

องค์ประกอบคอมพิวเตอร์และภาษาแอสเซมบลี: กรณีศึกษา

บอร์ด Raspberry Pi

รศ.ดร.สุรินทร์ กิตติกรกุล
ภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

คำนำ (Preface)

ตำรา เล่มนี้ใช้ประกอบ การเรียน การสอน วิชา Computer Organization and Assembly Language ตามหลักสูตรวิศวกรรมคอมพิวเตอร์ ภาควิชาวิศวกรรมคอมพิวเตอร์ (www.ce.kmitl.ac.th) สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง โดยใช้คอมพิวเตอร์บอร์ด เดียวชื่อ Raspberry Pi3/Pi4 เป็นกรณีศึกษาหลัก ซึ่งภายในชิปประกอบด้วย ชิปเซ็ต Cortex A53/A72 จากบริษัท ARM เป็นชิปเซ็ตมัลติคอร์ จำนวน 4 แกน ประมวลผลต่อชิป ผู้เรียนสามารถพื้นฐานวิชาออกแบบจรดจิทัล และตระกราก และวิชาการเขียนโปรแกรมคอมพิวเตอร์โดยเฉพาะภาษา C/C++ เนื้อหาในตำราเล่มนี้สามารถนำไปประยุกต์ใช้กับภาษา C ภาษาไพธอน (Python) หรือ บอร์ด Raspberry Pi3/Pi4 ได้ และสามารถใช้เป็นพื้นฐานของวิชาลำดับถัดไป เช่น สถาปัตยกรรม หรือระบบคอมพิวเตอร์ ระบบปฏิบัติการ คอมพิวเตอร์ เป็นต้น ผู้เขียนจะเน้นการใช้งานเลขยกกำลังสองและเลขฐานสิบ หากในตำราเล่มนี้ เพื่อให้สอดคล้องกับการทำงานของคอมพิวเตอร์

ตารางที่ 1: ตารางเปรียบเทียบตัวคูณด้วยค่าสองยกกำลังและสิบยกกำลัง

สองยกกำลัง	อ่านว่า	ค่าฐานสิบ	สิบยกกำลัง	อ่านว่า	ค่าฐานสิบ
2^{10}	คิบี (kibi)	1024	10^3	กิโล (kilo)	1000
2^{20}	เมบี (mebi)	1024^2	10^6	เมกะ (mega)	1000^2
2^{30}	กิบี (gibi)	1024^3	10^9	กิกะ (giga)	1000^3

เนื้อหาในตำราครอบคลุม ทฤษฎี และปฏิบัติ โดยมีภาค ผนวก เป็นการทดลองทั้งหมด 13 การทดลองโดยใช้บอร์ดตระกูล Raspberry Pi แต่ละทมีการสรุป/คำตามท้ายบท และกิจกรรมท้ายการทดลองเสริมความเข้าใจ ผู้อ่านสามารถค้นคว้าศึกษาเพิ่มเติม เนื้อหา ทฤษฎี และปฏิบัติ เพิ่มเติม โดยการคลิกลิงก์ (ตัวอักษรสีน้ำเงิน) ไปยัง เว็บไซต์/เว็บเพจ/คลิป/รูปภาพ ในอินเทอร์เน็ต เลขบท/ภาค ผนวก เลขรูป เลขตาราง เลขหัวข้อ/นิยาม/ตัวอย่าง/สมการ เอกสารอ้างอิง ที่ผู้เขียนอ้างอิงถึงได้ สะดวกรวดเร็ว และกดปุ่มถอยกลับ (Back) เพื่อย้อนกลับมาที่เนื้อหาปัจจุบันได้ ทั้งนี้ การใช้งานขึ้นอยู่กับซอฟต์แวร์ที่ใช้เปิดอ่านไฟล์ PDF ที่ใช้

ผู้เขียนแนะนำเครื่องคอมพิวเตอร์ชนิดต่างๆ ในบทที่ 1 เพื่อเชื่อมโยงกับเครื่องคอมพิวเตอร์ชนิดบอร์ดเดียว (Single Board Computer) ชื่อ Raspberry Pi รุ่นต่างๆ บทที่ 2 วางแผนพื้นฐานด้านข้อมูล/ตัวแปรชนิดต่างๆ และคณิตศาสตร์ของเลขฐานสอง ในเครื่องคอมพิวเตอร์ เพื่อเชื่อมโยงกับการพัฒนาโปรแกรม ด้วยภาษา C/C++ บทที่ 3 อธิบายโครงสร้างด้านฮาร์ดแวร์และซอฟต์แวร์ ด้านฮาร์ดแวร์เน้นที่ชิปเซ็ต หน่วยความจำหลัก อินพุต/เอาต์พุต และอุปกรณ์เก็บรักษาข้อมูล ด้านซอฟต์แวร์ อธิบายการทำงานของระบบปฏิบัติ

การของบอร์ด Raspberry Pi3/Pi4 โมเดล B ด้วยวิธีบนลงล่าง (Top Down) เพื่อ เชื่อมโยงการพัฒนา ซอฟต์แวร์ด้วยภาษาและสเซมบลีของซีพียู ARM ในบทที่ 4 ซึ่งจำกัดเฉพาะคำสั่งภาษาและสเซมบลี (Assembly Instruction) และภาษาเครื่องความยาว 32 บิต และระบบปฏิบัติการ Raspberry Pi OS เวอร์ชัน 32 บิต บทที่ 5 เชื่อมโยงการทำงานของเวอร์ชัลเม莫รีสำหรับบอร์ด Pi3/Pi4 พื้นฐานของแคชชนิด Direct Map และชนิด N-Way Set Associative หน่วยความจำสแต็ติก แรม และหน่วยความจำ DDR2 SDRAM บทที่ 6 นำเสนอ alike ของอุปกรณ์และวงจรอินพุต/เอาต์พุตที่สำคัญของบอร์ด Pi3/Pi4 โดยเฉพาะขาซีพียูชนิด General Purpose Input Output (GPIO) เพื่อ เชื่อมโยงกับหลักการ Memory Mapped I/O การขัดจังหวะหรืออินเทอร์รัปต์ (Interrupt) และการเข้าถึงหน่วยความจำโดยตรง หรือ Direct Memory Access ซึ่งจะทำงานประสานกับระบบไฟล์ของอุปกรณ์เก็บรักษาข้อมูลโดยละเอียดในบทที่ 7 ท้ายที่สุดคือ บทที่ 8 นำหลักการคำนวณแบบขนาดมาประยุกต์ใช้กับบอร์ด Pi3/Pi4 ด้วยภาษา C และไลบรารี OpenMP ซึ่งเป็นที่ยอมรับในเครื่องคอมพิวเตอร์แบบขนาดใหญ่ที่สุดในปัจจุบันจนถึงปัจจุบัน เปอร์คอมพิวเตอร์ชื่อ Fugaku ซึ่งภายใต้เครื่องประกอบด้วยซีพียู ARM ขนาด 64 บิต จำนวน 7.6 ล้านแกนประมาณผล

กิตติกรรมประกาศ (Acknowledgments)

ผู้เขียนขอแสดงความนับถือบุคคลเหล่านี้ที่ช่วยให้ตำราเล่มนี้สำเร็จ ดังนี้

- ผศ.ดร.ชัยวัฒน์ หนูทอง ภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์ สจล.
- ดร.รัฐศิลป์ รานอกภานุวัชร์ หลักสูตรวิศวกรรมคอมพิวเตอร์ วิทยาลัยนวัตกรรมด้านเทคโนโลยีและวิศวกรรมศาสตร์ มหาวิทยาลัยธุรกิจบัณฑิตย์
- ผศ.ดร.อภิสิทธิ์ รัตนาตรา楠 รักษา ภาควิชา วิทยาการ คอมพิวเตอร์ และสารสนเทศ คณะวิทยาศาสตร์ ประยุกต์ มหาวิทยาลัยเทคโนโลยีพระจอมเกล้าพระนครเหนือ และ
- นักศึกษาภาควิชาฯ หลาย ๆ รุ่น

ที่ เคยใช้ตำราเล่มนี้ประกอบ การเรียน การสอน ทั้งแบบบอ้อนไลท์ และออนไลน์ ผ่านทาง **ยูทูบ** แนะนำและขอเชิญ หากต้องการยังมีจุดผิดพลาด หลงเหลืออยู่ ผู้เขียนขอ น้อมรับไว้แต่เพียงผู้เดียว ทั้งนี้ผู้อ่านสามารถกรอกข้อความเพื่อขอความคิดเห็นได้ทางฟอร์มต่อไปนี้ <https://forms.gle/mNd3L67BC8Er8EML7> หรือเขียนอีเมลมาที่ surin.ki@kmitl.ac.th

สารบัญ (Contents)

คำนำ (Preface)	i
กิตติกรรมประกาศ (Acknowledgments)	ii
สารบัญตาราง (List of Tables)	vii
สารบัญรูปภาพ (List of Figures)	ix
1 บทนำ (Introduction)	1
1.1 ชนิดของเครื่องคอมพิวเตอร์	1
1.1.1 คอมพิวเตอร์ตั้งโต๊ะ (Desktop Computer)	1
1.1.2 คอมพิวเตอร์พกพา (Portable Computers)	2
1.1.3 คอมพิวเตอร์ฝังตัว (Embedded Computer)	2
1.1.4 คอมพิวเตอร์เซิร์ฟเวอร์หรือแม่ข่าย (Server Computer)	2
1.1.5 ชูเปอร์คอมพิวเตอร์ (Supercomputer)	3
1.2 แนวโน้มของจำนวนอุปกรณ์คอมพิวเตอร์ชนิดต่างๆ	3
1.3 อุปกรณ์ดิจิทัลที่ใช้ชิปปูที่ออกแบบโดยบริษัท ARM	4
1.4 คอมพิวเตอร์บอร์ดเดียว (Single Board) ตระกูล Raspberry Pi	5
1.5 ขั้นตอนการผลิตชิป (Chip) หรือวงจรรวม (Integrated Circuit)	7
1.6 สรุปท้ายบท	9
1.7 คำถามท้ายบท	9
2 ข้อมูลและเลขคณิตคอมพิวเตอร์ (Computer Data and Arithmetic)	11
2.1 ข้อมูลพื้นฐานชนิดต่างในภาษา C/C++	12
2.2 เลขจำนวนเต็มฐานสอง: รูปแบบและค่าฐานสิบ	15
2.2.1 ชนิดไม่มีเครื่องหมาย (Unsigned Integer)	15
2.2.2 ชนิดมีเครื่องหมาย (Signed Integer) แบบ 2's Complement	19
2.2.3 ชนิดมีเครื่องหมาย (Signed Integer) แบบ Sign-Magnitude	28

2.3	คณิตศาสตร์เลขจำนวนเต็มฐานสอง	30
2.3.1	ชนิดไม่มีเครื่องหมาย	31
2.3.2	ชนิดมีเครื่องหมายแบบ 2's Complement	37
2.4	เลขทศนิยมฐานสองชนิดจุดตรึง (Fixed Point)	39
2.5	เลขทศนิยมฐานสองชนิดจุดลอยตัว (Floating Point)	41
2.5.1	การบวกเลขทศนิยมฐานสองชนิดจุดลอยตัว	42
2.5.2	การคูณเลขทศนิยมฐานสองชนิดจุดลอยตัว	43
2.6	เลขทศนิยมฐานสองชนิดจุดลอยตัวมาตรฐาน IEEE754	45
2.6.1	รูปแบบมาตรฐาน IEEE754	45
2.6.2	ค่าสูงสุด ต่ำสุดและค่าอื่น ๆ ของมาตรฐาน IEEE754	49
2.6.3	การบวกเลขทศนิยมฐานสองชนิดจุดลอยตัวตาม IEEE754	51
2.6.4	การคูณเลขทศนิยมฐานสองชนิดจุดลอยตัวตาม IEEE754	54
2.7	ตัวอักษร (Character) และข้อความ (String)	56
2.8	สรุปท้ายบท	58
2.9	คำถามท้ายบท	59
3	ฮาร์ดแวร์และซอฟต์แวร์ของคอมพิวเตอร์ (Computer Hardware and Software)	63
3.1	ฮาร์ดแวร์ของเครื่องคอมพิวเตอร์: บอร์ด Pi3/Pi4	64
3.1.1	ซีพียู (CPU: ARM Cortex A53/A72)	66
3.1.2	หน่วยความจำหลัก (Main Memory)	67
3.1.3	อุปกรณ์อินพุต/เอาต์พุต (Input/Output Devices)	68
3.1.4	อุปกรณ์เก็บรักษาข้อมูล (Data Storage)	72
3.2	การพัฒนาซอฟต์แวร์หรือโปรแกรมคอมพิวเตอร์	73
3.2.1	โครงสร้างของชอร์สโค้ดโปรแกรมภาษา C/C++	73
3.2.2	การแปลซอร์สโค้ดภาษา C/C++ ให้กลายเป็นโปรแกรมคอมพิวเตอร์	74
3.2.3	โครงสร้างของชอร์สโค้ดภาษาแอสเซมบลีของซีพียู ARM	76
3.2.4	การรวมไฟล์อ็อบเจกต์ (Object) ด้วยลิงก์เกอร์ (Linker)	79
3.2.5	ตัวอย่างของคำสั่งภาษาเครื่อง (Machine Code) ของซีพียู ARM	80
3.3	การทำงานร่วมกันระหว่างฮาร์ดแวร์และซอฟต์แวร์ (Hardware - Software Operation)	83
3.3.1	การบูต (Boot) ระบบปฏิบัติการ	83
3.3.2	ระบบปฏิบัติการโหลดซอฟต์แวร์ประยุกต์จากไฟล์รูปแบบ ELF	86
3.3.3	ซีพียูเพทซ์คำสั่งภาษาเครื่องของซอฟต์แวร์ประยุกต์จากหน่วยความจำ	89
3.3.4	ซีพียูอ่าน (Load)/เขียน (Store) ข้อมูลในหน่วยความจำหลัก	91
3.3.5	ซีพียูใช้งานอินพุตและเอาต์พุตต่าง ๆ ตามคำสั่งของซอฟต์แวร์ประยุกต์	92

3.3.6	ชีพิญอ่าน/เขียนไฟล์ข้อมูลในอุปกรณ์เก็บรักษาข้อมูล	93
3.3.7	ผู้ใช้ชัตดาวน์ (Shut Down) ระบบปฏิบัติการ	94
3.4	สรุปท้ายบท	95
3.5	คำถามท้ายบท	95
4	ภาษาแอสเซมบลีของชีพิญ ARM ขนาด 32 บิต	97
4.1	โครงสร้างของชีพิญ ARM Cortex A53/A72	97
4.2	สถาปัตยกรรมชุดคำสั่ง (Instruction Set Architecture)	100
4.3	ตัวอย่างคำสั่งภาษาเครื่องในเท็กซ์เช็กเมนต์	102
4.4	คำสั่งประมวลและตั้งค่าเริ่มต้นของตัวแปรในดาต้าเซ็กเมนต์	104
4.5	คำสั่งถ่ายโอน (Transfer) ค่าตัวแปรระหว่างดาต้าเซ็กเมนต์และรีจิสเตอร์	105
4.6	คำสั่งประมวลผลข้อมูลในรีจิสเตอร์ (Register Data Processing Instructions)	108
4.6.1	คำสั่งทางคณิตศาสตร์ (Arithmetic Instructions)	108
4.6.2	คำสั่งเลื่อนบิตข้อมูล (Bit Shift Instructions)	110
4.6.3	คำสั่งทางคณิตศาสตร์และเลื่อนบิต (Arithmetic and Bit Shift Instructions)	111
4.6.4	คำสั่งทางตรรกศาสตร์ (Logical Instructions)	111
4.7	คำสั่งควบคุมการทำงาน (Control Instructions)	113
4.7.1	การตัดสินใจ IF	115
4.7.2	การตัดสินใจ IF-ELSE	118
4.7.3	การวนรอบชนิด FOR	121
4.7.4	การวนรอบชนิด WHILE	123
4.7.5	การวนรอบชนิด DO-WHILE	125
4.7.6	การวนรอบชนิด FOR จำนวน 2 ชั้น	127
4.8	การเรียกใช้ฟังก์ชัน (Function Call)	128
4.8.1	การเรียกใช้ฟังก์ชันในภาษา C/C++	128
4.8.2	คำสั่งเรียกใช้ฟังก์ชันในภาษาแอสเซมบลี	128
4.9	คำสั่งภาษาแอสเซมบลีอื่น ๆ ในชีพิญ ARM	132
4.10	สถาปัตยกรรมชุดคำสั่งภาษาแอสเซมบลีในชีพิญทั่วโลก	133
4.11	สรุปท้ายบท	133
4.12	คำถามท้ายบท	134
5	ลำดับชั้นของหน่วยความจำ (Hierarchy of Memory)	135
5.1	ลำดับชั้นของหน่วยความจำของบอร์ด Pi3/Pi4	136
5.2	เวอร์ชัลเม莫รี่ชนิดเพจ (Paging Virtual Memory)	138
5.2.1	หลักการพื้นฐานของเวอร์ชัลเม莫รี่	138

5.2.2	เวอร์ชั่ลเมมโมรีชนิดเพจ กรณีศึกษาบอร์ด Pi3/Pi4	141
5.3	หน่วยความจำแคช (Cache Memory)	145
5.3.1	แคชชนิด DM (Direct Map)	148
5.3.2	แคชชนิด N-way SA (Set Associative)	150
5.4	หน่วยความจำชนิดสแตติกแรม (Static RAM: SRAM)	154
5.4.1	โครงสร้างภายในของชิปหน่วยความจำสแตติกแรม	155
5.4.2	การทำงานของสแตติกแรม: ขบวนการอ่านข้อมูล	156
5.4.3	การทำงานของสแตติกแรม: ขบวนการเขียนข้อมูล	157
5.5	หน่วยความจำหลักชนิด SDRAM	158
5.5.1	โครงสร้างภายในของชิป SDRAM	159
5.5.2	การทำงานของ SDRAM: อ่านและเขียน	162
5.5.3	การรีเฟรช (Refresh) ข้อมูล	163
5.6	สรุปท้ายบท	164
5.7	คำถามท้ายบท	165
6	กลไกอินพุตและเอาต์พุต (Input and Output)	167
6.1	สัญญาณ HDMI สำหรับจอภาพ LCD ขนาดใหญ่	169
6.2	สัญญาณ DSI สำหรับจอภาพ LCD ขนาดเล็ก	172
6.3	สัญญาณ CSI สำหรับเชื่อมต่อกล้องขนาดเล็ก	174
6.4	สัญญาณ PCM สำหรับข้อมูลเสียงดิจิทัล	176
6.5	สัญญาณภาพและเสียงสำหรับจอทีวีและล็อก	178
6.6	สัญญาณ USB สำหรับอุปกรณ์ต่อพ่วงต่าง ๆ	179
6.7	สัญญาณสาย Ethernet เชื่อมต่อกับเครือข่ายอินเทอร์เน็ต	181
6.8	สัญญาณ WiFi และ Bluetooth สำหรับการสื่อสารไร้สาย	182
6.9	หลักการ Memory Mapped Input/Output	185
6.10	หัวเชื่อมต่อ 40 ขา (40-Pin Header)	187
6.11	ขา GPIO (General Purpose Input Output)	189
6.12	หลักการอินเทอร์รัปต์ (Interrupt)	194
6.13	หลักการเข้าถึงหน่วยความจำโดยตรง (Direct Memory Access)	199
6.14	แหล่งจ่ายไฟ (Power Supply)	201
6.15	สรุปท้ายบท	203
6.16	คำถามท้ายบท	203
7	อุปกรณ์เก็บรักษาข้อมูล (Data Storage Devices)	205
7.1	ระบบไฟล์ (File System)	206
7.1.1	การใช้งานไฟล์ (File Operations)	206

7.1.2	การแบ่งพาร์ทิชัน (Partition)	207
7.1.3	โครงสร้างของระบบไฟล์ยูนิกซ์ (Unix)	208
7.1.4	โครงสร้างข้อมูลไอโหนด (Inode)	209
7.2	ชิปหน่วยความจำแฟลช (Flash Memory)	213
7.2.1	การอ่านข้อมูลแบบเพจ (Page Read)	216
7.2.2	การเขียนข้อมูล (Program Data)	217
7.2.3	หน่วยความจำแฟลชชนิดอื่น ๆ	218
7.3	การ์ดหน่วยความจำ SD (Secure Digital)	219
7.3.1	การอ่านข้อมูล	221
7.3.2	การเขียนข้อมูล	222
7.4	โซลิดสเตทไดรฟ์ (Solid-State Disk: SSD)	223
7.5	ฮาร์ดดิสก์ไดรฟ์ (Hard Disk Drive: HDD)	225
7.5.1	โครงสร้างของฮาร์ดดิสก์ไดรฟ์เชิงกายภาพ	226
7.5.2	โครงสร้างของฮาร์ดดิสก์ไดรฟ์เชิงตรรกะ	226
7.5.3	ความจุและประสิทธิภาพของฮาร์ดดิสก์ไดรฟ์	227
7.6	สรุปท้ายบท	228
7.7	คำถามท้ายบท	229
8	การคำนวณแบบขนาน (Parallel Computing) ด้วยบอร์ด Pi3/Pi4	231
8.1	เครื่องคอมพิวเตอร์แบบขนานชนิดมัลติคอร์	233
8.1.1	กรณีศึกษาซีพียู ARM Cortex A72 และ Cortex A53	234
8.1.2	กรณีศึกษา Fujitsu A64FX ในเครื่องซูเปอร์คอมพิวเตอร์	235
8.2	ความขนาน (Parallelism)	237
8.2.1	ความขนานของข้อมูล (Data Parallelism)	238
8.2.2	ความขนานของงานย่อย (Task Parallelism)	238
8.2.3	ความขนานแบบผสม (Hybrid Parallelism)	238
8.3	การพัฒนาอัลกอริธึมแบบมัลติເର୍ଡ ແລະ ແບບขนานດ້ວຍໄລບରାଈ OpenMP	239
8.3.1	การคูณແມທ୍ରିକ୍ସ (Matrix Multiplication) ແບບขนาน	241
8.3.2	การเรียงข้อมูล (Sorting) ແບບขนาน	242
8.4	สรุปท้ายบท	246
8.5	คำถามท้ายบท	246
ภาคผนวก (Appendices)		249
A	การทดลองที่ 1 ข้อมูลและคณิตศาสตร์ในคอมพิวเตอร์	249
A.1	การแปลงและคณิตศาสตร์สำหรับเลขจำนวนเต็มฐานสอง	250

A.1.1	การทดลองแปลงเลขจำนวนเต็มฐานสอง	250
A.1.2	คณิตศาสตร์เลขจำนวนเต็มฐานสองขนาด 16 บิต	253
A.1.3	คณิตศาสตร์เลขจำนวนเต็มฐานสองขนาด 8 บิต	255
A.1.4	กิจกรรมท้ายการทดลอง	256
A.2	การแปลงและคณิตศาสตร์เลขทศนิยมฐานสองมาตรฐาน IEEE754	258
A.2.1	เลขทศนิยมชนิดจุดลอยตัวมาตรฐาน IEEE754 Single-Precision	258
A.2.2	กิจกรรมท้ายการทดลอง	261
A.3	รหัสของข้อมูลตัวอักษร	264
A.3.1	การทดลอง	264
A.3.2	กิจกรรมท้ายการทดลอง	265
B	การทดลองที่ 2 ตัวอย่างการประกอบและติดตั้งบอร์ด Raspberry Pi3	267
B.1	รายการอุปกรณ์ฮาร์ดแวร์	268
B.2	ตัวอย่างการประกอบบอร์ด Pi3 และกล่อง	269
B.2.1	ประกอบฮีทซิงก์ (Heat Sink)	269
B.2.2	ประกอบกล่องกับบอร์ด Pi3	269
B.3	เครื่องคอมพิวเตอร์ส่วนบุคคลจากบอร์ด Pi3/Pi4 โมเดล B	271
B.4	กิจกรรมท้ายการทดลอง	273
C	การทดลองที่ 3 การติดตั้งระบบปฏิบัติการ Raspberry Pi OS	275
C.1	การเตรียมการ์ดหน่วยความจำไมโคร SD	275
C.2	การติดตั้ง Raspberry Pi OS บนการ์ดหน่วยความจำไมโคร SD	276
C.3	การบูตระบบปฏิบัติการ RaspberryPi OS	280
C.4	การตั้งค่าบอร์ด Pi เพื่อใช้งาน	281
C.4.1	การตั้งค่าต่างๆ	281
C.4.2	การตั้งค่า WiFi เพื่อเชื่อมต่อกับอินเทอร์เน็ต	282
C.4.3	การรีสตาร์ตและชัตดาวน์	283
C.5	กิจกรรมท้ายการทดลอง	284
D	การทดลองที่ 4 การใช้งานระบบปฏิบัติการยูนิกซ์เบื้องต้น	285
D.1	การใช้งานระบบผ่านทาง GUI	285
D.1.1	หน้าจodesk็อป (Desktop)	285
D.1.2	ไฟล์เมเนจर์ (File Manager)	286
D.2	การใช้งานระบบผ่านทางโปรแกรม Terminal	287
D.2.1	คำสั่งพื้นฐานของระบบยูนิกซ์	289
D.2.2	การชัตดาวน์ (Shutdown)	289

D.3	ข้อมูลพื้นฐานของบอร์ด Pi	290
D.3.1	ข้อมูลพื้นฐานของซีพียู	290
D.3.2	ข้อมูลขั้นสูงของซีพียูและบอร์ด	291
D.4	กิจกรรมท้ายการทดลอง	292
E	การทดลองที่ 5 การพัฒนาโปรแกรมภาษา C บนลินุกซ์	293
E.1	การพัฒนาโดยใช้ IDE	293
E.2	การดีบัก (Debugging) โดยใช้ IDE	297
E.3	การพัฒนาโดยใช้ประโยชน์คำสั่ง (Command Line)	298
E.4	โครงสร้างของ Makefile	299
E.5	การพัฒนาโดยใช้ Makefile	299
E.6	การตรวจจับ Overflow กรณิศาสตร์เลขจำนวนเต็มฐานสอง	301
E.6.1	เลขจำนวนเต็มฐานสองไม่มีเครื่องหมาย	301
E.6.2	เลขจำนวนเต็มฐานสองมีเครื่องหมายชนิด 2's Complement	302
E.7	กิจกรรมท้ายการทดลอง	303
F	การทดลองที่ 6 การพัฒนาโปรแกรมภาษาแอลซีเมบลี	305
F.1	การพัฒนาโดยใช้ IDE	305
F.2	การดีบักโปรแกรมโดยใช้ IDE	308
F.3	การพัฒนาโดยใช้ประโยชน์คำสั่ง (Command Line)	308
F.4	การพัฒนาโดยใช้ Makefile	309
F.5	กิจกรรมท้ายการทดลอง	310
G	การทดลองที่ 7 การเรียกใช้และสร้างฟังก์ชันในโปรแกรมภาษาแอลซีเมบลี	313
G.1	การใช้งานตัวแปรในดาต้าเซกเมนต์	313
G.1.1	การโหลดค่าตัวแปรเดี่ยวจากหน่วยความจำมาพักในรีจิสเตอร์	314
G.1.2	การใช้งานตัวแปรชุดหรืออาร์เรย์ ชนิด word	316
G.1.3	การใช้งานตัวแปรอาร์เรย์ชนิด byte	317
G.1.4	การเรียกใช้ฟังก์ชันและตัวแปรชนิดประโยชน์รหัส ASCII	318
G.2	การสร้างฟังก์ชันเสริมด้วยภาษาแอลซีเมบลี	320
G.3	กิจกรรมท้ายการทดลอง	324
H	การทดลองที่ 8 การพัฒนาโปรแกรมภาษาแอลซีเมบลีขั้นสูง	327
H.1	ดีบักเกอร์ GDB	327
H.2	การใช้งานสเตกพอยน์เตอร์ (Stack Pointer)	332
H.3	การพัฒนาโปรแกรมภาษาแอลซีเมบลีร่วมกับภาษา C	335

H.4	กิจกรรมท้ายการทดลอง	336
I	การทดลองที่ 9 การศึกษาและปรับแก้อินพุตและเอาต์พุตต่างๆ	339
I.1	จอแสดงผลผ่านพอร์ต HDMI	339
I.1.1	การปรับแก้ความละเอียดของจอแสดงผล	339
I.1.2	การปรับแก้ขนาดหน่วยความจำของ GPU	341
I.2	ระบบเสียงดิจิทัล	342
I.2.1	การเลือกช่องสัญญาณเสียงเชื่อมต่อกับลำโพง	342
I.2.2	การควบคุมระดับเสียง	345
I.2.3	รายชื่ออุปกรณ์ในระบบเสียง	345
I.3	พอร์ตเชื่อมต่ออุปกรณ์ USB	347
I.3.1	รายชื่ออุปกรณ์กับพอร์ต USB	347
I.3.2	รายละเอียดการเชื่อมต่ออุปกรณ์กับพอร์ต USB	348
I.4	พอร์ตเชื่อมต่ออินเทอร์เน็ตผ่านสัญญาณ WiFi และ Ethernet	351
I.4.1	รายชื่ออุปกรณ์เครือข่าย	351
I.4.2	การเปิด/ปิดอุปกรณ์เครือข่าย	352
I.4.3	การตรวจสอบการเชื่อมต่อกับอินเทอร์เน็ตเบื้องต้น	353
I.5	กิจกรรมท้ายการทดลอง	354
J	การทดลองที่ 10 การเชื่อมต่อกับขา GPIO	355
J.1	ไลบรารี wiringPi	355
J.2	วงจรไฟ LED กระพริบ	357
J.3	โปรแกรมไฟ LED กระพริบภาษา C	359
J.4	โปรแกรมไฟ LED กระพริบภาษาแอสเซมบลี	360
J.5	กิจกรรมท้ายการทดลอง	362
K	การทดลองที่ 11 การเชื่อมต่ออินพุต-เอาต์พุตกับสัญญาณอินเทอร์รัปต์	365
K.1	การจัดการอินเทอร์รัปต์ของ WiringPi	365
K.2	วงจรสวิตซ์ปุ่มกดเชื่อมผ่านขา GPIO	366
K.3	โปรแกรมภาษา C สำหรับทดสอบวงจรอินเทอร์รัปต์	367
K.4	กิจกรรมท้ายการทดลอง	368
L	การทดลองที่ 12 การศึกษาอุปกรณ์เก็บรักษาข้อมูลและระบบไฟล์	371
L.1	ขนาดของไฟล์และไดเรกทอรี	371
L.2	ระบบไฟล์	373
L.3	อุปกรณ์อินพุต/เอาต์พุตในระบบไฟล์	376

L.4	กิจกรรมท้ายการทดลอง	378
M	การทดลองที่ 13 การพัฒนาอัลกอริธึมแบบขنانด้วยไลบรารี OpenMP	381
M.1	การวัด CPU Utilization	381
M.2	การคุณเมทริกซ์แบบขنان	382
M.3	ความซับซ้อน (Complexity) ของการคำนวณ	386
M.4	ประสิทธิภาพ (Performance) ของการคำนวณแบบขنان	387
M.5	กิจกรรมท้ายการทดลอง	388
บรรณานุกรม (Bibliography)		389
อภิธานศัพท์ (Glossary)		391
ดัชนี (Index)		393

สารบัญตาราง (List of Tables)

1	ตารางเปรียบเทียบตัวคูณด้วยค่าสองยกกำลังและสิบยกกำลัง	i
1.1	ชนิดและจำนวนอุปกรณ์ ($\times 10^6$) จำนวนชิป ($\times 10^6$) จำนวนชิปต่ออุปกรณ์ จำนวนชิป TAM (Total Available Market) ($\times 10^6$) จำนวนชิปของซีพียู ARM ($\times 10^6$) และเพอร์เซ็นต์ส่วนแบ่งการตลาด (Share) ของซีพียู ARM ในปี ค.ศ. 2010 ที่มา: zdnet.com หมายเหตุ TAM: Total Available Market	4
1.2	ชนิดอุปกรณ์ จำนวนชิป TAM (Total Available Market) ปี 2010 ($\times 10^6$) จำนวนชิปของซีพียู ARM ปี 2010 ($\times 10^6$) จำนวนอุปกรณ์ TAM ปี 2015 ($\times 10^6$) จำนวนชิปต่ออุปกรณ์ และจำนวนชิป TAM ปี 2015 ($\times 10^6$) ที่มา: 7boot.com หมายเหตุ TAM: Total Available Market	5
2.1	ชนิด ความยาว (บิต) ค่าฐานสิบต่ำสุดและสูงสุดของตัวแปรพื้นฐานแต่ละชนิดในภาษา C/C++ หมายเหตุ ค่าต่ำสุดและค่าสูงสุดของ IEEE754 เป็นค่าเลข_norm ลैลีซ์ (Normalize) เพื่อแสดงความสมมาตรของเลขยกกำลัง ที่มา: ntu.edu.sg	12
2.2	การแปลงค่าฐานสิบ เป็นเลขฐานสองแบบไม่มีเครื่องหมาย ความยาว $n=8$ บิต ด้วยการหาร	18
2.3	การแปลงค่าฐานสิบ เป็นเลขฐานสองแบบไม่มีเครื่องหมาย ความยาว $n=8$ บิต ด้วยวิธีการลบ	19
2.4	รูปแบบ (Pattern) ของเลขฐานสองขนาด $n=4$ บิตทั้งหมด $2^4=16$ แบบและการตีความหมายให้เป็นค่าฐานสิบแบบมีเครื่องหมาย 2's Complement และแบบไม่มีเครื่องหมาย	21
2.5	รูปแบบ (Pattern) ของเลขฐานสองขนาด $n=5$ บิตทั้งหมด $2^5=32$ แบบและการตีความหมายให้เป็นค่าฐานสิบแบบมีเครื่องหมาย 2's Complement และแบบไม่มีเครื่องหมาย	22
2.6	รูปแบบ (Pattern) ของเลขฐานสองขนาด $n=8$ บิตทั้งหมด $2^8=256$ แบบและการตีความหมายให้เป็นค่าฐานสิบแบบมีเครื่องหมาย 2's Complement และแบบไม่มีเครื่องหมาย	23
2.7	การแปลงค่าฐานสิบแบบมีเครื่องหมายให้เป็นเลขฐานสองความยาว $n=4$ บิต โดยแปลงให้เป็นค่าฐานสิบโดยใช้สมการที่ (2.41) กรณีที่ $X_{10,s} < 0$	26

2.8	การแปลงค่าฐานสิบแบบมีเครื่องหมายให้เป็นเลขฐานสองความยาว $n=5$ บิตโดยแปลงให้เป็นค่าฐานสิบโดยใช้สมการที่ (2.41) กรณีที่ $X_{10,s} < 0$	27
2.9	เลขจำนวนเต็มฐานสองความยาว $n=4$ บิตทั้งหมด $2^4=16$ แบบสามารถตีความเป็นเลขจำนวนเต็มฐานสิบแบบมีเครื่องหมายชนิด Sign-Magnitude, แบบมีเครื่องหมายชนิด 2's Complement, และแบบไม่มีเครื่องหมาย (Unsigned)	29
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	34
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	36
2.12	ตาราง สรุป ความแตกต่างระหว่างเลขทศนิยมฐานสองชนิดจุดลอยตัว ตามมาตรฐาน IEEE754 ชนิด Single Precision โดย $E_{2,IEEE}$ คือ ค่ายกกำลัง และ Y_2 คือ ทศนิยมซึ่งเป็นส่วนประกอบของค่านัยสำคัญ (x หมายถึง บิตข้อมูลมีค่าเท่ากับ '0' หรือ '1')	49
2.13	ชนิด ความยาวข้อมูล และการประยุกต์ใช้งานเลขฐานสองชนิดต่างๆ ในคอมพิวเตอร์	58
3.1	ตารางสรุปข้อมูลด้านฮาร์ดแวร์และซอฟต์แวร์ของบอร์ด Pi3 ไมเดล B และลิงก์เชื่อมไปยังหัวข้อที่แสดงรายละเอียดในบทต่างๆ	65
3.2	ตัวอย่างโปรแกรมภาษาแอสเซมบลีเพื่อประกาศและตั้งค่าเริ่มต้นให้ตัวแปรขนาด 32 บิต จำนวน 4 ตัวแปร และวนรอบบางค่าโดยใช้ตัวแปรอยู่น์เตอร์	77
4.1	ตัวอย่างโปรแกรมภาษาแอสเซมบลีเพื่อประกาศและตั้งค่าเริ่มต้นตัวแปรขนาด 32 บิต จำนวน 2 ตัวแปร ในพื้นที่ของดาต้าเซกเมนต์	104
4.2	ตัวอย่างโปรแกรมภาษาแอสเซมบลีเพื่ออ่านค่าตัวแปรจากดาต้าเซกเมนต์	106
4.3	ตัวอย่างโปรแกรมภาษาแอสเซมบลีเพื่อกำหนดรูปแบบ $x = (a + b) - c$	107
4.4	ตัวอย่างโปรแกรมตามประโยคเงื่อนไข if	117
4.5	ตัวอย่างโปรแกรมตามประโยคเงื่อนไข if-else	119
4.6	ตัวอย่างโปรแกรมประโยควนรอบ For ตามสมการที่ (4.1)	122
4.7	ตัวอย่างโปรแกรมตามประโยควนรอบชนิด WHILE ตามสมการที่ (4.1)	124
4.8	ตัวอย่างโปรแกรมตามประโยควนรอบชนิด DO-WHILE ตามสมการที่ (4.1)	126
4.9	ตัวอย่างโปรแกรมเรียกใช้ฟังก์ชันภาษาแอสเซมบลีที่คล้ายกับตัวอย่างที่ 4.8.1	130
5.1	ตัวอย่างโปรแกรมภาษาแอสเซมบลีเพื่อประกอบการอธิบายการทำงานของแคชทั้งสองชนิด	147
6.1	หมายเลขอื่นๆ และวัตถุประสงค์ของสายสัญญาณชนิด HDMI เวอร์ชัน 1.4 ที่มา: wikipedia.org	170

6.2	หมายเลขอ้างอิงที่ ของสายสัญญาณ DSI สำหรับจอภาพ LCD ขนาดเล็ก ประกอบด้วยข้อมูลภาพจำนวน 2 เลน ที่มา: wikipedia.org	172
6.3	หมายเลขอ้างอิง และวัตถุประสงค์ของสัญญาณชนิด CSI wikipedia.org	175
6.4	ตารางเชื่อมโยงระหว่าง VC/CPU บัสแอดเดรส อุปกรณ์อินพุต/เอาต์พุต (I/O Peripherals) และหมายเลขอ้างอิงของตราเล่มนี้ เริ่มต้นที่หมายเลข 0x7E00 0000 หมายเหตุ Rx: Receiver, Tx: Transmitter, UART: Universal Asynchronous/Synchronous Receiver Transmitter ที่มา: Broadcom Corp. (2012)	186
6.5	หมายเลขอ้างอิง ซึ่อๆ แต่ละตัวเลือก (ซึ่อ สัญญาณ) ที่ 0-5 ของหัวเชื่อมต่อสายทั้ง 40 ขา (GND: Ground) ที่มา: Broadcom Corp. (2012), ที่มา: Raspberry Pi (Trading) (2019)	188
6.6	ตารางเชื่อมโยงระหว่าง VC/CPU บัสแอดเดรส กับ รีจิสเตอร์ต่าง ๆ ของวงจร GPIO เพื่อตั้งค่าและสั่งงานขา GPIO ทุกขา โดยเริ่มต้นที่หมายเลข 0x7E20 0000 ที่มา: Broadcom Corp. (2012)	191
6.7	ตารางเลือกค่า V_{OUT} และค่าตัวเหนี่ยวนำ L_1 และ L_2 ที่มา: Diodes, Inc. (2012) .	202
7.1	ตารางเปรียบเทียบเซลล์หน่วยความจำแฟลชชนิด NAND (ซ้าย) และ NOR (ขวา) ตามโครงสร้างและผังการจัดวางของเซลล์หน่วยความจำ ที่มา: Choi (2010) หมายเหตุ F คือ ความกว้างของฟlot gate (Float Gate) มีหน่วยเป็น นาโนเมตร .	218
7.2	หมายเลขอ้างอิง และ วัตถุประสงค์ ของ สัญญาณ ใน โหมด SD 4 บิต ที่มา: wikipedia.org	221
7.3	การเปรียบเทียบประสิทธิภาพด้านต่าง ๆ ของอุปกรณ์เก็บรักษาข้อมูล	229
M.1	เวลาและ $\%CPU_{max}$ ของการคูณเมทริกซ์ที่ขนาด N และจำนวน紀錄เท่ากับ 1, 2, 4, 8 เ rekrd	385
M.2	อัตราส่วนเวลา การคูณ เมทริกซ์ที่ขนาด $N=400, 800, 1000$ เทียบกับเวลาที่ขนาด $N=200$ ที่จำนวน纪录เท่ากับ 1, 2, 4, 8 rekrd จากสมการที่ (M.2)	386
M.3	ผลการคำนวณ $Speedup(n)$ ของการคูณเมทริกซ์ที่ขนาด $N=200, 400, 800, 1000$ และจำนวน纪录เท่ากับ 2, 4, 8 rekrd เทียบกับ 1 rekrd ด้วยสมการที่ (M.5)	387

สารบัญรูปภาพ (List of Figures)

1.1	ตัวอย่างเครื่องคอมพิวเตอร์ชนิดเซิร์ฟเวอร์ จากบริษัท Hewlett Packard รุ่น Proliant ที่มา: datacenterknowledge.com	2
1.2	ปริมาณยอดขาย (ล้านหน่วย) เครื่องคอมพิวเตอร์ชนิดเดสก์ทอป ในตบุ๊ค スマาร์ตโฟน และแท็บเล็ตทั่วโลกในปี ค.ศ. 2010-2016 ที่มา: statista.com	3
1.3	ภาพถ่ายซิลิกอนดาย (Silicon Die) ของชิป BCM2835 บนบอร์ด Raspberry Pi ที่มา: raspberrypi.org	6
1.4	ขั้นตอนการผลิตชิป (Chip) ภายในประกอบด้วยวงจรรวม ไมโครโปรเซสเซอร์และ อื่น ๆ ที่มา: slideplayer.com	7
2.1	เลขจำนวนเต็มและเลขจำนวนจริงบนเส้นจำนวนในคณิตศาสตร์เลขฐานสิบ ที่มา: tutorvista.com	12
2.2	พื้นที่ในหน่วยความจำซึ่งบรรจุค่าเริ่มต้นของตัวแปรที่ได้ประกาศในตัวอย่างที่ 2.1.1	13
2.3	เส้น จำนวน เปรียบ เทียบ ข้อมูล เลข ฐาน สอง ความ yaw 4 บิต ใน รูป แบบ ของ จำนวน เต็ม ชนิด ไม่มี เครื่องหมาย (Unsigned) ชนิด มี เครื่อง หมาย และ 2's Complement และ ชนิด มี เครื่อง หมาย แบบ Sign-Magnitude ที่มา: Harris and Harris (2013)	15
2.4	สัญลักษณ์ของ ALU (Arithmetic Logic Unit) สำหรับ บวก/ลบ เลขจำนวนเต็ม ขนาด n บิต ชนิด ไม่มี เครื่อง หมาย ที่มา: teach-ict.com	30
2.5	วงจรคูณเลขขนาด $n=32$ บิต ชนิด ที่ 1 โดยใช้ ALU ขนาด $2n = 64$ บิต และ รีจิส เตอร์ตัวตั้งขนาด $2n = 64$ บิต ที่มา: Patterson and Hennessy (2016)	33
2.6	วงจรคูณเลขขนาด $n=32$ บิต ชนิด ที่ 2 โดยใช้ ALU ขนาด $n=32$ บิต และ รีจิส เตอร์ตัวตั้งขนาด $n=32$ บิต ที่มา: Patterson and Hennessy (2016)	35
2.7	โครงสร้างของเลขศนิยมฐานสองชนิด จุด ลอยตัวตามมาตรฐาน IEEE754 Double Precision และ Single Precision ที่มา: pybugs.com	45
2.8	เลขศนิยม ลอยตัว ชนิด นอร์มัล ไลซ์ ตั้งแต่ $\pm N_{min}$ ถึง $\pm N_{max}$ และ ชนิด ดินอร์มัล ไลซ์ ตั้งแต่ $\pm D_{min}$ ถึง $\pm D_{max}$ ที่มา: ntu.edu.sg	50
2.9	วงจรบวกเลขชนิด จุด ลอยตัวตามมาตรฐาน IEEE754 โดยรับค่า อินพุต จำนวน 2 ตัวด้านบน และ เอ้าต์พุตผลลัพธ์ด้านล่าง ที่มา: Patterson and Hennessy (2016)	51

2.10	วงจรคุณเลขทศนิยมฐานสองชนิดจุดลอยตัวตามมาตรฐาน IEEE754 โดยรับค่าอินพุตจำนวน 2 ตัวด้านบน และเอาต์พุตผลลัพธ์ด้านล่าง ที่มา: Patterson and Hennessy (2016)	54
2.11	การเก็บข้อมูลในหน่วยความจำในรูปของรหัสแอสกีด้วยตัวแปรสตริงซึ่งว่า str ซึ่งเป็นตัวแปรชนิด char[10] str="Hello!" ที่แอดเดรสเริ่มต้น 0x50 หรือ 50_{16} ที่มา: Harris and Harris (2013)	56
2.12	ตารางรหัส ASCII และ Unicode สำหรับตัวอักษรจำนวน $2^8=256$ ตัว ประกอบด้วยรหัสของตัวอักษรภาษาอังกฤษและภาษาไทย ที่มา: ascii-table.com	57
3.1	วงรอบ การเขื่อมโยง อินพุต-เอ้าต์พุต (Input-Output Loop) ระหว่างผู้ใช้ハードแวร์ และซอฟต์แวร์ (เกม) ที่มา: wikipedia.org	63
3.2	ตำแหน่งของอุปกรณ์ต่างๆ บนบอร์ด Pi3/Pi4 ที่มา: raspberryhome.net	65
3.3	บล็อกໄດอะแกรมของชิป AllWinner A64 ที่มีชิปปูย ARM Cortex A53 คล้ายกับชิป BCM2837 บนบอร์ด Raspberry Pi3 ที่มา: Allwinner Technology, Co. (2015)	67
3.4	หน่วยความจำ SDRAM ด้านล่างของบอร์ด Pi3/Pi4 โมเดล B ที่มา: robo-dyne.com	68
3.5	ตำแหน่งและรหัสประจำปุ่ม (Scan Code หรือ Position Code) บนคีย์บอร์ดชนิด 106 ปุ่ม ที่มา: ps-2.kev009.com	69
3.6	โครงสร้างภายในแม่สีชนิด Optical ประกอบด้วยหลอด LED ส่องแสงกระแทบกับพื้นผิว ด้านล่าง แล้วสะท้อนกลับมายังตัวรับแสงชนิด CMOS ที่มา: www.pinterest.com	70
3.7	โครงสร้างจอแสดงผลชนิด Color LCD ประกอบด้วยแหล่งกำเนิดแสง (Light Source) ปล่อยแสงทะลุชั้นต่างๆ ปราศจากแก่สายตาผู้ใช้ ที่มา: techterms.com	71
3.8	โครงสร้างของการหดหน่วยความจำ SD ชนิด SDHC ความจุ 16 กิกะไบต์ (GB) ประกอบด้วยชิปหน่วยความจำแฟลช วงจรควบคุม และวงจรเขื่อมต่อ ที่มา: wikipedia.org	72
3.9	ตัวอย่างซอฟต์แวร์ภาษา C ฟังก์ชันหลักชื่อ main() เมื่อทำงานเสร็จสิ้นจะรีเทิร์นค่า 0 ที่มา: zentut.com	74
3.10	ขั้นตอนการพัฒนาซอฟต์แวร์จากภาษา C/C++ ให้เป็นโปรแกรมประยุกต์ (Application Program) หรือย่อว่าแอป ที่มา: slideplayer.com	75
3.11	ตัวอย่างการลิงก์ (Link) หรือรวมไฟล์อีบเจกต์หลักไฟล์อีบเจกต์เสริม และไฟล์ไลบรารีเข้าด้วยกันเป็นไฟล์โปรแกรมหรือแอปพลิเคชัน ที่มา: google.com	79
3.12	ตัวอย่างการแปลงจากคำสั่งและเซมบลีของชิปปูย ARM ทางด้านขวาเป็นคำสั่งภาษาเครื่องทางด้านซ้าย ที่มา: Harris and Harris (2013)	80
3.13	โครงสร้างของไดเรกทอรี (Directory) สำคัญๆ ในระบบปฏิบัติการลินุกซ์ ที่มา: freedompenguin.com	84

3.14	โครงสร้างไฟล์ชนิด ELF สำหรับเก็บชุดคำสั่งภาษาเครื่อง และข้อมูลเป็นเลขฐานสองอยู่ในอุปกรณ์เก็บรักษาข้อมูล ที่มา: wikipedia.org	86
3.15	โครงสร้างของคอมพิวเตอร์ประกอบด้วยฮาร์ดแวร์ชั้นล่างสุด และหน่วยความจำเวอร์ชวล เมมโมรี (Virtual memory) ภายใต้ระบบลินุกซ์ แบ่งเป็นพื้นที่ เรียกว่า เคอร์เนลสเปช (ระบบปฏิบัติการ) และ ยูสเซอร์สเปช (ซอฟต์แวร์ประยุกต์) ที่มา: linux-india.org	87
3.16	การจัดวางคำสั่งภาษาเครื่องในเทกซ์เชิกเมนต์ และข้อมูลในดาต้า เชิกเมนต์ จากไฟล์โปรแกรมรูปแบบ ELF ไปยัง เวอร์ชวล เมมโมรี บริเวณ ยูสเซอร์สเปช (User Space) ที่มา: wordpress.com	89
3.17	(a) ขบวนการอ่าน (Load) ข้อมูลจากหน่วยความจำหลักที่ตำแหน่ง 111_2 (b) ขบวนการเขียน (Store) ข้อมูลในหน่วยความจำหลักที่ตำแหน่ง 101_2	91
3.18	โครงสร้างของลินุกซ์คอร์เนลในเวอร์ชวลเมมโมรี ประกอบด้วย I/O Subsystem, Memory Management Subsystem และ Process Management Subsystem ที่มา: wikipedia.org	92
3.19	หลักการของ Memory Mapped File ที่มา: rice.edu	93
4.1	โครงสร้างภายในแกนประมวลผลหมายเลข 0 ของชิปปี้ Cortex A53 ออกแบบโดยบริษัท ARM ที่มา: tomshardware.com	98
4.2	โครงสร้างเชิงตรรกะของแกนประมวลผลประกอบด้วยรีจิสเตอร์ R0-R15 แคชต่าง ๆ และหน่วยความจำหลัก, (VA: Virtual Address)	100
4.3	คำสั่งภาษาเครื่องที่ถูกอ่าน (Load) เข้าสู่หน่วยความจำและแสดงในรูปของภาษาแอสเซมบลีบนโปรแกรมซิมูเลเตอร์ (Simulator) ในบริเวณเทกซ์เชิกเมนต์ ที่มา: arm.com	102
4.4	รีจิสเตอร์สำหรับเก็บสถานะของชิปปี้ ณ ปัจจุบัน (Current Program Status Register: CPSR) พร้อมรายละเอียดทั้ง 32 บิต ที่มา: arm.com	109
4.5	คำสั่ง add r3, r4, r2, lsl #2 หมายถึง $r3 = r4 + (r2 \ll 2)$ โดยหากค่า r4 กับค่า r2 หลังจากเลื่อนบิตไปทางซ้ายจำนวน 2 บิตแล้วจึงเก็บผลลัพธ์ใน r3 ที่มา: CodeMachine.com หมายเหตุ ค่าที่ยังเก็บอยู่ในรีจิสเตอร์ r2 จะไม่เปลี่ยนแปลง .	112
4.6	ไฟล์ชาร์ตการทำงานของประโภค IF	115
4.7	ไฟล์ชาร์ตการทำงานของประโภค IF-ELSE	118
4.8	ไฟล์ชาร์ตการทำงานของประโภคการวนรอบชนิด FOR	121
4.9	ไฟล์ชาร์ตการทำงานของประโภคการวนรอบชนิด WHILE	123
4.10	ไฟล์ชาร์ตการทำงานของประโภคการวนรอบชนิด DO-WHILE	125
4.11	ไฟล์ชาร์ตการทำงานของประโภคการวนรอบชนิด For 2 ชั้น	127
4.12	ไฟล์ชาร์ตการทำงานของประโภคเรียกใช้ฟังก์ชัน sum(a, b) ในตัวอย่างที่ 4.8.1	129

4.13	การทำงานของคำสั่ง BL myFunction เมื่อมีการเรียกใช้ 2 ครั้งจากฟังก์ชัน main ในหน่วยความจำล้ายกับตัวอย่างที่ 4.8.2 หมายเหตุ ... หมายถึงประโยชน์คำสั่ง อื่น ๆ ที่ไม่เกี่ยวข้องกับตัวอย่าง	131
4.14	การควบรวมชุดคำสั่งภาษาแอสเซมบลีของซีพียู ARM ตั้งแต่เวอร์ชัน 5 ถึงเวอร์ชัน 8A โดยเวอร์ชันใหม่จะรวมคำสั่งของเวอร์ชันเก่าเพื่อรองรับการทำงานซอฟต์แวร์เดิม ที่มา: arm.com	132
5.1	ลำดับขั้นของหน่วยความจำชนิดต่าง ๆ สำหรับชิป BCM2837/BCM2711, (VA: Virtual Address)	136
5.2	การแมป (Map) เวอร์ชวลแอดдресส์ (Virtual Address) ขนาด 3 กิกะไบต์ (GiB) เป็นแอดเดรสกายภาพ (Physical Address) ขนาด 64 เมกะไบต์ (MiB) ตามหลักการเวอร์ชวลเมโมรีชนิดเพจ (Paging Virtual Memory) หมายเหตุ เส้นประ คือ รอยต่อระหว่างเพจของเวอร์ชวลเมโมรี	139
5.3	การแปลงหมายเลข เพจ เวอร์ชวล (Virtual Page Number) เป็นหมายเลข เพจ กายภาพ (Physical Page Number) ที่มา: slideplayer.com	140
5.4	โครงสร้างของเวอร์ชวลเมโมรีชนิดเพจของบอร์ด Pi3/Pi4 ไมเดล B ซึ่งใช้ชิปตระกูล BCM283x, x=5, 6, 7 หมายเหตุ ขนาดของรูปไม่เป็นไปตามพื้นที่ตามความเป็นจริง, MMU: Memory Management Unit ที่มา: Broadcom Corp. (2012)	141
5.5	โครงสร้างของซีพียู ARM Cortex A7 ประกอบด้วย TLB และแคชหลายระดับเพื่อรองรับการทำงานของเวอร์ชวลเมโมรีชนิดเพจ ที่มา: ARM Ltd. (2011) และ arm.com	143
5.6	โครงสร้างเชิงตรรกะเชื่อมโยงระหว่างรีจิสเตอร์ แคชลำดับที่ 1 และ 2 ภายในชิป BCM2837/BCM2711 และหน่วยความจำหลักภายนอก หมายเหตุ VA: Virtual Address	145
5.7	ไฟล์ชาร์ตการทำงานของแคชหนึ่งลำดับขั้นสำหรับการเฟห์ซคำสั่ง และสำหรับการอ่าน/เขียนข้อมูล	146
5.8	การทำงานของแคชคำสั่ง (Instruction Cache) ชนิด Direct Map เมื่อซีพียูเฟห์ซคำสั่งที่หน่วยความจำกัยภาพแอดdress (a) แคชคำสั่งยังไม่มีคำสั่งใด ๆ เลย (b) PC ออฟเซตเท่ากับ 000_{16} , (c) 004_{16} , (d) 008_{16}	148
5.9	การทำงานของแคชคำสั่ง (Instruction Cache) ชนิด Direct Map เมื่อซีพียูเฟห์ซคำสั่งที่ PC ออฟเซตเท่ากับ (e) 030_{16} , (f) 034_{16} , (g) $00C_{16}$, (h) 010_{16}	149
5.10	การทำงานของแคชลำดับที่ 1 แคชคำสั่ง (Instruction Cache) ชนิด 2-Way Set Associative เมื่อ PC เฟห์ซคำสั่งที่หน่วยความจำกัยภาพแอดdress (a) แคชคำสั่งยังไม่มีคำสั่งใด ๆ เลย (b) PC ออฟเซตเท่ากับ 000_{16} , (c) 004_{16} , (d) 008_{16}	151

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}	152
5.12	ตัวถังแบบ TSOP ของชิปหน่วยความจำชิโนด静态ติกแรม Cypress 62256 ที่มา: Cypress Semiconductor, Corp. (2002)	154
5.13	โครงสร้างของหน่วยความจำชิโนด静态ติก Cypress 62256 ที่มา: Cypress Semiconductor, Corp. (2002)	155
5.14	ไดอะแกรมเวลา (Timing Diagram) สำหรับการอ่าน (Read) ข้อมูลจากหน่วยความจำชิโนด静态ติกแรม ที่มา: Cypress Semiconductor, Corp. (2002)	156
5.15	ไดอะแกรมเวลา (Timing Diagram) สำหรับการเขียน (Write) ข้อมูลในหน่วยความจำชิโนด静态ติกแรม ที่มา: Cypress Semiconductor, Corp. (2002)	157
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	158
5.17	บล็อกไออะแกรมภายในชิปหน่วยความจำ SDRAM ชนิด DDR2 ประกอบด้วย Die 2 ชิ้น แต่ละชิ้นมีบล็อกข้อมูลขนาด 16 บิตเรียงขนาดกันเป็น 32 บิต ที่มา: Micron Technology, Inc. (2014)	161
5.18	ไดอะแกรมเวลา (Timing Diagram) สำหรับการอ่านต่อเนื่องของหน่วยความจำ SDRAM รุ่น Elpida B8132B4PB-8D-F คำสั่ง Burst READ โดย RL (Read Latency) เท่ากับ 5 ไบเกล BL (Burst Length) เท่ากับ 4 ตำแหน่ง หมายเหตุ รูปมี การตัดต่อเพื่อเน้นบริเวณที่สำคัญ, NOP=No Operation ที่มา: Micron Technology, Inc. (2014)	162
5.19	ไดอะแกรมเวลา (Timing Diagram) สำหรับการเขียนต่อเนื่องของหน่วยความจำ Elpida รุ่น B8132B4PB-8D-F ตามคำสั่ง Burst WRITE โดย WL (Write Latency) = 1 ไบเกล, BL (Burst Length) = 4 ตำแหน่ง หมายเหตุ: รูปมีการตัดต่อเพื่อเน้นบริเวณที่สำคัญ ที่มา: Micron Technology, Inc. (2014)	163
6.1	การเชื่อมต่ออุปกรณ์ต่าง ๆ บนบอร์ด Pi3 โดยมีชิป BCM2837 เป็นศูนย์กลาง ที่มา: xdevs.com	168
6.2	ภาพความละเอียด 720 จุด x480 เส้นที่ผู้ใช้มองเห็น แบ่งเป็นการส่งแพ็กเก็ตข้อมูล จุด ภาพ และแพ็กเก็ต ควบคุม ทางช่องสัญญาณ TMDS ภายใต้สัญญาณ HDMI ในช่วงเวลาต่าง ๆ ที่มา: freebsd.org	169
6.3	หัวเชื่อมต่อ HDMI ชนิด Female ประกอบด้วยขาสัญญาณทั้งหมด 19 ขา ที่มา: wikipedia.org	170
6.4	จอแสดงผลสำหรับ เชื่อมต่อระหว่างบอร์ด Pi3/Pi4 ด้วยสัญญาณการแสดงผลแบบอนุกรม (Display Serial Interface) ที่มา: element14.com	172

6.5	เบรี่ยบ เที่ยบ สัญญาณ DS1 ชนิด หนึ่ง เลน (Single Lane) และ ชนิด 4 เลน ที่มา: wikipedia.org	173
6.6	การ เชื่อมต่อ ระหว่างบอร์ด Pi3/Pi4 และ กล้องขนาดเล็ก ด้วย สัญญาณ กล้องแบบ อนุกรม (Camera Serial Interface) ที่มา: element14.com	174
6.7	สัญญาณ ดิจิทัล PCM (Pulse Code Modulation) ความ ละเอียด 16 ระดับ สุ่ม (Sampling) จาก สัญญาณ แอนะล็อก รูปคลื่น ไซน์ (Sine Wave) ด้วย ความถี่ สูง 26 เท่า ของ ความถี่ สูงสุด ($f_s=26f_{max}$) ที่มา: wolfcrow.com	176
6.8	มอ ดูล PCM ประกอบด้วย วงจร เชื่อมต่อ กับ ซีพียู ผ่าน บัส APB, หน่วย ความจำ บัฟเฟอร์ สำหรับ ส่ง และ รับ ข้อมูล เสียง ดิจิทัล และ วงจร เชื่อมต่อ ชนิด I ² S ที่มา: Broadcom Corp. (2012)	177
6.9	แจ็ค ขนาด 3.5 มม. (กลาง) ชนิด 4 ขั้ว สำหรับ เสียบ กับ บอร์ด Pi3/Pi4 (ขวา) ส่ง สัญญาณภาพ ไปยัง แจ็ค RCA (เหลือง) และ สัญญาณเสียง ไปยัง แจ็ค RCA (แดง และ ขาว) ที่มา: stackexchange.com	178
6.10	หัว เชื่อมต่อ USB ชนิด A (บน ฝั่ง โฮสต์) และ B (ล่าง ฝั่ง อุปกรณ์) ประกอบด้วย สัญญาณ 4 เส้น กราวน์ (0 โวลต์) (GND) Data+ Data- และ ไฟ กระแสตรง 5 โวลต์ ที่มา: quora.com	179
6.11	โครงสร้าง ของ ชิป LAN 9514 ภายใน ประกอบด้วย วงจร USB Hub และ วงจร Ethernet ที่มา: Microchip Technology, Inc. (2009)	180
6.12	หัว เชื่อมต่อ ชนิด RJ45 (ชาย สุด) สำหรับ การ เชื่อมต่อ เครือข่าย ท้องถิ่น (Local Area Network) แบบ มีสาย Ethernet ที่มา: cytron.io	181
6.13	การ เช้า ปลาย สาย RJ45 ทั้ง สอง ด้าน สำหรับ การ เชื่อมต่อ ระหว่าง เครื่อง คอมพิวเตอร์ และ อุปกรณ์ สวิตช์ (Switch) ตาม มาตรฐาน TIA T568B ที่มา: blogspot.com	181
6.14	รูป ถ่าย ด้านล่าง ของ บอร์ด Pi3 และ รูป ขยาย ของ ชิป BCM43438 สำหรับ เชื่อมต่อ เครือข่าย ไร้สาย WiFi และ เครือข่าย ไร้สาย บลูทูธ (Bluetooth) ที่มา: stackexchange.com	182
6.15	บล็อก ไดอะแกรม ภายใน ชิป BCM43438 ประกอบ ด้วย ขา สัญญาณ เชื่อมต่อ เสา อากาศ และ ขา เชื่อมต่อ กับ ไมโครคอนโทรลเลอร์ ที่มา: Cypress Semiconductor, Corp. (2017)	183
6.16	การ เชื่อม โยง ระหว่าง เวอร์ชัล แอดเดรส (ขวา) แอดเดรส ภายใน ภาพ (กลาง) และ VC/CPU บัส แอดเดรส (VC/CPU Bus Address) ที่มา: Broadcom Corp. (2012) หมายเหตุ VC: VideoCore	185
6.17	โครงสร้าง ภายใน ขา GPIO แต่ละ ขา ของ ชิป ตระกูล เดียว กับ BCM2835/2837/2711 ที่มา: Broadcom Corp. (2012)	189

6.18	ไดอะแกรมเวลาของโปรแกรมที่ต้องการเขื่อมต่อกับอุปกรณ์อินพุต/เอาต์พุตทำให้เกิดอินเทอร์รัปต์เกิดขึ้นเป็นระยะ ๆ ที่มา: virtualirfan.com	194
6.19	โครงสร้างส่วนหนึ่งภายในชิป BCM283x แสดงการเชื่อมต่อแกนประมวลผลต่าง ๆ กับวงจร Generic Interrupt Controller (GIC) ผ่าน ARM Advanced eXtensible Interface (AXI) หมายเหตุ APB: ARM Peripheral Bus ที่มา: arm.com	196
6.20	ไดอะแกรมเวลาการทำงานของ Generic Interrupt Controller (GIC) เพื่อตอบสนองต่อสัญญาณร้องขออินเทอร์รัปต์ที่มีความสำคัญไม่เท่ากัน ที่มา: ARM Ltd. (2017)	198
6.21	ลำดับการทำงานของ DMA Controller (DMAC) เพื่อควบคุมการอ่านข้อมูลจาก การ์ดหน่วยความจำ SD ไปบันทึกในหน่วยความจำหลัก (Main Memory)	199
6.22	โครงสร้าง ส่วน หนึ่ง ภายใน ชิป BCM283x/ BCM2711 แสดง การ เชื่อม ต่อ แกน ประมวลผลต่าง ๆ กับวงจร DMAC (DMA Controller) ผ่านวงจร Advanced eXtensible Interface (AXI) และ AXI-APB Bridge ที่มา: arm.com	200
6.23	การแปลงไฟกระแทรง 5 โวลต์ เป็นไฟกระแทรง ด้วยชิป PAM 2306 DC-DC Coverter คู่ กำหนดค่าเอาต์พุตด้วยค่าตัวเหนี่ยวนำ L ₁ และ L ₂ หน่วยเป็น ไมโคร เฮนรี ที่มา: Diodes, Inc. (2012)	202
7.1	โครงสร้างของระบบไฟล์บนอุปกรณ์ กีบ รักษาข้อมูลในระบบปฏิบัติการ ตระกูล ยูนิกซ์ ที่มา: Demblon and Spitzner (2004)	207
7.2	โครงสร้างของไอโหนดหนึ่งตัวและการเชื่อมโยงกับบล็อกข้อมูลของไฟล์หนึ่งไฟล์ ที่มา: Anderson and Dahlin (2012)	210
7.3	ชิปหน่วยความจำแฟลช NAND ผลิตโดยบริษัท Micron Technology โดยใช้ตัวถังชนิด TSOP (Thin Small Outline Package) จำนวน 48 ขา ที่มา: Micron Technology, Inc. (2004)	213
7.4	โครงสร้างภายในชิปหน่วยความจำแฟลช NAND ที่มา: Micron Technology, Inc. (2004)	214
7.5	ไดอะแกรมเวลาของ การ อ่าน ข้อมูล แบบ เพจ (Page Read) ของ หน่วย ความ จำ แฟลช NAND ที่มา: Micron Technology, Inc. (2004)	216
7.6	ไดอะแกรมเวลาของ การ เขียน ข้อมูล (Program Data) และ อ่าน สถานะ (Read Status) ของ หน่วย ความ จำ แฟลช NAND ที่มา: Micron Technology, Inc. (2004)	217
7.7	โครงสร้างภายใน การ์ด หน่วย ความ จำ SD ที่มา: SanDisk Corporation (2003)	220
7.8	ไดอะแกรมเวลาของ การ อ่าน ข้อมูล จำนวน หลาย บล็อก จาก การ์ด หน่วย ความ จำ SD ที่มา: SanDisk Corporation (2003)	221
7.9	ไดอะแกรมเวลาของ การ เขียน ข้อมูล จำนวน หลาย บล็อก จาก การ์ด หน่วย ความ จำ SD ที่มา: SanDisk Corporation (2003)	222

7.10	แผ่นวงจรพิมพ์ภายในอุปกรณ์ SSD ชนิด SATA III ประกอบด้วยชิพหน่วยความจำแฟลช ไอนามิกแรม ค่อนໂທຣເລອ່ວສໍາຮັບຄວບຄຸມ ທີ່ມາ: thatoldnews.site	223
7.11	ບລືອກ ໄດ້ອະແກຣມ ກາຍໃນ SSD ประกอบ ດ້ວຍ ສ່ວນ ເຊື່ອມ ຕ່ອງ ກັບ ເຄື່ອງ ໄນໂຄຣ ຂອນໂທຣເລອ່ວ ບັຟເຟອ່ວ ແລະ ມັນວ່າ ອາວນຈຳແພລື ທີ່ມາ: codecapsule.com	224
7.12	ໂຄຮສ້າງ ແລະ ອົງປະກອບຂອງອຸປະກຣນ໌ຫັບຕິສົກໄດຣົ່ງ (HDD) ຜົນດີ Serial ATA ທີ່ມາ: blogspot.com	225
7.13	ໂຄຮສ້າງ ຂອງ ຫັບຕິສົກໄດຣົ່ງ ແບ່ງ ເປັນ ໄຊ ລິນເດ ອົບ ແກ້ວກ ແລະ ເຊີກ ເຕົວ ທີ່ມາ: computableminds.com	227
8.1	ປະສົງທິກາພ ຂອງ ປະປະມວລ ພລ ດ້ວຍ ຈີປີພູ້ ຜົນດີ ມັດຕີ ຄອ້ວ ແລະ ແມນີ ຄອ້ວ ທີ່ມາ: royalsocietypublishing.org, John (2020)	232
8.2	ໂຄຮສ້າງ ກາຍໃນ ຂອງ ຈີປີພູ້ ARM ຕາມ ແທກໂນໂລຢີ big.LITTLE ເຊື່ອມ ຮ່ວ່າງ ແກນ ປະປະມວລ ພລ ດ້ວຍ Cache Coherent Interconnect (CCI) ທີ່ມາ: arm.com	234
8.3	ໂຄຮສ້າງ ຂອງ ຊູ້ເປົ້ອຮ່ວມພິວເຕົວ Fuqaku ປະກອບ ດ້ວຍ ຕູ້ແຮກ ຈຳນວນ 414 ຕູ້ ຖະ 384 ໂທນດ ແຕ່ລະ ໂທນດມື Fujitsu A64FX ຈຳນວນ 1 ຈີປີພູ້ ກາຍໃນ ປະກອບ ດ້ວຍ ຈີປີພູ້ ARM ຈຳນວນ 48 ແກນ ປະປະມວລ ພລ ທີ່ມາ: pcmag.com	235
8.4	ກາພຄ່າຍ ດາຍ ແລະ ບລືອກ ໄດ້ອະແກຣມ ຂອງ ຈີປີ Fujitsu A64FX ປະກອບ ດ້ວຍ ຈີປີພູ້ ARM ຈຳນວນ 48 ແກນ ປະປະມວລ ພລ ທີ່ມາ: fujitsu.com	236
8.5	ກາຮັດວຽກ ແບບ ຂະນາ ໂດຍອາຫັດ ອາວນ ພລ ຂອງ ຂໍ້ມູນ (Data Parallelism) ແລະ ອາວນ ພລ ຂອງ ອາວນ ພລ (Task Parallelism) ທີ່ມາ: manning.com	237
8.6	ຕ້ວອຍ່າງສ່ອງສົດ ໂັບການ ທຳມະນຸດ ແບບ ຂະນາ ແລະ ເຮັດ ປະກອບ ດ້ວຍ T0, T1 ແລະ T2 ພິມພື້ນຂໍ້ອາວນ Hello World ຕາມ ດ້ວຍ ມາຍເລີຂເຮັດ ປະຈຳຕົວ ໂດຍໃໝ່ໄລບຮາຣີ OpenMP ແກ້ໄຂຈາກ ທີ່ມາ: slideplayer.com	239
8.7	ກາຮັດວຽກ ພລ ອັດ ກອງ ວິຊີມ MergeSort ຕາມ ທັກ ພລ ແບ່ງ ແລະ ຍົດ ຄຮອງ ສາມາດ ນຳ ມາ ດັດແປລັງ ເປັນ ແບບ ອຸນຸກຮມ ແລະ ແບບ ຂະນາ ໄດ້ ໃນ ຕ້ວອຍ່າງ ທີ່ 8.3.2 ທີ່ມາ: slideshare.net	242
A.1	ໜ້າ ເວັບ ສໍາຮັບ ແປລັງ ເລຂຈຳນວນ ເຕີມ ຫຼູ້ານ ສອງ ເປັນ ຫຼູ້ານ ສີບ ອີ່ວິ້ວ ຫຼູ້ານ ສີບ ເປັນ ຫຼູ້ານ ສອງ ຢ່າຍ່າຍ ຜົນດີ	250
A.2	ກຣອກເລຂ -123 ລົງໃນ ກລ່ອງ ຂໍ້ອາວນ Decimal ເພື່ອ ໄທ້ ເປັນ ແບບ ແປລັງ ເລຂຈຳນວນ ເຕີມ -123 ເປັນ ເລຂຈຳນວນ ເຕີມ ຫຼູ້ານ ສອງ ມີ ເຄື່ອງ ມາຍ່າຍ ຜົນດີ 2's Complement	251
A.3	ຜລລັບຮັບ ພລ ເລຂ -123 ເປັນ ເລຂຈຳນວນ ເຕີມ ຫຼູ້ານ ສອງ ມີ ເຄື່ອງ ມາຍ່າຍ ຜົນດີ 2's Complement	251
A.4	ກາ ແປລັງ ເລຂ ຫຼູ້ານ ສອງ ມີ ເຄື່ອງ ມາຍ່າຍ 2's Complement 11111111 ອີ່ວິ້ວ ເທົກ ຫຼູ້ານ ສີບ ກົກ 0xFF	252

A.5	ผลลัพธ์การแปลงเลขฐานสองมีเครื่องหมายชนิด 2's Complement 11111111 หรือเท่ากับฐานสิบหก 0xFF	252
A.6	หน้าเว็บสำหรับศึกษาการทำงานของเครื่องคอมพิวเตอร์เลขจำนวนเต็มฐานสอง	253
A.7	ผลลัพธ์ ADD RC, RB, RA หรือ $RC = RB + RA$ โดย $RB=01111111111101_2$ และ $RA=1000000000000000_2$ ความยาว 16 บิต	254
A.8	ผลลัพธ์จากการแปลงเลข 123.0 ให้เป็นเลขทศนิยมฐานสองชนิด Single Precision	258
A.9	ผลลัพธ์ จาก การ แปลง เลข -123.0 ให้ เป็น เลข ทศนิยม ฐาน สอง ตาม มาตรฐาน IEEE754 ชนิด Single Precision	259
A.10	เมนูด้านล่างสุดของหน้าเว็บ เพื่อเลือกเลขทศนิยมฐานสองชนิด IEEE754 Single Precision (Binary32) และ Double Precision (Binary64)	259
A.11	ผลลัพธ์ จาก การ บวก เลข -123.0+123.0 ให้ เป็น เลข ทศนิยม ฐาน สอง ชนิด IEEE754 Single Precision	260
A.12	ผลลัพธ์ จาก การ คูณ เลข -123.0 × 123.0 ให้ เป็น เลข ทศนิยม ฐาน สอง ชนิด IEEE754 Single Precision	260
A.13	เว็บ สำหรับ การ ตอบ คำถาม เพื่อ สร้าง เลข หรือ แปลง เลข ฐาน สิบ ด้วย มาตรฐาน IEEE754 Single Precision การกดเลือกคือทำให้บิตนั้นเท่ากับ '1'	261
A.14	ผลลัพธ์จากการกรอกและแปลงตัวอักษร ໄ ຖ ຍ ກ ຂ ค a b c เป็นรหัสต่างๆ	264
B.1	รูปแสดงรายการอุปกรณ์สำหรับประกอบบอร์ด ที่มา: rs-online.com	268
B.2	บอร์ด Pi3 เมื่อติดฮีทชิป์เรียบร้อยแล้ว	269
B.3	แผ่นพลาสติก 5 ชิ้น สำหรับประกอบเป็นกล่องของบอร์ด Pi3	270
B.4	บอร์ด Pi ที่มีกล่องประกอบเรียบร้อยแล้ว	271
B.5	บอร์ด Pi เมื่อประกอบขาเขื่อมขยาย 2x20	271
B.6	การเชื่อมต่อคีย์บอร์ด และ เม้าส์กับช่องเสียบสาย USB บนบอร์ด Pi ให้เรียบร้อย . .	272
B.7	การเชื่อมต่อไฟเลี้ยงจากอแดปเตอร์ทางหัวไมโคร USB กับบอร์ด Pi	272
B.8	ภาพสีรุ้งบนจอยักษ์จากบอร์ด Pi3 ทำงานเป็นปกติ	273
C.1	หน้าต่าง ดาวน์โหลดไฟล์โปรแกรม Raspberry Pi Imager สำหรับติดตั้งระบบ ปฏิบัติการ Raspberry Pi OS ในการดูดหน่วยความจำไมโคร SD	276
C.2	หน้าต่างของโปรแกรม Raspberry Pi Imager	276
C.3	เมนูตัวเลือกใต้เมนู CHOOSE OS	277
C.4	เมนูตัวเลือกอื่น ๆ ใต้เมนู Raspberry Pi OS (other)	277
C.5	เมนูตัวเลือกอื่น ๆ ใต้เมนู Other general purpose OS	278
C.6	หน้าต่าง Imager	278
C.7	ปุ่ม WRITE ในหน้าต่าง Raspberry Pi Imager	278
C.8	หน้าต่างเตือนผู้ใช้ที่ต้องการเขียนทับการ์ดหน่วยความจำ SD	279

C.9	หน้าต่าง Raspberry Pi Imager ทายอยเขียนข้อมูลภายในการ์ด	279
C.10	หน้าต่าง Raspberry Pi Imager วนสอบ (Verify) ข้อมูลภายในการ์ด	279
C.11	ปุ่ม CONTINUE ในหน้าต่าง Raspberry Pi Imager เมื่อติดตั้งสำเร็จ	280
C.12	สอด การ์ด เข้าไปใน สล็อตบนบอร์ด Pi โดยหมายบอร์ดขึ้นมา โปรดสังเกต การ์ด หน่วยความจำจะต้องมีลักษณะดังรูป	280
C.13	หน้าต่าง Welcome ของระบบปฏิบัติการ Raspberry Pi OS	281
C.14	หน้าต่างตั้งค่าประเทศ โซนเวลา และภาษาในการใช้งานเมนูเป็นภาษาอังกฤษ	281
C.15	หน้าต่างสำหรับการเปลี่ยนรหัสผ่านใหม่ของ username ชื่อ pi โดยจะต้องกรอก รหัสผ่านใหม่ซ้ำจำนวน 2 ครั้ง	282
C.16	ตัวอย่างรายชื่อสัญญาณ WiFi รอบ ๆ ที่บอร์ด Pi มองเห็น ซึ่งจะแตกต่างกับของ ผู้อ่าน	282
C.17	หน้าต่างสำหรับกรอกพาสเวิร์ดของสัญญาณ WiFi ที่ต้องการเชื่อมต่อ	283
C.18	หน้าเพจเริ่มต้นของเว็บไซต์ www.raspberrypi.org	283
C.19	หน้าต่างสำหรับเมนู Shutdown เพื่อให้ผู้ใช้ ขัดดาวน์ รีบูต (Reboot) หรือล็อก เอาท์ (Logout)	284
D.1	หน้าต่างของไฟล์เมเนจเจอร์ (File Manager) ขณะที่เปิดไดเรกทอรีชื่อ /home/pi .	286
D.2	รูปไอคอนของโปรแกรม Terminal	287
D.3	หน้าต่างของโปรแกรม Terminal ซึ่งสามารถปรับแต่งสีพื้นและสีของตัวอักษรได้ .	287
D.4	หน้าต่างปรับแต่งสีพื้น (Background)	288
D.5	หน้าต่างปรับแต่งสีตัวอักษร (Foreground)	288
E.1	หน้าต่างเลือกชนิดโปรเจกต์ที่จะพัฒนาเป็นชนิด "Console application"	294
E.2	หน้าต่างเลือกภาษา C หรือ C++ สำหรับโปรเจกต์ที่จะพัฒนา	295
E.3	การเลือก คอนฟิก กูรูเรชัน (Configuration) Debug สำหรับ คอมไพเลอร์ GNU GCC ในโปรเจกต์ Lab5	295
E.4	การเปิดอ่านไฟล์ main.c ภายใต้โปรเจกต์ Lab5 ที่สร้างขึ้น	296
F.1	การย้ายไฟล์ main.c ออกจากโปรเจกต์	306
F.2	การเพิ่มไฟล์ใหม่ลงในโปรเจกต์	306
F.3	หน้าต่างกดปุ่ม "Yes" เพื่อยืนยัน	306
F.4	หน้าต่าง Save File ชื่อไฟล์ว่า main.s	307
I.1	หน้าต่าง Screen Layout Editor สำหรับกำหนดค่าต่าง ๆ กับพอร์ตแสดงผล HDMI .	340
I.2	หน้าต่าง Set Resolution สำหรับกำหนดความละเอียดหน้าจอแสดงผลที่ต้องการ .	340
I.3	หน้าต่าง Reboot needed กดปุ่ม Yes เมื่อต้องการรีบูต ณ เวลานั้น	341
I.4	หน้าต่าง Raspberry Pi Configuration	341

I.5	แท็บ Performance หน้าต่าง Raspberry Pi Configuration	342
I.6	หน้าต่างกำหนดขนาดหน่วยความจำสำหรับจีพียูที่ 16 MiB	342
I.7	หน้าต่างโปรแกรม raspi-config สำหรับบอร์ด Pi	343
I.8	เมนู System Options ในหน้าต่างโปรแกรม raspi-config สำหรับบอร์ด Pi	343
I.9	เมนู Audio ภายในเมนู System Options ในหน้าต่างโปรแกรม raspi-config สำหรับบอร์ด Pi	344
I.10	ผลลัพธ์ในหน้าต่างโปรแกรม speaker-test สำหรับบอร์ด Pi	344
I.11	โปรแกรม ALSA Mixer สำหรับควบคุมระดับเสียงทั้งด้านอินพุต (Capture: F4) และเอาต์พุต (Playback: F3) บนบอร์ด Pi	345
J.1	วงจรเชื่อมต่อหลอด LED กับบอร์ด Pi ในการทดลองที่ 10 เพื่อทดสอบว่าหลอด LED ทำงาน ที่มา: fritzing.org	358
K.1	วงจรสวิตซ์ปุ่มกดสำหรับทดลองการเขียนโปรแกรมอินเทอร์รัฟต์ในการทดลองที่ 11 ที่มา: fritzing.org	366
M.1	กราฟแสดงการใช้งานซีพียู (CPU Usage Monitor) ย้อนหลังและค่าสรุป ณ เวลาปัจจุบัน ที่มา: abload.de	382

บทที่ 1

บทนำ (Introduction)

คอมพิวเตอร์ ประกอบด้วย ฮาร์ดแวร์ และซอฟต์แวร์ต่าง ๆ มีประโยชน์ต่อเศรษฐกิจ การประกอบธุรกิจ อุตสาหกรรม การผลิต การศึกษา รวมถึงชีวิตประจำวันมากขึ้น สามารถใช้ประกอบการเรียน การสอนวิชาองค์ประกอบคอมพิวเตอร์ (Computer Organization) และ วิชาสถาปัตยกรรมคอมพิวเตอร์ (Computer Architecture) เป็นต้น โดยอาศัยบอร์ด Raspberry Pi3/Pi4 และชิปปี้ ARM รุ่น Cortex A53 และรุ่น Cortex A72 เป็นกรณีศึกษา อีกทั้งใช้เป็นเอกสารประกอบการพัฒนาโปรแกรมเบื้องต้นโดย มีฮาร์ดแวร์เสริมความเข้าใจ

1.1 ชนิดของเครื่องคอมพิวเตอร์

ระบบ คอมพิวเตอร์ ทั่วไป สามารถแบ่งออก เป็น คอมพิวเตอร์ตั้งโต๊ะ (Desktop Computer) คอมพิวเตอร์ เซิร์ฟเวอร์ หรือ แม่ข่าย (Server Computer) ซึ่ง เป็น คอมพิวเตอร์ (Supercomputer) คอมพิวเตอร์พกพา (Portable Computer) และ คอมพิวเตอร์ฝังตัว (Embedded Computer) หรือในชื่อ Internet of Things (IoT)

1.1.1 คอมพิวเตอร์ตั้งโต๊ะ (Desktop Computer)

คอมพิวเตอร์ตั้งโต๊ะ สามารถใช้งานได้หลากหลาย ขึ้นอยู่กับซอฟต์แวร์ที่นำมาติดตั้ง โดย ซอฟต์แวร์ ประยุกต์เหล่านี้ ขึ้นตรงกับซอฟต์แวร์ระบบปฏิบัติการ เป็นหลัก เช่น ระบบปฏิบัติการไมโครซอฟต์วินโดวส์ (Microsoft Windows) เวอร์ชันต่าง ๆ ระบบปฏิบัติการ MAC OS เวอร์ชัน 10.x ระบบปฏิบัติการลินุกซ์ (Linux) ซึ่งมีดิสทริบิวชัน (Distribution) เช่น Ubuntu SuSe RedHat เป็นต้น สามารถนี้จะอ้างอิงกับ ระบบปฏิบัติการ Raspberry Pi OS ซึ่งเป็นเวอร์ชันหนึ่งของระบบปฏิบัติการลินุกซ์สำหรับบอร์ด Raspberry Pi

1.1.2 คอมพิวเตอร์พกพา (Portable Computers)

คอมพิวเตอร์พกพา มีขนาดเล็ก และ พกพาง่าย ใช้กำลังไฟ จาก แบตเตอรี่ เป็นหลัก ได้แก่ คอมพิวเตอร์โน้ตบุ๊ค แท็บเล็ต คอมพิวเตอร์ และ โทรศัพท์เคลื่อนที่ สมาร์ตโฟน เป็นต้น ระบบปฏิบัติการ ที่ได้รับความนิยมสำหรับเครื่องเหล่านี้ ได้แก่ ระบบแอนดรอยด์ (Android) และ ระบบ Apple iOS ทั้งสองระบบ มีใช้งานโทรศัพท์สมาร์ตโฟน และ แท็บเล็ต มาากกว่า 90% ทั่วโลก ผู้ใช้สามารถติดตั้ง แอนดรอยด์ (Android) ซึ่ง พัฒนาต่อจาก ระบบปฏิบัติการ ลินุกซ์บนบอร์ด Pi3/Pi4 ได้ เช่น กัน รายละเอียดเพิ่มเติม สามารถศึกษา ได้จาก howtoraspberrypi.com

1.1.3 คอมพิวเตอร์ฝังตัว (Embedded Computer)

คอมพิวเตอร์ฝังตัว เป็นสมองกล ที่ซ่อนอยู่ ในระบบ คอมพิวเตอร์ชนิดนี้ สามารถทำงานอยู่ ใต้เงื่อนไขของ กำลังไฟ / สมรรถนะ / ราคา ปัจจุบัน คอมพิวเตอร์ฝังตัว ได้รับความนิยม เพื่อ ทำหน้าที่ เป็นเซนเซอร์ (Sensor) และ แขนกล (Actuator) สำหรับงาน ประภากลต่าง ๆ มากขึ้น และ เชื่อมต่อ กับ เครือข่าย อินเทอร์เน็ต ได้อย่าง แพร่หลาย ทำให้มีชื่อเรียกว่า Internet of Things หรือ IoT เช่น เครื่องบินไร้คนขับ (Unmanned Aerial Vehicle) โดรน (Drone) เป็นต้น



รูปที่ 1.1: ตัวอย่าง เครื่อง คอมพิวเตอร์ชนิด เชิร์ฟเวอร์ จาก บริษัท Hewlett Packard รุ่น Proliant ที่มา: datacenterknowledge.com

1.1.4 คอมพิวเตอร์เชิร์ฟเวอร์ หรือ แม่ข่าย (Server Computer)

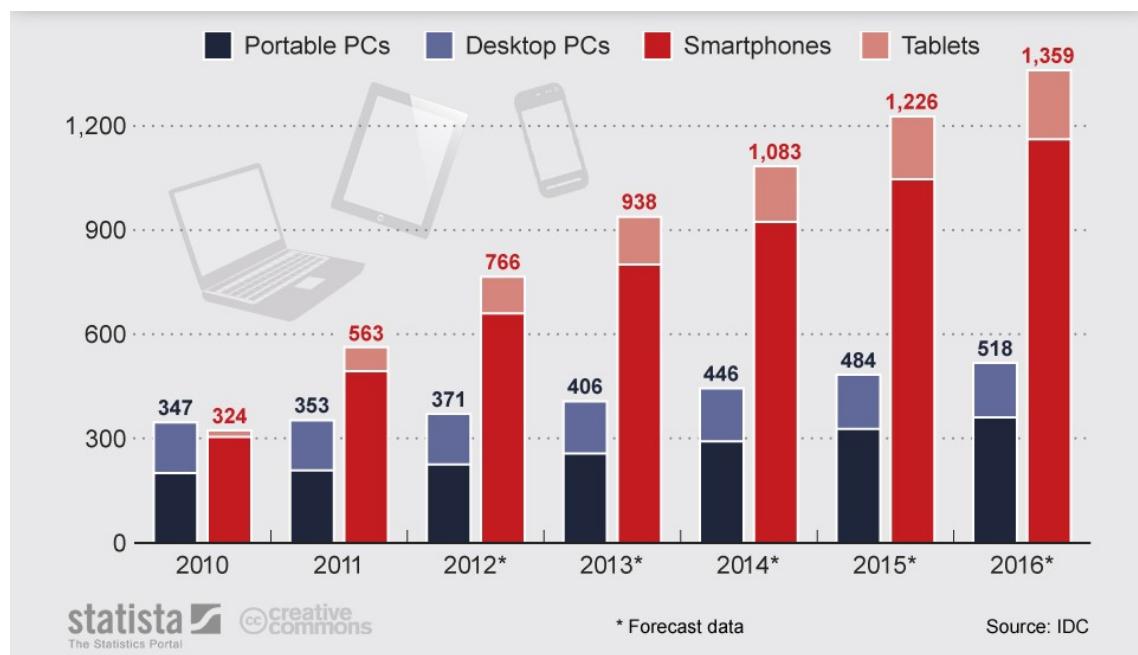
คอมพิวเตอร์ เชิร์ฟเวอร์ หรือ เครื่อง แม่ ข่าย จะ ต้อง ติด ตั้ง ระบบ ปฏิบัติ การ เช่น เดียว กับ เครื่อง คอมพิวเตอร์ ตั้ง โต๊ะ ยิ่ง ไป กว่า นั้น คอมพิวเตอร์ เชิร์ฟเวอร์ เหล่านี้ จะ ต้อง ให้ เปิด ให้ บริการ เครื่อง คอมพิวเตอร์ ลูก ข่าย (Client Computer) ผ่าน ทาง ระบบ เครือข่าย อินเทอร์เน็ต ตลอด เวลา ดังนั้น คอมพิวเตอร์ แม่ ข่าย จะ มี ความ จุ สมรรถนะ และ ความ เชื่อมั่น สูง เพื่อ รองรับ การ ใช้งาน จาก ผู้ใช้ เครื่อง คอมพิวเตอร์ จำนวนมาก ซึ่ง กระจาย อยู่ ทั่ว โลก ระบบ ปฏิบัติ การ ที่ ได้ รับ ความ นิยม สำหรับ เครื่อง เหล่านี้ ได้ แก่ ลินุกซ์ (Linux) และ

ไมโครซอฟต์วินโดวส์ (Microsoft Windows)

1.1.5 ชูเปอร์คอมพิวเตอร์ (Supercomputer)

คอมพิวเตอร์ที่มีสมรรถนะในการคำนวนสูงมาก ใช้งานด้านการประมวลผลข้อมูลขนาดใหญ่มาก (Big Data) ที่ได้จากการบันทึกของเครื่องเซิร์ฟเวอร์ และจากข้อมูลดิบจากการรวมของเครื่องคอมพิวเตอร์ ฝังตัว เพื่องานประยุกต์สำคัญ ๆ เช่น การจำลองการทำงานของยา วัคซีน ไวรัส การพยากรณ์อากาศ การใช้ปัญญาประดิษฐ์ (Artificial Intelligence) การเรียนรู้ของเครื่องจักร (Machine Learning) เป็นต้น

1.2 แนวโน้มของจำนวนอุปกรณ์คอมพิวเตอร์ชนิดต่างๆ



รูปที่ 1.2: ปริมาณยอดขาย (ล้านหน่วย) เครื่องคอมพิวเตอร์ชนิด เเดสก์ ทอป โน๊ตบุ๊ค สมาร์ตโฟน และแท็บเล็ตทั่วโลกในปี ค.ศ. 2010-2016 ที่มา: statista.com

แนวโน้มของจำนวนอุปกรณ์คอมพิวเตอร์ กลุ่มคอมพิวเตอร์ส่วนบุคคล ตั้ง โต๊ะ และ กลุ่มคอมพิวเตอร์ พกพา ประเภทสมาร์ตโฟน และ แท็บเล็ต ตั้งแต่ปี ค.ศ. 2005-2015 ตามกราฟในรูปที่ 1.2 ผู้อ่าน จะเห็นได้ว่า กลุ่มสมาร์ตโฟน และ แท็บเล็ต มีจำนวนเพิ่มมากขึ้นแบบก้าวกระโดด ในขณะที่ กลุ่มพีซี (PC: Personal Computer) หรือ คอมพิวเตอร์ ตั้ง โต๊ะ มีแนวโน้มลดลงไปเรื่อย ๆ โดย สมาร์ตโฟน ขนาดใหญ่ (Large Smartphone) จะได้รับความนิยมเพิ่มมากขึ้นเรื่อย ๆ ในขณะที่ ความนิยม สมาร์ตโฟน ขนาดเล็ก (Small Smartphone) มีแนวโน้มลดลง แท็บเล็ตขนาดใหญ่ (Large Tablet) และ แท็บเล็ตขนาดเล็ก (Small Tablet) มีแนวโน้มค่อนข้างคงที่

1.3 อุปกรณ์ดิจิทัลที่ใช้ชิปปี้ที่ออกแบบโดยบริษัท ARM

ตารางที่ 1.1: ชนิดและจำนวนอุปกรณ์ ($\times 10^6$) จำนวนชิป ($\times 10^6$) จำนวนชิปต่ออุปกรณ์ จำนวนชิป TAM (Total Available Market) ($\times 10^6$) จำนวนชิปของชิปปี้ ARM ($\times 10^6$) และเปอร์เซ็นต์ส่วนแบ่งการตลาด (Share) ของชิปปี้ ARM ในปี ค.ศ. 2010 ที่มา: 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	2-5	1,200	1,100	90%
	Feature Phone	1-3	1,900	1,700	90%
	Low End Voice	1	570	540	95%
	Portable Media Players	1-3	300	220	70%
	Mobile Computing* (apps only)	1	230	25	10%
Non-Mobile	PCs & Servers (apps only)	1	220	0	0%
	Digital Camera	1-2	200	160	80%
	Digital TV & Set-top-box	1-2	450	160	35%
	Networking	1-2	750	185	25%
	Printers	1	120	75	65%
	Hard Disk & Solid State Drives	1	670	560	85%
	Automotive	1	1,800	180	10%
	Smart Card	1	5,400	330	6%
	Microcontrollers	1	5,800	560	10%
	Others **	1	1,800	270	15%
Total	19,000		22,000	6,100	28%

ตารางที่ 1.1 และ 1.2 แสดงให้เห็นภาพรวมว่า จำนวนอุปกรณ์ทั่วโลกในปี 2010 เท่ากับ 19,000 ล้านตัวเพิ่มขึ้นเป็น 27,000 ล้านตัวในปี 2015 ซึ่งในปี 2010 บริษัท ARM มีส่วนแบ่งการตลาดของชิปชิปปี้ทั่วโลกเท่ากับ 28% คิดเป็นชิปชิปปี้ ARM จำนวน 6,100 ล้านชิปจากทั้งหมด (TAM 2010 Chips) 22,000 ล้านชิป และทั้งหมด (TAM 2015 Chips) เพิ่มขึ้นเป็น 34,000 ล้านชิปในปี 2015 ตัวเลขเหล่านี้รวมมาจากประเภทของสินค้าและอุปกรณ์ที่ใช้ชิปชิปปี้แบ่งเป็น **กลุ่มโทรศัพท์เคลื่อนที่ (Mobile)** และ **กลุ่มอื่นๆ (Non-Mobile)**

กลุ่มโทรศัพท์เคลื่อนที่แบ่งเป็น สมาร์ตโฟน (Smart Phone) ราคาสูง โทรศัพท์ใช้งานทั่วไป (Feature Phone) ราคาปานกลาง เครื่องเล่นสื่อพกพา (Portable Media Players) และ โน้ตบุ๊กและบันโทรศัพท์เคลื่อนที่ (Mobile Apps) โดยอุปกรณ์ที่ใช้ชิปปี้ ARM มีส่วนแบ่งการตลาดสูงที่สุด (95%) คือ Low End Voice รองลงมาคือ Smart Phone และ Feature Phone ตามลำดับ โทรศัพท์พื้นฐาน (Low End Voice) ราคาถูก คาดว่าตลาดของสมาร์ตโฟนจะเติบโตและได้รับความนิยมเพิ่มสูงขึ้นเรื่อยๆ รวมถึงตลาดของโน้ตบุ๊กและบัน

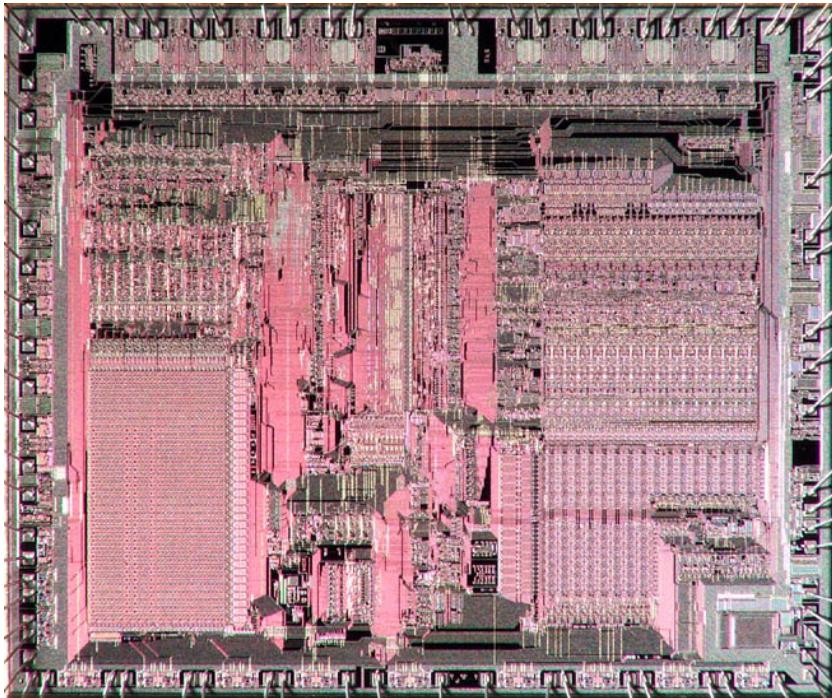
ตารางที่ 1.2: ชนิดอุปกรณ์ จำนวนชิป TAM (Total Available Market) ปี 2010 ($\times 10^6$) จำนวนชิปของ ชีพิญ ARM ปี 2010 ($\times 10^6$) จำนวนอุปกรณ์ TAM ปี 2015 ($\times 10^6$) จำนวนชิปต่ออุปกรณ์ และ จำนวนชิป TAM ปี 2015 ($\times 10^6$) ที่มา: 7boot.com หมายเหตุ TAM: Total Available Market

	Devices Shipped (Million of Units)	TAM 2010 Chips	10 ARM Share	TAM 2015 Devices	Chips/ Unit	TAM 2015 Chips
Mobile	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
Non-Mobile	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
Total		22,000	28%	27,000		34,000

ในส่วนของกลุ่ม อื่นๆ (Non-Mobile) ประกอบด้วย อุปกรณ์ที่หลากหลายกว่า โดย อุปกรณ์ที่ใช้ชีพิญ ARM มีส่วนแบ่งการตลาดสูงที่สุด (85%) คือ ฮาร์ดดิสก์ไดร์ฟและโซลิดสเตทไดร์ฟ (Hard Disk Drive& Solid State Drive) รองลงมา คือ กล้องดิจิทัลและพรินเตอร์ (Digital Camera และ Printers) ตามลำดับ โดย ชีพิญ ARM คาดว่าตลาดของฮาร์ดดิสก์ไดร์ฟและโซลิดสเตทไดร์ฟ จะเติบโตและได้รับความนิยมเพิ่ม สูงขึ้นเรื่อยๆ โดยเฉพาะโซลิดสเตทไดร์ฟ นอกจากนี้ ตลาดของทีวีดิจิทัลและกล่อง (Digital TV & Set Top Box) และตลาดของไมโครคอนโทรลเลอร์ (Microcontroller) น่าจะมีการขยายตัวในปี 2015 ตาม ตารางที่ 1.2 ส่วนตลาดใหม่ๆ ที่ชีพิญ ARM จะขยายตัวเพิ่ม คือ ชิปสำหรับเครื่องคอมพิวเตอร์โน๊ตบุ๊ค เช่น เครื่อง Apple MacBook Air และ Apple MacBook Pro ชิปสำหรับเครื่องคอมพิวเตอร์เซิร์ฟเวอร์ภายใน องค์กร และชิปสำหรับเครื่องซูเปอร์คอมพิวเตอร์ เพื่อรับรองการคำนวณแบบขนาดใหญ่มีรายละเอียดเพิ่มเติมในบทที่ 8

1.4 คอมพิวเตอร์บอร์ดเดี่ยว (Single Board) ตระกูล Raspberry Pi

คอมพิวเตอร์บอร์ดเดี่ยว (Single Board) ตระกูล Raspberry Pi ถูกออกแบบให้มีต้นทุนต่ำ ทำให้ราคา ถูกเหมาะสมกับการศึกษาและงานที่มีความซับซ้อนน้อย โดยบอร์ดตัวแรกอาศัยชิป BCM2835 เป็นชิปหลัก ตั้งแต่ปี 2012 ในปี 2015 บอร์ด Raspberry Pi2 ใช้ชิป BCM2836 ปี 2016 บอร์ด Raspberry Pi3 ใช้



รูปที่ 1.3: ภาพถ่ายซิลิกอนดาย (Silicon Die) ของชิป BCM2835 บนบอร์ด Raspberry Pi ที่มา: raspberrypi.org

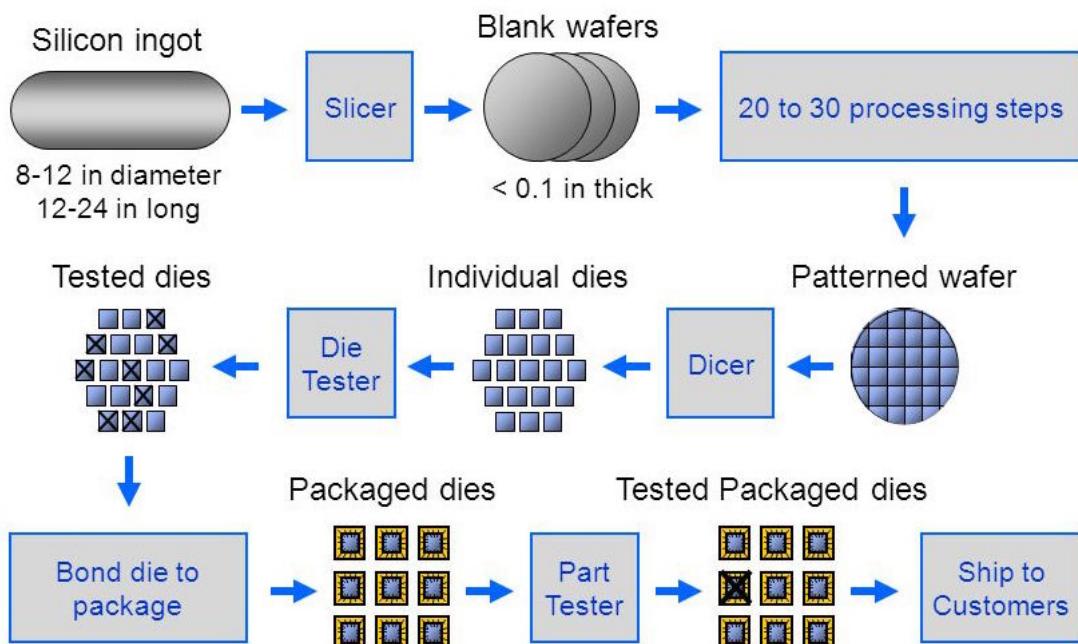
ชิป BCM2837 ล่าสุดปี 2019 บอร์ด Raspberry Pi4 ใช้ชิป BCM2711 โดยบริษัท Broadcom ประเทศสหรัฐอเมริกา เป็นผู้ผลิตชิปทั้งหมดนี้ รูปที่ 1.3 คือ ภาพถ่ายซิลิกอนดาย (Silicon Die) หรือแผ่นวงจรซิลิกอนของ BCM2835 ขนาดประมาณ 50-100 เท่าของซิปจริง โดยไม่มีแพ็คเกจพลาสติกสีดำห่อหุ้ม ชิป BCM2835 มีคุณลักษณะเฉพาะ (Specification) ดังนี้

- ชิปปี้ ARM1176JZF-S จำนวน 1 แกนประมวลผล (Core) ทำงานที่สัญญาณคล็อก (Clock) ความถี่ 700 เมกะเฮิรตซ์ (MHz) แคชลำดับที่ 1 (Level 1 ย่อว่า L1) ขนาด 16 KiB (KibiByte) แคชลำดับที่ 2 (Level 2 ย่อว่า L2) ขนาด 128 KiB (kibibyte)
- หน่วยประมวลผลด้านกราฟิก หรือจีปีย์ (GPU) ชื่อ VideoCore IV (4) ภายในชิปตัวเดียว กัน รายละเอียดเพิ่มเติม: [wikipedia](https://en.wikipedia.org/wiki/Raspberry_Pi)

บอร์ด Pi3/Pi4 ใช้เทคโนโลยีที่สูงขึ้น สมรรถนะและประสิทธิภาพเพิ่มขึ้น โดยมีจำนวน 4 แกนประมวลผล (Core) รายละเอียดเพิ่มเติมในบทที่ 3 บอร์ด Pi4 ใช้ชิปปี้ จำนวน 4 แกน ประมวลผล เท่าเดิม แต่มีสมรรถนะ และ ประสิทธิภาพ สูง ขึ้น ผู้อ่านสามารถ ค้นคว้าราย ละเอียด ทั้งหมด ของ บอร์ด ตระกูล นี้ได้ที่ github.com อย่างเป็นทางการ

1.5 ขั้นตอนการผลิตชิป (Chip) หรือวงจรรวม (Integrated Circuit)

ภายในเครื่องคอมพิวเตอร์ชนิดต่าง ๆ มีชิป หรือไมโครชิป (Microchip) ซึ่งในอดีตอาจเรียกว่า วงจรรวม (Integrated Circuit ย่อว่า IC) หรือ ไอซี เมื่อผู้ออกแบบระบบวงจรดิจิทัลซึ่งประกอบด้วยวงจรลอกิจเกตและวงจรทรานซิสเตอร์เข้ามาเพิ่มเป็นจำนวนมากขึ้น คำว่า IC จึงเปลี่ยนเป็น VLSI (Very Large System Integration) และ ULSI (Ultra Large System Integration) ตามลำดับ ภายในชิปจะมีวงจรดิจิทัลวงจรลอกิจเกตต่าง ๆ จำนวนมาก ทำให้มีจำนวนทรานซิสเตอร์มากตามและความสามารถที่หลากหลาย เช่น ชิปสามารถเข้ามือต่อกับจอแสดงผล อุปกรณ์และบูมต่าง ๆ ให้กล้ายเป็นเครื่องคอมพิวเตอร์ขนาดเล็กลงเรื่อย ๆ จนปัจจุบันมีชื่อเรียกใหม่ว่า SoC (System on Chip) ผู้อ่านควรทำความรู้จักขั้นตอนการผลิตชิปเหล่านี้เบื้องต้น เพราะให้เข้าใจวิวัฒนาการของ SoC ต่อไปในอนาคต



รูปที่ 1.4: ขั้นตอนการผลิตชิป (Chip) ภายในประกอบด้วยวงจรรวม ไมโครโปรเซสเซอร์ และอื่น ๆ ที่มา: slideplayer.com

ขั้นตอนการผลิตชิป (Chip) ตามรูปที่ 1.4 มีลักษณะคล้ายการทำแผ่นวงจรพิมพ์แต่มีความซับซ้อนและเทคโนโลยีในการผลิตสูง โดยเริ่มต้นจากการนำแท่งผลึกซิลิกอน (Silicon ingot) บริสุทธิ์มาทำการสะอาดแล้วขัดกลึงให้เป็นรูปทรงกระบอกขนาดเส้นผ่าศูนย์กลาง 10-12 นิ้ว แล้วจึงแล่หรือสไลซ์ (Slice) แท่งผลึกนี้ให้เป็นแผ่นวงกลมที่บางมาก เรียกว่า เวเฟอร์เปล่า (Blank Wafer) เมื่อนำแผ่นเวเฟอร์เปล่าเหล่านี้ไปผ่านขั้นตอนการปั๊กถ่ายสาร (Doping) และอื่น ๆ จำนวน 20-40 ขั้นตอน จนกลายเป็นแผ่นเวเฟอร์ที่มีลวดลาย (Patterned wafers) โดยเวเฟอร์หนึ่งแผ่นประกอบด้วย ตาย (Die) จำนวนมากหนึ่งกระจายในลักษณะตารางดังรูปที่ 1.4 หลังจากทดสอบโดยแต่ละตัวด้วยตัวทดสอบเวเฟอร์ (Wafer Tester) ด้วยตัวที่ไม่ผ่านการทดสอบจะถูกกำกับหากเพื่อทำการซ่อมแซม แล้ว แผ่นเวเฟอร์จะถูกตัดด้วยได

เซอร์ (Dicer) ให้ด้วยแยกตัวออกจากกันเป็นสี่เหลี่ยม ด้วยตัวที่ผ่านการทดสอบก็จะถูกยึด (Bond) เข้ากับตัวถัง (Package) เชื่อมต่อขา กับตัวถังด้วยลวดทองคำ ให้ตรงตามลักษณะตัวถังที่ต้องการ หลังจากการทดสอบด้วย Part Tester เพื่อทดสอบหาซิปที่ไม่ผ่าน เมื่อพบแล้วเครื่องจะทำเครื่องหมายและไม่ถูกส่ง (Ship) ไปยังลูกค้า สรุปคือ ชิป (Chip) คือ ด้วยที่ห่อหุ้มด้วยตัวถังและผ่านการทดสอบคุณสมบัติทางไฟฟ้า และทางกลตามมาตรฐานเรียบร้อยแล้ว

ซิลิกอน คือ ธาตุหมู่ที่ 4 มีวาเลนซ์อิเล็กตรอน (Valence Electron) วงนอกสุดจำนวน 4 ตัว แผ่นซิลิกอนเฟอร์ คือ การนำธาตุซิลิกอนที่สกัดและหลอมจนเป็นแท่ง (Silicon Ingot) มาแล้วเป็นแผ่นบาง ๆ การโอดีสาร คือ การเปลี่ยนแปลงแผ่นพื้นที่บนแผ่นเฟอร์เปล่าบางตำแหน่งเพื่อให้พื้นที่นั้นกลายเป็นสาร P โดยการเพิ่มไฮล (Hole) และกลายเป็นสาร N โดยการเพิ่มอิเล็กตรอน ตามลำดับ ทรานซิสเตอร์ คือ การเชื่อมต่อพื้นที่ P และพื้นที่ N บนเฟอร์ ตามรูปแบบมาตรฐาน โลจิกเกต (Logic Gate) คือ การนำทรานซิสเตอร์มาเชื่อมต่อกันตามรูปแบบมาตรฐาน เช่น แอนด์เกต (And Gate) แอนด์เกต (Nand Gate) เป็นต้น เพื่อทำงานตามพีช คณิต บูลีน (Boolean Algebra) ซึ่งผู้อ่านเรียนรู้ในวิชาการออกแบบวงจรดิจิทัล มอตูล คือ การเชื่อมต่อกันของโลจิกเกตเป็นวงจรจนกลายเป็นวงจรดิจิทัลที่มีความซับซ้อนตามที่กล่าวไปก่อนหน้านี้ โดยใช้ทองแดงหรืออลูมิเนียม (Aluminum) ตัวนำไฟฟ้าเชื่อมโยงทรานซิสเตอร์และวงจรเกตต่าง ๆ เข้าด้วยกัน ในแนวราบ เรียกว่า อินเตอร์คอนเนคต์ (Interconnect) และเรียกตัวนำในแนวตั้ง เรียกว่า เวีย (Via) ซึ่งคำศัพท์เหล่านี้จะปรากฏในบทต่อไป โดยเฉพาะบทที่ 5

ผู้อ่านที่สนใจเทคโนโลยีเชิงลึกกว่าที่述ราเล่มนี้จะอธิบายได้ สามารถศึกษาเพิ่มเติมจากตัวอย่างคลิปวิดีโอเหล่านี้

- **Silicon Run Lite** ทางเว็บ [youtube.com](https://www.youtube.com/watch?v=JyfXzrjwvIY) ซึ่งถ่ายทำพื้นฐาน และ วัสดุ ภาพ แอนิเมชัน ประกอบตั้งแต่ การสร้าง และ ทดสอบ ชิป การ ประกอบ และ ทดสอบ บน แผ่น วงจร พิมพ์ จนถึง ประกอบ และ ทดสอบ เครื่องคอมพิวเตอร์ ถึงแม้เทคโนโลยีจะล้าสมัย เพราะถ่ายทำเมื่อ 30 ปีที่แล้วเพื่อการศึกษา แต่เนื้อหาและหลักการไม่ต่างจากเทคโนโลยีในปัจจุบันมากนัก
- **Chip Manufacturing How are Microchips made?** ทางเว็บ [youtube.com](https://www.youtube.com/watch?v=JyfXzrjwvIY) โดยบริษัท Infineon ประเทศเยอรมนี ซึ่งเป็นผู้ผลิตชิปชั้นนำของโลก ด้วยเทคโนโลยี และ อุปกรณ์ที่ทันสมัย และ ใกล้เคียงกับปัจจุบัน
- **How they make Raspberry Pi in the UK** โดยวารสาร Electronics Weekly ถ่ายทำโดย Metropolis Multimedia ทางเว็บ [youtube.com](https://www.youtube.com/watch?v=JyfXzrjwvIY) อธิบายขั้นตอน การผลิต และ ประกอบ บอร์ด Raspberry Pi โดยละเอียด ซึ่งโรงงานใช้เครื่องจักรที่ทันสมัยขึ้น

1.6 สรุปท้ายบท

รูปแบบของเครื่องคอมพิวเตอร์มีความหลากหลายตามการประยุกต์ใช้งานในระบบต่าง ๆ นอกเหนือจาก เครื่องคอมพิวเตอร์ที่มองเห็นทั่วไป ใน การ คุณภาพ ขนาด ต่าง ๆ ยัง มี คอมพิวเตอร์ ภายใน รถยนต์ รถยนต์ไฟฟ้า หุ่นยนต์ต่าง ๆ เครื่องบิน อากาศยานไร้คนขับ (Unmanned Aeronautic Vehicle: UAV) โดรน (Drone) เป็นต้น ใน การ ตรวจ วัด ค่า สิ่ง แวดล้อม เช่น ลม ฝน คุณภาพ อากาศ การ พัฒนา ระบบ คอมพิวเตอร์ เหล่านี้ จึง ต้อง อาศัย ความ รู้ ความเข้าใจ ทั้ง ฮาร์ดแวร์ และ ซอฟต์แวร์ ควบคู่ กัน ไป เพื่อ ให้ ระบบ ทำงาน ได้ เต็ม ประสิทธิภาพ มี อายุ การ ใช้งาน ที่ เหมาะสม สม และ คุ้ม ค่า การ ลงทุน

1.7 คำถามท้ายบท

1. จง ให้ เหตุผล ว่า ทำไม คอมพิวเตอร์ ชนิด แท็บเล็ต และ สมาร์ตโฟน จึง ได้ รับ ความนิยม เพิ่มขึ้น จน เกือบ แทน ที่ คอมพิวเตอร์ ชนิด ตั้ง โต๊ะ และ โน้ตบุ๊ก ใน แห่ง มุ่ง ดัง ต่อไปนี้
 - ขนาด ของ อุปกรณ์
 - การ เชื่อม ต่อ กับ ผู้ ใช้ (Human Interface)
 - ราคา และ โปรโมชัน การ ขาย
 - อื่น ๆ
2. เรา สามารถ ใช้ เครื่อง คอมพิวเตอร์ ชนิด ตั้ง โต๊ะ ทดแทน เครื่อง ชนิด เซิร์ฟเวอร์ ได้ หรือ ไม่ เพราะ เหตุ ใด จง บอก เหตุผล ข้อดี และ ข้อเสีย ใน แห่ง มุ่ง ดัง ต่อไปนี้
 - ความ เชื่อมั่น ของ อุปกรณ์ (Reliability)
 - คุณสมบัติ จำเพาะ ของ อุปกรณ์ (Specification)
 - การ ลงทุน และ ค่า ใช้จ่าย อื่น ๆ
 - ระบบปฏิบัติ การ
 - อื่น ๆ
3. เหตุ ใด การ ผลิต ชิป จำเป็น ต้อง ควบคุม อากาศ ใน ด้าน ความ สะอาด ที่ เรียกว่า ห้อง คลีนรูม (Clean Room)
4. จง บอก สภาพ แวดล้อม ที่ ไม่ เหมาะสม ต่อ การ ทำงาน ของ เครื่อง คอมพิวเตอร์ ชนิด ต่าง ๆ ใน แห่ง มุ่ง ดัง ต่อไปนี้
 - อุณหภูมิ
 - ความชื้น

- สนามแม่เหล็กและสนามไฟฟ้า
- ความสูงจากระดับน้ำทะเล
- อื่น ๆ

5. การระบายน้ำร้อนของอุปกรณ์คอมพิวเตอร์มีความสำคัญต่อการทำงาน จงบอกข้อดีและข้อเสียของวิธีต่อไปนี้

- ฮีทซิงก์ (Heat Sink)
- พัดลม
- ของเหลว
- อื่น ๆ

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

7. จงค้นคว้าในอินเทอร์เน็ต ว่า SoC ในปัจจุบันมีจำนวนทรานซิสเตอร์สูงสุด เป็นจำนวนกี่พันล้านตัว ต่อชิป

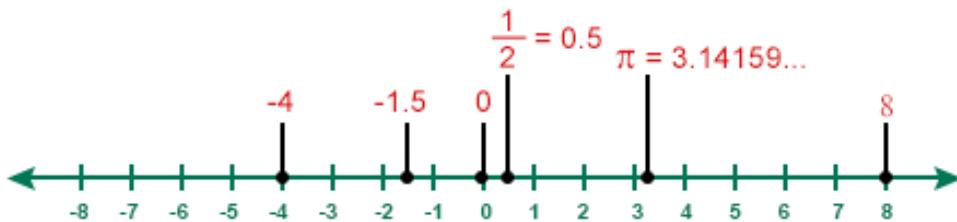
บทที่ 2

ข้อมูลและเลขคณิตคอมพิวเตอร์ (Computer Data and Arithmetic)

ซีพียูหรือไมโครโปรเซสเซอร์ (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 ไมโครโปรเซสเซอร์ในอดีตจะบันจำเป็นต้องคำนวณตัวเลขทั้งจำนวนเต็ม และ ทศนิยม เพื่อรองรับ การใช้งานทาง คณิตศาสตร์ และ ตรรกศาสตร์ ได้หลากหลายและซับซ้อนมากขึ้น

ในปัจจุบัน การใช้งาน เครื่อง คอมพิวเตอร์ ชนิด ต่าง ๆ เพื่อ ประมวล ผล ข้อมูล และ อักษร ให้ กล้าย เป็น อินฟอร์เมชัน (Information) ที่ เป็น ประโยชน์ ต่อ ธุรกิจ การค้า สื่อสังคม ออนไลน์ และ วัตถุ ประสงค์ อื่น ๆ การ ประมวล ผล ข้อมูล และ อักษร เหล่านี้ จะ เป็น ต้อง ใช้ คณิตศาสตร์ เช่น การ บวก/ลบ คูณ/หาร และ และ ตรรกศาสตร์ ด้วย วงจร ดิจิทัล ซึ่ง จัด เป็น ฮาร์ดแวร์ เพื่อ ให้ ใช้ เวลา คำนวณ น้อย ที่สุด โดย มุ่ง ยื่น เข้า ใจ ข้อมูล เชิง จำนวน ด้วย ตัว เลข ฐานสิบ แต่ คอมพิวเตอร์ จำ เป็น ต้อง ประมวล ผล ด้วย เลข ฐานสอง แทน ดังนั้น วัตถุ ประสงค์ ของ บทนี้ คือ

- เพื่อ ให้ ผู้ อ่าน เข้า ใจ รูป แบบ และ คณิตศาสตร์ ของ เลข จำนวน เต็ม ชนิด ไม่มี เครื่องหมาย และ มี เครื่องหมาย (Integer: Unsigned and Signed) ใน ภาษา คอมพิวเตอร์
- เพื่อ ให้ ผู้ อ่าน เข้า ใจ รูป แบบ และ คณิตศาสตร์ ของ เลข ทศนิยม ฐานสอง ชนิด จุด ตรึง (Fixed-Point) ใน ภาษา คอมพิวเตอร์
- เพื่อ ให้ ผู้ อ่าน เข้า ใจ รูป แบบ และ คณิตศาสตร์ ของ เลข ทศนิยม ฐานสอง ชนิด จุด ลอย ตัว (Floating-Point) ตาม มาตรฐาน IEEE754 ใน ภาษา คอมพิวเตอร์
- เพื่อ ให้ ผู้ อ่าน รู้ จัก รหัส เลข ฐานสอง ตาม มาตรฐาน ASCII และ Unicode สำหรับ ข้อมูล ตัว อักษร ซึ่ง และ ข้อ ความ ต่าง ๆ ใน เครื่อง คอมพิวเตอร์ และ ใน เครื่อง ข่าย อินเทอร์เน็ต



รูปที่ 2.1: เลขจำนวนเต็ม และ เลขจำนวนจริง บนเส้นจำนวน ใน คณิตศาสตร์ เลขฐานสิบ ที่มา: tutorvista.com

ดังนั้น เพื่อให้คอมพิวเตอร์สามารถคำนวณ หรือ กระทำการ เชิงคณิตศาสตร์ กับ เลขฐานสิบ ที่ เป็นจำนวนเต็ม หรือ จำนวนจริง ดังรูปเส้นจำนวน ในรูปที่ 2.1 ในรูปของ เลขฐานสอง ได้ เลขฐานสอง จึง แบ่ง เป็น **เลขจำนวนเต็ม** (Integer) และ **เลขจำนวนจริง** (Real number) คล้าย กับ เลขฐานสิบ ส่วนตัว อักษรแต่ละตัวที่มีนัยยะอ่านเข้าใจ คือ เลขฐานสอง เช่นเดียวกัน

ทั้งนี้เนื่องจากข้อมูล และ คำสั่ง จะอยู่ในรูปของ ระดับความต่าง ศักย์ หรือ โวลเตจ หรือ ทิศทางของการ ไฟลของกระแสไฟฟ้า เป็นเลข '1' หรือ '0' เรียกว่า บิต (bit: binary digit) ตามลำดับ ซึ่งการจัดเรียงบิต ข้อมูลเหล่านี้ หลาย ๆ บิต ให้กลายเป็น เลขฐานสอง และ เลขฐานสิบ หลัง จึงจำเป็นต้องใช้องค์ความรู้ด้าน คณิตศาสตร์ คอมพิวเตอร์ (Computer Arithmetic)

2.1 ข้อมูลพื้นฐานชนิดต่างในภาษา C/C++

ตารางที่ 2.1: ชนิด ความยาว (บิต) ค่าฐานสิบ ต่ำสุด และ สูงสุด ของ ตัวแปรพื้นฐานแต่ละชนิด ในภาษา C/C++ หมายเหตุ ค่าต่ำสุด และ ค่าสูงสุด ของ IEEE754 เป็นค่าเลขอนormalize (Normalize) เพื่อแสดงความ สมมาตรของเลขยกกำลัง ที่มา: ntu.edu.sg

ชนิด	ความยาว(บิต)	ค่าต่ำสุด ₁₀	ค่าสูงสุด ₁₀
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	$\pm 2^{-1023} = \pm 2.23 \times 10^{-308}$	$\pm 2 \times 2^{1023} = \pm 1.80 \times 10^{308}$

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

ในขณะที่ตัวแปรชนิด **int** (Integer) นั้น ซึ่งมีส่วนใหญ่ต้องการพื้นที่จัดเก็บตัวแปรชนิด **int** นี้ 1 ตัว เป็นพื้นที่ 32 บิต หรือ 4 ไบต์ โดยมีค่าฐานสิบต่ำสุดเท่ากับ -2^{31} หรือ -2,147,483,648 (ลบสองพันหนึ่งร้อยสี่สิบเจ็ดล้านสี่แสนแปดหมื่นสามพันหกร้อยสี่สิบแปด) และมีค่าฐานสิบสูงสุดเท่ากับ $+2^{31} - 1$ หรือ +2,147,483,647 (บวกสองพันหนึ่งร้อยสี่สิบเจ็ดล้านสี่แสนแปดหมื่นสามพันหกร้อยสี่สิบเจ็ด)

ตัวอย่าง การประกาศตัวแปรและตั้งค่าเริ่มต้นในภาษาคอมพิวเตอร์ เช่น ภาษา C/C++ และ Java ผู้เขียนหรือนักพัฒนาจะต้องประกาศชื่อตัวแปร คล้ายกับการตั้งชื่อตัวละคร แล้วจึงสามารถตั้งค่าเริ่มต้น (Initialize) ให้กับตัวแปรต่าง ๆ เมื่อโปรแกรมทำงาน จึงนำค่าเริ่มต้นของตัวแปรนั้นไปใช้งาน และอาจเปลี่ยนแปลงค่าจากการคำนวณหรือประมวลผลตามเงื่อนไขต่าง ๆ ที่เกิดขึ้นระหว่างที่โปรแกรมทำงาน ยกตัวอย่าง เช่น ประโยคภาษา C/C++ เหล่านี้ โดยสัญลักษณ์ /* */ ใช้ครอบประโยคคอมเม้นต์ (Comment) สำหรับให้โปรแกรมเมอร์อธิบายประโยคภาษาคอมพิวเตอร์ให้มนุษย์อ่านเข้าใจง่ายขึ้น

ตัวอย่างที่ 2.1.1 การประกาศตัวแปรชนิดสำคัญ ๆ และตั้งค่าเริ่มต้นด้วยภาษา C/C++

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

Address (Byte #)	Data	Variable Name
0x5A	01000010	
0x59	00101000	
0x58	00000000	
0x57	00000000	
0x56	00000000	a
0x55	00000000	
0x54	00000000	
0x53	00101010	
0x52	00000000	z
0x51	00101010	
0x50	00101010	y
0x4f		x
Memory		

รูปที่ 2.2: พื้นที่ในหน่วยความจำซึ่งบรรจุค่าเริ่มต้นของตัวแปรที่ได้ประกาศในตัวอย่างที่ 2.1.1

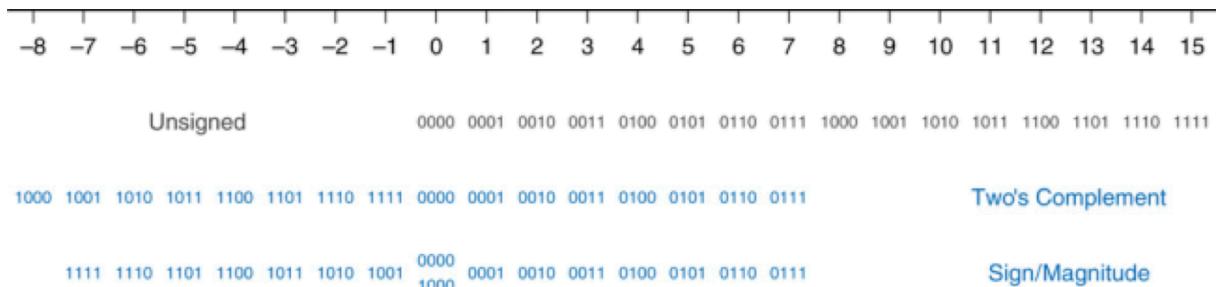
ผู้อ่านสามารถทำความเข้าใจโดยคิดประการตัวแปรและตั้งค่าเริ่มต้น ด้วยภาษา C/C++ จากตัวอย่างที่ [2.1.1](#) ตามลำดับ ดังนี้

- x เป็นตัวแปรชนิด char มีค่าเริ่มต้นเป็นตัวอักษร '*' ซึ่งตรงกับจำนวนเต็มชนิดมีเครื่องหมายมีค่าเริ่มต้นเท่ากับ 42 แต่ในหน่วยความจำขนาด 8 บิตที่จองไว้สำหรับตัวแปร x มีค่าเท่ากับ 00101010_2 หรือเท่ากับ $2A_{16}$ โดยที่เลขฐานสิบหก 1 ตัวเท่ากับเลขฐานสองจำนวน 4 บิต และสัญลักษณ์ 0x หมายถึงเลขฐานสิบหกในภาษา C/C++ ผู้อ่านสามารถเปิดตารางรหัส ASCII ในรูปที่ [2.12](#)
- y เป็นตัวแปรชนิด จำนวนเต็มชนิดมีเครื่องหมาย short มีค่าเริ่มต้นเท่ากับ 42_{10} แต่ในหน่วยความจำขนาด 16 บิตที่จองไว้สำหรับตัวแปร y มีค่าเท่ากับ 0000000000101010_2 หรือเท่ากับ $002A_{16}$
- z เป็นตัวแปรชนิด จำนวนเต็มชนิดไม่มีเครื่องหมาย unsigned long มีค่าเริ่มต้นเท่ากับ 0 แต่ในหน่วยความจำขนาด 32 บิตที่จองไว้สำหรับตัวแปร z มีค่าเท่ากับ $00000002A_{16}$
- a เป็นตัวแปรชนิดทศนิยม float มีค่าเริ่มต้นเท่ากับ 42.0_{10} ในความคิดของโปรแกรมเมอร์ ในการเป็นจริงตัวแปร a มีค่าเท่ากับ 42280000_{16} ตามมาตรฐาน IEEE754 ตัวแปรชนิดนี้ต้องการพื้นที่ในหน่วยความจำขนาด 32 บิต หรือ 4 ไบต์ เริ่มต้นที่ 42 หรือ $0x57$ และส่วนท้ายของตัวแปรจะเป็นเลขฐานสิบหก $0x5A$ ผู้อ่านสามารถทำความเข้าใจการแปลง 42280000_{16} เป็นเลขฐานสิบเท่ากับ 42.0_{10} ได้ในตัวอย่างที่ [2.6.1](#)

ก่อนที่โปรแกรม จะเริ่มทำงาน ระบบปฏิบัติการจะ จด พื้นที่ของ หน่วยความจำในลักษณะคล้าย กับรูปที่ [2.2](#) ซึ่งหมายเลขไบต์ (Byte Number) หรือแอดdress (Address) ในหน่วยความจำ (Memory) จะเรียกว่าเป็นชั้น ๆ ตั้งแต่หมายเลขไบต์ที่ 0 จำกัด้านล่างขึ้นไปหมายเลขไบต์ที่สูงขึ้นคล้ายกับตึกสูง แต่เริ่มต้นนับจากชั้นที่ 0 เสมอ และแต่ละชั้นบรรจุข้อมูลได้เพียง 1 ไบต์ หากต้องการพื้นที่มากกว่า 1 ไบต์ วงจรสามารถนำพื้นที่ของชั้นถัดไปมาใช้งานด้วย ยกตัวอย่าง เช่น ตัวแปร y และ z ในรูปที่ [2.2](#) ต้องการพื้นที่ 2 และ 4 ไบต์ ตามชนิดของตัวแปรในตารางที่ [2.1](#) ในบทนี้ ผู้อ่านจะได้ทำความเข้าใจกับข้อมูลชนิดจำนวนเต็ม ก่อน เพราะเป็นพื้นฐานของข้อมูลชนิดอื่น ๆ และทำความเข้าใจกับเนื้อหาด้วยการปฏิบัติตาม การทดลองที่ 1 ในภาคผนวก A

2.2 เลขจำนวนเต็มฐานสอง: รูปแบบและค่าฐานสิบ

เลขจำนวนเต็มฐานสอง (Integer) แบ่งเป็นเลขจำนวนเต็มไม่มีเครื่องหมาย (Unsigned Integer) และจำนวนเต็มมีเครื่องหมาย (Signed Integer) ซึ่งต้องมีกระบวนการการบวกและลบ การคูณและหาร และการตรวจจับโอเวอร์เฟล์ (Overflow Detection) เนื่องจากการคำนวณด้วยวงจรดิจิทัลภายในชิปจะต้องกำหนดจำนวนบิตที่ซัดเจนล่วงหน้า เช่น 32 หรือ 64 บิต



รูปที่ 2.3: เส้นจำนวนเปรียบเทียบข้อมูลเลขฐานสองความยาว 4 บิตในรูปแบบของเลขจำนวนเต็มชนิดไม่มีเครื่องหมาย (Unsigned) ชนิดมีเครื่องหมาย และ 2's Complement และชนิดมีเครื่องหมายแบบ 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 หมายความว่า การใช้นับจำนวนสิ่งของต่าง ๆ ซึ่งจำนวนจะมีค่าเริ่มต้นตั้งแต่ 0 จนถึงจำนวนที่ยาร์ดแวร์และซอฟต์แวร์รองรับ ตามหลักการ ดังนั้นหัวข้อนี้จึงมีความสำคัญและจำเป็นต้องทำการทดลองที่ 1 ในภาคผนวก A และการทดลองที่ 5 ภาคผนวก E เพื่อความเข้าใจอย่างลึกซึ้งถึงข้อจำกัดของคอมพิวเตอร์

นอกจากนี้ ยังมีตัวแปรชนิดพอยน์เตอร์ (Pointer) ซึ่งทำหน้าที่เก็บแอดдрес หรือ หมายเลขไปร์ หรือ หมายเลขชี้ของหน่วยความจำที่บรรจุค่าของตัวแปรนั้น ๆ หมายเหตุ แอดдрес คือ ตัวเลขด้านซ้ายของหน่วยความจำในรูปที่ 2.2 ยกตัวอย่าง เช่น ตัวแปร x ตั้งอยู่ที่หมายเลขแอดdress 0x50 บรรจุค่า 0x2A หรือ 42_{10} หมายเลขแอดdress 0x50 เป็นการเขียนแบบย่อ เมื่อเขียนให้ครบ 4 ไปร์ หรือ 32 บิตมีค่าเท่ากับ 0x00000050 สำหรับตัวแปรพอยน์เตอร์ในระบบปฏิบัติการ 32 บิต

ดังนั้น การใช้งานเลขฐานสองชนิดไม่มีเครื่องหมายในภาษาคอมพิวเตอร์ระดับสูง เช่น C/C++ และ java มีได้หลายวิธี ได้แก่ การประกาศตัวแปรชนิดพอยน์เตอร์ซึ่งจะเก็บค่าแอดdressของหน่วยความจำเท่านั้น การประกาศและตั้งค่าเริ่มต้นตัวแปรจำนวนเต็มชนิดไม่มีเครื่องหมายซึ่งตั้งด้วยคำว่า unsigned และตามด้วยชนิดของตัวแปรซึ่งจะเป็นตัวกำหนดความยาวของตัวแปรตามนิยามที่ 2.2.1

นิยามที่ 2.2.1 กำหนดให้ $X_{2,u}$ เป็นเลขจำนวนเต็มฐานสองชนิดไม่มีเครื่องหมาย (*Unsigned Integer*) ความยาว n บิต สามารถเขียนอยู่ในรูป

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

เมื่อ x_i คือ บิต ข้อมูลมีค่า “1” (ON) หรือ “0” (OFF) ในตำแหน่งที่ i และตำแหน่งของมือลูกคือ ตำแหน่งหรือบิตที่ $i=0$

การเข้ามายังระหว่างเลขฐานสองและเลขฐานสิบจะต้องอาศัยการแปลงเลขระหว่างทั้งสองฐานดังต่อไปนี้

การแปลงเลขฐานสองเป็นฐานสิบ

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

$$X_{10,u} = (x_{n-1} \times 2^{n-1}) + (x_{n-2} \times 2^{n-2}) + \dots + (x_1 \times 2^1) + (x_0 \times 2^0) \quad (2.2)$$

ดังนั้น ค่าฐานสิบ $X_{10,u}$ ออยู่ในช่วง 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

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

$$= 2^3 + 0 + 2^1 + 2^0 \quad (2.4)$$

$$= 8 + 0 + 2 + 1 \quad (2.5)$$

$$= 11_{10} \quad (2.6)$$

ดังนั้น เลขจำนวนเต็มฐานสองชนิดไม่มีเครื่องหมายขนาด $n = 4$ บิต จะมีค่าฐานสิบอยู่ในช่วง 0 ถึง $+15$

ตัวอย่างที่ 2.2.2 เลขจำนวนเต็มฐานสองชนิดไม่มีเครื่องหมาย $n = 8$ บิต

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

เทียบเท่ากับการประกาศและตั้งค่าเริ่มต้นตัวแปรชนิด *unsigned char*

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

มีค่าฐานสิบเท่ากับเท่าไหร่

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

$$X_{10,u} = (0 \times 2^7) + \dots + (1 \times 2^3) + (0 \times 2^2) + (1 \times 2^1) + (1 \times 2^0) \quad (2.7)$$

$$= 0 + \dots + 2^3 + 0 + 2^1 + 2^0 \quad (2.8)$$

$$= 0 + \dots + 8 + 0 + 2 + 1 \quad (2.9)$$

$$= 11_{10} \quad (2.10)$$

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

ตัวอย่างที่ 2.2.3 เลขจำนวนเต็มฐานสองชนิดไม่มีเครื่องหมาย $n=16$ บิต

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

เทียบเท่ากับการประกาศและตั้งค่าเริ่มต้นตัวแปรชนิด *unsigned short*

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

มีค่าฐานลิบท่ากับเท่าไหร่

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

$$X_{10,u} = (0 \times 2^{15}) + \dots + (1 \times 2^3) + (0 \times 2^2) + (1 \times 2^1) + (1 \times 2^0) \quad (2.11)$$

$$= 0 + \dots + 2^3 + 0 + 2^1 + 2^0 \quad (2.12)$$

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

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

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

ตัวอย่างที่ 2.2.4 เลขจำนวนเต็มฐานสองชนิดไม่มีเครื่องหมาย $n = 32$ บิต

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

เทียบเท่ากับการประกาศและตั้งค่าเริ่มต้นตัวแปรชนิด *unsigned int*

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

มีค่าฐานลิบท่ากับเท่าไหร่

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

$$X_{10,u} = (0 \times 2^{31}) + \dots + (1 \times 2^3) + (0 \times 2^2) + (1 \times 2^1) + (1 \times 2^0) \quad (2.15)$$

$$= 0 + \dots + 2^3 + 0 + 2^1 + 2^0 \quad (2.16)$$

$$= 0 + \dots + 8 + 0 + 2 + 1 \quad (2.17)$$

$$= 11_{10} \quad (2.18)$$

ดังนั้น ตัวแปรชนิด `unsigned int` หรือเลขจำนวนเต็มฐานสองชนิดไม่มีเครื่องหมายขนาด $n = 32$ บิต จะมีค่าฐานสิบอยู่ในช่วง 0 ถึง $+4,294,967,295$ (บวกสี่พันสองร้อยเก้าลิบสี่ล้านเก้าแสนหกหมื่นเจ็ดพันสองร้อยเก้าลิบห้า)

โดยสรุป ผู้อ่านจะเห็นได้จากตัวอย่างที่ 2.2.1 ถึง 2.2.4 เลขจำนวนเต็มฐานสิบที่มากกว่าเท่ากับศูนย์ค่าเดียวกัน เมื่อนำไปคำนวณด้วยวิจารณิคิทัล หรือที่มีความยาวต่างกัน หรือใช้ตัวแปรที่มีจำนวนบิตยาวกว่าสามารถรองรับค่าสูงสุดได้เพิ่มขึ้นตามตารางที่ 2.1 เช่น ตัวแปรชนิด `unsigned char`, `unsigned short` และ `unsigned int` ซึ่งมีความยาว 8, 16 และ 32 บิตตามลำดับ

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_{10} 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 หมายความว่า ผู้อ่านที่มีประสบการณ์กับการใช้เลขฐานสองบ่อยๆ

ตัวอย่างที่ 2.2.6 จงแปลง 123_{10} เป็นเลขฐานสองด้วยวิธีลบด้วยเลขยกกำลังสองเป็นจำนวน n ครั้ง

ตารางที่ 2.3: การแปลงค่าฐานสิบเป็นเลขฐานสองแบบไม่มีเครื่องหมายความยawa $n=8$ บิต ด้วยวิธีการลบ

บิตที่ i	2^i	ผลลัพธ์- 2^i	ตัวตั้ง $_{10}$	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

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

ตัวอย่างที่ 2.2.5 และ ตัวอย่างที่ 2.2.6 แสดง การ แปลง เลขฐานสิบ เป็น เลขฐานสอง ชนิด ไม่มีเครื่องหมายจำนวน n บิต ทั้งสองวิธี โดยที่ผู้อ่านสามารถทำตามวิธีการหารเอาเศษได้ง่ายกว่า ส่วนวิธีการลบด้วยเลขสองยกกำลังคือการแปลงเลขฐานโดยตรงแต่ต้องอาศัยความจำมากกว่า

2.2.2 ชนิดมีเครื่องหมาย (Signed Integer) แบบ 2's Complement

เลขจำนวนเต็มฐานสองชนิดมีเครื่องหมายในคอมพิวเตอร์ หมายความว่า การใช้แทนข้อมูลที่มีค่าทั้งบวก และลบ บนเส้นจำนวนตามที่ ฮาร์ดแวร์ และซอฟต์แวร์ จะรองรับตามหลักการ 2's Complement กีบองทั้งหมด ดังนั้น หัวข้อนี้จึงมีความสำคัญและจำเป็นต้องทำการทดลองที่ 1 และ 5 ในภาคผนวก A และภาคผนวก E ตามลำดับ เพื่อให้ผู้อ่านเข้าใจถึงข้อจำกัดของคอมพิวเตอร์อย่างลึกซึ้ง

```
char x = -5; /* x = 0b11111011 = 0xFB */
short y = -5; /* y = 0b111111111111011 = 0xFFFFB */
int z = -5; /* z = 0xFFFFFFFFB */
```

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

$$X_{2,s} = x_{n-1}x_{n-2}x_{n-3}\dots x_1x_0 \quad (2.19)$$

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

การแปลงเลขฐานสองเป็นฐานสิบ

การแปลงเลขจำนวนเต็มฐานสองชนิดมีเครื่องหมายแบบ 2's Complement จากนิยามที่ 2.2.2 ให้ เป็นค่าฐานสิบสามารถทำได้โดย

$$X_{10,s} = (-x_{n-1} \times 2^{n-1}) + (x_{n-2} \times 2^{n-2}) + \dots + (x_1 \times 2^1) + (x_0 \times 2^0) \quad (2.20)$$

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

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

ค่าฐานสิบของ $X_{2,s}$ ตามสมการที่ (2.20) คือ

$$X_{10,s} = (-1 \times 2^3) + (0 \times 2^2) + (1 \times 2^1) + (1 \times 2^0) \quad (2.21)$$

$$= -2^3 + 0 + 2^1 + 2^0 \quad (2.22)$$

$$= -8 + 0 + 2 + 1 \quad (2.23)$$

$$= -5_{10} \quad (2.24)$$

เลขจำนวนเต็มฐานสองชนิดมีเครื่องหมาย 2's Complement ขนาด $n = 4$ บิต จะมีค่าฐานสิบอยู่ใน ช่วง -8 ถึง $+7$

จากตัวอย่างที่ 2.2.7 เลขฐานสองความยาว $n=4$ บิตสามารถตีความแบบมีเครื่องหมาย และแบบไม่มีเครื่องหมาย ตามสมการที่ (2.20) และ สมการที่ (2.2) ตามลำดับ ในตารางที่ 2.4

ตารางที่ 2.4: รูปแบบ (Pattern) ของเลขฐานสองขนาด $n=4$ บิต ทั้งหมด $2^4=16$ แบบ และ การตีความหมายให้เป็นค่าฐานสิบแบบมีเครื่องหมาย 2's Complement และแบบไม่มีเครื่องหมาย

เลขฐานสอง $n=4$ บิต	$X_{10,s}$ ค่าฐานสิบ มีเครื่องหมาย สมการ (2.20)	$X_{10,u}$ ค่าฐานสิบ ไม่มีเครื่องหมาย สมการ (2.2)
1000	-8	8
1001	-7	9
1010	-6	10
1011	-5	11
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_2 สามารถตีความเป็นเลขจำนวนเต็มชนิดมีเครื่องหมายแบบ 2's Complement ในคอลัมน์กลางว่าเท่ากับ -5_{10} และเลขจำนวนเต็มชนิดไม่มีเครื่องหมายในคอลัมน์ขวาสุด ว่าเท่ากับ 11_{10}

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

ค่าฐานสิบของ $X_{2,s}$ ตามสมการที่ (2.20) คือ

$$X_{10,s} = (-1 \times 2^4) + (1 \times 2^3) + (0 \times 2^2) + (1 \times 2^1) + (1 \times 2^0) \quad (2.25)$$

$$= -2^4 + 2^3 + 0 + 2^1 + 2^0 \quad (2.26)$$

$$= -16 + 8 + 0 + 2 + 1 \quad (2.27)$$

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

เลขฐานสองแบบ 2 Complement ขนาด $n = 5$ บิต จะมีค่าฐานสิบอยู่ในช่วง $-2^{5-1} = -2^4 = -16$ ถึง $+2^{5-1} = +2^4 = 16$

เลขจำนวนเต็มฐานสองความหมาย $n=5$ บิตสามารถตีความแบบมีเครื่องหมายและแบบไม่มีเครื่องหมาย ตามสมการที่ (2.20) และสมการที่ (2.2) ตามลำดับ ในตารางที่ 2.5

ตารางที่ 2.5: รูปแบบ (Pattern) ของเลขฐานสองขนาด $n=5$ บิต ทั้งหมด $2^5=32$ แบบ และ การตีความหมายให้เป็นค่าฐานสิบแบบมีเครื่องหมาย 2's Complement และแบบไม่มีเครื่องหมาย

เลขฐานสอง $n=5$ บิต	$X_{10,s}$ ค่าฐานสิบ มีเครื่องหมาย สมการ (2.20)	$X_{10,u}$ 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
1 1011	-5	27
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 ผู้เขียนจะใจเว้นช่องว่างทุก ๆ 4 บิต เพื่อให้ผู้อ่านนับเลขฐานสองและแปลงเป็นเลขฐานสิบ หากได้จ่ายชั้น

โปรดสังเกตหมายเหตุ ฐานสอง 1 1011₂ สามารถตีความเป็นเลขจำนวนเต็มชนิดมีเครื่องหมายแบบ 2's Complement ในคอลัมน์กลางว่าเท่ากับ -5₁₀ และเลขจำนวนเต็มชนิดไม่มีเครื่องหมายในคอลัมน์ขวาสุด ว่าเท่ากับ 27₁₀

เลขจำนวนเต็มฐานสองความยาว $n=8$ บิตสามารถตีความแบบมีเครื่องหมาย 2's Complement และแบบไม่มีเครื่องหมาย ตามสมการที่ (2.20) 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 (2.20)	$X_{10,u}$ 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
1111 1011	-5	251
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 ผู้เขียนจะใจเว้นช่องว่างทุก ๆ 4 บิต เพื่อให้ผู้อ่านนับเลขฐานสองและแปลงเป็นเลขฐานสิบหากได้จ่ายขึ้น

โปรดสังเกตหมายเหตุ 1111 1011₂ สามารถตีความเป็นเลขจำนวนเต็มชนิดมีเครื่องหมายแบบ 2's Complement ในคอลัมน์กลางว่าเท่ากับ -5_{10} และเลขจำนวนเต็มชนิดไม่มีเครื่องหมายในคอลัมน์ขวาสุด ว่าเท่ากับ 251_{10} ผู้อ่านจะสังเกตเห็นได้ว่า 1111 ด้านซ้ายของ 1111 1011₂ ในตารางนี้ ทำหน้าที่เหมือน Sign bit ซึ่งสามารถประยุกต์ใช้กับเลขจำนวนเต็มฐานสองแบบ 2's Complement ได้ทุกครั้ง ยกเว้น n

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

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

เทียบเท่ากับการประกากและตึงค่าเริ่มต้นตัวแปรชนิด char

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

มีค่าฐานสิบเท่ากับเท่าไหร่

ค่าฐานสิบของ $X_{2,s}$ ตามสมการที่ (2.20) คือ

$$X_{10,s} = (-1 \times 2^7) + (1 \times 2^6) + (1 \times 2^5) + (1 \times 2^4) + (1 \times 2^3) + (0 \times 2^2) + (1 \times 2^1) + (1 \times 2^0) \quad (2.29)$$

$$= -2^7 + 2^6 + 2^5 + 2^4 + 2^3 + 0 + 2^1 + 2^0 \quad (2.30)$$

$$= -128 + 64 + 32 + 16 + 8 + 0 + 2 + 1 \quad (2.31)$$

$$= -5_{10} \quad (2.32)$$

ตัวแปรชนิด *char* หรือเลขจำนวนเต็มฐาน 8 บิต จะมีค่าฐานสิบอยู่ในช่วง -128 ถึง +127

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

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

เทียบเท่ากับการประมวลผลและตั้งค่าเริ่มต้นตัวแปรชนิด *short*

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

มีค่าฐานสิบเท่ากับเท่าไร

ค่าฐานสิบของ $X_{2,s}$ ตามสมการที่ (2.20) คือ

$$X_{10,s} = (-1 \times 2^{16}) + (1 \times 2^{15}) + \dots + (1 \times 2^3) + (0 \times 2^2) + (1 \times 2^1) + (1 \times 2^0) \quad (2.33)$$

$$= -2^{16} + 2^{15} + \dots + 2^3 + 0 + 2^1 + 2^0 \quad (2.34)$$

$$= -32,768 + 16,384 + \dots + 8 + 0 + 2 + 1 \quad (2.35)$$

$$= -5_{10} \quad (2.36)$$

ตัวแปรชนิด *short* หรือเลขจำนวนเต็มฐานสองชนิดมีเครื่องหมาย 2's Complement ขนาด $n=16$ บิต จะมีค่าฐานสิบอยู่ในช่วง -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 = 0xFFFFFFFF */
```

มีค่าฐานสิบเท่ากับเท่าไร

ค่าฐานสิบของ $X_{2,s}$ ตามสมการที่ (2.20) คือ

$$X_{10,s} = (-1 \times 2^{31}) + (1 \times 2^{30}) + \dots + (1 \times 2^3) + (0 \times 2^2) + (1 \times 2^1) + (1 \times 2^0) \quad (2.37)$$

$$= -2^{31} + 2^{30} + \dots + 2^3 + 0 + 2^1 + 2^0 \quad (2.38)$$

$$= -2,147,483,648 + 1,073,741,824 + \dots + 8 + 0 + 2 + 1 \quad (2.39)$$

$$= -5_{10} \quad (2.40)$$

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

โดยสรุป ผู้อ่านจะเห็นได้จากตัวอย่างที่ 2.2.7 ถึง 2.2.11 เลขฐานสิบค่าเดียวกัน เมื่อนำไปคำนวณด้วย วงจรดิจิทัลหรือที่มีความยาวต่างกัน หรือใช้ตัวแปรที่มีจำนวนบิตยาวกว่าสามารถรองรับค่าสูงสุด/ต่ำสุดได้เพิ่มขึ้นตามตารางที่ 2.1 เช่น ตัวแปรชนิด *char*, *short* และ *int* ซึ่งมีความยาว 8, 16 และ 32 บิตตามลำดับ

```
char X = -5; /* X = 0b11111011 = 0xFB */
short X = -5; /* X = 0b11111111111111011 = 0xFFFFB */
int X = -5; /* X = 0xFFFFFFFFFB */
```

คอมพิวเตอร์เกือบ 100% ในโลกนี้รองรับเลขจำนวนเต็มฐานสองชนิด 2's Complement อาศัยการเติมบิต เครื่องหมาย (Sign bit) ทางด้านซ้ายเพื่อให้เลขยาวขึ้น เรียกว่า การขยายเครื่องหมาย (Sign Extension) ดังนั้น การตั้งค่าเริ่มต้นให้กับตัวแปรทั้งสามชนิดนี้ ด้วยหมายเลข -5_{10} จึงใช้กลไกการขยายเครื่องหมายนี้ทำให้กลายเป็นเลขจำนวนเต็มฐานสองที่มีจำนวนบิตเพิ่มขึ้นเพื่อแทนเลขฐานสิบค่าเดียวกัน ซึ่งจะแสดงรายละเอียดเพิ่มเติมในหัวข้อการแปลงเลขฐานสิบเป็นฐานสอง

การแปลงเลขฐานสิบเป็นฐานสอง

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

$$X_{2,s} = \overline{[|X_{10,s}|]_{2,u}} + 1_2 \quad (2.41)$$

เมื่อ $|X_{10,s}|$ คือ ค่าสัมบูรณ์ของ $X_{10,s}$ และ $\overline{[|X_{10,s}|]_{2,u}}$ คือ การกลับค่าแต่ละบิตของ $X_{10,s}$ ให้เป็นค่าตรงกันข้าม ซึ่งก่อคือการแปลงเป็นค่า 1's Complement นั่นเอง

กรณี $X_{10,s} \geq 0$

$$X_{2,s} = [X_{10,s}]_2 = [X_{10,u}]_{2,u} \quad (2.42)$$

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

ตารางที่ 2.7: การแปลงค่าฐานสิบแบบมีเครื่องหมายให้เป็นเลขฐานสองความยาว $n=4$ บิต โดยแปลงให้เป็นค่าฐานสิบโดยใช้สมการที่ (2.41) 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 (2.41)
-8	$8=1000_2$	$0111_2 + 1_2 = 1000_2$
-7	$7=0111_2$	$1000_2 + 1_2 = 1001_2$
-6	$6=0110_2$	$1001_2 + 1_2 = 1010_2$
-5	$5=0101_2$	$1010_2 + 1_2 = \mathbf{1011}_2$
-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 (2.42)
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 เนื่องจากความต่างที่ความยาวของตัวเลข

ตารางที่ 2.8: การแปลงค่าฐานสิบแบบมีเครื่องหมายให้เป็นเลขฐานสองความยาว $n=5$ บิต โดยแปลงให้เป็นค่าฐานสิบโดยใช้สมการที่ (2.41) 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 (2.41)
-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	$6 = 00110_2$	$11001_2 + 1_2 = 1\ 1010_2$
-5	$5 = 00101_2$	$11010_2 + 1_2 = 1\ 1011_2$
-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 (2.42)
0		$0\ 0000_2$
1		$0\ 0001_2$
2		$0\ 0010_2$
3		$0\ 0011_2$
4		$0\ 0100_2$
5		$0\ 0101_2$
6		$0\ 0110_2$
7		$0\ 0111_2$
9		$0\ 1001_2$
...		...
15		$0\ 1111_2$

หมายเหตุ ผู้เขียนจะเจวันซ่องว่างทุก ๆ 4 บิต เพื่อให้ผู้อ่านนับเลขฐานสองและแปลงเป็นเลขฐานสิบหากได้ง่ายขึ้น โปรดเปรียบเทียบตัวเลขในตารางนี้กับตารางที่ 2.7 ว่าแตกต่างกันตรงที่ -1 ถึง -8 เป็นการขยายบิตเครื่องหมาย $x_4=1$ มาด้านหน้า และ 0 ถึง 7 ว่าเป็นการขยายบิตเครื่องหมาย $x_4=0$ มาด้านหน้าโดยไม่ทำให้ค่าฐานสิบเปลี่ยนแปลง

2.2.3 ชนิดมีเครื่องหมาย (Signed Integer) แบบ Sign-Magnitude

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

นิยามที่ 2.2.3 กำหนดให้ $X_{2,sm}$ เป็นเลขจำนวนเต็มฐานสองชนิดมีเครื่องหมาย (Signed Integer) แบบ Sign-Magnitude ความยาว n บิต สามารถเขียนอยู่ในรูป

$$X_{2,sm} = sx_{n-2}x_{n-3}..x_1x_0 \quad (2.43)$$

เมื่อ s คือบิตเครื่องหมาย (Sign bit) และ บิตข้อมูล x_i คือค่า “1” หรือ “0” ในตำแหน่งที่ i และตำแหน่ง x_0 มีอัตราคือตำแหน่งที่ $i = 0$

การแปลงเลขจำนวนเต็มฐานสองแบบ Sign-Magnitude ให้เป็นค่าฐานสิบสามารถทำได้โดย

$$X_{10,sm} = (-1)^s \times (x_{n-2} \times 2^{n-2} + .. + x_1 \times 2^1 + x_0 \times 2^0) \quad (2.44)$$

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

$$X_{2,sm} = \pm X_{2,u} \quad (2.45)$$

ดังนั้น คำว่า Magnitude แปลว่า ขนาด จึงหมายถึง ค่าฐานสิบของเลขฐานสองชนิดไม่มีเครื่องหมาย ในสมการนี้ 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

$$X_{10,sm} = (-1)^1 \times \{0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0\} \quad (2.46)$$

$$= -(0 + 2 + 1) \quad (2.47)$$

$$= -3_{10} \quad (2.48)$$

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

โดยสรุป จากรูปที่ 2.3 ตอนต้นของบทนี้ เลขฐานสองความยาว $n=4$ บิต จำนวน $2^4=16$ เลขสามารถตีความเป็นเลขจำนวนเต็มได้สามแบบ ดังนี้ แบบมีเครื่องหมายชนิด Sign-Magnitude แบบมี

เครื่องหมายชนิด 2's Complement และแบบไม่มีเครื่องหมาย ตามสมการที่ (2.44), (2.20) และ (2.2) ตามลำดับในตารางที่ 2.9

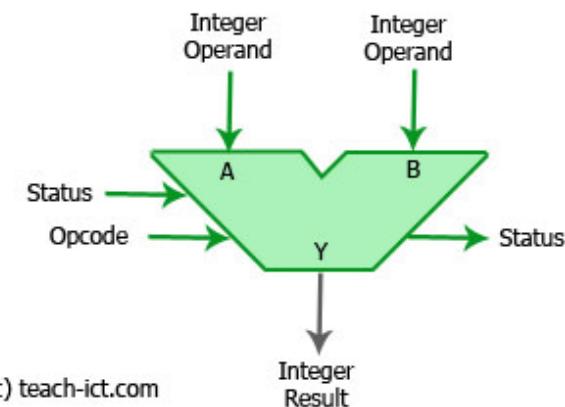
ตารางที่ 2.9: เลขจำนวนเต็มฐานสอง ความยาว $n=4$ บิต ทั้งหมด $2^4=16$ แบบ สามารถตีความเป็น เลขจำนวนเต็มฐานสิบแบบมีเครื่องหมายชนิด Sign-Magnitude, แบบมีเครื่องหมายชนิด 2's Complement, และแบบไม่มีเครื่องหมาย (Unsigned)

เลขฐานสอง $n=4$ บิต	$X_{10,sm}$ ค่าฐานสิบ Sign-Mag. สมการ (2.44)	$X_{10,s}$ ค่าฐานสิบ 2-Comp. สมการ (2.20)	$X_{10,u}$ ค่าฐานสิบ Unsigned สมการ (2.2)
1111	-7	-1	15
1110	-6	-2	14
1101	-5	-3	13
1100	-4	-4	12
1011	-3	-5	11
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

การคำนวณทางคณิตศาสตร์และตรรกศาสตร์ของเลขจำนวนเต็มฐานสองขนาด $n=8, 16, 32, 64$ และ 128 บิต จำเป็นต้องอาศัย วงจร ดิจิทัล เรียกว่า วงจร ALU (Arithmetic Logic Unit) เพื่อการทำงานที่รวดเร็ว รูปที่ 2.4 แสดง สัญลักษณ์ของ วงจร ALU สำหรับ เลขจำนวนเต็มฐานสองขนาด n บิต ชนิดไม่มีเครื่องหมาย ประกอบด้วย



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

- 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 เพื่อบอกว่าผลลัพธ์ Y มีค่าเท่ากับศูนย์ทุกบิต
 - ขาสัญญาณ ชื่อ Carry (c_n หรือ C) สำหรับบิตตัวท้ายที่ n
 - ขาสัญญาณ ชื่อ Overflow (V) เพื่อบอกความผิดพลาด โดย
 - * $V=0$ หมายถึง ไม่เกิดโอเวอร์โฟล์ว ซอฟต์แวร์สามารถนำค่านี้ไปใช้งานได้
 - * $V=1$ หมายถึง เกิดโอเวอร์โฟล์วทำให้ผลลัพธ์มีความผิดพลาด (Invalid) ซอฟต์แวร์ไม่สามารถนำค่านี้ไปใช้งานได้
 - * สำหรับเลขจำนวนเต็มชนิดไม่มีเครื่องหมายสามารถตรวจสอบโดยใช้สมการที่ (2.52)

- * สำหรับเลขจำนวนเต็มฐานสองมีเครื่องหมายแบบ 2's Complement สามารถตรวจสอบโดยใช้สมการที่ (2.56)

ขาสัญญาณ เอต์พุต เหล่านี้ จะบันทึกลงในรีจิส เตอร์สถานะ (Status Register) สำหรับให้ wang และโปรแกรมเมอร์ตรวจสอบด้วยwang จรดิจิทัลและคำสั่งภาษาแอสเซมบลี 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 และ เกิดตัวทด c_1 ใน wang จรดิจิทัล การบวกเลขจำนวนเต็มฐานสองจำนวน 3 บิตเข้าด้วยกัน จะทำให้เกิดผลลัพธ์จำนวน 2 บิต เรียงติดกัน คือ

$$c_{i+1}z_i = x_i + y_i + c_i \quad (2.49)$$

เมื่อ $i=0, 1, 2, \dots, n-1$ โดย $c_0 = 0$ และสัญลักษณ์ + คือการบวกเลข ไม่ใช่การ OR กันเชิงตรรกศาสตร์ ในวิชาออกแบบวงจรดิจิทัล เราเรียกว่า Full Adder โดยวงจรจะนำบิตข้อมูลจำนวน 3 บิตมากระทำการทางตรรกศาสตร์ด้วยพีซคณิตบูลลีน ได้ผลลัพธ์ z_i โดย

$$z_i = x_i \oplus y_i \oplus c_i \quad (2.50)$$

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

$$c_{i+1} = (x_i \& y_i) | (x_i \& c_i) | (y_i \& c_i) \quad (2.51)$$

เมื่อ & คือ กระบวนการ AND และ | คือ กระบวนการ OR วงจรบวกเลขชนิดไม่มีเครื่องหมายขนาด n บิตนี้สามารถตรวจจับการเกิดโอเวอร์โฟลว์ได้โดย

$$V = c_n \quad (2.52)$$

การบวกเลขจำนวนเต็มชนิดไม่มีเครื่องหมาย

การบวกเลขจำนวนเต็มชนิดไม่มีเครื่องหมาย 2 จำนวน ผลลัพธ์ที่ได้จะไม่มีเครื่องหมายด้วย เช่น กัน แต่ การบวกเลขขนาดใหญ่ที่เข้าใกล้ค่าสูงสุด สามารถเกิดความผิดพลาดได้ เรียกว่า การเกิดโอเวอร์โฟล์ว (Overflow) ในสมการที่ (2.52) ซึ่งเป็นผลสืบเนื่องจากการจารดิจิทัลที่สามารถประมวลผลได้จำกัดตามจำนวนบิตข้อมูลสูงสุดที่ทำได้ ในตัวอย่าง การแปลงเลขฐานสองเป็นฐานสิบที่ได้แสดงไปแล้ว ยกตัวอย่างเช่น การวนรอบหรือวนลูป (Loop) เพิ่มค่าอย่างต่อเนื่องโดยไม่ระวัง ตามประโยชน์ในภาษา C/C++ ประโยชน์ $i++$ หรือ $i=i+1$ นี้ หาก i มีค่าเพิ่มขึ้นเรื่อยๆ จนถึงค่าสูงสุด การบวกเพิ่มอีก 1 ไปเรื่อยๆ โดยไม่มีการตรวจสอบการเกิดโอเวอร์โฟล์วล่วงหน้า จะทำให้ค่าของ i กลายเป็นศูนย์ในที่สุด ซึ่งอาจทำให้เกิดผลร้ายตามมาอย่างรุนแรง ผู้อ่านสามารถทดสอบได้ตามกิจกรรมท้ายการทดลองในภาคผนวก E

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

การบวกเลขขนาด 4 บิตแบบไม่มีเครื่องหมาย: $5 + 9 = 14$ พร้อมตัวทด 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 $7 + 9$ ด้วยเลขจำนวนเต็มชนิดมีเครื่องหมายแบบ Unsigned ขนาด 4 บิต $7 + 9 = 0$ 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		

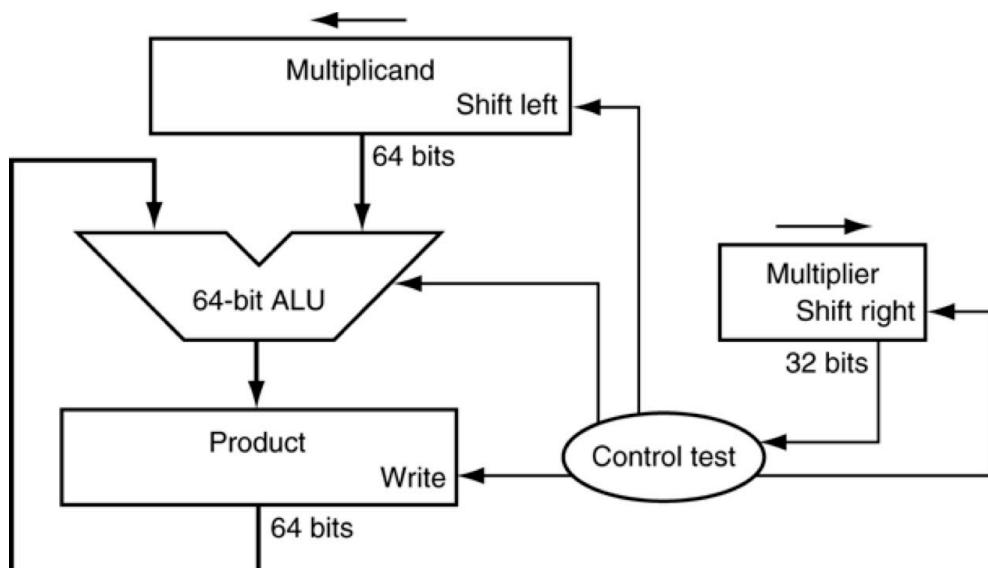
สาเหตุของการเกิด Overflow เนื่องจากผลลัพธ์มีค่าอยู่นอกย่านที่เป็นไปได้ โดยสามารถตรวจสอบอย่างง่ายโดย $V=c_4 = 1$ (V : Overflow) ตามสมการที่ (2.52) เมื่อเกิดโอเวอร์โฟล์ว ผลลัพธ์ที่ได้จึงมีค่าไม่ถูกต้อง (Invalid) ทำให้ซอฟต์แวร์ไม่สามารถนำค่าผลลัพธ์นี้ไปใช้

การคูณเลขจำนวนเต็มชนิดไม่มีเครื่องหมาย

การคูณเลขจำนวนเต็มชนิดไม่มีเครื่องหมายมีความสำคัญต่อการคำนวณทางคณิตศาสตร์ต่าง ๆ และเป็นพื้นฐานของการคำนวณเลขชนิดอื่น ๆ ซึ่งการคูณเลขมีความซับซ้อน (Complexity)มากกว่าการบวกเลขหลายเท่า เพื่อให้ผู้อ่านตระหนักรถึงว่าจุดเด่นที่จำเป็น และเวลาที่ซอฟต์แวร์ใช้คำนวณ ตำราเล่มนี้จึงขอวางแผนว่างรูปแบบการคูณเลขจำนวนเต็มชนิดไม่มีเครื่องหมายให้ผู้อ่านแต่พอสังเขป ดังนี้

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 การคูณเลขมีความซับซ้อนสูง จำเป็นต้องใช้วงจรดิจิทัลและเวลาในคำนวณ 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

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

ตารางที่ 2.10: ผลคูณเลขขนาด 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 ₂ (Product ₂)	ผลคูณ ₁₀ (Product ₁₀)	การทำ (Action)
-	$0000\ 1010_2$	1101_2	$0000\ 0000_2$	0_{10}	Initialized
0	$0000\ 1010_2$	1101_2	$0000\ 1010_2$	10_{10}	Add
	$0001\ 0100_2$	0110_2	$0000\ 1010_2$	10_{10}	Shift
1	$0001\ 0100_2$	0110_2	$0000\ 1010_2$	10_{10}	None
	$0010\ 1000_2$	0011_2	$0000\ 1010_2$	10_{10}	Shift
2	$0010\ 1000_2$	0011_2	$0011\ 0010_2$	50_{10}	Add
	$0101\ 0000_2$	0001_2	$0011\ 0010_2$	50_{10}	Shift
3	$0101\ 0000_2$	0001_2	$1000\ 0010_2$	130_{10}	Add
	$1010\ 0000_2$	0000_2	$1000\ 0010_2$	130_{10}	Shift

หมายเหตุ ผู้เขียนจะใจเว้นช่องว่างทุก ๆ 4 บิต เพื่อให้ผู้อ่านนับเลขฐานสองและแปลงเป็นเลขฐานสิบหากได้ง่ายขึ้น

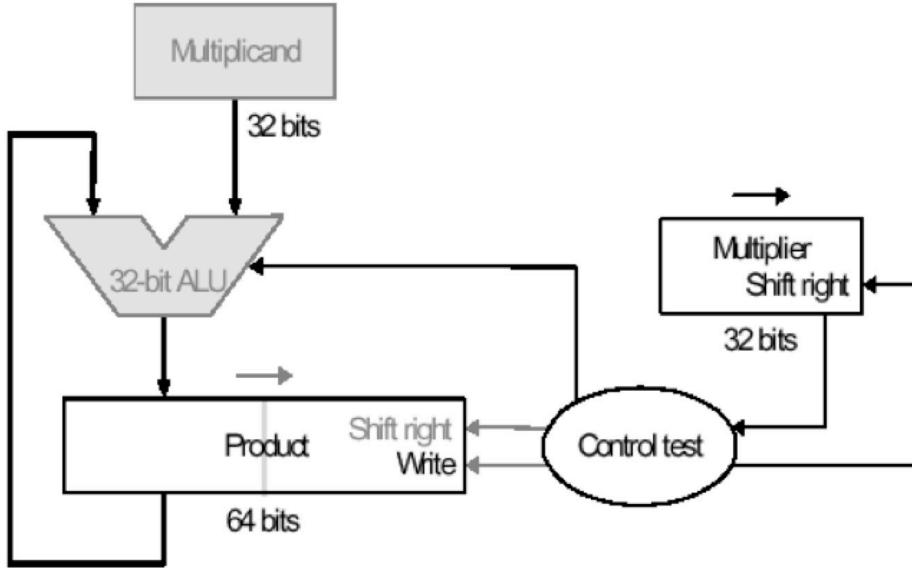
เราจะเห็นว่าการคูณคือ การบวกตัวตั้ง (Multiplicand) ที่ถูกเลื่อนบิตไปทางซ้ายรอบละ 1 บิต กับผลคูณในรอบก่อนหน้า โดยจำนวนรอบขึ้นอยู่กับจำนวนบิตของตัวคูณ (Multiplier) ในวิธีนิดที่ 2 จะใช้วิธีการที่ใช้ ALU ขนาดเล็กลง รีจิสเตอร์บางตัวลับลง และใช้จำนวนรอบเท่าเดิม

วงจรคูณเลขจำนวนเต็มฐานสองไม่มีเครื่องหมายนิดที่ 2

วงจรในรูปที่ 2.6 มีการปรับเปลี่ยนให้รีจิสเตอร์ตัวตั้ง และรีจิสเตอร์ตัวคูณเป็นเลขจำนวนเต็มไม่มีเครื่องหมายขนาด n บิตทั้งคู่ และรีจิสเตอร์ผลคูณจะมีขนาด $2n$ บิต มีการทำงานเป็นขั้นตอนดังนี้

1. ตั้งค่าเริ่มต้นของรีจิสเตอร์ทุกตัว

- ตั้งค่ารีจิสเตอร์ตัวตั้งขนาด n บิต ตามค่าตัวตั้งโดยให้บิตขวาสุดของตัวตั้งอยู่ที่บิต 0 ของรีจิส



រូបទី 2.6: វងរគុណលេខចាំនាន់ $n=32$ បិត ចានិតទី 2 ដើម្បី 32 បិត និង 32 បិត និង 64 បិត ទៅ 32 បិត ទៀតែ: [Patterson and Hennessy \(2016\)](#)

ពេលវេលា

- ត៉ែងការីសតោរ់ត៊ែវគុណចាំនាន់ n បិត តាមការត៊ែវគុណ
- ត៉ែងការីសតោរ់ផលគុណចាំនាន់ $2n$ បិត ដើម្បីបើក 0 ទុកបិត

2. តរាងសរបគាត់បិតទី 0 (ខ្សាសុទ) ឱងរីជិសតោរ់ត៊ែវគុណ

- ហាកមីគាត់បិតទី 0 បានគាត់ត៊ែវតែងក្នុង n បិត ទាន់មួយទីនៃគុណដែលមិនមែនត៊ែវតែងក្នុង n បិត (Action = Add)
- ហាកមីគាត់បិតទី 0 មិនមែនត៊ែវតែងក្នុង n បិត (Action = None)

3. តើនឹងឈើនូវឯកសារនៃរីជិសតោរ់ត៊ែវគុណ

- តើនឹងឈើនូវឯកសារនៃរីជិសតោរ់ត៊ែវគុណ និង រីជិសតោរ់ផលគុណ ដើម្បីបង្ហាញសង (Shift R)
- ចូល 0 ទៅការត៊ែវគុណនៃរីជិសតោរ់ត៊ែវគុណ ដើម្បីបង្ហាញសង (Shift R)
- ចូលគាត់ c_n ទៅការត៊ែវគុណនៃរីជិសតោរ់ផលគុណ ដើម្បីបង្ហាញសង (Shift R)

4. កត់បញ្ជីត៊ែវតែងក្នុង n បិត

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

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

ตารางที่ 2.11: ผลคูณเลขขนาด 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 ₂ (Product ₂)	ผลคูณ ₁₀ (Product ₁₀)	กระทำ (Action)
-	1010_2	1101_2	$0000\ 0000_2$	0_{10}	Initialized
0	1010_2	1101_2	$1010\ 0000_2$	160_{10}	Add
	1010_2	0110_2	$0101\ 0000_2$	80_{10}	Shift R
1	1010_2	0110_2	$0101\ 0000_2$	80_{10}	None
	1010_2	0011_2	$0010\ 1000_2$	40_{10}	Shift R
2	1010_2	0011_2	$1100\ 1000_2$	200_{10}	Add
	1010_2	0001_2	$0110\ 0100_2$	100_{10}	Shift R
3	1010_2	0001_2	$1\ 0000\ 0100_2$	260_{10}	Add
	1010_2	0000_2	$1000\ 0010_2$	130_{10}	Shift R

หมายเหตุ ผู้เขียนจะใจเว้นช่องว่างทุก ๆ 4 บิต เพื่อให้ผู้อ่านนับเลขฐานสองและแปลงเป็นเลขฐานสิบหากได้รับชิ้น

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

วงจรคูณเลขจำนวนเต็มทั้งสองชนิดนี้ เป็นแค่ตัวอย่างเพื่อให้ผู้อ่านเข้าใจการแลกเปลี่ยน (Trade off) ระหว่าง จำนวนรอบ (ระยะเวลา) กับ ความซับซ้อนของวงจร ซึ่งในทางปฏิบัติวงจรคูณจะมีความซับซ้อนมากกว่าแต่ใช้ระยะเวลาสั้นกว่า และสามารถออกแบบให้ทำงานแบบไปป์ไลน์ด้วย

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 และ ทำให้ $x_0+y_0+c_0$ ได้บิต z_0 และเกิดตัวทด c_1 ในวงจรดิจิทัล การบวกเลขฐานสองจำนวน 3 บิตเข้าด้วยกัน จะทำให้เกิดผลลัพธ์จำนวน 2 บิต เรียกว่าติดกัน คือ

$$c_{i+1}z_i = x_i + y_i + c_i \quad (2.53)$$

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

ผลลัพธ์ของการบวกเลขฐานสองจำนวน 3 บิต สามารถคำนวณได้จากการ Full Adder ดังนี้

$$z_i = x_i \oplus y_i \oplus c_i \quad (2.54)$$

เมื่อ \oplus คือ กระบวนการ Exclusive-OR

$$c_{i+1} = (x_i \& y_i) | (x_i \& c_i) | (y_i \& c_i) \quad (2.55)$$

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

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

$$V = c_n \oplus c_{n-1} \quad (2.56)$$

ข้อสังเกตุ เหล่านี้ จะบันทึกลงในรีจิสเตอร์สถานะ สำหรับให้โปรแกรมเมอร์ตรวจสอบ ด้วยภาษา แอสเซมบลี ในบทที่ 4 หัวข้อที่ 4.6 โปรดสังเกตความแตกต่างระหว่างการตรวจสอบโอเวอร์โฟล์วของ เลขจำนวนเต็มฐานสองมีเครื่องหมายในสมการที่ (2.56) และ เลขจำนวนเต็มฐานสองไม่มีเครื่องหมายใน สมการที่ (2.52)

การบวก/ลบเลขจำนวนเต็มชนิดมีเครื่องหมาย

การบวก/ลบเลขจำนวนเต็มชนิดมีเครื่องหมาย 2 จำนวน ผลลัพธ์ที่ได้จะมีเครื่องหมายด้วยเช่นกัน แต่ การบวก/ลบเลขขนาดใหญ่ที่เข้าใกล้ค่าสูงสุดหรือค่าต่ำสุด สามารถเกิดความผิดพลาดได้ เรียกว่า การเกิด โอเวอร์โฟล์ว (Overflow) ในสมการที่ (2.56) ซึ่งเป็นผลสืบเนื่องจากวงจรดิจิทัลที่สามารถประมวลผลได้ จำกัดตามจำนวนบิตข้อมูลสูงสุดที่ทำได้ ยกตัวอย่าง เช่น การวนรอบหรือวนลูป (Loop) เพิ่มค่าอย่างต่อเนื่องโดยไม่ระวัง ตามประโยคในภาษา C/C++ ประโยค $i++$ หรือ $i = i + 1$ นี้ หาก i มีค่าเพิ่มขึ้นเรื่อยๆ จนถึงค่าสูงสุด การบวกเพิ่มอีก 1 ไปเรื่อยๆ โดยไม่มีการตรวจสอบการเกิดโอเวอร์โฟล์วล่วงหน้า จะทำให้ค่าของ i กลายเป็นค่าลบในที่สุด ซึ่งอาจทำให้เกิดผลร้ายตามมาอย่างรุนแรง 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
	0	1	1	1	0	$V=0 \oplus 1=1$
$\times = 7$		0	1	1	1	+
$+Y = +3$		0	0	1	1	
$Z = -6$		1	0	1	0	

ผลการคำนวณผลลัพธ์ที่ไม่ถูกต้อง (Invalid) เนื่องจากเกิด โอเวอร์โฟล์ว (Overflow) เพราะฮาร์ดแวร์คำนวณ $V=c_4 \oplus c_3 = 0 \oplus 1 = 1$ ตามสมการที่ (2.56) ทำให้ซอฟต์แวร์นำค่าตอบนี้ไปใช้ไม่ได้

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
	1	1	1	0	0	$V=1 \oplus 1=0$
$\times = 7 +$		0	1	1	1	+
$-Y = -6$		1	0	1	0	
$Z = 1$		0	0	0	1	

ผลการคำนวณผลลัพธ์ที่ได้ถูกต้อง (Valid) เนื่องจากไม่เกิด โอเวอร์โฟล์ว (Overflow) เพราะฮาร์ดแวร์คำนวณ $V=c_4 \oplus c_3 = 1 \oplus 1 = 0$ ตามสมการที่ (2.56) ทำให้ซอฟต์แวร์นำค่าตอบนี้ไปใช้ได้

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
	1	0	0	0	0	$V=1 \oplus 0=1$
$\times = -3 +$		1	1	0	1	+
$-Y = -6$		1	0	1	0	
$Z = +7$		0	1	1	1	

ผลการคำนวณผลลัพธ์ที่ไม่ถูกต้อง (Invalid) เนื่องจากเกิดโอเวอร์โฟล์ว (Overflow) เพราะชาร์ดแวร์คำนวณ $V = c_4 \oplus c_3 = 1 \oplus 0 = 1$ ตามสมการที่ (2.56) ทำให้ซอฟต์แวร์คำนวณนี้ไปใช้ไม่ได้

2.4 เลขคณิตฐานสองชนิดจุดตรึง (Fixed Point)

เลขจำนวนจริงแบ่งเป็นเลขคณิตชนิดจุดตรึง (Fixed-Point Real Number) และ ชนิดจุดเลขคณิตลอยตัว (Floating-Point Real Number) ในหัวข้อนี้ ผู้อ่านจะได้ทำความเข้าใจเรื่องรูปแบบ (Format) ของเลขคณิตฐานสองชนิดจุดตรึง ซึ่งนิยมใช้ในตัวประมวลผลสัญญาณดิจิทัล (Digital Signal Processor) ส่วนการบวกและลบเลข ทศนิยมฐานสองชนิดนี้ จะใช้หลักการเดียวกับเลขจำนวนเต็มชนิด Sign-Magnitude โดยการตรวจจับการเกิดโอเวอร์โฟล์วสำหรับเลขที่มีค่าสัมบูรณ์มาก ๆ จนไม่สามารถคำนวณได้ และการเกิดอันเดอร์โฟล์วสำหรับเลขที่มีค่าสัมบูรณ์น้อย ๆ จนเกือบเป็นศูนย์

นิยามที่ 2.4.1 กำหนดให้ F_2 เป็นเลขคณิตฐานสองชนิด Signed Magnitude เขียนอยู่ในรูป

$$F_2 = [s][x_{n-1}x_{n-2}x_{n-3}\dots x_1x_0].[y_{-1}y_{-2}y_{-3}\dots y_{-m}] \quad (2.57)$$

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

โดย F_{10} คือ ค่าฐานสิบของเลขคณิตฐานสองชนิดจุดตรึง F_2 สามารถคำนวณได้จาก

$$F_{10} = (-1)^s \times [x_{n-1}2^{n-1} + \dots + x_12^1 + x_02^0 + y_{-1}2^{-1} + y_{-2}2^{-2} + y_{-3}2^{-3} + \dots + y_{-m}2^{-m}] \quad (2.58)$$

หรือ

$$F_{10} = (-1)^s \times [x_{n-1}2^{n-1} + \dots + x_12^1 + x_02^0 + \frac{y_{-1}}{2} + \frac{y_{-2}}{4} + \frac{y_{-3}}{8} + \dots + \frac{y_{-m}}{2^m}] \quad (2.59)$$

ตัวอย่างที่ 2.4.1 จงแปลงเลขฐานสิบต่อไปนี้เป็นเลขคณิตฐานสองชนิดจุดตรึง $m=n=2$ บิต

$$+0.75_{10} = 000.11_2 \quad (2.60)$$

$$+3.00_{10} = 011.00_2 \quad (2.61)$$

$$-3.75_{10} = 111.11_2 \quad (2.62)$$

ผู้อ่านสามารถสามารถเขียนเลขทศนิยมฐานสองชนิด จุดตรึงได้ตามรูปแบบนี้

$$F_2 = [s][X_{2,u}].[Y_2] \quad (2.63)$$

ค่าทศนิยม (Y_2) มีความยาว m บิต เขียนเป็นสัญลักษณ์ได้ดังนี้

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

เมื่อจำนวนบิตหลังจุดทศนิยมเพิ่มขึ้น ความละเอียดจะเพิ่มขึ้นตามตัวอย่างต่อไปนี้

ตัวอย่างที่ 2.4.2 จงแปลงเลขทศนิยมฐานสองชนิด จุดตรึงต่อไปนี้ให้เป็นเลขฐานสิบ

$$0.1111_2 = 0.9375_{10} = 0.5 + 0.25 + 0.125 + 0.0625 \quad (2.65)$$

$$0.11111_2 = 0.96875_{10} = 0.5 + 0.25 + 0.125 + 0.0625 + 0.03125 \quad (2.66)$$

$$0.111111_2 = 0.984375_{10} = 0.5 + 0.25 + 0.125 + 0.0625 + 0.03125 + 0.015625 \quad (2.67)$$

เมื่อจำนวนบิตที่เป็น 1 เพิ่ม ค่าฐานสิบของเลขทศนิยมฐานสองนี้จะถูกเพิ่มขึ้น 1.0 ยิ่งขึ้น ในขณะที่ การบวก/ลบ และ การคูณเลขชนิด จุดตรึง มีความคล้ายกับเลขจำนวนเต็มชนิด Sign-Magnitude โดยรายละเอียดจะกล่าวในหัวข้อถัดไป

2.5 เลขทศนิยมฐานสองชนิดจุดลอยตัว (Floating Point)

เลขทศนิยมฐานสองชนิดจุดลอยตัวหมายความว่าสำหรับข้อมูลที่มีพิสัย (Range) กว้างและเลขทศนิยมที่ต้องการความละเอียดสูง สำหรับการคำนวณทางวิทยาศาสตร์ (Scientific) ดังนี้

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

นิยามที่ 2.5.1 เลขทศนิยมชนิดจุดลอยตัวฐานสองที่อยู่ในรูปนอร์มัลไลร์ ประกอบด้วย 3 ส่วน คือ บิตเครื่องหมาย (Sign bit: s) ค่านัยสำคัญ (Significant: S_2) ในสมการที่ (2.70) และค่ายกกำลัง (Exponent: E_2) มีลักษณะดังนี้

$$(-1)^s \times [1.y_{-1}y_{-2}y_{-3}\dots y_{-m}]_2 \times 2^{E_2} \quad (2.68)$$

เลขยกกำลังเป็นเลขจำนวนเต็มฐานสองชนิด sign-magnitude ความยาว n บิต ดังนี้

$$E_2 = \pm[e_{n-1}e_{n-2}\dots e_0]_2 \quad (2.69)$$

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

จากนิยามที่ 2.68 ค่านัยสำคัญ (Significant) S_2 ความยาว $m+1$ บิต สามารถเขียนใหม่ได้ ดังนี้

$$S_2 = [1.y_{-1}y_{-2}y_{-3}\dots y_{-m}]_2 \quad (2.70)$$

ซึ่งมีความสำคัญต่อรูปแบบการเขียน เนื่องจากว่าจะต้องทำการนอร์มัลไลร์ผลลัพธ์ที่ได้จากการคำนวณเสมอ ค่านัยสำคัญที่นอร์มัลไลร์ข้างต้นสามารถคำนวณหาค่าฐานสิบ ได้ดังนี้

$$S_{10} = (-1)^s \times \left[1 + \frac{y_{-1}}{2} + \frac{y_{-2}}{4} + \frac{y_{-3}}{8} + \dots + \frac{y_{-m}}{2^m}\right] \times (2^{\pm E_2}) \quad (2.71)$$

ตัวอย่างที่ 2.5.1 จงแปลงเลขทศนิยมฐานสองชนิดจุดลอยตัวที่นอร์มัลไลร์แล้ว $(-1)^1 \times 1.0101_2 \times 2^3$ ให้เป็นเลขทศนิยมฐานลิบตามสมการที่ (2.71)

วิธีทำ

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

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

ตัวอย่างที่ 2.5.2 จงแปลงเลขทศนิยมฐานสองชนิดจุดลอยตัวที่นอร์มัลไลซ์แล้ว $(-1)^0 \times 1.1010_2 \times 2^{-3}$ ให้เป็นเลขทศนิยมฐานสิบตามสมการที่ (2.71)

วิธีทำ

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

- 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

เพื่อให้ผู้อ่านเข้าใจ กลไก การบวกเลขทศนิยมฐานสองชนิดจุดลอยตัว โดยใช้ตัวอย่าง จาก การบวกเลขทศนิยมฐานสิบ ก่อน แล้วจึงนำขั้นตอนที่ เมื่ອนกันไปประยุกต์ใช้กับ การบวกเลขทศนิยมฐานสอง ในตัวอย่างต่อไป

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

วิธีทำ

- เลื่อนตำแหน่งจุดทศนิยมของเลขนัยสำคัญ และปรับเลขยกกำลังที่มีค่าสัมบูรณ์น้อยกว่า ให้ตรงกับเลขยกกำลังของเลขที่มีค่าสัมบูรณ์มากกว่า
 $9.999 \times 10^1 + 0.016 \times 10^1$

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

- นอร์มัลไลซ์ค่าผลลัพธ์ และตรวจเช็คการเกิดโอเวอร์ไฟล์ และอันเดอร์ไฟล์
 1.0015×10^2

- ปัดค่านัยสำคัญให้เหลือ 4 หลัก และอาจต้องนอร์มัลไลซ์อีกรอบหากจำเป็น
 1.002×10^2

จะเห็นได้ว่า ผลลัพธ์ที่ได้จะเกิดความคลาดเคลื่อนจากค่าที่ควรจะเป็น เนื่องจากมีการจำกัดจำนวนตัวจิทของผลลัพธ์ให้เหลือ 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. เลื่อนตำแหน่งจุดทศนิยมของเลขนัยสำคัญและปรับเลขยกกำลังที่มีค่าสัมบูรณ์น้อยกว่าให้ตรงกับเลขยกกำลังของเลขที่มีค่าสัมบูรณ์มากกว่า

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

2. บวกค่านัยสำคัญที่เลื่อนตำแหน่งแล้วเข้าด้วยกัน จะได้ผลลัพธ์ดังนี้

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

3. นอร์มัลไลซ์ค่านัยสำคัญของผลลัพธ์และตรวจเช็คการเกิดโอเวอร์โฟล์วและอันเดอร์โฟล์ว

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

4. ปัดค่านัยสำคัญให้เหลือ 4 บิตและอาจต้องนอร์มัลไลซ์อีกรอบหากจำเป็น

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

2.5.2 การคูณเลขทศนิยมฐานสองชนิดจุดลอยตัว

เพื่อให้ผู้อ่านเข้าใจการคูณเลขทศนิยมฐานสองชนิด 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 และตรวจเช็คการเกิดโอเวอร์โฟล์วและอันเดอร์โฟล์ว

$$1.0212 \times 10^6$$

4. ปัดค่าให้เหลือ 4 หลักและอาจต้องนอร์มัลไลซ์เมื่อจำเป็น

$$1.021 \times 10^6$$

5. ปรับเครื่องหมายของผลลัพธ์ให้ถูกต้องจากตัวตั้งและตัวคูณ

$$+1.021 \times 10^6$$

จะเห็นได้ว่า ผลลัพธ์ที่ได้จะเกิดความคลาดเคลื่อนจากค่าที่ควรจะเป็น เนื่องจากมีการจำกัดจำนวนตัวเลขของผลลัพธ์ให้เหลือ 4 หลักตามโจทย์

ตัวอย่างที่ 2.5.6 จงคูณเลขทศนิยมฐานสองชนิดจุดลอยตัวขนาด 4 บิต (4-bit) ที่นอร์มัลไลซ์ 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. บวกค่ายกกำลังทั้งสองค่าเข้าด้วยกัน ทำให้ค่ายกกำลังใหม่: $-1 + -2 = -3$

2. คูณค่านัยสำคัญเข้าด้วยกัน

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

3. นอร์มัลไลซ์ค่านัยสำคัญของผลลัพธ์ และตรวจเช็คการเกิดโอเวอร์โฟล์วและอันเดอร์โฟล์ว

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

4. ปัดค่านัยสำคัญให้เหลือ 4 หลักและอาจต้องนอร์มัลไลซ์เมื่อจำเป็น

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

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 เพื่อให้โปรแกรมคอมพิวเตอร์สามารถคำนวณค่าเลขทศนิยมฐานสองบนเครื่องที่ใช้ชีพิญได้ๆ ก็ได้แล้วให้ผลลัพธ์ตรงกัน ปัจจุบันนี้มาตรฐานได้รับการยอมรับอย่างแพร่หลาย และปรับปรุงอย่างต่อเนื่อง สามารถรองรับเลขทศนิยมจุดลอยตัว 2 รูปแบบ หลัก คือ

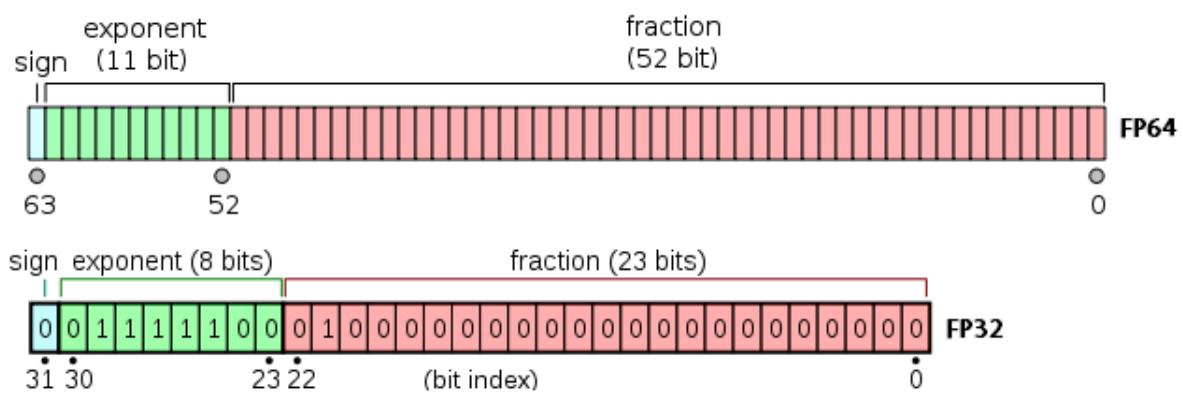
- ชนิด Single precision (32-bit) ตรงกับตัวแปรชนิด float ในภาษา C/C++ และ Java
- ชนิด Double precision (64-bit) ตรงกับตัวแปรชนิด double ในภาษา C/C++ และ Java เวอร์ชันล่าสุดของ IEEE754 คือ ปี ค.ศ. 2019 [รายละเอียดเพิ่มเติม](#)

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

นิยามที่ 2.6.1 เลขทศนิยมฐานสองชนิด จุดลอยตัว ที่นอร์มัลไลซ์แล้ว สามารถเขียนอยู่ในรูปของเลขฐานสองต่อไปนี้

$$F_{2,IEEE} = [s][E_{2,IEEE}][Y_2] \quad (2.72)$$

โดยค่าทศนิยม (Fraction): Y_2 ตามสมการที่ (2.73) มีความยาว $m=23$ และ 52 บิต ตามชนิด Single Precision และ Double Precision ตามลำดับ สามารถเขียนเป็นลัญลักษณ์คล้ายกับเลขทศนิยมฐานสองชนิดจุดตรึง ดังนี้

$$Y_2 = y_{-1}y_{-2}y_{-3}\dots y_{-m} \quad (2.73)$$

ดังนั้น ค่านัยสำคัญ (Significand: S_2) มีความยาว $m+1=24$ หรือ 53 บิต โดยมีรูปแบบตามสมการที่ (2.70) ในหัวข้อที่ 2.5

ค่ายกกำลัง เป็นเลขจำนวนเต็มชนิดไม่มีเครื่องหมาย ความยาว $n=8$ และ 11 บิต ขึ้นกับชนิด Single Precision และ Double Precision ตามลำดับ โดยมีลักษณะคล้ายกับเลขทศนิยมฐานสองชนิดจุดลอยตัว ในสมการที่ (2.69) แต่ต่างกันที่เลขยกกำลังของ IEEE754 มีค่าเป็นเลขจำนวนเต็มไม่มีเครื่องหมาย

$$E_{2,IEEE} = [e_{n-1}e_{n-2}\dots e_0] \quad (2.74)$$

โดยรายละเอียดเป็นดังนี้

- s : บิตเครื่องหมาย (Sign bit)
 - 0 หมายถึง เลขบวกหรือเท่ากับศูนย์ (non-negative)
 - 1 หมายถึง เลขลบ (negative)
- เลขนัยสำคัญ (Significand = $1+Fraction$): หมายความว่า ค่าสัมบูรณ์ (Absolute Value) ของเลขนัยสำคัญจะอยู่ในช่วง $1.0 \leq |Significand| < 2.0$
- จากสมการที่ (2.74) 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

$$E_2 = E_{2,IEEE} - E_{bias} \quad (2.75)$$

เลขทศนิยมฐานสองชนิดจุดลอยตัวตามมาตรฐาน IEEE754 ในสมการที่ (2.72) สามารถแปลงเป็นค่าเลขทศนิยมฐานสิบ ได้ดังนี้

$$F_{10,IEEE} = (-1)^s \times [1 + \frac{y_{-1}}{2} + \frac{y_{-2}}{4} + \frac{y_{-3}}{8} + \dots + \frac{y_{-m}}{2^m}] \times 2^{(E_{2,IEEE} - E_{bias})} \quad (2.76)$$

ตัวอย่างที่ 2.6.1 เลขทศนิยมฐานสองชนิดจุดลอยตัวมาตรฐาน IEEE754 ชนิด Single Precision ต่อไปนี้ มีค่าเท่าไรในฐานสิบตามสมการที่ (2.76)

$$4\ 2\ 2\ 8\ 0\ 0\ 0\ 0_{16} = [0100\ 0010\ 0010\ 1000\ 0000\ 0000\ 0000\ 0000]_2$$

วิธีทำ

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

2. จะพบว่าบิตเครื่องหมาย $s = 0$

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

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

$$F_{10,IEEE} = (-1)^0 \times (1 + .0101_2) \times 2^5 \quad (2.77)$$

$$= (-1)^0 \times (1.0101_2) \times 2^5 \quad (2.78)$$

$$= (-1)^0 \times (101010.0_2) \quad (2.79)$$

$$= (+1) \times (32 + 8 + 2) \quad (2.80)$$

$$= 42.0_{10} \quad (2.81)$$

ผู้อ่านสามารถดูการจัดเรียงตัวของเลขฐานสิบ 42.0 นี้ในหน่วยความจำของตัวแปร a ในรูปที่ 2.2

ตัวอย่างที่ 2.6.2 เลขทศนิยมฐานสองชนิดจุดลอยตัวมาตรฐาน IEEE754 ชนิด Single Precision ต่อไปนี้ มีค่าเท่าไรในฐานสิบตามสมการที่ (2.76)

$$C\ 0\ A\ 0\ 0\ 0\ 0_{16} = [1100\ 0000\ 1010\ 0000\ 0000\ 0000\ 0000]_2$$

วิธีทำ

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

2. จะพบว่าบิตเครื่องหมาย $s = 1$

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

ค่าทศนิยม $Y_2 = 010\ 0000\ 0000\ 0000\ 0000\ 0000_2$

$$F_{10,IEEE} = (-1)^1 \times (1 + .01_2) \times 2^2 \quad (2.82)$$

$$= (-1) \times (1.01_2) \times 2^2 \quad (2.83)$$

$$= (-1) \times (101.0_2) \quad (2.84)$$

$$= (-1) \times 5.0 \quad (2.85)$$

$$= -5.0_{10} \quad (2.86)$$

โปรด สังเกต ความแตกต่าง ระหว่าง เลข ทศนิยมฐานสอง ชนิด จุด ลอยตัว ความยาว 32 บิต ตามมาตรฐาน IEEE754 ในตัวอย่างนี้ กับ เลขจำนวนเต็มฐานสอง ความยาว 32 บิต ชนิด มีเครื่องหมาย ในตัวอย่างที่ ?? ซึ่ง ตรงกับ -5_{10} หั้งคู่

ตัวอย่างที่ 2.6.3 จงหาค่า ทศนิยมฐานลิบของเลข FP-32 ที่ เติมในรูปที่ 2.7 คือ เลข ทศนิยมมาตรฐาน IEEE754 ชนิด Single Precision ตามสมการที่ (2.76) ดังต่อไปนี้ $[0][011\ 1110\ 0][010\ 0000\ 0000\ 0000\ 0000]$

วิธีทำ

$$F_{10,IEEE} = (-1)^0 \times 1.01 \times 2^{(124-127)} \quad (2.87)$$

$$= 1 \times 1.01 \times 2^{-3} \quad (2.88)$$

$$= 0.00101_2 \quad (2.89)$$

$$= 2^{-3} + 2^{-5} \quad (2.90)$$

$$= \frac{1}{2^3} + \frac{1}{2^5} \quad (2.91)$$

$$= 0.125_{10} + 0.03125_{10} = 0.15625_{10} \quad (2.92)$$

ตัวอย่างที่ 2.6.4 จงแปลง เลข -0.75_{10} เป็น เลข ทศนิยมฐานสอง ชนิด จุด ลอยตัว ตาม มาตรฐาน IEEE754 หั้งสองชนิด

วิธีทำ

1. แปลง เลข ทศนิยมฐานลิบ ให้อยู่ในรูปฐานสองแบบ อร์มัลไลช์

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

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

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

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

2. ทำการน้อมลัลซ์ ตามสมการที่ (2.68)

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

ดังนั้น บิตเครื่องหมาย $s = 1$

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

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} เขียนในรูปของเลขคณิตฐานสองชนิดจุดลอยตัวตามมาตรฐาน IEEE754 ชนิด

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

Double: $[1][011\ 1111\ 1110][1000\ 0000\ 0000\dots0000]_2 = BFE8\ 0000\ 0000\ 0000_{16}$

2.6.2 ค่าสูงสุด ต่ำสุดและค่าอื่น ๆ ของมาตรฐาน IEEE754

การคำนวณค่าของตัวแปรชนิด float และ double ซอฟต์แวร์ในเครื่องคอมพิวเตอร์สามารถรองรับการคำนวณโดยใช้เลขคณิตฐานสองชนิดจุดลอยตัวตามมาตรฐาน IEEE754 แต่เนื่องจากเลขจำนวนจริงในทางคณิตศาสตร์มีจำนวนอนันต์ (Infinite) เลขคณิตฐานสองชนิดจุดลอยตัวตามมาตรฐาน IEEE754 จึงไม่สามารถรองรับได้ทั้งหมด และมีข้อจำกัดทางคณิตศาสตร์ ตามที่สรุปในตารางที่ 2.12

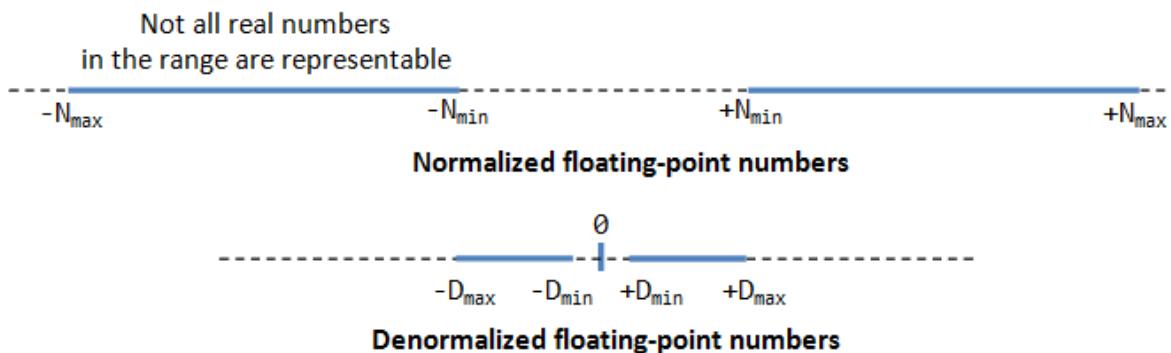
ตารางที่ 2.12: ตาราง สรุป ความแตกต่างระหว่าง เลขคณิตฐานสอง ชนิด จุด ลอยตัว ตาม มาตรฐาน IEEE754 ชนิด Single Precision โดย $E_{2,IEEE}$ คือ ค่ายกกำลัง และ Y_2 คือ ทศนิยมซึ่งเป็นส่วนประกอบของค่านัยสำคัญ (\times หมายเลขถึง บิตข้อมูลมีค่าเท่ากับ '0' หรือ '1')

ลำดับที่	เครื่องหมาย s (1 บิต)	ค่ายกกำลัง $E_{2,IEEE}$ (8 บิต) สมการ (2.75)	ทศนิยม Y_2 (23 บิต) สมการ (2.70)	ความหมาย
1	±	$0000\ 0001_2 - 1111\ 1110_2$	$xx..2$	เลขฐานสิบห้าไป (นอร์มัลไลซ์) สมการที่ (2.76)
2	±	$0000\ 0000_2$	$xx..2$	เลขฐานสิบห้าอยมาก แต่ไม่เท่ากับศูนย์ (ดีนอร์มัลไลซ์)
3	0	$0000\ 0000_2$	$0...0_2$	0.0_{10} (ศูนย์จุดศูนย์)
4	±	$1111\ 1111_2$	$0...0_2$	$\pm\infty$ (±อินฟินิตี้)
5	0	$1111\ 1111_2$	$xx..2$	Nan (Not a Number)

ผู้อ่านสามารถทำความเข้าใจตารางที่ 2.12 ตามลำดับที่ดังต่อไปนี้

1. เลขฐานสิบห้าไป (นอร์มัลไลซ์) ทั้งบวกและลบ เป็นย่างของเลขคณิตที่มาตรฐานสามารถแสดงค่าได้ โดยมีค่าสัมบูรณ์ (Absolute Value) อよรุระหว่างค่าต่ำสุด (N_{min}) ที่เข้าใกล้ศูนย์ จนถึง ค่าที่สูงมาก (N_{max}) ในรูปที่ 2.8 ตามค่าฐานสิบในสมการที่ (2.76)

2. เลขทศนิยมที่มีค่าสัมบูรณ์น้อยมากอยู่ในช่วง (D_{min} ถึง D_{max}) ในรูปที่ 2.8 เรียกว่า ค่าดีนอร์มัลไลซ์ (Denormalize)
3. เลขทศนิยม 0.0_{10} หรือ ศูนย์จุดศูนย์ เรียกว่า ศูนย์จริง (True zero)
4. ค่าบวกและลบอนันต์หรืออินฟินิตี้ สามารถใช้แทนผลลัพธ์ที่มีค่าสูงเกินขอบเขตของมาตรฐาน เนื่องจากผลลัพธ์ที่คำนวณได้ทางคณิตศาสตร์ เช่น การคูณและการบวกเลขขนาดใหญ่มากสองจำนวน หรือการหารด้วยตัวหารขนาดเล็กมากจนผลหารที่ได้เลยขอบเขตค่าสูงสุดที่มาตรฐานกำหนด
5. NaN เป็นสัญลักษณ์ซึ่งไม่สามารถแทนได้ด้วยตัวเลข ย่อมาจากคำว่า Not a Number แสดงถึงความผิดพลาดที่อาจเกิดจาก การเขียนโปรแกรมเพื่อใช้ตัวแปรหารค่าตัวตั้ง แต่ตัวแปรที่ใช้หารมีค่าที่เปลี่ยนแปลงไปจนกลายเป็น 0.0_{10} เมื่อนำไปหารจึงกลายเป็นสัญลักษณ์นี้ เพื่อบอกข้อผิดพลาดที่เกิดขึ้นแล้ว



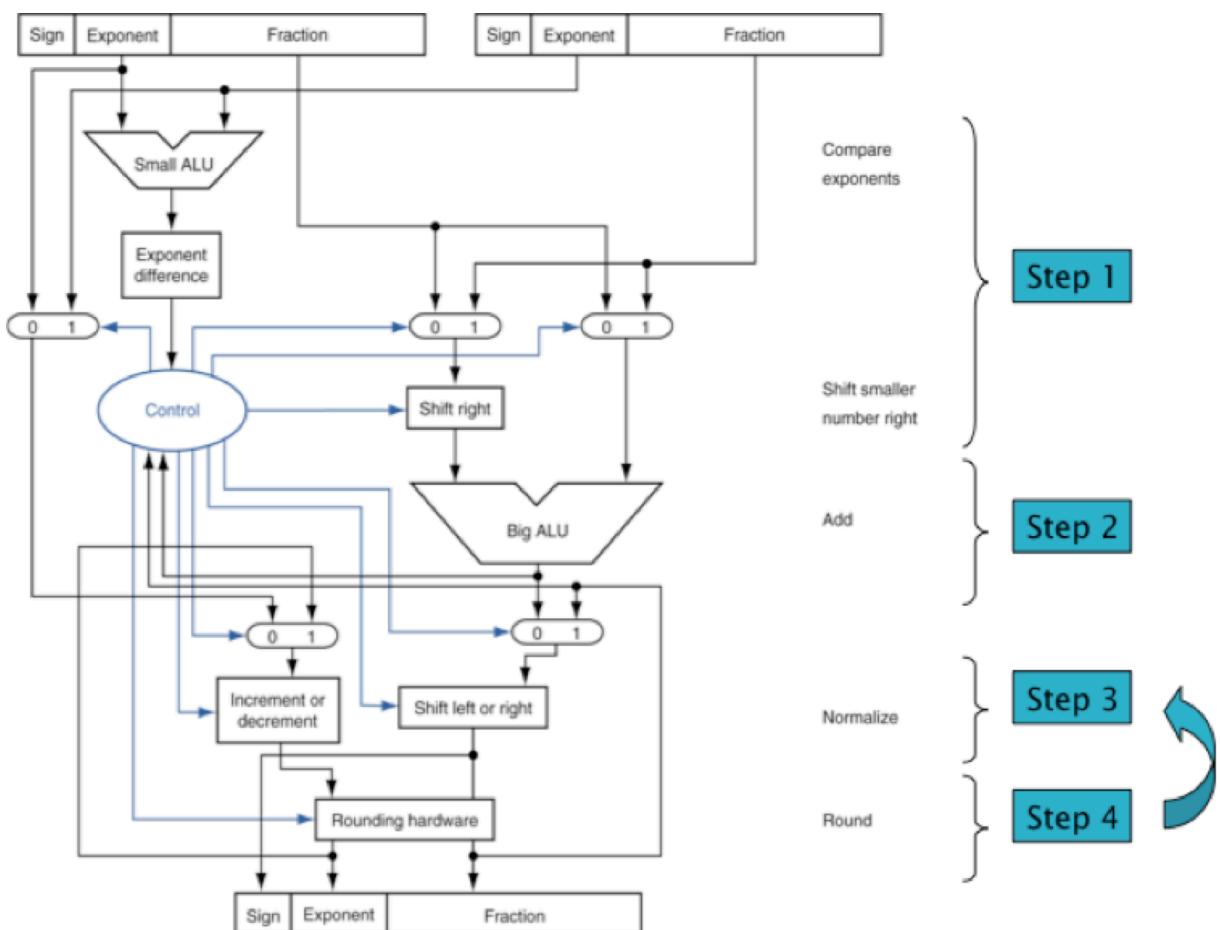
รูปที่ 2.8: เลขทศนิยมโดยตัวชนิด นอร์มัลไลซ์ ตั้งแต่ $\pm N_{min}$ ถึง $\pm N_{max}$ และชนิด ดีนอร์มัลไลซ์ ตั้งแต่ $\pm D_{min}$ ถึง $\pm D_{max}$ ที่มา: ntu.edu.sg

รูปที่ 2.8 แสดงค่าบนเส้นจำนวนของลำดับต่าง ๆ ในตารางที่ 2.12 ยกเว้น NaN เส้นด้านบนของรูปแสดงเลขต่าง ๆ ที่เขียนในรูปนอร์มัลไลซ์ได้ ซึ่งตรงกับลำดับที่ 1 เลขต่าง ๆ ที่เขียนในรูปนอร์มัลไลซ์ได้ จากซ้ายสุดถึงขวาสุดของเส้นจำนวน 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 แสดงเลขขนาดเล็กมาก ๆ ที่ไม่สามารถเขียนในรูปนอร์มัลไลซ์ได้ เรียกว่า เลขดีนอร์มัลไลซ์ ตรงกับลำดับที่ 2 ในตารางที่ 2.12

2.6.3 การบวกเลขคณิตฐานสองชนิดจุดลอยตัวตาม IEEE754

การบวกเลขคณิตฐานสองชนิดจุดลอยตัวตามมาตรฐาน IEEE754 จำเป็นต้องอาศัยวงจรสารดิจิทัลหรือ (Floating Point ALU) ช่วยคำนวณแบบไบพาสไลน์ (Pipeline) ซึ่งมีหลักการคล้ายกับวงจรประมวลผลยนต์ทำให้ซีพียู 1 แกนประมวลผลสามารถคำนวณการบวกเลขคณิตได้ในหลัก กิกะฟล็อปส์ (Giga Floating Point Operations Per Second: Giga FLOPS) หรือ พันล้านหรือ 10^9 ครั้งต่อวินาที ซึ่งการวัดสมรรถนะ (Performance) ของซีพียู รวมไปถึงซูเปอร์คอมพิวเตอร์ (Supercomputer) ในปัจจุบันใช้หน่วยวัดเพتاฟล็อปส์ (Peta FLOPS) หรือ 10^{15} ครั้งต่อวินาที ผู้อ่านสามารถศึกษาเรื่อง ฟล็อปส์ ได้ที่ [wikipedia](#)

ดังนั้น การทำงานของวงจรบวก/ลบเลขคณิตชนิดจุดลอยตัว จึงอาศัยหลักการเดียวกันกับ อัลกอริธึมการบวก ซึ่งมีความซับซ้อนใกล้เคียงกับการคูณเลขคณิต และทั้งคู่มีความซับซ้อนมากกว่าและใช้เวลานานกว่าการบวกและการคูณเลขจำนวนเต็มที่จำนวนบิตเท่ากัน



รูปที่ 2.9: วงจรบวกเลขชนิดจุดลอยตัวตามมาตรฐาน IEEE754 โดยรับค่าอินพุตจำนวน 2 ตัวด้านบน และเอาต์พุตผลลัพธ์ด้านล่าง ที่มา: [Patterson and Hennessy \(2016\)](#)

วงจรบวกเลขคณิตฐานสองชนิดจุดลอยตัวในรูปที่ 2.9 มีความซับซ้อนใกล้เคียงกับตัวอย่างที่แสดงในหัวข้อที่ 2.5 ที่ผ่านมา ดังนี้

- ขั้นตอนที่ (Step) 1 คือ ใช้งาน Small ALU ในรูปที่ 2.9 หากผลต่างของเลขยกกำลังจากเลขทั้ง

สองตัว ทำการเลื่อน (Shift) จุดศูนย์ของค่านัยสำคัญของเลขที่มีค่ายกกำลังน้อยกว่าไปทางขวา เป็นจำนวนบิตเท่ากับค่าสัมบูรณ์ (Absolute) ของผลต่าง ส่วนค่านัยสำคัญของเลขที่มีค่ายกกำลังมากกว่าจะไม่เปลี่ยนแปลง

- ขั้นตอนที่ (Step) 2 คือ ทำการบวก/ลบค่านัยสำคัญที่เลื่อนตำแหน่งแล้วเข้าด้วยกันโดยใช้วงจร ALU ขนาดใหญ่ (Big ALU) ในรูปที่ 2.9
- ขั้นตอนที่ (Step) 3 คือ ทำการวนอ้อมลัลเลซ์ค่านัยสำคัญของผลลัพธ์ และตรวจเช็คการเกิดโอเวอร์โฟลว์ และอันเดอร์โฟลว์ ซึ่งผลลัพธ์ที่ได้จากการบวกอาจมีค่าเพิ่มขึ้น หรือลดลงขึ้นอยู่กับเครื่องหมายและขนาดของเลขทั้งสองตัว ทำให้ต้องวนอ้อมลัลเลซ์ค่านัยสำคัญอีกรอบ
 - หากค่านัยสำคัญสัมบูรณ์ของผลลัพธ์เพิ่มมากขึ้นจนเกินค่าสูงสุดในลำดับที่ 1 ของตารางที่ 2.12 เรียกว่าเกิด โอเวอร์โฟลว์ (Overflow) ของเลข IEEE754
 - หากค่านัยสำคัญสัมบูรณ์ของผลลัพธ์มีค่าเข้าใกล้ศูนย์จนน้อยกว่าค่าต่ำสุดในลำดับที่ 1 ของตารางที่ 2.12 และไม่สามารถเขียนในรูปนอร์มัลໄลซ์ได้ ผลลัพธ์นั้นจะเขียนในรูปแบบดีนอร์มัลໄลซ์ (Denormalize) แทน เรียกว่าเกิดอันเดอร์โฟลว์ (Underflow) ของเลข IEEE754
- ขั้นตอนที่ (Step) 4 คือ ปัดค่านัยสำคัญให้เหลือ 24 บิตและอาจต้องวนอ้อมลัลเลซ์ค่านัยสำคัญอีกรอบ หากจำเป็น

ตัวอย่างที่ 2.6.5 การบวกเลขทศนิยมฐานสองชนิดจุดลอยตัว (Single Precision) มาตรฐาน IEEE754

$$-5_{10} + -0.75_{10} \text{ หรือเท่ากับ } C0A0\ 0000_{16} + BF40\ 0000_{16} = ?$$

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

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

วิธีทำ

ผู้อ่านจะต้องแปลงเลขฐานสองให้เป็นรูปแบบเลขฐานสองที่นอร์มัลໄลซ์ 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. เลื่อนตำแหน่งจุดศูนย์ของเลขนัยสำคัญและปรับเลขยกกำลังที่มีค่าสัมบูรณ์น้อยกว่าให้ตรงกับเลขยกกำลังของเลขที่มีค่าสัมบูรณ์มากกว่า

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

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

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

$$+1_{10} + -0.75_{10} \text{ หรือเท่ากับ } 3F80\ 0000_{16} + BF40\ 0000_{16} = ?$$

$$0011\ 1111\ 1000\ 0000\ 0000\ 0000\ 0000_2 + \\ 1011\ 1111\ 0100\ 0000\ 0000\ 0000\ 0000_2$$

วิธีทำ

ผู้อ่านจะต้องแปลงเลขฐานสองให้เป็นรูปแบบเลขฐานสองที่นอร์มัลไลซ์ ดังต่อไปนี้
 $= (-1)^0 \times (1.000\ 0000\ 0000\ 0000\ 0000)_2 \times 2^0 +$
 $(-1)^1 \times (1.100\ 0000\ 0000\ 0000\ 0000)_2 \times 2^{-1}$

1. เลื่อนตำแหน่งจุดทศนิยมของเลขนัยสำคัญและปรับเลขยกกำลังที่มีค่าสัมบูรณ์น้อยกว่า ให้ตรงกับเลขยกกำลังของเลขที่มีค่าสัมบูรณ์มากกว่า

$$(-1)^0 \times (1.000\ 0000\ 0000\ 0000\ 0000)_2 \times 2^0 + \\ (-1)^1 \times (0.110\ 0000\ 0000\ 0000\ 0000)_2 \times 2^0$$

2. นำเลขนัยสำคัญที่เป็นตัวตั้ง (ที่มีเลขกำลังมากกว่า) หักลบด้วย เลขนัยสำคัญที่เป็นตัวลบ (ที่มีเลขกำลังน้อยกว่า) สังเกตได้จาก Sign bit ด้านหน้า จะได้ผลลัพธ์ดังนี้
 $(-1)^0 \times (0.010\ 0000\ 0000\ 0000\ 0000_2 \times 2^0)$

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

4. ปัดค่านัยสำคัญให้เหลือ 24 บิตและอาจต้องนอร์มัลไลซ์อีกรอบหากจำเป็น

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

$$s = 0$$

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

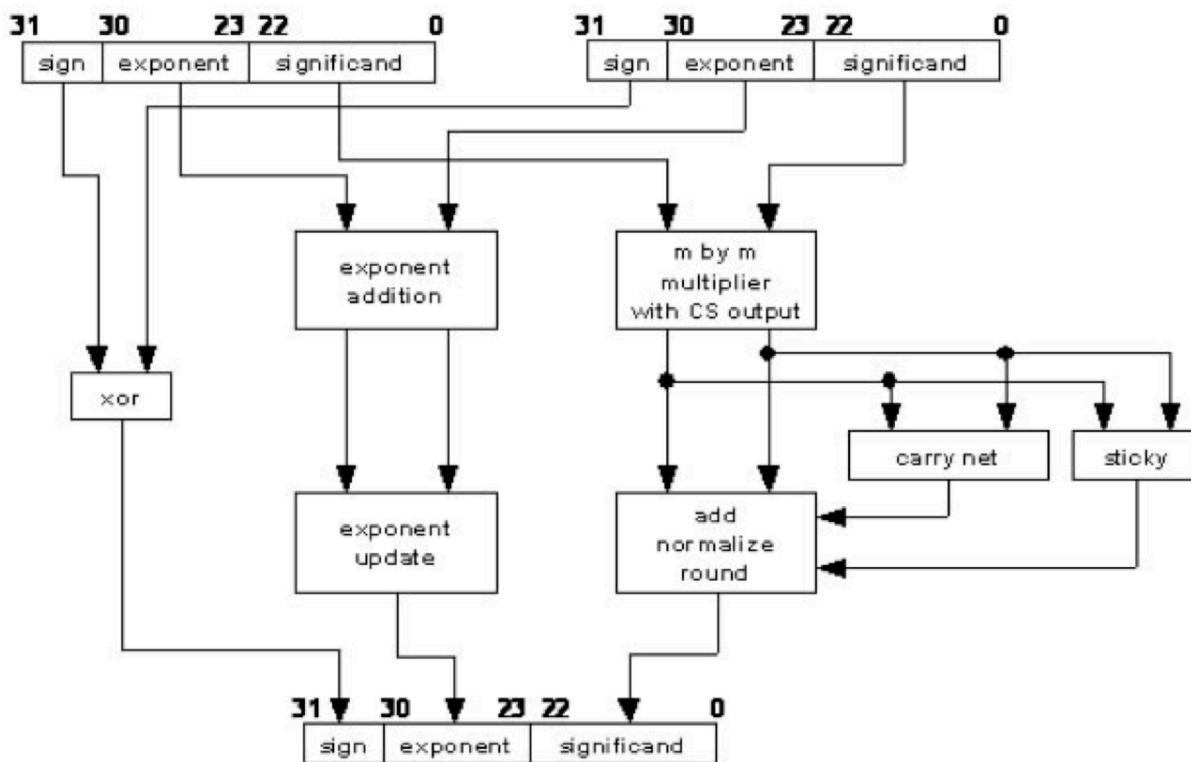
$$\text{ค่าทศนิยม } Y_2 = 000\ 0000\ 0000\ 0000\ 0000_2$$

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

โดยสรุป ตัวอย่างที่ 2.6.5 และ 2.6.6 แสดงให้เห็นว่า การบวกเลขทศนิยมฐานสองชนิดจุดลอยตัวตามมาตรฐาน IEEE754 ภายในเครื่องคอมพิวเตอร์โดยใช้รหัสมาตรฐาน IEEE754 สามารถทำงานตามขั้นตอนต่อๆ ตามบล็อกได้อย่างในรูปที่ 2.9

2.6.4 การคูณเลขทศนิยมฐานสองชนิดจุดลอยตัวตาม IEEE754

การคูณเลขทศนิยมฐานสองชนิดจุดลอยตัวตามมาตรฐาน IEEE754 ด้วยความเร็วสูงจำเป็นต้องอาศัยวงจราร์ดแวร์พิเศษ แต่การทำงานของวงจรนั้นในหลักการคล้ายกับอัลกอริธึมการคูณ ในหัวข้อที่ 2.3.1 ดังนั้น ผู้อ่านจำเป็นต้องศึกษาอัลกอริธึมให้เข้าใจอย่างถ่องแท้ก่อน



รูปที่ 2.10: วงจรคูณเลขทศนิยมฐานสองชนิดจุดลอยตัวตามมาตรฐาน IEEE754 โดยรับค่าอินพุตจำนวน 2 ตัวด้านบน และเอาต์พุตผลลัพธ์ด้านล่าง ที่มา: Patterson and Hennessy (2016)

วงจรคูณเลขทศนิยมฐานสองชนิดจุดลอยตัว ในรูปที่ 2.10 มีความซับซ้อนมากกว่า วงจรคูณเลขจำนวนเต็มในรูปที่ 2.5 และ 2.6 ดังนี้

- ขั้นตอนที่ 1 คือ ทำการคูณเฉพาะค่านัยสำคัญทั้งสองเพื่อหาค่านัยสำคัญของผลคูณ
- ขั้นตอนที่ 2 คือ ทำการบวกค่ายกกำลังทั้งสองเข้าด้วยกันแล้วลบค่าไบอสหนึ่งครั้ง
- ขั้นตอนที่ 3 คือ ทำการวนอ้อมลัลซ์ค่านัยสำคัญของผลลัพธ์ และตรวจเช็คโอล์ฟล์ และอันเดอร์โอล์ฟ เหมือนกับกรณีการบวกเลข

- ขั้นตอนที่ (Step) 4 คือ ปัดค่านัยสำคัญให้เหลือ 24 บิตและอาจต้องนำรีมัลล์ไซร์อีกรอบเมื่อจำเป็น

ตัวอย่างที่ 2.6.7 จงคูณเลขคณิตฐานสองชนิดจุดลอยตัว 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. แปลงเลขฐานสองตามากถูกของ IEEE754 Single Precision รูปแบบเลขฐานสองที่นำรีมัลล์ไซร์ 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

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

5. นำรีมัลล์ไซร์ค่านัยสำคัญของผลลัพธ์ และตรวจเช็คการเกิดโอเวอร์ไฟล์ว์และอันเดอร์ไฟล์ว์

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

6. ปัดค่านัยสำคัญให้เหลือ 24 บิตและอาจต้องนำรีมัลล์ไซร์อีกรอบหากจำเป็น

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

7. ปรับเครื่องหมายของผลลัพธ์ให้ถูกต้องจากตัวตั้งและตัวคูณ

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

$$\text{บิตเครื่องหมาย } s = 0$$

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

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

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

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

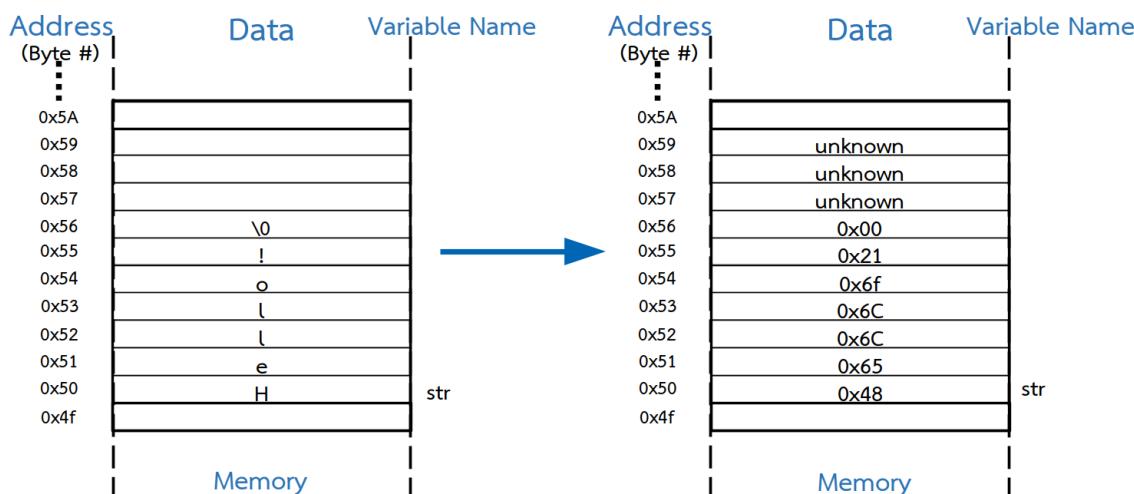
$$COA0\ 0000_{16} \times BF40\ 0000_{16} = 4070\ 0000_{16}$$

โดยสรุป ตัวอย่างที่ 2.6.7 แสดงให้เห็นว่า การคูณเลขทศนิยมฐานสองชนิดจุดลอยตัว IEEE754 ภายในเครื่องคอมพิวเตอร์ตามมาตรฐาน IEEE754 สามารถทำงานตามขั้นตอนต่าง ๆ ตามบล็อกໄ/doze แกรมในรูปที่ 2.10

2.7 ตัวอักษร (Character) และข้อความ (String)

ตัวอักษรในภาษาต่าง ๆ ล้วนเป็นข้อมูลที่สำคัญในการใช้งานคอมพิวเตอร์ โดยเฉพาะชื่อ นามสกุล หนังสือ ตำรา ข้อความ คำสนทนาระหว่างผู้คน ในเครือข่ายอินเทอร์เน็ต เพื่อนำมาสืบค้น สร้างความเชื่อมโยง ในหลากหลายมิติ ผ่านสื่อสังคมออนไลน์ต่าง ๆ ได้อย่างแพร่หลาย รูปที่ 2.11 แสดงการเก็บข้อความ (String) ในหน่วยความจำด้วยรหัสแอสกี (ASCII) ของตัวแปรข้อความหรือตัวแปรชนิดสตริงซึ่งชื่อว่า str ซึ่งเป็นตัวแปรชนิดอาร์เรย์ของตัวอักษร (Array of Character)

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



รูปที่ 2.11: การเก็บข้อความในหน่วยความจำในรูปของรหัสแอสกีด้วยตัวแปรสตริงซึ่งว่า str ซึ่งเป็นตัวแปรชนิด char[10] str="Hello!" ที่แอดเดรสเริ่มต้น 0x50 หรือ 50₁₆ ที่มา: [Harris and Harris \(2013\)](#)

การเรียงตัวอักษรในหน่วยความจำจะเริ่มต้นจากหมายเลขไบต์หรือแอดเดรสเริ่มต้น 0x50 หรือ 50₁₆ ไปยังแอดเดรสที่สูงขึ้นตามรูปที่ 2.11 โดยแอดเดรส 0x50 เก็บรหัสแอสกีหมายเลข 0x48 หรือ 48₁₆ ซึ่งตรงกับตัวอักษร 'H' ไปจนถึงแอดเดรส 0x56 ซึ่งเก็บตัวอักษรรหัส 0x00 หรือ NULL อ่านว่า นัล ซึ่งเป็นอักษรพิเศษแสดงว่าจบประโยค ส่วนไบต์ที่ 0x57, 0x58, 0x59 จะมีค่าเป็นเลขได้ ๆ (unknown) เพราะไม่มีผลต่อประโยคที่เก็บ ตัวแปร str นี้สามารถเก็บตัวอักษรจำนวนสูงสุด 9 ตัวและตัวสุดท้ายจะต้องเป็นตัวอักษร NULL เท่านั้น ยกตัวอย่างเช่น ประโยค "012345678" ประโยค "abcdefghijkl" เป็นต้น

รหัส æ อสกี (ASCII) คือ มาตรฐานของรูปแบบการใช้เลขฐานสองเพื่อแทนตัวอักษรตั้งแต่อดีต ซึ่งกำหนดขึ้นมาโดยหน่วยงานชื่อว่า ANSI (American National Standard Institute) รูปที่ 2.12 คือตารางรหัส ASCII กำหนดเลขฐานสองขนาด 8 บิต ที่มีแทนตัวอักษรจำนวน $2^8=256$ ตัว โดยมีหมายเลขเริ่มต้น คือ $00_{10} = 0000\ 0000_2 = 00_{16}$ 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

-0	-1	-2	-3	-4	-5	-6	-7	-8	-9	-A	-B	-C	-D	-E	-F	
0-	0001	0002	0003	0004	0005	0006	0007	0008	0009	000A	000B	000C	000D	000E	000F	
1-	0010	0011	0012	0013	0014	0015	0016	0017	0018	0019	001A	001B	001C	001D	001E	001F
2-	0020	!	"	#	\$	%	&	'	()	*	+	,	-	.	/
3-	0030	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>
4-	0040	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N
5-	0050	P	Q	R	S	T	U	V	W	X	Y	Z	[\	^	-
6-	0060	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n
7-	0070	p	q	r	s	t	u	v	w	x	y	z	{		}	~
8-																
9-																
A-	0E48	'	ກ	ຂ	ໝ	ຄ	ໜ	ໝ	່	ຈ	້	໇	໌	໔	໖	໘
B-	0E10	ສ	ໜ	໛	ດ	ໜ	ດ	ກ	ດ	ນ	ປ	ປ	ຜ	ຜ	ພ	ພ
C-	0E20	ກ	ນ	ຢ	ຮ	ດ	ລ	ກ	ວ	ສ	ໜ	ໜ	ໜ	ໜ	ໜ	ໜ
D-	0E30	ສ	ວ	ກ	ກ	ດ	ດ	ດ	ດ	ດ	ດ	ດ	ດ	ດ	ດ	ດ
E-	0E40	ເ	ແ	ໂ	ຈ	ົ	ົ	ົ	ົ	ົ	ົ	ົ	ົ	ົ	ົ	ົ
F-	0E50	ອ	ອ	ໂ	ຕ	ແ	ແ	ແ	ແ	ແ	ກ	ກ	ກ	ກ	ກ	ກ

รูปที่ 2.12: ตารางรหัส ASCII และ Unicode สำหรับตัวอักษรจำนวน $2^8=256$ ตัว ประกอบด้วยรหัสของตัวอักษรภาษาอังกฤษและภาษาไทย ที่มา: ascii-table.com

รหัส [Unicode](#) ถูกกำหนดให้เป็นมาตรฐานโดย ISO (International Standard Organization) เพื่อใช้ทดแทนรหัส ASCII และรองรับการใช้ภาษาต่าง ๆ 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 ตัวอักษร ซึ่งทำให้สามารถใช้เลขฐานสองจำนวน 2^{16} หรือ 65,536 แบบมาแทนตัวอักษรได้ແບບทุกตัวอักษรในภาษาสำคัญทั่วโลก ซึ่งทำให้วิธีการนี้ได้รับความนิยมน้อย โดยตำแหน่งเริ่มต้นของตารางรหัส Unicode จะเหมือนกับตารางรหัส ASCII ตัวภาษาอังกฤษและภาษาไทยในรูปที่ 2.12 ภายใต้ตัวอักษรมีค่ารหัส Unicode กำกับอยู่ด้วยยกตัวอย่างเช่น ตัวอักษรไทยเริ่มต้นที่ รหัส ASCII ของอักษร ก เท่ากับ A1₁₆ หรือ 1010 0001₂ ในรูปของเลขฐานสองขนาด 8 บิต และตรงกับรหัส Unicode คือ 0E01₁₆ ความยาว 16 บิต
- รหัส UTF-8 นิยมใช้ในเว็บเพจต่าง ๆ โดยแต่ละตัวอักษรจะใช้ความยาวตั้งแต่ 1 ไบต์ จนถึง 4 ไบต์ โดยตัวอักษร 128 ตัวแรก คือ รหัส ASCII ใช้ความยาว 1 ไบต์ ส่วนตัวอักษรในภาษาอื่น ๆ จะใช้จำนวนไบต์เพิ่มขึ้น
- รหัส UTF-16 คือ การขยายรหัส UCS-2 ให้ทันสมัยมากขึ้น โดยเพิ่มการเข้ารหัสเป็นขนาด 4 ไบต์ ด้วย

การทดลองในหัวข้อที่ A.3 ภาคผนวก A จะเปิดโอกาสให้ผู้อ่านได้ทดลองแปลงตัวอักษรต่าง ๆ ให้เป็นรหัส ASCII และรหัส Unicode ชนิด UCS-2 ด้วยตนเอง เพื่อสร้างความเข้าใจที่ลึกซึ้งมากขึ้น

2.8 สรุปท้ายบท

ตารางที่ 2.13: ชนิด ความยาว ข้อมูล และการประยุกต์ใช้งานเลขฐานสองชนิดต่าง ๆ 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	แอดเดรสซึ่งตำแหน่งข้อมูล ระบบ 32/64 บิต
unsigned int	32/64	จำนวน	จำนวนอุปกรณ์ I/O 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	ระยะทางไปยังดาวต่าง ๆ นอกโลก
double	64	± หน้างาน	หน้างานกัดดาวต่าง ๆ นำหนักอนุภาคเล็ก ๆ

การคำนวณทางคณิตศาสตร์ของเลขจำนวนเต็ม ชนิด มีเครื่องหมาย และไม่มีเครื่องหมาย จำเป็นต้องตรวจสอบ การเกิดโอเวอร์โฟล์ว เนื่องจาก ฮาร์ดแวร์ หรือ โปรเซสเซอร์ มีจำนวนบิตในการคำนวณที่จำกัด ปัจจุบัน คือ 32, 64 และ 128 บิต ในทำนองเดียวกัน การคำนวณโดยใช้เลขทศนิยมชนิดจุดตรึง (Fixed-point) และจุดลอยตัว (Floating-point) จำเป็นต้องตรวจสอบการเกิดโอเวอร์โฟล์วและอันเดอร์โฟล์ว และกรณีอื่น ๆ ด้วยเหตุผลที่ซับซ้อนกว่าเลขจำนวนเต็ม ทำให้ใช้ทรัพยากรด้านฮาร์ดแวร์และเวลามากขึ้น

ชนิด และ ความ ยาว ของ ข้อมูล ที่ หลัก หล่าย ตาม ที่ สรุป ใน ตาราง ที่ 2.13 ของ ซอฟต์แวร์ คอมพิวเตอร์ ต้องการ ความ สามารถ ของ จีพีयู หรือ ฮาร์ดแวร์ ให้ รองรับ ตาม เช่น กัน เพื่อ ให้ ประสบการณ์ การ ใช้งาน ของ ผู้ใช้ (User Experience) ดี ขึ้น เรื่อยๆ ยก ตัวอย่าง เช่น การ ใช้ จีพียู มา ช่วย คำนวณ ข้อมูล ที่ จัด เรียง ตัว กัน แบบ เวกเตอร์ (Vector) เพื่อ เพิ่ม ประสิทธิภาพ เป็น ต้น

2.9 คำถ้ามท้ายบท

1. จง แสดง วิธี บวก เลข จำนวนเต็ม ฐาน สิบ และ ฐาน สอง ชนิด ไม่มี เครื่องหมาย ขนาด 8 บิต ต่อไปนี้ และ ตรวจสอบว่า เกิด โอล์ฟล์ ตาม สมการ ที่ (2.52)
 - $255_{10} + 1_{10}$
 - $128_{10} + 127_{10}$
 - $1111\ 1110_2 + 0000\ 0011_2$
 - $1000\ 0000_2 + 0111\ 1111_2$
2. จง แสดง วิธี คูณ เลข จำนวนเต็ม ฐาน สอง ชนิด ไม่มี เครื่องหมาย ขนาด 8 บิต ต่อไปนี้ ด้วย วิธี คูณ เลข ชนิด ที่ 1 ใน รูป ที่ 2.5 ตาม ตัวอย่าง ที่ 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. จง แสดง วิธี คูณ เลข จำนวนเต็ม ฐาน สอง ชนิด ไม่มี เครื่องหมาย ขนาด 8 บิต ต่อไปนี้ ด้วย วิธี คูณ เลข ชนิด ที่ 2 ใน รูป ที่ 2.6 ตาม ตัวอย่าง ที่ 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. จง แสดง วิธี บวก เลข จำนวนเต็ม ฐาน สิบ และ ฐาน สอง ชนิด มี เครื่องหมาย 2's Complement ขนาด 8 บิต ต่อไปนี้ และ ตรวจสอบว่า เกิด โอล์ฟล์ ตาม สมการ ที่ (2.56)
 - $126_{10} + 2_{10}$
 - $128_{10} - 127_{10}$

- $1111\ 1110_2 + 0000\ 0011_2$

- $1000\ 0000_2 + 0111\ 1111_2$

5. จงแสดงวิธีบวกเลขทศนิยมชนิดจุดตรีง 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 และตรวจสอบว่าเกิดโอเวอร์โฟล์ว/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 และตรวจสอบว่าเกิดโอเวอร์โฟล์ว/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

- $0x0000\ 0000$

- $0x8000\ 0000$

- $0xFF80\ 0000$

- $0x8F80\ 0000$

- $0x8F80\ 0001$

- $0x8FFF\ FFFF$

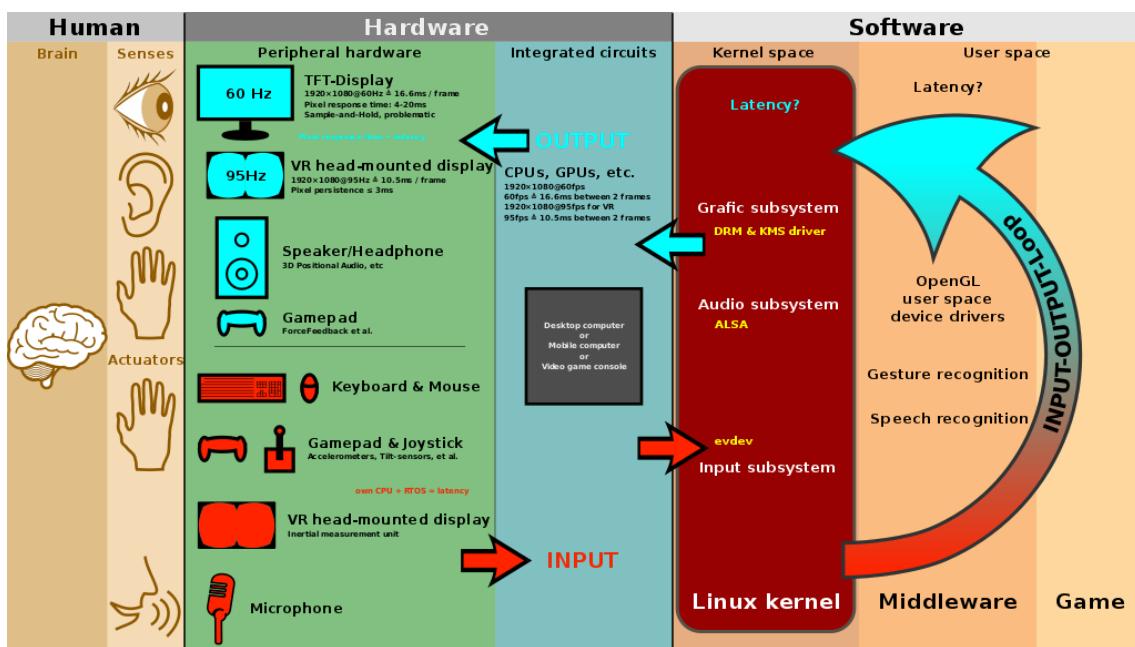
แต่ละตัวแล้วเติมลงในตาราง

- เลขจำนวนเต็มแบบไม่มีเครื่องหมาย โดยใช้สมการที่ (2.2)
- เลขจำนวนเต็มแบบมีเครื่องหมาย ชนิด 2's Complement โดยใช้สมการที่ (2.20)
- 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 (2.20)	$F_{10,IEEE}$ ค่าฐานสิบ IEEE754 สมการ (2.76)
0x0000 0000			
0x8000 0000			
0xFF80 0000			
0x8F80 0000			
0x8F80 0001			
0xFFFF FFFF			

บทที่ 3

ฮาร์ดแวร์และซอฟต์แวร์ของคอมพิวเตอร์ (Computer Hardware and Software)



รูปที่ 3.1: วงรอบการเชื่อมโยงอินพุต-เอาต์พุต (Input-Output Loop) ระหว่างผู้ใช้ ฮาร์ดแวร์ และซอฟต์แวร์ (เกม) ที่มา: wikipedia.org

รูปที่ 3.1 แสดงวงรอบการเชื่อมโยงอินพุต-เอาต์พุต (Input-Output Loop) ระหว่างผู้ใช้ ฮาร์ดแวร์ และซอฟต์แวร์ (เกม) ซึ่งเริ่มต้นจากการที่ผู้ใช้เรียกใช้งานซอฟต์แวร์ (เกม) นั้นผ่านทางเมนูของระบบปฏิบัติการ เมื่อซอฟต์แวร์เริ่มต้นทำงาน ผู้ใช้สามารถรับรู้ผ่านทางมือ ตาและหู เนื่องจากซอฟต์แวร์สร้างแรงตอบสนอง ภาพและเสียง ผ่านทางฮาร์ดแวร์ เช่น เกมแพด (Gamepad) จอภาร และลำโพง เมื่อสมองผู้ใช้รับรู้ ประมวลผล และตอบสนองต่อซอฟต์แวร์ ด้วยการขยับศีรษะ มือ และปากผ่านทางฮาร์ดแวร์ เช่น จอแสดงภาพแบบสามมิติ (Head Mounted Display) คีย์บอร์ด เม้าส์ เกมแพด ไมโครโฟน เป็นต้น กลับไปหาซอฟต์แวร์ (เกม) วงรอบนี้จะเกิดต่อเนื่องด้วยความรวดเร็วมาก หากผู้ใช้มีความสนใจหรือคุ้นเคยจะยิ่ง

ตอบสนองกับเครื่องคอมพิวเตอร์ได้ดียิ่งขึ้น และสามารถเปรียบเทียบได้ว่า ซอฟต์แวร์เดียวกันสามารถตอบสนองได้ดีหรือไม่บนเครื่องฮาร์ดแวร์ที่แตกต่างกัน

ดังนั้น เนื้อหาทั้งหมดในบทนี้จึงมีวัตถุประสงค์เหล่านี้

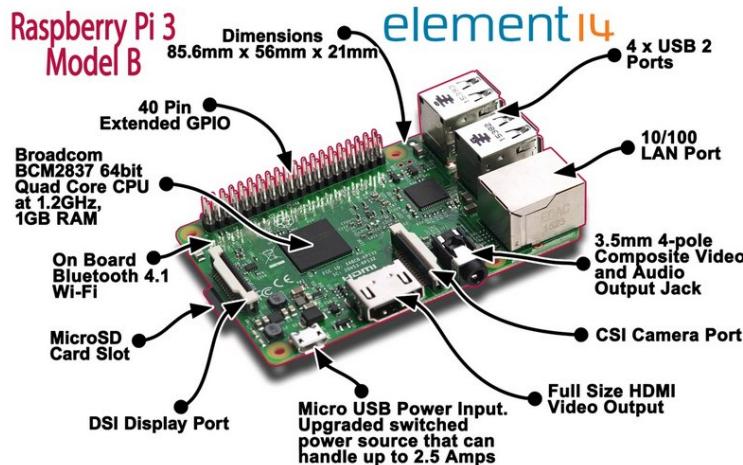
- เพื่อให้ผู้อ่านเข้าใจโครงสร้างโดยองค์รวมของคอมพิวเตอร์ กรณีศึกษาบอร์ด Raspberry Pi3/Pi4
- เพื่อให้ผู้อ่านเข้าใจ โครงสร้าง และ การทำงาน ของ ด้านฮาร์ดแวร์ กรณีศึกษา ARM Cortex A53 ภายในชิป BCM2837 บนบอร์ด Raspberry Pi3
- เพื่อให้ผู้อ่านเข้าใจ โครงสร้าง และ การทำงาน ของระบบปฏิบัติการลินุกซ์ (Linux) และ โครงสร้างของ ซอฟต์แวร์ประยุกต์
- เพื่อให้ผู้อ่านเข้าใจ การ ประสานงาน ระหว่าง ฮาร์ดแวร์ และ ซอฟต์แวร์ ขบวนการ รัน ซอฟต์แวร์ และ การสั่งงาน ให้ ฮาร์ดแวร์ ทำงาน
- เพื่อให้ผู้อ่านเข้าใจ ขบวนการ พัฒนา ซอฟต์แวร์ ด้วย ภาษา C/C++ และ ภาษา แอสเซมบลี (Assembly) ของ ชีพีью ARM

บอร์ด Raspberry Pi3 และ Pi4 เป็นคอมพิวเตอร์ขนาดเล็ก ที่ ได้รับ ความนิยม เนื่องจาก ราคาไม่สูง การออกแบบ ฮาร์ดแวร์ และ ซอฟต์แวร์ ที่ ลงตัว รวมไปถึง มี ซอฟต์แวร์ ประยุกต์ หรือ แอปพลิเคชัน (Application) ที่ หลากหลาย กรณีศึกษาบอร์ด Raspberry Pi ใน ตำแหน่งนี้ จะ หมายถึง บอร์ด Pi3/Pi4 ไม่เดล B ในรูปที่ 3.2 นอกจากไม่เดล B ซึ่งออกแบบมา เป็น คอมพิวเตอร์ ตั้งโต๊ะ อย่างง่าย ยังมีบอร์ด ไม่เดล A มีลักษณะคล้ายกัน แต่ ไม่เดล A จะเน้นการทำงาน แบบระบบฝังตัว (Embedded System) เป็นหลัก รายละเอียดเพิ่มเติมที่ wikidevi.com และ elinux.org

ด้วยเหตุนี้ บอร์ด Pi3/Pi4 ไม่เดล B สามารถประยุกต์ใช้ ได้ หลากหลาย เช่น เครื่องคอมพิวเตอร์ ส่วนตัว แท็บเล็ต โน้ตบุ๊ก ราคา ประหยัด อุปกรณ์ ผู้ร่วม ความปลอดภัย/สภาพแวดล้อม ในรูปแบบ IoT (Internet of Things) ระบบควบคุม ในบ้าน และ อุตสาหกรรม เชิร์ฟเวอร์ สำหรับ เครื่องพิมพ์ (Print server) อุปกรณ์ เชื่อมต่อ สัญญาณ อินเทอร์เน็ต ไร้สาย และ อื่น ๆ

3.1 ฮาร์ดแวร์ของเครื่องคอมพิวเตอร์: บอร์ด Pi3/Pi4

แผ่นวงจรพิมพ์ ของ บอร์ด Pi3/Pi4 ไม่เดล B ในรูปที่ 3.2 ประกอบด้วย ชิป BCM2837 ผลิตโดยบริษัท Broadcom Inc. ประเทศ สหรัฐอเมริกา นอกเหนือ จาก ชิป BCM2837 ซึ่ง ภายใน มี ชีพีью (CPU Central Processing Unit) บอร์ด Pi3/Pi4 มี หน่วย ความจำ หลัก SDRAM (Synchronous Dynamic Random Access Memory) ชนิด DDR2 (Double Data Rate 2) ขนาด ความจุ 1 กิกะไบต์ (GiB) (GiB) อุปกรณ์ กีบ รักษาข้อมูล (Data Storage) ชนิด การ์ด หน่วยความจำ SD (Secure Digital) ขนาด ความจุ 8-16 กิกะไบต์ (GB) เชื่อมต่อ อินเทอร์เน็ต ผ่าน ระบบ สื่อสาร ไร้สาย หรือ ผ่าน ช่อง เสียบ สาย เครือข่าย อีเธอร์เน็ต (Ethernet LAN) บอร์ด มี ช่อง เสียบ USB สำหรับ เชื่อมต่อ คีย์บอร์ด และ เม้าส์ เป็นต้น เพื่อ ให้ ระบบ สมบูรณ์ ผู้อ่าน สามารถ ทำ ความเข้าใจ ใน รายละเอียด ด้านต่าง ๆ ของ บอร์ด ได้ ใน ตาราง ที่ 3.1



รูปที่ 3.2: ตำแหน่งของอุปกรณ์ต่าง ๆ บนบอร์ด Pi3/Pi4 ที่มา: raspberryhome.net

ตารางที่ 3.1: ตารางสรุปข้อมูลด้านฮาร์ดแวร์และซอฟต์แวร์ของบอร์ด Pi3 โมเดล B และลิงก์เชื่อมไปยังหัวข้อที่แสดงรายละเอียดในบทต่างๆ

มอดูล	ภายในชิป (On chip) BCM2837	หัวข้อที่
BCM2837	เป็น SoC ผลิตโดยบริษัท Broadcom ประกอบด้วยชิปปัญญาณ 4 แกนประมวลผล ความถี่ 1.2 กิกะเฮิรตซ์	3.1.1
จอ LCD	สาย HDMI เวอร์ชัน 1.3 & 1.4 (ภาพและเสียง)	6.1
จอ LCD	สาย Display Serial Interface (DSI) 15 ขา ประกอบด้วยสัญญาณข้อมูล 2 คู่ สัญญาณคลื่อก 1 คู่	6.2
กล้องขนาดเล็ก	สาย Camera Serial Interface (CSI) 15-ขา ประกอบด้วยสัญญาณข้อมูล 2 ช่อง สัญญาณคลื่อก 1 ช่อง	6.3
จอทีวี และเสียง	สัญญาณคอมโพสิตวีดิโอ PAL/NTSC แจ็คขนาด 3.5 มม ชนิด 4 ขั้ว	6.5 6.4
GPIO	ขั้วต่อชนิด 2.54 มม 40 ขา ประกอบด้วย GPIO 27 ขา +3.3 และ +5V โวลต์	6.11
อุปกรณ์	ภายนอกชิป (Off chip) BCM2837	หัวข้อที่
ชิป SDRAM	หน่วยความจำชนิด DDR2 ความจุ 1 กิกะไบต์ (GiB) ชนิดประยัดพลังงาน	3.1.2 5.5
ชิป USB	ชิป USB 2.0 จำนวน 4 พอร์ต	6.6
ชิป Ethernet	เชื่อมต่ออินเทอร์เน็ตผ่านสาย ตัวอย่าง 10/100 Mbps	6.7
ชิป WiFi และ Bluetooth	เชื่อมต่ออินเทอร์เน็ตไร้สาย IEEE 802.11 b/g/n อัตราเร็วสูงสุด 150Mbps การเชื่อมต่อไร้สายเวอร์ชัน 4.1	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 และอื่น ๆ	
แหล่งจ่ายไฟ	ซ็อกเก็ตชนิด microUSB ขนาด 5 โวลต์ 2.5 แอมป์	6.14

องค์ประกอบสำคัญของบอร์ด คือ ชิป BCM2837/BCM2711 เป็น **SoC** ย่อมาจาก **System on Chip** ที่มีซีพียู ARM รุ่น Cortex A53/A72 จำนวน 4 แกนประมวลผล (Quad-core) รองรับภาษาแอสเซมบลีของซีพียู ARM เวอร์ชัน v8 หรือย่อว่า **ARMv8** ซึ่งมีความสามารถคำสั่งภาษาเครื่องความยาว 32 และ 64 บิต ซีพียู ARM Cortex A53/A72 นี้ใช้สัญญาณคล็อก (Clock) ความถี่ 1.20 กิกะ赫تز (GHz) ซึ่งแตกต่างจากกรณีของบอร์ด Pi2 ที่ใช้ชิป BCM2836 ซึ่งมีซีพียู ARM Cortex A7 รองรับภาษาแอสเซมบลี ARMv7 ชนิด 32 บิตเท่านั้น ทำงานที่ความถี่เพียง 900 เมกะเฮิรตซ์ แม้ว่าชิป BCM2836 จะมีจำนวน 4 แกนประมวลผล เช่นกัน แต่บอร์ด Pi2 มีหน่วยความจำเพียง 512 เมบิไบต์ (MiB) เท่านั้น ล่าสุดมูลนิธิได้ประกาศรายละเอียดของบอร์ด Pi4 แล้ว

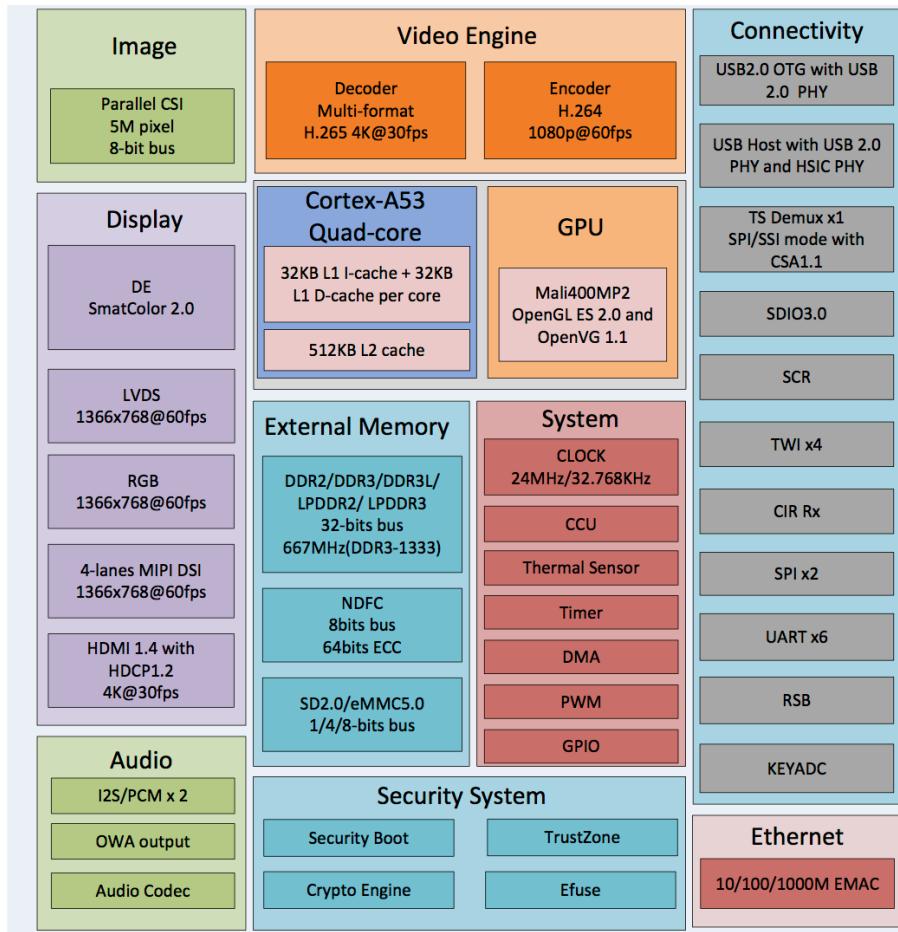
รูปที่ 3.2 แสดง การเชื่อมโยง อุปกรณ์ต่าง ๆ บนบอร์ด Pi3/Pi4 โดยมีชิป BCM2837 บอร์ด Pi3/Pi4 ไมเดล B เป็นคอมพิวเตอร์บอร์ดเดียว (Single Board Computer) ราคาเพื่อการศึกษา เป็นรุ่นถัดจาก Raspberry Pi2 (Pi2) ไมเดล B ออกแบบและพัฒนาโดยองค์กรที่มีชื่อว่า Raspberry Pi Foundation โดยทั้งบอร์ด Pi2 และ Pi3/Pi4 มีขนาดทางกายภาพเท่ากัน (86mm.x56mm.x21mm.) ทำให้ใช้กล่อง (Case) ด้วยกันได้ ตำแหน่งของพอร์ตและคอนเนกเตอร์คล้ายกัน แต่บอร์ด Pi3/Pi4 มีความสามารถในการประมวลผลที่สูงขึ้นและมีประสิทธิภาพดีกว่า ตัวอย่างการเปรียบเทียบบอร์ดรุ่นต่าง ๆ จัดทำโดยเว็บไซต์ medium.com

ดังนั้น การทดลองที่ 2 ภาคผนวก B เป็นการติดตั้งและใช้งานฮาร์ดแวร์ จะขอให้ผู้อ่านได้ประกอบบอร์ด Pi3/Pi4 เข้ากับจอภาพ LCD ขนาดใหญ่ และอุปกรณ์อินพุต/เอาต์พุตอื่น ๆ เพื่อใช้เป็นคอมพิวเตอร์ตั้งโต๊ะส่วนบุคคลประกอบการทดลองทั้งหมด

3.1.1 ซีพียู (CPU: ARM Cortex A53/A72)

ชิป BCM2837 ผลิตโดยบริษัท Broadcom ประกอบด้วย

- **ซีพียู (CPU)** ย่อมาจากคำว่า Central Processing Unit ARM Cortex A53 ออกแบบโดยบริษัท ARM จำนวน 4 แกนประมวลผลโดยแต่ละแกนประมวลผลคือ วงจรดิจิทัลทำงานตามสัญญาณคล็อกความถี่ 1.2 กิกะ赫ertz รองรับคำสั่งภาษาแอสเซมบลี (Assembly) ของซีพียู ARM เวอร์ชันต่าง ๆ ถึงเวอร์ชัน 8A รายละเอียดเพิ่มเติมอยู่ในหัวข้อที่ ??
- **จีพียู (GPU: Graphic Processing Unit)** ชื่อ VideoCore เป็นหน่วยประมวลผลด้านการแสดงผลบนจอภาพ ในชิป BCM2837 บรรจุจีพียูจำนวน 2 แกนประมวลผล ทำงานด้วยสัญญาณคล็อกความถี่ 400 เมกะเฮิรตซ์ ออกแบบโดยบริษัท Broadcom หน้าที่เป็น Multimedia Co-Processor รองรับ OpenGL ES เวอร์ชัน 2.0 และ OpenVG
- **มอดูลจัดการวีดีโอ (Video Engine)** สามารถจัดการหัสภาพวีดีโอที่ความละเอียด 1080x1920 อัตราเฟรชภาพ 30 เฟรมต่อวินาที มาตรฐาน ITU H.264 ความละเอียด High-Profile
- **วงจรเชื่อมต่อหน่วยความจำหลัก SDRAM ชนิด DDR2 (SDRAM Interface)**



รูปที่ 3.3: บล็อกไดอะแกรมของชิป AllWinner A64 ที่มีชิปปี้ ARM Cortex A53 คล้ายกับชิป BCM2837 บนบอร์ด Raspberry Pi3 ที่มา: [Allwinner Technology, Co. \(2015\)](#)

- วงจร เชื่อมต่อ การ์ด หน่วยความจำ SD เวอร์ชัน 2.0 (External Mass Media Controller) ที่มา: [Broadcom Corp. \(2012\)](#)
- มอดูลเชื่อมต่อ อินพุต และ เอาต์พุต ต่าง ๆ ตามรายละเอียดในตารางที่ 3.1

เนื่องจากข้อจำกัดด้านรูปภาพประกอบของชิป BCM2837/BCM2711 ผู้เขียนจึงขอใช้ข้อมูลจากชิปที่มีคุณสมบัติใกล้เคียงแทนด้วยบล็อกไดอะแกรมของชิป A64 จากบริษัท AllWinner ตามรูปที่ 3.3 เนื่องจาก มีชิปปี้ Cortex A53/A72 คล้ายกับชิป BCM2837/BCM2711 แต่มอดูลสำคัญ ๆ ต่างกัน เช่น จีพีสู, Ethernet เป็นต้น ที่มา: [Allwinner Technology, Co. \(2015\)](#)

3.1.2 หน่วยความจำหลัก (Main Memory)

หน่วยความจำหลักของบอร์ดเป็นชิปหน่วยความจำไดนามิกแรมแบบซิงโครนัส หรือ SDRAM (Synchronous Dynamic RAM) ชนิด DDR2 (Double Data Rate 2) เหมาะสำหรับใช้งานบนอุปกรณ์เคลื่อนที่ (Mobile) เพราะเป็นรุ่นประหยัด พลังงาน หรือชนิด LP (Low Power) และจำนวนขาห้อยเพื่อประหยัดพื้นที่บนบอร์ด ชิปหน่วยความจำหลักนี้รองรับสัญญาณความถี่คลื่อกสูงสุด 400 เมกะเฮิรตซ์ ทำให้เกิด



รูปที่ 3.4: หน่วยความจำ SDRAM ด้านล่างของบอร์ด Pi3/Pi4 โมเดล B ที่มา: robo-dyne.com

ความเร็วสูงสุด 800 เมกะ บิต/วินาที เนื่องด้วยหน่วยความจำหลักนี้ทำงานตามจังหวะความถี่สัญญาณคลือกที่สูง และตัวถังขนาดเล็กมีพื้นผิวสัมผัสกับอากาศได้น้อย ความร้อนที่ตัวถังจะเป็นอุปสรรคต่อการทำงาน ดังนั้น ผู้อ่านควรติดตั้งฮีทซิงก์ (Heat Sink) เพื่อเพิ่มพื้นที่ผิวสัมผัสกับอากาศในการระบายความร้อนออกจากชิปนี้ ตามการทดลองที่ 2 ภาคผนวก B การติดตั้งและใช้งานฮาร์ดแวร์

ชิปหน่วยความจำหลัก SDRAM นี้ผลิตโดยบริษัท Elpida รุ่น B8132B4PB-8D-F วางอยู่ด้านล่างของบอร์ด Pi3/Pi4 โมเดล B ดังรูปที่ 3.4 ขาดัญญาณต่าง ๆ จึงชื่อนอยู่ใต้ชิป เพื่อลดขนาดฟุตพรินท์ (Foot Print) บนแผ่นวงจรพิมพ์ ปัจจุบันนี้ บริษัท Elpida ได้เปลี่ยนผู้ถือหุ้นหลักเป็น บริษัท Micron Technology แล้ว ดังนั้น หน่วยความจำ SDRAM หมายเลขรุ่น EDB8132B4PB จึงสามารถค้นคว้าเพิ่มเติมได้ตามลิงก์ต่อไปนี้ www.micron.com ตำราเล่มนี้จะอธิบาย การทำงานและรายละเอียดเพิ่มเติมของหน่วยความจำ SDRAM นี้และชนิดต่าง ๆ ในบทที่ 5 ในหัวข้อที่ 5.5.1

3.1.3 อุปกรณ์อินพุต/เอาต์พุต (Input/Output Devices)

หัวข้อนี้จะกล่าวเฉพาะอุปกรณ์อินพุตและเอาต์พุตที่จำเป็น ได้แก่ คีย์บอร์ด เม้าส์ จอมอนิเตอร์ เพื่อใช้งานบอร์ด Pi3/Pi4 เป็นเครื่องคอมพิวเตอร์ตั้งโต๊ะส่วนตัว (Desktop Personal Computer) และทำการทดลองต่าง ๆ ตามตำราเล่มนี้ ส่วนอุปกรณ์อินพุตและเอาต์พุตอื่น ๆ ในบอร์ด Pi3/Pi4 ผู้อ่านสามารถอ่านเพิ่มเติมในบทที่ 6 ได้อย่างละเอียด

คีย์บอร์ด (Keyboard)

คีย์บอร์ดชนิด 106 ปุ่มนี้ เป็นคีย์บอร์ดที่ออกแบบโดยบริษัท IBM และพัฒนาต่อเนื่องมา มีลักษณะคล้ายกับปุ่มพิมพ์บนเครื่องพิมพ์ดีด โดยปกติจะประกอบด้วย

- ปุ่มตัวอักษร ประกอบด้วยอักษรสำหรับการป้อนข้อมูลที่มีทั้งตัวอักษร A-Z และ a-z ตัวเลข 0-9 และอักษรพิเศษ เช่น @, #, \$, % เป็นต้น

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)		
30	31	32	33	34	35	36	37	38	39	40	41	42	43		
44	45	46	47	48	49	50	51	52	53	54	55	56	57		
58		60	131	61	132	133	62		64						

106-Key Keyboard Position Codes

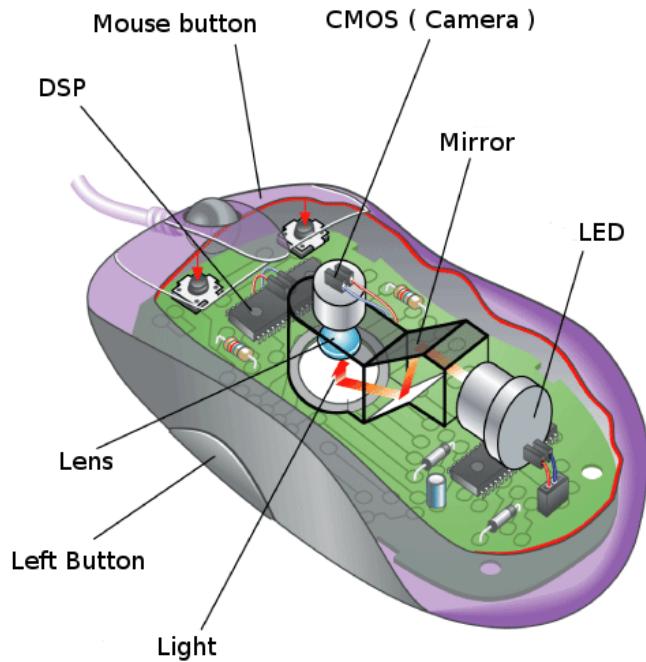
รูปที่ 3.5: ตำแหน่ง และรหัสประจำปุ่ม (Scan Code หรือ Position Code) บนคีย์บอร์ดชนิด 106 ปุ่ม
ที่มา: ps-2.kev009.com

- ปุ่มป้อนข้อมูลตัวเลข โดยจะมีสแกนโค้ดเท่ากับ 90-109 สำหรับการป้อนข้อมูลที่เป็นตัวเลขเรียงตัวกันทางขวาสุดของคีย์บอร์ด เพื่อความสะดวกต่อการใช้งานสำหรับนักบัญชี หรือผู้ที่กรอกข้อมูลบอยๆ
- 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) นิยมใช้เคลื่อนหรือเปลี่ยนตำแหน่งของเมาเซอร์ (Cursor) บนหน้าจอแสดงผลทั้งในโหมดกราฟิกและตัวอักษร

เมื่อผู้ใช้กดปุ่มตั้งแต่ 1 ปุ่มขึ้นไป วงจรดิจิทัลภายในตัวคีย์บอร์ดจะส่งรหัสสแกนโค้ด (Scan Code หรือ Position Code) ของปุ่มที่โดนกดตามที่ได้ออกแบบไว้ในรูปที่ 3.5 ผ่านสายไปยังพอร์ต USB เป็นหลัก หรืออาจจะเป็นคีย์บอร์ดไร้สาย เช่น ชนิดคลื่นวิทยุความถี่ต่ำ ชนิดคลื่นบลูทูธ เป็นต้น ไปยังระบบปฏิบัติ ตามหลักการเดียวกับรูปที่ 3.1 เพื่อให้โปรแกรมตอบสนองต่อรหัสนั้น ๆ เมื่อผู้ใช้กดปุ่มใด ๆ ก็ครั้งก์ตามระบบปฏิบัติ การจะรับรู้ได้ผ่านกลไกการเกิดอินเตอร์รัปท์ (Interrupt) ซึ่งมีรายละเอียดเพิ่มเติมในการทดลองที่ 11 ภาคผนวก K

เมาส์ (Mouse)

เมาส์ คือ อุปกรณ์ที่สามารถแปลงการเคลื่อนที่ของแขนผู้ใช้ในแกน 2 มิติเป็นการเคลื่อนที่ของพอยน์เตอร์ (Pointer) การเคลื่อนที่ของเมาส์จะสัมพัทธ์กับตำแหน่งปัจจุบันของพอยน์เตอร์บนจอแสดงผล ทั้งนี้ขึ้นอยู่กับความละเอียดของเมาส์เอง ความละเอียดของหน้าจอแสดงผล และแอปพลิเคชันนั้น ๆ เช่น เกมส์โปรแกรมวิดีโอและตัวต่อตัวจะต้องความละเอียดของเมาส์สูงกว่าเมาส์ปกติ เป็นต้น



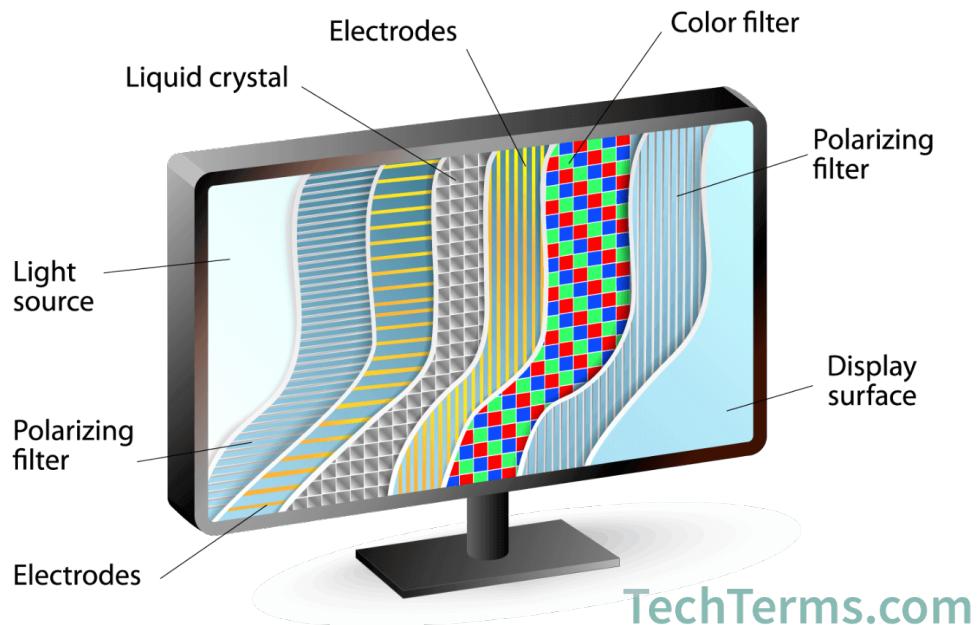
รูปที่ 3.6: โครงสร้างภายในเมาส์ชนิด Optical ประกอบด้วยหลอด LED ส่องแสงกระแทบกับพื้นผิวด้านล่างแล้วสะท้อนกลับมายังตัวรับแสงชนิด CMOS ที่มา: www.pinterest.com

เมาส์ส่วนใหญ่มีจำนวนปุ่มคลิกทางซ้ายและขวา ขึ้นอยู่กับการออกแบบของระบบปฏิบัติการ มี 1-3 ปุ่ม สำหรับการใช้งานปั๊จจุบัน มีการเพิ่มวงล้อ (Wheel) บนเมาส์สำหรับการเคลื่อนที่แบบพิเศษในมิติต่าง ๆ การคลิกปุ่มซ้าย (Left Click) ปุ่มขวา (Right Click) การคลิกปุ่มซ้ำ (Double Click) และ การสัมผัสขึ้นหรือลงบนปุ่มเมาส์ จะทำให้เกิดแอคชั่นต่าง ๆ ตามระบบปฏิบัติการ และโปรแกรมที่ใช้งาน ดังนั้น ผู้ใช้จะต้องใช้ความรู้ ความเข้าใจ ตลอดจนถึงความจำ เพื่อให้สามารถใช้งานโปรแกรมเดียวกันแต่ทำงานบนระบบปฏิบัติการที่ต่างกันได้

รูปที่ 3.6 แสดง โครงสร้างภายในเมาส์ชนิดอปติกัล (Optical Mouse) ประกอบด้วยหลอดแอลอีดี (LED) ส่องแสงกระแทบกับพื้นผิวด้านล่างก่อน แล้วจึงสะท้อนกลับมายังตัวรับแสงชนิด CMOS แสงจากหลอด LED มีลักษณะที่แตกต่างกันไปตามชนิดและราคาของเมาส์ เช่น หลอด LED เป็นสีแดงหรือสีเขียว ผู้ใช้มองเห็น และหลอด LED แสงเลเซอร์ (Laser) ซึ่งมองไม่เห็นด้วยตาเปล่าแต่มีความละเอียดเพิ่มสูงขึ้นมาก เพื่อประยัดการใช้แรงและเพิ่มความแม่นยำในการเคลื่อนที่ของเมาส์ ซึ่งแสงเลเซอร์จะให้ความละเอียดในการใช้งานสูงกว่า เหมาะกับซอฟต์แวร์ด้านกราฟิก และเกมส์ ด้วยความละเอียด 800-6,000 จุดต่อระยะทางหนึ่งนิ้ว (DPI: Dot Per Inch)

เมื่อผู้ใช้กดปุ่มใด ๆ หรือขยับเมาส์ ระบบปฏิบัติการจะรับรู้ได้ผ่านกลไกการเกิดอินเทอร์รัปท์ (Interrupt) เช่นเดียวกับคีย์บอร์ด ซึ่งมีรายละเอียดเพิ่มเติมในบทที่ 6 และการทดลองต่าง ๆ นอกเหนือจากเมาส์ อุปกรณ์อื่น ๆ ที่ใกล้เคียงสามารถทดแทนการใช้เมาส์ได้ เช่น ทัชแพด (Touch Pad) จօแสดงผลระบบสัมผัส เสียงพูด ท่าทาง (Gesture) ของมือและนิ้วที่ตรวจจับโดยกล้องวีดีโอ เป็นต้น ทำให้การใช้งานคอมพิวเตอร์ง่ายลง และได้ประสบการณ์เป็นธรรมชาติมากขึ้น สาขาด้าน Human Computer Interface จึงมีความสำคัญเพิ่มมากขึ้นเรื่อย ๆ

จอภาพ LCD (Liquid Crystal Display)



TechTerms.com

รูปที่ 3.7: โครงสร้างจอแสดงผลชนิด Color LCD ประกอบด้วยแหล่งกำเนิดแสง (Light Source) ปล่อยแสงท่าลุ้นต่าง ๆ ปรากฏแก่สายตาผู้ใช้ ที่มา: techterms.com

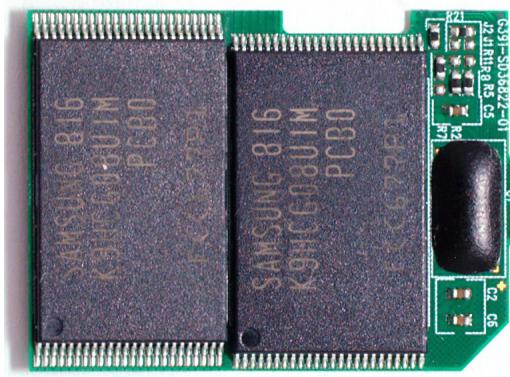
ในอดีต จอภาพของคอมพิวเตอร์มีขนาดใหญ่เทอะทะ น้ำหนักมาก สามารถแสดงผลได้เฉพาะตัวอักษรเท่านั้น และพัฒนาเรื่อยมาจนกลายเป็นจอภาพ LCD (Liquid Crystal Display) ทำให้มีขนาดบางลง น้ำหนักจึงเบาลง จอ LCD สามารถแสดงผลตัวอักษรและกราฟิกทั้งในรูปของ ภาพนิ่ง ภาพเคลื่อนไหว และเกมส์ เพื่อโต้ตอบ และสร้างปฏิสัมพันธ์กับผู้ใช้งานได้อย่างมีประสิทธิภาพมากยิ่งขึ้น การแสดงผลของเครื่องคอมพิวเตอร์จะมีพิษทางเป็นสามมิติและเสมือนจริงมากขึ้น เมื่อความก้าวหน้าด้านต่าง ๆ ถึงพร้อม

โครงสร้างจอภาพ LCD สี (Color LCD) ประกอบด้วยหลอดกำเนิดแสงชนิด LED ด้านหลัง (LED Back Light) ทำหน้าที่ปล่อยแสงท่าลุ้นต่าง ๆ มาแสดงผล และปรากฏแก่สายตาผู้ใช้ตามรูปที่ 3.7 สีที่เกิดขึ้นบนจอเกิดจากการผสมกันของจุดภาพย่อย (Sub Pixel) 3 จุด ๆ 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 ความสว่าง และการจัดเรียงตัวของหลอด LED Back Light การเรียงตัวของจุดภาพย่อย (RGB: Red Green Blue) ในแต่ละพิกเซล ชนิดของพื้นผิวน้ำจอ และองศาของมุมมองภาพ (Angle) ระยะเวลาในการตอบสนองต่อการแสดงผลแต่ละภาพ (Response Time) เพื่อตอบสนองต่อเนื้อหาที่เปลี่ยนแปลงอย่างรวดเร็ว เช่น การเล่นเกมส์ และนิเมชั่นความละเอียดสูง การเข้ามาระหว่างจอภาพกับเครื่องคอมพิวเตอร์ จึงต้องใช้สัญญาณชนิด HDMI รายละเอียดเพิ่มเติมในหัวข้อที่ 6.1

หลักการพื้นฐานของแต่ละจุดภาพย่อย หรือ สับพิกเซล (Subpixel) 1 จุดบนจอแสดงผล LCD ประกอบด้วย ข้าไฟฟ้าชนิดโปร่งแสง (Transparent Electrode) สองข้า เพื่อสร้างสนามไฟฟ้า จึงทำให้เกิดการเรียงตัวของผลึกคริสตัลในของเหลว (Liquid Crystal) หรือการปิดตัวพากลืนแสงจากเลนส์โพลาไรซ์ (Polar-

ize) ด้านหลัง สามารถเดินทางผ่านเลนส์โลลาไรซ์ด้านหน้า ทำให้ผู้ใช้มองเห็นความสว่างของจุดนั้น ในทางกลับกันเมื่อไม่มีประจุไฟฟ้า ผลึกจะไม่บิดเกลียวทำให้แสงทะลุผ่านแกนโลลาไลซ์ที่ตั้งฉากไม่ได้ 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) สำหรับใช้งานกับอุปกรณ์เคลื่อนที่ต่าง ๆ 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 โดยราคาในท้องตลาด จะแบ่งตามความจุ ความจุของการ์ดหน่วยความจำ SD มีแนวโน้มเพิ่มขึ้นตามวัตถุประสงค์การใช้งาน เนื่องจากตัวการ์ดหน่วยความจำมีขนาดเล็ก น้ำหนักเบา ความเร็วในการอ่านหรือเขียนที่รวดเร็วทันใจ และอายุการใช้งานยาวนาน

ในด้านโครงสร้างทางกายภาพ การ์ดหน่วยความจำ SD มี 3 ขนาด คือ ขนาดปกติ ขนาดมินิ (Mini) และขนาดไมโคร (Micro) โดยมีตัวแบ่งการ์ดขนาดมินิและขนาดไมโครให้เป็นขนาดปกติได้ องค์ประกอบภายในของการ์ดหน่วยความจำ SD ในรูปที่ 3.8 คือ ชิปหน่วยความจำแฟลช (Flash Memory) ผลิตโดยบริษัท Samsung Electronics ประเภทเกาหลีใต้ ตำราเล่มนี้จะกล่าวอธิบายรายละเอียดของหน่วยความจำแฟลชเพิ่มเติมในหัวข้อที่ 7.2

บอร์ด Pi3/Pi4 โมเดล B ใช้อุปกรณ์เก็บรักษาข้อมูลชนิดการ์ดหน่วยความจำไมโคร SD สำหรับบรรจุระบบปฏิบัติการ บันทึกและเก็บรักษาข้อมูลต่าง ๆ โดยผู้ใช้สามารถติดตั้งระบบปฏิบัติการ เช่น Raspberry Pi OS, Ubuntu Linux, Microsoft Windows 10 IoT เป็นต้น ลงบนการ์ดหน่วยความจำไมโคร SD นี้ ซึ่งผู้อ่านจะได้ฝึกติดตั้งระบบปฏิบัติการ Raspberry Pi OS ในการทดลองที่ 3 ภาคผนวก C

3.2 การพัฒนาซอฟต์แวร์หรือโปรแกรมคอมพิวเตอร์

ซอฟต์แวร์ คือ ไฟล์ซึ่ง ประกอบด้วย ชุด คำสั่ง ภาษาเครื่อง และ ข้อมูล ประกอบ ต่าง ๆ ซอฟต์แวร์ ทำ หน้าที่ ส่งงาน ซีพียู และ ฮาร์ดแวร์ อื่น ๆ ให้ ทำงาน ตาม ที่ โปรแกรม เมอร์ เขียน (Code) หรือ พัฒนา (Develop) ซอฟต์แวร์ มี โครงสร้าง ที่ ระบบปฏิบัติการ แต่ละ ระบบ กำหนด ไว้ เป็น มาตรฐาน เช่น รูปแบบไฟล์ ELF สำหรับ ระบบปฏิบัติการ ตระกูล ยูนิกซ์ รูปแบบไฟล์ EXE สำหรับ ระบบปฏิบัติการ Microsoft Windows เป็นต้น ซอฟต์แวร์ คอมพิวเตอร์ ทั่วไป แบ่ง เป็น 2 ชนิด หลัก ได้ แก่

- ซอฟต์แวร์ระบบ (System software) เป็น ซอฟต์แวร์ พื้นฐาน ที่ เครื่อง คอมพิวเตอร์ ต้อง มี ประกอบ ด้วย ระบบปฏิบัติการ (Operating System) ทำ หน้าที่ ดำเนิน การ ด้าน อินพุต และ เอาต์พุต บริหาร จัด การ หน่วย ความ จำ และ อุปกรณ์ กีบ รักษา ข้อมูล จัด ลำดับ การ ทำงาน และ การ ใช้งาน ทรัพยากร ของ โปรแกรม เพื่อ ให้ ซอฟต์แวร์ ประยุกต์ ทำงาน ได้อย่าง ถูกต้อง ปลอดภัย และ มี ประสิทธิภาพ
- ซอฟต์แวร์ประยุกต์ (Application software) เป็น ซอฟต์แวร์ ที่ ผู้ใช้ นำมา ประยุกต์ ในการ ทำงาน โดย ผู้ พัฒนา ซอฟต์แวร์ (Software Developer) พัฒนา หรือ เขียน ขึ้น ด้วย ภาษา ระดับ สูง (High-Level Programming Language) เช่น ภาษา Python C/C++ Java เป็นต้น

การ พัฒนา ซอฟต์แวร์ ทำ ได้ หลาย ระดับ ขึ้น อยู่ กับ สิ่ง แวดล้อม และ ความ ต้องการ ของ ซอฟต์แวร์ การ เขียน โปรแกรม ด้วย ภาษา สคริปต์ (Script Language)

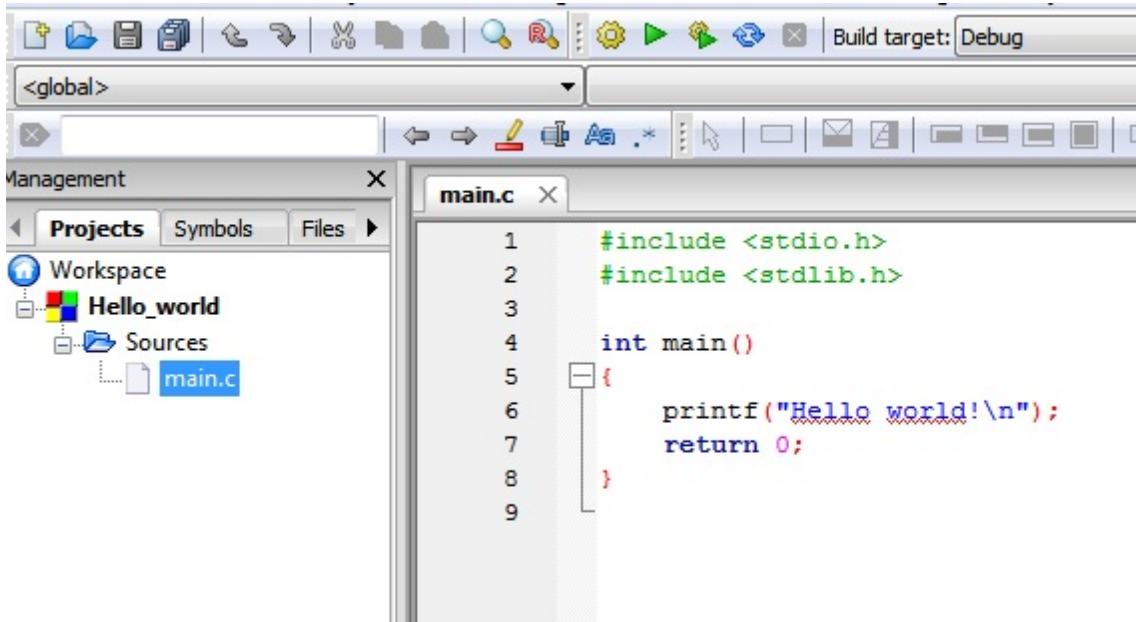
- โดย ใช้ การ ตีความ (Interpretor-based) เช่น ภาษา ไพร่อน (Python) ภาษา HTML (HyperText Markup Language) เป็นต้น ด้วย อินเทอร์เพตเตอร์ (Interpreter)
- โดย ใช้ การ ประมวล (Compiler-based) เช่น ภาษา C/C++ ภาษา จา瓦 (Java) เป็นต้น ด้วย คอมไพล์ เเลอร์ (Compiler)

การ พัฒนา ซอฟต์แวร์ ด้วย ภาษา ไพร่อน (Python) บน บอร์ด Pi3/Pi4 ได้ รับ ความ นิยม เนื่อง จาก ตัว ภาษา มี ความ ซับ ซ้อน น้อย กว่า ทำ ให้ เรียนรู้ เอง ได้ ง่าย มี ตัวอย่าง โปรแกรม ที่ นัก พัฒนา ทั่วโลก เปิด เผยแพร่ ซอฟต์แวร์ โค้ด ผ่าน ทาง เครือข่าย อินเทอร์เน็ต ผู้ อ่าน สามารถ ค้น คิว่า เพิ่ม เติม ได้ ที่ python.org

ตำรา เล่ม นี้ จะ เน้น การ พัฒนา ซอฟต์แวร์ โดย ใช้ คอมไพล์ เแล้ว ภาษา ระดับ สูง C/C++ บน ระบบ ปฏิบัติการ ตระกูล ยูนิกซ์ เป็น กรณี ศึกษา ตาม หลักสูตร วิศวกรรม คอมพิวเตอร์ เนื่อง จาก ผลลัพธ์ ที่ ได้ จาก การ คอมไพล์ (Compile) ภาษา C/C++ คือ คำสั่ง ภาษา แอสเซมบลี และ คำสั่ง ภาษา เครื่อง (Machine Code) ใน หัวข้อ ที่ 3.2.5 ซึ่ง จะ บรรจุ ออยู่ ใน ไฟล์ รูปแบบ ELF ตาม ที่ กล่าว ไป แล้ว ใน หัวข้อ ที่ 3.3.2

3.2.1 โครงสร้าง ของ ซอฟต์แวร์ โค้ด โปรแกรม ภาษา C/C++

การ เขียน โปรแกรม ด้วย ภาษา ระดับ สูง (High-level language) เช่น ภาษา C/C++ เพื่อ แก้ โจทย์ หรือ ปัญหา ที่ ต้อง การ ความ รวดเร็ว โปรแกรม เมอร์ สามารถ เขียน โปรแกรม และ สร้าง สรุค ผลงาน ได้ ง่าย กว่า ภาษา ระดับ ล่าง เช่น ภาษา แอสเซมบลี และ ภาษา เครื่อง เนื่อง จาก ภาษา ระดับ สูง มี ลักษณะ ใกล้ เคียง กับ



รูปที่ 3.9: ตัวอย่างชอร์สโค้ดภาษา C ฟังก์ชันหลักชื่อ main() เมื่อทำงานเสร็จสิ้น จะรีเทิร์นค่า 0 ที่มา: zentut.com

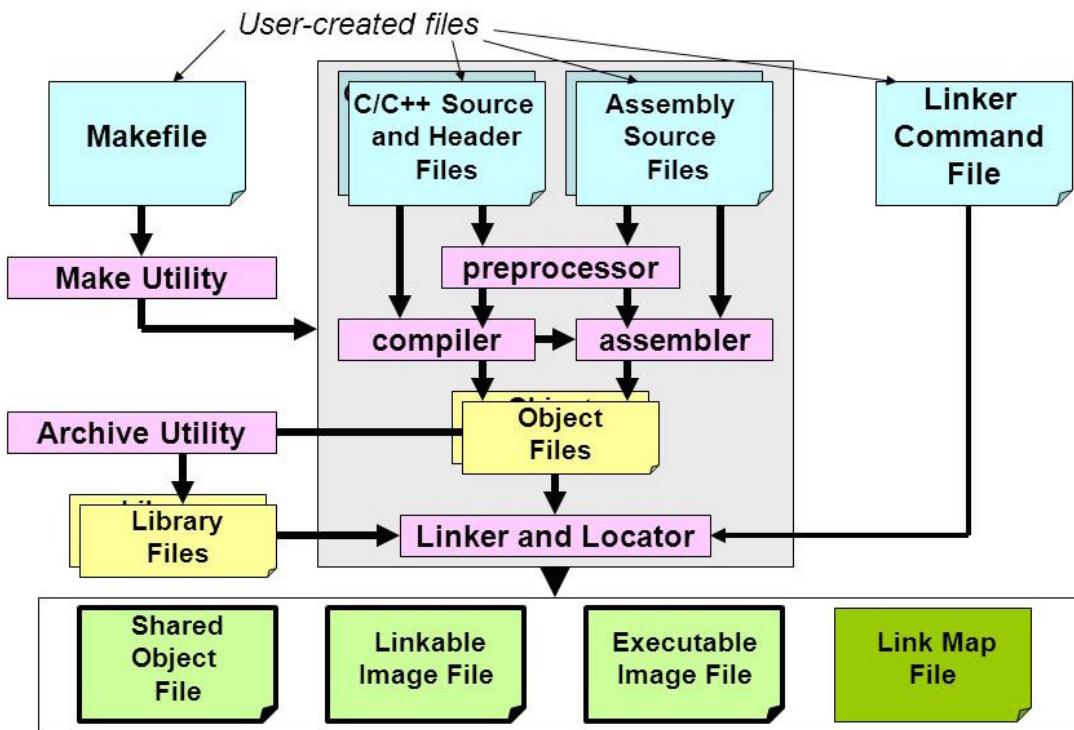
ประโยชน์ภาษาอังกฤษ ภาษาธรรมดับสูง จะไม่ยึดติดกับเครื่องหรือฮาร์ดแวร์ (Machine Independent) ในทางตรงกันข้าม ภาษาแอสเซมบลีและภาษาเครื่องจะผูกติดกับฮาร์ดแวร์หรือ ชีพียุของเครื่องคอมพิวเตอร์ (Machine Dependent) หากโปรแกรมเมอร์สามารถใช้คำสั่งภาษาแอสเซมบลีในการสั่งงานฮาร์ดแวร์ของเครื่องคอมพิวเตอร์ จะยิ่งช่วยให้โปรแกรมนั้น ๆ ทำงานได้เต็มประสิทธิภาพ ตัวรานี้จะใช้ภาษา C และภาษาแอสเซมบลีของชีพียุ ARM เป็นหลัก เพื่อเข้าถึงระบบปฏิบัติการ ภาษาเครื่อง และฮาร์ดแวร์ของชีพียุ ARM ได้ลึกซึ้งมากกว่าภาษาประเภทตีความ เช่น Python

ตัวอย่างชอร์สโค้ดภาษา C ในรูปที่ 3.9 ชื่อไฟล์ main.c ฟังก์ชันหลักชื่อ main() เป็นฟังก์ชันที่โปรแกรมเริ่มต้นทำงาน เมื่อทำงานเสร็จสิ้นโปรแกรมจะรีเทิร์นค่า 0 ซึ่งเป็นเลขจำนวนเต็มค่าหนึ่ง ซึ่งมักใช้บ่งบอกว่าเป็นการทำงานเสร็จสิ้นโดยไม่มีปัญหาหรือข้อผิดพลาด และหากโปรแกรมรีเทิร์นค่าอื่น ๆ ที่ไม่ใช่ 0 จะบ่งบอกหัสความผิดพลาดได้

3.2.2 การแปลซอร์สโค้ดภาษา C/C++ ให้กลายเป็นโปรแกรมคอมพิวเตอร์

ภาษาคอมพิวเตอร์ระดับสูง (High-level language) มีศักยภาพสูงที่จะพัฒนาเป็นซอฟต์แวร์ หรือโปรแกรมหรือแอปพลิเคชันได้สะดวก โปรแกรมเมอร์สามารถเขียนโปรแกรมเป็นไฟล์ภาษาต่าง ๆ เรียกโดยรวมว่า ชอร์สโค้ด (Source Code) รูปที่ 3.10 แสดงขั้นตอนการพัฒนาซอฟต์แวร์เริ่มจากชอร์สโค้ดภาษาคอมพิวเตอร์ และแปลให้เป็นไฟล์ Executable หรือโปรแกรมในระบบปฏิบัติการ ยูนิกซ์ กล่องสีชมพูในรูป คือ โปรแกรมระบบที่จำเป็นประกอบด้วย คอมไพล์러 แอสเซมเบลอร์ (Assembler) และลิงก์เกอร์ (Linker) เป็นต้น

คอมไпал์러เป็นโปรแกรมคอมพิวเตอร์ชนิดหนึ่ง ทำหน้าที่แปลซอร์สโค้ดภาษาสูง เช่น ภาษา C/C++ ให้กลายเป็นคำสั่งภาษาแอสเซมบลีและคำสั่งภาษาเครื่องในที่สุด แล้วจึงจัดเรียงคำสั่งภาษาเครื่องให้มี



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

โครงสร้างตามไฟล์รูปแบบไฟล์ ELF ซึ่งอธิบายแล้วในหัวข้อที่ 3.3.2 คือไฟเลอร์หลักในระบบปฏิบัติการ Unix/Linux คือ GCC ผู้อ่านสามารถค้นควารายละเอียดเพิ่มเติมได้ที่ gnu.org

คอมไฟเลอร์ทุกตัวต้องทำความเข้าใจตัวไฟล์ชอร์สโค้ดต่าง ๆ ที่โปรแกรมเมอร์พัฒนาไว้ (กล่องสีฟ้า) คอมไฟเลอร์จะอ่านไฟล์ชอร์สโค้ดเพื่อสแกนตัวอักษรที่เป็นเนื้อโปรแกรม จัดแยกโทเค็น (Token) ตรวจสอบความถูกต้อง แปลความหมายของโทเค็น สร้างแผนภูมิต้นไม้ไวยากรณ์ (Abstract Syntax Tree) เพื่อตรวจสอบไวยากรณ์และ การวิเคราะห์ความหมาย (Semantic Analysis) ของประโยคต่าง ๆ จนปราศจากข้อผิดพลาด คอมไฟเลอร์จะดำเนินการปรับปรุงการทำงานของโปรแกรมที่จะคอมไพล์ให้ดีขึ้น เรียกว่า การอปติไมซ์ (Optimize) โดยการสร้างกราฟการไหลของข้อมูล (Data-Flow graph) เพื่อประกอบการทำอปติไมเซชัน (Optimization) ทำให้โปรแกรมทำงานได้รวดเร็ว ยิ่งขึ้น หลังจากนั้น คอมไฟเลอร์จะสร้างคำสั่งภาษาเครื่อง (Machine Code) ตามชนิดของชิปปี้ที่ต้องการ ซึ่งตำราเล่มนี้จะใช้ชิปปี้ของชิปปี้ ARM เป็นกรณีศึกษา ผู้อ่านสามารถศึกษารายละเอียดของคอมไฟเลอร์เพิ่มเติมได้ที่ wikipedia โปรแกรมเมอร์จะกำหนดความสัมพันธ์หรือความเชื่อมโยงระหว่างชอร์สโค้ดเหล่านั้นใน Makefile เพื่อให้คอมไฟเลอร์สามารถสร้าง (Generate) ไฟล์ผลลัพธ์ (กล่องสีเขียว) ตามที่โปรแกรมเมอร์ต้องการ คือ ผู้อ่านควรทำการทดลองที่ 5 ภาคผนวก E เพื่อทดลองสร้างและใช้ Makefile เพื่อคอมไพล์และลิงค์โปรแกรมด้วยภาษา C และภาษาแอลกอริทึมชิปปี้ ARM ในลำดับถัดไป

- ไฟล์โปรแกรม (Executable Image File) ในรูปที่ 3.10 คือไฟล์ที่ได้จากการลิงค์หรือเชื่อมกับไฟล์อ้อมเจก์และไฟล์ไลบรารี ซึ่งคำสั่งภาษาเครื่องและข้อมูลภายในไฟล์เหล่านี้จะถูกจัดเรียงตามรูปแบบไฟล์ ELF ในหัวข้อที่ 3.3.2 ขั้นการลิงค์จะกล่าวโดยละเอียดในหัวข้อที่ 3.2.4 ถัดไป ไฟล์

โปรแกรมส่วนใหญ่จะมีชื่อแตกไปมีนามสกุล หมายเหตุ หากผู้ใช้ไม่กำหนดชื่ออื่น คอมไพล์ จะตั้งชื่อไฟล์ให้อัตโนมัติว่า **a.out**

- **ไฟล์อ็อบเจกต์** คือ ไฟล์อ็อบเจกต์ (กล่องสีเหลือง) ที่ได้จากการคอมไพล์ซอร์สโค้ดต่าง ๆ และระบบปฏิบัติการอนุญาตให้นำไปลิงก์ และโหลดในระหว่างที่รันโปรแกรมได้ ไฟล์อ็อบเจกต์ส่วนใหญ่จะมีชื่อตามด้วย .o รายละเอียดเพิ่มเติมที่ yolinux.com
- **ไฟล์ไลบรารี (Linkable Library File)** คือ ไฟล์ที่ได้จากขบวนการคอมไпал์โดยนักพัฒนาต่างๆ เพื่อนำมาลิงก์กับไฟล์หลักให้สมบูรณ์ ซึ่งไลบรารีหลักในการพัฒนาโปรแกรมภาษา C ชื่อว่า glibc ไฟล์ไลบรารีส่วนใหญ่จะมีชื่อตามด้วย .lib หรือ .a ผู้อ่านสามารถค้นคว้ารายละเอียดเพิ่มเติมได้ที่ gnu.org
- **ไฟล์แผนผังการลิงก์ (Link Map File)** เพื่ออธิบายการเชื่อมโยงกันของตัวแปรต่าง ๆ ในซอร์สโค้ด ไฟล์ชนิดนี้ส่วนใหญ่จะมีชื่อตามด้วย .map

ไฟล์ทั้งหมดนี้จะบรรจุคำสั่งภาษาเครื่องในรูปของเลขฐานสองหรือย่อให้อยู่ในรูปของเลขฐานสิบหก และข้อมูลต่าง ๆ ตามที่อธิบายในบทที่ 2 ในรูปของเลขฐานสองหรือย่อให้อยู่ในรูปของเลขฐานสิบหก เช่น กัน ตั้งนั้น เชิญเมนต์ต่าง ๆ ของไฟล์รูปแบบ ELF จะเป็นสิ่งกำกับว่า เลขฐานสองหรือเลขฐานสิบหกแท็ลล์ค่าในเชิญเมนต์เหล่านั้น คือ คำสั่ง หรือ ข้อมูล ยกตัวอย่างเช่น

- เลขฐานสองหรือเลขฐานสิบหกที่จัดเรียงอยู่ในเทกซ์เชิญเมนต์ คือ คำสั่งภาษาเครื่อง เท่านั้น
- เลขฐานสองหรือเลขฐานสิบหกที่จัดเรียงอยู่ในดาต้าเชิญเมนต์ คือ ข้อมูล เท่านั้นซึ่งอาจจะมีค่าเป็นเลขจำนวนเต็ม ถ้าตำแหน่งตรงกับตัวแปรชนิด จำนวนเต็ม หรือมีค่าเป็นเลขทศนิยมชนิด จุด ลอยตัว ตามมาตรฐาน IEEE754 ในบทที่ 2 ทั้งนี้ขึ้นอยู่กับการประ觥ชนิดตัวแปร รายละเอียดเพิ่มเติมในหัวข้อถัดไป

ผู้อ่านสามารถเสริมสร้างทักษะการพัฒนาโปรแกรมและความเข้าใจจาก การทดลองที่ 5 ภาคผนวก E และการทดลองอื่น ๆ หัวข้อต่อไปนี้จะอธิบายโครงสร้างของซอร์สโค้ดภาษาแอสเซมบลีของซีพียู ARM ซึ่งมีลักษณะใกล้เคียงกับภาษาเครื่องแต่โปรแกรมเมอร์ยังสามารถทำความเข้าใจได้ ก่อนที่ผู้อ่านจะได้รู้จักกับคำสั่งภาษาเครื่องในรูปของเลขฐานสองในหัวข้อที่ 3.2.5

3.2.3 โครงสร้างของซอร์สโค้ดภาษาแอสเซมบลีของซีพียู ARM

ภาษา แอ ส เซมบลี (Assembly language) เป็นคำสั่งภาษาคอมพิวเตอร์ที่อยู่ในรูปของคำย่อ (Mnemonic) จากภาษาอังกฤษ ทำให้โปรแกรมเมอร์เข้าใจง่ายกว่าตัวเลขฐานสองที่เป็นคำสั่งภาษาเครื่อง (Machine Code) ในตัวรานี้จะใช้ภาษาแอสเซมบลีของซีพียู ARM Cortex A เวอร์ชัน 32 บิตเป็นหลัก โดยมีรายละเอียดเพิ่มเติมในบทที่ 4

ตารางที่ 3.2 แสดงให้เห็นถึง โครงสร้างหลักของซอร์สโค้ดภาษาแอสเซมบลีของซีพียู ARM แบ่งเป็น .text หมายถึง เทกซ์เชิญเมนต์ และ .data คือ ดาต้าเชิญเมนต์ ตามลำดับ ส่วนคำสั่งภาษาแอสเซมบลี

(Assembly language) ในบรรทัดอื่น ๆ อยู่ในรูปของคำภาษาอังกฤษที่โปรแกรมเมอร์สามารถทำความเข้าใจได้ และมีความใกล้เคียงกับคำสั่งภาษาเครื่อง ประกอบด้วย พังก์ชันชื่อ **main** แบ่งเป็น 3 คอลัมน์ ดังนี้

- คอลัมน์ซ้าย ใช้กำหนดชื่อเลเบล (Label) ต่าง ๆ หมายถึง คำภาษาอังกฤษที่ตามด้วยสัญลักษณ์ :
- คอลัมน์กลาง ใช้กำหนดคำสั่ง (Operation) ว่าต้องการจะให้ชีพิญทำอะไร
- คอลัมน์ขวา ใช้ระบุรีจิสเตอร์ หรือ แอดเดรส หรือ เลเบล หรือ ค่าคงที่สำหรับคำสั่งในคอลัมน์ตรงกลาง

ตารางที่ 3.2: ตัวอย่างโปรแกรมภาษาแอสเซมบลีเพื่อประกาศและตั้งค่าเริ่มต้นให้ตัวแปรขนาด 32 บิตจำนวน 4 ตัวแปร และวนรอบบวกค่าโดยใช้ตัวแปรพอยน์เตอร์

บรรทัดที่	เลเบล	คำสั่ง	รีจิสเตอร์ หรือ แอดเดรส เลเบล 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

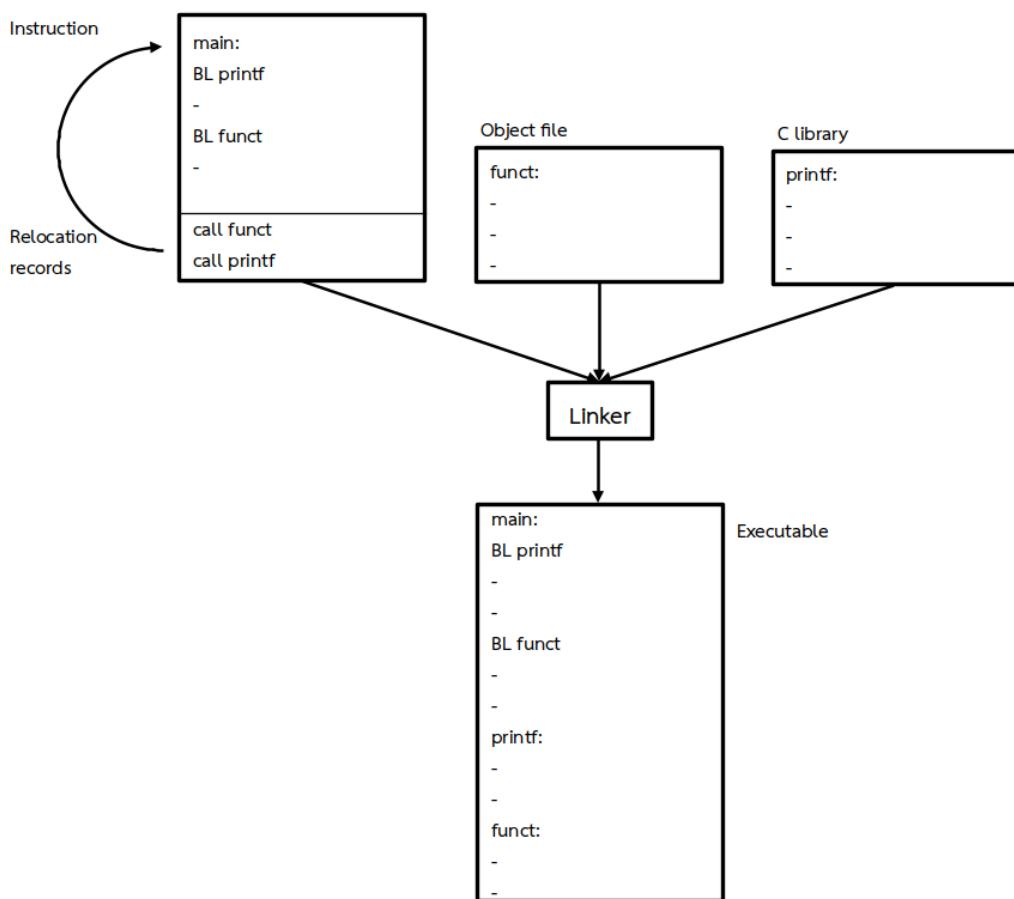
ตัวอย่างโปรแกรมในตารางที่ 3.2 ผู้เขียนสามารถอธิบายตามหมายเลขอรรถได้ดังนี้

- .text หมายถึง ตำแหน่งเริ่มต้นของเทกซ์เช็กเมนต์ ตามที่กล่าวในหัวข้อ 3.3.2
- .global main หมายถึง พังก์ชันชื่อ **main** ซึ่งจะตรงกับพังก์ชัน **main()** ในภาษา C/C++
- 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 โดยคำสั่งนี้จะกำหนดให้มีเลbel ชื่อ LOOP ซึ่งต้องการจะให้เกิดการทำซ้ำ
9. คำสั่ง ADD หมายถึง การบวกค่าภายในรีจิสเตอร์ โดย $R0=R0+R3$ แล้วเก็บผลรวมไว้ในรีจิสเตอร์ R0
10. คำสั่ง SUBS ย่อมาจาก **Subtract** และอัปเดตผลลัพธ์ในรีจิสเตอร์สถานะ (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 ก่อนหน้าโดย
 - หาก R1 มีค่ามากกว่า 0 แล้ว การทำงานจะกลับไปเริ่มที่คำสั่งที่ตรงกับเลbel LOOP ดังนั้นโปรแกรมนี้จะวนรอบ $M=4$ ครั้ง เพื่อบวกค่าของ 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 บอกรหัสสินสุดของชอร์สโค้ด เมื่อโปรแกรมเสร็จสิ้นการทำงานจะรีเทิร์นกลับไปหาฟังก์ชันที่เป็นผู้เรียกฟังก์ชัน (Caller Function) นั้น
16. .data หมายถึง ตำแหน่งเริ่มต้นของดาต้าเซ็กเมนต์
17. เลbel **SUM** ทำหน้าที่เป็นชื่อของตัวแปรชนิดจำนวนเต็ม (Integer) ขนาด 32 บิตชนิด 2's Complement และถูกกำหนดค่าเริ่มต้นให้มีค่าเป็น 0

18. เลbel M ทำหน้าที่เป็นชื่อของตัวแปรชนิดจำนวนเต็ม (Integer) ขนาด 32 บิตชนิด 2's Complement ถูกกำหนดค่าเริ่มต้นให้มีค่าเป็น 4 หมายถึง จำนวนสมาชิกของตัวแปรอาร์เรย์ชื่อ NUM
19. เลbel NUM ทำหน้าที่เป็นชื่อของตัวแปรชนิดอาร์เรย์ของเลขจำนวนเต็มที่จะถูกกำหนดค่าเริ่มต้นให้มีค่าเป็น 3, 5, 7, 9 ตามลำดับ โดย $\text{NUM}[0] = 3$, $\text{NUM}[1] = 5$, $\text{NUM}[2] = 7$ และ $\text{NUM}[3] = 9$
20. เลbel PTR ทำหน้าที่เป็นชื่อของตัวแปรที่ถูกกำหนดค่าเริ่มต้นเป็นแอดเดรสของ NUM ในการพัฒนาโปรแกรมภาษา C/C++ เรียกตัวแปรชนิดนี้ว่า พอยน์เตอร์ (Pointer) โดยตัวแปรชนิดพอยน์เตอร์จะเก็บแอดเดรสด้วยพื้นที่ขนาด 1 Word หรือเท่ากับ 4 ไบต์ เป็นตัวเลขจำนวนเต็มชนิด Unsigned ตามชนิดของระบบปฏิบัติการ ซึ่งในตัวเรียนนี้เป็นเวอร์ชัน 32 บิต

3.2.4 การรวมไฟล์อ็อบเจกต์ (Object) ด้วยลิงก์เกอร์ (Linker)



รูปที่ 3.11: ตัวอย่างการลิงก์ (Link) หรือรวมไฟล์อ็อบเจกต์หลัก ไฟล์อ็อบเจกต์เสริม และไฟล์ไลบรารีเข้าด้วยกันเป็นไฟล์โปรแกรมหรือแอปพลิเคชัน ที่มา: google.com

ลิงก์เกอร์ (Linker) เป็นหนึ่งในซอฟต์แวร์ระบบที่มีความสำคัญต่อขั้นตอนการพัฒนาโปรแกรม รูปที่ 3.10 แสดงการทำงานของลิงก์เกอร์เพื่อรวมไฟล์อ็อบเจกต์หลัก ไฟล์อ็อบเจกต์เสริม และไฟล์ไลบรารีของ

ภาษา C เข้าด้วยกัน ซึ่งได้แก่ ไลบรารี glibc และไลบรารี wiringPi ในการทดลองที่ 10 ภาคผนวก J และการทดลองที่ 11 ภาคผนวก K

ไฟล์อ็อบเจกต์ (*.o) คือ ไฟล์ที่รวมฟังก์ชันที่นักพัฒนาเก็บไว้ส่วนตัว ซึ่งตอนเอองมีชอร์สโค้ด แล้วแต่ไม่จำเป็นต้องคอมpileใหม่อีกรอบ ทำให้ประหยัดเวลาสำหรับการคอมpile ไฟล์ ไลบรารี คือ ไฟล์อ็อบเจกต์ที่รวมฟังก์ชันสำเร็จรูปที่ถูกคอมpile และพัฒนาอย่างต่อเนื่องจนน่าเชื่อถือ และเปิดให้นักพัฒนาอื่น ๆ ดาวน์โหลดไปใช้งานผ่านทางคอมไฟล์หรือหลัก ซึ่ง GCC เป็นต้น ไฟล์ทั้งหมดนี้อยู่ในรูปแบบ ELF ซึ่งมีรายละเอียดในหัวข้อที่ 3.3.2 รูปที่ 3.11 แสดงตัวอย่างการลิงก์ (Link) หรือรวมไฟล์อ็อบเจกต์หลัก ไฟล์อ็อบเจกต์เสริม และไฟล์ไลบรารีเข้าด้วยกันให้เป็นไฟล์โปรแกรมหรือแอปพลิเคชันที่สมบูรณ์ หมายเหตุ กล่องสีเหลืองในรูป คือ ไฟล์ที่บรรจุคำสั่งภาษาเครื่องและข้อมูลเป็นเลขฐานสอง แต่แสดงเป็นคำสั่งภาษาแอสเซมบลีแทนเพื่อให้ผู้อ่านเข้าใจได้ง่าย คำอธิบายเพิ่มเติมในหัวข้อที่ 4.8 เรื่องการเรียกใช้ฟังก์ชัน และการทดลองที่ 7 ภาคผนวก G

กล่าวโดยสรุป การลิงก์เป็นการขยายจีดความสามารถของโปรแกรมที่พัฒนาใหม่จากไฟล์อ็อบเจกต์ และไฟล์ไลบรารีที่มีอยู่เดิม โปรแกรมเมอร์สามารถเขียนโปรแกรมตามที่ต้องการโดยไม่จำเป็นต้องพัฒนาฟังก์ชันอื่น ๆ ทั้งหมดด้วยตนเอง แต่สามารถเรียกใช้งานฟังก์ชัน (Function Call) ที่มีผู้พัฒนาไว้แล้ว ตรงตามหลักการที่สำคัญพัฒนาซอฟต์แวร์ที่ดี เรียกว่า โมดูลาริตี้ (Modularity) รายละเอียดสามารถศึกษาเพิ่มเติมด้วยภาษา C จากการทดลองที่ 5 ภาคผนวก E และภาษาแอสเซมบลีในการทดลองที่ 6 ภาคผนวก F เป็นต้นไป

3.2.5 ตัวอย่างของคำสั่งภาษาเครื่อง (Machine Code) ของชีพีย์ ARM

ผู้อ่านได้ทำความสะอาดเข้าใจกระบวนการพัฒนาซอฟต์แวร์และทำการทดลองที่ 6 ในภาคผนวก F แล้ว หัวข้อนี้จะเสริมความเข้าใจเรื่องของคำสั่งภาษาแอสเซมบลีและภาษาเครื่องเข้าด้วยกัน โดยจะใช้ภาษาเครื่องของชีพีย์ ARM เป็นกรณีศึกษาเพื่อให้สอดคล้องกับบอร์ด Pi3/Pi4 รูปที่ 3.12 แสดงตัวอย่างการแปลงจากคำสั่งแอสเซมบลี (ตัวอักษร) ทางด้านขวา ให้เป็นคำสั่งภาษาเครื่อง (ตัวเลขฐานสอง) ของชีพีย์ ARM เวอร์ชันความยาว 32 บิตทางด้านซ้าย ซึ่งมีรูปแบบที่ชัดเจน โดย ARM เป็นผู้ออกแบบและกำหนดรายละเอียดเพื่อให้ผู้พัฒนาฮาร์ดแวร์และซอฟต์แวร์ต่าง ๆ ทำตาม

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

รูปที่ 3.12: ตัวอย่างการแปลงจากคำสั่งแอสเซมบลีของชีพีย์ 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 $E94E75001_{16}$ ผู้อ่านสามารถทำความเข้าใจบิตต่าง ๆ 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 นั่นคือ ไม่มีการตั้ง (Set) ค่า CPSR (Current Program Status Register) รายละเอียดเพิ่มเติมในหัวข้อที่ [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 หมายถึง จำนวนบิตที่ต้องการเลื่อนเท่ากับ 0 บิตนั้นคือ ไม่มีการเลื่อนบิตของรีจิสเตอร์ **Rm**
- บิตที่ **5-6** ความยาว 2 บิต คือ ตำแหน่งของ Sh (Shift Type) ของคำสั่งนี้มีค่าเท่ากับ 00_2 หมายความว่าเป็น การเลื่อนบิตทางซ้ายแบบตรรกะ (Logical Shift Left) 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 $R9 = \text{Mem}[R4+16_{10}]$ จะถูกแปลงเป็นคำสั่งภาษาเครื่องความยาว 32 บิตเท่ากับ $1110\ 01\ 011001\ 0100\ 1001\ 0000\ 0001\ 0000_2$ หรือ $E5949010_{16}$ ผู้อ่านสามารถทำความเข้าใจบิตต่าง ๆ ตามลำดับจากซ้ายไปขวา ดังนี้

- บิตที่ **28-31** ความยาว 4 บิต คือ ตำแหน่งของ cond (Condition) ซึ่ง 1110_2 หมายถึง คำสั่ง LDR นี้ทำงานโดยไม่มีเงื่อนไข (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
- 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) ซึ่งจะตีความหรือถอดรหัส (Decode) ได้ว่า คำสั่งที่อ่านเข้ามาเป็นคำสั่งอะไร ต้องการใช้รีจิสเตอร์ตัวไหน กำหนดค่าคงที่ (Immediate) เป็นเลขฐานสิบ也好 เป็นบวกหรือลบ และจึงสั่งงานควบคุมภายในซึ่งนำไปปฏิบัติ (Execute) ตาม รายละเอียดในการออกแบบวงจรหรือฮาร์ดแวร์ได้ในวิชาสถาปัตยกรรมคอมพิวเตอร์ ([Computer Architecture](#)) ส่วนการออกแบบพัฒนาวงจรนั้นสามารถทำได้โดยใช้ภาษา Hardware Description Language (HDL) ซึ่งมีลักษณะคล้ายกับภาษาโปรแกรมคอมพิวเตอร์ระดับสูง C/C++ และ Java ซึ่งได้รับความนิยม 2 ภาษาคือ

- ภาษา VHDL ([Very High Speed Integrated Circuit HDL](#)) และ
- ภาษา Verilog HDL ([Verilog Hardware Description Language](#)) เพื่อให้โปรแกรมลักษณะคล้ายคอมไฟเลอร์แต่เรียกว่า Synthesis Tool สั่งเคราะห์เป็นวงจรบนอุปกรณ์
- FPGA ([Field Programmable Gate Array](#)) โดย เป็นวงจรชนิดหนึ่ง ซึ่งผู้ใช้สามารถเปลี่ยนหรือโปรแกรม (Program) ให้ทำงานเป็นวงจรได้หลาย ๆ รอบ เพื่อทดสอบการทำงานจนถูกต้อง

3.3 การทำงานร่วมกันระหว่างฮาร์ดแวร์และซอฟต์แวร์ (Hardware - Software Operation)

การทำงานของคอมพิวเตอร์จะเริ่มต้นขึ้นเมื่อผู้ใช้กดปุ่มเปิดเครื่องเพื่อจ่ายไฟเลี้ยงให้กับคอมพิวเตอร์ระบบฮาร์ดแวร์จะเริ่มทำงานเพื่อตรวจสอบและเตรียมความพร้อม หลังจากนั้น ขั้นตอนการทำงานของซอฟต์แวร์ระบบและซอฟต์แวร์ประยุกต์สามารถสรุปได้ดังนี้

1. การบูต (Boot) ระบบปฏิบัติการจากอุปกรณ์เก็บรักษาข้อมูลเข้าสู่หน่วยความจำหลัก ในหัวข้อที่ [3.3.1](#)
2. ระบบปฏิบัติการโหลดไฟล์ซอฟต์แวร์แอปพลิเคชันจากอุปกรณ์เก็บรักษาข้อมูลเข้าสู่หน่วยความจำหลัก ในหัวข้อที่ [3.3.2](#)
3. ซีพียูเฟชคำสั่งภาษาเครื่องจากหน่วยความจำหลักไปปฏิบัติตาม ในหัวข้อที่ [3.3.3](#)
4. ซีพียูอ่าน/เขียนข้อมูลระหว่างหน่วยความจำหลักไปประมวลผล ในหัวข้อที่ [3.3.4](#)
5. ซีพียูเข้มต่ออุปกรณ์อินพุตต่างๆ เช่น คีย์บอร์ด เม้าส์ เครือข่าย อินเทอร์เน็ต เป็นต้น ในหัวข้อที่ [3.3.5](#)
6. ซีพียูอ่าน/เขียนไฟล์ข้อมูลระหว่างหน่วยความจำหลักและอุปกรณ์เก็บรักษาข้อมูล ในหัวข้อที่ [3.3.6](#)
7. ผู้ใช้ชัตดาวน์ (Shut Down) ระบบปฏิบัติการก่อนปิดเครื่อง ในหัวข้อที่ [3.3.7](#)

ขั้นตอนดังกล่าวจะแตกต่างกันออกไปตามรายละเอียดของฮาร์ดแวร์และระบบปฏิบัติการที่ใช้ แต่หลักการโดยรวมจะมีความคล้ายคลึงกัน การทดลองที่ 3 ภาคผนวก C จะแนะนำการติดตั้งระบบปฏิบัติการซึ่งว่า Raspberry Pi OS การทดลองที่ 4 ภาคผนวก D จะแนะนำการใช้งานฟังก์ชันพื้นฐานในระบบลินุกซ์และยูนิกซ์ (Unix) เป็นต้น ซึ่งยูนิกซ์เป็นระบบปฏิบัติการที่มีประวัติศาสตร์ยาวนานและเป็นต้นแบบของระบบปฏิบัติการต่างๆ ในวิทยาการคอมพิวเตอร์ 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 หรือจ่ายไฟให้บอร์ด Pi3/Pi4 ซีพียู ARM Cortex A53/A72 จะยังไม่ทำงานแต่จะใช้แกนประมวลผลขนาดเล็กภายในจีพียูอ่านคำสั่งที่บรรจุใน On-Chip รวมเรียกว่า บูตโลเดอร์ (Boot Loader) ซึ่งฝังอยู่ภายในชิป BCM2837/BCM2711

2. จีพียูสั่งงานฮาร์ดแวร์ที่ควบคุมอุปกรณ์เก็บรักษาข้อมูลการ์ดหน่วยความจำชนิดไมโคร SD เพื่อมองหาพาร์ทิชันซึ่งboot ที่ฟอร์แมทด้วยรูปแบบ FAT32 ภายในการ์ดนั้น

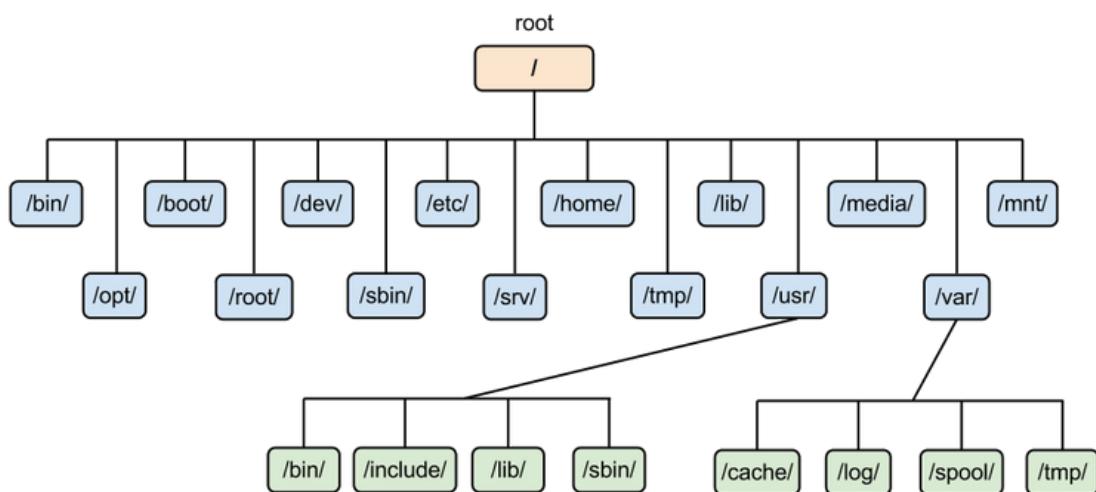
3. เมื่อเจอพาร์ทิชันซึ่งboot แล้วจีพียูอ่านชุดคำสั่งจากไฟล์ซึ่งbootcode.bin ซึ่งทำหน้าที่เป็นบูตโหลดเดอร์ โดยในระหว่างนี้จะยังไม่มีการใช้งานหน่วยความจำหลัก SDRAM บนบอร์ด Pi3/Pi4

4. จีพียูเริ่มต้นอ่านไฟล์ start.elf จากพาร์ทิชัน boot ใน การ์ด หน่วยความจำ SD ไปบรรจุในหน่วยความจำหลัก เพื่อให้ซีพียูเริ่มทำงาน โดยอาศัยไฟล์ fixup.dat ซึ่งมีข้อมูลการจัดแบ่งพาร์ทิชันภายในระหว่างจีพียูและซีพียู

5. ซีพียูเริ่มต้นทำงานตามคำสั่งภาษาเครื่องหน่วยความจำหลัก (SDRAM) ที่โหลดจากไฟล์ start.elf ก่อนเพื่ออ่านไฟล์ซึ่งkernel.img ไปบรรจุใน SDRAM ตามรายละเอียดในไฟล์ config.txt เพื่อติดตั้งค่าของระบบ

6. ซีพียูเริ่มต้นเฟทธ์และปฏิบัติตามคำสั่งโปรแกรมเครอร์เนลใน SDRAM ด้วยพารามิเตอร์ที่บรรจุอยู่ใน cmdline.txt

ผู้อ่านสามารถค้นคว้ารายละเอียดเพิ่มเติม เช่น การบูตจากพอร์ต USB และจากเครือข่าย จากหน้าเว็บหลักของบอร์ด Raspberry Pi ต่อไปนี้ raspberrypi.com เมื่อบอร์ด บูตระบบสำเร็จ เคอร์แนลหรือโปรแกรมหลักของระบบปฏิบัติการ จะครอบครองฮาร์ดแวร์ทั้งหมด และทำงานร่วมกับไฟล์ซอฟต์แวร์และไฟล์ข้อมูลต่าง ๆ ภายในการ์ด หน่วยความจำ SD ซึ่งจะแบ่งโครงสร้างการจัดเก็บไฟล์ต่าง ๆ ตามระบบปฏิบัติการลินุกซ์ เรียกว่า **ไดเรกทอรี** ในรูปที่ 3.13 ผู้อ่านสามารถทำความเข้าใจไดเรกทอรีตามลำดับความสำคัญ ดังนี้



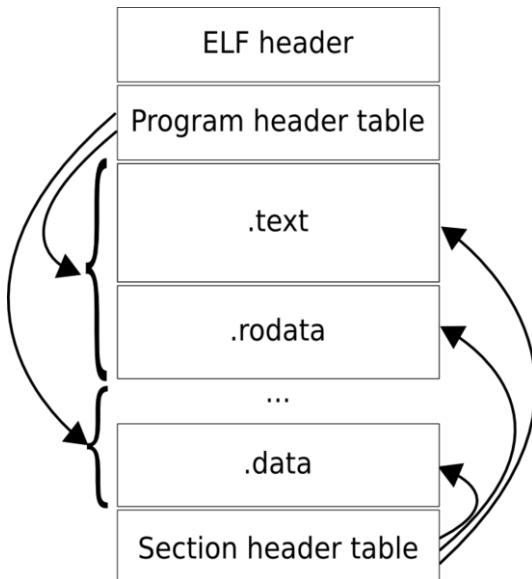
รูปที่ 3.13: โครงสร้างของไดเรกทอรี (Directory) สำคัญ ๆ ในระบบปฏิบัติการลินุกซ์ ที่มา: freedom-penguin.com

1. /root: เป็นไดเรกทอรีหลักของระบบ เรียกว่า รากไดเรกทอรี

2. /bin: เป็นไดเรกทอรีบรรจุไฟล์ซอฟต์แวร์ระบบ เช่น คำสั่ง (Command) ต่าง ๆ สำหรับผู้ใช้งานใหญ่

3. /sbin: เป็นไดเรกทอรีบรรจุไฟล์คำสั่ง (Command) ต่าง ๆ ของระบบ เพื่อให้ผู้ใช้งาน ชื่อ **root** ทำหน้าที่บริหารระบบ (Administrator) เท่านั้น
4. /lib: เป็นไดเรกทอรีสำหรับเก็บไฟล์ไลบรารี (Library) ต่าง ๆ ของระบบ
5. /home: เป็นไดเรกทอรีสำหรับพื้นที่ส่วนตัวของผู้ใช้แต่ละคน สำหรับเก็บไฟล์ต่าง ๆ โดยใช้ชื่อไดเรกทอรีตามชื่อผู้ใช้ ภายในไดเรกทอรีผู้ใช้แต่ละคนแบ่งเป็นพื้นที่หรือไดเรกทอรีต่าง ๆ เช่น Documents Desktop Downloads เพื่อใช้เป็นที่จัดเก็บไดเรกทอรีและไฟล์ส่วนตัวของผู้ใช้แต่ละคน ซึ่งจะแยกตามชื่อ ล็อกอิน เช่น /home/user1 /home/user2 เป็นต้น โดย user1 และ user2 คือชื่อล็อกอิน ตามลำดับ
6. /dev: เป็นไดเรกทอรีที่ถูกสร้างใหม่ทุกครั้งที่บูตเครื่อง เพื่อเก็บไฟล์ที่เป็นตัวแทนของอุปกรณ์ (Devices) อินพุต/เอาต์พุตต่าง ๆ ในยูสเซอร์สเปช (User Space) รายละเอียดเพิ่มเติมสามารถอ่านได้จากคำสั่ง [ls /dev](#) ในการทดลองที่ 9 ภาคผนวก [I](#)
7. /sys: เป็นระบบไฟล์เสมือน (Virtual file system) เพื่อให้โปรแกรมมอร์เข้าถึงอุปกรณ์ชาร์ดแวร์ เช่น ขา GPIO ต่าง ๆ ซึ่งจะกล่าวในการทดลองที่ 9 ภาคผนวก [I](#) หัวข้อที่ [L.3](#)
8. /proc: เป็นเวอร์ชวลไดเรกทอรี สำหรับแต่ละprocเซสในระบบ ยกตัวอย่าง เช่น /proc การทดลองที่ 4 ภาคผนวก [D](#)
9. /etc: เป็นไดเรกทอรีใช้เก็บไฟล์สำหรับบรรจุไฟล์คอนฟิก (Configuration File) ของระบบ และแอปพลิเคชันต่าง ๆ
10. /var: เป็นไดเรกทอรีสำหรับเก็บล็อก (Log) และข้อมูลอื่น ๆ ที่เกิดขึ้นระหว่างการรันระบบ
11. /mnt: เป็นจุดมาท์ (Mount) ชั่วคราว เช่น การแชร์ไฟล์ผ่านเครือข่าย
12. /media: เป็นจุดมาท์หลักสำหรับสื่อเคลื่อนที่ (Removable device) ต่าง ๆ เช่น แฟลชไดร์ฟ ซีดี/ดีวีดีรอม เป็นต้น
13. /opt: เป็นไดเรกทอรีสำหรับติดตั้งแอปพลิเคชันจากผู้ขาย
14. /run: เป็นไดเรกทอรีชั่วคราวสำหรับแอปพลิเคชันในระหว่างที่รันอยู่
15. /tmp: เป็นไดเรกทอรีใช้สำหรับบรรจุไฟล์ชั่วคราวจากแอปพลิเคชันและระบบ

3.3.2 ระบบปฏิบัติการโหลดซอฟต์แวร์ประยุกต์จากไฟล์รูปแบบ ELF



รูปที่ 3.14: โครงสร้างไฟล์ชนิด ELF สำหรับเก็บชุดคำสั่งภาษาเครื่อง และข้อมูลเป็นเลขฐานสองอยู่ในอุปกรณ์เก็บรักษาข้อมูล ที่มา: wikipedia.org

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

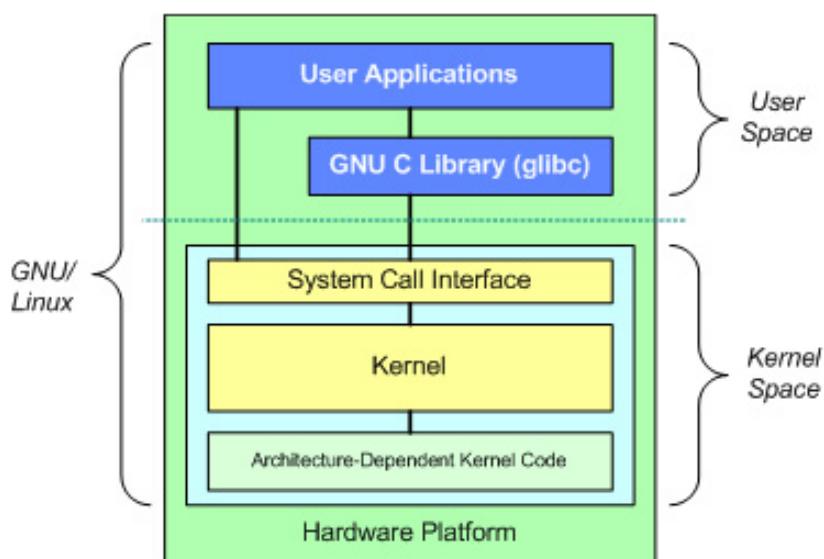
ผู้ใช้เครื่องคอมพิวเตอร์ได้รับสิทธิ์ (Permission) ติดตั้ง (Install) ไฟล์ซอฟต์แวร์เหล่านี้ลงในอุปกรณ์เก็บรักษาข้อมูลตามโครงสร้างของไดเรกทอรีต่าง ๆ ในหัวข้อที่ 3.3.1 โดยการทดลองที่ 4 ภาคผนวก D จะแนะนำการใช้งานคำสั่ง (Command) ต่าง ๆ และโครงสร้างของระบบปฏิบัติยูนิกซ์และลินุกซ์ ตำราเล่มนี้จะเน้นที่ระบบปฏิบัติการ Raspberry Pi OS เป็นลินุกซ์เวอร์ชันหนึ่งซึ่งพัฒนาสำหรับบอร์ดตระกูลนี้โดยเฉพาะ

รูปแบบไฟล์ ELF ย่อมาจาก Executable and Linkable Format เป็นชนิดและโครงสร้างของไฟล์โปรแกรม (Executable) และไฟล์อbjekต์ (Object) ที่ได้จากการคอมไพล์ ไฟล์รูปแบบ ELF มีความยืดหยุ่น สามารถขยายเพิ่มเติมได้ง่าย และรองรับการข้ามแพลทฟอร์ม (Cross Platform) ระหว่างซีพียูและชุดคำสั่ง (Instruction Set) หรือภาษาเครื่องเป็นเลขฐานสอง โครงสร้างของไฟล์ ELF ตามรูปที่ 3.14 ประกอบด้วยพื้นที่หลายส่วน เรียกว่า เช็คเมนต์ (Segment) ต่าง ๆ ดังนี้

- ส่วนเริ่มต้นไฟล์ (ELF Header) บ่งชี้ว่าจะอ้างถึงหน่วยความจำหลักด้วยเลขแออดเดรสขนาด 32 บิต หรือ 64 บิต ตามด้วยข้อมูลอื่น ๆ ตามลำดับถัดไป แบ่งเป็น
 - หากใช้ระบบ 32 บิต ส่วนหัวนี้จะมีความยาว 52 ไบต์
 - หากใช้ระบบ 64 บิต ส่วนหัวนี้จะมีความยาว 64 ไบต์

- ตารางโปรแกรม (Program Header Table) อธิบายเซกเมนต์ (Segment) ต่าง ๆ ในไฟล์ ELF ว่าจะสร้างเซกเมนต์เหล่านั้นอย่างไร ที่ตำแหน่งใดในหน่วยความจำ
- เทกซ์เซกเมนต์ (.text) ใช้สำหรับเก็บคำสั่ง (เป็นเลขฐานสอง)
- ดาต้าเซกเมนต์ (.rodata: Read-Only Data) ใช้สำหรับเก็บค่าของตัวแปรหรือข้อมูลที่เป็นค่าคงที่ (เป็นเลขฐานสอง)
- ดาต้าเซกเมนต์ (.data และ .data1) ใช้สำหรับเก็บตัวแปรที่มีค่าตั้งต้น (Initialized Data) และอาจเปลี่ยนแปลงได้เมื่อทำงาน
- ตาราง Section Header Table อธิบายเซกชัน (Section) ต่าง ๆ ในหน่วยความจำ ว่าซื้ออะไร มีความหมายเท่าไร

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



รูปที่ 3.15: โครงสร้างของคอมพิวเตอร์ประกอบด้วยชาร์ดแวร์ชั้นล่างสุด และหน่วยความจำเวอร์ชวล เมมโมรี (Virtual memory) ภายใต้ระบบลินุกซ์แบ่งเป็นพื้นที่ เรียกว่า เคอร์เนลสเปช (ระบบปฏิบัติการ) และ ยูสเซอร์สเปช (ซอฟต์แวร์ประยุกต์) ที่มา: linux-india.org

เมื่อเครื่องคอมพิวเตอร์พร้อมใช้งานหรือบูตระบบปฏิบัติการลินุกซ์สำเร็จแล้ว ผู้ใช้จึงสามารถเรียกใช้แอปพลิเคชันซอฟต์แวร์หรือโปรแกรมตามสิทธิ์ (Permission) ด้วยการกดตัวเบิลคลิกที่ไอคอน (Icon) หรือเรียกชื่อไฟล์แอปพลิเคชันนั้นผ่านเทอร์มินัล (Terminal) ในการทดลองที่ 4 ภาคผนวก D ระบบปฏิบัติการจึงอ่านหรือโหลดไฟล์แอปพลิเคชันนั้น ๆ จากอุปกรณ์เก็บรักษาข้อมูลไปบรรจุในหน่วยความจำหลัก เราเรียกชั้นตอนนี้ว่า การโหลดโปรแกรม (Program Loading) ก็คือเป็นโครงสร้างที่สมบูรณ์ทั้งหมดในรูปที่ 3.15 ประกอบด้วย ชาร์ดแวร์ และ หน่วยความจำเวอร์ชวลเมมโมรี (Virtual memory) รายละเอียดในหัวข้อ

ที่ 5.2 โดยหน่วยความจำเวอร์ชัล เมโมรีของหนึ่งโปรแกรมในระบบปฏิบัติการลินุกซ์แบ่งเป็นพื้นที่ 2 ส่วน เรียกว่า เคอร์เนลสเปซ และ ยูสเซอร์สเปซ ดังนี้

- **เคอร์เนล สเปซ (Kernel Space)** เป็นพื้นที่ในหน่วยความจำที่จัดสรรโดยเฉพาะ สำหรับ เคอร์เนล และองค์ประกอบสำคัญอื่น ๆ โครงสร้างภายในของหน่วยความจำบริเวณ เคอร์เนลสเปซ แบบ Top-Down ประกอบด้วย
 - **เคอร์เนล (Kernel)** คือ โปรแกรมหลักภายในระบบปฏิบัติการ ซึ่งปัจจุบันได้มีการพัฒนาอย่างต่อเนื่อง ผู้อ่านสามารถตรวจสอบเวอร์ชันล่าสุดได้ที่ www.kernel.org
 - **ซิสเต็ม คอล อิน เท อร์เฟซ (System Call Interface)** เป็นช่องทางให้ซอฟต์แวร์ ประยุกต์ เรียกใช้งาน เคอร์เนล เพื่อให้ เคอร์เนล ช่วยสั่งงาน อุปกรณ์ อินพุต/เอาต์พุต การ อ่าน/เขียนไฟล์ เป็นต้น
 - **คำสั่งภาษาเครื่องภายในเคอร์เนล** ที่ผูกติด กับ สถาปัตยกรรม ของ ฮาร์ดแวร์ (Architecture Dependent Kernel Code) สำหรับใช้งาน เชื่อมต่อกับ ฮาร์ดแวร์ ต่าง ๆ โดยเฉพาะ ซึ่ง คำสั่งภาษาเครื่องเหล่านี้ ของ เคอร์เนล จะเปลี่ยนแปลงไปตาม ชีพิญ และ ชิป ที่ ทำงาน เพื่อให้ เคอร์เนล ติดต่อกับ ทรัพยากร เหล่านั้น ได้อย่างถูก ต้อง และ มี ประสิทธิภาพ คำสั่ง และ ข้อมูล ประกอบนี้ เรียกว่า **ดิไวซ์ไดเรเวอร์**
- **ยูสเซอร์ สเปซ (User Space)** เป็น พื้นที่ หน่วยความจำ ที่ จัดสรร ให้ กับ โปรแกรม ประยุกต์ แต่ละ โปรแกรม เท่านั้น ประกอบด้วย
 - **คำสั่งภาษาเครื่อง และ ข้อมูล จาก โปรแกรม ประยุกต์** ที่โหลดจากไฟล์ ELF ที่ อธิบาย ใน หัวข้อนี้ และ
 - **คำสั่งภาษาเครื่อง และ ข้อมูล ภายใน ไลบรารี สำคัญ** ได้แก่ glibc (GNU C Library) เป็นต้น ซึ่ง จะ ช่วย ประหยัด เวลา ใน การ พัฒนา ฟังก์ชัน พื้นฐาน ชีพิญ จะ ทำงาน ตาม คำสั่งภาษาเครื่อง ของ ซอฟต์แวร์ ประยุกต์ และ ไลบรารี ของ ผู้ใช้ ที่ได้ ลือ กัน เข้าสู่ ระบบ ปฏิบัติการ เช่น โปรแกรม เว็บเบราว์เซอร์ สามารถ เชื่อมต่อกับ เครื่อข่าย และ ทรัพยากร อื่น ๆ ผ่าน ทาง ซิสเต็ม คอล อิน เทอร์เฟซ ซอฟต์แวร์ ประยุกต์ ต่าง ๆ ได้แก่ โปรแกรม ที่ ใช้งาน ประเภทต่าง ๆ เช่น เว็บเบราว์เซอร์ (Web Browser) เช่น โมซิลาไฟร์ฟ็อกซ์ (Mozilla Firefox) เกม ต่าง ๆ เป็นต้น โปรแกรม เมอร์ สามารถ พัฒนา ซอฟต์แวร์ ประยุกต์ เหล่านี้ แจก ในรูปของ ฟรีแวร์ (Freeware) และ ฟรี แวร์ บาง ตัว มี การ เปิด เผย ชอร์สโค้ด (Source Code) เรียกว่า โอเพนซอร์ส (Open Source) เพื่อ ให้ โปรแกรม เมอร์ ต่าง ๆ ทั่วโลก สามารถ พัฒนา เสริม เพิ่ม เติม ได้ ตาม เงื่อนไข ของการ เปิด เผย ชอร์สโค้ด เช่น GPL (GNU General Profile License), LGPL (Lesser GPL) เป็นต้น

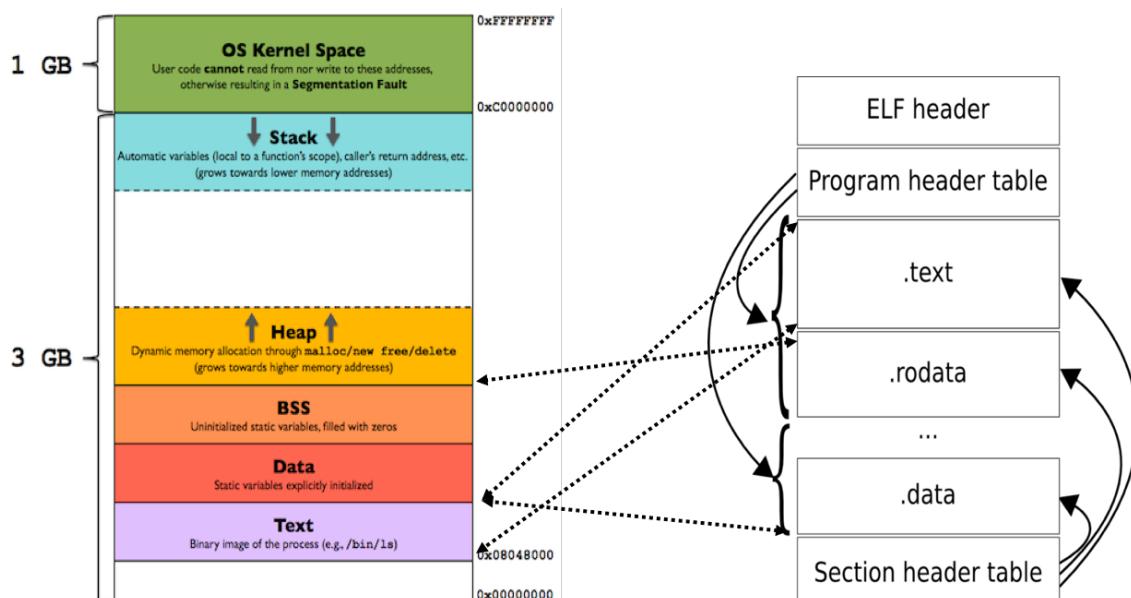
รายละเอียดเพิ่มเติม สามารถศึกษา และ ทดลอง ได้ จากการทดลอง ที่ 4 ภาคผนวก D และ การทดลอง อื่น ตาม ลำดับ

3.3.3 ชีพีย์เฟทซ์คำสั่งภาษาเครื่องของซอฟต์แวร์ประยุกต์จากหน่วยความจำ

หน่วยความจำเวอร์ชวลเมโมรีของแอปพลิเคชันบริเวณยูสเซอร์สเปซจะแบ่งเป็นเซกเมนต์ต่าง ๆ คล้ายกับโครงสร้างของไฟล์ ELF ในรูปที่ 3.16 ตามรายละเอียดดังนี้ ระบบปฏิบัติการจะแมป (Map) หรือจัดวาง

- เท็กซ์เซกเมนต์ เพื่อบรรจุคำสั่งภาษาเครื่องที่อ่านมาจากไฟล์แอปพลิเคชัน และ
- ดาต้าเซกเมนต์ เพื่อบรรจุข้อมูลเลขฐานสองที่อ่านมาจากไฟล์แอปพลิเคชัน

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



รูปที่ 3.16: การจัดวางคำสั่งภาษาเครื่องในเท็กซ์เซกเมนต์และข้อมูลในดาต้าเซกเมนต์จากไฟล์โปรแกรมรูปแบบ ELF ไปยังเวอร์ชวลเมโมรี บริเวณยูสเซอร์สเปซ (User Space) ที่มา: wordpress.com

หลักการนี้จะครอบคลุมระบบปฏิบัติการ 32 และ 64 บิต แต่ต่อมาเล่นนี้จะครอบคลุมเนื้อหาของระบบปฏิบัติการขนาด 32 บิตเท่านั้น ทำให้ระบบสร้างหน่วยความจำเวอร์ชวลเมโมรี ขนาด 2^{32} หรือ 4 กิกะไบต์ (GiB) เท่านั้น เคอร์แนลเองเป็นโปรแกรมตัวหนึ่งที่สามารถจับจองและจัดสรรพื้นที่เวอร์ชวลเมโมรีขนาด 1 กิกะไบต์ (GiB) เรียกว่า **ເຄືອງແນລສເປັດ (Kernel Space)** ตั้งแต่หมายเลข 0xC0000000 หรือ 0x0000 0000 ถึง 0xFFFFFFFF หรือ 0xFFFF FFFF ส่วนพื้นที่ຍູ້ສເຊອຣ໌ສເປັດ (User Space) ของเวอร์ชາລເມໂມຣີขนาด 3 กิกะไบต์ (GiB) จะเริ่มต้นที่หมายเลข 0x0000 0000 ถึง 0xBFFF FFFF ตามตัวอย่าง แบ่งเป็นเซกเมนต์ต่างๆ ดังนี้

- เท็กซ์เซกเมนต์ ตามที่กล่าวไปแล้วในหัวข้อที่ 3.3.2
- ดาต้าเซกเมนต์ ตามที่กล่าวไปแล้วในหัวข้อที่ 3.3.2

- **BSS เช็คเม้นต์** (“Block Started by Symbol”) คือ พื้นที่ส่วนหนึ่งในหน่วยความจำ สำหรับจัดเก็บค่าของตัวแปรชนิด global และ static ที่โปรแกรมเมอร์ยังไม่ได้ตั้งค่าเริ่มต้น เต่อระบบปฏิบัติการ จะทำการตั้งค่าเริ่มต้นให้กลายเป็น 0 โดยอัตโนมัติ ตำแหน่งของ BSS เช็คเม้นต์จะจัดเรียงต่อเนื่องจากตำแหน่งสุดท้ายของ\data\t\cheekment
- **สแต็กเช็คเม้นต์ (Stack Segment)** คือ พื้นที่ส่วนหนึ่งในหน่วยความจำ
 - สำหรับเก็บค่าของตัวแปรชนิดโลคอล (Local Variable) ในภาษา C/C++
 - สำหรับส่งผ่าน (Pass) ค่าตัวแปร เรียกว่า พัฟกชันพารามิเตอร์ (Function Parameter) โดยอาศัยพื้นที่ของสแต็คเก็บค่าตัวแปรที่ต้องการส่งไปยังพัฟกชันผู้ถูกเรียก (Callee Function) รายละเอียดเพิ่มเติมในหัวข้อที่ [4.8](#) และการทดลองที่ 7 ภาคผนวก [G](#)

พัฟกชันแต่ละพัฟกชัน จะ จogn พื้นที่ภายในสแต็กเช็คเม้นต์สำหรับใช้งานของตนเอง เรียกว่า **สแต็กเฟรม (Stack Frame)** โดยอาศัยรีจิสเตอร์ R13 หรือ **สแต็กพอยน์เตอร์ (Stack Pointer)** เก็บค่า แอดเดรสชันบนสุดของสแต็ก (Top of Stack) เช็คเม้นต์ เพื่อความสะดวกในการบริหารจัดการสแต็ก เช่น การจองและคืนพื้นที่ของสแต็กเฟรม เป็นต้น

การบริหารสแต็กเฟรมของแต่ละพัฟกชัน เมื่อมีการวางแผนของช้อนทับกัน ทำให้การบริหารจัดการมีลักษณะ **LIFO** (Last In First Out) นั่นคือ ก่อนเริ่มต้นรันแอปพลิเคชันสแต็กเช็คเม้นต์ของโปรแกรมนี้ยังว่าง เปลา ตำแหน่งสแต็กพอยน์เตอร์ (Stack Pointer) คือ ตำแหน่งบนสุดของสแต็ก เมื่อโปรแกรมทำงานและเรียกใช้พัฟกชัน โปรแกรมจะ จogn พื้นที่ในบริเวณสแต็กเช็คเม้นต์เพื่อเก็บค่าของตัวแปรชนิดโลคอลและอื่น ๆ เรียกว่า **สแต็กเฟรม ณ ตำแหน่งสแต็กพอยน์เตอร์**

เมื่อพัฟกชันทำงานเสร็จสิ้น โปรแกรมเมอร์จะต้องรีเทิร์นออกจากการพัฟกชัน โดยคืนพื้นที่ (Pop) สแต็กเฟรม เพื่อให้สแต็กกว่างลง ทำให้สแต็กพอยน์เตอร์เก็บ แอดเดรสตำแหน่งที่ต่ำลงมา รวมกับว่า มีการนำของที่วางช้อนออกไป ผู้อ่านสามารถศึกษาคำสั่งภาษาแอสเซมบลีที่เกี่ยวข้องกับสแต็กในหัวข้อที่ [4.5](#) และการใช้งานสแต็กโดยละเอียดในการทดลองที่ 8 ภาคผนวก [H](#)

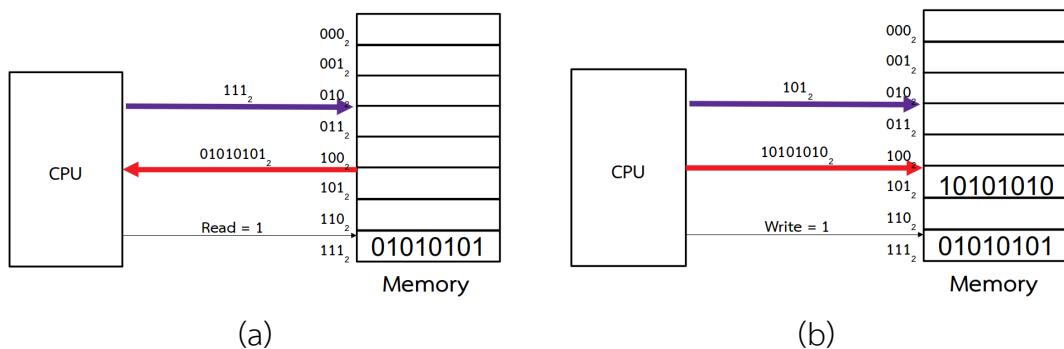
โปรแกรมสามารถเรียกพัฟกชันภายในพัฟกชัน (Nested Function) สแต็กเฟรมจะถูกวางเรียงช้อนทับต่อกันไป โดยเฉพาะ การเรียกพัฟกชันเรียกซ้ำ (Recursive Function) หรือแบบเรียกตัวเอง สแต็กเฟรมจะเรียงช้อนทับต่อกันไปเรื่อยๆ จนกว่าพัฟกชันจะรีเทิร์นกลับ หากไม่มีการรีเทิร์นกลับ พื้นที่ของสแต็กเช็คเม้นต์จะโตขึ้นจนถึงขีดจำกัด เรียกว่า **RLIMIT_STACK** หรือจนสแต็กพอยน์เตอร์ช้อนทับกับตำแหน่งของฮีปเช็คเม้นต์ และทำให้เกิด Exception เพื่อให้ระบบปฏิบัติการນับริหารจัดการต่อไป ซึ่งอยู่นอกเนื้อหาของวิชานี้ ผู้อ่านสามารถค้นคว้ารายละเอียดเพิ่มเติมที่ arm.com

- **ฮีปเช็คเม้นต์ (Heap Segment)** คือ พื้นที่ว่างภายในบูสเซอร์สเปช ที่ระบบปฏิบัติการอนุญาตให้โปรแกรมเมอร์จogn พื้นที่เพื่อเก็บค่าตัวแปรชนิดต่างๆ ตามความต้องการ ในขณะที่โปรแกรมรันอยู่ (Dynamic Allocation) เช่น ลิงก์ลิสต์ (Linked List) และโครงสร้างข้อมูลชนิดต่างๆ โดยใช้พัฟกชัน

malloc ในภาษา C และฟังก์ชัน new ในภาษา C++ โปรแกรมเมอร์จะต้องคืนหน่วยความจำที่จองในฮีปเชิงเมนต์นี้โดยใช้ฟังก์ชัน free ผู้อ่านสามารถศึกษาตัวอย่างการใช้งานฟังก์ชัน free ในภาษา C และโอเปอเรเตอร์ (Operator) delete ผู้อ่านสามารถศึกษาตัวอย่างการใช้งานฟังก์ชัน delete ในภาษา C++

การจองและคืนพื้นที่นี้จะทำให้ขนาดของฮีปเชิงเมนต์เปลี่ยนแปลงจนขอบของฮีปเชิงเมนต์เข้าใกล้ตำแหน่งของสแต็กพอยน์เตอร์ (Stack Pointer) ซึ่งสังเกตทิศทางหัวลูกศรการขยายตัวของสแต็กเชิงเมนต์และฮีปเชิงเมนต์ในรูปที่ 3.16 ว่ามีทิศทางพุ่งเข้าหากัน

3.3.4 ชีพิญอ่าน (Load)/เขียน (Store) ข้อมูลในหน่วยความจำหลัก

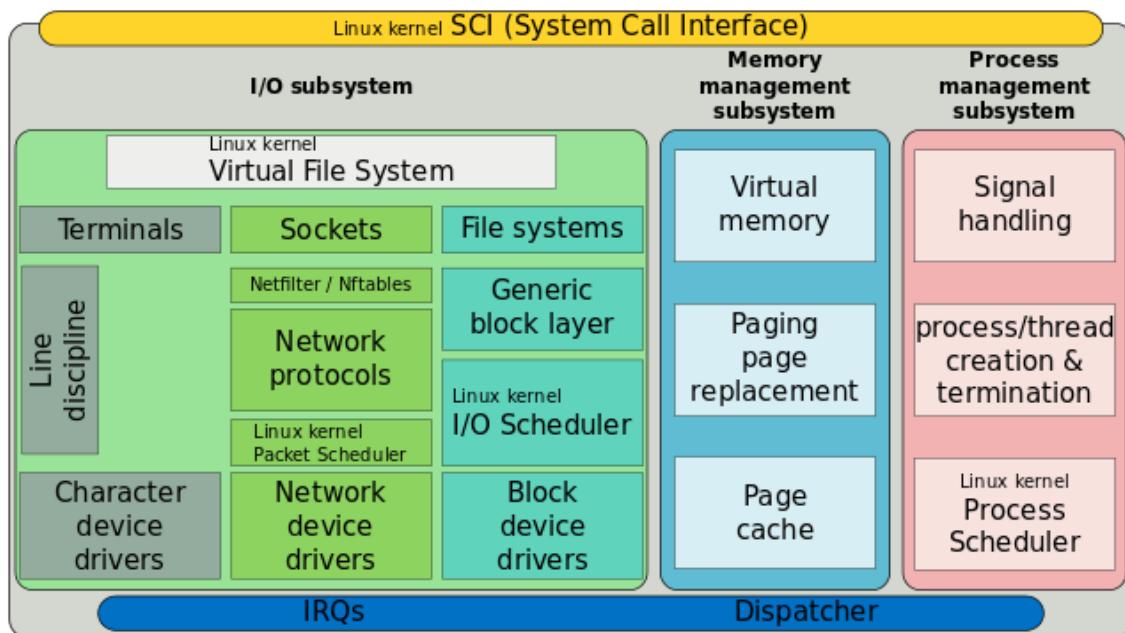


รูปที่ 3.17: (a) ขบวนการอ่าน (Load) ข้อมูลจากหน่วยความจำหลักที่ตำแหน่ง 111₂ (b) ขบวนการเขียน (Store) ข้อมูลในหน่วยความจำหลักที่ตำแหน่ง 101₂

ตัวแปรต่าง ๆ หรือ ข้อมูลชนิดค่าคงที่ เช่น ค่า π อาศัยพื้นที่ในหน่วยความจำ บริเวณดาต้าเชิงเมนต์ และ BSS เชิงเมนต์ เพื่อบรรจุค่าตามจำนวนบิต และชนิดของข้อมูลนั้นในตารางที่ 2.1 เมื่อต้องการนำค่าตัวแปรเหล่านี้ไปประมวลผลได้ ๆ ก็ตาม โปรแกรม จะต้องสั่งให้ชีพิญดำเนินการดังนี้ ชีพิญต้องอ่านค่า (Load) หรืออ่านตัวแปรจากหน่วยความจำหลักไปพักในรีจิสเตอร์ต้นทาง (Source Register) หลังจากนั้น ชีพิญจึงจะสามารถนำค่าในรีจิสเตอร์ต้นทางนั้นไปประมวลผลแล้วเก็บค่าผลลัพธ์ในรีจิสเตอร์ปลายทาง (Destination Register) เมื่อเสร็จสิ้นภารกิจ ชีพิญต้องนำค่าผลลัพธ์ในรีจิสเตอร์ปลายทางเก็บ (Store) หรือเขียนลงในหน่วยความจำหลัก ณ ตำแหน่งหรือแอดдресของตัวแปรนั้นๆ เราเรียก ชีพิญที่ทำงานในลักษณะนี้ว่า ชีพิญสถาปัตยกรรมโหลด/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 รีจิสเตอร์เลขทศนิยม จุดลอยตัว รีจิสเตอร์เลขจำนวนเต็ม จะใช้เก็บค่าตัวแปรชนิด char, int, unsigned int, long เป็นต้น ตัวอย่างโปรแกรมในตารางที่ 3.2 ส่วนรีจิสเตอร์เลขทศนิยม จุดลอยตัวจะใช้เก็บค่าตัวแปรชนิด IEEE754 Single Precision และ Double Precision ทั้งนี้ขึ้นอยู่กับรายละเอียดของชีพิญแต่ละตัว

3.3.5 ชีพีย์ใช้งานอินพุตและเอาต์พุตต่าง ๆ ตามคำสั่งของซอฟต์แวร์ประยุกต์



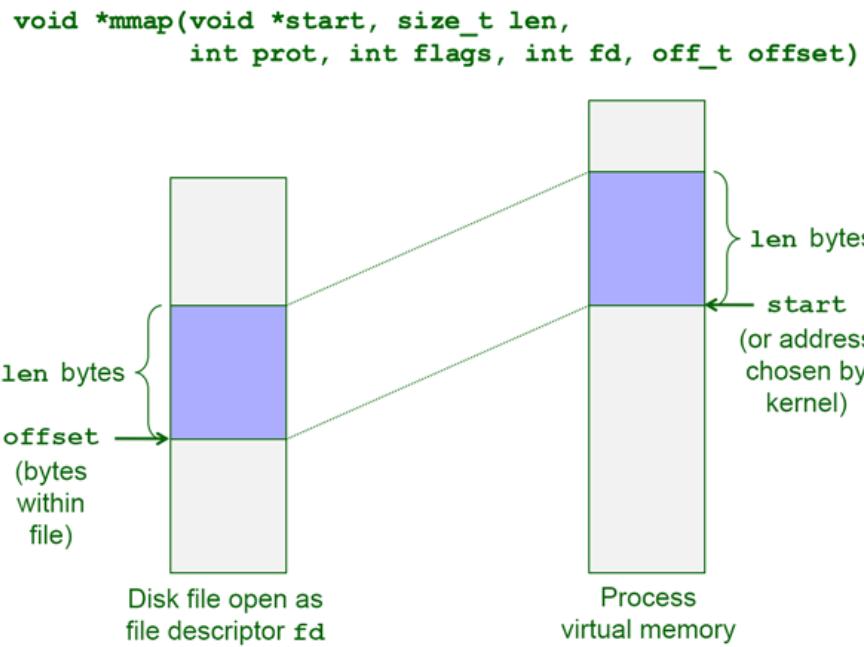
รูปที่ 3.18: โครงสร้างของลินุกซ์ เคอร์แนลในเวอร์ชวล เมโมรี ประกอบด้วย I/O Subsystem, Memory Management Subsystem และ Process Management Subsystem ที่มา: [wikipedia.org](https://en.wikipedia.org)

องค์ประกอบสำคัญภายในเคอร์แนลของระบบปฏิบัติการลินุกซ์ในเวอร์ชวล เมโมรี ประกอบด้วยมอดูลต่าง ๆ ที่สำคัญดังนี้ Process Management Subsystem, Memory Management Subsystem และ I/O subsystem ดังรูปที่ 3.18 ในหัวข้อนี้จะกล่าวถึง I/O Subsystem เป็นหลัก ซึ่งการใช้งานอินพุต/เอาต์พุต เช่น คีย์บอร์ด เม้าส์ พրินเตอร์บางรุ่น จะอาศัยการเชื่อมต่อ กับอุปกรณ์เหล่านี้ในลักษณะของ Character และมีซอฟต์แวร์เฉพาะเรียกว่า **ดีไวซ์ไดรเวอร์** (Device Driver) ซึ่งกำหนดรายละเอียดเอาไว้สำหรับผู้ผลิตและอุปกรณ์แต่ละรุ่น การเชื่อมต่อลักษณะนี้ ได้แก่ โปรแกรม Terminal ผู้อ่านสามารถเรียนรู้เพิ่มเติมจากการทดลองที่ 9 ภาคผนวก |

การเชื่อมต่อ กับเครือข่าย อินเทอร์เน็ต ผ่าน อุปกรณ์ สื่อสาร ชนิดสาย และ ไร้สาย จะอยู่ในรูปของ Network Interface ซึ่งจำเป็นต้องอาศัย โปรแกรม ดีไวซ์ไดรเวอร์ เช่น กัน การเชื่อมต่อ ลักษณะนี้ มีซอฟต์แวร์ เรียกว่า **ซ็อกเก็ต** (Socket)

การใช้งานอินพุต และ เอาต์พุต เช่น อุปกรณ์เก็บรักษาข้อมูล เครื่องอ่านซีดี/ดีวีดีรอม (CD/DVD Rom Drive) แฟลชไดรฟ์ (Flash Drive) เป็นต้น จะอาศัยการเชื่อมต่อ กับอุปกรณ์เหล่านี้ ในลักษณะของบล็อก (Block) ข้อมูล ขนาดตั้งแต่ 512 ไบต์ ขึ้นไป การอ่านหรือเขียนข้อมูลจากไฟล์ จะต้องผ่านกระบวนการนี้ เช่น กัน ซึ่งระบบปฏิบัติการจะเป็นผู้บริหารจัดการ เปิดไฟล์ อ่าน/เขียนข้อมูล และปิดไฟล์ การเชื่อมต่อ ลักษณะนี้ มีซอฟต์แวร์ เรียกว่า **ระบบไฟล์** (File Systems) การใช้งาน อุปกรณ์ อินพุต/เอาต์พุต ทั้งสามรูปแบบ เพื่อ เชื่อมต่อ อุปกรณ์ต่าง ๆ กับระบบปฏิบัติการ ในรูปแบบของ Virtual File System ซึ่งจะมีรายละเอียดเพิ่มเติม ในบทที่ 7 และ การทดลองที่ 12 ภาคผนวก L

3.3.6 ซีพียูอ่าน/เขียนไฟล์ข้อมูลในอุปกรณ์เก็บรักษาข้อมูล



รูปที่ 3.19: หลักการของ Memory Mapped File ที่มา: rice.edu

นิยามที่ 3.3.1 ไฟล์ คือ เลขฐานสองขนาดหลายบิต เรียงต่อเนื่องกัน ซึ่งทุก ๆ 8 บิตจะรวมกันเป็นไบต์ โดยที่แต่ละไบต์เรียงตัวต่อกันตั้งแต่ไบต์ที่ 0 (ต้นไฟล์) ไปจนถึงไบต์สุดท้าย ณ ตำแหน่งลิ้นสุดไฟล์ ซึ่งจะทำหน้าที่เป็นตัวอักษรหรือสัญลักษณ์ EOF (End of text File) ตัวอักษรหรือสัญลักษณ์ EOF เป็นเลขฐานสองขนาด 8 บิต เขียนเป็นฐานสิบหกเท่ากับ $1A_{16}$ หรือฐานสิบเท่ากับ 26_{10} ในตารางรหัสแอลกี ในรูปที่ 2.12

ไฟล์มีหลายชนิด เช่น

- ไฟล์โปรแกรมหรือแอปพลิเคชัน ทำหน้าที่บรรจุโปรแกรม (Executable File) หรือคำสั่ง (Command) ซึ่งมีรูปแบบ ELF ที่กล่าวในหัวข้อที่ 3.3.2 สำหรับระบบปฏิบัติการตระกูลยูนิกซ์และลินุกซ์
- ไฟล์ข้อมูล ได้แก่ ไฟล์เอกสาร Microsoft Word (.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 เรียกว่า Text File เป็นต้น ทั้งนี้ รายละเอียดโครงสร้างไฟล์ข้อมูลแต่ละชนิดจะแตกต่างกันไป
- ไฟล์ชนิดอื่นๆ ประกอบการทำงานของโปรแกรมแต่ละตัว ซึ่งจะมีรายละเอียดแตกต่างกันไปตามที่นักพัฒนาออกแบบ

รูปที่ 3.19 แสดงหลักการของ Memory Mapped File เป็นการจงหน่วยความจำสำหรับอ่านหรือเขียนไฟล์ด้วยฟังก์ชัน `mmap()` ในภาษา C ด้านซ้ายคือ โครงสร้างของไฟล์ในเชิงตรรกะที่ใช้คำสั่ง

`fopen()` เพื่อเปิดไฟล์ ด้านขวาคือ หน่วยความจำที่ถูกจงเพื่อใช้พกข้อมูลที่จะอ่านหรือเขียนในไฟล์ ที่มา: rice.edu

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

การอ่านไฟล์ คือ กระบวนการอ่านข้อมูลจากไฟล์ในอุปกรณ์เก็บรักษาข้อมูล มากบรรจุในหน่วยความจำ ในบริเวณที่จัดสรรให้ เรียกว่า บัฟเฟอร์ ด้วยขบวนการ DMA (Direct Memory Access) รายละเอียดเพิ่มเติมในหัวข้อที่ 6.13 การอ่านไฟล์ จึงเป็นการอ่านข้อมูลจากบัฟเฟอร์ โปรแกรมจะอ่านข้อมูลจากบัฟเฟอร์ จากตำแหน่งเริ่มต้นไปเรื่อย ๆ จนสิ้นสุดบัฟเฟอร์ หากไฟล์มีขนาดใหญ่มากเมื่อเทียบกับขนาดของบัฟเฟอร์ ซึ่งทำงานแบบวงกลม (Circular Buffer) ขบวนการ DMA จะนำข้อมูลชุดถัดไปเขียนวนทับไปเรื่อย ๆ จน เจอตัวอักษรสิ้นสุดไฟล์ หรือ EOF (End of File) เมื่อโปรแกรมอ่านไฟล์เสร็จสิ้นโปรแกรมเมอร์ควรปิดไฟล์ (Close File) เพื่อคืนหน่วยความจำบัฟเฟอร์ที่แมปไว้ให้ระบบปฏิบัติการ แม้ว่าจะเพียงแค่ เปิดไฟล์เพื่อ อ่านเท่านั้น

เมื่อโปรแกรมประมวลผลข้อมูลตามที่ได้รับมอบหมายเรียบร้อย โปรแกรมหรือแอปพลิเคชันควรจะบันทึกหรือเขียนข้อมูลหรือสารสนเทศ (Information) เก็บลงในไฟล์ เพื่อป้องกันการสูญหาย หรือเพื่อนำไปใช้ต่อ การเขียนไฟล์ คือ การเขียนข้อมูลไปยังหน่วยความจำบริเวณบัฟเฟอร์ที่แมปไว้ เมื่อบัฟเฟอร์เต็ม ระบบปฏิบัติการจะย้ายข้อมูลจากหน่วยความจำตำแหน่งนั้น ๆ ไปเขียนลงในอุปกรณ์เก็บรักษาข้อมูลจริง เป็นระยะ ดังนั้น เมื่อใช้งาน (อ่านหรือเขียน) ไฟล์เสร็จสิ้นโปรแกรมเมอร์จะต้องปิดไฟล์เสมอ เพื่อนำข้อมูล ที่ยังคงอยู่ในบัฟเฟอร์เขียนลงในอุปกรณ์เก็บรักษาข้อมูลด้วยขบวนการ DMA เช่นกัน ก่อนที่จะคืนหน่วยความจำบัฟเฟอร์ที่แมปไว้ให้ระบบปฏิบัติการ รายละเอียดเพิ่มเติมในหัวข้อที่ 7.1 และในกิจกรรมท้ายการทดลองที่ 5 ภาคผนวก E

3.3.7 ผู้ใช้ชัตดาวน์ (Shut Down) ระบบปฏิบัติการ

การชัตดาวน์เป็นการสั่งให้ระบบปฏิบัติการอัปเดตข้อมูลต่าง ๆ ของเครื่องในรูปของไฟล์ต่าง ๆ กลับลงในอุปกรณ์เก็บรักษาข้อมูล แล้วปิดไฟล์ที่เปิดค้างไว้ การปิดเครื่องโดยไม่สั่งชัตดาวน์อาจทำให้การบูตเครื่องครั้งต่อไปมีปัญหา และจะทำให้ระบบปฏิบัติการทำงานได้ไม่สมบูรณ์

ผู้ใช้งานระบบลินุกซ์สามารถถอดอาศัยสิทธิ์ (Permission) ของซูเปอร์ยูสเซอร์ (Super User) สั่ง ชัตดาวน์ ระบบ ได้หลายวิธี เช่น การใช้มาส์กกดปุ่มชัตดาวน์ด้านซ้ายบนสุด หรือ พิมพ์คำสั่ง shutdown -h บน โปรแกรม Terminal การชัตดาวน์ทั้งสองวิธี คือ การส่งสัญญาณ (Signaling) ไปยังมอดูล init ให้เปลี่ยนระดับการรัน (Run Level) ให้เป็นระดับ 0 (Halt) เพื่อปิดระบบ

การพิมพ์คำสั่ง shutdown -r เพื่อรีสตาร์ตเครื่อง เป็นการส่งสัญญาณไปยังมอดูล init ให้ระบบปฏิบัติ การเปลี่ยนระดับการรันเป็นระดับ 6 การรีสตาร์ตระบบนี้ยมใช้กับคอมพิวเตอร์ระบบฝั่งตัว เพื่อเริ่มต้นการทำงานของฮาร์ดแวร์และซอฟต์แวร์ใหม่ โปรแกรมเมอร์สามารถเขียนสคริปต์ (Script) ด้วยภาษาเชลล์ สคริปต์ (Shell Script) ที่ลับดเพื่อสั่งให้ระบบรีสตาร์ตโดยอัตโนมัติ ระบบปฏิบัติการไมโครซอฟต์วินโดวส์ สามารถเขียนคำสั่งในลักษณะนี้ได้ เช่นกัน เรียกว่า การเขียนคำสั่งแบบทช (Batch Command)

3.4 สรุปท้ายบท

เนื้อหาในบทนี้ได้นำเสนอพื้นฐานของเครื่องคอมพิวเตอร์ซึ่งมีองค์ประกอบหลัก คือ ฮาร์ดแวร์ และซอฟต์แวร์ ฮาร์ดแวร์ประกอบด้วยอุปกรณ์อิเล็กทรอนิกส์ต่าง ๆ ซึ่งมีซีพียูเป็นจุดศูนย์กลางเชื่อมหน่วยความจำหลัก และวงจร/อุปกรณ์อินพุตเอาต์พุตต่าง ๆ รวมถึงอุปกรณ์เก็บรักษาข้อมูล ในขณะที่ซอฟต์แวร์แบ่งเป็นซอฟต์แวร์ระบบปฏิบัติการ และซอฟต์แวร์ประยุกต์ต่าง ๆ บรรจุอยู่ในอุปกรณ์เก็บรักษาข้อมูล เช่น การ์ดหน่วยความจำ SD ซอฟต์แวร์เหล่านี้สามารถพัฒนาด้วยภาษาต่างๆ เช่น C/C++ Java และภาษาแอสเซมบลี ด้วยการคอมpile (แปลง) และลิงก์ (รวม) ให้กลายเป็นไฟล์รูปแบบ ELF บนระบบปฏิบัติการยูนิกซ์และลินุกซ์ ไฟล์ ELF มีโครงสร้างที่เป็นมาตรฐาน ประกอบด้วยเทกซ์เซกเมนต์ (Text Segment) ซึ่งรวมคำสั่งภาษาเครื่องในรูปของเลขฐานสอง และ ดาต้าเซกเมนต์ (Data Segment) ซึ่งรวมค่าตั้งต้นของตัวแปรที่ได้ประกาศไว้

3.5 คำถามท้ายบท

1. เหตุใดเครื่องคอมพิวเตอร์ทั่วไปควรมีระบบปฏิบัติการ เช่น Microsoft Windows, Apple MacOS, ลินุกซ์เวอร์ชันต่าง ๆ
2. ชิป BCM2837/BCM2711 และชิป AllWinner A64 มีความแตกต่างกันอย่างไร จากข้อมูลในรูปที่ 3.3 ในแต่ละชิปมุ่งเน้นไปที่
 - การรองรับหน่วยความจำหลักภายนอกชิป
 - ซีพียู
 - Video Engine ซึ่งประกอบด้วยวงจรเข้ารหัส (Encoder) และวงจรถอดรหัส (Decoder)
3. หน่วยความจำหลักที่วางขายมีลักษณะเป็นแผงวงจรเมมโมรี่ชิปเรียงกันหลายตัว เหตุใดบอร์ด Pi3/Pi4 จึงมีชิปเพียงตัวเดียว
4. คอมพิวเตอร์ทั่วไปอาจมีการ์ดจอเสริมการทำงานหรือเล่นเกมส์ บอร์ด Pi3/Pi4 มีการ์ดจอหรือไม่ เพราเหตุใด
5. ลองใช้งานเว็บไซต์ www.gsmarena.com เพื่อสืบค้นว่ามีการใช้งานซีพียู ARM Cortex A53/A72 ในเครื่องโทรศัพท์สมาร์ตโฟนหรือไม่ อย่างไร
6. ถ้าจะนำบอร์ด Pi3/Pi4 ไปปรับปรุงให้กลายเป็นเครื่องคอมพิวเตอร์แท็บเล็ต จะจินตนาการว่า จะต้องเพิ่มเติมอุปกรณ์อะไรบ้าง โดยหาข้อมูลเพิ่มเติมในอินเทอร์เน็ต
7. เลเบล main และ printf ในรูปที่ 3.11 ตรงกับฟังก์ชันชื่ออะไรบ้าง
8. เลเบล printf ในรูปที่ 3.11 จะต้อง include ไฟล์ .h ชื่ออะไรเพื่อให้คอมไพล์ทำงานสำเร็จ

9. คำสั่ง BL ในรูปที่ [3.11](#) ตรงกับคำสั่งอะไรในภาษาแอสเซมบลีของชีพีย์ ARM
10. เลเบล main ในรูปที่ [3.11](#) ตรงกับบรรทัดที่เท่าไหร่ในตารางที่ [3.2](#)
11. จงอธิบายการทำงานของโปรแกรมในตารางที่ [3.2](#) ว่าเป็นการวนรอบเพื่อหาค่าอะไรจากตัวแปรใด

บทที่ 4

ภาษาแอสเซมบลีของชิปีย์ ARM ขนาด 32 บิต

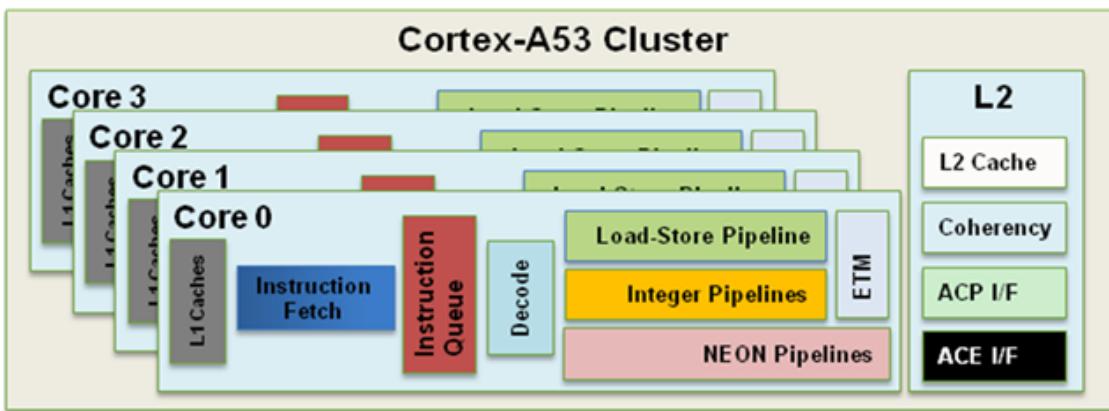
การพัฒนาโปรแกรมด้วยภาษา C/C++ และ แอสเซมบลี มีความเกี่ยวเนื่องกัน ผู้อ่านอาจมีพื้นฐานการพัฒนาโปรแกรมด้วยภาษา C/C++ มาบ้าง จากการทดลองที่ 5 ภาคผนวก E เนื่องจากคำสั่งภาษาแอสเซมบลี (Assembly Instruction) ของชิปีย์ ARM มีหลายเวอร์ชัน ได้แก่ 16, 32 และ 64 บิต อาจทำให้ผู้อ่านเข้าใจสับสน ดังนั้น บทนี้จะเน้นที่คำสั่งภาษาแอสเซมบลีขนาดหรือความยาว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

- เข้าใจโครงสร้างฮาร์ดแวร์ภายในชิปีย์ ARM Cortex A53/A72 บนบอร์ด Pi3/Pi4
- เข้าใจขั้นตอนการทำงานของคำสั่งภาษาแอสเซมบลีขนาดหรือความยาว 32 บิตที่สั่งให้ชิปีย์ ARM ทำงาน
- เข้าใจรูปแบบคำสั่ง และ การพัฒนาโปรแกรมด้วยคำสั่งภาษาแอสเซมบลีของชิปีย์ ARM โดยใช้พื้นฐานการพัฒนาโปรแกรมภาษา C
- รับรู้วิวัฒนาการของภาษาแอสเซมบลีในชิปีย์ที่บริษัท ARM ออกแบบและเป็นที่นิยมในตลาดโลก

4.1 โครงสร้างของชิปีย์ ARM Cortex A53/A72

จากรูปที่ 3.3 ผู้อ่านได้เรียนรู้โครงสร้างภายในของชิปที่คล้ายกับชิป BCM2837/BCM2711 ซึ่งเป็นศูนย์กลางของบอร์ด Pi3 โดยภายในชิปมีชิปีย์ ARM Cortex A53 จำนวน 4 แกนประมวลผล (Core) แกนประมวลผลหมายเลข 0 จนถึงแกนประมวลผลหมายเลข 3 ซึ่งเราสามารถยืนยันได้จากการตรวจสอบในการทดลองที่ 4 ภาคผนวก D แกนประมวลผลทั้งสี่มีองค์ประกอบหลักภายในแต่ละแกนเหมือนกัน นั่นคือ แกนประมวลผลที่ 0 ของชิปีย์ Cortex A53/A72 ของรูปที่ 4.1 มีโครงสร้างภายในที่ไม่ซับซ้อนและมีการทำงานแบบไปป์ไลน์ (Pipeline) ประกอบด้วยมอดูล/บล็อกการจราจรด้านซ้ายสุดไปด้านขวาสุด ดังนี้

- **แคชคำสั่ง (Instruction Cache)** ลำดับที่ 1 ทำหน้าที่เก็บคำสั่ง (Instruction) เรียกว่า I-Cache ลำดับที่ 1 โดยจะเก็บคำสั่งภาษาเครื่องล่าสุด ซึ่งคำสั่งเหล่านี้ถูกเฟทซ์มาจากเทกซ์เชิก



รูปที่ 4.1: โครงสร้างภายในแกนประมวลผลหมายเลข 0 ของชิปปี้ Cortex A53 ออกแบบโดยบริษัท ARM ที่มา: tomshardware.com

เมนต์ของเวอร์ชัลเมโมรี 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) ทำหน้าที่แปลเวอร์ชัลแอดเดรสเป็นแอดเดรสภายในภาพและส่งแอดเดรสภายในภาพไปยังแคชคำสั่ง (I-Cache) ลำดับที่ 1 และรอรับคำสั่งภาษาเครื่องที่เฟทซ์มาพักเก็บในคิว (Instruction Queue)
- คิวคำสั่ง** (Instruction Queue) ทำหน้าที่เป็นคิวเก็บพักคำสั่งภาษาเครื่องจำนวน 8-16 คำสั่ง เพื่อส่งต่อให้วงจรถอดรหัส
- วงจรถอดรหัส (Decode) ทำหน้าที่แปลความหมายของคำสั่งภาษาเครื่องซึ่งเป็นเลขฐานสองที่ส่งมาจากคิวคำสั่ง วงจรถอดรหัสแบ่งคำสั่งภาษาเครื่องออกเป็นชุดบิตต่าง ๆ ตามตัวอย่างในรูปที่ 3.12 หัวข้อที่ 3.2.5 เพื่อทำความเข้าใจและส่งสัญญาณควบคุม (Control Signal) ไปยังมอดูล/วงจรไปป์ไลน์ที่เหมาะสมตามชนิดของคำสั่งนั้น ๆ
- วงจรไปป์ไลน์ (Pipeline) สำหรับปฏิบัติตาม (Execute) คำสั่งภาษาเครื่องในหัวข้อที่ 3.2.5 ที่ผ่านการถอดรหัสแบ่งเป็นวงจร 3 ชนิดตามประเภทของคำสั่ง ประกอบด้วย
 - วงจรไปป์ไลน์สำหรับการอ่าน/เขียนค่าตัวแปรกับหน่วยความจำ (Load-Store Pipeline) เพื่ออ่าน/เขียนค่าของตัวแปรเหล่านั้นตามตำแหน่งและเลขทศนิยม IEEE754 กับหน่วยความจำเวอร์ชัลเมโมรีผ่านทางแคชข้อมูล (Data Cache หรือ D-Cache) ลำดับที่ 1 มีการทำงานเบื้องต้นคล้ายกับบัฟเฟอร์ขนาดเล็ก ทำหน้าที่พักเก็บข้อมูลหรือค่าของตัวแปรที่ซอฟต์แวร์ต้องการอ่าน หรือเขียน จำกัดตัวเข็ม เมนต์ อีป เข็ม เมนต์ สเต็ก เข็ม เมนต์ ยกเว้นเทิร์น เข็ม เมนต์ การทำงานเบื้องต้นของแคชข้อมูลคล้ายกับแคชคำสั่ง ต่างกันตรงที่ชิปปี้สามารถเปลี่ยนแปลงค่าของข้อมูลในแคชข้อมูลได้ แต่ไม่สามารถเปลี่ยนแปลงแคชคำสั่งได้ ค่าของตัวแปรจะถูกถ่าย

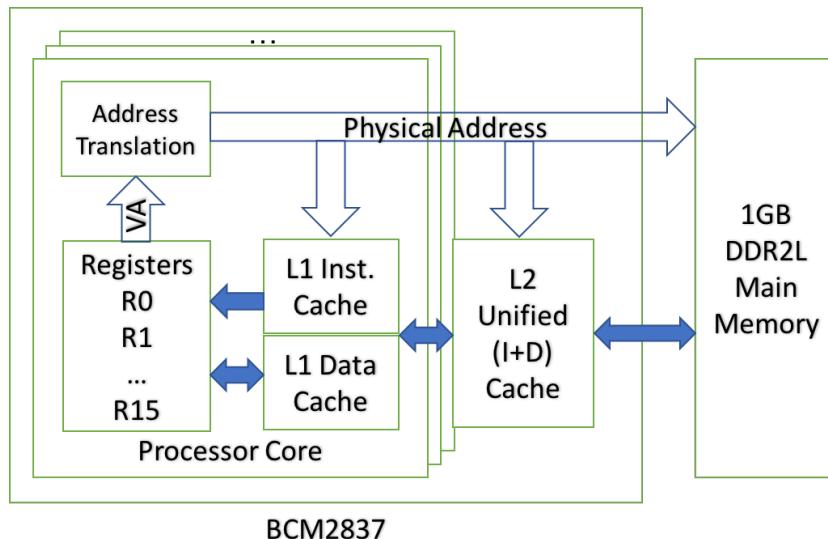
โอน (อ่าน/เขียน) ระหว่างรีจิสเตอร์กับแคชข้อมูลด้วยคำสั่งในหัวข้อที่ 4.5 เมื่อรีจิสเตอร์มีค่าของตัวแปรต่าง ๆ แล้ว โปรแกรมจึงสามารถสั่งให้ชีพีย์คำนวณค่าเหล่านั้นด้วยวงจรไปป์ไลน์สำหรับการประมวลผลเลขจำนวนเต็ม และ วงจรไปป์ไลน์สำหรับการประมวลผลเลขทศนิยม IEEE754

- วงจรไปป์ไลน์สำหรับการประมวลผลเลขจำนวนเต็ม (Integer Pipeline) ขนาด 8, 16, 32 และ 64 บิต ด้วยคำสั่งทางคณิตศาสตร์และตรรกศาสตร์ตามชนิดข้อมูลในบทที่ 2 ข้อมูลเหล่านี้เป็นข้อมูลเชิงเดียว หรือ สเกลาร์ (Scalar) ซึ่งมีค่าเก็บอยู่ในรีจิสเตอร์จำนวนเต็ม (R0-R15) เท่านั้น ตำราเล่มนี้จะเน้นที่การประมวลผลเลขจำนวนเต็มเป็นหลัก
- วงจรไปป์ไลน์สำหรับ การประมวลผลเลขทศนิยมฐานสอง ชนิดจุดลอยตัว IEEE754 ชื่อ NEON (Floating-Point Pipeline) สามารถรองรับการประมวลผลเลขทศนิยมฐานสองชนิดจุดลอยตัวมาตรฐาน IEEE754 ตามรายละเอียดในหัวข้อที่ 2.6 โดยวงจรสามารถรองรับการคำนวณข้อมูลเชิงเดียวหรือสเกลาร์ ได้แก่ เลขจำนวนจริง เลขทศนิยมฐานสิบ เป็นต้น และ ข้อมูลแบบเวกเตอร์ (Vector) ที่ต้องการความแม่นยำสูง ได้แก่ ตัวแปรอาร์เรย์ ข้อมูลตำแหน่งตัวละครในเกม 3 มิติ ข้อมูลจุดภาพ ข้อมูลเสียง เป็นต้น
- แคชลำดับที่ 2 (Level 2) มีความจุมากกว่า แคชลำดับที่ 1 หลายเท่า ตัวแต่ต้องใช้เวลาเข้าถึง (Access Time) นานกว่า แคชลำดับที่ 1 โดยที่ชีพีย์ทั้งสี่แกน ประมวลผล จะใช้งานแคชลำดับที่ 2 ร่วมกัน (Share) หมายความว่า ชีพีย์ทุกแกน ประมวลผลสามารถค้นหาคำสั่ง และ ข้อมูลในแคชลำดับที่ 2 หลังจากที่ไม่เจอคำสั่ง/ข้อมูลในแคชลำดับที่ 1 (L1) ภายในแต่ละแกน ประมวลผล หากไม่เจอในแคชลำดับที่ 2 อีก ชีพีย์จะต้องค้นหาในหน่วยความจำภายในภาพและต้องรอนานขึ้น อีกบทที่ 5 จะอธิบายการทำงานของหน่วยความจำต่าง ๆ โดยละเอียด

ผู้อ่านสามารถค้นควารายละเอียดเกี่ยวกับชีพีย์ ARM Cortex A53 เพิ่มเติมได้ที่ wikichip.org

วงจร วงจรไปป์ไลน์สำหรับการประมวลผลเลขจำนวนเต็ม ในแต่ละแกน ประมวลผลของชีพีย์ ARM Cortex A53 ในรูปที่ 4.2 ประกอบด้วย รีจิสเตอร์จำนวนเต็ม R0-R15 แคชต่าง ๆ และหน่วยความจำต่าง ๆ เพื่อให้สามารถประมวลผลตัวเลขจำนวนเต็มเท่านั้น โดยอาศัยวงจร ALU (Arithmetic Logic Unit) ตามที่ได้อธิบายไปแล้วในบทที่ 2

ส่วนของวงจร วงจรไปป์ไลน์สำหรับการอ่าน/เขียนค่าตัวแปร กับหน่วยความจำ จะเกี่ยวข้องกับหน่วยความจำชนิดต่าง ๆ โดยตัวแปรชื่อต่าง ๆ นั้นถูกกำหนดพื้นที่ไว้ในเช็คเม้นต์ต่าง ๆ ของหน่วยความจำยกเว้นเทิร์ชเช็คเม้นต์ แต่การประมวลผลค่าของตัวแปรเหล่านี้ โปรแกรมจะต้องสั่งให้ชีพีย์โหลด (Load) หรืออ่านค่าของตัวแปรชนิดเลขจำนวนเต็มจากหน่วยความจำ มาพักเก็บในรีจิสเตอร์ R0 - R12 ก่อน แล้วจึงใช้คำสั่งทางคณิตศาสตร์และตรรกศาสตร์ต่าง ๆ ประมวลผลเลขจำนวนเต็มในรีจิสเตอร์ R0 ถึง R12 เมื่อประมวลผลแล้วเสร็จ โปรแกรมต้องนำค่าจากรีจิสเตอร์ไปเก็บ (Store) หรือเขียนในหน่วยความจำ ตามตำแหน่งตัวแปรที่ได้โหลดมาด้วยเหตุผลด้านประสิทธิภาพ ซึ่งจะอธิบายในบทที่ 5 ดังนั้น เราจึงเรียกการทำงานลักษณะนี้ว่า สถาปัตยกรรมโหลด/สโตร์ (Load/Store Architecture)



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

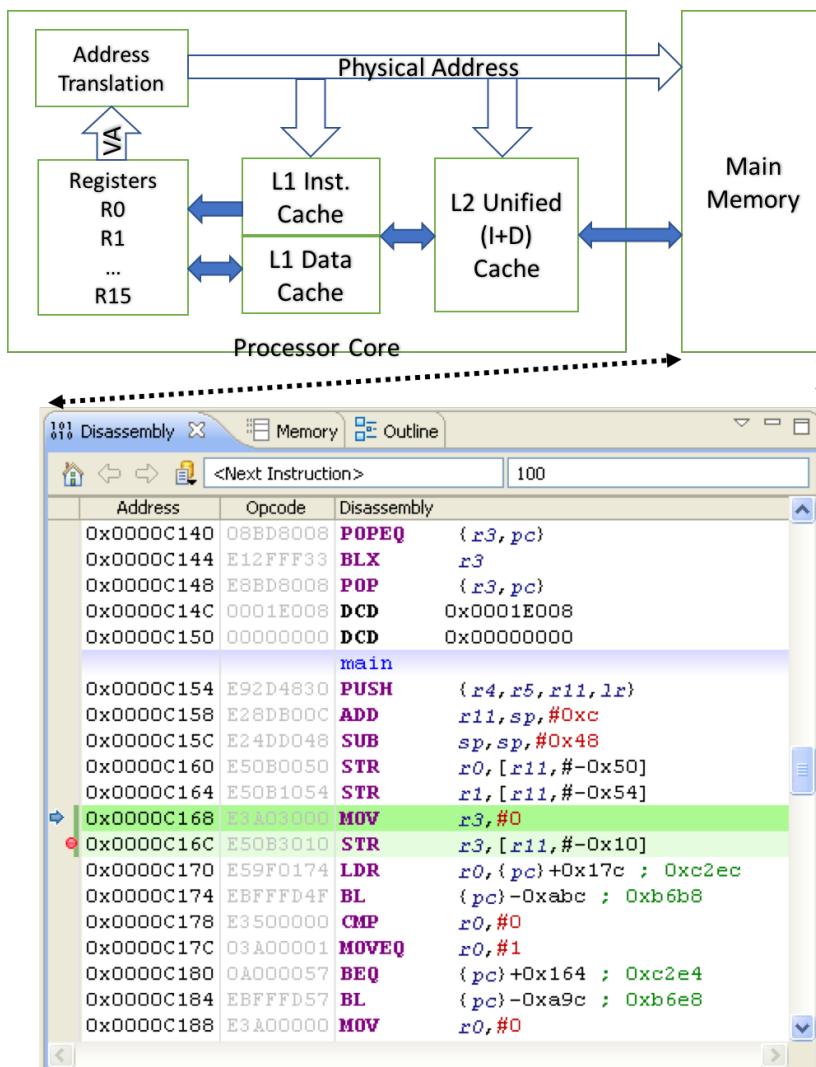
4.2 สถาปัตยกรรมชุดคำสั่ง (Instruction Set Architecture)

ตำราเล่มนี้จะเน้นคำสั่งภาษาและเซมบลิที่ตรงกับภาษาเครื่องของชีพิญ ARM ที่มีความยาว 32 บิตหรือ 4 ไบต์ ในหัวข้อที่ 3.2.5 สำหรับการประมวลผลเลขจำนวนเต็มชนิดไม่มีเครื่องหมาย และชนิดมีเครื่องหมายแบบ 2's Complement คำสั่งเหล่านี้ประกอบด้วยรายละเอียดในแต่ละตัวที่ เหล่านี้

- ชนิดของคำสั่งภาษาและเซมบลิ แบ่งเป็น
 - คำสั่งประการและตั้งค่าเริ่มต้นตัวแปร ในหัวข้อที่ 4.4
 - คำสั่งการถ่ายโอนข้อมูลระหว่างตัวแปรในหน่วยความจำกับรีจิสเตอร์ ในหัวข้อที่ 4.5
 - คำสั่งประมวลผลคณิตศาสตร์และตรรกศาสตร์จากข้อมูลในรีจิสเตอร์ ในหัวข้อที่ 4.6
 - คำสั่งการควบคุมการทำงาน เพื่อการตัดสินใจและการวนรอบทำซ้ำ ในหัวข้อที่ 4.7
 - คำสั่งเรียกใช้ฟังก์ชันและรีเทิร์น (Return) กลับ ในหัวข้อที่ 4.8
- รีจิสเตอร์ ทำหน้าที่พักเก็บข้อมูลชั่วคราว สำหรับประมวลผล ด้วยคำสั่งทางคณิตศาสตร์ ตรรกศาสตร์ และอื่น ๆ มีจำนวนรวม 16 ตัว เรียกว่า R0, R1, ..., R15 รีจิสเตอร์ทุกตัวมีความยาว 32 บิต แต่ละตัวมีหน้าที่แตกต่างกัน คือ
 - รีจิสเตอร์ R15 เรียกว่า โปรแกรมเคาน์เตอร์ (Program Counter: PC) คือ รีจิสเตอร์สำหรับเก็บแอดเดรส หรือหมายเลขไปต์ ของคำสั่งในเทิกซ์เซกเมนต์ของหน่วยความจำที่ชีพิญจะ
 - * เฟทช์(Fetch) หรืออ่านคำสั่งภาษาเครื่อง
 - * ถอดรหัส (Decode) คำสั่งที่เฟทช์มาเพื่อสร้างสัญญาณไปควบคุม
 - * วงจรปฏิบัติ (Execute) ตามคำสั่งนั้นและ

- * เปลี่ยนแปลงค่าในรีจิสเตอร์ PC เป็น $PC=PC+4$ เพื่อเก็บแอดเดรสของคำสั่งถัดไปหมายเลข 4 หน่วยเป็นไบต์ เนื่องจากทุกคำสั่งในตำราเล่มนี้มีความยาว 4 ไบต์ตามที่กล่าวมาข้างต้น หรือ เปลี่ยนแปลงเป็นค่าอื่นตามคำสั่งควบคุมการทำงานและคำสั่งเรียกใช้ฟังก์ชัน
- รีจิสเตอร์ R14 เรียกว่า **ลิงก์รีจิสเตอร์** (Link Register: LR) คือ รีจิสเตอร์สำหรับเก็บแอดเดรสของคำสั่งที่ต้องการจะเรียกกลับ โดยรีจิสเตอร์นี้ทำงานคู่กับคำสั่ง BL (Branch and Link) และคำสั่ง BX LR
- รีจิสเตอร์ R13 เรียกว่า **สแต็กพอยน์เตอร์** (Stack Pointer: SP) คือ รีจิสเตอร์สำหรับเก็บแอดเดรสหรือตำแหน่งยอด (Top) ของสแต็กเชิงเมนต์ ซึ่งเรียกว่า **สแต็ก** โดยรีจิสเตอร์นี้ทำงานคู่กับคำสั่งที่เกี่ยวข้องกับหน่วยความจำ ในหัวข้อที่ 4.5 ผู้อ่านสามารถทบทวนรายละเอียดเกี่ยวกับสแต็กในหัวข้อที่ 3.3.3 ที่ผ่านมา หมายเหตุ คำว่า **สแต็ก** ในตำราเล่มนี้หมายถึงสแต็กเชิงเมนต์ มิได้หมายถึงโครงสร้างข้อมูล (Data Structure) ชนิดหนึ่ง
- รีจิสเตอร์ R4-R12 เป็นรีจิสเตอร์สำหรับใช้งานทั่วไป
- รีจิสเตอร์ R0-R3 เป็นรีจิสเตอร์สำหรับสำหรับใช้งานทั่วไป และใช้ส่งค่าพารามิเตอร์ (Parameter) ไปให้ฟังก์ชัน โดยรีจิสเตอร์เหล่านี้ทำงานร่วมกับคำสั่ง BL (Branch and Link) ในหัวข้อที่ 4.8
- รีจิสเตอร์ R0 เป็นรีจิสเตอร์สำหรับรีเทิร์นค่าข้อมูลจากฟังก์ชัน โดยทำงานร่วมกับคำสั่ง BX LR
- **ชนิดและขนาดของตัวแปร** ผู้อ่านสามารถเทียบเคียงพื้นที่และขนาดของหน่วยความจำ (Memory Space) กับตารางที่ 2.1 โดยข้อมูลที่เก็บอยู่ในตัวแปรแต่ละชนิดต้องการพื้นที่ไม่เท่ากัน ดังนี้
 - ชนิด **ไบต์** (byte) มีขนาด 8 บิต เหมาะสำหรับตัวแปรชนิดอักขระ เช่น char สำหรับเก็บอักขระตามรหัสมาตรฐาน ASCII ด้วยพื้นที่ 8 บิตในหน่วยความจำ และ unsigned char สำหรับเลขจำนวนเต็มไม่มีเครื่องหมายความยาว 8 บิต ซึ่งได้สรุปไว้ในตารางที่ 2.13
 - ชนิด **halfword** (halfword) มีขนาด 16 บิต เช่น short และ unsigned short เหมาะสำหรับตัวแปรชนิดอักขระตามมาตรฐาน Unicode
 - ชนิด **เวิร์ด** (word) มีขนาด 32 บิต เหมาะสำหรับตัวแปรชนิดจำนวนเต็ม เช่น int และ unsigned int เป็นต้น
 - ชนิด **ดับเบิลเวิร์ด** (doubleword) 64 บิต เหมาะสำหรับตัวแปรชนิดจำนวนเต็ม เช่น long long เป็นต้น

4.3 ตัวอย่างคำสั่งภาษาเครื่องในเทิกซ์เซกเมนต์



รูปที่ 4.3: คำสั่งภาษาเครื่องที่ถูกอ่าน (Load) เข้าสู่หน่วยความจำและแสดงในรูปของภาษาแอสเซมบลีบนโปรแกรมซิมูเลเตอร์ (Simulator) ในบริเวณเทิกซ์เซกเมนต์ ที่มา: arm.com

นักพัฒนาโปรแกรมบน อุปกรณ์ที่ใช้ชิปปี้ ARM สามารถ จำลอง (Simulate) การทำงานบน เครื่องคอมพิวเตอร์ตั้ง โต๊ะ เพื่อ จำลอง การทำงานของซอฟต์แวร์ และ ฮาร์ดแวร์ ในสภาพ ใกล้ เคียง กับ สถานการณ์จริง โปรแกรมเมอร์ สามารถ แก้ปัญหา ที่เกิดขึ้นได้ ก่อน ทำให้ ประหยัด เวลา และ ต้นทุน รูปที่ 4.3 แสดง คำสั่งภาษาเครื่องที่ถูกอ่าน (Load) เข้าสู่หน่วยความจำและแสดงในรูปของภาษาแอสเซมบลีบน โปรแกรมซิมูเลเตอร์ (Simulator) ประกอบด้วย คอลัมน์ ต่าง ๆ ต่อไปนี้

- **แอดเดรส (Address)** เท่ากับ **หมายเลขไบต์** ในรูปแบบ เลขฐานสิบ กจำนวน 8 หลัก (ขึ้นต้นด้วย 0x) ซึ่งเท่ากับ เลขฐานสอง จำนวน 32 บิต ใน คอลัมน์ ทางซ้าย ผู้อ่าน ควร จินตนาการ หมายเลขไบต์ นี้ เป็น หมายเลข ชั้น ของ อาคาร ที่มี จำนวน ชั้น เยอะ มาก ซึ่ง ใน ที่นี่ มี จำนวน $2^{32} = 4,294,967,296$ ชั้น แต่ละ ชั้น สามารถบรรจุ ข้อมูล เพียง 1 ไบต์ โดยจะเริ่มนับ จาก ชั้น ที่ 0x0000 0000 หรือ 0000 0000₁₆ จนถึง ชั้น ที่ 0xFFFF FFFF หรือ FFFF FFFF₁₆

- **อปโค้ด (Opcode)** คือ คำสั่งภาษาเครื่องในรูปของเลขฐานสิบหก อปโค้ดมีความยาว 4 ไบต์ หรือ เลขฐานสิบหกจำนวน 8 หลัก หรือ เลขฐานสองจำนวน 32 บิต ผู้อ่านสามารถทำความเข้าใจคำสั่งภาษาเครื่องได้จากตัวอย่างในหัวข้อที่ [3.2.5](#)
- **ดิส แอกซ์ เซมบลี (Disassembly)** คือ การแปลคำสั่งภาษาเครื่องในคอลัมน์ อปโค้ด ให้กลับเป็นภาษาแอกซ์เซมบลี ARM โดยจะแปลภาษาเครื่องแต่ละบรรทัดเป็นคำสั่งภาษาแอกซ์เซมบลีเพื่อให้ผู้ใช้โปรแกรมซิมูเลเตอร์อ่านแล้วเข้าใจง่ายขึ้น

ยกตัวอย่างเช่น ที่หน่วยความจำตำแหน่ง $0x0000\ C140$ บรรจุอปโค้ด $08BD\ 8008_{16}$ ความยาว 4 ไบต์ซึ่งเป็นรหัสภาษาเครื่องของคำสั่ง POPEQ r3, pc

ตำแหน่งถัดไปคือ แอดдрес $0x0000\ C144 = 0x0000\ C140 + 0x0000\ 0004$ ห่างจากเดิมจำนวน 4 ไบต์บรรจุอปโค้ด $E12F\ FF33_{16}$ ซึ่งเป็นรหัสภาษาเครื่องของคำสั่ง BLX r3 หมายเหตุ ช่วงว่างทุก ๆ 4 หลัก คือ สัญลักษณ์ที่ผู้เขียนใส่เพิ่มเอง เพื่อช่วยให้ผู้อ่านสามารถอ่านหมายเลขที่เรียงติดกันทีละ 4 หลักได้ง่ายขึ้น มีหน้าที่คล้ายกับเครื่องหมาย , ในเลขฐานสิบที่ช่วยแบ่งเลขจำนวนมากทุก ๆ สามหลัก

pc หรือ PC คือ รีจิสเตอร์ R15 ทำหน้าที่เก็บแอดdress ของคำสั่งปัจจุบัน ในหน่วยความจำโดยสังเกตได้จากลูกศรสีน้ำเงิน เพื่อให้ซิพีย์สามารถอ่านคำสั่งภาษาเครื่องนั้นไปถอดรหัสและปฏิบัติตาม ลูกศรสีน้ำเงินในรูปที่ [4.3](#) ซึ่ง 0x0000 C168 ความหมายคือ ค่าในรีจิสเตอร์ pc เท่ากับ $0x0000\ C168$ ซึ่งหน่วยความจำที่ตำแหน่งนี้บรรจุคำสั่นหรืออปโค้ด $E3A0\ 3000_{16}$ ซึ่งเป็นเลขฐานสิบหกของคำสั่งภาษาแอกซ์เซมบลี MOV r3, #0

เมื่อ ประมวลผล คำสั่ง ที่ แอดdress $0x0000\ C168$ สำเร็จแล้ว pc จะบวกเพิ่มขึ้นอีก 4 ไบต์ ($pc=pc+4$) เป็น แอดdress $0x0000\ C16C$ เพื่อนำคำสั่ง $E50B\ 3010_{16}$ ซึ่งเป็นรหัสภาษาเครื่องของคำสั่ง STR r3, [r11, #-0x10] มาถอดรหัสและประมวลผลตามต่อไป แต่ในรูปที่ [4.3](#) ผู้ใช้ได้ตั้งตำแหน่งเบรกพอยน์ (Break Point) โดยสังเกตได้จากวงกลมสีแดงทางซ้ายสุด เบรกพอยน์ จะทำให้โปรแกรมซิมูเลเตอร์หยุดทำงานชั่วคราว เพื่อให้ผู้ใช้สามารถศึกษาการทำงานของคำสั่งจากค่ารีจิสเตอร์ต่าง ๆ ที่เกี่ยวข้อง หลังจากนั้นผู้ใช้สามารถดำเนินการรันซิมูเลชัน (Simulation) ต่อไป ผู้อ่านสามารถฝึกฝนการใช้โปรแกรมลักษณะนี้ในการทดลองที่ 5 ด้วยภาษา C ก่อน แล้วจึงทำการทดลองที่ 6 ด้วยภาษาแอกซ์เซมบลีของซิพีย์ ARM

4.4 คำสั่งประการและตั้งค่าเริ่มต้นของตัวแปรในดาต้าเช็กเมนต์

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

ตารางที่ 4.1: ตัวอย่างโปรแกรมภาษาแอสเซมบลีเพื่อประการและตั้งค่าเริ่มต้นตัวแปรขนาด 32 บิตจำนวน 2 ตัวแปร ในพื้นที่ของดาต้าเช็กเมนต์

บรรทัดที่	เลbel	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 คือการสั่งให้การจดข้อมูลในหน่วยความจำให้มีพื้นที่เรียงต่อเนื่องกัน 4 ไบต์ เพื่อให้ระบบบริหารจัดการได้ง่ายขึ้น ตัวแปรชนิดจำนวนเต็ม 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) ค่าตัวแปรระหว่างดาต้าเซ็กเมนต์และรีจิสเตอร์

คำ สั่ง ถ่าย โอน ข้อมูล ระหว่าง ตัวแปร ใน หน่วย ความ จำ และ รี จิ ส เตอร์ เป็น คำ สั่ง ที่ จำเป็น สำหรับ โปรแกรมคอมพิวเตอร์ เนื่องจากตัวแปรที่ประมวลคำนี้คือ ชื่อของข้อมูล หรือชื่อชุดข้อมูล หรือชื่ออาร์เรย์ใน หน่วยความจำ โดยขนาดของข้อมูลจะเปลี่ยนแปลงตามชนิด เช่น word' ขนาด 32 บิต byte มีขนาด 8 บิต เป็นต้น ดังนั้น ซึพิยจะต้องใช้คำสั่งโหลดหรืออ่านค่าของตัวแปรนั้นมาพักไว้ในรีจิสเตอร์ก่อน 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"> Rd = Mem[Rn] สำเนาข้อมูล 32 บิต จากหน่วยความจำที่แอ็อดдресส์ Rn ไปเขียนในรีจิสเตอร์ Rd
STR Rd, [Rn]	<ul style="list-style-type: none"> Mem[Rn] = Rd สำเนาข้อมูล 32 บิตในรีจิสเตอร์ Rd ไปเขียนในหน่วยความจำที่แอ็อดdress Rn
LDRB Rd, [Rn]	<ul style="list-style-type: none"> Rd = Mem[Rn] สำเนาข้อมูลจำนวน 8 บิตจากหน่วยความจำที่แอ็อดdress Rn ไปเขียนในรีจิสเตอร์ Rd
STRB Rd, [Rn]	<ul style="list-style-type: none"> Mem[Rn] = Rd สำเนาข้อมูล 8 บิตล่างสุดในรีจิสเตอร์ Rd ไปเขียนในหน่วยความจำที่แอ็อดdress Rn
PUSH {register list}	<ul style="list-style-type: none"> สำเนาข้อมูล 32 บิตจากรีจิสเตอร์ที่ปรากฏในรายชื่อรีจิสเตอร์ ไปวางบนยอดสเต็กช่วงคราวตามลำดับจากซ้ายไปขวา ปรับเปลี่ยนค่ารีจิสเตอร์ SP ให้สอดคล้องกับคำสั่ง
POP {register list}	<ul style="list-style-type: none"> สำเนาข้อมูล 32 บิตจากยอดสเต็กไปบรรจุในรีจิสเตอร์ ที่ปรากฏในรายชื่อรีจิสเตอร์ตามลำดับจากขวาไปซ้าย ปรับเปลี่ยนค่ารีจิสเตอร์ SP ให้สอดคล้องกับคำสั่ง

คำสั่ง LDR Rd, [Rn] จะสั่งให้ซิพิยสำเนาข้อมูล 32 บิต จากหน่วยความจำ ณ แอ็อดdressที่รีจิสเตอร์ Rn เก็บไปเขียนในรีจิสเตอร์ Rd ซึ่งตรงกับความหมายของประโยชน์นี้ $Rd = Mem[Rn]$

ในทางกลับกัน คำสั่ง Stoer จะสั่งให้ฮาร์ดแวร์ภายในซิพิยดำเนินการ โดยอ่านค่าจากรีจิสเตอร์ไป บันทึกในหน่วยความจำ ณ แอ็อดdressที่ตัวแปรนั้นตั้งอยู่ ยกตัวอย่าง คำสั่ง STR Rd, [Rn] จะสั่งให้ซิพิย สำเนาข้อมูล 32 บิต จากรีจิสเตอร์ Rd ไปเก็บยังหน่วยความจำ ณ แอ็อดdressที่รีจิสเตอร์ Rn ซึ่งตรงกับ ความหมายของประโยชน์นี้ $Mem[Rn] = Rd$

คำสั่ง PUSH และ POP มีลักษณะพิเศษกว่าคำสั่ง Load และ Store ทั่วไป คือ PUSH register list จะทำการสำเนาข้อมูลจากรีจิสเตอร์ไปวางบนสเต็กช่วงคราว แล้ว ขยับตำแหน่งบนสุดของสเต็กไปยัง หมายเลขแอ็อดdressใหม่ โดยสเต็ก คือ ชื่อย่อของสเต็กเซ็กเมนต์ เป็นพื้นที่สำคัญของหน่วยความจำในรูป

ที่ 3.16 สำหรับคำสั่ง POP register list จะทำงานตรงข้ามกับ คำสั่ง PUSH คือ สำเนาข้อมูลจากสแต็กกลับไปคืนให้รีจิสเตอร์ แล้วขับตำแหน่งบนสุดของสแต็กไปยังหมายเลขแอดเดรสใหม่ ผู้อ่านสามารถทำความเข้าใจจากการทดลองที่ 8 ในภาคผนวก H

ตาราง ต่อไปนี้สรุปคำสั่งโอนถ่ายข้อมูลระหว่างหน่วยความจำและรีจิสเตอร์ (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"> Rd = address(var) เขียนค่าแอดเดรสของตัวแปร var ลงในรีจิสเตอร์ Rd
LDR Rd, [Rn]	<ul style="list-style-type: none"> Rd = Mem[Rn] อ่านค่าจากหน่วยความจำที่แอดเดรส Rn และเขียนค่านั้นในรีจิสเตอร์ Rd
LDR Rd, [Rn, #Imm]	<ul style="list-style-type: none"> Rd = Mem[Rn + #Imm] อ่านค่าจากหน่วยความจำที่แอดเดรส Rn+Imm และเขียนค่านั้นในรีจิสเตอร์ Rd โดยที่ค่าในรีจิสเตอร์ Rn ไม่เปลี่ยนแปลง
LDR Rd, [Rn], #Imm	<ul style="list-style-type: none"> Rd = Mem[Rn] อ่านค่าจากหน่วยความจำที่แอดเดรส Rn และเขียนค่านั้นในรีจิสเตอร์ Rd หลังจากนั้นค่าในรีจิสเตอร์ Rn = Rn + #Imm
LDR Rd, [Rn, #Imm]!	<ul style="list-style-type: none"> Rd = Mem[Rn+ #Imm] อ่านค่าจากหน่วยความจำที่แอดเดรส Rn+Imm และเขียนค่านั้นในรีจิสเตอร์ Rd หลังจากนั้นค่าในรีจิสเตอร์ Rn = Rn + #Imm
LDR Rd [Rn, Rm]	<ul style="list-style-type: none"> Rd = Mem[Rn+Rm] (32 bit copy) อ่านค่าจากหน่วยความจำที่แอดเดรส Rn+Rm และเขียนค่านั้นในรีจิสเตอร์ Rd โดยที่ค่าในรีจิสเตอร์ Rn ไม่เปลี่ยนแปลง

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

ตารางที่ 4.2: ตัวอย่างโปรแกรมภาษาแอสเซมบลีเพื่ออ่านค่าตัวแปรจากデータเซ็กเมนต์

บรรทัดที่	เลbel	คำสั่ง	คอมเมนต์
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

$x = (a + b) - c;$

ผู้อ่านสามารถทำความเข้าใจการทำงานได้จากคำอธิบายภายใต้คอมเม้นต์ คอมเม้นต์ และคำอธิบายตามหมายเลขบรรทัด ดังนี้

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

บรรทัดที่	เลขบล็อก	คำสั่ง	คอมเม้นต์
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

1. คือ การโหลดตำแหน่งของตัวแปร a ไปบรรจุใน R4
2. คือ การโหลดค่าของตัวแปร a ไปบรรจุใน R0
3. คือ การโหลดตำแหน่งของตัวแปร b ไปบรรจุใน R4
4. คือ การโหลดค่าของตัวแปร b ไปบรรจุใน R1
5. คือ การคำนวณค่า $a + b$ ไปบรรจุใน R3
6. คือ การโหลดตำแหน่งของตัวแปร c ไปบรรจุใน R4
7. คือ การโหลดค่าของตัวแปร c ไปบรรจุใน R4
8. คือ การคำนวณค่าที่อยู่ใน $(a+b) - c$ ไปบรรจุใน R3
9. คือ การโหลดตำแหน่งของตัวแปร x ไปบรรจุใน R4
10. คือ การสตอร์ค่าที่อยู่ใน R3 = $((a+b)-c)$ ไปบรรจุในตำแหน่งของตัวแปร x

4.6 คำสั่งประมวลผลข้อมูลในรีจิสเตอร์ (Register Data Processing Instructions)

คำสั่งประมวลผลข้อมูลในรีจิสเตอร์ (Register Data Processing Instructions) เชื่อมโยงกับคณิตศาสตร์ และข้อมูลชนิดต่าง ๆ ในบทที่ 2 คำสั่งในหัวข้อนี้จะจำกัดอยู่ที่ข้อมูลจำนวนเต็มเท่านั้นเพื่อให้เนื้อหาไม่ซับซ้อนจนเกินไป แบ่งเป็นคำสั่งประเภทย่อย ๆ ดังนี้

- คำสั่งทางคณิตศาสตร์ เพื่อคำนวณเลขจำนวนเต็ม
- คำสั่งเลื่อนบิตข้อมูล เพื่อเลื่อนบิตข้อมูลในรีจิสเตอร์ไปทางซ้ายหรือทางขวาด้วยวงจรบาร์เรลชิฟเตอร์ (Barrel Shifter)
- คำสั่งทางคณิตศาสตร์และเลื่อนบิตข้อมูล เพื่อเลื่อนบิตข้อมูลไปทางซ้ายหรือทางขวา ก่อนนำไปคำนวณด้วยวงจร ALU
- คำสั่งทางตรรกศาสตร์ เพื่อคำนวณค่าทางตรรกศาสตร์ของเลขจำนวนเต็มด้วยวงจร ALU

4.6.1 คำสั่งทางคณิตศาสตร์ (Arithmetic Instructions)

คำสั่ง บวก/ลบ และคูณสำหรับเลขจำนวนเต็มชนิดมีเครื่องหมายและไม่มีเครื่องหมาย กระทำโดยวงจรอาร์ดแวร์ เรียกว่า ALU (Arithmetic and Logic Unit) ซึ่งประกอบด้วยวงจรบวก/ลบเลข และคูณเลขตามหลักการคณิตศาสตร์คอมพิวเตอร์ในหัวข้อที่ 2.3 โปรแกรมเมอร์สามารถสั่งให้ชีพิญโดยเฉพาะ 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 ที่ทำการคำนวณตามชนิดของข้อมูล และอปโค้ดต่าง ๆ โดย

- บิต Z (Zero) เท่ากับ

- 1 ใช้ตรวจสอบว่าผลลัพธ์มีค่าเท่ากับศูนย์
- 0 ใช้ตรวจสอบว่าผลลัพธ์มีค่าไม่เท่ากับศูนย์

- บิต C (Carry) เท่ากับ

- 1 บิตทด (Carry bit) c_n เท่ากับ 1
- 0 บิตทด (Carry bit) c_n เท่ากับ 0

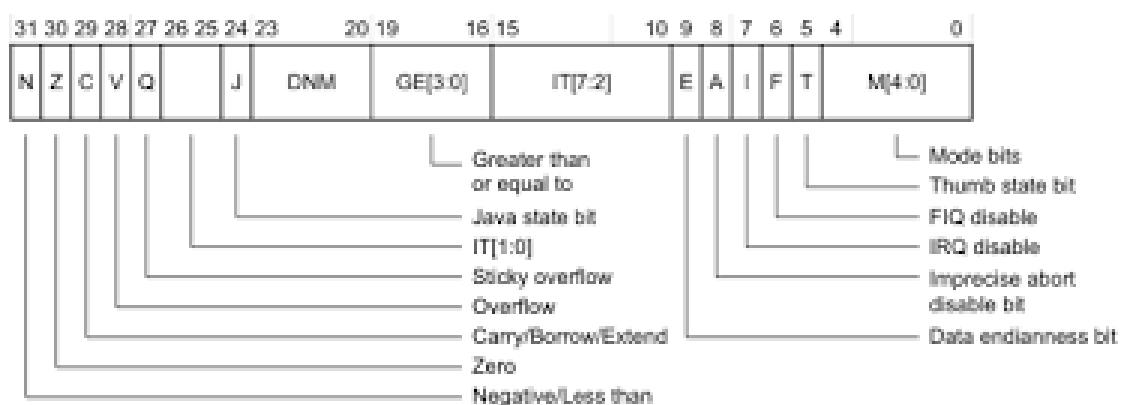
- บิต N (Negative) เท่ากับ

- 1 ใช้ตรวจสอบว่าผลลัพธ์มีค่าน้อยกว่าศูนย์ หรือ ติดลบ
- 0 ใช้ตรวจสอบว่าผลลัพธ์มีค่ามากกว่าศูนย์

- บิต V (oVerflow) เท่ากับ

- 1 ใช้ตรวจสอบว่าผลการคำนวนเกิดโอเวอร์โฟล์ว
- 0 ใช้ตรวจสอบว่าผลการคำนวนไม่เกิดโอเวอร์โฟล์ว

เพื่อสร้างความมั่นใจว่าผลลัพธ์ถูกต้อง (Valid) ตามตัวอย่างในหัวข้อที่ 2.3.1 และ 2.3.2



รูปที่ 4.4: รีจิสเตอร์สำหรับเก็บสถานะของซีพียู ปัจจุบัน (Current Program Status Register: CPSR)
พร้อมรายละเอียดทั้ง 32 บิต ที่มา: arm.com

4.6.2 คำสั่งเลื่อนบิตข้อมูล (Bit Shift Instructions)

คำสั่งเลื่อนบิตข้อมูลแบ่งเป็น การเลื่อนบิตทางซ้ายและทางขวา โดยวงจรบาร์เรลชิฟเตอร์ (Barrel Shifter) ในรูปที่ 4.5 การเลื่อนบิตไปทางขวาจำแนกได้เป็นการเลื่อนทางตรกศาสตร์ (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

- ทางตรกศาสตร์ (Logical Shift) นิยมใช้กับเลขจำนวนเต็มชนิดไม่มีเครื่องหมาย แบ่งเป็น
 - การเลื่อนบิตข้อมูลไปทางซ้าย วงจรจะป้อนศูนย์เข้ามาทางบิตขวาสุดแทนที่ข้อมูลเดิมที่ถูกเลื่อนไปทางซ้าย
 - การเลื่อนบิตข้อมูลไปทางขวา วงจรจะป้อนศูนย์เข้ามาทางบิตซ้ายสุดแทนที่ข้อมูลเดิมที่ถูกเลื่อนไปทางขวา
- ทางคณิตศาสตร์ (Arithmetic Shift) วงจรจะป้อนบิตทางซ้ายสุดด้วยบิตเครื่องหมายเข้ามาแทนที่ข้อมูลเดิมที่ถูกเลื่อนไปทางซ้ายหรือทางขวา เพื่อให้เครื่องหมายของข้อมูลในรีจิสเตอร์ไม่เปลี่ยนแปลง นิยมใช้กับเลขจำนวนเต็มชนิดมีเครื่องหมายแบบ 2's Complement

4.6.3 คำสั่งทางคณิตศาสตร์และเลื่อนบิต (Arithmetic and Bit 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

คำสั่งชนิดนี้ เป็นจุดเด่นของชิปปุ๊ย ARM เนื่องจากช่วยลดจำนวนคำสั่ง และประหยัดเวลาในการประมวลผล แบ่งเป็นขั้นตอนอยู่ ๆ 2 ขั้นตอน คือ

- อ่านข้อมูลจากรีจิสเตอร์ Rm เพื่อป้อนให้กับวงจรบาร์เรลชิฟ เตอร์ทำการเลื่อนบิตข้อมูลไปทางซ้ายหรือขวา เป็นจำนวนบิตตามค่า Shamt (Shift Amount) ซึ่งจัดเรียงในบิตที่ 7-11 ของคำสั่งคณิตศาสตร์ในรูปที่ 3.12
- นำผลลัพธ์ที่ได้จากการเลื่อนบิตไปคำนวณต่อตามคำสั่งหลัก หมายเหตุ ค่าที่ยังเก็บอยู่ในรีจิสเตอร์ Rm จะไม่เปลี่ยนแปลง

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

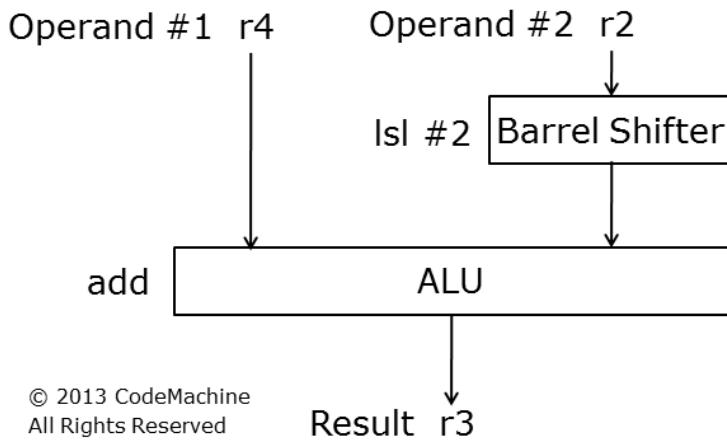
โดยค่าใน R4 เป็นตัวตั้งหรือโอเปอเรนด์ (Operand) ที่ 1 ค่าใน R2 เป็นตัวตั้งหรือโอเปอเรนด์ที่ 2 และ Shamt มีค่าเท่ากับ 2_{10} หรือ 00010_2 ความยาว 5 บิต ความหมาย คือ การนำ R2 เลื่อนบิตไปทางซ้ายจำนวน 2 บิต บวกกับ R4 และจึงเก็บผลลัพธ์ใน R3 หรือเท่ากับ $R3 = R4 + (R2 \times 4)$ โดยค่าที่เก็บในรีจิสเตอร์ R2 จะไม่เปลี่ยนแปลง

4.6.4 คำสั่งทางตรรกศาสตร์ (Logical Instructions)

ตัวอย่างภาษา C การคำนวณด้วยการกระทำทางตรรกศาสตร์ โปรแกรมเมอร์นิยมกระทำกับข้อมูลชนิดไม่มีเครื่องหมาย เพื่อคำนวณข้อมูลที่ละเอียด แต่การประมวลผลเกิดขึ้นพร้อมกันทุกบิต โดยบิตข้อมูลตัวตั้งจะกระทำกับบิตที่ตรงกันของตัวกระทำเท่านั้น (Bitwise Operation) ยกตัวอย่างเช่น

```
unsigned int a = 0xffff0000;
unsigned int b = 0x0000ffff;
unsigned int c;
c = a & b; /* c = 0x00000000 AND */
```

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



รูปที่ 4.5: คำสั่ง `add r3, r4, r2, lsl #2` หมายถึง $r3 = r4 + (r2 \ll 2)$ โดยบวกค่า r4 กับค่า r2 หลังจากเลื่อนบิตไปทางซ้ายจำนวน 2 บิตแล้วจึงเก็บผลลัพธ์ใน r3 ที่มา: CodeMachine.com หมายเหตุ ค่าที่ยังเก็บอยู่ในรีจิสเตอร์ r2 จะไม่เปลี่ยนแปลง

```
c = a | b; /* c = 0xffffffff OR */
c = !b;      /* c = 0xffff0000 INVERSE */
c = a ^ b;   /* c = 0xffffffff Ex-OR */
```

ชีพิญ ARM รองรับการทำงานด้วยคำสั่งแอสเซมบลีเหล่านี้

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 ผู้อ่านควรทราบหัวข้อพีชคณิตบูลลิน (Boolean Algebra) ในวิชาออกแบบวงจรดิจิทัล โดยเฉพาะเรื่องตารางความจริง (Truth Table) ของวงจรเกตชนิดต่าง ๆ

4.7 คำสั่งควบคุมการทำงาน (Control Instructions)

คำสั่งควบคุมการทำงาน (Control Instructions) ในภาษาสูง แบ่งเป็น ประโยชน์การตัดสินใจ เช่น ประโยชน์ IF, IF-ELSE, Switch-Case และประโยชน์การวนรอบชนิดต่าง ๆ เช่น FOR, WHILE, DO-WHILE เป็นต้น โครงสร้างคำสั่งภาษาแօสเซมบลีของซีพียู ARM ที่รองรับประโยชน์เหล่านี้ ประกอบด้วย

- คำสั่ง CMP (Compare) ทำหน้าที่เปรียบเทียบระหว่างค่า寄存器กับค่าคงที่ หรือระหว่างค่า寄存ร์ 2 ตัว โปรแกรมเมอร์สามารถเขียนคำสั่งนี้ได้ 2 รูปแบบตามตารางต่อไปนี้

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

ค่าบิต NZCV ในรีจิสเตอร์ CPSR (หัวข้อที่ 4.6) คือ ผลลัพธ์ที่ได้จากการคำสั่ง CMP โดยคำสั่งนี้ทำการเปรียบเทียบเลขจำนวนเต็มสองค่า ($Rn - Rm$) หรือ ($Rn - #Imm$)

- คำสั่ง B (Branch) จะย้ายการทำงานไปยังคำสั่งเป้าหมายตามเงื่อนไข (Condition) ที่ได้จากการตรวจสอบบิต NZCV ที่เกิดจากคำสั่ง CMP ในเชิงเทคนิค PC จะเปลี่ยนค่าเป็นค่าแอดเดรสที่ตรงกับเลเบลเป้าหมาย (Target Label: $PC = address(label)$) เพื่อเฟทธิ์คำสั่งที่ตำแหน่งนั้น ตามรูปแบบเงื่อนไขต่าง ๆ ในตารางต่อไปนี้

รูปแบบ	ความหมาย
B label	$pc = address(label)$ เฟทธิ์คำสั่งที่ตามหลัง label (อย่างไม่มีเงื่อนไข)
BEQ label	$pc = address(label)$ เฟทธิ์คำสั่งที่ตามหลัง label หากผลลัพธ์การเปรียบเทียบท่ากัน
BNE label	$pc = address(label)$ เฟทธิ์คำสั่งที่ตามหลัง label หากผลลัพธ์การเปรียบเทียบไม่เท่ากัน
BGT label	$pc = address(label)$ เฟทธิ์คำสั่งที่ตามหลัง label หากผลลัพธ์การเปรียบเทียบมากกว่า
BLT label	$pc = address(label)$ เฟทธิ์คำสั่งที่ตามหลัง label หากผลลัพธ์การเปรียบเทียบน้อยกว่า
BGE label	$pc = address(label)$ เฟทธิ์คำสั่งที่ตามหลัง label หากผลลัพธ์การเปรียบมากกว่าหรือเท่ากับ
BLE label	$pc = address(label)$ เฟทธิ์คำสั่งที่ตามหลัง label หากผลลัพธ์การเปรียบน้อยกว่าหรือเท่ากับ

คำสั่ง B (branch) ตรงกับคำสั่ง Jump ในภาษาแอสเซมบลีอื่น ๆ หรือ ประโยค go-to ในภาษา C/C++ คำสั่ง B ทั้งชนิดมีและไม่มีเงื่อนไขจะทำให้แอดเดรสในรีจิสเตอร์ PC เปลี่ยนค่าเป็น หมายเลขอาร์กูเมนต์ หรือ แอดเดรส หรือ ตำแหน่งของ label วงจรดิจิทัลในชิปปี้สามารถตรวจสอบผลลัพธ์ของเงื่อนไขต่าง ๆ 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 ผลการลบมากกว่าหรือเท่ากับ 0
7. **VS** (Overflow Set): เกิด Overflow ขึ้น คือ การตรวจสอบว่า $V=1$ หรือไม่ นั่นคือ เกิดโอเวอร์ฟลัวร์
8. **VC** (Overflow Clear): ไม่เกิด Overflow คือ การตรวจสอบว่า $V=0$ หรือไม่ นั่นคือ ไม่เกิดโอเวอร์ฟลัวร์
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

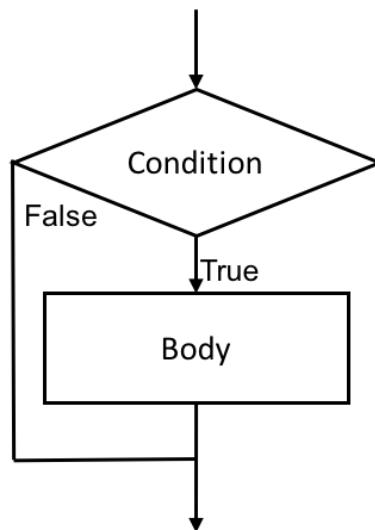
การตัดสินใจ IF ขึ้นอยู่กับเงื่อนไขของประโภค IF ว่าผลลัพธ์ที่ได้จะเป็นจริงหรือเท็จ

- หากจริง ซึ่งจะปฏิบัติตามประโภคคำสั่งที่โปรแกรมเมอร์ต้องการ
- หากเท็จ ซึ่งจะปฏิบัติตามประโภคคำสั่งที่อยู่ถัดจากประโภค IF ไป

การตัดสินใจพื้นฐานที่เข้าใจง่าย เพราะมีความซับซ้อนน้อย เหมาะสำหรับผู้ฝึกเขียนโปรแกรม และสามารถมองภาพรวมของโครงสร้างการตัดสินใจได้โดยง่าย

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 ผู้อ่านสามารถเขียนเป็นชุดคำสั่งภาษาและแบบลีขิ้นซึ่ง ARM ได้ในตารางที่ [4.4](#)

ผู้อ่านสามารถทำความเข้าใจการทำงานได้จากคำอธิบายภายใต้คอลัมน์ คอมเมนต์ และคำอธิบายตามหมายเลขอรรถทัศ ดังนี้

- คือ การโหลดตำแหน่ง (แอดเดรส) ของตัวแปร a ไปบรรจุใน R4
- คือ การโหลดค่าของตัวแปร a ไปบรรจุใน R0

3. คือ การโหลดตัวแหน่ง (แอดเดรส) ของตัวแปร b ไปบรรจุใน R4
4. คือ การโหลดค่าของตัวแปร b ไปบรรจุใน R1
5. คือ การคำนวณค่า $a + b$ ไปบรรจุใน R3
6. คือ การโหลดตัวแหน่ง (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 แอดเดรสในรีจิสเตอร์ PC จะเปลี่ยนเป็น ตำแหน่งของเลเบลชื่อ exit หรือ $PC = \text{address(exit)}$ หากไม่เป็นจริง แอดเดรสในรีจิสเตอร์ PC จะเปลี่ยนเป็นตำแหน่งของคำสั่งถัดไป หรือ $PC=PC+4$
10. คือ การโหลดตัวแหน่ง (แอดเดรส) ของตัวแปร x ไปบรรจุใน R4
11. คือ การโหลดค่าของตัวแปร x ไปบรรจุใน R5
12. คือ การโหลดตัวแหน่ง (แอดเดรส) ของตัวแปร y ไปบรรจุใน R4
13. คือ การโหลดค่าของตัวแปร y ไปบรรจุใน R6
14. คือ การคำนวณค่า $a + b$ ไปบรรจุใน R5
15. คือ การโหลดตัวแหน่ง (แอดเดรส) ของตัวแปร x ไปบรรจุใน R4
16. คือ การสโตร์ค่าของ R5 ไปบรรจุตำแหน่ง (แอดเดรส) ของตัวแปร x
17. คือ คำสั่งที่ขึ้นต้นด้วยเลเบล exit

ตารางที่ 4.4: ตัวอย่างโปรแกรมตามประযุคเนื่องใน if

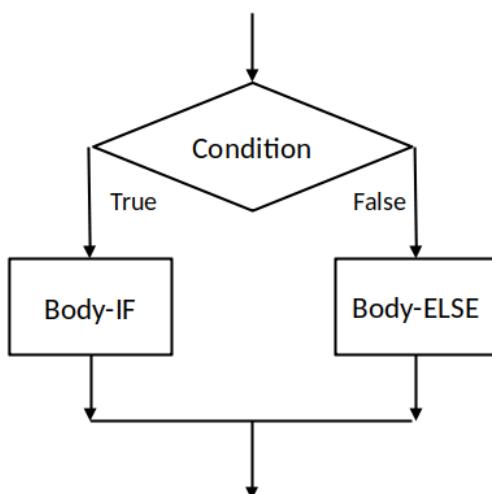
บรรทัดที่	เลbel	คำสั่ง	คอมเมนต์
1		LDR R4, =a	@ get address of variable a
2		LDR R0, [R4]	@ get value of variable a
3		LDR R4, =b	@ get address of variable b
4		LDR R1, [R4]	@ get value of variable b
5		ADD R3, R0, R1	@ compute a+b
6		LDR R4, =c	@ get address of variable c
7		LDR R2, [R4]	@ get value 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 = 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 ขึ้นอยู่กับเงื่อนไขของประโยค IF ว่าผลลัพธ์ที่ได้จะเป็นจริงหรือเท็จ

- หากจริง ชีพีย์ จะปฏิบัติตาม ประโยค คำสั่ง ที่โปรแกรมเมอร์ ต้องการ เมื่อแล้วเสร็จ ชีพีย์ จะข้าม ประโยคที่ไม่เกี่ยวข้อง เพื่อประมาณผลคำสั่งถัดไป
- หากเท็จ ชีพีย์ จะทำการคำสั่งที่มีเลbel ELSE นำหน้า เมื่อแล้วเสร็จจะประมาณผลประโยคอื่นถัดไป

การตัดสินใจพื้นฐานที่ซับซ้อนเพิ่มขึ้นหมายความว่าผู้ฝึกเขียนโปรแกรม และสามารถตรวจสอบฟลว์ชาร์ตตามรูปที่ 4.7 โดยคำสั่งภายในกล่อง Body-IF คือ คำสั่งที่ต้องการให้ประมาณผลหากเงื่อนไขเป็นจริง และคำสั่งภายในกล่อง Body-ELSE คือ คำสั่งที่ต้องการให้ประมาณผลหากเงื่อนไขเป็นเท็จ



รูปที่ 4.7: ฟลว์ชาร์ตการทำงานของประโยค IF-ELSE

ตัวอย่างการเขียนโปรแกรม ประโยค IF-ELSE ในภาษา C/C++

```

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

จาก ประโยค IF-ELSE นี้ ผู้อ่านสามารถเขียนเป็นชุดคำสั่งภาษาแอสเซมบลีของชีพีย์ ARM ได้ในตารางที่ 4.5 และทำความเข้าใจ การทำงานได้ จากคำ อธิบาย ภายใต้ คอลัมน์ คอม เมนต์ และคำ อธิบาย ตามหมายเลขบรรทัด ดังนี้

ตารางที่ 4.5: ตัวอย่างโปรแกรมตามประโยคเงื่อนไข if-else

บรรทัดที่	เลbel	คำสั่ง	คอมเมนต์
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 = 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 = x - y
23		LDR R4, =x	@ get address of variable x
24		STR R5, [R4]	@ store value of variable x
25	exit:	...	@ label exit

1. คือ การโหลดตำแหน่ง (แอดเดรส) ของตัวแปร a ไปบรรจุใน R4
2. คือ การโหลดค่าของตัวแปร a ไปบรรจุใน R0
3. คือ การโหลดตำแหน่ง (แอดเดรส) ของตัวแปร b ไปบรรจุใน R4
4. คือ การโหลดค่าของตัวแปร b ไปบรรจุใน R1
5. คือ การคำนวณค่า a + b ไปบรรจุใน R3
6. 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 ซึ่งจะกระโดดไปทำงานคำสั่งที่ขึ้นต้นด้วย exit หากไม่เป็นจริง ซึ่งจะทำงานประโยคที่ 10 ต่อไป

10. คือ การโหลดตัวแหน่ง (แอดเดรส) ของตัวแปร x ไปบรรจุใน R4
11. คือ การโหลดค่าของตัวแปร x ไปบรรจุใน R5
12. คือ การโหลดตัวแหน่ง (แอดเดรส) ของตัวแปร y ไปบรรจุใน R4
13. คือ การโหลดค่าของตัวแปร y ไปบรรจุใน R6
14. คือ การคำนวณค่า $x = x + y$ ไปบรรจุใน R5
15. คือ การโหลดตัวแหน่ง (แอดเดรส) ของตัวแปร x ไปบรรจุใน R4
16. คือ การสโตร์ค่าของ R5 ไปบรรจุที่ตำแหน่ง (แอดเดรส) ของตัวแปร x
17. คือ การบังคับให้ชีพิญกระโดดไปทำงานคำสั่งที่ขึ้นต้นด้วยเลเบล exit
18. คือ การโหลดตัวแหน่ง (แอดเดรส) ของตัวแปร x ไปบรรจุใน R4
19. คือ การโหลดค่าของตัวแปร x ไปบรรจุใน R5
20. คือ การโหลดตัวแหน่ง (แอดเดรส) ของตัวแปร y ไปบรรจุใน R4
21. คือ การโหลดค่าของตัวแปร y ไปบรรจุใน R6
22. คือ การคำนวณค่า $x = x - y$ ไปบรรจุใน R5
23. คือ การโหลดตัวแหน่ง (แอดเดรส) ของตัวแปร x ไปบรรจุใน R4
24. คือ การสโตร์ค่าของ R5 ไปบรรจุที่ตำแหน่ง (แอดเดรส) ของตัวแปร x
25. คือ คำสั่งที่ขึ้นต้นด้วยเลเบล exit

4.7.3 การวนรอบชนิด FOR

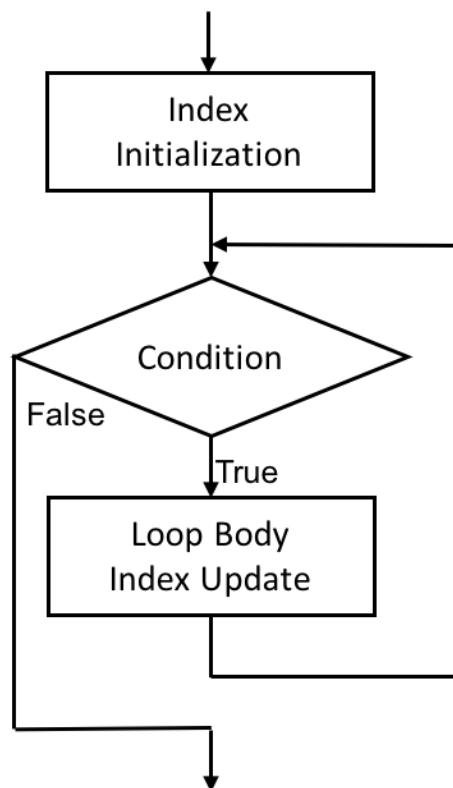
การพัฒนาโปรแกรมคอมพิวเตอร์ในอดีตมุ่งเน้นที่การคำนวณแก้ปัญหาทางคณิตศาสตร์ ยกตัวอย่าง เช่น

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

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

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

ซอฟต์แวร์นี้ทำการตั้งค่าเริ่มต้นให้ตัวแปร x เท่ากับ 0 ให้ตัวแปร i เท่ากับ 1 และจึงเพิ่มค่า x ด้วยค่า i และวนกลับมาทำประโยชน์นี้อีก ในขณะที่ i จะถูกเพิ่มค่าเป็น $i+1$ เช่นกัน ดังนั้น ประโยชน์ $x = x+i$ จะทำงานทั้งหมด 10 รอบตามเงื่อนไข $i \leq 10$



รูปที่ 4.8: โพลว์ชาร์ตการทำงานของประโยชน์การวนรอบชนิด FOR

เราเรียก i ว่าเป็นตัวแปรนับรอบ (Index) เนื่องจากเป็นตัวแปรที่ใช้นับเลขรอบ ประโยชน์ $i=1$ เรียกว่า Index Initialization คือ การตั้งค่าเริ่มต้นให้กับตัวแปรนับรอบ ซึ่งเป็นองค์ประกอบหนึ่งของประโยชน์ for ประโยชน์ $x=x+i$ เรียกว่า Loop Body ตามโฟล์ชาร์ตในรูปที่ 4.8 ประโยชน์เงื่อนไข $i \leq 10$ เรียกว่า Condition เพื่อตรวจสอบว่าเป็นจริงหรือเท็จ เมื่อ i มีค่าเท่ากับ 1 ถึง 10 เงื่อนไขจะให้ผลลัพธ์เป็นจริง (True) เมื่อ i มีค่าเท่ากับ 11 ผลลัพธ์จะกลายเป็นเท็จ (False) และจะไม่เกิดการวนรอบ เพื่อทำงานประโยชน์ที่ต่อจาก ประโยชน์ for ประโยชน์ $i=i+1$ เรียกว่า Index Update คือ การปรับเปลี่ยนค่าตัวแปรนับรอบให้สอดคล้อง กับเงื่อนไข ตามที่กล่าวมาข้างต้น

ตารางที่ 4.6: ตัวอย่างโปรแกรมประโยชน์ควบคุม For ตามสมการที่ (4.1)

บรรทัดที่	เลขบล	คำสั่ง	คอมเมนต์
1		MOV R0, 0	@ Initialize R0=0
2		MOV R1, 1	@ 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

ผู้อ่านสามารถทำความเข้าใจการทำงานได้จากการคำอธิบายภายใต้คอลัมน์ คอมเมนต์ และคำอธิบายตามหมายเลขอรรถัด ดังนี้

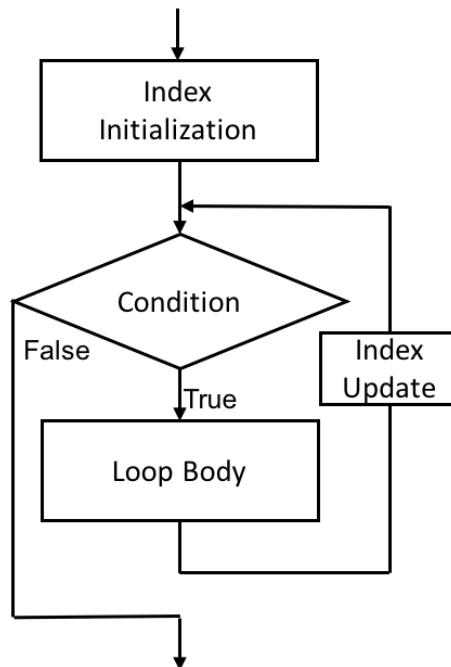
- คือ การตั้งค่าเริ่มต้นให้กับ $R0 = 0$
- คือ การตั้งค่าเริ่มต้นให้กับ $R1 = 1$
- คือ คำสั่งที่ขึ้นต้นด้วยเลขบล for: เพื่อทำการเปรียบเทียบ $R1$ กับค่าคงที่ 10_{10}
- หากเงื่อนไข Greater Than (GT) เป็นจริง ชีพีย์จะกระโดดไปทำงานคำสั่งที่ขึ้นต้นด้วย end หากไม่ เป็นจริง ชีพีย์จะทำงานประโยชน์ที่ 5 ต่อไป
- คือ การคำนวณค่า $R0 + R1$ ไปบรรจุใน $R0$
- คือ การคำนวณค่า $R1 + 1$ ไปบรรจุใน $R1$
- คือ การบังคับให้ชีพีย์กระโดดไปทำงานคำสั่งที่ขึ้นต้นด้วยเลขบล for
- คือ คำสั่งที่ขึ้นต้นด้วยเลขบล end

ตัวอย่างนี้เป็นการวนรอบชนิด For ในรูปที่ 4.8 การตั้งค่าเริ่มต้น (Initialization) ให้กับตัวแปรนับรอบ (Index) หลังจากนั้น ชีพีย์จะตรวจสอบเงื่อนไข Condition ของการวนรอบ หากเป็นจริง ชีพีย์จะทำงานตามคำสั่งจำนวนหนึ่ง ที่เรียกว่า Body เมื่อแล้วเสร็จ ชีพีย์จะเพิ่ม (Increment) หรือ ลด (Decrement)

ค่าตัวแปรลูป (Loop Index) และชีพีย์จะกลับไปทำงานในบรรทัดที่มีประযุคเงื่อนไขซ้ำแล้วซ้ำอีก จนเงื่อนไขเป็นเท็จ (False) การวนรอบชนิด FOR จะสิ้นสุดการทำงาน และวิธีการทำงานไปคำสั่งอื่นภายนอก

4.7.4 การวนรอบชนิด WHILE

การวนรอบชนิด WHILE คล้ายกับการวนรอบชนิด FOR คือ การวนรอบจะเกิดขึ้นอยู่กับเงื่อนไขของประยุค WHILE ว่าผลลัพธ์ที่ได้จะเป็นจริงหรือเท็จ หากจริง ชีพีย์จะปฏิบัติตามประยุคคำสั่งที่โปรแกรมเมอร์ต้องการ และวนกลับไปตรวจสอบเงื่อนไขอีก หากเท็จ ชีพีย์จะทำคำสั่งอื่น ๆ ต่อไป ในรูปที่ 4.9



รูปที่ 4.9: โฟล์ชาร์ตการทำงานของประยุคการวนรอบชนิด WHILE

สมการที่ (4.1) สามารถเขียนด้วยการวนรอบชนิด WHILE ดังนี้

```
i=1;
x=0;

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

ตัวอย่างนี้ทำงานเหมือนกับประยุค FOR เกือบทุกประการ ความแตกต่าง คือ การตั้งค่าตัวแปรเริ่มต้น การตรวจสอบเงื่อนไขในวงเล็บของประยุค WHILE และการปรับเปลี่ยนตัวแปร i ภายในลูป ดังนั้นตัวอย่างการเขียนโปรแกรมในตารางที่ 4.7 จึงมีความใกล้เคียงกันมากกับประยุค FOR

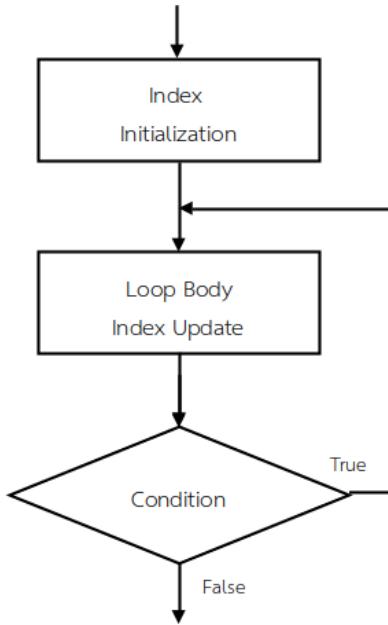
ตารางที่ 4.7: ตัวอย่างโปรแกรมตามประโยคุนรอบซึ่ด WHILE ตามสมการที่ (4.1)

บรรทัดที่	เลbel	คำสั่ง	คอมเมนต์
1		MOV R0, 0	@ Initialize R0=0
2		MOV R1, 1	@ 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 คอมเมนต์ และคำอธิบายตามหมายเลขอรรถัด 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 ชิปปี้จะกระโดดไปทำงานคำสั่งที่ขึ้นต้นด้วยเลเบล end หากไม่เป็นจริง ชิปปี้จะทำงานประโยคที่ 5 ต่อไป
5. คือ การคำนวณค่า $R0 + R1$ ไปบรรจุใน $R0$
6. คือ การคำนวณค่า $R1 + 1$ ไปบรรจุใน $R1$
7. คือ การบังคับให้ชิปปี้กระโดดไปทำงานคำสั่งที่ขึ้นต้นด้วยเลเบล for
8. คือ คำสั่งที่ขึ้นต้นด้วยเลเบล end

4.7.5 การวนรอบชนิด DO-WHILE



รูปที่ 4.10: โฟล์ชาร์ตการทำงานของประโยคการวนรอบชนิด DO-WHILE

การวนรอบอีกชนิด คือ การวนรอบ DO-WHILE มีโครงสร้างการทำงานตามโฟล์ชาร์ตในรูปที่ 4.10 การวนรอบ DO-WHILE แตกต่างกับการวนรอบ WHILE คือ ซีพียูจะปฏิบัติตามประโยคคำสั่งในลูป (Loop Body) อย่างน้อย 1 รอบ หลังจากนั้น ซีพียูจะตรวจสอบเงื่อนไขของประโยค WHILE ว่าผลลัพธ์ที่ได้จะเป็นจริงหรือเท็จ หากจริง ซีพียูจะวนกลับไปปฏิบัติตามประโยคอีก แล้วจึงตรวจสอบเงื่อนไข หากเท็จ ซีพียูจะทำคำสั่งอื่น ๆ ต่อไป

สมการที่ (4.1) สามารถเขียนด้วยการวนรอบชนิด DO-WHILE ดังนี้

```

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

และสามารถเขียนด้วยภาษาแอลกอริทึมได้ในตารางที่ 4.8

ผู้อ่านสามารถทำความเข้าใจการทำงานได้จากการคำอธิบายภายใต้คอลัมน์ คอมเมนต์ และคำอธิบายตามหมายเลขอรรถทัศ ดังนี้

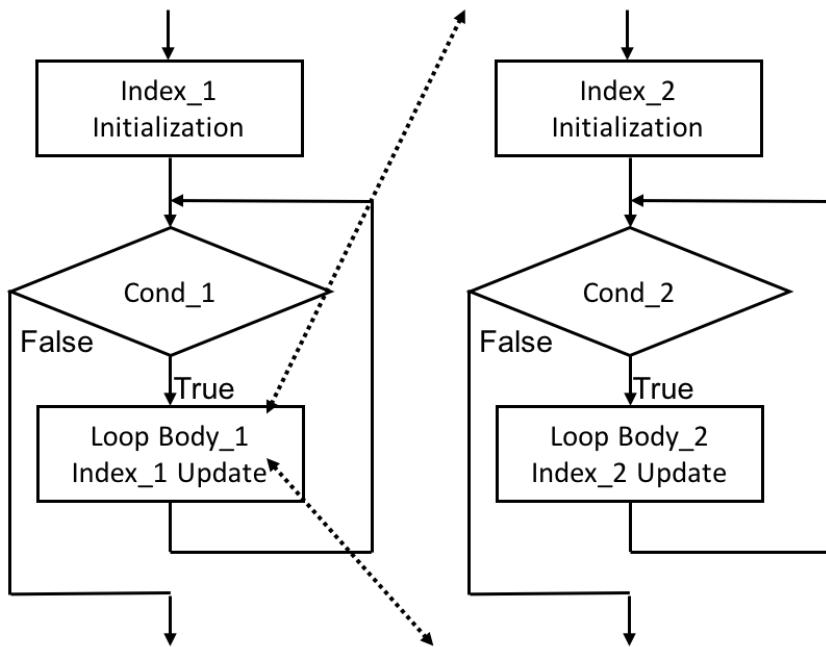
1. คือ การตั้งค่าเริ่มต้นให้กับ $R0 = 0$
2. คือ การตั้งค่าเริ่มต้นให้กับ $R1 = 0$

ตารางที่ 4.8: ตัวอย่างโปรแกรมตามประโยคุณรอบชนิด DO-WHILE ตามสมการที่ (4.1)

บรรทัดที่	เลbel	คำสั่ง	คอมเมนต์
1		MOV R0, 0	@ Initialize R0=0
2		MOV R1, 1	@ Initialize R1=1
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		...	@

3. คือ คำสั่งที่ขึ้นต้นด้วยเลเบล do เพื่อคำนวณค่า $R0 + R1$ ไปบรรจุใน R0
4. คือ การคำนวณค่า $R1 + 1$ ไปบรรจุใน R1
5. คือ การเปรียบเทียบ R1 กับค่าคงที่ 10_{10}
6. หากเงื่อนไข Less Than or Equal (LE) เป็นจริง ชีพีย์จะกระโดดไปทำงานคำสั่งที่ขึ้นต้นด้วย do หากไม่เป็นจริง ชีพีย์จะทำงานประโยคที่ 7 ต่อไป
7. คือ คำสั่งที่ขึ้นต้นด้วยเลเบล 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) แล้ววนกลับไปข้อที่ ii
 - หากเป็นเท็จ (False) โปรแกรมจะข้ามไปทำงานที่ประโยคถัดไปโดยไม่ทำงานลูปชั้นในแม้แต่รอบเดียว
 - iii. โปรแกรมจะเพิ่ม (Increment) หรือ ลด (Decrement) ตัวแปรนับรอบที่ 1 (Index_1) แล้ววนกลับไปข้อที่ 2
 - (b) หากเป็นเท็จ (False) โปรแกรมจะข้ามไปทำงานที่ประโยคถัดไปโดยไม่ทำงานลูปชั้นนอกแม้แต่รอบเดียว

4.8 การเรียกใช้ฟังก์ชัน (Function Call)

การเรียกใช้ฟังก์ชันในภาษาแอสเซมบลีคล้ายกับ 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

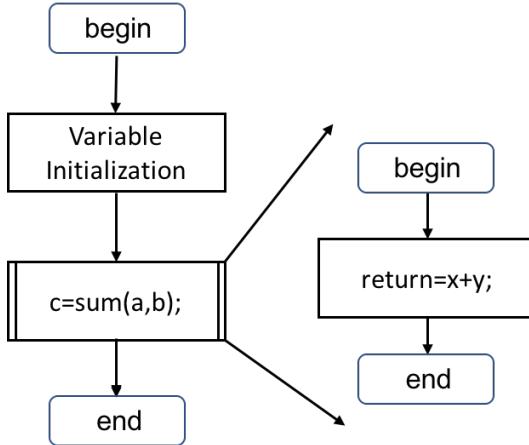
```
int main() {
    int a, b, c;
    ...
    c = sum(a, b);
    ...
    return 0;
}

int sum(int x, int y) {
    return x+y;
}
```

รูปที่ 4.12 แสดงโฟล์ชาร์ตการทำงานของการเรียกใช้ฟังก์ชันชื่อ sum(a, b) ในตัวอย่างที่ 4.8.1 กลไกนี้จะเกิดขึ้นเมื่อมีการเรียกใช้ฟังก์ชันมาตรฐานในโปรแกรมภาษา C/C++ ซึ่งสามารถเรียกใช้ข้ามแพลตฟอร์มได้ เพราะทำงานเป็นมอดูล นักพัฒนาสามารถนำฟังก์ชันที่พัฒนาเองไปเผยแพร่ ยกตัวอย่างเช่น ฟังก์ชัน printf ในไฟล์ stdio.h ซึ่งย่อมาจากคำว่า Standard I/O ซึ่งเป็นฟังก์ชันในไลบรารีหลักของภาษา C ชื่อ glibc ฟังก์ชันมาตรฐานเหล่านี้สามารถนำมายังรวมกับไฟล์อื่นๆ เช่น ไฟล์ .c หรือ .h ตามหลักการในรูปที่ 3.11

4.8.2 คำสั่งเรียกใช้ฟังก์ชันในภาษาแอสเซมบลี

ภาษาแอสเซมบลีของชีพิญต่าง ๆ มีศักยภาพที่ใกล้เคียงกับภาษาสูง เช่น C/C++ ที่โปรแกรมเมอร์สามารถสร้างฟังก์ชันของตนเองได้ คำสั่งภาษาแอสเซมบลีของชีพิญ ARM ที่รองรับการเรียกใช้ฟังก์ชัน คือ



รูปที่ 4.12: โฟล์ชาร์ตการทำงานของประโยชน์เรียกใช้ฟังก์ชัน $\text{sum}(a, b)$ ในตัวอย่างที่ 4.8.1

รูปแบบ	ความหมาย
BL func_name	<ul style="list-style-type: none"> $\cdot LR = PC+4$ $\cdot PC = \text{address(func_name)}$ \cdotเปลี่ยนค่าของ PC ไปยังเลbelชื่อฟังก์ชัน func_name
BX LR	$PC = LR$ เปลี่ยนค่าของ PC ไปยังแอดдресที่เก็บในรีจิสเตอร์ LR

คำสั่ง BL ย่อมาจากคำว่า Branch and Link คำว่า func_name คือ เลbelชื่อ ทำหน้าที่เป็นชื่อฟังก์ชันที่ต้องการเรียกใช้ ในระหว่างที่ซีพียูปฏิบัติตามคำสั่ง BL func_name ซีพียูจะทำงานย่อย 2 ขั้นดังต่อไปนี้

- $LR = PC+4$ เพื่อบันทึกค่า address ของคำสั่งถัดไปที่ติดกับคำสั่ง BL ไว้ในรีจิสเตอร์ LR และ
- $PC = \text{address(func_name)}$ เพื่อเปลี่ยนแปลงค่าแอดdressในรีจิสเตอร์ PC ไปเป็นแอดdressของ เลbel func_name

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

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

ผู้อ่านสามารถทำความเข้าใจการเรียกใช้ฟังก์ชันจากตัวอย่างต่อไปนี้

ตัวอย่าง 4.9: ตัวอย่างโปรแกรมเรียกใช้ฟังก์ชันภาษาแอสเซมบลีที่คล้ายกับตัวอย่างที่ 4.8.1

บรรทัดที่	เลbel	คำสั่ง	คอมเมนต์
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

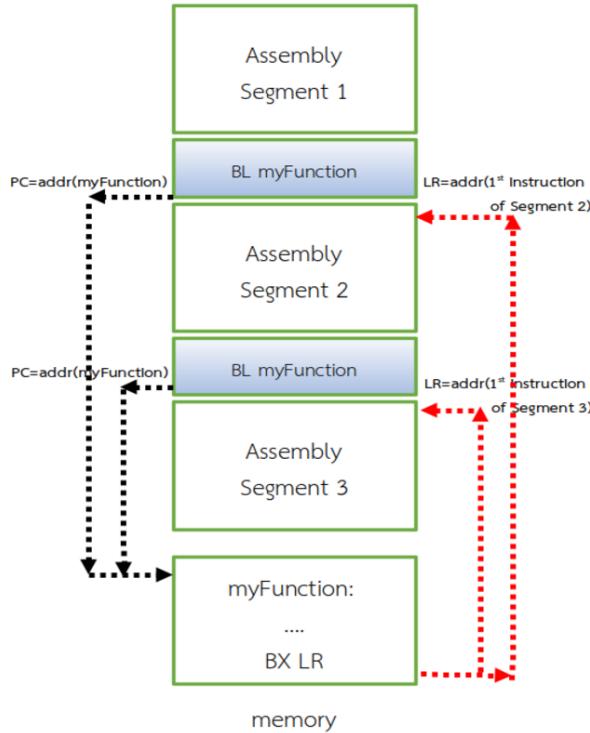
การทำงานของตัวอย่าง จะเริ่มที่ฟังก์ชัน main ตามปกติ ในบรรทัดที่ 3 จะเป็นการส่งค่าพารามิเตอร์ ของตัวแปร a ในรีจิสเตอร์ R0 และตัวแปร b ด้วยรีจิสเตอร์ R1 ไปยังฟังก์ชัน sum คล้ายกับโฟล์ชาร์ ทำการเรียกใช้ฟังก์ชันในรูปที่ 4.12 และส่งค่าผลลัพธ์กลับทาง R0 ประโยชน์ของการสร้างฟังก์ชัน คือ การเรียกใช้ฟังก์ชันข้าแล้วข้าอีก 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 ประโยชน์ค่าลั่งอื่น ๆ ที่ไม่เกี่ยวข้องกับตัวอย่าง



รูปที่ 4.13: การทำงานของคำสั่ง `BL myFunction` เมื่อมีการเรียกใช้ 2 ครั้งจากฟังก์ชัน `main` ในหน่วยความจำคล้ายกับตัวอย่างที่ 4.8.2 หมายเหตุ ... หมายถึงประโยชน์คำสั่งอื่น ๆ ที่ไม่เกี่ยวข้องกับตัวอย่าง

ผู้อ่านสามารถทำความเข้าใจการการทำงานของโปรแกรมในตัวอย่างที่ 4.8.2 จากรูปที่ 4.13 การทำงานของฟังก์ชัน `main()` เริ่มต้นตามปกติ เรียกใช้ฟังก์ชันชื่อ `myFunction()` สองครั้ง ซึ่งจะย้ายการทำงานไปใน `myFunction` จนสิ้นสุดแล้วรีเทิร์น 0 กลับมาที่ประโยชน์ถัดไป ซึ่งฟังก์ชัน `main()` ทำงานแล้วเสร็จ ซึ่งจะรีเทิร์น 0 กลับไปหาฟังก์ชันที่เรียกใช้ `main()` เช่นกัน

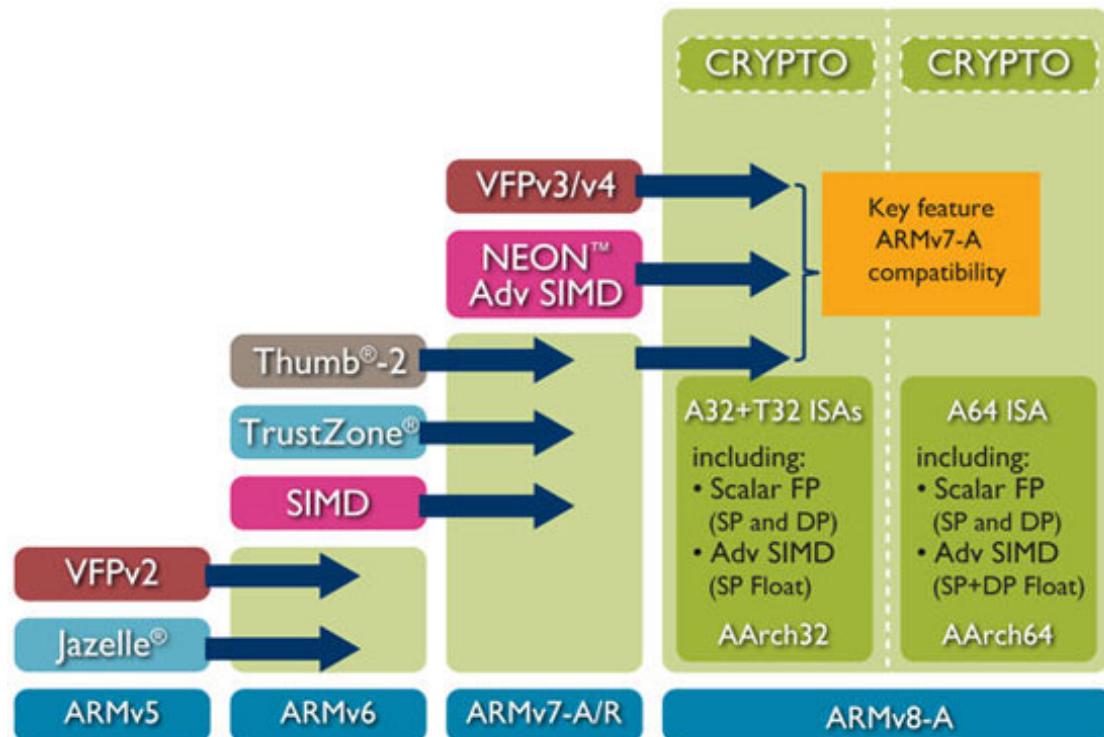
ในภาษาแอสเซมบลี คำสั่ง `BL myFunction` จะเก็บค่าแอดเดรสต่อจาก `BL myFunction` ในรีจิสเตอร์ `LR` โปรดสังเกตคำอธิบายทางขวา นั่นคือ $LR = PC + 4 =$ แอดเดรสของคำสั่งแรกใน Segment 2 และจะเปลี่ยนค่า `PC` เป็นแอดเดรสของฟังก์ชัน `myFunction` $PC = \text{address}(\text{myFunction})$ โปรดสังเกตเส้นประด้านซ้าย เมื่อทำงานฟังก์ชัน `myFunction` แล้วเสร็จ คำสั่ง `BX LR` จะเปลี่ยนค่า `PC` เป็นค่าของรีจิสเตอร์ `LR` ($PC = LR$) โปรดสังเกตเส้นประด้านขวาซึ่งเป็นแอดเดรสของคำสั่งถัดไป

หลังจากนั้น ฟังก์ชัน `main` จะเรียกคำสั่ง `BL myFunction` รอบที่สอง $LR = PC + 4 =$ แอดเดรสของคำสั่งแรกใน Segment 3 และ $PC = \text{address}(\text{myFunction})$ โปรดสังเกตเส้นประด้านซ้าย หลังจากที่ทำงาน `myFunction` แล้วเสร็จ ประโยชน์ `BX LR` จะรีเทิร์นกลับ ณ ตำแหน่งถัดไปภายในฟังก์ชัน `main` ($PC = LR$) โปรดสังเกตเส้นประด้านขวา หมายเหตุ `addr` ย่อมาจากคำว่า `address`

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

4.9 คำสั่งภาษาแอสเซมบลีอื่น ๆ ในชีพีย์ ARM

วิวัฒนาการของชุดคำสั่งของชีพีย์ ARM จากเริ่มก่อตั้งจนถึงปัจจุบันมีความเข้มข้นอย่างกับอุปกรณ์ที่ใช้ชิปเหล่านี้โดยตรง จะสังเกตได้จากความหลากหลายของอุปกรณ์ที่ใช้งานชิป ARM จากสมาร์ตการ์ด (Smart Card) ในบัตรเครดิตและบัตรเดบิต ไปจนถึงชูเปอร์คอมพิวเตอร์ แต่ชิปเหล่านี้มีเป้าหมายร่วมกันคือ การประหยัดพลังงานไฟฟ้า



รูปที่ 4.14: การควบรวมชุดคำสั่งภาษาแอสเซมบลีของชีพีย์ ARM ตั้งแต่เวอร์ชัน 5 ถึงเวอร์ชัน 8A โดยเวอร์ชันใหม่จะรวมคำสั่งของเวอร์ชันเก่าเพื่อรองรับการทำงานซอฟต์แวร์เดิม ที่มา: arm.com

ในอดีตที่ผ่านมาบริษัท ARM ได้พัฒนาภาษาแอสเซมบลีออกมาตั้งแต่เวอร์ชัน 1 มาเรื่อย ๆ โดยในรูปที่ 4.14 ภาษาแอสเซมบลีเวอร์ชัน 5 (ARMv5) และเวอร์ชัน 6 (ARMv6) ที่ผ่านมาในอดีตถูกควบรวมเป็นภาษาแอสเซมบลีเวอร์ชัน 7 (ARMv7) พร้อมชุดคำสั่งใหม่เพิ่มเติม ซึ่งรองรับได้เพียงระบบปฏิบัติการเวอร์ชัน 32 บิตเท่านั้น โดยแบ่งเป็น ARMv7-A สำหรับชีพีย์ ARM Cortex-A, ARMv7-M สำหรับชีพีย์ ARM Cortex-M และ ARMv7-R สำหรับชีพีย์ ARM Cortex-R และล่าสุดภาษาแอสเซมบลีเวอร์ชัน 8 (ARMv8-A) สำหรับชีพีย์ ARM Cortex-A รุ่นปัจจุบัน ซึ่งจะรองรับระบบปฏิบัติการ เวอร์ชัน 64 และ 32 บิตด้วย

นอกจากเนื้อหาภาษาแอสเซมบลีเวอร์ชันต่าง ๆ ที่กล่าวมาข้างต้น จุดเด่นของชีพีย์ ARM ยังมีชุดคำสั่งภาษาแอสเซมบลีของชีพีย์ ARM ที่มีความยาว 16 และ 32 บิต ซึ่ง ThUMB, ThUMB-2, T32 ซึ่งนิยมใช้สำหรับอุปกรณ์ที่มีหน่วยความจำความจุน้อย ตั้งทุนต่ำ รายละเอียดเพิ่มเติมใน [wikipedia](https://en.wikipedia.org/wiki/ARM_architecture)

คำสั่งที่กล่าวมาข้างต้นในบทนี้หมายความว่าสำหรับการคำนวณข้อมูลเชิงเดี่ยวตามที่ได้อธิบายไปแล้ว ARM ได้ออกแบบชุดคำสั่งภาษาแอสเซมบลีสำหรับประมวลผลข้อมูลที่เรียงตัวเนื่องกันคล้ายกับอาร์เรย์เรียกว่า เวกเตอร์ (Vector) ได้แก่ คำสั่งชนิด Vector Floating Point version 3/4 (VFPv3/v4), Sin-

gle Instruction Multiple Data (SIMD) (อ่านว่า ซิมดี) และ NEON Advanced SIMD (Adv SIMD) ซึ่งได้อธิบายในหัวข้อที่ 4.1 เมماะ กับข้อมูลมัลติมีเดีย เช่น ข้อมูลภาพ และข้อมูลเสียง รวมถึงเกมประเภทกราฟิก เพื่อสามารถคำนวณข้อมูลแบบขนาน (Parallel Computing) ตามเนื้อหาในบทที่ 8 เช่น ชูเปอร์คอมพิวเตอร์ นอกจากนี้ ผู้อ่านสามารถค้นควารายละเอียดอื่น ๆ เพิ่มเติมเกี่ยวกับสถาปัตยกรรมชุดคำสั่งของชิปีย ARM ได้ที่ [wikipedia](#)

4.10 สถาปัตยกรรมชุดคำสั่งภาษาแอสเซมบลีในชิปียท์ว่าโลก

เราสามารถแบ่งสถาปัตยกรรมของคำสั่งภาษาแอสเซมบลีในชิปียต่าง ๆ ทั่วโลกออกได้ดังนี้

- **สถาปัตยกรรมโหลด/สโตร์ (Load/Store Architecture)** ARM และชิปียของ RISC-V ออกแบบคำสั่งภาษาแอสเซมบลีตามหลักการ RISC (Reduced Instruction Set Computer) ข้อมูลเพิ่มเติมที่ [wikipedia](#) ทำให้ชิปียทำงานที่ความถี่สัญญาณคลื่อกสูงกว่าและบริโภคพลังงานน้อยกว่า เป็นต้น
- **สถาปัตยกรรม x86 (x86 Architecture)** ในภาษาแอสเซมบลีของ Intel 80x86 หรือเรียกย่อ ๆ ว่า x86 ซึ่งบางคำสั่งใช้การอ่านค่าจากหน่วยความจำเพื่อประมวลผล ทำให้การประมวลซับซ้อนและล่าช้า หรือที่เรียกว่า CISC (Complex Instruction Set Computer) ข้อมูลเพิ่มเติมที่ [wikipedia](#)
- **สถาปัตยกรรมแอคคิวมูเลเตอร์ (Accumulator Architecture)** ? ริจิสเตอร์หลักเพื่อเก็บผลลัพธ์เรียกว่า แอคคิวมูเลเตอร์ (Accumulator Register) Acc = Acc + Register เพื่อประหยัดชาร์ดแวร์ในอดีต ข้อมูลเพิ่มเติมที่ [wikipedia](#)
- **สแต็กแมชชีน (Stack Machine)** ได้แก่ ภาษาจาวาไบต์โค้ด (Java Bytecode) จัดเป็นคำสั่งภาษาแอสเซมบลีเมื่อแปลจากซอฟต์แวร์ Java เทคโนโลยีที่น่าสนใจของชิปีย ARM เรียกว่า Jazelle สามารถรองรับ การประมวลผลคำสั่ง Java Bytecode ด้วยชาร์ดแวร์ ทำให้ชิปบริโภคพลังงานต่ำ โดยเฉพาะโทรศัพท์สมาร์ตโฟนที่ใช้ระบบปฏิบัติการ Android ซึ่งซอฟต์แวร์ส่วนใหญ่ใช้ภาษา Java พัฒนา รายละเอียดเพิ่มเติมใน [wikipedia](#)

4.11 สรุปท้ายบท

คำสั่งภาษาแอสเซมบลีของชิปีย ARM จัดเป็นสถาปัตยกรรมโหลด/สโตร์ มีลักษณะเด่น เมื่อเปรียบเทียบกับสถาปัตยกรรมอื่น ๆ คือ การโหลดหรืออ่านค่าของตัวแปรมาพักในริจิสเตอร์ก่อน แล้วจึงใช้คำสั่งคำนวณค่าของตัวแปรเหล่านั้นให้ได้ผลลัพธ์เบื้องต้นแล้วเก็บลงในริจิสเตอร์ ผลลัพธ์สุดท้ายที่ได้จากการคำนวณซึ่งอยู่ในริจิสเตอร์จะถูกสำเนาค่า เพื่อนำไปสโตร์หรือเขียนหรือเก็บในตัวแปรภายในหน่วยความจำหลัก นอกจากนี้ ชิปีย ARM ยังมีคำสั่งที่เป็นเอกลักษณ์ เช่น คำสั่งเลื่อนบิตย่อยภายในคำสั่งคณิตศาสตร์ การตรวจสอบเงื่อนไขก่อนการปฏิบัติตามคำสั่งนั้น เป็นต้น ทำให้ชิปียที่ผลิตตามการออกแบบของชิปีย ARM บริโภคพลังงานน้อย จึงได้รับความนิยมในอุปกรณ์เคลื่อนที่มาเป็นระยะเวลาต่อเนื่อง

4.12 คำถ้ามท้ายบท

1. รีจิสเตอร์ PC (Program Counter) ทำหน้าที่อะไร และเกี่ยวข้องกับการรันของโปรแกรมอย่างไร
2. คำสั่ง POP สำหรับอ่านค่าข้อมูลออกจากสแต็กเชิ้กเมนต์ ทำงานอย่างไร
3. คำสั่ง PUSH สำหรับเก็บค่าข้อมูลลงในสแต็กเชิ้กเมนต์ ทำงานอย่างไร
4. คำสั่ง POP และ PUSH ทำงานร่วมกับรีจิสเตอร์ Stack Pointer (SP) อย่างไร
5. จงเขียนฟังก์ชันเพื่อคำนวนหาค่าสัมบูรณ์ของ R0-R1 แล้วรีเทิร์นค่าในค่า R0
6. จงเขียนโปรแกรมลูปวนรอบ 2 ชั้น โดยใช้การวนรอบชนิด
 - While
 - Do-While
7. การทำงานของฟังก์ชันเรียกซ้ำ (Recursive Function Call)
 - จงวัดเวลาการทำงานเพื่อแสดงการทำงานกรณีปกติและกรณีหยุดเรียกตัวเอง
 - จงวัดรูปการทำงานประกอบด้วยหน่วยความจำคล้ายกับรูปที่ [4.13](#)

บทที่ 5

ลำดับชั้นของหน่วยความจำ (Hierarchy of Memory)

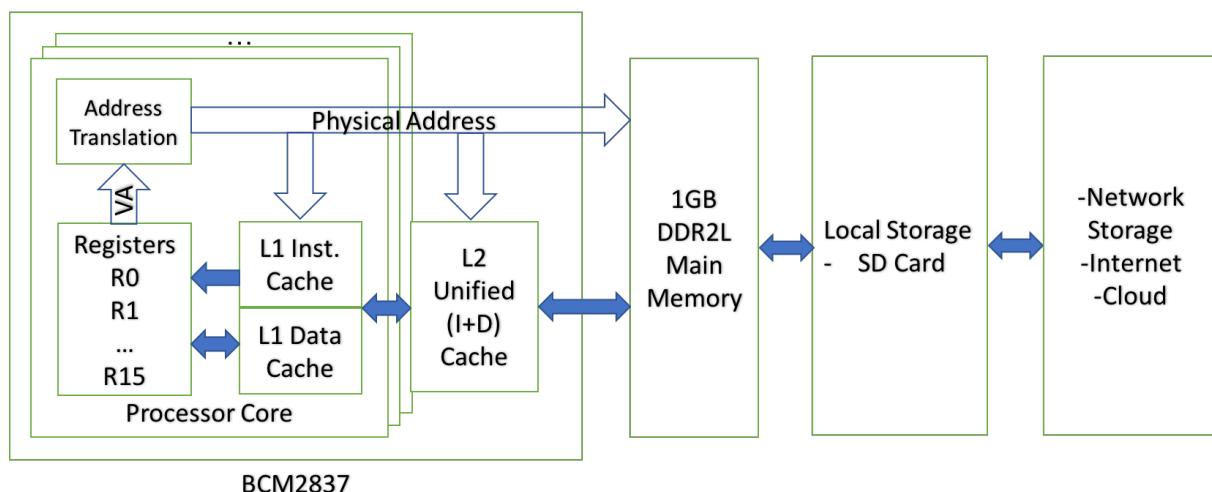
ลำดับชั้นของหน่วยความจำ (Hierarchy of Memory) อาศัยการทำงานร่วมกันของหน่วยความจำหลายชนิด ที่มีความจุ (Capacity) ที่แตกต่างกันมาทำงานร่วมกัน เพื่อผสานความเร็วของหน่วยความจำแต่ละชนิดเข้าด้วยกัน และช่วยประหยัดพื้นที่โดยรวมของระบบ บทนี้มีวัตถุประสงค์ดังนี้

- เพื่อให้เข้าใจถึงลำดับชั้นและการทำงานร่วมกันระหว่างหน่วยความจำชนิดต่าง ๆ ได้แก่ รีจิสเตอร์ แคชลำดับชั้นต่าง ๆ หน่วยความจำหลัก และอุปกรณ์เก็บรักษาข้อมูล
- เพื่อให้เข้าใจถึงการทำงานร่วมกันของเวอร์ชัล เมม莫รีชนิด เพจ (Paging Virtual Memory) ของหน่วยประมวลผลและระบบปฏิบัติการในทางทฤษฎี
- เพื่อให้เข้าใจถึงการทำงานของเวอร์ชัล เมม莫รีในระบบลินกุซ เวอร์ชัน 32 บิตสำหรับบอร์ด Pi3/Pi4
- เพื่อให้เข้าใจถึงการทำงานของแคชชนิด Direct Map (DM) และ Set Associative (SA) ซึ่งนิยมใช้ในชิปซีพียู
- เพื่อให้เข้าใจถึงความแตกต่างระหว่างหน่วยความจำสแตติก แรม และ SDRAM ในด้านโครงสร้างภายใน การทำงาน และประสิทธิภาพ

บทนี้จะอธิบายลำดับชั้นของหน่วยความจำว่า เหตุใดคอมพิวเตอร์จึงต้องประกอบด้วยหน่วยความจำหลายชนิด ซึ่งต่อเนื่องจากหัวข้อที่ 3.1.2 เนื้อหาในบทนี้อาศัยการทดลองที่ 4 ภาคผนวก D การใช้งานระบบปฏิบัติการยูนิฟอร์ม ซึ่งจะทำให้ผู้อ่านเข้าใจหลักการและรายละเอียดมากขึ้น

5.1 ลำดับชั้นของหน่วยความจำของบอร์ด Pi3/Pi4

หน่วยความจำมีหลายชนิดตามหลักการออกแบบ ความซับซ้อน/เทคโนโลยีการผลิต ความจุของหน่วยความจำแต่ละชนิดมีความหลากหลาย ความเร็วหรือสมรรถนะในการอ่านหรือเขียนข้อมูล และ ต้นทุนต่อความจุหนึ่งบิต ทำให้การจัดวางมีผลต่อประสิทธิภาพโดยรวม หน่วยความจำบางชนิดสามารถบรรจุในชิป (On Chip) เดียว กับซีพียู ในขณะที่บางชนิดจำเป็นต้องอยู่นอกชิป (Off Chip) ตำแหน่งนี้จะอธิบายเทคโนโลยี และ การทำงานหน่วยความจำชนิดต่าง ๆ ของบอร์ด Pi3/Pi4 เป็นกรณีศึกษาลำดับชั้น และการจัดวางหน่วยความจำชนิดต่าง ๆ



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

บอร์ด Pi3/Pi4 ประกอบด้วยหน่วยความจำที่หลากหลาย บางส่วนอยู่ในชิปเดียว กับซีพียู (On Chip) และบางส่วนอยู่นอกชิป (Off Chip) ในรูปที่ 5.1 หน่วยความจำเหล่านี้อยู่ในลำดับชั้นต่าง ๆ ทั้งในลิ๊บและไมโคร 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 ซึ่งใช้หน่วยความจำชนิดสแตติกแรม (Static RAM: SRAM) ทำให้สามารถเข้าถึงข้อมูลได้ภายในเวลาสั้น ๆ เพียง 0.5-1 นาโนวินาที (Nano Second)
- **แคชลำดับที่ 1 (Level 1)** ซึ่งแบ่งเป็นแคชคำสั่ง หรือ Instruction Cache และแคชข้อมูล หรือ Data Cache ซึ่งใช้หน่วยความจำสแตติกแรม มีขนาดประมาณ 16-64 KiB แต่ละแกนประมาณ 1/8 ของซีพียู ARM Cortex A53/A72 จะมีแคชทั้งสองชั้นอยู่ด้วยกัน
- **แคชลำดับถัดไป** ซึ่งใช้หน่วยความจำสแตติกแรม เช่น กัน โดยมีความจุ 64 KiB (kibibyte) ขึ้นไป จนถึงหลักหลายสิบเมบิไบต์ (MebiByte: MiB) แคชลำดับที่ 2 ทำหน้าที่พกเก็บห้องคำสั่งและข้อมูล (Unified) คล้ายหน่วยความจำขนาดเล็กของแคชคำสั่งและแคชข้อมูลลำดับที่ 1 ชิปซีพียูจากผู้ผลิต

บางรายมีแคชลำดับที่ 3 ที่มีความจุมากขึ้น ทำหน้าที่คล้ายหน่วยความจำภายในภาพของแคชลำดับที่ 2 ซึ่งต้นทุนในการผลิตชิปเหล่านี้แปรผันตรงกับความจุของแคชเช่นกัน

- **หน่วยความจำหลัก (Main Memory)** หรือบางตำราใช้คำว่า หน่วยความจำภายในภาพ (Physical Memory) หรือ หน่วยความจำปฐมภูมิ (Primary Memory) ซึ่งนิยมใช้เทคโนโลยีหน่วยความจำชนิดไดนามิกแรม (Dynamic RAM: SDRAM) หน่วยความจำหลักมีขนาด 512 เมบิไบต์ขึ้นไป จนถึงหลายสิบกิกิไบต์ (GiB) (GigaByte: GB) ขึ้นอยู่กับชนิดและต้นทุนของคอมพิวเตอร์นั้น ๆ
- อุปกรณ์เก็บรักษาข้อมูล (Storage) หรือเรียกว่า หน่วยความจำทุดิยภูมิ (Secondary Memory) ซึ่งมีทางเลือกจากหลายเทคโนโลยี เพื่อเก็บรักษาข้อมูลไม่ให้สูญหาย ถึงแม้ผู้ใช้จะปิดเครื่องหรือแหล่งจ่ายไฟ พลังงานหมด เช่น คอมพิวเตอร์ พกพา ซึ่งต้องอาศัยพลังงานจากแบตเตอรี่ ชนิดของอุปกรณ์เก็บรักษาข้อมูล โดยจะกล่าวรายละเอียดเพิ่มเติมในบทที่ 7
 - การ์ดหน่วยความจำ SD (Secure Digital) ซึ่งสร้างจากเทคโนโลยีหน่วยความจำแฟลช (Flash Memory) เช่นเดียวกับโซลิดสเตทไดร์ฟ และกล่าวถึงในหัวข้อที่ 3.1.4 ทำให้มีความจุมากระดับ 16 กิกิไบต์ (GiB) ขึ้นไป หน่วยความจำแฟลชความเร็วสูงกว่า เมื่อเทียบกับฮาร์ดดิสก์ หน่วยความจำ SD สามารถพกพาได้สะดวก เพราะการ์ดมีขนาดเล็กและน้ำหนักเบา จึงนิยมใช้บันทึกข้อมูลในกล้องดิจิทัล โทรศัพท์สมาร์ตโฟน และอื่นๆ
 - โซลิดสเตทไดร์ฟ (Solid State Drive: SSD) ซึ่งใช้เทคโนโลยีหน่วยความจำแฟลช (Flash Memory) มีขนาด 128 กิกิไบต์ (GiB) ขึ้นไป จนถึงหลายเทราไบต์ และคาดว่าจะทดแทนฮาร์ดดิสก์ในปัจจุบันและอนาคต
 - ฮาร์ดดิสก์ (Hard Disk) ซึ่งใช้เทคโนโลยีหน่วยความจำแผ่นแม่เหล็ก (Magnetic Disc) หมุนด้วยความเร็วสูงประมาณ 5,400-10,000 รอบต่อนาที ขึ้นกับราคาและความจุซึ่งมีขนาดหลายร้อยกิกิไบต์ (GiB) จนถึงหลายเทราไบต์ (TeraByte: TB) และสามารถเติบโตต่อเนื่องจนถึงเพتل่าไบต์ (PetaByte)
 - อุปกรณ์เก็บรักษาข้อมูลผ่านเครือข่าย (Network Storage) ซึ่งมีรูปแบบต่าง ๆ เช่น Network Attached Storage (NAS), Storage Area Network (SAN), Cloud Storage เป็นต้น

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

เครื่องคอมพิวเตอร์ทั้งหมดจะต้องมีอุปกรณ์เก็บรักษาข้อมูล หรือ Non-Volatile Memory ในลำดับชั้นต่อไป เพื่อเก็บรักษาไฟล์โปรแกรมและไฟล์ข้อมูลมิให้สูญหาย รายละเอียดเพิ่มเติมในบทที่ 7 ในทำนองเดียวกัน เครื่องคอมพิวเตอร์ที่ต้องทำงานตลอดเวลา เช่น เว็บไซต์ต่าง ๆ เครื่องเซิร์ฟเวอร์ และเครื่องซูเปอร์คอมพิวเตอร์ ควรมีแหล่งจ่ายไฟสำรองหรือ UPS (Uninterrupted Power Supply) เช่นเดียวกัน และหากไฟดับเป็นเวลานานอาจต้องมีเครื่องปั่นไฟสำรอง

หน่วยความจำหลักมีความจุขนาดใหญ่ แต่ใช้เวลาเข้าถึงข้อมูลนานกว่าแแคช และรีจิสเตอร์ ในขณะที่แแคชมีความจุรองลงมา และรีจิสเตอร์มีความจุน้อยที่สุด แต่ใช้เวลาเข้าถึงข้อมูลน้อยที่สุด เวลาเข้าถึงข้อมูล เป็นปัจจัยสำคัญในการออกแบบสถาปัตยกรรมของคำสั่ง โดยเฉพาะสถาปัตยกรรมโอลด์/สโตร์ การประมวลผลทางคณิตศาสตร์ และตรรกศาสตร์ ค่าของตัวแปรในหน่วยความจำหลัก จะต้องอาศัยขบวนการอ่าน (Load) ข้อมูลจากแแคชหรือหน่วยความจำมาพักเก็บในรีจิสเตอร์ก่อน แล้วจึงนำค่าในรีจิสเตอร์ไปประมวลผล แล้วพักเก็บเพื่อประมวลผลต่อไป เมื่อคำนวนแล้วเสร็จโปรแกรมจึงทำการเขียนหรือสโตร์ (Store) ค่าเก็บในหน่วยความจำหลัก ตามที่อธิบายไปในบทที่ 4

กล่าวโดยสรุปคือ รีจิสเตอร์อยู่ใกล้กับ ALU ใช้เวลาเข้าถึงสั้นกว่าสมอ (เร็ว) แต่จะมีความจุน้อย (เล็ก) เนื่องจากความซับซ้อนในการออกแบบและต้นทุนต่อความจุที่สูงกว่า การจัดแบ่งระดับชั้นต่าง ๆ ของหน่วยความจำอาศัยหลักการใช้งานซ้ำในแกนเวลา (Temporal Locality) จากการวนรอบหรือลูป และในพื้นที่ใกล้เคียง (Spatial Locality) ที่มา: [Harris and Harris \(2013\)](#) ในรูปของตัวแปรอาร์เรย์ หลักการทั้งหมดนี้ทำให้โปรแกรมมองเห็นหน่วยความจำขนาดใหญ่ในรูปของเวอร์ชวลเมโมรีที่มีต้นทุนต่ำ แต่มีเวลาเข้าถึงเฉลี่ยที่ไม่นานจนเกินไป

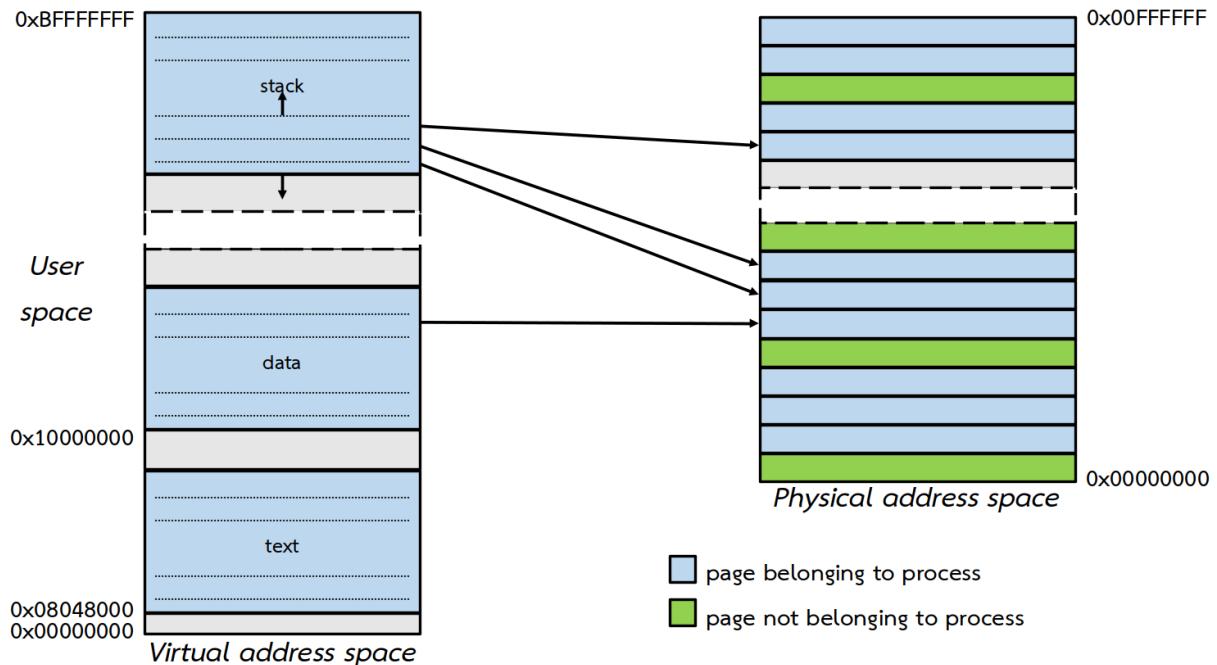
5.2 เวอร์ชวลเม莫รีชนิดเพจ (Paging Virtual Memory)

5.2.1 หลักการพื้นฐานของเวอร์ชวลเม莫รี

ระบบปฏิบัติการสามารถใช้ประโยชน์จากชีพีย์ที่รองรับการสร้างเวอร์ชวลเม莫รีได้ โดยให้ฮาร์ดแวร์ในชีพีย์ช่วยทำหน้าที่แปลงเวอร์ชวลแอดเดรส (Virtual Address) ให้เป็นแอดเดรสกายภาพ (Physical Address) เวอร์ชวลเม莫รีมีหน้าที่และลักษณะสำคัญดังนี้

- รองรับความต้องการความจุของหน่วยความจำภายในมากกว่า 4 กิกะไบต์ (GiB) ด้วยการใช้งานระบบปฏิบัติการเวอร์ชัน 64 บิต
- บริหารจัดการความเร็วที่แตกต่างระหว่างหน่วยความจำภายในมากกว่า 4 กิกะไบต์ (GiB) ด้วยการใช้งานรักษาข้อมูลซึ่งเป็นฮาร์ดดิสก์ชนิดจานหมุนตั้งแต่ในอดีต ถึงแม้หน่วยความจำแฟลช (หัวข้อที่ 7.2) จะมีความเร็วสูงขึ้นและแพร่หลายมากขึ้นในปัจจุบันและต่อไปในอนาคต

จากหัวข้อที่ 3.3.2 ระบบปฏิบัติการโหลดโปรแกรมหรือซอฟต์แวร์ประยุกต์จากไฟล์รูปแบบ ELF เพื่อเตรียมตัวรันซอฟต์แวร์นั้น ศัพท์ทางเทคนิคเรียกโปรแกรมนั้นว่า procress (Process) หมายถึง โปรแกรมที่ระบบปฏิบัติการอนุญาตให้ชีพีย์รัน โดยจะมีเวอร์ชวลเม莫รีของตนเองด้านซ้ายของรูปที่ 5.2 ซึ่งคล้ายกับรูปที่ 3.16 เวอร์ชูลเม莫รีของprocress ดี มีโครงสร้างตามรายละเอียดที่ได้อธิบายแล้วในหัวข้อที่ 3.3.3 เวอร์ชูลเม莫รีของprocress นั่งตัวมีขนาดใหญ่กว่าความจุของหน่วยความจำภายใน ทำให้ผู้เขียนโปรแกรมสามารถพัฒนาโปรแกรมได้อย่างอิสระมากขึ้น ไม่ต้องกังวลกับขนาดของหน่วยความจำภายในที่มีอยู่จริง



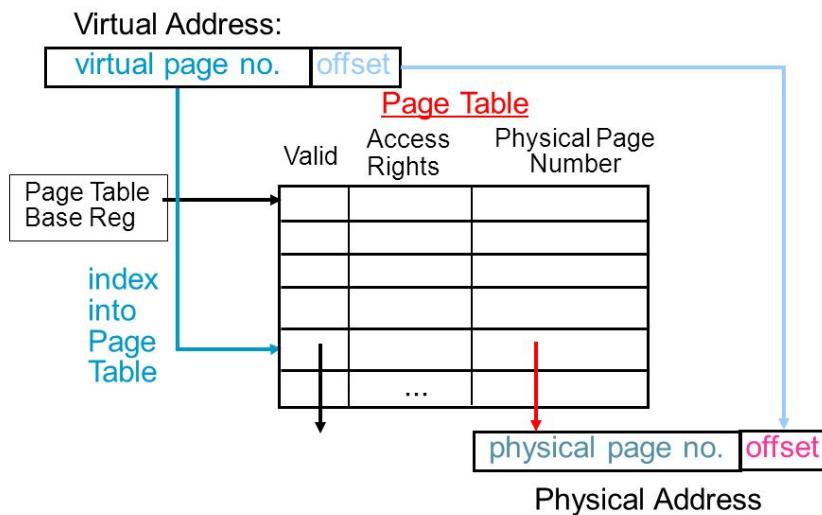
รูปที่ 5.2: การแมป (Map) เวอร์ชวลแอดเดรส (Virtual Address) ขนาด 3 กิกะไบต์ (GiB) เป็นแอดเดรสภายนอก (Physical Address) ขนาด 64 เมกะไบต์ (MiB) ตามหลักการเวอร์ชูลเม莫รีชานิเดจ (Paging Virtual Memory) หมายเหตุ เส้นประ คือ รอยต่อระหว่างเพจของเวอร์ชูลเม莫รี

รูปที่ 5.2 แสดงพื้นที่ยูสเซอร์สเปซความจุขนาด 3 กิกะไบต์ (GiB) หรือประมาณ 3 พันล้านไบต์ ซึ่งถูกแบ่งเป็นพื้นที่ย่อยขนาด 4 KiB (kibibyte) หรือ 4096 ไบต์ เรียกว่า เพจ (Page) ทั้งนี้ขึ้นอยู่กับรายละเอียดของฮาร์ดแวร์และระบบปฏิบัติการ ระบบลินกุกซ์กำหนดให้แต่ละเพจมีขนาด 4 KiB (kibibyte) หมายเหตุ เส้นประในเวอร์ชูลเม莫รี คือ รอยต่อระหว่างเพจของเวอร์ชูลเม莫รี การแบ่งเวอร์ชูลเม莫รีและหน่วยความจำภายในออกเป็นเพจจึงเป็นเรื่องที่เหมาะสม และง่ายต่อการบริหารจัดการ ยกตัวอย่างเช่น

ระบบปฏิบัติการจะจดพื้นที่หน่วยความจำภายในให้แต่ละโปรแกรมเท่าที่ใช้งานจริง มิได้จดพื้นที่ทั้งหมด 3 กิกะไบต์ (GiB) ซึ่งสีฟ้าในหน่วยความจำภายใน (ทางด้านขวาของรูปที่ 5.2) คือ เพจที่ใช้งานจริงของโปรแกรมทางด้านซ้ายที่ระบบปฏิบัติการจดในหน่วยความจำภายใน ดังนั้น เครื่องคอมพิวเตอร์ จึงมีความจุพื้นที่ภายในให้กับโปรแกรมเหลือแบ่งให้กับโปรแกรมอื่นๆ ซึ่งที่เป็นสีเขียว คือ เพจที่ระบบจดให้กับโปรแกรมอื่นๆ ซึ่งระบบปฏิบัติการที่รองรับการทำงานแบบนี้เรียกว่า **มัลติทาสกิ้ง** (Multitasking) ระบบปฏิบัติการจะเปิดโอกาสให้โปรแกรมหลาย ๆ ตัวใช้งานหน่วยความจำภายในได้พร้อม ๆ กัน รายละเอียดเพิ่มเติมที่ [wikipedia](#) ส่วนซองที่เป็นสีขาว หมายถึง เพจที่ว่าง เนื่องจากระบบปฏิบัติการยังไม่ได้จดเพจนี้ให้กับโปรแกรมใด ๆ

การที่ระบบปฏิบัติการรองรับการทำมัลติทาสกิ้งนี้และมีจำนวนโปรแกรมเพิ่มขึ้น จนหน่วยความจำภายในมีพื้นที่ว่างลดลง ระบบจะต้องอาศัยพื้นที่กันไว้ในอุปกรณ์เก็บรักษาข้อมูลเป็นที่พักเก็บเพจต่าง ๆ ชั่วคราว เราเรียกพื้นที่ส่วนนี้ในระบบลินกุกซ์ว่า **สวอปพาร์ทิชัน** (Swap Partition) โดยเครื่องเนลจะย้ายเพจที่ไม่ค่อยได้ใช้งานหรือล้นจากหน่วยความจำภายในมาพักเก็บที่นี่ เพื่อให้หน่วยความจำภายในมีพื้นที่ว่างพอที่ระบบปฏิบัติการจะจดให้กับโปรแกรมที่ต้องการพื้นที่เวอร์ชูลเม莫รีเพิ่มขึ้น เช่น ในพื้นที่ฮิปเซ็กเมนต์ หรือให้กับโปรแกรมใหม่

Address Mapping: Page Table



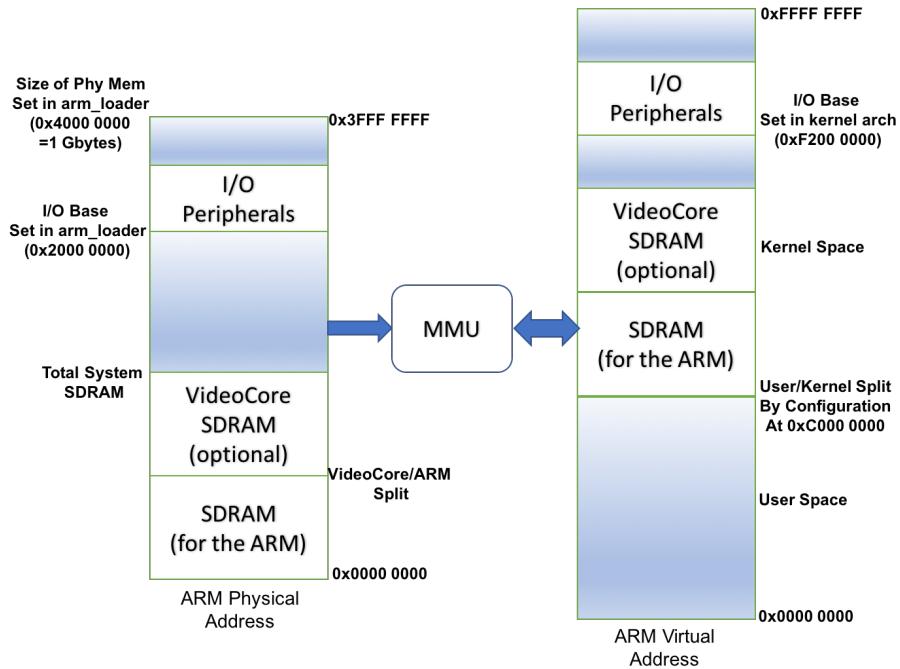
รูปที่ 5.3: การแปลงหมายเลขเวอร์ชวล (Virtual Page Number) เป็นหมายเลขกายภาพ (Physical Page Number) ที่มา: slideplayer.com

แอดเดรสของเวอร์ชวลเมโมรีชนิดเพจ เรียกว่า ๆ ว่า **เวอร์ชวลแอดเดรส** (Virtual Address) แบ่งเป็น 2 ส่วนคือ **หมายเลขเพจเวอร์ชวล** (Virtual Page Number) ความยาว 20 บิต และ **หมายเลขออฟเซต** (Offset) ความยาว 12 บิต ซึ่งคำนวนจากขนาดของเพจ $2^{12}=4096$ ไบต์ ระบบปฏิบัติการจะจองพื้นที่แต่ละเพจในหน่วยความจำกายภาพ เรียกว่า **เพจเฟรม** (Page Frame) หรืออีก 1 ว่า **เพจ** เช่นกัน รูปที่ 5.3 แสดงการแปลงหมายเลขเพจเวอร์ชวล เป็นหมายเลขเพจกายภาพ ส่วนที่เป็นเพจออฟเซตสามารถนำมาเข้ามาร่วมกับหมายเลขเพจกายภาพได้เป็น **แอดเดรสสากยภาพ** (Physical Address) เพื่อนำไปอ้างถึงคำสั่ง หรือข้อมูลในแคชลำดับต่าง ๆ

เครื่องเนลทำการ **จับคู่** (Map) หรือ **แปลงเพจเวอร์ชวล** (Virtual Page Number: VPN) กับ **หมายเลขกายภาพ** (Physical Page Number: PPN) สองเกตได้จากเพจที่เป็นสีฟ้าซึ่งมาจากพรีเซสเดียวกันนี้ ระบบปฏิบัติการเรียกตารางการแมพนี้ว่า **เพจเทเบิล** (Page Table) สำหรับจดบันทึกความสัมพันธ์ระหว่างเลขเพจเวอร์ชูลกับเลขกายภาพ ระบบปฏิบัติการจะเป็นผู้บริหารจัดการเพจเทเบิลนี้เท่านั้น เพราะต้องอยู่บริเวณส่วนของ Kernel Space ของหน่วยความจำกายภาพ

คอมพิวเตอร์ที่ใช้หน่วยความจำกายภาพจากเทคโนโลยี SDRAM ซึ่งบางตัวเรียกว่า หน่วยความจำหลัก เนื่องจากเทคโนโลยี SDRAM มีประสิทธิภาพสูงกว่า และมีต้นทุน (ต่อความจุหน่วย) ต่ำกว่าเทคโนโลยีหน่วยความจำ SRAM แต่ต้นทุนสูงกว่า อุปกรณ์เก็บรักษาข้อมูล ดังนั้น การอ่านหรือเขียนอุปกรณ์เก็บรักษาข้อมูลจำเป็นต้องอ่านหรือเขียนครั้งละมาก ๆ เนื่องจากเวลาเข้าถึง ที่ต้องรอเพื่ออ่านคำสั่ง และถ่ายโอน (อ่านหรือเขียน) ข้อมูลไปมาระหว่างหน่วยความจำหลักและอุปกรณ์เก็บรักษาข้อมูลรายละเอียดสามารถอ่านเพิ่มเติมในบทที่ 7

5.2.2 เวอร์ชูลเม莫รีชnid เพจ กรณีศึกษาบอร์ด Pi3/Pi4



รูปที่ 5.4: โครงสร้างของเวอร์ชูลเม莫รีชnid เพจของบอร์ด Pi3/Pi4 โดยเดล B ซึ่งใช้ชิปประภุล BCM283x, x=5, 6, 7 หมายเหตุ ขนาดของรูปไม่เป็นไปตามพื้นที่ตามความเป็นจริง, MMU: Memory Management Unit ที่มา: [Broadcom Corp. \(2012\)](#)

ระบบปฏิบัติการต่าง ๆ สำหรับบอร์ด Pi3/Pi4 ทำงานตามหลักการเวอร์ชูลเม莫รีชnid เพจ เช่นกัน ดังนั้น การแปลงเวอร์ชูลแอดเดรสเป็นแอดเดรสภายในภาพของชิปประภุล BCM2835-BCM2837/BCM2711 และ BCM2711 มีความคล้ายคลึงกันกับรูปที่ 5.4 โดยมีการดัวร์เรียกว่า **MMU** (Memory Management Unit) ทำหน้าที่แปลง (Address Translation) เวอร์ชูลแอดเดรสเป็นแอดเดรสภายในภาพ MMU จะทำงานร่วมกับเพจテー�เบิลซึ่งได้อธิบายรายละเอียดไปแล้วในหัวข้อก่อนหน้า ดังนี้

- เนื่องจากบอร์ด Pi1 ถึง Pi3/Pi4 ใช้ระบบปฏิบัติการลินุกซ์ ขนาด 32 บิตเดียวกัน พื้นที่เวอร์ชูลเม莫รีของทุก ๆ โพรเซสทางด้านขวาของรูปมีขนาดเท่ากัน คือ 4 กิกะไบต์ (GiB) โดยแบ่งเป็น
 - พื้นที่ขนาด 1 กิกะไบต์ (GiB) สำหรับเก็บคำสั่งและข้อมูลของคอร์แนล (Kernel) ในการบริหารจัดการเครื่อง เรียกว่า **Kernel Space** โดยเวอร์ชูลแอดเดรสของ Kernel เริ่มต้นที่หมายเลข 0xC000 0000 - 0xFFFF FFFF แบ่งเป็น
 - * พื้นที่ชื่อว่า **I/O Peripherals** เริ่มต้นที่หมายเลข 0xF200 0000 แมพไปอยู่ที่แอดเดรสภายในภาพ 0x2000 0000
 - * พื้นที่ชื่อว่า **VideoCore** คือ พื้นที่สำหรับจีพียูของชิป BCM2837/BCM2711 บนบอร์ด Pi3/Pi4 จะถูกกำหนดขนาดในไฟล์ start.elf ตั้งอยู่ในพาร์ติชัน boot ซึ่งฟอร์แมตด้วยรูปแบบ FAT32 โดยขนาดหน่วยความจำที่กำหนดให้จีพียูขั้นต่ำที่สุด ขนาด 32 เมบิไบต์

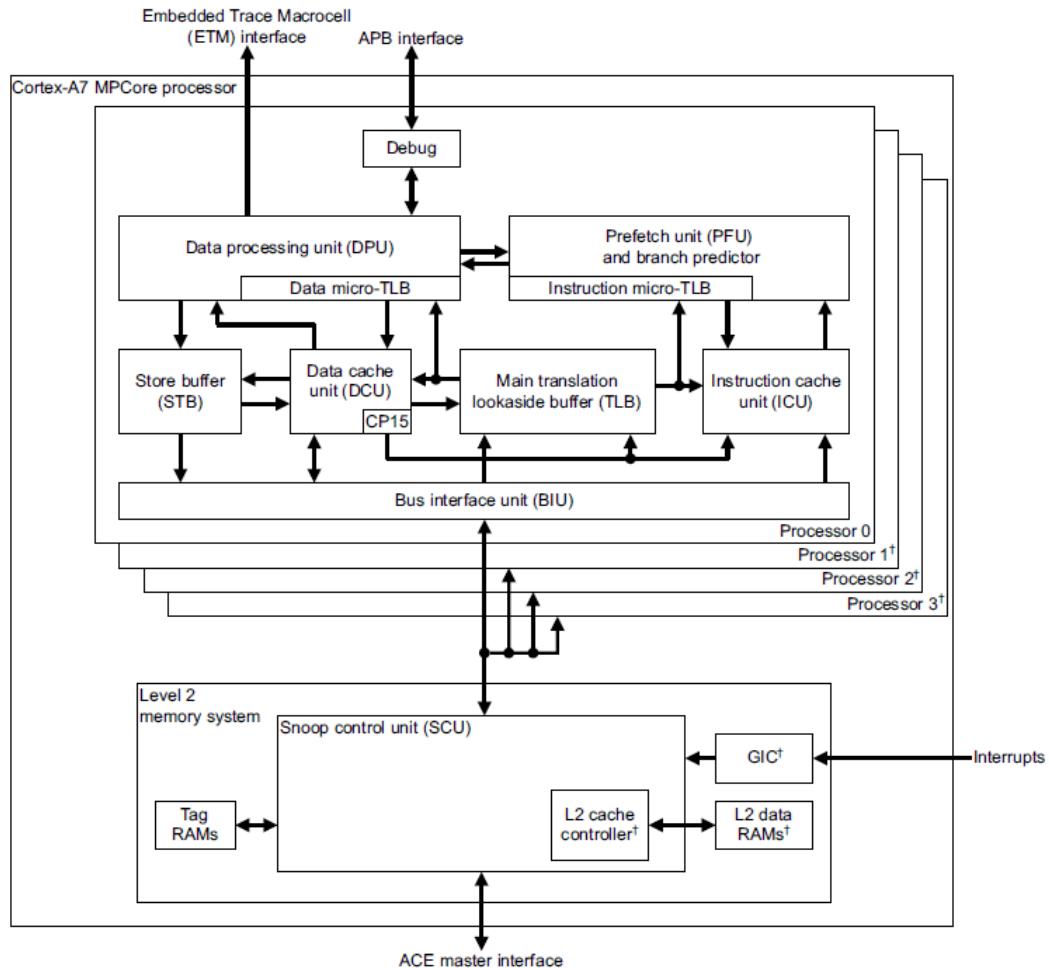
แต่ความละเอียดของการแสดงผลบนจอ 1080p30 ผ่านสาย HDMI ต้องการหน่วยความจำขนาด 64 เมบิไบต์ สำหรับหน้าที่เป็นบัฟเฟอร์สำหรับแสดงผล รายละเอียดเพิ่มเติมในการทดลองที่ 9 ในภาคผนวก I

- * พื้นที่ SDRAM (for the ARM) ซึ่งแมปไปอยู่ที่แอดเดรสภายใน 0x0000 0000 เป็นพื้นที่สำหรับบรรจุโปรแกรมเครื่องเนลสำหรับบริหารจัดการระบบทั้งหมด
- พื้นที่ขนาด 3 กิกะไบต์ (GiB) เรียกว่า **ยูสเซอร์สเปซ** แบ่งเป็นเซกเมนต์ต่าง ๆ สำหรับเก็บคำสั่ง และข้อมูลของแต่ละโพรเซสคล้ายกับรูปที่ 3.16 เริ่มต้นที่เวอร์ชัลแอดเดรสจากหมายเลข 0x0000 0000 จนถึง 0xBFFF FFFF โดยขนาดของยูสเซอร์สเปซ และ เครื่องเนลสเปซ (User/Kernel Split) กำหนดอยู่ในไฟล์ config.txt ซึ่งในรูปคือ ที่หมายเลข 0xC0000 0000 และในหัวข้อที่ 3.3.1
- พื้นที่หน่วยความจำภายในของบอร์ด Pi3/Pi4 อยู่ทางด้านซ้ายของรูป ขนาดความจุรวมเท่ากับ 1 กิกะไบต์ (GiB) โดยมีแอดเดรสภายในเริ่มต้นที่หมายเลข 0x0000 0000 - 0x3FFF FFFF แบ่งเป็นพื้นที่สำคัญ ดังนี้
 - SDRAM (for the ARM) เครื่องเนล จะใช้งานเพียงผู้เดียวสำหรับจัดเก็บเพจเทเบิล พื้นที่ส่วนนี้รวมกับ VideoCore SDRAM ถูกกำหนดขอบเขตโดย Total System SDRAM
 - พื้นที่สำหรับการแสดงผลทั้งหมดเท่ากับ 32 เมบิไบต์ (MiB) โดยสามารถตรวจสอบขนาดที่แท้จริงได้ในการทดลองที่ 4 ภาคผนวก D
 - พื้นที่สำหรับ เชื่อม ต่อ กับ อุปกรณ์ อินพุต/เอาต์พุต ซึ่งไม่ผ่านแคช (Uncached) เริ่มต้นที่ แอดเดรส 0x2000 0000 ตามหลักการ Memory Mapped I/O ซึ่งจะกล่าวต่อไปในหัวข้อ 6.9
 - พื้นที่อื่น ๆ ที่เหลือในรูปเป็นเขตสีฟ้า เครื่องเนลจะบริหารจัดการโดยแบ่งให้โปรแกรมต่าง ๆ ใช้งานร่วมกัน

ผู้อ่านสามารถค้นควารายละเอียดด้านต่าง ๆ ของบอร์ด Pi3/Pi4 เพิ่มเติมได้ที่ github.com กรณีศึกษาชีพิญ ARM Cortex A53/A72 บนบอร์ด Pi3/Pi4 พัฒนาต่อยอดจากชีพิญ ARM Cortex A7 ซึ่งทั้งคู่จัดเป็นชิประดับต้น (Entry Level) สำหรับโทรศัพท์เคลื่อนที่สมาร์ตโฟน ผู้อ่านสามารถทำความเข้าใจการทำงานของเวอร์ชัลเมโมรีของชีพิญ ARM Cortex A53/A72 จากชีพิญ ARM Cortex A7 จากรูปที่ 5.5 เนื่องจาก Cortex A53/A72 พัฒนาต่อจาก Cortex A7

การแบ่งหมายเลข เพจ เวอร์ชัล เป็นหมายเลข เพจ ก咽ภาพ จะต้องอาศัย เพจ เทเบิล ตามที่อธิบายไปแล้วก่อนหน้า โดยมี TLB (Translation Lookaside Buffer) ที่มา: [Tanenbaum and Austin \(2012\)](https://www.cs.cmu.edu/~tanenbaum/OS/Notes/TLB.pdf); [Harris and Harris \(2013\)](https://www.cs.cmu.edu/~tanenbaum/OS/Notes/TLB.pdf) หน้าที่เป็นเพจ เทเบิลขนาดเล็กอยู่ติดกับแกนประมวลผล และแคชล้ำดับที่ 1 ทั้งนี้ TLB แบ่งเป็น ITLB และ DTLB ซึ่งใช้หน่วยความจำสแตติกแรม เช่นเดียวกับแคชทั่วไป และแบ่งเป็นล้ำดับชั้นเหมือนกับแคชล้ำดับที่ 1 โดย ITLB ย่อมจำกัดว่า Instruction TLB หมายถึง TLB ล้ำดับที่ 1 แปลหมายเลข เพจ เวอร์ชัล ที่คำสั่งภาษาเครื่องอยู่ในพื้นที่เท็กซ์เซกเมนต์เท่านั้น ให้เป็นหมายเลข เพจ

ภายในภาพ และ DTLB ย่อมาจาก Data TLB หมายถึง TLB ลำดับที่ 1 ทำหน้าที่แปลงหมายเลขเพจเวอร์ชวลที่เก็บหมายเลขเพจซึ่งอยู่ใน เช็คเม้นต์ อื่นๆ และ TLB ลำดับที่ 2 จะรวมหมายเลขเพจเวอร์ชวลของทุกเช็คเม้นต์เข้าด้วยกัน คล้ายกับแคชลำดับที่ 1 และแคชลำดับที่ 2 ซึ่งได้กล่าวแล้วในหัวข้อที่ 5.1



รูปที่ 5.5: โครงสร้างของชีพีชี ARM Cortex A7 ประกอบด้วย TLB และแคชหลายระดับเพื่อรับรู้การทำงานของเวอร์ชวลเม莫รีชินิดเพจ ที่มา: [ARM Ltd. \(2011\)](#) และ [arm.com](#)

โครงสร้างของชีพีชี 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 เพื่อรับรู้การทำงานของเวอร์ชวลเม莫รีชินิดเพจ

ผู้อ่านสามารถเรียงลำดับการทำงานของการเฟทธ์คำสั่งในรูปที่ 5.5 เพื่อนำคำสั่งไปถอดรหัส และประมวลผลในไปป์ไลน์ ดังนี้

1. นำเลขเพจเวอร์ชวลใน PC (Program Counter) ไปสืบค้นหาค่าเลขเพจภายใน Instruction micro-TLB
 - หากเจอ เรียกว่า TLB ฮิต (Hit) จะใช้เวลาส่งค่าหมายเลขเพจภายในภาพไปใช้งานสั้นมาก ๆ

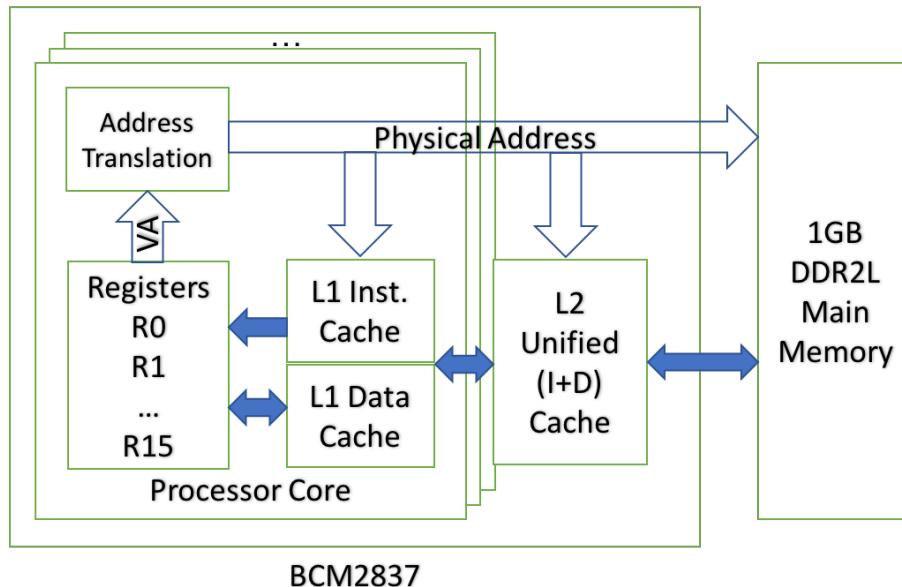
(0.5-1 นาโนวินาที)

- หากไม่เจอ เรียกว่า TLB มิส (Miss) วงจรจะนำหมายเลขเพจเวอร์ชัลไปสืบค้นใน Main TLB ต่อไปซึ่งจะใช้เวลาเพิ่มขึ้นอีก
 - หากชิต ส่งค่าหมายเลขเพจภายในภาพไปใช้งาน (2-4 นาโนวินาที)
 - หากมิสอีกรอบ จะต้องสืบค้นในเพจเทเบิลเป็นลำดับสุดท้ายซึ่งจะใช้เวลานานขึ้น
 - * หากชิต ส่งค่าหมายเลขเพจภายในภาพจากเพจเทเบิลไปใช้งาน (หลักสิบนาโนวินาที)
 - * หากมิสอีกรอบ เรียกว่า เพจฟอลท์ (Page Fault) แสดงว่าเครื่องเนลยังมิได้อ่านคำสั่งที่ต้องการจากเพจในอุปกรณ์เก็บรักษาข้อมูลมาบรรจุในหน่วยความจำภายในภาพซึ่งจะใช้เวลานานที่สุด (เกิน 100 นาโนวินาที) รายละเอียดเพิ่มเติมที่ [Tanenbaum and Bos \(2014\)](#) และ [wikipedia](#)

2. นำเลขเพจภายในภาพมาเข้ามอกับค่าออฟเซตเพื่อนำไปสืบค้นคำสั่งใน ICU ซึ่งเทียบเท่าแคชคำสั่งลำดับที่ 1

- หากเจอ เรียกว่า เกิดแคชชิตที่ลำดับที่ 1 (L1 Cache Hit) และนำคำสั่งส่งกลับไปยังชีพิญต่อไปซึ่งจะใช้เวลาสั้นที่สุด (0.5 - 1 นาโนวินาที)
- หากไม่เจอ เรียกว่า เกิดแคชมิสที่ลำดับที่ 1 (L1 Cache Miss) นำแอดเดรสภายในภาพไปค้นหาคำสั่งในแคชลำดับที่ 2
 - หากชิต ที่ลำดับที่ 2 (L2 Cache Hit) และนำคำสั่งส่งกลับไปยังชีพิญและแคชลำดับที่ 1 ซึ่งจะใช้เวลานานขึ้น (2 - 4 นาโนวินาที)
 - หากมิส ที่ลำดับที่ 2 (L2 Cache Miss) วงจรจะนำแอดเดรสภายในภาพไปค้นหาคำสั่งในหน่วยความจำหลัก (SDRAM) ผ่านทางวงจรเข้ามอกับหน่วยความจำ SDRAM ชีพิญจะต้องใช้เวลารอ (นานกว่าสิบนาโนวินาที) เพื่อที่คำสั่ง จะเดินทางมาจากหน่วยความจำภายในภาพ คำสั่งที่ได้จากแคชลำดับต่อๆ ๆ และจาก SDRAM จะเป็นเลขฐานสองจำนวน 4-8 คำสั่ง คิดเป็นความยาว 128-256 บิต ซึ่งอยู่กับความกว้างของแคชแต่ละบล็อก 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 วงจรแปลเวอร์ชวล แอดเดรส (VA) ซึ่งหมายถึง TLB และ เพจ เทเบิล ให้เป็น แอดเดรสภายในภาพ (PA: Physical Address) ส่งผ่านทางเส้นทึบสีขาว (Bus) ไปยังแคชลำดับที่ 1 ซึ่งแบ่ง เป็นแคชคำสั่งและแคชข้อมูล (Data Only) แอดเดรสภายในภาพจะส่งไปยังแคชลำดับที่ 2 และ หน่วยความจำภายในภาพ เช่นกัน วงจรเหล่านี้อ้างอิงคำสั่งและข้อมูลด้วยแอดเดรสภายในภาพ คำสั่งและข้อมูลจะวิ่งผ่าน เส้นทึบสีน้ำเงิน (Bus) กลับมายังรีจิสเตอร์ภายในแกนประมวลผล โดยจะเริ่มจากซีพียูเฟทซ์คำสั่ง จาก แคชคำสั่ง (Instruction Cache) ลำดับที่ 1 ก่อน หากไม่เจอจะค้นคำสั่งนั้นจากแคชลำดับที่ 2 ทั้งนี้ หากคำสั่งนั้นเป็นคำสั่งประเภท LOAD หรือ คำสั่ง STORE ซีพียูปฏิบัติตาม (Execute) จะค้นหาเพื่ออ่าน/เขียน ข้อมูลในแคชข้อมูล (Data Cache) ลำดับที่ 1 ก่อน เช่นกัน แคชลำดับที่ 2 เป็นแบบรวม (Unified Cache) ทำหน้าที่พักรีบคำสั่งและข้อมูล ซึ่งได้อธิบายโดยละเอียดในหัวข้อก่อนหน้าแล้ว

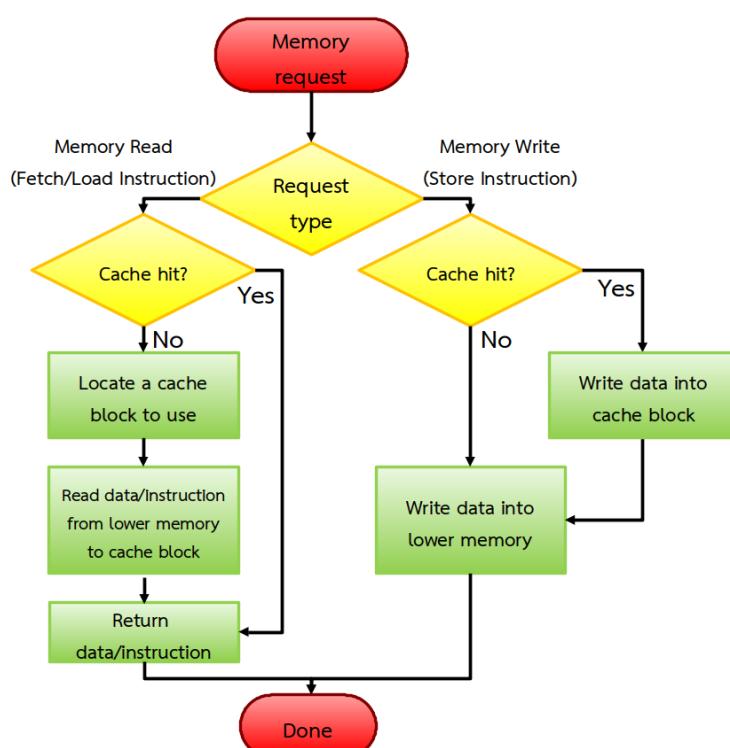
แคชทั้งสองลำดับอยู่บนชิปเดียวกับซีพียูช่วยลดเวลาการเข้าถึงข้อมูล ในขณะที่ หน่วยความจำหลัก หรือ หน่วยความจำภายในภาพสร้างเทคโนโลยี หน่วยความจำชนิด SDRAM ทำให้เวลาการเข้าถึงคำสั่งและ ข้อมูลนานขึ้น และมักติดตั้งอิสระจากซีพียู (Off-chip) ทำให้เพิ่มเวลาการเดินทางของคำสั่งและข้อมูล ซึ่ง จะได้กล่าวในหัวข้อที่ 5.5 ต่อไป

แคชจึงทำหน้าที่คล้ายกับบัฟเฟอร์เก็บพักคำสั่ง หรือ ข้อมูล ที่อ่านจากหน่วยความจำหลัก เพราะ การทำงานของซอฟต์แวร์ส่วนหนึ่ง จะเป็นการวนรอบ เช่น การวนรอบชนิดต่าง ๆ ในหัวข้อที่ 4.7.3 - 4.7.6 ทำให้ใช้คำสั่งเดิม ๆ ใน Loop Body ซ้ำแล้วซ้ำอีก เรียกว่า **Temporal Locality** ดังนั้น การพักเก็บ คำสั่งหรือข้อมูล ที่มีการเรียกใช้บ่อยในแคชลำดับต่าง ๆ เป็นประโยชน์ต่อระบบโดยภาพรวม เนื่องจาก ช่วยลดเวลาการเข้าถึงหน่วยความจำ SDRAM และเวลาเดินทางของสัญญาณจากชิปสู่ชิป (Chip-Chip

Propagation Time)

การอ่านหรือเขียนค่าตัวแปรทุกชนิดในเวอร์ชวลเม莫รีทุกตำแหน่งหรือทุกหมายเลขแอดเดรส จะต้องผ่านแคชเสมอ ดังนั้น แคชจะทำงานตามโพล์ชาร์ตในรูปที่ 5.7 เริ่มต้นจากการแปลจากแอดเดรสเวอร์ชวลเป็นแอดเดรสภายในภาพด้วย TLB และเพจテー�เบิลเสมอ การทำงานการเฟทซ์คำสั่งจากแคชคำสั่งหรือการโหลดข้อมูลจากแคชข้อมูลจะอยู่ด้านซ้ายของรูป การเขียนข้อมูลลงในแคชข้อมูลจะอยู่ด้านขวาของรูป การตรวจสอบว่า แคชฮิต หรือ แคชมิส ขึ้นอยู่กับรายละเอียดของแคชชนิดต่าง ๆ ซึ่งจะอธิบายในหัวข้อถัดไป

- การอ่านคำสั่ง/ข้อมูล พบในแคชหรือไม่
 - หากใช่ เรียกว่า แคชฮิต แสดงว่าแค้มีคำสั่งหรือข้อมูลแล้ว แคชจะส่งคำสั่งหรือข้อมูลนั้นให้กับแคชลำดับสูงกว่าต่อไปจนถึงชีพียู
 - หากไม่ เรียกว่า แคชมิส
- การเขียน (Store) ข้อมูล พบในแคชหรือไม่
 - หากใช่ เรียกว่า แคชฮิต แสดงว่าแคชมีข้อมูลแล้ว บล็อกข้อมูลนั้นจะถูกค่าใหม่เขียนทับลงไป
 - หากไม่ เรียกว่า แคชมิส วงจรจะเขียนข้อมูลในแคชลำดับถัดไป หรือ หน่วยความจำภายใน



รูปที่ 5.7: โพล์ชาร์ตการทำงานของแคช หนึ่งลำดับชั้นสำหรับการเฟทซ์คำสั่งและสำหรับการอ่าน/เขียนข้อมูล

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

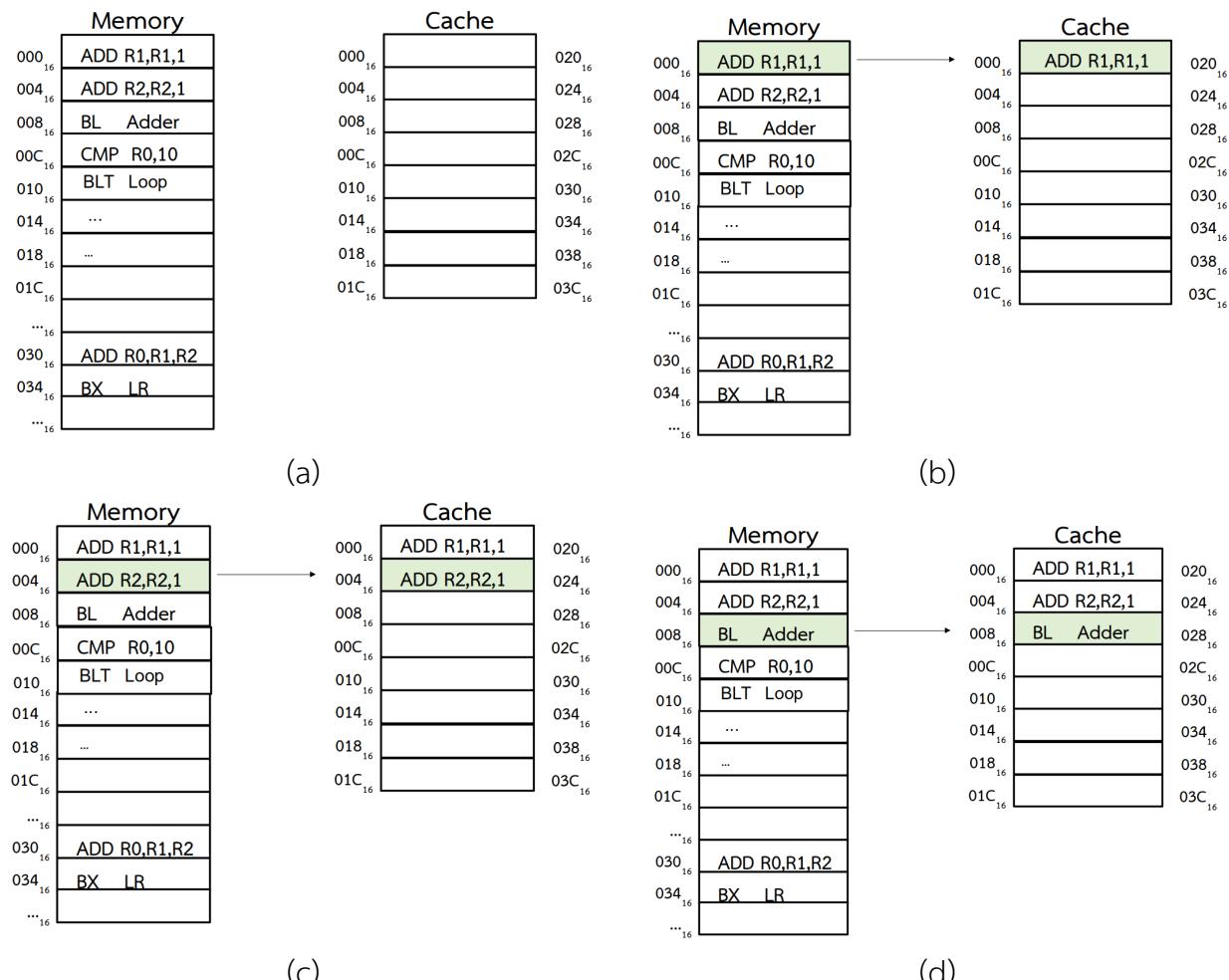
แคชลำดับที่ 1 นิยมออกแบบด้วยแคชชนิด SA (Set Associative) ซึ่งต้องอาศัยพื้นฐานของแคชชนิด DM (Direct Map) และมีสองชุด แบ่งเป็นแคชคำสั่ง และแคชข้อมูลตามรูปที่ 5.1 แคชลำดับที่ 2 เป็นแบบรวมทำหน้าที่พักเก็บคำสั่งและข้อมูล จึงมีความจุมากกว่าแคชลำดับที่ 1 นิยมออกแบบด้วยแคชชนิด SA แต่ต่อมาเล่มนี้จะเปรียบเทียบเพียงแค่สองชนิดที่นิยมและเข้าใจง่าย โดยใช้ชุดคำสั่งเดียวกันในตารางที่ 5.1 อธิบายการทำงานของแคชคำสั่งทั้งสองชนิด หมายเหตุ แอ็ดเดรสภายในภาพส่วนนี้ คือ ออฟเซตยาว 12 บิตของรีจิสเตอร์ PC ซึ่งไม่เปลี่ยนแปลงตามหลักการแปลงเวอร์ชวลแอ็ดเดรสเป็นแอ็ดเดรสภายในภาพซึ่งได้อธิบายในรูปที่ 5.3

ตารางที่ 5.1: ตัวอย่างโปรแกรมภาษาแอสเซมบลีเพื่อประกอบการอธิบายการทำงานของแคชทั้งสองชนิด

ออฟเซต	PC	เลbel	คำสั่ง	คอมเมนต์
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 บรรจุอยู่ในหน่วยความจำภายในภาพ ขนาด 4 KiB (kibibyte) หรือ 4096 ไบต์ ตามขนาดของเพจภายในภาพ แต่ละ夸บจะคำสั่งขนาด 32 บิตหรือ 4 ไบต์ ทำให้หมายเลขอффเซตเริ่มนับจาก 000₁₆, 004₁₆, 00C₁₆, 010₁₆ ไปเรื่อย ๆ จนถึง FFC₁₆

5.3.1 แคชชนิด DM (Direct Map)



รูปที่ 5.8: การทำงานของแคชคำสั่ง (Instruction Cache) ชนิด Direct Map เมื่อซีพียูเพทช์คำสั่งที่หน่วยความจำภายในภาพแอดเดรส (a) แคชคำสั่งยังไม่มีคำสั่งใด ๆ เลย (b) PC ออฟเซตเท่ากับ 000₁₆, (c) 004₁₆, (d) 008₁₆

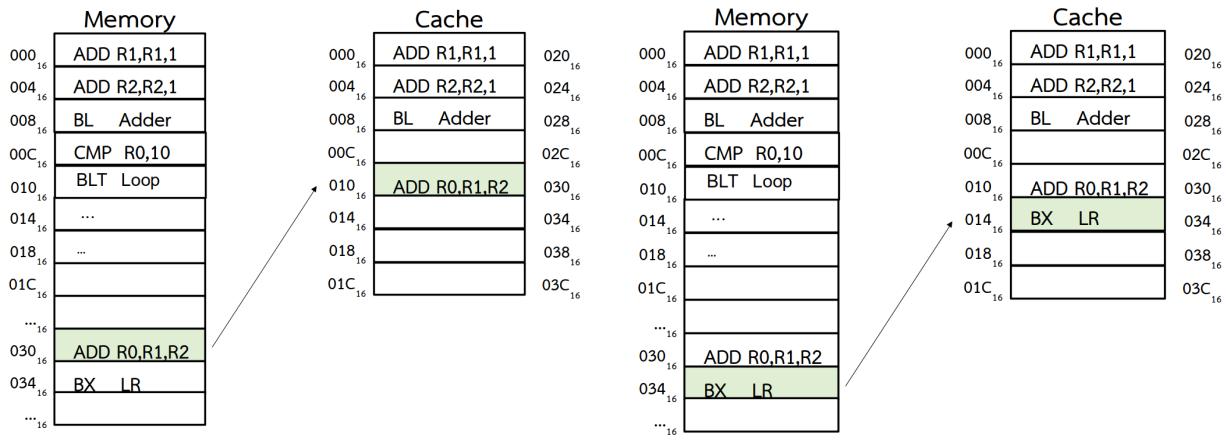
รูปที่ 5.8 และ 5.9 แสดงการทำงานของแคชคำสั่งชนิด DM ซึ่งเป็นแคชพื้นฐานของการทำงานของแคชลำดับต่าง ๆ การทำงานแคชชนิดนี้มีความซับซ้อนน้อยที่สุดและคล้ายหน่วยความจำบัฟเฟอร์ ซึ่งทำหน้าที่พักเก็บคำสั่งที่ใช้บ่อย ๆ แคชจำลองชนิด DM ในรูปมีขนาด 8 บล็อก แต่ละบล็อกมีความจุ 4 ไบต์ ทำหน้าที่เก็บพักคำสั่งจากแอดเดรสหมายเลข 000₁₆, 004₁₆ จนถึงหมายเลข 01C₁₆ โดยໄลจากบล็อกบนสุดลงมาวนซ้ำจากแอดเดรสหมายเลข 020₁₆, 024₁₆ จนถึงหมายเลข 03C₁₆ และวนซ้ำอีกหลายรอบจนถึงแอดเดรสหมายเลข FEO₁₆, FE4₁₆ จนถึงหมายเลข FFC₁₆

(a) เริ่มต้น แคชไม่มีข้อมูลใด ๆ

(b) ออฟเซต 000₁₆ ตรงกับคำสั่ง ADD R1, R1, 1 ยังเก็บอยู่ในหน่วยความจำภายในภาพ ซีพียูจึงตรวจสอบถามแคชบล็อกหมายเลข 000₁₆ ซึ่งว่างเปล่า เราเรียกเหตุการณ์นี้ว่า **โคลด์มิส** (Cold Miss) วงจรควบคุมแคชจะนำคำสั่งนี้จากหน่วยความจำภายในไปเก็บในแคชบล็อกที่ 000₁₆ พร้อมกับส่งให้ซีพียูอุดรหัสและ PC ออฟเซตจะเปลี่ยนเป็นหมายเลข 004₁₆

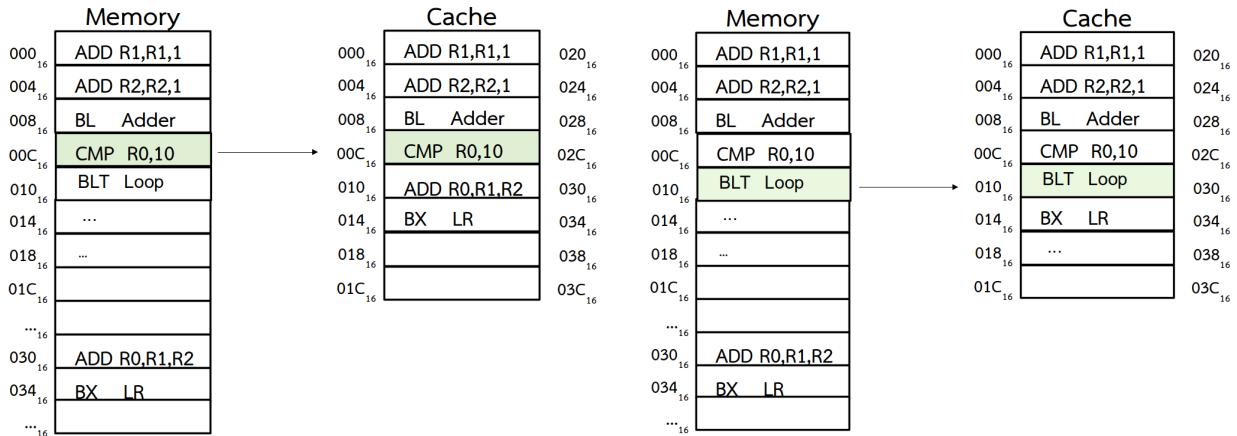
(c) ออฟเซต 004_{16} ตรงกับคำสั่ง **ADD R2, R2, 1** ยังเก็บอยู่ในหน่วยความจำภายในชีพียู จึงตรวจสอบแคชบล็อกหมายเลข 004_{16} ซึ่งว่างเปล่า วงจรควบคุมแคชจึงนำคำสั่งนี้จากหน่วยความจำภายในชีพียูไปเก็บในแคชบล็อกที่ 004_{16} พร้อมกับส่งให้ชีพียูถอดรหัสและ PC ออฟเซตจึงเปลี่ยนเป็นหมายเลข 008_{16}

(d) ออฟเซต 008_{16} ตรงกับคำสั่ง **BL Adder** ยังเก็บอยู่ในหน่วยความจำภายในชีพียู จึงตรวจสอบแคชบล็อกหมายเลข 008_{16} ซึ่งว่างเปล่า วงจรควบคุมแคชจึงนำคำสั่งนี้จากหน่วยความจำภายในชีพียูไปเก็บในแคชบล็อกที่ 008_{16} พร้อมกับส่งให้ชีพียูถอดรหัสและ PC ออฟเซตจึงเปลี่ยนเป็นหมายเลข 030_{16}



(e)

(f)



(g)

(h)

รูปที่ 5.9: การทำงานของแคชคำสั่ง (Instruction Cache) ชนิด Direct Map เมื่อชีพียูเฟห์ซ์คำสั่งที่ PC ออฟเซตเท่ากับ (e) 030_{16} , (f) 034_{16} , (g) $00C_{16}$, (h) 010_{16}

(e) ออฟเซต 030_{16} ตรงกับคำสั่ง **ADD R0, R1, R2** ยังเก็บอยู่ในหน่วยความจำภายในชีพียู จึงตรวจสอบแคชบล็อกหมายเลข 030_{16} ซึ่งว่างเปล่า วงจรควบคุมแคชจึงนำคำสั่งนี้จากหน่วยความจำภายในชีพียูไปเก็บในแคชบล็อกที่ 030_{16} พร้อมกับส่งให้ชีพียูถอดรหัสและ PC ออฟเซตจึงเปลี่ยนเป็นหมายเลข 034_{16}

(f) ออฟเซต 034_{16} ตรงกับคำสั่ง **BX LR** ยังเก็บอยู่ในหน่วยความจำภายในชีพียู จึงตรวจสอบแคชบล็อกหมายเลข 034_{16} ซึ่งว่างเปล่า วงจรควบคุมแคชจึงนำคำสั่งนี้จากหน่วยความจำภายในชีพียูไปเก็บในแคชบล็อกที่ 034_{16} พร้อมกับส่งให้ชีพียูถอดรหัสและ PC ออฟเซตจึงเปลี่ยนเป็นหมายเลข $00C_{16}$ เพื่อเรียกกลับ (Return)

(g) ออฟเซต $00C_{16}$ ตรงกับคำสั่ง **CMP R0, 10** ยังเก็บอยู่ในหน่วยความจำภายในชีพียู จึงตรวจสอบ

แคชบล็อกหมายเลข $00C_{16}$ ซึ่งว่างเปล่า วงจรควบคุมแคชจึงนำคำสั่งนี้จากหน่วยความจำภายในไปเก็บในแคชบล็อกที่ $00C_{16}$ พร้อมกับส่งให้ชีพิญถอดรหัสและ PC ออฟเซตจึงเปลี่ยนเป็นหมายเลข 010_{16}

(h) ออฟเซต 010_{16} ตรงกับคำสั่ง **BLT Loop** ยังเก็บอยู่ในหน่วยความจำภายใน ชีพิญจึงตรวจสอบแคชบล็อกหมายเลข 010_{16} มีคำสั่งอื่นอยู่ เราเรียกเหตุการณ์นี้ว่า **คอนเทนเอนชันมิส** (Contention Miss) วงจรควบคุมแคชจึงนำคำสั่งนี้จากหน่วยความจำภายในมาเขียนทับ แคชบล็อกนี้แทนพร้อมกับนำไปถอดรหัส และ PC ออฟเซตจะเปลี่ยนเป็น 000_{16} หรือไม่ขึ้นอยู่กับค่าของ R0 ว่าเงื่อนไข LT (Less Than) ซึ่งจะแปลเป็นเงื่อนไข $R0 < 10$ เป็นจริง หรือ เป็นเท็จ

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

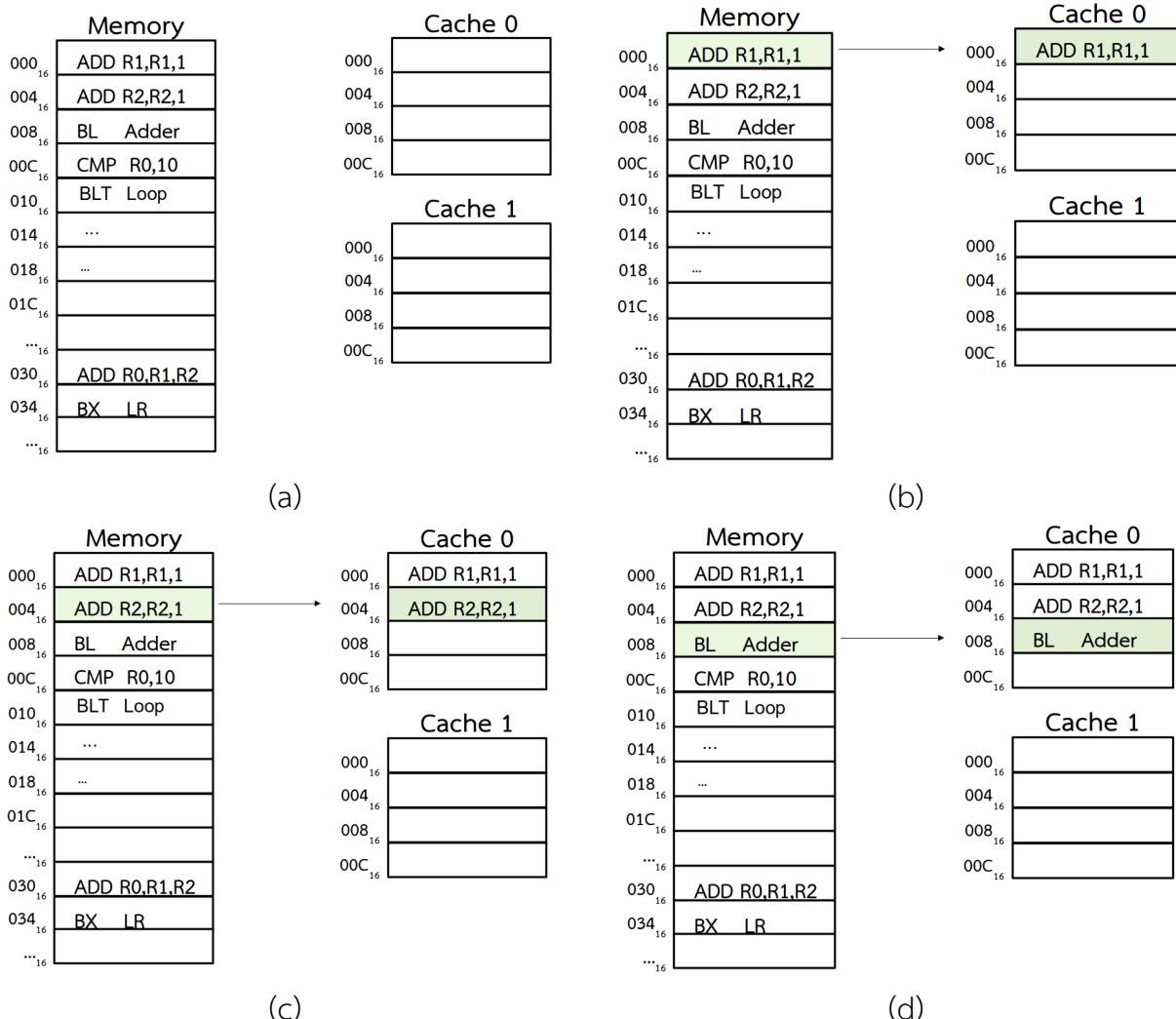
ผู้อ่านจะสังเกตเห็นว่า คำสั่งจากหน่วยความจำภายในอย่างน้อย 2 ตำแหน่งถูก mapped ให้ใช้แคชบล็อกตำแหน่งเดียวกัน เช่น แคชตำแหน่งที่ 000_{16} จะเป็นเป้าหมายของหน่วยความจำตำแหน่งที่ 000_{16} และ 020_{16} เป็นต้น ยิ่งไปกว่านั้น แคชลำดับที่ 1 ที่ใช้งานในชีพิญต่าง ๆ มีความจุน้อยกว่าหน่วยความจำภายในมาก เพราะหน่วยความจำภายในของบอร์ด Pi3/Pi4 มีขนาดอย่างน้อย 1×2^{30} ไบต์ หรือ 1 กิกะไบต์ (GiB) ในขณะที่ขนาดแคชลำดับที่ 1 มีขนาดเล็กเพียง 16×2^{10} ไบต์ หรือ 16 KiB (kibibyte) คิดเป็น $2^{16} : 1$ ทำให้เกิดการแข่งขัน (Contention) เป็นแบบ Many to One ดังนั้นเมื่อแคชมีขนาดเล็ก อัตราการแข่งขัน (Contention Rate) จะยิ่งเพิ่มสูงมากขึ้น ในทางปฏิบัติ ชีพิญต้องมีแคชหลายลำดับชั้นเพิ่มขึ้นเพื่อช่วยลดอัตราการแข่งขันนี้

5.3.2 แคชชนิด N-way SA (Set Associative)

แคชคำสั่งชนิด N-Way SA (Set Associative) นิยมใช้ออกแบบแคชทุกลำดับ จำนวนเวย์ (Way) จะเพิ่มเป็นจำนวนสองยกกำลัง เช่น $N = 8, 16, 32$ เวย์ เป็นต้น แคชจำลองชนิด 2-Way SA ในรูปที่ 5.10 และ 5.11 ประกอบด้วย Cache 0 และ Cache 1 หมายถึง แคชชนิด DM เวย์ที่ 0 และ เวย์ที่ 1 ตามลำดับ แต่ละเวย์มีขนาด 4 บล็อก แต่ละบล็อกมีความจุ 4 ไบต์ สามารถบรรจุคำสั่งบล็อกละ 1 คำสั่ง ทำหน้าที่เก็บพักคำสั่งจากแอดเดรสหมายเลข $000_{16}, 004_{16}$ จนถึงหมายเลข $00C_{16}$ โดยไล่จากบล็อกบนสุดลงมาวนซ้ำจากแอดเดรสหมายเลข $010_{16}, 014_{16}$ จนถึงหมายเลข $01C_{16}$ และวนซ้ำอีกหลายรอบจนถึงแอดเดรสหมายเลข $FF0_{16}, FF4_{16}$ จนถึงหมายเลข FFC_{16}

(a) เริ่มต้น แคชไม่มีข้อมูลใด ๆ

(b) ออฟเซต 000_{16} ตรงกับคำสั่ง **ADD R1, R1, 1** ยังเก็บอยู่ในหน่วยความจำภายใน ชีพิญจึงตรวจสอบแคชบล็อกหมายเลข 000_{16} ของ Cache 0 และ Cache 1 ยังว่างเปล่า เราเรียกเหตุการณ์นี้ว่า **โคลด์มิส** (Cold Miss) วงจรควบคุมแคชจึงนำคำสั่งนี้จากหน่วยความจำภายในไปเก็บในแคช 0 บล็อกที่ 000_{16} พร้อมกับส่งให้ชีพิญถอดรหัสและ PC ออฟเซตจึงเปลี่ยนเป็นหมายเลข 004_{16}

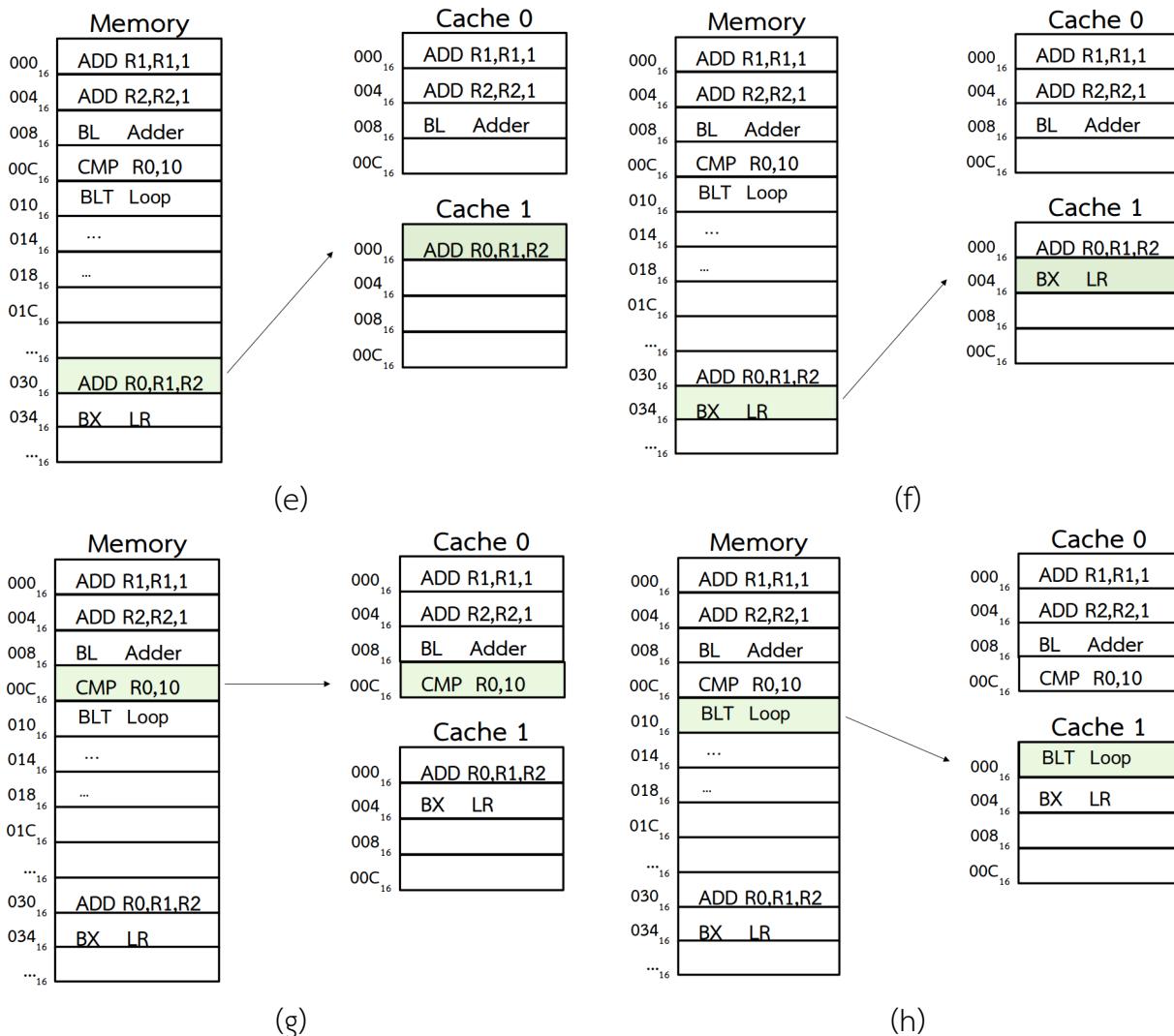


รูปที่ 5.10: การทำงานของแคชลำดับที่ 1 แคชคำสั่ง (Instruction Cache) ชนิด 2-Way Set Associative เมื่อ PC เฟทซ์คำสั่งที่หน่วยความจำภายในภาพแอ็ตเตรส (a) แคชคำสั่งยังไม่มีคำสั่งใด ๆ เลย (b) PC ออฟเซตเท่ากับ 000₁₆, (c) 004₁₆, (d) 008₁₆

(c) ออฟเซต 004₁₆ ตรงกับคำสั่ง ADD R2, R2, 1 ยังเก็บอยู่ในหน่วยความจำภายในภาพ ซีพียูจึงตรวจสอบแคชบล็อกหมายเลข 004₁₆ ของ Cache 0 และ Cache 1 ยังว่างเปล่า วงจรควบคุมแคชจึงนำคำสั่งนี้จากหน่วยความจำภายในภาพไปเก็บในแคช 0 บล็อกที่ 004₁₆ พร้อมกับส่งให้ซีพียูถอดรหัสและ PC ออฟเซตจึงเปลี่ยนเป็นหมายเลข 008₁₆

(d) ออฟเซต 008₁₆ ตรงกับคำสั่ง BL Adder ยังเก็บอยู่ในหน่วยความจำภายในภาพ ซีพียูจึงตรวจสอบแคชบล็อกหมายเลข 008₁₆ ของ Cache 0 และ Cache 1 ยังว่างเปล่า วงจรควบคุมแคชจึงนำคำสั่งนี้จากหน่วยความจำภายในภาพไปเก็บในแคช 0 บล็อกที่ 008₁₆ พร้อมกับส่งให้ซีพียูถอดรหัสและ PC ออฟเซตเปลี่ยนเป็นหมายเลข 030₁₆

(e) ออฟเซต 030₁₆ ตรงกับคำสั่ง ADD R0, R1, R2 ยังเก็บอยู่ในหน่วยความจำภายในภาพ ซีพียูจึงตรวจสอบแคชบล็อกหมายเลข 000₁₆ ของ Cache 0 และ Cache 1 เพราะเป็นตัวแทนของหน่วยความจำภายในภาพที่ 030₁₆ พบร่วมกับแคชบล็อกที่ 000₁₆ ของ Cache 1 ยังว่าง วงจรควบคุมแคชจึงนำคำสั่งนี้จากหน่วยความจำภายในภาพไปเก็บในแคช 1 บล็อกที่ 000₁₆ พร้อมกับส่งให้ซีพียูถอดรหัสและ PC ออฟเซต



รูปที่ 5.11: การทำงานของแคชลำดับที่ 1 แคชคำสั่ง (Instruction Cache) ชนิด 2-Way Set Associative เมื่อ PC เฟห์ซคำสั่งที่ PC ออฟเซตเท่ากับ (e) 030₁₆, (f) 034₁₆, (g) 00C₁₆, (h) 010₁₆

เปลี่ยนเป็นหมายเลข 034₁₆

(f) ออฟเซต 034₁₆ ตรงกับคำสั่ง BX LR ยังเก็บอยู่ในหน่วยความจำภายใน CPU ซีพียูจึงตรวจสอบแคชบล็อกหมายเลข 004₁₆ ของ Cache 0 และ Cache 1 พบร่วมกับแคชบล็อกที่ 004₁₆ ของ Cache 1 ยังว่าง วงจรควบคุมแคชจึงนำคำสั่งนี้จากหน่วยความจำภายใน CPU ไปเก็บในแคช 1 บล็อกที่ 004₁₆ พร้อมกับส่งให้ซีพียูถอดรหัสและ PC ออฟเซตเปลี่ยนเป็นหมายเลข 00C₁₆ เพื่อรีเทิร์นกลับ (Return)

(g) ออฟเซต 00C₁₆ ตรงกับคำสั่ง CMP R0, 10 ยังเก็บอยู่ในหน่วยความจำภายใน CPU ซีพียูจึงตรวจสอบแคชบล็อกหมายเลข 00C₁₆ ของ Cache 0 และ Cache 1 พบร่วมกับแคชบล็อกที่ 00C₁₆ ของ Cache 0 ยังว่าง วงจรควบคุมแคชจึงนำคำสั่งนี้จากหน่วยความจำภายใน CPU ไปเก็บในแคช 0 บล็อกที่ 00C₁₆ พร้อมกับส่งให้ซีพียูถอดรหัสและ PC ออฟเซตเปลี่ยนเป็นหมายเลข 010₁₆

(h) ออฟเซต 010₁₆ ตรงกับคำสั่ง BLT Loop ยังเก็บอยู่ในหน่วยความจำภายใน CPU ซีพียูจึงตรวจสอบแคชบล็อกหมายเลข 000₁₆ ของ Cache 0 และ Cache 1 พบร่วมกับไม่ว่างทั้งคู่ วงจรควบคุมแคชจึงนำคำสั่งนี้จากหน่วยความจำภายใน CPU ไปเก็บในแคช 1 บล็อกที่ 00C₁₆ ด้วยการเดาสุ่ม (Random) พร้อมกับส่งให้

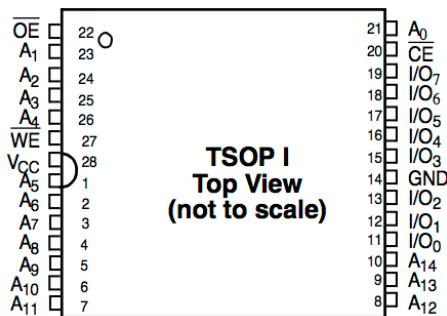
ซึ่งพิจารณาผลการหัตถและ PC ออฟเซต จะเปลี่ยนเป็นหมายเลข 000_{16} หรือไม่ขึ้นอยู่กับค่าของ RO ว่าจะเงื่อนไข LT (Less Than) ซึ่งจะแปลเป็นเงื่อนไข $RO < 10$ เป็นจริง หรือ เป็นเท็จ

- หากเงื่อนไข $RO < 10$ เป็นจริง PC ออฟเซตจะเปลี่ยนเป็นหมายเลข 000_{16} กระบวนการต่าง ๆ จะวนซ้ำคำสั่งที่อยู่ในแคชจะถูกอ่านซ้ำ เรียกว่า แคชชิต ทำให้เวลาในการเฟทช์คำสั่งสั้นลงเมื่อเทียบกับรอบแรก และหากมีการวนรอบมากกว่า 10 รอบ เช่น 1000 รอบ เวลาที่มีสินรอบแรกจะยิ่งคุ้มค่ามาก
- หากเป็นเท็จ PC ออฟเซตจะเปลี่ยนเป็นหมายเลข 014_{16}

ผู้อ่านจะสังเกตเห็นว่า แคชคำสั่งชนิด SA มีการทำงานคล้ายกับแคชคำสั่งชนิด DM แต่มีจำนวน 2 เว็บนั่นคือ แคชหมายเลขบล็อกเดียว กันมี 2 ทางเลือก คือ เวบ 0 และเวบ 1 ขึ้นอยู่กับว่า เวบใดว่าง หากไม่ว่างทั้งคู่ แคชจะตัดสินใจให้เขียนทั้งสอง เรียกว่า **Cache Replacement Algorithm** ที่มา: [Tanenbaum and Austin \(2012\)](#) หากใช้อัลกอริธึมการตัดสินใจที่เรียกว่า **Least Recently Used** แคชจะเขียนทั้งหมดในเวบที่ไม่ได้ใช้งานล่าสุด จึงเห็นได้ว่า แคชชนิด SA นี้ เป็นการขยายการทำงานของแคชชนิด DM ให้มีความยืดหยุ่นมากขึ้น ปัจจุบัน ซึ่งพิจารณาใช้แคชคำสั่ง/ข้อมูลชนิด SA ขนาด $N = 8-16$ เวบ เพื่อประสิทธิภาพที่ดีขึ้น

หลักการของแคชมีบทบาทต่อประสิทธิภาพของระบบโดยองค์รวม แคชชนิด N-way Set Associative ได้รับความนิยม เนื่องจากมีโครงสร้างเหมือนกับแคชชนิด DM และมีความสามารถคล้ายแคชชนิด Fully Associative มีการนำหลักการ Locality Principle ที่มา: [Tanenbaum and Austin \(2012\)](#) มาประยุกต์ใช้กับหน่วยความจำประเภทต่าง ๆ และหลากหลาย รวมถึงหลักการเรอർชาลเมโมรี เรอർชาลเมโมรีอาศัยการทำงานร่วมกันระหว่าง ฮาร์ดแวร์ และระบบปฏิบัติการ เพื่อเพิ่มลำดับชั้นในการบริหารจัดการหน่วยความจำทุกลำดับชั้นดังได้กล่าวไปแล้ว

5.4 หน่วยความจำชนิด静态 RAM (Static RAM: SRAM)



รูปที่ 5.12: ตัวถังแบบ TSOP ของชิปหน่วยความจำชนิด static RAM Cypress 62256 ที่มา: [Cypress Semiconductor, Corp. \(2002\)](#)

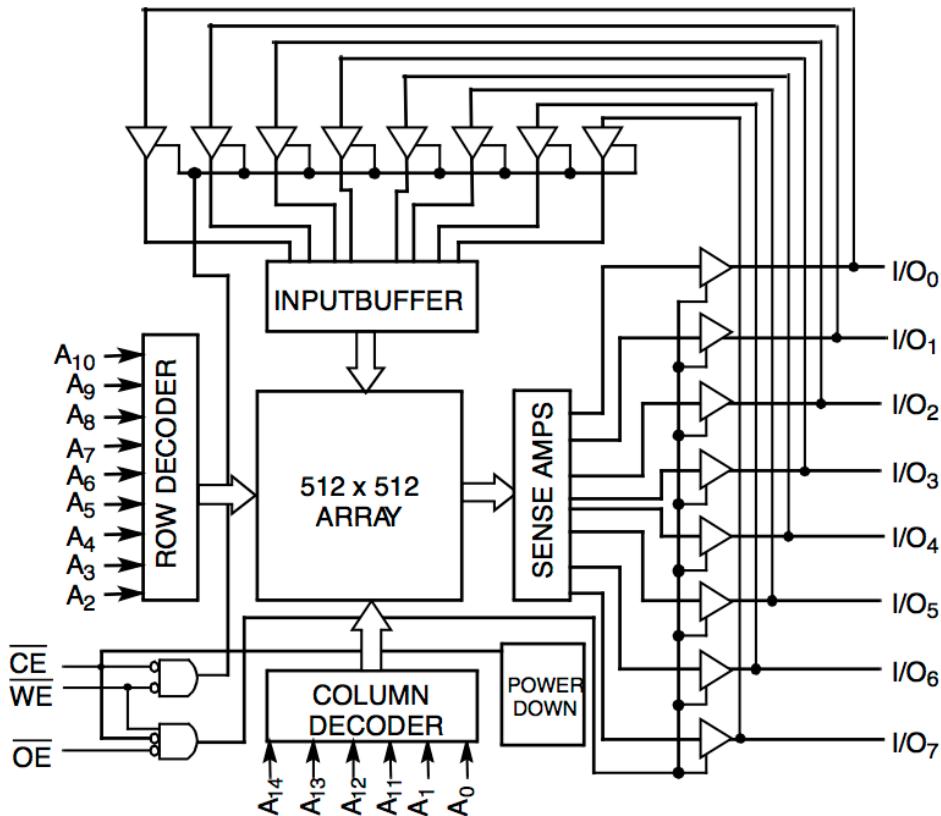
แคชลำดับที่ 1 และ 2 ในชิป BCM2837/BCM2711 และ แคชลำดับที่ 3 ในชิปซีพียูอื่น ๆ สร้างจากหน่วยความจำ static RAM หน่วยความจำ SRAM มีโครงสร้างที่ไม่ตับซ้อนและสามารถออกแบบให้ผลิตพร้อมกับวงจรทรานซิสเตอร์ในซีพียู นอกจากนี้ หน่วยความจำ static RAM นี้ยังนิยมใช้งานเป็นหน่วยความจำหลักภายในชิปไมโครคอนโทรลเลอร์ ที่ต้องการสมรรถนะต่ำถึงปานกลาง โดยมีความจุหลายขนาด ตั้งแต่ 16 KiB (kilobyte) ถึงหลายเมบิไบต์ ผู้เขียนขอใช้หน่วยความจำ static RAM (Static RAM: SRAM) หมายเลขอารบิก CY62256 เป็นกรณีศึกษา ชิป CY62256 ใช้เทคโนโลยีการผลิตชนิด CMOS ในปี ค.ศ. 2002 เอกสารคุณลักษณะ (Datasheet) ของชิปหน่วยความจำ static RAM นี้ ที่มา: [Cypress Semiconductor, Corp. \(2002\)](#) สามารถค้นเจอที่ ecee.colorado.edu ถึงแม้ว่าชิป CY62256 จะเป็นชิปที่เก่าแล้วแต่ชิปมีลักษณะเรียบง่ายและโดยดีเด่น ดังนี้

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

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

ชิปอาจอยู่ในตัวถังแบบ TSOP (Thin Small Outline Package) ในทางกายภาพ เพื่อให้เหมาะสมสำหรับการบัดกรีเพื่อยืดตัวถังชิปบนแผ่นวงจรพิมพ์ (Surface Mount) ผู้อ่านสามารถค้นคว้าเรื่องการห่อหุ้มชิปด้วยแพ็กเกจชนิด TSOP และอื่น ๆ ได้ที่ [wikipedia](https://en.wikipedia.org) ซึ่งแต่ละชนิดมีข้อได้เปรียบและเสียเปรียบแตกต่างกันไป

5.4.1 โครงสร้างภายในของชิปหน่วยความจำ static แต่ติกแรม



รูปที่ 5.13: โครงสร้างของหน่วยความจำชนิด static แต่ติก Cypress 62256 ที่มา: [Cypress Semiconductor, Corp. \(2002\)](#)

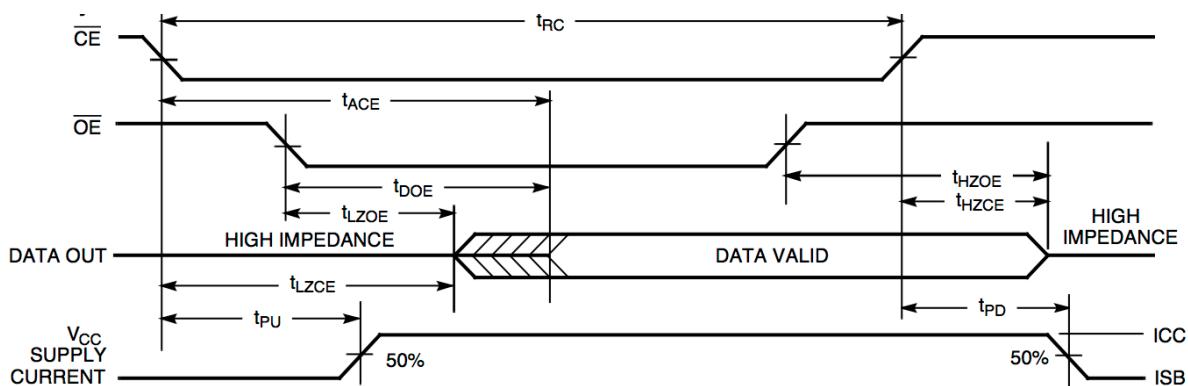
โครงสร้างของชิป CY62256 จัดเรียงเป็น $2^5 \times 2^{10} \times 2^3 = 2^{9+9}$ หรือ 512×512 บิต ซึ่งจะตรงกับ อาร์เรย์ของเซลล์หน่วยความจำในรูปที่ 5.13

- ซีพียูเริ่มต้นขั้นตอนการอ่านข้อมูล โดยเปลี่ยนสัญญาณที่ขาดังต่อไปนี้
 - ขาควบคุม $\overline{WE}=1$, ขา $\overline{CE}=0$ และ $\overline{OE}=0$
 - ขาแอดเดรส $A_0 - A_{14}$ ทั้งหมด 15 ขา รับสัญญาณแอดเดรส เพื่อป้อนให้วงจรถอดรหัสแนว นอน (Row Decoder) และแนวคอลัมน์ (Column Decoder) เพื่อสร้างสัญญาณจำนวน 512 เส้นในแนวนอนและ 64 เส้นในแนวตั้ง เพื่อเลือกบิตเซลล์จำนวน 8 บิตที่ต้องการพร้อมกันใน อาร์เรย์
 - มองุลขยายสัญญาณความไวสูง (Sense Amplifier) เป็นวงจรแยกหลอกขยายสัญญาณ ใช้ สำหรับขยายสัญญาณที่ส่งออกมาจากอาร์เรย์หน่วยความจำ เพราะสัญญาณที่ได้มีระดับโวล เตจต่ำมาก รายละเอียดเพิ่มเติมที่ [wikipedia](#)
 - สัญญาณที่ผ่านการขยาย และตรวจจับแล้ว จะกล้ายเป็นข้อมูลดิจิทัลผ่านเกตไดรเวอร์สาม สถานะให้ลอกไปยังขา I/O_0 จนถึง I/O_7 พร้อมกันทั้ง 8 บิต

- ซีพียูเริ่มต้นกระบวนการเขียนข้อมูล โดยเปลี่ยนสัญญาณที่ขาตั้งต่อไปนี้
 - ขาควบคุม $\overline{WE}=0$, ขา $\overline{CE}=0$ และขา $\overline{OE}=1$
 - ซีพียูจะส่งข้อมูลจำนวน 8 บิตให้เหล่าทางขา I/O_0 จนถึง I/O_7 ผ่านเกตไดรเวอร์สามสถานะมาพักใน
 - มอดูลบัฟเฟอร์ขาเข้า (Input Buffer) เพื่อพักข้อมูลชั่วคราวและรอเขียนในบิตเซลล์ที่ต้องการพร้อมกันทั้ง 8 บิต
 - ขาแอดเดรส $A_0 - A_{14}$ ทั้งหมด 15 ขา รับสัญญาณแอดเดรสให้วงจรลดรหัสแนวนอน (Row Decoder) และแนวคอลัมน์ (Column Decoder) ทำหน้าที่สร้างสัญญาณจำนวน 512 เส้นในแนวนอนและแนวแนวนอนตั้งจำนวน 64 เส้น เพื่อเลือกบิตเซลล์ที่ต้องการเขียนในอาร์เรย์พร้อมกันทั้ง 8 บิต

แต่ละบิตเซลล์ในอาร์เรย์ของหน่วยความจำ สแตติก แรม ประกอบด้วย ทรานซิสเตอร์จำนวนหนึ่งเชื่อมต่อกันคล้ายวงจรฟลิปฟล็อป (Flip-Flop) และเกตอินเวอร์เตอร์ (Inverter) จัดเรียงเป็นลูปปั่นกลับ (Feedback Loop) เพื่อเก็บรักษาข้อมูลโดยไม่ต้องใช้กระบวนการอื่นเพิ่มเติม และใช้เวลาเข้าถึงรวดเร็ว ทำให้แต่ละบิตเซลล์ใช้พื้นที่ซิลิกอน (Silicon Area) คิดเป็นจำนวนทรานซิสเตอร์มากกว่าเท่ากับทรานซิสเตอร์ 6 ตัวขึ้นไปต่อความจุข้อมูล 1 บิต ผู้อ่านสามารถค้นควารายละเอียดเพิ่มเติมได้ที่ [wikipedia](#)

5.4.2 การทำงานของสแตติกแรม: ขบวนการอ่านข้อมูล

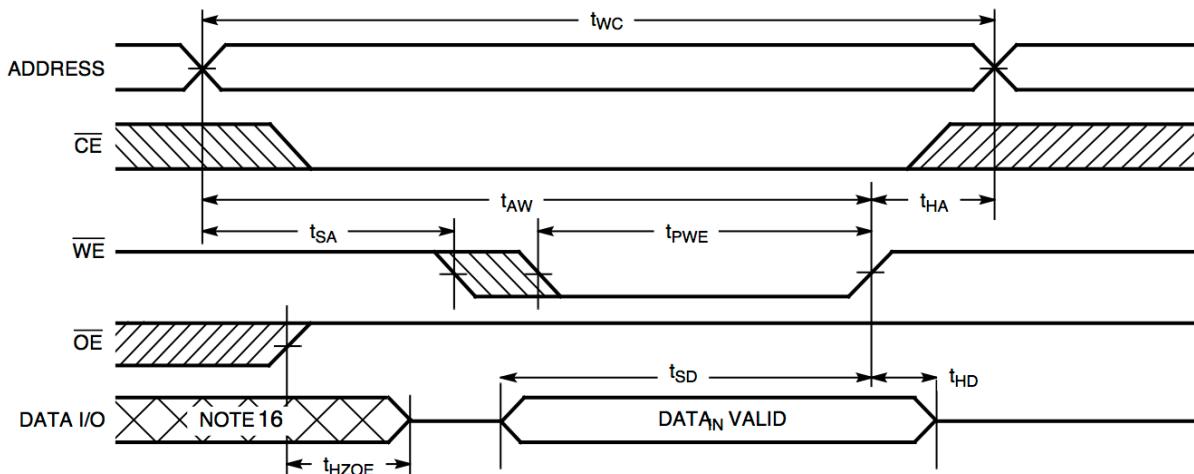


รูปที่ 5.14: ไดอะแกรมเวลา (Timing Diagram) สำหรับการอ่าน (Read) ข้อมูลจากหน่วยความจำชนิดสแตติกแรม ที่มา: [Cypress Semiconductor, Corp. \(2002\)](#)

ผู้อ่านสามารถทำความเข้าใจขบวนการอ่านข้อมูลจากไดอะแกรมเวลาในรูปที่ 5.14 หมายเหตุ แกนนอนเป็นแกนเวลาเริ่มต้นจากซ้ายสุดของแกน การอ่านเริ่มต้นจากที่ซีพียูส่งแอดเดรส $A_0 - A_{14}$ เป็นอย่างหน่วยความจำ แล้วจึงเปลี่ยนขาสัญญาณ \overline{CE} และสัญญาณ \overline{OE} จาก "1" เป็น "0" เพื่อกระตุ้นการทำงานของหน่วยความจำ ซีพียูจะต้องรอเป็นระยะเวลาอย่างน้อย t_{ACE} ข้อมูล ณ แอดเดรสนั้นจะปรากฏบนบัสข้อมูล $I/O_0 - I/O_7$ การอ่านค่าข้อมูลนี้ช่วงเวลาต่าง ๆ กำกับดังนี้

- t_{ACE} เรียกว่า เวลาเข้าถึงข้อมูล หรือ Access Time โดยเริ่มนับจากเมื่อสัญญาณ \overline{CE} เปลี่ยนเป็น 0 จนข้อมูลที่ถูกต้องปรากฏ
- t_{RC} เรียกว่า คาบเวลาที่สั้นที่สุดในการอ่านข้อมูลจาก static RAM (Read Cycle Time) อย่างต่อเนื่องโดย $t_{RC} > t_{ACE}$
- 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 การทำงานของ static RAM: ขบวนการเขียนข้อมูล



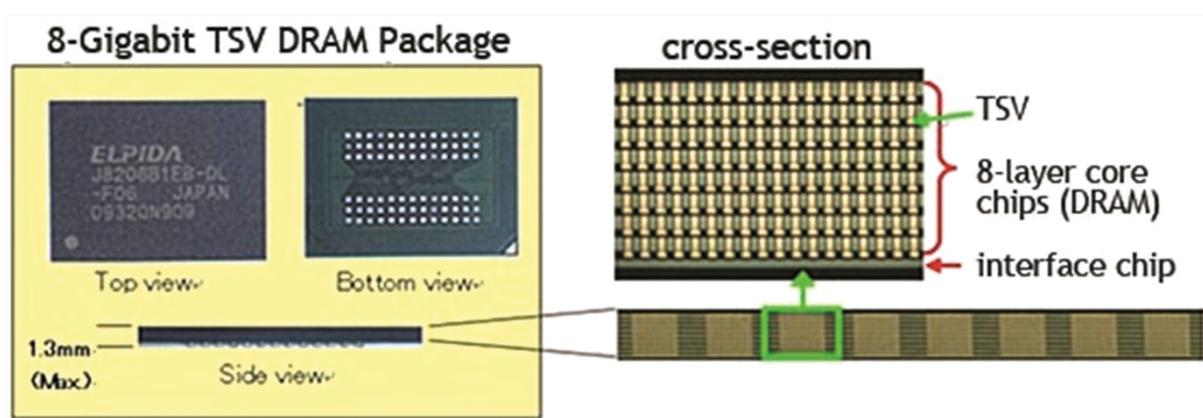
รูปที่ 5.15: ไดอะแกรมเวลา (Timing Diagram) สำหรับการเขียน (Write) ข้อมูลในหน่วยความจำชนิด static RAM ที่มา: [Cypress Semiconductor, Corp. \(2002\)](#)

ผู้อ่านสามารถทำความเข้าใจขบวนการเขียนข้อมูลของหน่วยความจำ static RAM จากไดอะแกรมเวลาในรูปที่ 5.15 หมายเหตุ แก่นอนเป็นแกนเวลาเริ่มต้นจากชัยสุดของแกน การเขียนเริ่มต้นจากที่ซีพียูส่ง แอดเดรส $A_0 - A_{14}$ ไปยังหน่วยความจำ แล้วจึงเปลี่ยนขาสัญญาณควบคุม \overline{CE} และ \overline{WE} จาก "1" เป็น "0" เพื่อกระตุ้นการทำงานของหน่วยความจำ ซีพียูจะต้องส่งข้อมูลบนบัสข้อมูล $I/O_0 - I/O_7$ เป็นระยะเวลาอย่างน้อย t_{SD} เพื่อยืนยันว่าข้อมูลที่ถูกต้องจะเขียน ณ แอดเดรสนั้น การเขียนค่าข้อมูลมีช่วงเวลาต่าง ๆ กำกับ ดังนี้

- t_{SD} เรียกว่า เวลาที่จะตรึงข้อมูลให้เสถียร (Data Stable Time) ก่อนสัญญาณ Write Enable (\overline{WE}) เปลี่ยนเป็น 1

- t_{HD} เรียกว่า เวลาที่จะต้องข้อมูลให้เสถียร (Data Hold Time) ต่อจาก t_{SD} เมื่อ Write Enable (\overline{WE}) = 1 แล้ว
- t_{WC} เรียกว่า ค่าเวลาที่สั้นที่สุดในการเขียนข้อมูลในหน่วยความจำสแตติกแรม (Write Cycle Time) อย่างต่อเนื่อง
- t_{AW} เรียกว่า เวลาเข้าถึงสำหรับการเขียน (Access Write Time)

5.5 หน่วยความจำหลักชนิด SDRAM



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

ผู้ผลิตเครื่องคอมพิวเตอร์โน๊ตบุ๊ก โทรศัพท์เคลื่อนที่สมาร์ตโฟน และอุปกรณ์พกพาต่างนิยมออกแบบติดตั้งชิปหน่วยความจำ SDRAM บน เมนบอร์ด (Main Board) เช่นเดียวกับบอร์ด Pi3/Pi4 เพื่อลดขนาดและปริมาตรของเครื่องให้มีขนาดเท่ากับบอร์ดเครดิต แต่ในเทคโนโลยีขบวนการผลิต SDRAM ยังไม่สามารถรวมกับขบวนการผลิตไมโครโปรเซสเซอร์ได้ ผู้ผลิตจึงจำเป็นต้องผลิตชิป SDRAM แยกต่างหาก

ชิป DDR2 SDRAM กรณีศึกษาที่ใช้บนบอร์ด Pi3/Pi4 ซึ่งได้อธิบายเบื้องต้นแล้วในหัวข้อที่ 3.1.2 ผลิตโดยบริษัท Elpida ประเทศญี่ปุ่นในปี ค.ศ. 2014 ที่มา: [Micron Technology, Inc. \(2014\)](http://Micron Technology, Inc. (2014)) และต่อมาบริษัท มีครอน เทคโนโลยี ประเทศไทยได้เข้าถือหุ้นแทน ชิป DDR SDRAM นี้ใช้โครงสร้างแพ็คเกจและขนาด BGA (Ball Grid Array) จำนวน 168 ขาทางด้านล่าง เพียงด้านเดียว ตัวชิปมีขนาด 12mm.x12mm.x0.8mm. ผู้ออกแบบบอร์ด Pi3/Pi4 ยึดติดชิป SDRAM อยู่ใต้บอร์ด ตามรูปที่ 3.4 ผู้อ่านจะสังเกตได้ว่าแพ็คเกจหรือตัวถังชนิด BGA นี้จะใช้พื้นที่บนแผ่นวงจรพิมพ์เล็กเท่ากับตัวชิป ช่วยประหยัดพื้นที่บนบอร์ดและสามารถเชื่อมตอกับชิป BCM2837/BCM2711 ใช้ระยะทางสั้นลง ทำให้ชิปทั้งสองสามารถทำงานด้วยสัญญาณคลื่นความถี่สูงสุดตามคุณลักษณะเฉพาะ (Specification) เนื่องจากใช้เทคโนโลยีขบวนการผลิตที่ทันสมัยกว่า ในขณะที่ชิปหน่วยความจำสแตติกแรมใช้ตัวถังชนิด TSOP ในรูปที่ 5.12 ซึ่งเก่ากว่ามีข้อจำกัดด้านข้างจึงต้องการพื้นที่บนแผ่นวงจรพิมพ์มากขึ้น

ชิป SDRAM มีความจุข้อมูลต่อชิปสูงมาก เนื่องจากใช้พื้นที่ชิปเท่ากับทรานซิสเตอร์เพียง 1-2 ตัวต่อความจุข้อมูล 1 บิต แต่มีข้อเสีย คือ ใช้เทคโนโลยีและกระบวนการผลิตที่ซับซ้อนกว่าหน่วยความจำสแตดติก แรม ผู้ใช้ต้องการความเร็วมั่นในตัวอุปกรณ์สูง และความถี่คลื่อกสูง เช่นกัน ทำให้ต้นทุนต่อความจุข้อมูลหนึ่งบิตของ SDRAM สูงกว่าอุปกรณ์เก็บรักษาข้อมูล ในบทที่ 7

ผู้อ่านสามารถทำความเข้าใจโครงสร้างเชิงกายภาพที่ซับซ้อนของ SDRAM ชนิดนี้จากรูปที่ 5.16 ซึ่งเป็นชิป SDRAM ความจุขนาด 8 กิกะบิต (8×2^{30} บิต) โดยบริษัท Elpida เช่นเดียวกัน ภาพถ่ายด้านบน (Top view) มีชื่อเรียกว่า ภาพถ่ายด้านล่างแสดงขาสำหรับเชื่อมต่อ เข้าสู่ภายนอกในชิปจำนวน 2 แฉก ๆ ละ 39 ขา และรอยปากตรงมุมขวาล่าง เมื่อพลิกคว่ำลง จะตรงกับจุดวงกลมด้านซ้ายล่างของภาพถ่ายด้านข้างของชิป (Side view) มีความหนาเท่ากับ 1.3 มิลลิเมตร ผู้อ่านจะมองเห็นขาที่ยื่นออกมาใต้ตัวถัง เมื่อขยายภาพตัดขวาง (cross-section) ให้ใหญ่ขึ้น แผ่นซิลิกอนแต่ละชั้นมีเชื่อมต่อสัญญาณในแนวราบภายนอกในแผ่นหรือชั้นเดียวกัน ภาพตัดขวางที่ขยายทางด้านขวา มีสุดจะมองเห็นเป็น 8 ชั้น (8-layer core) แสดงให้การเชื่อมต่อระหว่างแผ่นซิลิกอนเข้าด้วยกันในแนวตั้งโดยใช้เทคโนโลยี TSV (Through Silicon Via) ที่มา: [wikipedia](#) ซึ่งเชื่อมสัญญาณควบคุม สัญญาณคลื่อก ไฟเลี้ยง กราวน์ด และอื่น ๆ การเชื่อมต่อสัญญาณครบทั้งสามมิตินี้ช่วยให้ประยุกต์พื้นที่ เชื่อมต่อสัญญาณเหล่านี้บนแพนซิลิกอนในแนวราบ

5.5.1 โครงสร้างภายในของชิป SDRAM

ในรูปที่ 5.17 โครงสร้างภายในชิปหน่วยความจำ SDRAM ชนิด DDR2 ชิปกรณ์ศึกษา ประกอบด้วย ตาย (Die) 2 ชิ้น แต่ละชิ้นมีบัสข้อมูลขนาด 16 บิตเรียงต่อกันเป็น 32 บิต ที่มา: [Micron Technology, Inc. \(2014\)](#) แต่ละตายประกอบด้วย แฟลชเซลล์หน่วยความจำชนิด SDRAM จำนวน 8 ชั้น หรือ 8 แบงค์ (Bank) ๆ ละ 32×2^{20} เซลล์ \times 16 บิต คิดเป็น 2 ตาย \times 8 แบงค์ $\times 32 \times 2^{20}$ เซลล์ \times 16 บิต หรือ $2^1 \times 2^3 \times 2^5 \times 2^{20} \times 2^4 = 2^{33} = 2^3 \times 2^{30} = 8$ กิกะบิตหรือ 1 กิกะไบต์ (GiB)

แบ่งค์ที่ 0 ถึง 7 แต่ละ แบ่งค์ประกอบด้วยวงจรลดรหัสและドレス (Address Decoder) ในแนวอน (Row Decoder) และแนวตั้ง (Column Decoder) ภายในชิปประกอบด้วยขาสัญญาณต่าง ๆ เรียงตามลำดับความสำคัญ ดังนี้

- **CS_n** 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 ใช้สำหรับ เปิด/ปิดการทำงานของคล็อกที่จ่ายให้กับชิป SDRAM นี้ ซึ่งมีความสามารถควบคุมสัญญาณ CKE=0 เพื่อ พักการใช้งาน SDRAM ชั่วคราวเพื่อช่วยประหยัดพลังงาน
 - **CK (Clock)** และ **CK# (Clock Not หรือ Clock bar)** คือ สัญญาณคล็อกสองสัญญาณที่มีเฟส (Phase) หรือขั้วตรงข้ามกัน เรียกว่า **คู่ดิฟเฟอเรนเชียล** (Differential Pair) หน่วยเป็นเมกะเอิร์ตซ์ สัญญาณคล็อกความถี่สูงสุด 400 เมกะเอิร์ตซ์

ชิป DDR4 ล่าสุดสำหรับคอมพิวเตอร์ตั้งโต๊ะความถี่สูงสุด มากกว่า 3,000 เมกะเอิร์ตซ์ และมีแหน้มเพิ่มขึ้นตามเทคโนโลยีการผลิตที่พัฒนาอย่างต่อเนื่อง ผู้อ่านสามารถค้นคว้าเพิ่มเติมได้ที่ [wikipedia](#)

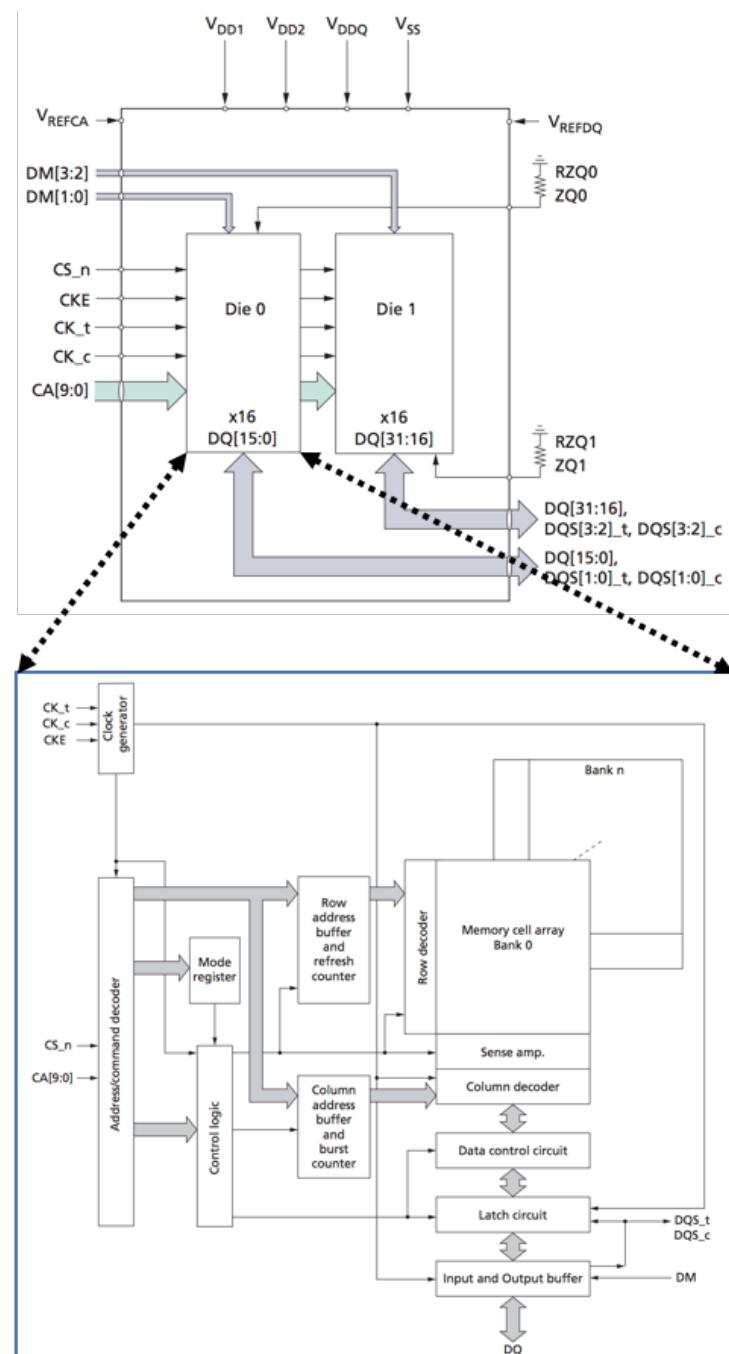
- **Command/Address** CA[0:9] ขนาด 10 บิต ใช้มัลติเพล็กซ์ (Multiplex) สัญญาณคำสั่ง (Command) และ ออดเดรส (Address) เพื่อรับคำสั่ง (Command) และสัญญาณ ออดเดรส (Address) ต่างหัวเวลา กัน
 - ชีพิยุจะส่งคำสั่ง (Command) ต่าง ๆ ดังนี้ Activate, **Burst Read**, **Burst Write**, Refresh, Power Down, Precharge และ Burst Terminate เป็นต้น เพื่อกำหนดโหมดการทำงานของหน่วยความจำ SDRAM ได้แก่ Power Up, Deep Power Down, Active, Idle, Reading, Writing, Precharging, Refreshing เป็นต้น
 - **สัญญาณ ออดเดรส แถว** (Row Address) จำนวน 14 บิต และ **ออดเดรส คอลัมน์** (Column Address) จำนวน 11 บิต จะพักเก็บในวงจรบีฟเฟอร์ เพื่อป้อนให้กับวงจร รถดอร์หัส (Decoder) คล้ายกับการทำงานของหน่วยความจำสแตติกแรมในหัวข้อที่ [5.13 การรับสัญญาณ ออดเดรส เกิดขึ้น ณ ขอบขาขึ้น และขอบขาลงของแต่ละสัญญาณ คลือก](#)
- **ดาต้าส ไตรบ** 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 ตำแหน่งต่อเนื่อง กัน
- **ดาต้ามาสก์** DM (Data Mask) [0:3] ขนาด 4 บิต ใช้สำหรับ มาสก์ (Mask) หรือปิด เพื่อควบคุมการเขียนข้อมูลแต่ละไบต์ โดย DM[0] ควบคุมไบต์ที่ 0, DM[1] ควบคุมไบต์ที่ 1 ตามลำดับ ทำให้การเขียนข้อมูลแต่ละไบต์เป็นอิสระจากกันได้

ผู้อ่านจะสังเกตเห็นว่า วงจรรอบ ๆ เชลล์หน่วยความจำ (Cell Array) มีลักษณะที่คล้ายกับ อาร์เรย์ของ เชลล์หน่วยความจำของสแตติกแรมในรูปที่ [5.13](#) ดังนี้

- วงจร รถดอร์หัส แนว ราบ (Row Decoder) สร้างสัญญาณ จำนวน $2^{14} = 16,384$ เส้น เรียกว่า สายบิต ไลน์ (Bit line) ในแนวราบ
- วงจร รถดอร์หัส แนว ตั้ง (Column Decoder) สร้างสัญญาณ จำนวน $2^{11} = 2048$ เส้น เรียกว่า สาย เวิร์ด ไลน์ (Word line) ในแนวตั้ง
- วงจรขยายสัญญาณ ความไวสูง (Sense Amplifier) ใช้สำหรับขยายสัญญาณ ในขบวนการอ่าน เช่น เดียวกับการอ่านของหน่วยความจำสแตติกแรม

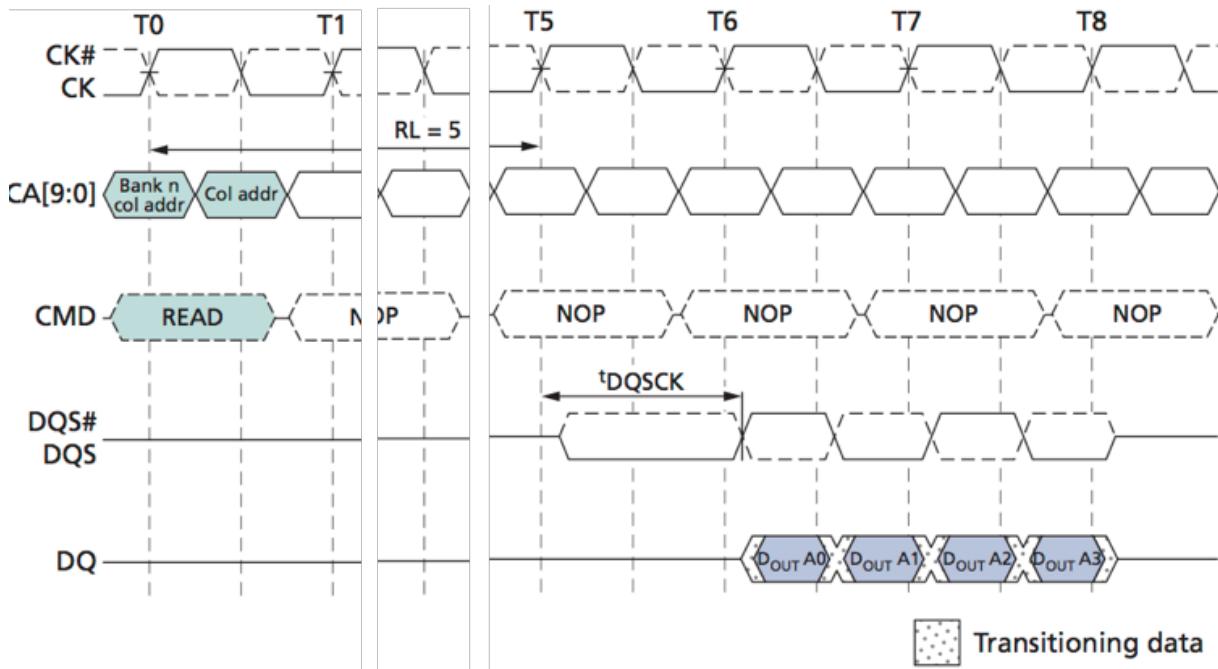
จุดตัด ระหว่าง สายบิต ไลน์ และ สาย เวิร์ด ไลน์ คือ ตำแหน่งของ บิต เชลล์ ที่ต้องการ จะอ่าน หรือ เขียน คล้ายกับ การทำงานของหน่วยความจำสแตติกแรม แต่ หน่วยความจำ SDRAM มีโครงสร้างของ บิต เชลล์

ที่แตกต่าง กล่าวคือ บิตเซลล์แต่ละบิตของ SDRAM ประกอบด้วยทรานซิสเตอร์ชนิด MOS จำนวน 2 ตัว ทรานซิสเตอร์ตัวที่หนึ่งทำหน้าที่เก็บประจุเมื่อมีอนค่าปานกลาง เมื่อค่าปานกลาง มีประจุ แทนข้อมูล 1 หรือไม่มีประจุ แทนข้อมูล 0 ทรานซิสเตอร์ตัวที่สองทำหน้าที่ควบคุมการอ่านและเขียนข้อมูล ผู้อ่านสามารถคลิก [ไดนามิกแรมขนาด 4x4 บิต เพื่อศึกษาโครงสร้างบิตเซลล์ของ SDRAM เพิ่มเติม](#)



รูปที่ 5.17: บล็อกไออะแกรมภายในชิปหน่วยความจำ SDRAM ชนิด DDR2 ประกอบด้วยダイ (Die) 2 ชิ้น แต่ละชิ้นมีบัสข้อมูลขนาด 16 บิตเรียงขานกันเป็น 32 บิต ที่มา: [Micron Technology, Inc. \(2014\)](#)

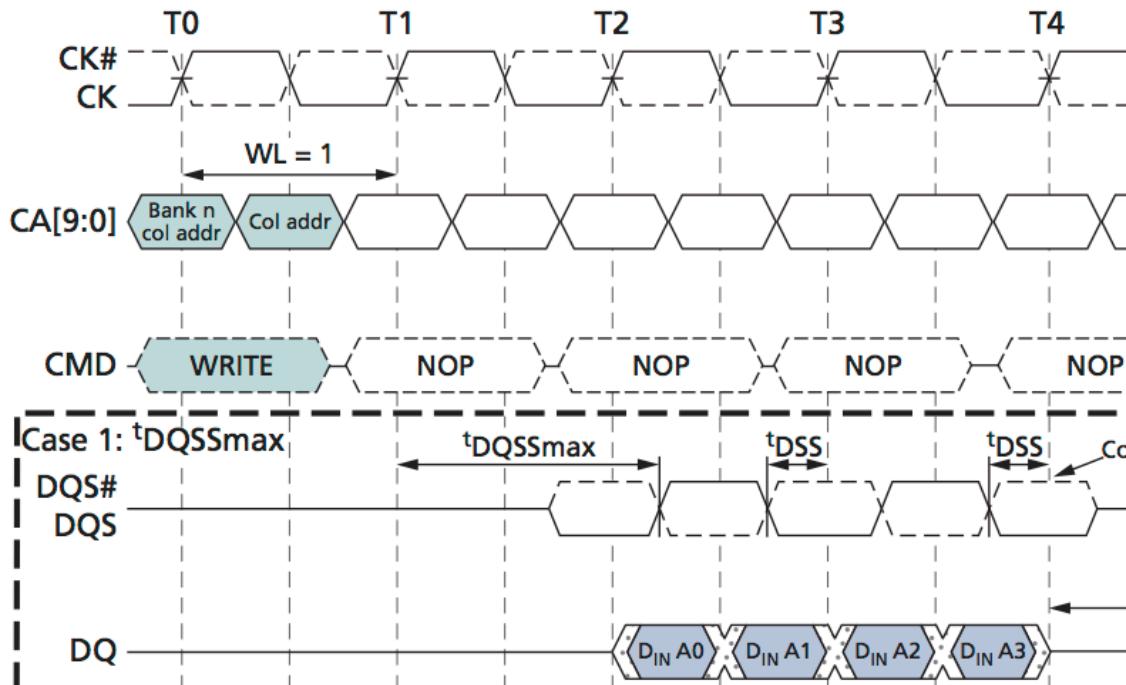
5.5.2 การทำงานของ SDRAM: อ่านและเขียน



รูปที่ 5.18: ไดอะแกรมเวลา (Timing Diagram) สำหรับการอ่านต่อเนื่องของหน่วยความจำ SDRAM รุ่น Elpida B8132B4PB-8D-F คำสั่ง Burst READ โดย RL (Read Latency) เท่ากับ 5 ไซเกิล BL (Burst Length) เท่ากับ 4 ตำแหน่ง หมายเหตุ รูปมีการตัดต่อเพื่อเน้นบริเวณที่สำคัญ, NOP=No Operation ที่มา: [Micron Technology, Inc. \(2014\)](#)

การทำงานของหน่วยความจำ SDRAM ผลิตโดยบริษัท Elpida รุ่น B8132B4PB-8D-F มีความหลากหลาย แต่ผู้อ่านสามารถทำความเข้าใจคำสั่งที่ใช้งานบ่อยที่สุด คือ คำสั่ง Burst READ และ คำสั่ง Burst Write ดังนี้

คำสั่ง **Burst READ** เริ่มต้นโดย ทำให้สัญญาณ $CS\#=0$, $CA0=1$, $CA1=0$ และ $CA2=1$ ณ ขอบขาขึ้นของสัญญาณคล็อก รูปที่ 5.18 แสดงถึงไดอะแกรมเวลาสำหรับการอ่านข้อมูลของหน่วยความจำชนิด DDR2 แบบชิงโครนัสซึ่งต้องทำงานด้วยขอบสัญญาณทั้งขาขึ้นและขาลงของสัญญาณคล็อก ในรูปนี้เป็นการทำงานตามคำสั่ง Burst READ ด้วยค่า RL (Read Latency) เท่ากับ 5 ไซเกิล โดยนับจาก T1 ถึง T5 เป็นการนับจำนวนคล็อกหรือจำนวนไซเกิล หรือเวลาของสัญญาณคล็อก การนับเริ่มจากขอบขาขึ้นของคำสั่ง READ หมายเหตุ รูปมีการตัดต่อเพื่อเน้นบริเวณที่สำคัญ และ อ่านข้อมูลต่อเนื่องเป็นจำนวน BL (Burst Length) = 4 ตำแหน่ง เรียงติดกันโดยเริ่มจากแอดдресที่น้อยกว่า (A0) วงจรจะใช้การเปลี่ยนแปลงของสัญญาณ DQS (Data Strobe) เพื่อจิงโครainซ์กับการอ่านข้อมูล ที่มา: [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, Inc. \(2014\)](#)

ตัวอย่าง สำหรับการเขียนต่อเนื่องของหน่วยความจำ Elpida รุ่น B8132B4PB-8D-F ด้วยคำสั่ง Burst WRITE ในรูปที่ 5.19 โดย WL (Write Latency) เท่ากับ 1 ไซเกล และ 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 การอ่านข้อมูลที่อยู่ในบิตเซลล์ต่าง ๆ และเขียนซ้ำที่บิตเซลล์เดิม เพื่อป้องกันไม่ให้ประจุที่เก็บอยู่ในบิตเซลล์ต่าง ๆ รั่วไหลหายไป เนื่องจากเซลล์ต่าง ๆ ที่ใช้เก็บข้อมูล ทำหน้าที่มี ('1') หรือไม่มี ('0') ประจุไฟฟ้า สำหรับบิตเซลล์ที่มีประจุ ๆ เหล่านี้อาจรั่วไหลหายไปเมื่อเวลาผ่านไปไม่กี่มิลลิวินาที เมื่อจำนวนประจุลดลง ทำให้การแยกแยะระหว่างบิตเซลล์ที่มีและไม่มีประจุยากขึ้น และอาจทำให้ตัวความไม่ถูกต้องและเกิดข้อผิดพลาดในการอ่านข้อมูล

วงจรควบคุมจะส่งคำสั่งรีเฟรชทุก ๆ 32 มิลลิวินาที หากมีการรีเฟรชดำเนินการอยู่ การอ่านหรือเขียนหน่วยความจำจะต้องหยุดรอ เพื่อให้ขบวนการรีเฟรชนั้นเสร็จสิ้น ในทำนองเดียวกัน หากมีการอ่านหรือเขียนข้อมูลจริงอยู่ การรีเฟรชจะต้องหยุดรอ ก่อน เพื่อให้ขบวนการอ่านหรือเขียนนั้นเสร็จสิ้น

หน่วยความจำสแตติกแรมไม่ต้องมีการรีเฟรชข้อมูล เนื่องจากการจัดเก็บข้อมูลใช้วิธีการเก็บข้อมูลที่แตกต่างกับหน่วยความจำ SDRAM ทำให้หน่วยความจำสแตติกแรมมีสมรรถนะและประสิทธิภาพสูงกว่า

แต่ต้องใช้จำนวนทรานซิสเตอร์ต่อบิตเซลล์มากกว่า จึงทำให้ใช้พื้นที่บนแผ่นซิลิกอนต่อความจุข้อมูล 1 บิตใหญ่กว่า เช่นกัน ผู้อ่านสามารถค้นคว้าเพิ่มเติมได้ที่ [wikipedia](#)

5.6 สรุปท้ายบท

หน่วยความจำลำดับชั้นอาศัยเทคโนโลยีหน่วยความจำหลายชนิด หลายขนาด ความจุข้อมูลเข้าด้วยกัน ยกตัวอย่างเช่น

- เทคโนโลยี SRAM เป็นหน่วยความจำที่ใช้เวลาเข้าถึงข้อมูลรวดเร็ว แต่ใช้พื้นที่บนชิปซีพียูต่อความจุข้อมูล 1 บิตสูงมาก จึงใช้งานเป็นรีจิสเตอร์ และ แคชลำดับต่าง ๆ
- เทคโนโลยี SDRAM เป็นหน่วยความจำความจุข้อมูลสูงกว่า แต่ต้องการเวลาเข้าถึงนานกว่าหน่วยความจำสแตติกแรมนำมาใช้งานเป็นหน่วยความจำหลัก
- เทคโนโลยีหน่วยความจำแฟลชเป็นหน่วยความจำที่มีความจุข้อมูลสูงกว่าแต่เวลาเข้าถึงข้อมูลนานกว่า SDRAM นำมาใช้งานเป็นอุปกรณ์เก็บรักษาข้อมูล รายละเอียดเพิ่มเติมในบทที่ 7

เพื่อให้คอมพิวเตอร์มีความจุข้อมูลเพียงพอ และตอบสนองต่อความต้องการใช้งานระบบโดยเฉลี่ยได้รวดเร็วขึ้น โดยการผสานจุดเด่นของหน่วยความจำแต่ละชนิดเข้าด้วยกัน และช่วยประยุกต์ต้นทุนของระบบ

เนื้อหาในบทนี้ว่า ด้วยเรื่องของหน่วยความจำผนวกกับความรู้ในบทก่อนหน้า ผู้เขียนจึงขออุปมาอุปมาภัยการทำงานของคอมพิวเตอร์โดยรวม ดังนี้

- ข้อมูล คือ **วัตถุดิบสำหรับประกอบอาหาร** อาจมาจากแหล่งข้อมูลหลาย ๆ แหล่ง เช่น ผู้ใช้กรอกข้อมูลในโปรแกรมไฟล์ข้อมูลในหน่วยสำรอง เป็นต้น
- ซอฟต์แวร์ คือ **สูตรหรือขั้นตอนการประกอบอาหารอย่างละเอียด** เพื่อให้พ่อครัวประกอบอาหารให้สุกและมีรสชาติดี
- ซีพียู คือ **พ่อครัวและเครื่องครัวที่ประกอบอาหารตามสูตรเท่านั้น** ประกอบด้วย
 - ALU คือ อุปกรณ์เครื่องครัว เช่น มีด เครื่อง Gratophase หม้อ เป็นต้น
 - รีจิสเตอร์ คือ งานหรือที่พักอาหารที่ปูร่องค้างไว้ เพื่อรอทำให้เป็นอาหารที่ปูร่องสำเร็จต่อไป
 - วงจรลื้น ๆ เช่น แคชลำดับต่าง ๆ มีหน้าที่เสริมการทำงานของซีพียูให้มีประสิทธิภาพดียิ่งขึ้น
- หน่วยความจำภายใน คือ **ชั้นวางวัตถุดิบ (ข้อมูล)** ขนาดเล็กที่ไม่สามารถเก็บวัตถุดิบได้นาน
- Information หรือ สารสนเทศ คือ **อาหารที่ประกอบและปูร่องสำเร็จแล้ว** โดยอาศัยซีพียูและซอฟต์แวร์ที่กล่าวมาข้างต้น

- อุปกรณ์เก็บรักษาข้อมูล คือ ตู้แช่เย็นขนาดใหญ่ที่สามารถบรรจุวัตถุติดไฟได้ปริมาณมาก และระยะเวลา หากต้องการเก็บข้อมูลหรือวัตถุติดไฟทันต่อไป ข้อมูลติดไฟหรือวัตถุติดไฟมักมีปริมาณมาก ๆ วัตถุติดไฟเหล่านี้จะเก็บในรูปของไฟล์ข้อมูล ซอฟต์แวร์สามารถอ่านหรือเขียนข้อมูลจากไฟล์ที่มีลักษณะต้องการ
- เครือข่ายอินเทอร์เน็ตช่วยถ่ายโอนข้อมูลจากเซิร์ฟเวอร์ต่าง ๆ บนในรูปของไฟล์ข้อมูล หรือ ในรูปของข้อมูลที่กระจายจากระยะทางเซนเซอร์ต่าง ๆ ด้วยเทคโนโลยี IoT (Internet of Thing) คือ เครือข่ายการขนส่งวัตถุติดไฟจากต่างประเทศมาปruzอาหาร โดยจะต้องพกเก็บในตู้แช่เย็นระหว่างทาง
- การคำนวณแบบขนาน (Parallel Computing) ต้องการซีพียูและ/หรือจีพียูหลาย ๆ แกนประมวลผล (พ่อครัวและเครื่องครัวจำนวนมาก) ช่วยกันประมวลผลข้อมูลปริมาณมาก ๆ ให้เสร็จสิ้นเร็วขึ้น ซึ่งบทที่ 8 จะอธิบายเรื่องนี้พอสังเขป

5.7 คำถ้ามท้ายบท

1. หน่วยความจำในบานี้ทั้งหมด สามารถเก็บข้อมูลได้หรือไม่หากไม่มีไฟเลี้ยง
2. เหตุใดเครื่องคอมพิวเตอร์จึงต้องมีอุปกรณ์เก็บรักษาข้อมูลคู่กับหน่วยความจำภายในภาพ เช่น SDRAM และ SRAM ในบานี้
3. จงเปรียบเทียบการอ่านของหน่วยความจำสารแตติกแรมและ DDR-SDRAM ในแง่มุมเหล่านี้
 - ขนาดของข้อมูลขั้นต่ำที่สามารถอ่านได้ หน่วยเป็นไบต์
 - ระยะเวลาเข้าถึง (Access Time) เพื่อรอให้ข้อมูลปรากฏบนบัสข้อมูล
4. จงเปรียบเทียบการเขียนของหน่วยความจำสารแตติกแรมและ DDR-SDRAM ในแง่มุมเหล่านี้
 - ระยะเวลาเข้าถึง (Access Time) เพื่อรอให้ข้อมูลเขียนสำเร็จ
 - ขนาดของข้อมูลขั้นต่ำและขั้นสูงที่สามารถเขียนได้
5. เหตุใดหน่วยความจำสารแตติกแรมจึงไม่ต้องรีเฟรชข้อมูล
6. เหตุใดหน่วยความจำ SDRAM จึงต้องรีเฟรชข้อมูลภายใต้เงื่อนไขในข้อ 5

บทที่ 6

กลไกอินพุตและเอาต์พุต (Input and Output)

ผู้ใช้สามารถอาศัยเครื่องคอมพิวเตอร์ติดต่อกับโลกภายนอกผ่านทางอุปกรณ์อินพุต/เอาต์พุต เช่น การเชื่อมต่อกับผู้ใช้ผ่านทางคีย์บอร์ด เม้าส์ หน้าจอสัมผัส การขยับ (Motion) / การเอียง (Tilt) / ความเร่ง (Acceleration) ของการเคลื่อนไหว การเชื่อมต่อกับเครือข่ายอินเทอร์เน็ต ผ่านทางเครือข่ายมีสาย และเครือข่ายไร้สาย เป็นต้น ดังนั้น การเชื่อมต่อกับอุปกรณ์เหล่านี้จำเป็นต้องทำตามมาตรฐาน เพื่อลดความยุ่งยากในการออกแบบและตั้งค่าโดยรวมของคอมพิวเตอร์

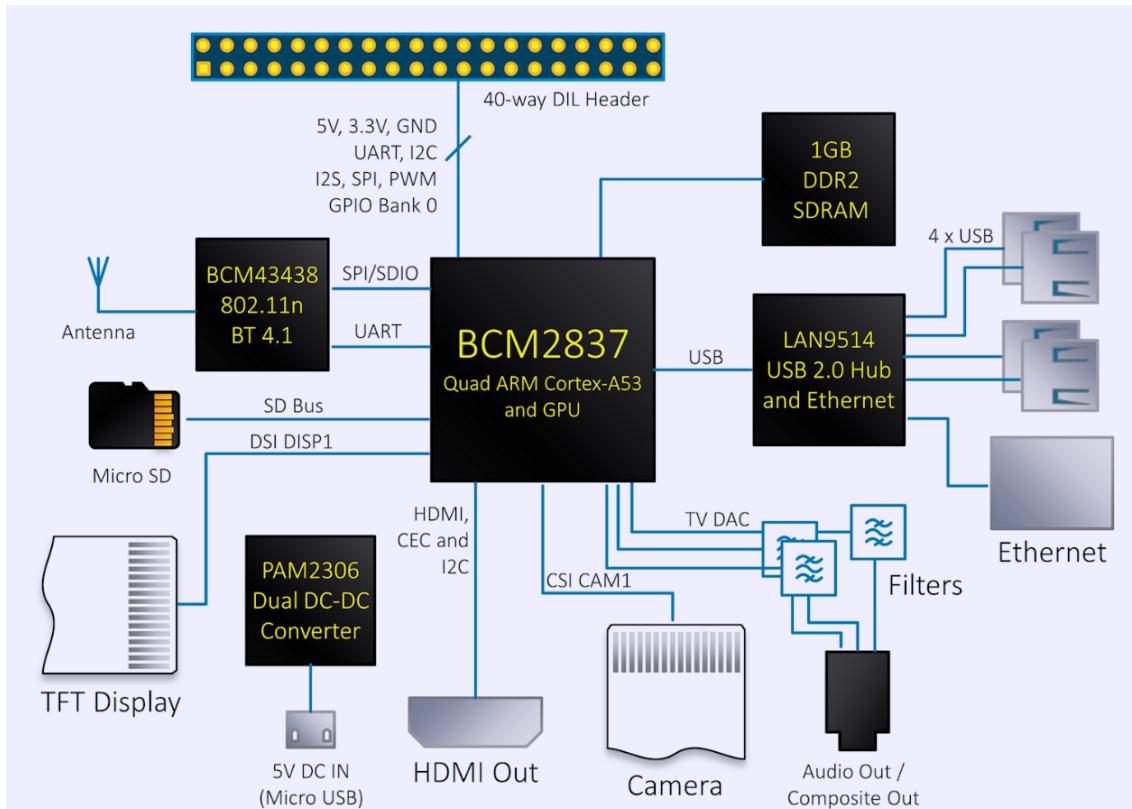
ระบบปฏิบัติการจะทำหน้าที่ควบคุมและบริหารจัดการอุปกรณ์อินพุต/เอาต์พุต เพื่อให้โปรแกรมต่างๆ สามารถใช้ทรัพยากรเหล่านี้ร่วมกัน โดยจะต้องมีการป้องกันและการจัดลำดับการใช้งาน (Protection and Scheduling) เนื่องจาก การใช้งาน อุปกรณ์อินพุต/เอาต์พุต ทำให้เกิดการอินเทอร์รูปต์ (Interrupt) แปลงว่า ขัดจังหวะการทำงานของซีพียู ระบบปฏิบัติการจะต้องจัดเตรียมวิธีการเชื่อมต่อกับอุปกรณ์อินพุต/เอาต์พุต โดยมีอาร์ดแวร์ที่จะควบคุมการทำงานของอุปกรณ์ต่างๆ แทนโดยตรง เพื่อให้ซีพียูรับเฉพาะงานที่สำคัญโดยเน้นที่การประมวลผลข้อมูลได้อย่างต่อเนื่อง ซึ่งวิธีนี้จะเพิ่มประสิทธิภาพและการตอบสนองต่อผู้ใช้โดยรวมได้ และเพื่อให้โปรแกรมเมอร์สามารถพัฒนาโปรแกรมได้อย่างรวดเร็ว ประสิทธิภาพสูง และปลอดภัย เนื้อหาในบทนี้มีวัตถุประสงค์ดังต่อไปนี้

- เพื่อให้เข้าใจสัญญาณการเชื่อมต่อกับอุปกรณ์อินพุต/เอาต์พุตชนิดต่างๆ ของบอร์ด Pi3/Pi4
- เพื่อให้รู้จักโครงสร้างและการทำงานด้านอินพุต/เอาต์พุตของบอร์ด Pi3/Pi4
- เพื่อให้เข้าใจกลไกการติดต่อกับอุปกรณ์อินพุต/เอาต์พุตชนิดต่างๆ
- เพื่อให้เข้าใจหลักการ Memory Mapped I/O, อินเทอร์รูปต์ และ Direct Memory Access โดยใช้ซีพียู ARM เป็นกรณีศึกษา

เนื้อหาในบทต่างๆ ที่ผ่านมาอธิบายโครงสร้างของบอร์ด Pi3/Pi4 ตามรูปที่ 6.1 ว่าประกอบด้วย ชิป BCM2837/BCM2711 ภายใต้ชื่อซีพียู 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
- ชิปหรืออุปกรณ์ภายนอกต่าง ๆ เช่น ชิป USB/Ethernet ชิป WiFi และ Bluetooth



รูปที่ 6.1: การเชื่อมต่ออุปกรณ์ต่าง ๆ บนบอร์ด Pi3 โดยมีชิป BCM2837 เป็นศูนย์กลาง ที่มา: xdevs.com

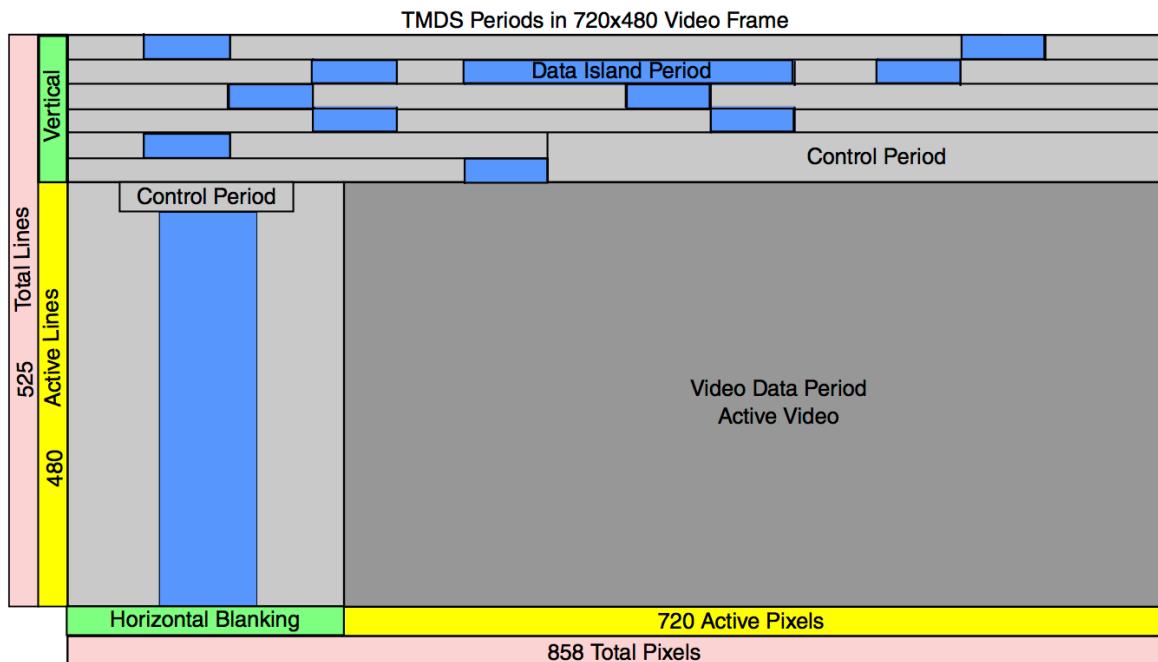
เนื้อหาในบทนี้ จะเน้นเรื่องของการเชื่อมต่อชิป BCM2837 กับวงจรและอุปกรณ์ภายนอก ตัวชิป BCM2837 นี้ประกอบด้วยขาทั้งหมด 200 ขา ซึ่งส่วนใหญ่จะเป็นขาเชื่อมต่อ อุปกรณ์หรือวงจร อินพุตและเอาต์พุต เช่น HDMI USB เครื่อข่ายไร้สาย เครื่อข่ายสาย เป็นต้น

ด้านการแสดงผล บอร์ด Pi3/Pi4 รองรับการเชื่อมต่อกับจอแสดงผลได้ 3 ชนิด คือ จอโทรทัศน์ด้วยสัญญาณคอมโพสิตวีดีโอ จอภาพ LCD ขนาดใหญ่ผ่านสายสัญญาณ HDMI และ จอภาพ LCD ขนาดเล็กผ่านสายสัญญาณ DSI บอร์ดมีพอร์ต USB 2.0 จำนวน 4 พอร์ต เช่น คีย์บอร์ด เม้าส์ เป็นต้น บอร์ดสามารถเชื่อมต่ออินเทอร์เน็ตด้วยสายสัญญาณ Ethernet หรือสายแลน (LAN: Local Area Network) และเชื่อมต่อกับระบบเครือข่ายแบบไร้สาย ด้วยชิป BCM43438 ซึ่งภายในมีโมดูล 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 สำหรับเชื่อมต่อจอมอนิเตอร์ HDMI คือ สัญญาณสำหรับการเชื่อมต่ออุปกรณ์ภาพและเสียง เพื่อแทนที่การเชื่อมต่อรูปแบบเดิม ๆ เช่น สัญญาณคอมโพสิตวีดีโอ และแบบ S-Video



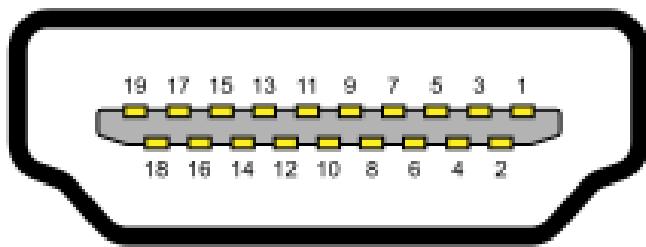
รูปที่ 6.2: ภาพความละเอียด 720 จุด x480 เส้นที่ผู้ใช้มองเห็น แบ่งเป็นการส่งแพ็กเก็ตข้อมูลจุดภาพและแพ็กเก็ตควบคุมทางช่องสัญญาณ TMDS ภายใต้สัญญาณ HDMI ในช่วงเวลาต่าง ๆ ที่มา: freebsd.org

การเชื่อมต่อแบบ HDMI เป็นการถ่ายโอนสัญญาณแบบดิจิทัล สามารถส่งได้ทั้งข้อมูลภาพดิจิทัล และข้อมูลเสียงดิจิทัลไปพร้อม ๆ กันด้วยอัตราบิตร率สูงระดับกิกะบิตต่อวินาที ตัวอย่างในรูปที่ 6.2 เป็นภาพความละเอียด 720 จุด x 480 เส้นที่ผู้ใช้มองเห็น แบ่งเป็น การส่งแพ็กเก็ตข้อมูลจุดภาพ แพ็กเก็ตข้อมูลเสียง และแพ็กเก็ตควบคุมในช่วงเวลาต่าง ๆ

การเชื่อมต่อด้วยสัญญาณ HDMI เหมาะสำหรับการแสดงผลจากเครื่องคอมพิวเตอร์ หรือเครื่องเล่นมีเดีย (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 ซึ่งสัญญาณ HDMI เวอร์ชันปัจจุบันขณะที่เขียนตำรา คือ เวอร์ชัน 2.1 รองรับภาพที่ละเอียดสูงเพرمล 8000 จุดต่อเส้นที่ 60 เฟรมต่อวินาที (ย่อว่า 8K60) และเพرمล 4000 จุดต่อเส้นที่ 120 เฟรมต่อวินาที (ย่อว่า 4K120) และเพิ่มความละเอียดเพิ่มถึงเพرمล 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 ที่มาและรายละเอียดเพิ่มเติม:

wikipedia



รูปที่ 6.3: หัวเชื่อมต่อ HDMI ชนิด Female ประกอบด้วยขาสัญญาณทั้งหมด 19 ขา ที่มา: [wikipedia.org](https://en.wikipedia.org)

ตารางที่ 6.1: หมายเลขขา ชื่อ และ วัตถุประสงค์ ของ สาย สัญญาณ ชนิด HDMI เวอร์ชัน 1.4 ที่มา: [wikipedia.org](https://en.wikipedia.org)

ขา	ชื่อ	วัตถุประสงค์
1	TMDS Data2+	ข้อมูลช่วง Control Period ข้อมูลเลน 2 ชั่วบวก
2	TMDS Data2 Shield	ชีลด์สำหรับข้อมูลเลน 2
3	TMDS Data2-	ข้อมูลเลน 2 ชั่วลบ
4	TMDS Data1+	ข้อมูลเลน 1 ชั่วบวก
5	TMDS Data1 Shield	ชีลด์สำหรับข้อมูลเลน 1
6	TMDS Data1-	ข้อมูลเลน 1 ชั่วลบ
7	TMDS Data0+	ข้อมูลเลน 0 ชั่วบวก
8	TMDS Data0 Shield	ชีลด์สำหรับข้อมูลเลน 0
9	TMDS Data0-	ข้อมูลเลน 0 ชั่วลบ
10	TMDS Clock+	สัญญาณคลือกชั่วบวก
11	TMDS Clock Shield	ชีลด์สำหรับสัญญาณคลือก
12	TMDS Clock-	สัญญาณคลือกชั่วลบ
13	CEC	ควบคุมอุปกรณ์ต่อพ่วง
14	HEAC+	Ethernet and Audio Return+
15	SCL	สัญญาณคลือกสำหรับช่อง 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 สาย HDMI มีช่องสื่อสารย่อย 5 ช่องแยกอิสระ จากกันได้แก่

- ช่อง TMDS (Transition-Minimized Differential Signaling) สำหรับส่งแพ็คเก็ต (Packet) ข้อมูล

เป็นดิจิทัลในห่วงเวลาของการแสดงผลภาพเพียง 1 เฟรม ประกอบด้วย ช่วงส่งแพ็กเก็ตข้อมูลภาพ (Video Data Period) ช่วงส่งแพ็กเก็ตข้อมูลเสียง (Audio Data Period) และช่วงส่งแพ็กเก็ตควบคุม (Control Period) สำหรับสัญญาณควบคุม ยกตัวอย่าง เช่น สัญญาณ HSYNC (Horizontal Synchronization) และ VSYNC (Vertical Synchronization) เป็นต้น

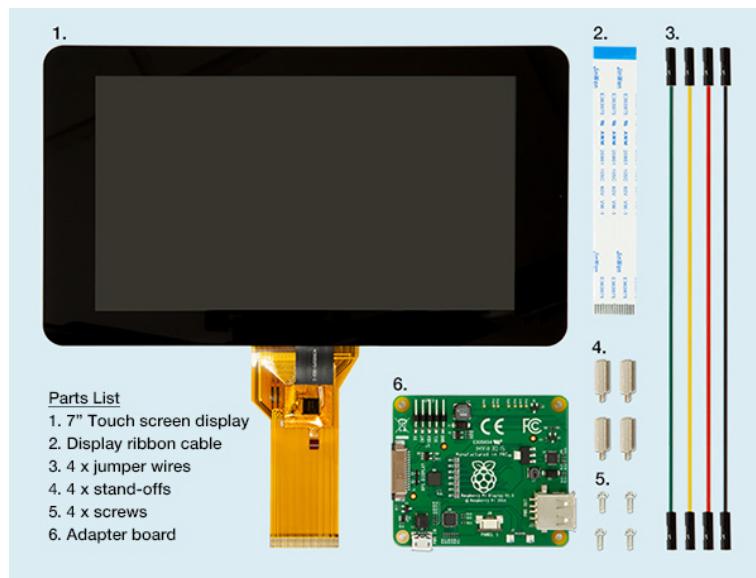
รูปที่ 6.2 แสดงตัวอย่างช่วงเวลาของการส่งแพ็กเก็ตข้อมูลภาพ จะใช้สัญลักษณ์สีเทา และแพ็กเก็ตควบคุมจะใช้สัญลักษณ์สีฟ้า ในช่วงเวลาต่าง ๆ สำหรับการแสดงภาพด้วยความละเอียด 720x480 หรือ 480 เส้น ๆ ละ 720 จุดต่อ 1 เฟรม แต่ละเฟรมใช้เวลา 33 มิลลิวินาทีเพื่อการแสดงผล หรือคิดเป็น 30 เฟรมต่อวินาที ภายในระยะเวลา 33 มิลลิวินาที มีการส่งสัญญาณภาพจริงเป็นจำนวน 525 เส้น ๆ ละ 858 พิกเซล แต่ผู้ใช้งานจะมองเห็นเพียง 480 เส้น ๆ ละ 720 พิกเซลในตำแหน่งที่มีสีเทาเข้มเท่านั้น ส่วนจำนวนเส้นและจุดที่เกินเพื่อไว้สำหรับช่วงเวลา Vertical Blanking และ Horizontal Blanking ช่วงเวลาทั้งสอง จะใช้ในการส่งสัญญาณควบคุม และข้อมูลอื่น ๆ โดยแพ็กเก็ตข้อมูลภาพเริ่มต้นจากภาพเส้น (Line) บนสุด และตำแหน่งจุดภาพซ้ายสุดไปจุดภาพขวาสุดแล้วจะขยับลงมา 1 เส้นเพื่อเริ่มจากตำแหน่งจุดภาพซ้ายสุดไปจุดภาพขวาสุด เช่นเดิม การส่งแพ็กเก็ตจะขยับลงมาเรื่อย ๆ จนถึงเส้นภาพสุดล่างสุด แล้วจึงเริ่มต้นภาพเฟรมใหม่ด้วยเส้นภาพบนสุดเช่นเดิม

ในสาย HDMI หนึ่งเส้นประกอบด้วยเลนสัญญาณ TMDS จำนวน 3 เลน สำหรับส่งแพ็กเก็ตข้อมูลพร้อมกัน รายละเอียดเพิ่มเติมเกี่ยวกับสัญญาณ TMDS ที่ [Wikipedia](#)

- ช่อง DDC (Display Data Channel) ใช้สื่อสารระหว่างจอแสดงผล กับเครื่อง เล่น ด้วยมาตรฐาน I²C (อ่านว่า ไอส์แคร์ซี) เพื่อกำหนดรูปแบบและความละเอียดของภาพวีดีโอและเสียง และยังใช้คุ้มครองเนื้อหาดิจิทัลแบบดิจิตอลสูง (High-bandwidth Digital Content Protection: HDCP) เช่นภาพยนต์ที่มีลิขสิทธิ์ เป็นต้น
- ช่อง CEC (Consumer Electronics Control) ผู้ใช้งานสามารถควบคุมอุปกรณ์ต่อพ่วงผ่านช่อง CEC นี้ได้มากถึง 15 ตัว เป้าหมายคือ อุปกรณ์สามารถทำงานร่วมกันและใช้เครือข่ายอินเทอร์เน็ตไปพร้อม ๆ กัน
- ช่อง ARC (Audio Return Channel) หรือช่องสัญญาณเสียงคืนกลับ เพื่อใช้สาย HDMI เชื่อมตอกับตัว盒อุปกรณ์เสียงและเครื่องขยายเสียงผ่านทางช่อง ARC
- ช่อง HEC (HDMI Ethernet Channel) ใช้เชื่อมต่ออุปกรณ์อื่น ๆ ผ่านทางสาย HDMI ตั้งแต่เวอร์ชัน 1.3 เป็นต้นมา

การทดลองที่ 9 ภาคผนวก | มีการปรับแต่งความละเอียดของการแสดงผลผ่านสาย HDMI

6.2 สัญญาณ DSI สำหรับจอภาพ LCD ขนาดเล็ก



รูปที่ 6.4: จอแสดงผล สำหรับเชื่อมต่อระหว่างบอร์ด Pi3/Pi4 ด้วยสัญญาณการแสดงผลแบบอนุกรม (Display Serial Interface) ที่มา: element14.com

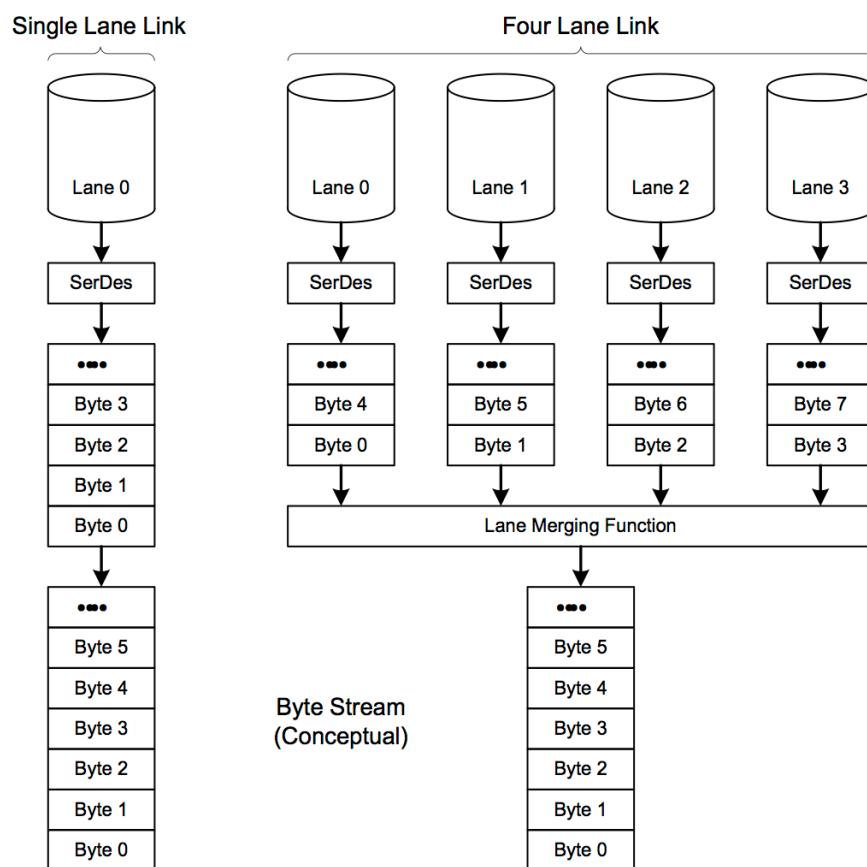
สัญญาณ DSI (Display Serial Interface) เป็นมาตรฐานสำหรับเชื่อมต่อจอภาพ LCD ขนาดเล็ก กับชิปซีพียูบนอุปกรณ์เคลื่อนที่ เช่น โทรศัพท์สมาร์ตโฟน แท็บเล็ต คอมพิวเตอร์โน๊ตบุ๊ก เป็นต้น เพื่อการแสดงผลในรูปของกราฟิก荷ะด้วยความละเอียดสูง สัญญาณ DSI นี้ถูกกำหนดให้เป็นมาตรฐานโดยองค์กรชื่อ MIPI (Mobile Industry Processor Interface)

ตารางที่ 6.2: หมายเลข และหน้าที่ของสายสัญญาณ DSI สำหรับจอภาพ LCD ขนาดเล็ก ประกอบด้วยข้อมูลภาพจำนวน 2 เลน ที่มา: 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-	เลนคล็อกขาลง
6	Clock+	เลนคล็อกขาขึ้น
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 สายที่ใช้จะต้องเป็นสายจำนวนหลายเส้นเรียงติดกันแบบราบ เรียกว่า **สายแฟ** (Ribbon) จำนวน 15 ชา สามารถส่งข้อมูลพร้อม ๆ กันจำนวน 2 เลน แต่ละเลนเป็นการส่งข้อมูลภาพแบบอนุกรม (Serial) ตารางที่ 6.2 แสดงหมายเลข ชื่อและวัตถุประสงค์ของชาทั้ง 15 ชา โดย ชา 5 และ 6 จะถูกกำหนดให้เป็นเลนสัญญาณคลือก ชา 8 และ 9 จะถูกกำหนดให้เป็นเลนข้อมูลภาพหมายเลข 0 ชา 2 และ 3 จะถูกกำหนดให้เป็นเลนข้อมูลภาพหมายเลข 1 เป็นต้น

จอภาพ LCD ขนาดเล็กจะทำงานใน **โหมดรับคำสั่ง** เพื่อให้ซีพียูกำหนดค่าต่าง ๆ ของรีจิสเตอร์ควบคุมการทำงานของจอตามที่ต้องการ โดยอาศัยการรับส่งข้อมูลภาพและคำสั่งในเลนที่ 0 ได้ทั้งสองทิศทาง (BiDirectional)



รูปที่ 6.5: เปรียบเทียบสัญญาณ DSI ชนิดหนึ่งเลน (Single Lane) และชนิด 4 เลน ที่มา: wikipedia.org

การทำงานในโหมดแสดงผล จอ LCD จะรับข้อมูลภาพจากทุก ๆ เลน โดยเลนที่ 1 เป็นต้นไปจะทำหน้าที่รับข้อมูลภาพได้เพียงทิศทางเดียวเท่านั้น ตามรูปที่ 6.5 มาตรฐานสัญญาณ DSI แบ่งการส่งข้อมูลภาพเป็นชนิดเลนเดียว (Single Lane) และหลาย ๆ เลนโดยมีจำนวนตั้งแต่ 2 เลนขึ้นไป เพื่อกระจายข้อมูลภาพแต่ละไบต์ไปที่ลําเลน ยกตัวอย่าง เช่น การส่งภาพแบบ 4 เลน ข้อมูลภาพไบต์ที่ 0, 4, 8, ... จะส่งมาทางเลนหมายเลข 0 ข้อมูลภาพไบต์ที่ 1, 5, 9, ... จะส่งมาทางเลนหมายเลข 1 ข้อมูลภาพไบต์ที่ 2, 6, 10, ... จะส่งมาทางเลนหมายเลข 2 ข้อมูลภาพไบต์ที่ 3, 7, 11, ... จะส่งมาทางเลนหมายเลข 3 และสลับกันไปแบบนี้เรื่อย ๆ เมื่อปลายทาง (จอภาพ) รับข้อมูลได้สำเร็จ วงจรรับจะนำข้อมูลภาพเหล่านั้นมารวม

กัน (Lane Merging Function) เป็นภาพเฟรมเดียวกัน การส่งข้อมูลภาพจำนวนหลาย ๆ เลนพร้อมกันช่วยให้แสดงภาพแต่ละเฟรมเร็วขึ้น รองรับการแสดงผลที่ละเอียดมากขึ้น รองรับอัตราการเปลี่ยนแปลงภาพต่อวินาทีได้มากขึ้น การเคลื่อนไหวของภาพจึงต่อเนื่องไม่กระตุก เพิ่มอรรถรสในการรับชมมากขึ้น

6.3 สัญญาณ CSI สำหรับเชื่อมต่อกล้องขนาดเล็ก



รูปที่ 6.6: การเชื่อมต่อระหว่างบอร์ด Pi3/Pi4 และกล้องขนาดเล็กด้วยสัญญาณกล้องแบบอนุกรม (Camera Serial Interface) ที่มา: element14.com

บอร์ด Pi3/Pi4 สามารถเชื่อมต่อกล้องขนาดเล็กตามมาตรฐาน CSI (Camera Serial Interface) ซึ่งกำหนดโดยองค์กร MIPI.org จึงมีความคล้ายคลึงกับสัญญาณ DSI ข้อมูลภาพจากกล้องจะส่งผ่านสาย CSI ผ่านเลนข้อมูล ตามรูปที่ 6.5 เพื่อไปรวมกันเป็นภาพเดียวที่ปลายทาง (หน่วยความจำบัฟเฟอร์)

กล้อง จะเชื่อมกับช่องเก็ตหมายเลข S5 บนบอร์ด Pi3/Pi4 ซึ่ง เป็นหัวช่องเก็ตชนิด ZIF (Zero Insertion Force) ชนิด 15 ขา ติดยึดบนพื้นผิวของแผ่นวงจรพิมพ์ (Surface Mount) มาตรฐาน CSI กำหนดให้สัญญาณที่ใช้สำหรับเลนข้อมูลเป็น ชนิด Low Voltage Differential Signalling (SubLVDS) เช่นกัน โดยปรับปรุงมาตรฐานสัญญาณ IEEE1596.3 LVDS สำหรับอุปกรณ์ที่ใช้ไฟกระแสตรง ต่าประมาณ 1.2 โวลต์ เพื่อให้สามารถส่งข้อมูลต่อเลนได้สูงสุด 800 - 1,000 เมกะบิตต่อวินาที (Mbps)

ตารางที่ 6.3 แสดงหมายเลขขา ชื่อ และวัตถุประสงค์ของสัญญาณ CSI ดังนี้

- ขา CAM1_D0-/CAM1_D0+ และ ขา CAM1_D1-/CAM1_D1+ คือ ขั้วลบและบวกของเลนข้อมูล ที่ 0 และ 1 ตามลำดับ โดยชิปในกล้องจะสร้างสัญญาณแล้วส่งผ่านสายแพให้กับวงจรที่รับภาพในชิป BCM2837 ผ่านเลนข้อมูลทั้งสองเลนนี้คล้ายกับสัญญาณ DSI แบบหลายเลนในรูปที่ 6.5

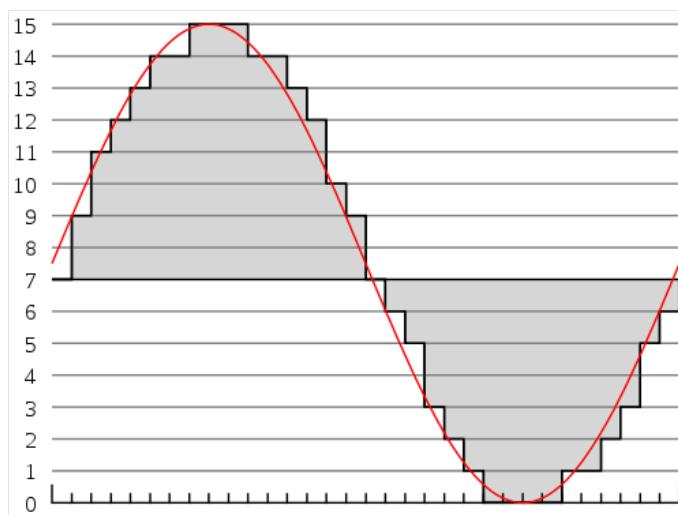
ผู้ใช้สามารถกำหนดรูปแบบของข้อมูลภาพที่ส่งแบบต่าง ๆ ดังนี้

- RGB (Red Green และ Blue)
- RAW หรือข้อมูลภาพที่ไม่มีการปรับแต่งใด ๆ
- YUV (Y: Luminance หรือ ความสว่าง U และ V คือ องค์ประกอบของสี Chrominance 2 ชนิด)
- ขา CAM1_C- และขา CAM1_C+ คือ ขาลบและขาบวกของสัญญาณคลือกตามลำดับ เพื่อซิงโครในซึ่การส่งข้อมูลภาพในเลนต่าง ๆ ที่ความเร็วสูง
- ขา SDA0 และขา SCL0 คือ สัญญาณข้อมูล และสัญญาณคลือกตามมาตรฐาน I²C เพื่อควบคุมการทำงานของกล้อง เช่น ความละเอียดของภาพ รูปแบบของข้อมูล เป็นต้น โดยการรับส่งข้อมูล SDA0 จะซิงโครในซึ กับสัญญาณคลือก SCL0 ซึ่งมีความถี่ต่ำกว่าสัญญาณคลือก 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 ขาลบ
3	CAM1_D0+	ข้อมูลภาพเลน 0 ขาบวก
4	Ground	กราวน์ด (0 โวลต์)
5	CAM1_D1-	ข้อมูลภาพเลน 1 ขาลบ
6	CAM1_D1+	ข้อมูลภาพเลน 1 ขาบวก
7	Ground	กราวน์ด (0 โวลต์)
8	CAM1_C-	สัญญาณคลือกขาลบ
9	CAM1_C+	สัญญาณคลือกขาบวก
10	Ground	กราวน์ด (0 โวลต์)
11	CAM_GPIO	ขา GPIO
12	CAM_CLK	ขาสัญญาณคลือก
13	SCL0	สัญญาณคลือกสำหรับ I ² C
14	SDA0	สัญญาณข้อมูลสำหรับ I ² C
15	+3.3 V	ไฟกระแสตรง +3.3 โวลต์

6.4 สัญญาณ PCM สำหรับข้อมูลเสียงดิจิทัล



รูปที่ 6.7: สัญญาณดิจิทัล PCM (Pulse Code Modulation) ความละเอียด 16 ระดับสูง (Sampling) จากสัญญาณแอนะล็อกรูปคลื่นไอน์ (Sine Wave) ด้วยความถี่สูง 26 เท่าของความถี่สูงสุด ($f_s = 26 f_{max}$) ที่มา: wolfcrow.com

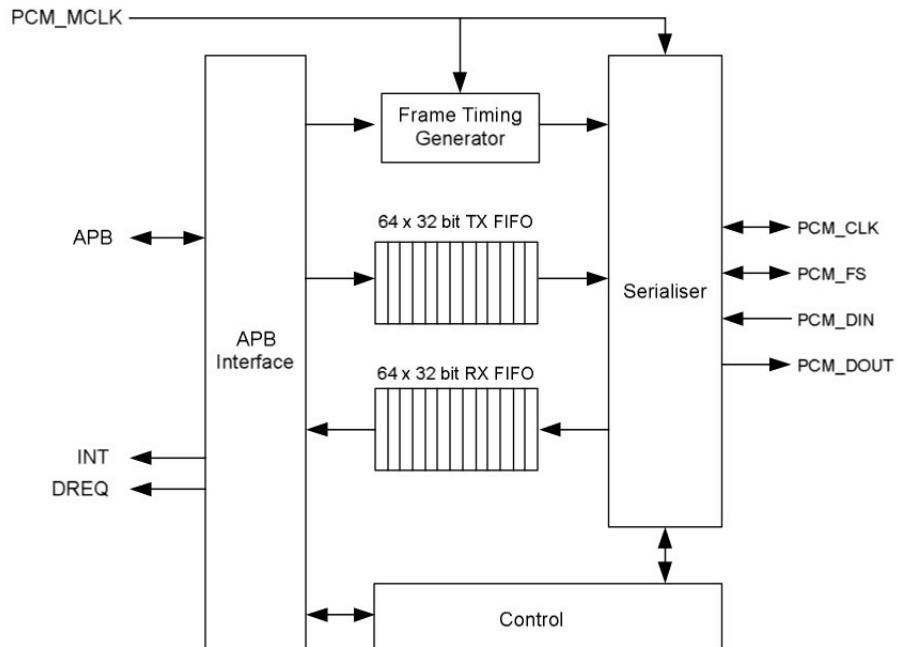
สัญญาณชนิด PCM คือ สัญญาณดิจิทัลพื้นฐาน ได้จากการแปลงสัญญาณแอนะล็อกเป็นดิจิทัล (Analog to Digital: A2D) สัญญาณ PCM ได้รับความนิยมแพร่หลายในอดีตจนถึงปัจจุบัน เช่น ข้อมูลเสียงในแผ่นชีดี้เพลย์ (Audio Compact Disc) โทรศัพท์บ้านพื้นฐาน และอีน ๆ ชิป BCM2837/BCM2711 บนบอร์ด Pi3/Pi4 สามารถรับข้อมูลเสียงดิจิทัล ในรูปแบบ PCM นี้ ไปประมวลผล แล้วแปลงสัญญาณดิจิทัลให้เป็นสัญญาณแอนะล็อกเพื่อส่งต่อให้กับลำโพงภายนอก ในหัวข้อที่ 6.5

สัญญาณแอนะล็อกรูปคลื่นไอน์สูงสูม (Sampling) ด้วยความถี่สูม (Sampling Frequency, f_s) ที่ความถี่สูงกว่า 2 เท่าความถี่สูงสุดของสัญญาณแอนะล็อก ($f_s > 2f_{max}$) โดย f_{max} คือ ความถี่สูงสุดของสัญญาณแอนะล็อก ตามกฎของ Nyquist ยกตัวอย่าง เช่น

- ตัวอย่างที่ 1 สัญญาณรูปคลื่นไอน์ (Sine Wave) ในรูปที่ 6.7 สูมด้วยความถี่สูงเป็น 26 เท่าของความถี่สูงสุด ($f_s = 26 f_{max}$) และทำการแปลงเป็นสัญญาณดิจิทัลชนิด PCM (Pulse Code Modulation) ด้วยความละเอียด $16 = 2^4$ ระดับ กล้ายเป็นข้อมูลเลขจำนวนเต็มชนิดไม่มีเครื่องหมายขนาด 4 บิตต่อการสูม 1 ครั้ง
- ตัวอย่างที่ 2 สัญญาณเสียงสนทนาผ่านโทรศัพท์พื้นฐาน จะสูมด้วยความถี่ 8,000 ครั้ง ต่อวินาที ($f_s = 8$ กิโลเฮิรตซ์) ซึ่งจะตรงกับคาบเวลา $1/8000 = 125$ มิโครวินาที ด้วยความละเอียด $256 = 2^8$ ระดับ กล้ายเป็นเลขจำนวนเต็มชนิดไม่มีเครื่องหมายยาว 8 บิตต่อการสูม 1 ครั้ง
- ตัวอย่างที่ 3 สัญญาณเสียงเพลง คุณภาพระดับ แผ่นชีดี้ จะสูมด้วยความถี่ 44,100 ครั้ง ต่อวินาที ($f_s = 44.1$ กิโลเฮิรตซ์) ซึ่งจะตรงกับคาบเวลา $1/44100 = 22.67$ มิโครวินาที ด้วยระดับความละเอียด $65,536 = 2^{16}$ ระดับ เพื่อให้เป็นข้อมูลเสียงดิจิทัล PCM หรือ เลขจำนวนเต็มชนิดไม่มี

เครื่องหมายยา 16 บิตต่อการสุ่ม 1 ครั้ง ทั้งนี้ ช่องสัญญาณเสียงด้านซ้ายและด้านขวาจะการสุ่มพร้อมกัน รวมเป็นข้อมูล 32 บิตต่อการสุ่ม 1 ครั้ง

ในรูปที่ 6.8 มอดูล PCM ภายในชิป BCM2837/BCM2711 ประกอบด้วย



รูปที่ 6.8: มอดูล PCM ประกอบด้วย วงจรเขื่อมต่อ กับชิปปิค์ผ่านบัส APB, หน่วยความจำบัฟเฟอร์สำหรับส่งและรับข้อมูลเสียงดิจิทัล และวงจรเขื่อมต่อชนิด I²S ที่มา: [Broadcom Corp. \(2012\)](#)

- วงจรเขื่อมต่อ กับชิปปิค์ ARM Cortex A53/A72 ผ่านบัส APB (ARM Peripheral Bus)
- หน่วยความจำบัฟเฟอร์สำหรับส่ง (Transmit) และรับ (Receive) ขนาด 64x32 บิต เป็นบัฟเฟอร์ด้านรับ 1 ชุด และด้านส่งอีก 1 ชุด รวม 2 ชุด ทำหน้าที่พักเก็บข้อมูลชั่วคราวระหว่างหน่วยความจำภายในภาพ เพื่อรอให้ชิปปิค์ประมวลผลและส่งสัญญาณเสียงไปยังลำโพง
- มอดูลเชื่อมต่อกับอุปกรณ์ภายนอกตามมาตรฐาน I²S (Inter IC Sound) (อ่านว่า ไอส์แคร์อส) เพื่อเชื่อมต่อกับภายนอกชิป BCM2837/BCM2711 ประกอบด้วยสายสัญญาณจำนวน 4 เส้นแบบอนุกรม ได้แก่
 - สัญญาณ PCM_CLK - สัญญาณคลื่อกสำหรับส่งข้อมูลแต่ละบิตด้วยความถี่สูง
 - สัญญาณ PCM_DIN - สัญญาณข้อมูลเสียงขาเข้าหรือขารับ (Receive) จะรับสัญญาณตามความถี่ของ PCM_CLK โดยจะรับบิตสูง (Most Significant Bit) ก่อนและบิตสุดท้ายคือ บิตต่ำสุด (Least Significant Bit) เสมอ
 - สัญญาณ PCM_DOUT - สัญญาณข้อมูลเสียงขาออกหรือขาส่ง (Transmit) จะมีพิธีทางที่ตรงกันข้ามกับ PCM_DIN

- **สัญญาณ PCM_FS** - สัญญาณซิงค์เฟรมข้อมูล (Frame Sync) เพื่อใช้ประกาศการเริ่มต้นและสิ้นสุดเฟรมข้อมูล โดยหนึ่งเฟรมสามารถกำหนดความยาวได้ ทั้งนี้ขึ้นอยู่กับขนาดจำนวนบิตข้อมูล และความถี่สุ่ม (Sampling Frequency)

รายละเอียดเพิ่มเติมที่ [wikipedia](#)

มอดูลนี้รองรับการทำงานทั้งสามรูปแบบ คือ การโพลลิ่ง (Polling) การอินเทอร์รัปต์ (Interrupt) รายละเอียดเพิ่มเติมในหัวข้อที่ [6.12](#) และ การเข้าถึงหน่วยความจำโดยตรง (Direct Memory Access) รายละเอียดเพิ่มเติมในหัวข้อที่ [6.13](#)

6.5 สัญญาณภาพและเสียงสำหรับจอทีวีและแล็ปท็อป

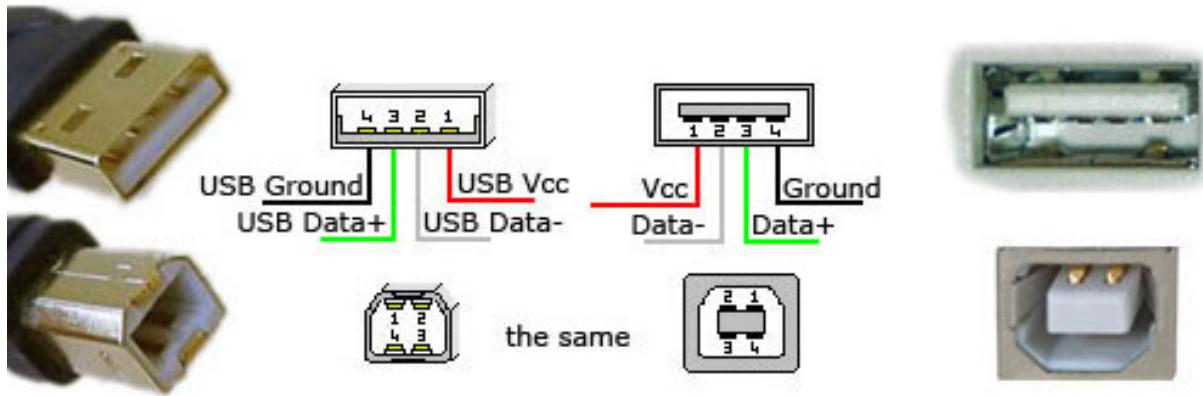


รูปที่ 6.9: แจ็คขนาด 3.5 มม. (กลาง) ชนิด 4 ขั้วสำหรับเสียบกับบอร์ด Pi3/Pi4 (ขวา) ส่งสัญญาณภาพไปยังแจ็ค RCA (เหลือง) และสัญญาณเสียงไปยังแจ็ค RCA (แดงและขาว) ที่มา: [stackexchange.com](#)

ช่องเสียบแจ็คในรูปที่ [6.9](#) ด้านขวา ใช้เชื่อมต่อสัญญาณภาพ (แอนะล็อก) และเสียง (แอนะล็อก) จากบอร์ด Pi3/Pi4 เข้ากับจอทีวีหรือจอมอนิเตอร์ที่มีช่องอินพุตเป็น คอมโพสิตวีดิโอ (Composite Video) และเสียงสเตอริโอ (Stereo) หรือ โทรทัศน์บางเครื่องเรียกว่าช่อง AV (Audio/Video) ซึ่งสามารถใช้ทดแทนจอภาพ LCD ได้ แต่จะได้สัญญาณภาพความละเอียดต่ำกว่าสัญญาณ HDMI โดยแจ็ค 3.5 มม. (กลาง) นี้ เป็นชนิด 4 ขั้วสำหรับเสียบกับบอร์ด Pi3/Pi4 เชื่อมต่อสัญญาณภาพไปยังแจ็ค RCA (เหลือง) และสัญญาณเสียงไปยังแจ็ค RCA (แดงและขาว) ผู้อ่านสามารถหาซื้อสายสัญญาณสำเร็จรูปนี้ได้ทั่วไป หรือจะบัดกรีเชื่อมต่อสายเองได้เช่นกัน

สัญญาณภาพดิจิทัล ชื่อ TV DAC ภายในชิป BCM2837 ถูกแปลงเป็นสัญญาณภาพแอนะล็อกโดยใช้ DAC (Digital to Analog Converter) ชนิดตัวกรองความถี่ต่ำผ่าน หรือ Low-Pass Filter ในรูปที่ [6.1](#) ในทำนองเดียวกัน สัญญาณเสียงดิจิทัลชนิด PCM ในหัวข้อที่ [6.4](#) แบ่งเป็นสัญญาณเสียงช่องซ้ายและช่องขวาแบบสเตอริโอ จะถูกแปลงเป็นสัญญาณเสียงแอนะล็อกโดยใช้ DAC (ชนิดตัวกรองความถี่ต่ำผ่าน หรือ Low-Pass Filter) ภายในชิป BCM2837 เช่นกัน

6.6 สัญญาณ USB สำหรับอุปกรณ์ต่อพ่วงต่าง ๆ



รูปที่ 6.10: หัวเชื่อมต่อ USB ชนิด A (บน ฝั่งโฮสท์) และ B (ล่าง ฝั่งอุปกรณ์) ประกอบด้วยสัญญาณ 4 เส้น กราวน์ด (0 โวลต์) (GND) Data+ Data- และไฟกระแสตรง 5 โวลต์ที่มา: quora.com

เนื้อหาในหัวข้อนี้ต่อเนื่องจากเรื่อง คีย์บอร์ด ในหัวข้อที่ 3.1.3 และเม้าส์ ในหัวข้อที่ 3.1.3 ซึ่งนิยมใช้เป็นแบบ USB เพื่อต่อเข้ากับบอร์ด Pi3/Pi4 และคอมพิวเตอร์ทั่วไป เวอร์ชันปัจจุบันของ USB คือ 3.1 ซึ่งมีความสามารถสูงขึ้นและได้รับความนิยมเพิ่มขึ้น ในตำราเล่มนี้จะกล่าวถึง USB เวอร์ชัน 2.0 ซึ่งเป็นพื้นฐานและมีคุณสมบัติ ดังนี้

- สามารถโอนถ่ายข้อมูลทั่วไป 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 แอม培ร์สำหรับพอร์ต USB พิเศษ
- สายเคเบิลมีความยาวไม่เกิน 5 เมตร เนื่องจากความต้านทานของสายจะทำให้เกิดโวลต์เจตตากร่อง (Voltage Drop) ในสาย จนทำให้ความต่างศักย์ไปจ่ายอุปกรณ์ไม่เพียงพอ
- "Hot Swapping" รองรับการต่อเข้า/拔出 ตลอดเวลา และรีเซ็ตอุปกรณ์ที่ต่ออยู่โดยไม่ต้องรีเซ็ตหรือรีบูตระบบปฏิบัติการ

รูปที่ 6.10 แสดง หัวเชื่อมต่อ USB ชนิด A (ฝั่งโฮสต์หรือคอมพิวเตอร์) และชนิด B (ฝั่งอุปกรณ์) ในการเชื่อมต่อทางไฟฟ้าของ USB นั้นจะสายเคเบิลแบบ 4 แกน เพียง 1 เส้นต่อ 1 อุปกรณ์เท่านั้น ซึ่งมีตำแหน่งข้างดังนี้

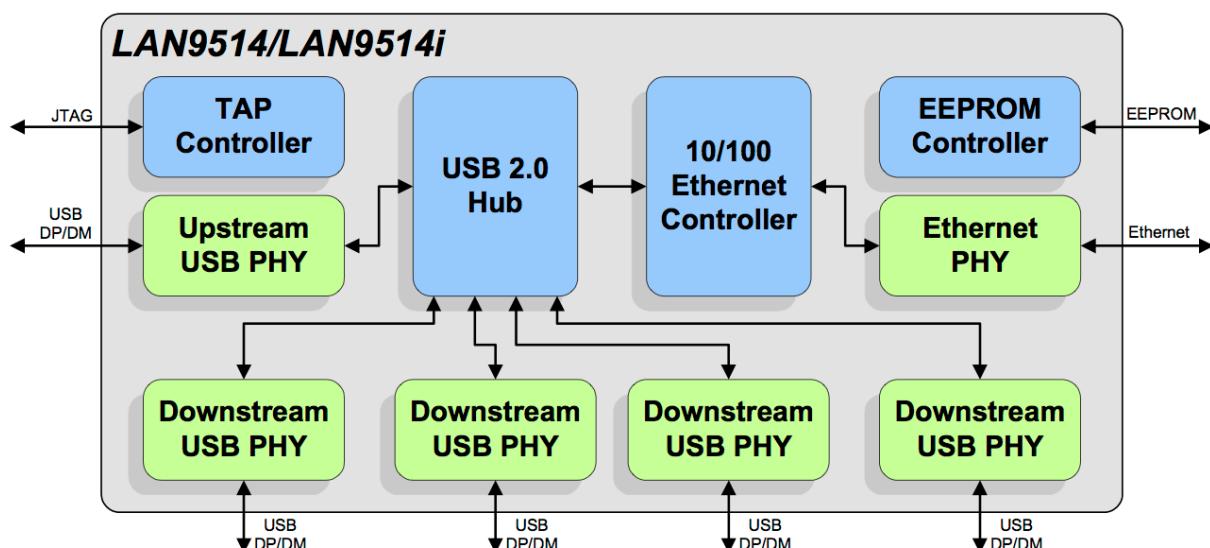
- ขา 1 เป็น +5 โวลต์ สำหรับจ่ายไฟกระแสตรงให้กับอุปกรณ์ขนาดเล็ก เช่น แฟลชไดรฟ์ กล้องเว็บ แคม โทรศัพท์สมาร์ตโฟน เป็นต้น

- 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 เป็นสายสัญญาณรับส่งข้อมูลแบบดิฟเฟอเรนเชียล (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 จะมีรูทชิป (Root Hub) เพียง 1 พอร์ต โครงสร้างของชิป LAN9514 ตามรูปที่ 6.11 ถูกออกแบบให้ LAN9514 มี USB Hub (Upstream) จำนวน 1 พอร์ต เพื่อ เชื่อม กับรูทชิป ในชิป BCM2837 และขยายจำนวนพอร์ต (Downstream) เพิ่มเป็น 4 พอร์ต ทำให้บอร์ดสามารถต่อเข้ากับคีย์บอร์ด เม้าส์ และอุปกรณ์ USB อื่น ๆ รายละเอียดเพิ่มเติมอยู่ใน การทดลองที่ 9 ภาคผนวก I 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, Inc. (2009)

6.7 สัญญาณสาย Ethernet เชื่อมต่อกับเครือข่ายอินเทอร์เน็ต



รูปที่ 6.12: หัวเชื่อมต่อชนิด RJ45 (ษ้ายสุด) สำหรับการเชื่อมต่อเครือข่ายท้องถิ่น (Local Area Network) แบบมีสาย Ethernet ที่มา: cytron.io

เครือข่ายอีเธอร์เน็ต (Ethernet) คือ เทคโนโลยีเครือข่าย LAN (Local Area Network) ที่นิยมใช้กันอย่างกว้างขวางในอดีตมาจนถึงปัจจุบัน เพราะเป็นการรับส่งข้อมูลแบบอนุกรมด้วยความเร็วสูง ใช้สายทองแดงในการติดตั้ง และต้นทุนไม่สูง เนื่องจากมีอุปกรณ์สนับสนุนเพื่อใช้งานมากที่สุด รูปที่ 6.12 แสดงตำแหน่งของพอร์ตเชื่อมต่อสาย Ethernet บริเวณมุมของบอร์ด Pi3/Pi4 หัวเชื่อมต่อชนิด RJ45 แบบตัวเมีย (Female) โดยใช้สาย CAT5e จิ้นไป

บอร์ด Pi3/Pi4 นี้ รองรับการเชื่อมต่อ Ethernet ตามมาตรฐาน IEEE 802.3 ชนิด 10/100 BaseT ซึ่งเป็นที่นิยมทั่วโลกและปัจจุบัน ด้วยอัตราบิตเรต (Bit Rate) 10/100 เมกะบิตต่อวินาที (Mbps) สายที่ใช้มีชื่อว่าสาย CAT5e หรืออาจจะใช้สายที่มาตรฐานสูงกว่าได้ เช่น CAT6 CAT6A เป็นต้น โดยจะต่อเชื่อมบอร์ดเข้ากับอุปกรณ์เครือข่าย ที่เรียกว่า อีเธอร์เน็ตสวิตช์ (Ethernet Switch) ตามลำดับขั้นและเชื่อมต่อกับเครือข่ายอินเทอร์เน็ต แสดงรายละเอียดของ Ethernet การตั้งค่า (Configuration) และอื่น ๆ เพื่อให้บอร์ดทำงานที่เป็นเซิร์ฟเวอร์ (Server) เช่น เว็บเซิร์ฟเวอร์ (Web Server) FTP เซิร์ฟเวอร์ เป็นต้น เพื่อรองรับการทำงานร่วมกับอุปกรณ์ IoT จากการเชื่อมต่อกับเซนเซอร์ต่าง ๆ รายละเอียดเพิ่มเติมอยู่ในการทดลองที่ 9 ภาคผนวก | หัวข้อที่ 1.4



รูปที่ 6.13: การเข้าปลายสาย RJ45 ทั้งสองด้านสำหรับการเชื่อมต่อระหว่างเครื่องคอมพิวเตอร์ และอุปกรณ์สวิตช์ (Switch) ตามมาตรฐาน TIA T568B ที่มา: blogspot.com

การเชื่อมต่อเครือข่ายท้องถิ่น (Local Area Network) แบบอีเออร์เน็ตในรูปที่ 6.13 สามารถเชื่อมต่อเครื่องคอมพิวเตอร์จำนวนหลายเครื่องบนเครือข่ายเดียวกัน โดยอาศัยหลักการ CSMA/CD (Carrier Sense Multiple Access/Collision Detection) รายละเอียดเพิ่มเติม ผู้อ่านสามารถศึกษาเพิ่มเติมได้ที่ลิงก์ต่อไปนี้ [wikipedia](#) หรือในรายวิชาอื่น ๆ เช่น การสื่อสารข้อมูล (Data Communication) เครือข่ายคอมพิวเตอร์ (Computer Network) เป็นต้น

6.8 สัญญาณ WiFi และ Bluetooth สำหรับการสื่อสารไร้สาย

นอกเหนือจากการเชื่อมต่อแบบใช้สายแล้ว บอร์ด Pi3/Pi4 ได้ถูกออกแบบให้ทันสมัย และรองรับการเชื่อมต่อแบบไร้สายถึงสองชนิด คือ

- เครือข่ายไร้สาย WiFi สำหรับเชื่อมต่อกับเครือข่ายอินเทอร์เน็ตแบบไร้สาย และ
- Bluetooth สำหรับเชื่อมต่อกับอุปกรณ์ระยะสั้นแบบไร้สาย เช่น โทรศัพท์เคลื่อนที่ คลื่อก เป็นต้น

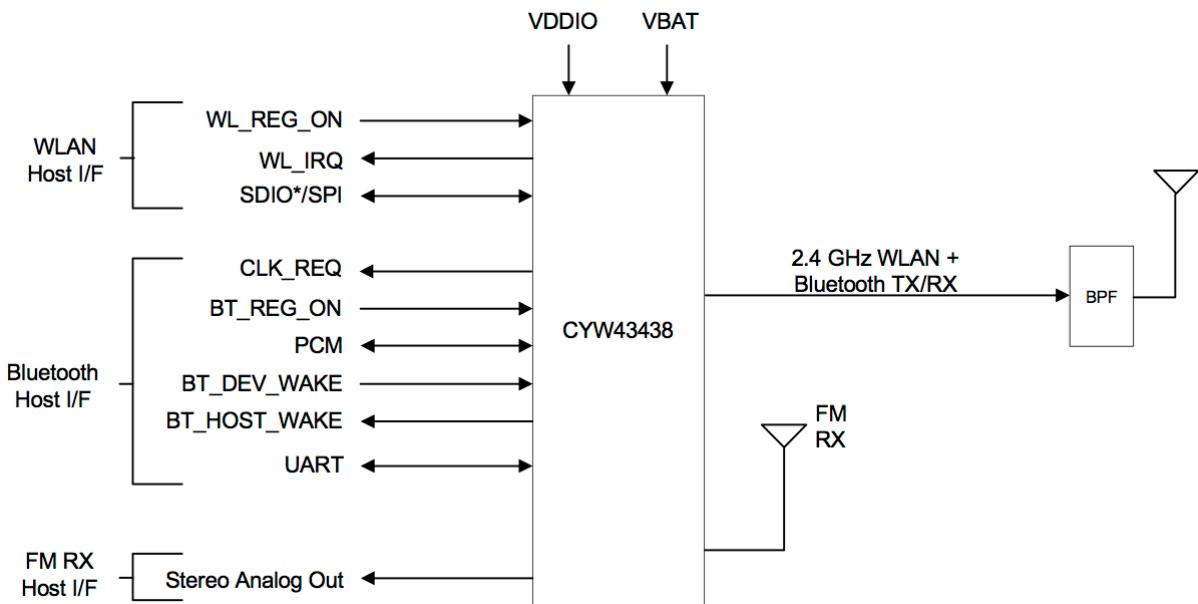
วงจรสำหรับการเชื่อมต่อแบบไร้สายทั้งสองรวมอยู่ในชิป BCM43438 บนบอร์ด RPi3/Pi4 ในรูปที่ 6.14



รูปที่ 6.14: รูปถ่ายด้านล่างของบอร์ด Pi3 และรูปขยายของชิป BCM43438 สำหรับเชื่อมต่อเครือข่ายไร้สาย WiFi และเครือข่ายไร้สายบลูทูธ (Bluetooth) ที่มา: stackexchange.com

WiFi เป็นมาตรฐานเทคโนโลยี การ เชื่อม ต่อ อินเทอร์เน็ต แบบ ไร้ สาย สำหรับ อุปกรณ์ ต่าง ๆ เช่น คอมพิวเตอร์ส่วนบุคคล เครื่องเล่นเกมส์ โทรศัพท์สมาร์ตโฟน แท็บเล็ต กล้องดิจิทัล และ เครื่องเสียงดิจิทัล โดยใช้คลื่นวิทยุที่ช่วงความถี่ 2.4 และ 5 กิกะเฮิรตซ์ อุปกรณ์ทั้งหมดสามารถเชื่อมต่อกับอินเทอร์เน็ตได้ผ่าน อุปกรณ์ที่เรียกว่า WiFi เรตเตอร์ (Router) แอคเซสพอยน์ (Access Point) หรือ ฮอตสปอต (Hot Spot) และ ระยะทำการของสัญญาณประมาณ 10 เมตรเมื่ออยู่ภายนอกอาคาร และ ระยะทำการจะเพิ่มขึ้นถ้าเป็น ที่โล่งแจ้ง และ ไม่มีสิ่งกีดขวาง ทั้งนี้ขึ้นอยู่กับมาตรฐานและกำลังส่งของอุปกรณ์ที่เชื่อมต่อกัน ซึ่งมาตรฐานของสัญญาณ WiFi อย่างเป็นทางการ คือ IEEE 802.11 ผู้อ่านสามารถค้นคว้ารายละเอียดเพิ่มเติมได้ที่ wikipedia.org

ชิป BCM43438 บนบอร์ด RPi3/Pi4 รองรับสัญญาณ IEEE 802.11b/g/n ที่ย่านความถี่คลื่นพาร์ 2.4 กิกะเฮิรตซ์ เท่านั้น และสัญญาณ Bluetooth เวอร์ชัน 4.1 รวมถึงตัวรับสัญญาณวิทยุ FM ที่มา: [Cypress Semiconductor, Corp. \(2017\)](#) บล็อกไดอะแกรมของชิป BCM43438 ประกอบด้วยขาสัญญาณเชื่อมต่อเสาอากาศ และขาเชื่อมต่อกับไมโครคอนโทรลเลอร์ (Host Interface) ทั้งสองสัญญาณใช้สายอากาศ (Antenna) และความถี่พาร์ (Carrier Frequency) ในย่านความถี่ 2.4 กิกะเฮิรตซ์เดียวกัน บลูทูธใช้หลักการ Frequency Hopping และกำลังส่งที่ต่ำกว่าสัญญาณ WiFi ทำให้ไม่เกิดการรบกวนกัน



รูปที่ 6.15: บล็อกไดอะแกรมภายในชิป BCM43438 ประกอบด้วยขาสัญญาณเชื่อมต่อเสาอากาศ และขาเชื่อมต่อกับไมโครคอนโทรลเลอร์ ที่มา: [Cypress Semiconductor, Corp. \(2017\)](#)

บลูทูธ เชื่อมต่อกับ อุปกรณ์รอบ ๆ ตัว เรียกว่า Personal Area Network (PAN). การเชื่อมต่อของ Bluetooth จะเป็นแบบ Master-Slave โดยมาสแตอร์ (Master) 1 ตัวจะสามารถรองรับสเลฟ (Slave) ได้มากถึง 7 ตัว เเรียกว่า พิโโคเน็ต (Piconet) เช่น โทรศัพท์สมาร์ตโฟน ทำหน้าที่เป็นมาสแตอร์ เชื่อมต่อกับหูฟังซึ่งเป็นสเลฟ หากมีสเลฟหลายตัวต่อเขื่อมพร้อมกันกับมาสแตอร์หนึ่งตัว มาสแตอร์จะสื่อสารกับสเลฟเหล่านั้นแบบรัวรัดโรบิน (Round Robin) นอกจากนี้ มาสแตอร์สามารถกระจาย (Broadcast) ข้อมูลไปยังสเลฟทุกตัวได้เช่นกัน

บลูทูธ จะใช้ความถี่คลื่นพาร์ ในย่าน 2.4 กิกะเฮิรตซ์ จะใช้ช่วง 2.400 ถึง 2.4835 กิกะเฮิรตซ์ และแบ่งออกเป็น 79 ช่องสัญญาณ และจะใช้ช่องสัญญาณที่แบ่งนี้ เพื่อส่งข้อมูลสลับช่องไปมา (Frequency Hopping) 1,600 ครั้งต่อ 1 วินาที (1600 Hops/Sec) ระยะทำการของ Bluetooth ประมาณ 5-100 เมตร เป็นการป้องกันการตักจับสัญญาณระหว่างสื่อสาร โดยระบบจะสลับช่องสัญญาณไปมา ผู้อ่านสามารถค้นคว้ารายละเอียดเพิ่มเติมที่ bluetooth.com

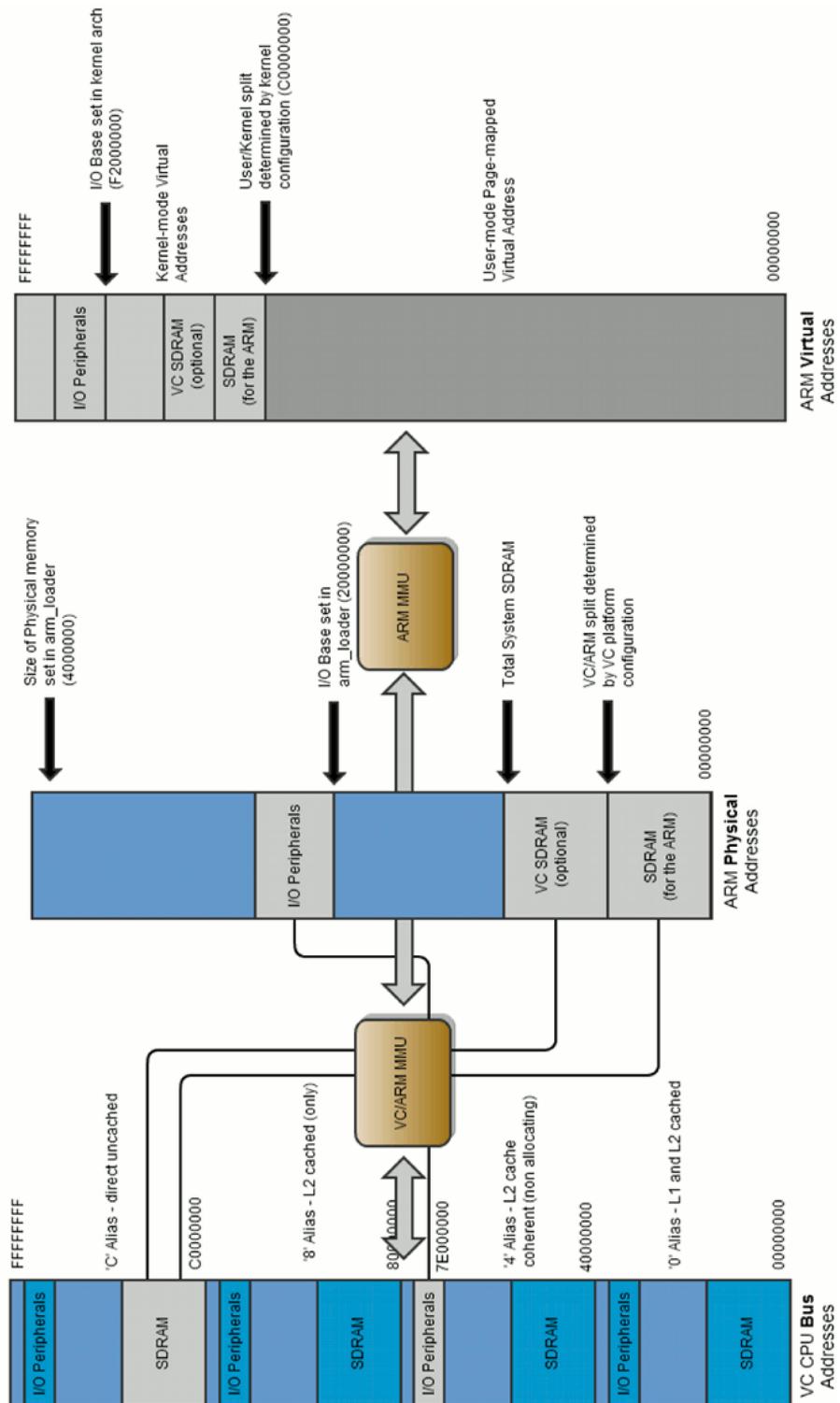
บลูทูธ ได้รับการออกแบบมาเพื่อใช้กับ อุปกรณ์ที่ มีขนาดเล็ก หรือ สามารถเคลื่อนย้ายได้ ง่าย เช่น โทรศัพท์เคลื่อนที่ แท็บเล็ต หูฟัง ลำโพง คล็อก รถยนต์ เป็นต้น เพื่อแอปพลิเคชันต่าง ๆ สามารถเชื่อมต่อกัน รับส่งหรือถ่ายโอนไฟล์ภาพ, เสียง, ข้อมูล โดยการใช้งานบลูทูธ จะต้องมีการ Pair up หรือจับคู่ เป็นกลไกรักษาความปลอดภัย โดยผู้ใช้อุปกรณ์บลูทูธ ทั้งสองฝ่าย จะต้องป้อนรหัสเดียวกันก่อนการเชื่อมต่อ

โดยอาศัยโปรไฟล์ (Profile) สำคัญที่มีในอุปกรณ์ทั้งสอง เช่น

- Human Interface Device Profile (HID) สำหรับเชื่อมต่ออุปกรณ์ เช่น เม้าส์ คีย์บอร์ด ของเครื่องคอมพิวเตอร์
- Dial-up Networking Profile (DUN) สำหรับใช้โน็ตบุ๊ก หรือ เครื่องคอมพิวเตอร์ เชื่อมต่อ กับอินเทอร์เน็ตผ่านโทรศัพท์เคลื่อนที่
- Advanced Audio Distribution Profile (A2DP) สำหรับเชื่อมต่อหูฟังชนิดบลูทูธ

รายละเอียดเพิ่มเติม ผู้อ่านสามารถค้นคว้าได้ที่ [Wikipedia](#) การทดลองเพื่อแสดงรายละเอียดของ WiFi และ Bluetooth ในการทดลองที่ 9 ภาคผนวกที่ [1](#)

6.9 หลักการ Memory Mapped Input/Output



รูปที่ 6.16: การเชื่อมโยงระหว่าง เวอร์ชวล แอดเดรส (ขวา) และเดรสนายภาพ (กลาง) และ VC/CPU บัส แอดเดรส (VC/CPU Bus Address) ที่มา: [Broadcom Corp. \(2012\)](#) หมายเหตุ VC: VideoCore

ตารางที่ 6.4: ตารางเชื่อมโยงระหว่าง VC/CPU บัสแอดเดรส อุปกรณ์อินพุต/เอาต์พุต (I/O Peripherals) และหมายเลขหัวข้อของตำราเล่มนี้ เริ่มต้นที่หมายเลข 0x7E00 0000 หมายเหตุ Rx: Receiver, Tx: Transmitter, UART: Universal Asynchronous/Synchronous Receiver Transmitter ที่มา: [Broadcom Corp. \(2012\)](#)

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	6.13
0x7E00 B000	Interrupt Controller	วงจรควบคุมอินเทอร์รูปต์	6.12
0x7E00 B400	Timer	วงจรจับเวลาสำหรับการใช้งานทั่วไป	-
0x7E20 0000	General Purpose I/O	ขา GPIO ทั้งหมด	6.11
0x7E20 1000	Universal Async. Rx Tx	การสื่อสารแบบอนุกรม	-
0x7E20 3000	Pulse Code Modulation	สัญญาณเสียง	6.4
0x7E20 4000	SPI0	Serial Peripheral Interface 0	-
0x7E20 5000	Serial Controller (I ² C)	สัญญาณ I ² C	-
0x7E21 4000	SPI/BSC Slave	Serial Peripheral Interface/ Serial Controller (I ² C)	-
0x7E21 5000	mini UART, SPI1, SPI2	การสื่อสารแบบอนุกรม	-
0x7E30 0000	External Mass Media Controller	วงจรควบคุม อุปกรณ์เก็บรักษาข้อมูล	7.3
0x7E98 0000	Universal Serial Bus	USB	6.6

ภายใต้ระบบปฏิบัติการลินุกซ์ โปรแกรมต่าง ๆ มีเวอร์ชวลเมโมรีของตนเอง แต่โปรแกรมเหล่านี้จะไม่สามารถเข้าถึงแอดเดรสภายนอกโดยตรงได้เลย โดยเฉพาะอุปกรณ์อินพุต/เอาต์พุต จะเข้าถึงได้ผ่าน System Call Interface ของระบบปฏิบัติการ เท่านั้น เช่น ฟังก์ชัน printf ในไลบรารี glibc เพื่อแสดงผลข้อมูลต่าง ๆ บนหน้าจอแสดงผล รายละเอียดเพิ่มเติมในการทดลองที่ 7 ภาคผนวก G ในเชิงโครงสร้างระบบเวอร์ชูลเมโมรี เชื่อมโยงกับอินพุต/เอาต์พุต ผ่านทาง MMU (Memory Management Unit) ตัวที่ 1 (ARM MMU) จะแปลเวอร์ชูลแอดเดรสให้กับหน่วยเป็นแอดเดรสภายนอก ตามรูปที่ [5.4](#) และ MMU ตัวที่ 2 (VC/ARM MMU) จะแปลแอดเดรสภายนอกให้กับหน่วยเป็นบัสแอดเดรสที่เชื่อมซึ่พิยูกับหน่วยความจำและอุปกรณ์อินพุต/เอาต์พุตต่าง ๆ ตามรูปที่ [6.16](#)

ในเชิงของฮาร์ดแวร์ TLB ทำงานร่วมกันกับเพจテー�เบล เรียกว่า ARM MMU ทำหน้าที่แปลเวอร์ชูลแอดเดรสเป็นแอดเดรสภายนอก หลังจากนั้น VC (VideoCore)/ARM MMU จะถูกแปลแอดเดรสภายนอกเป็น VC/CPU บัสแอดเดรส ซึ่งเชื่อมต่อกับซีพิคью วิดีโโคอร์ อุปกรณ์อินพุต/เอาต์พุต และหน่วยความจำชนิด SDRAM โดยในรูปที่ [6.16](#) แอดเดรสภายนอก 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 การเชื่อมต่อ กับอุปกรณ์เหล่านี้จะใช้การตั้งค่า VC/CPU บัสแอดเดรส เพื่ออ่านข้อมูลและเขียนข้อมูลเหล่านี้ แล้ว

แปล VC/CPU บัส แอดเดรสเหล่านี้กับ แอดเดรสภายนอกในรูปที่ 6.16 การอ่านหรือเขียนข้อมูลไปยังแอดเดรสภายนอกเหล่านี้ทำได้การใช้คำสั่ง LDR และ STR ที่เวอร์ชัลแอดเดรสเพื่อให้ ARM MMU แปลเป็น แอดเดรสภายนอก และ VC/CPU MMU แปลเป็น VC/CPU บัส แอดเดรสที่ถูกต้อง โดยผู้ผลิตชาร์ดแวร์ได้กำหนดหมายเลข VC/CPU บัส แอดเดรสสำหรับอุปกรณ์อินพุต/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 แอดเดรสภายนอก 0x0000 0000 ถูกemap เป็นบัส แอดเดรสที่ตำแหน่ง 0xC000 0000 ในพื้นที่สีเทา โดยจะมีตัวแปร VC/ARM Split เป็นตัวกำหนดขนาดหน่วยความจำที่จะแบ่งไว้ให้กับจีพียูซึ่งจะได้ทดลองเปลี่ยนแปลงค่าในการทดลองที่ 9 ภาคผนวก I

6.10 หัวเชื่อมต่อ 40 ขา (40-Pin Header)

เนื่องจากบอร์ดตระกูล Raspberry Pi นี้ถูกออกแบบให้มีราคาอยู่ในช่วง 10-20 ดอลลาร์ ผู้ออกแบบจึงเขียนชีมต่อของทางส่วนของชิป BCM283x มาให้ผู้อ่านได้เรียนรู้ทางหัวเชื่อมต่อจำนวน 40 ขาในรูปที่ 6.1 ซึ่งประกอบด้วย ส่วนต่าง ๆ เหล่านี้

- ขา GPIO (General Purpose input/output) ได้แก่ ขา GPIO00-GPIO27 สำหรับใช้เชื่อมต่อกับวงจรที่ต้องการทดลอง
- ขาสำหรับการสื่อสารชนิด UART ได้แก่ ขา RxD0/TxD0, ขา RxD1/TxD1 สำหรับเชื่อมต่อกับบอร์ดอื่น ๆ ผ่านทางพอร์ต UART ซึ่งมีอัตราบิตเรตสูงสุด 115,000 บิตต่อวินาที
- ขาสำหรับการเชื่อมต่อกับชิปภายนอกชนิด SPI ได้แก่ ขา SPI1_CE0, SPI1_CE1, SPI1_CE2, SPI1_MISO, SPI1_MOSI, SPI1_SCLK สำหรับเชื่อมต่อกับบอร์ดอื่น ๆ ผ่านทางพอร์ต SPI ซึ่งมีอัตราบิตเรตสูงกว่า 115,000 บิตต่อวินาที
- ขาสำหรับการเชื่อมต่อสัญญาณเสียง ได้แก่ ขา PCM_DIN, PCM_DOUT, PCM_FS, PCM_CLK ตามมาตรฐาน I²S ในหัวข้อที่ 6.4
- ขาสำหรับการเชื่อมต่อกับอุปกรณ์ภายนอกชนิด PWM และ PPM ได้แก่ ขา PWM0, PWM1 สำหรับการทดลอง เช่น การควบคุมการหมุนของมอเตอร์ไฟฟ้า การควบคุมความสว่างของหลอดไฟ เป็นต้น
- ขาสำหรับการเชื่อมต่อกับชิปภายนอกตามมาตรฐาน I²C ได้แก่ ขา SDA0, SCL0, SDA1, SCL1
- ขาสำหรับการเชื่อมต่อกับการ์ดหน่วยความจำ SD ได้แก่ ขา SD0_CMD, SD0_CLK, SD0_DAT0, SD0_DAT0-SD0_DAT3
- ไฟกระเจ粲ต์ที่ต่ออยู่กับบอร์ด 3.3 โวลต์, 5.0 โวลต์ สำหรับจ่ายวงจรภายนอกที่กินกระแสอย่างมาก

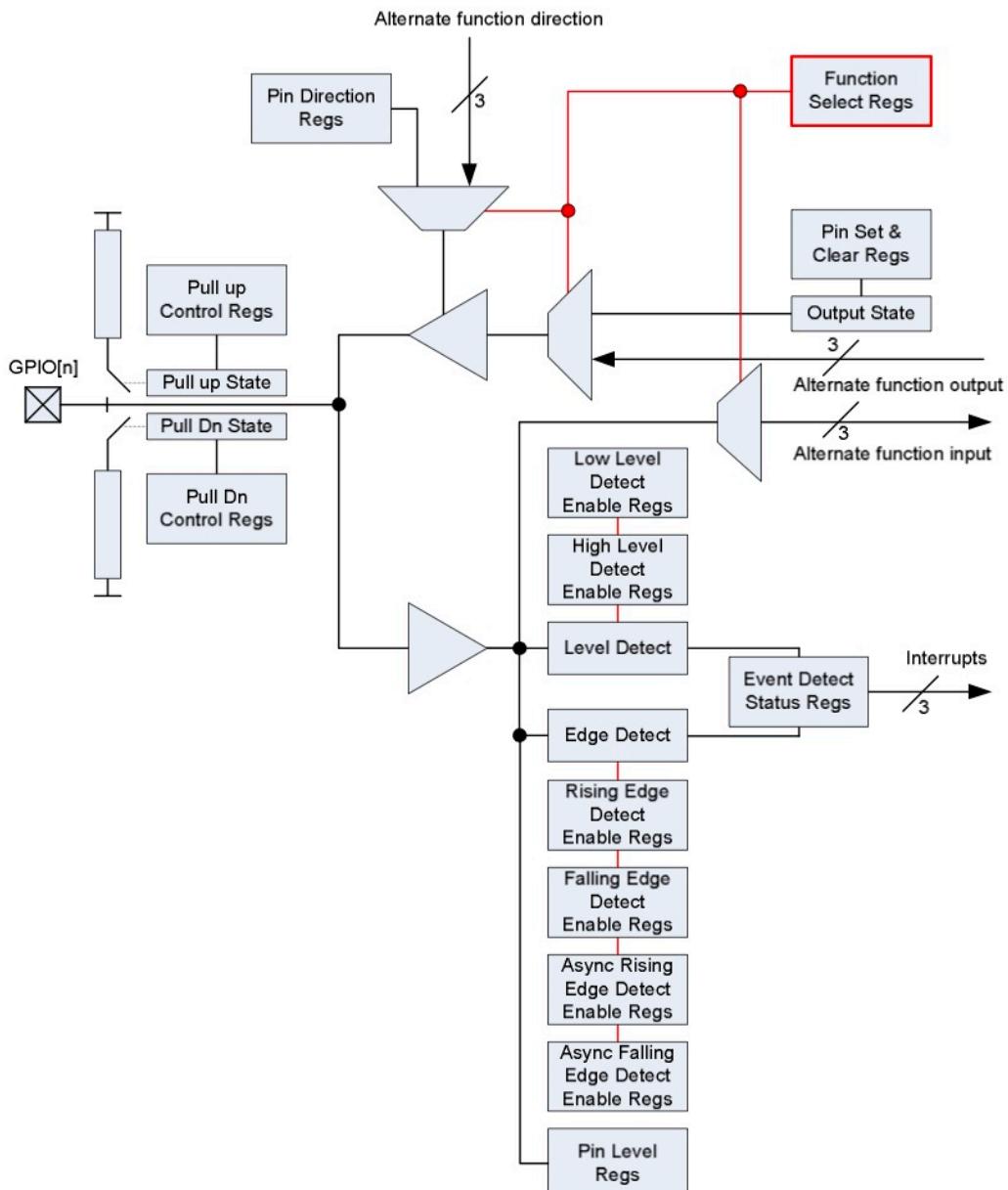
ตารางที่ 6.5: หมายเลข ชื่อขา และตัวเลือก (ชื่อสัญญาณ) ที่ 0-5 ของหัวเชื่อมต่อสายทั้ง 40 ขา (GND: Ground) ที่มา: [Broadcom Corp. \(2012\)](#), ที่มา: [Raspberry Pi \(Trading\) \(2019\)](#)

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

ขาสัญญาณเหล่านี้สามารถเขียนโปรแกรมควบคุมสั่งการด้วยภาษาแอลซีเมบลี C และ Python ในการทดลองต่าง ๆ เนื่องจากชิป BCM2837 มีฟุตพรินท์ (Foot Print) ขนาดเล็กทำให้จำนวนขา (Leads) จำกัด

เพียง 200 ขา ที่มา: [Raspberry Pi \(Trading\) \(2019\)](#) จึงต้องใช้การมัลติเพล็กซ์ (Multiplex) หรือเลือก (Select) สัญญาณที่ต้องการเพียงสัญญาณเดียวมาที่ขา โดยโปรแกรมเมอร์จะต้องกำหนดว่าขาหมายเลขนี้ มัลติเพล็กซ์เป็นสัญญาณอะไร ทำให้ขาเดียวกันมีสัญญาณตัวเลือก (Alternate function) หลายสัญญาณ ตามที่ได้สรุปในตารางที่ [6.5](#)

6.11 ขา GPIO (General Purpose Input Output)



รูปที่ 6.17: โครงสร้างภายในขา GPIO แต่ละขาของชิปตระกูลเดียวกับ BCM2835/2837/2711 ที่มา: [Broadcom Corp. \(2012\)](#)

ขา GPIO ทุกขา มีโครงสร้างภายในเหมือนกันหมดตามรูปที่ [6.17](#) ขา **GPIO[n]** คือ ขาที่ $n = 0$ ถึง $n = 53$ ทั้งหมด 54 ขาสำหรับชิปตระกูล BCM283x ผู้อ่านสามารถทำความเข้าใจง่ายๆ ซึ่งประกอบด้วย

- รีจิสเตอร์ **Function Select** ทำหน้าที่เลือกสัญญาณที่ต้องการ ได้แก่ สัญญาณ GPIO ขาเข้า ขาออก และสัญญาณตัวเลือก (Alternate function) 0 - สัญญาณเลือก 5 ตามตารางที่ 6.5 ภายใต้การควบคุมของรีจิสเตอร์นี้ ด้วยการใช้ตัวเลขจำนวน 3 บิตเพื่อเลือกการทำงานของ GPIO แต่ละขั้นนั้น 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** ใช้กำหนดทิศทางของสัญญาณเป็นขาเข้า (Direction = Input) หรือเป็นขาออก (Direction = Output)
- วงจรเอาต์พุต (Output) ซึ่งอยู่ด้านบนของรูป ประกอบด้วย
 - รีจิสเตอร์ **Pin Set** และรีจิสเตอร์ **Pin Clear** ซึ่งเก็บค่าของเอาต์พุตไว้โปรแกรมเมอร์สามารถตั้งค่ารีจิสเตอร์ตัวนี้ให้มีค่าล็อกจิก 1 สัญลักษณ์สี่เหลี่ยมคงที่ แทนอุปกรณ์มัลติเพล็กเซอร์ ทำหน้าที่เลือกสัญญาณเอาต์พุต จากสัญญาณอินพุต จำนวนหลายเส้น สัญลักษณ์สามเหลี่ยมแทนวงจรบัฟเฟอร์ซึ่งล็อกจิก $O/p = i/p$ แต่ทำหน้าที่เพิ่มกระแสให้กับสัญญาณเอาต์พุต
- 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) เพื่อเก็บค่าล็อกจิกที่ได้จากการตรวจจับสัญญาณว่าตรวจจับเหตุการณ์ได้หรือไม่
 - รีจิสเตอร์เก็บค่าระดับสัญญาณ (Pin Level) ที่รับได้ โดยใช้การสุ่มตามสัญญาณคลื่อก
 - ตัวต้านทานจำนวน 2 ตัว ซึ่งทำหน้าที่ pull up หรือ pull down ผู้ใช้สามารถใช้งาน ตัวต้านทานภายในเพื่อ pull up (Pull up) เชื่อมต่อกับไฟกระแสตรง (Vdd) 3.3 โวลต์ หรือ pull down (Pull

Down) เชื่อมต่อขา GPIO กับกราวน์ด์ (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](#) คือ รายละเอียดของแอดเดรสหมายเลข 0x7E20 0000 ในตารางที่ [6.4](#)

ตารางที่ 6.6: ตารางเชื่อมโยงระหว่าง VC/CPU บัสแอดเดรสกับรีจิสเตอร์ต่าง ๆ ของวงจร GPIO เพื่อตั้งค่าและสั่งงานขา GPIO ทุกขา โดยเริ่มต้นที่หมายเลข 0x7E20 0000 ที่มา: [Broadcom Corp. \(2012\)](#)

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	GPLEN0	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 เท่ากับ เป็นการตั้งค่าของรีจิสเตอร์ **GPFSEL0** เพื่อควบคุมการทำงานของขาที่ 0 - ขาที่ 9 ทั้งหมด

10 ขา ๆ ละ 3 บิตเท่ากับใช้ข้อมูลเพียง 30 บิต โดยขาที่ 0 ใช้งานบิตที่ 0-2 ตามลำดับจนถึงขาที่ 9 ใช้งานบิตที่ 27-29

- การเขียนข้อมูลที่แอ็ตเตรส 0x7E20 0004 - 0x7E20 0007 รวม 4 ไบต์ หรือ 32 บิต เท่ากับเป็นการตั้งค่าของรีจิสเตอร์ **GPFSEL1** เพื่อควบคุมการทำงานของขาที่ 10 - ขาที่ 19 ทั้งหมด 10 ขา ๆ ละ 3 บิตเท่ากับใช้ข้อมูลเพียง 30 บิต โดยขาที่ 10 ใช้งานบิตที่ 0-2 ตามลำดับจนถึงขาที่ 19 ใช้งานบิตที่ 27-29
- ...
- การเขียนข้อมูลที่แอ็ตเตรส 0x7E20 0014 - 0x7E20 0015 รวม 2 ไบต์ หรือ 16 บิต เท่ากับเป็นการตั้งค่าของรีจิสเตอร์ **GPFSEL5** เพื่อควบคุมการทำงานของขาที่ 50 - ขาที่ 53 ทั้งหมด 4 ขา ๆ 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
 - * แอ็ตเตรส 0x7E20 001C - 0x7E20 001F จำนวน 4 ไบต์ หรือ 32 บิต ตรงกับรีจิสเตอร์ **GPSET0** ควบคุมขาที่ 0 - 31 จำนวน 32 ขา ๆ ละ 1 บิต โดยขาที่ 0 ควบคุมโดยบิตที่ 0 จนถึงขาที่ 31 ควบคุมโดยบิตที่ 31 ตามลำดับ
 - * แอ็ตเตรส 0x7E20 0020 - 0x7E20 0022 จำนวน 3 ไบต์ หรือ 24 บิต ตรงกับรีจิสเตอร์ **GPSET1** ควบคุมขาที่ 32 - 53 จำนวน 22 ขา ๆ ละ 1 บิต โดยขาที่ 32 ควบคุมโดยบิตที่ 0 จนถึงขาที่ 53 ควบคุมโดยบิตที่ 21 ตามลำดับ
- รีจิสเตอร์ **GPCLR0** (GPIO Pin Output Clear) และ **GPCLR1** ใช้สำหรับตั้งค่าเอาต์พุตเป็นลอจิก 0
 - * แอ็ตเตรส 0x7E20 0028 - 0x7E20 002B จำนวน 4 ไบต์ หรือ 32 บิต ตรงกับรีจิสเตอร์ **GPCLR0** ควบคุมขาที่ 0 - 31 จำนวน 32 ขา ๆ ละ 1 บิต โดยขาที่ 0 ควบคุมโดยบิตที่ 0 จนถึงขาที่ 31 ควบคุมโดยบิตที่ 31 ตามลำดับ
 - * แอ็ตเตรส 0x7E20 002C - 0x7E20 002e จำนวน 3 ไบต์ หรือ 24 บิต ตรงกับรีจิสเตอร์ **GPCLR1** ควบคุมขาที่ 32 - 53 จำนวน 22 ขา ๆ ละ 1 บิต โดยขาที่ 32 ควบคุมโดยบิตที่ 0 จนถึงขาที่ 53 ควบคุมโดยบิตที่ 21 ตามลำดับ

- การตั้งขาเป็นขาอินพุต ($GPFSEL=000_2$) จะมีความซับซ้อนมากกว่า เป็นขาเอาต์พุต 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) และ รีจิสเตอร์ **GPHEN** (GPIO Pin High Detect Enable) ตัวใดตัวหนึ่งให้เท่ากับ ลอจิก 1 ที่เหลือ เป็น 0 ค่าอินพุตที่รับเข้ามายจะบันทึกในรีจิสเตอร์ **GPLEV** (Pin Level)
- การตรวจจับขอบสัญญาณ (Edge Detection) แบ่งเป็น
 - * ขอบขึ้น (Rising) และ ขอบขอล (Falling) เมื่อตรวจจับได้ จะส่งสัญญาณอินเทอร์รัปต์ เข้าสู่ซีพียูต่อไป
 - * การตรวจจับขอบสัญญาณแบ่งเป็น ชนิดซิงโครนัส (Synchronous) และ อะซิงโครนัส (Asynchronous) ความหมายคือ ซิงโครไนซ์ (Synchronize) กับสัญญาณคล็อกหรือไม่
 - . ชนิดซิงโครนัส การตรวจจับจะสุมตรวจตามความถี่ของสัญญาณคล็อกที่กำหนด หมายความว่า กับเหตุการณ์ที่เกิดขึ้นบ่อย
 - . ชนิดอะซิงโครนัส การตรวจจับจะสุมตรวจโดยไม่ขึ้นกับความถี่ของสัญญาณคล็อก หมายความว่า กับเหตุการณ์ที่นาน ๆ จะเกิดขึ้น

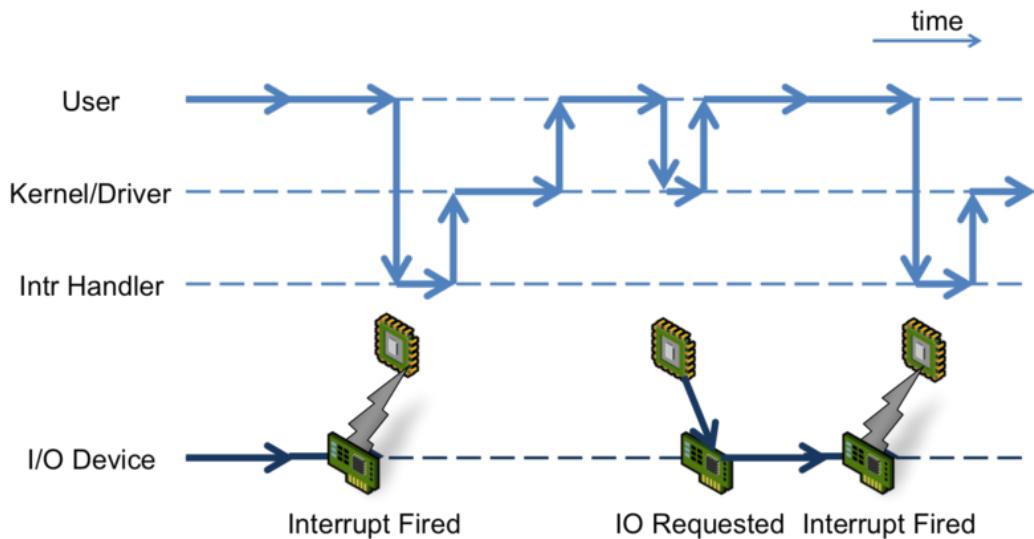
โดยการควบคุมที่รีจิสเตอร์ทั้ง 4 ชนิดนี้เกิดเป็นสัญญาณอินเทอร์รัปต์ร้องขอไปยังซีพียูได้แก่

 - . รีจิสเตอร์ **GPREN** (GPIO Synchronous Rising Edge Enable)
 - . รีจิสเตอร์ **GPFEN** (GPIO Synchronous Falling Edge Enable)
 - . รีจิสเตอร์ **GPAREN** (GPIO Asynchronous Rising Edge Enable)
 - . รีจิสเตอร์ **GPAFEN** (GPIO Asynchronous Falling Edge Enable)- รีจิสเตอร์ **GPEDS** (Pin Event Detect Status) จะเก็บค่า ลอจิก 1 เมื่อตรวจจับเหตุการณ์ที่รับเข้ามาได้ตามเงื่อนไขที่ต้องการ และจะถูกเปลี่ยนเป็น ลอจิก 0 เมื่อซีพียูตอบสนองต่อการร้องขอเรียบร้อยแล้ว รายละเอียดเพิ่มเติมในหัวข้อที่ [6.12](#)

ชิป BCM2837/BCM2711 เป็น ลอจิกชนิด CMOS (Complementary Metal Oxide Semiconductor) ลอจิก 1 ของขาทั้งหมดใช้ความต่างศักย์ 3.3 โวลต์ เท่านั้น ในขณะที่ไฟกระแสตรง 5.0 โวลต์ ใช้สำหรับจ่ายไฟให้กับซิปอิน ฯ บางตัวที่เป็น ลอจิกชนิดอื่น นักพัฒนาจะต้องระวังการเชื่อมต่อ วงจรทั้งสองชนิดเข้าด้วยกัน การใช้ขาทำงานในลักษณะ GPIO นักพัฒนาจะต้องเขื่อมต่อ วงจรและเขียนโปรแกรมควบคุมทิศทางการไหลของสัญญาณของขา GPIO แต่ละขาว่า เป็น ขารับสัญญาณอินพุต หรือ ขาปล่อยสัญญาณเอาต์พุต ตามวงจรที่ต้องไว้ให้ถูกต้อง มิเช่นนั้นอาจทำให้ชิป BCM2837 เสียหาย รวมถึงการต่อไฟกระแสตรง 3.3 และ 5.0 โวลต์ กับกราวน์ (0 โวลต์) โดยไม่ได้ตั้งใจ

ผู้อ่านสามารถใช้งานขา GPIO เหล่านี้ใน การทดลองที่ 10 ภาคผนวก [J](#) และ การทดลองที่ 11 ภาคผนวก [K](#) โดยพัฒนาโปรแกรมประยุกต์ด้วยภาษา C และ แօสเซมบลีทำงานร่วมกับไลบรารี wiringPi และ กิจกรรมท้ายการทดลองจะช่วยเสริมให้ผู้อ่านเข้าใจการตั้งค่ารีจิสเตอร์เหล่านี้ด้วย

6.12 หลักการอินเทอร์รัปต์ (Interrupt)



รูปที่ 6.18: ไดอะแกรมเวลาของโปรแกรมที่ต้องการ เชื่อมต่อ กับ อุปกรณ์อินพุต/เอ้าต์พุตทำให้เกิดอินเทอร์รัปต์เกิดขึ้นเป็นระยะ ๆ ที่มา: virtualirfan.com

ไดอะแกรมเวลาของโปรแกรมที่ต้องการ เชื่อมต่อ กับ อุปกรณ์อินพุต/เอ้าต์พุตในรูปที่ 6.18 ทำให้เกิดการขัดจังหวะการทำงานของซีพียูหรืออินเทอร์รัปต์เป็นระยะ ๆ โดยเรียงลำดับเวลาจากซ้ายสุดไปขวาสุด ซีพียูจะเปลี่ยนโหมดการทำงานของซีพียู (โปรแกรม) จากโหมดผู้ใช้ (User Mode) เป็นโหมดเครื่องเนล (Kernel Mode) เพื่อเปลี่ยนซีพียูไปรับคำสั่งของอินเทอร์รัปต์แฮนด์เลอร์ (Interrupt Handler) ซึ่งตั้งอยู่ที่แอดเดรสหรือเรียกว่า เวกเตอร์ (Vector Table) ของตาราง Interrupt Vector ในการทดลองที่ 9 หัวข้อที่ 1.3.2 ณ ตำแหน่ง Virtual kernel memory layout

บางครั้งเราเรียกอินเทอร์รัปต์แฮนด์เลอร์ว่า อินเทอร์รัปต์เซอร์วิส รูทีน (Interrupt Service Routine) ย่อว่า ISR เมื่อซีพียูจัดการกับข้อมูลที่ได้รับแล้ว ซีพียูจะเปลี่ยนกลับไปทำงาน(โปรแกรม)ในโหมดผู้ใช้ต่อ หากโปรแกรมมีความต้องการจะเชื่อมต่อกับอุปกรณ์อินพุต/เอ้าต์พุต โปรแกรมจะต้องร้องขอให้เครื่องเนลสั่งงานอุปกรณ์เหล่านั้นผ่านทางดีไวซ์ไดเรเวอร์ในรูป แล้วจึงกลับไปทำงาน(โปรแกรม)ในโหมดผู้ใช้ต่อ จนกว่าจะมีเหตุการณ์เกิดขึ้น สัญญาณร้องขออินเทอร์รัปต์จะขัดจังหวะการทำงานของซีพียู ยกตัวอย่าง เมื่อนั้นคำสั่งในอินเทอร์รัปต์แฮนด์เลอร์จะเป็นผู้จัดการขบวนการรับส่งข้อมูลแทนโปรแกรมแอปพลิเคชัน เพื่อให้ซีพียูไม่เสียเวลาอคoyer (Latency) การทำงานของวงจร IO มากจนทำให้ซีพียูไม่มีโอกาสทำงานโปรแกรมอื่น ๆ

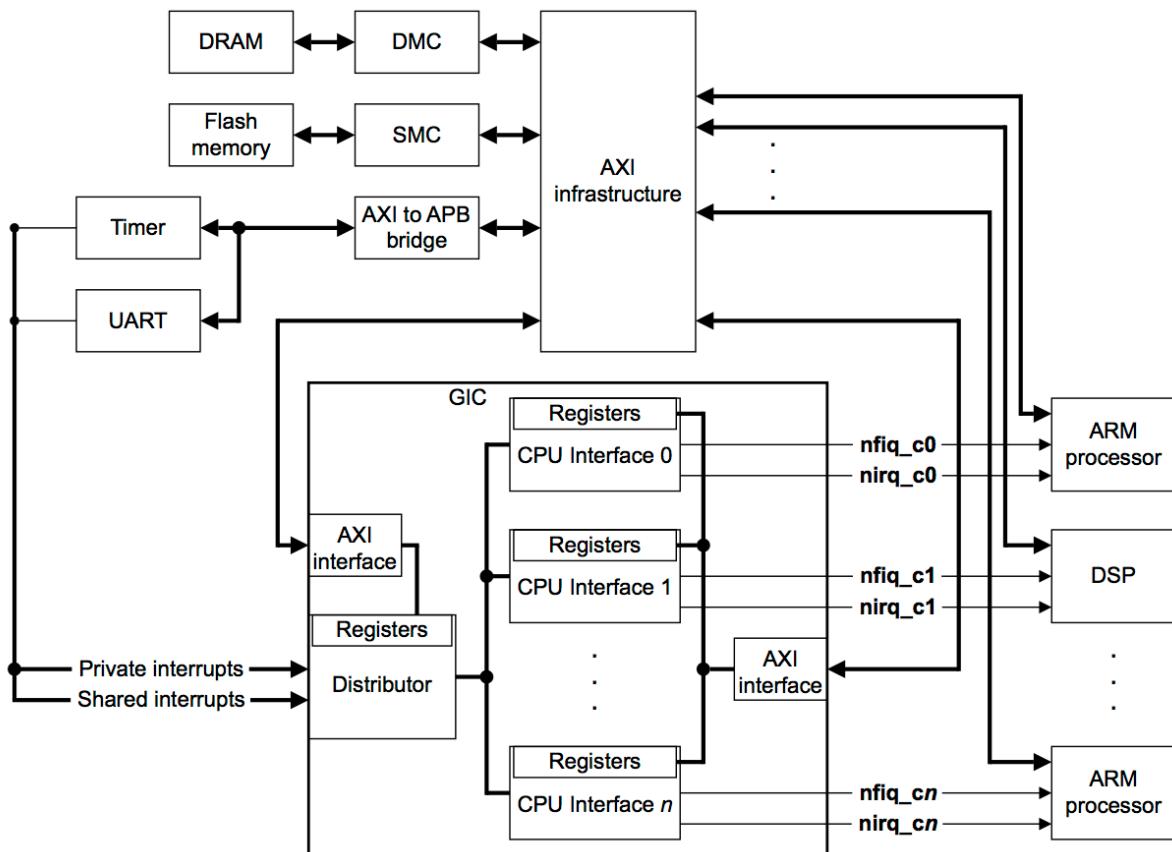
สัญญาณร้องขออินเทอร์รัปต์ (Interrupt Request) คือ สัญญาณที่เกิดขึ้นจากอุปกรณ์อินพุต/เอ้าต์พุตจากจีพียูและจากเหตุการณ์พิเศษ สัญญาณเหล่านี้จะทำให้ CPU หยุดพักโปรแกรมที่กำลังรันอยู่เป็นการชั่วคราว และเปลี่ยนไปรับคำสั่งในอินเทอร์รัปต์แฮนด์เลอร์ หรือ อินเทอร์รัปต์เซอร์วิส รูทีน เพื่อตอบสนอง (Respond) หรือให้บริการ (Service) ต่อเหตุการณ์ที่เกิดขึ้น การตอบสนองหรือการบริการ เพื่อกำหนดขั้นตอนการทำงานของซีพียูในการตอบสนองการอินเทอร์รัปต์

รายละเอียดการทำงานของอินเทอร์รัปต์ในกรณีศึกษาซีพียู ARM มีขั้นตอนดังนี้

1. เกิดเหตุการณ์อินเทอร์รัปต์ (Interrupt Event): เหตุการณ์แบ่งเป็น 2 ชนิด คือ

- อินเทอร์รัปต์จากการทำงานของฮาร์ดแวร์ (Hardware Interrupts): เหตุการณ์ที่เกิดจาก การทำงานร่วมกับอุปกรณ์อินพุตและเอาต์พุต เช่น บูมเกดต่าง ๆ เช่นเซอร์ต่าง ๆ ในตัวอุปกรณ์ วงจรเชื่อมต่อกับทางขา GPIO, ตัวจับเวลา (Timer) ถึงเวลาที่ตั้งไว้, การแปลงแอนะล็อกเป็น ดิจิทัล เสร์จสมบูรณ์, ตัวจับเวลาเวอท์ดอก (Watchdog Timer) การรับส่งข้อมูลแบบอนุกรม (Serial communication) เป็นต้น
 - อินเทอร์รัปต์จากการทำงานของซอฟต์แวร์ (Software Interrupts): เหตุการณ์ที่เกิดจาก การทำงานหรือสั่งการโดยซอฟต์แวร์ เช่น การเรียกใช้บริการจากระบบปฏิบัติการ ความผิดพลาดของโปรแกรม เป็นต้น
2. ฮาร์ดแวร์ที่เกี่ยวข้องกับเหตุการณ์ส่งสัญญาณร้องขอการอินเทอร์รัปต์ (Interrupt Request) ไปยังวงจรบริหารจัดการอินเทอร์รัปต์ วงจนนี้เรียกว่า อินเทอร์รัปต์คอนโทรลเลอร์
3. อินเทอร์รัปต์ คอนโทรลเลอร์ทำหน้าที่จัดลำดับความสำคัญของสัญญาณร้องขอตามชนิดของ ฮาร์ดแวร์ และ ลำดับเวลา ที่สัญญาณร้องขอเข้ามา แล้วจึงกระจายสัญญาณร้องขอไปให้แก่ ประมวลผลที่เหมาะสม เช่น ว่างอยู่ หรือกำลังปฏิบัติงานที่มีความสำคัญน้อยกว่า
4. ซีพียูจำนวนหนึ่งแกนประมวลผลปฏิบัติตามขั้นตอนของ ISR : แกนประมวลผลนั้นจะหยุดพัก การทำงานได้ ณ เวลาหนึ่น เพื่อปฏิบัติตามขั้นตอนของฟังก์ชันที่เขียนไว้ล่วงหน้า และให้บริการตามเหตุการณ์ที่ร้องขอภายใต้โหมดเดอร์เนล

กรณีศึกษา วงจรควบคุม อินเทอร์รัปต์ คอนโทรลเลอร์ ของซีพียู ARM ชื่อ Generic Interrupt Controller (GIC) ในรูปที่ 6.19 ออกแบบมาสำหรับซีพียู ARM Cortex A รุ่นต่าง ๆ เพื่อรองรับการอินเทอร์รัปต์ที่ซับซ้อน เนื่องจากซีพียูจำนวน 2 แกนประมวลผลขึ้นไปและมีอุปกรณ์อินพุต/เอาต์พุตที่หลากหลาย โดยทั้งหมดเชื่อมต่อกันด้วย ARM Advanced eXtensible Interface (AXI) ซึ่งเป็นบัสชนิดหนึ่งมีโครงสร้างและโปรโตคอลที่ซับซ้อน ผู้อ่านสามารถค้นคว้ารายละเอียดและวิธีการของ AXI เพิ่มเติมที่ [wikipedia](#)



รูปที่ 6.19: โครงสร้างส่วนหนึ่งภายในชิป BCM283x แสดงการเชื่อมต่อแกนประมวลผลต่าง ๆ กับวงจร Generic Interrupt Controller (GIC) ผ่าน ARM Advanced eXtensible Interface (AXI) หมายเหตุ APB: ARM Peripheral Bus ที่มา: arm.com

เหตุการณ์อินเทอร์รัปต์มีความสำคัญ (Priority) แตกต่างกัน รูปที่ 6.20 แสดงไดอะแกรมเวลา การทำงานของ GIC เมื่อเกิดการเกิดอินเทอร์รัปต์ซ้อน (Nested Interrupt) นั่นคือ สัญญาณร้องขออินเทอร์รัปต์ (Interrupt Request) จำนวนสองสัญญาณ คือ Interrupt M และ Interrupt N มาถึงยังวงจรควบคุมไม่พร้อมกัน การตอบสนองต่อการร้องขอที่เกิดขึ้นนั้นจะขึ้นกับความสำคัญ (Priority) เป็นหลัก ซึ่งสามารถหยุดพัก (Pending) การร้องขอที่มาถึงก่อนแต่มีความสำคัญต่ำกว่าชั่วคราว เพื่อตอบสนองต่อสัญญาณการร้องขอที่มีความสำคัญสูงกว่า (Higher Priority) และจึงตอบสนองสัญญาณที่พักไว้ต่อ ตามขั้นตอนต่อไปนี้

แกนนอนในรูปที่ 6.20 เป็นแกนเวลาเริ่มต้นจากเวลาที่ T0 จนถึงเวลาที่ T55 วงจรภายในรูปนี้ทำงานที่ขอบขั้นของสัญญาณคล็อกเท่านั้น ตามลำดับดังนี้

- คาบเวลา T1 วงจรแจกจ่ายอินเทอร์รัปต์ (Distributor) ตรวจจับการร้องขอของ Interrupt M ของอุปกรณ์ M
- คาบเวลา T2 วงจรแจกจ่ายอินเทอร์รัปต์พัก (Pending) การร้องขอจาก Interrupt M
- คาบเวลา T4 วงจรแจกจ่ายอินเทอร์รัปต์มอบหมาย Interrupt M ให้กับแกนประมวลผล n ผ่านทางสัญญาณร้องขอการอินเทอร์รัปต์ **nirq_c<n>**

- ควบเวลา T5 ว่าจะแจกจ่ายอินเทอร์รัปต์สามารถตรวจจับการร้องขอของ Interrupt N ซึ่งมีความสำคัญสูงกว่า Interrupt M จึงต้องพักรการทำงานของ Interrupt M ต่อไป

- ควบเวลา T6 ว่าจะแจกจ่ายอินเทอร์รัปต์มอบหมาย Interrupt N ให้กับแกนประมวลผลที่ n ผ่านทางสัญญาณร้องขอการอินเทอร์รัปต์ `nirq_c<n>` แทน

- ควบเวลา T8 ซึ่งมีผู้ตอบกลับ (Acknowledge) ด้วยสัญญาณ ACK สัญญาณร้องขอของ Interrupt N

- ควบเวลา T9 แกนประมวลผลที่ n อ่านค่าและตอบสนองต่อ `nirq_c<n>` โดยเปลี่ยนค่าของรีจิสเตอร์ Interrupt Acknowledge Register จากโลจิก 0 ให้เป็นโลจิก 1

- ควบเวลา T10 ว่าจะแจกจ่ายอินเทอร์รัปต์ตั้งค่าสถานะของ Interrupt N จาก Pending เป็น Active และเปลี่ยนแปลงค่ารีจิสเตอร์ Active Status Register

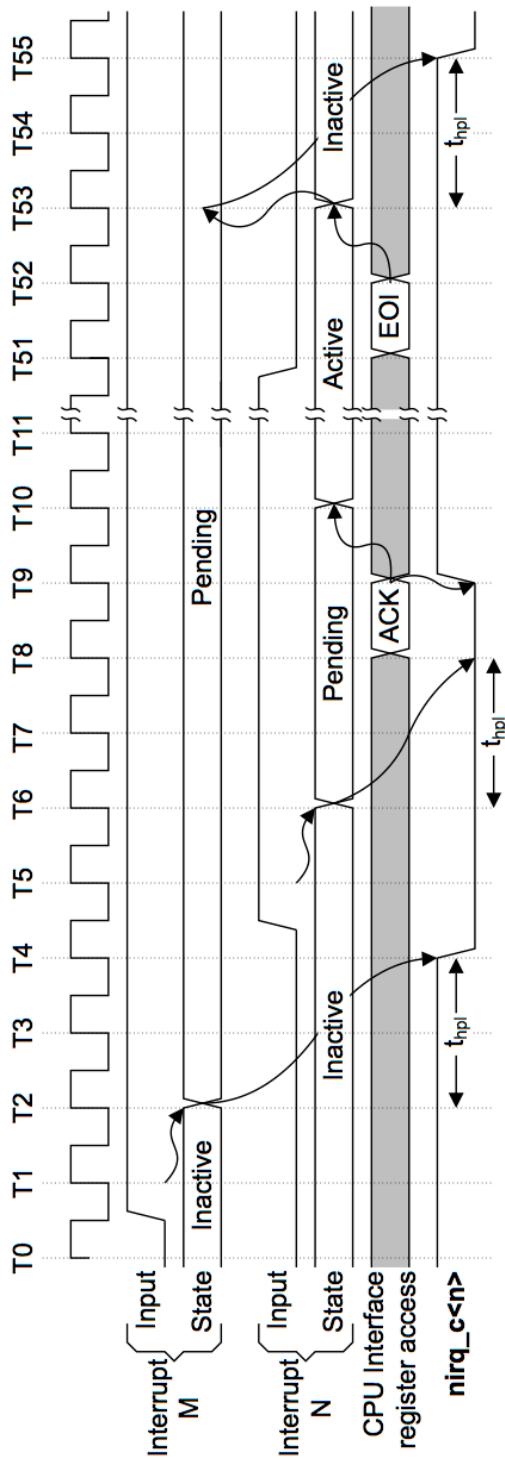
- ควบเวลา T11-T50 แกนประมวลผลที่ n ปฏิบัติตามคำสั่งใน ISR ของ Interrupt N จนแล้วเสร็จ

- ควบเวลา T51 แกนประมวลผลที่ n บันทึกค่ารีจิสเตอร์เป็น End of Interrupt (EOI) ด้วยหมายเลข INTID (Interrupt ID) ของ Interrupt N

- ควบเวลา T52 ว่าจะแจกจ่ายอินเทอร์รัปต์ตั้งค่าสถานะของ Interrupt N เป็น Inactive และเปลี่ยนแปลงค่ารีจิสเตอร์ Active Status Register เป็นโลจิก 0 เพื่อรับสัญญาณอินเทอร์รัปต์ในอนาคต

- ควบเวลา T53 ว่าจะแจกจ่ายอินเทอร์รัปต์ แจ้ง ว่าจะ อินเตอร์เฟส ของ แกน ประมวล ผล ที่ n ว่า สัญญาณ interrupt M ยังคงอยู่และมีความสำคัญสูงที่สุด ณ เวลานี้

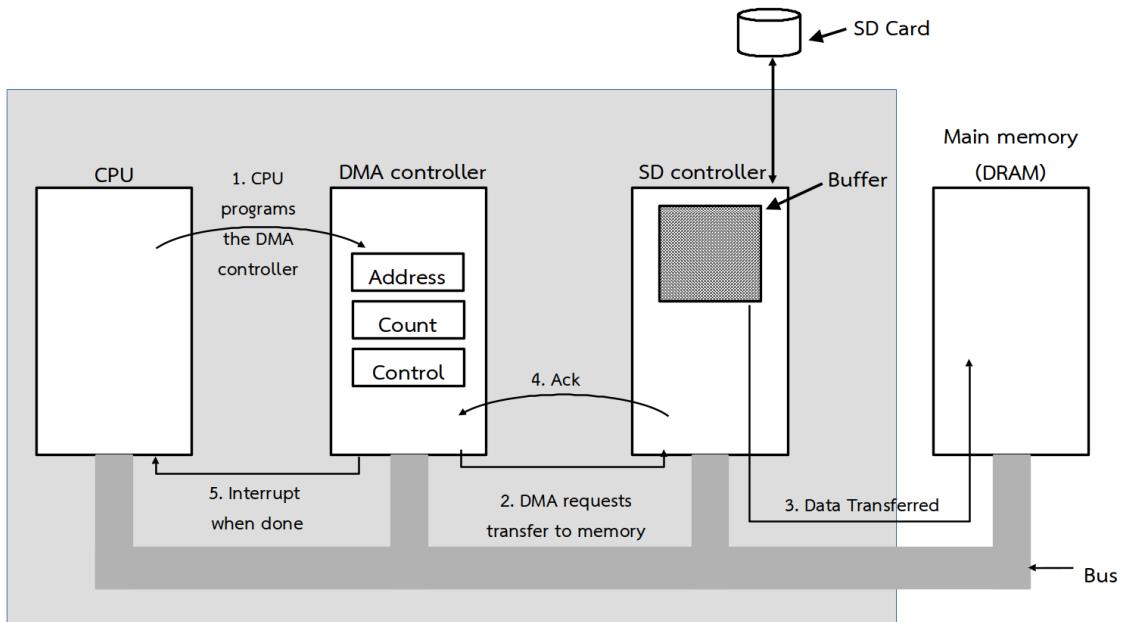
- ควบเวลา T55 ว่าจะมอบหมาย Interrupt M ให้กับแกนประมวลผลที่ n โดยเปลี่ยนสัญญาณร้องขอ `nirq_c<n>` ให้กลับเป็นโลจิก 0 เพื่อขัดจังหวะการทำงานเป็นลำดับถัดไป



รูปที่ 6.20: “โค้ดโปรแกรมเวลา การทำงานของ Generic Interrupt Controller (GIC) เพื่อตอบสนองต่อสัญญาณร้องขออินเทอร์รัปต์ที่มีความสำคัญไม่เท่ากัน” ที่มา: [ARM Ltd. \(2017\)](#)

ในการทดลองที่ 11 ภาคผนวก K ผู้อ่านสามารถใช้งานขา GPIO ทำงานร่วมกับการอินเทอร์รัปต์ โดยพัฒนาโปรแกรมประยุกต์ด้วยภาษา C และภาษาแอลซเมบลิ์ทำงานร่วมกับไลบรารี wiringPi และกิจกรรมท้ายการทดลอง จะช่วยเสริมให้ผู้อ่านเข้าใจการทำงานของอินเทอร์รัปต์เพิ่มมากขึ้น

6.13 หลักการเข้าถึงหน่วยความจำโดยตรง (Direct Memory Access)

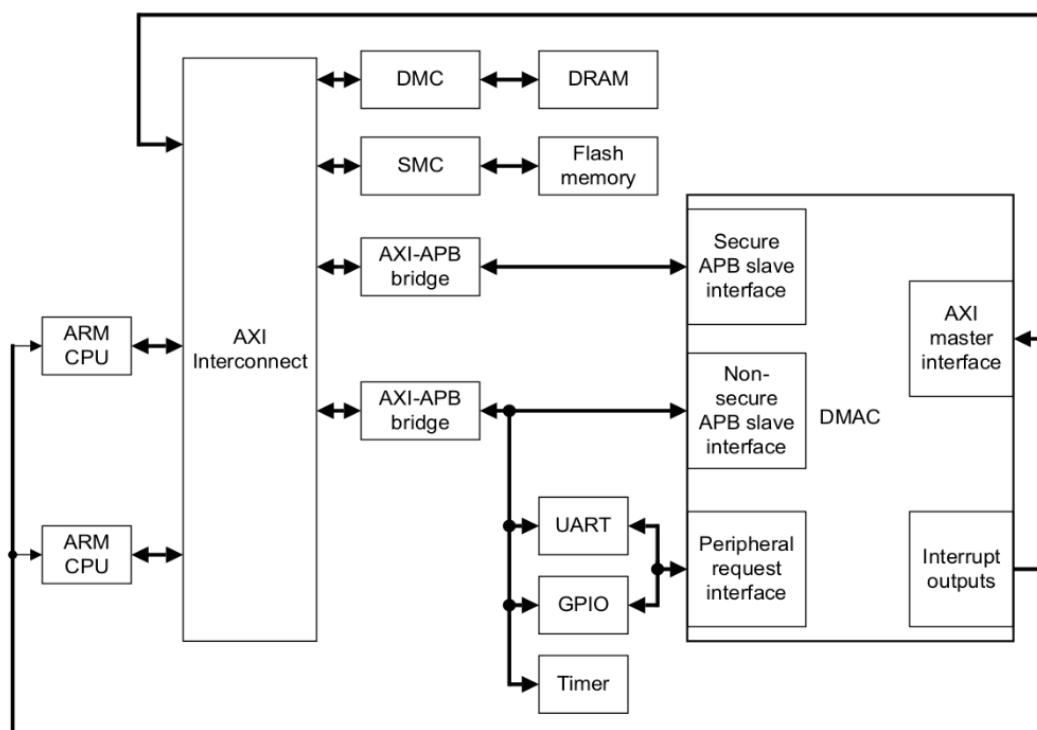


รูปที่ 6.21: ลำดับการทำงานของ DMA Controller (DMAC) เพื่อควบคุมการอ่านข้อมูลจาก การ์ดหน่วยความจำ SD ไปบันทึกในหน่วยความจำหลัก (Main Memory)

การรับส่งข้อมูลผ่าน GPIO มีข้อจำกัดในเรื่องของความเร็ว จึงไม่เหมาะสมกับการถ่ายโอนข้อมูลจำนวนมาก โดยเฉพาะ อุปกรณ์ที่ เกี่ยวข้อง กับ ข้อมูล จำนวนมาก ในเวลา ที่ จำกัด ดังนั้น นัก พัฒนา ชีพียู จึงได้ ออกแบบ แบบ บวน การ อิน พุต / เอาต์ พุต เรียกว่า การเข้าถึงหน่วยความจำโดยตรง (Direct Memory Access) เพื่อ ทำงาน ร่วม กับ การ อิน เทอร์รัปต์ ยก ตัวอย่าง เช่น การรับส่ง ข้อมูล ผ่าน เครือข่าย Ethernet ในรูปของ เพرم ข้อมูล ที่ มี ความ ยาว ขั้น ต่ำ 64 ไบต์ และ ไม่ เกิน 1522 ไบต์ ดังนั้น ใน การ รับ/ส่ง ข้อมูล กับ บัฟเฟอร์ ของ อุปกรณ์ อิน พุต / เอาต์ พุต DMA จะ ทำ หน้าที่ เคลื่อนย้าย ข้อมูล จาก บัฟเฟอร์ กับ หน่วยความจำ หลัก ยก ตัวอย่าง เช่น รูปที่ 6.21 แสดง ลำดับ การ ทำงาน ของ DMAC เพื่อ ควบคุม การ อ่าน ข้อมูล จาก บัฟเฟอร์ ภายใน วงจร SD Controller ซึ่ง ทำ หน้าที่ ควบคุม การ ทำงาน ของ การ์ด หน่วยความจำ SD ไปบันทึก ใน หน่วยความจำ หลัก (SDRAM) ประกอบด้วย ขั้นตอน ต่อไปนี้

1. ชีพียู โปรแกรม การ ทำงาน ของ DMAC โดย ตั้ง ค่า รีจิส เโทรร์ ที่ เกี่ยวข้อง ได้แก่ แอดเดรส เริ่ม ต้น ใน หน่วยความจำ หลัก จำนวน ไบต์ ที่ ต้องการ อ่าน ค่า (Count) และ ราย ละเอียด อื่น ๆ ของ ตู้ DMA Controller (DMAC) ถูก แมพ บน บัส แอดเดรส เริ่ม ต้น ที่ หมายเลข 0x7E00 7000 ของ ตาราง ที่ 6.4 สำหรับ กรณี ศึกษา ชิป BCM2837
2. DMAC เริ่ม ต้น ทำการ อ่าน ข้อมูล จาก บัฟเฟอร์ ภายใน SD Controller ไปบันทึก หน่วยความจำ หลัก ณ แอดเดรส เริ่ม ต้น ที่ ตั้ง ค่า จากข้อ 1

3. ข้อมูลจะถูกอ่านจากบอร์ดภายใน SD Controller (ต้นทาง) เดินทางผ่านบัสความเร็วสูง AXI และเขียนลงในหน่วยความจำหลักที่แอดเดรสเริ่มต้นจนแล้วเสร็จตามจำนวนไบต์ (Count) ที่ตั้งค่าไว้
4. วงจร SD Controller ส่งสัญญาณตอบรับ ACK ไปยัง DMAC
5. DMAC ส่งสัญญาณอินเทอร์รัปต์แจ้งซีพียูว่าดำเนินการแล้วเสร็จผ่านทางวงจรควบคุมอินเทอร์รัปต์
6. ซีพียุตอบสนองต่อสัญญาณอินเทอร์รัปต์ที่ได้รับจาก DMAC



รูปที่ 6.22: โครงสร้างส่วนหนึ่งภายในชิป BCM283x/BCM2711 แสดงการเชื่อมต่อแกนประมวลผลต่างๆ กับวงจร DMAC (DMA Controller) ผ่านวงจร Advanced eXtensible Interface (AXI) และ AXI-APB Bridge ที่มา: arm.com

ตำราเล่มนี้จะใช้กรณีศึกษา DMAC ของซีพียู ARM ในรูปที่ 6.22 โครงสร้างส่วนหนึ่งภายในชิป BCM2837 ซีพียู Cortex A53/A72 ทั้ง 4 แกนประมวลผล เชื่อมต่อกับวงจร DMAC ผ่าน ARM Advanced eXtensible Interface (AXI) และ AXI-APB Bridge ชิป SDRAM จะเชื่อมต่อกับบัส AXI ในขณะที่อุปกรณ์อินพุต/เอาต์พุตส่วนใหญ่จะเชื่อมต่อกับบัส APB ซึ่งมีความเร็วต่ำกว่าทำให้ DMAC สามารถรับข่าวสาร DMA ได้พร้อมกัน 16 ช่องหรือ 16 อุปกรณ์ โดยแต่ละช่องสามารถทำงานได้อิสระจากกัน

ข่าวสาร 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 จะส่งสัญญาณอินเทอร์รัปต์มายัง GIC เพื่อแจ้งเตือนซีพียูว่างานเสร็จสิ้นแล้ว ตามที่ได้กล่าวไปแล้วใน

หัวข้อที่ [6.12](#) สำหรับกรณีศึกษาชิป BCM283x วงจร DMAC จะ mapped แบตเตอรี่เริ่มต้นที่ 0x7E00 7000 ของตารางที่ [6.4](#) โดยเรียงลำดับตามหมายเลขช่องดังนี้

- การทำงานของ DMA ช่อง 0 เริ่มต้นการตั้งค่าที่แอดเดรส 0x7E00 7000
- การทำงานของ DMA ช่อง 1 เริ่มต้นการตั้งค่าที่แอดเดรส 0x7E00 7100
- การทำงานของ DMA ช่อง 2 เริ่มต้นการตั้งค่าที่แอดเดรส 0x7E00 7200
- ...
- การทำงานของ DMA ช่อง 14 เริ่มต้นการตั้งค่าที่แอดเดรส 0x7E00 7E00
- การทำงานของ DMA ช่อง 15 เริ่มต้นการตั้งค่าที่แอดเดรส 0x7EE0 5000

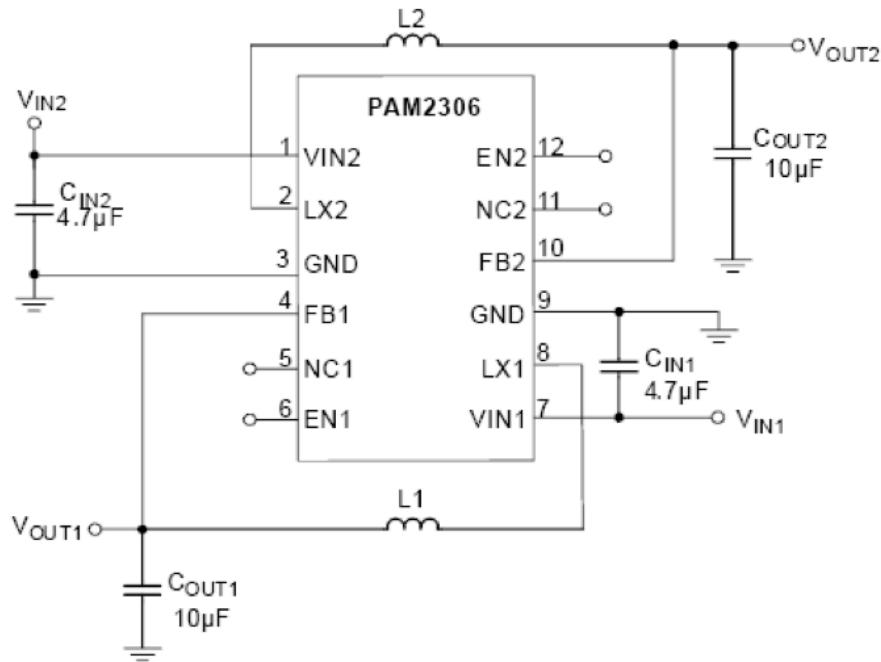
การทำงานของ DMA แต่ละช่องจะมีลักษณะเหมือนกัน คือ ตั้งค่าการทำงานของรีจิสเตอร์ด้วยชุดข้อมูล Control Block (CB) ชุดละ 256 ไบต์ CB แต่ละชุดจะเชื่อมโยงกันเป็นลิงก์ลิสต์ (Linked List) ในหน่วยความจำ เมื่อการทำงานตามรายละเอียดของของ CB ชุดแรกเสร็จสิ้น ค่า CB ชุดต่อไปจะถูกอ่านเพื่อไปตั้งค่าการทำงานชุดต่อไปโดยอัตโนมัติ ทำให้มีพื้นที่ทำงานอีกนิดหนึ่งที่สำคัญกว่าได้ต่อเนื่อง และรับทราบผลเมื่อแต่ละขบวนการแล้วเสร็จผ่านกลไกการอินเทอร์รัปต์

6.14 แหล่งจ่ายไฟ (Power Supply)

แหล่งจ่ายไฟของบอร์ดมาจากอแดปเตอร์ (Adaptor) ที่แปลงไฟฟ้ากระแสสลับ 220 โวลต์ เป็นไฟกระแสตรงความต่างศักย์ 5.1 โวลต์ โดยใช้หัวคอนเนคเตอร์ไมโคร USB จะต้องจ่ายกระแสไฟฟ้าได้ไม่น้อยกว่า 2.6 แอมป์ ทั้งนี้ เพื่อให้บอร์ด Pi3/Pi4 สามารถทำงานได้เต็มประสิทธิภาพ กระแสจะไหลผ่านพิวเวอร์แลบได้โดย เพื่อจ่ายไฟให้กับบอร์ด Pi3/Pi4 เล้าจึงแปลงไฟกระแสตรงความต่างศักย์ 5.1 โวลต์ให้ลดลง (Step Down) เป็นไฟกระแสตรงความต่างศักย์ 3.3 และ 1.8 โวลต์ด้วยชิป PAM2306 และ 1.2 โวลต์ด้วยชิป RT8088A ตามลำดับ

ชิป PAM2306 จะทำหน้าที่แปลงไฟกระแสตรงความต่างศักย์ 5.0 โวลต์ ให้เป็นไฟกระแสตรงความต่างศักย์ 3.3 โวลต์ เพื่อเลี้ยงชิปต่าง ๆ และวงจรทั้งหมด PAM2306 รองรับอินพุตไฟกระแสตรงตั้งแต่ 2.5 ถึง 5.5 โวลต์ โดยสามารถแปลงเอาต์พุตโวลต์เจได้เป็นสองระดับพร้อม ๆ 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) เพื่อให้เกิดเสถียรภาพ และรักษาอายุการใช้งานของแหล่งจ่ายไฟ เมื่อต้องจ่ายกระแสมากขึ้น และเปลี่ยนมาใช้หลักการ PSM (Pulse-Skipping Modulation) เพื่อลดกระแสในขณะที่จ่ายกระแสอยู่

ขา V_{IN1} จะถูกแปลงให้มีความต่างศักย์ลดลงเป็น V_{OUT1} ขา V_{IN2} จะถูกแปลงให้มีความต่างศักย์ลดลงเป็น V_{OUT2} โดยจะควบคุมการเปิดปิดด้วยขา EN1 (Enable1) และ EN2 (Enable2) ตามลำดับ ค่าโวลต์เจทั้ง V_{OUT1} และ ค่าโวลต์เจทั้ง V_{OUT2} ถูกกำหนดที่หมายเลขตัวถังของชิปมาจากการผลิต ผู้



รูปที่ 6.23: การแปลงไฟกระแสตรง 5 โวลต์เป็นไฟกระแสตรงด้วยชิป PAM 2306 DC-DC Covertor คู่กำหนดค่าเอาต์พุตด้วยค่าตัวเหนี่ยวนำ L_1 และ L_2 หน่วยเป็นไมโครเอนรี ที่มา: [Diodes, Inc. \(2012\)](#)

ออกแบบต้องเลือกค่าตัวเหนี่ยวนำให้เหมาะสม สมเพื่อกำหนดความต่างศักย์ของ 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, Inc. \(2012\)](#)

V_{OUT} (โวลต์)	L (μH)
1.2	2.2
1.5	2.2
1.8	2.2
2.5	4.7
3.3	4.7

เอกสารคุณสมบัติความต้องการด้านกำลังไฟฟ้าสำหรับบอร์ด 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 โดยผู้ใช้มิ่งต้องกังวลว่าการจ่ายกระแสมากเกินไปแล้วจะเป็นผลเสีย เพราะบอร์ดจะบริโภคกระแสเท่าที่จำเป็นและมีพิวส์ค่อยซวยหากกระแส 2.5 แอมเบร็ต ออกจากนี้ บอร์ด Pi3/Pi4 และ Pi2 โมเดล B+ มีชิปหมายเลข APX803 ทำหน้าที่เฝ้าระวัง (Monitor) โวลต์เจต กรณีไฟเลี้ยงจาก USB มีความต่างศักย์อยู่ในช่วง 4.63 ± 0.07 โวลต์ ไฟ LED สีแดงบนบอร์ดจะสว่างขึ้น สัญลักษณ์สายฟ้าและข้อความว่า Under Voltage ผ่านทางหน้าจอตรงมุมขวาบน จะเตือนให้ผู้ใช้ทราบว่าไฟเลี้ยงของบอร์ดมีค่าต่ำไปแต่บอร์ดยังใช้งานได้ตามปกติ

ข้อควรระวังเรื่องแหล่งจ่ายไฟให้กับบอร์ด Pi3/Pi4 คือ ขนาดและคุณภาพของสายไมโคร USB ที่จ่าย

ไฟเลี้ยง 5 โวลต์ สายที่มีขนาดเล็กเกินไปจะทำให้ความต้านทานของสายสูงขึ้น เมื่อกระแสไฟลั่นจะทำให้เกิดโวลเตจตกคร่อมในสายสูงขึ้นตาม จนทำให้บอร์ดได้รับโวลเตจไม่สูงพอ นั่นคือ ต่ำกว่า 4.63 โวลต์ จนส่งผลถึงความต่างศักย์ด้านเอาร์พตของ PAM2306 ที่อาจต่ำกว่า 3.3 โวลต์ และทำให้ชิป BCM2837 ขาดเสถียรภาพ

6.15 สรุปท้ายบท

แกนประมวลผลทั้งหมด หน่วยความจำภายในภาพ และอุปกรณ์เก็บรักษาข้อมูลของเครื่องคอมพิวเตอร์บอร์ดเดียว Pi3/Pi4 เชื่อมเข้าหากันด้วยการเชื่อมต่อความเร็วสูง ซึ่ง AXI (Advanced eXtensible Interface) ซึ่งทำหน้าที่เป็นบัสความเร็วสูง เพื่อรองรับการถ่ายโอนข้อมูลจำนวนมากด้วยกระบวนการ DMA ระหว่างหน่วยความจำภายในภาพ กับ

- อุปกรณ์เก็บรักษาข้อมูล (ในบทที่ 7)
- อุปกรณ์ USB (ในหัวข้อที่ 6.6)

ส่วนวงจร (มอดูล) ด้านอินพุต/เอาต์พุตยืน ๆ 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/เขียน รีจิสเตอร์ภายในวงจรหรือมอดูลอุปกรณ์นั้น ๆ โดยการจัดวาง (map) หมายเลขบัสแอดдресซิ่งแบล็คจากแอดдресภายในภาพ และเวอร์ชัลแอดdress อีกด้วย หนึ่งให้ตรงกับรีจิสเตอร์เหล่านั้น เรียกว่า หลักการ **Memory Mapped Input/Output** ที่มา: Tanenbaum and Austin (2012); Harris and Harris (2013) กลไกหลักที่จะช่วยแบ่งเบาภาระการทำงานด้านอินพุต/เอาต์พุตของซีพียู คือ กลไกอินเทอร์รัปต์มีวงจรควบคุมอินเทอร์รัปต์ 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. จงบอกชื่อขาที่ชิป BCM2837 สำหรับเชื่อมต่อกับการ์ดหน่วยความจำ SD ทั้งหมด

5. จงบอกรจำนวนเลนข้อมูลของการเชื่อมต่อกับจอภาพชนิด HDMI ที่ชิป BCM2837 รองรับได้สูงสุด
6. จงบอกรจำนวนเลนข้อมูลของการเชื่อมต่อกับกล้องชนิด CSI ที่ชิป BCM2837 รองรับได้สูงสุด
7. จงบอกรจำนวนเลนข้อมูลของการเชื่อมต่อกับจอภาพชนิด DSI ที่ชิป BCM2837 รองรับได้สูงสุด
8. จงบอกรตำแหน่งเริ่มต้น ตำแหน่งสิ้นสุด และขนาดของแอดเดรสบัส ในรูปที่ [6.16](#)
9. จงบอกรความหมายของ L1 and L2 cached ในรูปที่ [6.16](#)
10. จงบอกรความหมายของ L2 cache coherent (non allocating) ในรูปที่ [6.16](#)
11. จงบอกรความหมายของ L2 cached (only) ในรูปที่ [6.16](#)
12. จงบอกรความหมายของ direct uncached ในรูปที่ [6.16](#)
13. การขัดจังหวะของซีพียูเกิดขึ้นจากอุปกรณ์อินพุต/เอาต์พุตอะไรบ้าง จงยกตัวอย่างโทรศัพท์สมาร์ตโฟน
14. เหตุใดการเชื่อมต่อกับกล้องและจอภาพจึงต้องมีสัญญาณคลือกด้วย
15. เหตุใดการเชื่อมต่อกับ USB, Ethernet และ อื่น ๆ จึงไม่ต้องมีสัญญาณคลือกทั้ง ๆ ที่เป็นการส่งด้วยเทคนิค Differential Voltage Signaling เช่นเดียวกับกล้องและจอภาพ
16. นอกเหนือจากสัญญาณ WiFi และ Bluetooth ชิป BCM43438 รองรับสัญญาณใดเพิ่มเติม
17. จงอธิบายหลักการเชื่อมต่อกับ GPIO ของชิป BCM2837/BCM2711 และการขัดจังหวะ โดยตรวจจับสัญญาณอินพุตที่ขอบขาลง (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 ทำหน้าที่บันทึกและเก็บไฟล์ทั้งหมดของระบบปฏิบัติการไฟล์โปรแกรม ไฟล์ข้อมูล และไฟล์อื่น ๆ เพื่อให้เครื่องคอมพิวเตอร์สามารถทำงานได้ต่อเนื่องเมื่อปิดแล้ว เปิดเครื่องใหม่ อีกรอบ หลังจากการซัตดาวน์ การปิดเครื่องคอมพิวเตอร์อาจมีสาเหตุหลายประการ เช่น การประหดพลังงาน การบำรุงรักษาระบบ หรือกรณีฉุกเฉินไฟฟ้าดับ หรือแบตเตอรี่หมด ซึ่งอาจทำให้ไฟล์เสียหาย หรือถึงขั้นสูญหาย ดังนั้น ผู้ใช้และซอฟต์แวร์ต้องมีหน้าที่บันทึก (Save) ข้อมูลลงในไฟล์ข้อมูลระหว่างการปฏิบัติงานด้วย เช่น กัน กรณีผู้ใช้ไม่สามารถซัตดาวน์ระบบอาจทำให้ระบบไฟล์และไฟล์เสียหายได้ ทั้งนี้ขึ้นอยู่กับชนิดของระบบไฟล์

เทคโนโลยีที่ใช้สร้างอุปกรณ์เก็บรักษาข้อมูลที่ได้รับความนิยมในปัจจุบัน ได้แก่ เทคโนโลยีสารกึ่งตัวนำ เทคโนโลยีสารแม่เหล็ก และ เทคโนโลยีสารสะท้อนแสง ดังนั้น อุปกรณ์เก็บรักษาข้อมูลจึงแบ่งเป็นชนิดต่าง ๆ ตามสารที่ใช้ ดังนี้

- อุปกรณ์เก็บรักษาข้อมูลชนิดสารกึ่งตัวนำ ได้แก่ โซลิดสเตทไดร์ฟ และ หน่วยความจำการ์ด SD ซึ่งอาศัยชิปหน่วยความจำชนิดแฟลช ทำงานที่สัญญาณคลือกความถี่สูงระดับ 100 เมกะเฮิรตซ์
- อุปกรณ์เก็บรักษาข้อมูลชนิดสารแม่เหล็ก ได้แก่ ฮาร์ดดิสก์ไดร์ฟอาศัยการหมุนของajan แม่เหล็กแข็งที่ความเร็วตอบสนองสูง
- อุปกรณ์เก็บรักษาข้อมูลชนิดสารสะท้อนแสง ได้แก่ แผ่นซีดี (CD: Compact Disc) แผ่นดิวีดี (DVD: Digital Video Disc) และแผ่นบลูเรย์ (Blu-ray) อาศัยการหมุนของajan สะท้อนแสงที่ความเร็วตอบสนองกลาง

โปรดสังเกตว่า อุปกรณ์เหล่านี้ล้วนทำงานที่สัญญาณคลือกความถี่ 100-500 เมกะเฮิรตซ์ ซึ่งต่ำกว่าหน่วยความจำ SDRAM และ SRAM ซึ่งทำงานที่สัญญาณคลือกความถี่หลัก กิกะ เฮิรตซ์ แต่ อุปกรณ์เก็บรักษาข้อมูลไม่ต้องอาศัยไฟเลี้ยง เพื่อรักษาข้อมูลมิให้สูญหาย หรือเสียหายตลอดเวลา เทคโนโลยีการเก็บรักษา

ข้อมูลเหล่านี้มีความแตกต่างกัน มีข้อได้เปรียบเสียเปรียบต่างกัน โดยบทนี้จะเน้นที่ อุปกรณ์เก็บรักษาข้อมูลชนิดสารกึ่งตัวนำ และ อุปกรณ์เก็บรักษาข้อมูลชนิดสารแม่เหล็ก โดยมีวัตถุประสงค์ต่อไปนี้

- เพื่อให้เข้าใจโครงสร้างและกลไกการทำงานเบื้องต้นของระบบไฟล์ (File System) ในระบบปฏิบัติการตระกูลยูนิกซ์
- เพื่อให้รู้จักชนิดและกลไกการทำงานเบื้องต้นของอุปกรณ์เก็บรักษาข้อมูล ได้แก่ หน่วยความจำแฟลช การ์ดหน่วยความจำ SD เป็นต้น

ผู้อ่านควรย้ำทำความเข้าใจคำสั่งพื้นฐานของระบบยูนิกซ์ในการทดลองที่ 4 ภาคผนวก D ซึ่งจะช่วยเสริมความเข้าใจระบบไฟล์ของระบบลินุกซ์เพิ่มเติมในการทดลองที่ 12 ภาคผนวก L

7.1 ระบบไฟล์ (File System)

ไฟล์จะเก็บอยู่ในอุปกรณ์เก็บรักษาข้อมูลตามโครงสร้างหรือระบบไฟล์ของระบบปฏิบัติการ ผู้ใช้และนักพัฒนาสามารถใช้งานไฟล์โดยไม่เห็นความแตกต่างใด ๆ เช่น สร้างไฟล์ (Create) เขียน/บันทึก/อ่านไฟล์เปลี่ยนชื่อ (Rename) ไฟล์ ลบ (Remove/Delete) ไฟล์ สำเนา (Copy) ไฟล์ ย้ายตำแหน่ง (Move) ไฟล์ และกู้คืน (Recover) ไฟล์ เมื่ออุปกรณ์เกิดปัญหา โดยไม่จำเป็นต้องรับรู้ว่าระบบไฟล์เป็นชนิดใด เนื่องจากระบบปฏิบัติการได้ซ่อนรายละเอียดไว้ เพื่อให้ผู้ใช้และนักพัฒนาสามารถเขียนโปรแกรมได้สะดวกมากขึ้น ตำราเล่มนี้จะเน้นที่พื้นฐานของระบบไฟล์ดังเดิมของระบบยูนิกซ์ ซึ่งเป็นระบบไฟล์พื้นฐานให้ระบบอื่น ๆ ได้พัฒนาต่ออยู่ด้วย เป็นระบบบริหารจัดการไฟล์ที่สำคัญและเป็นที่นิยม ได้แก่

- ระบบ NTFS (File System) สำหรับระบบวินโดว์ส รายละเอียดเพิ่มเติมที่ ntfs.com
- ระบบ EXT4 สำหรับระบบลินุกซ์ รายละเอียดเพิ่มเติมที่ wikipedia
- ระบบ Apple File System (APFS) สำหรับระบบ MAC OS รายละเอียดเพิ่มเติมที่ wikipedia

7.1.1 การใช้งานไฟล์ (File Operations)

การเปิดไฟล์ คือ การจອງເວຼ່ອຮ່າວເມໂນຣິຈຳນວນໜຶ່ງເພື່ອທຳຫັນທີເປັນບັຟເຟອ໌ ໂປຣແກຣມເມອົບຕ້ອງເປີດໄຟລ໌ໂດຍໃຊ້ຝຶກ໌ໜັນ fopen() ໃນພາສາ C ໂດຍຮີເທີຣິນຄ່າພອຍິນເຕຸອ໌ ອີຣີ ແອດເດຣສ ເຮີມຕັ້ນຂອງບັຟເຟອ໌ ຂອງໄຟລ໌ໃຫ້ກັບໂປຣແກຣມທີ່ທຳການເປີດໄຟລ໌ ການເປີດໄຟລ໌ ບັນດາເປັນເພື່ອການອ່ານຍ່າງເດືອນ (Read Only) ເພື່ອການເຂົ້ານອ່າງເດືອນ (Write Only) ແລະເພື່ອການແລະເຂົ້ານ (Read-Write)

การอ່ານ/ເຂົ້ານຂໍ້ມູນໃນໜ່າຍ່າງມີຢູ່ໃນຮູບພາບຂອງການອ່ານເຂົ້ານໄຟລ໌ແທນ ພາສາ C/C++ ມີຝຶກ໌ໜັນ fread() ສໍາຮັບອ່ານຂໍ້ມູນໃນໄຟລ໌ ແລະຝຶກ໌ໜັນ fwrite() ສໍາຮັບເຂົ້ານຂໍ້ມູນ ແລະເມື່ອຕັ້ງການເລີກໃຊ້ການໄຟລ໌ຂໍ້ມູນໄດ້

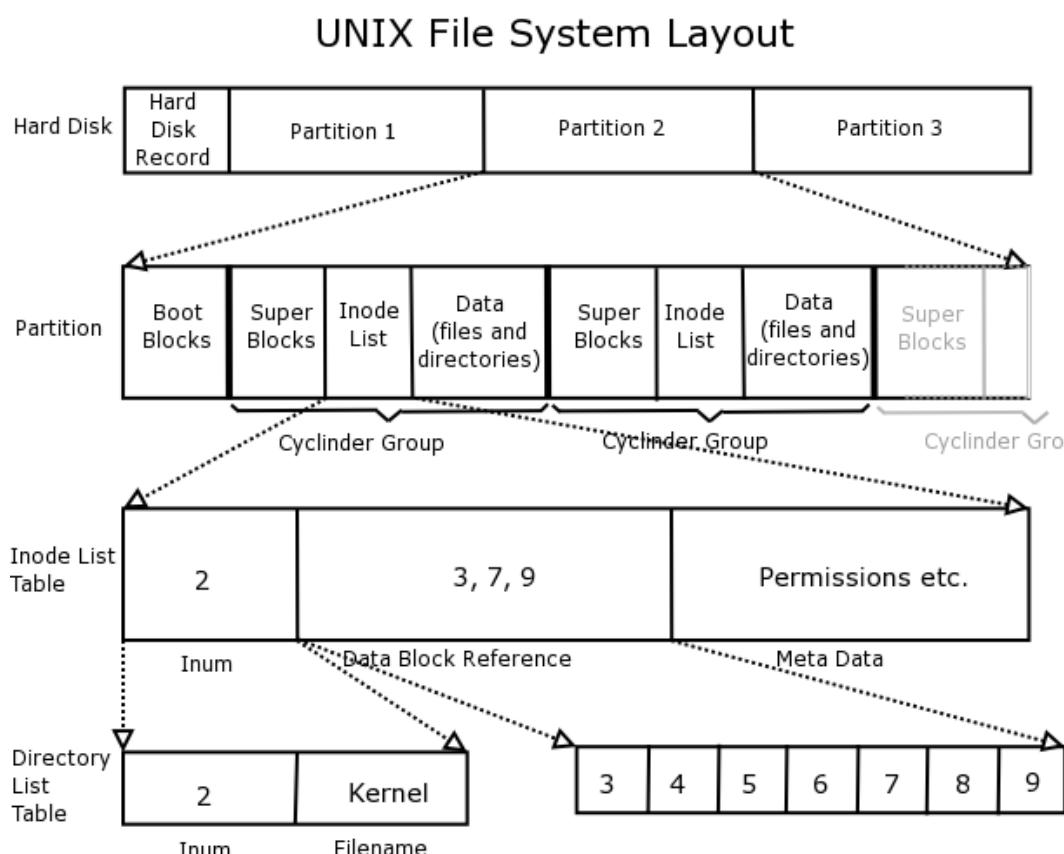
ເມື່ອໂປຣແກຣມເມອົບຕ້ອງເຮີຍໃຊ້ຝຶກ໌ໜັນ fread() ຮັບປິດຕົວການຈະທຳຫັນທີ່ອ່ານຂໍ້ມູນໃນເວຼ່ອຮ່າວເມໂນຣິ ປຣເວລນ ຊື່ຈອງໄວ້ເປັນບັຟເຟອ໌ ຕາມ ລັກການ Memory Mapped File ຊື່ຈະເຮີມຕັ້ນຫື່ນເນື່ອຂບວນການ

DMA ในหัวข้อที่ 6.13 อ่านข้อมูลจากการดูดหน่วยความจำ SD มาเขียนลงในบัฟเฟอร์ เมื่อบัฟเฟอร์มีข้อมูลไม่เพียงพอ ขบวนการ DMA จะนำข้อมูลมาเติมให้จนอ่านเจอตัวอักขระสิ้นสุดไฟล์ (End of text File: EOF) เท่ากับ $1A_{16}$ หรือ 26_{10} ในตารางรหัสแอกซ์กี ในรูปที่ 2.12 และ softwareforeducation.com

การเขียนข้อมูลเพื่อเก็บในไฟล์ จะมีพิธีทางการให้ที่ตรงข้ามกัน เมื่อโปรแกรมเมอร์เรียกใช้ฟังก์ชัน `fwrite()` ระบบปฏิบัติการทำหน้าที่เขียนข้อมูลในบัฟเฟอร์ ซึ่งจะอิงตามหลักการ **Memory Mapped File** เช่นกัน เมื่อบัฟเฟอร์เต็ม ขบวนการ DMA จะอ่านข้อมูลไปเขียนในการ์ดหน่วยความจำ SD ทำให้บัฟเฟอร์ว่างลง ระบบปฏิบัติสามารถเขียนข้อมูลเพิ่มในบริเวณบัฟเฟอร์ และเมื่อบัฟเฟอร์เต็ม ขบวนการ DMA จะเกิดขึ้นอีกจนสิ้นสุดขบวนเขียนและปิดไฟล์นั้นในที่สุด

การปิดไฟล์ คือ การคืนพื้นที่บัฟเฟอร์ให้กับระบบ โปรแกรมเมอร์จะต้องปิดไฟล์ด้วยฟังก์ชัน `fclose()` ระบบลินุกซ์ได้พัฒนาฟังก์ชันเหล่านี้ให้กับโปรแกรมเมอร์ ซึ่งจะทำหน้าที่ติดต่อกับอุปกรณ์เก็บรักษาข้อมูลผ่านวงจรต่างๆ

7.1.2 การแบ่งพาร์ทิชัน (Partition)



รูปที่ 7.1: โครงสร้างของระบบไฟล์บนอุปกรณ์เก็บรักษาข้อมูลในระบบปฏิบัติการตระกูลยูนิกซ์ ที่มา: Demblon and Spitzner (2004)

เราจะจินตนาการว่าอุปกรณ์เก็บรักษาข้อมูลในแควนสุดของรูปที่ 7.1 เช่น การ์ดหน่วยความจำ SD, โซลิดสเตทไดร์ฟ และฮาร์ดดิสก์ไดร์ฟ เหล่านี้ มีลักษณะเป็นแบบ **ข้อมูลที่มีความยาวตามขนาดความจุ**

และแบ่งແຄບອອກເປັນພື້ນທີ່ຍ່ອຍ ຈະເຮັດວຽກກ່າວ໌ພາຣີທີ່ສັນ (Partition) ຈຶ່ງອຸປະກອນຮັກຫາຂໍ້ມູນ 1 ຕ້ວສາມາຮັດແບ່ງເປັນຫລາຍພາຣີທີ່ສັນ (Partition) ຜູ້ໃຊ້ສາມາຮັດຕິດຕັ້ງຮະບບໍ່ໄຟລໍ (File System) ຂອງແຕ່ລະພາຣີທີ່ສັນໄດ້ຍ່າງອີສະຮະ

ກາຮັດແບ່ງພາຣີທີ່ສັນ ຄື່ອ ກາຮັດແບ່ງພື້ນທີ່ອຸປະກອນຮັກຫາຂໍ້ມູນອອກເປັນສ່ວນຕ່າງໆ ເພື່ອຕອບສົນອະຄວາມຕ້ອງກາຮັດທີ່ຫລາກຫລາຍ ໄດ້ແກ່ ກາຮັດຮະບບໍ່ປົກປັດກາຮັດໄດ້ຫລາຍຮະບບໍ່ກາຍໃນເຄື່ອງເດືອກກັນ ກາຮັດເກັບຂໍ້ມູນໃນພາຣີທີ່ສັນເພັະ ເປັນຕົ້ນ ຜູ້ໃຊ້ສາມາຮັດຕິດຕັ້ງຮະບບໍ່ປົກປັດກາຮັດໃນແຕ່ລະພາຣີທີ່ສັນໄດ້ໂດຍອີສະຮະຈາກກັນ ທີ່ມາ: [archlinux](#) ແລະ [datadoctor.biz](#) ໂດຍຕໍາແໜ່ງເຮັດຕົ້ນຈະເປັນພື້ນທີ່ສໍາຫຼັບເກັບ ຕາຮາງພາຣີທີ່ສັນ (Partition Table) ຈຶ່ງຕຽບກັບບຣິເວລີນ Hard Disk Record ຂອງແຄວນສຸດໃນຮູບທີ່ 7.1 ປະກອບດ້ວຍຮາຍຊື່ ແລະ ຂໍ້ມູນປະກອບຂອງແຕ່ລະພາຣີທີ່ສັນ ມາຕຽບນຸ້າຂອງຕາຮາງພາຣີທີ່ສັນທີ່ສໍາຄັນໃນທາງປົກປັດ ໄດ້ແກ່

- ມາສເຕ່ອງບຸຕ ເຮັດວຽກ (Master Boot Record: MBR) ທຳນັ້າທີ່ເກັບຮາຍຊື່ພາຣີທີ່ສັນ ຕໍາແໜ່ງເຮັດຕົ້ນ ອີເມຍເລຂ Logical Block Address (LBA) ໃນຍັງດີສົກໄວ້ໃນຕາຮາງພາຣີທີ່ສັນ (Partition Table) ພື້ນທີ່ສ່ວນໜຶ່ງຂອງມາສເຕ່ອງບຸຕ ເຮັດວຽກ ທຳນັ້າທີ່ເກັບຕາຮາງພາຣີທີ່ສັນ ຈຶ່ງຕັ້ງຢູ່ໃນເຊັກເຕົອງແຮກ ອີເມຍເລຂ 0 ຂອງຍັງດີສົກ ໂດຍໃຊ້ພື້ນທີ່ 64 ໄບຕໍ່ ຈຶ່ງຍັງດີສົກ 1 ຕ້ວສາມາຮັດແບ່ງພາຣີທີ່ສັນໄດ້ມາກທີ່ສຸດເປັນຈຳນວນ 4 ພາຣີທີ່ສັນ ຮາຍລະເອີ້ດເພີ່ມເຕີມທີ່ [wikipedia](#)
- ຕາຮາງ GPT ຕາຮາງພາຣີທີ່ສັນໃນອຸປະກອນຮັກຫາຂໍ້ມູນຂອງເຄື່ອງຄອມພິວເຕອີປ່າຈຸບັນ ຊຶ່ວ່າ Globally Unique Identification Partition Table ອີເມຍເລຂ GUIDP ອີເມຍວ່າຕາຮາງ GPT ທຳນັ້າທີ່ຄໍາລັຍກັບ MBR ແຕ່ຮອງຮັບຈຳນວນພາຣີທີ່ສັນໄດ້ມາກຈຶ່ງ ແລະ ຮອງຮັບມາຕຽບນຸ້າຂອງໄບອອສ ອີເມຍເລຂ Unified Extensible Firmware Interface: UEFI ຮາຍລະເອີ້ດເພີ່ມເຕີມເກີຍກັບ GPT ທີ່ [wikipedia](#) ຕາຮາງ GPT ສາມາຮຽນຮັບຮະບບໍ່ປົກປັດນິດ 64 ບີຕ ທຳໃຫ້ອຸປະກອນຮັກຫາຂໍ້ມູນໂດຍເພັະຍາວົດດີສົກໄດ້ຮັບທີ່ມີຄວາມຈຸເປີ່ມສູງຂຶ້ນຮະດັບເປົ້າໂປ່ຕໍ່ (Peta Byte) ອີເມຍເລຂ 10^{15} ໄບຕໍ່ ອີເມຍເລຂ 1000 ເທຣາໄປຕໍ່

7.1.3 ໂຄງສ້າງຂອງຮະບບໍ່ໄຟລໍຢູ່ນິກໍ (Unix)

ທຳນັ້າທີ່ຂອງຮະບບໍ່ໄຟລໍ ຄື່ອ ຮອງຮັບກາຮັດເກັບໄຟລໍຕ່າງໆ ບັນທຶກປະວັດກາຮັດເຂົ້າຖຶງ (Access) ໄຟລໍ ກາຮັດບິຫຼື (Permission) ກາຮັດເຂົ້າຖຶງໄຟລໍ/ໄດຣເກຫອງຕ່າງໆ ດີເລີກ ຜູ້ໃຊ້ຈຳນວນນັກ ກາຮັດໃໝ່ໄຟລໍ ກາຮັດໄຟລໍ ກາຮັດກຸ່ມືນໄຟລໍ ເປັນຕົ້ນ ຕາມທີ່ຜູ້ໃຊ້ແລະ ຜົນປະວັດປະຍຸກຕໍ່ຮອງຂອງ

ໂຄງສ້າງຂອງຮະບບໍ່ໄຟລໍໃນຮະບບໍ່ປົກປັດ ກາຮັດຕະກູບຢູ່ນິກໍມີໂຄງສ້າງ ແລະ ທຳນັ້າທີ່ຂອງຮະບບໍ່ໄຟລໍເປັນຫັນກາຈັດກາ 3 ຫັ້ນ ໂດຍເຮັດວຽກຕ່າງໆ ດັ່ງນີ້

- ຫັນຕຽກ (Logical Layer) ເປັນຫັນບຸນສຸດ ທຳນັ້າທີ່ເຂື່ອມຕ່ອງກັບໂອຟໍຕະວັດປະຍຸກຕໍ່ ໂດຍໃຊ້ຝຶກໍຫັນເປີດໄຟລໍ ປັດໄຟລໍ ອ່ານໄຟລໍ ເປົ້າໄຟລໍ ເປັນຕົ້ນ ຈຶ່ງໄຟລໍທີ່ໂອຟໍຕະວັດປະຍຸກຕໍ່ຕ້ອງກາຮັດໃໝ່ໄຟລໍ ໂດຍເນື້ອຫາໃນບໍ່ທີ່ຈະເນັ້ນທີ່ກາຮັດໃໝ່ໃນຫັນຕຽກ ແລະ ຫັນກາຍກາພ
- ຫັນເລີມ (Virtual Layer) ທຳນັ້າທີ່ເຂື່ອມຕ່ອງກັບຫັນຕຽກ ແລະ ຫັນກາຍກາພ ຮາຍລະເອີ້ດຫົ້ນອູ່ກັບວິທີກາຮັດພັນນາໂປຣແກຣມຂອງແຕ່ລະຮະບບໍ່ປົກປັດກາຮັດ

- **ชั้นกายภาพ (Physical Layer)** เป็นชั้นล่างสุด ทำหน้าที่จัดการบล็อกข้อมูลบนอุปกรณ์เก็บรักษาข้อมูลแต่ละชนิด เพื่อตอบสนองต่อการร้องขอจากชั้นเสมอ ซึ่งกลไกสำคัญคือ การบริหารบัฟเฟอร์ (Buffer) ในหน่วยความจำกายภาพ การเข้าถึงหน่วยความจำกายภาพโดยตรง (DMA) การจัดวางตำแหน่งบล็อกข้อมูลบนอุปกรณ์เหล่านั้น เพื่อประสิทธิภาพการอ่านหรือเขียนต่อเนื่อง และ การเชื่อมต่อกับดีไวซ์ไดเรอเวอร์ของอุปกรณ์เก็บรักษาข้อมูล

ผู้ใช้สามารถจัดแบ่งพาร์ทิชันตามความจุที่ต้องการเอง หรืออนุญาตให้ระบบปฏิบัติการจัดแบ่งอัตโนมัติ หลังจากนั้น ระบบปฏิบัติการจะทำการเขียนโครงสร้างของพาร์ทิชันตามที่ได้ออกแบบไว้ เรียกว่า **การฟอร์แมตต์พาร์ทิชัน (Format)** ออกเป็นพื้นที่ต่างๆ ในแควที่สองจากบนสุดของรูปที่ 7.1 โดยหลักการคือ พาร์ทิชันสามารถแบ่งเป็นบล็อกบล็อก (Boot Blocks) จำนวน 1 ชุด และไซลินเดอร์กรุ๊ป (Cylinder Group) จำนวนหนึ่งตามขนาดความจุของพาร์ทิชันนั้น โดยแต่ละไซลินเดอร์กรุ๊ปแบ่งเป็นพื้นที่ 3 ส่วนย่อยๆ ดังนี้

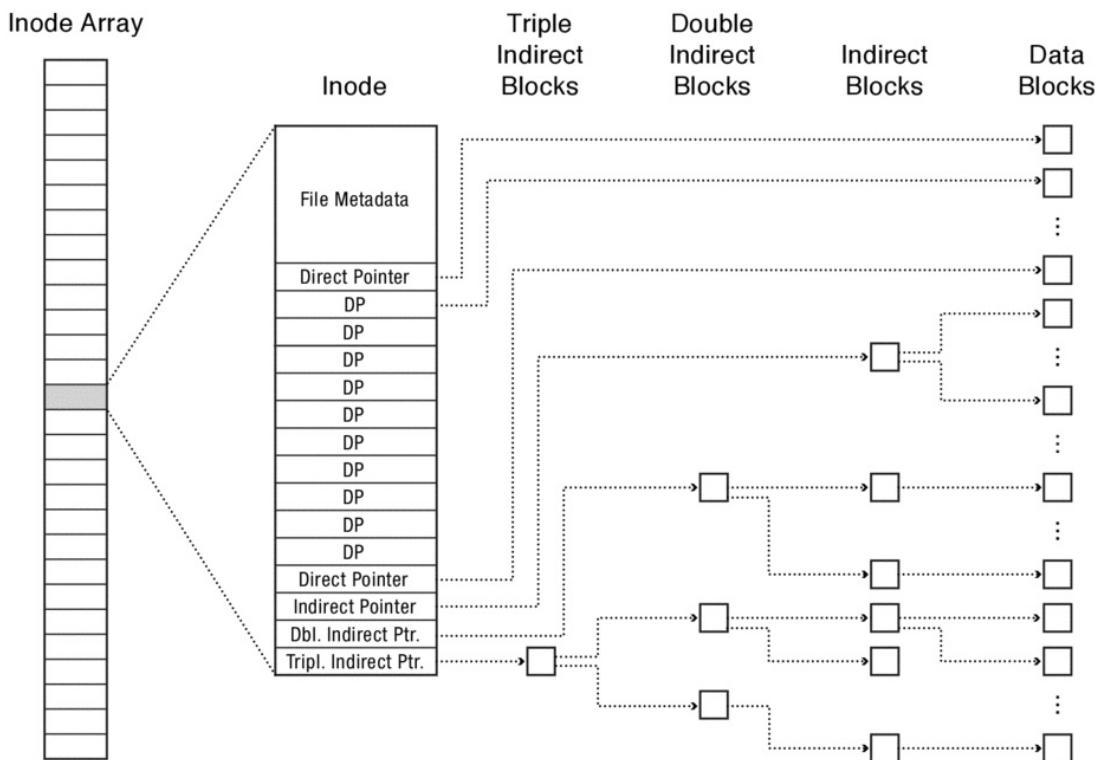
- **ชูเพอร์บล็อก (Superblock)** ทำหน้าที่เก็บรายละเอียดต่างๆ ของระบบไฟล์ ขนาดของบล็อกข้อมูล รายละเอียดการใช้งานบล็อกข้อมูลต่างๆ เช่น สถานะว่างหรือใช้งานอยู่ เป็นต้น
- **ตารางไอโหนด (Inode Table)** หรืออาจเรียกว่า ไอโหนดอาร์เรย์ (Inode Array) หรือ ไอโหนดลิสต์ (Inode List) ทำหน้าที่เก็บโครงสร้างข้อมูล Inode จำนวนหนึ่งภายใต้ไซลินเดอร์กรุ๊ปนี้ รายละเอียดเพิ่มเติมในหัวข้อถัดไป
- **บล็อกข้อมูลจำนวนหนึ่ง (Data Blocks)** ทำหน้าที่บรรจุข้อมูลของไฟล์หนึ่งไฟล์ให้เรียงต่อกันไปจนครบขนาดไฟล์ โดยทั่วไปขนาดของบล็อกข้อมูลแต่ละบล็อกมีความจุเท่ากับ 4096 ไบต์ หรือ 4 KiB (kibibyte) สำหรับระบบปฏิบัติการยูนิกซ์และอื่นๆ ทั้งนี้ขึ้นกับชนิดของเทคโนโลยีและระบบไฟล์ที่ใช้ เมื่อกำหนดขนาดความจุของแต่ละบล็อกแล้ว จำนวนบล็อกข้อมูลจะ分配ตามขนาดความจุของไซลินเดอร์กรุ๊ปนั้นๆ

7.1.4 โครงสร้างข้อมูลไอโหนด (Inode)

ในการติดตั้ง (Install) ระบบปฏิบัติการแต่ละชนิดบนอุปกรณ์เก็บรักษาข้อมูล ระบบไฟล์จะกำหนดจำนวนไอโหนดสูงสุด ตามขนาดหรือความจุของแต่ละพาร์ทิชัน ไอโหนด (Inode) คือ โครงสร้างข้อมูล (Data Structure) กลางรูปที่ 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 เป็นเลขจำนวนเต็มชนิดไม่มีเครื่องหมายความยawa 32 บิต สำหรับระบบไฟล์ Ext4 ใหม่ 32 บิต ที่มา: kernel.org



รูปที่ 7.2: โครงสร้างของไอโอนดหนึ่งตัวและการเข้าถึงแบบล็อกข้อมูลของไฟล์หนึ่งไฟล์ ที่มา: [Anderson and Dahlin \(2012\)](#)

- รายละเอียดเสริมของไฟล์ (File Metadata) แบ่งเป็น
 - ชนิดไฟล์: ไฟล์ (file), ไดเรกทอรี (directory), ไบป์ (pipe) เป็นต้น
 - * ไฟล์ ตามที่เคยนิยามที่ 3.3.1 ในหัวข้อที่ 3.3.6 ว่าเป็นการเรียงตัวกันของตัวเลขฐานสอง ทีละไบต์ ๆ โดยอาศัยพื้นที่จัดเก็บในอุปกรณ์รักษาข้อมูล เรียกว่า บล็อกข้อมูล ไฟล์เกิดจากการเรียงของบล็อกข้อมูลอย่างน้อย 1 บล็อกขึ้นไป เพื่อจัดเก็บข้อมูลหรือคำสั่งต่าง ๆ ลงในอุปกรณ์เก็บรักษาข้อมูล
 - * ไดเรกทอรี หรือ โฟลเดอร์ คือ ไฟล์ชนิดหนึ่งที่เก็บหมายเลขไอโอนดที่อยู่ภายใต้โครงสร้างนี้โดยไดเรกทอรีสามารถซ่อนกันได้ เพื่อความสะดวกในการจัดหมวดหมู่ของไฟล์และไดเรกทอรี
 - * ไบป์ (pipe) คำราเล่นนี้ไม่ครอบคลุม ผู้อ่านสามารถค้นคว้าเพิ่มเติมได้จาก [Wikipedia](#)
 - หมายเลขผู้ใช้ (User ID) ซึ่งเป็นเจ้าของ
 - หมายเลขกรุ๊ป (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) โดยบิตแต่ละบิตมีค่าล็อกจิก 1 คือ อนุญาต และค่าล็อกจิก 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
 - ขนาดที่เท่าจริงของไฟล์หรือไดเรกทอรี (หน่วยเป็นไบต์)
 - บันทึกเวลา (Time stamp): ประกอบด้วยบันทึกเวลานิดต่าง ๆ ดังนี้
 - * เวลาเข้าถึง (access time) หรือเวลาที่อ่านไฟล์หรือไดเรกทอรี
 - * เวลาที่เปลี่ยนแปลง (modification time) คือ เวลาที่ไฟล์หรือไดเรกทอรีถูกเปลี่ยนแปลง
 - * เวลาที่ไอโอนดถูกเปลี่ยนแปลง (change time)
 - อื่น ๆ
- พอยน์เตอร์เชื่อมโยงไปยังบล็อกข้อมูล (Pointers to Data Blocks) ทำหน้าที่เก็บหมายเลขบล็อกข้อมูล (บล็อกสีเหลี่ยม จัตุรัส ทางด้านขวาของรูปที่ 7.2) ซึ่งทำหน้าที่เก็บข้อมูลจริง ๆ โดยแต่ละบล็อกข้อมูลในระบบไฟล์มีขนาดเท่ากับ 4 KiB (kibibyte) ไฟล์ 1 ไฟล์ หรือไดเรกทอรี 1 ไดเรกทอรีเกิดจากการเชื่อมโยงบล็อกข้อมูลหลาย ๆ บล็อกเข้าด้วยกัน พอยน์เตอร์เชื่อมโยงแบ่งเป็น 2 ชนิดหลัก คือ
 - พอยน์เตอร์บล็อกข้อมูลทางตรง (Direct Pointers: DP) ในรูปที่ 7.2 ไอโอนดหนึ่งตัวมีจำนวน DP เท่ากับ 12 หมายเลขบล็อกข้อมูล ทำให้สามารถรองรับไฟล์ขนาดเล็กที่มีขนาดไม่เกิน 48 KiB (kibibyte)
 - พอยน์เตอร์บล็อกข้อมูลทางอ้อม ใช้สำหรับกรณีที่ไฟล์มีขนาดใหญ่กว่า 48 KiB (kibibyte) จนถึงหลายกิกะไบต์ (GiB) ระบบไฟล์สามารถเก็บข้อมูลไฟล์โดยใช้จำนวนบล็อกข้อมูลมากขึ้นตามขนาดไฟล์ พอยน์เตอร์บล็อกข้อมูลทางอ้อม จะทำหน้าที่เก็บหมายเลขบล็อกข้อมูลจำนวนมาก ๆ เพื่อรองรับไฟล์ที่มีขนาดใหญ่ขึ้นดังกล่าว พอยน์เตอร์บล็อกข้อมูลทางอ้อมแบ่งเป็น 3 ระดับ ดังนี้
 - * พอยน์เตอร์หนึ่งชั้น (Indirect Pointer) เมื่อไฟล์มีขนาดใหญ่ขึ้นอีกจนเกิน 48 KiB ทำให้จำนวนพอยน์เตอร์ทางตรงไม่เพียงพอ ระบบไฟล์จะอาศัยพอยน์เตอร์หนึ่งชั้น (Indirect Pointer) ในโครงสร้างไอโอนด เพื่อเก็บหมายเลขบล็อกพิเศษ เรียกว่า บล็อกทางอ้อมหนึ่งชั้น (Indirect Block) ขนาด 4096 ไบต์ บล็อกทางอ้อมหนึ่งชั้นทำหน้าที่เก็บหมายเลขบล็อกข้อมูลที่ $13 = 12 + 1$ จนถึงหมายเลขบล็อกที่ $1036 = 12 + 1024$ โปรดสังเกตบล็อกสีเหลี่ยมจัตุรัสที่ชั้นระหว่างช่องพอยน์เตอร์หนึ่งชั้น (Indirect Pointer) และบล็อกข้อมูลทางขวาของรูปที่ 7.2 หมายเหตุ สมมติว่าหมายเลขบล็อก 1 หมายเลขต้องการพื้นที่ขนาด 4 ไบต์

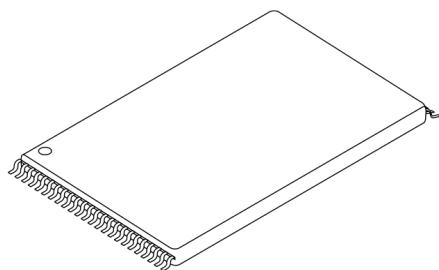
- * พอยน์เตอร์สองชั้น (Double Indirect Pointer) เมื่อไฟล์มีขนาดใหญ่ขึ้นอีก จนพอยน์เตอร์หนึ่งชั้นไม่เพียงพอก ระบบไฟล์จะอาศัยพอยน์เตอร์สองชั้นในโครงสร้างไอโหนด เพื่อเก็บหมายเลขบล็อกพิเศษ เรียกว่า **บล็อกทางอ้อมสองชั้น** (Double Indirect Blocks) ขนาด 4096 ไบต์ บล็อกทางอ้อมสองชั้นทำหน้าที่เก็บหมายเลขบล็อกทางอ้อมหนึ่งชั้น เพื่อให้บล็อกทางอ้อมหนึ่งชั้นทำหน้าที่เก็บหมายเลขหรือพอยน์เตอร์บล็อกข้อมูลที่ $1037 = 12 + 1024 + 1$ เป็นต้นไป โปรดสังเกตบล็อกสี่เหลี่ยมจัตุรัสที่อยู่ในคอลัมน์ Double Indirect Blocks ของรูปที่ 7.2
- * พอยน์เตอร์สามชั้น (Triple Indirect Pointer) เมื่อไฟล์มีขนาดใหญ่ขึ้นอีก จนพอยน์เตอร์สองชั้นไม่เพียงพอก ระบบไฟล์จะอาศัยพอยน์เตอร์สามชั้นในโครงสร้างไอโหนด เพื่อเก็บหมายเลขบล็อกพิเศษจำนวนหนึ่ง เรียกว่า **บล็อกทางอ้อมสามชั้น** (Triple Indirect Blocks) ทำหน้าที่เก็บหมายเลขบล็อกทางอ้อมสองชั้น โปรดสังเกตบล็อกสี่เหลี่ยมจัตุรัสที่อยู่ในคอลัมน์ Triple Indirect Blocks ของรูปที่ 7.2

ตัวอย่างเช่น ไฟล์ซึ่อ Kernel ตำแหน่งสองแควล่างสุดของรูปที่ 7.1 ตรงกับไอโหนดหมายเลข (Inum) = 2 และเป็นหมายเลขอ้างอิงที่ไม่ซ้ำกัน ซึ่งข้อมูลภายในไฟล์นี้จะบันทึกอยู่ในบล็อกข้อมูลจำนวน 3 บล็อก เป็นลิงก์ทางตรง (DP: Direct Pointer) ไปยังบล็อกข้อมูลหมายเลข 3, 7, 9 ส่วนซึ่อไฟล์ Kernel จะเก็บบันทึกในไดเรกทอรีที่ไฟล์นี้ตั้งอยู่ ซึ่งใช้หมายเลขไอโหนด (Inum) = 2 เป็นหมายเลขอ้างอิงที่ได้กล่าวไว้ก่อนหน้า ส่วนรายละเอียดเพิ่มเติมสามารถค้นจากส่วนที่เรียกว่า **รายละเอียดเสริมของไฟล์** ภายใต้โครงสร้างของไอโหนดในรูปที่ 7.2

หัวข้อถัดไปจะอธิบายการทำงานของเทคโนโลยีหน่วยความจำชนิดต่าง ๆ โดยเริ่มจากหน่วยความจำแฟลช การ์ดหน่วยความจำ SD โซลิดสเตทไดรฟ์ และhybridไดรฟ์ ตามลำดับ

7.2 ชิปหน่วยความจำแฟลช (Flash Memory)

หน่วยความจำแฟลช เดิมเรียกว่า **แฟลชรอม** (Flash ROM) คำว่า ROM ย่อมาจากคำว่า Read-Only Memory วัตถุประสงค์เดิมของแฟลชรอม คือ ทำหน้าที่เป็นหน่วยความจำที่สร้างจากเทคโนโลยีสารกึ่งตัวนำสำหรับใช้เก็บคำสั่งและข้อมูลเพื่อการอ่านเป็นหลัก นักพัฒนาระบบจึงประยุกต์ใช้แฟลชรอมสำหรับเก็บคำสั่งประจำเครื่อง หรือ **เฟิร์มแวร์** (Firmware) หรือ **ไบอส** (BIOS: Basic Input Output System) มาจนถึงปัจจุบัน หน่วยความจำแฟลชจึงถูกพัฒนาอย่างต่อเนื่องตามความนิยม จนสามารถเขียนหรือแก้ไขข้อมูลภายในแฟลชรอมได้รวดเร็วขึ้น และกลไก เป็นอุปกรณ์เก็บรักษาข้อมูลในปัจจุบัน เพราะแฟลชรอม หรือหน่วยความจำแฟลชนี้ จุดเด่นหลายด้าน ได้แก่ ส่วนประกอบการทำงานน้อยขึ้นกว่า น้ำหนักเบากว่า ปริมาตรที่เล็กกว่า ความเร็วในการอ่านหรือเขียนข้อมูลได้รวดเร็วกว่า ความจุที่เพิ่มมากขึ้น ใช้กำลังไฟและพลังงานน้อยกว่า อุปกรณ์เก็บรักษาชนิดอื่น ๆ และสามารถเก็บข้อมูลได้ยาวนานขึ้น



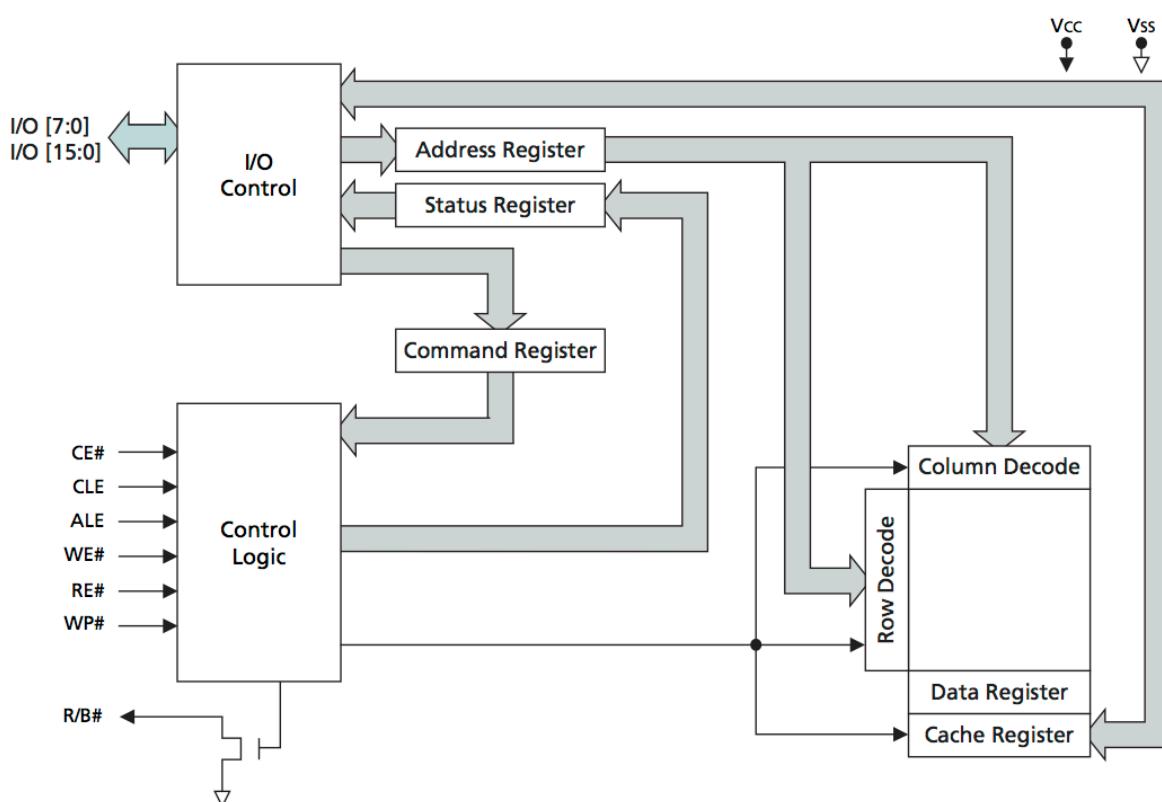
รูปที่ 7.3: ชิปหน่วยความจำแฟลชนี้ เป็น หน่วยความจำแฟลชนิด NAND ผลิตโดยบริษัท Micron Technology โดยใช้ตัวถังชนิด TSOP (Thin Small Outline Package) จำนวน 48 ขา ที่มา: [Micron Technology, Inc. \(2004\)](#)

แฟลชรอมกรณีศึกษาของตราเล่มนี้ เป็น หน่วยความจำแฟลชนิด NAND ผลิตโดยบริษัท Micron Technology ในปี ค.ศ. 2004 ใช้ตัวถังชนิด TSOP (Thin Small Outline Package) ในรูปที่ 7.3 ตัวชิป มีการผลิตออกแบบด้วยความจุที่แตกต่างกัน ขึ้นอยู่กับ จำนวนสาย จำนวนบล็อกข้อมูล และจำนวนสายต่อชิป โดยความจุข้อมูลต่อ 1 สายเริ่มต้นที่ ขนาด 2 กิบิบิต หรือ 2×2^{30} บิต ซึ่งประกอบด้วยบล็อก (Block) ข้อมูลจำนวน 2048 (2×2^{10}) บล็อก แต่ละบล็อกประกอบด้วย 64 (2^6) เพจ แต่ละเพจ (Page) มีความจุ 2048 (2×2^{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

หน่วยความจำแฟลชนี้จะมีประสิทธิภาพโดยวัดจากเวลาเข้าถึง (Access Time) และความจุต่อชิปเพิ่มสูงขึ้นเรื่อย ๆ เมื่อเทคโนโลยีพัฒนาไปเรื่อย ๆ โดยทั่วไปประสิทธิภาพการอ่านข้อมูลจะดีกว่าการเขียน ประสิทธิภาพการเขียนจะดีกว่าการลบตามลำดับ โดยอายุการใช้งานซึ่งนับจากจำนวนการลบข้อมูล (Erase) ดังนี้

- ประสิทธิภาพการอ่านข้อมูลขึ้นอยู่กับตำแหน่ง และรูปแบบการอ่าน และรูปแบบการเรียงตัวของข้อมูล เช่น การอ่านข้อมูลที่เรียงตัวต่อเนื่อง (Sequential Address) จะใช้เวลาเข้าถึง สั้นมากเพียง 30 นาโนวินาที การอ่านข้อมูลที่ไม่เรียงตัวต่อเนื่องและสุ่มตำแหน่ง (Random Address) จะใช้เวลาเข้าถึงนานขึ้นเป็น 25 ไมโครวินาที จะเห็นได้ว่าใช้เวลาเพิ่มขึ้นเกือบ 1,000 เท่า เทียบกับการอ่านข้อมูลแบบเรียงตัวต่อเนื่อง

- ประสิทธิภาพการเขียนข้อมูลมีลักษณะ เช่นเดียวกับ การอ่านข้อมูล โดย การเขียนข้อมูลต้องเขียนครั้งละเพจ (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 แต่ชิปสามารถรักษาข้อมูลได้เป็นระยะเวลายาวนานถึง 10 ปีตามที่แจ้งไว้ในเอกสารคุณสมบัติ



รูปที่ 7.4: โครงสร้างภายในชิปหน่วยความจำแฟลช NAND ที่มา: [Micron Technology, Inc. \(2004\)](#)

รูปที่ 7.4 แสดงโครงสร้างภายในหน่วยความจำแฟลช NAND ประกอบด้วย

- อาร์เรย์ของเซลล์หน่วยความจำทางด้านขวาล่าง
- วงจรควบคุม (Control Logic) การทำงานโดยรวม และ
- วงจรควบคุมอินพุตและเอาต์พุต (I/O Control) เพื่อเชื่อมต่อกับวงจรภายนอกตัวชิป

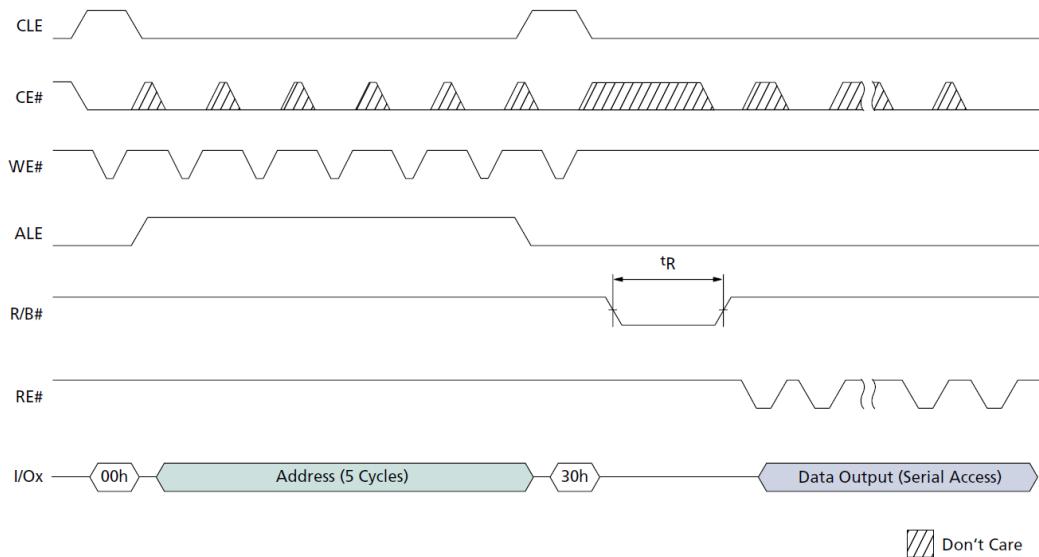
โดยตัวชิปหน่วยความจำแฟลชมีขาสัญญาณควบคุม ขาสัญญาณแอดเดรสและขาสัญญาณข้อมูล ดังนี้

- 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 R/\overline{B} คือ สถานะ Ready หากมีค่าเท่ากับลอจิก 1 หรือ Busy bar หากมีค่าเท่ากับลอจิก 0
- ขาสัญญาณ **I/O[0:15]** ทั้งหมด 16 ขา มีหน้าที่มัลติเพล็กซ์ (Multiplex) หรือใช้งานร่วมกันคนละช่วงเวลา เพื่อการรับและเดรส รับ/ส่งข้อมูล ส่งสถานะ และรับคำสั่งไปยังตัวชิป
 - **สัญญาณ แอดเดรส** จะถูกเก็บพักในรีจิสเตอร์ แอดเดรส (Address Register) เพื่อนำไปผ่านชุด_Decode_ โดยวงจร Column Decoder เพื่อสร้างสัญญาณบิตไลน์ (Bit Line) และ Row Decoder เพื่อสร้างสัญญาณเวิร์ดไลน์ (Word Line) ในรูปที่ 7.1
 - **สัญญาณข้อมูลที่อ่านหรือเขียน** จะถูกเก็บพักในรีจิสเตอร์แคช (Cache Register) ชั่วคราวแล้วจึงย้ายเข้าหรือออกจากรีจิสเตอร์ข้อมูล (Data Register)
 - **สถานะของการทำงานของแฟลช** จะเก็บพักในรีจิสเตอร์สถานะ (Status Register) เพื่อให้วงจรเชื่อมต่อ (Interface Circuit) รับทราบได้แก่ 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 และวงจรควบคุม การอ่านหรือเขียนข้อมูลลงไปในอาร์เรย์นี้ การเข้าถึงเซลล์หน่วยความจำอาศัยแอดเดรส แท่ง (列址) และแอดเดรสคอลัมน์ (เลขคอลัมน์) เพื่อบ่งชี้ตำแหน่งเซลล์นั้น ๆ ในอาร์เรย์ แต่การทำงานของหน่วยความจำแฟลชมีความซับซ้อน เนื่องจากมีการอ่านหลายชนิด การเขียนหลายชนิด และการลบข้อมูลตามที่กล่าวไปก่อนหน้า ที่มา: [Micron Technology, Inc. \(2004\)](#) ตารางเล่มนี้จะอธิบายการอ่านข้อมูลแบบเพจ และการเขียนข้อมูลแบบเพจพอสั่งเข้าไป เนื่องจากเป็นการใช้งานหน่วยความจำแฟลชที่เกิดขึ้นบ่อยที่สุด ลำดับสุดท้ายจะเป็นการเปรียบเทียบที่หน่วยความจำแฟลชชนิดอื่น ๆ ที่สำคัญ

7.2.1 การอ่านข้อมูลแบบเพจ (Page Read)



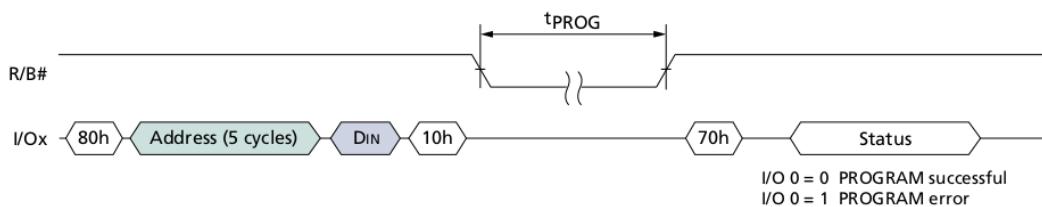
รูปที่ 7.5: ไดอะแกรมเวลาของการอ่านข้อมูลแบบเพจ (Page Read) ของหน่วยความจำแฟลช NAND
ที่มา: Micron Technology, Inc. (2004)

การอ่านข้อมูลทั้งเพจ (Page Read) ซึ่งมีประสิทธิภาพสูงกว่าการอ่านแบบสุ่ม (Random Read) เนื่องจากต้นโดยการเปลี่ยนสัญญาณเหล่านี้ แกนเวลาอยู่ในแนวนอนในรูปที่ 7.5 โดยแกนเวลาเริ่มต้นจากทางซ้ายไปทางขวา

- **สัญญาณ CLE** เปลี่ยนจาก 0 เป็น 1 เพื่ออ่านค่าคำสั่งที่ปรากฏบนบัส I/O โดยคำสั่งจะมีรูปแบบคำสั่ง 00h-Address (5 Cycles)-30h คือ การอ่านข้อมูลทั้งเพจ จะ แสดง adressthat ได้รับ
- **สัญญาณ CE#** หรือ \overline{CE} เปลี่ยนจาก 1 เป็น 0 เพื่อส่งซิปให้เริ่มต้นทำงาน
- **สัญญาณ WE#** หรือ \overline{WE} จะมีการเปลี่ยนแปลงจาก 1 เป็น 0 ลักษณะเดียวกันในรูปที่ 7.5 เพื่อเขียนคำสั่ง และแสดง adressthat ให้แก่พิกัดในรีจิสเตอร์ต่าง ๆ
- **สัญญาณ ALE** จะเปลี่ยนแปลงจาก 0 เป็น 1 ในระหว่างที่บัส I/O รับ adressthat เป็นระยะเวลา 5 คลาบเวลา (Cycles)
- **สัญญาณ R/B#** หรือ $\overline{R/B}$ จะเปลี่ยนแปลงจาก 1 เป็น 0 ระยะเวลาหนึ่ง (t_R) เพื่อบ่งบอกว่าซิป มีสถานะยุ่ง (Busy) แล้วกลับไปเป็น 1 เพื่อบ่งบอกว่าซิปพร้อม (Ready) แล้ว
- **สัญญาณ RE#** หรือ \overline{RE} จะเปลี่ยนแปลงจาก 1 เป็น 0 ลักษณะเดียวกันในรูปที่ 7.5 เมื่อตรวจสอบว่าสัญญาณ R/B# เปลี่ยนจาก 0 เป็น 1 เพื่ออ่านข้อมูลทางขา I/O โดยใช้อบเชิงหรืออบขาลงของสัญญาณ RE#

- สัญญาณ I/O[0:15] ทั้งหมด 16 ขา จะเปลี่ยนแปลงตามลำดับดังนี้
 - คำสั่ง 00h (00_{16}) จำนวน 1 ค疤เวลา
 - แอดเดรส จำนวน 5 ค疤เวลา
 - คำสั่ง 30h (30_{16}) จำนวน 1 ค疤痕เวลา
 - ข้อมูลที่ตั้งกับแอดเดรสจะปรากฏขึ้น เมื่อรอเป็นระยะเวลา t_R จำนวนหลายค疤เวลาตามขนาดของเพจ ข้อมูล โดยการอ่านข้อมูลเรียงตามลำดับหมายเลขแอดเดรส (Serial Access)

7.2.2 การเขียนข้อมูล (Program Data)



รูปที่ 7.6: ໄດ້ອະແກມເວລາຂອງ ການເຂີຍນີ້ຂໍ້ມູນ (Program Data) ແລະ ອ່ານສຕານະ (Read Status) ຂອງ ຜັງວ່າຍຄວາມຈຳແພລະ NAND ທີ່ມາ: [Micron Technology, Inc. \(2004\)](#)

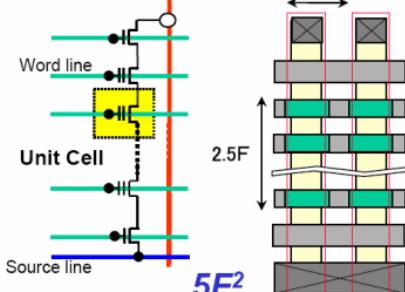
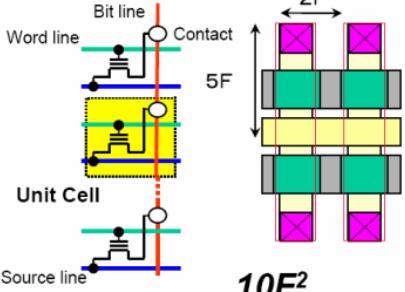
ການເຂີຍນີ້ຂໍ້ມູນຂາດເລື້ກ (Program Data) ແລະ ອ່ານສຕານະ (Read Status) ເຮັດວຽກໂດຍການເປີ່ຍນ ສັງຄູນເຫຼຸ່ນ ແລ້ວ ແກນເວລາອູ້່ໃນແນວນອນໃນຮູບທີ່ 7.6 ໂດຍແກນເວລາເຮັດວຽກຈາກທາງໜ້າຢ່າງໄປທາງຂວາ

- สัญญาณ CLE เปลี่ยนจาก 0 เป็น 1 เพื่ອอ่านคำสั่งที่ปรากฏบนบัส I/O โดยคำสั่งจะมีรูปแบบ คำสั่ง 80h-Address (5 Cycles)-D_{IN}-10h គິ່ອ ການເຂີຍນີ້ຂໍ້ມູນຂາດເລື້ກ ດັ່ງ ແລະ ແກນເວລາທີ່ຕ້ອງການ
- สัญญาณ CE# ຢ້ອງ \overline{CE} ເປີ່ຍນຈາກ 1 ເປັນ 0 ເພື່ອສັ່ງຈີປໍໃຫ້ເຮັດວຽກ
- สัญญาณ WE# ຢ້ອງ \overline{WE} ຈະມີການເປີ່ຍນແປລງຈາກ 1 ເປັນ 0 ສລັບໄປນາ ເພື່ອເຂີຍນີ້ຂໍ້ມູນ ແລະ ແກນເວລາທາງໆ I/O ໄປເກີບພັກໃນຮີຈິສເຕວົວຕ່າງໆ
- สัญญาณ ALE ຈະເປີ່ຍນແປລງຈາກ 0 ເປັນ 1 ໃນຮ່ວ່າງທີ່ບັສ I/O ຮັບ ແລະ ແກນເວລາເປີ່ຍນ 5 ຄະເວລາ (Cycles) ດ້ວຍກັບການອ່ານຂໍ້ມູນ
- สัญญาณ I/O[0:15] ທັງໝົດ 16 ຂາ ຈະເປີ່ຍນແປລງຕາມລຳດັບດັ່ງນີ້
 - คำสั่ง 80h (80_{16}) จำนวน 1 ค疤痕เวลา
 - ແກນເວລາ 5 ค疤痕เวลา
 - ข່ອມູນທີ່ຕ້ອງການເຂີຍ (D_{IN}) จำนวน 1 ค疤痕เวลา
 - คำสั่ง 10h (10_{16}) จำนวน 1 ค疤痕เวลา

- สัญญาณ R/B# หรือ R/\overline{B} เปลี่ยนจาก 1 เป็น 0 และค้างไว้เป็นระยะเวลาหนึ่ง แล้วจึงเปลี่ยนจาก 0 เป็น 1 เพื่อบ่งบอกว่าชิป มีสถานะยุ่ง (Busy) และกลับไปเป็นสถานะพร้อม (Ready=1) เพื่อบ่งบอกว่าสิ้นสุดการเขียนข้อมูลและพร้อมจะรับคำสั่งต่อไป
- คำสั่ง 70h (70_{16}) จำนวน 1 ควบเวลา เพื่ออ่านสถานะของการเขียน
- สัญญาณ Status จากวจ器 Status Register เพื่อให้เช็คพื้นที่ 0 หากมีค่าเท่ากับ
 - * 1 แสดงว่าการเขียนสำเร็จ
 - * 0 แสดงว่าการเขียนยังไม่สำเร็จ

7.2.3 หน่วยความจำแฟลชชนิดอื่น ๆ

ตารางที่ 7.1: ตารางเปรียบเทียบ เซลล์หน่วยความจำแฟลชชนิด NAND (ซ้าย) และ NOR (ขวา) ตามโครงสร้างและผังการจัดวางของเซลล์หน่วยความจำ ที่มา: Choi (2010) หมายเหตุ F คือ ความกว้างของไฟลต์เกต (Float Gate) มีหน่วยเป็น นาโนเมตร

	NAND	NOR
Cell Array & Size	 Word line	 Bit line
Cross-section		
Features	Small Cell Size, High Density Low Power & Good Endurance → Mass Storage	Large Cell Current, Fast Random Access → Code Storage

ตารางที่ 7.1 แสดงการเปรียบเทียบเซลล์หน่วยความจำแฟลชชนิด NAND (ซ้าย) และ NOR (ขวา) เชิงโครงสร้าง ภาพแนวตัดขวาง (Cross Section) ผังการจัดวางหรือเลย์เอาต์ (Lay Out) และขนาดของเซลล์ (Cell Size) ผู้อ่านจะเห็นได้ว่าอาร์เรย์ของเซลล์หน่วยความจำ (Cell Array) มีโครงสร้างการเชื่อมต่อของเซลล์ต่าง ๆ เข้าด้วยกัน โดยใช้จุดตัดกันของสายบิตไลน์ (Bit line) และสายเวิร์ดไลน์ (Word line) ระบุตำแหน่งของเซลล์ที่ต้องการอ่านหรือเขียนคล้ายกับหน่วยความจำ SRAM และ SDRAM ในรูปที่ 5.13 และรูปที่ 5.17 ตามลำดับ

การเชื่อมต่อของเซลล์หน่วยความจำมีลักษณะที่แตกต่างกัน คือ

- แฟลชชนิด NAND ต่อเซลล์เข้าด้วยกันแบบอนุกรม คล้ายกับวงจรเกท NAND ทำให้ประหยัดพื้นที่บนแผ่นซิลิกอนเท่ากับ $5F^2$ ต่อหนึ่งเซลล์ จึงทำให้ต้นทุนในการผลิตต่ำกว่าแฟลชชนิด AND และ NOR แต่ต้องใช้เวลาอ่านหรือเขียนหลายเซลล์ตามลำดับ
- แฟลชชนิด NOR ต่อเซลล์เข้าด้วยแบบขนาน คล้ายกับวงจรเกท NOR โดยใช้สายเวิร์ดไลน์แยกกัน การต่อเซลล์แบบขนาน สามารถอ่านและเขียนข้อมูลแต่ละเซลล์ได้独立 ทีละเซลล์ แต่ใช้พื้นที่บนแผ่นซิลิกอนเพิ่มเป็น $10F^2$ ต่อเซลล์ หรือเป็น 2 เท่าของแฟลช NAND

ขนาดของพื้นที่ผังการจัดวางหรือเลเยอร์ (Layout) และภาพตัดขวาง (Cross Section) ของแต่ละเซลล์สามารถคำนวณตามความกว้างคูณความยาวของแต่ละเซลล์ โดย F คือ ความกว้างของขาไฟลต์เกต (Float Gate) ของทรานซิสเตอร์ ที่สามารถผลิตได้ซึ่งมีขนาดเล็กลงเรื่อยๆ จาก 32 นาโนเมตรเป็น 12 นาโนเมตร ที่มา: [wikipedia](#)

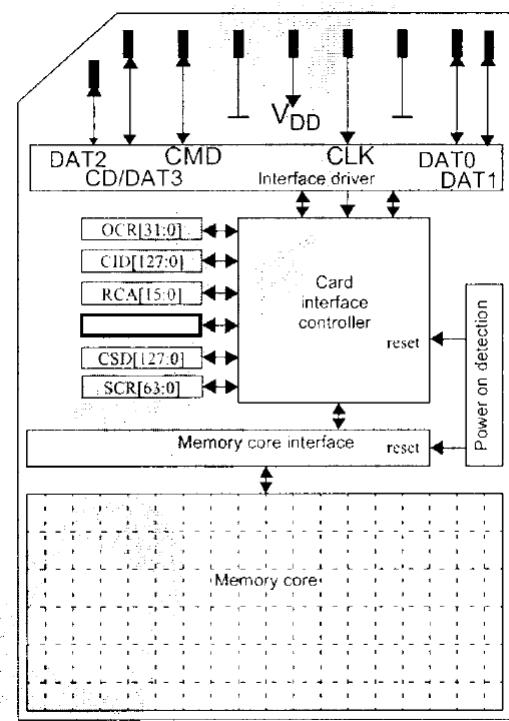
โครงสร้างภายในของแต่ละเซลล์ในแฟลชรอมได้ถูกพัฒนาให้เป็นเซลล์ชนิด TLC (Triple Level Cell) และ QLC (Quad Level Cell) ในปัจจุบัน เพื่อเพิ่มความจุหรือจำนวนบิตต่อเซลล์ให้มากขึ้น โดยโครงสร้างชนิด TLC และ QLC สามารถเพิ่มความจุเป็นสามเท่า (Triple) และสี่เท่า (Quad) โดยสามารถเก็บข้อมูลได้จำนวน 3 และ 4 บิต ต่อ 1 เซลล์ ข้อมูล ตาม ลำดับ ผู้อ่านสามารถค้นคว้ารายละเอียดเพิ่มเติมได้ที่ [embedded-computing.com](#) และ [wikipedia](#)

7.3 การ์ดหน่วยความจำ SD (Secure Digital)

เนื้อหาในหัวข้อนี้ จะเสริมเพิ่มเติมหัวข้อที่ 3.1.4 ให้ละเอียดมากขึ้น เนื่องจากโครงสร้างภายในของ การ์ดหน่วยความจำ SD เป็นหน่วยความจำแฟลชชนิด NAND เป็นหลัก โครงสร้างภายในของ การ์ดหน่วยความจำ SD ในรูปที่ 3.8 ประกอบด้วย ชิปหน่วยความจำแฟลชเป็นองค์ประกอบสำคัญ ที่มา: [wikipedia](#) การ์ดหน่วยความจำจะทำงานแบบซิงโครนัส (Synchronous) ตามสัญญาณคล็อก (CLK) ที่วงจรควบคุม ส่งมา ซึ่งมักจะมีความถี่สูงสุดเป็นจำนวนเท่าของ 25 เมกะเฮิรตซ์ เช่น 50 เมกะเฮิรตซ์ 100 เมกะเฮิรตซ์ เป็นต้น

โครงสร้างภายในการ์ดหน่วยความจำ SD ในรูปที่ 7.7 ประกอบด้วย

- อาร์เรย์เซลล์หน่วยความจำชนิดแฟลช ตามขนาดของความจุที่ต้องการใช้
- วงจรเชื่อมต่อ กับแกนหน่วยความจำ (Memory Core Interface) เพื่อควบคุมการทำงานหน่วยความจำแฟลชที่อยู่ภายนอกการ์ด
- วงจรควบคุม (Card Interface Controller) เพื่อเชื่อมต่อ กับโซล์ฟต์แวร์ โดยทำงานร่วมกับรีจิสเตอร์ต่างๆ เหล่านี้
 - รีจิสเตอร์ CID[27:0] (Card Identification) ขนาด 28 บิต เพื่อกีบหมายเลขประจำตัวการ์ด
 - รีจิสเตอร์ OCR[31:0] (Operation Condition Register) ขนาด 32 บิต เพื่อกีบสถานะการทำงานของ การ์ด



รูปที่ 7.7: โครงสร้างภายในการ์ดหน่วยความจำ SD ที่มา: SanDisk Corporation (2003)

- รีจิสเตอร์ **RCA[15:0]** (Relative Card Address) ขนาด 32 บิต เพื่อเก็บหมายเลขแอดเดรสของкарดซึ่งจะเปลี่ยนแปลงระหว่างที่ถอดเข้าออกจากระบบ
 - รีจิสเตอร์ **CSD[27:0]** (Card Specific Data) ขนาด 28 บิต เพื่อเก็บข้อมูลจำเพาะของкарด
 - รีจิสเตอร์ **SCR[63:0]** (SD Configuration Register) ขนาด 64 บิต เพื่อเก็บการตั้งค่าพิเศษประจำตัวการ์ด

การเขื่อมต่อ CPU กับการ์ดหน่วยความจำ SD นี้สามารถเลือกได้โดยใช้การเขื่อมต่อโหมด SPI และโหมด SDIO การเขื่อมต่อกับการ์ดโหมด SDIO สามารถทำได้อีกสองโหมดย่อย คือ การเขื่อมต่อโหมด SD 1 บิต และ การเขื่อมต่อโหมด SD 4 บิต ซึ่งมีประสิทธิภาพเร็วที่สุดเนื่องจากสามารถอ่าน/เขียนได้พร้อมกัน 4 บิต ซึ่งมีรายละเอียดของขาต่างๆ ในตารางที่ [7.2](#) ดังนี้

ซึ่งพิจัย จะควบคุมการทำงานของ การ์ด หน่วยความจำ SD โดยส่งคำสั่งต่าง ๆ ผ่านทางขา CMD ซึ่งประกอบด้วยโท เก็บคำสั่ง (CMD: Command) สำคัญ ๆ ในโหมด SD ผ่านขา CMD ของ การ์ด ดังต่อไปนี้

- CMD12: Stop (หยุดการอ่านหรือเขียนข้อมูล)
 - CMD17: Read Single Block (อ่านข้อมูลจำนวน 1 บล็อก)
 - CMD18: Read Multiple Block (อ่านข้อมูลจำนวนหลายบล็อก)

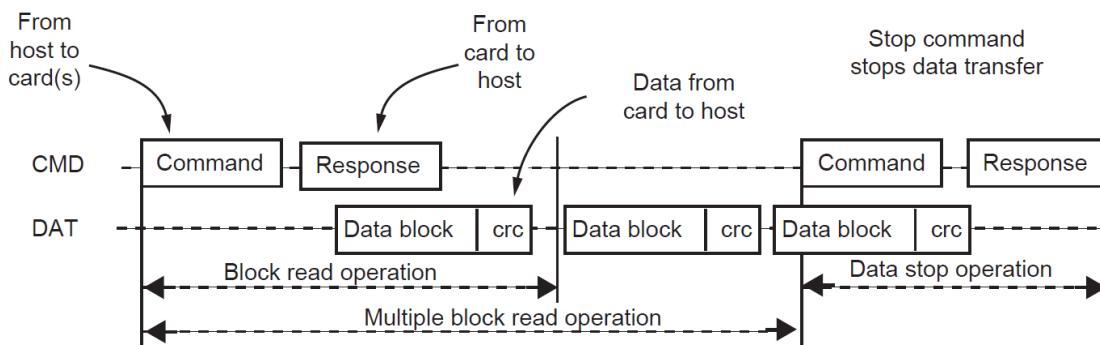
ตารางที่ 7.2: หมายเลขชื่อ และวัตถุประสงค์ของสัญญาณในเมモรี SD 4 บิต ที่มา: wikipedia.org

ขา	ชื่อ	วัตถุประสงค์
1	DAT2	Data บิตที่ 2
2	DAT3/CD	Data บิตที่ 3 หรือ Card Detect
3	CMD	ขารับคำสั่ง (Command)
4	VDD	แหล่งจ่ายไฟข้อมูล
5	CLK	สัญญาณคล็อก
6	VSS	แหล่งจ่ายไฟกราวน์ด
7	DAT0	Data บิตที่ 0
8	DAT1	Data บิตที่ 1

- CMD24: Write Single Block (เขียนข้อมูลจำนวน 1 บล็อก)
- CMD25: Write Multiple Block (เขียนข้อมูลจำนวนหลายบล็อก)
- CMD32: Erase Block Start (หมายเลขอร่องเริ่มต้นสำหรับการลบข้อมูล)
- CMD33: Erase Block End (หมายเลขอร่องสิ้นสุดสำหรับการลบข้อมูล)
- CMD38: Erase (ทำการลบข้อมูล)

7.3.1 การอ่านข้อมูล

การอ่านข้อมูลจากการ์ดหน่วยความจำ SD อยู่ในรูปของบล็อกข้อมูลที่เรียงตัวตามระบบไฟล์ ระบบไฟล์ที่นิยมในการ์ดหน่วยความจำ SD ชื่อ ระบบ FAT32 กระบวนการเปิดไฟล์ อ่านไฟล์ และปิดไฟล์ จะอาศัยคำสั่งระดับล่าง คือ トイเค็น CMOS ตามที่ได้กล่าวไปก่อนหน้าสำหรับดำเนินการในระดับชั้นภายนอก ในหัวข้อที่ 7.1.3



รูปที่ 7.8: ไอดีอะแกรมเวลาของการอ่านข้อมูลจำนวนหลายบล็อกจากการ์ดหน่วยความจำ SD ที่มา: SanDisk Corporation (2003)

การอ่านข้อมูลจากการ์ดหน่วยความจำ SD แบ่งเป็น การอ่านข้อมูลจำนวน 1 บล็อก และ การอ่านจำนวนหลายบล็อก การทำงานของทั้งสองวิธีคล้ายกันมาก ดังนี้ โอดิสท์ส์トイเค็นคำสั่ง **CMD17** ไปยังการ์ด

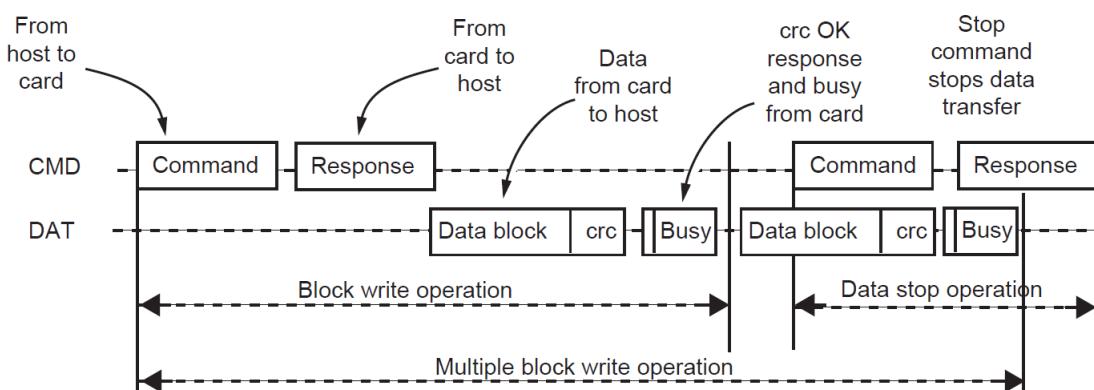
หน่วยความจำ การดูแลความจำ จึงตอบกลับด้วยโทเค็น Response การอ่าน 1 บล็อกจะสิ้นสุดการทำงานแค่นี้โดยไม่ต้องมีการโต้ตอบระหว่างไฮสท์และการดึง

แต่ถ้าส่งโทเค็นคำสั่ง **CMD18** เป็นอ่านหลายบล็อก การทำงานจะหมุนวนไปเรื่อย ๆ ตามจำนวนบล็อกที่ไฮสท์ต้องการจะอ่าน การดูแลความจำจะส่งบล็อกข้อมูลไปยังไฮสท์เรื่อย ๆ จนถึงข้อมูลบล็อกสุดท้าย เมื่อครบตามจำนวนที่ต้องการ หลังจากนั้น ไฮสท์จะส่งโทเค็น **CMD12 Stop** เพื่อหยุดกระบวนการ และการดูแลความจำจะตอบกลับด้วยโทเค็น Response ไปยังไฮสท์

บล็อกข้อมูลจะมี CRC ที่วงจรภายในการดูแลความจำไว้พ่วงตามท้ายมาด้วย เพื่อให้วัดร่องรอยของไฮสท์ทำหน้าที่ตรวจสอบความผิดพลาดระหว่างเดินทางไปยังไฮสท์ ย่อมาจาก **Cyclic Redundancy Check** ซึ่งไฮสท์จะนำข้อมูลที่ได้รับมาคำนวณหา CRC แล้วเปรียบเทียบกับ CRC ที่พ่วงมา หากตรงกันแสดงว่าข้อมูลที่ได้รับไม่มีความผิดพลาด ผู้อ่านสามารถศึกษาการทำงานของ CRC เพิ่มเติมได้จาก [wikipedia.org](https://en.wikipedia.org)

7.3.2 การเขียนข้อมูล

การเขียนข้อมูลลงในการดูแลความจำ SD สามารถทำได้โดยเขียนข้อมูลเป็นบล็อก ๆ ละ 2048-4096 ไบต์ ขึ้นกับระบบไฟล์ เช่น FAT32, EXT4 เป็นต้น ซึ่งได้อธิบายในหัวข้อที่ [7.1](#) โดยจะต้องอาศัยกระบวนการเปิดไฟล์ เขียนไฟล์ และปิดไฟล์ ตามลำดับ



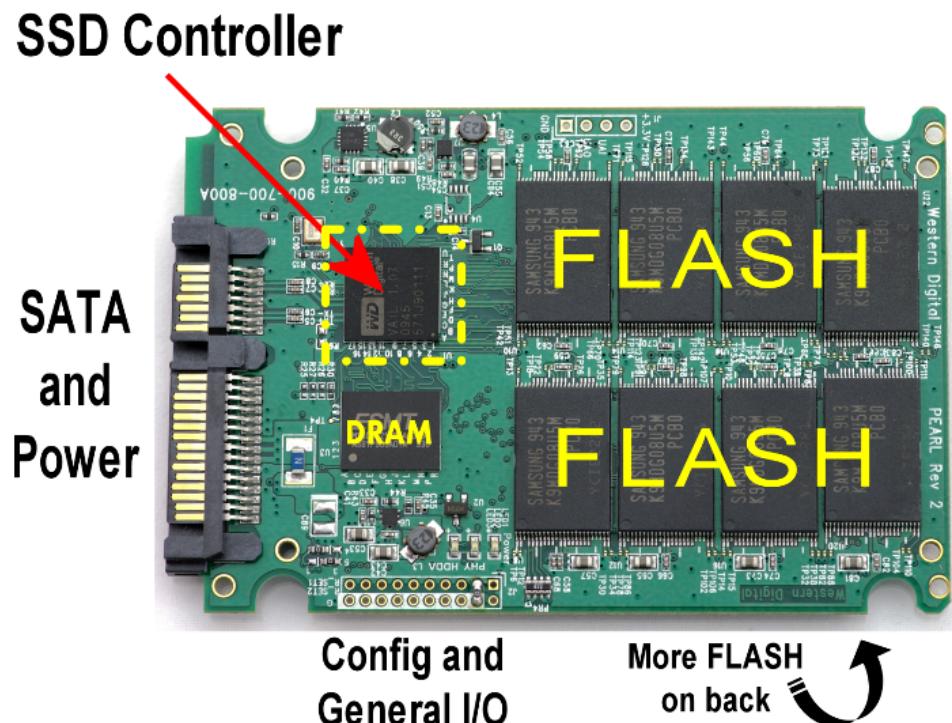
รูปที่ 7.9: ไดอะแกรมเวลาของการเขียนข้อมูลจำนวนหลายบล็อกจากการดูแลความจำ SD ที่มา: [San-Disk Corporation \(2003\)](#)

การเขียนข้อมูลจากการดูแลความจำ SD แบ่งเป็น การเขียนข้อมูลจำนวน 1 บล็อก และการเขียนจำนวนหลายบล็อก การทำงานของทั้งสองวิธีคล้ายกันมาก ดังนี้ ไฮสท์ส่งโทเค็นคำสั่ง **CMD25** ไปยังการดูแลความจำ การดูแลความจำจึงตอบกลับด้วยโทเค็น Response ไปยังไฮสท์ ไฮสท์จึงส่งบล็อกข้อมูลที่ต้องการเขียนไปยังการดูแลความจำ เรื่อย ๆ จนถึงบล็อกข้อมูลสุดท้าย เมื่อครบตามจำนวนที่ต้องการ ไฮสท์จะส่งโทเค็นคำสั่ง **CMD12 Stop** เพื่อหยุดกระบวนการและการดูแลความจำจะตอบกลับด้วยโทเค็น Response ไปยังไฮสท์ หากเป็นโทเค็นคำสั่ง **CMD24** จะหมายถึง การเขียนเพียงบล็อกเดียว ขบวนการจะเสร็จสิ้นเร็วกว่าการเขียนจำนวนหลายบล็อก โดยไม่ต้องมีการโต้ตอบระหว่างไฮสท์และการดึง

บล็อกข้อมูลที่เขียนจะมี CRC ที่ไฮสท์คำนวณก่อนเพิ่มเติมด้วย เพื่อทำหน้าที่ตรวจสอบความผิดพลาดของข้อมูลที่เขียนระหว่างเดินทางจากไฮสท์ไปยังการดูแลความจำ ซึ่งการดูแลความจำนำข้อมูลที่ได้รับมาคำนวณหา CRC

แล้วเบรียบเทียบกับ CRC ที่พ่วงมา หาก CRC ทั้งสองชุดมีค่าตรงกันแสดงว่าข้อมูลที่ได้รับไม่มีความผิดพลาด ระหว่างที่คำนวนค่าของ CRC นี้ การ์ดจะแจ้งสถานะ Busy กลับไปยังไฮส์ต์เพื่อให้ไฮส์ต์รอ หลังจากนั้น ไฮส์ต์จึงส่งบล็อกข้อมูลที่ต้องการต่อเมื่อสถานะเปลี่ยนจาก Busy เป็น Ready

7.4 โซลิดสเตทไดรฟ์ (Solid-State Disk: SSD)



รูปที่ 7.10: แผ่นวงจรพิมพ์ภายในอุปกรณ์ SSD ชนิด SATA III ประกอบด้วยชิพหน่วยความจำแฟลช ไอนามิกแรม คอนโทรลเลอร์สำหรับควบคุม ที่มา: thatoldnews.site

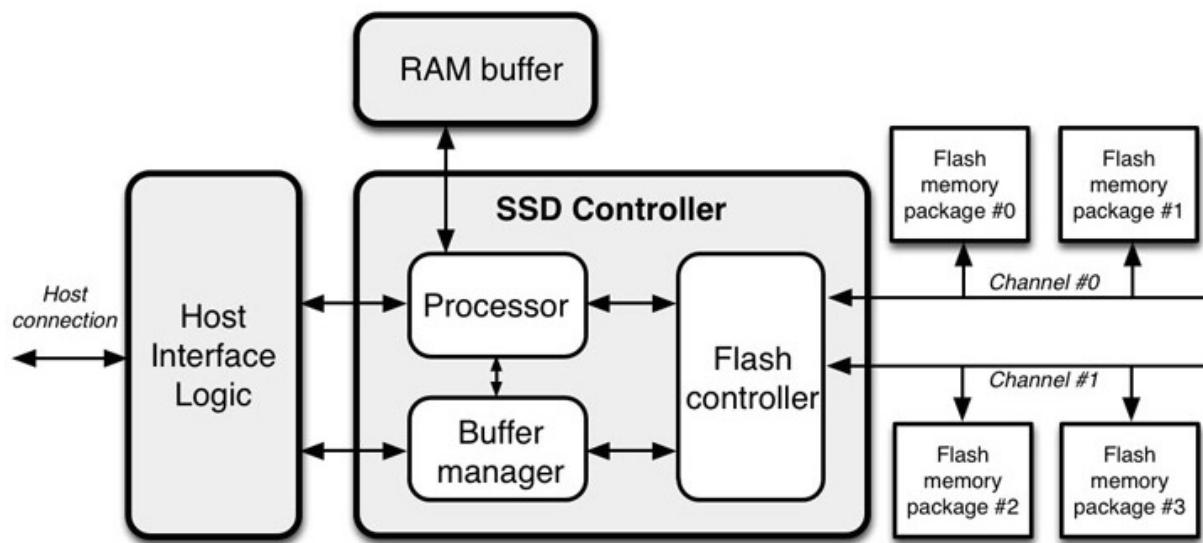
SSD ใช้ชิปที่ใกล้เคียงกับฮาร์ดดิสก์ไดรฟ์ เพื่อความหมายที่ใกล้เคียงกัน ความจุของ SSD มีแนวโน้มเพิ่มสูงขึ้น ทำให้ SSD มีขนาดเริ่มต้นตั้งแต่ 120-128 กิกะไบต์ (GiB) ขึ้นไป แนวโน้มตันทุน/ความจุหนึ่งหน่วยของ SSD จะถูกกลางเรื่อยๆ จนใกล้เคียงกับตันทุน/ความจุหนึ่งหน่วยฮาร์ดดิสก์ไดรฟ์ในอนาคต โดยองค์ประกอบหลักที่สำคัญ คือ หน่วยความจำแฟลช NAND ที่มีความจุต่อชิปเพิ่มสูงขึ้นไปอีก และใช้หน่วยความจำ SDRAM เพื่อทำหน้าที่เป็น แคช หรือ บัฟเฟอร์ และเพิ่มประสิทธิภาพการทำงานให้รวดเร็วขึ้น

โครงสร้างภายในของ SSD ในรูปที่ 7.10 ประกอบด้วย แผ่นวงจรพิมพ์ภายในอุปกรณ์ SSD ชนิด SATA III ส่วนเชื่อมต่อ กับ เมนบอร์ด คอมพิวเตอร์ ไมโครคอนโทรลเลอร์ บัฟเฟอร์ หรือ แคช และชิปหน่วยความจำแฟลชจำนวนหนึ่งตามขนาดความจุ

- ชิปหน่วยความจำแฟลช NAND ความจุสูงเรียงตัวต่อเนื่องกันจนได้ความจุมากพอตามที่ระบุ โดยการจัดเรียงทั้งด้านบน (Channel 0) และด้านล่าง (Channel 1) ของแผ่นวงจรพิมพ์หลัก เนื่องจาก

หน่วยความจำแฟลชมีเวลาเข้าถึงนานกว่าหน่วยความจำ SDRAM นักออกแบบจึงติดตั้งชิปหน่วยความจำ SDRAM เพิ่มเติม เพื่อทำหน้าที่เป็นแคชหรือบัฟเฟอร์ให้กับ SSD ผู้อ่านจะสังเกตว่าชิปหน่วยความจำแฟลชนี้ลักษณะตัวถังคล้ายกับรูปที่ 7.3

- วงจรควบคุม (Controller) นิยมใช้ไมโคร คอนโทรลเลอร์ และ เฟิร์มแวร์ ทำงาน โดยเฉพาะไมโคร คอนโทรลเลอร์ของบริษัท ARM ตระกูล Cortex R ซึ่ง R ย่อมาจากคำว่า Real Time เมماะกับการควบคุมบางส่วนของรัตนยนต์ แขนหุ่นยนต์ ฮาร์ดดิสก์ไดรฟ์ เครื่องมือทางการแพทย์ อุปกรณ์เครื่องข่าย เป็นต้น ที่มา: anandtech.com ผู้อ่านสามารถค้นคว้าเรื่องนี้จากข้อมูลการตลาดของชีพีย ARM ในหัวข้อที่ ??
- วงจรเชื่อมต่อกับไฮสต์ (Host Interface) การเชื่อมต่อที่ได้รับความนิยม ได้แก่ SATA III และ NVMe Express (NVMe) SATA III ได้รับความนิยมในระยะแรก เนื่องจากเป็นมาตรฐานของฮาร์ดดิสก์ไดรฟ์ มาก่อน ในขณะที่ NVMe ได้รับความนิยมเพิ่มมากขึ้นเรื่อยๆ เนื่องจากประสิทธิภาพสูงกว่า SATA III ผู้อ่านสามารถศึกษาเพิ่มเติมได้ที่ pcworld.com



รูปที่ 7.11: บล็อกโดยแกรมภายใน SSD ประกอบด้วย ส่วนเชื่อมต่อกับเครื่อง ไมโคร คอนโทรลเลอร์ บัฟเฟอร์ และหน่วยความจำแฟลช ที่มา: codecapsule.com

ผู้อ่านสามารถศึกษาบล็อกโดยแกรมภายใน SSD เพิ่มเติมได้ในรูปที่ 7.11 ซึ่งประกอบด้วย

- โปรเซสเซอร์ (Processor)** สำหรับรันคำสั่งในเฟิร์มแวร์ สำหรับควบคุมการทำงานภายใน และ เชื่อมต่อกับคอมพิวเตอร์ไฮสต์ ดังนั้น ผู้อ่านควรตรวจสอบผู้ผลิตเป็นระยะๆ ว่ามีการอัปเดตเฟิร์มแวร์ของ SSD รุ่นที่ใช้หรือไม่
- แฟลช คอนโทรลเลอร์ (Flash Controller)** สำหรับควบคุม การ อ่าน/เขียน ข้อมูล หน่วย ความ จำ แฟลชนี้ลักษณะตัวถังคล้ายกับการทำงานของหน่วยความจำ SD ซึ่งกล่าวในหัวข้อที่ 7.2

- **บัฟเฟอร์ (Buffer)** และ **การบริหารจัดการบัฟเฟอร์ (Buffer Management)** อาศัยหน่วยความจำ SDRAM เป็นบัฟเฟอร์เพื่อพักเก็บข้อมูลช่วงระหว่างที่เข้ามายังตัวฮาร์ดดิสก์ ทำให้การอ่าน/เขียนมีระยะเวลาเข้าถึงเฉลี่ย ดังนั้น การใช้หน่วยความจำ SDRAM ให้เต็มประสิทธิภาพ และคุ้มค่า จึงต้องมีการบริหารจัดการบัฟเฟอร์ ทำให้ต้นทุนการผลิตต่ำลง

7.5 ฮาร์ดดิสก์ไดรฟ์ (Hard Disk Drive: HDD)



รูปที่ 7.12: โครงสร้างและองค์ประกอบของ อุปกรณ์ ฮาร์ดดิสก์ไดรฟ์ (HDD) ชนิด Serial ATA ที่มา: blogspot.com

อุปกรณ์เก็บรักษาข้อมูลที่ได้รับความนิยมจากในอดีต และพัฒนามาเป็นเวลายาวนาน แผ่นงานแม่เหล็กหมุน มีเส้นผ่าศูนย์กลางสองขนาดที่นิยมผลิต คือ 2.5 นิ้ว และ 3.5 นิ้ว หมุนด้วยความเร็วสูงตั้งแต่ 5400 ถึง 10000 รอบต่อนาที จุดเด่นของหน่วยความจำขนาดสารแม่เหล็ก คือ ความจุข้อมูลที่มากกว่า เริ่มต้นที่หลายร้อยกิกะไบต์ (GB) จนถึงหลายเทราไบต์ (Tera Byte) โดย 1 เทราไบต์ ประมาณเท่ากับ 1000 กิกะไบต์ (GB) ทำให้ราคาต่อความจุต่ำลง ต้นทุนโดยรวมจึงถูกลง มีอายุการใช้งานที่ยาวนานพอสมควร จุดอ่อน คือ ประสิทธิภาพด้านความเร็วที่ต่ำกว่า ฮาร์ดดิสก์ไดรฟ์ต้องการปริมาณและน้ำหนักสำหรับจัดเก็บมากกว่า ทำให้โครงสร้างใหญ่ขึ้น ตัวอุปกรณ์รองรับแรงกระแทกมากเมื่อเทียบกับการตกลงหรือแผ่นดินไหวได้น้อยกว่า บริโภคพลังงานสูงกว่าโซลิดสเตทไดรฟ์ เนื่องจากมีการหมุนงานแม่เหล็กเกือบทั้งหมดเวลา จึงเกิดความร้อนแฝงมากกว่า จึงต้องอาศัยอุปกรณ์อื่น ๆ เช่น พัดลมช่วยระบายความร้อน หรือ เครื่องปรับอากาศในห้องคอมพิวเตอร์และศูนย์ข้อมูล

7.5.1 โครงสร้างของฮาร์ดไดร์ฟเชิงกายภาพ

โครงสร้างและองค์ประกอบของอุปกรณ์ฮาร์ดไดร์ฟ (HDD) เชิงกายภาพใน รูปที่ 7.12 ประกอบด้วย

- แผ่นจานแม่เหล็ก (Platter) ซ้อนกันหลาย ๆ ชั้น เก็บข้อมูลบนจานแม่เหล็ก เพื่อเพิ่มความจุโดยรวม และหมุนพร้อมกันด้วยความเร็ว 5,400 - 10,000 รอบต่อนาที ซึ่งเท่ากับ 90-134 รอบต่อวินาที
- แขนกล (Actuator Arm) มีหัวอ่าน/เขียนข้อมูลสำหรับแต่ละจานยึดอยู่บนแขนกลเดียวกัน แผ่นละ 2 หัว แต่ละจานมีหัวอ่าน/เขียน 1 คู่ ด้านบนและด้านล่าง โปรดสังเกตขนาดของหัวอ่าน/เขียนที่เล็กมาก
- แผ่นวงจรควบคุม (Controller Board หรือ Logic Board) ประกอบด้วยชิปไมโครคอนโทรลเลอร์ที่มีซีพียู ARM ตระกูล Cortex R และอื่น ๆ ทำหน้าที่ควบคุมการทำงานโดยรวม โดยซีพียู ARM มีส่วนแบ่งการตลาดสูงมากจากการสำรวจในปี ค.ศ. 2010 ในตารางที่ 1.1
- วงจรเชื่อมต่อ กับ ไฮสต์ (Host Interface) และเครื่องคอมพิวเตอร์ผ่านทาง External Interface ได้แก่ Parallel ATA ([wikipedia](#)) Serial ATA ([wikipedia](#)) NVMe ([wikipedia](#)) เป็นต้น ขนาดบัฟเฟอร์หรือแคช หน่วยเป็น เมบิไบต์ (MiB) เพื่อพักเก็บข้อมูลล่าสุดที่ได้ทำการอ่านหรือเขียนไว้ก่อนทำให้ระยะเวลาเข้าถึงสั้นลง หรือประสิทธิภาพสูงขึ้น

7.5.2 โครงสร้างของฮาร์ดไดร์ฟเชิงตรรกะ

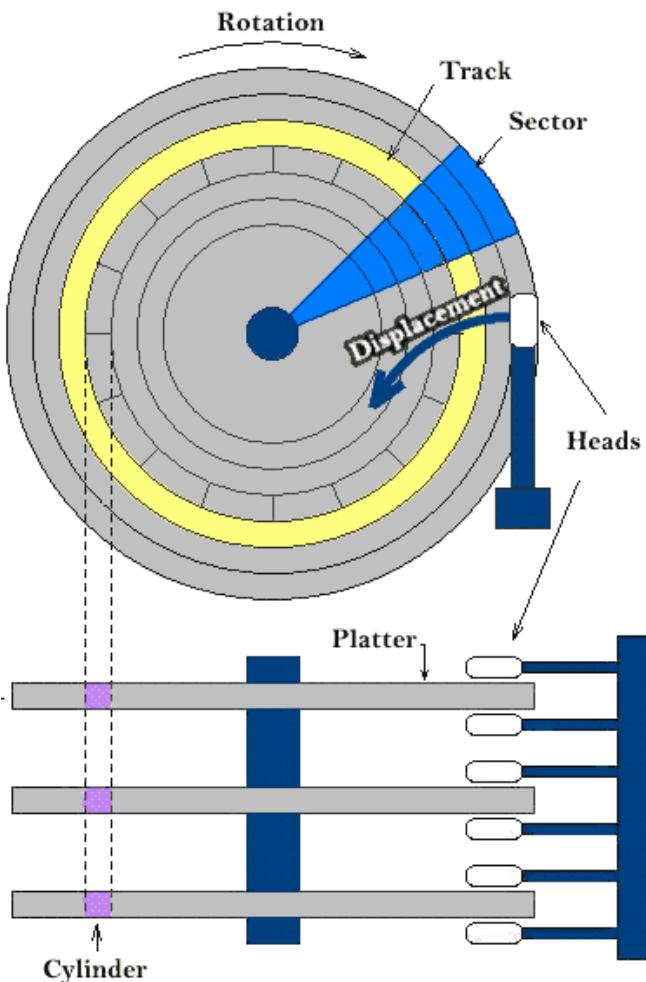
แผ่นจานแม่เหล็กแบ่งเป็นหลาย ๆ แทร็ก (Track) หรือ วงรอบ แต่ละแทร็กจะมีจำนวนเซกเตอร์ (Sector) เท่า ๆ กัน พื้นที่หนึ่งเซกเตอร์ มีขนาดความจุ 512 ไบต์เสมอ แต่ละแทร็กจะแบ่งพื้นที่ในแนวรัศมีจากจุดศูนย์กลางmany ขอบตามรูปที่ 7.13

ไซลินเดอร์ (Cylinder) คือ กลุ่มของแทร็กที่อยู่บนจานแม่เหล็กต่าง ๆ และอยู่ห่างจากจุดศูนย์กลางเท่ากันเรียงตัวกันเป็นทรงกระบอก โดยจะนับจากไซลินเดอร์หรือแทร็กหมายเลข 0 ซึ่งอยู่ขอบนอกสุดของ จานแม่เหล็ก โดยจะนับจากขอบนอกสุดเข้าสู่จุดศูนย์กลาง

บล็อก (Block) หรือ **คลัสเตอร์ (Cluster)** ประกอบด้วย เซกเตอร์ที่ต่อเนื่องกัน จำนวน 2^n เซกเตอร์เสมอ เช่น 2 4 8 .. เซกเตอร์ในแทร็กเดียวกัน โดยระบบปฏิบัติการจะกำหนดจำนวนเซกเตอร์ที่ต้องการยกตัวอย่าง เช่น คลัสเตอร์ขนาด 4096 ไบต์ต้องการพื้นที่จำนวน 8 เซกเตอร์

หมายเลขบล็อก LBA: Logical Block Addressing คือ การตั้งหมายเลขบล็อกที่กล่าวมาก่อนหน้านี้ให้เรียงตัวตาม หมายเลขไซลินเดอร์ หมายเลขหัวอ่าน และหมายเลขแทร็ก ทำให้ง่ายต่อการบริหารจัดการ และเป็นพื้นฐานของระบบบริหารจัดการไฟล์ ที่มา: [wikipedia](#)

ระบบไฟล์ จะจอง พื้นที่บนฮาร์ดไดร์ฟอย่างน้อย 1 บล็อก หรือ 1 คลัสเตอร์ให้กับไฟล์หนึ่งไฟล์ ยกตัวอย่าง เช่น ไฟล์ขนาด 800 ไบต์ ต้องการใช้พื้นที่ 8 เซกเตอร์ เช่นเดียวกับไฟล์ขนาด 4096 ไบต์ ในอุปกรณ์เก็บรักษาข้อมูลใด ๆ ผู้ใช้ทั่วไปสามารถเก็บข้อมูลหรือแอปพลิเคชันในรูปของไฟล์เท่านั้น ในด้าน



รูปที่ 7.13: โครงสร้างของฮาร์ดดิสก์ไดรฟ์แบ่งเป็นไซลินเดอร์ แทร็ค และเซกเตอร์ ที่มา: computable-minds.com

การบริหารจัดการไฟล์ ผู้ใช้สามารถคัดลอกหรือสำเนา (Copy) ลบ (Delete หรือ Remove) ย้าย (Move) คุ้น (Recover) ไฟล์ต่าง ๆ ได้ผ่านระบบปฏิบัติการ ในด้านการบริหารจัดการพื้นที่ ผู้ใช้สามารถจัดหมวดหมู่ไฟล์โดยการ สร้างไฟล์เดอร์ ตั้งชื่อ คัดลอก ย้าย ลบ คุ้นไฟล์เดอร์ต่าง ๆ เช่นกัน ผู้อ่านสามารถทำความเข้าใจจากการทดลองที่ 12 ภาคผนวก L

7.5.3 ความจุและประสิทธิภาพของฮาร์ดดิสก์ไดรฟ์

ความจุของฮาร์ดดิสก์ไดรฟ์สามารถคำนวณได้จากผลคูณของความจุต่อหนึ่งเซกเตอร์ จำนวนเซกเตอร์ต่อแทร็ค จำนวนไซลินเดอร์ และจำนวนหัวอ่าน ยกตัวอย่าง เช่น $512 \text{ ไบต์ต่อเซกเตอร์} \times 63 \text{ เซกเตอร์} \times 1024 \text{ ไซลินเดอร์} \times 256 \text{ หัวอ่าน} = 8,455,716,864 \text{ ไบต์}$ หรือ $8.4 \times 10^9 \text{ ไบต์}$ โดยประมาณ ทำให้ผู้ผลิตใช้หน่วยเป็น 8.4 กิกะไบต์ (GB)

อายุการใช้งานของฮาร์ดดิสก์ไดรฟ์ วัดจากจำนวนชั่วโมงที่เปิดใช้งาน ซึ่งจะมีความแตกต่างจากการวัดอายุของหน่วยความจำแฟลช ที่วัดจากจำนวนครั้งที่เขียนหรือลบข้อมูล ทำให้ฮาร์ดดิสก์ไดรฟ์มีต้นทุนการใช้งานต่อความจุและต่ออายุที่ต่ำและคุ้มค่ากว่า

การวัดประสิทธิภาพของฮาร์ดไดร์ฟอาศัยการวัด เวลาการเข้าถึง (T_{acc} , Access Time) หน่วยเป็น มิลลิวินาที ประกอบด้วย ช่วงเวลาการรอเฉลี่ย T_{rotate} เพื่อให้ตำแหน่งที่ต้องการอ่านหมุนมาอยังหัวอ่าน เรียกว่า **Rotation Latency** ช่วงเวลาการขับหัวอ่านมายังแทร็กที่ต้องการ T_{head} และช่วงเวลาการถ่ายโอนข้อมูล (T_{xfer} , Transfer Time)

$$T_{acc} = T_{rotate} + T_{head} + T_{xfer} \quad (7.1)$$

ดังนั้น เวลาการเข้าถึงจะมีความสัมพันธ์กับความเร็วรอบในการหมุนของจานแม่เหล็ก ตามสมการต่อไปนี้

$$T_{rotate} = \frac{0.5}{V_{rotate}} \quad (7.2)$$

องค์ประกอบสุดท้าย คือ T_{xfer} ขึ้นอยู่กับชนิดการเข้ามต่อ กับคอมพิวเตอร์โซลูชัน SATA จะมีการพัฒนาประสิทธิภาพสูงขึ้นเป็น SATA III ตามที่ได้กล่าวไว้ก่อนหน้า

ช่วงเวลาการรอเฉลี่ย T_{rotate} แปรผันกับ V_{rotate} หรือ ความเร็วรอบในการหมุนของจาน หน่วยเป็นรอบต่อวินาที (Round per Minute: RPM) และตำแหน่งของแทร็กที่ต้องขับหัวอ่านไป ประสิทธิภาพของการอ่านและการเขียนจะขึ้นกับตำแหน่งของข้อมูล การอ่านหรือเขียนข้อมูลที่เรียงต่อเนื่อง (Sequential) จะดีกว่าการอ่านหรือเขียนข้อมูลที่กระจายตัวไม่ต่อเนื่อง (Fragmented) เช่นเดียวกับการอ่านและเขียนข้อมูลของหน่วยความจำแฟลช ถ้ามีการจัดเรียงไฟล์ข้อมูลในแต่ละเซกเตอร์ให้ต่อเนื่องกัน หรือเรียกว่า **Defragmentation** จะทำให้เวลาการเข้าถึงของฮาร์ดไดร์ฟเฉลี่ยน้อยลงรวมถึงการจัดเรียงลำดับการอ่านเขียนข้อมูลบนดิสก์ (Disk Scheduling) จะช่วยให้ประสิทธิภาพเพิ่มขึ้น ผู้อ่านสามารถศึกษาอัลกอริズึมได้ที่ geeksforgeeks.org

7.6 สรุปท้ายบท

ระบบไฟล์ และ อุปกรณ์เก็บรักษาข้อมูล จะต้องประสานงานกัน เพื่อให้เครื่องเนล ผู้ใช้งาน และแอปพลิเคชัน อื่น ๆ สามารถบริหารจัดการไฟล์โปรแกรมไฟล์ข้อมูลต่าง ๆ ได้อย่างถูกต้อง และมีประสิทธิภาพ สอดคล้องกับภารกิจของระบบคอมพิวเตอร์ นอกจากนี้ ผู้อ่านจะสังเกตเห็นว่า ขนาดหรือความจุของบล็อกข้อมูลในอุปกรณ์เก็บรักษาข้อมูล เช่น

- หน่วยความจำแฟลชขนาด 1 เพจจะมีขนาดเท่ากับ 2^{11} หรือ 2048 ไบต์
- การ์ดหน่วยความจำ SD ขนาด 1 บล็อกจะมีขนาดเท่ากับ 2^{11} หรือ 2048 ไบต์
- ฮาร์ดไดร์ฟขนาด 1 เซ็กเตอร์จะมีขนาดเท่ากับ 2^9 หรือ 512 ไบต์

สอดคล้องกับ ขนาดของบล็อกข้อมูลในระบบบริหารจัดการไฟล์ และ ขนาดของเพจข้อมูลในเวอร์ชวลเม莫รี่ ซึ่งจะมีขนาดเป็นจำนวนเท่าของสองเลข และสำหรับลินก์มีขนาดเท่ากับ 4096 หรือ 2^{12} ไบต์ ในหัวข้อที่ 7.1.3

ตารางที่ 7.3 เป็นการเปรียบเทียบประสิทธิภาพของอุปกรณ์เก็บรักษาข้อมูลหลายชนิดในด้านต่าง ๆ ประกอบด้วยชิปหน่วยความจำแฟลช NAND, อุปกรณ์ SSD และอุปกรณ์ HDD โดยผู้อ่านจะต้องใช้ปีที่ผลิตเป็นหลักยึดในการทำความเข้าใจ เนื่องจากปีที่ผลิตจะมีผลต่อประสิทธิภาพ ความจุและกำลังไฟฟ้าสูงสุด เพราะเทคโนโลยีการผลิตอุปกรณ์เหล่านี้เปลี่ยนแปลงตลอดเวลา

ตารางที่ 7.3: การเปรียบเทียบประสิทธิภาพด้านต่าง ๆ ของอุปกรณ์เก็บรักษาข้อมูล

อุปกรณ์	แฟลช NAND	SSD	HDD
ผู้ผลิต หมายเลขโมเดล ที่มา: ปี ค.ศ. ความจุ	Micron MT29F2G08AABWP micron.com 2004 25 MiB	Micron MTFDDAK120MAV micron.com 2013 120 KiB	Western Digital 5K1000 hgst.com 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 -
โวลเตจสูงสุด กำลังไฟสูงสุด	4.6 โวลต์ 23 มิลลิวัตต์	5.0 โวลต์ 150 มิลลิวัตต์	5.0 โวลต์ 1.6 วัตต์

อุปกรณ์เก็บรักษาข้อมูล จะเชื่อมต่อกับหน่วยความจำหลัก SDRAM ผ่านวงจรด้านอินพุต/เอาต์พุต โดยใช้กลไก DMA จากหัวข้อที่ 6.13 และ กลไกการทำอินเทอร์รัปต์ จากหัวข้อที่ 6.12 ในชั้นกายภาพ (Physical) ร่วมกับหลักการ Memory Mapped File ในรูปที่ 3.19 ในระดับสูงขึ้น

7.7 คำถ้ามท้ายบท

1. จงเปรียบเทียบ การอ่าน หรือ เขียน ของ อุปกรณ์เก็บรักษาข้อมูลชนิดต่าง ๆ ได้แก่ การดีดหน่วยความจำ SD, SSD, HDD ในแบบเหล่านี้
 - ขนาดของข้อมูลขั้นต่ำที่สามารถอ่าน หรือเขียนได้ หน่วยเป็นไบต์
 - ระยะเวลาเข้าถึง (Access Time) และช่วงเวลาอย่างไร
2. จงเปรียบเทียบ กำลังไฟ สูงสุด ของ หน่วยความจำ ชนิดต่าง ๆ ได้แก่ หน่วยความจำแฟลช, การดีดหน่วยความจำ SD, SSD ที่ความจุเท่ากัน เท่าที่จะหาข้อมูลได้
3. เหตุใดการอ่าน หรือเขียน ข้อมูล ใน อุปกรณ์เก็บรักษาข้อมูล จึงต้องอาศัยกลไกของ DMA และ การอินเทอร์รัปต์ ทำงานร่วมกัน
4. จงเปรียบเทียบ การเชื่อมต่อ ชนิด PATA และ SATA ใน แบบ ประสิทธิภาพ

5. จงเปรียบเทียบการเชื่อมต่อชนิด SATA SATA II และ SATA III ในแต่ละสิทธิภาพ
6. จงเปรียบเทียบการเชื่อมต่อชนิด SATA III และ NVMe ในแต่ละสิทธิภาพ
7. จงเปรียบเทียบหน่วยความจำ SD คลาสต่าง ๆ ในแต่ละสิทธิภาพ

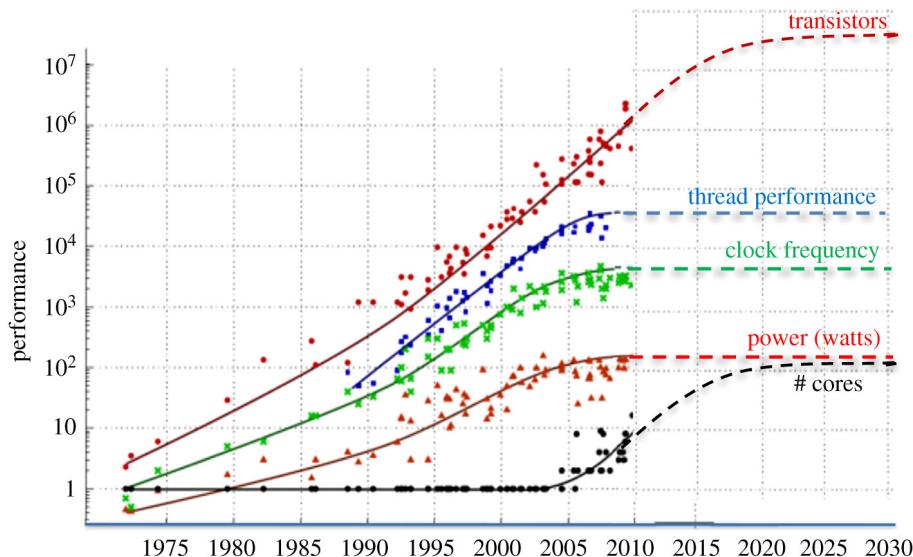
บทที่ 8

การคำนวณแบบขนาน (Parallel Computing) ด้วยบอร์ด Pi3/Pi4

จำนวนทรานซิสเตอร์ และ ประสิทธิภาพของซีพียู เพิ่มขึ้นเฉลี่ยไม่มาก ประมาณ 10 เท่า ในเวลา 10 ปี จากเส้นกราฟในรูปที่ 8.1 แต่ในช่วงปี ค.ศ. 1995-2005 จำนวนทรานซิสเตอร์ และ ประสิทธิภาพเพิ่มขึ้นเฉลี่ยเป็น 2 เท่าทุก 1.5 ปี หรือ 10 เท่า ใน 5 ปี โดยประมาณตามกฎของ Moore (Moore's Law) ที่มา: [wikipedia](#) สืบเนื่องจากการเพิ่มจำนวนบิตข้อมูลที่ประมวลผลได้จาก 4 บิต เป็น 8, 16, 32 และ 64 บิต ต่อ 1 คากเวลา (Cycle Time) ของสัญญาณคล็อก การปรับโครงสร้างการทำงานจากไปป์ไลน์ เป็นซูเปอร์สเกลาร์ (SuperScalar) ซึ่งเป็นการเริ่มต้นของการทำงานแบบขนานระดับคำสั่ง (Instruction Level Parallelism) เป็นการทำงานแบบขนานระดับเฟลด (Thread Level Parallelism) สำหรับชิปที่รองรับมัลติเฟลด (Multithread) ที่มา: [Patterson and Hennessy \(2016\)](#) และ มีจำนวนแกนประมวลผล (Core) ต่อชิปมากขึ้นเรื่อยๆ เรียกว่า ซีพียูมัลติคอร์ (Multi-Core) ที่มา: [Tanenbaum and Austin \(2012\)](#); [Tanenbaum and Bos \(2014\)](#) และ [wikipedia](#) และ จำนวนเพิ่มขึ้นถึงหนึ่งร้อยแกนประมวลผลต่อชิป เรียกว่า ซีพียูแมนีคอร์ (Many Core) ที่มา: [Tanenbaum and Bos \(2014\)](#) และ [wikipedia](#) ในปัจจุบันและอนาคตอันใกล้

แกนนอน คือ หมายเลขปี ค.ศ. แกนตั้ง คือ ประสิทธิภาพ (Performance) หน่วยเป็นจำนวนเท่าของค่าประสิทธิภาพเทียบกับซีพียูตัวแรก เนื่องจากเลขประสิทธิภาพเพิ่มขึ้นรวดเร็วมากจนต้องใช้แกนเลขสิบยกกำลัง จุดสีแต่ละจุด คือ ประสิทธิภาพในด้านต่างๆ ของซีพียูหนึ่งตัว ได้แก่ จำนวนทรานซิสเตอร์ภายในซีพียู (จุดกลมแดง) ค่าประสิทธิภาพของซีพียูเพียง 1 แกนประมวลผล (จุดสีเหลี่ยมฟ้า) ความถี่สัญญาณคล็อกสูงสุด (kakbat สีเขียว) ค่ากำลังไฟสูงสุด (สามเหลี่ยมสีแดง) และ จำนวนแกนประมวลผลต่อชิป (จุดกลมดำ) เส้นทิบสีต่างๆ คือ เส้นลดตอน (Regression) ของประสิทธิภาพด้านต่างๆ จากข้อมูลจุดสีน้ำเงิน เส้นประสีต่างๆ คือ เส้นพยากรณ์ที่ผู้เขียนเอกสารอ้างอิงทำนายไว้ต่อเนื่องจากเส้นทิบ สรุปได้ว่า

- จำนวนทรานซิสเตอร์ต่อชิปมีจำนวนเพิ่มขึ้นจากชิปซีพียูตัวแรก ในปี ค.ศ. 1971 จนถึงจุดอิมตัวที่จำนวน 10^8 เท่า หรือ ประมาณหลายพันล้านตัว ในปัจจุบัน และ มีแนวโน้มเพิ่มขึ้นตามจำนวนแกนประมวลผลที่เพิ่มขึ้น



รูปที่ 8.1: ประสิทธิภาพของการประมวลผลด้วยชิปซีพียูชนิดมัลติคอร์และแมนีคอร์ ที่มา: royalsociety-publishing.org, John (2020)

- ความถี่สัญญาณคลือกสูงสุด หน่วยเป็น เมกะเอิร์ตซ์ ในปี ค.ศ. 1971 - 1995 เพิ่มขึ้นต่อเนื่องประมาณ 10 เท่าในเวลา 12 - 15 ปี และเพิ่มขึ้นแบบก้าวกระโดดประมาณ 10 เท่าภายในระยะเวลา 10 ปี ระหว่าง ค.ศ. 1995 - 2005 และถึงจุดอิ่มตัวในปี ค.ศ. 2005 เป็นต้นมา
- ค่าประสิทธิภาพของซีพียู 1 แกนประมวลผล (Single Thread Performance) ซีพียูในอดีตมีเพียง 1 แกนประมวลผลเท่านั้น การพัฒนาซีพียูในอดีตนั้นการเพิ่มประสิทธิภาพของซีพียู 1 แกนประมวลผล ด้วยเทคโนโลยีการผลิตซีพียูจากหลักไมโครเมตร จนเหลือหลักนาโนเมตร เรียกว่า การย่อส่วนเทคโนโลยี (Technology Scaling) ที่มา: [Muller \(2003\)](https://www.semiconductortribune.com/technology/scaling/) และตัวอย่างใน [wikipedia](https://en.wikipedia.org/wiki/Technology_scaling) ทำให้จำนวนทรานซิสเตอร์และความถี่สัญญาณคลือกเพิ่มขึ้นอย่างมีนัยสำคัญ จึงส่งผลให้ประสิทธิภาพซีพียูเพิ่มขึ้นอย่างก้าวกระโดดตั้งแต่ปี ค.ศ. 1990-2005 ตั้งแต่ปี ค.ศ. 2005 เป็นต้นมา ค่าประสิทธิภาพของซีพียู 1 แกนประมวลผลเกือบคงที่ แต่ประสิทธิภาพโดยรวมของซีพียูจะเพิ่มขึ้นเนื่องจากซีพียูมีจำนวนแกนประมวลผลเพิ่มขึ้น
- จำนวนแกนประมวลผลต่อชิป ในอดีตมีเพียง 1 แกนประมวลผล และเพิ่มจำนวนขึ้นอย่างก้าวกระโดด และเพิ่มขึ้นอย่างต่อเนื่องจนอาจจะอิ่มตัวที่จำนวน 128 แกนประมวลผลโดยประมาณ
- ค่ากำลังไฟสูงสุด (Maximum Power) ต่อชิปนั่วยังเป็นวัตต์ มีค่าเพิ่มขึ้นตามความถี่สัญญาณคลือกสูงสุด จะมีแนวโน้มค่อนข้างคงที่ เพราะข้อจำกัดด้านการระบายความร้อนออกจากตัวชิป แนวโน้มการบริโภคพลังงานที่ไม่เพิ่มสูงอาจเนื่องมาจากเทคโนโลยีการผลิตที่พัฒนาในอนาคต จึงเป็นที่มาของเนื้อหาในบทนี้ โดยมีวัตถุประสงค์ดังต่อไปนี้
 - เพื่อให้เข้าใจโครงสร้างและการทำงานของซีพียูชนิดมัลติคอร์ตามหลักการคอมพิวเตอร์แบบขนานชนิด SMP: Shared Memory Multiprocessor ที่มา: [Patterson and Hennessy \(2016\)](https://www.cs.cmu.edu/~bhennedy/parallel/) และ [wikipedia](https://en.wikipedia.org/wiki/Shared-memory_parallel_computing)

- เพื่อให้เข้าใจความแตกต่างระหว่างการคำนวณแบบอนุกรม (Serial) และแบบขนาน (Parallel)
- เพื่อให้เข้าใจ การพัฒนาซอฟต์แวร์แบบมัลติ เฮอด และแบบขนาน ด้วย ไลบรารี OpenMP ใน การทดลองที่ 13 ภาคผนวก M

8.1 เครื่องคอมพิวเตอร์แบบขนานชนิดมัลติคอร์

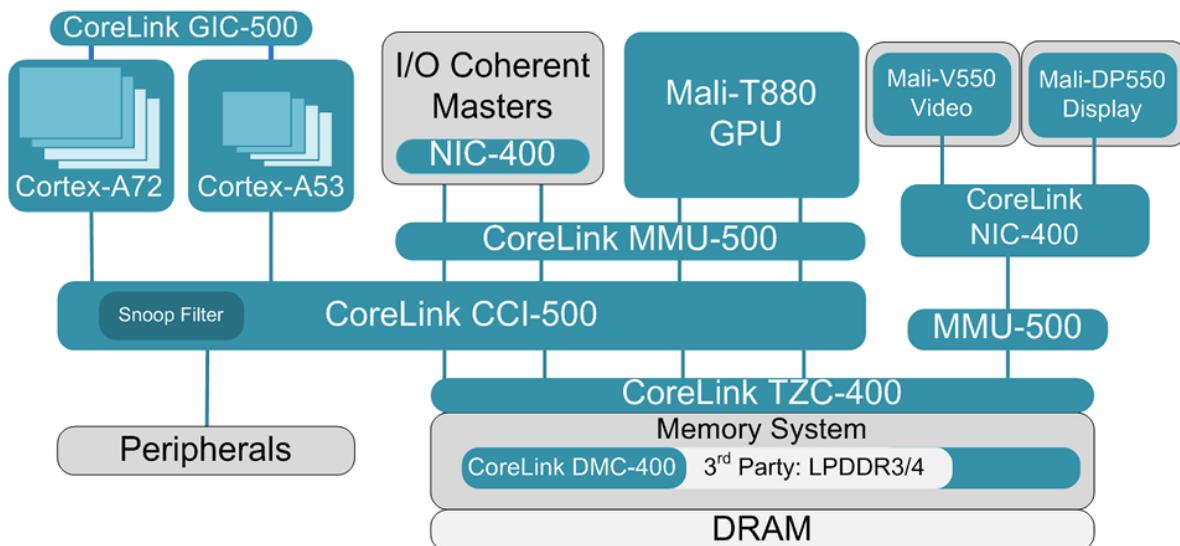
เทคโนโลยีสาร กึง ตัวนำ ที่ใช้สร้าง คอมพิวเตอร์ ทุก ชนิด ที่ได้ แจกแจง ในหัวข้อ ที่ 1.1 ได้ พัฒนาอย่างต่อเนื่อง เป็นระยะ เวลา ไม่น้อย กว่า ครึ่งศตวรรษ ทำให้ คอมพิวเตอร์ มี วิวัฒนาการ อย่าง รวดเร็ว และ ประสิทธิภาพ เพิ่มขึ้นอย่าง ก้าวกระโดด การคำนวณแบบขนาน เป็นทางเลือกหนึ่ง ที่เพิ่ม ประสิทธิภาพ โดย สามารถ แบ่งเป็น ชนิด หลัก ๆ ตาม การใช้งาน หน่วย ความจำ หลัก หรือ หน่วย ความจำ ภายใน ภาพ ที่มา: Patterson and Hennessy (2016) คือ

- เครื่องคอมพิวเตอร์แบบขนานชนิด SMP (Shared Memory Multiprocessor) ทุก ๆ แกน ประมวลผล ใช้ หน่วย ความจำ หลัก หรือ SDRAM และ อินพุต/เออร์พุต ร่วมกัน เป็นต้น แบบของ ซีพียู ชนิด มัลติ คอร์ ใน เครื่องคอมพิวเตอร์ ทุก ชนิด ปัจจุบัน ซีพียู เหล่านี้ มาจาก ของ ผู้ผลิต ราย สำคัญ เช่น บริษัท Intel, บริษัท AMD และ บริษัท ARM เป็น หลัก กรณี ศึกษา ใน หัวข้อ ที่ 8.1.1 และ 8.1.2 ซีพียู ARM Cortex A53/A72 ทั้ง 4 แกน ประมวลผล บน บอร์ด Pi3/Pi4 กรณี ศึกษา ใน ตำรา เล่ม นี้ ใช้ หลัก การ หน่วย ความจำ ร่วม กัน (SMP) ซึ่ง ระบบปฏิบัติ การ Raspberry Pi OS (ลิ奴กซ์) สามารถ รองรับ การ ทำงาน แบบ มัลติ เฮอด และ การ คำนวณ แบบ ขนาน (Parallel Computing) ได้ ชิป ARM ใน ปัจจุบัน รองรับ จำนวน แกน ประมวลผล มาก ขึ้น เรียกว่า ซีพียู มัลติ คอร์ และ มาก ขึ้น จน เรียกว่า ซีพียู เมนี คอร์ ส่วน รับ ใช้ เป็น ซีพียู ส่วน รับ เครื่องคอมพิวเตอร์ ตั้ง โต๊ะ โน๊ตบุ๊ค และแท็บเล็ต ยก ตัวอย่าง เช่น ชิป Apple Silicon M1 มี จำนวน ซีพียู 8 แกน ประมวลผล ที่มา: wikipedia เนื่อง จาก เทคโนโลยี ในการ ผลิต มี ขนาด เล็ก ลง ทำ ให้ ประหยัด พลังงาน และ สามารถ บรรจุ จำนวน แกน ประมวลผล ต่อ ชิป ได้ เพิ่ม ขึ้น ตาม แนวโน้ม ใน รูป ที่ 8.1
- มัลติ คอมพิวเตอร์ (Multicomputer) ที่มา: Tanenbaum and Bos (2014); Patterson and Hennessy (2016) มี ลักษณะ สำคัญ คือ เครื่อง คอมพิวเตอร์ จำนวนมาก แต่ ละ เครื่อง สร้าง จาก ซีพียู ชนิด มัลติ คอร์ หรือ ซีพียู เมนี คอร์ ที่ เหมือน กัน และ ใช้ ระบบ ปฏิบัติ การ เดียวกัน เรียกว่า โหนด (Node) เข้า อก กัน ด้วย เครือ ข่าย ประสิทธิภาพ และ ความเร็ว สูง ภายใน ศูนย์ ข้อมูล เดียวกัน แต่ ละ เครื่อง ใช้ หน่วย ความจำ ภายใน ภาพ และ อินพุต/เออร์พุต แยก อิสระ จาก กัน การ ทำงาน ของ โปรแกรม บน เครื่อง คอมพิวเตอร์ ใช้ การ รับ ส่ง ข้อมูล ระหว่าง กัน (Message Passing) บน เครือ ข่าย เพื่อ ประสาน งาน และ สื่อสาร กัน และ สามารถ ช่วย กัน แก้ ปัญหา เดียวกัน แบบ ขนาน ได้ กรณี ศึกษา คือ เครื่อง ซูเปอร์ คอมพิวเตอร์ ใน หัวข้อ ที่ 8.1.2

8.1.1 กรณีศึกษาชิปปี้ ARM Cortex A72 และ Cortex A53

โครงสร้างของชิปปี้สำหรับคอมพิวเตอร์เคลื่อนที่ชนิดแท็บเล็ต สมาร์ตโฟน และระบบสมองกลฝังตัวที่ได้รับความนิยม อุปกรณ์เหล่านี้ประกอบด้วยชิปปี้ ARM Cortex A72 หรือรุ่นอื่น ๆ ที่ทันสมัยกว่าจำนวน 1 ถึง 4 แกนประมวลผลและ Cortex A53/A72 หรือรุ่นอื่น ๆ ที่ทันสมัยกว่าจำนวนไม่น้อยกว่า 4 แกนประมวลผล ตั้งรูปที่ 8.2 ทำงานร่วมกันตามหลักการ big.LITTLE ของบริษัท ARM โดยชิปปี้ big คือ Cortex A72 เนื่องจากแต่ละแกนประมวลผลมีประสิทธิภาพสูงทำให้บริโภคพลังงานมากกว่าเหมาะสำหรับเกมและมัลติมีเดียคุณภาพสูง และชิปปี้ LITTLE คือ Cortex A53/A72 เพราะแต่ละแกนประมวลผลมีประสิทธิภาพด้อยกว่าทำให้บริโภคพลังงานน้อยกว่าเหมาะสำหรับโปรแกรมทั่วไปที่ใช้งานปกติ ชิปปี้ทั้งหมดเชื่อมต่อกันด้วย Cache Coherence Interconnect ตัวอย่างชิปที่ใช้ชิปปี้ทั้งสองชนิดนี้ ได้แก่

- ชิป Helio X20 ที่มา: ผลิตโดยบริษัท MediaTek wikichip.org
- ชิป RK3399 ผลิตโดยบริษัท Rockchip สำหรับคอมพิวเตอร์บอร์ดเดียว Rock Pi 4 ที่มา: wiki-dot.com/rockpi.org และ
- ชิป i.MX8 NXP ผลิตโดยบริษัท NXP สำหรับบอร์ดพัฒนาระบบฝังตัว ที่มา: nxp.com และ variscite.com เป็นต้น



รูปที่ 8.2: โครงสร้างภายในของชิปปี้ ARM ตามเทคโนโลยี big.LITTLE เชื่อมระหว่างแกนประมวลผลด้วย Cache Coherent Interconnect (CCI) ที่มา: arm.com

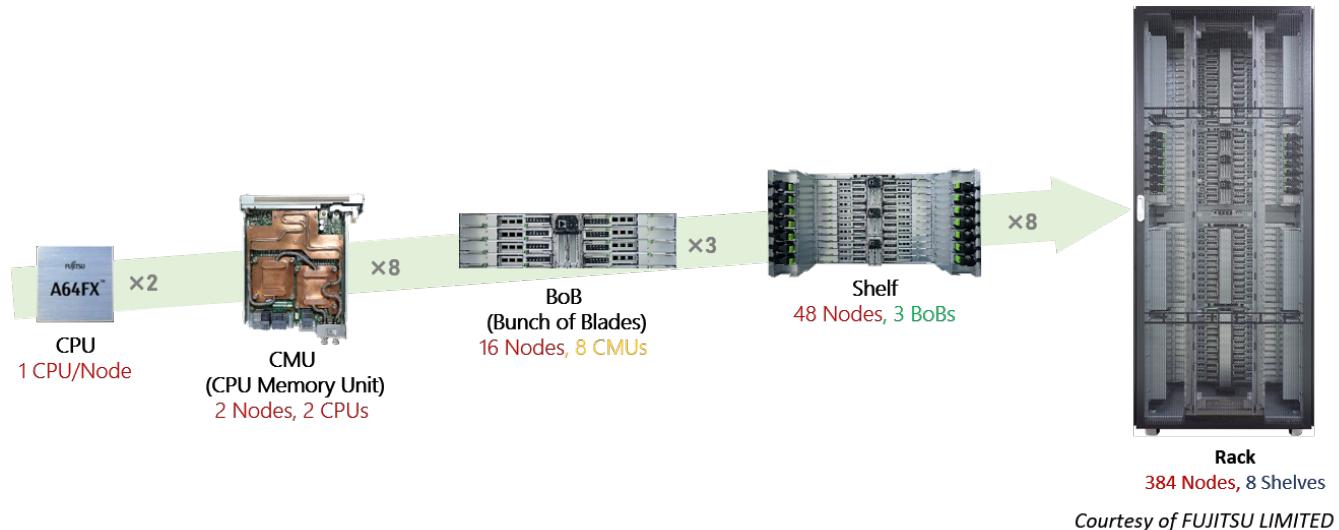
การเชื่อมต่อชิปปี้จำนวนหลาย ๆ แกนประมวลผลและใช้หน่วยความจำหลักร่วมกัน หรือ SMP มีความซับซ้อนต่ำทำให้เพิ่มจำนวนแกนประมวลผลได้ไม่ยาก การพัฒนาโปรแกรมแบบขนานบนหลักการ เชิร์ฟหน่วยความจำหลักร่วมกัน จึงมีจุดเด่นหลายประการแต่มีจุดอ่อนด้วย เช่น กัน เนื่องจากชิปปี้แต่ละแกนประมวลผลมีแคชข้อมูลลำดับที่ 1 ของตนเองและอิสระจากกันในรูปที่ 4.1 ปัญหาจะเกิดขึ้นเมื่อโปรแกรมที่ทำงานแบบมัลติเรตต์ เพื่อให้สามารถคำนวณแบบขนาน โดยมีเรตต์อย่างน้อย 2 ตัวรันอยู่ในชิปปี้ 2

แกนประมวลผล ต้องการเข้าถึงตัวแปรตัวเดียวกันในหน่วยความจำภายใน ทำให้แคชลำดับที่ 2 ต้องโหลดค่าของตัวแปรนั้นมาพักเก็บในแคชข้อมูลลำดับที่ 1 (L1 data Cache) ของทั้งสองแกนประมวลผล โดยแต่ละเรซิสเตอร์สามารถเปลี่ยนแปลงค่าตัวแปรนั้นโดยอิสระ จึงทำให้เรซิสเตอร์ทั้งสองมองเห็นค่าตัวแปรซึ่งเดียวกันแต่ค่าไม่ตรงกัน เราเรียกปัญหาสำคัญนี้ว่า **ปัญหาแคชโค希เรนท์** (Cache Coherent Problem) ที่มา: [Patterson and Hennessy \(2016\)](#)

กลไกสำคัญที่ใช้มีแกนประมวลผลเหล่านี้เข้าด้วยกัน ประกอบด้วย มอดูลชื่อ CoreLink CCI (Cache Coherent Interconnect) เพื่อรองรับจำนวนแกนประมวลผลที่เพิ่มมากขึ้น และรองรับการเชื่อมต่อระหว่างซีพียูและจีพียู ทำหน้าที่ประสานงานแคชที่กระจายอยู่ในแกนประมวลผลต่าง ๆ ให้ทำงานร่วมกันโดยไม่เกิดปัญหา ตามรูปที่ 8.2 บริษัท ARM ได้พัฒนาการเชื่อมต่อระหว่างแคชมาอย่างต่อเนื่อง โดย CCI จะดักฟัง (Snoop) ว่าซีพียูแต่ละแกนประมวลผลใช้ค่าตัวแปรเดียวกันหรือไม่ ผู้อ่านสามารถค้นคว้าเรื่องนี้เพิ่มเติมได้ใน [wikipedia.org](#)

8.1.2 กรณีศึกษา Fujitsu A64FX ในเครื่องซูเปอร์คอมพิวเตอร์

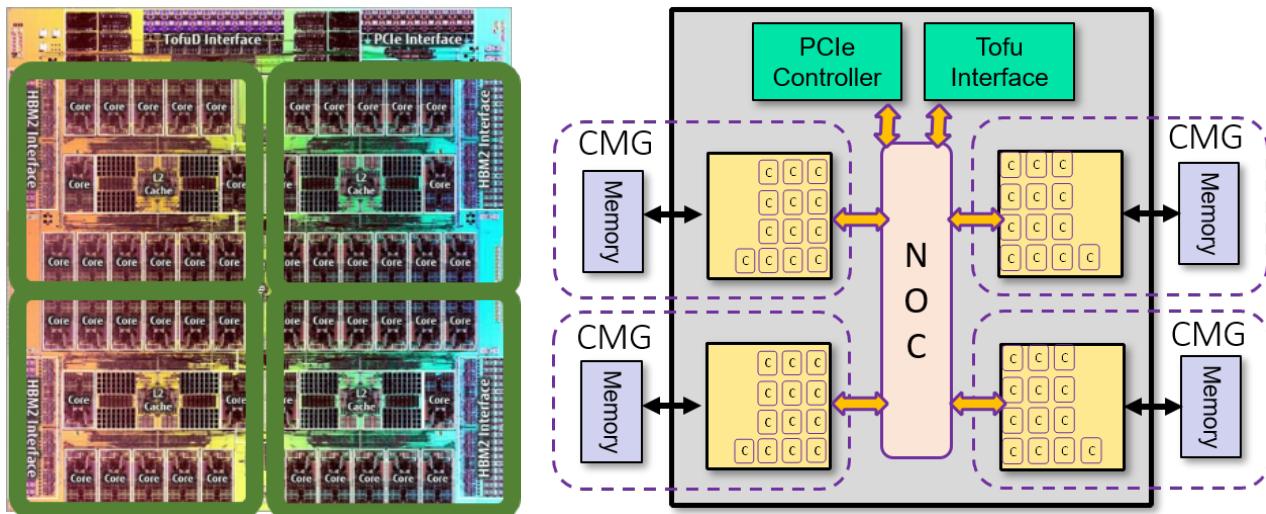
เครื่องซูเปอร์คอมพิวเตอร์ Fugaku จัดเป็นมัลติ คอมพิวเตอร์ชนิด คลัสเตอร์ขนาดใหญ่มาก ตั้งอยู่ที่ศูนย์ RIKEN Center for Computational Science ประเทศญี่ปุ่น ที่มา: [top500.org](#) ซึ่งเว็ปไซต์นี้ได้จัดอันดับ Fugaku ให้เป็นซูเปอร์คอมพิวเตอร์อันดับ 1 ของโลกในเดือนพฤษจิกายน ปี ค.ศ. 2020 จากค่าประสิทธิภาพสูงสุดหน่วยเป็น FLOPS (Floating-Point Operations Per Second) เป็นการวัดจำนวนการบวก/ลบ/คูณ เลขทศนิยมฐานสองมาตรฐาน IEEE754 ต่อวินาที ในหัวข้อที่ 2.6



รูปที่ 8.3: โครงสร้างของซูเปอร์คอมพิวเตอร์ Fugaku ประกอบด้วยตู้แร็กจำนวน 414 ตู้ ๆ ละ 384 โนนด แต่ละโนนดมี Fujitsu A64FX จำนวน 1 ชิปซึ่งภายในประกอบด้วยซีพียู ARM จำนวน 48 แกนประมวลผล ที่มา: [pcmag.com](#)

โครงสร้างของซูเปอร์คอมพิวเตอร์ Fugaku ในรูปที่ 8.3 ประกอบด้วยตู้แร็กจำนวน 414 ตู้ ๆ ละ 384 โนนด รวมทั้งสิ้นจำนวน 158,976 โนนด แต่ละโนนดมีชิป Fujitsu A64FX 1 ชิปภายในประกอบด้วยซีพียู

ARM จำนวน 48 แกนประมวลผล และหน่วยความจำหลัก (RAM) 32 กิกะไบต์ (GiB) คิดเป็นจำนวนแกนประมวลผลรวมทั้งหมด 7,630,848 แกนประมวลผล รวมหน่วยความจำหลักหรือหน่วยความจำภายในรวมทั้งหมด 4968 เพต้า (10¹⁵) ไป忒 รายละเอียดเพิ่มเติมที่ [wikipedia](#)



รูปที่ 8.4: ภาพถ่ายโดยและบล็อกไดอะแกรมของชิป Fujitsu A64FX ประกอบด้วยชีพีเมีย ARM จำนวน 48 แกนประมวลผล ที่มา: [fujitsu.com](#)

รูปที่ 8.4 แสดงให้เห็นโครงสร้างภายในของชีพีเมียแม่นีคอร์ชื่อ Fujitsu A64FX ประกอบด้วยชีพีเมีย ARM จำนวน 4 ชุด ๆ ละ 12 แกนประมวลผล หรือเท่ากับ 48 แกนประมวลผล และทำงานที่ความถี่คลือก 2.2 กิกะ เฮิรตซ์ ชีพีเมียแต่ละแกนประมวลผลรองรับคำสั่งและสเซมบลี ARM เวอร์ชัน v8.2-A ความยาว 64 บิต รองรับเทคโนโลยี SVE (Scalable Vector Extension) โดยบริษัท Fujitsu และบริษัท ARM ออกแบบร่วมกัน ผู้อ่านสามารถค้นควารายละเอียดเพิ่มเติมที่ [wikipedia](#) ชิปชีพีเมีย Fujitsu A64FX นี้ผลิตโดยบริษัท TSMC (Taiwan Semiconductor Manufacturing Company) ด้วยเทคโนโลยี 7 นาโนเมตร ผู้อ่านสามารถค้นควารายละเอียดเกี่ยวกับชิปเพิ่มเติมที่ลิงก์ต่อไปนี้ [github.com](#)

ชีพีเมียในแต่ละโหนด เชื่อมต่อกันด้วยเครือข่ายบนชิป (Network on Chip: NoC) ชนิดบัสวงแหวน (Ring Bus) และเชื่อมต่อกับหน่วยความจำภายในของชิป ประสาทสิทธิภาพสูงด้วยเทคโนโลยี HBM2 (High Bandwidth Memory 2) ความจุรวม 32 กิกะไบต์ (GiB) รายละเอียดเกี่ยวกับ HBM2 เพิ่มเติมที่ [wikipedia](#) แต่ละโหนดจัดเป็นเครื่องคอมพิวเตอร์แบบขนาดชนิดแม่นีคอร์และใช้หน่วยความจำร่วมกัน (SMP) และเชื่อมต่อกันระหว่างกันด้วยเครือข่ายความเร็วสูงชื่อ ToFuD Interface ผ่านอินเตอร์เฟส PCIe (Peripheral Component Interconnect Express) ของแต่ละโหนด เราจึงสรุปได้ว่า ชูเปอร์คอมพิวเตอร์ Fugaku เป็นมัลติคอมพิวเตอร์ที่มีหน่วยความจำหลักอิสระจากกัน ชนิดคลัสเตอร์ขนาดใหญ่มาก

ซอฟต์แวร์ระบบของเครื่องชูเปอร์คอมพิวเตอร์ Fugaku เป็นระบบปฏิบัติการลินุกซ์ Red Hat Enterprise Linux 8 สำหรับเครื่องควบคุมโหนดต่าง ๆ และมีเครื่องเนลเสริมในแต่ละโหนด ชื่อ McKernel เพื่อทำหน้าที่ควบคุมการคำนวณแบบขนาน โปรแกรมเมอร์สามารถพัฒนาโปรแกรมแบบขนาน โดยใช้คอมไพเลอร์ภาษา C/C++ และไลบรารี OpenMP ซึ่งจะได้กล่าวถึงในหัวข้อที่ 8.3 และประสานงานระหว่างโหนดด้วยไลบรารี MPI (Message Passing Interface) ของ Fujitsu ซึ่งพัฒนาเพิ่มเติมจากไลบรารี OpenMPI

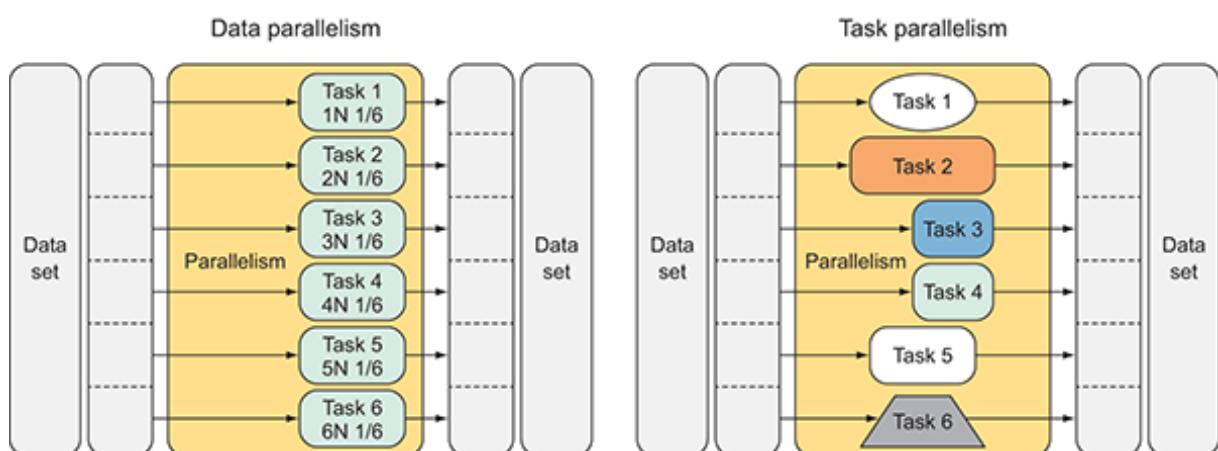
ที่มา: fujitsu.com เครื่องซูเปอร์คอมพิวเตอร์ Fujaku นี้สามารถรองรับภาษาอื่น ๆ เช่น ภาษา Java และภาษา Python นอกจากภาษา C/C++ ที่กล่าวมาข้างต้น

8.2 ความขนาด (Parallelism)

การคำนวณแบบขนาดตั้งอยู่บนพื้นฐานของฮาร์ดแวร์และความขนาด (Parallelism) ของปัญหา การคำนวณแบบขนาด ประกอบด้วยองค์ประกอบสำคัญ 2 ส่วนคือ

- อัลกอริธึมแบบขนาด (Parallel Algorithm) สำหรับแก้ปัญหาโจทย์ต่าง ๆ ที่สามารถใช้ประโยชน์จากความขนาดของปัญหาหรือโจทย์นั้น ๆ
- เครื่องคอมพิวเตอร์แบบขนาด (Parallel Computer) ชนิดมัลติคอร์และชนิดแม่นิคอร์ซึ่งใช้หน่วยความจำภายในร่วมกันมีประสิทธิภาพสูงขึ้นและราคาต่ำลง การพัฒนาโปรแกรมแบบขนาดบนเครื่องคอมพิวเตอร์ชนิดนี้อาศัยการพัฒนาโปรแกรมแบบมัลติเฟรด (Multithread Programming) ด้วยไลบรารีมาตรฐาน เช่น [Pthread](#) และ [OpenMP](#)

เมื่อนำข้อมูล มหัศจรรย์ (Big Data) มาประมวลผล ด้วย อัลกอริธึมแบบขนาด ซึ่ง โปรแกรมเมอร์ สามารถพัฒนาให้ทำงานแบบมัลติเฟรด ดังที่กล่าวไปแล้ว เพื่อรับบน เครื่องคอมพิวเตอร์ ชนิดมัลติคอร์ หรือ ชนิดแม่นิคอร์ และสูงขึ้นไปถึงระดับซูเปอร์คอมพิวเตอร์ จะทำให้อัลกอริธึมแบบขนาดนั้นทำงานได้เร็วขึ้นกว่า อัลกอริธึมแบบอนุกรม ความขนาดจะแบ่งเป็น 2 ชนิดหลัก ๆ คือ ความขนาดของข้อมูล (Data Parallelism) ที่มา: [Hillis and Steele \(1986\); Patterson and Hennessy \(2016\)](#) และความขนาดของงานย่อย (Task Parallelism) ที่มา: [Patterson and Hennessy \(2016\)](#)



รูปที่ 8.5: การคำนวณแบบขนาดโดยอาศัยความขนาดของข้อมูล (Data Parallelism) และความขนาดของงานย่อย (Task Parallelism) ที่มา: manning.com

8.2.1 ความขนาดของข้อมูล (Data Parallelism)

ความขนาดของข้อมูลนิยมใช้ในอัลกอริธึมต่าง ๆ เช่น การประมวลผลภาพ วิดีโอ หรือข้อมูลขนาดใหญ่มาก การเรียงข้อมูล การสืบค้นข้อมูล การจัดทำดัชนีข้อมูล เป็นต้น ในรูปที่ 8.5 ด้านซ้าย อัลกอริธึมสามารถแบ่งข้อมูลจำนวน $6N$ จะแบ่งออกเป็น 6 ส่วนเท่า ๆ กัน และมอบหมายให้กับแต่ละเครื่องรับภาระงานจำนวน N หน่วย รวมทั้งหมด 6 เครื่องเพื่อสั่งงานซีพียูจำนวนมากกว่าหรือเท่ากับ 6 แกนประมวลผลพร้อม ๆ กัน และเกิดประสิทธิภาพสูงสุด การทดลองที่ 13 ภาคผนวก M จะนิยามการวัดประสิทธิภาพของ การคำนวณแบบขนาน ความขนาดของข้อมูลเหมาะสมสำหรับการคำนวณแบบขนานด้วยจีพียู ซีพียู อาร์เรย์ (Processor Array) และฮาร์ดแวร์ชนิด FPGA ซึ่งแตกต่างกันตามโจทย์หรือปัญหาและอัลกอริธึมสำหรับแก้ปัญหานั้น ๆ

8.2.2 ความขนาดของงานย่อย (Task Parallelism)

หลักการพื้นฐานของการพัฒนาซอฟต์แวร์ คือ การแบ่งงานหลักในรูปที่ 8.5 ด้านขวา ที่มีความซับซ้อนมากให้เป็นงานย่อยหรือทาสก์ (Task) จำนวนหนึ่งในรูปที่ 8.5 ด้านซ้าย จากฐานะที่มีความสามารถแบ่งเป็นงานย่อย คือ ทาสก์ 1 (Task 1) จนถึง ทาสก์ 6 (Task 6) ที่ทำงานต่างกันขึ้นอยู่กับรายละเอียด แล้วจึงมอบหมายงานย่อยแต่ละงานให้กับเครื่อง 1 เครื่องเพื่อรันงาน โปรดสังเกตสัญลักษณ์ที่แตกต่างกันในรูป ซึ่งอาจช่วยทำให้การทำงานหลักโดยภาพรวมใช้เวลาประมวลผลน้อยลง นั่นคือ ประสิทธิภาพสูงขึ้น

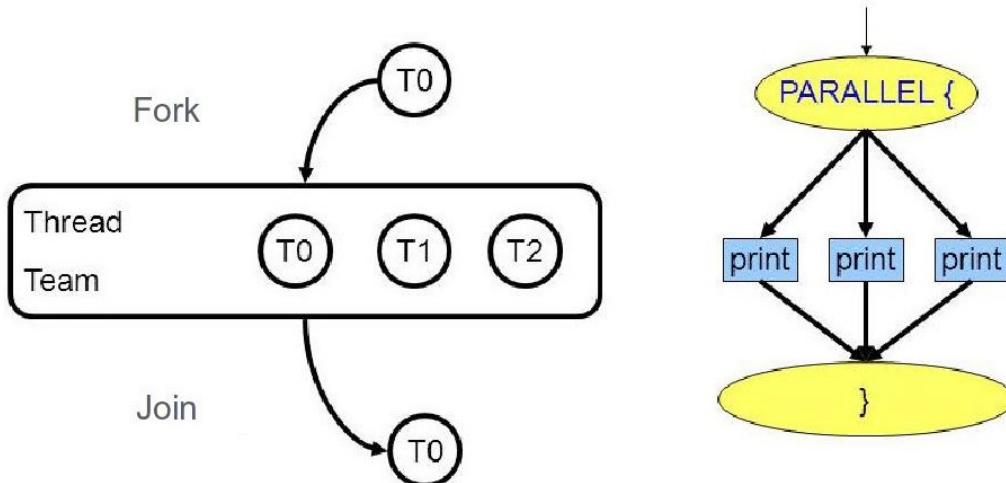
8.2.3 ความขนาดแบบผสม (Hybrid Parallelism)

องค์ประกอบสำคัญในทางปฏิบัติของการพัฒนาโปรแกรมแบบขนาน ด้านฮาร์ดแวร์ คือ จำนวนแกนประมวลผล สถาปัตยกรรมคำสั่ง โครงสร้างของซีพียูโดยเฉพาะ ความจุ/ขนาดและลำดับชั้นหน่วยความจำ ที่มีระยะเวลาเข้าถึง (Access Time) ที่แตกต่างกัน เช่น แคชลำดับต่าง ๆ หน่วยความจำภายใน รวมไปถึงอุปกรณ์เก็บรักษาข้อมูล องค์ประกอบด้านซอฟต์แวร์ ได้แก่ ระบบปฏิบัติการ ภาษาคอมพิวเตอร์ และไลบรารี รวมไปถึงชนิดและขนาดของข้อมูล ดังนั้น การคำนวณแบบขนานจึงมีการผสมผสานระหว่างความขนาดทั้งสองชนิด เนื่องจากอัลกอริธึมแต่ละตัวมีลักษณะพิเศษเฉพาะตัว ยกตัวอย่าง เช่น งานย่อยแต่ละงานมีการคุณ/การบวกເວກເຕອර์ หรือ การคุณ/การบวกເມທິກົກ໌ ซึ่งนิยมใช้ความขนาดของข้อมูลเนื่องจากข้อมูลมีการแบ่งแยกผลลัพธ์ที่ชัดเจน

8.3 การพัฒนาอัลกอริธึมแบบมัลติเรลดและแบบขนานด้วยไลบรารี OpenMP

การพัฒนาอัลกอริธึมแบบขนานในปัจจุบันมีความสำคัญมากขึ้น เนื่องจากอุปกรณ์คอมพิวเตอร์มีชีพียุคใหม่มัลติคอร์ และชนิดแม่นิคอร์เพิ่มมากขึ้นเรื่อยๆ ตามกรณีศึกษาที่กล่าวไปแล้วในหัวข้อที่ 8.1 ดังนั้นโปรแกรมเมอร์สามารถแปลงฟังก์ชันที่ถูกเรียกใช้บ่อยๆ ให้กลายเป็นเรลดของการทำงานบนคอร์พร้อมๆ กันได้ โดยมีระบบปฏิบัติการที่รองรับการทำงานแบบมัลติเรลด ภาษาที่รองรับการทำงานแบบมัลติเรลด ได้แก่ ภาษา C/C++ ใช้ไลบรารี (Library) มาช่วยเสริมจึงรองรับการทำงานแบบมัลติเรลด เช่น PThread และ OpenMP เป็นต้น ส่วนภาษา JAVA เป็นภาษาที่ถูกออกแบบให้รองรับการทำงานมัลติเรลดโดยสรุป โปรแกรมต่างๆ ที่มีอัลกอริธึมแบบขนานสามารถทำงานอยู่บนชีพียุคใหม่มัลติคอร์ จะต้องสร้างจำนวนเรลดให้มากกว่าหรือเท่ากับจำนวนคอร์ที่มี เพื่อกระจายเรลดต่างๆ ไปยังทุกๆ แกนประมวลผลให้ทำงานพร้อมๆ กัน

```
#pragma omp parallel
{
    printf("Hello world %d\n", omp_get_thread_num());
}
```



รูปที่ 8.6: ตัวอย่างซอฟต์แวร์สโค้ด โฟล์กการทำงานแบบขนาน และเรลดประกอบด้วย T0, T1 และ T2 พิมพ์ข้อความ Hello World ตามด้วยหมายเลขเรลดประจำตัว โดยใช้ไลบรารี OpenMP แก้ไขจากที่มา: slideplayer.com

ไลบรารี openMP มี ประโยชน์ โครงสร้าง (Construct) หลาย ชนิด ที่ เสริม การ พัฒนา โปรแกรม คอมพิวเตอร์ ภาษา C/C++ และ ภาษา Fortran ให้ สามารถ ทำงาน แบบ มัลติ เรลด ได้ โดย องค์กร openmp.org ได้ กำหนด เป็น มาตรฐาน และ พัฒนา เป็น เวอร์ชันต่างๆ เรื่อยมา เพื่อ ให้ คอมไพล์ แต่ละ ตัว ทำ หน้าที่ แปลง ซอฟต์แวร์ ตาม มาตรฐาน ซึ่ง ต่อ ราย เลื่อน นี้ ใช้ คอมไابل์ GCC เป็น เครื่อง มือ หลัก

```
#pragma omp parallel ...
{
// Parallel Region
}
```

ประโยค #pragma omp parallel ... เป็นประโยคที่ใช้กำหนดบริเวณที่เรียกว่า **Parallel Region** หมายเหตุ ... หมายถึง โครงสร้าง (Construct) ชนิดต่าง ๆ ที่กำหนดในมาตรฐาน OpenMP ซึ่งจะยกตัวอย่างต่อไป

ทุกโปรแกรมที่คอมไพล์และลิงก์ด้วยไลบรารี OpenMP จะเริ่มต้นรันแบบอนุกรม (Serial) ด้วยเหตุผล หรือ (**Master Thread**) ก่อนเสมอ เมื่อเหตุผลนี้แล้ว ห้ามรันมาถึงโปรแกรมบริเวณ Parallel Region เหตุผลสามารถสร้างเหตุผลนี้ได้โดยการใช้คำสั่ง **Fork** ในรูปที่ 8.6 เมื่อแต่ละเหตุผลทำงานเสร็จสิ้นแล้วก็จะรวมกับเหตุผลนี้ แล้วดำเนินการ **Join** ให้เหตุผลนี้ดำเนินการต่อแบบอนุกรม จนกว่าเหตุผลนี้จะรันถึงบริเวณ Parallel Region อีก 1 ครั้ง ไลบรารี OpenMP ตั้งชื่อการทำงานลักษณะนี้ว่า **ไม้เดล Fork-Join**

ตามชื่อเรื่องที่ตัวอย่างในรูปที่ 8.6 T0 ทำงานอยู่เพียงเหตุผลเดียวจนถึง Parallel Region T0 จึงสร้าง (**Fork**) เหตุผลนี้ แล้วก็มี 2 เหตุผลเพื่อการประมวลแบบขนานรวม 3 เหตุผล ประกอบด้วย T0, T1 และ T2 เพื่อส่งพิมพ์ข้อความ Hello World ตามด้วยหมายเลขเหตุผลประจำตัว (**Thread ID**) เท่ากับ 0 เสมอ ส่วน T1 และ T2 คือเหตุผลนี้จะทำงานในรูปแบบขนาน แต่ต้องรันถึง Parallel Region จึงจะสามารถอ่านค่าหมายเลขเหตุผลประจำตัวของแต่ละเหตุผลได้จากฟังก์ชัน **omp_get_thread_num()**; ระบบปฏิบัติการจะกระจายเหตุผลนี้ไปรันบนแกนประมวลผลที่ว่างๆ เวลาหนึ่ง เพื่อให้ชีพิญแต่ละแกนประมวลผลนั้นเพิ่มประสิทธิภาพในการทำงานใน Parallel Region มาตรฐานและประมวลผลตามปกติ

ในรูปที่ 8.6 คำสั่ง printf ในพื้นที่ Parallel Region จะพิมพ์ประโยค

```
Hello world 1
Hello world 0
Hello world 2
```

หรือแตกต่างจากนี้ ขึ้นกับว่าเหตุผลใดได้เรียกใช้ฟังก์ชัน printf() ก่อนเหตุผลอื่น ๆ ตัวอย่างนี้แสดงการใช้งานเอตเต็ปต์ด้วยฟังก์ชัน printf ซึ่งชีพิญจำนวนมากกว่า 1 แกนประมวลผลขึ้นไปที่กำลังรันเหตุผลทั้ง 3 เหตุผลนี้ต้องผลัดกันใช้งานอินพุตและเอตเต็ปต์ในกรณีนี้คือ จะแสดงผลตามคำสั่ง printf ทำให้การรันแต่ละรอบไม่เหมือนเดิม เนื่องจากชีพิญทุกแกนใช้อุปกรณ์อินพุต/เอตเต็ปต์ร่วมกัน เมื่อเหตุผลนี้ทำงานเสร็จสิ้น โปรแกรมจะยกเลิกเหตุผลนี้ให้เหลือเพียงเหตุผล T0 เพียงเหตุผลเดียวที่จะทำงานต่อไป เรียกว่า **Join**

ตัวอย่างอัลกอริธึมแบบขนานสำหรับทำความเข้าใจเบื้องต้น คือ การคูณเมทริกซ์ ในหัวข้อที่ 8.3.1 และ การเรียงข้อมูล ในหัวข้อที่ 8.3.2

8.3.1 การคูณเมทริกซ์ (Matrix Multiplication) แบบขาน

การทดลองที่ 13 การพัฒนาโปรแกรมแบบขานด้วย OpenMP ภาคผนวก M การคูณเมทริกซ์ เป็นตัวอย่างการคำนวณแบบขานที่เข้าใจง่ายและนิยมใช้จริงในการคำนวณทางวิทยาศาสตร์และคณิตศาสตร์ทั่วไป เช่นภาษาคำนวณประสิทธิภาพสูง (High Performance Computing) เช่นภาษา R Project และภาษา Matlab ซึ่งรองรับการประมวลผลข้อมูลหรือตัวแปรชนิดเมทริกซ์และเวกเตอร์เป็นต้น การคูณเมทริกซ์แบบอนุกรมสามารถทำงานได้อย่างรวดเร็วหากเมทริกซ์มีขนาดเล็กแต่เมื่อเมทริกซ์มีขนาดใหญ่ขึ้น โปรแกรมจำเป็นต้องอาศัยการคำนวณแบบขานเนื่องจากความซับซ้อนเฉลี่ยของการคูณเมทริกซ์เท่ากับ $O(N^3)$ ตามพีชคณิต Big-O ที่มา: Rosen (2002) ดังนั้น

ตัวอย่างที่ 8.3.1 การคูณเมทริกซ์แบบขานบนบอร์ด Pi3/Pi4 ซึ่งมีซีพียูทั้งหมด 4 แกนประมวลผลใน การทดลองที่ 13 ภาคผนวก M

A, B, C คือ เมทริกซ์จตุรัสมีขนาด $N \times N$ โดยมีวนรอบ for ซ้อนกัน 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

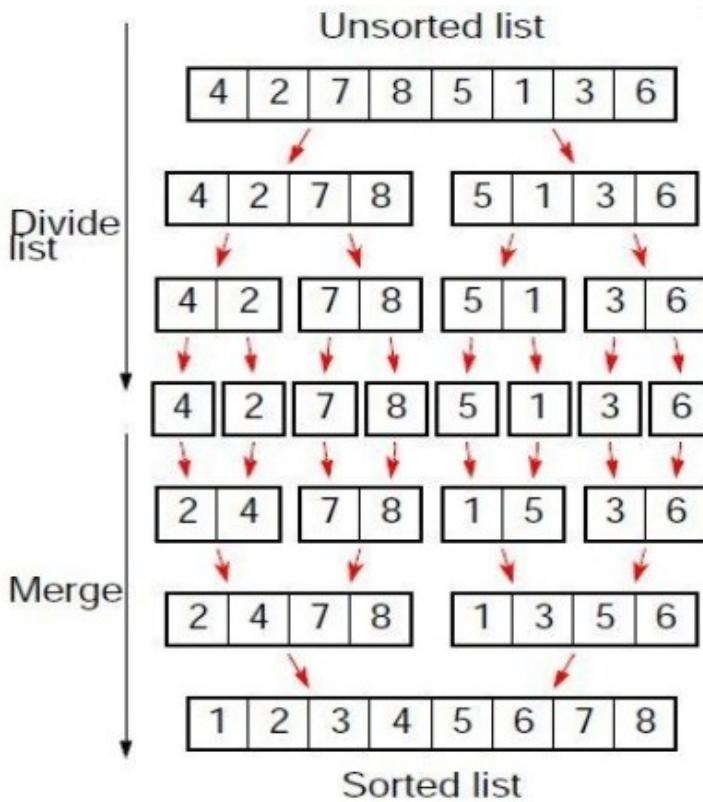
การทดลองจับเวลาเฉพาะการวนรอบนี้เพื่อเปรียบเทียบและคำนวณต่าง ๆ เฮอดหลักจะสร้าง Hera ด้วยที่ประโภคโครงสร้าง **#pragma omp parallel for** ตามจำนวน Hera ที่ผู้ใช้ต้องการ ยกตัวอย่างเช่น 4 Hera นับรวม Hera หลัก ตามหลักการ SMP แต่ละ Hera ซึ่งประจำอยู่ที่ซีพียูแต่ละแกนประมวลผลจะมองเห็นค่าของตัวแปรทุกตัว ยกเว้น j และ k ซึ่งจะอธิบายต่อไป

Hera หลักจะแบ่งข้อมูลตามตัวแปรแล้ว 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 ของตนเองตามประโยชน์ **private(j, k)** ตามลำดับ เพื่อที่จะคำนวณได้อย่างอิสระจาก Hera อื่น ๆ

8.3.2 การเรียงข้อมูล (Sorting) แบบขนาน



รูปที่ 8.7: การทำงานของอัลกอริธึม MergeSort ตามหลักการแบ่งและยึดครองสามารถนำมารัดแปลง เป็นแบบอนุกรมและแบบขนานได้ในตัวอย่างที่ 8.3.2 ที่มา: slideshare.net

การเรียงข้อมูลเป็นอัลกอริธึมพื้นฐานที่ผู้อ่านควรจะได้เรียนในวิชา เช่น วิชา Algorithm Analysis and Design หรือวิชา Data Structure and Algorithm เป็นต้น อัลกอริธึมเรียงข้อมูลเริ่มต้นจากการทำงานแบบอนุกรม และพัฒนาให้เป็นอัลกอริธึมแบบขนานเมื่อ Harvard Werner ร้องรับ 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 เมماะกับข้อมูลที่เป็นเลขจำนวนเต็ม
- การเรียงข้อมูลด้วยหลักการแบ่งและยึดครอง (Divide and Conquer) เช่น MergeSort และ QuickSort มีความซับซ้อนเฉลี่ยเป็น $O(N \log N)$ เมماะกับข้อมูลทุกประเภท

การทำงานแบบขนานของอัลกอริธึมการเรียงข้อมูลนิยมใช้ความขนาดของข้อมูลเป็นหลัก โดยเฉพาะอัลกอริธึมหลักการแบ่งและยึดครอง つまり เล่มนี้ขอใช้ MergeSort เป็นกรณีศึกษา เนื่องจากเป็นอัลกอริธึมที่เข้าใจง่ายกว่า QuickSort และมักใช้การเรียกฟังก์ชันเรียกซ้ำ (Recursive)

ຕ້ອງຢ່າງທີ່ 8.3.2 ຂອຮັສໂຄດ້ ກາຫຍາ C/C++ ຂອງອັດກອ ຮິທີມການເຮືອງຂໍ້ມູນ MergeSort ແບບຂານດ້ວຍໄລບຣາຣີ OpenMP ແກ້ໄຂດັດແປລັງຈາກທີ່ມາ: github.com

ພຶກ່ນໍ້າ main() ສ້າງຂໍ້ມູນຈຳນວນ SIZE ຕົວແບບສຸ່ນແລ້ວເກີບໃນຕົວແປຣອົບເຮົຍ a[] ທີ່ເປັນອົບເຮົຍຂໍ້ມູນຕົ້ນນັບກ່ອນເຮືອງແລະ ລັ້ງການເຮືອງເສົ້າຈີນ ອົບເຮົຍ temp ຄື່ອ ອົບເຮົຍສໍາຫຼັບເກີບພັກຂໍ້ມູນຊ່ວຍກາວຕົວແປຣ SIZE ອີ່ນາດຂອງອົບເຮົຍທີ່ຕ້ອງການເຮືອງຂໍ້ມູນ ແລ້ວຈຶ່ງເຮືອງພຶກ່ນໍ້າ mergesort_parallel_omp() ດ້ວຍຈຳນວນເຮຣດ ເຮີມຕົ້ນໃນຄ່າຕົວ ແປຣ num_threads ມາກກວ່າ ອີ່ນີ້ເຫັນກັບຈຳນວນແກນປະມວລຜລ ຂອງເຄຣືອງ

```

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
}

```

ກາຍໃນພຶກ່ນໍ້າ mergesort_omp() ພາຣາມີເຕອຣ໌ threads ຮັບຄ່າຕົ້ນຈາກຕົວແປຣ num_threads ແປ່ງກາຣທຳການເປັນ 2 ກຣົມີ ອີ່ນີ້

- ກຣົມີ threads > 1

ໜີ້ຕອນການແບ່ງ (Divide) ເຮີມຕົ້ນທີ່ປະໂຍດ ໂຄງສ້າງ #pragma omp parallel sections ທີ່ຈະເກີດ Parallel Region ໜີ້ເຮຣດ ພັກ ຈະສ້າງເຮຣດ ເສົ້າມີມາຮວມ 2 ເຮຣດ ຕາມປະໂຍດ

#pragma omp section ทั้งสอง ประโภค เพื่อ ทำงาน ฟังก์ชัน mergesort_omp() อาร์เรย์ ข้อมูลด้านซ้ายและด้านขวาตามลำดับ การที่ เฮอด (หลัก หรือ เฮอด เสริม) เรียก ฟังก์ชัน แต่ละ รอบ ตัวแปร threads จะมีค่าลดลงเหลือประมาณครึ่งหนึ่งของค่าเดิม และ เฮอด (หลัก หรือ เฮอด เสริม) จะเรียกฟังก์ชัน mergesort_omp() ซ้ำๆ จนค่าของตัวแปร threads = 1

- กรณี threads = 1

เฮอด (เฮอด หลัก หรือ เฮอด เสริม) 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() แบ่ง อาร์เรย์ ที่รับมาที่ลักษณะ เช่นเดียวกับ mergesort_omp แต่จะเรียกใช้ ฟังก์ชัน mergesort_serial() ซ้ำๆ อีก เพื่อเรียงข้อมูลด้านซ้ายก่อน จน ตัวแปร size = 2 แล้ว จึง กระทำ กับ ข้อมูล ด้านขวา หาก คาด แผนภูมิ การ ทำงาน ของ ฟังก์ชัน mergesort_serial() จะ มี ลักษณะ เหมือน แผนภูมิ ต้นไม้ (Tree) ใน รูปที่ 8.7 ลักษณะ การ ทำงาน แบบ เรียก ซ้ำๆ ของ ฟังก์ชัน mergesort_serial() เปรียบเหมือน การ เดินทาง บน แผนภูมิ ต้นไม้ แบบ เชิงลึก ก่อน 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) เริ่มต้นเมื่อ พังก์ชัน `merge()` ทำหน้าที่ ประสานข้อมูลขนาดเล็ก ๆ ที่ พังก์ชัน `mergesort_serial()` ได้แบ่งไว้ก่อนหน้า และจึงเริ่นย้อนกลับไปประสาน (Merge) ข้อมูลขนาดใหญ่ขึ้น ๆ ที่ พังก์ชัน `mergesort_omp()` ได้แบ่งไว้ หลังจากนั้นจึงลำเนาข้อมูลที่เรียงแล้วจาก `temp` ไปยัง อาร์เรย์ `a` ด้วยพังก์ชัน `memcpy()` เป็นจำนวนไปต์เท่ากับ $\text{size} \times \text{sizeof(int)}$ (ขนาดของตัวแปร `int` เท่ากับ 4 ไบต์)

งานวิจัยที่เกี่ยวข้อง การเรียงข้อมูลแบบขนาดของอัลกอริธึม QuickSort โดยผู้เขียน ได้แก่

- Parallel Partition and Merge QuickSort หรือ **PPMQSort** ที่มา: [Ranokpanuwat and Kittitornkun \(2016\)](#) ซึ่งพัฒนาด้วยภาษา C มีความซับซ้อนเชิงเวลา (Run Time Complexity) เท่ากับ $O(\frac{N}{c} \log \frac{N}{2c} + N)$ โดยที่ N และ c คือ ขนาดของอาร์เรย์ข้อมูล และจำนวนแกนประมวลผล ตาม ลำดับ และ
- MultiStack Parallel Partition Sorting หรือ **MSPSort** ที่มา: [Rattanantranurak and Kittitornkun \(2020\)](#) ซึ่งพัฒนาด้วยภาษา C++ และผู้เขียนทำการวิจัยและพัฒนาเพิ่มเติม เพื่อให้รองรับ การทำงานบนซีพียูมัลติคอร์ ARM Cortex A72 บนบอร์ด Raspberry Pi4 ที่มา: [Rattanantranurak and Kittitornkun \(2021\)](#)

8.4 สรุปท้ายบท

การคำนวณแบบอนุกรมมาจากการขบวนการคิดพื้นฐานของมนุษย์แต่ปัจจุบันด้านต่าง ๆ ที่ซับซ้อนและปริมาณข้อมูลที่เพิ่มมากขึ้นเรื่อยๆ จึงต้องอาศัยคอมพิวเตอร์ช่วยคำนวณ ทำให้วิวัฒนาการของคอมพิวเตอร์ไม่เงยมาเพื่อรับการคำนวณแบบขนานมากขึ้น คอมพิวเตอร์แบบขนานมี 2 ชนิดแบ่งตามการใช้งานหน่วยความจำหลัก คือ คอมพิวเตอร์แบบขนานชนิดมัลติคอร์และชนิดแมนีคอร์ซึ่งใช้หน่วยความจำภายในร่วมกัน (Shared Memory) และคอมพิวเตอร์แบบขนานชนิดมัลติคอมพิวเตอร์มีหน่วยความจำอิสระจากกัน ประกอบด้วยเครื่องคอมพิวเตอร์ชนิดมัลติคอร์และแมนีคอร์จำนวนมากเข้ามาร่วมกันด้วยเครือข่ายความเร็วสูง การสั่งงานเครื่องคอมพิวเตอร์ทั้งสองชนิดนี้ ด้วยภาษาคอมพิวเตอร์ระดับสูง เช่น C/C++, Java, Python เป็นต้น ต้องเปลี่ยนขบวนการคิดแบบอนุกรมเป็นแบบขนานและใช้ประโยชน์จากความขนาดของข้อมูล ความขนาดของงานย่อย และความขนาดแบบผสม พัฒนาร่วมกับไลบรารี OpenMP และไลบรารี MPI (Message Passing Interface) ซึ่งเป็นที่ยอมรับและนิยมทั่วโลกรวมถึงญี่ปุ่นเครื่องคอมพิวเตอร์ Fugaku

8.5 คำถามท้ายบท

1. จงเปรียบเทียบ คำนวณแบบขนาน กับ การทำการบ้านจำนวนหลายข้อ โดยนักศึกษาหลายคนช่วยกันทำโดย
 - 1 คนรับผิดชอบ 1 ข้อ แล้วนำคำตอบมารวมกัน
 - 2 คนช่วยกันทำการบ้านทีละข้อ เพื่อหาคำตอบที่ดีที่สุด แล้วนำคำตอบมารวมกัน
 - ทุกคนช่วยกันทำการบ้านทีละข้อ เพื่อหาคำตอบที่ดีที่สุด
 วิธีไหนจะใช้เวลาต่างกันเท่าไหร่ และมีข้อดี ข้อเสียต่างกันอย่างไร
2. จงบอกความสัมพันธ์ระหว่างจำนวนเรตต์ และจำนวนแกนประมวลผล
3. จงอธิบายวิธีการแบ่งงาน (Share Work) ของไลบรารี OpenMP
4. จงค้นคว้า [กฎของ Amdahl](#) ในอินเทอร์เน็ต เพื่อนำมาประยุกต์ใช้กับการคำนวณค่า $Speedup(n)$ ของการคูณเมทริกซ์แบบขนาน
5. จงค้นคว้าว่า คอมพิวเตอร์ชนิด Shared Memory Multiprocessor มีประเภทอยู่ในทั้งหมด ในอนุกรมวิธานของ [Flynn](#) (Flynn's Taxonomy)

ภาคผนวก (Appendices)

ภาคผนวก A

การทดลองที่ 1 ข้อมูลและคณิตศาสตร์ในคอมพิวเตอร์

การทดลองนี้เป็นการทบทวนความเข้าใจ และแบบฝึกหัดเสริมของเนื้อหาในบทที่ 2 เนื่องจากจำนวนบิตข้อมูลที่ยาวขึ้นจำเป็นต้องใช้โปรแกรมคอมพิวเตอร์ช่วยคำนวณแทน โดยมีวัตถุประสงค์ ดังต่อไปนี้

- เพื่อให้เข้าใจ การแปลง และคณิตศาสตร์สำหรับเลขจำนวนเต็มฐานสองชนิด ไม่มีเครื่องหมาย และมีเครื่องหมายแบบ 2's Complement
- เพื่อให้เข้าใจการแปลง และคณิตศาสตร์สำหรับเลขทศนิยมฐานสองมาตรฐาน IEEE754 ชนิด Single Precision
- เพื่อให้เข้าใจรหัส ASCII และ Unicode สำหรับข้อมูลตัวอักษร

นอกจากเนื้อหาในบทที่ 2 แล้ว ผู้อ่านสามารถศึกษาเว็บเพจเพิ่มเติม เพื่อทำความเข้าใจอย่างลึกซึ้งได้แก่

- 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 การทดลองแปลงเลขจำนวนเต็มฐานสอง

เนื่องจากการแปลงเลขฐานสิบเป็นฐานสองชนิดไม่มีเครื่องหมาย (unsigned) ผู้อ่านสามารถใช้เครื่องคิดเลขทางวิทยาศาสตร์ทั่วไป ดังนั้น การทดลองนี้จะเน้นที่การแปลงเป็นเลขจำนวนเต็มฐานสองชนิดมีเครื่องหมายแบบ 2's Complement สอดคล้องกับเนื้อหาในหัวข้อที่ 2.2 โดยผ่านเว็บเบราว์เซอร์ที่ผู้อ่านくなดคลิกที่ชื่อลิงก์ต่อไปนี้ <https://www.binaryconvert.com/> ขอให้ผู้อ่านปฏิบัติตามการทดลองดังนี้

TYPE	BITS	MINIMUM	MAXIMUM	DECIMAL FORMAT
Unsigned char	8	0	255	Integer
Signed char	8	-128	127	Integer
Unsigned short	16	0	65535	Integer
Signed short	16	-32768	32767	Integer
Unsigned int	32	0	4294967295	Integer

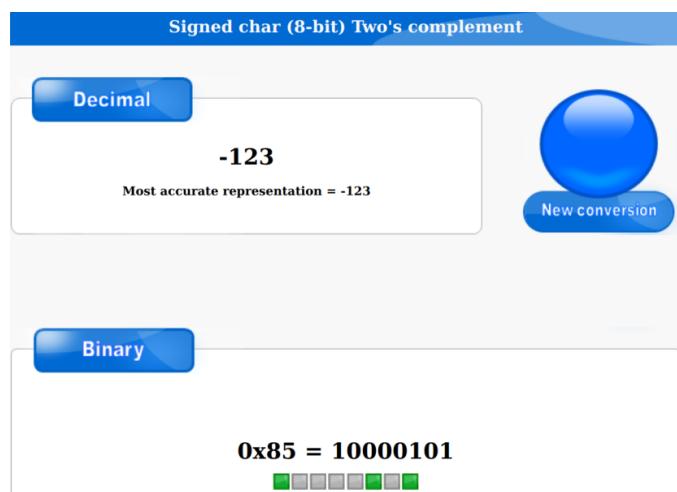
รูปที่ A.1: หน้าเว็บสำหรับแปลงเลขจำนวนเต็มฐานสองเป็นฐานสิบหรือฐานสิบเป็นฐานสองหลายชนิด

- คลิกที่หัวข้อ Signed Char เพื่อทดลองการแปลงเลขจำนวนเต็มมีเครื่องหมายชนิด 2's Complement ขนาด 8 บิต
- กรอกเลข -123 ลงในกล่องข้อความ Decimal เพื่อให้โปรแกรมแปลงเลขจำนวนเต็ม -123 เป็นเลขฐานสองมีเครื่องหมายชนิด 2's Complement ดังรูปที่ A.2



รูปที่ A.2: กรอกเลข -123 ลงในกล่องข้อความ Decimal เพื่อให้โปรแกรมแปลงเลขจำนวนเต็ม -123 เป็นเลขจำนวนเต็มฐานสองมีเครื่องหมายชนิด 2's Complement

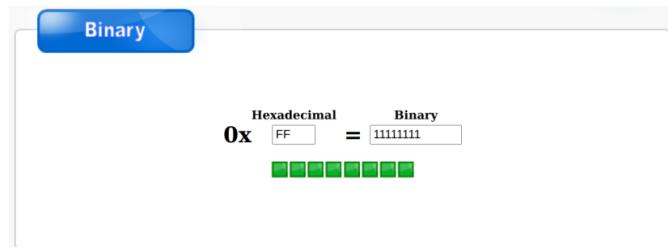
กดปุ่ม Convert to binary เพื่อดำเนินการ บันทึกผลลัพธ์ที่ได้จากการแปลงดังต่อไปนี้



รูปที่ A.3: ผลลัพธ์การแปลงเลข -123 เป็นเลขจำนวนเต็มฐานสองมีเครื่องหมายชนิด 2's Complement

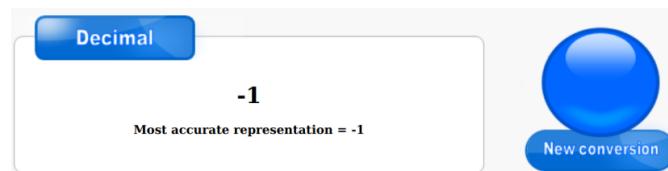
- Binary (2's Complement) _ _ _ _ _ _ _ _
- Hexadecimal (0x) _ _
- แสดงวิธีทำตามสมการที่ (2.20) ที่ $n=8$ บิต เพื่อแปลงเลขให้ตรงตามรูป

3. กรอกเลขฐานสองมีเครื่องหมายชนิด 2's Complement 11111111 ขนาด 8 บิตลงในกล่องข้อความ Binary เพื่อให้โปรแกรมแปลงเลขจำนวนเต็มฐานสิบ ดังรูปที่



รูปที่ A.4: การแปลงเลขฐานสองมีเครื่องหมายชนิด 2's Complement 11111111 หรือเท่ากับฐานสิบ 8 บิต 0xFF

กดปุ่ม Convert to decimal ทางด้านขวาเพื่อดำเนินการ อ่านค่าผลลัพธ์ที่ได้จากการแปลงดังต่อไปนี้



รูปที่ A.5: ผลลัพธ์การแปลงเลขฐานสองมีเครื่องหมายชนิด 2's Complement 11111111 หรือเท่ากับฐานสิบ 8 บิต 0xFF

4. กดปุ่ม Signed short บนเมนูด้านบนสุด เพื่อเปลี่ยนความยาวเป็น 16 บิต กรอกเลข -123 ลงในกล่องข้อความ Decimal กดปุ่ม Convert to binary เพื่อดำเนินการ บันทึกผลลัพธ์ที่ได้จากการแปลงดังต่อไปนี้

- Binary (2's Complement) _____
- Hexadecimal (0x) _____
- แสดงวิธีทำตามสมการที่ (2.20) ที่ $n=16$ บิตเพื่อแปลงเลขให้ตรงตามรูป

5. กดปุ่ม Signed int บนเมนูด้านบนสุด เพื่อเปลี่ยนความยาวเป็น 32 บิต กรอกเลข -123 ลงในกล่องข้อความ Decimal กดปุ่ม Convert to binary เพื่อดำเนินการ บันทึกผลลัพธ์ที่ได้จากการแปลงดังต่อไปนี้

- Binary _____
- Hexadecimal (0x) _____
- แสดงวิธีทำตามสมการที่ (2.20) ที่ $n=32$ บิตเพื่อแปลงเลขให้ตรงตามรูป

A.1.2 คณิตศาสตร์เลขจำนวนเต็มฐานสองขนาด 16 บิต

ผู้อ่านสามารถศึกษาคณิตศาสตร์เลขจำนวนเต็มฐานสองชนิดไม่มีเครื่องหมาย (unsigned) และชนิดมีเครื่องหมาย (Signed 2's Complement) ไปพร้อมๆ กัน และสอดคล้องกับเนื้อหาในหัวข้อที่ 2.2 โดยผ่านเว็บเบราว์เซอร์ที่ซื่อสัตย์ต่อไปนี้ <http://www.ee.unb.ca/cgi-bin/tervo/alu.pl> ซึ่งเป็น ALU (Arithmetic Logic Unit) Simulator ในวิชา EE3221 Digital Systems II ของภาควิชา Department of Electrical and Computer Engineering มหาวิทยาลัย University of New Brunswick ประเทศ Canada

รูปที่ A.6: หน้าเว็บสำหรับศึกษาการทำนายจำนวนจรคณิตศาสตร์เลขจำนวนเต็มฐานสอง

ขอให้ผู้อ่านปฏิบัติตามการทดลอง ดังนี้

1. แปลงเลขฐานสิบ 32765 และ -32768 เป็นฐานสองชนิด 2's Complement ขนาด 16 บิต ด้วยเว็บแปลงเลขที่ใช้ในการทดลองที่แล้ว
2. ก็อปปีเลขฐานสองที่ได้ไปวาง (Paste) จากเว็บแปลงเลขลงในช่อง RB (Reigster B) และ RA (Register A) ตามลำดับ เลือก ADD และจึงกดปุ่ม Compute เพื่อบวกเลข RB และ RA เข้าด้วยกัน ดังรูป (ไม่ต้องกังวลซึ่งว่างหายในเลข 16 บิต)

RB
 RA
 ADD SUB AND OR XOR

Result: ADD RC, RB, RA

RB	7FFD	0111111111111101	FLAGS
RA	8000	1000000000000000	V C N Z
		-----	-----
RC	FFFD	1111111111111101	0 0 1 0

Integer Interpretation

REGS	UNSIGNED	SIGNED
RB:	(32765)	(+32765)
RA:	+ (32768)	+ (-32768)

RC:	(65533)	(-3)
CHECK:	OK (C=0)	OK (V=0)

รูปที่ A.7: ผลลัพธ์ ADD RC, RB, RA หรือ $RC = RB + RA$ โดย $RB=0111111111111101_2$ และ $RA=1000000000000000_2$ ความยาว 16 บิต

3. อธิบายผลการทดลอง RC (Register C) และ Flag: VCNZ ที่ได้ โดยมองตัวเลข RA, RB และ $RC = RB + RA$ เป็นเลขจำนวนเต็มฐานสองเป็นสองกรณี คือ ฐานสิบหก (คอลัมน์ด้านซ้าย) และ ฐานสอง (คอลัมน์ด้านขวา)

Results: ADD RC, RB, RA หรือ $RC = RB + RA$

RB

RA

RC

V (Overflow)=

C (Carry)=

N (Negative)=

Z (Zero)=

4. อธิบายผลการทดลองโดยมองตัวเลข RA, RB และ $RC = RB + RA$ เป็นเลขจำนวนเต็มฐานสองเป็นสองกรณี คือ unsigned (คอลัมน์ด้านซ้าย) และ signed 2's Complement (คอลัมน์ด้านขวา)

Integer Interpretation RB:

RA:

RC:

CHECK:

เพราะเหตุได้จึง OK

A.1.3 คณิตศาสตร์เลขจำนวนเต็มฐานสองขนาด 8 บิต

ผู้อ่านสามารถใช้ความเข้าใจจากการบวกเลขในหัวข้อที่แล้วมาทำการบวกเลขด้วยตนเอง โดยใช้เลขจำนวนเต็มฐานสองขนาด 8 บิตแทน

- กรอก เลข ที่ได้ จาก การ แปลง ฐาน สิบ เป็น ฐาน สอง ลง ใน ช่อง ว่าง ที่ จัด ไว้ แสดง วิธี ทำการ บวก
เลข จำนวนเต็ม ฐาน สอง มี เครื่องหมาย ชนิด 2's Complement ขนาด 8 บิต และ คำนวณ ค่า โวเวอร์โฟล์ว
 V

ซอฟต์แวร์สามารถนำผลลัพธ์ Z ไปใช้งานต่อได้หรือไม่ เพราะเหตุใด

2. กรอกเลขที่ได้จากการแปลงลงในช่องว่างที่จัดไว้ แสดงวิธีทำการบวกเลขจำนวนเต็มฐานสองมีเครื่องหมายชนิด 2's Complement ขนาด 8 บิตและคำนวณค่าโอล์ฟล์วี

ซอฟต์แวร์สามารถนำผลลัพธ์ Z ไปใช้งานต่อได้หรือไม่ เพราะเหตุใด

3. กรอกเลขที่ได้จากการแปลงลงในช่องว่างที่จัดไว้ แสดงวิธีทำการบวกเลขจำนวนเต็มฐานสองมีเครื่องหมายชนิด 2's Complement ขนาด 8 บิตและคำนวณค่าโอล์ฟล์วี

ซอฟต์แวร์สามารถนำผลลัพธ์ Z ไปใช้งานต่อได้หรือไม่ เพราะเหตุใด

4. กรอกเลขที่ได้จากการแปลงลงในช่องว่างที่จัดไว้ แสดงวิธีทำการบวกเลขจำนวนเต็มฐานสองมีเครื่องหมายชนิด 2's Complement ขนาด 8 บิตและคำนวณค่าโอล์ฟล์วี

ขอฟ์แวร์สามารถนำผลลัพธ์ Z ไปใช้งานต่อได้หรือไม่ เพราะเหตุใด

A.1.4 กิจกรรมท้ายการทดลอง

จงทำการทดลอง และตอบคำถามต่อไปนี้ โดยแสดงวิธีทำการทดลองตามเนื้อหาในหัวข้อที่ 2.2.2 และตรวจคำตอบตามวิธีทำการทดลองที่ได้ทำไป

- จงแปลงเลขจำนวนเต็มฐานสิบชนิดไม่มีเครื่องหมายต่อไปนี้ให้เป็นเลขจำนวนเต็มฐานสอง 16 บิต และฐานสิบหกจำนวน 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

- จงแปลงเลขจำนวนเต็มฐานสิบ ต่อไปนี้ให้เป็นเลขจำนวนเต็มฐานสอง และฐานสิบหก ชนิด มีเครื่องหมายแบบ 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

- จงบวกเลข 2's Complement ต่อไปนี้แล้วบันทึกผลลัพธ์เป็นฐานสองความยาว 16 บิต ฐานสิบหก ฐานสิบ โอเวอร์โฟล์วหรือไม่ และอธิบายเหตุผลว่าทำไมจึงไม่ตรงกัน

- $1000000000000000 + 0000000000000001$

- ผลลัพธ์ = 2

- ผลลัพธ์ = 16

- ผลลัพธ์ = 10

- โอเวอร์โฟล์วหรือไม่.....

- เหตุผล.....
- $1000000000000000 + 1000000000000000$
 - ผลลัพธ์ = _____²
 - ผลลัพธ์ = _____¹⁶
 - ผลลัพธ์ = _____¹⁰
 - โอเวอร์โฟล์วหรือไม่.....
 - เหตุผล.....
- $1000000000000000 - 0000000000000001$
 - ผลลัพธ์ = _____²
 - ผลลัพธ์ = _____¹⁶
 - ผลลัพธ์ = _____¹⁰
 - โอเวอร์โฟล์วหรือไม่.....
 - เหตุผล.....
- $1000000000000000 - 1000000000000000$
 - ผลลัพธ์ = _____²
 - ผลลัพธ์ = _____¹⁶
 - ผลลัพธ์ = _____¹⁰
 - โอเวอร์โฟล์วหรือไม่.....
 - เหตุผล.....

A.2 การ แปลง และ คณิตศาสตร์ เลข ทศนิยม ฐาน สอง มาตรฐาน IEEE754

การทดลอง เพื่อ ให้ เข้าใจ การ แปลง เลข ทศนิยม ฐาน สิบ ให้ เป็น เลข ฐาน สอง ตาม รูป แบบ และ ฝึก การ คำนวณ โดย ใช้ คณิตศาสตร์ มาตรฐาน IEEE754 Single Precision มี ความ สอดคล้อง กับ เนื้อหา ใน หัวข้อ ที่ 2.6

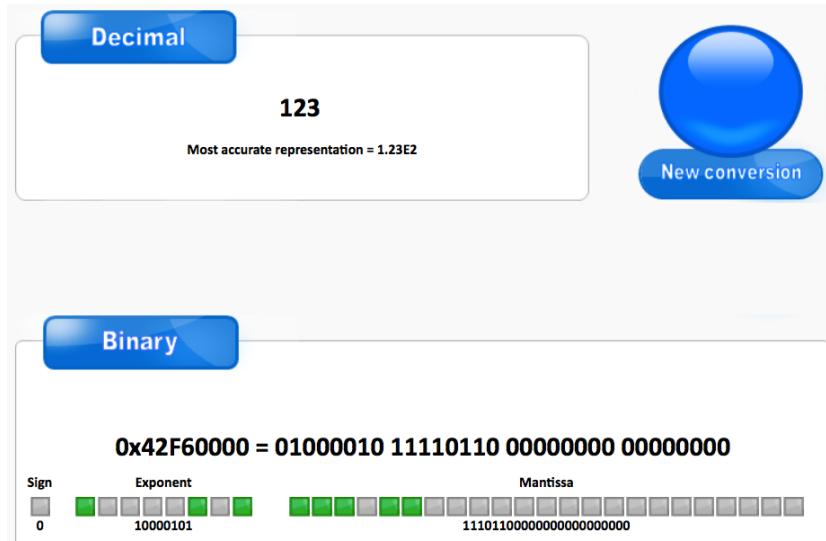
A.2.1 เลขทศนิยมชนิดจุดลอยตัว มาตรฐาน IEEE754 Single-Precision

การ ทดลอง นี้ จะ เน้น ที่ การ แปลง เลข ทศนิยม ฐาน สิบ ให้ เป็น เลข ทศนิยม ฐาน สอง ชนิด จุด ลอยตัว สอดคล้อง กับ เนื้อหา ใน หัวข้อ ที่ 2.6 ใน รูป แบบ Single Precision โดย ผ่าน เว็บ เบราเซอร์ ที่ ผู้ อ่าน คลิก ที่ ลิงก์ ต่อไปนี้

http://www.binaryconvert.com/convert_float.html

เมื่อ เว็บ เพจ ปรากฏขึ้น ขอ ให้ ผู้ อ่าน ปฏิบัติ ตาม การ ทดลอง ดังนี้

- กรอก เลข 123 ลง ใน กล่อง ข้อความ และ กด ปุ่ม Convert to binary ได้ รูปที่ A.8

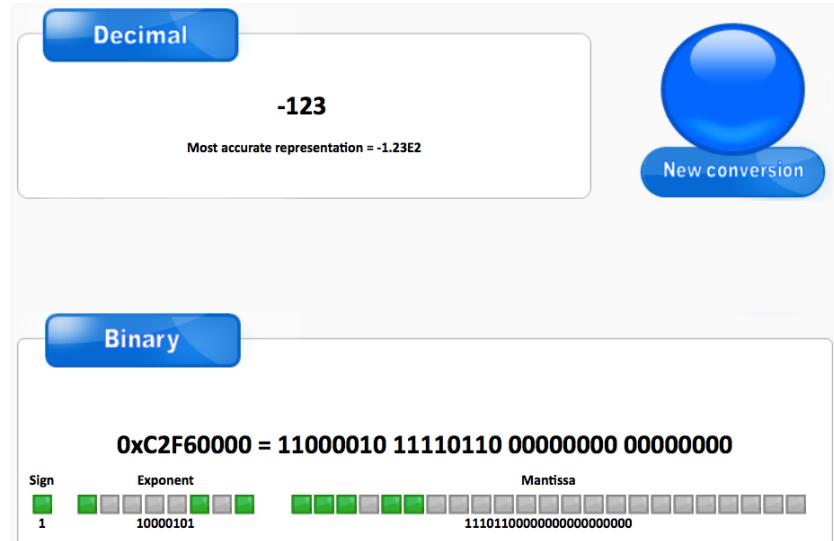


รูปที่ A.8: ผลลัพธ์จากการแปลงเลข 123.0 ให้เป็นเลขทศนิยมฐานสองชนิด Single Precision

การ เรียง ตัว ของ ผลลัพธ์ เลข ฐาน สิบ ให้ ทาง ซ้าย มี อยู่ จาก เลข ฐาน สอง ทาง ขวา มี อยู่ ซึ่ง เกิด จาก บิต ข้อมูล ทั้งหมด 32 บิต ตาม รูป แบบ ของ มาตรฐาน IEEE754 ชนิด Single Precision โปรด สังเกต กล่อง สี เหลือง สี เขียว ตรง กับ บิต ที่ เป็น '1' กล่อง สี เทา ตรง กับ บิต ที่ เป็น '0' 0x หมายถึง เลข ฐาน สิบ หก

แสดง วิธี ทำ ตาม สมการ ที่ (2.76) เพื่อ แปลง เลข ให้ ตรง ตาม รูป

- กรอก เลข -123.0 ลง ใน กล่อง ข้อความ และ กด ปุ่ม Convert to binary ได้ รูปที่ A.9



รูปที่ A.9: ผลลัพธ์จากการแปลงเลข -123.0 ให้เป็นเลขทศนิยมฐานสองตามมาตรฐาน IEEE754 ชนิด Single Precision

โปรดสังเกตตำแหน่งของกล่องสีเหลืองหรือสีเทาที่ตรงกับบิต Sign Exponent และ Mantissa ดังนั้นเราจะเห็นได้ว่าเฉพาะ Sign ที่มีการเปลี่ยนแปลง

แสดงวิธีทำตามสมการที่ (2.76) เพื่อแปลงเลขให้ตรงตามรูป

3. คลิกบนลิงก์นี้ เพื่อทดลองบวกและคูณเลขในรูปแบบ Single Precision ด้วยลิงก์ต่อไปนี้
<http://weitz.de/ieee/> เลื่อนหน้าเว็บลงไปด้านล่างสุด เพื่อค้นหาแถบเมนูตามรูปที่ A.10 แล้วกดเลือกปุ่ม binary32 เพื่อทดลองการบวกและคูณเลข IEEE754 Single Precision

[binary16](#) [binary32](#) [binary64](#) [binary128](#)

รูปที่ A.10: เมนูด้านล่างสุดของหน้าเว็บ เพื่อเลือกเลขทศนิยมฐานสองชนิด IEEE754 Single Precision (Binary32) และ Double Precision (Binary64)

4. เลื่อนหน้าเว็บกลับไปด้านบนสุดเพื่อกรอกเลข -123.0 ลงในกล่องข้อความซ้ายบน และ กรอกเลข 123.0 ลงในกล่องข้อความถัดลงมา แล้วกดปุ่ม + และจะได้ผลลัพธ์ดังรูปต่อไปนี้

IEEE 754 Calculator

(See info at bottom of page.)

Sign	Significand	Exponent
-123.0	1 - 1.921875 0xC2F6000 0b11000010111101100000000000000000	10000101 +6
123.0	0 + 1.921875 0x42F6000 0b01000010111101100000000000000000	10000101 +6
	+ - × /	
0.0	0 + 0.0 0x00000000 0b00000000000000000000000000000000	00000000 +0

รูปที่ A.11: ผลลัพธ์จากการบวกเลข $-123.0 + 123.0$ ให้เป็นเลขทศนิยมฐานสองชนิด IEEE754 Single Precision

จะสังเกตเห็นว่า ผลลัพธ์ที่ได้เรียกว่า True Zero ตามตารางที่ 2.12

5. กดปุ่ม \times (คูณ) และจะได้ผลลัพธ์ของ -123×123 ดังรูปต่อไปนี้

IEEE 754 Calculator

(See info at bottom of page.)

Sign	Significand	Exponent
-123.0	1 - 1.921875 0xC2F6000 0b11000010111101100000000000000000	10000101 +6
123.0	0 + 1.921875 0x42F6000 0b01000010111101100000000000000000	10000101 +6
+	- × /	
-15129.0	1 - 1.8468018 0xC66C6400 0b11000110011011000110010000000000	10001100 +13

รูปที่ A.12: ผลลัพธ์จากการคูณเลข -123.0×123.0 ให้เป็นเลขทศนิยมฐานสองชนิด IEEE754 Single Precision

แสดงวิธีทำตามสมการที่ (2.76) เพื่อแปลงเลขให้ตรงตามผลคูณในรูปที่ A.12

A.2.2 กิจกรรมท้ายการทดลอง

จงใช้ลิงก์ของเว็บเพจต่อไปนี้ในการตอบคำถาม

<https://www.h-schmidt.net/FloatConverter/IEEE754.html>

รูปที่ A.13: เว็บสำหรับการตอบคำถามเพื่อสร้างเลขหรือแปลงเลขฐานสิบด้วยมาตรฐาน IEEE754 Single Precision การกดเลือกคือทำให้บิตนั้นเท่ากับ '1'

โดยแสดงวิธีทำตามเนื้อหาในหัวข้อที่ 2.6 และตรวจคำตอบตามวิธีทำการทดลองที่ได้ทำไป และบันทึกผลลัพธ์ลงบนเส้นประที่จัดไว้ให้เท่านั้น ผู้อ่านสามารถกดเปลี่ยนเครื่องหมายถูก ซึ่งแทนลอจิก 1 หากไม่มีเครื่องหมายถูกแทนลอจิก 0 ยกตัวอย่างเช่น

1. จงสร้างเลขทศนิยมฐานสอง IEEE754 ที่มีค่าเท่ากับ -0.0_{10} โดยการกดเลือกปุ่มสีเหลืองในส่วน Sign เท่านั้น

$$\text{เลขฐานสอง} = 1\ 0\ 0\ 0|0\ 0\ 0\ 0|0\ 0\ 0\ 0|0\ 0\ 0\ 0|0\ 0\ 0\ 0|0\ 0\ 0\ 0|0\ 0\ 0\ 0|0\ 0\ 0\ 0_2$$

$$\text{จำนวนสิบหก} = 8 \quad 0 \quad 0_{16}$$

ค่าฐานสิบที่จัดเก็บ (Value actually stored in float).....

ความผิดพลาด (Error due to conversion).....

2. จงสร้างเลขทศนิยมฐานสอง IEEE754 ที่มีค่าเท่ากับ -1.0_{10} โดยการกดเลือกปุ่มสีเหลี่ยมในส่วน Exponent เท่านั้น ต่อจากข้อที่แล้ว

ฐานสิบหก = 16

ค่าฐานสิบที่จัดเก็บ (Value actually stored in float).....

ความผิดพลาด (Error due to conversion).....

3. จงสร้างเลขทศนิยมฐานสอง IEEE754 ที่มีค่าเท่ากับ -1.55_{10} หรือ $1.55e0$ โดยการกดเลือกปุ่มสีเหลืองในส่วน Mantissa เท่านั้น ต่อจากข้อที่แล้ว

เลขฐานสอง = _____|_____|_____|_____|_____|_____|_____|_____ $\times 2^{\text{_____}}$

ฐานสิบหก = _____ - - - - - - - - - - - - - - - $\times 10^{16}$

ค่าฐานสิบที่จัดเก็บ (Value actually stored in float).....

ความผิดพลาด (Error due to conversion).....

4. จงสร้างเลขทศนิยมฐานสอง IEEE754 ที่มีค่าเท่ากับ $1.17549435082 \times 10^{-38}$ หรือ $1.17549435082e-38$ ซึ่งเป็นค่าอร์มัลไลซ์ที่น้อยที่สุด (Normalize)

เลขฐานสอง = _____|_____|_____|_____|_____|_____|_____|_____ $\times 2^{\text{_____}}$

ฐานสิบหก = _____ - - - - - - - - - - - - - - - $\times 10^{16}$

ค่าฐานสิบที่จัดเก็บ (Value actually stored in float).....

ความผิดพลาด (Error due to conversion).....

5. จงสร้างเลขทศนิยมฐานสอง IEEE754 ที่มีค่าเท่ากับ $1.17549421069 \times 10^{-38}$ หรือ $1.17549421069e-38$ ซึ่งอยู่ในรูปดินอร์มัลไลซ์ (Denormalize) เพราะมีค่าน้อยกว่าค่าอร์มัลไลซ์ที่ต่ำที่สุด

เลขฐานสอง = _____|_____|_____|_____|_____|_____|_____|_____ $\times 2^{\text{_____}}$

ฐานสิบหก = _____ - - - - - - - - - - - - - - - $\times 10^{16}$

ค่าฐานสิบที่จัดเก็บ (Value actually stored in float).....

ความผิดพลาด (Error due to conversion).....

6. จงสร้างเลขทศนิยมฐานสอง IEEE754 ที่มีค่าเท่ากับ $1.40129846432 \times 10^{-45}$ หรือ $1.40129846432e-45$ ซึ่งอยู่ในรูปดินอร์มัลไลซ์ (Denormalize) และต่ำที่สุด

เลขฐานสอง = _____|_____|_____|_____|_____|_____|_____|_____ $\times 2^{\text{_____}}$

ฐานสิบหก = _____ - - - - - - - - - - - - - - - $\times 10^{16}$

ค่าฐานสิบที่จัดเก็บ (Value actually stored in float).....

ความผิดพลาด (Error due to conversion).....

7. จงสร้างเลขทศนิยมฐานสอง IEEE754 ที่มีค่าเท่ากับ 1.0×10^{-46} หรือ $1e-46$ ซึ่งอยู่ในรูปดินอร์มัลไลซ์

ลีแล็ซ (Denormalize) และจัดเก็บด้วยค่า 0.0 แทน

เลขฐานสอง = _____ 2

ฐานสิบหก = — — — — — — — — — 16

ค่าฐานสิบที่จัดเก็บ (Value actually stored in float).....

ความผิดพลาด (Error due to conversion).....

8. จะ สร้าง เลข ทศนิยม ฐาน สอง IEEE754 ที่ มี ค่า เท่ากับ $3.40282346640 \times 10^{38}$ หรือ $3.40282346640e38$ ซึ่งเป็นค่านอร์มัลไลต์ที่มากที่สุด

เลขฐานสอง = 2

ଜ୍ଞାନସିବହକ = _____ 16

ค่าฐานสิบที่จัดเก็บ (Value actually stored in float).....

ความผิดพลาด (Error due to conversion).....

9. จงสร้างเลขทศนิยมฐานสอง IEEE754 ที่มีค่าเท่ากับ 3.5×10^{38} หรือ $3.5e+38$ ซึ่งมากกว่าค่าอนร์ม ไลอซ์ที่มากที่สุด ซึ่งหมายถึงค่าอนันต์ (∞ : Infinity) ตามตารางที่ [2.12](#)

เลขฐานสอง = _____ 2

ฐานสิบหก = —————— —————— —————— —————— —————— —————— —————— —————— 16

ค่าฐานสิบที่จัดเก็บ (Value actually stored in float).....

ความผิดพลาด (Error due to conversion).....

10. จงสร้างเลขทศนิยมฐานสอง IEEE754 ที่มีค่าเท่ากับ NaN (Not a Number) ตามตารางที่ 2.12

เลขฐานสอง = | | | | | | | | 2

รายงานสิบหก = ๑๖

ค่าฐานสิบที่จัดเก็บ (Value actually stored in float).....

ความผิดพลาด (Error due to conversion).

A.3 รหัสของข้อมูลตัวอักษร

A.3.1 การทดลอง

การทดลองในหัวข้อนี้ จะเป็นการแปลงรหัสตัวอักษรภาษาอังกฤษ และไทย เป็นรหัส ASCII และ Unicode ชนิด UCS-2 ตามเนื้อหา ในหัวข้อ 2.7 ผ่านทางเว็บไซต์ <https://www.branah.com/ascii-converter> ที่มีนักพัฒนาเพื่อเผยแพร่ความรู้เป็นวิทยาทานเช่นเดียวกับเว็บที่ได้ทดลองมา

1. เปิดเว็บตามลิงก์ต่อไปนี้ หรือ กดปุ่มซ้ายบนชื่อลิงก์ <https://www.branah.com/ascii-converter>

2. กรอกข้อความต่อไปนี้ ลงไว้ในกล่องข้อความ ASCII

ไทย ก ข ค a b c

โปรดสังเกต ระหว่างตัวอักษรมีช่องว่าง 1 ตัวอักษรเสมอ

3. กดปุ่ม Convert ซ้ายบนสุด 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 11100100010 111000000001 111000000010 111000000100 01100001 01100010 01100011

Convert Base64 (Example: YSBiIGM=)

RCAXIClgsASACIAQgYSBiIGM=

รูปที่ A.14: 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 - ๙
2. จงอธิบายวิธีการหาค่าฐานสิบ 0 - 9 จากรหัส Unicode ของตัวอักษร ๐ - ๙
3. จงเปิดเว็บที่มีข้อความภาษาไทย เช่น เว็บข่าว แล้วทดลองเปลี่ยนการนำเสนอ บน จอ เพื่อ View source เช่น Google Chrome ใช้เมนู Tool-> View Source แล้ว Find หรือกดปุ่ม CTRL-F คำว่า charset ว่ามีค่าเท่ากับ utf-8 หรือไม่ เพราะเหตุใด

ภาคผนวก B

การทดลองที่ 2 ตัวอย่างการประกอบและติดตั้งบอร์ด Raspberry Pi3

การทดลองนี้ เสริมสร้างประสบการณ์ให้ผู้อ่านได้มีโอกาสจับต้องบอร์ด และ อุปกรณ์เสริม และสนับสนุนเนื้อหาของหัวข้อที่ 3.1 ในส่วนของฮาร์ดแวร์ โดยมีวัตถุประสงค์ ดังต่อไปนี้

- เพื่อให้รู้จักโครงสร้างและรายละเอียดต่าง ๆ ของบอร์ด Pi3
- เพื่อให้เข้าใจลักษณะการระบายความร้อนของอุปกรณ์อิเล็กทรอนิกส์
- เพื่อให้ทดสอบการเชื่อมต่อกับอุปกรณ์อินพุต/เอาต์พุตที่จำเป็น

การทดลองนี้ต้องเป็นตัวอย่างการประกอบบอร์ดเข้ากับกล่อง ซึ่งอาจแตกต่างกันไปตามรายละเอียดของกล่องที่ผู้อ่านจัดหามา และอาจต้องใช้อุปกรณ์อื่น ๆ ได้แก่ จอมอนิเตอร์ที่รองรับสัญญาณ HDMI หากไม่มีผู้อ่านสามารถใช้สายแปลงที่จำเป็น เช่น แปลงจากสัญญาณ HDMI เป็นสัญญาณ VGA หรือแปลงเป็นสัญญาณ DVI คีย์บอร์ด เม้าส์ กล่องสำหรับรับบอร์ด และอุปกรณ์ระบบบายความร้อนหรือชิปซิงก์ ตามความเหมาะสม

บอร์ด Raspberry Pi เป็นคอมพิวเตอร์ขนาดเล็ก เท่ากับบัตรเครดิต เริ่มต้นออกแบบ และผลิตในประเทศสหราชอาณาจักร โดยมูลนิธิ Raspberry Pi Foundation ซึ่งเริ่มต้นจากการเป็นคอมพิวเตอร์ราคาถูกสำหรับการศึกษา บอร์ดมีการประยุกต์ใช้งานเพิ่มขึ้นเรื่อย ๆ ตั้งแต่ปี 2012 ทำให้มีการพัฒนามาเรื่อย ๆ จนเป็นเจนเนอเรชันที่ 3 และ 4 ในปัจจุบัน โดยมีสองรุ่น คือ โมเดล A และ โมเดล B โมเดล A เป็นอุปกรณ์ที่เรียบง่าย ตั้งทุนต่ำกว่าโมเดล B ในขณะที่ โมเดล B เหมาะกับการใช้งานเป็นคอมพิวเตอร์ตั้งโต๊ะ และเซอร์ฟเวอร์ เนื่องจากมีการเชื่อมต่อกับเครือข่ายด้วยสายแลนหรือสาย Ethernet และแบบไร้สายได้แก่ WiFi และ Bluetooth



รูปที่ B.1: รูปแสดงรายการอุปกรณ์สำหรับประกอบบอร์ด ที่มา: rs-online.com

B.1 รายการอุปกรณ์ฮาร์ดแวร์

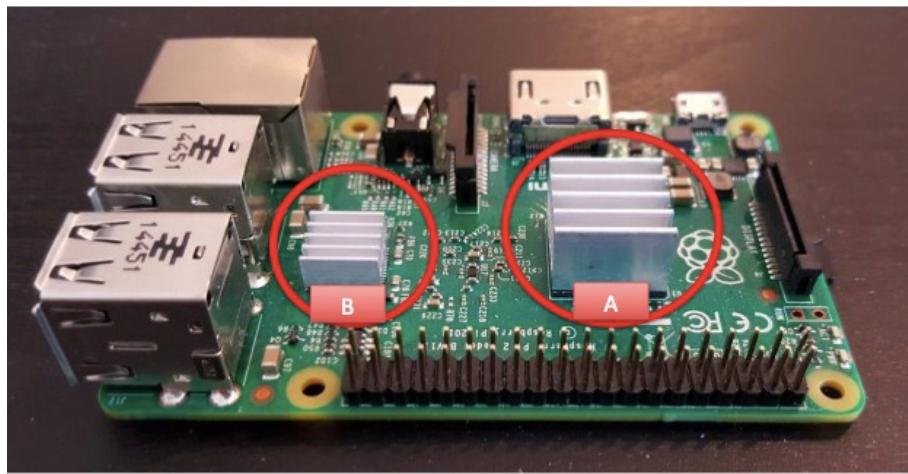
ผู้อ่านสามารถถอดสั่งชิ้นส่วนที่ต้องใช้ (Toolkit) สำหรับบอร์ด Pi3/Pi4 ในรูปที่ B.1 ประกอบด้วยรายการดังต่อไปนี้

ลำดับ	ชื่อและรายละเอียด	จำนวน	มี/ไม่มี
1	บอร์ด Pi3 โมเดล B แรม 1 กิกะไบต์	1	
2	อడิโอแ dik แปลงไฟกระแทก 5.0 โวลต์ 2.5 แอมเปอร์ (ปลายสายเป็นหัวไมโคร USB ชนิด B)	1	
3	สายเชื่อมต่อชนิด HDMI	1	
4	แผ่นวงจรprotoboard (Protoboard)	1	
5	หัวขยายการเชื่อมต่อ GPIO สำหรับบอร์ดโมเดล B	1	
6	สายแพ GPIO 40 ขาสำหรับบอร์ดโมเดล B	1	
7	ชิ้นซิจิกขนาดเล็ก	1-2	
8	การ์ดหน่วยความจำไมโคร SD ขนาด 16GB	1	
9	กล่องสำหรับบอร์ด Pi	1 ชุด	
10	หัวเชื่อมต่อสายแพขนาด 40 ขา	1	
11	สายแพขนาด 40 ขาชนิดตัวเมี้ยx2	1	
12	สายต่อวงจรชนิดตัวเมี้ยx2	1 ชุด	
13	สายแปลง HDMI เป็น VGA	1	

B.2 ตัวอย่างการประกอบบอร์ด Pi3 และกล่อง

ตัวอย่างการประกอบบอร์ด Pi3 เพื่อให้พร้อมใช้งานสำหรับการศึกษาและทดลอง จะต้องเสริมความแข็งแรงให้บอร์ด เพิ่มความสามารถในการระบายความร้อน ต่อขยายขาเขื่อมบันบอร์ดให้ยาวขึ้น โดยเรียงลำดับดังนี้

B.2.1 ประกอบฮีทซิงก์ (Heat Sink)



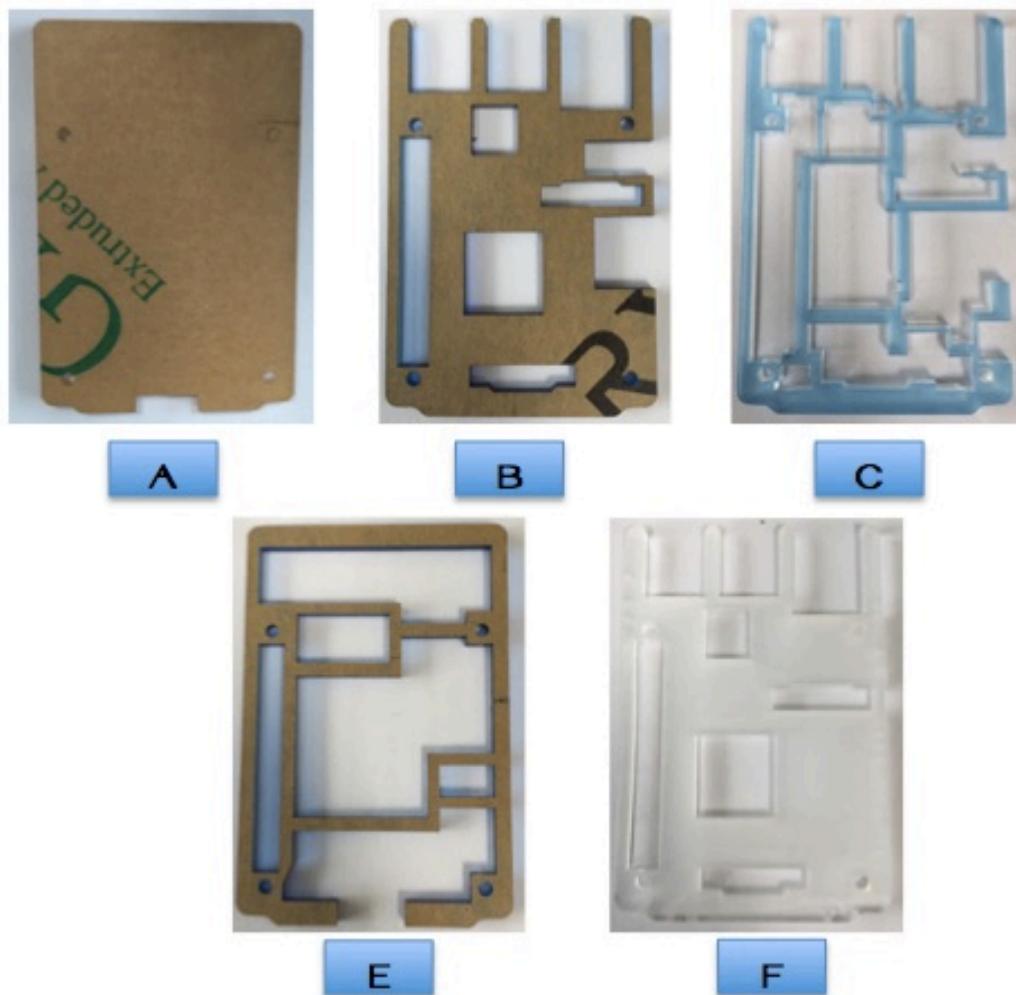
รูปที่ B.2: บอร์ด Pi3 เมื่อติดฮีทซิงก์เรียบร้อยแล้ว

1. ตรวจสอบความเรียบร้อยของบอร์ด Pi ที่ได้รับมา หากมีร่องรอยชำรุด หรือ ใหม่ ขอให้แจ้งกับผู้ดูแล
2. หยิบฮีทซิงก์ที่มีขนาดใหญ่ที่สุด แกะแผ่นเคลือบการสองหน้าออก วางด้านที่มีการลงบนชิป BCM2837 ตรงกลาง ณ ตำแหน่ง A ในรูปที่ B.2
3. หยิบฮีทซิงก์ที่มีขนาดกลาง แกะแผ่นเคลือบการสองหน้าออก วางด้านที่มีการลงบนชิป LAN9514 ณ ตำแหน่ง B ในรูปที่ B.2

ฮีทซิงก์ A ติดบนชิป BCM2837 ขนาดใหญ่กว่า เนื่องจากชิปทำงานที่ความถี่สัญญาณคล็อกสูงกว่า ชิปซึ่งมากกว่า ในขณะที่ฮีทซิงก์ B หากมีให้มา จะติดบนชิป LAN9514 ซึ่งภายในคือ รูทહับ (Root Hub) ของ USB 2.0 และ Ethernet Controller 10/100 เมกะบิตต่อวินาที

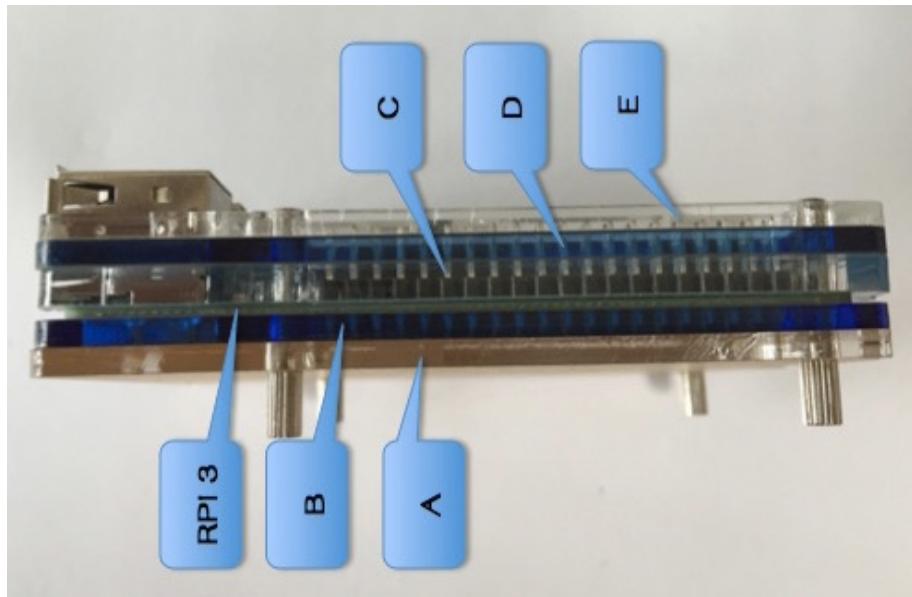
B.2.2 ประกอบกล่องกับบอร์ด Pi3

กล่องที่จำหน่ายมาพร้อมบอร์ดของบริษัทแห่งหนึ่ง ประกอบด้วยแผ่นพลาสติก 5 ชิ้น ตามรูปที่ B.3

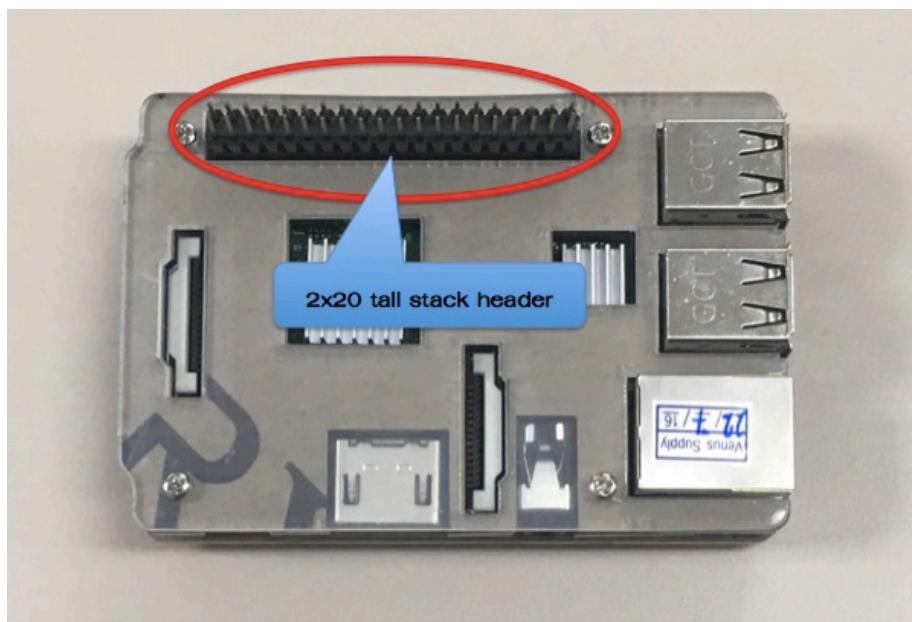


รูปที่ B.3: แผ่นพลาสติก 5 ชิ้น สำหรับประกอบเป็นกล่องของบอร์ด Pi3

1. ประกอบแผ่นพลาสติกและบอร์ด Pi ด้วยกันตามรูปที่ B.4 แผ่น B ประกอบกับบอร์ด Pi ด้านล่าง และช้อนบนแผ่น A ซึ่งอยู่ชั้นล่างสุด ประกอบแผ่น C D E เข้าด้วยกัน โดยแผ่น E อยู่ชั้นบนสุด แล้ววางช้อนบนบอร์ด
2. ใช้สกรูและไขควงแผ่นพลาสติกและบอร์ด Pi3 เข้าด้วยกันทั้ง 4 มุน
3. ติดตั้งขาเชื่อมขยายชนิด 2x20 หรือ 2 แคล ๆ ละ 20 ขาบนบอร์ด โดยสังเกตจากในรูปที่ B.5 ว่าขาหมายเลข 1 อยู่ตรงมุมซ้ายล่าง



รูปที่ B.4: บอร์ด Pi ที่มีกล่องประกอบเรียบร้อยแล้ว

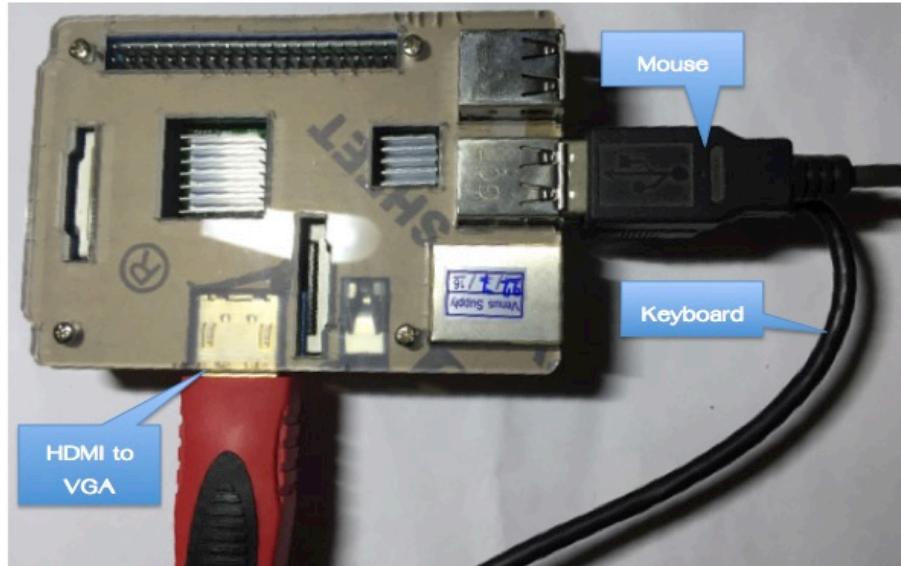


รูปที่ B.5: บอร์ด Pi เมื่อประกอบขาเชื่อมขยาย 2x20

B.3 เครื่องคอมพิวเตอร์ส่วนบุคคลจากบอร์ด Pi3/Pi4 โมเดล B

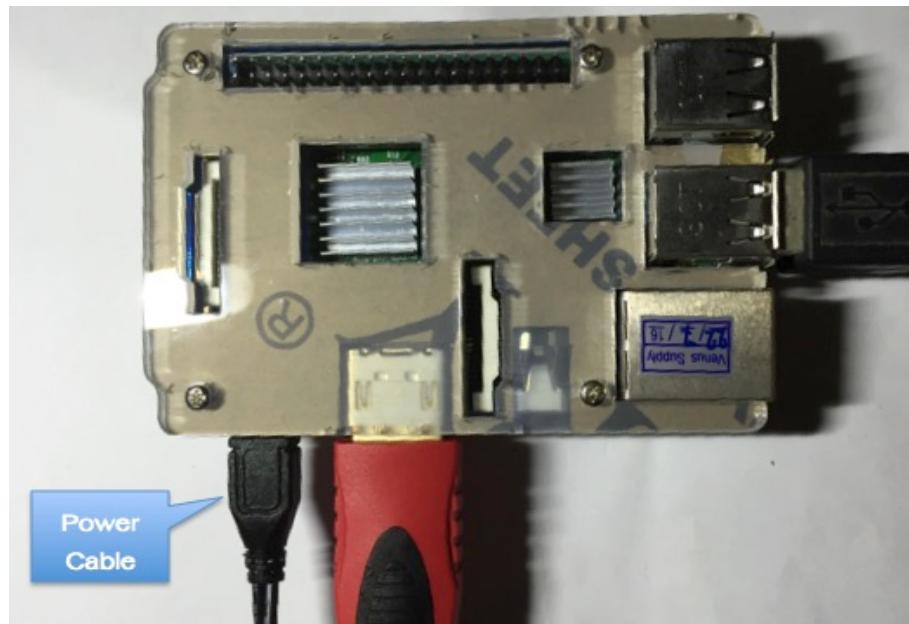
1. ถอนอแดปเตอร์ไฟ 5 โวลต์ จากเต้าเสียบไฟ 220 โวลต์ ก่อนเพื่อความปลอดภัยของผู้ใช้งานและวงจร
2. เชื่อมต่อสายแอลจี HDMI กับบอร์ด Pi และจึงเชื่อมสาย VGA กับจอคอมอนิเตอร์ เลือกอินพุตของจอเป็น Analog Input หรือ
3. เชื่อมต่อสายแอลจี HDMI ไมโคร กับบอร์ด Pi3/Pi4 และจึงเชื่อมต่อกับจอคอมอนิเตอร์ช่อง HDMI

4. เชื่อมต่อคีย์บอร์ดและมาส์กับช่องเสียบสาย USB สีดำบนบอร์ด Pi โปรดสังเกตสัญลักษณ์ของ USB บนหัวสายจะต้องหมายขึ้นดังรูปที่ [B.6](#)



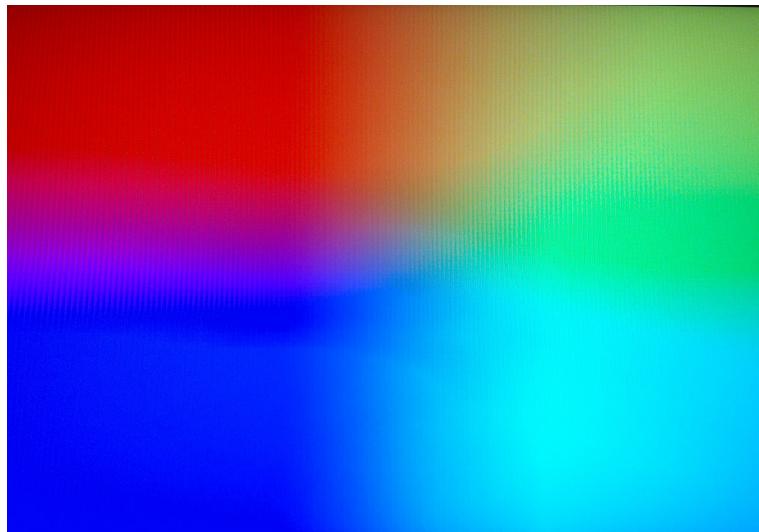
รูปที่ B.6: การเชื่อมต่อคีย์บอร์ดและมาส์กับช่องเสียบสาย USB บนบอร์ด Pi ให้เรียบร้อย

5. เชื่อมต่อหัวไมโคร USB (กรณี Pi3) หรือ USB-C (กรณี Pi4) กับบอร์ดก่อน แล้วจึงเสียบอแดปเตอร์เข้ากับเต้ารับไฟ 220 โวลต์ โปรดสังเกตลักษณ์ของ USB บนหัวสายจะต้องหมายขึ้น ดังรูปที่ [B.7](#)



รูปที่ B.7: การเชื่อมต่อไฟเลี้ยงจากอแดปเตอร์ทางหัวไมโคร USB กับบอร์ด Pi

6. โปรดสังเกตหลอดไฟ LED สีแดงจะสว่างขึ้นเมื่อไม่มีอะร์พิดเพลต หลังจากนั้น ภาพสีรุ้งจะปรากฏขึ้นบนจอด้านซ้ายบน ดังรูปที่ [B.8](#)



รูปที่ B.8: ภาพสีรุ่งบนจออันเกิดจากบอร์ด Pi3 ทำงานเป็นปกติ

B.4 กิจกรรมท้ายการทดลอง

กิจกรรมต่อไปนี้อาศัยความชำนาญของผู้อ่านส่วนตัว เพื่อป้องกันความเสียหายที่อาจเกิดขึ้น ผู้เขียนขอแนะนำสำหรับผู้อ่านที่มีความชำนาญหรืออวนกลับมาทำในภายหลังที่ทำการทดลองจนครบแล้ว

1. ถอดอแดปเตอร์ออกจากเต้าเสียบ ประกอบสายแพกับหัวต่อ เชื่อมกับprotoบอร์ด และเชื่อมต่อสายมิเตอร์สีดำกับกราวน์ของบอร์ด
2. เสียบอแดปเตอร์ในเต้าเสียบ วัดความต่างศักย์ของไฟ 5 โวลต์ และ 3.3 โวลต์ จากหัวต่อ 40 ขา โดยใช้สายค่อนเน็คเตอร์ไปเสียบบนprotoบอร์ดแล้วจึงทำการวัด

ภาคผนวก C

การทดลองที่ 3 การติดตั้งระบบปฏิบัติการ Raspberry Pi OS

การทดลองนี้เสริมสร้างประสบการณ์ให้ผู้อ่านได้มีโอกาสติดตั้งระบบปฏิบัติการ Raspberry Pi OS และโปรแกรมเสริมอื่น ๆ โดยอาศัยเครื่องคอมพิวเตอร์ที่ทำงานระบบปฏิบัติการ เช่น ลินุกซ์ Ubuntu, ไมโครซอฟต์วินโดวส์ และ Mac OS ติดตั้งลงบนการ์ดหน่วยความจำไมโคร SD เป็นอุปกรณ์สำรองข้อมูล การทดลองจะช่วยเสริมสร้างความเข้าใจเนื้อหาของบทที่ 3 ในส่วนของซอฟต์แวร์ โดยมีวัตถุประสงค์ดังต่อไปนี้

- เพื่อให้เข้าใจการติดตั้งระบบปฏิบัติการ Raspberry Pi OS ผ่านทางเครือข่ายอินเทอร์เน็ต
- เพื่อประกอบการใช้งานและพัฒนาโปรแกรมบนระบบปฏิบัติการ Raspberry Pi OS ซึ่งเป็นลินุกซ์เวอร์ชันสำหรับบอร์ดตระกูล Raspberry Pi

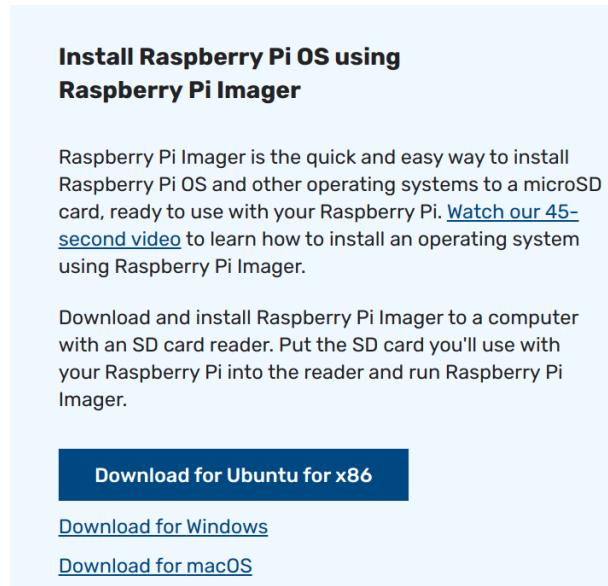
ก่อนผู้อ่านจะติดตั้งระบบปฏิบัติการ Raspberry Pi OS บนบอร์ด Pi ผู้อ่านจะต้องเตรียมการ์ดหน่วยความจำไมโคร SD ขนาดความจุไม่น้อยกว่า 16 กิกะไบต์ ให้เรียบร้อย แล้วจึงติดตั้งโปรแกรมตามขั้นตอนดังต่อไปนี้

C.1 การเตรียมการ์ดหน่วยความจำไมโคร SD

- ทำการดาวน์โหลดไฟล์โปรแกรม Raspberry Pi Imager สำหรับติดตั้งระบบปฏิบัติการ Raspberry Pi OS ในการ์ดหน่วยความจำไมโคร SD ตามลิงก์ต่อไปนี้

<https://www.raspberrypi.org/software/>

- เลือกดาวน์โหลดตามระบบปฏิบัติการที่ผู้อ่านใช้งานอยู่ เช่น วินโดวส์ Mac OS หรือ ลินุกซ์ Ubuntu



รูปที่ C.1: หน้าต่างดาวน์โหลดไฟล์โปรแกรม Raspberry Pi Imager สำหรับติดตั้งระบบปฏิบัติการ Raspberry Pi OS ในการ์ดหน่วยความจำไมโคร SD

3. ทำการติดตั้งโดยตาม installShield Wizard จนแล้วเสร็จ

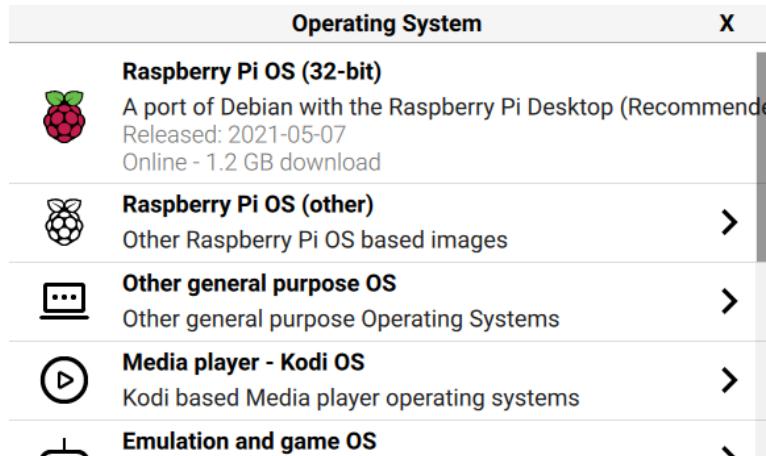
C.2 การติดตั้ง Raspberry Pi OS บนการ์ดหน่วยความจำไมโคร SD

1. รันโปรแกรม Raspberry Pi Imager ซึ่งมีหน้าต่างหลักในรูปที่ C.2



รูปที่ C.2: หน้าต่างของโปรแกรม Raspberry Pi Imager

2. กดปุ่ม CHOOSE OS เพื่อเลือกระบบปฏิบัติการ ในรูปที่ C.3

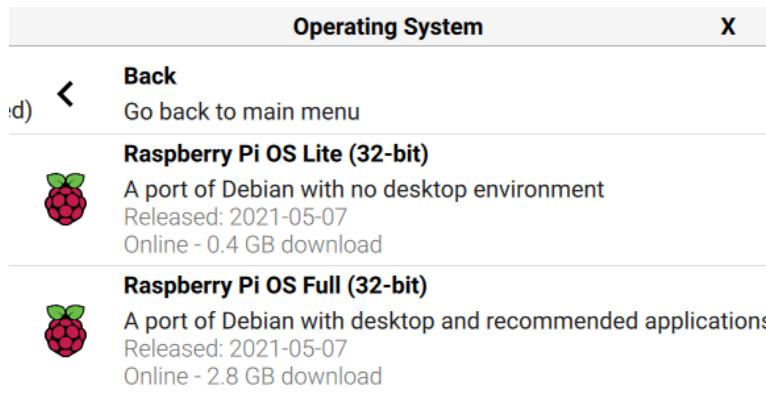


รูปที่ C.3: เมนูตัวเลือกใต้เมนู CHOOSE OS

3. กดเลือก **Raspberry Pi OS (32-bit)** ลำดับบนสุด ซึ่งเป็นตัวเลือกที่เว็บไซต์แนะนำ (Recommended)

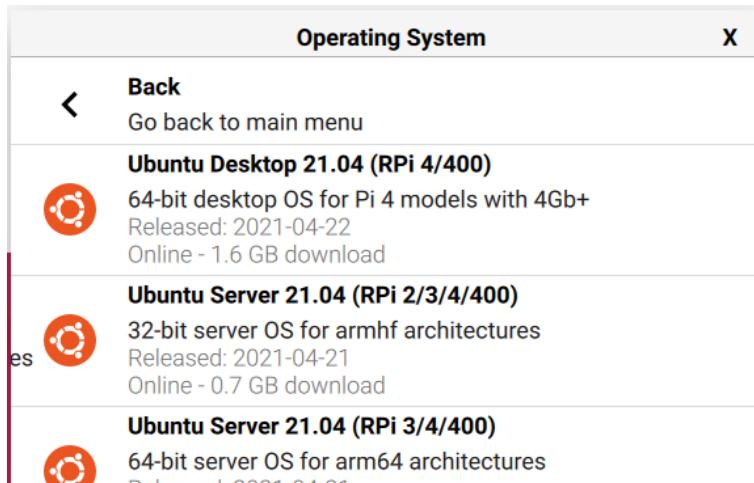
4. ผู้อ่านขั้นสูงสามารถเลือก Raspberry Pi OS อื่น ๆ ได้ ดังนี้

- กด Raspberry Pi OS (other) เพื่อเลือกระบบปฏิบัติการอื่น ๆ ที่ตรงตามความต้องการในรูปที่ C.4



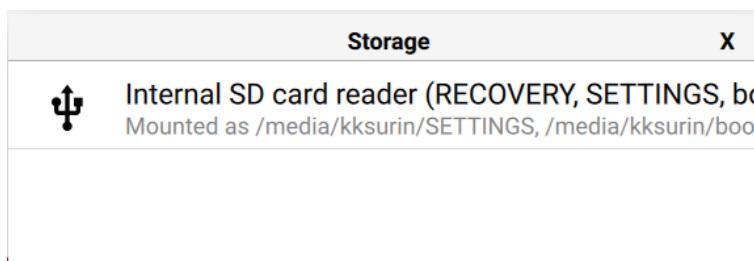
รูปที่ C.4: เมนูตัวเลือกอื่น ๆ ใต้เมนู Raspberry Pi OS (other)

- กดเลือก Other general purpose OS เพื่อเลือกระบบปฏิบัติการอื่น ๆ ที่ไม่ใช่ Raspberry Pi OS ในรูปที่ C.5



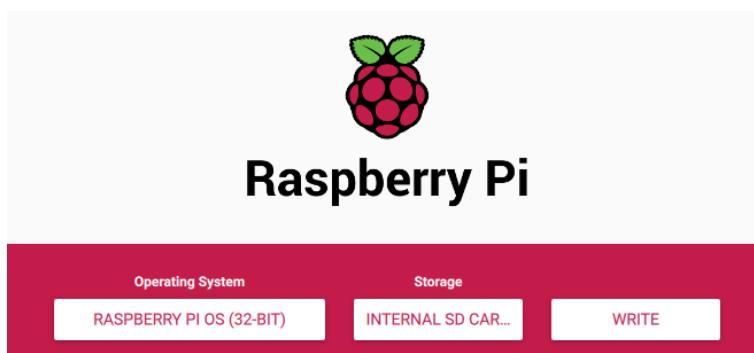
รูปที่ C.5: เมนูตัวเลือกอื่น ๆ ใต้เมนู Other general purpose OS

- กดปุ่ม CHOOSE STORAGE เพื่อเลือกการ์ดหน่วยความจำ SD ที่ต้องการ กรณีที่เสียบอยู่หลายการ์ด ในรูปที่ C.6



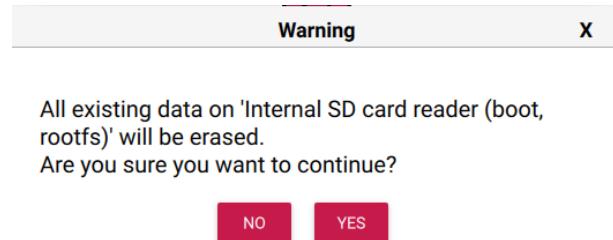
รูปที่ C.6: หน้าต่าง Imager

- เมื่อตั้งค่าตัวเลือกภายใต้ปุ่ม CHOOSE OS และ CHOOSE STORAGE แล้ว ผู้ใช้จึงสามารถกดปุ่ม WRITE เพื่อเริ่มต้นการดาวน์โหลดและเขียนได้ ในรูปที่ C.7



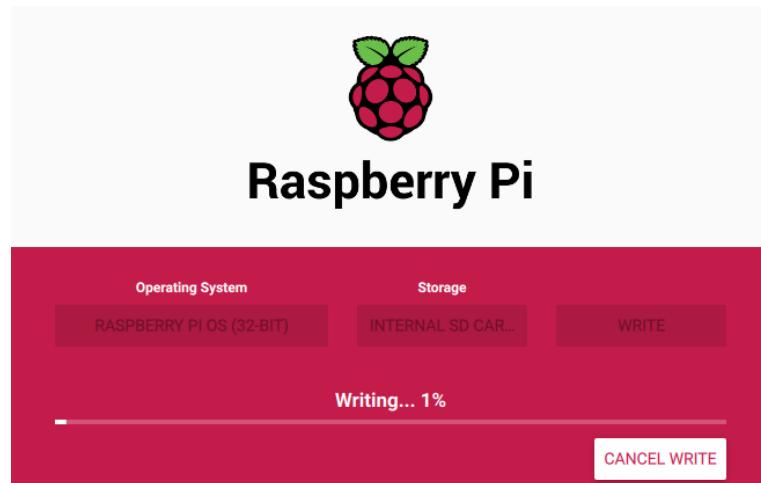
รูปที่ C.7: ปุ่ม WRITE ในหน้าต่าง Raspberry Pi Imager

- หากการ์ดหน่วยความจำมีข้อมูลเดิม หน้าต่างเตือนจะปรากฏขึ้นตามรูปที่ C.8 หากต้องการเขียนทับให้กดปุ่ม Yes และหากไม่มั่นใจให้กดปุ่ม No



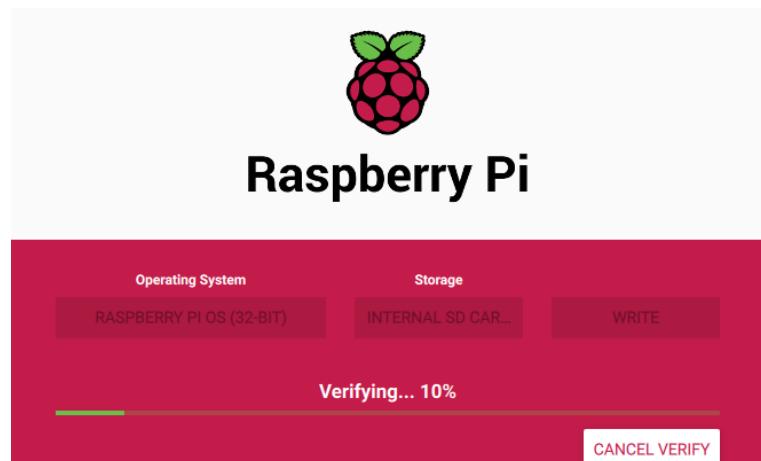
รูปที่ C.8: หน้าต่างเตือนผู้ใช้ที่ต้องการเขียนทับการ์ดหน่วยความจำ SD

- ระหว่างที่ดำเนินการดาวน์โหลด และ เขียนการ์ดไปพร้อม ๆ กัน ผู้อ่านต้องดูแล การเชื่อมต่อไม่ให้ติดขัด ในรูปที่ C.9



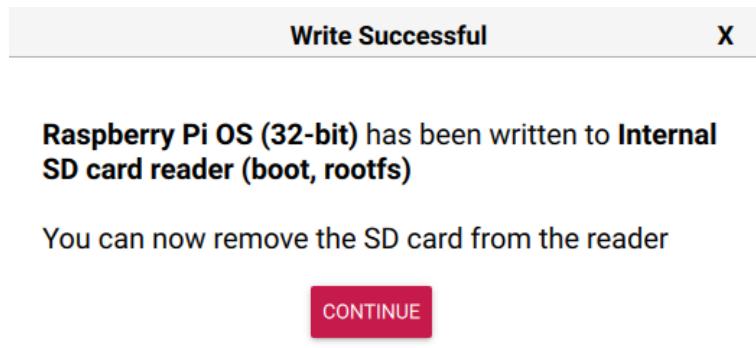
รูปที่ C.9: หน้าต่าง Raspberry Pi Imager ทยอยเขียนข้อมูลภายในการ์ด

- เมื่อโปรแกรมเขียนการ์ดจนครบ 100% แล้ว จึงเริ่มทวนสอบ (Verify) ข้อมูลภายในการ์ด ในรูปที่ C.10



รูปที่ C.10: หน้าต่าง Raspberry Pi Imager ทวนสอบ (Verify) ข้อมูลภายในการ์ด

10. เมื่อเขียนหรือติดตั้งลงในкар์ดหน่วยความจำสำเร็จ (Successful) การ์ดจะมีพาร์ทิชันชื่อ boot และ rootfs ในรูปที่ [C.11](#)

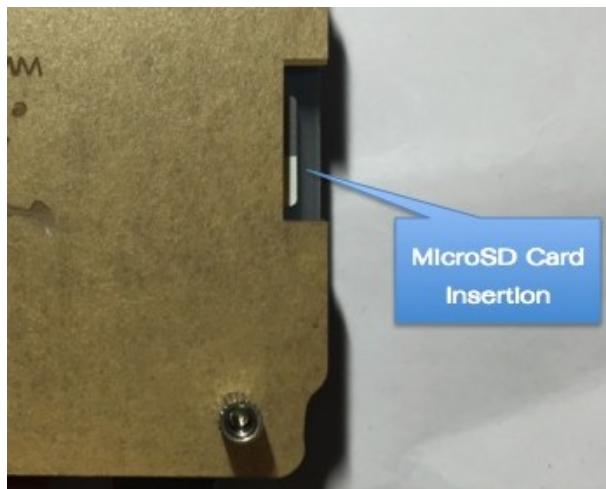


รูปที่ C.11: ปุ่ม CONTINUE ในหน้าต่าง Raspberry Pi Imager เมื่อติดตั้งสำเร็จ

11. กดปุ่ม CONTINUE แล้วปิดโปรแกรม และสั่งให้ระบบปฏิบัติการหลักปลดการ์ดหน่วยความจำไมโคร SD ออกจากเครื่อง

C.3 การบูตระบบปฏิบัติการ RaspberryPi OS

1. หมายบอร์ด Pi แล้วจึงสอดการ์ดหน่วยความจำไมโคร SD ที่ได้เข้าไปในสล็อตบนบอร์ด Pi โดยสังเกตว่าการ์ดคว่ำหน้าลงดังรูป เพื่อให้หน้าสัมผัสรองกับบอร์ด



รูปที่ C.12: สอดการ์ดเข้าไปในสล็อตบนบอร์ด Pi โดยหมายบอร์ดขึ้นมา โปรดสังเกตการ์ดหน่วยความจำจะต้องมีลักษณะดังรูป

2. ตรวจสอบว่าการ์ดหน่วยความจำไมโคร SD เสียบถูกต้องแล้วจึงเสียบอแดปเตอร์ไฟเลี้ยงให้กับบอร์ด
3. ตรวจสอบว่าบอร์ดทำงานเมื่อจ่ายไฟให้ตามรูปที่ [C.13](#) บอร์ดจะเริ่มต้นทำงาน



รูปที่ C.13: หน้าต่าง Welcome ของระบบปฏิบัติการ Raspberry Pi OS

C.4 การตั้งค่าบอร์ด Pi เพื่อใช้งาน

C.4.1 การตั้งค่าต่างๆ

เมื่อบอร์ดสามารถบูตระบบปฏิบัติการได้สำเร็จตามรายละเอียดในหัวข้อที่ 3.3.1 ผู้ใช้จะต้องตั้งค่าต่างๆ (Configure) บอร์ดให้พร้อมสำหรับใช้งานต่อไป ดังนี้

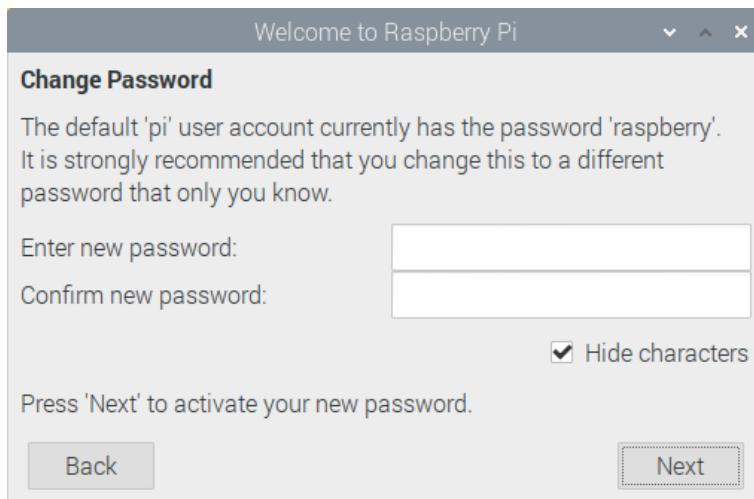
1. ตั้งค่าประเทศ โซนเวลา และภาษาในการใช้งานเมนูเป็นภาษาอังกฤษ ตามรูปที่ C.14 เพื่อใช้เมนูเป็นภาษาอังกฤษซึ่งจะช่วยให้เรียนรู้คอมพิวเตอร์ดีกว่า



รูปที่ C.14: หน้าต่างตั้งค่าประเทศ โซนเวลา และภาษาในการใช้งานเมนูเป็นภาษาอังกฤษ

2. สำหรับผู้อ่านขึ้นเริ่มต้น ผู้อ่านไม่ควรปรับแก้ใด ๆ ระบบจะตั้งชื่อ username อัตโนมัติคือ pi โดยมีรหัสผ่าน (Password) คือ raspberry

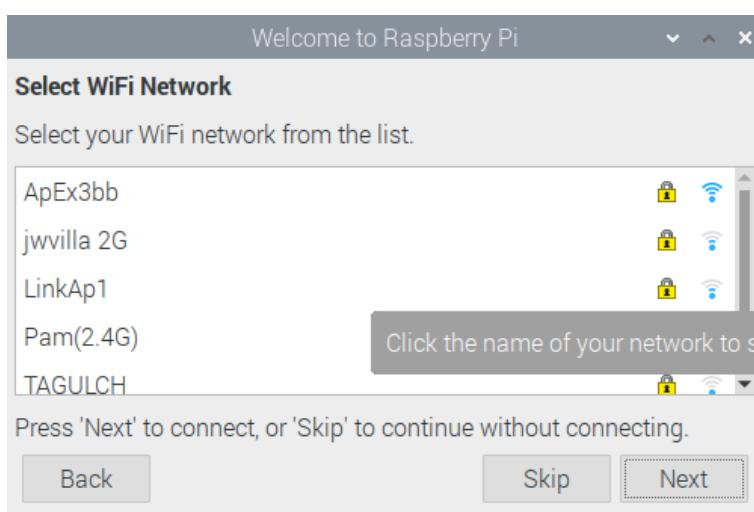
3. สำหรับผู้อ่านขึ้นสูงทำการตั้งชื่อผู้ใช้ และ พาสเวิร์ด ซึ่งผู้อ่านควรใช้ชื่อ pi และ พาสเวิร์ดใหม่ที่ปลอดภัยในรูปที่ C.15 เพื่อความปลอดภัยในอนาคต



รูปที่ C.15: หน้าต่างสำหรับการเปลี่ยนรหัสผ่านใหม่ของ username ชื่อ pi โดยจะต้องกรอกรหัสผ่านใหม่ จำนวน 2 ครั้ง

C.4.2 การตั้งค่า WiFi เพื่อเชื่อมต่อกับอินเทอร์เน็ต

1. ระบบมองเห็นสัญญาณ WiFi และแสดงรายชื่อของสัญญาณ WiFi (SSID) ที่อยู่รอบ ๆ บริเวณบอร์ด ตามตัวอย่างในรูปที่ C.16 สัญญาณแม่กุญแจ หมายถึง การเข้ารหัสป้องกันซึ่งผู้ใช้ต้องกรอกพาสเวิร์ดก่อนเขื่อมต่อ



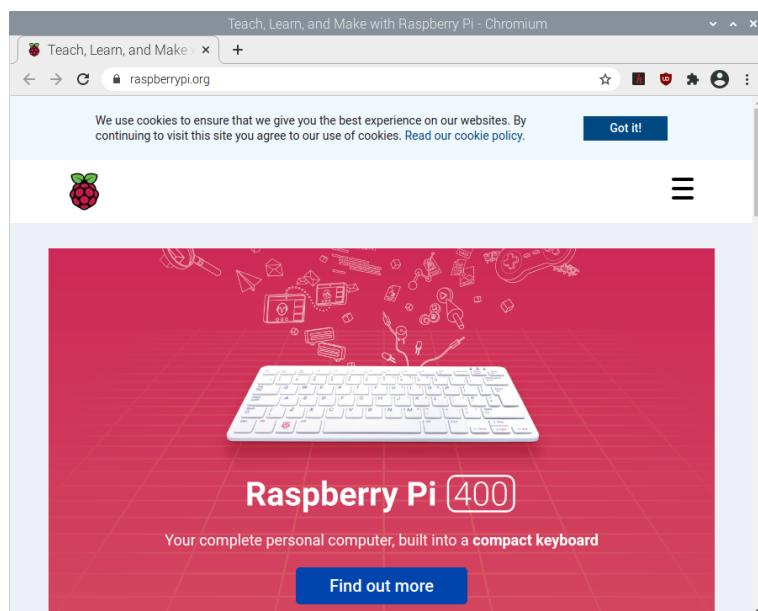
รูปที่ C.16: ตัวอย่างรายชื่อสัญญาณ WiFi รอบ ๆ ที่บอร์ด Pi มองเห็น ซึ่งจะแตกต่างกับของผู้อ่าน

2. คลิกเลือกรายชื่อสัญญาณที่ต้องการ ตามตัวอย่างในรูปที่ C.17



รูปที่ C.17: หน้าต่างสำหรับกรอกพาสเวิร์ดของสัญญาณ WiFi ที่ต้องการเชื่อมต่อ

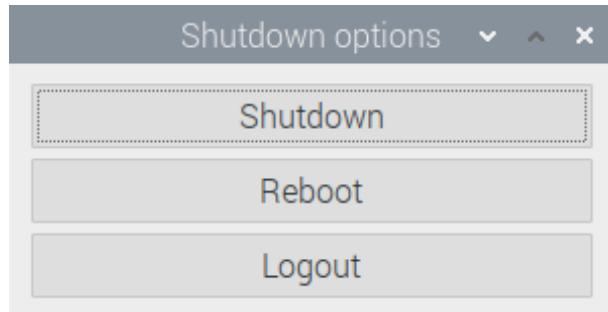
3. เมื่อเชื่อมต่อสัญญาณ WiFi ลูกต้องแล้ว เปิดโปรแกรมเบราว์เซอร์ชื่อ Chrome เพื่อทดสอบการเชื่อมต่อเครือข่ายอินเทอร์เน็ตได้โดย



รูปที่ C.18: หน้าเพจเริ่มต้นของเว็บไซต์ www.raspberrypi.org

C.4.3 การรีสตาร์ตและซัตดาวน์

- เมื่อติดตั้งค่าระบบแล้วเสร็จ ผู้อ่านควรทำการรีบูต หรือ รีสตาร์ตเครื่อง เลื่อนเมาส์ไปคลิกปุ่มสัญลักษณ์รูปผล Raspberry ซึ่งทำหน้าที่เป็นปุ่มเมนูหลัก เลือกเมนูย่อย Logout ณ ตำแหน่งล่างสุด เมนูนี้มักใช้เรียกเมื่อผู้ใช้ต้องการหลังการอัปเดตซอฟต์แวร์ต่าง ๆ ที่จำเป็น หรือ ผู้ใช้ต้องการปรับแก้อาการต่าง ๆ 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 สีเขียวจะกระพริบจนดับลงค่อยๆ ตอนเดปเตอร์ออกจากเต้าเสียบไฟ 220 โวลต์

C.5 กิจกรรมท้ายการทดลอง

- การติดตั้งระบบจากไฟล์ config.txt เพื่อแจ้งให้ ARM Loader ทำการบูตระบบตามรายละเอียดในไฟล์นั้น ผู้อ่านสามารถอ่านค่าโดยใช้คำสั่ง

```
$ cat /boot/config.txt
```

ขอให้ผู้อ่านสังเกต และบันทึก ประโยชน์ ที่ไม่chein ตน ด้วย สัญลักษณ์ # เพื่อค้นคว้าเพิ่มเติมใน google.com

- คำสั่ง sudo ย่อมาจากคำว่าอะไร และทำไมต้องใช้คำสั่งนี้หน้าคำสั่งอื่น ๆ ในโปรแกรม Terminal
- ค้นคว้าเพิ่มเติมว่าไฟ LED สีเขียวบ่งบอกสัญญาณอะไร เหตุใดจึงต้องรอให้ดับก่อนตอนเดปเตอร์ออก
- สำรวจส่วนต่าง ๆ ของหน้าเดสก์ท็อป (Desktop) และวัดตามคร่าว ๆ พร้อมรายละเอียดสำคัญ
- สำรวจเมนูหลัก และเมนูรองว่ามีรายละเอียดอะไรบ้าง และวัดเป็นแนวภูมิต้นไม้
- ค้นหาวิธีการเพิ่มคีย์บอร์ดภาษาไทยเพื่อใช้งานบนระบบปฏิบัติการ Raspberry Pi OS

ภาคผนวก D

การทดลองที่ 4 การใช้งานระบบปฏิบัติการยูนิกซ์เบื้องต้น

ยูนิกซ์ (Unix) เป็นระบบปฏิบัติลำดับแรก ๆ ของโลกที่เป็นต้นแบบการสร้างระบบปฏิบัติการต่าง ๆ รวมทั้งระบบปฏิบัติการลินุกซ์ และ Raspberry Pi OS ผู้อ่านสามารถเรียนรู้การใช้งานคำสั่งพื้นฐานด้วยการพิมพ์คำสั่งทางคีย์บอร์ด และกราฟิกไปพร้อมกัน โดยมีวัตถุประสงค์ดังต่อไปนี้

- เพื่อเปรียบเทียบการทำงานแบบกราฟิกและแบบคำสั่งทางคีย์บอร์ด
- เพื่อให้ผู้อ่านใช้คำสั่งเพื่อบริหารจัดการไฟล์ในไดเรกทอรีหรือโฟลเดอร์เบื้องต้น
- เพื่อวางแผนการใช้งานระบบปฏิบัติการยูนิกซ์เบื้องต้นสำหรับพัฒนาโปรแกรมภาษาต่าง ๆ
- เพื่อค้นคว้าข้อมูลขั้นสูงของบอร์ด Pi

ผู้อ่านที่คุ้นเคยกับระบบปฏิบัติการวินโดว์ส และการพิมพ์คำสั่งทางคีย์บอร์ด (Command Line) ของระบบปฏิบัติการดอส (DOS: Disk Operating System) ในอดีต จะค้นพบว่า คำสั่งเหล่านี้มีความใกล้เคียงกัน แต่ยูนิกซ์จะเข้มงวดกว่า วินโดว์ส และ DOS ขอให้ผู้อ่านปฏิบัติตามคำสั่งอย่างระมัดระวัง และสังเกตตัวพิมพ์อย่างละเอียดว่าเป็นตัวพิมพ์ใหญ่ หรือเล็ก เพื่อสร้างความคุ้นเคยกับการพัฒนาโปรแกรมด้วยภาษาอื่น ๆ ต่อไป

D.1 การใช้งานระบบผ่านทาง GUI

D.1.1 หน้าจอเดสก์ท็อป (Desktop)

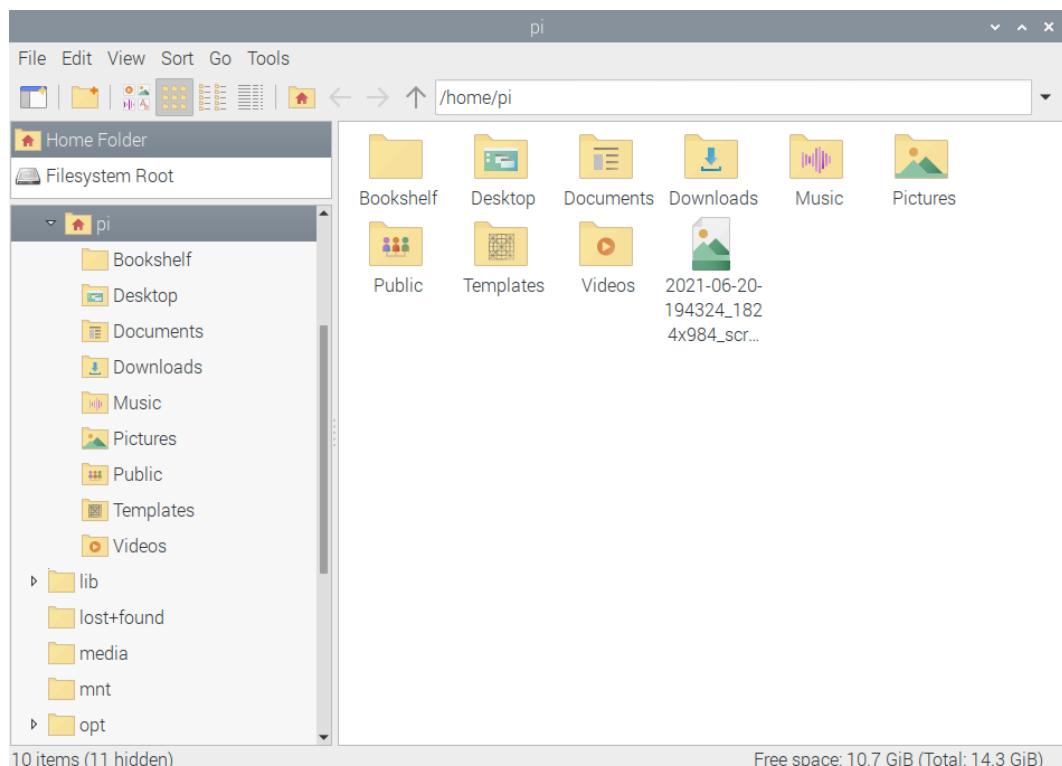
หน้าจอหลักของระบบในรูปที่ D.3 มีลักษณะคล้ายกับหน้าจอหลักของระบบปฏิบัติการอื่น ๆ เช่น ปุ่มเมนูหลัก แถบแสดงรายชื่อโปรแกรมที่กำลังทำงานอยู่ ปุ่มไอคอนของโปรแกรมที่นิยมใช้บ่อย (Favorites) ไอคอนแสดงการเชื่อมต่อสัญญาณ WiFi คลิ๊ก เป็นต้น สิ่งที่แตกต่าง คือ ตำแหน่งที่จัดวางของปุ่มหรือไอคอนเหล่านี้อาจแตกต่างกันได้ตามการปรับแต่งโดยผู้ใช้งาน ตารางต่อไปนี้เป็นการเปรียบเทียบระหว่าง

ไอคอนและปุ่มต่าง ๆ ของ Raspberry Pi OS และ Windows ซึ่งผู้อ่านจะต้องวัดเติมลงไปด้วยตนเองตามรายชื่อปุ่มด้านซ้าย

ปุ่ม	Raspberry Pi OS	Windows
เมนูหลัก(Main Menu)		
ปิด (Close)		
ย่อ (Minimize)		
ขยาย (Maximize)		

D.1.2 ไฟล์เมเนจเจอร์ (File Manager)

ไฟล์เมเนจเจอร์ คือ โปรแกรมสำหรับเบราว์ส (Browse) โครงสร้าง รายชื่อไดเรกทอรี รายชื่อไฟล์ต่าง ๆ ภายในอุปกรณ์เก็บรักษาข้อมูล เช่น การดูหน่วยความจำไมโคร SD เป็นต้น รูปที่ D.1 แสดงหน้าต่างของไฟล์เมเนจเจอร์ (File Manager) ขณะที่เปิดไดเรกทอรีชื่อ /home/pi ทางด้านขวา และโครงสร้างของอุปกรณ์เก็บรักษาข้อมูลทางด้านซ้าย โปรดสังเกตพื้นที่ว่าง (Free space) ของการดูหน่วยความจำ SD ที่ใช้งาน บริเวณมุมขวาล่างของหน้าต่าง



รูปที่ D.1: หน้าต่างของไฟล์เมเนจเจอร์ (File Manager) ขณะที่เปิดไดเรกทอรีชื่อ /home/pi

D.2 การใช้งานระบบผ่านทางโปรแกรม Terminal



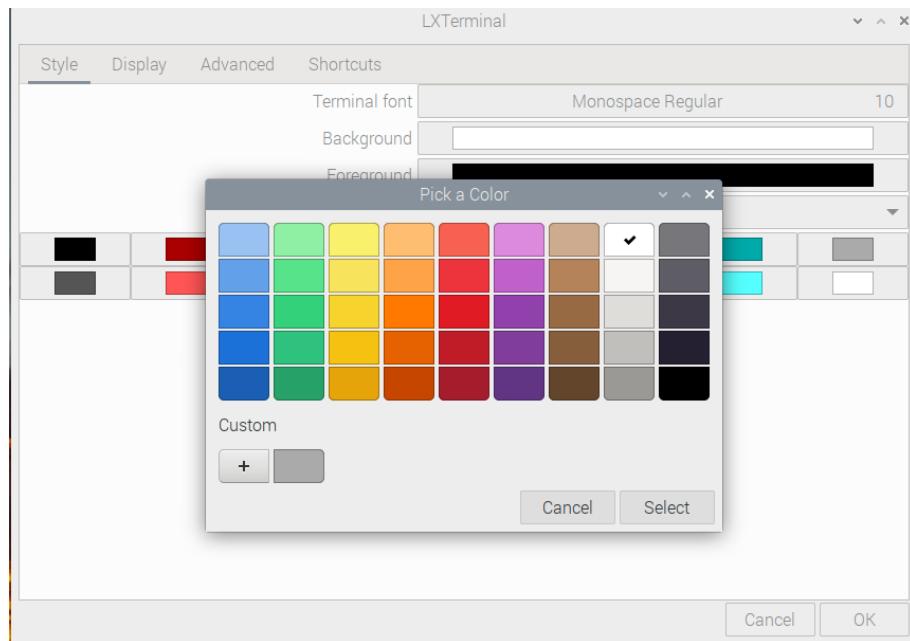
รูปที่ D.2: รูปไอคอนของโปรแกรม Terminal

ในอดีตผู้ใช้งานระบบยูนิเก็ตจะต้องคีย์คำสั่งต่าง ๆ ผ่านทางโปรแกรม Terminal เท่านั้น เรียกว่า การใช้แบบคอมมานด์ไลน์ (Command Line) ซึ่งผู้ใช้จะต้องฝึกฝนและจำคำสั่งต่าง ๆ ทำให้การใช้งานแบบคอมมานด์ไลน์ยุ่งยากและไม่น่าสนใจ เมื่อൺการใช้งานแบบ GUI เมื่อ่อนในปัจจุบัน แต่ผู้ใช้งานที่เชี่ยวชาญสามารถเข้าใจการทำงานได้ลึกซึ้งกว่า คำสั่งพื้นฐานและคำสั่งซัตดาวน์ในการทดลองนี้จะช่วยเสริมความเข้าใจของผู้อ่านได้เป็นอย่างดี โดยผู้ใช้สามารถเปิดโปรแกรม Terminal ด้วยการคลิกบนปุ่มที่มีรูปเหมือนไอคอนในรูปที่ D.2 บนแถบแสดงรายชื่อโปรแกรม (Taskbar) รูปที่ D.3 แสดงหน้าต่างของโปรแกรม Terminal ซึ่งผู้ใช้ยังได้ปรับแต่งสีพื้นและสีของตัวอักษรให้เหมาะสม



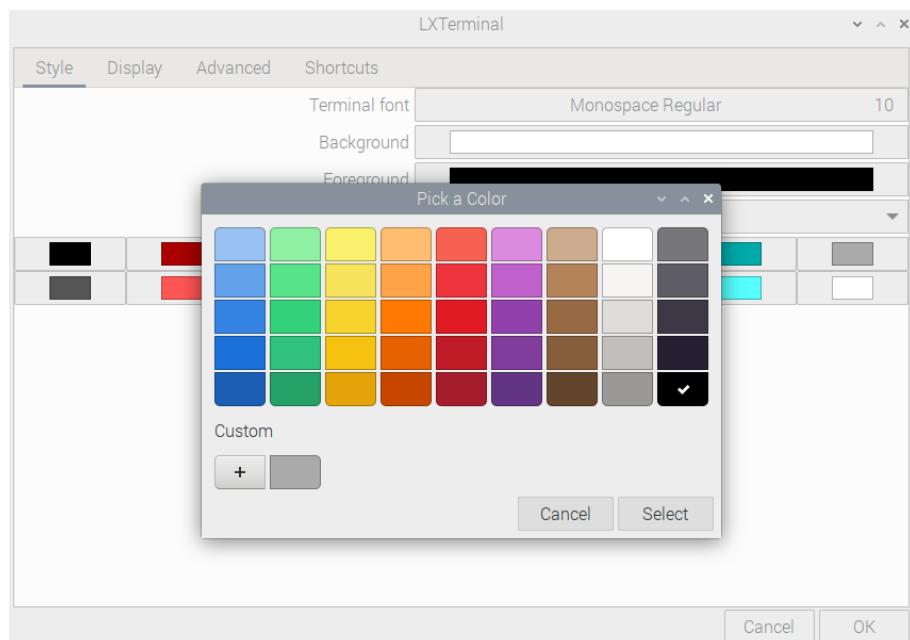
รูปที่ D.3: หน้าต่างของโปรแกรม Terminal ซึ่งสามารถปรับแต่งสีพื้นและสีของตัวอักษรได้

1. เปิดโปรแกรม Terminal บนเมนูหลัก
2. คลิกเมนู Edit -> Preferences
3. คลิกที่แท็บสีของ Background จากให้เลือกสีขาว ดังรูป คลิกปุ่ม Select



รูปที่ D.4: หน้าต่างปรับแต่งสีพื้น (Background)

4. คลิกที่แท็บสีของ Foreground จากให้เลือกสีดำ ดังรูป คลิกปุ่ม Select แล้วจึงคลิกปุ่ม OK ดังรูป



รูปที่ D.5: หน้าต่างปรับแต่งสีตัวอักษร (Foreground)

5. ทดสอบด้วยการปิดโปรแกรมแล้วเปิดอีกรอบว่า สีที่เลือกยังคงอยู่

D.2.1 คำสั่งพื้นฐานของระบบยูนิคซ์

ผู้อ่านสามารถฝึกใช้คำสั่งเหล่านี้บนโปรแกรมเทอร์มินัล (Terminal) ตามตารางต่อไปนี้ โปรดสังเกต สัญลักษณ์ \$ หมายถึง ตำแหน่งเริ่มต้นคำสั่งคอมมานด์ไลน์ในโปรแกรม Terminal

ลำดับที่	รายละเอียด	คำสั่ง
1	แสดงรายชื่อไฟล์และไดเรกทอรี	ls <parameter> Ex.: \$ ls แสดงรายชื่อไฟล์และไดเรกทอรีในไดเรกทอรีปัจจุบัน Ex.: \$ ls -l แสดงรายละเอียดต่าง ๆ ของไฟล์และไดเรกทอรีในไดเรกทอรีปัจจุบัน Ex.: \$ ls -la แสดงรายละเอียดต่าง ๆ ของไฟล์และไดเรกทอรีทั้งหมดในไดเรกทอรีปัจจุบัน โปรดสังเกตสัญลักษณ์ต่อไปนี้บริเวณสองแฉวบนสุดของผลลัพธ์ ”.” หมายถึง ไดเรกทอรีปัจจุบัน (current directory) ”..” หมายถึง ไดเรกทอรีที่อยู่เหนือขึ้นไป (parent directory)
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	แสดงชื่อไดเรกทอรีปัจจุบัน	pwd Ex.: \$ pwd
6	สร้างไดเรกทอรีใหม่	mkdir <directory_name> Ex.: \$ mkdir /home/pi/asm สร้างไดเรกทอรีใหม่ ชื่อ "asm" ภายใต้ไดเรกทอรี "/home/pi/" เพื่อใช้จัดเก็บไฟล์สำหรับการทดลองต่อไป
7	Change directory	cd <destination> Ex.: \$ cd /home/pi/asm โปรดสังเกตสัญลักษณ์ต่อไปนี้ในประโยชน์ /home/pi/asm "/" ตำแหน่งช้ายสุด หมายถึง ไดเรกทอรีราก (root directory) "/" ตำแหน่งถัดมา หมายถึง สัญลักษณ์คั่นระหว่างชื่อไดเรกทอรี

D.2.2 การขัตดาวน์ (Shutdown)

การรีบูต หรือ รีสตาร์ตเครื่อง มักใช้เรียกเมื่อระบบต้องการหลังการอัปเดตซอฟต์แวร์ต่าง ๆ ที่จำเป็น หรือ ผู้ใช้ต้องการแก้อาการต่าง ๆ โดย

- พิมพ์คำสั่ง **sudo reboot** ในหน้าต่าง Terminal เพื่อรีบูตบอร์ด Pi และระบบปฏิบัติการในกรณีที่ผู้ใช้ต้องการเริ่มต้นระบบใหม่

```
$ sudo reboot
```

ผู้อ่านสามารถรีบูตหรือรีสตาร์ทบอร์ดใหม่ด้วยคำสั่ง

```
$ 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 ซึ่งนักคอมพิวเตอร์ส่วนใหญ่นิยมใช้ศัพท์คำนี้ในการสั่งให้ระบบปฏิบัติการหรือโปรแกรมใด ๆ หยุดการทำงาน

โปรดรอไฟ LED สีเขียวที่ติดกับไฟ LED สีแดง กระพริบจนดับเสียก่อนจึงค่อยกดอแดปเตอร์ออกจากเต้าเสียบไฟ 220 โวลต์

D.3 ข้อมูลพื้นฐานของบอร์ด Pi

การใช้งานทางคอมมานด์ไลน์มีประโยชน์หลายด้าน เนื่องจากผู้ใช้สามารถเรียกใช้คำสั่งเกือบทั้งหมดในระบบ รวมถึงการเขียนโปรแกรมชেลล์สคริปต์ (Shell Script) เพื่อสั่งงานคอมมานด์ไลน์ได้อัตโนมัติ ผู้อ่านควรจะฝึกใช้ให้คล่องเพื่อเตรียมความพร้อมไปเป็นนักพัฒนาโปรแกรม และพัฒนาระบบต่อไป โดยการทดลองนี้จะใช้คำสั่งพิเศษอ่านค่าข้อมูลของซีพีयุและข้อมูลขั้นสูงอื่น ๆ

D.3.1 ข้อมูลพื้นฐานของซีพีyu

ผู้อ่านสามารถศึกษารายละเอียดเกี่ยวกับซีพีyuที่ใช้งานอยู่บนบอร์ด โดยใช้คำสั่ง

```
$ cat /proc/cpuinfo
```

จดผลลัพธ์ที่ได้จากบอร์ด Pi ลงในช่องที่กำหนดให้ ซึ่งอาจแตกต่างกันสำหรับผู้ใช้ Raspberry 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	รายละเอียดของการ์ดหน่วยความจำไมโคร SD
4	\$ cat /proc/version	รายละเอียดของระบบปฏิบัติการ
5	\$ vcgencmd measure_temp	อ่านค่าอุณหภูมิ ณ จุดต่าง ๆ
6	\$ vcgencmd measure_volts core	อ่านค่าโวลต์ของแกนประมวลผล
7	\$ vcgencmd measure_volts sram_c	อ่านค่าโวลต์ของ SD-RAM
8	\$ vcgencmd measure_volts sram_i	อ่านค่าโวลต์ของ SD-RAM I/O

ยกตัวอย่าง เช่น 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. จงบอกความแตกต่างระหว่างคำสั่ง cat และคำสั่ง ls
2. จงบอกความแตกต่างระหว่างคำสั่ง cp และคำสั่ง mv
3. คำสั่ง vcgencmd ย่อมาจากคำว่าอะไร
4. ชิป BCM2_____ บนบอร์ดมีจำนวนซีพียูกี่แกนประมวลผล
5. ชิป BCM2835 เกี่ยวข้องกับ ชิป BCM2_____ ในข้อก่อนหน้าอย่างไร
6. จงบอกหมายเลขรุ่น (CPU Revision) ของซีพียู ARM Cortex A_____ ที่ได้จากคำสั่ง cpuinfo
7. ในหัวข้อที่ [D.3.2](#) จงบวกขนาดของหน่วยความจำ MemAvail, Buffers, Cached เพื่อเปรียบเทียบกับ MemTotal ว่าแตกต่างกันหรือไม่ อย่างไร
8. ผู้อ่านสามารถตรวจสอบขนาดของ SDRAM ที่มีบนบอร์ดกับข้อมูลที่ได้จาก meminfo ในหัวข้อได้ และแปลงหน่วยคิกิไบต์ (KiB) เป็นกิกิไบต์ (GiB) ได้อย่างไร (โปรดศึกษาบทอภิธานศัพท์ [M.5](#))
9. จงบอกเวอร์ชัน (Version) และรายละเอียดอื่น ๆ ของระบบปฏิบัติการ Raspberry Pi OS ที่ติดตั้ง
10. จงบอกความต่างศักย์ของแกนประมวลผล หน่วยความจำภายในภาพ และอินพุต/เอาต์พุต และเปรียบเทียบกันว่าแตกต่างกันหรือไม่ อย่างไร
11. จงบอกอุณหภูมิของซีพียูและตำแหน่งอื่น ๆ บนบอร์ดว่าทำงานที่กีองศาเซลเซียส และเปรียบเทียบกันว่าแตกต่างกันหรือไม่ อย่างไร

ภาคผนวก E

การทดลองที่ 5 การพัฒนาโปรแกรมภาษา C บน ลินุกซ์

การทดลองนี้คัดว่าผู้อ่านผ่านหัวข้อที่ 3.2 และมีประสบการณ์การเขียนหรือพัฒนาโปรแกรมด้วยภาษา C มาแล้ว ผู้อ่านอาจมีความคุ้นเคยกับ IDE (Integrated Development Environment) จากการพัฒนาโปรแกรมและการดีบักโปรแกรมด้วยภาษา 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 พัฒนา ทดสอบ และอาจรวมถึงควบคุมซอฟต์แวร์สโคล์ดให้เป็นปัจจุบัน ขั้นตอนการทดลองนี้เริ่มต้นโดย

- ตรวจสอบภายในเครื่องว่ามีโปรแกรมชื่อ CodeBlocks ติดตั้งแล้วหรือไม่ โดยพิมพ์คำสั่งเหล่านี้ลงบนโปรแกรม Terminal

```
$ codeblocks
```

- หากติดตั้งแล้ว ให้ผู้อ่านข้ามไปข้อที่ 4 ได้ หากไม่มีโปรแกรม ผู้อ่านจะต้องติดตั้ง CodeBlocks โดยพิมพ์คำสั่งเหล่านี้ลงบนโปรแกรม Terminal

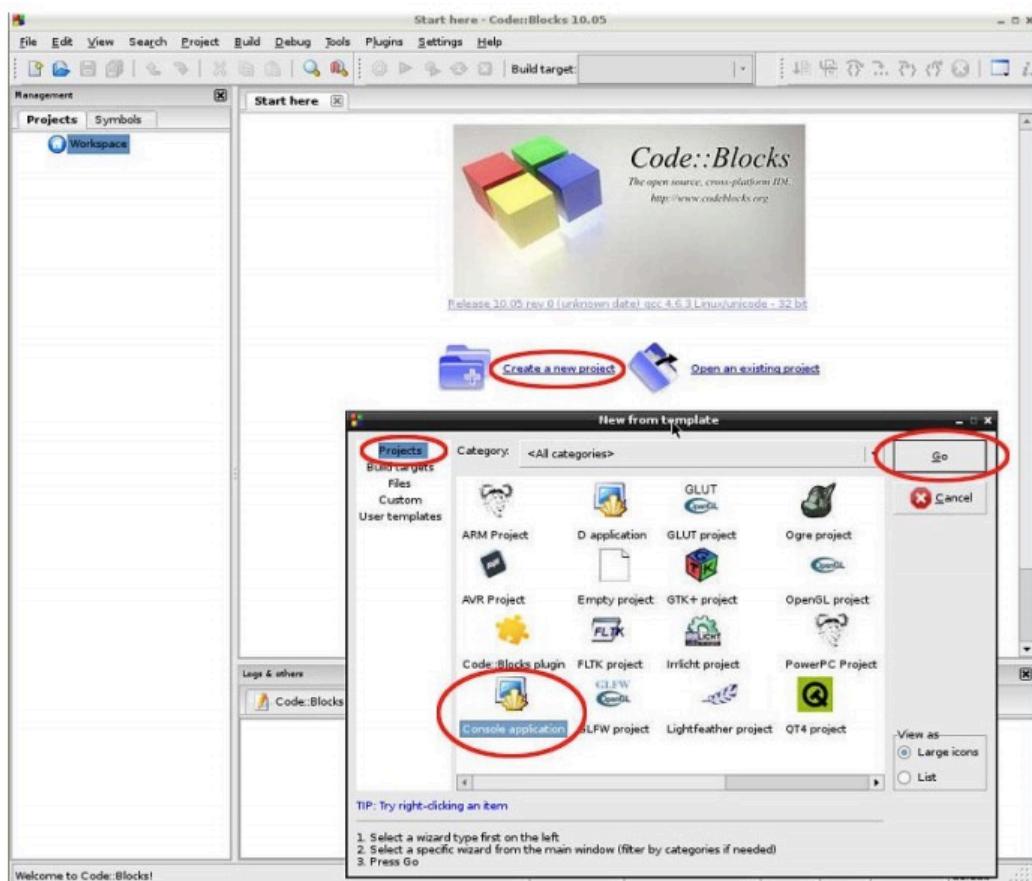
```
$ sudo apt-get install codeblocks
```

คำสั่ง sudo นำหน้าคำสั่งได้ ๆ นี้จะเป็นการเรียกใช้งานคำสั่งนั้นด้วยสิทธิ์ระดับ SuperUser การติดตั้งจะดาวน์โหลดโปรแกรมผ่านทางเครือข่ายอินเทอร์เน็ต และจำเป็นต้องใช้สิทธิ์ระดับสูงสุดนี้

- เมื่อติดตั้งเสร็จสิ้น พิมพ์คำสั่งนี้เพื่อเริ่มต้นใช้งาน CodeBlocks

```
$ codeblocks
```

- การใช้งาน CodeBlocks ครั้งแรกจะเป็นการติดตั้งค่า compiler plug-ins เป็น GCC หรือ GNU C Compiler.
- หน้าต่างหลักจะปรากฏขึ้น หลังจากนั้น ผู้อ่านควรกด "Create a new project" เพื่อสร้างโปรเจกต์ใหม่ในหน้าต่าง "New from template"



รูปที่ E.1: หน้าต่างเลือกชนิดโปรเจกต์ที่จะพัฒนาเป็นชนิด "Console application"

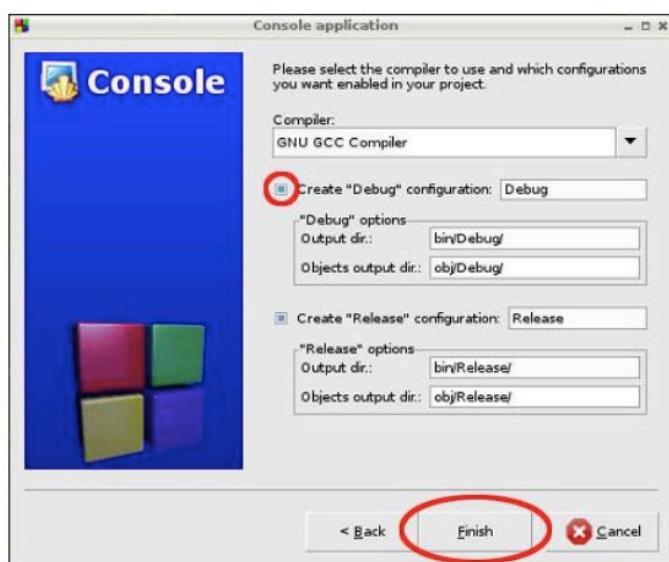
- เลือก "New Projects" ในช่องด้านซ้าย แล้วเลือก "Console application" ในช่องด้านขวาเพื่อสร้างโปรแกรมในรูปแบบเทกซ์mode (Text Mode) กดปุ่ม "Go" ตามรูปที่ E.1
- กดปุ่ม Next> เพื่อดำเนินการต่อ

8. หน้าต่าง "Console application" จะปรากฏขึ้น กดเลือกภาษา "C" เพื่อพัฒนาโปรแกรมแล้วกดปุ่ม "Next >" ตามรูปที่ E.2)



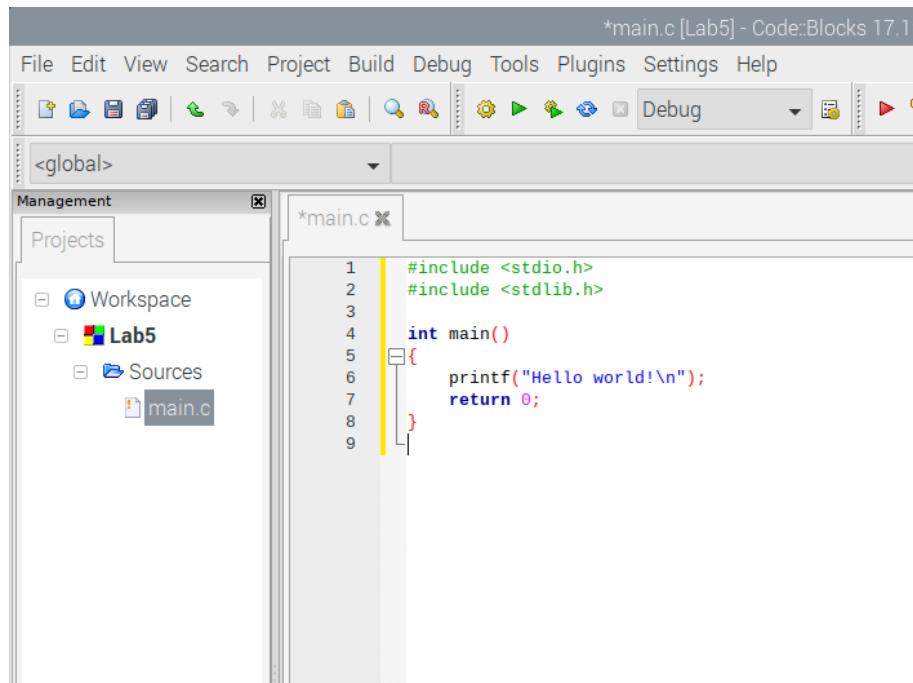
รูปที่ E.2: หน้าต่างเลือกภาษา C หรือ C++ สำหรับโปรเจกต์ที่จะพัฒนา

9. กรอกชื่อโปรเจกต์ใหม่ชื่อ Lab5 ในช่อง Project title: และกรอกชื่อไดเรกทอรี /home/pi/asm/ ในช่อง Folder to create project in: โปรดสังเกตข้อความในช่อง Project filename: ว่าตรงกับ Lab5.cbp หรือไม่
10. กดปุ่ม "Next >" เพื่อดำเนินการต่อ และสุดท้ายจะเป็นขั้นตอนการเลือกคอนฟิกกูเรชัน (Configuration) สำหรับคอมไพล์เตอร์ในรูปที่ E.3 โดย Debug เหมาะสำหรับการเริ่มต้นและแก้ไขข้อผิดพลาด แล้วจึงกดปุ่ม "Finish" เมื่อเสร็จสิ้นการติดตั้ง



รูปที่ E.3: การเลือกคอนฟิกกูเรชัน (Configuration) Debug สำหรับคอมไابل์เตอร์ GNU GCC ในโปรเจกต์ Lab5

11. คลิกช้ายบันชื่อ Lab5 ในหน้าต่าง Management/Workspace ด้านซ้ายมือ เพื่อขยายไดเรกทอรี Sources แล้วจึงคลิกบนไฟล์ไอคอนชื่อ main.c



รูปที่ E.4: การเปิดอ่านไฟล์ main.c ภายใต้โปรเจกต์ 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 โดยสังเกตจากหน้าต่างย่อยด้านล่างสุด

14. รันโปรแกรมเพื่อทดสอบการทำงาน

E.2 การดีบัก (Debugging) โดยใช้ IDE

การดีบักโปรแกรม คือ การตรวจสอบการทำงานของโปรแกรมอย่างละเอียด CodeBlocks รองรับการดีบัก ผ่านเมนู Debug ผู้อ่านสามารถเริ่มต้นโดย

1. กด Debug บนเมนูแถบบนสุด เลือก Active Debuggers GDB/CDB Debugger เป็นค่าดีฟอลท์ (Target's Default)
2. เลื่อนเมาส์ซอร์ (Cursor) ไปยังบรรทัดที่ต้องการศึกษา กดปุ่ม F5 เพื่อตั้งเบรกพอยน์ (Break Point) ตรงบรรทัดปัจจุบันของเมาส์ซอร์ โปรดสังเกตต้นประโยคด้านซ้ายสุดจะมีวงกลมสีแดงปรากฏขึ้น และเมื่อกด F5 อีกครั้งวงกลมสีแดงจะหายไป เรียกว่า การท็อคเกิล (Toggle) เบรกพอยน์ กด F5 อีกครั้งเพื่อสร้างวงกลมสีแดงตรงบรรทัดที่สนใจเพียงจุดเดียวเท่านั้น จำภาพหน้าต่างที่ได้วางไว้ได้คำสั่งนี้เพื่อให้ตรวจสอบ
3. กดปุ่ม F8 (Start/Continue) บนคีย์บอร์ดเพื่อรันโปรแกรมอีกรอบ โปรแกรมจะรันไปจนหยุดตรงประโยคที่มีวงกลมสีแดงนั้น โปรดสังเกตสัญลักษณ์สามเหลี่ยมสีเหลืองซ้อนทับกันอยู่ หลังจากนั้น กดปุ่ม F7 (Next line) เพื่อดำเนินการต่อทีละบรรทัด
4. เลื่อนเมาส์ซอร์ไปยังประโยคที่มีวงกลมสีแดง กดปุ่ม F5 บนคีย์บอร์ดเพื่อปลดวงกลมสีแดงออก หรือยกเลิกเบรกพอยน์
5. เริ่มต้นการดีบักใหม่เพื่อศึกษาการทำงานของปุ่ม F4 (Run to cursor) โดยเลื่อนเมาส์ซอร์ไปวางบนประโยคที่สนใจ กดปุ่ม F4 และสังเกตว่าสามเหลี่ยมสีเหลืองจะปรากฏหน้าประโยค เพื่อระบุว่าเครื่องรันนามาถึงประโยคนี้แล้ว
6. กดปุ่ม F8 เพื่อรันต่อไป จนสิ้นสุดการทำงานของโปรแกรม
7. ใช้โปรแกรมไฟล์เมเนเจอร์ค้นหาในไดเรกทอรี `/home/pi/asm/Lab5` ว่าไฟล์ `main.o` ที่เป็นไฟล์อ๊อบเจกต์อยู่ในไดเรกทอรีใด
8. ใช้โปรแกรมไฟล์เมเนเจอร์ค้นหาในไดเรกทอรี `/home/pi/asm/Lab5` ว่าไฟล์ `Lab5` ที่เป็นไฟล์โปรแกรมหรือไฟล์ Executable อยู่ในไดเรกทอรีใด

E.3 การพัฒนาโดยใช้ประโยชน์คำสั่ง (Command Line)

ผู้อ่านควรเข้าใจคำสั่งพื้นฐานในการแปลงโปรแกรมภาษา C ที่สร้างขึ้นใน CodeBlocks ก่อนหน้านี้ ตามขั้นตอนต่อไปนี้

1. เปิดโปรแกรม Terminal หน้าต่างใหม่ แล้วย้ายไฟล์ที่ได้จากห้องไปยัง `/home/pi/asm/Lab5` โดยใช้คำสั่ง `cd`
2. ทำการคอมไพล์ (Compile) ไฟล์ซอร์ฟโค้ดให้เป็นไฟล์อีบเจกต์ (.o) โดยเรียกใช้คอมไเพเลอร์ชื่อ `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) และแปลงไฟล์อีบเจกต์เป็นไฟล์โปรแกรม (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. เปรียบเทียบ ผลลัพธ์ ที่ ปรากฏ ขึ้น ว่า ตรง กับ ผล การ รัน ใน CodeBlocks หรือ ไม่
..... เพราะเหตุใด

E.4 โครงสร้างของ Makefile

นอกเหนือจากการพัฒนาโปรแกรมด้วย IDE แล้ว การพัฒนาด้วย Makefile จะช่วยให้นักพัฒนาเมื่อสมัครเล่นและเมื่ออาชีพดำเนินการได้ถูกต้องและรวดเร็ว ไฟล์ซึ่ง Makefile เป็นไฟล์อักษรหรือเทกซ์ไฟล์ (text file) ง่าย ๆ ที่อธิบายความสัมพันธ์ระหว่างไฟล์ซอฟต์แวร์ต่าง ๆ ไฟล์อ้อมเบกต์ และไฟล์โปรแกรม แต่ละบรรทัดจะมีโครงสร้างดังนี้

```
target : prerequisites ...
<tab>recipe
<tab>    ...
<tab>...
```

- target หมายถึง ชื่อไฟล์ที่จะถูกสร้างขึ้น โดยอาศัยไฟล์ต่าง ๆ จากส่วนที่เรียกว่า prerequisites นอกจากชื่อไฟล์แล้ว คำสั่ง 'clean' สามารถใช้เป็น target ได้ จึงนิยมใช้สำหรับลบไฟล์ต่าง ๆ ที่ไม่ต้องการ
- recipe หมายถึง คำสั่งหรือการกระทำที่จะใช้รายชื่อไฟล์ใน prerequisites นำมารังสรรค์ไฟล์ target ได้สำเร็จ โดยแต่ละบรรทัดจะต้องเริ่มต้นด้วยปุ่ม tab เสมอ

E.5 การพัฒนาโดยใช้ Makefile

ตัวอย่างนี้เป็นการสร้าง Makefile เพื่อใช้คอมไฟล์และลิงก์โปรแกรมเดิมที่เรามีอยู่ ผู้อ่านจะได้เข้าใจกลไกการทำงานที่ง่ายที่สุดก่อน หลังจากนั้นผู้อ่านสามารถศึกษาเพิ่มเติมด้วยตนเองได้จากเว็บไซต์หรือตัวอย่างโปรแกรมโอเพนซอร์สที่ซับซ้อนขึ้นเรื่อย ๆ 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 อย่างใด อย่างหนึ่งโดยไม่มีนามสกุล 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 โปรดสังเกตสิ่งของข้อไฟล์ต่าง ๆ ว่ามีสีอะไรบ้าง และบ่งบอกอะไรมาสิ่นนั้น ๆ

7. พิมพ์คำสั่งนี้ใน Terminal

```
$ ./Lab5
```

เพื่อรันโปรแกรม Lab5 ให้ชีพิญ ปฏิบัติ ตาม โดย . หมายถึง / หมายถึง วางแผนหน้าต่างที่ได้วางไว้ให้คำสั่งนี้เพื่อให้ตรวจสอบ

8. คลิกบนลิงก์ต่อไปนี้ sunshine2k.de เพื่อเปิดเบราว์เซอร์และอัปโหลดไฟล์ Lab5 ที่ได้จากการคอมไพล์และลิงก์ก่อนหน้านี้ ตรวจสอบค่า Status: File successfully loaded หรือไม่
9. เปิดเบราว์เซอร์ให้เต็มจอแล้วเลื่อนหน้าจอแสดงผลขึ้นเพื่อเปรียบเทียบกับโครงสร้างของไฟล์ ELF ในรูปที่ 3.14 แคปเจอร์หน้าจอบริเวณแท็บเช็กเม้นต์และดาต้าเช็กเม้นต์ของ Lab5 วางแผนหน้าต่างที่ได้วางไว้ใต้คำสั่งนี้เพื่อให้ตรวจสอบ

E.6 การตรวจจับ Overflow คณิตศาสตร์เลขจำนวนเต็มฐานสอง

E.6.1 เลขจำนวนเต็มฐานสองไม่มีเครื่องหมาย

หัวข้อที่ 2.3.1 กล่าวถึงการบวกเลขจำนวนเต็มชนิดไม่มีเครื่องหมาย 2 จำนวน ผลลัพธ์ที่ได้จะไม่มีเครื่องหมายด้วยเช่นกัน แต่การบวกเลขขนาดใหญ่ที่เข้าใกล้ค่าสูงสุด สามารถเกิดความผิดพลาดได้ เรียกว่า การเกิดโอเวอร์โฟล์ (Overflow) ในสมการที่ (2.52) 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 프로그ตัวแปรจึงตั้งค่าเริ่มต้น unsigned int i=1;
3. การวนลูปเพิ่มค่า i=i+1 จน i มีค่าเป็นศูนย์แล้วแสดงผลค่าของ i มาทางหน้าจอ เมื่อเกิดเหตุการณ์ อะไร เพราะเหตุใด
4. จับภาพหน้าต่างที่ได้วางไว้ให้คำสั่งนี้เพื่อให้ตรวจสอบ

E.6.2 เลขจำนวนเต็มฐานสองมีเครื่องหมายชนิด 2's Complement

หัวข้อที่ [2.3.2](#) กล่าวถึง การ บวก เลขจำนวนเต็ม ชนิด มี เครื่องหมาย 2 จำนวน ผลลัพธ์ที่ได้ จะ มี เครื่องหมายด้วย เช่น กัน แต่ การบวกเลขขนาดใหญ่ที่เข้าใกล้ค่าสูงสุด สามารถเกิดความผิดพลาดได้ เรียกว่า การเกิดโอเวอร์ฟลอร์ (Overflow) ในสมการที่ [\(2.56\)](#) ซึ่งเป็นผลสืบเนื่องจากวงจรดิจิทัลที่สามารถ ประมวลผลได้จำกัด ตามจำนวนบิตข้อมูลสูงสุดที่ทำได้ ในตัวอย่างการแปลงเลขฐานสองเป็นฐานสิบที่ได้ แสดงไปแล้ว ยกตัวอย่างเช่น การวนรอบหรือวนลูป (Loop) เพิ่มค่าอย่างต่อเนื่องโดยไม่ระวัง ตามประ喜悦 ในภาษา 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 เพราะเหตุใด
4. จับภาพหน้าต่างที่ได้วางไว้ใต้คำสั่งนี้เพื่อให้ตรวจสอบ

E.7 กิจกรรมท้ายการทดลอง

1. จงเปรียบเทียบไฟล์การพัฒนาโปรแกรมในภาคผนวกนี้กับรูปที่ 3.9
2. ใน Terminal จงย้ายไฟล์ที่ได้รับจากอาจารย์ไปที่ /home/pi/asm/Lab5

\$ ls -la

เพื่ออ่านรหัสสีของชื่อไฟล์ต่าง ๆ

3. จงพัฒนาโปรแกรมภาษา C โดยประการตัวแปรและตั้งค่าเริ่มต้น int i=-1 และให้วนลูปลดค่า i=i-1 จน i มีค่าเป็นบวกแล้วแสดงผลค่าของ i ออกมาทางหน้าจอ โดยใช้โปรแกรมในหัวข้อที่ E.6.2 เป็นต้นแบบ
4. จงพัฒนาโปรแกรมภาษา C ให้สามารถอ่านไฟล์ Makefile เพื่อแสดงตัวอักษรในไฟล์ทีละตัวและค่ารหัส ASCII ฐานสิบหกของตัวอักษรนั้นบนหน้าจอ แล้วปิดไฟล์เมื่อเสร็จสิ้น
5. จงพัฒนาโปรแกรมภาษา 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 ด้วยภาษา C/C++ ด้วย CodeBlocks ดังนั้น การทดลองมีวัตถุประสงค์เหล่านี้

- เพื่อให้เข้าใจการพัฒนาและติดตั้ง IDE ชื่อ CodeBlocks บนระบบปฏิบัติการตระกูลยูนิกซ์
- เพื่อให้เข้าใจความแตกต่างระหว่างการพัฒนาโปรแกรมภาษาแอสเซมบลีด้วย IDE และ Makefile

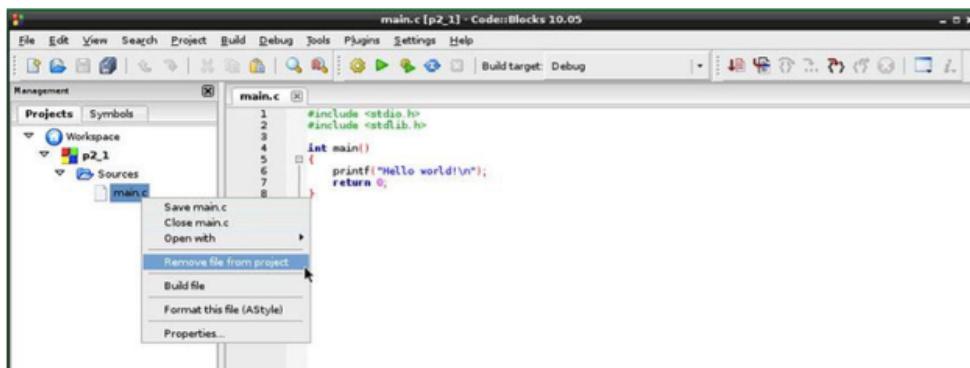
F.1 การพัฒนาโดยใช้ IDE

- พิมพ์คำสั่งนี้ในโปรแกรม Terminal เพื่อเริ่มต้นใช้งาน CodeBlocks

```
$ codeblocks
```

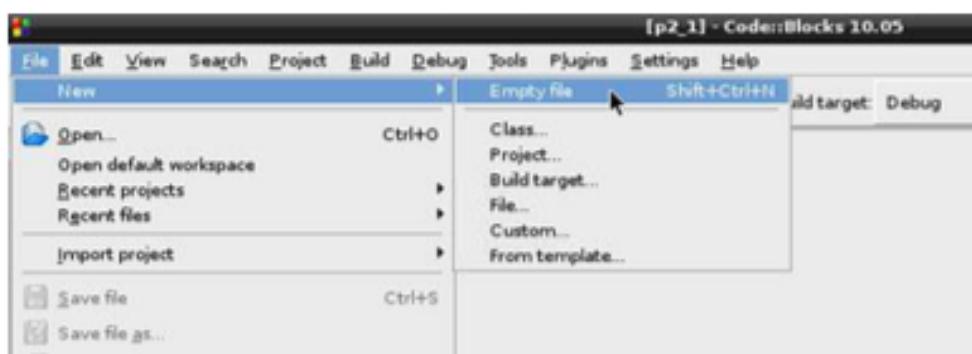
- หน้าต่างหลักจะปรากฏขึ้น หลังจากนั้น ผู้อ่านสามารถสร้างโปรเจกต์ใหม่โดยเลือก "Create a new project" ในช่องด้านซ้าย แล้วเลือก "Console application" ในช่องด้านขวาเพื่อสร้างโปรแกรม
- กรอกชื่อโปรเจกต์ใหม่ชื่อ Lab6 ในช่อง Project title: และกรอกชื่อไดเรกทอรี /home/pi/asm ในช่อง Folder to create project in: โปรดสังเกตข้อความในช่อง Project filename: ว่าตรงกับ Lab6.cbp ใช่หรือไม่ แล้วจึงกด Next>
- โปรแกรม CodeBlocks จะสร้างไดเรกทอรีต่าง ๆ ภายใต้ไดเรกทอรีชื่อ /home/pi/asm/Lab6/
- กดปุ่ม "Next>" เพื่อดำเนินการต่อ และสุดท้ายจะเป็นขั้นตอนการเลือกค่า Configuration สำหรับคอมไฟเลอร์ เลือก option Debug หมายสำหรับการเริ่มต้นและแก้ไขข้อผิดพลาด แล้วจึงกดปุ่ม "Finish" เมื่อเสร็จสิ้น

6. คลิก ชื่อ Workspace ในหน้าต่าง ด้านซ้าย เพื่อขยายโครงสร้างโปรเจกต์แล้วค้นหาไฟล์ชื่อ "main.c" คลิกขวาบนชื่อไฟล์แล้วเลือกเมนู "Remove file from project" ตามรูปที่ F.1



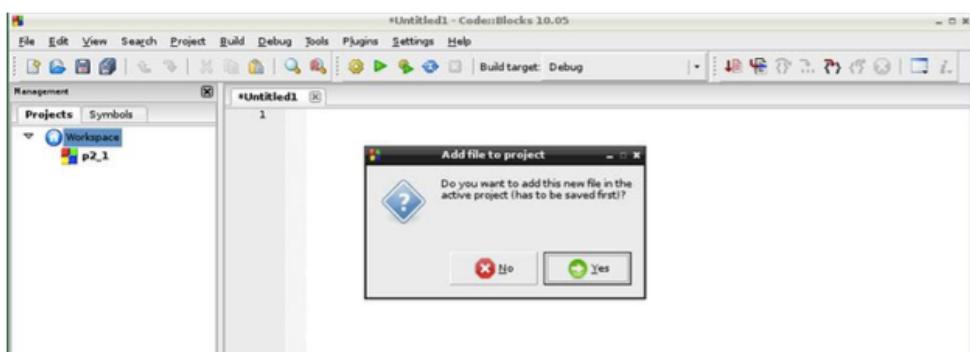
รูปที่ F.1: การรื้อไฟล์ main.c ออกจากโปรเจกต์

7. เพิ่มไฟล์ใหม่ลงในโปรเจกต์โดยกดเมนู File->New->Empty file ตามรูปที่ F.2



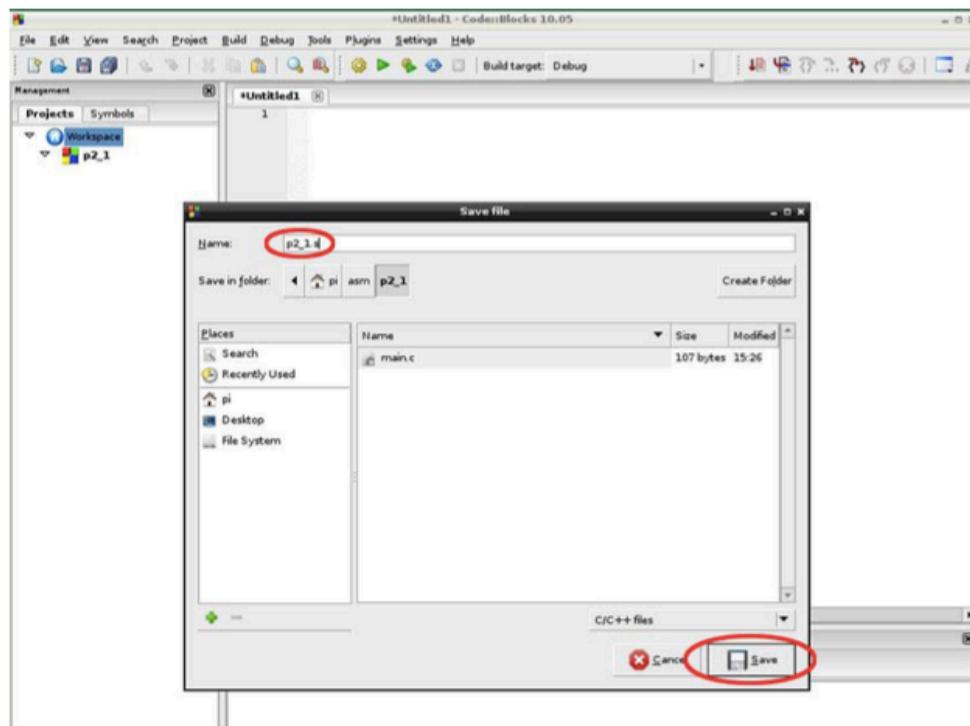
รูปที่ F.2: การเพิ่มไฟล์ใหม่ลงในโปรเจกต์

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 เข้าไปในโปรเจกต์
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. อ่านและบันทึกประโยคที่เกิดขึ้นในหน้าต่าง Terminal ที่ปรากฏขึ้นมา

F.2 การดีบักโปรแกรมโดยใช้ IDE

1. ในไฟล์ main.s เลื่อนเครื่องเซอร์ไพร์ทัตที่มีคำสั่ง ORR R0, R1, R2 คลิกเมนู Debug->breakpoint หรือกดปุ่ม F5 ผู้อ่านจะสังเกตว่ากลมสีแดงปรากฏขึ้นด้านซ้ายของคำสั่ง ORR
2. กด เมนู Debug->Debugging Windows->CPU Registers เพื่อแสดงค่าของ CPU register ในหน้าต่างที่ปรากฏขึ้นมาเพิ่มเติม
3. เมื่อพร้อมแล้ว ผู้อ่านสามารถเริ่มต้นการดีบักโดยกด เมนู Debug->Start/Continue หรือกดปุ่ม F8 โปรแกรมจะเริ่มต้นทำงานตั้งแต่ประযุกแรกจนหยุดที่คำสั่ง ORR R0, R1, R2
4. อ่านและบันทึกค่าของ R0 และ PC ในหน้าต่าง CPU Registers
5. ประมวลผลคำสั่งถัดไปโดยกด เมนู 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 การพัฒนาโดยใช้ประยุคคำสั่ง (Command Line)

ผู้อ่านควรเข้าใจคำสั่งพื้นฐานในการแปลงโปรแกรมภาษาแอสเซมบลีที่สร้างขึ้นใน CodeBlocks ก่อนหน้านี้ ตามขั้นตอนต่อไปนี้

1. ใช้โปรแกรมไฟล์เมเนจอร์เพื่อเบรน์ไฟล์ในไดเรกทอรี /home/pi/asm/Lab6
2. ดับเบิล คลิก บน ชื่อ ไฟล์ main.s เพื่อ เปิด อ่าน ไฟล์ และ เปรียบ เทียบ กับ ไฟล์ ที่ เขียน ใน โปรแกรม CodeBlocks
3. เปิดโปรแกรม Terminal หน้าต่างใหม่ แล้วป้ายไดเรกทอรีปัจจุบัน (cd: change directory) ไปยัง /home/pi/asm/Lab6 โดยใช้คำสั่ง

```
$ cd /home/pi/asm/Lab6
```

4. แปลไฟล์ซอร์สโค๊ดให้เป็นไฟล์อ๊อบเจกต์ โดยเรียกใช้คำสั่ง as (assembler) ดังนี้

```
$ as -o main.o main.s
```

5. ใช้คำสั่ง ls -la ใน Terminal เพื่อค้นหาไฟล์อ๊อบเจกต์ชื่อ main.o ว่ามีจริงหรือไม่

6. ทำการลิงก์และแปลงไฟล์อ็อบเจกต์เป็นไฟล์โปรแกรมโดย

```
$ gcc -o Lab6 main.o
```

7. ใช้คำสั่ง ls -la ใน Terminal เพื่อค้นหาไฟล์โปรแกรมชื่อ Lab6 ว่ามีจริงหรือไม่

8. เรียกโปรแกรม Lab6 โดยพิมพ์

```
$ ./Lab6
```

```
$ echo $?
```

9. เปรียบเทียบหมายเลขที่ปรากฏขึ้นว่าตรงกับผลการรันใน IDE หรือไม่ อย่างไร

F.4 การพัฒนาโดยใช้ Makefile

การใช้ makefile สำหรับพัฒนาโปรแกรมภาษาแอสเซมบลีคล้ายกับการทดลองที่ 5 ในภาคผนวก E ก่อนหน้านี้

1. เปิดไดเรกทอรี /home/pi/asm/Lab6 ด้วยโปรแกรมไฟล์เมเนจอร์
2. กดปุ่มขวาบนมาส์ในพื้นที่ไดเรกทอรีเพื่อสร้างไฟล์เปล่าใหม่ (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. ลบไฟล์อ็อบเจกต์ที่มีอยู่โดยใช้คำสั่ง

```
$ make clean
```

ในโปรแกรม Terminal เพื่อเปรียบเทียบหลังจากที่รันคำสั่ง make clean

6. ใช้คำสั่ง ls -la ใน Terminal ค้นหาไฟล์อืบเจกต์ main.o และ Lab6 ว่าถูกลบหรือไม่
7. ทำการแอสเซมเบิล main.s โดยใช้คำสั่ง make ในโปรแกรม Terminal และขอให้สังเกตวันเวลาของไฟล์ต่าง ๆ

```
$ make
```

8. ใช้คำสั่ง ls -la ใน Terminal เพื่อค้นหาไฟล์ชื่อ main.o และ Lab6 ว่ามีจริงหรือไม่

9. เรียกโปรแกรม Lab6 โดยพิมพ์

```
$ ./Lab6
```

```
$ echo $?
```

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 เป็นตั้งนี้ จดบันทึกผลการทดสอบและอธิบาย

```
.data
.balign 4
var1: .word 1
.text
.global main
main:
    MOV R1, #2
    LDR R2, var1addr
    STR R1, [R2]
    LDR R0, [R2]
    BX LR
var1addr: .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

ตัวแปรต่าง ๆ 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 เนื่องจากฟังก์ชันคือชุดคำสั่งย่อยที่ฟังก์ชัน main() ในภาษา C หรือ main: ในภาษาแอสเซมบลีเป็นผู้เรียกใช้ และเมื่อทำงานเสร็จสิ้น ฟังก์ชันนั้นจะต้องรีเทิร์นกลับมาหาฟังก์ชัน main() หรือ main: ในที่สุด ดังนั้น ตัวแปรชนิดโลคอล จึงใช้พื้นที่จัดเก็บค่าในสแต็กเพرمภายในสแต็กเซ็กเมนต์แทน เพราสแต็กเพرم จะมีการจองพื้นที่ (PUSH) และคืนพื้นที่ (POP) ในรูป

แบบ Last In First Out ตามที่อธิบายในหัวข้อที่ 3.3.3 ทำให้ไม่จำเป็นต้องใช้พื้นที่ในบริเวณด้าม เช็คเมนต์ ผู้อ่านสามารถทำความเข้าใจหัวข้อนี้เพิ่มเติมในการทดลองที่ 8 ภาคผนวก H

G.1.1 การโหลดค่าตัวแปรเดียวจากหน่วยความจำมาพักในรีจิสเตอร์

1. ย้ายไดเรกทอรีไปยัง `/home/pi/asm` โดยใช้คำสั่ง `$ cd /home/pi/asm`
2. สร้างไดเรกทอรี `Lab7` โดยใช้คำสั่ง `$ mkdir Lab7`
3. ย้ายไดเรกทอรีเข้าไปใน `Lab7`
4. ตรวจสอบว่าไดเรกทอรีปัจจุบันโดยใช้คำสั่ง `pwd`
5. สร้างไฟล์ `Lab7_1.s` ตามขอร์สโค้ดต่อไปนี้ ผู้อ่านสามารถข้ามประโยชน์คอมเมนต์ได้ เมื่อทำความเข้าใจแต่ละคำสั่งแล้ว

```
.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 ภายในไดเรกทอรี Lab7 และกรอกคำสั่งดังนี้

Lab7_1:

```
gcc -o Lab7_1 Lab7_1.s
```

7. ทำการ make และรันโปรแกรมโดยใช้คำสั่ง

```
$ make Lab7_1
```

```
$ ./Lab7_1
```

```
$ echo $?
```

8. สร้างไฟล์ Lab7_2.s ตามโค้ดต่อไปนี้จากไฟล์ Lab7_1.s ผู้อ่านสามารถข้ามประโยชน์คอมเม้นต์ได้ เมื่อทำการเข้าใจแต่ละคำสั่งแล้ว

```
.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. เพิ่มประโยคต่อไปนี้ใน makefile ให้รองรับ Lab7_2

Lab7_2:

```
gcc -o Lab7_2 Lab7_2.s
```

10. ทำการ make และรันโปรแกรมโดยใช้คำสั่ง

```
$ make Lab7_2
$ ./Lab7_2
$ echo $?
```

บันทึกผลและอธิบายผลที่เกิดขึ้นเพื่อเปรียบเทียบกับข้อที่แล้ว

G.1.2 การใช้งานตัวแปรชุดหรืออาร์เรย์ ชนิด word

ภาษาแอสเซมบลี จะกำหนดชนิดตามหลังชื่อตัวแปร เช่น .word, .hword, และ .byte ใช้กำหนดขนาดของตัวแปรนั้น ๆ ขนาด 32, 16 และ 8 บิตตามลำดับ ยกตัวอย่าง คือ:

```
numbers: .word 1, 2, 3, 4
```

เป็นการประกาศและตั้งค่าตัวแปรชนิดอาร์เรย์ของ word ซึ่งต้องการพื้นที่ 4 ไบต์ต่อข้อมูลหนึ่งตำแหน่ง ซึ่งจะตรงกับประโยคต่อไปนี้ในภาษา C

```
int numbers={1, 2, 3, 4}
```

1. สร้างไฟล์ Lab7_3.s ตามโค้ดต่อไปนี้ ผู้อ่านสามารถข้ามประযุคคอมเมนต์ได้ เมื่อทำความเข้าใจแต่ละคำสั่งแล้ว

```
.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

```

2. เพิ่มประโยชน์ต่อไปนี้ใน makefile ให้รองรับ Lab7_3

Lab7_3:

```
gcc -o Lab7_3 Lab7_3.s
```

3. ทำการ make และรันโปรแกรมโดยใช้คำสั่ง

```

$ make Lab7_3
$ ./Lab7_3
$ echo $?

```

G.1.3 การใช้งานตัวแปรอาร์เรย์ชนิด byte

คำสั่ง LDRB ทำงานคล้ายกับคำสั่ง LDR แต่เป็นการอ่านค่าของตัวแปรอาร์เรย์ชนิด byte

1. สร้างไฟล์ Lab7_4.s ตามโค้ดต่อไปนี้ ผู้อ่านสามารถข้ามประโยชน์คอมเมนต์ได้ เมื่อทำความเข้าใจแต่ละคำสั่งแล้ว

```

.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
```

2. เพิ่มประโยคต่อไปนี้ใน makefile ให้รองรับ Lab7_4

Lab7_4:

```
gcc -o Lab7_4 Lab7_4.s
```

3. ทำการ make และรันโปรแกรมโดยใช้คำสั่ง

```
$ make Lab7_4
$ ./Lab7_4
$ echo $?
```

G.1.4 การเรียกใช้ฟังก์ชันและตัวแปรชนิดประโยครหัส ASCII

ฟังก์ชันสำเร็จรูปที่เข้าใจง่ายและใช้สำหรับเรียนรู้การพัฒนาโปรแกรมภาษา C เป็นต้น คือ ฟังก์ชัน printf ซึ่งถูกกำหนดอยู่ในไฟล์ヘดเดอร์ stdio.h ตามตัวอย่างซอฟต์แวร์โค้ด ในรูปที่ 3.9 และการทดลองที่ 5 ภาคผนวก E ในการทดลองต่อไปนี้ ผู้อ่านจะสังเกตเห็นว่าการเรียกใช้ฟังก์ชัน printf ในภาษาแอสเซมบลี โดยอาศัยตัวแปรชนิดประโยค (String) ในรูปที่ 2.11 โดยใช้คำสำคัญ (Key Word) เหล่านี้ คือ .ascii และ .asciz ตัวแปรชนิด asciz จะมีตัวอักษรพิเศษ เรียกว่า อักขระนัลล์ NULL หรือ /0 ปิดท้ายประโยคเสมอ และอักขระ NULL จะมีรหัส ASCII เท่ากับ 00₁₆ ตามตารางรหัสแอสกี ในรูปที่ 2.12

1. กรอกคำสั่งต่อไปนี้ลงในไฟล์ใหม่ชื่อ Lab7_5.s และทำความสะอาดให้ประโยคคอมเมนต์แต่ละบรรทัด

```
.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 ตามลำดับไปยังฟังก์ชันที่ถูกเรียก (Callee Function) ฟังก์ชันบางตัวต้องการจำนวนพารามิเตอร์มากกว่า 4 ค่า โปรแกรมเมอร์สามารถส่งพารามิเตอร์ผ่านทางสแต็กโดยคำสั่ง 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) ในสแต็กโดยเฉพาะรีจิสเตอร์ LR ก่อนเรียกใช้ฟังก์ชัน LR ตามที่อธิบายในหัวข้อที่ [4.8.2](#)

ผู้อ่านสามารถสำเนาซอฟต์แวร์สโคดด์ในการทดลองที่แล้วมาปรับแก้เป็นการทดลองนี้ได้

1. 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. มีการแบ่งกอปปี้ค่าของ LR ลงในสเตก หรือไม่ หากไม่มีแล้วในการทดลองเก็บค่าของ LR ไว้ที่ใด เพราะเหตุใด

7. วิธีการแบ่งกอปปี้ค่า LR ใน การทดลองสามารถใช้กับฟังก์ชันเรียกซ้ำ (Recursive Function) ได้หรือไม่ เพราะเหตุใด

G.3 กิจกรรมท้ายการทดลอง

1. จงเปรียบเทียบการเรียกใช้ฟังก์ชัน printf และ scanf ในภาษา C จากการทดลองที่ 5 ภาคผนวก E กับการทดลองนี้ด้านการส่งพารามิเตอร์

2. จงบอกความแตกต่างระหว่างการส่งค่าพารามิเตอร์แบบ Pass by Values และ Pass by Reference
3. จงยกตัวอย่างการเรียกใช้ฟังก์ชัน printf ด้วยการส่งค่าพารามิเตอร์แบบ Pass by Values
4. จงยกตัวอย่างการเรียกใช้ฟังก์ชัน scanf ด้วยการส่งค่าพารามิเตอร์แบบ 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) ด้วยคำสั่งภาษาแอสเซมบลี และแสดงผลลัพธ์ ตามตารางในข้อ 6

ภาคผนวก H

การทดลองที่ 8 การพัฒนาโปรแกรมภาษาแอลซีเมบลีขั้นสูง

การพัฒนาโปรแกรมภาษาแอลซีเมบลีขั้นสูง จะเน้นการพัฒนาร่วมกับภาษา C เพื่อเพิ่มศักยภาพของโปรแกรมภาษา C ให้ทำงานได้มีประสิทธิภาพยิ่งขึ้น โดยเฉพาะฟังก์ชันที่สำคัญและต้องเชื่อมต่อกับฮาร์ดแวร์อย่างลึกซึ้ง และถ้ามีประสบการณ์การดีบักโปรแกรมภาษา C จากการทดลองที่ 5 จะยิ่งทำให้ผู้อ่านเข้าใจการทดลองนี้ได้เพิ่มขึ้น ดังนั้น การทดลองมีวัตถุประสงค์เหล่านี้

- เพื่อฝึกการดีบักโปรแกรมภาษาแอลซีเมบลีโดยใช้โปรแกรม GDB แบบคอมมานด์ไลน์ (Command Line)
- เพื่อพัฒนาพัฒนาโปรแกรมแอลซีเมบลีโดยใช้ Stack Pointer (SP) หรือ R13
- เพื่อพัฒนาโปรแกรมภาษาแอลซีเมบลีร่วมกับภาษา C
- เพื่อเสริมความเข้าใจเรื่องเวอร์ชั่นเมโมรีในหัวข้อที่ [5.2](#)

H.1 ดีบักเกอร์ GDB

ดีบักเกอร์เป็นโปรแกรมคอมพิวเตอร์ที่หน้าที่รันโปรแกรมที่กำลังพัฒนา เพื่อให้โปรแกรมเมอร์ตรวจสอบการทำงานได้ลึกซึ้งยิ่งขึ้น ทำให้โปรแกรมเมอร์สามารถเข้าใจการทำงานของโปรแกรมอย่างถ่องแท้ และหากโปรแกรมมีปัญหาหรือ ดีบัก ที่บรรทัดไหน ตำแหน่งใด ดีบักเกอร์เป็นเครื่องมือที่จะช่วยแก้ปัญหานั้นได้ในที่สุด

GDB เป็นดีบักเกอร์มาตรฐานทำงานในระบบปฏิบัติการยูนิกซ์ สามารถช่วยโปรแกรมเมอร์แก้ปัญหาของโปรแกรมที่พัฒนาจากภาษา C/C++ รวมถึงภาษาแอลซีเมบลีของซีพียูนั้น ๆ เช่น แอลซีเมบลีของซีพียู ARM บนบอร์ด Pi นี้

ผู้อ่านสามารถย้อนกลับไปศึกษาการทดลองที่ 5 หัวข้อ E.2 และการทดลองที่ 6 หัวข้อ F.2 อีกรอบเพื่อสังเกตรายละเอียดการสร้างโปรเจกต์ได้ว่า เราได้เลือกใช้ GDB เป็นดีบักเกอร์ ผู้อ่านสามารถเรียนรู้การดีบักโปรแกรมและภาษาแอสเซมบลี พร้อม ๆ กับทำความเข้าใจคำสั่งใน GDB ไปพร้อม ๆ กัน ดังนี้

1. เปิดโปรแกรม Terminal และย้ายไฟล์เดิมที่อยู่ใน /home/pi/asm
2. สร้างไฟล์ใหม่ชื่อ Lab8
3. 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 และกรอกประโยชน์คำสั่งต่อไปนี้

```
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 เพื่อใช้ประกอบการทดลองต่อไป

7. ตั้งค่าเบรกพอยน์เพื่อยุดการรันโปรแกรมชั่วคราว และเปิดโอกาสให้โปรแกรมเมอร์สามารถตรวจสอบค่าของรีจิสเตอร์ต่าง ๆ 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 เป็นหมายเลขบรรทัดที่ตรงกับคำสั่งได

9. โปรดสังเกตว่า (gdb) ปรากฏขึ้นแสดงว่าโปรแกรมหยุดที่เบรกพอยน์แล้ว พิมพ์คำสั่ง (gdb) info r เพื่อแสดงค่าภายในรีจิสเตอร์ต่าง ๆ 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

- อธิบายรายงานบนหน้าจอว่าคอลัมน์แต่ละคอลัมน์มีความหมายอย่างไร และแตกต่างกับหน้าจอของผู้อ่านอย่างไร
- เหตุใดรีจิสเตอร์ **cpsr** มีค่าเป็นเลขฐานสิบในคอลัมน์ขวา สุด มีค่าติดลบ หมายเหตุ ศึกษาเรื่องเลขจำนวนเต็มฐานสองชนิดมีเครื่องหมาย แบบ 2's Complement ในหัวข้อที่ [2.2.2](#)

10. พิมพ์คำสั่ง (**gdb**) **c[ontinue]** เพื่อรันโปรแกรมต่อไปจนกว่าจะวนรอบกลับมาที่เบรกพอยน์ที่ตั้งไว้

11. พิมพ์คำสั่ง (**gdb**) **info r** เพื่อแสดงค่าภายในรีจิสเตอร์ต่าง ๆ ทั้งหมด และบันทึกค่าของรีจิสเตอร์เหล่านี้ **r0, r1, sp, pc, cpsr** เพื่อสังเกตการเปลี่ยนแปลงกับรอบที่แล้ว

12. เริ่มต้นการทดลองโดยพิมพ์คำสั่งต่อไปนี้เพื่อหาว่า เลbel **_loop** ตรงกับหน่วยความจำตำแหน่งใด

```
(gdb) disassemble _loop
```

บันทึกผลที่ได้โดย หมายเลขอ้ายสุด คือ แอดдресในหน่วยความจำ ที่คำสั่งนั้นบรรจุอยู่ หมายเลขตำแหน่งถัดมา คือ จำนวนไบต์นับจากจุดเริ่มต้นของชื่อเลbelนั้น แล้วตรวจสอบว่าเลbel **main** อยู่ห่างจากตำแหน่งเริ่มต้นของโปรแกรมกี่ไบต์

Dump of assembler code for function **_loop**:

```
0x00010414 <+0>: add r0, r0, r1
```

End of assembler dump.

13. พิมพ์คำสั่ง (**gdb**) **c[ontinue]** เพื่อรันโปรแกรมต่อไปจนกว่าจะวนรอบกลับมาที่เบรกพอยน์ที่ตั้งไว้อีกรอบ
14. คำสั่ง **x/ [count] [format] [address]** แสดงค่าในหน่วยความจำ ณ ตำแหน่ง address เป็นต้นไป เป็นจำนวน /count ตาม format ที่ต้องการ ยกตัวอย่าง เช่น **x/10i main** คือ แสดงค่าในหน่วยความจำ ณ ตำแหน่งเลขบล 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 **<_>** สອงตำแหน่ง
 - อธิบายว่า หมายเลขที่มาแทนที่ **<_>** ได้อย่างไร
 - โปรดสังเกตและอธิบายว่าเครื่องหมายลูกศร => 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] b[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 _ ทั้งสองช่อง

18. คำสั่ง **d[elete] b[reakpoints] number** ลบการตั้งเบรกพอยน์ที่บรรทัด **number** ที่ตั้งไว้ก่อนหน้า ผู้อ่านสามารถลบเบรกพอยน์ทั้งหมดพร้อมกันโดยพิมพ์

```
(gdb) d
Delete all breakpoints? (y or n)
```

แล้วตอบ y เพื่อยืนยัน

19. พิมพ์คำสั่ง **(gdb) c** เพื่อรันโปรแกรมต่อไปจนเสร็จสิ้น จะได้ผลลัพธ์ต่อไปนี้

```
(gdb) c
Continuing.
[Iinferior 1 (process 1688) exited with code 012]
```

20. พิมพ์คำสั่งต่อไปนี้เพื่ออกจากโปรแกรม GDB

```
(gdb) q
```

H.2 การใช้งานสเต็กพอยน์เตอร์ (Stack Pointer)

ตำแหน่งของหน่วยความจำบริเวณที่เรียกว่า **สเต็ก เซกเมนต์** (Stack Segment) จากรูปที่ 3.16 สเต็กเซกเมนต์ตั้งในบริเวณแอ็ดเดรสสูง (High Address) หน้าที่เก็บค่าข้อมูลของตัวแปรชนิดโลคอล (Local Variable) รับค่าพารามิเตอร์ระหว่างฟังก์ชัน กรณีที่มีจำนวนเกิน 4 ตัว พักเก็บค่าของรีจิสเตอร์ที่สำคัญ ๆ เช่น LR เป็นต้น

สเต็ก พอยน์เตอร์ คือ รีจิสเตอร์ R13 มีหน้าที่เก็บแอ็ดเดรสตำแหน่งบนสุดของสเต็ก (Top of Stack: TOS) ซึ่งจะเป็นตำแหน่งที่เกิดการ PUSH และ POP ข้อมูลเข้าและออกจากสเต็กตามลำดับ โปรแกรมเมอร์สามารถจินตนาการได้ว่า **สเต็ก** คือ กองสิ่งของที่วางซ้อนกันโดยโปรแกรมเมอร์ และสามารถหยิบสิ่งของออก (POP) หรือวาง (PUSH) ของที่ซั่นบนสุดเท่านั้น โดยเราเรียกกองสิ่งของ (ตัวแปรโลคอลและอื่นๆ) นี้ว่า **สเต็กเฟรม** ซึ่งได้อธิบายในหัวข้อที่ 3.3.3 เราสามารถทำความเข้าใจการทำงานของสเต็กแบบง่าย ๆ ได้ดังนี้

สแตก พอยน์เตอร์ คือ หมายเลข ชั้น สิ่งของ ซึ่ง ตำแหน่ง จะ ลด ลง/เพิ่ม ขึ้น เมื่อ โปรแกรมเมอร์ ใช้ คำ สั่ง PUSH/POP ตาม ลำดับ ซึ่ง มี รายละเอียด เพิ่ม เติม ใน หัวข้อ ที่ 4.5 ทั้งนี้ เรา สามารถ อ้าง อิง จาก เวอร์ชัล เมโมรี ของ ระบบ ลินุกซ์ ใน รูป ที่ 3.16 และ รูป ที่ 5.2 ประกอบ

คำ สั่ง STM (Store Multiple) ทำ หน้าที่ PUSH ข้อมูล หรือ ค่า ของ รีจิสเตอร์ จำนวน หนึ่ง ลง บน สแตก ณ ตำแหน่ง TOS คำ สั่ง LDM (Load Multiple) ทำ หน้าที่ POP ข้อมูล ออก จาก สแตก ณ ตำแหน่ง TOS มา กีบ ใน รีจิสเตอร์ จำนวน หนึ่ง การเปลี่ยน แปลง ตำแหน่ง ของ TOS เป็น ไปได้ สอง ทิศทาง คือ เพิ่ม ขึ้น (Ascending)/ลด ลง (Descending). ดังนั้น คำ สั่ง STM/LDM สามารถ สม กับ ทิศทาง และ ลำดับ การ กระทำ คือ ก่อน (Before) /หลัง (After) รวม เป็น 8 แบบ ดังนี้

- LDMIA/STMIA : IA ย่อ จำก Increment After
- LDMIB/STMIB : IB ย่อ จำก Increment Before
- LDMDA/STMDA : DA ย่อ จำก Decrement After
- LDMDB/STMDB : DB ย่อ จำก Decrement Before

คำ Increment/Decrement หมายถึง การ เพิ่ม/ลด ค่า ของ รีจิสเตอร์ ที่ เกี่ยว ข้อง โดย มาก ใช้งาน ร่วม กับ รีจิสเตอร์ SP คำ after/before หมายถึง ก่อน/หลัง การ ปฏิบัติ (Execute) ตาม คำ สั่ง นั้น ยก ตัวอย่าง การ ใช้งาน คำ สั่ง เพื่อ PUSH รีจิสเตอร์ ลง ใน สแตก โดย ใช้ STMDB และ POP ค่า จาก สแตก จะ คู่ กับ คำ สั่ง LDMIA ความ หมาย คือ สแตก จะ เติบ โต ใน ทิศทาง ที่ แอด เดอะ ลด ลง (Decrement Before) ซึ่ง เป็น ที่ นิยม และ ตรง กับ รูป การ จัด วาง เวอร์ชัล เม莫รี หรือ หน่วย ความ จำ เสมือน ใน รูป ที่ 3.16 ผู้ อ่าน สามารถ บท ทวน เรื่องนี้ ใน หัวข้อ ที่ 5.2

1. สร้าง ไฟล์ Lab8_2.s ตาม โค้ด ต่อไปนี้ ผู้ อ่าน สามารถ ข้าม ประ โยค คอม เมนต์ ได้ เมื่อ ทำการ เข้า ใจ แต่ ละ คำ สั่ง แล้ว

```
.global main
main:
    MOV R1, #1
    MOV R2, #2

    @ Push (store) R1 onto stack at SP-4, then SP = SP-4 bytes
    @ The ! (Write-Back symbol) updates the register SP
    STR R1, [sp, #-4]!
    STR R2, [sp, #-4]!

    @ Pop (load) the value at SP 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 decremented by 8 bytes before pushing R4 and R5
    @ The ! (Write-Back symbol) updates SP accordingly.
    STMDB SP!, {R4, R5}

    @ R2 and R1 are popped, respectively.
    @ SP is incremented after (IA) 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 สามารถเชื่อมต่อ กับ ฮาร์ดแวร์ และ ทำงาน ได้ รวดเร็ว ใกล้ เคียง กับ ภาษา แอสเซมบลี แต่ การเสริม การทำงาน ของ โปรแกรมภาษา C ด้วย ภาษา แอสเซมบลี ยัง มี ความ จำเป็น โดยเฉพาะ โปรแกรม ที่ เรียกว่า ดีไวซ์ ไดรเวอร์ (Device Driver) ซึ่ง เป็น โปรแกรม ขนาดเล็ก ที่ เชื่อมต่อ กับ ฮาร์ดแวร์ ที่ ต้องการ ความ รวดเร็ว และ ประสิทธิภาพ สูง การ ทดลอง นี้ จะ แสดง ให้ ผู้อ่าน เห็น การ เชื่อมต่อ พัฟ์ชัน ภาษา แอสเซมบลี กับ ภาษา C อย่าง ง่าย

1. เปิด โปรแกรม CodeBlocks
2. สร้าง โปรเจกต์ Lab8_4 ภายใต้ ไดเรกทอรี /home/pi/asm/Lab8
3. สร้าง ไฟล์ ชื่อ add_s.s และ ป้อน คำสั่ง ต่อไปนี้

```
.global add_s
add_s:
ADD R0, R0, R1
BX LR
```

4. เพิ่ม ไฟล์ add_s.s ใน โปรเจกต์ Lab8_4 ที่ สร้าง ไว้ ก่อนหน้า
5. สร้าง ไฟล์ ชื่อ main.c และ ป้อน คำสั่ง ต่อไปนี้

```
#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 และ แก้ไข หาก มี ข้อผิดพลาด จน สำเร็จ
7. Run และ สังเกต การเปลี่ยนแปลง
8. อธิบาย ว่า เหตุ ใด การทำงาน จึง ถูก ต้อง พัฟ์ชัน add_s รับ ข้อมูล ทาง รีจิส เตอร์ ตัว ไหน บ้าง และ รีเทิร์น ค่า ที่ คำนวณ เสร็จ แล้ว ทาง รีจิส เตอร์ อะไร

```
Lab8_4: main.c add_s.s
gcc -o Lab8_4 main.c add_s.s
```

9. อธิบายว่าเหตุใดฟังก์ชัน add_s จึงไม่ต้องแบ่งกับค่าของรีจิสเตอร์ LR

ในทางปฏิบัติ การบวกเลขในภาษา C สามารถทำได้โดยใช้เครื่องหมาย + โดยตรง และทำงานได้รวดเร็ว กว่า การทดลอง ตัวอย่างนี้ เป็นการนำเสนอว่า ผู้อ่านสามารถเขียนโปรแกรมอย่างไรที่จะบรรลุวัตถุประสงค์เท่านั้น ฟังก์ชันภาษาแอสเซมบลีที่จะลิงก์เข้ากับโปรแกรมหลักที่เป็นภาษา C ควรจะมีอรรถประโยชน์มากกว่านี้ และเชื่อมโยงกับฮาร์ดแวร์โดยตรงได้ดีกว่าคำสั่งในภาษา C เช่น ตัวชี้ไดรเวอร์

H.4 กิจกรรมท้ายการทดลอง

1. จงตีบักโปรแกรม Lab8_1 ด้วย GDB พร้อมกันจำนวน 2 Terminal เพื่อแสดงค่าของรีจิสเตอร์ PC ที่รันคำสั่งแรกของโปรแกรม Lab8_1 ในทั้งสองหน้าต่าง และเปรียบเทียบค่า PC ว่าเท่ากันหรือแตกต่างกันหรือไม่ เพราะเหตุใด
2. หากค่าของรีจิสเตอร์ PC ทั้งสองค่าในข้อ 1 ตรงกัน จะใช้ความรู้เรื่องเวอร์ชวลเม莫รีหรือหน่วยความจำเสมือนในหัวข้อ 5.2 เพื่อตอบคำถาม
3. จงใช้โปรแกรม GDB เพื่อแสดงรายละเอียดของสแต็กกระหว่างที่รันโปรแกรม Lab8_2 และบอกลำดับการ PUSH และการ POP ที่เกิดขึ้นภายในโปรแกรมจากแต่ละคำสั่ง
4. จงใช้โปรแกรม GDB เพื่อแสดงรายละเอียดของสแต็กกระหว่างที่รันโปรแกรม Lab8_3 และบอกลำดับการ PUSH และการ POP ที่เกิดขึ้นภายในโปรแกรมจากแต่ละคำสั่ง
5. จงนำโปรแกรมภาษาแอสเซมบลีสำหรับคำ mod ใน การทดลองที่ 7 มาเรียกใช้ผ่านโปรแกรมภาษา C
6. จงนำโปรแกรมภาษาแอสเซมบลีสำหรับคำ GCD ใน การทดลองที่ 7 มาเรียกใช้ผ่านโปรแกรมภาษา C
7. จงตีบักโปรแกรมภาษา C บนโปรแกรม Codeblocks ที่พัฒนาในข้อ 2 และ 3 เพื่อบันทึกการเปลี่ยนแปลงของ PC ก่อนระหว่าง และหลังเรียกใช้ฟังก์ชันภาษา Assembly ว่าเปลี่ยนแปลงอย่างไร และตรงกับทฤษฎีที่เรียนหรือไม่ อย่างไร
8. เครื่องหมาย -d ใน makefile ต่อไปนี้

```
debug: Lab8_1
      as -g -o Lab8_1.o Lab8_1.s
```

มีความหมายอย่างไร

ภาคผนวก I

การทดลองที่ 9 การศึกษาและปรับแก้อินพุตและ เอาต์พุตต่างๆ

การทดลองในภาคผนวกนี้ จะช่วยอธิบายเนื้อหาในบทที่ 6 ซึ่งเกี่ยวข้องกับ อุปกรณ์อินพุต/เอาต์พุตที่ หลากหลายบนเครื่องคอมพิวเตอร์ตั้งโต๊ะ โดยมีวัตถุประสงค์เหล่านี้

- เพื่อให้เข้าใจการปรับแก้อุปกรณ์อินพุตและเอาต์พุตชนิดต่าง ๆ บนระบบปฏิบัติการ Raspberry Pi OS
- เพื่อให้เข้าใจความแตกต่างระหว่างอุปกรณ์อินพุตและเอาต์พุตชนิดต่าง ๆ บนบอร์ด Pi
- เพื่อให้สามารถอ่านข้อมูลความแสดงรายการละเอียดของอุปกรณ์อินพุตและเอาต์พุตชนิดต่างๆ

หลักการและพื้นฐานความเข้าใจจะช่วยแนะนำทางให้ผู้อ่านสามารถศึกษาค้นคว้า อินพุต/เอาต์พุต อื่น ๆ ในชิปและบันบอร์ดได้เพิ่มเติม รวมไปถึงบันโทรศัพท์เคลื่อนที่ แท็บเล็ตคอมพิวเตอร์ และ อุปกรณ์อินเทอร์เน็ตสตรอร์สิ่ง (Internet of Things)

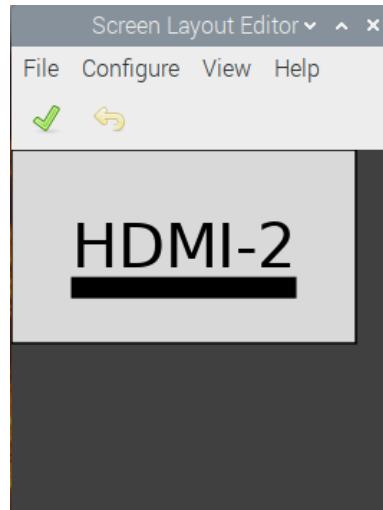
I.1 จอแสดงผลผ่านพอร์ต HDMI

I.1.1 การปรับแก้ความละเอียดของจอแสดงผล

เมื่อขนาดหน่วยความจำสำหรับการใช้งานและแสดงผลของจีพียูมีปริมาณเพียงพอ ซึ่งตั้งอยู่บริเวณชื่อว่า VC SDRAM ในพื้นที่หน่วยความจำกายภาพ (ARM Physical Memory) ของรูปที่ 6.16 ผู้ใช้สามารถปรับเพิ่มหรือลดความละเอียดของจอแสดงผลได้โดยกดปุ่มบนเมนูดังนี้

menu->Preferences->Screen Configuration

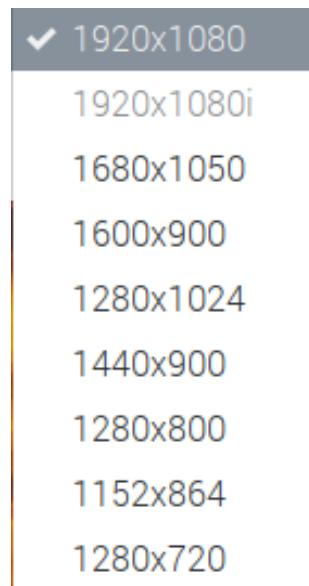
นี่เป็นการเรียกใช้โปรแกรม Screen Layout Editor



รูปที่ I.1: หน้าต่าง Screen Layout Editor สำหรับกำหนดค่าต่าง ๆ กับพอร์ตแสดงผล HDMI

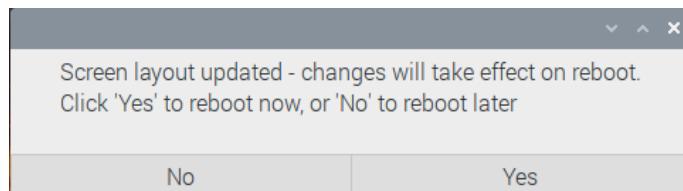
กดปุ่มบนเมนูบนโปรแกรมเพื่อปรับความละเอียดของพอร์ตที่จะแสดงผล เชื่อมต่ออยู่ และรองรับ

Configure->Screens->HDMI-1 (HDMI-2)->Resolution



รูปที่ I.2: หน้าต่าง Set Resolution สำหรับกำหนดความละเอียดหน้าจอแสดงผลที่ต้องการ

กดปุ่ม เลือก ความ ละเอียด หน้า จอ ที่ เหมาะสม สม กับ จอ ที่ เชื่อม ต่อ อยู่ คือ ไม่เกิน ความ ละเอียด 1920x1080 หลังจากนั้นกดปุ่ม เครื่องหมายถูก ในหน้าต่างหลักของ Scrren Layout Editor เพื่อยืนยัน หน้าต่างต่อไปนี้จะปรากฏขึ้น



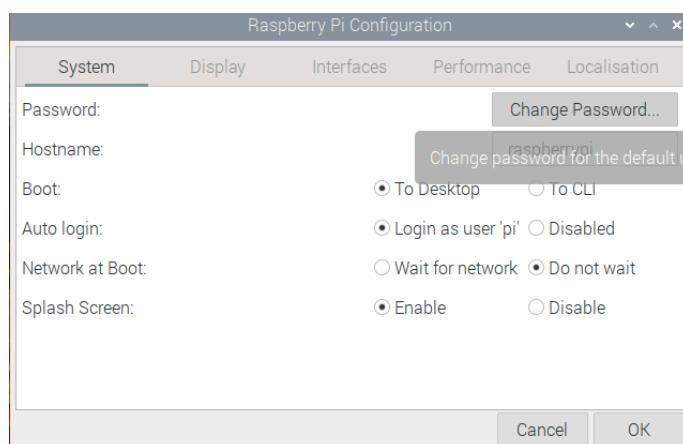
รูปที่ I.3: หน้าต่าง Reboot needed กดปุ่ม Yes เมื่อต้องการรีบูต ณ เวลานั้น

กด Yes เพื่อรีบูตระบบใหม่ ผู้อ่านสามารถค้นคว้าเรื่องสายมาตรฐาน HDMI ในหัวข้อที่ 6.1 เพิ่มเติม ก่อนค้นคว้าเพิ่มเติมในอินเทอร์เน็ต

I.1.2 การปรับแก้ขนาดหน่วยความจำของ GPU

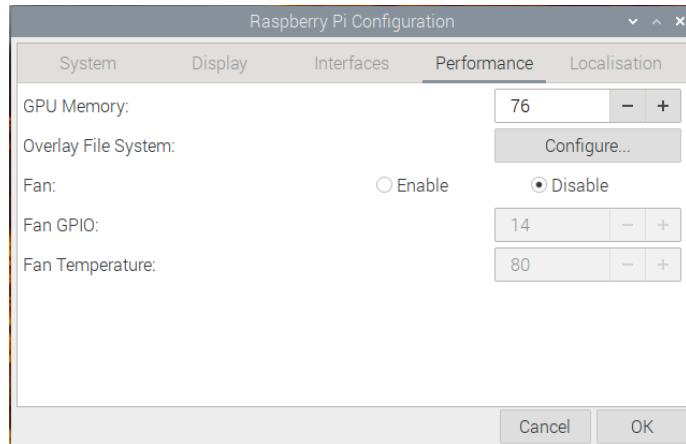
หน่วยความจำสำหรับจ่อแสดงผลหรือจีพียู (Graphic Processing Unit) ถูกแบ่งเป็นที่ออกจากหน่วยความจำ SDRAM บนบอร์ด เพื่อใช้งานร่วมกันทำให้ประหยัดต้นทุน แต่มีข้อเสียในด้านประสิทธิภาพจะลดลง เมื่อผู้ใช้งานต้องการภาพที่มีอัตราเฟรมเรท (Frame Rate) สูง เช่น ภาพวิดีโอเคลื่อนไหว เกม 3 มิติ ความละเอียดของจ่อแสดงผลขึ้นตรงกับขนาดของหน่วยความจำของจีพียู ผู้อ่านสามารถปรับแก้ขนาดหน่วยความจำของจีพียูได้ดังนี้

menu->Preferences->Raspberry Pi Configuration



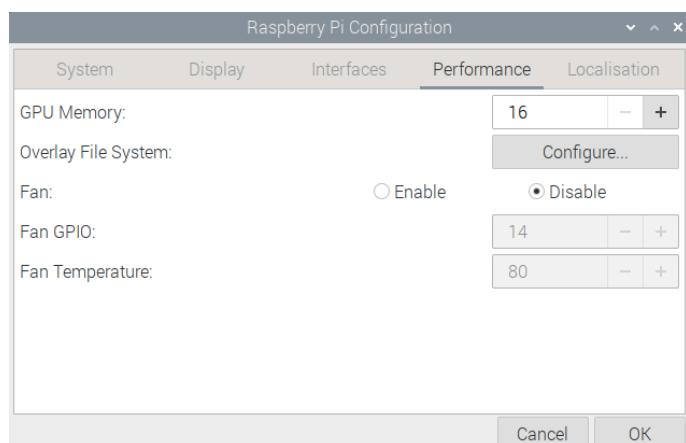
รูปที่ I.4: หน้าต่าง Raspberry Pi Configuration

กดแท็บ Performance เพื่อปรับค่าต่าง ๆ 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 และปรับแต่งระดับเสียงได้ เช่น กัน

I.2.1 การเลือกช่องสัญญาณเสียงเขื่อมต่อกับลำโพง

ผู้อ่านสามารถเชื่อมต่อสัญญาณเสียงกับลำโพงภายนอกผ่านช่องแจ็ค 3.5 มม. หรือ ลำโพงของจอทีวี LCD ผ่านช่องสัญญาณ HDMI จากการทดลองต่อไปนี้

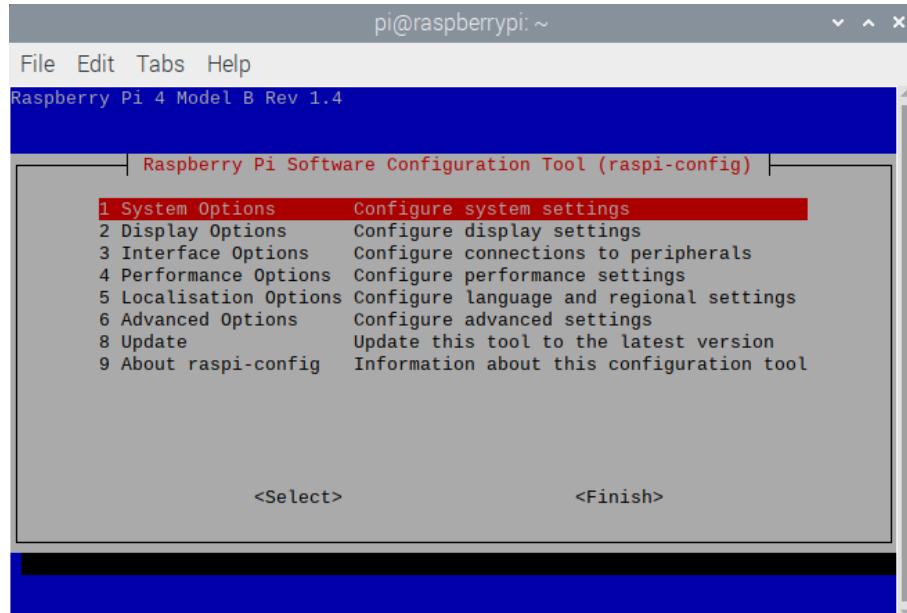
1. ใช้คำสั่งต่อไปนี้ในโปรแกรม Terminal

```
$ sudo raspi-config
```

จะบอกเหตุผลว่าคำสั่ง sudo ที่นำหน้ามีความสำคัญอย่างไร

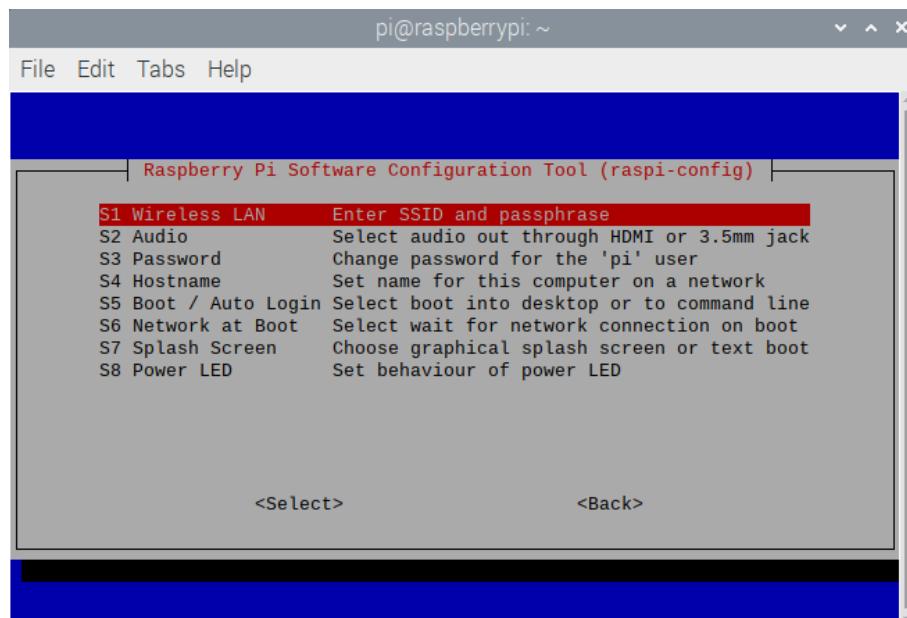
- กดปุ่มลูกศรขึ้นลงเพื่อเลือกเมนู System Options ในรูป

```
$ sudo raspi-config
```



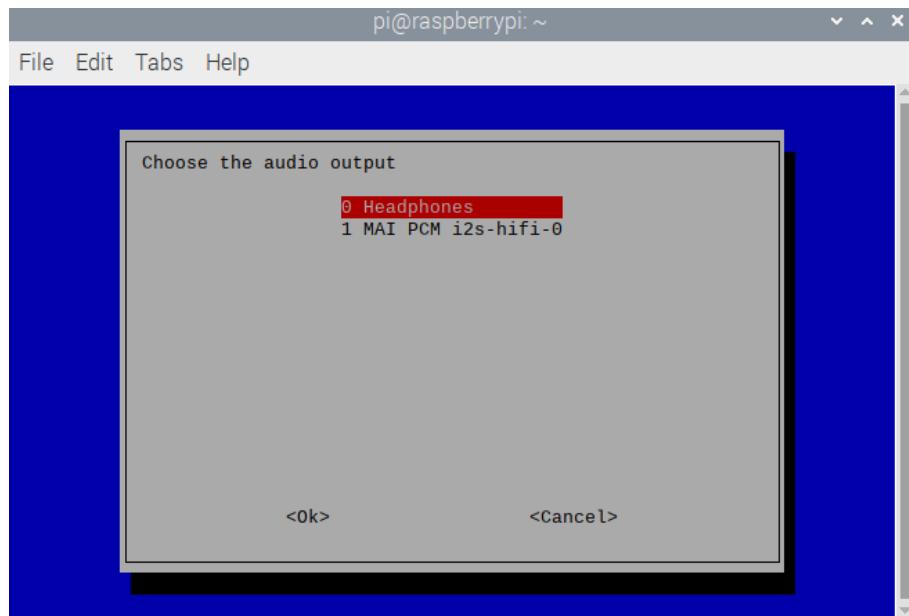
รูปที่ 1.7: หน้าต่างโปรแกรม raspi-config สำหรับบอร์ด Pi

- กดปุ่มลูกศรขึ้นลงเพื่อเลือกเมนู S2 Audio ในรูป



รูปที่ 1.8: เมนู System Options ในหน้าต่างโปรแกรม raspi-config สำหรับบอร์ด Pi

- กดปุ่มลูกศรขึ้นลงในรูปเพื่อเลือกเมนู 0 Headphones สำหรับแจ็ค 3.5 มม. หรือ 1 MAI PCM i2s-hifi-0 สำหรับพอร์ต HDMI

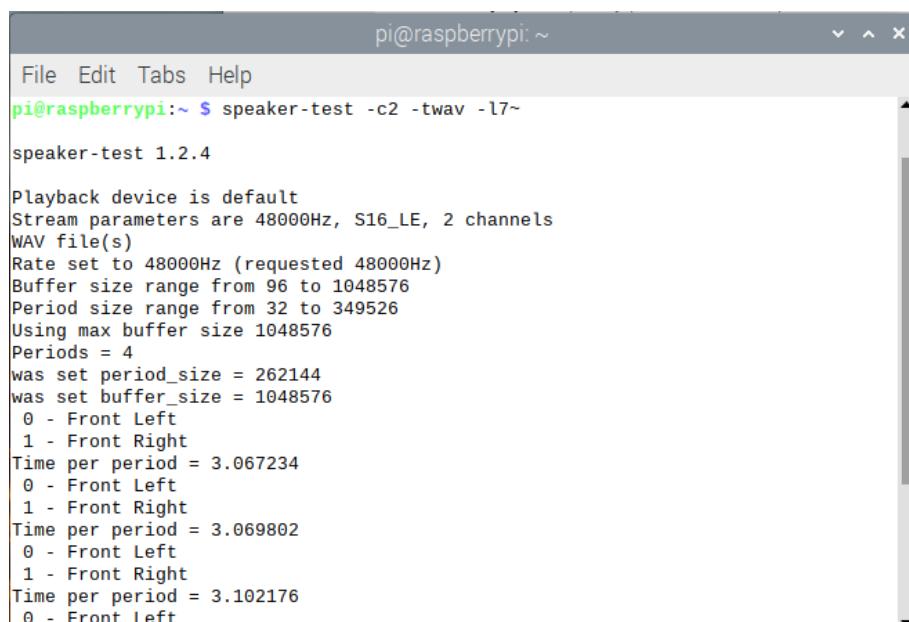


รูปที่ I.9: เมนู Audio ภายในเมนู System Options ในหน้าต่างโปรแกรม raspi-config สำหรับบอร์ด Pi

5. เมื่อเลือกเมนูที่ต้องการแล้ว กดปุ่ม Esc(ape) เพื่อถอยกลับออกมาจากเมนู และกดจนออกจากโปรแกรม
6. ใช้คำสั่งต่อไปนี้ในโปรแกรม Terminal เพื่อทดสอบสัญญาณเสียงกับลำโพงที่เลือกต่อ

```
$ speaker-test -c2 -twav -l7
```

หากสำเร็จ ผู้อ่านจะได้ยินเสียงและผลลัพธ์คล้ายรูปต่อไปนี้



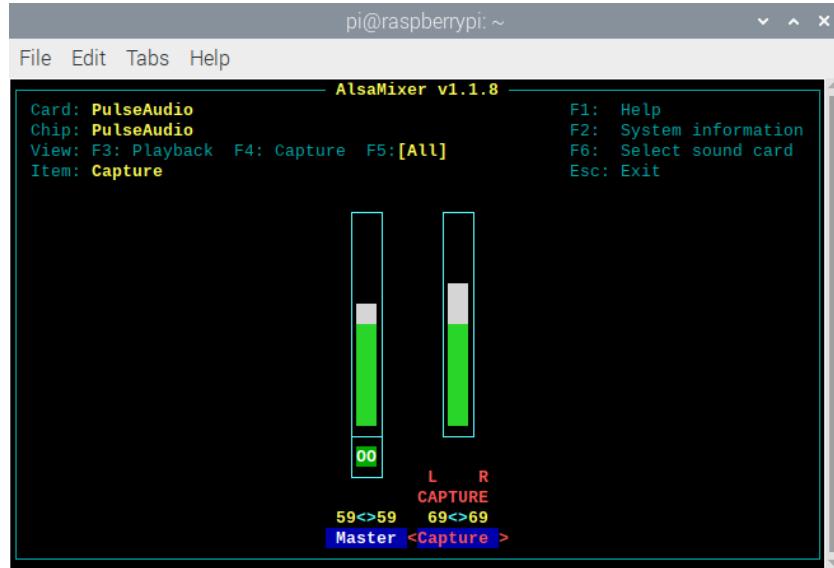
รูปที่ I.10: ผลลัพธ์ในหน้าต่างโปรแกรม speaker-test สำหรับบอร์ด Pi

I.2.2 การควบคุมระดับเสียง

นอกเหนือจากระดับเสียงที่ไอคอนรูปลำโพงด้านขวาบนของจอ 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 ผู้อ่านจะสังเกตได้ว่า ไดเรกทอรี /proc/asound/pcm จะเชื่อมโยงกับเนื้อหาในหัวข้อที่ 6.4 และเห็นว่ามีไดเรกทอรีชื่อ card0 อยู่สองตำแหน่งคือ ในแควร์ก และแควร์ที่มีชื่อบ1 -> card0 สัญลักษณ์ -> เรียกว่า **ซิมบอลิกลิงก์** (Symbolic Link) หมายความว่า ไดเรกทอรีชื่อบ1 คือไดเรกทอรี card0 ส่วนแควร์ที่มีชื่อ Headphones -> card1 สัญลักษณ์ -> เรียกว่า **ซิมบอลิกลิงก์** (Symbolic Link) หมายความว่า ไดเรกทอรีชื่อ Headphones คือไดเรกทอรี card1

- ผู้อ่านสามารถค้นเพิ่มเติมโดยพิมพ์คำสั่งต่อไปนี้

```
$ cat /proc/asound/cards
```

บันทึกผลลัพธ์ในพื้นที่ว่างต่อไปนี้

- ค้นคว้าว่า b1 และ Headphones คือ อุปกรณ์ใด ทั้งสองอุปกรณ์นี้แตกต่างกันหรือไม่

- 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_____ แต่ยังใช้ไดรเวอร์เสียงเดียวกันกับ BCM283_____ โดย หมายเลข 0 คือ หมายเลขของระบบเสียงที่ติดตั้งใช้งานเพียงระบบเดียว และตรง กับอุปกรณ์ชื่อ _____0

I.3 พور์ตเชื่อมต่ออุปกรณ์ USB

I.3.1 รายชื่ออุปกรณ์กับพอร์ต USB

1. ในการทดลองนี้ ขอผู้อ่านให้ดึงหัว เชื่อมต่อ USB ของมาส์ที่ใช้อู่ออก 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 ตัวรับส่งสัญญาณคีย์บอร์ดและเมาส์ไร้สายของผู้ใช้ หมายเลข 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. ผู้อ่านย้ายคีย์บอร์ดหรือเมาส์จากพอร์ต USB 2.0 ไปยัง USB 3.0 แล้วแสดงรายชื่ออุปกรณ์ USB ด้วยคำสั่ง

\$ lsusb

เข่นเดิม 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

- รันคำสั่งนี้ แล้วเลื่อนหน้าต่างขึ้นไปที่ตำแหน่งวินาทีที่ 0.000000

```
$ dmesg
```

- จะเปรียบเทียบข้อความที่ผู้อ่านได้กับข้อความต่อไปนี้ ระบุตำแหน่งที่แตกต่างและเขียนข้อความนั้นลงในผลการทดลอง

```
[    0.000000] Booting Linux on physical CPU 0x000000000000 [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 ข้อความที่แสดงเป็น 0.000000 เนื่องจากเครื่องเนลอยู่ระหว่างการเริ่มต้น

- เริ่มต้นการบูตระบบปฏิบัติการด้วยแกนประมวลผลหมายเลข _____
- จดบันทึกหมายเลขเวอร์ชันของลินุกซ์ของผู้อ่านโดยละเอียด 5._____._____._____
- จดบันทึกคำสั่งภาษาแอสเซมบลีเวอร์ชัน _____ บิต
- แสดงผลการตรวจว่าเป็นบอร์ด Raspberry Pi _____ Model _____ Rev _____
- cma ย่อ มา จาก _____ สำหรับ ขบวนการ DMA เริ่ม ต้น ที่ แอดเดรส 0x_____ ขนาด _____ เมบิไบต์
- ...

ในการทดลองนี้ ระบบสามารถตรวจจับอุปกรณ์ USB และติดตั้งไดรเวอร์ได้อย่างถูกต้องปราศจากข้อผิดพลาด

- ผู้อ่านสามารถล้างบัฟเฟอร์โดยใช้คำสั่ง ต่อไปนี้

```
$ sudo dmesg -C
```

โดย -C คือ Clear เป็นคำสั่งเพิ่มเติมให้ dmesg ล้างข้อความในบัฟเฟอร์ออก โปรดสังเกต ตัว C พิมพ์ใหญ่ หลังจากนั้น ผู้อ่านทดสอบโดยการถอดเม้าส์ออก แล้วเสียบกลับเข้าไปใหม่

- ผู้อ่านจะต้องถอดและเสียบเม้าส์กลับเข้าไปใหม่อีกรอบ
- ผู้อ่านสามารถแสดงข้อความที่เพิ่มเข้ามาในบัฟเฟอร์ได้อีก โดยเรียกคำสั่ง

```
$ dmesg
```

- จดบันทึกเฉพาะ 4 บรรทัดแรก

- อภิปรายผลลัพธ์ที่บันทึกได้ในพื้นที่ว่างต่อไปนี้

ในการเชื่อมต่อพอร์ต USB หากระบบแจ้งชื่ออุปกรณ์โดยไม่มีข้อมูลพิเศษ แต่อุปกรณ์นั้นยังไม่สามารถทำงานได้ แสดงว่า อุปกรณ์ขาดซอฟต์แวร์ซึ่งทำหน้าที่เป็นดีไวซ์ไดรเวอร์ ขอให้ผู้อ่านค้นหาจากหมายเลขประจำตัวของผู้ผลิต (idVendor) หากผู้ผลิตมิได้เปิดเผยซอฟต์แวร์ ผู้อ่านจำเป็นต้องดาวน์โหลดหรือคุณไฟล์เองจากนักพัฒนารายอื่นแทน

I.4 พอร์ตเชื่อมต่ออินเทอร์เน็ตผ่านสัญญาณ WiFi และ Ethernet

I.4.1 รายชื่ออุปกรณ์เครือข่าย

- ผู้อ่านสามารถตรวจสอบรายชื่ออุปกรณ์สำหรับเชื่อมต่อเครือข่ายได้จากคำสั่ง `ifconfig` ทางโปรแกรม Terminal ตัวอย่างผลลัพธ์เป็นดังนี้

```
$ ifconfig
```

- เติมข้อมูลหรือตัวเลขในช่องว่าง _____ ที่เตรียมไว้ให้จากผลลัพธ์ที่ได้ต่อไปนี้ ซึ่งลำดับรายการอาจแตกต่างกัน

```
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 _____
```

- โปรดสังเกตคำเริ่มต้นในแต่ละรายการ ค้นคว้า และกรอกรายละเอียดเพิ่มเติม ดังนี้

- eth0 หมายถึง
- lo หมายถึง

- wlan0 หมายถึง

ผู้อ่านสามารถค้นคว้าเพิ่มเติมได้ที่หน้าเว็บต่อไปนี้

<https://www.tecmint.com/ifconfig-command-examples/>

I.4.2 การเปิด/ปิดอุปกรณ์เครือข่าย

- ผู้อ่านสามารถเปิดอุปกรณ์ eth0 ได้ตามต้องการแล้วทำการตรวจสอบ ดังนี้

```
$ sudo ifconfig eth0 down
$ ifconfig
```

จดว่าข้อความใดที่บ่งบอกว่า eth0 ไม่ทำงานแล้ว

- ผู้อ่านสามารถเปิดอุปกรณ์ eth0 ได้ตามต้องการแล้วทำการตรวจสอบ ดังนี้

```
$ sudo ifconfig eth0 up
$ ifconfig
```

จดว่าข้อความใดที่บ่งบอกว่า eth0 ทำงานแล้ว

- ผู้อ่านสามารถใช้คำสั่ง ifconfig สำหรับปิด อุปกรณ์ wlan0 ดังนี้

```
$ sudo ifconfig wlan0 down
$ ifconfig
```

- ผู้อ่านสามารถใช้คำสั่ง ifconfig สำหรับเปิด อุปกรณ์ wlan0 ดังนี้

```
$ sudo ifconfig wlan0 up
$ ifconfig
```

จดว่าข้อความใดที่บ่งบอกว่า wlan0 ทำงานแล้ว

- นอกเหนือจากการเปิดปิดอุปกรณ์เครือข่าย ผู้อ่านสามารถตรวจสอบรายชื่อเครือข่าย WiFi ที่บอร์ด เคยเชื่อมต่อสำเร็จได้จากไฟล์ wpa_supplicant.conf ซึ่งจะบันทึกรายละเอียดต่าง ๆ ของการเชื่อมต่อนั้น ๆ รวมถึงรหัสเซอร์ฟเวอร์ด (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

I.4.3 การตรวจสอบการเชื่อมต่อ กับ อินเทอร์เน็ต เป็นต้น

เมื่อผู้อ่านเปิดและทำการเชื่อมต่อสำเร็จแล้ว จึงสามารถตรวจสอบการเชื่อมต่อในระดับขั้นเครือข่ายโดยใช้คำสั่ง ping ใน Terminal ดังนี้

```
$ ping <ip address or host name>
```

การตรวจสอบการเชื่อมต่อเป็นต้น 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 ที่ผู้อ่านจะต้องเติมหมายเลขลงใน _____ 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 แล้วรออุปกรณ์หมายเลขนี้ตอบกลับมา ของแพ็กเก็ตลำดับที่ _____ (icmp_seq=_____) เป็นระยะเวลา ____ มิลลิวินาที ส่วน ttl=____ ย่อมาจากคำว่า time to live หมายถึง เลขจำนวนเต็มที่ผู้ส่งกำหนดค่าอายุของ

แพ็คเก็ตที่สามารถเดินทางผ่านเครือข่าย หากตั้งไว้น้อยจะทำให้แพ็คเก็ตข้อมูลนี้อายุสั้นและอาจเดินทางไปไม่ถึงปลายทางเนื่องจากหมดอายุก่อน โดย ttl=64 เป็นค่าปกติ

ผู้อ่าน จะสังเกตเห็นว่า ระยะเวลา มีค่าตั้งแต่ _. ___ - ___. มิลลิวินาที ขึ้นอยู่กับคุณภาพของสาย Ethernet หรือความแรงของสัญญาณ WiFi คุณภาพดีจะทำให้ระยะเวลาสั้นกว่า หลังจากตรวจสอบว่าบอร์ดสามารถเชื่อมต่อ กับเราเตอร์ต้นทางได้ตามตัวอย่างก่อนหน้า ผู้อ่านสามารถใช้ตรวจสอบการเชื่อมต่อได้ว่า เราเตอร์ต้นทางสามารถเชื่อมต่อ กับเครือข่ายอินเทอร์เน็ตได้สำเร็จหรือไม่ โดย Host name คือ ชื่อเซิร์ฟเวอร์ปลายทางที่จะทะเบียนโดเมนเนม (Domain Name) เรียบร้อยแล้ว เช่น ping www.google.com

I.5 กิจกรรมท้ายการทดลอง

1. จงค้นหาว่าความละเอียดของการแสดงผลผ่านพอร์ต HDMI ในหัวข้อที่ [I.1.1](#) เก็บบันทึกลงในไฟล์ชื่ออะไร
2. ใช้คำสั่ง ifconfig ปิดอุปกรณ์ 10 แล้วใช้คำสั่ง ping 127.0.0.1 ว่ามีการตอบสนองกลับมาหรือไม่ เปิดอุปกรณ์ 10 แล้ว ping อีกรอบ จะอธิบายว่า 127.0.0.1 คืออะไร
3. ใช้คำสั่ง ping เพื่อทดสอบ เราเตอร์ที่อยู่ต้นทางของผู้อ่าน 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

ภาคผนวก J

การทดลองที่ 10 การเชื่อมต่อกับขา GPIO

การทดลองนี้คาดว่าผู้อ่านเคยศึกษาและทดลองเขียนหรือพัฒนาโปรแกรมด้วยภาษา C มาบ้างแล้ว และมีความคุ้นเคยกับ IDE (Integrated Development Environment) จากพัฒนาโปรแกรมและการติดต่อกับโปรแกรมด้วยภาษา C/C++ และแօสเซมบลี ดังนั้น การทดลองมีวัตถุประสงค์เหล่านี้

- เพื่อปฏิบัติการเชื่อมต่อวงจรกับขา GPIO บนบอร์ด Pi ตามเนื้อหาในบทที่ 6 หัวข้อที่ 6.11
 - เพื่อพัฒนาโปรแกรมภาษา C ควบคุมการทำงานของขา GPIO ด้วยไลบรารี wiringPi
 - เพื่อพัฒนาโปรแกรมภาษาแօสเซมบลีควบคุมการทำงานของขา GPIO ด้วยไลบรารี wiringPi
- โปรดสังเกตตัวอักษร W ที่คำว่า wiringPi ต้องเป็นตัวอักษรพิมพ์เล็ก

J.1 ไลบรารี wiringPi

ไลบรารี wiringPi รวบรวมฟังก์ชันที่พัฒนาด้วยภาษา C สำหรับบอร์ด Pi เป็น OpenSource ภายใต้ GNU LGPLv3 license สามารถเรียกใช้งานผ่านภาษา C and C++ รวมถึงภาษาแօสเซมบลี

เนื่องจาก ไลบรารี wiringPi เป็นซอฟต์แวร์แบบโอเพนซอร์ส แจกให้แก่นักพัฒนาทั่วโลก ผ่านทาง <https://github.com/WiringPi> และมีการปรับปรุงแก้ไขตลอดเวลาโดยทีมนักพัฒนา ดังนั้น ผู้อ่านควรต้องติดตั้งและปรับปรุงระบบปฏิบัติการให้ทันสมัยและติดตั้งตามขั้นตอนต่อไปนี้

- ผู้อ่านควรตรวจสอบว่าบอร์ดที่มีติดตั้งไลบรารี WiringPi แล้วหรือยัง โดยใช้คำสั่ง

```
$ gpio -v
```

- หากบอร์ดยังไม่ได้ติดตั้งผู้อ่านควรปรับปรุงระบบปฏิบัติการให้เป็นปัจจุบันก่อน โดยพิมพ์คำสั่งนี้บนโปรแกรม 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 `gpio -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. จงเติมหมายเลขอื่นคอลัมน์ wPi (wiringPi) ให้ตรงกับขาเชื่อมต่อ 40 ขานบอร์ด 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		
-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+												

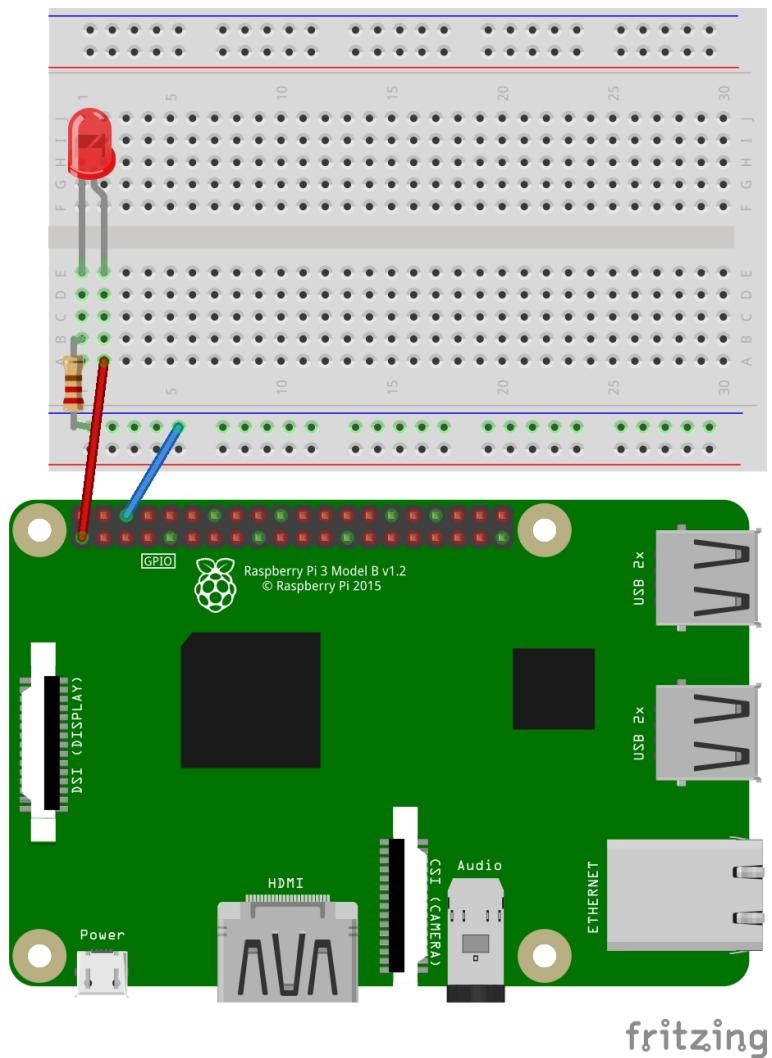
J.2 วงจรไฟ LED กระแสพิรบ

1. รายการอุปกรณ์ที่ต้องใช้:

- หลอด LED จำนวน 1 หลอด
- ตัวต้านทาน (Resistor) ขนาด 2-10 กิโลโวท์ม จำนวน 1 ตัว
- แผ่นต่อวงจรprotoboard
- สายต่อวงจรชนิดต่างๆ: ผู้-เมีย (Male-Female) และ ผู้-ผู้ (Male-Male) จำนวนหนึ่ง

2. ชัตดาวน์และตัดไฟเลี้ยงออกจากบอร์ด Pi เพื่อความปลอดภัยในการต่อวงจร

3. ศึกษาตารางที่กรอกก่อนหน้านี้ให้เข้าใจ แล้วจึงต่อวงจรตามรูปที่ J.1



รูปที่ J.1: วงจรเชื่อมต่อหลอด LED กับบอร์ด Pi ในการทดลองที่ 10 เพื่อทดสอบว่าหลอด LED ทำงาน
ที่มา: fritzing.org

4. จงวัดวงจรที่ต่อในรูปที่ J.1 ประกอบด้วย ตัวต้านทานไฟเลี้ยง 3.3 โวลต์ ขา LED และกราวน์ด (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..." เลือก แท็บ "Linker settings" และกดปุ่ม "Add"
4. ป้อนประযุค "/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 ขอความช่วยเหลือจากผู้ควบคุมการทดลอง
10. จับเวลาช่วงเวลาที่หลอดสว่าง และตั้งแต่เริ่มรันโปรแกรมจนเสร็จสิ้น เพื่อหาค่าเฉลี่ยของการสว่างตั้ง 1 รอบ

J.4 โปรแกรมไฟ LED กระพริบภาษาแอสเซมบลี

1. เปิดไดร์กอรี `/home/pi/asm` ในโปรแกรมไฟล์เมเนเจอร์
2. สร้างไดร์กอรีใหม่ชื่อ `Lab10`
3. สร้างไฟล์ใหม่ชื่อ `Lab10.s` โดยใช้คำสั่ง `touch`
4. กรอกโปรแกรมภาษาแอสเซมบลีเหล่านี้โดยใช้ editor ที่ถนัด

```
#-----
# 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. จับเวลาช่วงเวลาที่หลอดสว่าง และดับตั้งแต่เริ่มรันโปรแกรมจนเสร็จสิ้น เพื่อหาค่าเฉลี่ยของการสว่างดับ 1 รอบ

J.5 กิจกรรมท้ายการทดลอง

1. ไลบรารี libwiringPi.so ทำหน้าที่อะไร และเกี่ยวข้องกับ #include <wiringPi.h> อย่างไร
2. ประโยชน์ \$ gcc -o Lab10 Lab10.o -lwiringPi มีความหมายอย่างไร และเชื่อมโยงกับคำคำขอที่แล้วอย่างไร
3. พิงก์ชัน digitalWrite ใช้กับขา GPIO ในโหมดไหน
4. ประโยชน์ PUSH {ip, lr} ทำหน้าที่อะไร เหตุใดจึงต้องเรียกใช้ก่อนประโยชน์อื่น ๆ
5. ประโยชน์ POP {ip, pc} ทำหน้าที่อะไร เหตุใดจึงต้องเรียกใช้เป็นประโยชน์สุดท้าย
6. คลิก บน ลิงก์ ชื่อ <https://github.com/WiringPi/WiringPi/blob/master/wiringPi/wiringPi.c> เพื่อใช้เบราว์เซอร์เปิดและตอบคำถามต่อไปนี้
7. สำรวจไฟล์ชื่อ wiringPi.c ที่เปิดเพื่อค้นหาตัวแปรชื่อ piGpioBase ว่า

- ใช้งานในฟังก์ชันชื่ออะไร
- ได้รับการตั้งค่าที่ฟังก์ชันชื่ออะไร และค่าเท่ากับเท่าไร
- นำตัวแปร piGpioBase นี้ไปใช้ทำอะไรต่อได้อีก จงยกตัวอย่าง
- หมายเลข แอดเดรส 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) ;
```

- อยู่ในฟังก์ชันชื่ออะไร
- ตัวแปร fd มาจากไหน เกี่ยวข้องกับไฟล์ /mem และไฟล์ /dev/gpiomem อย่างไร
- ฟังก์ชัน mmap() มีหน้าที่อะไร รีเทิร์นค่าอะไรกลับมา และเป็นตัวแปรชนิดใด เหตุใดจึงต้องมีประโยชน์ (uint32_t *) นำหน้า
- นำตัวแปร gpiob นี้ไปใช้ทำอะไรต่อได้อีก จงยกตัวอย่าง
- จงอธิบายว่าตัวแปร gpiob นี้เกี่ยวข้องกับหลักการ Memory Map IO อย่างไร

9. จงตอบคำถามจากประโยชน์ต่อไปนี้

```
GPIO_BASE = piGpioBase + 0x00200000 ;
```

- อยู่ในฟังก์ชันชื่ออะไร
- ตัวแปร GPIO_BASE มีหน้าที่อะไร
- เมื่อ บวก แล้ว ได้ ผลลัพธ์ เป็น หมายเลข แอดเดรส อะไร และ เกี่ยวข้อง กับ หมายเลข 0x7E20 0000 ในตารางที่ [6.6](#) อย่างไร
- นำตัวแปร GPIO_BASE นี้ไปใช้ทำอะไรต่อได้อีก จงยกตัวอย่าง
- จงอธิบายว่าตัวแปร GPIO_BASE นี้เกี่ยวข้องกับขา gpiob แต่ละขาอย่างไร

10. ต่อหลอด LED เพิ่มอีก 2 ดวงรวมเป็น 3 ดวงแล้วพัฒนาโปรแกรมภาษา C เดิมให้นับเลขจำนวนเต็มฐานสิบ 0 - 7 และแสดงผลทางหลอด LED เป็นเลขฐานสองวนไปเรื่อย ๆ

11. ใช้วงจรหลอด LED 3 ดวงที่มีอยู่ และพัฒนาโปรแกรมภาษา C และเซมบลิ เดิมให้นับเลขจำนวนเต็มฐานสิบ 0 - 7 และแสดงผลทางหลอด LED เป็นเลขฐานสองวนไปเรื่อย ๆ

ภาคผนวก K

การทดลองที่ 11 การเชื่อมต่ออินพุต-เอาต์พุตกับสัญญาณอินเทอร์รัปต์

การทดลองนี้คาดว่าผู้อ่านเคยเรียนการเขียนหรือพัฒนาโปรแกรมด้วยภาษา C และแօสเซมบลีจาก การทดลองก่อนหน้า ดังนั้น การทดลองมีวัตถุประสงค์เหล่านี้

- เพื่อพัฒนาการทำงานของอินเทอร์รัปต์ร่วมโปรแกรมภาษา C และแօสเซมบลี ตามเนื้อหาในหัวข้อที่ [6.12](#)
- เพื่อศึกษาการทำงานของอินเทอร์รัปต์ร่วมกับขา GPIO ตามเนื้อหาในหัวข้อที่ [6.11](#)

K.1 การจัดการอินเทอร์รัปต์ของ WiringPi

ไลบรารี wiringPi รองรับการทำอินเทอร์รัปต์ของ GPIO ได้ ทำให้โปรแกรมหลักสามารถทำงาน หลักได้ตามปกติ เมื่อเกิดสัญญาณอินเทอร์รัปต์ขึ้น ไม่ว่าจะเป็นสัญญาณจากการกดปุ่มสวิตช์ ทำให้เกิดขอบขาขึ้นหรือขอบขาลง หรือทั้งสองขอบ โดยการเรียกใช้คำสั่ง

```
wiringPiISR(pin, edgeType, &callback)
```

โดย pin หมายถึง เลขชาที่ wiringPi กำหนด edgeType กำหนดจากค่าคงที่ 4 ค่านี้

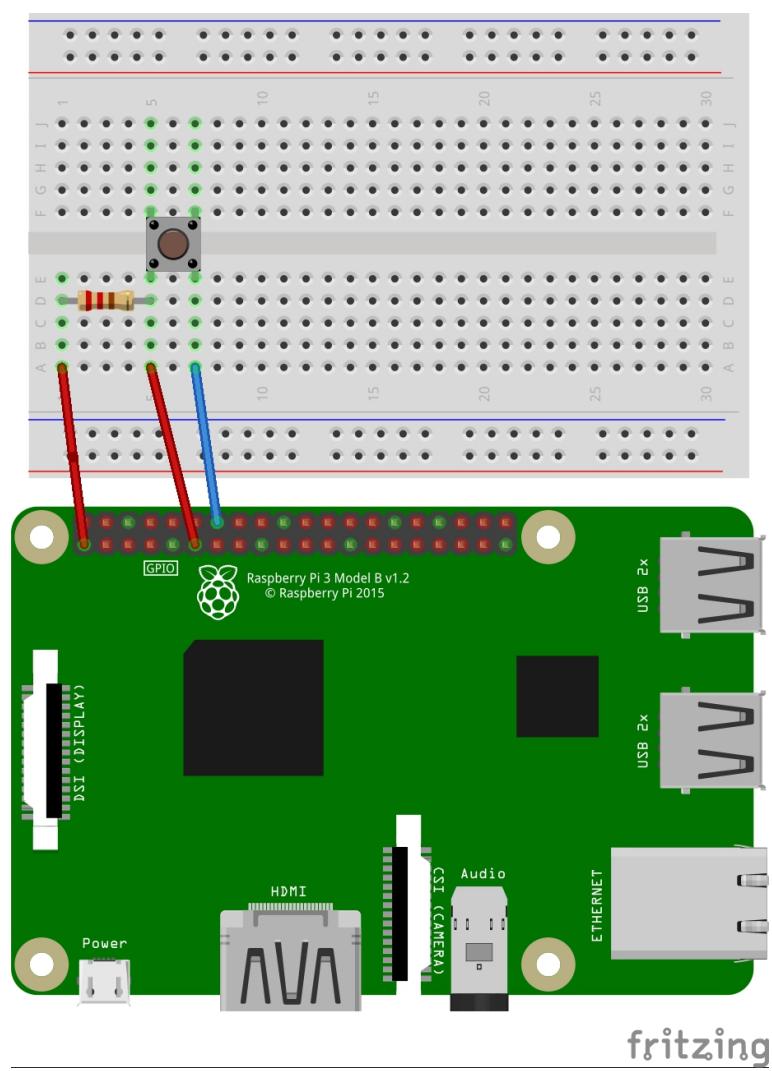
- INT_EDGE_FALLING,
- INT_EDGE_RISING,
- INT_EDGE_BOTH
- INT_EDGE_SETUP.

การกำหนดชนิดของขาเป็น 3 ชนิด แรก ไลบรารี จะตั้งค่าเริ่มต้น (Initialization) ให้โดยอัตโนมัติ หากกำหนดชนิดของเป็น INT_EDGE_SETUP ไลบรารีจะไม่ตั้งค่าเริ่มต้น (Initialization) ให้ เนื่องจาก โปรแกรมเมอร์จะต้องกำหนดเอง

พารามิเตอร์ **callback** คือ ชื่อฟังก์ชันที่จะทำหน้าที่เป็น ISR สัญลักษณ์ & หมายถึง แอดเดรสของ ฟังก์ชัน callback ฟังก์ชัน callback นี้จะเริ่มต้นทำงานโดยแจ้งต่อว่าง Dispatcher ในหัวข้อที่ 6.12 ก่อนจะเริ่มต้นทำงาน โดยฟังก์ชัน callback จะสามารถอ่าน หรือเขียนค่าของตัวแปรโกลบอลในโปรแกรม ได้ ซึ่งตัวอย่างการทำงานจะได้กล่าวในหัวข้อถัดไป

K.2 วงจรสวิตซ์ปุ่มกดเชื่อมผ่านขา GPIO

1. ซัตดาวน์และตัดไฟเลี้ยงออกจากบอร์ด Pi เพื่อความปลอดภัยในการต่อวงจร
2. ต่อวงจรตามรูปที่ K.1



รูปที่ K.1: วงจรสวิตซ์ปุ่มกดสำหรับทดลองการเขียนโปรแกรมอินเทอร์รัปต์ในการทดลองที่ 11 ที่มา: fritzing.org

3. จงวัดวงจรที่ต่อในรูปที่ K.1 ประกอบด้วย สวิตช์ปุ่มกด ตัวต้านทาน ไฟเลี้ยง 3.3 โวลต์ ขา BUT-TON_PIN และกราวน์ด (0 โวลต์)
4. ตรวจสอบความถูกต้อง โดยให้ผู้ควบคุมการทดลองตรวจสอบ
5. สร้าง project ใหม่ชื่อ Lab11 ภายใต้ไดเรกทอรี /home/pi/asm/Lab11

K.3 โปรแกรมภาษา C สำหรับทดสอบวงจรอินเทอร์รัปต์

ผู้อ่านต้องทำความเข้าใจ กับ ตัวโปรแกรม ก่อน คอมไฟล์ หรือ รันโปรแกรม เพื่อ ความเข้าใจ สูงสุด โดย เนพาะ ชื่อ ตัวแปร ชนิด ของ ตัวแปร evenCounter การติดตั้ง พังก์ชัน wiringPiISR เพื่อ เชื่อมโยง กับขา GPIO ชนิด ของการ ตรวจจับ และ ชื่อ พังก์ชัน myInterrupt ซึ่ง ทำหน้าที่ เป็น ISR หรือ พังก์ชัน 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)
}

```

- ป้อนโปรแกรมด้านบนใน main.c โดยใช้โปรแกรม Text Editor ทั่วไป
- สร้าง makefile สำหรับคอมpile และลิงค์โปรแกรม จากการทดลอง ก่อนหน้านี้ จะไม่เกิดข้อผิดพลาด
- รันโปรแกรม ทดสอบการทำงานด้วยการกดปุ่มสวิตช์ที่ต่อไว้ สังเกตผลลัพธ์ทางหน้าจอ Terminal ที่รัน

\$ sudo ./Lab11

K.4 กิจกรรมท้ายการทดลอง

- จงวัดสัญญาณที่ขา BUTTON_PIN ก่อนกดปุ่มสวิตช์ ระหว่างกดปุ่มสวิตช์ และปล่อยมือจากสวิตช์ บูมกด โดยให้แก่นอนเป็นแกนเวลา แกนตั้งเป็นค่าโวลเตจ หรือ ค่าลอจิกของขาสัญญาณ BUTTON_PIN
- จงบอกความหมายและการประยุกต์ใช้งานตัวแปรชนิด volatile
- ปรับแก้ volatile ออกรีอแคร์ int eventCount = 0;
make แล้วจึงรันโปรแกรมทดสอบการทำงานด้วยการกดปุ่มสวิตช์ที่ต่อไว้ สังเกตผลลัพธ์ทางหน้าจอ Terminal ที่รัน เปรียบเทียบการทำงานของโปรแกรมก่อนและหลังการปรับแก้ และหาเหตุผล
- จงปรับแก้โปรแกรมที่ทดลองตามประโยชน์ต่อไปนี้

```

if (wiringPiISR (BUTTON_PIN, INT_EDGE_RISING, &myInterrupt) < 0) {
    ...
}

```

ทำ make ใหม่และทดลองกดปุ่มสวิตช์ สังเกตการเปลี่ยนแปลงและอธิบาย

- จงตอบคำถามจากประโยชน์ต่อไปนี้

```

if (wiringPiISR (BUTTON_PIN, INT_EDGE_FALLING, &myInterrupt) < 0) {
    ...
}

```

- พังก์ชัน wiringPilSR ทำหน้าที่อะไร เหตุใดอยู่ในประโยชน์เชื่อว่า if
 - ตัวแปร &myInterrupt คืออะไร เหตุใดจึงมีสัญลักษณ์ & นำหน้า
 - พังก์ชันนี้เชื่อมโยงกับตารางที่ [6.6](#) อย่างไร
6. จงใช้วงจรหลอด LED 3 ดวงและโปรแกรมจากการทดลองที่ 10 นับขึ้นจาก 0-7-0 โดยเพิ่มสวิตซ์ปุ่มกดในการทดลองนี้ และเพิ่มพังก์ชันการอินเทอร์รัปต์จากโปรแกรม 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 โดยเพิ่มสวิตซ์ปุ่มกดในการทดลองนี้ และเพิ่มพังก์ชันการอินเทอร์รัปต์จากโปรแกรม Lab11.2 นี้ เมื่อกดปุ่มแต่ละครั้งจะทำให้ความเร็วในการนับลดลง หรือ Delay เพิ่มขึ้นเท่าตัว เมื่อกดครั้งที่ 2 Delay เพิ่มขึ้นอีกเท่าตัว เมื่อกดครั้งที่ 3 จะทำให้ Delay กลับไปเป็นค่าเริ่มต้น

ภาคผนวก L

การทดลองที่ 12 การศึกษาอุปกรณ์เก็บรักษาข้อมูลและระบบไฟล์

การทดลองนี้อธิบายและเขียนอย่างเนื้อหาความรู้ของทุกบทเข้าด้วยกัน แต่จะเน้นบทที่ 6 และบทที่ 7 เพื่อให้ผู้อ่านมองเห็นอุปกรณ์อินพุตและเอาต์พุตเมื่อไฟล์แต่ละไฟล์ โดยมีวัตถุประสงค์ดังนี้

- เพื่อให้เข้าใจการวัดขนาดของไฟล์และไดเรกทอรีในระบบไฟล์
- เพื่อให้รู้จักโครงสร้างและระบบไฟล์ของการ์ดหน่วยความจำไมโคร SD ที่ใช้งานในปัจจุบัน
- เพื่อให้เข้าใจระบบไฟล์ (File System) ชนิดต่าง ๆ บนบอร์ด Pi
- เพื่อให้สามารถเขียนอย่างอุปกรณ์อินพุต/เอาต์พุตชนิดต่าง ๆ กับระบบไฟล์

L.1 ขนาดของไฟล์และไดเรกทอรี

ผู้อ่านสามารถตรวจสอบขนาดของไฟล์ได ฯ ชื่อ filename ที่แท้จริง หน่วยเป็นไบต์ ด้วยคำสั่ง du (Disk Usage) โดยทำตามขั้นตอนต่อไปนี้

- ย้ายไดเรกทอรีปัจจุบันไปที่ /home/pi ซึ่งเป็นไดเรกทอรีหลักของผู้ใช้ชื่อ pi

```
$ cd /home/pi
```

- สร้างไฟล์ข้อความ test.txt ด้วยโปรแกรม nano ด้วยคำสั่งต่อไปนี้

```
$ nano test.txt
```

พิมพ์ข้อความ fdd ลงในไฟล์ ทำการ Write โดยกดปุ่ม Ctrl แซ่ต้ามด้วยปุ่ม 0 ออกจากโปรแกรม โดยกดปุ่ม Ctrl แซ่ต้ามด้วยปุ่ม x

- คำสั่ง ‘du -b filename’ จะแสดงผลขนาดเป็นจำนวนไบต์หน้าชื่อไฟล์นั้น

```
$ du -b test.txt
```

```
_ test.txt
```

ตัวเลข _____ หมายถึง เลขจำนวนไบต์ที่คำสั่ง du แสดงผลมาตามพารามิเตอร์ b ที่ส่งไป เพื่อบอกค่าขนาดของไฟล์ test.txt เป็นจำนวน _____ ไบต์

- คำสั่ง ‘du -B1 filename’ ผู้อ่านสามารถตรวจสอบขนาดของไฟล์ได ๆ ชื่อ filename ที่จัดเก็บเป็นจำนวนเท่าของ _____ ไบต์ ในอุปกรณ์เก็บรักษาข้อมูล SD ด้วยคำสั่งต่อไปนี้

```
$ du -B1 test.txt
```

```
_____ test.txt
```

ตัวเลข _____ หมายถึง เลขจำนวนไบต์ที่คำสั่ง du แสดงผลมาตามพารามิเตอร์ B1 ที่ส่งไป โดยผู้อ่านจะสังเกตเห็นความแตกต่าง ถึงแม้ไฟล์มีข้อมูลจำนวนน้อยเพียงไม่ถึงไบต์ แต่การจองพื้นที่ในอุปกรณ์สำรองจะมีขนาดเป็นจำนวนเท่าของ _____ ไบต์เสมอ เช่น 8192, 16384 เป็นต้น

- คำสั่ง ‘du -h’ จะแสดงผลขนาด หรือ จำนวนไบต์โดยใช้หน่วย เช่น K (Kibi: 1024) M (Mebi: 1048576) G (Gibi: 1073741824) หน้าชื่อไดเรกทอรี หรือ โฟลเดอร์ที่อยู่ใต้ไดเรกทอรีปัจจุบัน และจดบันทึก 5 รายการแรกในตาราง

```
$ du -h
```

Size	Folder Name

L.2 ระบบไฟล์

ผู้ใช้หรือผู้ดูแลระบบลินุกซ์ สามารถตรวจสอบการใช้งานอุปกรณ์เก็บรักษาข้อมูล เช่น ฮาร์ดดิสก์ไดรฟ์ โซลิดสเตทไดรฟ์ 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’ จะแสดงรายการต่าง ๆ ดังนี้ จดบันทึก 5 รายการที่ตรงกับตารางที่แล้ว

```
$ df -i
```

Filesystem	Inodes	IUsed	IFree	IUse%	Mounted on

โดยคอลัมน์ที่ 2 จากทางซ้ายจะแสดงผลเป็นจำนวน ไオหนด แทน รายละเอียดเรื่องไอหนด ผู้อ่าน สามารถค้นคว้าเพิ่มเติมได้ในบทที่ 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: _____
Access: ( _____ / _____ drwxr-xr-x) Uid: ( _____ /
_____) Gid: ( _____ / _____ )
Access: ...
Modify: ...
Change: ...
Birth: -
```

ผู้อ่านจะต้องกรอกผลลัพธ์ในช่องว่าง ดังต่อไปนี้

- ชื่อ asm

- ขนาด _____ ไบต์ ใช้พื้นที่จำนวน _____ Blocks ซึ่งหมายถึง 8 เข็กเตอร์ ๆ ละ 512 ไบต์ คิดเป็น _____

- มีหมายเลข Device = h/_/_/_d หรือเท่ากับ 16/_/_/_10
- มีหมายเลข Inode = 10 จำนวน _ Links
- สิทธิ์เข้าถึง (Access Permission) ด้วยรหัส 16 หรือ 2:_2:_2 โดยผู้ใช้หมายเลข Uid (User ID)= ชื่อผู้ใช้ (Username)= ในกรุ๊ปหมายเลข Groupid= ชื่อกรุ๊ป
--
- เข้าถึง (Access) ...
- เปลี่ยนแปลง (Modify) ...
- เวลาที่ Inode เปลี่ยนแปลง (Change) ...

เบื้องต้นผู้เขียนขอให้ผู้อ่านสร้างไฟล์ผลลัพธ์จากคำสั่ง stat ไปเก็บในไฟล์ เพื่อมาใช้ประกอบการทดลองต่อไป โดย

```
$ stat asm > stat_asm.txt
```

หลังจากนั้น เราสามารถตรวจสอบสถานะของไฟล์ stat_asm.txt ได้ดังนี้

```
$ stat stat_asm.txt
```

```
File: stat_asm.txt
Size:        Blocks:        IO Block:         
Device: h/_/_/_d      Inode:        Links:  
Access: (-rw-r--r--)  Uid: (/ )  Gid: (/ )
           )
Access: ...
Modify: ...
Change: ...
Birth: -
```

ผู้อ่านจะต้องกรอกผลลัพธ์ในช่องว่าง ดังต่อไปนี้เป็นรายบรรทัด

- ชื่อ stat_asm.txt
- ขนาด ไบต์ ใช้พื้นที่จำนวน _ Blocks ซึ่งหมายถึง 8 เช็กเตอร์ ๆ ละ 512 ไบต์ คิดเป็น ไบต์ เป็น
- มีหมายเลข Device = h/_/_/_d หรือเท่ากับ 16/_/_/_10

- มีหมายเลข Inode = _____₁₀ จำนวน _____ Links
- สิทธิ์เข้าถึง (Access Permission) ด้วยรหัส _____₁₆ หรือ _____₂:_____₂:_____₂ โดยผู้ใช้หมายเลข Uid (User ID)= _____ ชื่อผู้ใช้ (Username)= _____ ในกรุ๊ปหมายเลข Groupid= _____ ชื่ogrุ๊ป _____
- 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 0765₁₆ มีความหมายดังต่อไปนี้
 - 111₂: เป็นของครอ _____
 - 110₂: เป็นของครอ _____
 - 101₂: เป็นของครอ _____

L.3 อุปกรณ์อินพุต/เอาต์พุตในระบบไฟล์

การทดลองในหัวข้อนี้จะเขียนต่อ กับเนื้อหาในบทที่ 3 และ การทดลองที่ 4 ภาคผนวก D หลักการของระบบปฏิบัติการยูนิกซ์ คือ การเมอาท์ (Mount) อุปกรณ์กับ ไดเรกทอรีด้วยระบบไฟล์ (File System) ที่แตกต่างกัน โดยใช้ชื่อ ไดเรกทอรี ที่แตกต่างกัน โดยมี ไดเรกทอรีราก (Root Directory) หรือ โฟลเดอร์ราก เป็นตำแหน่งเริ่มต้น ผู้อ่านสามารถพิมพ์คำสั่งใน Terminal

```
$ mount
```

คำสั่งนี้จะแสดงรายชื่อ การเมอาท์ หรือ ผูกยึด อุปกรณ์อินพุต/เอาต์พุต เข้ากับระบบไฟล์ที่เหมาะสม กับ อุปกรณ์นั้น ๆ ด้วยชื่อ ไดเรกทอรี หรือ ชื่อไฟล์ของระบบปฏิบัติการ ผู้อ่าน จะต้องกรอกผลลัพธ์ที่สำคัญ ใน ช่องว่าง และศึกษาคำอธิบายต่อไปนี้

- /dev/mmcblk0p_____ on / type ext4 (rw,noatime) เป็นระบบไฟล์ **ext4** ซึ่งเป็นระบบไฟล์ หลักของลินุกซ์ ย่อมาจากคำว่า Fourth Extended File System เป็นเวอร์ชันที่ 4 พัฒนาจากชื่อ 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) สำหรับระบบสำคัญต่าง ๆ 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) สำหรับอุปกรณ์อินพุตเอาต์พุตต่าง ๆ รายละเอียดเพิ่มเติมที่ [wikipedia](#)
- ...
- /dev/mmcblk0p on /boot type vfat ระบบไฟล์ **vfat** เป็นส่วนต่อขยายของระบบไฟล์ FAT ซึ่งย่อมาจากคำว่า File Allocation Table เพื่อรองรับชื่อไฟล์ที่ยาวกว่า FAT ที่มา: [wikipedia](#)
- ...

รายชื่อต่อไปนี้คือ ตัวเลือกคุณสมบัติ (Attribute) ที่สำคัญของระบบไฟล์ เช่น

- rw : read/write สามารถอ่านและเขียนได้
- noatime และ atime: No/ Access Time หมายถึง ไม่มี/มีการบันทึกเวลาในการสร้าง อ่านหรือเขียนไฟล์ทุกครั้ง
- relatime หมายถึง มีการบันทึกเวลาในการสร้าง อ่านหรือเขียนไฟล์ เมื่อเกิดการแก้ไขไฟล์ หรือการอ่านหรือเข้าถึงไฟล์มากกว่าเวลาที่บันทึกไว้ก่อนหน้าอย่างน้อย 24 ชั่วโมง
- nosuid: No SuperUser ID เป็นการป้องกันไม่ให้ผู้ดูแลระบบ (SuperUser) กระทำการใด ๆ ได้เพื่อความมั่นคงปลอดภัย
- noexec: No Execution เพื่อตั้งค่าไม่ให้รันไฟล์ที่อยู่ในไดร์กหรือนี้ได้ 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
```

ผู้อ่านต้องเปรียบเทียบกับชื่ออุปกรณ์ที่ผู้เขียนตัวหนาไว้ว่าตรงกันหรือไม่ อย่างไร เพื่อให้ผู้อ่านมองเห็นชัดว่า mmcblk0p2 มีอยู่จริง และระบบได้ทำการมาที่เข้ากับไดเรกทอรีรoot (Root) นั่นคือไดเรกทอรี / 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 kmsq 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 rpidid-h264mem rpidid-hevcmem rpidid-initc rpidid-vp9mem serial1 shm snd stderr
stdin stdout tty tty0 ... ttyAMA0 ttyprintk uhid uinput urandom vchiq vcio vc-mem vcs ...
watchdog watchdog0 zero
```

นอกจากนี้ อุปกรณ์สำคัญอื่น ๆ เช่น stdin (standard input) stdout (standard output) และ stderr (standard error) นั้นเกี่ยวข้องกับโปรแกรม Terminal ซึ่งเขียนโดยกับภาษา C ในการทดลองที่ 5 ภาคผนวก E

```
#include <stdio.h>
```

L.4 กิจกรรมท้ายการทดลอง

1. ใช้โปรแกรม File Manager แล้วคลิกขวาบนชื่อไฟล์เพื่อแสดงคุณสมบัติ (Properties) ต่าง ๆ บนแท็บ General และอธิบายโดยเฉพาะหัวข้อ Total size of files และ Size on disk ว่าเหตุใดถึงแตกต่างกัน
2. สร้างไฟล์ (New) ด้วยโปรแกรม nano คีย์ข้อความด้วยตัวอักษรจำนวน 1 ตัวแล้วบันทึก (Save) ใช้คำสั่ง `Ctrl + X` เพื่อแสดงรายละเอียดของไดเรกทอรีที่บรรจุไฟล์นั้น เพื่อหาขนาดไฟล์ที่แท้จริง
3. โปรดสังเกตว่าใน test.txt ที่สร้างด้วยโปรแกรม nano เราได้พิมพ์ประโยค fdd คิดเป็นจำนวน 3 ตัวอักษร ๆ ละ 1 ไบต์เท่านั้น จงหาว่าไบต์ที่ 4 คือตัวอักษรใดในรูปที่ [2.12](#)

4. เพิ่มจำนวนตัวอักษรไปเรื่อย ๆ ใน test.txt จนทำให้ไฟล์มีขนาดมากกว่าเท่ากับ 4096 ไบต์ แล้วใช้คำสั่ง du -B1 test.txt ตรวจสอบขนาดไฟล์อีกรอบ บันทึกและอธิบายผลที่ได้โดยเฉพาะ จำนวน Blocks ที่ได้จากการคำสั่งว่าเท่ากับกี่เซกเตอร์
5. จงเปรียบเทียบผลลัพธ์ของคำสั่ง stat ระหว่าง ไดเรกทอรี และไฟล์
6. สิทธิ์การเข้าถึง (Permission) ของไดเรกทอรีหรือของไฟล์ประกอบด้วยบิตจำนวน 9 บิต แบ่งเป็น 3 ชุด ๆ ละ 3 บิต จงเรียงลำดับชุดต่าง ๆ ว่าเป็นของสิทธิ์ของใครบ้าง
7. จงใช้คำสั่งต่อไปนี้ เพื่อแสดงรายละเอียดของชื่อไดเรกทอรีและไฟล์ และ อธิบายผลว่าหมายเลขอุปกรณ์ที่อยู่ด้านซ้ายสุดคืออะไร และเหตุใดจึงมีค่าซ้ำ

```
$ ls -i -l /
```

8. จงใช้คำสั่งต่อไปนี้ เพื่อแสดงรายละเอียดของชื่อไดเรกทอรีคู่ที่ซ้ำจากข้อที่แล้ว และ อธิบายผลว่ามีอะไรที่แตกต่างกัน เพราะเหตุใด

```
$ stat /proc  
$ stat /sys
```

```
$ stat /dev  
$ stat /run
```

9. จงใช้คำสั่งต่อไปนี้ เพื่อแสดงรายละเอียดของอุปกรณ์ และ อธิบายว่ามีผลลัพธ์ที่แตกต่างกันหรือไม่ เพราะเหตุใด

```
$ stat /dev/mmcblk0p2  
$ stat /
```

10. จงอธิบายว่าเหตุใดไดเรกทอรี asound จึงอยู่ใต้ /proc ในหัวข้อที่ [I.2.3 การทดลองที่ 9 ภาคผนวก I](#)

11. จงอธิบายความเชื่อมโยงระหว่าง gpiomem ที่ได้จากการคำสั่ง ls /dev กับกิจกรรมท้ายการทดลองที่ 10 ภาคผนวก [J](#)

ภาคผนวก M

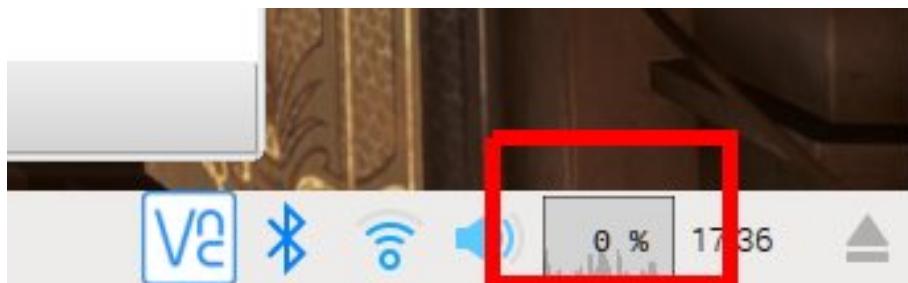
การทดลองที่ 13 การพัฒนาอัลกอริธึมแบบขนาน ด้วยไลบรารี OpenMP

การพัฒนาอัลกอริธึมแบบขนานบนชีพียูชนิดมัลติคอร์ในปัจจุบันจำเป็นต้องอาศัยภาษาคอมพิวเตอร์ระดับสูง เช่น ภาษา C/C++ และภาษา Java เป็นต้น เพื่อช่วยลดเวลาการ 실행 (Run Time) ซึ่งเท่ากับการเร่งความเร็ว (Speedup) ให้อัลกอริธึมหรือโปรแกรม โดยการสร้างเฟรดผู้ช่วย (Worker Thread) และมอบหมายงานให้ไปรันบนแกนประมวลผลที่ยังว่างอยู่ ผู้อ่านสามารถประยุกต์ใช้หลักการนี้บนเครื่องคอมพิวเตอร์ทั่วไป จนถึงเครื่องเซิร์ฟเวอร์คอมพิวเตอร์ตามเนื้อหาในบทที่ 8 ดังนั้น การทดลองนี้วัดถูกประสิทธิภาพดังนี้

- เพื่อพัฒนาโปรแกรมภาษา C ด้วยไลบรารี OpenMP ให้สามารถทำงานแบบมัลติเฟรด และใช้งานชีพียูชนิดมัลติคอร์ได้เต็มที่
- เพื่อเรียนรู้การวัด CPU Utilization (%CPU) เวลาจริง (T_{real}) เวลาผู้ใช้ (T_{user}) และเวลาระบบ (T_{sys}) ในชีพียูชนิดมัลติคอร์
- เพื่อทำความเข้าใจการวัดประสิทธิภาพของอัลกอริธึมแบบขนานจากการประเมินความซับซ้อนเชิงเวลาด้วยพีชคณิต BigO ที่มา: [Rosen \(2002\)](#) และตัวชี้วัด Speedup ที่มา: [Patterson and Hennessy \(2016\)](#) จากเวลาที่วัดได้

M.1 การวัด CPU Utilization

ผู้อ่านสามารถติดตั้งเครื่องมือและกราฟในรูปที่ [M.1](#) แสดงการใช้งานชีพียู (CPU Usage Monitor) ย้อนหลังและค่าสรุป ณ เวลาปัจจุบันของบอร์ด Pi ประกอบการทดลองที่ 13 ตามขั้นตอนเหล่านี้



รูปที่ M.1: กราฟแสดงการใช้งานซีพียู (CPU Usage Monitor) ย้อนหลังและค่าสรุป ณ เวลาปัจจุบัน ที่มา: abload.de

1. เลื่อนเมาส์ไปบนตำแหน่งว่างของ Task Bar
2. คลิกขวา เพื่อให้เมนูต่อไปนี้ปรากฏขึ้นแล้วคลิกซ้ายเลือก Add/Remove Panel Items
3. คลิกที่แท็บ Panel Applets
4. เลื่อนรายการขึ้นลงเพื่อหารายการชื่อ CPU Usage Monitor และคลิก 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 โดยแท่งขาวสุดคือ วินาทีล่าสุด
7. เลื่อนเมาส์ไปบนกราฟแล้วคลิกขวาเพื่อเพิ่มการแสดงผลเป็นตัวเลขหน่วยเป็นเปอร์เซ็นต์ (%)
8. ทดสอบการทำงานโดยการเปิดคลิปเดียวกันบน [YouTube.com](https://www.youtube.com) ที่ความละเอียดแตกต่างกัน เช่น 240p, 480p และ 720p ทีละค่าเพื่อให้เห็นค่า $\%CPU_{max}$ ที่แตกต่าง

M.2 การคุณเมทริกซ์แบบขบวน

$$C = A \times B \quad (\text{M.1})$$

การคุณเมทริกซ์เป็นพื้นฐานของการคำนวณพื้นฐานทางวิทยาศาสตร์ และวิศวกรรมศาสตร์ กำหนดให้เมทริกซ์จตุรัส a ขนาด $N \times N$ สามารถเขียนในรูปแบบของอาร์เรย์ 2 มิติในภาษา C/C++ ได้ดังนี้

$$A = (A[i][j])$$

โดยดัชนีตัวแรก i คือ หมายเลขแถว มีค่าตั้งแต่ 0 ถึง N-1 ดัชนีตัวที่สอง j คือ หมายเลขคอลัมน์ มีค่าตั้งแต่ 0 ถึง N-1 ดังนั้น

$$A = \begin{pmatrix} A[0][0] & A[0][1] & \dots & A[0][N-1] \\ A[1][0] & A[1][1] & \dots & A[1][N-1] \\ \vdots & \vdots & \ddots & \vdots \\ A[N-1][0] & A[N-1][1] & \dots & A[N-1][N-1] \end{pmatrix}$$

เมื่อทำความเข้าใจพื้นฐานของเมทริกซ์ในรูปแบบของอาร์เรย์ 2 มิติแล้ว ผู้อ่านสามารถทำการทดลองตามขั้นตอนต่อไปนี้

- สร้างไดเรกทอรี `/home/pi/asm/Lab13` บนโปรแกรม Terminal ด้วยคำสั่งต่อไปนี้ตามลำดับ

```
$ cd /home/pi/asm
$ mkdir Lab13
$ cd Lab13
$ nano multMatrix.c
```

- 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 เพื่อคอมไพล์โปรแกรมด้วยคำสั่งต่อไปนี้

```
$ gcc -fopenmp multMatrix.c -o mulMatrix
```

แก้ไขหากมีข้อผิดพลาดจนกว่าจะคอมไпал์ล์โปรแกรมสำเร็จและมีไฟล์ชื่อ mulMatrix

4. ตั้งค่าจำนวน-thread $n=1$ ของโปรแกรมด้วยคำสั่งต่อไปนี้

```
$ export OMP_NUM_THREADS=1
```

5. รันโปรแกรมจับเวลาด้วยคำสั่ง time ตั้งนี้จำนวน 5 ครั้งเพื่อหาค่าเฉลี่ย ขณะทำการทดลองขอให้ผู้อ่านใช้คลือกข้อมูลจับเวลาไปพร้อม ๆ กัน เพื่อเปรียบเทียบกับค่าของ $T_{mul,n}$ และ T_{real}

```
$ time ./mulMatrix
```

ซึ่งจะรายงานผลการจับเวลาการทำงานของทั้งโปรแกรมในแต่ละมุมต่าง ๆ

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 $\%CPU_{max}$ ของการคูณเมทริกซ์ที่ขนาด N และจำนวนเรตเท่ากับ 1, 2, 4, 8 เเรต

เวลาเฉลี่ย (วินาที)	$N=200$ (วินาที)	$N=400$ (วินาที)	$N=800$ (วินาที)	$N=1000$ (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

ตารางที่ M.2: อัตราส่วนเวลาการคูณเมทริกซ์ที่ขนาด $N=400, 800, 1000$ เทียบกับเวลาที่ขนาด $N=200$ ที่จำนวนเฟรดเท่ากับ 1, 2, 4, 8 เฟรด จากสมการที่ ([M.2](#))

	$N=200$	$N=400$	$N=800$	$N=1000$
$n=1$ เฟรด $T_{N,1}/T_{200,1}$	1.00			
	$\sqrt[2]{T_{N,1}/T_{200,1}}$	1.00		
	$\sqrt[3]{T_{N,1}/T_{200,1}}$	1.00		
$n=2$ เฟรด $T_{mul,N}/T_{mul,200}$	1.00			
	$\sqrt[2]{T_{mul,N}/T_{mul,200}}$	1.00		
	$\sqrt[3]{T_{mul,N}/T_{mul,200}}$	1.00		
$n=4$ เฟรด $T_{mul,N}/T_{mul,200}$	1.00			
	$\sqrt[2]{T_{mul,N}/T_{mul,200}}$	1.00		
	$\sqrt[3]{T_{mul,N}/T_{mul,200}}$	1.00		
$n=8$ เฟรด $T_{mul,N}/T_{mul,200}$	1.00			
	$\sqrt[2]{T_{mul,N}/T_{mul,200}}$	1.00		
	$\sqrt[3]{T_{mul,N}/T_{mul,200}}$	1.00		

ความซับซ้อนเชิงเวลา (Run Time Complexity) ของการคูณเมทริกซ์เท่ากับ $O(N^3)$ ในทางทฤษฎีผู้อ่านสามารถประยุกต์ใช้อัตราส่วนระหว่าง $O(N_2^3)$ และ $O(N_1^3)$ ที่ภาระงานขนาด $N_2:N_1$ และจำนวน n เฟรดเหมือนกัน 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 เฮρดจากหัวข้อที่ผ่านมา ผู้อ่านสามารถคำนวณค่าอัตราส่วนของเวลาในตารางที่ M.2 เพื่อใช้วิเคราะห์ต่อไป

M.4 ประสิทธิภาพ (Performance) ของการคำนวณแบบบานาน

ผู้อ่านสามารถวัดประสิทธิภาพ (Performance) ของอัลกอริธึมได้ ๆ ได้จากอัตราส่วนของเวลาเดิม (T_{old}) และเวลาใหม่ (T_{new}) ที่ได้ทำการปรับปรุงอัลกอริธึมนั้น ๆ ที่มา: Patterson and Hennessy (2016)

$$\frac{Perf_{new}}{Perf_{old}} = \frac{T_{old}}{T_{new}} \quad (\text{M.3})$$

ดังนั้น ประสิทธิภาพของการคำนวณแบบบานานสามารถวัดได้จากอัตราส่วนระหว่างระยะเวลา $T_{alg,1}$ ของ 1 เฮรดและ $T_{alg,n}$ ของ n เฮรด และตั้งชื่อเรียกว่า $Speedup(n)$ ด้วยสมการต่อไปนี้

$$Speedup(n) = \frac{T_{alg,1}}{T_{alg,n}} \quad (\text{M.4})$$

โดย $T_{alg,n}$ คือ ช่วงการรันโปรแกรมอัลกอริธึมด้วยจำนวน n เฮรด โดยไม่รวมช่วงเวลาอื่น ๆ ซึ่งไม่ได้เกี่ยวข้องกับการอัลกอริธึมแบบบานาน ผู้อ่านสามารถประยุกต์ตัวชี้วัดนี้กับอัลกอริธึมการคูณเมทริกซ์ดังนี้

$$Speedup(n) = \frac{T_{mul,1}}{T_{mul,n}} \quad (\text{M.5})$$

โดย $T_{mul,n}$ คือ ช่วงการรันโปรแกรมคำนวณเมทริกซ์จริง ๆ ด้วยจำนวน n เฮรด ที่ขนาด N เท่ากันโดยไม่รวมช่วงเวลาสุ่มค่าตั้งต้น และการแสดงผลอื่น ๆ ผู้อ่านคำนวณค่า $Speedup(n)$ และกรอกในตารางที่ M.3 เพื่อวิเคราะห์ผลการคำนวณที่ได้โดยตอบคำถามในกิจกรรมท้ายการทดลอง

ตารางที่ M.3: ผลการคำนวณ $Speedup(n)$ ของการคูณเมทริกซ์ที่ขนาด $N=200, 400, 800, 1000$ และจำนวนเฮรดเท่ากับ 2, 4, 8 เฮรดเทียบกับ 1 เฮรดด้วยสมการที่ (M.5)

Speedup	$N=200$	$N=400$	$N=800$	$N=1000$
$n=2$ เฮรด $Speedup(2) = T_{mul,1}/T_{mul,2}$				
$n=4$ เฮรด $Speedup(4) = T_{mul,1}/T_{mul,4}$				
$n=8$ เฮรด $Speedup(8) = T_{mul,1}/T_{mul,8}$				

M.5 กิจกรรมท้ายการทดลอง

1. เหตุใดการทดลองจึงต้องใช้การหาค่าเฉลี่ยเวลาต่าง ๆ
2. T_{sys} หมายถึง เวลาซึ่งทำงานประเภทไหน
3. T_{user} หมายถึง เวลาซึ่งทำงานประเภทไหน
4. T_{real} มีความสัมพันธ์กับ T_{mul} อย่างไร
5. T_{user} มีความสัมพันธ์กับ T_{mul} และจำนวนเฟรด n อย่างไร
6. เมื่อ $N_1=200$ จะเปรียบเทียบค่าผลการคำนวนของ $\sqrt[2]{T_{mul,N_2}/T_{mul,200}}$ 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 เฟรดตามลำดับ ว่ามีค่าใกล้เคียงกับ $N_2/200= 2, 4, 5$ ตามลำดับอย่างไร เพราะเหตุใด
7. จำนวนเฟรด และ จำนวนแกนประมวลผล มีผลต่อค่า Speedup อย่างไร วิเคราะห์ทั้งหมด 3 กรณี ดังนี้
 - จำนวนเฟรด < จำนวนแกนประมวลผล
 - จำนวนเฟรด = จำนวนแกนประมวลผล
 - จำนวนเฟรด > จำนวนแกนประมวลผล
8. เหตุใดค่าเฉลี่ยเวลา T_{user} จึงไม่แตกต่างกัน ที่ N คนที่
9. เวลาส่วนใหญ่ของการรัน T_{real} T_{user} และ T_{sys} สัมพันธ์กันอย่างไร จะสร้างสมการ
10. จำนวนเฟรดที่เพิ่มขึ้นทำให้การคำนวนเร็วขึ้นอย่างไร มีข้อจำกัดหรือไม่
11. ที่ขนาดข้อมูล $N=1000$ จำนวนเฟรดที่เพิ่มขึ้นทำให้ T_{user} เปลี่ยนแปลงอย่างไร มีข้อจำกัดหรือไม่
12. ที่ขนาดข้อมูล N ต่าง ๆ ค่า $\%CPU_{max}$ มีการเปลี่ยนแปลงและมีความสัมพันธ์กับ จำนวนเฟรด n อย่างไร
13. ขนาดข้อมูล N ที่เพิ่มขึ้นมีผลต่อ $Speedup(n)$ ที่ $n=1, 2, 4$ และ 8 หรือไม่ อย่างไร

បរណាន់ក្រម (Bibliography)

- Allwinner Technology, Co. (2015). *Allwinner A64 Mobile Application Processor Datasheet Version 1.1*. Allwinner Technology, Co.
- Anderson, T. and M. Dahlin (2012). *Principles and Practice (Second Edition)* (2nd ed.). USA: Recursive Books, Ltd.
- ARM Ltd. (2011). *Cortex-A7 MPCore Revision: Technical Reference Manual* (r0p3 ed.). UK: ARM Ltd.
- ARM Ltd. (2017). *ARM Generic Interrupt Controller Architecture Specification GIC architecture version 3.0 and version 4.0* (1 ed.). UK: ARM Ltd.
- Broadcom Corp. (2012). *BCM2835 ARM Peripherals*. Broadcom Corp.
- Choi, K. (2010). *NAND Flash Memory*. Samsung Electronics, Co., Ltd.
- Cypress Semiconductor, Corp. (2002). *256K(32Kx8) Static RAM CY62256*. Cypress Semiconductor, Corp.
- Cypress Semiconductor, Corp. (2017). *Single-Chip IEEE 802.11ac b/g/n MAC/Baseband/ Radio with Integrated Bluetooth 4.1 and FM Receiver*. Cypress Semiconductor, Corp.
- Demblon, S. and S. Spitzner (2004). *Linux Internals: (to the power of -1)* (1 ed.). The Shuttleworth Foundation.
- Diodes, Inc. (2012). *PAM2306 Dual High-Efficiency PWM Step-Down DC-DC Converter*. Diodes, Inc.
- 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 of London*.
- Mano, M. M. and C. Kime (2007). *Logic and Computer Design Fundamentals* (4th ed.). USA: Prentice Hall Press.
- Microchip Technology, Inc. (2009). *LAN9514/LAN9514i USB 2.0 Hub and 10/100 Ethernet Controller*. Microchip Technology, Inc.
- Micron Technology, Inc. (2004). *NAND Flash Memory*. Micron Technology, Inc.
- Micron Technology, Inc. (2014). *Embedded LPDDR2 SDRAM EDB8132B4PB-8D-F*. Micron Technology, Inc.
- Muller, Richard S.; Kamins, T. I. C. M. (2003). *Device Electronics for Intergrated Circuits* (3rd ed. ed.). New Delhi;: Wiley india.,
- 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 (ppmq-sort) 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.
- Rattanatranurak, A. and S. Kittitornkun (2020). A multistack parallel (msp) partition algorithm applied to sorting. *Journal of Mobile Multimedia*, 293–316.
- Rattanatranurak, 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 (2003). *SanDisk Secure Digital Card, Product Manual Version 1.9*. SanDisk Corporation.
- 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 กิกะ เฮิรตซ์ มีค่าฐานสิบเท่ากับ 10^9 เท่ากับ 1,000,000,000 เฮิรตซ์ ที่มา: [wikipedia](#) หน้า. 62

GiB ย่อจาก GibiByte อ่านว่า กิบิไบต์ มีค่าฐานสิบเท่ากับ 2^{30} เท่ากับ $(1024)^3$ เท่ากับ 1,073,741,824 ไบต์ ที่มา: [wikipedia](#) หน้า. 60, 135, 209, 234, 288

KiB ย่อจาก KibiByte อ่านว่า คิบิไบต์ มีค่าฐานสิบเท่ากับ 2^{10} เท่ากับ $(1024)^1$ เท่ากับ 1,024 ไบต์ ที่มา: [wikipedia](#) หน้า. 4, 94, 134, 209, 288, 367

Mbps ย่อจาก Mega bits per second หรือ เมกะบิตต่อวินาที มีค่าฐานสิบเท่ากับ 10^6 เท่ากับ 1,000,000 บิตต่อวินาที ที่มา: [wikipedia](#) หน้า. 61, 172

MiB ย่อจาก MebiByte อ่านว่า เมบิไบต์ มีค่าฐานสิบเท่ากับ 2^{20} เท่ากับ $(1024)^2$ เท่ากับ 1,048,576 ไบต์ ที่มา: [wikipedia](#) หน้า. 62, 134, 140, 224

SoC ย่อจาก System on Chip คือ การย่อส่วนระบบลงบนชิปเดียวกัน เพื่อลดขนาดโดยรวมของระบบ, หน้า. 5, 62

คอมพิวเตอร์ เครื่องจักรไฟฟ้าซึ่ง ประกอบด้วย ฮาร์ดแวร์ หรือ วงจร อิเล็กทรอนิกส์ ซึ่ง ทำงานตามสัญญาณนาฬิกา ความถี่ สูง ทำให้สามารถปฏิบัติตาม ซอฟต์แวร์ หรือ คำสั่ง ภาษา เครื่อง และ ข้อมูล ในรูปของ เลขฐานสอง ฮาร์ดแวร์ ทำหน้าที่ ประมวล ข้อมูล จาก อุปกรณ์ อินพุต ทำหน้าที่ รับ ข้อมูล ตาม ที่ คำสั่ง ภาษา เครื่อง เขียน ไว้ จน ได้ ผลลัพธ์ เพื่อ แสดงผล ทาง อุปกรณ์ เอาต์พุต, หน้า. 1

ซอฟต์แวร์ ส่วน ประกอบ สำคัญ ของ เครื่อง คอมพิวเตอร์ ประกอบ ด้วย คำสั่ง ภาษา เครื่อง และ ข้อมูล ซึ่ง ใช้ เลขฐานสอง เป็น หลัก แบ่ง เป็น ซอฟต์แวร์ ระบบ และ ซอฟต์แวร์ ประยุกต์, หน้า. 1, 60

ฮาร์ดแวร์ ส่วน ประกอบ สำคัญ ของ เครื่อง คอมพิวเตอร์ ประกอบ ด้วย วงจร อิเล็กทรอนิกส์ แบ่ง เป็น ส่วน ประมวลผล (Processor) หน่วย ความ จำ ภายใน ภาพ วงจร/อุปกรณ์ อินพุต และ เอาต์พุต และ อุปกรณ์ เก็บ รักษา ข้อมูล ทั้ง หมด นี้ ควบคุม ให้ ทำงาน ตาม ซอฟต์แวร์ หรือ คำสั่ง ภาษา เครื่อง และ ข้อมูล, หน้า. 1, 60

ดัชนี (Index)

- 2's Complement, 20, 29, 37
- A2D, 176
- Access Time, 99, 138, 140, 157, 213, 228
- Access Time, Average, 225
- Address, 14, 15, 58, 102, 156, 333
- ALU, IEEE754, 51, 136, 164
- ALU, Integer, 30, 34, 108, 136, 164
- Android, 2
- ARM, 4, 234
- ASCII, 14, 56, 93, 101, 207, 265, 318
- Assembler, 74, 308
- BCM2835, 5, 141
- BCM2836, 5, 66
- BCM2837, 6, 64, 97, 154, 178, 269
- big.LITTLE, 234
- BL: Branch and Link, 101, 129
- Bluetooth, 65, 182
- Boot Loader, 83
- Boot OS, 83, 290, 340
- Branch, 78, 113, 114
- Break Point, 103, 297, 308, 329
- BX LR, 78, 129, 131
- byte, 101, 105, 316, 317
- Cache Coherent, 204, 235
- Carry bit, 30, 109, 114
- char, 12, 13, 56, 101
- char, unsigned, 12, 101
- Chip, 4, 7, 65, 234
- CMP: Compare, 78, 113
- Command Line, 287, 327
- CPSR, 78, 108, 113, 329
- CPU Utilization, 381
- CPU: Central Processing Unit, 6, 66, 164, 224
- CSI: Camera Serial Interface, 65, 174
- D2A, 178
- DAC, 178
- Debug, 295, 305, 327
- Decode, 98
- Decode Instruction, 82, 98
- Denormalize, IEEE754, 50, 52
- Device Driver, 92, 335, 345
- Directory, 84, 209, 210, 374, 376
- Disassembly, 103
- Divide and Conquer, 242
- DMA: Direct Memory Access, 94, 199, 209, 229
- DMAC, 186, 199
- DO-WHILE, 125
- double, 12, 45, 49
- Double Precision, IEEE754, 45, 46
- doubleword, 101
- DSI: Display Serial Interface, 65, 172
- ELF, 75, 76, 86, 138, 301
- EOF, 93, 207
- Ethernet, 65, 181, 351

Execute Instruction, 82, 100, 145, 328
 Execute Permission, 210, 377, 378
 Exponent, IEEE754, 41, 46, 259
 Fetch Instruction, 82, 89, 98, 100, 146
 File System, 92, 206
 Flash Memory, 72, 92, 137, 213
 float, 12, 45, 49
 FLOPS, 235
 FOR, 121
 Fork-Join, 240
 Format Partition, 209
 Full Adder, 31, 37
 Function, 90, 128, 320
 GiB, 64, 89, 139, 167
 go-to, 114
 GPIO, 65, 85, 186, 189, 203, 355, 365
 GPU, 6, 66, 341
 halfword, 316
 HDMI, 65, 339
 Heap Segment, 90
 hword, 101, 316
 I2S, 177
 IEEE754, 14, 45, 49, 99
 IF, 115
 IF-ELSE, 118
 Immediate, 81, 106
 Inode, 209, 374
 int, 12, 13, 101
 int, unsigned, 101
 Interrupt, 70, 167, 194, 229
 Interrupt Priority, 196
 Interrupt Request, 194, 196
 IoT, 2, 58, 64, 165, 181, 339
 ISR: Interrupt Service Routine, 194, 365, 367
 Java Bytecode, 133
 Jump, 114
 Kernel Space, 88, 89, 140, 141
 Label, 77, 113
 Latency, 162, 194, 228
 LCD, 71, 168, 188
 Library, 76, 79, 85, 359
 Linker, 74, 79, 298, 359
 Load/Store, 91, 99, 104
 Load/Store Architecture, 91, 99, 133
 long long, 12
 long long, unsigned, 12, 101
 Loop, 32, 38, 121, 123, 125, 301, 302
 LR: Link Register, 101, 320, 332
 Machine Code, 74, 80, 86, 103
 main, 77, 89, 130
 main(), 77, 131, 243, 313
 Makefile, 75, 299, 309, 315, 328, 368
 makefile, 75, 299, 309, 315, 328, 368
 Many-Core CPU, 231
 Memory Mapped File, 93, 206, 229
 Memory Mapped I/O, 185
 MergeSort, 242
 Metadata, 212
 Moore's Law, 231
 Mount, 85, 376
 Multi-Core CPU, 231
 Multicomputer, 233
 Multitasking, 139
 NaN, IEEE754, 49, 50
 Negative bit, 30, 109, 114
 Non-Volatile Memory, 137
 Normalize, IEEE754, 12, 41
 NZCV, 108, 113, 114

- Object File, 80, 86, 308, 309
Opcode, 81, 103
OpenMP, 381
Overflow, IEEE754, 52, 53
Overflow, Integer, 15, 30, 109
Parallel Computing, 165
Parallel Region, 240
Parallelism, 237
Parallelism, Data, 237
Parallelism, Task, 237
Parameter, 101
Partition, 208
PC: Program Counter, 100, 113, 143, 320
PCM, 176
Permission, 208, 210, 379
Physical Address, 138, 139
Physical Memory, 137, 164
Pipeline, 51, 97–99
Pointer, 15, 79
Pop, 90, 105, 332
Process, 138
Push, 105, 134, 320, 332
QuickSort, 242
Refresh, 163
Register, 164
RISC, 133
SATA, 226, 229
Scalar, 99
Scan Code, 69
SDRAM, 68
Segment, Heap, 90
Segment, Stack, 90
Segment, Text, 102
short, 12, 14, 24
Shut Down, 83, 94
Sign Bit, IEEE754, 41, 259
Sign Bit, Integer, 20, 28, 39, 41, 46
Sign-Magnitude, 28
Signed Integer, 19, 28
Significand, IEEE754, 41, 259
SIMD, 133
Single Precision, 45, 46, 48, 50, 258, 261
Smartphone, 4
SMP: Shared Memory Multiprocessor, 233
SoC, 7, 65, 66
Socket Interface, 92
Software, 164
Source Code, 88
SP: Stack Pointer, 90, 101, 105, 333
Speedup, 381
Stack Machine, 133
Stack Pointer, 90, 320
Stack Segment, 90, 105
Storage, 165
String, 56, 318
string, 56, 318
Super User, 94
SuperScalar, 231
Swap Partition, 139
System Call Interface, 88
System Software, 73
Technology Scaling, 232
Text Segment, 102
Thumb, 132
Time, Access, 157
TLB: Translation Lookaside Buffer, 142, 186
Top of Stack, 332
UCS-2, 58

- Underflow, IEEE754, 52, 53
 Unicode, 57, 265
 unsigned char, 12, 101
 unsigned int, 12, 101
 Unsigned Integer, 15, 29, 32, 33, 301
 unsigned long, 14
 unsigned short, 12, 17
 UPS, 137
 USB, 64, 65, 69, 168, 179, 186, 347
 User Space, 88, 89
 UTF-16, 58
 UTF-8, 58
 Variable, 164
 Variable, Global, 104, 313
 Variable, Local, 313, 332
 Vector, 59, 99
 Virtual Address, 100, 138, 145
 Virtual File System, 377
 Volatile Memory, 137
 WHILE, 123
 WiFi, 65, 182, 351
 wiringPi, 355, 365
 word, 101, 105, 316
 Worker Thread, 381
 Zero, bit, 30, 109
 Zero, IEEE754, 50, 260
 การฟอร์แมท, 209
 กิบิไบต์, 64, 89, 139, 167
 ความขนาดของข้อมูล, 237
 ความขนาดของงานย่อย, 237
 ความถี่สูง, 176
 คอมมานด์ไลน์, 287, 327
 คอมเพเลอร์, 74, 295
 คำนวณแบบบานาน, 165
 ค่านัยสำคัญ, IEEE754, 41
 ค่ายกกำลัง, IEEE754, 41, 46
 ชัตดาวน์, 94, 205
 ชิป, 4, 7, 65, 234
 ซอฟต์แวร์, 73
 ซอฟต์แวร์ประยุกต์, 63, 64
 ซอฟต์แวร์ระบบ, 73
 ซีพียูมัลติคอร์, 231, 233
 ซีพียูแมเนเน็คคอร์, 231, 233
 ชูเปอร์สเกลาร์, 231
 ดาต้าเซ็กเมนต์, 76, 87, 313
 ตินอร์มัลไลซ์, IEEE754, 50
 ดีไวซ์ไดเรเวอร์, 92, 209, 350
 ตัวแปร, 13, 91, 99–101, 104, 164
 ถอดรหัส, 82, 98
 ทรานซิสเตอร์, 7, 8, 154, 161
 นอร์มัลไลซ์, IEEE754, 12, 41, 52, 55
 บิตเครื่องหมาย, IEEE754, 41
 บูตโหลดเดอร์, 83
 พารามิเตอร์, 101
 พาร์ทิชัน, 208
 พังก์ชัน, 74, 80, 90, 128
 ภาษาเครื่อง, 73, 74, 76, 80, 86, 102, 103
 มัลติคอมพิวเตอร์, 233, 235
 มัลติทาสกิ้ง, 139
 มัลติเฟรด, 231, 234, 239
 ระบบไฟล์, 92, 206, 208
 ระบบไฟล์เสมือน, 85
 รีจิสเตอร์, 31, 82, 91, 99, 100, 164
 ล็อกิกเกต, 7, 8
 ลิงก์เกอร์, 74, 298
 วงจรรวม, 4, 7, 234
 สถาปัตยกรรมโหลด/สโตร์, 91, 99, 133
 สมาร์ตโฟน, 3, 4, 142
 สวอปพาร์ทิชัน, 139

- สเกลาร์, 99
 สแกนโค้ด, 69
 สแต็ก, 101, 105, 320, 332, 333
 สแต็กเซ็กเมนต์, 90, 105, 333
 สแต็กเฟรม, 90, 332
 สแต็กแมชชีน, 133
 หน่วยความจำภายในภาพ, 137, 140, 164, 209
 หน่วยความจำหลัก, 137, 140
 ออปโค้ด, 81, 103
 อันเดอร์โฟล์ว, IEEE754, 52, 54
 อาร์เรย์, 56, 79, 105, 147, 316, 317
 อินพินิตี้, IEEE754, 49, 50
 อินเทอร์รัปต์, 70, 167, 194, 229
 อุปกรณ์เก็บรักษาข้อมูล, 137, 159, 165
 อีบจีบีจี, 80, 299, 309
 ไฮป์เซ็กเมนต์, 90
 เครื่องข่ายบันซิป, 236
 เคอร์เนล, 84, 88, 92
 เซ็กเมนต์, ดาต้า, 87, 313
 เซ็กเมนต์, สแต็ก, 90, 105
 เทกซ์เซ็กเมนต์, 76, 87
 เพทซ์คำสั่ง, 89, 98, 100, 146
 เลเบล, 77, 106, 107, 113, 313, 330
 เวกเตอร์, 99
 เวลารอคอย, 194
 เวลาเข้าถึง, 99, 138, 140, 154, 157, 213, 214
 เวลาเข้าถึงเฉลี่ย, 225
 เวอร์ชวลเม莫รี่, 135, 153, 206
 เวอร์ชวลแอดเดรส, 100, 138, 145, 186
 แท็บเล็ต, 3
 แอดเดรส, 14, 15, 58, 102, 156, 333
 แอดเดรสภายในภาพ, 138, 140, 186
 แอสเซมเบลอร์, 74, 308
 โปรแกรม เคน์เตอร์, 100, 113, 143, 308, 320,