที่หุ้มด้วยตัวถังและการทดสอบคุณสมบัติทางไฟฟ้าและทางกลตามมาตรฐานเรียบร้อยแล้ว

**ซิลิกอน** คือ ธาตุหมู่ที่ 4 มีวาเลนซ์อิเล็กตรอน (Valence Electron) วงนอกสุดจำนวน 4 ตัว แผ่นซิลิกอนเวเฟอร์ คือ การนำธาตุซิลิกอนที่สกัดและหลอมจนเป็นแท่ง (Silicon Ingot) มาแล้วเป็นแผ่นบางๆ การโดดสาร คือ การเปลี่ยนแปลงแผ่นพื้นที่บนแผ่นเวเฟอร์เปล่าบางตำแหน่งเพื่อให้พื้นที่นั้นกลายเป็นสาร P โดยการเพิ่มไฮด (Hole) และกลายเป็นสาร N โดยการเพิ่มอิเล็กตรอน ตามลำดับ **ทรานซิสเตอร์** คือ การเชื่อมต่อพื้นที่ P และพื้นที่ N บนเวเฟอร์ ตามรูปแบบมาตรฐาน **ลอจิกเกต** (Logic Gate) คือ การนำทรานซิสเตอร์มาเชื่อมต่อกันตามรูปแบบมาตรฐาน เช่น แอนด์เกต (And Gate) แอนด์เกต (Nand Gate) เป็นต้น เพื่อทำงานตามพีชคณิตบูลลีน (Boolean Algebra) ซึ่งผู้อ่านเรียนรู้ในวิชาการออกแบบวงจรดิจิทัล **มอดูล** คือ การเชื่อมต่อกันของลอจิกเกตเป็นวงจรกลยุทธ์ที่มีความซับซ้อนตามที่กล่าวไปก่อนหน้านี้ โดยใช้ทองแดงหรืออลูมิเนียม (Aluminum) ตัวนำไฟฟ้าเชื่อมโดยทรานซิสเตอร์และวงจรเกทต่างๆ เข้าด้วยกัน ในแนวราบ เรียกว่า **เวีย** (Via) ซึ่งคำศัพท์เหล่านี้จะปรากฏในบทต่อๆ ไป

ผู้อ่านที่สนใจเทคโนโลยีเชิงลึกกว่าที่ทำรายเล่มนี้จะอธิบายได้ สามารถศึกษาเพิ่มเติมจากตัวอย่างคลิปวิดีโอเหล่านี้

- **Silicon Run Lite** ทางเว็บ [youtube.com](https://www.youtube.com) ซึ่งถ่ายทำพื้นฐานและวิวัฒนาการของวงจร ตั้งแต่

การสร้างและทดสอบชิป การประกอบและทดสอบบนแผ่นวงจรพิมพ์ จนถึงประกอบและทดสอบเครื่องคอมพิวเตอร์ ถึงแม้เทคโนโลยีจะล้าสมัย เพราะถ่ายทำเมื่อ 30 ปีที่แล้วเพื่อการศึกษา แต่เนื้อหาและหลักการไม่ต่างจากเทคโนโลยีในปัจจุบันมากนัก

- **Chip Manufacturing How are Microchips made?** ทางเว็บ [youtube.com](https://www.youtube.com) โดยบริษัท Infineon ประเทศเยอรมนี ซึ่งเป็นผู้ผลิตชิปชั้นนำของโลกด้วยเทคโนโลยีและอุปกรณ์ที่ทันสมัยและใกล้เคียงกับปัจจุบัน
- **How they make Raspberry Pi in the UK** โดยวารสาร Electronics Weekly ถ่ายทำโดย Metropolis Multimedia ทางเว็บ [youtube.com](https://www.youtube.com) อธิบายขั้นตอนการผลิตและประกอบบอร์ด Raspberry Pi โดยละเอียด ซึ่ง rogernan ใช้เครื่องจักรที่ทันสมัยขึ้น

## 1.5 สรุปท้ายบท

รูปแบบของเครื่องคอมพิวเตอร์มีความหลากหลายตามการประยุกต์ใช้งานในระบบต่างๆ นอกเหนือจากเครื่องคอมพิวเตอร์ที่มองเห็นทั่วไป ในการคมนาคมขนส่งต่างๆ ยังมีคอมพิวเตอร์ภายในรถยนต์ รถยนต์ไฟฟ้า หุ่นยนต์ต่างๆ เครื่องบิน อากาศยานไร้คนขับ (Unmanned Aeronautic Vehicle: UAV) โดรน (Drone) เป็นต้น ในการตรวจวัดค่าสิ่งแวดล้อม เช่น ลม ฝน คุณภาพอากาศ การพัฒนาระบบคอมพิวเตอร์เหล่านี้จึงต้องอาศัยความรู้ความเข้าใจทั้ง硬件และซอฟต์แวร์ควบคู่กันไป เพื่อให้ระบบทำงานได้เต็มประสิทธิภาพ มีอายุการใช้งานที่เหมาะสมและคุ้มค่าการลงทุน

## 1.6 คำถามท้ายบท

1. จงให้เหตุผลว่าทำไมคอมพิวเตอร์ชนิดแท็บเล็ตและสมาร์ตโฟนจึงได้รับความนิยมเพิ่มขึ้น จนเกือบแทนที่คอมพิวเตอร์ชนิดตั้งโต๊ะและโน้ตบุ๊ก ในແນ່ມຸນົມດັ່ງຕ້ອໄປນີ້
  - ขนาดของอุปกรณ์
  - การเชื่อมต่อกับผู้ใช้ (Human Interface)
  - ราคาและໂປຣໂມຊັ້ນການຂາຍ
  - ອື່ນາ
2. เราสามารถใช้เครื่องคอมพิวเตอร์ชนิดตั้งโต๊ะทดแทนเครื่องชนิดเซิร์ฟເວຼອຣີໄດ້ຫົວໜ້າໄວ່ ເພື່ອໃຫຍ່ໃຫຍ່ ຈະບອກເຫຼຸຜູ້ ຂໍອົດືສະໝັກສິນ ໃນແນ່ມຸນົມດັ່ງຕ້ອໄປນີ້
  - ຄວາມເຂົ້ມໍ່ນຂອງອຸປະກອນ (Reliability)
  - ຄຸນສົມບັດຈຳເພາະຂອງອຸປະກອນ (Specification)
  - ກາຮລົງທຸນແລະຄໍາໃຊ້ຈ່າຍອື່ນາ
  - ຮະບບປົງປັບປຸງ
  - ອື່ນາ
3. ເຫຼຸດໃດກາຮັດຊີປິຈຳເປັນຕ້ອງຄວບຄຸມອາກາສໃນດ້ານຄວາມສະອາດທີ່ເຮັດວຽກ ທ້ອງຄລິນຽມ (Clean Room)

4. จงบอกสภาพแวดล้อมที่ไม่เหมาะสมต่อการทำงานของเครื่องคอมพิวเตอร์ชนิดต่างๆ ในแต่ละดังต่อไปนี้

- อุณหภูมิ
- ความชื้น
- สนามแม่เหล็กและสนามไฟฟ้า
- ความสูงจากระดับน้ำทะเล
- อื่นๆ

5. การระบายน้ำร้อนของอุปกรณ์คอมพิวเตอร์มีความสำคัญต่อการทำงาน จงบอกข้อดีและข้อเสียของวิธีต่อไปนี้

- ฮีทซิงก์ (Heat Sink)
- พัดลม
- ของเหลว
- อื่นๆ

6. จงอธิบายคำว่า System on Chip (SoC) ว่าแตกต่างกับไมโครโปรเซสเซอร์ (Microprocessor) และไมโครคอนโทรลเลอร์ (Microcontroller) อย่างไร

7. จงค้นคว้าในอินเทอร์เน็ตว่า SoC ในปัจจุบันมีจำนวนทรานซิสเตอร์สูงสุดเป็นจำนวนกี่พันล้านตัวต่อชิป

## บทที่ 2

# ข้อมูลและคณิตศาสตร์ในคอมพิวเตอร์ (Computer Data and Arithmetic)

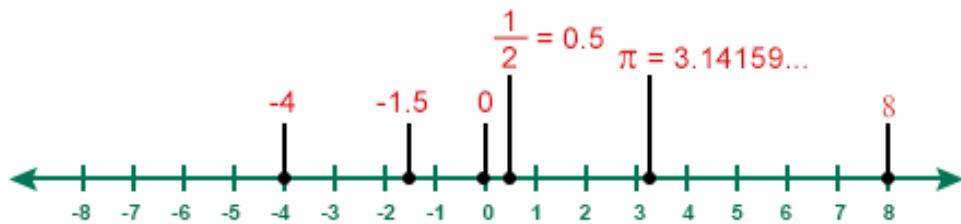
ชิปปี้หรือไมโครโปรเซสเซอร์ (Microprocessor) ตัวแรกของโลก คือ Intel 4004 ในปี ค.ศ. 1971 ซึ่งประดิษฐ์โดยบริษัท Intel สามารถคำนวณเลขจำนวนเต็มได้เพียง 4 บิตโดยใช้รหัส BCD (Binary Coded Decimal) ซึ่งรหัส BCD นี้สามารถใช้คำนวณเลขฐานสิบได้ โดยใช้เลขฐานสองจำนวน 4 บิตตั้งแต่  $0000_2$  ถึง  $1001_2$  แทนเลขฐานสิบจาก  $0_{10}$  ถึง  $9_{10}$  ตามลำดับ ดังนั้น ไมโครโปรเซสเซอร์ในอดีตจะเป็นต้องคำนวณตัวเลขทั้งจำนวนเต็มและทศนิยม เพื่อรองรับการใช้งานทางคณิตศาสตร์และตรรกศาสตร์ได้หลากหลายและซับซ้อนมากขึ้น

ในปัจจุบัน การใช้งานเครื่องคอมพิวเตอร์ชนิดต่างๆ เพื่อประมวลผลข้อมูลและอักษรให้กลายเป็นอินฟอร์เมชัน (Information) ที่เป็นประโยชน์ต่อธุรกิจการค้า สื่อสังคม และวัตถุประสงค์อื่นๆ การประมวลผลข้อมูลและอักษรเหล่านี้ จำเป็นต้องใช้คณิตศาสตร์ เช่น การบวก/ลบ คูณ/หาร และและตรรกศาสตร์ด้วยวงจรดิจิทัลซึ่งจัดเป็นชาร์ดแวร์เพื่อให้ใช้เวลาคำนวณน้อยที่สุด โดยมีนุ่มย์เข้าใจข้อมูลเชิงจำนวนด้วยตัวเลขฐานสิบ แต่คอมพิวเตอร์จำเป็นต้องประมวลผลด้วยเลขฐานสองแทน ดังนั้น วัตถุประสงค์ของบทนี้ คือ

- เพื่อให้ผู้อ่านเข้าใจรูปแบบและคณิตศาสตร์ของเลขจำนวนเต็มชนิดไม่มีเครื่องหมายและมีเครื่องหมาย (Integer: Unsigned and Signed) ในเครื่องคอมพิวเตอร์
- เพื่อให้ผู้อ่านเข้าใจรูปแบบและคณิตศาสตร์ของเลขทศนิยมฐานสองชนิดจุดตึง (Fixed-Point) ในเครื่องคอมพิวเตอร์
- เพื่อให้เข้าใจรูปแบบและคณิตศาสตร์ของเลขทศนิยมฐานสองชนิดจุดลอยตัว (Floating-Point) ตามมาตรฐาน IEEE754 ในเครื่องคอมพิวเตอร์
- เพื่อให้ผู้อ่านรู้จักรหัสเลขฐานสองตามมาตรฐาน ASCII และ Unicode สำหรับข้อมูลตัวอักษร ซึ่ง และข้อความต่างๆ ในเครื่องคอมพิวเตอร์

ดังนั้น เพื่อให้คอมพิวเตอร์สามารถคำนวณหรือกระทำการบวนการ เชิงคณิตศาสตร์กับเลขฐานสิบที่เป็นจำนวนเต็ม หรือจำนวนจริง ดังรูปเส้น จำนวนในรูปที่ 2.1 ในรูปของเลขฐานสองได้ เลขฐานสองจึงแบ่งเป็นเลขจำนวนเต็ม (Integer) และเลขจำนวนจริง (Real number) คล้ายกับเลขฐานสิบ ส่วนตัวอักษรแต่ละตัวที่มีนุ่มย์อ่านเข้าใจ คือ เลขฐานสองเช่นเดียวกัน

ทั้งนี้เนื่องจากข้อมูลและคำสั่งจะอยู่ในรูปของระดับความต่างศักย์หรือโวลเตจ หรือทิศทางของการไหลของกระแสไฟฟ้า เป็นเลข '1' หรือ '0' เรียกว่า บิต (bit: binary digit) ตามลำดับ ซึ่งการจัดเรียงบิตข้อมูลเหล่านี้



รูปที่ 2.1: เลขจำนวนเต็มและเลขจำนวนจริงบนเส้นจำนวนในคณิตศาสตร์เลขฐานสิบ ที่มา: [tutorvista.com](http://tutorvista.com)

หลาย ๆ บิต ให้กล้ายเป็นเลขฐานสองและเลขฐานสิบหาก จึงจำเป็นต้องใช้องค์ความรู้ด้านคณิตศาสตร์คอมพิวเตอร์ (Computer Arithmetic)

## 2.1 ข้อมูลพื้นฐานชนิดต่างในภาษา C/C++

ผู้อ่านควรมีพื้นฐานการเขียนหรือพัฒนาโปรแกรมคอมพิวเตอร์ด้วยภาษา C หรือ C++ มาบ้าง เพื่อช่วยให้เข้าใจเนื้อหาที่นี้และบทต่อๆ ไปได้ดีขึ้น ตารางที่ 2.1 แสดงรายชื่อชนิดของตัวแปรพื้นฐาน ความยาว (บิต) ค่าฐานสิบต่ำสุด (Minimum) และค่าฐานสิบสูงสุด (Maximum) สำหรับภาษา C/C++ ยกตัวอย่างเช่น ตัวแปรชนิด `char` นั้น ต้องการพื้นที่จำนวน 8 บิต หรือ 1 ไบต์ (Byte) โดยมีค่าฐานสิบต่ำสุดเท่ากับ -128 หรือ  $-2^7$  และมีค่าฐานสิบสูงสุดเท่ากับ +127 หรือ  $+2^7 - 1$

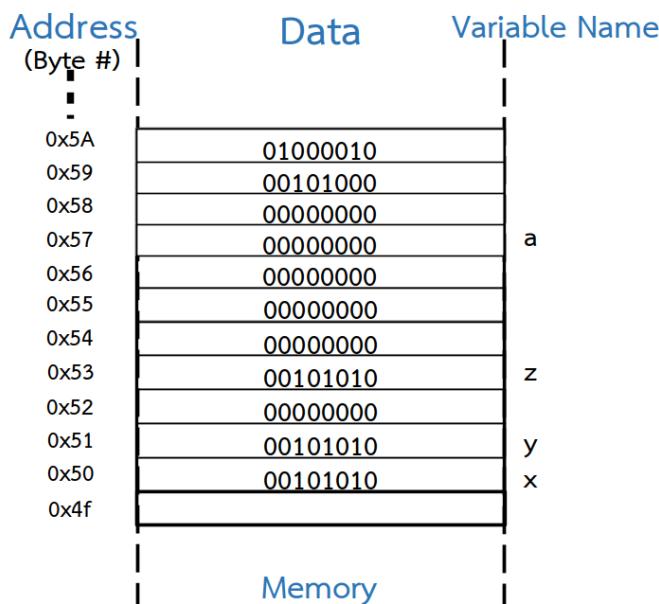
ตารางที่ 2.1: ชนิด ความยาว (บิต) ค่าฐานสิบต่ำสุดและสูงสุดของตัวแปรพื้นฐานแต่ละชนิดในภาษา C/C++ หมายเหตุ ค่าต่ำสุดและค่าสูงสุดของ IEEE754 เป็นค่าเลขอนอร์มัลไลซ์ (Normalize) เพื่อแสดงความสมมาตรของเลขยกกำลัง ที่มา: [ntu.edu.sg](http://ntu.edu.sg)

ชนิด	ความยาว(บิต)	ค่าต่ำสุด <sub>10</sub>	ค่าสูงสุด <sub>10</sub>
unsigned char	8	0	$2^8 - 1 = 255$
char	8	$-2^7 = -128$	$+2^7 - 1 = +127$
unsigned short	16	0	$2^{16} - 1 = 65,535$
short	16	$-2^{15} = -32,768$	$+2^{15} - 1 = +32,767$
unsigned int	32	0	$2^{32} - 1 = 4,294,967,295$
int	32	$-2^{31} = -2,147,483,648$	$+2^{31} - 1 = +2,147,483,647$
unsigned long long	64	0	$+2^{64} - 1$
long long	64	$-2^{63}$	$+2^{63} - 1$
float	32	$\pm 2^{-127} = \pm 1.18 \times 10^{-38}$	$\pm 2 \times 2^{127} = \pm 3.40 \times 10^{38}$
double	64	$\pm 2^{-1023} = \pm 2.23 \times 10^{-308}$	$\pm 2 \times 2^{1023} = \pm 1.80 \times 10^{308}$

ในขณะที่ตัวแปรชนิด `int` (Integer) นั้น ซึ่งส่วนใหญ่ต้องการพื้นที่จัดเก็บตัวแปรชนิด `int` นี้ 1 ตัวเป็นพื้นที่ 32 บิต หรือ 4 ไบต์ โดยมีค่าฐานสิบต่ำสุดเท่ากับ  $-2^{31}$  หรือ  $-2,147,483,648$  (ลบสองพันหนึ่งร้อยสี่สิบเจ็ดล้านสี่

แสэнแพดหมื่นสามพันหกร้อยสี่สิบแปด) และมีค่าฐานสิบสูงสุดเท่ากับ  $+2^{31}-1$  หรือ +2,147,483,647 (บวกสองพันหนึ่งร้อยสี่สิบเจ็ดล้านสี่แสэнแพดหมื่นสามพันหกร้อยสี่สิบเจ็ด)

ตัวอย่างการประกาศตัวแปรและตั้งค่าเริ่มต้นในภาษาสูง เช่น ภาษา C/C++ และ Java ผู้เขียนหรือนักพัฒนาจะต้องประกาศชื่อตัวแปร คล้ายกับการตั้งชื่อตัวละคร ทำการตั้งค่าเริ่มต้น (Initialize) เมื่อโปรแกรมทำงานจึงนำค่าเริ่มต้นของตัวแปรนั้นไปใช้งาน และอาจเปลี่ยนแปลงค่าจากการคำนวณหรือประมวลผลตามเงื่อนไขต่างๆ ที่เกิดขึ้นระหว่างที่โปรแกรมทำงาน ยกตัวอย่างเช่นประโยคภาษา C/C++ เหล่านี้ โดยสัญลักษณ์ /\* \*/ ใช้ครอบประโยคคอมเม้นท์ (Comment) สำหรับให้โปรแกรมเมอร์อธิบายประโยคภาษาคอมพิวเตอร์ให้มุชย์อ่านเข้าใจ



รูปที่ 2.2: พื้นที่ในหน่วยความจำซึ่งบรรจุค่าเริ่มต้นของตัวแปรที่ได้ประกาศในตัวอย่างที่ 2.1.1

### ตัวอย่างที่ 2.1.1 การประกาศตัวแปรและตั้งค่าเริ่มต้นด้วยภาษา C/C++

```
char x = '*' ;           /* x = 0x2A */
unsigned short y = 42;    /* y = 0x002A */
unsigned int z = 42;      /* z = 0x0000002A */
float a = 42.0;          /* a = 0x42280000 */
```

ผู้อ่านสามารถทำความเข้าใจประโยคประกาศตัวแปรและตั้งค่าเริ่มต้น ด้วยภาษา C/C++ จากตัวอย่างที่ 2.1.1 ดังนี้

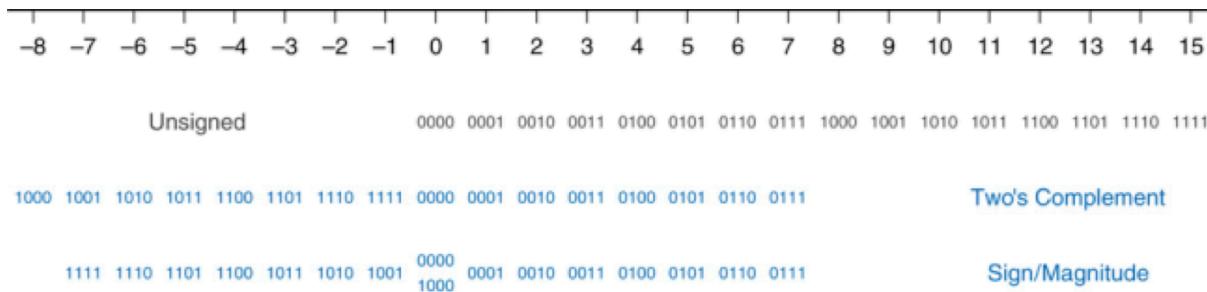
- x เป็นตัวแปรชนิด char มีค่าเริ่มต้นเป็นตัวอักษร '\*' ซึ่งตรงกับจำนวนเต็มชนิดมีเครื่องหมายมีค่าเริ่มต้นเท่ากับ 42 แต่ในหน่วยความจำขนาด 8 บิตที่จองไว้สำหรับตัวแปร x มีค่าเท่ากับ 00101010<sub>2</sub> หรือเท่ากับ 2A<sub>16</sub> โดยที่เลขฐานสิบหก 1 ตัวเท่ากับเลขฐานสองจำนวน 4 บิต และสัญลักษณ์ 0x หมายถึงเลขฐานสิบหกในภาษา C/C++ ผู้อ่านสามารถเปิดตารางรหัส ASCII ในรูปที่ 2.12
- y เป็นตัวแปรชนิดจำนวนเต็มชนิดมีเครื่องหมาย short มีค่าเริ่มต้นเท่ากับ 42<sub>10</sub> แต่ในหน่วยความจำขนาด 16 บิตที่จองไว้สำหรับตัวแปร y มีค่าเท่ากับ 0000000000101010<sub>2</sub> หรือเท่ากับ 002A<sub>16</sub>

- z เป็นตัวแปรชนิดจำนวนเต็มชนิดไม่มีเครื่องหมาย unsigned long มีค่าเริ่มต้นเท่ากับ 0 แต่ในหน่วยความจำขนาด 32 บิตที่จะอ้างไว้สำหรับตัวแปร z มีค่าเท่ากับ  $00000002A_{16}$
- a เป็นตัวแปรชนิดทศนิยม float มีค่าเริ่มต้นเท่ากับ  $42.0_{10}$  ในความคิดของโปรแกรมเมอร์ ในความเป็นจริงตัวแปร a มีค่าเท่ากับ  $42280000_{16}$  ตามมาตรฐาน IEEE754 ตัวแปรชนิดนี้ต้องการพื้นที่ในหน่วยความจำขนาด 32 บิต หรือ 4 ไบต์เริ่มต้นที่  $42280000_{16}$  ผู้อ่านสามารถทำการคำนวณแล้วนำผลมาใส่ในตัวอย่างที่ 2.6.1

ก่อนที่โปรแกรมจะเริ่มทำงาน ระบบปฏิบัติการจะจดจำพื้นที่ของหน่วยความจำในลักษณะคล้ายกับรูปที่ 2.2 ซึ่งหมายเลขไบต์ (Byte Number) หรือแอดdress (Address) ในหน่วยความจำ (Memory) จะเรียกว่าเป็นชั้นๆ ตั้งแต่หมายเลขไบต์ที่ 0 จนถึงตัวสุดท้ายที่  $2^{32}-1$  ที่อยู่ในหน่วยความจำขนาด 32 บิต หรือ 4 ไบต์ แต่ต้องทราบว่าในหน่วยความจำขนาด 32 บิต สามารถจัดเก็บข้อมูลได้มากกว่า 4 ไบต์ วงจรสามารถนำพื้นที่ของชั้นๆ ไปมาใช้งานด้วย ยกตัวอย่างเช่น ตัวแปร y และ z ในรูปที่ 2.2 ต้องการพื้นที่ 2 และ 4 ไบต์ ตามชนิดของตัวแปรในตารางที่ 2.1 ในบทนี้ ผู้อ่านจะได้ทำความเข้าใจกับข้อมูลชนิดจำนวนเต็มก่อน เพราะเป็นพื้นฐานของข้อมูลชนิดอื่นๆ และทำความเข้าใจกับเนื้อหาด้วยการปฏิบัติตามการทดลองที่ 1 ในภาคผนวก A

## 2.2 เลขจำนวนเต็มฐานสอง: รูปแบบและค่าฐานสิบ

เลขจำนวนเต็มฐานสอง (Integer) แบ่งเป็นเลขจำนวนเต็มไม่มีเครื่องหมาย (Unsigned Integer) และจำนวนเต็มมีเครื่องหมาย (Signed Integer) ซึ่งต้องมีกระบวนการตรวจสอบตัวบวกและลบ การคูณและการหาร และการตรวจจับโอเวอร์โฟล์ว (Overflow Detection) เนื่องจากการคำนวณด้วยวงจรติจิทัลภายในชิปจะต้องกำหนดจำนวนบิตที่ชัดเจนล่วงหน้า เช่น 32 หรือ 64 บิต



รูปที่ 2.3: เส้นจำนวนเปรียบเทียบข้อมูลเลขฐานสองความยาว 4 บิตในรูปแบบของเลขจำนวนเต็มชนิด Unsigned ชนิด 2's Complement และชนิด Sign-Magnitude ที่มา: [Harris and Harris \(2013\)](#)

รูปที่ 2.3 เส้นจำนวนเปรียบเทียบข้อมูลขนาด 4 บิตในรูปแบบของเลขจำนวนเต็มฐานสองชนิดไม่มีเครื่องหมาย (Unsigned) และชนิดมีเครื่องหมาย (Signed) แบบ 2's Complement

## 2.2.1 ชนิดไม่มีเครื่องหมาย (Unsigned Integer)

เลขจำนวนเต็มฐานสองชนิดไม่มีเครื่องหมายในคอมพิวเตอร์ เหมาะสำหรับการใช้นับจำนวนสิ่งของต่างๆ ซึ่งจำนวนจะมีค่าเริ่มต้นตั้งแต่ 0 จนถึงจำนวนที่ฮาร์ดแวร์และซอฟต์แวร์รองรับ ตามหลักการ ดังนั้น หัวข้อนี้จึงมีความสำคัญและจำเป็นต้องทำการทดลองที่ 1 ในภาคผนวก A และการทดลองที่ 5 ภาคผนวก E เพื่อความเข้าใจอย่างลึกซึ้งถึงข้อจำกัดของคอมพิวเตอร์

นอกจากนี้ ยังมีตัวแปรชนิดพอยน์เตอร์ (Pointer) ซึ่งทำหน้าที่เก็บแอดдрес หรือ หมายเลขไปร์ หรือ หมายเลขชี้ของหน่วยความจำที่บรรจุค่าของตัวแปรนั้นๆ หมายเลข 40 แอดдрес คือ ตัวเลขด้านซ้ายของหน่วยความจำในรูปที่ 2.2 ยกตัวอย่างเช่น ตัวแปร  $x$  ตั้งอยู่ที่หมายเลขแอดdress 0x50 บรรจุค่า 0x2A หรือ  $42_{10}$  หมายเลขแอดdress 0x50 เป็นการเขียนแบบย่อ เมื่อเขียนให้ครบ 4 ไปร์ หรือ 32 บิตมีค่าเท่ากับ 0x00000050 สำหรับตัวแปรพอยน์เตอร์ในระบบปฏิบัติการ 32 บิต

ตัวแปรชนิดพอยน์เตอร์มีความกว้างเท่ากับศูนย์ ตัวอย่างภาษา C/C++ ที่ใช้ประกาศและตั้งค่าเริ่มต้นตัวแปรจำนวนเต็มชนิดไม่มีเครื่องหมาย ซึ่งจะสังเกตเห็นว่าขึ้นต้นด้วยคำว่า unsigned และตามด้วยชนิดของตัวแปรซึ่งจะเป็นตัวกำหนดความยาวของตัวแปรตามนิยามที่ 2.2.1

**นิยามที่ 2.2.1** กำหนดให้ เลขจำนวนเต็มฐานสองชนิดไม่มีเครื่องหมาย (Unsigned Integer)  $X_{2,u}$  ความยาว  $n$  บิตเป็นเลขฐานสองเขียนอยู่ในรูป

$$X_{2,u} = x_{n-1}x_{n-2}x_{n-3}\dots x_1x_0 \quad (2.1)$$

เมื่อ  $x_i$  คือค่า “1” หรือ “0” ในตำแหน่งที่  $i$  และตำแหน่งของมีอสูดคือตำแหน่งที่  $i=0$

การเข้มโภกระหว่างเลขฐานสองและเลขฐานสิบจะต้องอาศัยการแปลงระหว่างทั้งสองฐานดังต่อไปนี้

### การแปลงเลขฐานสองเป็นฐานสิบ

จากนิยามที่ 2.2.1 ค่าจำนวนเต็มฐานสิบ  $X_{10,u}$  ของเลข  $X_{2,u}$  สามารถคำนวณได้จากการสมการ

$$X_{10,u} = x_{n-1}2^{n-1} + x_{n-2}2^{n-2} + \dots + x_12^1 + x_02^0 \quad (2.2)$$

ดังนั้น ค่าฐานสิบ  $X_{10,u}$  อยู่ในช่วง 0 ถึง  $+2^n - 1$

**ตัวอย่างที่ 2.2.1** เลขจำนวนเต็มฐานสองชนิดไม่มีเครื่องหมาย  $n=4$  บิต  $X_{2,u} = 1011_2 = B_{16}$  ค่าฐานสิบของ  $X_{2,u}$  ตามสมการที่ (2.2) คือ

$$X_{10,u} = 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 \quad (2.3)$$

$$= 8 + 0 + 2 + 1 \quad (2.4)$$

$$= 11_{10} \quad (2.5)$$

ดังนั้น เลขจำนวนเต็มฐานสองชนิดไม่มีเครื่องหมายขนาด  $n = 4$  บิตจะมีค่าฐานสิบอยู่ในช่วง 0 ถึง  $+15$

ตัวอย่างที่ 2.2.2 เลขจำนวนเต็มฐานสองชนิดไม่มีเครื่องหมาย  $n = 8$  บิต

$$X_{2,u} = 0000\ 1011_2 = 0B_{16}$$

เทียบเท่ากับการประมวลผลและตั้งค่าเริ่มต้นตัวแปรชนิด *unsigned char*

```
unsigned char X = 11; /* X = 0b00001011 = 0x0B */
```

มีค่าฐานสิบเท่ากับเท่าไหร่

ค่าฐานสิบของ  $X_{2,u}$  ตามสมการที่ (2.2) คือ

$$X_{10,u} = 0 + \dots + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 \quad (2.6)$$

$$= 0 + \dots + 8 + 0 + 2 + 1 \quad (2.7)$$

$$= 11_{10} \quad (2.8)$$

ตัวแปรชนิด *unsigned char* หรือเลขจำนวนเต็มฐานสองชนิดไม่มีเครื่องหมายขนาด  $n = 8$  บิตจะมีค่าฐานสิบอยู่ในช่วง 0 ถึง +255

ตัวอย่างที่ 2.2.3 เลขจำนวนเต็มฐานสองชนิดไม่มีเครื่องหมาย  $n = 16$  บิต

$$X_{2,u} = 0000\ 0000\ 0000\ 1011_2 = 000B_{16}$$

เทียบเท่ากับการประมวลผลและตั้งค่าเริ่มต้นตัวแปรชนิด *unsigned short*

```
unsigned short X = 11; /* X = 0x000B */
```

มีค่าฐานสิบเท่ากับเท่าไหร่

ค่าฐานสิบของ  $X_{2,u}$  ตามสมการที่ (2.2) คือ

$$X_{10,u} = 0 + \dots + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 \quad (2.9)$$

$$= 0 + \dots + 8 + 0 + 2 + 1 \quad (2.10)$$

$$= 11_{10} \quad (2.11)$$

ดังนั้น ตัวแปรชนิด *unsigned short* หรือเลขจำนวนเต็มฐานสองชนิดไม่มีเครื่องหมายขนาด  $n = 16$  บิตจะมีค่าฐานสิบอยู่ในช่วง 0 ถึง +65,535

ตัวอย่างที่ 2.2.4 เลขจำนวนเต็มฐานสองชนิดไม่มีเครื่องหมาย  $n = 32$  บิต

$$X_{2,u} = 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 1011_2 = 0000000B_{16}$$

เทียบเท่ากับการประมวลผลและตั้งค่าเริ่มต้นตัวแปรชนิด *unsigned int*

```
unsigned int X = 11; /* X = 0x0000000B */
```

มีค่าฐานสิบเท่ากับเท่าไหร่

ค่าฐานสิบของ  $X_{2,u}$  ตามสมการที่ (2.2) คือ

$$X_{10,u} = 0 + \dots + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 \quad (2.12)$$

$$= 0 + \dots + 8 + 0 + 2 + 1 \quad (2.13)$$

$$= 11_{10} \quad (2.14)$$

ดังนั้น ตัวแปรชนิด *unsigned int* หรือเลขจำนวนเต็มฐานสองชนิดไม่มีเครื่องหมายขนาด  $n = 32$  บิตจะมีค่าฐานสิบอยู่ในช่วง 0 ถึง  $+4,294,967,295$  (บวกสิ่งของร้อยเก้าสิบสิ่ล้านเก้าแสนหกหมื่นเจ็ดพันสองร้อยเก้าสิบห้า)

ผู้อ่านจะเห็นได้จากตัวอย่างที่ 2.2.1 ถึง 2.2.4 เลขฐานสิบค่าเดียวกัน เมื่อนำไปคำนวณด้วยวงจรดิจิทัลหรือด้วยชนิดตัวแปรที่มีความยาวต่างกัน

### การแปลงเลขฐานสิบเป็นฐานสอง

การแปลงเลขฐานสิบเป็นฐานสองชนิดไม่มีเครื่องหมาย (Unsigned) แบ่งเป็น 2 วิธี คือ

- การหารเอาเศษด้วย 2 เป็นจำนวน  $n$  ครั้ง
- การลบด้วยเลขยกกำลังสองเป็นจำนวน  $n$  ครั้ง

ผู้อ่านสามารถศึกษาการทำงานและความแตกต่างระหว่างสองวิธีได้ตามตัวอย่างที่ 2.2.5 และตัวอย่างที่ 2.2.6 ตามลำดับ โดยใช้เลขฐานสิบค่าเดียวกันแปลงให้เป็นเลขฐานสองขนาด  $n=8$  บิต

ตัวอย่างที่ 2.2.5 จงแปลง 123 เป็นเลขฐานสองด้วยวิธีหารเอาเศษด้วย 2 เป็นจำนวน  $n$  ครั้ง

1. การหารเอาเศษด้วย 2 เป็นจำนวน  $n$  ครั้ง เป็นวิธีที่เข้าใจง่ายและซับซ้อนน้อยกว่าวิธีที่สอง

ตารางที่ 2.2: การแปลงค่าฐานสิบเป็นเลขฐานสองแบบไม่มีเครื่องหมายความยาว  $n=8$  บิต ด้วยการหาร

บิตที่	เลขฐานสิบ	ผลหาร	เศษ
-	123		
0	123/2	61	1
1	61/2	30	1
2	30/2	15	0
3	15/2	7	1
4	7/2	3	1
5	3/2	1	1
6	1/2	0	1
7	0/2	0	0

ดังนั้น  $X_{2,u}$  ของ  $123_{10} = 0111\ 1011_2 = 7B_{16}$

2. การลบด้วยเลขสองยกกำลัง ( $2^i$  เมื่อ  $i=n-1$  ถึง 0) เป็นวิธีที่ซับซ้อนมากกว่าวิธีการหารเอาเศษ โดยผู้ใช้ควรจะจำเลขยกกำลังสองได้ หมายสำคัญที่มีประสบการณ์กับการใช้เลขฐานสองบ่อยๆ

ตัวอย่างที่ 2.2.6 จงแปลง 123 เป็นเลขฐานสองด้วยวิธีลับด้วยเลขยกกำลังสองเป็นจำนวน  $n$  ครั้ง

ตารางที่ 2.3: การแปลงค่าฐานสิบเป็นเลขฐานสองแบบไม่มีเครื่องหมายความยาว  $n=8$  บิต ด้วยวิธีการลับ

บิตที่ $i$	$2^i$	ผลลัพธ์- $2^i$	ตัวตั้ง <sub>10</sub>	$x_i$
-			123	
7	$2^7=128$	123-128	123	0
6	$2^6=64$	123-64	59	1
5	$2^5=32$	59-32	27	1
4	$2^4=16$	27-16	11	1
3	$2^3=8$	11-8	3	1
2	$2^2=4$	3-4	3	0
1	$2^1=2$	3-2	1	1
0	$2^0=1$	1-1	0	1

- บิต  $x_i$  จะเท่ากับ 1 หากตัวตั้งลบค่าประจำตำแหน่งได้ และตัวตั้งจะมีค่าลดลง
- บิต  $x_i$  จะเท่ากับ 0 หากตัวตั้งลบค่าประจำตำแหน่งไม่ได้ และตัวตั้งจะมีค่าคงเดิม

ตัวนั้น  $X_{2,u}$  ของ  $123_{10} = 0111\ 1011_2 = 7B_{16}$  เช่นกัน

ตัวอย่างที่ 2.2.5 และ ตัวอย่างที่ 2.2.6 แสดงการแปลงเลขฐานสิบ เป็นเลขฐานสองชนิดไม่มีเครื่องหมายจำนวน  $n$  บิตทั้งสองวิธี โดยที่ผู้อ่านสามารถทำตามวิธีการหารเอาเศษได้ยากกว่า ส่วนวิธีการลับด้วยเลขสองยกกำลังคือการแปลงเลขฐานโดยตรง แต่ต้องอาศัยความจำมากกว่า

## 2.2.2 ชนิดมีเครื่องหมาย (Signed Integer) แบบ 2's Complement

เลขจำนวนเต็มฐานสองชนิดมีเครื่องหมายในคอมพิวเตอร์ เนماสสำหรับการใช้แทนข้อมูลที่มีค่าทั้งบวกและลบบนเส้นจำนวนตามที่ฮาร์ดแวร์และซอฟต์แวร์จะรองรับตามหลักการ 2's Complement เกือบทั้งหมด ดังนั้น หัวข้อนี้จึงมีความสำคัญและจำเป็นต้องทำการทดลองที่ 1 และ 5 ในภาคผนวก A และภาคผนวก E ตามลำดับ เพื่อความเข้าใจอย่างลึกซึ้งถึงข้อจำกัดของคอมพิวเตอร์

```
char x = -5; /* x = 0b11111011 = 0xFB */
short y = -5; /* y = 0b1111111111111011 = 0xFFFFB */
int z = -5; /* z = 0xFFFFFFFFB */
```

นิยามที่ 2.2.2 กำหนดให้ เลขจำนวนเต็มฐานสองชนิดมีเครื่องหมาย (Signed Integer) แบบ 2's Complement  $X_{2,s}$  ความยาว  $n$  บิต เชียนอยู่ในรูป

$$X_{2,s} = x_{n-1}x_{n-2}x_{n-3}..x_1x_0 \quad (2.15)$$

เมื่อ  $x_{n-1}$  ทำหน้าที่เป็นบิตเครื่องหมาย (Sign bit) มีค่า 1 เมื่อเป็นเลขลบ และ 0 เมื่อเป็นเลขบวก และ  $x_i$  คือ บิตข้อมูลมีค่า 1 หรือ 0 ในตำแหน่งที่  $i$  และตำแหน่งของมีสุดคือตำแหน่งที่  $i = 0$  ที่มา: Patterson and Hennessy (2016)

### การแปลงเลขฐานสองเป็นฐานสิบ

การแปลงเลขจำนวนเต็มฐานสองชนิดมีเครื่องหมายแบบ 2's Complement จากนิยามที่ 2.2.2 ให้เป็นค่าฐานสิบสามารถทำได้โดย

$$X_{10,s} = -x_{n-1} \times 2^{n-1} + x_{n-2}2^{n-2} + .. + x_12^1 + x_02^0 \quad (2.16)$$

ดังนั้น ค่าฐานสิบ  $X_{10,s}$  อยู่ในช่วง  $-2^{n-1}$  ถึง  $+2^{n-1} - 1$

ตัวอย่างที่ 2.2.7 เลขจำนวนเต็มฐานสองชนิดมีเครื่องหมายแบบ 2's Complement ขนาด  $n = 4$  บิต  $X_{2,s} = 1011_2 = B_{16}$  มีค่าฐานสิบเท่ากับเท่าไร

ค่าฐานสิบของ  $X_{2,s}$  ตามสมการที่ (2.16) คือ

$$X_{10,s} = -1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 \quad (2.17)$$

$$= -8 + 0 + 2 + 1 \quad (2.18)$$

$$= -5_{10} \quad (2.19)$$

เลขจำนวนเต็มฐานสองชนิดมีเครื่องหมาย 2's Complement ขนาด  $n = 4$  บิตจะมีค่าฐานสิบอยู่ในช่วง -8 ถึง +7

จากตัวอย่างที่ 2.2.7 เลขฐานสองความยาว  $n=4$  บิตสามารถตีความแบบมีเครื่องหมาย และแบบไม่มีเครื่องหมาย ตามสมการที่ (2.16) และ สมการที่ (2.2) ตามลำดับ ในตารางที่ 2.4

ตารางที่ 2.4: รูปแบบ (Pattern) ของเลขฐานสองขนาด  $n=4$  บิตทั้งหมด  $2^4=16$  แบบและการตีความหมายให้เป็นค่าฐานสิบแบบมีเครื่องหมาย 2's Complement และแบบไม่มีเครื่องหมาย

เลขฐานสอง $n=4$ บิต	$X_{10,s}$ ค่าฐานสิบ มีเครื่องหมาย สมการ (2.16)	$X_{10,u}$ ค่าฐานสิบ ไม่มีเครื่องหมาย สมการ (2.2)
1000	-8	8
1001	-7	9
1010	-6	10
<b>1011</b>	<b>-5</b>	<b>11</b>
1100	-4	12
1101	-3	13
1110	-2	14
1111	-1	15
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7

โปรดสังเกตหมายเหตุฐานสอง 1011<sub>2</sub> สามารถตีความเป็นเลขจำนวนเต็มชนิดมีเครื่องหมายแบบ 2's Complement ในคอลัมน์กลางว่าเท่ากับ -5<sub>10</sub> และเลขจำนวนเต็มชนิดไม่มีเครื่องหมายในคอลัมน์ขวาสุดว่าเท่ากับ 11<sub>10</sub>

ตัวอย่างที่ 2.2.8 เลขจำนวนเต็มฐานสองแบบ 2 Complement  $n = 5$  บิต  $X_{2,s} = 11011_2 = 1B_{16}$  มีค่าฐานสิบเท่ากับเท่าไร

ค่าฐานสิบของ  $X_{2,s}$  ตามสมการที่ (2.16) คือ

$$X_{10,s} = -1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 \quad (2.20)$$

$$= -16 + 8 + 0 + 2 + 1 \quad (2.21)$$

$$= -5_{10} \quad (2.22)$$

เลขฐานสองแบบ 2 Complement ขนาด  $n = 5$  บิตจะมีค่าฐานสิบอยู่ในช่วง  $-2^{5-1}$  ถึง  $+2^{5-1} - 1$  หรือ -16 ถึง 15

เลขจำนวนเต็มฐานสองความยาว  $n=5$  บิตสามารถตีความแบบมีเครื่องหมายและแบบไม่มีเครื่องหมาย ตามสมการที่ (2.16) และสมการที่ (2.2) ตามลำดับ ในตารางที่ 2.5

ตารางที่ 2.5: รูปแบบ (Pattern) ของเลขฐานสองขนาด  $n=5$  บิตทั้งหมด  $2^5=32$  แบบและการตีความหมายให้เป็นค่าฐานสิบแบบมีเครื่องหมาย 2's Complement และแบบไม่มีเครื่องหมาย

เลขฐานสอง $n=5$ บิต	$X_{10,s}$ ค่าฐานสิบ มีเครื่องหมาย สมการ (2.16)	$X_{10,u}$ ค่าฐานสิบ ไม่มีเครื่องหมาย สมการ (2.2)
1 0000	-16	16
...	...	...
1 0111	-9	23
1 1000	-8	24
1 1001	-7	25
1 1010	-6	26
<b>1 1011</b>	<b>-5</b>	<b>27</b>
1 1100	-4	28
1 1101	-3	29
1 1110	-2	30
1 1111	-1	31
0 0000	0	0
0 0001	1	1
0 0010	2	2
0 0011	3	3
0 0100	4	4
0 0101	5	5
0 0110	6	6
0 0111	7	7
...	...	...
0 1111	15	15

หมายเหตุ ผู้เขียนจะใจเว้นช่องว่างทุกๆ 4 บิต เพื่อให้ผู้อ่านนับเลขฐานสองและแปลงเป็นเลขฐานสิบหากได้จ่ายชื่น

โปรดสังเกตหมายเหตุ สำหรับตีความเป็นเลขจำนวนเต็มชนิดมีเครื่องหมายแบบ 2's Complement ในคอลัมน์กลางว่าเท่ากับ  $-5_{10}$  และเลขจำนวนเต็มชนิดไม่มีเครื่องหมายในคอลัมน์ขวาสุดว่าเท่ากับ  $27_{10}$

ตัวอย่างที่ 2.2.9 เลขจำนวนเต็มฐานสองชนิดมีเครื่องหมาย 2's Complement  $n = 8$  บิต

$$X_{2,s} = 1111\ 1011_2 = FB_{16}$$

เทียบเท่ากับการประกาศและตั้งค่าเริ่มต้นตัวแปรชนิด char

```
char X = -5; /* X = 0b11111011 = 0xFB */
```

มีค่าฐานสิบเท่ากับเท่าไหร่

ค่าฐานสิบของ  $X_{2,s}$  ตามสมการที่ (2.16) คือ

$$X_{10,s} = -1 \times 2^7 + 1 \times 2^6 + 1 \times 2^5 + 1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 \quad (2.23)$$

$$= -128 + 64 + 32 + 16 + 8 + 0 + 2 + 1 \quad (2.24)$$

$$= -5_{10} \quad (2.25)$$

ตัวแปรชนิด *char* หรือเลขจำนวนเต็มขนาด  $n = 8$  บิต จะมีค่าฐานสิบอยู่ในช่วง -128 ถึง +127

เลขจำนวนเต็มฐานสองความยาว  $n=8$  บิตสามารถตีความแบบมีเครื่องหมาย 2's Complement และแบบไม่มีเครื่องหมาย ตามสมการที่ (2.16) และสมการที่ (2.2) ตามลำดับ คล้ายกับกรณี  $n=5$  บิต ในตารางที่ 2.6

ตารางที่ 2.6: รูปแบบ (Pattern) ของเลขฐานสองขนาด  $n=8$  บิตทั้งหมด  $2^8=256$  แบบและการตีความหมายให้เป็นค่าฐานสิบแบบมีเครื่องหมาย 2's Complement และแบบไม่มีเครื่องหมาย

เลขฐานสอง $n=8$ บิต	$X_{10,s}$ ค่าฐานสิบ มีเครื่องหมาย สมการ (2.16)	$X_{10,u}$ ค่าฐานสิบ ไม่มีเครื่องหมาย สมการ (2.2)
1000 0000	-128	128
...	...	...
1111 0111	-9	247
1111 1000	-8	248
1111 1001	-7	249
1111 1010	-6	250
<b>1111 1011</b>	<b>-5</b>	<b>251</b>
1111 1100	-4	252
1111 1101	-3	253
1111 1110	-2	254
1111 1111	-1	255
0000 0000	0	0
0000 0001	1	1
0000 0010	2	2
0000 0011	3	3
0000 0100	4	4
0000 0101	5	5
0000 0110	6	6
0000 0111	7	7
0000 1000	8	8
...	...	...
0111 1111	127	127

หมายเหตุ ผู้เขียนจะใจเว้นช่องว่างทุกๆ 4 บิต เพื่อให้ผู้อ่านนับเลขฐานสองและแปลงเป็นเลขฐานสิบหลังได้ง่ายขึ้น

โปรดสังเกตหมายเหตุ  $1111\ 1011_2$  สามารถตีความเป็นเลขจำนวนเต็มชนิดมีเครื่องหมายแบบ 2's Complement ในคอลัมน์กลางว่าเท่ากับ  $-5_{10}$  และเลขจำนวนเต็มชนิดไม่มีเครื่องหมายในคอลัมน์ขวาสุดว่าเท่ากับ  $251_{10}$  ผู้อ่านจะสังเกตเห็นได้ว่า  $1111$  ด้านซ้ายของ  $1111\ 1011_2$  ในตารางนี้ ทำหน้าที่เหมือน Sign bit ซึ่งสามารถประยุกต์ใช้กับเลขจำนวนเต็มฐานสองแบบ 2's Complement ได้ทุกความยาว  $n$

ตัวอย่างที่ 2.2.10 เลขจำนวนเต็มฐานสองแบบ 2's Complement  $n = 16$  บิต

$$X_{2,s} = 1111\ 1111\ 1111\ 1011_2 = FFFF B_{16}$$

เทียบเท่ากับการประมวลผลและตั้งค่าเริ่มต้นตัวแปรชนิด short

```
short X = -5; /* X = 0b1111111111111011 = 0xFFFFB */
```

มีค่าฐานสิบเท่ากับเท่าไหร่

ค่าฐานสิบของ  $X_{2,s}$  ตามสมการที่ (2.16) คือ

$$X_{10,s} = -1 \times 2^1 + 1 \times 2^{14} + \dots + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 \quad (2.26)$$

$$= -32,768 + \dots + 8 + 0 + 2 + 1 \quad (2.27)$$

$$= -5_{10} \quad (2.28)$$

ตัวแปรชนิด short หรือเลขจำนวนเต็มฐานสองชนิดมีเครื่องหมาย 2's Complement ขนาด  $n=16$  บิตจะมีค่าฐานสิบอยู่ในช่วง  $-32,768$  ถึง  $+32,767$

ตัวอย่างที่ 2.2.11 เลขจำนวนเต็มฐานสองแบบ 2's Complement  $n = 32$  บิต

$$X_{2,s} = 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1011_2 = FFFFFFFFB_{16}$$

เทียบเท่ากับการประมวลผลและตั้งค่าเริ่มต้นตัวแปรชนิด int

```
int X = -5; /* X = 0xFFFFFFFFFB */
```

มีค่าฐานสิบเท่ากับเท่าไหร่

ค่าฐานสิบของ  $X_{2,s}$  ตามสมการที่ (2.16) คือ

$$X_{10,s} = -1 \times 2^{31} + 1 \times 2^{30} + \dots + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 \quad (2.29)$$

$$= -2,147,483,648 + \dots + 8 + 0 + 2 + 1 \quad (2.30)$$

$$= -5_{10} \quad (2.31)$$

ตัวแปรชนิด int หรือเลขจำนวนเต็มฐานสองชนิดมีเครื่องหมาย 2's Complement ขนาด  $n = 32$  บิตจะมีค่าฐานสิบอยู่ในช่วง  $-2,147,483,648$  ถึง  $+2,147,483,647$

ผู้อ่านจะเห็นได้จากตัวอย่างที่ 2.2.7 ถึง 2.2.11 เลขฐานสิบค่าเดียวกัน เมื่อนำไปคำนวณในวงจรดิจิทัลที่มีความยาวต่างกัน จะมีการเติมบิตเครื่องหมาย (Sign bit) ทางด้านซ้ายเพื่อให้ขยายขึ้น เรียกว่า การขยายเครื่องหมาย (Sign Extension)

```
char X = -5; /* X = 0b11111011 = 0xFB */
short X = -5; /* X = 0b1111111111111011 = 0xFFFFB */
int X = -5; /* X = 0xFFFFFFFFFB */
```

สรุปจากตัวอย่างการประมวลผลตัวแปรทั้งสามชนิดนี้ หมายเลขอฐานสิบ  $-5$  สามารถใช้ตัวแปรหลายชนิดในการคำนวณได้ ทั้งนี้ขึ้นกับการใช้งานตัวแปรนั้นมีค่าต่ำสุดและสูงสุดที่แตกต่างกัน เนื่องจากตัวแปรแต่ละชนิดใช้จำนวนบิตข้อมูลไม่เท่ากัน ซึ่งจะแสดงรายละเอียดเพิ่มเติมในหัวข้อถัดไป

### การแปลงเลขฐานสิบเป็นฐานสอง

การแปลงเลขฐานสิบที่มีเครื่องหมาย  $X_{10,s}$  ให้เป็นเลขฐานสองแบบ 2's Complement อาศัยพื้นฐานของ การแปลงเลขฐานสิบที่ไม่มีเครื่องหมาย แบ่งเป็น 2 กรณี คือ กรณี  $X_{10,s} < 0$

$$X_{2,s} = \overline{[|X_{10,s}|]_{2,u}} + 1_2 \quad (2.32)$$

เมื่อ  $|X_{10,s}|$  คือ ค่าสัมบูรณ์ของ  $X_{10,s}$  และกรณี  $X_{10,s} \geq 0$

$$X_{2,s} = [X_{10,s}]_2 = [X_{10,u}]_{2,u} \quad (2.33)$$

โดย  $[X_{10,u}]_{2,u}$  คือ การแปลงเลขจำนวนเต็มฐานสิบให้เป็นฐานสองไม่มีเครื่องหมาย ตามสมการที่ (2.2) ตารางที่ 2.7 และ 2.8 เป็นตัวอย่างการแปลงเลขจำนวนเต็มฐานสิบเป็นเลขฐานสองที่ความยาว  $n=4$  และ 5 บิต ตามลำดับ

ตารางที่ 2.7: การแปลงค่าฐานสิบแบบมีเครื่องหมายให้เป็นเลขฐานสองความยาว  $n=4$  บิต โดยแปลงให้เป็นค่าฐานสิบโดยใช้สมการที่ (2.32) กรณีที่  $X_{10,s} < 0$

เลขฐานสิบ $X_{10,s} < 0$	เลขฐานสอง $[ X_{10,s} ]_{2,u}$	เลขฐานสอง $X_{2,s}$ $\overline{[ X_{10,s} ]_{2,u}} + 1_2$ สมการ (2.32)
-8	$8=1000_2$	$0111_2 + 1_2 = 1000_2$
-7	$7=0111_2$	$1000_2 + 1_2 = 1001_2$
-6	$6=0110_2$	$1001_2 + 1_2 = 1010_2$
<b>-5</b>	$5=0101_2$	$1010_2 + 1_2 = \mathbf{1011}_2$
-4	$4=0100_2$	$1011_2 + 1_2 = 1100_2$
-3	$3=0011_2$	$1100_2 + 1_2 = 1101_2$
-2	$2=0010_2$	$1101_2 + 1_2 = 1110_2$
-1	$1=0001_2$	$1110_2 + 1_2 = 1111_2$
เลขฐานสิบ $X_{10,s} \geq 0$		เลขฐานสอง $X_{2,s}$ สมการ (2.33)
0		$0000_2$
1		$0001_2$
2		$0010_2$
3		$0011_2$
4		$0100_2$
5		$0101_2$
6		$0110_2$
7		$0111_2$

โปรดเปรียบเทียบตัวเลขในตารางนี้กับตารางที่ 2.4 เนื่องจากความต่างที่ด้านซ้ายว่าไม่แตกต่างกัน

ตารางที่ 2.8: การแปลงค่าฐานสิบแบบมีเครื่องหมายให้เป็นเลขฐานสองความยาว  $n=5$  บิต โดยแปลงให้เป็นค่าฐานสิบโดยใช้สมการที่ (2.32) กรณีที่  $X_{10,s} < 0$

เลขฐานสิบ $X_{10,s} < 0$	เลขฐานสอง $[\lceil X_{10,s} \rceil]_{2,u}$	เลขฐานสอง $X_{2,s}$ $\overline{[\lceil X_{10,s} \rceil]_{2,u}} + 1_2$ สมการ (2.32)
-16	$16 = 10000_2$	$01111_2 + 1_2 = 1\ 0000_2$
...	...	...
-9	$9 = 01001_2$	$10110_2 + 1_2 = 1\ 0111_2$
-8	$8 = 01000_2$	$10111_2 + 1_2 = 1\ 1000_2$
-7	$7 = 00111_2$	$11000_2 + 1_2 = 1\ 1001_2$
-6	$6 = 00110_2$	$11001_2 + 1_2 = 1\ 1010_2$
<b>-5</b>	$5 = 00101_2$	$11010_2 + 1_2 = 1\ 1011_2$
-4	$4 = 00100_2$	$11011_2 + 1_2 = 1\ 1100_2$
-3	$3 = 00011_2$	$11100_2 + 1_2 = 1\ 1101_2$
-2	$2 = 00010_2$	$11101_2 + 1_2 = 1\ 1110_2$
-1	$1 = 00001_2$	$11110_2 + 1_2 = 1\ 1111_2$
เลขฐานสิบ $X_{10,s} \geq 0$		เลขฐานสอง $X_{2,s}$ สมการ (2.33)
0		<b>0</b> 0000 <sub>2</sub>
1		<b>0</b> 0001 <sub>2</sub>
2		<b>0</b> 0010 <sub>2</sub>
3		<b>0</b> 0011 <sub>2</sub>
4		<b>0</b> 0100 <sub>2</sub>
5		<b>0</b> 0101 <sub>2</sub>
6		<b>0</b> 0110 <sub>2</sub>
7		<b>0</b> 0111 <sub>2</sub>
9		<b>0</b> 1001 <sub>2</sub>
...		...
15		<b>0</b> 1111 <sub>2</sub>

หมายเหตุ ผู้เขียนจะใจเว้นช่องว่างทุกๆ 4 บิต เพื่อให้ผู้อ่านนับเลขฐานสองและแปลงเป็นเลขฐานสิบหากได้จ่ายชี้น

โปรดเปรียบเทียบตัวเลขในตารางนี้กับตารางที่ 2.7 ว่าแตกต่างกันตรงที่ -1 ถึง -8 เป็นการขยายบิตเครื่องหมาย  $x_4=1$  มาด้านหน้า และ 0 ถึง 7 ว่าเป็นการขยายบิตเครื่องหมาย  $x_4=0$  มาด้านหน้า โดยไม่ทำให้ค่าฐานสิบเปลี่ยนแปลง

### 2.2.3 ชนิดมีเครื่องหมาย (Signed Integer) แบบ Sign-Magnitude

ตัวเลขແຕวล่างสุดในรูปที่ 2.3 แสดงค่าจำนวนเต็มบนเส้นจำนวนตามการตีความแบบ Sign/Magnitude ความยาว  $n=4$  บิต เราสามารถนิยามได้ดังนี้

**นิยามที่ 2.2.3** กำหนดให้ เลขจำนวนเต็มฐานสองชนิดมีเครื่องหมาย (Signed Integer) แบบ Sign-Magnitude  $X_{2,sm}$  ความยาว  $n$  บิตเขียนอยู่ในรูป

$$X_{2,sm} = sx_{n-2}x_{n-3} \dots x_1x_0 \quad (2.34)$$

เมื่อ  $s$  คือบิตเครื่องหมาย (Sign bit) และ  $x_i$  คือค่า “1” หรือ “0” ในตำแหน่งที่  $i$  และตำแหน่งของขวามือสุดคือตำแหน่งที่  $i = 0$

การแปลงเลขจำนวนเต็มฐานสองแบบ Sign-Magnitude ให้เป็นค่าฐานสิบสามารถทำได้โดย

$$X_{10,sm} = (-1)^s \times (x_{n-2} \times 2^{n-2} + \dots + x_1 \times 2^1 + x_0 \times 2^0) \quad (2.35)$$

ดังนั้น ค่าฐานสิบ  $X_{10,sm}$  อยู่ในช่วง  $-(2^{n-1}-1)$  ถึง  $+(2^{n-1}-1)$

จากนิยามที่ 2.2.1 และนิยามที่ 2.2.3 เลขจำนวนเต็มฐานสองชนิดมีเครื่องหมาย แบบ Sign Magnitude สามารถเขียนใหม่ในรูปของเลขฐานสองแบบไม่มีเครื่องหมายที่ความยาว  $n - 1$  บิต ดังนี้

$$X_{2,sm} = \pm X_{2,u} \quad (2.36)$$

ดังนั้น คำว่า Magnitude แปลว่า ขนาด จึงหมายถึง ค่าฐานสิบของเลขฐานสองชนิดไม่มีเครื่องหมายในสมการนี้ หลักการนี้สามารถนำไปประยุกต์ใช้กับเลขทศนิยมทั้งชนิดจุดตรึง (Fixed-Point) และชนิดจุดลอยตัว (Floating-Point)

**ตัวอย่างที่ 2.2.12** เลขจำนวนเต็มฐานสองชนิดมีเครื่องหมายแบบ Sign Magnitude  $n = 4$  บิต  $X_{2,sm} = 1011_2$  ค่าฐานสิบของ  $X_{2,sm}$  คือ

$$X_{10,sm} = (-1)^1 \times \{0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0\} \quad (2.37)$$

$$= -(0 + 2 + 1) \quad (2.38)$$

$$= -3_{10} \quad (2.39)$$

เลขจำนวนเต็มแบบ Sign-Magnitude ยาว  $n = 4$  บิตจะมีค่าฐานสิบอยู่ในช่วง  $-7$  ถึง  $+7$

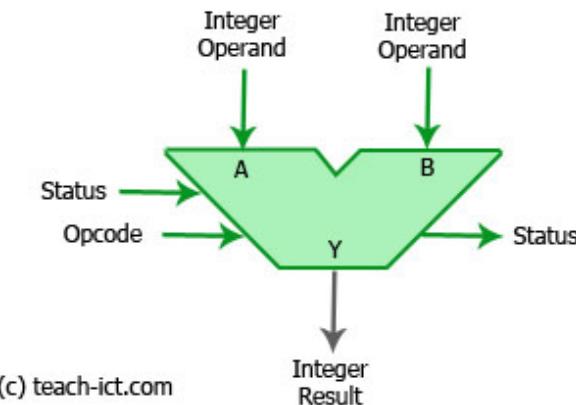
โดยสรุป จากรูปที่ 2.3 ตอนต้นของบทนี้ เลขฐานสองความยาว  $n=4$  บิต จำนวน  $2^4=16$  เลข สามารถตีความ เป็นเลขจำนวนเต็มได้สามแบบ ดังนี้ แบบมีเครื่องหมายชนิด Sign-Magnitude แบบมีเครื่องหมายชนิด 2's Complement และแบบไม่มีเครื่องหมาย ตามสมการที่ (2.35), (2.16) และ (2.2) ตามลำดับในตารางที่ 2.9

ตารางที่ 2.9: เลขจำนวนเต็มฐานสองความยาว  $n=4$  บิตทั้งหมด  $2^4=16$  แบบ สามารถตีความเป็นเลขจำนวนเต็มฐานสิบแบบมีเครื่องหมายนิด Sign-Magnitude, แบบมีเครื่องหมายนิด 2's Complement, และแบบไม่มีเครื่องหมาย (Unsigned)

เลขฐานสอง $n=4$ บิต	$X_{10,sm}$ ค่าฐานสิบ Sign-Mag. สมการ (2.35)	$X_{10,s}$ ค่าฐานสิบ 2-Comp. สมการ (2.16)	$X_{10,u}$ ค่าฐานสิบ Unsigned สมการ (2.2)
1111	-7	-1	15
1110	-6	-2	14
1101	-5	-3	13
1100	-4	-4	12
<b>1011</b>	<b>-3</b>	<b>-5</b>	<b>11</b>
1010	-2	-6	10
1001	-1	-7	9
1000	-0	-8	8
0000	+0	0	0
0001	1	1	1
0010	2	2	2
0011	3	3	3
0100	4	4	4
0101	5	5	5
0110	6	6	6
0111	7	7	7

รูปแบบของ Sign-Magnitude นิยมใช้งานร่วมกับข้อมูลชนิดเลขทศนิยม ซึ่งจะกล่าวต่อไปในหัวข้อที่ 2.4 และหัวข้อที่ 2.5 เพื่อนำไปประยุกต์ใช้กับเลขทศนิยมฐานสองชนิดจุดลอยตัวมาตรฐาน IEEE754

## 2.3 คณิตศาสตร์เลขจำนวนเต็มฐานสอง



รูปที่ 2.4: สัญลักษณ์ของ ALU (Arithmetic Logic Unit) สำหรับบวก/ลบเลขจำนวนเต็มขนาด  $n$  บิตชนิดไม่มีเครื่องหมาย ที่มา: [teach-ict.com](http://teach-ict.com)

การคำนวณทางคณิตศาสตร์และตรรกศาสตร์ของเลขจำนวนเต็มฐานสองขนาด  $n=8, 16, 32, 64$  และ  $128$  บิตจำเป็นต้องอาศัยวงจรดิจิทัล เรียกว่า **วงจร ALU** (Arithmetic Logic Unit) เพื่อการทำงานที่รวดเร็ว รูปที่ 2.4 แสดงสัญลักษณ์ของวงจร ALU สำหรับเลขจำนวนเต็มฐานสองขนาด  $n$  บิตชนิดไม่มีเครื่องหมาย ประกอบด้วย

- ขาสัญญาณอินพุต (Input) ข้อมูล A และ B ขนาด  $n$  บิต สำหรับนำข้อมูลฐานสอง โดยเริ่มนับจากบิตที่ 0 ถึงบิตที่  $n-1$
- สัญญาณอินพุต ชื่อ Opcode เพื่อสั่งการทำงาน เช่น บวก ลบ เป็นต้น
- สัญญาณเอาต์พุต ชื่อ Y เป็นผลลัพธ์เลขฐานสองจำนวนเต็มชนิดไม่มีเครื่องหมายขนาด  $n$  บิต
- สัญญาณเอาต์พุต Status ประกอบด้วย
  - ขาสัญญาณ ชื่อ Negative ( $N$ ) สำหรับเลขจำนวนเต็มชนิดมีเครื่องหมายเท่านั้น
  - ขาสัญญาณ ชื่อ Zero ( $Z$ ) = 1 เพื่อบ่งบอกว่าผลลัพธ์ Y มีค่าเท่ากับศูนย์ทุกบิต
  - ขาสัญญาณ ชื่อ Carry ( $c_n$  หรือ  $C$ ) สำหรับบิตตัวท้ายที่  $n$
  - ขาสัญญาณ ชื่อ Overflow ( $V$ ) เพื่อบ่งบอกความผิดพลาด โดย
    - \*  $V=0$  หมายถึง ไม่เกิดโอเวอร์เฟลว์ ซอฟต์แวร์สามารถนำค่านี้ไปใช้งานได้
    - \*  $V=1$  หมายถึง เกิดโอเวอร์เฟลว์ทำให้ผลลัพธ์มีความผิดพลาด (Invalid) ซอฟต์แวร์ไม่สามารถนำค่านี้ไปใช้งานได้
    - \* สำหรับเลขจำนวนเต็มชนิดไม่มีเครื่องหมายสามารถตรวจสอบโดยใช้สมการที่ (2.43)
    - \* สำหรับเลขจำนวนเต็มชนิดมีเครื่องหมายแบบ 2's Complement สามารถตรวจสอบโดยใช้สมการที่ (2.47)

ขาสัญญาณเอาต์พุตเหล่านี้จะบันทึกลงในรีจิสเตอร์สถานะ (Status Register) สำหรับให้ wang และโปรแกรมเมอร์ตรวจสอบด้วยวงจรดิจิทัลและคำสั่งภาษาแอสเซมบลี ในบทที่ 4

### 2.3.1 ชนิดไม่มีเครื่องหมาย

เลขจำนวนเต็มฐานสองชนิดไม่มีเครื่องหมาย  $X_{2,u}$  และ  $Y_{2,u}$  ความยาว  $n$  บิต ตามนิยามที่ 2.2.1 สามารถบวกกันได้ การบวกเลขจำนวนเต็มฐานสองขนาด  $n$  บิตแบบไม่มีเครื่องหมาย: โดยจะได้ผลลัพธ์  $Z_{2,u} = X_{2,u} + Y_{2,u}$  พร้อมตัวทด  $c_i$  ดังนี้

	$c_n$	$c_{n-1}$	$c_{n-2}$	..	$c_2$	$c_1$	$c_0$	
$X_{2,u} +$		$x_{n-1}$	$x_{n-2}$	..	$x_2$	$x_1$	$x_0$	+
$Y_{2,u}$		$y_{n-1}$	$y_{n-2}$	..	$y_2$	$y_1$	$y_0$	
$Z_{2,u}$		$z_{n-1}$	$z_{n-2}$	..	$z_2$	$z_1$	$z_0$	

การบวกเลขจำนวนเต็มฐานสองเหมือนกับการบวกเลขจำนวนเต็มฐานสิบ โดยจะเริ่มจากบิตที่มีนัยสำคัญต่ำที่สุด (Least Significant Bit: LSB) คือ บิตที่ 0 โดย  $c_0=0$  เสมอ ทำให้  $x_0+y_0+c_0$  ได้บิต  $z_0$  และเกิดตัวทด  $c_1$  ในวงจรดิจิทัล การบวกเลขจำนวนเต็มฐานสองจำนวน 3 บิตเข้าด้วยกัน จะทำให้เกิดผลลัพธ์จำนวน 2 บิต เรียกว่า ติดกัน คือ

$$c_{i+1}z_i = x_i + y_i + c_i \quad (2.40)$$

เมื่อ  $i=0, 1, 2, \dots, n-1$  โดย  $c_0 = 0$  และสัญลักษณ์ + คือการบวกเลข ไม่ใช่การ OR กันเชิงตรรกศาสตร์

ในวิชาออกแบบวงจรดิจิทัล เราเรียกว่างบวกเลขชนิดนี้ว่า วงจร Full Adder โดยวงจรจะนำบิตข้อมูลจำนวน 3 บิตมากระทำการทางตรรกศาสตร์ด้วยพิชคงตบูลลีน ได้ผลลัพธ์  $z_i$  โดย

$$z_i = x_i \oplus y_i \oplus c_i \quad (2.41)$$

เมื่อ  $\oplus$  คือ กระบวนการ Exclusive-OR และบิตตัวทด  $c_{i+1}$

$$c_{i+1} = (x_i \& y_i) | (x_i \& c_i) | (y_i \& c_i) \quad (2.42)$$

เมื่อ & คือ กระบวนการ AND และ | คือ กระบวนการ OR วงจรบวกเลขชนิดไม่มีเครื่องหมายขนาด  $n$  บิตนี้ สามารถตรวจจับการเกิดโอเวอร์โฟล์วได้โดย

$$V = c_n \quad (2.43)$$

### การบวกเลขจำนวนเต็มชนิดไม่มีเครื่องหมาย

การบวกเลขจำนวนเต็มชนิดไม่มีเครื่องหมาย 2 จำนวน ผลลัพธ์ที่ได้จะไม่มีเครื่องหมายด้วยเข่นกัน แต่การบวกเลขขนาดใหญ่ที่เข้าใกล้ค่าสูงสุด สามารถเกิดความผิดพลาดได้ เรียกว่า การเกิดโอเวอร์โฟล์ว (Overflow) ในสมการที่ (2.43) ซึ่งเป็นผลสืบเนื่องจากการดิจิทัลที่สามารถประมวลผลได้จำกัด ตามจำนวนบิตข้อมูลสูงสุด ที่ทำได้ ในตัวอย่างการแปลงเลขฐานสองเป็นฐานสิบที่ได้แสดงไปแล้ว ยกตัวอย่าง เช่น การวนรอบหรือวนลูป (Loop) เพิ่มค่าอย่างต่อเนื่องโดยไม่ระวัง ตามภาษา C/C++ ประযุค  $i++$  หรือ  $i=i+1$  นี้ หาก  $i$  มีค่าเพิ่มขึ้นเรื่อยๆ จนถึงค่าสูงสุด การบวกเพิ่มอีก 1 ไปเรื่อยๆ โดยไม่มีการตรวจจับการเกิดโอเวอร์โฟล์วล่วงหน้า จะทำให้ค่าของ  $i$  กลับเป็นศูนย์ในที่สุด ซึ่งอาจทำให้เกิดผลร้ายตามมาอย่างรุนแรง ผู้อ่านสามารถทดสอบได้ตามกิจกรรมท้ายการทดลองในภาคผนวก E

ตัวอย่างที่ 2.3.1 จงคำนวณหาค่าของ  $5 + 9$  ด้วยเลขจำนวนเต็มชนิดมีเครื่องหมาย แบบ Unsigned ขนาด 4 บิต  $5 + 9 = 14$  ดังนั้น ในเครื่องคอมพิวเตอร์ขนาด 4 บิต สามารถคำนวณได้ดังนี้

การบวกเลขขนาด 4 บิตแบบไม่มีเครื่องหมาย:  $5 + 9 = 14$  พร้อมตัวหนา และผลลัพธ์ถูกต้องเนื่องจากไม่เกิดโอเวอร์โฟล์ว ( $V=c_n=0$ )

	$c_4$	$c_3$	$c_2$	$c_1$	$c_0$	Overflow=False
	0	0	0	1	0	$V=c_n=0$
$X=5 +$	0	1	0	1		+
$Y=9$	1	0	0	1		
$Z=14$	1	1	1	0		

ตัวอย่างที่ 2.3.2 จงคำนวณหาค่าของ  $7 + 9$  ด้วยเลขจำนวนเต็มชนิดมีเครื่องหมาย แบบ Unsigned ขนาด 4 บิต  $7 + 9 = 16 = 0$  ดังนั้น ในเครื่องคอมพิวเตอร์ขนาด 4 บิต ซึ่งไม่สามารถแสดงผลค่า  $16_{10}$  ได้ดังนี้

	$c_4$	$c_3$	$c_2$	$c_1$	$c_0$	Overflow=True
	1	1	1	1	0	$V=c_n=1$
$X=7 +$	0	1	1	1		+
$Y=9$	1	0	0	1		
$Z=16$	0	0	0	0		

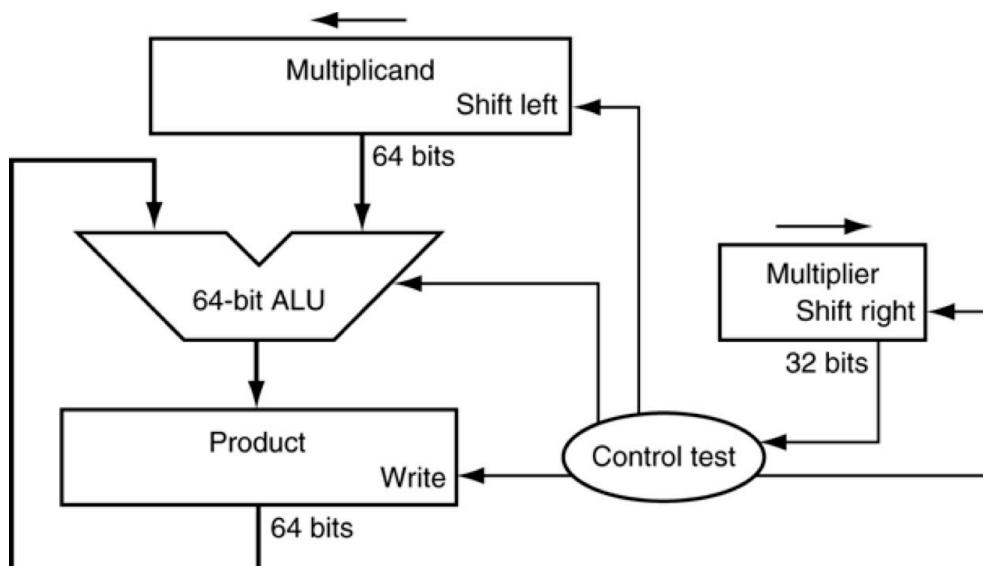
สาเหตุของการเกิด Overflow เนื่องจากผลลัพธ์มีค่าอยู่นอกย่านที่เป็นไปได้ โดยสามารถตรวจสอบอย่างง่ายดายโดย  $V=c_4 = 1$  ( $V$ : Overflow) ตามสมการที่ (2.43) เมื่อเกิดโอเวอร์โฟล์ว ผลลัพธ์ที่ได้จึงมีค่าไม่ถูกต้อง (Invalid) ทำให้ซอฟต์แวร์ไม่สามารถนำค่าผลลัพธ์นี้ไปใช้

## การคูณเลขจำนวนเต็มชนิดไม่มีเครื่องหมาย

การคูณเลขจำนวนเต็มชนิดไม่มีเครื่องหมายมีความสำคัญต่อการคำนวณทางคณิตศาสตร์ต่างๆ และเป็นพื้นฐานของการคำนวณเลขชนิดอื่นๆ ซึ่งการคูณเลขมีความซับซ้อน (Complexity) มากกว่าการบวกเลขหลายเท่า เพื่อให้ผู้อ่านตระหนักรู้ดิจิทัลที่จำเป็น และเวลาที่ซอฟต์แวร์ใช้คำนวณ ตัวราชเล่มนี้จึงขอวางแผนพื้นฐานของกระบวนการคูณเลขจำนวนเต็มชนิดไม่มีเครื่องหมายให้ผู้อ่านแต่เพียงเบื้องต้น

การคูณเลขฐานสองชนิดไม่มีเครื่องหมายขนาด  $n$  บิต 2 จำนวน จะได้ผลลัพธ์เป็นเลขฐานสองชนิดไม่มีเครื่องหมายเช่นกัน แต่ต้องการจำนวนบิตเพื่อจัดเก็บเพิ่มขึ้นเป็น  $2n$  บิต ยกตัวอย่าง เช่น  $1111_2 \times 1111_2 (15_{10} \times 15_{10})$  จะได้ผลลัพธ์เท่ากับ  $1110\ 0001_2 = 225_{10} = 128_{10} + 64_{10} + 32_{10} + 1_{10}$

วงจรคูณเลขจำนวนเต็มไม่มีเครื่องหมายชนิดที่ 1 การคูณเลขมีความซับซ้อนสูงจำเป็นต้องใช้วงจรดิจิทัล และเวลาในคำนวณ การคูณเลขที่ง่ายที่สุดคือ การบวกเลขแล้ววนบวกซ้ำ ตามวงจรที่จะอธิบายในรูปที่ 2.5 และ 2.6 ต่อไปนี้



รูปที่ 2.5: วงจรคูณเลขขนาด  $n=32$  บิต ชนิดที่ 1 โดยใช้ ALU ขนาด  $2n = 64$  บิต และรีจิสเตอร์ตัวตั้งขนาด  $2n = 64$  บิต ที่มา: [Patterson and Hennessy \(2016\)](#)

วงจรคูณเลขจำนวนเต็มฐานสองไม่มีเครื่องหมายในรูปที่ 2.5 ประกอบด้วย รีจิสเตอร์ตัวคูณ (Multiplier Register) เก็บเลขจำนวนเต็มไม่มีเครื่องหมายขนาด  $n$  บิต รีจิสเตอร์ตัวตั้ง (Multiplicand Register) และรีจิสเตอร์ผลคูณ (Product Register) จะมีขนาด  $2n$  บิตทั้งคู่ มีการทำงานเป็นขั้นตอนดังนี้

1. ตั้งค่าเริ่มต้นของรีจิสเตอร์ทุกตัว (Initialize)
  - ตั้งค่ารีจิสเตอร์ตัวตั้งขนาด  $2n$  บิต ตามค่าตัวตั้งโดยให้บิตขวาสุดของตัวตั้งอยู่ที่บิต 0 ของรีจิสเตอร์
  - ตั้งค่ารีจิสเตอร์ตัวคูณขนาด  $n$  บิตตามค่าตัวคูณ
  - ตั้งค่ารีจิสเตอร์ผลคูณขนาด  $2n$  บิตให้เป็น 0 ทุกบิต
2. ตรวจสอบค่าบิตที่ 0 (ขวาสุด) ของรีจิสเตอร์ตัวคูณ
  - หากมีค่าเป็น 1 บวกค่าตัวตั้งกับค่าผลคูณ ( $n$  บิตซ้ายสุด) เข้าด้วยกัน (Action = Add)
  - หากมีค่าเป็น 0 ไม่ต้องบวก (Action = None)

## 3. เลื่อน (Shift) ข้อมูลในรีจิสเตอร์

- เลื่อนรีจิสเตอร์ตัวตั้งไปทางซ้าย 1 บิต โดยป้อน 0 เข้าทางขวาไปแทนที่บิตที่ถูกเลื่อนไปทางซ้าย
- เลื่อนรีจิสเตอร์ตัวคูณไปทางขวา 1 บิต โดยป้อน 0 เข้าทางซ้ายไปแทนที่บิตที่ถูกเลื่อนไปทางขวา

4. กลับไปทำข้อ 2 จนครบ  $n$  รอบ

**ตัวอย่างที่ 2.3.3** จงคูณ  $1010_2$  ( $10_{10}$ ) ด้วย  $1101_2$  ( $13_{10}$ ) ตามวิธีในรูปที่ 2.5

ผลคูณเลขฐานสิบ 4 บิต ตัวตั้ง= $1010_2$  ( $10_{10}$ ) ตัวคูณ= $1101_2$  ( $13_{10}$ ) เท่ากับ  $130_{10}$  ด้วยวิธีในรูปที่ 2.5

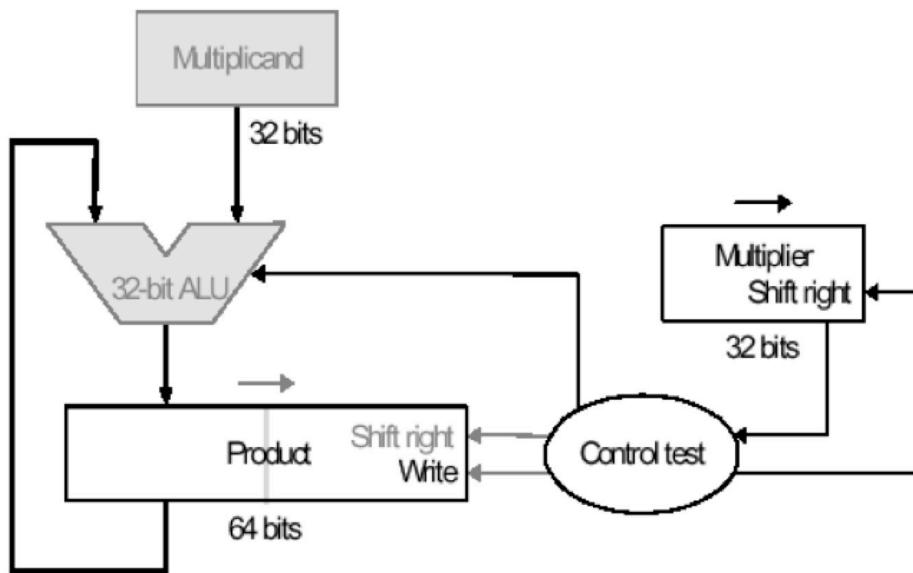
**ตารางที่ 2.10:** ผลคูณเลขฐานสิบ 4 บิต ตัวตั้ง= $1010_2$  ( $10_{10}$ ) ตัวคูณ= $1101_2$  ( $13_{10}$ ) เท่ากับ  $130_{10}$  ด้วยวิธีในรูปที่ 2.5 หมายเหตุ Shift คือ การเลื่อนบิต

รอบ (Iteration)	ตัวตั้ง (Multiplicand)	ตัวคูณ (Multiplier)	ผลคูณ <sub>2</sub> (Product <sub>2</sub> )	ผลคูณ <sub>10</sub> (Product <sub>10</sub> )	กระทำ (Action)
-	0000 1010 <sub>2</sub>	1101 <sub>2</sub>	0000 0000 <sub>2</sub>	$0_{10}$	Initialized
0	0000 1010 <sub>2</sub>	1101 <sub>2</sub>	0000 1010 <sub>2</sub>	$10_{10}$	Add
	0001 0100 <sub>2</sub>	0110 <sub>2</sub>	0000 1010 <sub>2</sub>	$10_{10}$	Shift
1	0001 0100 <sub>2</sub>	0110 <sub>2</sub>	0000 1010 <sub>2</sub>	$10_{10}$	None
	0010 1000 <sub>2</sub>	0011 <sub>2</sub>	0000 1010 <sub>2</sub>	$10_{10}$	Shift
2	0010 1000 <sub>2</sub>	0011 <sub>2</sub>	0011 0010 <sub>2</sub>	$50_{10}$	Add
	0101 0000 <sub>2</sub>	0001 <sub>2</sub>	0011 0010 <sub>2</sub>	$50_{10}$	Shift
3	0101 0000 <sub>2</sub>	0001 <sub>2</sub>	1000 0010 <sub>2</sub>	$130_{10}$	Add
	1010 0000 <sub>2</sub>	0000 <sub>2</sub>	1000 0010 <sub>2</sub>	$130_{10}$	Shift

หมายเหตุ ผู้เขียนจะใช้วิธีนี้ของวิธีทั่วไป 4 บิต เพื่อให้ผู้อ่านนับเลขฐานสองและแปลงเป็นเลขฐานสิบให้ได้ง่าย

เราจะเห็นว่าการคูณคือ การบวกตัวตั้ง (Multiplicand) ที่ถูกเลื่อนบิตไปทางซ้ายรอบละ 1 บิต กับผลคูณในรอบก่อนหน้า โดยจำนวนรอบขึ้นอยู่กับจำนวนบิตของตัวคูณ (Multiplier) ในวิธีนี้จะใช้วิธีการที่ใช้ ALU ขนาดเล็กลง รีจิสเตอร์บางตัวลื้นลง และใช้จำนวนรอบเท่าเดิม

## วงจรคูณเลขจำนวนเต็มฐานสองไม่มีเครื่องหมายชนิดที่ 2



รูปที่ 2.6: วงจรคูณเลขขนาด  $n=32$  บิต ชนิดที่ 2 โดยใช้ ALU ขนาด  $n=32$  บิต และรีจิสเตอร์ตัวตั้งขนาด  $n=32$  บิต ที่มา: [Patterson and Hennessy \(2016\)](#)

วงจรในรูปที่ 2.6 มีการปรับเปลี่ยนให้รีจิสเตอร์ตัวตั้งและรีจิสเตอร์ตัวคูณเป็นเลขจำนวนเต็มไม่มีเครื่องหมายขนาด  $n$  บิตทั้งคู่ และรีจิสเตอร์ผลคูณจะมีขนาด  $2n$  บิต มีการทำงานเป็นขั้นตอนดังนี้

1. ตั้งค่าเริ่มต้นของรีจิสเตอร์ทุกตัว
  - ตั้งค่ารีจิสเตอร์ตัวตั้งขนาด  $n$  บิต ตามค่าตัวตั้งโดยให้บิตขวาสุดของตัวตั้งอยู่ที่บิต 0 ของรีจิสเตอร์
  - ตั้งค่ารีจิสเตอร์ตัวคูณขนาด  $n$  บิตตามค่าตัวคูณ
  - ตั้งค่ารีจิสเตอร์ผลคูณขนาด  $2n$  บิตให้เป็น 0 ทุกบิต
2. ตรวจสอบค่าบิตที่ 0 (ขวาสุด) ของรีจิสเตอร์ตัวคูณ
  - หากมีค่าเป็น 1 บวกค่าตัวตั้งกับ  $n$  บิต ทางซ้ายของค่าผลคูณเข้าด้วยกัน (Action = Add)
  - หากมีค่าเป็น 0 ไม่ต้องบวก (Action = None)
3. เลื่อนข้อมูลในรีจิสเตอร์เหล่านี้ไปทางขวา
  - เลื่อนรีจิสเตอร์ตัวคูณและรีจิสเตอร์ผลคูณไปทางขวา 1 บิต (Shift R)
  - ป้อน 0 เข้าทางซ้ายของรีจิสเตอร์ตัวคูณ เพื่อไปแทนที่บิตที่ถูกเลื่อนไป
  - ป้อนค่า  $c_n$  เข้าทางซ้ายของรีจิสเตอร์ผลคูณ เพื่อไปแทนที่บิตที่ถูกเลื่อน
4. กลับไปทำข้อ 2 จนครบ  $n$  รอบ

ตัวอย่างที่ 2.3.4 จงคูณ  $1010_2$  ( $10_{10}$ ) ด้วย  $1101_2$  ( $13_{10}$ ) ตามวิธีในรูปที่ 2.6

ผลคูณเลขฐาน 4 บิต ตัวตั้ง=  $1010_2$  ( $10_{10}$ ) ตัวคูณ=  $1101_2$  ( $13_{10}$ ) เท่ากับ  $130_{10}$  ด้วยวิธีในรูปที่ 2.6

ตารางที่ 2.11: ผลคูณเลขฐาน 4 บิต ตัวตั้ง=  $1010_2$  ( $10_{10}$ ) ตัวคูณ=  $1101_2$  ( $13_{10}$ ) เท่ากับ  $130_{10}$  ด้วยวิธีในรูปที่ 2.6 หมายเหตุ Shift R คือ การเลื่อนบิตไปทางขวา

รอบ (Iteration)	ตัวตั้ง (Multiplicand)	ตัวคูณ (Multiplier)	ผลคูณ <sub>2</sub> (Product <sub>2</sub> )	ผลคูณ <sub>10</sub> (Product <sub>10</sub> )	กระทำ (Action)
-	$1010_2$	$1101_2$	0000 0000 <sub>2</sub>	$0_{10}$	Initialized
0	$1010_2$	$1101_2$	1010 0000 <sub>2</sub>	$160_{10}$	Add
	$1010_2$	$0110_2$	0101 0000 <sub>2</sub>	$80_{10}$	Shift R
1	$1010_2$	$0110_2$	0101 0000 <sub>2</sub>	$80_{10}$	None
	$1010_2$	$0011_2$	0010 1000 <sub>2</sub>	$40_{10}$	Shift R
2	$1010_2$	$0011_2$	1100 1000 <sub>2</sub>	$200_{10}$	Add
	$1010_2$	$0001_2$	0110 0100 <sub>2</sub>	$100_{10}$	Shift R
3	$1010_2$	$0001_2$	1 0000 0100 <sub>2</sub>	$260_{10}$	Add
	$1010_2$	$0000_2$	1000 0010 <sub>2</sub>	$130_{10}$	Shift R

หมายเหตุ ผู้เขียนจะใจเว้นช่องว่างทุกๆ 4 บิต เพื่อให้ผู้อ่านนับเลขฐานสองและแบ่งเป็นเลขฐานสิบหากได้ยังงั้น

วงจรชนิดที่ 2 จะทำงานตรงข้ามกับวงจรชนิดที่ 1 คือ ทำการบวกเลขตัวตั้ง กับ  $n$  บิตทางซ้ายของผลคูณก่อนแล้วจึงเลื่อนผลคูณไปทางขวาแทน ทำให้ประหยัดขนาดวงจร ALU เหลือเพียงขนาด  $n$  บิต จะเห็นได้ชัดว่าที่รอบที่ 3 บิตที่เกิดขึ้นทางซ้ายสุดของผลคูณ จะถูกป้อนกลับเข้ามาทางซ้าย เพื่อให้ได้ผลลัพธ์ที่ถูกต้อง

วงจรคูณเลขจำนวนเต็มทั้งสองชนิดนี้ เป็นแค่ตัวอย่างเพื่อให้ผู้อ่านเข้าใจการแลกเปลี่ยน (Trade off) ระหว่างจำนวนรอบ (ระยะเวลา) กับ ความซับซ้อนของวงจร ซึ่งในทางปฏิบัติวงจรคูณจะมีความซับซ้อนมากกว่าแต่ ใช้ระยะเวลาสั้นกว่า และสามารถออกแบบให้ทำงานแบบไปปีลิน์ด้วย

### 2.3.2 ชนิดมีเครื่องหมายแบบ 2's Complement

เลขจำนวนเต็มฐานสองชนิดมีเครื่องหมายแบบ 2's Complement  $X_{2,s}$  และ  $Y_{2,s}$  ความยาว  $n$  บิต ตามนิยามที่ 2.2.2 สามารถบวกกันได้ การบวกเลขขนาด  $n$  บิตแบบมีเครื่องหมาย: โดยจะได้ผลลัพธ์ :  $Z_{2,s} = X_{2,s} + Y_{2,s}$  พร้อมตัวทด  $c_i$

	$c_n$	$c_{n-1}$	$c_{n-2}$	..	$c_2$	$c_1$	$c_0$	
$X_{2,s} +$	$x_{n-1}$	$x_{n-2}$	..	$x_2$	$x_1$	$x_0$	+	
$Y_{2,s}$	$y_{n-1}$	$y_{n-2}$	..	$y_2$	$y_1$	$y_0$		
$Z_{2,s}$	$z_{n-1}$	$z_{n-2}$	..	$z_2$	$z_1$	$z_0$		

การบวกเลขจำนวนเต็มฐานสองชนิดมีเครื่องหมายเหมือนกับการบวกเลขฐานสิบชนิดมีเครื่องหมาย โดยจะเริ่มจากบิตที่มีนัยสำคัญต่ำที่สุด (Least Significant Bit: LSB) คือ บิตที่ 0 โดย  $c_0=0$  เสมอ และทำให้  $x_0+y_0+c_0$  ได้บิต  $z_0$  และเกิดตัวทด  $c_1$  ในวงจรดิจิทัล การบวกเลขฐานสองจำนวน 3 บิตเข้าด้วยกัน จะทำให้เกิดผลลัพธ์จำนวน 2 บิต เรียงติดกัน คือ

$$c_{i+1}z_i = x_i + y_i + c_i \quad (2.44)$$

เมื่อ  $i=0, 1, 2, \dots, n-1$  โดย  $c_0 = 0$

ผลลัพธ์ของการบวกเลขฐานสองจำนวน 3 บิต สามารถคำนวณได้จากการ Full Adder ดังนี้

$$z_i = x_i \oplus y_i \oplus c_i \quad (2.45)$$

เมื่อ  $\oplus$  คือ กระบวนการ Exclusive-OR

$$c_{i+1} = (x_i \& y_i) | (x_i \& c_i) | (y_i \& c_i) \quad (2.46)$$

เมื่อ  $\&$  คือ กระบวนการ AND และ  $|$  คือ กระบวนการ OR

การเกิดโอลู์ฟล์ของ การบวกเลขชนิดมีเครื่องหมาย 2's Complement ที่มา: Mano and Kime (2007) ได้โดย

$$V = c_n \oplus c_{n-1} \quad (2.47)$$

ขาสัญญาณเหล่านี้จะบันทึกลงในรีจิสเตอร์สถานะ สำหรับให้โปรแกรมเมอร์ตรวจสอบ ด้วยภาษาแอสเซมบลี ในบทที่ 4 หัวข้อที่ 4.6.1 โปรดสังเกตความแตกต่างระหว่างการตรวจสอบโอลู์ฟล์ของเลขจำนวนเต็มฐานสองมีเครื่องหมายในสมการที่ (2.47) และเลขจำนวนเต็มฐานสองไม่มีเครื่องหมายในสมการที่ (2.43)

## การบวก/ลบเลขจำนวนเต็มชนิดมีเครื่องหมาย

การบวก/ลบเลขจำนวนเต็มชนิดมีเครื่องหมาย 2 จำนวน ผลลัพธ์ที่ได้จะมีเครื่องหมายด้วยเช่นกัน แต่การบวก/ลบเลขขนาดใหญ่ที่เข้าใกล้ค่าสูงสุดหรือค่าต่ำสุด สามารถเกิดความผิดพลาดได้ เรียกว่า การเกิดโอเวอร์ไฟล์ (Overflow) ในสมการที่ (2.47) ซึ่งเป็นผลสืบเนื่องจากวงจรติจิทัลที่สามารถประมวลผลได้จำกัดตามจำนวนบิตข้อมูลสูงสุดที่ทำได้ ยกตัวอย่าง เช่น การวนรอบหรือวนลูป (Loop) เพิ่มค่าอย่างต่อเนื่องโดยไม่ระวัง ตามประโยชน์ในภาษา C/C++ ประโยชน์  $i++$  หรือ  $i = i + 1$  นี้ หาก  $i$  มีค่าเพิ่มขึ้นเรื่อยๆ จนถึงค่าสูงสุด การบวกเพิ่มอีก 1 ไปเรื่อยๆ โดยไม่มีการตรวจสอบการเกิดโอเวอร์ไฟล์ล่วงหน้า จะทำให้ค่าของ  $i$  กลายเป็นค่าลบในที่สุด ซึ่งอาจทำให้เกิดผลลัพธ์ตามมาอย่างรุนแรง ผู้อ่านสามารถทดสอบได้ตามกิจกรรมท้ายการทดลองในภาคผนวก E

**ตัวอย่างที่ 2.3.5** จงคำนวณหาค่าของ  $7 + 3$  ด้วยเลขจำนวนเต็มชนิดมีเครื่องหมาย แบบ 2's Complement ขนาด 4 บิต ดังนี้ ในเครื่องคอมพิวเตอร์ขนาด 4 บิต สามารถคำนวณได้ดังนี้

	$c_4$	$c_3$	$c_2$	$c_1$	$c_0$	Overflow=True $V=0 \oplus 1=1$
	0	1	1	1	0	
$X = 7$		0	1	1	1	+
$+Y = +3$		0	0	1	1	
$Z = -6$		1	0	1	0	

ผลการคำนวณผลลัพธ์ที่ไม่ถูกต้อง (Invalid) เนื่องจากเกิดโอเวอร์ไฟล์ (Overflow) เพราะชาร์ดแวร์คำนวณ  $V=c_4 \oplus c_3 = 0 \oplus 1 = 1$  ตามสมการที่ (2.47) ทำให้ซอฟต์แวร์นำคำตอบนี้ไปใช้ไม่ได้

**ตัวอย่างที่ 2.3.6** จงคำนวณหาค่าของ  $7 - 6$  ด้วยเลขจำนวนเต็มชนิดมีเครื่องหมาย แบบ 2's Complement ขนาด 4 บิต  $7 - 6 = 7 + (-6)$  ดังนี้ ในเครื่องคอมพิวเตอร์ขนาด 4 บิต สามารถคำนวณได้ดังนี้

	$c_4$	$c_3$	$c_2$	$c_1$	$c_0$	Overflow=False $V=1 \oplus 1=0$
	1	1	1	0	0	
$X = 7 +$		0	1	1	1	+
$-Y = -6$		1	0	1	0	
$Z = 1$		0	0	0	1	

ผลการคำนวณผลลัพธ์ที่ได้ถูกต้อง (Valid) เนื่องจากไม่เกิดโอเวอร์ไฟล์ (Overflow) เพราะชาร์ดแวร์คำนวณ  $V=c_4 \oplus c_3 = 1 \oplus 1 = 0$  ตามสมการที่ (2.47) ทำให้ซอฟต์แวร์นำคำตอบนี้ไปใช้ได้

**ตัวอย่างที่ 2.3.7** จงคำนวณหาค่าของ  $-3 - 6$  ด้วยเลขจำนวนเต็มชนิดมีเครื่องหมาย แบบ 2's Complement ขนาด 4 บิต  $-3 - 6 = (-3) + (-6)$  ดังนี้ ในเครื่องคอมพิวเตอร์ขนาด 4 บิต ดังนี้

	$c_4$	$c_3$	$c_2$	$c_1$	$c_0$	Overflow=True $V=1 \oplus 0=1$
	1	0	0	0	0	
$X = -3 +$		1	1	0	1	+
$-Y = -6$		1	0	1	0	
$Z = +7$		0	1	1	1	

ผลการคำนวณผลลัพธ์ที่ไม่ถูกต้อง (Invalid) เนื่องจากเกิดโอเวอร์ไฟล์ (Overflow) เพราะชาร์ดแวร์คำนวณ  $V=c_4 \oplus c_3 = 1 \oplus 0 = 1$  ตามสมการที่ (2.47) ทำให้ซอฟต์แวร์นำคำตอบนี้ไปใช้ไม่ได้

## 2.4 เลขทศนิยมฐานสองชนิดจุดตรึง (Fixed Point)

เลขจำนวนจริงแบ่งเป็นเลขทศนิยมชนิดจุดตรึง (Fixed-Point Real Number) และ ชนิดจุดทศนิยมโดยตัว (Floating-Point Real Number) ในหัวข้อนี้ ผู้อ่านจะได้ทำความเข้าใจเรื่องรูปแบบ (Format) ของเลขทศนิยมฐานสองชนิดจุดตรึง ซึ่งนิยมใช้ในตัวประมวลผลสัญญาณดิจิทัล (Digital Signal Processor) ส่วนการบวกและลบเลขทศนิยมฐานสองชนิดนี้จะใช้หลักการเดียวกับเลขจำนวนเต็ม Sign-Magnitude โดยการตรวจจับการเกิดโอลีฟอล์ว์สำหรับเลขที่มีค่าสัมบูรณ์มากๆ จนไม่สามารถคำนวณได้ และการเกิดอันเดอร์ฟอล์ว์สำหรับเลขที่มีค่าสัมบูรณ์น้อยๆ จนเกือบเป็นศูนย์

**นิยามที่ 2.4.1** กำหนดให้  $F_2$  เป็นเลขทศนิยมฐานสองชนิด Signed Magnitude เชียนอยู่ในรูป

$$F_2 = [s][x_{n-1}x_{n-2}x_{n-3}\dots x_1x_0].[y_{-1}y_{-2}y_{-3}\dots y_{-m}] \quad (2.48)$$

นิยมใช้ชนิดขนาด-เครื่องหมาย ประกอบด้วย 3 ส่วน คือ บิตเครื่องหมาย (Sign bit:  $s$  หรือ  $\pm$ ) โดย  $s=0$  แทนเครื่องหมายบวก และ  $s=1$  แทนเครื่องหมายลบ ส่วนจำนวนเต็ม (Integer:  $X_{2,u}$ ) มีความยาว  $n$  บิต และ ส่วนทศนิยม (Fraction:  $F_2$ ) ยาว  $m$  บิต รวมความยาวทั้งหมด  $m+n+1$  บิต

โดย  $F_{10}$  คือ ค่าฐานสิบของเลขทศนิยมฐานสองชนิดจุดตรึง  $F_2$  สามารถคำนวณได้จาก

$$F_{10} = (-1)^s \times [x_{n-1}2^{n-1} + \dots + x_12^1 + x_02^0 + y_{-1}2^{-1} + y_{-2}2^{-2} + y_{-3}2^{-3} + \dots + y_{-m}2^{-m}] \quad (2.49)$$

หรือ

$$F_{10} = (-1)^s \times [x_{n-1}2^{n-1} + \dots + x_12^1 + x_02^0 + \frac{y_{-1}}{2} + \frac{y_{-2}}{4} + \frac{y_{-3}}{8} + \dots + \frac{y_{-m}}{2^m}] \quad (2.50)$$

**ตัวอย่างที่ 2.4.1** จงแปลงเลขฐานสิบต่อไปนี้เป็นเลขทศนิยมฐานสองชนิดจุดตรึง  $m=n=2$  บิต

$$+0.75_{10} = 000.11_2 \quad (2.51)$$

$$+3.00_{10} = 011.00_2 \quad (2.52)$$

$$-3.75_{10} = 111.11_2 \quad (2.53)$$

ผู้อ่านสามารถสามารถเขียนเลขทศนิยมฐานสองชนิดจุดตรึงได้ตามรูปแบบนี้

$$F_2 = [s][X_{2,u}].[Y_2] \quad (2.54)$$

ค่าทศนิยม ( $Y_2$ ) มีความยาว  $m$  บิต เขียนเป็นสัญลักษณ์ได้ดังนี้

$$Y_2 = y_{-1}y_{-2}y_{-3}\dots y_{-m} \quad (2.55)$$

เมื่อจำนวนบิตหลังจุดทศนิยมเพิ่มขึ้น ความละเอียดจะเพิ่มขึ้นตามตัวอย่างต่อไปนี้

#### ตัวอย่างที่ 2.4.2 จงแปลงเลขทศนิยมฐานสองชนิดจุดตรึงต่อไปนี้ให้เป็นเลขฐานลิบ

$$0.1111_2 = 0.9375_{10} = 0.5 + 0.25 + 0.125 + 0.0625 \quad (2.56)$$

$$0.11111_2 = 0.96875_{10} = 0.5 + 0.25 + 0.125 + 0.0625 + 0.03125 \quad (2.57)$$

$$0.111111_2 = 0.984375_{10} = 0.5 + 0.25 + 0.125 + 0.0625 + 0.03125 + 0.015625 \quad (2.58)$$

เมื่อจำนวนบิตที่เป็น 1 เพิ่ม ค่าฐานสิบของเลขทศนิยมฐานสองนี้จะถูกเข้าหา 1.0 ยิ่งขึ้น ในขณะที่ การบวก/ลบ และการคูณเลขชนิดจุดตรึงมีความคล้ายกับเลขจำนวนเต็มชนิด Sign-Magnitude โดยรายละเอียดจะกล่าวในหัวข้อถัดไป

## 2.5 เลขทศนิยมฐานสองชนิดจุดลอยตัว (Floating Point)

เลขทศนิยมฐานสองชนิดชนิดจุดลอยตัวหมายความว่าสามารถตัวหารบวกหรือลบได้ทั้งหมด ซึ่งต้องการความละเอียดสูง สำหรับการคำนวณทางวิทยาศาสตร์ (Scientific) ดังนี้

- $-2.34 \times 10^{56}$  ซึ่งต้องเปลี่ยนอยู่ในลักษณะที่นอร์มัลไลซ์ (Normalize) แล้ว
- $+0.002 \times 10^{-4}$  ซึ่งจะต้องนอร์มัลไลซ์ต่อไปเป็น  $+2.000 \times 10^{-7}$
- $+987.02 \times 10^9$  ซึ่งจะต้องนอร์มัลไลซ์ต่อไปเป็น  $+9.8702 \times 10^{11}$

นิยามที่ 2.5.1 เลขทศนิยมชนิดจุดลอยตัวฐานสองที่อยู่ในรูปนอร์มัลไลซ์ ประกอบด้วย 3 ส่วน คือ บิตเครื่องหมาย (Sign bit:  $s$ ) ค่านัยสำคัญ (Significand:  $S_2$ ) ในสมการที่ (2.61) และค่ายกกำลัง (Exponent:  $E_2$ ) มีลักษณะดังนี้

$$(-1)^s \times [1.y_{-1}y_{-2}y_{-3}\dots y_{-m}]_2 \times 2^{E_2} \quad (2.59)$$

เลขยกกำลังเป็นเลขจำนวนเต็มฐานสองชนิด sign-magnitude ความยาว  $n$  บิต ดังนี้

$$E_2 = \pm[e_{n-1}e_{n-2}\dots e_0]_2 \quad (2.60)$$

เมื่อ  $e_i$  แต่ละบิตมีค่า “1” หรือ “0” ในตำแหน่งที่  $i$   $s$  คือ Sign bit และ  $n$  คือ จำนวนบิตซึ่งกำหนดไว้ก่อนจะออกแบบวงจร

จากนิยามที่ 2.59 ค่านัยสำคัญ (Significand)  $S_2$  ความยาว  $m+1$  บิต สามารถเขียนใหม่ได้ ดังนี้

$$S_2 = [1.y_{-1}y_{-2}y_{-3}\dots y_{-m}]_2 \quad (2.61)$$

ซึ่งมีความสำคัญต่อรูปแบบการเขียน เนื่องจากจะจะต้องทำการน้อมลัลไล์ช์ผลลัพธ์ที่ได้จากการคำนวณเสมอ ค่านัยสำคัญที่น้อมลัลไล์ช์ข้างต้นสามารถคำนวณหาค่าฐานสิบ ได้ดังนี้

$$S_{10} = (-1)^s \times [1 + \frac{y_{-1}}{2} + \frac{y_{-2}}{4} + \frac{y_{-3}}{8} + \dots + \frac{y_{-m}}{2^m}] \times (2^{\pm E_2}) \quad (2.62)$$

**ตัวอย่างที่ 2.5.1** จงแปลงเลขทศนิยมฐานสองชนิดจุดลอยตัวที่น้อมลัลไล์ช์แล้ว  $(-1)^1 \times 1.0101_2 \times 2^3$  ให้เป็นเลขทศนิยมฐานสิบตามสมการที่ (2.62)

วิธีทำ

- ปรับจุดทศนิยม เพื่อให้เป็นเลขฐานสองชนิด *Sign-Magnitude* และเลขยกกำลังเท่ากับ 0  $-1010.1_2 \times 2^0$

2. แปลงค่าเลขฐานสองชนิดที่เลื่อนตำแหน่งแล้วให้เป็นฐานสิบ

$$-\{1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 + 1 \times 2^{-1}\} = -\{8 + 0 + 2 + 0 + 0.5\} = -10.5$$

**ตัวอย่างที่ 2.5.2** จงแปลงเลขทศนิยมฐานสองชนิดจุดลอยตัวที่น้อมลัลไล์ช์แล้ว  $(-1)^0 \times 1.1010_2 \times 2^{-3}$  ให้เป็นเลขทศนิยมฐานสิบตามสมการที่ (2.62)

วิธีทำ

- ปรับจุดทศนิยม เพื่อให้เป็นเลขฐานสองชนิด *Sign-Magnitude* และเลขยกกำลังเท่ากับ 0  $+0.001101_2 \times 2^0$

2. แปลงค่าเลขฐานสองชนิดที่เลื่อนตำแหน่งแล้วให้เป็นฐานสิบ

$$(1 \times 2^{-3}) + (1 \times 2^{-4}) + (1 \times 2^{-6})$$

$$= 0.125 + 0.0625 + 0.015625$$

$$= 0.203125_{10}$$

### 2.5.1 การบวกเลขทศนิยมฐานสองชนิดจุดลอยตัว

เพื่อให้ผู้อ่านเข้าใจกลไกการบวกเลขทศนิยมฐานสองชนิดจุดลอยตัว โดยใช้ตัวอย่างจากการบวกเลขทศนิยมฐานสิบก่อน แล้วจึงนำขั้นตอนที่เหมือนกันไปประยุกต์ใช้กับการบวกเลขทศนิยมฐานสองในตัวอย่างต่อไป

**ตัวอย่างที่ 2.5.3** กำหนดให้เลขทศนิยมฐานสิบทั้งสองตัวมีขนาดไม่เกิน 4 หลัก (4-digit) จงบวกเลขฐานสิบชนิดจุดลอยตัวที่น้อมลัลไล์ช์แล้ว ดังต่อไปนี้  $9.999 \times 10^1 + 1.610 \times 10^{-1}$

วิธีทำ

- เลื่อนตำแหน่งจุดทศนิยมของเลขทั้งสองตัวให้มีค่าสัมบูรณ์มากกว่า ให้ตรงกับเลขยกกำลังของเลขที่มีค่าสัมบูรณ์มากกว่า  $9.999 \times 10^1 + 0.016 \times 10^1$

2. บวกค่านัยสำคัญที่เลื่อนตำแหน่งแล้วเข้าด้วยกัน จะได้ผลลัพธ์ดังนี้

$$(9.999 + 0.016) \times 10^1 = 10.015 \times 10^1$$

3. นอร์มัลไลซ์ค่าผลลัพธ์และตรวจเช็คการเกิดโอเวอร์โฟล์และอันเดอร์โฟล์

$$1.0015 \times 10^2$$

4. ปัดค่านัยสำคัญให้เหลือ 4 หลักและอาจต้องนอร์มัลไลซ์อีกรอบหากจำเป็น

$$1.002 \times 10^2$$

จะเห็นได้ว่า ผลลัพธ์ที่ได้ จะเกิดความคลาดเคลื่อนจากค่าที่ควรจะเป็น เนื่องจากมีการจำกัดจำนวนดิจิทของผลลัพธ์ให้เหลือ 4 หลักตามโจทย์

**ตัวอย่างที่ 2.5.4** จงบวกเลขทศนิยมฐานสองชนิดจุดลอยตัวที่นอร์มัลไลซ์แล้ว ดังต่อไปนี้  
วิธีทำ

สมมติว่ามีเลขทศนิยมฐานสองขนาด 4 บิต (4-bit)

$$1.000_2 \times 2^{-1} + -1.110_2 \times 2^{-2} \text{ หรือ } (0.5_{10} + -0.4375_{10})$$

1. เลื่อนตำแหน่งจุดทศนิยมของเลขนัยสำคัญและปรับเลขยกกำลังที่มีค่าสัมบูรณ์น้อยกว่า ให้ตรงกับเลขยกกำลังของเลขที่มีค่าสัมบูรณ์มากกว่า

$$1.000_2 \times 2^{-1} + -0.111_2 \times 2^{-1}$$

2. บวกค่านัยสำคัญที่เลื่อนตำแหน่งแล้วเข้าด้วยกัน จะได้ผลลัพธ์ดังนี้

$$(1.000_2 - 0.111_2) \times 2^{-1} = 0.001_2 \times 2^{-1}$$

3. นอร์มัลไลซ์ค่านัยสำคัญของผลลัพธ์และตรวจเช็คการเกิดโอเวอร์โฟล์และอันเดอร์โฟล์

$$1.000_2 \times 2^{-4}, \text{ ไม่เกิดโอเวอร์โฟล์และอันเดอร์โฟล์}$$

4. ปัดค่านัยสำคัญให้เหลือ 4 บิตและอาจต้องนอร์มัลไลซ์อีกรอบหากจำเป็น

$$1.000_2 \times 2^{-4} (\text{ไม่เปลี่ยนแปลง}) = 0.0625_{10}$$

## 2.5.2 การคูณเลขทศนิยมฐานสองชนิดจุดลอยตัว

เพื่อให้ผู้อ่านเข้าใจกลไกการคูณเลขทศนิยมฐานสองชนิด Floating-Point เราจะใช้ตัวอย่างจากการคูณเลขทศนิยมฐานสิบก่อนในทำนองเดียวกับการบวกเลขทศนิยมฐานสองชนิดจุดลอยตัว แล้วจึงนำขั้นตอนที่เหมือนกันไปประยุกต์ใช้กับการคูณเลขทศนิยมฐานสองในตัวอย่างต่อไป

**ตัวอย่างที่ 2.5.5** จงคูณเลขทศนิยมฐานสิบชนิดจุดลอยตัวที่นอร์มัลไลซ์ ดังต่อไปนี้

สมมติว่ามีเลขฐานสิบขนาด 4 หลัก (4-digit)  $(1.110_{10} \times 10^{10}) \times (9.200_{10} \times 10^{-5})$

วิธีทำ

1. บวกค่ายกกำลังเข้าด้วยกัน ทำให้ค่ายกกำลังใหม่  $= 10 + -5 = 5$

2. คูณค่านัยสำคัญเข้าด้วยกัน

$$1.110 \times 9.200 = 10.212 = 10.212 \times 10^5$$

3. นอร์มัลไลซ์ค่าผลลัพธ์ และตรวจเช็คการเกิดโอเวอร์โฟล์และอันเดอร์โฟล์

$$1.0212 \times 10^6$$

4. ปั๊ดค่าให้เหลือ 4 หลักและอาจต้องนำรմัลไลซ์เมื่อจำเป็น

$$1.021 \times 10^6$$

5. ปรับเครื่องหมายของผลลัพธ์ให้ถูกต้องจากตัวตั้งและตัวคูณ

$$+1.021 \times 10^6$$

จะเห็นได้ว่า ผลลัพธ์ที่ได้จะเกิดความคลาดเคลื่อนจากค่าที่ควรจะเป็น เนื่องจากมีการจำกัดจำนวนตัวจิทของผลลัพธ์ให้เหลือ 4 หลักตามโจทย์

ตัวอย่างที่ 2.5.6 จงคูณเลขทศนิยมฐานสองชนิดจุดลอยตัวขนาด 4 บิต (4-bit) ที่นำร์มัลไลซ์ ดังต่อไปนี้ สมมติว่ามีเลขฐานสองขนาด 4 บิต (4-bit)

$$1.000_2 \times 2^{-1} \times -1.110_2 \times 2^{-2} \text{ หรือเท่ากับ } (0.5_{10} \times -0.4375_{10})$$

วิธีทำ

1. บวกค่ายกกำลังหัวลงค่าเข้าด้วยกัน ทำให้ค่ายกกำลังใหม่:  $-1 + -2 = -3$

2. คูณค่านัยสำคัญเข้าด้วยกัน

$$1.000_2 \times 1.110_2 = 1.110_2 = 1.110_2 \times 2^{-3}$$

3. นำร์มัลไลซ์ค่านัยสำคัญของผลลัพธ์ และตรวจสอบการเกิดโอเวอร์โฟล์และอันเดอร์โฟล์

$$1.110_2 \times 2^{-3} \text{ (ไม่เปลี่ยนแปลง) และไม่เกิดโอเวอร์โฟล์และอันเดอร์โฟล์}$$

4. ปั๊ดค่านัยสำคัญให้เหลือ 4 หลักและอาจต้องนำร์มัลไลซ์เมื่อจำเป็น

$$1.110_2 \times 2^{-3} \text{ (ไม่เปลี่ยนแปลง)}$$

5. ปรับเครื่องหมายของผลลัพธ์ให้ถูกต้องจากตัวตั้งและตัวคูณ

$$-1.110_2 \times 2^{-3} = -0.21875_{10}$$

## 2.6 เลขทศนิยมฐานสองชนิดจุดลอยตัวมาตรฐาน IEEE754

มาตรฐานของเลขทศนิยมฐานสองชนิดจุดลอยตัวได้ถูกกำหนดโดย IEEE (Institute of Electrical and Electronics Engineers) เรียกว่า มาตรฐาน IEEE754 ในปี ค.ศ.1985 เพื่อให้โปรแกรมคอมพิวเตอร์สามารถคำนวณค่าเลขทศนิยมฐานสองบนเครื่องที่ใช้ชิปปิชได้แล้วให้ผลลัพธ์ตรงกัน ปัจจุบันนี้มาตรฐานได้รับการยอมรับอย่างแพร่หลายและปรับปรุงอย่างต่อเนื่อง สามารถรองรับเลขทศนิยมจุดลอยตัว 2 รูปแบบหลัก คือ

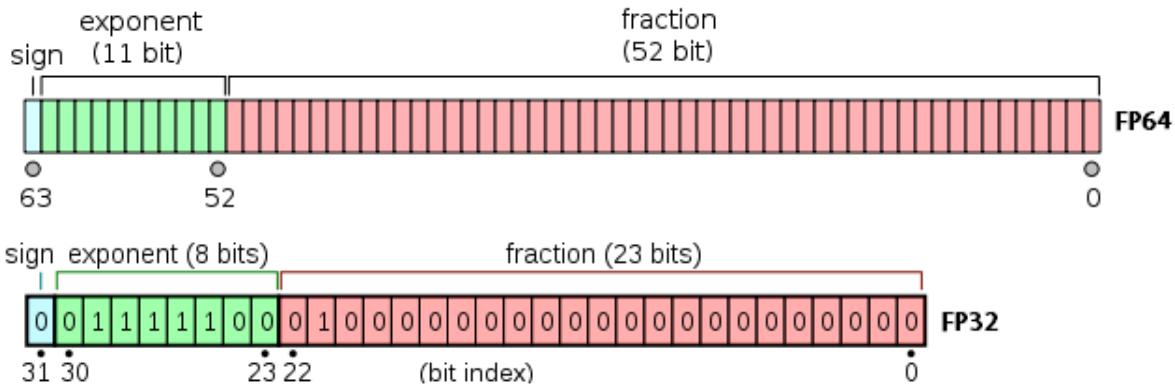
- ชนิด Single precision (32-bit) ตรงกับตัวแปรชนิด float ในภาษา C/C++ และ Java
- ชนิด Double precision (64-bit) ตรงกับตัวแปรชนิด double ในภาษา C/C++ และ Java เวอร์ชันล่าสุดของ IEEE754 คือ ปี ค.ศ. 2019 รายละเอียดเพิ่มเติม

ตัวอย่างการประกาศและตั้งค่าตัวแปรที่ใช้มาตรฐานนี้

```
float a = -5; /* a = 0xC0A00000 */
double b = -0.75; /* b = 0xBFE8000000000000 */
```

เมื่อผู้อ่านทำความเข้าใจทฤษฎีและตัวอย่างการคำนวณเบื้องต้นแล้ว ผู้อ่านสามารถทำการทดลองเพิ่มเติมในการทดลองที่ 1 ภาคผนวก A ข้อมูลและคณิตศาสตร์ในคอมพิวเตอร์ หัวข้อที่ A.2

### 2.6.1 รูปแบบมาตรฐาน IEEE754



รูปที่ 2.7: โครงสร้างของเลขทศนิยมฐานสองชนิดจุดลอยตัวตามมาตรฐาน IEEE754 Double Precision และ Single Precision ที่มา: [pybugs.com](http://pybugs.com)

นิยามที่ 2.6.1 เลขทศนิยมฐานสองชนิดจุดลอยตัวที่นิยมมัลไวซ์แล้ว สามารถเขียนอยู่ในรูปของเลขฐานสองต่อไปนี้

$$F_{2,IEEE} = [s][E_{2,IEEE}][Y_2] \quad (2.63)$$

โดยค่าทศนิยม (Fraction):  $Y_2$  ตามสมการที่ (2.64) มีความยาว  $m=23$  และ 52 บิต ตามชนิด Single Precision และ Double Precision ตามลำดับ สามารถเขียนเป็นสัญลักษณ์คล้ายกับเลขทศนิยมฐานสองชนิดจุดตรึง ดังนี้

$$Y_2 = y_{-1}y_{-2}y_{-3}\dots y_{-m} \quad (2.64)$$

ดังนั้น ค่านัยสำคัญ (Significand:  $S_2$ ) มีความยาว  $m+1=24$  หรือ 53 บิต โดยมีรูปแบบตามสมการที่ (2.61) ในหัวข้อที่ 2.5

ค่ายกกำลัง เป็นเลขจำนวนเต็มชนิดไม่มีเครื่องหมาย ความยาว  $n=8$  และ 11 บิต ขึ้นกับชนิด Single Precision และ Double Precision ตามลำดับ โดยมีลักษณะคล้ายกับเลขทศนิยมฐานสองชนิดจุดลอยตัวในสมการที่ (2.60) แต่ต่างกันที่เลขยกกำลังของ IEEE754 มีค่าเป็นเลขจำนวนเต็มไม่มีเครื่องหมาย

$$E_{2,IEEE} = [e_{n-1} e_{n-2} \dots e_0] \quad (2.65)$$

โดยรายละเอียดเป็นดังนี้

- $s$ : บิตเครื่องหมาย (Sign bit)
  - 0 หมายถึง เลขบวกหรือเท่ากับศูนย์ (non-negative)
  - 1 หมายถึง เลขลบ (negative)
- เลขนัยสำคัญ (Significand =  $1 + \text{Fraction}$ ): หมายความว่า ค่าสัมบูรณ์ (Absolute Value) ของเลขนัยสำคัญจะอยู่ในช่วง  $1.0 \leq |\text{Significand}| < 2.0$
- จากสมการที่ (2.65) เลขยกกำลัง (Exponent:  $E_{2,IEEE}$ ) = เลขยกกำลังจริง (True Exponent:  $E_2$ ) + ไบแอส ( $E_{bias}$ )
 

การบวกค่า  $E_{bias}$  มีวัตถุประสงค์เพื่อปรับให้ค่ายกกำลังเป็นเลขที่ไม่มีเครื่องหมาย (unsigned) โดย

  - ชนิด Single Precision ค่า  $E_{bias} = 01111111_2 = 127_{10}$
  - ชนิด Double Precision ค่า  $E_{bias} = 011111111111_2 = 1023_{10}$

ดังนั้น เลขยกกำลังจริงจึงสามารถคำนวณได้โดย

$$E_2 = E_{2,IEEE} - E_{bias} \quad (2.66)$$

เลขทศนิยมฐานสองชนิดจุดลอยตัวตามมาตรฐาน IEEE754 ในสมการที่ (2.63) สามารถแปลงเป็นค่าเลขทศนิยมฐานสิบ ได้ดังนี้

$$F_{10,IEEE} = (-1)^s \times \left[ 1 + \frac{y_{-1}}{2} + \frac{y_{-2}}{4} + \frac{y_{-3}}{8} + \dots + \frac{y_{-m}}{2^m} \right] \times 2^{(E_{2,IEEE} - E_{bias})} \quad (2.67)$$

ตัวอย่างที่ 2.6.1 เลขคณิตมฐานสองชนิดจุดลอยตัวมาตรฐาน IEEE754 ชนิด Single Precision ต่อไปนี้ มีค่าเท่าไรในฐานสิบตามสมการที่ (2.67)

$$42280000_16 = [0100\ 0010\ 0010\ 1000\ 0000\ 0000\ 0000]_2$$

วิธีทำ

1. แปลงจากเลขฐานสิบให้เป็นฐานสองและองค์ประกอบต่างๆ ได้ดังนี้  
 $s=[0][E_{2,IEEE}=100\ 0010\ 0][Y_2=010\ 1000\ 0000\ 0000\ 0000]_2$

2. จะพบว่าบิตเครื่องหมาย  $s = 0$

$$\text{ค่ายกกำลังจริง } E_{true} = 1000\ 0100_2 - E_{bias} = 132 - 127 = 5$$

$$\text{ค่าทศนิยม } Y_2 = 010\ 1000\ 0000\ 0000\ 0000_2$$

$$F_{10,IEEE} = (-1)^0 \times (1 + .0101_2) \times 2^5 \quad (2.68)$$

$$= (-1)^0 \times (1.0101_2) \times 2^5 \quad (2.69)$$

$$= (-1)^0 \times (101010.0_2) \quad (2.70)$$

$$= (+1) \times (32 + 8 + 2) \quad (2.71)$$

$$= 42.0_{10} \quad (2.72)$$

ผู้อ่านสามารถถูกใจการจัดเรียงตัวของเลขฐานสิบ 42.0 นี้ในหน่วยความจำในตัวแปรชื่อ  $a$  ในรูปที่ 2.2

ตัวอย่างที่ 2.6.2 เลขคณิตมฐานสองชนิดจุดลอยตัวมาตรฐาน IEEE754 ชนิด Single Precision ต่อไปนี้ มีค่าเท่าไรในฐานสิบตามสมการที่ (2.67)

$$C0A00000_16 = [1100\ 0000\ 1010\ 0000\ 0000\ 0000\ 0000]_2$$

วิธีทำ

1. แปลงจากเลขฐานสิบให้เป็นฐานสองและองค์ประกอบต่างๆ ได้ดังนี้  
 $s=[1][E_{2,IEEE}=100\ 0000\ 1][Y_2=010\ 0000\ 0000\ 0000\ 0000]_2$

2. จะพบว่าบิตเครื่องหมาย  $s = 1$

$$\text{ค่ายกกำลังจริง } E_{true} = 1000\ 0001_2 - E_{bias} = 129 - 127 = 2$$

$$\text{ค่าทศนิยม } Y_2 = 010\ 0000\ 0000\ 0000\ 0000_2$$

$$F_{10,IEEE} = (-1)^1 \times (1 + .01_2) \times 2^2 \quad (2.73)$$

$$= (-1) \times (1.01_2) \times 2^2 \quad (2.74)$$

$$= (-1) \times (101.0_2) \quad (2.75)$$

$$= (-1) \times 5.0 \quad (2.76)$$

$$= -5.0_{10} \quad (2.77)$$

ในตัวอย่างนี้กับเลขจำนวนเต็มฐานสองความยาว 32 บิตชนิดมีเครื่องหมายในตัวอย่างที่ 2.2.11 ซึ่งตรงกับ  $-5_{10}$  ทั้งคู่

ตัวอย่างที่ 2.6.3 จงหาค่าทศนิยมฐานสิบของเลข FP-32 ที่เติมในรูปที่ 2.7 คือ เลขทศนิยมมาตราฐาน IEEE754 ชนิด Single Precision ตามสมการที่ (2.67) ดังต่อไปนี้  $[0][011\ 1110\ 0][010\ 0000\ 0000\ 0000\ 0000]$

วิธีทำ

$$F_{10,IEEE} = (-1)^0 \times 1.01 \times 2^{(124-127)} \quad (2.78)$$

$$= 1 \times 1.01 \times 2^{-3} \quad (2.79)$$

$$= 0.00101_2 \quad (2.80)$$

$$= 2^{-3} + 2^{-5} \quad (2.81)$$

$$= \frac{1}{2^3} + \frac{1}{2^5} \quad (2.82)$$

$$= 0.125_{10} + 0.03125_{10} = 0.15625_{10} \quad (2.83)$$

ตัวอย่างที่ 2.6.4 จงแปลงเลข  $-0.75_{10}$  เป็นเลขทศนิยมฐานสองชนิดจุดลอยตัวตามมาตราฐาน IEEE754 ทั้งสองชนิด

วิธีทำ

1. แปลงเลขทศนิยมฐานสิบให้อยู่ในรูปฐานสองแบบนอร์มัลไลร์

$$-0.75_{10} = (-1)^1 \times (0.5_{10} + 0.25_{10})$$

$$= (-1)^1 \times (\frac{1}{2} + \frac{1}{4})$$

$$= (-1)^1 \times (0.1_2 + 0.01_2)$$

$$= (-1)^1 \times 0.11_2 \times 2^0$$

2. ทำการนอร์มัลไลร์ ตามสมการที่ (2.59)

$$-0.75_{10} = (-1)^1 \times 1.1_2 \times 2^{-1}$$

ดังนั้น บิตเครื่องหมาย  $s = 1$

ค่าทศนิยม  $Y_2 = 100\ 0000\ 0000\ 0000\ 0000_2$

ค่ายกกำลัง  $E_{2,IEEE} = -1 + E_{bias}$

โดยชนิด Single ค่ายกกำลัง (8 บิต):  $E_{2,IEEE} = -1 + 127 = 126$  หรือ  $E_{2,IEEE} = 0111\ 1110_2$

โดยชนิด Double ค่ายกกำลัง (11 บิต):  $E_{2,IEEE} = -1 + 1023 = 1022$  หรือ  $E_{2,IEEE} = 011\ 1111\ 1110_2$

3. ดังนั้น  $-0.75_{10}$  เขียนในรูปของเลขทศนิยมชนิดลอยตัวตามมาตราฐาน IEEE754 ชนิด

Single:  $[1][011\ 1111\ 0][100\ 0000\ 0000\ 0000\ 0000]_2 = BF40\ 0000_{16}$

Double:  $[1][011\ 1111\ 1110][1000\ 0000\ 0000\ ...0000]_2 = BFE8\ 0000\ 0000\ 0000_{16}$

## 2.6.2 ค่าสูงสุด ต่ำสุดและค่าอื่นๆ ของมาตรฐาน IEEE754

การคำนวณค่าของตัวแปรชนิด float และ double ของฟ์เฟร์ในเครื่องคอมพิวเตอร์สามารถรองรับการคำนวณโดยใช้เลขทศนิยมฐานสองชนิดจุดลอยตัวตามมาตรฐาน IEEE754 แต่เนื่องจากเลขจำนวนจริงในทางคณิตศาสตร์มีจำนวนอนันต์ (Infinite) เลขทศนิยมฐานสองชนิดจุดลอยตัวตามมาตรฐาน IEEE754 จึงไม่สามารถรองรับได้ทั้งหมด และมีข้อจำกัดทางคณิตศาสตร์ ตามที่สรุปในตารางที่ 2.12

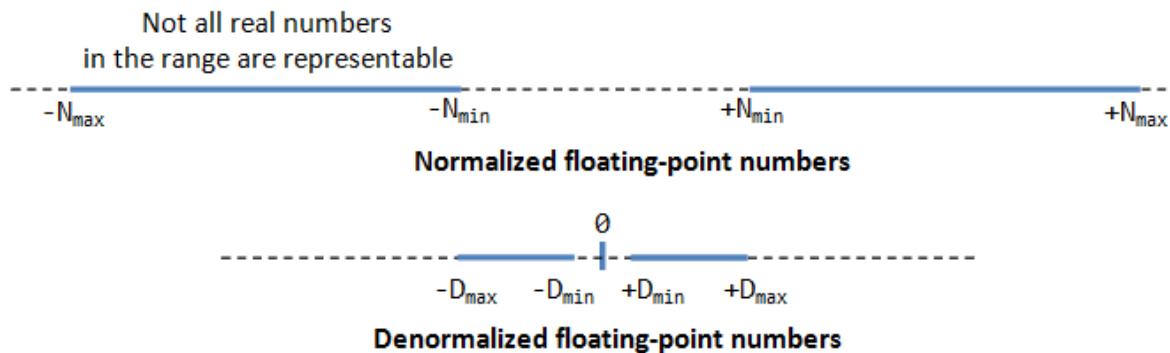
ตารางที่ 2.12: ตารางสรุปความแตกต่างระหว่างเลขทศนิยมฐานสองชนิดจุดลอยตัวตามมาตรฐาน IEEE754 ชนิด Single Precision โดย  $E_{2,IEEE}$  คือ ค่ายกกำลัง และ  $Y_2$  คือ ทศนิยมซึ่งเป็นส่วนประกอบของค่านัยสำคัญ ( $\times$  หมายถึง บิตข้อมูลมีค่าเท่ากับ '0' หรือ '1')

ลำดับที่	เครื่องหมาย $s$ (1 บิต)	ค่ายกกำลัง $E_{2,IEEE}$ (8 บิต) สมการ (2.66)	ทศนิยม $Y_2$ (23 บิต) สมการ (2.61)	ความหมาย
1	±	0000 0001 <sub>2</sub> - 1111 1110 <sub>2</sub>	xx.. <sub>2</sub>	เลขฐานสิบห้าไป (นอร์มัลไลซ์) สมการที่ (2.67)
2	±	0000 0000 <sub>2</sub>	xx.. <sub>2</sub>	เลขฐานสิบห้าอยมาก แต่ไม่เท่ากับศูนย์ (ดินอร์มัลไลซ์)
3	0	0000 0000 <sub>2</sub>	0...0 <sub>2</sub>	0.0 <sub>10</sub> (ศูนย์จุดศูนย์)
4	±	1111 1111 <sub>2</sub>	0...0 <sub>2</sub>	±∞ (±infinity)
5	0	1111 1111 <sub>2</sub>	xx.. <sub>2</sub>	Nan (Not a Number)

ผู้อ่านสามารถทำความเข้าใจตารางที่ 2.12 ตามลำดับที่ดังต่อไปนี้

1. เลขฐานสิบห้าไป (นอร์มัลไลซ์) ทั้งบวกและลบเป็นย่านของเลขทศนิยมที่มาตรฐานสามารถแสดงค่าได้โดยมีค่าสัมบูรณ์ (Absolute Value) อยู่ระหว่างค่าต่ำสุด ( $N_{min}$ ) ที่เข้าใกล้ศูนย์ จนถึง ค่าที่สูงมาก ( $N_{max}$ ) ในรูปที่ 2.8 ตามค่าฐานสิบในสมการที่ (2.67)
2. เลขทศนิยมที่มีค่าสัมบูรณ์น้อยมากอยู่ในช่วง ( $D_{min}$  ถึง  $D_{max}$ ) ในรูปที่ 2.8 เรียกว่า ค่าดินอร์มัลไลซ์ (Denormalize)
3. เลขทศนิยม 0.0<sub>10</sub> หรือ ศูนย์จุดศูนย์ เรียกว่า ศูนย์จริง (True zero)
4. ค่าบวกและลบอนันต์หรืออินฟินิตี้ สามารถใช้แทนผลลัพธ์ที่มีค่าสูงเกินขอบเขตของมาตรฐาน เนื่องจากผลลัพธ์ที่คำนวณได้ทางคณิตศาสตร์ เช่น การคูณและการบวกเลขขนาดใหญ่มากสองจำนวน หรือการหารด้วยตัวหารขนาดเล็กมากจนผลหารที่ได้เหลือขอบเขตค่าสูงสุดที่มาตรฐานกำหนด
5. NaN เป็นสัญลักษณ์ที่ไม่สามารถแทนได้ด้วยตัวเลข ย่อมาจากคำว่า Not a Number แสดงถึงความผิดพลาดที่อาจเกิดจากการเขียนโปรแกรมเพื่อใช้ตัวแปรหารค่าตัวตั้ง แต่ตัวแปรที่ใช้หารมีค่าที่เปลี่ยนแปลงไปจนกลายเป็น 0.0<sub>10</sub> เมื่อนำไปหารจึงกลายเป็นสัญลักษณ์นี้ เพื่อบ่งบอกข้อผิดพลาดที่เกิดขึ้นแล้ว

รูปที่ 2.8 แสดงค่าบนเส้นจำนวนของลำดับต่างๆ ในตารางที่ 2.12 ยกเว้น NaN เส้นด้านบนของรูปแสดงเลขต่างๆ ที่เขียนในรูปนอร์มัลไลซ์ได้ ซึ่งตรงกับลำดับที่ 1 เลขต่างๆ ที่เขียนในรูปนอร์มัลไลซ์ได้จากซ้ายสุดถึงขวาสุด ของเส้นจำนวน สำหรับ Single Precision มีค่าตั้งแต่  $-N_{max} = -3.40 \times 10^{38}$  ถึง  $-N_{min} = -1.18 \times 10^{-38}$  และ  $N_{min} = +1.18 \times 10^{-38}$  ถึง  $N_{max} = +3.40 \times 10^{38}$  ซึ่งสอดคล้องกับค่าในตารางที่ 2.1 ส่วนเส้นด้านล่างของรูปที่ 2.8 แสดงเลขขนาดเล็กมากๆ ที่ไม่สามารถเขียนในรูปนอร์มัลไลซ์ได้ เรียกว่า เลขดินอร์มัลไลซ์ ตรงกับลำดับที่ 2 ในตารางที่ 2.12



รูปที่ 2.8: เลขทศนิยมโดยตัวชนิดนอร์มัลไลซ์ตั้งแต่  $\pm N_{min}$  ถึง  $\pm N_{max}$  และชนิดดินอร์มัลไลซ์ ตั้งแต่  $\pm D_{min}$  ถึง  $\pm D_{max}$  ที่มา: [ntu.edu.sg](http://ntu.edu.sg)

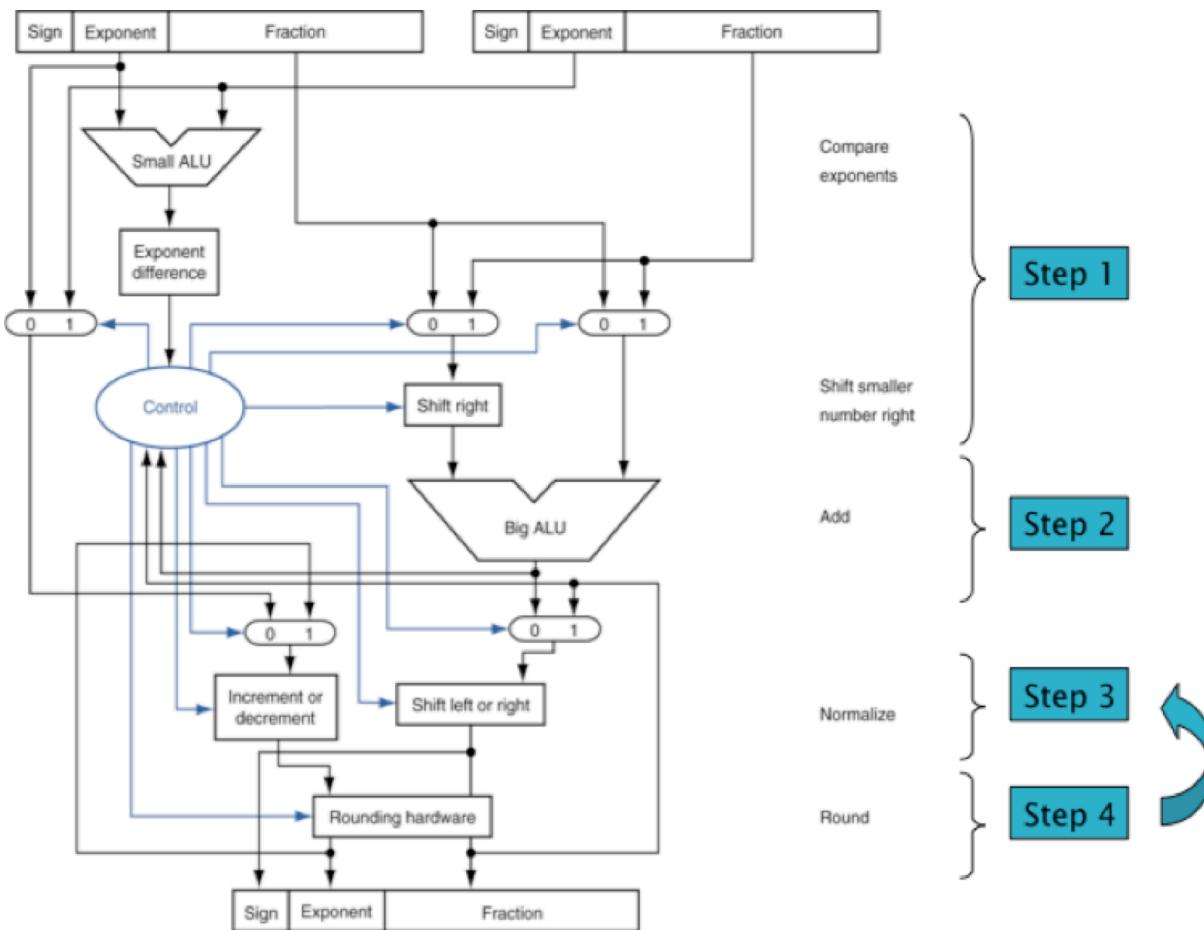
### 2.6.3 การบวกเลขทศนิยมฐานสองชนิดจุดโดยตัวตาม IEEE754

การบวกเลขทศนิยมฐานสองชนิดจุดโดยตัวตามมาตรฐาน IEEE754 จำเป็นต้องอาศัยวงจรฮาร์ดแวร์หรือ (Floating Point ALU) ช่วยคำนวณแบบไปป์ไลน์ (Pipeline) ซึ่งมีหลักการคล้ายกับโรงงานประกอบรถยนต์ทำให้ซีพียู 1 แกนประมวลผลสามารถคำนวณการบวกเลขทศนิยมได้ในหลัก กิกะฟล็อปส์ (Giga Floating Point Operations Per Second: Giga FLOPS) หรือ พันล้านหรือ  $10^9$  ครั้งต่อวินาที ซึ่งการวัดสมรรถนะ (Performance) ของซีพียู รวมไปถึงซูเปอร์คอมพิวเตอร์ (Supercomputer) ในปัจจุบันใช้หน่วยวัดเพطاฟล็อปส์ (Peta FLOPS) หรือ  $10^{15}$  ครั้งต่อวินาที ผู้อ่านสามารถศึกษาเรื่อง ฟล็อปส์ ได้ที่ [wikipedia](https://en.wikipedia.org/wiki/Floating_point)

ดังนั้น การทำงานของวงจรบวก/ลบเลขทศนิยมชนิดจุดโดยตัว จึงอาศัยหลักการเดียวกันกับ อัลกอริธึมการบวก ซึ่งมีความซับซ้อนใกล้เคียงกับการคูณเลขทศนิยม และทั้งคู่มีความซับซ้อนมากกว่าและใช้เวลานานกว่าการบวกและการคูณเลขจำนวนเต็มที่จำนวนบิตเท่ากัน

วงจรบวกเลขทศนิยมฐานสองชนิดจุดโดยตัวในรูปที่ 2.9 มีความซับซ้อนใกล้เคียงกับตัวอย่างที่แสดงในหัวข้อที่ 2.5 ที่ผ่านมา ดังนี้

- ขั้นตอนที่ (Step) 1 คือ ใช้วงจร Small ALU ในรูปที่ 2.9 หากค่าผลต่างของเลขยกกำลังจากเลขทั้งสองตัว ทำการเลื่อน (Shift) จุดทศนิยมของค่านัยสำคัญของเลขที่มีค่ายกกำลังน้อยกว่าไปทางขวา เป็นจำนวนบิตเท่ากับค่าสัมบูรณ์ (Absolute) ของผลต่าง ส่วนค่านัยสำคัญของเลขที่มีค่ายกกำลังมากกว่าจะไม่เปลี่ยนแปลง
- ขั้นตอนที่ (Step) 2 คือ ทำการบวก/ลบค่านัยสำคัญที่เลื่อนตำแหน่งแล้วเข้าด้วยกัน โดยใช้วงจร ALU ขนาดใหญ่ (Big ALU) ในรูปที่ 2.9
- ขั้นตอนที่ (Step) 3 คือ ทำการนอร์มัลไลซ์ค่านัยสำคัญของผลลัพธ์ และตรวจเชิงการเกิดโอเวอร์โฟลว์ และอันเดอร์โฟลว์ ซึ่งผลลัพธ์ที่ได้จากการบวกอาจมีค่าเพิ่มขึ้น หรือ ลดลงขึ้นอยู่กับเครื่องหมายและขนาดของเลขทั้งสองตัว ทำให้ต้องนอร์มัลไลซ์ค่านัยสำคัญอีกรอบ
  - หากค่านัยสำคัญสัมบูรณ์ของผลลัพธ์เพิ่มมากขึ้นจนเกินค่าสูงสุดในลำดับที่ 1 ของตารางที่ 2.12 เรียกว่าเกิด โอเวอร์โฟลว์ (Overflow) ของเลข IEEE754
  - หากค่านัยสำคัญสัมบูรณ์ของผลลัพธ์มีค่าเข้าใกล้ศูนย์จนน้อยกว่าค่าต่ำสุดในลำดับที่ 1 ของตารางที่ 2.12 และไม่สามารถเขียนในรูปนอร์มัลไลซ์ได้ ผลลัพธ์นั้นจะเขียนในรูปแบบดีนอร์มัลไลซ์ (Denormalize) แทน เรียกว่าเกิดอันเดอร์โฟลว์ (Underflow) ของเลข IEEE754



รูปที่ 2.9: วงจรbaughเลขชินดิจิตอลอยตัวตามมาตรฐาน IEEE754 โดยรับค่าอินพุตจำนวน 2 ตัวด้านบน และ เอาต์พุตผลลัพธ์ด้านล่าง ที่มา: [Patterson and Hennessy \(2016\)](#)

- ขั้นตอนที่ (Step) 4 คือ ปัดค่านัยสำคัญให้เหลือ 24 บิตและอาจต้องนำมัลไทเพอร์ค่านัยสำคัญอีกรอบหากจำเป็น

ตัวอย่างที่ 2.6.5 การบวกเลขทศนิยมฐานสองชนิดจุดลอยตัว (Single Precision) มาตรฐาน IEEE754  
 $-5_{10} + -0.75_{10}$  หรือเท่ากับ  $C0A0\ 0000_{16} + BF40\ 0000_{16} = ?$

$$1100\ 0000\ 1010\ 0000\ 0000\ 0000\ 0000_2 +$$

$$1011\ 1111\ 0100\ 0000\ 0000\ 0000\ 0000_2$$

วิธีทำ

ผู้อ่านจะต้องแปลงเลขฐานสองให้เป็นรูปแบบเลขฐานสองที่นอร์มัลไลซ์ ดังต่อไปนี้

$$(-1)^1 \times (1.010\ 0000\ 0000\ 0000\ 0000)_2 \times 2^2 +$$

$$(-1)^1 \times (1.100\ 0000\ 0000\ 0000\ 0000)_2 \times 2^{-1}$$

1. เลื่อนตำแหน่งจุดทศนิยมของเลขนัยสำคัญและปรับเลขยกกำลังที่มีค่าสัมบูรณ์น้อยกว่า ให้ตรงกับเลขยกกำลังของเลขที่มีค่าสัมบูรณ์มากกว่า

$$(-1)^1 \times (1.010\ 0000\ 0000\ 0000\ 0000)_2 \times 2^2 +$$

$$(-1)^1 \times (0.001\ 1000\ 0000\ 0000\ 0000)_2 \times 2^2$$

2. บวกค่านัยสำคัญที่เลื่อนตำแหน่งแล้วเข้าด้วยกัน จะได้ผลลัพธ์ดังนี้

$$(-1)^1 \times (1.011\ 1000\ 0000\ 0000\ 0000_2 \times 2^2)$$

3. นอร์มัลไลซ์ค่านัยสำคัญของผลลัพธ์ และตรวจเช็คการเกิดโอเวอร์โฟล์ และอันเดอร์โฟล์

$$(-1)^1 \times (1.011\ 1000\ 0000\ 0000\ 0000_2 \times 2^2) \Rightarrow \text{ไม่เกิดโอเวอร์โฟล์และอันเดอร์โฟล์}$$

4. ปัดค่านัยสำคัญให้เหลือ 24 บิตและอาจต้องนอร์มัลไลซ์เมื่อจำเป็น

$$(-1)^1 \times (1.011\ 1000\ 0000\ 0000\ 0000_2 \times 2^2)$$

$$s = 1$$

$$\text{ค่ายกกำลัง } E_{2,IEEE} = 2+127 = 129 = 1000\ 0001_2$$

$$\text{ค่าทศนิยม } Y_2 = 011\ 1000\ 0000\ 0000\ 0000_2$$

$$F_{2,IEEE} = [1][100\ 0000\ 1][011\ 1000\ 0000\ 0000\ 0000]_2 = C0B8\ 0000_{16}$$

ตัวอย่างที่ 2.6.6 การบวกเลขทศนิยมฐานสองชนิดจุดลอยตัว (Single Precision) มาตรฐาน IEEE754  
 $+1_{10} + -0.75_{10}$  หรือเท่ากับ  $3F80\ 0000_{16} + BF40\ 0000_{16} = ?$

$$0011\ 1111\ 1000\ 0000\ 0000\ 0000\ 0000_2 +$$

$$1011\ 1111\ 0100\ 0000\ 0000\ 0000\ 0000_2$$

วิธีทำ

ผู้อ่านจะต้องแปลงเลขฐานสองให้เป็นรูปแบบเลขฐานสองที่น้อยกว่า ดังต่อไปนี้

$$= (-1)^0 \times (1.000\ 0000\ 0000\ 0000\ 0000)_2 \times 2^0 +$$

$$(-1)^1 \times (1.100\ 0000\ 0000\ 0000\ 0000)_2 \times 2^{-1}$$

- เลื่อนตำแหน่งจุดทศนิยมของเลขนัยสำคัญและปรับเลขยกกำลังที่มีค่าสัมบูรณ์น้อยกว่า ให้ตรงกับเลขยกกำลังของเลขที่มีค่าสัมบูรณ์มากกว่า

$$(-1)^0 \times (1.000\ 0000\ 0000\ 0000\ 0000)_2 \times 2^0 +$$

$$(-1)^1 \times (0.110\ 0000\ 0000\ 0000\ 0000)_2 \times 2^0$$

- นำเลขนัยสำคัญที่เป็นตัวตั้ง (ที่มีเลขกำลังมากกว่า) หักลบด้วย เลขนัยสำคัญที่เป็นตัวลบ (ที่มีเลขกำลังน้อยกว่า) สังเกตได้จาก Sign bit ด้านหน้า จะได้ผลลัพธ์ดังนี้

$$(-1)^0 \times (0.010\ 0000\ 0000\ 0000\ 0000_2 \times 2^0)$$

- นำรัมลไลซ์นัยสำคัญของค่าผลลัพธ์ และตรวจเชิงการเกิดโอเวอร์ไฟล์ และอันเดอร์ไฟล์

$$(-1)^0 \times (1.000\ 0000\ 0000\ 0000\ 0000_2 \times 2^{-2}) \Rightarrow \text{ไม่เกิด}$$

- ปัดค่านัยสำคัญให้เหลือ 24 บิตและอาจต้องนำรัมลไลซ์อีกรอบหากจำเป็น

$$(-1)^0 \times (1.000\ 0000\ 0000\ 0000\ 0000_2 \times 2^{-2})$$

$$s = 0$$

$$\text{ค่ายกกำลัง } E_{2,IEEE} = -2+127 = 125 = 0111\ 1101_2$$

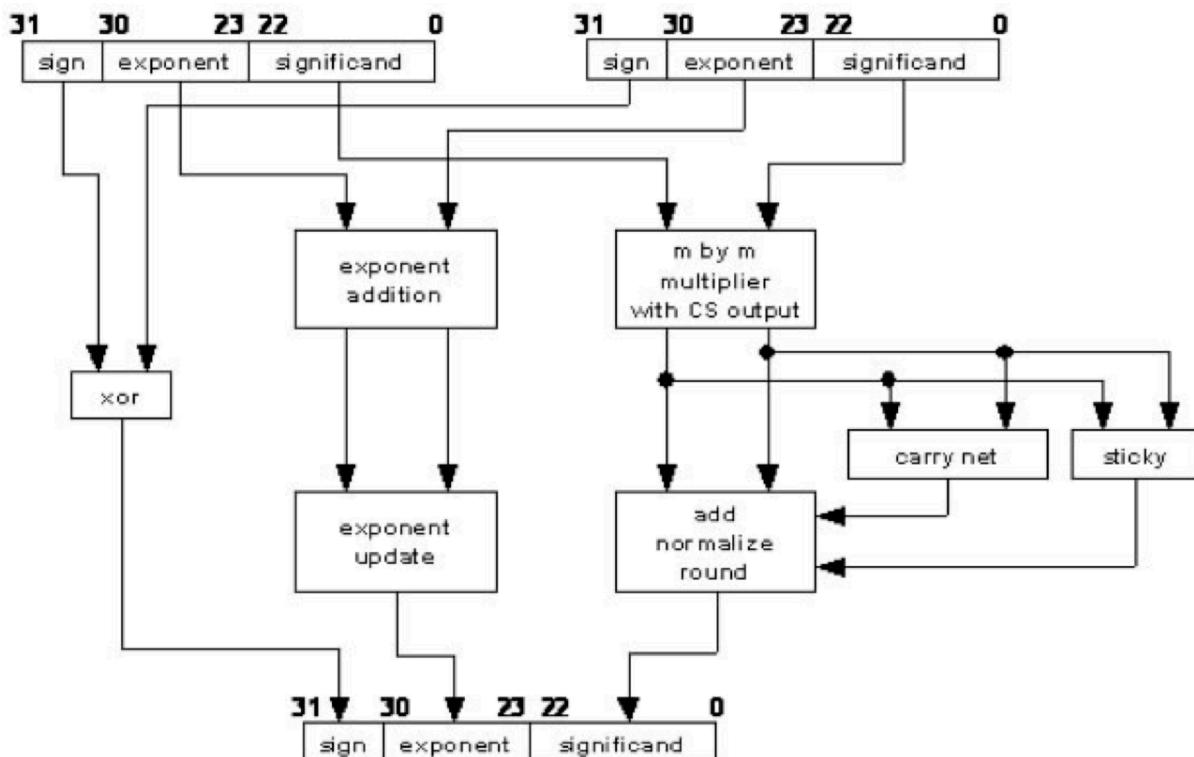
$$\text{ค่าทศนิยม } Y_2 = 000\ 0000\ 0000\ 0000\ 0000_2$$

$$F_{2,IEEE} = [0][011\ 1110\ 1][000\ 0000\ 0000\ 0000\ 0000]_2 \Rightarrow 3E80\ 0000_{16}$$

โดยสรุป ตัวอย่างที่ 2.6.5 และ 2.6.6 แสดงให้เห็นว่า การบวกเลขทศนิยมฐานสองชนิดจุดลอยตัวมาตรฐาน IEEE754 ภายในเครื่องคอมพิวเตอร์โดยใช้รหัสมาตรฐาน IEEE754 สามารถทำงานตามขั้นตอนต่างๆ ตามบล็อกได้อย่างแม่นยำ

### 2.6.4 การคูณเลขทศนิยมฐานสองชนิดจุดลอยตัวตาม IEEE754

การคูณเลขทศนิยมฐานสองชนิดจุดลอยตัวตามมาตรฐาน IEEE754 ด้วยความเร็วสูงจำเป็นต้องอาศัยวงจรฮาร์ดแวร์พิเศษ แต่การทำงานของวงจรนั้นในหลักการคล้ายกับอัลกอริธึมการคูณ ในหัวข้อที่ 2.3.1 ดังนั้น ผู้อ่านจำเป็นต้องศึกษาอัลกอริธึมให้เข้าใจอย่างถ่องแท้ก่อน



รูปที่ 2.10: วงจรคูณเลขทศนิยมฐานสองชนิดจุดลอยตัวตามมาตรฐาน IEEE754 โดยรับค่าอินพุตจำนวน 2 ตัว ด้านบน และเอาต์พุตผลลัพธ์ด้านล่าง ที่มา: [Patterson and Hennessy \(2016\)](#)

วงจรคูณเลขทศนิยมฐานสองชนิดจุดลอยตัว ในรูปที่ 2.10 มีความซับซ้อนมากกว่าวงจรคูณเลขจำนวนเต็มในรูปที่ 2.5 และ 2.6 ดังนี้

- ขั้นตอนที่ (Step) 1 คือ ทำการคูณเฉพาะค่านัยสำคัญทั้งสองเพื่อหาค่านัยสำคัญของผลคูณ
- ขั้นตอนที่ (Step) 2 คือ ทำการบวกค่ายกกำลังทั้งสองเข้าด้วยกันแล้วบวกค่าไบอสหนึ่งครั้ง
- ขั้นตอนที่ (Step) 3 คือ ทำการนормัลไลซ์ค่านัยสำคัญของผลลัพธ์ และตรวจสอบ戈เวอร์ฟล์ และอันเดอร์ฟล์ เหมือนกับกรณีการบวกเลข
- ขั้นตอนที่ (Step) 4 คือ ปัดค่านัยสำคัญให้เหลือ 24 บิตและอาจต้องนормัลไลซ์อีกรอบเมื่อจำเป็น

ตัวอย่างที่ 2.6.7 จงคูณเลขทศนิยมฐานสองชนิดจุดลอยตัว IEEE754 แบบ Single Precision ดังต่อไปนี้  
 EX:  $-5_{10} \times -0.75_{10} = COA0\ 0000_{16} \times BF40\ 0000_{16} = ?$

### วิธีทำ

1. แปลงเลขฐานสิบหกให้เป็นเลขฐานสอง

$$\begin{aligned} 1100\ 0000\ 1010\ 0000\ 0000\ 0000\ 0000_2 \times \\ 1011\ 1111\ 0100\ 0000\ 0000\ 0000\ 0000_2 \end{aligned}$$

2. แปลงเลขฐานสองตามกฎของ IEEE754 Single Precision รูปแบบเลขฐานสองที่นอร์มัลไลซ์ ดังต่อไปนี้

$$\begin{aligned} &= (-1)^1 \times (1.010\ 0000\ 0000\ 0000\ 0000)_2 \times 2^2 \times \\ &(-1)^1 \times (1.100\ 0000\ 0000\ 0000\ 0000)_2 \times 2^{-1} \end{aligned}$$

3. บวกเลขยกกำลังเข้าด้วยกัน:  $[2^2 \times 2^{-1}] = 2^1$

$$\begin{aligned} &= (-1)^1 \times (-1)^1 \times [(1.010\ 0000\ 0000\ 0000\ 0000)_2 \times \\ &(1.100\ 0000\ 0000\ 0000\ 0000)_2] \times [2^2 \times 2^{-1}] \end{aligned}$$

4. คูณค่านัยสำคัญเข้าด้วยกัน

$$\begin{aligned} 1.010\ 0000\ 0000\ 0000\ 0000\ 0000_2 \times \\ 1.100\ 0000\ 0000\ 0000\ 0000\ 0000_2 \\ = 1.111\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000_2 \end{aligned}$$

5. นอร์มัลไลซ์ค่านัยสำคัญของผลลัพธ์ และตรวจเช็คการเกิดโอเวอร์ฟลั่วและอันเดอร์ฟลั่ว

$$1.111\ 0000\ 0000\ 0000\ 0000\ 0000_2 \times 2^1 \text{ (ค่าเป็นปกติ)}$$

6. บัดค่านัยสำคัญให้เหลือ 24 บิตและอาจต้องนอร์มัลไลซ์อีกรอบหากจำเป็น

$$1.111\ 0000\ 0000\ 0000\ 0000\ 0000_2 \times 2^1 \text{ (ไม่มีการเปลี่ยนแปลงเกิดขึ้น)}$$

7. ปรับเครื่องหมายของผลลัพธ์ให้ถูกต้องจากตัวตั้งและตัวคูณ

$$(-1)^0 \times (1.111\ 0000\ 0000\ 0000\ 0000\ 0000_2 \times 2^1)$$

บิตเครื่องหมาย  $s = 0$

$$\text{ค่ายกกำลัง } E_{2,IEEE} = 1 + 127 = 128 = 1000\ 0000_2$$

$$\text{ค่าทศนิยม } Y_2 = 111\ 0000\ 0000\ 0000\ 0000\ 0000_2$$

$$F_{2,IEEE} = 0100\ 0000\ 0111\ 0000\ 0000\ 0000\ 0000_2 = 4070\ 0000_{16}$$

ดังนั้น  $-5_{10} \times -0.75_{10} = 3.75_{10}$  แต่ในเครื่องคอมพิวเตอร์จะเห็นเป็นค่า

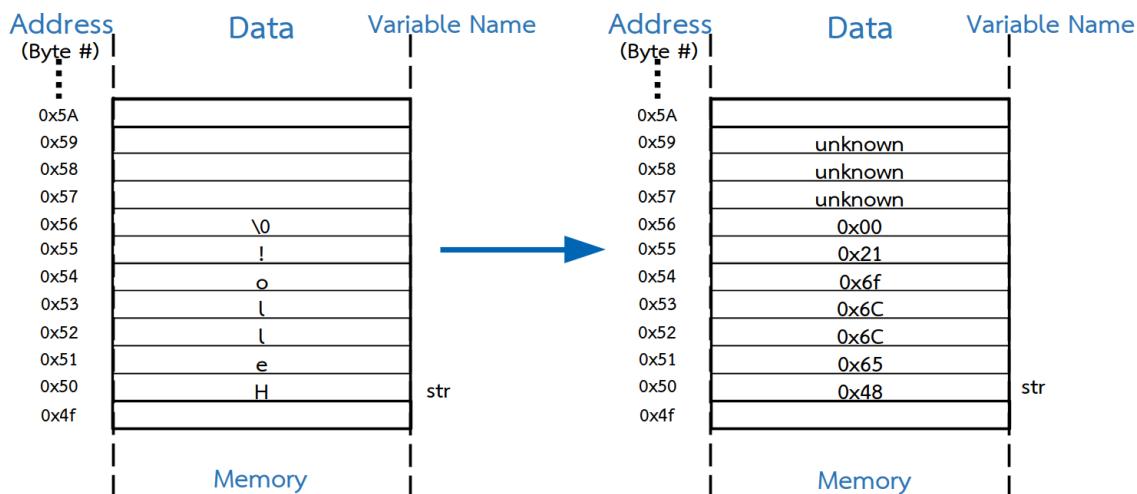
$$COA0\ 0000_{16} \times BF40\ 0000_{16} = 4070\ 0000_{16}$$

โดยสรุป ตัวอย่างที่ 2.6.7 แสดงให้เห็นว่า การคูณเลขทศนิยมฐานสองชนิดจุดลอยตัว IEEE754 ภายในเครื่องคอมพิวเตอร์ตามมาตรฐาน IEEE754 สามารถทำงานตามขั้นตอนต่างๆ ตามบล็อกไดอะแกรมในรูปที่ 2.10

## 2.7 ตัวอักษร (Character) และข้อความ (String)

ตัวอักษรในภาษาต่างๆ ล้วนเป็นข้อมูลที่สำคัญในการใช้งานคอมพิวเตอร์ โดยเฉพาะชื่อ นามสกุล หนังสือ ตำรา ข้อความ คำสอนทนาต่างๆ ในเครือข่ายอินเทอร์เน็ต เพื่อนำมาสืบค้น สร้างความเชื่อมโยงในหลากหลายมิติ ผ่านสื่อสังคมออนไลน์ต่างๆ ได้อย่างแพร่หลาย รูปที่ 2.11 แสดงการเก็บข้อความ (String) ในหน่วยความจำด้วยรหัสแอสกี (ASCII) ของตัวแปรข้อความหรือตัวแปรชนิดสตริงซึ่งว่า str ซึ่งเป็นตัวแปรชนิดอาร์เรย์ของตัวอักษร (Array of Character)

```
char[10] str="Hello!"
```



รูปที่ 2.11: การเก็บข้อความในหน่วยความจำในรูปของรหัสแอสกีด้วยตัวแปรสตริงซึ่งว่า str ซึ่งเป็นตัวแปรชนิด char[10] str="Hello!" ที่แอดเดรสเริ่มต้น 0x50 หรือ  $50_{16}$  ที่มา: [Harris and Harris \(2013\)](#)

การเรียงตัวอักษรในหน่วยความจำจะเริ่มต้นจากหมายเลขไบต์หรือแอดเดรสเริ่มต้น 0x50 หรือ  $50_{16}$  ไปยัง แอดเดรสที่สูงขึ้นตามรูปที่ 2.11 โดยแอดเดรส 0x50 เก็บรหัสแอสกีหมายเลข 0x48 หรือ  $48_{16}$  ซึ่งตรงกับตัวอักษร 'H' ไปจนถึงแอดเดรส 0x56 ซึ่งเก็บตัวอักษรรหัส 0x00 หรือ NULL อ่านว่า นัล ซึ่งเป็นอักษรพิเศษ แสดงว่าจบประโยค ส่วนไบต์ที่ 0x57, 0x58, 0x59 จะมีค่าเป็นเลขไดๆ (unknown) เพราะไม่มีผลต่อประโยคที่เก็บ ตัวแปร str นี้สามารถเก็บตัวอักษรจำนวนสูงสุด 9 ตัวและตัวสุดท้ายจะต้องเป็นตัวอักษร NULL เพ่านั้น ยกตัวอย่างเช่น ประโยค "012345678" ประโยค "abcdefghijkl" เป็นต้น

รหัสแอสกี (ASCII) คือ มาตรฐานของรูปแบบการใช้เลขฐานสองเพื่อแทนตัวอักษรตั้งแต่ต่ำตีติ ซึ่งกำหนดขึ้นมาโดยหน่วยงานชื่อว่า ANSI (American National Standard Institute) รูปที่ 2.12 คือ ตารางรหัส ASCII กำหนดเลขฐานสองขนาด 8 บิต ที่เพื่อแทนตัวอักษรจำนวน  $2^8=256$  ตัว โดยมีหมายเลขเริ่มต้น คือ  $0000_{10} = 0000_2 = 00_{16}$  ถึง  $255_{10} = 1111\ 1111_2 = FF_{16}$  โดยรหัส  $00_{16}$  แทนอักษร NULL ที่ได้กล่าวไปก่อนหน้านี้ ไมโครซอฟต์พัฒนารหัสภาษาไทยของตนเอง เรียกว่า รหัส Windows-874 โดยใช้มาตรฐาน TIS-620 เป็นพื้นฐาน สำหรับตัวอักษรภาษาไทยสำหรับการแลกเปลี่ยนข้อมูลในระบบปฏิบัติการ Windows

## Codepage 874 - Thai

รูปที่ 2.12: ตารางรหัส ASCII และ Unicode สำหรับตัวอักษรจำนวน  $2^8=256$  ตัว ประกอบด้วยรหัสของตัวอักษรภาษาอังกฤษและภาษาไทย ที่มา: [ascii-table.com](http://ascii-table.com)

รหัส **Unicode** ถูกกำหนดให้เป็นมาตรฐานโดย ISO (International Standard Organization) เพื่อใช้ทดแทนรหัส ASCII และรองรับการใช้ภาษาต่างๆ ทั่วโลก รหัส **Unicode** กำหนดวิธีการเข้ารหัสที่หลากหลายตามวัตถุประสงค์การใช้งาน เช่น รหัส UTF-8 รหัส UTF-16 รหัส UCS-2 เป็นต้น

- รหัส UCS-2 จะใช้พื้นที่ 2 ไบต์ หรือ 16 บิต ต่อ 1 ตัวชาร์ ซึ่งทำให้สามารถใช้เลขฐานสองจำนวน  $2^{16}$  หรือ 65,536 แบบมาแทนตัวอักษรได้แทบทุกตัวอักษรในภาษาสำคัญทั่วโลก ซึ่งทำให้วิธีการนี้ได้รับความนิยมอย่างแพร่หลาย เนื่องจาก UCS-2 สามารถเขียนตัวอักษรในภาษาไทยได้โดยไม่ต้องเปลี่ยนแปลงรูปแบบของตัวอักษร เช่น ตัวอักษร 'ก' ใน UCS-2 จะมีค่าเท่ากับ 0x0E01 แต่ใน ASCII จะมีค่าเท่ากับ 0x47 ซึ่งต้องเปลี่ยนแปลงรูปแบบของตัวอักษรที่เขียนในภาษาไทย
  - รหัส UTF-8 นิยมใช้ในเว็บเพจต่างๆ โดยแต่ละตัวอักษรจะใช้ความยาวตั้งแต่ 1 ไบต์ จนถึง 4 ไบต์ โดยตัวอักษร 128 ตัวแรก คือ รหัส ASCII ใช้ความยาว 1 ไบต์ ส่วนตัวอักษรในภาษาอื่นๆ จะใช้จำนวนไบต์เพิ่มขึ้น

- รหัส **UTF-16** คือ การขยายรหัส UCS-2 ให้ทันสมัยมากขึ้น โดยเพิ่มการเข้ารหัสเป็นขนาด 4 ไบต์ด้วย

การทดลองในหัวข้อที่ [A.3](#) ภาคผนวก A จะเปิดโอกาสให้ผู้อ่านได้ทดลองแปลงตัวอักษรต่างๆ ให้เป็นรหัส ASCII และรหัส Unicode ชนิด UCS-2 ด้วยตนเอง เพื่อสร้างความเข้าใจที่ลึกซึ้งมากขึ้น

## 2.8 สรุปท้ายบท

การคำนวณทางคณิตศาสตร์ของเลขจำนวนเต็มชนิดมีเครื่องหมายและไม่มีเครื่องหมาย จำเป็นต้องตรวจจับการเกิดโอเวอร์โฟล์ว เนื่องจากหารดware หรือโพรเซสเซอร์มีจำนวนบิตในการคำนวณที่จำกัด ปัจจุบัน คือ 32 64 และ 128 บิต ในทำนองเดียวกัน การคำนวณโดยใช้เลขศนนิยมชนิดจุดตรึง (Fixed-point) และจุดลอยตัว (Floating-point) จำเป็นต้องตรวจจับการเกิดโอเวอร์โฟล์วและอันเดอร์โฟล์ว และกรณีอื่นๆ ด้วยเหตุผลที่ซับซ้อนกว่าเลขจำนวนเต็ม ทำให้ใช้ทรัพยากรด้านฮาร์ดแวร์และเวลามากขึ้น

ชนิดและความยาวของข้อมูลที่หลากหลายตามที่สรุปในตารางที่ [2.13](#) ของซอฟต์แวร์คอมพิวเตอร์ ต้องการความสามารถของชีพิญหรือฮาร์ดแวร์ให้รองรับตามเข่นกัน เพื่อให้ประสบการณ์การใช้งานของผู้ใช้ (User Experience) ดีขึ้นเรื่อยๆ ยกตัวอย่างเช่น การใช้ชีพิญมาช่วยคำนวณข้อมูลที่จัดเรียงตัวกันแบบเวกเตอร์ (Vector) เพื่อเพิ่มประสิทธิภาพ เป็นต้น

ตารางที่ 2.13: ชนิด ความยาว ข้อมูล และการประยุกต์ใช้งานเลขฐานสองชนิดต่างๆ ในคอมพิวเตอร์

ชนิด	บิต	ข้อมูล	การประยุกต์ใช้งาน
char	8	ตัวอักษร	ข้อความ อีเมล ชื่อ นามสกุล
unsigned char	8	จุดภาพ	รูปภาพขาวดำ และ Gray Scale
unsigned char	8	จุดภาพ	รูปภาพสี RGB Bitmap JPEG
unsigned int	32/64	พอยน์เตอร์	แอดเดรสชีต์แมเนจเม้นต์ข้อมูล ระบบ 32/64 บิต
unsigned int	32/64	จำนวน	จำนวนอุปกรณ์ IoT จำนวนดาวต่างๆ
int	32/64	จำนวน	เลขจำนวนเต็ม
ทศนิยมจุดตรึง	16-32	เสียง	ข้อมูลเสียงดนตรี
ทศนิยมจุดตรึง	16-32	จุดภาพ	ข้อมูลภาพความละเอียดสูง
float	32	จุดภาพ	ข้อมูลภาพความละเอียดสูง
float	32	ระยะทาง	เกม 3 มิติ
double	64	± ระยะทาง	ระยะทางไปยังดาวต่างๆ นอกโลก
double	64	± น้ำหนัก	น้ำหนักดาวต่างๆ น้ำหนักอนุภาคเล็กๆ

## 2.9 คำถ้ามห้ายบท

1. จงแสดงวิธีบวกเลขจำนวนเต็มฐานสิบและฐานสองชนิดไม่มีเครื่องหมาย ขนาด 8 บิต ต่อไปนี้ และตรวจสอบว่าเกิดโอเวอร์โฟล์วตามสมการที่ (2.43)
  - $255_{10} + 1_{10}$
  - $128_{10} + 127_{10}$
  - $1111\ 1110_2 + 0000\ 0011_2$
  - $1000\ 0000_2 + 0111\ 1111_2$
2. จงแสดงวิธีคูณเลขจำนวนเต็มฐานสองชนิดไม่มีเครื่องหมาย ขนาด 8 บิต ต่อไปนี้ ด้วยวงจรคูณเลขชนิดที่ 1 ในรูปที่ 2.5 ตามตัวอย่างที่ 2.10
  - $0000\ 1111_2 \times 0000\ 1000_2$
  - $0000\ 1111_2 \times 0000\ 1111_2$
  - $1010\ 1010_2 \times 0101\ 0101_2$
  - $1111\ 1111_2 \times 1111\ 1111_2$
3. จงแสดงวิธีคูณเลขจำนวนเต็มฐานสองชนิดไม่มีเครื่องหมาย ขนาด 8 บิต ต่อไปนี้ ด้วยวงจรคูณเลขชนิดที่ 2 ในรูปที่ 2.6 ตามตัวอย่างที่ 2.11
  - $0000\ 1111_2 \times 0000\ 1000_2$
  - $0000\ 1111_2 \times 0000\ 1111_2$
  - $1010\ 1010_2 \times 0101\ 0101_2$
  - $1111\ 1111_2 \times 1111\ 1111_2$
4. จงแสดงวิธีบวกเลขจำนวนเต็มฐานสิบและฐานสองชนิดมีเครื่องหมาย 2's Complement ขนาด 8 บิต ต่อไปนี้ และตรวจสอบว่าเกิดโอเวอร์โฟล์วตามสมการที่ (2.47)
  - $126_{10} + 2_{10}$
  - $128_{10} - 127_{10}$
  - $1111\ 1110_2 + 0000\ 0011_2$
  - $1000\ 0000_2 + 0111\ 1111_2$
5. จงแสดงวิธีบวกเลขทศนิยมชนิดจุดตรีน ขนาด 4.4 บิต ต่อไปนี้ และตรวจสอบว่าเกิดโอเวอร์โฟล์วหรือไม่ (ใช้หลักการเดียวกับเลขจำนวนเต็มฐานสองชนิดไม่มีเครื่องหมาย)
  - $+1111.1110_2 + +0000.0011_2$
  - $+1111.1110_2 + -0000.0011_2$
  - $-1111.1110_2 + +0000.0011_2$
  - $-1111.1110_2 + -0000.0011_2$

6. จงแสดงวิธีบวกเลขทศนิยมฐานสิบต่อไปนี้ ตามมาตรฐาน IEEE754 Single Precision และตรวจสอบว่า เกิดโอเวอร์โฟล์ว/อันเดอร์โฟล์วหรือไม่ ตามตัวอย่างที่ 2.6.5

- $1.25_{10} + 2.50_{10}$
- $1.25_{10} + -2.50_{10}$
- $-1.25_{10} + 2.50_{10}$
- $-1.25_{10} + -2.50_{10}$

7. จงแสดงวิธีคูณเลขทศนิยมฐานสิบต่อไปนี้ ตามมาตรฐาน IEEE754 Single Precision และตรวจสอบว่า เกิดโอเวอร์โฟล์ว/อันเดอร์โฟล์วหรือไม่ ตามตัวอย่างที่ 2.6.7

- $1.25_{10} \times 2.50_{10}$
- $1.25_{10} \times -2.50_{10}$
- $-1.25_{10} \times 2.50_{10}$
- $-1.25_{10} \times -2.50_{10}$

8. จงแปลงเลขฐานสิบหกหรือเลขฐานสองความยาว 32 บิต ต่อไปนี้

- 0x0000 0000
- 0x8000 0000
- 0xFF80 0000
- 0x8F80 0000
- 0x8F80 0001
- 0x8FFF FFFF

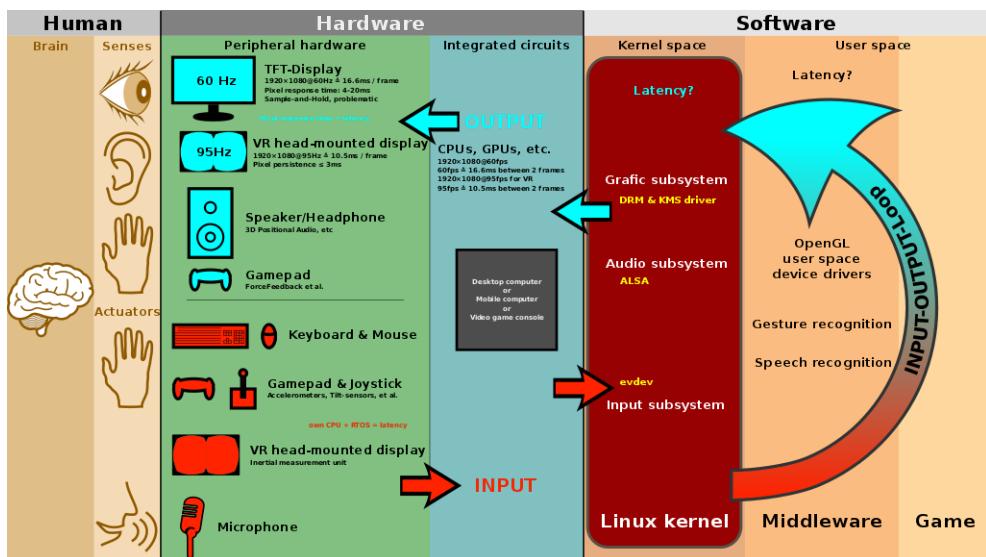
แต่ละตัวแล้วเติมลงในตาราง

- เลขจำนวนเต็มแบบไม่มีเครื่องหมาย โดยใช้สมการที่ (2.2)
- เลขจำนวนเต็มแบบมีเครื่องหมาย ชนิด 2's Complement โดยใช้สมการที่ (2.16)
- เลขทศนิยมชนิดจุดลอยตัว โดยใช้สมการที่ (eq:float:ieee:10) และเว็บเพจ <https://www.h-schmidt.net/FloatConverter/IEEE754.html>

เลขฐานสอง $n=32$ บิต	$X_{10,u}$ ค่าฐานสิบ Unsigned สมการ (2.2)	$X_{10,s}$ ค่าฐานสิบ 2-Comp. สมการ (2.16)	$F_{10,IEEE}$ ค่าฐานสิบ IEEE754 สมการ (2.67)
0x0000 0000			
0x8000 0000			
0xFF80 0000			
0x8F80 0000			
0x8F80 0001			
0x8FFF FFFF			

## บทที่ 3

# ฮาร์ดแวร์และซอฟต์แวร์ของคอมพิวเตอร์ (Computer Hardware and Software)



รูปที่ 3.1: วงรอบการเชื่อมโยงอินพุต-เอาต์พุต (Input-Output Loop) ระหว่าง ผู้ใช้ ฮาร์ดแวร์ และซอฟต์แวร์ (เกม) ที่มา: [wikipedia.org](https://en.wikipedia.org)

รูปที่ 3.1 แสดงว่างรอบการเชื่อมโยงอินพุต-เอาต์พุต (Input-Output Loop) ระหว่าง ผู้ใช้ ฮาร์ดแวร์ และซอฟต์แวร์ (เกม) ซึ่งเริ่มต้นจากการที่ผู้ใช้เรียกใช้งานซอฟต์แวร์ (เกม) นั่นผ่านทางเมนูของระบบปฏิบัติการ เมื่อซอฟต์แวร์เริ่มต้นทำงาน ผู้ใช้สามารถรับรู้ผ่านทางมือ ตาและหู เนื่องจากซอฟต์แวร์สร้างแรงตอบสนอง ภาพและเสียง ผ่านทางฮาร์ดแวร์ เช่น เกมแพด (Gamepad) จอป้า และลำโพง เมื่อสมองผู้ใช้รับรู้ ประมวลผล และตอบสนองต่อซอฟต์แวร์ด้วยการขยับศีรษะ มือและปากผ่านทางฮาร์ดแวร์ เช่น จอแสดงภาพแบบสวมศีรษะ (Head Mounted Display) คีย์บอร์ด มาส์ต เกมแพด ไมโครโฟน เป็นต้น กลับไปหาซอฟต์แวร์ (เกม) ว่างอบนี้จะเกิดต่อเนื่องด้วยความรวดเร็วมาก หากผู้ใช้มีความถันดหรือคุณเคยจะยิงตอบสนองกับเครื่องคอมพิวเตอร์ได้ดียิ่งขึ้น และสามารถเปรียบเทียบได้ว่า ซอฟต์แวร์เดียวกันสามารถตอบสนองได้ดีหรือไม่บนเครื่องฮาร์ดแวร์ที่แตกต่างกัน ดังนั้น เนื้อหาทั้งหมดในบทนี้จึงมีวัตถุประสงค์เหล่านี้

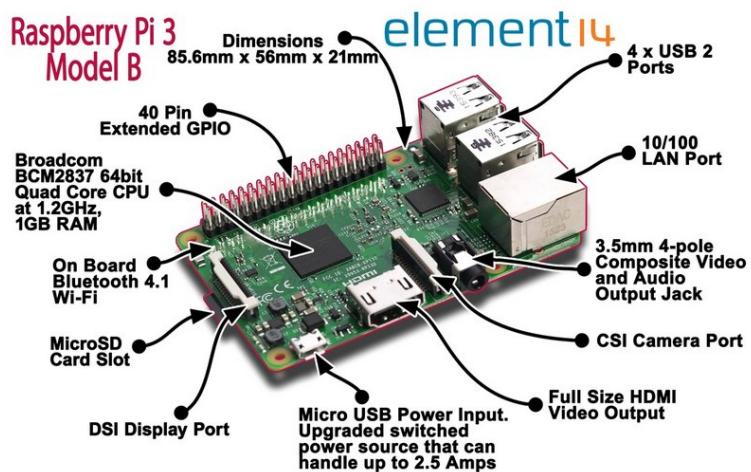
- เพื่อให้ผู้อ่านเข้าใจโครงสร้างโดยองค์รวมของคอมพิวเตอร์ กรณีศึกษาบอร์ด Raspberry Pi3/Pi4

- เพื่อให้ผู้อ่านเข้าใจโครงสร้างและการทำงานของด้านฮาร์ดแวร์ กรณีศึกษา ARM Cortex A53/A72 ภายในชิป BCM2837 บนบอร์ด Raspberry Pi3/Pi4
- เพื่อให้ผู้อ่านเข้าใจโครงสร้างและการทำงานของระบบปฏิบัติการลินุกซ์ (Linux) และโครงสร้างของซอฟต์แวร์ประยุกต์
- เพื่อให้ผู้อ่านเข้าใจการประสานงานระหว่างฮาร์ดแวร์และซอฟต์แวร์ ขบวนการรันซอฟต์แวร์ และการสั่งงานให้ฮาร์ดแวร์ทำงาน
- เพื่อให้ผู้อ่านเข้าใจขบวนการพัฒนาซอฟต์แวร์ด้วยภาษา C/C++ และภาษาแอสเซมบลี (Assembly) ของ ARM

บอร์ด Raspberry Pi3/Pi4 และ Pi4 เป็นคอมพิวเตอร์ขนาดเล็กที่ได้รับความนิยม เนื่องจากราคาไม่สูง การออกแบบฮาร์ดแวร์และซอฟต์แวร์ที่ลงตัว รวมไปถึงมีซอฟต์แวร์ประยุกต์ หรือ แอปพลิเคชัน (Application) ที่หลากหลาย กรณีศึกษางบบอร์ด Raspberry Pi ในตำราเล่มนี้จะหมายถึง บอร์ด Pi3/Pi4 ไมเดล B ในรูปที่ 3.2 นอกจากไมเดล B ซึ่งออกแบบมาเป็นคอมพิวเตอร์ตั้งโต๊ะอย่างง่าย ยังมีบอร์ดไมเดล A มีลักษณะคล้ายกันแต่ไมเดล A จะเน้นการทำงานแบบระบบฝังตัว (Embedded System) เป็นหลัก รายละเอียดเพิ่มเติมที่ [wikidevi.com](http://wikidevi.com) และ [elinux.org](http://elinux.org)

ด้วยเหตุนี้ บอร์ด Pi3/Pi4 ไมเดล B สามารถประยุกต์ใช้ได้หลากหลาย เช่น เครื่องคอมพิวเตอร์ส่วนตัว แท็บเล็ต โน้ตบุ๊ก ราคาราคาประหยัด อุปกรณ์เฝ้าระวังความปลอดภัย/สภาพแวดล้อมในรูปแบบ IoT (Internet of Things) ระบบควบคุมในบ้านและอุตสาหกรรม เซิร์ฟเวอร์สำหรับเครื่องพิมพ์ (Print server) อุปกรณ์เชื่อมต่อสัญญาณอินเทอร์เน็ตไร้สายและอื่นๆ

### 3.1 ฮาร์ดแวร์ของเครื่องคอมพิวเตอร์: บอร์ด Pi3/Pi4



รูปที่ 3.2: ตำแหน่งของอุปกรณ์ต่างๆ บนบอร์ด Pi3/Pi4 ที่มา: [raspberryhome.net](http://raspberryhome.net)

แผ่นวงจรพิมพ์ของบอร์ด Pi3/Pi4 ไมเดล B ในรูปที่ 3.2 ประกอบด้วยชิป BCM2837 ผลิตโดยบริษัท Broadcom Corp. ประเทศสหรัฐอเมริกา นอกเหนือจากชิป BCM2837 ซึ่งภายในมีซีพียู (CPU Central Processing Unit) บอร์ด Pi3/Pi4 มีหน่วยความจำหลัก SDRAM (Synchronous Dynamic Random Access Memory)

ชนิด DDR2 (Double Data Rate 2) ขนาดความจุ 1 กิกะไบต์ (GiB) (GiB) อุปกรณ์เก็บรักษาข้อมูล (Data Storage) ชนิดการ์ดหน่วยความจำ SD (Secure Digital) ขนาดความจุ 8-16 กิกะไบต์ (GB) เชื่อมต่ออินเทอร์เน็ตผ่านระบบสื่อสารไร้สาย หรือผ่านช่องเสียบสายเครือข่ายอีเธอร์เน็ต (Ethernet LAN) บอร์ดมีช่องเสียบ USB สำหรับเชื่อมต่อคีย์บอร์ดและเมาส์ เป็นต้นเพื่อให้ระบบสมบูรณ์ ผู้อ่านสามารถทำความเข้าใจในรายละเอียดด้านต่างๆ ของบอร์ดได้ในตารางที่ 3.1

ตารางที่ 3.1: ตารางสรุปข้อมูลด้านฮาร์ดแวร์และซอฟต์แวร์ของบอร์ด Pi3/Pi4 โมเดล B และลิงก์เชื่อมไปยังหัวข้อที่แสดงรายละเอียดในบทต่างๆ

มอดูล	ภายในชิป (On chip) BCM2837	ในหัวข้อที่
BCM2837	เป็น SoC ผลิตโดยบริษัท Broadcom ประกอบด้วย	3.1.1
ซีพีyu	ARM Cortex-A53/A72 จำนวน 4 แกนประมวลผล ความถี่ 1.2 กิกะเฮิรตซ์	
จีพีyu	Dual (2) VideoCore IV ความถี่ 400 เมกะเฮิรตซ์ (MHz)	
จอ LCD	สาย HDMI เวอร์ชัน 1.3 & 1.4 (ภาพและเสียง)	6.1
จอ LCD	สาย Display Serial Interface (DSI) 15 ขา ประกอบด้วยสัญญาณข้อมูล 2 คู่ สัญญาณคล็อก 1 คู่	6.2
กล้องขนาดเล็ก	สาย Camera Serial Interface (CSI) 15-ขา ประกอบด้วยสัญญาณข้อมูล 2 ช่อง สัญญาณคล็อก 1 ช่อง	6.3
จอทีวี และเสียง	สัญญาณคอมโพสิทวิดีโอ PAL/NTSC แจ็คขนาด 3.5 มม ชนิด 4 ชี้ว้า	6.5 6.4
GPIO	ชี้ว้าต่อนิย德 2.54 มม 40 ขา ประกอบด้วย GPIO 27 ขา +3.3 และ +5V โวลต์	6.11
อุปกรณ์	ภายนอกชิป (Off chip) BCM2837	ในหัวข้อที่
ชิป SDRAM	หน่วยความจำชนิด DDR2 ความจุ 1 กิกะไบต์ (GiB) ชนิดประยุกต์พลั่งงาน	3.1.2 5.5
ชิป USB	ชิป USB 2.0 จำนวน 4 พอร์ต	6.6
ชิป Ethernet	เชื่อมต่ออินเทอร์เน็ตผ่านสาย ด้วยอัตรา 10/100 Mbps	6.7
ชิป WiFi และ Bluetooth	เชื่อมต่ออินเทอร์เน็ตไร้สาย IEEE 802.11 b/g/n อัตราเร็วสูงสุด 150Mbps การเชื่อมต่อไร้สายเวอร์ชัน 4.1	6.8 6.8
การ์ด SD	ความจุ 8-16 กิกะไบต์ (GB)	3.1.4, 7.3
ซอฟต์แวร์	ระบบปฏิบัติการในการ์ดหน่วยความจำ SD	3.3.1
ระบบ	Raspberry Pi OS, Linux, Windows 10 IoT และอื่นๆ	
แหล่งจ่ายไฟ	ช่องเก็ตชนิด microUSB ขนาด 5 โวลต์ 2.5 แอม培ร์	6.14

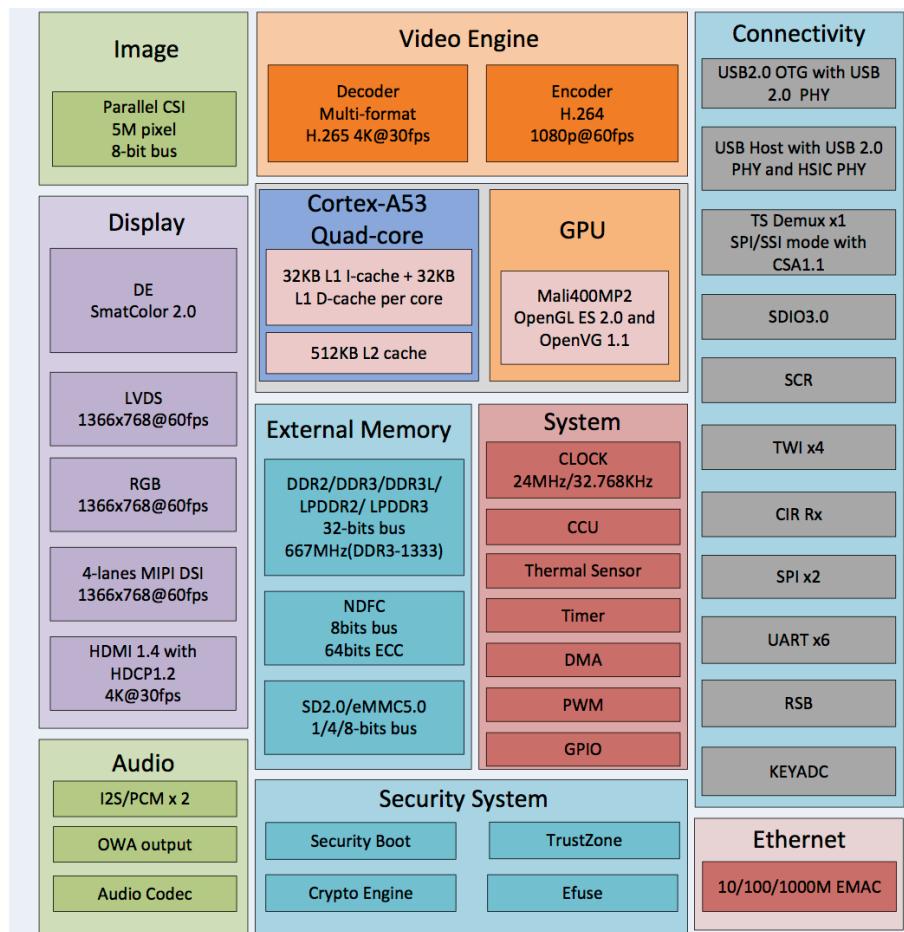
องค์ประกอบสำคัญของบอร์ด คือ ชิป BCM2837/BCM2711 เป็น SoC ย่อมาจาก System on Chip ที่มีซีพีyu ARM รุ่น Cortex A53/A72 จำนวน 4 แกนประมวลผล (Quad-core) รองรับภาษาและเซมบลีของ ARM เวอร์ชัน v8 หรือย่อว่า ARMv8 ซึ่งมีความยาวคำสั่งภาษาเครื่องความยาว 32 และ 64 บิต ซีพีyu ARM Cortex A53/A72 นี้ใช้สัญญาณคล็อก (Clock) ความถี่ 1.20 กิกะเฮิรตซ์ (GHz) ซึ่งแตกต่างจากกรณีของบอร์ด Pi2 ที่ใช้ชิป BCM2836 ซึ่งมีซีพีyuของ ARM รุ่น Cortex A7 รองรับภาษาและเซมบลี ARMv7 ชนิด 32 บิตเท่านั้น ทำงานที่ความถี่เพียง 900 เมกะเฮิรตซ์ แม้ว่าชิป BCM2836 จะมีจำนวน 4 แกนประมวลผล เช่นกัน แต่บอร์ด Pi2 มี

หน่วยความจำเพียง 512 เมบีไบต์ (MiB) เท่านั้น ล่าสุดมุ่งนิธิได้ประกาศรายละเอียดของบอร์ด Pi4 แล้ว

รูปที่ 3.2 แสดงการเขื่อมโยงอุปกรณ์ต่างๆ บนบอร์ด Pi3/Pi4 โดยมีชิป BCM2837 บอร์ด Pi3/Pi4 ไม่เดล B เป็นคอมพิวเตอร์บอร์ดเดียว (Single Board Computer) ราคาเพื่อการศึกษา เป็นรุ่นถัดจาก Raspberry Pi2 (Pi2) ไม่เดล B ออกแบบและพัฒนาโดยองค์กรที่มีชื่อว่า Raspberry Pi Foundation โดยทั้งบอร์ด Pi2 และ Pi3/Pi4 มีขนาดทางกายภาพเท่ากัน (86มม.x56มม.x21มม.) ทำให้ใช้กล่อง (Case) ด้วยกันได้ ตำแหน่งของพอร์ตและคอนเนกเตอร์คล้ายกัน แต่บอร์ด Pi3/Pi4 มีความสามารถในการประมวลผลที่สูงขึ้นและมีประสิทธิภาพดีกว่า ตัวอย่างการเปรียบเทียบบอร์ดรุ่นต่างๆ จัดทำโดยเว็บไซต์ [medium.com](http://medium.com)

ดังนั้น การทดลองที่ 2 ภาคผนวก B เป็นการติดตั้งและใช้งานฮาร์ดแวร์ จะขอให้ผู้อ่านได้ประกอบบอร์ด Pi3/Pi4 เข้ากับจอภาพ LCD ขนาดใหญ่และอุปกรณ์อินพุต/เอาต์พุตอื่นๆ เพื่อใช้เป็นคอมพิวเตอร์ตั้งโต๊ะส่วนบุคคล ประกอบการทดลองทั้งหมด

### 3.1.1 ซีพียู (CPU: ARM Cortex A53/A72)



รูปที่ 3.3: บล็อกໄດอະແກຣມຂອງชີປ AllWinner A64 ທີ່ມີซືບີ່ງ ARM Cortex A53/A72 ຄ້າຍກັບຊີປ BCM2837/BCM2711 ບນບอร์ດ Raspberry Pi3/Pi4 ທີ່ມາ: [Allwinner Technology \(2015a\)](http://Allwinner Technology (2015a)) ແລະ [Allwinner Technology \(2015b\)](http://Allwinner Technology (2015b))

ຊີປ BCM2837 ພິລິຕໂດຍບຣີ່ຫັກ Broadcom ປະກອບດ້ວຍ

- **ซືບີ່ງ (CPU)** ຍ່ອມາຈັກຄໍາວ່າ Central Processing Unit ຮູ່ນ Cortex A53/A72 ອອກແບບໂດຍບຣີ່ຫັກ **ARM** ຈຳນວນ 4 ແກນປະກາດໂດຍແຕ່ລະແກນປະກາດຄື້ອງ ວົງຈະດິຈິທັລ໌ທຳການຕາມສັນຄູານຄືອກຄວາມຖື 1.2

กิจกรรมที่ 3 รองรับคำสั่งภาษาแอสเซมบลี (Assembly) ของ ARM เวอร์ชัน 8A และครอบคลุมเวอร์ชันเก่า ด้วย รายละเอียดเพิ่มเติมอยู่ในหัวข้อที่ [4.9.1](#)

- จีพีью (GPU: Graphic Processing Unit) ชื่อ [VideoCore](#) เป็นหน่วยประมวลผลด้านการแสดงผลบนจอภาพ ในชิป BCM2837 บรรจุจีพีьюจำนวน 2 แกนประมวลผล ทำงานด้วยสัญญาณคลือกความถี่ 400 เมกะเฮิรตซ์ ออกแบบโดยบริษัท BroadCom ทำหน้าที่เป็น Multimedia Co-Processor รองรับ [OpenGL ES](#) เวอร์ชัน 2.0 และ [OpenVG](#)
- มอดูลจัดการวีดีโอ (Video Engine) สามารถดอร์หัสภาพวีดีโอที่ความละเอียด 1080x1920 อัตราเฟรชภาพ 30 เฟรมต่อวินาที มาตรฐาน ITU H.264 ความละเอียด High-Profile
- วงจรเชื่อมต่อหน่วยความจำหลัก SDRAM ชนิด DDR2 (SDRAM Interface)
- วงจรเชื่อมต่อการ์ดหน่วยความจำ SD เวอร์ชัน 2.0 (External Mass Media Controller) ที่มา: [Broadcom \(2012\)](#)
- มอดูลเชื่อมต่ออินพุตและเอาต์พุตต่างๆ ตามรายละเอียดในตารางที่ [3.1](#)

เนื่องจากข้อจำกัดด้านรูปภาพประกอบของชิป BCM2837/BCM2711 ผู้เขียนจึงขอใช้ข้อมูลจากชิปที่มีคุณสมบัติใกล้เคียงแทนด้วยบล็อกไดอะแกรมของชิป A64 จากบริษัท AllWinner ตามรูปที่ [3.3](#) เนื่องจากมีจีพีью Cortex A53/A72 คล้ายกับชิป BCM2837/BCM2711 แต่มอดูลสำคัญๆ ต่างกัน เช่น จีพีyu, Ethernet เป็นต้น ที่มา: [Allwinner Technology \(2015a\)](#) และ [Allwinner Technology \(2015b\)](#)

### 3.1.2 หน่วยความจำหลัก (Main Memory)



รูปที่ 3.4: หน่วยความจำ SDRAM ด้านล่างของบอร์ด Pi3/Pi4 โมเดล B ที่มา: [robo-dyne.com](#)

หน่วยความจำหลักของบอร์ด เป็นชิปหน่วยความจำไดนามิก รัมแบบซิงโครนัส หรือ SDRAM (Synchronous Dynamic RAM) ชนิด DDR2 (Double Data Rate 2) เหมาะสำหรับใช้งานบนอุปกรณ์เคลื่อนที่ (Mobile) เพราะเป็นรุ่นประหยัดพลังงานหรือชนิด LP (Low Power) และจำนวนขาต้องเพื่อประหยัดพื้นที่บนบอร์ด ชิปหน่วยความจำหลักนี้รองรับสัญญาณความถี่คลือกสูงสุด 400 เมกะเฮิรตซ์ ทำให้เกิดความเร็วสูงสุด 800 เมกะบิต/วินาที เนื่องด้วยหน่วยความจำหลักนี้ทำงานตามจังหวะความถี่สัญญาณคลือกที่สูง และตัวถังขนาดเล็ก มีพื้นผิวสัมผัสกับอากาศได้น้อย ความร้อนที่ตัวถังจึงเป็นอุปสรรคต่อการทำงาน ดังนั้น ผู้อ่านควรติดตั้งฮีทชิ้ง

(Heat Sink) เพื่อเพิ่มพื้นที่ผิวสัมผัสกับอากาศในการระบายความร้อนออกจากชิปนี้ ตามการทดลองที่ 2 ภาคผนวก B การติดตั้งและใช้งานฮาร์ดแวร์

ชิปหน่วยความจำหลัก SDRAM นี้ผลิตโดยบริษัท Elpida รุ่น B8132B4PB-8D-F วางอยู่ด้านล่างของบอร์ด Pi3/Pi4 ไม่เดล B ดังรูปที่ 3.4 ขาสัญญาณต่างๆ จึงชื่อนอยู่ใต้ชิป เพื่อลดขนาดฟุตพรินท์ (Foot Print) บนแผ่นวงจรพิมพ์ ปัจจุบันนี้ บริษัท Elpida ได้เปลี่ยนผู้ถือหุ้นหลักเป็น บริษัท Micron Technology และ ดังนั้น หน่วยความจำ SDRAM หมายเลขรุ่น EDB8132B4PB จึงสามารถค้นคว้าเพิ่มเติมได้ตามลิงก์ต่อไปนี้ [www.micron.com](http://www.micron.com) ตำราเล่มนี้จะอธิบาย การทำงานและรายละเอียดเพิ่มเติมของหน่วยความจำ SDRAM นี้และชนิดต่างๆ ในบทที่ 5 ในหัวข้อที่ 5.5.1

### 3.1.3 อุปกรณ์อินพุต/เอาต์พุต (Input/Output Devices)

หัวข้อนี้จะกล่าวเฉพาะอุปกรณ์อินพุตและเอาต์พุตที่จำเป็น ได้แก่ คีย์บอร์ด เม้าส์ จอモニเตอร์ เพื่อใช้งานบอร์ด Pi3/Pi4 เป็นเครื่องคอมพิวเตอร์ตั้งโต๊ะส่วนตัว (Desktop Personal Computer) และทำการทดลอง ต่างๆ ตามตารางเล่มนี้ ส่วนอุปกรณ์อินพุตและเอาต์พุตอื่นๆ ในบอร์ด Pi3/Pi4 ผู้อ่านสามารถอ่านเพิ่มเติมในบทที่ 6 ได้อย่างละเอียด

#### คีย์บอร์ด (Keyboard)

110	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126		
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15			
16	17	18	19	20	21	22	23	24	25	26	27	28	(29)	75	80	85	
30	31	32	33	34	35	36	37	38	39	40	41	42	43	76	81	86	
44	45	46	47	48	49	50	51	52	53	54	55	56	57	90	95	100	105
58		60	131		61	132		133	62		64			91	96	101	106
														92	97	102	
														93	98	103	108
														(94)	99	104	(109)

106-Key Keyboard Position Codes

รูปที่ 3.5: ตำแหน่งและรหัสประจำปุ่ม (Scan Code หรือ Position Code) บนคีย์บอร์ดชนิด 106 ปุ่ม ที่มา: [ps-2.kev009.com](http://ps-2.kev009.com)

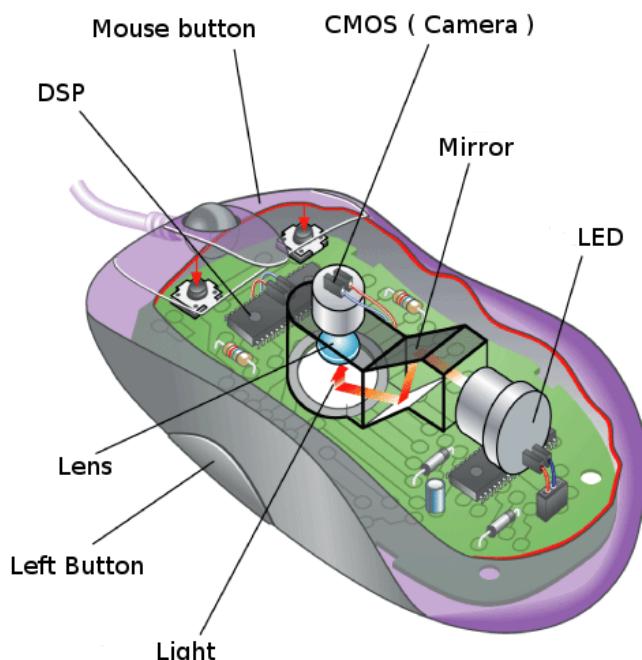
คีย์บอร์ดชนิด 106 ปุ่มนี้ เป็นคีย์บอร์ดที่ออกแบบโดยบริษัท IBM และพัฒนาต่อเนื่องมา มีลักษณะคล้ายกับปุ่มพิมพ์บนเครื่องพิมพ์ตีด โดยปกติจะประกอบด้วย

- ปุ่มตัวอักษร ประกอบด้วยอักษรสำหรับการป้อนข้อมูลที่มีทั้งตัวอักษร A-Z และ a-z ตัวเลข 0-9 และอักษรพิเศษ เช่น @, #, \$, % เป็นต้น
- ปุ่มป้อนข้อมูลตัวเลข โดยจะมีสแกนโค้ดเท่ากับ 90-109 สำหรับการป้อนข้อมูลที่เป็นตัวเลขเรียงตัวกันทางขวาสุดของคีย์บอร์ด เพื่อความสะดวกต่อการใช้งานสำหรับนักบัญชี หรือ ผู้ที่กรอกข้อมูลบ่อยๆ
- ปุ่มฟังก์ชัน ประกอบด้วย ปุ่ม F1 ถึง F12 ซึ่งระบบและซอฟต์แวร์แอปพลิเคชันบางตัวสามารถใช้เป็นคีย์ทางลัด (Short Cut Key) โดยจะมีสแกนโค้ดเท่ากับ 112-123
- ปุ่มควบคุมการทำงาน เช่น Return, Del (Delete), Esc (Escape), Ctrl (Control), Alt (Alternate), Shift, ScrLck (Scroll Lock) เป็นต้น

- ปุ่มควบคุมทิศทาง ได้แก่ ปุ่มลูกศรขึ้น (สแกนโค้ด 83) ลง (สแกนโค้ด 84) ซ้าย (สแกนโค้ด 79) ขวา (สแกนโค้ด 89) นิยมใช้เคลื่อนหรือเปลี่ยนตำแหน่งของเมาส์เซอร์ (Cursor) บนหน้าจอแสดงผลทั้งในโหมดกราฟิก และตัวอักษร

เมื่อผู้ใช้กดปุ่มตั้งแต่ 1 ปุ่มขึ้นไป วงจรดิจิทัลภายในตัวคีย์บอร์ดจะส่งรหัสสแกนโค้ด (Scan Code หรือ Position Code) ของปุ่มที่โดนกดตามที่ได้ออกแบบไว้ในรูปที่ 3.5 ผ่านสายไปยังพอร์ต USB เป็นหลัก หรืออาจจะเป็นคีย์บอร์ดไร้สาย เช่น ชนิดคลื่นวิทยุความถี่ต่ำ ชนิดคลื่นบลูทูธ เป็นต้น ไปยังระบบปฏิบัติตามหลักการเดียวกับรูปที่ 3.1 เพื่อให้โปรแกรมตอบสนองต่อรหัสนั้นๆ เมื่อผู้ใช้กดปุ่มใดๆ ก็ครั้งกีตาม ระบบปฏิบัติการจะรับรู้ได้ผ่านกลไกการเกิดอินเทอร์รัปท์ (Interrupt) ซึ่งมีรายละเอียดเพิ่มเติมในการทดลองที่ 11 ภาคผนวก K

### มาส์ (Mouse)



รูปที่ 3.6: โครงสร้างภายในมาส์ชนิด Optical ประกอบด้วยหลอด LED ส่องแสงกระทบกับพื้นผิวด้านล่างแล้วสะท้อนกลับมายังตัวรับแสงชนิด CMOS ที่มา: [www.pinterest.com](http://www.pinterest.com)

มาส์ คือ อุปกรณ์ที่สามารถแปลงการเคลื่อนที่ของแขนผู้ใช้ในแกน 2 มิติเป็นการเคลื่อนที่ของพอยน์เตอร์ (Pointer) การเคลื่อนที่ของมาส์จะสัมพัทธ์กับตำแหน่งปั๊จจุบันของพอยน์เตอร์บนจอแสดงผล ทั้งนี้ขึ้นอยู่กับความละเอียดของมาส์เอง ความละเอียดของหน้าจอแสดงผล และแอปพลิเคชันนั้นๆ เช่น เกมส์ โปรแกรมวาดและตกแต่งภาพจะต้องการความละเอียดของมาส์สูงกว่ามาส์ปกติ เป็นต้น

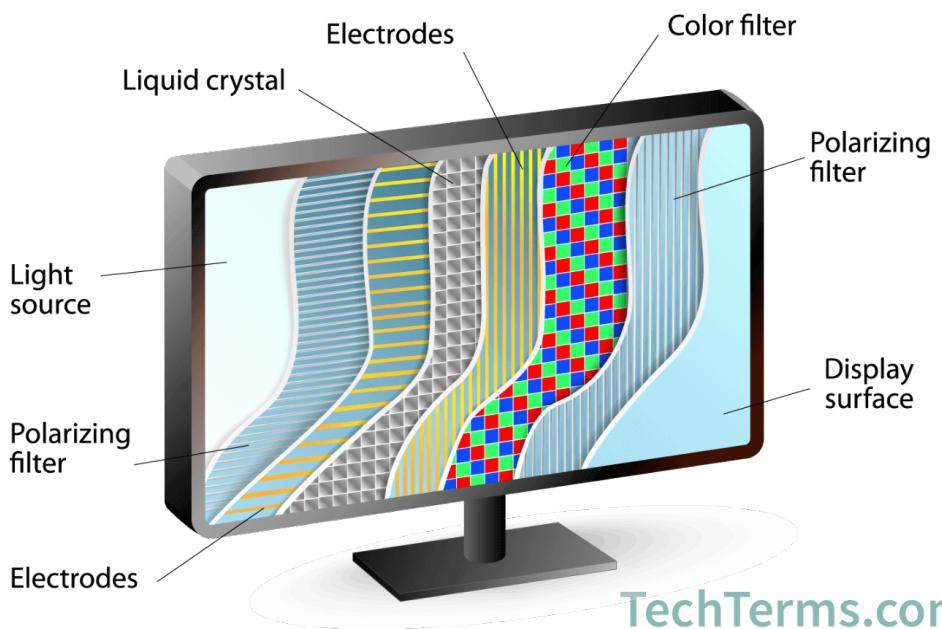
มาส์ส่วนใหญ่มีจำนวนปุ่มคลิกทางซ้ายและขวา ขึ้นอยู่กับการออกแบบของระบบปฏิบัติการ มี 1-3 ปุ่ม สำหรับการใช้งานปั๊จจุบัน มีการเพิ่มวงล้อ (Wheel) บนมาส์สำหรับการเคลื่อนที่แบบพิเศษในมิติต่างๆ การคลิกปุ่มซ้าย (Left Click) ปุ่มขวา (Right Click) การคลิกปุ่มซ้ำ (Double Click) และ การสัมผัสขึ้นหรือลงบนปุ่มมาส์ จะทำให้เกิดแอคชั่นต่างๆ ตามระบบปฏิบัติการ และโปรแกรมที่ใช้งาน ดังนั้น ผู้ใช้จะต้องใช้ความรู้ ความเข้าใจ ตลอดจนถึงความจำ เพื่อให้สามารถใช้งานโปรแกรมเดียวกันแต่ทำงานบนระบบปฏิบัติการที่ต่างกันได้

รูปที่ 3.6 แสดง โครงสร้างภายในมาส์ชนิดօปติกัล (Optical Mouse) ประกอบด้วยหลอดแอลอีดี (LED) ส่องแสงกระทบกับพื้นผิวด้านล่างก่อน แล้วจึงสะท้อนกลับมายังตัวรับแสงชนิด CMOS แสงจากหลอด LED มี

ลักษณะที่แตกต่างกันไปตามชนิดและราคาของมาส์ เช่น หลอด LED เป็นแสงธรรมด้าซึ่งผู้ใช้มองเห็น และหลอด LED แสงเลเซอร์ (Laser) ซึ่งมองไม่เห็นด้วยตาเปล่าแต่มีความละเอียดเพิ่มสูงขึ้นมาก เพื่อประหยัดการใช้แรงและเพิ่มความแม่นยำในการเคลื่อนที่ของมาส์ ซึ่งแสงเลเซอร์จะให้ความละเอียดในการใช้งานสูงกว่า เหมาะกับซอฟต์แวร์ด้านกราฟิก และเกมส์ ด้วยความละเอียด 800-6,000 จุดต่อรั้งทางหนึ่งนิ้ว (DPI: Dot Per Inch)

เมื่อผู้ใช้กดปุ่มใดๆ หรือขับมาส์ ระบบปฏิบัติการจะรับรู้ได้ผ่านกลไกการเกิดอินเทอร์รูปท์ (Interrupt) เช่นเดียวกับคีย์บอร์ด ซึ่งมีรายละเอียดเพิ่มเติมในบทที่ 6 และการทดลองต่างๆ นอกเหนือจากมาส์ อุปกรณ์อื่นๆ ที่ใกล้เคียงสามารถดูแลการใช้มาส์ได้ เช่น ทัชแพด (Touch Pad) จะแสดงผลระบบสัมผัส เสียงพูด ท่าทาง (Gesture) ของมือและนิ้วที่ตรวจสอบโดยกล้องวิดีโอ เป็นต้น ทำให้การใช้งานคอมพิวเตอร์ง่ายลง และได้ประสบการณ์เป็นธรรมชาติมากขึ้น สาขางาน Human Computer Interface จึงมีความสำคัญเพิ่มมากขึ้น เรื่อยๆ

### จอภาพ LCD (Liquid Crystal Display)



TechTerms.com

รูปที่ 3.7: โครงสร้างจอแสดงผลชนิด Color LCD ประกอบด้วยแหล่งกำเนิดแสง (Light Source) ปล่อยแสงที่ลุกขึ้นต่างๆ pragmata ที่มา: [techterms.com](http://techterms.com)

ในอดีตจอภาพของคอมพิวเตอร์มีขนาดใหญ่เทอะทะ น้ำหนักมาก สามารถแสดงผลได้เฉพาะตัวอักษรเท่านั้น และพัฒนาเรื่อยมาจนกลายเป็นจอภาพ LCD (Liquid Crystal Display) ทำให้มีขนาดบางลง น้ำหนักจึงเบาลง จอ LCD สามารถแสดงผลตัวอักษรและกราฟิกทั้งในรูปของ ภาพนิ่ง ภาพเคลื่อนไหว และเกมส์ เพื่อตอบสนองและสร้างปฏิสัมพันธ์กับผู้ใช้งานได้อย่างมีประสิทธิภาพยิ่งขึ้น การแสดงผลของเครื่องคอมพิวเตอร์จะมีทิศทางเป็นสามมิติและเสมือนจริงมากขึ้น เมื่อความก้าวหน้าด้านต่างๆ ถึงพร้อม

โครงสร้างจอภาพ LCD สี (Color LCD) ประกอบด้วยหลอดกำเนิดแสงชนิด LED ด้านหลัง (LED Back Light) ทำหน้าที่ปล่อยแสงที่ลุกขึ้นต่างๆ มาแสดงผลและ pragmata ที่ตามรูปที่ 3.7 สีที่เกิดขึ้นบนจอเกิดจากการผสมกันของจุดภาพย่อย (Sub Pixel) 3 จุดๆ ละสี คือ แดง เขียว น้ำเงิน ด้วยระดับความสว่างที่ไม่เท่ากัน ทำให้เกิดการผสมของสีที่หลากหลายได้ตามจำนวนบิตข้อมูลสี ซึ่งจอ LCD ในปัจจุบันใช้ข้อมูลสีอย่างน้อยสีละ 8 บิต ทำให้เกิดการผสมของสีทั้งหมดคิดเป็น  $2^{3 \times 8} = 16,777,216$  หรือ 16.78 ล้านสี นอกจากนี้ ความละเอียดในการแสดงผลนับตามจำนวนจุดภาพหรือพิกเซล (Pixel) คุณภาพของการแสดงผลขึ้นกับ จำนวนจุดพิกเซล ความ

สว่าง และการจัดเรียงตัวของหลอด LED Back Light การเรียงตัวของจุดภาพย่อย (RGB: Red Green Blue) ในแต่ละพิกเซล ขนาดของพื้นผิวน้ำเง่า และองศาของมุมมองภาพ (Angle) ระยะเวลาในการตอบสนองต่อการแสดงผลแต่ละภาพ (Response Time) เพื่อตอบสนองต่อเนื้อหาที่เปลี่ยนแปลงอย่างรวดเร็ว เช่น การเล่นเกมส์ แอนิเมชันความละเอียดสูง การเชื่อมต่อระหว่างจอภาพกับเครื่องคอมพิวเตอร์ จึงต้องใช้สัญญาณชนิด HDMI รายละเอียดเพิ่มเติมในหัวข้อที่ 6.1

หลักการพื้นฐานของแต่ละจุดภาพย่อย หรือ สับพิกเซล (Subpixel) 1 จุดบนจอแสดงผล LCD ประกอบด้วย ข้าไฟฟ้านินิติโปร่งแสง (Transparent Electrode) สองข้าวเพื่อสร้างสนามไฟฟ้า จึงทำให้เกิดการเรียงตัวของผลึกคริสตัลในของเหลว (Liquid Crystal) หรือการบิดตัวพาคลีนแสงจากเลนส์โพลาไรซ์ (Polarize) ด้านหลัง สามารถเดินทางผ่านเลนส์โพลาไรซ์ด้านหน้า ทำให้ผู้ใช้มองเห็นความสว่างของจุดนั้น ในทางกลับกันเมื่อมีประจุไฟฟ้า ผลึกจะไม่บิดเกลี้ยวด้วยไฟที่แสงทะลุผ่านแกนโพลาไรซ์ที่ตั้งฉากไม่ได้ ผู้ใช้จึงเห็นจุดนั้นเป็นสีดำ

### 3.1.4 อุปกรณ์เก็บรักษาข้อมูล (Data Storage)



รูปที่ 3.8: โครงสร้างของการ์ดหน่วยความจำ SD ชนิด SDHC ความจุ 16 กิกะไบต์ (GB) ประกอบด้วยชิปหน่วยความจำแฟลช วงจรควบคุม และวงจรเชื่อมต่อ ที่มา: [wikipedia.org](https://en.wikipedia.org)

การ์ดหน่วยความจำ SD (Secure Digital) ถูกพัฒนาขึ้นมาโดยองค์กร ชื่อ SD Card Association (SDA) สำหรับใช้งานกับอุปกรณ์เคลื่อนที่ต่างๆ เช่น กล้องถ่ายรูป โทรศัพท์ เครื่องเล่นเพลง เป็นต้น ในเดือนสิงหาคม 1999 โดยความร่วมมือกันของ บริษัท SanDisk Panasonic และ Toshiba ในด้านความจุ (Capacity) ของหน่วยความจำ SD แบ่งเป็นสี่รุ่น ได้แก่ Standard-Capacity (SDSC), High-Capacity (SDHC), eXtended-Capacity (SDXC), และ SDIO ซึ่งรวมฟังก์ชัน input/output กับการบันทึกข้อมูลเข้าด้วยกัน โดยราคาในท้องตลาดจะแปรผันตามความจุ ความจุของการ์ดหน่วยความจำ SD มีแนวโน้มเพิ่มขึ้นตามวัตถุประสงค์การใช้งานเนื่องจากตัวการ์ดหน่วยความจำมีขนาดเล็ก น้ำหนักเบา ความเร็วในการอ่านหรือเขียนที่รวดเร็วทันใจ และอายุการใช้งานยาวนาน

ในด้านโครงสร้างทางกายภาพ การ์ดหน่วยความจำ SD มี 3 ขนาด คือ ขนาดปกติ ขนาดมินิ (Mini) และขนาดไมโคร (Micro) โดยมีตัวแปลงการ์ดขนาดมินิและขนาดไมโครให้เป็นขนาดปกติได้ องค์ประกอบภายในของ การ์ดหน่วยความจำ SD ในรูปที่ 3.8 คือ ชิปหน่วยความจำแฟลช (Flash Memory) ผลิตโดยบริษัท Samsung Electronics ประเทศเกาหลีใต้ สำหรับลักษณะนี้จะกล่าวอธิบายรายละเอียดของหน่วยความจำแฟลชเพิ่มเติมในหัวข้อที่ 7.2

บอร์ด Pi3/Pi4 โมเดล B ใช้อุปกรณ์เก็บรักษาข้อมูลชนิดการ์ดหน่วยความจำไมโคร SD สำหรับระบบปฏิบัติการ บันทึกและเก็บรักษาข้อมูลต่างๆ โดยผู้ใช้สามารถติดตั้งระบบปฏิบัติการ เช่น Raspberry Pi OS,

Ubuntu Linux, Microsoft Windows 10 IoT เป็นต้น ลงบนการ์ดหน่วยความจำในโคร์ SD นี้ ซึ่งผู้อ่านจะได้ฝึกติดตั้งระบบปฏิบัติการ Raspberry Pi OS ในการทดลองที่ 3 ภาคผนวก C

## 3.2 การพัฒนาซอฟต์แวร์หรือโปรแกรมคอมพิวเตอร์

ซอฟต์แวร์ คือ ไฟล์ซึ่งประกอบด้วยชุดคำสั่งภาษาเครื่องและข้อมูลประกอบต่างๆ ซอฟต์แวร์ทำหน้าที่สั่งงานซีพียูและฮาร์ดแวร์อื่นๆ ให้ทำงานตามที่โปรแกรมเมอร์เขียน (Code) หรือพัฒนา (Develop) ซอฟต์แวร์มีโครงสร้างที่ระบบปฏิบัติการแต่ละระบบกำหนดไว้เป็นมาตรฐาน เช่น รูปแบบไฟล์ ELF สำหรับระบบปฏิบัติการตระกูลยูนิกซ์ รูปแบบไฟล์ EXE สำหรับระบบปฏิบัติการ Microsoft Windows เป็นต้น ซอฟต์แวร์คอมพิวเตอร์ทั่วไปแบ่งเป็น 2 ชนิดหลัก ได้แก่

- **ซอฟต์แวร์ระบบ (System software)** เป็นซอฟต์แวร์พื้นฐานที่เครื่องคอมพิวเตอร์ต้องมีประกอบด้วยระบบปฏิบัติการ (Operating System) ทำหน้าที่ดำเนินการด้านอินพุตและเอาต์พุต บริหารจัดการหน่วยความจำและอุปกรณ์เก็บรักษาข้อมูล จัดลำดับการทำงานและการใช้งานทรัพยากรของโปรแกรม เพื่อให้ซอฟต์แวร์ประยุกต์ทำงานได้อย่างถูกต้อง ปลอดภัย และมีประสิทธิภาพ
- **ซอฟต์แวร์ประยุกต์ (Application software)** เป็นซอฟต์แวร์ที่ผู้ใช้นำมาประยุกต์ในการทำงาน โดยผู้พัฒนาซอฟต์แวร์ (Software Developer) พัฒนาหรือเขียนขึ้นด้วยภาษาระดับสูง (High-Level Programming Language) เช่น ภาษา Python C/C++ Java เป็นต้น

การพัฒนาซอฟต์แวร์ทำได้หลายระดับ ขึ้นอยู่กับสิ่งแวดล้อมและความต้องการของซอฟต์แวร์ การเขียนโปรแกรมด้วยภาษาสคริปต์ (Script Language)

- โดยใช้การตีความ (Interpreter-based) เช่น ภาษาไพธอน (Python) ภาษา HTML (HyperText Markup Language) เป็นต้น ด้วยอินเทอร์พรีเตอร์ (Interpreter)
- โดยใช้การประมวล (Compiler-based) เช่น ภาษา C/C++ ภาษาจาวา (Java) เป็นต้น ด้วยคอมไพล์ร์ (Compiler)

การพัฒนาซอฟต์แวร์ด้วยภาษาไพธอน (Python) บนบอร์ด Pi3/Pi4 ได้รับความนิยม เนื่องจากตัวภาษามีความซับซ้อนน้อยกว่าทำให้เรียนรู้เองได้ง่าย มีตัวอย่างโปรแกรมที่นักพัฒนาทั่วโลกเปิดเผยแพร่源代码ผ่านทางเครือข่ายอินเทอร์เน็ต ผู้อ่านสามารถค้นคว้าเพิ่มเติมได้ที่ [python.org](http://python.org)

ตำราเล่มนี้จะเน้นการพัฒนาซอฟต์แวร์โดยใช้คอมไпал์ร์และภาษาระดับสูง C/C++ บนระบบปฏิบัติการตระกูลยูนิกซ์เป็นกรณีศึกษา ตามหลักสูตรวิศวกรรมคอมพิวเตอร์ เนื่องจากผลลัพธ์ที่ได้จากการคอมไпал์ (Compile) ภาษา C/C++ คือ คำสั่งภาษาแอสเซมบลี และ คำสั่งภาษาเครื่อง (Machine Code) ซึ่งจะบรรจุอยู่ในไฟล์รูปแบบ ELF ตามที่กล่าวไปแล้วในหัวข้อที่ 3.3.2

### 3.2.1 โครงสร้างของชอร์สโค้ดโปรแกรมภาษา C/C++

การเขียนโปรแกรมด้วยภาษาระดับสูง (High-level language) เช่น ภาษา C/C++ เพื่อแก้โจทย์หรือปัญหาที่ต้องการความรวดเร็ว โปรแกรมเมอร์สามารถเขียนโปรแกรมและสร้างสรรค์ผลงานได้ง่ายกว่าภาษาระดับล่าง เช่น ภาษาแอสเซมบลี และภาษาเครื่อง เนื่องจากภาษาระดับสูงมีลักษณะใกล้เคียงกับประโยคภาษาอังกฤษ ภาษาระดับสูงจะไม่ยึดติดกับเครื่องหรือฮาร์ดแวร์ (Machine Independent) ในทางตรงกันข้าม ภาษาแอส

The screenshot shows a software development environment with a toolbar at the top labeled 'Build target: Debug'. Below the toolbar is a menu bar with '<global>' selected. A 'Management' window is open on the left, showing a 'Projects' tab with a 'Hello\_world' workspace containing a 'Sources' folder and a file named 'main.c'. The main editor window on the right displays the following C code:

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main()
5 {
6     printf("Hello world!\n");
7     return 0;
8 }
9

```

รูปที่ 3.9: ตัวอย่างซอฟต์แวร์โค้ดภาษา C ฟังก์ชันหลักชื่อ main() เมื่อทำงานเสร็จสิ้นจะรีเทิร์นค่า 0 ที่มา: [zen-tut.com](http://zen-tut.com)

เช่มบลีและภาษาเครื่องจะผูกติดกับฮาร์ดแวร์หรือ ซีพียูของเครื่องคอมพิวเตอร์ (Machine Dependent) หากโปรแกรมเมอร์สามารถใช้คำสั่งภาษาแอสเซมบลีในการสั่งงานฮาร์ดแวร์ของเครื่องคอมพิวเตอร์ จะยิ่งช่วยให้โปรแกรมนั้นๆ ทำงานได้เต็มประสิทธิภาพ สำหรับนักเขียนซอฟต์แวร์ คำแนะนำจะใช้ภาษา C และภาษาแอสเซมบลีของ ARM เป็นหลัก เพื่อเข้าถึงระบบปฏิบัติการ ภาษาเครื่อง และฮาร์ดแวร์ของ ARM ได้ลึกซึ้งมากกว่าภาษาประเภทต่อไปนี้ Python

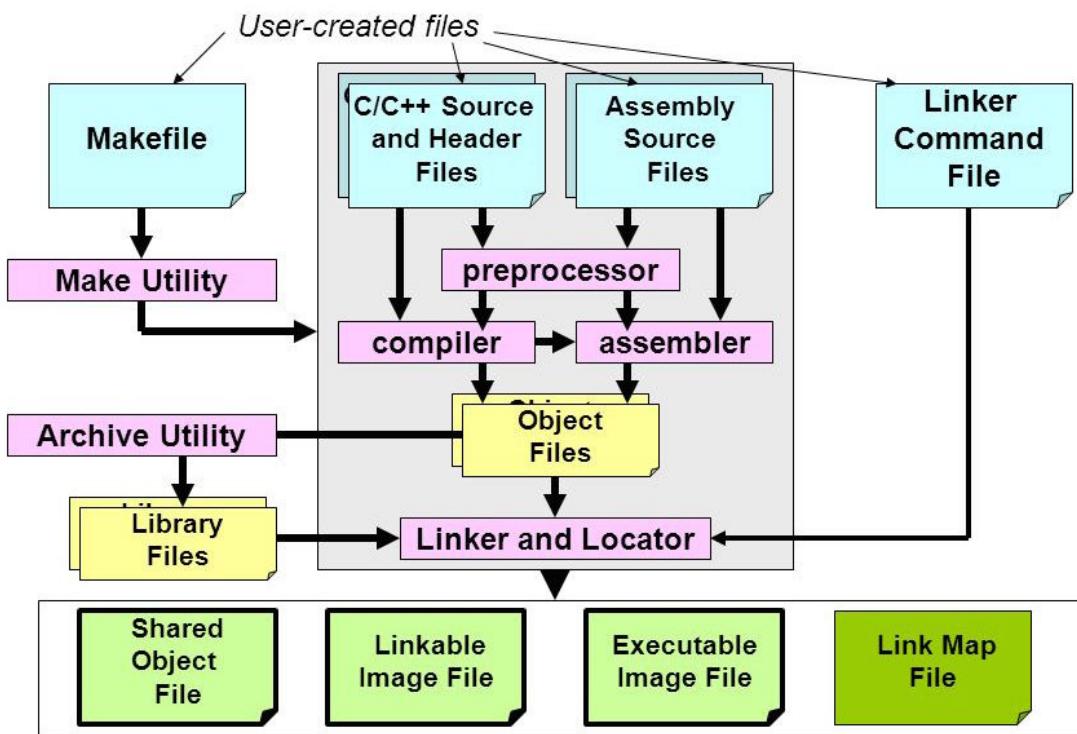
ตัวอย่างซอฟต์แวร์โค้ดภาษา C ในรูปที่ 3.9 ชื่อไฟล์ main.c ฟังก์ชันหลักชื่อ main() เป็นฟังก์ชันที่โปรแกรมเริ่มต้นทำงาน เมื่อทำงานเสร็จสิ้นโปรแกรมจะรีเทิร์นค่า 0 ซึ่งเป็นเลขจำนวนเต็มค่าหนึ่ง ซึ่งมักใช้บ่งบอกว่าเป็นการทำงานเสร็จสิ้นโดยไม่มีปัญหาหรือข้อผิดพลาด และหากโปรแกรมรีเทิร์นค่าอื่นๆ ที่ไม่ใช่ 0 จะบ่งบอกว่ามีความผิดพลาดได้

### 3.2.2 การแปลซอร์สโค้ดภาษา C/C++ ให้กลายเป็นโปรแกรมคอมพิวเตอร์

ภาษา คอมพิวเตอร์ ระดับ สูง (High-level language) มี ศักยภาพ สูง ที่ จะ พัฒนา เป็น ซอฟต์แวร์ หรือ โปรแกรม หรือแอปพลิเคชัน ได้ สะดวก โปรแกรมเมอร์ สามารถ เขียน โปรแกรม เป็น ไฟล์ ภาษา ต่างๆ เรียก โดย รวม ว่า ซอฟต์แวร์ (Source code) รูปที่ 3.10 แสดง ขั้นตอน การ พัฒนา ซอฟต์แวร์ รีเริ่ม จาก ซอฟต์แวร์ คอมพิวเตอร์ และ แปล ให้ เป็น ไฟล์ Executable หรือ โปรแกรม ใน ระบบ ปฏิบัติ ภูมิ นิก ซ์ กล่อง สี ชัม พู ใน รูป คือ โปรแกรม ระบบ ที่ จำ เป็น ประ กอบ ด้วย คอมไพล์ แอสเซมเบลอร์ (Assembler) และ ลิง ก์ กอร์ (Linker) เป็น ต้น

คอมไพล์ เป็น โปรแกรม คอมพิวเตอร์ ชนิด นี้ ทำ หน้าที่ แปล ซอร์ส โค้ด ภาษา สูง เช่น ภาษา C/C++ ให้ กลาย เป็น คำ สั่ง ภาษา แอสเซมบลี และ คำ สั่ง ภาษา เครื่อง ใน ที่ สุด และ จึง จัด เรียง คำ สั่ง ภาษา เครื่อง ให้มี โครงสร้าง ตาม ไฟล์ ไฟล์ รูปแบบ ELF ซึ่ง อธิบาย แล้ว ใน หัวข้อ ที่ 3.3.2 คอมไпал์ หลัก ใน ระบบ ปฏิบัติ ภูมิ Unix/Linux คือ GCC ผู้ อ่าน สาม า ร ค ั น ค ว า ราย ล ะ อ ย ด เพิ่ม เติ่ม ได้ ที่ [gnu.org](http://gnu.org)

คอมไпал์ ทุก ตัว ต้อง ทำ ค ว า ใจ ตัว ไฟล์ ซอร์ส โค้ด ต ่ ง ๆ ที่ โปร แกร บ เม อ ร์ พัฒนา ไว้ (กล่อง สี ฟ้า) คอมไпал์ จะ อ่าน ไฟล์ ซอร์ส โค้ด เพื่อ สแกน ตัว อ ก ษ ร ที่ เป็น เนื้อ โปร แกร บ จัด แยก โ ท ค ี น (Token) ตรวจ คำ สะ กด แปล ค ว า หมาย ของ โ ท ค ี น สร้าง แผน ภูมิ ต้น ไม้ ไ ย า ภ ร ณ์ (Abstract Syntax Tree) เพื่อ ตรวจ สอบ ไ ย า ภ ร ณ์ และ การ วิเคราะห์ ค ว า หมาย (Semantic Analysis) ของ ประ โย ค ต ่ ง ๆ จน ประ ชา จ ก ข ้อ ผ ด ล า ด คอมไpal์ จะ



รูปที่ 3.10: ขั้นตอนการพัฒนาซอฟต์แวร์จากภาษา C/C++ ให้เป็นโปรแกรมประยุกต์ (Application Program) หรือย่อว่า แอป ที่มา: [slideplayer.com](#)

ดำเนินการปรับปรุงการทำงานของโปรแกรมที่จะคอมไพล์ให้ดีขึ้น เรียกว่า การอปติไมซ์ (Optimize) โดยการสร้างกราฟการไหลของข้อมูล (Data-Flow graph) เพื่อประกอบการทำอปติไมเซชัน (Optimization) ทำให้โปรแกรมทำงานได้รวดเร็วขึ้น หลังจากนั้น คอมไพล์러จะสร้างคำสั่งภาษาเครื่อง (Machine Code) ตามชนิดของชีพิญที่ต้องการ ซึ่งตัวเปลี่ยนนี้จะใช้ชีพิญของ ARM เป็นกรณีศึกษา ผู้อ่านสามารถศึกษารายละเอียดของคอมไเพลอร์เพิ่มเติมได้ที่ [wikipedia](#) โปรแกรมเมอร์จะกำหนดความสัมพันธ์หรือความเชื่อมโยงระหว่างชอร์สโค้ดเหล่านั้นใน Makefile เพื่อให้คอมไเพลอร์สามารถสร้าง (Generate) ไฟล์ผลลัพธ์ (กล่องสีเขียว) ตามที่โปรแกรมเมอร์ต้องการ คือ ผู้อ่านควรทำการทดลองที่ 5 ภาคผนวก E เพื่อทดลองสร้างและใช้ Makefile เพื่อคอมไเพล์และลิงก์ โปรแกรมด้วยภาษา C และภาษาแอสเซมบลีของ ARM ในลำดับถัดไป

- ไฟล์โปรแกรม (Executable Image File) ในรูปที่ 3.10 คือ ไฟล์ที่ได้จากการลิงก์หรือเชื่อมกับไฟล์อื่นๆ เช่นไฟล์ไลบรารี ซึ่งคำสั่งภาษาเครื่องและข้อมูลภายในไฟล์เหล่านี้จะถูกจัดเรียงตามรูปแบบไฟล์ ELF ในหัวข้อที่ 3.3.2 ขั้นการลิงก์จะกล่าวโดยละเอียดในหัวข้อที่ 3.2.4 ถัดไป ไฟล์โปรแกรมส่วนใหญ่จะมีชื่อเต็มเมื่มีนามสกุล หมายเหตุ หากผู้ใช้มีกำหนดชื่ออื่น คอมไเพลอร์จะตั้งชื่อไฟล์ให้อัตโนมัติว่า **a.out**
- ไฟล์อืบเจกต์ คือ ไฟล์อืบเจกต์ (กล่องสีเหลือง) ที่ได้จากการคอมไเพล์ชอร์สโค้ดต่างๆ และระบบปฏิบัติการอนุญาตให้นำไปลิงก์ และโหลดในระหว่างที่รันโปรแกรมได้ ไฟล์อืบเจกต์ส่วนใหญ่จะมีชื่อตามด้วย .o รายละเอียดเพิ่มเติมที่ [yolinux.com](#)
- ไฟล์ไลบรารี (Linkable Library File) คือ ไฟล์ที่ได้จากการคอมไเพล์โดยนักพัฒนาต่างๆ เพื่อนำมาลิงก์กับไฟล์หลักให้สมบูรณ์ ซึ่งไลบรารีหลักในการพัฒนาโปรแกรมภาษา C ชื่อว่า glibc ไฟล์ไลบรารีส่วนใหญ่จะมีชื่อตามด้วย .lib หรือ .a ผู้อ่านสามารถค้นคว้ารายละเอียดเพิ่มเติมได้ที่ [gnu.org](#)
- ไฟล์แผนผังการลิงก์ (Link Map File) เพื่ออธิบายการเชื่อมโยงกันของตัวแปรต่างๆ ในชอร์สโค้ด ไฟล์

ชนิดนี้ส่วนใหญ่จะมีชื่อตามด้วย .map

ไฟล์ทั้งหมดนี้จะบรรจุคำสั่งภาษาเครื่องในรูปของเลขฐานสองหรือย่อให้อยู่ในรูปของเลขฐานสิบหก และข้อมูลต่างๆ ตามที่อธิบายในบทที่ 2 ในรูปของเลขฐานสองหรือย่อให้อยู่ในรูปของเลขฐานสิบหก เช่นกัน ดังนั้น เช็คเมนต์ต่างๆ ของไฟล์รูปแบบ ELF จะเป็นสิ่งกำกับว่า เลขฐานสองหรือเลขฐานสิบหกแต่ละค่าในเช็คเมนต์เหล่านั้น คือ คำสั่ง หรือ ข้อมูล ยกตัวอย่างเช่น

- เลขฐานสองหรือเลขฐานสิบหกที่จัดเรียงอยู่ในเท็กซ์เช็คเมนต์ คือ คำสั่งภาษาเครื่อง เท่านั้น
- เลขฐานสองหรือเลขฐานสิบหกที่จัดเรียงอยู่ในดาต้าเช็คเมนต์ คือ ข้อมูล เท่านั้นซึ่งอาจจะมีค่าเป็นเลขจำนวนเต็มถ้าตำแหน่งตรงกับตัวแปรชนิดจำนวนเต็ม หรือมีค่าเป็นเลขทศนิยมชนิดจุดลอยตัว ตามมาตรฐาน IEEE754 ในบทที่ 2 ทั้งนี้ขึ้นอยู่กับการประกาศชนิดตัวแปร รายละเอียดเพิ่มเติมในหัวข้อถัดไป

ผู้อ่านสามารถเสริมสร้างทักษะการพัฒนาโปรแกรมและความเข้าใจจาก การทดลองที่ 5 ภาคผนวก E และ การทดลองอื่นๆ หัวข้อต่อไปนี้จะอธิบายโครงสร้างของชอร์สโค้ดภาษาแอสเซมบลีของ ARM ซึ่งมีลักษณะใกล้เคียงกับภาษาเครื่องแต่โปรแกรมเมอร์ยังสามารถทำความเข้าใจได้ ก่อนที่ผู้อ่านจะได้รู้จักกับคำสั่งภาษาเครื่องในรูปของเลขฐานสองในหัวข้อที่ 3.2.5

### 3.2.3 โครงสร้างของชอร์สโค้ดภาษาแอสเซมบลีของ ARM

ภาษาแอสเซมบลี (Assembly language) เป็นคำสั่งภาษาคอมพิวเตอร์ที่อยู่ในรูปของคำย่อ (Mnemonic) จากภาษาอังกฤษ ทำให้โปรแกรมเมอร์เข้าใจง่ายกว่าตัวเลขฐานสองที่เป็นคำสั่งภาษาเครื่อง (Machine Code) ในตำนานี้จะใช้ภาษาแอสเซมบลีของ ARM Cortex A เวอร์ชัน 32 บิตเป็นหลัก โดยมีรายละเอียดเพิ่มเติมในบทที่ 4

ตารางที่ 3.2 แสดงให้เห็นถึง โครงสร้างหลักของชอร์สโค้ดภาษาแอสเซมบลีของ ARM แบ่งเป็น .text หมายถึง เท็กซ์เช็คเมนต์ และ .data คือ ดาต้าเช็คเมนต์ ตามลำดับ ส่วนคำสั่งภาษาแอสเซมบลี (Assembly language) ในบรรทัดอื่นๆ อยู่ในรูปของคำภาษาอังกฤษที่โปรแกรมเมอร์สามารถทำความเข้าใจได้ และมีความใกล้เคียงกับคำสั่งภาษาเครื่อง ประกอบด้วย พังก์ชันชื่อ main แบ่งเป็น 3 คอลัมน์ ดังนี้

- คอลัมน์ชัย ใช้กำหนดชื่อเลเบล (Label) ต่างๆ หมายถึง คำภาษาอังกฤษที่ตามด้วยสัญลักษณ์ :
- คอลัมน์กลาง ใช้กำหนดคำสั่ง (Operation) ว่าต้องการจะให้ชีพิญทำอะไร
- คอลัมน์ขวา ใช้ระบุจิสเตอร์ หรือ แอดเดรส หรือ เลเบล หรือ ค่าคงที่สำหรับคำสั่งในคอลัมน์ตรงกลาง

ตารางที่ 3.2: ตัวอย่างโปรแกรมภาษาแอสเซมบลีเพื่อประกาศและตั้งค่าเริ่มต้นให้ตัวแปรขนาด 32 บิต จำนวน 4 ตัวแปร และวนรอบบวกค่าโดยใช้ตัวแปรพอยน์เตอร์

บรรทัดที่	เลbel	คำสั่ง	รีจิสเตอร์ หรือ แอดเดรส เลbel หรือ ค่าคงที่
1		.text	
2		.global main	
3	main:		
4		LDR	R1, =M
5		LDR	R1, [R1]
6		LDR	R2, POINTR
7		MOV	R0, #0
8	LOOP:	LDR	R3, [R2], #4
9		ADD	R0, R0, R3
10		SUBS	R1, R1, #1
11		CMP	R1, #0
12		BGT	LOOP
13		LDR	R4, =SUM
14		STR	R0, [R4]
15		BX	LR
16		.data	
17	SUM :	.word	#0
18	M:	.word	#4
19	NUM:	.word	3, 5, 7, 9
20	POINTR:	.word	NUM

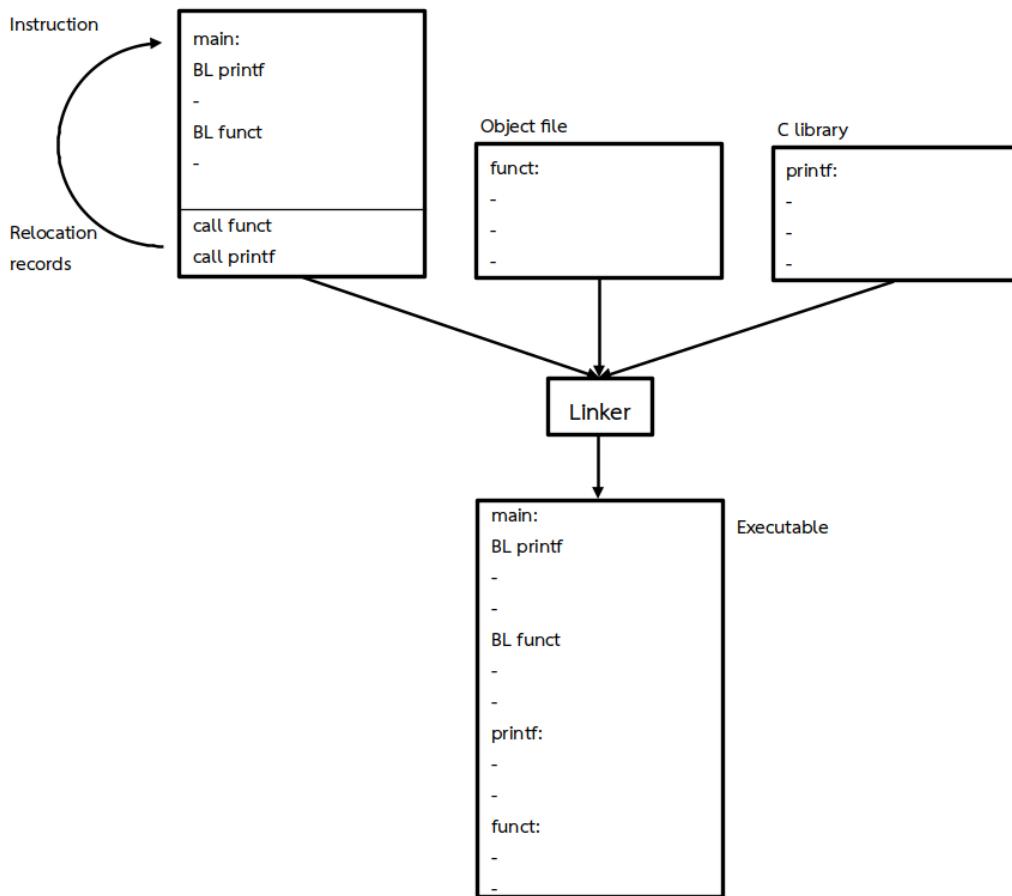
ตัวอย่างโปรแกรมในตารางที่ 3.2 ผู้เขียนสามารถอธิบายตามหมายเลขอรรถทัดได้ดังนี้

1. .text หมายถึง ตำแหน่งเริ่มต้นของเทกซ์ต์เชิงเมนต์ ตามที่กล่าวในหัวข้อ 3.3.2
2. .global main หมายถึง พังก์ชันชื่อ main ซึ่งจะตรงกับพังก์ชัน main() ในภาษา C/C++
3. main หมายถึง เลเบลชื่อ main กำหนดตำแหน่งเริ่มต้นของพังก์ชัน main
4. คำสั่ง LDR ย่อมาจาก Load Data Register ดังนั้น LDR R1, =M คือ การโหลดค่าแอดเดรสของตัวแปร M มาเก็บในรีจิสเตอร์ R1
5. คำสั่ง LDR R1, [R1] นำเอาหมายเลขแอดเดรสของตัวแปร M ที่อยู่ในรีจิสเตอร์ R1 ไปอ่านค่าของ M มาบรรจุในรีจิสเตอร์ R1
6. คำสั่ง LDR R2, POINTR คือ การโหลดค่าของตัวแปร POINTR ซึ่งเก็บแอดเดรสของตัวแปร NUM มาเก็บในรีจิสเตอร์ R2
7. คำสั่ง MOV ย่อมาจาก MOVE ดังนั้น MOV R0, #0 คือ การตั้งค่าของรีจิสเตอร์ R0 ให้มีค่าเป็น 0
8. คำสั่ง LDR R3, [R2], #4 คือ การโหลดค่าของตัวแปร NUM[0] มาเก็บในรีจิสเตอร์ R3 แล้วจึงเพิ่มค่า R2 เป็น R2+4 โดยคำสั่งนี้จะกำหนดให้มีเลเบล ชื่อ LOOP ซึ่งต้องการจะให้เกิดการทำซ้ำ

9. คำสั่ง ADD หมายถึง การบวกค่าภายในรีจิสเตอร์ โดย  $R0=R0+R3$  และเก็บผลรวมในรีจิสเตอร์ R0
10. คำสั่ง SUBS ย่อมาจาก **Subtract** และอปเดตผลลัพธ์ในรีจิสเตอร์สถานะ (CPSR: Current Program Status Register) ซึ่งรายละเอียดจะกล่าวในบทที่ 4 ดังนั้น R1 จะลดลง 1 หน่วย
11. คำสั่ง CMP R1, 0 หมายถึง การเปรียบเทียบ (Compare) รีจิสเตอร์ R1 กับค่าคงที่ 0
12. คำสั่ง BGT ย่อมาจาก **Branch Greater Than** คำสั่งนี้จะตรวจสอบผลของคำสั่ง CMP ก่อนหน้า โดย
  - หาก R1 มีค่ามากกว่า 0 และ การทำงานจะกลับไปเริ่มที่คำสั่งที่ตรงกับเลbel LOOP ดังนั้น โปรแกรมนี้จะวนรอบ  $M=4$  ครั้ง เพื่อบรรคค่าของ NUM[0] จนถึง NUM[3]
  - หาก R1 มีค่าน้อยกว่าหรือเท่ากับ 0 ซึ่งจะทำงานที่คำสั่งต่อไป คือ STR R0, [R4] เพราะวนครับจำนวน  $M=4$  รอบแล้ว
13. คำสั่ง LDR R4, =SUM คือ การโหลดค่าแอดเดรสของตัวแปร SUM มาเก็บในรีจิสเตอร์ R4 โดยที่สัญลักษณ์  $=SUM$  หมายถึง แอดเดรสของตัวแปร SUM
14. คำสั่ง STR ย่อมาจาก **Store Register** คือ การนำผลรวมในรีจิสเตอร์ R0 ไปเก็บไว้ที่หน่วยความจำ Mem[R4] ซึ่งตรงกับแอดเดรสของตัวแปร SUM
15. BX LR บอกจุดสิ้นสุดของชอร์สโค้ด เมื่อโปรแกรมเสร็จสิ้นการทำงานจะรีเทิร์นกลับไปหาฟังก์ชันที่เป็นผู้เรียกฟังก์ชัน (Caller Function) นี้
16. .data หมายถึง ตำแหน่งเริ่มต้นของ DATA เซกเมนต์
17. เลbel **SUM** ทำหน้าที่เป็นชื่อของตัวแปรชนิดจำนวนเต็ม (Integer) ขนาด 32 บิตชนิด 2's Complement และถูกกำหนดค่าเริ่มต้นให้มีค่าเป็น 0
18. เลbel **M** ทำหน้าที่เป็นชื่อของตัวแปรชนิดจำนวนเต็ม (Integer) ขนาด 32 บิตชนิด 2's Complement ถูกกำหนดค่าเริ่มต้น ให้มีค่าเป็น 4 หมายถึง จำนวนสมาชิกของตัวแปรอาร์เรย์ชื่อ NUM
19. เลbel **NUM** ทำหน้าที่เป็นชื่อของตัวแปรชนิดอาร์เรย์ของเลขจำนวนเต็มที่จะถูกกำหนดค่าเริ่มต้น ให้มีค่า เป็น 3, 5, 7, 9 ตามลำดับ โดย  $NUM[0] = 3$ ,  $NUM[1] = 5$ ,  $NUM[2] = 7$  และ  $NUM[3] = 9$
20. เลbel **POINTR** ทำหน้าที่เป็นชื่อของตัวแปรที่ถูกกำหนดค่าเริ่มต้นเป็นแอดเดรสของ NUM ในการพัฒนา โปรแกรมภาษา C/C++ เรียกตัวแปรชนิดนี้ว่า พอยน์เตอร์ (Pointer) โดยตัวแปรชนิดพอยน์เตอร์จะเก็บ แอดเดรสด้วยพื้นที่ขนาด 1 Word หรือเท่ากับ 4 ไบต์ เป็นตัวเลขจำนวนเต็มชนิด Unsigned ตามชนิด ของระบบปฏิบัติการ ซึ่งในตัวเราถูกกำหนดเป็นเวอร์ชัน 32 บิต

### 3.2.4 การรวมไฟล์อ้อมเจกต์ (Object) ด้วยลิงก์เกอร์ (Linker)

ลิงก์เกอร์ (Linker) เป็นหนึ่งในซอฟต์แวร์ระบบที่มีความสำคัญต่อขั้นตอนการพัฒนาโปรแกรม รูปที่ 3.10 แสดงการทำงานของลิงก์เกอร์เพื่อรวมไฟล์อ้อมเจกต์หลัก ไฟล์อ้อมเจกต์เสริม และไฟล์ไลบรารีของภาษา C เข้าด้วยกัน ซึ่งได้แก่ ไลบรารี glibc และไลบรารี wiringPi ในการทดลองที่ 10 ภาคผนวก J และการทดลองที่ 11 ภาคผนวก K



รูปที่ 3.11: ตัวอย่างการลิงก์ (Link) หรือรวมไฟล์อีบเจกต์หลัก ไฟล์อีบเจกต์เสริม และไฟล์ไลบรารีเข้าด้วยกัน เป็นไฟล์โปรแกรมหรือแอปพลิเคชัน ที่มา: [google.com](http://google.com)

ไฟล์อีบเจกต์ (\*.o) คือ ไฟล์ที่รวบรวมฟังก์ชันที่นักพัฒนาเก็บไว้ส่วนตัว ซึ่งตอนแรกมีชอร์สโค้ด แล้วแต่ไม่จำเป็นต้องคอมไพล์ใหม่ อีกรอบ ทำให้ประหยัดเวลาสำหรับการคอมไпал์ ไฟล์ไลบรารี คือ ไฟล์อีบเจกต์ที่รวบรวมฟังก์ชันสำเร็จรูปที่ถูกคอมไпал์แล้ว พัฒนาอย่างต่อเนื่องจนนำไปใช้ แล้วเปิดให้นักพัฒนาอื่นๆ ดาวน์โหลดไปใช้งานผ่านทางคอมไเพเลอเรอร์หลัก ซึ่งว่า GCC เป็นต้น ไฟล์ทั้งหมดนี้อยู่ในรูปแบบ ELF ซึ่งมีรายละเอียดในหัวข้อที่ 3.3.2 รูปที่ 3.11 แสดงตัวอย่างการลิงก์ (Link) หรือรวมไฟล์อีบเจกต์หลัก ไฟล์อีบเจกต์เสริม และไฟล์ไลบรารีเข้าด้วยกันให้เป็นไฟล์โปรแกรมหรือแอปพลิเคชันที่สมบูรณ์ หมายเหตุ กล่องสีเหลืองในรูปคือ ไฟล์ที่บรรจุคำสั่งภาษาเครื่องและข้อมูลเป็นเลขฐานสอง แต่แสดงเป็นคำสั่งภาษาแอสเซมบลีแทนเพื่อให้ผู้อ่านเข้าใจได้ง่าย คำอธิบายเพิ่มเติมในหัวข้อที่ 4.8 เรื่องการเรียกใช้ฟังก์ชัน และการทดลองที่ 7 ภาคผนวก G

กล่าวโดยสรุป การลิงก์เป็นการขยายขีดความสามารถของโปรแกรมที่พัฒนาใหม่จากไฟล์อีบเจกต์ และไฟล์ไลบรารีที่มีอยู่เดิม โปรแกรมเมอร์สามารถเขียนโปรแกรมตามที่ต้องการโดยไม่จำเป็นต้องพัฒนาฟังก์ชันอื่นๆ ทั้งหมดด้วยตนเอง แต่สามารถเรียกใช้งานฟังก์ชัน (Function Call) ที่มีผู้พัฒนาไว้แล้ว ตรงตามหลักการที่สำคัญ พัฒนาซอฟต์แวร์ที่ดี เรียกว่า **มอดูลาริตี้** (Modularity) รายละเอียดสามารถศึกษาเพิ่มเติมด้วยภาษา C จากการทดลองที่ 5 ภาคผนวก E และภาษาแอสเซมบลีในการทดลองที่ 6 ภาคผนวก F เป็นต้นไป

### 3.2.5 ตัวอย่างของคำสั่งภาษาเครื่อง (Machine Code) ของ ARM

ผู้อ่านได้ทำการเข้าใจกระบวนการพัฒนาซอฟต์แวร์และทำการทดลองที่ 6 ในภาคผนวก F แล้ว หัวข้อนี้จะเสริมความเข้าใจเรื่องของคำสั่งภาษาแอสเซมบลีและภาษาเครื่องเข้าด้วยกัน โดยจะใช้ภาษาเครื่องของ ARM เป็นกรณีศึกษาเพื่อให้สอดคล้องกับบอร์ด Pi3/Pi4 รูปที่ 3.12 แสดงตัวอย่างการแปลงจากคำสั่งแอสเซมบลี (ตัวอักษร) ทางด้านขวา ให้เป็นคำสั่งภาษาเครื่อง (ตัวเลขฐานสอง) ของ ARM เวอร์ชันความยาว 32 บิตทางด้านซ้าย ซึ่งมีรูปแบบที่ชัดเจน โดย ARM เป็นผู้ออกแบบและกำหนดรายละเอียด เพื่อให้ผู้พัฒนาสามารถเขียนซอฟต์แวร์และต่อวงจร ตามที่ต้องการ

Field Values											Assembly Code	
31:28 27:26 25 24:21 20 19:16 15:12 11:7 6:5 4 3:0											SUB R5, R7, R1	
<b>1110<sub>2</sub> 00<sub>2</sub> 0 2 0 7 5 0 0 0 1</b>												
cond op I cmd S Rn Rd shamt5 sh Rm												
31:28 27:26 25:20 19:16 15:12 11:0											LDR R9, [R4, #16]	
<b>1110<sub>2</sub> 01<sub>2</sub> 25 4 9 16</b>												
cond op I PUBLW Rn Rd imm12												

รูปที่ 3.12: ตัวอย่างการแปลงจากคำสั่งแอสเซมบลีของ ARM ทางด้านขวาเป็นคำสั่งภาษาเครื่องทางด้านซ้าย  
ที่มา: [Harris and Harris \(2013\)](#)

คำสั่ง **SUB R5, R7, R1** ความหมาย คือ  $R5 = R7 - R1$  จะถูกแปลงเป็นคำสั่งภาษาเครื่องความยาว 32 บิต เท่ากับ  $1110\ 00\ 0\ 0010\ 0\ 0111\ 0101\ 00000\ 00\ 0\ 0001_2$  หรือ  $E94E75001_{16}$  ผู้อ่านสามารถทำความเข้าใจบิตต่างๆ ตามลำดับจากซ้ายไปขวา ดังนี้

- บิตที่ 28-31 ความยาว 4 บิต คือ ตำแหน่งของ cond (Condition) ซึ่ง  $1110_2$  หมายถึง คำสั่ง SUB นี้ทำงานโดยไม่มีเงื่อนไข (Always) รายละเอียดเพิ่มเติมในหัวข้อที่ [4.7](#)
- บิตที่ 26-27 ความยาว 2 บิต คือ ตำแหน่งของ OpCode (Operation Code) ของคำสั่ง SUB มีค่าเท่ากับ  $00_2$
- บิตที่ 25 ความยาว 1 บิต คือ ตำแหน่งของ Imm (Immediate) ของคำสั่ง SUB มีค่าเท่ากับ  $0_2$  หมายความว่า บิตที่ 0-3 ของคำสั่งนี้เป็นหมายเลขรีจิสเตอร์
- บิตที่ 21-24 ความยาว 4 บิต คือตำแหน่งของ cmd (Command) ของคำสั่ง SUB มีค่าเท่ากับ  $2_{10}$  หรือ  $0010_2$
- บิตที่ 20 ความยาว 1 บิต คือ ตำแหน่งของ S (Set Condition Code) ของคำสั่งนี้มีค่าเท่ากับ  $0_2$  นั่นคือไม่มีตั้ง (Set) ค่า CPSR (Current Program Status Register) มีรายละเอียดเพิ่มเติมในหัวข้อที่ [4.7](#)
- บิตที่ 15-19 ความยาว 4 บิต คือ หมายเลขของรีจิสเตอร์ Rn มีค่าเท่ากับ  $7_{10}$  หรือ  $0111_2$  หมายถึง R7
- บิตที่ 12-15 ความยาว 4 บิต คือ หมายเลขของรีจิสเตอร์ Rd มีค่าเท่ากับ  $5_{10}$  หรือ  $0101_2$  หมายถึง R5
- บิตที่ 7-11 ความยาว 5 บิต คือ ตำแหน่งของ Shamt5 (5-bit Shift Amount) มีค่าเท่ากับ  $0_{10}$  หรือ  $00000_2$  หมายถึงจำนวนบิตที่ต้องการเลื่อนเท่ากับ 0 บิตนั้นคือ ไม่มีการเลื่อนบิตของรีจิสเตอร์ Rm

- บิตที่ **5-6** ความยาว 2 บิต คือ ตำแหน่งของ Sh (Shift Type) ของคำสั่งนี้มีค่าเท่ากับ  $00_2$  หมายความว่า เป็น การเลื่อนบิตทางซ้ายแบบตรรกะ (Left Logical Shift) รายละเอียดเพิ่มเติมในหัวข้อที่ [4.6.2](#)
- บิตที่ **4** ความยาว 1 บิต มีค่าเท่ากับ  $0_2$  เสมอ
- บิตที่ **0-3** ความยาว 4 บิต คือ หมายเลขของรีจิสเตอร์ **Rm** มีค่าเท่ากับ  $1_{10}$  หรือ  $0001_2$  หมายถึง **R1**

คำสั่ง **LDR R9, [R4, #16]** ความหมาย คือ  $R9 = \text{Mem}[R4+16_{10}]$  จะถูกแปลงเป็นคำสั่งภาษาเครื่อง ความยาว 32 บิตเท่ากับ  $1110\ 01\ 011001\ 0100\ 1001\ 0000\ 0001\ 0000_2$  หรือ  $E5949010_{16}$  ผู้อ่านสามารถ ทำความเข้าใจบิตต่างๆ ตามลำดับจากซ้ายไปขวา ดังนี้

- บิตที่ **28-31** ความยาว 4 บิต คือ ตำแหน่งของ cond (Condition) ซึ่ง  $1110_2$  หมายถึง คำสั่ง **LDR** นี้ ทำงานโดยไม่มีเงื่อนไข (Always) รายละเอียดเพิ่มเติมในหัวข้อที่ [4.7](#)
- บิตที่ **26-27** ความยาว 2 บิต คือ ตำแหน่งของ OpCode (Operation Code) ของคำสั่ง LDR มีค่าเท่ากับ  $01_2$
- บิตที่ **20-25** ความยาว 6 บิต คือ ตำแหน่งของ IPUBWL ของคำสั่ง LDR มีค่าเท่ากับ  $25_{10}$  หรือ  $011001_2$  รายละเอียดเพิ่มเติมที่ [osuosl.org](http://osuosl.org)
- บิตที่ **15-19** ความยาว 4 บิต คือ หมายเลขของรีจิสเตอร์ **Rn** มีค่าเท่ากับ  $4_{10}$  หรือ  $0100_2$  หมายถึง **R4**
- บิตที่ **12-15** ความยาว 4 บิต คือ หมายเลขของรีจิสเตอร์ **Rd** มีค่าเท่ากับ  $9_{10}$  หรือ  $1001_2$  หมายถึง **R9**
- บิตที่ **0-11** ความยาว 12 บิต คือ ตำแหน่งของ imm12 (12-bit Immediate) มีค่าเท่ากับ  $16_{10}$  หรือ  $0000\ 0001\ 0000_2$  หมายถึง **#16** ท้ายคำสั่งนี้

คำสั่งที่อ่าน (Fetch) จากหน่วยความจำเป็นตัวเลขฐานสองเท่านั้น (ภาษาเครื่อง) ซึ่งจะตีความหรือ ถอดรหัส (Decode) ได้ว่า คำสั่งที่อ่านเข้ามาเป็นคำสั่งอะไร ต้องการใช้รีจิสเตอร์ตัวไหน กำหนดค่าคงที่ (Immediate) เป็น เลขฐานสิบหรือ ฐานสอง แล้วจึงสั่งว่าควบคุมภายนอกให้�行 (Execute) ตาม รายละเอียด ในการออกแบบจรหรือฮาร์ดแวร์ได้ในวิชาสถาปัตยกรรมคอมพิวเตอร์ (Computer Architecture) ส่วนการ ออกแบบพัฒนาวงจรนั้นสามารถทำได้โดยใช้ภาษา Hardware Description Language (HDL) ซึ่งมีลักษณะ คล้ายกับภาษาโปรแกรมคอมพิวเตอร์ระดับสูง C/C++ และ Java ซึ่งได้รับความนิยม 2 ภาษาคือ

- ภาษา VHDL ([Very High Speed Integrated Circuit HDL](#)) และ
- ภาษา Verilog HDL ([Verilog Hardware Description Language](#)) เพื่อให้โปรแกรมลักษณะคล้ายคอม ไฟเลอร์แต่เรียกว่า Synthesis Tool สังเคราะห์เป็นวงจรบนอุปกรณ์
- FPGA ([Field Programmable Gate Array](#)) โดยเป็นวงจรชนิดหนึ่งซึ่งผู้ใช้สามารถเปลี่ยนหรือโปรแกรม (Program) ให้ทำงานเป็นวงจรได้หลายๆ รอบ เพื่อทดสอบการทำงานจนถูกต้อง

### 3.3 การทำงานร่วมกันระหว่างฮาร์ดแวร์และซอฟต์แวร์ (Hardware-Software Operation)

การทำงานของคอมพิวเตอร์จะเริ่มต้นขึ้นเมื่อผู้ใช้กดปุ่มเปิดเครื่องเพื่อจ่ายไฟเลี้ยงให้กับคอมพิวเตอร์ ระบบฮาร์ดแวร์จะเริ่มทำงานเพื่อตรวจสอบและเตรียมความพร้อม หลังจากนั้น ขั้นตอนการทำงานของซอฟต์แวร์ระบบและซอฟต์แวร์ประยุกต์สามารถสรุปได้ดังนี้

1. การบูต (Boot) ระบบปฏิบัติการจากอุปกรณ์เก็บรักษาข้อมูลเข้าสู่หน่วยความจำหลัก ในหัวข้อที่ [3.3.1](#)
2. ระบบปฏิบัติการโหลดไฟล์ซอฟต์แวร์แอปพลิเคชันจากอุปกรณ์เก็บรักษาข้อมูลเข้าสู่หน่วยความจำหลัก ในหัวข้อที่ [3.3.2](#)
3. ซีพียูคำสั่งภาษาเครื่องจากหน่วยความจำหลักไปปฏิบัติตาม ในหัวข้อที่ [3.3.3](#)
4. ซีพียูอ่าน/เขียนข้อมูลระหว่างหน่วยความจำหลักไปประมวลผล ในหัวข้อที่ [3.3.4](#)
5. ซีพียูเข้มต่ออุปกรณ์อินพุตต่างๆ เช่น คีย์บอร์ด เม้าส์ เครื่อข่ายอินเทอร์เน็ต เป็นต้น ในหัวข้อที่ [3.3.5](#)
6. ซีพียูอ่าน/เขียนไฟล์ข้อมูลระหว่างหน่วยความจำหลักและอุปกรณ์เก็บรักษาข้อมูล ในหัวข้อที่ [3.3.6](#)
7. ผู้ใช้ตัดงาน (Shutdown) ระบบปฏิบัติการก่อนปิดเครื่อง ในหัวข้อที่ [3.3.7](#)

ขั้นตอนดังกล่าวจะแตกต่างกันออกไปตามรายละเอียดของฮาร์ดแวร์และระบบปฏิบัติการที่ใช้ แต่หลักการโดยรวมจะมีความคล้ายคลึงกัน การทดลองที่ 3 ภาคผนวก C จะแนะนำการติดตั้งระบบปฏิบัติการซึ่งว่า Raspberry Pi OS การทดลองที่ 4 ภาคผนวก D จะแนะนำการใช้งานพัฒนาชั้นพื้นฐานในระบบลินุกซ์และยูนิกซ์ (Unix) เป็นต้น ซึ่งยูนิกซ์เป็นระบบปฏิบัติการที่มีประวัติศาสตร์ยาวนานและเป็นต้นแบบของระบบปฏิบัติการต่างๆ ในวิชาการคอมพิวเตอร์ และการทดลองที่ 5 จะแนะนำการพัฒนาโปรแกรมด้วยภาษา C สำหรับยูนิกซ์

#### 3.3.1 การบูต (Boot) ระบบปฏิบัติการ

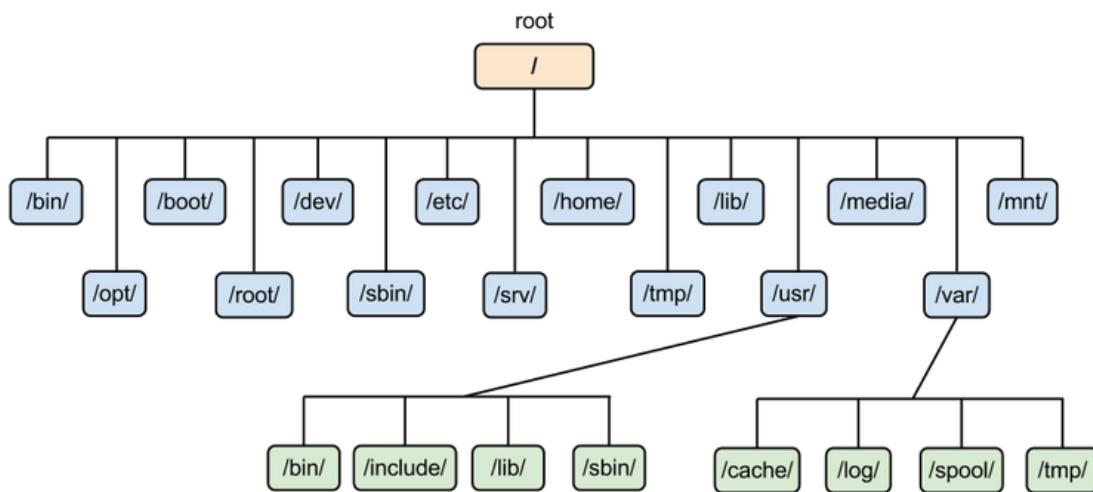
การบูตระบบปฏิบัติการ คือ การรันระบบปฏิบัติการเพื่อครอบครองฮาร์ดแวร์ทั้งหมดของเครื่องคอมพิวเตอร์ ขั้นตอนและรายละเอียดการบูตจะแตกต่างกันไปตามเงื่อนไขของฮาร์ดแวร์และระบบปฏิบัติการ การบูตระบบปฏิบัติการ Raspberry Pi OS บนบอร์ด Pi3/Pi4 มีขั้นตอนโดยละเอียดตามลำดับดังนี้

1. เมื่อเปิดเครื่อง หรือ จ่ายไฟให้บอร์ด Pi3/Pi4 ซีพียู ARM Cortex A53/A72 จะยังไม่ทำงาน แต่จะใช้แกนประมวลผลขนาดเล็กภายในจีพียูอ่านคำสั่งที่บรรจุใน On-Chip รวม เรียกว่า บูตโลดเดอร์ (Boot Loader) ซึ่งฝังอยู่ภายในชิป BCM2837/BCM2711
2. จีพียูสั่งงานฮาร์ดแวร์ที่ควบคุมอุปกรณ์เก็บรักษาข้อมูลการ์ดหน่วยความจำชนิดไมโคร SD เพื่อมองหา파ร์ทิชันชื่อ boot ที่ฟอร์แมทด้วยรูปแบบ FAT32 ภายในการดันน์
3. เมื่อเจอพาร์ทิชันชื่อ boot เล็วจีพียูอ่านชุดคำสั่งจากไฟล์ชื่อ bootcode.bin ซึ่งทำหน้าที่เป็นบูตโลดเดอร์ โดยในระหว่างนี้จะยังไม่มีการใช้งานหน่วยความจำหลัก SDRAM บนบอร์ด Pi3/Pi4
4. จีพียูเริ่มต้นอ่านไฟล์ start.elf จากพาร์ทิชัน boot ในการ์ดหน่วยความจำ SD ไปบรรจุในหน่วยความจำหลักเพื่อให้จีพียูเริ่มทำงาน โดยอาศัยไฟล์ fixup.dat ซึ่งมีข้อมูลการจัดแบ่งพาร์ทิชันภายในระหว่างจีพียูและซีพียู

5. ชีพิญเริ่มต้นทำงานตามคำสั่งภาษาเครื่องหน่วยความจำหลัก (SDRAM) ที่โหลดจากไฟล์ start.elf ก่อน เพื่ออ่านไฟล์ชื่อ kernel.img ไปบรรจุใน SDRAM ตามรายละเอียดในไฟล์ config.txt เพื่อติดตั้งค่าของระบบ

6. ชีพิญเริ่มต้นเฟทธ์และปฏิบัติตามคำสั่งโปรแกรมเครอร์เนลใน SDRAM ด้วยพารามิเตอร์ที่บรรจุอยู่ใน cmdline.txt

**เมื่อบูตระบบสำเร็จ** เครอร์เนลหรือโปรแกรมหลักของระบบปฏิบัติการจะครอบคลุมฮาร์ดแวร์ทั้งหมด และทำงานร่วมกับไฟล์ซอฟต์แวร์และไฟล์ข้อมูลต่างๆ ภายในการดหน่วยความจำ SD ซึ่งจะแบ่งโครงสร้างการจัดเก็บไฟล์ต่างๆ ตามระบบปฏิบัติการลินุกซ์ เรียกว่า **ไดเรกทอรี** ในรูปที่ 3.13 ผู้อ่านสามารถทำความเข้าใจไดเรกทอรีตามลำดับความสำคัญ ดังนี้

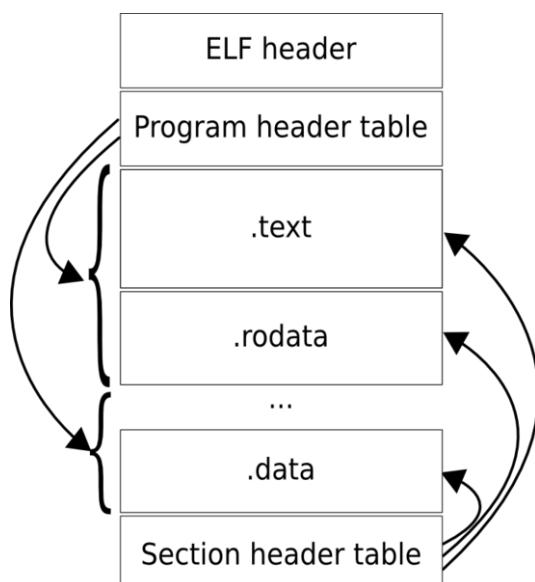


รูปที่ 3.13: โครงสร้างของไดเรกทอรี (Directory) สำคัญๆ ในระบบปฏิบัติการลินุกซ์ ที่มา: [freedomopen-guin.com](http://freedomopen-guin.com)

1. /root: เป็นไดเรกทอรีหลักของระบบ เรียกว่า **รูทไดเรกทอรี**
2. /bin: เป็นไดเรกทอรีบรรจุไฟล์ซอฟต์แวร์ระบบ เช่น คำสั่ง (Command) ต่างๆ สำหรับผู้ใช้ส่วนใหญ่
3. /sbin: เป็นไดเรกทอรีบรรจุไฟล์คำสั่ง (Command) ต่างๆ ของระบบ เพื่อให้ผู้ใช้งาน ชื่อ **root** ทำหน้าที่บริหารระบบ (Administrator) เท่านั้น
4. /lib: เป็นไดเรกทอรีสำหรับเก็บไฟล์ไลบรารี (Library) ต่างๆ ของระบบ
5. /home: เป็นไดเรกทอรีสำหรับพื้นที่ส่วนตัวของผู้ใช้แต่ละคน สำหรับเก็บไฟล์ต่างๆ โดยใช้ชื่อไดเรกทอรีตามชื่อผู้ใช้ ภายใต้ไดเรกทอรีผู้ใช้แต่ละคนแบ่งเป็นพื้นที่หรือไดเรกทอรีต่างๆ เช่น Documents Desktop Downloads เพื่อใช้เป็นที่จัดเก็บไดเรกทอรีและไฟล์ส่วนตัวของผู้ใช้แต่ละคน ซึ่งจะแยกตามชื่อ ล็อกอิน เช่น /home/user1 /home/user2 เป็นต้น โดย user1 และ user2 คือ ชื่อล็อกอิน ตามลำดับ
6. /dev: เป็นไดเรกทอรีที่ถูกสร้างใหม่ทุกครั้งที่บูตเครื่อง เพื่อกีบไฟล์ที่เป็นตัวแทนของอุปกรณ์ (Devices) อินพุต/เอาต์พุตต่างๆ ในยูสเซอร์สเปช (User Space) รายละเอียดเพิ่มเติมสามารถอ่านได้จากคำสั่ง ls /dev ในการทดลองที่ 9 ภาคผนวก | หัวข้อที่ L.3
7. /sys: เป็นระบบไฟล์เสมือน (Virtual file system) เพื่อให้โปรแกรมเมอร์เข้าถึงอุปกรณ์ฮาร์ดแวร์ เช่น ขา GPIO ต่างๆ ซึ่งจะกล่าวในกราฟิกทดลองที่ 9 ภาคผนวก | หัวข้อที่ L.3

8. /proc: เป็นเวอร์ชัลไดเรกทอรี สำหรับแต่ละprocในระบบ ยกตัวอย่างเช่น /proc การทดลองที่ 4 ภาคผนวก D
9. /etc: เป็นไดเรกทอรีใช้เก็บไฟล์สำหรับบรรจุไฟล์คอนฟิก (Configuration File) ของระบบ และแอปพลิเคชันต่างๆ
10. /var: เป็นไดเรกทอรีสำหรับเก็บล็อก (Log) และข้อมูลอื่นๆ ที่เกิดขึ้นระหว่างการรันระบบ
11. /mnt: เป็นจุดมาท์ (Mount) ชั่วคราว เช่น การแชร์ไฟล์ผ่านเครือข่าย
12. /media: เป็นจุดมาท์หลักสำหรับสื่อเคลื่อนที่ (Removable device) ต่างๆ เช่น แฟลชไดร์ฟ ซีดี/ดีวีดี รวม เป็นต้น
13. /opt: เป็นไดเรกทอรีสำหรับติดตั้งแอปพลิเคชันจากผู้ขาย
14. /run: เป็นไดเรกทอรีชั่วคราวสำหรับแอปพลิเคชันในระหว่างที่รันอยู่
15. /tmp: เป็นไดเรกทอรีใช้สำหรับบรรจุไฟล์ชั่วคราวจากแอปพลิเคชันและระบบ

### 3.3.2 ระบบปฏิบัติการโหลดซอฟต์แวร์ประยุกต์จากไฟล์รูปแบบ ELF



รูปที่ 3.14: โครงสร้างไฟล์ชนิด ELF สำหรับเก็บชุดคำสั่งภาษาเครื่องและข้อมูลเป็นเลขฐานสองอยู่ในอุปกรณ์เก็บรักษาข้อมูล ที่มา: [wikipedia.org](https://wikipedia.org)

ซอฟต์แวร์ หรือ โปรแกรม คือ ไฟล์รูปแบบ ELF (Executable and Linkable Format) ทำหน้าที่เก็บคำสั่งภาษาเครื่อง (Machine Code) และข้อมูลเลขฐานสองประกอบการทำงาน ตามโครงสร้างที่ระบบปฏิบัติการยุนิกซ์กำหนดรูปที่ 3.14 ระบบปฏิบัติการไมโครซอฟต์วินโดวส์กำหนดรูปแบบไฟล์โปรแกรมประยุกต์ที่คล้ายคลึงกัน เรียกว่า รูปแบบ EXE (Executable)

ผู้ใช้เครื่องคอมพิวเตอร์ได้รับสิทธิ์ (Permission) ติดตั้ง (Install) ไฟล์ซอฟต์แวร์เหล่านี้ลงในอุปกรณ์เก็บรักษาข้อมูลตามโครงสร้างของไดเรกทอรีต่างๆ ในหัวข้อที่ 3.3.1 โดยการทดลองที่ 4 ภาคผนวก D จะแนะนำการใช้งาน

คำสั่ง (Command) ต่างๆ และโครงสร้างของระบบปฏิบัติยูนิกซ์และลินุกซ์ ตำราเล่มนี้จะเน้นที่ระบบปฏิบัติการ Raspberry Pi OS เป็นลินุกซ์เวอร์ชันหนึ่งซึ่งพัฒนาสำหรับบอร์ดทดลองลินุกซ์โดยเฉพาะ

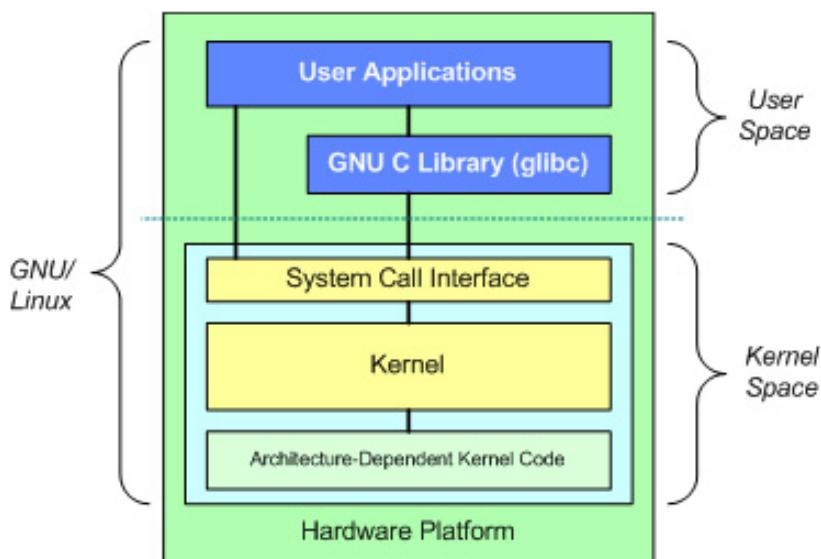
รูปแบบไฟล์ ELF ย่อมาจาก Executable and Linkable Format เป็นชนิดและโครงสร้างของไฟล์โปรแกรม (Executable) และไฟล์อีบเจกต์ (Object) ที่ได้จากการคอมไพล์ ไฟล์รูปแบบ ELF มีความยืดหยุ่น สามารถขยายเพิ่มเติมได้ง่าย และรองรับการข้ามแพลทฟอร์ม (Cross Platform) ระหว่างซีพียู และชุดคำสั่ง (Instruction Set) หรือภาษาเครื่อง เป็นเลขฐานสอง โครงสร้างของไฟล์ ELF ตามรูปที่ 3.14 ประกอบด้วยพื้นที่หลายส่วนและเซกเมนต์ (Segment) ต่างๆ คือ

- ส่วนเริ่มต้นไฟล์ (ELF Header) บ่งชี้ว่าจะอ้างถึงหน่วยความจำหลักด้วยเลขแอดเดรสขนาด 32 บิต หรือ 64 บิต ตามด้วยข้อมูลอื่นๆ ตามลำดับถัดไป แบ่งเป็น
  - หากใช้ระบบ 32 บิต ส่วนหัวนี้จะมีความยาว 52 ไบต์
  - หากใช้ระบบ 64 บิต ส่วนหัวนี้จะมีความยาว 64 ไบต์
- ตารางโปรแกรม (Program Header Table) อธิบายเซกเมนต์ (Segment) ต่างๆ ในไฟล์ ELF ว่าจะสร้างเซกเมนต์เหล่านั้นอย่างไร ที่ตำแหน่งใดในหน่วยความจำ
- เท็กซ์เซกเมนต์ (.text) ใช้สำหรับเก็บคำสั่ง (เป็นเลขฐานสอง)
- ดาต้าเซกเมนต์ชนิดอ่านอย่างเดียว (.rodata: Read-Only Data) ใช้สำหรับเก็บค่าคงตัวแปรหรือข้อมูลที่เป็นค่าคงที่ (เป็นเลขฐานสอง)
- ดาต้าเซกเมนต์ (.data และ .data1) ใช้สำหรับเก็บตัวแปรที่มีค่าตั้งต้น (Initialized Data) และอาจเปลี่ยนแปลงได้เมื่อทำงาน
- ตาราง Section Header Table อธิบายเซกชัน (Section) ต่างๆ ในหน่วยความจำ ว่าซื้ออะไร มีความยาวเท่าไร

ผู้อ่านสามารถสืบค้นรายละเอียดและข้อมูลเพิ่มเติมเกี่ยวกับโครงสร้างของไฟล์ ELF เพื่อจัดเก็บคำสั่งภาษาเครื่อง (Machine Instruction) และข้อมูลต่างๆ ได้ทางลิงก์ต่อไปนี้ [wikipedia.org](https://en.wikipedia.org)

เมื่อเครื่องคอมพิวเตอร์พร้อมใช้งาน หรือบูตระบบปฏิบัติการลินุกซ์สำเร็จแล้ว ผู้ใช้จึงสามารถเรียกใช้แอปพลิเคชันซอฟต์แวร์หรือโปรแกรมตามสิทธิ์ (Permission) ด้วยการกดตัวเบลคลิกที่ไอคอน (Icon) หรือเรียกชื่อไฟล์แอปพลิเคชันนั้นผ่านเทอร์มินัล (Terminal) ในการทดลองที่ 4 ภาคผนวก D ระบบปฏิบัติการจึงอ่านหรือโหลดไฟล์แอปพลิเคชันนั้นๆ จากอุปกรณ์เก็บรักษาข้อมูลไปบรรจุในหน่วยความจำหลัก เราเรียกขั้นตอนนี้ว่า การโหลดโปรแกรม (Program Loading) เกิดเป็นโครงสร้างที่สมบูรณ์ทั้งหมดในรูปที่ 3.15 ประกอบด้วย ฮาร์ดแวร์และหน่วยความจำเวอร์ชวลเมโมรี (Virtual memory) รายละเอียดในหัวข้อที่ 5.2 โดยหน่วยความจำเวอร์ชวลเมโมรีของหนึ่งโปรแกรมในระบบปฏิบัติการลินุกซ์แบ่งเป็นพื้นที่ 2 ส่วน เรียกว่า เคอร์เนลสเปซ และ ယูสเซอร์สเปซ ดังนี้

- เคอร์เนลสเปซ (Kernel Space) เป็นพื้นที่ในหน่วยความจำที่จัดสรรโดยเฉพาะสำหรับເຄອർນېල และองค์ประกอบสำคัญอื่นๆ โครงสร้างภายในของหน่วยความจำบริเวณເຄອർเนลสเปซ แบบ Top-Down ประกอบด้วย



รูปที่ 3.15: โครงสร้างของคอมพิวเตอร์ประกอบด้วยฮาร์ดแวร์ชั้นล่างสุด และหน่วยความจำเวอร์ชัลเม莫เรี (Virtual memory) ภายใต้ระบบลินุกซ์แบ่งเป็นพื้นที่ เรียกว่า เคอร์เนลสเปช (ระบบปฏิบัติการ) และ ยูสเซอร์สเปช (ซอฟต์แวร์ประยุกต์) ที่มา: [linux-india.org](http://linux-india.org)

- เคอร์เนล (Kernel) คือ โปรแกรมหลักภายในระบบปฏิบัติการ ซึ่งปัจจุบันได้มีการพัฒนาอย่างต่อเนื่อง ผู้อ่านสามารถตรวจสอบเวอร์ชันล่าสุดได้ที่ [www.kernel.org](http://www.kernel.org)
- ชิสเต็มคลอลินเทอร์เฟช (System Call Interface) เป็นช่องทางให้ซอฟต์แวร์ประยุกต์เรียกใช้งาน เคอร์เนล เพื่อให้เคอร์เนลช่วยสิ่งงานอุปกรณ์อินพุต/เอ้าต์พุต การอ่าน/เขียนไฟล์ เป็นต้น
- คำสั่งภาษาเครื่องภายในเคอร์เนลที่ผูกติดกับสถาปัตยกรรมของฮาร์ดแวร์ (Architecture Dependent Kernel Code) สำหรับใช้งานเชื่อมต่อ กับฮาร์ดแวร์ต่างๆ โดยเฉพาะซึ่งคำสั่งภาษาเครื่องเหล่านี้ของเคอร์เนลจะเปลี่ยนแปลงไปตามชิปปิค์และชิปที่ทำงาน เพื่อให้เคอร์เนลติดต่อกับทรัพยากรเหล่านั้นได้อย่างถูกต้องและมีประสิทธิภาพ คำสั่งและข้อมูลประกอบนี้ เรียกว่า ดีไวซ์ไดรเวอร์
- ยูสเซอร์สเปช (User Space) เป็นพื้นที่หน่วยความจำที่จัดสรรให้กับโปรแกรมประยุกต์แต่ละโปรแกรมเท่านั้น ประกอบด้วย
  - คำสั่งภาษาเครื่องและข้อมูลจากโปรแกรมประยุกต์ที่โหลดจากไฟล์ ELF ที่อธิบายในหัวข้อนี้ และ
  - คำสั่งภาษาเครื่องและข้อมูลภายในไลบรารีสำคัญ ได้แก่ glibc (GNU C Library) เป็นต้น ซึ่งจะช่วยประหยัดเวลาในการพัฒนาฟังก์ชันพื้นฐาน

ชิปปิค์จะทำงานตามคำสั่งภาษาเครื่องของซอฟต์แวร์ประยุกต์และไลบรารีของผู้ใช้ที่ได้ล็อกอินเข้าสู่ระบบปฏิบัติการ เช่น โปรแกรมเบราว์เซอร์สามารถเชื่อมต่อกับเครือข่ายและทรัพยากรอื่นๆ ผ่านทางชิสเต็มคลอลินเทอร์เฟช ซอฟต์แวร์ประยุกต์ต่างๆ ได้แก่ โปรแกรมที่ใช้งานประเภทต่างๆ เช่น เว็บเบราว์เซอร์ (Web Browser) เช่น โมซิล่าไฟร์ฟ็อกซ์ (Mozilla Firefox) เกมต่างๆ เป็นต้น โปรแกรมมอร์สามารถพัฒนาซอฟต์แวร์ประยุกต์เหล่านี้แยกในรูปของฟรีแวร์ (Freeware) และฟรีแวร์บางตัวมีการเปิดเผยชอร์สโค้ด (Source Code) เรียกว่า โอเพนซอร์ส (Open Source) เพื่อให้โปรแกรมเมอร์ต่างๆ ทั่วโลกสามารถพัฒนาเสริมเพิ่มเติมได้ตามเงื่อนไขของการเปิดเผยชอร์สโค้ด เช่น GPL (GNU General Profile License), LGPL (Lesser GPL) เป็นต้น

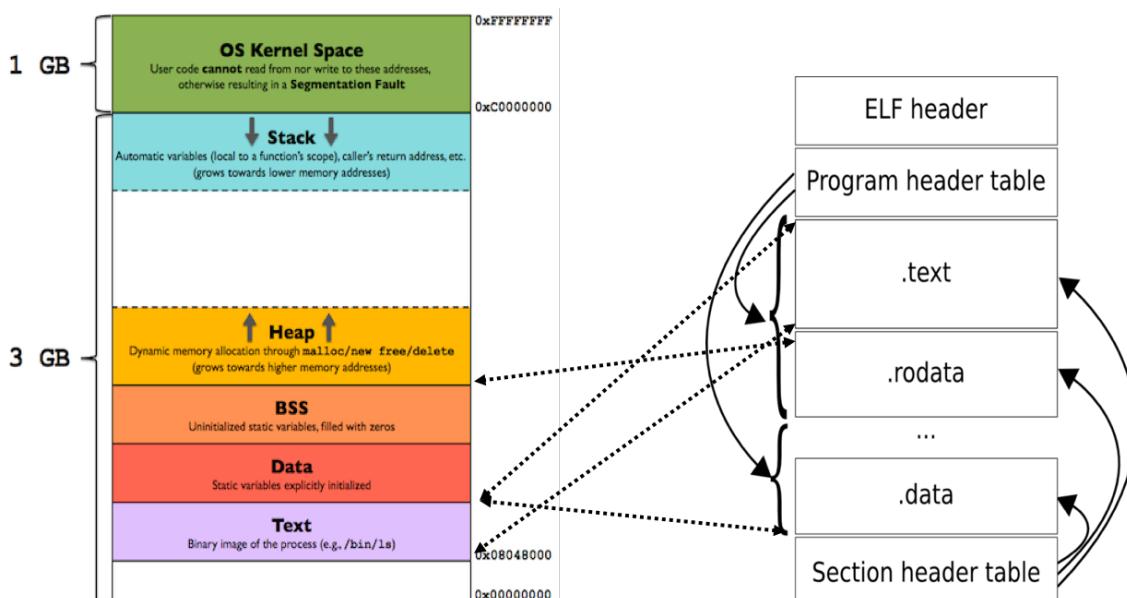
รายละเอียดเพิ่มเติมสามารถศึกษาและทดลองได้จากการทดลองที่ 4 ภาคผนวก D และการทดลองอื่นตามลำดับ

### 3.3.3 ชีพียุทธ์คำสั่งภาษาเครื่องของซอฟต์แวร์ประยุกต์จากหน่วยความจำ

หน่วยความจำ (เวอร์ชวลเม莫รี) ของแอปพลิเคชันบริเวณยูสเซอร์สเปซจะแบ่งเป็นเซ็กเมนต์ต่างๆ คล้ายกับโครงสร้างของไฟล์ ELF ในรูปที่ 3.16 ตามรายละเอียดดังนี้ ระบบปฏิบัติการจะแมป (Map) หรือจัดวางเวอร์ชวลเม莫รีบริเวณที่เรียกว่า

- เทิกซ์เซ็กเมนต์ เพื่อบรรจุคำสั่งภาษาเครื่องที่อ่านมาจากไฟล์แอปพลิเคชัน และ
- ดาต้าเซ็กเมนต์ เพื่อบรรจุข้อมูลเลขฐานสองที่อ่านมาจากไฟล์แอปพลิเคชัน

ผู้อ่านสามารถศึกษารายละเอียดเกี่ยวกับการแมปนี้เพิ่มเติมในหัวข้อที่ 5.2 หลังจากนั้น ระบบปฏิบัติการจะเปิดโอกาสให้แอปพลิเคชันนั้น ทำงานตามสิทธิ์ (Permission) ที่ได้รับ โดยชีพียุทธ์เริ่มต้นเฟทช์ (Fetch) หรืออ่านคำสั่งภาษาเครื่องจากฟังก์ชัน ชื่อ main เสมอ ซึ่งบรรจุอยู่ในพื้นที่ เรียกว่า เทิกซ์เซ็กเมนต์



รูปที่ 3.16: การจัดวางคำสั่งภาษาเครื่องในเทิกซ์เซ็กเมนต์และข้อมูลในดาต้าเซ็กเมนต์จากไฟล์โปรแกรมรูปแบบ ELF ไปยังเวอร์ชวลเม莫รี บริเวณยูสเซอร์สเปซ (User Space) ที่มา: [wordpress.com](http://wordpress.com)

หลักการนี้จะครอบคลุมระบบปฏิบัติการ 32 และ 64 บิต แต่ต่อมาเล่นนี้จะครอบคลุมเนื้อหาของระบบปฏิบัติการขนาด 32 บิตเท่านั้น ทำให้ระบบสร้างหน่วยความจำ (เวอร์ชวลเม莫รี) ขนาด  $2^{32}$  หรือ 4 กิกะไบต์ (GiB) เท่านั้น เคอร์เนลเองเป็นโปรแกรมตัวหนึ่งที่สามารถจับจองและจัดสรรพื้นที่เวอร์ชวลเม莫รีขนาด 1 กิกะไบต์ (GiB) เรียกว่า **ເຄືອງລສເປີ່ງ (Kernel Space)** ตั้งแต่หมายเลข 0xC0000000 หรือ 0x0000 0000 ถึง 0xFFFF FFFF ส่วนพื้นที่ຢູ່ສເຊອ້ຣ໌ສເປີ່ງ (User Space) ของเวอร์ชວලເມໂມຣີຂາດ 3 ກົບໄບຕໍ່ (GiB) ຈະເຮີ່ມຕົ້ນທີ່ໜາຍເລີກ 0x0000 0000 ປຶ້ງ 0xBFFF FFFF ຕາມຕ້ວຍໆຢ່າງ ແບ່ງເປັນເຊັກມັດຕ່າງໆ ດັ່ງນີ້

- เทิกซ์ເຊັກມັດຕ່າງໆ ຕາມທີ່ກ່າວໄປແລ້ວໃນຫ້ວັນທີ 3.3.2
- ดาຕ້າເຊັກມັດຕ່າງໆ ຕາມທີ່ກ່າວໄປແລ້ວໃນຫ້ວັນທີ 3.3.2

- **BSS เช็คเมนต์** (“Block Started by Symbol”) คือ พื้นที่ส่วนหนึ่งในหน่วยความจำ สำหรับจัดเก็บค่าของตัวแปรชนิด global และ static ที่โปรแกรมเมอร์ยังไม่ได้ตั้งค่าเริ่มต้น แต่ระบบปฏิบัติการจะทำการตั้งค่าเริ่มต้นให้กulary เป็น 0 โดยอัตโนมัติ ตำแหน่งของ BSS เช็คเมนต์จะจัดเรียงต่อเนื่องจากตำแหน่งสุดท้ายของดาต้าเช็คเมนต์
- **สแต็กเช็คเมนต์** (Stack Segment) คือ พื้นที่ส่วนหนึ่งในหน่วยความจำ
  - สำหรับเก็บค่าของตัวแปรชนิดโลคอล (Local Variable) ในภาษา C/C++
  - สำหรับส่งผ่าน (Pass) ค่าตัวแปร เรียกว่า พิงก์ชันพารามิเตอร์ (Function Parameter) โดยอาศัยพื้นที่ของสแต็คเก็บค่าตัวแปรที่ต้องการส่งไปยังฟิงก์ชันผู้ถูกเรียก (Callee Function) รายละเอียดเพิ่มเติมในหัวข้อที่ [4.8](#) และการทดลองที่ 7 ภาคผนวก G

ฟิงก์ชันแต่ละฟิงก์ชันจะจองพื้นที่ภายในสแต็กเช็คเมนต์สำหรับใช้งานของตนเอง เรียกว่า **สแต็กเฟรม** (Stack Frame) โดยอาศัยรีจิสเตอร์ R13 หรือ **สแต็กพอยน์เตอร์** (Stack Pointer) เก็บค่าแอดเดรสชันบนสุดของสแต็ค (Top of Stack) เช็คเมนต์ เพื่อความสะดวกในการบริหารจัดการสแต็ค เช่น การจองและคืนพื้นที่ของสแต็กเฟรม เป็นต้น

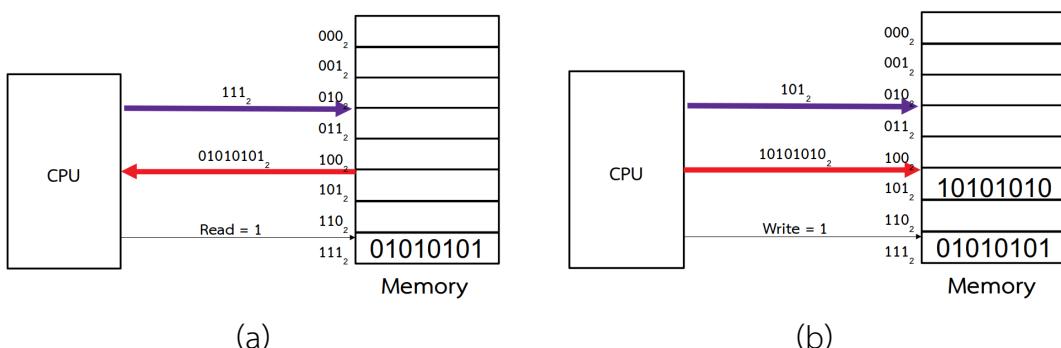
การบริหารสแต็กเฟรมของแต่ละฟิงก์ชันเหมือนการวางแผนของช้อนทับกัน ทำให้การบริหารจัดการมีลักษณะ **LIFO** (Last In First Out) นั่นคือ ก่อนเริ่มต้นรันแอปพลิเคชันสแต็กเช็คเมนต์ของโปรแกรมนี้ยังว่างเปล่า ตำแหน่งสแต็กพอยน์เตอร์ (Stack Pointer) คือ ตำแหน่งบนสุดของสแต็ค เมื่อโปรแกรมทำงานและเรียกใช้ฟิงก์ชัน โปรแกรมจะจองพื้นที่ในบริเวณสแต็กเช็คเมนต์เพื่อเก็บค่าของตัวแปรชนิดโลคอลและอื่นๆ เรียกว่า **สแต็กเฟรม** ณ ตำแหน่งสแต็กพอยน์เตอร์

เมื่อฟิงก์ชันทำงานเสร็จสิ้น โปรแกรมเมอร์จะต้องรีเทิร์นออกจากฟิงก์ชัน โดยคืนพื้นที่ (Pop) สแต็กเฟรม เพื่อให้สแต็กว่างลง ทำให้สแต็กพอยน์เตอร์เก็บแอดเดรสตำแหน่งที่ต่ำลงมา รวมกับว่ามีการนำของที่วางช้อนออกไป ผู้อ่านสามารถศึกษาคำสั่งภาษาแอสเซมบลีที่เกี่ยวข้องกับสแต็คในหัวข้อที่ [4.5](#) และการใช้งานสแต็คโดยละเอียดในการทดลองที่ 8 ภาคผนวก H

โปรแกรมสามารถเรียกฟิงก์ชันภายในฟิงก์ชัน (Nested Function) สแต็กเฟรมจะถูกวางเรียงช้อนทับต่อกันไป โดยเฉพาะการเรียกฟิงก์ชันแบบรีเครอฟ์สีฟ (Recursive Function) หรือแบบเรียกตัวเอง สแต็กเฟรมจะเรียงช้อนทับต่อกันไปเรื่อยๆ จนกว่าฟิงก์ชันจะรีเทิร์นกลับ หากไม่มีการรีเทิร์นกลับ พื้นที่ของสแต็กเช็คเมนต์จะโตขึ้นจนถึงขีดจำกัด เรียกว่า **RLIMIT\_STACK** หรือจอนสแต็กพอยน์เตอร์ช้อนทับกับตำแหน่งของฮีปเช็คเมนต์ และทำให้เกิด Exception เพื่อให้ระบบปฏิบัติการมาบริหารจัดการต่อไป ซึ่งอยู่นอกเนื้อหาของวิชานี้ ผู้อ่านสามารถค้นคว้ารายละเอียดเพิ่มเติมที่ [arm.com](http://arm.com)

- **ฮีป เช็คเมนต์** (Heap Segment) คือ พื้นที่ว่างภายในบูตเซอร์ สเปช ที่ระบบปฏิบัติการอนุญาตให้โปรแกรมเมอร์จองพื้นที่เพื่อเก็บค่าตัวแปรชนิดต่างๆ ตามความต้องการ ในขณะที่โปรแกรมรันอยู่ (Dynamic Allocation) เช่น ลิงก์ลิสต์ (Linked List) และโครงสร้างข้อมูลชนิดต่างๆ โดยใช้ฟิงก์ชัน malloc ในภาษา C และฟิงก์ชัน new ในภาษา C++ โปรแกรมเมอร์จะต้องคืนหน่วยความจำที่จองในฮีปเช็คเมนต์นี้โดยใช้ฟิงก์ชัน free ผู้อ่านสามารถศึกษาตัวอย่างการใช้งานฟิงก์ชันในภาษา C และโ้อเปอเรเตอร์ (Operator) delete ผู้อ่านสามารถศึกษาตัวอย่างการใช้งานฟิงก์ชันในภาษา C++ การจองและคืนพื้นที่นี้ จะทำให้ขนาดของฮีปเช็คเมนต์ลดหรือขยายขนาดเพิ่มเติมได้จนถึงตำแหน่งบนสุดของสแต็กพอยน์เตอร์ (Stack Pointer) โปรดสังเกตทิศทางหัวลูกศรการขยายตัวของสแต็กเช็คเมนต์และฮีปเช็คเมนต์ในรูปที่ [3.16](#) ว่ามีทิศทางพุ่งเข้าหากัน

### 3.3.4 ชีพิญอ่าน (Load)/เขียน (Store) ข้อมูลในหน่วยความจำหลัก



รูปที่ 3.17: (a) ขบวนการอ่าน (Load) ข้อมูลจากหน่วยความจำหลักที่ตำแหน่ง  $111_2$  (b) ขบวนการเขียน (Store) ข้อมูลในหน่วยความจำหลักที่ตำแหน่ง  $101_2$

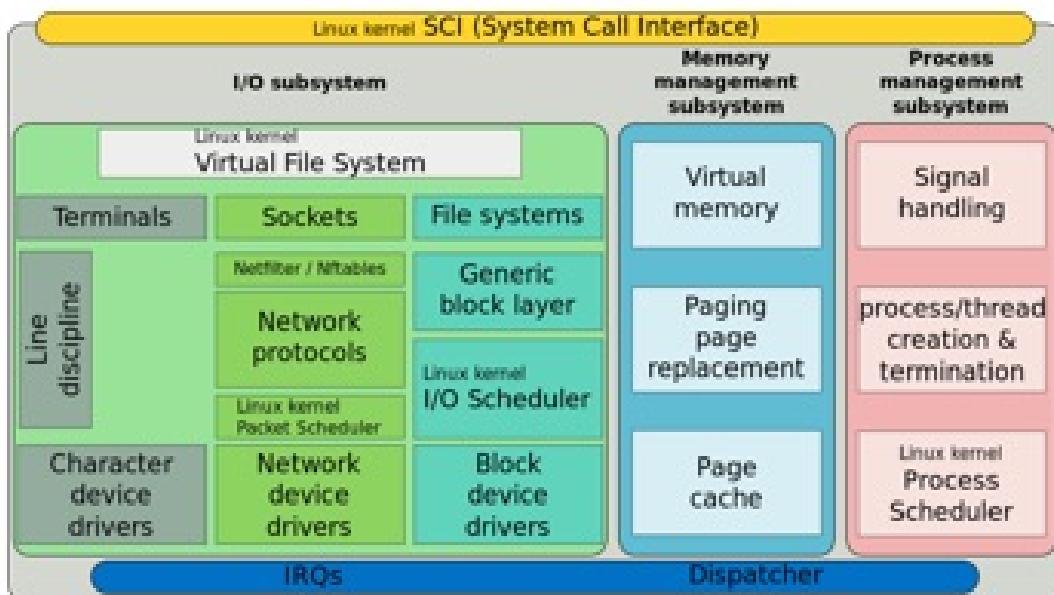
ตัวแปรต่างๆ หรือ ข้อมูลชนิดค่าคงที่ เช่น ค่า  $\pi$  อาศัยพื้นที่ในหน่วยความจำ บริเวณเดาต้าเซกเมนต์และ BSS เซกเมนต์ เพื่อบรรจุค่าตามจำนวนบิตและชนิดของข้อมูลนั้นในตารางที่ 2.1 เมื่อต้องการนำค่าตัวแปรเหล่านี้ไปประมวลผลใดๆ ก็ตาม โปรแกรมจะต้องส่งให้ชีพิญดำเนินการดังนี้ ชีพิญต้องอ่านค่า (Load) หรืออ่านตัวแปรจากหน่วยความจำหลักไปพักในรีจิสเตอร์ต้นทาง (Source Register) หลังจากนั้น ชีพิญจึงสามารถนำค่าในรีจิสเตอร์ต้นทางนั้นไปประมวลผลแล้วเก็บค่าผลลัพธ์ในรีจิสเตอร์ปลายทาง (Destination Register) เมื่อเสร็จสิ้นภารกิจ ชีพิญต้องนำค่าผลลัพธ์ในรีจิสเตอร์ปลายทางเก็บ (Store) หรือเขียนลงในหน่วยความจำหลัก ณ ตำแหน่งหรือแอดдресของตัวแปรนั้นๆ เราเรียก ชีพิญที่ทำงานในลักษณะนี้ว่า ชีพิญสถาปัตยกรรม โหลด/สโตร์ (Load/Store Architecture)

คำสั่งภาษาแอสเซมบลีที่บอกให้ชีพิญอ่านหรือเขียนข้อมูล เรียกว่า คำสั่งประเภท Load/Store ซึ่งจะได้กล่าวในรายละเอียดในบทที่ 4 หัวข้อที่ 4.5 และ การทดลองที่ 6 การพัฒนาโปรแกรมภาษาแอสเซมบลีในภาคผนวก F รีจิสเตอร์ภายในชีพิญมักมีจำนวนหลายชุด แบ่งตามชนิดของข้อมูลในบทที่ 2 ได้แก่ รีจิสเตอร์เลขจำนวนเต็ม และ รีจิสเตอร์เลขทศนิยมจุดลอยตัว รีจิสเตอร์เลขจำนวนเต็ม จะใช้เก็บค่าตัวแปรชนิด char, int, unsigned int, long เป็นต้น ตัวอย่างโปรแกรมในตารางที่ 3.2 ส่วนรีจิสเตอร์เลขทศนิยมจุดลอยตัวจะใช้เก็บค่าตัวแปรชนิด IEEE754 Single Precision และ Double Precision ทั้งนี้ขึ้นอยู่กับรายละเอียดของชีพิญแต่ละตัว

### 3.3.5 ชีพิญใช้งานอินพุตและเอาต์พุตต่างๆ ตามคำสั่งของซอฟต์แวร์ประยุกต์

องค์ประกอบสำคัญภายในเครื่องเนลของระบบปฏิบัติการลินุกซ์ในเวอร์ชั่นเมโมรี ประกอบด้วยมอดูลต่างๆ ที่สำคัญดังนี้ Process Management Subsystem, Memory Management Subsystem และ IO subsystem ดังรูปที่ 3.18 ในหัวข้อนี้จะกล่าวถึง I/O Subsystem เป็นหลัก ซึ่งการใช้งานอินพุต/เอาต์พุต เช่น คีย์บอร์ด เม้าส์ พринเตอร์บางรุ่น จะอาศัยการเชื่อมต่อ กับอุปกรณ์เหล่านี้ในลักษณะของ Character และมีซอฟต์แวร์เฉพาะ เรียกว่า ดีไวซ์ไดรเวอร์ (Device Driver) ซึ่งกำหนดรายละเอียดเอาไว้สำหรับผู้ผลิตและอุปกรณ์แต่ละรุ่น การเชื่อมต่อลักษณะนี้ ได้แก่ โปรแกรม Terminal ผู้อ่านสามารถเรียนรู้เพิ่มเติมจากการทดลองที่ 9 ภาคผนวก I

การเชื่อมต่อ กับเครื่องข่ายอินเทอร์เน็ตผ่านอุปกรณ์สื่อสารชนิดสายและไร้สาย จะอยู่ในรูปของ Network Interface ซึ่งจำเป็นต้องอาศัยโปรแกรมดีไวซ์ไดรเวอร์เช่นกัน การเชื่อมต่อลักษณะนี้ มีชื่อเฉพาะเรียกว่า ช้อกเก็ต (Socket)



รูปที่ 3.18: โครงสร้างของลินุกซ์เครื่องanelในเวอร์ชวลเมโมรี ประกอบด้วย I/O Subsystem, Memory Management Subsystem และ Process Management Subsystem ที่มา: [wikipedia.org](https://en.wikipedia.org)

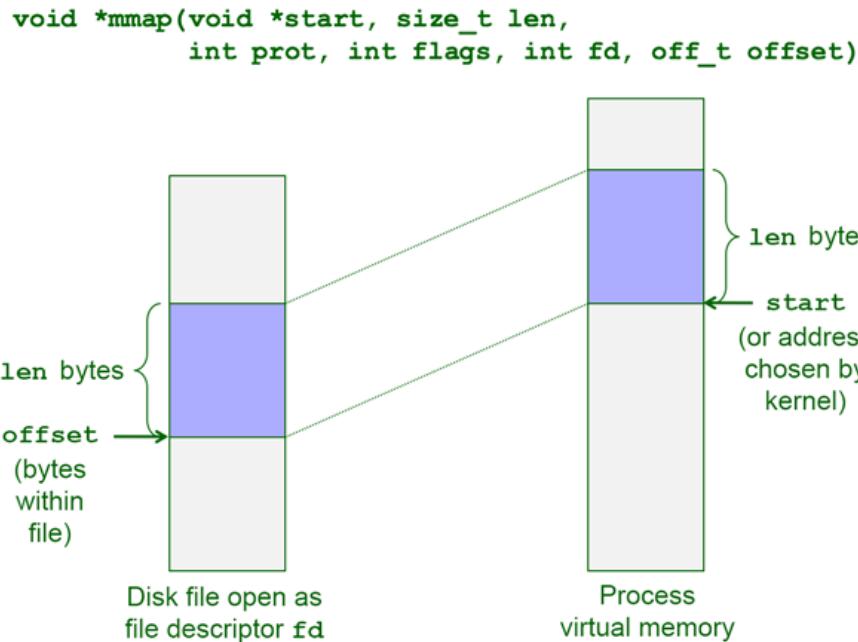
การใช้งานอินพุตและเอาต์พุต เช่น อุปกรณ์เก็บรักษาข้อมูล เครื่องอ่านจีดี/ดีวีดีรอม (CD/DVD Rom Drive) แฟลชไดรฟ์ (Flash Drive) เป็นต้น จะอาศัยการเข้ามต่อ กับ อุปกรณ์เหล่านี้ในลักษณะของบล็อก (Block) ข้อมูลขนาดตั้งแต่ 512 ไบต์ขึ้นไป การอ่านหรือเขียนข้อมูลจากไฟล์ จะต้องผ่านกระบวนการนี้ เช่นกัน ซึ่งระบบปฏิบัติการจะเป็นผู้บริหารจัดการ เปิดไฟล์ อ่าน/เขียนข้อมูล และปิดไฟล์ การเข้ามต่อลักษณะนี้ มีชื่อเฉพาะเรียกว่า ระบบไฟล์ (File Systems) การใช้งานอุปกรณ์อินพุต/เอาต์พุตทั้งสามรูปแบบเพื่อเข้ามต่อ อุปกรณ์ต่างๆ กับระบบปฏิบัติการ ในรูปแบบของ Virtual File System ซึ่งจะมีรายละเอียดเพิ่มเติมในบทที่ 7 และการทดลองที่ 12 ภาคผนวก L

### 3.3.6 ชีพิญอ่าน/เขียนไฟล์ข้อมูลในอุปกรณ์เก็บรักษาข้อมูล

นิยามที่ 3.3.1 ไฟล์ คือ เลขฐานสองขนาดหลายบิตเรียงต่อเนื่องกัน ซึ่งทุกๆ 8 บิตจะรวมกันเป็นไบต์ โดยที่แต่ละไบต์เรียงตัวต่อกันตั้งแต่ไบต์ที่ 0 (ต้นไฟล์) ไปจนถึงไบต์สุดท้าย ณ ตำแหน่งลิ้นสุดไฟล์ ซึ่งจะทำหน้าที่เป็นตัวอักขระหรือสัญลักษณ์ EOF (End of text File) ตัวอักขระหรือสัญลักษณ์ EOF เป็นเลขฐานสองขนาด 8 บิต เขียนเป็นฐานสิบหกเท่ากับ  $1A_{16}$  หรือฐานสิบเท่ากับ  $26_{10}$  ในตารางรหัสแอลกิ ใบรูปที่ 2.12

ไฟล์มีหลายชนิด เช่น

- ไฟล์โปรแกรมหรือแอปพลิเคชัน ทำหน้าที่บรรจุโปรแกรม (Executable File) หรือคำสั่ง (Command) ซึ่งมีรูปแบบ ELF ที่กล่าวในหัวข้อที่ 3.3.2 สำหรับระบบปฏิบัติการตระกูลยูนิกซ์และลินุกซ์
- ไฟล์ข้อมูล ได้แก่ ไฟล์เอกสาร Microsoft Words (.docx) ไฟล์เอกสาร Microsoft Excel (.xlsx) ไฟล์เอกสาร Microsoft PowerPoint (.pptx) ไฟล์รูปภาพ JPEG (.jpg) ไฟล์รูปภาพ PNG (.png) ไฟล์เสียง MP3 (MPEG2 Layer 3) ไฟล์เสียง WAV (.wav) ไฟล์ภาพเคลื่อนไหว MPEG4 (.mp4) ไฟล์ตัวอักษร เรียก ว่า Text File เป็นต้น ทั้งนี้ รายละเอียดโครงสร้างไฟล์ข้อมูลแต่ละชนิดจะแตกต่างกันไป



รูปที่ 3.19: หลักการของ Memory Mapped File ที่มา: [rice.edu](http://rice.edu)

- ไฟล์ชนิดอื่นๆ ประกอบการทำงานของโปรแกรมแต่ละตัว ซึ่งจะมีรายละเอียดแตกต่างกันไปตามที่นักพัฒนาออกแบบ

รูปที่ 3.19 แสดงหลักการของ Memory Mapped File เป็นการจองหน่วยความจำสำหรับอ่านหรือเขียนไฟล์ด้วยฟังก์ชัน mmap() ในภาษา C ด้านซ้ายคือ โครงสร้างของไฟล์ในเข็มตรรกะที่ใช้คำสั่ง fopen() เพื่อเปิดไฟล์ ด้านขวาคือ หน่วยความจำที่ถูกจงเพื่อใช้พักข้อมูลที่จะอ่านหรือเขียนในไฟล์ ที่มา: [rice.edu](http://rice.edu)

การใช้งานไฟล์จะต้องเริ่มด้วยการเปิดไฟล์เสมอ การเปิดไฟล์ (Open File) คือ การจองหน่วยความจำ เรียกว่า บัฟเฟอร์ (Buffer) ให้กับไฟล์ที่ต้องการเพื่ออ่าน หรือ เขียน การเปิดไฟล์มีหลายทางเลือก ได้แก่ เปิดไฟล์เพื่ออ่าน (Read Only) เพื่อเขียน (Write Only) เพื่ออ่าน/เขียน (Read/Write) เป็นต้น

การอ่านไฟล์ คือ กระบวนการอ่านข้อมูลจากไฟล์ในอุปกรณ์เก็บรักษาข้อมูล ມาบรรจุในหน่วยความจำในบริเวณที่จัดสรรให้ เรียกว่า บัฟเฟอร์ ด้วยขั้นตอน DMA (Direct Memory Access) รายละเอียดเพิ่มเติมในหัวข้อที่ 6.13 การอ่านไฟล์จึงเป็นการอ่านข้อมูลจากบัฟเฟอร์ โปรแกรมจะอ่านข้อมูลจากบัฟเฟอร์จากตำแหน่งเริ่มต้นไปเรื่อยๆ จนสิ้นสุดบัฟเฟอร์ หากไฟล์มีขนาดใหญ่มากเมื่อเทียบกับขนาดของบัฟเฟอร์ ซึ่งทำงานแบบวงกลม (Circular Buffer) ขั้นตอน DMA จะนำข้อมูลชุดถัดไปเขียนวนทับไปเรื่อยๆ จนเจอตัวอักษรสิ้นสุดไฟล์ หรือ EOF (End of File) เมื่อโปรแกรมอ่านไฟล์เสร็จสิ้นโปรแกรมเมอร์ควรปิดไฟล์ (Close File) เพื่อคืนหน่วยความจำบัฟเฟอร์ที่แมปไว้ให้ระบบปฏิบัติการแม้ว่าจะเพียงแค่เปิดไฟล์เพื่ออ่านเท่านั้น

เมื่อโปรแกรมประมวลผลข้อมูลตามที่ได้รับมอบหมายเรียบร้อย โปรแกรมหรือแอปพลิเคชันควรจะบันทึกหรือเขียนข้อมูลหรือสารสนเทศ (Information) เก็บลงในไฟล์ เพื่อป้องกันการสูญหายหรือเพื่อนำไปใช้ต่อ การเขียนไฟล์ คือ การเขียนข้อมูลไปยังหน่วยความจำบริเวณบัฟเฟอร์ที่แมปไว้ เมื่อบัฟเฟอร์เต็มระบบปฏิบัติการจะย้ายข้อมูลจากหน่วยความจำตำแหน่งนั้นๆ ไปเขียนลงในอุปกรณ์เก็บรักษาข้อมูลจริงเป็นระยะ ดังนั้น เมื่อใช้งาน (อ่านหรือเขียน) ไฟล์เสร็จสิ้นโปรแกรมเมอร์จะต้องปิดไฟล์เสมอ เพื่อนำข้อมูลที่ยังคงอยู่ในบัฟเฟอร์เขียนลงในอุปกรณ์เก็บรักษาข้อมูลด้วยขั้นตอน DMA เช่นกัน ก่อนที่จะคืนหน่วยความจำบัฟเฟอร์ที่แมปไว้ให้ระบบปฏิบัติการ รายละเอียดเพิ่มเติมในหัวข้อที่ 7.1 และในกิจกรรมท้ายการทดลองที่ 5 ภาคผนวก E

### 3.3.7 ผู้ใช้ชัตดาวน์ (Shut Down) ระบบปฏิบัติการ

การชัตดาวน์เป็นการสั่งให้ระบบปฏิบัติการอัปเดทข้อมูลต่างๆ ของเครื่องในรูปของไฟล์ต่างๆ กลับลงในอุปกรณ์เก็บรักษาข้อมูล และปิดไฟล์ที่เปิดค้างไว้ การปิดเครื่องโดยไม่สั่งชัตดาวน์อาจทำให้การบูตเครื่องครั้งต่อไปมีปัญหา และจะทำให้ระบบปฏิบัติการทำงานได้ไม่สมบูรณ์

ผู้ใช้งานระบบลินุกซ์สามารถถูกอนุญาติ (Permission) ของซูเปอร์ยูสเซอร์ (Super User) สั่ง ชัตดาวน์ระบบได้หลายวิธี เช่น การใช้มาสก์กดปุ่มชัตดาวน์ด้านซ้ายบนสุด หรือ พิมพ์คำสั่ง shutdown -h บนโปรแกรม Terminal การชัตดาวน์ทั้งสองวิธี คือ การส่งสัญญาณ (Signaling) ไปยังมอดูล init ให้เปลี่ยนระดับการรัน (Run Level) ให้เป็นระดับ 0 (Halt) เพื่อปิดระบบ

การพิมพ์คำสั่ง shutdown -r เพื่อเริ่มต้นเครื่อง เป็นการส่งสัญญาณไปยังมอดูล init ให้ระบบปฏิบัติการเปลี่ยนระดับการรันเป็นระดับ 6 การเริ่มต้นระบบนิยมใช้กับคอมพิวเตอร์ระบบฝั่งตัว เพื่อเริ่มต้นการทำงานของฮาร์ดแวร์และซอฟต์แวร์ใหม่ โปรแกรมเมอร์สามารถเขียนสคริปต์ (Script) ด้วยภาษาเชลล์สคริปต์ (Shell Script) ที่ถนนดเพื่อสั่งให้ระบบเริ่มต้นโดยอัตโนมัติ ระบบปฏิบัติการไมโครซอฟต์วินโดว์สามารถเขียนคำสั่งในลักษณะนี้ได้เช่นกัน เรียกว่า การเขียนคำสั่งเบทช์ (Batch Command)

## 3.4 สรุปท้ายบท

เนื้อหาในบทนี้ได้นำเสนอพื้นฐานของเครื่องคอมพิวเตอร์ซึ่งมีองค์ประกอบหลัก คือ ฮาร์ดแวร์ และ ซอฟต์แวร์ ฮาร์ดแวร์ประกอบด้วย อุปกรณ์อิเล็กทรอนิกส์ต่างๆ ซึ่งมีซีพียูเป็นจุดศูนย์กลาง เชื่อมหน่วยความจำหลัก และ วงจร/อุปกรณ์อินพุต เอาท์พุตต่างๆ รวมถึง อุปกรณ์เก็บรักษาข้อมูล ในขณะที่ซอฟต์แวร์ แบ่งเป็นซอฟต์แวร์ ระบบปฏิบัติการ และซอฟต์แวร์ประยุกต์ต่างๆ บรรจุอยู่ในอุปกรณ์เก็บรักษาข้อมูล เช่น การ์ดหน่วยความจำ SD ซอฟต์แวร์เหล่านี้สามารถพัฒนาด้วยภาษาต่างๆ เช่น C/C++ Java และภาษาแอสเซมบลี ด้วยการคอมไพล์ (Compile) และลิงก์ (Link) ให้กลายเป็นไฟล์รูปแบบ ELF บนระบบปฏิบัติการยูนิคซ์และลินุกซ์ ไฟล์ ELF มีโครงสร้างที่เป็นมาตรฐาน ประกอบด้วยเทกซ์เซกเมนต์ (Text Segment) ซึ่งรวมรวมคำสั่งภาษาเครื่องในรูปของเลขฐานสอง และ ดาต้าเซกเมนต์ (Data Segment) ซึ่งรวมรวมค่าตั้งต้นของตัวแปรที่ได้ประกาศไว้

## 3.5 คำถามท้ายบท

1. เหตุใดเครื่องคอมพิวเตอร์ทั่วไปควรมีระบบปฏิบัติการ เช่น Microsoft Windows, Apple MacOS, ลินุกซ์เวอร์ชันต่างๆ
2. ชิป BCM2837/BCM2711 และชิป AllWinner A64 มีความแตกต่างกันอย่างไร จากข้อมูลในรูปที่ 3.3 ในแต่ละชิปต่อไปนี้
  - การรองรับหน่วยความจำหลักภายนอกชิป
  - จีพียู
  - Video Engine ซึ่งประกอบด้วยวงจรเข้ารหัส (Encoder) และวงจรถอดรหัส (Decoder)
3. หน่วยความจำหลักที่วางขายมีลักษณะเป็นแพงแงะของจرمีซีปเปรียงกันหลายตัว เหตุใดบอร์ด Pi3/Pi4 จึงมีชิปเพียงตัวเดียว

4. คอมพิวเตอร์ทั่วไปอาจมีการ์ดจอเสริมการทำงานหรือเล่นเกมส์ บอร์ด Pi3/Pi4 มีการ์ดจอหรือไม่ เพราะเหตุใด
5. จงใช้งานเว็บไซต์ [www.gsmarena.com](http://www.gsmarena.com) เพื่อสืบค้นว่ามีการใช้งานชิปปี้ ARM Cortex A53/A72 ในเครื่องโทรศัพท์สมาร์ตโฟนหรือไม่ อย่างไร
6. ถ้าจะนำบอร์ด Pi3/Pi4 ไปปรับปรุงให้กล้ายเป็นเครื่องคอมพิวเตอร์แท็ปเล็ต จงจินตนาการว่าจะต้องเพิ่มเติมอุปกรณ์อะไรบ้าง โดยหาข้อมูลเพิ่มเติมในอินเทอร์เน็ต
7. เลเบล main และ printf ในรูปที่ 3.11 ตรงกับฟังก์ชันชื่ออะไรบ้าง
8. เลเบล printf ในรูปที่ 3.11 จะต้อง include ไฟล์ .h ซึ่งอะไรมีให้คอมไพล์ทำงานสำเร็จ
9. คำสั่ง BL ในรูปที่ 3.11 ตรงกับคำสั่งอะไรในภาษาแอสเซมบลีของ ARM
10. เลเบล main ในรูปที่ 3.11 ตรงกับบรรทัดที่เท่าไหร่ในตารางที่ 3.2
11. จงอธิบายการทำงานของโปรแกรมในตารางที่ 3.2 ว่าเป็นการวนรอบเพื่อหาค่าอะไรจากตัวแปรใด

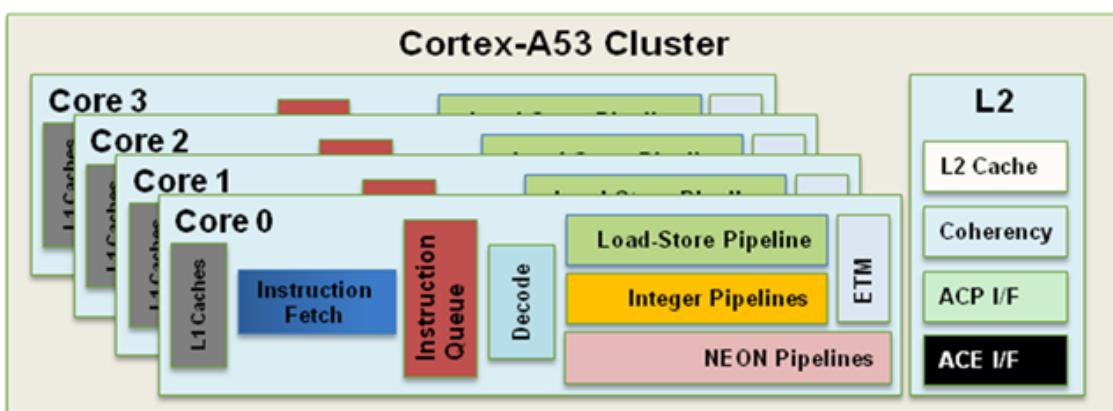
# บทที่ 4

## ภาษาแอสเซมบลีของ ARM ขนาด 32 บิต

การพัฒนาโปรแกรมด้วยภาษา C/C++ และแอสเซมบลีมีความเกี่ยวเนื่องกัน ผู้อ่านอาจมีพื้นฐานการพัฒนาโปรแกรมด้วยภาษา C/C++ มาบ้าง จากการทดลองที่ 5 ภาคผนวก E เนื่องจากคำสั่งภาษาแอสเซมบลี (Assembly Instruction) ของ ARM มีหลายเวอร์ชัน ได้แก่ 16, 32 และ 64 บิต อาจทำให้ผู้อ่านเข้าใจสับสน ดังนั้น บทนี้จะเน้นที่คำสั่งภาษาแอสเซมบลีขนาดหรือความยาว 32 บิต เป็นหลัก โดยมีวัตถุประสงค์ดังต่อไปนี้

- เข้าใจโครงสร้าง硬件ภายในชิปปี้ ARM Cortex A53/A72 บนบอร์ด Pi3/Pi4
- เข้าใจขั้นตอนการทำงานของคำสั่งภาษาแอสเซมบลีขนาดหรือความยาว 32 บิตที่ส่งให้ชิปปี้ ARM ทำงาน
- เข้าใจรูปแบบคำสั่งและการพัฒนาโปรแกรมด้วยคำสั่งภาษาแอสเซมบลีของ ARM โดยใช้พื้นฐานการพัฒนาโปรแกรมภาษา C
- รับรู้วิวัฒนาการของภาษาแอสเซมบลีในชิปปี้ที่ ARM ออกแบบและเป็นที่นิยมในตลาดโลก

### 4.1 โครงสร้างของชิปปี้ ARM Cortex A53/A72



รูปที่ 4.1: โครงสร้างภายในแกนประมวลผล 0 ทั้งหมด 4 แกนประมวลผลของ ชิปปี้ Cortex A53/A72 ออกแบบโดยบริษัท ARM ที่มา: [tomshardware.com](http://tomshardware.com)

จากรูปที่ 3.3 ผู้อ่านได้เรียนรู้โครงสร้างภายในของชิปที่คล้ายกับชิป BCM2837/BCM2711 ซึ่งเป็นศูนย์กลางของบอร์ด Pi3/Pi4 โดยภายในชิปมีชิปปี้ ARM Cortex A53/A72 จำนวน 4 แกนประมวลผล (Core) แกน

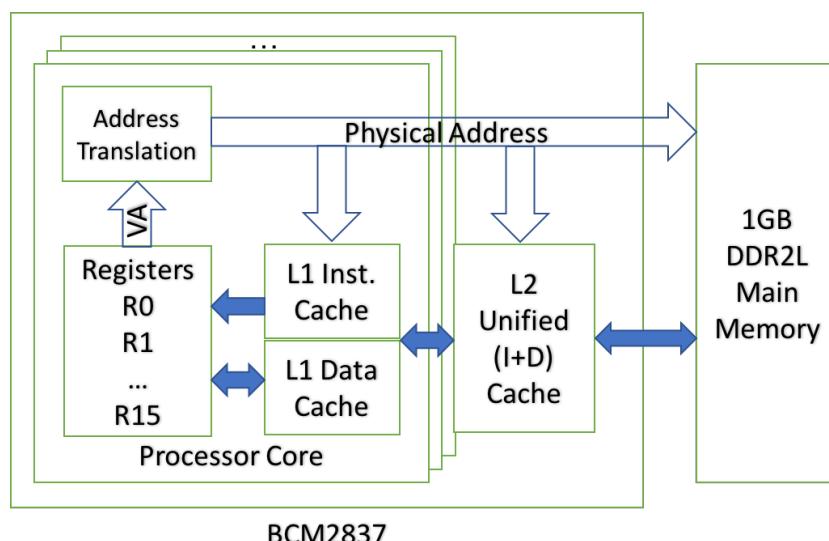
ประมวลผลหมายเลข 0 จนถึงแกนประมวลผลหมายเลข 3 ซึ่งความสามารถยืนยันได้จากผลการตรวจสอบในการทดลองที่ 4 ภาคผนวก D แกนประมวลผลทั้งสี่มีองค์ประกอบหลักภายในแต่ละแกนเหมือนกัน นั่นคือ แกนประมวลผลที่ 0 ของชิปปี้ Cortex A53/A72 ของรูปที่ 4.1 มีโครงสร้างภายในที่ไม่ซับซ้อนและมีการทำงานแบบไปป์ไลน์ (Pipeline) ประกอบด้วยมอดูล/บล็อกของจะจากด้านซ้ายสุดไปด้านขวาสุด ดังนี้

- **แคชคำสั่งลำดับที่ 1** ทำหน้าที่เก็บคำสั่ง (Instruction) เรียกว่า I-Cache ลำดับที่ 1 โดยจะเก็บคำสั่งภาษาเครื่องล่าสุด ซึ่งคำสั่งเหล่านี้ถูกเพทซ์มาจากเทกซ์เชิกเมนท์ของเวอร์ชวลเมโมรี การทำงานเบื้องต้นของแคชคำสั่งคล้ายกับบัฟเฟอร์ ทำหน้าที่เก็บคำสั่งล่าสุดที่อ่านมาเพื่อถอดรหัส (Decode) แคชคำสั่งนี้มีขนาดเล็กประมาณ 16-64 KiB สามารถจุคำสั่งได้ 4,096-16,536 คำสั่ง (แต่ละคำสั่งยาว 32 บิต หรือ 4 ไบต์ต่อคำสั่ง) รายละเอียดการทำงานจะอธิบายโดยละเอียดในหัวข้อที่ 5.3
- **วงจรเฟทซ์คำสั่งภาษาเครื่อง** (Instruction Fetch) ทำหน้าที่แปลเวอร์ชวล แอดเดรสเป็นแอดเดรสภายในภาพและส่งแอดเดรสภายในภาพไปยังแคชคำสั่ง (I-Cache) ลำดับที่ 1 และรอรับคำสั่งภาษาเครื่องที่เพทซ์มาพักเก็บในคิว (Instruction Queue)
- **คิวคำสั่ง** (Instruction Queue) ทำหน้าที่เป็นคิวเก็บพักคำสั่งภาษาเครื่องจำนวน 8-16 คำสั่ง เพื่อส่งต่อให้วงจรถอดรหัส
- **วงจรถอดรหัส** (Decode) ทำหน้าที่แปลความหมายของคำสั่งภาษาเครื่องซึ่งเป็นเลขฐานสองที่ส่งมาจากคิวคำสั่ง วงจรถอดรหัสแบ่งคำสั่งภาษาเครื่องออกเป็นชุดบิตต่างๆ ตามตัวอย่างในรูปที่ 3.12 หัวข้อที่ 3.2.5 เพื่อทำความสะอาดเข้าใจและส่งสัญญาณควบคุม (Control Signal) ไปยังมอดูล/วงจรไปป์ไลน์ที่เหมาะสมตามชนิดของคำสั่งนั้นๆ
- **วงจรไปป์ไลน์** (Pipeline) สำหรับปฏิบัติตาม (Execute) คำสั่งภาษาเครื่องที่ผ่านการถอดรหัสแบ่งเป็นวงจร 3 ชนิดตามประเภทของคำสั่ง ประกอบด้วย
  - **วงจรไปป์ไลน์สำหรับการอ่าน/เขียนค่าตัวแปรกับหน่วยความจำ** (Load-Store Pipeline) เพื่ออ่าน/เขียนค่าของตัวแปรเลขจำนวนเต็มและเลขทศนิยม IEEE754 กับหน่วยความจำเวอร์ชวลเม莫รีผ่านทางแคชข้อมูล (Data Cache หรือ D-Cache) ลำดับที่ 1 มีการทำงานเบื้องต้นคล้ายกับบัฟเฟอร์ขนาดเล็ก ทำหน้าที่พากเก็บข้อมูลหรือค่าของตัวแปรที่ซอฟต์แวร์ต้องการอ่านหรือเขียน จำกัดตัวเชิกเมนท์ สเตกเชิกเมนท์ ยกเว้นเทกซ์เชิกเมนท์ การทำงานเบื้องต้นของแคชข้อมูลคล้ายกับแคชคำสั่ง ต่างกันตรงที่ชิปปี้สามารถเปลี่ยนแปลงค่าของข้อมูลในแคชข้อมูลได้ แต่ไม่สามารถเปลี่ยนแปลงแคชคำสั่งได้ ค่าของตัวแปรจะถูกถ่ายโอน (อ่าน/เขียน) ระหว่างรีจิสเตอร์กับแคชข้อมูลด้วยคำสั่งในหัวข้อที่ 4.5 เมื่อรีจิสเตอร์มีค่าของตัวแปรต่างๆ แล้ว โปรแกรมจึงสามารถสั่งให้ชิปปี้คำนวณค่าเหล่านั้นด้วยวงจรไปป์ไลน์สำหรับการประมวลผลเลขจำนวนเต็ม และวงจรไปป์ไลน์สำหรับการประมวลผลเลขทศนิยม IEEE754
  - **วงจรไปป์ไลน์สำหรับการประมวลผลเลขจำนวนเต็ม** (Integer Pipeline) ขนาด 8, 16, 32 และ 64 บิต ด้วยคำสั่งทางคณิตศาสตร์และตรรกศาสตร์ตามชนิดข้อมูลในบทที่ 2 ข้อมูลเหล่านี้เป็นข้อมูลเชิงเต็ยว หรือ สเกลาร์ (Scalar) ซึ่งมีค่าเก็บอยู่ในรีจิสเตอร์จำนวนเต็ม (R0-R15) เท่านั้น ตัวรากล่มนี้จะเน้นที่การประมวลผลเลขจำนวนเต็มเป็นหลัก
  - **วงจรไปป์ไลน์สำหรับการประมวลผลเลขทศนิยมฐานสองชนิดจุดลอยตัว** IEEE754 หรือ NEON (Floating-Point Pipeline) สามารถรองรับการประมวลผลเลขทศนิยมฐานสองชนิดจุดลอยตัว

มาตรฐาน IEEE754 ตามรายละเอียดในหัวข้อที่ 2.6 โดยจะสามารถรองรับการคำนวณข้อมูล เชิงเดี่ยวหรือสเกลาร์ ได้แก่ เลขจำนวนจริง เลขทศนิยมฐานสิบ เป็นต้น และข้อมูลแบบเวกเตอร์ (Vector) ที่ต้องการความแม่นยำสูง ได้แก่ ตัวแปรอาร์เรย์ ข้อมูลตำแหน่งตัวละครในเกม 3 มิติ ข้อมูลจุดภาพ ข้อมูลเสียง เป็นต้น

- แคชลำดับที่ 2 (Level 2) มีความจุมากกว่าแคชลำดับที่ 1 หลายเท่าตัวแต่ต้องใช้เวลาเข้าถึง (Access Time) นานกว่าแคชลำดับที่ 1 โดยที่ซีพียูทั้งสิ้นแกนประมวลผลจะใช้งานแคชลำดับที่ 2 ร่วมกัน (Share) หมายความว่า ซีพียุกแกนประมวลผลสามารถค้นหาคำสั่งและข้อมูลในแคชลำดับที่ 2 หลังจากที่ไม่เจอคำสั่ง/ข้อมูลในแคชลำดับที่ 1 (L1) ภายในแต่ละแกนประมวลผล หากไม่เจอในแคชลำดับที่ 2 อีก ซีพียูจะต้องค้นหาในหน่วยความจำภายในตัวและต้องรอนานขึ้นอีก บทที่ 5 จะอธิบายการทำงานของหน่วยความจำต่างๆ โดยละเอียด

ผู้อ่านสามารถค้นคว้ารายละเอียดเกี่ยวกับซีพียู ARM Cortex A53/A72 เพิ่มเติมได้ที่ [wikichip.org](http://wikichip.org)



รูปที่ 4.2: โครงสร้างเชิงตรรกะของแกนประมวลผลประกอบด้วยรีจิสเตอร์ R0-R15 แคชต่างๆ และหน่วยความจำหลัก, (VA: Virtual Address)

วงจร วงจรไปป์ไลน์สำหรับการประมวลผลเลขจำนวนเต็ม ในแต่ละแกนประมวลผลของ ARM Cortex A53/A72 ในรูปที่ 4.2 ประกอบด้วย รีจิสเตอร์จำนวนเต็ม R0-R15 แคชต่างๆ และหน่วยความจำต่างๆ เพื่อให้สามารถประมวลผลตัวเลขจำนวนเต็มเท่านั้น โดยอาศัยวงจร ALU (Arithmetic Logic Unit) ตามที่ได้อธิบายไปแล้วในบทที่ 2

ส่วนของวงจร วงจรไปป์ไลน์สำหรับการอ่าน/เขียนค่าตัวแปรกับหน่วยความจำ จะเกี่ยวข้องกับหน่วยความจำชนิดต่างๆ โดยตัวแปรซึ่งต่างๆ นั้นถูกกำหนดพื้นที่ไว้ในเซกเมนต์ต่างๆ ของหน่วยความจำยกเว้นเทิกซ์เซกเมนต์ แต่การประมวลผลค่าของตัวแปรเหล่านี้ โปรแกรมจะต้องสั่งให้ซีพียูโหลด (Load) หรืออ่านค่าของตัวแปรชนิดเลขจำนวนเต็มจากหน่วยความจำ มาพักเก็บในรีจิสเตอร์ R0 - R12 ก่อน แล้วจึงใช้คำสั่งทางคณิตศาสตร์และตรรกศาสตร์ต่างๆ ประมวลผลเลขจำนวนเต็มในรีจิสเตอร์ R0 ถึง R12 เมื่อประมวลผลแล้วเสร็จ โปรแกรมต้องนำค่าจากรีจิสเตอร์ไปเก็บ (Store) หรือเขียนในหน่วยความจำ ณ ตำแหน่งตัวแปรที่ได้โหลดมาด้วยเหตุผลด้านประสิทธิภาพ ซึ่งจะอธิบายในบทที่ 5 ดังนั้น เราจึงเรียกการทำงานลักษณะนี้ว่า สถาปัตยกรรมโหลด/สโตร์ (Load/Store Architecture)

## 4.2 สถาปัตยกรรมชุดคำสั่ง (Instruction Set Architecture)

ตารางเลื่อนี้จะเน้นคำสั่งภาษาแอสเซมบลีที่ตรงกับภาษาเครื่องของ ARM ที่มีความยาว 32 บิตหรือ 4 ไบต์ สำหรับการประมวลผลเลขจำนวนเต็มชนิดไม่มีเครื่องหมาย และชนิดมีเครื่องหมาย แบบ 2's Complement คำสั่งเหล่านี้ประกอบด้วยรายละเอียดในเบื้องต้นต่างๆ เหล่านี้

- ชนิดของคำสั่งภาษาแอสเซมบลี แบ่งเป็น
    - คำสั่งประการและตั้งค่าเริ่มต้นตัวแปร ในหัวข้อที่ 4.4
    - คำสั่งการถ่ายโอนข้อมูลระหว่างตัวแปรในหน่วยความจำกับรีจิสเตอร์ ในหัวข้อที่ 4.5
    - คำสั่งประมวลผลคณิตศาสตร์และตรรกศาสตร์จากข้อมูลในรีจิสเตอร์ ในหัวข้อที่ 4.6.1
    - คำสั่งการควบคุมการทำงาน เพื่อการตัดสินใจและการวนรอบทำซ้ำ ในหัวข้อที่ 4.7
    - คำสั่งเรียกใช้ฟังก์ชันและรีเทิร์น (Return) กลับ ในหัวข้อที่ 4.8.2
  - รีจิสเตอร์ ทำหน้าที่พักเก็บข้อมูลชั่วคราวสำหรับประมวลผลด้วยคำสั่งทางคณิตศาสตร์ ตรรกศาสตร์ และอื่นๆ มีจำนวนรวม 16 ตัว เรียกว่า R0, R1, ..., R15 รีจิสเตอร์ทุกตัวมีความยาว 32 บิต แต่ละตัวมีหน้าที่แตกต่างกัน คือ
    - รีจิสเตอร์ R15 เรียกว่า โปรแกรมเดาน์เตอร์ (Program Counter: PC) คือ รีจิสเตอร์สำหรับเก็บแอดเดรส หรือ หมายเลขไปต์ ของคำสั่งในเก็ปเมนต์ของหน่วยความจำที่ซีพียูจะ
      - \* เฟทช์(Fetch) หรืออ่านคำสั่งภาษาเครื่อง
      - \* ถอดรหัส (Decode) คำสั่งที่เฟทช์มาเพื่อสร้างสัญญาณไปควบคุม
      - \* วงจรปฏิบัติ (Execute) ตามคำสั่งนั้นและ
      - \* เปลี่ยนแปลงค่าในรีจิสเตอร์ PC เป็น  $PC=PC+4$  เพื่อกีบแอดเดรสของคำสั่งถัดไป หมายเลข 4 หน่วยเป็นไปต์ เนื่องจากทุกคำสั่งในตำราเล่มนี้มีความยาว 4 ไบต์ตามที่กล่าวมาข้างต้น หรือเปลี่ยนแปลงเป็นค่าอื่นตามคำสั่งควบคุมการทำงานและคำสั่งเรียกใช้ฟังก์ชัน
    - รีจิสเตอร์ R14 เรียกว่า ลิงก์รีจิสเตอร์ (Link Register: LR) คือ รีจิสเตอร์สำหรับเก็บแอดเดรสของคำสั่งที่ต้องการจะรีเทิร์นกลับ โดยรีจิสเตอร์นี้ทำงานคู่กับคำสั่ง BL (Branch and Link) และคำสั่ง BX LR
    - รีจิสเตอร์ R13 เรียกว่า สเตกพอยน์เตอร์ (Stack Pointer: SP) คือ รีจิสเตอร์สำหรับเก็บแอดเดรส หรือตำแหน่งยอด (Top) ของสเตกเก็ปเมนต์ ซึ่งเรียกสั้นๆ ว่า สเตก โดยรีจิสเตอร์นี้ทำงานคู่กับคำสั่งที่เกี่ยวข้องกับหน่วยความจำ ในหัวข้อที่ 4.5 ผู้อ่านสามารถทบทวนรายละเอียดเกี่ยวกับสเตกในหัวข้อที่ 3.3.3 ที่ผ่านมา หมายเหตุ คำว่า สเตก ในตำราเล่มนี้หมายถึงสเตกเก็ปเมนต์ มิได้หมายถึงโครงสร้างข้อมูล (Data Structure) ชนิดหนึ่ง
    - รีจิสเตอร์ R4-R12 เป็นรีจิสเตอร์สำหรับใช้งานทั่วไป
    - รีจิสเตอร์ R0-R3 เป็นรีจิสเตอร์สำหรับสำหรับใช้งานทั่วไป และใช้ส่งค่าพารามิเตอร์ (Parameter) ไปให้ฟังก์ชัน โดยรีจิสเตอร์เหล่านี้ทำงานร่วมกับคำสั่ง BL (Branch and Link)
    - รีจิสเตอร์ R0 เป็นรีจิสเตอร์สำหรับรีเทิร์นค่าข้อมูลจากฟังก์ชัน โดยทำงานร่วมกับคำสั่ง BX LR

- ชนิด และ ขนาด ของ ตัวแปร ผู้อ่านสามารถ เทียบ เคียง พื้นที่ และ ขนาด ของ หน่วย ความจำ (Memory Space) กับ ตารางที่ [2.1](#) โดยข้อมูลที่เก็บอยู่ใน ตัวแปร แต่ละ ชนิด ต้อง การ พื้นที่ ไม่เท่า กัน ดังนี้
  - ชนิด **ไบต์** (byte) มี ขนาด 8 บิต เหมาะสำหรับ ตัวแปร ชนิด อักขระ เช่น char สำหรับ เก็บ อักขระ ตาม รหัส มาตรฐาน ASCII ด้วย พื้นที่ 8 บิต ใน หน่วย ความจำ และ unsigned char สำหรับ เลข จำนวนเต็ม ไม่มีเครื่องหมาย ความ ยาว 8 บิต ซึ่งได้ สรุปไว้ ใน ตารางที่ [2.13](#)
  - ชนิด **halfword** มี ขนาด 16 บิต เช่น short และ unsigned short เหมาะสำหรับ ตัวแปร ชนิด อักขระ ตาม มาตรฐาน Unicode
  - ชนิด **word** มี ขนาด 32 บิต เหมาะสำหรับ ตัวแปร ชนิด จำนวนเต็ม เช่น int และ unsigned int เป็นต้น
  - ชนิด **doubleword** 64 บิต เหมาะสำหรับ ตัวแปร ชนิด จำนวนเต็ม เช่น unsigned long long เป็นต้น

### 4.3 ตัวอย่าง คำสั่งภาษาเครื่อง ใน เทิกซ์เซกเมนต์

นักพัฒนา โปรแกรม บน อุปกรณ์ ที่ใช้ ชีพิญ ของ ARM สามารถ จำลอง (Simulate) การ ทำงาน บน เครื่อง คอมพิวเตอร์ ตั้ง โต๊ะ เพื่อ จำลอง การ ทำงาน ของ ซอฟต์แวร์ และ ฮาร์ดแวร์ ใน สภาพ ใกล้ เดียวกับ สถานการณ์ จริง โปรแกรม เมอร์ สามารถ แก้ ปัญหา ที่เกิด ขึ้น ได้ ก่อน ทำ ให้ ประหัด เวลา และ ต้น ทุน รูปที่ [4.3](#) แสดง คำสั่งภาษาเครื่อง ที่ ถูก อ่าน (Load) เข้า สู่ หน่วย ความจำ และ แสดง ใน รูป ของ ภาษา แอสเซมบลี บน โปรแกรม ซิมูเลเตอร์ (Simulator) ประกอบ ด้วย คอลัมน์ ต่างๆ ต่อไปนี้

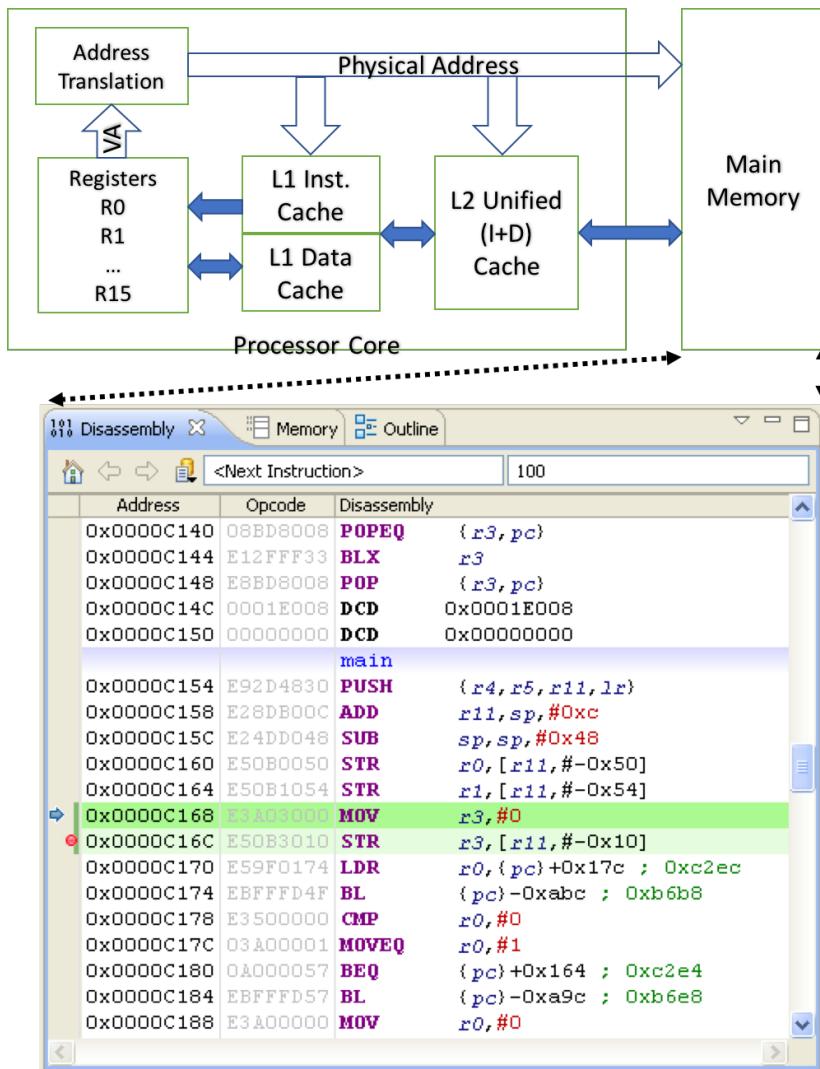
- แอดเดรส (Address)** เท่า กับ หมาย เลข ไบต์ ใน รูปแบบ เลขฐาน สิบ หลัก จำนวน 8 หลัก (ขีด ตัน ด้วย 0x) ซึ่ง เท่า กับ เลขฐาน สิบ จำนวน 32 บิต ใน คอลัมน์ ทาง ซ้าย ผู้อ่าน ควร จินตนาการ หมาย เลข ไบต์ นี้ เป็น หมาย เลข ชั้น ของ อาคาร ที่ มี จำนวน ชั้น เยอะ มาก ซึ่ง ใน ที่นี่ มี จำนวน  $2^{32} = 4,294,967,296$  ชั้น แต่ ละ ชั้น สามารถ บรรจุ ข้อมูล เพียง 1 ไบต์ โดย จะเริ่ม นับ จาก ชั้น ที่ 0x0000 0000 หรือ 0000 0000<sub>16</sub> จน ถึง ชั้น ที่ 0xFFFF FFFF หรือ FFFF FFFF<sub>16</sub>

- อปโคด (Opcode)** คือ คำสั่งภาษาเครื่อง ใน รูปของ เลขฐาน สิบ หลัก หรือ เลขฐาน สิบ จำนวน 8 หลัก หรือ เลขฐาน สิบ จำนวน 32 บิต ผู้อ่าน สามารถ ทำความเข้าใจ คำสั่งภาษาเครื่อง ได้ จาก ตัวอย่าง ใน หัวข้อ [3.2.5](#)

- ดิสแอสเซมบลี (Disassembly)** คือ การ แปล คำสั่งภาษาเครื่อง ใน คอลัมน์ อปโคด ให้ กลับ เป็น ภาษา แอสเซมบลี ARM โดย จะ แปล ภาษาเครื่อง แต่ละ บรรทัด เป็น คำสั่งภาษา แอสเซมบลี เพื่อ ให้ ผู้ใช้ โปรแกรม ซิมูเลเตอร์ อ่าน และเข้าใจ ง่าย ขึ้น

ยก ตัวอย่าง เช่น ที่ หน่วย ความจำ ตำแหน่ง 0x0000 C140 บรรจุ อปโคด 08BD 8008<sub>16</sub> ความ ยาว 4 ไบต์ ซึ่ง เป็น รหัส ภาษาเครื่อง ของ คำสั่ง POPEQ r3, pc

ตำแหน่ง กด ไป คือ แอดเดรส  $0x0000 C144 = 0x0000 C140 + 0x0000 0004$  ห่าง จำก เดิม จำนวน 4 ไบต์ บรรจุ อปโคด E12F FF33<sub>16</sub> ซึ่ง เป็น รหัส ภาษาเครื่อง ของ คำสั่ง BLX r3 หมายเหตุ ช่วง ว่าง ทุกๆ 4 หลัก คือ สัญลักษณ์ ที่ ผู้เขียน ใส่ เพิ่ม เอง เพื่อ ช่วย ให้ ผู้อ่าน สามารถ อ่าน หมาย เลข ที่ เรียบ ติด กัน ทีละ 4 หลัก ได้ ง่าย ขึ้น มี หน้า ที่ คล้าย กับ เครื่อง หมาย , ใน เลขฐาน สิบ ที่ ช่วย แบ่ง เลข จำนวน มาก ทุกๆ สาม หลัก



รูปที่ 4.3: คำสั่งภาษาเครื่องที่ถูกอ่าน (Load) เข้าสู่หน่วยความจำและแสดงในรูปของภาษาแอสเซมบลีบนโปรแกรมซิมูเลเตอร์ (Simulator) ในบริเวณแท็บเมนูที่มา: [arm.com](http://arm.com)

pc หรือ PC คือ รีจิสเตอร์ R15 ทำหน้าที่เก็บแอดเดรสของคำสั่งปัจจุบันในหน่วยความจำ โดยสังเกตได้จาก ลูกศรสีน้ำเงิน เพื่อให้ชี้ไปยังตำแหน่งคำสั่งภาษาเครื่องนั้นไปลดอัตราการเข้าสู่หน่วยความจำตาม ลูกศรสีน้ำเงินใน รูปที่ 4.3 ซึ่งแสดงให้เห็นว่าคำสั่ง MOV r3, #0 ที่อยู่ที่地址 0x0000 C168 ความหมายคือ ค่าในรีจิสเตอร์ pc เท่ากับ 0x0000 C168 ซึ่งหน่วยความจำที่ตำแหน่งนี้บรรจุคำสั่งหรืออพโอด E3A0 3000<sub>16</sub> ซึ่งเป็นเลขฐานสิบหกของคำสั่งภาษาแอสเซมบลี MOV r3, #0

เมื่อประมวลผลคำสั่งที่แอดเดรส 0x0000 C168 สำเร็จแล้ว pc จะบวกเพิ่มขึ้นอีก 4 ไบต์ ( $pc = pc + 4$ ) เป็นแอดเดรส 0x0000 C16C เพื่อนำคำสั่ง E50B 3010<sub>16</sub> ซึ่งเป็นรหัสภาษาเครื่องของคำสั่ง STR r3, [r11, #-0x10] มาถอดรหัสและประมวลผลตามต่อไป แต่ในรูปที่ 4.3 นี้ผู้ใช้ได้ตั้งตำแหน่งเบรกพอยน์ (Break Point) โดยสังเกตได้จากการกลมสีแดงทางซ้ายสุด เบรกพอยน์ จะทำให้โปรแกรมซิมูเลเตอร์หยุดทำงาน ชั่วคราว เพื่อให้ผู้ใช้สามารถศึกษาการทำงานของคำสั่งจากค่ารีจิสเตอร์ต่างๆ ที่เกี่ยวข้อง หลังจากนั้นผู้ใช้สามารถดำเนินการรันซิมูเลชัน (Simulation) ต่อไป ผู้อ่านสามารถฝึกฝนการใช้โปรแกรมลักษณะนี้ในการทดลองที่ 5 ด้วยภาษา C ก่อน และจึงทำการทดลองที่ 6 ด้วยภาษาแอสเซมบลีของ ARM

## 4.4 คำสั่งประการและตั้งค่าเริ่มต้นของตัวแปรในดาต้าเช็กเมนต์

ผู้อ่านต้องระลึกไว้เสมอว่าตัวแปรทุกชนิดอาศัยพื้นที่เก็บค่าตั้งต้นในดาต้าเช็กเมนต์เสมอ ค่าของตัวแปรในหน่วยความจำจะถูกอ่าน (Load) มาพักเก็บในรีจิสเตอร์ก่อน รีจิสเตอร์จะเป็นแค่ที่เก็บพักค่าข้อมูลชั่วคราวของตัวแปรใดๆ สำหรับการประมวลผลเท่านั้น เมื่อคำนวณได้ผลลัพธ์เสร็จแล้ว ค่าหรือข้อมูลในรีจิสเตอร์จะถูกเขียน (Store) ณ ตำแหน่งของตัวแปรตัวแปร (Variable) ในหน่วยความจำ

ตารางที่ 4.1: ตัวอย่างโปรแกรมภาษาแอสเซมบลีเพื่อประการและตั้งค่าเริ่มต้นตัวแปรขนาด 32 บิต จำนวน 2 ตัวแปร ในพื้นที่ของดาต้าเช็กเมนต์

#	เลข	คำสั่ง	คอมเมนต์
1		.data	@Data Segment begins here
2		.balign 4	@Align this variable to 4-byte space
3	wordvar1:	.word 7	@wordvar1=7
4		.balign 4	@Align this variable to 4-byte space
5	wordvar2:	.word 3	@wordvar2=3

ตารางที่ 4.1 ตัวอย่างโปรแกรมภาษาแอสเซมบลีเพื่อประการและตั้งค่าตัวแปรขนาด 32 บิต จำนวน 2 ตัวแปร บรรทัดที่ 1 คำสั่ง .data คือ การบ่งบอกถึงการเริ่มต้นประการตัวแปร คำสั่ง .balign คือการสั่งให้การจองข้อมูลในหน่วยความจำให้มีพื้นที่เรียงต่อเนื่องกัน 4 ไบต์ เพื่อให้ง่ายต่อการจัดการได้ง่ายขึ้น ตัวแปรนิดจำนวนเต็ม ความยาว 1 เวิร์ดหรือ 4 ไบต์ซึ่งในตารางที่ 4.1

- บรรทัดที่ 3 เป็นการตั้งค่าเริ่มต้นของตัวแปร wordvar1 เท่ากับ  $7_{10}$  หรือเท่ากับ  $0000\ 0007_{16}$
- บรรทัดที่ 5 เป็นการตั้งค่าเริ่มต้นของตัวแปร wordvar2 เท่ากับ  $3_{10}$  หรือเท่ากับ  $0000\ 0003_{16}$

## 4.5 คำสั่งถ่ายโอน (Transfer) ค่าตัวแปรระหว่างดาต้าเช็กเมนต์และรีจิสเตอร์

คำสั่งถ่ายโอนข้อมูลระหว่างตัวแปรในหน่วยความจำ และรีจิสเตอร์ เป็นคำสั่งที่จำเป็นสำหรับโปรแกรมคอมพิวเตอร์ เนื่องจากตัวแปรที่ประการไว้ คือ ชื่อของข้อมูล หรือชื่อชุดข้อมูล หรือชื่ออาร์เรย์ในหน่วยความจำ โดยขนาดของข้อมูลจะเปลี่ยนแปลงตามชนิด เช่น word ขนาด 32 บิต byte มีขนาด 8 บิต เป็นต้น ดังนั้น ซึ่งจะต้องใช้คำสั่งโหลดหรืออ่านค่าของตัวแปรนั้นมาพักไว้ในรีจิสเตอร์ก่อน โดยใช้คำสั่ง LDR สำหรับข้อมูลชนิด word และ LDRB สำหรับข้อมูลชนิด byte ในตารางต่อไปนี้

รูปแบบ	ความหมาย (Rd, Rm และ Rn คือ รีจิสเตอร์ R0 ถึง R12)
LDR Rd, [Rn]	<ul style="list-style-type: none"> <li><math>Rd = \text{Mem}[Rn]</math></li> <li>สำเนาข้อมูล 32 บิต จากหน่วยความจำที่แอดเดรส <math>Rn</math> ไปเขียนในรีจิสเตอร์ <math>Rd</math></li> </ul>
STR Rd, [Rn]	<ul style="list-style-type: none"> <li><math>\text{Mem}[Rn] = Rd</math></li> <li>สำเนาข้อมูล 32 บิตในรีจิสเตอร์ <math>Rd</math> ไปเขียนในหน่วยความจำที่แอดเดรส <math>Rn</math></li> </ul>
LDRB Rd, [Rn]	<ul style="list-style-type: none"> <li><math>Rd = \text{Mem}[Rn]</math></li> <li>สำเนาข้อมูลจำนวน 8 บิตจากหน่วยความจำที่แอดเดรส <math>Rn</math> ไปเขียนในรีจิสเตอร์ <math>Rd</math></li> </ul>
STRB Rd, [Rn]	<ul style="list-style-type: none"> <li><math>\text{Mem}[Rn] = Rd</math></li> <li>สำเนาข้อมูล 8 บิตล่างสุดในรีจิสเตอร์ <math>Rd</math> ไปเขียนในหน่วยความจำที่แอดเดรส <math>Rn</math></li> </ul>
PUSH {register list}	<ul style="list-style-type: none"> <li>สำเนาข้อมูล 32 บิตจากรีจิสเตอร์ที่ปรากฏในรายชื่อรีจิสเตอร์ที่ปะรุงรักไว้ไปวางบนยอดสแต็กซึ่งคราวตามลำดับจากซ้ายไปขวา</li> <li>ปรับเปลี่ยนค่ารีจิสเตอร์ <math>SP</math> ให้สอดคล้องกับคำสั่ง</li> </ul>
POP {register list}	<ul style="list-style-type: none"> <li>สำเนาข้อมูล 32 บิตจากยอดสแต็กไปบรรจุในรีจิสเตอร์ที่ปรากฏในรายชื่อรีจิสเตอร์ตามลำดับจากขวาไปซ้าย</li> <li>ปรับเปลี่ยนค่ารีจิสเตอร์ <math>SP</math> ให้สอดคล้องกับคำสั่ง</li> </ul>

คำสั่ง LDR  $Rd, [Rn]$  จะสั่งให้ชี้พิภูมิสำเนาข้อมูล 32 บิต จากหน่วยความจำ ณ แอดเดรสที่รีจิสเตอร์  $Rn$  เก็บไปเขียนในรีจิสเตอร์  $Rd$  ซึ่งตรงกับความหมายของประโยชน์นี้  $Rd = \text{Mem}[Rn]$

ในทางกลับกัน คำสั่งสโตร์จะสั่งให้อาร์ดแวร์ภายในชีพิญุดำเนินการ โดยอ่านค่าจากรีจิสเตอร์ไปบันทึกในหน่วยความจำ ณ แอดเดรสที่ตัวแพรนั้นตั้งอยู่ ยกตัวอย่าง คำสั่ง STR  $Rd, [Rn]$  จะสั่งให้ชีพิญุสำเนาข้อมูล 32 บิต จากรีจิสเตอร์  $Rd$  ไปเก็บยังหน่วยความจำ ณ แอดเดรสที่รีจิสเตอร์  $Rn$  ซึ่งตรงกับความหมายของประโยชน์นี้  $\text{Mem}[Rn] = Rd$

คำสั่ง PUSH และ POP มีลักษณะพิเศษกว่าคำสั่ง Load และ Store ทั่วไป คือ PUSH register list จะทำการสำเนาข้อมูลจากรีจิสเตอร์ไปวางบนสแต็กซึ่งคราว แล้วขยับตำแหน่งบนสุดของสแต็กไปยังหมายเลขแอดเดรสใหม่ โดยสแต็ก คือ ชื่อย่อของสแต็กเช็คเม้นต์ เป็นพื้นที่สำคัญของหน่วยความจำในรูปที่ 3.16 สำหรับคำสั่ง POP register list จะทำงานตรงข้ามกับ คำสั่ง PUSH คือ สำเนาข้อมูลจากสแต็กลับไปคืนให้รีจิสเตอร์ แล้วขยับตำแหน่งบนสุดของสแต็กไปยังหมายเลขแอดเดรสใหม่ ผู้อ่านสามารถทำความเข้าใจจากการทดลองที่ 8 ในภาคผนวก H

ตารางต่อไปนี้สรุปคำสั่งโอนถ่ายข้อมูลระหว่างหน่วยความจำและรีจิสเตอร์ (Memory-Register Transfer Instructions) และโหมดการอ้างแอดเดรส (Addressing Mode) ชนิดต่างๆ

รูปแบบ	ความหมาย
	หมายเหตุ Rd Rm และ Rn คือ หมายเลขรีจิสเตอร์มีค่าเท่ากับ R0 - R12
LDR Rd, =var	<ul style="list-style-type: none"> <li>Rd = address(var)</li> <li>เขียนค่าแอดเดรสของตัวแปร var ลงในรีจิสเตอร์ Rd</li> </ul>
LDR Rd, [Rn]	<ul style="list-style-type: none"> <li>Rd = Mem[Rn]</li> <li>อ่านค่าจากหน่วยความจำที่แอดเดรส Rn และเขียนค่านั้นในรีจิสเตอร์ Rd</li> </ul>
LDR Rd, [Rn, #Imm]	<ul style="list-style-type: none"> <li>Rd = Mem[Rn + #Imm]</li> <li>อ่านค่าจากหน่วยความจำที่แอดเดรส Rn+Imm และเขียนค่านั้นในรีจิสเตอร์ Rd โดยที่ค่าในรีจิสเตอร์ Rn ไม่เปลี่ยนแปลง</li> </ul>
LDR Rd, [Rn], #Imm	<ul style="list-style-type: none"> <li>Rd = Mem[Rn]</li> <li>อ่านค่าจากหน่วยความจำที่แอดเดรส Rn และเขียนค่านั้นในรีจิสเตอร์ Rd หลังจากนั้นค่าในรีจิสเตอร์ Rn = Rn + #Imm</li> </ul>
LDR Rd, [Rn, #Imm]!	<ul style="list-style-type: none"> <li>Rd = Mem[Rn+ #Imm]</li> <li>อ่านค่าจากหน่วยความจำที่แอดเดรส Rn+Imm และเขียนค่านั้นในรีจิสเตอร์ Rd หลังจากนั้นค่าในรีจิสเตอร์ Rn = Rn + #Imm</li> </ul>
LDR Rd [Rn, Rm]	<ul style="list-style-type: none"> <li>Rd = Mem[Rn+Rm] (32 bit copy)</li> <li>อ่านค่าจากหน่วยความจำที่แอดเดรส Rn+Rm และเขียนค่านั้นในรีจิสเตอร์ Rd โดยที่ค่าในรีจิสเตอร์ Rn ไม่เปลี่ยนแปลง</li> </ul>

คำสั่ง MOV ย่อมาจากคำว่า MOVE ใช้สำหรับตั้งค่าเริ่มต้นให้รีจิสเตอร์ ค่าเริ่มต้นนี้เรียกว่า ค่า Immediate คำสั่ง LDR ย่อมาจากคำว่า Load Register ใช้สำหรับอ่านค่าตัวแปรในหน่วยความจำมาพักในรีจิสเตอร์

ตารางที่ 4.2: ตัวอย่างโปรแกรมภาษาแอสเซมบลีเพื่ออ่านค่าตัวแปรจากดาต้าเซ็กเมนต์

#	เลขบล	คำสั่ง	คอมเมนท์
1		LDR R1, =var1addr	@ load address of var1
2		LDR R2, =var2addr	@ load address of var2
3		LDR R1, [R1]	@ load value of var1
4		LDR R2, [R2]	@ load value of var2
5		SUB R0, R1, R2	@ R0 = R1 - R2

#### ตัวอย่างการเขียนโปรแกรม

x = (a + b) - c;

ผู้อ่านสามารถทำความเข้าใจ การทำงานได้จากคำอธิบายภายใต้คอลัมน์ คอมเมนท์ และคำอธิบายตามหมายเลขอรรถทัด (#) ดังนี้

ตารางที่ 4.3: ตัวอย่างโปรแกรมภาษาแอสเซมบลีเพื่อคำนวณ  $x = (a + b) - c$ 

#	เลขบล็อก	คำสั่ง	คอมเมนท์
1		LDR R4, =a	@ get address of variable a
2		LDR R0, [R4]	@ assign value of variable a to R0
3		LDR R4, =b	@ get address of variable b
4		LDR R1, [R4]	@ assign value of variable b to R1
5		ADD R3, R0, R1	@ a+b
6		LDR R4, =c	@ get address of variable c
7		LDR R2, [R4]	@ assign value of variable c to R2
8		SUB R3, R3, R2	@ $x = (a+b)-c$
9		LDR R4, =x	@ get address of variable x
10		STR R3, [R4]	@ store value of R3 to variable x

- คือ การโหลดตัวแหน่งของตัวแปร  $a$  ไปบรรจุใน R4
- คือ การโหลดค่าของตัวแปร  $a$  ไปบรรจุใน R0
- คือ การโหลดตัวแหน่งของตัวแปร  $b$  ไปบรรจุใน R4
- คือ การโหลดค่าของตัวแปร  $b$  ไปบรรจุใน R1
- คือ การคำนวณค่า  $a + b$  ไปบรรจุใน R3
- คือ การโหลดตัวแหน่งของตัวแปร  $c$  ไปบรรจุใน R4
- คือ การโหลดค่าของตัวแปร  $c$  ไปบรรจุใน R4
- คือ การคำนวณค่า  $(a+b) - c$  ไปบรรจุใน R3
- คือ การโหลดตัวแหน่งของตัวแปร  $x$  ไปบรรจุใน R4
- คือ การสตอร์ค่า  $((a+b)-c)$  ไปบรรจุในตัวแหน่งของตัวแปร  $x$

## 4.6 คำสั่งประมวลผลข้อมูล (Data Processing) จากรีจิสเตอร์

คำสั่งประมวลผล (Data Processing Instructions) จากรีจิสเตอร์ เชื่อมโยงกับคณิตศาสตร์ และข้อมูลชนิดต่างๆ คำสั่งในหัวข้อนี้จะจำกัดอยู่ที่ข้อมูลจำนวนเต็มเท่านั้นเพื่อให้เนื้อหาไม่ซับซ้อนจนเกินไป แบ่งเป็นคำสั่งประเกทย่อยๆ ดังนี้

- คำสั่งทางคณิตศาสตร์ เพื่อคำนวณเลขจำนวนเต็มชนิดมีและไม่มีเครื่องหมายด้วยวิธีพื้นฐาน
- คำสั่งเลื่อนบิต เพื่อเลื่อนบิตข้อมูลจากรีจิสเตอร์ไปทางซ้ายและขวาด้วยวงจรชิฟเตอร์
- คำสั่งทางคณิตศาสตร์และเลื่อนบิต เพื่อเลื่อนบิตไปทางซ้ายหรือขวาของจำนวนเต็มชนิดมีและไม่มีเครื่องหมาย
- คำสั่งทางตรรกศาสตร์ เพื่อคำนวณค่าทางตรรกศาสตร์ของเลขจำนวนเต็มชนิดไม่มีเครื่องหมาย

### 4.6.1 คำสั่งทางคณิตศาสตร์ (Arithmetic Instructions)

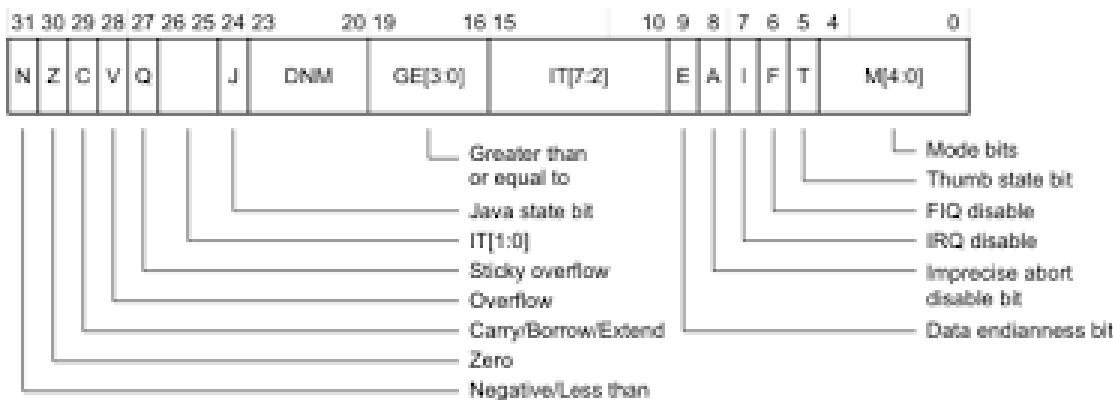
คำสั่ง บวก/ลบ และคูณสำหรับเลขจำนวนเต็มชนิดมีเครื่องหมายและไม่มีเครื่องหมาย กระทำโดยวงจรอาร์ดแวร์ เรียกว่า ALU (Arithmetic and Logic Unit) ซึ่งประกอบด้วยวงจรบวก/ลบเลข และคูณเลขตามหลักการคณิตศาสตร์คอมพิวเตอร์ในหัวข้อที่ 2.3 โปรแกรมเมอร์สามารถสั่งให้ชิปปุ๊ดโดยเฉพาะ ALU ด้วยคำสั่งตัวอย่างเหล่านี้

รูปแบบ	ความหมาย
ADD Rd, Rn, Rm	$Rd = Rn + Rm$
ADD Rd, Rn, #Imm	$Rd = Rn + \#Imm$
SUB Rd, Rn, Rm	$Rd = Rn - Rm$
SUB Rd, Rn, #Imm	$Rd = Rn - \#Imm$
RSB Rd, Rn, Rm	$Rd = Rm - Rn$ (Reverse Subtract)
RSB Rd, Rn, #Imm	$Rd = \#Imm - Rn$ (Reverse Subtract)
MUL Rd, Rn, Rm	$Rd = (Rn * Rm)$ (Only lower 32 bits)
UMULL Rhi, Rlo, Rn, Rm	$[Rhi\ Rlo] = (Rn * Rm)$ (Unsigned)
SMULL Rhi, Rlo, Rn, Rm	$[Rhi\ Rlo] = (Rn * Rm)$ (Signed)

การคำนวณเลขจำนวนเต็มชนิดมีเครื่องหมายด้วย ALU จำเป็นต้องตรวจสอบค่าบิตสถานะ NZCV ซึ่งตรงกับบิตที่ 31-28 ของรีจิสเตอร์สถานะปัจจุบัน (CPSR: Current Program Status Register) ในรูปที่ 4.4 ซึ่งเลขทั้ง 4 บิตจะเปลี่ยนแปลงตามผลลัพธ์ที่ ALU ที่ทำการคำนวณตามชนิดของข้อมูลและออฟโค้ดต่างๆ โดย

- บิต Z (Zero) เท่ากับ
  - 1 ใช้ตรวจสอบว่าผลลัพธ์มีค่าเท่ากับศูนย์
  - 0 ใช้ตรวจสอบว่าผลลัพธ์มีค่าไม่เท่ากับศูนย์
- บิต C (Carry) เท่ากับ
  - 1 บิตหนด (Carry bit)  $c_n$  เท่ากับ 1
  - 0 บิตหนด (Carry bit)  $c_n$  เท่ากับ 0
- บิต N (Negative) เท่ากับ
  - 1 ใช้ตรวจสอบว่าผลลัพธ์มีค่าน้อยกว่าศูนย์ หรือ ติดลบ
  - 0 ใช้ตรวจสอบว่าผลลัพธ์มีค่ามากกว่าศูนย์
- บิต V (oOverflow) เท่ากับ
  - 1 ใช้ตรวจสอบว่าผลการคำนวณเกิดโอเวอร์ไฟล์
  - 0 ใช้ตรวจสอบว่าผลการคำนวณไม่เกิดโอเวอร์ไฟล์

เพื่อสร้างความมั่นใจว่าผลลัพธ์ถูกต้อง (Valid) ตามตัวอย่างในหัวข้อที่ 2.3.1 และ 2.3.2



รูปที่ 4.4: รีจิสเตอร์สำหรับเก็บสถานะของชีพិឃុ ณ ปัจจุบัน (Current Program Status Register: CPSR) พร้อมรายละเอียดทั้ง 32 บิต ที่มา: [arm.com](http://arm.com)

#### 4.6.2 คำสั่งเลื่อนบิตข้อมูล (Shift Instructions)

คำสั่งเลื่อนบิตข้อมูลแบ่งเป็น การเลื่อนบิตทางซ้ายและทางขวา โดยวงจรบาร์เรลชิฟเตอร์ (Barrel Shifter) ในรูปที่ 4.5 การเลื่อนบิตไปทางขวาจำแนกได้เป็นการเลื่อนทางตรรกศาสตร์ (Logical Shift Right) และการเลื่อนทางคณิตศาสตร์ (Arithmetic Shift Right) ดังตารางต่อไปนี้

รูปแบบ	ความหมาย
LSL Rd, Rn, Rm	Rd = Rn « Rm (Logical Shift Left)
LSL Rd, Rn, #Imm	Rd = Rn « #Imm (Logical Shift Left)
LSR Rd, Rn, Rm	Rd = Rn » Rm (Logical Shift Right)
LSR Rd, Rn, #Imm	Rd = Rn » #Imm (Logical Shift Right)
ASL Rd, Rn, Rm	Rd = Rn « Rm (Arithmetic Shift Left)
ASL Rd, Rn, #Imm	Rd = Rn « #Imm (Arithmetic Shift Left)
ASR Rd, Rn, Rm	Rd = Rn » Rm (Arithmetic Shift Right)
ASR Rd, Rn, #Imm	Rd = Rn » #Imm (Arithmetic Shift Right)

หมายเหตุ สัญลักษณ์  $Rn \ll Rm$  คือ การเลื่อนบิตทั้งหมดของ  $Rn$  ไปทางซ้ายเป็นจำนวนบิตเท่ากับค่าใน  $Rm$  และสัญลักษณ์  $Rn \gg #Imm$  คือ การเลื่อนบิตทั้งหมดของ  $Rn$  ไปทางขวาเป็นจำนวนบิตเท่ากับค่าใน  $#Imm$

คำสั่งการเลื่อนบิตข้อมูล สามารถแบ่งเป็น

- ทางตรรกศาสตร์ (Logical Shift) นิยมใช้กับเลขจำนวนเต็มชนิดไม่มีเครื่องหมาย แบ่งเป็น
  - การเลื่อนบิตข้อมูลไปทางซ้าย วงจรจะป้อนศูนย์เข้ามาทางบิตขวาสุดแทนที่ข้อมูลเดิมที่ถูกเลื่อนไปทางซ้าย
  - การเลื่อนบิตข้อมูลไปทางขวา วงจรจะป้อนศูนย์เข้ามาทางบิตซ้ายสุดแทนที่ข้อมูลเดิมที่ถูกเลื่อนไปทางขวา
- ทางคณิตศาสตร์ (Arithmetic Shift) วงจรจะป้อนบิตทางซ้ายสุดด้วยบิตเครื่องหมายเข้ามาแทนที่ข้อมูลเดิมที่ถูกเลื่อนไปทางซ้ายหรือทางขวา เพื่อให้เครื่องหมายของข้อมูลในรีจิสเตอร์ไม่เปลี่ยนแปลง นิยมใช้กับเลขจำนวนเต็มชนิดมีเครื่องหมาย แบบ 2's Complement

### 4.6.3 คำสั่งทางคณิตศาสตร์และเลื่อนบิต (Arithmetic and Shift Instructions)

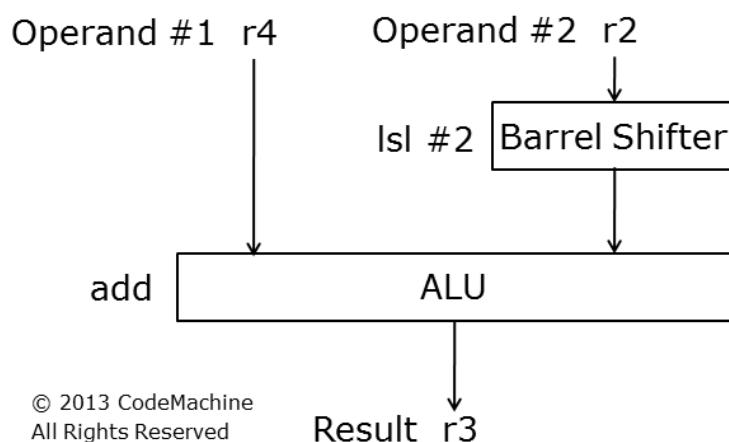
รูปแบบ	ความหมาย
ADD Rd, Rn, Rm LSL #shamt	$Rd = Rn + (Rm \ll \#shamt)$ ค่าที่เก็บใน Rm ไม่เปลี่ยนแปลง
ADD Rd, Rn, Rm LSR #shamt	$Rd = Rn + (Rm \gg \#shamt)$ ค่าที่เก็บใน Rm ไม่เปลี่ยนแปลง
ADD Rd, Rn, Rm ASR #shamt	$Rd = Rn + (Rm \gg \#shamt)$ (Signed) ค่าที่เก็บใน Rm ไม่เปลี่ยนแปลง

คำสั่งชนิดนี้เป็นจุดเด่นของชีพียู ARM เนื่องจากช่วยลดจำนวนคำสั่งและประหยัดเวลาในการประมวลผล แบ่งเป็นขั้นตอนอยู่ๆ 2 ขั้นตอน คือ

- อ่านข้อมูลจาก寄存器 Rm เพื่อป้อนให้กับวงจรบาร์เรลชิฟเตอร์ทำการเลื่อนบิตข้อมูลไปทางซ้ายหรือขวา เป็นจำนวนบิตตามค่า Shamt (Shift Amount) ซึ่งจัดเรียงในบิตที่ 7-11 ของคำสั่งคณิตศาสตร์ในรูปที่ 3.12
- นำผลลัพธ์ที่ได้จากการเลื่อนบิตไปคำนวณต่อตามคำสั่งหลัก หมายเหตุ ค่าที่ยังเก็บอยู่ใน寄存器 Rm จะไม่เปลี่ยนแปลง

ผู้อ่านสามารถทำความเข้าใจตัวอย่างการใช้คำสั่งคณิตศาสตร์และเลื่อนบิต add R3, R4, R2 lsl \$2 จากรูปต่อไปนี้

add r3, r4, r2, lsl #2; r3 = r4 + (r2 << 2)



รูปที่ 4.5: คำสั่ง  $r3 = r4 + (r2 \ll 2)$  โดยหากค่า r4 กับค่า r2 หลังจากเลื่อนบิตไปทางซ้ายจำนวน 2 บิตแล้วจะเก็บผลลัพธ์ใน r3 ที่มา: [CodeMachine.com](http://CodeMachine.com) หมายเหตุ ค่าที่ยังเก็บอยู่ใน寄存器 r2 จะไม่เปลี่ยนแปลง

โดยค่าใน R4 เป็นตัวตั้งหรือโอเปอร์เรนด์ (Operand) ที่ 1 ค่าใน R2 เป็นตัวตั้งหรือโอเปอร์เรนด์ที่ 2 และ Shamt มีค่าเท่ากับ  $2_{10}$  หรือ 00010<sub>2</sub> ความยาว 5 บิต ความหมาย คือ การนำ R2 เลื่อนบิตไปทางซ้ายจำนวน 2 บิต บวกกับ R4 แล้วจึงเก็บผลลัพธ์ใน R3 หรือเท่ากับ  $R3 = R4 + (R2 \times 4)$  โดยค่าที่เก็บใน寄存器 R2 จะไม่เปลี่ยนแปลง

#### 4.6.4 คำสั่งทางตรรกศาสตร์ (Logical Instructions)

ตัวอย่างประโยคภาษา C การคำนวณด้วยการกระทำทางตรรกศาสตร์ โปรแกรมเมอร์นิยมกระทำกับข้อมูลชนิดไม่มีเครื่องหมาย เพื่อคำนวณข้อมูลทีละบิต แต่การประมวลผลเกิดขึ้นพร้อมกันทุกบิต โดยบิตข้อมูลตัวตั้งจะกระทำกับบิตที่ตรงกันของตัวกระทำเท่านั้น (Bitwise Operation) ยกตัวอย่างเช่น

```
unsigned int a = 0xfffff0000;
unsigned int b = 0x0000ffff;
unsigned int c;
c = a & b; /* c = 0x00000000 AND */
c = a | b; /* c = 0xffffffff OR */
c = !b;      /* c = 0xfffff0000 INVERSE */
c = a ^ b; /* c = 0xffffffff Ex-OR */
```

ชีพีย์ ARM รองรับการทำงานด้วยคำสั่งแอลซีเมบลีเหล่านี้

รูปแบบ	ความหมาย
AND Rd, Rn, Rm	Rd = Rn & Rm (bitwise AND)
AND Rd, Rn, #Imm	Rd = Rn & #Imm (bitwise AND)
ORR Rd, Rn, Rm	Rd = Rn   Rm (bitwise OR)
ORR Rd, Rn, #Imm	Rd = Rn   #Imm (bitwise OR)
MVN Rd, Rm	Rd = $\overline{Rm}$ (bitwise Inverse)
MVN Rd, #Imm	Rd = $\overline{\#Imm}$ (bitwise Inverse)
EOR Rd, Rn, Rm	Rd = Rn $\oplus$ Rm (bitwise XOR)
EOR Rd, Rn, #Imm	Rd = Rn $\oplus$ #Imm (bitwise XOR)

หมายเหตุ ตัวเลขคงที่ #Imm จะถูกขยายเครื่องหมาย (Sign Extend) หัวข้อที่ [2.2.2](#) ให้กลายเป็นข้อมูลขนาด 32 บิต เพื่อให้สอดคล้องกับข้อมูลขนาด 32 บิตในรีจิสเตอร์ Rn ผู้อ่านควรทราบหัวข้อพีซคณิตบูลลิน (Boolean Algebra) ในวิชาออกแบบวงจรดิจิทัล โดยเฉพาะเรื่องตารางความจริง (Truth Table) ของวงจรเกตชนิดต่างๆ

## 4.7 คำสั่งควบคุมการทำงาน (Control Instructions)

คำสั่งควบคุมการทำงาน (Control Instructions) ในภาษาสูงแบ่งเป็น ประโยชน์การตัดสินใจ เช่น ประโยชน์ IF, IF-ELSE, Switch-Case และประโยชน์การวนรอบชนิดต่างๆ เช่น FOR, WHILE, DO-WHILE เป็นต้น โครงสร้างคำสั่งภาษาแอสเซมบลีของ ARM ที่รองรับประโยชน์เหล่านี้ ประกอบด้วย

- คำสั่ง CMP (Compare) ทำหน้าที่เปรียบเทียบระหว่างค่าเรจิสเตอร์กับค่าคงที่ หรือระหว่างค่าเรจิสเตอร์ 2 ตัว โปรแกรมเมอร์สามารถใช้คำสั่งนี้ได้ 2 รูปแบบตามตารางต่อไปนี้

รูปแบบ	ความหมาย
CMP Rn, Rm	$NZCV \leftarrow Rn - Rm$
CMP Rn, #Imm	$NZCV \leftarrow Rn - #Imm$

ค่าบิต NZCV ในเรจิสเตอร์ CPSR (หัวข้อที่ 4.6.1) คือ ผลลัพธ์ที่ได้จากการคำสั่ง CMP โดยคำสั่งนี้ทำการเปรียบเทียบเลขจำนวนเต็มสองค่า ( $Rn - Rm$ ) หรือ ( $Rn - #Imm$ )

- คำสั่ง B (Branch) จะย้ายการทำงานไปยังคำสั่งปีழມายตามเงื่อนไข (Condition) ที่ได้จากการตรวจสอบบิต NZCV ที่เกิดจากคำสั่ง CMP ในเชิงเทคนิค PC จะเปลี่ยนค่าเป็นค่าแอดเดรสที่ตรงกับเลbel เป้ำหมาด (Target Label: PC = address(label)) เพื่อเพทช์คำสั่งที่ตำแหน่งนั้น ตามรูปแบบเงื่อนไขต่างๆ ในตารางต่อไปนี้

รูปแบบ	ความหมาย
B label	pc = address(label) เพทช์คำสั่งที่ตามหลัง label (อย่างไม่มีเงื่อนไข)
BEQ label	pc = address(label) เพทช์คำสั่งที่ตามหลัง label หากผลลัพธ์การเปรียบเทียบเท่ากัน
BNE label	pc = address(label) เพทช์คำสั่งที่ตามหลัง label หากผลลัพธ์การเปรียบเทียบไม่เท่ากัน
BGT label	pc = address(label) เพทช์คำสั่งที่ตามหลัง label หากผลลัพธ์การเปรียบเทียบมากกว่า
BLT label	pc = address(label) เพทช์คำสั่งที่ตามหลัง label หากผลลัพธ์การเปรียบเทียบน้อยกว่า
BGE label	pc = address(label) เพทช์คำสั่งที่ตามหลัง label หากผลลัพธ์การเปรียบเทียบมากกว่าหรือเท่ากับ
BLE label	pc = address(label) เพทช์คำสั่งที่ตามหลัง label หากผลลัพธ์การเปรียบเทียบน้อยกว่าหรือเท่ากับ

คำสั่ง B (branch) ตรงกับคำสั่ง Jump ในภาษาแอสเซมบลีอื่นๆ หรือ ประโยชน์ go-to ในภาษา C/C++ คำสั่ง B ทั้งชนิดมีและไม่มีเงื่อนไขจะทำให้แอดเดรสในเรจิสเตอร์ PC เปลี่ยนค่าเป็น หมายเลขไปต์ หรือ แอดเดรส หรือ ตำแหน่งของ label ว่าจะดิจิทัลในชีพิญญาณารถตรวจสอบผลลัพธ์ของเงื่อนไขต่างๆ เหล่านี้ได้จาก บิต NZCV รวมทั้งหมด 15 แบบ ดังนี้

1. **EQ** (Equal): Z Set คือการตรวจสอบว่า  $Z=1$  หรือไม่ นั่นคือ ผลการลบเท่ากับ 0
2. **NE** (Not Equal): Z Not Set คือการตรวจสอบว่า  $Z=0$  หรือไม่ นั่นคือ ผลการลบไม่เท่ากับ 0
3. **CS** (Carry Set): Unsigned Higher คือ เงื่อนไขสำหรับข้อมูลชนิดไม่มีเครื่องหมายว่ามากกว่า โดยตรวจสอบว่า Carry =1
4. **CC** (Carry Clear): Unsigned Lower คือ เงื่อนไขสำหรับข้อมูลชนิดไม่มีเครื่องหมายว่าน้อยกว่า โดยตรวจสอบว่า Carry = 0
5. **MI** (Minus): Negative Set คือ การตรวจสอบว่า  $N=1$  หรือไม่ นั่นคือ ผลการลบน้อยกว่า 0
6. **PL** (Plus or Zero): Negative Not Set คือ การตรวจสอบว่า  $N=0$  หรือไม่ นั่นคือ ผลการลบมากกว่า หรือเท่ากับ 0
7. **VS** (Overflow Set): เกิด Overflow ขึ้น คือ การตรวจสอบว่า  $V=1$  หรือไม่ นั่นคือ เกิดโอเวอร์โฟล์
8. **VC** (Overflow Clear): ไม่เกิด Overflow คือ การตรวจสอบว่า  $V=0$  หรือไม่ นั่นคือ ไม่เกิดโอเวอร์โฟล์
9. **HI** (Unsigned Higher): คือ เงื่อนไขสำหรับข้อมูลชนิดไม่มีเครื่องหมายว่ามากกว่า โดยตรวจสอบว่า  $C=1$  หรือ  $Z=0$
10. **LS** (Unsigned Lower or Same): คือ เงื่อนไขสำหรับข้อมูลชนิดไม่มีเครื่องหมายว่าน้อยกว่าหรือเท่ากัน โดยตรวจสอบว่า Carry=0 or Zero=1
11. **GE** (Greater Than or Equal): Negative == Overflow คือ การตรวจสอบว่า  $N=V$  หรือไม่ นั่นคือ ผลการลบมากกว่าหรือเท่ากับ 0
12. **LT** (Less Than): Negative != Overflow คือ การตรวจสอบว่า  $N!=V$  หรือไม่ นั่นคือ ผลการลบน้อยกว่า 0
13. **GT** (Greater Than): !Zero && Negative = Overflow คือ การตรวจสอบว่า  $Z=0$  และ  $N=V$  หรือไม่ นั่นคือ ผลการลบมากกว่า 0
14. **LE** (Less Than or Equal): Zero && Negative != Overflow คือ การตรวจสอบว่า  $Z=1$  และ  $N!=V$  หรือไม่ นั่นคือ ผลการลบน้อยกว่าหรือเท่ากับ 0
15. **AL** (Always): เงื่อนไขเป็นจริงเสมอ ซึ่งได้เกริ่นในหัวข้อที่ [3.2.5](#)

### 4.7.1 การตัดสินใจ IF

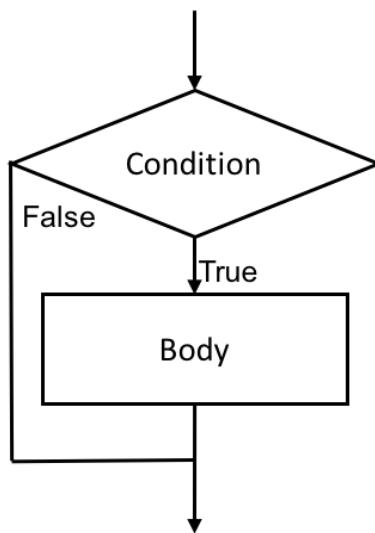
การตัดสินใจ IF ขึ้นอยู่กับเงื่อนไขของประโภค IF ว่าผลลัพธ์ที่ได้จะเป็นจริงหรือเท็จ

- หากจริง ซึ่งปฎิบัติตามประโภคคำสั่งที่โปรแกรมเมอร์ต้องการ
- หากเท็จ ซึ่งปฎิบัติตามประโภคคำสั่งที่อยู่ถัดจากประโภค IF ไป

การตัดสินใจพื้นฐานที่เข้าใจง่าย เพราะมีความซับซ้อนน้อยหมายความว่ารับผู้ฝึกเขียนโปรแกรม และสามารถคาด忖ว่าด้วยวิธีการที่ได้ตามรูปที่ 4.6

ตัวอย่างการเขียนโปรแกรม ประโภค IF ในภาษา C/C++

```
if ((a+b) > c) {
    x=x+y; /* Body */
}
```



รูปที่ 4.6: โฟล์ชาร์ตการทำงานของประโภค IF

จากประโภค IF นี้ เปรียบเทียบผลบวกของ  $a$  และ  $b$  มากกว่าค่าของ  $c$  หรือไม่ หากจริง  $x = x + y$  หากเท็จ  $x$  ไม่เปลี่ยนแปลง ผู้อ่านสามารถเขียนเป็นชุดคำสั่งภาษาแอสเซมบลีของ ARM ได้ในตารางที่ 4.4

ผู้อ่านสามารถทำงานเข้าใจการทำงานได้จากคำอธิบายภายใต้คอลัมน์ คอมเมนท์ และคำอธิบายตามหมายเลขบรรทัด (#) ดังนี้

- คือ การโหลดตำแหน่ง (แอดเดรส) ของตัวแปร  $a$  ไปบรรจุใน R4
- คือ การโหลดค่าของตัวแปร  $a$  ไปบรรจุใน R0
- คือ การโหลดตำแหน่ง (แอดเดรส) ของตัวแปร  $b$  ไปบรรจุใน R4
- คือ การโหลดค่าของตัวแปร  $b$  ไปบรรจุใน R1
- คือ การคำนวณค่า  $a + b$  ไปบรรจุใน R3
- คือ การโหลดตำแหน่ง (แอดเดรส) ของตัวแปร  $c$  ไปบรรจุใน R4

7. คือ การโหลดค่าของตัวแปร  $c$  ไปบรรจุใน R4
8. คือ การเปรียบเทียบค่าของ R2 และ R3
9. คือ หากเงื่อนไข Less Than or Equal (LE) เป็นจริง แล้วเดรสในรีจิสเตอร์ PC จะเปลี่ยนเป็นตำแหน่งของเลbelชื่อ exit หรือ  $PC = \text{address(exit)}$  หากไม่เป็นจริง แล้วเดรสในรีจิสเตอร์ PC จะเปลี่ยนเป็นตำแหน่งของคำสั่งถัดไป หรือ  $PC=PC+4$
10. คือ การโหลดตำแหน่ง (แอดเดรส) ของตัวแปร  $x$  ไปบรรจุใน R4
11. คือ การโหลดค่าของตัวแปร  $x$  ไปบรรจุใน R5
12. คือ การโหลดตำแหน่ง (แอดเดรส) ของตัวแปร  $y$  ไปบรรจุใน R4
13. คือ การโหลดค่าของตัวแปร  $y$  ไปบรรจุใน R6
14. คือ การคำนวนค่า  $a + b$  ไปบรรจุใน R5
15. คือ การโหลดตำแหน่ง (แอดเดรส) ของตัวแปร  $x$  ไปบรรจุใน R4
16. คือ การสโตรค่าของ R5 ไปบรรจุตำแหน่ง (แอดเดรส) ของตัวแปร  $x$
17. คือ คำสั่งที่ขึ้นต้นด้วยเลbel exit

ตารางที่ 4.4: ตัวอย่างโปรแกรมตามประโยชน์ของเงื่อนไข if

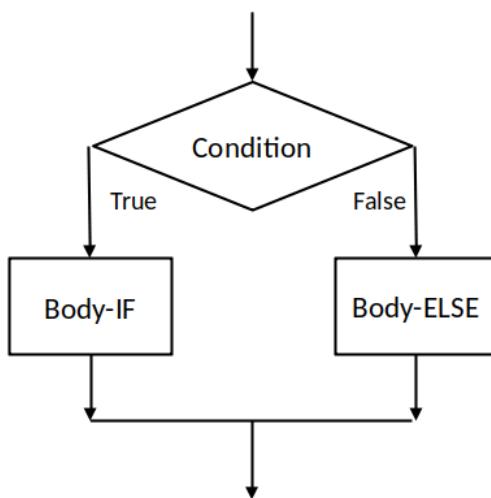
#	เลbel	คำสั่ง	คอมเมนท์
1		LDR R4, =a	@ get address of variable a
2		LDR R0, [R4]	@ get value of variable a
3		LDR R4, =b	@ get address of variable b
4		LDR R1, [R4]	@ get value of variable b
5		ADD R3, R0, R1	@ compute a+b
6		LDR R4, =c	@ get address of variable c
7		LDR R2, [R4]	@ get value of variable c
8		CMP R3, R2	@ compare (a+b) with c
9		BLE exit	@ jump to exit if R3 <= R2
10		LDR R4, =x	@ get address of variable x
11		LDR R5, [R4]	@ get value of variable x
12		LDR R4, =y	@ get address of variable y
13		LDR R6, [R4]	@ get value of variable y
14		ADD R5, R5, R6	@ x += y
15		LDR R4, =x	@ get address of variable x
16		STR R5, [R4]	@ store value of variable x
17	exit:	...	@ exit label

### 4.7.2 การตัดสินใจ IF-ELSE

การตัดสินใจ IF-ELSE ขึ้นอยู่กับเงื่อนไขของประโภค IF ว่าผลลัพธ์ที่ได้จะเป็นจริงหรือเท็จ

- หากจริง ซึ่งจะปฏิบัติตามประโภคคำสั่งที่โปรแกรมเมอร์ต้องการ เมื่อแล้วเสร็จซึ่งจะข้ามประโภคที่ไม่เกี่ยวข้อง เพื่อประมวลผลคำสั่งถัดไป
- หากเท็จ ซึ่งจะทำการคำสั่งที่มีเลbel ELSE นำหน้า เมื่อแล้วเสร็จจะประมวลผลประโภคอื่นถัดไป

การตัดสินใจพื้นฐานที่ซับซ้อนเพิ่มขึ้นหมายความว่ารับผู้ฝึกเขียนโปรแกรม และสามารถวางแผนฟลุ๊ชาร์ตตามรูปที่ 4.7 โดยคำสั่งภายในกล่อง Body-IF คือ คำสั่งที่ต้องการให้ประมวลผลหากเงื่อนไขเป็นจริง และคำสั่งภายในกล่อง Body-ELSE คือ คำสั่งที่ต้องการให้ประมวลผลหากเงื่อนไขเป็นเท็จ



รูปที่ 4.7: ฟลุ๊ชาร์ตการทำงานของประโภค IF-ELSE

ตัวอย่างการเขียนโปรแกรม ประโภค IF-ELSE ในภาษา C/C++

```

if ((a+b) >c) {
    x+=y; /* Body-IF */
}

else {
    x-=y; /* Body-ELSE */
}
  
```

จากประโภค IF-ELSE นี้ ผู้อ่านสามารถเปลี่ยนเป็นชุดคำสั่งภาษาแอลซีเมบลีของ ARM ได้ในตารางที่ 4.5 และทำความเข้าใจการทำงานได้จากคำอธิบายภายใต้คอลัมน์ คอมเมนท์ และคำอธิบายตามหมายเลขบรรทัด (#) ดังนี้

ตารางที่ 4.5: ตัวอย่างโปรแกรมตามประโยคเงื่อนไข if-else

#	เลbel	คำสั่ง	คอมเมนท์
1		LDR R4, =a	@ get address of variable a
2		LDR R0, [R4]	@ get value of variable a
3		LDR R4, =b	@ get address of variable b
4		LDR R1, [R4]	@ get value of variable b
5		ADD R3, R0, R1	@ compute a+b
6		LDR R4, =c	@ get address of variable c
7		LDR R2, [R4]	@ get value of variable c
8		CMP R3, R2	@ compare (a+b) with c
9		BLE else	@ jump to else if R3 <= R2
10		LDR R4, =x	@ get address of variable x
11		LDR R5, [R4]	@ get value of variable x
12		LDR R4, =y	@ get address of variable y
13		LDR R6, [R4]	@ get value of variable y
14		ADD R5, R5, R6	@ x += y
15		LDR R4, =x	@ get address of variable x
16		STR R5, [R4]	@ store value of variable x
17		B exit	@ jump to exit label
18	else:	LDR R4, =x	@ get address of variable x
19		LDR R5, [R4]	@ get value of variable x
20		LDR R4, =y	@ get address of variable y
21		LDR R6, [R4]	@ get value of variable y
22		SUB R5, R5, R6	@ x -= y
23		LDR R4, =x	@ get address of variable x
24		STR R5, [R4]	@ store value of variable x
25	exit:	...	@ label exit

- คือ การโหลดตำแหน่ง (แอดเดรส) ของตัวแปร *a* ไปบรรจุใน R4
- คือ การโหลดค่าของตัวแปร *a* ไปบรรจุใน R0
- คือ การโหลดตำแหน่ง (แอดเดรส) ของตัวแปร *b* ไปบรรจุใน R4
- คือ การโหลดค่าของตัวแปร *b* ไปบรรจุใน R1
- คือ การคำนวณค่า *a + b* ไปบรรจุใน R3
- คือ การโหลดตำแหน่ง (แอดเดรส) ของตัวแปร *c* ไปบรรจุใน R4
- คือ การโหลดค่าของตัวแปร *c* ไปบรรจุใน R4
- คือ การเปรียบเทียบค่าของ R2 และ R3
- คือ หากเงื่อนไข Less Than or Equal (LE) เป็นจริง ซึ่งจะกระโดดไปทำงานคำสั่งที่ขึ้นต้นด้วย exit หากไม่เป็นจริง ซึ่งจะทำงานประโยคที่ 10 ต่อไป
- คือ การโหลดตำแหน่ง (แอดเดรส) ของตัวแปร *x* ไปบรรจุใน R4

11. คือ การโหลดค่าของตัวแปร  $x$  ไปบรรจุใน R5
12. คือ การโหลดคำແໜ່ງ (ແອດເດຣສ) ของตัวแปร  $y$  ไปบรรจุใน R4
13. คือ การโหลดค่าของตัวแปร  $y$  ไปบรรจุใน R6
14. คือ การคำนวณค่า  $x = x + y$  ไปบรรจุใน R5
15. คือ การโหลดคำແໜ່ງ (ແອດເດຣສ) ของตัวแปร  $x$  ไปบรรจุใน R4
16. คือ การສຕ້ວລົກຄ່າຂອງ R5 ไปປະຈຸທີ່ຕຳແໜ່ງ (ແອດເດຣສ) ของตัวແປຣ  $x$
17. คือ การບັງຄັບໃຫ້ປີ່ພິ່ງກະຮະໂດດໄປທຳນານຄຳສັ່ງທີ່ເຂື້ນຕົ້ນດ້ວຍເລເບລ exit
18. คือ การโหลดคำແໜ່ງ (ແອດເດຣສ) ของตัวແປຣ  $x$  ไปบรรจุใน R4
19. คือ การโหลดค่าของตัวແປຣ  $x$  ไปบรรจุใน R5
20. คือ การโหลดคำແໜ່ງ (ແອດເດຣສ) ของตัวແປຣ  $y$  ไปบรรจุใน R4
21. คือ การโหลดค่าของตัวແປຣ  $y$  ไปบรรจุใน R6
22. คือ การคำนวณค่า  $x = x - y$  ไปบรรจุใน R5
23. คือ การโหลดคำແໜ່ງ (ແອດເດຣສ) ของตัวແປຣ  $x$  ไปบรรจุใน R4
24. คือ การສຕ້ວລົກຄ່າຂອງ R5 ไปປະຈຸທີ່ຕຳແໜ່ງ (ແອດເດຣສ) ของตัวແປຣ  $x$
25. คือ ຄຳສັ່ງທີ່ເຂື້ນຕົ້ນດ້ວຍເລເບລ exit

### 4.7.3 การวนรอบชนิด FOR

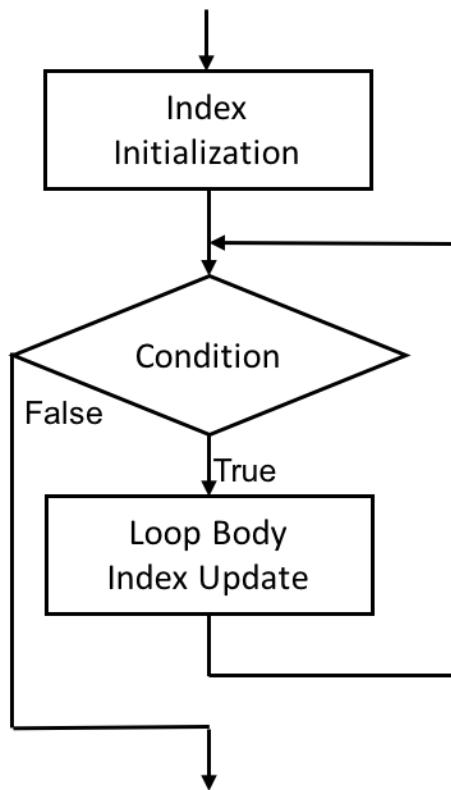
การพัฒนาโปรแกรมคอมพิวเตอร์ในอดีตมุ่งเน้นที่การคำนวณแก้ปัญหาทางคณิตศาสตร์ ยกตัวอย่างเช่น

$$x = \sum_{i=1}^{10} i \quad (4.1)$$

สมการที่ (4.1) คือ การบวกเลขตั้งแต่ค่า  $i = 1$  จนถึง  $i = 10$  ในรูปแบบทางคณิตศาสตร์นี้ โปรแกรมเมอร์สามารถใช้ภาษา C เขียนการบวกนี้ในรูปของประโยควนรอบชนิด for ซึ่งเข้าใจได้ง่ายที่สุด ดังนี้

```
x=0;
for (i=1; i<=10; i=i+1) {
    x=x+i; /* Body */
}
```

ซอฟต์แวร์โค้ดนี้ทำการตั้งค่าเริ่มต้นให้ตัวแปร  $x$  เท่ากับ 0 ให้ตัวแปร  $i$  เท่ากับ 1 และจึงเพิ่มค่า  $x$  ด้วยค่า  $i$  แล้ววนกลับมาทำประโยคนี้อีก ในขณะที่  $i$  จะถูกเพิ่มค่าเป็น  $i+1$  เช่นกัน ดังนั้น ประโยค  $x = x+i$  จะทำงานทั้งหมด 10 รอบตามเงื่อนไข  $i \leq 10$



รูปที่ 4.8: โฟล์ชาร์ตการทำงานของประโยคการวนรอบชนิด FOR

เราเรียก  $i$  ว่าเป็นตัวแปรนับรอบ (Index) เนื่องจากเป็นตัวแปรที่ใช้นับเลขรอบ ประโยค  $i=1$  เรียกว่า Index Initialization คือ การตั้งค่าเริ่มต้นให้กับตัวแปรนับรอบ ซึ่งเป็นองค์ประกอบหนึ่งของประโยค for ประโยค  $x=x+i$  เรียกว่า Loop Body ตามโฟล์ชาร์ตในรูปที่ 4.8 ประโยคเงื่อนไข  $i \leq 10$  เรียกว่า Condition เพื่อตรวจสอบว่าเป็นจริงหรือเท็จ เมื่อ  $i$  มีค่าเท่ากับ 1 ถึง 10 เงื่อนไขจะให้ผลลัพธ์เป็นจริง (True) เมื่อ  $i$  มีค่าเท่ากับ 11 ผลลัพธ์จะกลายเป็นเท็จ (False) และจะไม่เกิดการวนรอบ เพื่อทำงานประโยคที่ต่อจากประโยค for ประโยค

$i=i+1$  เรียกว่า Index Update คือ การปรับเปลี่ยนค่าตัวแปรนับรอบให้สอดคล้องกับเงื่อนไข ตามที่กล่าวมาข้างต้น

ตารางที่ 4.6: ตัวอย่างโปรแกรมประโภควนรอบ For ตามสมการที่ (4.1)

#	เลbel	คำสั่ง	คอมเมนท์
1		...	@ Initialize R0=0
2		...	@ Initialize R1=1
3	for:	CMP R1, #10	@ Compare R1 with 10
4		BGT end	@ if greater than goto end
5		ADD R0, R0, R1	@ else R0 = R0 + R1
6		ADD R1, R1, #1	@ Increment R1 by 1
7		B for	@ Branch back to Label for
8	end:	...	@ End of for loop

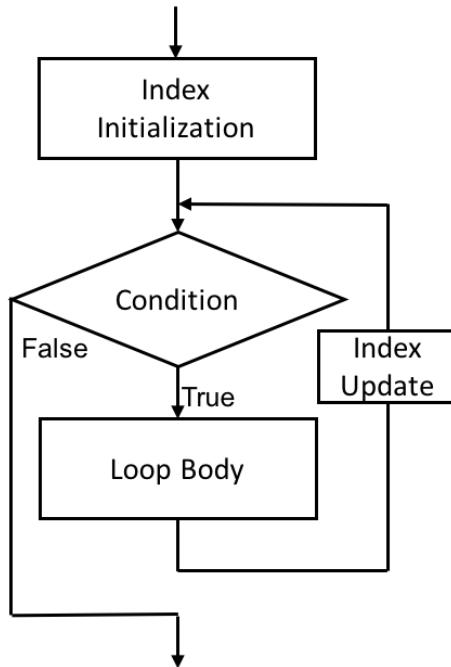
ผู้อ่านสามารถทำความเข้าใจการทำงานได้จากคำอธิบายภายใต้คอลัมน์ คอมเมนท์ และคำอธิบายตามหมายเลขบรรทัด (#) ดังนี้

- คือ การตั้งค่าเริ่มต้นให้กับ  $R0 = 0$
- คือ การตั้งค่าเริ่มต้นให้กับ  $R1 = 1$
- คือ คำสั่งที่ขึ้นต้นด้วยเลเบล for: เพื่อทำการเปรียบเทียบ  $R1$  กับค่าคงที่  $10_{10}$
- หากเงื่อนไข Greater Than (GT) เป็นจริง ซึ่งจะกระโดดไปทำงานคำสั่งที่ขึ้นต้นด้วย end หากไม่เป็นจริง ซึ่งจะทำงานประโภคที่ 5 ต่อไป
- คือ การคำนวณค่า  $R0 + R1$  ไปบรรจุใน  $R0$
- คือ การคำนวณค่า  $R1 + 1$  ไปบรรจุใน  $R1$
- คือ การบังคับให้ซีพียูกระโดดไปทำงานคำสั่งที่ขึ้นต้นด้วยเลเบล for
- คือ คำสั่งที่ขึ้นต้นด้วยเลเบล end

ตัวอย่างนี้เป็นการวนรอบชนิด For ในรูปที่ 4.8 การตั้งค่าเริ่มต้น (Initialization) ให้กับตัวแปรนับรอบ (Index) หลังจากนั้น ซีพียูจะตรวจสอบเงื่อนไข Condition ของการวนรอบ หากเป็นจริง ซีพียูจะทำงานตามคำสั่งจำนวนหนึ่ง ที่เรียกว่า Body เมื่อแล้วเสร็จ ซีพียูจะเพิ่ม (Increment) หรือ ลด (Decrement) ค่าตัวแปรลูป (Loop Index) และซีพียูจะกลับไปทำงานในบรรทัดที่มีประโภคเงื่อนไขซ้ำแล้วซ้ำอีก จนเงื่อนไขเป็นเท็จ (False) การวนรอบชนิด FOR จะสิ้นสุดการทำงาน และจึงย้ายการทำงานไปคำสั่งอื่นภายนอก

#### 4.7.4 การวนรอบชนิด WHILE

การวนรอบชนิด WHILE คล้ายกับการวนรอบชนิด FOR คือ การวนรอบจะเกิดขึ้นอยู่กับเงื่อนไขของประโภค WHILE ว่าผลลัพธ์ที่ได้จะเป็นจริงหรือเท็จ หากจริง ซีพียูจะปฏิบัติตามประโภคคำสั่งที่โปรแกรมเมอร์ต้องการแล้ววนกลับไปตรวจสอบเงื่อนไขอีก หากเท็จ ซีพียูจะทำคำสั่งอื่นๆ ต่อไป ในรูปที่ 4.9



รูปที่ 4.9: โฟล์ชาร์ตการทำงานของประโภคการวนรอบชนิด WHILE

สมการที่ (4.1) สามารถเขียนด้วยการวนรอบชนิด WHILE ดังนี้

```

i=1;
x=0;
while (i<=10) {
    x=x+i; /* Body */
    i++; /* Index Update */
}
  
```

ตัวอย่างนี้ทำงานเหมือนกับประโภค FOR เกือบทุกประการ ความแตกต่าง คือ การตั้งค่าตัวแปรเริ่มต้น การตรวจสอบเงื่อนไขในวงเล็บของประโภค WHILE และการปรับเปลี่ยนตัวแปร  $i$  ภายในลูป ดังนั้น ตัวอย่างการเขียนโปรแกรม ในตารางที่ 4.7 จึงมีความใกล้เคียงกันมากกับประโภค FOR

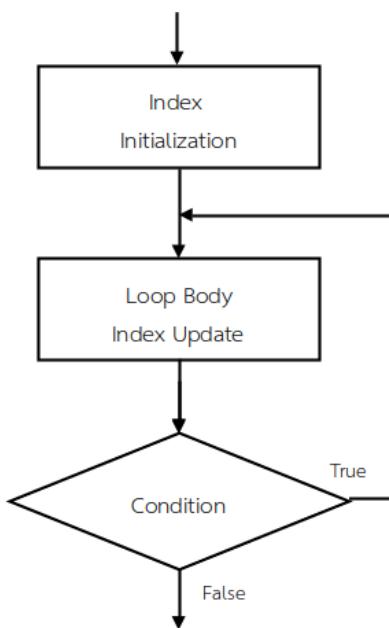
ตารางที่ 4.7: ตัวอย่างโปรแกรมตามประโยคุนรอบซnid WHILE ตามสมการที่ (4.1)

#	เลbel	คำสั่ง	คอมเมนท์
1		...	@ Initialize R0=0
2		...	@ Initialize R1=1
3	while:	CMP R1, #10	@ Compare R1 with 10
4		BGT end	@ if greater than goto end
5		ADD R0, R0, R1	@ R0 = R0+R1
6		ADD R1, R1, #1	@ Increment R1 by 1
7		B while	@ Branch back to Label while
8	end:	...	@ End of while loop
9		...	@

ผู้อ่านสามารถทำความเข้าใจการทำงานได้จากคำอธิบายภายใต้คอลัมน์ คอมเมนท์ และคำอธิบายตามหมายเลขบรรทัด (#) ดังนี้

1. คือ การตั้งค่าเริ่มต้นให้กับ  $R0 = 0$
2. คือ การตั้งค่าเริ่มต้นให้กับ  $R1 = 1$
3. คือ คำสั่งที่ขึ้นต้นด้วยเลเบล while เพื่อทำการเปรียบเทียบ  $R1$  กับค่าคงที่  $10_{10}$
4. หากเงื่อนไข Greater Than (GT) เป็นจริง ซึ่งจะกระโดดไปทำงานคำสั่งที่ขึ้นต้นด้วยเลเบล end หากไม่เป็นจริง ซึ่งจะทำงานประโยคที่ 5 ต่อไป
5. คือ การคำนวณค่า  $R0 + R1$  ไปบรรจุใน  $R0$
6. คือ การคำนวณค่า  $R1 + 1$  ไปบรรจุใน  $R1$
7. คือ การบังคับให้ซีพียูกระโดดไปทำงานคำสั่งที่ขึ้นต้นด้วยเลเบล for
8. คือ คำสั่งที่ขึ้นต้นด้วยเลเบล end

#### 4.7.5 การวนรอบชนิด DO-WHILE



รูปที่ 4.10: โฟล์ชาร์ตการทำงานของประโยชน์การวนรอบชนิด DO-WHILE

การวนรอบอีกชนิดคือ การวนรอบ DO-WHILE มีโครงสร้างการทำงานตามโฟล์ชาร์ตในรูปที่ 4.10 การวนรอบ DO-WHILE แตกต่างกับการวนรอบ WHILE คือ ซึ่งจะปฏิบัติตามประโยชน์คำสั่งในลูป (Loop Body) อย่างน้อย 1 รอบ หลังจากนั้น ซึ่งจะตรวจสอบเงื่อนไขของประโยชน์ WHILE ว่าผลลัพธ์ที่ได้จะเป็นจริงหรือเท็จ หากจริง ซึ่งจะวนกลับไปปฏิบัติตามประโยชน์อีก แล้วจึงตรวจสอบเงื่อนไข หากเท็จ ซึ่งจะทำคำสั่งอื่นๆ ต่อไป สมการที่ (4.1) สามารถเขียนด้วยการวนรอบชนิด DO-WHILE ดังนี้

```
i=1;
x=0;
do {
    x=x+i; /* Body */
    i++; /* Index Update */
} while (i<=10);
```

และสามารถเขียนด้วยภาษาแอสเซมบลีได้ในตารางที่ 4.8

ผู้อ่านสามารถทำความเข้าใจการทำงานได้จากคำอธิบายภายใต้คอลัมน์ คอมเมนท์ และคำอธิบายตามหมายเลขอรรถทัด (#) ดังนี้

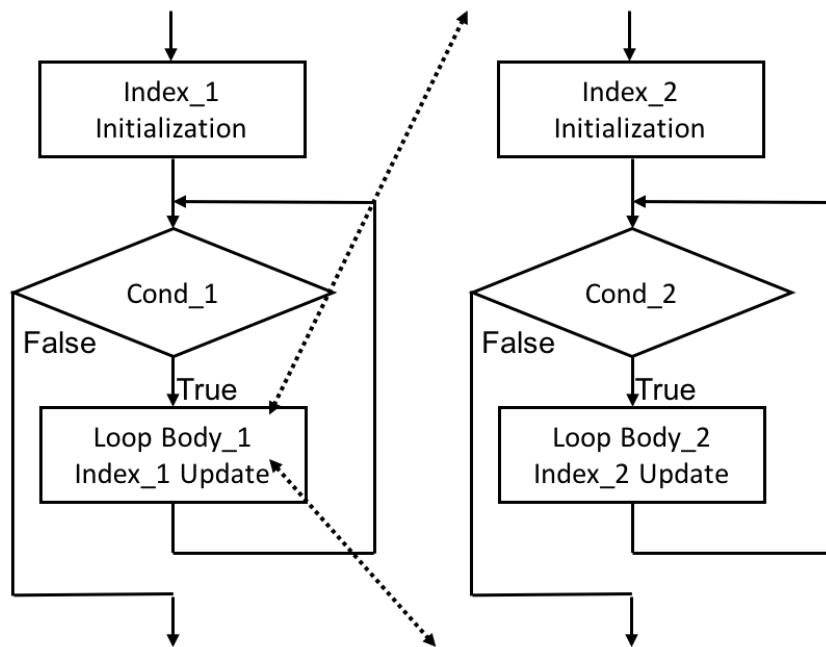
1. คือ การตั้งค่าเริ่มต้นให้กับ R0 = 0
2. คือ การตั้งค่าเริ่มต้นให้กับ R1 = 0
3. คือ คำสั่งที่ขึ้นต้นด้วยเลขเบต do เพื่อกำหนดค่า R0 + R1 ไปบรรจุใน R0
4. คือ การคำนวนค่า R1 + 1 ไปบรรจุใน R1
5. คือ การเปรียบเทียบ R1 กับค่าคงที่ 10<sub>10</sub>

ตารางที่ 4.8: ตัวอย่างโปรแกรมตามประโยคุณรอบชนิด DO-WHILE ตามสมการที่ (4.1)

#	เลbel	คำสั่ง	คอมเมนท์
1		...	@ Initialize R0 = 0
2		...	@ Initialize R1 = 0
3	do:	ADD R0, R0, R1	@ R0 = R0+R1
4		ADD R1, R1, #1	@ R1 = R1+1
5		CMP R1, #10	@ Compare R1 with 10
6		BLE do	@ if less than or equal goto do
7	end:	...	@ End of do-while loop
8		...	@

6. หากเงื่อนไข Less Than or Equal (LE) เป็นจริง ชีพิญจะกระโดดไปทำงานคำสั่งที่ขึ้นต้นด้วย do หากไม่เป็นจริง ชีพิญจะทำงานประโยคที่ 7 ต่อไป
7. คือ คำสั่งที่ขึ้นต้นด้วยเลbel end

#### 4.7.6 การวนรอบชนิด FOR จำนวน 2 ชั้น



รูปที่ 4.11: โฟล์ชาร์ตการทำงานของประโยคการวนรอบชนิด For 2 ชั้น

โฟล์ชาร์ตนี้เป็นการวนรอบ 2 ชั้น ในรูปที่ 4.11 มีการทำงานเบื้องต้นดังนี้

1. โปรแกรมจะตั้งค่าเริ่มต้น (Initialization) ตัวแปรนับรอบที่ 1 (Index\_1)
2. โปรแกรมจะตรวจสอบเงื่อนไข Condition\_1 ของลูปชั้นนอก (Outer Loop)
  - (a) หากเป็นจริง (True) โปรแกรมจะทำงานส่วนที่เป็น Loop Body\_1

- i. เริ่มต้นโดยตั้งค่าเริ่มต้นตัวแปรนับรอบที่ 2 (index\_2)
  - ii. โปรแกรมจะตรวจสอบเงื่อนไข Condition\_2 ของลูปชั้นใน (Inner Loop)
    - หากเป็นจริง (True) โปรแกรมจะทำงานตาม Loop Body\_2 เมื่อแล้วเสร็จ โปรแกรมจะเพิ่ม (Increment) หรือ ลด (Decrement) ตัวแปรนับรอบที่ 2 (Index\_2) และวนกลับไปข้อที่ ii
    - หากเป็นเท็จ (False) โปรแกรมจะข้ามไปทำงานที่ประโยคถัดไปโดยไม่ทำงานลูปชั้นในแม้แต่รอบเดียว
  - iii. โปรแกรมจะเพิ่ม (Increment) หรือ ลด (Decrement) ตัวแปรนับรอบที่ 1 (Index\_1) และวนกลับไปข้อที่ 2
- (b) หากเป็นเท็จ (False) โปรแกรมจะข้ามไปทำงานที่ประโยคถัดไปโดยไม่ทำงานลูปชั้นนอกแม้แต่รอบเดียว

## 4.8 การเรียกใช้ฟังก์ชัน (Function Call)

การเรียกใช้ฟังก์ชันในภาษาแอลซเซมบลีคล้ายกับ การเรียกใช้ฟังก์ชันในภาษา C/C++ ดังนั้น ผู้อ่านควรทำความเข้าใจการเรียกใช้ฟังก์ชันในภาษา C/C++ ให้ลึกซึ้งก่อน

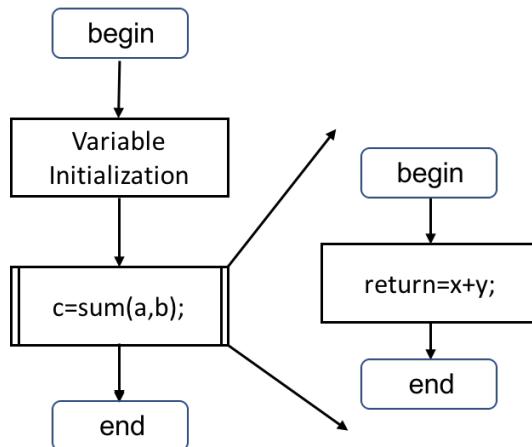
### 4.8.1 การเรียกใช้ฟังก์ชันในภาษา C/C++

ตัวอย่างการสร้างฟังก์ชันในภาษา C/C++ และเรียกใช้ฟังก์ชันด้วยการส่งพารามิเตอร์ (Parameter) จำนวน 2 ตัว คือ ตัวแปร a และตัวแปร b และรับค่ารีเทิร์นกลับด้วยตัวแปร c

**ตัวอย่างที่ 4.8.1 ตัวอย่างการสร้างฟังก์ชันในภาษา C/C++ และเรียกใช้ฟังก์ชัน**

```
int main() {
    int a, b, c;
    ...
    c = sum(a, b);
    ...
    return 0;
}
int sum(int x, int y) {
    return x+y;
}
```

รูปที่ 4.12 แสดงไฟล์ชาร์ตการทำงานของการเรียกใช้ฟังก์ชันชื่อ sum( a, b ) ในตัวอย่างที่ 4.8.1 กลไกนี้จะเกิดขึ้นเมื่อกับการเรียกใช้ฟังก์ชันมาตรฐานในโปรแกรมภาษา C/C++ ซึ่งสามารถเรียกใช้ช้าแล้วช้าอีกได้ เพราะทำงานเป็นมอดูล นักพัฒนาสามารถนำฟังก์ชันที่พัฒนาเองไปเผยแพร่ ยกตัวอย่างเช่น ฟังก์ชัน printf ในไฟล์ stdio.h ซึ่งย่อมาจากคำว่า Standard I/O ซึ่งเป็นฟังก์ชันในไลบรารีหลักของภาษา C ชื่อ glibc ฟังก์ชันมาตรฐานเหล่านี้สามารถนำมาร่วมกับไฟล์อื่นๆ เช่น stdio.h และ stdlib.h ฯลฯ ตามหลักการในรูปที่ 3.11



รูปที่ 4.12: โฟล์ชาร์ตการทำงานของประโภคเรียกใช้ฟังก์ชัน sum( a, b ) ในตัวอย่างที่ 4.8.1

#### 4.8.2 คำสั่งเรียกใช้ฟังก์ชันในภาษาแอสเซมบลี

ภาษาแอสเซมบลีของซีพียูต่างๆ มีศักยภาพที่ใกล้เคียงกับภาษาสูง เช่น C/C++ ที่โปรแกรมเมอร์สามารถสร้างฟังก์ชันของตนเองได้ คำสั่งภาษาแอสเซมบลีของ ARM ที่รองรับการเรียกใช้ฟังก์ชัน คือ

รูปแบบ	ความหมาย
BL func_name	<ul style="list-style-type: none"> <li>· LR = PC+4</li> <li>· PC = address(func_name)</li> <li>· เปลี่ยนค่าของ PC ไปยังเลขเบลชื่อฟังก์ชัน func_name</li> </ul>
BX LR	<ul style="list-style-type: none"> <li>PC = LR</li> <li>เปลี่ยนค่าของ PC ไปยังแอดเดรสที่เก็บในรีจิสเตอร์ LR</li> </ul>

คำสั่ง BL ย่อมาจากคำว่า **B**ranch and **L**ink คำว่า func\_name คือ เลเบลชื่อหน้าที่เป็นชื่อฟังก์ชันที่ต้องการเรียกใช้ ในระหว่างที่ซีพียุปปฏิบัติตามคำสั่ง BL func\_name ซีพียุจะทำงานย่อย 2 ขั้นดังต่อไปนี้

- LR = PC+4 เพื่อบันทึกค่า address ของคำสั่งถัดไปที่ติดกับคำสั่ง BL ไว้ในรีจิสเตอร์ LR และ
- PC = address(func\_name) เพื่อเปลี่ยนแปลงค่าแอดเดรสในรีจิสเตอร์ PC ไปเป็นแอดเดรสของเลเบล func\_name

หลังจากนั้น ซีพียุจะปฏิบัติตามคำสั่งแรกภายในฟังก์ชันนั้นทีละคำสั่งไปเรื่อยๆ ( $PC=PC+4$ ) จนแล้วเสร็จ และคำสั่ง BX LR ในบรรทัดสุดท้ายของฟังก์ชัน เป็นการสั่งให้ซีพียุทำงานย่อย 2 ขั้นตอน ดังต่อไปนี้

- PC = LR ความหมายคือ PC จะถูกเปลี่ยนแปลงเป็นค่าตำแหน่ง address ที่รีจิสเตอร์ LR เก็บค่าไว้ก่อนหน้า
- นำค่าในรีจิสเตอร์ R0 ส่งกลับ (Return) ไปยังฟังก์ชันที่เรียก

ผู้อ่านสามารถทำความเข้าใจการเรียกใช้ฟังก์ชันจากตัวอย่างต่อไปนี้

ตัวอย่างโปรแกรมเรียกใช้ฟังก์ชันภาษาแอสเซมบลีที่คล้ายกับตัวอย่างที่ 4.8.1

#	เลbel	คำสั่ง	คอมเมนท์
1	main:	...	@ Initialize R4 (variable a)
2		...	@ Initialize R5 (variable b)
3		MOV R0, R4	@ Pass R0 to function sum
4		MOV R1, R5	@ Pass R1 to function sum
5		BL sum	@ Call function sum
6		...	@ an instruction
7		...	@ an instruction
8		BX LR	@ Return to kernel
9		...	@ other functions
10	sum:	ADD R0, R0, R1	@ entry point of function sum
11		BX LR	@ Return the result in R0 to main

การทำงานของตัวอย่างจะเริ่มที่ฟังก์ชัน main ตามปกติ ในบรรทัดที่ 3 จะเป็นการส่งค่าพารามิเตอร์ของตัวแปร a ในรีจิสเตอร์ R0 และตัวแปร b ด้วยรีจิสเตอร์ R1 ไปยังฟังก์ชัน sum คล้ายกับโฟลว์ชาร์ตการเรียกใช้ฟังก์ชันในรูปที่ 4.12 และส่งค่าผลลัพธ์กลับทาง R0 ประโยชน์ของการสร้างฟังก์ชัน คือ การเรียกใช้ฟังก์ชันซ้ำแล้วซ้ำอีก ตามตัวอย่างโปรแกรมภาษา C ต่อไปนี้

ตัวอย่างที่ 4.8.2 ตัวอย่างโปรแกรมภาษา C เรียกใช้ฟังก์ชันชื่อ myFunction() สองครั้ง

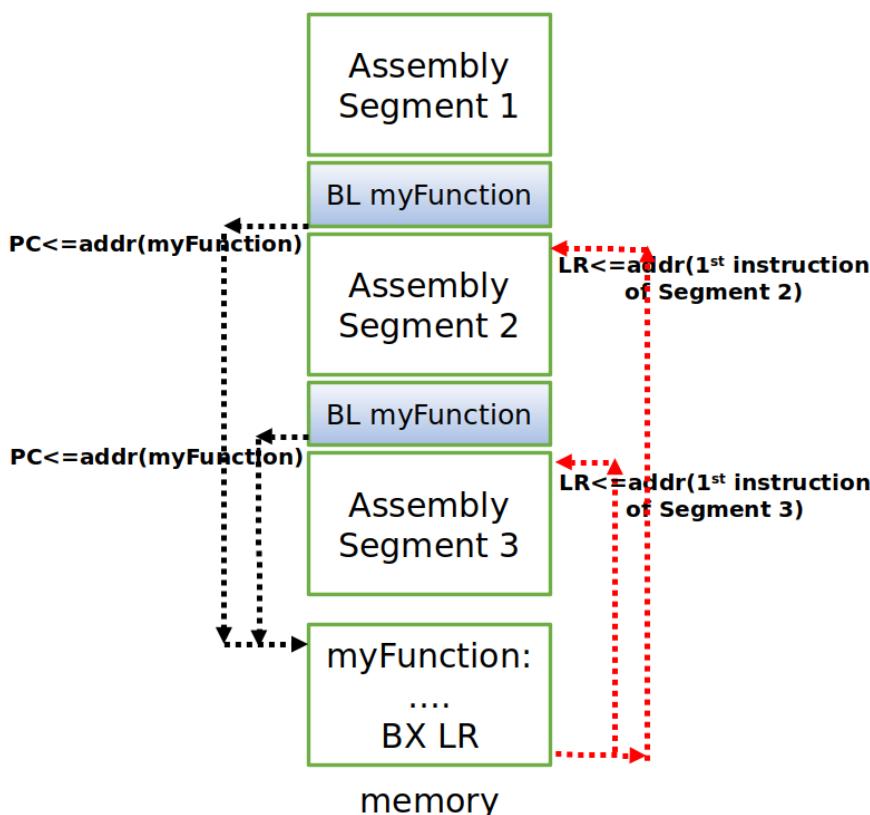
```
int main() {
    int a, b, c;
    ...
    a = myFunction();
    ...
    b = myFunction();
    ...
    return 0;
}
int myFunction() {
    ...
    return 0;
}
```

หมายเหตุ ... คือ ประโยชน์ค่าลับสืบสานๆ ที่ไม่เกี่ยวข้องกับตัวอย่าง

ผู้อ่านสามารถทำความเข้าใจการทำงานของโปรแกรมในตัวอย่างที่ 4.8.2 จากรูปที่ 4.13 การทำงานของฟังก์ชัน main() เริ่มต้นตามปกติ เรียกใช้ฟังก์ชันชื่อ myFunction() สองครั้ง ซึ่งจะย้ายการทำงานไปใน myFunction จนสิ้นสุดแล้วรีเทิร์น 0 กลับมาที่ประโยชน์คัดไป ซึ่งฟังก์ชัน main() ทำงานแล้วเสร็จ ซึ่งจะรีเทิร์น 0 กลับไปหาฟังก์ชันที่เรียกใช้ main() เช่นกัน

ในภาษาแอสเซมบลี คำสั่ง BL myFunction จะเก็บค่าแอดเดรสต์จาก BL myFunction ในรีจิสเตอร์ LR โปรดสังเกตคำอธิบายทางขวา นั่นคือ  $LR = PC + 4 = \text{address of 1st instruction of Segment 2}$  และจะเปลี่ยนค่า PC เป็นแอดเดรสของคำสั่งแรกในของฟังก์ชัน myFunction  $PC = \text{address(myFunction)}$  โปรดสังเกตเส้นประสีดำเนินบน เมื่อทำงานฟังก์ชัน myFunction แล้วเสร็จ คำสั่ง BX LR จะเปลี่ยนค่า PC เป็นค่าของรีจิสเตอร์ LR ( $PC = LR$ ) โปรดสังเกตเส้นประสีแดงเดินบนซึ่งเป็นแอดเดรสของคำสั่งถัดไป

หลังจากนั้น ฟังก์ชัน main จะเรียกคำสั่ง BL myFunction รอบที่สอง  $LR = PC + 4 = \text{address of 1st instruction of Segment 3}$  และ  $PC = \text{address(myFunction)}$  โปรดสังเกตเส้นประสีดำเนินกล่าง หลังจากที่ทำงาน myFunction แล้วเสร็จ ประโยชน์ BX LR จะรีเทิร์นกลับ ณ ตำแหน่งถัดไปภายในฟังก์ชัน main ( $PC = LR$ ) โปรดสังเกตเส้นประสีแดงเดินกล่าง หมายเหตุ addr ย่อมาจากคำว่า address



รูปที่ 4.13: การทำงานของคำสั่ง BL myFunction เมื่อมีการเรียกใช้ 2 ครั้งจากฟังก์ชัน main ในหน่วยความจำคล้ายกับตัวอย่างที่ 4.8.2 หมายเหตุ ... หมายถึงประโยชน์คำสั่งอื่นๆ ที่ไม่เกี่ยวข้องกับตัวอย่าง

ผู้อ่านสามารถอ่านรายละเอียดการพัฒนาเพิ่มเติมได้ที่ การทดลองที่ 5 การพัฒนาโปรแกรมด้วยภาษา C ในภาคผนวก E และการทดลองที่ 6 ถึง 8 การพัฒนาโปรแกรมด้วยภาษาแอสเซมบลี ในภาคผนวก F ถึงภาคผนวก H

## 4.9 อุปกรณ์และคำสั่งภาษาแอลซีเมบลีเวอร์ชันอื่นของ ARM

วิวัฒนาการของชุดคำสั่งของซีพียู ARM จากเริ่มก่อตั้งจนถึงปัจจุบันมีความเข้มข้นอย่างกับอุปกรณ์ที่ใช้ชิปเหล่านั้นโดยตรง จะสังเกตได้จากความหลากหลายของอุปกรณ์ที่ใช้งานชิป ARM จากสมาร์ตการ์ด (Smart Card) ในบัตรเครดิตและบัตรเดบิต ไปจนถึงชูเปอร์คอมพิวเตอร์ แต่ชิปเหล่านี้มีเป้าหมายร่วมกัน คือ การประหยัดพลังงาน ไฟฟ้า

### 4.9.1 อุปกรณ์ดิจิทัลที่ใช้ซีพียูที่ออกแบบโดยบริษัท ARM

ตารางที่ 4.10: ชนิดและจำนวนอุปกรณ์ ( $\times 10^6$ ) จำนวนชิป ( $\times 10^6$ ) จำนวนชิปต่ออุปกรณ์ จำนวนชิป TAM (Total Available Market) ( $\times 10^6$ ) จำนวนชิปของ ARM ( $\times 10^6$ ) และเปอร์เซ็นต์ส่วนแบ่งการตลาด (Share) ของ ARM ในปี ค.ศ. 2010 ที่มา: [zdnet.com](http://zdnet.com) หมายเหตุ TAM: Total Available Market

	Devices Shipped (Million of Units)	2010 Devices	Chips/ Device	TAM 2010 Chips	2010 ARM	2010 Share
Mobile	Smart Phone	280	2-5	1,200	1,100	90%
	Feature Phone	760	1-3	1,900	1,700	90%
	Low End Voice	570	1	570	540	95%
	Portable Media Players	150	1-3	300	220	70%
	Mobile Computing* (apps only)	230	1	230	25	10%
Non-Mobile	PCs & Servers (apps only)	220	1	220	0	0%
	Digital Camera	130	1-2	200	160	80%
	Digital TV & Set-top-box	350	1-2	450	160	35%
	Networking	670	1-2	750	185	25%
	Printers	120	1	120	75	65%
	Hard Disk & Solid State Drives	670	1	670	560	85%
	Automotive	1,800	1	1,800	180	10%
	Smart Card	5,400	1	5,400	330	6%
	Microcontrollers	5,800	1	5,800	560	10%
	Others **	1,700	1	1,800	270	15%
<b>Total</b>		<b>19,000</b>		<b>22,000</b>	<b>6,100</b>	<b>28%</b>

ตารางที่ 4.10 และ 4.11 แสดงให้เห็นภาพรวมว่า จำนวนอุปกรณ์ทั่วโลกในปี 2010 เท่ากับ 19,000 ล้านตัว เพิ่มขึ้นเป็น 27,000 ล้านตัวในปี 2015 ซึ่งในปี 2010 ARM มีส่วนแบ่งการตลาดของซิปซีพียูทั่วโลกเท่ากับ 28% คิดเป็นชิป ARM จำนวน 6,100 ล้านชิปจากทั้งหมด (TAM 2010 Chips) 22,000 ล้านชิป และทั้งหมด (TAM 2015 Chips) เพิ่มขึ้นเป็น 34,000 ล้านชิปในปี 2015 ตัวเลขเหล่านี้รวมจากประเภทของสินค้าและอุปกรณ์ ที่ใช้ซิปซีพียูแบ่งเป็น กลุ่มโทรศัพท์เคลื่อนที่ (Mobile) และกลุ่มอื่นๆ (Non-Mobile)

กลุ่มโทรศัพท์เคลื่อนที่แบ่งเป็น สมาร์ตโฟน (Smart Phone) ราคาสูง โทรศัพท์ใช้งานทั่วไป (Feature Phone) ราคากันกลาง เครื่องเล่นสื่อพกพา (Portable Media Players) และ โน๊ตบุ๊คแบบโทรศัพท์เคลื่อนที่ (Mobile Apps) โดยอุปกรณ์ที่ ARM มีส่วนแบ่งการตลาดสูงที่สุด (95%) คือ Low End Voice รองลงมาคือ

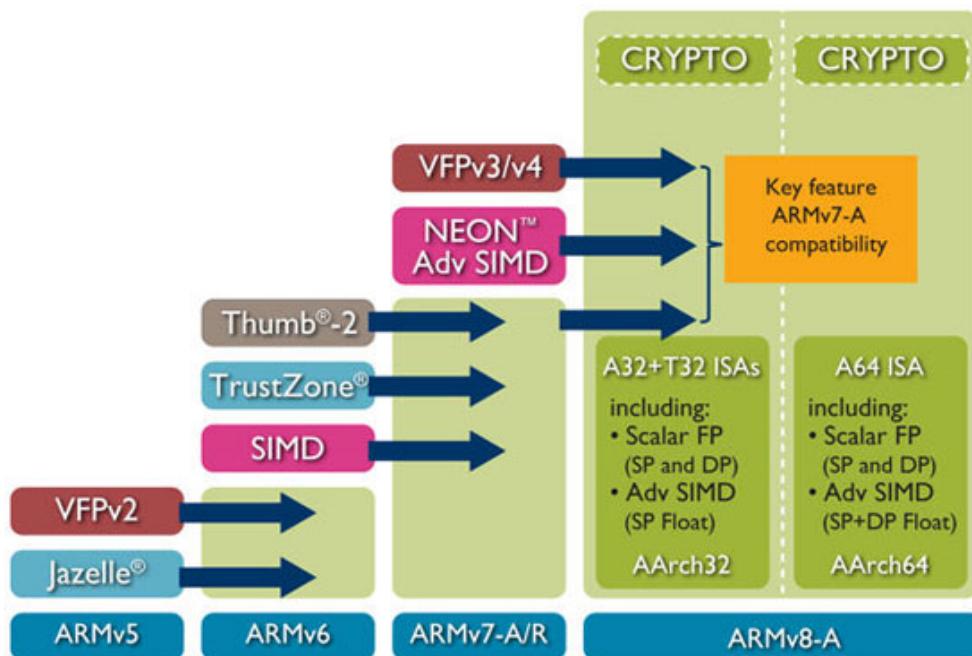
Smart Phone และ Feature Phone ตามลำดับ โทรศัพท์พื้นฐาน (Low End Voice) ราคาถูก คาดว่าตลาดของสมาร์ตโฟนจะเติบโตและได้รับความนิยมเพิ่มสูงขึ้นเรื่อยๆ รวมถึงตลาดของโน้บบุ๊กและแท็บเล็ต

**ตารางที่ 4.11:** ชนิดอุปกรณ์ จำนวนชิป TAM (Total Available Market) ปี 2010 ( $\times 10^6$ ) จำนวนชิปของ ARM ปี 2010 ( $\times 10^6$ ) จำนวนอุปกรณ์ TAM ปี 2015 ( $\times 10^6$ ) จำนวนชิปต่ออุปกรณ์ และจำนวนชิป TAM ปี 2015 ( $\times 10^6$ ) ที่มา: [7boot.com](http://7boot.com) หมายเหตุ TAM: Total Available Market

Devices Shipped (Million of Units)	TAM 2010 Chips	10 ARM Share	TAM 2015 Devices	Chips/ Unit	TAM 2015 Chips
Smart Phone	1,200	90%	1,100	3-5	4,000
Feature Phone	1,900	90%	650	2-3	2,000
Low End Voice	570	95%	700	1-2	1,300
Portable Media Players	300	70%	120	1-3	250
Mobile Computing* (apps only)	230	10%	750	1	750
PCs & Servers (apps only)	220	0%	250	1	250
Digital Camera	200	80%	150	1-2	250
Digital TV & Set-top-box	450	35%	500	1-4	1,200
Networking	750	25%	800	1-2	1,400
Printers	120	65%	200	1	200
Hard Disk & Solid State Drives	670	85%	1,100	1	1,100
Automotive	1,800	10%	2,200	1	2,200
Smart Card	5,400	6%	7,700	1	7,700
Microcontrollers	5,800	10%	9,000	1	9,000
Others **	1,800	15%	2,000	1	2,000
<b>Total</b>	<b>22,000</b>	<b>28%</b>	<b>27,000</b>		<b>34,000</b>

ในส่วนของกลุ่มอื่นๆ (Non-Mobile) ประกอบด้วย อุปกรณ์ที่หลากหลายกว่า โดยอุปกรณ์ที่ ARM มีส่วนแบ่งการตลาดสูงที่สุด (85%) คือ ฮาร์ดดิสก์ไดร์ฟและโซลิดสเตทไดร์ฟ (Hard Disk Drive& Solid State Drive) รองลงมา คือ กล้องดิจิทัลและพรินเตอร์ (Digital Camera และ Printer) ตามลำดับ โดย ARM คาดว่าตลาดของ ฮาร์ดดิสก์ไดร์ฟและโซลิดสเตทไดร์ฟจะเติบโตและได้รับความนิยมเพิ่มสูงขึ้นเรื่อยๆ โดยเฉพาะโซลิดสเตทไดร์ฟ นอกจากนี้ ตลาดของทีวีดิจิทัลและกล่อง (Digital TV & Set Top Box) และตลาดของไมโครคอนโทรลเลอร์ (Microcontroller) น่าจะมีการขยายตัวในปี 2015 ตามตารางที่ 4.11 ส่วนตลาดใหม่ๆ ที่ ARM จะขยายตัวเพิ่ม คือ ชิปสำหรับเครื่องคอมพิวเตอร์เซิร์ฟเวอร์ภายในองค์กร และชิปสำหรับเครื่องซูเปอร์คอมพิวเตอร์ เพื่อรับการคำนวณแบบขนานซึ่งมีรายละเอียดเพิ่มเติมในบทที่ 8

#### 4.9.2 คำสั่งภาษาและซีมบลีเวอร์ชันอื่นของ ARM



รูปที่ 4.14: การควบรวมชุดคำสั่งภาษาและซีมบลีของ ARM ตั้งแต่เวอร์ชัน 5 ถึงเวอร์ชัน 8A โดยเวอร์ชันใหม่จะรวมคำสั่งของเวอร์ชันเก่าเพื่อรองรับการทำงานซอฟต์แวร์เดิม ที่มา: [arm.com](http://arm.com)

ในอดีตที่ผ่านมา ARM ได้พัฒนาภาษาและซีมบลีออกมาน้ำหนึ่งตัวเดียว คือ ARMv1 ตามที่แสดงในรูปที่ 4.14 ภาษาและซีมบลีเวอร์ชัน 5 (ARMv5) และเวอร์ชัน 6 (ARMv6) ที่ผ่านมาในอดีตถูกควบรวมเป็นภาษาและซีมบลีเวอร์ชัน 7 (ARMv7) พร้อมชุดคำสั่งใหม่เพิ่มเติม ซึ่งรองรับได้เพียงระบบปฏิบัติการ เวอร์ชัน 32 บิตเท่านั้น โดยแบ่งเป็น ARMv7-A สำหรับชิปปุ๋ย ARM Cortex-A, ARMv7-M สำหรับชิปปุ๋ย ARM Cortex-M และ ARMv7-R สำหรับชิปปุ๋ย ARM Cortex-R และล่าสุดภาษาและซีมบลีเวอร์ชัน 8 (ARMv8-A) สำหรับชิปปุ๋ย ARM Cortex-A รุ่นปัจจุบัน ซึ่งรองรับระบบปฏิบัติการ เวอร์ชัน 64 และ 32 บิตด้วย

นอกจากภาษาและซีมบลีเวอร์ชันต่างๆ ที่กล่าวมาข้างต้น จุดเด่นของ ARM ยังมีชุดคำสั่งภาษาและซีมบลี ARM ที่มีความยาว 16 และ 32 บิต ซึ่งเรียกว่า Thumb, Thumb-2, T32 ซึ่งนิยมใช้สำหรับอุปกรณ์ที่มีหน่วยความจำความจุน้อย ต้นทุนต่ำ รายละเอียดเพิ่มเติมใน [wikipedia](https://en.wikipedia.org/wiki/ARM_architecture)

คำสั่งที่กล่าวมาข้างต้นในบทนี้หมายความว่า คำสั่งภาษาและซีมบลีที่ได้อธิบายไปแล้ว ARM ได้ออกแบบชุดคำสั่งภาษาและซีมบลีสำหรับประมวลผลข้อมูลที่เรียกว่า Vector Floating Point version 3/4 (VFPv3/v4), Single Instruction Multiple Data (SIMD) (อ่านว่า ซิมดี) และ NEON Advanced SIMD (Adv SIMD) ซึ่งได้อธิบายในหัวข้อที่ 4.1 หมายความว่า คำสั่งที่มีความสามารถในการคำนวณข้อมูลแบบขนาน (Parallel Computing) ตามเนื้อหาในบทที่ 8 เช่น ชุดประมวลผลพิวเตอร์ นอกเหนือจากนี้ ผู้อ่านสามารถศึกษารายละเอียดอื่นๆ เพิ่มเติมเกี่ยวกับภาพรวมของ ARM ได้ที่ [wikipedia](https://en.wikipedia.org/wiki/ARM_architecture)

## 4.10 สรุปท้ายบท

คำสั่งภาษาแօสเซมบลีของ ARM มีลักษณะเด่นเมื่อเปรียบเทียบกับภาษาอื่นๆ เช่น การมีคำสั่งเลื่อนบิตภายในคำสั่งคณิตศาสตร์ การตรวจสอบเงื่อนไขก่อนการปฏิบัติตามคำสั่งนั้น เป็นต้น ทำให้ชิปซีพียูที่ผลิตตามการออกแบบของ ARM บริโภคพลังงานน้อย จึงได้รับความนิยมในอุปกรณ์เคลื่อนที่มาเป็นระยะเวลาต่อเนื่อง ที่ผ่านมาสถาปัตยกรรมของคำสั่งสามารถแบ่งออกได้เป็น

- **สถาปัตยกรรมโหลด/สโตร์ (Load/Store Architecture)** ARM ออกแบบคำสั่งภาษาแօสเซมบลีตามหลักการ RISC (Reduced Instruction Set Computer) ข้อมูลเพิ่มเติมที่ [wikipedia](#) และแตกต่างจากคำสั่งจากสถาปัตยกรรมอื่นๆ ดังนี้
- **สถาปัตยกรรม x86 (x86 Architecture)** ในภาษาแօสเซมบลีของ Intel 80x86 หรือเรียกว่า x86 ซึ่งบางคำสั่งใช้การอ่านค่าจากหน่วยความจำเพื่อประมวลผล ทำให้การประมวลซับซ้อนและล่าช้า ข้อมูลเพิ่มเติมที่ [wikipedia](#)
- **สแต็กแมชชีน (Stack Machine)** ได้แก่ ภาษาจาวาไบต์โค้ด (Java Bytecode) จัดเป็นคำสั่งภาษาแօสเซมบลีเมื่อแปลจากซอฟต์แวร์ Java เทคโนโลยีที่น่าสนใจของ ARM เรียกว่า Jazelle สามารถรองรับการประมวลผลคำสั่ง Java Bytecode ด้วยฮาร์ดแวร์ทำให้ชิปบริโภคพลังงานต่ำโดยเฉพาะโทรศัพท์สมาร์ตโฟนที่ใช้ระบบปฏิบัติการ Android ซึ่งซอฟต์แวร์ส่วนใหญ่ใช้ภาษา Java พัฒนา รายละเอียดเพิ่มเติมใน [wikipedia](#)

## 4.11 คำถ้าท้ายบท

1. รีจิสเตรอร์ PC (Program Counter) ทำหน้าที่อะไร และเกี่ยวข้องกับการรันของโปรแกรมอย่างไร
2. คำสั่ง POP สำหรับอ่านค่าข้อมูลออกจากสแต็กเซกเมนต์ ทำงานอย่างไร
3. คำสั่ง PUSH สำหรับเก็บค่าข้อมูลลงในสแต็กเซกเมนต์ ทำงานอย่างไร
4. คำสั่ง POP และ PUSH ทำงานร่วมกับรีจิสเตรอร์ Stack Pointer (SP) อย่างไร
5. สแต็กเซกเมนต์ มีพิธีทางขยายขนาดไปในพิธีทางใด เพราะอะไร
6. รีจิสเตรอร์ LR (Link Register) ทำหน้าที่อะไร และเกี่ยวข้องกับการรันของโปรแกรมอย่างไร
7. จงอธิบายว่า BX LR ทำงานอย่างไร เหตุใดจึงต้องเป็นคำสั่งสุดท้ายของฟังก์ชัน
8. จงเขียนฟังก์ชันเพื่อกำหนดหาค่าสัมบูรณ์ของ R0-R1 และรีเทิร์นค่าในค่า R0
9. จงเขียนโปรแกรมลูปวนรอบ 2 ชั้น โดยใช้การวนรอบชนิด
  - While
  - Do-While
10. การทำงานของฟังก์ชันที่เรียกตัวเอง (Recursive Function Call)

- จงวาดโพลาร์ชาร์ตเพื่อแสดงการทำงานกรณีปกติและกรณีหยุดเรียกตัวเอง
- จงวาดรูปการทำงานประกอบด้วยหน่วยความจำคล้ายกับรูปที่ [4.13](#)

## บทที่ 5

# ลำดับชั้นของหน่วยความจำ (Hierarchy of Memory)

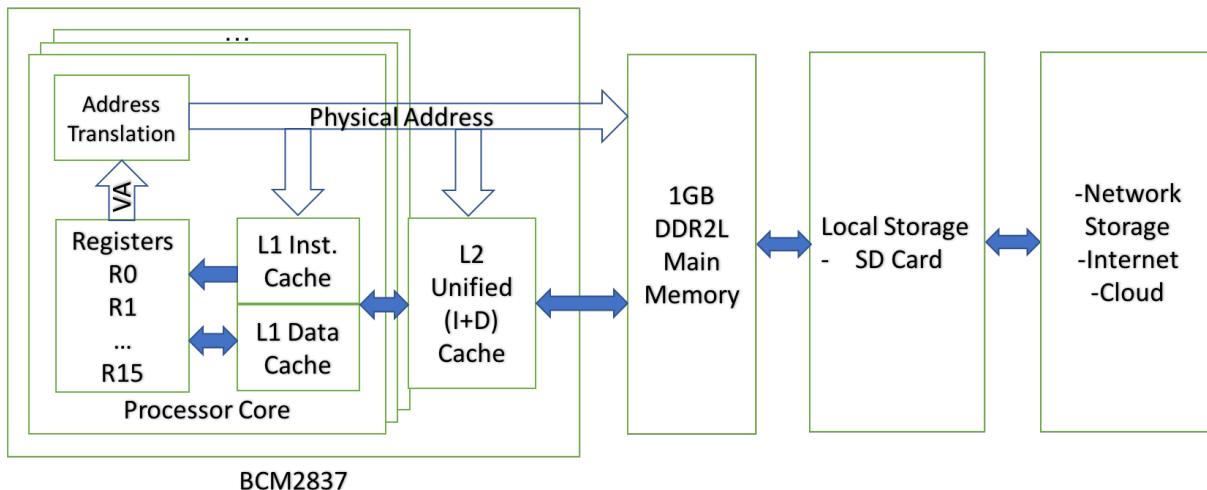
ลำดับชั้นของหน่วยความจำ (Hierarchy of Memory) อาศัยการทำงานร่วมกันของหน่วยความจำหลายชนิด ที่มีความจุ (Capacity) ที่แตกต่างกันมาทำงานร่วมกัน เพื่อผ่อนความเร็วของหน่วยความจำแต่ละชนิดเข้าด้วยกัน และช่วยประหยัดต้นทุนโดยรวมของระบบ บทนี้มีวัตถุประสงค์ดังนี้

- เพื่อให้เข้าใจถึงลำดับชั้นและการทำงานร่วมกันระหว่างหน่วยความจำชนิดต่างๆ ได้แก่ รีจิสเตอร์ แคช ลำดับชั้นต่างๆ หน่วยความจำหลัก และอุปกรณ์เก็บรักษาข้อมูล
- เพื่อให้เข้าใจถึงการทำงานร่วมกันของเวอร์ชัลเมโมรีในระบบลินุกซ์ เวอร์ชัน 32 บิตสำหรับบอร์ด Pi3/Pi4 ประมวลผลและระบบปฏิบัติการในทางทฤษฎี
- เพื่อให้เข้าใจถึงการทำงานของเวอร์ชัลเมโมรีในระบบลินุกซ์ เวอร์ชัน 32 บิตสำหรับบอร์ด Pi3/Pi4
- เพื่อให้เข้าใจถึงการทำงานของแคชชนิด Direct Map (DM) และ Set Associative (SA) ซึ่งนิยมใช้ในชิปซีพียู
- เพื่อให้เข้าใจถึงความแตกต่างระหว่างหน่วยความจำสามแทติกแรมและ SDRAM ในด้านโครงสร้างภายใน การทำงาน และประสิทธิภาพ

บทนี้จะอธิบายลำดับชั้นของหน่วยความจำว่า เหตุใดคอมพิวเตอร์จึงต้องประกอบด้วยหน่วยความจำหลายชนิด ซึ่งต่อเนื่องจากหัวข้อที่ 3.1.2 เนื้อหาในบทนี้อศัยการทดลองที่ 4 ภาคผนวก D การใช้งานระบบปฏิบัติการยูนิกซ์ ซึ่งจะทำให้ผู้อ่านเข้าใจหลักการและรายละเอียดมากขึ้น

### 5.1 ลำดับชั้นของหน่วยความจำของบอร์ด Pi3/Pi4

หน่วยความจำมีหลายชนิดตามหลักการออกแบบ ความซับซ้อน/เทคโนโลยีการผลิต ความจุของหน่วยความจำแต่ละชนิดมีความหลากหลาย ความเร็วหรือสมรรถนะในการอ่านหรือเขียนข้อมูล และ ต้นทุนต่อความจุหนึ่งบิต ทำให้การจัดวางมีผลต่อประสิทธิภาพโดยรวม หน่วยความจำบางชนิดสามารถบรรจุในชิป (On Chip) เดียว กับซีพียู ในขณะที่บางชนิดจำเป็นต้องอยู่คู่กับซีพียู (Off Chip) ตัวมาเล่นนี้จะอธิบายเทคโนโลยีและการทำงานหน่วยความจำชนิดต่างๆ ของบอร์ด Pi3/Pi4 เป็นกรณีศึกษาลำดับชั้นและการจัดวางหน่วยความจำชนิดต่างๆ



รูปที่ 5.1: ลำดับขั้นของหน่วยความจำชนิดต่างๆ สำหรับชิป BCM2837/BCM2711, (VA: Virtual Address)

บอร์ด Pi3/Pi4 ประกอบด้วยหน่วยความจำที่หลากหลาย บางส่วนอยู่ในชิปเดียวกับซีพียู (On Chip) และบางส่วนอยู่นอกชิป (Off Chip) ในรูปที่ 5.1 หน่วยความจำเหล่านี้อยู่ในลำดับชั้นต่างๆ ทั้งโกล์และไกลจาก ALU (Arithmetic Logic Unit) ชนิด Integer และ Floating Point IEEE754 ประกอบด้วย รูปที่ 4.2 เพิ่มอุปกรณ์เก็บรักษาข้อมูลและเครือข่าย

- **รีจิสเตอร์ (Register)** ในชุดคำสั่ง ARMv7 ซึ่งเป็นเวอร์ชัน 32 บิต ประกอบด้วยรีจิสเตอร์ R0, R1, ..., R15 ซึ่งใช้หน่วยความจำชนิด静态ติกแรม (Static RAM: SRAM) ทำให้สามารถเข้าถึงข้อมูลได้ภายในเวลาสั้นๆ เพียง 0.5-1 นาโนวินาที (Nano Second)
  - **แคชลำดับที่ 1 (Level 1)** ซึ่งแบ่งเป็นแคชคำสั่ง หรือ Instruction Cache และแคชข้อมูล หรือ Data Cache ซึ่งใช้หน่วยความจำ静态ติกแรม มีขนาดประมาณ 16-64 KiB แต่ละแกนประมวลผลของ ARM Cortex A53/A72 จะมีแคชทั้งสองชั้นอยู่ด้วยกัน
  - **แคชลำดับถัดไป** ซึ่งใช้หน่วยความจำ静态ติกแรม เช่น กัน โดยมีความจุ 64 KiB (kilobyte) ขึ้นไปจนถึงหลักหลายสิบเมบิไบต์ (MebiByte: MiB) แคชลำดับที่ 2 ทำหน้าที่พักเก็บห้องคำสั่งและข้อมูล (Unified) คล้ายหน่วยความจำขนาดเล็กของแคชคำสั่งและแคชข้อมูลลำดับที่ 1 ซึ่งเชื่อมโยงผู้ผลิตบางรายมีแคชลำดับที่ 3 ที่มีความจุมากขึ้น ทำหน้าที่คล้ายหน่วยความจำภายในภาพของแคชลำดับที่ 2 ซึ่งต้นทุนในการผลิตซิปเหล่านี้แปรผันตรงกับความจุของแคชเช่นกัน

- หน่วยความจำหลัก (Main Memory) หรือบางครั้งเรียกว่า หน่วยความจำภายในภาพ (Physical Memory) หรือ หน่วยความจำปฐมภูมิ (Primary Memory) ซึ่งนิยมใช้เทคโนโลยีหน่วยความจำชนิดไดนามิก แรม (Dynamic RAM: SDRAM) หน่วยความจำหลักมักมีขนาด 512 เมบิไบต์ขึ้นไป จนถึงหลายสิบกิกิไบต์ (GiB) (GigaByte: GB) ขึ้นอยู่กับชนิดและต้นทุนของคอมพิวเตอร์นั้นๆ
- อุปกรณ์เก็บรักษาข้อมูล (Storage) หรือเรียกว่า หน่วยความจำทุติยภูมิ (Secondary Memory) ซึ่งมีทางเลือกจากหลายเทคโนโลยี เพื่อเก็บรักษาข้อมูลไม่ให้สูญหาย ถึงแม่ผู้ใช้จะปิดเครื่องหรือแหล่งจ่ายไฟ พลังงานหมด เช่น คอมพิวเตอร์พกพาซึ่งต้องอาศัยพลังงานจากแบตเตอรี่ ชนิดของอุปกรณ์เก็บรักษาข้อมูลโดยจะกล่าวรายละเอียดเพิ่มเติมในบทที่ 7
  - การ์ดหน่วยความจำ SD (Secure Digital) ซึ่งสร้างจากเทคโนโลยีหน่วยความจำแฟลช (Flash Memory) ที่นิยมใช้กับโทรศัพท์ และกล้องถ่ายในหัวข้อที่ 3.1.4 ทำให้มีความจุมาก ระดับ 16 กิกิไบต์ (GiB) ขึ้นไป หน่วยความจำแฟลชความเร็วสูงกว่าเมื่อเทียบกับฮาร์ดดิสก์ หน่วยความจำ SD สามารถพกพาได้สะดวก เพราะขนาดเล็กและน้ำหนักเบา จึงนิยมใช้บันทึกข้อมูลในกล้องดิจิทัล โทรศัพท์สมาร์ตโฟน และอื่นๆ
  - โซลิดสเตทไดรฟ์ (Solid State Drive: SSD) ซึ่งใช้เทคโนโลยีหน่วยความจำแฟลช (Flash Memory) มีขนาด 128 กิกิไบต์ (GiB) ขึ้นไปจนถึงหลายเทอราไบต์ และคาดว่าจะทดแทนฮาร์ดดิสก์ในปัจจุบันและอนาคต
  - ฮาร์ดดิสก์ (Hard Disk) ซึ่งใช้เทคโนโลยีหน่วยความจำแผ่นแม่เหล็ก (Magnetic Disc) หมุนด้วยความเร็วสูงประมาณ 5,400-10,000 รอบต่อนาที ขึ้นกับราคาและความจุซึ่งมีขนาดหลายร้อยกิกิไบต์ (GiB) จนถึงหลายเทอราไบต์ (TeraByte: TB) และสามารถเติบโตต่อเนื่องจนถึงเพتل่าไบต์ (Petabyte)
  - อุปกรณ์เก็บรักษาข้อมูลผ่านเครือข่าย (Network Storage) ซึ่งมีรูปแบบต่างๆ เช่น Network Attached Storage (NAS), Storage Area Network (SAN), Cloud Storage เป็นต้น

รีจิสเตอร์ แคช และหน่วยความจำภายในภาพไม่สามารถเก็บรักษาข้อมูลได้ (Volatile Memory) หากผู้ใช้ปิดเครื่อง ไฟฟ้าขัดข้อง ไฟฟ้ากระชากหรือแรงดันตกช่วงขณะ หรือแบตเตอรี่พลังงานหมด ยกตัวอย่าง เช่น คอมพิวเตอร์พกพา โทรศัพท์เคลื่อนที่สมาร์ตโฟนซึ่งต้องอาศัยพลังงานจากแบตเตอรี่เป็นหลัก

เครื่องคอมพิวเตอร์ทั้งหมดจะต้องมีอุปกรณ์เก็บรักษาข้อมูล หรือ Non-Volatile Memory ในลำดับชั้นถัดไป เพื่อเก็บรักษาไฟล์โปรแกรมและไฟล์ข้อมูลไม่ให้สูญหาย รายละเอียดเพิ่มเติมในบทที่ 7 ในทำนองเดียวกัน เครื่องคอมพิวเตอร์ที่ต้องทำงานตลอดเวลา เช่น เว็บไซต์ต่างๆ เครื่องเซิร์ฟเวอร์ และเครื่องซูเปอร์คอมพิวเตอร์ ควรมีแหล่งจ่ายไฟฟ้าสำรองหรือ UPS (Uninterrupted Power Supply) เช่นเดียวกัน และหากไฟดับเป็นเวลานาน อาจต้องมีเครื่องปั่นไฟสำรอง

หน่วยความจำหลักมีความจุขนาดใหญ่แต่ใช้เวลาเข้าถึงข้อมูลนานกว่าแคชและรีจิสเตอร์ ในขณะที่แคชมีความจุร่องลงมา และรีจิสเตอร์มีความจุน้อยที่สุดแต่ใช้เวลาเข้าถึงข้อมูลน้อยที่สุด เวลาเข้าถึงข้อมูล เป็นปัจจัยสำคัญในการออกแบบสถาปัตยกรรมของคำสั่ง โดยเฉพาะสถาปัตยกรรมโหลด/สโตร์ การประมวลผลทางคณิตศาสตร์และตรรกศาสตร์ค่าของตัวแปรในหน่วยความจำหลัก จะต้องอาศัยขบวนการอ่าน (Load) ข้อมูลจากแคชหรือหน่วยความจำมาพกเก็บในรีจิสเตอร์ก่อน แล้วจึงนำค่าในรีจิสเตอร์ไปประมวลผลแล้วพกเก็บเพื่อประมวลผลต่อไป เมื่อคำนวณแล้วเสร็จโปรแกรมจึงทำการเขียนหรือสโตร์ (Store) ค่าเก็บในหน่วยความจำหลักตามที่อธิบายไว้ในบทที่ 4

กล่าวโดยสรุปคือ รีจิสเตอร์อยู่ใกล้กับ ALU ใช้ เวลาเข้าถึง สั้นกว่าเสมอ (เร็ว) แต่จะมีความจุน้อย (เล็ก) เนื่องจากความซับซ้อนในการออกแบบและต้นทุนต่อความจุที่สูงกว่า การจัดแบ่งระดับชั้นต่างๆ ของหน่วยความจำอาศัยหลักการใช้งานขึ้นในแกนเวลา (**Temporal Locality**) จากการวนรอบหรือลูป และในพื้นที่ใกล้เคียง (**Spatial Locality**) ที่มา: [Harris and Harris \(2013\)](#) ในรูปของตัวแปรอาร์เรย์ หลักการทั้งหมดนี้ทำให้โปรแกรมมองเห็นหน่วยความจำขนาดใหญ่ในรูปของเวอร์ชวลเม莫รีที่มีต้นทุนต่ำ แต่มีเวลาเข้าถึงเฉลี่ยที่ไม่นานจนเกินไป

## 5.2 เวอร์ชวลเม莫รีชนิดเพจ (Paging Virtual Memory)

### 5.2.1 หลักการพื้นฐานของเวอร์ชวลเม莫รี

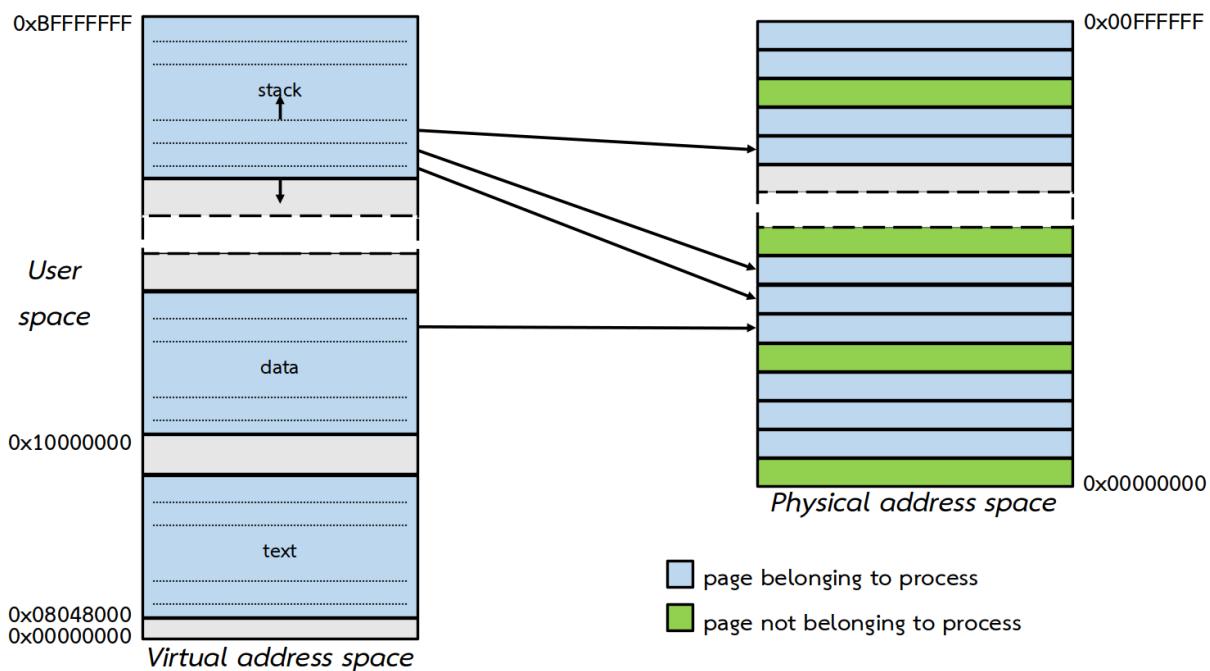
ระบบปฏิบัติการสามารถใช้ประโยชน์จากชีพียูที่รองรับการสร้างเวอร์ชูลเม莫รีได้ โดยให้ฮาร์ดแวร์ในชีพียูช่วยทำหน้าที่แปลงเวอร์ชูลแอดเดรส (Virtual Address) ให้เป็นแอดเดรสกายภาพ (Physical Address) เวอร์ชูลเม莫รีมีหน้าที่และลักษณะสำคัญดังนี้

- รองรับความต้องการความจุของหน่วยความจำภายในภาพมากกว่า 4 กิกะไบต์ (GiB) ด้วยการใช้งานระบบปฏิบัติการเวอร์ชัน 64 บิต
- บริหารจัดการความเร็วที่แตกต่างระหว่างหน่วยความจำภายในภาพหรือ SDRAM และอุปกรณ์เก็บรักษาข้อมูลซึ่งเป็นฮาร์ดดิสก์ชนิดจานหมุนตั้งแต่ในอดีต ถึงแม้หน่วยความจำแฟลช (หัวข้อที่ 7.2) จะมีความเร็วสูงขึ้นและแพร่หลายมากขึ้นในปัจจุบันและต่อไปในอนาคต

จากหัวข้อที่ 3.3.2 ระบบปฏิบัติการโหลดโปรแกรมหรือซอฟต์แวร์ประยุกต์จากไฟล์รูปแบบ ELF เพื่อเตรียมตัวรันซอฟต์แวร์นั้น ศัพท์ทางเทคนิคเรียกโปรแกรมนั้นว่า **procress** (Process) หมายถึง โปรแกรมที่ระบบปฏิบัติการอนุญาตให้ชีพียูรัน โดยจะมีเวอร์ชูลเม莫รีของตนเองด้านซ้ายของรูปที่ 5.2 ซึ่งคล้ายกับรูปที่ 3.16 เวอร์ชูลเม莫รีของprocressใดๆ มีโครงสร้างตามรายละเอียดที่ได้อธิบายแล้วในหัวข้อที่ 3.3.3 เวอร์ชูลเม莫รีของprocressนั้นตัวมีขนาดใหญ่กว่าความจุของหน่วยความจำภายในภาพ ทำให้ผู้เขียนโปรแกรมสามารถพัฒนาโปรแกรมได้อย่างอิสระมากขึ้น ไม่ต้องกังวลกับขนาดของหน่วยความจำภายในภาพที่มีอยู่จริง

รูปที่ 5.2 แสดงพื้นที่ยูสเซอร์สเปชความจุขนาด 3 กิกะไบต์ (GiB) หรือประมาณ 3 พันล้านไบต์ ซึ่งถูกแบ่งเป็นพื้นที่อยู่ขนาด 4 KiB (kibibyte) หรือ 4096 ไบต์ เรียกว่า **เพจ** (Page) ทั้งนี้ขึ้นอยู่กับรายละเอียดของฮาร์ดแวร์และระบบปฏิบัติการ ระบบลินุกซ์กำหนดให้แต่ละเพจมีขนาด 4 KiB (kibibyte) หมายเหตุ เส้นประในเวอร์ชูลเม莫รี คือ รอยต่อระหว่างเพจของเวอร์ชูลเม莫รี การแบ่งเวอร์ชูลเม莫รีและหน่วยความจำภายในภาพออกเป็นเพจจึงเป็นเรื่องที่เหมาะสม และง่ายต่อการบริหารจัดการ ยกตัวอย่างเช่น

ระบบปฏิบัติการจะจองพื้นที่หน่วยความจำภายในภาพให้แต่ละprocressเท่าที่ใช้งานจริง มีได้จองพื้นที่ทั้งหมด 3 กิกะไบต์ (GiB) ซึ่งสิ่ฟ้าในหน่วยความจำภายในภาพ (ทางด้านขวาของรูปที่ 5.2) คือ เพจที่ใช้งานจริงของprocress ทางด้านซ้ายที่ระบบปฏิบัติการจองในหน่วยความจำภายในภาพ ดังนั้น เครื่องคอมพิวเตอร์จึงมีความจุพื้นที่ภายในภาพเหลือแบ่งให้กับprocressอื่นๆ ซึ่งที่เป็นสีเขียว คือ เพจที่ระบบจองให้กับprocressอื่นๆ ซึ่งระบบปฏิบัติการที่รองรับการทำงานแบบนี้เรียกว่า **มัลติทาสกิ้ง** (Multitasking) ระบบปฏิบัติการจะเปิดโอกาสให้procressหลายๆ ตัวใช้งานหน่วยความจำภายในภาพได้พร้อมๆ กัน รายละเอียดเพิ่มเติมที่ [wikipedia](#) ส่วนซึ่งที่เป็นสีขาว หมายถึง เพจที่ว่าง เนื่องจากระบบปฏิบัติการยังไม่ได้จองเพจนี้ให้กับprocressใดๆ



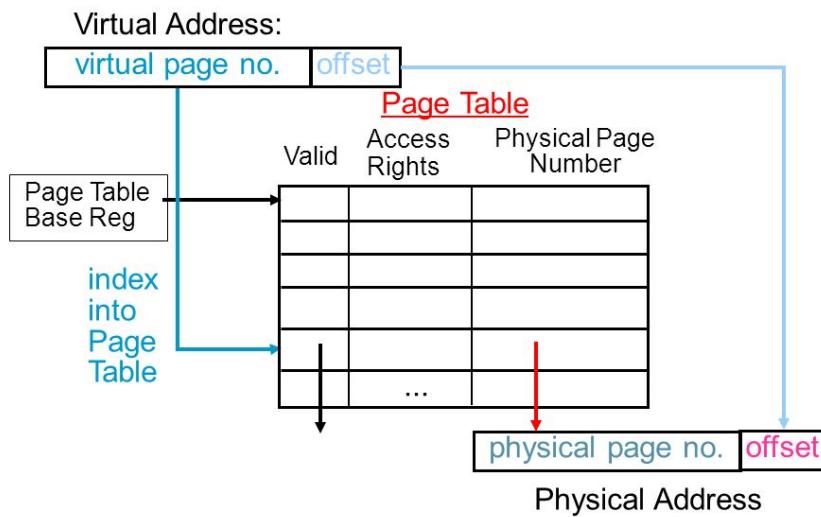
รูปที่ 5.2: การแมป (Map) เวอร์ชวลแอดдресส์ (Virtual Address) ขนาด 3 กิกะไบต์ (GiB) เป็นแอดเดรสกายภาพ (Physical Address) ขนาด 64 เมบิไบต์ (MiB) ตามหลักการเวอร์ชวลเมมโมรีชันดิเพจ (Paging Virtual Memory) หมายเหตุ เส้นประ คือ รอยต่อระหว่างเพจของเวอร์ชวลเมมโมรี

การที่ระบบปฏิบัติการรองรับการทำมัลติทาสกิنجึงนี้และมีจำนวนโปรเซสเพิ่มขึ้นจนหน่วยความจำภายใน Pam ฟื้นที่ว่างลดลง ระบบจะต้องอาศัยพื้นที่กันไว้ในอุปกรณ์เก็บรักษาข้อมูลเป็นที่พักเก็บเพจต่างๆ ชั่วคราว เราเรียกพื้นที่ส่วนนี้ในระบบลินิกซ์ว่า **swap partition** โดยเครื่องเนลจะย้ายเพจที่ไม่ค่อยได้ใช้งานหรือลับจากหน่วยความจำภายใน Pam มาพักเก็บที่นี่ เพื่อให้หน่วยความจำภายใน Pam มีพื้นที่ว่างพอที่ระบบปฏิบัติการจะจองให้กับโปรเซสที่ต้องการพื้นที่เว่อร์ชัลเมโมรี่เพิ่มขึ้น เช่น ในพื้นที่ซีปเซกเมนต์ หรือให้กับโปรเซสใหม่

แอ็ดเดรสของเวอร์ชัลเมโมรีชนิดเพจ เรียกว่าๆ ว่า เวอร์ชัลแอดเดรส (Virtual Address) แบ่งเป็น 2 ส่วน คือ หมายเลขเวอร์ชัล (Virtual Page Number) ความยาว 20 บิต และหมายเลขอффเซต (Offset) ความยาว 12 บิต ซึ่งคำนวณจากขนาดของเพจ  $2^{12}=4096$  ไปต์ ระบบปฏิบัติการจะจดจำพื้นที่แต่ละเพจในหน่วยความจำภายในภาพ เรียกว่า เพจเฟรม (Page Frame) หรือย่อๆ ว่า เพจ เช่นกัน รูปที่ 5.3 แสดงการแปลงหมายเลขเวอร์ชัล เป็นหมายเลข物理ภาพ ส่วนที่เป็นเพจอффเซตสามารถนำมาร่วมกับหมายเลขเพจภายในภาพได้เป็น แอ็ดเดรสภายนอก (Physical Address) เพื่อนำไปอ้างถึงคำสั่งหรือข้อมูลในแคชลำดับต่างๆ

คอมพิวเตอร์ทั่วไปใช้หน่วยความจำภายในจากเทคโนโลยี SDRAM ซึ่งบางตัวเรียกว่า หน่วยความจำหลักเนื่องจากเทคโนโลยี SDRAM มีประสิทธิภาพสูงกว่า และมีต้นทุน (ต่อความจุหนึ่งหน่วย) ต่ำกว่าเทคโนโลยีหน่วยความจำ SRAM แต่ต้นทุนสูงกว่าอุปกรณ์เก็บรักษาข้อมูล ดังนั้น การอ่านหรือเขียนอุปกรณ์เก็บรักษาข้อมูลจำเป็นต้องอ่านหรือเขียนครั้งละมากๆ เนื่องจากเวลาเข้าถึง ที่ต้องรอเพื่ออ่านคำสั่ง และถ่ายโอน (อ่านหรือเขียน) ข้อมูลไปมาระหว่างหน่วยความจำหลักและอุปกรณ์เก็บรักษาข้อมูล รายละเอียดสามารถอ่านเพิ่มเติมในบทที่ 7

## Address Mapping: Page Table

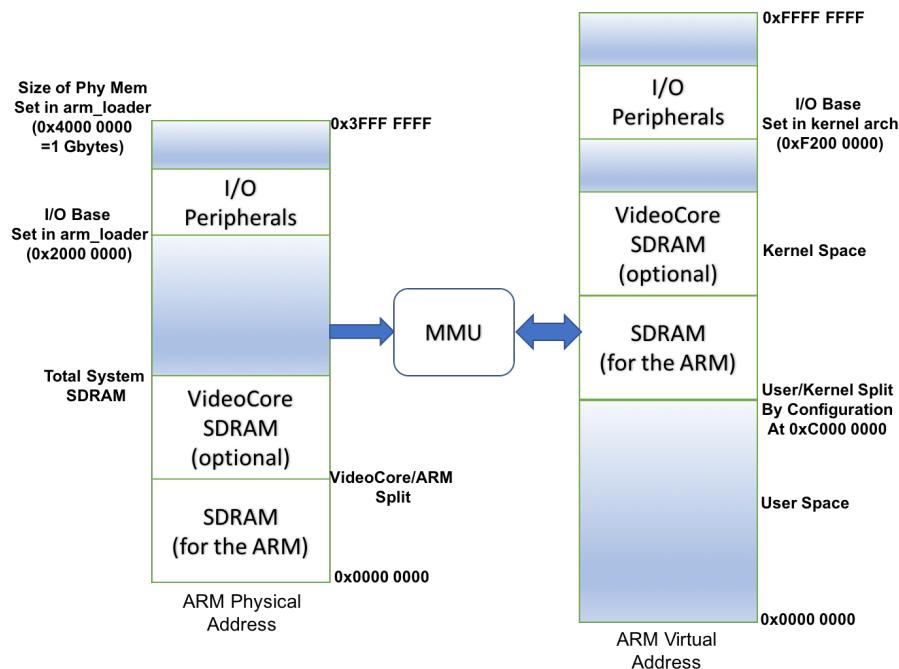


รูปที่ 5.3: การแปลงหมายเลขเวอร์ชวล (Virtual Page Number) เป็นหมายเลขกายภาพ (Physical Page Number) ที่มา: [slideplayer.com](http://slideplayer.com)

### 5.2.2 เวอร์ชวลเมโมรีชันดิเพจ กรณีศึกษาบอร์ด Pi3/Pi4

ระบบปฏิบัติการต่างๆ สำหรับบอร์ด Pi3/Pi4 ทำงานตามหลักการเวอร์ชวล เมมโมรีชันดิเพจ เช่นกัน ดังนั้น การแปลงเวอร์ชูลแอดเดรสเป็นแอดเดรสกายภาพของชิปประภุล BCM2835-BCM2837/BCM2711 และ BCM2711 มีความคล้ายคลึงกันกับรูปที่ 5.4 โดยมีฮาร์ดแวร์ เรียกว่า **MMU** (Memory Management Unit) ทำหน้าที่แปลง (Address Translation) เวอร์ชูลแอดเดรสเป็นแอดเดรสกายภาพ MMU จะทำงานร่วมกับเพจテー�เบิลซึ่งได้อธิบายรายละเอียดไปแล้วในหัวข้อก่อนหน้า ดังนี้

- เนื่องจากบอร์ด Pi1 ถึง Pi3/Pi4 ใช้ระบบปฏิบัติการลินุกซ์ ขนาด 32 บิตเดียว กัน พื้นที่เวอร์ชูลเมมโมรีของทุกๆ processor ทางด้านขวาของรูปมีขนาดเท่ากัน คือ 4 กิกะไบต์ (GiB) โดยแบ่งเป็น
  - พื้นที่ขนาด 1 กิกะไบต์ (GiB) สำหรับเก็บคำสั่งและข้อมูลของคอร์เนล (Kernel) ในการบริหารจัดการเครื่อง เรียกว่า **Kernel Space** โดยเวอร์ชูลแอดเดรสของ Kernel เริ่มต้นที่หมายเลข 0xC000 0000 - 0xFFFF FFFF แบ่งเป็น
    - \* พื้นที่ซึ่งเรียกว่า **I/O Peripherals** เริ่มต้นที่หมายเลข 0xF200 0000 แมพไปอยู่ที่แอดเดรสกายภาพ 0x2000 0000
    - \* พื้นที่ซึ่งเรียกว่า **VideoCore** คือ พื้นที่สำหรับจีพียูของชิป BCM2837/BCM2711 บนบอร์ด Pi3/Pi4 จะถูกกำหนดขนาดในไฟล์ start.elf ตั้งอยู่ในพาร์ติชัน boot ซึ่งฟอร์แมตด้วยรูปแบบ FAT32 โดยขนาดหน่วยความจำที่กำหนดให้จีพียูขั้นต่ำที่สุด ขนาด 32 เมบิไบต์ แต่ความละเอียดของการแสดงผลบนจอ 1080p30 ผ่านสาย HDMI ต้องการหน่วยความจำขนาด 64 เมบิไบต์ สำหรับหน้าที่เป็นบัฟเฟอร์สำหรับการแสดงผล รายละเอียดเพิ่มเติมในการทดลองที่ 9 ในภาคผนวก I
    - \* พื้นที่ **SDRAM (for the ARM)** ซึ่งแมพไปอยู่ที่แอดเดรสกายภาพ 0x0000 0000 เป็นพื้นที่สำหรับบรรจุโปรแกรมคอร์เนลสำหรับบริหารจัดการระบบทั้งหมด



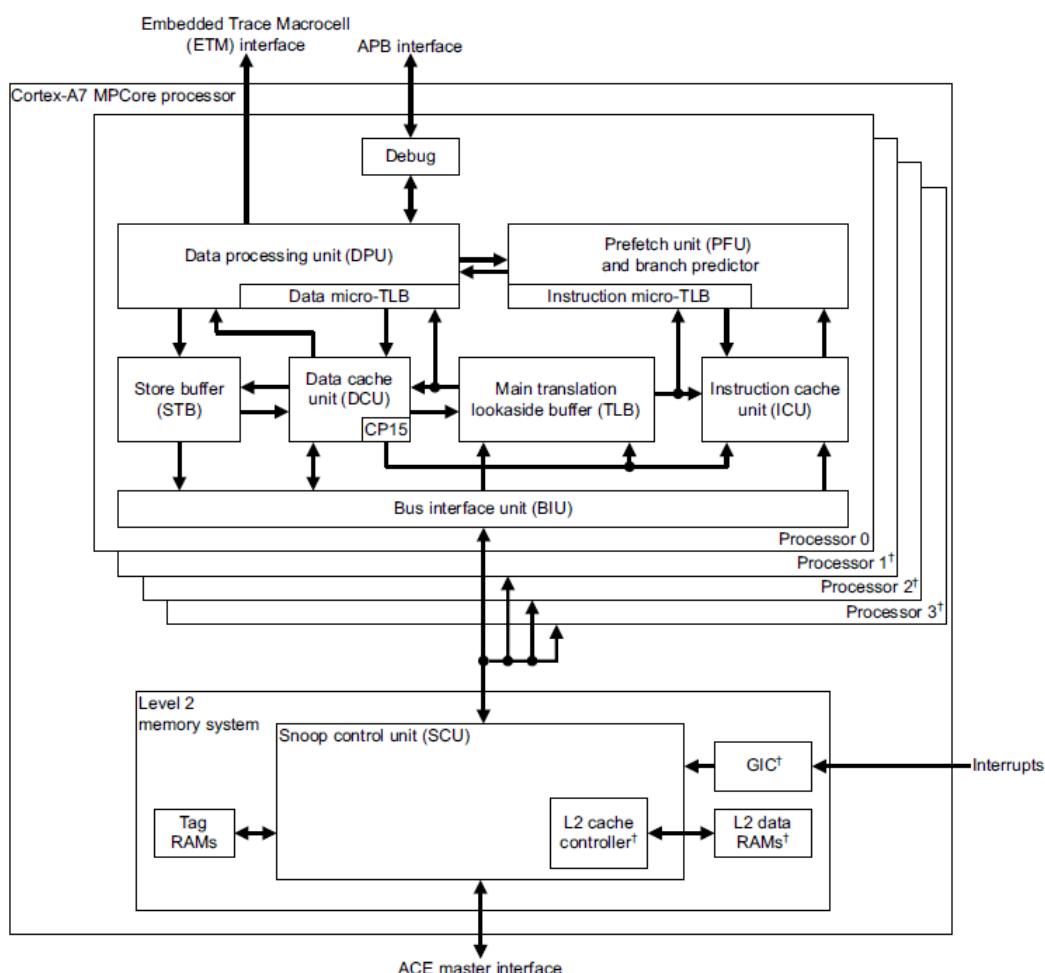
รูปที่ 5.4: โครงสร้างของเวอร์ชัลเม莫เรี่ยนนิดเพจของบอร์ด Pi3/Pi4 โมเดล B ซึ่งใช้ชิปตระกูล BCM283x, x=5, 6, 7 หมายเหตุ ขนาดของรูปไม่เป็นไปตามพื้นที่ตามความเป็นจริง, MMU: Memory Management Unit ที่มา: [Broadcom \(2012\)](#)

- พื้นที่ขนาด 3 กิกะไบต์ (GiB) เรียกว่า **ยูสเซอร์สเปซ** แบ่งเป็นเซกเมนต์ต่างๆ สำหรับเก็บคำสั่งและข้อมูลของแต่ละโพรเซสคล้ายกับรูปที่ 3.16 เริ่มต้นที่เวอร์ชัลแอดдресจากหมายเลข 0x0000 0000 จนถึง 0xBFFF FFFF โดยขนาดของยูสเซอร์สเปซและคอร์เนลสเปซ (User/Kernel Split) กำหนดอยู่ในไฟล์ config.txt ซึ่งในรูปคือ ที่หมายเลข 0xC000 0000 และในหัวข้อที่ 3.3.1
- พื้นที่หน่วยความจำภายในภาพของบอร์ด Pi3/Pi4 อยู่ทางด้านซ้ายของรูป ขนาดความจุรวมเท่ากับ 1 กิกะไบต์ (GiB) โดยมีแอดдресภายในเริ่มต้นที่หมายเลข 0x0000 0000 - 0x3FFF FFFF แบ่งเป็นพื้นที่สำคัญดังนี้
  - **SDRAM (for the ARM)** คอร์เนลจะใช้งานเพียงผู้เดียวสำหรับจัดเก็บเพจเทเบิล พื้นที่ส่วนนี้รวมกับ VideoCore SDRAM ถูกกำหนดขอบเขตโดย Total System SDRAM
  - **พื้นที่สำหรับการแสดงผล** ขนาด 32 เมกะไบต์ (MiB) โดยสามารถตรวจสอบขนาดที่แท้จริงได้ในกราฟิกบอร์ดที่ 4 ภาคผนวก D
  - **พื้นที่สำหรับเข้ารหัสอุปกรณ์อินพุต/เอาต์พุต** ซึ่งไม่ผ่านแคช (Uncached) เริ่มต้นที่แอดdress 0x2000 0000 ตามหลักการ Memory Mapped I/O ซึ่งจะกล่าวต่อไปในหัวข้อ 6.9
  - **พื้นที่อื่นๆ** ที่เหลือในรูปเป็นเขตสีฟ้า คอร์เนลจะบริหารจัดการโดยแบ่งให้โปรแกรมต่างๆ ใช้งันร่วมกัน

ผู้อ่านสามารถค้นควารายละเอียดด้านต่างๆ ของบอร์ด Pi3/Pi4 เพิ่มเติมได้ที่ [github.com](https://github.com) กรณีศึกษาที่พิจารณา ARM Cortex A53/A72 บนบอร์ด Pi3/Pi4 พัฒนาต่อยอดจากชิปปี้ ARM Cortex A7 ซึ่งทั้งคู่จัดเป็นชิประดับต้น (Entry Level) สำหรับโทรศัพท์เคลื่อนที่สมาร์ตโฟน ผู้อ่านสามารถทำความเข้าใจการทำงานของเวอร์ชัลเม

โมรีของซีพียู ARM Cortex A53/A72 จากซีพียู ARM Cortex A7 จากรูปที่ 5.5 เนื่องจาก Cortex A53/A72 พัฒนาต่อจาก Cortex A7

การแปลงหมายเลขเวอร์ชวลเป็นหมายเลข物理ภายในช่วงเวลาที่ต้องอาศัยเพจเทเบิลตามที่อธิบายไปแล้วก่อนหน้า โดยมี TLB (Translation Lookaside Buffer) ที่มา: [Tanenbaum and Austin \(2012\)](#); [Harris and Harris \(2013\)](#) ทำหน้าที่เป็นเพจเทเบิลขนาดเล็กอยู่ติดกับแกนประมวลผลและแคชลำดับที่ 1 ทั้งนี้ TLB แบ่งเป็น ITLB และ DTLB ซึ่งใช้หน่วยความจำสแตติกแรม เช่นเดียวกับแคชทั่วไป และแบ่งเป็นลำดับชั้นเหมือนกับแคชลำดับที่ 1 โดย ITLB ย่อมาจากคำว่า Instruction TLB หมายถึง TLB ลำดับที่ 1 แปลงหมายเลขเวอร์ชูลที่คำสั่งภาษาเครื่องอยู่ในพื้นที่ทึ่กซึ่งเช็คเมนต์เท่านั้นให้เป็นหมายเลข物理 และ DTLB ย่อมาจาก Data TLB หมายถึง TLB ลำดับที่ 1 ทำหน้าที่แปลงหมายเลขเวอร์ชูลที่เก็บหมายเลขเพจซึ่งอยู่ใน เช็คเมนต์อื่นๆ และ TLB ลำดับที่ 2 จะรวมหมายเลขเวอร์ชูลของทุกเช็คเมนต์เข้าด้วยกัน คล้ายกับแคชลำดับที่ 1 และแคชลำดับที่ 2 ซึ่งได้กล่าวแล้วในหัวข้อที่ 5.1



รูปที่ 5.5: โครงสร้างของชิป ARM Cortex A7 ประกอบด้วย TLB และแคชหลายระดับเพื่อรับรองการทำงานของเวอร์ชูลเมมโมรีชนิดเพจ ที่มา: [ARM \(2011\)](#) และ [arm.com](#)

โครงสร้างของชิป ARM Cortex A7 ในรูปที่ 5.5 ประกอบด้วย Instruction micro-TLB เทียบเท่า ITLB ลำดับที่ 1, Data micro-TLB เทียบเท่า DTLB ลำดับที่ 1, Main TLB เทียบเท่า TLB ลำดับที่ 2, Instruction Cache Unit (ICU) เทียบเท่าแคชคำสั่งลำดับที่ 1, Data Cache Unit เทียบเท่าแคชข้อมูลลำดับที่ 1, แคชลำดับที่ 2 และหน่วยความจำหลัก SDRAM เพื่อรับรองการทำงานของเวอร์ชูลเมมโมรีชนิดเพจ

ผู้อ่านสามารถเรียกดูรายละเอียดของการทำงานของการเพจซึ่งอยู่ในรูปที่ 5.5 เพื่อนำคำสั่งไปอุดรหัส และประมวลผล

ในไปป์ไลน์ ดังนี้

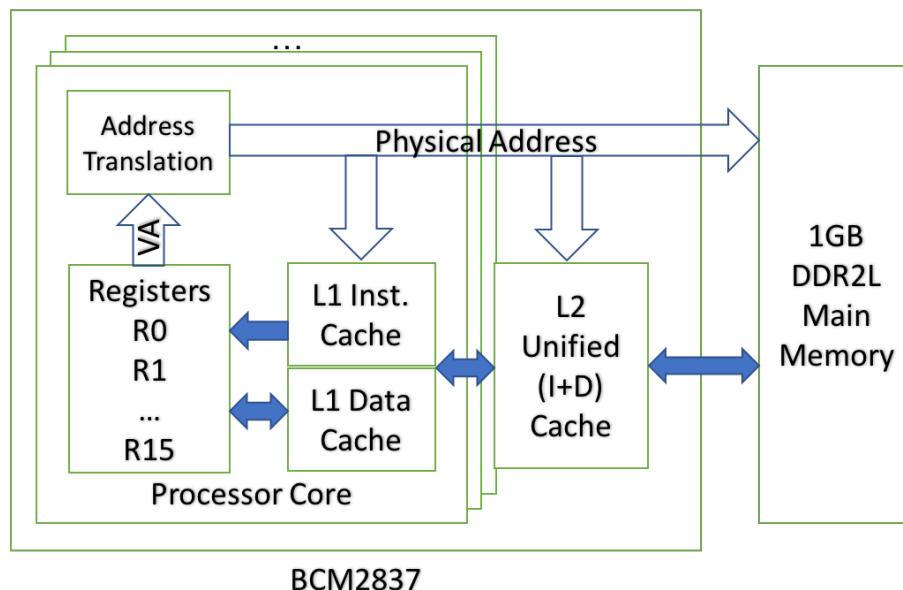
1. นำเลขเพจเวอร์ชวลใน PC (Program Counter) ไปสืบค้นหาค่าเลขเพจ กายภาพใน Instruction micro-TLB

- หากเจอ เรียกว่า TLB ฮิต (Hit) จะใช้เวลาส่งค่าหมายเลขเพจกายภาพไปใช้งานสั้นมากๆ (0.5-1 นาโนวินาที)
- หากไม่เจอ เรียกว่า TLB มิส (Miss) วงจรจะนำหมายเลขเพจเวอร์ชวลไปสืบค้นใน Main TLB ต่อไปซึ่งจะใช้เวลาเพิ่มขึ้นอีก
  - หากฮิต ส่งค่าหมายเลขเพจกายภาพไปใช้งาน (2-4 นาโนวินาที)
  - หากมิสอีกรอบ จะต้องสืบค้นในเพจเทเบิลเป็นลำดับสุดท้ายซึ่งจะใช้เวลานานขึ้น
    - \* หากฮิต ส่งค่าหมายเลขเพจกายภาพจากเพจเทเบิลไปใช้งาน (หลักสิบนาโนวินาที)
    - \* หากมิสอีกรอบ เรียกว่า เพจฟอลท์ (Page Fault) แสดงว่าเครื่องเนลยังไม่ได้อ่านคำสั่งที่ต้องการจากเพจในอุปกรณ์เก็บรักษาข้อมูลมาบรรจุในหน่วยความจำกายน้ำซึ่งจะใช้เวลานานที่สุด (เกิน 100 นาโนวินาที) รายละเอียดเพิ่มเติมที่ [Tanenbaum and Bos \(2014\)](#) และ [wikipedia](#)

2. นำเลขเพจกายภาพมาเชื่อมกับค่าออฟเซตเพื่อนำไปสืบค้นคำสั่งใน ICU ซึ่งเทียบเท่าแคชคำสั่งลำดับที่ 1

- หากเจอ เรียกว่า เกิดแคชฮิตที่ลำดับที่ 1 (L1 Cache Hit) และนำคำสั่งส่งกลับไปยังซีพียูต่อไปซึ่งจะใช้เวลาสั้นที่สุด (0.5 - 1 นาโนวินาที)
- หากไม่เจอ เรียกว่า เกิดแคชมิสที่ลำดับที่ 1 (L1 Cache Miss) นำแอ็ดเดรสกายภาพไปค้นหาคำสั่งในแคชลำดับที่ 2
  - หากฮิต ที่ลำดับที่ 2 (L2 Cache Hit) และนำคำสั่งส่งกลับไปยังซีพียูและแคชลำดับที่ 1 ซึ่งจะใช้เวลานานขึ้น (2 - 4 นาโนวินาที)
  - หากมิส ที่ลำดับที่ 2 (L2 Cache Miss) วงจรจะนำแอ็ดเดรสกายภาพไปค้นหาคำสั่งในหน่วยความจำหลัก (SDRAM) ผ่านทางจรวจเชื่อมต่อกับหน่วยความจำ SDRAM ซีพียูจะต้องใช้เวลารอ (นานกว่าสิบนาโนวินาที) เพื่อที่คำสั่งจะเดินทางมาจากหน่วยความจำกายน้ำซึ่งต้องมาจากแคชลำดับต่างๆ และจาก SDRAM จะเป็นเลขฐานสองจำนวน 4-8 คำสั่ง คิดเป็นความยาว 128-256 บิต ขึ้นอยู่กับความกว้างของแคชแต่ละบล็อก ซึ่งจะอธิบายในหัวข้อถัดไป

### 5.3 หน่วยความจำแคช (Cache Memory)



รูปที่ 5.6: โครงสร้างเชิงตรรกะเชื่อมโยงระหว่างรีจิสเตอร์ แคชลำดับที่ 1 และ 2 ภายในชิป BCM2837/BCM2711 และหน่วยความจำหลักภายนอก หมายเหตุ VA: Virtual Address

รูปที่ 5.6 แสดงโครงสร้างเชิงตรรกะเชื่อมโยงระหว่างรีจิสเตอร์ แคชลำดับที่ 1 และ 2 และหน่วยความจำภายในภาพ ประกอบด้วย วงจรแปลงเวอร์ชัลแอดเดรส (VA) ซึ่งหมายถึง TLB และเพจเทเบิล ให้เป็นแอดเดรสภายในภาพ (PA: Physical Address) ส่งผ่านทางเส้นทีบสีขาว (Bus) ไปยังแคชลำดับที่ 1 ซึ่งแบ่งเป็นแคชคำสั่งและแคชข้อมูล (Data Only) แอดเดรสภายในภาพจะส่งไปยังแคชลำดับที่ 2 และหน่วยความจำภายในภาพเข่นกัน วงจรเหล่านี้อ้างอิงคำสั่งและข้อมูลด้วยแอดเดรสภายในภาพ คำสั่งและข้อมูลจะวิ่งผ่านเส้นทีบสีน้ำเงิน (Bus) กลับมายังรีจิสเตอร์ภายในแกนประมวลผล โดยจะเริ่มจากซีพียูเฟลช์คำสั่งจากแคชคำสั่ง (Instruction Cache) ลำดับที่ 1 ก่อน หากไม่เจอจะค้นคำสั่งนั้นจากแคชลำดับที่ 2 ที่ถัดไป หากคำสั่งนั้นเป็นคำสั่งประเภท LOAD หรือ คำสั่ง STORE ซีพียูปฏิบัติตาม (Execute) จะค้นหาเพื่ออ่าน/เขียนข้อมูลในแคชข้อมูล (Data Cache) ลำดับที่ 1 ก่อนเข่นกัน แคชลำดับที่ 2 เป็นแบบรวม (Unified Cache) ทำหน้าที่พักเก็บคำสั่งและข้อมูล ซึ่งได้อธิบายโดยละเอียดในหัวข้อก่อนหน้าแล้ว

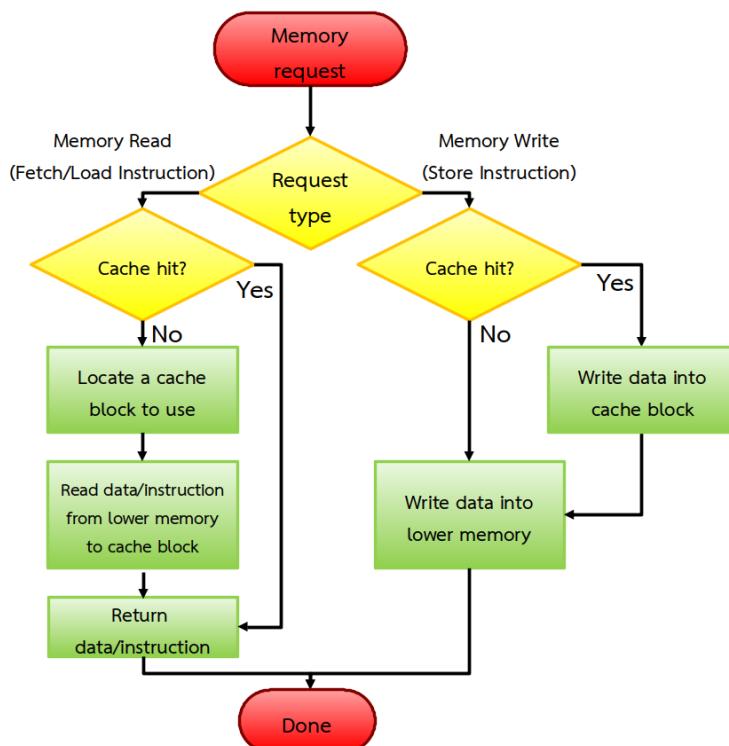
แคชทั้งสองลำดับอยู่บนชิปเดียวกับซีพียูช่วยลดเวลาการเข้าถึงข้อมูล ในขณะที่ หน่วยความจำหลักหรือหน่วยความจำภายในภาพสร้างเทคโนโลยีหน่วยความจำชนิด SDRAM ทำให้เวลาการเข้าถึงคำสั่งและข้อมูลนานขึ้น และมักติดตั้งอิสระจากซีพียู (Off-chip) ทำให้เพิ่มเวลาการเดินทางของคำสั่งและข้อมูล ซึ่งจะได้กล่าวในหัวข้อที่ 5.5 ต่อไป

แคชจึงทำหน้าที่คล้ายกับบัฟเฟอร์เก็บพักคำสั่งหรือข้อมูลที่อ่านจากหน่วยความจำหลัก เพื่อการทำงานของซอฟต์แวร์ส่วนหนึ่งจะเป็นการวนรอบ เช่น การวนรอบชนิดต่างๆ ในหัวข้อที่ 4.7.3 - 4.7.6 ทำให้ใช้คำสั่งเดิมๆ ใน Loop Body ซ้ำแล้วซ้ำอีก เรียกว่า **Temporal Locality** ดังนั้น การพักเก็บคำสั่งหรือข้อมูลที่มีการเรียกใช้บ่อยในแคชลำดับต่างๆ เป็นประโยชน์ต่อระบบโดยภาพรวม เนื่องจากช่วยลดเวลาการเข้าถึงหน่วยความจำ SDRAM และเวลาเดินทางของสัญญาณจากชิปสู่ชิป (Chip-Chip Propagation Time)

การอ่านหรือเขียนค่าตัวแปรทุกชนิดในเวอร์ชัลเมโมรีทุกตำแหน่งหรือทุกหมายเลขแอดเดรสจะต้องผ่านแคชเสมอ ดังนั้น แคชจะทำงานตามโฟล์ชาร์ตในรูปที่ 5.7 เริ่มต้นจากการแปลงจากแอดเดรสเวอร์ชัลเป็นแอดเดรสภายในภาพด้วย TLB และเพจเทเบิลเสมอ การทำงานการเฟลช์คำสั่งจากแคชคำสั่งหรือการโหลดข้อมูล

จากเดชข้อมูลจะอยู่ด้านซ้ายของรูป การเขียนข้อมูลลงในเดชข้อมูลจะอยู่ด้านขวาของรูป การตรวจสอบว่า แคชชิต หรือ แคชมิส ขึ้นอยู่กับรายละเอียดของเดชชนิดต่างๆ ซึ่งจะอธิบายในหัวข้อถัดไป

- การอ่านคำสั่ง/ข้อมูล พบในเดชหรือไม่
  - หากใช่ เรียกว่า **แคชชิต** แสดงว่าแคชมีคำสั่งหรือข้อมูลแล้ว แคชจะส่งคำสั่งหรือข้อมูลนั้นให้กับเดช ลำดับสูงกว่าต่อไปจนถึงที่พี่ยุ
  - หากไม่ เรียกว่า **แคชมิส**
- การเขียน (Store) ข้อมูล พบในเดชหรือไม่
  - หากใช่ เรียกว่า **แคชชิต** แสดงว่าแคชมีข้อมูลแล้ว บล็อกข้อมูลนั้นจะถูกค่าใหม่เขียนทับลงไป
  - หากไม่ เรียกว่า **แคชมิส** วงจรจะเขียนข้อมูลในเดชลำดับถัดไป หรือ หน่วยความจำภายนอก



รูปที่ 5.7: โพล์ชาร์ตการทำงานของเดชหนึ่งลำดับชั้นสำหรับการเฟทธ์คำสั่งและสำหรับการอ่าน/เขียนข้อมูล

การอ่าน/เขียนข้อมูลโดยเฉพาะข้อมูลชนิดอาร์เรย์ ซึ่งมีการจัดเรียงคล้ายกับรูปที่ 2.11 มีรูปแบบ (Pattern) ของความต่อเนื่องกันตามลำดับ เช่น จากอาร์เรย์ตำแหน่งตั้นๆ ไปยังตำแหน่งถัดไปจากการวนรอบชนิดต่างๆ ดังนั้น การทำงานลักษณะนี้เรียกว่า **Spatial Locality** ที่มา: [Harris and Harris \(2013\)](#) ความหมายคือ เชิงพื้นที่ (Space) สอดคล้องกับการอ่าน/เขียนข้อมูลจาก SDRAM แบบ Burst Read/Burst Write รายละเอียดเพิ่มเติมในหัวข้อที่ 5.5 การทำงานซ้ำๆ ในเชิงเวลาและเชิงพื้นที่ เรียกรวมกันว่า **Locality Principle** ที่มา: [Tanenbaum and Austin \(2012\)](#)

เดชลำดับที่ 1 นิยมออกแบบด้วยเดชชนิด SA (Set Associative) ซึ่งต้องอาศัยพื้นฐานของเดชชนิด DM (Direct Map) และมีสองชุด แบ่งเป็นเดชคำสั่ง และเดชข้อมูลตามรูปที่ 5.1 เดชลำดับที่ 2 เป็นแบบรวมทำหน้าที่พักเก็บคำสั่งและข้อมูล จึงมีความจุมากกว่าเดชลำดับที่ 1 นิยมออกแบบด้วยเดชชนิด SA แต่ต่ำราคารีมีน้ำ

จะเปรียบเทียบเพียงแค่สองชนิดที่นิยมและเข้าใจง่าย โดยใช้ชุดคำสั่งเดียวกันในตารางที่ 5.1 อธิบายการทำงานของแคชคำสั่งทั้งสองชนิด หมายเหตุ แอ็ตเตอร์สกายภาพส่วนนี้ คือ ออฟเซตยาว 12 บิตของรีจิสเตอร์ PC ซึ่งไม่เปลี่ยนแปลงตามหลักการแปลงเวอร์ชวลแอ็ตเตอร์สเป็นแอ็ตเตอร์สกายภาพซึ่งได้อธิบายในรูปที่ 5.3

ตารางที่ 5.1: ตัวอย่างโปรแกรมภาษาแอสเซมบลีเพื่อประกอบการอธิบายการทำงานของแคชทั้งสองชนิด

PC	เลbel	คำสั่ง	คอมเมนท์
<b>ออฟเซต</b>			
$000_{16}$	Loop:	ADD R1,R1,1	@ R1=R1+1
$004_{16}$		ADD R2,R2,1	@ R2=R2+1
$008_{16}$		BL Adder	@ call R0 = R1+R2
$00C_{16}$		CMP R0,10	@ Compare R0 with 10
$010_{16}$		BLT Loop	@ if Less Than, PC = Loop
$014_{16}$		...	
$018_{16}$		...	
$01C_{16}$		...	
...			
$030_{16}$	Adder:	ADD R0, R1, R2	@ R0=R1+R2
$034_{16}$		BX LR	@ Return R0
...			
$FFC_{16}$		...	

ตัวอย่างโปรแกรมภาษาแอสเซมบลีในตารางที่ 5.1 บรรจุอยู่ในหน่วยความจำภายในขนาด 4 KiB (kilobyte) หรือ 4096 ไบต์ตามขนาดของเพจภายใน แต่ละเพจบรรจุคำสั่งขนาด 32 บิตหรือ 4 ไบต์ ทำให้หมายเลขออฟเซตเริ่มนับจาก  $000_{16}$ ,  $004_{16}$ ,  $00C_{16}$ ,  $010_{16}$  ไปเรื่อยๆ จนถึง  $FFC_{16}$

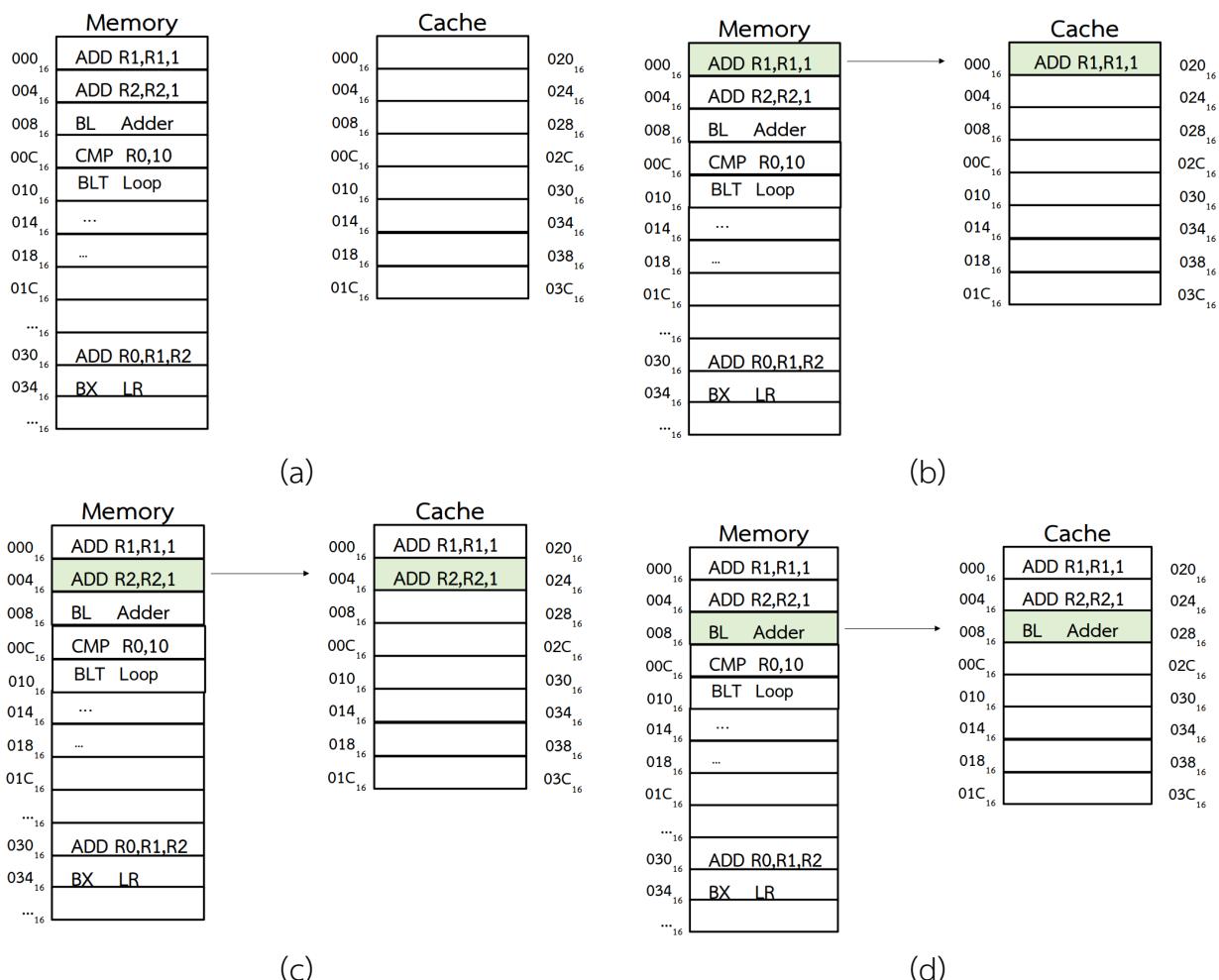
### 5.3.1 แคชชนิด DM (Direct Map)

รูปที่ 5.8 และ 5.9 แสดงการทำงานของแคชคำสั่งชนิด DM ซึ่งเป็นแคชพื้นฐานของการทำงานของแคชลำดับต่างๆ การทำงานแคชชนิดนี้มีความซับซ้อนน้อยที่สุดและคล้ายหน่วยความจำบัฟเฟอร์ ซึ่งทำหน้าที่พักเก็บคำสั่งที่ใช้บ่อยๆ แคชจำลองชนิด DM ในรูปมีขนาด 8 บล็อก แต่ละบล็อกมีความจุ 4 ไบต์ ทำหน้าที่เก็บพักคำสั่งจากแอ็ตเตอร์หมายเลข  $000_{16}$ ,  $004_{16}$  จนถึงหมายเลข  $01C_{16}$  โดยได้จากบล็อกบนสุดลงมาวนซ้ำจากแอ็ตเตอร์หมายเลข  $020_{16}$ ,  $024_{16}$  จนถึงหมายเลข  $03C_{16}$  และวนซ้ำอีกหลายรอบจนถึงแอ็ตเตอร์หมายเลข  $FE0_{16}$ ,  $FE4_{16}$  จนถึงหมายเลข  $FFC_{16}$

(a) เริ่มต้น แคชไม่มีข้อมูลใดๆ

(b) ออฟเซต  $000_{16}$  ตรงกับคำสั่ง ADD R1, R1, 1 ยังเก็บอยู่ในหน่วยความจำภายใน ซีพียูจึงตรวจสอบแคชบล็อกหมายเลข  $000_{16}$  ซึ่งว่างเปล่า เราเรียกเหตุการณ์นี้ว่า โคลด์มิส (Cold Miss) วงจรควบคุมแคชจึงนำคำสั่งนี้จากหน่วยความจำภายในไปเก็บในแคชบล็อกที่  $000_{16}$  พร้อมกับส่งให้ซีพียูอัตรัฐและ PC ออฟเซตจึงเปลี่ยนเป็นหมายเลข  $004_{16}$

(c) ออฟเซต  $004_{16}$  ตรงกับคำสั่ง ADD R2, R2, 1 ยังเก็บอยู่ในหน่วยความจำภายใน ซีพียูจึงตรวจสอบแคชบล็อกหมายเลข  $004_{16}$  ซึ่งว่างเปล่า วงจรควบคุมแคชจึงนำคำสั่งนี้จากหน่วยความจำภายในไปเก็บในแคชบล็อกที่  $004_{16}$  พร้อมกับส่งให้ซีพียูอัตรัฐและ PC ออฟเซตจึงเปลี่ยนเป็นหมายเลข  $008_{16}$



รูปที่ 5.8: การทำงานของแคชคำสั่ง (Instruction Cache) ชนิด Direct Map เมื่อซีพียูเพทซ์คำสั่งที่หน่วยความจำภายในภาพเออดเดรส (a) แคชคำสั่งยังไม่มีคำสั่งใดๆ เลย (b) PC ออฟเซตเท่ากับ 000<sub>16</sub>, (c) 004<sub>16</sub>, (d) 008<sub>16</sub>

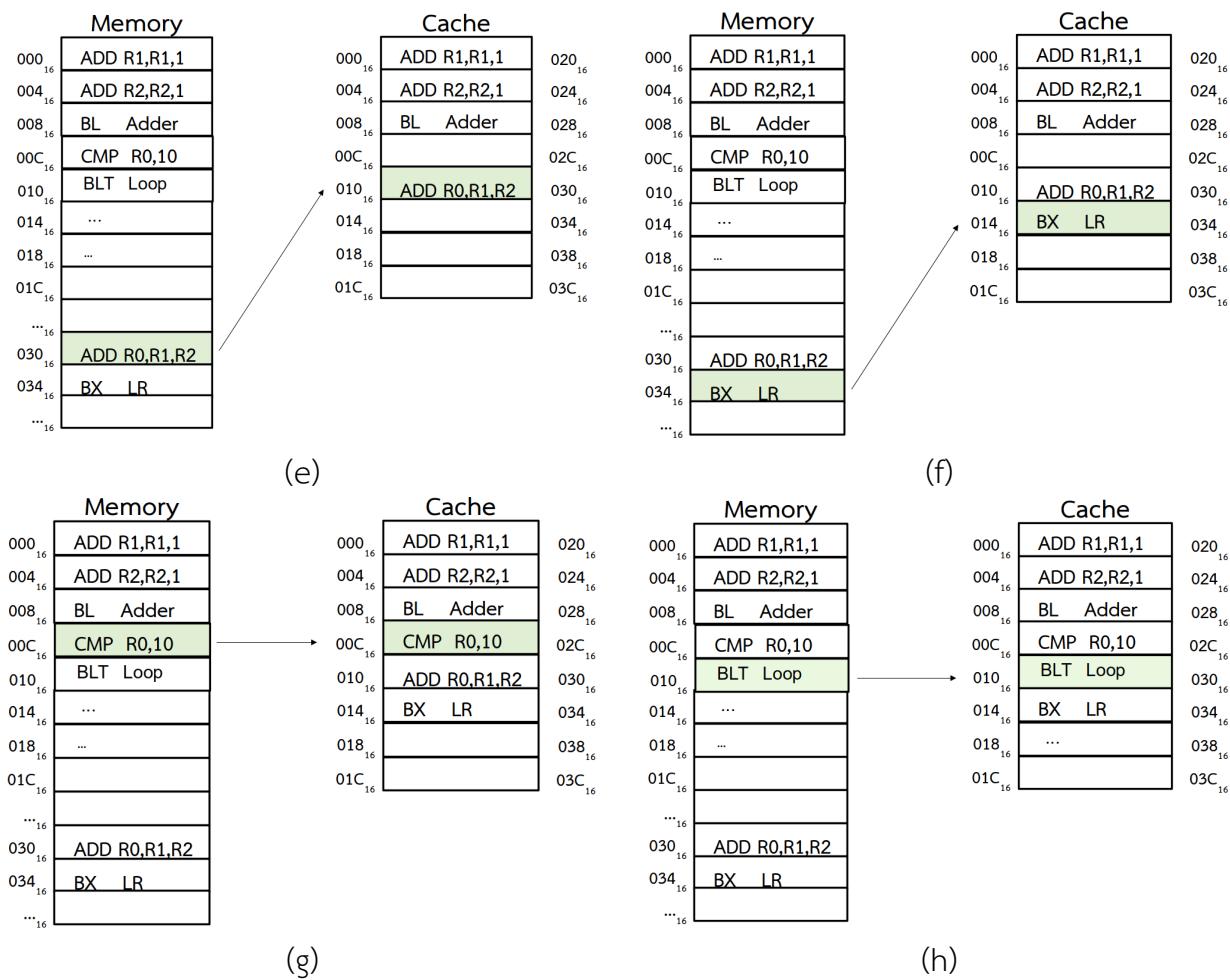
(d) ออฟเซต 008<sub>16</sub> ตรงกับคำสั่ง **BL Adder** ยังเก็บอยู่ในหน่วยความจำภายในภาพ ซีพียูจึงตรวจสอบแคชบล็อกหมายเลข 008<sub>16</sub> ซึ่งว่างเปล่า วงจรควบคุมแคชจึงนำคำสั่งนี้จากหน่วยความจำภายในภาพไปเก็บในแคชบล็อกที่ 008<sub>16</sub> พร้อมกับส่งให้ซีพียูถอดรหัสและ PC ออฟเซตจึงเปลี่ยนเป็นหมายเลข 030<sub>16</sub>

(e) ออฟเซต 030<sub>16</sub> ตรงกับคำสั่ง **ADD R0, R1, R2** ยังเก็บอยู่ในหน่วยความจำภายในภาพ ซีพียูจึงตรวจสอบแคชบล็อกหมายเลข 030<sub>16</sub> ซึ่งว่างเปล่า วงจรควบคุมแคชจึงนำคำสั่งนี้จากหน่วยความจำภายในภาพไปเก็บในแคชบล็อกที่ 030<sub>16</sub> พร้อมกับส่งให้ซีพียูถอดรหัสและ PC ออฟเซตจึงเปลี่ยนเป็นหมายเลข 034<sub>16</sub>

(f) ออฟเซต 034<sub>16</sub> ตรงกับคำสั่ง **BX LR** ยังเก็บอยู่ในหน่วยความจำภายในภาพ ซีพียูจึงตรวจสอบแคชบล็อกหมายเลข 034<sub>16</sub> ซึ่งว่างเปล่า วงจรควบคุมแคชจึงนำคำสั่งนี้จากหน่วยความจำภายในภาพไปเก็บในแคชบล็อกที่ 034<sub>16</sub> พร้อมกับส่งให้ซีพียูถอดรหัสและ PC ออฟเซตจึงเปลี่ยนเป็นหมายเลข 00C<sub>16</sub> เพื่อรีเทิร์นกลับ (Return)

(g) ออฟเซต 00C<sub>16</sub> ตรงกับคำสั่ง **CMP R0, 10** ยังเก็บอยู่ในหน่วยความจำภายในภาพ ซีพียูจึงตรวจสอบแคชบล็อกหมายเลข 00C<sub>16</sub> ซึ่งว่างเปล่า วงจรควบคุมแคชจึงนำคำสั่งนี้จากหน่วยความจำภายในภาพไปเก็บในแคชบล็อกที่ 00C<sub>16</sub> พร้อมกับส่งให้ซีพียูถอดรหัสและ PC ออฟเซตจึงเปลี่ยนเป็นหมายเลข 010<sub>16</sub>

(h) ออฟเซต 010<sub>16</sub> ตรงกับคำสั่ง **BLT Loop** ยังเก็บอยู่ในหน่วยความจำภายในภาพ ซีพียูจึงตรวจสอบแคชบล็อกหมายเลข 010<sub>16</sub> มีคำสั่งอื่นอยู่ เราเรียกเหตุการณ์นี้ว่า **คอนเทนเคนชันมิส (Contention Miss)** วงจรควบคุมแคชจึงนำคำสั่งนี้จากหน่วยความจำภายในภาพมาเขียนทับแคชบล็อกนี้แทนพร้อมกับนำไปถอดรหัส และ PC ออฟเซตจะเปลี่ยนเป็น 000<sub>16</sub> หรือไม่ขึ้นอยู่กับค่าของ R0 ว่าเงื่อนไข LT (Less Than) ซึ่งจะแปลเป็นเงื่อนไข R0 <

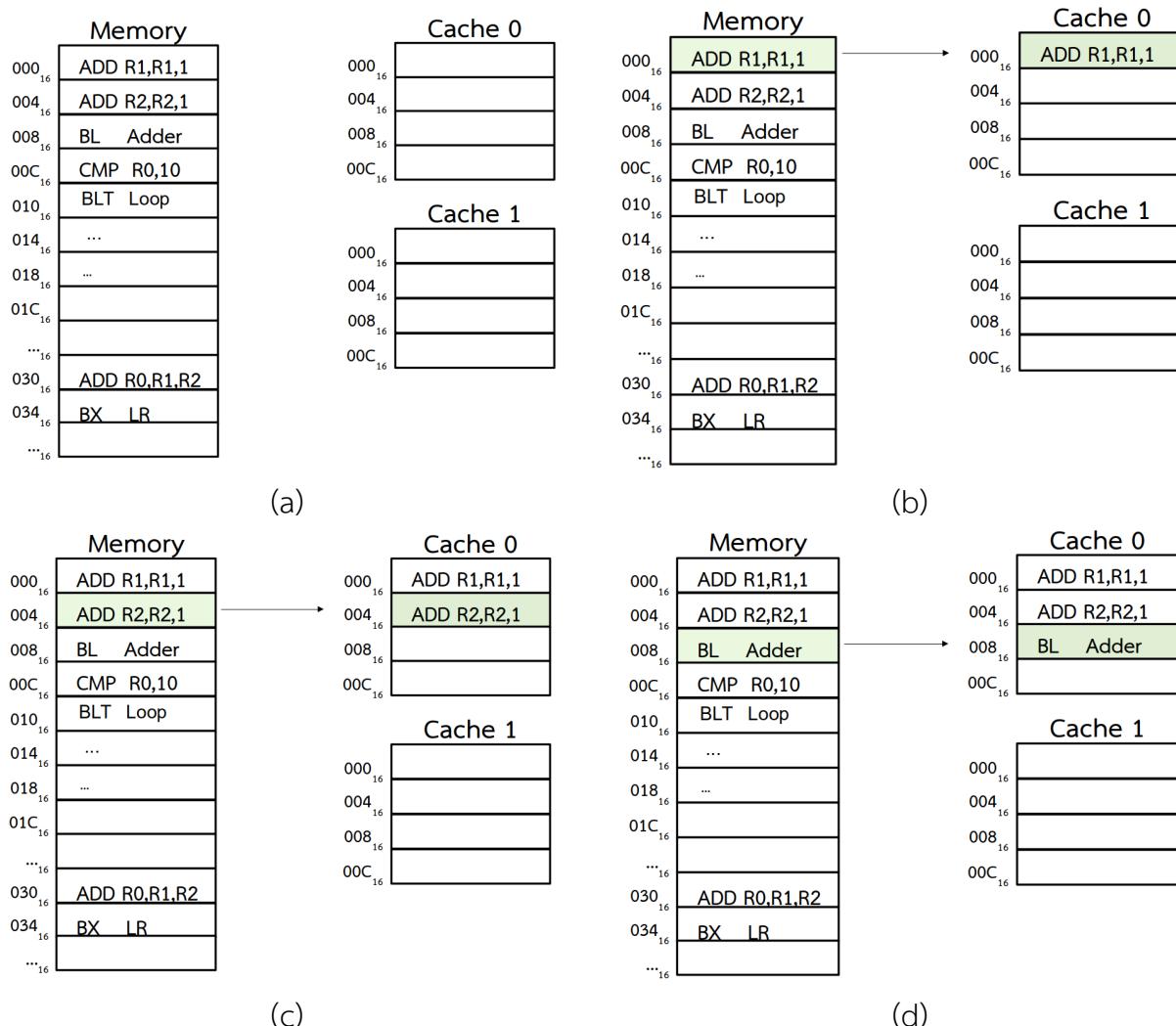


รูปที่ 5.9: การทำงานของแคชคำสั่ง (Instruction Cache) ชนิด Direct Map เมื่อซีพิยุพื้นที่คำสั่งที่ PC ของเซตเท่ากับ (e) 030<sub>16</sub>, (f) 034<sub>16</sub>, (g) 00C<sub>16</sub>, (h) 010<sub>16</sub>

10 เป็นจริง หรือ เป็นเท็จ

- หากเงื่อนไข  $R0 < 10$  เป็นจริง PC ของเซตจะเปลี่ยนเป็นหมายเลข 000<sub>16</sub> กระบวนการต่างๆ จะวนซ้ำคำสั่งที่อยู่ในแคชจะถูกอ่านซ้ำ เรียกว่า แคชซิท ทำให้เวลาในการเฟ้นคำสั่งสั้นลงเมื่อเทียบกับรอบแรก และหากมีการวนรอบมากกว่า 10 รอบ เช่น 1000 รอบ เวลาที่มีสินรอบแรกจะยิ่งคุ้มค่ามาก
- หากเป็นเท็จ PC ของเซตจะเปลี่ยนเป็นหมายเลข 014<sub>16</sub>

ผู้อ่านจะสังเกตเห็นว่า คำสั่งจากหน่วยความจำภายในภาพอย่างน้อย 2 ตำแหน่งถูกแมปให้ใช้แคชบล็อกตำแหน่งเดียวกัน เช่น แคชตำแหน่งที่ 000<sub>16</sub> จะเป็นเป้าหมายของหน่วยความจำตำแหน่งที่ 000<sub>16</sub> และ 020<sub>16</sub> เป็นต้น ยิ่งไปกว่านั้น แคชลำดับที่ 1 ที่ใช้งานในซีพิยุต่างๆ มีความจุอย่างกว้างขวางกว่าหน่วยความจำภายในภาพมาก เพราะหน่วยความจำภายในภาพของบอร์ด Pi3/Pi4 มีขนาดอย่างน้อย  $1 \times 2^{30}$  ไบต์ หรือ 1 กิกะไบต์ (GiB) ในขณะที่ขนาดแคชลำดับที่ 1 มีขนาดเล็กเพียง  $16 \times 2^{10}$  ไบต์ หรือ 16 KiB (kibibyte) คิดเป็น  $2^{16} : 1$  ทำให้เกิดการแข่งขัน (Contention) เป็นแบบ Many to One ดังนั้นมือแคชมีขนาดเล็ก อัตราการแข่งขัน (Contention Rate) จะยิ่งเพิ่มสูงมากขึ้น ในทางปฏิบัติ ซีพิยุต้องมีแคชหลายลำดับซึ่งเพิ่มขึ้นเพื่อช่วยลดอัตราการแข่งขันนี้



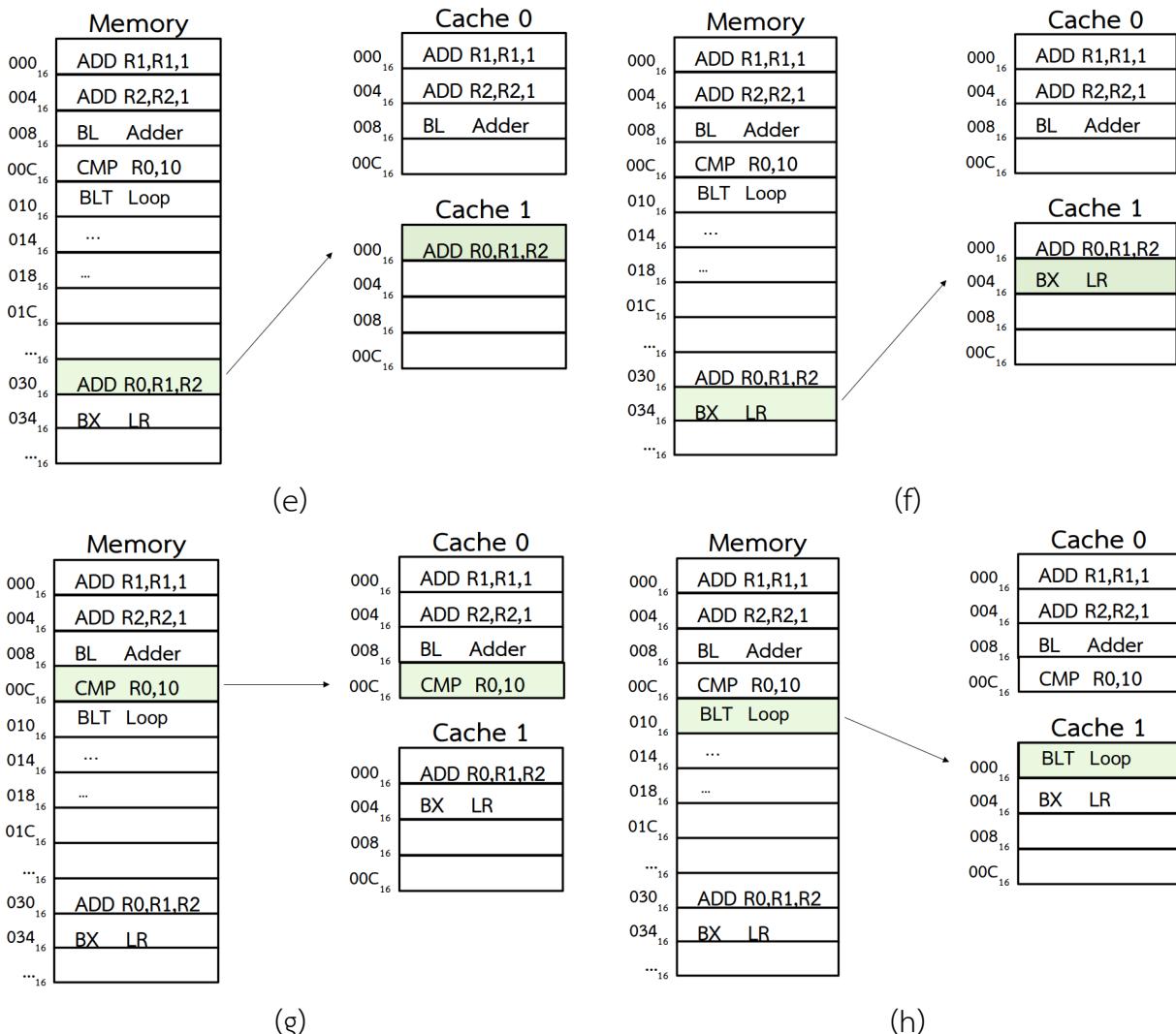
รูปที่ 5.10: การทำงานของแคชลำดับที่ 1 แคชคำสั่ง (Instruction Cache) ชนิด 2-Way Set Associative เมื่อ PC เฟลชคำสั่งที่หน่วยความจำภายภาพแอ็ดдрес (a) แคชคำสั่งยังไม่มีคำสั่งใดๆ เลย (b) PC ออฟเซตเท่ากับ  $000_{16}$ , (c)  $004_{16}$ , (d)  $008_{16}$

### 5.3.2 แคชชนิด N-way SA (Set Associative)

แคชคำสั่งชนิด N-Way SA (Set Associative) นิยมใช้ออกแบบแคชทุกลำดับ จำนวนเวย์ (Way) จะเพิ่มเป็นจำนวนสองยกกำลัง เช่น  $N = 8, 16, 32$  เว耶 เป็นต้น แคชจำลองชนิด 2-Way SA ในรูปที่ 5.10 และ 5.11 ประกอบด้วย Cache 0 และ Cache 1 หมายถึง แคชชนิด DM เว耶ที่ 0 และเว耶ที่ 1 ตามลำดับ แต่ละเว耶มีขนาด 4 บล็อก แต่ละบล็อกมีความจุ 4 ไบต์ สามารถบรรจุคำสั่งบล็อกละ 1 คำสั่ง ทำหน้าที่เก็บพักคำสั่งจากแอ็ดdressหมายเลข  $000_{16}, 004_{16}$  จนถึงหมายเลข  $00C_{16}$  โดยไล่จากบล็อกบนสุดลงมาวนซ้ำจากแอ็ดdressหมายเลข  $010_{16}, 014_{16}$  จนถึงหมายเลข  $01C_{16}$  และวนซ้ำอีกหลายรอบจนถึงแอ็ดdressหมายเลข  $FF0_{16}, FF4_{16}$  จนถึงหมายเลข  $FFC_{16}$

(a) เริ่มต้น แคชไม่มีข้อมูลใดๆ

(b) ออฟเซต  $000_{16}$  ตรงกับคำสั่ง `ADD R1, R1, 1` ยังเก็บอยู่ในหน่วยความจำภายภาพ ซีพียูจึงตรวจสอบแคชบล็อกหมายเลข  $000_{16}$  ของ Cache 0 และ Cache 1 ยังว่างเปล่า เราเรียกเหตุการณ์นี้ว่า **โคลด์มิส (Cold Miss)** วงจรควบคุมแคชจึงนำคำสั่งนี้จากหน่วยความจำภายภาพไปเก็บในแคช 0 บล็อกที่  $000_{16}$  พร้อมกับส่งให้ซีพียูอ่านทรัพยากรและ PC ออฟเซตจึงเปลี่ยนเป็นหมายเลข  $004_{16}$



รูปที่ 5.11: การทำงานของแคชลำดับที่ 1 แคชคำสั่ง (Instruction Cache) ชนิด 2-Way Set Associative เมื่อ PC เฟลชคำสั่งที่ PC ออฟเซตเท่ากับ (e) 030<sub>16</sub>, (f) 034<sub>16</sub>, (g) 00C<sub>16</sub>, (h) 010<sub>16</sub>

(c) ออฟเซต 004<sub>16</sub> ตรงกับคำสั่ง ADD R2, R2, 1 ยังเก็บอยู่ในหน่วยความจำภายใน CPU ซีพียูจึงตรวจสอบแคชบล็อกหมายเลข 004<sub>16</sub> ของ Cache 0 และ Cache 1 ยังว่างเปล่า วงจรควบคุมแคชจึงนำคำสั่งนี้จากหน่วยความจำภายใน CPU มาเก็บในแคช 0 บล็อกที่ 004<sub>16</sub> พร้อมกับส่งให้ซีพียูถอดรหัสและ PC ออฟเซตจึงเปลี่ยนเป็นหมายเลข 008<sub>16</sub>

(d) ออฟเซต 008<sub>16</sub> ตรงกับคำสั่ง BL Adder ยังเก็บอยู่ในหน่วยความจำภายใน CPU ซีพียูจึงตรวจสอบแคชบล็อกหมายเลข 008<sub>16</sub> ของ Cache 0 และ Cache 1 ยังว่างเปล่า วงจรควบคุมแคชจึงนำคำสั่งนี้จากหน่วยความจำภายใน CPU มาเก็บในแคช 0 บล็อกที่ 008<sub>16</sub> พร้อมกับส่งให้ซีพียูถอดรหัสและ PC ออฟเซตเปลี่ยนเป็นหมายเลข 030<sub>16</sub>

(e) ออฟเซต 030<sub>16</sub> ตรงกับคำสั่ง ADD R0, R1, R2 ยังเก็บอยู่ในหน่วยความจำภายใน CPU ซีพียูจึงตรวจสอบแคชบล็อกหมายเลข 000<sub>16</sub> ของ Cache 0 และ Cache 1 เพราะเป็นตัวแทนของหน่วยความจำภายใน CPU ที่ 030<sub>16</sub> พบว่าแคชบล็อกที่ 000<sub>16</sub> ของ Cache 1 ยังว่าง วงจรควบคุมแคชจึงนำคำสั่งนี้จากหน่วยความจำภายใน CPU มาเก็บในแคช 1 บล็อกที่ 000<sub>16</sub> พร้อมกับส่งให้ซีพียูถอดรหัสและ PC ออฟเซตเปลี่ยนเป็นหมายเลข 034<sub>16</sub>

(f) ออฟเซต 034<sub>16</sub> ตรงกับคำสั่ง BX LR ยังเก็บอยู่ในหน่วยความจำภายใน CPU ซีพียูจึงตรวจสอบแคชบล็อกหมายเลข 004<sub>16</sub> ของ Cache 0 และ Cache 1 พบร่วมกันของ Cache 1 ยังว่าง วงจรควบคุม

แคชจึงนำคำสั่งนี้จากหน่วยความจำภายในมาให้แก่ CPU ไปเก็บในแคช 1 บล็อกที่  $004_{16}$  พร้อมกับส่งให้ซีพียูอ่านหัสและ PC ออฟเซตเปลี่ยนเป็นหมายเลข  $00C_{16}$  เพื่อรีเทิร์นกลับ (Return)

(g) ออฟเซต  $00C_{16}$  ตรงกับคำสั่ง **CMP R0, 10** ยังเก็บอยู่ในหน่วยความจำภายในมาให้แก่ CPU ซึ่งตรวจสอบแคช บล็อกหมายเลข  $00C_{16}$  ของ Cache 0 และ Cache 1 พบร่วมกับแคชบล็อกที่  $00C_{16}$  ของ Cache 0 ยังว่าง วงจรควบคุมแคชจึงนำคำสั่งนี้จากหน่วยความจำภายในมาให้แก่ CPU ไปเก็บในแคช 0 บล็อกที่  $00C_{16}$  พร้อมกับส่งให้ซีพียูอ่านหัสและ PC ออฟเซตเปลี่ยนเป็นหมายเลข  $010_{16}$

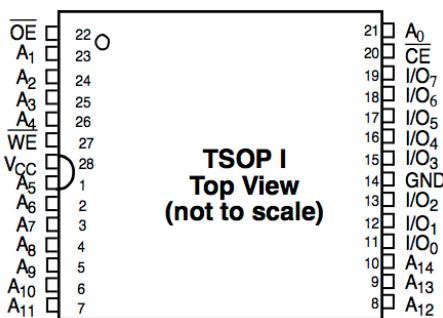
(h) ออฟเซต  $010_{16}$  ตรงกับคำสั่ง **BLT Loop** ยังเก็บอยู่ในหน่วยความจำภายในมาให้แก่ CPU ซึ่งตรวจสอบแคช บล็อกหมายเลข  $000_{16}$  ของ Cache 0 และ Cache 1 พบร่วมกับแคชบล็อกที่  $000_{16}$  ด้วยการเดาสุ่ม (Random) พร้อมกับส่งให้ซีพียูอ่านหัสและ PC ออฟเซตจะเปลี่ยนเป็นหมายเลข  $000_{16}$  หรือไม่ขึ้นอยู่กับค่าของ R0 ว่าจะเงื่อนไข LT (Less Than) ซึ่งจะแปลเป็นเงื่อนไข  $R0 < 10$  เป็นจริง หรือ เป็นเท็จ

- หากเงื่อนไข  $R0 < 10$  เป็นจริง PC ออฟเซตจะเปลี่ยนเป็นหมายเลข  $000_{16}$  กระบวนการต่างๆ จะวนซ้ำคำสั่งที่อยู่ในแคชจะถูกอ่านซ้ำ เรียกว่า แคชชิต ทำให้เวลาในการเฟทช์คำสั่งสั้นลงเมื่อเทียบกับรอบแรก และหากมีการวนรอบมากกว่า 10 รอบ เช่น 1000 รอบ เวลาที่มิสในรอบแรกจะยิ่งคุ้มค่ามาก
- หากเป็นเท็จ PC ออฟเซตจะเปลี่ยนเป็นหมายเลข  $014_{16}$

ผู้อ่านจะสังเกตเห็นว่า แคชคำสั่งชนิด SA มีการทำงานคล้ายกับแคชคำสั่งชนิด DM แต่มีจำนวน 2 เวiy นั่นคือ แคชหมายเลขบล็อกเดียวที่มี 2 ทางเลือก คือ เวiy 0 และเวiy 1 ขึ้นอยู่กับว่าเวiy ใดว่าง หากไม่ว่างทั้งคู่ แคชจะตัดสินใจให้เขียนทับ เรียกว่า **Cache Replacement Algorithm** ที่มา: [Tanenbaum and Austin \(2012\)](#) หากใช้อัลกอริธึมการตัดสินใจที่เรียกว่า **Least Recently Used** แคชจะเขียนทับตำแหน่งในเวiy ที่ไม่ได้ใช้งานล่าสุด จึงเห็นได้ว่า แคชชนิด SA นี้เป็นการขยายการทำงานของแคชชนิด DM ให้มีความยืดหยุ่นมากขึ้น ปัจจุบันซีพียูนิยมใช้แคชคำสั่ง/ข้อมูลชนิด SA ขนาด  $N = 8-16$  เวiy เพื่อประสิทธิภาพที่สิ้น

หลักการของแคชมีบทบาทต่อประสิทธิภาพของระบบโดยองค์รวม แคชชนิด N-way Set Associative ได้รับความนิยม เนื่องจากมีโครงสร้างเหมือนกับแคชชนิด DM และมีความสามารถคล้ายแคชชนิด Fully Associative มีการนำหลักการ Locality Principle ที่มา: [Tanenbaum and Austin \(2012\)](#) มาประยุกต์ใช้กับหน่วยความจำประเภทต่างๆ และหลากหลาย รวมถึงหลักการเวอร์ชวลเม莫รี เวอร์ชวลเม莫รีอาศัยการทำงานร่วมกันระหว่างฮาร์ดแวร์และระบบปฏิบัติการ เพื่อเพิ่มลำดับขั้นในการบริหารจัดการหน่วยความจำทุกลำดับขั้นดังได้กล่าวไปแล้ว

## 5.4 หน่วยความจำชนิด静态 RAM (Static RAM: SRAM)



รูปที่ 5.12: ตัวถังแบบ TSOP ของชิปหน่วยความจำชนิด静态 RAM Cypress 62256 ที่มา: [Cypress Semiconductor \(2002\)](#)

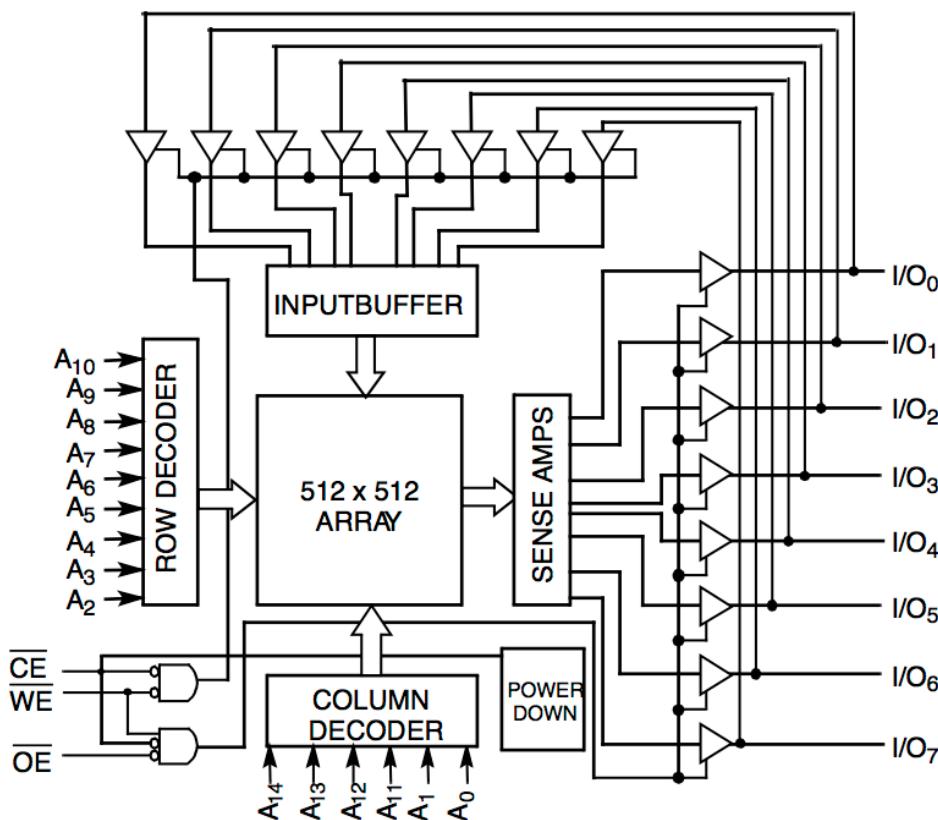
แคชลำดับที่ 1 และ 2 ในชิป BCM2837/BCM2711 และแคชลำดับที่ 3 ในชิปซีพียูอื่นๆ สร้างจากหน่วยความจำ static RAM หน่วยความจำ SRAM มีโครงสร้างที่ไม่ซับซ้อนและสามารถออกแบบให้ผลิตพร้อมกับวงจรทรานซิสเตอร์ในซีพียู นอกจ้านี้ หน่วยความจำ static RAM นี้ยังนิยมใช้งานเป็นหน่วยความจำหลักภายในชิปไมโครคอนโทรลเลอร์ ที่ต้องการสมรรถนะต่ำถึงปานกลาง โดยมีความจุหลายขนาด ตั้งแต่ 16 KiB (kilobyte) ถึงหลายเมบิไบต์ ผู้เขียนขอใช้หน่วยความจำ static RAM (Static RAM: SRAM) หมายเลข CY62256 เป็นกรณีศึกษา ชิป CY62256 ใช้เทคโนโลยีการผลิตซึ่ง CMOS ในปี ค.ศ. 2002 เอกสารคุณลักษณะ (Datasheet) ของชิปหน่วยความจำ static RAM นี้ ที่มา: [Cypress Semiconductor \(2002\)](#) สามารถค้นเจอที่ [ecee.colorado.edu](http://ecee.colorado.edu) ถึงแม้ว่าชิป CY62256 จะเป็นชิปที่เก่าแล้วแต่ชิปมีลักษณะเรียบง่ายและโดดเด่น ดังนี้

- ใช้กับแหล่งจ่ายไฟตั้งแต่ 4.5 - 5.5 โวลต์
- รองรับการทำงานความเร็วสูง เนื่องจากใช้เวลาเข้าถึงสั้นเท่ากับ 55 นาโนวินาที
- ชิปรีโ哥คกำลังไฟน้อยโดยมีค่าสูงสุดเพียง 275 มิลลิวัตต์ระหว่างปฏิบัติงาน
- ชิปรีโ哥คกำลังไฟน้อยโดยมีค่าสูงสุดเพียง 28 มิลลิวัตต์ระหว่างไม่ทำงาน (Stand by)

ชิปตัวนี้สามารถลดการใช้พลังงานได้สูงสุดถึง 99.99% เนื่องจากซีพียูสามารถเปิด/ปิดการใช้งานชิปโดยใช้ขาสัญญาณ  $\overline{CE}$  (Chip Enable bar) และ  $\overline{OE}$  (Output Enable bar) ซึ่งทำงานที่ล็อกจิคศูนย์ (Active Low) การอ่านและเขียนข้อมูลหน่วยความจำใช้บัสข้อมูลขนาด 8 บิตเดียวกัน  $I/O_0$  ถึง  $I/O_7$  เพื่อประหยัดจำนวนขา โดยภายในชิปมีเกตไดเรอร์สามสถานะ (Three State Driver Gate) บังคับทิศทางการไหลของสัญญาณ

ชิปอาจมีตัวถังแบบ TSOP (Thin Small Outline Package) ในทางกายภาพ เพื่อให้เหมาะสมสำหรับการบัดกรี เพื่อยืดตัวถังชิปบนแผ่นวงจรพิมพ์แบบ Surface Mount ผู้อ่านสามารถค้นคว้าเรื่องการห่อหุ้มชิปด้วยแพ็กเกจชนิด TSOP และอื่นๆ ได้ที่ [wikipedia](#) ซึ่งแต่ละชนิดมีข้อได้เปรียบและเสียเปรียบแตกต่างกันไป

### 5.4.1 โครงสร้างภายในของชิปหน่วยความจำสแตติกแรม



รูปที่ 5.13: โครงสร้างของหน่วยความจำขนาดนิดสแตติก Cypress 62256 ที่มา: [Cypress Semiconductor \(2002\)](#)

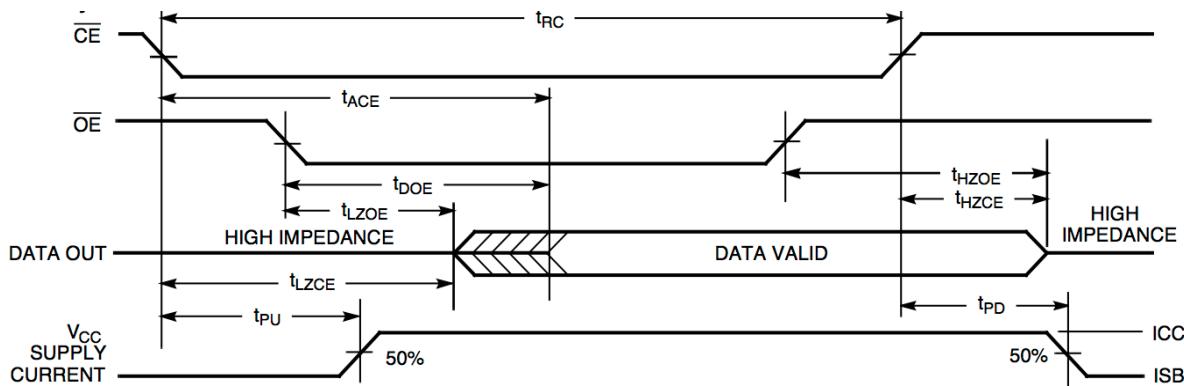
โครงสร้างของชิป CY62256 จัดเรียงเป็น  $2^5 \times 2^{10} \times 2^3 = 2^{9+9}$  หรือ  $512 \times 512$  บิต ซึ่งจะตรงกับการเรียกของเซลล์หน่วยความจำในรูปที่ 5.13

- ชิปยุ่งเริ่มต้นขบวนการอ่านข้อมูล โดยเปลี่ยนสัญญาณที่ขาดต่อไปนี้
  - ขาควบคุม  $\overline{WE}=1$ , ขา  $\overline{CE}=0$  และ  $\overline{OE}=0$
  - ขาแอดเดรส  $A_0 - A_{14}$  ทั้งหมด 15 ขา รับสัญญาณแอดเดรส เพื่อป้อนให้วงจรต่อครั้งหัสแนวนอน (Row Decoder) และแนวคอลัมน์ (Column Decoder) เพื่อสร้างสัญญาณจำนวน 512 เส้นในแนวนอนและ 64 เส้นในแนวตั้ง เพื่อเลือกบิตเซลล์จำนวน 8 บิตที่ต้องการพร้อมกันในอาร์เรย์
  - มอตูลขยายสัญญาณความไวสูง (Sense Amplifier) เป็นวงจรณาลีอกขยายสัญญาณ ใช้สำหรับขยายสัญญาณที่ส่งออกมาจากอาร์เรย์หน่วยความจำ เพราะสัญญาณที่ได้มีระดับเวลาแตกต่างมาก รายละเอียดเพิ่มเติมที่ [wikipedia](#)
  - สัญญาณที่ผ่านการขยายและตรวจจับแล้วจะกล้ายเป็นข้อมูลดิจิทัลผ่านเกตต์ไดรเวอร์สามสถานะให้ออกไปยังขา  $I/O_0$  จนถึง  $I/O_7$  พร้อมกันทั้ง 8 บิต
- ชิปยุ่งเริ่มต้นขบวนการเขียนข้อมูล โดยเปลี่ยนสัญญาณที่ขาดต่อไปนี้
  - ขาควบคุม  $\overline{WE}=0$ , ขา  $\overline{CE}=0$  และขา  $\overline{OE}=1$

- ซีพียูจะส่งข้อมูลจำนวน 8 บิตให้แหล่งเข้าทางขา  $I/O_0$  จนถึง  $I/O_7$  ผ่านเกตไดรเวอร์สามสถานะมาพักใน
- มอดูลบัฟเฟอร์ขาเข้า (Input Buffer) เพื่อพักข้อมูลชั่วคราวและรอเขียนในบิตเซลล์ที่ต้องการพร้อมกันทั้ง 8 บิต
- ขาเออดเดรส  $A_0 - A_{14}$  ทั้งหมด 15 ขา รับสัญญาณเออดเดรสให้วงจรต่อรหัสแนวอน (Row Decoder) และแนวคอลัมน์ (Column Decoder) ทำหน้าที่สร้างสัญญาณจำนวน 512 เส้นในแนวนอนและแนวตั้งจำนวน 64 เส้น เพื่อเลือกบิตเซลล์ที่ต้องการเขียนในอาร์เรย์พร้อมกันทั้ง 8 บิต

แต่ละบิตเซลล์ในอาร์เรย์ของหน่วยความจำสแตติกแรม ประกอบด้วยทรานซิสเตอร์จำนวนหนึ่งเชื่อมต่อกันคล้ายวงจรฟลิปฟล็อป (Flip-Flop) และเกตอินเวอร์เตอร์ (Invertor) จัดเรียงเป็นลูปป้อนกลับ (Feedback Loop) เพื่อเก็บรักษาข้อมูลโดยไม่ต้องใช้ขบวนการอื่นเพิ่มเติม และใช้เวลาเข้าถึงรวดเร็ว ทำให้แต่ละบิตเซลล์ใช้พื้นที่ซิลิโคน (Silicon Area) คิดเป็นจำนวนทรานซิสเตอร์มากกว่าเท่ากับทรานซิสเตอร์ 6 ตัวขึ้นไปต่อข้อมูลความจุ 1 บิต ผู้อ่านสามารถค้นค่าวารายละเอียดเพิ่มเติมได้ที่ [wikipedia](#)

#### 5.4.2 การทำงานของสแตติกแรม: ขบวนการอ่านข้อมูล



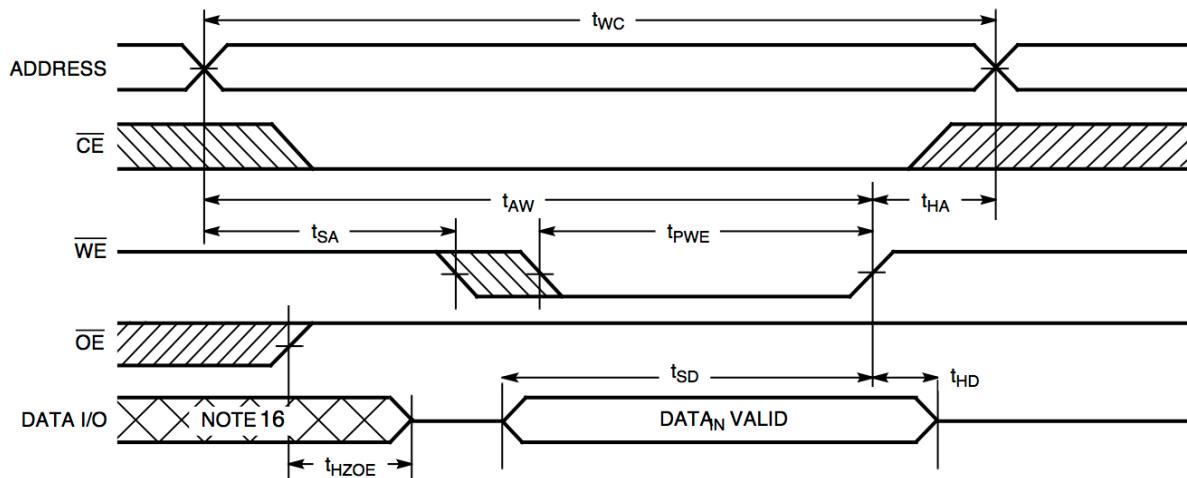
รูปที่ 5.14: ไดอะแกรมเวลา (Timing Diagram) สำหรับการอ่าน (Read) ข้อมูลจากหน่วยความจำชนิดสแตติกแรม ที่มา: [Cypress Semiconductor \(2002\)](#)

ผู้อ่านสามารถทำความเข้าใจขบวนการอ่านข้อมูลจากไดอะแกรมเวลาในรูปที่ 5.14 หมายเหตุ แก่นอนเป็นแกนเวลาเริ่มต้นจากขัยสุดของแกน การอ่านเริ่มต้นจากที่ซีพียูส่งเออดเดรส  $A_0 - A_{14}$  ไปยังหน่วยความจำ แล้วจึงเปลี่ยนขาสัญญาณ  $\overline{CE}$  และสัญญาณ  $\overline{OE}$  จาก "1" เป็น "0" เพื่อกระตุ้นการทำงานของหน่วยความจำ ซีพียูจะต้องรอเป็นระยะเวลาอย่างน้อย  $t_{ACE}$  ข้อมูล ณ เออดเดรสนั้นจะปรากฏบนบัสข้อมูล  $I/O_0 - I/O_7$  การอ่านค่าข้อมูลมีช่วงเวลาต่างๆ กำกับ ดังนี้

- $t_{ACE}$  เรียกว่า เวลาเข้าถึงข้อมูล หรือ Access Time โดยเริ่มนับจากเมื่อสัญญาณ  $\overline{CE}$  เปลี่ยนเป็น 0 จนข้อมูลที่ถูกต้องปรากฏ
- $t_{RC}$  เรียกว่า คาดเวลาที่สั้นที่สุดในการอ่านข้อมูลจากสแตติกแรม (Read Cycle Time) อย่างต่อเนื่อง โดย  $t_{RC} > t_{ACE}$
- $t_{OE}$  เรียกว่า เวลาที่ข้อมูลบนบัสข้อมูลยังถูกต้อง (Valid) เมื่อขาสัญญาณ  $\overline{OE}$  เปลี่ยนเป็น 1 แล้ว

- $t_{CE}$  เรียกว่า เวลาที่ข้อมูลบนบัสข้อมูลยังถูกต้อง (Valid) เมื่อขาสัญญาณ  $\overline{CE}$  เปลี่ยนเป็น 1 แล้ว
- $t_{PU}$  เรียกว่า เวลาที่กระแสไฟฟ้าเพิ่มขึ้นจาก 0% เป็น 50% โดยเริ่มนับจาก  $\overline{CE}$  เปลี่ยนจาก 1 เป็น 0 (Power Up Time)
- $t_{PD}$  เรียกว่า เวลาที่กระแสไฟฟ้าลดลงจาก 100% เป็น 50% โดยเริ่มนับจาก  $\overline{CE}$  เปลี่ยนจาก 0 เป็น 1 (Power Down Time)

### 5.4.3 การทำงานของสแตติกแรม: ขบวนการเขียนข้อมูล



รูปที่ 5.15: ไดอะแกรมเวลา (Timing Diagram) สำหรับการเขียน (Write) ข้อมูลในหน่วยความจำชนิดสแตติกแรม ที่มา: [Cypress Semiconductor \(2002\)](#)

ผู้อ่านสามารถทำความเข้าใจขบวนการเขียนข้อมูลของหน่วยความจำสแตติกแรมจากไดอะแกรมเวลาในรูปที่ 5.15 หมายเหตุ แก่นอนเป็นแก่นเวลาเริ่มต้นจากช้าสุดของแก่น การเขียนเริ่มต้นจากที่ซีพียูส่งออเดรส์  $A_0 - A_{14}$  ไปยังหน่วยความจำ แล้วจึงเปลี่ยนขาสัญญาณควบคุม  $\overline{CE}$  และ  $\overline{WE}$  จาก "1" เป็น "0" เพื่อกระตุ้นการทำงานของหน่วยความจำ ซีพียูจะต้องส่งข้อมูลบนบัสข้อมูล  $I/O_0 - I/O_7$  เป็นระยะเวลาอย่างน้อย  $t_{SD}$  เพื่อยืนยันว่าข้อมูลที่ถูกต้องจะเขียน ณ ออเดรสนั้น การเขียนค่าข้อมูลมีช่วงเวลาต่างๆ กำกับ ดังนี้

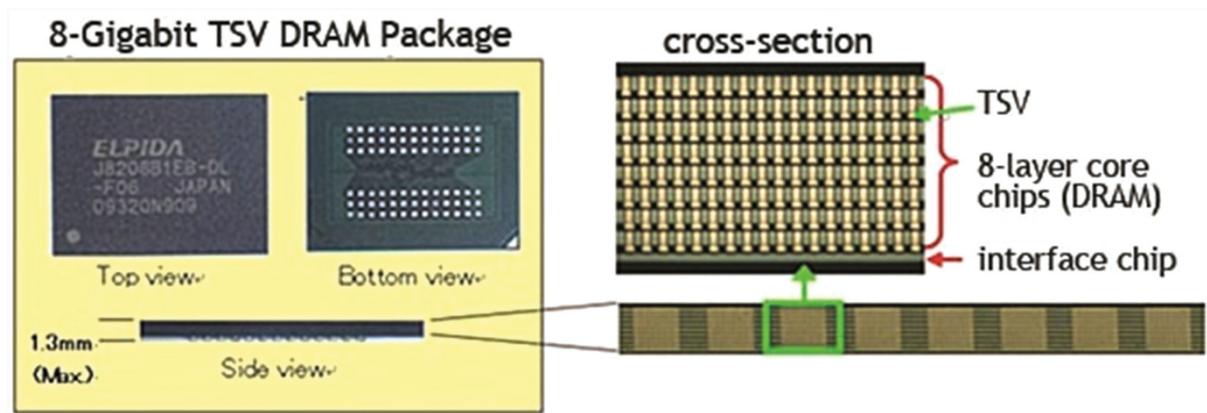
- $t_{SD}$  เรียกว่า เวลาที่จะตรึงข้อมูลให้เสถียร (Data Stable Time) ก่อนสัญญาณ Write Enable ( $\overline{WE}$ ) เปลี่ยนเป็น 1
- $t_{HD}$  เรียกว่า เวลาที่จะตรึงข้อมูลให้เสถียร (Data Hold Time) ต่อจาก  $t_{SD}$  เมื่อ Write Enable ( $\overline{WE}$ ) = 1 แล้ว
- $t_{WC}$  เรียกว่า คาบเวลาที่สั้นที่สุดในการเขียนข้อมูลในหน่วยความจำสแตติกแรม (Write Cycle Time) อย่างต่อเนื่อง
- $t_{AW}$  เรียกว่า เวลาเข้าถึงสำหรับการเขียน (Access Write Time)

## 5.5 หน่วยความจำหลักชนิด SDRAM

ผู้ผลิตเครื่องคอมพิวเตอร์โน้ตบุ๊ก โทรศัพท์เคลื่อนที่สมาร์ตโฟน และอุปกรณ์พกพาต่างนิยมออกแบบแบบติดตั้งชิปหน่วยความจำ SDRAM บน เมนบอร์ด (Main Board) เช่นเดียวกับบอร์ด Pi3/Pi4 เพื่อลดขนาดและปริมาตรของเครื่องให้มีขนาดเท่ากับบอร์ดเครดิต แต่ในเทคโนโลยีขั้นการผลิต SDRAM ยังไม่สามารถรวมกับขั้นการผลิตไมโครไพรเซสเซอร์ได้ ผู้ผลิตจึงจำเป็นต้องผลิตชิป SDRAM แยกต่างหาก

ชิป DDR2 SDRAM กรณีศึกษาที่ใช้บนบอร์ด Pi3/Pi4 ซึ่งได้อธิบายเบื้องต้นแล้วในหัวข้อที่ 3.1.2 ผลิตโดยบริษัท Elpida ประเทศญี่ปุ่นในปี ค.ศ. 2014 ที่มา: [Micron Technology \(2014\)](#) และต่อมาบริษัท ไมโครอนเทคโนโลยี ประเทศไทยได้เข้าถือหุ้นแทน ชิป DDR3 SDRAM นี้ใช้โครงสร้างแพ็คเกจและขาชนิด BGA (Ball Grid Array) จำนวน 168 ขาทางด้านล่างเพียงด้านเดียว ตัวชิปมีขนาด 12มม.x12มม.x0.8มม. ผู้ออกแบบบอร์ด Pi3/Pi4 ยึดติดชิป SDRAM อยู่ใต้บอร์ดตามรูปที่ 3.4 ผู้อ่านจะสังเกตได้ว่าแพ็คเกจหรือตัวถังชนิด BGA นี้จะใช้พื้นที่บนแผ่นวงจรพิมพ์เล็กเท่ากับตัวชิป ช่วยประหยัดพื้นที่บนบอร์ดและสามารถเชื่อมต่อกับชิป BCM2837/BCM2711 ใช้ระยะทางสั้นลง ทำให้ชิปทั้งสองสามารถทำงานด้วยสัญญาณล็อกความถี่สูงสุดตามคุณลักษณะเฉพาะ (Specification) เนื่องจากใช้เทคโนโลยีขั้นการผลิตที่ทันสมัยกว่า ในขณะที่ชิปหน่วยความจำสแตติกแรมใช้ตัวถังชนิด TSOP ในรูปที่ 5.12 ซึ่งเก่ากว่ามีข่ายนิ่นออกมายังทำให้ใช้พื้นที่จัดวางมากขึ้น

ชิป SDRAM มีความจุข้อมูลต่อชิปสูงมาก เนื่องจากใช้พื้นที่ชิปเท่ากับทรานซิสเตอร์เพียง 1-2 ตัวต่อข้อมูลความจุ 1 บิต แต่มีข้อเสีย คือ ใช้เทคโนโลยีและขั้นการผลิตที่ซับซ้อนกว่าหน่วยความจำสแตติกแรม ผู้ใช้ต้องการความเชื่อมั่นในตัวอุปกรณ์สูงและความถี่ล็อกสูง เช่นกัน ทำให้ต้นทุนต่อข้อมูลความจุหนึ่งบิตของ SDRAM สูงกว่าอุปกรณ์เก็บรักษาข้อมูล ในบทที่ 7

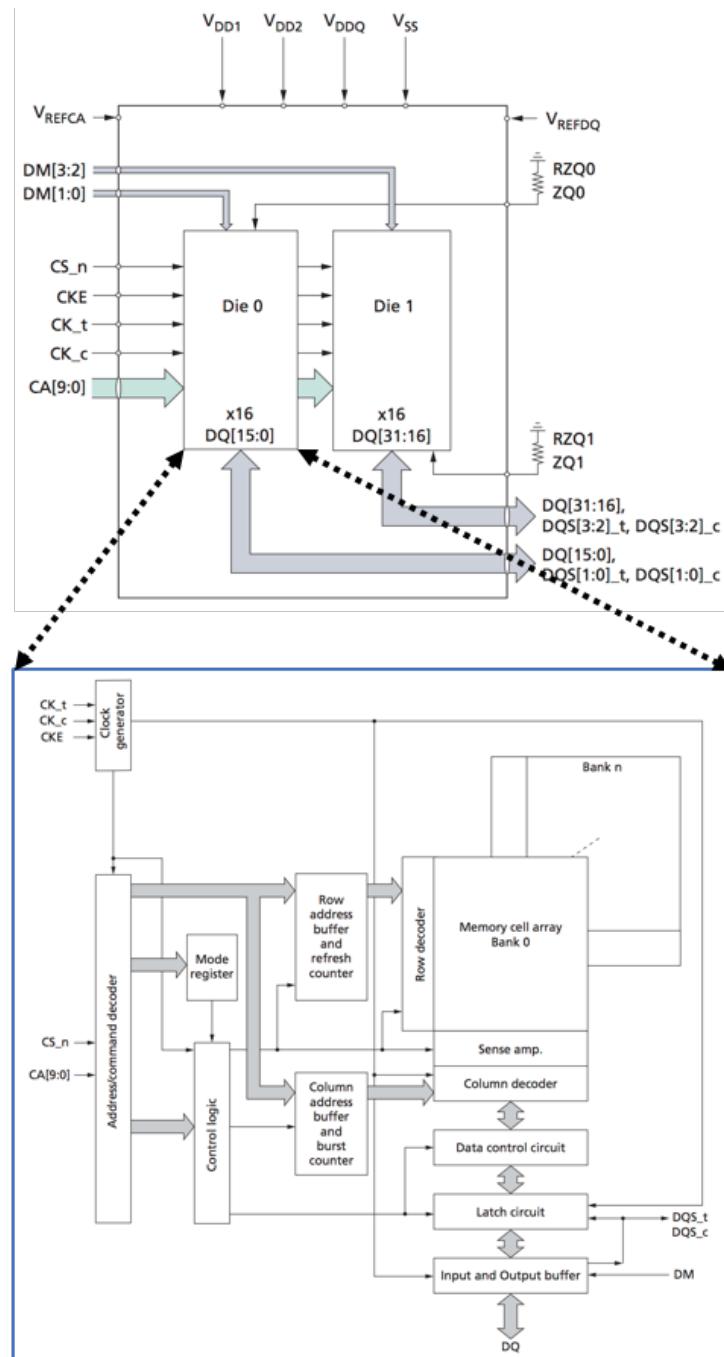


รูปที่ 5.16: รูปถ่ายด้านบน (Top View) ด้านล่าง (Bottom View) ด้านข้าง (Side View) และภาพตัดขวาง (Cross Section) ของชิป SDRAM โดยใช้เทคโนโลยี TSV (Through Silicon Via) ผู้ผลิตบริษัท Elpida ที่มา: [electroiq.com](#)

ผู้อ่านสามารถทำความเข้าใจโครงสร้างเชิงกายภาพที่ซับซ้อนของ SDRAM ชนิดนี้จากรูปที่ 5.16 ซึ่งเป็นชิป SDRAM ความจุขนาด 8 กิกะบิต ( $8 \times 2^{30}$  บิต) โดยบริษัท Elpida เช่นเดียวกับ ภาพถ่ายด้านบน (Top view) มีชื่อ บริษัทผู้ผลิต หมายเลขอุปกรณ์ ประเทศ และอื่นๆ ภาพถ่ายด้านล่างแสดงขาสำหรับเชื่อมต่อเข้าสู่ภายในชิปจำนวน 2 แถวๆ ละ 39 ขา และรอยบางครั้งมุมขวาวลาง เมื่อพลิกคว่ำลงจะตรงกับจุดวงกลมด้านข้างล่างของภาพถ่ายด้านข้างของชิป (Side view) มีความหนาเท่ากับ 1.3 มิลลิเมตร ผู้อ่านจะมองเห็นขาที่ยื่นออกมาใต้ตัวถังเมื่อขยายภาพตัดขวาง (cross-section) ให้ใหญ่ขึ้น แผ่นซิลิโคนแต่ละชั้นมีเชื่อมต่อสัญญาณในแนวราบภายในแผ่นหรือชั้นเดียวกัน ภาพตัดขวางที่ขยายทางด้านขวาไว้มีสูดจะมองเห็นเป็น 8 ชั้น (8-layer core) แสดงให้การเชื่อมต่อระหว่างแผ่นซิลิโคนเข้าด้วยกันในแนวตั้งโดยใช้เทคโนโลยี TSV (Through Silicon Via) ที่มา: [wikipedia](#) ซึ่ง

เชื่อมสัญญาณควบคุม สัญญาณคล็อก ไฟเลี้ยง กราวน์ด และอื่นๆ การเชื่อมต่อสัญญาณครบทั้งสามมิตินี้ช่วยให้ประยุกต์พิเศษที่เชื่อมต่อสัญญาณเหล่านี้บนแผ่นซิลิกอนในแนวราบ

### 5.5.1 โครงสร้างภายในของชิป SDRAM



รูปที่ 5.17: ปล็อกไออะแกรมภายในชิปหน่วยความจำ SDRAM ชนิด DDR2 ประกอบด้วยดาย (Die) 2 ชิ้น แต่ละชิ้นมีบัสข้อมูลขนาด 16 บิตเรียงขนานกันเป็น 32 บิต ที่มา: [Micron Technology \(2014\)](#)

ในรูปที่ 5.17 โครงสร้างภายในชิปหน่วยความจำ SDRAM ชนิด DDR2 ชิปกรณ์ศึกษา ประกอบด้วยดาย (Die) 2 ชิ้น แต่ละชิ้นมีบัสข้อมูลขนาด 16 บิตเรียงต่อกันเป็น 32 บิต ที่มา: [Micron Technology \(2014\)](#) แต่ละดายประกอบด้วย แบงชล์ดหน่วยความจำชนิด SDRAM จำนวน 8 ชิ้น หรือ 8 แบงค์ (Bank) ๆ ละ  $32 \times 2^{20}$  เซลล์

16 บิต คิดเป็น  $2 \text{ ดา�} \times 8 \text{ แบงค์} \times 32 \times 2^{20} \text{ เซลล์} \times 16 \text{ บิต}$  หรือ  $2^1 \times 2^3 \times 2^5 \times 2^{20} \times 2^4 = 2^{33} = 2^3 \times 2^{30}$   
 $= 8 \text{ กิกะไบต์} \text{ หรือ } 1 \text{ กิกะไบต์ (GiB)}$

แบงค์ที่ 0 ถึง 7 แต่ละแบงค์จะประกอบด้วยวงจรตอตรัหัสแอดเดรส (Address Decoder) ในแนวนอน (Row Decoder) และแนวตั้ง (Column Decoder) ภายในชิปประกอบด้วยขาสัญญาณต่างๆ เรียงตามลำดับความสำคัญ ดังนี้

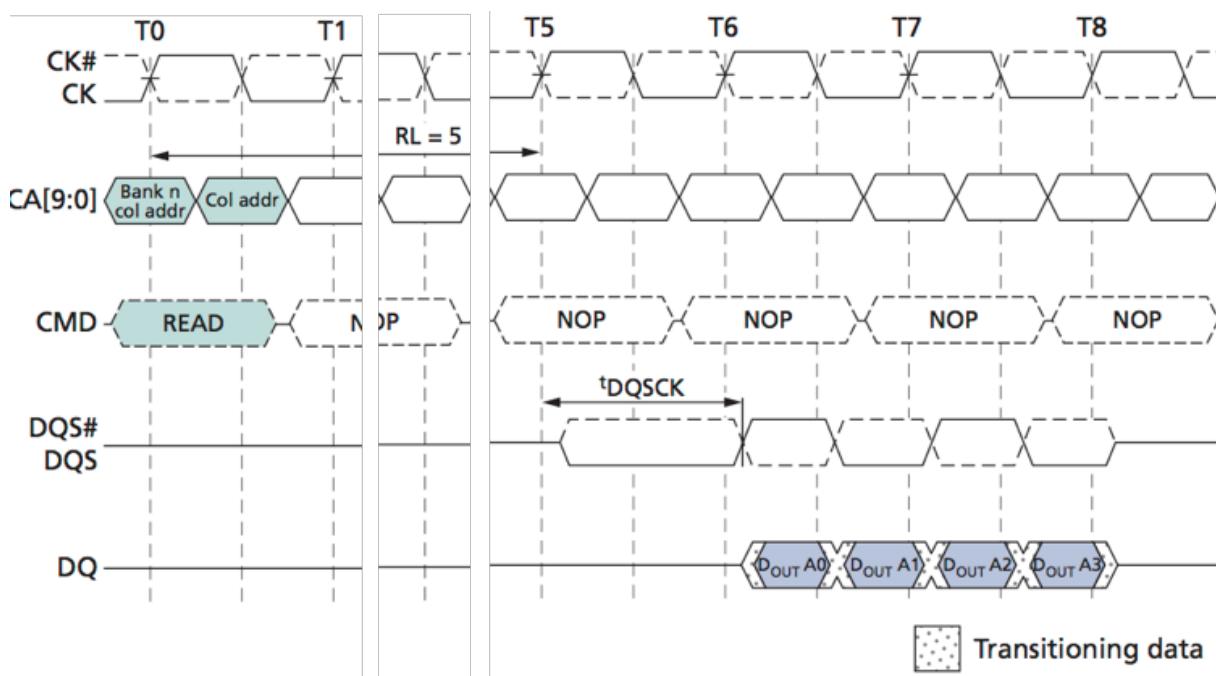
- **CS<sub>n</sub>** หรือ (Chip Select Not) หรือ  $\overline{CS}$  หรือ (Chip Select bar) ใช้เปิด/ปิดการทำงานของชิป เพื่อช่วยประหยัดพลังงาน
- **CKE (Clock Enable)** เมื่อสัญญาณ  $\overline{CS}=0$  เพื่อให้ชิปทำงาน หลังจากนั้น สัญญาณ CKE ใช้สำหรับเปิด/ปิดการทำงานของคล็อกที่จ่ายให้กับชิป SDRAM นี้ ซึ่งสามารถควบคุมสัญญาณ CKE=0 เพื่อพักการใช้งาน SDRAM ชั่วคราวเพื่อช่วยประหยัดพลังงาน
- **CK (Clock)** และ **CK# (Clock Not)** หรือ Clock bar คือ สัญญาณคล็อกสองสัญญาณที่มีเฟส (Phase) หรือขั้วตรงข้ามกัน เรียกว่า คูดิฟเฟอเรนเชียล (Differential Pair) หน่วยเป็นเมกะเอิร์ตซ์ สัญญาณคล็อกความถี่สูงสุด 400 เมกะเอิร์ตซ์
- ชิป DDR4 ล่าสุดสำหรับคอมพิวเตอร์ตั้งโต๊ะความถี่สูงสุด มากกว่า 3,000 เมกะเอิร์ตซ์ และมีแนวโน้มเพิ่มขึ้นตามเทคโนโลยีการผลิตที่พัฒนาอย่างต่อเนื่อง ผู้อ่านสามารถค้นคว้าเพิ่มเติมได้ที่ [wikipedia](#)
- **Command/Address CA[0:9]** ขนาด 10 บิต ใช้มัลติเพล็กซ์ (Multiplex) สัญญาณคำสั่ง (Command) และแอดเดรส (Address) เพื่อรับคำสั่ง (Command) และสัญญาณแอดเดรส (Address) ต่างหัวใจเวลากัน
  - ซึ่งจะส่งคำสั่ง (Command) ต่างๆ ดังนี้ Activate, Burst Read, Burst Write, Refresh, Power Down, Precharge และ Burst Terminate เป็นต้น เพื่อกำหนดโหมดการทำงานของหน่วยความจำ SDRAM ได้แก่ Power Up, Deep Power Down, Active, Idle, Reading, Writing, Precharging, Refreshing เป็นต้น
  - **สัญญาณแอดเดรสแถว (Row Address)** จำนวน 14 บิต และ **แอดเดรสคอลัมน์ (Column Address)** จำนวน 11 บิต จะพักเก็บในวงจรบัฟเฟอร์ เพื่อป้อนให้กับวงจรตอตรัหัส (Decoder) คล้ายกับการทำงานของหน่วยความจำสแตติกแรมในหัวข้อที่ [5.13 การรับสัญญาณแอดเดรสเกิดขึ้น](#) ขอบขั้นและขอบขั้นของแต่ละสัญญาณคล็อก
- **ดาต้าสตอเรบ DQS[0:3]** ขนาด 4 บิต ใช้สำหรับควบคุมการอ่านและการเขียนข้อมูล
- **ดาต้าบัส DQ[0:31]** จำนวน 32 บิต หรือ 4 ไบต์ สำหรับอ่านข้อมูลจากชิป SDRAM และเขียนข้อมูลในชิป ชิป DDR2 นี้ สามารถกำหนดขนาดข้อมูลจากการอ่านเขียนต่อเนื่อง (Burst length) เป็น 4, 8 และ 16 ตำแหน่งต่อเนื่องกัน
- **ดาต้ามาสก์ DM (Data Mask) [0:3]** ขนาด 4 บิต ใช้สำหรับมาสก์ (Mask) หรือปิด เพื่อควบคุมการเขียนข้อมูลแต่ละไบต์ โดย DM[0] ควบคุมไบต์ที่ 0, DM[1] ควบคุมไบต์ที่ 1 ตามลำดับ ทำให้การเขียนข้อมูลแต่ละไบต์เป็นอิสระจากกันได้

ผู้อ่านจะสังเกตเห็นว่า วงจรรอบๆ เซลล์หน่วยความจำ (Cell Array) มีลักษณะที่คล้ายกับอาร์เรย์ของเซลล์หน่วยความจำของสแตติกแรมในรูปที่ [5.13](#) ดังนี้

- วงจรอตอร์หัสแนวราบ (Row Decoder) สร้างสัญญาณจำนวน  $2^{14} = 16,384$  เส้น เรียกว่า สายบิตไลน์ (Bit line) ในแนวราบ
- วงจรอตอร์หัสแนวตั้ง (Column Decoder) สร้างสัญญาณจำนวน  $2^{11} = 2048$  เส้น เรียกว่า สายเวิร์ดไลน์ (Word line) ในแนวตั้ง
- วงจรขยายสัญญาณความไวสูง (Sense Amplifier) ใช้สำหรับขยายสัญญาณในขบวนการอ่าน เช่นเดียว กับการอ่านของหน่วยความจำสแต็ติกแรม

จุดตัดระหว่างสายบิตไลน์และสายเวิร์ดไลน์ คือ ตำแหน่งของบิตเซลล์ที่ต้องการจะอ่านหรือเขียน คล้ายกับการทำงานของหน่วยความจำสแต็ติกแรมแต่หน่วยความจำ SDRAM มีโครงสร้างของบิตเซลล์ที่แตกต่าง กล่าวคือ บิตเซลล์แต่ละบิตของ SDRAM ประกอบด้วยทรานซิสเตอร์ชนิด MOS จำนวน 2 ตัว ทรานซิสเตอร์ตัวที่หนึ่งทำหน้าที่เก็บประจุเมื่อมีอนค่าปาราซิเตอร์ เมื่อค่าปาราซิเตอร์ มีประจุ แทนข้อมูล 1 หรือไม่มีประจุ แทนข้อมูล 0 ทรานซิสเตอร์ตัวที่สองทำหน้าที่ควบคุมการอ่านและเขียนข้อมูล ผู้อ่านสามารถคลิก ไดนามิคแรมขนาด  $4 \times 4$  บิต เพื่อศึกษาโครงสร้างบิตเซลล์ของ SDRAM เพิ่มเติม

### 5.5.2 การทำงานของ SDRAM: อ่านและเขียน

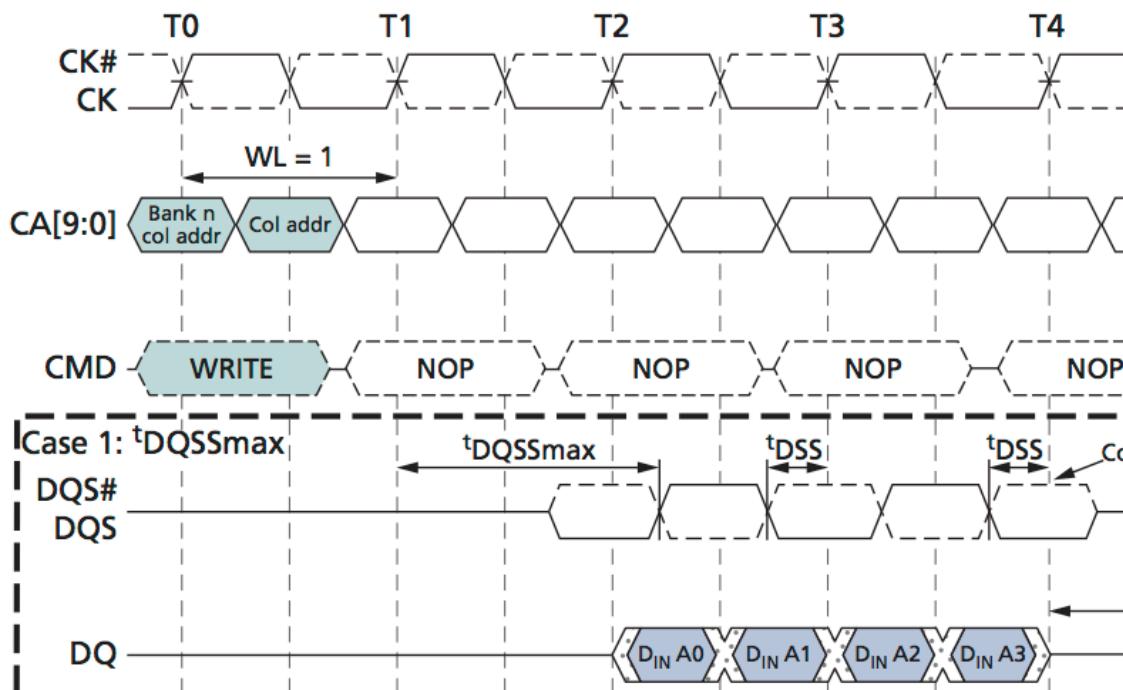


รูปที่ 5.18: ไดอะแกรมเวลา (Timing Diagram) สำหรับการอ่านต่อเนื่องของหน่วยความจำ SDRAM รุ่น Elpida B8132B4PB-8D-F คำสั่ง Burst READ โดย RL (Read Latency) เท่ากับ 5 ใช้กับ BL (Burst Length) เท่ากับ 4 ตำแหน่ง หมายเหตุ รูปมีการตัดต่อเพื่อเน้นบริเวณที่สำคัญ, NOP=No Operation ที่มา: [Micron Technology \(2014\)](#)

การทำงานของหน่วยความจำ SDRAM ผลิตโดยบริษัท Elpida รุ่น B8132B4PB-8D-F มีความหลากหลาย แต่ผู้อ่านสามารถทำความเข้าใจคำสั่งที่ใช้งานบ่อยที่สุด คือ คำสั่ง Burst READ และ คำสั่ง Burst Write ดังนี้

คำสั่ง **Burst READ** เริ่มต้นโดย ทำให้สัญญาณ  $CS\#$  หรือ  $\overline{CS}=0$ ,  $CA0=1$ ,  $CA1=0$  และ  $CA2=1$  ณ ขอบขาขึ้นของสัญญาณคล็อก รูปที่ 5.18 แสดงถึงไดอะแกรมเวลาสำหรับการอ่านข้อมูลของหน่วยความจำชนิด DDR2 แบบซิงโครนัสซึ่งต้องทำงานด้วยขอบสัญญาณทั้งขาขึ้นและขาลงของสัญญาณคล็อก ในรูปนี้เป็นการทำงานตาม

คำสั่ง Burst READ ด้วยค่า RL (Read Latency) เท่ากับ 5 ไซเกล โดยนับจาก T1 ถึง T5 เป็นการนับจำนวนคลีกหรือจำนวนไซเกล หรือเวลาของสัญญาณคลีก การนับเริ่มจากขอบขาขึ้นของคำสั่ง READ หมายเหตุ รูปมีการตัดต่อเพื่อเน้นบริเวณที่สำคัญ และอ่านข้อมูลต่อเนื่องเป็นจำนวน BL (Burst Length) = 4 ตำแหน่ง เรียงติดกันโดยเริ่มจากแอดเดรสที่น้อยกว่า (A0) วงจรจะใช้การเปลี่ยนแปลงของสัญญาณ DQS (Data Strobe) เพื่อซิงโครไนซ์กับการอ่านข้อมูล ที่มา: [micron.com](http://micron.com)



รูปที่ 5.19: ไดอะแกรมเวลา (Timing Diagram) สำหรับการเขียนต่อเนื่องของหน่วยความจำ Elpida รุ่น B8132B4PB-8D-F ตามคำสั่ง Burst WRITE โดย WL (Write Latency) = 1 ไซเกล, BL (Burst Length) = 4 ตำแหน่ง หมายเหตุ: รูปมีการตัดต่อเพื่อเน้นบริเวณที่สำคัญ ที่มา: [Micron Technology \(2014\)](http://Micron Technology (2014))

ตัวอย่างสำหรับการเขียนต่อเนื่องของหน่วยความจำ Elpida รุ่น B8132B4PB-8D-F ด้วยคำสั่ง Burst WRITE ในรูปที่ 5.19 โดย WL (Write Latency) เท่ากับ 1 ไซเกลและ BL (Burst Length) = 4 ตำแหน่ง โดยแอดเดรสที่เขียนจะเรียงติดกันโดยเริ่มจากแอดเดรสที่น้อยกว่า (A0) วงจรจะใช้การเปลี่ยนแปลงของสัญญาณ DQS (Data Strobe) เพื่อซิงโครไนซ์กับการเขียนข้อมูล ผู้อ่านจะสังเกตเห็นว่า ขอบสัญญาณ DQS จะเปลี่ยนแปลงตรงกลางข้อมูลสำหรับการเขียน แต่ในรูปที่ 5.18 ขอบสัญญาณ DQS จะเปลี่ยนแปลงตามข้อมูลสำหรับการอ่าน

### 5.5.3 การรีเฟรช (Refresh) ข้อมูล

การรีเฟรช (Refresh) คือ การอ่านข้อมูลที่อยู่ในบิตเซลล์ต่างๆ แล้วเขียนซ้ำที่บิตเซลล์เดิม เพื่อป้องกันไม่ให้ประจุที่เก็บอยู่ในบิตเซลล์ต่างๆ รั่วไหลหายไป เนื่องจากเซลล์ต่างๆ ที่ใช้เก็บข้อมูล ทำหน้าที่มี ('1') หรือไม่มี ('0') ประจุไฟฟ้า สำหรับบิตเซลล์ที่มีประจุฯ เหล่านี้อาจรั่วไหลหายไปเมื่อเวลาผ่านไปไม่กี่มิลลิวินาที เมื่อจำนวนประจุลดลง ทำให้การแยกแยะระหว่างบิตเซลล์ที่มีและไม่มีประจุยากขึ้น และอาจทำให้ตัวความไม่ถูกต้องและเกิดข้อผิดพลาดในการอ่านข้อมูล

วงจรควบคุมจะส่งคำสั่งรีเฟรชทุกๆ 32 มิลลิวินาที หากมีการรีเฟรชดำเนินการอยู่ การอ่านหรือเขียนหน่วยความจำจะต้องหยุดรอ เพื่อให้ขบวนการรีเฟรชนั้นเสร็จสิ้น ในทำนองเดียวกัน หากมีการอ่านหรือเขียนข้อมูลจริงอยู่ การรีเฟรชจะต้องหยุดรอ ก่อน เพื่อให้ขบวนการอ่านหรือเขียนนั้นเสร็จสิ้น

หน่วยความจำสแตติกแรมไม่ต้องมีการรีเฟรชข้อมูล เนื่องจากการจัดเก็บข้อมูลใช้วิธีการเก็บข้อมูลที่แตกต่างกับหน่วยความจำ SDRAM ทำให้หน่วยความจำสแตติกแรมมีสมรรถนะและประสิทธิภาพสูงกว่า แต่ต้องใช้จำนวนทรานซิสเตอร์ต่อบิตเซลล์มากกว่า จึงทำให้ใช้พื้นที่บนแผ่นซิลิกอนต่อข้อมูลความจำ 1 บิตใหญ่กว่า เช่นกัน ผู้อ่านสามารถค้นคว้าเพิ่มเติมได้ที่ [wikipedia](#)

## 5.6 สรุปท้ายบท

หน่วยความจำลำดับชั้นอนาคัญเทคโนโลยีหน่วยความจำหลายชนิด หลายขนาดข้อมูลความจำเข้าด้วยกัน ยกตัวอย่างเช่น

- เทคโนโลยี SRAM เป็นหน่วยความจำที่ใช้เวลาเข้าถึงข้อมูลรวดเร็ว แต่ใช้พื้นที่บนซิปซีพียูต่อข้อมูลความจำ 1 บิตสูงมาก จึงใช้งานเป็นรีจิสเตอร์ และ แคชลำดับต่างๆ
- เทคโนโลยี SDRAM เป็นหน่วยความจำข้อมูลความจำสูงกว่าแต่ต้องการเวลาเข้าถึงนานกว่าหน่วยความจำสแตติกแรมนำมาใช้งานเป็นหน่วยความจำหลัก
- เทคโนโลยีหน่วยความจำแฟลชเป็นหน่วยความจำที่มีข้อมูลความจำสูงกว่าแต่เวลาเข้าถึงข้อมูลนานกว่า SDRAM นำมาใช้งานเป็นอุปกรณ์เก็บรักษาข้อมูล รายละเอียดเพิ่มเติมในบทที่ 7

เพื่อให้คอมพิวเตอร์มีข้อมูลความจำเพียงพอและตอบสนองต่อความต้องการใช้งานระบบโดยเฉลี่ยได้รวดเร็วขึ้น โดยการผสานจุดเด่นของหน่วยความจำแต่ละชนิดเข้าด้วยกัน และช่วยประหยัดต้นทุนของระบบ

เนื้อหาในบทนี้ว่าด้วยเรื่องของหน่วยความจำผ่านภารกิจความรู้ในบทก่อนหน้า ผู้เขียนจึงขออุปมาอุปมาภัยการทำงานของคอมพิวเตอร์โดยรวม ดังนี้

- **ข้อมูล** คือ วัตถุดิบสำหรับประกอบอาหาร อาจมาจากแหล่งข้อมูลหลายๆ แหล่ง เช่น ผู้ใช้กรอกข้อมูลในโปรแกรม ไฟล์ข้อมูลในหน่วยสำรอง เป็นต้น
- **ซอฟต์แวร์** คือ สูตรหรือขั้นตอนการประกอบอาหารอย่างละเอียด เพื่อให้พ่อครัวประกอบอาหารให้สูก และมีรสชาติดี
- **ซีพียู** คือ พ่อครัวและเครื่องครัวที่ประกอบอาหารตามสูตรเท่านั้น ประกอบด้วย
  - ALU คือ อุปกรณ์เครื่องครัว เช่น มีด เย็บ กระทะ หม้อ เป็นต้น
  - รีจิสเตอร์ คือ งานหรือที่พักอาหารที่ปูร่องไว้ เพื่อรอทำให้เป็นอาหารที่ปูร่องสำเร็จต่อไป
  - วงจรอินๆ เช่น แคชลำดับต่างๆ มีหน้าที่เสริมการทำงานของซีพียูให้มีประสิทธิภาพดียิ่งขึ้น
- **หน่วยความจำภายในภาพ** คือ ชั้นวางวัตถุดิบ (ข้อมูล) ขนาดเล็กที่ไม่สามารถเก็บวัตถุดิบได้นาน
- **Information** หรือสารสนเทศ คือ อาหารที่ประกอบและปูร่องสำเร็จแล้ว โดยอาศัยซีพียูและซอฟต์แวร์ ที่กล่าวมาข้างต้น
- **อุปกรณ์เก็บรักษาข้อมูล** คือ ตู้แช่เย็นขนาดใหญ่ที่สามารถบรรจุวัตถุดิบได้ปริมาณมาก และระยะเวลา นาน หากต้องการเก็บข้อมูลหรือวัตถุดิบไว้ทานต่อไป ข้อมูลดิบหรือวัตถุดิบมักมีปริมาณมากๆ วัตถุดิบเหล่านี้จะเก็บในรูปของไฟล์ข้อมูล ซอฟต์แวร์สามารถอ่านหรือเขียนข้อมูลจากไฟล์ที่มีลักษณะต้องการ

- เครือข่ายอินเทอร์เน็ตช่วยถ่ายโอนข้อมูลจากเซิร์ฟเวอร์ต่างๆ บนในรูปของไฟล์ข้อมูล หรือ ในรูปของข้อมูลที่กระจัดกระจายมาทางเซ็นเซอร์ต่างๆ ด้วยเทคโนโลยี IoT (Internet of Thing) คือ เครือข่ายการขนส่งวัตถุดิบจากต่างประเทศมาปรุงอาหาร โดยจะต้องพักรเก็บในตู้แช่เย็นระหว่างทาง
- การคำนวณแบบขนาน (Parallel Computing) ต้องการซีพียูและ/หรือจีพียูหลายๆ แกนประมวลผล (พ่อครัวและเครื่องครัวจำนวนมาก) ช่วยกันประมวลผลข้อมูลปริมาณมากๆ ให้เสร็จสิ้นเร็วขึ้น ซึ่งบทที่ 8 จะอธิบายเรื่องนี้พอสังเขป

## 5.7 คำตามท้ายบท

- หน่วยความจำในบทนี้ทั้งหมด สามารถเก็บข้อมูลได้หรือไม่หากไม่มีไฟเลี้ยง
- เหตุใดเครื่องคอมพิวเตอร์จึงต้องมีอุปกรณ์เก็บรักษาข้อมูล คู่กับ หน่วยความจำภายในภาพเช่น SDRAM และ SRAM ในบทนี้
- จงเปรียบเทียบการอ่านของหน่วยความจำสแตติกแรมและ DDR-SDRAM ในแง่�ุมเหล่านี้
  - ขนาดของข้อมูลขั้นต่ำที่สามารถอ่านได้ หน่วยเป็นไบต์
  - ระยะเวลาเข้าถึง (Access Time) เพื่อรอให้ข้อมูลปรากฏบนบัสข้อมูล
- จงเปรียบเทียบการเขียนของหน่วยความจำสแตติกแรม และ DDR-SDRAM ในแง่՞ມุมเหล่านี้
  - ระยะเวลาเข้าถึง (Access Time) เพื่อให้รอให้ข้อมูลเขียนสำเร็จ
  - ขนาดของข้อมูลขั้นต่ำและขั้นสูงที่สามารถเขียนได้
- เหตุใดหน่วยความจำสแตติกแรมจึงไม่ต้องรีเฟรชข้อมูล
- เหตุใดหน่วยความจำ SDRAM จึงต้องรีเฟรชข้อมูลภายใต้เงื่อนไขในระยะๆ

## บทที่ 6

# กลไกอินพุตและเอาต์พุต (Input and Output)

ผู้ใช้สามารถอาศัยเครื่องคอมพิวเตอร์ติดต่อกับโลกภายนอกผ่านทางอุปกรณ์อินพุต/เอาต์พุต เช่น การเชื่อมต่อกับผู้ใช้ผ่านทางคีย์บอร์ด เม้าส์ หน้าจอสัมผัส การขยับ (Motion) / การเอียง (Tilt) / ความเร่ง (Acceleration) ของการเคลื่อนไหว การเชื่อมต่อกับเครื่องข่ายอินเทอร์เน็ต ผ่านทางเครือข่ายมีสายและเครือข่ายไร้สาย เป็นต้น ดังนั้น การเชื่อมต่อกับอุปกรณ์เหล่านี้จำเป็นต้องทำตามมาตรฐาน เพื่อลดความยุ่งยากในการออกแบบและต้นทุนโดยรวมของคอมพิวเตอร์

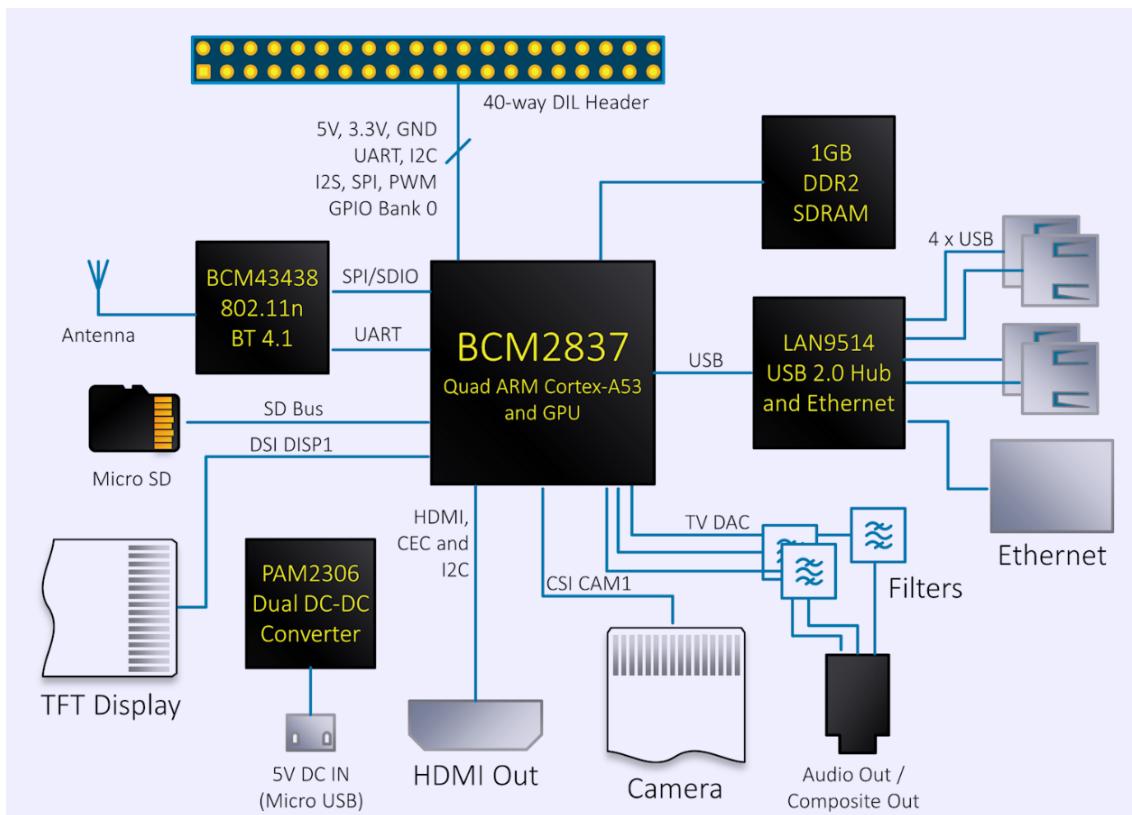
ระบบปฏิบัติการ จะทำหน้าที่ควบคุม และบริหาร จัดการ อุปกรณ์อินพุต/เอาต์พุต เพื่อให้โปรแกรม ต่างๆ สามารถใช้ทรัพยากรเหล่านี้ร่วมกัน โดยจะต้องมีการป้องกันและการจัดลำดับการใช้งาน (Protection and Scheduling) เนื่องจากการใช้งานอุปกรณ์อินพุต/เอาต์พุต ทำให้เกิดการอินเทอร์รัปท์ (Interrupt) แปลว่า ขัดจังหวะการทำงานของซีพียู ระบบปฏิบัติการจะต้องจัดเตรียมวิธีการเชื่อมต่อกับอุปกรณ์อินพุต/เอาต์พุต โดยมีฮาร์ดแวร์ที่จะควบคุมการทำงานของอุปกรณ์ต่างๆ แทนโดยตรง เพื่อให้ซีพียูรับเฉพาะงานที่สำคัญโดยเน้นที่ การประมวลผลข้อมูลได้อย่างต่อเนื่อง ซึ่งวิธีนี้จะเพิ่มประสิทธิภาพและการตอบสนองต่อผู้ใช้โดยรวมได้ และเพื่อให้โปรแกรมเมอร์สามารถพัฒนาโปรแกรมได้อย่างรวดเร็ว ประสิทธิภาพสูง และปลอดภัย เนื้อหาในบทนี้มีวัตถุประสงค์ดังต่อไปนี้

- เพื่อให้เข้าใจสัญญาณการเชื่อมต่อกับอุปกรณ์อินพุต/เอาต์พุตชนิดต่างๆ ของบอร์ด Pi3/Pi4
- เพื่อให้รู้จักโครงสร้างและการทำงานด้านอินพุต/เอาต์พุตของบอร์ด Pi3/Pi4
- เพื่อให้เข้าใจกลไกการติดต่อกับอุปกรณ์อินพุต/เอาต์พุตชนิดต่างๆ
- เพื่อให้เข้าใจหลักการ Memory Mapped I/O, อินเทอร์รัปท์ และ Direct Memory Access โดยใช้ซีพียูของ ARM เป็นกรณีศึกษา

เนื้อหาในบทต่างๆ ที่ผ่านมาอธิบายโครงสร้างของบอร์ด Pi3/Pi4 ตามรูปที่ 6.1 ว่าประกอบด้วย ชิป BCM2837/BCM2711 ภายใต้ชีพียู ARM Cortex A53/A72 จำนวน 4 แกนประมวลผลเป็นศูนย์กลาง เชื่อมต่อด้วยสายสัญญาณจำนวนมากกับ

- ชิป SDRAM ชนิด DDR2 ความจุขนาด 1 กิกะไบต์ (GiB) ทำหน้าที่เป็นหน่วยความจำภายในภาพ
- การ์ดหน่วยความจำชนิดไมโคร SD ผ่านสาย SDIO (Secure Digital Input/Output)
- วงจรอินพุต/เอาต์พุตภายในชิป BCM2837 และ

- ชิปหรืออุปกรณ์ภายนอกต่างๆ เช่น ชิป USB/Ethernet ชิป WiFi และ Bluetooth



รูปที่ 6.1: การเชื่อมต่ออุปกรณ์ต่างๆ บนบอร์ด Pi3 โดยมีชิป BCM2837 เป็นศูนย์กลาง ที่มา: [xdevs.com](http://xdevs.com)

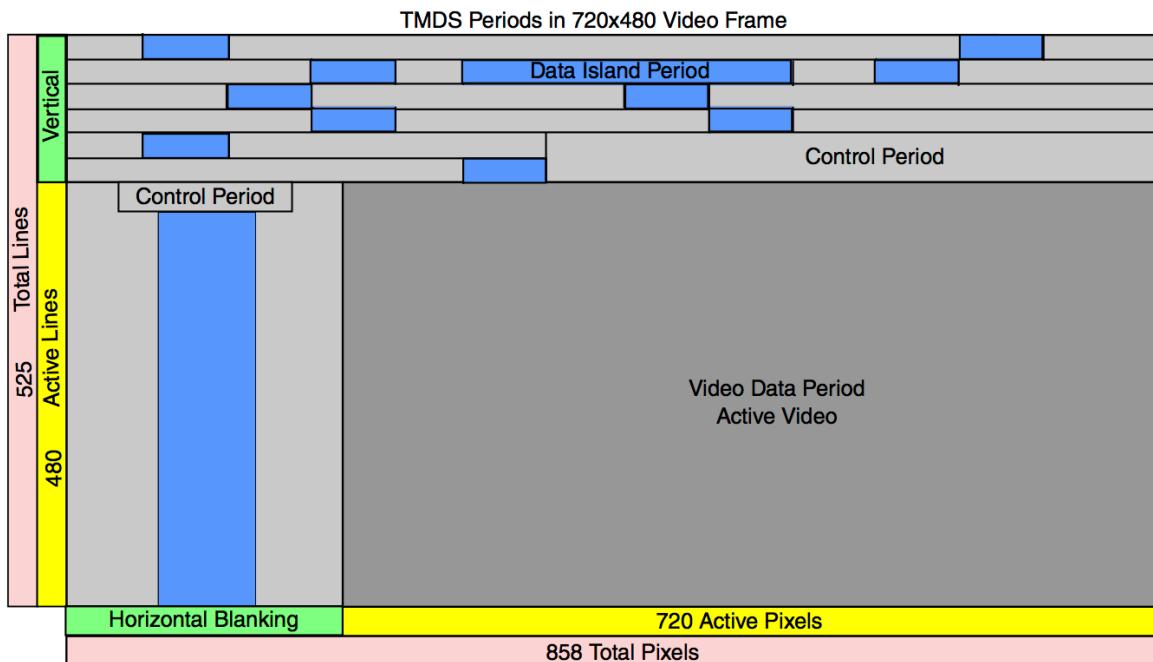
เนื้อหาในบทนี้จะเน้นเรื่องของการเชื่อมต่อชิป BCM2837 กับวงจรและอุปกรณ์ภายนอก ตัวชิป BCM2837 นี้ประกอบด้วยขาทั้งหมด 200 ขา ซึ่งส่วนใหญ่จะเป็นขาเชื่อมต่ออุปกรณ์หรือวงจรอินพุตและเอาต์พุต เช่น HDMI USB เครือข่ายไร้สาย เครือข่ายสาย เป็นต้น

ด้านการแสดงผล บอร์ด Pi3/Pi4 รองรับการเชื่อมต่อกับจอแสดงผลได้ 3 ชนิด คือ จอมอนิเตอร์ด้วยสายสัญญาณคอมโพสิตวีดีโอ จอภาพ LCD ขนาดใหญ่ผ่านสายสัญญาณ HDMI และ จอภาพ LCD ขนาดเล็กผ่านสายสัญญาณ DS1 บอร์ดมีพอร์ต USB 2.0 จำนวน 4 พอร์ต เช่น คีย์บอร์ด เม้าส์ เป็นต้น บอร์ดสามารถเชื่อมต่ออินเทอร์เน็ตด้วยสายสัญญาณ Ethernet หรือสายแลน (LAN: Local Area Network) และเชื่อมต่อกับระบบเครือข่ายแบบไร้สาย ด้วยชิป BCM43438 ซึ่งภายในมีมอดูล WiFi และ Bluetooth

กลไกการเชื่อมต่อที่ลึกลงไป จะปรากฏในการทดลอง การทดลองที่ 9 ภาคผนวก | การศึกษาและปรับแก้ อินพุต/เอาต์พุตต่างๆ

## 6.1 สัญญาณ HDMI สำหรับจอภาพ LCD ขนาดใหญ่

เนื้อหาในหัวข้อนี้ต่อเนื่องจาก HDMI (High-Definition Multimedia Interface) ในหัวข้อที่ 3.1.3 สำหรับเข้ามต่อจอแสดงผลภายนอก สัญญาณ HDMI คือ สัญญาณสำหรับการเข้ามต่ออุปกรณ์ภาพและเสียง เพื่อแทนที่การเชื่อมต่อรูปแบบเดิมๆ เช่น สัญญาณคอมโพสิตวีดีโอ และแบบ S-Video

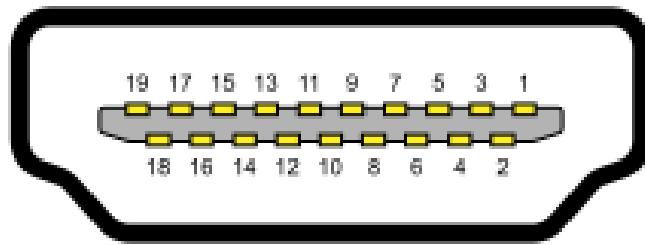


รูปที่ 6.2: ภาพความละเอียด 720 จุด x480 เส้นที่ผู้ใช้มองเห็น แบ่งเป็นการส่งแพ็กเก็ตข้อมูลจุดภาพและแพ็กเก็ตควบคุมทางช่องสัญญาณ TMDS ภายในช่วงเวลาต่างๆ ที่มา: [freebsd.org](http://freebsd.org)

การเชื่อมต่อแบบ HDMI เป็นการถ่ายโอนสัญญาณแบบดิจิทัล สามารถส่งได้ทั้งข้อมูลภาพดิจิทัลและข้อมูลเสียงดิจิทัลไปพร้อมๆ กันด้วยอัตราบิตร率สูงระดับกิกะบิตต่อวินาที ตัวอย่างในรูปที่ 6.2 เป็นภาพความละเอียด 720 จุด x 480 เส้นที่ผู้ใช้มองเห็น แบ่งเป็น การส่งแพ็กเก็ตข้อมูลจุดภาพ แพ็กเก็ตข้อมูลเสียง และแพ็กเก็ตควบคุมในช่วงเวลาต่างๆ

การเชื่อมต่อด้วยสัญญาณ HDMI เหมาะสำหรับการแสดงผลจากเครื่องคอมพิวเตอร์หรือเครื่องเล่นมีเดีย (Media Player) กับจอภาพความละเอียดสูงระดับเอชดี (HD: High Definition) ซึ่งแบ่งเป็นความละเอียด 1280 จุดต่อเส้น x 720 เส้น (ย่อว่า 720p) และความละเอียด 1920 จุดต่อเส้น x 1080 เส้น (ย่อว่า 1080p) ความละเอียดภาพที่สูงกว่า เรียกว่า ความละเอียด Ultra HD ช่องสัญญาณ HDMI เวอร์ชันปัจจุบันขณะที่เขียนตำรา คือ เวอร์ชัน 2.1 รองรับภาพที่ละเอียดสูงเฟรมละ 8000 จุดต่อเส้นที่ 60 เฟรมต่อวินาที (ย่อว่า 8K60) และเฟรมละ 4000 จุดต่อเส้นที่ 120 เฟรมต่อวินาที (ย่อว่า 4K120) และเพิ่มความละเอียดเพิ่มถึงเฟรมละ 10,000 จุดต่อเส้น (ย่อว่า 10K) ซึ่งจะทำให้อัตราบิตร率เพิ่มสูงเป็น 48 กิกะบิตต่อวินาที

สาย HDMI ที่ใช้เชื่อมต่อส่วนใหญ่จะเป็นตัวเมีย (Female) ทั้งสองด้าน รูปที่ 6.3 แสดงภาพตัดขวางของหัวเชื่อมต่อ HDMI ชนิดตัวเมีย (Female) ประกอบด้วยขา 19 ขา มีรายชื่อตามตารางที่ 6.1 ตามหมายเลขอ้างอิง และวัตถุประสงค์ของสัญญาณชนิด HDMI เวอร์ชัน 1.4 ที่มาและ รายละเอียดเพิ่มเติม: [wikipedia](https://en.wikipedia.org)



รูปที่ 6.3: หัวเชื่อมต่อ HDMI ชนิด Female ประกอบด้วยขาสัญญาณทั้งหมด 19 ขา ที่มา: [wikipedia.org](https://en.wikipedia.org)

ตารางที่ 6.1: หมายเลขขา ชื่อ และวัตถุประสงค์ของสายสัญญาณชนิด HDMI เวอร์ชัน 1.4 ที่มา: [wikipedia.org](https://en.wikipedia.org)

ขา	ชื่อ	วัตถุประสงค์
1	TMDS Data2+	ข้อมูลช่วง Control Period ข้อมูลเลน 2 ขั้วบวก
2	TMDS Data2 Shield	ชีล์ดสำหรับข้อมูลเลน 2
3	TMDS Data2-	ข้อมูลเลน 2 ขั้วลบ
4	TMDS Data1+	ข้อมูลเลน 1 ขั้วบวก
5	TMDS Data1 Shield	ชีล์ดสำหรับข้อมูลเลน 1
6	TMDS Data1-	ข้อมูลเลน 1 ขั้วลบ
7	TMDS Data0+	ข้อมูลภาพดิจิทัล ข้อมูลเลน 0 ขั้วบวก
8	TMDS Data0 Shield	ชีล์ดสำหรับข้อมูลเลน 0
9	TMDS Data0-	ข้อมูลเลน 0 ขั้วลบ
10	TMDS Clock+	สัญญาณคล็อกขั้วบวก
11	TMDS Clock Shield	ชีล์ดสำหรับสัญญาณคล็อก
12	TMDS Clock-	สัญญาณคล็อกขั้วลบ
13	CEC	ควบคุมอุปกรณ์ต่อพ่วง
14	HEAC+	Ethernet and Audio Return+
15	SCL	สัญญาณคล็อกสำหรับช่อง DDC
16	SDA	สายข้อมูลสำหรับช่อง DDC
17	Ground	กราวน์ด (0 โวลต์)
18	+5.0 V	ไฟกระแสตรง 5.0 โวลต์
19	Plug Detect HEAC-	ขาตรวจจับการเชื่อมต่อ Ethernet and Audio Return-

จากตารางที่ 6.1 ผู้อ่านจะสังเกตได้ว่า สัญญาณ HDMI มีช่องสื่อสารอยู่ 5 ช่องแยกอิสระจากกัน ได้แก่

- ช่อง TMDS (Transition-Minimized Differential Signaling) สำหรับส่งแพ็กเก็ต (Packet) ข้อมูลเป็นติจิทัลในทั้งเวลาของการแสดงภาพเพียง 1 เฟรม ประกอบด้วย ช่วงส่งแพ็กเก็ตข้อมูลภาพ (Video Data Period) ช่วงส่งแพ็กเก็ตข้อมูลเสียง (Audio Data Period) และช่วงส่งแพ็กเก็ตควบคุม (Control Period) สำหรับสัญญาณควบคุม ยกตัวอย่างเช่น สัญญาณ HSYNC (Horizontal Synchronization) และ VSYNC (Vertical Synchronization) เป็นต้น

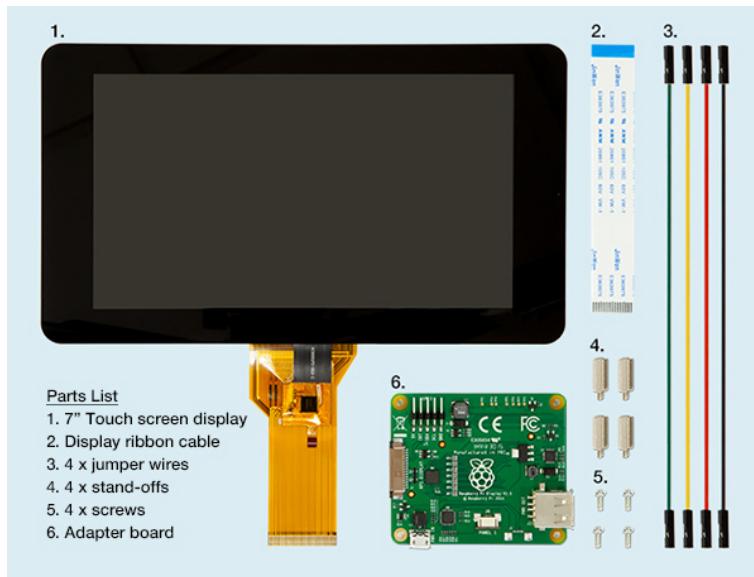
รูปที่ 6.2 แสดงตัวอย่างช่วงเวลาของการส่งแพ็กเก็ตข้อมูลภาพจะใช้สัญลักษณ์สีเทา และแพ็กเก็ตควบคุม จะใช้สัญลักษณ์สีฟ้า ในช่วงเวลาต่างๆ สำหรับการแสดงภาพด้วยความละเอียด 720x480 หรือ 480 เส้นๆ ละ 720 จุดต่อ 1 เฟรม แต่ละเฟรมใช้เวลา 33 มิลลิวินาทีเพื่อการแสดงผลหรือคิดเป็น 30 เฟรมต่อวินาที ภายในระยะเวลา 33 มิลลิวินาที มีการส่งสัญญาณภาพจริงเป็นจำนวน 525 เส้นๆ ละ 858 พิกเซล แต่ผู้ใช้งานจะมองเห็นเพียง 480 เส้นๆ ละ 720 พิกเซลในตำแหน่งที่มีสีเทาเข้มเท่านั้น ส่วนจำนวนเส้นและจุดที่เกินเพื่อไว้สำหรับช่วงเวลา Vertical Blanking และ Horizontal Blanking ช่วงเวลาทั้งสองจะใช้ในการส่งสัญญาณควบคุมและข้อมูลอื่นๆ โดยแพ็กเก็ตข้อมูลภาพเริ่มต้นจากภาพเส้น (Line) บนสุดและตำแหน่งจุดภาพซ้ายสุดไปจุดภาพขวาสุด แล้วจะขยับลงมา 1 เส้นเพื่อเริ่มจากตำแหน่งจุดภาพซ้ายสุดไปจุดภาพขวาสุดเช่นเดิม การส่งแพ็กเก็ตจะขยับลงมาเรื่อยๆ จนถึงเส้นภาพสุดล่างสุด แล้วจึงเริ่มต้นภาพเพริมใหม่ด้วยเส้นภาพบนสุดเช่นเดิม

ในสาย HDMI หนึ่งเส้นประกอบด้วยเลนสัญญาณ TMDS จำนวน 3 เลน สำหรับส่งแพ็กเก็ตข้อมูลพร้อมกัน รายละเอียดเพิ่มเติมเกี่ยวกับสัญญาณ TMDS ที่ [Wikipedia](#)

- **ช่อง DDC** (Display Data Channel) ใช้สื่อสารระหว่างจอแสดงผลกับเครื่องเล่นด้วยมาตรฐาน I<sup>2</sup>C (อ่านว่า ไอสแควร์ซี) เพื่อกำหนดรูปแบบและความละเอียดของภาพวีดีโอและเสียง และยังใช้คุ้มครองเนื้อหาดิจิทัลแบนด์วิดธ์สูง (High-bandwidth Digital Content Protection: HDCP) เช่น ภาพยนต์ที่มีลิขสิทธิ์ เป็นต้น
- **ช่อง CEC** (Consumer Electronics Control) ผู้ใช้งานสามารถควบคุมอุปกรณ์ต่อพ่วงผ่านช่อง CEC นี้ได้มากถึง 15 ตัว เป้าหมายคือ อุปกรณ์สามารถทำงานร่วมกันและใช้เครือข่ายอินเทอร์เน็ตไปพร้อมๆ กัน
- **ช่อง ARC** (Audio Return Channel) หรือช่องสัญญาณเสียงคืนกลับ เพื่อใช้สาย HDMI เชื่อมต่อกับตัวถอดรหัสเสียงและเครื่องขยายเสียงผ่านทางช่อง ARC
- **ช่อง HEC** (HDMI Ethernet Channel) ใช้เชื่อมต่ออุปกรณ์อื่นๆ ผ่านทางสาย HDMI ตั้งแต่เวอร์ชัน 1.3 เป็นต้นมา

การทดลองที่ 9 ภาคผนวก I มีการปรับแต่งความละเอียดของการแสดงผลผ่านสาย HDMI

## 6.2 สัญญาณ DSI สำหรับจอภาพ LCD ขนาดเล็ก



รูปที่ 6.4: จอยแสดงผลสำหรับเชื่อมต่อระหว่างบอร์ด Pi3/Pi4 ด้วยสัญญาณการแสดงผลแบบอนุกรม (Display Serial Interface) ที่มา: [element14.com](http://element14.com)

**สัญญาณ DSI** (Display Serial Interface) เหมาะสำหรับเชื่อมต่อจอภาพ LCD ขนาดเล็ก กับชิปซีพียูบนอุปกรณ์เคลื่อนที่ เช่น โทรศัพท์สมาร์ตโฟน แท็บเล็ต คอมพิวเตอร์โน้ตบุ๊ก เป็นต้น เพื่อการแสดงผลในรูปของกราฟิกใหม่ด้วยความละเอียดสูง สัญญาณ DSI นี้ถูกกำหนดให้เป็นมาตรฐานโดยองค์กรชื่อ MIPI (Mobile Industry Processor Interface)

ตารางที่ 6.2: หมายเลข และหน้าที่ของสายสัญญาณ DSI สำหรับจอภาพ LCD ขนาดเล็ก ประกอบด้วยข้อมูลภาพจำนวน 2 เลน ที่มา: [wikipedia.org](http://wikipedia.org)

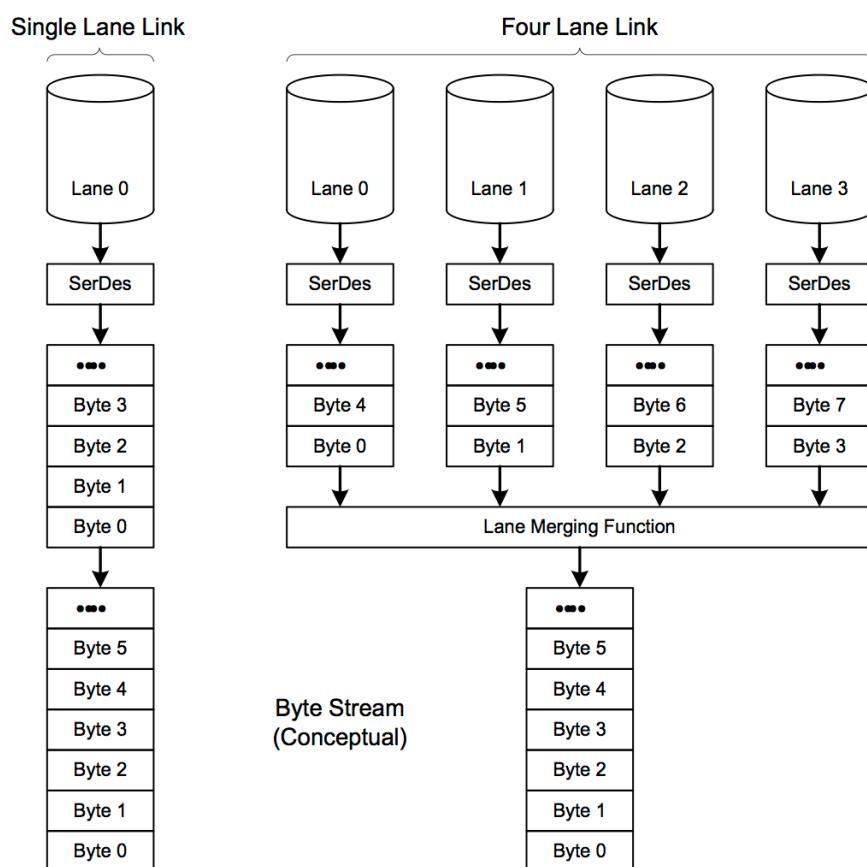
ขา	ชื่อ	หน้าที่
1	Ground	กราวน์ด (0 โวลต์)
2	Data Lane 1-	เลนข้อมูล 1 ขาลง
3	Data Lane 1+	เลนข้อมูล 1 ขาขึ้น
4	Ground	กราวน์ด (0 โวลต์)
5	Clock-	เลนคลีอกขาลง
6	Clock+	เลนคลีอกขาขึ้น
7	Ground	กราวน์ด (0 โวลต์)
8	Data Lane 0-	เลนข้อมูล 0 ขาลง
9	Data Lane 0+	เลนข้อมูล 0 ขาขึ้น
10	Ground	กราวน์ด (0 โวลต์)
11		
12		
13	Ground	กราวน์ด (0 โวลต์)
14	+3.3 V	ไฟกระแสตรง +3.3 โวลต์
15	+3.3 V	ไฟกระแสตรง +3.3 โวลต์

คอนเนคเตอร์ S2 บนบอร์ด Pi3/Pi4 เป็นหัวเชื่อมต่อสัญญาณ DSI สายที่ใช้จะต้องเป็นสายจำนวนหลายเส้น

เรียงติดกันบนราบ เรียกว่า **สายแพ** (Ribbon) จำนวน 15 ชา สามารถส่งข้อมูลพร้อมๆ กันจำนวน 2 เลน แต่ละเลนเป็นการส่งข้อมูลภาพแบบอนุกรม (Serial) ตารางที่ 6.2 แสดงหมายเลข ชื่อและวัตถุประสงค์ของชาทั้ง 15 ชา โดย ชา 5 และ 6 จะถูกกำหนดให้เป็นเลนสัญญาณคลื่อก ชา 8 และ 9 จะถูกกำหนดให้เป็นเลนข้อมูลภาพหมายเลข 0 ชา 2 และ 3 จะถูกกำหนดให้เป็นเลนข้อมูลภาพหมายเลข 1 เป็นต้น

จอภาพ LCD ขนาดเล็กจะทำงานใน **โหมดรับคำสั่ง** เพื่อให้ซีพียูกำหนดค่าต่างๆ ของรีจิสเตอร์ควบคุมการทำงานของจอตามที่ต้องการ โดยอาศัยการรับส่งข้อมูลภาพและคำสั่งในเลนที่ 0 ได้ทั้งสองทิศทาง (BiDirectional)

การทำงานในโหมดแสดงผล จอ LCD จะรับข้อมูลภาพจากทุกๆ เลน โดยเลนที่ 1 เป็นต้นไปจะทำหน้าที่รับข้อมูลภาพได้เพียงทิศทางเดียวเท่านั้น ตามรูปที่ 6.5 มาตรฐานสัญญาณ DSI แบ่งการส่งข้อมูลภาพเป็นชนิดเลนเดียว (Single Lane) และหลายๆ เลนโดยมีจำนวนตั้งแต่ 2 เลนขึ้นไป เพื่อกระจายข้อมูลภาพแต่ละใบต่อไปทีละเลน ยกตัวอย่างเช่น การส่งภาพแบบ 4 เลน ข้อมูลภาพใบต์ที่ 0, 4, 8, ... จะส่งมาทางเลนหมายเลข 0 ข้อมูลภาพใบต์ที่ 1, 5, 9, ... จะส่งมาทางเลนหมายเลข 1 ข้อมูลภาพใบต์ที่ 2, 6, 10, ... จะส่งมาทางเลนหมายเลข 2 ข้อมูลภาพใบต์ที่ 3, 7, 11, ... จะส่งมาทางเลนหมายเลข 3 และสลับกันไปแบบนี้เรื่อยๆ เมื่อปลายทาง (จอภาพ) รับข้อมูลได้สำเร็จ วงจรรับจะนำข้อมูลภาพเหล่านั้นรวมกัน (Lane Merging Function) เป็นภาพเฟรมเดียวกัน การส่งข้อมูลภาพจำนวนหลายๆ เลนพร้อมกันช่วยให้แสดงภาพแต่ละเฟรมเร็วขึ้น รองรับการแสดงผลที่ละเอียดมากขึ้น รองรับอัตราการเปลี่ยนแปลงภาพต่อวินาทีได้มากขึ้น การเคลื่อนไหวของภาพจึงต่อเนื่องไม่กระตุก เพิ่มอรรถรสในการรับชมมากขึ้น



รูปที่ 6.5: เปรียบเทียบสัญญาณ DSI ชนิดหนึ่งเลน (Single Lane) และชนิด 4 เลน ที่มา: [wikipedia.org](https://wikipedia.org)

### 6.3 สัญญาณ CSI สำหรับเชื่อมต่อกล้องขนาดเล็ก



รูปที่ 6.6: การเชื่อมต่อระหว่างบอร์ด Pi3/Pi4 และกล้องขนาดเล็กด้วยสัญญาณกล้องแบบอนุกรม (Camera Serial Interface) ที่มา: [element14.com](http://element14.com)

บอร์ด Pi3/Pi4 สามารถเชื่อมต่อกล้องขนาดเล็กตามมาตรฐาน CSI (Camera Serial Interface) ซึ่งกำหนดโดยองค์กร [MIPi.org](http://MIPi.org) จึงมีความคล้ายคลึงกับสัญญาณ DSI ข้อมูลภาพจากกล้องจะส่งผ่านสาย CSI ผ่านเลนข้อมูลตามรูปที่ 6.5 เพื่อไปรวมกันเป็นภาพเดียวที่ปลายทาง (หน่วยความจำบัฟเฟอร์)

กล้องจะเชื่อมกับบอร์ด Pi3/Pi4 ซึ่ง เป็นหัวช้อแก็ตชนิด ZIF (Zero Insertion Force) ชนิด 15 ขา ติดยึดบนพื้นผิวของแผ่นวงจรพิมพ์ (Surface Mount) มาตรฐาน CSI กำหนดให้สัญญาณที่ใช้สำหรับเลนข้อมูลเป็น ชนิด Low Voltage Differential Signalling (SubLVDS) เช่นกัน โดยปรับปรุงมาจากมาตรฐานสัญญาณ IEEE1596.3 LVDS สำหรับอุปกรณ์ที่ใช้ไฟกระแทกแรงต่ำประมาณ 1.2 โวลต์ เพื่อให้สามารถส่งข้อมูลต่อเลนได้สูงสุด 800 - 1,000 เมกะบิตต่อวินาที (Mbps)

ตารางที่ 6.3 แสดงหมายเลขอ้างอิงและวัตถุประสงค์ของสัญญาณ CSI ดังนี้

- ขา CAM1\_D0-/CAM1\_D0+ และ ขา CAM1\_D1-/CAM1\_D1+ คือ ขั้วลบและบวกของเลนข้อมูลที่ 0 และ 1 ตามลำดับ โดยชิปในกล้องจะสร้างสัญญาณแล้วส่งผ่านสายแพให้กับวงจรที่รับภาพในชิป BCM2837 ผ่านเลนข้อมูลทั้งสองเลนนี้คล้ายกับสัญญาณ DSI แบบหลายเลนในรูปที่ 6.5

ผู้ใช้สามารถกำหนดรูปแบบของข้อมูลภาพที่ส่งแบบต่างๆ ดังนี้

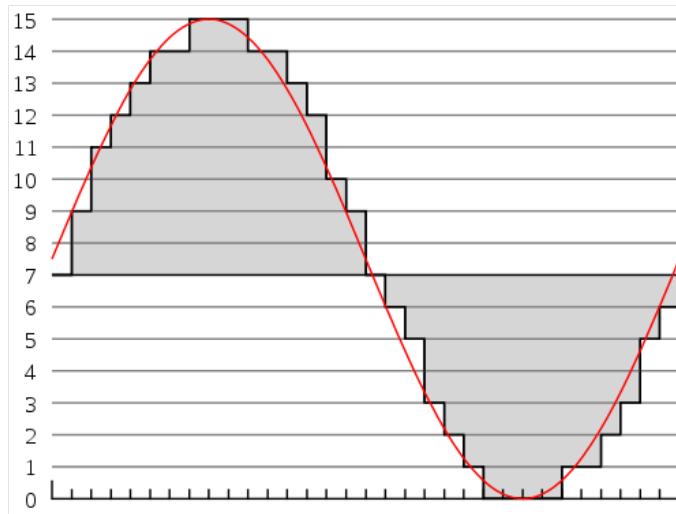
- RGB (Red Green และ Blue)
- RAW หรือข้อมูลภาพที่ไม่มีการปรับแต่งใดๆ
- YUV (Y: Luminance หรือ ความสว่าง U และ V คือ องค์ประกอบของสี Chrominance 2 ชนิด)

- ขา CAM1\_C- และขา CAM1\_C+ คือ ขาลงและขาขึ้นของสัญญาณคลีกอกตามลำดับ เพื่อซิงโครไนซ์การส่งข้อมูลภาพในเลนต่างๆ ที่ความเร็วสูง
- ขา SDA0 และขา SCL0 คือ สัญญาณข้อมูลและสัญญาณคลีกอกตามมาตรฐาน I<sup>2</sup>C เพื่อควบคุมการทำงานของกล้อง เช่น ความละเอียดของภาพ รูปแบบของข้อมูล เป็นต้น โดยการรับส่งข้อมูล SDA0 จะซิงโครไนซ์กับสัญญาณคลีกอก SCL0 ซึ่งมีความถี่ต่ำกว่าสัญญาณคลีกอก CAM1\_C มาก ผู้อ่านสามารถค้นควารายละเอียดเพิ่มเติมที่ [wikipedia](https://en.wikipedia.org)

ตารางที่ 6.3: หมายเลขขา ชื่อ และวัตถุประสงค์ของสัญญาณชนิด CSI [wikipedia.org](https://en.wikipedia.org)

ขา	ชื่อ	วัตถุประสงค์
1	Ground	กราวน์ด (0 โวลต์)
2	CAM1_D0-	ข้อมูลภาพเลน 0 ขั้วลง
3	CAM1_D0+	ข้อมูลภาพเลน 0 ขั้วขึ้น
4	Ground	กราวน์ด (0 โวลต์)
5	CAM1_D1-	ข้อมูลภาพเลน 1 ขั้วลง
6	CAM1_D1+	ข้อมูลภาพเลน 1 ขั้วขึ้น
7	Ground	กราวน์ด (0 โวลต์)
8	CAM1_C-	สัญญาณคลีกอกขั้วลง
9	CAM1_C+	สัญญาณคลีกอกขั้วขึ้น
10	Ground	กราวน์ด (0 โวลต์)
11	CAM_GPIO	ขา GPIO
12	CAM_CLK	ขาสัญญาณคลีกอก
13	SCL0	สัญญาณคลีกอกสำหรับ I <sup>2</sup> C
14	SDA0	สัญญาณข้อมูลสำหรับ I <sup>2</sup> C
15	+3.3 V	ไฟกระแสตรง +3.3 โวลต์

## 6.4 สัญญาณ PCM สำหรับข้อมูลเสียงดิจิทัล



รูปที่ 6.7: สัญญาณดิจิทัล PCM (Pulse Code Modulation) ความละเอียด 16 ระดับสูม (Sampling) จากสัญญาณอนาล็อกรูปคลื่นไซน์ (Sine Wave) ด้วยความถี่สูง 26 เท่าของความถี่สูงสุด ( $f_s=26f_{max}$ ) ที่มา: [wolfcrow.com](http://wolfcrow.com)

สัญญาณชนิด PCM คือ สัญญาณดิจิทัลพื้นฐาน ได้จากการแปลงสัญญาณอนาล็อกเป็นดิจิทัล (Analog to Digital: A2D) สัญญาณ PCM ได้รับความนิยมแพร่หลายในอดีตจนถึงปัจจุบัน เช่น ข้อมูลเสียงในแผ่นซีดีเพลง (Audio Compact Disc) โทรศัพท์บ้านพื้นฐาน และอื่นๆ ชิป BCM2837/BCM2711 บนบอร์ด Pi3/Pi4 สามารถรับข้อมูลเสียงดิจิทัล ในรูปแบบ PCM นี้ ไปประมวลผล แล้วแปลงสัญญาณดิจิทัลให้เป็นสัญญาณอนาล็อกเพื่อส่งต่อให้กับลำโพงภายนอก ในหัวข้อที่ 6.5

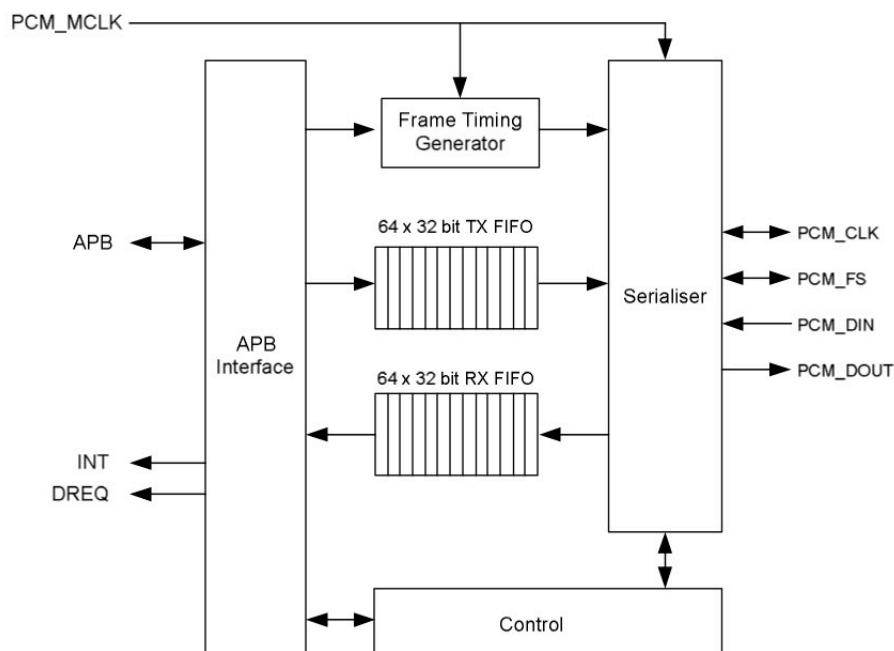
สัญญาณอนาล็อกรูปคลื่นไซน์ถูกสูม (Sampling) ด้วยความถี่สูม (Sampling Frequency,  $f_s$ ) ที่ความถี่สูงกว่า 2 เท่าความถี่สูงสุดของสัญญาณอนาล็อก ( $f_s > 2f_{max}$ ) โดย  $f_{max}$  คือ ความถี่สูงสุดของสัญญาณอนาล็อก ตามกฎของ Nyquist ยกตัวอย่าง เช่น

- ตัวอย่างที่ 1 สัญญาณรูปคลื่นไซน์ (Sine Wave) ในรูปที่ 6.7 สูมด้วยความถี่สูงเป็น 26 เท่าของความถี่สูงสุด ( $f_s = 26f_{max}$ ) และทำการแปลงเป็นสัญญาณดิจิทัลชนิด PCM (Pulse Code Modulation) ด้วยความละเอียด  $16=2^4$  ระดับ กล้ายเป็นข้อมูลเลขจำนวนเต็มชนิดไม่มีเครื่องหมายขนาด 4 บิตต่อการสูม 1 ครั้ง
- ตัวอย่างที่ 2 สัญญาณเสียงสนทนาผ่านโทรศัพท์พื้นฐานจะสูมด้วยความถี่ 8,000 ครั้งต่อวินาที ( $f_s=8$  กิโลเฮิรตซ์) ซึ่งจะตรงกับคาบเวลา  $1/8000 = 125$  มิลลิวินาที ด้วยความละเอียด  $256=2^8$  ระดับ กล้ายเป็นเลขจำนวนเต็มชนิดไม่มีเครื่องหมายยาว 8 บิตต่อการสูม 1 ครั้ง
- ตัวอย่างที่ 3 สัญญาณเสียงเพลงคุณภาพระดับแผ่นซีดีจะสูมด้วย ความถี่ 44,100 ครั้งต่อวินาที ( $f_s=44.1$  กิโลเฮิรตซ์) ซึ่งจะตรงกับคาบเวลา  $1/44100 = 22.67$  มิลลิวินาที ด้วยระดับความละเอียด  $65,536=2^{16}$  ระดับ เพื่อให้เป็นข้อมูลเสียงดิจิทัล PCM หรือเลขจำนวนเต็มชนิดไม่มีเครื่องหมายยาว 16 บิตต่อการสูม 1 ครั้ง ทั้งนี้ ของสัญญาณเสียงด้านซ้ายและด้านขวาจะการสูมพร้อมกัน รวมเป็นข้อมูล 32 บิตต่อการสูม 1 ครั้ง

ในรูปที่ 6.8 มอดูล PCM ภายในชิป BCM2837/BCM2711 ประกอบด้วย

- วงจรเชื่อมต่อกับซีพียู ARM Cortex A53/A72 ผ่านบัส APB (ARM Peripheral Bus)
- หน่วยความจำบันเฟอร์สำหรับส่ง (Transmit) และรับ (Receive) ขนาด  $64 \times 32$  บิต เป็นบันเฟอร์ด้านรับ 1 ชุด และด้านส่งอีก 1 ชุด รวม 2 ชุด ทำหน้าที่พักรักษาข้อมูลชั่วคราวระหว่างหน่วยความจำภายในภาพ เพื่อรอให้ซีพียูประมวลผลและส่งสัญญาณเสียงไปยังลำโพง
- มодูลเชื่อมต่อกับอุปกรณ์ภายนอกตามมาตรฐาน I<sup>2</sup>S (Inter IC Sound) (อ่านว่า ไอสแควร์เอส) เพื่อเชื่อมต่อกับภายนอกชิป BCM2837/BCM2711 ประกอบด้วยสายสัญญาณจำนวน 4 เส้นแบบอนุกรม ได้แก่
  - สัญญาณ PCM\_CLK - สัญญาณคลื่นสำหรับส่งข้อมูลแต่ละบิตด้วยความถี่สูง
  - สัญญาณ PCM\_DIN - สัญญาณข้อมูลเสียงขาเข้าหรือขารับ (Receive) จะรับสัญญาณตามความถี่ของ PCM\_CLK โดยจะรับบิตสูง (Most Significant Bit) ก่อนและบิตสุดท้ายคือ บิตต่ำสุด (Least Significant Bit) เช่นเดียวกับ PCM\_CLK
  - สัญญาณ PCM\_DOUT - สัญญาณข้อมูลเสียงขาออกหรือขาส่ง (Transmit) จะมีทิศทางที่ตรงกันข้ามกับ PCM\_DIN
  - สัญญาณ PCM\_FS - สัญญาณซิงค์เฟรมข้อมูล (Frame Sync) เพื่อใช้ประกาศการเริ่มต้นและสิ้นสุดเฟรมข้อมูล โดยหนึ่งเฟรมสามารถกำหนดความยาวได้ ทั้งนี้ขึ้นอยู่กับขนาดจำนวนบิตข้อมูล และความถี่สูง (Sampling Frequency)

รายละเอียดเพิ่มเติมที่ [wikipedia](#)



รูปที่ 6.8: มอดูล PCM ประกอบด้วย วงจรเชื่อมต่อกับซีพียูผ่านบัส APB, หน่วยความจำบันเฟอร์สำหรับส่งและรับข้อมูลเสียงดิจิทัล และวงจรเชื่อมต่อชนิด I<sup>2</sup>S ที่มา: [Broadcom \(2012\)](#)

มอดูลนี้รองรับการทำงานทั้งสามรูปแบบ คือ การโพลลิ่ง (Polling) การอินเทอร์รัปท์ (Interrupt) รายละเอียดเพิ่มเติมในหัวข้อที่ [6.12](#) และ การเข้าถึงหน่วยความจำโดยตรง (Direct Memory Access) รายละเอียดเพิ่มเติมในหัวข้อที่ [6.13](#)

## 6.5 สัญญาณภาพและเสียงสำหรับจอทีวีอนาคต

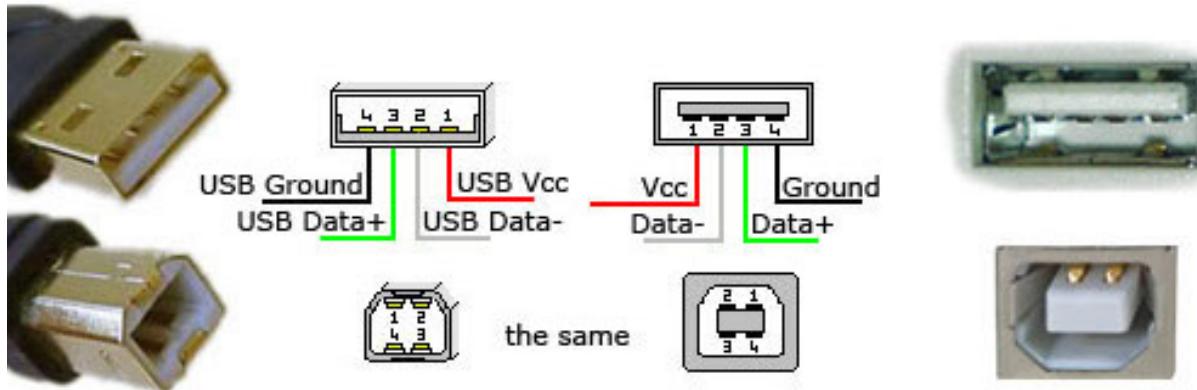


รูปที่ 6.9: แจ็คขนาด 3.5 มม. (กลาง) ชนิด 4 ขั้วสำหรับเสียบกับบอร์ด Pi3/Pi4 (ขวา) ส่งสัญญาณภาพไปยังแจ็ค RCA (เหลือง) และสัญญาณเสียงไปยังแจ็ค RCA (แดงและขาว) ที่มา: [stackexchange.com](http://stackexchange.com)

ช่องเสียบแจ็คในรูปที่ 6.9 ด้านขวา ใช้เชื่อมต่อสัญญาณภาพ (อนาคต) และเสียง (อนาคต) จากบอร์ด Pi3/Pi4 เข้ากับจอทีวีหรือจอมอนิเตอร์ที่มีช่องอินพุตเป็น คอมโพลิตวิดีโอ (Composite Video) และเสียงสเตอริโอ (Stereo) หรือ โทรทัศน์บางเครื่องเรียกว่าช่อง AV (Audio/Video) ซึ่งสามารถใช้ทดแทนจอภาพ LCD ได้ แต่จะได้สัญญาณภาพความละเอียดต่ำกว่าสัญญาณ HDMI โดยแจ็ค 3.5 มม. (กลาง) นี้เป็นชนิด 4 ขั้วสำหรับเสียบกับบอร์ด Pi3/Pi4 เชื่อมต่อสัญญาณภาพไปยังแจ็ค RCA (เหลือง) และสัญญาณเสียงไปยังแจ็ค RCA (แดงและขาว) ผู้อ่านสามารถหาชิ้นสายสัญญาณสำเร็จรูปนี้ได้ทั่วไป หรือจะบัดกรีเชื่อมต่อสายเองได้เช่นกัน

สัญญาณภาพดิจิทัล ชื่อ TV DAC ภายในชิป BCM2837 ถูกแปลงเป็นสัญญาณภาพอนาคตโดยใช้ DAC (Digital to Analog Converter) ชนิดตัวกรองความถี่ต่ำผ่าน หรือ Low-Pass Filter ในรูปที่ 6.1 ในทำนองเดียวกัน สัญญาณเสียงดิจิทัลชนิด PCM ในหัวข้อที่ 6.4 แบ่งเป็นสัญญาณเสียงช่องซ้ายและช่องขวาแบบสเตอริโอ จะถูกแปลงเป็นสัญญาณเสียงอนาคตโดยใช้ DAC (ชนิดตัวกรองความถี่ต่ำผ่าน หรือ Low-Pass Filter) ภายในชิป BCM2837 เช่นกัน

## 6.6 สัญญาณ USB สำหรับอุปกรณ์ต่อพ่วงต่างๆ



รูปที่ 6.10: หัวเชื่อมต่อ USB ชนิด A (บน ฝั่งไฮสต์) และ B (ล่าง ฝั่งอุปกรณ์) ประกอบด้วยสัญญาณ 4 เส้น gravitational (0 โวลต์) (GND) Data+ Data- และไฟกระแสตรง 5 โวลต์ ที่มา: [quora.com](https://www.quora.com)

เนื้อหาในหัวข้อนี้ต่อเนื่องจากเรื่อง คีย์บอร์ด ในหัวข้อที่ 3.1.3 และมาส์ ในหัวข้อที่ 3.1.3 ซึ่งนิยมใช้เป็นแบบ USB เพื่อต่อเข้ามกับบอร์ด Pi3/Pi4 และคอมพิวเตอร์ทั่วไป เวอร์ชันปัจจุบันของ USB คือ 3.1 ซึ่งมีความสามารถสูงขึ้นและได้รับความนิยมเพิ่มขึ้น ในทำราเล่มนี้จะกล่าวถึง USB เวอร์ชัน 2.0 ซึ่งเป็นพื้นฐานและมีคุณสมบัติ ดังนี้

- สามารถถ่ายข้อมูลทั่วไป สัญญาณเสียง และสัญญาณภาพได้สูงสุดถึง 1.5 (Low Speed) 12 (Full Speed) และ 480 (High Speed) หน่วยเป็นเมกะบิตต่อวินาที (Mbps)
- สามารถจ่ายไฟกระแสตรงความต่างศักย์ 5 โวลต์ 0.5 แอม培ร์ให้แก่อุปกรณ์ขนาดเล็ก และสูงสุด 1 แอมเบร็ฟสำหรับพอร์ต USB พิเศษ
- สายเคเบิลมีความยาวไม่เกิน 5 เมตร เนื่องจากความต้านทานของสายจะทำให้เกิดโวลต์เจตกรคร่อม (Voltage Drop) ในสาย จนทำให้ความต่างศักย์ไปจ่ายอุปกรณ์ไม่เพียงพอ
- ”Hot Swapping” รองรับการต่อเข้าม ตลอดเวลา และรีเซตอุปกรณ์ที่ต่ออยู่โดยไม่ต้องรีเซตหรือรีบูตระบบปฏิบัติการ

รูปที่ 6.10 แสดง หัวเชื่อมต่อ USB ชนิด A (ฝั่งไฮสต์หรือคอมพิวเตอร์) และ B (ฝั่งอุปกรณ์) ในการเชื่อมต่อทางไฟฟ้าของ USB นั้นจะสายเคเบิลแบบ 4 แกนประมวลผล เพียง 1 เส้นต่อ 1 อุปกรณ์เท่านั้น ซึ่งมีตำแหน่งขาดังนี้

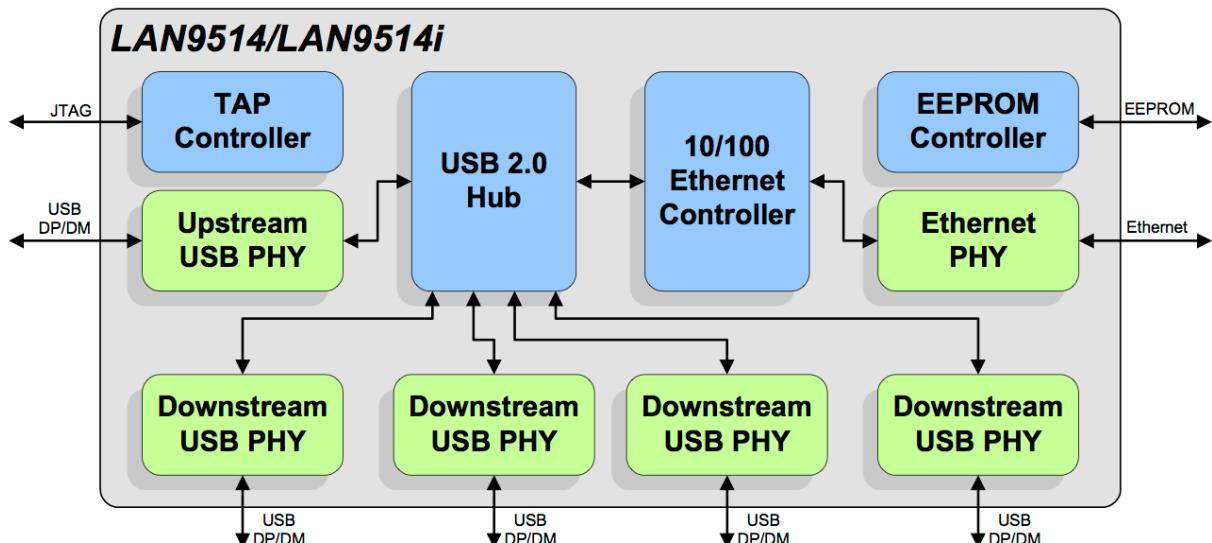
- ขา 1 เป็น +5 โวลต์ สำหรับจ่ายไฟกระแสตรงให้กับอุปกรณ์ขนาดเล็ก เช่น แฟลชไดรฟ์ กล้องเว็บแคม โทรศัพท์สมาร์ตโฟน เป็นต้น
- ขา 2 เป็น D- เป็นสายสัญญาณรับส่งข้อมูลแบบ Differential

- ขา 3 เป็น D+ เป็นสายสัญญาณรับส่งข้อมูลแบบ Differential
- ขา 4 เป็น GND เป็นขากราวน์ด (0 โวลต์) สำหรับไฟกระแสตรง 5 โวลต์

สายส่งข้อมูลของระบบ USB มี 2 สัญญาณ คือ สัญญาณ D+ และ D- ในการส่งสัญญาณแบบ เป็นสายสัญญาณรับส่งข้อมูลแบบติฟเฟอร์เรนเชียล (Differential) คือ

- กรณีในการส่งสัญญาณ ”0” สายสัญญาณ D+ จะมีระดับแรงดันที่ต่ำกว่า D- อย่างน้อย 200 mV (มิลลิโวลต์)
- กรณีในการส่งสัญญาณ ”1” สายสัญญาณ D+ จะมีระดับแรงดันที่สูงกว่า D- อย่างน้อย 200 mV (มิลลิโวลต์)

จากรูปที่ 6.1 BCM2837 จะเชื่อมต่อกับชิป LAN9514 เพื่อขยายเป็น 4 พอร์ต เนื่องจากภายในชิป BCM2837 จะมีรูหัวบ (Root Hub) เพียง 1 พอร์ต โครงสร้างของชิป LAN9514 ตามรูปที่ 6.11 ถูกออกแบบให้ LAN9514 มี USB Hub (Upstream) จำนวน 1 พอร์ต เพื่อเชื่อมกับรูหัวบในชิป BCM2837 และขยายจำนวนพอร์ต (Downstream) เพิ่มเป็น 4 พอร์ต ทำให้บอร์ดสามารถต่อเชื่อมกับคีย์บอร์ด เม้าส์ และอุปกรณ์ USB อื่นๆ รายละเอียดเพิ่มเติมอยู่ในการทดลองที่ 9 ภาคผนวก | หัวข้อที่ I.3.2 นอกจากนี้ ภายใน LAN9514 ยังมีมอดูล Ethernet สำหรับเชื่อมต่อกับเครือข่ายอินเทอร์เน็ตแบบใช้สาย ซึ่งจะได้กล่าวต่อไป



รูปที่ 6.11: โครงสร้างของชิป LAN 9514 ภายในประกอบด้วยวงจร USB Hub และวงจร Ethernet ที่มา: Microchip Technology (2009)

## 6.7 สัญญาณ Ethernet สำหรับสายเชื่อมต่อกับเครือข่ายอินเทอร์เน็ต



รูปที่ 6.12: หัวเชื่อมต่อชนิด RJ45 (ชาญสุด) สำหรับการเชื่อมต่อเครือข่ายท้องถิ่น (Local Area Network) แบบมีสาย Ethernet ที่มา: [cytron.io](http://cytron.io)

เครือข่ายอีเธอร์เน็ต (Ethernet) คือ เทคโนโลยีเครือข่าย LAN (Local Area Network) ที่นิยมใช้กันอย่าง กว้างขวางในอดีตมานั่งปัจจุบัน เพราะเป็นการรับส่งข้อมูลแบบอนุกรมด้วยความเร็วสูง ใช้สายทองแดงในการ ติดตั้ง และต้นทุนไม่สูง เนื่องจากมีอุปกรณ์สนับสนุนเพื่อใช้งานมากที่สุด รูปที่ 6.12 แสดงตำแหน่งของพอร์ต เชื่อมต่อสาย Ethernet บริเวณมุมของบอร์ด Pi3/Pi4 หัวเชื่อมต่อชนิด RJ45 แบบตัวเมีย (Female) โดยใช้สาย CAT5e ขึ้นไป

บอร์ด Pi3/Pi4 นี้ รองรับการเชื่อมต่อ Ethernet ตามมาตรฐาน IEEE 802.3 ชนิด 10/100 BaseT ซึ่งเป็นที่ นิยมทั่วไปในอดีตและปัจจุบัน ด้วยอัตราบิตเรท (Bit Rate) 10/100 เมกะบิตต่อวินาที (Mbps) สายที่ใช้มีชื่อว่าสาย CAT5e หรืออาจจะใช้สายที่มาตรฐานสูงกว่าได้ เช่น CAT6 CAT6A เป็นต้น โดยจะต่อเข้าบอร์ดเข้ากับอุปกรณ์ เครือข่าย ที่เรียกว่า อีเธอร์เน็ตสวิตช์ (Ethernet Switch) ตามลำดับชั้นและเชื่อมต่อกับเครือข่ายอินเทอร์เน็ต แสดงรายละเอียดของ Ethernet การตั้งค่า (Configuration) และอื่นๆ เพื่อให้บอร์ดทำหน้าที่เป็นเซิร์ฟเวอร์ (Server) เช่น เว็บเซิร์ฟเวอร์ (Web Server) FTP เซิร์ฟเวอร์ เป็นต้น เพื่อรองรับการทำงานร่วมกับอุปกรณ์ IoT จากการเชื่อมต่อกับเซิร์ฟเวอร์ต่างๆ รายละเอียดเพิ่มเติมอยู่ในกราฟิกด้านล่าง | หัวข้อที่ 1.4



รูปที่ 6.13: การเข้าปลายสาย RJ45 ทั้งสองด้านสำหรับการเชื่อมต่อระหว่างเครื่องคอมพิวเตอร์และอุปกรณ์สวิตช์ (Switch) ตามมาตรฐาน TIA T568B ที่มา: [blogspot.com](http://blogspot.com)

การเชื่อมต่อเครือข่ายท้องถิ่น (Local Area Network) แบบอีเธอร์เน็ตในรูปที่ 6.13 สามารถเชื่อมต่อเครื่อง คอมพิวเตอร์จำนวนหลายเครื่องบนเครือข่ายเดียวกัน โดยอาศัยหลักการ CSMA/CD (Carrier Sense Multiple

Access/Collision Detection) รายละเอียดเพิ่มเติม ผู้อ่านสามารถศึกษาเพิ่มเติมได้ที่ลิงก์ต่อไปนี้ [wikipedia](#) หรือในรายวิชาอื่นๆ เช่น การสื่อสารข้อมูล (Data Communication) เครือข่ายคอมพิวเตอร์ (Computer Network) เป็นต้น

## 6.8 สัญญาณ WiFi และ Bluetooth สำหรับการสื่อสารไร้สาย



รูปที่ 6.14: รูปถ่ายด้านล่างของบอร์ด Pi3 และรูปขยายของชิป BCM43438 สำหรับเชื่อมต่อเครือข่ายไร้สาย WiFi และเครือข่ายไร้สายบลูทูธ (Bluetooth) ที่มา: [stackexchange.com](https://stackexchange.com)

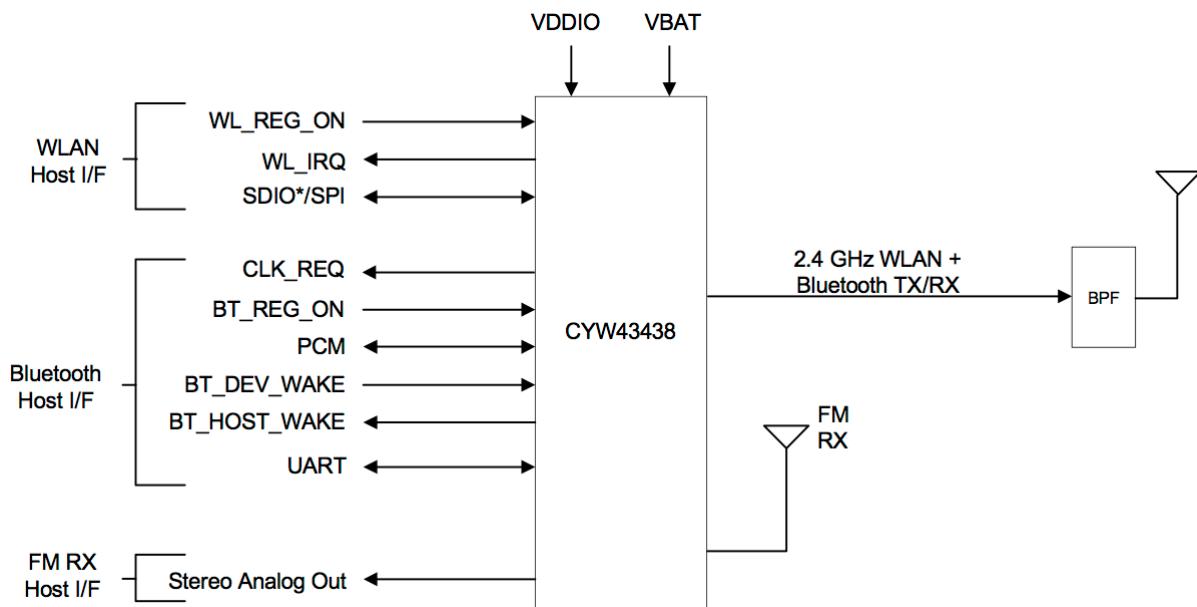
นอกเหนือจากการเชื่อมต่อแบบใช้สายแล้ว บอร์ด Pi3/Pi4 ได้ถูกออกแบบให้ทันสมัย และรองรับการเชื่อมต่อแบบไร้สายถึงสองชนิด คือ

- เครือข่ายไร้สาย WiFi สำหรับเชื่อมต่อกับเครือข่ายอินเทอร์เน็ตแบบไร้สาย และ
- Bluetooth สำหรับเชื่อมต่อกับอุปกรณ์ระยะสั้นแบบไร้สาย เช่น โทรศัพท์เคลื่อนที่ คลี๊ก เป็นต้น

วงจรสำหรับการเชื่อมต่อแบบไร้สายทั้งสองรวมอยู่ในชิป BCM43438 บนบอร์ด RPi3/Pi4 ในรูปที่ 6.14

WiFi เป็นเทคโนโลยีการเชื่อมต่ออินเทอร์เน็ตแบบไร้สาย สำหรับอุปกรณ์ต่างๆ เช่น คอมพิวเตอร์ส่วนบุคคล เครื่องเล่นเกมส์ โทรศัพท์สมาร์ตโฟน แท็บเล็ต กล้องดิจิตอลและเครื่องเสียงดิจิตอล โดยใช้คลื่นวิทยุที่ช่วงความถี่ 2.4 และ 5 กิกะเฮิรตซ์ อุปกรณ์ที่ไว้สามารถเชื่อมต่อกับอินเทอร์เน็ตได้ผ่านอุปกรณ์ที่เรียกว่า WiFi Router แอคเซส พอยต์ (Access Point) หรือ ซอตสปอต (Hot Spot) และบริเวณที่ระยะทำการของแอคเซสพอยต์ครอบคลุมอยู่ ที่ประมาณ 20 ม. ในอาคาร ถ้าเป็นที่โล่งแจ้งระยะทำการจะเพิ่มขึ้น

ชิป BCM43438 ที่มา: [Cypress Semiconductor \(2017\)](https://www.cypress.com/) บนบอร์ด RPi3/Pi4 รองรับสัญญาณ IEEE 802.11b/g/n ที่ย่านความถี่คลื่นพาร์ท 2.4 กิกะเฮิรตซ์ เท่านั้น และสัญญาณ Bluetooth เวอร์ชัน 4.1 รวมถึงตัวรับสัญญาณวิทยุ FM ที่มา: [Cypress Semiconductor \(2017\)](https://www.cypress.com/) บล็อกโดยแกรมของชิป BCM43438 ประกอบด้วยขาสัญญาณเชื่อมต่อเสาอากาศ และขาเชื่อมต่อกับไมโครคอนโทรลเลอร์ (Host Interface) ทั้งสองสัญญาณ ใช้สายอากาศ (Antenna) และความถี่พาร์ท (Carrier Frequency) ในย่านความถี่ 2.4 กิกะเฮิรตซ์เดียวกัน บลูทูธ ใช้หลักการ Frequency Hopping และกำลังส่งที่ต่ำกว่าสัญญาณ WiFi ทำให้มีเกิดการรบกวนกัน



รูปที่ 6.15: บล็อกไซโอดอกแกรมภายในชิป BCM43438 ประกอบด้วยขาสัญญาณเชื่อมต่อเสาอากาศ และขาเชื่อมต่อกับไมโครคอนโทรลเลอร์ ที่มา: [Cypress Semiconductor \(2017\)](#)

บลูทูธเชื่อมต่อ กับอุปกรณ์รอบๆ ตัว เรียกว่า Personal Area Network (PAN). การเชื่อมต่อของ Bluetooth จะเป็นแบบ Master Slave โดย Master 1 ตัวจะสามารถรับ Slave ได้มากถึง 7 ตัว เรียกว่า Piconet เช่น โทรศัพท์สมาร์ตโฟน เป็น Master เชื่อมต่อกับ หูฟัง เป็น Slave หากมี Slave หลายตัวต่อ เชื่อมพร้อมกัน Master จะสื่อสารกับ Slave เหล่านั้นแบบ Round Robin นอกจากนี้ Master สามารถ Broadcast ข้อมูลไปยัง Slave ทุกตัวได้ เช่นกัน

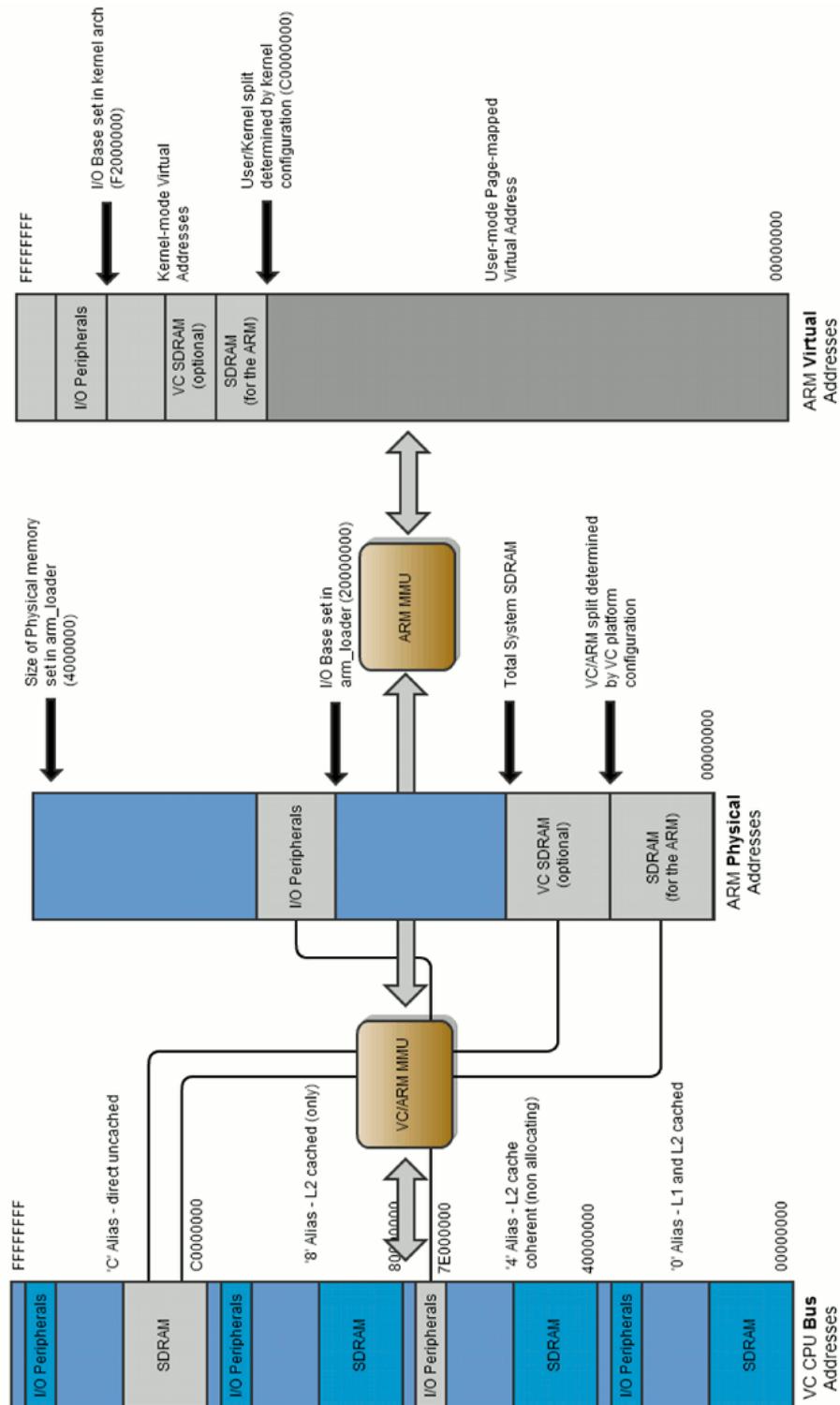
Bluetooth จะใช้ความถี่ความถี่คลื่นพาห์ ในย่าน 2.4 กิกะ赫ertz จะใช้ช่วง 2.400 ถึง 2.4835 กิกะ赫ertz และแบ่งออกเป็น 79 ช่องสัญญาณ และจะใช้ช่องสัญญาณที่แบ่งนี้ เพื่อส่งข้อมูลสลับช่องไปมา (Frequency Hopping) 1,600 ครั้งต่อ 1 วินาที (1600 Hops/Sec) ระยะทำการของ Bluetooth ประมาณ 5-100 เมตร เป็นการป้องกันการดักจับสัญญาณระหว่างสื่อสาร โดยระบบจะสลับช่องสัญญาณไปมา ผู้อ่านสามารถค้นคว้ารายละเอียดเพิ่มเติมที่ [bluetooth.com](http://bluetooth.com)

บลูทูธได้รับการออกแบบมาเพื่อใช้กับอุปกรณ์ที่มีขนาดเล็ก หรือสามารถเคลื่อนย้ายได้ง่าย เช่น โทรศัพท์เคลื่อนที่ แท็บเล็ต หูฟัง ลำโพง คลี๊ก รถยนต์ เป็นต้น เพื่อแอปพลิเคชันต่างๆ สามารถเชื่อมต่อกัน รับส่งหรือถ่ายโอนไฟล์ภาพ, เสียง, ข้อมูล โดยการใช้งานบลูทูธจะต้องมีการ Pair up หรือจับคู่ เป็นกลไกรักษาความปลอดภัย โดยผู้ใช้อุปกรณ์บลูทูธทั้งสองฝ่ายจะต้องป้อนรหัสเดียวกันก่อนการเชื่อมต่อ โดยอาศัยโปรไฟล์ (Profile) สำคัญที่มีในอุปกรณ์ทั้งสอง เช่น

- Human Interface Device Profile (HID) สำหรับเชื่อมต่ออุปกรณ์ เช่น เม้าส์ คีย์บอร์ด ของเครื่องคอมพิวเตอร์
- Dial-up Networking Profile (DUN) สำหรับใช้โน็ตบุ๊กหรือเครื่องคอมพิวเตอร์เชื่อมต่อกับอินเทอร์เน็ตผ่านโทรศัพท์เคลื่อนที่
- Advanced Audio Distribution Profile (A2DP) สำหรับเชื่อมต่อหูฟังชนิดบลูทูธ

รายละเอียดเพิ่มเติม ผู้อ่านสามารถค้นคว้าได้ที่ [Wikipedia](https://en.wikipedia.org/wiki/Bluetooth) การทดลองเพื่อแสดงรายละเอียดของ WiFi และ Bluetooth ในการทดลองที่ 9 ภาคผนวกที่ 1

## 6.9 หลักการ Memory Mapped Input/Output



รูปที่ 6.16: การเชื่อมโยงระหว่าง เวอร์ชัล แอดเดรส (ขวา) แอดเดรสภายนอก (กลาง) และ VC/CPU บัส แอดเดรส (VC/CPU Bus Address) ที่มา: [Broadcom \(2012\)](#) หมายเหตุ VC: VideoCore

ภายใต้ระบบปฏิบัติการลินุกซ์ โปรแกรมต่างๆ มีเวอร์ชัลเม莫รีของตนเอง แต่โปรแกรมเหล่านี้จะไม่สามารถเข้าถึงแอดเดรสภายนอกโดยตรงได้เลย โดยเฉพาะอุปกรณ์อินพุต/เอาต์พุต จะเข้าถึงได้ผ่าน System Call Interface ของระบบปฏิบัติการเท่านั้น เช่น พิงก์ชัน printf ในไลบรารี glibc เพื่อแสดงผลข้อมูลต่างๆ บนหน้า

จะแสดงผล รายละเอียดเพิ่มเติมในการทดลองที่ 7 ภาคผนวก G ในเชิงโครงสร้างระบบเวอร์ชัลเมโมรี เชื่อมโยง กับอินพุต/เอาต์พุต ผ่านทาง MMU (Memory Management Unit) ตัวที่ 1 (ARM MMU) จะแปลเวอร์ชัล แอดเดรสให้กลายเป็นแอดเดรสสากยภาพ ตามรูปที่ 5.4 และ MMU ตัวที่ 2 (VC/ARM MMU) จะแปลแอดเดรส กายภาพให้กลายเป็นบัสแอดเดรสที่เชื่อมซีพียูกับหน่วยความจำและอุปกรณ์อินพุต/เอาต์พุตต่างๆ ตามรูปที่ 6.16

ตารางที่ 6.4: ตารางเชื่อมโยงระหว่าง VC/CPU บัสแอดเดรส อุปกรณ์อินพุต/เอาต์พุต (I/O Peripherals) และ หมายเลขหัวข้อของตำราเล่มนี้ เริ่มต้นที่หมายเลข 0x7E00 0000 หมายเหตุ Rx: Receiver, Tx: Transmitter, UART: Universal Asynchronous/Synchronous Receiver Transmitter ที่มา: [Broadcom \(2012\)](#)

VC/CPU บัสแอดเดรส	ชื่อ (Name)	รายละเอียด (Details)	หัวข้อ ในตำรา
0x7E00 0000	...	...	
0x7E00 1000	...	...	
0x7E00 2000	...	...	
0x7E00 3000	System Timer	วงจรจับเวลาสำหรับระบบปฏิบัติการ	-
0x7E00 7000	DMA Controller	การเข้าถึงหน่วยความจำโดยตรง	<a href="#">6.13</a>
0x7E00 B000	Interrupt Controller	วงจรควบคุมอินเทอร์รัปท์	<a href="#">6.12</a>
0x7E00 B400	Timer	วงจรจับเวลาสำหรับการใช้งานทั่วไป	-
0x7E20 0000	General Purpose I/O	ขา GPIO ทั้งหมด	<a href="#">6.11</a>
0x7E20 1000	Universal Async. Rx Tx	การสื่อสารแบบอนุกรม	-
0x7E20 3000	Pulse Code Modulation	สัญญาณเสียง	<a href="#">6.4</a>
0x7E20 4000	SPI0	Serial Peripheral Interface 0	-
0x7E20 5000	Serial Controller (I <sup>2</sup> C)	สัญญาณ I <sup>2</sup> C	-
0x7E21 4000	SPI/BSC Slave	Serial Peripheral Interface/ Serial Controller (I <sup>2</sup> C)	-
0x7E21 5000	mini UART, SPI1, SPI2	การสื่อสารแบบอนุกรม	-
0x7E30 0000	External Mass Media Controller	วงจรควบคุม อุปกรณ์เก็บรักษาข้อมูล	<a href="#">7.3</a>
0x7E98 0000	Universal Serial Bus	USB	<a href="#">6.6</a>

ในเชิงของฮาร์ดแวร์ TLB และเพจテーブลภายในตัว ARM MMU ทำหน้าที่แปลเวอร์ชัลแอดเดรสเป็นแอดเดรส กายภาพ หลังจากนั้น VC (VideoCore)/ARM MMU จึงถูกแปลงแอดเดรสสากยภาพเป็น VC/CPU บัสแอดเดรส ซึ่ง เชื่อมต่อกับซีพียู วิดีโโคลร์ อุปกรณ์อินพุต/เอาต์พุต และหน่วยความจำชนิด SDRAM โดยในรูปที่ 6.16 แอดเดรส กายภาพ 0x20000000 หรือ 0x2000 0000 เป็นค่าตัวแปร I/O Base ในไฟล์ ARM Loader จะแปลเป็น VC/ CPU บัสแอดเดรสหมายเลข 0x7E000000 หรือ 0x7E00 0000

ภายในชิป BCM283x มีวงจรหรืออุปกรณ์/อินพุต/เอาต์พุต (I/O Peripherals) จำนวนมาก การเชื่อมต่อกับ อุปกรณ์เหล่านี้จะใช้การตั้งค่า VC/CPU บัสแอดเดรส เพื่ออ่านข้อมูลและเขียนข้อมูลเหล่านี้ แล้วแปล VC/CPU บัสแอดเดรสเหล่านี้กับ แอดเดรสสากยภาพ ในรูปที่ 6.16 การอ่านหรือเขียนข้อมูลไปยังแอดเดรสสากยภาพเหล่านี้ทำได้การใช้คำสั่ง LDR และ STR ที่เวอร์ชัลแอดเดรสเพื่อให้ ARM MMU แปลเป็น แอดเดรสสากยภาพ และ VC/CPU MMU แปลเป็น VC/CPU บัสแอดเดรสที่ถูกต้อง โดยผู้ผลิตฮาร์ดแวร์ได้กำหนดหมายเลข VC/CPU บัสแอดเดรสสำหรับอุปกรณ์อินพุต/เอาต์พุต (I/O Peripherals) ตามตารางที่ 6.4 ต่อไปนี้ โดย VC/CPU บัส แอดเดรสเริ่มต้นที่หมายเลข 0x7E000000 หรือ 0x7E00 0000

นอกจาก I/O Peripherals แล้ว หน่วยความจำ SDRAM ณ แอดเดรสสากยภาพ 0x0000 0000 ถูก-map เป็น

บัสแอดเดรสที่ตำแหน่ง 0xC000 0000 ในพื้นที่สีเทา โดยจะมีตัวแปร VC/ARM Split เป็นตัวกำหนดขนาดหน่วยความจำที่จะแบ่งไว้ให้กับจีพียูซึ่งจะได้ทดลองเปลี่ยนแปลงค่าในการทดลองที่ 9 ภาคผนวก |

## 6.10 หัวเชื่อมต่อ 40 ขา (40-Pin Header)

เนื่องจากบอร์ดตระกูล Raspberry Pi นี้ถูกออกแบบให้มีราคาอยู่ในระดับกลาง เนื่องจากบอร์ดต้องมีชิป BCM283x มาให้ผู้อ่านได้เรียนรู้ ทางหัวเชื่อมต่อจำนวน 40 ขาในรูปที่ 6.1 ซึ่งประกอบด้วย ส่วนต่างๆ เหล่านี้

- ขา GPIO (General Purpose input/output) ได้แก่ ขา GPIO00-GPIO27 สำหรับใช้เชื่อมต่อกับวงจรที่ต้องการทดลอง
- ขาสำหรับการสื่อสารชนิด UART ได้แก่ ขา RXD0/TXD0, ขา RXD1/TXD1 สำหรับเชื่อมต่อกับบอร์ดอื่นๆ ผ่านทางพอร์ต UART ซึ่งมีอัตราบิตเรตสูงสุด 115,000 บิตต่อวินาที
- ขาสำหรับการเชื่อมต่อกับชิปภายนอกชนิด SPI ได้แก่ ขา SPI1\_CE0, SPI1\_CE1, SPI1\_CE2, SPI1\_MISO, SPI1\_MOSI, SPI1\_SCLK สำหรับเชื่อมต่อกับบอร์ดอื่นๆ ผ่านทางพอร์ต SPI ซึ่งมีอัตราบิตเรตสูงกว่า 115,000 บิตต่อวินาที
- ขาสำหรับการเชื่อมต่อสัญญาณเสียง ได้แก่ ขา PCM\_DIN, PCM\_DOUT, PCM\_FS, PCM\_CLK ตามมาตรฐาน I<sup>2</sup>S ในหัวข้อที่ 6.4
- ขาสำหรับการเชื่อมต่อกับอุปกรณ์ภายนอกชนิด PWM และ PPM ได้แก่ ขา PWM0, PWM1 สำหรับการทดลอง เช่น การควบคุมการหมุนของมอเตอร์ไฟฟ้า การควบคุมความสว่างของหลอดไฟ เป็นต้น
- ขาสำหรับการเชื่อมต่อกับชิปภายนอกตามมาตรฐาน I<sup>2</sup>C ได้แก่ ขา SDA0, SCL0, SDA1, SCL1
- ขาสำหรับการเชื่อมต่อกับการ์ดหน่วยความจำ SD ได้แก่ ขา SD0\_CMD, SD0\_CLK, SD0\_DAT0, SD0\_DAT0-SD0\_DAT3
- ไฟกระเจ粲และแสดงความต่างศักย์ 3.3 โวลต์, 5.0 โวลต์ สำหรับจ่ายวงจรภายนอกที่กินกระแสอย่างมาก

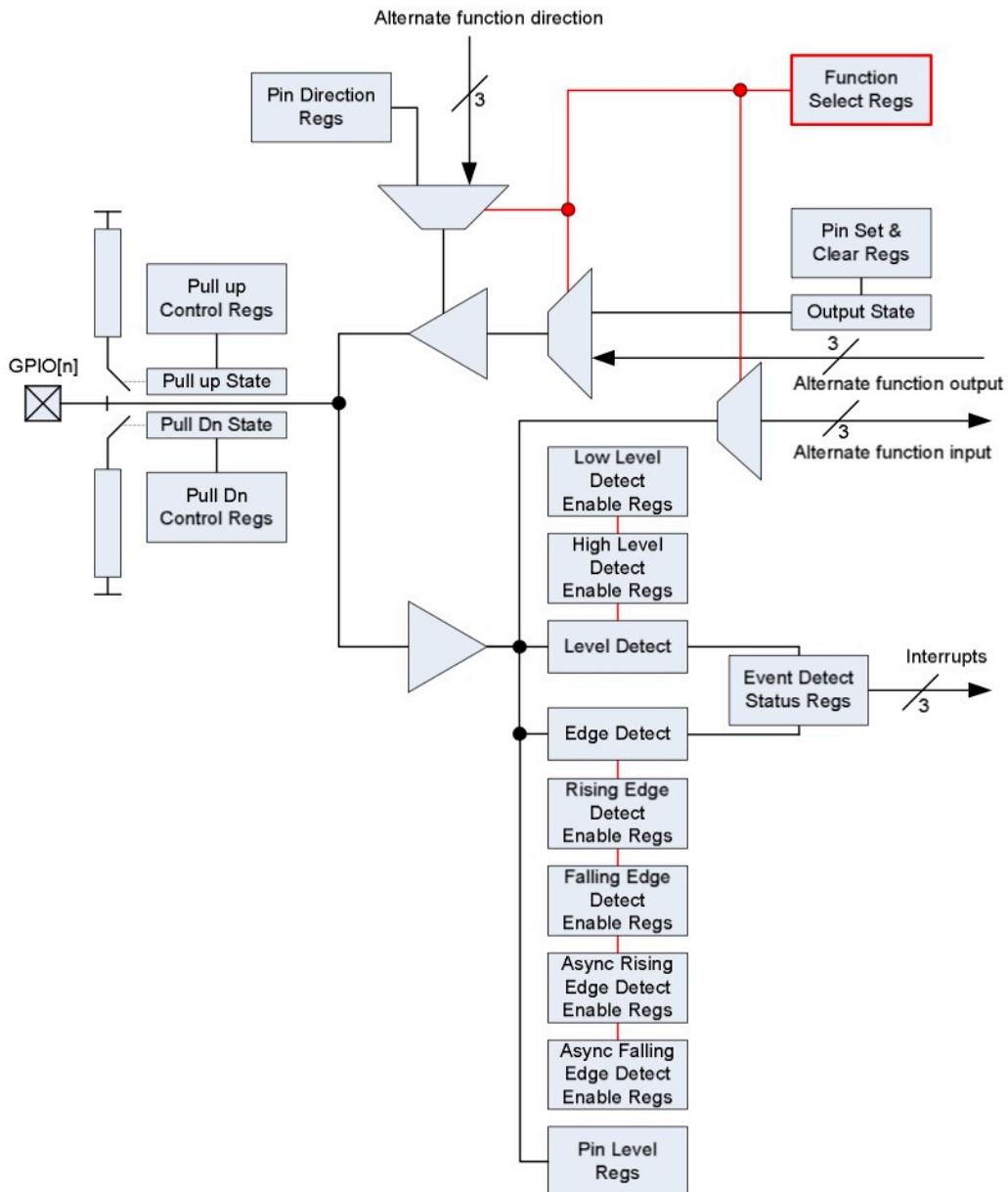
ขาสัญญาณเหล่านี้สามารถเขียนโปรแกรมควบคุมสั่งการด้วยภาษาแอลกอริทึม C และ Python ในการทดลองต่างๆ

เนื่องจากชิป BCM2837 มีฟุตพรินท์ (Foot Print) ขนาดเล็กทำให้จำนวนขา (Leads) จำกัด เพียง 200 ขา ที่มา: [Raspberry Pi \(Trading\) \(2019\)](#) จึงต้องใช้การมัลติเพล็กซ์ (Multiplex) หรือเลือก (Select) สัญญาณที่ต้องการเพียงสัญญาณเดียวมาที่ขา โดยโปรแกรมเมอร์จะต้องกำหนดว่าขาหมายเลขใดมีมัลติเพล็กซ์เป็นสัญญาณอะไร ทำให้ขาเดียวกันมีสัญญาณตัวเลือก (Alternate function) หลายสัญญาณ ตามที่ได้สรุปในตารางที่ 6.5

ตารางที่ 6.5: หมายเลขขา ชื่อขา และตัวเลือก (ชื่อสัญญาณ) ที่ 0-5 ของหัวเชื่อมต่อสายห้อง 40 ขา (GND: Ground) ที่มา: [Broadcom \(2012\)](#), ที่มา: [Raspberry Pi \(Trading\) \(2019\)](#)

ขา	ชื่อ	ตัวเลือก0	ตัวเลือก1	ตัวเลือก2	ตัวเลือก3	ตัวเลือก4
1	3.3V					
2	5.0V					
3	GPIO02	SDA1	SA3	LCD_VSYNC		
4	5.0V					
5	GPIO03	SCL1	SA2	LCD_HSYNC		
6	GND					
7	GPIO04	GPCLK0	SA1	DPI_D0		
8	GPIO14	TxD0	SD6	DPI_D10		
9	GND					
10	GPIO15	RxD0	SD7	DPI_D11		
11	GPIO17	FL1	SD9	DPI_D13	RTS0	SPI1_CE1_N
12	GPIO18	PCM_CLK	SD10	DPI_D14		SPI1_CE0_N
13	GPIO27	SD0_DAT3	TE1	DPI_D23	SD1_DAT3	ARM_TMS
14	GND					
15	GPIO22	SD0_CLK	SD14	DPI_D18	SD1_CLK	ARM_TRST
16	GPIO23	SD0_CMD	SD15	DPI_D19	SD1_CMD	ARM_RTCK
17	3.3V					
18	GPIO24	SD0_DAT0	SD16	DPI_D20	SD0_DAT0	ARM_TDO
19	GPIO10	SPI0_MOSI	SD2	DPI_D6		
20	GND					
21	GPIO09	SPI0_MISO	SD1	DPI_D5		
22	GPIO25	SD0_DAT1	SD17	DPI_D21	SD0_DAT1	ARM_TCK
23	GPIO11	SPI0_CLK	SD3	DPI_D7		
24	GPIO08	SPI0_CE0_N	SD0	DPI_D4		
25	GND					
26	GPIO07	SPI0_CE1_N	SWE_N	DPI_D3		
27	GPIO00	SDA0	SA5			
28	GPIO01	SCL0	SA4			
29	GPIO05	GPCLK1	SA0	DPI_D0		
30	GND					
31	GPIO06	GPCLK2	SOE_N	DPI_D2		
32	GPIO12	PWM0	SD4	DPI_D8		
33	GPIO13	PWM1	SD5	DPI_D9		
34	GND					
35	GPIO19	PCM_FS	SD11	DPI_D15		SPI1_MISO
36	GPIO16	FL0	SD8	DPI_D12	CTS0	SPI1_CE2_N
37	GPIO26	SD0_DAT2	TE0	DPI_D22	SD1_DAT2	ARM_TDI
38	GPIO20	PCM_DIN	SD12	DPI_D16		SPI1_MOSI
39	GND					
40	GPIO21	PCM_DOUT	SD13	DPI_D17		SPI1_SCLK

## 6.11 ขา GPIO (General Purpose Input Output)



รูปที่ 6.17: โครงสร้างภายในขา GPIO สำหรับชิปะระกุลเดียว กับ BCM2835 ที่มา: [Broadcom \(2012\)](#)

ขา GPIO ทุกขา มีโครงสร้างภายในเหมือนกันหมดตามรูปที่ 6.17 ขา  $\text{GPIO}[n]$  คือ ขาที่  $n = 0$  ถึง  $n = 53$  ทั้งหมด 54 ขาสำหรับชิปะระกุล BCM283x ผู้อ่านสามารถทำความเข้าใจง่ายๆ ซึ่งประกอบด้วย

- รีจิสเตอร์ **Function Select** ทำหน้าที่เลือกสัญญาณที่ต้องการ ได้แก่ สัญญาณ GPIO ขาเข้า ขาออก และ สัญญาณตัวเลือก (Alternate function) 0 - สัญญาณเลือก 5 ตามตารางที่ 6.5 ภายใต้การควบคุมของ รีจิสเตอร์นี้ ด้วยการใช้ตัวเลขจำนวน 3 บิตเพื่อเลือกการทำงานของ GPIO แต่ละชานั้น ใช้วิธีการกำหนด ดังนี้

- $000_2$  = GPIO อินพุต (Input)
- $001_2$  = GPIO เอาต์พุต (Output)

- $100_2$  = ตัวเลือก 0 ในตารางที่ 6.5
- $101_2$  = ตัวเลือก 1 ในตารางที่ 6.5
- $110_2$  = ตัวเลือก 2 ในตารางที่ 6.5
- $111_2$  = ตัวเลือก 3 ในตารางที่ 6.5
- $011_2$  = ตัวเลือก 4 ในตารางที่ 6.5
- $010_2$  = ตัวเลือก 5 ในตารางที่ 6.5

- รีจิสเตอร์ Pin Direction ใช้กำหนดทิศทางของสัญญาณเป็นเข้า/ออก (Direction = Input) หรือเป็นขาออก (Direction = Output)
- วงจรเอาต์พุต (Output) ซึ่งอยู่ด้านบนของรูป ประกอบด้วย
  - รีจิสเตอร์ Pin Set และรีจิสเตอร์ Pin Clear ซึ่งเก็บค่าของเอาต์พุตไว้ โปรแกรมเมอร์สามารถตั้งค่ารีจิสเตอร์ตัวนี้ให้มีค่าลอจิก 1 สัญลักษณ์สีเหลืองค้างหมุน แทนอุปกรณ์มัลติเพล็กเซอร์ หน้าที่เลือกสัญญาณเอาต์พุตจากสัญญาโนินพุตจำนวนหลายเส้น สัญลักษณ์สามเหลี่ยมแทนวงจรบัฟเฟอร์ซึ่งลอจิก O/p = i/p แต่หน้าที่เพิ่มกระแสให้กับสัญญาณเอาต์พุต
- วงจรอินพุต (Input) ซึ่งอยู่ด้านล่างของรูป ประกอบด้วย
  - วงจรตรวจจับระดับสัญญาณ (Level Detect) ประกอบด้วย รีจิสเตอร์ High/Low Level Enable เพื่อเปิด (Enable) การทำงานการตรวจจับเหตุการณ์โดยใช้ระดับสัญญาณแต่ละชนิด
  - วงจรตรวจจับขอบสัญญาณ (Edge Detect) ประกอบด้วย รีจิสเตอร์ Sync/Async Rising/Falling Edge Detect Enable เพื่อเปิด (Enable) การทำงานการตรวจจับเหตุการณ์ขอบสัญญาณแต่ละชนิด หากไม่ระบุว่าเป็น Asynchronous แสดงว่าเป็นชนิด Synchronous
  - รีจิสเตอร์สถานะตรวจจับเหตุการณ์ (Event Detect Status Register) เพื่อเก็บค่าลอจิกที่ได้จากการตรวจจับสัญญาณว่าตรวจจับเหตุการณ์ได้หรือไม่
  - รีจิสเตอร์เก็บค่าระดับสัญญาณ (Pin Level) ที่รับได้ โดยใช้การสุ่มตามสัญญาณคลือก
  - ตัวต้านทานจำนวน 2 ตัว ซึ่งทำหน้าที่พูลอปหรือพูลดาวน์ ผู้ใช้สามารถใช้งาน ตัวต้านทานภายในเพื่อพูลอป (Pull up) เชื่อมต่อกับไฟกระแสตรง (Vdd) 3.3 โวลต์ หรือ พูลดาวน์ (Pull Down) เชื่อมต่อขา GPIO กับกราวน์ (0 โวลต์) ด้วยการกำหนดค่าในรีจิสเตอร์ GPPUD ขนาด 2 บิต เพื่อตั้งค่าของรีจิสเตอร์ Pull Up Control และ รีจิสเตอร์ Pull Down Control

ผู้อ่านสามารถเรียนรู้การใช้งานขา GPIO เหล่านี้ในการทดลองที่ 10 และ 11 ตามลำดับ ตารางที่ 6.6 คือรายละเอียดของแอดเดรสหมายเลข 0x7E20 0000 ในตารางที่ 6.4

ตารางที่ 6.6: ตารางเข้ามายัง VC/CPU บัสแอดเดรสกับรีจิสเตอร์ต่างๆ ของวงจร GPIO เพื่อตั้งค่าและสั่งงานขา GPIO ทุกขา โดยเริ่มต้นที่หมายเลข 0x7E20 0000 ที่มา: [Broadcom \(2012\)](#)

บัสแอดเดรส	รีจิสเตอร์	รายละเอียด	บิต	R/W	ขา
0x7E20 0000	GPFSEL0	GPIO Function Select 0	32	R/W	0-9
0x7E20 0004	GPFSEL1	GPIO Function Select 1	32	R/W	10-19
0x7E20 0008	GPFSEL2	GPIO Function Select 2	32	R/W	20-29
0x7E20 000C	GPFSEL3	GPIO Function Select 3	32	R/W	30-39
0x7E20 0010	GPFSEL4	GPIO Function Select 4	32	R/W	40-49
0x7E20 0014	GPFSEL5	GPIO Function Select 5	12	R/W	50-53
0x7E20 001C	GPSET0	GPIO Pin Output Set 0	32	W	0-31
0x7E20 0020	GPSET1	GPIO Pin Output Set 1	22	W	32-53
0x7E20 0028	GPCLR0	GPIO Pin Output Clear 0	32	W	0-31
0x7E20 002C	GPCLR1	GPIO Pin Output Clear 1	22	W	32-53
0x7E20 0034	GPLEV0	GPIO Pin Level 0	32	R	0-31
0x7E20 0038	GPLEV1	GPIO Pin Level 1	22	R	32-53
0x7E20 0040	GPEDS0	GPIO Pin Event Detect Status 0	32	R/W	0-31
0x7E20 0044	GPEDS1	GPIO Pin Event Detect Status 1	22	R/W	32-53
0x7E20 004C	GPREN0	GPIO Pin Rising Edge Detect Enable 0	32	R/W	0-31
0x7E20 0050	GPREN1	GPIO Pin Rising Edge Detect Enable 1	22	R/W	32-53
0x7E20 0058	GPFEN0	GPIO Pin Falling Edge Detect Enable 0	32	R/W	0-31
0x7E20 005C	GPFEN1	GPIO Pin Falling Edge Detect Enable 1	22	R/W	32-53
0x7E20 0064	GPHEN0	GPIO Pin High Detect Enable 0	32	R/W	0-31
0x7E20 0068	GPHEN1	GPIO Pin High Detect Enable 1	22	R/W	32-53
0x7E20 0070	GPLENO	GPIO Pin Low Detect Enable 0	32	R/W	0-31
0x7E20 0074	GPLEN1	GPIO Pin Low Detect Enable 1	22	R/W	32-53
0x7E20 007C	GPAREN0	GPIO Pin Async. Rising Edge Detect 0	32	R/W	0-31
0x7E20 0080	GPAREN1	GPIO Pin Async. Rising Edge Detect 1	22	R/W	32-53
0x7E20 0088	GPAFEN0	GPIO Pin Async. Falling Edge Detect 0	32	R/W	0-31
0x7E20 008C	GPAFEN1	GPIO Pin Async. Falling Edge Detect 1	22	R/W	32-53
0x7E20 0094	GPPUD	GPIO Pin Pull-Up/Down Enable	2	R/W	-
0x7E20 0098	GPPUDLK0	GPIO Pin Pull-Up/Down Enable Clk0	32	R/W	0-31
0x7E20 009C	GPPUDLK1	GPIO Pin Pull-Up/Down Enable Clk1	22	R/W	32-53

- รีจิสเตอร์ **GPFSEL0** (GPIO Function Select 0) ถึง **GPFSEL5** (GPIO Function Select 5) ควบคุมขาที่ 0 - 53 ทั้งหมด 54 ขา โดยมีรายละเอียดดังนี้

- การเขียนข้อมูลที่แอดเดรส 0x7E20 0000 - 0x7E20 0003 รวม 4 ไบต์ หรือ 32 บิต เท่ากับเป็นการตั้งค่าของรีจิสเตอร์ **GPFSEL0** เพื่อควบคุมการทำงานของขาที่ 0 - ขาที่ 9 ทั้งหมด 10 ขาฯ ละ 3 บิตเท่ากับใช้ข้อมูลเพียง 30 บิต โดยขาที่ 0 ใช้งานบิตที่ 0-2 ตามลำดับจนถึงขาที่ 9 ใช้งานบิตที่ 27-29
- การเขียนข้อมูลที่แอดเดรส 0x7E20 0004 - 0x7E20 0007 รวม 4 ไบต์ หรือ 32 บิต เท่ากับเป็นการตั้งค่าของรีจิสเตอร์ **GPFSEL1** เพื่อควบคุมการทำงานของขาที่ 10 - ขาที่ 19 ทั้งหมด 10 ขาฯ ละ 3 บิตเท่ากับใช้ข้อมูลเพียง 30 บิต โดยขาที่ 10 ใช้งานบิตที่ 0-2 ตามลำดับจนถึงขาที่ 19 ใช้งานบิตที่ 27-29
- ...

- การเขียนข้อมูลที่แอดเดรส 0x7E20 0014 - 0x7E20 0015 รวม 2 ไบต์ หรือ 16 บิต เท่ากับเป็นการตั้งค่าของรีจิสเตอร์ **GPFSEL5** เพื่อควบคุมการทำงานของขาที่ 50 - ขาที่ 53 ทั้งหมด 4 ขา ละ 3 บิตเท่ากับใช้ข้อมูลเพียง 12 บิต โดยขาที่ 50 ใช้งานบิตที่ 0-2 ตามลำดับจนถึงขาที่ 53 ใช้งานบิตที่ 9 - 11
- การตั้งขาเป็นเอาต์พุต ( $GPFSEL=001_2$ )
  - รีจิสเตอร์ **GPSET0** (GPIO Pin Output Set0) และ **GPSET1** ใช้สำหรับตั้งค่าเอาต์พุตเป็นลอจิก 1
    - \* แอดเดรส 0x7E20 001C - 0x7E20 001F จำนวน 4 ไบต์ หรือ 32 บิต ตรงกับรีจิสเตอร์ **GPSET0** ควบคุมขาที่ 0 - 31 จำนวน 32 ขา ละ 1 บิต ขาที่ 0 ควบคุมโดยบิตที่ 0 จนถึงขาที่ 31 ควบคุมโดยบิตที่ 31 ตามลำดับ
    - \* แอดเดรส 0x7E20 0020 - 0x7E20 0022 จำนวน 3 ไบต์ หรือ 24 บิต ตรงกับรีจิสเตอร์ **GPSET1** ควบคุมขาที่ 32 - 53 จำนวน 22 ขา ละ 1 บิต ขาที่ 32 ควบคุมโดยบิตที่ 0 จนถึงขาที่ 53 ควบคุมโดยบิตที่ 21 ตามลำดับ
  - รีจิสเตอร์ **GPCLR0** (GPIO Pin Output Clear) และ **GPCLR1** ใช้สำหรับตั้งค่าเอาต์พุตเป็นลอจิก 0
    - \* แอดเดรส 0x7E20 0028 - 0x7E20 002B จำนวน 4 ไบต์ หรือ 32 บิต ตรงกับรีจิสเตอร์ **GPCLR0** ควบคุมขาที่ 0 - 31 จำนวน 32 ขา ละ 1 บิต ขาที่ 0 ควบคุมโดยบิตที่ 0 จนถึงขาที่ 31 ควบคุมโดยบิตที่ 31 ตามลำดับ
    - \* แอดเดรส 0x7E20 002C - 0x7E20 002e จำนวน 3 ไบต์ หรือ 24 บิต ตรงกับรีจิสเตอร์ **GPCLR1** ควบคุมขาที่ 32 - 53 จำนวน 22 ขา ละ 1 บิต ขาที่ 32 ควบคุมโดยบิตที่ 0 จนถึงขาที่ 53 ควบคุมโดยบิตที่ 21 ตามลำดับ
- การตั้งขาเป็นขาอินพุต ( $GPFSEL=000_2$ ) จะมีความซับซ้อนมากกว่าเป็นขาเอาต์พุต โดยโปรแกรมเมอร์จะต้องตั้งค่ารีจิสเตอร์ว่าเป็น ชนิดตรวจจับระดับสัญญาณ (Level Detect) หรือ ชนิดตรวจจับขอบ (Edge Detect)
  - การตรวจจับระดับสัญญาณ (Level Detect) แบ่งเป็น ลอจิก 0 (Low Level) และ ลอจิก 1 (High Level) โดยการตั้งค่าในรีจิสเตอร์ **GPLEN** (GPIO Pin Low Detect Enable) และรีจิสเตอร์ **GPHEN** (GPIO Pin High Detect Enable) ตัวได้ตัวหนึ่งให้เท่ากับลอจิก 1 ที่เหลือเป็น 0 ค่าอินพุตที่รับเข้ามาจะบันทึกในรีจิสเตอร์ **GPLEV** (Pin Level)
  - การตรวจจับขอบสัญญาณ (Edge Detection) แบ่งเป็น
    - \* ขอบขาขึ้น (Rising) และ ขอบขาลง (Falling) เมื่อตรวจจับได้ จะส่งสัญญาโนินเทอร์รัปท์เข้าสู่ซีพียูต่อไป
    - \* การตรวจจับขอบสัญญาณแบ่งเป็น ชนิดซิงโครนัส (Synchronous) และ อะซิงโครนัส (Asynchronous) ความหมายคือ ซิงโครไนซ์ (Synchronize) กับสัญญาณคล็อกหรือไม่
      - ชนิดซิงโครนัส การตรวจจับจะขึ้นกับความถี่ของสัญญาณคล็อก เหมาะกับเหตุการณ์ที่เกิดขึ้นบ่อย
      - ชนิดอะซิงโครนัส การตรวจจับจะไม่ขึ้นกับความถี่ของสัญญาณคล็อก 때문에กับเหตุการณ์ที่นานๆ จะเกิดขึ้น

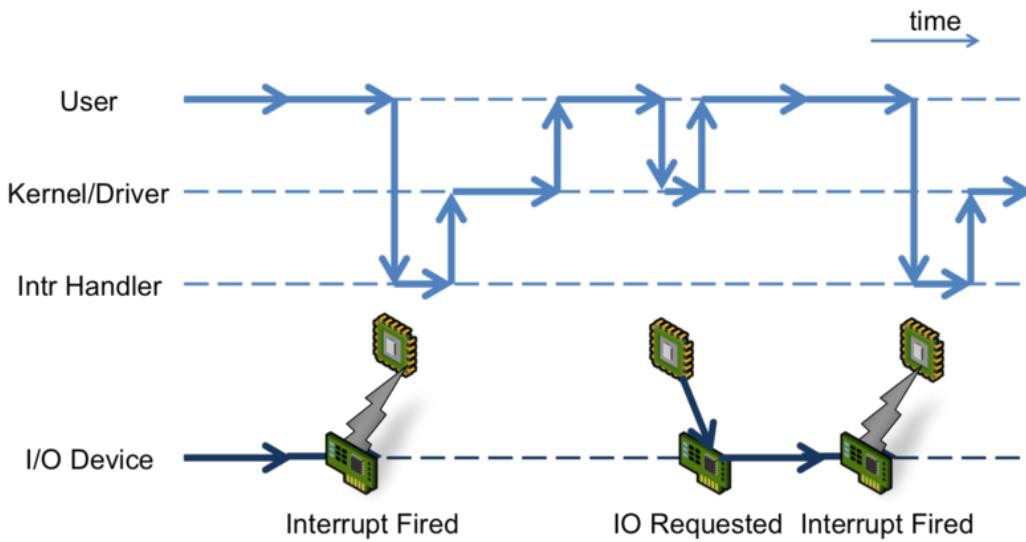
โดยการควบคุมที่รีจิสเตอร์ทั้ง 4 ชนิดนี้เกิดเป็นสัญญาณอินเทอร์รัปท์ร้องขอไปยังซีพียู ได้แก่

- รีจิสเตอร์ **GPREN** (GPIO Synchronous Rising Edge Enable) และ
  - รีจิสเตอร์ **GPFEN** (GPIO Synchronous Falling Edge Enable)
  - รีจิสเตอร์ **GPAREN** (GPIO Asynchronous Rising Edge Enable) และ
  - รีจิสเตอร์ **GPAFEN** (GPIO Asynchronous Falling Edge Enable)
- รีจิสเตอร์ **GPEDS** (Pin Event Detect Status) จะเก็บค่าล็อกิค 1 เมื่อตรวจจับเหตุการณ์ที่รับเข้ามาได้ตามเงื่อนไขที่ต้องการ และจะถูกเปลี่ยนเป็นล็อกิค 0 เมื่อซีพียูตอบสนองต่อการร้องขอ เรียบร้อยแล้ว รายละเอียดเพิ่มเติมในหัวข้อที่ [6.12](#)

ชิป BCM2837/BCM2711 เป็นล็อกิคชนิด CMOS (Complementary Metal Oxide Semiconductor) ล็อกิค 1 ของขาทั้งหมดใช้ความต่างศักย์ 3.3 โวลต์เท่านั้น ในขณะที่ไฟกระแสตรง 5.0 โวลต์ใช้สำหรับจ่ายไฟให้กับซิปอินๆ บางตัวที่เป็นล็อกิคชนิดอื่น นักพัฒนาจะต้องระวังการเชื่อมต่อวงจรทั้งสองชนิดเข้าด้วยกัน การใช้งานในลักษณะ GPIO นักพัฒนาจะต้องเชื่อมต่อวงจรและเขียนโปรแกรมควบคุมทิศทางการไหลของสัญญาณของขา GPIO แต่ละขาว่า เป็น ขาอับสัญญาณอินพุต หรือ ขาปล่อยสัญญาณเอาต์พุต ตามวงจรที่ต่อไว้ให้ถูกต้อง มิเช่นนั้นอาจทำให้ชิป BCM2837 เสียหาย รวมถึงการต่อไฟกระแสตรง 3.3 และ 5.0 โวลต์กับกราวน์ (0 โวลต์) โดยไม่ได้ตั้งใจ

ผู้อ่านสามารถใช้งานขา GPIO เหล่านี้ใน การทดลองที่ 10 ภาคผนวก [J](#) และการทดลองที่ 11 ภาคผนวก [K](#) โดยพัฒนาโปรแกรมประยุกต์ด้วยภาษา C และแอดเซมบลีทำงานร่วมกับไลบรารี wiringPi และกิจกรรมทั้งการทดลองจะช่วยเสริมให้ผู้อ่านเข้าใจการตั้งค่ารีจิสเตอร์เหล่านี้ด้วย

## 6.12 หลักการอินเทอร์รัปท์ (Interrupt)



รูปที่ 6.18: ไดอะแกรมเวลาของโปรแกรมที่ต้องการเชื่อมต่อกับอุปกรณ์อินพุต/เอาต์พุตทำให้เกิดอินเทอร์รัปท์ เกิดขึ้นเป็นระยะๆ ที่มา: [virtualirfan.com](http://virtualirfan.com)

ไดอะแกรมเวลาของโปรแกรมที่ต้องการเชื่อมต่อกับอุปกรณ์อินพุต/เอาต์พุตในรูปที่ 6.18 ทำให้เกิดการขัดจังหวะการทำงานของซีพียูหรืออินเทอร์รัปท์เป็นระยะๆ โดยเรียงลำดับเวลาจากช้าสุดไปเร็วสุด ซีพียูจะเปลี่ยนโหมดการทำงานของซีพียู (โปรแกรม) จากโหมดยูสเซอร์ (User Mode) เป็นโหมดเครอร์เนล (Kernel Mode) เพื่อเปลี่ยนซีพียูไปรับคำสั่งของอินเทอร์รัปต์แฮนด์เลอร์ (Interrupt Handler) ซึ่งตั้งอยู่ที่แอดเดรสหรือเรียกว่า เวกเตอร์ (Vector Table) ของตาราง Interrupt Vector ในการทดลองที่ 9 หัวข้อที่ 1.3.2 ณ ตำแหน่ง Virtual kernel memory layout

บางตัวราเรียกอินเทอร์รัปต์แฮนด์เลอร์ว่า อินเทอร์รัปท์เซอร์วิสทรูทิน (Interrupt Service Routine) ย่อว่า ISR เมื่อซีพียูจัดการกับข้อมูลที่ได้รับแล้ว ซีพียูจะเปลี่ยนกลับไปทำงาน(โปรแกรม)ในโหมดยูสเซอร์ต่อ หากโปรแกรมมีความต้องการ จะเชื่อมต่อกับอุปกรณ์อินพุต/เอาต์พุต โปรแกรมจะต้องร้องขอให้เครอร์เนลสั่งงาน อุปกรณ์เหล่านั้นผ่านทางดีไวซ์ไดเรอร์ในรูป แล้วจึงกลับไปทำงาน(โปรแกรม)ในโหมดยูสเซอร์ต่อจนกว่าจะมีเหตุการณ์เกิดขึ้น สัญญาณร้องขออินเทอร์รัปท์จะขัดจังหวะการทำงานของซีพียูอีกรอบ เมื่อนั้นคำสั่งในอินเทอร์รัปต์แฮนด์เลอร์จะเป็นผู้จัดการขบวนการรับส่งข้อมูลแทนโปรแกรมแอปพลิเคชัน เพื่อให้ซีพียูไม่เสียเวลาเรื่อง latency การทำงานของวงจร IO มากจนทำให้ซีพียูไม่มีโอกาสทำงานโปรแกรมอื่นๆ

สัญญาณร้องขออินเทอร์รัปท์ (Interrupt Request) คือ สัญญาณที่เกิดขึ้นจากอุปกรณ์อินพุต/เอาต์พุต จากจีพียูและจากเหตุการณ์พิเศษ สัญญาณเหล่านี้จะทำให้ CPU หยุดพักโปรแกรมที่กำลังรันอยู่เป็นการชั่วคราว และเปลี่ยนไปรับคำสั่งในอินเทอร์รัปต์แฮนด์เลอร์ หรือ อินเทอร์รัปท์เซอร์วิสทรูทิน เพื่อตอบสนอง (Respond) หรือให้บริการ (Service) ต่อเหตุการณ์ที่เกิดขึ้น การตอบสนองหรือการบริการ เพื่อกำหนดขั้นตอนการทำงานของซีพียูในการตอบสนองการอินเทอร์รัปท์

รายละเอียดการทำงานของอินเทอร์รัปท์ในกรณีศึกษาซีพียู ARM มีขั้นตอนดังนี้

1. เกิดเหตุการณ์อินเทอร์รัปท์ (Interrupt Event): เหตุการณ์แบ่งเป็น 2 ชนิด คือ

- อินเทอร์รัปท์จากการทำงานของชาร์ดแวร์ (Hardware interrupts): เหตุการณ์ที่เกิดจากการทำงานร่วมกับอุปกรณ์อินพุตและเอาต์พุต เช่น ปุ่มกดต่างๆ เซ็นเซอร์ต่างๆ ในตัวอุปกรณ์ วงจรเชื่อม

ต่อกับทางขา GPIO, ตัวจับเวลา (Timer) ถึงเวลาที่ตั้งไว้, การแปลงอนาล็อกเป็นดิจิทัลเสรีจสมบูรณ์, ตัวจับเวลาวือท์ดอก (Watchdog Timer) การรับส่งข้อมูลแบบอนุกรม (Serial communication) เป็นต้น

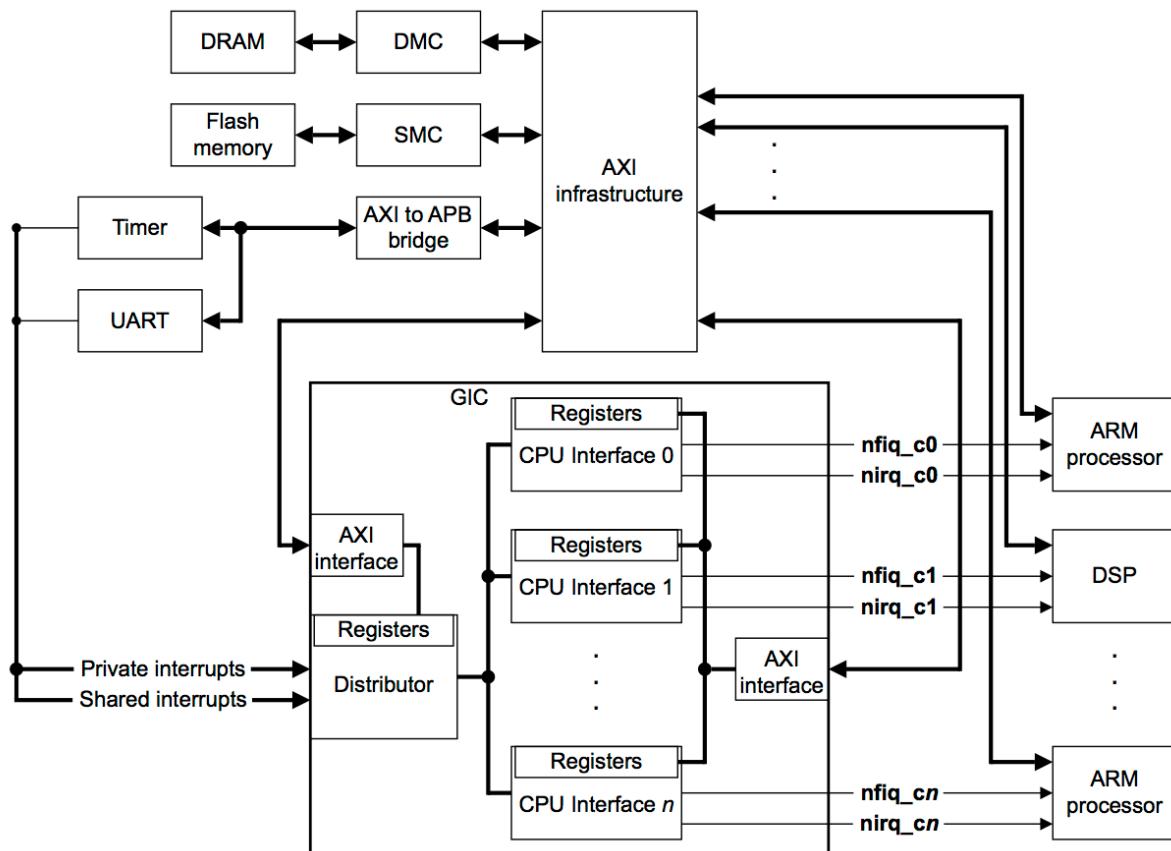
- **อินเทอร์รัปท์จากการทำงานของซอฟต์แวร์ (Software interrupts):** เหตุการณ์ที่เกิดจากการทำงานหรือสั่งการโดยซอฟต์แวร์ เช่น การเรียกใช้บริการจากระบบปฏิบัติการ ความผิดพลาดของโปรแกรม เป็นต้น
2. ฮาร์ดแวร์ที่เกี่ยวข้องกับเหตุการณ์ส่งสัญญาณร้องขอการอินเทอร์รัปท์ (Interrupt Request) ไปยังวงจรบริหารจัดการอินเทอร์รัปท์ วงจนนี้เรียกว่า อินเทอร์รัปท์คอนโทรลเลอร์
  3. อินเทอร์รัปท์คอนโทรลเลอร์ทำหน้าที่จัดลำดับความสำคัญของสัญญาณร้องขอตามชนิดของฮาร์ดแวร์ และลำดับเวลาที่สัญญาณร้องขอเข้ามา แล้วจึงกระจายสัญญาณร้องขอไปให้แกนประมวลผลที่เหมาะสม เช่น ว่างอยู่ หรือ กำลังปฏิบัติงานที่มีความสำคัญน้อยกว่า
  4. **ซีพียูจำนวนหนึ่งแกนประมวลผลปฏิบัติตามขั้นตอนของ ISR :** แกนประมวลผลนั้นจะหยุดพักการทำงานใดๆ ณ เวลานั้น เพื่อปฏิบัติตามขั้นตอนของฟังก์ชันที่เขียนไว้ล่วงหน้า และให้บริการตามเหตุการณ์ที่ร้องขอภายใต้หมวดเครื่องเนล

กรณีศึกษาของระบบควบคุมอินเทอร์รัปท์คอนโทรลเลอร์ของซีพียู ARM ชื่อ Generic Interrupt Controller (GIC) ในรูปที่ 6.19 ออกแบบมาสำหรับซีพียู ARM Cortex A รุ่นต่างๆ เพื่อรับการอินเทอร์รัปท์ที่ซับซ้อนเนื่องจากซีพียูจำนวนซีพียูตั้งแต่ 2 แกนประมวลผลขึ้นไปและมีอุปกรณ์อินพุต/เอาต์พุตที่หลากหลาย โดยทั้งหมดเชื่อมต่อกันด้วย ARM Advanced eXtensible Interface (AXI) ซึ่งเป็นบัสชนิดหนึ่งมีโครงสร้างและโปรโตคอลที่ซับซ้อน ผู้อ่านสามารถค้นควารายละเอียดและวิวัฒนาการของ AXI เพิ่มเติมที่ [wikipedia](#)

เหตุการณ์อินเทอร์รัปท์มีความสำคัญ (Priority) แตกต่างกัน รูปที่ 6.20 แสดงได้อย่างภาพการทำงานของ GIC เมื่อเกิดการเกิดอินเทอร์รัปท์ซ้อน (Nested Interrupt) นั่นคือ สัญญาณร้องขออินเทอร์รัปท์ (Interrupt Request) จำนวนสองสัญญาณ คือ Interrupt M และ Interrupt N มาถึงยังวงจรควบคุมไม่พร้อมกัน การตอบสนองต่อการร้องขอที่เกิดขึ้นนั้นจะขึ้นกับความสำคัญ (Priority) เป็นหลัก ซีพียูสามารถหยุดพัก (Pending) การร้องขอที่มาถึงก่อนแต่มีความสำคัญต่ำกว่าชั่วคราว เพื่อตอบสนองต่อสัญญาณการร้องขอที่มีภาระหนักแต่มีความสำคัญสูงกว่า (Higher Priority) แล้วจึงตอบสนองสัญญาณที่พักไว้ต่อ ตามขั้นตอนต่อไปนี้

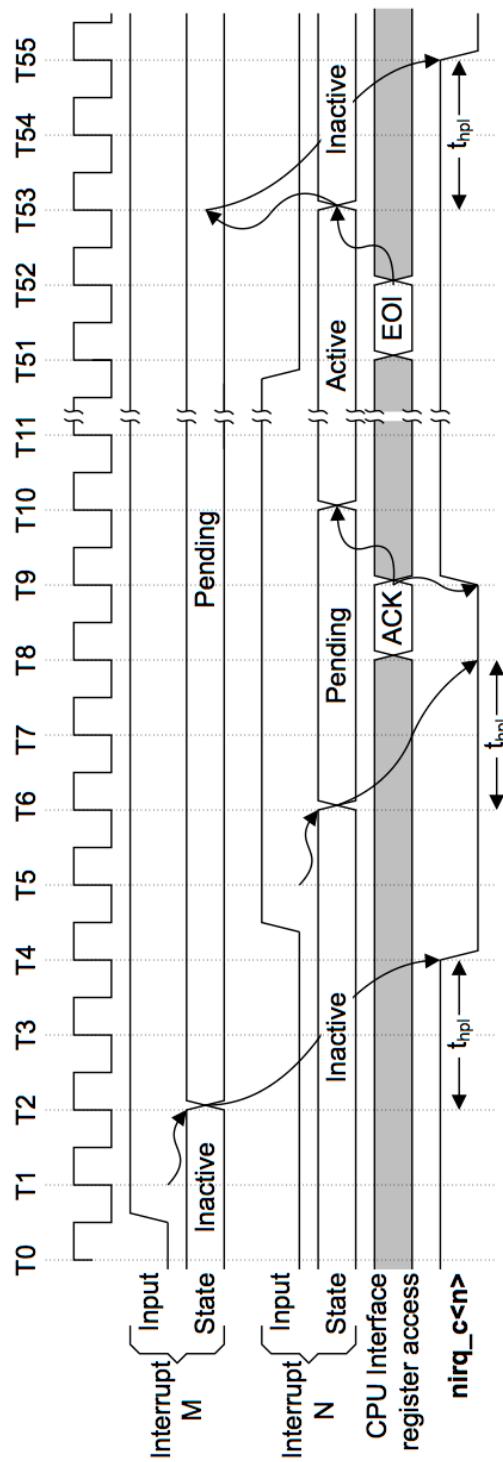
แกนนอนในรูปที่ 6.20 เป็นแกนเวลาเริ่มต้นจากค่าเวลาที่ T0 จนถึงค่าเวลาที่ T55 วงร่ายในรูปนี้ทำงานที่ขอบขั้นของสัญญาณคลื่อเท่านั้น ตามลำดับดังนี้

- ค่าเวลา T1 วงจรแจกจ่ายอินเทอร์รัปท์ (Distributor) ตรวจจับการร้องขอของ Interrupt M ของอุปกรณ์ M
- ค่าเวลา T2 วงจรแจกจ่ายอินเทอร์รัปท์พักการร้องขอจาก Interrupt M
- ค่าเวลา T4 วงจรแจกจ่ายอินเทอร์รัปท์ ตั้งค่า Interrupt M ให้กับแกนประมวลผล n ผ่านทางสัญญาณร้องขอการอินเทอร์รัปท์ **nirq\_c<n>**
- ค่าเวลา T5 วงจรแจกจ่ายอินเทอร์รัปท์ตรวจจับการร้องขอของ Interrupt N ซึ่งมีความสำคัญสูงกว่า Interrupt M จึงต้องพักการทำงานของ Interrupt M ในค่าเวลาถัดไป
- ค่าเวลา T6 วงจรแจกจ่ายอินเทอร์รัปท์ Interrupt N ให้กับแกนประมวลผลที่ n ผ่านทางสัญญาณร้องขอการอินเทอร์รัปท์ **nirq\_c<n>** แทน



รูปที่ 6.19: โครงสร้างส่วนหนึ่งภายในชิป BCM283x แสดงการเชื่อมต่อแกนประมวลผลต่างๆ กับวงจร Generic Interrupt Controller (GIC) ผ่าน ARM Advanced eXtensible Interface (AXI) หมายเหตุ APB: ARM Peripheral Bus ที่มา: [arm.com](http://arm.com)

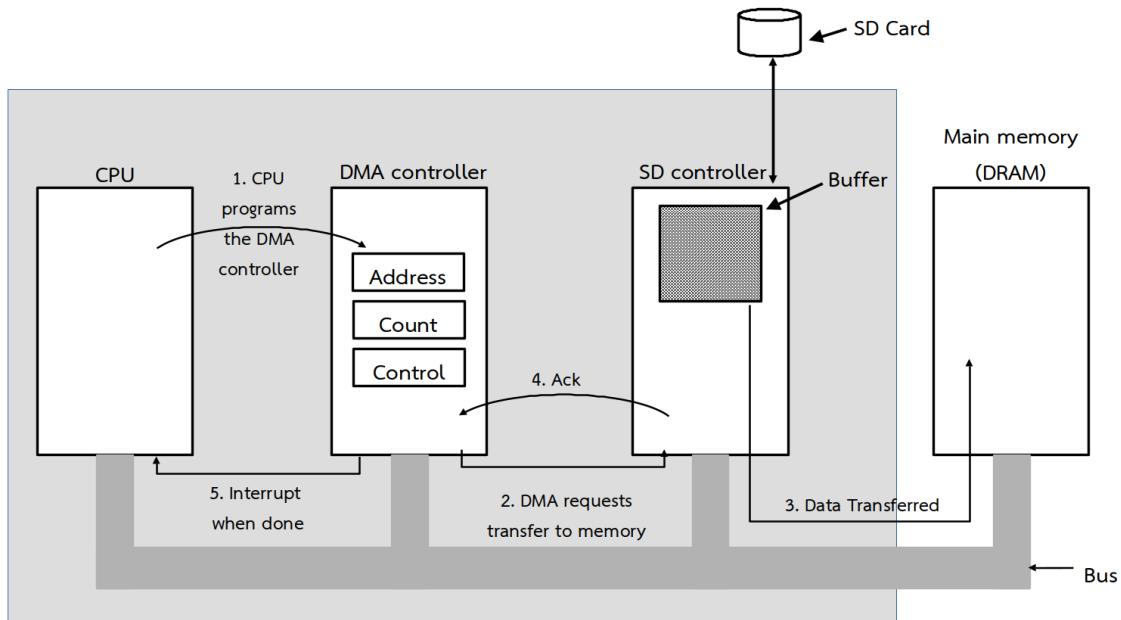
- คาดเวลา T8 ซีพียูตอบกลับ (Acknowledge) ด้วยสัญญาณ ACK สัญญาณร้องขอของ Interrupt N
- คาดเวลา T9 แกนประมวลผลที่ n อ่านค่าและตอบสนองต่อ  $nirq\_c< n>$  โดยเปลี่ยนค่าของรีจิสเตอร์ Interrupt Acknowledge Register จากโลจิก 0 ให้เป็นโลจิก 1
- คาดเวลา T10 วจจะแจกจ่ายอินเทอร์รัปท์ตั้งค่าสถานะของ Interrupt N เป็น Active และเปลี่ยนแปลง ค่ารีจิสเตอร์ Active Status Register
- คาดเวลา T11-T50 แกนประมวลผลที่ n ทำงาน ISR ของ Interrupt N จนแล้วเสร็จ
- คาดเวลา T51 แกนประมวลผลที่ n บันทึกค่ารีจิสเตอร์ End of Interrupt (EOI) ด้วยหมายเลข INTID (Interrupt ID) ของ Interrupt N
- คาดเวลา T52 วจจะแจกจ่ายอินเทอร์รัปท์ตั้งค่าสถานะของ Interrupt N เป็น Inactive และเปลี่ยนแปลง ค่ารีจิสเตอร์ Active Status Register เป็นโลจิก 0
- คาดเวลา T53 วจจะแจกจ่ายอินเทอร์รัปท์แจ้งวจอินเตอร์เฟสของแกนประมวลผลที่ n ว่า interrupt M ยังคงอยู่และมีความสำคัญสูงที่สุด ณ เวลานี้
- คาดเวลา T55 วจจะแจกจ่าย Interrupt M ให้กับแกนประมวลผลที่ n โดยเปลี่ยนสัญญาณร้องขอ  $nirq\_c< n>$  ให้กลายเป็นโลจิก 0 เพื่อขัดจังหวะการทำงานเป็นลำดับถัดไป



รูปที่ 6.20: ไดอะแกรมเวลาการทำงานของ Generic Interrupt Controller (GIC) เพื่อตอบสนองต่อสัญญาณร้องขออินเทอร์รัปท์ที่มีความสำคัญไม่เท่ากัน ที่มา: [Ltd. \(2017\)](#)

ในการทดลองที่ 11 ภาคผนวก K ผู้อ่านสามารถใช้งานขา GPIO ทำงานร่วมกับการอินเทอร์รัปท์ โดยพัฒนาโปรแกรมประยุกต์ด้วยภาษา C และภาษาแอสเซมบลีทำงานร่วมกับไลบรารี wiringPi และกิจกรรมท้ายการทดลองจะช่วยเสริมให้ผู้อ่านเข้าใจการทำงานของอินเทอร์รัปท์เพิ่มมากขึ้น

## 6.13 หลักการเข้าถึงหน่วยความจำโดยตรง (Direct Memory Access)

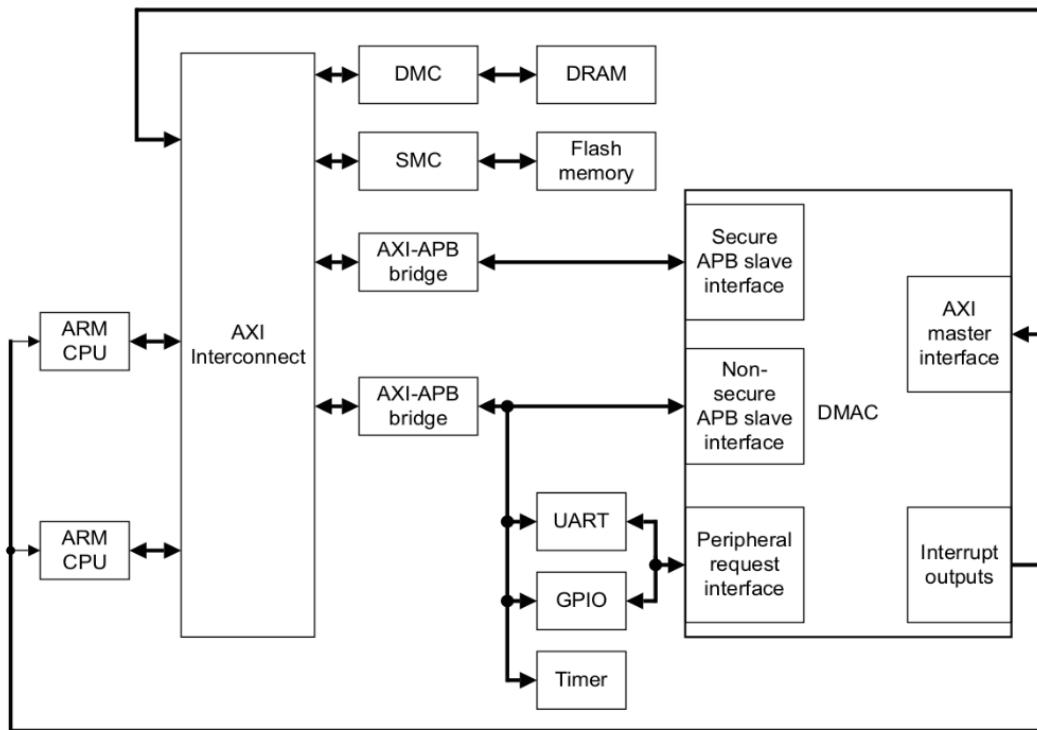


รูปที่ 6.21: ลำดับการทำงานของ DMA Controller (DMAC) เพื่อควบคุมการอ่านข้อมูลจากการ์ดหน่วยความจำ SD ไปบันทึกในหน่วยความจำหลัก (Main Memory)

การรับส่งข้อมูลผ่าน GPIO มีข้อจำกัดในเรื่องของความเร็ว จึงไม่เหมาะสมกับการถ่ายโอนข้อมูลจำนวนมาก โดยเฉพาะอุปกรณ์ที่เกี่ยวข้องกับข้อมูลจำนวนมากในเวลาที่จำกัด ดังนั้น นักพัฒนาซิ皮ยูจึงได้ออกแบบขบวนการ อินเทอร์รัปท์ ยกตัวอย่าง เช่น การรับส่งข้อมูลผ่านเครือข่าย Ethernet ในรูปของเฟรมข้อมูลที่มีความยาวขั้นต่ำ 64 ไบต์และไม่เกิน 1522 ไบต์ ดังนั้น ในการรับ/ส่งข้อมูลกับบอร์ดเฟอร์ชองอุปกรณ์อินพุต/เอาต์พุต DMA จะทำ หน้าที่เคลื่อนย้ายข้อมูลจากบอร์ดหน่วยความจำหลัก ยกตัวอย่าง เช่น รูปที่ 6.21 แสดงลำดับการทำงานของ DMAC เพื่อควบคุมการอ่านข้อมูลจากบอร์ดเฟอร์กายในวงจร SD Controller ซึ่งทำหน้าที่ควบคุมการทำงานของการ์ดหน่วยความจำ SD ไปบันทึกในหน่วยความจำหลัก (SDRAM) ประกอบด้วยขั้นตอนต่อๆ ดังนี้

1. ซิปิยูโปรแกรมการทำงานของ DMAC โดยตั้งค่ารีจิสเตอร์ที่เกี่ยวข้อง ได้แก่ แอดเดรสเริ่มต้นในหน่วยความจำหลัก จำนวนไบต์ที่ต้องการอ่านค่า (Count) และรายละเอียดอื่นๆ ของ DMA Controller (DMAC) ถูกemapบนบัสแอดเดรสเริ่มต้นที่หมายเลข 0x7E00 7000 ของตารางที่ 6.4 สำหรับกรณีศึกษา ชิป BCM2837
2. DMAC เริ่มต้นทำการอ่านข้อมูลจากบอร์ดเฟอร์กายใน SD Controller ไปบันทึกหน่วยความจำหลัก ณ แอดเดรสเริ่มต้นที่ตั้งค่าจากข้อ 1
3. ข้อมูลจะถูกอ่านจากบอร์ดเฟอร์กายใน SD Controller (ต้นทาง) เดินทางผ่านบัสความเร็วสูง AXI และเขียนลงในหน่วยความจำหลักที่แอดเดรสเริ่มต้นจนแล้วเสร็จตามจำนวนไบต์ (Count) ที่ตั้งค่าไว้
4. วงจร SD Controller ส่งสัญญาณตอบรับ ACK ไปยัง DMAC

5. DMAC ส่งสัญญาณอินเทอร์รัปท์แจ้งซีพียูว่าดำเนินการแล้วเสร็จผ่านทางวงจรควบคุมอินเทอร์รัปท์
6. ซีพียูตอบสนองต่อสัญญาณอินเทอร์รัปท์ที่ได้รับจาก DMAC



รูปที่ 6.22: โครงสร้างส่วนหนึ่งภายในชิป BCM283x/BCM2711 แสดงการเชื่อมต่อแกนประมวลผลต่างๆ กับวงจร DMAC (DMA Controller) ผ่าน ARM Advanced eXtensible Interface (AXI) และ AXI-APB Bridge ที่มา: [arm.com](http://arm.com)

ตำราเล่มนี้จะใช้กรณีศึกษา DMAC ของ ARM ในรูปที่ 6.22 โครงสร้างส่วนหนึ่งภายในชิป BCM2837 ซีพียู Cortex A53/A72 ทั้ง 4 แกนประมวลผลเชื่อมต่อกับวงจร DMAC ผ่าน ARM Advanced eXtensible Interface (AXI) และ AXI-APB Bridge ชิป SDRAM จะเชื่อมต่อกับบัส AXI ในขณะที่อุปกรณ์อินพุต/เอาต์พุตส่วนใหญ่จะเชื่อมต่อกับบัส APB ซึ่งมีความเร็วต่ำกว่า ทำให้ DMAC สามารถรองรับขบวนการ DMA ได้พร้อมกัน 16 ช่องหรือ 16 อุปกรณ์ โดยแต่ละช่องสามารถทำงานได้อิสระจากกัน

ขบวนการ DMA ในแต่ละช่องเริ่มต้นจาก DMAC อ่านค่าการติดตั้งใน CB (Control Block) ในหน่วยความจำไปตั้งค่าริจิสเตอร์ CONBLK\_AD (Controller Block Address) ประกอบด้วย หมายเลขแอดเดรสต้นทาง หมายเลขแอดเดรสปลายทาง และจำนวนไบต์ที่ต้องการถ่ายโอน เป็นต้น เพื่อสั่งงาน DMA ช่อง (Channel) ที่ต้องการ หลังจากนั้น วงจร DMA ช่องนั้นจะเริ่มทำงานด้วยตนเอง เมื่อทำงานแล้วเสร็จ DMA จะส่งสัญญาณขัดจังหวะไปแจ้งเตือนซีพียูว่างานเสร็จสิ้นแล้ว ผ่าน GIC ตามที่ได้กล่าวไปแล้วในหัวข้อที่ 6.12 สำหรับกรณีศึกษาชิป BCM283x วงจร DMAC จะແນພັບສແດດເຊີຣມຕົ້ນທີ່ 0x7E00 7000 ຂອງຕາറາງທີ່ 6.4 ໂດຍເຮັງລຳດັບຕາມ หมายเลขช່ອງດັ່ງນີ້

- การทำงานของ DMA ช่อง 0 เริ่มต้นการตั้งค่าທີ່ແດດເຊີຣມ 0x7E00 7000
- การทำงานของ DMA ช่อง 1 เริ่มต้นการตั้งค่าທີ່ແດດເຊີຣມ 0x7E00 7100
- การทำงานของ DMA ช่อง 2 เริ่มต้นการตั้งค่าທີ່ແດດເຊີຣມ 0x7E00 7200

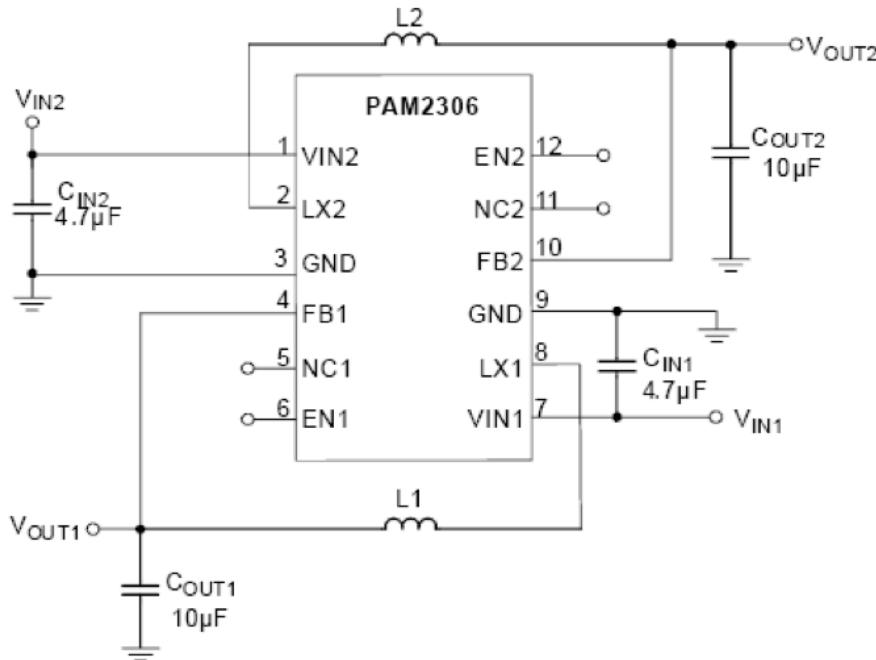
- ...

- การทำงานของ DMA ช่อง 14 เริ่มต้นการตั้งค่าที่แอดเดรส 0x7E00 7E00
- การทำงานของ DMA ช่อง 15 เริ่มต้นการตั้งค่าที่แอดเดรส 0x7EE0 5000

การทำงานของ DMA แต่ละช่องจะมีลักษณะเหมือนกัน คือ ตั้งค่าการทำงานของรีจิสเตอร์ด้วยชุดข้อมูล Control Block (CB) ชุดละ 256 ไบต์ CB แต่ละชุดจะเชื่อมโยงกันเป็นลิงก์ลิสต์ (Linked List) ในหน่วยความจำ เมื่อการทำงานตามรายละเอียดของ CB ชุดแรกเสร็จสิ้น ค่า CB ชุดต่อไปจะถูกอ่านเพื่อไปตั้งค่าการทำงานชุดต่อไปโดยอัตโนมัติ ทำให้ชิปยังสามารถทำงานอื่นที่สำคัญกว่าได้ต่อเนื่อง และรับทราบผลเมื่อแต่ละขั้นตอนการแล้วเสร็จผ่านกลไกการอินเทอร์รัปท์

## 6.14 แหล่งจ่ายไฟ (Power Supply)

แหล่งจ่ายไฟของบอร์ดมาจากการแอดปเตอร์ (Adaptor) ที่แปลงไฟฟ้ากระแสสลับ 220 โวลต์เป็นไฟกระแสตรง ความต่างศักย์ 5.1 โวลต์ โดยใช้หัวคอนเนคเตอร์ไมโคร USB จะต้องจ่ายกระแสไฟฟ้าได้ไม่น้อยกว่า 2.6 แอม培ร์ ทั้งนี้ เพื่อให้บอร์ด Pi3/Pi4 สามารถทำงานได้เต็มประสิทธิภาพ กระแสจะไหลผ่านพาวเวอร์แลดได้อด เพื่อจ่ายไฟให้กับบอร์ด Pi3/Pi4 แล้วจึงแปลงไฟกระแสตรงความต่างศักย์ 5.1 โวลต์ให้ลดลง (Step Down) เป็นไฟกระแสตรง ความต่างศักย์ 3.3 และ 1.8 โวลต์ด้วยชิป PAM2306 และ 1.2 โวลต์ด้วยชิป RT8088A ตามลำดับ



รูปที่ 6.23: การแปลงไฟกระแสตรง 5 โวลต์เป็นไฟกระแสตรงด้วยชิป PAM 2306 DC-DC Covertor คู่ กำหนดค่าเอาต์พุตด้วยค่าตัวเหนี่ยววน L<sub>1</sub> และ L<sub>2</sub> หน่วยเป็น ไมโครเอนวี ที่มา: [Diodes \(2012\)](#)

ชิป PAM2306 จะทำหน้าที่แปลงไฟกระแสตรงความต่างศักย์ 5.0 โวลต์ ให้เป็นไฟกระแสตรงความต่างศักย์ 3.3 โวลต์ เพื่อเลี้ยงชิปต่างๆ และวงจรทั้งหมด PAM2306 รองรับอินพุตไฟกระแสตรงตั้งแต่ 2.5 ถึง 5.5 โวลต์ โดยสามารถแปลงเอาต์พุตโวลาเจได้เป็นสองระดับพร้อมๆ กัน ซึ่งผู้ออกแบบสามารถเลือกได้ดังนี้ 3.3, 2.8, 2.5, 1.8, 1.5, 1.2 โวลต์ หรือใช้ความต้านทานปรับค่าได้ PAM2306 ทำงานในลักษณะ DC-DC Converter ควบคุมการทำงานแบบ Step Down ด้วยกระแส PWM (Pulse Width Modulation) เพื่อให้เกิดเสียงรบกวน และรักษา

อายุการใช้งานของแหล่งจ่ายไฟเมื่อต้องจ่ายกระแสมากขึ้น และเปลี่ยนมาใช้หลักการ PSM (Pulse-Skipping Modulation) เพื่อลดกระแสในขณะที่จ่ายกระแสอยู่

หาก  $V_{IN1}$  จะถูกแปลงให้มีความต่างศักย์ลดลงเป็น  $V_{OUT1}$  ชา  $V_{IN2}$  จะถูกแปลงให้มีความต่างศักย์ลดลงเป็น  $V_{OUT2}$  โดยจะควบคุมการเปิดปิดด้วยขา EN1 (Enable1) และ EN2 (Enable2) ตามลำดับ ค่าโวลเทจทั้ง  $V_{OUT1}$  และ ค่าโวลเทจทั้ง  $V_{OUT2}$  ถูกกำหนดที่หมายเลขตัวถังของซิปมาจากการผลิต ผู้ออกแบบต้องเลือกค่าตัวเหนี่ยวนำให้เหมาะสมเพื่อกำหนดความต่างศักย์ของ  $V_{OUT1}$  และ  $V_{OUT2}$  โดยกำหนดจากค่าตัวเหนี่ยวนำ  $L_1$  และ  $L_2$  หน่วยเป็นไมโครเอนรี (Micro Henry) ตามลำดับได้ตามตารางที่ 6.7

ตารางที่ 6.7: ตารางเลือกค่า  $V_{OUT}$  และค่าตัวเหนี่ยวนำ  $L_1$  และ  $L_2$  ที่มา: [Diodes \(2012\)](#)

$V_{OUT}$ (โวลต์)	$L$ ( $\mu\text{H}$ )
1.2	2.2
1.5	2.2
1.8	2.2
2.5	4.7
3.3	4.7

เอกสารคุณสมบัติความต้องการด้านกำลังไฟสำหรับบอร์ด Pi3/Pi4 ได้ระบุว่าบอร์ดต้องการกระแสสูงถึง 2.5 แอม培ร์ ขึ้นอยู่กับโมเดลที่ใช้และการใช้งานจริง โดยเชื่อมต่อกับอุปกรณ์ เช่น เม้าส์ คีย์บอร์ด และ WiFi ผ่านพอร์ต USB แหล่งจ่ายไฟควรมีความต่างศักย์  $5 \pm 0.25\text{V}$  ที่กระแส 1 แอม培ร์หรือมากกว่า โดยผู้ใช้ไม่ต้องกังวลว่าการจ่ายกระแสมากเกินไปแล้วจะเป็นผลเสีย เพราะบอร์ดจะปรับกระแสเท่าที่จำเป็นและมีฟิล์สค้อยช่วยจำกัดกระแส 2.5 แอมเบอร์ นอกจานี้ บอร์ด Pi3/Pi4 และ Pi2 ไม่เดล B+ มีซิปหมายเลข APX803 ทำหน้าที่เฝ้าระวัง (Monitor) โวลเทจ กรณีที่ไฟเลี้ยงจาก USB มีความต่างศักย์อยู่ในช่วง  $4.63 \pm 0.07$  โวลต์ ไฟ LED สีแดงบนบอร์ดจะสว่างขึ้น สัญลักษณ์สายฟ้าและข้อความว่า Under Voltage ผ่านทางหน้าจอตรงมุมขวาบน จะเตือนให้ผู้ใช้ทราบว่าไฟเลี้ยงของบอร์ดมีค่าต่ำไปแต่บอร์ดยังใช้งานได้ตามปกติ

ข้อควรระวังเรื่องแหล่งจ่ายไฟให้กับบอร์ด Pi3/Pi4 คือ ขนาดและคุณภาพของสายไมโคร USB ที่จ่ายไฟเลี้ยง 5 โวลต์ สายที่มีขนาดเล็กเกินไปจะทำให้ความต้านทานของสายสูงขึ้น เมื่อกระแสไฟ流ผ่านจะทำให้เกิดโวลเทจตกคร่อมในสายสูงขึ้นตาม จนทำให้บอร์ดได้รับโวลเทจไม่สูงพอ นั่นคือ ต่ำกว่า 4.63 โวลต์ จนส่งผลถึงความต่างศักย์ด้านเอาต์พุตของ PAM2306 ที่อาจต่ำกว่า 3.3 โวลต์ และทำให้ชิป BCM2837 ขาดเสียราก

## 6.15 สรุปท้ายบท

แกนประมวลผลทั้งหมด หน่วยความจำภายในภาพ และอุปกรณ์เก็บรักษาข้อมูลของเครื่องคอมพิวเตอร์บอร์ดเดี่ยว Pi3/Pi4 เชื่อมเข้าหากันด้วยการเชื่อมต่อความเร็วสูง ซึ่งว่า AXI (Advanced eXtensible Interface) ซึ่งทำหน้าที่เป็นบัสความเร็วสูง เพื่อรับการถ่ายโอนข้อมูลจำนวนมากด้วยกระบวนการ DMA ระหว่างหน่วยความจำภายในภาพ กับ

- อุปกรณ์เก็บรักษาข้อมูล (ในบทที่ [7](#))
- อุปกรณ์ USB (ในหัวข้อที่ [6.6](#))

ส่วนวงจร (มอดูล) ด้านอินพุต/เอาต์พุตอื่นๆ ที่มีความเร็วต่ำถึงปานกลาง เช่น วงจรข้อมูลเสียงดิจิทัลชนิด PCM (ในหัวข้อที่ [6.4](#)) วงจร GPIO (ในหัวข้อที่ [6.11](#)) เป็นต้น จะเชื่อมอยู่บนบัสอุปกรณ์ต่อพ่วง (APB: ARM Peripheral Bus) ซึ่งทำหน้าที่เป็นบัสความเร็วปานกลาง โดยจะมีวงจรเชื่อมทั้งสองบัสเข้าด้วยกันเรียกว่า AXI-APB บริดจ์ (Bridge)

ซึ่งสามารถควบคุมการทำงานของอุปกรณ์อินพุต/เอาต์พุตเหล่านี้ ด้วยการอ่าน/เขียนรีจิสเตอร์ภายในวงจร หรือมองดูลูกอุปกรณ์นั้นๆ โดยการจัดวาง (map) หมายเลขอัตโนมัติเดรสซึ่งแปลจากแอดเดรสภายในภาพ และเวอร์ชัลแอดเดรสอีกทอดหนึ่งให้ตรงกับรีจิสเตอร์เหล่านั้น เรียกว่า หลักการ **Memory Mapped Input/Output** ที่มา: [Tanenbaum and Austin \(2012\)](#); [Harris and Harris \(2013\)](#) กลไกหลักที่จะช่วยแบ่งเบาภาระการทำงานด้านอินพุต/เอาต์พุตของซีพียู คือ กลไกอินเทอร์รัปท์มีวงจรควบคุมอินเทอร์รัปท์ เรียกว่า GIC และกระบวนการ DMA มีวงจรควบคุม เรียกว่า DMA Controller (DMAC) สำหรับกรณีศึกษาซีพียู ARM ประเด็นสุดท้ายและสำคัญที่สุดที่มักถูกมองข้าม คือ แหล่งจ่ายไฟที่จะทำให้บอร์ดทำงานเชื่อมต่อกับอุปกรณ์เหล่านี้อย่างมีเสถียรภาพ

## 6.16 คำถามท้ายบท

1. BCM2837 มีจำนวนขาสัญญาณเพื่อเชื่อมต่อกับบอร์ดจำนวนกี่ขา
2. จงบอกจำนวนกล้องชนิด CSI ที่ชิป BCM2837 รองรับได้สูงสุด
3. จงบอกจำนวนจอภาพชนิด DSI ที่ชิป BCM2837 รองรับได้สูงสุด
4. จงบอกซี่อขาที่ชิป BCM2837 สำหรับเชื่อมต่อกับการดحن่วยความจำ SD ทั้งหมด
5. จงบอกจำนวนเลนช์ข้อมูลของการเชื่อมต่อกับจอภาพชนิด HDMI ที่ชิป BCM2837 รองรับได้สูงสุด
6. จงบอกจำนวนเลนช์ข้อมูลของการเชื่อมต่อกับกล้องชนิด CSI ที่ชิป BCM2837 รองรับได้สูงสุด
7. จงบอกจำนวนเลนช์ข้อมูลของการเชื่อมต่อกับจอภาพชนิด DSI ที่ชิป BCM2837 รองรับได้สูงสุด
8. จงบอกตำแหน่งรีเมตัน ตำแหน่งสิ้นสุด และขนาดของแอดเดรสบัส ในรูปที่ [6.16](#)
9. จงบอกความหมายของ L1 and L2 cached ในรูปที่ [6.16](#)
10. จงบอกความหมายของ L2 cache coherent (non allocating) ในรูปที่ [6.16](#)

11. จงบอกความหมายของ L2 cached (only) ในรูปที่ [6.16](#)
12. จงบอกความหมายของ direct uncached ในรูปที่ [6.16](#)
13. การขัดจังหวะของชีพียูเกิดขึ้นจากอุปกรณ์อินพุต/เอาต์พุตอะไรบ้าง จงยกตัวอย่างโทรศัพท์สมาร์ตโฟน
14. เหตุใดการเชื่อมต่อกับกล้องและจอภาพจึงต้องมีสัญญาณคลื่อกด้วย
15. เหตุใดการเชื่อมต่อกับ USB, Ethernet และอื่นๆ จึงไม่ต้องมีสัญญาณคลื่อกทั้งๆ ที่เป็นการส่งด้วยเทคนิค Differential Voltage Signaling เช่นเดียวกับกล้องและจอภาพ
16. นอกเหนือจากสัญญาณ WiFi และ Bluetooth ชิป BCM43438 รองรับสัญญาณใดเพิ่มเติม
17. จงอธิบายหลักการเชื่อมต่อกับ GPIO ของชิป BCM2837/BCM2711 และการขัดจังหวะ โดยตรวจจับ สัญญาณอินพุตที่ขอบขาลง (Falling Edge)
18. เหตุใดการบริโภคพลังงานของบอร์ดจึงต้องการความต่างศักย์ที่ 5 โวลต์แต่ต้องการกระแสไฟตั้งแต่ 700 mA - 2.5 A
19. เหตุใดบอร์ดจึงต้องแปลงไฟเลี้ยง 5 โวลต์เป็น 3.3 และ 1.8 โวลต์

## บทที่ 7

# อุปกรณ์เก็บรักษาข้อมูล (Data Storage Devices)

ผู้เขียนใช้คำว่า อุปกรณ์เก็บรักษาข้อมูล ในขณะที่ทำงานเล่น และเว็บไซต์บางที่เรียกว่า หน่วยความจำ หรือ หน่วยเก็บข้อมูล หรือ หน่วยบันทึกข้อมูล ทำหน้าที่บันทึกและเก็บไฟล์ทั้งหมดของระบบปฏิบัติการ ไฟล์โปรแกรม ไฟล์ข้อมูล และไฟล์อื่นๆ เพื่อให้เครื่องคอมพิวเตอร์สามารถทำงานได้ต่อเนื่องเมื่อปิดแล้วเปิดเครื่องใหม่ อีกรอบหลังจากการซัตดาวน์ การปิดเครื่องคอมพิวเตอร์อาจมีสาเหตุหลายประการ เช่น การประยัดพลังงาน การบำรุงรักษาระบบ หรือกรณีฉุกเฉินไฟฟ้าดับ หรือแบตเตอรี่หมด ซึ่งอาจทำให้ไฟล์เสียหายหรือถึงขั้นสูญหาย ดังนั้น ผู้ใช้และซอฟต์แวร์ต้องมีหน้าที่บันทึก (Save) ข้อมูลลงในไฟล์ข้อมูลระหว่างการปฏิบัติงานด้วย เช่น กัน กรณีผู้ใช้ไม่สามารถซัตดาวน์ระบบอาจทำให้ระบบไฟล์และไฟล์เสียหายได้ ทั้งนี้ขึ้นอยู่กับชนิดของระบบไฟล์

เทคโนโลยีที่ใช้สร้างอุปกรณ์เก็บรักษาข้อมูลที่ได้รับความนิยมในปัจจุบัน ได้แก่ เทคโนโลยีสารกึ่งตัวนำ เทคโนโลยีสารแม่เหล็ก และ เทคโนโลยีสารสะท้อนแสง ดังนั้น อุปกรณ์เก็บรักษาข้อมูลจึงแบ่งเป็นชนิดต่างๆ ตามสารที่ใช้ ดังนี้

- อุปกรณ์เก็บรักษาข้อมูลชนิดสารกึ่งตัวนำ ได้แก่ โซลิดสเตทไดร์ฟ และ หน่วยความจำการ์ด SD ซึ่งอาศัย ชิปหน่วยความจำขนาดแฟลช ทำงานที่สัญญาณคลื่นความถี่สูงระดับ 100 เมกะเฮิรตซ์
- อุปกรณ์เก็บรักษาข้อมูลชนิดสารแม่เหล็ก ได้แก่ ฮาร์ดดิสก์ไดร์ฟอาศัยการหมุนของจานแม่เหล็กแข็งที่ ความเร็วรอบสูง
- อุปกรณ์เก็บรักษาข้อมูลชนิดสารสะท้อนแสง ได้แก่ แผ่นซีดี (CD: Compact Disc) แผ่นดิวีดี (DVD: Digital Video Disc) และแผ่นบลูเรย์ (Blu-ray) อาศัยการหมุนของจานสะท้อนที่ความเร็วรอบปานกลาง

โปรดสังเกตว่าอุปกรณ์เหล่านี้ล้วนทำงานที่สัญญาณคลื่นความถี่ 100-500 เมกะเฮิรตซ์ ซึ่งต่ำกว่าหน่วยความจำ SDRAM และ SRAM ซึ่งทำงานที่สัญญาณคลื่นความถี่หลักกิกะเฮิรตซ์ แต่ อุปกรณ์เก็บรักษาข้อมูลไม่ต้องอาศัยไฟเลี้ยงเพื่อรักษาข้อมูลมิให้สูญหายหรือเสียหายตลอดเวลา เทคโนโลยีการเก็บรักษาข้อมูลเหล่านี้มีความแตกต่าง กัน มีข้อได้เปรียบเสียเปรียบต่างกัน โดยบทนี้จะเน้นที่ อุปกรณ์เก็บรักษาข้อมูลชนิดสารกึ่งตัวนำ และ อุปกรณ์เก็บรักษาข้อมูลชนิดสารแม่เหล็ก โดยมีวัตถุประสงค์ต่อไปนี้

- เพื่อให้เข้าใจโครงสร้างและกลไกการทำงานเบื้องต้นของระบบไฟล์ (File System) ในระบบปฏิบัติการ ตรรกะล瑜นิกซ์
- เพื่อให้รู้จักชนิดและกลไกการทำงานเบื้องต้นของ อุปกรณ์เก็บรักษาข้อมูล ได้แก่ หน่วยความจำแฟลช การ์ดหน่วยความจำ SD เป็นต้น

ผู้อ่านควรย้ำทำความเข้าใจคำสั่งพื้นฐานของระบบยูนิกซ์ในการทดลองที่ 4 ภาคผนวก D ซึ่งจะช่วยเสริมความเข้าใจระบบไฟล์ของระบบลินุกซ์เพิ่มเติมในการทดลองที่ 12 ภาคผนวก L

## 7.1 ระบบไฟล์ (File System)

ไฟล์จะเก็บอยู่ในอุปกรณ์เก็บรักษาข้อมูลตามโครงสร้างหรือระบบไฟล์ของระบบปฏิบัติการ ผู้ใช้และนักพัฒนาสามารถใช้งานไฟล์โดยไม่เห็นความแตกต่างใดๆ เช่น สร้างไฟล์ (Create) เขียน/บันทึก/อ่านไฟล์ เปลี่ยนชื่อ (Rename) ไฟล์ ลบ (Remove/Delete) ไฟล์ ทำสำเนา (Copy) ไฟล์ ย้ายตำแหน่ง (Move) ไฟล์ และกู้คืน (Recover) ไฟล์ เมื่ออุปกรณ์เกิดปัญหา โดยไม่จำเป็นต้องรับรู้ว่าระบบไฟล์เป็นชนิดใด เนื่องจากระบบปฏิบัติการได้ซ่อนรายละเอียดไว้ เพื่อให้ผู้ใช้และนักพัฒนาสามารถเขียนโปรแกรมได้สะดวกมากขึ้น ตำราเล่มนี้จะเน้นที่พื้นฐานของระบบไฟล์ดังเดิมของระบบยูนิกซ์ ซึ่งเป็นระบบไฟล์พื้นฐานให้ระบบอื่นๆ ได้พัฒนาต่ออยอดเป็นระบบบริหารจัดการไฟล์ที่สำคัญและเป็นที่นิยม ได้แก่

- ระบบ NTFS (File System) สำหรับระบบบินโดเวิร์ส รายละเอียดเพิ่มเติมที่ [ntfs.com](http://ntfs.com)
- ระบบ EXT4 สำหรับระบบลินุกซ์ รายละเอียดเพิ่มเติมที่ [wikipedia](http://wikipedia)
- ระบบ Apple File System (APFS) สำหรับระบบ MAC OS รายละเอียดเพิ่มเติมที่ [wikipedia](http://wikipedia)

### 7.1.1 การใช้งานไฟล์ (File Operations)

การเปิดไฟล์ คือ การจ่องเรอร์ชวอลเมโมรีจำนวนหนึ่งเพื่อทำหน้าที่เป็นบัฟเฟอร์ โปรแกรมเมอร์ต้องเปิดไฟล์โดยใช้ฟังก์ชัน fopen() ในภาษา C โดยเริ่มนค่าพอยน์เตอร์ หรือ แอดдресเริ่มต้นของบัฟเฟอร์ ของไฟล์ให้กับโปรแกรมที่ทำการเปิดไฟล์ การเปิดไฟล์ แบ่งเป็น เพื่อการอ่านอย่างเดียว (Read Only) เพื่อการเขียนอย่างเดียว (Write Only) และเพื่ออ่านและเขียน (Read-Write)

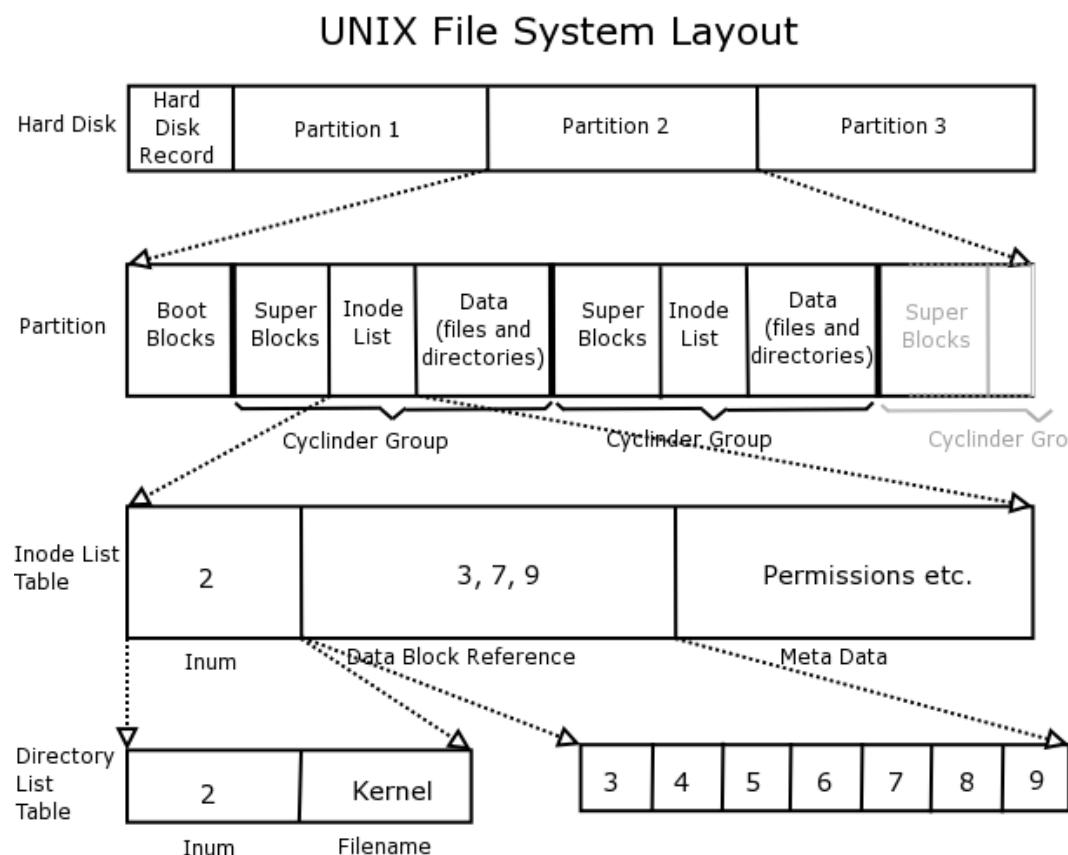
การอ่าน/เขียนข้อมูลในหน่วยความจำอยู่ในรูปของการอ่านเขียนไฟล์แทน ภาษา C/C++ มีฟังก์ชัน fread() สำหรับอ่านข้อมูลในไฟล์ และฟังก์ชัน fwrite() สำหรับเขียนข้อมูล และเมื่อต้องการเลิกใช้งานไฟล์ข้อมูลใดๆ

เมื่อโปรแกรมเมอร์เรียกใช้ฟังก์ชัน fread() ระบบปฏิบัติการ จะทำหน้าที่อ่านข้อมูลในเรอร์ชวอลเมโมรี ณ บริเวณซึ่งจะอ่านได้เป็นบัฟเฟอร์ตามหลักการ **Memory Mapped File** ซึ่งจะเริ่มต้นขึ้นเมื่อขบวนการ DMA ในหัวข้อที่ 6.13 อ่านข้อมูลจากการด่าน่วยความจำ SD มาเขียนลงในบัฟเฟอร์ เมื่อบัฟเฟอร์มีข้อมูลไม่เพียงพอ ขบวนการ DMA จะนำข้อมูลมาเติมให้จนอ่านเจตตัวอักษรสิ้นสุดไฟล์ (End of text File: EOF) เท่ากับ 1A<sub>16</sub> หรือ 26<sub>10</sub> ในตารางรหัสแอสกี ในรูปที่ 2.12 และ [softwareforeducation.com](http://softwareforeducation.com)

การเขียนข้อมูลเพื่อเก็บในไฟล์จะมีพิธีทางการไฟล์ที่ตรงข้ามกัน เมื่อโปรแกรมเมอร์เรียกใช้ฟังก์ชัน fwrite() ระบบปฏิบัติการทำหน้าที่เขียนข้อมูลในบัฟเฟอร์ ซึ่งจะอ่านตามหลักการ **Memory Mapped File** เช่นกัน เมื่อบัฟเฟอร์เต็มขบวนการ DMA จะอ่านข้อมูลไปเขียนในการด่าน่วยความจำ SD ทำให้บัฟเฟอร์ร่วงลง ระบบปฏิบัติการสามารถเขียนข้อมูลเพิ่มในบริเวณบัฟเฟอร์ และเมื่อบัฟเฟอร์เต็มขบวนการ DMA จะเกิดขึ้นอีกจนสิ้นสุดขบวนเขียนและปิดไฟล์ นั้นในที่สุด

การปิดไฟล์ คือ การคืนพื้นที่บัฟเฟอร์ให้กับระบบ โปรแกรมเมอร์จะต้องปิดไฟล์ด้วยฟังก์ชัน fclose() ระบบลินุกซ์ ได้พัฒนาฟังก์ชันเหล่านี้ให้กับโปรแกรมเมอร์ ซึ่งจะทำหน้าที่ติดต่อ กับอุปกรณ์เก็บรักษาข้อมูลผ่านวงจรต่างๆ

### 7.1.2 การแบ่งพาร์ทิชัน (Partition)



รูปที่ 7.1: โครงสร้างของระบบไฟล์บนอุปกรณ์เก็บรักษาข้อมูลในระบบปฏิบัติการตรวจสอบยุนิกซ์ ที่มา: [Demblon and Spitzner \(2004\)](#)

เราจะจินตนาการว่าอุปกรณ์เก็บรักษาข้อมูลในแวดวงสุดของรูปที่ 7.1 เช่น การ์ดหน่วยความจำ SD, โซลิดสเตตไทร์ฟ และฮาร์ดดิสก์ไทร์ฟ เหล่านี้ มีลักษณะเป็นแบบข้อมูลที่มีความยาวตามขนาดความจุ และแบ่งແນບออกเป็นพื้นที่อย่างๆ เรียกว่า พาร์ทิชัน (Partition) ซึ่งอุปกรณ์รักษาข้อมูล 1 ตัวสามารถแบ่งเป็นหลายพาร์ทิชัน (Partition) ผู้ใช้สามารถติดตั้งระบบไฟล์ (File System) ของแต่ละพาร์ทิชันได้อย่างอิสระ

การแบ่งพาร์ทิชัน คือ การแบ่งพื้นที่อุปกรณ์เก็บรักษาข้อมูลออกเป็นส่วนต่างๆ เพื่อตอบสนองความต้องการที่หลากหลาย ได้แก่ การบูตระบบปฏิบัติการได้หลายระบบภายในเครื่องเดียวกัน การจัดเก็บข้อมูลในพาร์ทิชันเฉพาะ เป็นต้น ผู้ใช้สามารถติดตั้งระบบปฏิบัติการในแต่ละพาร์ทิชันได้โดยอิสระจากกัน ที่มา: [archlinux](#) และ [datadoctor.biz](#) โดยตำแหน่งเริ่มต้นจะเป็นพื้นที่สำหรับเก็บ ตารางพาร์ทิชัน (Partition Table) ซึ่งตรงกับบริเวณ Hard Disk Record ของแวดวงสุดในรูปที่ 7.1 ประกอบด้วยรายชื่อและข้อมูลประกอบของแต่ละพาร์ทิชัน มาตรฐานของตารางพาร์ทิชันที่สำคัญในทางปฏิบัติ ได้แก่

- มาสเตอร์บูตเรคอร์ด (Master Boot Record: MBR) ทำหน้าที่เก็บรายชื่อพาร์ทิชัน ตำแหน่งเริ่มต้น หรือหมายเลข Logical Block Address (LBA) ในฮาร์ดดิสก์ไว้ในตารางพาร์ทิชัน (Partition Table) พื้นที่ส่วนหนึ่งของมาสเตอร์บูตเรคอร์ดทำหน้าที่เก็บตารางพาร์ทิชัน ซึ่งต้องอยู่ในเซกเตอร์แรก หรือเซกเตอร์หมายเลข 0 ของฮาร์ดดิสก์ โดยใช้พื้นที่ 64 ไบต์ ซึ่งฮาร์ดดิสก์ 1 ตัวสามารถแบ่งพาร์ทิชันได้มากที่สุด เป็นจำนวน 4 พาร์ทิชัน รายละเอียดเพิ่มเติมที่ [wikipedia](#)

- ตาราง **GPT** ตารางพาร์ทิชันในอุปกรณ์เก็บรักษาข้อมูลของเครื่องคอมพิวเตอร์ปัจจุบัน ซึ่งว่า Globally Unique Identification Partition Table หรือ **GUIDPT** หรือย่อว่าตาราง **GPT** ทำหน้าที่คล้ายกับ MBR แต่รองรับจำนวนพาร์ทิชันได้มากขึ้น และรองรับมาตรฐานของ BIOS หรือเฟิร์มแวร์ของเครื่องคอมพิวเตอร์ ซึ่งว่า **Unified Extensible Firmware Interface**: UEFI รายละเอียดเพิ่มเติมเกี่ยวกับ GPT ที่ [wikipedia](#) ตาราง GPT สามารถรองรับระบบปฏิบัติชนิด 64 บิต ทำให้อุปกรณ์เก็บรักษาข้อมูลโดยเฉพาะสามารถจัดเก็บข้อมูลสูงสุดระดับเพ特้าไบต์ (Peta Byte) หรือ  $10^{15}$  ไบต์ หรือประมาณ 1000 เทอร่าไบต์

### 7.1.3 โครงสร้างของระบบไฟล์ยูนิกซ์ (Unix)

หน้าที่ของระบบไฟล์ คือ รองรับการจัดเก็บไฟล์ต่างๆ บันทึกประวัติการเข้าถึง (Access) ไฟล์ การบริหารสิทธิ์ (Permission) การเข้าถึงไฟล์/ไดเรกทอรีต่างๆ โดยผู้ใช้งานจำนวนมาก การใช้งานไฟล์ การลบไฟล์ การกู้คืนไฟล์ เป็นต้น ตามที่ผู้ใช้และซอฟต์แวร์ประยุกต์ร้องขอ

โครงสร้างของระบบไฟล์ในระบบปฏิบัติการตระกูลยูนิกซ์มีโครงสร้าง และหน้าที่ของระบบไฟล์เป็นขั้นการจัดการ 3 ขั้น โดยเรียงลำดับ ดังนี้

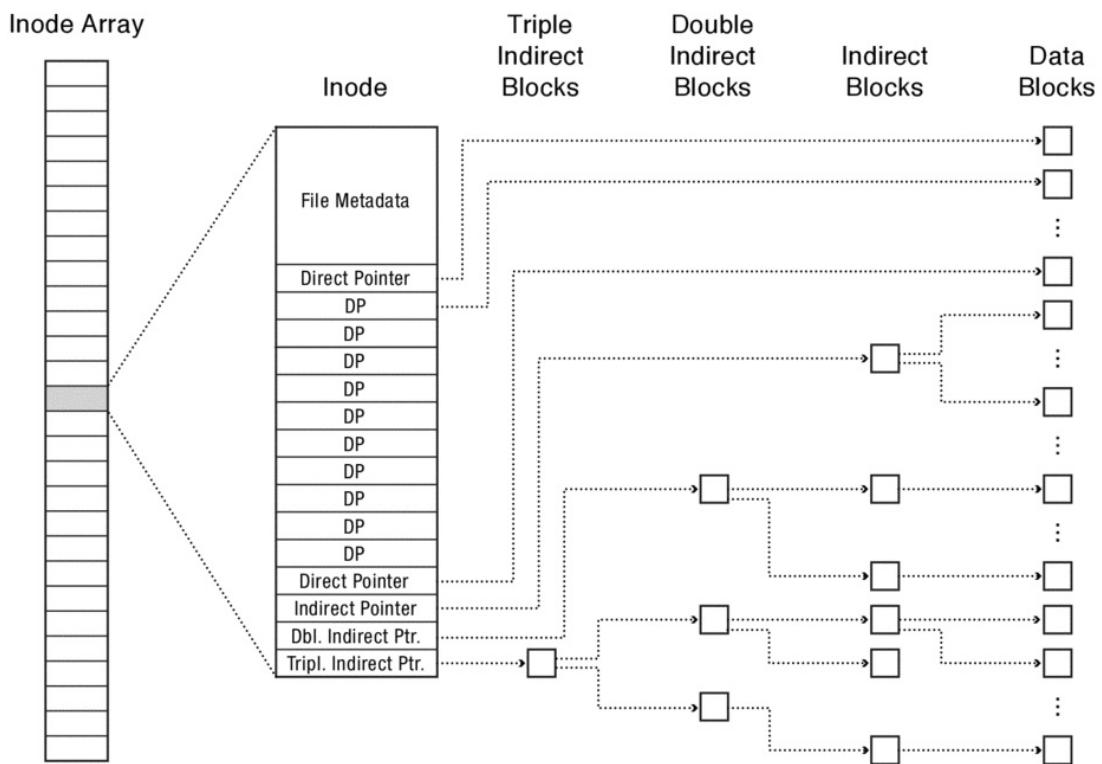
- ชั้นตรรกะ (Logical Layer) เป็นขั้นบนสุดทำหน้าที่เชื่อมต่อกับซอฟต์แวร์ประยุกต์ โดยใช้ฟังก์ชันเปิดไฟล์ ปิดไฟล์ อ่านไฟล์ เขียนไฟล์ เป็นต้น ซึ่งไฟล์ที่ซอฟต์แวร์ประยุกต์ต้องการใช้งาน โดยเนื้อหาในบทนี้จะเน้นที่การทำงานในชั้นตรรกะ และชั้นกายภาพ
- ชั้นเสมือน (Virtual Layer) ทำหน้าที่เชื่อมต่อระหว่างชั้นตรรกะและชั้นกายภาพ รายละเอียดขึ้นอยู่กับวิธีการพัฒนาโปรแกรมของแต่ละระบบปฏิบัติการ
- ชั้นกายภาพ (Physical Layer) เป็นชั้นล่างสุด ทำหน้าที่จัดการบล็อกข้อมูลบนอุปกรณ์เก็บรักษาข้อมูล แต่ละชนิด เพื่อตอบสนองต่อการร้องขอจากชั้นเสมือน ซึ่งกลไกสำคัญคือ การบริหารบัฟเฟอร์ (Buffer) ในหน่วยความจำกายภาพ การเข้าถึงหน่วยความจำกายภาพโดยตรง (DMA) การจัดวางตำแหน่งบล็อกข้อมูลบนอุปกรณ์เหล่านั้น เพื่อประสิทธิภาพการอ่านหรือเขียนได้ชี้ แล้ว การเชื่อมต่อกับไดไวซ์ไดเรกอร์ของอุปกรณ์เก็บรักษาข้อมูล

ผู้ใช้สามารถจัดแบ่งพาร์ทิชันตามความจุที่ต้องการเอง หรืออนุญาตให้ระบบปฏิบัติการจัดแบ่งอัตโนมัติ หลังจากนั้น ระบบปฏิบัติการจะทำการเขียนโครงสร้างของพาร์ทิชันตามที่ได้ออกแบบไว้ เรียกว่า การฟอร์แมทแต่ละพาร์ทิชัน (Format) ออกเป็นพื้นที่ต่างๆ ในแควที่สองจากบนสุดของรูปที่ 7.1 โดยหลักการ คือ พาร์ทิชันสามารถแบ่งเป็นบูตบล็อก (Boot Blocks) จำนวน 1 ชุด และไซลินเดอร์กรุ๊ป (Cylinder Group) จำนวนหนึ่งตามขนาดความจุของพาร์ทิชันนั้น โดยแต่ละไซลินเดอร์กรุ๊ปแบ่งเป็นพื้นที่ 3 ส่วนย่อยๆ ดังนี้

- ชูเปอร์บล็อก (Superblock) ทำหน้าที่เก็บรายละเอียดต่างๆ ของระบบไฟล์ ขนาดของบล็อกข้อมูล รายละเอียดการใช้งานบล็อกข้อมูลต่างๆ เช่น สถานะว่างหรือใช้งานอยู่ เป็นต้น
- ตารางไอลอนด์ (Inode Table) หรืออาจเรียกว่า ไอลอนด์อาร์เรย์ (Inode Array) หรือ ไอลอนด์ลิสต์ (Inode List) ทำหน้าที่เก็บโครงสร้างข้อมูล Inode จำนวนหนึ่งภายใต้ไซลินเดอร์กรุ๊ปนี้ รายละเอียดเพิ่มเติมในหัวข้อถัดไป
- บล็อกข้อมูลจำนวนหนึ่ง (Data Blocks) ทำหน้าที่บรรจุข้อมูลของไฟล์หนึ่งไฟล์ให้เรียงต่อกันไปจนครบขนาดไฟล์ โดยทั่วไปขนาดของบล็อกข้อมูลแต่ละบล็อกมีความจุเท่ากับ 4096 ไบต์ หรือ 4 KiB

(**kibibyte**) สำหรับระบบปฏิบัติการยูนิกซ์และอื่นๆ ทั้งนี้ขึ้นกับชนิดของเทคโนโลยีและระบบไฟล์ที่ใช้ เมื่อกำหนดขนาดความจุของแต่ละบล็อกแล้ว จำนวนบล็อกข้อมูลจะแบร์พันตามขนาดความจุของไฟล์เดียว ครุ่นน้ำ

#### 7.1.4 ไอโหนด (Inode)



รูปที่ 7.2: โครงสร้างของไอโหนดหนึ่งตัวและการเชื่อมโยงกับบล็อกข้อมูลของไฟล์หนึ่งไฟล์ ที่มา: [Anderson and Dahlin \(2012\)](#)

ในการติดตั้ง (Install) ระบบปฏิบัติการแต่ละชนิดบนอุปกรณ์เก็บรักษาข้อมูล ระบบไฟล์จะกำหนดจำนวนไอโหนดสูงสุด ตามขนาดหรือความจุของแต่ละพาร์ทิชัน ไอโหนด (Inode) คือ โครงสร้างข้อมูล (Data Structure) คล้ายรูปที่ 7.2 ไอโหนด 1 ตัวต้องการพื้นที่ประมาณ 128 ไบต์ ทำหน้าที่เป็นตัวแทนของไฟล์ หรือ ไดเรกทอรี (Directory) อย่างโดยย่างหนึ่ง ไอโหนดแต่ละไอโหนดมีหมายเลขกำกับประจำตัว (Inode Number: Inum)

ตารางไอโหนด (Inode Table หรือ Inode List) จะจัดเรียงไอโหนดทั้งหมด โดยเริ่มต้นจากไอโหนดหมายเลข 1 ด้านซ้ายของรูปที่ 7.2 ที่มา: [kernel.org](#) สำหรับเก็บรายละเอียดของไฟล์หรือไดเรกทอรี ดังนี้

- หมายเลขไอโหนด (Inode Number: Inum): หมายเลขประจำตัวของไฟล์หรือไดเรกทอรีนั้นๆ เป็นเลขจำนวนเต็มชนิดไม่มีเครื่องหมายความกว้าง 32 บิต สำหรับระบบไฟล์ Ext4 ใหม่ 32 บิต ที่มา: [kernel.org](#)
- รายละเอียดเสริมของไฟล์ (File Metadata) แบ่งเป็น
  - ชนิดไฟล์: ไฟล์ (file), ไดเรกทอรี (directory), ไปป์ (pipe) เป็นต้น

- \* **ไฟล์** ตามที่เคยนิยามที่ 3.3.1 ในหัวข้อที่ 3.3.6 ว่าเป็นการเรียงตัวกันของตัวเลขฐานสองที่ลงทะเบียนๆ โดยอาศัยพื้นที่จัดเก็บในอุปกรณ์รักษาข้อมูล เรียกว่า **บล็อกข้อมูล** ไฟล์เกิดจากการเรียงของบล็อกข้อมูลอย่างน้อย 1 บล็อกขึ้นไป เพื่อจัดเก็บข้อมูลหรือคำสั่งต่างๆ ลงในอุปกรณ์เก็บรักษาข้อมูล
  - \* **ไดเรกทอรี** หรือ **โฟลเดอร์** คือ ไฟล์ชนิดหนึ่งที่เก็บหมายเลขอ่อนดที่อยู่ภายใต้โครงสร้างนี้โดยไดเรกทอรีสามารถซ้อนกันได เพื่อความสะดวกในการจัดหมวดหมู่ของไฟล์และไดเรกทอรี
  - \* **ไบป์ (pipe)** คำราเล่นนี้ไม่ครอบคลุม ผู้อ่านสามารถค้นคว้าเพิ่มเติมได้จาก [Wikipedia](#)
  - หมายเลขผู้ใช้ (User ID) ซึ่งเป็นเจ้าของ
  - หมายเลขกลุ่ม (Group ID) ซึ่งเจ้าของเป็นสมาชิกอยู่
  - **สิทธิ์การเข้าถึง (Permission)** ไฟล์หรือไดเรกทอรี: คือ บิตควบคุมจำนวน 9 บิต แบ่งเป็น 3 ชุด คือ rwx rwx rwx อ่าน (r: read), เขียน (w: write) และรัน (x: execute) โดยบิตแต่ละบิตมีค่าລອຈິກ 1 คือ อนุญาต และค่าລອຈິກ 0 คือ ไม่อนุญาต แต่ละชุดเรียงลำดับจากซ้ายไปขวา ดังนี้
    - \* ชุด **Owner** คือ เจ้าของไฟล์หรือไดเรกทอรีซึ่งล็อกอินเข้าระบบมาแล้วสร้างไฟล์นี้ โดยชื่อเจ้าของไฟล์ คือ Username ที่ล็อกอิน รายละเอียดเพิ่มเติมในการทดลองที่ 12 ภาคผนวก L หัวข้อที่ L.2
    - \* ชุด **Group of Owner** คือ กลุ่มของเจ้าของไฟล์หรือไดเรกทอรีที่ Username นั้นเป็นสมาชิกอยู่
    - \* ชุด **Everyone** หมายถึง ผู้ใช้งานทุกคนที่สามารถเข้าถึงไฟล์หรือไดเรกทอรีนี้
  - ขนาดที่แท้จริงของไฟล์หรือไดเรกทอรี (หน่วยเป็นไบต์)
  - **บันทึกเวลา (Time stamp):** ประกอบด้วยบันทึกเวลาชนิดต่างๆ ดังนี้
    - \* **เวลาเข้าถึง (access time)** หรือเวลาที่อ่านไฟล์หรือไดเรกทอรี
    - \* **เวลาที่เปลี่ยนแปลง (modification time)** คือ เวลาที่ไฟล์หรือไดเรกทอรีถูกเปลี่ยนแปลง
    - \* **เวลาที่ไอโหนดถูกเปลี่ยนแปลง (change time)**
  - อื่นๆ
- พอยน์เตอร์เชื่อมโยงไปยังบล็อกข้อมูล (Pointers to Data Blocks) ทำหน้าที่เก็บหมายเลขอับล็อกข้อมูล (บล็อกสี่เหลี่ยมจัตุรัสทางด้านขวาของรูปที่ 7.2) ซึ่งทำหน้าที่เก็บข้อมูลจริงๆ โดยแต่ละบล็อกข้อมูลในระบบไฟล์มีขนาดเท่ากับ 4 KiB (kibibyte) ไฟล์ 1 ไฟล์ หรือไดเรกทอรี 1 ไดเรกทอรีเกิดจากการเชื่อมโยงบล็อกข้อมูลหลายๆ บล็อกเข้าด้วยกัน พอยน์เตอร์เชื่อมโยงแบ่งเป็น 2 ชนิดหลัก คือ
- พอยน์เตอร์บล็อกข้อมูลทางตรง (Direct Pointers: DP) ในรูปที่ 7.2 ไอโหนดหนึ่งตัวมีจำนวน DP เท่ากับ 12 หมายเลขบล็อกข้อมูล ทำให้สามารถรองรับไฟล์ขนาดเล็กที่มีขนาดไม่เกิน 48 KiB (kibibyte)
  - พอยน์เตอร์บล็อกข้อมูลทางอ้อม ใช้สำหรับกรณีที่ไฟล์มีขนาดใหญ่กว่า 48 KiB (kibibyte) จนถึงหลายกิกะไบต์ (GiB) ระบบไฟล์สามารถเก็บข้อมูลไฟล์โดยใช้จำนวนบล็อกข้อมูลมากขึ้นตามขนาดไฟล์ พอยน์เตอร์บล็อกข้อมูลทางอ้อมจะทำหน้าที่เก็บหมายเลขอับล็อกข้อมูลจำนวนมากๆ เพื่อรองรับไฟล์ที่มีขนาดใหญ่ขึ้นดังกล่าว พอยน์เตอร์บล็อกข้อมูลทางอ้อมแบ่งเป็น 3 ระดับ ดังนี้

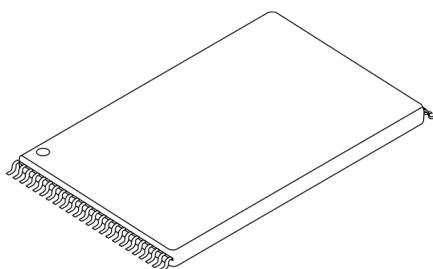
- \* พอยน์เตอร์หนึ่งชั้น (Indirect Pointer) เมื่อไฟล์มีขนาดใหญ่ขึ้นอีกจนเกิน 48 KiB ทำให้จำนำวนพอยน์เตอร์ทางตรงไม่เพียงพอ ระบบไฟล์จะอาศัยพอยน์เตอร์หนึ่งชั้น (Indirect Pointer) ในโครงสร้างไอโอนด เพื่อเก็บหมายเลขล็อกพิเศษ เรียกว่า บล็อกทางอ้อมหนึ่งชั้น (Indirect Block) ขนาด 4096 ไบต์ บล็อกทางอ้อมหนึ่งชั้นทำหน้าที่เก็บหมายเลขล็อกข้อมูลที่  $13=12+1$  จนถึงหมายเลขล็อกที่  $1036=12+1024$  โปรดสังเกตบล็อกสี่เหลี่ยมจัตุรัสที่ขึ้นระหว่างช่องพอยน์เตอร์หนึ่งชั้น (Indirect Pointer) และบล็อกข้อมูลทางขวาของรูปที่ 7.2 หมายเหตุ สมมติว่าหมายเลขล็อก 1 หมายเลขต้องการพื้นที่ขนาด 4 ไบต์
- \* พอยน์เตอร์สองชั้น (Double Indirect Pointer) เมื่อไฟล์มีขนาดใหญ่ขึ้นอีกจนพอยน์เตอร์หนึ่งชั้นไม่เพียงพอ ระบบไฟล์จะอาศัยพอยน์เตอร์สองชั้นในโครงสร้างไอโอนด เพื่อเก็บหมายเลขล็อกพิเศษ เรียกว่า บล็อกทางอ้อมสองชั้น (Double Indirect Blocks) ขนาด 4096 ไบต์ บล็อกทางอ้อมสองชั้นทำหน้าที่เก็บหมายเลขล็อกทางอ้อมหนึ่งชั้น เพื่อให้บล็อกทางอ้อมหนึ่งชั้นทำหน้าที่เก็บหมายเลขหรือพอยน์เตอร์บล็อกข้อมูลที่  $1037=12+1024+1$  เป็นต้นไป โปรดสังเกตบล็อกสี่เหลี่ยมจัตุรัสที่อยู่ในคอลัมน์ Double Indirect Blocks ของรูปที่ 7.2
- \* พอยน์เตอร์สามชั้น (Triple Indirect Pointer) เมื่อไฟล์มีขนาดใหญ่ขึ้นอีกจนพอยน์เตอร์สองชั้นไม่เพียงพอ ระบบไฟล์จะอาศัยพอยน์เตอร์สามชั้นในโครงสร้างไอโอนด เพื่อเก็บหมายเลขบล็อกพิเศษจำนวนหนึ่ง เรียกว่า บล็อกทางอ้อมสามชั้น (Triple Indirect Blocks) ทำหน้าที่เก็บหมายเลขบล็อกทางอ้อมสองชั้น โปรดสังเกตบล็อกสี่เหลี่ยมจัตุรัสที่อยู่ในคอลัมน์ Triple Indirect Blocks ของรูปที่ 7.2

ตัวอย่างเช่น ไฟล์ชื่อ Kernel ตำแหน่งสองแคลร่างสุดของรูปที่ 7.1 ตรงกับไอโอนดหมายเลข (Inum) = 2 และเป็นหมายเลขอ้างอิงที่ไม่ซ้ำกัน ซึ่งข้อมูลภายในไฟล์นี้จะบันทึกอยู่ในบล็อกข้อมูลจำนวน 3 บล็อก เป็นลิงก์ทางตรง (DP: Direct Pointer) ไปยังบล็อกข้อมูลหมายเลข 3, 7, 9 ส่วนชื่อไฟล์ Kernel จะเก็บบันทึกในไดเรกทอรีที่ไฟล์นี้ตั้งอยู่ ซึ่งใช้หมายเลขไอโอนด (Inum) = 2 เป็นหมายเลขอ้างอิงที่ได้กล่าวไว้ก่อนหน้า ส่วนรายละเอียดเพิ่มเติมสามารถค้นจากส่วนที่เรียกว่า **รายละเอียดเสริมของไฟล์** ภายใต้โครงสร้างของไอโอนดในรูปที่ 7.2

หัวข้อถัดไปจะอธิบายการทำงานของเทคโนโลยีหน่วยความจำชนิดต่างๆ โดยเริ่มจากหน่วยความจำแฟลชการ์ดหน่วยความจำ SD โซลิดสเตทไดรฟ์ และฮาร์ดดิสก์ไดรฟ์ ตามลำดับ

## 7.2 ชิปหน่วยความจำแฟลช (Flash Memory)

หน่วยความจำแฟลช เดิมเรียกว่า **แฟลชROM** (Flash ROM) คำว่า ROM ย่อมาจากคำว่า Read-Only Memory วัตถุประสงค์เดิมของแฟลชROM คือ ทำหน้าที่เป็นหน่วยความจำที่สร้างจากเทคโนโลยีสารัชนาการที่ตัวนำสำหรับใช้เก็บคำสั่งและข้อมูลเพื่อการอ่านเป็นหลัก นักพัฒนาระบบจึงประยุกต์ใช้แฟลชROMสำหรับเก็บคำสั่งประจำเครื่อง หรือ **เฟิร์มแวร์ (Firmware)** หรือ  **BIOS** (Basic Input Output System) เป็นหลัก ด้วยความนิยมในเทคโนโลยีชนิดนี้ หน่วยความจำแฟลชจึงถูกพัฒนาอย่างต่อเนื่อง จนสามารถใช้แทนหรือแก้ไขข้อมูลภายในแฟลชROMได้รวดเร็วขึ้น และกลายเป็นอุปกรณ์เก็บรักษาข้อมูลในปัจจุบัน เพราะแฟลชROM มีจุดเด่น คือ ส่วนประกอบการทำงานน้อยกว่า น้ำหนักเบากว่า ปริมาตรที่เล็กกว่า ความเร็วในการอ่านหรือเขียนข้อมูลได้รวดเร็วกว่า ความจุที่เพิ่มมากขึ้น และสามารถเก็บข้อมูลได้ยาวนานขึ้น



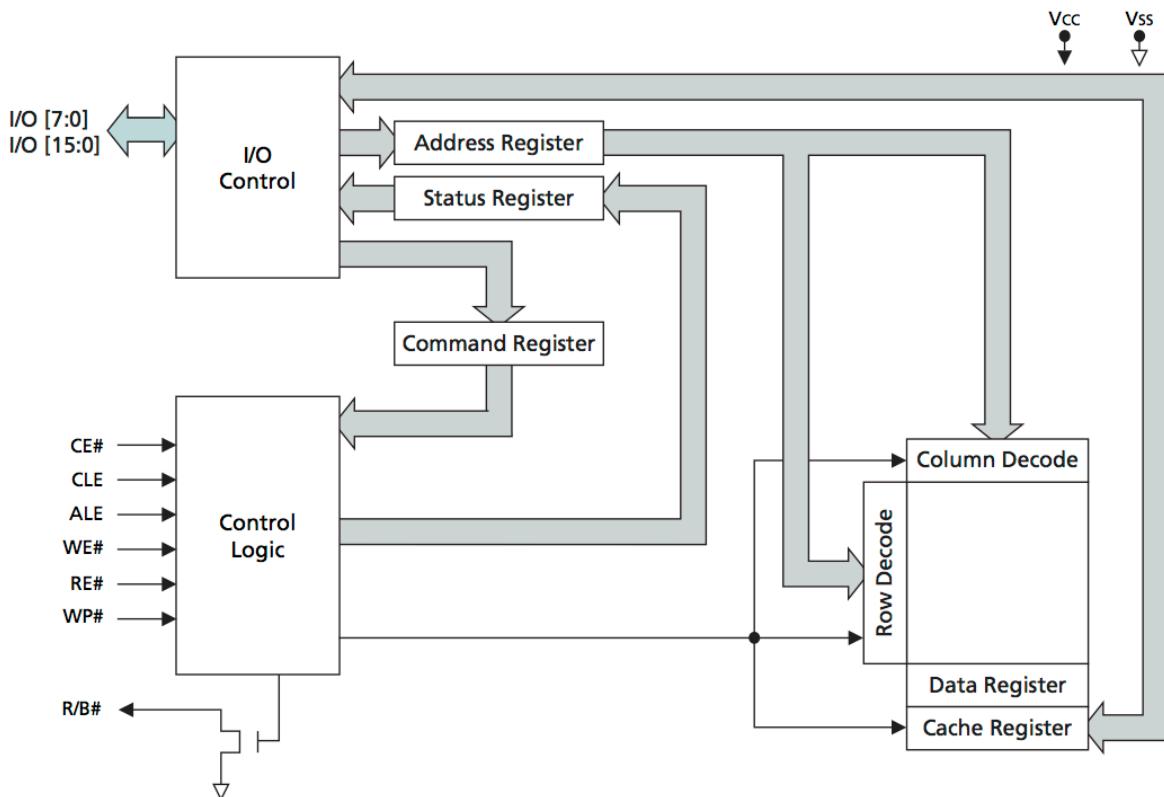
รูปที่ 7.3: ชิปหน่วยความจำแฟลช NAND ผลิตโดยบริษัท Micron Technology โดยใช้ตัวถังชนิด TSOP (Thin Small Outline Package) จำนวน 48 ขา ที่มา: [Micron Technology \(2004\)](#)

แฟลชROMกรณีศึกษาของตราเล่มนี้ เป็น หน่วยความจำแฟลชนิด NAND ผลิตโดยบริษัท Micron Technology ในปี ค.ศ. 2004 ใช้ตัวถังชนิด TSOP (Thin Small Outline Package) ในรูปที่ 7.3 ตัวชิปมีการผลิตออกมากด้วยความจุที่แตกต่างกัน ขึ้นอยู่กับ จำนวนสาย จำนวนบล็อกข้อมูล และจำนวนสายต่อชิป โดยความจุข้อมูลต่อ 1 ดาyreimตันที่ ขนาด 2 กิกะบิต หรือ  $2 \times 2^{30}$  บิต ซึ่งประกอบด้วยบล็อก (Block) ข้อมูลจำนวน 2048 ( $2 \times 2^{10}$ ) บล็อก แต่ละบล็อกประกอบด้วย 64 ( $2^6$ ) เพจ แต่ละเพจ (Page) มีความจุ 2048 ( $2 \times 2^{10}$ ) ไบต์ โดย 1 ไบต์ เท่ากับ  $2^3$  บิต รวมทั้งหมดคิดเป็นจำนวน  $2^{11} \times 2^6 \times 2^{11} \times 2^3$  เท่ากับ  $2^{31}$  บิต

หน่วยความจำแฟลช NAND จะมีประสิทธิภาพโดยวัดจากเวลาเข้าถึง (Access Time) และความจุต่อชิปเพิ่มสูงขึ้นเรื่อยๆ เมื่อเทคโนโลยีพัฒนาไปเรื่อยๆ โดยที่ไปประสิทธิภาพการอ่านข้อมูลจะดีกว่าการเขียนประสิทธิภาพการเขียนจะดีกว่าการลบตามลำดับ โดยอายุการใช้งานซึ่งนับจากจำนวนการลบข้อมูล (Erase) ดังนี้

- ประสิทธิภาพการอ่านข้อมูลขึ้นอยู่กับตำแหน่งและรูปแบบการอ่านและรูปแบบการเรียงตัวของข้อมูล เช่น การอ่านข้อมูลที่เรียงตัวต่อเนื่อง (Sequential Address) จะใช้เวลาเข้าถึง สั้นมากเพียง 30 นาโนวินาที การอ่านข้อมูลที่ไม่เรียงตัวต่อเนื่องและสุ่มตำแหน่ง (Random Address) จะใช้เวลาเข้าถึงนานขึ้นเป็น 25 ไมโครวินาที จะเห็นได้ว่าใช้เวลาเพิ่มขึ้นเกือบ 1,000 เท่า เทียบกับการอ่านข้อมูลแบบเรียงตัวต่อเนื่อง
- ประสิทธิภาพการเขียนข้อมูลมีลักษณะเช่นเดียวกับการอ่านข้อมูล โดย การเขียนข้อมูลต้องเขียนครั้งละ เพจ (Page Program) หรือ 2048 ไบต์ ซึ่งจะใช้เวลาเข้าถึงยาวนานกว่าการอ่านข้อมูลเป็น 300 ไมโครวินาที ใช้เวลาเพิ่มขึ้นเป็น 10 เท่า เทียบกับการอ่านข้อมูลแบบสุ่ม
- ประสิทธิภาพการลบข้อมูล (Erase) จะใช้เวลาเพิ่มขึ้นสูงมากถึง 2 มิลลิวินาที จะเห็นได้ว่าใช้เวลาเพิ่มขึ้นเกือบ 100 เท่าเทียบกับการเขียนข้อมูลจำนวนหนึ่งเพจ

- อายุการใช้งานของชิป จะนับจากจำนวนครั้งที่เขียนข้อมูล หรือ จำนวนครั้งที่ลบข้อมูลประมาณ 100,000 ครั้งเท่านั้น แต่ชิปสามารถรักษาข้อมูลได้เป็นระยะเวลานานถึง 10 ปีตามที่แจ้งไว้ในเอกสารคุณสมบัติ



รูปที่ 7.4: โครงสร้างภายในชิปหน่วยความจำแฟลช NAND ที่มา: [Micron Technology \(2004\)](#)

รูปที่ 7.4 แสดงโครงสร้างภายในหน่วยความจำแฟลช NAND ประกอบด้วย

- อาร์เรย์ของเซลล์หน่วยความจำทางด้านขวาล่าง
- วงจรควบคุม (Control Logic) การทำงานโดยรวม และ
- วงจรควบคุมอินพุตและเอาต์พุต (I/O Control) เพื่อเชื่อมต่อ กับวงจรภายนอกทั่วชิป

โดยตัวชิปหน่วยความจำแฟลชมีขาสัญญาณควบคุม ขาสัญญาณแอ็ตเตอร์เน็ตและขาสัญญาณข้อมูล ดังนี้

- ขา **CE#** หรือ  $\overline{CE}$  คือ สัญญาณ Chip Enable bar ทำงานเมื่อได้รับลอจิก 0
- ขา **CLE** คือ สัญญาณ Command Latch Enable ทำงานเมื่อได้รับลอจิก 1
- ขา **ALE** คือ สัญญาณ Address Latch Enable ทำงานเมื่อได้รับลอจิก 1
- ขา **WE#** หรือ  $\overline{WE}$  คือ สัญญาณ Write Enable bar ทำงานเมื่อได้รับลอจิก 0
- ขา **RE#** หรือ  $\overline{RE}$  คือ สัญญาณ Read Enable bar ทำงานเมื่อได้รับลอจิก 0
- ขา **WP#** หรือ  $\overline{WP}$  คือ สัญญาณ Write Protect bar ทำงานเมื่อได้รับลอจิก 0
- ขา **R/B#** หรือ  $\overline{R/B}$  คือ สถานะ Ready หากมีค่าเท่ากับลอจิก 1 หรือ Busy bar หากมีค่าเท่ากับลอจิก 0

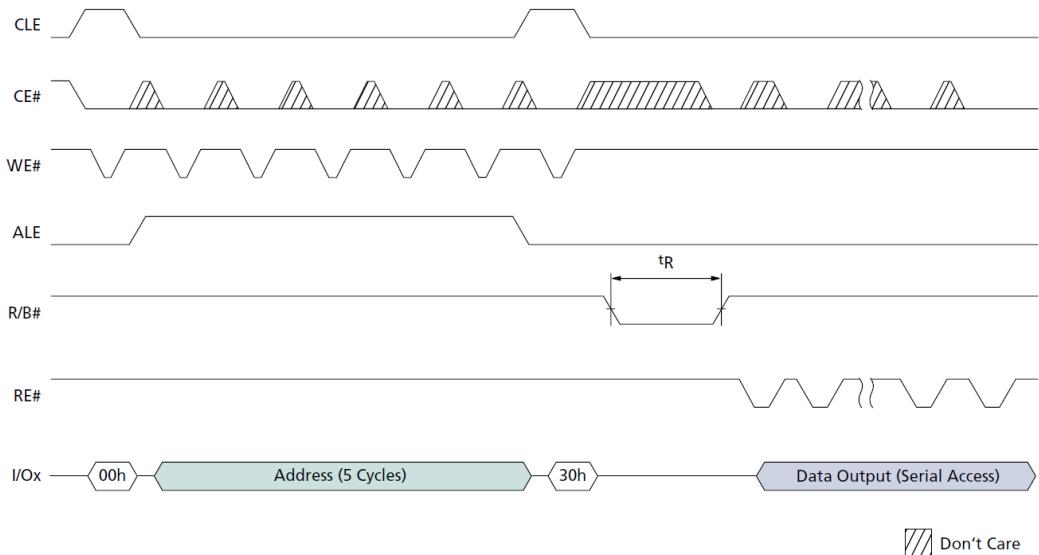
- **ขาสัญญาณ I/O[0:15]** ทั้งหมด 16 ขา มีหน้าที่มัลติเพล็กซ์ (Multiplex) หรือใช้งานร่วมกันคนละช่วงเวลา เพื่อการรับแอดเดรส รับ/ส่งข้อมูล ส่งสถานะและรับคำสั่งไปยังตัวชิป
  - **สัญญาณแอดเดรส** จะถูกเก็บพักในรีจิสเตอร์แอดเดรส (Address Register) เพื่อนำไปถอดรหัส (Decode) โดยวิจารณ์ Column Decoder เพื่อสร้างสัญญาณบิตไลน์ (Bit Line) และ Row Decoder เพื่อสร้างสัญญาณเวิร์ดไลน์ (Word Line) ในรูปที่ 7.1
  - **สัญญาณข้อมูลที่อ่านหรือเขียน** จะถูกเก็บพักในรีจิสเตอร์แคช (Cache Register) ชั่วคราว แล้วจึงย้ายเข้าหรือออกจากรีจิสเตอร์ข้อมูล (Data Register)
  - **สถานะของการทำงานของแฟลชจะเก็บพักในรีจิสเตอร์สถานะ (Status Register)** เพื่อให้ง่ายต่อ (Interface Circuit) รับทราบ ได้แก่ สถานะ เป็นต้น
  - **คำสั่งจะเก็บพักในรีจิสเตอร์คำสั่ง (Command Register)** เพื่อป้อนให้กับวงจรควบคุม ได้แก่
    - \* คำสั่งอ่านข้อมูลเพจ (Page Read)
    - \* คำสั่งอ่านข้อมูลสุ่ม (Random Data Read)
    - \* คำสั่งเขียนข้อมูลในเพจ (Program Page)
    - \* คำสั่งลบ (Block Erase) เป็นต้น

ผู้อ่านจะสังเกตได้ว่าโครงสร้างภายในชิปแฟลช NAND มีลักษณะคล้ายกับหน่วยความจำ SRAM ในรูปที่ 5.13 และของ SDRAM ในรูปที่ 5.17 คือ มีเซลล์หน่วยความจำเรียงต่อกันเป็นอาร์เรย์ และวงจรควบคุมการอ่านหรือเขียนข้อมูลไปในอาร์เรย์นี้ การเข้าถึงเซลล์หน่วยความจำอาศัยแอดเดรสແກ้า (เลขແກ้า) และแอดเดรสคอลัมน์ (เลขคอลัมน์) เพื่อบ่งชี้ตำแหน่งเซลล์นั้นๆ ในอาร์เรย์ แต่การทำงานของหน่วยความจำแฟลชมีความซับซ้อน เนื่องจากมีการอ่านหลายชนิด การเขียนหลายชนิดและการลบข้อมูลตามที่กล่าวไปก่อนหน้า ที่มา: [Micron Technology \(2004\)](#) ตำราเล่มนี้จะอธิบายการอ่านข้อมูลแบบเพจและการเขียนข้อมูลแบบเพจพอสังเขป เนื่องจากเป็นการใช้งานหน่วยความจำแฟลชที่เกิดขึ้นบ่อยที่สุด ลำดับสุดท้ายจะเป็นการเปรียบเทียบที่หน่วยความจำแฟลชนิดอื่นๆ ที่สำคัญ

### 7.2.1 การอ่านข้อมูลแบบเพจ (Page Read)

การอ่านข้อมูลทั้งเพจ (Page Read) ซึ่งมีประสิทธิภาพสูงกว่าการอ่านแบบสุ่ม (Random Read) เริ่มต้นโดยการเปลี่ยนสัญญาณเหล่านี้ แกนเวลาอยู่ในแนวโน้มในรูปที่ 7.5 โดยแกนเวลาเริ่มต้นจากทางซ้ายไปทางขวา

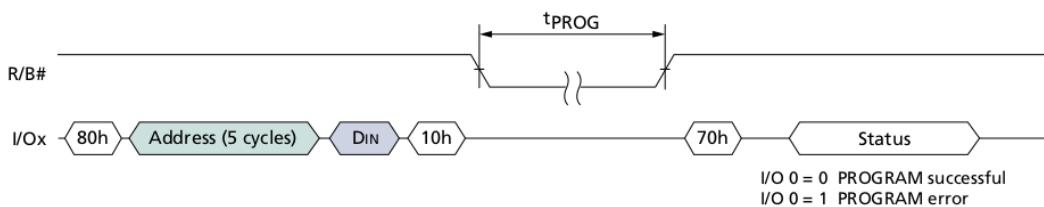
- **สัญญาณ CLE** เป็นสัญญาณที่เปลี่ยนจาก 0 เป็น 1 เพื่ออ่านคำสั่งที่ปรากฏบนบัส I/O โดยคำสั่งจะมีรูปแบบ คำสั่ง 00h-Address (5 Cycles)-30h คือ การอ่านข้อมูลทั้งเพจ ณ แอดเดรสที่ได้รับ
- **สัญญาณ CE#** หรือ  $\overline{CE}$  เป็นสัญญาณที่เปลี่ยนจาก 1 เป็น 0 เพื่อส่งชิปให้เริ่มต้นทำงาน
- **สัญญาณ WE#** หรือ  $\overline{WE}$  จะมีการเปลี่ยนแปลงจาก 1 เป็น 0 ลักษณะเป็น 0 ตลอดเวลา เพื่อเขียนคำสั่งและแอดเดรสทางขา I/O ไปเก็บพักในรีจิสเตอร์ต่างๆ
- **สัญญาณ ALE** จะเปลี่ยนแปลงจาก 0 เป็น 1 ในระหว่างที่บัส I/O รับแอดเดรสเป็นระยะเวลา 5 คากาบเวลา (Cycles)
- **สัญญาณ R/B#** หรือ  $R/\overline{B}$  จะเปลี่ยนแปลงจาก 1 เป็น 0 ระยะเวลาหนึ่ง ( $t_R$ ) เพื่อบ่งบอกว่าชิป มีสถานะยุ่ง (Busy) แล้วกลับไปเป็น 1 เพื่อบ่งบอกว่าชิปพร้อม (Ready) และ



รูปที่ 7.5: ไดอะแกรมเวลาของการอ่านข้อมูลแบบเพจ (Page Read) ของหน่วยความจำแฟลช NAND ที่มา: Micron Technology (2004)

- สัญญาณ **RE#** หรือ  $\overline{RE}$  จะเปลี่ยนแปลงจาก 1 เป็น 0 สลับไปมา เมื่อตรวจพบว่าสัญญาณ **R/B#** เปลี่ยนจาก 0 เป็น 1 เพื่ออ่านข้อมูลทางขา **I/O** โดยใช้ขอบขาขึ้นหรือขอบขาลงของสัญญาณ **RE#**
- สัญญาณ **I/O[0:15]** ทั้งหมด 16 ขา จะเปลี่ยนแปลงตามลำดับดังนี้
  - คำสั่ง **00h** ( $00_{16}$ ) จำนวน 1 คิบเวลา
  - แอดเดรส จำนวน 5 คิบเวลา
  - คำสั่ง **30h** ( $30_{16}$ ) จำนวน 1 คิบเวลา
  - ข้อมูลที่ตรงกับแอดเดรสจะปรากฏขึ้น เมื่อรอเป็นระยะเวลา  $t_R$  จำนวนหลายคิบเวลาตามขนาดของเพจข้อมูล โดยการอ่านข้อมูลเรียงตามลำดับหมายเลขแอดเดรส (Serial Access)

## 7.2.2 การเขียนข้อมูล (Program Data)



รูปที่ 7.6: ไดอะแกรมเวลาของการเขียนข้อมูล (Program Data) และอ่านสถานะ (Read Status) ของหน่วยความจำแฟลช NAND ที่มา: [Micron Technology \(2004\)](#)

การเขียนข้อมูลขนาดเล็ก (Program Data) และอ่านสถานะ (Read Status) เริ่มต้นโดยการเปลี่ยนสัญญาณเหล่านี้ แกนเวลาอยู่ในแนวนอนในรูปที่ 7.6 โดยแกนเวลาเริ่มต้นจากทางซ้ายไปทางขวา

- **สัญญาณ CLE** เปลี่ยนจาก 0 เป็น 1 เพื่ออ่านค่าคำสั่งที่ปรากฏบนบัส I/O โดยคำสั่งจะมีรูปแบบ คำสั่ง 80h-Address (5 Cycles)-D<sub>IN</sub>-10h คือ การเขียนข้อมูลขนาดเล็ก ณ แอดเดรสที่ต้องการ
- **สัญญาณ CE#** หรือ  $\overline{CE}$  เปลี่ยนจาก 1 เป็น 0 เพื่อสั่งซิปให้เริ่มต้นทำงาน
- **สัญญาณ WE#** หรือ  $\overline{WE}$  จะมีการเปลี่ยนแปลงจาก 1 เป็น 0 สถาปนาไว้เพื่อเขียนคำสั่งและแอดเดรสทางขวา I/O ไปเก็บพักในรีจิสเตอร์ต่างๆ
- **สัญญาณ ALE** จะเปลี่ยนแปลงจาก 0 เป็น 1 ในระหว่างที่บัส I/O รับแอดเดรสเป็นระยะเวลา 5 คลาบเวลา (Cycles) คล้ายกับการอ่านข้อมูล
- **สัญญาณ I/O[0:15]** ทั้งหมด 16 ขา จะเปลี่ยนแปลงตามลำดับดังนี้
  - คำสั่ง 80h ( $80_{16}$ ) จำนวน 1 คลาบเวลา
  - แอดเดรส จำนวน 5 คลาบเวลา
  - ข้อมูลที่ต้องการเขียน ( $D_{IN}$ ) จำนวน 1 คลาบเวลา
  - คำสั่ง 10h ( $10_{16}$ ) จำนวน 1 คลาบเวลา
- **สัญญาณ R/B#** หรือ  $R/\overline{B}$  เปลี่ยนจาก 1 เป็น 0 และค้างไว้เป็นระยะเวลาหนึ่ง แล้วจึงเปลี่ยนจาก 0 เป็น 1 เพื่อบ่งบอกว่าซิป มีสถานะยุ่ง (Busy) แล้วกลับไปเป็นสถานะพร้อม (Ready=1) เพื่อป้องบวกว่าสิ้นสุดการเขียนข้อมูลและพร้อมจะรับคำสั่งถัดไป
- คำสั่ง 70h ( $70_{16}$ ) จำนวน 1 คลาบเวลา เพื่ออ่านสถานะของการเขียน
- **สัญญาณ Status** จากวงจร Status Register เพื่อให้ชี้พิจารณาสถานะบิตที่ 0 หากมีค่าเท่ากับ
  - \* 1 แสดงว่าการเขียนสำเร็จ
  - \* 0 แสดงว่าการเขียนยังไม่สำเร็จ

### 7.2.3 หน่วยความจำแฟลชชนิดอื่นๆ

ตารางที่ 7.1: ตารางเปรียบเทียบเซลล์หน่วยความจำแฟลชชนิด NAND (ซ้าย) และ NOR (ขวา) ตามโครงสร้าง และผังการจัดวางของเซลล์หน่วยความจำ ที่มา: Choi (2010) หมายเหตุ F คือ ความกว้างของฟล็อตเกต (Float Gate) มีหน่วยเป็น นาโนเมตร

	NAND	NOR
Cell Array & Size	 Unit Cell	 Unit Cell
Cross-section		
Features	<b>Small Cell Size, High Density Low Power &amp; Good Endurance → Mass Storage</b>	<b>Large Cell Current, Fast Random Access → Code Storage</b>

ตารางที่ 7.1 แสดงการเปรียบเทียบเซลล์หน่วยความจำแฟลช ชนิด NAND (ซ้าย) และ NOR (ขวา) เชิงโครงสร้าง ภาพแนวตัดขวาง (Cross Section) ผังการจัดวางหรือเลย์เอาต์ (Lay Out) และขนาดของเซลล์ (Cell Size) ผู้อ่านจะเห็นได้ว่าอาร์เรย์ของเซลล์หน่วยความจำ (Cell Array) มีโครงสร้างการเข้ามต่อของเซลล์ต่างๆ เข้าด้วยกัน โดยใช้จุดตัดกันของสายบิตไลน์ (Bit line) และสายเวิร์ดไลน์ (Word line) ระบุตำแหน่งของเซลล์ที่ต้องการอ่านหรือเขียนคล้ายกับหน่วยความจำ SRAM และ SDRAM ในรูปที่ 5.13 และรูปที่ 5.17 ตามลำดับ การเข้ามต่อของเซลล์หน่วยความจำมีลักษณะที่แตกต่างกัน คือ

- แฟลชชนิด NAND ต่อเซลล์เข้าด้วยกันแบบอนุกรม คล้ายกับวงจรเกท NAND ทำให้ประหยัดพื้นที่บนแผ่นซิลิโคนเท่ากับ  $5F^2$  ต่อหนึ่งเซลล์ จึงทำให้ต้นทุนในการผลิตต่ำกว่าแฟลชชนิด AND และ NOR แต่ต้องใช้เวลาอ่านหรือเขียนหลายเซลล์ตามลำดับ
- แฟลชชนิด NOR ต่อเซลล์เข้าด้วยแบบขนาน คล้ายกับวงจรเกท NOR โดยใช้สายเวิร์ดไลน์แยกกัน การต่อเซลล์แบบขนาน สามารถอ่านและเขียนข้อมูลแต่ละเซลล์ได้อิสระทีละเซลล์ แต่ใช้พื้นที่บนแผ่นซิลิโคนเพิ่มเป็น  $10F^2$  ต่อเซลล์ หรือเป็น 2 เท่าของแฟลช NAND

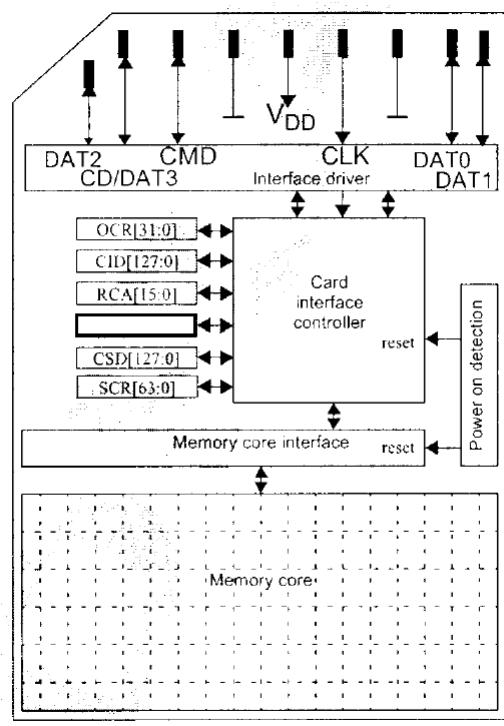
ขนาดของพื้นที่ผังการจัดวางหรือเลย์เอาต์ (Layout) และภาพตัดขวาง (Cross Section) ของแต่ละเซลล์ สามารถคำนวณตามความกว้างคุณความยาวของแต่ละเซลล์ โดย F คือ ความกว้างของชาฟล็อตเกต (Float Gate) ของทรานซิสเตอร์ ที่สามารถผลิตได้ซึ่งมีขนาดเด็กลงเรื่อยๆ จาก 32 นาโนเมตรเป็น 12 นาโนเมตร ที่มา: [wikipedia](#)

โครงสร้างภายในของแต่ละเซลล์ในแฟลชรวมได้ถูกพัฒนาให้เป็นเซลล์ชนิด TLC (Triple Level Cell) และ QLC (Quad Level Cell) ในปัจจุบัน เพื่อเพิ่มความจุหรือจำนวนบิตต่อเซลล์ให้มากขึ้น โดยโครงสร้างชนิด TLC

และ QLC สามารถเพิ่มความจุเป็นสามเท่า (Triple) และสี่เท่า (Quad) โดยสามารถเก็บข้อมูลได้จำนวน 3 และ 4 บิตต่อ 1 เซลล์ข้อมูล ตามลำดับ ผู้อ่านสามารถค้นคว้ารายละเอียดเพิ่มเติมได้ที่ [embedded-computing.com](http://embedded-computing.com) และ [wikipedia](https://en.wikipedia.org)

### 7.3 การ์ดหน่วยความจำ SD (Secure Digital)

เนื้อหาในหัวข้อนี้จะเสริมเพิ่มเติมหัวข้อที่ 3.1.4 ให้ละเอียดมากขึ้น เนื่องจากโครงสร้างภายในของการ์ดหน่วยความจำ SD เป็นหน่วยความจำแฟลชชนิด NAND เป็นหลัก โครงสร้างภายในของการ์ดหน่วยความจำ SD ในรูปที่ 3.8 ประกอบด้วย ชิปหน่วยความจำแฟลชเป็นองค์ประกอบสำคัญ ที่มา: [wikipedia](https://en.wikipedia.org) การ์ดหน่วยความจำจะทำงานแบบซิงโครนัส (Synchronous) ตามสัญญาณคล็อก (CLK) ที่วงจรควบคุมส่งมา ซึ่งมักจะมีความถี่สูงสุดเป็นจำนวนเท่าของ 25 เมกะเฮิรตซ์ เช่น 50 เมกะเฮิรตซ์ 100 เมกะเฮิรตซ์ เป็นต้น



รูปที่ 7.7: โครงสร้างภายในของการ์ดหน่วยความจำ SD ที่มา: [SanDisk Corporation \(2003\)](http://www.sandisk.com)

โครงสร้างภายในของการ์ดหน่วยความจำ SD ในรูปที่ 7.7 ประกอบด้วย

- อาร์เรย์เซลล์หน่วยความจำชนิดแฟลช ตามขนาดของความจุที่ต้องการใช้
- วงจรเชื่อมต่อกับแกนหน่วยความจำ (Memory Core Interface) เพื่อควบคุมการทำงานหน่วยความจำแฟลชที่อยู่ภายในการ์ด
- วงจรควบคุม (Card Interface Controller) เพื่อเชื่อมต่อกับไอซ์พีชี โดยทำงานร่วมกับรีจิสเตอร์ต่างๆ เหล่านี้
  - รีจิสเตอร์ CID[27:0] (Card Identification) ขนาด 28 บิต เพื่อเก็บหมายเลขประจำการ์ด

- รีจิสเตอร์ OCR[31:0] (Operation Condition Register) ขนาด 32 บิต เพื่อเก็บสถานะการทำงานของкар์ด
- รีจิสเตอร์ RCA[15:0] (Relative Card Address) ขนาด 32 บิต เพื่อเก็บหมายเลขแอดเดรสของ การ์ดซึ่งจะเปลี่ยนแปลงระหว่างที่ถอดเข้าออกจากระบบ
- รีจิสเตอร์ CSD[27:0] (Card Specific Data) ขนาด 28 บิต เพื่อเก็บข้อมูลจำเพาะของการ์ด
- รีจิสเตอร์ SCR[63:0] (SD Configuration Register) ขนาด 64 บิต เพื่อเก็บการตั้งค่าพิเศษประจำตัวการ์ด

การเชื่อมต่อ CPU กับการ์ดหน่วยความจำ SD นี้สามารถเลือกได้โดยใช้การเชื่อมต่อโหมด SPI และโหมด SDIO การเชื่อมตอกับการ์ดโหมด SDIO สามารถทำได้อีกสองโหมดด้วยกัน คือ การเชื่อมต่อโหมด SD 1 บิต และ การเชื่อมต่อโหมด SD 4 บิต ซึ่งมีประสิทธิภาพเร็วที่สุดเนื่องจากสามารถอ่าน/เขียนได้พร้อมกัน 4 บิต ซึ่งมีรายละเอียดของขาต่างๆ ในตารางที่ 7.2 ดังนี้

ตารางที่ 7.2: หมายเลขขา ชื่อ และวัตถุประสงค์ของสัญญาณในโหมด SD 4 บิต ที่มา: [wikipedia.org](https://en.wikipedia.org)

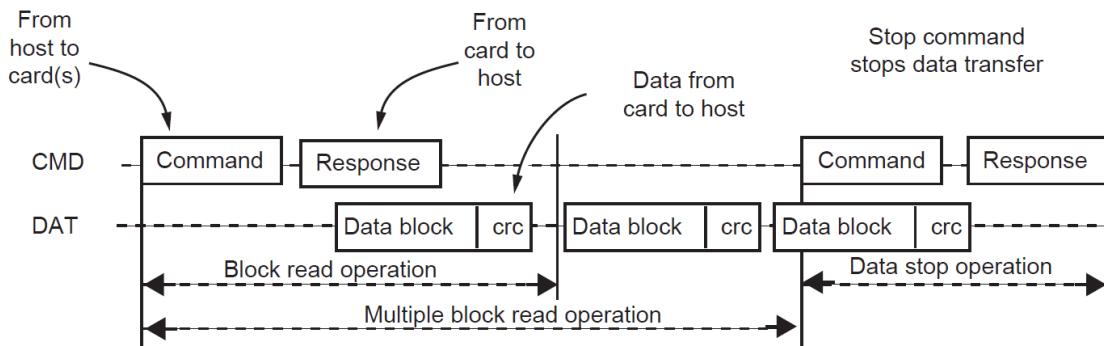
ขา	ชื่อ	วัตถุประสงค์
1	DAT2	Data บิตที่ 2
2	DAT3/CD	Data บิตที่ 3 หรือ Card Detect
3	CMD	ขารับคำสั่ง (Command)
4	VDD	แหล่งจ่ายไฟขั้วบวก
5	CLK	สัญญาณคลื่อ
6	VSS	แหล่งจ่ายไฟขาร่วนด์
7	DAT0	Data บิตที่ 0
8	DAT1	Data บิตที่ 1

ซึ่งพิจารณาด้วยความคุ้มการทำงานของการ์ดหน่วยความจำ SD โดยส่งคำสั่งต่างๆ ผ่านทางขา CMD ซึ่งประกอบด้วย คำสั่ง (CMD: Command) สำคัญๆ ในโหมด SD ผ่านขา CMD ของการ์ด ดังต่อไปนี้

- CMD12: Stop (หยุดการอ่านหรือเขียนข้อมูล)
- CMD17: Read Single Block (อ่านข้อมูลจำนวน 1 บล็อก)
- CMD18: Read Multiple Block (อ่านข้อมูลจำนวนหลายบล็อก)
- CMD24: Write Single Block (เขียนข้อมูลจำนวน 1 บล็อก)
- CMD25: Write Multiple Block (เขียนข้อมูลจำนวนหลายบล็อก)
- CMD32: Erase Block Start (หมายเลขอริ่มต้นสำหรับการลบข้อมูล)
- CMD33: Erase Block End (หมายเลขอริ่งสุดสำหรับการลบข้อมูล)
- CMD38: Erase (ทำการลบข้อมูล)

### 7.3.1 การอ่านข้อมูล

การอ่านข้อมูลจากการดูดหน่วยความจำ SD อยู่ในรูปของบล็อกข้อมูลที่เรียงตัวตามระบบไฟล์ ระบบไฟล์ที่นิยมในการดูดหน่วยความจำ SD ชื่อ ระบบ FAT32 กระบวนการเปิดไฟล์ อ่านไฟล์ และปิดไฟล์จะอาศัยคำสั่งระดับล่าง คือ โทเค็น CMD ตามที่ได้กล่าวไปก่อนหน้าสำหรับดำเนินการในระดับชั้นภายนอก ในหัวข้อที่ 7.1.3



รูปที่ 7.8: ไดอะแกรมเวลาของการอ่านข้อมูลจำนวนหลายบล็อกจากการดูดหน่วยความจำ SD ที่มา: San-Disk Corporation (2003)

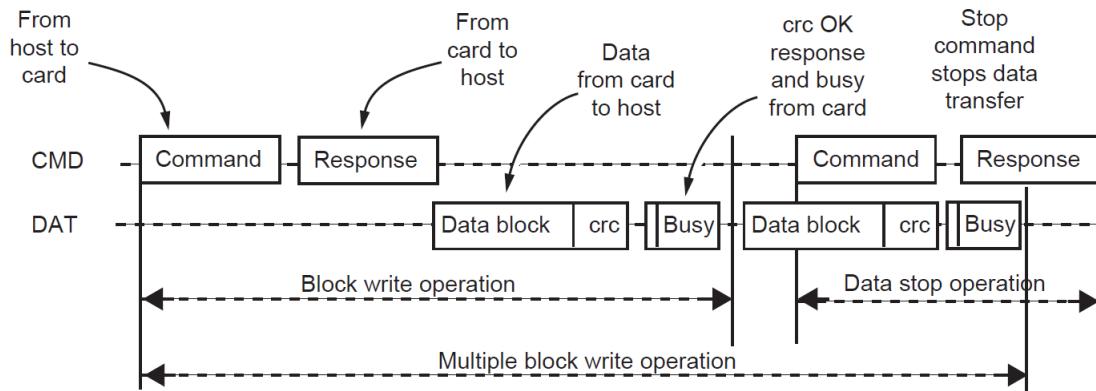
การอ่านข้อมูลจากการดูดหน่วยความจำ SD แบ่งเป็น การอ่านข้อมูลจำนวน 1 บล็อกและการอ่านจำนวนหลายบล็อก การทำงานของทั้งสองวิธีคล้ายกันมาก ดังนี้ ไฮสท์ส่งโทเค็นคำสั่ง **CMD17** ไปยังการดูดหน่วยความจำ การดูดหน่วยความจำจึงตอบกลับด้วยโทเค็น Response การอ่าน 1 บล็อกจะสิ้นสุดการทำงานแค่นี้โดยไม่ต้องมีการต่อоборระหว่างไฮสท์และการดูดอีก

แต่ถ้าส่งโทเค็นคำสั่ง **CMD18** เป็นอ่านหลายบล็อก การทำงานจะหมุนวนไปเรื่อยๆ ตามจำนวนบล็อกที่ไฮสท์ต้องการจะอ่าน การดูดจะส่งบล็อกข้อมูลไปยังไฮสท์เรื่อยๆ จนถึงข้อมูลบล็อกสุดท้ายเมื่อครบตามจำนวนที่ต้องการ หลังจากนั้น ไฮสท์จะส่งโทเค็น **CMD12 Stop** เพื่อยุดกระบวนการ และการดูดหน่วยความจำจะตอบกลับด้วยโทเค็น Response ไปยังไฮสท์

บล็อกข้อมูลจะมี CRC ที่ว่างรายในการดูดคำนวนไว้พ่วงตามท้ายมาด้วย เพื่อให้วงจรผู้รับของไฮสท์ทำหน้าที่ตรวจสอบความผิดพลาดระหว่างเดินทางไปยังไฮสท์ ย่อมมาจาก **Cyclic Redundancy Check** ซึ่งไฮสท์จะนำข้อมูลที่ได้รับมาคำนวนหา CRC แล้วเปรียบเทียบกับ CRC ที่พ่วงมา หากตรงกันแสดงว่าข้อมูลที่ได้รับไม่มีความผิดพลาด ผู้อ่านสามารถศึกษาการทำงานของ CRC เพิ่มเติมได้จาก [wikipedia.org](https://en.wikipedia.org)

### 7.3.2 การเขียนข้อมูล

การเขียนข้อมูลลงในкар์ดหน่วยความจำ SD สามารถทำได้โดยเขียนข้อมูลเป็นบล็อกๆ ละ 2048-4096 ไบต์ ขึ้นกับระบบไฟล์ เช่น FAT32, EXT4 เป็นต้น ซึ่งได้อธิบายในหัวข้อที่ 7.1 โดยจะต้องอาศัยกระบวนการเปิดไฟล์ เขียนไฟล์ และปิดไฟล์ ตามลำดับ



รูปที่ 7.9: ไดอะแกรมเวลาของการเขียนข้อมูลจำนวนหลายบล็อกจาก การ์ดหน่วยความจำ SD ที่มา: [San-Disk Corporation \(2003\)](#)

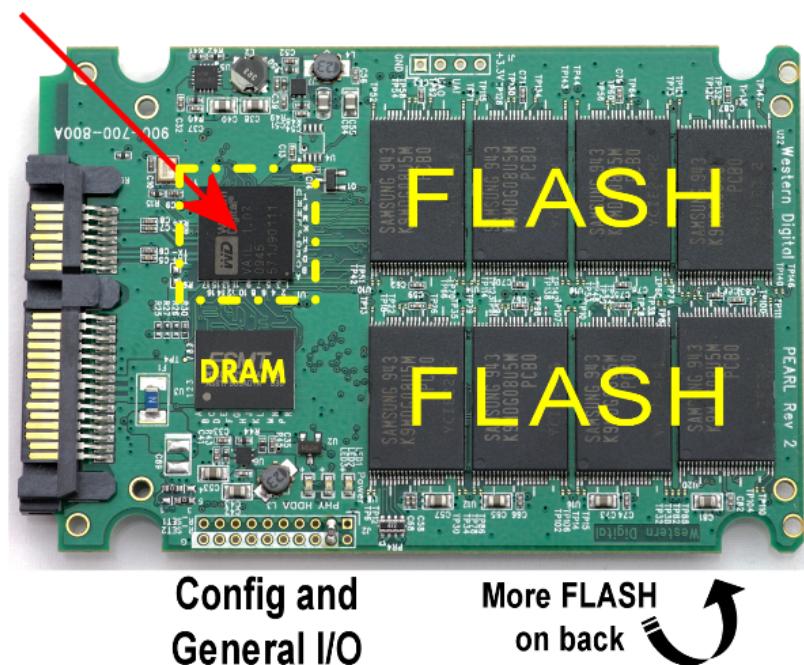
การเขียนข้อมูลจากการ์ดหน่วยความจำ SD แบ่งเป็น การเขียนข้อมูลจำนวน 1 บล็อก และการเขียนจำนวนหลายบล็อก การทำงานของทั้งสองวิธีคล้ายกันมาก ดังนี้ โไฮส์ท์ส่งโทเค็นคำสั่ง **CMD25** ไปยังการ์ดหน่วยความจำ การ์ดหน่วยความจำจึงตอบกลับด้วยโทเค็น Response ไปยังโไฮส์ท์ โไฮส์จึงส่งบล็อกข้อมูลที่ต้องการเขียนไปยังการ์ดเรียบร้อยๆ จนถึงบล็อกข้อมูลสุดท้าย เมื่อครบตามจำนวนที่ต้องการโไฮส์ท์จะส่งโทเค็นคำสั่ง **CMD12 Stop** เพื่อหยุดกระบวนการและการ์ดหน่วยความจำจะตอบกลับด้วยโทเค็น Response ไปยังโไฮส์ท์ หากเป็นโทเค็นคำสั่ง **CMD24** จะหมายถึง การเขียนเพียงบล็อกเดียว กระบวนการจะเสร็จสิ้นเร็วกว่าการเขียนจำนวนหลายบล็อก โดยไม่ต้องมีการโต้ตอบระหว่างโไฮส์ท์และการ์ดอีกรอบ

บล็อกข้อมูลที่เขียนจะมี CRC ที่โไฮส์ท์คำนวณก่อนเพิ่มเติมด้วย เพื่อทำหน้าที่ตรวจจับความผิดพลาดของข้อมูลที่เขียนระหว่างเดินทางจากโไฮส์ท์ไปยังการ์ด ซึ่งการ์ดจะนำข้อมูลที่ได้รับมาคำนวณหา CRC แล้วเปรียบเทียบกับ CRC ที่พ่วงมา หาก CRC ทั้งสองชุดมีค่าตรงกันแสดงว่าข้อมูลที่ได้รับไม่มีความผิดพลาด ระหว่างที่คำนวณค่าของ CRC นี้ การ์ดจะแจ้งสถานะ Busy กลับไปยังโไฮส์ท์เพื่อให้โไฮส์ท์รอ หลังจากนั้น โไฮส์ท์จึงส่งบล็อกข้อมูลที่ต้องการต่อเมื่อสถานะเปลี่ยนจาก Busy เป็น Ready

## 7.4 โซลิดสเตทไดรฟ์ (Solid-State Disk: SSD)

### SSD Controller

SATA  
and  
Power



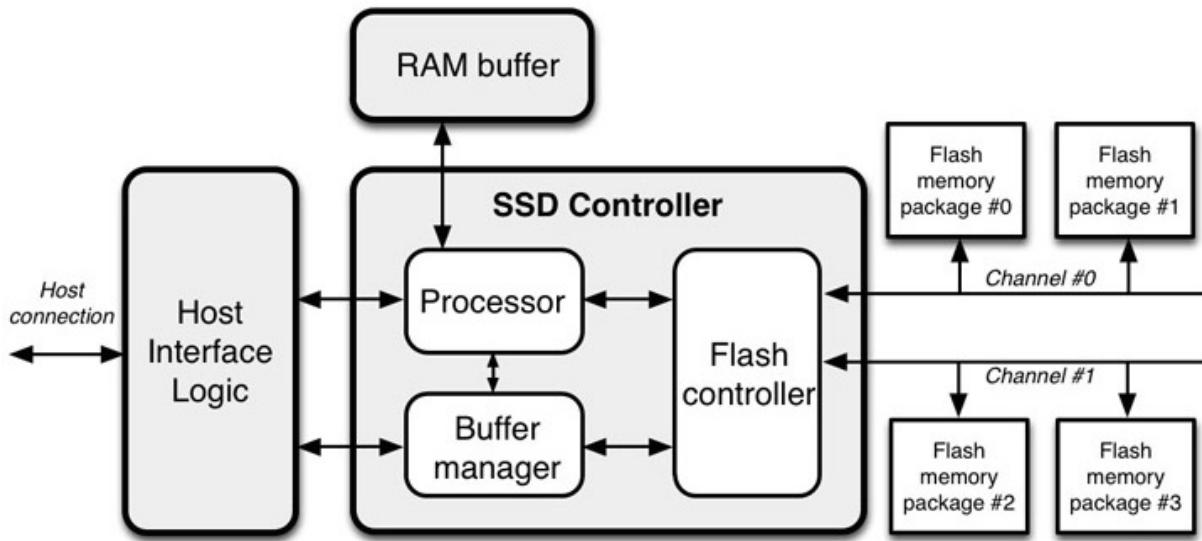
รูปที่ 7.10: แผ่นวงจรพิมพ์ภายในอุปกรณ์ SSD ชนิด SATA III ประกอบด้วยชิพหน่วยความจำแฟลช ไดนามิก แรม คอนโทรลเลอร์สำหรับควบคุม ที่มา: [thatoldnews.site](http://thatoldnews.site)

SSD ใช้ชิปที่ไม่เคลื่อนย้ายกับฮาร์ดดิสก์ไดรฟ์ เพื่อความหมายที่ไม่เคลื่อนย้ายกัน ความจุของ SSD มีแนวโน้มเพิ่มสูงขึ้น ทำให้ SSD มีขนาดเริ่มต้นตั้งแต่ 120-128 กิกะไบต์ (GiB) ขึ้นไป แนวโน้มต้นทุน/ความจุหนึ่งหน่วยของ SSD จะถูก ลงเรื่อยๆ จนไม่เคลื่อนย้ายกับต้นทุน/ความจุหนึ่งหน่วยฮาร์ดดิสก์ไดรฟ์ในอนาคต โดยองค์ประกอบหลักที่สำคัญ คือ หน่วยความจำแฟลช NAND ที่มีความจุต่อชิปเพิ่มสูงขึ้นไปอีก และใช้หน่วยความจำ SDRAM เพื่อทำหน้าที่เป็น แคช หรือ บัฟเฟอร์ และเพิ่มประสิทธิภาพการทำงานให้รวดเร็วขึ้น

โครงสร้างภายในของ SSD ในรูปที่ 7.10 ประกอบด้วย แผ่นวงจรพิมพ์ภายในอุปกรณ์ SSD ชนิด SATA III ส่วนเชื่อมต่อกับเมนบอร์ดคอมพิวเตอร์ ไม่ใช่คอนโทรลเลอร์ บัฟเฟอร์หรือแคช และชิปหน่วยความจำแฟลช จำนวนหนึ่งตามขนาดความจุ

- **ชิปหน่วยความจำแฟลช NAND** ความจุสูงเรียงตัวต่อเนื่องกันจนได้ความจุมากพอตามที่ระบุ โดยการจัด เรียงทั้งด้านบน (Channel 0) และด้านล่าง (Channel 1) ของแผ่นวงจรพิมพ์หลัก เนื่องจากหน่วยความจำ แฟลชมีเวลาเข้าถึงนานกว่าหน่วยความจำ SDRAM นักออกแบบจึงต้องจัดตั้งชิปหน่วยความจำ SDRAM เพิ่ม เติม เพื่อทำหน้าที่เป็นแคชหรือบัฟเฟอร์ให้กับ SSD ผู้อ่านจะสังเกตว่าชิปหน่วยความจำแฟลชมีลักษณะตัว ถังคล้ายกับรูปที่ 7.3
- **วงจรควบคุม (Controller)** นิยมใช้ไมโคร คอน โทร ล เล อ ร์ และ เฟิร์มแวร์ ทำงาน โดยเฉพาะไมโคร คอนโทรลเลอร์ระดับ ARM รุ่น Cortex R ซึ่ง R ย่อมาจากคำว่า Real Time เหมาะกับการควบคุมบาง ส่วนของรถยนต์ แขนหุ่นยนต์ ฮาร์ดดิสก์ไดรฟ์ เครื่องมือทางการแพทย์ อุปกรณ์เครือข่าย เป็นต้น ที่มา: [anandtech.com](http://anandtech.com) ผู้อ่านสามารถค้นคว้าเรื่องนี้จากข้อมูลการตลาดของ ARM ในหัวข้อที่ 4.9.1

- วงจรเชื่อมต่อกับไฮสต์ (Host Interface) การเชื่อมต่อที่ได้รับความนิยม ได้แก่ SATA III และ NVMe Express SATA III ได้รับความนิยมในระยะแรก เนื่องจากเป็นมาตรฐานของฮาร์ดดิสก์ไดรฟ์มาก่อน ในขณะที่ NVMe M.2 ได้รับความนิยมเพิ่มมากขึ้นเรื่อยๆ เนื่องจากประสิทธิภาพสูงกว่า SATA III ผู้อ่านสามารถศึกษาเพิ่มเติมได้ที่ [pcworld.com](http://pcworld.com)



รูปที่ 7.11: บล็อกไซด์แกรมภายใน SSD ประกอบด้วย ส่วนเชื่อมต่อกับเครื่อง ไมโครคอนโทรลเลอร์ บัฟเฟอร์ และหน่วยความจำแฟลช ที่มา: [codecapsule.com](http://codecapsule.com)

ผู้อ่านสามารถศึกษาบล็อกไซด์แกรมภายใน SSD เพิ่มเติมได้ในรูปที่ 7.11 ซึ่งประกอบด้วย

- procressor** (Processor) สำหรับรันคำสั่งในเฟิร์มแวร์สำหรับควบคุมการทำงานภายในและเชื่อมต่อกับคอมพิวเตอร์ไฮสต์ ดังนั้น ผู้อ่านควรตรวจสอบผู้ผลิตเป็นระยะๆ ว่ามีการอัปเดทเฟิร์มแวร์ของ SSD รุ่นที่ใช้หรือไม่
- แฟลชคอนโทรลเลอร์** (Flash Controller) สำหรับควบคุมการอ่าน/เขียนข้อมูลหน่วยความจำแฟลช คล้ายกับการทำงานของหน่วยความจำ SD ซึ่งกล่าวในหัวข้อที่ 7.2
- บัฟเฟอร์** (Buffer) และ **การบริหารจัดการบัฟเฟอร์** (Buffer Management) อาศัยหน่วยความจำ SDRAM เป็นบัฟเฟอร์เพื่อพักเก็บข้อมูลชั่วคราวระหว่างที่ เชื่อมต่อกับคอมพิวเตอร์ไฮสต์ ทำให้การอ่าน/เขียนมีระยะเวลาเข้าถึงเฉลี่ย ดังนั้น การใช้หน่วยความจำ SDRAM ให้เต็มประสิทธิภาพ และคุ้มค่า จึงต้องมีการบริหารจัดการบัฟเฟอร์ ทำให้ต้นทุนการผลิตต่ำลง

## 7.5 ฮาร์ดดิสก์ไดร์ฟ (Hard Disk Drive: HDD)



รูปที่ 7.12: โครงสร้าง และ องค์ประกอบ ของ อุปกรณ์ ฮาร์ด ดิสก์ ไดร์ฟ (HDD) ชนิด Serial ATA ที่มา: [blogspot.com](http://blogspot.com)

อุปกรณ์เก็บรักษาข้อมูลที่ได้รับความนิยมจากในอดีต และพัฒนามาเป็นเวลาระยะนาน แผ่นจานแม่เหล็กหมุน มีเส้นผ่าศูนย์กลางสองขนาดที่นิยมผลิต คือ 2.5 นิ้ว และ 3.5 นิ้ว หมุนด้วยความเร็วสูงตั้งแต่ 5400 ถึง 10000 รอบต่อนาที จุดเด่นของหน่วยความจำขนาดสารแม่เหล็ก คือ ความจุข้อมูลที่มากกว่า เริ่มต้นที่หลายร้อยกิกะไบต์ (GB) จนถึงหลายเทอร่าไบต์ (Tera Byte) โดย 1 เทอร่าไบต์ ประมาณเท่ากับ 1000 กิกะไบต์ (GB) ทำให้ราคา ต่อความจุต่ำลง ต้นทุนโดยรวมจึงถูกลง มีอายุการใช้งานที่ยาวนานพอสมควร จุดอ่อน คือ ประสิทธิภาพด้าน ความเร็วที่ต่ำกว่า ฮาร์ดดิสก์ไดร์ฟต้องการปริมาณและน้ำหนักสำหรับจัดเก็บมากกว่า ทำให้โครงสร้างใหญ่ขึ้น ตัวอุปกรณ์รองรับแรงกระแทกกระเทือนจากการตกหล่นหรือแผ่นดินไหวได้น้อยกว่า บริโภคพลังงานสูงกว่าโซลิดส์ เทหายอดร์ฟ เนื่องจากมีการหมุนจานแม่เหล็กเกือบตลอดเวลา จึงเกิดความร้อนแฟ้มากกว่า จึงต้องอาศัยอุปกรณ์ อื่นๆ เช่น พัดลมช่วยระบายความร้อน หรือ เครื่องปรับอากาศในห้องคอมพิวเตอร์และศูนย์ข้อมูล

### 7.5.1 โครงสร้างของฮาร์ดดิสก์เชิงกายภาพ

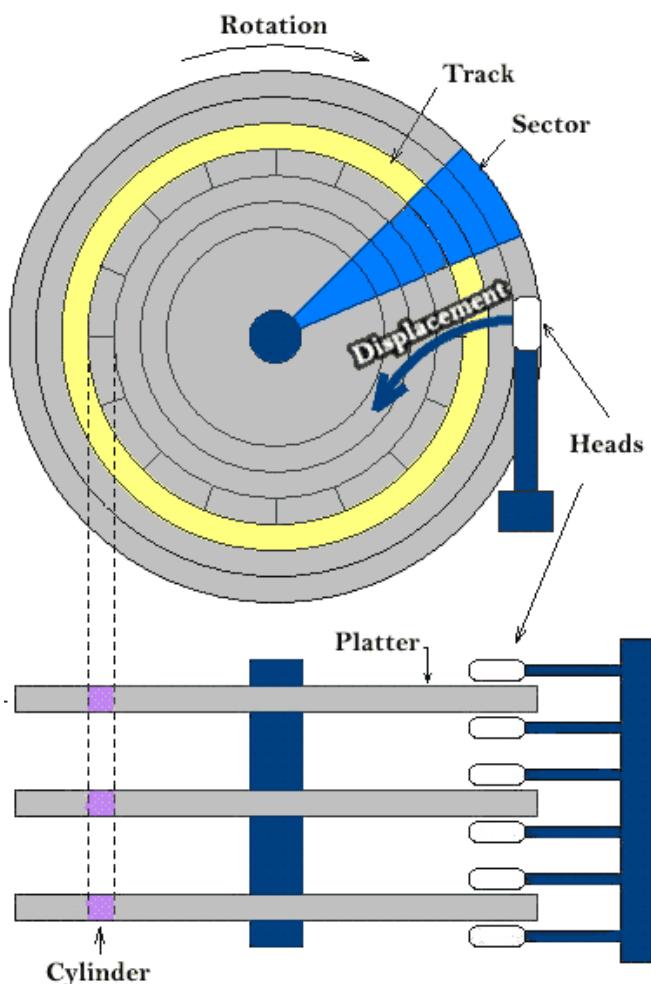
โครงสร้างและองค์ประกอบของอุปกรณ์ฮาร์ดดิสก์ไดร์ฟ (HDD) เชิงกายภาพใน รูปที่ 7.12 ประกอบด้วย

- แผ่นจานแม่เหล็ก (Platter) ซ้อนกันหลายๆ ชั้น เก็บข้อมูลบนจานแม่เหล็ก เพื่อเพิ่มความจุโดยรวม และ หมุนพร้อมกันด้วยความเร็ว 5,400 - 10,000 รอบต่อนาที 90-134 รอบต่อวินาที
- แขนกล (Actuator Arm) มีหัวอ่าน/เขียนข้อมูลสำหรับแต่ละจานยึดอยู่บนแขนกลเดียวกัน แผ่นละ 2 หัว แต่ละจานมีหัวอ่าน/เขียน 1 คู่ ด้านบนและด้านล่าง โปรดสังเกตขนาดของหัวอ่าน/เขียนที่เล็กมาก
- แผ่นวงจรควบคุม (Controller Board หรือ Logic Board) ประกอบด้วยชิปไมโครคอนโทรลเลอร์ที่มี ชิปปี้ ARM ตระกูล Cortex R และอื่นๆ ทำหน้าที่ควบคุมการทำงานโดยรวม โดย ARM จะมีส่วนแบ่งการ

ตลาดสูงมากจากการสำรวจในปี ค.ศ. 2010 ในตารางที่ 4.10

- วงจรเชื่อมต่อกับไฮสต์ (Host Interface) และเครื่องคอมพิวเตอร์ผ่านทาง External Interface ได้แก่ Parallel ATA ([wikipedia](#)) Serial ATA ([wikipedia](#)) NVMe ([wikipedia](#)) เป็นต้น ขนาดบัฟเฟอร์หรือแคช หน่วยเป็น เมบิไบต์ (MiB) เพื่อพกเก็บข้อมูลล่าสุดที่ได้ทำการอ่านหรือเขียนไว้ก่อนทำให้ระยะเวลาเข้าถึงสั้นลง หรือประสิทธิภาพสูงขึ้น

### 7.5.2 โครงสร้างของฮาร์ดดิสก์ไดร์ฟ์เชิงترรอก



รูปที่ 7.13: โครงสร้างของฮาร์ดดิสก์ไดร์ฟ์แบ่งเป็นไซลินเดอร์ แทร็ก และเซกเตอร์ ที่มา: [computable-minds.com](http://computable-minds.com)

แผ่นจานแม่เหล็กแบ่งเป็นหลายๆ แทร็ก (Track) หรือ วงรอบ แต่ละแทร็กจะมีจำนวนเซกเตอร์ (Sector) เท่าๆ กัน พื้นที่หนึ่งเซกเตอร์ มีขนาดความจุ 512 ไบต์เสมอ แต่ละแทร็กจะแบ่งพื้นที่ในแนวรัศมีจากจุดศูนย์กลาง many ขอบตามรูปที่ 7.13

**ไซลินเดอร์ (Cylinder)** คือ กลุ่มของแทร็กที่อยู่บนจานแม่เหล็กต่างๆ และอยู่ห่างจากจุดศูนย์กลางเรียงตัวกันเป็นทรงกระบอก โดยจะนับจากไซลินเดอร์หรือแทร็กหมายเลข 0 ซึ่งอยู่ขอบนอกสุดของ จานแม่เหล็ก โดยจะนับจากขอบนอกสุดเข้าสู่จุดศูนย์กลาง

**บล็อก (Block)** หรือ **คลัสเตอร์ (Cluster)** ประกอบด้วย เซกเตอร์ที่ต่อเนื่องกัน จำนวน  $2^n$  เซกเตอร์เสมอ เช่น 2 4 8 .. เซกเตอร์ในแทร็กเดียวกัน โดยระบบปฏิบัติการจะกำหนดจำนวนเซกเตอร์ที่ต้องการ ยกตัวอย่างเช่น

คลัสเตอร์ขนาด 4096 ไบต์ต้องการพื้นที่จำนวน 8 เซกเตอร์

หมายเลขล็อก LBA: Logical Block Addressing คือ การตั้งหมายเลขล็อกที่กล่าวมาก่อนหน้านี้ให้เรียงตัวตาม หมายเลขไซลินเดอร์ หมายเลขหัวอ่าน และหมายเลขแทร็ก ทำให้ง่ายต่อการบริหารจัดการและเป็นพื้นฐานของระบบบริหารจัดการไฟล์ ที่มา: [wikipedia](https://en.wikipedia.org/wiki/Logical_Block_Address)

ระบบไฟล์จะจองพื้นที่บนฮาร์ดดิสก์โดยอย่างน้อย 1 บล็อกหรือ 1 คลัสเตอร์ให้กับไฟล์หนึ่งไฟล์ ยกตัวอย่าง เช่น ไฟล์ขนาด 800 ไบต์ ต้องการใช้พื้นที่ 8 เซกเตอร์ เช่นเดียวกับไฟล์ขนาด 4096 ไบต์ ในอุปกรณ์เก็บรักษาข้อมูลใดๆ ผู้ใช้ที่ว่าไปสามารถเก็บข้อมูลหรือแอปพลิเคชันในรูปของไฟล์เท่านั้น ในด้านการบริหารจัดการไฟล์ ผู้ใช้สามารถคัดลอกหรือสำเนา (Copy) ลบ (Delete หรือ Remove) ย้าย (Move) คืน (Recover) ไฟล์ต่างๆ ได้ผ่านระบบปฏิบัติการ ในด้านการบริหารจัดการพื้นที่ ผู้ใช้สามารถจัดหมวดหมู่ไฟล์โดยการ สร้างโฟลเดอร์ ตั้งชื่อคัดลอก ย้าย ลบ คืนไฟลเดอร์ต่างๆ เช่นกัน ผู้อ่านสามารถทำความเข้าใจจากการทดลองที่ 12 ภาคผนวก L

### 7.5.3 ความจุและประสิทธิภาพของฮาร์ดดิสก์ไดร์ฟ

ความจุของฮาร์ดดิสก์ไดร์ฟสามารถคำนวณได้จากผลคูณของความจุต่อหนึ่งเซกเตอร์ จำนวนเซกเตอร์ต่อแทร็ก จำนวนไซลินเดอร์ และจำนวนหัวอ่าน ยกตัวอย่าง เช่น  $512 \text{ ไบต์ต่อเซกเตอร์} \times 63 \text{ เซกเตอร์ต่อแทร็ก} \times 1024 \text{ ไซลินเดอร์} \times 256 \text{ หัว เท่ากับ } 8,455,716,864 \text{ ไบต์ หรือ } 8.4 \times 10^9 \text{ ไบต์โดยประมาณ ทำให้ผู้ผลิตใช้หน่วยเป็น } 8.4 \text{ กิกะไบต์ (GB)}$

อายุการใช้งานของฮาร์ดดิสก์ไดร์ฟวัดจากจำนวนชั่วโมงที่เปิดใช้งาน ซึ่งจะมีความแตกต่างจากการวัดอายุของหน่วยความจำแฟลช ที่วัดจากจำนวนครั้งที่เขียนหรือลบข้อมูล ทำให้ฮาร์ดดิสก์ไดร์ฟมีต้นทุนการใช้งานต่อความจุและต่ออายุที่ต่ำและคุ้มค่ากว่า

การวัดประสิทธิภาพของฮาร์ดดิสก์ไดร์ฟอาศัยการวัด เวลาการเข้าถึง ( $T_{acc}$ , Access Time) หน่วยเป็น มิลลิวินาที ประกอบด้วย ช่วงเวลาการรอเฉลี่ย  $T_{rotate}$  เพื่อให้ตำแหน่งที่ต้องการอ่านหมุนมาอยังหัวอ่าน เรียกว่า Rotation Latency ช่วงเวลาการขยับหัวอ่านมายังแทร็กที่ต้องการ  $T_{head}$  และช่วงเวลาการถ่ายโอนข้อมูล ( $T_{xfer}$ , Transfer Time)

$$T_{acc} = T_{rotate} + T_{head} + T_{xfer} \quad (7.1)$$

ดังนั้น เวลาการเข้าถึงจึงมีความสัมพันธ์กับความเร็วรอบในการหมุนของจานแม่เหล็ก ตามสมการต่อไปนี้

$$T_{rotate} = \frac{0.5}{V_{rotate}} \quad (7.2)$$

องค์ประกอบสุดท้าย คือ  $T_{xfer}$  ขึ้นอยู่กับชนิดการเชื่อมต่อกับคอมพิวเตอร์ไฮส豕ท์ เช่น SATA จะมีการพัฒนาประสิทธิภาพสูงขึ้นเป็น SATA III ตามที่ได้กล่าวไปก่อนหน้า

ช่วงเวลาการอเฉลี่ย  $T_{rotate}$  แปรผกผันกับ  $V_{rotate}$  หรือ ความเร็วรอบในการหมุนของจาน หน่วยเป็นรอบต่อวินาที (Round per Minute: RPM) และตำแหน่งของแทร็กที่ต้องขยับหัวอ่านไป ประสิทธิภาพของการอ่านและการเขียนจะขึ้นกับตำแหน่งของข้อมูล การอ่านหรือเขียนข้อมูลที่เรียงตัวต่อเนื่อง (Sequential) จะดีกว่าการอ่านหรือเขียนข้อมูลที่กระจายตัวไม่ต่อเนื่อง (Fragmented) เช่นเดียวกับการอ่านและเขียนข้อมูลของหน่วยความจำแฟลช ถ้ามีการจัดเรียงไฟล์ข้อมูลในแต่ละเซกเตอร์ให้ต่อเนื่องกัน หรือเรียกว่า Defragmentation จะทำให้เวลาการเข้าถึงของฮาร์ดดิสก์ไดร์ฟเฉลี่ยน้อยลงรวมถึงการจัดเรียงลำดับการอ่านเขียนข้อมูลบนดิสก์ (Disk Scheduling) จะช่วยให้ประสิทธิภาพเพิ่มขึ้น ผู้อ่านสามารถศึกษาอัลกอริธึมได้ที่ [geeksforgeeks.org](https://www.geeksforgeeks.org/disk-scheduling-algorithms/)

## 7.6 สรุปท้ายบท

ระบบไฟล์และอุปกรณ์เก็บรักษาข้อมูลจะต้องประสานงานกัน เพื่อให้เครื่องเนล ผู้ใช้งานและแอปพลิเคชัน อื่นๆ สามารถบริหารจัดการไฟล์โปรแกรม ไฟล์ข้อมูลต่างๆ ได้อย่างถูกต้อง และมีประสิทธิภาพสอดคล้องกับ การกิจของระบบคอมพิวเตอร์ นอกจากนี้ ผู้อ่านจะสังเกตเห็นว่า ขนาดหรือความจุของบล็อกข้อมูลในอุปกรณ์ เก็บรักษาข้อมูล เช่น

- หน่วยความจำแฟลชขนาด 1 เพจจะมีขนาดเท่ากับ  $2^{11}$  หรือ 2048 ไบต์
- การ์ดหน่วยความจำ SD ขนาด 1 บล็อกจะมีขนาดเท่ากับ  $2^{11}$  หรือ 2048 ไบต์
- ฮาร์ดดิสก์ไดรฟ์ขนาด 1 เซ็กเตอร์จะมีขนาดเท่ากับ  $2^9$  หรือ 512 ไบต์

สอดคล้องกับ ขนาดของบล็อกข้อมูลในระบบบริหารจัดการไฟล์ และ ขนาดของเพจข้อมูลในเวอร์ชัลเม莫รี ซึ่ง จะมีขนาดเป็นจำนวนเท่าของสองเลข และสำหรับลินกุซ์มีขนาดเท่ากับ 4096 หรือ  $2^{12}$  ไบต์ ในหัวข้อที่ 7.1.3

ตารางที่ 7.3 เป็นการเปรียบเทียบประสิทธิภาพของอุปกรณ์เก็บรักษาข้อมูลหลายชนิดในด้านต่างๆ ประกอบด้วยชิปหน่วยความจำแฟลช NAND, อุปกรณ์ SSD และอุปกรณ์ HDD โดยผู้อ่านจะต้องใช้ปีที่ผลิตเป็นหลักยึดในการทำความเข้าใจ เนื่องจากปีที่ผลิตจะมีผลต่อประสิทธิภาพ ความจุและกำลังไฟฟ้าสูงสุด เพราะเทคโนโลยีการ ผลิตอุปกรณ์เหล่านี้เปลี่ยนแปลงตลอดเวลา

ตารางที่ 7.3: การเปรียบเทียบประสิทธิภาพของอุปกรณ์เก็บรักษาข้อมูลชนิดต่างๆ

อุปกรณ์	แฟลช NAND	SSD	HDD
ผู้ผลิต หมายเลขไมโคร: ที่มา: ปี ค.ศ. ความจุ	Micron MT29F2G08AABWP <a href="http://micron.com">micron.com</a> 2004 25 MiB	Micron MTFDDAK120MAV <a href="http://micron.com">micron.com</a> 2013 120 KiB	Western Digital 5K1000 <a href="http://hgst.com">hgst.com</a> 2016 1000 กิกะไบต์ (GB)
การอ่าน: เวลาเข้าถึง - $T_{acc,avg}$ - $T_{acc,max}$	30 นาโนวินาที 25 ไมโครวินาที	160 ไมโครวินาที 5 มิลลิวินาที	5.5 มิลลิวินาที -
การเขียน: เวลาเข้าถึง - $T_{acc,avg}$ - $T_{acc,max}$	300 ไมโครวินาที 2 มิลลิวินาที	40 ไมโครวินาที 25 มิลลิวินาที	5.5 มิลลิวินาที -
เวลาเจสูงสุด กำลังไฟสูงสุด	4.6 โวลต์ 23 มิลลิวัตต์	5.0 โวลต์ 150 มิลลิวัตต์	5.0 โวลต์ 1.6 วัตต์

อุปกรณ์เก็บรักษาข้อมูลจะเชื่อมต่อกับหน่วยความจำหลัก SDRAM ผ่านวงจรด้านอินพุต/เอาต์พุต โดยใช้ กลไก DMA จากหัวข้อที่ 6.13 และ กลไกการทำอินเทอร์รัปท์ จากหัวข้อที่ 6.12 ในชั้นกายภาพ (Physical) ร่วม กับหลักการ Memory Mapped File ในรูปที่ 3.19 ในระดับสูงขึ้น

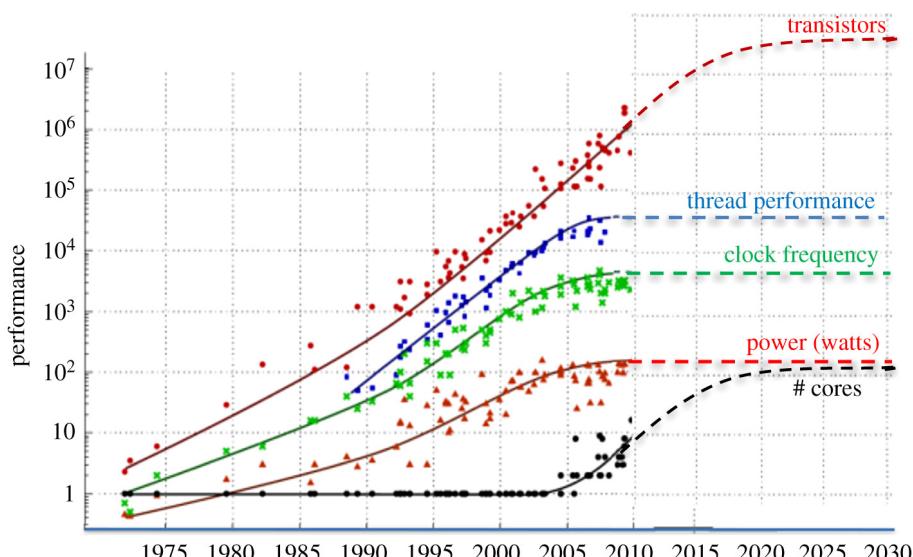
## 7.7 คำถ้ามห้ายบท

1. จงเปรียบเทียบการอ่านหรือเขียนของอุปกรณ์เก็บรักษาข้อมูลชนิดต่างๆ ได้แก่ การ์ดหน่วยความจำ SD, SSD, HDD ในแต่ละอย่าง
  - ขนาดของข้อมูลขั้นต่ำที่สามารถอ่านหรือเขียนได้ หน่วยเป็นไบต์
  - ระยะเวลาเข้าถึง (Access Time) และช่วงเวลาอย่างไร
2. จงเปรียบเทียบกำลังไฟสูงสุดของหน่วยความจำชนิดต่างๆ ได้แก่ หน่วยความจำแฟลช, การ์ดหน่วยความจำ SD, SSD ที่ความจุเท่ากัน เท่าที่จะหาข้อมูลได้
3. เหตุใดการอ่านหรือเขียนข้อมูลในอุปกรณ์เก็บรักษาข้อมูลจึงต้องอาศัยกลไกของ DMA และการอินเทอร์รัปท์ทำงานร่วมกัน
4. จงเปรียบเทียบการเชื่อมต่อชนิด PATA และ SATA ในแต่ละประสิทธิภาพ
5. จงเปรียบเทียบการเชื่อมต่อชนิด SATA II และ SATA III ในแต่ละประสิทธิภาพ
6. จงเปรียบเทียบการเชื่อมต่อชนิด SATA III และ NVMe ในแต่ละประสิทธิภาพ
7. จงเปรียบเทียบหน่วยความจำ SD คลาสต่างๆ ในแต่ละประสิทธิภาพ

## บทที่ 8

# การคำนวณแบบขนาน (Parallel Computing) ด้วยบอร์ด Pi3/Pi4

จำนวนทรานซิสเตอร์และประสิทธิภาพของชิปเพิ่มขึ้นเฉลี่ยไม่มาก ประมาณ 10 เท่าในเวลา 10 ปี จากเส้นกราฟในรูปที่ 8.1 แต่ในช่วงปี ค.ศ. 1995-2005 จำนวนทรานซิสเตอร์และประสิทธิภาพเพิ่มขึ้นเฉลี่ยเป็น 2 เท่าทุก 1.5 ปี หรือ 10 เท่าใน 5 ปี โดยประมาณตามกฎของ Moore (Moore's Law) ที่มา: [wikipedia](#) สืบเนื่องจาก การเพิ่มจำนวนบิตข้อมูลที่ประมวลผลได้จาก 4 บิตเป็น 8, 16, 32 และ 64 บิตต่อ 1 คาบเวลา (Cycle Time) ของ สัญญาณคล็อก การปรับโครงสร้างการทำงานจากไปป์ไลน์ เป็นซูเปอร์สเกลาร์ (SuperScalar) ซึ่งเป็นการเริ่มต้น ของการทำงานแบบขนานระดับคำสั่ง (Instruction Level Parallelism) เป็นการทำงานแบบขนานระดับเรเด (Thread Level Parallelism) สำหรับชิปที่รองรับมัลติธread (Multithread) ที่มา: [Patterson and Hennessy \(2016\)](#) และมีจำนวนแกนประมวลผล (Core) ต่อชิปมากขึ้นเรื่อยๆ เรียกว่า ชิปมัลติคอร์ (Multi-Core) ที่มา: [Tanenbaum and Austin \(2012\); Tanenbaum and Bos \(2014\)](#) และ [wikipedia](#) และจำนวนเพิ่มขึ้นถึง หลายสิบแกนประมวลผลต่อชิป เรียกว่า ชิปแมนีคอร์ (Many Core) ที่มา: [Tanenbaum and Bos \(2014\)](#) และ [wikipedia](#) ในปัจจุบันและอนาคตอันใกล้



รูปที่ 8.1: ประสิทธิภาพของการประมวลผลด้วยชิปชิปยูนิมัลติคอร์ ที่มา: [royalsocietypublishing.org, John \(2020\)](#)

แกนนอน คือ หมายเลขปี ค.ศ. แกนตั้งคือ ประสิทธิภาพ (Performance) หน่วยเป็นจำนวนเท่าของค่า

ประสิทธิภาพเทียบกับซีพียูตัวแรก เนื่องจากเลขประสิทธิภาพเพิ่มขึ้นรวดเร็วมากจนต้องใช้แกนเลขสิบยกกำลัง จุดสีแต่ละจุด คือ ประสิทธิภาพในด้านต่างๆ ของซีพียูหนึ่งตัว ได้แก่ จำนวนทรานซิสเตอร์ภายในซีพียู (จุดกลมแดง) ค่าประสิทธิภาพของซีพียูเพียง 1 แกนประมวลผล (จุดสีเหลี่ยมฟ้า) ความถี่สัญญาณคล็อกสูงสุด (ภาคบาทสีเขียว) ค่ากำลังไฟสูงสุด (สามเหลี่ยมสีแดง) และจำนวนแกนประมวลผลต่อชิป (จุดกลมดำ) เส้นทึบสีต่างๆ คือ เส้นทดถอย (Regression) ของประสิทธิภาพด้านต่างๆ จากข้อมูลจุดสีน้ำเงิน เส้นประสีต่างๆ คือ เส้นพยากรณ์ที่ผู้เขียนเอกสารอ้างอิงทำนายไว้ต่อเนื่องจากเส้นทึบ สรุปได้ว่า

- จำนวนทรานซิสเตอร์ต่อชิปมีจำนวนเพิ่มขึ้นจากซีพียูตัวแรก ในปี ค.ศ. 1971 จนถึงจุดอิ่มตัวที่จำนวน  $10^8$  เท่า หรือประมาณหลายพันล้านตัวในปัจจุบัน และมีแนวโน้มเพิ่มขึ้นตามจำนวนแกนประมวลผลที่เพิ่มขึ้น
  - ความถี่สัญญาณคล็อกสูงสุด หน่วยเป็น เมกะเฮิรตซ์ ในปี ค.ศ. 1971-1995 เพิ่มขึ้นช้าประมาณ 10 เท่า ในเวลา 12-15 ปีและเพิ่มขึ้นแบบก้าวกระโดดประมาณ 10 เท่าในปี ค.ศ. 1995-2005 และถึงจุดอิ่มตัวในปี ค.ศ. 2005 เป็นต้นมา
  - ค่าประสิทธิภาพของซีพียูเพียง 1 เหรด คือ เนื่องจากซีพียูในอดีตมีเพียง 1 แกนประมวลผลเท่านั้น ประสิทธิภาพของเรดรดเดี่ยว (Single Thread Performance) จึงเพิ่มขึ้นอย่างก้าวกระโดดตั้งแต่ปี ค.ศ. 1990-2005 เพราะแพร่พันตามความถี่สัญญาณคล็อกและจำนวนทรานซิสเตอร์ แต่ค่าประสิทธิภาพมีแนวโน้มจะเปลี่ยนแปลงน้อยตั้งแต่ปี ค.ศ. 2005 เป็นต้นมา แต่ประสิทธิภาพโดยรวมของซีพียูจะเพิ่มขึ้น เนื่องจากจำนวนแกนประมวลผลที่เพิ่มขึ้น
  - จำนวนแกนประมวลผลต่อชิป ในอดีตมีเพียง 1 แกนประมวลผลและเพิ่มจำนวนขึ้นอย่างก้าวกระโดด และเพิ่มขึ้นอย่างต่อเนื่องจนอาจจะอิ่มตัวที่จำนวน 128 แกนประมวลผลโดยประมาณ
  - ค่ากำลังไฟสูงสุด (Power) หน่วยเป็นวัตต์ต่อชิป มีค่าเพิ่มขึ้นตามความถี่สัญญาณคล็อกสูงสุด จะมีแนวโน้มค่อนข้างคงที่ เพราะข้อจำกัดด้านการระบายความร้อนออกจากตัวชิป แนวโน้มการบริโภคพลังงานที่ไม่เพิ่มสูงจากเนื้องมาจากการเทคโนโลยีการผลิตที่พัฒนาในอนาคต
- จึงเป็นที่มาของเนื้อหาในบทนี้ โดยมีวัตถุประสงค์ดังต่อไปนี้
- เพื่อให้เข้าใจโครงสร้างและการทำงานของซีพียูชนิดมัลติคอร์ตามหลักการคอมพิวเตอร์แบบขนานชนิด SMP: Shared Memory Multiprocessor ที่มา: [Patterson and Hennessy \(2016\)](#) และ [wikipedia](#)
  - เพื่อให้เข้าใจความแตกต่างระหว่างการคำนวณแบบอนุกรม (Serial) และแบบขนาน (Parallel)
  - เพื่อให้เข้าใจการพัฒนาซอฟต์แวร์แบบมัลติເର୍ଡ ແລະ ແບບຂ່ານາດ ດ້ວຍໄລບຣາີ OpenMP ໃນການທົດລອງທີ 13 ກາຄພນກ [M](#)

## 8.1 เครื่องคอมพิวเตอร์แบบขนานชนิดมัลติคอร์

เทคโนโลยีสารกึ่งตัวนำที่ใช้สร้างคอมพิวเตอร์ทุกชนิดที่ได้แจกแบ่งในหัวข้อที่ 1.1 ได้พัฒนาอย่างต่อเนื่องเป็นระยะเวลาไม่น้อยกว่าครึ่งศตวรรษ ทำให้คอมพิวเตอร์มีวัฒนาการอย่างรวดเร็วและประสิทธิภาพเพิ่มขึ้นอย่างก้าวกระโดด การคำนวณแบบขนานเป็นทางเลือกหนึ่งที่เพิ่มประสิทธิภาพ โดยสามารถแบ่งเป็นชนิดหลักๆ ตามการใช้งานหน่วยความจำหลักหรือหน่วยความจำภายในที่ [Patterson and Hennessy \(2016\)](#) คือ

- เครื่องคอมพิวเตอร์แบบขนานชนิด **SMP** (Shared Memory Multiprocessor) ทุกๆ แกนประมวลผลใช้หน่วยความจำหลักหรือ SDRAM และอินพุต/เอาต์พุตร่วมกัน เป็นต้นแบบของซีพียูชนิดมัลติคอร์ในเครื่องคอมพิวเตอร์ทุกชนิดปัจจุบัน ซีพียูเหล่านี้มาจากการผลิตรายสำคัญ เช่น Intel, AMD และ ARM เป็นหลัก กรณีศึกษาในหัวข้อที่ [8.1.1](#) และ [8.1.2](#)

ซีพียู ARM ตระกูล Cortex A53/A72 ทั้ง 4 แกนประมวลผลบนบอร์ด Pi3/Pi4 กรณีศึกษาในตำราเล่มนี้ใช้หลักการหน่วยความจำร่วมกัน (SMP) ซึ่งระบบปฏิบัติการ Raspberry Pi OS (ลินุกซ์) สามารถรองรับการทำงานแบบมัลติเรลดและการคำนวณแบบขนาน (Parallel Computing) ได้ ชิป ARM ในปัจจุบันรองรับจำนวนแกนประมวลผลมากขึ้น เรียกว่า **ซีพียูมัลติคอร์** และมากขึ้นจนเรียกว่า **ซีพียูแมเนคอร์** สำหรับใช้เป็นซีพียูสำหรับเครื่องคอมพิวเตอร์ตั้งโต๊ะ โน้ตบุ๊ค และแท็บเล็ต ยกตัวอย่าง เช่น ชิป Apple Silicon M1 มีจำนวนซีพียู 8 แกนประมวลผล ที่มา: [wikipedia](#) เนื่องจากเทคโนโลยีในการผลิตมีขนาดเล็กลง ทำให้ประหยัดพลังงาน และสามารถบรรจุจำนวนแกนประมวลผลต่อชิปได้เพิ่มขึ้นตามแนวโน้มในรูปที่ [8.1](#)

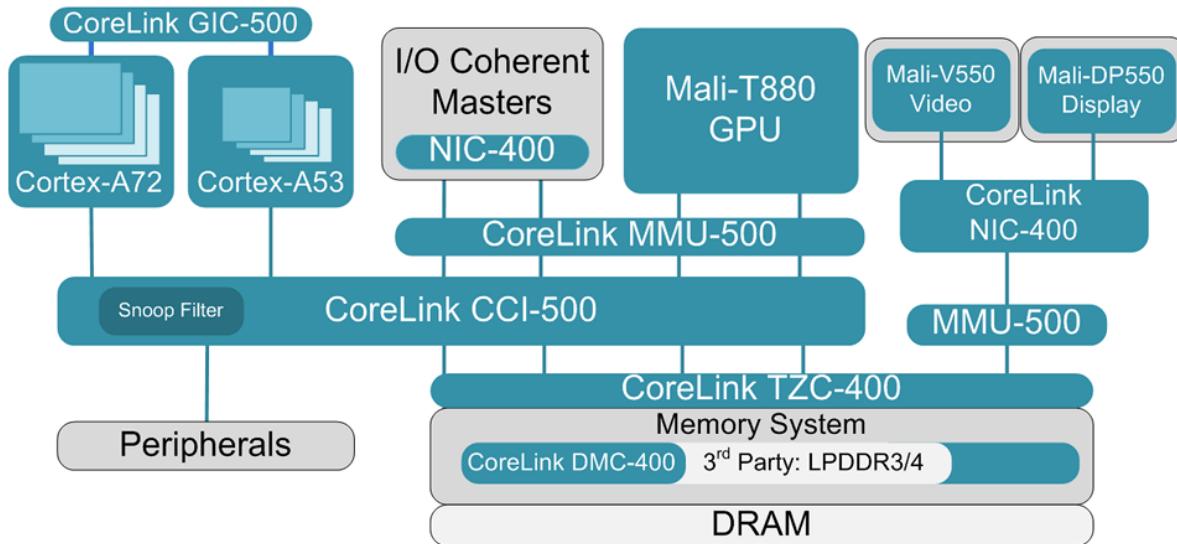
- มัลติคอมพิวเตอร์** (Multicomputer) ที่มา: [Tanenbaum and Bos \(2014\)](#); [Patterson and Hennessy \(2016\)](#) มีลักษณะสำคัญคือ เครื่องคอมพิวเตอร์จำนวนมาก แต่ละเครื่องสร้างจากซีพียูชนิดมัลติคอร์หรือซีพียูแมเนคอร์ที่เหมือนกันและใช้ระบบปฏิบัติการเดียวกัน เรียกว่า **โนนด** (Node) เชื่อมต่อกันด้วยเครือข่ายประสิทธิภาพและความเร็วสูงภายใต้ศูนย์ข้อมูลเดียวกัน แต่ละเครื่องใช้หน่วยความจำภายในที่แยกกัน แต่จะมีการแลกเปลี่ยนข้อมูลผ่านช่องทางเครือข่าย เช่น การทำงานของโปรแกรมบนเครื่องคอมพิวเตอร์ใช้การรับส่งข้อมูลระหว่างกัน (Message Passing) บนเครือข่ายเพื่อประสานงานและสื่อสารกัน และสามารถช่วยกันแก้ปัญหาเดียวกันแบบขนานได้ กรณีศึกษา คือ เครื่องซูเปอร์คอมพิวเตอร์ ในหัวข้อที่ [8.1.2](#)

### 8.1.1 กรณีศึกษา ARM Cortex A72 และ Cortex A53/A72

โครงสร้างของซีพียูสำหรับคอมพิวเตอร์เคลื่อนที่ชนิดแท็บเล็ต สมาร์ตโฟน และระบบสมองกลฝังตัวที่ได้รับความนิยม อุปกรณ์เหล่านี้ประกอบด้วยซีพียู ARM Cortex A72 หรือรุ่นอื่นๆ ที่ทันสมัยกว่าจำนวน 1 ถึง 4 แกนประมวลผลและ Cortex A53/A72 หรือรุ่นอื่นๆ ที่ทันสมัยกว่าจำนวนไม่น้อยกว่า 4 แกนประมวลผลดังรูปที่ [8.2](#) ทำงานร่วมกันตามหลักการ **big.LITTLE** ของบริษัท ARM โดยซีพียู big คือ Cortex A72 เนื่องจากแต่ละแกนประมวลผลมีประสิทธิภาพสูงทำให้บริโภคพลังงานมากกว่าเหมาะสมสำหรับเกมและมัลติมีเดียคุณภาพสูง และซีพียู LITTLE คือ Cortex A53/A72 เพราะแต่ละแกนประมวลผลมีประสิทธิภาพด้อยกว่าทำให้บริโภคพลังงานน้อยกว่าเหมาะสมสำหรับโปรแกรมทั่วไปที่ใช้งานปกติ ซีพียูทั้งหมดเชื่อมต่อกันด้วย Cache Coherence Interconnect ตัวอย่างซีพียูที่ใช้ซีพียูทั้งสองชนิดนี้ ได้แก่

- ชิป Helio X20 ที่มา: [wikichip.org](#) ผลิตโดยบริษัท MediaTek
- ชิป RK3399 ที่มา: [wikidot.com](#) ผลิตโดยบริษัท Rockchip สำหรับคอมพิวเตอร์บอร์ดเดียว Rock Pi 4 ที่มา: [rockpi.org](#) และ

- ชิป i.MX8 NXP ที่มา: [nxp.com](http://nxp.com) ผลิตโดยบริษัท NXP สำหรับบอร์ดพัฒนาระบบฝังตัว ที่มา: [variscite.com](http://variscite.com) เป็นต้น



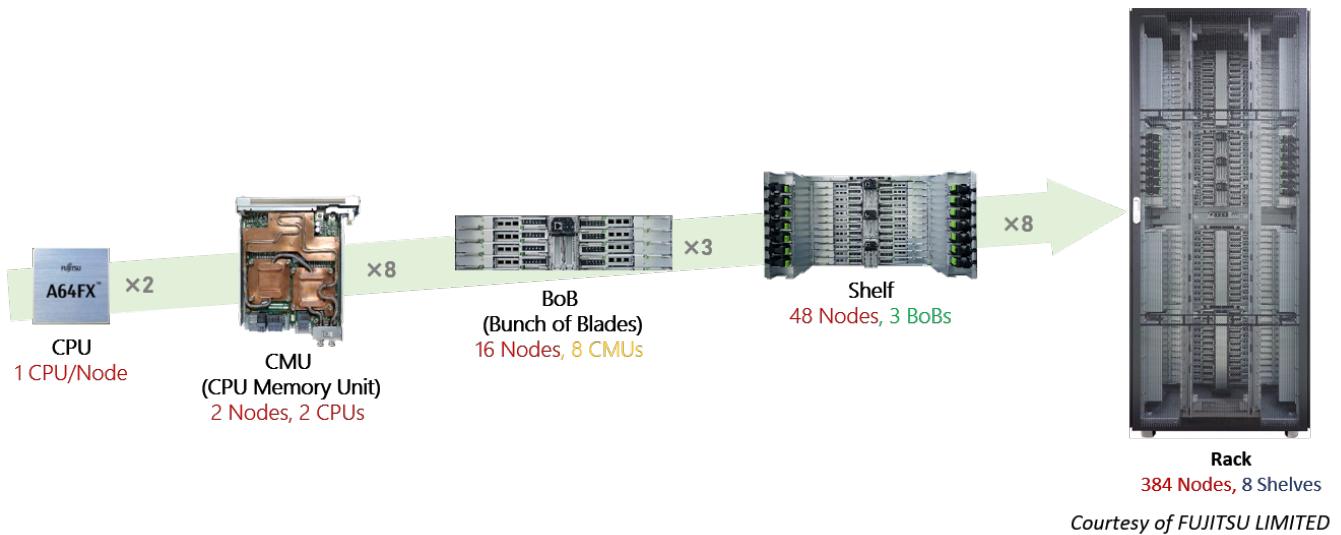
รูปที่ 8.2: โครงสร้างภายในของชิป ARM ตามเทคโนโลยี big.LITTLE เชื่อมระหว่างแกนประมวลผลด้วย Cache Coherent Interconnect (CCI) ที่มา: [arm.com](http://arm.com)

การเชื่อมต่อซีพียูจำนวนหลายๆ แกนประมวลผลและใช้หน่วยความจำหลักร่วมกัน หรือ SMP มีความซับซ้อนต่ำทำให้เพิ่มจำนวนแกนประมวลผลได้ไม่ยาก การพัฒนาโปรแกรมแบบขนานบนหลักการแชร์หน่วยความจำหลักร่วมกัน จึงมีจุดเด่นหลายประการแต่มีจุดอ่อนด้วยเช่นกัน เนื่องจากซีพียูแต่ละแกนประมวลผลมีแคชข้อมูลลำดับที่ 1 ของตนเองและอิสระจากกันในรูปที่ 4.1 ปัญหาจะเกิดขึ้นเมื่อโปรแกรมที่ทำงานแบบมัลติเรตติเพื่อให้สามารถคำนวณแบบขนาน โดยมีเรตติอย่างน้อย 2 ตัวรันอยู่ในซีพียู 2 แกนประมวลผล ต้องการเข้าถึงตัวแปรตัวเดียวกันในหน่วยความจำภายใน ทำให้แคชลำดับที่ 2 ต้องโหลดค่าของตัวแปรนั้นมาพักเก็บในแคชข้อมูลลำดับที่ 1 (L1 data Cache) ของทั้งสองแกนประมวลผล โดยแต่ละเรตติสามารถเปลี่ยนแปลงค่าตัวแปรนั้นโดยอิสระ จึงทำให้เรตติทั้งสองมองเห็นค่าตัวแปรซึ่งเดียวกันแต่ค่าไม่ตรงกัน เราเรียกปัญหาสำคัญนี้ว่า **ปัญหาแคชโค希เรนท์ (Cache Coherent Problem)** ที่มา: [Patterson and Hennessy \(2016\)](https://www.cs.berkeley.edu/~patterson/cs152/papers/hennessy-patterson-2016.pdf)

กลไกสำคัญที่เชื่อมแกนประมวลผลเหล่านี้เข้าด้วยกัน ประกอบด้วย มอดูล ชื่อ CoreLink CCI (Cache Coherency Interconnect) เพื่อรับจำนวนแกนประมวลผลที่เพิ่มมากขึ้น และรองรับการเชื่อมต่อระหว่างซีพียูและจีพียู ทำหน้าที่ประสานงานแคชที่กระจายอยู่ในแกนประมวลผลต่างๆ ให้ทำงานร่วมกันโดยไม่เกิดปัญหาตามรูปที่ 8.2 ARM ได้พัฒนาการเชื่อมต่อระหว่างแคชมาอย่างต่อเนื่อง โดย CCI จะดักฟัง (Snoop) ว่าซีพียูแต่ละแกนประมวลผลใช้ค่าตัวแปรเดียวกันหรือไม่ ผู้อ่านสามารถค้นคว้าเรื่องนี้เพิ่มเติมได้ใน [wikipedia.org](https://en.wikipedia.org/wiki/Cache_coherence)

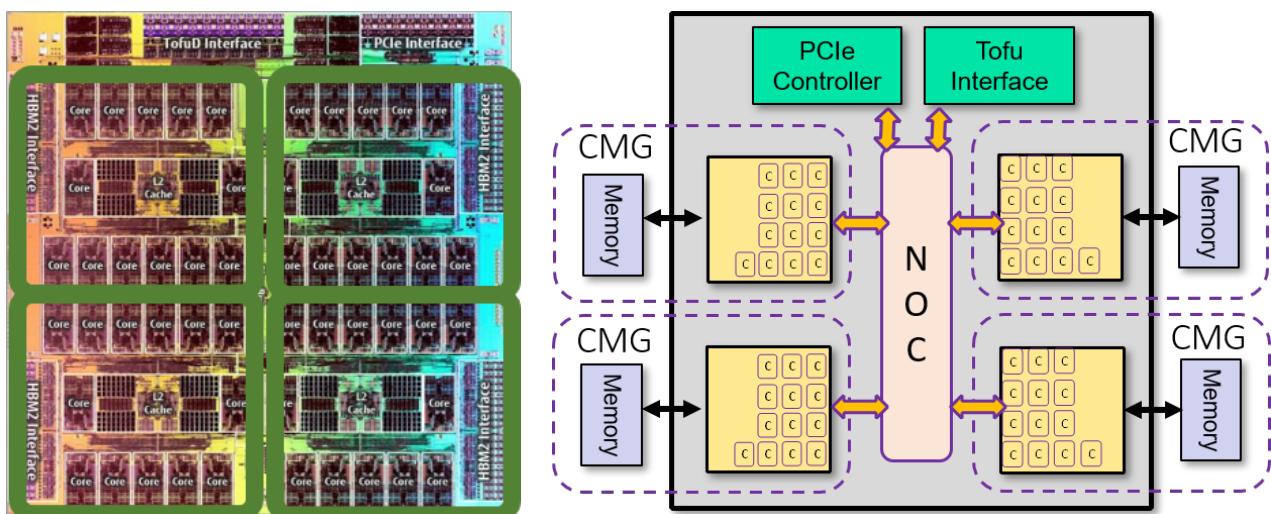
### 8.1.2 กรณีศึกษา Fujitsu A64FX ในเครื่องซูเปอร์คอมพิวเตอร์

เครื่องซูเปอร์คอมพิวเตอร์ Fugaku จัดเป็นมัลติคอมพิวเตอร์ชนิดคลัสเตอร์ขนาดใหญ่มาก ตั้งอยู่ที่ศูนย์ RIKEN Center for Computational Science ประเทศญี่ปุ่น ที่มา: [top500.org](http://top500.org) จัดอันดับ Fugaku เป็นซูเปอร์คอมพิวเตอร์อันดับ 1 ของโลกในเดือนพฤษภาคม ปี ค.ศ. 2020 ตามหน่วย FLOPS (Floating-Point Operations Per Second) ด้วยเลขทศนิยมฐานสองมาตรฐาน IEEE754 ซึ่งอธิบายอยู่ในหัวข้อที่ 2.6



รูปที่ 8.3: โครงสร้างของชูเปอร์คอมพิวเตอร์ Fugaku ประกอบด้วยตู้เร็กจำนวน 414 ตู้ๆ ละ 384 โนนด แต่ละ โนนดมี Fujitsu A64FX จำนวน 1 ชิปซึ่งภายในประกอบด้วยชีพี้ย ARM จำนวน 48 แกนประมวลผล ที่มา: [pcmag.com](http://pcmag.com)

โครงสร้างของชูเปอร์คอมพิวเตอร์ Fugaku ในรูปที่ 8.3 ประกอบด้วยตู้เร็กจำนวน 414 ตู้ๆ ละ 384 โนนด รวมทั้งสิ้นจำนวน 158,976 โนนด แต่ละโนนดมีชีพี้ย Fujitsu A64FX 1 ชิปและหน่วยความจำหลัก 32 กิกะไบต์ (GiB) ภายในประกอบด้วยชีพี้ย ARM จำนวน 48 แกนประมวลผล คิดเป็นจำนวนแกนประมวลผลรวมทั้งหมด 7,630,848 แกนประมวลผล รวมหน่วยความจำหลักหรือหน่วยความจำภายในภาพทั้งหมด 4968 เพตะไบต์ รายละเอียดเพิ่มเติมที่ [wikipedia](https://en.wikipedia.org/wiki/Fugaku_(supercomputer))



รูปที่ 8.4: ภาพถ่ายโดยละเอียดของแกรมของชิป Fujitsu A64FX ประกอบด้วยชีพี้ย ARM จำนวน 48 แกนประมวลผล ที่มา: [fujitsu.com](http://fujitsu.com)

รูปที่ 8.4 แสดงให้เห็นโครงสร้างภายในของชีพี้ย Fujitsu A64FX ประกอบด้วยชีพี้ย ARM จำนวน 4 ชุดๆ ละ 12 แกนประมวลผล หรือเท่ากับ 48 แกนประมวลผลและทำงานที่ความถี่คลื่อก 2.2 กิกะเอิรตซ์ ชีพี้ยแต่ละแกนประมวลผลรองรับคำสั่งแอสเซมบลี ARM เวอร์ชัน v8.2-A ความยาว 64 บิต รองรับเทคโนโลยี SVE (Scalable Vector Extension) โดย Fujitsu และ ARM ออกแบบร่วมกัน ผู้อ่านสามารถค้นคว้ารายละเอียดเพิ่มเติมที่ [wikipedia](https://en.wikipedia.org/wiki/Fujitsu_A64FX) ชิปชีพี้ย Fujitsu A64FX นี้ผลิตโดยบริษัท TSMC (Taiwan Semiconductor Manufacturing

Company) ด้วยเทคโนโลยี 7 นาโนเมตร ผู้อ่านสามารถค้นควารายละเอียดเกี่ยวกับชิปเพิ่มเติมที่ลิงก์ต่อไปนี้ [github.com](https://github.com)

ซีพียูในแต่ละโหนดเชื่อมต่อกันด้วยเครือข่ายบนชิป (Network on Chip: NoC) ชนิดบัสวงแหวน (Ring Bus) และเชื่อมต่อกับหน่วยความจำภายในภาพประสีทิชิปสูงด้วยเทคโนโลยี HBM2 (High Bandwidth Memory 2) ความจุรวม 32 กิกะไบต์ (GiB) รายละเอียดเกี่ยวกับ HBM2 เพิ่มเติมที่ [wikipedia](https://en.wikipedia.org) แต่ละโหนดจัดเป็นเครื่องคอมพิวเตอร์แบบขนาดนิ่มมัลติคอร์และใช้หน่วยความจำร่วมกัน (SMP) และเชื่อมต่อกันระหว่างกันด้วยเครือข่ายความเร็วสูงชื่อ ToFuD Interface ผ่านอินเตอร์เฟส PCIe (Peripheral Component Interconnect Express) ของแต่ละโหนด เราจึงสรุปได้ว่า ชูเปอร์คอมพิวเตอร์ Fugaku เป็นมัลติคอมพิวเตอร์ที่มีหน่วยความจำหลักอิสระจากกัน ชนิดคลัสเตอร์ขนาดใหญ่มาก

ซอฟต์แวร์ระบบของเครื่องชูเปอร์คอมพิวเตอร์ Fugaku เป็นระบบปฏิบัติการลินุกซ์ Red Hat Enterprise Linux 8 สำหรับเครื่องควบคุมโหนดต่างๆ และมีเครื่องเนลเสริมในแต่ละโหนด ชื่อ [McKernel](#) เพื่อทำหน้าที่ควบคุมการคำนวณแบบขนาน โปรแกรมเมอร์สามารถพัฒนาโปรแกรมแบบขนาน โดยใช้คอมไฟเลอร์ภาษา C/C++ และไลบรารี OpenMP ซึ่งจะได้กล่าวถึงในหัวข้อที่ [8.3](#) และประสานงานระหว่างโหนดด้วยไลบรารี MPI (Message Passing Interface) ของ Fujitsu ซึ่งพัฒนาเพิ่มเติมจากไลบรารี OpenMPI ที่มา: [fujitsu.com](https://www.fujitsu.com) เครื่องชูเปอร์คอมพิวเตอร์ Fugaku นี้สามารถรองรับภาษาอื่นๆ เช่น ภาษา Java และภาษา Python นอกจากที่กล่าวมาข้างต้น

## 8.2 ความขนาน (Parallelism)

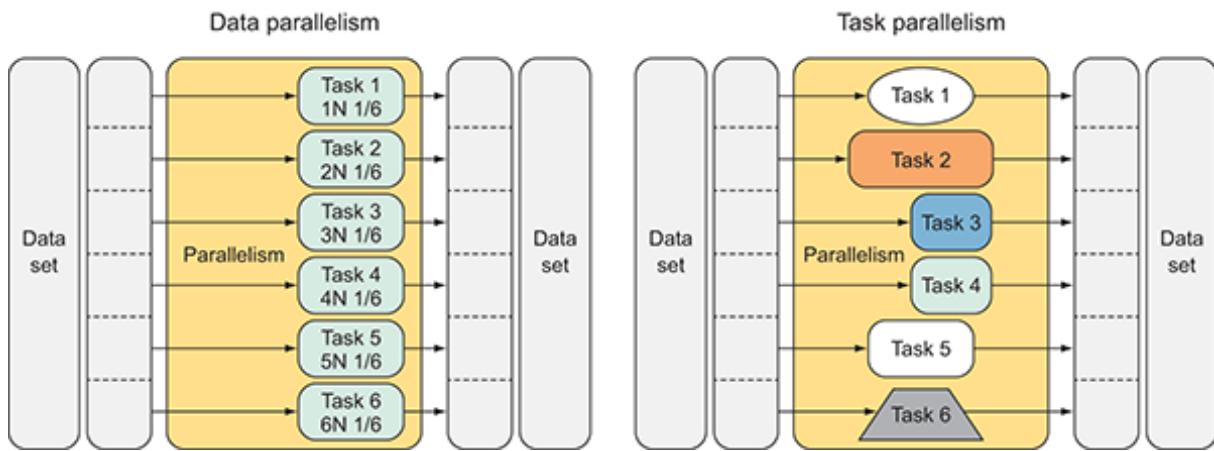
การคำนวณแบบขนานตั้งอยู่บนพื้นฐานของอาร์ดแวร์และความขนาน (Parallelism) ของปัญหา การคำนวณแบบขนาน ประกอบด้วยองค์ประกอบสำคัญ 2 ส่วนคือ

- อัลกอริธึมแบบขนาน (Parallel Algorithm) สำหรับแก้ปัญหาโจทย์ต่างๆ ที่สามารถใช้ประโยชน์จากการคำนวณของปัญหารือโจทย์นั้นๆ
- เครื่องคอมพิวเตอร์แบบขนาน (Parallel Computer) ชนิดมัลติคอร์ และชนิดแมนิคอร์ซึ่งใช้หน่วยความจำภายในร่วมกันมีประสิทธิภาพสูงขึ้นและราคาต่ำลง การพัฒนาโปรแกรมแบบขนานบนเครื่องคอมพิวเตอร์ชนิดนี้อาศัยการพัฒนาโปรแกรมแบบมัลติ-thread (Multithread Programming) ด้วยไลบรารีมาตรฐาน เช่น [Pthread](#) และ [OpenMP](#)

เมื่อนำข้อมูลหัก (Big Data) มาประมวลผลด้วยอัลกอริธึมแบบขนานซึ่งโปรแกรมเมอร์สามารถพัฒนาให้ทำงานแบบมัลติ-thread ตั้งที่กล่าวไปแล้ว เพื่อรับบนเครื่องคอมพิวเตอร์ชนิดมัลติคอร์หรือชนิดแมนิคอร์และสูงขึ้นไปถึงระดับชูเปอร์คอมพิวเตอร์ จะทำให้อัลกอริธึมแบบขนานนั้นทำงานได้เร็วขึ้นกว่าอัลกอริธึมแบบอนุกรม ความขนานแบ่งเป็น 2 ชนิดหลักๆ คือ ความขนานของข้อมูล (Data Parallelism) ที่มา: [Hillis and Steele \(1986\)](#); [Patterson and Hennessy \(2016\)](#) และความขนานของงานย่อย (Task Parallelism) ที่มา: [Patterson and Hennessy \(2016\)](#)

### 8.2.1 ความขนานของข้อมูล (Data Parallelism)

ความขนานของข้อมูลนิยมใช้ในอัลกอริธึมต่างๆ เช่น การประมวลผลภาพ วิดีโอ หรือข้อมูลขนาดใหญ่มาก การเรียงข้อมูล การสืบค้นข้อมูล การจัดทำดัชนีข้อมูล เป็นต้น ในรูปที่ [8.5](#) ด้านซ้าย อัลกอริธึมสามารถแบ่งข้อมูลจำนวน  $6N$  จะแบ่งออกเป็น 6 ส่วนเท่าๆ กัน และมอบหมายให้กับแต่ละเรดรับภาระงานจำนวน  $N$  หน่วย รวม



รูปที่ 8.5: การคำนวณแบบขนานโดยอาศัยความขนาดของข้อมูล (Data Parallelism) และความขนาดของงานย่อย (Task Parallelism) ที่มา: [manning.com](https://manning.com)

ทั้งหมด 6 เหรียญเพื่อสิ่งงานซึ่งมีพื้นที่สำหรับการคำนวณมากกว่าหรือเท่ากับ 6 แกนประมวลผลพร้อมๆ กัน และเกิดประสิทธิภาพสูงสุด การทดลองที่ 13 ภาคผนวก M จะนิยามการวัดประสิทธิภาพของการคำนวณแบบขนาน ความขนาดของข้อมูลเหมาะสมสำหรับการคำนวณแบบขนานด้วยจีพีਯู ชีพีyu อาร์เรย์ (Processor Array) และฮาร์ดแวร์ชนิด FPGA ซึ่งแตกต่างกันตามโจทย์หรือปัญหาและอัลกอริธึมสำหรับแก้ปัญหานั้นๆ

### 8.2.2 ความขนาดของงานย่อย (Task Parallelism)

หลักการพื้นฐานของการพัฒนาซอฟต์แวร์ คือ การแบ่งงานหลักในรูปที่ 8.5 ด้านขวา ที่มีความซับซ้อนมากให้เป็นงานย่อยหรือทาสก์ (Task) จำนวนหนึ่งในรูปที่ 8.5 ด้านซ้าย จากรูปงานหลักสามารถแบ่งเป็นงานย่อย คือ ทาสก์ 1 (Task 1) จนถึง ทาสก์ 6 (Task 6) ที่ทำงานต่างกันนี้อยู่กับรายละเอียด แล้วจึงมอบหมายงานย่อยแต่ละงานให้กับเครดิต 1 เหรียญรันงาน โปรดสังเกตสัญลักษณ์ที่แตกต่างกันในรูป ซึ่งอาจช่วยทำให้การทำงานหลักโดยรวมใช้เวลาประมวลผลน้อยลง นั่นคือ ประสิทธิภาพสูงขึ้น

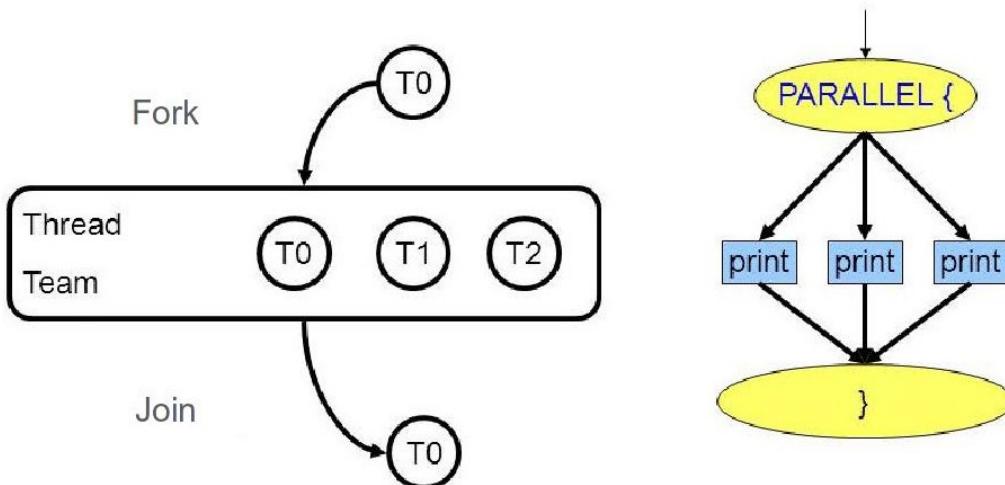
### 8.2.3 ความขนาดแบบผสม (Hybrid Parallelism)

องค์ประกอบสำคัญในทางปฏิบัติของการพัฒนาโปรแกรมแบบขนาน ด้านฮาร์ดแวร์ คือ จำนวนแกนประมวลผล สถาปัตยกรรมคำสั่ง โครงสร้างของจีพีyu โดยเฉพาะความจุ/ขนาดและลำดับขั้นหน่วยความจำที่มีระยะเวลาเข้าถึง (Access Time) ที่แตกต่างกัน เช่น แคชลำดับต่างๆ หน่วยความจำภายใน รวมไปถึงอุปกรณ์เก็บรักษาข้อมูล องค์ประกอบด้านซอฟต์แวร์ ได้แก่ ระบบปฏิบัติการ ภาษาคอมพิวเตอร์ และไลบรารี รวมไปถึงชนิดและขนาดของข้อมูล ดังนั้น การคำนวณแบบขนานจึงมีการผสมผสานระหว่างความขนาดทั้งสองชนิด เนื่องจากอัลกอริธึมแต่ละตัวมีลักษณะพิเศษเฉพาะตัว ยกตัวอย่าง เช่น งานย่อยแต่ละงานมีการคุณ/การบวกເກເຕັບ หรือการคูณ/การบวกເມທຣິກ່າ ซึ่งนิยมใช้ความขนาดของข้อมูลเนื่องจากข้อมูลมีการแบ่งแยกผลลัพธ์ที่ชัดเจน

### 8.3 การพัฒนาอัลกอริธึมแบบมัลติ เฮอด และแบบขนานด้วยไลบรารี OpenMP

การพัฒนาอัลกอริธึมแบบขนานในปัจจุบันมีความสำคัญมากขึ้น เนื่องจากอุปกรณ์คอมพิวเตอร์มีชีพยุชนิดมัลติคอร์และชนิดแม่นิคอร์เพิ่มมากขึ้นเรื่อยๆ ตามกรณีศึกษาที่กล่าวไว้แล้วในหัวข้อที่ 8.1 ดังนั้น โปรแกรมเมอร์สามารถแปลงฟังก์ชันที่ถูกเรียกใช้บ่อยๆ ให้กลายเป็นเรดรของการทำงานบนมัลติคอร์พร้อมๆ กันได้ โดยมีระบบปฏิบัติการที่รองรับการทำงานแบบมัลติเ赫อด ภาษาที่รองรับการทำงานแบบมัลติเ赫อด ได้แก่ ภาษา C/C++ ใช้ไลบรารี (Library) มาช่วยเสริมจึงจะรองรับการทำงานแบบมัลติเ赫อด เช่น PThread และ OpenMP เป็นต้น ส่วนภาษา JAVA เป็นภาษาที่ถูกออกแบบให้รองรับการทำงานทำมัลติเ赫อด โดยสรุป โปรแกรมต่างๆ ที่มีอัลกอริธึมแบบขนานสามารถทำงานอยู่บนชีพยุชนิดมัลติคอร์ จะต้องสร้างจำนวนเรดรให้มากกว่าหรือเท่ากับจำนวนคอร์ที่มี เพื่อกระจายเรดรต่างๆ ไปยังทุกๆ แกนประมวลผลให้ทำงานพร้อมๆ กัน

```
#pragma omp parallel
{
    printf("Hello world %d\n", omp_get_thread_num());
}
```



รูปที่ 8.6: ตัวอย่างซอฟต์แวร์สโค้ด โฟล์กการทำงานแบบขนาน และเรดรประกอบด้วย T0, T1 และ T2 พิมพ์ข้อความ Hello World ตามด้วยหมายเลขเรดรประจำตัว โดยใช้ไลบรารี OpenMP แก้ไขจากที่มา: [slideplayer.com](http://slideplayer.com)

ไลบรารี **openMP** มีประโยชน์โครงสร้าง (Construct) หลายชนิด ที่เสริมการพัฒนาโปรแกรมคอมพิวเตอร์ภาษา C/C++ และภาษา Fortran ให้สามารถทำงานแบบมัลติเ赫อดได้ โดยองค์กร [openmp.org](http://openmp.org) ได้กำหนดเป็นมาตรฐานและพัฒนาเป็นเวอร์ชันต่างๆ เรื่อยมา เพื่อให้คอมไพล์เตอร์แต่ละตัวท่าน้าที่แปลซอร์สโค้ดตามมาตรฐาน ซึ่งต่ำราเล่มนี้ใช้คอมไпал์เตอร์ GCC เป็นเครื่องมือหลัก

```
#pragma omp parallel ...
{
// Parallel Region
}
```

ประโยค #pragma omp parallel ... เป็นประโยคที่ใช้กำหนดบริเวณที่เรียกว่า Parallel Region หมายเหตุ ... หมายถึง โครงสร้าง (Construct) ชนิดต่างๆ ที่กำหนดในมาตรฐาน OpenMP ซึ่งจะยกตัวอย่างต่อไป

ทุกโปรแกรมที่คอมไพล์และลิงก์ด้วยไลบรารี OpenMP จะเริ่มต้นรันแบบอนุกรม (Serial) ด้วย-thread หลัก หรือ (Master Thread) ก่อนเสมอ เมื่อ thread หลักรันมาถึงโปรแกรมบริเวณ Parallel Region thread หลักสามารถสร้าง thread ผู้ช่วย หรือ (Worker Thread) ตามรายละเอียดของประโยคนั้น และแจกจ่ายภาระงานให้ thread ผู้ช่วย และตัว thread หลักเองช่วยกันทำงาน ขั้นตอนนี้ เรียกว่า การ Fork ในรูปที่ 8.6 เมื่อแต่ละ thread ทำงานเสร็จสิ้นแล้ว ก็จะรวมกับ thread หลัก เรียกว่า การ Join เพื่อเพียง thread หลักทำงานต่อแบบอนุกรมจนกว่า thread หลักจะรันถึง บริเวณ Parallel Region อื่นๆ อีก ไลบรารี OpenMP ตั้งชื่อการทำงานลักษณะนี้ว่า ไมเดล Fork-Join

ตามชอร์สโค้ดตัวอย่างในรูปที่ 8.6 T0 ทำงานอยู่เพียง thread เดียวจนถึง Parallel Region T0 จึงสร้าง (การ Fork) thread ผู้ช่วยอีก 2 thread เพื่อการประมวลผลแบบขนานรวม 3 thread ประกอบด้วย T0, T1 และ T2 เพื่อสั่งพิมพ์ข้อความ Hello World ตามด้วยหมายเลข thread ประจำตัว T0 ในไลบรารี OpenMP หมายถึง thread หลักซึ่งมีหมายเลข thread ประจำตัว (Thread ID) เท่ากับ 0 เสมอ ส่วน T1 และ T2 คือ thread ผู้ช่วย ซึ่งโปรแกรมเมอร์สามารถอ่านค่าหมายเลข thread ประจำตัวของแต่ละ thread ได้จากฟังก์ชัน `omp_get_thread_num()`; ระบบปฏิบัติการจะกระจายthread ผู้ช่วยไปรันบนแกนประมวลผลที่ว่าง ณ เวลานั้น เพื่อให้ซีพียูแต่ละแกนประมวลผลนั้นเพฟ์ซ์คำสั่งภายใน Parallel Region มาลดรหัสและประมวลผลตามปกติ

ในรูปที่ 8.6 คำสั่ง printf ในพื้นที่ Parallel Region จะพิมพ์ประโยค

```
Hello world 1
Hello world 0
Hello world 2
```

หรือแตกต่างจากนี้ ขึ้นกับว่า thread ใดได้เรียกใช้ฟังก์ชัน printf() ก่อน thread อื่นๆ ตัวอย่างนี้แสดงการใช้งานเอกสารพุต ด้วยฟังก์ชัน printf ซึ่งซีพียูจำนวนมากกว่า 1 แกนประมวลผลขึ้นไปที่กำลังรัน thread ทั้ง 3 thread นี้ต้องผลัดกันใช้งานอินพุตและเอาต์พุต ในกรณีนี้คือ จะแสดงผลตามคำสั่ง printf ทำให้การรันแต่ละรอบไม่เหมือนเดิม เนื่องจากซีพียูทุกแกนใช้อุปกรณ์อินพุต/เอาต์พุตร่วมกัน เมื่อ thread ผู้ช่วยทำงานเสร็จสิ้นโปรแกรมจะยกเลิก thread ผู้ช่วยให้เหลือเพียง thread หลัก คือ T0 เพียง thread เดียวที่จะทำงานต่อไป เรียกว่า การ Join

ตัวอย่างอัลกอริธึมแบบขนานสำหรับทำความเข้าใจเบื้องต้น คือ การคูณเมตริกซ์ ในหัวข้อที่ 8.3.1 และ การเรียงข้อมูล ในหัวข้อที่ 8.3.2

### 8.3.1 การคูณเมตริกซ์ (Matrix Multiplication) แบบขนาน

การทดลองที่ 13 การพัฒนาโปรแกรมแบบขนานด้วย OpenMP ภาคผนวก M การคูณเมตริกซ์เป็นตัวอย่าง การคำนวณแบบขนานที่เข้าใจง่ายและนิยมใช้จริงในการคำนวณทางวิทยาศาสตร์และคณิตศาสตร์ทั่วไป เช่น ภาษาคำนวณประสิทธิภาพสูง (High Performance Computing) เช่น ภาษา R Project และภาษา Matlab ซึ่งรองรับการประมวลผลข้อมูลหรือตัวแปรชนิดเมตริกซ์และเวกเตอร์ เป็นต้น การคูณเมตริกซ์แบบอนุกรมสามารถทำงานได้อย่างรวดเร็วหากเมตริกซ์มีขนาดเล็ก แต่เมื่อเมตริกซ์มีขนาดใหญ่ขึ้น โปรแกรมจำเป็นต้องอาศัยการคำนวณแบบขนาน เนื่องจากความซับซ้อนเฉลี่ยวของ การคูณเมตริกซ์ เท่ากับ  $O(N^3)$  ตามพิชณิต Big-O Rosen (2002) ดังนั้น

ตัวอย่างที่ 8.3.1 การคูณเมตริกซ์แบบขนานบนบอร์ด Pi3/Pi4 ซึ่งมีซีพียูทั้งหมด 4 แกนประมวลผลในการทดลองที่ 13 ภาคผนวก M

$A, B, C$  คือ เมทริกซ์จัตุรัสมีขนาด  $N \times N$  โดยมีวนรอบ *for* ชั้นกัน 3 ชั้น

```
#pragma omp parallel for private(k, j)
for(i=0;i<N;i++) {
    for(j=0;j<N;j++) {
        C[i][j]=0.; // set initial value of resulting matrix C = 0
        for(k=0;k<N;k++) {
            C[i][j]=C[i][j]+A[i][k]*B[k][j];
        } // k
    } // j
} // i
```

หมายเหตุ  $i$  และ  $j$  คือหมายเลขแถวและหมายเลขคอลัมน์ของเมทริกซ์  $C$  ตามลำดับ

การทดลองจับเวลาเฉพาะการวนรอบนี้เพื่อเปรียบเทียบและคำนวณต่างๆ เฮอดหลักจะสร้าง Hera ผู้ช่วยที่ประโภคโครงสร้าง **#pragma omp parallel for** ตามจำนวน Hera ที่ผู้ใช้ต้องการ ยกตัวอย่าง เช่น 4 Hera นับรวม Hera หลัก ตามหลักการ SMP แต่ละ Hera ซึ่งประจำอยู่ที่ชิพยูแต่ละแกนประมาณ 4 เท่าของตัวแปรทุกตัว ยกเว้น  $j$  และ  $k$  ซึ่งจะอธิบายต่อไป

เฮอดหลักจะแบ่งข้อมูลตามตัวแปร  $i$  เท่ากันทั้ง 4 Hera ดังนี้

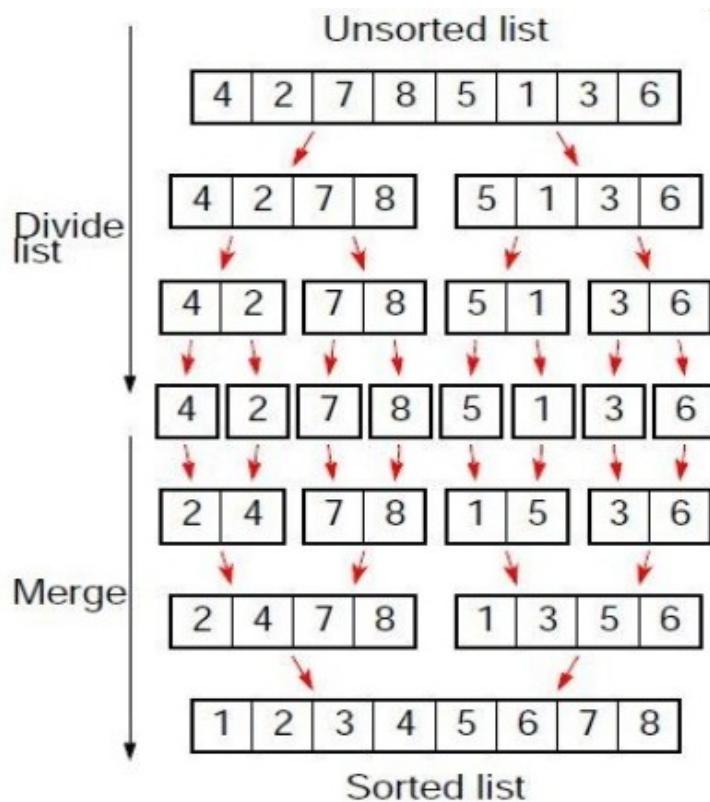
- แถวที่  $i=0$  ถึง  $\frac{N}{4}-1$  เพื่อคำนวณ  $C[i][j]$  จากคอลัมน์  $j=0$  ถึง  $N-1$
- แถวที่  $i=\frac{N}{4}$  ถึง  $\frac{N}{2}-1$  เพื่อคำนวณ  $C[i][j]$  จากคอลัมน์  $j=0$  ถึง  $N-1$
- แถวที่  $i=\frac{N}{2}$  ถึง  $\frac{3N}{4}-1$  เพื่อคำนวณ  $C[i][j]$  จากคอลัมน์  $j=0$  ถึง  $N-1$
- แถวที่  $i=\frac{3N}{4}$  ถึง  $N-1$  เพื่อคำนวณ  $C[i][j]$  จากคอลัมน์  $j=0$  ถึง  $N-1$

เมื่อได้รับภาระงานแล้ว แต่ละ Hera จะวนรอบตัวแปรคอลัมน์  $j$  และตัวแปรการบวก  $k$  ของตนเองตามประโภค **private(j, k)** ตามลำดับ เพื่อที่จะคำนวณได้อย่างอิสระจาก Hera อื่นๆ

### 8.3.2 การเรียงข้อมูล (Sorting) แบบขนาน

การเรียงข้อมูลเป็นอัลกอริธึมพื้นฐานที่ผู้อ่านควรจะได้เรียนในวิชา เช่น วิชา Algorithm Analysis and Design หรือวิชา Data Structure and Algorithm เป็นต้น อัลกอริธึมเรียงข้อมูลเริ่มต้นจากการทำงานแบบอนุกรม และพัฒนาให้เป็นอัลกอริธึมแบบขนานมีอาร์ดแวร์รองรับ  $N$  คือ ขนาดของข้อมูลที่ต้องการเรียงลำดับ ความซับซ้อนเฉลี่ยในรูปของพีชคณิต  $O(\cdot)$  ขึ้นกับอัลกอริธึมที่เลือก ยกตัวอย่าง เช่น

- การเรียงข้อมูลพื้นฐาน เช่น อัลกอริธึม Bubble Sort และ Insertion Sort มีความซับซ้อนเฉลี่ยเท่ากับ  $O(N^2)$  สามารถใช้ได้กับข้อมูลทุกประเภท
- การเรียงข้อมูลตามค่าเลขประจำตำแหน่ง เช่น Radix Sort เหมาะกับข้อมูลที่เป็นเลขจำนวนเต็ม
- การเรียงข้อมูลด้วยหลักการแบ่งและยึดครอง (Divide and Conquer) เช่น MergeSort และ Quick-Sort มีความซับซ้อนเฉลี่ยเป็น  $O(N \log N)$  เหมาะกับข้อมูลทุกประเภท



รูปที่ 8.7: การทำงานของอัลกอริธึม MergeSort ตามหลักการแบ่งและยืดครองสามารถนำมาตัดแปลงเป็นแบบอนุกรมและแบบขนานได้ในตัวอย่างที่ 8.3.2 ที่มา: [slideshare.net](http://slideshare.net)

การทำงานแบบขนานของอัลกอริธึมการเรียงข้อมูลนิยมใช้ความนานของข้อมูลเป็นหลัก โดยเฉพาะอัลกอริธึมหลักการแบ่งและยืดครอง ทำรายเล่นนี้ขอใช้ MergeSort เป็นกรณีศึกษา เนื่องจากเป็นอัลกอริธึมที่เข้าใจง่ายกว่า QuickSort และมักใช้การเขียนโปรแกรมแบบเรียกซ้ำ (Recursive)

ตัวอย่างที่ 8.3.2 ชอร์สโค้ดภาษา C/C++ ของอัลกอริธึมการเรียงข้อมูล MergeSort แบบขนานด้วยไลบรารี OpenMP แก้ไขตัดแปลงจากที่มา: [github.com](https://github.com)

ฟังก์ชัน `main()` สร้างข้อมูลจำนวน `SIZE` ตัวแบบสุ่มแล้วเก็บในตัวแปรอาร์เรย์ `a[ ]` ซึ่งเป็นอาร์เรย์ข้อมูล ต้นฉบับก่อนเรียงและหลังการเรียงเสร็จสิ้น อาร์เรย์ `temp` คือ อาร์เรย์สำหรับเก็บพักข้อมูลชั่วคราว ตัวแปร `SIZE` คือ ขนาดของอาร์เรย์ที่ต้องการเรียงข้อมูล แล้วจึงเรียกฟังก์ชัน `mergesort_parallel_omp()` ด้วยจำนวนเรตติ้งตันในค่าตัวแปร `num_threads` หากว่าหรือเท่ากับจำนวนแกนประมวลผลของเครื่อง

```
int main() {
    int a[SIZE];
    int temp[SIZE];
    int num_threads;
    mergesort_omp(a, SIZE, temp, num_threads);
}

void mergesort_omp(int a[], int size, int temp[], int threads) {
    if (threads == 1) {
        mergesort_serial(a, size, temp);
```

```

    }

else if (threads > 1) {
    #pragma omp parallel sections
    {
        #pragma omp section
        mergesort_omp(a, size/2, temp, threads/2);
        #pragma omp section
        mergesort_omp(a+size/2, size-size/2, temp+size/2, threads-threads/2);
    }
    merge(a, size, temp);
} // threads > 1
}

```

ภายในฟังก์ชัน `mergesort_omp()` พารามิเตอร์ `threads` รับค่าตั้งต้นจากตัวแปร `num_threads` แบ่งการทำงานเป็น 2 กรณี คือ

- กรณี `threads > 1`

ขั้นตอนการแบ่ง (Divide) เริ่มต้นที่ประโภคโครงสร้าง `#pragma omp parallel sections` ซึ่งจะเกิด Parallel Region ขึ้น เฮอดหลักจะสร้างเฮอดเสริมขึ้นมารวม 2 เฮอดตามประโภค `#pragma omp section` ทั้งสองประโภค เพื่อทำงานฟังก์ชัน `mergesort_omp()` อาจเรียกชื่อ `มูลด้านซ้ายและด้านขวาตามลำดับ` การที่ `ヘอด` (หลักหรือ `ヘอดเสริม`) เรียกฟังก์ชันแต่ละรอบตัวแปร `threads` จะมีค่าลดลงเหลือประมาณครึ่งหนึ่งของค่าเดิม และ `ヘอด` (หลักหรือ `ヘอดเสริม`) จะเรียกฟังก์ชัน `mergesort_omp()` ซ้ำจนค่าของตัวแปร `threads = 1`

- กรณี `threads = 1`

`ヘอด` (หลักหรือ `ヘอดเสริม`) ที่มีค่านี้จะเปลี่ยนไปเรียกฟังก์ชัน `mergesort_serial()` แทน

```

void mergesort_serial(int a[], int size, int temp[]) {
    int i;
    if (size == 2) {
        if (a[0] <= a[1])
            return;
        else {
            SWAP(a[0], a[1]);
            return;
        }
    }
    mergesort_serial(a, size/2, temp);
    mergesort_serial(a + size/2, size - size/2, temp);
    merge(a, size, temp);
}

```

ฟังก์ชัน `mergesort_serial()` แบ่งอาเรย์ที่รับมาทีละครึ่ง เช่นเดียวกับ `mergesort_omp` แต่จะเรียกใช้ฟังก์ชัน `mergesort_serial()` ซ้ำอีกเพื่อเรียงข้อมูลด้านซ้ายก่อนจนตัวแปร `size = 2` และจึงกระทำกับข้อมูลด้านขวา

หากเราดูแผนภูมิการทำงานของฟังก์ชัน `mergesort_serial()` จะมีลักษณะเหมือนแผนภูมิต้นไม้ (Tree) ในรูปที่ 8.7 ลักษณะการทำงานแบบเรียกซ้ำของ ฟังก์ชัน `mergesort_serial()` เปรียบเหมือนการเดินทางบนแผนภูมิต้นไม้แบบเชิงลึกก่อน หรือ **Depth-First Traversal**

```
void merge(int a[], int size, int temp[]) {
    int i1 = 0, it = 0;
    int i2 = size / 2;
    while(i1 < size/2 && i2 < size) {
        if (a[i1] <= a[i2]) {
            temp[it] = a[i1];
            i1 += 1;
        } else {
            temp[it] = a[i2];
            i2 += 1;
        }
        it += 1;
    }
    while (i1 < size/2) {
        temp[it] = a[i1];
        i1++; it++;
    }
    while (i2 < size) {
        temp[it] = a[i2];
        i2++; it++;
    }
    memcpy(a, temp, size*sizeof(int)); // copy temp to a
}
```

ขั้นตอนการยึดครอง (Conquer) เริ่มต้นเมื่อฟังก์ชัน `merge()` ทำหน้าที่ประสานข้อมูลขนาดเล็กๆ ที่ฟังก์ชัน `mergesort_serial()` ได้แบ่งไว้ก่อนหน้า และจึงรีเทิร์นย้อนกลับไปประสาน (Merge) ข้อมูลขนาดใหญ่ขึ้นๆ ที่ฟังก์ชัน `mergesort_omp()` ได้แบ่งไว้ หลังจากนั้นจึงดำเนินมาข้อมูลที่เรียงแล้วจาก `temp` ไปยังอาร์เรย์ `a` ด้วย ฟังก์ชัน `memcpy()` เป็นจำนวนไบต์เท่ากับ  $\text{size} \times \text{sizeof(int)}$  (ขนาดของตัวแปร `int` เท่ากับ 4 ไบต์)

งานวิจัยที่เกี่ยวข้อง การเรียงข้อมูลแบบขนานของอัลกอริธึม QuickSort โดยผู้เขียน ได้แก่

- Parallel Partition and Merge QuickSort หรือ **PPMQSort** ที่มา: [Ranokpanuwat and Kittitornkun \(2016\)](#) ซึ่งพัฒนาด้วยภาษา C มีความซับซ้อนเฉลี่วเวลา (Run Time Complexity) เท่ากับ  $O(\frac{N}{c} \log \frac{N}{2c} + N)$  โดยที่  $N$  และ  $c$  คือ ขนาดของอาร์เรย์ข้อมูล และจำนวนแกนประมวลผล ตามลำดับ และ
- MultiStack Parallel Partition Sorting หรือ **MSPSort** ที่มา: [Rattanatranurak and Kittitornkun \(2020\)](#) ซึ่งพัฒนาด้วยภาษา C++ และผู้เขียนทำการวิจัยและพัฒนาเพิ่มเติม เพื่อให้รองรับการทำงานบน

ชีพิญมัลติคอร์ ARM Cortex A72 บนบอร์ด Raspberry Pi4 ที่มา: [Rattanatranurak and Kittitornkun \(2021\)](#)

## 8.4 สรุปท้ายบท

การคำนวณแบบอนุกรมมาจากขั้นตอนการคิดพื้นฐานของมนุษย์ แต่ปัญหาด้านต่างๆ ที่ซับซ้อน และปริมาณข้อมูลที่เพิ่มมากขึ้นเรื่อยๆ จึงต้องอาศัยคอมพิวเตอร์ช่วยคำนวณ ทำให้วิัฒนาการของคอมพิวเตอร์โน้มเอ่นมาเพื่อรองรับการคำนวณแบบขนานมากขึ้น คอมพิวเตอร์แบบขนานมี 2 ชนิดแบ่งตามการใช้งานหน่วยความจำหลัก คือ คอมพิวเตอร์แบบขนานชนิดมัลติคอร์และชนิดแมนีคอร์ซึ่งใช้หน่วยความจำภายในพาร์เมร์กัน (Shared Memory) และคอมพิวเตอร์แบบขนานชนิดมัลติคอมพิวเตอร์มีหน่วยความจำอิสระจากกัน ประกอบด้วยเครื่องคอมพิวเตอร์ชนิดมัลติคอร์และแมนีคอร์จำนวนมาก เชื่อมต่อ กันด้วยเครือข่ายความเร็วสูง การสังงานเครื่องคอมพิวเตอร์ทั้งสองชนิดนี้ ด้วยภาษาคอมพิวเตอร์ระดับสูง เช่น C/C++, Java, Python เป็นต้น ต้องเปลี่ยนขบวนการคิดแบบอนุกรมเป็นแบบขนานและใช้ประโยชน์จากความขนาดของข้อมูล ความขนาดของงานย่อย และความขนาดแบบผสม พัฒนาร่วมกับไลบรารี OpenMP และไลบรารี MPI (Message Passing Interface) ซึ่งเป็นที่ยอมรับและนิยมทั่วโลกรวมถึงชูเปอร์คอมพิวเตอร์ Fugaku

## 8.5 คำถามท้ายบท

1. จงเปรียบเทียบคำนวณแบบขนานกับการทำบ้านจำนวนหลายข้อ โดยนักศึกษาหลายคนช่วยกันทำโดย
  - 1 คนรับผิดชอบ 1 ข้อ และนำคำตอบมารวมกัน
  - 2 คนช่วยกันทำการบ้านทีละข้อ เพื่อหาคำตอบที่ดีที่สุด และนำคำตอบมารวมกัน
  - ทุกคนช่วยกันทำการบ้านทีละข้อ เพื่อหาคำตอบที่ดีที่สุด
 วิธีไหนจะใช้เวลาต่างกันเท่าไหร่และมี ข้อดี ข้อเสียต่างกันอย่างไร
2. จงบอกความสัมพันธ์ระหว่างจำนวนเรตและจำนวนแกนประมวลผล
3. จงอธิบายวิธีการแบ่งงาน (Share Work) ของไลบรารี OpenMP
4. จงค้นคว้า [กฎของ Amdahl](#) ในอินเทอร์เน็ต เพื่อนำมาประยุกต์ใช้กับการคำนวณค่า  $Speedup(n)$  ของการคุณเมทริกซ์แบบขนาน
5. จงค้นคว้าว่าคอมพิวเตอร์ชนิด Shared Memory Multiprocessor มีปรากฏอยู่ในหัวข้อใดใน[อนุกรมวิธานของ Flynn](#) (Flynn's Taxonomy)

## ภาคผนวก (Appendices)

## ภาคผนวก A

### การทดลองที่ 1 ข้อมูลและคณิตศาสตร์ในคอมพิวเตอร์

การทดลองนี้เป็นการทบทวนความเข้าใจและแบบฝึกหัดเสริมของเนื้อหาในบทที่ 2 เนื่องจากจำนวนบิตข้อมูลที่ยาวขึ้นจำเป็นต้องใช้โปรแกรมคอมพิวเตอร์ช่วยคำนวณแทน โดยมีรัตตุประสงค์ ดังต่อไปนี้

- เพื่อให้เข้าใจ การแปลง และ คณิตศาสตร์ สำหรับ เลขจำนวนเต็มฐานสอง ชนิดไม่มีเครื่องหมาย และ มีเครื่องหมายแบบ 2's Complement
- เพื่อให้เข้าใจการแปลงและคณิตศาสตร์สำหรับเลขทศนิยมฐานสองมาตรฐาน IEEE754 ชนิด Single Precision
- เพื่อให้เข้าใจรหัส ASCII และ Unicode สำหรับข้อมูลตัวอักษร

นอกจากเนื้อหาในบทที่ 2 แล้ว ผู้อ่านสามารถศึกษาเว็บเพจเพิ่มเติม เพื่อทำความเข้าใจอย่างลึกซึ้ง ได้แก่

- [https://www.tutorialspoint.com/cprogramming/c\\_data\\_types.htm](https://www.tutorialspoint.com/cprogramming/c_data_types.htm)
- <https://www3.ntu.edu.sg/home/ehchua/programming/java/datarepresentation.html>

ผู้อ่านจะพบว่าเนื้อหาในเว็บของมหาวิทยาลัยนันยาง ประเทศสิงคโปร์ เป็นการสอนพื้นภาษา Java ใช้งานข้อมูลเป็นเลขฐานสองเหมือนกับภาษา C/C++ ในเว็บที่สอง การทดลองจะครอบคลุมเนื้อหาตามทฤษฎี โดยจะเริ่มจากเลขจำนวนเต็ม เลขทศนิยม และตัวอักษรตามลำดับ

## A.1 การแปลงและคณิตศาสตร์สำหรับเลขจำนวนเต็มฐานสอง

### A.1.1 การทดลองแปลงเลขฐาน

เนื่องจากการแปลงเลขฐานสิบเป็นฐานสองชนิดไม่มีเครื่องหมาย (unsigned) ผู้อ่านสามารถใช้เครื่องคิดเลขทางวิทยาศาสตร์ทั่วไป ดังนี้ การทดลองนี้จะเน้นที่การแปลงเป็นเลขจำนวนเต็มฐานสองชนิดมีเครื่องหมายแบบ 2's Complement สอดคล้องกับเนื้อหาในหัวข้อที่ 2.2 โดยผ่านเว็บเบราว์เซอร์ที่ผู้อ่านถนัด คลิกที่ชื่อลิงก์ต่อไปนี้ <https://www.binaryconvert.com/> ขอให้ผู้อ่านปฏิบัติตามการทดลอง ดังนี้

TYPE	BITS	MINIMUM	MAXIMUM	DECIMAL FORMAT
<b>Convert</b> Unsigned char	8	0	255	Integer
<b>Convert</b> Signed char	8	-128	127	Integer
<b>Convert</b> Unsigned short	16	0	65535	Integer
<b>Convert</b> Signed short	16	-32768	32767	Integer
<b>Convert</b> Unsigned int	32	0	4294967295	Integer

รูปที่ A.1: หน้าเว็บสำหรับแปลงเลขจำนวนเต็มฐานสองเป็นฐานสิบหรือฐานสิบเป็นฐานสองหลายชนิด

- คลิกที่หัวข้อ Signed Char เพื่อทดลองการแปลงเลขจำนวนเต็มมีเครื่องหมายชนิด 2's Complement ขนาด 8 บิต
- กรอกเลข -123 ลงในกล่องข้อความ Decimal เพื่อให้โปรแกรมแปลงเลขจำนวนเต็ม -123 เป็นเลขฐานสองมีเครื่องหมายชนิด 2's Complement ดังรูปที่ A.2

**Signed char (8-bit) Two's complement**

**Decimal**

**Convert to binary**

**Convert to decimal**

**Binary**

**Hexadecimal**

**0x**  =   
Binary  
█ █ █ █ █ █ █ █

รูปที่ A.2: กรอกเลข -123 ลงในกล่องข้อความ Decimal เพื่อให้โปรแกรมแปลงเลขจำนวนเต็ม -123 เป็นเลขจำนวนเต็มฐานสองมีเครื่องหมายชนิด 2's Complement

กดปุ่ม Convert to binary เพื่อดำเนินการ บันทึกผลลัพธ์ที่ได้จากการแปลงดังต่อไปนี้

**Signed char (8-bit) Two's complement**

**Decimal**

**-123**

Most accurate representation = -123

**New conversion**

**Binary**

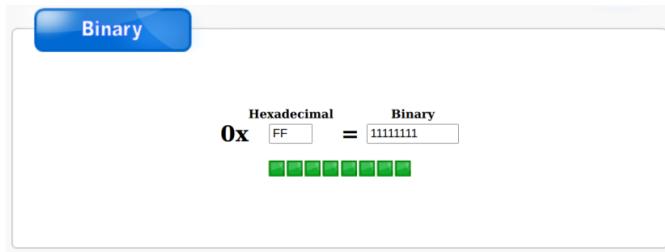
**0x85 = 10000101**

█ █ █ █ █ █ █ █

รูปที่ A.3: ผลลัพธ์การแปลงเลข -123 เป็นเลขจำนวนเต็มฐานสองมีเครื่องหมายชนิด 2's Complement

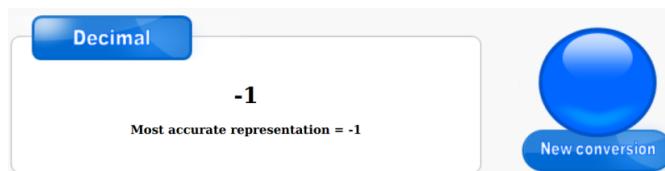
- Binary (2's Complement) \_ \_ \_ \_ \_ \_ \_ \_
- Hexadecimal (0x) \_ \_
- แสดงวิธีทำตามสมการที่ (2.16) ที่  $n=8$  บิตเพื่อแปลงเลขให้ตรงตามรูป

3. กรอกเลขฐานสองมีเครื่องหมายชนิด 2's Complement 11111111 ขนาด 8 บิตลงในกล่องข้อความ Binary เพื่อให้โปรแกรมแปลงเลขจำนวนเต็มฐานสิบ ดังรูปที่



รูปที่ A.4: การแปลงเลขฐานสองมีเครื่องหมายชนิด 2's Complement 11111111 หรือเท่ากับฐานสิบหก 0xFF

กดปุ่ม Convert to decimal ทางด้านขวาเพื่อดำเนินการ อ่านค่าผลลัพธ์ที่ได้จากการแปลงดังต่อไปนี้



รูปที่ A.5: ผลลัพธ์การแปลงเลขฐานสองมีเครื่องหมายชนิด 2's Complement 11111111 หรือเท่ากับฐานสิบหก 0xFF

4. กดปุ่ม Signed short บนเมนูด้านบนสุด เพื่อเปลี่ยนความยาวเป็น 16 บิต กรอกเลข -123 ลงในกล่องข้อความ Decimal Convert to binary เพื่อดำเนินการ บันทึกผลลัพธ์ที่ได้จากการแปลงดังต่อไปนี้

- Binary (2's Complement) \_\_\_\_\_
- Hexadecimal (0x) \_\_\_\_\_
- แสดงวิธีทำตามสมการที่ (2.16) ที่  $n=16$  บิตเพื่อแปลงเลขให้ตรงตามรูป

5. กดปุ่ม Signed int บนเมนูด้านบนสุด เพื่อเปลี่ยนความยาวเป็น 32 บิต กรอกเลข -123 ลงในกล่อง ข้อความ Decimal กดปุ่ม Convert to binary เพื่อคำนวณ การ บันทึกผลลัพธ์ที่ได้จากการแปลงดังต่อไปนี้

- Binary \_\_\_\_\_
- Hexadecimal (0x) \_\_\_\_\_
- แสดงวิธีทำตามสมการที่ (2.16) ที่  $n=32$  บิตเพื่อแปลงเลขให้ตรงตามรูป

### A.1.2 คณิตศาสตร์เลขจำนวนเต็มฐานสอง

1. กรอกเลขที่ได้จากการแปลงลงในช่องว่างที่จัดไว้ แสดงวิธีทำการบวกเลขจำนวนเต็มฐานสองมีเครื่องหมายชนิด 2's Complement ขนาด 8 บิตและคำนวนค่าโอลูร์ฟล์  $V$

	$c_8$	$c_7$	$c_6$	$c_5$	$c_4$	$c_3$	$c_2$	$c_1$	$c_0$	$V = c_8 \oplus c_7$
	—	—	—	—	—	—	—	—	—	0
$X$	-123	—	—	—	—	—	—	—	—	$V = \underline{\quad} \oplus \underline{\quad}$
$+Y$	+ -1	+	—	—	—	—	—	—	—	
$Z$	—	—	—	—	—	—	—	—	—	

ซอฟต์แวร์สามารถนำผลลัพธ์  $Z$  ไปใช้งานต่อได้หรือไม่ เพราจะเหตุใด

2. กรอกเลขที่ได้จากการแปลงลงในช่องว่างที่จัดไว้ แสดงวิธีทำการบวกเลขจำนวนเต็มฐานสองมีเครื่องหมายชนิด 2's Complement ขนาด 8 บิตและคำนวนค่าโอลูร์ฟล์  $V$

	$c_8$	$c_7$	$c_6$	$c_5$	$c_4$	$c_3$	$c_2$	$c_1$	$c_0$	$V = c_8 \oplus c_7$
	—	—	—	—	—	—	—	—	—	0
$X$	-123	—	—	—	—	—	—	—	—	$V = \underline{\quad} \oplus \underline{\quad}$
$+Y$	+ -123	+	—	—	—	—	—	—	—	
$Z$	—	—	—	—	—	—	—	—	—	

ซอฟต์แวร์สามารถนำผลลัพธ์  $Z$  ไปใช้งานต่อได้หรือไม่ เพราจะเหตุใด

3. กรอกเลขที่ได้จากการแปลงลงในช่องว่างที่จัดไว้ แสดงวิธีทำการบวกเลขจำนวนเต็มฐานสองมีเครื่องหมายชนิด 2's Complement ขนาด 8 บิตและคำนวนค่าโอลูร์ฟล์  $V$

	$c_8$	$c_7$	$c_6$	$c_5$	$c_4$	$c_3$	$c_2$	$c_1$	$c_0$	$V = c_8 \oplus c_7$
	—	—	—	—	—	—	—	—	—	0
$X$	-123	—	—	—	—	—	—	—	—	$V = \underline{\quad} \oplus \underline{\quad}$
$+Y$	+ 1	+	—	—	—	—	—	—	—	
$Z$	—	—	—	—	—	—	—	—	—	

ซอฟต์แวร์สามารถนำผลลัพธ์  $Z$  ไปใช้งานต่อได้หรือไม่ เพราจะเหตุใด

4. กรอกเลขที่ได้จากการแปลงลงในช่องว่างที่จัดไว้ แสดงวิธีทำการบวกเลขจำนวนเต็มฐานสองมีเครื่องหมายชนิด 2's Complement ขนาด 8 บิตและคำนวนค่าโอลูร์ฟล์  $V$

	$c_8$	$c_7$	$c_6$	$c_5$	$c_4$	$c_3$	$c_2$	$c_1$	$c_0$	$V = c_8 \oplus c_7$
	—	—	—	—	—	—	—	—	—	0
$X$	-123	—	—	—	—	—	—	—	—	$V = \underline{\quad} \oplus \underline{\quad}$
$+Y$	+ 123	—	—	—	—	—	—	—	—	
$Z$	—	—	—	—	—	—	—	—	—	

ซอฟต์แวร์สามารถนำผลลัพธ์  $Z$  ไปใช้งานต่อได้หรือไม่ เพราจะเหตุใด

### A.1.3 กิจกรรมท้ายการทดลอง

จงทำการทดลองและตอบคำถามต่อไปนี้ โดยแสดงวิธีทำตามเนื้อหาในหัวข้อที่ 2.2.2 และตรวจคำตอบตามวิธีทำการทดลองที่ได้ทำไป

1. จงแปลงเลขจำนวนเต็มฐานสิบชนิดไม่มีเครื่องหมายต่อไปนี้ให้เป็นเลขจำนวนเต็มฐานสอง 16 บิตและฐานสิบหกจำนวน 4 หลัก และบันทึกผลลัพธ์ที่ได้ลงในตาราง

ฐานสิบ	ฐานสอง	ฐานสิบหก
7	-----2	-----16
8	-----2	-----16
15	-----2	-----16
16	-----2	-----16
255	-----2	-----16
256	-----2	-----16
65535	-----2	-----16
65536	-----2	-----16

2. จงแปลงเลขจำนวนเต็มฐานสิบต่อไปนี้ให้เป็นเลขจำนวนเต็มฐานสองและฐานสิบหกชนิดมีเครื่องหมายแบบ 2's Complement ความยาว 16 บิตแล้วบันทึกผลลัพธ์ที่ได้ลงในตาราง

ฐานสิบ	ฐานสอง	ฐานสิบหก
+1	-----2	-----16
-1	-----2	-----16
+15	-----2	-----16
-16	-----2	-----16
+255	-----2	-----16
-256	-----2	-----16
+65535	-----2	-----16
-65536	-----2	-----16

3. จงบวกเลข 2's Complement ต่อไปนี้ แล้วบันทึกผลลัพธ์เป็นฐานสองความยาว 16 บิต ฐานสิบหกฐานสิบ โอเวอร์โฟล์วหรือไม่ และอธิบายเหตุผลว่าทำไมจึงไม่ตรงกัน

- $1000000000000000 + 0000000000000001$

- ผลลัพธ์ = -----2
- ผลลัพธ์ = -----16
- ผลลัพธ์ = -----10
- โอเวอร์โฟล์วหรือไม่.....
- เหตุผล.....

- $1000000000000000 + 1000000000000000$

- ผลลัพธ์ = -----2
- ผลลัพธ์ = -----16

- ผลลัพธ์ = \_\_\_\_\_<sup>10</sup>
- โอเวอร์เฟล์วหรือไม่.....
- เหตุผล.....
- 1000000000000000 - 0000000000000001
  - ผลลัพธ์ = \_\_\_\_\_<sup>2</sup>
  - ผลลัพธ์ = \_\_\_\_\_<sup>16</sup>
  - ผลลัพธ์ = \_\_\_\_\_<sup>10</sup>
  - โอเวอร์เฟล์วหรือไม่.....
  - เหตุผล.....
- 1000000000000000 - 1000000000000000
  - ผลลัพธ์ = \_\_\_\_\_<sup>2</sup>
  - ผลลัพธ์ = \_\_\_\_\_<sup>16</sup>
  - ผลลัพธ์ = \_\_\_\_\_<sup>10</sup>
  - โอเวอร์เฟล์วหรือไม่.....
  - เหตุผล.....

## A.2 การแปลงและคณิตศาสตร์เลขทศนิยมฐานสองมาตรฐาน IEEE754

การทดลองเพื่อให้เข้าใจการแปลงเลขทศนิยมฐานสิบให้เป็นเลขฐานสองตามรูปแบบและฝึกการคำนวณโดยใช้คณิตศาสตร์มาตรฐาน IEEE754 Single Precision มีความสอดคล้องกับเนื้อหาในหัวข้อที่ 2.6

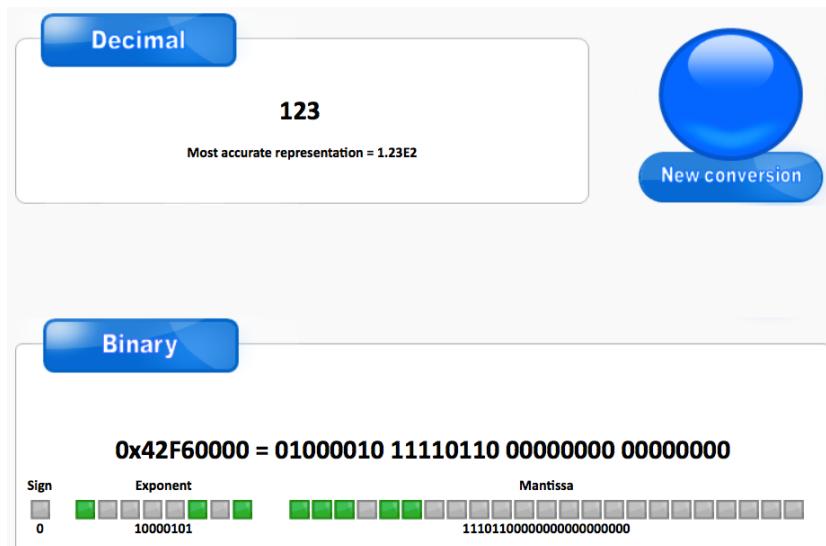
### A.2.1 เลขทศนิยมชนิดจุดลอยตัวมาตรฐาน IEEE754 Single-Precision

การทดลองนี้จะเน้นที่การแปลงเลขทศนิยมฐานสิบให้เป็นเลขทศนิยมฐานสองชนิดจุดลอยตัว สอดคล้องกับเนื้อหาในหัวข้อที่ 2.6 ในรูปแบบ Single Precision โดยผ่านเว็บเบราว์เซอร์ที่ผู้อ่านสนใจ คลิกที่ชื่อลิงก์ต่อไปนี้

[http://www.binaryconvert.com/convert\\_float.html](http://www.binaryconvert.com/convert_float.html)

เมื่อเว็บเพจปรากฏขึ้น ขอให้ผู้อ่านปฏิบัติตามการทดลอง ดังนี้

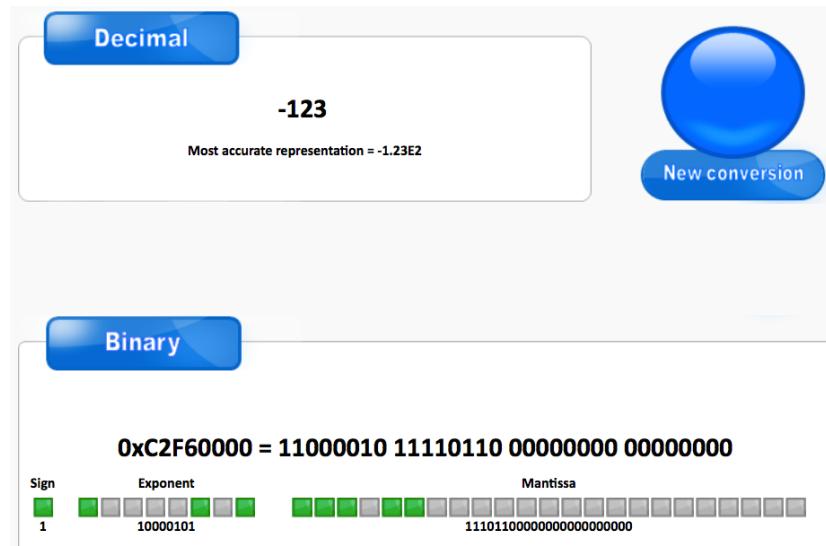
- กรอกเลข 123 ลงในกล่องข้อความ และกดปุ่ม Convert to binary ได้รูปที่ A.6



รูปที่ A.6: ผลลัพธ์จากการแปลงเลข 123.0 ให้เป็นเลขทศนิยมฐานสองชนิด Single Precision

การเรียงตัวของผลลัพธ์เลขฐานสิบหากทางซ้ายมีมาจากเลขฐานสองทางขวาเมื่อ ซึ่งเกิดจากบิตข้อมูลทั้งหมด 32 บิตตามรูปแบบของมาตรฐาน IEEE754 ชนิด Single Precision โปรดสังเกต กล่องสีเหลืองสีเขียวตรงกับบิตที่เป็น '1' กล่องสีเทาตรงกับบิตที่เป็น '0' 0x หมายถึง เลขฐานสิบหากแสดงวิธีทำตามสมการที่ (2.67) เพื่อแปลงเลขให้ตรงตามรูป

2. กรอกเลข -123.0 ลงในกล่องข้อความ และกดปุ่ม Convert to binary ได้รูปที่ A.7



รูปที่ A.7: ผลลัพธ์จากการแปลงเลข -123.0 ให้เป็นเลขทศนิยมฐานสองตามมาตรฐาน IEEE754 ชนิด Single Precision

โปรดสังเกตตำแหน่งของกล่องสีเหลี่ยมหรือสีเทาที่ตรงกับบิต Sign Exponent และ Mantissa ดังนั้น เราจะเห็นได้ว่าเฉพาะ Sign ที่มีการเปลี่ยนแปลง

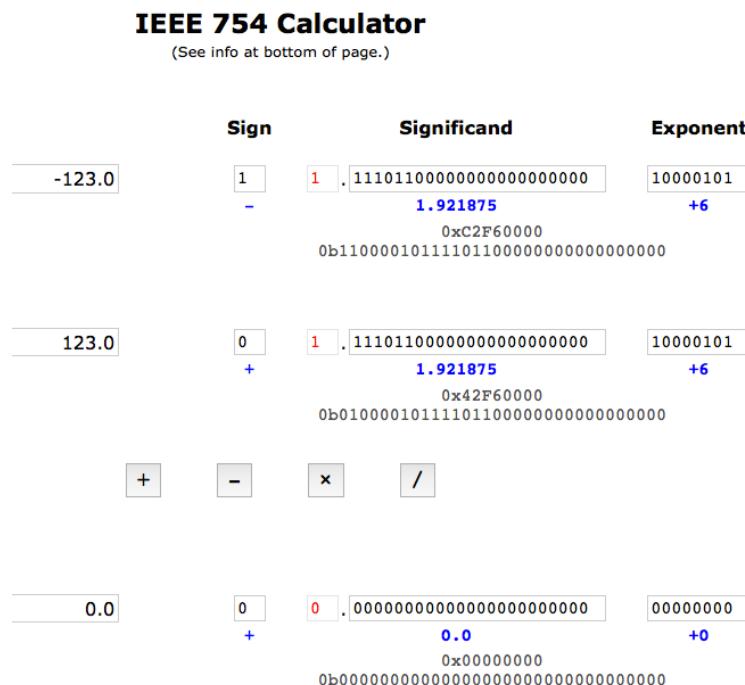
แสดงวิธีทำการสมการที่ (2.67) เพื่อแปลงเลขให้ตรงตามรูป

3. คลิกบนลิงก์นี้ เพื่อทดลองบวกและคูณเลขในรูปแบบ Single Precision ด้วยลิงก์ต่อไปนี้ <http://weitz.de/ieee/> เลื่อนหน้าเว็บลงไปด้านล่างสุด เพื่อค้นหาແບມenuตามรูปที่ A.8 แล้วกดเลือกปุ่ม binary32 เพื่อทดลองการบวกและคูณเลข IEEE754 Single Precision

[binary16](#) [binary32](#) [binary64](#) [binary128](#)

รูปที่ A.8: เมนูด้านล่างสุดของหน้าเว็บ เพื่อเลือกเลขทศนิยมฐานสองชนิด IEEE754 Single Precision (Binary32) และ Double Precision (Binary64)

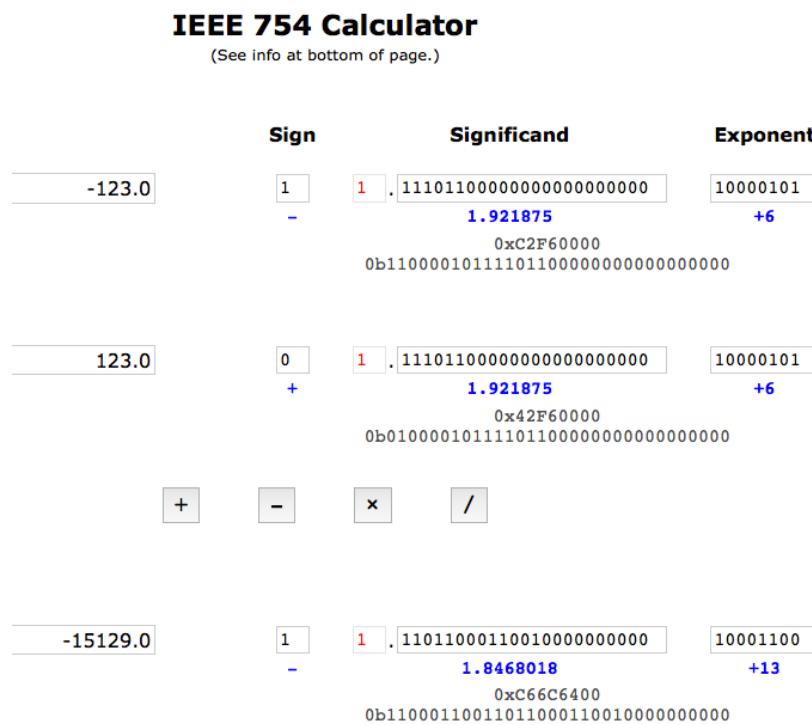
4. เลื่อนหน้าเว็บกลับไปด้านบนสุดเพื่อกรอกเลข -123.0 ลงในกล่องข้อความซ้ายบน และ กรอกเลข 123.0 ลงในกล่องข้อความถัดลงมา แล้วกดปุ่ม + และจะได้ผลลัพธ์ดังรูปต่อไปนี้



รูปที่ A.9: ผลลัพธ์จากการบวกเลข -123.0+123.0 ให้เป็นเลขทศนิยมฐานสองชนิด IEEE754 Single Precision

จะสังเกตเห็นว่า ผลลัพธ์ที่ได้เรียกว่า True Zero ตามตารางที่ 2.12

5. กดปุ่ม x (คูณ) และจะได้ผลลัพธ์ของ  $-123 \times 123$  ดังรูปต่อไปนี้



รูปที่ A.10: ผลลัพธ์จากการคูณเลข  $-123.0 \times 123.0$  ให้เป็นเลขทศนิยมฐานสองชนิด IEEE754 Single Precision

แสดงวิธีทำตามสมการที่ (2.67) เพื่อแปลงเลขให้ตรงตามผลคูณในรูปที่ A.10

#### A.2.2 กิจกรรมท้ายการทดลอง

จงใช้ลิงก์ของเว็บเพจต่อไปนี้ในการตอบคำถาม

<https://www.h-schmidt.net/FloatConverter/IEEE754.html>

รูปที่ A.11: เว็บสำหรับการตอบคำถามเพื่อสร้างเลขหรือแปลงเลขฐานสิบด้วยมาตรฐาน IEEE754 Single Precision การกดเลือกคือทำให้ปิดนั้นเท่ากับ '1'

โดยแสดงวิธีทำตามเนื้อหาในหัวข้อที่ 2.6 และตรวจคำตอบตามวิธีทำการทดลองที่ได้ทำไป และบันทึกผลลัพธ์ลงบนเส้นประที่จัดไว้ให้เท่านั้น ผู้อ่านสามารถกดเปลี่ยนเครื่องหมายถูก ซึ่งแทนล็อก 1 หากไม่มีเครื่องหมายถูกแทนล็อก 0 ยกตัวอย่างเช่น

1. จงสร้างเลขทศนิยมฐานสอง IEEE754 ที่มีค่าเท่ากับ  $-0.0_{10}$  โดยการกดเลือกปุ่มสีเหลืองในส่วน Sign ท่านั้น

ค่าฐานสิบที่จัดเก็บ (Value actually stored in float).....

ความผิดพลาด (Error due to conversion).....

2. จงสร้างเลขศนย์มฐานสอง IEEE754 ที่มีค่าเท่ากับ  $-1.0_{10}$  โดยการกดเลือกปุ่มสีเหลี่ยมในส่วน Exponent ท่านั้น ต่อจากข้อที่แล้ว

ເລຂົ້ານສອງ = \_\_\_\_\_ 2

ଜ୍ଞାନସିବହୁ = —————— —————— —————— —————— —————— —————— —————— —————— 16

ค่าฐานสิบที่จัดเก็บ (Value actually stored in float).....

ความผิดพลาด (Error due to conversion).....

3. จงสร้างเลขศนิยมฐานสอง IEEE754 ที่มีค่าเท่ากับ  $-1.55_{10}$  หรือ  $1.55\text{e}0$  โดยการกดเลือกปุ่มสีเหลือง ในส่วน Mantissa เท่านั้น ต่อจากข้อที่แล้ว

เลขฐานสอง =  2  
 ฐานสิบหก =  16  
 ค่าฐานสิบที่จัดเก็บ (Value actually stored in float).....  
 ความผิดพลาด (Error due to conversion).....

4. จงสร้างเลขทศนิยมฐานสอง IEEE754 ที่มีค่าเท่ากับ  $1.17549435082 \times 10^{-38}$  หรือ  $1.17549435082e-38$  ซึ่งเป็นค่าค่านอร์มัลไลซ์ที่น้อยที่สุด (Normalize)

เลขฐานสอง =  2  
 ฐานสิบหก =  16  
 ค่าฐานสิบที่จัดเก็บ (Value actually stored in float).....  
 ความผิดพลาด (Error due to conversion).....

5. จงสร้างเลขทศนิยมฐานสอง IEEE754 ที่มีค่าเท่ากับ  $1.17549421069 \times 10^{-38}$  หรือ  $1.17549421069e-38$  ซึ่งอยู่ในรูป ดีnor์มัลไลซ์ (Denormalize) เพราะมีค่าน้อยกว่าค่าค่านอร์มัลไลซ์ที่ต่ำที่สุด

เลขฐานสอง =  2  
 ฐานสิบหก =  16  
 ค่าฐานสิบที่จัดเก็บ (Value actually stored in float).....  
 ความผิดพลาด (Error due to conversion).....

6. จงสร้างเลขทศนิยมฐานสอง IEEE754 ที่มีค่าเท่ากับ  $1.40129846432 \times 10^{-45}$  หรือ  $1.40129846432e-45$  ซึ่งอยู่ในรูป ดีnor์มัลไลซ์ (Denormalize) และต่ำที่สุด

เลขฐานสอง =  2  
 ฐานสิบหก =  16  
 ค่าฐานสิบที่จัดเก็บ (Value actually stored in float).....  
 ความผิดพลาด (Error due to conversion).....

7. จงสร้างเลขทศนิยมฐานสอง IEEE754 ที่มีค่าเท่ากับ  $1.0 \times 10^{-46}$  หรือ  $1e-46$  ซึ่งอยู่ในรูป ดีnor์มัลไลซ์ (Denormalize) และจัดเก็บด้วยค่า 0.0 แทน

เลขฐานสอง =  2  
 ฐานสิบหก =  16  
 ค่าฐานสิบที่จัดเก็บ (Value actually stored in float).....  
 ความผิดพลาด (Error due to conversion).....

8. จงสร้างเลขทศนิยมฐานสอง IEEE754 ที่มีค่าเท่ากับ  $3.40282346640 \times 10^{38}$  หรือ  $3.40282346640e38$  ซึ่งเป็นค่าค่านอร์มัลไลซ์ที่มากที่สุด

เลขฐานสอง =  2  
 ฐานสิบหก =  16  
 ค่าฐานสิบที่จัดเก็บ (Value actually stored in float).....

ความผิดพลาด (Error due to conversion).....

9. จงสร้างเลขทศนิยมฐานสอง IEEE754 ที่มีค่าเท่ากับ  $3.5 \times 10^{38}$  หรือ 3.5e+38 ซึ่งมากกว่าค่าอนร์มัลໄල์ที่มากที่สุด ซึ่งหมายถึงค่าอนันต์ ( $\infty$ : Infinity) ตามตารางที่ [2.12](#)

เลขฐานสอง =  2

ค่าฐานสิบที่จัดเก็บ (Value actually stored in float).....

ความผิดพลาด (Error due to conversion).....

10. จงสร้างเลขทศนิยมฐานสอง IEEE754 ที่มีค่าเท่ากับ NaN (Not a Number) ตามตารางที่ 2.12

លេខចុះត្រូវសងសម្បត្តិការណ៍ = \_\_\_\_\_ 2

ฐานสิบหก = \_\_\_\_\_ 16

ค่าฐานสิบที่จัดเก็บ (Value actually stored in float).....

ความผิดพลาด (Error due to conversion).....

### A.3 รหัสของข้อมูลตัวอักษร

### A.3.1 การทดสอบ

การทดลองในหัวข้อนี้จะเป็นการแปลงรหัสตัวอักษรภาษาอังกฤษและไทย เป็นรหัส ASCII และ Unicode ชนิด UCS-2 ตามเนื้อหาในหัวข้อ 2.7 ผ่านทางเว็บไซต์ <https://www.branah.com/ascii-converter> ที่มีนักพัฒนาเพื่อเผยแพร่ความรู้เป็นวิทยาทานเช่นเดียวกับเว็บที่ได้ทดลองมา

1. เปิดเว็บตามลิงก์ต่อไปนี้ หรือ กดปุ่มช้ายบนชื่อลิงก์  
<https://www.branah.com/ascii-converter>
  2. กรอกข้อความต่อไปนี้ ลงไปในกล่องข้อความ ASCII  
ไทย ก ข ค a b c  
โปรดสังเกต ระหว่างตัวอักษรเมื่อ ซ่องว่าง 1 ตัวอักษรเสมอ
  3. กดปุ่ม Convert ช้ายบนสุด จะได้ผลลัพธ์ดังรูปต่อไปนี้

### ASCII Converter - Hex, decimal, binary, base64, and ASCII converter

Convert ASCII (Example: a b c)

ไทย ก ข ค a b c

Add spaces Remove spaces  Convert white space characters

Convert Hex (Example: 0x61 0x62 0x63)  Remove 0x

e44 e17 e22 e01 e02 e04 61 62 63

Convert Decimal (Example: 97 98 99)

3652 3607 3618 3585 3586 3588 097 098 099

Convert Binary (Example: 01100001 01100010 01100011)

111001000100 111000010111 111000100010 111000000001 111000000010 111000000100 01100001 01100010  
01100011

Convert Base64 (Example: YSBiIGM=)

RCAxIClqASACIAQgYSBiIGM=

รูปที่ A.12: ผลลัพธ์จากการกรอกและแปลงตัวอักษร ไทย ก ข ค a b c เป็นรหัสต่างๆ

4. กล่องข้อความ Hex จะแสดงค่า Unicode สำหรับภาษาไทย และ ASCII สำหรับภาษาอังกฤษ ในรูปผู้เขียนได้กดเลือก Remove 0x เพื่อความสะดวกในการอ่านค่า

#### A.3.2 กิจกรรมท้ายการทดลอง

1. จงอธิบายวิธีการหาค่าฐานสิบ 0 - 9 จากรหัส ASCII ของตัวอักษร 0 - 9
2. จงอธิบายวิธีการหาค่าฐานสิบ 0 - 9 จากรหัส Unicode ของตัวอักษร ๐ - ๙
3. จงเปิดเว็บที่มีข้อความภาษาไทย เช่น เว็บข่าว แล้วทดลองเปลี่ยนการนำเสนอบนจอเพื่อ View source เช่น Google Chrome ใช้เมนู Tool-> View Source แล้ว Find หรือกดปุ่ม CTRL-F คำว่า charset ว่า มีค่าเท่ากับ utf-8 หรือไม่ เพราะเหตุใด

## ภาคผนวก B

### การทดลองที่ 2 ตัวอย่างการประกอบและติดตั้งบอร์ด Pi

การทดลองนี้เสริมสร้างประสบการณ์ให้ผู้อ่านได้มีโอกาสจับต้องบอร์ดและอุปกรณ์เสริม และสนับสนุนเนื้อหาของหัวข้อที่ [3.1](#) ในส่วนของฮาร์ดแวร์ โดยมีวัตถุประสงค์ ดังต่อไปนี้

- เพื่อให้รู้จักโครงสร้างและรายละเอียดต่างๆ ของบอร์ด Pi
- เพื่อให้เข้าใจลักษณะรับสัญญาณ HDMI หากไม่มีผู้อ่านสามารถใช้สายแปลงที่จำเป็น เช่น แปลงจากสัญญาณ HDMI เป็นสัญญาณ VGA หรือแปลงเป็นสัญญาณ DVI
- เพื่อให้ทดสอบการเชื่อมต่อกับอุปกรณ์อินพุต/เอาต์พุตที่จำเป็น

การทดลองนี้ต้องเป็นตัวอย่างการประกอบบอร์ดเข้ากับกล่อง ซึ่งอาจแตกต่างกันไปตามรายละเอียดของกล่องที่ผู้อ่านจัดหามา และอาจต้องใช้อุปกรณ์อื่นๆ ได้แก่ จอมอนิเตอร์ที่รองรับสัญญาณ HDMI หากไม่มีผู้อ่านสามารถใช้สายแปลงที่จำเป็น เช่น แปลงจากสัญญาณ HDMI เป็นสัญญาณ VGA หรือแปลงเป็นสัญญาณ DVI คีย์บอร์ด เม้าส์ กล้องสำหรับบอร์ด และอุปกรณ์รับสัญญาณรีโมทคอนโทรล ตามความเหมาะสม

บอร์ด Raspberry Pi เป็นคอมพิวเตอร์ขนาดเล็กเท่ากับบัตรเครดิต เริ่มต้นออกแบบและผลิตในประเทศไทย ราชอาณาจักร โดยมูลนิธิ Raspberry Pi Foundation ซึ่งเริ่มต้นจากการเป็นคอมพิวเตอร์ราคาถูกสำหรับการศึกษา บอร์ดมีการประยุกต์ใช้งานเพิ่มขึ้นเรื่อยๆ ตั้งแต่ปี 2012 ทำให้มีการพัฒนามาเรื่อยๆ จนเป็นเจนเนอเรชันที่ 3 และ 4 ในปัจจุบัน โดยมีสองรุ่น คือ โมเดล A และ โมเดล B โมเดล A เป็นอุปกรณ์ที่เรียบง่าย ตันทุนต่ำกว่า โมเดล B ในขณะที่ โมเดล B เหมาะกับการใช้งานเป็นคอมพิวเตอร์ตั้งโต๊ะ และเซิร์ฟเวอร์ เนื่องจากมีการเชื่อมต่อกับเครือข่ายด้วยสายแลนหรือสาย Ethernet และแบบไร้สายได้แก่ WiFi และ Bluetooth

## B.1 รายการอุปกรณ์ฮาร์ดแวร์



รูปที่ B.1: รูปแสดงรายการอุปกรณ์สำหรับประกอบบอร์ด ที่มา: [rs-online.com](http://rs-online.com)

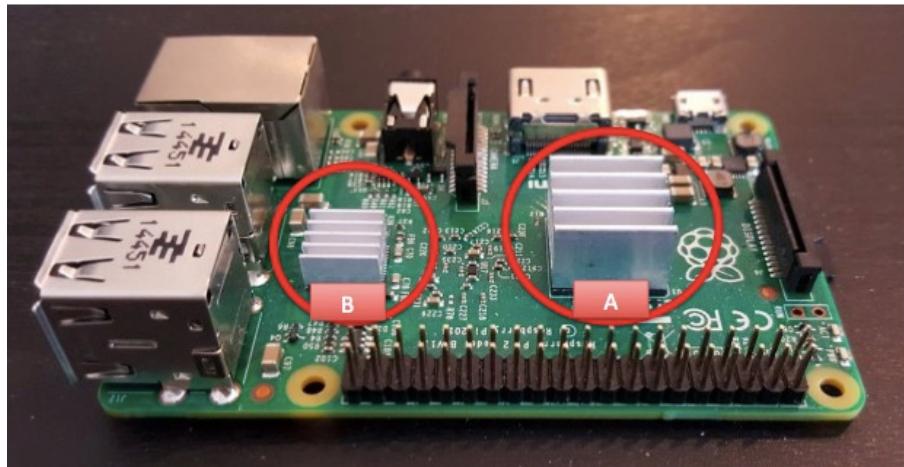
ผู้อ่านสามารถสั่งซื้อชุดทูลคิท (Toolkit) สำหรับบอร์ด Pi ในรูปที่ B.1 ประกอบด้วย รายการดังต่อไปนี้

ลำดับ	ชื่อและรายละเอียด	จำนวน	มี/ไม่มี
1	บอร์ด Pi 3 โมเดล B แรม 1 กิกะไบต์	1	
2	อแดปเตอร์แปลงไฟกระแทกตรง 5.0 โวลต์ 2.5 แอม培ร์ (ปลายสายเป็นหัวไมโคร USB ชนิด B)	1	
3	สายเชื่อมต่อชนิด HDMI	1	
4	แผ่นวงจรโพร์ตอิบอร์ด (Protoboard)	1	
5	หัวขยายการเชื่อมต่อ GPIO สำหรับบอร์ดโมเดล B	1	
6	สายแพ GPIO 40 ขาสำหรับบอร์ดโมเดล B	1	
7	ჸิทซิงก์ขนาดเล็ก	1-2	
8	การ์ดหน่วยความจำไมโคร SD ขนาด 16GB	1	
9	กล่องสำหรับบอร์ด Pi	1 ชุด	
10	หัวเชื่อมต่อสายแพขนาด 40 ขา	1	
11	สายแพขนาด 40 ขาชนิดตัวเมี้ยx2	1	
12	สายต่อวิจารณ์ชนิดตัวเมี้ยx2	1 ชุด	
13	สายแปลง HDMI เป็น VGA	1	

## B.2 ตัวอย่างการประกอบบอร์ด Pi และกล่อง

ตัวอย่างการประกอบบอร์ด Pi เพื่อให้พร้อมใช้งานสำหรับการศึกษาและทดลอง จะต้องเสริมความแข็งแรงให้บอร์ด เพิ่มความสามารถในการระบายความร้อน ต่อขยายขาเข้า/ออกบนบอร์ดให้ยาวขึ้น โดยเรียงลำดับดังนี้

### B.2.1 ประกอบอีทซิงก์ (Heat Sink)



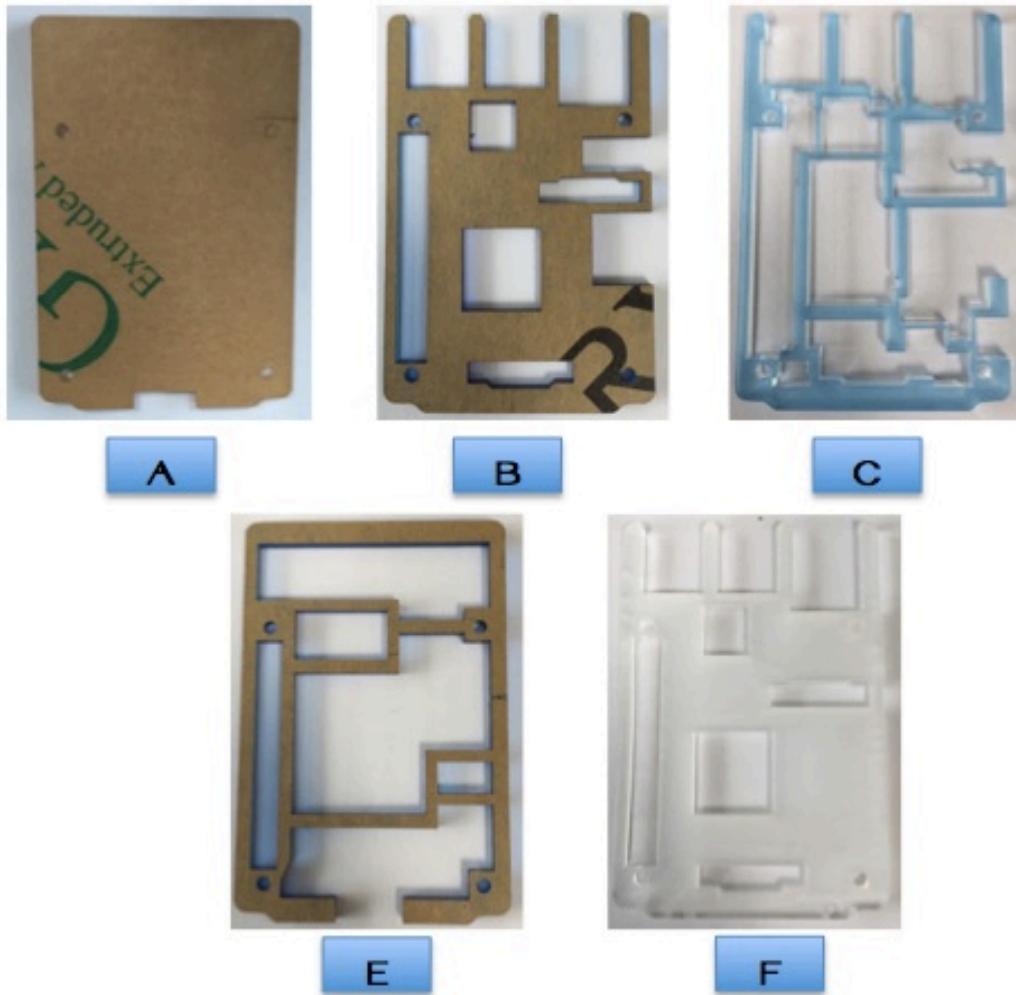
รูปที่ B.2: บอร์ด Pi 3 เมื่อติดอีทซิงก์เรียบร้อยแล้ว

- ตรวจสอบความเรียบร้อยของบอร์ด Pi ที่ได้รับมา หากมีร่องรอยชำรุด หรือ ใหม่ ขอให้แจ้งกับผู้คูณแล
- หยับอีทซิงก์ที่มีขนาดใหญ่ที่สุด แกะแผ่นเคลือบการสองหน้าออก วางด้านที่มีการลงบนชิป BCM2837 ตรงกลาง ณ ตำแหน่ง A ในรูปที่ B.2
- หยับอีทซิงก์ที่มีขนาดกลาง แกะแผ่นเคลือบการสองหน้าออก วางด้านที่มีการลงบนชิป LAN9514 ณ ตำแหน่ง B ในรูปที่ B.2

อีทซิงก์ A ติดบนชิป BCM2837 ขนาดใหญ่กว่า เนื่องจากชิปทำงานที่ความถี่สัญญาณคลีอกสูงกว่า ซึ่งช้อนมากกว่า ในขณะที่อีทซิงก์ B หากมีใหม่ จะติดบนชิป LAN9514 ซึ่งภายในคือ รูทฮับ (Root Hub) ของ USB 2.0 และ Ethernet Controller 10/100 เมกะบิตต่อวินาที

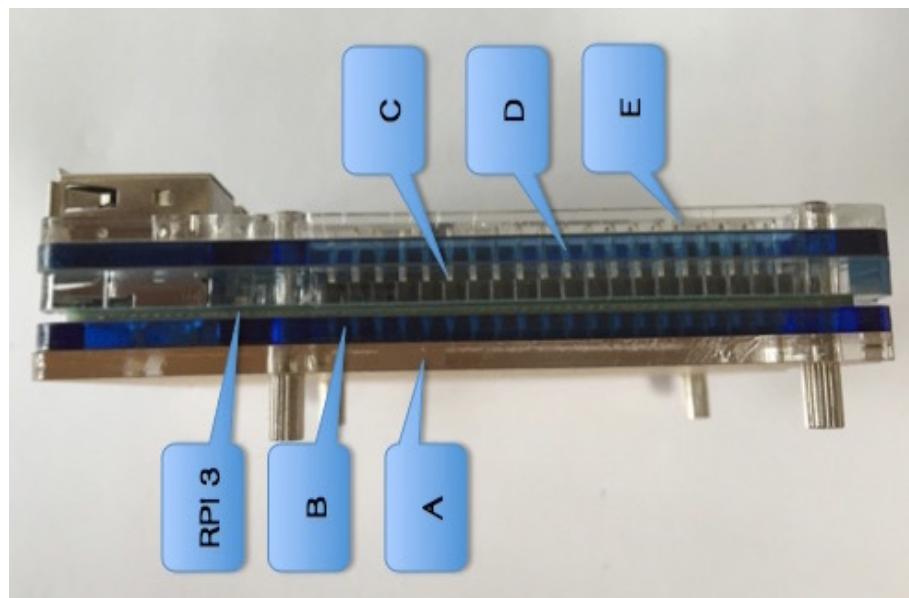
### B.2.2 ประกอบกล่องกับบอร์ด Pi 3

กล่องที่จำหน่ายมาพร้อมบอร์ดของบริษัทแห่งหนึ่ง ประกอบด้วยแผ่นพลาสติก 5 ชิ้น ตามรูปที่ B.3

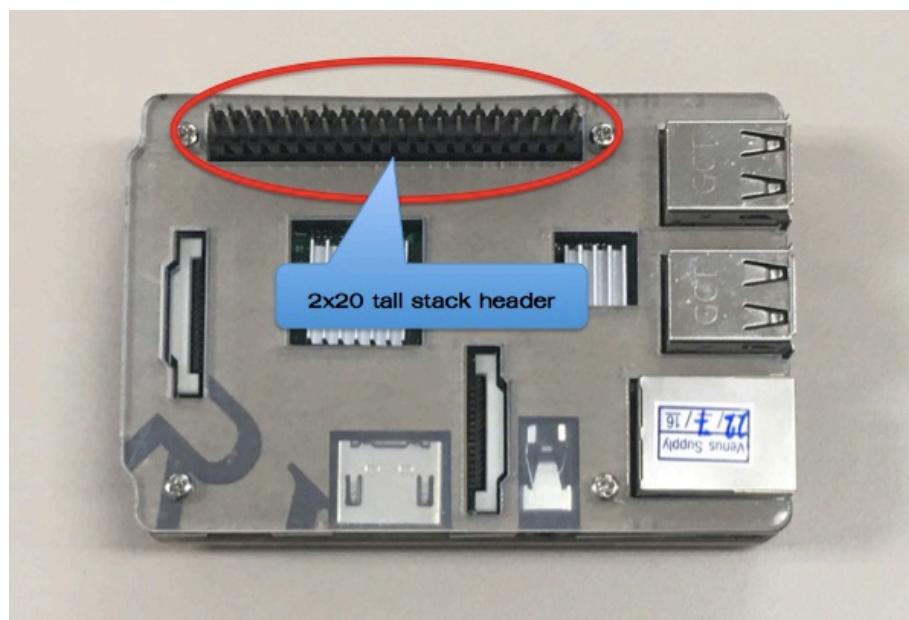


รูปที่ B.3: แผ่นพลาสติก 5 ชิ้น สำหรับประกอบเป็นกล่องของบอร์ด Pi3

1. ประกอบแผ่นพลาสติกและบอร์ด Pi ด้วยกันตามรูปที่ B.4 แผ่น B ประกอบกับบอร์ด Pi ด้านล่าง และช่องบนแผ่น A ซึ่งอยู่ขั้นล่างสุด ประกอบแผ่น C D E เข้าด้วยกัน โดยแผ่น E อยู่ชั้นบนสุด และวางช่องบนบอร์ด
2. ใช้สกรูและขายืดแผ่นพลาสติกและบอร์ด Pi เข้าด้วยกันทั้ง 4 มุก
3. ติดตั้งขาเขื่อนขยายชนิด 2x20 หรือ 2 แคลๆ ละ 20 ขาบนบอร์ด โดยสังเกตจากในรูปที่ B.5 ว่าขาหมายเลข 1 อยู่ตรงมุมซ้ายล่าง



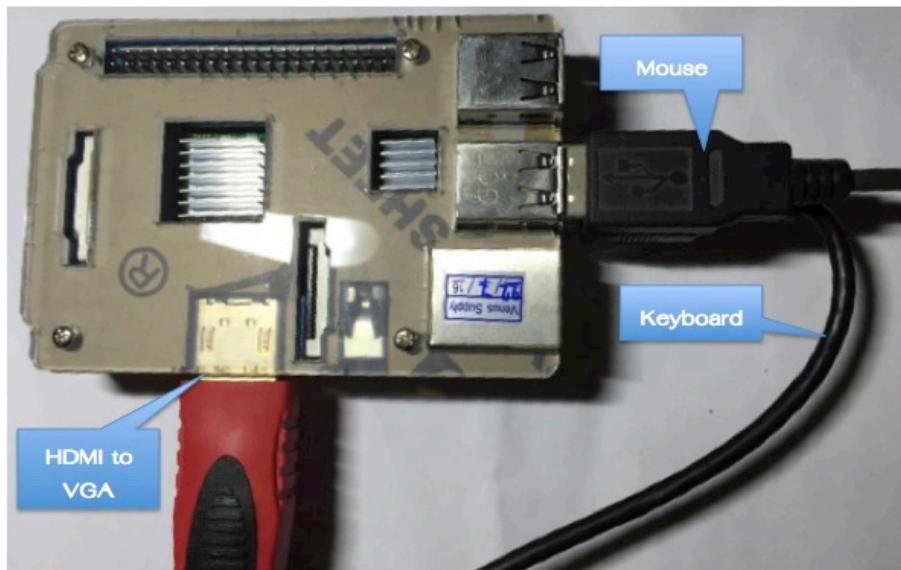
รูปที่ B.4: บอร์ด Pi ที่ประกอบประกอบเรียบร้อยแล้ว



รูปที่ B.5: บอร์ด Pi เมื่อประกอบขาเชื่อมขยาย 2x20

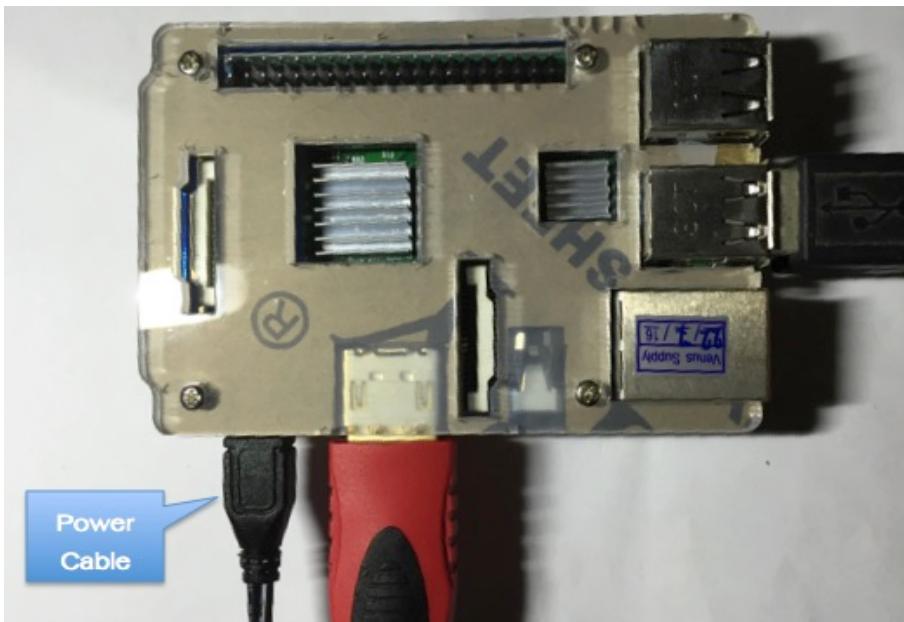
### B.3 เครื่องคอมพิวเตอร์ส่วนบุคคลจากบอร์ด Pi 3 โมเดล B

1. จอดอแดปเตอร์ไฟ 5 โวลต์ จากเต้าเสียบไฟ 220 โวลต์ ก่อนเพื่อความปลอดภัยของผู้ใช้งานและวงจร
2. เชื่อมต่อสายแปลงไมโคร HDMI กับบอร์ด Pi และวิ่งเชื่อมสาย VGA กับจอคอมอนิเตอร์ เลือกอินพุตของจอ เป็น Analog Input
3. เชื่อมต่อคีย์บอร์ดและมาส์กับช่องเสียบสาย USB สีดำบนบอร์ด Pi โปรดสังเกตสัญลักษณ์ของ USB บนหัวสายจะต้องหมายขึ้นดังรูปที่ B.6



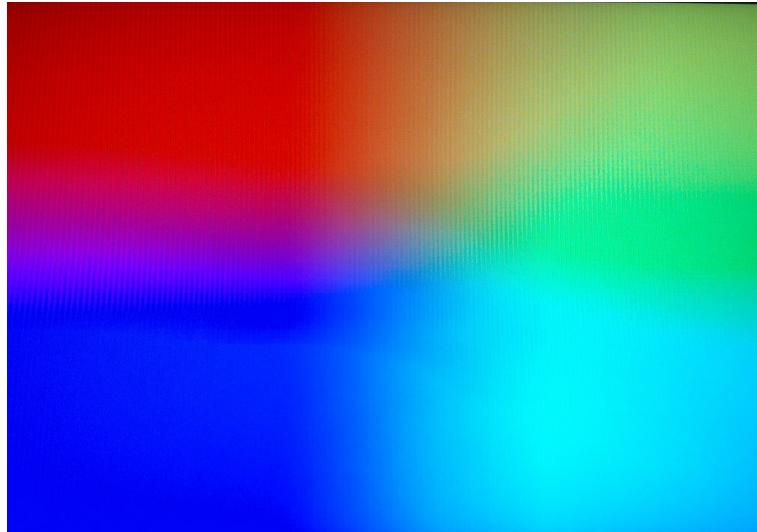
รูปที่ B.6: การเชื่อมต่อคีย์บอร์ดและมาส์กับช่องเสียบสาย USB บนบอร์ด Pi ให้เรียบร้อย

4. เชื่อมต่อหัว USB-C บนบอร์ด Pi ก่อน แล้วจึงเสียบอแดปเตอร์เข้ากับเต้ารับไฟ 220 โวลต์ โปรดสังเกตสัญลักษณ์ของ USB บนหัวสายจะต้องหมายขึ้น ดังรูปที่ B.7



รูปที่ B.7: การเชื่อมต่อไฟเลี้ยงจากอแดปเตอร์ทางหัวไมโคร USB กับบอร์ด Pi

5. โปรดสังเกตหลอดไฟ LED สีแดงจะสว่างขึ้นเมื่อไม่มีอะไรมิดพลาด หลังจากนั้น ภาพสีรุ้งจะปรากฏขึ้นบนจอด้านซ้ายบน ดังรูปที่ B.8



รูปที่ B.8: ภาพสีรุ้งบนจออันเกิดจากบอร์ด Pi ทำงานเป็นปกติ

#### B.4 กิจกรรมท้ายการทดลอง

กิจกรรมต่อไปนี้อาศัยความชำนาญของผู้อ่านส่วนตัว เพื่อป้องกันความเสียหายที่อาจเกิดขึ้น ผู้เขียนขอแนะนำสำหรับผู้อ่านที่มีความชำนาญหรืออวนกลับมาทำในภายหลังที่ทำการทดลองจนครบแล้ว

1. ลอกอแดปเตอร์ออกจากเต้าเสียบ ประกอบสายแพรกับหัวต่อเชื่อมกับโปรโตบอร์ด และเชื่อมต่อสายมิเตอร์สีดำกับกราวน์ดของบอร์ด
2. เสียบอแดปเตอร์ในเต้าเสียบ วัดความต่างศักย์ของไฟ 5 โวลต์ และ 3.3 โวลต์ จากหัวต่อ 40 ขา โดยใช้สายค่อนเน็คเตอร์ไปเสียบบนโปรโตบอร์ดแล้วจึงทำการวัด

## ภาคผนวก C

# การทดลองที่ 3 การติดตั้งระบบปฏิบัติการ Raspberry Pi OS

การทดลองนี้เสริมสร้างประสบการณ์ให้ผู้อ่านได้มีโอกาสติดตั้งระบบปฏิบัติการ Raspberry Pi OS และโปรแกรมเสริมอื่นๆ โดยอาศัยเครื่องคอมพิวเตอร์ที่ทำงานระบบปฏิบัติการ เช่น ลินุกซ์ Ubuntu, ไมโครซอฟต์ วินโดวส์ และ Mac OS ติดตั้งลงบนการ์ดหน่วยความจำไมโคร SD เป็นอุปกรณ์สำรองข้อมูล การทดลองจะช่วยเสริมสร้างความเข้าใจเนื้อหาของบทที่ 3 ในส่วนของซอฟต์แวร์ โดยมีวัตถุประสงค์ ดังต่อไปนี้

- เพื่อให้เข้าใจกลไกการติดตั้งระบบปฏิบัติการ Raspberry Pi OS ผ่านทางเครือข่ายอินเทอร์เน็ต
- เพื่อประกอบการใช้งานและพัฒนาโปรแกรมบนระบบปฏิบัติการ Raspberry Pi OS ซึ่งเป็นลินุกซ์เวอร์ชัน สำหรับบอร์ดตระกูล Raspberry Pi

ก่อนผู้อ่านจะติดตั้งระบบปฏิบัติการ Raspberry Pi OS บนบอร์ด Pi ผู้อ่านจะต้องเตรียมการ์ดหน่วยความจำไมโคร SD ขนาดความจุไม่น้อยกว่า 16 กิกะไบต์ ให้เรียบร้อย แล้วจึงติดตั้งโปรแกรมตามขั้นตอนต่อไปนี้

### C.1 การเตรียมการ์ดหน่วยความจำไมโคร SD

- ทำการดาวน์โหลดไฟล์โปรแกรม Raspberry Pi Imager สำหรับติดตั้งระบบปฏิบัติการ Raspberry Pi OS ในการ์ดหน่วยความจำไมโคร SD ตามลิงก์ต่อไปนี้  
<https://www.raspberrypi.org/software/>
- เลือกดาวน์โหลดตามระบบปฏิบัติการที่ผู้อ่านใช้งานอยู่ เช่น วินโดวส์ MacOS หรือ ลินุกซ์ Ubuntu

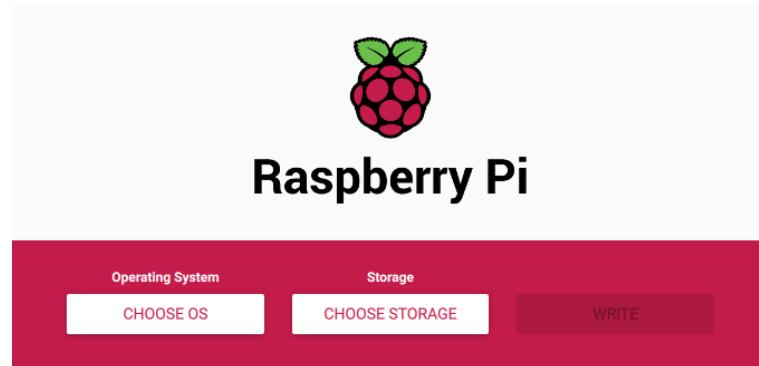


รูปที่ C.1: หน้าต่างดาวน์โหลดไฟล์โปรแกรม Raspberry Pi Imager สำหรับติดตั้งระบบปฏิบัติการ Raspberry Pi OS ในการ์ดหน่วยความจำไมโคร SD

3. ทำการติดตั้งโดยทำตาม installShield Wizard จนแล้วเสร็จ

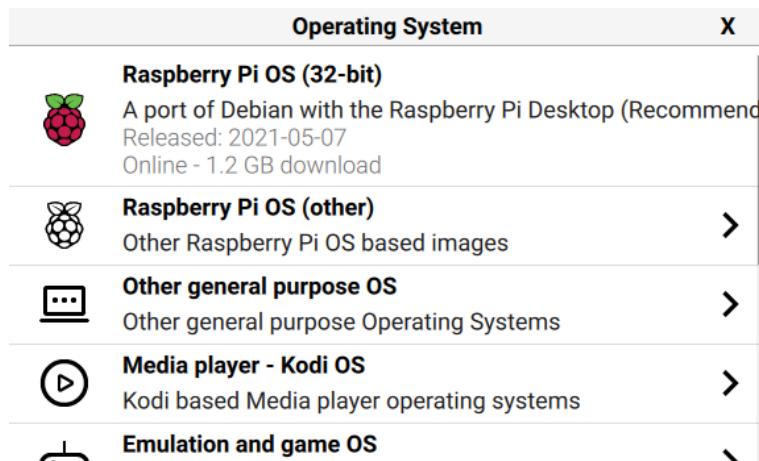
## C.2 การติดตั้ง RaspberryPi OS บนการ์ดหน่วยความจำไมโคร SD

1. รันโปรแกรม RaspberryPi Imager ซึ่งมีหน้าต่างหลักในรูปที่ C.2



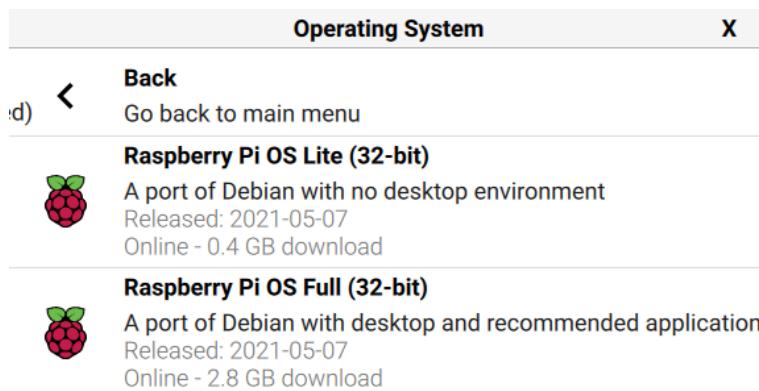
รูปที่ C.2: หน้าต่างของโปรแกรม Raspberry Pi Imager

2. กดปุ่ม CHOOSE OS เพื่อเลือกระบบปฏิบัติการ ในรูปที่ C.3



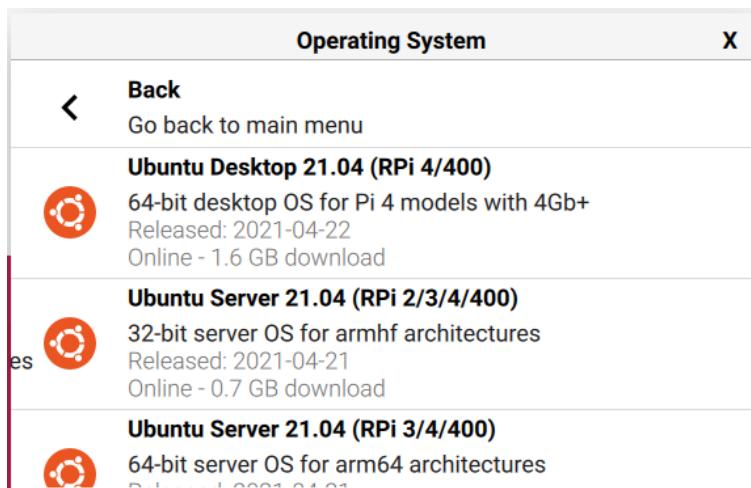
รูปที่ C.3: เมนูตัวเลือกใต้เมนู CHOOSE OS

3. กดเลือก **Raspberry Pi OS (32-bit)** ลำดับบนสุด ซึ่งเป็นตัวเลือกที่เว็บไซต์แนะนำ (Recommended)
4. ผู้อ่านขึ้นสูงสามารถเลือก Raspberry Pi OS อื่นๆ ได้ ดังนี้
  - กด **Raspberry Pi OS (other)** เพื่อเลือกระบบปฏิบัติการอื่นๆ ที่ตรงตามความต้องการในรูปที่ C.4



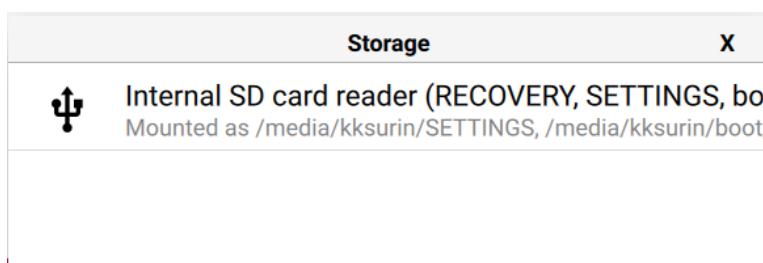
รูปที่ C.4: เมนูตัวเลือกอื่นๆ ใต้เมนู Raspberry Pi OS (other)

- กดเลือก **Other general purpose OS** เพื่อเลือกระบบปฏิบัติการอื่นๆ ที่ไม่ใช่ Raspberry Pi OS ในรูปที่ C.5



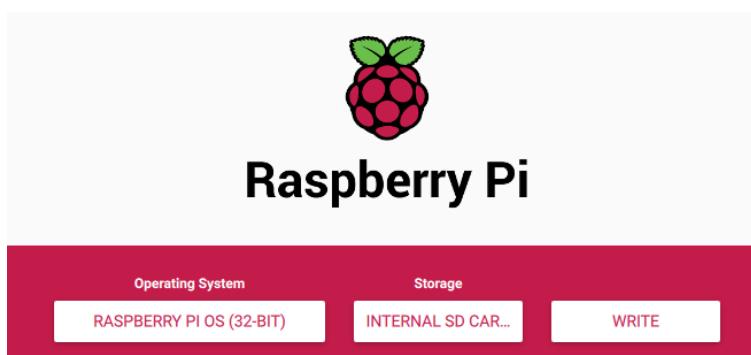
รูปที่ C.5: เมนูตัวเลือกอื่นๆ ให้เมนู Other general purpose OS

- กดปุ่ม CHOOSE STORAGE เพื่อเลือกรายการด้านขวาของความจำ SD ที่ต้องการ กรณีที่เสียบอยู่หลายการ์ด ในรูปที่ C.6



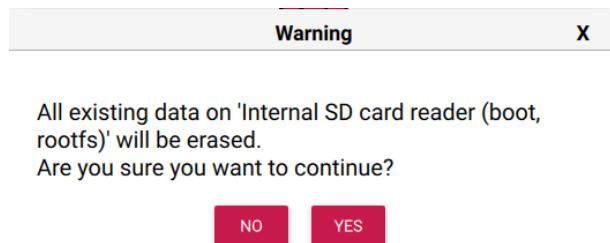
รูปที่ C.6: หน้าต่าง Imager

- เมื่อตั้งค่าตัวเลือกภายในตัวเลือก CHOOSE OS และ CHOOSE STORAGE แล้ว ผู้ใช้จะสามารถกดปุ่ม WRITE เพื่อเริ่มต้นการดาวน์โหลดและเขียนได้ ในรูปที่ C.7



รูปที่ C.7: ปุ่ม WRITE ในหน้าต่าง Raspberry Pi Imager

- หากการ์ดหน่วยความจำมีข้อมูลเดิม หน้าต่างเตือนจะปรากฏขึ้นตามรูปที่ C.8 หากต้องการเขียนทับให้กดปุ่ม Yes และหากไม่มั่นใจให้กดปุ่ม No



รูปที่ C.8: หน้าต่างเตือนผู้ใช้ที่ต้องการเขียนหับการ์ดหน่วยความจำ SD

8. ระหว่างที่ดำเนินการดาวน์โหลดและเขียนการ์ดไปพร้อมๆ กัน ผู้อ่านต้องดูแลการเชื่อมต่อเมืองให้ติดขัด ในรูปที่ C.9



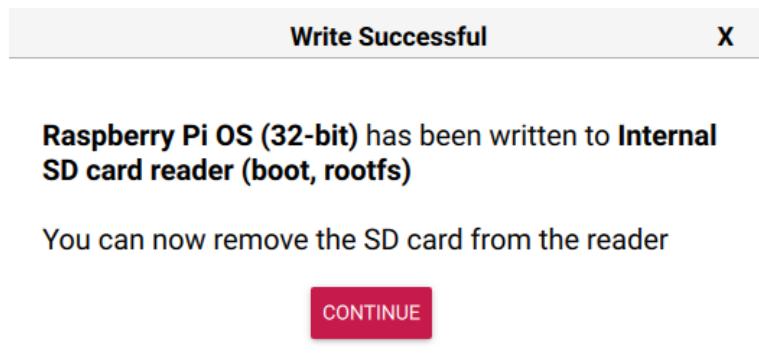
รูปที่ C.9: หน้าต่าง Raspberry Pi Imager ทายอยเขียนข้อมูลภายในการ์ด

9. เมื่อโปรแกรมเขียนการ์ดจนครบ 100% แล้วจึงเริ่มทวนสอบ (Verify) ข้อมูลภายในการ์ด ในรูปที่ C.10



รูปที่ C.10: หน้าต่าง Raspberry Pi Imager ทวนสอบ (Verify) ข้อมูลภายในการ์ด

10. เมื่อเขียนหรือติดตั้งลงในการ์ดหน่วยความจำสำเร็จ (Successful) การ์ดจะมีพาร์ทิชัน ชื่อ boot และ rootfs ในรูปที่ C.11

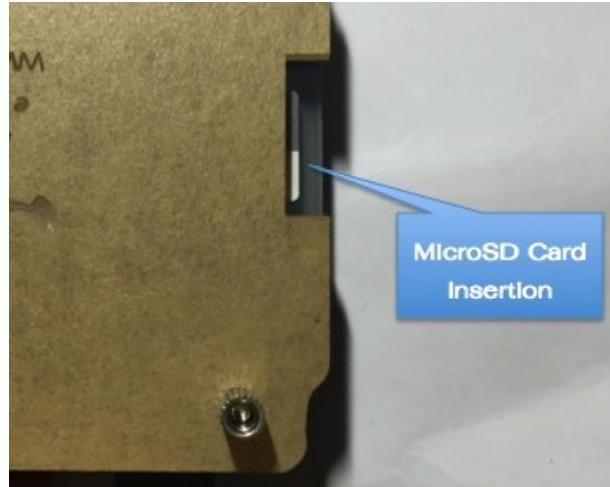


รูปที่ C.11: ปุ่ม CONTINUE ในหน้าต่าง Raspberry Pi Imager เมื่อติดตั้งสำเร็จ

- กดปุ่ม CONTINUE และปิดโปรแกรม และสั่งให้ระบบปฏิบัติการหลักปลดการ์ดหน่วยความจำไมโคร SD ออกจากเครื่อง

### C.3 การบูตระบบปฏิบัติการ RaspberryPi OS

- หมายบอร์ด Pi และจึงสอดการ์ดหน่วยความจำไมโคร SD ที่ได้เข้าไปในสล็อตบนบอร์ด Pi โปรดสังเกตว่า การ์ดคว่าหน้างลงดังรูป เพื่อให้หน้าสามผัสดตรงกับบอร์ด



รูปที่ C.12: สอดการ์ดเข้าไปในสล็อตบนบอร์ด Pi โดยหมายบอร์ดขึ้นมา โปรดสังเกตการ์ดหน่วยความจำจะต้อง มีลักษณะดังรูป

- ตรวจสอบว่าการ์ดหน่วยความจำไมโคร SD เสียบถูกต้องแล้วจึงเสียบอడเปเตอร์ไฟเลี้ยงให้กับบอร์ด
- ตรวจสอบว่าบอร์ดทำงานเมื่อจ่ายไฟให้ตามรูปที่ C.13 บอร์ดจะเริ่มต้นทำงาน



รูปที่ C.13: หน้าต่าง Welcome ของระบบปฏิบัติการ Raspberry Pi OS

## C.4 การตั้งค่าบอร์ด Pi เพื่อใช้งาน

### C.4.1 การตั้งค่าต่างๆ

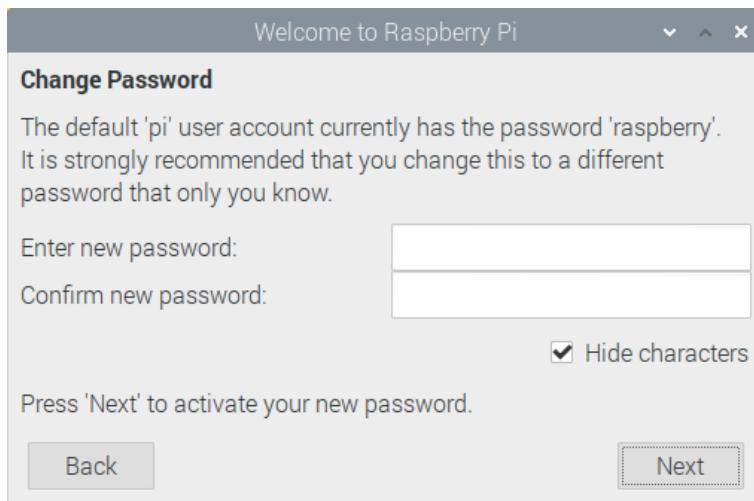
เมื่อบอร์ดสามารถรับระบบปฏิบัติการได้สำเร็จตามรายละเอียดในหัวข้อที่ 3.3.1 ผู้ใช้จะต้องตั้งค่าต่างๆ (Configure) บอร์ดให้พร้อมสำหรับใช้งานต่อไป ดังนี้

- ตั้งค่าประเทศ โซนเวลา และภาษาในการใช้งานเมนูเป็นภาษาอังกฤษ ตามรูปที่ C.14 เพื่อใช้เมนูเป็นภาษาอังกฤษซึ่งจะช่วยให้เรียนรู้คอมพิวเตอร์ดีกว่า



รูปที่ C.14: หน้าต่างตั้งค่าประเทศ โซนเวลา และภาษาในการใช้งานเมนูเป็นภาษาอังกฤษ

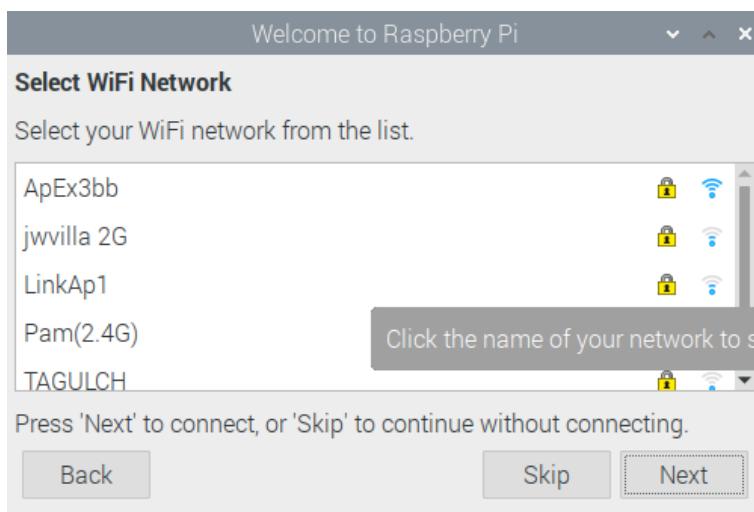
- สำหรับผู้อ่านขึ้นเริมต้น ผู้อ่านไม่ควรปรับแก้ใดๆ ระบบจะตั้งชื่อ username อัตโนมัติคือ pi โดยมีรหัสผ่าน (Password) คือ raspberry
- สำหรับผู้อ่านขั้นสูงทำการตั้งชื่อผู้ใช้ และพาสเวิร์ด ซึ่งผู้อ่านควรใช้ชื่อ pi และพาสเวิร์ดใหม่ที่ปลอดภัยในรูปที่ C.15 เพื่อความปลอดภัยในอนาคต



รูปที่ C.15: หน้าต่างสำหรับการเปลี่ยนรหัสผ่านใหม่ของ username ชื่อ pi โดยจะต้องกรอกรหัสผ่านใหม่ซ้ำจำนวน 2 ครั้ง

#### C.4.2 การตั้งค่า WiFi เพื่อเชื่อมต่อกับอินเทอร์เน็ต

- ระบบมองเห็นสัญญาณ WiFi และแสดงรายชื่อของสัญญาณ WiFi (SSID) ที่อยู่รอบๆ บริเวณบอร์ด ตามตัวอย่างในรูปที่ C.16 สัญญาณแม่กุญแจ หมายถึง การเข้ารหัสป้องกันซึ่งผู้ใช้ต้องกรอกพาสเวิร์ดก่อนเชื่อมต่อ



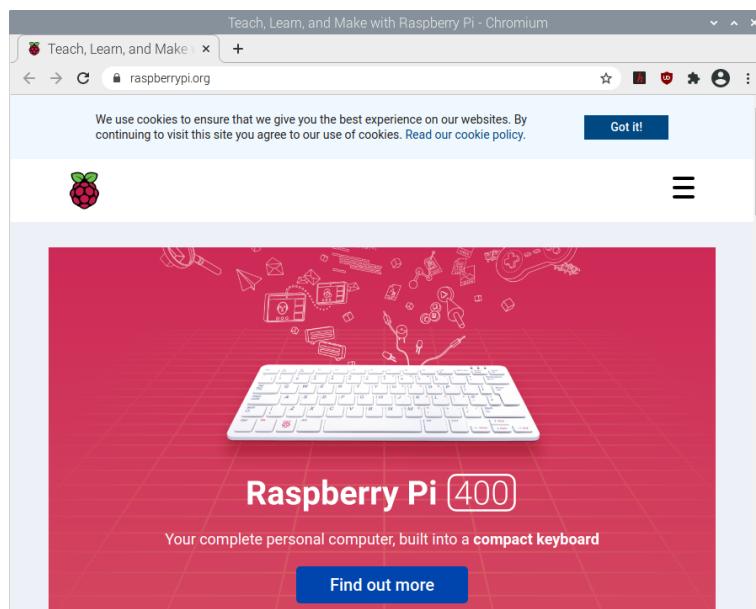
รูปที่ C.16: ตัวอย่างรายชื่อสัญญาณ WiFi รอบๆ ที่บอร์ด Pi มองเห็น ซึ่งจะแตกต่างกับของผู้อ่าน

- คลิกเลือกรายชื่อสัญญาณที่ต้องการ ตามตัวอย่างในรูปที่ C.17



รูปที่ C.17: หน้าต่างสำหรับกรอกพาสเวิร์ดของสัญญาณ WiFi ที่ต้องการเชื่อมต่อ

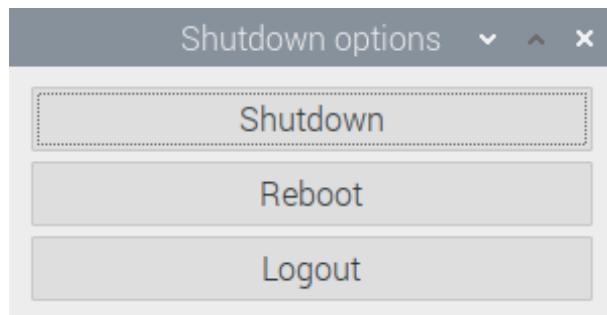
3. เมื่อเชื่อมต่อสัญญาณ WiFi ลูกต้องแล้ว เปิดโปรแกรมเบราว์เซอร์ชื่อ Chrome เพื่อทดสอบการเชื่อมต่อเครือข่ายอินเทอร์เน็ตไร้สาย



รูปที่ C.18: เว็บเพจเริ่มต้นของเว็บไซต์ [www.raspberrypi.org](http://www.raspberrypi.org)

### C.4.3 การรีสตาร์ตและขัดawan

- เมื่อติดตั้งค่าระบบแล้วเสร็จ ผู้อ่านควรทำการรีบูต หรือ รีสตาร์ตเครื่อง เลื่อนมาสู่ไปคลิกปุ่มสัญลักษณ์รูปผล Raspberry ซึ่งทำหน้าที่เป็นปุ่มเมนูหลัก เลือกเมนูย่อย Logout ณ ตำแหน่งล่างสุด เมนูนี้มักใช้เรียกเมื่อผู้ใช้ต้องการหลังการอัปเดตซอฟต์แวร์ต่างๆ ที่จำเป็น หรือ ผู้ใช้ต้องการปรับแก้อาการต่างๆ ตามรูป



รูปที่ C.19: หน้าต่างสำหรับเมนู Shutdown เพื่อให้ผู้ใช้ ซัตดาวน์ รีบูต (Reboot) หรือล็อกเอาท์ (Logout)

- กดปุ่ม Shutdown ในรูปที่ C.19 เพื่อปิดเครื่องตามที่อธิบายในหัวข้อที่ 3.3.7 โปรดสังเกตหลอดไฟ LED สีเขียวที่ติดกับหลอดไฟ LED สีแดง ไฟ LED สีเขียวจะกระพริบจนดับจึงค่อยถอนแเดปเตอร์ออกจากเต้าเสียบไฟ 220 โวลต์

## C.5 กิจกรรมท้ายการทดลอง

- การติดตั้งระบบจากไฟล์ config.txt เพื่อแจ้งให้ ARM Loader ทำการบูตระบบตามรายละเอียดในไฟล์นั้น ผู้อ่านสามารถอ่านค่าโดยใช้คำสั่ง

```
$ cat /boot/config.txt
```

ขอให้ผู้อ่านสังเกตและบันทึกประযุคที่ไม่เขียนต้นด้วยสัญลักษณ์ # เพื่อค้นคว้าเพิ่มเติมใน google.com

- คำสั่ง sudo ย่อมาจากคำว่าอะไร และทำไมต้องใช้คำสั่งนี้นำหน้าคำสั่งอื่นๆ ในโปรแกรม Terminal
- ค้นคว้าเพิ่มเติมว่าไฟ LED สีเขียวบ่งบอกสัญญาณอะไร เหตุใดจึงต้องรอให้ดับก่อนถอดแเดปเตอร์ออก
- สำรวจส่วนต่างๆ ของหน้า Desktop และวัดตามคร่าวๆ พร้อมรายละเอียดสำคัญ
- สำรวจเมนูหลัก และเมนูรองว่ามีรายละเอียดอะไรบ้าง และวัดเป็นแนวภูมิต้นไม้
- ค้นหาริการเพิ่มคีย์บอร์ดภาษาไทยเพื่อใช้งานบนระบบปฏิบัติการ Raspberry Pi OS

## ภาคผนวก D

# การทดลองที่ 4 การใช้งานระบบปฏิบัติการยูนิกซ์เบื้องต้น

ยูนิกซ์ (Unix) เป็นระบบปฏิบัติลำดับแรกๆ ของโลกที่เป็นต้นแบบการสร้างระบบปฏิบัติการต่างๆ รวมทั้งระบบปฏิบัติการลินุกซ์ และ Raspberry Pi OS ผู้อ่านสามารถเรียนรู้การใช้งานคำสั่งพื้นฐานด้วยการพิมพ์คำสั่งทางคีย์บอร์ด และกราฟิกไปพร้อมกัน โดยมีวัตถุประสงค์ดังต่อไปนี้

- เพื่อเปรียบเทียบการทำงานแบบกราฟิกและแบบคำสั่งทางคีย์บอร์ด
- เพื่อให้ผู้อ่านใช้คำสั่งเพื่อบริหารจัดการไฟล์ในไดเรกทอรีหรือโฟลเดอร์เบื้องต้น
- เพื่อวางแผนการใช้งานระบบปฏิบัติการยูนิกซ์เบื้องต้นสำหรับพัฒนาโปรแกรมภาษาต่างๆ
- เพื่อค้นคว้าข้อมูลขั้นสูงของบอร์ด Pi

ผู้อ่านที่คุ้นเคยกับระบบปฏิบัติการวินโดว์ส และการพิมพ์คำสั่งทางคีย์บอร์ด (Command Line) ของระบบปฏิบัติการดอส (DOS: Disk Operating System) ในอดีต จะคุ้นพบว่า คำสั่งเหล่านี้มีความใกล้เคียงกัน แต่ยูนิกซ์จะเข้มงวดกว่า DOS ขอให้ผู้อ่านปฏิบัติตามคำสั่งอย่างระมัดระวัง และสังเกตตัวพิมพ์อย่างละเอียดว่าเป็นตัวพิมพ์ใหญ่หรือเล็ก เพื่อสร้างความคุ้นเคยกับการพัฒนาโปรแกรมด้วยภาษาอื่นๆ ต่อไป

## D.1 การใช้งานระบบผ่านทาง GUI

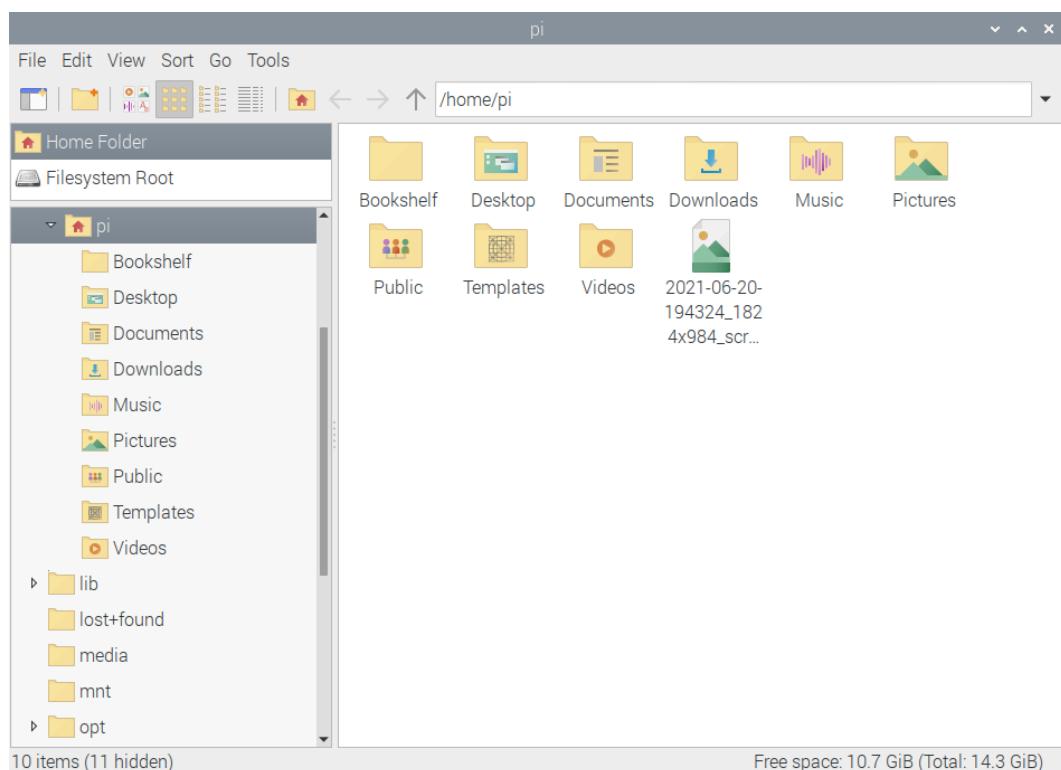
### D.1.1 หน้าจอหลัก (Desktop)

หน้าจอหลักของระบบในรูปที่ D.3 มีลักษณะคล้ายกับหน้าจอหลักของระบบปฏิบัติการอื่นๆ เช่น ปุ่มเมนูหลัก แถบแสดงรายชื่อโปรแกรมที่กำลังทำงานอยู่ ปุ่มไอคอนของโปรแกรมที่นิยมใช้บ่อย (Favorites) ไอคอนแสดงการเชื่อมต่อสัญญาณ WiFi คลือก เป็นต้น สิ่งที่แตกต่าง คือ ตำแหน่งที่จัดวางของปุ่มหรือไอคอนเหล่านี้อาจแตกต่างกันได้ตามการปรับแต่งโดยผู้ใช้งาน ตารางต่อไปนี้เป็นการเปรียบเทียบระหว่างไอคอนและปุ่มต่างๆ ของ Raspberry Pi OS และ Windows ซึ่งผู้อ่านจะต้องวัดเติมลงไปด้วยตนเอง ตามรายชื่อปุ่มด้านซ้าย

ปุ่ม	Raspberry Pi OS	Windows
เมนูหลัก(Main Menu)		
ปิด (Close)		
ย่อ (Minimize)		
ขยาย (Maxmimize)		

## D.1.2 ไฟล์เมเนจเจอร์ (File Manager)

ไฟล์เมเนจเจอร์ คือ โปรแกรมสำหรับเบราว์เซอร์ (Browse) โครงสร้าง รายชื่อไดเรกทอรี รายชื่อไฟล์ต่างๆ ภายในอุปกรณ์เก็บรักษาข้อมูล เช่น การ์ดหน่วยความจำไมโคร SD เป็นต้น รูปที่ D.1 แสดงหน้าต่างของไฟล์เมเนจเจอร์ (File Manager) ขณะที่เปิดไดเรกทอรีชื่อ /usr ทางด้านขวา และโครงสร้างของอุปกรณ์เก็บรักษาข้อมูลทางด้านซ้าย



รูปที่ D.1: หน้าต่างของไฟล์เมเนจเจอร์ (File Manager) ขณะที่เปิดไดเรกทอรีชื่อ /usr

## D.2 การใช้งานระบบผ่านทางโปรแกรม Terminal



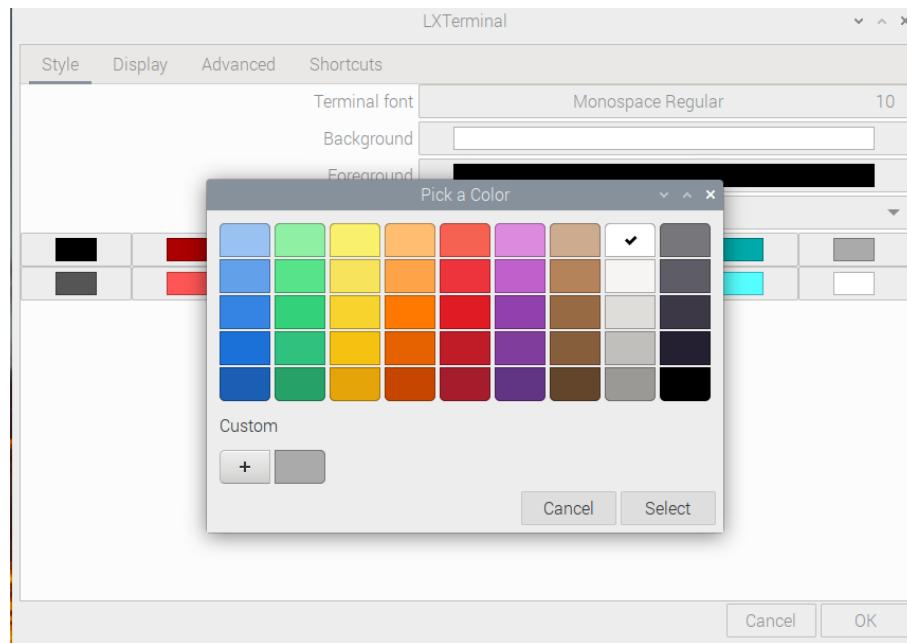
รูปที่ D.2: รูปไอคอนของโปรแกรม Terminal

ในอดีตผู้ใช้งานระบบยูนิกซ์จะต้องคีย์คำสั่งต่างๆ ผ่านทางโปรแกรม Terminal เท่านั้น เรียกว่า การใช้แบบคอมมานด์ไลน์ (Command Line) ซึ่งผู้ใช้จะต้องฝึกฝนและจำคำสั่งต่างๆ ทำให้การใช้งานแบบคอมมานด์ไลน์ยุ่งยากและไม่น่าสนใจเมื่อเทียบกับการใช้งานแบบ GUI เมื่อมีอนในปัจจุบัน แต่ผู้ใช้งานที่เชี่ยวชาญสามารถเข้าใจการทำงานได้ลึกซึ้งกว่า คำสั่งพื้นฐานและคำสั่งชัตดาวน์ในการทดลองนี้จะช่วยเสริมความเข้าใจของผู้อ่านได้เป็นอย่างดี โดยผู้ใช้สามารถเปิดโปรแกรม Terminal ด้วยการคลิกบนปุ่มที่มีรูปเหมือนไอคอนในรูปที่ D.2 บนแถบแสดงรายการซอฟต์แวร์ (Taskbar) รูปที่ D.3 แสดงหน้าต่างของโปรแกรม Terminal ซึ่งผู้เขียนได้ปรับแต่งสีพื้นและสีของตัวอักษรให้เหมาะสม



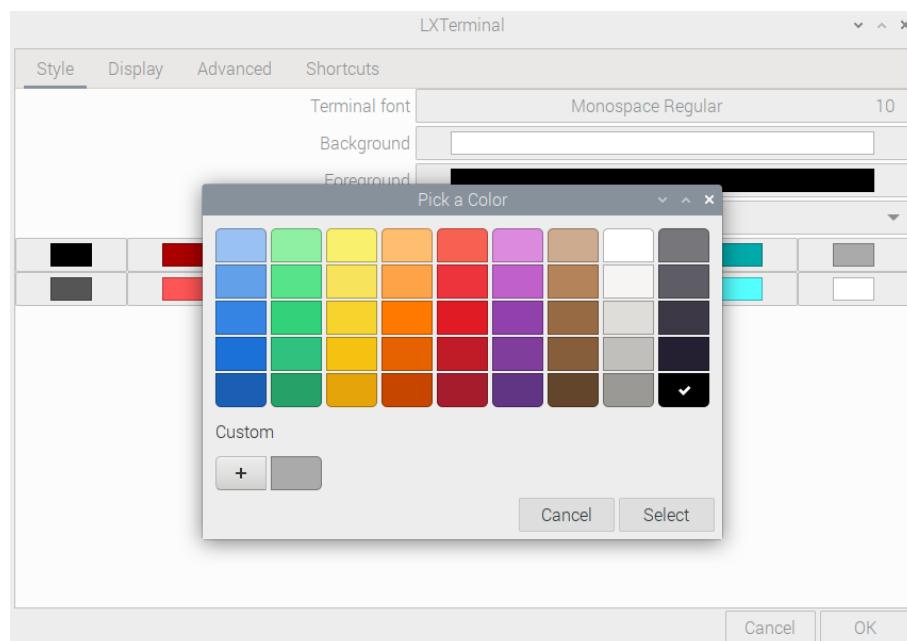
รูปที่ D.3: หน้าต่างของโปรแกรม Terminal ซึ่งสามารถปรับแต่งสีพื้นและสีของตัวอักษรได้

1. เปิดโปรแกรม Terminal บนเมนูหลัก
2. คลิกเมนู Edit -> Preferences
3. คลิกที่แท็บสีของ Background จากให้เลือกสีขาว ดังรูป คลิกปุ่ม Select



รูปที่ D.4: หน้าต่างปรับแต่งสีพื้น (Background)

4. คลิกที่แถบสีของ Foreground จากให้เลือกสีดำ ดังรูป คลิกปุ่ม Select แล้วจึงคลิกปุ่ม OK ดังรูป



รูปที่ D.5: หน้าต่างปรับแต่งสีตัวอักษร (Foreground)

5. ทดสอบด้วยการปิดโปรแกรมแล้วเปิดอีกรอบว่าสีที่เลือกยังคงอยู่

### D.2.1 คำสั่งพื้นฐานของระบบยูนิกซ์

ผู้อ่านสามารถฝึกใช้คำสั่งเหล่านี้บนโปรแกรมเทอร์มินัล (Terminal) ตามตารางต่อไปนี้ โปรดสังเกต สัญลักษณ์ \$ หมายถึง คำสั่งชนิดคอมマンด์ไลน์ในโปรแกรม Terminal

ลำดับที่	รายละเอียด	คำสั่ง
1	แสดงรายชื่อไฟล์และไดเรกทอรี Ex.: \$ ls แสดงรายชื่อไฟล์และไดเรกทอรีในไดเรกทอรีปัจจุบัน Ex.: \$ ls -l แสดงรายละเอียดต่างๆ ของไฟล์และไดเรกทอรีในไดเรกทอรีปัจจุบัน Ex.: \$ ls -la แสดงรายละเอียดต่างๆ ของไฟล์และไดเรกทอรีทั้งหมดในไดเรกทอรีปัจจุบัน โปรดสังเกตสัญลักษณ์ต่อไปนี้บริเวณสองแฉวบนสุดของผลลัพธ์ ”.” หมายถึง ไดเรกทอรีปัจจุบัน (current directory) ”..” หมายถึง ไดเรกทอรีที่อยู่เหนือขึ้นไป (parent directory)	ls <parameter>
2	สร้างไฟล์เปล่า	touch <file_name>
	Ex.: \$ touch test.txt สร้างไฟล์เปล่าชื่อ “text.txt”	
3	ทำไฟล์สำเนา	cp <source_file_name> <destination_file_name>
	Ex.: \$ cp test.txt test2.txt	
4	เปลี่ยนชื่อไฟล์	mv <source_file_name> <destination_file_name>
	Ex.: \$ mv test.txt test3.txt	
5	แสดงชื่อดireกทอรีปัจจุบัน	pwd
	Ex.: \$ pwd	
6	สร้างไดเรกทอรีใหม่	mkdir <directory_name>
	Ex.: \$ mkdir /home/pi/asm สร้างไดเรกทอรีใหม่ ชื่อ “asm” ภายในไดเรกทอรี ”/home/pi/” เพื่อใช้จัดเก็บไฟล์สำหรับการทดลองต่อไป	
7	Change directory	cd <destination>
	Ex.: \$ cd /home/pi/asm โปรดสังเกตสัญลักษณ์ต่อไปนี้ในประโยชน์ /home/pi/asm ”/” ตำแหน่งช้ายสุด หมายถึง ไดเรกทอรีราก (root directory) ”..” ตำแหน่งถัดมา หมายถึง สัญลักษณ์คันระหว่างชื่อไดเรกทอรี	

### D.2.2 การชัตดาวน์ (Shutdown)

การรีบูต หรือ รีสตาร์ตเครื่อง มักใช้เรียกเมื่อระบบต้องการหลังการอัปเดตซอฟต์แวร์ต่างๆ ที่จำเป็น หรือ ผู้ใช้ต้องการแก้อาการต่างๆ โดย

- พิมพ์คำสั่ง sudo reboot ในหน้าต่าง Terminal เพื่อรีบูตบอร์ด Pi และระบบปฏิบัติการในกรณีที่ผู้ใช้ต้องการเริ่มต้นระบบใหม่

\$ sudo reboot

ผู้อ่านสามารถรีบูตหรือรีสตาร์ตบอร์ดใหม่ด้วยคำสั่ง

```
$ shutdown -r now
```

โดย -r หมายถึง restart และ now หมายถึง ณ บัดนี้

- พิมพ์คำสั่ง **sudo shutdown -h now** ในหน้าต่าง Terminal เพื่อเตรียมพร้อมก่อนปิดเครื่อง ตามที่กล่าวในหัวข้อที่ [3.3.7](#)

```
$ shutdown -h now
```

โดย -h หมายถึง halt แปลว่า หยุด ซึ่งนักคอมพิวเตอร์ส่วนใหญ่นิยมใช้คำสั่งนี้ในสั่งให้เครื่องคอมพิวเตอร์สิ้นสุดการทำงาน

โปรดรอไฟ LED สีเขียวที่ติดกับไฟ LED สีแดง กระพริบจนดับเสียก่อนจึงค่อยกดอแดปเตอร์ออกจากเต้าเสียบไฟ 220 โวลต์

## D.3 ข้อมูลพื้นฐานของบอร์ด Pi

การใช้งานทางคอมมานด์ไลน์มีประโยชน์หลายด้าน เนื่องจากผู้ใช้สามารถเรียกใช้คำสั่งเกือบทั้งหมดในระบบรวมถึงการเขียนโปรแกรมชั้ลล์สคริปต์ (Shell Script) เพื่อสั่งงานคอมมานด์ไลน์ได้อัตโนมัติ ผู้อ่านควรจะฝึกใช้ให้คล่องเพื่อเตรียมความพร้อมไปเป็นนักพัฒนาโปรแกรม และพัฒนาระบบท่อไป โดยการทดลองนี้จะใช้คำสั่งพิเศษอ่านค่าข้อมูลของชีพิญและข้อมูลขั้นสูงอื่นๆ

### D.3.1 ข้อมูลพื้นฐานของชีพิญ

ผู้อ่านสามารถศึกษารายละเอียดเกี่ยวกับชีพิญที่ใช้งานอยู่บนบอร์ด โดยใช้คำสั่ง

```
$ cat /proc/cpuinfo
```

จดผลลัพธ์ที่ได้จากบอร์ด Pi ลงในช่องที่กำหนดให้ ซึ่งอาจแตกต่างกันสำหรับผู้ใช้ Raspberr Pi OS เวอร์ชัน 32 และ 64 บิต

- processor : \_\_\_\_\_
- model name : ARMv \_\_\_\_\_ rev \_\_\_\_\_ (\_\_\_\_\_)
- BogoMIPS : \_\_\_\_\_.
- Features : \_\_\_\_\_
- CPU implementer : \_\_\_\_\_
- CPU architecture : \_\_\_\_\_
- CPU variant : 0x\_\_\_\_\_
- CPU part : 0x\_\_\_\_\_

- CPU revision : \_\_\_\_\_
- Hardware : BCM \_\_\_\_\_
- Revision : \_\_\_\_\_
- Serial : \_\_\_\_\_
- Model : \_\_\_\_\_

### D.3.2 ข้อมูลขั้นสูงของซีพียูและบอร์ด

นอกเหนือจากข้อมูลพื้นฐานของซีพียูแล้ว ผู้อ่านสามารถสอบถามข้อมูลด้านฮาร์ดแวร์ขั้นสูงจากคำสั่งต่อไปนี้

ลำดับที่	คำสั่ง	รายละเอียด
1	\$ cat /proc/cpuinfo	รายละเอียดของซีพียูในการทดลองก่อนหน้า
2	\$ cat /proc/meminfo	รายละเอียดของหน่วยความจำภายในภาพ
3	\$ cat /proc/partitions	รายละเอียดของการตัดหน่วยความจำไมโคร SD
4	\$ cat /proc/version	รายละเอียดของระบบปฏิบัติการ
5	\$ vcgencmd measure_temp	อ่านค่าอุณหภูมิ ณ จุดต่างๆ
6	\$ vcgencmd measure_volts core	อ่านค่าโวลเทจของแกนประมวลผล
7	\$ vcgencmd measure_volts sdram_c	อ่านค่าโวลเทจของ SD-RAM
8	\$ vcgencmd measure_volts sdram_i	อ่านค่าโวลเทจของ SD-RAM I/O

ยกตัวอย่างเช่น ข้อมูลด้านหน่วยความจำภายในภาพ ที่เราเรียกว่า RAM หรือ SDRAM จะถูกบันทึกในไฟล์ /proc/meminfo ผู้อ่านสามารถแสดงข้อมูลในไฟล์โดย

```
$ cat /proc/meminfo
```

จดผลลัพธ์ที่สำคัญของบอร์ด Pi ที่ใช้

```
MemTotal:  _____ kB (KiB)
MemFree:   _____ kB (KiB)
MemAvail:  _____ kB (KiB)
Buffers:   _____ kB (KiB)
Cached:    _____ kB (KiB)
SwapCached: _____ kB (KiB)
SwapTotal: _____ kB (KiB)
SwapFree:  _____ kB (KiB)
PageTables: _____ kB (KiB)
```

## D.4 กิจกรรมท้ายการทดลอง

1. จะใช้โปรแกรมไฟล์เมเนเจอร์เพื่อทำการสำรวจโครงสร้างของไดเรกทอรีต่างๆ ของบอร์ด Pi
2. จะเปรียบเทียบโครงสร้างของไดเรกทอรีต่างๆ กับรูปที่ [3.13](#) ว่าแตกต่างกันอย่างไร
3. จะใช้โปรแกรม Terminal และคำสั่งที่จำเป็น เพื่อทำการสำรวจโครงสร้างของไดเรกทอรีต่างๆ ในเครื่อง และเปรียบเทียบกับข้อที่แล้ว
4. จะใช้โปรแกรมไฟล์เมเนเจอร์เพื่อทำการสำเนาหรือก็อปปี้ไฟล์ ลบไฟล์ สร้างไดเรกทอรีใหม่
5. จะใช้โปรแกรมไฟล์เมเนเจอร์แสดงแบบ List พร้อมรายละเอียดของไฟล์ หรือไดเรกทอรี เช่น ขนาด (Size) ของไฟล์ ชนิด (Type) วันเวลาที่แก้ไข
6. จะใช้โปรแกรม Terminal และคำสั่ง ls -la เพื่อเปรียบเทียบกับผลที่ได้จากข้อที่แล้ว
7. จะบอกรความแตกต่างระหว่างคำสั่ง cat และคำสั่ง ls
8. จะบอกรความแตกต่างระหว่างคำสั่ง cp และคำสั่ง mv
9. คำสั่ง vcgencmd ย่อมาจากคำว่าอะไร
10. ชิป BCM2\_\_\_\_ บนบอร์ดมีจำนวนซีพียูกี่แกนประมวลผล
11. ชิป BCM2835 เกี่ยวข้องกับ ชิป BCM2\_\_\_\_ ในข้อก่อนหน้าอย่างไร
12. จะบอกรหัสเลขรุ่น (CPU Revision) ของซีพียุ ARM Cortex A\_\_\_\_ ที่ได้จากคำสั่ง cpuinfo
13. ในหัวข้อที่ [D.3.2](#) จะบวกขนาดของหน่วยความจำ MemAvail, Buffers, Cached เพื่อเปรียบเทียบกับ MemTotal ว่าแตกต่างกันหรือไม่ อย่างไร
14. ผู้อ่านสามารถตรวจสอบขนาดของ SDRAM ที่มีบนบอร์ดกับข้อมูลที่ได้จาก meminfo ในหัวข้อใด และ แปลงหน่วยคิบีไบต์ (KiB) เป็นกิกะไบต์ (GiB) ได้อย่างไร (โปรดศึกษาบทอภิธานศัพท์ [M.5](#))
15. จะบอกรเวอร์ชัน (Version) และรายละเอียดอื่นๆ ของระบบปฏิบัติการ Raspberry Pi OS ที่ติดตั้ง
16. จะบอกรความต่างศักย์ของแกนประมวลผล หน่วยความจำภายในภาพ และอินพุต/เอาต์พุตและเปรียบเทียบ กันว่าแตกต่างกันหรือไม่ อย่างไร
17. จะบอกรูณหภูมิของซีพียุและตำแหน่งอื่นๆ บนบอร์ดว่าทำงานที่กีองศากเซลเซียส และเปรียบเทียบกันว่า แตกต่างกันหรือไม่ อย่างไร

## ภาคผนวก E

# การทดลองที่ 5 การพัฒนาโปรแกรมภาษา C บน- นุกซ์

การทดลองนี้คาดว่าผู้อ่านผ่านหัวข้อที่ 3.2 และมีประสบการณ์การเขียนหรือพัฒนาโปรแกรมด้วยภาษา C มาแล้ว ผู้อ่านควรมีความคุ้นเคยกับ IDE (Integrated Development Environment) จากการพัฒนาโปรแกรม และการดีบักโปรแกรมด้วยภาษา C/C++ ดังนั้น การทดลองมีวัตถุประสงค์เหล่านี้

- เพื่อให้เข้าใจการพัฒนาซอฟต์แวร์ด้วย IDE ชื่อ CodeBlocks บนระบบปฏิบัติการ Raspberry Pi OS/Linux/Unix
- เพื่อให้สามารถสร้าง Makefile เพื่อพัฒนาศักยภาพการทำงานเป็นนักพัฒนาอาชีพ
- เพื่อให้เข้าใจความแตกต่างระหว่างการพัฒนาโปรแกรมภาษา C ด้วย IDE และ Makefile

### E.1 การพัฒนาโดยใช้ IDE

โปรแกรมหรือแอปพลิเคชัน IDE ย่อมาจาก Integrated Development Environment ทำหน้าที่ช่วยเหลือโปรแกรมเมอร์ ทดสอบ และอาจรวมถึงควบคุมซอฟต์แวร์สโตร์ให้เป็นปัจจุบัน ขั้นตอนการทดลองนี้เริ่มต้นโดย

- ตรวจสอบภายในเครื่องว่ามีโปรแกรมชื่อ CodeBlocks ติดตั้งแล้วหรือไม่ โดยพิมพ์คำสั่งเหล่านี้ลงบนโปรแกรม Terminal

```
$ codeblocks
```

- หากติดตั้งแล้ว ให้ผู้อ่านข้ามไปข้อที่ 4 ได้ หากไม่มีโปรแกรม ผู้อ่านจะต้องติดตั้ง CodeBlocks โดยพิมพ์คำสั่งเหล่านี้ลงบนโปรแกรม Terminal

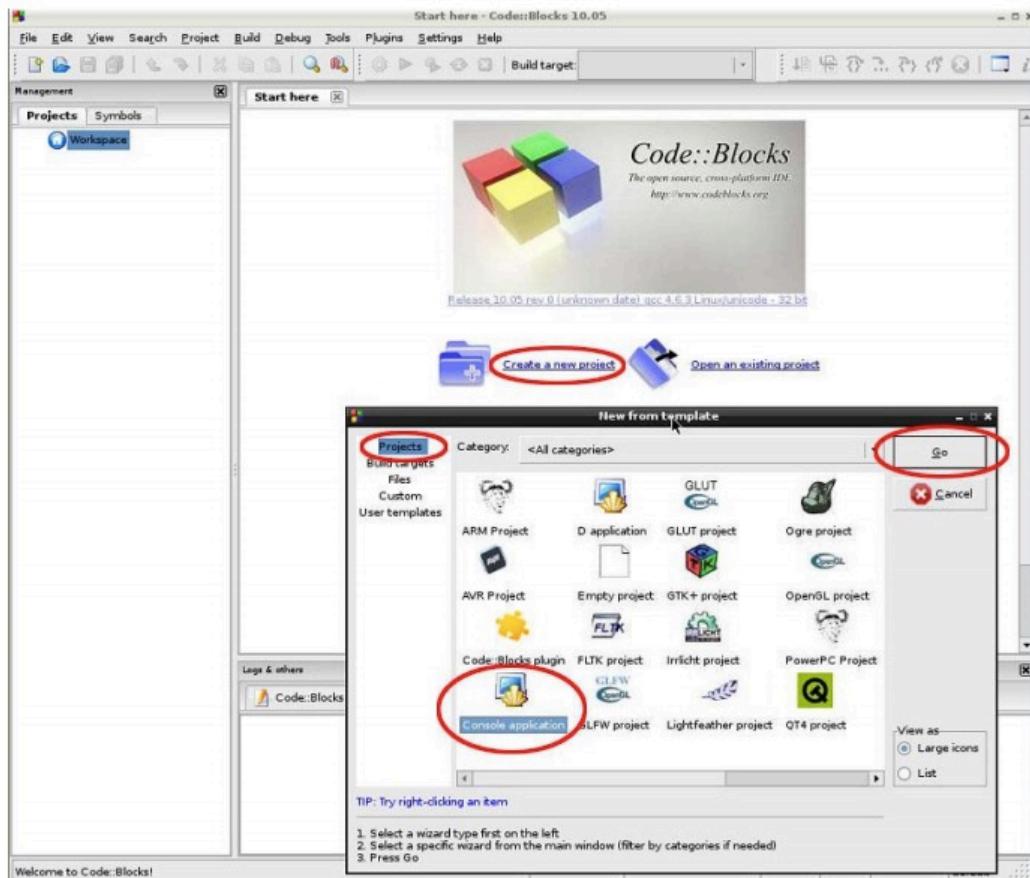
```
$ sudo apt-get install codeblocks
```

คำสั่ง sudo นำหน้าคำสั่งใดๆ นี้จะเป็นการเรียกใช้งานคำสั่งนั้นด้วยสิทธิ์ระดับ SuperUser การติดตั้งจะดาวน์โหลดโปรแกรมผ่านทางเครือข่ายอินเทอร์เน็ต และจำเป็นต้องใช้สิทธิ์ระดับสูงสุดนี้

- เมื่อติดตั้งเสร็จสิ้น พิมพ์คำสั่งนี้เพื่อเริ่มต้นใช้งาน CodeBlocks

\$ codeblocks

4. การใช้งาน CodeBlocks ครั้งแรกจะเป็นการติดตั้งค่า compiler plug-ins เป็น GCC หรือ GNU C Compiler.
5. หน้าต่างหลักจะปรากฏขึ้น หลังจากนั้น ผู้อ่านควรกด "Create a new project" เพื่อสร้างโปรเจ็คใหม่ ในหน้าต่าง "New from template"



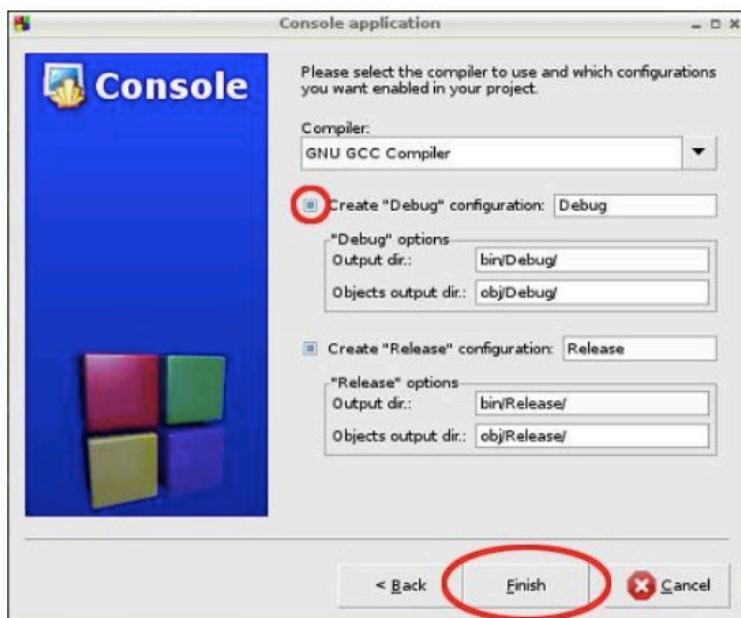
รูปที่ E.1: หน้าต่างเลือกชนิดโปรเจ็คที่จะพัฒนาเป็นชนิด "Console application"

6. เลือก "New Projects" ในช่องด้านซ้าย และเลือก "Console application" ในช่องด้านขวาเพื่อสร้างโปรแกรมในรูปแบบเท็กซ์mode (Text Mode) กดปุ่ม "Go" ตามรูปที่ E.1
7. กดปุ่ม Next> เพื่อดำเนินการต่อ
8. หน้าต่าง "Console application" จะปรากฏขึ้น กดเลือกภาษา "C" เพื่อพัฒนาโปรแกรมแล้วกดปุ่ม "Next>" ตามรูปที่ E.2



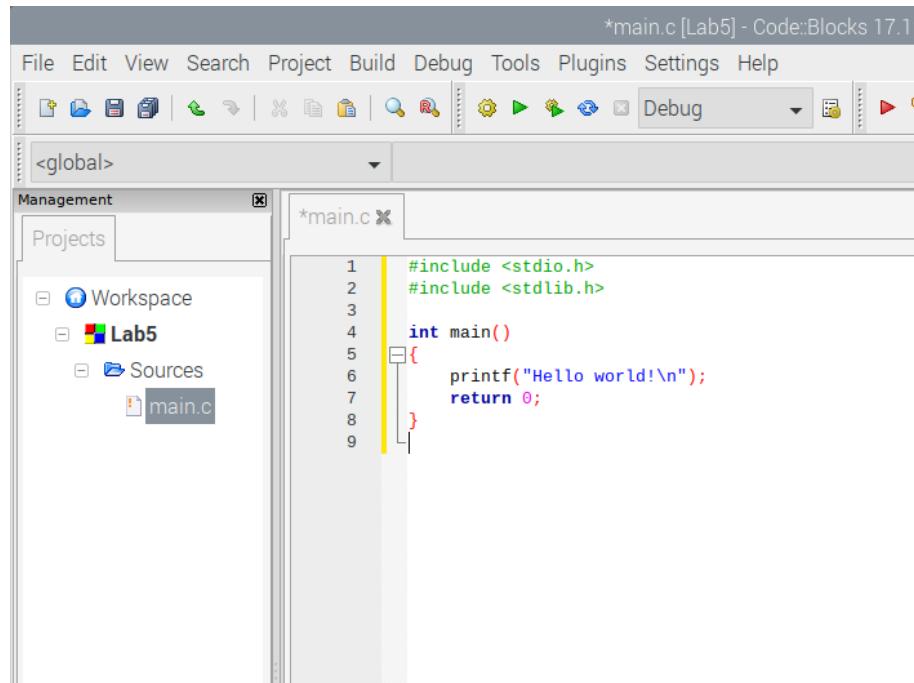
รูปที่ E.2: หน้าต่างเลือกภาษา C หรือ C++ สำหรับโปรเจคท์ที่จะพัฒนา

9. กรอกชื่อโปรเจคท์ใหม่ชื่อ Lab5 ในช่อง Project title: และกรอกชื่อไดเรกทอรี /home/pi/asm/ ในช่อง Folder to create project in: โปรดสังเกตข้อความในช่อง Project filename: ว่าตรงกับ Lab5.cbp ใช้หรือไม่
10. กดปุ่ม "Next >" เพื่อดำเนินการต่อและสุดท้ายจะเป็นขั้นตอนการเลือกคอนฟิกกูเรชัน (Configuration) สำหรับคอมไพล์เตอร์ในรูปที่ E.3 โดย Debug เหมาะสำหรับการเริ่มต้นและแก้ไขข้อผิดพลาด แล้วจึงกดปุ่ม "Finish" เมื่อเสร็จสิ้นการติดตั้ง



รูปที่ E.3: การเลือกคอนฟิกกูเรชัน (Configuration) Debug สำหรับคอมไเพลอร์ GNU GCC ในโปรเจคท์ Lab5

11. คลิกช้ายบันชื่อ Lab5 ในหน้าต่าง Management/Workspace ด้านซ้ายมือ เพื่อขยายไดเรกทอรี Sources แล้วจึงคลิกบนไฟล์ไอคอนชื่อ main.c



รูปที่ E.4: การเปิดอ่านไฟล์ main.c ภายใต้โปรเจคท์ Lab5 ที่สร้างขึ้น

คำสั่งเริ่มต้นที่ CodeBlocks สร้างไว้อัตโนมัติในไฟล์ main.c คือ Hello world

12. ป้อนโปรแกรมนี้แทนที่ของเดิมในไฟล์ main.c

```

#include <stdio.h>
int main(void)
{
    int a;
    printf("Please input an integer: ");
    scanf("%d", &a);
    printf("You entered the number: %d\n", a);
    return 0;
}

```

13. Build->Compile โปรแกรม จนไม่มีข้อผิดพลาด โดยสังเกตจากหน้าต่างด้านล่างสุด

14. รันโปรแกรมเพื่อทดสอบการทำงาน

## E.2 การดีบัก (Debugging) โดยใช้ IDE

การดีบักโปรแกรม คือ การตรวจสอบการทำงานของโปรแกรมอย่างละเอียด CodeBlocks รองรับการดีบักผ่านเมนู Debug ผู้อ่านสามารถเริ่มต้นโดย

- กด Debug บนเมนูแถบบนสุด เลือก Active Debuggers GDB/CDB Debugger เป็นค่าดีฟอลท์ (Target's Default)

2. เลื่อนเมาส์ cursor ไปยังบรรทัดที่ต้องการศึกษา กดปุ่ม F5 เพื่อตั้งเบรกพอยน์ (Break Point) ตรงบรรทัดปัจจุบันของเมาส์ cursor โปรดสังเกตต้นประโยคด้านซ้ายสุดจะมีวงกลมสีแดงประกายขึ้น และเมื่อกด F5 อีกครั้งวงกลมสีแดงจะหายไป เรียกว่า การทิอกเกิล (Toggle) เบรกพอยน์ กด F5 อีกครั้งเพื่อสร้างวงกลมสีแดงตรงบรรทัดที่สนใจเพียงจุดเดียวเท่านั้น จับภาพหน้าต่างที่ได้วางไว้ใต้คำสั่งนี้เพื่อให้ตรวจสอบ
  
  
  
  
  
  
3. กดปุ่ม F8 (Start/Continue) บนคีย์บอร์ดเพื่อรันโปรแกรมอีกรอบ โปรแกรมจะรันไปจนหยุดตรงประโยคที่มีวงกลมสีแดงนั้น โปรดสังเกตสัญลักษณ์สามเหลี่ยมสีเหลืองซ้อนทับกันอยู่ หลังจากนั้น กดปุ่ม F7 (Next line) เพื่อดำเนินการต่อทีละบรรทัด
4. เลื่อนเมาส์ cursor ไปยังประโยคที่มีวงกลมสีแดง กดปุ่ม F5 บนคีย์บอร์ดเพื่อปลดวงกลมสีแดงออก หรือยกเลิกเบรกพอยน์
5. เริ่มต้นการดิบักใหม่เพื่อศึกษาการทำงานของปุ่ม F4 (Run to cursor) โดยเลื่อนเมาส์ cursor ไปวางบนประโยคที่สนใจ กดปุ่ม F4 และสังเกตว่าสามเหลี่ยมสีเหลืองจะปรากฏหน้าประโยค เพื่อรับรู้ว่าเครื่องรันมาถึงประโยคนี้แล้ว
6. กดปุ่ม F8 เพื่อรันต่อไป จนสิ้นสุดการทำงานของโปรแกรม
7. ใช้โปรแกรมไฟล์เมเนจero ค้นหาในไดเรกทอรี `/home/pi/asm/Lab5` ว่า ไฟล์ `main.o` ที่เป็นไฟล์อ็อบเจกต์อยู่ในไดเรกทอรีใด .....
8. ใช้โปรแกรมไฟล์เมเนจero ค้นหาในไดเรกทอรี `/home/pi/asm/Lab5` ว่า ไฟล์ `Lab5` ที่เป็นไฟล์โปรแกรมหรือไฟล์ Executable อยู่ในไดเรกทอรีใด .....

### E.3 การพัฒนาโดยใช้ประโยคคำสั่งทีละขั้นตอน

ผู้อ่านควรเข้าใจคำสั่งพื้นฐานในการแปลโปรแกรมภาษา C ที่สร้างขึ้นใน CodeBlocks ก่อนหน้านี้ ตามขั้นตอนต่อไปนี้

1. เปิดโปรแกรม Terminal หน้าต่างใหม่ แล้วป้อนไดเรกทอรีไปยัง `/home/pi/asm/Lab5` โดยใช้คำสั่ง `cd`
  2. ทำการคอมไพล์ (Compile) ไฟล์ชอร์สโค๊ดให้เป็นไฟล์อ็อบเจกต์ (`.o`) โดยเรียกใช้คอมไพล์เวอร์ชื่อ `gcc` ดังนี้
- ```
$ gcc -c main.c
```

ไฟล์ผลลัพธ์ ชื่อ `main.o` จะปรากฏขึ้น ผู้อ่านต้องตรวจสอบโดยใช้คำสั่ง `ls -la` เพื่อตรวจสอบวันที่และขนาดของไฟล์ เปรียบเทียบการใช้งานกับรูปที่ 3.10 จับภาพหน้าต่างที่ได้วางไว้ใต้คำสั่งนี้เพื่อให้ตรวจสอบ

3. ทำการลิงก์ (Link) โดยใช้ gcc ทำหน้าที่เป็นลิงก์เกอร์ (Linker) และแปลงไฟล์ออบเจกต์เป็นไฟล์โปรแกรม (Executable file) โดย

```
$ gcc main.o -o Lab5
```

ไฟล์โปรแกรม ชื่อ Lab5 จะปรากฏขึ้น ผู้อ่านต้องตรวจสอบโดยใช้คำสั่ง ls -la เพื่อตรวจสอบวันที่และขนาดของไฟล์เพื่อเปรียบเทียบกับ man.o จับภาพหน้าต่างที่ได้วางไว้ใต้คำสั่งนี้เพื่อให้ตรวจสอบ

4. รัน (Run) โปรแกรม Lab5 โดยพิมพ์

```
$ ./Lab5
```

5. เปรียบเทียบผลลัพธ์ที่ปรากฏขึ้นว่าตรงกับผลการรันใน CodeBlocks หรือไม่ .....  
อย่างไร .....

## E.4 โครงสร้างของ Makefile

นอกเหนือจากการพัฒนาโปรแกรมด้วย IDE แล้ว การพัฒนาด้วย Makefile จะช่วยให้นักพัฒนามีอิสระในการกำหนดเงื่อนไขต่อไปนี้

- กำหนดชื่อไฟล์ที่จะถูกสร้างขึ้น โดยอาศัยไฟล์ต่างๆ จากส่วนที่เรียกว่า prerequisites นอกจากชื่อไฟล์แล้ว คำสั่ง 'clean' สามารถใช้เป็น target ได้ จึงนิยมใช้สำหรับลบไฟล์ต่างๆ ที่ไม่ต้องการ
- กำหนดคำสั่งหรือการกระทำที่จะใช้รายชื่อไฟล์ใน prerequisites นั้นมาสร้างไฟล์ target ได้ สำเร็จ โดยแต่ละบรรทัดจะต้องเริ่มต้นด้วยปุ่ม tab เสมอ

```
target : prerequisites ...
<tab>recipe
<tab>    ...
<tab>...
```

- target หมายถึง ชื่อไฟล์ที่จะถูกสร้างขึ้น โดยอาศัยไฟล์ต่างๆ จากส่วนที่เรียกว่า prerequisites นอกจากชื่อไฟล์แล้ว คำสั่ง 'clean' สามารถใช้เป็น target ได้ จึงนิยมใช้สำหรับลบไฟล์ต่างๆ ที่ไม่ต้องการ
- recipe หมายถึง คำสั่งหรือการกระทำที่จะใช้รายชื่อไฟล์ใน prerequisites นั้นมาสร้างไฟล์ target ได้ สำเร็จ โดยแต่ละบรรทัดจะต้องเริ่มต้นด้วยปุ่ม tab เสมอ

## E.5 การพัฒนาโดยใช้ Makefile

ตัวอย่างนี้เป็นการสร้าง Makefile เพื่อใช้คอมไพล์และลิงก์โปรแกรมเดิมที่เรามีอยู่ ผู้อ่านจะได้เข้าใจกลไกการทำงานที่ง่ายที่สุดก่อน หลังจากนั้นผู้อ่านสามารถศึกษาเพิ่มเติมด้วยตนเองได้จากเว็บไซต์หรือตัวอย่างโปรแกรมโอเพนซอร์สที่ซับซ้อนขึ้นเรื่อยๆ ต่อไป

1. ในโปรแกรม Terminal ย้ายไดเรกทอรีปัจจุบันไปที่ `/home/pi/asm/Lab5`
2. เรียกใช้โปรแกรม nano ในหน้าต่าง Terminal

```
$ nano
```

กรอกข้อความเหล่านี้ในไฟล์เปล่าโดยใช้ nano

```
Lab5: main.o
      gcc main.o -o Lab5
main.o: main.c
      gcc -c main.c
clean:
      rm *.o
```

3. เมื่อกรอกเสร็จแล้ว ให้ทำการบันทึก หรือ save โดยตั้งชื่อไฟล์ว่า Makefile หรือ makefile อย่างใดอย่างหนึ่งโดยไม่มีนามสกุล หลังจากนั้น และบันทึกในไดเรกทอรี `/home/pi/asm/Lab5` แล้วปิดโปรแกรม nano

4. พิมพ์คำสั่งนี้ใน Terminal

```
$ make clean
```

เพื่อเรียกใช้คำสั่ง `rm *.o` ผ่านทาง Makefile เพื่อลบ (Remove) ไฟล์ที่มีนามสกุล `.o` ทั้งหมด

5. พิมพ์คำสั่งนี้ใน Terminal

```
$ make Lab5
```

เพื่อเรียกใช้คำสั่ง `gcc -c main.c` และ `gcc -g main.c -o Lab5` เพื่อสร้างไฟล์คำสั่ง `Lab5` ที่จะทำงานตามชอร์สโค้ด `main.c` ที่กรอกไป โดยไฟล์ `Lab5` ที่เกิดขึ้นใหม่จะมีโครงสร้างรูปแบบ ELF

6. พิมพ์คำสั่งนี้ใน Terminal

```
$ ls -la
```

เพื่ออ่านค่าเวลาที่ไฟล์ `Lab5` ที่เพิ่งถูกสร้าง โปรดสังเกตสิ่งซึ่ไฟล์ต่างๆ ว่ามีสีอะไรบ้าง และป่งบอกอะไรตามสีนั้นๆ

## 7. พิมพ์คำสั่งนี้ใน Terminal

\$ ./Lab5

เพื่อรันโปรแกรม Lab5 ให้ชี้พิยูปฏิบัติตาม โดย . หมายถึง ..... / หมายถึง .....  
วางแผนหน้าต่างที่ได้วางไว้ต่อคำสั่งนี้เพื่อให้ตรวจสอบ

8. คลิกบนลิงก์ต่อไปนี้ [sunshine2k.de](http://sunshine2k.de) เพื่อเปิดเบราว์เซอร์และอัปโหลดไฟล์ Lab5 ที่ได้จากการคอมไพล์ และลิงก์ก่อนหน้านี้ ตรวจสอบค่า Status: File successfully loaded หรือไม่
9. เปิดเบราว์เซอร์ให้เต็มจอแล้วเลื่อนหน้าจอแสดงผลขึ้นเพื่อเปรียบเทียบกับโครงสร้างของไฟล์ ELF ในรูปที่ 3.14 แคปเจอร์หน้าจอบริเวณเท็กซ์เช็คเมนต์และดาต้าเช็คเมนต์ของ Lab5 วางแผนหน้าต่างที่ได้วางไว้ต่อคำสั่งนี้เพื่อให้ตรวจสอบ

## E.6 การตรวจจับ Overflow คณิตศาสตร์เลขจำนวนเต็มฐานสอง

### E.6.1 เลขจำนวนเต็มฐานสองไม่มีเครื่องหมาย

หัวข้อที่ 2.3.1 กล่าวถึงการบวกเลขจำนวนเต็มชนิดไม่มีเครื่องหมาย 2 จำนวน ผลลัพธ์ที่ได้จะไม่มีเครื่องหมายด้วยเช่นกัน แต่การบวกเลขขนาดใหญ่ที่เข้าใกล้ค่าสูงสุด สามารถเกิดความผิดพลาดได้ เรียกว่า การเกิดโอเวอร์ไฟล์ (Overflow) ในสมการที่ (2.43) ซึ่งเป็นผลสืบเนื่องจากวงจรดิจิทัลที่สามารถประมวลผลได้จำกัด ตามจำนวนบิตข้อมูลสูงสุดที่ทำได้ ในตัวอย่างการแปลงเลขฐานสองเป็นฐานสิบที่ได้แสดงไปแล้ว ยกตัวอย่างเช่น การวนรอบหรือวนลูป (Loop) เพิ่มค่าอย่างต่อเนื่องโดยไม่ระวัง ตามประโยชน์ในภาษา C/C++ ตามการทดลองต่อไปนี้

#### 1. ทดสอบโปรแกรมภาษา C ต่อไปนี้

```
#include <stdio.h>
int main()
{
    unsigned int i=1;
    while (i>0) {
```

```

    i=i+1;
    if (i==0) {
        printf("i was %10u before\n", i-1);
        printf("i is %10u now\n", i);
    }
}
return 0;
}

```

2. อธิบายว่า ประการตัวแปรจึงตั้งค่าเริ่มต้น unsigned int i=1; .....
3. การวนลูปเพิ่มค่า i=i+1 จน i มีค่าเป็นศูนย์แล้วแสดงผลค่าของ i มาทางหน้าจอ เมื่อเกิดเหตุการณ์อะไร เพราะเหตุใด .....
4. จับภาพหน้าต่างที่ได้วางไว้ใต้คำสั่งนี้เพื่อให้ตรวจสอบ

## E.6.2 เลขจำนวนเต็มฐานสองมีเครื่องหมาย 2's Complement

หัวข้อที่ ?? กล่าวถึงการบวกเลขจำนวนเต็มชนิดมีเครื่องหมาย 2 จำนวน ผลลัพธ์ที่ได้จะมีเครื่องหมายด้วย เช่นกัน แต่การบวกเลขขนาดใหญ่ที่เข้าใกล้ค่าสูงสุด สามารถเกิดความผิดพลาดได้ เรียกว่า การเกิดโอเวอร์โฟล์ว (Overflow) ในสมการที่ (2.47) ซึ่งเป็นผลสืบเนื่องจากการจดจำตัวเลขที่สามารถประมวลผลได้จำกัด ตามจำนวนบิต ข้อมูลสูงสุดที่ทำได้ ในตัวอย่างการแปลงเลขฐานสองเป็นฐานสิบที่ได้แสดงไปแล้ว ยกตัวอย่างเช่น การวนรอบหรือ วนลูป (Loop) เพิ่มค่าอย่างต่อเนื่องโดยไม่ระวัง ตามประโยคในภาษา C/C++ ตามการทดลองต่อไปนี้

1. ทดสอบโปรแกรมภาษา C ต่อไปนี้

```

#include <stdio.h>
int main()
{
    int i=1;
    while (i>0) {
        i=i+1;
        if (i<0) {
            printf("i was %10d before\n", i-1);

```

```

        printf("i    is %10d now\n", i);
        break;
    }
}

return 0;
}

```

2. อธิบายว่า ประการตัวแปรจึงตั้งค่าเริ่มต้น int i=1; .....
3. การวนลูปเพิ่มค่า i=i+1 จน i มีค่าน้อยกว่าศูนย์แล้วแสดงผลค่าของ i มาทางหน้าจอ เมื่อเกิดเหตุการณ์อะไร เพรายเหตุใด .....
4. จับภาพหน้าต่างที่ได้วางไว้ใต้คำสั่งนี้เพื่อให้ตรวจสอบ

## E.7 กิจกรรมท้ายการทดลอง

1. งดเปรียบเทียบไฟล์การพัฒนาโปรแกรมในภาคผนวกนี้กับรูปที่ 3.9
2. ใน Terminal จงย้ายไฟล์ที่ได้จากการติดตั้งมาไว้ที่ /home/pi/asm/Lab5

\$ ls -la

เพื่ออ่านรหัสสีของชื่อไฟล์ต่างๆ

3. งดพัฒนาโปรแกรมภาษา C โดยประการตัวแปรและตั้งค่าเริ่มต้น int i=-1 และให้วนลูปลดค่า i=i-1 จน i มีค่าเป็นบวกแล้วแสดงผลค่าของ i ออกมาทางหน้าจอ โดยใช้โปรแกรมในหัวข้อที่ E.6.2 เป็นต้นแบบ
4. งดพัฒนาโปรแกรมภาษา C ให้สามารถอ่านไฟล์ Makefile เพื่อแสดงตัวอักษรในไฟล์ทีละตัวและค่ารหัส ASCII ฐานสิบหกของตัวอักษรนั้นบนหน้าจอ แล้วปิดไฟล์เมื่อเสร็จสิ้น
5. งดพัฒนาโปรแกรมภาษา C เพื่อสั่งพิมพ์เลขอนุกรม Fibonacci โดยรับค่าเลขเป้าหมาย n ซึ่งเกิดจาก  $n = (n-1) + (n-2)$  และรายละเอียดเพิ่มเติมได้จาก [wikipedia](#) ตัวอย่างต่อไปนี้ n=5 และพิมพ์ผลลัพธ์ดังนี้  
1 1 2 3 5

## ภาคผนวก F

# การทดลองที่ 6 การพัฒนาโปรแกรมภาษาแอสเซมบลี

การทดลองนี้คาดว่าผู้อ่านผ่านการเขียนหรือพัฒนาโปรแกรมด้วยภาษา C ใน การทดลองที่ 5 ภาคผนวก E แล้ว และมีความคุ้นเคยกับ IDE จากการพัฒนาโปรแกรมและการติดตั้ง IDE ด้วย Code-Blocks ดังนั้น การทดลองมีวัตถุประสงค์เหล่านี้

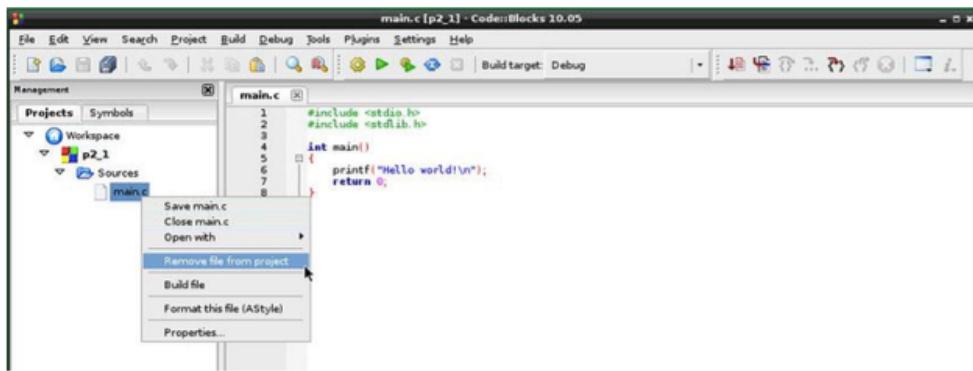
- เพื่อให้เข้าใจการพัฒนาและดีบัก (Debug) โปรแกรมภาษาแอสเซมบลีด้วย IDE ชื่อ CodeBlocks บนระบบปฏิบัติการตระกูลยูนิกซ์
- เพื่อให้เข้าใจความแตกต่างระหว่างการพัฒนาโปรแกรมภาษาแอสเซมบลีด้วย IDE และ Makefile

### F.1 การพัฒนาโดยใช้ IDE

- พิมพ์คำสั่งนี้ในโปรแกรม Terminal เพื่อเริ่มต้นใช้งาน CodeBlocks

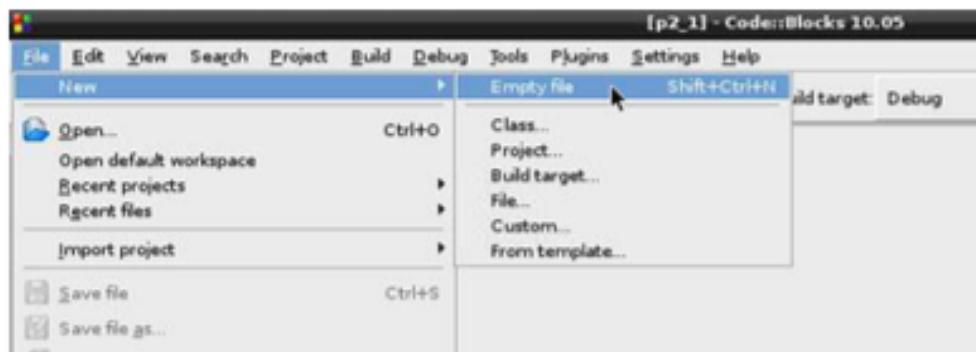
```
$ codeblocks
```

- หน้าต่างหลักจะปรากฏขึ้น หลังจากนั้น ผู้อ่านสามารถสร้างโปรเจ็คใหม่โดยเลือก "Create a new project" ในช่องด้านซ้าย และเลือก "Console application" ในช่องด้านขวาเพื่อสร้างโปรแกรม
- กรอกชื่อโปรเจ็คใหม่ชื่อ Lab6 ในช่อง Project title: และกรอกชื่อไดเรกทอรี /home/pi/asm ในช่อง Folder to create project in: โปรดสังเกตข้อความในช่อง Project filename: ว่าตรงกับ Lab6.cbp ใช่หรือไม่ และจึงกด Next>
- โปรแกรม CodeBlocks จะสร้างไดเรกทอรีต่างๆ ภายใต้ไดเรกทอรีชื่อ /home/pi/asm/Lab6/
- กดปุ่ม "Next>" เพื่อดำเนินการต่อและสุดท้ายจะเป็นขั้นตอนการเลือกคอนฟิกแวร์ชัน (Configuration) สำหรับคอมไฟเลอร์ เลือกออพชัน Debug เน茫สำหรับการเริ่มต้นและแก้ไขข้อผิดพลาด และจึงกดปุ่ม "Finish" เมื่อเสร็จสิ้น
- คลิกชื่อ Workspace ในหน้าต่างด้านซ้ายเพื่อขยายโครงสร้างโปรเจ็คท์แล้วค้นหาไฟล์ชื่อ "main.c" คลิกขวาบนชื่อไฟล์ และเลือกเมนู "Remove file from project" ตามรูปที่ F.1



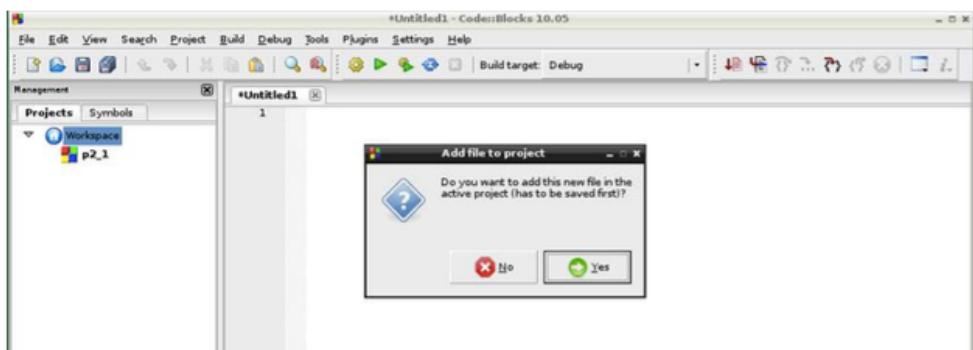
รูปที่ F.1: การย้ายไฟล์ main.c ออกจากโปรเจคท์

7. เพิ่มไฟล์ใหม่ลงในโปรเจคท์โดยกดเมนู File->New->Empty file ตามรูปที่ F.2



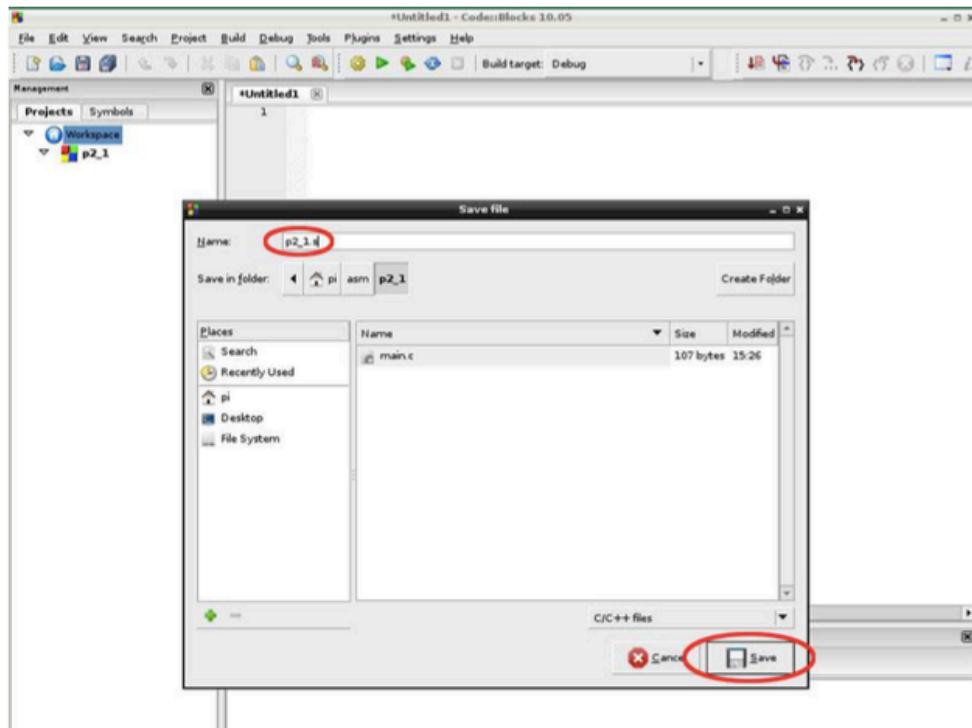
รูปที่ F.2: การเพิ่มไฟล์ใหม่ลงในโปรเจคท์

8. คลิกปุ่ม "Yes" เพื่อยืนยันในรูปที่ F.3



รูปที่ F.3: หน้าต่างกดปุ่ม "Yes" เพื่อยืนยัน

9. หน้าต่าง "Save file" จะปรากฏขึ้น กรอกชื่อไฟล์ว่า main.s และจึงกดปุ่ม "Save" ดังรูปที่ F.4



รูปที่ F.4: หน้าต่าง Save File ชื่อไฟล์ว่า main.s

10. คลิกชื่อ Workspace ในหน้าต่างด้านซ้าย และคลิกขวาเพิ่ม (Add) ไฟล์ main.s เข้าไปในโปรเจคท์
11. ป้อนคำสั่งเหล่านี้ในไฟล์ main.s

```
.global main
main:
    MOV R0, #0
    MOV R1, #2
    MOV R2, #4
    ORR R0, R1, R2
    BX LR
```

12. เลือกเมนู Build->Build เพื่อแปล (Assemble) โปรแกรมที่เขียนให้เป็นโปรแกรมภาษาเครื่อง
13. เลือกเมนู Build->Run เพื่อรันโปรแกรม
14. อ่านและบันทึกประโยชน์ที่เกิดขึ้นในหน้าต่าง Terminal ที่ปรากฏขึ้นมา

## F.2 การดีบักโปรแกรมโดยใช้ IDE

1. ในไฟล์ main.s เลื่อนเครื่องเซอร์เบิร์ทที่มีคำสั่ง ORR R0, R1, R2 คลิกเมนู Debug->breakpoint หรือกดปุ่ม F5 ผู้อ่านจะสังเกตวงกลมสีแดงปรากฏขึ้นด้านซ้ายของคำสั่ง ORR
2. กดเมนู Debug->Debugging Windows->CPU Registers เพื่อแสดงค่าของ CPU register ในหน้าต่างที่ปรากฏขึ้นมาเพิ่มเติม
3. เมื่อพร้อมแล้ว ผู้อ่านสามารถเริ่มต้นการดีบักโดยกดเมนู Debug->Start/Continue หรือกดปุ่ม F8 โปรแกรมจะเริ่มต้นทำงานตั้งแต่ประโยคแรกจนหยุดที่คำสั่ง ORR R0, R1, R2
4. อ่านและบันทึกค่าของ R0 และ PC ในหน้าต่าง CPU Registers
5. ประมวลผลคำสั่งถัดไปโดยกดเมนู Debug->Next Instruction หรือกดปุ่ม Alt+F7 พร้อมกัน
6. อ่านและบันทึกค่าของ R0 และ PC ในหน้าต่าง CPU Registers และสังเกตการเปลี่ยนแปลงที่เกิดขึ้น โดยเปรียบเทียบกับค่า R0 และ PC ในข้อ 4 กับข้อนี้
7. อธิบายว่าเกิดอะไรขึ้นกับค่าของรีจิสเตอร์ R0 และ PC

## F.3 การพัฒนาโดยใช้ประโยคคำสั่งทีละขั้นตอน

ผู้อ่านควรเข้าใจคำสั่งพื้นฐานในการแปลงโปรแกรมภาษาแอสเซมบลีที่สร้างขึ้นใน CodeBlocks ก่อนหน้านี้ ตามขั้นตอนต่อไปนี้

1. ใช้โปรแกรมไฟล์เมเนจเจอร์เพื่อเบราส์ไฟล์ในไดเรกทอรี **/home/pi/asm/Lab6**
2. ดับเบิลคลิกบนชื่อไฟล์ **main.s** เพื่อเปิดอ่านไฟล์และเปรียบเทียบกับไฟล์ที่เขียนในโปรแกรม CodeBlocks
3. เปิดโปรแกรม Terminal หน้าต่างใหม่ แล้วย้ายไดเรกทอรีปัจจุบัน (cd: change directory) ไปยัง **/home/pi/asm/Lab6** โดยใช้คำสั่ง

```
$ cd /home/pi/asm/Lab6
```

4. แปลไฟล์ซอร์สโค้ดให้เป็นไฟล์อ๊อบเจกต์ โดยเรียกใช้คำสั่ง as (assembler) ดังนี้

```
$ as -o main.o main.s
```

5. ใช้คำสั่ง ls -la ใน Terminal เพื่อค้นหาไฟล์อ๊อบเจกต์ชื่อ main.o ว่ามีจริงหรือไม่
6. ทำการลิงก์และแปลงไฟล์อ๊อบเจกต์เป็นไฟล์โปรแกรมโดย

```
$ gcc -o Lab6 main.o
```

7. ใช้คำสั่ง ls -la ใน Terminal เพื่อค้นหาไฟล์โปรแกรมชื่อ Lab6 ว่ามีจริงหรือไม่
8. เรียกโปรแกรม Lab6 โดยพิมพ์

```
$ ./Lab6
```

9. เปรียบเทียบหมายเลขที่ปรากฏขึ้นว่าตรงกับผลการรันใน IDE หรือไม่ อย่างไร

## F.4 การพัฒนาโดยใช้ Makefile

การใช้ **makefile** สำหรับพัฒนาโปรแกรมภาษาแอสเซมบลีคล้ายกับการทดลองที่ 5 ในภาคผนวก E ก่อนหน้านี้

1. เปิดไดเรกทอรี **/home/pi/asm/Lab6** ด้วยโปรแกรมไฟล์เมเนจอร์
2. กดปุ่มขวาบนมาสีในพื้นที่ไดเรกทอรีเพื่อสร้างไฟล์เปล่าใหม่ (New Empty File) โดยกำหนดชื่อ makefile
3. ป้อนข้อความเหล่านี้ลงในไฟล์ makefile:

```
Lab6: main.o
    gcc -o Lab6 main.o
main.o: main.s
    as -o main.o main.s
clean:
    rm *.o Lab6
```

4. บันทึกไฟล์แล้วปิดหน้าต่างบันทึก ผู้อ่านควรตรวจสอบรายชื่อไฟล์ที่อยู่ภายใต้ไดเรกทอรีนี้ว่ามีไฟล์อะไรบ้าง
5. ลบไฟล์อ้อมใจที่มีอยู่โดยใช้คำสั่ง

```
$ make clean
```

ในโปรแกรม Terminal เพื่อเปรียบเทียบหลังจากที่รันคำสั่ง make clean

6. ใช้คำสั่ง ls -la ใน Terminal ค้นหาไฟล์อ้อมใจ main.o และ Lab6 ว่าถูกลบหรือไม่

7. ทำการแอกซ์เพลิค main.s โดยใช้คำสั่ง make ในโปรแกรม Terminal และขอให้สังเกตวันเวลาของไฟล์ต่างๆ

```
$ make
```

8. ใช้คำสั่ง ls -la ใน Terminal เพื่อค้นหาไฟล์ชื่อ main.o และ Lab6 ว่ามีจริงหรือไม่

9. เรียกโปรแกรม Lab6 โดยพิมพ์

```
$ ./Lab6
```

## F.5 กิจกรรมท้ายการทดลอง

1. จงปรับแก้คำสั่ง ORR เป็นคำสั่ง AND ในโปรแกรม main.s และตรวจสอบผลการเปลี่ยนแปลงแล้วจึงอธิบาย
2. จงปรับแก้โปรแกรมใน main.s เป็นดังนี้ จดบันทึกผลการทดลองและอธิบาย

```
.global main
main:
    MOV R5, #1
loop:
    CMP R4, #0
    BLE end
else:
    MOV R5, #2
end:
    MOV R0, R5
    BX LR
```

3. จงปรับแก้โปรแกรมใน main.s เป็นดังนี้ จดบันทึกผลการทดลองและอธิบาย

```
.data
.balign 4
var1: .word 1
.text
.global main
main:
    MOV R1, #2
```

```
LDR R2, varladdr  
STR R1, [R2]  
LDR R0, [R2]  
BX LR  
varladdr: .word var1
```

## ภาคผนวก G

### การทดลองที่ 7 การเรียกใช้และสร้างฟังก์ชันในโปรแกรมภาษาแอสเซมบลี

ผู้อ่านควรจะต้องทำความเข้าใจเนื้อหาของบทที่ 4 หัวข้อ 4.8 และ ทำการทดลองที่ 5 และการทดลองที่ 6 ในภาคผนวกก่อนหน้า โดยการทดลองนี้จะเสริมความเข้าใจของผู้อ่านให้เพิ่มมากขึ้น ตามวัตถุประสงค์เหล่านี้

- เพื่อพัฒนาโปรแกรมภาษาแอสเซมบลีเรียกใช้งานตัวแปรเดี่ยวหรือตัวแปรสเกลาร์ (Scalar)
- เพื่อพัฒนาโปรแกรมแอสเซมบลีเรียกใช้งานตัวแปรชุดหรืออาร์เรย์ (Array)
- เพื่อเรียกใช้ฟังก์ชันจากไลบรารีพื้นฐานด้วยโปรแกรมภาษาแอสเซมบลี ในหัวข้อที่ 4.8
- เพื่อสร้างฟังก์ชันเสริมในโปรแกรมภาษาแอสเซมบลี

#### G.1 การใช้งานตัวแปรในดาต้าเซ็กเมนต์

ตัวแปรต่างๆ ที่ประกาศโดยใช้ชื่อ เลเบล ต้องการพื้นที่ในหน่วยความจำสำหรับจัดเก็บค่าตามที่ได้สรุปในตารางที่ 2.1 ตัวแปรมีสองชนิดแบ่งตามพื้นที่ในการจัดเก็บค่า คือ

- ตัวแปรชนิดโกลบอล (Global Variable) อาศัยพื้นที่สำหรับเก็บค่าของตัวแปรเหล่านี้ เรียกว่า ดาต้าเซ็กเมนต์ (Data Segment) ซึ่งผู้เขียนได้กล่าวไว้แล้วในบทที่ 4 และ
- ตัวแปรชนิดโลคอล (Local Variable) อาศัยพื้นที่ภายในสแต็กเซ็กเมนต์ (Stack Segment) สำหรับจัดเก็บค่าชั่วคราว เป็นจากฟังก์ชันคือชุดคำสั่งย่อที่ฟังก์ชัน main() ในภาษา C หรือ main: ในภาษาแอสเซมบลีเป็นผู้เรียกใช้ และเมื่อทำงานเสร็จสิ้น ฟังก์ชันนั้นจะต้องรีเทิร์นกลับมาหาฟังก์ชัน main() หรือ main: ในที่สุด ดังนั้น ตัวแปรชนิดโลคอลจึงใช้พื้นที่จัดเก็บค่าในสแต็กเพرمภายในสแต็กเซ็กเมนต์แทน เพราสแต็กเพرمจะมีการจองพื้นที่ (PUSH) และคืนพื้นที่ (POP) ในรูปแบบ Last In First Out ตามที่อธิบายในหัวข้อที่ 3.3.3 ทำให้มีจำเป็นต้องใช้พื้นที่ในบริเวณดาต้าเซ็กเมนต์ ผู้อ่านสามารถทำความเข้าใจหัวข้อนี้เพิ่มเติมในการทดลองที่ 8 ภาคผนวก H

### G.1.1 การโหลดค่าตัวแปรเดี่ยวยจากหน่วยความจำมาพักในรีจิสเตอร์

1. ย้ายไฟล์asm เข้าไปใน /home/pi/asm โดยใช้คำสั่ง \$ cd /home/pi/asm
2. สร้างโฟลเดอร์ Lab7 โดยใช้คำสั่ง \$ mkdir Lab7
3. ย้ายไฟล์asm เข้าไปใน Lab7
4. ตรวจสอบว่าได้รูปปัจจุบันโดยใช้คำสั่ง pwd
5. สร้างไฟล์ Lab7\_1.s ตามชอร์สโค้ดต่อไปนี้ ผู้อ่านสามารถข้ามประযุคคอมเมนต์ได้ เมื่อทำการเข้าใจแต่ละคำสั่งแล้ว

```

.data
    .balign 4          @ Request 4 bytes of space
fifteen: .word 15      @ fifteen = 15

    .balign 4          @ Request 4 bytes of space
thirty: .word 30       @ thirty = 30

.text
    .global main
main:
    LDR R1, addr_fifteen      @ R1 <- address_fifteen
    LDR R1, [R1]                @ R1 <- Mem[address_fifteen]
    LDR R2, addr_thirty       @ R2 <- address_thirty
    LDR R2, [R2]                @ R2 <- Mem[address_thirty]
    ADD R0, R1, R2
end:
    BX LR

addr_fifteen: .word fifteen
addr_thirty: .word thirty

```

6. สร้าง makefile ภายใต้โฟลเดอร์ Lab7 และกรอกคำสั่งดังนี้

Lab7\_1:

```
gcc -o Lab7_1 Lab7_1.s
```

7. ทำการ make และรันโปรแกรมโดยใช้คำสั่ง

```
$ make Lab7_1
$ ./Lab7_1
```

8. สร้างไฟล์ **Lab7\_2.s** ตามโค้ดต่อไปนี้จากไฟล์ **Lab7\_1.s** ผู้อ่านสามารถข้ามประยุคคอมเมนต์ได้ เมื่อทำความเข้าใจแต่ละคำสั่งแล้ว

```

.data
    .balign 4          @ Request 4 bytes of space
fifteen: .word 0      @ fifteen = 0
    .balign 4          @ Request 4 bytes of space
thirty: .word 0       @ thirty = 0

.text
.global main
main:
    LDR R1, addr_fifteen  @ R1 <- address_fifteen
    MOV R3, #15           @ R3 <- 15
    STR R3, [R1]          @ Mem[address_fifteen] <- R3
    LDR R2, addr_thirty  @ R2 <- address_thirty
    MOV R3, #30           @ R3 <- 30
    STR R3, [R2]          @ Mem[address_thirty] <- R2

    LDR R1, addr_fifteen  @ Load address
    LDR R1, [R1]          @ R1 <- Mem[address_fifteen]
    LDR R2, addr_thirty  @ Load address
    LDR R2, [R2]          @ R2 <- Mem[address_thirty]
    ADD R0, R1, R2

end:
    BX LR

@ Labels for addresses in the data section
addr_fifteen: .word fifteen
addr_thirty: .word thirty

```

9. เพิ่มประยุคต่อไปนี้ใน makefile ให้รองรับ Lab7\_2

Lab7\_2:

```
gcc -o Lab7_2 Lab7_2.s
```

10. ทำการ make และรันโปรแกรมโดยใช้คำสั่ง

```
$ make Lab7_2
```

```
$ ./Lab7_2
```

บันทึกผลและอธิบายผลที่เกิดขึ้นเพื่อเปรียบเทียบกับข้อที่แล้ว

### G.1.2 การใช้งานตัวแปรชุดหรืออาร์เรย์ ชนิด word

ภาษาแอสเซมบลีจะกำหนดชนิดตามหลังชื่อตัวแปร เช่น .word, .hword, และ .byte ใช้กำหนดขนาดของตัวแปรนั้นๆ ขนาด 32, 16 และ 8 บิตตามลำดับ ยกตัวอย่าง คือ:

```
numbers: .word 1, 2, 3, 4
```

เป็นการประกาศและตั้งค่าตัวแปรชนิดอาร์เรย์ของ word ซึ่งต้องการพื้นที่ 4 ไบต์ต่อข้อมูลหนึ่งตำแหน่ง ซึ่งจะตรงกับประโยชน์ต่อไปนี้ในภาษา C

```
int numbers={1, 2, 3, 4}
```

- สร้างไฟล์ Lab7\_3.s ตามโค้ดต่อไปนี้ ผู้อ่านสามารถเข้ามาระบุคอมเมนต์ได้ เมื่อทำการเข้าใจแต่ละคำสั่งแล้ว

```
.data
primes:
    .word 2
    .word 3
    .word 5
    .word 7

.text
.global main

main:
    LDR R3, =primes    @ Load the address for the data in R3
    LDR R0, [R3, #4]    @ Get the next item in the list
end:
    BX LR
```

- เพิ่มประโยชน์ต่อไปนี้ใน makefile ให้รองรับ Lab7\_3

Lab7\_3:

```
gcc -o Lab7_3 Lab7_3.s
```

- ทำการ make และรันโปรแกรมโดยใช้คำสั่ง

```
$ make Lab7_3
$ ./Lab7_3
```

### G.1.3 การใช้งานตัวแปรอาร์เรย์ชนิด byte

คำสั่ง LDRB ทำงานคล้ายกับคำสั่ง LDR แต่เป็นการอ่านค่าของตัวแปรอาร์เรย์ชนิด byte

- สร้างไฟล์ Lab7\_4.s ตามโค้ดต่อไปนี้ ผู้อ่านสามารถข้ามประযุคคอมเมนต์ได้ เมื่อทำการเข้าใจแต่ละคำสั่งแล้ว

```
.data
numbers:      .byte 1, 2, 3, 4, 5

.text
.global main
main:
    LDR R3, =numbers        @ Get address
    LDRB R0, [R3, #2]       @ Get next two bytes
end:
    BX LR
```

- เพิ่มประยุคต์อไปนี้ใน makefile ให้รองรับ Lab7\_4

Lab7\_4:

```
gcc -o Lab7_4 Lab7_4.s
```

- ทำการ make และรันโปรแกรมโดยใช้คำสั่ง

```
$ make Lab7_4
$ ./Lab7_4
```

#### G.1.4 การเรียกใช้ฟังก์ชันและตัวแปรชนิดประโยครหัส ASCII

ฟังก์ชันสำหรับรูปที่เข้าใจง่ายและใช้สำหรับเรียนรู้การพัฒนาโปรแกรมภาษา C เป็นต้น คือ ฟังก์ชัน printf ซึ่งถูกกำหนดอยู่ในไฟล์ヘดเดอร์ stdio.h ตามตัวอย่างซอฟต์แวร์สโค็ต ในรูปที่ 3.9 และการทดลองที่ 5 ภาคผนวก E ในการทดลองต่อไปนี้ ผู้อ่านจะสังเกตเห็นว่าการเรียกใช้ฟังก์ชัน printf ในภาษาแอลซีเอ็มบี โดยอาศัยตัวแปรชนิดประโยค (String) ในรูปที่ 2.11 โดยใช้คำสำคัญ (Key Word) เหล่านี้ คือ .ascii และ .asciz ตัวแปรชนิด asciz จะมีตัวอักษรพิเศษ เรียกว่า อักขระนัล NULL หรือ /0 ปิดท้ายประโยคเสมอ และอักขระ NULL จะมีรหัส ASCII เท่ากับ 00<sub>16</sub> ตามตารางรหัสแอสกิ ในรูปที่ 2.12

- กรอกคำสั่งต่อไปนี้ลงในไฟล์ใหม่ชื่อ Lab7\_5.s และทำความเข้าใจประโยคคอมเม้นต์แต่ละบรรทัด

```
.data
.balign 4
question: .asciz "What is your favorite number?"

.balign 4
message: .asciz "%d is a great number \n"

.balign 4
pattern: .asciz "%d"

.balign 4
number: .word 0

.balign 4
lr_bu: .word 0

.text @ Text segment begins here

@ Used by the compiler to tell libc where main is located
.global main
.func main

main:
    @ Backup the value inside Link Register
    LDR R1, addr_lr_bu
    STR lr, [R1]      @ Mem[addr_lr_bu] <- LR

    @ Load and print question
    LDR R0, addr_question
    BL printf
```

```

@ Define pattern to scanf and where to store number
LDR R0, addr_pattern
LDR R1, addr_number
BL scanf

@ Print the message with number
LDR R0, addr_message
LDR R1, addr_number
LDR R1, [R1]
BL printf

@ Load the value of lr_bu to LR
LDR lr, addr_lr_bu
LDR lr, [lr]      @ LR <- Mem[addr_lr_bu]
BX lr

@ Define addresses of variables
addr_question: .word question
addr_message: .word message
addr_pattern: .word pattern
addr_number: .word number
addr_lr_bu: .word lr_bu

@ Declare printf and scanf functions to be linked with
.global printf
.global scanf

```

## 2. เพิ่มประโยชน์ใน makefile ให้รองรับ Lab7\_5

Lab7\_5:

```
gcc -o Lab7_5 Lab7_5.s
```

## 3. ทำการ make และรันโปรแกรมโดยใช้คำสั่ง

```
$ make Lab7_5
$ ./Lab7_5
```

## G.2 การสร้างฟังก์ชันเสริมด้วยภาษาแอสเซมบลี

หัวข้อที่ [4.8](#) อธิบายโดยวิธีการทำงานของฟังก์ชัน โดยใช้งานรีจิสเตอร์ R0 - R12 ดังนี้

- รีจิสเตอร์ R0, R1, R2, และ R3 การส่งผ่านพารามิเตอร์ผ่านทางรีจิสเตอร์ R0 ถึง R3 ตามลำดับ ไปยังฟังก์ชันที่ถูกเรียก (Callee Function) ฟังก์ชันบางตัวต้องการจำนวนพารามิเตอร์มากกว่า 4 ค่า โปรแกรมเมอร์สามารถส่งพารามิเตอร์ผ่านทางสเต็ปโดยคำสั่ง PUSH หรือคำสั่งที่ใกล้เคียง
- รีจิสเตอร์ R0 สำหรับรีเทิร์นหรือส่งค่ากลับไปหาฟังก์ชันผู้เรียก (Caller Function)
- R4 - R12 สำหรับการใช้งานทั่วไป การใช้งานรีจิสเตอร์เหล่านี้ ควรตั้งค่าเริ่มต้นก่อนแล้วจึงสามารถนำค่าไปคำนวณต่อได้
- รีจิสเตอร์เฉพาะ ได้แก่ Stack Pointer (SP หรือ R13) Link Register (LR หรือ R14) และ Program Counter (PC หรือ R15) โปรแกรมเมอร์จะต้องเก็บค่าของรีจิสเตอร์เหล่านี้เก็บไว้ (Back up) ในสเต็ป โดยเฉพาะรีจิสเตอร์ LR ก่อนเรียกใช้ฟังก์ล์ LR ตามที่อธิบายในหัวข้อที่ [4.8.2](#)

ผู้อ่านสามารถสำเนาซอฟต์แวร์สโคดูในการทดลองที่แล้วมาปรับแก้เป็นการทดลองนี้ได้

- ปรับแก้ **Lab7\_5.s** ที่มีให้เป็นไฟล์ใหม่ชื่อ **Lab7\_6.s** ดังต่อไปนี้

```
.data
@ Define all the strings and variables
.balign 4
get_num_1: .asciz "Number 1 :\n"

.balign 4
get_num_2: .asciz "Number 2 :\n"

@ printf and scanf use %d in decimal numbers
.balign 4
pattern: .asciz "%d"

@ Declare and initialize variables: num_1 and num_2
.balign 4
num_1: .word 0

.balign 4
num_2: .word 0

@ Output message pattern
.balign 4
output: .asciz "Result of %d + %d = %d\n"
```

```

@ Variables to backup link register
.balign 4
lr_bu: .word 0

.balign 4
lr_bu_2: .word 0

.text
sum_func:
    @ Save (Store) Link Register to lr_bu_2
    LDR R2, addr_lr_bu_2
    STR lr, [R2]      @ Mem[addr_lr_bu_2] <- LR

    @ Sum values in R0 and R1 and return in R0
    ADD R0, R0, R1

    @ Load Link Register from back up 2
    LDR lr, addr_lr_bu_2
    LDR lr, [lr]      @ LR <- Mem[addr_lr_bu_2]

    BX lr

    @ address of Link Register back up 2
addr_lr_bu_2: .word lr_bu_2

    @ main function
    .global main

main:
    @ Store (back up) Link Register
    LDR R1, addr_lr_bu
    STR lr, [R1]      @ Mem[addr_lr_bu] <- LR

    @ Print Number 1 :
    LDR R0, addr_get_num_1
    BL printf

    @ Get num_1 from user via keyboard
    LDR R0, addr_pattern

```

```

LDR R1, addr_num_1
BL scanf

@ Print Number 2 :
LDR R0, addr_get_num_2
BL printf

@ Get num_2 from user via keyboard
LDR R0, addr_pattern
LDR R1, addr_num_2
BL scanf

@ Pass values of num_1 and num_2 to sum_func
LDR R0, addr_num_1
LDR R0, [R0]      @ R0 <- Mem[addr_num_1]
LDR R1, addr_num_2
LDR R1, [R1]      @ R1 <- Mem[addr_num_2]
BL sum_func

@ Copy returned value from sum_func to R3
MOV R3, R0      @ to printf

@ Print the output message, num_1, num_2 and result
LDR R0, addr_output
LDR R1, addr_num_1
LDR R1, [R1]
LDR R2, addr_num_2
LDR R2, [R2]
BL printf

@ Restore Link Register to return
LDR lr, addr_lr_bu
LDR lr, [lr]      @ LR <- Mem[addr_lr_bu]
BX lr

@ Define pointer variables
addr_get_num_1: .word get_num_1
addr_get_num_2: .word get_num_2
addr_pattern:   .word pattern
addr_num_1:     .word num_1

```

```

addr_num_2:      .word num_2
addr_output:     .word output
addr_lr_bu:      .word lr_bu

@ Declare printf and scanf functions to be linked with
.global printf
.global scanf

```

2. เพิ่มประโยชน์ใน makefile ให้รองรับ Lab7\_6 ดังนี้

Lab7\_6:

```
gcc -o Lab7_6 Lab7_6.s
```

3. ทำการ make และรันโปรแกรมโดยใช้คำสั่ง

```
$ make Lab7_6
$ ./Lab7_6
```

4. ระบุชอร์สโค้ดใน Lab7\_6.s ว่าตรงกับประโยชน์ภาษา C ต่อไปนี้

```
int num1, num2
```

5. ระบุชอร์สโค้ดใน Lab7\_6.s ว่าตรงกับประโยชน์ภาษา C ต่อไปนี้ **sum = num1 + num2**

6. มีการแบ็กอัปค่าของ LR ลงในสแต็กหรือไม่ หากไม่มีแล้วในการทดลองเก็บค่าของ LR ไว้ที่ใด เพราะเหตุใด

7. วิธีการแบ็กอัปค่า LR ในการทดลองสามารถใช้กับฟังก์ชันชนิดรีקורסีฟ (Recursive) ได้หรือไม่ เพราะเหตุใด

### G.3 กิจกรรมท้ายการทดลอง

1. จงเปรียบเทียบการเรียกใช้พังค์ชัน printf และ scanf ในภาษา C จากการทดลองที่ 5 ภาคผนวก E กับการทดลองนี้ด้านการส่งค่าparamiเตอร์
2. จงบอกความแตกต่างระหว่างการส่งค่าparamiเตอร์แบบ Pass by Values และ Pass by Reference
3. จงยกตัวอย่างการเรียกใช้ฟังก์ชัน printf ด้วยการส่งค่าparamiเตอร์แบบ Pass by Values
4. จงยกตัวอย่างการเรียกใช้ฟังก์ชัน scanf ด้วยการส่งค่าparamiเตอร์แบบ Pass by Reference
5. จงพัฒนาโปรแกรมด้วยภาษา C เพื่อรับตัวเลขจำนวน 2 ตัวจากผู้ใช้ผ่านทางคีย์บอร์ด เรียกว่า A และ B แล้วคำนวณและแสดงผลลัพธ์ ตามตารางต่อไปนี้ "A % B = <Result>".

| Input | Output     |
|-------|------------|
| 5 2   | 5 % 2 = 1  |
| 18 6  | 18 % 6 = 0 |
| 5 10  | 5 % 10 = 5 |
| 10 5  | 10 % 5 = 0 |

6. จงเปรียบเทียบฟังก์ชัน scanf และ printf ในการทดลองนี้กับการทดลองที่ 5 ภาคผนวก E
7. จงพัฒนาโปรแกรมด้วยภาษา C เพื่อรับตัวเลขจำนวน 2 ตัวจากผู้ใช้ผ่านทางคีย์บอร์ด เรียกว่า A และ B แล้วคำนวณหาค่า หารร่วมมาก (Greatest Common Divisor) หรือ หرم (GCD) และแสดงผลลัพธ์ตามตัวอย่างในตารางต่อไปนี้

| Input | Output |
|-------|--------|
| 5 2   | 1      |
| 18 6  | 6      |
| 49 42 | 7      |
| 81 18 | 9      |

8. จงพัฒนาโปรแกรมด้วยภาษา Assembly เพื่อรับตัวเลขจำนวน 2 ตัวจากผู้ใช้ผ่านทางคีย์บอร์ด เรียกว่า A และ B และแสดงผลลัพธ์ A หรือ B ที่มีค่ามากกว่าด้วยคำสั่งภาษาแอสเซมบลี
9. จงพัฒนาโปรแกรมด้วยภาษา Assembly เพื่อรับตัวเลขจำนวน 2 ตัวจากผู้ใช้ผ่านทางคีย์บอร์ด เรียกว่า A และ B และแสดงผลลัพธ์ค่า A modulus B ซึ่งเท่ากับ ค่าเศษจากการคำนวณ A/B ด้วยคำสั่งภาษาแอสเซมบลี
10. จงพัฒนาโปรแกรมด้วยภาษา Assembly เพื่อรับตัวเลขจำนวน 2 ตัวจากผู้ใช้ผ่านทางคีย์บอร์ด เรียกว่า A และ B แล้วคำนวณหาค่า หารร่วมมาก (Greatest Common Divisor) หรือ หرم (GCD) ด้วยคำสั่งภาษาแอสเซมบลีและแสดงผลลัพธ์ ตามตารางในข้อ 6

## ภาคผนวก H

# การทดลองที่ 8 การพัฒนาโปรแกรมภาษาแอสเซมบลีขั้นสูง

การพัฒนาโปรแกรมภาษาแอสเซมบลีขั้นสูง จะเน้นการพัฒนาร่วมกับภาษา C เพื่อเพิ่มศักยภาพของโปรแกรมภาษา C ให้ทำงานได้มีประสิทธิภาพยิ่งขึ้น โดยเฉพาะฟังก์ชันที่สำคัญและต้องเชื่อมต่อกับ hardware อย่างลึกซึ้ง และถ้ามีประสบการณ์การดีบักโปรแกรมภาษา C จากการทดลองที่ 5 จะยิ่งทำให้ผู้อ่านเข้าใจการทดลองนี้ได้เพิ่มขึ้น ดังนั้น การทดลองมีวัตถุประสงค์เหล่านี้

- เพื่อฝึกการดีบักโปรแกรมภาษาแอสเซมบลีโดยใช้โปรแกรม GDB แบบคอมมานด์ไลน์ (Command Line)
- เพื่อพัฒนาพัฒนาโปรแกรมแอสเซมบลีโดยใช้ Stack Pointer (SP) หรือ R13
- เพื่อพัฒนาโปรแกรมภาษาแอสเซมบลีร่วมกับภาษา C
- เพื่อเสริมความเข้าใจเรื่องเวอร์ชวลเมโมรีในหัวข้อที่ [5.2](#)

## H.1 ดีบักเกอร์ GDB

ดีบัก เกอร์ เป็น โปรแกรม คอมพิวเตอร์ ทำ หน้าที่ รัน โปรแกรม ที่ กำลัง พัฒนา เพื่อ ให้ โปรแกรม เมอร์ตริว สอบการทำงานได้ลึกซึ้งยิ่งขึ้น ทำให้โปรแกรมเมอร์สามารถเข้าใจการทำงานของโปรแกรมอย่างถ่องแท้ และหากโปรแกรมมีปัญหาหรือ ดีบัก ที่บรรทัดไหน ตำแหน่งใด ดีบักเกอร์เป็นเครื่องมือที่จะช่วยแก้ปัญหานั้นได้ในที่สุด

GDB เป็นดีบักเกอร์มาตรฐานทำงานในระบบปฏิบัติการยูนิกซ์ สามารถช่วยโปรแกรมเมอร์แก้ปัญหาของโปรแกรมที่พัฒนาจากภาษา C/C++ รวมถึงภาษาแอสเซมบลีของซีพียูนั้นๆ เช่น แอสเซมบลีของ ARM บนบอร์ด Pi นี้

ผู้อ่านสามารถย้อนกลับไปศึกษาการทดลองที่ 5 หัวข้อ [E.2](#) และการทดลองที่ 6 หัวข้อ [F.2](#) อีกรอบ เพื่อสังเกตรายละเอียดการสร้างโปรเจคที่ได้ว่า เราได้เลือกใช้ GDB เป็นดีบักเกอร์ ผู้อ่านสามารถเรียนรู้การดีบักโปรแกรมแอสเซมบลี พร้อมๆ กับทำความเข้าใจคำสั่งใน GDB ไปพร้อมๆ กัน ดังนี้

- เปิดโปรแกรม Terminal และย้ายไดเรกทอรีไปที่ `/home/pi/asm`
- สร้างไดเรกทอรีใหม่ชื่อ `Lab8`
- สร้างไฟล์ชื่อ `Lab8_1.s` ด้วยเทกซ์อีดิเตอร์ nano เพื่อกรอกคำสั่งภาษาแอสเซมบลี ต่อไปนี้

```

.global main

main:
    MOV R0, #0
    MOV R1, #1
    B _continue_loop

_loop:
    ADD R0, R0, R1
_continue_loop:
    CMP R0, #9
    BLE _loop

end:
    BX LR

```

#### 4. สร้าง makefile และกรอกประโยชน์ค่าสั่งต่อไปนี้

```

debug: Lab8_1
    as -g -o Lab8_1.o Lab8_1.s
    gcc -o Lab8_1 Lab8_1.o
    gdb Lab8_1

```

บันทึกไฟล์และออกจากโปรแกรม nano อีดิเตอร์

#### 5. รันคำสั่งต่อไปนี้ เพื่อทดสอบว่า makefile ถูกต้องหรือไม่ หากถูกต้องโปรแกรม Lab8\_1 จะรันได้ GDB เพื่อให้ผู้อ่านดีบักโปรแกรม

```
$ make debug
```

#### 6. พิมพ์คำสั่ง list หลังสัญลักษณ์ (gdb) เพื่อแสดงคำสั่งภาษาแอสเซมบลีที่จะ execute ทั้งหมด

```
(gdb) list
```

ค้นหาตำแหน่งของคำสั่ง CMP R0, #9 ว่าอยู่ ณ บรรทัดที่เท่าไหร่ สมมติให้เป็นตัวแปร x เพื่อใช้ประกอบการทดลองตัดไป

#### 7. ตั้งค่าเบรกพอยน์เพื่อยุดการรันโปรแกรมชั่วคราว และเปิดโอกาสให้โปรแกรมเมอร์สามารถตรวจสอบค่าของรีจิสเตอร์ต่างๆ ได้ โดยใช้คำสั่ง

```
(gdb) b x
```

โดย x คือ หมายเลขบรรทัดที่คำสั่ง CMP R0, #9 ตั้งอยู่

### 8. รันโปรแกรม โดยพิมพ์คำสั่งต่อไปนี้ บันทึกและอธิบายผลลัพธ์

(gdb) run

จะได้ผลตอบรับจาก GDB ดังนี้

```
Breakpoint 1, _continue_loop () at Lab8_1.s: x
```

โปรดสังเกตค่า x เป็นหมายเลขบรรทัดที่ตรงกับคำสั่งใด

9. โปรดสังเกตว่า (gdb) ปรากฏขึ้นแสดงว่าโปรแกรมหยุดที่เบรกพอยน์แล้ว พิมพ์คำสั่ง (gdb) info r เพื่อแสดงค่าภายในรีจิสเตอร์ต่างๆ ทั้งหมด และบันทึกค่าฐานสิบหกของรีจิสเตอร์เหล่านี้ r0, r1, r9, sp, pc, cpsr หลังรันโปรแกรม เพื่อเปรียบเทียบในลำดับถัดไป

(gdb) info r

|      |            |                            |
|------|------------|----------------------------|
| r0   | 0x0        | 0                          |
| r1   | 0x1        | 1                          |
| r2   | 0x7effefec | 2130702316                 |
| r3   | 0x10408    | 66568                      |
| r4   | 0x10428    | 66600                      |
| r5   | 0x0        | 0                          |
| r6   | 0x102e0    | 66272                      |
| r7   | 0x0        | 0                          |
| r8   | 0x0        | 0                          |
| r9   | 0x0        | 0                          |
| r10  | 0x76fff000 | 1996484608                 |
| r11  | 0x0        | 0                          |
| r12  | 0x7effef10 | 2130702096                 |
| sp   | 0x7effee90 | 0x7effee90                 |
| lr   | 0x76e7a678 | 1994892920                 |
| pc   | 0x1041c    | 0x1041c <_continue_loop+4> |
| cpsr | 0x80000010 | -2147483632                |

จงตอบคำถามต่อไปนี้ประกอบความเข้าใจ

- อธิบายรายงานบนหน้าจอว่าคอลัมน์แต่ละ colum มีความหมายอย่างไร และแตกต่างกับหน้าจอของผู้อ่านอย่างไร

- เหตุใดรีจิสเตอร์ cpsr มีค่าเป็นเลขฐานสิบในคอลัมน์ขวาสุดมีค่าติดลบ หมายเหตุ ศึกษาเรื่อง เลขจำนวนเต็มฐานสองชนิดมีเครื่องหมาย แบบ 2's Complement ในหัวข้อที่ 2.2.2

- พิมพ์คำสั่ง (**gdb**) **c[ontinue]** เพื่อรันโปรแกรมต่อไปจนกว่าจะวนรอบกลับมาที่เบรกพอยน์ที่ตั้งไว้
- พิมพ์คำสั่ง (**gdb**) **info r** เพื่อแสดงค่าภายในรีจิสเตอร์ต่างๆ ทั้งหมด และบันทึกค่าของรีจิสเตอร์เหล่านี้ **r0, r1, r9, sp, pc, cpsr** เพื่อสังเกตการเปลี่ยนแปลงเทียบกับข้อ 9
- เริ่มต้นการทดลองโดยพิมพ์คำสั่งต่อไปนี้เพื่อหาว่า เลเบล **\_loop** ตรงกับหน่วยความจำตำแหน่งใด

```
(gdb) disassemble _loop
```

บันทึกผลที่ได้โดย หมายเลขอ้ายสุด คือ แอดเดรสในหน่วยความจำ ที่คำสั่งนั้นบรรจุอยู่ หมายเลขอตำแหน่ง ถัดมา คือ จำนวนไบต์นับจากจุดเริ่มต้นของชื่อเลเบลนั้น แล้วตรวจสอบว่าเลเบล main อยู่ห่างจาก ตำแหน่งเริ่มต้นของโปรแกรมกี่ไบต์

Dump of assembler code for function **\_loop**:

```
0x00010414 <+0>: add r0, r0, r1
End of assembler dump.
```

- พิมพ์คำสั่ง (**gdb**) **c[ontinue]** เพื่อรันโปรแกรมต่อไปจนกว่าจะวนรอบกลับมาที่เบรกพอยน์ที่ตั้งไว้อีก รอบ
- คำสั่ง **x/ [count] [format] [address]** แสดงค่าใน หน่วยความจำ ณ ตำแหน่ง address เป็นต้นไป เป็น จำนวน /count ตาม format ที่ต้องการ ยกตัวอย่างเช่น **x/10i main** คือ แสดงค่าในหน่วยความจำ ณ ตำแหน่งเลเบล main จำนวน 10 ค่าตามรูปแบบ instruction ดังตัวอย่างต่อไปนี้

```
(gdb) x/10i main
0x10408 <main>: mov r0, #0
0x1040c <main+4>: mov r1, #1
0x10410 <main+8>: b 0x10418 <_continue_loop>
0x10414 <_      >: add r0, r0, r1
0x10418 <_continue_loop>: cmp r0, #9
=> 0x1041c <_continue_loop+4>: ble 0x10414 <_      >
0x10420 <end>: mov r7, #1
0x10424 <end+4>: svc 0x00000000
0x10428 <__libc_csu_init>: push {r4, r5, r6, r7, r8, r9, r10, lr}
0x1042c <__libc_csu_init+4>: mov r7, r0
```

จงตอบคำถามต่อไปนี้

- เติมตัวอักษรที่เว้นว่างไว้จากหน้าจอของผู้อ่านในเครื่องหมาย <\_> สองตำแหน่ง
  - อธิบายว่า หมายเลขอารบิกที่มาแทนที่ <\_> ได้อย่างไร
  - โปรดสังเกต และ อธิบายว่า เครื่องหมายลูกศร => ด้านซ้ายสุดหน้าบรรทัดคำสั่ง หมายถึงอะไร
- .....

15. คำสั่ง **r[tep]** i ระหว่างที่เบรกการรันโปรแกรม ผู้ใช้สามารถสั่งให้โปรแกรมทำงานต่อเพียง i คำสั่ง เพื่อตรวจสอบ

16. คำสั่ง **n[ext]** i ทำงานคล้ายคำสั่ง **step** i แต่ถ้าคำสั่งต่อไปที่จะทำงานเป็นการเรียกฟังก์ชัน คำสั่งนี้เรียกใช้ฟังก์ชันนั้นจนสำเร็จ แล้วจึงเบรกให้ผู้ใช้ตรวจสอบ

17. พิมพ์คำสั่ง **i[nfo] ib[reak]** เพื่อแสดงรายการเบรกพอยน์ทั้งหมดที่ตั้งไว้ก่อนหน้า ดังนี้

```
(gdb) i b
Num      Type            Disp Enb Address      What
1        breakpoint      keep y    0x0001041c Lab8_1.s:_
breakpoint already hit _ times
```

ผู้อ่านจะต้องทำความเข้าใจรายงานที่ได้บนหน้าจอ โดยเฉพาะคอลัมน์ Address และ What โดยเติมตัวอักษรลงในช่องว่าง \_ ทึ่งสองช่อง

18. คำสั่ง **d[elete] b[reakpoints] number** ลบการตั้งเบรกพอยน์ที่บรรทัด number ที่ตั้งไว้ก่อนหน้า หากผู้อ่านต้องการลบเบรกพอยน์ทั้งหมดพร้อมกันโดยพิมพ์

```
(gdb) d
Delete all breakpoints? (y or n)
```

แล้วตอบ y เพื่อยืนยัน

19. พิมพ์คำสั่ง **(gdb) c** เพื่อรันโปรแกรมต่อไปจนเสร็จสิ้นจะได้ผลลัพธ์ต่อไปนี้

```
(gdb) c
Continuing.
[Inferior 1 (process 1688) exited with code 012]
```

20. พิมพ์คำสั่งต่อไปนี้เพื่อออกจากโปรแกรม GDB

```
(gdb) q
```

## H.2 การใช้งานสแต็กพอยน์เตอร์ (Stack Pointer)

ตำแหน่งของหน่วยความจำบริเวณที่เรียกว่า **สแต็กเซกเม้นต์** (Stack Segment) จากรูปที่ 3.16 สแต็กเซกเม้นต์ตั้งในบริเวณแอดเดรสสูง (High Address) หน้าที่เก็บค่าข้อมูลของตัวแปรชนิดโลคอล (Local Variable) รับค่าพารามิเตอร์ระหว่างฟังก์ชัน กรณีที่มีจำนวนเกิน 4 ตัว พักเก็บค่าของรีจิสเตอร์ที่สำคัญ เช่น LR เป็นต้น

**สแต็กพอยน์เตอร์** คือ รีจิสเตอร์ R13 มีหน้าที่เก็บแอดเดรสตำแหน่งบนสุดของสแต็ก (Top of Stack: TOS) ซึ่งจะเป็นตำแหน่งที่เกิดการ PUSH และ POP ข้อมูลเข้าและออกจากสแต็กตามลำดับ โปรแกรมเมอร์สามารถจินตนาการได้ว่า **สแต็ก** คือ กองสิ่งของที่วางซ้อนกันโดยโปรแกรมเมอร์ และสามารถหยิบสิ่งของออก (POP) หรือวาง (PUSH) ของที่ซับบนสุดเท่านั้น โดยเราเรียกกองสิ่งของ (ตัวแปรโลคอลและอื่นๆ) นี้ว่า **สแต็กเฟรม** ซึ่งได้อธิบายในหัวข้อที่ 3.3.3 เราสามารถทำความเข้าใจการทำงานของสแต็กแบบง่ายๆ ได้ดังนี้

สแต็กพอยน์เตอร์ คือ หมายเลขอันสิ่งของซึ่งตำแหน่งจะลดลง/เพิ่มขึ้น เมื่อโปรแกรมเมอร์ใช้คำสั่ง PUSH/POP ตามลำดับ ซึ่งมีรายละเอียดเพิ่มเติมในหัวข้อที่ 4.5 ทั้งนี้เราสามารถอ้างอิงจากเวอร์ชัลเมโมรีของระบบลินก์ ในรูปที่ 3.16 และรูปที่ 5.2 ประกอบ

คำสั่ง STM (Store Multiple) ทำหน้าที่ PUSH ข้อมูลหรือค่าของรีจิสเตอร์จำนวนหนึ่งลงบนสแต็ก ณ ตำแหน่ง TOS คำสั่ง LDM (Load Multiple) ทำหน้าที่ POP ข้อมูลออกจากสแต็ก ณ ตำแหน่ง TOS มาเก็บในรีจิสเตอร์จำนวนหนึ่ง การเปลี่ยนแปลงตำแหน่งของ TOS เป็นไปได้สองทิศทาง คือ เพิ่มขึ้น (Ascending)/ลดลง (Descending). ดังนั้น คำสั่ง STM/LDM สามารถสมกับทิศทางและลำดับการกระทำ คือ ก่อน (Before) /หลัง (After) รวมเป็น 8 แบบ ดังนี้

- LDMIA/STMIA : IA ย่อจาก Increment After
- LDMIB/STMIB : IB ย่อจาก Increment Before
- LDMDA/STMDA : DA ย่อจาก Decrement After
- LDMDB/STMDB : DB ย่อจาก Decrement Before

คำ **Increment/Decrement** หมายถึง การเพิ่ม/ลดค่าของรีจิสเตอร์ที่เกี่ยวข้องโดยมักใช้งานร่วมกับ รีจิสเตอร์ SP คำ **after/before** หมายถึง ก่อน/หลังการปฏิบัติ (Execute) ตามคำสั่งนั้น ยกตัวอย่าง การใช้งานคำสั่งเพื่อ PUSH รีจิสเตอร์ลงในสแต็กโดยใช้ STMDB และ POP ค่าจากสแต็กจะคู่กับคำสั่ง LDMIA ความหมาย คือ สแต็กจะเติบโตในทิศทางที่แอดเดรสลดลง (Decrement Before) ซึ่งเป็นที่นิยมและตรงกับรูปการจัดวางเวอร์ชัลเม莫รีหรือหน่วยความจำเสมือนในรูปที่ 3.16 ผู้อ่านสามารถทบทวนเรื่องนี้ในหัวข้อที่ 5.2

1. สร้างไฟล์ Lab8\_2.s ตามโค้ดต่อไปนี้ ผู้อ่านสามารถข้ามประযุคคอมเมนต์ได้ เมื่อทำความเข้าใจแต่ละคำสั่งแล้ว

```
.global main
main:
    MOV R1, #1
    MOV R2, #2
    @ Push (store) R1 onto stack, then subtract SP by 4 bytes
    ! (Write-Back symbol) updates the register SP
```

```

STR R1, [sp, #-4]!
STR R2, [sp, #-4]!

@ Pop (load) the value and add 4 to SP
LDR R0, [sp], #+4
LDR R0, [sp], #+4

end:
BX LR

```

2. รันโปรแกรม บันทึกและอธิบายผลลัพธ์

3. สร้างไฟล์ Lab8\_3.s ตามโค้ดต่อไปนี้ ผู้อ่านสามารถข้ามประযุคคอมเมนต์ได้ เมื่อทำการเข้าใจแต่ละคำสั่งแล้ว

```

.global main
main:
    MOV R1, #0
    MOV R2, #1
    MOV R4, #2
    MOV R5, #3

    @ SP is subtracted by 8 bytes to save R4 and R5, respectively.
    @ The ! (Write-Back symbol) updates SP.
    STMDB SP!, {R4, R5}

    @ Pop (load) the values and increment SP after that
    LDMIA SP!, {R1, R2}
    ADD R0, R1, #0
    ADD R0, R0, R2

end:
BX LR

```

4. รันโปรแกรม บันทึกและอธิบายผลลัพธ์

5. ค้นคว้าการประยุกต์ใช้งานคำสั่ง STM/LDM สำหรับการทำงานของระบบปฏิบัติการ

### H.3 การพัฒนาโปรแกรมภาษาแอสเซมบลีร่วมกับภาษา C

การพัฒนาโปรแกรมด้วยภาษา C สามารถเขียนต่อ กับ ชาร์ดแวร์ และทำงานได้รวดเร็ว ใกล้เคียงกับภาษาแอสเซมบลี แต่การเสริมการทำงานของโปรแกรมภาษา C ด้วยภาษาแอสเซมบลียังมีความจำเป็น โดยเฉพาะโปรแกรมที่เรียกว่า **ดิไวซ์ไดรเวอร์** (Device Driver) ซึ่งเป็นโปรแกรมขนาดเล็กที่เขียนต่อ กับ ชาร์ดแวร์ที่ต้องการความรวดเร็วและประสิทธิภาพสูง การทดลองนี้จะแสดงให้ผู้อ่านเห็นการเขียนต่อฟังก์ชันภาษาแอสเซมบลีกับภาษา C อย่างง่าย

1. เปิดโปรแกรม CodeBlocks

2. สร้างโปรเจคท์ Lab8\_4 ภายใต้ไดเรกทอรี /home/pi/asm/Lab8

3. สร้างไฟล์ชื่อ add\_s.s และป้อนคำสั่งต่อไปนี้

```
.global add_s
add_s:
ADD R0, R0, R1
BX LR
```

4. เพิ่มไฟล์ add\_s.s ในโปรเจคท์ Lab8\_4 ที่สร้างไว้ก่อนหน้า

5. สร้างไฟล์ชื่อ main.c และป้อนคำสั่งต่อไปนี้

```
#include <stdio.h>
int main(){
    int a = 16;
    int b = 4;
    int i = add_s(a, b);
    printf("%d + %d = %d \n", a, b, i);
    return 0;
}
```

6. ทำการ Build และแก้ไขหากมีข้อผิดพลาดจนสำเร็จ

7. Run และสังเกตการเปลี่ยนแปลง

8. อธิบายว่าเหตุใดการทำงานจึงถูกต้อง ฟังก์ชัน add\_s รับข้อมูลทางรีจิสเตอร์ตัวไหนบ้างและรีเทิร์นค่าที่คำนวนเสร็จแล้วทางรีจิสเตอร์อะไร

9. อธิบายว่าเหตุใดฟังก์ชัน add\_s จึงไม่ต้องแบกอัปค่าของรีจิสเตอร์ LR

ในทางปฏิบัติ การบวกเลขในภาษา C สามารถทำได้โดยใช้เครื่องหมาย + โดยตรง และทำงานได้รวดเร็ว กว่า การทดลองตัวอย่างนี้เป็นการนำเสนอว่าผู้อ่านสามารถเขียนโปรแกรมอย่างไรที่จะบรรลุวัตถุประสงค์ เท่านั้น ฟังก์ชันภาษาแอสเซมบลีที่จะลิงก์เข้ากับโปรแกรมหลักที่เป็นภาษา C ควรจะมีอรรถประยุก্ত์มากกว่านี้ และเชื่อมโยงกับชาร์ดแวร์โดยตรงได้ดีกว่าคำสั่งในภาษา C เช่น ดิไวซ์ไดรเวอร์

## H.4 กิจกรรมท้ายการทดลอง

1. จดบันทึกโปรแกรม Lab8\_1 ด้วย GDB พร้อมกันจำนวน 2 Terminal เพื่อแสดงค่าของรีจิสเตอร์ PC ที่รันคำสั่งแรกของโปรแกรม Lab8\_1 ในทั้งสองหน้าต่าง และเปรียบเทียบค่า PC ว่าเท่ากันหรือแตกต่างกันหรือไม่ เพราะเหตุใด
2. หากค่าของรีจิสเตอร์ PC ทั้งสองค่าในข้อ 1 ตรงกัน จะใช้ความรู้เรื่องเวอร์ชวลเมโมรีหรือหน่วยความจำเสมือนในหัวข้อ [5.2](#) เพื่อตอบคำถาม
3. จดบันทึกโปรแกรม GDB เพื่อแสดงรายละเอียดของสแต็กระหว่างที่รันโปรแกรม Lab8\_2 และบอกลำดับการ PUSH และการ POP ที่เกิดขึ้นภายในโปรแกรมจากแต่ละคำสั่ง
4. จดบันทึกโปรแกรม GDB เพื่อแสดงรายละเอียดของสแต็กระหว่างที่รันโปรแกรม Lab8\_3 และบอกลำดับการ PUSH และการ POP ที่เกิดขึ้นภายในโปรแกรมจากแต่ละคำสั่ง
5. จงนำโปรแกรมภาษาแอสเซมบลีสำหรับคำนวนค่า mod ใน การทดลองที่ 7 มาเรียกใช้ผ่านโปรแกรมภาษา C
6. จงนำโปรแกรมภาษาแอสเซมบลีสำหรับคำนวนค่า GCD ใน การทดลองที่ 7 มาเรียกใช้ผ่านโปรแกรมภาษา C
7. จดบันทึกโปรแกรมภาษา C บนโปรแกรม Codeblocks ที่พัฒนาในข้อ 2 และ 3 เพื่อบันทึกการเปลี่ยนแปลงของ PC ก่อน ระหว่าง และหลังเรียกใช้ฟังก์ชันภาษา Assembly ว่าเปลี่ยนแปลงอย่างไร และตรงกับทฤษฎีที่เรียนหรือไม่ อย่างไร
8. เครื่องหมาย -g ใน **makefile** ต่อไปนี้

```
debug: Lab8_1
    as -g -o Lab8_1.o Lab8_1.s
```

มีความหมายอย่างไร

# ภาคผนวก I

## การทดลองที่ 9 การศึกษาและปรับแก้อินพุตและเอาต์พุตต่างๆ

การทดลองในภาคผนวกนี้จะช่วยอธิบายเนื้อหาในบทที่ 6 ซึ่งเกี่ยวข้องกับอุปกรณ์อินพุต/เอาต์พุตที่หลากหลายบนเครื่องคอมพิวเตอร์ตั้งโต๊ะ โดยมีวัตถุประสงค์เหล่านี้

- เพื่อให้เข้าใจการปรับแก้อุปกรณ์อินพุตและเอาต์พุตชนิดต่างๆ บนระบบปฏิบัติการ Raspberry Pi OS
- เพื่อให้เข้าใจความแตกต่างระหว่างอุปกรณ์อินพุตและเอาต์พุตชนิดต่างๆ บนบอร์ด Pi
- เพื่อให้สามารถอ่านข้อมูลความแสดงรายละเอียดของอุปกรณ์อินพุตและเอาต์พุตชนิดต่างๆ

หลักการและพื้นฐานความเข้าใจจะช่วยแนะนำทางให้ผู้อ่านสามารถศึกษาค้นคว้า อินพุต/เอาต์พุตอื่นๆ ในชิปและบอร์ดได้เพิ่มเติม รวมไปถึงบนโทรศัพท์เคลื่อนที่ แท็บเล็ตคอมพิวเตอร์ และอุปกรณ์อินเทอร์เน็ตสิ่ง (Internet of Things)

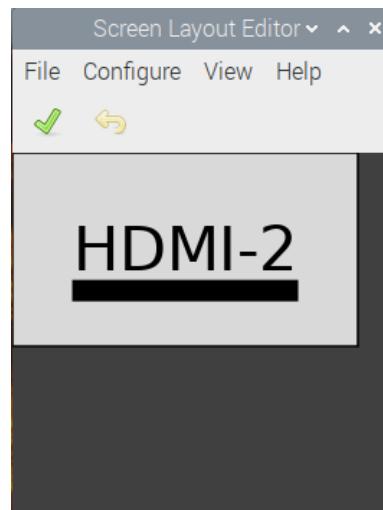
### I.1 จอแสดงผลผ่านพอร์ต HDMI

#### I.1.1 การปรับแก้ความละเอียดของจอแสดงผล

เมื่อขนาดหน่วยความจำสำหรับการใช้งานและแสดงผลของจีพียูมีปริมาณเพียงพอ ซึ่งตั้งอยู่บริเวณซี่อ่าว VCSRAM ในพื้นที่หน่วยความจำภายใน (ARM Physical Memory) ของรูปที่ 6.16 ผู้ใช้สามารถปรับเพิ่มหรือลดความละเอียดของจอแสดงผลได้โดยกดปุ่มบนเมนูดังนี้

menu->Preferences->Screen Configuration

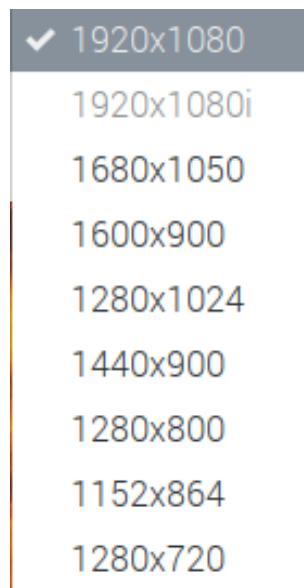
นี่เป็นการเรียกใช้โปรแกรม Screen Layout Editor



รูปที่ I.1: หน้าต่าง Screen Layout Editor สำหรับกำหนดค่าต่างๆ กับพอร์ตแสดงผล HDMI

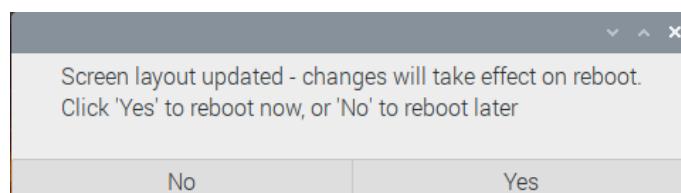
กดปุ่มบนเมนูบันปอограмเพื่อปรับความละเอียดของพอร์ตที่จะแสดงผลเข้ามต่ออยู่และรองรับ

Configure->Screens->HDMI-1 (HDMI-2)->Resolution



รูปที่ I.2: หน้าต่าง Set Resolution สำหรับกำหนดความละเอียดหน้าจอแสดงผลที่ต้องการ

กดปุ่มเลือกความละเอียดหน้าจอที่เหมาะสมกับจอที่เชื่อมต่ออยู่ คือ ไม่เกินความละเอียด 1920x1080 หลังจากนั้นกดปุ่ม เครื่องหมายถูก ในหน้าต่างหลักของ Scrren Layout Editor เพื่อยืนยัน หน้าต่างต่อไปนี้จะปรากฏขึ้น



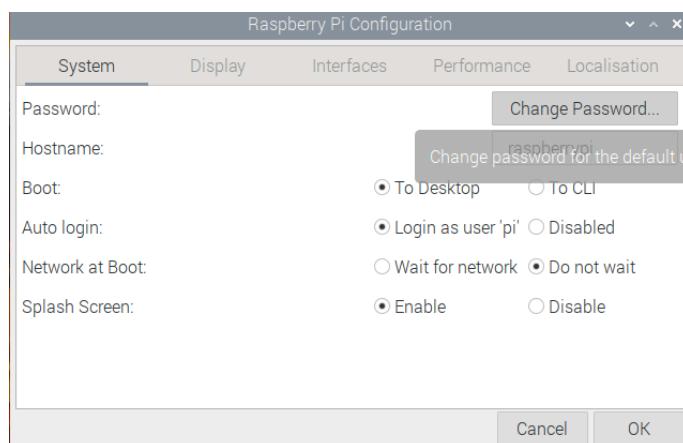
รูปที่ I.3: หน้าต่าง Reboot needed กดปุ่ม Yes เมื่อต้องการรีบูต ณ เวลาใดก็ได้

กด Yes เพื่อรีบูตระบบใหม่ ผู้อ่านสามารถค้นคว้าเรื่องสายมาตรฐาน HDMI ในหัวข้อที่ 6.1 เพิ่มเติมก่อนค้นคว้าเพิ่มเติมในอินเทอร์เน็ต

### I.1.2 การปรับแก้ขนาดหน่วยความจำของ GPU

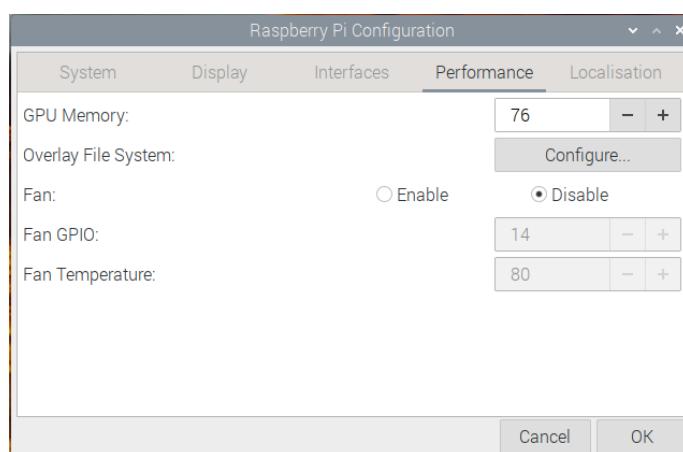
หน่วยความจำสำหรับจัดแสดงผลหรือจีพีयู (Graphic Processing Unit) ถูกแบ่งพื้นที่ออกจากหน่วยความจำ SDRAM บนบอร์ด เพื่อใช้งานร่วมกันทำให้ประหยัดต้นทุน แต่มีข้อเสียในด้านประสิทธิภาพจะลดลง เมื่อผู้ใช้งานต้องการภาพที่มีอัตราเฟรมเรท (Frame Rate) สูง เช่น ภาวดีโอเคลื่อนไหว เกม 3 มิติ ความละเอียดของจอแสดงผลขึ้นตรงกับขนาดของหน่วยความจำของจีพียู ผู้อ่านสามารถปรับแก้ขนาดหน่วยความจำของจีพียูได้ดังนี้

menu->Preferences->Raspberry Pi Configuration



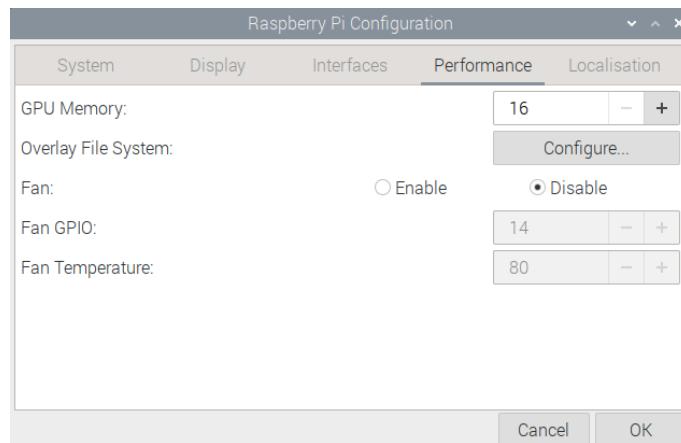
รูปที่ I.4: หน้าต่าง Raspberry Pi Configuration

กดแท็บ Performance เพื่อปรับค่าต่างๆ เกี่ยวกับจีพียู ผู้อ่านสามารถปรับลด (-) หรือเพิ่ม (+) ขนาดหน่วยความจำซึ่งเท่ากับ 76 เมบิไบต์ในรูป



รูปที่ I.5: แท็บ Performance หน้าต่าง Raspberry Pi Configuration

โดยหน้าต่างที่ปรากฏขึ้นมีลักษณะดังนี้ ผู้ใช้สามารถกำหนดขนาดที่ต้องการโดยขั้นต่ำคือ 16 เมบิไบต์ (MiB)



รูปที่ I.6: หน้าต่างกำหนดขนาดหน่วยความจำสำหรับจีพียูที่ 16 MiB

## I.2 ระบบเสียงดิจิทัล

อุปกรณ์ระบบเสียงดิจิทัลที่ติดตั้งมาบนบอร์ด Pi จากโรงงาน ผู้ใช้สามารถเพิ่มเติมได้ผ่านพอร์ต USB และปรับแต่งระดับเสียงได้เข่นกัน

### I.2.1 การเลือกช่องสัญญาณเสียงเชื่อมต่อกับลำโพง

ผู้อ่านสามารถเชื่อมต่อสัญญาณเสียงกับลำโพงภายนอกผ่านช่องแจ็ค 3.5 มม. หรือ ลำโพงของจอทีวี LCD ผ่านช่องสัญญาณ HDMI จากการทดลองต่อไปนี้

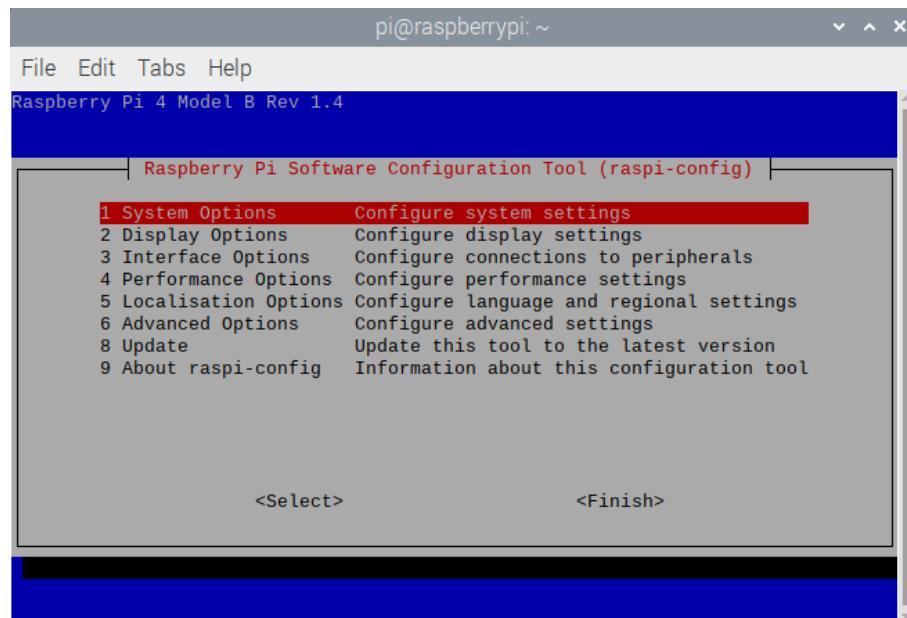
1. ใช้คำสั่งต่อไปนี้ในโปรแกรม Terminal

```
$ sudo raspi-config
```

จะบอกเหตุผลว่าคำสั่ง sudo ที่นำหน้ามีความสำคัญอย่างไร

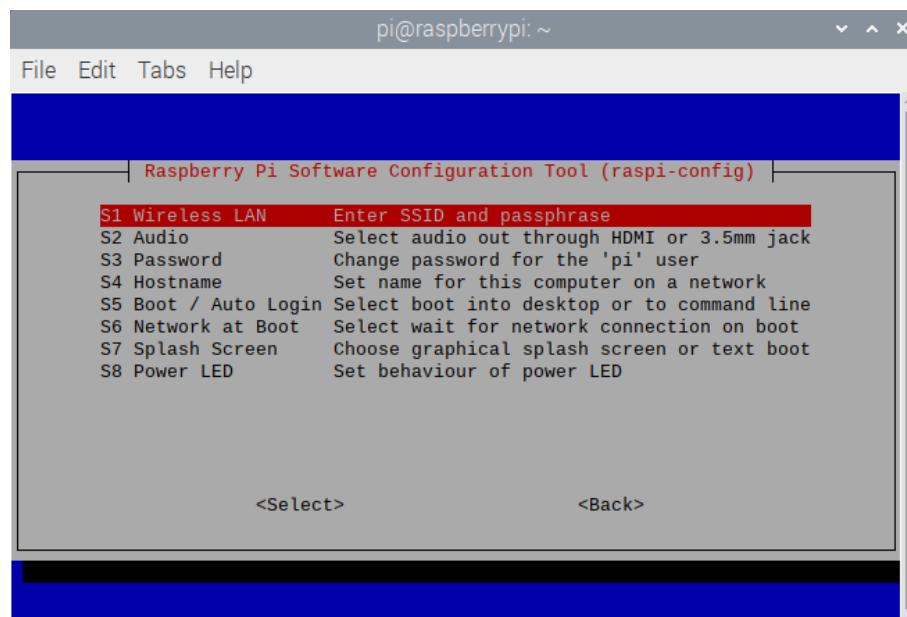
2. กดปุ่มลูกศรขึ้นลงเพื่อเลือกเมนู System Options ในรูป

```
$ sudo raspi-config
```



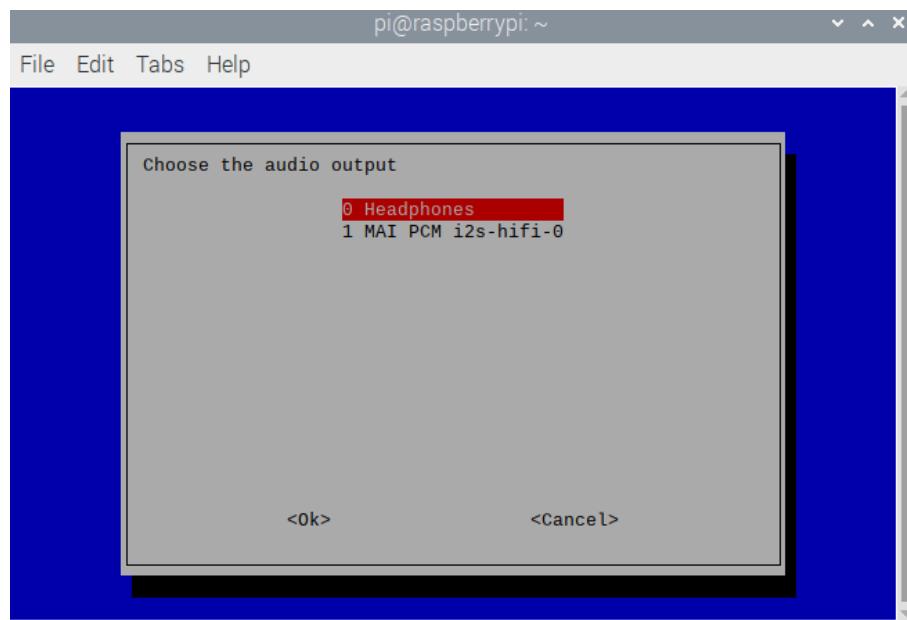
รูปที่ I.7: หน้าต่างโปรแกรม raspi-config สำหรับบอร์ด Pi

### 3. กดปุ่มลูกศรขึ้นลงเพื่อเลือกเมนู S2 Audio ในรูป



รูปที่ I.8: เมนู System Options ในหน้าต่างโปรแกรม raspi-config สำหรับบอร์ด Pi

### 4. กดปุ่มลูกศรขึ้นลงในรูปเพื่อเลือกเมนู 0 Headphones สำหรับแจ็ค 3.5 มม. หรือ 1 MAI PCM i2s-hifi-0 สำหรับพอร์ต HDMI

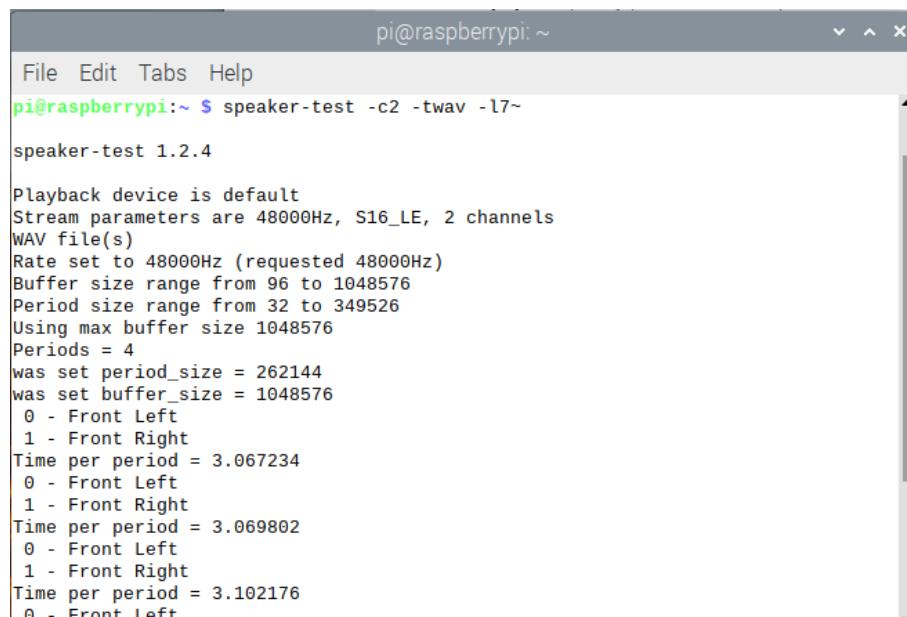


รูปที่ I.9: เมนู Audio ภายในเมนู System Options ในหน้าต่างโปรแกรม raspi-config สำหรับบอร์ด Pi

5. เมื่อเลือกเมนูที่ต้องการแล้ว กดปุ่ม Esc(ape) เพื่อถอยกลับออกจากเมนู และกดจนออกจากโปรแกรม
6. ใช้คำสั่งต่อไปนี้ในโปรแกรม Terminal เพื่อทดสอบสัญญาณเสียงกับลำโพงที่เลือกต่อ

```
$ speaker-test -c2 -twav -l7
```

หากสำเร็จ ผู้อ่านจะได้ยินเสียงและผลลัพธ์คล้ายรูปต่อไปนี้



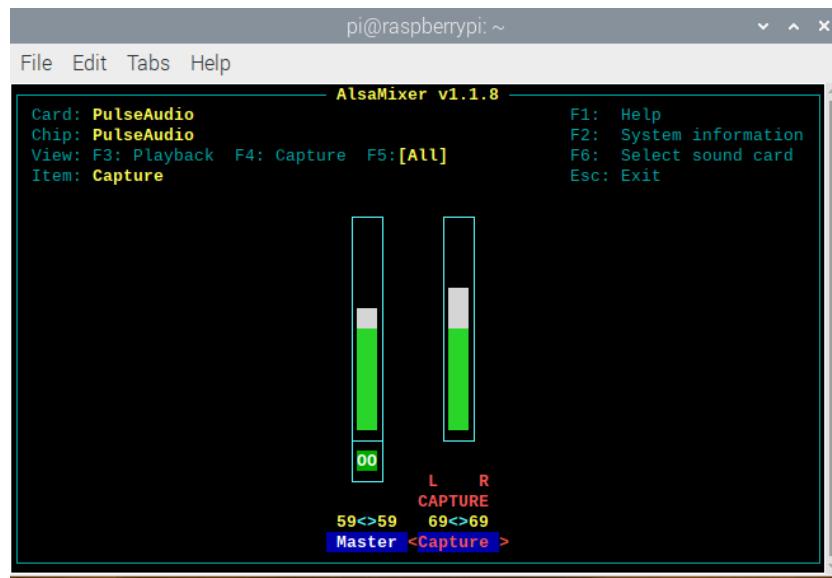
รูปที่ I.10: ผลลัพธ์ในหน้าต่างโปรแกรม speaker-test สำหรับบอร์ด Pi

### I.2.2 การควบคุมระดับเสียง

นอกเหนือจากการดับเสียงที่โอลองรูปลำโพงด้านข้างของจอ ผู้อ่านสามารถควบคุมระดับความดังของเสียงทั้งด้านอินพุต (Capture) และเอาต์พุต (Playback) โดยพิมพ์คำสั่งนี้

```
$ alsamixer
```

หน้าต่างโปรแกรม alsamixer จะปรากฏขึ้น ผู้อ่านสามารถกดปุ่มลูกศรขึ้น/ลง เพื่อเพิ่ม/ลด ระดับความดังของ Playback ด้วยปุ่ม F3 ของ Capture ด้วยปุ่ม F4 และแสดงผลทั้งสอง ด้วยปุ่ม F5



รูปที่ I.11: โปรแกรม ALSA Mixer สำหรับควบคุมระดับเสียงทั้งด้านอินพุต (Capture: F4) และเอาต์พุต (Playback: F3) บนบอร์ด Pi

### I.2.3 รายชื่ออุปกรณ์ในระบบเสียง

ระบบเสียงในระบบปฏิบัติการ Linux ควบคุมการทำงานของเสียงผ่านระบบ ALSA (Advanced Linux Sound Architecture) ซึ่งจัดเตรียมไดรเวอร์ (Device Driver) สำหรับเสียงให้กับเครื่องเนล และอุปกรณ์ที่เกี่ยวข้องกับเสียงผ่านพอร์ต HDMI ช่องเสียบหูฟัง (Headphone) พอร์ต USB เช่น ไมโครโฟน, หูฟังพร้อมไมโครโฟน, เว็บแคม เป็นต้น

สำหรับการควบคุมอุปกรณ์เสียงขั้นสูง ผู้อ่านสามารถแสดงรายชื่อไฟล์หรือไดเรกทอรีที่เกี่ยวข้องกับระบบเสียงดังนี้

```
$ ls -l /proc/asound
total 0
lrwxrwxrwx 1 root root 5 Jun 21 19:48 b1 -> card0
dr-xr-xr-x 4 root root 0 Jun 20 19:45 card0
dr-xr-xr-x 4 root root 0 Jun 20 19:45 card1
-r--r--r-- 1 root root 0 Jun 21 19:48 cards
-r--r--r-- 1 root root 0 Jun 21 19:48 devices
lrwxrwxrwx 1 root root 5 Jun 21 19:48 Headphones -> card1
```

```
-r--r--r-- 1 root root 0 Jun 21 19:48 modules
dr-xr-xr-x 4 root root 0 Jun 21 19:48 oss
-r--r--r-- 1 root root 0 Jun 21 19:48 pcm
dr-xr-xr-x 2 root root 0 Jun 21 19:48 seq
-r--r--r-- 1 root root 0 Jun 21 19:48 timers
-r--r--r-- 1 root root 0 Jun 21 19:48 version
```

ผลลัพธ์คือ รายชื่ออุปกรณ์ที่เกี่ยวข้องกับเสียง ซึ่งได้แสดงไปก่อนหน้านี้ ผู้อ่านจะสังเกตได้ว่าไดเรกทอรี /proc/asound/pcm จะเข้มโงยกับเนื้อหาในหัวข้อที่ 6.4 และเห็นว่ามีไดเรกทอรีชื่อ card0 อยู่สองตำแหน่งคือ ในแคลรอก และแคลร์ที่มีชื่อ b1 -> card0 สัญลักษณ์ -> เรียกว่า **ซิมบอลิกลิงก์** (Symbolic Link) หมายความว่า ไดเรกทอรีชื่อ b1 คือไดเรกทอรี card0 ส่วนแคลร์ที่มีชื่อ Headphones -> card1 สัญลักษณ์ -> เรียกว่า **ซิมบอลิกลิงก์** (Symbolic Link) หมายความว่า ไดเรกทอรีชื่อ Headphones คือไดเรกทอรี card1

1. ผู้อ่านสามารถค้นเพิ่มเติมโดยพิมพ์คำสั่งต่อไปนี้

```
$ cat /proc/asound/cards
```

บันทึกผลลัพธ์ในพื้นที่ว่างต่อไปนี้

2. ค้นคว้าว่า b1 และ Headphones คือ อุปกรณ์ใด ทั้งสองอุปกรณ์นี้แตกต่างกันหรือไม่

3. ค้นคว้าเพิ่มเติมเพื่อทำความหมายของ Symbolic Link และจดบันทึก

4. พิมพ์คำสั่งนี้ในโปรแกรม Terminal

```
$ cat /proc/asound/cards
```

โดยคำสั่ง cat ซึ่งได้อธิบายแล้วในการทดลองที่ 4 ภาคผนวก D สามารถอ่านไฟล์และแสดงข้อมูลภายในไฟล์ผ่านทางหน้าจอแสดงผล บันทึกในที่ว่างต่อไปนี้

อภิปรายผลที่ได้ ดังนี้ ผลลัพธ์ได้จากบอร์ด Pi4 ใช้ชิป BCM\_\_\_\_\_ แต่ยังใช้ไดเรเวอร์เสียงเดียวกันกับ BCM283 โดย หมายเลข 0 คือ หมายเลขของระบบเสียงที่ติดตั้งใช้งานเพียงระบบเดียว และตรงกับ อุปกรณ์ชื่อ \_\_\_\_\_ 0

## I.3 พортเชื่อมต่ออุปกรณ์ USB

### I.3.1 รายชื่ออุปกรณ์กับพอร์ต USB

1. ใน การทดลองนี้ ขอผู้อ่านให้ดึงหัวเชื่อมต่อ USB ของมาส์ที่ใช้อยู่ออก และพิมพ์คำสั่งนี้ในโปรแกรม Terminal

```
$ lsusb
```

เพื่อแสดงรายชื่ออุปกรณ์ USB ที่เชื่อมต่ออยู่ทั้งหมดในบอร์ด ดังตัวอย่างต่อไปนี้

```
Bus 002 Device 001: ID 1d6b:0003 Linux Foundation 3.0 root hub
Bus 001 Device 003: ID 046d:c534 Logitech, Inc. Unifying Receiver
Bus 001 Device 002: ID 2109:3431 VIA Labs, Inc. Hub
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
```

ผู้อ่านจะเห็นรายชื่ออุปกรณ์ที่เชื่อมต่อกับพอร์ต USB เรียงลำดับย้อนกลับ จาก Bus 001 และ Bus 002 แล้วจึงเรียงจาก Device 003 - Device 001 โดย

- Bus 002 Device 001 คือ วงจร Root Hub เป็นวงจรภายในชิป BCM2711 สำหรับเชื่อมต่อพอร์ต USB 3.0 เพิ่มเติม สังเกตได้จากพลาสติกสีฟ้า มีหมายเลข ID = 1d6b:0003
- Bus 001 Device 003 คือ ตัวรับส่งสัญญาณคีย์บอร์ดและเมาส์ไร้สายของผู้เขียน มีหมายเลข ID = 046d:c534 ผลิตโดย บริษัท Logitech
- Bus 001 Device 002 คือ วงจร USB Hub สำหรับเชื่อมต่อพอร์ต USB เพิ่มเติม มีหมายเลข ID = 2109:3431 ผลิตโดย บริษัท VIA Labs
- Bus 001 Device 001 คือ วงจร Root Hub เป็นวงจรภายในชิป BCM2711 สำหรับเชื่อมต่อพอร์ต USB 2.0 เพิ่มเติม สังเกตได้จากพลาสติกสีดำ มีหมายเลข ID = 1d6b:0002

2. บันทึกผลลัพธ์ของผู้อ่าน

Bus 00\_ Device 00\_ : ID = \_ \_ \_ \_ \_ : \_ \_ \_ \_

Bus 00\_ Device 00\_ : ID = \_ \_ \_ \_ \_ : \_ \_ \_ \_

Bus 00\_ Device 00\_ : ID = \_ \_ \_ \_ \_ : \_ \_ \_ \_

Bus 00\_ Device 00\_ : ID = \_ \_ \_ \_ \_ : \_ \_ \_ \_

Bus 00\_ Device 00\_ : ID = \_ \_ \_ \_ \_ : \_ \_ \_ \_

3. ผู้อ่านย้ายคีย์บอร์ดหรือมาส์จากพอร์ต USB 2.0 ไปยัง USB 3.0 แล้วแสดงรายชื่ออุปกรณ์ USB ด้วยคำสั่ง

\$ lsusb

เช่นเดิม บันทึกเฉพาะผลที่เปลี่ยนแปลง

Bus 00\_ Device 00\_ : ID = \_ \_ \_ \_ \_ : \_ \_ \_ \_

### I.3.2 รายละเอียดการเชื่อมต่ออุปกรณ์กับพอร์ต USB

คำสั่งต่อไป คือ **dmesg** สามารถแสดงรายการการทำงาน หรือ Log ของระบบปฏิบัติการว่าตั้งแต่เริ่มเปิดเครื่อง โดยคำว่า **dmesg** ย่อมาจากคำสั่ง “display message or display driver” ซึ่งเครื่องเนลได้บันทึกไว้ในบัฟเฟอร์ชนิดวงแหวน (Ring Buffer) ซึ่งข้อมูลจะถูกเขียนทับเมื่อบัฟเฟอร์เต็ม

1. รันคำสั่งนี้ และเลื่อนหน้าต่างขึ้นไปที่ตำแหน่งวินาทีที่ 0.000000

\$ dmesg

2. จงเปรียบเทียบข้อมูลที่ผู้อ่านได้กับข้อมูลต่อไปนี้ ระบุตำแหน่งที่แตกต่างและเขียนข้อมูลนั้นลงในผลการทดลอง

```
[    0.000000] Booting Linux on physical CPU 0x0000000000 [0x410fd083]
[    0.000000] Linux version 5.10.63-v8+ (dom@buildbot)
(aarch64-linux-gnu-gcc-8 (Ubuntu/Linaro 8.4.0-3ubuntu1) 8.4.0,
GNU ld (GNU Binutils for Ubuntu) 2.34) #1459 SMP PREEMPT
Wed Oct 6 16:42:49 BST 2021
[    0.000000] random: fast init done
[    0.000000] Machine model: Raspberry Pi 4 Model B Rev 1.4
[    0.000000] efi: UEFI not found.
```

```
[    0.000000] Reserved memory: created CMA memory pool at
0x000000001ac00000, size 320 MiB
[    0.000000] OF: reserved mem: initialized node linux,cma,
compatible id shared-dma-pool
...
```

ใช้ผลการทดลองของผู้อ่านเอง เติมรายละเอียดใน \_\_\_\_\_ ที่เว้นว่างไว้ ซึ่งเรียงลำดับตามเหตุการณ์ที่ได้จากคำสั่ง \$ dmesg สัญลักษณ์ [xxxx.yyyyyy] แสดงลำดับที่เกิดขึ้นตามเวลา โดย xxxx คือเลขวินาทีตั้งแต่เครื่องเนลเริ่มทำงาน และ yyyyyy คือเศษวินาที ข้อมูลที่แสดงเป็น 0.000000 เนื่องจากเครื่องเนลอยู่ระหว่างการเริ่มต้น

- เริ่มต้นการบูตระบบปฏิบัติการด้วยแกนประมวลผลหมายเลข \_\_\_\_\_
- จดบันทึกหมายเลขเวอร์ชันของลินุกซ์ของผู้อ่านโดยละเอียด 5.\_\_\_\_\_.\_\_\_\_\_
- จดบันทึกคำสั่งภาษาแอสเซมบลีเวอร์ชัน \_\_\_\_\_ บิต
- แสดงผลการตรวจจับว่าเป็นบอร์ด Raspberry Pi \_\_\_\_\_ Model \_\_\_\_\_ Rev \_\_\_\_\_
- cma ย่อ มา จาก \_\_\_\_\_ สำหรับ ขบวนการ DMA เริ่ม ต้น ที่ แอดเดรส 0x\_\_\_\_\_ ขนาด \_\_\_\_\_ เมบิไบต์
- ...

ในการทดลองนี้ ระบบสามารถตรวจจับอุปกรณ์ USB และติดตั้งไดรเวอร์ได้อย่างถูกต้องปราศจากข้อผิดพลาด

1. ผู้อ่านสามารถล้างบัฟเฟอร์โดยใช้คำสั่ง ต่อไปนี้

```
$ sudo dmesg -C
```

โดย -C คือ Clear เป็นคำสั่งเพิ่มเติมให้ dmesg ล้างข้อมูลในบัฟเฟอร์ออก โปรดสังเกต ตัว C พิมพ์ใหญ่ หลังจากนั้น ผู้อ่านทดสอบโดยการกดเม้าส์ออก และเสียบกลับเข้าไปใหม่

2. ผู้อ่านจะต้องกดและเสียบเม้าส์กลับเข้าไปใหม่อีกรอบ
3. ผู้อ่านสามารถแสดงข้อมูลที่เพิ่มเข้ามาในบัฟเฟอร์ได้อีก โดยเรียกคำสั่ง

```
$ dmesg
```

4. จดบันทึกเฉพาะ 4 บรรทัดแรก

## 5. อภิรายผลลัพธ์ที่บันทึกได้ในพื้นที่ว่างต่อไปนี้

ในการเชื่อมต่อพอร์ต USB หากระบบแจ้งข้ออุปกรณ์ไม่มีข้อมูลใดอยู่ในบันทึก แสดงว่าอุปกรณ์ขาดซอฟต์แวร์ซึ่งทำหน้าที่เป็นดิไวซ์ไดเรเวอร์ ขอให้ผู้อ่านค้นหาจากหมายเลขประจำตัวของผู้ผลิต (idVendor) หากผู้ผลิตมีได้เปิดเผยซอฟต์แวร์ ผู้อ่านจำเป็นต้องดาวน์โหลดหรือคอมไพล์เองจากนักพัฒนารายอื่นแทน

## I.4 พอร์ตเชื่อมต่อเครือข่าย WiFi และ Ethernet

### I.4.1 รายชื่ออุปกรณ์เครือข่าย

- ผู้อ่านสามารถตรวจสอบรายชื่ออุปกรณ์สำหรับเชื่อมต่อเครือข่ายได้จากคำสั่ง `ifconfig` ทางโปรแกรม Terminal ตัวอย่างผลลัพธ์เป็นดังนี้

```
$ ifconfig
```

- เติมข้อมูลหรือตัวเลขในช่องว่าง \_\_\_\_\_ ที่เตรียมไว้ให้จากผลลัพธ์ที่ได้ต่อไปนี้ ซึ่งสำคัญการอาจแตกต่างกัน

```
eth0: flags=____<UP, BROADCAST, RUNNING, MULTICAST> mtu _____
      inet _____._____._____._____
        netmask _____._____._____._____
        broadcast _____._____._____._____
      ...
lo: flags=____<UP, LOOPBACK, RUNNING> mtu _____
      inet _____._____._____._____
        netmask _____._____._____._____
      inet6 ::1 prefixlen 128 scopeid 0x10<host>
        loop txqueuelen 1000 (Local Loopback) ...

wlan0: flags=____<UP, BROADCAST, MULTICAST> mtu _____
      inet _____._____._____._____
        netmask _____._____._____._____
        broadcast _____._____._____._____
      ...
%       ether ____:_:_:_:_:_
```

3. โปรดสังเกตคำเริ่มต้นในแต่ละรายการ ค้นคว้า และกรอกรายละเอียดเพิ่มเติม ดังนี้

- eth0 หมายถึง

- lo หมายถึง

- wlan0 หมายถึง

ผู้อ่านสามารถค้นคว้าเพิ่มเติมได้ที่ <https://www.tecmint.com/ifconfig-command-examples/>

#### I.4.2 การเปิด/ปิดอุปกรณ์เครือข่าย

1. ผู้อ่านสามารถเปิดอุปกรณ์ eth0 ได้ตามต้องการแล้วทำการตรวจสอบ ดังนี้

```
$ sudo ifconfig eth0 down
$ ifconfig
```

จดว่าข้อความใดที่บ่งบอกว่า eth0 ไม่ทำงานแล้ว

2. ผู้อ่านสามารถเปิดอุปกรณ์ eth0 ได้ตามต้องการแล้วทำการตรวจสอบ ดังนี้

```
$ sudo ifconfig eth0 up
$ ifconfig
```

จดว่าข้อความใดที่บ่งบอกว่า eth0 ทำงานแล้ว

3. ผู้อ่านสามารถใช้คำสั่ง ifconfig สำหรับปิด อุปกรณ์ wlan0 ดังนี้

```
$ sudo ifconfig wlan0 down
$ ifconfig
```

4. ผู้อ่านสามารถใช้คำสั่ง ifconfig สำหรับเปิด อุปกรณ์ wlan0 ดังนี้

```
$ sudo ifconfig wlan0 up
$ ifconfig
```

จดว่าข้อความใดที่บ่งบอกว่า wlan0 ทำงานแล้ว

5. นอกเหนือจากการเปิดปิดอุปกรณ์เครือข่าย ผู้อ่านสามารถตรวจสอบรายชื่อเครือข่าย WiFi ที่บอร์ดเคยเชื่อมต่อสำเร็จได้จากไฟล์ wpa\_supplicant.conf ซึ่งจะบันทึกรายละเอียดต่างๆ ของการเชื่อมต่อนั้นๆ รวมถึงพาสเวิร์ด (password) โดยพิมพ์คำสั่งต่อไปนี้ในโปรแกรม Terminal

```
$ cat /etc/wpa_supplicant/wpa_supplicant.conf
```

บันทึกผลที่ได้โดยกรอกในช่อง \_ เท่านั้น

```
network={  
    ssid="_____"  
    psk="*****"  
    key_mgmt=_____  
}
```

- ssid หมายถึง
- ssid ย่อมาจาก
- psk ย่อมาจาก
- key\_mgmt คือ

ผู้อ่านสามารถค้นคว้าเพิ่มเติมได้ที่ [https://wiki.archlinux.org/title/wpa\\_supplicant](https://wiki.archlinux.org/title/wpa_supplicant)

#### I.4.3 การตรวจสอบการเชื่อมต่อกับเครือข่ายเบื้องต้น

เมื่อผู้อ่านเปิดและทำการเชื่อมต่อสำเร็จ แล้วจึงสามารถตรวจสอบการเชื่อมต่อในระดับชั้นเครือข่าย โดยใช้คำสั่ง ping ใน Terminal ดังนี้

```
$ ping <ip address or host name>
```

การตรวจสอบการเชื่อมต่อเบื้องต้น คือ การ ping ไปหาเราเตอร์ผ่านต้นทางที่บอร์ดเชื่อมต่อ ผู้อ่านสามารถสืบค้นหมายเลขไอพีของเราเตอร์ที่ต้นทาง โดยสังเกตที่ inet ของ eth0 หรือ wlan0 ว่าเริ่มต้นด้วยหมายเลข 192.168.x.y ซึ่งเราเตอร์ต้นทางมักจะมีหมายเลข 192.168.x.1 หรือ 192.168.x.254

นี่เป็นตัวอย่างผลลัพธ์ที่ได้ของคำสั่ง ping 192.168.1.1 ที่ผู้อ่านจะต้องเติมหมายเลขใน \_ ที่เตรียมไว้ให้

PING 192.168.\_.1 (192.168.\_.1) 56(84) bytes of data.

64 bytes from 192.168.\_.1: icmp\_seq=\_ ttl=\_ time=\_.\_ ms

64 bytes from 192.168.\_.1: icmp\_seq=\_ ttl=\_ time=\_.\_ ms

64 bytes from 192.168.\_.1: icmp\_seq=\_ ttl=\_ time=\_.\_ ms

โดย 192.168.\_.1 คือหมายเลขไอพีแอดเดรสของอุปกรณ์ที่คำสั่งจะส่งแพ็กเก็ต ICMP (Internet Control Message Protocol) ความยาว 64 ไบต์ไป และรออุปกรณ์หมายเลขนี้ตอบกลับมาอย่างบอร์ด Pi โดยจับเวลาตั้งแต่ส่งไปและรอตอบกลับมา ของแพ็กเก็ตลำดับที่ \_ (icmp\_seq=\_) เป็นระยะเวลา \_.\_ มิลลิวินาที ส่วน ttl=\_ ย่อมาจากคำว่า time to live หมายถึง เลขจำนวนเต็มที่ผู้ส่งกำหนดค่าอายุของแพ็กเก็ตที่สามารถเดินทางผ่านเครือข่าย หากตั้งไว้น้อยจะทำให้แพ็กเก็ตข้อมูลนี้อายุสั้นและอาจเดินทางไปไม่ถึงปลายทางเนื่องจากหมดอายุก่อน โดย ttl=64 เป็นค่าปกติ

ผู้อ่านจะสังเกตเห็นว่า ระยะเวลาไม่ค่าตั้งแต่ \_\_\_\_ - \_\_\_\_ มิลลิวินาที ขึ้นอยู่กับคุณภาพ ของสาย Ethernet หรือความแรงของสัญญาณ WiFi คุณภาพดีจะทำให้ระยะเวลาสั้นกว่า หลังจากตรวจสอบว่าบอร์ดสามารถเชื่อมต่อกับเราเตอร์ต้นทางได้ตามตัวอย่างก่อนหน้า ผู้อ่านสามารถใช้ตรวจสอบการเชื่อมต่อได้ว่า เราเตอร์ต้นทางสามารถเชื่อมต่อกับเครือข่ายอินเทอร์เน็ตได้สำเร็จหรือไม่ โดย Host name คือ ชื่อเซิร์ฟเวอร์ปลายทางที่จะทะเบียนโดเมนเนม (Domain Name) เรียบร้อยแล้ว เช่น ping www.google.com

#### I.5 กิจกรรมท้ายการทดลอง

1. จงค้นหาว่าความละเอียดของการแสดงผลผ่านพอร์ต HDMI ในหัวข้อที่ [1.1.1](#) เก็บบันทึกลงในไฟล์ชื่ออะไร
  2. ใช้คำสั่ง ifconfig ปิดอุปกรณ์ 10 แล้วใช้คำสั่ง ping 127.0.0.1 ว่ามีการตอบสนองกลับมาหรือไม่ เปิดอุปกรณ์ 10 แล้ว ping อีกรอบ จนอธิบายว่า 127.0.0.1 คือ อะไร
  3. ใช้คำสั่ง ping เพื่อทดสอบเราเตอร์ที่อยู่ด้านหนทางของผู้อ่าน เช่น ping 192.168.x.1 หรือ 192.168.x.254 โดย x มีค่าเท่ากับ 0, 1, 2, ... จนกว่าจะมีการตอบสนองกลับมา
  4. ใช้คำสั่ง ping เพื่อตรวจสอบการเชื่อมต่อไปยัง [www.google.com](http://www.google.com)

## ภาคผนวก J

### การทดลองที่ 10 การเชื่อมต่อกับขา GPIO

การทดลองนี้คาดว่าผู้อ่านเคยศึกษาและทดลองเขียนหรือพัฒนาโปรแกรมด้วยภาษา C มาบ้างแล้ว และมีความคุ้นเคยกับ IDE (Integrated Development Environment) จากพัฒนาโปรแกรมและการดีบักโปรแกรมด้วยภาษา C/C++ และแอดเซมบลี ดังนั้น การทดลองมีวัตถุประสงค์เหล่านี้

- เพื่อปฏิบัติการเชื่อมต่อวงจรกับขา GPIO บนบอร์ด Pi ตามเนื้อหาในบทที่ 6 หัวข้อที่ 6.11
- เพื่อพัฒนาโปรแกรมภาษา C ควบคุมการทำงานของขา GPIO ด้วยไลบรารี wiringPi
- เพื่อพัฒนาโปรแกรมภาษาแอดเซมบลีควบคุมการทำงานของขา GPIO ด้วยไลบรารี wiringPi

โปรดสังเกตตัวอักษร W ที่คำว่า wiringPi ต้องเป็นตัวอักษรพิมพ์เล็ก

#### J.1 ไลบรารี wiringPi

ไลบรารี wiringPi รวบรวมฟังก์ชันที่พัฒนาด้วยภาษา C สำหรับบอร์ด Pi เป็น OpenSource ภายใต้ GNU LGPLv3 license สามารถเรียกใช้งานผ่าน ภาษา C and C++ รวมถึงภาษาแอดเซมบลี

เนื่องจากไลบรารี wiringPi เป็นซอฟต์แวร์แบบโอเพนซอร์สแจกให้แก่นักพัฒนาทั่วโลกผ่านทาง <https://github.com/WiringPi> และมีการปรับปรุงแก้ไขตลอดเวลาโดยทีมนักพัฒนา ดังนั้น ผู้อ่านควรต้องติดตั้งและปรับปรุงระบบปฏิบัติการให้ทันสมัยและติดตั้ง ตามขั้นตอนต่อไปนี้

- ผู้อ่านควรตรวจสอบว่าบอร์ดที่มีติดตั้งไลบรารี WiringPi แล้วหรือยัง โดยใช้คำสั่ง

```
$ gpio -v
```

- หากบอร์ดยังไม่ได้ติดตั้งผู้อ่านควรปรับปรุงระบบปฏิบัติการให้เป็นปัจจุบันก่อน โดยพิมพ์คำสั่นนี้บนโปรแกรม Terminal โดยใช้สิทธิ์ของ SuperUser:

```
$ sudo apt-get update  
$ sudo apt-get upgrade
```

ขั้นตอนนี้จะใช้เวลานานและความอดทน รวมถึงการเชื่อมต่อกับเครือข่ายอินเทอร์เน็ตที่มีเสถียรภาพ

3. ติดตั้งด้วยไลบรารี wiringPi โดยพิมพ์คำสั่งนี้บน Terminal โดยใช้สิทธิ์ของ SuperUser:

```
$ sudo apt-get install wiringpi
```

คำสั่งนี้จะติดตั้งไลบรารีลงบนการ์ดหน่วยความจำ SD ในบอร์ด

4. หากบอร์ดติดตั้งแล้ว คำสั่ง `upio -v` ได้ผลลัพธ์ของการเรียกดังนี้

```
$ gpio -v
gpio version: _._ _
Copyright (c) _ _ _ _ _ - _ _ _ _ _ Gordon Henderson
This is free software with ABSOLUTELY NO WARRANTY.
For details type: gpio -warranty
```

## Raspberry Pi Details:

```
Type: Pi _, Revision: _ _, Memory: _ _ _ MiB, Maker: Sony
* Device tree is enabled.
*--> Raspberry Pi _ Model B Rev _._
* This Raspberry Pi supports user-level GPIO access.
```

5. เรียกคำสั่ง `gpio readall` เพื่อตรวจสอบและบันทึกผลลัพธ์ที่แสดงบนหน้าต่าง Terminal ลงในตารางหน้าถัดไป

```
$ gpio readall
```

6. จงเติมหมายเลขในคอลัมน์ wPi (wiringPi) ให้ตรงกับขาเข้าอิมต่อ 40 ของบอร์ด Pi ตามที่แสดงบนหน้าจอลงในตารางต่อไปนี้ เพื่อใช้ประกอบการต่อวงจรที่ถูกต้อง

| Pi 3B |     |         |   |          |   |  |         |     |     |  |
|-------|-----|---------|---|----------|---|--|---------|-----|-----|--|
| BCM   | wPi | Name    | V | Physical | V |  | Name    | wPi | BCM |  |
|       |     | 3.3v    |   | 1    2   |   |  | 5v      |     |     |  |
| 2     | _   | SDA.1   | 1 | 3    4   |   |  | 5v      |     |     |  |
| 3     | _   | SCL.1   | 1 | 5    6   |   |  | 0v      |     |     |  |
| 4     | _   | GPIO. 7 | 1 | 7    8   | 0 |  | TxD     | _   | 14  |  |
|       |     | 0v      |   | 9    10  | 1 |  | RxD     | _   | 15  |  |
| 17    | _   | GPIO. 0 | 0 | 11    12 | 0 |  | GPIO. 1 | _   | 18  |  |
| 27    | _   | GPIO. 2 | 0 | 13    14 |   |  | 0v      |     |     |  |
| 22    | _   | GPIO. 3 | 0 | 15    16 | 0 |  | GPIO. 4 | _   | 23  |  |
|       |     | 3.3v    |   | 17    18 | 0 |  | GPIO. 5 | _   | 24  |  |
| 10    | _   | MOSI    | 0 | 19    20 |   |  | 0v      |     |     |  |
| 9     | _   | MISO    | 0 | 21    22 | 0 |  | GPIO. 6 | _   | 25  |  |
| 11    | _   | SCLK    | 0 | 23    24 | 1 |  | CE0     | _   | 8   |  |
|       |     | 0v      |   | 25    26 | 1 |  | CE1     | _   | 7   |  |
| 0     | _   | SDA.0   | 1 | 27    28 | 1 |  | SCL.0   | _   | 1   |  |
| 5     | _   | GPIO.21 | 1 | 29    30 |   |  | 0v      |     |     |  |
| 6     | _   | GPIO.22 | 1 | 31    32 | 0 |  | GPIO.26 | _   | 12  |  |
| 13    | _   | GPIO.23 | 0 | 33    34 |   |  | 0v      |     |     |  |
| 19    | _   | GPIO.24 | 0 | 35    36 | 0 |  | GPIO.27 | _   | 16  |  |
| 26    | _   | GPIO.25 | 0 | 37    38 | 0 |  | GPIO.28 | _   | 20  |  |
|       |     | 0v      |   | 39    40 | 0 |  | GPIO.29 | _   | 21  |  |

| Pi 3B |     |      |   |          |   |  |      |     |     |  |
|-------|-----|------|---|----------|---|--|------|-----|-----|--|
| BCM   | wPi | Name | V | Physical | V |  | Name | wPi | BCM |  |
|       |     |      |   |          |   |  |      |     |     |  |

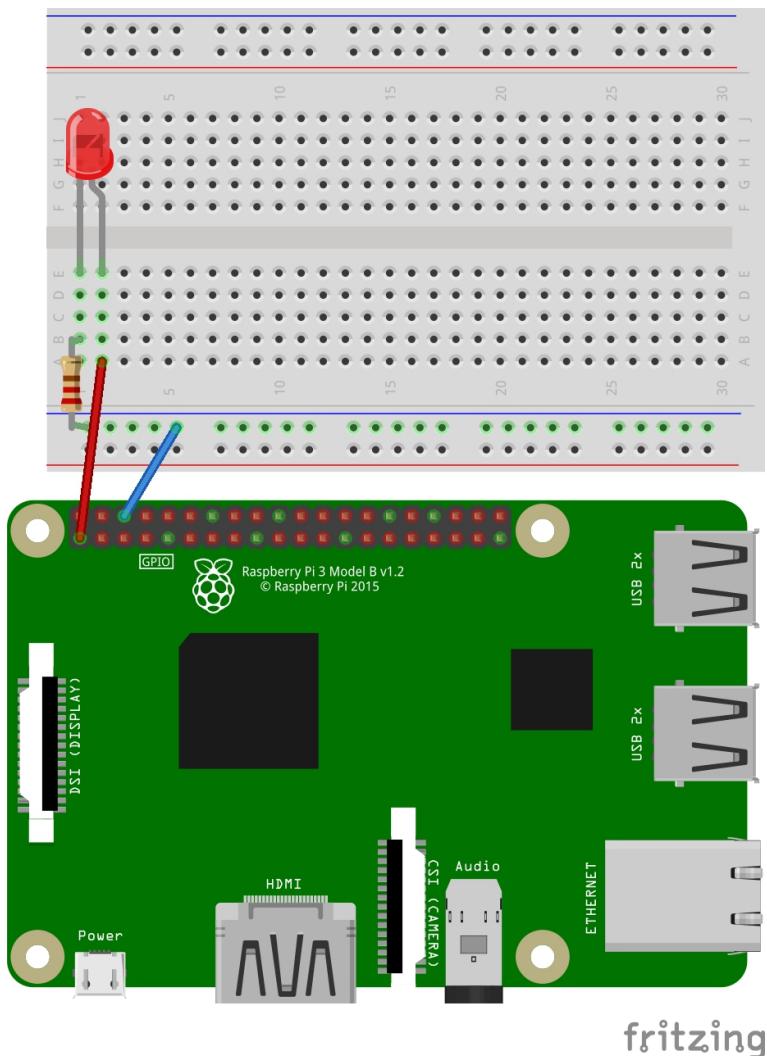
## J.2 วงจรไฟ LED กระแสเดียว

1. รายการอุปกรณ์ที่ต้องใช้:

- หลอด LED จำนวน 1 หลอด
- ตัวต้านทาน (Resistor) ขนาด 2-10 กิโลโหร์ม จำนวน 1 ตัว
- แผ่นต่อวงจรโพร์เตบอร์ด
- สายต่อวงจรชนิดต่างๆ ผู้-เมีย (Male-Female) และ ผู้-ผู้ (Male-Male) จำนวนหนึ่ง

2. ซัพพลายและตัดไฟเลี้ยงออกจากบอร์ด Pi เพื่อความปลอดภัยในการต่อวงจร

3. ศึกษาตารางที่กรอกก่อนหน้านี้ให้เข้าใจ แล้วจึงต่อวงจรตามรูปที่ [J.1](#)



รูปที่ J.1: วงจรเชื่อมต่อหลอด LED กับบอร์ด Pi ในการทดลองที่ 10 เพื่อทดสอบว่าหลอด LED ทำงาน ที่มา: [fritzing.org](http://fritzing.org)

4. จงดาวงจรที่ต่อในรูปที่ J.1 ประกอบด้วย ตัวต้านทาน ไฟเลี้ยง 3.3 โวลต์ ขา LED และกราวน์ (0 โวลต์)
5. ตรวจสอบความถูกต้อง โดยให้ผู้ควบคุมการทดลองตรวจสอบ
6. จ่ายไฟเลี้ยงให้กับบอร์ดแล้วสังเกตการเปลี่ยนแปลงที่หลอด LED หากหลอด LED ไม่สว่าง ขอความช่วยเหลือจากผู้ควบคุมการทดลอง

### J.3 โปรแกรมไฟ LED กระพริบภาษา C

1. เรียกโปรแกรม Code::Blocks ผ่านทาง Terminal โดยใช้สิทธิ์ของ SuperUser ดังนี้

```
$ sudo codeblocks
```

2. สร้าง project ใหม่ชื่อ Lab10 จนเสร็จสิ้น
3. คลิกเมนู "Setting/Compiler..." เลือกแท็บ "Linker settings" และกดปุ่ม "Add"

4. ป้อนประโภค ”/usr/lib/libwiringPi.so;” ในหน้าต่าง Add Library และกดปุ่ม ”OK” เพื่อปิดหน้าต่าง
5. กดปุ่ม ”OK” เพื่อยืนยัน
6. ป้อนโปรแกรมลงในไฟล์ใหม่ที่สร้างขึ้นโดยให้ชื่อว่า main.c

```
#include <stdio.h>
#include <stdlib.h>
#include <wiringPi.h>
int main ( void ) {
int pin = 7;
printf("LED blinking by wiringPi\n");
if (wiringPiSetup() == -1) {
    printf( "Setting up problem ... Abort!" );
    exit (1);
}
pinMode(pin, OUTPUT); /* set pin=7 to Output mode */
int i;
for ( i=0; i<10; i++ ) {
    digitalWrite(pin, 1); /* LED On */
    delay(500);
    digitalWrite(pin, 0); /* LED Off */
    delay(500);
}
return 0;
}
```

7. ทำการ Build และแก้ไขหากมีข้อผิดพลาดจนสำเร็จ
8. ย้ายสายจากขา 1 ของหัวเชื่อมต่อ 40 ขาไปยังขาหมายเลข 7 ซึ่งจะตรงกับ pin = 7 หรือ GPIO 7 ในตารางที่กรอกก่อนหน้า
9. Run และสังเกตการเปลี่ยนแปลงที่หลอดไฟ LED หากหลอด LED ไม่สว่าง ขอความช่วยเหลือจากผู้ควบคุม การทดลอง
10. จับเวลาช่วงเวลาที่หลอดสว่างและดับตั้งแต่เริ่มรันโปรแกรมจนเสร็จสิ้น เพื่อหาค่าเฉลี่ยของการสว่างดับ 1 รอบ

## J.4 โปรแกรมไฟ LED กระพริบภาษาแอสเซมบลี

1. เปิดไดเรกทอรี `/home/pi/asm` ในโปรแกรมไฟล์เมเนเจอร์
2. สร้างไดเรกทอรีใหม่ชื่อ `Lab10`
3. สร้างไฟล์ใหม่ชื่อ `Lab10.s` โดยใช้คำสั่ง `touch`
4. กรอกโปรแกรมภาษาแอสเซมบลีเหล่านี้โดยใช้ editor ที่คุณด

```

#-----
# data segment
#-----

.data
.balign 4

intro: .asciz "LED blinking by wiringPi\n"
errMsg: .asciz "Setting up problem ... Abort!\n"
pin:    .int    7
i:       .int    0
duration:.int   500
OUTPUT   = 1    @constant

#-----
# text segment
#-----

.text
.global main
.extern printf
.extern wiringPiSetup
.extern delay
.extern digitalWrite
.extern pinMode

main:  PUSH    {ip, lr} @push link return register on stack segment
      LDR     R0, =intro
      BL     printf
      BL     wiringPiSetup
      MOV     R1, #-1
      CMP     R0, R1
      BNE     init
      LDR     R0, =errMsg
      BL     printf
      B      done

```

```

init:
    LDR      R0, =pin
    LDR      R0, [R0]
    MOV      R1, #OUTPUT
    BL       pinMode
    LDR      R4, =i
    LDR      R4, [R4]
    MOV      R5, #10

forLoop:
    CMP      R4, R5
    BGT      done
    LDR      R0, =pin
    LDR      R0, [R0]
    MOV      R1, #1
    BL       digitalWrite
    LDR      R0, =duration
    LDR      R0, [R0]
    BL       delay
    LDR      R0, =pin
    LDR      R0, [R0]
    MOV      R1, #0
    BL       digitalWrite
    LDR      R0, =duration
    LDR      R0, [R0]
    BL       delay
    ADD      R4, #1
    B        forLoop

done:
    POP     {ip, pc} @pop return address into pc

```

5. ทำการแปลงและลิงก์ Lab10.s กับด้วยไลบรารี wiringPi จนกว่าจะสำเร็จ:

```

$   as -o Lab10.o Lab10.s
$   gcc -o Lab10 Lab10.o -lwiringPi

```

6. รันโปรแกรม Lab10 ด้วยสิทธิ์ของ SuperUser และสังเกตการเปลี่ยนแปลงที่หลอดไฟ LED

```
$   sudo ./Lab10
```

7. จับเวลาช่วงเวลาที่หลอดสว่างและตับตี้แต่เริ่มรันโปรแกรมจนเสร็จสิ้น เพื่อหาค่าเฉลี่ยของการสว่างตับ 1 รอบ

## J.5 กิจกรรมท้ายการทดลอง

1. ไลบรารี libwiringPi.so ทำหน้าที่อะไร และเกี่ยวข้องกับ #include <wiringPi.h> อย่างไร
2. ประโยชน์ § gcc -o Lab10 Lab10.o -lwiringPi มีความหมายอย่างไร และเขื่อมโยงกับคำถามข้อที่แล้ว อย่างไร
3. พังก์ชัน digitalWrite ใช้กับขา GPIO ในโหมดไหน
4. ประโยชน์ PUSH {ip, lr} ทำหน้าที่อะไร เหตุใดจึงต้องเรียกใช้ก่อนประโยชน์อื่นๆ
5. ประโยชน์ POP {ip, pc} ทำหน้าที่อะไร เหตุใดจึงต้องเรียกใช้เป็นประโยชน์สุดท้าย
6. คลิกบนลิงก์ชื่อ <https://github.com/WiringPi/WiringPi/blob/master/wiringPi/wiringPi.c> เพื่อใช้เบราว์เซอร์เปิดและตอบคำถามต่อไปนี้
7. สำรวจไฟล์ชื่อ wiringPi.c ที่เปิดเพื่อค้นหาตัวแปรชื่อ piGpioBase ว่า
  - ใช้งานในพังก์ชันชื่ออะไร
  - ได้รับการตั้งค่าที่พังก์ชันชื่ออะไร และค่าเท่ากับเท่าไหร่
  - นำตัวแปร piGpioBase นี้ไปใช้ทำอะไรต่อได้อีก จงยกตัวอย่าง
  - หมายเลขแอดเดรส 0x2000 0000 นี้เกี่ยวข้องกับหมายเลข 0x7E00 0000 ในตารางที่ 6.4 และรูปที่ 6.16 อย่างไร
8. จงค้นหาประโยชน์และตอบคำถามต่อไปนี้
 

```
gpio = (uint32_t *) mmap(0, BLOCK_SIZE, PROT_READ | PROT_WRITE,
                           MAP_SHARED, fd, GPIO_BASE) ;
```

  - อยู่ในพังก์ชันชื่ออะไร
  - ตัวแปร fd มาจากไหน เกี่ยวข้องกับไฟล์ /mem และไฟล์ /dev/gpiomem อย่างไร
  - พังก์ชัน mmap() มีหน้าที่อะไร รีเทิร์นค่าอะไรกลับมา และเป็นตัวแปรชนิดใด เหตุใดจึงต้องมีประโยชน์ (uint32\_t \*) นำหน้า
  - นำตัวแปร gpio นี้ไปใช้ทำอะไรต่อได้อีก จงยกตัวอย่าง
  - จงอธิบายว่าตัวแปร gpio นี้เกี่ยวข้องกับหลักการ Memory Map IO อย่างไร
9. จงตอบคำถามจากประโยชน์ต่อไปนี้

```
GPIO_BASE = piGpioBase + 0x00200000 ;
```

- อยู่ในพังก์ชันชื่ออะไร
- ตัวแปร GPIO\_BASE มีหน้าที่อะไร
- เมื่อบวกแล้วได้ผลลัพธ์เป็นหมายเลขแอดเดรสอะไร และเกี่ยวข้องกับหมายเลข 0x7E20 0000 ในตารางที่ 6.6 อย่างไร

- นำตัวแปร GPIO\_BASE นี้ไปใช้ทำอะไรต่อได้อีก จงยกตัวอย่าง
- จงอธิบายว่าตัวแปร GPIO\_BASE นี้เกี่ยวข้องกับขา 4pio แต่ละขาอย่างไร

10. ต่อหลดต LED เพิ่มอีก 2 ดวงรวมเป็น 3 ดวงแล้วพัฒนาโปรแกรมภาษา C เดิมให้นับเลขจำนวนเต็มฐานสิบ 0 - 7 และแสดงผลทางหลดต LED เป็นเลขฐานสองวนไปเรื่อยๆ

11. ใช้งจรหลดต LED 3 ดวงที่มีอยู่และพัฒนาโปรแกรมภาษาแอกซ์เพรสเซนบลีเดิมให้นับเลขจำนวนเต็มฐานสิบ 0 - 7 และแสดงผลทางหลดต LED เป็นเลขฐานสองวนไปเรื่อยๆ

# ภาคผนวก K

## การทดลองที่ 11 การเชื่อมต่ออินพุต-เอาต์พุตกับสัญญาณอินเทอร์รัปท์

การทดลองนี้คาดว่าผู้อ่านเคยเรียนการเขียนหรือพัฒนาโปรแกรมด้วยภาษา C และแօสเซมบลีจากการทดลองก่อนหน้า ดังนั้น การทดลองมีวัตถุประสงค์เหล่านี้

- เพื่อพัฒนาการทำงานของอินเทอร์รัปท์ร่วมโปรแกรมภาษา C และแօสเซมบลี ตามเนื้อหาในหัวข้อที่ [6.12](#)
- เพื่อศึกษาการทำงานของอินเทอร์รัปท์ร่วมกับขา GPIO ตามเนื้อหาในหัวข้อที่ [6.11](#)

### K.1 การจัดการอินเทอร์รัปท์ของ WiringPi

ไลบรารี wiringPi รองรับการทำอินเทอร์รัปท์ของ GPIO ได้ ทำให้โปรแกรมหลักสามารถทำงาน หลักได้ตามปกติ เมื่อเกิดสัญญาณอินเทอร์รัปท์ขึ้น ไม่ว่าจะเป็นสัญญาณจากการกดปุ่มสวิตซ์ ทำให้เกิดขอบขาขึ้นหรือขอบขาลงหรือทั้งสองขอบ โดยการเรียกใช้คำสั่ง

```
wiringPiISR(pin, edgeType, &callback)
```

โดย pin หมายถึง เลขชาติ wiringPi กำหนด edgeType กำหนดจากค่าคงที่ 4 ค่านี้

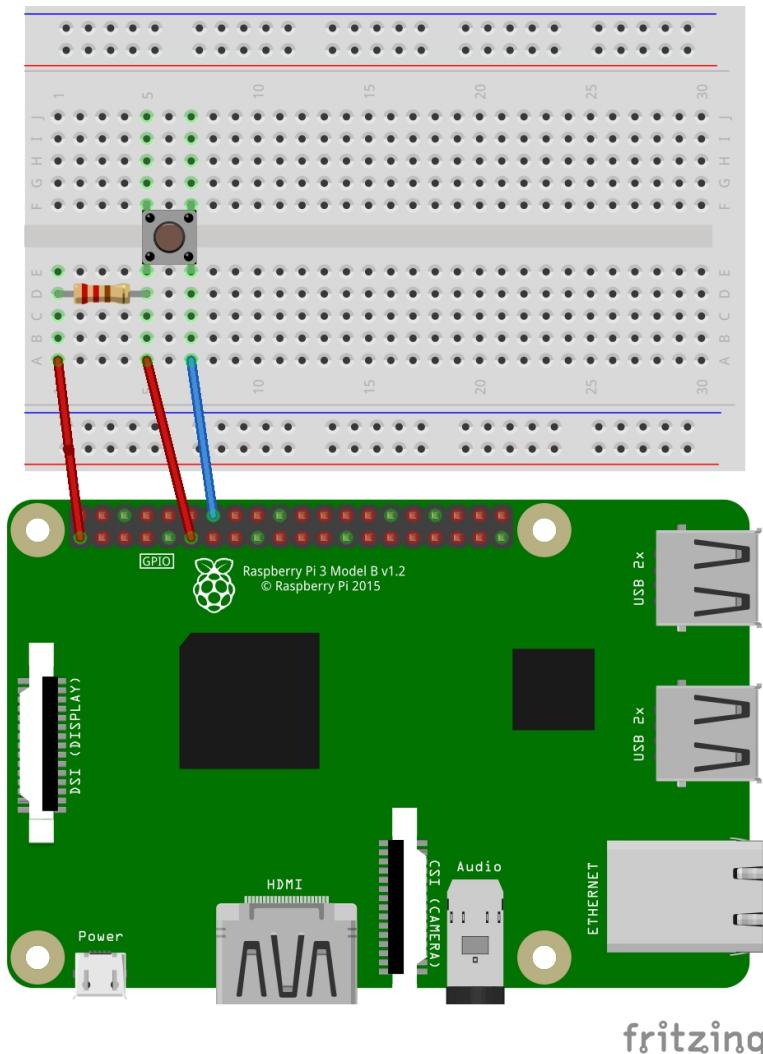
- INT\_EDGE\_FALLING,
- INT\_EDGE\_RISING,
- INT\_EDGE\_BOTH
- INT\_EDGE\_SETUP.

การทำหนดชนิดขอบขา เป็น 3 ชนิด แรก ไลบรารี จะตั้ง ค่าเริ่มต้น (Initialization) ให้โดยอัตโนมัติ หากกำหนดชนิดขอบเป็น INT\_EDGE\_SETUP ไลบรารี จะไม่ตั้งค่าเริ่มต้น (Initialization) ให้ เนื่องจากโปรแกรมเมอร์จะต้องกำหนดเอง

พารามิเตอร์ callback คือ ชื่อฟังก์ชันที่จะทำหน้าที่เป็น ISR สัญลักษณ์ & หมายถึง แອดเดรสของฟังก์ชัน callback ฟังก์ชัน callback นี้จะเริ่มต้นทำงานโดยแจ้งต่อวงจร Dispatcher ในหัวข้อที่ [6.12](#) ก่อนจะเริ่มต้นทำงาน โดยฟังก์ชัน callback จะสามารถอ่าน หรือเขียนค่าของตัวแปรโกลบอลในโปรแกรมได้ ซึ่งตัวอย่างการทำงานจะได้กล่าวในหัวข้อถัดไป

## K.2 วงจรสวิตซ์ปุ่มกดเชื่อมผ่านขา GPIO

1. ซัตดาวน์และตัดไฟเลี้ยงออกจากบอร์ด Pi เพื่อความปลอดภัยในการต่อวงจร
2. ต่อวงจรตามรูปที่ [K.1](#)



รูปที่ K.1: วงจรสวิตซ์ปุ่มกดสำหรับทดลองการเขียนโปรแกรมอินเทอร์รัปต์ในการทดลองที่ 11 ที่มา: [fritzing.org](http://fritzing.org)

3. จัดวงจรที่ต้องในรูปที่ [K.1](#) ประกอบด้วย สวิตซ์ปุ่มกด ตัวต้านทาน ไฟเลี้ยง 3.3 โวลต์ ขา BUTTON\_PIN และกราวน์ (0 โวลต์)
4. ตรวจสอบความถูกต้อง โดยให้ผู้ควบคุมการทดลองตรวจสอบ
5. สร้าง project ใหม่ชื่อ Lab11 ภายใต้ไดเรกทอรี `/home/pi/asm/Lab11`

## K.3 โปรแกรมภาษา C สำหรับทดลองอินเทอร์รัปท์

ผู้อ่านต้องทำความเข้าใจกับตัวโปรแกรมก่อนคอมไพล์หรือรันโปรแกรม เพื่อความเข้าใจสูงสุด โดยเฉพาะชื่อตัวแปร ชนิดของตัวแปร evenCounter การติดตั้งฟังก์ชัน wiringPiISR เพื่อเชื่อมโยงกับขา GPIO ชนิดของการ

ตรวจจับ และขีดฟังก์ชัน myInterrupt ซึ่งทำหน้าที่เป็น ISR หรือ ฟังก์ชัน callback

```
#include <stdio.h>
#include <errno.h>
#include <wiringPi.h>
#define BUTTON_PIN 0
// Use GPIO Pin 17 = Pin 0 of wiringPi library
volatile int eventCount = 0;
void myInterrupt(void) { // called every time an event occurs
    eventCount++; // the event counter
}
int main(void) {
    if (wiringPiSetup() < 0) // check the existence of wiringPi library
    {
        printf ("Cannot setup wiringPi: %s\n", strerror (errno));
        return 1; // error code = 1
    }
    // set wiringPi Pin 0 to generate an interrupt from 1-0 transition
    // myInterrupt() = my Interrupt Service Routine
    if (wiringPiISR (BUTTON_PIN, INT_EDGE_FALLING, &myInterrupt) < 0) {
        printf ("Cannot setup ISR: %s\n", strerror (errno));
        return 2; // error code = 2
    }
    // display counter value every second
    while(1) {
        printf("%d\n", eventCount);
        eventCount = 0;
        delay(1000); // wait 1000 milliseconds = 1 second
    }
    return 0; // error code = 0 (No Error)
}
```

1. ป้อนโปรแกรมด้านบนใน main.c โดยใช้โปรแกรม Text Editor ทั่วไป
2. สร้าง makefile สำหรับคอมไพล์และลิงก์โปรแกรมจากการทดลองก่อนหน้านี้ จนไม่เกิดข้อผิดพลาด
3. รันโปรแกรม ทดสอบการทำงานด้วยการกดปุ่มสวิตช์ที่ต่อໄว สังเกตผลลัพธ์ทางหน้าจอ Terminal ที่รัน

§ sudo ./Lab11

## K.4 กิจกรรมท้ายการทดลอง

1. จงวัดสัญญาณที่ขา BUTTON\_PIN ก่อนกดปุ่มสวิตช์ ระหว่างกดปุ่มสวิตช์ และปล่อยมือจากสวิตช์ปุ่มกด โดยให้แก่นอนเป็นแกนเวลา แกนตั้งเป็นค่าเวลา หรือ ค่าลอจิกของขาสัญญาณ BUTTON\_PIN
2. จงบอกความหมายและการประยุกต์ใช้งานตัวแปรชนิด volatile
3. ปรับแก้ volatile ออกเหลือแค่ int eventCount = 0; make แล้วจึงรันโปรแกรมทดสอบการทำงานด้วยการกดปุ่มสวิตช์ที่ต่อไว้ สังเกตผลลัพธ์ทางหน้าจอ Terminal ที่รัน เปรียบเทียบการทำงานของโปรแกรมก่อนและหลังการปรับแก้ และหาเหตุผล
4. จงปรับแก้โปรแกรมที่ทดลองตามประโยชน์ต่อไปนี้

```
if (wiringPiISR (BUTTON_PIN, INT_EDGE_RISING, &myInterrupt) < 0) {
    ...
}
```

ทำ make ใหม่และทดลองกดปุ่มสวิตช์ สังเกตการเปลี่ยนแปลงและอธิบาย

5. จงตอบคำถามจากประโยชน์ต่อไปนี้

```
if (wiringPiISR (BUTTON_PIN, INT_EDGE_FALLING, &myInterrupt) < 0) {
    ...
}
```

- พังก์ชัน wiringPiISR ทำหน้าที่อะไร เหตุใดอยู่ในประโยชน์เงื่อนไข if
- ตัวแปร &myInterrupt คืออะไร เหตุใดจึงมีสัญลักษณ์ & นำหน้า
- พังก์ชันนี้เชื่อมโยงกับตารางที่ 6.6 อย่างไร

6. จงใช้วงจรหลอด LED 3 ดวงและโปรแกรมจากการทดลองที่ 10 นับขึ้นจาก 0-7-0 โดยเพิ่มสวิตช์ปุ่มกดในการทดลองนี้ และเพิ่มฟังก์ชันการอินเทอร์รัปท์จากโปรแกรม Lab11.2 นี้ เมื่อกดปุ่มแต่ละครั้งจะทำให้ความเร็วในการนับเพิ่มขึ้น หรือ Delay สั้นลงครึ่งหนึ่ง เมื่อกดครั้งที่ 2 จะสั้นลงอีกครึ่งหนึ่ง เมื่อกดครั้งที่ 3 จะทำให้ Delay กลับไปเป็นค่าเริ่มต้น
7. จงใช้วงจรหลอด LED 3 ดวงและโปรแกรมจากการทดลองที่ 10 แต่นับลงจาก 7-0-7 โดยเพิ่มสวิตช์ปุ่มกดในการทดลองนี้ และเพิ่มฟังก์ชันการอินเทอร์รัปท์จากโปรแกรม Lab11.2 นี้ เมื่อกดปุ่มแต่ละครั้งจะทำให้ความเร็วในการนับลดลง หรือ Delay เพิ่มขึ้นเท่าตัว เมื่อกดครั้งที่ 2 Delay เพิ่มขึ้นอีกเท่าตัว เมื่อกดครั้งที่ 3 จะทำให้ Delay กลับไปเป็นค่าเริ่มต้น

## ภาคผนวก L

# การทดลองที่ 12 การศึกษาอุปกรณ์เก็บรักษาข้อมูล และระบบไฟล์

การทดลองนี้อธิบายและเข้มโยงเนื้อหาความรู้ของทุกบทเข้าด้วยกัน แต่จะเน้นบทที่ 6 และบทที่ 7 เพื่อให้ผู้อ่านมองเห็นอุปกรณ์อินพุตและเอาต์พุตเหมือนไฟล์เต็มไฟล์ โดยมีวัตถุประสงค์ดังนี้

- เพื่อให้เข้าใจการวัดขนาดของไฟล์และไดเรกทอรีในระบบไฟล์
- เพื่อให้รู้จักรองสร้างและระบบไฟล์ของการดحن่วยความจำไมโคร SD ที่ใช้งานในปัจจุบัน
- เพื่อให้เข้าใจระบบไฟล์ (File System) ชนิดต่างๆ บนบอร์ด Pi
- เพื่อให้สามารถเชื่อมโยงอุปกรณ์อินพุต/เอาต์พุตชนิดต่างๆ กับระบบไฟล์

### L.1 ขนาดของไฟล์และไดเรกทอรี

ผู้อ่านสามารถตรวจสอบขนาดของไฟล์ใดๆ ชื่อ filename ที่แท้จริง หน่วยเป็นไบต์ ด้วยคำสั่ง du (Disk Usage) โดยตามขั้นตอนต่อไปนี้

- ย้ายไดเรกทอรีปัจจุบันไปที่ /home/pi ซึ่งเป็นไดเรกทอรีหลักของผู้ใช้ชื่อ pi

```
$ cd /home/pi
```

- สร้างไฟล์ข้อความ test.txt ด้วยโปรแกรม nano ด้วยคำสั่งต่อไปนี้

```
$ nano test.txt
```

พิมพ์ข้อความ fdd ลงในไฟล์ ทำการ Write โดยกดปุ่ม Ctrl แซ่ตตามด้วยปุ่ม O ออกจากโปรแกรมโดยกดปุ่ม Ctrl แซ่ตตามด้วยปุ่ม X

- คำสั่ง ‘du -b filename’ จะแสดงผลขนาดเป็นจำนวนไบต์นำหน้าชื่อไฟล์นั้น

```
$ du -b test.txt
_ test.txt
```

ตัวเลข \_ หมายถึง เลขจำนวนไบต์ที่คำสั่ง du แสดงผลมาตามพารามิเตอร์ b ที่ส่งไป เพื่อบอกค่าขนาดของไฟล์ test.txt เป็นจำนวน \_ ไบต์

- คำสั่ง ‘du -B1 filename’ ผู้อ่านสามารถตรวจสอบขนาดของไฟล์ใดๆ ชื่อ filename ที่จัดเก็บเป็นจำนวนเท่าของ \_\_\_\_\_ ไบต์ ในอุปกรณ์เก็บรักษาข้อมูล SD ด้วยคำสั่งต่อไปนี้

```
$ du -B1 test.txt
____ test.txt
```

ตัวเลข \_\_\_\_\_ หมายถึง เลขจำนวนไบต์ที่คำสั่ง du แสดงผลมาตามพารามิเตอร์ B1 ที่ส่งไป โดยผู้อ่านจะสังเกตเห็นความแตกต่าง ถึงแม้ไฟล์มีข้อมูลจำนวนน้อยเพียงไม่ถึงไบต์ แต่การของพื้นที่ในอุปกรณ์สำรองจะมีขนาดเป็นจำนวนเท่าของ \_\_\_\_\_ ไบต์เสมอ เช่น 8192, 16384 เป็นต้น

- คำสั่ง ‘du -h’ จะแสดงผลขนาดหรือจำนวนไบต์โดยใช้หน่วยเช่น K (Kibi: 1024) M (Mebi: 1048576) G (Gibi: 1073741824) นำหน้าชื่อไดเรกทอรีหรือไฟล์เดอร์ที่อยู่ใต้ไดเรกทอรีปัจจุบัน และจดบันทึก 5 รายการแรกในตาราง

```
$ du -h
```

| Size | Folder Name |
|------|-------------|
|      |             |
|      |             |
|      |             |
|      |             |
|      |             |

## L.2 ระบบไฟล์

ผู้ใช้หรือผู้ดูแลระบบลินุกซ์ สามารถตรวจสอบการใช้งานอุปกรณ์เก็บรักษาข้อมูล เช่น ฮาร์ดดิสก์ไดร์ฟ โซลิดสเตทไดร์ฟ การ์ดหน่วยความจำ SD ได้โดยคำสั่ง

- คำสั่ง **df** (Disk File System) สามารถแสดงรายละเอียดของอุปกรณ์เก็บรักษาข้อมูลในเครื่อง
- คำสั่ง ‘**df -h**’ จะแสดงรายการ ดังต่อไปนี้ จดบันทึก 5 รายการแรกลงในตารางเพื่อเปรียบเทียบกับตารางที่แล้ว

```
$ df -h
```

| Filesystem | Size | Used | Available | Use% | Mounted on |
|------------|------|------|-----------|------|------------|
|            |      |      |           |      |            |
|            |      |      |           |      |            |
|            |      |      |           |      |            |
|            |      |      |           |      |            |
|            |      |      |           |      |            |

โดย Size จะแสดงผลขนาดหรือจำนวนไบต์โดยใช้ตัวคูณที่แตกต่างกัน เช่น K (Kibi: 1024) M (Mebi: 1048576) G (Gibi: 1073741824)

- คำสั่ง ‘**df -T**’ จะเพิ่มคอลัมน์ชนิด (Type) ของแต่ละรายการในการแสดงผล และขนาดเป็นจำนวนเท่าของ 1 KiB (KibiByte) (1K) แทน จดบันทึก 5 รายการที่ตรงกับตารางที่แล้ว

```
$ df -T
```

| Filesystem | Type | 1K-blocks Used | Available | Use% | Mounted on |
|------------|------|----------------|-----------|------|------------|
|            |      |                |           |      |            |
|            |      |                |           |      |            |
|            |      |                |           |      |            |
|            |      |                |           |      |            |
|            |      |                |           |      |            |

- คำสั่ง ‘df -i’ จะแสดงรายการต่างๆ ดังนี้ จดบันทึก 5 รายการที่ตรงกับตารางที่แล้ว

```
$ df -i
```

| Filesystem | Inodes | IUsed | IFree | IUse% | Mounted on |
|------------|--------|-------|-------|-------|------------|
|            |        |       |       |       |            |
|            |        |       |       |       |            |
|            |        |       |       |       |            |
|            |        |       |       |       |            |
|            |        |       |       |       |            |

โดยคอลัมน์ที่ 2 จากทางซ้ายจะแสดงผลเป็นจำนวน ไโหนด แทน รายละเอียดเรื่องไโหนด ผู้อ่านสามารถค้นคว้าเพิ่มเติมได้ในบทที่ 7 และทาง [wikipedia](#)

- คำสั่ง **stat** แสดงรายละเอียดของไฟล์หรือไดเรกทอรี การทดลองนี้จะใช้ไดเรกทอรี asm ที่มีอยู่ และเติมตัวเลขในช่องว่าง

```
$ cd /home/pi
$ stat asm
```

```
File: asm
Size: _____ Blocks: _____ IO Block: _____
Device: _____h/_____d Inode: _____ Links: 3
Access: (_____/drwxr-xr-x) Uid: ( ____/____) Gid: ( ____/____)
Access: ...
Modify: ...
Change: ...
Birth: -
```

ผู้อ่านจะต้องกรอกผลลัพธ์ในช่องว่าง ดังต่อไปนี้

- ชื่อ asm

- ขนาด \_\_\_\_\_ ไบต์ ใช้พื้นที่จำนวน \_\_\_\_\_ Blocks ซึ่งหมายถึง 8 เซ็กเตอร์ฯ ละ 512 ไบต์ เป็น \_\_\_\_\_
- มีหมายเลข Device = \_\_\_\_\_h/\_\_\_\_\_d หรือเท่ากับ \_\_\_\_\_<sub>16</sub>/\_\_\_\_\_<sub>10</sub>
- มีหมายเลข Inode = \_\_\_\_\_<sub>10</sub> จำนวน 3 Links
- สิทธิ์เข้าถึง (Access) ด้วยรหัส \_\_\_\_\_<sub>16</sub> หรือ \_\_\_\_\_<sub>2</sub>:\_\_\_\_\_<sub>2</sub>:\_\_\_\_\_<sub>2</sub> โดยผู้ใช้หมายเลข Uid (User ID)=\_\_\_\_\_ ชื่อผู้ใช้ (Username)=\_\_\_\_\_ ในกรุํปหมายเลข Groupid=\_\_\_\_\_ ชื่อกรุํป \_\_\_\_\_

- เข้าถึง (Access) ...
- เปลี่ยนแปลง (Modify) ...
- เวลาที่ Inode เปลี่ยนแปลง (Change) ...

เบื้องต้นผู้เขียนขอให้ผู้อ่านสร้างไฟล์ผลลัพธ์จากคำสั่ง stat ไปเก็บในไฟล์ เพื่อมาใช้ประกอบการทดลองต่อไปโดย

```
$ stat asm > stat_asm.txt
```

หลังจากนั้น เราสามารถตรวจสอบสถานะของไฟล์ stat\_asm.txt ได้ดังนี้

```
$ stat stat_asm.txt
```

```
File: stat_asm.txt
Size: _____ Blocks: _____ IO Block: 4096 _____
Device: _____h/_____d Inode: _____ Links: 1
Access: (____/-rw-r--r--) Uid: ( ____/____) Gid: ( ____/____)
Access: ...
Modify: ...
Change: ...
Birth: -
```

ผู้อ่านจะต้องกรอกผลลัพธ์ในช่องว่าง ดังต่อไปนี้

- ชื่อ stat\_asm.txt
- ขนาด \_\_\_\_\_ ไบต์ ใช้พื้นที่จำนวน \_\_\_\_\_ Blocks ซึ่งหมายถึง 8 เฮกเตอร์ฯ ละ 512 ไบต์ เป็น \_\_\_\_\_
- มีหมายเลข Device = \_\_\_\_\_h/\_\_\_\_\_d หรือเท่ากับ \_\_\_\_\_<sub>16</sub>/\_\_\_\_\_<sub>10</sub>
- มีหมายเลข Inode = \_\_\_\_\_<sub>10</sub> จำนวน 1 Links
- สิทธิ์เข้าถึง (Access Permission) ด้วยรหัส \_\_\_\_\_<sub>16</sub> หรือ \_\_\_\_\_<sub>2</sub>:\_\_\_\_\_<sub>2</sub>:\_\_\_\_\_<sub>2</sub> โดยผู้ใช้หมายเลข Uid (User ID)=\_\_\_\_\_ ชื่อผู้ใช้ (Username)=\_\_\_\_\_ ในกรุํปหมายเลข Groupid=\_\_\_\_\_ ชื่อกรุํป \_\_\_\_\_
- เข้าถึง (Access) ...
- เปลี่ยนแปลง (Modify) ...
- เวลาที่ Inode เปลี่ยนแปลง (Change) ...
- หมายเลข Inode ของ asm กับ หมายเลข Inode ของ stat\_asm.txt ตรงกันหรือไม่ เพราะเหตุใด
- asm เป็น ไดเรกทอรี ในขณะที่ stat\_asm.txt เป็น \_\_\_\_\_

- สิทธิ์เข้าถึง (Access Permission) รหัส 0765<sub>16</sub> มีความหมายดังต่อไปนี้
  - 111<sub>2</sub>: เป็นของใคร \_\_\_\_\_
  - 110<sub>2</sub>: เป็นของใคร \_\_\_\_\_
  - 101<sub>2</sub>: เป็นของใคร \_\_\_\_\_

### L.3 อุปกรณ์อินพุต/เอาต์พุตในระบบไฟล์

การทดลองในหัวข้อนี้จะเชื่อมต่อกับเนื้อหาในบทที่ 3 และ การทดลองที่ 4 ภาคผนวก D หลักการของระบบปฏิบัติการยูนิกซ์ คือ การมาท์ (Mount) อุปกรณ์กับไดเรกทอรีด้วยระบบไฟล์ (File System) ที่แตกต่างกัน โดยใช้ชื่อไดเรกทอรีที่แตกต่างกัน โดยมีไดเรกทอรีราก (Root Directory) หรือโฟลเดอร์ราก เป็นตำแหน่งเริ่มต้น ผู้อ่านสามารถพิมพ์คำสั่งใน Terminal

\$ mount

คำสั่งนี้จะแสดงรายชื่อการมาท์ หรือ ผูกยึด อุปกรณ์อินพุต/เอาต์พุต เข้ากับระบบไฟล์ที่เหมาะสมกับอุปกรณ์นั้นๆ ด้วยชื่อไดเรกทอรีหรือชื่อไฟล์ของระบบปฏิบัติการ ผู้อ่านจะต้องกรอกผลลัพธ์ที่สำคัญในช่องว่าง และศึกษาคำอธิบายต่อไปนี้

- /dev/mmcblk0p<sub>n</sub> on / type ext4 (rw,noatime) เป็นระบบไฟล์ **ext4** ซึ่งเป็นระบบไฟล์หลักของลินุกซ์ ย่อมาจากคำว่า Fourth Extended File System เป็นเวอร์ชันที่ 4 พัฒนาจากชนิด ext3 ซึ่งพัฒนาจากระบบยูนิกซ์ตามรายละเอียดในหัวข้อที่ 7.1 และ [wikipedia](#)
- devtmpfs on /dev type devtmpfs (rw, relatime, size=3834564k, nr\_inodes=958641, mode=755)
- proc on /proc type proc (rw, relatime) เป็นระบบไฟล์เสมือน (Virtual File System) สำหรับระบบสำคัญต่างๆ เช่น CPU, โดยจะสร้างขึ้นเมื่อบูตเครื่อง และลบทิ้งเมื่อชัตดาวน์ระบบ **รายละเอียดเพิ่มเติมที่ wikipedia**
- sysfs on /sys type sysfs (rw,nosuid,nodev,noexec,relatime) เป็นระบบไฟล์เสมือน (Virtual File System) **รายละเอียดเพิ่มเติมที่ wikipedia**
- securityfs on /sys/kernel/security type securityfs (rw, nosuid, nodev, noexec, relatime)
- tmpfs on /dev/shm type tmpfs (rw, nosuid, nodev) ย่อมาจากคำว่า Temporary File System **รายละเอียดเพิ่มเติมที่ wikipedia**
- devpts on /dev/pts type devpts (rw, nosuid, noexec, relatime, gid=5, mode=620, ptmxmode=000) เป็นระบบไฟล์เสมือน (Virtual File System) สำหรับอุปกรณ์อินพุตเอาต์พุตต่างๆ **รายละเอียดเพิ่มเติมที่ wikipedia**
- ...

- **/dev/mmcblk0p<sub>n</sub>** on /boot type vfat ระบบไฟล์ **vfat** เป็นส่วนต่อขยายของระบบไฟล์ FAT ซึ่งย่อมาจากคำว่า File Allocation Table เพื่อรองรับชื่อไฟล์ที่ยาวกว่า FAT ที่มา: [wikipedia](#)

- ...

รายชื่อต่อไปนี้ คือ ตัวเลือกคุณสมบัติ (Attribute) ที่สำคัญของระบบไฟล์ เช่น

- rw : read/write สามารถอ่านและเขียนได้
- noatime และ atime: No/ Access Time หมายถึง ไม่มี/มีการบันทึกเวลาในการสร้าง อ่านหรือเขียนไฟล์ทุกครั้ง
- relatime หมายถึง มีการบันทึกเวลาในการสร้าง อ่านหรือเขียนไฟล์ เมื่อเกิดการแก้ไขไฟล์ หรือ การอ่านหรือเข้าถึงไฟล์มากกว่าเวลาที่บันทึกไว้ก่อนหน้าอย่างน้อย 24 ชั่วโมง
- nosuid: No SuperUser ID เป็นการป้องกันไม่ให้ผู้ดูแลระบบ (SuperUser) กระทำการใดๆ ได้ เพื่อความมั่นคงปลอดภัย
- noexec: No Execution เพื่อตั้งค่าไม่ให้รันไฟล์ที่อยู่ในไดเรกทอรีได้ เช่น ไฟล์ที่เป็นไวรัสหรือมัลแวร์ (Malware) ที่แอบแฝงเข้ามา
- nodev: No Device หมายถึง การไม่อนุญาตให้สร้างหรืออ่านโหนด (Node) ซึ่งเป็นไฟล์ชนิดพิเศษ
- mode หมายถึง สิทธิ์การเข้าถึงไฟล์หรือไดเรกทอรี ประกอบด้วย บิตควบคุม Read Write Execute 3 ชุด รวมทั้งหมด 9 บิต ซึ่งได้อธิบายแล้วในหัวข้อที่ [7.1.4](#)

ผู้อ่านสามารถแสดงรายชื่อไฟล์หรือไดเรกทอรีหรือชื่ออุปกรณ์ภายใต้ไดเรกทอรี /dev โดยพิมพ์คำสั่งบนโปรแกรม Terminal

```
$ ls /dev
```

ผู้อ่านต้องเปรียบเทียบกับชื่ออุปกรณ์ที่ผู้เขียนตัวหน้าไว้ว่าตรงกันหรือไม่ อย่างไร เพื่อให้ผู้อ่านมองเห็นชัดว่า **mmcblk0p2** มีอยู่จริงและระบบได้ทำการเม้าท์เข้ากับไดเรกทอรีรoot (Root) นั้นคือ ไดเรกทอรี / ด้วยชนิด ext4 ตามที่ได้แสดงในคำสั่งก่อนหน้าแล้ว

```
ashem autofs block btrfs-control bus cachefiles cec0 cec1 char console cpu_dma_latency
cuse disk dma_heap dri fb0 fd full fuse gpiochip0 gpiochip1 gpiochip2 gpiomem hidraw0
hidraw1 hidraw2 hidraw3 hwrng i2c-20 i2c-21 initctl input kms kvm log loop0 loop1 loop2
loop3 loop4 loop5 loop6 loop7 loop-control mapper media0 media1 mem mmcblk0
mmcblk0p_ mmcblk0p_ mqueue net null port ppp ptmx pts ram0 ram1 ram10 ram11
ram12 ram13 ram14 ram15 ram2 ram3 ram4 ram5 ram6 ram7 ram8 ram9 random raw rfkill
rpivid-h264mem rpivid-hevcmem rpivid-initc rpivid-vp9mem serial1 shm snd stderr stdin
stdout tty tty0 ... ttyAMA0 ttyprintk uhid uinput urandom vchiq vcio vc-mem vcs ... watchdog
watchdog0 zero
```

นอกจากนี้ อุปกรณ์สำคัญอื่นๆ เช่น stdin (standard input) stdout (standard output) และ stderr (standard error) นั้นเกี่ยวข้องกับโปรแกรม Terminal ซึ่งเชื่อมโยงกับประโยชน์ในภาษา C ในการทดลองที่ 5 ภาคผนวก E

```
#include <stdio.h>
```

## L.4 กิจกรรมท้ายการทดลอง

1. ใช้โปรแกรม File Manager แล้วคลิกขวาบนชื่อไฟล์เพื่อแสดงคุณสมบัติ (Properties) ต่างๆ บนแท็บ General และอธิบายโดยเฉพาะหัวข้อ Total size of files และ Size on disk ว่าเหตุใดถึงแตกต่างกัน
2. สร้างไฟล์ (New) ด้วยโปรแกรม nano คือข้อความด้วยตัวอักษรจำนวน 1 ตัวแล้วบันทึก (Save) ใช้คำสั่ง ls -l เพื่อแสดงรายละเอียดของไฟล์นั้น เพื่อหาขนาดไฟล์ที่แท้จริง
3. โปรดสังเกตว่าใน test.txt ที่สร้างด้วยโปรแกรม nano เราได้พิมพ์ประโยชน์ fdd คิดเป็นจำนวน 3 ตัว อักษรฯ ละ 1 ไปต่อเนื่อง จนกว่าไปต่อที่ 4 คือตัวอักษรใดในรูปที่ [2.12](#)
4. เพิ่มจำนวนตัวอักษรไปเรื่อยๆ ใน test.txt จนทำให้ไฟล์มีขนาดมากกว่าเท่ากับ 4096 ไบต์ แล้วใช้คำสั่ง du -B1 test.txt ตรวจสอบขนาดไฟล์อีกรอบ บันทึกและอธิบายผลที่ได้โดยเฉพาะจำนวน Blocks ที่ได้จากคำสั่งว่าเท่ากับกี่เซกเตอร์
5. จะเปรียบเทียบผลลัพธ์ของคำสั่ง stat ระหว่าง ไดเรกทอรี และ ไฟล์
6. สิทธิ์การเข้าถึง (Permission) ของไดเรกทอรีหรือของไฟล์ประกอบด้วยบิตจำนวน 9 บิต แบ่งเป็น 3 ชุดๆ ละ 3 บิต จงเรียงลำดับชุดต่างๆ ว่าเป็นของสิทธิ์ของใครบ้าง
7. ใช้คำสั่งต่อไปนี้ เพื่อแสดงรายชื่อไดเรกทอรีและไฟล์ และอธิบายผลว่าหมายเลขอื่นๆ ด้านซ้ายสุดคืออะไร และเหตุใดจึงมีค่าซ้ำ

```
$ ls -i -l /
```

8. ใช้คำสั่งต่อไปนี้ เพื่อแสดงรายละเอียดของชื่อไดเรกทอรีและไฟล์ และอธิบายผลว่ามีอะไรที่แตกต่างกัน เพราะเหตุใด

```
$ stat /proc  
$ stat /sys
```

```
$ stat /dev  
$ stat /run
```

9. ใช้คำสั่งต่อไปนี้ เพื่อแสดงรายละเอียดของอุปกรณ์ และอธิบายว่ามีผลลัพธ์ที่แตกต่างกันหรือไม่ เพราะเหตุใด

```
$ stat /dev/mmcblk0p2  
$ stat /
```

10. จงอธิบายว่าเหตุใดไดเรกทอรี asound จึงอยู่ใต้ /proc ในหัวข้อที่ [1.2.3](#) การทดลองที่ 9 ภาคผนวก I
11. จงอธิบายความเชื่อมโยงระหว่าง gpiomem ที่ได้จากคำสั่ง ls /dev กับกิจกรรมท้ายการทดลองที่ 10 ภาคผนวก J

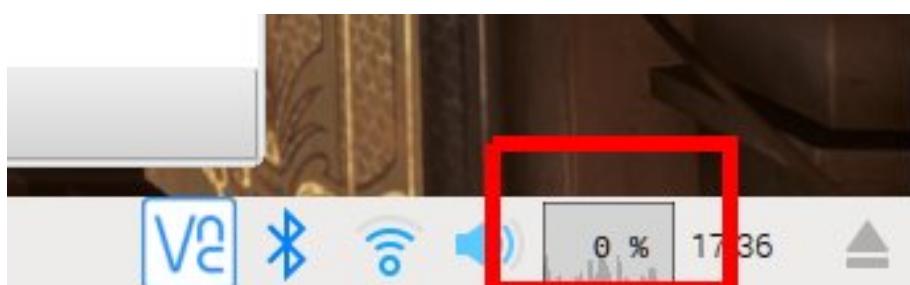
# ภาคผนวก M

## การทดลองที่ 13 การพัฒนาอัลกอริธึมแบบขนานด้วยไลบรารี OpenMP

การพัฒนาอัลกอริธึมแบบขนานบนชิปปี้ชนิดมัลติคอร์ในปัจจุบันจำเป็นต้องอาศัยภาษาคอมพิวเตอร์ระดับสูง เช่น ภาษา C/C++ ภาษา Java เป็นต้น เพื่อช่วยลดเวลา.ran (Run Time) ซึ่งเท่ากับเร่งความเร็ว (Speedup) ให้ อัลกอริธึมหรือโปรแกรม โดยการสร้างered ผู้ช่วย (Worker Thread) และมอบหมายงานให้ไปรับหนาแกนประมวลผลที่ยังว่างอยู่ ผู้อ่านสามารถประยุกต์ใช้หลักการนี้บนเครื่องคอมพิวเตอร์ทั่วไปจนถึงเครื่องซูเปอร์คอมพิวเตอร์ ตามเนื้อหาในบทที่ 8 ดังนั้น การทดลองมีวัตถุประสงค์ดังนี้

- เพื่อพัฒนาโปรแกรมภาษา C ด้วยไลบรารี OpenMP ให้สามารถทำงานแบบมัลติ-thread และใช้งานชิปปี้ชนิดมัลติคอร์ได้เต็มที่
- เพื่อเรียนรู้การวัด CPU Utilization (%CPU) เวลาจริง ( $T_{real}$ ) เวลาผู้ใช้ ( $T_{user}$ ) และเวลาระบบ ( $T_{sys}$ ) ในชิปปี้ชนิดมัลติคอร์
- เพื่อทำความเข้าใจการวัดประสิทธิภาพของอัลกอริธึมแบบขนานจากการประเมินความซับซ้อนเชิงเวลา ด้วยพีชคณิต BigO ที่มา: [Rosen \(2002\)](#) และตัวชี้วัด Speedup ที่มา: [Patterson and Hennessy \(2016\)](#) จากเวลาที่วัดได้

### M.1 การวัด CPU Utilization



รูปที่ M.1: กราฟแสดงการใช้งานชิปปี้ (CPU Usage Monitor) ย้อนหลังและค่าสรุป ณ เวลาปัจจุบัน ที่มา: [abload.de](#)

ผู้อ่านสามารถติดตั้งเครื่องมือและกราฟในรูปที่ M.1 แสดงการใช้งานซีพียู (CPU Usage Monitor) ย้อนหลัง และค่าสรุป ณ เวลาปัจจุบันของบอร์ด Pi ประกอบการทดลองที่ 13 ตามขั้นตอนเหล่านี้

1. เลื่อนมาสู่ไปบนตำแหน่งว่างของ Task Bar
2. คลิกขวา เพื่อให้เมนูต่อไปนี้ปรากฏขึ้นแล้วคลิกซ้ายเลือก Add/Remove Panel Items
3. คลิกที่แท็บ Panel Applets
4. เลื่อนรายการขึ้นลงเพื่อหารายการชื่อ CPU Usage Monitor แล้วคลิก Add
5. กดปุ่ม Up และ Down เพื่อวางตำแหน่งของ CPU Usage Monitor ในตำแหน่งที่ต้องการ โปรดสังเกตรายชื่อ เมื่อได้ตำแหน่งที่ต้องการแล้วกด Close หมายเหตุ Spacer หมายถึง ช่องว่างที่คั่นระหว่าง Applet ที่อยู่บน Task Bar
6. สังเกตด้านขวาของ Task Bar จะมีจอสีเทาขนาดเล็กแสดงเป็นกราฟแท่ง โดยแท่งขวा�สุดคือ วินาทีล่าสุด
7. เลื่อนมาสู่ไปบนกราฟแล้วคลิกขวาเพื่อเพิ่มการแสดงผลเป็นตัวเลขหน่วยเป็นเปอร์เซ็นต์ (%)
8. ทดสอบการทำงานโดยการเปิดคลิปเดียวกันบน YouTube.com ที่ความละเอียดแตกต่างกัน เช่น 240p, 480p และ 720p ทีลักษณะเพื่อให้เห็นค่า  $\%CPU_{max}$  ที่แตกต่าง

## M.2 การคูณเมทริกซ์แบบบานาน

$$C = A \times B \quad (\text{M.1})$$

การคูณเมทริกซ์เป็นพื้นฐานของการคำนวณพื้นฐานทางวิทยาศาสตร์ และวิศวกรรมศาสตร์ กำหนดให้เมทริกซ์จตุรัส  $A$  ขนาด  $N \times N$  สามารถเขียนในรูปแบบของอาร์เรย์ 2 มิติในภาษา C/C++ ได้ดังนี้

$$A = \left( A[i][j] \right)$$

โดยดัชนีตัวแรก  $i$  คือ หมายเลขแถว มีค่าตั้งแต่ 0 ถึง  $N-1$  ดัชนีตัวที่สอง  $j$  คือ หมายเลขคอลัมน์ มีค่าตั้งแต่ 0 ถึง  $N-1$  ดังนั้น

$$A = \begin{pmatrix} A[0][0] & A[0][1] & \dots & A[0][N-1] \\ A[1][0] & A[1][1] & \dots & A[1][N-1] \\ \vdots & \vdots & \ddots & \vdots \\ A[N-1][0] & A[N-1][1] & \dots & A[N-1][N-1] \end{pmatrix}$$

เมื่อทำการเข้าใจพื้นฐานของเมทริกซ์ในรูปแบบของอาร์เรย์ 2 มิติแล้ว ผู้อ่านสามารถทำการทดลองตามขั้นตอนต่อไปนี้

1. สร้างไดเรกทอรี /home/pi/asm/Lab13 บนโปรแกรม Terminal ด้วยคำสั่งต่อไปนี้ตามลำดับ

```
$ cd /home/pi/asm
$ mkdir Lab13
```

```
$ cd Lab13
$ nano multMatrix.c
```

2. กรอกโปรแกรมต่อไปนี้ด้วยโปรแกรม nano และบันทึกในไฟล์ชื่อ **multMatrix.c** ในไดเรกทอรีที่สร้างไว้

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <omp.h>

#define N 200
float A[N][N], B[N][N], C[N][N]; // matrices of NxN elements

int main () {
/* DECLARING VARIABLES */
int i, j, k; // indices for matrix multiplication
double t_mul; // Multiply time
double start, stop; // start time and stop time

/* FILLING MATRICES WITH RANDOM NUMBERS */
srand ( time(NULL) );
for(i=0;i<N;i++) {
    for(j=0;j<N;j++) {
        A[i][j]= rand();
        B[i][j]= rand();
    }
}
/* MATRIX MULTIPLICATION */
printf("Max number of threads: %i \n",omp_get_max_threads());
#pragma omp parallel
printf("Number of threads: %i \n",omp_get_num_threads());
start=omp_get_wtime(); // time measure: start time
#pragma omp parallel for private(k, j)
for(i=0;i<N;i++) {
    for(j=0;j<N;j++) {
        C[i][j]=0; // set initial value of resulting matrix C = 0
        for(k=0;k<N;k++) {
            C[i][j]=C[i][j]+A[i][k]*B[k][j];
        }
    }
}
```

```

stop=omp_get_wtime(); // time measure: stop time
t_mul = stop-start; // Multiply time
printf("Mutiply Time: %2.4f \n",t_mul);

/* TERMINATE PROGRAM */
return 0;
}

```

3. exit ออกจากโปรแกรม nano เพื่อคอมpile โปรแกรมด้วยคำสั่งต่อไปนี้

```
$ gcc -fopenmp multMatrix.c -o mulMatrix
```

แก้ไขหากมีข้อผิดพลาดจนกว่าจะคอมpile โปรแกรมสำเร็จและมีไฟล์ชื่อ mulMatrix

4. ตั้งค่าจำนวนเรตติ้ง  $n=1$  ของโปรแกรมด้วยคำสั่งต่อไปนี้

```
$ export OMP_NUM_THREADS=1
```

5. รันโปรแกรมจับเวลาด้วยคำสั่ง time ดังนี้จำนวน 5 ครั้งเพื่อหาค่าเฉลี่ย ขณะทำการทดลองขอให้ผู้อ่านใช้คลีอฟข้อมือจับเวลาไปพร้อมๆ กัน เพื่อเปรียบเทียบกับค่าของ  $T_{mul,n}$  และ  $T_{real}$

```
$ time ./mulMatrix
```

ซึ่งจะรายงานผลการจับเวลาการทำงานของทั้งโปรแกรมในแต่ละมุ่งต่างๆ

6. จดบันทึกค่า CPU Utilization สูงสุดหรือ  $\%CPU_{max}$  ที่สังเกตได้ หากค่าเฉลี่ยของ  $T_{mul,n}$   $T_{real}$   $T_{user}$  และ  $T_{sys}$  ที่ได้เป็นวินาทีลงในตารางที่ M.1

ตารางที่ M.1: เวลาและ  $\%CPU_{max}$  ของการคูณเมทริกซ์ขนาด  $N$  และจำนวนเรตเท่ากับ 1, 2, 4, 8 เรต

| เวลาเฉลี่ย<br>(วินาที) | $N=200$<br>(วินาที) | $N=400$<br>(วินาที) | $N=800$<br>(วินาที) | $N=1000$<br>(วินาที) |
|------------------------|---------------------|---------------------|---------------------|----------------------|
| $n=1$ เรต              |                     |                     |                     |                      |
| $T_{mul,1}$            |                     |                     |                     |                      |
| $T_{real}$             |                     |                     |                     |                      |
| $T_{user}$             |                     |                     |                     |                      |
| $T_{sys}$              |                     |                     |                     |                      |
| $\%CPU_{max}$          |                     |                     |                     |                      |
| $n=2$ เรต              |                     |                     |                     |                      |
| $T_{mul,2}$            |                     |                     |                     |                      |
| $T_{real}$             |                     |                     |                     |                      |
| $T_{user}$             |                     |                     |                     |                      |
| $T_{sys}$              |                     |                     |                     |                      |
| $\%CPU_{max}$          |                     |                     |                     |                      |
| $n=4$ เรต              |                     |                     |                     |                      |
| $T_{mul,4}$            |                     |                     |                     |                      |
| $T_{real}$             |                     |                     |                     |                      |
| $T_{user}$             |                     |                     |                     |                      |
| $T_{sys}$              |                     |                     |                     |                      |
| $\%CPU_{max}$          |                     |                     |                     |                      |
| $n=8$ เรต              |                     |                     |                     |                      |
| $T_{mul,8}$            |                     |                     |                     |                      |
| $T_{real}$             |                     |                     |                     |                      |
| $T_{user}$             |                     |                     |                     |                      |
| $T_{sys}$              |                     |                     |                     |                      |
| $\%CPU_{max}$          |                     |                     |                     |                      |

7. เปลี่ยนจำนวนเรตเท่ากับ  $n=2$  เรต ด้วยคำสั่งต่อไปนี้

```
$ export OMP_NUM_THREADS=2
```

แล้ววนกลับไปทำข้อ 5 เพื่อกรอกค่าเฉลี่ยเวลาในตารางที่ M.1 จนครบ แล้วจึงเปลี่ยนจำนวนเรต  $n=4$  และ 8 เรต

8. เปลี่ยนขนาดข้อมูล  $N=400$  แล้วกลับไปเริ่มทำข้อ 3 จนถึงข้อ 8 จนครบ  $N=800$  และ 1000

จากตารางที่ M.1 ผู้อ่านสามารถใช้ประกอบการคำนวณประสิทธิภาพการคำนวณแบบขนาดในหัวข้อถัดไป

### M.3 ความซับซ้อน (Complexity) ของการคำนวณ

ความซับซ้อนเชิงเวลา (Run Time Complexity) ของการคูณเมทริกซ์เท่ากับ  $O(N^3)$  ในทางทฤษฎี ผู้อ่านสามารถประยุกต์ใช้อัตราส่วนระหว่าง  $O(N_2^3)$  และ  $O(N_1^3)$  ที่ภาระงานขนาด  $N_2:N_1$  และจำนวน  $n$  เฮอดเมื่อกัน เพื่อวัดความซับซ้อนของอัลกอริธึมได้ดังสมการต่อไปนี้

$$\frac{O(N_2^3)}{O(N_1^3)} = \frac{T_{mul,N_2}}{T_{mul,N_1}} \quad (\text{M.2})$$

สำหรับการคูณเมทริกซ์  $T_{mul,N}$  คือ เป็นระยะเวลาเฉลี่ยของการคูณเมทริกซ์ขนาด  $N \times N$  ด้วยจำนวน  $n$  เฮอด จากหัวข้อที่ผ่านมา ผู้อ่านสามารถคำนวณค่าอัตราส่วนของเวลาในตารางที่ [M.2](#) เพื่อใช้วิเคราะห์ต่อไป

ตารางที่ M.2: อัตราส่วนเวลาการคูณเมทริกซ์ที่ขนาด  $N$  และเวลาที่ขนาด 200 ที่จำนวนヘอดเท่ากับ 1, 2, 4, 8 เฮอด จากสมการที่ [\(M.2\)](#)

|                                       | $N=200$                           | $N=400$ | $N=800$ | $N=1000$ |
|---------------------------------------|-----------------------------------|---------|---------|----------|
| $n=1$ เฮอด<br>$T_{N,1}/T_{200,1}$     | 1.00                              |         |         |          |
|                                       | $\sqrt[2]{T_{N,1}/T_{200,1}}$     | 1.00    |         |          |
|                                       | $\sqrt[3]{T_{N,1}/T_{200,1}}$     | 1.00    |         |          |
| $n=2$ เฮอด<br>$T_{mul,N}/T_{mul,200}$ | 1.00                              |         |         |          |
|                                       | $\sqrt[2]{T_{mul,N}/T_{mul,200}}$ | 1.00    |         |          |
|                                       | $\sqrt[3]{T_{mul,N}/T_{mul,200}}$ | 1.00    |         |          |
| $n=4$ เฮอด<br>$T_{mul,N}/T_{mul,200}$ | 1.00                              |         |         |          |
|                                       | $\sqrt[2]{T_{mul,N}/T_{mul,200}}$ | 1.00    |         |          |
|                                       | $\sqrt[3]{T_{mul,N}/T_{mul,200}}$ | 1.00    |         |          |
| $n=8$ เฮอด<br>$T_{mul,N}/T_{mul,200}$ | 1.00                              |         |         |          |
|                                       | $\sqrt[2]{T_{mul,N}/T_{mul,200}}$ | 1.00    |         |          |
|                                       | $\sqrt[3]{T_{mul,N}/T_{mul,200}}$ | 1.00    |         |          |

## M.4 ประสิทธิภาพ (Performance) ของการคำนวณแบบบานด์

ผู้อ่านสามารถวัดประสิทธิภาพ (Performance) ของอัลกอริธึมใดๆ ได้จากอัตราส่วนของเวลาเดิม ( $T_{old}$ ) และเวลาใหม่ ( $T_{new}$ ) ที่ได้ทำการปรับปรุงอัลกอริธึมนั้นๆ ที่มา: [Patterson and Hennessy \(2016\)](#)

$$\frac{Perf_{new}}{Perf_{old}} = \frac{T_{old}}{T_{new}} \quad (\text{M.3})$$

ดังนั้น ประสิทธิภาพของการคำนวณแบบบานด์สามารถวัดได้จากอัตราส่วนระหว่างระยะเวลา  $T_{alg,1}$  ของ 1 เฮρดและ  $T_{alg,n}$  ของ  $n$  เฮρด และตั้งชื่อเรียกว่า  $Speedup(n)$  ด้วยสมการท่อไปนี้

$$Speedup(n) = \frac{T_{alg,1}}{T_{alg,n}} \quad (\text{M.4})$$

โดย  $T_{alg,n}$  คือ ช่วงการรันโปรแกรมอัลกอริธึมด้วยจำนวน  $n$  เฮρด โดยไม่รวมช่วงเวลาอื่นๆ ซึ่งไม่ได้เกี่ยวข้องกับการอัลกอริธึมแบบบานด์ ผู้อ่านสามารถประยุกต์ตัวชี้วัดนี้กับอัลกอริธึมการคูณเมทริกซ์ ดังนี้

$$Speedup(n) = \frac{T_{mul,1}}{T_{mul,n}} \quad (\text{M.5})$$

โดย  $T_{mul,n}$  คือ ช่วงการรันโปรแกรมคำนวณเมทริกซ์จริงๆ ด้วยจำนวน  $n$  เฮρด ที่ขนาด  $N$  เท่ากันโดยไม่รวมช่วงเวลาสู่มค่าตั้งต้น และการแสดงผลอื่นๆ ผู้อ่านคำนวณค่า  $Speedup(n)$  และกรอกในตารางที่ [M.3](#) เพื่อวิเคราะห์ผลการคำนวณที่ได้โดยตอบคำถามในกิจกรรมท้ายการทดลอง

ตารางที่ M.3: ผลการคำนวณ  $Speedup(n)$  ของการคูณเมทริกซ์ที่ขนาด  $N$  และจำนวนເຮັດທைກັບ 2, 4, 8 ເຮັດເທື່ອບກັບ 1 ເຮັດດ້ວຍສາມາດທີ່ ([M.5](#))

| Speedup                                          | $N=200$ | $N=400$ | $N=800$ | $N=1000$ |
|--------------------------------------------------|---------|---------|---------|----------|
| $n=2$ ເຮັດ<br>$Speedup(2) = T_{mul,1}/T_{mul,2}$ |         |         |         |          |
| $n=4$ ເຮັດ<br>$Speedup(4) = T_{mul,1}/T_{mul,4}$ |         |         |         |          |
| $n=8$ ເຮັດ<br>$Speedup(8) = T_{mul,1}/T_{mul,8}$ |         |         |         |          |

## M.5 กิจกรรมท้ายการทดลอง

1. เหตุใดการทดลองจึงต้องใช้การหาค่าเฉลี่ยเวลาต่างๆ
2.  $T_{sys}$  หมายถึง เวลาซึ่งทำงานประเภทไหน
3.  $T_{user}$  หมายถึง เวลาซึ่งทำงานประเภทไหน
4.  $T_{real}$  มีความสัมพันธ์กับ  $T_{mul}$  อย่างไร
5.  $T_{user}$  มีความสัมพันธ์กับ  $T_{mul}$  และจำนวนเรต  $n$  อย่างไร
6. เมื่อ  $N_1=200$  จะเปรียบเทียบค่าผลการคำนวณของ  $\sqrt[2]{T_{mul,N_2}/T_{mul,200}}$  และ  $\sqrt[3]{T_{mul,N_2}/T_{mul,200}}$  ที่ได้ในตารางที่ M.2 เมื่อ  $N_2 = 400, 800$  และ  $1000$  และ  $n = 1, 2, 4$  และ  $8$  เฮดตตามลำดับ ว่ามีค่าใกล้เคียงกับ  $N_2/200 = 2, 4, 5$  ตามลำดับอย่างไร เพาะเหตุใด
7. จำนวนเรต และ จำนวนแกนประมวลผล มีผลต่อค่า Speedup อย่างไร วิเคราะห์ทั้งหมด 3 กรณีดังนี้
  - จำนวนเรต < จำนวนแกนประมวลผล
  - จำนวนเรต = จำนวนแกนประมวลผล
  - จำนวนเรต > จำนวนแกนประมวลผล
8. เหตุใดค่าเฉลี่ยเวลา  $T_{user}$  จึงไม่แตกต่างกัน ที่  $N$  คนที่
9. เวลาส่วนใหญ่ของการรัน  $T_{real}$   $T_{user}$  และ  $T_{sys}$  สัมพันธ์กันอย่างไร จงสร้างสมการ
10. จำนวนเรตที่เพิ่มขึ้นทำให้การคำนวนเร็วขึ้นอย่างไร มีข้อจำกัดหรือไม่
11. ที่ขนาดข้อมูล  $N=1000$  จำนวนเรตที่เพิ่มขึ้นทำให้  $T_{user}$  เปลี่ยนแปลงอย่างไร มีข้อจำกัดหรือไม่
12. ที่ขนาดข้อมูล  $N$  ต่างๆ ค่า  $\%CPU_{max}$  มีการเปลี่ยนแปลงและมีความสัมพันธ์กับจำนวนเรต  $n$  อย่างไร
13. ขนาดข้อมูล  $N$  ที่เพิ่มขึ้นมีผลต่อ  $Speedup(n)$  ที่  $n=1, 2, 4$  และ  $8$  หรือไม่ อย่างไร

# ເອກສາຣ້ອງຈິງ (Bibliography)

- Allwinner Technology, C. (2015a). Allwinner a64 mobile application processor datasheet version 1.1.
- Allwinner Technology, C. (2015b). Allwinner a64 user manual version 1.0.
- Anderson, T. and M. Dahlin (2012). *Principles and Practice (Second Edition)* (2nd ed.). USA: Recursive Books, Ltd.
- ARM (2011). *Cortex-A7 MPCore Revision: Technical Reference Manual* (r0p3 ed.). UK: ARM.
- Broadcom, C. (2012). Bcm2835 arm peripherals.
- Choi, K. (2010). Nand flash memory.
- Cypress Semiconductor, C. (2002). 256k(32kx8) static ram cy62256.
- Cypress Semiconductor, C. (2017). Single-chip ieee 802.11ac b/g/n mac/baseband/radio with integrated bluetooth 4.1 and fm receiver.
- Demblon, S. and S. Spitzner (2004). *Linux Internals: (to the power of -1)* (1 ed.). The Shuttleworth Foundation.
- Diodes, I. (2012). Pam2306 dual high-efficiency pwm step-down dc-dc converter.
- Harris, D. and S. Harris (2013). *Digital Design and Computer Architecture* (1st ed.). USA: Morgan Kauffman Publishing.
- Hillis, W. D. and G. L. Steele (1986, December). Data parallel algorithms. *Commun. ACM* 29(12), 1170–1183.
- John, S. (2020). The future of computing beyond moore’s law. *Philosophical Transactions of The Royal Society A*.
- Ltd., A. (2017). *ARM Generic Interrupt Controller Architecture Specification GIC architecture version 3.0 and version 4.0* (1 ed.). UK: ARM.
- Mano, M. M. and C. Kime (2007). *Logic and Computer Design Fundamentals* (4th ed.). USA: Prentice Hall Press.
- Microchip Technology, I. (2009). Lan9514/lan9514i usb 2.0 hub and 10/100 ethernet controller.

- Micron Technology, I. (2004). Nand flash memory: Mt29f2g08aabwp/mt29f2g16aabwp/mt29f4g08babwp/mt29f4g16babwp/mt29f8g08fabwp.
- Micron Technology, I. (2014). Embedded lpddr2 sdram edb8132b4pb-8d-f.
- Patterson, D. A. and J. L. Hennessy (2016). *Computer Organization and Design: The Hardware Software Interface ARM Edition*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.
- Ranokpanuwat, R. and S. Kittitornkun (2016). Parallel partition and merge quicksort (ppmqsort) on multicore cpus. *Journal of Supercomputing* 72(3), 1063–1091.
- Raspberry Pi (Trading), L. (2019). *Raspberry Pi Compute Module 3+ and Raspberry Pi Compute Module 3+ Lite* (1 ed.). Raspberry Pi (Trading) Ltd.
- Rattanantranurak, A. and S. Kittitornkun (2020). A multistack parallel (msp) partition algorithm applied to sorting. *Journal of Mobile Multimedia*, 293–316.
- Rattanantranurak, A. and S. Kittitornkun (2021). Optimizing multistack parallel (msp) sorting algorithm. *Journal of Mobile Multimedia* 17(4), 723–748.
- Rosen, K. H. (2002). *Discrete Mathematics and Its Applications* (5th ed.). McGraw-Hill Higher Education.
- SanDisk Corporation, I. (2003). Sandisk secure digital card, product manual version 1.9 document no. 80-13-00169.
- Tanenbaum, A. S. and T. Austin (2012). *Structured Computer Organization* (6 ed.). Boston, MA: Pearson.
- Tanenbaum, A. S. and H. Bos (2014). *Modern Operating Systems* (4 ed.). Boston, MA: Pearson.

# อภิธานศัพท์ (Glossary)

**GHz** ย่อจาก Giga Hertz หรือ กิกะเอิรตซ์ มีค่าฐานสิบเท่ากับ  $10^9$  เท่ากับ 1,000,000,000 เอิรตซ์ ที่มา: [wikipedia](#) หน้า. 59

**GiB** ย่อจาก GibiByte อ่านว่า กิบไบต์ มีค่าฐานสิบเท่ากับ  $2^{30}$  เท่ากับ  $(1024)^3$  เท่ากับ 1,073,741,824 ไบต์ ที่มา: [wikipedia](#) หน้า. 58, 129, 199, 223

**KiB** ย่อจาก KibiByte อ่านว่า คิบไบต์ มีค่าฐานสิบเท่ากับ  $2^{10}$  เท่ากับ  $(1024)^1$  เท่ากับ 1,024 ไบต์ ที่มา: [wikipedia](#) หน้า. 3, 90, 128, 200

**Mbps** ย่อจาก Mega bits per second หรือ เมกกะบิตต่อวินาที มีค่าฐานสิบเท่ากับ  $10^6$  เท่ากับ 1,000,000 บิตต่อวินาที ที่มา: [wikipedia](#) หน้า. 59, 162

**MiB** ย่อจาก MebiByte อ่านว่า เมบิไบต์ มีค่าฐานสิบเท่ากับ  $2^{20}$  เท่ากับ  $(1024)^2$  เท่ากับ 1,048,576 ไบต์ ที่มา: [wikipedia](#) หน้า. 60, 128, 133, 214

**SoC** ย่อจาก System on Chip คือ การย่อส่วนระบบลงบนชิปเดียวกัน เพื่อลดขนาดโดยรวมของระบบ, หน้า. 4, 59

**คอมพิวเตอร์** เครื่องจักรไฟฟ้าซึ่งประกอบด้วย ฮาร์ดแวร์หรือวงจรอิเล็กทรอนิกส์ซึ่งทำงานตามสัญญาณนาฬิกา ความถี่สูง ทำให้สามารถปฏิบัติตามซอฟต์แวร์หรือคำสั่งภาษาเครื่อง และข้อมูลในรูปของเลขฐานสอง ฮาร์ดแวร์ทำหน้าที่ประมวลข้อมูลจากอุปกรณ์อินพุตทำหน้าที่รับข้อมูลตามที่คำสั่งภาษาเครื่องเขียนไว้จนได้ผลลัพธ์เพื่อแสดงผลทางอุปกรณ์เอาต์พุต, หน้า. 1

**ซอฟต์แวร์** ส่วนประกอบสำคัญของเครื่องคอมพิวเตอร์ ประกอบด้วยคำสั่งภาษาเครื่องและข้อมูลซึ่งใช้เลขฐานสองเป็นหลัก แบ่งเป็นซอฟต์แวร์ระบบและซอฟต์แวร์ประยุกต์, หน้า. 1

**ลอกิคเกต** เป็นส่วนประกอบพื้นฐานของวงจรดิจิทัล ภายในประกอบด้วยทรานซิสเตอร์ชนิด MOSFET (Metal Oxide Semiconductor Field Effect Transistor) หรือชนิดอื่น จำนวนหนึ่งเข้มต่อกันเพื่อให้ทำงานเป็นสมการบูลลีน (Boolean) ชนิดต่างๆ เช่น NOT เกต, AND เกต, NOR เกต เป็นต้น, หน้า. 5

**ฮาร์ดแวร์** ส่วนประกอบสำคัญของเครื่องคอมพิวเตอร์ ประกอบด้วยวงจรอิเล็กทรอนิกส์ แบ่งเป็น ส่วนประมวลผล (Processor) หน่วยความจำภายใน วงจร/อุปกรณ์อินพุตและเอาต์พุต และอุปกรณ์เก็บรักษาข้อมูล ทั้งหมดนี้ควบคุมให้ทำงานตามซอฟต์แวร์หรือคำสั่งภาษาเครื่องและข้อมูล, หน้า. 1

# ดัชนี (Index)

- 2's Complement, 16, 23, 32  
A2D, 158  
Access Time, 87, 123, 125, 140, 192, 206  
Access Time, Average, 203  
Address, 11, 12, 52, 89, 140, 299  
ALU, IEEE754, 44, 122, 147  
ALU, Integer, 25, 29, 95, 122, 147  
Android, 1  
ARM, 116, 211  
ASCII, 10, 50, 81, 89, 186, 239, 287  
Assembler, 65, 278  
  
BCM2835, 3, 126  
BCM2836, 3, 57  
BCM2837, 3, 56, 85, 138, 160, 242  
big.LITTLE, 211  
BL: Branch and Link, 88, 113  
Bluetooth, 57, 164  
Boot Loader, 73  
Boot OS, 73, 261, 304  
Branch, 69, 99  
Break Point, 90, 269, 278, 296  
BX LR, 69, 113, 115  
byte, 89, 91, 285, 286  
  
Cache Coherent, 183, 212  
Carry bit, 25, 95, 100  
char, 9, 50, 89  
char, unsigned, 9, 89  
Chip, 4, 57, 116, 211  
CMP: Compare, 69, 99  
Command Line, 259, 294  
CPSR, 69, 95, 99, 296  
  
CPU Utilization, 339  
CPU: Central Processing Unit, 3, 58, 147, 202  
CSI: Camera Serial Interface, 57, 156  
  
D2A, 160  
DAC, 160  
Debug, 267, 275, 294  
Decode, 86  
Decode Instruction, 72, 86  
Denormalize, IEEE754, 43, 44  
Device Driver, 80, 301, 309  
Directory, 74, 189, 190, 334, 336  
Disassembly, 89  
Divide and Conquer, 218  
DMA: Direct Memory Access, 82, 167, 179, 188, 207  
DMAC, 179  
DO-WHILE, 110  
double, 9, 39, 43  
Double Precision, IEEE754, 39, 40  
doubleword, 89  
DSI: Display Serial Interface, 57, 154  
  
ELF, 66, 67, 75, 124, 272  
EOF, 81, 186  
Ethernet, 57, 163, 314  
Execute Instruction, 72, 88, 130, 295  
Execute Permission, 190, 337  
Exponent, IEEE754, 35, 40, 233  
  
Fetch Instruction, 72, 78, 86, 88, 130  
File System, 81, 186  
Flash Memory, 63, 81, 123, 192  
float, 9, 39, 43

FOR, 106  
Fork-Join, 217  
Format Partition, 188  
Full Adder, 26, 32  
Function, 79, 112, 289  
  
GiB, 57, 78, 125, 149  
go-to, 99  
GPIO, 57, 74, 167, 170, 183, 318, 327  
GPU, 4, 59, 305  
  
halfword, 285  
HDMI, 57, 303  
Heap Segment, 79  
hword, 89, 285  
  
I2S, 159  
IEEE754, 11, 39, 43, 87  
IF, 101  
IF-ELSE, 103  
Immediate, 71, 93  
Inode, 189, 334  
int, 9, 89  
int, unsigned, 89  
Interrupt, 62, 149, 175, 207  
Interrupt Priority, 176  
Interrupt Request, 175, 176  
IoT, 2, 52, 56, 148, 163, 303  
ISR: Interrupt Service Routine, 175, 327, 328  
  
Java Bytecode, 119  
Jump, 99  
  
Kernel Space, 76, 78, 125, 126  
  
Label, 67, 99  
Latency, 146, 175, 206  
LCD, 62, 150, 169  
Library, 66, 69, 74, 322  
Linker, 65, 69, 270, 321  
Load/Store, 80, 87, 91  
Load/Store Architecture, 80, 87, 119  
long long, 9  
  
long long, unsigned, 9, 89  
Loop, 26, 33, 106, 108, 110, 272, 273  
LR: Link Register, 88, 289, 299  
  
Machine Code, 89  
main, 67, 68, 78, 114  
main(), 68, 114, 219, 282  
Makefile, 66, 270, 279, 283, 295, 329  
makefile, 66, 270, 279, 283, 295, 329  
Many-Core CPU, 209  
Memory Mapped File, 82, 186, 207  
Memory Mapped I/O, 166  
MergeSort, 219  
Metadata, 191  
Moore's Law, 209  
Mount, 75, 336  
Multi-Core CPU, 209  
Multicomputer, 211  
Multitasking, 124  
  
NaN, IEEE754, 43  
Negative bit, 25, 95, 100  
Non-Volatile Memory, 123  
Normalize, IEEE754, 9, 35  
NZCV, 95, 99  
  
Object File, 70, 76, 278, 279  
Opcode, 71, 89  
OpenMP, 339  
Overflow, IEEE754, 44, 46  
Overflow, Integer, 11, 25, 95  
  
Parallel Computing, 148  
Parallel Region, 217  
Parallelism, 214  
Parallelism, Data, 214  
Parallelism, Task, 214  
Parameter, 88  
Partition, 187  
PC: Program Counter, 88, 99, 289  
PCM, 158  
Permission, 188, 190, 338  
Physical Address, 124, 125

- Physical Memory, 123, 147  
Pipeline, 44, 86  
Pointer, 12, 69  
Pop, 79, 92, 299  
Process, 124  
Program Counter, 129  
Push, 92, 119, 289, 299
- QuickSort, 219
- Refresh, 146  
Register, 147  
RISC, 119
- SATA, 205, 208  
Scalar, 86  
Scan Code, 60  
SDRAM, 60  
Segment, Heap, 79  
Segment, Stack, 79  
Segment, Text, 90  
short, 9, 10, 20  
Shut Down, 73, 83  
Sign Bit, IEEE754, 35, 233  
Sign Bit, Integer, 16, 23, 34, 35, 40  
Sign-Magnitude, 23  
Signed Integer, 16, 23  
Significand, IEEE754, 35, 233  
SIMD, 118  
Single Precision, 39, 40, 42, 43, 232, 236  
Smartphone, 116  
SMP: Shared Memory Multiprocessor, 211  
SoC, 4, 57  
Socket Interface, 80  
Software, 147  
Source Code, 77  
SP: Stack Pointer, 79, 88, 92, 299  
Speedup, 339  
Stack Machine, 119  
Stack Pointer, 79, 289  
Stack Segment, 79, 92  
Storage, 147
- String, 50, 287  
string, 50, 287  
Super User, 83  
SuperScalar, 209  
Swap Partition, 125  
System Call Interface, 77  
System Software, 64
- Text Segment, 90  
Thumb, 118  
Time, Access, 140  
TLB: Translation Lookaside Buffer, 128, 167  
Top of Stack, 299
- UCS-2, 51  
Underflow, IEEE754, 44, 46  
Unicode, 51, 239  
unsigned char, 9, 89  
unsigned int, 9, 89  
Unsigned Integer, 11, 12, 23, 26, 28, 272  
unsigned long, 11  
unsigned short, 9, 13  
UPS, 123  
USB, 57, 61, 150, 161, 167, 311  
User Space, 77, 78  
UTF-16, 52  
UTF-8, 51
- Variable, 147  
Variable, Global, 91, 282  
Variable, Local, 282, 299  
Vector, 52, 87  
Virtual Address, 87, 124, 130  
Virtual File System, 336  
Volatile Memory, 123
- WHILE, 108  
WiFi, 57, 164, 314  
wiringPi, 318, 327  
word, 89, 91, 285  
Worker Thread, 339
- Zero, bit, 25, 95

- Zero, IEEE754, 43, 234
- การฟอร์แมท, 188
- กิบีไบต์, 57, 78, 125, 149
- ความขนาดของข้อมูล, 214
- ความขนาดของงานย่อย, 214
- ความถี่สูง, 158
- คอมมานด์ไลน์, 259, 294
- คอมไฟเลอร์, 65, 267
- คำนวณแบบบานาน, 148
- ค่าনัยสำคัญ, IEEE754, 35
- ค่ายกกำลัง, IEEE754, 35, 40
- ชัตดาวน์, 83, 185
- ชีป, 4, 57, 116, 211
- ซอฟต์แวร์, 64
- ซอฟต์แวร์ประยุกต์, 55, 56
- ซอฟต์แวร์ระบบ, 64
- ซีพียูมัลติคอร์, 209, 211
- ซีพียูแมเนคอร์, 209, 211
- ซูเปอร์สเกลาร์, 209
- ดาต้าเซ็กเมนต์, 67, 76, 282
- ดีนอร์มัลໄල์, IEEE754, 43
- ดีไวซ์ไดเรเวอร์, 80, 188, 314
- ตัวแปร, 9, 80, 87–89, 91, 147
- ถอดรหัส, 72, 86
- ทราบชิสเตอร์, 4, 5, 138, 145
- นอร์มัลໄල์, IEEE754, 9, 35, 36, 45, 49
- บิตเครื่องหมาย, IEEE754, 35
- บูตโหลดเดอร์, 73
- พารามิเตอร์, 88
- พาร์ทิชัน, 187
- พังก์ชัน, 65, 70, 79, 112
- ภาษาเครื่อง, 64, 65, 67, 71, 76, 89
- มัลติคอมพิวเตอร์, 211, 212
- มัลติทาสกิ้ง, 124
- มัลติเรต, 209, 212, 216
- ระบบไฟล์, 81, 186, 188
- ระบบไฟล์เสมือน, 74
- รีจิสเตอร์, 25, 72, 80, 86, 88, 147
- ลอจิกเกต, 4, 5
- ลิงก์เกอร์, 65, 270
- วงจรรวม, 4, 116, 211
- สถาปัตยกรรมโหลด/สโตร์, 80, 87, 119
- สมาร์ตโฟน, 3, 116, 127
- ซอฟพาร์ทิชัน, 125
- สเกลาร์, 87
- สแกนโค้ด, 60
- สแต็ก, 88, 92, 289, 299
- สแต็กเซ็กเมนต์, 79, 92, 299
- สแต็กเฟรม, 79, 299
- สแต็กแมชชีน, 119
- หน่วยความจำภายใน, 123, 125, 147, 188
- หน่วยความจำหลัก, 123, 125
- อອพโค้ด, 71, 89
- อันเดอร์ไฟล์, IEEE754, 44, 48
- อาร์เรย์, 50, 69, 91, 131, 285, 286
- อินพินิตี, IEEE754, 43
- อินเทอร์รัปท์, 62, 149, 175, 207
- อุปกรณ์เก็บรักษาข้อมูล, 123, 142, 147
- อ็อบเจกต์, 70, 270, 278
- อีปเซ็กเมนต์, 79
- เครื่องข่ายบันชีป, 214
- เคอร์เนล, 74, 77, 80
- เซ็กเมนต์, ดาต้า, 76, 282
- เซ็กเมนต์, สแต็ก, 79, 92
- เทิกซ์เซ็กเมนต์, 67, 76
- เฟทซ์คำสั่ง, 78, 86, 88, 130
- เลเบล, 67, 93, 94, 99, 282, 297
- เวกเตอร์, 87
- เวลารอคอย, 175
- เวลาเข้าถึง, 87, 123, 125, 138, 140, 192
- เวลาเข้าถึงเฉลี่ย, 203
- เวอร์ชวลเม莫รี, 121, 137, 186
- เวอร์ชวลแอดเดรส, 87, 124, 130, 167
- แท็บเล็ต, 3
- แอดเดรส, 11, 12, 52, 89, 140, 299
- แอดเดรสภายใน, 124, 125, 166, 167
- แօสเซมเบลอร์, 65, 278
- โปรแกรมเคาน์เตอร์, 88, 99, 129, 278, 289, 296
- โพรเชส, 124
- ไอເວອຣິໄຟລ໌, IEEE754, 44, 48

- โอลเวอร์ไฟล์ว์, มีเครื่องหมาย, 32, 33, 273
- โอลเวอร์ไฟล์ว์, เลขจำนวนเต็ม, 25, 95, 100
- โอลเวอร์ไฟล์ว์, ไม่มีเครื่องหมาย, 26, 272
- ไดเรกทอรี, 74, 189, 190, 334, 336
- ไบต์, 9
- ไบป์ไลน์, 31, 86, 129, 209
- ไฟล์, 81, 147, 189, 190
- ไฟล์อ็อบเจกต์, 66, 69, 70, 76, 269, 278, 279
- ไลบรารี, 66, 69, 70, 74, 166, 327
- ไอโอนด, 189, 334