

องค์ประกอบของเครื่องคอมพิวเตอร์และภาษาแอลซีเมบลี

ARM

Computer Organization and ARM Assembly

Language

ผศ.ดร.สุรินทร์ กิตติธรกุล

ผศ.ดร.ชัยวัฒน์ หนูทอง

2 สิงหาคม 2561 *

*ภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง First drafted on 2 สิงหาคม 2559

Contents

Contents	i
List of Tables	ix
List of Figures	xi
1 บทนำ (Introduction)	1
1.1 ชนิดของเครื่องคอมพิวเตอร์	1
1.1.1 คอมพิวเตอร์ตั้งโต๊ะ (Desktop computers)	1
1.1.2 คอมพิวเตอร์เซิร์ฟเวอร์หรือแม่ข่าย (Server computers)	1
1.1.3 คอมพิวเตอร์พกพา (Portable Computers)	2
1.1.4 คอมพิวเตอร์ฝังตัว (Embedded computers)	2
1.2 แนวโน้มของจำนวนอุปกรณ์คอมพิวเตอร์ชนิดต่างๆ	2
1.3 ขั้นตอนการผลิตไมโครชิป	3
1.4 บอร์ด Raspberry Pi และชิป Broadcom BCM 2835	4
1.5 สรุปท้ายบท	5
1.6 คำถามท้ายบท	5
2 ข้อมูลและคณิตศาสตร์ในคอมพิวเตอร์ (Computer Data and Arithmetic)	7
2.1 ตัวแปรชนิดต่างในภาษา C/C++	8
2.2 เลขจำนวนเต็มฐานสอง	10
2.2.1 ชนิดไม่มีเครื่องหมาย (Unsigned Integer)	10
2.2.1.1 การแปลงเลขฐานสองเป็นฐานสิบ	11
2.2.1.2 การแปลงเลขฐานสิบเป็นฐานสอง	12
2.2.2 ชนิดมีเครื่องหมาย (Signed Integer) แบบ 2-Complement	13
2.2.2.1 การแปลงเลขฐานสองเป็นฐานสิบ	14
2.2.2.2 การแปลงเลขฐานสิบเป็นฐานสอง	17
2.2.3 ชนิดมีเครื่องหมาย (Signed Integer) แบบ Sign-Magnitude	18
2.3 คณิตศาสตร์เลขจำนวนเต็ม	20
2.3.1 ชนิดไม่มีเครื่องหมาย	20

2.3.1.1 การบวกเลขจำนวนเต็มชนิดไม่มีเครื่องหมายและการตรวจจับโอเวอร์เฟล์ว	21
2.3.1.2 การคูณเลขจำนวนเต็มชนิดไม่มีเครื่องหมาย	23
2.3.2 ชนิดมีเครื่องหมายแบบ 2-Complement	26
2.3.2.1 การบวก/ลบเลขจำนวนเต็มชนิดมีเครื่องหมายและการตรวจจับโอเวอร์เฟล์	26
2.4 เลขทศนิยมฐานสองชนิดจุดคงที่ (Binary Fixed Point)	28
2.5 เลขทศนิยมฐานสองชนิดจุดลอยตัว (Binary Floating Point)	30
2.5.1 การบวกเลขทศนิยมฐานสองชนิดจุดลอยตัว	31
2.5.2 การคูณเลขทศนิยมฐานสองชนิดจุดลอยตัว	33
2.6 เลขทศนิยมฐานสองชนิดจุดลอยตัวมาตรฐาน IEEE 754	34
2.6.1 รูปแบบของเลข IEEE 754	34
2.6.2 ค่าสูงสุด ต่ำสุดและค่าอื่นๆ ของเลข Floating Point	38
2.6.3 การบวกเลขทศนิยมชนิดจุดลอยตัวตามมาตรฐาน IEEE 754	39
2.6.4 การคูณเลขทศนิยมฐานสองชนิดจุดลอยตัวตามมาตรฐาน IEEE 754	43
2.7 ตัวอักษร (Character)	45
2.8 สรุปท้ายบท	47
2.9 คำถ้ามท้ายบท	48
3 ฮาร์ดแวร์และซอฟต์แวร์ของคอมพิวเตอร์ (Computer Hardware & Software)	51
3.1 ฮาร์ดแวร์ของเครื่องคอมพิวเตอร์	52
3.1.1 ชีพียู (CPU: ARM Cortex A53)	53
3.1.2 หน่วยความจำหลัก (1GB DDR2 SDRAM)	55
3.1.3 อุปกรณ์อินพุท/เอาท์พุท (Input/Output Devices)	56
3.1.3.1 คีย์บอร์ด (Keyboard)	56
3.1.3.2 เม้าส์ (Mouse)	57
3.1.3.3 จอภาพ LCD (Liquid Crystal Display)	59
3.1.4 อุปกรณ์เก็บรักษาข้อมูล (Data Storage)	60
3.2 ซอฟต์แวร์หรือโปรแกรมคอมพิวเตอร์	61
3.2.1 การบูทรับบปฎิบัติการจากอุปกรณ์เก็บรักษาข้อมูลเข้าสู่หน่วยความจำหลัก	62
3.2.2 การโหลดซอฟต์แวร์ประยุกต์จากไฟล์ ELF เข้าสู่หน่วยความจำหลัก	65
3.2.3 การอ่านคำสั่งของซอฟต์แวร์ประยุกต์จากหน่วยความจำหลักเพื่อไปปฏิบัติตาม	66
3.2.4 ซอฟต์แวร์ประยุกต์อ่าน/เขียนข้อมูลในหน่วยความจำเพื่อให้ชีพียูประมวลผล	68
3.2.5 การใช้งานอินพุทและเอาท์พุทต่างๆ	68
3.2.6 การอ่าน/เขียนข้อมูลระหว่างหน่วยความจำหลักและอุปกรณ์เก็บรักษาข้อมูล	69
3.2.7 การซัพดาวน์ (Shut Down) ระบบปฏิบัติการก่อนหยุดจ่ายไฟ	71
3.3 การพัฒนาซอฟต์แวร์หรือโปรแกรมคอมพิวเตอร์	72

3.3.1	โครงสร้างของชอร์สโค้ดโปรแกรมภาษา C/C++	72
3.3.2	การคอมไพล์ชอร์สโค้ดให้กลายเป็นโปรแกรมคอมพิวเตอร์	72
3.3.3	โครงสร้างของชอร์สโค้ดโปรแกรมภาษาแอสเซมบลีของ ARM	74
3.3.4	การเขื่อมไฟล์อ้อมเบ็คท์ด้วยลิงค์เกอร์ (Linker)	77
3.3.5	ตัวอย่างของคำสั่งภาษาเครื่อง (Machine Code)	78
3.4	สรุปท้ายบท	79
3.5	คำถ้ามท้ายบท	80
4	ภาษาแอสเซมบลีของ ARM เวอร์ชัน 32 บิต	81
4.1	โครงสร้างของชีพិយ ARM Cortex A53 ใน BCM2837	81
4.2	สถาปัตยกรรมชุดคำสั่ง (Instruction Set Architecture)	84
4.3	ตัวอย่างคำสั่งภาษาเครื่องในหน่วยความจำ	85
4.4	การประมวลและตั้งค่าตัวแปรในหน่วยความจำหลัก	87
4.5	คำสั่งถ่ายโอนข้อมูลระหว่างหน่วยความจำและรีจิสเตรอร์	88
4.6	คำสั่งประมวลผลข้อมูลในรีจิสเตรอร์ (Register Data Processing Instructions)	91
4.6.1	คำสั่งทางคณิตศาสตร์	91
4.6.2	คำสั่งเลื่อนบิทข้อมูล	92
4.6.3	คำสั่งทางคณิตศาสตร์และเลื่อนบิทพร้อมกัน	93
4.6.4	คำสั่งทางตรรกศาสตร์	93
4.7	คำสั่งควบคุมการทำงาน (Control Instructions)	94
4.7.1	การตัดสินใจ IF	96
4.7.2	การตัดสินใจ IF-ELSE	97
4.7.3	การวนรอบชนิด FOR	100
4.7.4	การวนรอบชนิด WHILE	102
4.7.5	การวนรอบชนิด DO-WHILE	103
4.7.6	การวนรอบชนิด FOR จำนวน 2 ชั้น	104
4.8	การเรียกใช้ฟังก์ชัน (Function Call)	106
4.8.1	การเรียกใช้ฟังก์ชันในภาษา C/C++	106
4.8.2	การเรียกใช้ฟังก์ชันในภาษาแอสเซมบลี	107
4.9	อุปกรณ์และวิวัฒนาการของชุดคำสั่ง ARM	108
4.9.1	อุปกรณ์ดิจิทัลที่ใช้ชีพិយ ARM	108
4.9.2	วิวัฒนาการของชุดคำสั่ง ARM	109
4.10	สรุปท้ายบท	110
4.11	คำถ้ามท้ายบท	110
5	หน่วยความจำลำดับชั้น (Memory Hierarchy)	113
5.1	โครงสร้างของลำดับชั้นหน่วยความจำของบอร์ด Pi3	114

Contents

5.2	หน่วยความจำเสมือน (Virtual Memory)	116
5.2.1	หน่วยความจำเสมือนของ Raspberry Pi3	117
5.2.2	หน่วยความจำเสมือนชนิดเพจของ ARM Cortex A7	119
5.3	หลักการพื้นฐานของหน่วยความจำแคช (Cache)	121
5.3.1	แคชชนิดไดเรคท์แมป (Direct Map Cache)	122
5.3.2	แคชชนิดเซ็ตแอนโซไซอีฟ (Set Associative Cache)	124
5.4	หน่วยความจำชนิดสแตติกแรม (Static RAM: SRAM)	126
5.4.1	โครงสร้างภายในของชิป SRAM	127
5.4.2	การทำงานของสแตติกแรม: อ่านและเขียน	128
5.5	หน่วยความจำหลักชนิดไดนามิกแรม (Dynamic RAM: DRAM)	130
5.5.1	โครงสร้างภายในของชิป DRAM	132
5.5.2	การทำงานของไดนามิกแรม: อ่านและเขียน	134
5.5.3	การรีเฟรชข้อมูล	135
5.6	สรุปท้ายบท	137
5.7	คำถາมท้ายบท	139
6	อุปกรณ์/วงจรอินพุตและเอาท์พุต (Input and Output Devices/Circuit)	141
6.1	สัญญาณ HDMI สำหรับจอภาพ LCD ขนาดใหญ่	143
6.2	สัญญาณ DSI สำหรับจอภาพ LCD ขนาดเล็ก	146
6.3	สัญญาณ CSI สำหรับเชื่อมต่อกล้องขนาดเล็ก	149
6.4	สัญญาณ PCM สำหรับสัญญาณเสียง	151
6.5	สัญญาณภาพและเสียงสำหรับจอทีวี	153
6.6	สัญญาณ USB 2.0 สำหรับอุปกรณ์ต่อพ่วงต่างๆ	154
6.7	สัญญาณ Ethernet สำหรับสายเชื่อมต่อกับเครือข่ายอินเทอร์เน็ต	156
6.8	สัญญาณ WiFi และ Bluetooth สำหรับการสื่อสารไร้สาย	157
6.9	หลักการ Memory Mapped Input/Output	160
6.10	หัวเชื่อมต่อ 40 ขา (40-Pin Header)	162
6.11	ขา GPIO (General Purpose Input Output)	164
6.12	การขัดจังหวะ (Interrupt)	169
6.13	การเข้าถึงหน่วยความจำโดยตรง (Direct Memory Access)	172
6.14	แหล่งจ่ายไฟ (Power Supply) ของบอร์ด Pi3	174
6.15	สรุปท้ายบท	175
6.16	คำถາมท้ายบท	175
7	อุปกรณ์เก็บรักษาข้อมูล (Data Storage Devices)	177
7.1	ระบบไฟล์ (File System)	178
7.1.1	การใช้งานไฟล์ (File Operations)	178

7.1.2 การแบ่งพาร์ติชัน (Partition)	179
7.1.3 โครงสร้างของระบบไฟล์ Unix (Structure of Unix File System)	180
7.1.4 ไอโนนด์ (Inode) คืออะไร	181
7.2 ชิปหน่วยความจำแฟลช (Flash Memory Chip)	184
7.2.1 โครงสร้างภายในชิปหน่วยความจำแฟลชนิด NAND	185
7.2.2 การอ่านข้อมูล	187
7.2.3 หน่วยความจำแฟลชนิดอื่นๆ	188
7.3 การ์ดหน่วยความจำ SD (Secure Digital)	190
7.3.1 การอ่านข้อมูลจากการ์ด	191
7.3.2 การเขียนข้อมูลจากการ์ด	192
7.4 โซลิเดสเตทไดสก์ (Solid-State Disk: SSD)	193
7.5 ฮาร์ดดิสก์ไดร์ (Hard Disk Drive: HDD)	195
7.5.1 โครงสร้างของฮาร์ดดิสก์เชิงกายภาพ	195
7.5.2 การจัดเรียงข้อมูลในฮาร์ดดิสก์	196
7.5.3 ความจุและประสิทธิภาพของฮาร์ดดิสก์	197
7.6 สรุปท้ายบท	198
7.7 คำถามท้ายบท	198
 A การทดลองที่ 1 ข้อมูลและคณิตศาสตร์ในคอมพิวเตอร์	201
A.1 การแปลงและคณิตศาสตร์สำหรับเลขฐานสองจำนวนเต็ม	202
A.1.1 การทดลอง	202
A.1.2 กิจกรรมท้ายการทดลอง	205
A.2 การแปลงและคณิตศาสตร์สำหรับ Floating-Point IEEE754	207
A.2.1 การทดลองสำหรับ Single-Precision	207
A.2.2 การทดลองสำหรับ Double-Precision	210
A.2.3 กิจกรรมท้ายการทดลอง	212
A.3 รหัสของข้อมูลตัวอักษร	213
A.3.1 การทดลอง	213
A.3.2 กิจกรรมท้ายการทดลอง	214
 B การทดลองที่ 2 การประกอบและติดตั้งบอร์ด Pi3	215
B.1 รายการอุปกรณ์ฮาร์ดแวร์	216
B.2 ประกอบบอร์ด Pi3 และกล่อง	217
B.2.0.1 ประกอบฮีทซิงค์ (Heat Sink)	217
B.2.0.2 ประกอบกล่องใส่บอร์ด Pi 3	217
B.3 เครื่องคอมพิวเตอร์ส่วนบุคคลจากบอร์ด Pi 3 โมเดล B	219
B.4 กิจกรรมท้ายการทดลอง	221

C การทดลองที่ 3 การติดตั้งระบบปฏิบัติการ Raspbian	223
C.1 การเตรียมการ์ดหน่วยความจำ MicroSD	223
C.2 การติดตั้ง NOOBS (โนอบส์) บนการ์ด microSD	225
C.3 การติดตั้งระบบปฏิบัติการ Raspbian	227
C.4 การตั้งค่าบอร์ด Pi3 เพื่อใช้งาน	229
C.4.1 การตั้งชื่อและพาสเวิร์ด	229
C.4.2 การตั้งค่า Wi-Fi เพื่อเชื่อมต่อกับอินเตอร์เน็ท	229
C.4.3 การรีสตาร์ทและซัฟดาวน์	231
C.4.4 กิจกรรมท้ายการทดลอง	231
D การทดลองที่ 4 การใช้งานระบบปฏิบัติการยูนิกซ์เบื้องต้น	233
D.1 การใช้งาน Unix ผ่านทาง GUI	233
D.1.1 หน้าจอหลัก (Desktop)	233
D.1.2 ไฟล์เมเนจเจอร์ (File Manager)	234
D.1.3 การซัฟดาวน์ (Shutdown)	234
D.2 การใช้งาน Unix ผ่านทางคีย์บอร์ด	235
D.2.1 คำสั่งพื้นฐานของระบบ Unix	235
D.2.2 การซัฟดาวน์ (Shutdown)	236
D.3 ข้อมูลพื้นฐานของบอร์ด Pi3	237
D.3.1 ข้อมูลของซีพียู	237
D.3.2 ข้อมูลขั้นสูงของซีพียูและบอร์ด	238
D.4 กิจกรรมท้ายการทดลอง	238
E การทดลองที่ 5 การพัฒนาโปรแกรมด้วยภาษา C	241
E.1 การพัฒนาโดยใช้ IDE	241
E.2 การดีบัก (Debugging) โดยใช้ IDE	245
E.3 การพัฒนาโดยใช้ประโยชน์คำสั่งทีลัชั่นตอน	245
E.4 โครงสร้างของ Makefile	246
E.5 การพัฒนาโดยใช้ Makefile	246
E.6 กิจกรรมท้ายการทดลอง	248
F การทดลองที่ 6 การพัฒนาโปรแกรมภาษาแอลกอริทึม	249
F.1 การพัฒนาโดยใช้ IDE	249
F.2 การดีบักโปรแกรมโดยใช้ IDE	252
F.3 การพัฒนาโดยใช้ประโยชน์คำสั่งทีลัชั่นตอน	252
F.4 การพัฒนาโดยใช้ Makefile	253
F.5 กิจกรรมท้ายการทดลอง	254

G การทดลองที่ 7 การสร้างฟังค์ชันในโปรแกรมภาษาแอลซีเมบลี	257
G.1 การใช้งานตัวแปรใน Data Segment	257
G.1.1 การโหลดค่าตัวแปรจากหน่วยความจำ	257
G.1.2 การใช้งานตัวแปรอะเรย์ชนิดต่างๆ ใน Data Segment	260
G.1.3 การใช้งานตัวแปรอะเรย์ชนิด Byte	261
G.2 การเรียกใช้ฟังค์ชันสำเร็จรูปและตัวแปรชนิดประโยค	261
G.3 การสร้างฟังค์ชันด้วยภาษาแอลซีเมบลี	263
G.4 กิจกรรมท้ายการทดลอง	267
H การทดลองที่ 8 การพัฒนาโปรแกรมภาษาแอลซีเมบลีขั้นสูง	269
H.1 ตีบักเกอร์ GDB	269
H.2 การใช้งานสแต็คพอยท์เตอร์ (Stack Pointer)	274
H.3 การพัฒนาโปรแกรมภาษาแอลซีเมบลีร่วมกับภาษา C	276
H.4 กิจกรรมท้ายการทดลอง	277
I การทดลองที่ 9 การศึกษาและปรับแก้寅พุทธและเอาท์พุทธต่างๆ	279
I.1 จอแสดงผลผ่านพอร์ท HDMI	279
I.1.1 การปรับแก้ขนาดหน่วยความจำของ GPU	279
I.1.2 การปรับแก้ความละเอียดของจอแสดงผล	280
I.2 ระบบเสียง	282
I.2.1 รายชื่ออุปกรณ์ด้านระบบเสียง	282
I.2.2 การควบคุมระดับเสียง	283
I.3 พورทเชื่อมต่ออุปกรณ์ USB	284
I.3.1 รายชื่ออุปกรณ์กับพอร์ท USB	284
I.3.2 รายละเอียดการเชื่อมต่ออุปกรณ์กับพอร์ท USB	285
I.4 พอร์ทเชื่อมต่อเครือข่าย WiFi และ Ethernet	288
I.4.1 รายชื่ออุปกรณ์เครือข่าย	288
I.4.2 การเปิดปิดอุปกรณ์เครือข่าย	289
I.4.3 การตรวจสอบการเชื่อมต่อกับเครือข่ายเบื้องต้น	290
I.5 กิจกรรมท้ายการทดลอง	291
J การทดลองที่ 10 การเชื่อมต่อกับ GPIO	293
J.1 ไลบรารี wiringPi	293
J.2 วงจรไฟ LED กระพริบ	296
J.3 โปรแกรมไฟ LED กระพริบภาษา C	297
J.4 โปรแกรมไฟ LED กระพริบภาษา Assembly	298
J.5 กิจกรรมท้ายการทดลอง	300

Contents

K การทดลองที่ 11 การเชื่อมต่อกับอินเทอร์รัพท์	303
K.1 อินเทอร์รัพท์	303
K.2 การจัดการอินเตอร์รัพท์ (Interrupt Handling)	304
K.2.1 การจัดการอินเทอร์รัพท์ของ WiringPi	304
K.2.2 วงจรปุ่มกด Push Button เชื่อมผ่านขา GPIO	304
K.2.3 โปรแกรมภาษา C สำหรับทดสอบวงจรอินเทอร์รัพท์	306
K.3 กิจกรรมท้ายการทดลอง	307
L การทดลองที่ 12 การศึกษาอุปกรณ์เก็บรักษาข้อมูลและระบบไฟล์	309
L.1 ขนาดของไฟล์และไดเรคทอรี	309
L.2 ระบบไฟล์	311
L.3 อุปกรณ์อินพุตและเอาท์พุตในระบบไฟล์	314
L.4 กิจกรรมท้ายการทดลอง	316
Bibliography	317
Index	319

List of Tables

2.1	ชนิดตัวแปร ความยาว (บิต) ค่าฐานสิบต่ำสุดและสูงสุดของตัวแปรแต่ละชนิดในภาษา C/C++	8
2.2	การแปลงค่าฐานสิบเป็นเลขฐานสองแบบไม่มีเครื่องหมายความยาว $n=8$ บิต ด้วยการหาร	12
2.3	การแปลงค่าฐานสิบเป็นเลขฐานสองแบบไม่มีเครื่องหมายความยาว $n=8$ บิต ด้วยการลบ	13
2.4	รูปแบบ (Pattern) ของเลขฐานสองขนาด $n=4$ บิตทั้งหมด $2^4=16$ แบบและการตีความหมายให้เป็นค่าฐานสิบแบบมีเครื่องหมาย 2-Complement และแบบไม่มีเครื่องหมาย	15
2.5	รูปแบบ (Pattern) ของเลขฐานสองขนาด $n=5$ บิตทั้งหมด $2^5=32$ แบบและการตีความหมายให้เป็นค่าฐานสิบแบบมีเครื่องหมาย 2-Complement และแบบไม่มีเครื่องหมาย	16
2.6	รูปแบบ (Pattern) ของเลขฐานสองขนาด $n=8$ บิตทั้งหมด $2^8=256$ แบบและการตีความหมายให้เป็นค่าฐานสิบแบบมีเครื่องหมาย 2-Complement และแบบไม่มีเครื่องหมาย	17
2.7	การแปลงค่าฐานสิบแบบมีเครื่องหมายให้เป็นเลขฐานสองความยาว $n=4$ บิต โดยแปลงให้เป็น ค่าฐานสิบโดยใช้สูตร $2^n + X_{10,s}$ กรณีที่ $X_{10,s} < 0$ ตามสมการที่ 1.32	18
2.8	ค่าฐานสิบแบบมีเครื่องหมาย Sign-Magnitude, แบบ 2-Complement, และแบบไม่มีเครื่องหมาย (Unsigned) ของเลขฐานสองความยาว $n=4$ บิตทั้งหมด $2^4=16$ แบบ	19
2.9	ผลคูณเลขขนาด 4 บิต ตัวตั้ง= 1010_2 (10_{10}) ตัวคูณ= 1101_2 (13_{10}) เท่ากับ 130_{10} ด้วยวิจารณรูปที่ 1.5 หมายเหตุ Shift R คือ การเลื่อนบิตไปทางขวา	24
2.10	ผลคูณเลขขนาด 4 บิต ตัวตั้ง= 1010_2 (10_{10}) ตัวคูณ= 1101_2 (13_{10}) เท่ากับ 130_{10} ด้วยวิจารณรูปที่ 1.6 หมายเหตุ Shift R คือ การเลื่อนบิตไปทางขวา	25
2.11	ตารางสรุปความแตกต่างระหว่างเลขศนย์ฐานสองชนิดจุดลอยตัวตามมาตรฐาน IEEE 754 ชนิด Single Precision โดย E_2 คือ ค่ายกกำลัง และ Y_2 คือ เทคนิยซึ่งเป็นส่วนประกอบของค่านัยสำคัญ (x หมายถึง บิตข้อมูลมีค่าเท่ากับ '0' หรือ '1')	38
2.12	ชนิด ความยาว ข้อมูล และการประยุกต์ใช้งานเลขฐานสองชนิดต่างๆ ในคอมพิวเตอร์	47
3.1	ตารางสรุปข้อมูลด้านอาร์ดแวร์และซอฟต์แวร์ของบอร์ด Pi3 โนแมล B และลิงค์เชื่อมไปยังหัวข้อที่แสดงรายละเอียดในบทต่างๆ	54
3.2	ระดับการรัน (Run Level), โหมดการทำงานของระบบ และชื่อไฟล์เดอร์ที่บรรจุไฟล์โปรแกรมและไฟล์ข้อมูลสำหรับการบูทบอร์ด Pi3	63
3.3	ตัวอย่างโปรแกรมภาษาแอสเซมบลีเพื่อประกาศและตั้งค่าเริ่มให้ตัวแปรขนาด 32 บิตจำนวน 4 ตัวแปร	75

4.1 ตัวอย่างโปรแกรมภาษาแオスเซมบลีเพื่อสร้างตัวแปรขนาด 32 บิต จำนวน 2 ตัวแปร ในพื้นที่ของเซกเมนท์ Data	87
4.2 ตัวอย่างโปรแกรมภาษาแオスเซมบลีเพื่ออ่านค่าตัวแปรจากหน่วยความจำ โดยการอ่านตำแหน่งของตัวแปร	89
4.3 ตัวอย่างโปรแกรมภาษาแオスเซมบลีเพื่อกำหนดค่า $x = (a + b) - c$	90
4.4 ตัวอย่างโปรแกรมตามประโยคเงื่อนไข if	97
4.5 ตัวอย่างโปรแกรมตามประโยคเงื่อนไข if-else	99
4.6 ตัวอย่างโปรแกรมตามประโยควนรอบ For	101
4.7 ตัวอย่างโปรแกรมตามประโยควนรอบ While	103
4.8 ตัวอย่างโปรแกรมตามประโยควนรอบ Do-While	104
4.9 ตัวอย่างโปรแกรมเรียกใช้ฟังก์ชันด้วยคำสั่ง BL และ BX	107
4.10 ส่วนแบ่งการตลาดของบริษัท ARM ในปี ค.ศ. 2010 ที่มา: zdnet.com	109
 5.1 ตัวอย่างโปรแกรมภาษาแオスเซมบลีเพื่อประกอบการทำงานของแอดเดรส PA: Physical Address)	122
6.1 หมายเลขขา ซีอี และวัตถุประสงค์ของคอนเนคเตอร์ชนิด HDMI เวอร์ชัน 1.4 ที่มา: wikipedia	144
6.2 หมายเลข และหน้าที่ของสายสัญญาณ DSI สำหรับจอภาพ LCD ขนาดเล็กด้วยข้อมูลจำนวน 2 เลน ที่มา: wikipedia	147
6.3 หมายเลข ซีอี และวัตถุประสงค์ของคอนเนคเตอร์ชนิด CSI wikipedia	150
6.4 ตารางแอ็ดเดรสบัสเริ่มต้นที่หมายเลข 0x7E00_0000 สำหรับอุปกรณ์อินพุตเอาท์พุต (IO Peripherals) และหัวข้อ ที่มา: Broadcom (2012)	161
6.5 หมายเลข ซีอี ตัวเลือกที่ 0-5 ของหัวเชื่อมต่อสายทั้ง 40 ขา (GND: Ground) ที่มา: Broadcom (2012)	163
6.6 ตารางแอ็ดเดรสในหน่วยความจำเริ่มต้นที่หมายเลข 0x7E20_0000 สำหรับ GPIO ที่มา: Broadcom (2012)	166
6.7 ตารางเลือกค่า V_{OUT} และค่าตัวเหนี่ยวนำ L_1 และ L_2 ที่มา: Diodes (2012)	175
 7.1 ตารางเปรียบเทียบเซลหน่วยความจำ Flash ชนิด NAND (ซ้าย) และ NOR (ขวา) ตามโครงสร้างและเลี้ยงเอาท์ของทรานซิสเตอร์ ที่มา: Choi (2010) หมายเหตุ F คือ ความกว้างของเฟลตเกต (Float Gate) มีหน่วยเป็น นาโนเมตร	188
7.2 หมายเลข ซีอี และวัตถุประสงค์ของ SD ใน荷ะดการทำงาน荷ะด SD 4 บิต ที่มา: wikipedia	191
7.3 การเปรียบเทียบประสิทธิภาพของอุปกรณ์เก็บรักษาข้อมูลชนิดต่างๆ	199

List of Figures

1.1	ตัวอย่างเครื่องคอมพิวเตอร์ชนิดเซิร์ฟเวอร์ จากบริษัท Hewlett Packard รุ่น Proliant ที่มา: datacenterknowledge.com	2
1.2	ปริมาณยอดขายเครื่องคอมพิวเตอร์ชนิดเดสค์ทอป ในตบุค สมาร์ทโฟน และแท็บเล็ท ทั่วโลกในปี ค.ศ. 2005-2015 ที่มา: slideshare.net	3
1.3	ขั้นตอนการผลิตของชิป (Chip) วงจรดิจิทัลไมโครโปรเซสเซอร์และอื่นๆ ที่มา: slideplayer.com	4
1.4	ภาพถ่ายซิลิกอนดาย (Silicon Die) ของชิป BCM 2835 บนบอร์ด Raspberry Pi ที่มา: raspberrypi.org	5
2.1	เลขจำนวนเต็มและเลขจำนวนจริงบนเส้นจำนวนในคณิตศาสตร์เลขฐานสิบ ที่มา: tutrvista.com	8
2.2	พื้นที่ในหน่วยความจำซึ่งบรรจุค่าเริ่มต้นของตัวแปรที่ได้ประกาศก่อนหน้า ที่มา: Harris and Harris (2013)	9
2.3	เส้นจำนวนเปรียบเทียบข้อมูลขนาด 8 บิตในรูปแบบของเลขจำนวนเต็มฐานสองชนิดไม่มีเครื่องหมาย (Unsigned) และมีเครื่องหมายแบบ 2-Complement ที่มา: stackoverflow.com	10
2.4	สัญลักษณ์ของ ALU (Arithmetic Logic Unit) สำหรับบวก/ลบเลขจำนวนเต็มขนาด n บิตชนิดไม่มีเครื่องหมาย ที่มา: teach-ict.com	20
2.5	วงจorcูณเลขขนาด $n = 32$ บิต ชนิดที่ 1 โดยใช้ ALU ขนาด $2n = 64$ บิต และรีจิสเตอร์ตัวตั้งขนาด $2n = 64$ บิต (ที่มา: Patterson and Hennessy (2000))	23
2.6	วงจorcูณเลขขนาด $n = 32$ บิต ชนิดที่ 2 โดยใช้ ALU ขนาด $n = 32$ บิต และรีจิสเตอร์ตัวตั้งขนาด $n = 32$ บิต (ที่มา: Patterson and Hennessy (2000))	25
2.7	โครงสร้างของเลขศนิยมฐานสองชนิดจุดลอยตัวตามมาตรฐาน IEEE 754 Double-Precision และ Single-Precision ที่มา: pybugs.com	34
2.8	เลขศนิยมโดยตัวชนิดนอมัลໄล์ช์ตั้งแต่ $\pm N_{min}$ ถึง $\pm N_{max}$ และชนิดที่ไม่สามารถเขียนแบบนอมัลໄล์ช์ ตั้งแต่ $\pm D_{min}$ ถึง $\pm D_{max}$	39
2.9	วงจรบวกเลขชนิดจุดลอยตัวตามมาตรฐาน IEEE 754 โดยรับค่าอินพุทจำนวน 2 ตัวด้านบน และเอาท์พุทผลลัพธ์ด้านล่าง (ที่มา: Patterson and Hennessy (2000))	40

List of Figures

2.10 วิจารคุณเลขทศนิยมฐานสองชนิดจุดลอยตัวตามมาตรฐาน IEEE 754 โดยรับค่าอินพุทจำนวน 2 ตัวด้านบน และเอาท์พุตผลลัพธ์ด้านล่าง (ที่มา: Patterson and Hennessy (2000))	43
2.11 การเก็บข้อความในหน่วยความจำในรูปของรหัส ASCII ด้วยตัวแปร str ซึ่งเป็นตัวแปรชนิด char[10] str="Hello!" ที่แอ็ดเดรสเริ่มต้น 0x50 (ที่มา: Harris and Harris (2013))	45
2.12 ตารางรหัสแอกซ์เพรสส์ (ASCII) และ Unicode สำหรับตัวอักษรจำนวน $2^8=256$ ตัว ภาษาอังกฤษและภาษาไทย ที่มา: ascii-table.com	46
3.1 ตำแหน่งของอุปกรณ์ต่างๆ บนบอร์ด Pi3 ที่มา: www.raspberryhome.net	52
3.2 การเชื่อมโยงอุปกรณ์ต่างๆ บนบอร์ด Pi3 โดยมีชิป BCM2837 เป็นศูนย์กลาง ที่มา: xdevs.com	53
3.3 บล็อกไดอะแกรมของชิป AllWinner A64 (ที่มา: Allwinner Technology (2015a) และ Allwinner Technology (2015b)) ที่มี ARM Cortex A53 คล้ายกับ BCM2837 บนบอร์ด Raspberry Pi3	55
3.4 หน่วยความจำไดนามิกแรมด้านล่างของบอร์ด Pi3 ไม่เดล B	56
3.5 ตำแหน่งและรหัสประจำปุ่ม (Scan Code หรือ Position Code) บนคีย์บอร์ดชนิด 106 ปุ่ม ที่มา: ps-2.kev009.com	57
3.6 โครงสร้างภายในมาส์ชนิด Optical ประกอบด้วยหลอด LED ส่องแสงกระแทบกับพื้นผิวด้านล่างแล้วสะท้อนกลับมายังตัวรับแสงชนิด CMOS ที่มา: www.pinterest.com	58
3.7 โครงสร้างจอแสดงผลชนิด Color LCD ประกอบด้วยแหล่งกำเนิดแสง (Light Source) ปล่อยแสงทะลุชั้นต่างๆ มาแสดงผล ที่มา: techterms.com	59
3.8 โครงสร้างของอุปกรณ์เก็บรักษาข้อมูล SD ชนิด SDHC ความจุ 16 GB ประกอบด้วยชิปหน่วยความจำแฟลช วงจรควบคุม และวงจรเชื่อมต่อ ที่มา: wikipedia	60
3.9 โครงสร้างหน่วยความจำเสมือนเมื่อทำการบูทรับบัญชีการ Linux สำเร็จ โดยแบ่งเป็น Kernel Space สำหรับ Linux เองและ User Space สำหรับแอพพลิเคชัน ที่มา: linux-india.org	62
3.10 โครงสร้างของโฟลเดอร์ (Folder) หรือไดเรกทอรี (Directory) สำคัญๆ ในระบบปฏิบัติการ Linux ที่มา: freedompenguin.com	64
3.11 โครงสร้างไฟล์ชนิด ELF สำหรับเก็บชุดคำสั่งและข้อมูลในอุปกรณ์เก็บรักษาข้อมูล ที่มา: wikipedia	65
3.12 การบรรจุชุดคำสั่งและข้อมูลจากแอพพลิเคชันหรือโปรแกรมซึ่งบรรจุอยู่ในไฟล์ชนิด ELF ไปยังหน่วยความจำเสมือนส่วนที่เป็น User Space ที่มา: wordpress.com	66
3.13 ขบวนการอ่านข้อมูลจากหน่วยความจำหลักที่ดำเนินการ 125 ที่มา: www.phatcode.net	68
3.14 โครงสร้างของลีนูกซ์เครื่องเนล โดยเน้นส่วนที่เป็น I/O, Memory Management และ Process Management wikipedia	69
3.15 หลักการของ Memory Map File คือการแบ่งหน่วยความจำสำหรับอ่านหรือเขียนไฟล์ ด้านขวาคือ โครงสร้างของไฟล์ในเชิงตรรกะ ด้านซ้ายคือ หน่วยความจำที่ถูกจัดเพื่อใช้พักข้อมูลในไฟล์ ที่มา: safaribooksonline.com	70

3.16 ตัวอย่างซอฟต์แวร์ภาษา C ฟังค์ชันหลักชื่อ main() เมื่อทำงานเสร็จสิ้นจะรีเทิร์นค่า 0	73
3.17 ไฟล์การพัฒนาซอฟต์แวร์จากซอฟต์แวร์ให้เป็นไฟล์ Executable (ELF) หรือโปรแกรมประยุกต์ (Application Program) หรือย่อๆว่า แอพ (ที่มา: slideplayer.com)	73
3.18 ตัวอย่างการลิงค์ (Link) หรือรวมไฟล์อ้อมบเจ็คท์หลัก ไฟล์อ้อมบเจ็คท์เสริม และไฟล์ไลบรารี เข้าด้วยกันเป็นไฟล์โปรแกรมหรือแอพพลิเคชัน ที่มา: google.com	77
3.19 ตัวอย่างการแปลงจากคำสั่งแอสเซมบลีทางด้านขวา เป็นคำสั่งภาษาเครื่องทางด้านซ้าย (ที่มา: Harris and Harris (2013))	78
4.1 โครงสร้างภายในของ ARM Cortex A53 จำนวน 4 คอร์ ที่มา: tomshardware.com	81
4.2 โครงสร้างของชีพียูแต่ละคอร์ประกอบด้วยรีจิสเตอร์ R0-R15 แคชต่างๆ และหน่วยความจำหลัก, (VA: Virtual Address)	83
4.3 คำสั่งของโปรแกรมที่ถูกอ่าน (Load) เข้าสู่หน่วยความจำและแสดงในรูปของภาษาแอสเซมบลีบนโปรแกรมซิมูเลเตอร์ (Simulator) ในบริเวณเข็มเม้นท์ Text ของหน่วยความจำ ที่มา: infocenter.arm.com	85
4.4 รีจิสเตอร์สำหรับเก็บสถานะของชีพียู ณ ปัจจุบัน (Current Program Status Register: CPSR) infocenter.arm.com	92
4.5 ตัวอย่างคำสั่งการบวกค่าในรีจิสเตอร์ที่ได้จากการซิฟท์ไปทางซ้ายจำนวน 2 บิต ที่มา: CodeMachine.com	93
4.6 ไฟล์ชาร์ตการทำงานของโครงสร้างการเขียนโปรแกรม IF	96
4.7 ไฟล์ชาร์ตการทำงานของโครงสร้างการเขียนโปรแกรม IF-ELSE	98
4.8 ไฟล์ชาร์ตการทำงานของการวนรอบชนิด FOR	101
4.9 ไฟล์ชาร์ตการทำงานของโครงสร้างการเขียนลูป While	102
4.10 ไฟล์ชาร์ตการทำงานของโครงสร้างการเขียนลูป Do While	103
4.11 ไฟล์ชาร์ตการทำงานของการวนรอบ For 2 ชั้น	105
4.12 ไฟล์ชาร์ตการทำงานของการเรียกใช้ฟังค์ชัน sum(x,y)	106
4.13 การทำงานของคำสั่ง BL myFunction เมื่อมีการเรียกใช้ 2 ครั้ง	108
4.14 การควบรวมชุดคำสั่งภาษาแอสเซมบลีของ ARM ตั้งแต่เวอร์ชัน 5 ถึงเวอร์ชัน 8A โดยเวอร์ชันใหม่จะรวมเวอร์ชันเก่าเพื่อรับการทำงานซอฟต์แวร์เดิม ที่มา: community.arm.com	110
5.1 ลำดับชั้นของหน่วยความจำชนิดต่างๆ สำหรับชิป BCM2837, (VA: Virtual Address)	114
5.2 การแมป (Map) แอดเดรสเสมือน (Virtual Address) ขนาด 2GB เป็นแอดเดรสกายภาพ (Physical Address) ขนาด 64MB ตามหลักการหน่วยความจำเสมือนชนิดเพจ (Paging Virtual Memory หมายเหตุ เส้นประ คือ รอยต่อระหว่างเพจของหน่วยความจำเสมือน) ที่มา: wikiwand.com	117
5.3 โครงสร้างของหน่วยความจำเสมือน (Virtual Memory) ของบอร์ด Pi3 โมเดล B ซึ่งใช้ชิปตระกูล BCM283x, x=5, 6, 7 หมายเหตุ ขนาดของรูปไม่เป็นไปตามพื้นที่ตามความเป็นจริง, MMU: Memory Management Unit ที่มา: Broadcom (2012)	118

5.4 โครงสร้างภายในของชิป ARM Cortex A7 ประกอบด้วย TLB และ แคช หลายระดับเพื่อรองรับการทำงานของหน่วยความจำเสมือน (Virtual Memory)	119
5.5 การเข้มต่อระหว่างรีจิสเตอร์ แคชลำดับที่ 1 และ 2 และหน่วยความจำหลัก (VA: Virtual Address) ภายในชิป BCM2837	121
5.6 การทำงานของแคชคำสั่ง (Instruction Cache) ชนิด Direct Map ที่มา: Clements (2013)	123
5.7 การทำงานของแคชลำดับที่ 1 แคชคำสั่ง (Instruction Cache) ชนิด Set Associative ที่มา: Clements (2013)	124
5.8 ตัวถังแบบ TSOP ของชิปหน่วยความจำชนิดสแตติกแรม Cypress 62256 ที่มา: Cypress Semiconductor (2002)	126
5.9 โครงสร้างของหน่วยความจำชนิดสแตติก Cypress 62256 ที่มา: Cypress Semiconductor (2002)	127
5.10 ไดอะแกรมเวลา (Timing Diagram) สำหรับการอ่าน (Read) ข้อมูลของหน่วยความจำชนิดสแตติกแรม ที่มา: Cypress Semiconductor (2002)	128
5.11 ไดอะแกรมเวลา (Timing Diagram) สำหรับการเขียน (Write) ข้อมูลของหน่วยความจำชนิดสแตติกแรม ที่มา: Cypress Semiconductor (2002)	129
5.12 รูปถ่ายด้านบน (Top View) ด้านล่าง (Bottom View) ด้านข้าง (Side View) และภาพตัดขวาง (Cross Section) ของชิป DRAM โดยใช้เทคโนโลยี TSV (Through Silicon Via) ผู้ผลิตบริษัท Elpida ที่มา: Micron Technology (2014)	130
5.13 โครงสร้างภายในชิปหน่วยความจำ DRAM ชนิด DDR2 ประกอบด้วยダイ (Die) 2 ชิ้น แต่ละชิ้นมีบล็อกความกว้าง 16 บิต รวมเป็น 32 บิต ที่มา: Micron Technology (2014)	132
5.14 ไดอะแกรมเวลา (Timing Diagram) สำหรับการอ่านข้อมูลของหน่วยความจำ DRAM รุ่น Elpida B8132B4PB-8D-F คำสั่ง Burst READ โดย RL= 5 BL= 4 หมายเหตุ รูปมีการตัดต่อเพื่อเน้นบริเวณที่สำคัญ, NOP=No Operation	134
5.15 ไดอะแกรมเวลา (Timing Diagram) สำหรับการเขียนข้อมูลของหน่วยความจำ Elpida รุ่น B8132B4PB-8D-F ตามคำสั่ง Burst WRITE – WL = 1, BL = 4 หมายเหตุ รูปมีการตัดต่อเพื่อเน้นบริเวณที่สำคัญ ที่มา: Micron Technology (2014)	135
5.16 ไดอะแกรมเวลาสำหรับการรีเฟรชหน่วยความจำชนิด DRAM ขนาด 1 กิกะบิต ที่มา: Micron Technology (2014)	136
5.17 การประกอบอาหารตามสูตร ที่มา: https://www.pinterest.com	138
6.1 การเข้มโยงอุปกรณ์ต่างๆ บนบอร์ด Pi3 โดยมีชิป BCM2837 หรือ CM3 (Compute Module 3) เป็นศูนย์กลางด้วยขาจำนวน 200 ขา ที่มา: Ltd (2019)	142
6.2 หัวเชื่อมต่อ HDMI ชนิด Female ประกอบด้วยขาสัญญาณทั้งหมด 19 ขา ที่มา: wikipedia	143
6.3 การส่งแพ็กเก็ตข้อมูลและควบคุมด้วยช่องสัญญาณ TMDS ในช่วงต่างๆ สำหรับความละเอียดในการแสดงผล 720x480 ต่อเฟรม ที่มา: freebsd.org	145
6.4 จอแสดงผลสำหรับเชื่อมต่อระหว่างบอร์ด Pi3 ด้วยอินเตอร์เฟสการแสดงผลแบบอนุกรม (Display Serial Interface) ที่มา: element14.com	146

6.5	สัญญาณ DSI แบ่งเป็นชนิดหนึ่งเลน (Single Lane) และชนิด 4 เลน ที่มา: wikipedia . . .	148
6.6	การเชื่อมต่อระหว่างบอร์ด Pi3 และกล้องขนาดเล็กด้วยอินเตอร์เฟสกล้องแบบอนุกรม (Camera Serial Interface) ที่มา: element14.com	149
6.7	รูปคลื่นไซน์ (Sine Wave) และสัญญาณ PCM (Pulse Code Modulation) 16 ระดับ ที่มา: wolfcrow.com	151
6.8	บัฟเฟอร์สำหรับส่งและรับ ข้อมูลเสียงดิจิทัลชนิด PCM จากการเชื่อมต่อชนิด I2S ที่มา: Broadcom (2012)	152
6.9	แจ็ค 3.5 มม. (กลาง) ชนิด 4 ขั้วสำหรับเสียบกับบอร์ด Pi3 (ขวา) เพื่อส่งสัญญาณภาพไปยังแจ็ค RCA (เหลือง) และสัญญาณเสียงไปยังแจ็ค RCA (แดงและขาว) ที่มา: stackexchange.com	153
6.10	หัวเชื่อมต่อ USB ชนิด A (บน ฝั่งโซส์ท) และ B (ล่าง ฝั่งอุปกรณ์) ประกอบด้วยสัญญาณ 4 เส้น กราวด์ (GND) Data+ Data- และไฟเลี้ยง 5 โวลท์ ที่มา:	154
6.11	โครงสร้างของชิป LAN 9514 ภายในประกอบด้วยวงจร USB Hub และวงจร Ethernet ที่มา: Microchip Technology (2009)	155
6.12	หัวเชื่อมต่อชนิด RJ45 (ซ้ายสุด) สำหรับการเชื่อมต่อเครือข่ายท้องถิ่น (Local Area Network) แบบมีสาย Ethernet ที่มา:	156
6.13	การเข้าปลายสาย RJ45 ทั้งสองด้าน สำหรับการเชื่อมต่อระหว่างเครื่องคอมพิวเตอร์และอุปกรณ์สวิทช์ (Switch) ตามมาตรฐาน TIA T568B ที่มา:	156
6.14	รูปถ่ายและรูปขยายของชิป BCM 43438 สำหรับเชื่อมต่อเครือข่ายไร้สายท้องถิ่น (Wireless Local Area Network) หรือ WiFi และเครือข่ายไร้สายบลูทูธ (Bluetooth) ที่มา: Corporation (2017)	157
6.15	บล็อกโดยรวมของชิป BCM 43438 ประกอบด้วยขาสัญญาณเชื่อมต่อเสาอากาศ และขาเชื่อมต่อกับไมโครคอนโทรลเลอร์ ที่มา: Corporation (2017)	158
6.16	การแมปหน่วยความจำระหว่างแอดเดรสบัส (VC ชีพิยูBus Address) และแอดเดรสภายนอกของ ARM ที่มา: Broadcom (2012) หมายเหตุ VC: Video Core	160
6.17	โครงสร้างภายในของ GPIO สำหรับชิปะรากุลเดียวกับ BCM2835 ที่มา: Broadcom (2012)	164
6.18	โครงสร้างภายในของ Generic Interrupt Controller (GIC) และการเชื่อมโยงกับโมดูล อื่นๆ กับ ARM Advanced eXtensible Interface (AXI) ที่มา: arm.com, APB (ARM Peripheral Bus)	169
6.19	การทำงานและโดยรวมเวลาของ Generic Interrupt Controller (GIC) ที่มา: ?gic	170
6.20	การเชื่อมต่อระหว่าง Cortex A5 และโมดูล DMA (Direct Memory Access) ด้วยบัส AXI (Advanced eXtensible Interface) ภายในชิป ที่มา: design-reuse.com	172
6.21	การแปลงไฟกระแทก 5 โวลท์เป็นไฟกระแทก ด้วยไอซี PAM 2306 DC-DC Coverter คู่ กำหนดค่าเอาท์พุทด้วยค่าตัวหนีนี่ยืน L ₁ และ L ₂ หน่วยเป็น ไมโครเอนรี ที่มา: Diodes (2012)	174

List of Figures

7.1	โครงสร้างของระบบไฟล์ในระบบปฏิบัติการตระกูล Unix ที่มา: Demblon and Spitzner (2004)	179
7.2	โครงสร้างของไอโอเนตและการเข้ามายกับบล็อกข้อมูลของไฟล์หนึ่งไฟล์ ที่มา: ?	181
7.3	หน่วยความจำแฟลช NAND พลิตโดยบริษัท Micron Technology โดยใช้ตัวถังชนิด TSOP (Thin Small Outline Package) จำนวน 48 ขา ที่มา: Micron Technology (2004)	184
7.4	โครงสร้างภายในหน่วยความจำ Flash NAND ที่มา: Micron Technology (2004)	185
7.5	ໄດ້ອະແກນเวลาของการอ่านข้อมูลแบบเพจ (Page Read) ของหน่วยความจำ Flash NAND ที่มา: Micron Technology (2004)	187
7.6	โครงสร้างภายในการ์ด SD Memory ที่มา: SanDiskCorporation (2003)	190
7.7	ໄດ້ອະແກນเวลาของการอ่านข้อมูลจำนวนหลายบล็อกจากการ์ด SD Memory ที่มา: SanDiskCorporation (2003)	192
7.8	ໄດ້ອະແກນเวลาของการเขียนข้อมูลจำนวนหลายบล็อกจากการ์ด SD Memory ที่มา: SanDiskCorporation (2003)	192
7.9	ແຜ່ນວງຈົບປັດພົມພາຍໃນອຸປະກຣົນ SSD ຊົນດ SATA III ປະກອບດ້ວຍຝິທິນ່ວຍຄວາມຈຳແພລະ ໄດ້ນາມືກແຮມ ຄອນໂທຣເລອ້ຽສໍາຫັບຄວບຄຸມ ที่มา: thatoldnews.site	193
7.10	ບລືອກໄດ້ອະແກນພາຍໃນ SSD ປະກອບດ້ວຍ ສ່ວນເຂື່ອມຕ່ອງກັບເຄື່ອງ ໄມໂຄຣຄອນໂທຣເລອ້ຽບັຟເພື່ອຮັບແລ້ວ ແລ້ວຍຄວາມຈຳແພລະ ที่มา: codecapsule.com	194
7.11	โครงสร้างและองค์ประกอบของອຸປະກຣົນຢາຣດີສົກີໂດຣວ (HDD)	195
7.12	โครงสร้างของຢາຣດີສົກີໂດຣວແບ່ງເປັນໄຊລິນເດວົຣ ແທຶກແລະເຊື້ອເດວົຣ ที่มา: computable-minds.com	197
A.1	กรອกເລີຂ -123 ລົງໃນກລ່ອງຂໍ້ຄວາມ ແລະ ຄລິກເລືອກທີ່ປຸ່ມ Signed ເພື່ອໃຫ້ເປັນເລຂ້ານສອງ ຊົນດ Signed	202
A.2	ຜລັບຮັດຈາກການແປ່ງເລີຂ -123 ໃຫ້ເປັນເລຂ້ານສອງ ຊົນດ Signed 2-Complement ຄວາມຍາວ 24 ປີທ	203
A.3	ຜລັບຮັດຈາກການແປ່ງເລີຂ -123 ໃຫ້ເປັນເລຂ້ານສົບທກ ຊົນດ Signed 2-Complement ຄວາມຍາວ 24 ປີທ ອີ່ 6 ຕັ້ງເລີຂ	203
A.4	ໜ້າຕ່າງວາງເລີກແປ່ງເລີຂ -123 ໃຫ້ເປັນເລຂ້ານສອງ ຊົນດ Signed 2-Complement ຄວາມຍາວ 32 ປີທ	204
A.5	ຜລັບຮັດເລີກແປ່ງເລີຂ -123 ໃຫ້ເປັນເລຂ້ານສອງ ຊົນດ Signed 2-Complement ຄວາມຍາວ 32 ປີທ	204
A.6	ຜລັບຮັດຈາກການແປ່ງເລີຂ 123 ໃຫ້ເປັນເລຂ້ານສອງ ຊົນດ Single Precision	207
A.7	ຜລັບຮັດຈາກການແປ່ງເລີຂ -123 ໃຫ້ເປັນເລຂ້ານສອງ ຊົນດ Single Precision	208
A.8	ເມນູດ້ານລ່າງສຸດຂອງໜ້າເວັບ ເພື່ອເລືອກເລຂ້ານສອງ ຊົນດ Single Precision (Binary32) ແລະ Double Precision (Binary64)	208
A.9	ຜລັບຮັດຈາກການບວກເລີຂ -123+123 ໃຫ້ເປັນເລຂ້ານສອງ ຊົນດ Single Precision	209
A.10	ຜລັບຮັດຈາກການຄຸນເລີຂ -123 x 123 ໃຫ້ເປັນເລຂ້ານສອງ ຊົນດ Single Precision	209

A.11	ผลลัพธ์จากการแปลงเลข 123 ให้เป็นเลขฐานสองชนิด Double Precision	210
A.12	ผลลัพธ์จากการแปลงเลข -123 ให้เป็นเลขฐานสองชนิด Double Precision	210
A.13	ผลลัพธ์จากการบวกเลข -123+123 ให้เป็นเลขฐานสองชนิด Double Precision	211
A.14	ผลลัพธ์จากการคูณเลข -123 x 123 ให้เป็นเลขฐานสองชนิด Double Precision	211
A.15	เว็บสำหรับการตอบคำถามเพื่อสร้างเลขหรือแปลงเลขฐานสิบด้วยมาตรฐาน IEEE 754 Single Precision การกดเลือกคือทำให้ปุ่มนั้นเท่ากับ '1'	212
A.16	ผลลัพธ์จากการกรอกและแปลงตัวอักษร ໄ ທ ຍ ก ຂ ຄ a b c เป็นรหัสต่างๆ	214
B.1	รูปแสดงรายการอุปกรณ์สำหรับประกอบบอร์ด	216
B.2	บอร์ด Pi 3 เมื่อติดอีทซิงค์เรียบร้อยแล้ว	217
B.3	แผ่นพลาสติก 5 ชิ้น สำหรับประกอบเป็นกล่องของบอร์ด Pi3	218
B.4	บอร์ด Pi3 ที่มีกล่องประกอบเรียบร้อยแล้ว	219
B.5	บอร์ด Pi3 เมื่อประกอบขาเข้ามายาว 2x20	219
B.6	การเชื่อมต่อคีย์บอร์ดและมาสเตอร์กับช่องเสียบสาย USB บนบอร์ด Pi3	220
B.7	การเชื่อมต่อไฟเลี้ยงจากอแดปเตอร์ทางหัวไมโคร USB กับบอร์ด Pi3	220
B.8	ภาพสีรุ้งบนจออันเกิดจากบอร์ด Pi3	221
C.1	การฟอร์แมตการ์ด microSD บนระบบ Windows	224
C.2	การฟอร์แมตการ์ด microSD บนระบบ Mac OS	224
C.3	ไฟล์เดอร์ในระบบปฏิบัติการวินโดว์ส หลังจาก Unzip ไฟล์ NOOBS.zip	225
C.4	ไฟล์เดอร์ในระบบปฏิบัติการ Mac OS X หลังจาก Unzip ไฟล์ NOOBS.zip	226
C.5	การสอดหน่วยความจำ microSD เข้าไปในสล็อตบนบอร์ด Pi3 โดยหมายบอร์ดขึ้นมาโปรดสังเกตการ์ดหน่วยความจำจะต้องหมายขึ้นดังรูป	227
C.6	การติดตั้งระบบ Raspbian	228
C.7	Graphical User Interface ของระบบปฏิบัติการ Raspbian	228
C.8	แสดงรายชื่อสัญญาณ Wi-Fi รอบๆ ที่บอร์ด Pi3 มองเห็น	229
C.9	รายชื่อสัญญาณ Wi-Fi ที่ต้องการเชื่อมต่อ	230
C.10	หน้าต่างสำหรับกรอกรหัส Wi-Fi ที่ต้องการเชื่อมต่ออย่างปลอดภัย	230
C.11	ชื่อสัญญาณ Wi-Fi ที่เชื่อมต่อสำเร็จพร้อมกับหมายเลขแอดเดรส IP ที่ได้รับมอบหมาย	231
D.1	หน้าต่างของไฟล์เมเนจเจอร์ (File Manager)	234
D.2	Shutdown	234
D.3	Icon Terminal	235
D.4	Terminal	235
E.1	หน้าต่างเลือกชนิดโปรแกรมที่จะพัฒนาเป็นชนิด "Console application"	242
E.2	หน้าต่างเลือกภาษาสำหรับโปรแกรมที่จะพัฒนา	243
E.3	การเลือกคอนฟิก Debug สำหรับคอมไฟเลอร์ GNU GCC ในโปรแกรม Lab5	244

List of Figures

E.4 การเพิ่มไฟล์เปล่าให้กับโปรเจคท์ Lab5 ที่สร้างขึ้น	244
F.1 การย้ายไฟล์ main.c ออกจากโปรเจคท์	250
F.2 การเพิ่มไฟล์ใหม่ลงในโปรเจคท์	250
F.3 หน้าต่างกดปุ่ม "Yes" เพื่อยืนยัน	250
F.4 หน้าต่าง Save File ชื่อไฟล์ว่า main.s	251
I.1 หน้าต่างกำหนดขนาดหน่วยความจำสำหรับ GPU ที่ 64 เมกะไบท์	280
I.2 หน้าต่าง Raspberry Pi Configuration แท็บ System สำหรับกำหนดความละเอียดหน้าจอแสดงผล (Resolution)	280
I.3 หน้าต่าง Set Resolution สำหรับกำหนดความละเอียดหน้าจอที่ต้องการ	281
I.4 หน้าต่าง Reboot needed กดปุ่ม Yes เมื่อต้องการรีบูต ณ เวลานั้น	281
I.5 โปรแกรม ALSA Mixer สำหรับควบคุมระดับเสียงบนบอร์ด Pi3	283
J.1 วงจรเข็มต่อหลอด LED กับบอร์ด Pi3 ในการทดลองที่ 10 ที่มา: fritzing.org	296
K.1 วงจรกดปุ่มสำหรับทดลองการเขียนโปรแกรมอินเทอร์รัพต์ในการทดลองที่ 11 ที่มา: fritzing.org	305

บทนำ (Introduction)

คอมพิวเตอร์และซอฟต์แวร์ต่างๆ มีประโยชน์ต่อเศรษฐกิจ การประกอบธุรกิจ อุตสาหกรรมการผลิต การศึกษา รวมถึงชีวิตประจำวันมากขึ้น ตำราเล่มนี้สามารถใช้ประกอบการเรียนการสอนวิชาองค์ประกอบของคอมพิวเตอร์ (Computer Organization) และ วิชาสถาปัตยกรรมคอมพิวเตอร์ (Computer Architecture) เป็นต้น โดยอาศัยบอร์ด Raspberry Pi3 และ ARM Cortex A53 เป็นกรณีศึกษา อีกทั้งใช้เป็นเอกสารประกอบการพัฒนาโปรแกรมเบื้องต้นโดยมีฮาร์ดแวร์เสริมความเข้าใจ

1.1 ชนิดของเครื่องคอมพิวเตอร์

ระบบคอมพิวเตอร์ทั่วไป สามารถแบ่งออกเป็น คอมพิวเตอร์ตั้งโต๊ะ (Desktop computers) คอมพิวเตอร์เซิร์ฟเวอร์หรือแม่ข่าย (Server computers) คอมพิวเตอร์พกพา (Portable Computers) และคอมพิวเตอร์ฝังตัว (Embedded computers) หรือในชื่อ Internet of Things (IoT)

1.1.1 คอมพิวเตอร์ตั้งโต๊ะ (Desktop computers)

คอมพิวเตอร์ตั้งโต๊ะสามารถใช้งานได้หลากหลายชีวิตร่วมกับซอฟต์แวร์ที่นำมาติดตั้ง โดยซอฟต์แวร์ประยุกต์เหล่านี้ขึ้นตรงกับซอฟต์แวร์ระบบปฏิบัติการเป็นหลัก เช่น ระบบปฏิบัติการไมโครซอฟต์วินโดวส์ (Microsoft Windows) เวอร์ชันต่างๆ ระบบปฏิบัติการ MAC OS เวอร์ชัน 10.x ระบบปฏิบัติการลีนุกซ์ (Linux) ซึ่งมีดิสทริบิวชัน (Distribution) เช่น Ubuntu SuSe RedHat เป็นต้น ตำราเล่มนี้จะอ้างอิงกับระบบปฏิบัติการ Raspbian ซึ่งเป็นเวอร์ชันหนึ่งของลีนุกซ์ สำหรับบอร์ด Pi3

1.1.2 คอมพิวเตอร์เซิร์ฟเวอร์หรือแม่ข่าย (Server computers)

คอมพิวเตอร์เซิร์ฟเวอร์หรือเครื่องแม่ข่าย จะต้องติดตั้งระบบปฏิบัติการ เช่นเดียวกับเครื่องคอมพิวเตอร์ตั้งโต๊ะ ยิ่งไปกว่านั้น **คอมพิวเตอร์เซิร์ฟเวอร์เหล่านี้จะต้องให้เปิดให้บริการเครื่องคอมพิวเตอร์ลูกข่าย (Client Computer)** ผ่านทางระบบเครือข่ายอินเทอร์เน็ตตลอดเวลา ดังนั้น คอมพิวเตอร์แม่ข่ายจะมีความจุ สมรรถนะ และความเชื่อมั่นสูงเพื่อรับรองรับการใช้งาน จากผู้ใช้เครื่องคอมพิวเตอร์จำนวนมากซึ่งกระจายอยู่ทั่วโลก ระบบปฏิบัติการที่ได้รับความนิยมสำหรับเครื่องเหล่านี้ ได้แก่ ลีนุกซ์ และไมโครซอฟต์วินโดวส์



Figure 1.1: ตัวอย่างเครื่องคอมพิวเตอร์ชนิดเซิร์ฟเวอร์ จากบริษัท Hewlett Packard รุ่น Proliant ที่มา: datacenterknowledge.com

1.1.3 คอมพิวเตอร์พกพา (Portable Computers)

คอมพิวเตอร์พกพา มีขนาดเล็กและพกพาง่าย ใช้กำลังไฟจากแบตเตอรี่เป็นหลัก ได้แก่ คอมพิวเตอร์โน๊ตบุ๊ค แท็บเล็ตคอมพิวเตอร์ และโทรศัพท์เคลื่อนที่สมาร์ทโฟน (Smart Phone) เป็นต้น ระบบปฏิบัติการที่ได้รับความนิยมสำหรับเครื่องเหล่านี้ ได้แก่ แอนดรอยด์ (Android) และ แอปเปิล iOS ทั้งสองระบบนี้มีใช้บนโทรศัพท์สมาร์ทโฟนและแท็บเล็ต มากกว่า 90% ทั่วโลก ผู้ใช้สามารถติดตั้งแอปพลิเคชันต่อจากระบบปฏิบัติการลีนุกซ์บนบอร์ด Pi3 ได้เช่นกัน รายละเอียดเพิ่มเติมสามารถศึกษาได้จาก howtoraspberry.com

1.1.4 คอมพิวเตอร์ฝังตัว (Embedded computers)

คอมพิวเตอร์ฝังตัว เป็นสมองกลที่ซ่อนอยู่ในระบบ เช่น รถยนต์ เครื่องบิน ทีวี ตู้เย็น ฯลฯ คอมพิวเตอร์ชนิดนี้สามารถทำงานอยู่ได้เงื่อนไขของกำลังไฟ/สมรรถนะ/ราคา ปัจจุบัน คอมพิวเตอร์ฝังตัวได้รับความนิยมเพื่อทำหน้าที่เป็นเซ็นเซอร์ (Sensor) และแขนกล (Actuator) สำหรับงานประเภทต่างๆ มากขึ้นและเชื่อมต่อกับเครือข่ายอินเทอร์เน็ตได้อย่างแพร่หลาย ทำให้มีชื่อเรียกว่า Internet of Things หรือ IoT

1.2 แนวโน้มของจำนวนอุปกรณ์คอมพิวเตอร์ชนิดต่างๆ

แนวโน้มของจำนวนอุปกรณ์คอมพิวเตอร์ กลุ่มคอมพิวเตอร์ส่วนบุคคลตั้งตื้ง และ กลุ่มคอมพิวเตอร์พกพา ประเภทสมาร์ทโฟนและแท็บเล็ต ตั้งแต่ปี ค.ศ. 2005-2015 ตามกราฟในรูปที่ 1.2 ผู้อ่านจะเห็นได้ว่า กลุ่มสมาร์ทโฟนและแท็บเล็ต มีจำนวนเพิ่มมากขึ้นแบบก้าวกระโดด ในขณะที่ กลุ่มพีซีหรือคอมพิวเตอร์ตั้งตื้ง ต้องมีแนวโน้มลดลงไปเรื่อยๆ โดยรายละเอียด คือ สมาร์ทโฟนขนาดใหญ่ (Large Smartphone) จะได้รับความนิยมเพิ่มมากขึ้นเรื่อยๆ ในขณะที่ความนิยมสมาร์ทโฟนขนาดเล็ก (Small Smartphone) มีแนวโน้มลดลง แท็บเล็ตขนาดใหญ่ (Large Tablet) และแท็บเล็ตขนาดเล็ก (Small Tablet) มีแนวโน้มค่อนข้าง

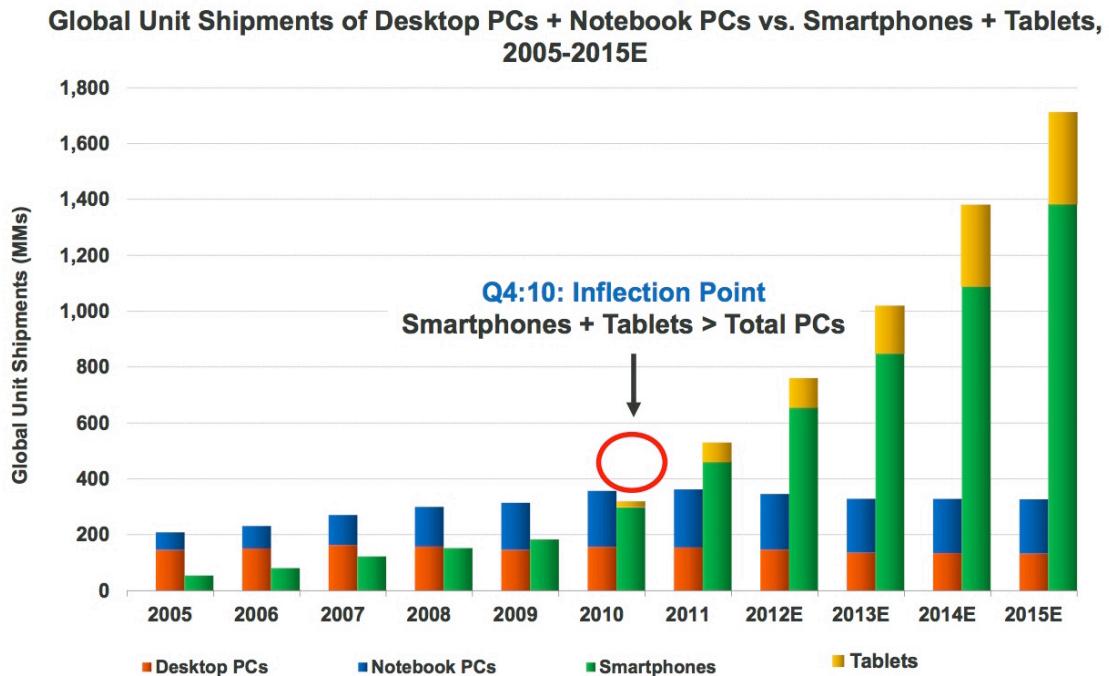


Figure 1.2: ปริมาณยอดขายเครื่องคอมพิวเตอร์ชนิดเดสค์ทอป โน๊ตบุ๊ค สมาร์ทโฟน และแท็บเล็ต ทั่วโลก ในปี ค.ศ. 2005-2015 ที่มา: slideshare.net

คงที่

1.3 ขั้นตอนการผลิตไมโครชิพ

ภายในเครื่องคอมพิวเตอร์ชนิดต่างๆ มีชิพหรือไมโครชิพ (Microchip) ซึ่งในอดีตอาจเรียกว่า วงจรรวม (Integrated Circuit ย่อว่า IC) หรือ ไอซี เมื่อร่วมวงจรดิจิทัลเข้ามานานั้น ซึ่งจะเปลี่ยนเป็น VLSI (Very Large System Integration) และ ULSI (Ultra Large System Integration) ตามลำดับ 'ภายในไมโครชิพจึงมีวงจรดิจิทัลต่างๆ จำนวนมาก จำนวนทรานซิสเตอร์จึงมากตามและหน้าที่หลากหลาย สามารถเขียนโปรแกรมต่อ กันกับจุดแสดงผล อุปกรณ์และปุ่มต่างๆ ให้กล้ายเป็นเครื่องคอมพิวเตอร์ขนาดเล็กลง เรื่อยๆ จนปัจจุบันมีชื่อเรียกใหม่ว่า SoC (System on Chip) ผู้อ่านควรทำความรู้จักขั้นตอนการผลิตชิพเหล่านี้เบื้องต้น เพราะจะช่วยให้เห็นวิวัฒนาการ ต่อไปในอนาคต

ขั้นตอนการผลิต (Process) ชิพ (Chip) ตามรูปที่ 1.3 มีลักษณะคล้ายการทำแผ่นวงจรพิมพ์แต่มีความซับซ้อนและเทคโนโลยีในการผลิตสูง โดยเริ่มต้นจากการนำแท่งผลึกซิลิกอน (Silicon ingot) บริสุทธิ์มาสไลซ์ (Slice) หรือแล้วให้เป็นแผ่นบาง เรียกว่า เวเฟอร์เปล่า (Blank Wafer) เมื่อนำแผ่นเวเฟอร์เปล่าเหล่านี้ไปผ่านขั้นตอนการปลูกถ่ายสาร (Doping) และอื่นๆ จำนวน 20-40 ขั้นตอน จนกลายเป็นแผ่นเวเฟอร์ที่มีลวดลาย (Patterned wafers) โดยเวเฟอร์หนึ่งแผ่นประกอบด้วย ダイ (Die) จำนวนหนึ่งกระจายในลักษณะตารางดังรูป หลังจากการทดสอบダイแต่ละตัวด้วยตัวทดสอบเวเฟอร์ (Wafer tester) ダイตัวที่ไม่ผ่านการทดสอบจะถูกกำกับเพื่อทำเครื่องหมายแล้ว แผ่นเวเฟอร์จะถูกตัดด้วยไดเซอร์ (Dicer) ให้ダイแยกตัวออกจากกันเป็นสี่เหลี่ยม ด้วยตัวที่ผ่านการทดสอบก็จะถูกยึด (Bond) เข้ากับตัวถัง (Package) เชื่อมต่อจากับตัวถังด้วยลวดทองคำ ให้ตรงตามลักษณะที่ต้องการ หลังจากการทดสอบด้วย Part tester

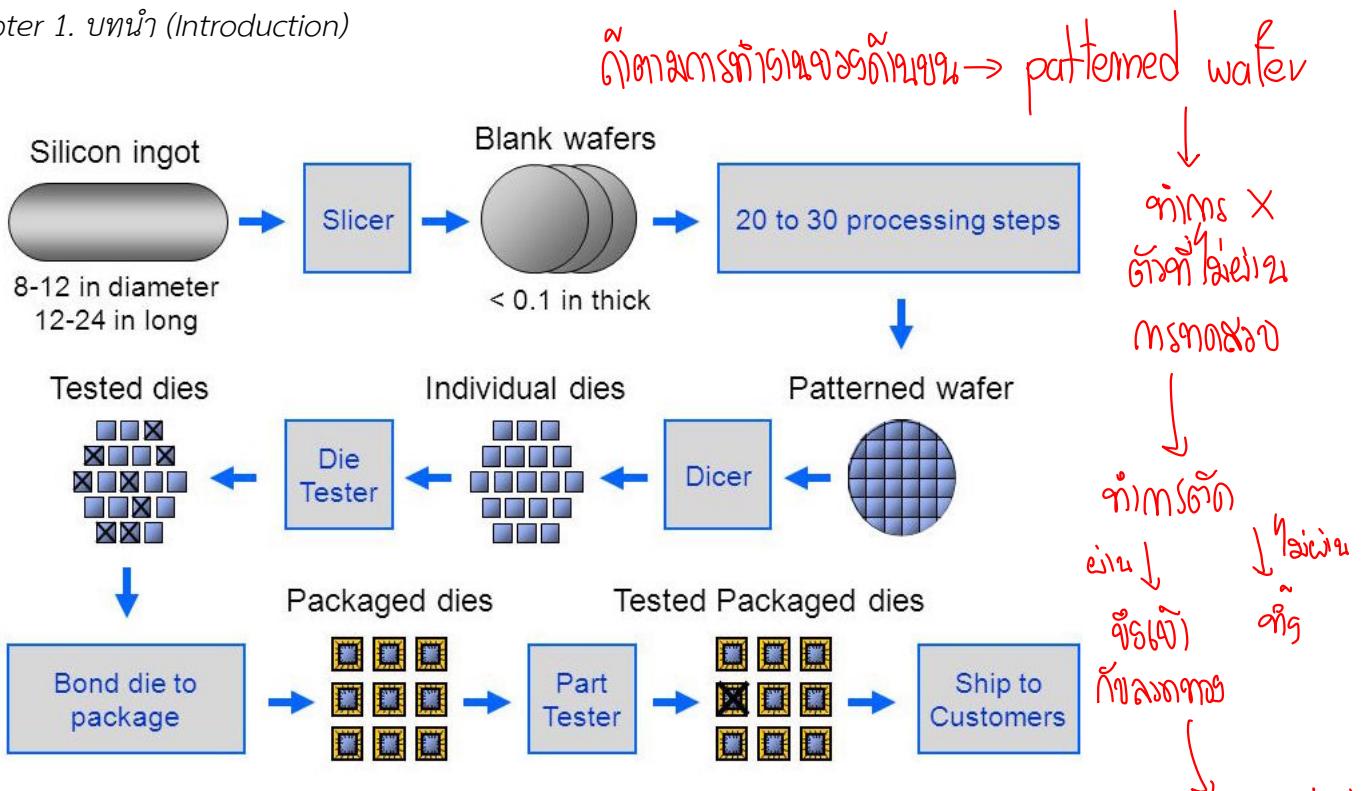


Figure 1.3: ขั้นตอนการผลิตของชิป (Chip) วงจรดิจิทัลไมโครโปรเซสเซอร์และอื่นๆ ที่มา: slide-player.com

เพื่อทดสอบหาชิปที่ไม่ผ่าน เมื่อพับแล้วเครื่องจะทำเครื่องหมายและไม่ถูกส่ง (Ship) ไปยังลูกค้า

ชิป (Chip) คือ ダイที่ห่อหุ้มด้วยแพ็คเกจเรียบร้อยแล้ว ซิลิกอน คือ ธาตุหมู่ที่ 4 มีวาเลนซ์อิเลคตรอน (Valence Electron) วงนอกสุดจำนวน 4 ตัว แผ่นซิลิกอนเวย์ฟอร์ คือ การนำธาตุซิลิกอนที่สกัดและหลอมจนเป็นแท่ง (Silicon Ingot) มาแล่เป็นแผ่นบางๆ **การโดปสาร** คือ การเปลี่ยนแปลงแผ่นเวย์ฟอร์บางตำแหน่งเพื่อให้กลไกเป็นสาร P โดยการเพิ่มไฮล (Hole) และ กลไกเป็นสาร N โดยการเพิ่มอิเลคตรอน ทำให้เกิดการเชื่อมโยงกันเป็น ทรานซิสเตอร์ 2 ชนิด คือ N-MOS และ P-MOS กลไกเป็น วงจรลอจิคเกต (Logic Gate) ต่างๆ และโมดูลต่างๆ โดยใช้แร่ทองแดง และอลูมิเนียม (Aluminum) ทำหน้าที่เป็นสายตัวนำไฟฟ้าเชื่อมโยงทรานซิสเตอร์และวงจรเกทต่างๆ เข้าด้วยกันจนกลไกเป็นวงจรดิจิทัลที่มีความซับซ้อน ตามที่กล่าวไว้ก่อนหน้านี้ เราเรียก ตัวนำไฟฟ้าในแนวราบว่า อินเตอร์คอนเนกต์ (Interconnect) และเรียกตัวนำในแนวตั้งว่า เวีย (Via)

1.4 บอร์ด Raspberry Pi และชิป Broadcom BCM 2835

บอร์ด Raspberry Pi ตัวแรกอาศัยชิป BCM 2835 เป็นชิปหลักตั้งแต่ปี 2012 ไปปี 2015 บอร์ด Raspberry Pi2 ใช้ชิป BCM 2836 ปี 2016 บอร์ด Raspberry Pi3 ใช้ชิป BCM 2837 ล่าสุดปี 2019 บอร์ด Raspberry Pi4 ใช้ชิป BCM 2711 โดยชิปทั้งหมดนี้ บริษัท Broadcom ประเทศสหรัฐอเมริกา เป็นผู้ผลิต รูปที่ 1.4 คือ ภาพถ่ายダイ (Die) หรือแผ่นวงจรซิลิกอนของ BCM2835 ขนาดประมาณ 50-100 เท่าของชิปจริง โดยปราศจากแพ็คเกจพลาสติกสีดำห่อหุ้ม ชิป BCM2835 มีคุณลักษณะเฉพาะ (Specification) ดังนี้

- **โปรเซสเซอร์ ARM1176JZF-S** จำนวน 1 คอร์ (Core) ทำงานที่สัญญาณนาฬิกาความถี่ 700 MHz
- แคชล้ำดับที่ 1 (L1) ขนาด 16 กิโลไบท์ แคชล้ำดับที่ 2 (L2) ขนาด 128 กิโลไบท์

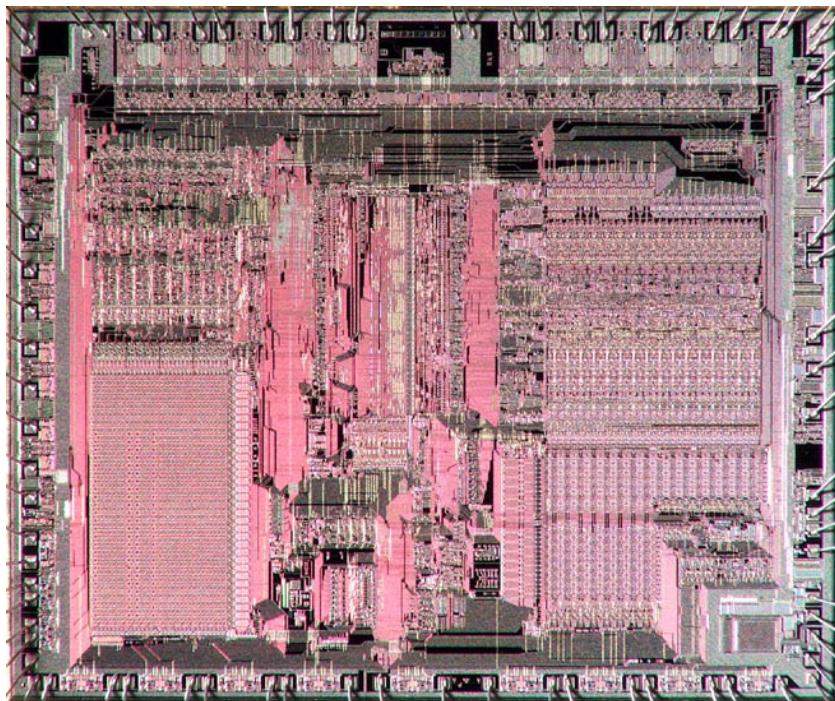


Figure 1.4: ภาพถ่ายซิลิกอนด้วย (Silicon Die) ของชิป BCM 2835 บนบอร์ด Raspberry Pi ที่มา: raspberrypi.org

- หน่วยประมวลผลด้านกราฟิก หรือ GPU รุ่น VideoCoreIV ภายในชิปตัวเดียวกัน รายละเอียดเพิ่มเติม

บอร์ด Pi3 ใช้เทคโนโลยีที่สูงขึ้น สมรรถนะและประสิทธิภาพเพิ่มขึ้น โดยมีจำนวนซีพียูคอร์ (Core) 4 คอร์ รายละเอียดเพิ่มเติมในบทที่ 3 ล่าสุด บอร์ด Pi4 ในปัจจุบัน ใช้เทคโนโลยีที่สูงขึ้นกว่า สมรรถนะและประสิทธิภาพเพิ่มขึ้นอีก แต่มีซีพียูจำนวน 4 คอร์เท่าเดิม

1.5 สรุปท้ายบท

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

1.6 คำถามท้ายบท

- จงให้เหตุผลว่าทำไมคอมพิวเตอร์ชนิดแท็บเล็ตและสมาร์ทโฟนจึงได้รับความนิยมเพิ่มขึ้น จนเกือบแทนที่คอมพิวเตอร์ชนิดตั้งโต๊ะและโน้ตบุ๊ก ในแร่ปัจจุบัน

- ขนาดของอุปกรณ์ → ใจกลางเดลล์เรนโนลส์ตัน และแม่ข่ายที่มีคุณภาพดีที่สุดในโลก, ฐาน one app
- การเชื่อมต่อกับผู้ใช้ (Human Interface) →
- ราคาและโปรโมชันการขาย → ร้านเช่นๆ ห้างสรรพสินค้า ห้างสรรพสินค้า ห้างสรรพสินค้า
- อื่นๆ

2. เราสามารถใช้เครื่องคอมพิวเตอร์ชนิดตั้งโต๊ะแทนเครื่องชนิดเซิร์ฟเวอร์ได้หรือไม่ เพราะเหตุใด
จะบอกเหตุผล ข้อดีและข้อเสีย ในแบบมุมดังต่อไปนี้

- ความเชื่อมั่นของอุปกรณ์ (Reliability) → ตัวอย่าง เช่น server ลูกค้าของบริษัทฯ
- คุณสมบัติจำเพาะของอุปกรณ์ (Specification) → ตัวอย่าง เช่น server ลูกค้าของบริษัทฯ
- การลงทุนและค่าใช้จ่ายอื่นๆ → server (เงินต่อ 月) ตัวอย่าง เช่น บริษัทฯ
- ระบบปฏิบัติการ → ตัวอย่าง เช่น windows server ตัวอย่าง เช่น บริษัทฯ
- อื่นๆ

3. เหตุใดการผลิตชิพจำเป็นต้องควบคุมอากาศในด้านความสะอาดที่เรียกว่า ห้องคลีนรูม (Clean Room)

4. จะบอกสภาพแวดล้อมที่ไม่เหมาะสมต่อการทำงานของเครื่องคอมพิวเตอร์ชนิดต่างๆ ในแบบมุมดังต่อไปนี้

- อุณหภูมิ
- ความชื้น
- สารเคมี
- ความสูงจากระดับน้ำทะเล
- อื่นๆ

แหล่งกำเนิดความร้อน

5. การระบายน้ำร้อนของอุปกรณ์คอมพิวเตอร์มีความสำคัญต่อการทำงาน จงบอกข้อดีและข้อเสีย
ของวิธีต่อไปนี้

- ฮีทซิงค์ (Heat Sink) → ตัวช่วยลดความร้อนให้คงที่มากขึ้น (ประสิทธิภาพ)
- พัดลม → ลดความร้อน
- ของเหลว → ตัวช่วยลดความร้อน เช่น น้ำ, น้ำยา
- อื่นๆ

ข้อมูลและคณิตศาสตร์ในคอมพิวเตอร์ (Computer Data and Arithmetic)

ไมโครโปรเซสเซอร์ตัวแรกของโลก คือ Intel 4004 ในปี ค.ศ. 1971 ซึ่งประดิษฐ์โดยบริษัท Intel สามารถคำนวณเลขจำนวนเต็มได้เพียง 4 บิตโดยใช้รหัส BCD (Binary Coded Decimal) ซึ่งรหัส BCD นี้สามารถใช้คำนวณเลขฐานสิบได้ โดยใช้เลขฐานสองสี่บิตตั้งแต่ 0000 ถึง 1001 แทนเลขฐานสิบจาก 0 ถึง 9 ตามลำดับ ดังนั้น ไมโครโปรเซสเซอร์ในอดีตจะเป็นต้องคำนวณตัวเลขทั้งจำนวนเต็มและทศนิยมเพื่อรองรับการใช้งานทางคณิตศาสตร์และตรรกศาสตร์ได้หลากหลายและซับซ้อนมากขึ้น

ในปัจจุบัน การใช้งานเครื่องคอมพิวเตอร์นิดต่างๆ เพื่อประมวลผลข้อมูลและอักษรให้กลายเป็นสารสนเทศ (Information) ที่เป็นประโยชน์ต่อธุรกิจการค้า สื่อสังคม และวัตถุประสงค์อื่นๆ การประมวลผลข้อมูลและอักษรเหล่านี้ จะเป็นต้องใช้คณิตศาสตร์ เช่น การบวก/ลบ คูณ/หาร และและตรรกศาสตร์ด้วยวงจรดิจิทัล ซึ่งจัดเป็นฮาร์ดแวร์เพื่อให้ใช้เวลาคำนวณน้อยที่สุด โดยมุ่งเน้นไปที่ความสามารถในการเข้าใจข้อมูลเชิงจำนวนเป็นตัวเลขฐานสิบ แต่คอมพิวเตอร์จำเป็นต้องประมวลผลด้วยเลขฐานสองแทน ดังนั้น วัตถุประสงค์ของบทนี้ คือ

- เพื่อให้ผู้อ่านเข้าใจเลขจำนวนเต็มชนิดไม่มีเครื่องหมายและมีเครื่องหมาย (Integers: Unsigned and Signed) และคณิตศาสตร์ในเครื่องคอมพิวเตอร์
- เพื่อให้ผู้อ่านเข้าใจเลขทศนิยมฐานสองชนิดจุดคงที่ (Fixed-point real numbers) และคณิตศาสตร์ในเครื่องคอมพิวเตอร์
- เพื่อให้เข้าใจเลขทศนิยมฐานสองชนิดจุดลอยตัว (Floating-point real numbers) และคณิตศาสตร์ในเครื่องคอมพิวเตอร์
- เพื่อให้ผู้อ่านรู้จกรหัสเลขฐานสองตามมาตรฐาน ASCII และ Unicode สำหรับข้อมูลตัวอักษร ชื่อ และข้อความ

ดังนั้น เพื่อให้คอมพิวเตอร์สามารถคำนวณหรือกระทำการเชิงคณิตศาสตร์กับเลขฐานสิบที่เป็นจำนวนเต็ม หรือจำนวนจริงดังรูปเส้นจำนวนในรูปที่ 1.1 ในรูปของเลขฐานสองได้ เลขฐานสองจึงแบ่งเป็นเลขจำนวนเต็ม (Integer) และเลขจำนวนจริง (Real number) คล้ายกับเลขฐานสิบ ส่วนตัวอักษรแต่ละตัวที่มุ่งเน้นอ่านเข้าใจ คือ เลขฐานสองเข่นเดียวกัน

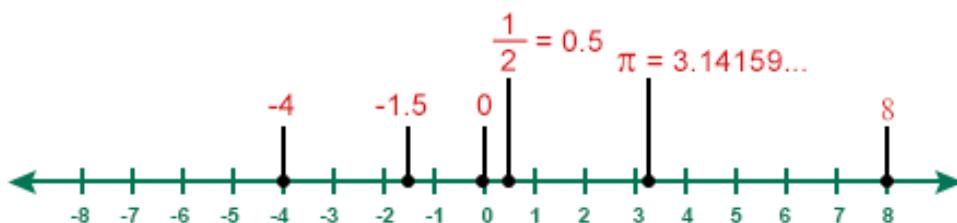


Figure 2.1: เลขจำนวนเต็ม และ เลขจำนวนจริงบนเส้นจำนวน ในคณิตศาสตร์เลขฐานสิบ ที่มา: tutorvista.com

ทั้งนี้เนื่องจากข้อมูลและคำสั่งจะอยู่ในรูปของระดับความต่างศักย์หรือโอลเตจ และทิศทางของกระแสไฟฟ้า เป็นเลข '1' หรือ '0' เรียกว่า บิท เพื่อแทนระดับโอลเตจสูงและโอลเตจต่ำตามลำดับ ซึ่งการจัดเรียงบิทข้อมูลเหล่านี้หลายๆ บิท ให้กลายเป็นเลขฐานสอง และจำเป็นต้องใช้องค์ความรู้ด้านคณิตศาสตร์คอมพิวเตอร์ (Computer Arithmetic)

2.1 ตัวแปรชนิดต่างในภาษา C/C++

ผู้อ่านควรมีพื้นฐานการเขียนหรือพัฒนาโปรแกรมคอมพิวเตอร์ด้วยภาษา C หรือ C++ มาบ้าง เพื่อช่วยให้เข้าใจบทนี้และบทต่อๆ ไปได้ดีขึ้น ตารางที่ 1.1 แสดง ชนิดของตัวแปร ขนาด (บิท) ค่าฐานสิบต่ำสุด (Minimum) และค่าฐานสิบสูงสุด (Maximum) สำหรับภาษา C/C++ ยกตัวอย่าง เช่น ตัวแปรชนิด char นั้น ต้องการพื้นที่จำนวน 8 บิท หรือ 1 ไบต์ โดยมีค่าฐานสิบต่ำสุดเท่ากับ -128 หรือ -2^7 และมีค่าฐานสิบสูงสุดเท่ากับ $+127$ หรือ $+2^7 - 1$

Table 2.1: ชนิดตัวแปร ความยาว (บิท) ค่าฐานสิบต่ำสุดและสูงสุดของตัวแปรแต่ละชนิดในภาษา C/C++

ชนิดตัวแปร	ความยาว(บิท)	ค่าต่ำสุด ₁₀	ค่าสูงสุด ₁₀
char	8	-128	+127
unsigned char	8	0	255
short	16	-32,768	+32,767
unsigned short	16	0	65535
int	32	-2^{31}	$+2^{31} - 1$
unsigned int	32	0	$+2^{32} - 1$
long long	64	-2^{63}	$+2^{63} - 1$
unsigned long long	64	0	$+2^{64} - 1$
float	32	$\pm 2^{-126}$ หรือ $\pm 1.18 \times 10^{-38}$	$\pm 2^{127}$ หรือ $\pm 3.4 \times 10^{38}$
double	64	$\pm 2^{-1023}$ หรือ $\pm 1.23 \times 10^{-88}$	$\pm 2^{1022}$ หรือ $\pm 1.23 \times 10^{88}$

ในขณะที่ตัวแปรชนิด int (integer) นั้น ไม่ครอบคลุมส่วนใหญ่ต้องการพื้นที่จัดเก็บตัวแปรชนิด int นี้ 1 ตัวเป็นพื้นที่ 32 บิท หรือ 4 ไบต์ โดยมีค่าฐานสิบต่ำสุดเท่ากับ -2^{31} หรือ $-2,147,483,648$ และมีค่าฐานสิบสูงสุดเท่ากับ $+2^{31} - 1$ หรือ $+2,147,483,647$

ตัวอย่างการประกาศตัวแปรและตั้งค่าเริ่มต้นในภาษาสูง เช่น ภาษา C/C++ Java ผู้เขียนหรือนักพัฒนาจะต้องประกาศชื่อตัวแปร เมื่อนอกับชื่อตัวแอลคร ก่อนจะตั้งค่าเริ่มต้น (Initialize) หรือเปลี่ยนแปลงค่าของตัวแปรนั้นจากการคำนวณหรือประมวลผลได้

```
unsigned char x = 42; // x = 0b00101010
short y = -10; // y = 0b1111111111110110
unsigned int z = 0; // z = 0x00000000
```

จากตัวอย่างข้างต้น การประกาศตัวแปรและตั้งค่าเริ่มต้น ดังนี้

- x เป็นตัวแปรชนิดจำนวนเต็มชนิดมีเครื่องหมาย char มีค่าเริ่มต้นเท่ากับ 42
- y เป็นตัวแปรชนิดจำนวนเต็มชนิดมีเครื่องหมาย short มีค่าเริ่มต้นเท่ากับ -10
- z เป็นตัวแปรชนิดจำนวนเต็มชนิดไม่มีเครื่องหมาย unsigned long มีค่าเริ่มต้นเท่ากับ 0

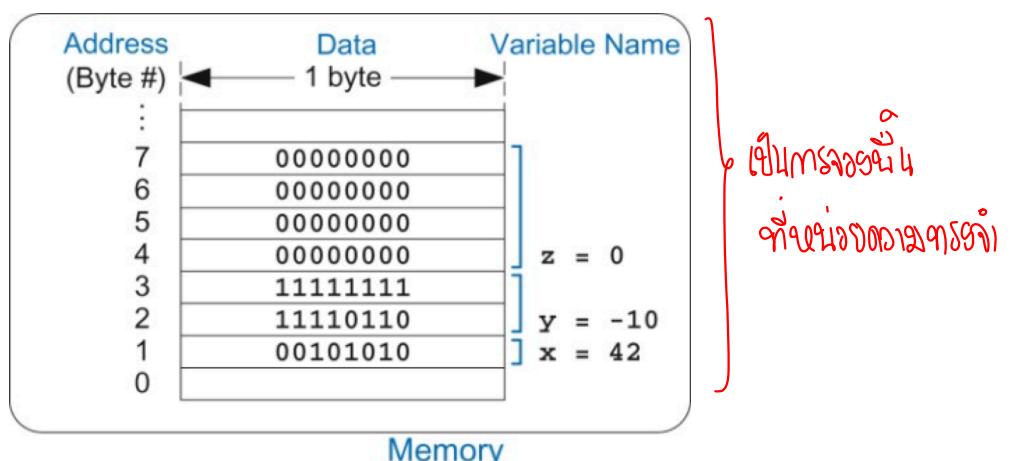


Figure 2.2: พื้นที่ในหน่วยความจำซึ่งบรรจุค่าเริ่มต้นของตัวแปรที่ได้ประกาศก่อนหน้า ที่มา: [Harris and Harris \(2013\)](#)

ก่อนที่โปรแกรมจะเริ่มทำงาน ระบบปฏิบัติการจะจองพื้นที่ของหน่วยความจำในลักษณะคล้ายกับรูปที่ 1.2 ซึ่งหมายเลขไบท์ (Byte Number) ในหน่วยความจำหลัก (Main Memory) จะเรียกว่าเป็นชั้นๆ ตั้งแต่หมายเลขไบท์ที่ 0 จากด้านล่างขึ้นไปหมายเลขไบท์ที่สูงขึ้นคล้ายกับตึกสูง แต่เริ่มต้นนับจากชั้นที่ 0 และแต่ละชั้นบรรจุข้อมูลได้เพียง 1 ไบท์ หากต้องการพื้นที่มากกว่า 1 ไบท์ วงจรสามารถนำพื้นที่ของชั้นถัดไปมาใช้งานด้วย ยกตัวอย่างเช่น ตัวแปร y และ z ในรูปที่ 1.2 ต้องการพื้นที่ 2 และ 4 ไบท์ ตามชนิดของตัวแปรในตารางที่ 1.1

ในบทนี้ ผู้อ่านจะได้ทำความเข้าใจกับข้อมูลชนิดจำนวนเต็มก่อน เพราะเป็นพื้นฐานของข้อมูลชนิดอื่นๆ และทำความเข้าใจกับเนื้อหาด้วยการปฏิบัติตามการทดลองที่ 1 ในภาคผนวก A

2.2 เลขจำนวนเต็มฐานสอง

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

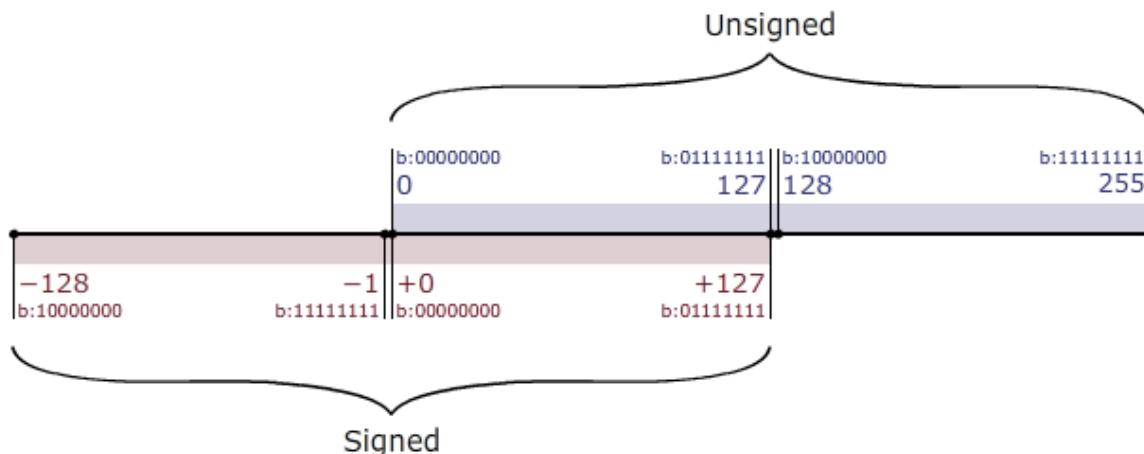


Figure 2.3: เส้นจำนวนเปรียบเทียบข้อมูลขนาด 8 บิตในรูปแบบของเลขจำนวนเต็มฐานสองชนิดไม่มีเครื่องหมาย (Unsigned) และมีเครื่องหมายแบบ 2-Complement ที่มา: stackoverflow.com

รูปที่ 1.3 เส้นจำนวนเปรียบเทียบข้อมูลขนาด 8 บิตในรูปแบบของเลขจำนวนเต็มฐานสองชนิดไม่มีเครื่องหมาย (Unsigned) และชนิดมีเครื่องหมาย (Signed) แบบ 2-Complement

2.2.1 ชนิดไม่มีเครื่องหมาย (Unsigned Integer)

เลขจำนวนเต็มฐานสองชนิดไม่มีเครื่องหมายในคอมพิวเตอร์ หมายความว่าการใช้นับจำนวนสิ่งของต่างๆ ซึ่งจำนวนจะมีค่าเริ่มต้นตั้งแต่ 0 จนถึงจำนวนที่ยาร์ดแวร์และซอฟต์แวร์รองรับ

unsigned char x = 42; // x = 0b00101010 → ดูดูด char จะมี 8 บิต
 unsigned short y = 10; // y = 0b000000000000001010
 unsigned int z = 0; // z = 0x00000000

จันธุ์ กันง่า เก้า ไบรท์ตัน format ไก่

นิยามที่ 2.2.1. กำหนดให้ เลขจำนวนเต็มฐานสองชนิดไม่มีเครื่องหมาย (Unsigned Integer) $X_{2,u}$ เป็น เลขฐานสองเขียนอยู่ในรูป

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

เมื่อ x_i คือค่า “1” หรือ “0” ในตำแหน่งหรือบิตที่ i และตำแหน่งขวามือสุดคือตำแหน่งหรือบิตที่ $i = 0$

2.2.1.1 การแปลงเลขฐานสองเป็นฐานสิบ

จากนิยามที่ 1.2.1 ค่าจำนวนเต็มฐานสิบ $X_{10,u}$ ของเลข $X_{2,u}$ สามารถคำนวณได้จาก

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

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

ตัวอย่างที่ 2.2.1. เมื่อ $n = 4$ บิต เลขจำนวนเต็มฐานสองชนิดไม่มีเครื่องหมาย $X_{2,u} = 1011_2$

ค่าฐานสิบของ $X_{2,u}$ คือ

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

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

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

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

ตัวอย่างที่ 2.2.2. เมื่อ $n = 8$ บิต เลขจำนวนเต็มฐานสองชนิดไม่มีเครื่องหมาย $X_{2,u} = 0000\ 1011_2$

ค่าฐานสิบของ $X_{2,u}$ คือ

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

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

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

เลขจำนวนเต็มฐานสองชนิดไม่มีเครื่องหมายขนาด $n = 8$ บิตจะมีค่าฐานสิบอยู่ในช่วง 0 ถึง $+255$

ตัวอย่างที่ 2.2.3. เมื่อ $n = 16$ บิต เลขจำนวนเต็มฐานสองชนิดไม่มีเครื่องหมาย $X_{2,u} = 0000\ 0000\ 0000\ 1011_2$

ค่าฐานสิบของ $X_{2,u}$ คือ

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

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

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

ดังนั้น เลขจำนวนเต็มฐานสองชนิดไม่มีเครื่องหมายขนาด $n = 16$ บิตจะมีค่าฐานสิบอยู่ในช่วง 0 ถึง $+65,535$

ตัวอย่างที่ 2.2.4. เมื่อ $n = 32$ บิต $X_{2,u} = 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 1011_2$

ค่าฐานสิบของ $X_{2,u}$ คือ

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

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

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

ดังนั้น เลขจำนวนเต็มฐานสองชนิดไม่มีเครื่องหมายขนาด $n = 32$ บิตจะมีค่าฐานสิบอยู่ในช่วง 0 ถึง $+4,294,967,295$

ผู้อ่านจะเห็นได้จากตัวอย่างที่ 1.2.1 ถึง 1.2.4 เลขฐานสิบค่าเดียวกัน เมื่อนำไปคำนวณในวงจรดิจิทัล ที่มีความยาวต่างกัน

2.2.1.2 การแปลงเลขฐานสิบเป็นฐานสอง

การแปลงเลขฐานสิบเป็นฐานสองชนิดไม่มีเครื่องหมาย (Unsigned) แบ่งเป็น 2 วิธี คือ การหาร และ การลบ

- การหาร เป็นวิธีที่เข้าใจง่ายและซับซ้อนน้อยกว่าวิธีการลบ

ตัวอย่างที่ 2.2.5. การแปลง 123 เป็นเลขฐานสองด้วยวิธีหาร

Table 2.2: การแปลงค่าฐานสิบเป็นเลขฐานสองแบบไม่มีเครื่องหมายความยาว $n=8$ บิต ด้วยการหาร

บิทที่	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 $123_{10} = 0111\ 1011_2$

- การลบ เป็นวิธีที่เข้าใจง่ายและซับซ้อนมากกว่าวิธีการหาร โดยผู้ใช้ควรจะจำเลขยกกำลังสองได้ เหมาะสำหรับผู้อ่านที่มีประสบการณ์กับการใช้เลขฐานสองบ่อยๆ

ตัวอย่างที่ 2.2.6. การแปลง 123 เป็นเลขฐานสองด้วยวิธีลบ

Table 2.3: การแปลงค่าฐานสิบเป็นเลขฐานสองแบบไม่มีเครื่องหมายความยาว $n=8$ บิต ด้วยการลบ

บิตที่ i	2^i	ผลลัพธ์ -2^i	ผลลัพธ์	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

ตั้งนี้ $X_{2,u}$ ของ $123_{10} = 0111\ 1011_2$ เช่นกัน

ตัวอย่างที่ 1.2.5 และ ตัวอย่างที่ 1.2.6 แสดงการแปลงเลขฐานสิบเป็นเลขฐานสองชนิดไม่มีเครื่องหมายทั้งสองวิธี

2.2.2 ชนิดมีเครื่องหมาย (Signed Integer) แบบ 2-Complement

เลขจำนวนเต็มฐานสองชนิดมีเครื่องหมายในคอมพิวเตอร์ หมายความว่าการใช้แทนข้อมูลที่มีค่าทั้งบวกและลบบนเส้นจำนวนตามที่ฮาร์ดแวร์และซอฟต์แวร์จะรองรับ

```
char x = -2; // x = 0b11111110
short y = -2; // y = 0b1111111111111110
int z = -2; // z = 0xFFFFFE
```

นิยามที่ 2.2.2. กำหนดให้ เลขจำนวนเต็มฐานสองชนิดมีเครื่องหมาย (Signed Integer) แบบ 2-Complement $X_{2,s}$ เขียนอยู่ในรูป

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

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

2.2.2.1 การแปลงเลขฐานสองเป็นฐานสิบ

การแปลงเลขจำนวนเต็มฐานสองชนิดมีเครื่องหมายแบบ 2-Complement จากนิยามที่ 1.2.2 ให้เป็นค่าฐานสิบสามารถทำได้โดย

$$X_{10,s} = -x_{n-1} \times 2^{n-1} + x_{n-2} 2^{n-2} + \dots + x_1 2^1 + x_0 2^0 \quad (2.16)$$

ดังนั้น ค่าฐานสิบ $X_{10,s}$ อยู่ในช่วง -2^{n-1} ถึง $+2^{n-1} - 1$
↓ ดูตัวอย่างที่ 2.2.7

ตัวอย่างที่ 2.2.7. เมื่อ $n = 4$ บิต เลขจำนวนเต็มฐานสองชนิดมีเครื่องหมายแบบ 2 Complement $X_{2,s} = 1011_2$ มีค่าฐานสิบเท่ากับเท่าไร

ค่าฐานสิบของ $X_{2,s}$ คือ

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

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

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

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

จากตัวอย่างที่ 1.2.7 เลขฐานสองความยาว $n=4$ บิตสามารถตีความแบบมีเครื่องหมายและแบบไม่มีเครื่องหมาย ตามสมการที่ 1.16 และ สมการที่ 1.2 ตามลำดับ ในตารางที่ 1.4

ตัวอย่างที่ 2.2.8. เมื่อ $n = 5$ บิต เลขจำนวนเต็มฐานสองแบบ 2 Complement $X_{2,s} = 11011_2$ มีค่าฐานสิบเท่ากับเท่าไร

ค่าฐานสิบของ $X_{2,s}$ คือ

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

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

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

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

เลขจำนวนเต็มฐานสองความยาว $n=5$ บิตสามารถตีความแบบมีเครื่องหมายและแบบไม่มีเครื่องหมาย ตามสมการที่ 1.16 และ 1.2 ตามลำดับ ในตารางที่ 1.5

Table 2.4: รูปแบบ (Pattern) ของเลขฐานสองขนาด $n=4$ บิตทั้งหมด $2^4=16$ แบบและการตีความหมายให้เป็นค่าฐานสิบแบบมีเครื่องหมาย 2-Complement และแบบไม่มีเครื่องหมาย

เลขฐานสอง $n=4$	$X_{10,s}$ ค่าฐานสิบ มีเครื่องหมาย	$X_{10,u}$ ค่าฐานสิบ ไม่มีเครื่องหมาย
$\begin{array}{r} 1000 \\ \oplus \\ 110 \\ \hline 111 = -8 \end{array}$	$1000 \rightarrow 111 + 1000$ -8	8
$\begin{array}{r} 1001 \\ \oplus \\ 110 \\ \hline 1000 = -7 \end{array}$	$1001 \rightarrow 101 + 1000$ -7	9
$\begin{array}{r} 1010 \\ \oplus \\ 110 \\ \hline 101 = -6 \end{array}$	$1010 \rightarrow 101 + 1000$ -6	10
$\begin{array}{r} 1011 \\ \oplus \\ 110 \\ \hline 110 = -5 \end{array}$	$1011 \rightarrow 110 + 1000$ -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

ตัวอย่างที่ 2.2.9. เมื่อ $n = 8$ บิต เลขจำนวนเต็มฐานสองชนิดมีเครื่องหมาย 2-Complement $X_{2,s} = 1111\ 1011_2$ มีค่าฐานสิบเท่ากับเท่าไร

char X = -5; // X = 0b11111011 (-5 ต่างๆ) 0101 ผลบวก +1
 ค่าฐานสิบของ $X_{2,s}$ คือ 1010
0101 1011

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

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

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

เลขจำนวนเต็มขนาด $n = 8$ บิตจะมีค่าฐานสิบอยู่ในช่วง -128 ถึง +127

เลขจำนวนเต็มฐานสองความยาว $n=8$ บิตสามารถตีความแบบมีเครื่องหมาย 2-Complement และแบบไม่มีเครื่องหมาย ตามสมการที่ 1.16 และ 1.2 ตามลำดับ คล้ายกับกรณี $n=5$ บิต ในตารางที่ 1.6
 ผู้อ่านจะสังเกตเห็นได้ว่า 1111 ด้านซ้ายของ $1111\ 1011_2$ ในตารางนี้ ทำหน้าที่เหมือน Sign bit ซึ่งสามารถประยุกต์ใช้กับเลขจำนวนเต็มฐานสองแบบ 2-Complement ได้ทุกความยาว n

Table 2.5: รูปแบบ (Pattern) ของเลขฐานสองขนาด $n=5$ บิตทั้งหมด $2^5=32$ แบบและการตีความหมายให้เป็นค่าฐานสิบแบบมีเครื่องหมาย 2-Complement และแบบไม่มีเครื่องหมาย

เลขฐานสอง $n=5$	$X_{10,s}$ ค่าฐานสิบ มีเครื่องหมาย	$X_{10,u}$ ค่าฐานสิบ ไม่มีเครื่องหมาย
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

ตัวอย่างที่ 2.2.10. เมื่อ $n = 16$ บิต เลขจำนวนเต็มฐานสองแบบ 2-Complement $X_{2,s} = 1111\ 1111\ 1111\ 1011_2$

short X = -5; // X = 0b1111111111111011

ค่าฐานสิบของ $X_{2,s}$ คือ

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

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

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

เลขจำนวนเต็มฐานสองชนิดมีเครื่องหมาย 2-Complement ขนาด $n = 16$ บิตจะมีค่าฐานสิบอยู่ในช่วง $-32,768$ ถึง $+32,767$

Table 2.6: รูปแบบ (Pattern) ของเลขฐานสองขนาด $n=8$ บิตทั้งหมด $2^8=256$ แบบและการตีความหมายให้เป็นค่าฐานสิบแบบมีเครื่องหมาย 2-Complement และแบบไม่มีเครื่องหมาย

เลขฐานสอง $n=8$	$X_{10,s}$ ค่าฐานสิบ มีเครื่องหมาย	$X_{10,u}$ ค่าฐานสิบ ไม่มีเครื่องหมาย
1000 0000	-128	128
...
1111 0111	-9	
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

ตัวอย่างที่ 2.2.11. เมื่อ $n = 32$ บิต เลขจำนวนเต็มฐานสองแบบ 2-Complement

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

```
int X = -5; // X = 0xFFFFFFFFB
```

ค่าฐานสิบของ $X_{2,s}$ คือ

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

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

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

เลขจำนวนเต็มฐานสองชนิดมีเครื่องหมาย 2-Complement ขนาด $n = 32$ บิตจะมีค่าฐานสิบอยู่ในช่วง $-2,147,483,648$ ถึง $+2,147,483,647$

ผู้อ่านจะเห็นได้จากตัวอย่างที่ 1.2.7 ถึง 1.2.11 เลขฐานสิบค่าเดียวกัน เมื่อนำไปคำนวณในวงจรดิจิทัล ที่มีความยาวต่างกัน จะมีการเติมบิตเครื่องหมาย (Sign bit) ทางด้านซ้ายเพื่อให้ขยายขึ้น เรียกว่า การขยายเครื่องหมาย (Sign Extension)

2.2.2.2 การแปลงเลขฐานสิบเป็นฐานสอง

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

$$X_{2,s} = [2^n + X_{10,s}]_2 \quad (2.32)$$

และ กรณี $X_{10,s} \geq 0$

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

เมื่อ $[X_{10}]_2$ คือการแปลงเลขฐานสิบชนิดไม่มีเครื่องหมายให้เป็นฐานสองชนิดไม่มีเครื่องหมาย [1.32](#)

Table 2.7: การแปลงค่าฐานสิบแบบมีเครื่องหมายให้เป็นเลขฐานสองความยาว $n=4$ บิต โดยแปลงให้เป็น ค่าฐานสิบโดยใช้สูตร $2^n + X_{10,s}$ กรณีที่ $X_{10,s} < 0$ ตามสมการที่ [1.32](#)

$X_{10,s} < 0$	แปลงเป็น $2^n + X_{10,s}$	เลขฐานสอง $X_{2,s} = [2^n + X_{10,s}]_2$
-8	$2^4 - 8 = 8$	1000_2
-7	$2^4 - 7 = 9$	1001_2
-6	$2^4 - 6 = 10$	1010_2
-5	$2^4 - 5 = 11$	1011_2
-4	$2^4 - 4 = 12$	1100_2
-3	$2^4 - 3 = 13$	1101_2
-2	$2^4 - 2 = 14$	1110_2
-1	$2^4 - 1 = 15$	1111_2
$X_{10,s} \geq 0$	แปลงเป็น	เลขฐานสอง $X_{2,s}$
0	0	0000_2
1	1	0001_2
2	2	0010_2
3	3	0011_2
4	4	0100_2
5	5	0101_2
6	6	0110_2
7	7	0111_2

2.2.3 ชนิดมีเครื่องหมาย (Signed Integer) แบบ Sign-Magnitude

นิยามที่ 2.2.3. กำหนดให้ เลขจำนวนเต็มฐานสองชนิดมีเครื่องหมาย (Signed Integer) แบบ Sign-Magnitude $X_{2,sm}$ เขียนอยู่ในรูป

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

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

การแปลงเลขจำนวนเต็มฐานสองแบบ Sign-Magnitude ให้เป็นค่าฐานสิบสามารถทำได้โดย

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

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

จากนิยามที่ 1.2.1 และนิยามที่ 1.2.3 เลขจำนวนเต็มฐานสองชนิดมีเครื่องหมาย แบบ Sign Magnitude สามารถเขียนใหม่ในรูปของเลขฐานสองแบบไม่มีเครื่องหมายที่ความยาว $n - 1$ บิต ดังนี้

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

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

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

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

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

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

เลขฐานสองความยาว $n=4$ บิตสามารถตีความแบบมีเครื่องหมายและแบบไม่มีเครื่องหมาย ตามสมการที่ 1.35, 1.16 และ 1.2 ตามลำดับ ในตารางที่ 1.8 ตรงกับรูปที่ 1.3 เช่นกัน

ตารางนี้ตรงกับรูปที่ 1.3 รูปแบบของ Sign-Magnitude นิยมใช้งานร่วมกับข้อมูลชนิดเลขทศนิยม ซึ่งจะกล่าวต่อไปในหัวข้อที่ 1.4 และหัวข้อที่ 1.5

0 → 1 ตัวนับบวกลบ + ลบ -

Table 2.8: ค่าฐานสิบแบบมีเครื่องหมาย Sign-Magnitude, แบบ 2-Complement, และแบบไม่มีเครื่องหมาย (Unsigned) ของเลขฐานสองความยาว $n=4$ บิตทั้งหมด $2^4=16$ แบบ

เลขฐานสอง $n=4$ บิต	$X_{10,sm}$ ค่าฐานสิบ Sign-Mag	$X_{10,s}$ ค่าฐานสิบ 2-Comp.	$X_{10,u}$ ค่าฐานสิบ Unsigned
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

2.3 คณิตศาสตร์เลขจำนวนเต็ม

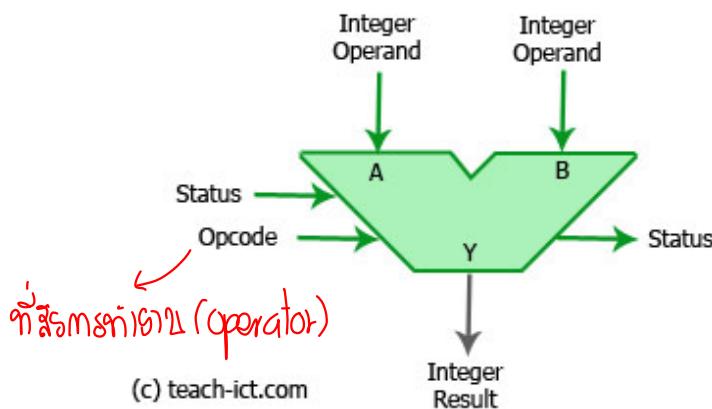


Figure 2.4: สัญลักษณ์ของ ALU (Arithmetic Logic Unit) สำหรับบวก/ลบเลขจำนวนเต็มขนาด n บิต ชนิดไม่มีเครื่องหมาย ที่มา: teach-ict.com

การคำนวณทางคณิตศาสตร์และตรรกศาสตร์ของเลขจำนวนเต็มฐานสองขนาด $n=8, 16, 32, 64$ และ 128 บิตจำเป็นต้องอาศัยจุดเด่นที่เรียกว่า วงจร ALU (Arithmetic Logic Unit) เพื่อการทำงานที่รวดเร็ว รูปที่ 1.4 แสดงสัญลักษณ์ของวงจร ALU สำหรับเลขจำนวนเต็มฐานสองขนาด n บิตชนิดไม่มีเครื่องหมาย ประกอบด้วย

- ขาข้อมูล A และ B ขนาด n บิต สำหรับนำข้อมูลฐานสอง โดยเริ่มนับจากบิตที่ 0 ถึงบิตที่ $n-1$

- สัญญาณ Opcode เพื่อสั่งการทำงาน เช่น บวก ลบ เป็นต้น
 - ผลลัพธ์ Y เป็นจำนวนเต็มชนิดไม่มีเครื่องหมายขนาด n บิต
 - สัญญาณ Status ประกอบด้วย
 - ขาเครื่องหมาย N (Negative) สำหรับเลขจำนวนเต็มชนิดมีเครื่องหมาย
 - ขา Z (Zero)=1 เพื่อบ่งบอกว่าผลลัพธ์ Y มีค่าเท่ากับศูนย์ทุกบิต
 - ขา Carry c_n สำหรับตัวทดบิตที่ n
 - ขา Overflow (V) เพื่อบ่งบอกความผิดพลาด
- ขาสัญญาณเหล่านี้จะบันทึกลงในรีจิสเตอร์สถานะ (Status Register) สำหรับให้วงจรและโปรแกรมเมอร์ตรวจสอบด้วยว่างจติกิทลและคำสั่งภาษาแอสเซมบลี ในบทที่ 4

2.3.1 ชนิดไม่มีเครื่องหมาย

เลขจำนวนเต็มฐานสองชนิดไม่มีเครื่องหมาย $X_{2,u}$ และ $Y_{2,u}$ ความยาว n บิต ตามนิยามที่ 1.2.1 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 $c_0=0$ เสมอและบวกเข้ากับ x_0+y_0 เพื่อให้ได้ค่า z_0 และเกิดตัวทด c_1 ในวงจรดิจิทัล การบวกเลขจำนวนเต็มฐานสองจำนวน 3 บิตเข้าด้วยกัน จะทำให้เกิดผลลัพธ์จำนวน 2 บิต เรียงติดกัน คือ

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

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

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

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

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

เมื่อ $\&$ คือ กระบวนการ AND และ $|$ คือ กระบวนการ OR วงจรบวกเลขชนิดไม่มีเครื่องหมายขนาด n บิต นี้สามารถตรวจจับการเกิดโอเวอร์โฟล์วได้โดย

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

2.3.1.1 การบวกเลขจำนวนเต็มชนิดไม่มีเครื่องหมายและการตรวจจับโอเวอร์โฟล์ว

การบวกเลขจำนวนเต็มชนิดไม่มีเครื่องหมาย 2 จำนวน ผลลัพธ์ที่ได้จะไม่มีเครื่องหมายด้วยเช่นกัน แต่ การบวกเลขขนาดใหญ่ที่เข้าใกล้ค่าสูงสุด สามารถเกิดความผิดพลาดได้ เรียกว่า การเกิดโอเวอร์โฟล์ว (Overflow) ในสมการที่ 1.43 ซึ่งเป็นผลสืบเนื่องจากการดิจิทัลที่สามารถประมวลผลได้จำกัด ตามจำนวนบิตข้อมูลสูงสุดที่ทำได้ ในตัวอย่างการแปลงเลขฐานสองเป็นฐานสิบที่ได้แสดงไปแล้ว ยกตัวอย่างเช่น การวนรอบหรือวนลูป (Loop) เพิ่มค่าอย่างต่อเนื่องโดยไม่ระวัง ตามประโยชน์ในภาษา C/C++ ประโยชน์ $i++$ หรือ $i = i + 1$ นี้ หาก i มีค่าเพิ่มขึ้นเรื่อยๆ จนถึงค่าสูงสุด การบวกเพิ่มอีก 1 ไปเรื่อยๆ โดยไม่มีการตรวจจับการเกิดโอเวอร์โฟล์วล่วงหน้า จะทำให้ค่าของ i กลายเป็นศูนย์ในที่สุด ซึ่งอาจทำให้เกิดผลร้ายตามมาอย่างรุนแรง

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

การบวกเลขขนาด 4 บิตแบบบวกมีเครื่องหมาย: $5 + 9 = 14$ พร้อมตัวทด และผลลัพธ์ถูกต้องเนื่องจากไม่เกิดโอเวอร์ฟอล์ (V=c_n=0)

	c ₄	c ₃	c ₂	c ₁	c ₀	Overflow
	0	0	0	1	0	V=0
X=5 +		0	1	0	1	+
Y=9		1	0	0	1	
Z=14		1	1	1	0	

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

	c ₄	c ₃	c ₂	c ₁	c ₀	Overflow
	1	1	1	1	0	V=c _n =1
X=7 +		0	1	1	1	+
Y=9		1	0	0	1	
Z=16		0	0	0	0	

สาเหตุของการเกิด Overflow เนื่องจากผลลัพธ์มีค่าอยู่นอกย่านที่เป็นไปได้ โดยสามารถตรวจสอบอย่างง่ายดายโดย $c_4 = 1$ (V: Overflow) เมื่อเกิดโอเวอร์ฟอล์ ผลลัพธ์ที่ได้จึงมีค่าไม่ถูกต้อง (Invalid)

2.3.1.2 การคูณเลขจำนวนเต็มชนิดไม่มีเครื่องหมาย

การคูณเลขจำนวนเต็มชนิดไม่มีเครื่องหมายมีความสำคัญต่อการคำนวณทางคณิตศาสตร์ต่างๆ และเป็นพื้นฐานของการคำนวณเลขชนิดอื่นๆ ซึ่งการคูณเลขมีความซับซ้อนมากกว่าการบวกเลขหลายเท่า ตามที่เล่นนี้จึงขอวิเคราะห์การคูณเลขจำนวนเต็มชนิดไม่มีเครื่องหมายให้ผู้อ่านแต่เพียงเบื้องต้น ดังนี้

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

วงจรคูณเลขชนิดที่ 1

การคูณเลขมีความซับซ้อนสูงจำเป็นต้องใช้วงจรดิจิทัลและเวลาในคำนวณ การคูณเลขที่ง่ายที่สุดคือ

การบวกเลขแล้ววนบวกซ้ำ ตามวงจรที่จะอธิบายในรูปที่ 1.5 และ 1.6 ต่อไปนี้

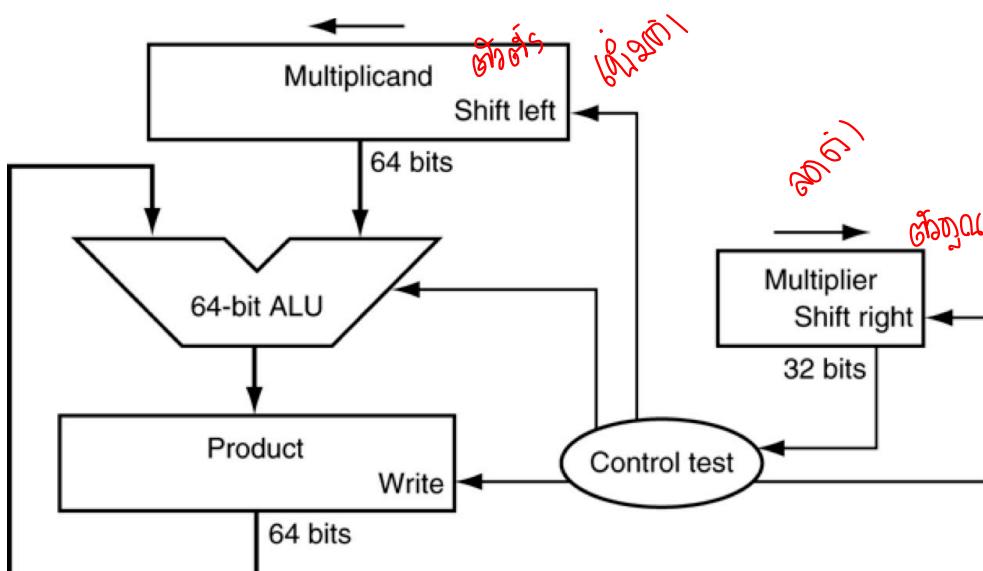


Figure 2.5: วงจรคูณเลขขนาด $n = 32$ บิต ชนิดที่ 1 โดยใช้ ALU ขนาด $2n = 64$ บิต และรีจิสเตอร์ตัวตั้งขนาด $2n = 64$ บิต (ที่มา: Patterson and Hennessy (2000))

วงจรในรูปที่ 1.5 ให้รีจิสเตอร์ตัวคูณ (Multiplier) บรรจุเลขจำนวนเต็มไม่มีเครื่องหมายขนาด n บิต ตัวตั้ง (Multiplicand) และผลคูณ (Product) จะมีขนาด $2n$ บิตทั้งคู่ มีการทำงานเป็นขั้นตอนดังนี้

1. ตั้งค่าเริ่มต้นของรีจิสเตอร์ทุกตัว (Initialize)
 - ตั้งค่ารีจิสเตอร์ตัวตั้ง (Multiplicand) ในรีจิสเตอร์ขนาด $2n$ บิต โดยให้บิตขวาสุดของตัวตั้งอยู่ที่บิต 0 ของรีจิสเตอร์
 - ตั้งค่ารีจิสเตอร์ตัวคูณ (Multiplier) ในรีจิสเตอร์ขนาด n บิต
 - ตั้งค่ารีจิสเตอร์ผลคูณ (Product) ขนาด $2n$ 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}) ตามวิธี 1.5

ผลคูณเลขขนาด 4 บิต ตัวตั้ง= 1010_2 (10_{10}) ตัวคูณ= 1101_2 (13_{10}) เท่ากับ 130_{10} ด้วยวิธีในรูปที่ 1.5

Table 2.9: ผลคูณเลขขนาด 4 บิต ตัวตั้ง= 1010_2 (10_{10}) ตัวคูณ= 1101_2 (13_{10}) เท่ากับ 130_{10} ด้วยวิธีในรูปที่ 1.5 หมายเหตุ Shift R คือ การเลื่อนบิตไปทางขวา

รอบ Iteration	ตัวตั้ง Multicand	ตัวคูณ Multiplier	ผลคูณ ₂ Product ₂	ผลคูณ ₁₀ Product ₁₀	กระทำ 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 R
1	0001 0100 ₂	0110 ₂	0000 1010 ₂	10 ₁₀	None
	0010 1000 ₂	0011 ₂	0000 1010 ₂	10 ₁₀	Shift R
2	0010 1000 ₂	0011 ₂	0011 0010 ₂	50 ₁₀	Add
	0101 0000 ₂	0001 ₂	0011 0010 ₂	50 ₁₀	Shift R
3	0101 0000 ₂	0001 ₂	1000 0010 ₂	130 ₁₀	Add
	1010 0000 ₂	0000 ₂	1000 0010 ₂	130 ₁₀	Shift R

วงจรคูณเลขชนิดที่ 2

วงจรในรูปที่ 1.6 ให้รีจิสเตอร์ตัวตั้ง (Multiplicand) และตัวคูณ (Multiplier) เป็นเลขจำนวนเต็มไม่ลบ เครื่องหมายขนาด n บิตทั้งคู่ และผลคูณ (Product) จะมีขนาด $2n$ บิต มีการทำงานเป็นขั้นตอนดังนี้

1. ตั้งค่าเริ่มต้นของรีจิสเตอร์ทุกตัว

- ตั้งค่ารีจิสเตอร์ตัวตั้ง (Multiplicand) ในรีจิสเตอร์ขนาด n บิต
- ตั้งค่ารีจิสเตอร์ตัวคูณ (Multiplier) ในรีจิสเตอร์ขนาด n บิต
- ตั้งค่ารีจิสเตอร์ผลคูณ (Product) ขนาด $2n$ บิตให้เป็น 0 ทุกบิต

2. ตรวจสอบค่าบิตที่ 0 (ขวาสุด) ของรีจิสเตอร์ตัวคูณ

- หากมีค่าเป็น 1 บวกค่าตัวตั้งกับค่าผลคูณเข้าด้วยกัน (Action = Add)
- หากมีค่าเป็น 0 ไม่ต้องบวก (Action = None)

3. เลื่อน (Shift) ข้อมูลในรีจิสเตอร์

- เลื่อนรีจิสเตอร์ตัวคูณและรีจิสเตอร์ผลคูณไปทางขวา 1 บิต โดยป้อน 0 เข้าทางซ้ายของรีจิสเตอร์ตัวคูณ เพื่อไปแทนที่บิตที่ถูกเลื่อนไปทางขวา แต่ป้อนค่า c_n กลับเข้ามาทางซ้ายของรีจิสเตอร์ผลคูณ เพื่อไปแทนที่บิตที่ถูกเลื่อนไปทางขวา

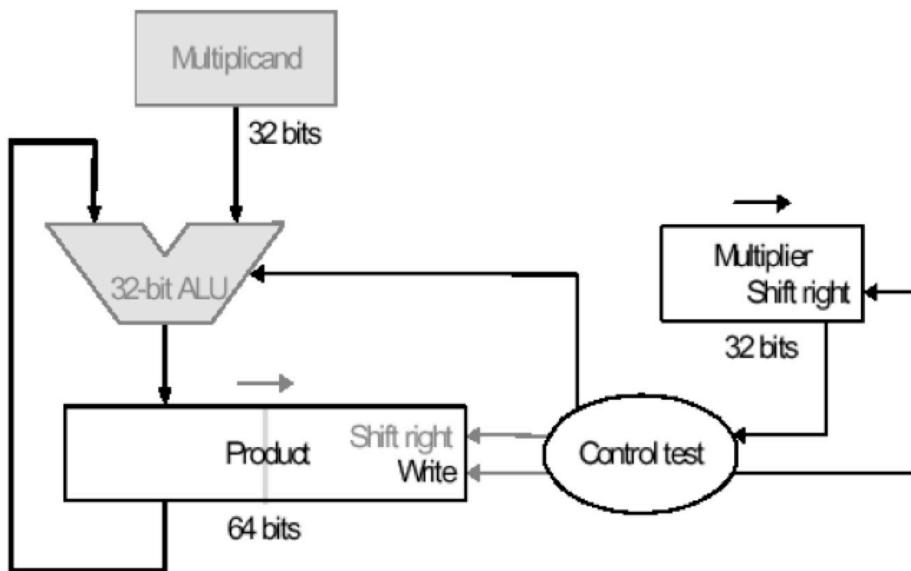


Figure 2.6: วงจรคูณเลขขนาด $n = 32$ บิต ชนิดที่ 2 โดยใช้ ALU ขนาด $n = 32$ บิต และรีจิสเตอร์ตัวตั้งขนาด $n = 32$ บิต (ที่มา: Patterson and Hennessy (2000))

4. กลับไปทำข้อ 2 จนครบ n รอบ

ตัวอย่างที่ 2.3.4. จงคูณ 1010_2 (10_{10}) ด้วย 1101_2 (13_{10}) ตามวิธีที่ 1.6

ผลคูณเลขขนาด 4 บิต ตัวตั้ง= 1010_2 (10_{10}) ตัวคูณ= 1101_2 (13_{10}) เท่ากับ 130_{10} ด้วยวิธีในรูปที่ 1.6

Table 2.10: ผลคูณเลขขนาด 4 บิต ตัวตั้ง= 1010_2 (10_{10}) ตัวคูณ= 1101_2 (13_{10}) เท่ากับ 130_{10} ด้วยวิธีในรูปที่ 1.6 หมายเหตุ Shift R คือ การเลื่อนบิตไปทางขวา

รอบ Iteration	ตัวตั้ง Multicand	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

จะเห็นได้ว่าที่รอบที่ 3 บิตที่เกิดขึ้นทางซ้ายสุดของผลคูณ จะถูกป้อนกลับเข้ามาทางซ้าย เพื่อให้ได้ผลลัพธ์ที่ถูกต้อง

วงจรคุณเลขจำนวนเต็มทั้งสองชนิดนี้ เป็นแค่ตัวอย่างเพื่อให้ผู้อ่านเข้าใจ การแลกเปลี่ยน (Trade off) ระหว่าง จำนวนรอบ (ระยะเวลา) กับ ความซับซ้อนของวงจร ซึ่งในทางปฏิบัติวงจรคุณจะมีความซับซ้อนมากกว่าแต่ ใช้ระยะเวลาสั้นกว่า และสามารถออกแบบให้ทำงานแบบไปปีลีน์ด้วย

2.3.2 ชนิดมีเครื่องหมายแบบ 2-Complement

เลขจำนวนเต็มฐานสองชนิดมีเครื่องหมายแบบ 2-Complement $X_{2,s}$ และ $Y_{2,s}$ ความยาว n บิต ตามนิยามที่ 1.2.2 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	

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

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

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

ผลลัพธ์ของการบวกเลขจำนวน 3 บิต สามารถคำนวณได้จากการ Full Adder ดังนี้

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

เมื่อ \oplus คือ กระบวนการ Exclusive-OR

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

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

การเกิดโอเวอร์โฟล์ของ การบวกเลขชนิดมีเครื่องหมาย 2-Complement ได้โดย

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

ขาสัญญาณเหล่านี้จะบันทึกลงในรีจิสเตอร์สถานะ สำหรับให้โปรแกรมเมอร์ตรวจสอบ ด้วยภาษาแอสเซมบลี ในบทที่ 4

2.3.2.1 การบวก/ลบเลขจำนวนเต็มชนิดมีเครื่องหมายและการตรวจจับโอเวอร์โฟล์

การบวก/ลบเลขจำนวนเต็มชนิดมีเครื่องหมาย 2 จำนวน ผลลัพธ์ที่ได้จะไม่มีเครื่องหมายด้วยเช่นกัน แต่ การบวก/ลบเลขขนาดใหญ่ที่เข้าใกล้ค่าสูงสุดหรือค่าต่ำสุด สามารถเกิดความผิดพลาดได้ เรียกว่า การเกิด โอเวอร์โฟล์ (Overflow) ในสมการที่ 1.47 ซึ่งเป็นผลสืบเนื่องจากการจัดจิทัลที่สามารถประมวลผลได้ จำกัด ตามจำนวนบิตข้อมูลสูงสุดที่ทำได้ ยกตัวอย่าง เช่น การวนรอบหรือวนลูป (Loop) เพิ่มค่าอย่างต่อเนื่องโดยไม่ระวัง ตามประโยคในภาษา C/C++ ประโยค $i++$ หรือ $i = i + 1$ นี้ หาก i มีค่าเพิ่มขึ้นเรื่อยๆ จนถึงค่าสูงสุด การบวกเพิ่มอีก 1 ไปเรื่อยๆ โดยไม่มีการตรวจจับการเกิดโอเวอร์โฟล์ล่วงหน้า จะทำให้ค่าของ i กลับเป็นค่าลบในที่สุด ซึ่งอาจทำให้เกิดผลลัพธ์ตามมาอย่างรุนแรง

ตัวอย่างที่ 2.3.5. จงคำนวณหาค่าของ $7 + 3$ ด้วยเลขจำนวนเต็มชนิดมีเครื่องหมาย แบบ 2-Complement ขนาด 4 บิต ดังนี้ ในเครื่องคอมพิวเตอร์ขนาด 4 บิต สามารถคำนวณได้ดังนี้

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

เนื่องจากฮาร์ดแวร์คำนวณ $c_4 \text{ xor } c_3 = 0 \text{ xor } 1 = 1$ ทำให้ตรวจได้ว่าเกิด Overflow ทำให้ผลลัพธ์มีค่าไม่ถูกต้อง

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

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

เนื่องจากฮาร์ดแวร์คำนวณ $c_4 \text{ xor } c_3 = 1 \text{ xor } 1 = 0$ ทำให้ตรวจได้ว่าไม่เกิด Overflow ทำให้ผลลัพธ์มีค่าถูกต้อง

ตัวอย่างที่ 2.3.7. จงคำนวณหาค่าของ $-3 - 6$ ด้วยเลขจำนวนเต็มชนิดมีเครื่องหมาย แบบ 2-Complement ขนาด 4 บิต $-3 - 6 = (-3) + (-6)$ ดังนั้น ในเครื่องคอมพิวเตอร์ขนาด 4 บิต สามารถคำนวณได้แต่ผลลัพธ์ กลับไม่ถูกต้องเนื่องจากเกิดโอเวอร์โฟล์ ดังนี้

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

เนื่องจากหารด้วยร์คำนวณ $c_4 \text{ xor } c_3 = 1 \text{ xor } 0 = 1$ ทำให้ตรวจได้ว่าเกิด Overflow ทำให้ผลลัพธ์มีค่าไม่ถูกต้อง

2.4 เลขทศนิยมฐานสองชนิดจุดคงที่ (Binary Fixed Point)

เลขจำนวนจริงแบ่งเป็นชนิดจุดทศนิยมคงที่ (Fixed-Point Real Number) และ ชนิดจุดทศนิยมลอยตัว (Floating-Point Real Number) ในหัวข้อนี้ ผู้อ่านจะได้ทำความเข้าใจเรื่องรูปแบบ (Representation) ของเลขทศนิยมฐานสองชนิดจุดคงที่ ซึ่งนิยมใช้ในตัวประมวลผลสัญญาณดิจิทัล (Digital Signal Processor) ส่วนการบวกและลบเลขทศนิยมฐานสองชนิดนี้จะใช้หลักการเดียวกับเลขจำนวนเต็มชนิด Sign-Magnitude

นิยามที่ 2.4.1. กำหนดให้ F_2 เป็นเลขทศนิยมฐานสองชนิด Signed Magnitude เก็บอยู่ในรูป

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

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

โดย F_{10} คือ ค่าฐานสิบของเลขทศนิยมฐานสองชนิดจุดคงที่ F_2 สามารถคำนวณได้จาก

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

หรือ

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

ตัวอย่างที่ 2.4.1. จะแปลงเลขฐานสิบต่อไปนี้เป็นเลขทศนิยมฐานสองชนิดจุดคงที่ $m=n=2$ บิท

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

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

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

ผู้อ่านสามารถสามารถเขียนเลขทศนิยมฐานสองชนิดจุดคงที่ได้ตามรูปแบบนี้

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

ค่าทศนิยม (Y_2) มีความยาว m บิท เก็บเป็นสัญลักษณ์ได้ดังนี้

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

เมื่อจำนวนบิทหลังจุดทศนิยมเพิ่มขึ้น ความละเอียดจะเพิ่มขึ้นตามตัวอย่างต่อไปนี้

ตัวอย่างที่ 2.4.2. จงแปลงเลขทศนิยมฐานสองชนิดจุดคงที่ต่อไปนี้ให้เป็นเลขฐานสิบ

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

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

$$0.111111_2 = 0.5 + 0.25 + 0.125 + 0.0625 + 0.03125 + \dots = 0.99_{10} \quad (2.58)$$

เมื่อจำนวนบิทที่เป็น 1 เพิ่ม ค่าฐานสิบของเลขทศนิยมฐานสองนี้จะถูกลุ้นเข้าหา 1.0 ยิ่งขึ้น ในขณะที่ การบวก/ลบ และการคูณเลขชนิดจุดคงที่มีความคล้ายกับเลขจำนวนเต็มชนิด Sign-Magnitude โดยรายละเอียดจะกล่าวในหัวข้อถัดไป

2.5 เลขทศนิยมฐานสองชนิดจุดลอยตัว (Binary Floating Point)

เลขทศนิยมฐานสองชนิดชนิดจุดลอยตัวหมายความว่าสำหรับข้อมูลที่มีพิสัย (Range) กว้างและเลขทศนิยมที่ต้องการความละเอียดสูง สำหรับการคำนวณทางวิทยาศาสตร์ (Scientific) ดังนี้

- -2.34×10^{56} ซึ่งเขียนอยู่ในลักษณะที่น้อมัลไล์ซ์ (normalize) แล้ว
- $+0.002 \times 10^{-4}$ ซึ่งจะต้องน้อมัลไล์ซ์ต่อไปเป็น $+2.000 \times 10^{-7}$
- $+987.02 \times 10^9$ ซึ่งจะต้องน้อมัลไล์ซ์ต่อไปเป็น $+9.8702 \times 10^{11}$

นิยามที่ 2.5.1. เลขทศนิยมชนิดจุดลอยตัวฐานสองที่อยู่ในรูปน้อมัลไล์ซ์ ประกอบด้วย 3 ส่วน คือ บิทเครื่องหมาย (Sign bit: s) ค่านัยสำคัญ (Significant: S_2) และค่ายกกำลัง (Exponent: E_2) มีลักษณะดังนี้

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

เลขยกกำลังเป็นเลขจำนวนเต็มฐานสอง sign-magnitude ความยาว n บิท ดังนี้

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

เมื่อ e_i แต่ละบิทมีค่า “1” หรือ “0” ในตำแหน่งที่ i s คือ Sign bit และ n คือ จำนวนบิทซึ่งกำหนดไว้ก่อนจะออกแบบวงจร

จากนิยามที่ 1.59 ค่านัยสำคัญ (Significant) S_2 ความยาว $m+1$ บิท สามารถเขียนใหม่ได้ดังนี้

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

ซึ่งมีความสำคัญต่อรูปแบบการเขียน เนื่องจากว่าจะต้องทำการน้อมัลไล์ซ์ผลลัพธ์ที่ได้จากการคำนวณเสมอ ค่านัยสำคัญที่น้อมัลไล์ซ์ข้างต้นสามารถคำนวณหาค่าฐานสิบ ได้ดังนี้

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

ตัวอย่างที่ 2.5.1. จงแปลงเลขทศนิยมฐานสองชนิดจุดลอยตัวที่น้อมัลไล์ซ์แล้วขนาด $n=4$ บิทให้เป็นเลขทศนิยมฐานสิบ

วิธีทำ
 $(-1)^1 \times 1.0101_2 \times 2^3$

$$\begin{aligned} & -1010.1 \times 2^3 \\ & -8+2+0.5 \\ & -10.5 \end{aligned}$$

1. ปรับจุดทศนิยม เพื่อให้เป็นเลขฐานสองชนิด Sign-Magnitude และเลขยกกำลังเท่ากับ 0

$$-1010.1_2 \times 2^0$$

2. แปลงค่าเลขฐานสองชนิดที่เลื่อนตำแหน่งแล้วให้เป็นฐานสิบ

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

normalize $\xrightarrow{a \rightarrow b}$ ฐานสิบ

ตัวอย่างที่ 2.5.2. จงแปลงเลขทศนิยมฐานสองชนิดจุดลอยตัวที่น่อมัลไล์ซ์แล้ว $n=4$ บิตให้เป็นเลขทศนิยมฐานสิบ

วิธีทำ

$$(-1)^0 \times 1.1010_2 \times 2^{-3}$$

1. ปรับจุดทศนิยม เพื่อให้เป็นเลขฐานสองชนิด Sign-Magnitude และเลขยกกำลังเท่ากับ 0

$$+0.001101_2 \times 2^0$$

2. แปลงค่าเลขฐานสองชนิดที่เลื่อนตำแหน่งแล้วให้เป็นฐานสิบ

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

$$= 0.125 + 0.0625 + 0.015625$$

$$= 0.203125_{10}$$

2.5.1 การบวกเลขทศนิยมฐานสองชนิดจุดลอยตัว

เพื่อให้ผู้อ่านเข้าใจการบวกเลขทศนิยมฐานสองชนิดจุดลอยตัว โดยใช้ตัวอย่างจากเลขทศนิยมฐานสิบ ก่อน แล้วจึงนำขั้นตอนที่เหมือนกันไปประยุกต์ใช้กับเลขทศนิยมฐานสองในตัวอย่างต่อไป

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

วิธีทำ

1. เลื่อนตำแหน่งจุดทศนิยมและปรับเลขยกกำลังที่มีค่าน้อยกว่า ให้ตรงกับเลขยกกำลังของเลขที่มีค่ามากกว่า

$$9.999 \times 10^1 + 0.016 \times 10^1$$

2. บวกค่านัยสำคัญที่เลื่อนตำแหน่งแล้ว

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

3. น่อมัลไล์ซ์ค่าผลลัพธ์และตรวจเช็คการเกิดโอเวอร์โฟล์และอันเดอร์โฟล์

$$1.0015 \times 10^2$$

4. ปัดค่าให้เหลือ 4 หลักและอาจต้องน่อมัลไล์ซ์เมื่อจำเป็น

$$1.002 \times 10^2$$

จะเห็นได้ว่า ผลลัพธ์ที่ได้จะเกิดความคลาดเคลื่อนจากค่าที่ควรจะเป็น เนื่องจากมีการจำกัดจำนวนดิจิทของผลลัพธ์ให้เหลือ 4 หลักตามโจทย์

ตัวอย่างที่ 2.5.4. จงบวกเลขทศนิยมฐานสองชนิดจุดลอยตัวที่น่อมัลไล์ซ์แล้ว 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. เลื่อนตำแหน่งจุดทศนิยมและปรับเลขยกกำลังที่มีค่าน้อยกว่า ให้ตรงกับเลขยกกำลังของเลขที่มีค่ามากกว่า

$$1.000_2 \times 2^{-1} - 0.110_2 \times 2^{-2}$$

$$\begin{array}{r} 1.000 \\ - 0.110 \\ \hline 0.010 \end{array}$$

$$0.001 \rightarrow \text{normaliz.}$$

$$0.001 \times 2^0 = 0.001$$

Chapter 2. ข้อมูลและคณิตศาสตร์ในคอมพิวเตอร์ (Computer Data and Arithmetic)

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

2. บวกค่านัยสำคัญที่เลื่อนตำแหน่งแล้ว

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

3. น้อมลัลเลซ์ค่าผลลัพธ์และตรวจเช็คการเกิดโอเวอร์ฟล็อวและอันเดอร์ฟล็อว

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

4. ปดค่าให้เหลือ 4 บิตและอาจต้องน้อมลัลเลซ์เมื่อจำเป็น

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

2.5.2 การคูณเลขทศนิยมฐานสองชนิดจุดลอยตัว

ในทำนองเดียวกับการบวกเลข เพื่อให้ผู้อ่านเข้าใจการคูณเลขทศนิยมฐานสองชนิด Floating-Point เราจะใช้ตัวอย่างจากการคูณเลขทศนิยมฐานสิบก่อน และวิธีคำนวณที่เหมือนกันไปประยุกต์ใช้กับเลขทศนิยมฐานสองในตัวอย่างต่อไป

ตัวอย่างที่ 2.5.5. จงคูณเลขทศนิยมฐานสิบชนิดจุดลอยตัวที่น่อมัลไล์ช์ ดังต่อไปนี้ สมมติว่ามีเลขฐานสิบขนาด 4 หลัก (4-digit) $(1.110_{10} \times 10^{10}) \times (9.200_{10} \times 10^{-5})$

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

2. คูณค่านัยสำคัญเข้าด้วยกัน

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

3. น่อมัลไล์ช์ค่าผลลัพธ์ และตรวจเช็คการเกิดโอเวอร์โฟล์และอันเดอร์โฟล์

$$1.0212 \times 10^6$$

4. ปัดค่าให้เหลือ 4 หลักและอาจต้องน่อมัลไล์ช์เมื่อจำเป็น

$$1.021 \times 10^6$$

5. ปรับเครื่องหมายของผลลัพธ์ให้ถูกต้องจากตัวตั้งและตัวคูณ

$$+1.021 \times 10^6$$

จะเห็นได้ว่า ผลลัพธ์ที่ได้จะเกิดความคลาดเคลื่อนจากค่าที่ควรจะเป็น เนื่องจากมีการจำกัดจำนวนดิจิทของผลลัพธ์ให้เหลือ 4 หลักตามโจทย์

ตัวอย่างที่ 2.5.6. จงคูณเลขทศนิยมฐานสองชนิดจุดลอยตัวขนาด 4 บิต (4-bit) ที่น่อมัลไล์ช์ 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

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

3. น่อมัลไล์ช์ค่าผลลัพธ์ และตรวจเช็คการเกิดโอเวอร์โฟล์และอันเดอร์โฟล์

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

4. ปัดค่าให้เหลือ 4 หลักและอาจต้องน่อมัลไล์ช์เมื่อจำเป็น

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

5. ปรับเครื่องหมายของผลลัพธ์ให้ถูกต้องจากตัวตั้งและตัวคูณ

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

2.6 เลขทศนิยมฐานสองชนิดจุดลอยตัวมาตรฐาน IEEE 754

มาตรฐานของเลขทศนิยมฐานสองชนิดจุดลอยตัวได้ถูกกำหนดโดย IEEE (Institute of Electrical and Electronics Engineers) เรียกว่า มาตรฐาน IEEE 754 ในปี ค.ศ. 1985 เพื่อให้โปรแกรมคอมพิวเตอร์สามารถคำนวณค่าเลขทศนิยมฐานสองบนเครื่องที่ใช้ชิปปิค์เก็ตได้แล้วให้ผลลัพธ์ตรงกัน ปัจจุบันนี้ได้รับการยอมรับอย่างแพร่หลาย มีสองรูปแบบคือ

- 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 IEEE 754 คือ ปี ค.ศ. 2019 [รายละเอียดเพิ่มเติม](#)

ตัวอย่างการประกาศและตั้งค่าตัวแปรที่ใช้มาตรฐานนี้

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

เมื่อผู้อ่านทำความเข้าใจทฤษฎีและตัวอย่างการคำนวณเบื้องต้นแล้ว ผู้อ่านสามารถทำการทดลองเพิ่มเติมใน การทดลองที่ 1 ภาคผนวก A ข้อมูลและคณิตศาสตร์ในคอมพิวเตอร์ หัวข้อที่ A.2

2.6.1 รูปแบบของเลข IEEE 754

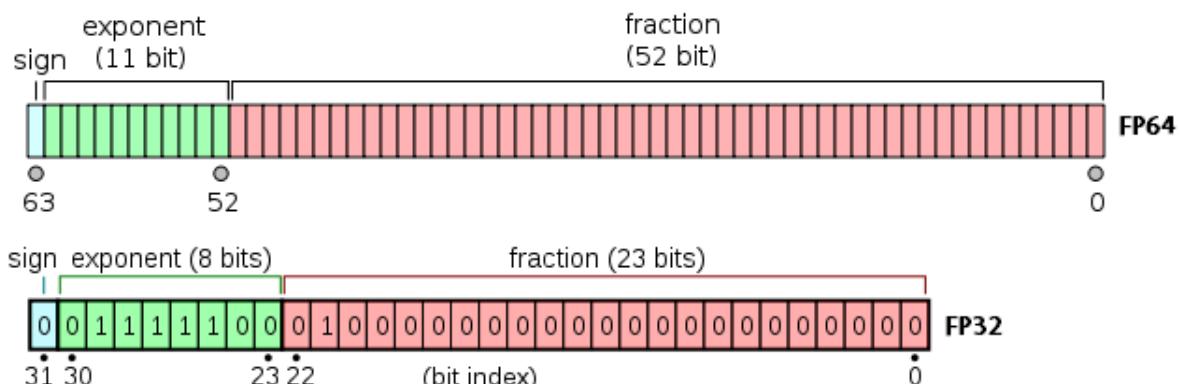


Figure 2.7: โครงสร้างของเลขทศนิยมฐานสองชนิดจุดลอยตัวตามมาตรฐาน IEEE 754 Double-Precision และ Single-Precision ที่มา: pybugs.com

นิยามที่ 2.6.1. เลขทศนิยมฐานสองชนิดจุดลอยตัวที่อนนัลไลซ์แล้ว สามารถเขียนอยู่ในรูปของเลขฐานสองต่อไปนี้

$$F_{2, IEEE} = [s][E_2][Y_2] \quad (2.63)$$

โดยค่าทศนิยม (Fraction): Y_2 มีความยาว $m=23$ และ 51 บิต ตามชนิด Single Precision และ Double Precision ตามลำดับ สามารถเขียนเป็นสัญลักษณ์คล้ายกับเลขทศนิยมฐานสองชนิดจุดคงที่ ดังนี้

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

ดังนั้น ค่านัยสำคัญ (Significand: S_2) มีความยาว $m + 1 = 24$ หรือ 52 บิต โดยมีรูปแบบตามสมการที่ 1.61 ในหัวข้อที่ 1.5

ค่ายกกำลัง เป็นเลขจำนวนเต็มชนิดไม่มีเครื่องหมาย ความยาว $n=8$ และ 11 บิต ขึ้นกับชนิด Single Precision และ Double Precision ตามลำดับ โดยมีลักษณะคล้ายกับเลขทศนิยมฐานสองชนิดจุดลอยตัว ในสมการที่ 1.60 แต่ต่างกันที่เลขยกกำลังของ IEEE 754 มีค่าเป็นเลขจำนวนเต็มไม่มีเครื่องหมาย

$$E_2 = [e_{n-1} e_{n-2} \dots e_0] \quad (2.65)$$

โดยรายละเอียดเป็นดังนี้

- s : บิตเครื่องหมาย (Sign bit)
 - 0 ==> มากกว่าหรือเท่ากับศูนย์ (non-negative),
 - 1 ==> น้อยกว่าศูนย์ (negative)
- เลขนัยสำคัญ (Significand= 1+Fraction): $1.0 \leq |\text{Significand}| < 2.0$
- จากสมการที่ 1.65 เลขยกกำลัง (Exponent: E_2) = เลขยกกำลังจริง (True Exponent: E_{true}) + ไบเออส (E_{bias})

การบวกค่า E_{bias} มีวัตถุประสงค์เพื่อปรับให้ค่ายกกำลังเป็นเลขที่ไม่มีเครื่องหมาย (unsigned) โดย

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

ดังนั้น เลขยกกำลังจริงจึงสามารถคำนวณได้โดย

$$E_{true} = E_2 - E_{bias} \quad (2.66)$$

เลขทศนิยมฐานสองชนิดจุดลอยตัวตามมาตรฐาน IEEE 754 ในสมการที่ 1.63 สามารถแปลงเป็นค่าเลขทศนิยมฐานสิบ ได้ดังนี้

$$F_{10, IEEE} = \pm \left[1 + \frac{y_{-1}}{2} + \frac{y_{-2}}{4} + \frac{y_{-3}}{8} + \dots + \frac{y_{-m}}{2^m} \right] \times 2^{(E_2 - E_{bias})} \quad (2.67)$$

ตัวอย่างที่ 2.6.1. จงหาค่าทศนิยมฐานสิบของเลข FP-32 ที่เติมในรูปที่ 1.7 คือ เลขทศนิยมมาตรฐาน IEEE 754 ชนิด Single Precision ดังต่อไปนี้
 $[0][011\ 1110\ 0][010\ 0000\ 0000\ 0000\ 0000]$

$$F_{10,IEEE} = 1.01 \times 2^{(124-127)} \quad (2.68)$$

$$= 1.01 \times 2^{-3} \quad (2.69)$$

$$= 0.00101_2 \quad (2.70)$$

$$= 2^{-3} + 2^{-5} \quad (2.71)$$

$$= \frac{1}{2^3} + \frac{1}{2^5} \quad (2.72)$$

$$= 0.125_{10} + 0.03125_{10} = 0.15625_{10} \quad (2.73)$$

ตัวอย่างที่ 2.6.2. จงแปลงเลข -0.75_{10} เป็นเลขทศนิยมฐานสองชนิดจุดลอยตัวตามมาตรฐาน IEEE 754 ทั้งสองชนิด

วิธีทำ

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

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

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

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

ทำการน้อมัลไลร์ ตามสมการที่ 1.59

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

ดังนั้น บิตเครื่องหมาย $s = 1$

$$\text{ค่าทศนิยม } Y_2 = 100\ 0000\ 0000\ 0000\ 0000_2$$

$$\text{ค่ายกกำลัง } E_2 = -1 + E_{bias}$$

โดยชนิด Single ค่ายกกำลัง (8 บิต): $E_2 = -1 + 127 = 126$ หรือ $E_2 = 0111\ 1110_2$

โดยชนิด Double ค่ายกกำลัง (11 บิต): $E_2 = -1 + 1023 = 1022$ หรือ $E_2 = 011\ 1111\ 1110_2$

ดังนั้น -0.75_{10} เที่ยวนในรูปของเลขทศนิยมชนิดจุดลอยตัวตามมาตรฐาน IEEE 754 ชนิด

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

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

ตัวอย่างที่ 2.6.3. เลขทศนิยมฐานสองชนิดจุดลอยตัวมาตรฐาน IEEE 754 ชนิด Single-Precision ต่อไปนี้ มีค่าเท่าไรในฐานสิบ

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

วิธีทำ

แปลงจากเลขฐานสิบหกให้เป็นฐานสองและองค์ประกอบต่างๆ ได้ดังนี้

$$s=[1][E_2=100\ 0000\ 1][Y_2=010\ 0000\ 0000\ 0000\ 0000]_2$$

จะพบว่าบิทเครื่องหมาย $s = 1$

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

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

$$F_{10,IEEE} = (-1)^1 \times (1 + .01_2) \times 2^2 \quad (2.74)$$

$$= (-1)^1 \times (1.01_2) \times 2^2 \quad (2.75)$$

$$= (-1)^1 \times (101.0_2) \quad (2.76)$$

$$= (-1) \times 5.0 \quad (2.77)$$

$$= -5.0_{10} \quad (2.78)$$

ตัวอย่างที่ 2.6.4. เลขทศนิยมฐานสองชนิดจุดลอยตัวมาตรฐาน IEEE 754 ชนิด Single-Precision ต่อไปนี้ มีค่าเท่าไรในฐานสิบ

$$4\ 1\ B\ 0\ 0\ 0\ 0\ 0_{16} = [0100\ 0001\ 1011\ 0000\ 0000\ 0000\ 0000]_2$$

วิธีทำ

แปลงจากเลขฐานสิบหกให้เป็นฐานสองและองค์ประกอบต่างๆ ได้ดังนี้

$$s=[0][E_2=100\ 0001\ 1][Y_2=011\ 0000\ 0000\ 0000\ 0000]_2$$

จะพบว่าบิทเครื่องหมาย $s = 0$

$$\text{ค่ายกกำลังจริง } E_{true} = 1000\ 0011_2 - E_{bias} = 131 - 127 = 4$$

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

$$F_{10,IEEE} = (-1)^0 \times (1 + .011_2)2^4 \quad (2.79)$$

$$= (-1)^0 \times (1.011_2) \times 2^4 \quad (2.80)$$

$$= (-1)^0 \times (10110.0_2) \quad (2.81)$$

$$= (+1) \times (16 + 4 + 2) \quad (2.82)$$

$$= 22.0_{10} \quad (2.83)$$

2.6.2 ค่าสูงสุด ต่ำสุดและค่าอื่นๆ ของเลข Floating Point

การคำนวณในเครื่องคอมพิวเตอร์สามารถรองรับการคำนวณที่หลากหลาย โดยใช้เลขทศนิยมฐานสองชนิดจุดลอยตัวตามมาตรฐาน IEEE 754 แต่เนื่องจากเลขจำนวนจริงในทางคณิตศาสตร์มีจำนวนอนันต์ (Infinite) เลขทศนิยมฐานสองตามมาตรฐาน IEEE 754 จึงไม่สามารถรองรับได้ทั้งหมด และมีข้อจำกัดทางคณิตศาสตร์ สรุปในตารางที่ 1.11

Table 2.11: ตารางสรุปความแตกต่างระหว่างเลขทศนิยมฐานสองชนิดจุดลอยตัวตามมาตรฐาน IEEE 754 ชนิด Single Precision โดย E_2 คือ ค่ายกกำลัง และ Y_2 คือ ทศนิยมซึ่งเป็นส่วนประกอบของค่านัยสำคัญ (\times หมายถึง บิทข้อมูลมีค่าเท่ากับ '0' หรือ '1')

ลำดับที่	เครื่องหมาย s (1 บิต)	ค่ายกกำลัง E_2 (8 บิต)	ทศนิยม Y_2 (23 บิต)	ความหมาย
1	±	0000 0001 ₂ - 1111 1110 ₂	xx.. ₂	เลขฐานสิบห้าไป (น้อมลัลไล์ช์)
2	±	0000 0000 ₂	xx.. ₂	เลขฐานสิบห้าอยมาก แต่ไม่เท่ากับศูนย์ (ดีนอมลัลไล์ช์)
3	0	0000 0000 ₂	0...0 ₂	0.0 ₁₀ (ศูนย์จุดศูนย์)
4	±	1111 1111 ₂	0...0 ₂	±∞ (±อินฟินิตี้)
5	0	1111 1111 ₂	xx.. ₂	Nan (Not a Number)

ผู้อ่านสามารถทำความเข้าใจตารางที่ 1.11 ตามลำดับที่ดังต่อไปนี้

1. ย่างของเลขทศนิยมที่มาตรฐานที่สามารถแสดงค่าได้ โดยมีค่าอยู่ระหว่างค่าต่ำสุดที่เข้าใกล้ศูนย์จนถึง ค่าที่มากแต่ยังไม่ใช่ค่าอนันต์หรืออินฟินิตี้ ซึ่งได้กล่าวไว้แล้ว
2. เลขทศนิยมที่มาตรฐานนี้ไม่สามารถแสดงค่าได้ เนื่องจากมีค่าน้อยมาก เรียกว่า ค่าดีนอมลัลไล์ช์ (Denormalize)
3. เลขทศนิยม 0.0₁₀ หรือ ศูนย์จุดศูนย์ เรียกว่า True zero
4. ค่าบวกและลบอนันต์หรืออินฟินิตี้ สามารถใช้แทนผลลัพธ์ที่มีค่าสูงเกินขอบเขตของการนำเสนอ เนื่องมาจากผลลัพธ์ที่คำนวณได้ทางคณิตศาสตร์ เช่น การคูณและการบวกเลขขนาดใหญ่มาก หรือ การหารด้วยตัวหารขนาดเล็กมาก
5. NaN เป็นสัญลักษณ์ซึ่งไม่สามารถแทนได้ด้วยตัวเลข ย่อมาจากคำว่า Not a Number แสดงถึง ความผิดพลาดที่อาจเกิดจากการเขียนโปรแกรมเพื่อใช้ตัวแปรหารค่าตัวตั้ง แต่ตัวแปรที่ใช้หารมีค่าที่เปลี่ยนแปลงไปจนกลายเป็น 0.0₁₀ เมื่อนำไปหารจะเกิดสัญลักษณ์นี้

รูปที่ 1.8 แสดงค่าบันเด้นจำนวนของ แควรต่างๆ ในตารางที่ 1.11 ยกเว้น NaN เส้นด้านบนของรูปแสดงเลขต่างๆ ที่เขียนในรูปน้อมลัลไล์ช์ได้ เส้นด้านล่างแสดงเลขขนาดเล็กมากๆ ที่ไม่สามารถเขียนในรูปน้อมลัลไล์ช์ได้ หรือ เลขดีนอมลัลไล์ช์ รูปที่ 1.8 แสดงเลขต่างๆ ที่เขียนในรูปน้อมลัลไล์ช์ได้จากซ้ายสุดถึงขวาสุด ของเส้นจำนวน สำหรับ Double Precision แต่สำหรับ Single Precision มีค่าตั้งแต่ -3.4×10^{38} ถึง -1.175×10^{-38} และ $+1.175 \times 10^{-38}$ ถึง $+3.4 \times 10^{38}$ ซึ่งตรงกับค่าในตารางที่ 1.1

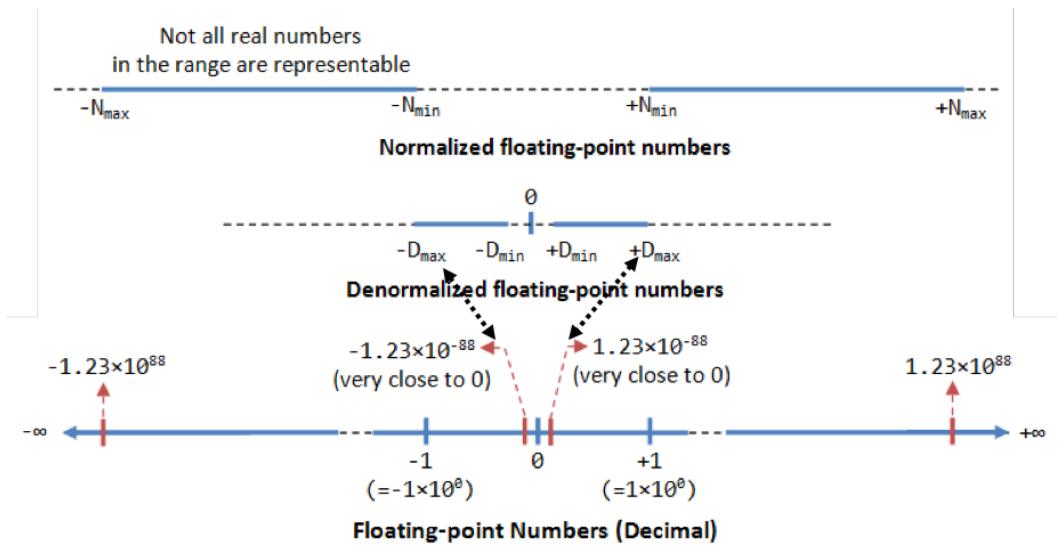


Figure 2.8: เลขทศนิยมโดยตัวชนิดนั่นน้อมลález ตั้งแต่ $\pm N_{min}$ ถึง $\pm N_{max}$ และชนิดที่ไม่สามารถเขียนแบบบัน omnález ตั้งแต่ $\pm D_{min}$ ถึง $\pm D_{max}$

2.6.3 การบวกเลขทศนิยมชนิดจุดโดยตัวตามมาตรฐาน IEEE 754

การบวกเลขทศนิยมฐานสองชนิดจุดโดยตัวตามมาตรฐาน IEEE 754 จำเป็นต้องอาศัยวงจรไฮาร์ดแวร์หรือ (Floating Point ALU) ช่วยคำนวณแบบไปป์ไลน์ (Pipeline) ซึ่งมีหลักการคล้ายกับ โรงงานประกอบรถยนต์ ทำให้ซีพียู 1 คอร์สสามารถคำนวณการบวกเลขทศนิยมได้ในหลัก กิกะฟล็อปส์ (Giga Floating Point Operations Per Second) หรือ พันล้านหรือ 10^9 ครั้งต่อวินาที ซึ่งในอดีตจนถึงปัจจุบัน การวัดสมรรถนะ (Performance) ของซีพียู รวมไปถึงชูเปอร์คอมพิวเตอร์ (Super Computer) ใช้หน่วยวัดเพตราฟล็อปส์ (Peta Flops) หรือ 10^{15} ครั้งต่อวินาที ผู้อ่านสามารถศึกษาเรื่อง ฟล็อปส์ ได้ที่ [wikipedia](#)

ดังนั้น การทำงานของวงจรบวก/ลบเลขทศนิยมชนิดจุดโดยตัว จึงอาศัยหลักการเดียวกันกับ อัลกอริทึมการบวก ซึ่งมีความซับซ้อนใกล้เคียงกับการคูณเลขทศนิยม และทั้งคู่มีความซับซ้อนมากกว่าการบวกและการคูณเลขจำนวนเต็มที่จำนวนบิตเท่ากัน

วงจรบวกเลขทศนิยมฐานสองชนิดจุดโดยตัวในรูปที่ 1.9 มีความซับซ้อนใกล้เคียงกับตัวอย่างที่แสดงในหัวข้อที่ 1.5 ที่ผ่านมา ดังนี้

- ขั้นตอนที่ 1 คือ ปรับจุดทศนิยมของค่าบิตสำหรับให้ตรงกันโดยเลื่อนตำแหน่งเลขบิตสำหรับของเลขที่มีค่ายกกำลังน้อยกว่า ผลต่างของค่ายกกำลังจากเลขทั้งสองตัวก็จะทราบว่าเลขตัวใดมีค่ายกกำลังมากกว่าเท่าใด และตัวใดที่มีค่ายกกำลังน้อยกว่าจะถูกเลื่อน (Shift) ไปทางขวาเป็นจำนวนบิตเท่ากับค่าสัมบูรณ์ (Absolute) ของผลต่าง
- ขั้นตอนที่ 2 คือ ทำการบวก/ลบค่าบิตสำหรับที่เลื่อนตำแหน่งแล้วเข้าด้วยกัน โดยก่อนหน้านั้นค่าทศนิยมของเลขทั้งสองตัวจะต้องบวกกับ 1 ที่ซ่อนอยู่ เพื่อปรับให้เป็นค่าบิตสำหรับก่อน ตัวที่มีค่ายกกำลังน้อยกว่าจะถูกเลื่อนไปทางขวาเสมอ ส่วนค่าบิตสำหรับที่มีค่ายกกำลังมากกว่าจะไม่เปลี่ยนแปลง

จึงหมายความว่าต้องเพิ่ม 1 เบิร์ก

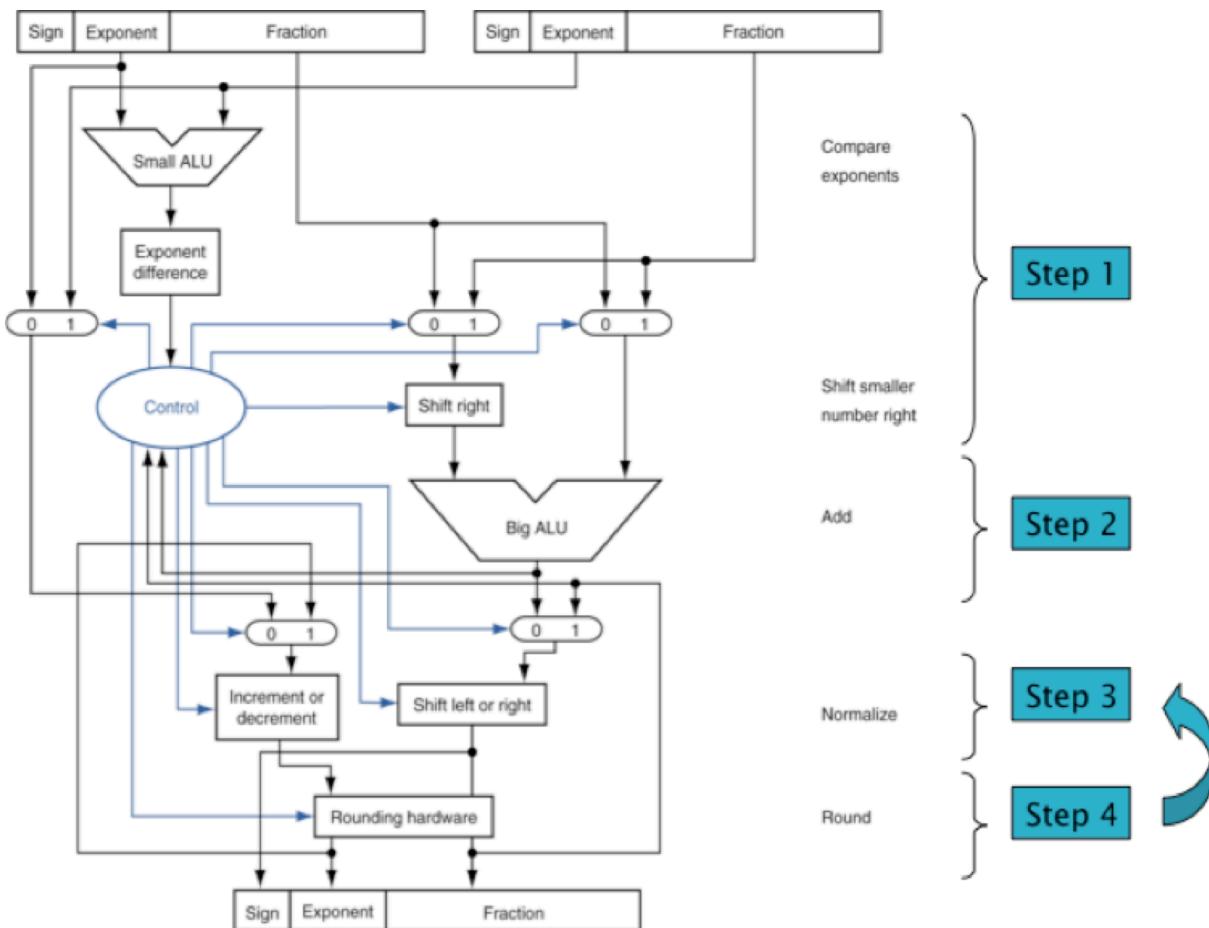


Figure 2.9: วิจารบกเลขชนิดจุดลอยตัวตามมาตรฐาน IEEE 754 โดยรับค่าอินพุทจำนวน 2 ตัวด้านบน และเอาท์พุทผลลัพธ์ด้านล่าง (ที่มา: Patterson and Hennessy (2000))

- ขั้นตอนที่ (Step) 3 คือ ทำการน้อมลัลไซด์ค่าผลลัพธ์ และตรวจเช็คการเกิดโอเวอร์โฟล์วและอันเดอร์โฟล์ว ซึ่งผลลัพธ์ที่ได้จากการบวกอาจมีค่าเพิ่มขึ้น หรือ ลดลงขึ้นอยู่กับเครื่องหมายและขนาดของเลขทั้งสองตัว ทำให้ต้องน้อมลัลไซด์ค่านัยสำคัญอีกรอบ ระหว่างนั้น
 - หากค่าสัมบูรณ์ของผลลัพธ์เพิ่มมากขึ้นจนเกินค่าสูงสุด เรียกว่าเกิด โอเวอร์โฟล์ว (Overflow) ของเลข IEEE754
 - หากค่าสัมบูรณ์ของผลลัพธ์มีค่าเข้าใกล้ศูนย์ และไม่สามารถเขียนในรูปน้อมลัลไซด์ได้ ผลลัพธ์นั้นจะเขียนในรูปแบบดินอนมลลaise (Denormalize) แทน เรียกว่าเกิดอันเดอร์โฟล์ว (Underflow) ของเลข IEEE754
- ขั้นตอนที่ (Step) 4 คือ ปัดค่าทศนิยมให้เหลือ 23 บิตและอาจต้องน้อมลัลไซด์ค่านัยสำคัญอีกรอบ เมื่อจำเป็น

ตัวอย่างที่ 2.6.5. การบวกเลขทศนิยมฐานสองชนิดจุดลอยตัว (Single Precision) มาตรฐาน IEEE754

$$-5_{10} + -0.75_{10} \text{ หรือเท่ากับ } COA0\ 0000_{16} + BF40\ 0000_{16} = ?$$

$$\begin{array}{r} 1100\ 0000\ 1010\ 0000\ 0000\ 0000\ 0000_2 \\ 1011\ 1111\ 0100\ 0000\ 0000\ 0000\ 0000_2 \\ \hline \end{array} \quad \begin{array}{l} 1.010 \times 2^0 \\ 1.100 \times 2^{-1} \\ 0.0011 \times 2^2 \end{array} \quad 1.0111$$

วิธีทำ

ผู้อ่านจะต้องแปลงเลขฐานสองให้เป็นรูปแบบเลขฐานสองที่น้อมลัลเลช์ 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. เลื่อนตำแหน่งจุดทศนิยมและปรับเลขยกกำลังที่มีค่าน้อยกว่า ให้ตรงกับเลขยกกำลังของเลขที่มีค่ามากกว่า

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

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

2. บวกค่านัยสำคัญที่เลื่อนตำแหน่งแล้ว

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

3. น้อมลัลเลช์ค่าผลลัพธ์ และตรวจสอบการเกิดโอเวอร์โฟล์วและอันเดอร์โฟล์ว

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

4. ปัดค่าให้เหลือ 23 บิตและอาจต้องน้อมลัลเลช์เมื่อจำเป็น

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

$$s = 1$$

$$\text{ค่ายกกำลัง } E_2 = 2+127 = 129 = 1000\ 0001_2$$

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

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

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

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

ผู้อ่านจะต้องแปลงเลขฐานสองให้เป็นรูปแบบเลขฐานสองที่นิยม ดังต่อไปนี้
 $= (-1)^0 \times (1.000\ 0000\ 0000\ 0000\ 0000)_2 \times 2^0 +$
 $(-1)^1 \times (1.100\ 0000\ 0000\ 0000\ 0000)_2 \times 2^{-1} \longrightarrow \text{WRONG}$

1. เลื่อนตำแหน่งจุดทศนิยมและปรับเลขยกกำลังที่มีค่าน้อยกว่า ให้ตรงกับเลขยกกำลังของเลขที่มีค่ามากกว่า

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

~~101~~

2. บวกค่านัยสำคัญที่เลื่อนตำแหน่งแล้ว

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

3. นำมัลไทร์ค่าผลลัพธ์ และตรวจเช็คการเกิดโอเวอร์โฟล์วและอันเดอร์โฟล์ว

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

4. ปัดค่าให้เหลือ 23 บิตและอาจต้องนำมัลไทร์อีกรอบหากจำเป็น

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

$$s = 0$$

$$\text{ค่ายกกำลัง } E_2 = -2 + 127 = 125 = 0111\ 1101_2$$

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

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

โดยสรุป ตัวอย่างที่ 1.6.5 และ 1.6.6 แสดงให้เห็นว่า การบวกเลขทศนิยมฐานสองภายในเครื่องคอมพิวเตอร์โดยใช้รัฐมาตรฐาน IEEE 754 สามารถทำงานตามขั้นตอนต่างๆ ตามบล็อกไดอะแกรมในรูปที่ 1.9

2.6.4 การคูณเลขทศนิยมฐานสองชนิดจุดลอยตัวตามมาตรฐาน IEEE 754

การคูณเลขทศนิยมฐานสองชนิดจุดลอยตัวตามมาตรฐาน IEEE 754 ด้วยความเร็วสูง จำเป็นต้องอาศัยวงจรฮาร์ดแวร์พิเศษ แต่การทำงานของวงจรนี้ในหลักการคล้ายกับอัลกอริทึมการคูณ ในหัวข้อที่ 1.3.1.2 ดังนั้น ผู้อ่านจำเป็นต้องศึกษาอัลกอริทึมให้เข้าใจอย่างถ่องแท้ก่อน

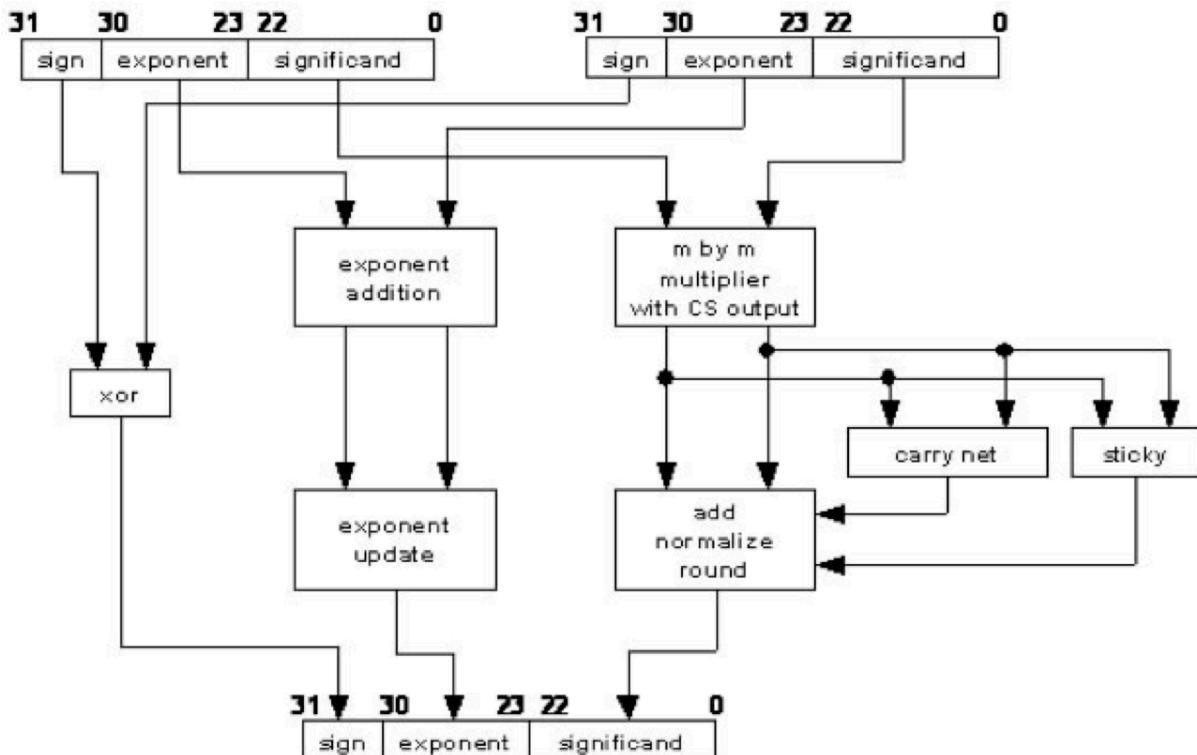


Figure 2.10: วงจรคูณเลขทศนิยมฐานสองชนิดจุดลอยตัวตามมาตรฐาน IEEE 754 โดยรับค่าอินพุตจำนวน 2 ตัวด้านบน และเอาท์พุตผลลัพธ์ด้านล่าง (ที่มา: Patterson and Hennessy (2000))

วงจรคูณเลขทศนิยมฐานสองชนิดจุดลอยตัว ในรูปที่ 1.10 มีความซับซ้อนมากกว่าวงจรคูณเลขจำนวนเต็มในรูปที่ 1.5 และ 1.6 ดังนี้

- ขั้นตอนที่ (Step) 1 คือ ทำการคูณเฉพาะค่านัยสำคัญทั้งสองเพื่อหาค่านัยสำคัญของผลคูณ
- ขั้นตอนที่ (Step) 2 คือ ทำการบวกค่าไถ่กำลังทั้งสองเข้าด้วยกันแล้วลบค่าไบอสหนึ่งครั้ง
- ขั้นตอนที่ (Step) 3 คือ ทำการวนรอบลัพธ์ และตรวจสอบเช็คโควอร์ฟล์และอันเดอร์ฟล์ เหมือนกับกรณีการบวกเลข
- ขั้นตอนที่ (Step) 4 คือ ปัดค่าทศนิยมให้เหลือ 23 บิตและอาจต้องวนรอบเมื่อจำเป็น

ตัวอย่างที่ 2.6.7. จงคูณเลขทศนิยมฐานสองชนิดจุดลอยตัวแบบ Single Precision ดังต่อไปนี้

$$\text{EX: } -5_{10} \times -0.75_{10} = \text{COA0 } 0000_{16} \times \text{BF40 } 0000_{16} = ?$$

1. แปลงเลขฐานสิบหกให้เป็นเลขฐานสอง

$$1100\ 0000\ 1010\ 0000\ 0000\ 0000\ 0000_2 \times$$

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

2. แปลงเลขฐานสองตามกฎของ IEEE 754 Single Precision รูปแบบเลขฐานสองที่น้อมลัลีซ์ ดังต่อไปนี้

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

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

1010

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

1100

$$= (-1)^1 \times (-1)^1 \times [(1.010\ 0000\ 0000\ 0000\ 0000)_2 \times$$

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

101000
1010000
1111000

4. คูณค่านัยสำคัญเข้าด้วยกัน

$$1.010\ 0000\ 0000\ 0000\ 0000_2 \times$$

$$_1.100\ 0000\ 0000\ 0000\ 0000_2$$

$$= 1.111\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000_2$$

5. น้อมลัลีซ์ค่าผลลัพธ์ และตรวจเช็คการเกิดโอเวอร์โฟล์วและอันเดอร์โฟล์ว

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

6. ปัดค่าให้เหลือ 23 บิตและอาจต้องน้อมลัลีซ์อีกรอบหากจำเป็น

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

7. ปรับเครื่องหมายของผลลัพธ์ให้ถูกต้องจากตัวตั้งและตัวคูณ

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

บิตเครื่องหมาย s = 0

$$\text{ค่ายกกำลัง } E_2 = 1+127 = 128 = 1000\ 0000_2$$

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

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

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

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

โดยสรุป ตัวอย่างที่ 1.6.7 แสดงให้เห็นว่า การคูณเลขทศนิยมฐานสองภายในเครื่องคอมพิวเตอร์ตามมาตรฐาน IEEE 754 สามารถทำงานตามขั้นตอนต่างๆ ตามบล็อกไดอะแกรมในรูปที่ 1.10

2.7 ตัวอักษร (Character)

ตัวอักษรในภาษาต่างๆ ล้วนเป็นข้อมูลที่สำคัญในการใช้งานคอมพิวเตอร์ โดยเฉพาะชื่อ นามสกุล หนังสือ ตำรา ข้อความ คำสอน那麼ต่างๆ ในเครือข่ายอินเทอร์เน็ต เพื่อนำมาสืบค้น สร้างความเขื่อมโยงในหลากหลายมิติผ่านสื่อสังคมออนไลน์ต่างๆ ได้อย่างแพร่หลาย รูปที่ 1.11 แสดง การเก็บข้อความในหน่วยความจำในรูปของรหัส ASCII ด้วยตัวแปร str ซึ่งเป็นตัวแปรชนิด一字串ของตัวอักษร (Array of Character)

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

ที่ดำเนินการหรือแอ็ดเดรสเริ่มต้น 0x50 การเรียงไปที่ตัวอักษรจะเริ่มต้นจากไบท์ที่แอ็ดเดรสต่าไปยังแอ็ดเดรสที่สูงขึ้น ดังรูป แอ็ดเดรส 0x50 เก็บรหัส ASCII หมายเลข 0x48 ซึ่งตรงกับตัวอักษร 'H' ไปจนถึงแอ็ดเดรส 0x56 ซึ่งเก็บตัวอักษรหัส 0x00 หรือ NULL อ่านว่า นัล ซึ่งเป็นอักษรพิเศษแสดงว่าจบประโยค รูปที่ 1.11 ส่วนไบท์ที่ 0x57, 0x58, 0x59 จะมีค่าเป็น unknown เพราะไม่มีผลต่อค่าของประโยคที่เก็บตัวแปร str นี้สามารถเก็บตัวอักษรความยาวสูงสุด 10 ตัวและตัวสุดท้ายจะต้องเป็นตัวอักษร NULL เท่านั้นยกตัวอย่างเช่น ประโยค "012345678" ประโยค "abcdefghi" เป็นต้น

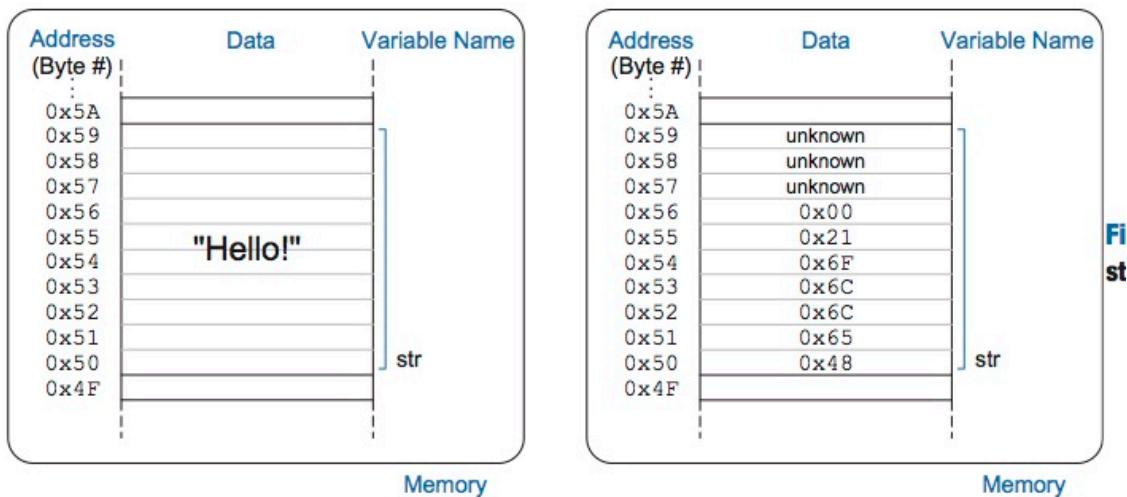


Figure 2.11: การเก็บข้อความในหน่วยความจำในรูปของรหัส ASCII ด้วยตัวแปร str ซึ่งเป็นตัวแปรชนิด char[10] str="Hello!" ที่แอ็ดเดรสเริ่มต้น 0x50 (ที่มา: [Harris and Harris \(2013\)](#))

รหัส ASCII คือ มาตรฐานของรูปแบบการใช้เลขฐานสองเพื่อแทนตัวอักษรตั้งแต่อดีต ซึ่งกำหนดขึ้นมาโดยหน่วยงานชื่อว่า ANSI (American National Standard Institute) ตารางที่ 1.12 คือ ตารางรหัสแอลกอริทึม (ASCII) กำหนดเลขฐานสองขนาด 8 บิต เพื่อแทนตัวอักษรจำนวน $2^8=256$ ตัว โดยมีรหัสเริ่มต้น คือ $00_{10} = 0000\ 0000_2 = 00_{16}$ ถึง $255_{10} = 111\ 1111_2 = FF_{16}$ โดยรหัส 00_{16} แทนอักษร NULL ไม่គรุชอฟต์ พัฒนารหัสภาษาไทยของตนเอง เรียกว่า 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	o
7-	0070	p	q	r	s	t	u	v	w	x	y	z	{		}	~
8-																
9-																
A-	0E48	ກ	ຂ	ໝ	ຄ	ໜ	ໝ	ໝ	ຈ	ໜ	ໜ	ໜ	ໜ	ໜ	ໜ	ໜ
B-	0E10	ໜ	ໜ	ໜ	ດ	ດ	ດ	ດ	ກ	ກ	ນ	ບ	ປ	ຜ	ພ	ພ
C-	0E20	ກ	ມ	ຢ	ຮ	ຖ	ຄ	ກ	ວ	ສ	ໜ	ຫ	ຟ	ວ	ອ	ໜ
D-	0E30	ໜ	ໜ	ໜ	ໜ	ໜ	ໜ	ໜ	ໜ	ໜ	ໜ	ໜ	ໜ	ໜ	ໜ	ໜ
E-	0E40	ເ	ເ	ໂ	ໜ	ໜ	ໜ	ໜ	ໜ	ໜ	ໜ	ໜ	ໜ	ໜ	ໜ	ໜ
F-	0E50	ອ	ອ	ເ	ຕ	ແ	ແ	ແ	ແ	ແ	ແ	ແ	ແ	ແ	ແ	ແ

Figure 2.12: ตารางรหัสแอสกี้ (ASCII) และ Unicode สำหรับตัวอักษรจำนวน $2^8=256$ ตัว ภาษาอังกฤษ และภาษาไทย ที่มา: ascii-table.com

รหัส [Unicode](#) ถูกกำหนดให้เป็นมาตรฐานโดย ISO (International Standard Organization) เพื่อมาทดแทนรหัส ASCII เนื่องจากความต้องการใช้ภาษาทั่วโลกที่เพิ่มขึ้นเรื่อยๆ รหัส [Unicode](#) กำหนดวิธีการเข้ารหัสที่หลากหลายตามวัตถุประสงค์การใช้งาน เช่น รหัส UTF-8 รหัส UTF-16 รหัส UCS-2 เป็นต้น

- รหัส [UTF-8](#) นิยมใช้ในเว็บเพจต่างๆ โดยแต่ละตัวอักษรจะใช้ความยาว ตั้งแต่ 1 ไบท์ จนถึง 4 ไบท์ โดยตัวอักษร 128 ตัวแรกคือรหัส ASCII ใช้ความยาว 1 ไบท์ ส่วนตัวอักษรในภาษาอื่นๆ จะใช้จำนวนไบท์เพิ่มขึ้น โดยตำแหน่งเริ่มต้นของตารางรหัส Unicode จะเหมือนกับตารางรหัส ASCII ตัวภาษาอังกฤษและภาษาไทย ในรูปที่ 1.12 ภายใต้ตัวอักษร มีค่ารหัส Unicode กำกับอยู่ด้วยยกตัวอย่างเช่น ตัวอักษรไทยเริ่มต้นที่ รหัส ASCII เท่ากับ A1 คือ ก ในรูปของเลขฐานสองขนาด 8 บิต ตรงกับรหัส Unicode 0E01 ความยาว 16 บิต
- รหัส [UCS-2](#) จะใช้พื้นที่ 2 ไบท์ หรือ 16 บิต ต่อ 1 ตัวอักษร ซึ่งทำให้สามารถใช้เลขฐานสองจำนวน 216 หรือ 65,536 แบบมาแทนตัวอักษร ซึ่งทำให้วิธีการนี้ไม่ได้รับความนิยม

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

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

2.8 สรุปท้ายบท

การคำนวณทางคณิตศาสตร์ของเลขจำนวนเต็มชนิดมีเครื่องหมาย และไม่มีเครื่องหมายจำเป็นต้องตรวจจับการเกิดโอเวอร์โฟล์ว เนื่องจากฮาร์ดแวร์หรือโปรเซสเซอร์มีจำนวนบิตในการคำนวณที่จำกัด ปัจจุบันคือ 32/64 และ 128 บิต ในทำนองเดียวกัน การคำนวณโดยใช้เลขทศนิยมชนิดจุดทศนิยมคงที่ หรือ Fixed Point และloyตัว หรือ Floating-point จำเป็นต้องตรวจจับการเกิดโอเวอร์โฟล์ว/อันเดอร์โฟล์ว และกรณีอื่นๆ ด้วยเหตุผลที่ซับซ้อนกว่าเลขจำนวนเต็ม ทำให้ใช้ทรัพยากรด้านฮาร์ดแวร์และเวลาหรือจำนวนคลื่อกมากกว่า

ชนิดและความยาวของข้อมูลที่หลากหลายตามที่สรุปในตารางที่ [1.12](#) ของซอฟต์แวร์คอมพิวเตอร์ ต้องการความสามารถของไมโครโปรเซสเซอร์หรือฮาร์ดแวร์ให้รองรับตามเข่นกัน เพื่อให้ประสบการณ์การใช้งานของผู้ใช้ (User Experience) ดีขึ้นเรื่อยๆ ยกตัวอย่างเช่น การใช้ GPU มาช่วยคำนวณ การคำนวณข้อมูลที่จัดเรียงตัวกันแบบเวกเตอร์ (Vector) เพื่อเพิ่มประสิทธิภาพ เป็นต้น

Table 2.12: ชนิด ความยาว ข้อมูล และการประยุกต์ใช้งานเลขฐานสองชนิดต่างๆ ในคอมพิวเตอร์

ชนิด	บิต	ข้อมูล	การประยุกต์ใช้งาน
char	8	ตัวอักษร	ข้อความ อีเมล
unsigned char	8	จุดภาพ	รูปภาพขาวดำ และ Gray Scale
unsigned char	8		รูปภาพสี RGB Bitmap JPEG
unsigned int	32/64	แอดเดรส	พอยท์เตอร์ซึ่งตำแหน่งข้อมูล ระบบ 32/64 บิต
unsigned int	32/64	จำนวน	จำนวนอุปกรณ์ IoT จำนวนดาวเทา
int	32/64		
ทศนิยม Fixed	16-32	เสียง	ข้อมูลเสียงดนตรี
ทศนิยม Fixed	16-32	จุดภาพ	ข้อมูลภาพความละเอียดสูง
float	32	จุดภาพ	ข้อมูลภาพความละเอียดสูง
float	32	ระยะทาง	เกม 3 มิติ
double	64	± ระยะทาง	ระยะทางไปยังดาวต่างๆ นอกโลก
double	64	± น้ำหนัก	น้ำหนักดาวต่างๆ นำหนักอนุภาคเล็กๆ

2.9 คำถ้ามห้ายบท

- จงแสดงวิธีบวกเลขจำนวนเต็มฐานสิบและฐานสองชนิดไม่มีเครื่องหมาย ขนาด 8 บิต ต่อไปนี้ และตรวจสอบว่าเกิดโอลเวอร์ฟอล์ตามสมการที่ 1.43
 - $255_{10} + 1_{10}$
 - $128_{10} + 127_{10}$
 - $1111\ 1110_2 + 0000\ 0011_2$
 - $1000\ 0000_2 + 0111\ 1111_2$
- จงแสดงวิธีคูณเลขจำนวนเต็มฐานสองชนิดไม่มีเครื่องหมาย ขนาด 8 บิต ต่อไปนี้ ด้วยวงจรคูณเลขชนิดที่ 1 ในรูปที่ 1.5 ตามตัวอย่างที่ 1.9
 - $0000\ 1111_2 \times 0000\ 1000_2$
 - $0000\ 1111_2 \times 0000\ 1111_2$
 - $1010\ 1010_2 \times 0101\ 0101_2$
 - $1111\ 1111_2 \times 1111\ 1111_2$
- จงแสดงวิธีคูณเลขจำนวนเต็มฐานสองชนิดไม่มีเครื่องหมาย ขนาด 8 บิต ต่อไปนี้ ด้วยวงจรคูณเลขชนิดที่ 2 ในรูปที่ 1.6 ตามตัวอย่างที่ 1.10
 - $0000\ 1111_2 \times 0000\ 1000_2$
 - $0000\ 1111_2 \times 0000\ 1111_2$
 - $1010\ 1010_2 \times 0101\ 0101_2$
 - $1111\ 1111_2 \times 1111\ 1111_2$
- จงแสดงวิธีบวกเลขจำนวนเต็มฐานสิบและฐานสองชนิดมีเครื่องหมาย 2-Complement ขนาด 8 บิต ต่อไปนี้ และตรวจสอบว่าเกิดโอลเวอร์ฟอล์ตามสมการที่ 1.47
 - $126_{10} + 2_{10}$
 - $128_{10} - 127_{10}$
 - $1111\ 1110_2 + 0000\ 0011_2$
 - $1000\ 0000_2 + 0111\ 1111_2$
- จงแสดงวิธีบวกเลขทศนิยมชนิดจุดคงที่ ขนาด 4.4 บิต ต่อไปนี้ และตรวจสอบว่าเกิดโอลเวอร์ฟอล์ หรือไม่ (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 IEEE 754 Single-Precision และตรวจ

สอบว่าเกิดโอเวอร์โฟล์ว/อันเดอร์โฟล์วหรือไม่ ตามตัวอย่างที่ 1.6.5

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

$$\begin{aligned} & [1][0111\ 111][01000\dots] + \\ & [1][1000\ 000][01000\dots] \\ & [1][0111\ 111][1.01000\dots] + \\ & [1][1000\ 000][1.01000\dots] \\ & \downarrow [1][1000\ 000][0.101] \\ & [1][0111\ 111][1.01] \\ & [1][111] [0100\dots] \end{aligned}$$

7. จงแสดงวิธีคูณเลขทศนิยมฐานสิบต่อไปนี้ ตามมาตรฐาน IEEE 754 Single-Precision และตรวจ

สอบว่าเกิดโอเวอร์โฟล์ว/อันเดอร์โฟล์วหรือไม่ ตามตัวอย่างที่ 1.6.7

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

$$\begin{aligned} & \begin{array}{c} 1 - 0.75 \\ 1.0 \quad 0.75 \\ \hline 0.25 \end{array} \quad 1.1 \times 2^{-1} \\ & [0][0111\ 111][0000] \quad [1.1 \times 2^{-1}] \\ & [1][0111\ 110][1000] \quad [1.1 \times 2^{-1}] \\ & [0][0111\ 111][0.0100] \\ & [0][0111\ 1110][0.100] \\ & [0][0111\ 1101][1.000] \\ & \quad 2^{125-127} \\ & \quad 2^{-2} \times 1.0 \\ & \quad 0.25 \end{aligned}$$

$$\begin{aligned} & 4.5 - 6.25 \\ & 100.1 \quad 110.01 \\ & 1.001 \times 2^2 \quad 1.1001 \times 2^2 \\ & [1][1000\ 0001][1.1001] \leftrightarrow [1001] \\ & [0][1000\ 0001][1.0010] \leftrightarrow [0010] \\ & [1][1000\ 0001] 0.0111 \\ & [0111\dots] [11] \\ & \quad 1.11 \times 2^1 \end{aligned}$$

$$-4.75 + 5.50$$

$$(-100.11 \times 2^0) + (101.1 \times 2^0)$$

$$1.0011 \times 2^0 + 1.011 \times 2^0$$

$$\begin{array}{r} [1000\ 0001][1.011] \\ , , [1.0011] \\ \hline \end{array}$$

$$\begin{array}{r} 0.0011 \\ [0111\ 1101][1] \\ \hline 1.1 \times 2^{-1} \\ 0.11 \end{array}$$

ฮาร์ดแวร์และซอฟต์แวร์ของคอมพิวเตอร์ (Computer Hardware & Software)

บอร์ด Raspberry Pi3 เป็นคอมพิวเตอร์ขนาดเล็กที่ได้รับความนิยม เนื่องจากราคาไม่แพง การออกแบบ ฮาร์ดแวร์และซอฟต์แวร์ที่ลงตัว รวมไปถึงมีซอฟต์แวร์ประยุกต์ หรือ อ�플ิเคชัน (Application) ที่หลากหลาย กรณีศึกษาบอร์ด Raspberry Pi3 ในทำาระล่มนี้จะหมายถึงบอร์ด Pi3 โมเดล B ในรูปที่ 3.1 นอกจาก โมเดล B บอร์ด Pi3 โมเดล A มีลักษณะต่างกันไม่น่าจะมากแต่จะเน้นการทำงาน แบบระบบฝังตัว (Embedded System) เป็นหลัก รายละเอียดเพิ่มเติมที่ wikidevi.com และ elinux.org

วัตถุประสงค์

- เพื่อให้ผู้อ่านเข้าใจโครงสร้างโดยองค์รวมของคอมพิวเตอร์ กรณีศึกษาบอร์ด Raspberry Pi3
- เพื่อให้ผู้อ่านเข้าใจโครงสร้างและการทำงานของด้านฮาร์ดแวร์ กรณีศึกษา ARM Cortex A53 ภายใน ชิพ BCM2837 บนบอร์ด Raspberry Pi3
- เพื่อให้ผู้อ่านเข้าใจโครงสร้างและการทำงานของระบบปฏิบัติการลีนุกซ์ และโครงสร้างของซอฟต์แวร์ ประยุกต์
- เพื่อให้ผู้อ่านเข้าใจการประสานงานระหว่างฮาร์ดแวร์และซอฟต์แวร์ เช่น การรันซอฟต์แวร์ และ การสั่งงานให้ฮาร์ดแวร์ทำงาน
- เพื่อให้ผู้อ่านเข้าใจขบวนการพัฒนาซอฟต์แวร์ด้วยภาษา C และภาษาแอสเซมบลี

บอร์ด Pi3 โมเดล B สามารถประยุกต์ใช้ได้หลากหลาย เช่น เครื่องคอมพิวเตอร์ส่วนตัว แท็บเล็ต โน๊ต บุคราคายาด อุปกรณ์เฝ้าระวังความปลอดภัย/สภาพแวดล้อมในรูปแบบ IoT (Internet of Things) ระบบควบคุมในบ้านและอุตสาหกรรม เชิร์ฟเวอร์สำหรับเครื่องพิมพ์ (Print server) อุปกรณ์เชื่อมต่อสัญญาณ WiFi กับเครือข่ายสาย และอื่นๆ เป็นต้น

3.1 ฮาร์ดแวร์ของเครื่องคอมพิวเตอร์

บอร์ด Pi3 โมเดล B เป็นคอมพิวเตอร์บอร์ดเดี่ยว (Single Board) ราคาเพื่อการศึกษา เป็นรุ่นถัดจาก Raspberry Pi 2 (Pi 2) โมเดล B ออกแบบและพัฒนาโดยองค์กรที่มีชื่อว่า Raspberry Pi Foundation โดยทั้งบอร์ด Pi2 และ Pi3 มีขนาดทางกายภาพเท่ากัน (86mm x 56mm x 21mm) ทำให้ใช้กล่อง (Case) ด้วยกันได้ ตำแหน่งของพอร์ตและคอนเนกเตอร์คล้ายกัน แต่บอร์ด Pi3 มีความสามารถในการประมวลผลที่สูงขึ้นและมีประสิทธิภาพดีกว่า ตัวอย่างการเปรียบเทียบบอร์ดรุ่นต่างๆ จัดทำโดย medium.com

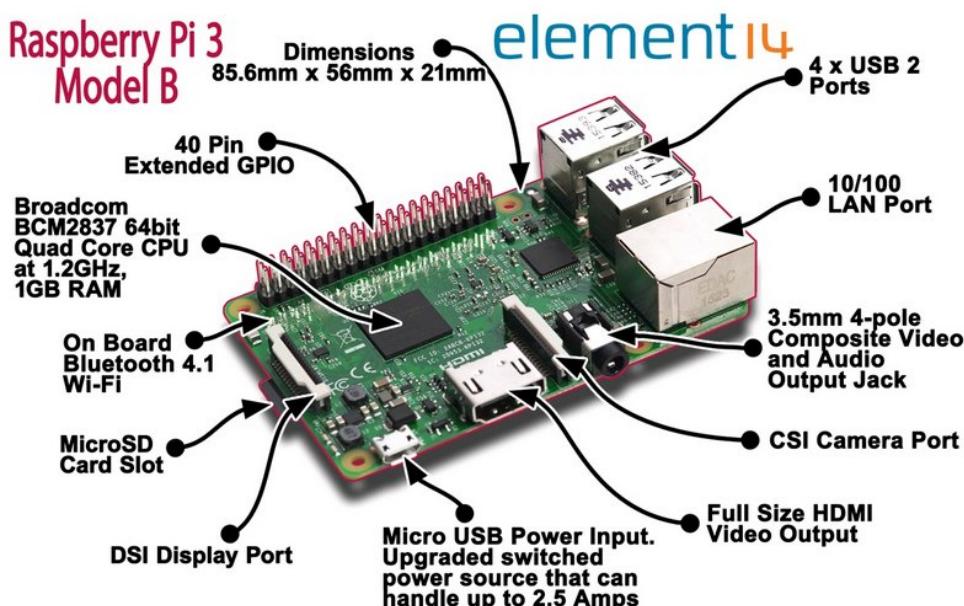


Figure 3.1: ตำแหน่งของอุปกรณ์ต่างๆ บนบอร์ด Pi3 ที่มา: www.raspberryhome.net

แผ่นวงจรพิมพ์ของบอร์ด Pi3 โมเดล B ในรูปที่ 3.1 ประกอบด้วยชิป Broadcom Corp. ประเทศไทย หน่วยความจำหลัก SDRAM (Synchronous Dynamic Random Access Memory) ชนิด DDR2 (Double Data Rate 2) ขนาดความจุ 1 จิกะไบท์ อุปกรณ์เก็บรักษาข้อมูลชนิด SD (Secure Digital) Memory card ขนาด 8-16 จิกะไบท์ เชื่อมต่ออินเตอร์เน็ตผ่านระบบสื่อสารไร้สาย WiFi, Bluetooth และช่องเสียบสาย Ethernet LAN ช่องเสียบ USB สำหรับเชื่อมต่อคีย์บอร์ดและเมาส์ เป็นต้น

องค์ประกอบสำคัญของบอร์ด คือ ชิป BCM2837 เป็น SoC ย่อมาจาก **System on Chip** ที่มีชิปยู ARM Cortex A53 ARMv8 แบบสี่คอร์ (Quad-core) ขนาด 32 และ 64 บิต ใช้ความถี่สัญญาณคลื่น 1.2GHz ซึ่งแตกต่างจากกรณีของบอร์ด Pi2 ที่ใช้ BCM2836 ซึ่งเป็น SoC ที่มีชิปยู Cortex-A7 ARMv7 ขนาด 32 บิต ทำงานที่ความถี่ 900MHz เมื่อว่าจะเป็นแบบสี่คอร์ (Quad-core) เช่นกัน แต่บอร์ด RPi2 มีหน่วยความจำเพียง 512 MB เท่านั้น นอกจากนั้น บอร์ด Pi3 ยังมีชิปสำหรับเชื่อมต่อ Wi-Fi เวอร์ชัน IEEE 802.11n และ Bluetooth เวอร์ชัน 4.1 ได้โดยตรง ในขณะที่บอร์ด Pi2 ต้องซื้ออุปกรณ์มาต่อเพิ่ม มูลนิธิได้ประกาศรายละเอียดของบอร์ด Raspberry Pi4 แล้ว

จุดเด่นของบอร์ด Pi3 โมเดล B คือ การเชื่อมต่อกับอุปกรณ์อินพุท/เอาท์พุทที่หลากหลาย รูปที่ 3.2 แสดงการเชื่อมโยงอุปกรณ์ต่างๆ บนบอร์ด Pi3 โดยมีชิป BCM2837 เชื่อมต่อกับอุปกรณ์ภายนอกเท่าที่

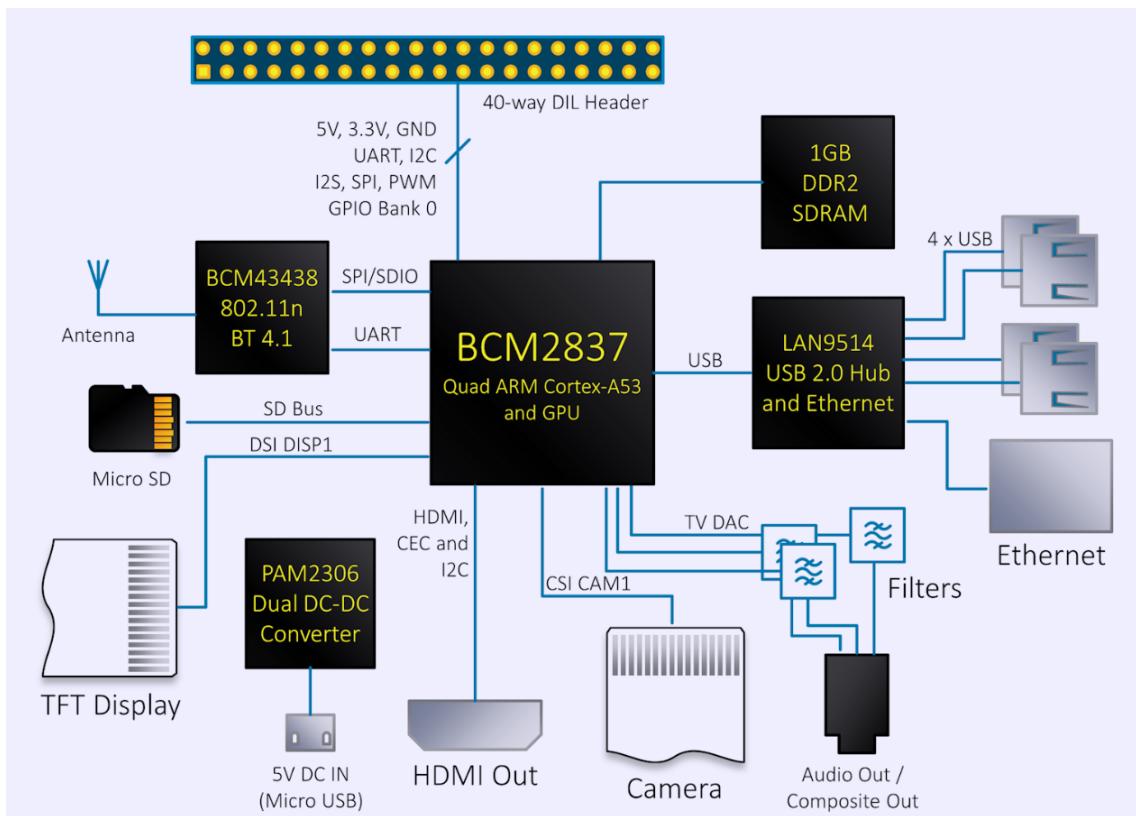


Figure 3.2: การเชื่อมโยงอุปกรณ์ต่างๆ บนบอร์ด Pi3 โดยมีชิป BCM2837 เป็นศูนย์กลาง ที่มา: xdevs.com

จำเป็น เพื่อให้ระบบสมบูรณ์ ได้แก่ หน่วยความจำหลัก SDRAM ขนาด 1GB อุปกรณ์เก็บรักษาข้อมูลชนิด Micro SD ผ่านสาย SDIO (Secure Digital Input/Output) บอร์ด Pi3 รองรับการเชื่อมต่อกับจอแสดงผลได้ 3 ชนิด คือ จอโทรทัศน์ด้วยสัญญาณคอมโพสิตวีดีโอ จอด LCD ขนาดใหญ่ผ่านสาย HDMI และ จอกาฟ LCD ขนาดเล็กผ่านสาย DSI บอร์ดมีพอร์ต USB 2.0 จำนวน 4 พอร์ต สามารถเชื่อมต่ออินเทอร์เน็ตด้วยสาย Ethernet หรือสายแลน (LAN: Local Area Network) และเชื่อมต่อกับระบบเครือข่ายแบบไร้สาย ด้วยชิป BCM43438 ซึ่งภายในมี WiFi และ Bluetooth โมดูล โดยเชื่อมกับชิป BCM2837 ผ่านโมดูล UART และ SPI/SDIO ตามลำดับ ซึ่งได้สรุปในตารางที่ 3.1

3.1.1 ชีพียู (CPU: ARM Cortex A53)

CPU ย่อมาจากคำว่า Central Processing Unit เป็นวงจรดิจิทัลอยู่ภายใน ชิป BCM2837 ผลิตโดยบริษัท Broadcom ประกอบด้วย

- ชีพียู ARM Cortex A53 จำนวน 4 คอร์ (Quad core) ใช้สัญญาณนาฬิกาความถี่ 1.2 GHz รองรับคำสั่งภาษาแอสเซมบลี (Assembly) ของ ARM เวอร์ชัน 8A
- จีพียู (GPU: Graphic Processing Unit) เป็นหน่วยประมวลผลด้านกราฟิก จำนวน 2 คอร์ ใช้สัญญาณนาฬิกาความถี่ 400 MHz ออกแบบโดยบริษัท BroadCom ทำหน้าที่เป็น Multimedia Co-Processor รองรับไลบรารี OpenGL ES เวอร์ชัน 2.0 และถอดรหัสัญญาณภาพวีดีโอมาตรฐาน

Table 3.1: ตารางสรุปข้อมูลด้านชาร์ดแวร์และซอฟต์แวร์ของบอร์ด Pi3 โมเดล B และลิงค์เชื่อมไปยังหัวข้อที่แสดงรายละเอียดในบทต่างๆ

โมดูล	ภายในชิป (On chip) BCM2837	ในหัวข้อที่
CPU	SoC ผลิตโดยบริษัท Broadcom ประกอบด้วย Quad (4)-Core ARM Cortex-A53 ความถี่ 1.2 GHz	3.1.1
GPU	Dual (2) VideoCoreIV ความถี่ 400 MHz	
จอ LCD	สาย HDMI เวอร์ชัน 1.3 & 1.4 (ภาพและเสียง)	2.1
จอ LCD	สาย Display Serial Interface (DSI) 15 ขา ประกอบด้วยสัญญาณข้อมูล 2 คู่ สัญญาณคล็อก 1 คู่	2.2
กล้องขนาดเล็ก	สาย Camera Serial Interface (CSI) 15-ขา ประกอบด้วยสัญญาณข้อมูล 2 คู่ สัญญาณคล็อก 1 คู่	2.3
จอทีวี และเสียง	สัญญาณคอมโพลีทวีดิจิทัล PAL/NTSC แจ็ค 3.5 มม ชนิด 4 ขา	2.5 2.4
GPIO	ขาต่อชนิด 2.54 มม 40 ขา ประกอบด้วย GPIO 27 ขา +3.3 และ +5V โวลท์	2.11
อุปกรณ์	ภายนอกชิป (Off chip) BCM2837	ในหัวข้อที่
ชิป SDRAM	หน่วยความจำชนิด DDR2 ความจุ 1 จิกะไบท์ ชนิดประยุกต์พลังงาน	3.1.2, 5.5
ชิป USB	ชิป USB 2.0 จำนวน 4 พอร์ต	2.6
ชิป Ethernet	เชื่อมต่ออินเทอร์เน็ตผ่านสาย ด้วยอัตรา 10/100 Mbps	2.7
ชิป WiFi และ Bluetooth	เชื่อมต่ออินเทอร์เน็ตไร้สาย IEEE 802.11 b/g/n อัตราเร็วสูงสุด 150Mbps การเชื่อมต่อไร้สายเวอร์ชัน 4.1	2.8
แหล่งจ่ายไฟ	ช็อกเก็ตชนิด microUSB ขนาด 5 โวลท์ 2.5 แอมเปอร์	2.14
การ์ด MicroSD	อุปกรณ์เก็บรักษาข้อมูลความจุ 4-16 จิกะไบท์	3.1.4, 7.3
ซอฟต์แวร์	ระบบปฏิบัติการบูทจากการ์ด MicroSD รองรับ Raspbian Linux Windows 10 IoT และอื่นๆ	3.2.1

H.264 High-Profile ที่ความละเอียดภาพ 1080p ความถี่ภาพ 30 เฟรมต่อวินาที [รายละเอียดเพิ่มเติม](#) และรองรับ OpenGL ES เวอร์ชัน 2.0 และ OpenVG

- โมดูลจัดการวีดิจิทัล (Video Engine) สามารถจัดการหัสภาพวีดิจิทัลที่ความละเอียด 1080p30 ตามมาตรฐาน ITU H.264 ด้วยอัตรา 1 พันล้านพิกเซลต่อวินาที และโมดูล DMA (Direct Memory Access)
- วงจรเชื่อมต่อหน่วยความจำหลัก SDRAM ชนิด DDR2
- วงจรเชื่อมต่ออุปกรณ์เก็บรักษาข้อมูล SD เวอร์ชัน 2.0
- โมดูลเชื่อมต่ออินพุตและเอาท์พุต ซึ่งรายละเอียดในตารางที่ [3.1](#)

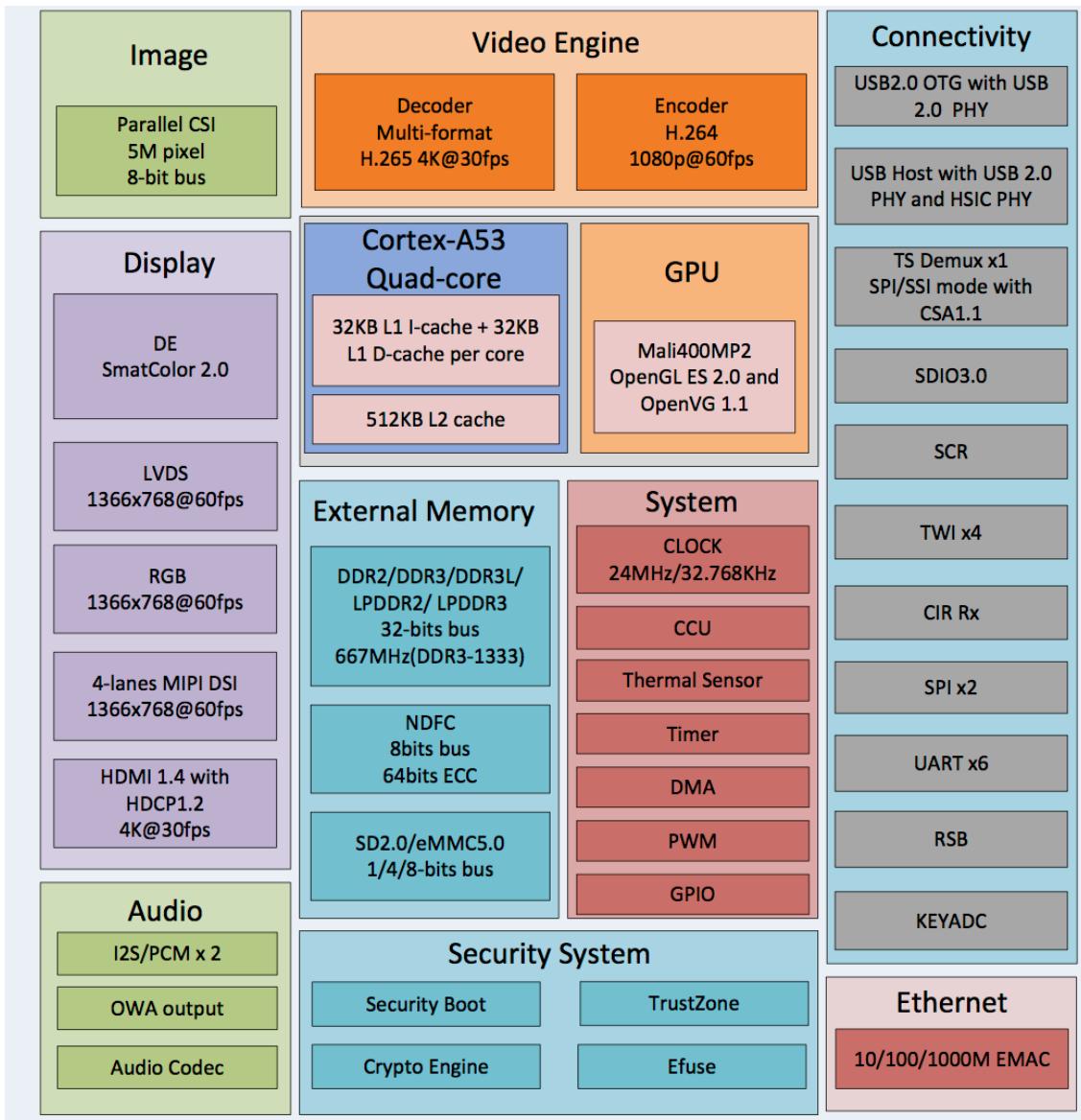


Figure 3.3: บล็อกไดอะแกรมของชิป AllWinner A64 (ที่มา: [Allwinner Technology \(2015a\)](#) และ [Allwinner Technology \(2015b\)](#)) ที่มี ARM Cortex A53 คล้ายกับ BCM2837 บนบอร์ด Raspberry Pi3

ผู้เขียนขอใช้ข้อมูลจากชิปที่มีคุณสมบัติใกล้เคียงกันกับ BCM2837 ตามรูปที่ 3.3 บล็อกໄดอะแกรมของชิป A64 จากบริษัท AllWinner ที่มา: [Allwinner Technology \(2015a\)](#) และ [Allwinner Technology \(2015b\)](#) เนื่องจากมี ARM Cortex A53 คล้ายกับ BCM2837 แต่ไม่ดูหลำคัญๆ ต่างกัน เช่น GPU, Ethernet เป็นต้น

3.1.2 หน่วยความจำหลัก (1GB DDR2 SDRAM)

เป็นชิพหน่วยความจำไดนามิกแรมแบบซิงโครนัส หรือ SDRAM (Synchronous Dynamic RAM) ชนิด DDR2 (Double Data Rate 2) เหมาะสำหรับใช้งานบนอุปกรณ์เคลื่อนที่ (Mobile) เพราะเป็นรุ่นประหยัดพลังงาน ชนิด LP (Low Power) และจำนวนขาน้อยเพื่อประหยัดพื้นที่บนบอร์ด ชิพหน่วยความจำหลักนี้รองรับสัญญาณความถี่นาฬิกาสูงสุด 400 MHz ทำให้เกิดความเร็วสูงสุด 800 เมกะบิต/วินาที เนื่องด้วย



Figure 3.4: หน่วยความจำไดนามิกแรมด้านล่างของบอร์ด Pi3 โมเดล B

หน่วยความจำหลักนี้ทำงานตามจังหวะความถี่สัญญาณคลื่นที่สูง และตัวถังขนาดเล็กมีพื้นผิวสัมผัสกับอากาศได้น้อย ความร้อนที่ตัวถังจึงเป็นอุปสรรคต่อการทำงาน ดังนั้น ผู้อ่านควรติดตั้งฮีทซิงค์ (Heat Sink) เพื่อเพิ่มพื้นที่ผิวและระบายความร้อนออกจากชิปนี้ ตามการทดลองที่ 2 ภาคผนวก B การติดตั้งและใช้งานชาร์ดแวร์

ชิพหน่วยความจำหลัก SDRAM นี้ผลิตโดยบริษัท Elpida รุ่น B8132B4PB-8D-F วางอยู่ด้านล่างของบอร์ด Pi3 โมเดล B ดังรูปที่ 3.4 ขาสัญญาณต่างๆ จึงช่อนอยู่ใต้ชิพ เพื่อลดขนาดฟุตพรินต์ (Foot Print) บนแผ่นวงจรพิมพ์ โดยภายในชิปประกอบด้วย แผงอะเรย์ DRAM จำนวน 2 ชุดๆ ละ 8 ชิ้นๆ ละ 16×32 เมกะเซล คิดเป็น 32 เมกะเซล $\times 16$ บิท $\times 8$ ชิ้น $\times 2$ ตาย (die) ต่อชิพ ตามวิธีการคำนวณต่อไปนี้

$$2^5 \times 2^{20} \times 2^4 \times 2^3 \times 2^1 = 2^{33} = 2^3 \times 2^{30} = 8 \text{ จิกะบิท} = 1 \text{ กิกะไบต์}$$

ปัจจุบันนี้ บริษัท Elpida ได้เปลี่ยนผู้ถือหุ้นหลักเป็น บริษัท Micron Technology และ ดังนั้น ไดนา มิคแรมหมายเลขอุปกรณ์ EDB8132B4PB จึงสามารถค้นคว้าเพิ่มเติมได้ตามลิงค์ต่อไปนี้ www.micron.com

หนังสือเล่มนี้จะอธิบาย การทำงานและรายละเอียดเพิ่มเติมของหน่วยความจำ SDRAM นี้และชนิดต่างๆ ในบทที่ 5 ในหัวข้อที่ 5.5.1

3.1.3 อุปกรณ์อินพุท/เอาท์พุท (Input/Output Devices)

หัวข้อนี้จะกล่าวเฉพาะอุปกรณ์อินพุทและเอาท์พุทที่จำเป็น ได้แก่ คีย์บอร์ด เม้าส์ จอモนิเตอร์ เพื่อใช้งานเป็นเครื่องคอมพิวเตอร์ตั้งโต๊ะส่วนตัว (Desktop Personal Computer) และทำการทดลองต่างๆ ส่วนอุปกรณ์อินพุทและเอาท์พุทอื่นๆ ในบอร์ด Pi3 ผู้อ่านสามารถอ่านเพิ่มเติมในบทที่ 2 ได้อย่างละเอียด

3.1.3.1 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)	75	80	85				
30	31	32	33	34	35	36	37	38	39	40	41	42	43	90	95	100	105			
44	45	46	47	48	49	50	51	52	53	54	55	56	57	76	81	86	91	96	101	106
58		60	131		61	132		133	62		64			83			92	97	102	
														79	84	89	93	98	103	108
														(94)	99	104				(109)

106-Key Keyboard Position Codes

Figure 3.5: ตำแหน่งและรหัสประจำปุ่ม (Scan Code หรือ Position Code) บนคีย์บอร์ดชนิด 106 ปุ่ม ที่มา: ps-2.key009.com

- ปุ่มป้อนข้อมูลตัวเลข โดยจะมีสแกนโค้ดเท่ากับ 90-109 สำหรับการป้อนข้อมูลที่เป็นตัวเลขเรียงตัวกันทางขวาสุดของคีย์บอร์ด เพื่อความสะดวกต่อการใช้งานสำหรับนักบัญชี หรือ ผู้ที่กรอกข้อมูลบอยๆ
 - ปุ่มฟังค์ชัน ประกอบด้วย ปุ่ม F1 ถึง F12 ซึ่งระบบและแอพพลิเคชันบางตัวสามารถใช้เป็นคีย์ทางลัด (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) นิยมใช้เคลื่อนหรือเปลี่ยนตำแหน่งของเมาเซอร์เซอร์ (Cursor) บนหน้าจอแสดงผลทั้งในโหมดกราฟิกและตัวอักษร โดยจะมีสแกนโค้ดเท่ากับ

เมื่อผู้ใช้กดปุ่มตั้งแต่ 1 ปุ่มขึ้นไป คีย์บอร์ดจะส่งรหัสของปุ่มที่ถูกกดนั้น 送往คีย์บอร์ด จะส่งสแกนโค้ดของปุ่มที่โดนกดตามที่ได้ออกแบบไว้ ตามรูปที่ [3.5](#) ผ่านสายไปยังพอร์ต USB เป็นหลัก หรืออาจจะเป็นคีย์บอร์ดไร้สาย เช่น ชนิดคลื่นวิทยุความถี่ต่ำ ชนิดคลื่นบลูทูธ เป็นต้น ไปยังระบบปฏิบัติ เพื่อให้โปรแกรมตอบสนองต่อรหัสนั้นๆ เมื่อผู้ใช้กดปุ่มใดๆ ก็ครั้งกีตาม ระบบปฏิบัติการจะรับรู้ได้ผ่านกลไกการเกิดอินเทอร์รัพท์ (Interrupt) ซึ่งมีรายละเอียดเพิ่มเติมในการทดลองที่ [11](#) ภาคผนวก [K](#)

3.1.3.2 เม้าส์ (Mouse)

มาส์ คือ อุปกรณ์ที่สามารถเปลี่ยนการเคลื่อนที่ของข้อมูลในแกน 2 มิติเป็นการเคลื่อนที่ของพอยท์เตอร์ (Pointer) การเคลื่อนที่ของมาส์จะสัมพัทธ์กับตำแหน่งปัจจุบันของพอยเตอร์บนจอแสดงผล ทั้งนี้ขึ้นอยู่กับความละเอียดของมาส์เอง ความละเอียดของหน้าจอแสดงผล และแอพพลิเคชันนั้นๆ เช่น เกมส์ โปรแกรม วิดีโอและตัวแปรต่างๆ จะต้องการความละเอียดของมาส์สูงกว่ามาส์ปกติ เป็นต้น

เม้าส์ส่วนใหญ่มีจำนวนปุ่มคลิกทางซ้ายและขวา ขึ้นอยู่กับการออกแบบของระบบໂຄເອສ มี 1-3 ปุ่ม สำหรับการใช้งานปัจจุบัน มีการเพิ่มวงล้อ (Wheel) บนเม้าส์สำหรับการเคลื่อนแบบพิเศษในมิติต่างๆ การคลิกปุ่มซ้าย (Left Click) ปุ่มขวา (Right Click) การคลิกปุ่มซ้ำ (Double Click) และ การสัมผัสขึ้นหรือ

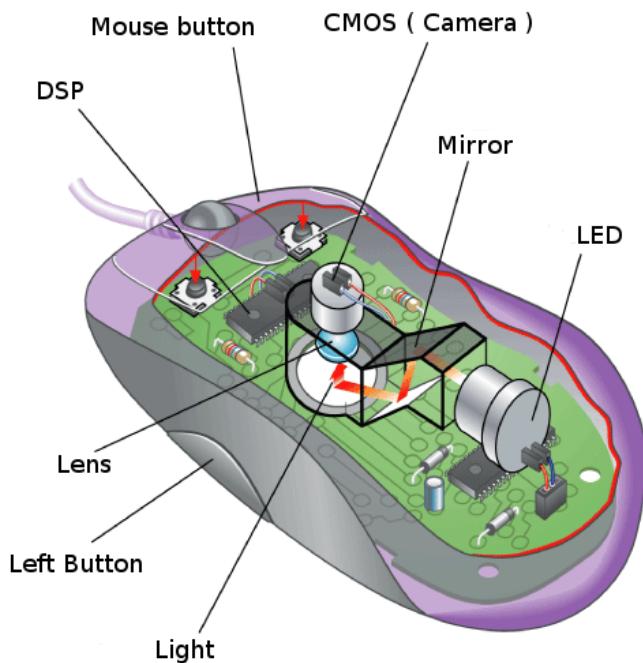


Figure 3.6: โครงสร้างภายในมาส์ชนิด Optical ประกอบด้วยหลอด LED ส่องแสงกระแทบกับพื้นผิวด้านล่างแล้วสะท้อนกลับมายังตัวรับแสงชนิด CMOS ที่มา: www.pinterest.com

ลงบนปุ่มมาส์ จะทำให้เกิดแอคชั่นต่างๆ ตามระบบปฏิบัติการ และโปรแกรมที่ใช้งาน ดังนั้น ผู้ใช้จะต้องใช้ความรู้ ความเข้าใจ ตลอดจนถึงความจำ เพื่อให้สามารถใช้งานโปรแกรมเดียวกันแต่ทำงานบนระบบปฏิบัติการที่ต่างกันได้

รูปที่ 3.6 แสดง โครงสร้างภายในมาส์ชนิดออพติคอล (Optical) ประกอบด้วยหลอดแอลอีดี (LED) ส่องแสงกระแทบกับพื้นผิวด้านล่างก่อน แล้วจึงสะท้อนกลับมายังตัวรับแสงชนิด CMOS แสงจากหลอด LED มีลักษณะที่แตกต่างกันไปตามชนิดและราคาของมาส์ เช่น หลอด LED เป็นแสงธรรมชาติซึ่งผู้ใช้มองเห็น และหลอด LED แสงเลเซอร์ (Laser) ซึ่งมองไม่เห็นด้วยตาเปล่า แต่มีความละเอียดเพิ่มสูงขึ้นมาก เพื่อประยุกต์การใช้แรงและเพิ่มความแม่นยำในการเคลื่อนที่ของมาส์ ซึ่งแสงเลเซอร์จะให้ความละเอียดในการใช้งานสูงกว่า เหมาะกับซอฟต์แวร์ด้านกราฟิก และเกมส์ ด้วยความละเอียด 800-6,000 จุดต่อรับบททางหนึ่งนิ้ว (DPI: Dot Per Inch)

เมื่อผู้ใช้กดปุ่มใดๆ หรือขยับมาส์ ระบบปฏิบัติการจะรับรู้ได้ผ่านกลไกการเกิดอินเทอร์รัพท์ (Interrupt) เช่นเดียวกับคีย์บอร์ด ซึ่งมีรายละเอียดเพิ่มเติมในการทดลองที่ 11 ภาคผนวก K นอกจากนี้จากมาส์ อุปกรณ์อื่นๆ ที่ใกล้เคียงสามารถทดแทนการใช้มาส์ได้ เช่น ทัชแพด (Touch Pad) จะแสดงผลระบบสัมผัส เสียงพูด ท่าทางของมือและนิ้วที่ตรวจจับโดยกล้องวีดีโอ เป็นต้น ทำให้การใช้งานคอมพิวเตอร์ง่ายลง และได้ประสบการณ์เป็นธรรมชาติมากขึ้น สาขาด้าน Human Computer Interface จึงมีความสำคัญเพิ่มมากขึ้นเรื่อยๆ

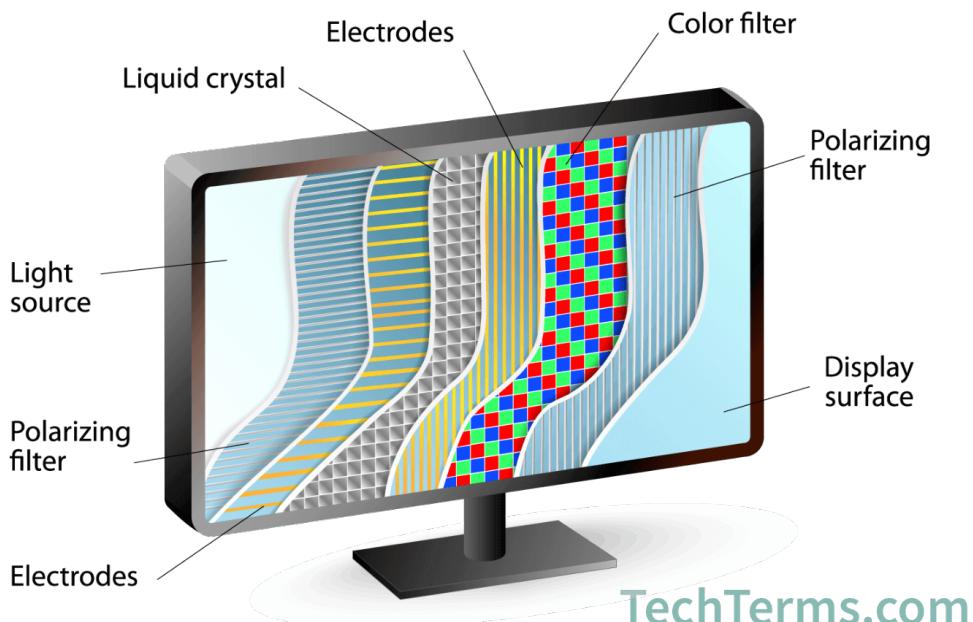


Figure 3.7: โครงสร้างจอแสดงผลชนิด Color LCD ประกอบด้วยแหล่งกำเนิดแสง (Light Source) ปล่อยแสงหลักชั้นต่างๆ มาแสดงผล ที่มา: techterms.com

3.1.3.3 จอภาพ LCD (Liquid Crystal Display)

จอภาพ LCD ทำหน้าที่แสดงผลตัวอักษรและกราฟิกทั้งในรูปของ ภาพนิ่ง ภาพเคลื่อนไหว และเกมส์ เพื่อโต้ตอบและสร้างปฏิสัมพันธ์กับผู้ใช้งาน ซึ่งในอดีตจอภาพของคอมพิวเตอร์สามารถแสดงผลได้เฉพาะตัวอักษรเท่านั้น

โครงสร้างจอภาพ LCD สี (Color LCD) ประกอบด้วยหลอด LED ด้านหลัง (LED Back Light) ของจอ ปล่อยแสงหลักชั้นต่างๆ มาแสดงผลทางด้านหน้า ตามรูปที่ 3.7 ความละเอียดในการแสดงผล นับตามจำนวนจุดภาพหรือพิกเซล (Pixel) นอกจากความละเอียดหรือจำนวนจุดภาพแล้ว คุณภาพการแสดงผลขึ้นกับ จำนวน ความสว่างและการจัดเรียงตัวของหลอด LED Back Light การเรียงตัวของจุดภาพอยู่ (RGB: Red Green Blue) ในแต่ละพิกเซล ขนาดของพื้นผิวน้ำยา แสดงองศาของมุมมองภาพ (Angle) ระยะเวลาในการตอบสนองต่อการสแกน (Response Time) การเชื่อมต่อกับบอร์ดด้วยสายภายนอก HDMI หรือ DVI หรือ RGB ฯลฯ

หลักการพื้นฐาน คือ จุดภาพ 1 จุดบนจอแสดงผล LCD ประกอบด้วย ชั้วไฟฟ้านิดโปร่งแสง (Transparent Electrode) ส่องขาว และการเรียงตัวของผลึกคริสตัลในของเหลว (Liquid Crystal) เมื่อมีสนามไฟฟาระหว่างชั้วไฟฟ้าทั้งด้านหน้าและด้านหลัง จะทำให้ผลึกเกิดการบิดตัวพาเคลื่อนแสงจากเลนส์โพลาไรซ์ (Polarize) ด้านหลัง ทະลุผ่านเลนส์โพลาไรซ์ด้านหน้า ทำให้ผู้ใช้มองเห็นความสว่างของจุดนั้น เมื่อไม่มีประจุไฟฟ้า ผลึกจะไม่บิดเกลี้ยง ทำให้แสงหลักไปเม็ดเดียว ผู้ใช้จะเห็นจุดนั้นเป็นสีเดียว ที่เกิดขึ้นบนจอ เกิดจากการผสมกันของจุดภาพอยู่ (Sub Pixel) 3 จุดๆ ละสี คือ แดง เขียว น้ำเงิน ด้วยระดับความสว่างที่ไม่เท่ากัน ทำให้เกิดการผสมของสีที่หลากหลายได้ตามจำนวนบิทข้อมูลสี ซึ่งปัจจุบัน ใช้ข้อมูลสีอย่างน้อย 8 บิต ทำให้เกิดการผสมของสีทั้งหมดคิดเป็น $2^{3 \times 8} = 16,777,216$ หรือ 16.78 ล้านสี

การทดลองที่ 2 ภาคผนวก B การติดตั้งและใช้งานฮาร์ดแวร์ จะเปิดให้ผู้อ่านได้ประกอบบอร์ด Pi3 เข้ากับจอภาพ LCD และอุปกรณ์อินพุตเอาท์พุทอื่นๆ เพื่อใช้เป็นคอมพิวเตอร์ตั้งโต๊ะส่วนบุคคลประกอบการ

ทดลองที่เหลือ

3.1.4 อุปกรณ์เก็บรักษาข้อมูล (Data Storage)



Figure 3.8: โครงสร้างของอุปกรณ์เก็บรักษาข้อมูล SD ชนิด SDHC ความจุ 16 GB ประกอบด้วยชิปหน่วยความจำแฟลช วงจรควบคุม และวงจรเชื่อมต่อ ที่มา: [wikipedia](#)

การ์ดหน่วยความจำ SD (Secure Digital) ถูกพัฒนาขึ้นมาโดยองค์กร ชื่อ SD Card Association (SDA) สำหรับใช้งานกับอุปกรณ์เคลื่อนที่ต่างๆ เช่น กล้องถ่ายรูป โทรศัพท์ เครื่องเล่นเพลง เป็นต้น ในเดือนสิงหาคม 1999 โดยความร่วมมือกันของ บริษัท SanDisk Panasonic และ Toshiba ในเชิงความจุ (Capacity) ของหน่วยความจำ SD แบ่งเป็นสี่รุ่น ได้แก่ Standard-Capacity (SDSC), High-Capacity (SDHC), eXtended-Capacity (SDXC), และ SDIO ซึ่งรวมฟังก์ชัน input/output กับการบันทึกข้อมูลเข้าด้วยกัน โดยราคาในท้องตลาดจะแปรผันตามความจุ ความจุของ SD มีแนวโน้มเพิ่มขึ้นตามวัตถุประสงค์ การใช้งาน เนื่องจากตัวการ์ดหน่วยความจำมีขนาดเล็ก นำหันกabe ความเร็วในการอ่านหรือเขียนที่รวดเร็ว ทนทาน และอายุการใช้งานยาวนาน

ในทางกายภาพหน่วยความจำ SD มี 3 ขนาด ขนาดปกติ ขนาดมินิ และขนาดไมโคร โดยมีตัวแปลงให้เป็นขนาดปกติได้ โครงสร้างทางกายภาพของอุปกรณ์เก็บรักษาข้อมูลชนิด SD Card นี้ จะมีกระดิ่งเพื่อช่วยล็อกกับสล็อตที่เสียบ องค์ประกอบภายในของหน่วยความจำ SD แสดงในรูปที่ 3.8 คือ ชิปหน่วยความจำแฟลช ผลิตโดยบริษัท Samsung หนังสือเล่มนี้จะกล่าวอธิบาย รายละเอียดของหน่วยความจำแฟลช เพิ่มเติมโดยละเอียด ในบทที่ 7

บอร์ด Pi3 โมเดล B ใช้อุปกรณ์เก็บรักษาข้อมูลชนิด micro SD สำหรับระบุระบบปฏิบัติการ บันทึก และเก็บรักษาข้อมูลต่างๆ โดยผู้ใช้สามารถติดตั้งระบบปฏิบัติการ เช่น Raspbian (Linux), Ubuntu Linux, Microsoft Windows 10 IoT เป็นต้น ลงบนการ์ด micro SD นี้ ซึ่งผู้อ่านสามารถติดตั้ง (Install) ระบบปฏิบัติการ Raspbian ลงในอุปกรณ์เก็บรักษาข้อมูลชนิดหน่วยความจำ SD หัวข้อที่ 7.3 ในการทดลองที่ 3 ภาคผนวก C

3.2 ซอฟต์แวร์หรือโปรแกรมคอมพิวเตอร์

โครงสร้างทางซอฟต์แวร์ของคอมพิวเตอร์ทั่วไป แบ่งเป็น

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

โดยสรุป ซอฟต์แวร์ คือ ชุดคำสั่งที่คอยตรวจสอบและสั่งงานฮาร์ดแวร์ โดยเฉพาะซีพียูให้ทำงานตามที่โปรแกรมเมอร์เขียน (Code) หรือพัฒนา (Develop) โดยใช้ภาษาระดับต่างๆ รายละเอียดจะได้กล่าวในหัวข้อที่ 3.3

เมื่อผู้ใช้กดปุ่มเปิดเครื่องเพื่อจ่ายไฟเลี้ยงให้กับคอมพิวเตอร์ ระบบฮาร์ดแวร์จะเริ่มทำงาน เพื่อตรวจสอบและเตรียมความพร้อม หลังจากนั้น ขั้นตอนการทำงานของซอฟต์แวร์จะแบ่งเป็น

- การบูทรับบปฏิบัติการจากอุปกรณ์เก็บรักษาข้อมูลเข้าสู่หน่วยความจำหลัก ในหัวข้อที่ 3.2.1
- การโหลดไฟล์แอพพลิเคชันจากอุปกรณ์เก็บรักษาข้อมูลเข้าสู่หน่วยความจำหลัก ในหัวข้อที่ 3.2.2
- การอ่านคำสั่งจากหน่วยความจำหลักไปปฏิบัติตาม ในหัวข้อที่ 3.2.3
- การอ่าน/เขียนข้อมูลระหว่างหน่วยความจำหลักไปประมวลผล ในหัวข้อที่ 3.2.4
- การเชื่อมต่ออุปกรณ์อินพุตต่างๆ เช่น คีย์บอร์ด เม้าส์ เครื่อข่ายอินเทอร์เน็ต เป็นต้น ในหัวข้อที่ 3.2.5
- การอ่าน/เขียนข้อมูลระหว่างหน่วยความจำหลักและอุปกรณ์เก็บรักษาข้อมูล ในหัวข้อที่ 3.2.6
- การซัพเดต (Shut Down) ระบบปฏิบัติการก่อนปิดเครื่อง ในหัวข้อที่ 3.2.7

ขั้นตอนดังกล่าวจะแตกต่างกันออกไปตามรายละเอียดของฮาร์ดแวร์และระบบโอเอสที่ใช้ แต่หลักการโดยรวมจะมีความคล้ายคลึงกัน

ตัวอย่างเช่นที่ระบบปฏิบัติการ Raspbian ซึ่งเป็น Linux เวอร์ชันหนึ่งที่โครงสร้างสำคัญตามรูปที่ 3.9 ประกอบด้วย Kernel Space และ User Space โครงสร้างของ Kernel Space ประกอบด้วย

- Kernel คือ โปรแกรมหลักของระบบปฏิบัติการ ซึ่งปัจจุบันได้มีการพัฒนาเป็นเวอร์ชัน 4.20 ผู้อ่านสามารถตรวจสอบเวอร์ชันล่าสุดได้ที่ www.kernel.org
- Architecture Dependent Kernel Code สำหรับ Kernel เชื่อมต่อกับฮาร์ดแวร์เฉพาะ และ
- System Call Interface สำหรับ Kernel เชื่อมต่อกับซอฟต์แวร์ประยุกต์

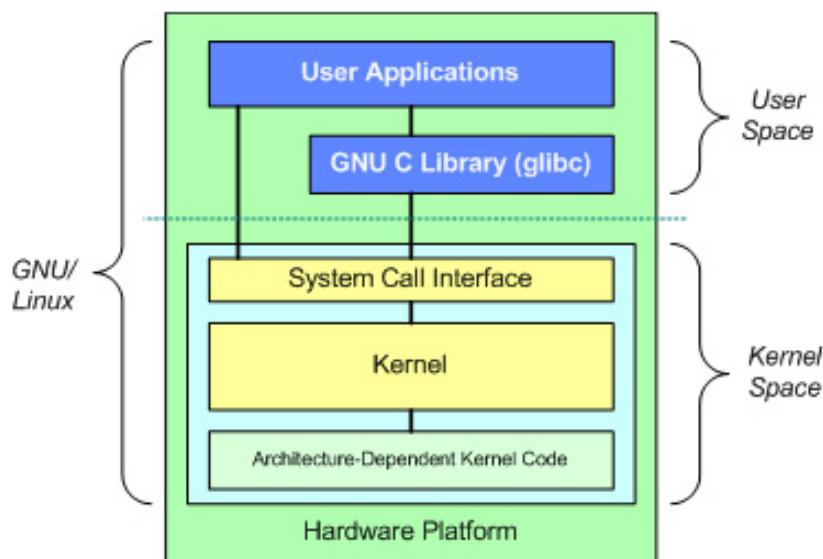


Figure 3.9: โครงสร้างหน่วยความจำเสมือนเมื่อทำการบุทระบบปฏิบัติการ Linux สำเร็จ โดยแบ่งเป็น Kernel Space สำหรับ Linux เองและ User Space สำหรับแอพพลิเคชัน ที่มา: linux-india.org

โครงสร้างของ User Space ประกอบด้วย User Application หรือซอฟต์แวร์ประยุกต์ต่างๆ ได้แก่ โปรแกรมที่ใช้งานประเภทต่างๆ เช่น เว็บเบราว์เซอร์ (Web Browser) เช่น โมซิล่าไฟร์ฟ็อกซ์ (Mozilla Firefox) เกมประเภทต่างๆ เป็นต้น ซึ่งโปรแกรมเมอร์พัฒนาซอฟต์แวร์ประยุกต์เหล่านี้จะถูกนำไปในรูปของฟรีแวร์ (Freeware) และฟรีแวร์บางตัวมีการเปิดเผยแพร่โค้ด เรียกว่า **Open Source** เพื่อให้โปรแกรมเมอร์ต่างๆ ทั่วโลก สามารถพัฒนาเสริมเพิ่มเติมได้ ตามเงื่อนไขของการเปิดเผยแพร่โค้ด เช่น GPL (GNU General Public License), LGPL (Lesser GPL) เป็นต้น

ซอฟต์แวร์ประยุกต์เกือบทุกตัวจะทำงานใน User Space ของผู้ใช้ที่กำลังล็อกอิน (Log In) โดย ซอฟต์แวร์ประยุกต์ จะทำงานได้ต้องอาศัยพื้นที่ในหน่วยความจำหลักและความยินยอม (Permission) จากระบบปฏิบัติการ เว็บเบราว์เซอร์สามารถเชื่อมต่อกับเครือข่ายและทรัพยากรื่นๆ ผ่านทาง System Call Interface เพื่อให้ Kernel ติดต่อกับทรัพยากรเหล่านั้นแทน การทำงานนี้จะดำเนิน Kernel Space

การทดลองที่ 3 ภาคผนวก C จะแนะนำการติดตั้งระบบปฏิบัติการชื่อว่า Raspbian ซึ่งเป็นลีนุกซ์ที่พัฒนาสำหรับใช้งานบนบอร์ด Pi3 ไมเดล B นี้ การทดลองที่ 4 ภาคผนวก D จะแนะนำการใช้งานคำสั่งพื้นฐานในระบบลีนุกซ์และยูนิกซ์ (Unix) เป็นต้น ซึ่งยูนิกซ์เป็นโอเอสที่มีประวัติศาสตร์ยาวนานและเป็นต้นแบบของโอเอสต่างๆ ในวิทยาการคอมพิวเตอร์ และการทดลองที่ 5 จะแนะนำการพัฒนาโปรแกรมด้วยภาษา C สำหรับยูนิกซ์

3.2.1 การบุทระบบปฏิบัติการจากอุปกรณ์เก็บรักษาข้อมูลเข้าสู่หน่วยความจำหลัก

การบุทระบบโอเอส คือ การเริ่มต้นใช้งานเครื่องคอมพิวเตอร์ โดยหลักการชาร์ดแวร์จะอ่านไฟล์ kernel.img จากอุปกรณ์เก็บรักษาข้อมูลเข้าสู่หน่วยความจำหลัก ขั้นตอนและรายละเอียดการบุทจะแตกต่างกันไปตามเงื่อนไขของชาร์ดแวร์และระบบปฏิบัติการ สำหรับระบบปฏิบัติการ Raspbian บน

บอร์ด Pi3 มีขั้นตอนโดยละเอียด ดังนี้

1. เมื่อเปิดเครื่อง หรือ จ่ายไฟให้บอร์ด Pi3 ชิปปี้ ARM Cortex A53 จะยังไม่ทำงาน แต่ GPU จะใช้ชิปปี้บูตloader ซึ่งฝังอยู่ภายในชิป BCM2837
2. GPU ส่งงานฮาร์ดแวร์ที่ควบคุมอุปกรณ์เก็บรักษาข้อมูล SD Card เพื่อมองหา파ร์ติชันที่ฟอร์แมทด้วยรูปแบบ FAT32 ภายในการ์ดหน่วยความจำ SD นั้น
3. เมื่อเจอพาร์ติชันแล้ว GPU อ่านชุดคำสั่งจากไฟล์ชื่อ bootcode.bin ซึ่งทำหน้าที่เป็น Boot Loader จากไฟล์เดอร์ /bin โดยในระหว่างนี้จะยังไม่มีการใช้งานหน่วยความจำหลัก SDRAM บนบอร์ด Pi3
4. GPU เริ่มต้นอ่านไฟล์ start.elf ในหน่วยความจำ SD ไปเก็บใน RAM เพื่อให้ชิปปี้เริ่มทำงาน โดยอาศัยไฟล์ fixup.dat ซึ่งมีข้อมูลการจัดแบ่งพาร์ติชันภายในระหว่าง GPU และชิปปี้
5. ชิปปี้เริ่มต้นทำงานตามคำสั่งภายใน start.elf จากหน่วยความจำหลัก แล้วจึงอ่านไฟล์ kernel.img ไปบรรจุใน RAM ตามรายละเอียดในไฟล์ config.txt เพื่อติดตั้งค่าของระบบ
6. ชิปปี้เริ่มต้นอ่านและรันคำสั่งจาก kernel.img ตามพารามิเตอร์ที่บรรจุอยู่ใน cmdline.txt โดยโมดูล init จะเริ่มต้นทำงานตามระดับที่ตั้งค่าอยู่ในไฟล์ /etc/inittab ระดับการรัน (Run Level) ปกติจะตั้งค่าไว้ที่ระดับ 3 หรือ ระดับ 5 ตารางที่ [3.2](#) แสดงระดับการรัน (Run Level), โหมดการทำงานของระบบ และชื่อไฟล์เดอร์ที่บรรจุไฟล์โปรแกรมและไฟล์ข้อมูลสำหรับการบูต

Table 3.2: ระดับการรัน (Run Level), โหมดการทำงานของระบบ และชื่อไฟล์เดอร์ที่บรรจุไฟล์โปรแกรม และไฟล์ข้อมูลสำหรับการบูตบอร์ด Pi3

ระดับการรัน	โหมดการทำงาน	ชื่อไฟล์เดอร์
0	Halt (ปิดเครื่อง)	/etc/rc.d/rc0.d/
1	Single user (ผู้ใช้คนเดียว)	/etc/rc.d/rc1.d/
2	Multiuser mode without NFS (ผู้ใช้หลายคน ยกเว้น NFS)	/etc/rc.d/rc2.d/
3	Full Multiuser mode (ผู้ใช้หลายคนและ NFS)	/etc/rc.d/rc3.d/
4	Unused (ไม่ใช้)	/etc/rc.d/rc4.d/
5	X11 (กราฟิก)	/etc/rc.d/rc5.d/
6	Reboot (ปิดแล้วเปิดเครื่อง)	/etc/rc.d/rc6.d/

เมื่อบูตระบบสำเร็จ kernel จะครอบครองฮาร์ดแวร์ทั้งหมด และทำงานร่วมกับไฟล์ในไฟล์เดอร์ต่างๆ ของอุปกรณ์เก็บรักษาข้อมูล SD ซึ่งภายในประกอบด้วยไฟล์เดอร์หรือไดเรกทอรีในระบบปฏิบัติการลีนุกซ์ หรือ ยูนิกซ์ ดังรูปที่ [3.10](#) เรียงตามลำดับความสำคัญดังนี้

1. /root: เป็นไฟล์เดอร์หลักของผู้ใช้งานหลักของระบบ (root)
2. /boot: บรรจุโปรแกรม bootloader สำหรับบูตตัวเครื่องเนล
3. /bin: บรรจุไฟล์ ELF ของระบบ เช่น คำสั่งต่างๆ สำหรับผู้ใช้ส่วนใหญ่
4. /sbin: บรรจุไฟล์คำสั่งต่างๆ ของระบบ เพื่อให้ผู้ใช้งานที่เป็น administrator หรือ root เท่านั้น

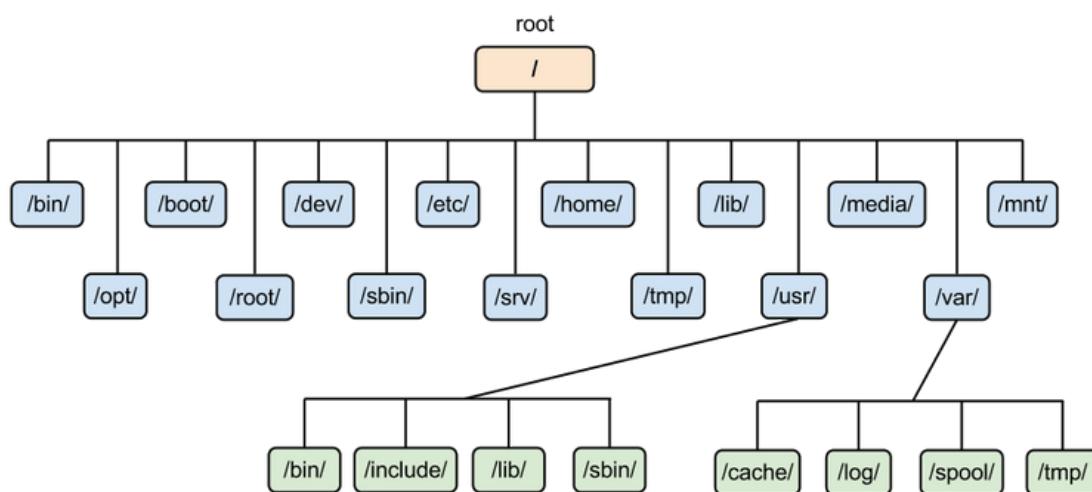


Figure 3.10: โครงสร้างของโฟลเดอร์ (Folder) หรือไดเรคทอรี (Directory) สำคัญๆ ในระบบปฏิบัติการ Linux ที่มา: freedompenguin.com

5. /lib: เป็นโฟลเดอร์สำหรับแชร์ไฟล์ต่างๆ ของระบบ
6. /sys: เป็นระบบไฟล์เสมือน (Virtual filesystem) เพื่อให้โปรแกรมเมอร์เข้าถึงอุปกรณ์ชาร์ดแวร์ เช่น ขา GPIO ต่างๆ ซึ่งจะกล่าวในการทดลองที่ 12 ภาคผนวก | หัวข้อที่ ??
7. /dev: เป็นโฟลเดอร์ที่ถูกสร้างใหม่ทุกครั้งที่บูทเครื่อง เพื่อเก็บไฟล์ที่เป็นตัวแทนของอุปกรณ์อินพุต/เอาท์พุตต่างๆ ใน User Space รายละเอียดเพิ่มเติมสามารถอ่านได้จากคำสั่ง ls /dev
8. /etc: ใช้เก็บไฟล์สำหรับบรรจุไฟล์คอนฟิกของระบบ และแอพพลิเคชันต่างๆ
9. /home: เป็นพื้นที่ส่วนตัวสำหรับผู้ใช้แต่ละคน สำหรับเก็บไฟล์ต่างๆ โดยใช้ชื่อโฟลเดอร์ตามชื่อผู้ใช้ภายในโฟลเดอร์ผู้ใช้แต่ละคนแบ่งเป็นพื้นที่หรือโฟลเดอร์ต่างๆ เช่น Documents Desktop Downloads เพื่อใช้เป็นที่จัดเก็บโฟลเดอร์และไฟล์ส่วนตัวของผู้ใช้แต่ละคน ซึ่งจะแยกตามชื่อ ล็อกอิน เช่น /home/user1 /home/user2 เป็นต้น โดย user1 และ user2 คือ ชื่อล็อกอิน ตามลำดับ
10. /var: เป็นโฟลเดอร์สำหรับเก็บล็อก และข้อมูลอื่นๆ ที่เกิดขึ้นระหว่างการรันระบบ
11. /mnt: เป็นจุดมาท์ (Mount) ชั่วคราว เช่น การแชร์ไฟล์ผ่านเครือข่าย
12. /media: เป็นจุดมาท์หลักสำหรับสื่อเคลื่อนที่ (Removable device) ต่างๆ เช่น แฟลชไดร์ ซีดี/ดีวีดีรอม เป็นต้น
13. /opt: เป็นโฟลเดอร์สำหรับติดตั้งแอพพลิเคชันจากผู้ขาย
14. /proc: เป็นเวอร์ชวลไดเรคทอรี สำหรับแต่ละโพรเซสในระบบ ยกตัวอย่างเช่น /proc
15. /run: เป็นโฟลเดอร์ชั่วคราวสำหรับแอพพลิเคชันในระหว่างที่รันอยู่
16. /tmp: ใช้สำหรับบรรจุไฟล์ชั่วคราวจากแอพพลิเคชันและระบบ

3.2.2 การโหลดซอฟต์แวร์ประยุกต์จากไฟล์ ELF เข้าสู่หน่วยความจำหลัก

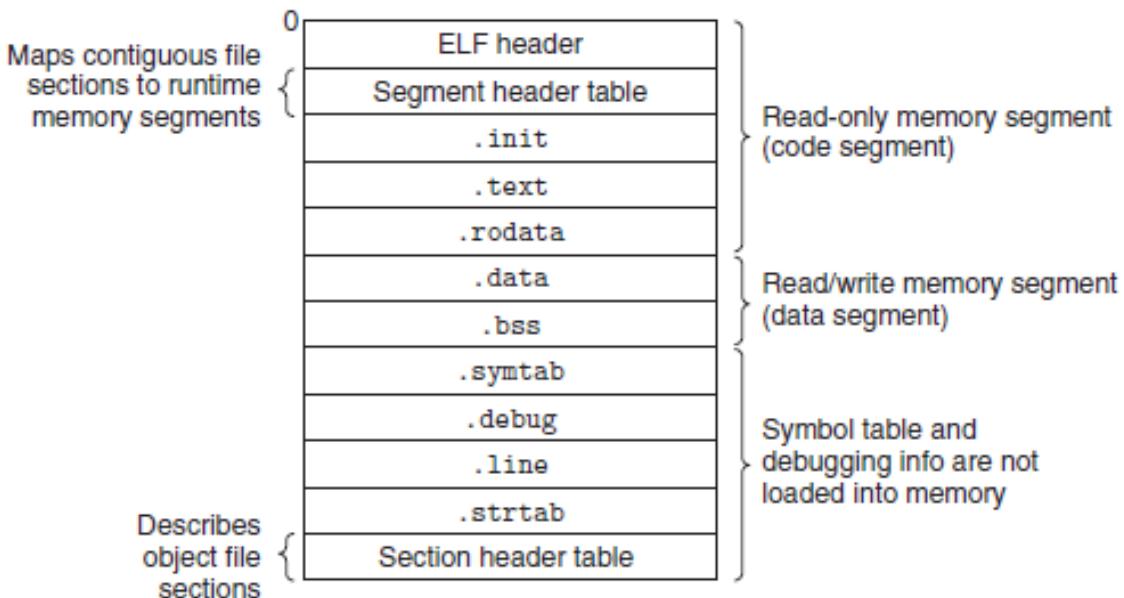


Figure 3.11: โครงสร้างไฟล์ชนิด ELF สำหรับเก็บชุดคำสั่งและข้อมูลในอุปกรณ์เก็บรักษาข้อมูล ที่มา: [wikipedia](#)

เมื่อเครื่องคอมพิวเตอร์พร้อมใช้งานหรือบูทเครื่องสำเร็จแล้ว ผู้ใช้จึงสามารถเริ่มต้นเรียกใช้งานซอฟต์แวร์ประยุกต์ได้ ซึ่งสามารถทำได้ตามค่านอนใน การทดลองที่ 4 ภาคผนวก D คำสั่งหรือแอพพลิเคชันเหล่านี้คือ ไฟล์รูปแบบ ELF ที่เก็บชุดคำสั่งและข้อมูลประกอบการทำงานเอาไว้ตามระบบปฏิบัติการยุนิกซ์ ตามรูปที่ 3.11 ส่วนตระกูล Microsoft Windows ไฟล์โปรแกรมจะอยู่เปลี่ยนตามรูปแบบ EXE ไฟล์เหล่านี้อยู่ในอุปกรณ์เก็บรักษาข้อมูล ดังนั้น ก่อนที่โอเอจะรันคำสั่งหรือแอพพลิเคชันได้ โอเอจะต้องอ่านหรือโหลดไฟล์คำสั่งหรือแอพพลิเคชันนั้นๆ จากอุปกรณ์เก็บรักษาข้อมูลเข้าสู่หน่วยความจำหลัก

ELF ย่อมาจาก Executable and Linkable Format เป็นชนิดและโครงสร้างของไฟล์โปรแกรม (Executable) และไฟล์อ็อบเจกต์ (Object) ที่ได้จากการคอมไพล์ ELF มีความยืดหยุ่น ขยายเพิ่มเติมได้ง่าย และรองรับการข้ามแพลทฟอร์ม (Cross Platform) ระหว่างไมโครโปรเซสเซอร์ และสถาปัตยกรรมของชุดคำสั่งภาษาเครื่องหรือแมชชีนโค้ด โครงสร้างของไฟล์ ELF ตามรูปที่ 3.11 ประกอบด้วยพื้นที่หลายส่วน และเซกเมนต์ (Segment) ต่างๆ คือ

- ส่วนหัว (ELF Header) บ่งชี้ว่าจะอ้างถึงหน่วยความจำหลักด้วยเลขเบอร์เดรสขนาด 32 บิต หรือ 64 บิต ตามด้วยข้อมูลอื่นๆ ตามลำดับถัดไป แบ่งเป็น
 - หากใช้ระบบ 32 บิต ส่วนหัวนี้จะมีความยาว 52 ไบท์
 - หากใช้ระบบ 64 บิต ส่วนหัวนี้จะมีความยาว 64 ไบท์
- ตาราง Program header table อธิบายเซกเมนต์ (Segment) ต่างๆ ในไฟล์ ELF ว่าจะสร้างอย่างไร ที่ตำแหน่งใดในหน่วยความจำ
- เท็กซ์เซกเมนต์ (.text) ใช้สำหรับเก็บชุดคำสั่งหรือแมชชีนโค้ด

- ดาต้าเช็กเม้นท์ชนิดอ่านอย่างเดียว (.rodata: Read-Only Data) ใช้สำหรับเก็บข้อมูลที่เป็นค่าคงที่
- ดาต้าเช็กเม้นท์ (.data และ .data1) ใช้สำหรับเก็บข้อมูลหรือชื่อตัวแปรที่มีค่าตั้งต้น (Initialized Data) และอาจเปลี่ยนแปลงได้เมื่อทำงาน
- ตาราง Section header table อธิบายเซ็คชัน (Section) ต่างๆ ในหน่วยความจำ ว่าซึ่ออะไร มีความยาวเท่าไร

โครงสร้างของไฟล์ หน่วยความจำเพื่อจัดเก็บคำสั่งและข้อมูลต่างๆ รายละเอียดและข้อมูลเพิ่มเติม สามารถสืบค้นได้ทางลิงค์ต่อไปนี้ [wikipedia](#)

3.2.3 การอ่านคำสั่งของซอฟต์แวร์ประยุกต์จากหน่วยความจำหลักเพื่อไปปฏิบัติ ตาม

เมื่อออเอสอ่านไฟล์แอพพลิเคชันขึ้นมาแล้ว โอเอสจะสร้างพื้นที่ในหน่วยความจำเสมือนซึ่งได้แมป (Map) ให้ตรงกับตำแหน่งของแอพพลิเคชันนั้นในหน่วยความจำหลัก รายละเอียดเพิ่มเติมในบทที่ เมื่อโหลดไฟล์ สำเร็จแล้ว หน่วยความจำเสมือนของแอพพลิเคชันนั้น จะประกอบด้วยเซ็กเมนท์ต่างๆ ตามโครงสร้างของไฟล์ ELF ในรูปที่ 3.12 หลังจากนั้น โอเอสจะเปิดโอกาสให้แอพพลิเคชันนั้น ทำงานตามสิทธิ์ (Permission) ที่ได้รับ โดยซึ่พิจจะเริ่มต้นอ่านคำสั่งจากเทกซ์เซ็กเมนท์ก่อนเสมอ และคำสั่งนั้นจะบรรจุอยู่ในฟังค์ชัน ซึ่ง main ซึ่งขบวนการที่กล่าวมาแล้วนี้เกิดขึ้นเร็วมาก

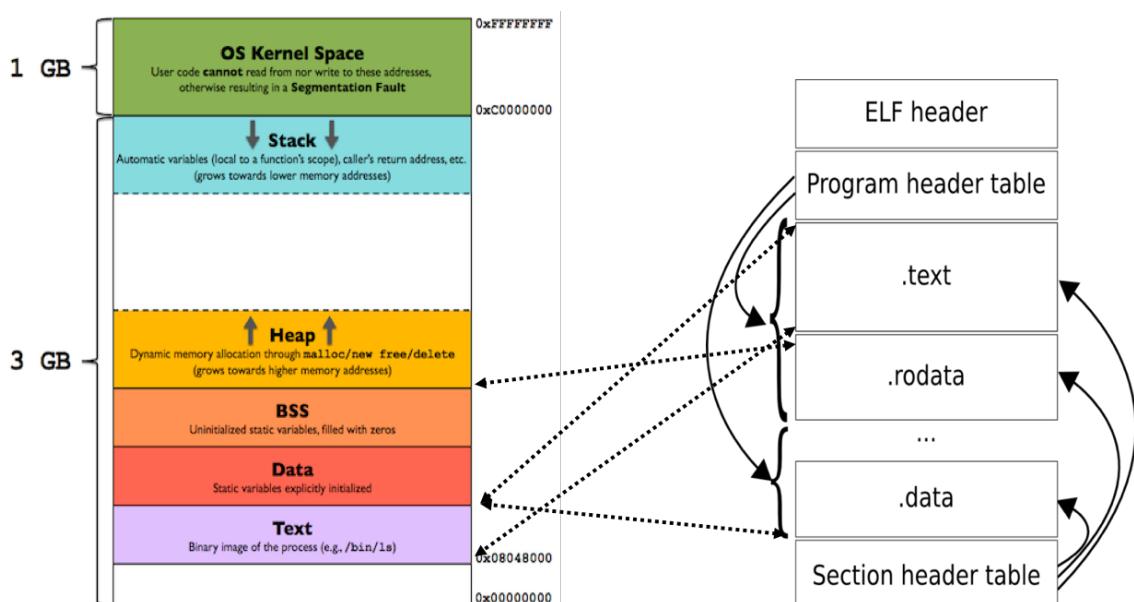


Figure 3.12: การบรรจุคำสั่งและข้อมูลจากแอพพลิเคชันหรือโปรแกรมซึ่งบรรจุอยู่ในไฟล์ชนิด ELF ไปยังหน่วยความจำเสมือนส่วนที่เป็น User Space ที่มา: [wordpress.com](#)

โดยตัวราเลมนี้จะครอบคลุมเนื้อหาของระบบปฏิบัติการขนาด 32 บิตเท่านั้น ซึ่งจะทำให้ระบบสร้างหน่วยความจำเสมือนขนาด 2^{32} หรือ 4 จิกะไบท์เท่านั้น Kernel เองเป็นโปรแกรมตัวหนึ่ง อาศัยพื้นที่หน่วยความจำเสมือนขนาด 1 จิกะไบท์ ตั้งแต่หมายเลข 0xC0000000 หรือ 0xC000_0000 ถึง 0xFFFFFFFF

หรือ 0xFFFF_FFFF เรียกว่า **Kernel Space** โปรแกรมแต่ละตัวจะได้รับพื้นที่หน่วยความจำเสมือน เรียกว่า **User Space** โดยพื้นที่หน่วยความจำส่วนใหญ่เลข 0x0000_0000 ถึง 0xBFFF_FFFF ขนาดรวม 3 กิกะไบต์ แบ่งเป็นเซกเมนท์ต่างๆ ดังนี้

- **เซกเมนท์ BSS** (“Block Started by Symbol”) คือ พื้นที่ส่วนหนึ่งในหน่วยความจำเสมือน สำหรับจัดเก็บค่าของตัวแปรชนิด global และ static ที่โปรแกรมเมอร์ยังไม่ได้ตั้งค่าเริ่มต้น แต่โอลูสจะทำการตั้งค่าเริ่มต้นให้กลายเป็น 0 โดยอัตโนมัติ ตำแหน่งของ BSS จะเริ่มต้นต่อเนื่องจากตำแหน่งสุดท้ายของ-data เชกเมนท์
- **เซกเมนท์สเต็ค (Stack Segment)** คือ พื้นที่ส่วนหนึ่งในหน่วยความจำเสมือน สำหรับให้โปรแกรมเมอร์ใช้เก็บค่าของตัวแปรชนิดโลคอล (Local Variable) บางระบบปฏิบัติการจะใช้พื้นที่ของสแต็คเก็บค่าตัวแปรในฟังก์ชันเพื่อส่งผ่าน (Pass) และคืนค่า (Return) พารามิเตอร์ระหว่างฟังก์ชันผู้เรียก (Caller Function) และฟังก์ชันผู้ถูกเรียก (Callee Function) ซึ่งพื้นที่สำหรับใช้งานในแต่ละฟังก์ชัน เรียกว่า **สแต็คเฟรม (Stack Frame)** มีสแต็คพอยท์เตอร์ (Stack Pointer) เก็บค่าเอດเดรสหน่วยความจำขั้นบนสุด เพื่อความสะดวกในการบริหารจัดการ

การบริหารสแต็คเฟรมของแต่ละฟังก์ชัน จะมีลักษณะเป็น LIFO (Last In First Out) นั่นคือ ในตอนเริ่มต้นรันแอปพลิเคชัน สแต็คเชกเมนท์ของโปรแกรมนี้ยังว่างเปล่า เมื่อมีการเรียกใช้ฟังก์ชันโปรแกรมเมอร์หรือคอมไฟเลอร์ จะสร้าง หรือ จogn สแต็คเฟรมลงไปในบริเวณสแต็คเชกเมนท์ ณ ตำแหน่งสแต็คพอยเตอร์ปัจจุบัน (Current Stack Pointer) ทำให้สแต็คพอยเตอร์ย้ายตำแหน่งไปที่ตำแหน่งใหม่ โปรแกรมเมอร์สามารถใช้งานพื้นที่ภายในสแต็คเฟรมนี้ ตามที่กล่าวไปแล้ว เมื่อฟังก์ชันทำงานเสร็จสิ้น โปรแกรมเมอร์จะต้องรีเทิร์นออกจากฟังก์ชัน โดยคืนพื้นที่ (Pop) สแต็คเฟรมออกจากหน่วยความจำ เพื่อให้สแต็คว่างลง

โปรแกรมสามารถเรียกฟังก์ชันภายในฟังก์ชัน (Nested Function) สแต็คเฟรมจะถูกวางเรียงต่อกันไป โดยเฉพาะการเรียกฟังก์ชันแบบเรียกตัวเอง สแต็คเฟรมจะเรียงต่อกันไปเรื่อยๆ จนกว่าจะรีเทิร์นกลับ หากไม่มีการรีเทิร์นกลับ พื้นที่ของสแต็คเชกเมนท์จะโตขึ้นจนถึงขีดจำกัด เรียกว่า **RLIMIT_STACK** หรือจอนสแต็คพอยท์เตอร์ซ้อนทับกับตำแหน่งของชิป และทำให้เกิด Exception เพื่อให้ระบบปฏิบัติการมาบริหารจัดการต่อไป

- **ไฮป (Heap)** คือ เชกเมนท์หนึ่งในหน่วยความจำเสมือน อนุญาตให้โปรแกรมเมอร์จองหน่วยความจำเพิ่มเติม เช่น ลิงค์ลิสต์ (Linked List) และโครงสร้างข้อมูลชนิดต่างๆ โดยใช้คำสั่ง malloc ในภาษา C และคำสั่ง new ในภาษา C++ ไฮปสามารถขยายขนาดเพิ่มเติมได้ จนถึงตำแหน่งบนสุดของสแต็คพอยท์เตอร์ (Stack Pointer) โปรแกรมเมอร์สามารถคืนหน่วยความจำที่จองได้ โดยใช้คำสั่ง free ในภาษา C และคำสั่ง delete ในภาษา C++

รายละเอียดเพิ่มเติมสามารถศึกษาและทดลองได้จากการทดลองที่ 4 ภาคผนวก D และการทดลองอื่นตามลำดับ

3.2.4 ซอฟต์แวร์ประยุกต์อ่าน/เขียนข้อมูลในหน่วยความจำเพื่อให้ซีพียูประมวลผล

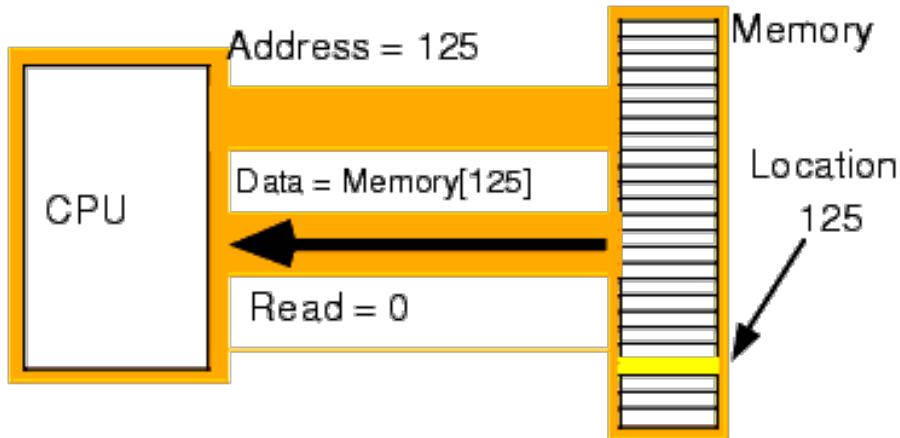


Figure 3.13: ขบวนการอ่านข้อมูลจากหน่วยความจำหลักที่ตำแหน่ง 125 ที่มา: www.phatcode.net

ข้อมูลชนิดค่าคงที่ เช่น ค่า π หรือ ตัวแปรต่างๆ อาศัยพื้นที่ในหน่วยความจำหลักในการเก็บค่าตามชนิดของข้อมูลนั้น ตามตารางที่ 1.1 เมื่อต้องการค่าตัวแปรเหล่านี้ไปประมวลผล ซีพียูต้องอ่านค่า (Load) ไปพักในรีจิสเตอร์ โดยแบ่งเป็น รีจิสเตอร์เลขจำนวนเต็ม และ รีจิสเตอร์เลขทศนิยม รีจิสเตอร์เลขจำนวนเต็ม จะใช้เก็บค่าตัวแปรชนิด char, int, unsigned int, long เป็นต้น ส่วนรีจิสเตอร์เลขทศนิยม จะใช้เก็บค่าตัวแปรชนิด IEEE Single Precision และ Double Precision ทั้งนี้ขึ้นอยู่กับชาร์ดแวร์

เมื่อประมวลผลเสร็จสิ้น ซีพียูต้องนำค่าผลลัพธ์ในรีจิสเตอร์ไปเก็บ (Store) หรือเขียนกลับเข้าสู่หน่วยความจำหลัก ณ ตำแหน่งหรือแอดเดรสของตัวแปรนั้นๆ ตัวอย่างโปรแกรมในตารางที่ 3.3 คำสั่งที่บอกให้ซีพียูอ่านหรือเขียนข้อมูล เรียกว่าคำสั่งประเภท Load/Store ซึ่งจะได้ล่าวในรายละเอียดในบทที่ 4 และการทดลองที่ 6 การพัฒนาโปรแกรมภาษาแอสเซมบลีในภาคผนวก F

ในด้านความปลอดภัย ซอฟต์แวร์หรือแอพพลิเคชันใดๆ อ่านหรือเขียนข้อมูล ได้เฉพาะในพื้นที่ User Space ของตนเองเท่านั้น โดยเฉพาะデータเซ็กเมนท์และสแต็คเซ็กเมนท์ แอพพลิเคชันใดๆ ไม่สามารถเขียนหรือแก้ไขหน่วยความจำในเทิร์กเซ็กเมนท์ของตนเอง และยังไม่สามารถแก้ไขหน่วยความจำสมேือนของแอพพลิเคชันอื่นๆ ได้ โดยโอเอสมีหน้าที่ป้องกันและตรวจจับพฤติกรรมเหล่านี้

3.2.5 การใช้งานอินพุตและเอาท์พุตต่างๆ

ภายใน Kernel ของ Linux ประกอบด้วยโมดูลต่างๆ ที่สำคัญดังนี้ Process Management Subsystem, Memory Management Subsystem และ IO subsystem ดังรูปที่ 3.14

ในหัวข้อนี้จะกล่าวถึง I/O Subsystem เป็นหลัก ซึ่งการใช้งานอินพุต/เอาท์พุต เช่น คีย์บอร์ด เม้าส์ พринเตอร์บางรุ่น จะอาศัยการเชื่อมต่อกับอุปกรณ์เหล่านี้ในลักษณะของ Character และมีซอฟต์แวร์เฉพาะเรียกว่า ดีไวซ์ไดรเวอร์ Device Driver ซึ่งกำหนดรายละเอียดเอาไว้สำหรับผู้ผลิตและอุปกรณ์แต่ละรุ่น การเชื่อมต่อลักษณะนี้ได้แก่ โปรแกรม Terminal

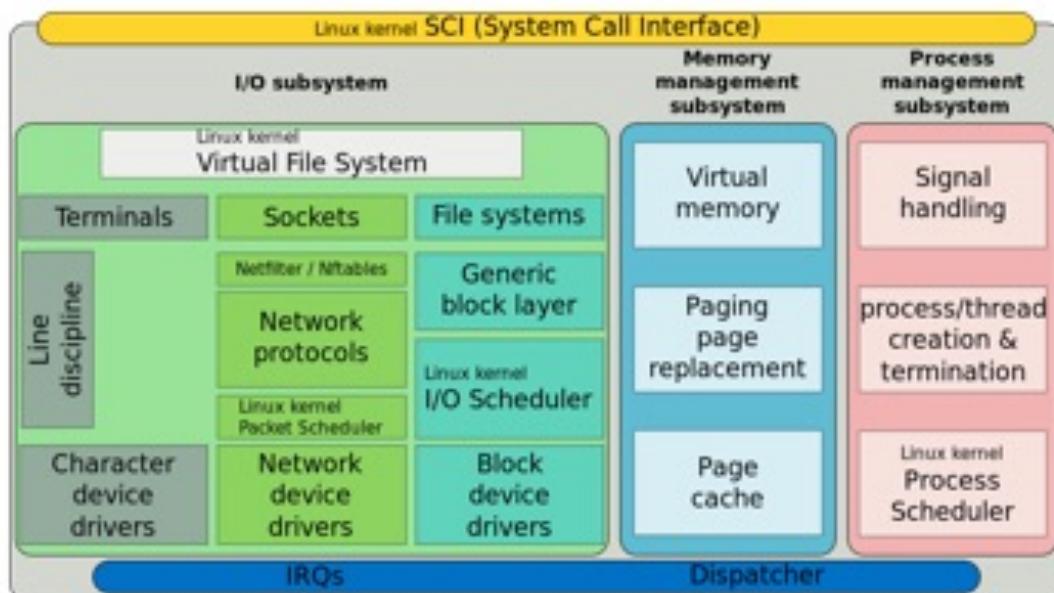


Figure 3.14: โครงสร้างของลีนุกซ์เครื่องเนล โดยแบ่งส่วนที่เป็น I/O, Memory Management และ Process Management [wikipedia](#)

การเข้มต่อกับเครือข่ายอินเทอร์เน็ตผ่านอุปกรณ์สื่อสารชนิดสายและไร้สาย จะอยู่ในรูปของ Network Interface ซึ่งจำเป็นต้องอาศัยโปรแกรมตัวเวิร์ค์ไดเร่อร์เข่นกัน การเข้มต่อลักษณะนี้ มีชื่อเฉพาะเรียกว่า ซ็อกเก็ต (Socket)

การใช้งานอินพุตและเอาท์พุต เช่น อุปกรณ์สำรองข้อมูล เครื่องอ่านซีดี/ดิวีดีรอม (CD/DVD Rom Drive) แฟลชไดร์ (Flash Drive) เป็นต้น จะอาศัยการเข้มต่อกับอุปกรณ์เหล่านี้ในลักษณะของ Block ข้อมูล ขนาดตั้งแต่ 512 ไบท์ขึ้นไป การอ่านหรือเขียนข้อมูลจากไฟล์ จะต้องผ่านกระบวนการนี้เข่นกัน ซึ่ง โวเอสจะเป็นผู้บริหารจัดการ เปิดไฟล์ อ่าน/เขียนข้อมูล และปิดไฟล์ การเข้มต่อลักษณะนี้ มีชื่อเฉพาะเรียกว่า ระบบไฟล์ (File Systems) การใช้งานอุปกรณ์อินพุต/เอาท์พุต ทั้งสามรูปแบบ เพื่อเข้มต่อ อุปกรณ์ต่างๆ กับระบบปฏิบัติการ ในรูปแบบของ Virtual File System ซึ่งจะมีรายละเอียดเพิ่มเติมในบทที่ 7 และการทดลองที่ 12 ภาคผนวก [L](#)

3.2.6 การอ่าน/เขียนข้อมูลระหว่างหน่วยความจำหลักและอุปกรณ์เก็บรักษาข้อมูล

ไฟล์ คือ ข้อมูลขนาดใหญ่ที่แต่ละไบท์เรียงตัวต่อกันตั้งแต่ต้นไฟล์ไปจนสิ้นสุดไฟล์ เมื่อเจอสัญลักษณ์ EOF (End of File) ซึ่งเป็นตัวอักษรขนาด 1 ไบท์ตามรหัสแอสกี ไฟล์มีหลายชนิด เช่น ไฟล์ตัวอักษรเรียกว่า Text File ไฟล์ข้อมูล ทั้งนี้ รายละเอียดโครงสร้างไฟล์ข้อมูลแต่ละชนิดจะแตกต่างกันไป เช่น ไฟล์รูปภาพ JPEG (.jpg) ไฟล์รูปภาพ PNG (.png) ไฟล์เสียง WAV (.wav) ไฟล์ภาพเคลื่อนไหว MPEG4 (.mp4) เป็นต้น นอกจากไฟล์ข้อมูลชนิดต่างๆ ที่กล่าวมาแล้ว คอมพิวเตอร์จำเป็นต้องมีไฟล์ที่บรรจุโปรแกรมคำสั่ง (Executable File) ตามรูปแบบ ELF ที่กล่าวในหัวข้อที่ [3.2.2](#)

การใช้งานไฟล์จะต้องเริ่มด้วยการเปิดไฟล์เสมอ การเปิดไฟล์ (Open File) คือ การจองหน่วยความจำให้กับไฟล์ที่ต้องการเพื่ออ่าน หรือ เขียน การเปิดไฟล์มีหลายทางเลือก ได้แก่ เปิดไฟล์เพื่ออ่าน (Read

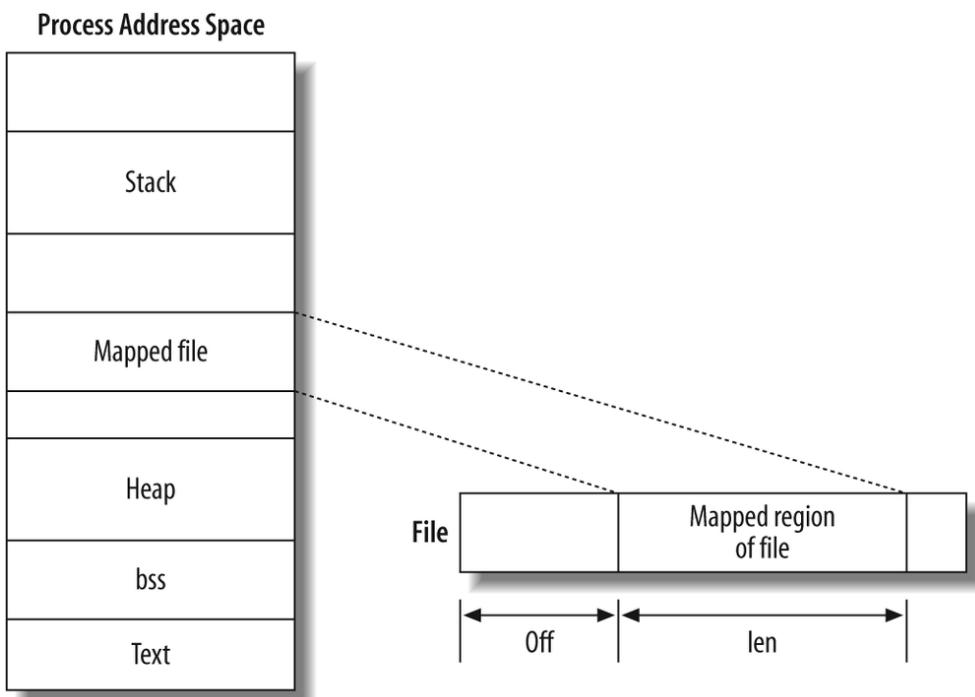


Figure 3.15: หลักการของ Memory Map File คือการแบ่งหน่วยความจำสำหรับอ่านหรือเขียนไฟล์ ด้านขวาคือ โครงสร้างของไฟล์ในเชิงตรรกะ ด้านซ้ายคือ หน่วยความจำที่ถูกจงเพื่อใช้พักข้อมูลในไฟล์ ที่มา: safaribooksonline.com

Only) เพื่อเขียน (Write Only) เพื่ออ่านและเขียน (Read/Write) เป็นต้น เมื่อเริ่มต้นเปิดไฟล์เพื่ออ่านข้อมูลในไฟล์จะถูกอ่านเข้ามาพักเก็บในหน่วยความจำที่ถูกจง ตามหลักการที่เรียกว่า Memory Map File คือการแบ่งหน่วยความจำสำหรับอ่านหรือเขียนไฟล์ รูปที่ 3.15 การอ่านไฟล์ คือ การอ่านข้อมูลจากหน่วยความจำบริเวณที่ตรงกับไฟล์ โดยข้อมูลจากไฟล์ในอุปกรณ์เก็บรักษาข้อมูลจะถูกอ่านขึ้นมาบรรจุในหน่วยความจำตำแหน่งนั้นๆ

เมื่อโปรแกรมประมวลผลข้อมูลตามที่ได้รับมอบหมายเรียบร้อย โปรแกรมหรือแอพพลิเคชันควรจะบันทึกข้อมูลหรือสารสนเทศเก็บลงในไฟล์ เพื่อป้องกันการสูญหาย หรือเพื่อนำไปใช้ต่อ ไฟล์จะเก็บอยู่ในอุปกรณ์เก็บรักษาข้อมูลตามโครงสร้างของระบบปฏิบัติการนั้นๆ สำหรับระบบลีนุกซ์จะมีโครงสร้างตามรูปที่ 3.14

การเขียนไฟล์ คือ การเขียนข้อมูลไปยังหน่วยความจำบริเวณที่ตรงกับไฟล์ โดยระบบปฏิบัติการจะย้ายข้อมูลจากหน่วยความจำตำแหน่งนั้นๆ ไปเขียนลง ในอุปกรณ์เก็บรักษาข้อมูลจริง ตามขนาดของบัฟเฟอร์ ดังนั้น เมื่อใช้งานไฟล์เสร็จสิ้นโปรแกรมเมอร์จะต้องปิดไฟล์เสมอ การปิดไฟล์ (Close File) คือ การนำข้อมูลที่ยังค้างในหน่วยความจำเพื่อย้ายข้อมูลเหล่านั้นไปเขียนต่อในอุปกรณ์เก็บรักษาข้อมูลจริง ด้วยขบวนการ DMA (Direct Memory Access) และคืนหน่วยความจำที่จองไว้ให้ระบบปฏิบัติการ แม้ว่าจะเพียงแค่เปิดไฟล์เพื่ออ่านเท่านั้น

3.2.7 การซัฟดาวน์ (Shut Down) ระบบปฏิบัติการก่อนหยุดจ่ายไฟ

การซัฟดาวน์เป็นการสั่งให้ระบบปฏิบัติการอัพเดทข้อมูลต่างๆ ของเครื่องในรูปของไฟล์ต่างๆ กลับลงในอุปกรณ์เก็บรักษาข้อมูล แล้วปิดไฟล์ที่เปิดค้างไว้ การปิดเครื่องโดยไม่สั่งซัฟดาวน์จะทำให้การบูทเครื่องครั้งต่อไปมีปัญหา และจะทำให้ระบบปฏิบัติการทำงานได้ไม่สมบูรณ์

ผู้ใช้งานระบบลินุกซ์สามารถซัฟดาวน์ได้หลายวิธี เช่น การใช้มาสเตอร์กดปุ่มซัฟดาวน์ด้านซ้ายบนสุด หรือพิมพ์คำสั่ง shutdown -h บนโปรแกรม Terminal การซัฟดาวน์ทั้งสองวิธี คือ การส่งสัญญาณ (Signalling) ไปยังโมดูล init ให้เปลี่ยนระดับการรัน (Run Level) ให้เป็นระดับ 0 (Halt) เพื่อปิดระบบ ซึ่งปกติจะอยู่ระดับ 3 หรือระดับ 5 เป็นค่าเดフォลต์ (Default)

การประยุกต์ใช้ซัฟดาวน์ คือ คำสั่ง shutdown -r เพื่อรีสตาร์ทเครื่อง เป็นการส่งสัญญาณไปยังโมดูล init ให้เปลี่ยนระดับการรันเป็นระดับ 6 การรีสตาร์ทระบบนิยมใช้กับคอมพิวเตอร์ระบบฝังตัว เพื่อเริ่มต้นการทำงานใหม่ โปรแกรมเมอร์สามารถเขียนสคริปต์ (Script) ด้วย Permission ของซูเปอร์ยูสเซอร์ (Super User) ระบบ Microsoft Windows ทำได้เช่นกัน แต่เรียกว่า การเขียนคำสั่งเบท์ (Batch Command)

3.3 การพัฒนาซอฟต์แวร์หรือโปรแกรมคอมพิวเตอร์

การพัฒนาซอฟต์แวร์ทำได้หลายระดับ ขึ้นอยู่กับสิ่งแวดล้อมและความต้องการของซอฟต์แวร์ การเขียนโปรแกรมด้วยภาษาสคริปต์ (Script Language)

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

การพัฒนาซอฟต์แวร์ด้วยภาษาไพธอน (Python) บนบอร์ด Pi3 ได้รับความนิยม เนื่องจากตัวภาษามีความซับซ้อนต่ำ เรียนรู้ง่ายได้ด้วย มีตัวอย่างโปรแกรมที่นักพัฒนาทั่วโลกเปิดเผยแพร่สโคเด็คผ่านทางเครือข่ายอินเทอร์เน็ต ผู้อ่านสามารถค้นคว้าเพิ่มเติมได้ที่ลิงค์ต่อไปนี้ [wikipedia](#) เป็นต้น

ตำราเล่มนี้จะเน้นการพัฒนาซอฟต์แวร์โดยใช้การประมวลและภาษา C/C++ เป็นกรณีศึกษา เนื่องจากผลลัพธ์ที่ได้จากการพัฒนาซอฟต์แวร์ คือ คำสั่งภาษาแอสเซมบลี และคำสั่งภาษาเครื่อง (Native Machine Level Code) ตามรูปแบบของไฟล์ ELF

3.3.1 โครงสร้างของชอร์สโคเด็คโปรแกรมภาษา C/C++

ภาษาระดับสูง (High-level language) การเขียนโปรแกรมด้วยภาษาระดับสูงเพื่อแก้โจทย์หรือปัญหา โปรแกรมเมอร์สามารถเขียนโปรแกรมได้ง่ายและสร้างสรรค์ผลงานได้ง่ายกว่าภาษาระดับล่าง เช่น ภาษาแอสเซมบลี และภาษาเครื่อง เนื่องจากภาษาระดับสูงมีลักษณะใกล้เคียงกับประโยคภาษาอังกฤษ ภาษาระดับสูงจะไม่ยึดติดกับเครื่องหรืออาร์ดแวร์ (Machine Independent) ในทางตรงกันข้าม ภาษาระดับล่าง เช่นภาษาแอสเซมบลีและภาษาเครื่องจะเข้มข้นโดยตรงกับอาร์ดแวร์หรือชิปปุญญาในเครื่องคอมพิวเตอร์นั้นๆ (Machine Specific) แต่โปรแกรมเมอร์สามารถใช้คำสั่งภาษาแอสเซมบลีในการสั่งงานอาร์ดแวร์ของเครื่องคอมพิวเตอร์ จึงเหมาะสมกับงานที่ต้องการความรวดเร็ว ตำนานี้จะใช้ภาษา C และภาษาแอสเซมบลี เป็นหลัก เพื่อเข้าถึงภาษาเครื่อง ระบบปฏิบัติการและอาร์ดแวร์ได้ลึกซึ้งมากกว่า

ตัวอย่างชอร์สโคเด็คภาษา C ฟังก์ชันหลัก ในรูปที่ 3.16 ชื่อไฟล์ main.c ฟังก์ชันหลักชื่อ main() เป็นฟังก์ชันที่โปรแกรมเริ่มต้นทำงาน เปรียบได้กับ การทำงานโดยไม่แสดงรายละเอียด เมื่อทำงานเสร็จสิ้นจะรีเทิร์นค่า 0 ซึ่งเป็นเลขจำนวนเต็มค่าหนึ่ง ซึ่งมักใช้ตรวจสอบว่าเป็นการทำงานเสร็จสิ้นโดยไม่มีปัญหาหรือข้อผิดพลาด และรีเทิร์นค่าอื่นๆ ที่ไม่ใช่ 0 เพื่อบ่งบอกหัสความผิดพลาดได้

3.3.2 การคอมไพล์ชอร์สโคเด็คให้กลายเป็นโปรแกรมคอมพิวเตอร์

ภาษาระดับสูง (High-level language) มีศักยภาพสูงที่จะพัฒนาเป็นโปรแกรมหรือแอพพลิเคชันได้เร็ว ซึ่งโปรแกรมเมอร์สามารถเขียนโปรแกรมในรูปแบบไฟล์ภาษาต่างๆ เรียกโดยรวมว่า Source code ไฟล์การพัฒนาซอฟต์แวร์จากชอร์สโคเด็คให้เป็นไฟล์ Executable หรือโปรแกรม บนระบบปฏิบัติการยูนิกซ์ในรูปที่ 3.17 ประกอบด้วยโปรแกรมคอมไพล์อร์ แอสเซมเบลอร์ และลิงค์เกอร์ เป็นหลัก

คอมไพล์อร์เป็นโปรแกรมคอมพิวเตอร์ชนิดหนึ่งที่หน้าที่แปลชอร์สโคเด็คภาษาสูงให้กลายเป็น คำสั่งภาษาแอสเซมบลีและคำสั่งภาษาเครื่องในที่สุด แล้วจึงจัดเรียงคำสั่งภาษาเครื่องให้มีโครงสร้างตามไฟล์

```

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

```

Figure 3.16: ตัวอย่างซอฟต์แวร์โค้ดภาษา C ฟังก์ชันหลักชื่อ main() เมื่อทำงานเสร็จสิ้นจะรีเทิร์นค่า 0

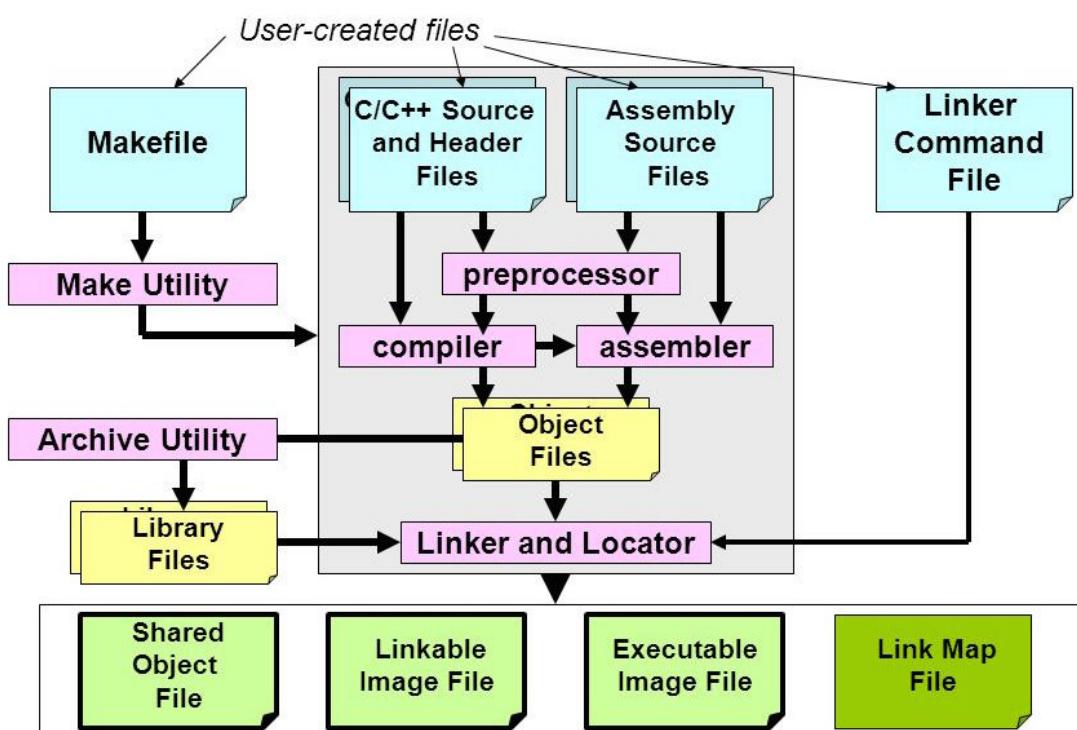


Figure 3.17: ไฟล์การพัฒนาซอฟต์แวร์จากซอฟต์แวร์ให้เป็นไฟล์ Executable (ELF) หรือโปรแกรมประยุกต์ (Application Program) หรือย่อๆว่า แอพ (ที่มา: slideplayer.com)

ELF ซึ่งอธิบายแล้วในหัวข้อที่ 3.2.2 คอมไพล์เตอร์ต้องทำความเข้าใจตัวซอฟต์แวร์โค้ดต่างๆ โดยอ่านไฟล์ซอฟต์แวร์โค้ดเพื่อสแกนตัวอักษรที่เป็นเนื้อโปรแกรม จัดกลุ่มคำ (Token) ตรวจคำสังกัด แปลความหมายของคำ สร้างแผนภูมิต้นไม้ (Abstract Syntax Tree) เพื่อตรวจสอบไวยากรณ์และ ความหมาย (Semantic Analysis) ของประโยคต่างๆ เมื่อปราศจากข้อผิดพลาด คอมไпал์เตอร์จะดำเนินการต่อ เพื่อปรับปรุงการทำงานของโปรแกรมที่จะคอมไพล์ให้ดีขึ้น เรียกว่า การอptyimize (Optimize) โดยการสร้าง Data-Flow graph

เพื่อประกอบการวิเคราะห์การไฟล์ของข้อมูล เพื่อประกอบการทำอptyไมเซชัน (Optimization) เพื่อให้โปรแกรมทำงานได้ดียิ่งขึ้น หลังจากนั้น คอมไพล์จะสร้างแมชีนโค้ด (Machine Code) ผู้อ่านสามารถศึกษารายละเอียดเพิ่มเติมได้ที่ [wikipedia](#)

โปรแกรมเมอร์จะกำหนดความสัมพันธ์หรือความเชื่อมโยงระหว่างชอร์สโค้ดเหล่านั้นใน Makefile เพื่อให้คอมไпал์เตอร์สามารถสร้าไฟล์ผลลัพธ์ตามที่โปรแกรมเมอร์ต้องการ คือ

- Shared Object File (*.so) คือ ไฟล์อ็อบเจ็คท์ที่ได้จากการคอมไпал์ซอร์สโค้ดของโปรแกรมเมอร์เอง เพื่อนำไปใช้ร่วมกับไฟล์อื่นๆ ต่อไป
- Linkable Image File (*.lib) คือ ไฟล์ไลบรารีที่ได้จากการคอมไпал์โดยนักพัฒนาอื่น เพื่อนำมาลิงค์กับไฟล์หลักให้สมบูรณ์
- Executable Image File (ชื่อไฟล์ a.out) คือ ไฟล์โปรแกรมที่ได้จากการลิงค์หรือเชื่อมกับไฟล์อ็อบเจ็คท์และไฟล์ไลบรารีที่กล่าวมาข้างต้น ซึ่งจัดเรียงตามรูปแบบไฟล์ ELF ในรูปที่ 3.11 ขบวนการลิงค์จะกล่าวโดยละเอียดในหัวข้อที่ 3.4
- Link Map File (*.map) เพื่ออธิบายการเชื่อมโยงกันของตัวแปรต่างๆ ในชอร์สโค้ด

ไฟล์ทั้งหมดนี้จะบรรจุคำสั่งภาษาเครื่องในรูปของเลขฐานสอง และข้อมูลในรูปของเลขฐานสอง เช่น กันตั้งนั้น เชิญเมนท์ต่างๆ ของไฟล์ ELF จะเป็นสิ่งกำกับว่า เลขฐานสองแต่ละค่า คือ คำสั่ง หรือ ข้อมูล ยกตัวอย่างเช่น

- เลขฐานสองที่จัดเรียงอยู่ใน Text เชิญเมนท์ คือ คำสั่งภาษาเครื่อง เท่านั้น
- เลขฐานสองที่จัดเรียงอยู่ใน Data เชิญเมนท์ คือ ข้อมูล เท่านั้น ซึ่งอาจจะมีค่าเป็นเลขจำนวนเต็ม ถ้าตำแหน่งตรงกับตัวแปรชนิดจำนวนเต็ม หรือมีค่าเป็นเลขทศนิยมชนิดจุดลอยตัว ตามมาตรฐาน IEEE 754 Single Precision จะตรงกับตัวแปรชนิด float เป็นต้น

ก่อนที่ผู้อ่านจะได้รู้จักกับคำสั่งภาษาเครื่องในรูปของเลขฐานสอง หัวข้อต่อไปนี้จะอธิบาย ภาษาแอกซ์เพมบลีซึ่งมีลักษณะใกล้เคียงกับภาษาเครื่องแต่โปรแกรมเมอร์ สามารถทำความเข้าใจได้

3.3.3 โครงสร้างของชอร์สโค้ดโปรแกรมภาษาแอกซ์เพมบลีของ ARM

ภาษาแอกซ์เพมบลี (Assembly language) เป็นคำสั่งภาษาคอมพิวเตอร์ที่อยู่ในรูปของคำย่อ (Mnemonic) จากภาษาอังกฤษ ทำให้โปรแกรมเมอร์เข้าใจง่ายกว่าตัวเลขฐานสองที่เป็นคำสั่งภาษาเครื่อง (Machine Instruction) ในตำนานี้จะใช้ภาษาแอกซ์เพมบลีของ ARM Cortex A เวอร์ชัน 32 บิตเป็นหลัก รายละเอียดในบทที่ 4

โดยภาพรวมของตารางที่ 3.3 แสดงให้เห็นถึง โครงสร้างหลักของชอร์สโค้ดภาษาแอกซ์เพมบลีของ ARM แบ่งเป็น .text หมายถึง Code เชิญเมนท์ และ .data คือ ดาต้าเชิญเมนท์ ตามลำดับ ส่วนคำสั่งภาษาแอกซ์เพมบลี (Assembly language) ในบรรทัดอื่นๆ อยู่ในรูปของคำภาษาอังกฤษที่โปรแกรมเมอร์สามารถทำความเข้าใจได้ และมีความใกล้เคียงกับคำสั่งภาษาเครื่อง ประกอบด้วย ฟังค์ชัน main ประกอบด้วย 3 คอลัมน์ ดังนี้

- คอลัมน์ซ้าย ใช้กำหนดค่าเลเบล (Label) ต่างๆ
- คอลัมน์กลาง ใช้กำหนดคำสั่ง (Operation) ว่าต้องการจะให้ชีพิญทำอะไร
- คอลัมน์ขวา ใช้ระบุค่า รีจิสเตรอร์ หรือ แอดเดรส หรือ เลเบล หรือ ค่าคงที่สำหรับคำสั่งในคอลัมน์ ตรงกันๆ

Table 3.3: ตัวอย่างโปรแกรมภาษาแอสเซมบลีเพื่อประกาศและตั้งค่าเริ่มให้ตัวแปรขนาด 32 บิต จำนวน 4 ตัวแปร

บรรทัดที่	เลเบล	คำสั่ง	รีจิสเตรอร์ หรือ แอดเดรส หรือ 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 3.3 สามารถอธิบายตามหมายเหตุบรรทัดได้ดังนี้

1. .text หมายถึง ตำแหน่งเริ่มต้นของเทกซ์เช็กเมนท์ ตามที่กล่าวในหัวข้อ 3.2.2
2. .global main หมายถึง พังค์ชันชื่อ main ซึ่งจะตรงกับพังค์ชัน main() ในภาษา C/C++
3. main: หมายถึง ตำแหน่งเริ่มต้นของพังค์ชัน main
4. คำสั่ง LDR ย่อมาจาก **Load Data Register** ดังนั้น LDR R1, =M คือ การโหลดค่าแอดเดรสของตัวแปร M มาเก็บในรีจิสเตรอร์ R1
5. คำสั่ง LDR R1, [R1] นำเอาหมายเลขแอดเดรสของตัวแปร M ที่อยู่ในรีจิสเตรอร์ R1 ไปอ่านค่าของ M มาบรรจุในรีจิสเตรอร์ R1

6. คำสั่ง LDR R2, POINTR คือ การโหลดค่าแอ็ดเดรสของตัวแปร POINTR ซึ่งเก็บแอ็ดเดรสของตัวแปร NUM มาเก็บในรีจิสเตอร์ R2
7. คำสั่ง MOV ย่อมาจาก MOVE ดังนี้ 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 โดยคำสั่งนี้จะกำหนดให้มี Label ชื่อ LOOP ซึ่งต้องการจะให้เกิดการทำซ้ำ
9. คำสั่ง ADD หมายถึง การบวกค่าภายในรีจิสเตอร์ โดยบวกค่า R0+R3 แล้วเก็บผลลัพธ์ในรีจิสเตอร์ R0
10. คำสั่ง SUBS ย่อมาจาก Subtract และอัพเดทผลลัพธ์ในรีจิสเตอร์สถานะ (CPSR: Current Program Status Register) ซึ่งรายละเอียดจะกล่าวในบทที่ 4 ดังนี้ 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 การทำงานจะกลับไปเริ่มที่คำสั่งที่ตรงกับเลเบล LOOP ดังนี้ โปรแกรมนี้จะวนรอบ 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 การโหลดค่าแอ็ดเดรสของตัวแปร SUM มาเก็บในรีจิสเตอร์ R4 โดยที่ สัญลักษณ์ =SUM หมายถึง แอ็ดเดรสของตัวแปร SUM
14. คำสั่ง STR ย่อมาจาก Store Register คือ การนำผลบวกในรีจิสเตอร์ R0 ไปเก็บไว้ที่รีจิสเตอร์ R4 ซึ่งตรงกับแอ็ดเดรสของตัวแปร SUM ซึ่งอยู่ในหน่วยความจำ
15. BX LR บอกรุ่นสินสุดของชอร์สโค้ด เมื่อโปรแกรมที่เขียนนี้เสร็จสิ้นการทำงานจะรีเทิร์นกลับไปหา โอเอส
16. .data หมายถึง ตำแหน่งเริ่มต้นของเซกเมนต์ data
17. SUM ซึ่งของตัวแปรชนิดจำนวนเต็ม (Integer) ขนาด 32 บิตชนิด 2-Complement ถูกกำหนดค่าเริ่มต้น ให้มีค่าเป็น 0
18. M คือ ชื่อของตัวแปรชนิดจำนวนเต็ม (Integer) ขนาด 32 บิตชนิด 2-Complement ถูกกำหนดค่าเริ่มต้น ให้มีค่าเป็น 4 หมายถึง จำนวนสมาชิกของตัวแปรอะเรย์ NUM
19. NUM คือ ชื่อของตัวแปรชนิดอะเรย์ของเลขจำนวนเต็มที่จะถูกกำหนดค่าเริ่มต้น ให้มีค่าเป็น 3, 5, 7, 9 ตามลำดับ โดย NUM[0] = 3 และ NUM[3] = 9

20. POINTR ชื่อของตัวแปรที่ถูกกำหนดค่าเริ่มต้นเป็นแอดเดรสของ NUM ในการพัฒนาโปรแกรมภาษา C/C++ เรียกตัวแปรนี้ว่า พอยท์เตอร์ (Pointer) โดยตัวแปรชนิดพอยท์เตอร์จะเก็บแอดเดรส ด้วยพื้นที่ขนาด 1 Word หรือเท่ากับ 4 ไบท์ ตามชนิดของระบบปฏิบัติการ ซึ่งในทำราเล่นนี้เป็นชนิด 32 บิต หรือ 4 ไบท์นั่นเอง

3.3.4 การเข้ามไฟล์อืบเจ็คท์ด้วยลิงค์เกอร์ (Linker)

ลิงค์เกอร์ (Linker) เป็นหนึ่งในซอฟต์แวร์ระบบที่มีความสำคัญต่อขบวนการพัฒนาโปรแกรม ในรูปที่ 3.17 แสดงการทำงานของลิงค์เกอร์เพื่อเข้ามไฟล์อืบเจ็คท์ หลัก ไฟล์อืบเจ็คท์เสริม และไฟล์ไลบรารีของภาษา C เข้าด้วยกัน ไฟล์ทั้งหมดนี้อยู่ในรูปแบบ ELF ในหัวข้อที่ 3.2.2 ไฟล์ไลบรารีมาตรฐานของแต่ละภาษา คือ ไฟล์อืบเจ็คท์ที่รวมฟังก์ชันสำหรับที่ถูกคอมไพล์แล้วพัฒนาอย่างต่อเนื่องจนน่าเชื่อถือ ไฟล์อืบเจ็คท์ คือ ไฟล์ที่รวมฟังก์ชันที่นักพัฒนาเก็บไว้ส่วนตัว ซึ่งตอนลงเมืองมีชอร์สโค้ดแล้วแต่ไม่จำเป็นต้องคอมไพล์ใหม่อีกรอบ ทำให้ประหยัดเวลาสำหรับการคอมไพล์

รูปที่ 3.18 แสดงตัวอย่างการลิงค์ (Link) หรือรวมไฟล์อืบเจ็คท์หลัก ไฟล์อืบเจ็คท์เสริม และไฟล์ไลบรารีเข้าด้วยกันให้เป็นไฟล์โปรแกรมหรือแอพพลิเคชันที่สมบูรณ์ กล่องสี่เหลี่ยมในรูป คือ ไฟล์ที่บรรจุคำสั่งและข้อมูลเป็นเลขฐานสอง แต่เพื่อให้ผู้อ่านเข้าใจได้ง่าย คำสั่งจึงแสดงเป็นภาษาแอสเซมบลีแทน คำอธิบายเพิ่มเติมในหัวข้อที่ 4.8 เรื่องการเรียกใช้ฟังก์ชัน

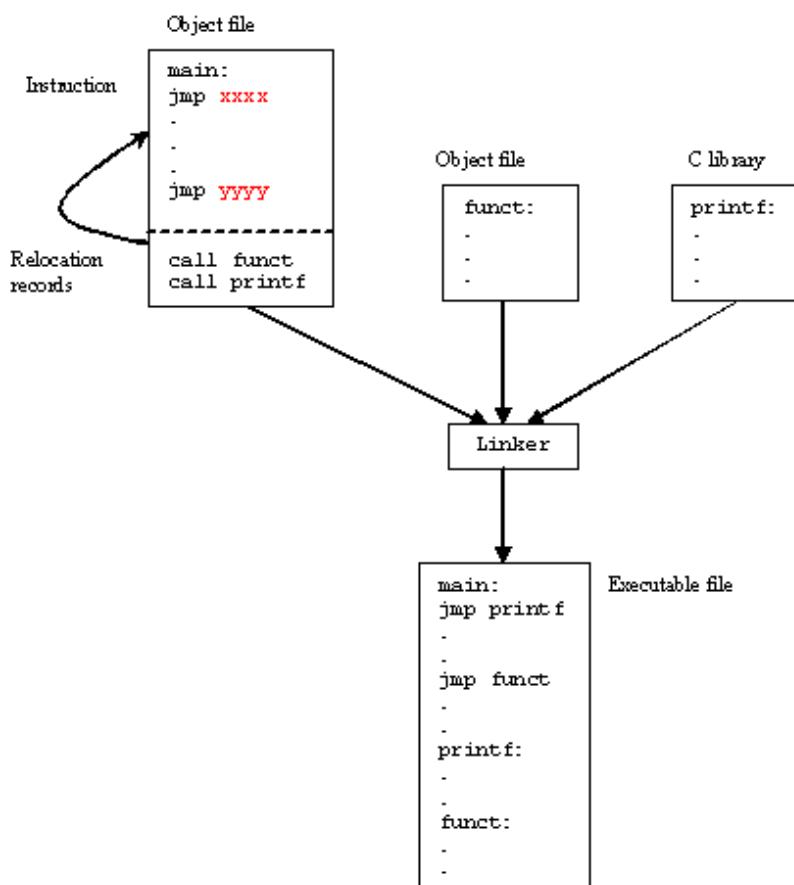


Figure 3.18: ตัวอย่างการลิงค์ (Link) หรือรวมไฟล์อืบเจ็คท์หลัก ไฟล์อืบเจ็คท์เสริม และไฟล์ไลบรารีเข้าด้วยกันเป็นไฟล์โปรแกรมหรือแอพพลิเคชัน ที่มา: google.com

การลิงค์เป็นการขยายขีดความสามารถของโปรแกรมที่พัฒนาใหม่จากไฟล์อืบเจ็คท์และไฟล์ไลบรารีที่มีอยู่เดิม โปรแกรมเมอร์สามารถเรียกโปรแกรมที่ต้องการและเรียกใช้งานฟังก์ชัน (Function Call) ที่มีผู้พัฒนาไว้แล้ว ตามหลักการที่สำคัญพัฒนาซอฟต์แวร์ที่ดี เรียกว่า **โมดูลาริตี้** (Modularity) รายละเอียดสามารถศึกษาเพิ่มเติมด้วยภาษา C จากการทดลองที่ 5 ภาคผนวก E ถึงการทดลองที่ 8 ภาคผนวก H.3

3.3.5 ตัวอย่างของคำสั่งภาษาเครื่อง (Machine Code)

ผู้อ่านได้ทำความเข้าใจกระบวนการพัฒนาซอฟต์แวร์และอาจได้ทำการทดลองในภาคผนวกแล้ว หัวข้อนี้จะเสริมความเข้าใจเรื่องของคำสั่งภาษาแอสเซมบลีและภาษาเครื่องเข้าด้วยกัน โดยจะใช้ภาษาเครื่องของ ARM เป็นกรณีศึกษา เพื่อให้สอดคล้องกับบอร์ด Pi3

รูปที่ 3.19 แสดง ตัวอย่างการแปลงจากคำสั่งแอสเซมบลี (ตัวอักษร) ด้านขวา เป็นคำสั่งภาษาเครื่อง (ตัวเลขฐานสอง) ด้านซ้าย ซึ่งมีรูปแบบที่ชัดเจน โดย ARM เป็นผู้ออกแบบและกำหนดรายละเอียดเพื่อให้ผู้พัฒนาทำตาม

คำสั่ง SUB R5, R7, R1 จะถูกแปลงตามบิทต่างๆ ดังนี้

- บิทที่ 28-31 ความยาว 4 บิท คือตำแหน่งของ cond (Condition) คือเงื่อนไขของการทำงานคำสั่ง ปัจจุบันนี้ ซึ่ง 1110_2 หมายถึง ไม่มีเงื่อนไข รายละเอียดเพิ่มเติมในหัวข้อที่ 4.7
- บิทที่ 26-27 ความยาว 2 บิท คือตำแหน่งของ OpCode (Operation Code) ของคำสั่ง SUB มีค่าเท่ากับ 2_{10}
- บิทที่ 25 ความยาว 1 บิท คือตำแหน่งของ Imm (Immediate) ของคำสั่ง SUB มีค่าเท่ากับ 2_{10}
- บิทที่ 21-24 ความยาว 4 บิท คือตำแหน่งของ cmd (Command) ของคำสั่ง SUB มีค่าเท่ากับ 2_{10}
- บิทที่ 15-19 ความยาว 4 บิท คือตำแหน่งของ Rn มีค่าเท่ากับ 7_{10} หรือ 0111_2 หมายถึง R7
- บิทที่ 12-15 ความยาว 4 บิท คือตำแหน่งของ Rd มีค่าเท่ากับ 5_{10} หรือ 0101_2 หมายถึง R5
- บิทที่ 0-3 ความยาว 4 บิท คือตำแหน่งของ Rm มีค่าเท่ากับ 1_{10} หรือ 0001_2 หมายถึง R1

Field Values										Assembly Code	
31:28 27:26 25 24:21 20 19:16 15:12 11:7 6:5 4 3:0										SUB R5, R7, R1	
1110₂ 00₂ 0 2 0 7 5 0 0 0 1											
cond op I cmd S Rn Rd shamt5 sh Rm											
31:28 27:26 25:20 19:16 15:12 11:0										LDR R9, [R4, #16]	
1110₂ 01₂ 25 4 9 16											
cond op IPUBWL Rn Rd imm12											

Figure 3.19: ตัวอย่างการแปลงจากคำสั่งแอสเซมบลีทางด้านขวา เป็นคำสั่งภาษาเครื่องทางด้านซ้าย (ที่มา: Harris and Harris (2013))

คำสั่ง LDR R9, [R4, #16] จะถูกแปลงตามบิทต่างๆ ดังนี้

- บิทที่ **28-31** ความยาว 4 บิต คือตำแหน่งของ cond (Condition) คือเงื่อนไขของการทำงานคำสั่ง ปัจจุบันนี้ ซึ่ง 1110_2 หมายถึง ไม่มีเงื่อนไข รายละเอียดเพิ่มเติมในหัวข้อที่ [4.7](#)
- บิทที่ **26-27** ความยาว 2 บิต คือตำแหน่งของ OpCode (Operation Code) ของคำสั่ง LDR มีค่าเท่ากับ 01₂
- บิทที่ **20-25** ความยาว 6 บิต คือตำแหน่งของ IPUWL ของคำสั่ง LDR มีค่าเท่ากับ 25_{10}
- บิทที่ **15-19** ความยาว 4 บิต คือตำแหน่งของ Rn มีค่าเท่ากับ 4_{10} หรือ 0100_2 หมายถึง R4
- บิทที่ **12-15** ความยาว 4 บิต คือตำแหน่งของ Rd มีค่าเท่ากับ 9_{10} หรือ 1001_2 หมายถึง R9
- บิทที่ **0-11** ความยาว 12 บิต คือตำแหน่งของ imm12 มีค่าเท่ากับ 16_{10} หรือ 0000 0001 0000₂ หมายถึง #16

คำสั่งที่อ่าน (Fetch) เข้ามาเป็นตัวเลขฐานสองในตำแหน่งบิทต่างๆ เหล่านี้ ชาร์ดแวร์จะตีความหรือถอดรหัส (Decode) ได้ว่า คำสั่งที่อ่านเข้ามาเป็นคำสั่งอะไร ต้องการใช้รีจิสเตอร์ตัวไหน กำหนดค่าคงที่ (Immediate) เป็นเลขฐานสิบอะไรม เป็นบวกหรือลบ และจึงสั่งงานหรือชาร์ดแวร์ที่มีอยู่ให้ปฏิบัติ (Execute) ตาม รายละเอียดในการออกแบบจรหรือชาร์ดแวร์ได้ในวิชาสถาปัตยกรรมคอมพิวเตอร์ ([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](#)) เพื่อให้โปรแกรมลักษณะคล้ายคอมพิวเตอร์แต่เรียกว่า Synthesis Tool แปลเป็นวงจรบนอุปกรณ์
- FPGA ([Field Programmable Gate Array](#)) โดยเป็นวงจรชนิดหนึ่งซึ่งผู้ใช้สามารถเปลี่ยนหรือโปรแกรม (Program) ให้ทำงานเป็นวงจรได้หลายๆ รอบ เพื่อทดสอบการทำงานจนถูกต้อง

3.4 สรุปท้ายบท

เนื้อหาในบทนี้ได้นำเสนอพื้นฐานของเครื่องคอมพิวเตอร์ซึ่งมีองค์ประกอบหลักคือเป็นชาร์ดแวร์และซอฟต์แวร์ ชาร์ดแวร์คืออุปกรณ์อิเลคทรอนิกส์ต่างๆ ซึ่งมีซีพียูเป็นจุดศูนย์กลาง เชื่อมหน่วยความจำหลัก และอินเทล เอาท์พุตต่างๆ ในขณะที่ซอฟต์แวร์แบ่งเป็นซอฟต์แวร์ระบบปฏิบัติการ และซอฟต์แวร์แอพพลิเคชันต่างๆ ซอฟต์แวร์เหล่านี้สามารถพัฒนาด้วยภาษาต่างๆ เช่น C/C++ Java และภาษาต่างๆ เช่น แอสเซมบลี เพื่อคอมไพล์หรือแปลงและลิงค์หรือรวม ให้กลายเป็นไฟล์รูปแบบ ELF บนระบบปฏิบัติการยูนิกซ์ ไฟล์ ELF มีโครงสร้างที่เป็นมาตรฐาน ประกอบด้วยแท็กซีกเมนท์ ซึ่งมีคำสั่งภาษาเครื่องในรูปของเลขฐานสอง ตามมาเซกเมนท์ซึ่งรวมข้อมูลจากตัวแปรที่ได้ประกาศไว้

3.5 คำถ้ามหัยบท

1. เหตุใดเครื่องคอมพิวเตอร์ทั่วไปควรมีระบบปฏิบัติการ เช่น Microsoft Windows, Apple MacOS, Linux เวอร์ชันต่างๆ
2. BCM2837 และ AllWinner A64 มีความแตกต่างกันอย่างไร จากข้อมูลในรูปที่ 3.3 ในแต่ละมุมต่อไปนี้
 - การรองรับหน่วยความจำหลักภายนอกชิป
 - จีพียู
 - Video Engine ซึ่งประกอบด้วยวงจรเข้ารหัส (Encoder) และวงจรถอดรหัส (Decoder)
3. หน่วยความจำหลักที่วางขายมีลักษณะเป็นแพลงวะจะมีชิปเรียงกันหลายตัว เหตุใดบอร์ด Pi3 จึงมีชิปเพียงตัวเดียว
4. คอมพิวเตอร์ทั่วไป อาจมีการดจօเสริมการทำงานหรือเล่นเกมส์ บอร์ด Pi3 มีการดจօหรือไม่ เพราะเหตุใด
5. จงใช้งานเว็บไซต์ www.gsmarena.com เพื่อสืบค้นว่ามีการใช้งานชิปยู ARM Cortex A53 ในเครื่องโทรศัพท์สมาร์ทโฟนหรือไม่ อย่างไร
6. ถ้าจะนำบอร์ด Pi3 ไปปรับปรุงให้กล้ายเป็นเครื่องคอมพิวเตอร์แท็บเล็ต จงจินตนาการว่าจะต้องเพิ่มเติมอุปกรณ์อะไรบ้าง โดยหาข้อมูลเพิ่มเติมในอินเทอร์เน็ต
7. เลเบล main และ printf ในรูปที่ 3.18 ตรงกับฟังก์ชันชี้ออะไرب้าง
8. เลเบล printf ในรูปที่ 3.18 จะต้อง include ไฟล์ .h ชื่ออะไรเพื่อให้คอมไพล์เรอร์ทำงานสำเร็จ
9. คำสั่ง jmp ในรูปที่ 3.18 ตรงกับคำสั่งอะไรในภาษา C/C++
10. คำสั่ง call ในรูปที่ 3.18 ตรงกับคำสั่งอะไรในภาษาแอสเซมบลีของ ARM
11. เลเบล main ในรูปที่ 3.18 ตรงกับบรรทัดที่เท่าไหร่ในตารางที่ 3.3
12. คำสั่ง jmp ในรูปที่ 3.18 ตรงกับคำสั่งใดในตารางที่ 3.3
13. จงอธิบายการทำงานของโปรแกรมในตารางที่ 3.3 ว่าเป็นการวนรอบเพื่อหาค่าอะไรจากตัวแปรใด

ภาษาแอสเซมบลีของ ARM เวอร์ชัน 32 บิต

การพัฒนาโปรแกรมด้วยภาษา C/C++ และแอสเซมบลีมีความเกี่ยวเนื่องกัน ผู้อ่านอาจมีพื้นฐานการพัฒนาโปรแกรมด้วยภาษา C/C++ มาบ้าง จากการทดลองที่ 4 ภาคผนวก E เนื่องจากภาษาแอสเซมบลีของ ARM มีหลายเวอร์ชัน ทั้ง 16, 32 และ 64 บิต ทำให้ผู้อ่านเข้าใจสับสน ดังนั้น บทนี้มีจุดเน้นที่คำสั่งเวอร์ชัน 32 บิตก่อน และวัตถุประสงค์อื่นๆ ดังต่อไปนี้

- เข้าใจโครงสร้าง hart หรือ core ในซีพียู ARM Cortex A53
- เข้าใจภาษาแอสเซมบลีของ ARM เวอร์ชัน 32 บิต
- เข้าใจรูปแบบคำสั่งภาษาแอสเซมบลีของ ARM โดยใช้โครงสร้างของภาษาระดับสูง
- รับรู้วิธีการภาษาแอสเซมบลีของ ARM ซึ่งเป็นที่นิยมในตลาดโลก

4.1 โครงสร้างของซีพียู ARM Cortex A53 ใน BCM2837

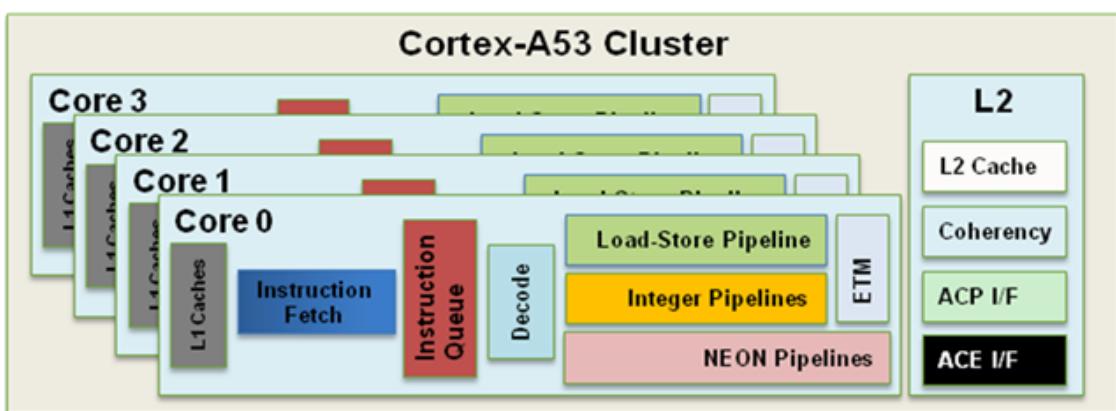


Figure 4.1: โครงสร้างภายในของ ARM Cortex A53 จำนวน 4 คอร์ ที่มา: tomshardware.com

ในบทที่ 3 ผู้อ่านได้เรียนรู้โครงสร้างของชิป BCM2837 ซึ่งเป็นศูนย์กลางของบอร์ด Pi3 โดยมีซีพียู ARM Cortex A53 จำนวน 4 คอร์ ตามรูปที่ 3.3 เป็นองค์ประกอบหลักภายในชิป โดยรูปที่ 4.1 แสดงราย

ลงทะเบียดของ ARM Cortex A53 มีโครงสร้างที่ไม่ซับซ้อน มีการทำงานแบบไปป์ไลน์ (Pipeline) แต่ละคอร์มีโครงสร้างภายในเหมือนกัน ประกอบด้วยวงจรจากข่ายไปข่าว ได้แก่

- **แคชคำสั่งลำดับที่ 1** หรือแคชลำดับที่ 1 Instruction Cache หรือลำดับที่ 1 I-Cache จะเก็บคำสั่งล่าสุด ซึ่งคำสั่งเหล่านี้มาจากการเช็คเม้นท์ Text ของหน่วยความจำเสมือน การทำงานเบื้องต้นของแคชคำสั่งคล้ายกับหน่วยความจำขนาดเล็กเพื่อเก็บคำสั่งล่าสุดที่อ่านมาเพื่อมาอุดรหัส (Decode) และมีขนาดเล็กประมาณ 16-64 กิโลไบท์ หรือเท่ากับ 4,096-16,536 คำสั่ง (4 ไบต์ต่อคำสั่ง)
- **โมดูลหรือวงจรเฟทช์ (Fetch)** ทำหน้าที่ส่งแอดเดรสของคำสั่งที่ต้องการไปยังแคชคำสั่ง (Instruction Cache) ลำดับที่ 1 และรอรับคำสั่งมาพักเก็บในคิว (Instruction Queue)
- **คิวคำสั่ง (Instruction Queue)** ทำหน้าที่เป็นคิวเก็บพักคำสั่งจำนวน 8-16 คำสั่ง เพื่อส่งต่อให้วงจรอุดรหัส
- **วงจรอุดรหัส (Decode)** ทำหน้าที่แปลความหมายของคำสั่งซึ่งเป็นเลขฐานสองที่ส่งมาจากคิวคล้ายกับรูปที่ 3.19 เพื่อส่งสัญญาณควบคุม (Control Signal) ไปยังวงจรไปป์ไลน์ที่เหมาะสมต่อไป
- **วงจรไปป์ไลน์ (Pipeline)** มี 3 ชนิด ประกอบด้วย
 - วงจรไปป์ไลน์สำหรับการอ่าน/เขียนข้อมูลกับหน่วยความจำหลัก (Load-Store Pipeline) เพื่ออ่าน/เขียนข้อมูลกับหน่วยความจำผ่านทาง แคชข้อมูล (Data Cache หรือ D-Cache) ลำดับที่ 1 หรือลำดับที่ 1 D-Cache และแคชลำดับที่ 2 (Level 2)
 - **แคชข้อมูล** หรือ Data Cache หรือลำดับที่ 1 D-Cache มีการทำงานเบื้องต้น คล้ายกับบัฟเฟอร์ขนาดเล็ก เพื่อเก็บข้อมูลล่าสุดที่อ่านหรือเขียนบ่อยๆ ซึ่งข้อมูลเหล่านี้มาจากการเช็คเม้นท์ต่างๆ ที่ไม่ใช่เช็คเม้นท์ Text การทำงานเบื้องต้นของแคชข้อมูลคล้ายกับ แคชคำสั่ง ต่างกันตรงที่ชีพิญสามารถเปลี่ยนแปลงค่าของข้อมูลในแคชข้อมูลได้
 - วงจรไปป์ไลน์สำหรับการประมวลผลเลขจำนวนเต็ม (Integer Pipeline) ขนาด 32 และ 64 บิต การประมวลผล คณิตศาสตร์และตรรกศาสตร์ ข้อมูลแบบสเกลาร์ (Scalar) หรือ ข้อมูลเชิงเดี่ยว Integer Pipeline เพื่อประมวลผลตัวเลขจำนวนเต็มขนาด 32 และ 64 บิต,
 - วงจรไปป์ไลน์สำหรับการประมวลผลเลขจำนวนจริงชนิดทศนิยมลอยตัว (NEON Floating-Point Pipeline) รองรับข้อมูลเชิงเดี่ยวและแบบเวคเตอร์ (Vector) ข้อมูลแบบเวคเตอร์ ได้แก่ ข้อมูลตำแหน่งในเกม 3 มิติ ข้อมูลภาพและเสียง โดยจะสามารถใช้คำสั่งเดี่ยวประมวลผลข้อมูลพร้อมๆ กันหลายตัว เรียกว่า SIMD (Single Instruction Multiple Data)
- **แคชลำดับที่ 2** หรือลำดับที่ 2 โดยทั้งสี่คอร์จะใช้งานแคชลำดับที่ 2 ด้วยกัน หากคอร์ใดหากำคำสั่งหรือข้อมูล ในแคชลำดับที่ 1 (L1) ของตนเองไม่เจอ จะต้องค้นหาคำสั่งหรือข้อมูลในแคชลำดับที่ 2 ต่อไป หากไม่เจออีกจึงค้นต่อในหน่วยความจำหลัก บทที่ 5 จะอธิบายการทำงานของแคชทั้งสองลำดับนี้โดยละเอียด

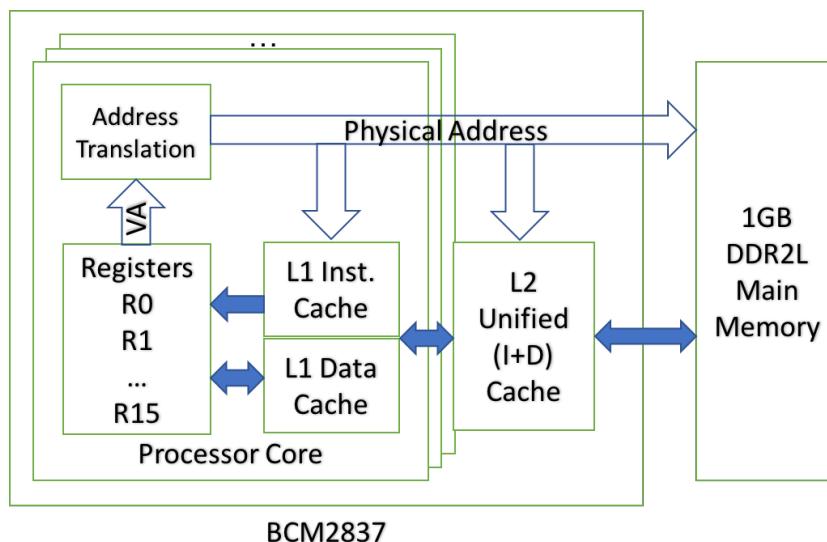


Figure 4.2: โครงสร้างของชิปปี้แต่ละคอร์ประกอบด้วยรีจิสเตอร์ R0-R15 แคชต่างๆ และหน่วยความจำหลัก, (VA: Virtual Address)

ในเชิงโครงสร้าง Integer Pipeline แต่ละคอร์ของ ARM Cortex A53 ในรูปที่ 4.2 ประกอบด้วย รีจิสเตอร์ R0-R15 แคชต่างๆ และหน่วยความจำหลัก เพื่อให้สามารถประมวลผลตัวเลขจำนวนเต็มขนาด 32 บิต โดยอาศัยวงจร ALU (Arithmetic Logic Unit) ตามที่ได้อธิบายไปแล้วในบทที่ 1

ส่วนของวงจร Load-Store Pipeline จะเกี่ยวข้องกับหน่วยความจำ โดยตัวแปรชื่อต่างๆ นั้นถูกกำหนดพื้นที่ไว้ในเซกเมนท์ Data ของหน่วยความจำเมื่อแรกเริ่ม แต่การประมวลผลค่าของตัวแปรเหล่านี้ ชิปปี้จะต้องโหลด (Load) หรืออ่านค่าของตัวแปรจากเซกเมนท์ Data มาพักเก็บในรีจิสเตอร์ก่อน เมื่อประมวลผลแล้วเสร็จ จึงนำค่าจากรีจิสเตอร์ไปเก็บ (Store) หรือเขียนในตำแหน่งที่ได้โหลดมา ด้วยเหตุผลด้านประสิทธิภาพ ซึ่งจะอธิบายในบทที่ 5 ดังนั้น เราจึงเรียกการทำงานลักษณะนี้ว่า สถาปัตยกรรมอ่าน/เขียน (Load/Store Architecture)

4.2 สถาปัตยกรรมชุดคำสั่ง (Instruction Set Architecture)

สถาปัตยกรรมชุดคำสั่งของ ARM เวอร์ชัน 32 บิตเฉพาะการประมวลผลเลขจำนวนเต็มชนิด ไม่มีเครื่องหมาย และ ชนิดมีเครื่องหมาย แบบ 2-Complement ประกอบด้วย

- ชนิดของคำสั่ง แบ่งเป็น
 - คำสั่งประมวลผลตัวแปร
 - คำสั่งการถ่ายโอนข้อมูลระหว่างหน่วยความจำกับรีจิสเตอร์
 - คำสั่งประมวลผลคณิตศาสตร์และตรรกศาสตร์จากข้อมูลในรีจิสเตอร์
 - คำสั่งการควบคุมการทำงาน เพื่อการตัดสินใจ การวนรอบทำซ้ำ และการเรียกฟังค์ชันย่อย
- รีจิสเตอร์สำหรับเก็บข้อมูลชั่วคราว จำนวน 16 ตัว เพื่อรองการประมวลผลด้วยคำสั่งทางคณิตศาสตร์ ตรรกศาสตร์ และอื่นๆ รีจิสเตอร์เหล่านี้ ประกอบด้วย R0, R1, ..., R15 ทุกตัวมีความยาว 32 บิต โดย
 - R15 เรียกว่า โปรแกรมเค้าท์เตอร์ (Program Counter: PC) คือ รีจิสเตอร์สำหรับเก็บ adres ของคำสั่งที่ต้องการจะเพาห์ม่า
 - R14 เรียกว่า ลิงค์รีจิสเตอร์ (Link Register: LR) คือ รีจิสเตอร์สำหรับเก็บ adres ของคำสั่งที่ต้องการจะรีเทิร์นกลับ
 - R13 เรียกว่า สเต็คพอยท์เตอร์ (Stack Pointer: SP) คือ รีจิสเตอร์สำหรับเก็บ adres ของสเต็ค สุดของสเต็ค
 - R4-R12 เป็นรีจิสเตอร์สำหรับใช้งานทั่วไป
 - R0-R3 เป็นรีจิสเตอร์สำหรับการรับ/ส่งข้อมูลระหว่างฟังค์ชัน
- ชนิดและขนาดของข้อมูล พื้นที่และขนาดของหน่วยความจำ (Memory Space) เทียบเคียงตารางที่ 1.1 โดยข้อมูลแต่ละชนิดต้องการพื้นที่ไม่เท่ากัน ดังนี้
 - Byte 8 บิต เหมาะสำหรับตัวแปรชนิดอักขระ ขึ้นอยู่กับผู้เขียนว่าต้องการเก็บอักขระตามรหัสมาตรฐาน ASCII ด้วยพื้นที่ 8 บิตในหน่วยความจำ
 - Halfword 16 บิต เหมาะสำหรับตัวแปรชนิดอักขระตามมาตรฐาน Unicode ด้วยความยาว 16 บิต
 - Word 32 บิต เหมาะสำหรับตัวแปรชนิดจำนวนเต็ม เช่น unsigned int และ int เป็นต้น

ผู้อ่านสามารถค้นควารายละเอียดเพิ่มเติมได้ที่ wikichip.org

4.3 ตัวอย่างคำสั่งภาษาเครื่องในหน่วยความจำ

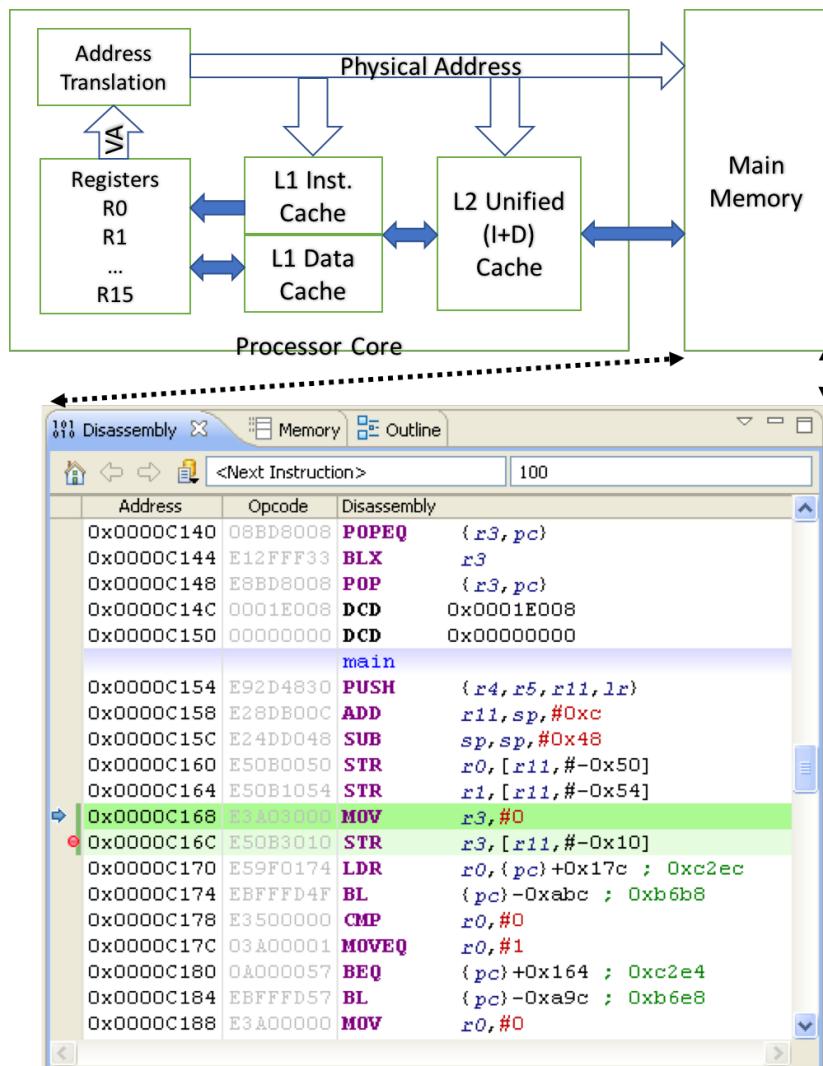


Figure 4.3: คำสั่งของโปรแกรมที่ถูกอ่าน (Load) เข้าสู่หน่วยความจำและแสดงในรูปของภาษาแอสเซมบลีบนโปรแกรมซิมูเลเตอร์ (Simulator) ในบริเวณเซกเมนต์ Text ของหน่วยความจำ ที่มา: infocenter.arm.com

รูปที่ 4.3 แสดงคำสั่งของโปรแกรมที่ถูกอ่าน (Load) เข้าสู่หน่วยความจำและแสดงในรูปของภาษาแอสเซมบลี บนโปรแกรมซิมูเลเตอร์ (Simulator) ประกอบด้วยคอลัมน์ข้อต่อไปนี้

- แอดเดรส (Address) ระบุหมายเลขไปที่บรรจุอพโค้ดในคอลัมน์ทางขวา
- อพโค้ด (Opcode) คือ คำสั่งภาษาเครื่องในรูปของเลขฐานสิบหก ความยาว 32 บิต หรือ 4 ไบต์
- ดิสแอสเซมบลี (Disassembly) คือ การแปลคำสั่งภาษาเครื่องให้กลับเป็นภาษาแอสเซมบลี ARM

ยกตัวอย่างเช่น หน่วยความจำตำแหน่งที่ 0x0000_C140 จำนวน 4 ไบต์บรรจุอพโค้ด ค่า 0x08BD_8008 ซึ่งตรงกับคำสั่ง POPEQ r3,pc ขีดล่าง '_' ที่ผู้เขียนใส่เพิ่มทำให้ผู้อ่านสามารถอ่านหมายเลขที่เรียงติดกันทีละ 4 หลักได้ง่ายขึ้น

ตำแหน่งถัดไปคือ 0x0000_C144 จำนวน 4 byte บรรจุอพโคด ค่า 0xE12F_FF33 ซึ่งตรงกับคำสั่ง BLX r3 คำสั่ง POPEQ หมายถึง คำสั่ง POP เมื่อผลการคำนวนเปรียบเทียบก่อนหน้ามีค่าเท่ากัน (EQ: EQUAL) การ POP r3, pc คือ การอ่านข้อมูลจากหน่วยความจำที่ pc เก็บ ไปบรรจุใน r3 เมื่อประมวลผลคำสั่งที่แอ็ดเดรส 0x0000_C144 สำเร็จแล้ว pc จะเพิ่มขึ้น 4 byte เป็น 0x0000_C148 เพื่อนำอพโคด ค่า 0xE12F_FF33 ซึ่งตรงกับคำสั่ง BLX r3 ต่อไป

pc คือ R15 รีจิสเตอร์สำหรับเก็บแอ็ดเดรสในหน่วยความจำของคำสั่งปัจจุบัน โดยสังเกตได้จากลูกศรสีนำเงิน เพื่อให้ซีพียูสามารถอ่านคำสั่งนั้น ไปประมวลผล ซึ่งในรูป คือ หมายเลข 0x0000_C168 บรรจุอพโคด ค่า 0xE3A0_3000 ซึ่งตรงกับคำสั่ง MOV r3, #0

เมื่อประมวลผลคำสั่งที่แอ็ดเดรส 0x0000_C168 สำเร็จแล้ว pc จะเพิ่มขึ้น 4 byte เป็น 0x0000_C16C เพื่อนำอพโคด ค่า 0xE50B_3010 ซึ่งตรงกับคำสั่ง STR r3, [r11, #-0x10] ต่อไป แต่ในรูป ผู้ใช้ได้ใส่ Break Point โดยสังเกตได้จากวงกลมสีแดงทางซ้ายสุด Break Point จะทำให้โปรแกรมหยุดทำงานชั่วขณะ เพื่อให้ผู้ใช้สามารถศึกษาการทำงานของคำสั่งจากค่ารีจิสเตอร์ต่างๆ ที่เกี่ยวข้อง หลังจากนั้น ผู้ใช้สามารถดำเนินการรันต่อไปได้

4.4 การประกาศและตั้งค่าตัวแปรในหน่วยความจำหลัก

ผู้อ่านต้องระลึกไว้ว่าเสมอว่าตัวแปรต่างๆ อยู่ในหน่วยความจำเสมอ (Variables are always in memory.) รีจิสเตอร์เป็นแค่ที่เก็บพกข้อมูลชั่วคราวสำหรับการประมวลผล เมื่อคำนวณเสร็จแล้ว ข้อมูลในรีจิสเตอร์จะถูกถ่ายโอนกลับไปยังหน่วยความจำ คำว่า หน่วยความจำ หมายถึง หน่วยความจำหลัก

รูปแบบ	ความหมาย
VAR_LABEL DCD 0	ตัวแปร VAR_LABEL มีค่าเริ่มต้นเท่ากับ 0
var_label: .word 7	ตัวแปร var_label มีค่าเริ่มต้นเท่ากับ 7 และมีขนาดเท่ากับ 1 word = 4 ไบท์

Table 4.1: ตัวอย่างโปรแกรมภาษาแอสเซมบลีเพื่อสร้างตัวแปรขนาด 32 บิต จำนวน 2 ตัวแปร ในพื้นที่ของเซกเมนท์ Data

#	เลขบล.	คำสั่ง	คอมเมนท์
1		.data	@Variable definition
2		.balign 4	@Align variable to 4-byte space
3	wordvar1:	.word 7	@wordvar1=7
4		.balign 4	@Align variable to 4-byte space
5	wordvar2:	.word 3	@wordvar2=3

ตารางที่ 4.1 ตัวอย่างโปรแกรมภาษาแอสเซมบลีเพื่อสร้างตัวแปรขนาด 32 บิต จำนวน 2 ตัวแปร บรรทัดที่ 1 คำสั่ง .data คือ การบ่งบอกถึงการเริ่มต้นประกาศตัวแปร คำสั่ง .balign คือการสั่งให้การจด ข้อมูลในหน่วยความจำ ตรงกับพื้นที่ 4 ไบท์ เพื่อให้วงจรบริหารจัดการได้ง่ายขึ้น ตัวแปรชนิดจำนวนเต็ม ความยาว 1 Word ต้องการพื้นที่ 4 ไบท์ ซึ่งในบรรทัดที่ 3 ของตารางที่ 4.1 เป็นการตั้งค่าเริ่มต้นเท่ากับ 7_{10} ให้กับตัวแปร wordvar1 หรือเท่ากับ 0000_0007_{16} ในบรรทัดที่ 5 เป็นการตั้งค่าเริ่มต้นเท่ากับ 3_{10} ให้กับตัวแปร wordvar2 หรือเท่ากับ 0000_0003_{16}

4.5 คำสั่งถ่ายโอนข้อมูลระหว่างหน่วยความจำและรีจิสเตอร์

คำสั่งถ่ายโอนข้อมูลระหว่างหน่วยความจำและรีจิสเตอร์ เป็นคำสั่งที่จำเป็นสำหรับโปรแกรม คอมพิวเตอร์ เนื่องจากตัวแปรที่ประกาศไว้ คือ ชื่อของข้อมูลและชุดข้อมูลหรืออเรย์ในหน่วยความจำ โดยขนาดของข้อมูลจะเปลี่ยนแปลงตามชนิด เช่น word'ขนาด 32 บิต byte มีขนาด 8 บิต เป็นต้น ดังนั้น ซึ่งมีจะต้องใช้คำสั่งโหลดหรืออ่านค่าของตัวแปรนั้นมาพักไว้ในรีจิสเตอร์ก่อน โดยใช้คำสั่ง LDR สำหรับข้อมูลชนิด word และ LDRB สำหรับข้อมูลชนิด byte ในตารางต่อไปนี้

รูปแบบ	ความหมาย (m และ n คือ หมายเลขรีจิสเตอร์มีค่าเท่ากับ 0-15)
LDR Rd [Rn]	Rd = Mem[Rn] (ก็อปปี้ข้อมูล 32 บิต)
STR Rd [Rn]	Mem[Rn] = Rd (ก็อปปี้ข้อมูล 32 bit)
LDRB Rd [Rn]	Rd = Mem[Rn] (ก็อปปี้ข้อมูล 8 บิตล่างสุด)
STRB Rd [Rn]	Mem[Rn] = Rd (ก็อปปี้ข้อมูล 8 บิตล่างสุด)
PUSH register list	ก็อปปี้ข้อมูลจากรีจิสเตอร์ไปวางบนสแต็คชั่วร้า
POP register list	ก็อปปี้ข้อมูลจากสแต็คกลับไปคืนให้รีจิสเตอร์

ในทางกลับกัน คำสั่ง Sto จะสั่งให้หารดแวร์ภายในซีพียูดำเนินการ อ่านค่าจากรีจิสเตอร์ไปบันทึกในหน่วยความจำ ณ แอดเดรสที่ตัวแปรนั้นตั้งอยู่

คำสั่ง LDR Rd [Rn] จะสั่งให้ซีพียูก็อปปี้ข้อมูล 32 บิต จากหน่วยความจำ ณ แอดเดรสที่รีจิสเตอร์ Rn เก็บไปฝากไว้ในรีจิสเตอร์ Rd ซึ่งตรงกับความหมายของproxycon นี้ $Rd = Mem[Rn]$

คำสั่ง STR Rd [Rn] จะสั่งให้ซีพียูก็อปปี้ข้อมูล 32 บิต จากรีจิสเตอร์ Rd ไปเก็บยังหน่วยความจำ ณ แอดเดรสที่รีจิสเตอร์ Rn ซึ่งตรงกับความหมายของ proxycon นี้ $Mem[Rn] = Rd$

คำสั่ง PUSH และ POP มีลักษณะพิเศษกว่าคำสั่ง Load และ Store ทั่วไป คือ PUSH register list จะทำการก็อปปี้ข้อมูลจากรีจิสเตอร์ไปวางบนสแต็คชั่วร้า โดยสแต็ค คือ ชื่อย่อของสแต็คเชิงเมนท์ เป็นพื้นที่ในหน่วยความจำเสมือน ในรูปที่ 3.12 สำหรับ คำสั่ง POP register list จะทำงานตรงข้ามกับคำสั่ง PUSH คือ ก็อปปี้ข้อมูลจากสแต็คกลับไปคืนให้รีจิสเตอร์

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

รูปแบบ	ความหมาย (m และ n คือ หมายเลขเรจิสเตอร์มีค่าเท่ากับ 0-15)
MOV Rd, #Imm	Rd = #Imm
LDR Rd, [Rn]	Rd = Mem[Rn]
LDR Rd, [Rn, #Imm]	Rd = Mem[Rn + #Imm] Rn unchanged
LDR Rd, [Rn], #Imm	Rd = Mem[Rn] Rn = Rn + #Imm
LDR Rd, [Rn, #Imm]!	Rd = Mem[Rn+ #Imm] Rn = Rn + #Imm
LDR Rd, [Rn, Rm]	Rd = Mem[Rn+Rm] (32 bit copy) Rn unchanged
STR Rd [Rn, Rm]	Mem[Rn+Rm] = Rd (32 bit copy) Rn unchanged
LDR Rd, =varaddr	Rd = address(var)

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

Table 4.2: ตัวอย่างโปรแกรมภาษาแอสเซมบลีเพื่ออ่านค่าตัวแปรจากหน่วยความจำ โดยการอ่านตำแหน่งของตัวแปร

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

ตัวอย่างการเขียนโปรแกรม

x = (a + b) - c;

Table 4.3: ตัวอย่างโปรแกรมภาษาแอสเซมบลีเพื่อคำนวณประยะค $x = (a + b) - c$

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

4.6 คำสั่งประมวลผลข้อมูลในรีจิสเตอร์ (Register Data Processing Instructions)

คำสั่งประมวลผลรีจิสเตอร์ (Register Processing Instructions) เชื่อมโยงกับคณิตศาสตร์ และข้อมูลชนิดต่างๆ ในหัวข้อนี้ จะจำกัดอยู่ที่ข้อมูลจำนวนเต็มก่อน เพื่อให้เนื้อหาไม่ซับซ้อนจนเกินไป เนื้อหาแบ่งเป็น

- คำสั่งทางคณิตศาสตร์ เพื่อคำนวณเลขจำนวนเต็มชนิดมีและไม่มีเครื่องหมาย
- คำสั่งเลื่อนบิต เพื่อเลื่อนบิตข้อมูลไปทางซ้ายและขวา
- คำสั่งทางคณิตศาสตร์และเลื่อนบิต เพื่อคำนวณเลขจำนวนเต็มชนิดมีและไม่มีเครื่องหมาย หลังจากที่มีการเลื่อนบิตไปทางซ้ายหรือขวา
- คำสั่งทางตรรกศาสตร์ เพื่อคำนวณค่าทางตรรกศาสตร์ของเลขจำนวนเต็มชนิดมีและไม่มีเครื่องหมาย

4.6.1 คำสั่งทางคณิตศาสตร์

คำสั่งสำหรับเลขจำนวนเต็มชนิดไม่มีเครื่องหมาย บวก ลบ คูณ kratทำโดยวงจรไฮาร์ดแวร์ เรียกว่า ALU (Arithmetic and Logic Unit) ซึ่งประกอบด้วยวงจรบวก/ลบเลข คูณเลข ตามหลักการที่ได้กล่าวมาในหัวข้อที่ 1.3

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)

การตรวจสอบใบเวอร์ไฟล์ เครื่องหมาย ศูนย์ และตัวทดบิตสุดท้าย ผู้อ่านสามารถศึกษารายละเอียดของการตรวจจับใบเวอร์ไฟล์ ได้ในหัวข้อที่ 1.3.1.1 และ 1.3.2 โดยค่าบิต NZCV ในรีจิสเตอร์สถานะ (Status Register) จะเปลี่ยนแปลงตามผลลัพธ์ที่ ALU คำนวณตามชนิดของข้อมูลและออพโค้ด ต่างๆ โดย

- บิต Z (Zero) = 1 ใช้ตรวจสอบว่าผลลัพธ์มีค่าเท่ากับ ศูนย์
- บิต C (Carry) = 1 ใช้ตรวจสอบว่าผลลัพธ์จากการคำนวณมีบิตทด (Carry bit c_n) เท่ากับ 1
- บิต N (Negative) = 1 ใช้ตรวจสอบว่าผลลัพธ์มีค่าน้อยกว่าศูนย์ หรือ ติดลบ

- บิต V ($\circ\text{Overflow}$) = 1 ใช้ตรวจสอบว่าผลการคำนวนเกิดโอเวอร์โฟล์ว

บิตที่ 31-28 ของรูปที่ 4.4 ค่าของรีจิสเตอร์เหล่านี้เปลี่ยนแปลงตามคำสั่งและข้อมูลภายในรีจิสเตอร์ R0-R12 ที่นำมาประมวลผล เพื่อนำไปเป็นผลลัพธ์ที่ จริง (True) หรือ เท็จ (False) ของเงื่อนไขต่างๆ ในคำสั่งควบคุมการทำงานในหัวข้อที่ 4.7

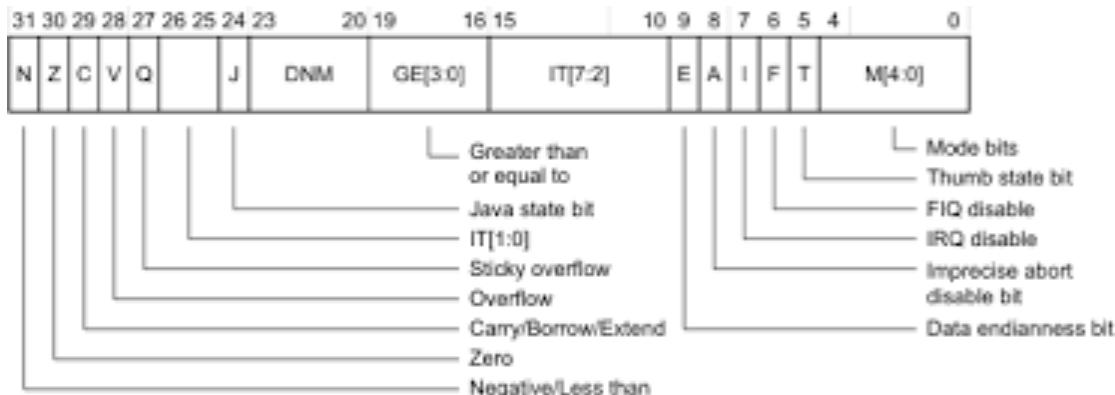


Figure 4.4: รีจิสเตอร์สำหรับเก็บสถานะของซีพียู ณ ปัจจุบัน (Current Program Status Register: CPSR) infocenter.arm.com

4.6.2 คำสั่งเลื่อนบิตข้อมูล

คำสั่งเลื่อนบิตข้อมูลแบ่งเป็น การเลื่อนบิตทางซ้ายและทางขวา โดยวงจรบาร์เรลชิฟเตอร์ (Barrel Shifter) ในรูปที่ 4.5 การเลื่อนบิตไปทางขวาจะแบ่งเป็นการเลื่อนทางตรรกศาสตร์ (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)
ASR Rd, Rn, Rm	Rd = Rn » Rm (Arithmetic Shift Right)
ASR Rd, Rn, #Imm	Rd = Rn » #Imm (Arithmetic Shift Right)

การเลื่อนบิตข้อมูลไปทางซ้าย คือ การคูณ ด้วยสองยกกำลังจำนวนบิตที่เลื่อน โดยการเลื่อนทางตรรกศาสตร์ บิตทางขวาสุดจะป้อนศูนย์เข้ามาแทนที่ข้อมูลเดิมที่ถูกเลื่อนไปทางซ้าย

การเลื่อนบิตข้อมูลไปทางขวา คือ การหาร ด้วยสองยกกำลังจำนวนบิตที่เลื่อน โดยการเลื่อนทางตรรกศาสตร์ บิตทางซ้ายสุดจะป้อนศูนย์เข้ามาแทนที่ข้อมูลเดิมที่ถูกเลื่อนไปทางขวาโดยใช้กับเลขจำนวนเต็มชนิดไม่มีเครื่องหมาย

แต่การเลื่อนบิตทางคณิตศาสตร์ (Arithmetic Shift) จะจะป้อนบิตทางซ้ายสุดด้วยบิตเครื่องหมายเข้ามาแทนที่ข้อมูลเดิมที่ถูกเลื่อนไปทางขวา เพื่อให้เครื่องหมายไม่เปลี่ยนแปลง นิยมใช้กับเลขจำนวนเต็มชนิดมีเครื่องหมาย แบบ 2-Complement

4.6.3 คำสั่งทางคณิตศาสตร์และเลื่อนบิทพร้อมกัน

คำสั่งสามารถทำการเลื่อนบิทข้อมูลก่อนจะนำผลที่ได้ไปคำนวณต่อ โดยไม่กระทบต่อค่าของรีจิสเตอร์ที่ถูกเลื่อน

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

จุดเด่นของ ARM คือ คำสั่ง MOV และคำสั่งทางคณิตศาสตร์ เช่น ADD, SUB และ RSB สามารถใช้งานร่วมกับคำสั่งการเลื่อนบิทหรือการซิฟต์ ตามรูปต่อไปนี้

add r3, r4, r2, lsl #2; $r3 = r4 + (r2 \ll 2)$

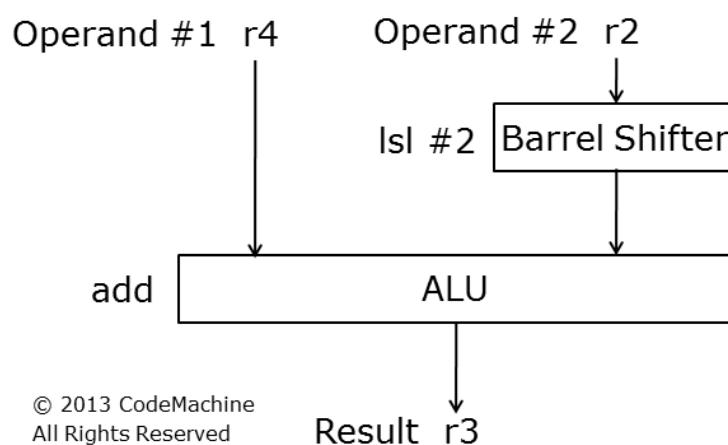


Figure 4.5: ตัวอย่างคำสั่งการบวกค่าในรีจิสเตอร์ที่ได้จากการซิฟท์ไปทางซ้ายจำนวน 2 บิต ที่มา: CodeMachine.com

ตัวอย่างคำสั่งการบวกค่าในรีจิสเตอร์ที่ได้จากการซิฟท์ไปทางซ้ายจำนวน 2 บิต หรือเท่ากับ $R3 = R4 + (R2 \times 4)$

4.6.4 คำสั่งทางตรรกศาสตร์

คำสั่งสำหรับเลขจำนวนเต็มชนิดไม่มีเครื่องหมาย ได้แก่ กระบวนการ AND OR NOT ประมวลผลพร้อมกันทุกบิท โดยบิทข้อมูลตัวตั้งจะกระทำกับบิทที่ตรงกันของตัวกระทำเท่านั้น

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 = Rm (bitwise Inverse)
MVN Rd, #Imm	Rd = #Imm (bitwise Inverse)
EOR Rd, Rn, Rm	Rd = Rn ^ Rm (bitwise XOR)
EOR Rd, Rn, #Imm	Rd = Rn ^ #Imm (bitwise XOR)

4.7 คำสั่งควบคุมการทำงาน (Control Instructions)

คำสั่งควบคุมการทำงาน (Control Instructions) นำไปประกอบเป็น ประโยชน์การตัดสินใจ เช่น ประโยชน์ IF IF-ELSE Switch-Case การวนรอบชนิดต่างๆ เช่น FOR, WHILE, DO-WHILE ในภาษาสูง ใน ARM จะตรงกับคำสั่ง Branch เพื่อย้ายการทำงานไปยังคำสั่งเป้าหมาย โดยเปรียบเทียบค่ารีจิสเตอร์ตามตัวอย่าง คำสั่ง CMP สำหรับการเปรียบเทียบของเลขจำนวนเต็ม ผลลัพธ์ที่ได้จากการรันคำสั่ง CMP คือ ค่าในรีจิสเตอร์ CPSR โดยเฉพาะบิต NZCV ในหัวข้อที่ 1.3 จะเกิดการเปลี่ยนแปลง คำสั่งเปรียบเทียบ CMP เนื่องได้ 2 รูปแบบตามตารางต่อไปนี้

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

เมื่อทำการเปรียบเทียบด้วยคำสั่ง CMP และ โปรแกรมจะทำการกระโดดหรือเปลี่ยน คำสั่งการทำงาน ไปยัง เลเบลเป้าหมาย (Target Label) โดยใช้คำสั่งชนิด branch (B) ตามรูปแบบต่างๆ ในตารางต่อไปนี้

รูปแบบ	ความหมาย
B label	กระโดดไปยัง label (อย่างไม่มีเงื่อนไข)
BEQ label	กระโดดไปยัง label หากผลลัพธ์การเปรียบเทียบท่ากัน
BNE label	กระโดดไปยัง label หากผลลัพธ์การเปรียบเทียบไม่เท่ากัน
BGT label	กระโดดไปยัง label หากผลลัพธ์การเปรียบเทียบมากกว่าจริง
BLT label	กระโดดไปยัง label หากผลลัพธ์การเปรียบเทียบน้อยกว่าจริง
BGE label	กระโดดไปยัง label หากผลลัพธ์การเปรียบเทียบมากกว่าหรือเท่ากับจริง
BLE label	กระโดดไปยัง label หากผลลัพธ์การเปรียบเทียบน้อยกว่าหรือเท่ากับจริง

คำสั่ง B (branch) ตรงกับคำสั่ง Jump ในภาษาแอสเซมบลีอื่นๆ หรือ การกระโดด Go-to ในภาษา C/C++ ในเชิงลึก คำสั่ง branch จะทำให้ค่าของรีจิสเตอร์ PC เปลี่ยนแปลง ตามค่าตำแหน่ง label ที่จะกระโดดไป

วงจรดิจิทัลในซีพียูสามารถตรวจสอบผลลัพธ์ของเงื่อนไขต่างๆ ของคำสั่ง branch 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 or Same คือ เงื่อนไขสำหรับข้อมูลชนิดไม่มีเครื่องหมายว่ามากกว่า โดยตรวจสอบว่า Carry =1
4. **CC** (Carry Clear): Unsigned Lower คือ เงื่อนไขสำหรับข้อมูลชนิดไม่มีเครื่องหมายว่าน้อยกว่า โดยตรวจสอบว่า Carry = 0
5. **MI** (Minus): Negative Set คือการตรวจสอบว่า $N=1$ หรือไม่ นั่นคือ ผลการลบน้อยกว่า 0
6. **PL** (Plus or Zero): Negative Not Set คือการตรวจสอบว่า $N=0$ หรือไม่ นั่นคือ ผลการลบมากกว่าหรือเท่ากับ 0
7. **VS** (Overflow Set): เกิด Overflow ขึ้น คือการตรวจสอบว่า $V=1$ หรือไม่ นั่นคือ เกิดโอเวอร์เฟล์ว
8. **VC** (Overflow Clear): ไม่เกิด Overflow คือการตรวจสอบว่า $V=0$ หรือไม่ นั่นคือ ไม่เกิดโอเวอร์เฟล์ว
9. **HI** (Unsigned Higher): คือ เงื่อนไขสำหรับข้อมูลชนิดไม่มีเครื่องหมายว่ามากกว่า โดยตรวจสอบว่า Carry=1 or Zero=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 คือการตรวจสอบว่า $N!=V$ หรือไม่ นั่นคือ ผลการลบน้อยกว่า 0
13. **GT** (Greater Than): !Zero && Negative = Overflow คือการตรวจสอบว่า $Z=0$ และ $N=V$ หรือไม่ นั่นคือ ผลการลบมากกว่า 0
14. **LE** (Less Than or Equal): Zero && Negative != Overflow คือการตรวจสอบว่า $Z=1$ และ $N!=V$ หรือไม่ นั่นคือ ผลการลบน้อยกว่าหรือเท่ากับ 0
15. **AL** (Always): ไม่ตรวจสอบ

4.7.1 การตัดสินใจ IF

การตัดสินใจ IF ขึ้นอยู่กับเงื่อนไขของประโภค IF ว่าผลลัพธ์ที่ได้จะเป็นจริงหรือเท็จ เมื่อจริง ซีพียูจะปฏิบัติตามประโยคคำสั่งที่โปรแกรมเมอร์ต้องการ หากเท็จ ซีพียูจะทำคำสั่งอื่นๆ ต่อไป

ตัวอย่างการเขียนโปรแกรม ประโภค IF ในภาษา C/C++

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

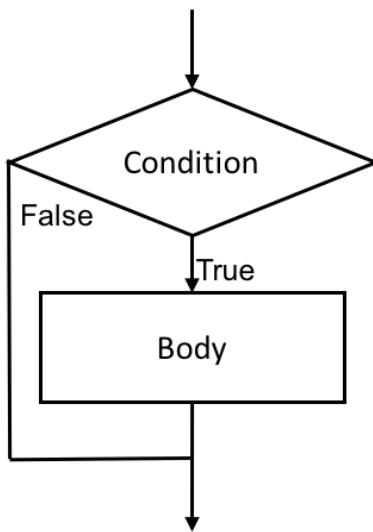


Figure 4.6: โฟลว์ชาร์ตการทำงานของโครงสร้างการเขียนโปรแกรม IF

จากประโภค IF นี้ คอมไපเลอร์สามารถแปลเป็นชุดคำสั่งภาษาแอสเซมบลีของ ARM ได้ในตารางที่ 4.4 ผู้อ่านสามารถทำความเข้าใจการทำงานได้จากคำอธิบายภายใต้คอลัมน์ คอมเมนท์ และคำอธิบายเป็นภาษาไทยตามหมายเลขอรรถัด ดังนี้

1. คือ การโหลดตัวแหน่งของตัวแปร a ไปบรรจุใน R4
2. คือ การโหลดค่าของตัวแปร a ไปบรรจุใน R0
3. คือ การโหลดตัวแหน่งของตัวแปร b ไปบรรจุใน R4
4. คือ การโหลดค่าของตัวแปร b ไปบรรจุใน R1
5. คือ การคำนวณค่า $a + b$ ไปบรรจุใน R3
6. คือ การโหลดตัวแหน่งของตัวแปร c ไปบรรจุใน R4
7. คือ การโหลดค่าของตัวแปร c ไปบรรจุใน R4
8. คือ การเปรียบเทียบค่าของ R2 และ R3

Table 4.4: ตัวอย่างโปรแกรมตามประโยคเงื่อนไข if

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

9. คือ หากเงื่อนไข Less Than or Equal (LE) เป็นจริง ซึ่งจะกระโดดไปทำงานคำสั่งที่ขึ้นต้นด้วย exit หากไม่เป็นจริง ซึ่งจะทำงานประโยคที่ 10 ต่อไป
10. คือ การโหลดตัวแหน่งของตัวแปร x ไปบรรจุใน R4
11. คือ การโหลดค่าของตัวแปร x ไปบรรจุใน R5
12. คือ การโหลดตัวแหน่งของตัวแปร y ไปบรรจุใน R4
13. คือ การโหลดค่าของตัวแปร y ไปบรรจุใน R6
14. คือ การคำนวนค่า a + b ไปบรรจุใน R5
15. คือ การโหลดตัวแหน่งของตัวแปร x ไปบรรจุใน R4
16. คือ การสโตร์ค่าของ R5 ไปบรรจุตัวแหน่งของตัวแปร x
17. คือ คำสั่งที่ขึ้นต้นด้วย exit

4.7.2 การตัดสินใจ IF-ELSE

การตัดสินใจ IF-ELSE ขึ้นอยู่กับเงื่อนไขของประโยค IF ว่าผลลัพธ์ที่ได้จะเป็นจริงหรือเท็จ เมื่อจริง ซึ่งจะปฏิบัติตามประโยคคำสั่งที่โปรแกรมเมอร์ต้องการ เมื่อแล้วเสร็จซึ่งจะกระโดดข้ามส่วน ELSE เพื่อประมวลผลต่อ หากเท็จ ซึ่งจะทำคำสั่งส่วนที่ ELSE นำหน้า เมื่อแล้วเสร็จจะประมวลผลต่อ ตามโฟล์ชาร์ตในรูปที่ 4.7

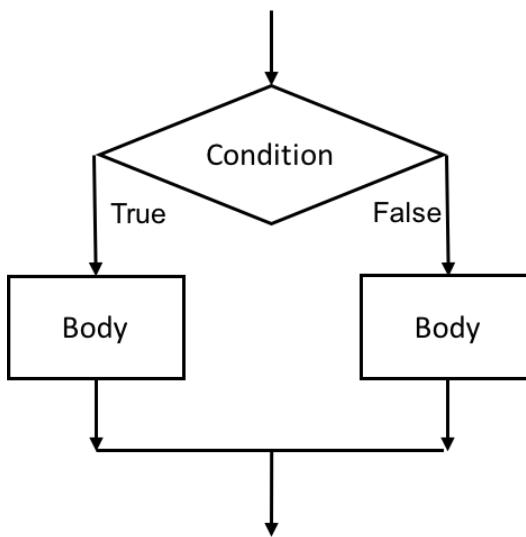


Figure 4.7: โฟล์ชาร์ตการทำงานของโครงสร้างการเขียนโปรแกรม IF-ELSE

ตัวอย่างการเขียนโปรแกรม ประโยค IF-ELSE ในภาษา C/C++

```

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

จากประโยค IF-ELSE นี้ คอมไพล์เตอร์สามารถแปลเป็นชุดคำสั่งภาษาแอสเซมบลีของ ARM ได้ในตารางที่ 4.5 ผู้อ่านสามารถทำความเข้าใจการทำงานได้จากคำอธิบายภายใต้คอลัมน์ คอมเมนท์ และคำอธิบายเป็นภาษาไทยตามหมายเลขอรรถทั้ด ดังนี้

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 การเปรียบเทียบค่าของ R2 และ R3

Table 4.5: ตัวอย่างโปรแกรมตามประโยคเงื่อนไข if-else

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

9. คือ หากเงื่อนไข Less Than or Equal (LE) เป็นจริง ซึ่งจะกระโดดไปทำงานคำสั่งที่ขึ้นต้นด้วย exit หากไม่เป็นจริง ซึ่งจะทำงานประโยคที่ 10 ต่อไป
10. คือ การโหลดตำแหน่งของตัวแปร x ไปบรรจุใน R4
11. คือ การโหลดค่าของตัวแปร x ไปบรรจุใน R5
12. คือ การโหลดตำแหน่งของตัวแปร y ไปบรรจุใน R4
13. คือ การโหลดค่าของตัวแปร y ไปบรรจุใน R6
14. คือ การคำนวณค่า $x = x + y$ ไปบรรจุใน R5
15. คือ การโหลดตำแหน่งของตัวแปร x ไปบรรจุใน R4
16. คือ การสโตร์ค่าของ R5 ไปบรรจุตำแหน่งของตัวแปร x
17. คือ การบังคับให้ซึ่งจะกระโดดไปทำงานคำสั่งที่ขึ้นต้นด้วย exit

18. คือ การโหลดตัวแหน่งของตัวแปร x ไปบรรจุใน R4
19. คือ การโหลดค่าของตัวแปร x ไปบรรจุใน R5
20. คือ การโหลดตัวแหน่งของตัวแปร y ไปบรรจุใน R4
21. คือ การโหลดค่าของตัวแปร y ไปบรรจุใน R6
22. คือ การคำนวนค่า $x = x - y$ ไปบรรจุใน R5
23. คือ การโหลดตัวแหน่งของตัวแปร x ไปบรรจุใน R4
24. คือ การสโตร์ค่าของ R5 ไปบรรจุตัวแหน่งของตัวแปร x
25. คือ คำสั่งที่ขึ้นต้นด้วย exit

4.7.3 การวนรอบชนิด FOR

ในอดีต การพัฒนาโปรแกรมคอมพิวเตอร์มุ่งเน้นที่การคำนวนแก้ปัญหาทางคณิตศาสตร์ ยกตัวอย่างเช่น

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

สมการที่ 4.1 คือ การบวกเลขตั้งแต่ค่า $i = 1$ จนถึง $i = 10$ ในรูปแบบทางคณิตศาสตร์ อย่างง่าย โปรแกรมเมอร์สามารถใช้ภาษาได้ภาษาหนึ่ง เช่น ภาษา C เขียนการบวกนี้ในรูปของประโยชน์ค วนรอบ for ซึ่งเข้าใจได้ง่ายที่สุด ดังนี้

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

ซอฟต์แวร์นี้ ทำการตั้งค่าเริ่มต้นให้ตัวแปร i เท่ากับ 1 และจึงการบวกค่า x กับ i และนำผลลัพธ์ที่ได้เก็บค่าไว้ ในตัวแปร x เพื่อที่จะวนกลับมาทำประโยชน์อีก โดย i จะถูกเพิ่มค่าเป็น $i + 1$ เช่นกัน ประโยชน์ค $x = x + i$ จะทำงานทั้งหมด 10 รอบตามเงื่อนไข $i \leq 10$

เราเรียก i ว่าเป็นตัวแปรวนรอบ (Index) เนื่องจากเป็นตัวแปรที่ใช้นับเลขรอบ ประโยชน์ค $i=1$ เรียกว่า Index Initialization คือ การตั้งค่าเริ่มต้นให้กับตัวแปรวนรอบ ซึ่งเป็นองค์ประกอบหนึ่งของประโยชน์ค for ประโยชน์ค $x=x + i$ เรียกว่า Loop Body ตามโฟล์ชาร์ตในรูปที่ 4.8 ประโยชน์คเงื่อนไข $i \leq 10$ เรียกว่า Condition เพื่อตรวจสอบว่าเป็นจริงหรือเท็จ เมื่อ i มีค่าเท่ากับ 1 ถึง 10 เงื่อนไขจะให้ผลลัพธ์เป็นจริง (True) เมื่อ i มีค่าเท่ากับ 11 ผลลัพธ์จะกลายเป็นเท็จ (False) และจะไม่เกิดการวนรอบ เพื่อทำงานประโยชน์คที่ต่อจากประโยชน์ค for ประโยชน์ค $i=i+1$ เรียกว่า Index Update คือ การเปลี่ยนแปลงค่าตัวแปรวนรอบให้สอดคล้องกับเงื่อนไข ตามที่กล่าวมาข้างต้น

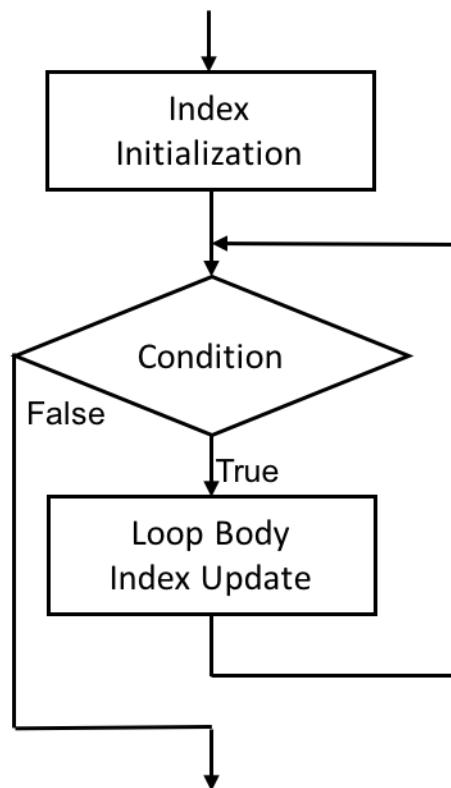


Figure 4.8: โฟล์ชาร์ตการทำงานของการวนรอบชนิด FOR

Table 4.6: ตัวอย่างโปรแกรมตามประโยคุนรอบ For

#	เลbel	คำสั่ง	คอมเมนท์
1		...	@ Initialize R0=0
2		...	@ Initialize R1=1
3	for:	CMP R1, #10	@ Compute R1-10
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 while
8		...	@ End of while loop

ผู้อ่านสามารถทำความเข้าใจการทำงานได้จากคำอธิบายภายใต้คอลัมน์ คอมเมนท์ และคำอธิบายเป็นภาษาไทยตามหมายเลขอรรถทัด ดังนี้

1. คือ การตั้งค่าเริ่มต้นให้กับ $R0 = 0$
2. คือ การตั้งค่าเริ่มต้นให้กับ $R1 = 1$
3. คือ คำสั่งที่ขึ้นต้นด้วย for เพื่อทำการเปรียบเทียบ $R1$ กับค่าคงที่ 10_{10}
4. คือ หากเงื่อนไข Less Than or Equal (LE) เป็นจริง ซึ่งจะกระโดดไปทำงานคำสั่งที่ขึ้นต้นด้วย end หากไม่เป็นจริง ซึ่งจะทำงานประโยคที่ 5 ต่อไป
5. คือ การคำนวณค่า $a + b$ ไปบรรจุใน $R3$

6. คือ การโหลดตำแหน่งของตัวแปร C ไปบรรจุใน R4
7. คือ การโหลดค่าของตัวแปร C ไปบรรจุใน R4
8. คือ การเปรียบเทียบค่าของ R2 และ R3
9. คือ การบังคับให้ซีพียูกระโดดไปทำงานคำสั่งที่ขึ้นต้นด้วย end
10. คือ คำสั่งที่ขึ้นต้นด้วย end

โดยทั่วไป ตัวอย่างนี้เป็นการวนรอบชนิด For ในรูปที่ 4.8 การตั้งค่าเริ่มต้น (Initialization) ให้กับตัวแปรวนรอบ (Index) หลังจากนั้น โปรแกรมจะตรวจสอบเงื่อนไข Condition ของลูป หากเป็นจริง โปรแกรมจะทำงานตามคำสั่งจำนวนหนึ่ง ที่เรียกว่า Body เมื่อแล้วเสร็จ โปรแกรมจะเพิ่ม (Increment) หรือ ลด (Decrement) ตัวแปรลูป (Index) แล้วจึงกลับไปบรรทัดที่มีประโยคเงื่อนไขข้ามแล้วข้ามอีก จนเงื่อนไขเป็นเท็จ (False) ลูป FOR จะสิ้นสุดการทำงาน และจึงย้ายการทำงานไปคำสั่งอื่นภายนอกลูป

4.7.4 การวนรอบชนิด WHILE

การวนรอบ WHILE คล้ายกับการวนรอบ FOR คือ การวนรอบจะเกิดขึ้นอยู่กับเงื่อนไขของประโยค WHILE ว่าผลลัพธ์ที่ได้จะเป็นจริงหรือเท็จ เมื่อจริง ซีพียูจะปฏิบัติตามประโยคคำสั่งที่โปรแกรมเมอร์ต้องการ แล้วก็ลับไปตรวจสอบเงื่อนไขอีกรอบ หากเท็จ ซีพียูจะทำคำสั่งอื่นๆ ต่อไป ในรูปที่ 4.9

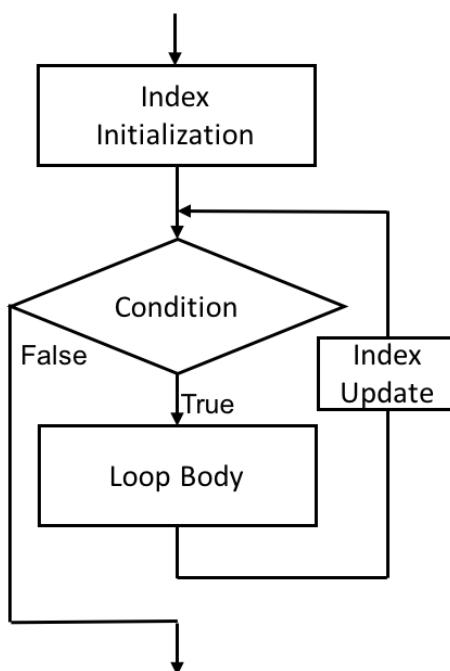


Figure 4.9: โฟลว์ชาร์ตการทำงานของโครงสร้างการเขียนลูป While

```
i=1;
x=0;
while (i<=10) {
```

```

x+=i; /* Body */
i++; /* Index Update */
}

```

ตัวอย่างนี้ ทำงานเหมือนกับประโยค FOR ทุกประการ ความแตกต่างคือ การตั้งค่าตัวแปรเริ่มต้น การตรวจสอบเงื่อนไขในวงเล็บของประโยค WHILE และการ Update ตัวแปร i ภายในลูป ดังนั้น ตัวอย่างการเขียนโปรแกรม ในตารางที่ 4.7 จึงมีความใกล้เคียงกันมากกับประโยค FOR

Table 4.7: ตัวอย่างโปรแกรมตามประโยควนรอบ While

#	เลขบล	คำสั่ง	คอมเมนท์
1	...		@ Initialize R0=0
2	...		@ Initialize R1=1
3	while:	CMP R1, #10	@ Compute R1-10
4		BGT end	@ if greater than goto end
5		SUB R0, R0, R1	@ else subtract R1 from R2
6		ADD R1, R1, #1	@ Increment R1 by 1
7		B while	@ Branch back to Label while
8	end:	...	@ End of while loop
9	...		@

4.7.5 การวนรอบชนิด DO-WHILE

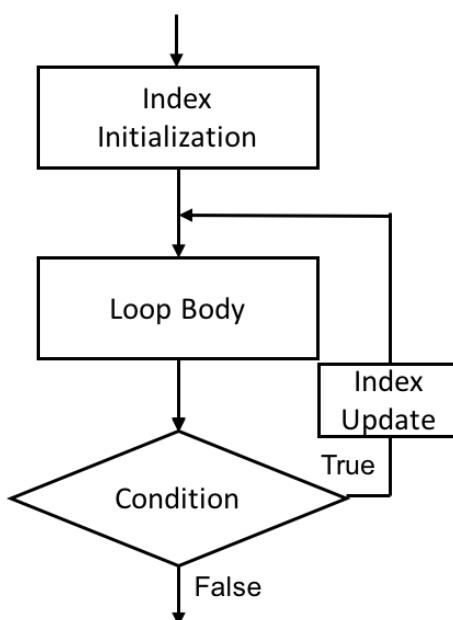


Figure 4.10: โฟล์ชาร์ตการทำงานของโครงสร้างการเขียนลูป Do While

การวนรอบอีกชนิดคือ การวนรอบ DO-WHILE มีโครงสร้างการทำงานตามรูปที่ 4.10 การวนรอบ DO-WHILE คล้ายกับการวนรอบ WHILE คือ ซึ่งมีจุดปฏิบัติตามประโยคคำสั่งที่โปรแกรมเมอร์ต้องการอย่าง

น้อย 1 ครั้งก่อน หลังจากนั้น ซีพียูจะตรวจสอบเงื่อนไขของประโภค WHILE ว่าผลลัพธ์ที่ได้จะเป็นจริงหรือเท็จ หากจริง ซีพียูจะปฏิบัติตามประโภคอีกรอบ และวิ่งตรวจสอบเงื่อนไข หากเท็จ ซีพียูจะทำการสั่งอันๆ ต่อไป

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

ตัวอย่างการเขียนโปรแกรม ในตารางที่ 4.8

Table 4.8: ตัวอย่างโปรแกรมตามประโภควนรอบ Do-While

#	เลขบล.	คำสั่ง	คอมเมนท์
1		...	@ Initialize R0 = 0
2		...	@ Initialize R1 = 0
3	do:	SUB R0, R0, R1	@ else subtract R1 from R2
4		ADD R1, R1, #1	@ R1 = R1+1
5		CMP R1, #10	@ Compute R1-10
6		BLE do	@ if less than or equal goto do
7	end:	...	@ End of do-while loop
8		...	@

4.7.6 การวนรอบชนิด FOR จำนวน 2 ชั้น

ไฟล์ชาร์ตนี้เป็นการวนรอบ 2 ชั้น ในรูปที่ 4.11 มีการทำงานเบื้องต้นดังนี้

1. การตั้งค่าเริ่มต้น (Initialization) ตัวนับลูปที่ 1 (Loop Counter 1) และ ตัวนับลูปที่ 2 (Loop Counter 2)
2. หลังจากนั้น โปรแกรมจะตรวจสอบเงื่อนไข Condition_1 ของลูปชั้นนอก
3. หากเป็นจริง ตรวจสอบเงื่อนไข Condition_2 ของลูปชั้นใน
4. หากเป็นจริง (True) โปรแกรมจะทำงานตาม Statements หรือคำสั่งจำนวนหนึ่ง
5. เมื่อแล้วเสร็จ โปรแกรมจะเพิ่ม (Increment) หรือ ลด (Decrement) ตัวนับลูปที่ 2 (Loop Counter 2)
6. จนเงื่อนไข Condition_2 ของลูปชั้นใน เป็นเท็จ (False)

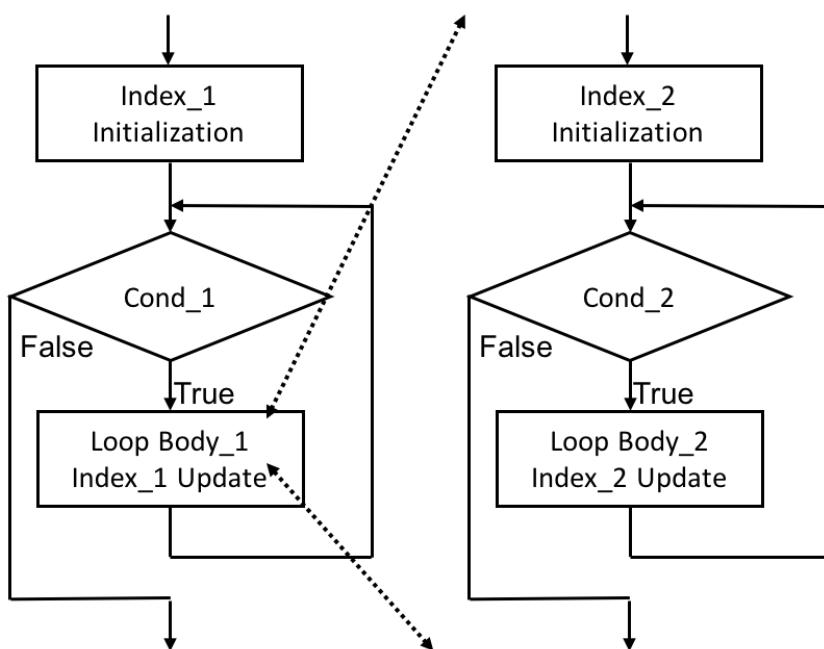


Figure 4.11: ໂຟລ່ວໜ້າຮັດການທຳການຂອງກາຣຽນຮົບ For 2 ຊັ້ນ

7. ໂປຣແກຣມຈະເພີ່ມ (Increment) ອີ່ວື ລົດ (Decrement)
8. ຕັວນັບລູບທີ 1 (Loop Counter 1) ກລັບໄປປະຫຼິດທີ x
9. ແຕ່ໜັກເຈື່ອນໄຂເປັນເທິງ (False) ລູບຊັ້ນນອກຈະສິນສຸດການທຳການ

4.8 การเรียกใช้ฟังค์ชัน (Function Call)

4.8.1 การเรียกใช้ฟังค์ชันในภาษา C/C++

ตัวอย่างการเขียนโปรแกรมโดยการเรียกใช้ฟังค์ชันในภาษา C/C++

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

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

โฟล์ชาร์ตการทำงานของการเรียกใช้ function ชื่อ `sum(x,y)` ในรูปที่ 4.12 กลไกนี้จะเกิดขึ้นเมื่อมีอน กับ การทำงานของฟังค์ชันในโปรแกรมภาษา C/C++ ซึ่งสามารถเรียกใช้ช้าแล้วช้าอีกได้ มีการทำงานเป็น โมดูล สามารถนำฟังค์ชันไปเผยแพร่เป็นฟังค์ชันมาตรฐานให้กับ ยกตัวอย่าง เช่น ฟังค์ชัน `printf` ในไฟล์ `stdio.h` ซึ่งย่อมาจากชื่อ Standard I/O ในรูปที่ 3.16 ฟังค์ชันมาตรฐานเหล่านี้ จะต้องนำมายิงค์ตามหลัก การในรูปที่ 3.18 ซึ่งเรียกว่า ไลบรารี (*.lib) หรือนามสกุลอื่นๆ ที่ใกล้เคียง ส่วนฟังค์ชันที่โปรแกรมเมอร์ พัฒนาเอง จะอยู่ในรูปของไฟล์ Object (*.o) หรือนามสกุลอื่นๆ ที่ใกล้เคียง

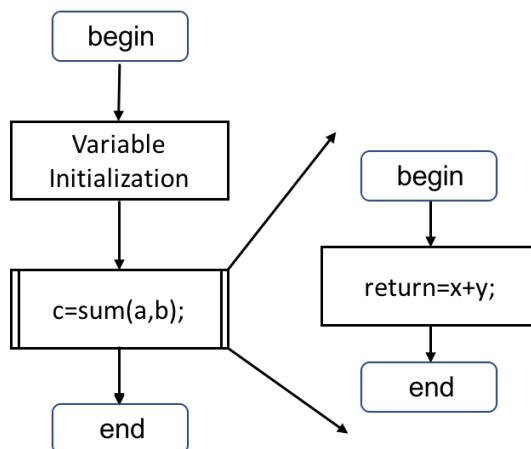


Figure 4.12: โฟล์ชาร์ตการทำงานของการเรียกใช้ฟังค์ชัน `sum(x,y)`

4.8.2 การเรียกใช้ฟังค์ชันในภาษาแอสเซมบลี

ภาษาแอสเซมบลีต่างๆ มีศักยภาพที่ใกล้เคียงกับภาษาสูง เช่น C/C++ ที่โปรแกรมเมอร์และคอมไพร์เตอร์สามารถสร้างฟังค์ชันของตนเองได้ คำสั่งภาษาแอสเซมบลีของ ARM ที่รองรับการเรียกใช้ฟังค์ชัน คือ BL function_name โดยคำว่า **function_name** คือ label ทำหน้าที่เป็นชื่อฟังค์ชันที่ต้องการเรียกใช้ BL ย่อมาจากคำว่า Branch and Link การเรียกกลับไปยังคำสั่งถัดไปเมื่อสิ้นสุดการเรียกใช้ฟังค์ชัน และคำสั่ง BX LR จะเป็นการเรียกกลับมายังตำแหน่ง address ที่รีจิสเตอร์ LR เก็บค่าไว้ ค่านี้จะเก็บไว้ก่อนซึ่งปัจจุบันจะทำงานคำสั่งแรกของฟังค์ชัน

รูปแบบ ความหมาย

BL label กระโดดไปยัง label ซึ่งเป็นชื่อฟังค์ชัน label โดยไม่มีเงื่อนไข

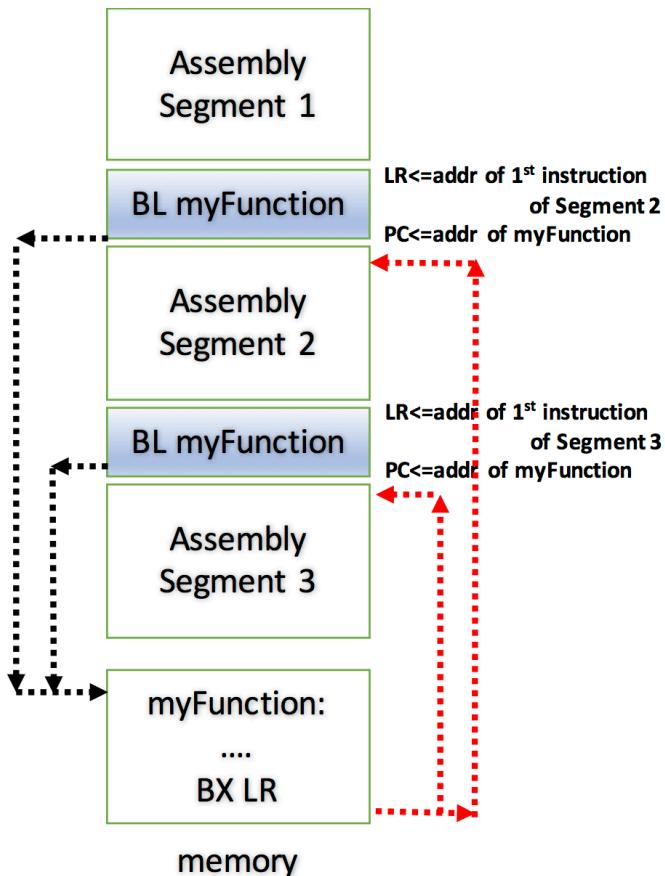
BX LR กระโดดกลับไปทำงานคำสั่งที่อยู่ถัดจากคำสั่ง BL label ก่อนหน้า โดยไม่มีเงื่อนไข

Table 4.9: ตัวอย่างโปรแกรมเรียกใช้ฟังค์ชันด้วยคำสั่ง BL และ BX

#	เลbel	คำสั่ง	คอมเมนท์
1	main:	...	@ Initialize R4 (a)
2		...	@ Initialize R5 (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		...	@
7	sum:	ADD R0, R0, R1	@ entry point of function sum
8		BX LR	@ Return the result in R0

การทำงานของโปรแกรมจะย้ายจากคำสั่ง BL myFunction ไปยังคำสั่งแรกสุดของ คือ คำสั่งที่มีเลเบล myFunction ซึ่ง เมื่อทำงานฟังค์ชันนี้แล้วเสร็จ คำสั่งสุดท้าย คือ คำสั่ง BX LR จะย้ายการทำงานของโปรแกรมกลับไปยัง คำสั่งถัดไปต่อจากคำสั่ง BL myFunction ในรูปที่ 4.13

รายละเอียดการพัฒนา สามารถอ่านเพิ่มเติมได้ที่ การทดลองที่ 5 การพัฒนาโปรแกรมด้วยภาษา C ในภาคผนวก E และการทดลองที่ 6 การพัฒนาโปรแกรมด้วยภาษาแอสเซมบลี ในภาคผนวก F

Figure 4.13: การทำงานของคำสั่ง `BL myFunction` เมื่อมีการเรียกใช้ 2 ครั้ง

4.9 อุปกรณ์และวิวัฒนาการของชุดคำสั่ง ARM

4.9.1 อุปกรณ์ดิจิทัลที่ใช้ชีพยุค ARM

ตารางที่ 4.10 แสดงให้เห็นภาพรวมว่า ในปี 2010 ARM มีส่วนแบ่งการตลาดทั่วโลก 28% คิดเป็นชิพจำนวน 6,100 ล้านชิพ จากทั้งหมด 22,000 ล้านชิพ ในอุปกรณ์ทั้งหมด 19,000 ล้านตัว ตลาดของ ARM แบ่งเป็นอุปกรณ์ที่เป็นโทรศัพท์เคลื่อนที่ (Mobile) และอื่นๆ (Non-Mobile) ตลาดโทรศัพท์เคลื่อนที่ประกอบด้วยสมาร์ทโฟน (Smart Phone) ราคาสูง โทรศัพท์ใช้งานทั่วไป (Feature Phone) ราคากันกลาง โทรศัพท์พื้นฐาน (Low End Voice) ราคาถูก เครื่องเล่นสื่อพกพา (Portable Media Players) และโมบายล์แอปบนโทรศัพท์เคลื่อนที่ (Mobile Apps) โดยอุปกรณ์ที่ ARM มีส่วนแบ่งการตลาดสูงที่สุด (95%) คือ Low End Voice รองลงมาคือ Smart Phone และ Feature Phone ตามลำดับ โดย ARM คาดว่า ตลาดของสมาร์ทโฟนจะเติบโตและได้รับความนิยมเพิ่มสูงขึ้นเรื่อยๆ รวมถึงตลาดของโมบายล์แอป

ในส่วนของตลาดอื่นๆ (Non-Mobile) ประกอบด้วย อุปกรณ์ที่หลากหลายกว่า โดยอุปกรณ์ที่ ARM มีส่วนแบ่งการตลาดสูงที่สุด (85%) คือ ฮาร์ดดิสก์และโซลิดสเตตดิสก์ (Hard Disk & Solid State Drives) รองลงมา คือ กล้องดิจิทัลและปรินเตอร์ (Digital Camera และ Printer) ตามลำดับ โดย ARM คาดว่า ตลาดของฮาร์ดดิสก์และโซลิดสเตตดิสก์ จะเติบโตและได้รับความนิยมเพิ่มสูงขึ้นเรื่อยๆ โดยเฉพาะโซลิดสเตตดิสก์ นอกจากนี้ ตลาดของทีวีดิจิทัลและกล่อง (Digital TV & Set Top Box) และตลาดของไมโคร

Table 4.10: ส่วนแบ่งการตลาดของบริษัท ARM ในปี ค.ศ. 2010 ที่มา: [zdnet.com](http://www.zdnet.com)

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%

คอนโตรลเลอร์ (Microcontroller) น่าจะมีการขยายตัวในปี 2015 เช่นกัน

4.9.2 วิวัฒนาการของชุดคำสั่ง ARM

ในอดีตที่ผ่านมา ARM ได้พัฒนาภาษาอุ กมาห์ลายเวอร์ชัน โดย ARMv5 และ ARMv6 คือ ภาษาแอสเซมบลีเวอร์ชัน 5 และ 6 ตามลำดับ ARMv7 คือ ภาษาแอสเซมบลีเวอร์ชัน 7 ซึ่งรองรับการทำงานของ ไอโอเอส 32 บิตเท่านั้น แบ่งเป็น v7A สำหรับ ARM Cortex-A, v7M สำหรับ ARM Cortex-M และ v7R สำหรับ ARM Cortex-R ARMv8-A คือ ภาษาแอสเซมบลีเวอร์ชัน 8 Cortex-A ซึ่งเป็นเวอร์ชัน 64 บิต

นอกเหนือจากภาษาแอสเซมบลีเวอร์ชันต่างๆ ที่กล่าวมาข้างต้น จุดขายของ ARM ยังมีภาษาแอสเซมบลีอิกเวอร์ชัน ซึ่ว่า Thumb, Thumb-2, T32 นี่คือ ชุดคำสั่งภาษาแอสเซมบลีของ ARM ที่มีความยาว 16 และ 32 บิต ซึ่งนิยมใช้สำหรับอุปกรณ์ที่มีหน่วยความจำขนาดเล็ก รายละเอียดเพิ่มเติมใน [wikipedia](https://en.wikipedia.org/wiki/Thumb_(instruction_set))

การคำนวนปกติจะกระทำการทากับข้อมูลเชิงเดียวตามที่ได้อธิบายไปแล้ว ซึ่งไม่เหมาะสมกับข้อมูลมัลติมีเดีย ดังนั้น ARM จึงได้ออกแบบชุดคำสั่งภาษาแอสเซมบลีที่ประมวลผลข้อมูลหลายๆ ตัวพร้อมกัน เรียกว่า SIMD, Adv SIMD (Advanced SIMD) หรือ NEON เหมาะสำหรับข้อมูลภาพและเสียง รวมไปถึงเกม ซึ่งเป็นข้อมูลที่เรียงตัวกันจำนวนหนึ่ง เรียกว่า เวคเตอร์ (Vector) รายละเอียดเพิ่มเติมใน [wikipedia](https://en.wikipedia.org/wiki/Advanced SIMD)

จุดเด่นที่น่าสนใจของ ARM เรียกว่า Jazelle คือ การรองรับการประมวลผลคำสั่งแอสเซมบลีของภาษา Java เรียกว่า Java Byteคำสั่ง รายละเอียดเพิ่มเติมใน <https://en.wikipedia.org/wiki/Jazelle> นอกจากนี้จากที่ได้กล่าวมาแล้ว ผู้อ่านสามารถค้นควาระรายละเอียดอื่นๆ เพิ่มเติมได้ที่ [wikipedia](https://en.wikipedia.org/wiki/Java_bytecode)

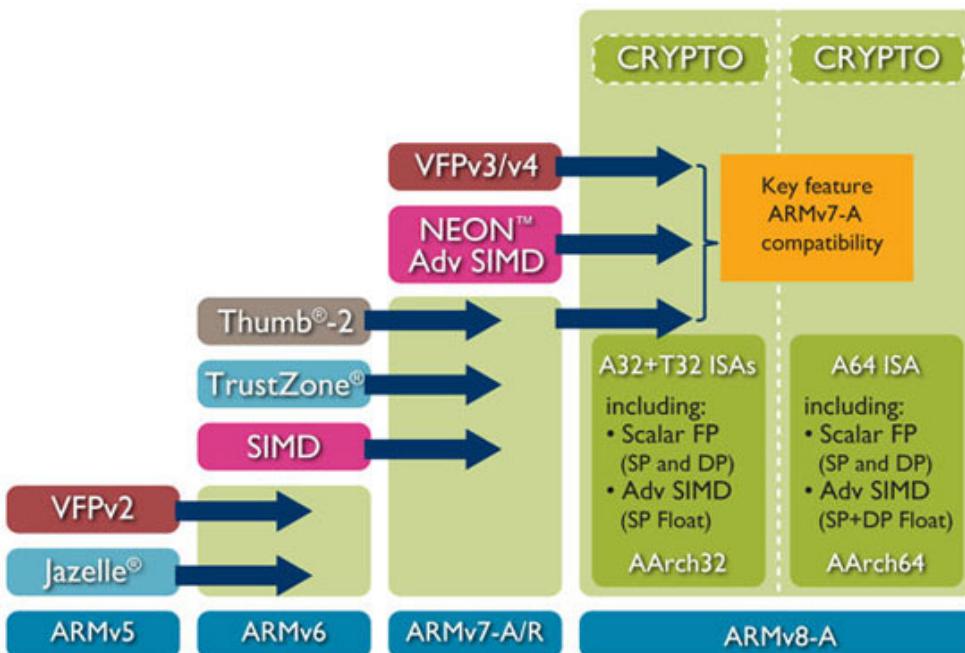


Figure 4.14: การรวมซุดคำสั่งภาษาและเซมบลีของ ARM ตั้งแต่เวอร์ชัน 5 ถึงเวอร์ชัน 8A โดยเวอร์ชันใหม่จะรวมเวอร์ชันเก่าเพื่อรองรับการทำงานซอฟต์แวร์เดิม ที่มา: community.arm.com

4.10 สรุปท้ายบท

คำสั่งภาษาและเซมบลีของ ARM มีลักษณะเด่นเมื่อเปรียบเทียบกับภาษาอื่นๆ เช่น การมีคำสั่งเลื่อนบิตภายในคำสั่งคณิตศาสตร์ การตรวจสอบเงื่อนไขก่อนการปฏิบัติตามคำสั่งนั้น เป็นต้น ทำให้ชิปที่ผลิตตามการออกแบบของ ARM บริโภคพลังงานต่ำ จึงได้รับความนิยมในอุปกรณ์เคลื่อนที่มาเป็นระยะเวลาต่อเนื่อง

ARM ออกแบบคำสั่งภาษาและเซมบลีตามหลักการ RISC (Reduced Instruction Set Computer) และสถาปัตยกรรมอ่าน/เขียน (Load/Store Architecture) ซึ่งแตกต่างจากการทำงานสถาปัตยกรรมชนิดอื่นๆ เช่น

- สถาปัตยกรรมเมมโมรี/เมม莫รี (Memory/Memory Architecture) ในภาษาและเซมบลีของ Intel 80x86 หรือเรียกว่าย่อๆ ว่า x86 ซึ่งบางคำสั่งใช้การอ่านค่าจากหน่วยความจำเพื่อประมวลผล แล้วจึงเขียนผลลัพธ์กลับไปยังหน่วยความจำ
- สถาปัตยกรรมสเต็ค (Stack Architecture) ในภาษา Java ไบท์โค้ด (Java Byte Code) ซึ่งจัดเป็นคำสั่งภาษาและเซมบลีเมื่อแปลจากซอฟต์แวร์ส์โค้ดภาษา Java

4.11 คำถามท้ายบท

- รีจิสเตอร์ PC (Program Counter) ทำหน้าที่อะไร และเกี่ยวข้องกับการรันของโปรแกรมอย่างไร
- คำสั่ง POP สำหรับอ่านค่าข้อมูลออกจากสแต็คເชັກເມນໍທີ່ ทำงานอย่างไร
- คำสั่ง PUSH สำหรับเก็บค่าข้อมูลลงในสแต็คເชັກເມນໍທີ່ ทำงานอย่างไร

4. คำสั่ง POP และ PUSH ทำงานร่วมกับรีจิสเตอร์ Stack Pointer (SP) อย่างไร
5. สเต็คเช็กเม้นท์ มีทิศทางขยายขนาดไปในทิศทางใด เพราะอะไร
6. รีจิสเตอร์ LR (Link Register) ทำหน้าที่อะไร และเกี่ยวข้องกับการรันของโปรแกรมอย่างไร
7. จ�述ิบายว่า BX LR ทำงานอย่างไร เหตุใดจึงต้องเป็นคำสั่งสุดท้ายของฟังก์ชัน
8. จงเขียนฟังก์ชันเพื่อคำนวนหาค่าสัมบูรณ์ของ R0-R1 แล้วรีเทิร์นค่าในค่า R0
9. จงเขียนโปรแกรมลูปวนรอบ 2 ชั้น โดยใช้การวนรอบชนิด
 - While
 - Do-While
10. การทำงานของฟังก์ชันที่เรียกตัวเอง (Recursive Function Call)
 - จงวาดโฟล์ชาร์ตเพื่อแสดงการทำงานกรณีปกติและกรณีหยุดเรียกตัวเอง
 - จงวัดรูปการทำงานประกอบด้วยหน่วยความจำคล้ายกับรูปที่ [4.13](#)

Chapter 5

หน่วยความจำลำดับชั้น (Memory Hierarchy)

หน่วยความจำลำดับชั้น (Memory Hierarchy) อาศัยการทำงานร่วมกันของหน่วยความจำหลายชนิด ต้นทุนที่หลากหลาย และหลายความจุเข้าด้วยกัน เพื่อผ่อนจุดเด่นของหน่วยความจำแต่ละชนิดเข้าด้วยกัน และช่วยประหยัดต้นทุนโดยรวมของระบบ บทนี้มีวัตถุประสงค์ดังนี้

- เพื่อให้เข้าใจถึงลำดับชั้นและการทำงานร่วมกันระหว่างหน่วยความจำชนิดต่างๆ ได้แก่ รีจิสเตอร์ แคชลำดับชั้นต่างๆ หน่วยความจำหลัก และอุปกรณ์เก็บรักษาข้อมูล
- เพื่อให้เข้าใจถึงการทำงานร่วมกันของหน่วยความจำเสมือนชนิดเพจ (Paging Virtual Memory) ของหน่วยประมวลผลและระบบปฏิบัติการในทางทฤษฎี
- เพื่อให้เข้าใจถึงการทำงานร่วมกันของหน่วยความจำเสมือนของบอร์ด Pi3 และระบบลีนักซ์ ขนาด 32 บิตโดยละเอียด
- เพื่อให้เข้าใจถึงการทำงานของแคชชนิด Direct Map และ Set Associative ซึ่งนิยมใช้ออกแบบ แคชในชิปประมวลผล
- เพื่อให้เข้าใจถึงความแตกต่างระหว่างหน่วยความจำชนิดต่างๆ ในด้านกายภาพ การทำงาน และประสิทธิภาพ

บทนี้จะอธิบายลำดับชั้นของหน่วยความจำ (Memory Hierarchy) ว่าเหตุใดคอมพิวเตอร์ จึงต้องประกอบด้วยหน่วยความจำหลายชนิด ซึ่งต่อเนื่องจากหัวข้อที่ [3.1.2](#) เนื้อหาในบทนี้อาศัยการทดลองที่ 4 ภาคผนวก D การใช้งานระบบปฏิบัติการยูนิกซ์ ซึ่งจะทำให้ผู้อ่านเข้าใจหลักการและรายละเอียดมากขึ้น

5.1 โครงสร้างของลำดับชั้นหน่วยความจำของบอร์ด Pi3

หน่วยความจำมีหลายชนิดตามหลักการออกแบบ ความซับซ้อน/เทคโนโลยีการผลิต ความจุของหน่วยความจำแต่ละชนิดมีความหลากหลาย ความเร็วหรือสมรรถนะในการอ่านหรือเขียนข้อมูล และ ต้นทุนต่อความจุหนึ่งบิต ทำให้การจัดวางมีผลต่อประสิทธิภาพโดยรวม หน่วยความจำบางชนิดสามารถบรรจุในชิป (On Chip) ในขณะที่บางชนิดจำเป็นต้องอยู่คู่กับชิปภายนอก (Off Chip) ตำแหน่งนี้อาศัยหน่วยความจำชนิดต่างๆ ของบอร์ด Pi3 ซึ่งใช้ชิป BCM2837 เป็นกรณีศึกษาลำดับชั้นและการจัดวางหน่วยความจำชนิดต่างๆ

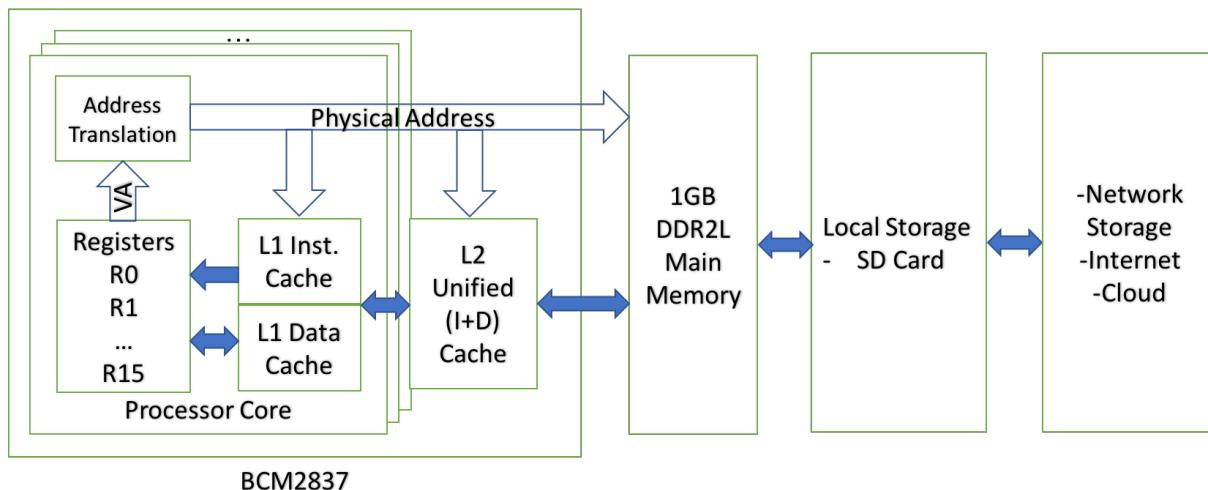


Figure 5.1: ลำดับชั้นของหน่วยความจำชนิดต่างๆ สำหรับชิป BCM2837, (VA: Virtual Address)

บอร์ด Pi3 ประกอบด้วย หน่วยความจำบางส่วนอยู่ในชิปเดียวกับไมโครโปรเซสเซอร์ (On Chip) และ อยู่นอกชิป (Off Chip) ในรูปที่ 5.1 หน่วยความจำเหล่านี้อยู่ในลำดับชั้น (Hierarchy) ต่างๆ ทั้งใกล้และ ไกลจาก ALU (Arithmetic Logic Unit) ประกอบด้วย รูปที่ 4.2 เพิ่มอุปกรณ์เก็บรักษาข้อมูล และเครื่อง ข่าย

- **รีจิสเตอร์ (Register)** ในสถาปัตยกรรมคำสั่ง ARMv7 ซึ่งเป็นเวอร์ชัน 32 บิต ประกอบด้วยรีจิสเตอร์ R0, R1, ... R15 ซึ่งสร้างมาจากการหน่วยความจำชนิดสแตติกแรม (Static RAM: SRAM) สามารถเข้าถึงข้อมูลได้ภายในเวลาสั้นๆ เพียง 0.5-1 นาโนวินาที (Nano Second)
- **แคชลำดับที่ 1 (Level 1)** ซึ่งแบ่งเป็นแคชคำสั่ง หรือ Instruction Cache และแคชข้อมูล หรือ Data Cache ซึ่งใช้เทคโนโลยีสแตติกแรม มีขนาดประมาณ 16-64 กิโลไบต์ (KiloByte: KB) แต่ละ ชิปปีกอร์ของ ARM Cortex A53 จะมีแคชทั้งสองจะอยู่ด้วยกัน
- **แคชลำดับถัดไป** ซึ่งใช้เทคโนโลยีสแตติกแรมเข่นกัน โดยมีขนาด 64 กิโลไบต์ขึ้นไป (KiloByte: KB) จนถึงหลักเมกะไบต์ (MegaByte: MB) แคชต่างมีการประสานงานร่วมกัน เพื่อให้แคชลำดับที่ 2 ทำหน้าที่คล้ายหน่วยความจำขนาดเล็กของแคชลำดับที่ 1 และในไมโครโปรเซสเซอร์จากผู้ผลิตบางราย ได้ออกแบบให้มีแคชลำดับที่ 3 ที่มีความจุใหญ่ขึ้น ทำหน้าที่คล้ายหน่วยความจำขนาดเล็กของ แคชลำดับที่ 2 ซึ่งต้นทุนในการผลิตชิปเหล่านี้แปรผันตรงกับความจุของแคชเช่นกัน

- หน่วยความจำหลัก (Main Memory) หรือบางตำราใช้คำว่า Primary Memory หรือ หน่วยความจำปั๊มนิ่ม ซึ่งนิยมใช้หน่วยความจำชนิดไดนามิกแรม (Dynamic RAM: DRAM) มีขนาด 512 เมกะไบต์ขึ้นไป จนถึงจิกะไบต์ (GigaByte: GB) รีจิสเตอร์ แคช และหน่วยความจำหลักไม่สามารถเก็บรักษาข้อมูลได้ หากผู้ใช้ปิดเครื่อง ไฟฟ้าขัดข้อง ไฟฟ้ากระชากหรือแรงดันตกช่วง ขณะ หรือแหล่งจ่ายไฟหมดพลังงาน เช่นใน คอมพิวเตอร์พกพาซึ่งต้องอาศัยพลังงานจากแบตเตอรี่ ทำให้เครื่องคอมพิวเตอร์จะต้องมีอุปกรณ์เก็บรักษาข้อมูลในลำดับชั้นลัดไป ในทำนองเดียวกัน เครื่องคอมพิวเตอร์ที่ต้องเปิดทำงานนานๆ ควรมีแหล่งจ่ายไฟสำรองหรือ UPS (Uninterrupted Power Supply) เช่นเดียวกัน

ในด้านของความจุ หน่วยความจำหลัก มีความจุขนาดใหญ่แต่ใช้เวลานานกว่า ในขณะที่แคชมีความจุรองลงมา และรีจิสเตอร์มีความจุน้อยที่สุดแต่มีความเร็วสูงสุด การประมวลผลทางคณิตศาสตร์ และตรรกศาสตร์ข้อมูลเหล่านี้ จะต้องอาศัยขบวนการอ่าน (Load) ข้อมูลจากแคชหรือหน่วยความจำพกเก็บในรีจิสเตอร์ก่อน แล้วจึงนำค่าในรีจิสเตอร์ไปประมวลผลแล้วพักเก็บเพื่อประมวลผลต่อไป เมื่อคำนวณแล้วเสร็จโปรแกรมจึงทำการเขียน (Store) ค่าไปเก็บในหน่วยความจำหลัก

- อุปกรณ์เก็บรักษาข้อมูล (Storage) หรือเรียกว่า หน่วยความจำที่ติดภายนอก (Secondary Memory) ซึ่งมีทางเลือกจากหลายเทคโนโลยี เพื่อเก็บรักษาข้อมูลไม่ให้สูญหาย ถึงแม้ผู้ใช้จะปิดเครื่องหรือแหล่งจ่ายไฟหมดพลังงาน เช่นใน คอมพิวเตอร์พกพาซึ่งต้องอาศัยพลังงานจากแบตเตอรี่ ชนิดของอุปกรณ์เก็บรักษาข้อมูล โดยจะกล่าวรายละเอียดเพิ่มเติมในบทที่ 7
 - หน่วยความจำ SD (Secure Digital) ซึ่งสร้างจากเทคโนโลยีแฟลช (Flash) เช่นเดียวกับโซลิดสเตทดิสก์ และกล่าวถึงในหัวข้อที่ 3.1.4 ทำให้มีความจุมาก ระดับ 16 จิกะไบต์ขึ้นไป หน่วยความจำแฟลชความเร็วสูงกว่าเมื่อเทียบกับฮาร์ดดิสก์ หน่วยความจำ SD สามารถพกพาได้สะดวก เพราะขนาดเล็กและน้ำหนักเบา จึงนิยมใช้บันทึกข้อมูลในกล้องดิจิทัล โทรศัพท์スマาร์ทโฟน และอื่นๆ
 - โซลิดสเตทดิสก์ (Solid State Disk: SSD) ซึ่งใช้เทคโนโลยีหน่วยความจำแฟลช (Flash Memory) มีขนาด 128 จิกะไบต์ขึ้นไปจนถึงหลายเทราไบต์ และคาดว่าจะทดแทนฮาร์ดดิสก์ในปัจจุบันและอนาคต
 - ฮาร์ดดิสก์ (Hard Disk) ซึ่งใช้เทคโนโลยีหน่วยความจำแผ่นแม่เหล็ก (Magnetic Disc) หมุนด้วยความเร็วสูงคงที่ประมาณ 5,400-10,000 รอบต่อนาที มีขนาดหลายจิกะไบต์จนถึงหลายเทอร่าไบต์ (TeraByte: TB)
 - อุปกรณ์เก็บรักษาข้อมูลผ่านเครือข่าย (Network Storage) ซึ่งมีรูปแบบต่างๆ เช่น Network Attached Storage (NAS), Storage Area Network (SAN) Cloud Storage เป็นต้น

กล่าวโดยสรุปคือ หน่วยความจำในรูปของรีจิสเตอร์อยู่ใกล้กับ ALU ใช้ เวลาเข้าถึง (Access Time) สั้นกว่าเสมอ แต่จะมีความจุน้อย เนื่องจากความซับซ้อนและต้นทุนต่อความจุที่สูงกว่า การจัดแบ่งระดับชั้นต่างๆ ของหน่วยความจำอาศัยหลักการใช้งานซ้ำในแกนเวลา (Temporal Locality) และในพื้นที่ใกล้

เคียง (Spatial Locality) สามารถทำให้ผู้ใช้มองเห็นหน่วยความจำขนาดใหญ่ที่มีต้นทุนต่ำ แต่มีความเร็วและการตอบสนองที่รวดเร็วพอประมาณ

5.2 หน่วยความจำเสมือน (Virtual Memory)

ระบบปฏิบัติการสามารถใช้ประโยชน์จากซีพียูที่รองรับการสร้างหน่วยความจำเสมือนได้ โดยให้ฮาร์ดแวร์ในซีพียูช่วยทำหน้าที่แปลงแอดเดรสเสมือน (Virtual Address) ให้เป็นแอดเดรสกายภาพ (Physical Address) หน่วยความจำเสมือนมีหน้าที่และลักษณะสำคัญดังนี้

- รองรับความต้องการความจุของหน่วยความจำหลักที่เพิ่มสูงขึ้น เนื่องจากระบบปฏิบัติการ 64 บิต ที่สามารถอ้างอิงหน่วยความจำหลักได้เกิน 4 GB
- บริหารจัดการความเร็วที่แตกต่างระหว่างหน่วยความจำหลักหรือ DRAM และหน่วยเก็บรักษาข้อมูลซึ่งเป็นฮาร์ดดิสก์ขนาดหมุนตั้งแต่ในอดีต ถึงแม้หน่วยความจำแฟลชจะมีความเร็วสูงขึ้นและแพร่หลายมากขึ้นในปัจจุบันและต่อไปในอนาคต

การทำงานของแต่ละโปรแกรม ให้มีหน่วยความจำเสมือนของตนเอง ดังรูปที่ 3.12 ข้อดี คือ โปรแกรมของผู้ใช้สามารถมีขนาดใหญ่กว่าหน่วยความจำจริงได้ทำให้ผู้เขียนโปรแกรมทำการเขียน โปรแกรมได้อย่างอิสระมากขึ้น ไม่ต้องกังวลต่อขนาดของหน่วยความจำ การเชื่อมต่อกับหน่วยความจำภายในภาพ รูปที่ 5.2 หน่วยความจำเสมือนจะถูกแบ่งเป็นเพจ หรือ พื้นที่ขนาด 1 ถึง 8 กิโลไบต์ ขึ้นอยู่กับฮาร์ดแวร์และระบบปฏิบัติการที่จะตกลงกัน พื้นที่เหล่านี้จะแบ่งเป็นส่วนที่เก็บคำสั่ง (Text Segment) ส่วนเก็บตัวแปร (Data Segment) ส่วนสำหรับสร้างสเต็ค (Stack) และ 希พ (Heap) ในพื้นที่ส่วนที่เหลือเรียกว่า ສเต็คเซกเม้นท์ (Stack Segment) 希พ คือ พื้นที่สำหรับจองพื้นที่ตัวแปรในขณะที่โปรแกรมกำลังทำงาน เช่น ตัวแปรโครงสร้างข้อมูล (Data Structure) ชนิดลิงค์ลิสต์ (Linked List), ทรี (Tree), กราฟ (Graph) และอื่นๆ

เมื่อระบบปฏิบัติการได้จดพื้นที่ขนาดเท่ากับหนึ่งเพจในหน่วยความจำภายในภาพ เรียกว่า เพจเฟรม หรือ เフレม (Frame) และทำการแมป (Map) หรือ แปลงเลขเพจเสมือน (Virtual Page Number) ให้ตรงกับเลขเฟรมภายในภาพ (Physical Frame Number) สังเกตได้จากเฟรมที่เป็นสีม่วงซึ่งมาจากโปรแกรมเดียวกันนี้ ตารางการแมปนี้ ระบบปฏิบัติการเรียกว่า ตารางเพจ (Page Table) ตารางเพจ คือ ตารางสำหรับจดบันทึกความสัมพันธ์ระหว่างเลขเพจเสมือน กับ เลขเฟรม ตั้งอยู่บริเวณส่วนของ Kernel Space ของหน่วยความจำภายในภาพ

ตามรูปที่ 5.2 ช่องที่เป็นสีชมพูในหน่วยความจำภายในภาพ คือ เฟรมที่ว่างอยู่ หรือ ถูกจับจองโดยโปรแกรมอื่นๆ ด้วย ซึ่งระบบปฏิบัติการที่รองรับการทำงานแบบมัลติโปรแกรมมิ่ง (Multi Programming) ในปัจจุบัน โอลีสเปิดโอกาสให้ โปรแกรมหลายๆ ตัวใช้งานหน่วยความจำภายในภาพได้พร้อมๆ กัน ส่วนเฟรมช่องที่เป็นสีขาว หมายถึง เฟรมที่ยังว่างอยู่ หมายเหตุ เส้นประ คือ รอยต่อระหว่างเพจของหน่วยความจำเสมือน

เนื่องจากหน่วยความจำหลัก เช่น ROM/RAM ยังมีต้นทุน (ต่อความจุหนึ่งหน่วย) สูงกว่า เมื่อเทียบกับอุปกรณ์เก็บรักษาข้อมูล ดังนั้น โอลีสจำเป็นต้องใช้หน่วยความจำหลักทำหน้าที่เป็นแคชของหน่วยเก็บรักษาข้อมูล การอ่านหรือเขียนหน่วยเก็บรักษาข้อมูล จะเป็นต้องอ่านหรือเขียนครั้งละมากๆ เนื่องจากเวลา

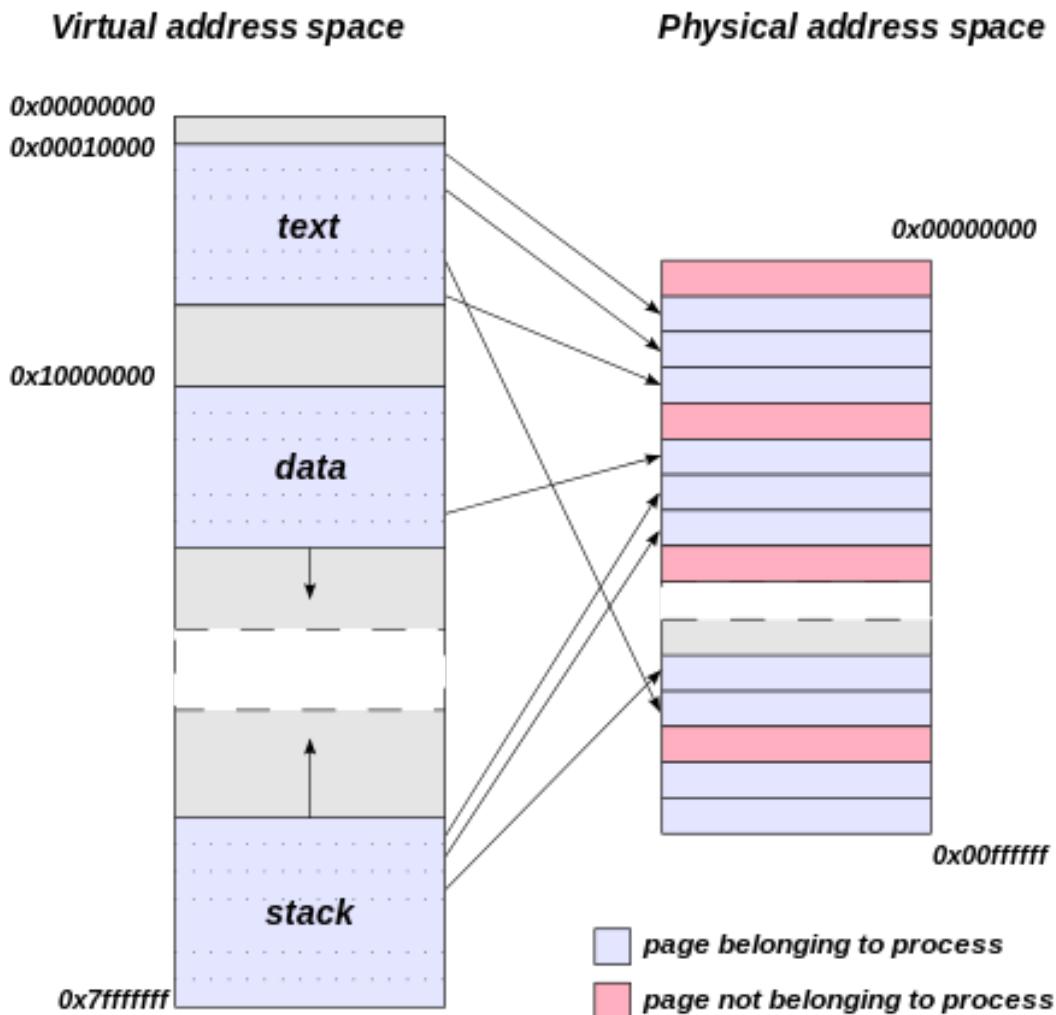


Figure 5.2: การແມ່ນ (Map) ແອດເຊີສເສມືອນ (Virtual Address) ພາຍໃນແອດເຊີສກາຍກາພ (Physical Address) ພາຍໃນ 64MB ຕາມຫລັກກາຮ່ານ່ວຍຄວາມຈຳເສມືອນໜິດເພຈ (Paging Virtual Memory ມາຍເຫຼຸດ ເສັ້ນປະ ອື່ບໍ່ຮອຍຕ່ອງຮ່ວງພົງຂອງໜ່ວຍຄວາມຈຳເສມືອນ) ທີ່ມາ: wikiwand.com

ໜ່ວງ (Latency) ທີ່ຕ້ອງຮອເພື່ອ ໂອນຍ້າຍ (ອ່ານຫຼືເຂີຍ) ຊົ້ມຸລ໌ຫຼືຄຳສັ່ງ ຮະຫວ່າງໜ່ວຍຄວາມຈຳຫລັກ ແລະ ໜ່ວຍເກີບຮັກຂາຂໍ້ມູນ ຮາຍລະເອີດສາມາດອ່ານເພີ່ມເຕີມໃນບທທີ່ 7 ດັ່ງນັ້ນ ການແບ່ງໜ່ວຍຄວາມຈຳເສມືອນ ແລະ ໜ່ວຍຄວາມຈຳຫລັກອອກເປັນເພິ່ງຈຶ່ງເປັນເຮື່ອງທີ່ເໝາະສົມ ແລະ ຈ່າຍຕ່ອງການບວງຈັດການ ຍັກຕ້ວອຍ່າງເຊັ່ນ ຮະບບລື້ນຸກໆ ແລະ ຮະບບ Windows ກຳທັດໃຫ້ແຕ່ລະເພິ່ມຂາດ 4 ກິໂລໄບທ໌

5.2.1 ໜ່ວຍຄວາມຈຳເສມືອນຂອງ Raspberry Pi3

ການແປລງແອດເຊີສເສມືອນເປັນແອດເຊີສກາຍກາພຂອງຊີພຕຣະກຸລ BCM2835-BCM2837 ຈະເໜືອນກັນ ໃນ ຮູບທີ່ 5.5 ໂດຍມີ MMU (Memory Management Unit) ທຳຫັນທີ່ແປລ (Address Translation) ແອດເຊີສເສມືອນເປັນແອດເຊີສກາຍກາພ ຈຶ່ງໄດ້ອີນບາຍຮາຍລະເອີດໄປແລ້ວໃນຫ຾້ອກກ່ອນໜັ້ນ ຮູບທີ່ 5.3 ແສດການແປລ ແອດເຊີສໂດຍລະເອີດ ດັ່ງນີ້ ພື້ນທີ່ຂອງໜ່ວຍຄວາມຈຳເສມືອນຂອງແຕ່ລະໂປຣແກຣມຫຼືໂພຣເໜສຖາງດ້ານຂວາ ມີ ພື້ນທີ່ທັງໝົດ 4 ຈິກະໄບທ໌ ເນື່ອຈາກບ້ອດ Pi1 ຄື Pi3 ໃຊ້ຮະບບປົງບັດກາລື້ນຸກໆ ພາຍໃນ 32 ບິທດ້ວຍກັນ ພື້ນທີ່ທັງໝົດແປ່ງເປັນ 1 ຈິກະໄບທ໌ສໍາຫຼັບເກີບຄຳສັ່ງແລະ ຊົ້ມຸລ໌ຂອງ ເຄື່ອງເນີລ (Kernel) ໃນການບວງຈັດການເຄື່ອງ

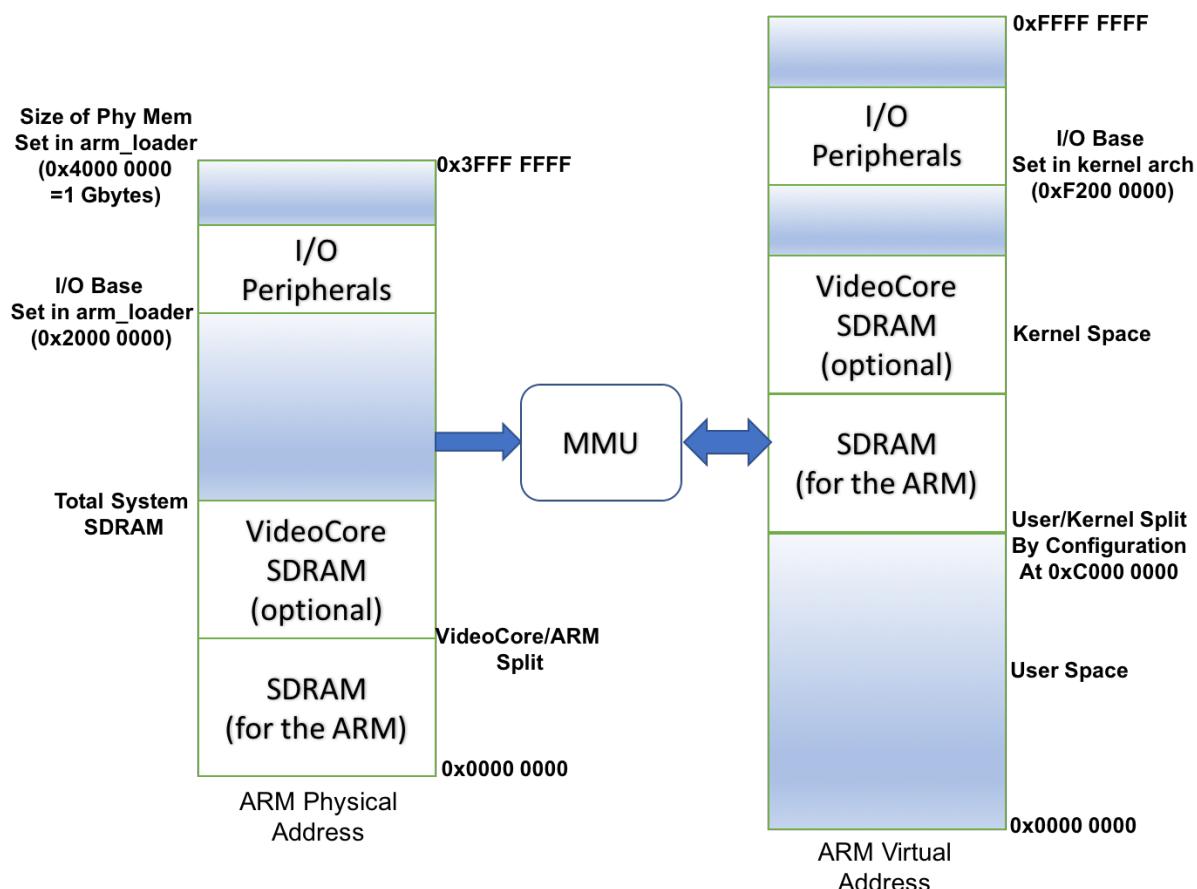


Figure 5.3: โครงสร้างของหน่วยความจำเสมือน (Virtual Memory) ของบอร์ด Pi3 โนมเดล B ซึ่งใช้ชิปตระกูล BCM283x, x=5, 6, 7 หมายเหตุ ขนาดของรูปไม่เป็นไปตามพื้นที่ความเป็นจริง, MMU: Memory Management Unit ที่มา: [Broadcom \(2012\)](#)

เรียกว่า **Kernel Space** โดยแอดเดรสเสมือนของ Kernel เริ่มต้นที่หมายเลข 0xC000_0000 มีพื้นที่ขนาด 1 GB โดยประมาณ โดยพื้นที่ 1 จิกะไบท์นี้ proc ทุกตัวจะเห็นเป็นพื้นที่เดียวกัน และ 3 จิกะไบท์สำหรับเก็บคำสั่งและข้อมูลของprocแต่ละตัว เรียกว่า **User Space** ซึ่งคล้ายกับรูปที่ 3.12 แอดเดรสเสมือนของ User เริ่มต้นจากหมายเลข 0x0000_0000 จนถึง 0xBFFF_FFFF. ขนาดเท่ากับ 3 GB

พื้นที่หน่วยความจำเสมือนของ Kernel แบ่งเป็น หน่วยความจำ SDRAM ของ ARM และของ GPU หรือ VideoCore จะถูกกำหนดโดยไฟล์ start.elf ในบูตพาร์ติชันซึ่งฟอร์แมตด้วย FAT32 โดยหน่วยความจำที่กำหนดให้ GPU ขึ้นต่ำที่สุด ขนาด 32 เมกะไบท์ แต่ความละเอียด 1080p30 ต้องการหน่วยความจำมาก 64 เมกะไบท์ สำหรับทำหน้าที่เป็นบัฟเฟอร์สำหรับแสดงผล

แอดเดรสภายในทางด้านซ้ายของบอร์ด Pi3 เริ่มต้นที่หมายเลข 0x0000_0000 - 0x3FFF_FFFF เท่ากับ 1 จิกะไบท์ โดยแบ่งออกเป็น

- พื้นที่สำหรับการแสดงผลขั้นต่ำเท่ากับ 32 MB โดยสามารถตรวจสอบขนาดที่แท้จริงได้ในการทดลองที่ 4 ภาคผนวก D
- พื้นที่สำหรับเชื่อมต่อกับอุปกรณ์อินพุตเอาท์พุต ซึ่งไม่ผ่านแคช (Uncached) เริ่มต้นที่แอดเดรส 0x2000_0000 ตามหลักการ Memory Mapped I/O ซึ่งจะกล่าวต่อไปในบทที่ 2

- เคอร์แนลจะบริหารจัดการโดยแบ่งให้โปรแกรมและโคร์แนลใช้งานพื้นที่อื่นๆ ที่เหลือร่วมกัน

ผู้อ่านสามารถค้นควารายละเอียดเพิ่มเติมได้ที่ลิงค์ต่อไปนี้ <https://github.com/raspberrypi>

5.2.2 หน่วยความจำเสมือนชนิดเพจของ ARM Cortex A7

กรณีศึกษา ชิปปุย ARM Cortex A7 เป็นต้นแบบการพัฒนาของ Cortex A53 ซึ่งทั้งคู่จัดเป็นชิพระดับต้น (Entry Level) สำหรับโทรศัพท์เคลื่อนที่สมาร์ทโฟน [ลิงค์สำหรับที่มาของรูปที่ 5.4](#) ประกอบด้วยการทำงานร่วมกันของหน่วยความจำหลายชนิด รูปที่ 5.4 ที่มา: การแปลง Virtual Address เป็น Physical Address จะต้องอาศัยตารางเพจ (Page Table) ในการเก็บหมายเลขเพจเสมือน (Virtual Page Number) และหมายเลขเพจกายภาพ (Physical Page Number) โดยมี TLB ทำหน้าที่เป็นแคชให้กับหน่วยความจำบริเวณตารางเพจ (Page Table) ที่แอลบี (TLB: Translation Look Aside Buffer) ซึ่งแบ่งเป็น I (Instruction) TLB และ D (Data) TLB ซึ่งใช้เทคโนโลยีสแตติคแรมเช่นกัน

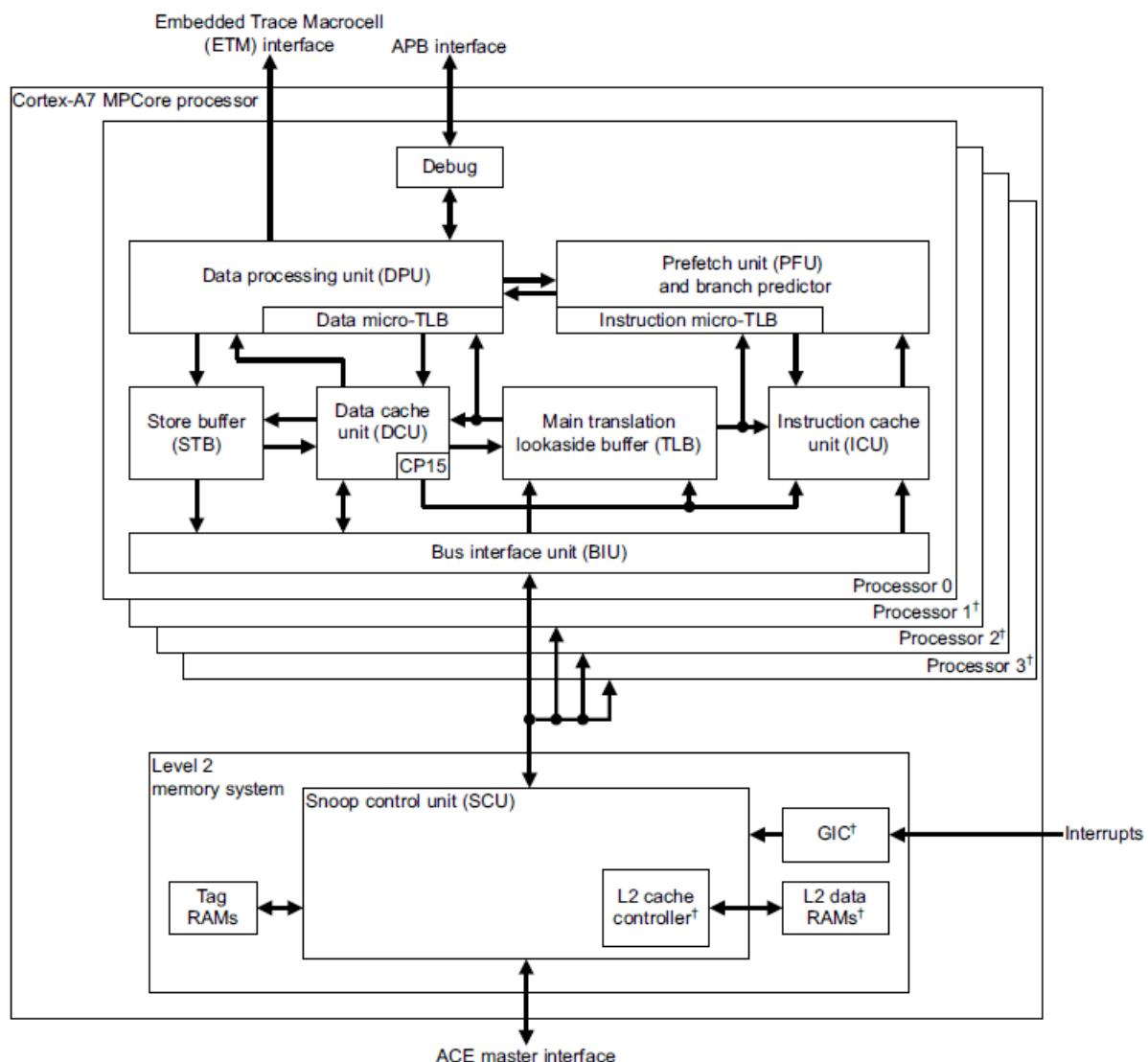


Figure 5.4: โครงสร้างภายในของชิป ARM Cortex A7 ประกอบด้วย TLB และ แคช หลักระดับเพื่อรับการทำงานของหน่วยความจำเสมือน (Virtual Memory)

Chapter 5. หน่วยความจำลำดับชั้น (Memory Hierarchy)

การทำงานของระบบหน่วยความจำโดยรวมซึ่งประกอบด้วย Instruction micro-TLB, Data micro-TLB, Main TLB, ลำดับที่ 1 Instruction Cache Unit (ICU), ลำดับที่ 1 DCache, ลำดับที่ 2 Cache, ลำดับที่ 3 Cache และหน่วยความจำหลัก DRAM ลำดับการทำงานของรูปที่ 5.4 เป็นการแสดงลำดับการอ่าน (Fetch) คำสั่งเพื่อนำไป Decode/Execute ในไปเป็นชีวนิรัน ดังนี้

1. นำค่าแอดเดรสเสมือนใน PC (Program Counter) ส่วนที่เป็น Virtual Page# ไป สืบค้นหาค่าเลข เพจพิสิคัล (Physical Page#) ใน Instruction micro-TLB
2. นำค่า Physical Page# จะนำไปตรวจสอบค่า Tag ว่าตรงกันหรือไม่ ถ้าตรงกันคือ TLB ฮิต (Hit) ถ้า ไม่ตรงกัน คือ TLB มิส (Miss) แล้วจึงนำหมายเลข Virtual Page# ไปสืบค้นใน Main TLB ต่อไป หากมิสอีกรอบ จะต้องสืบค้นในตารางเพจเป็นลำดับสุดท้าย
3. นำค่า Virtual Page Offset จาก PC มาเชื่อมกับ Physical Page# เพื่อนำไปสืบค้นคำสั่งใน ICU หากพบว่าตรงกัน เรียกว่า เกิดแคชฮิตที่ลำดับที่ 1 (L1 Cache Hit) แล้วนำคำสั่งส่งกลับไปยังวงจร Fetch ต่อไป หากพบว่าไม่ตรงกันเรียกว่า เกิดแคชมิสที่ลำดับที่ 1 (L1 Cache Miss)
4. นำแอดเดรสภายในภาพไปค้นหาคำสั่งในแคชลำดับที่ 2 หากพบว่าตรงกันเรียกว่า เกิดแคชฮิตที่ลำดับ ที่ 2 (L2 Cache Hit) แล้วนำคำสั่งส่งกลับไปยังวงจร Fetch ต่อไป หากพบว่าไม่ตรงกันเรียกว่า เกิด แคชมิสที่ลำดับที่ 2 (L2 Cache Miss)
5. หากระบบมีแคชลำดับที่ 3 จะนำแอดเดรสภายในภาพไปค้นหาคำสั่งในแคชลำดับที่ 3 หากระบบ ไม่มีแคชลำดับที่ 3 จะนำแอดเดรสภายในภาพไปค้นหาคำสั่งในหน่วยความจำหลัก (DRAM) ผ่านทาง Memory Interface ต่อไป
6. หน่วยความจำหลักได้รับแอดเดรสภายในภาพ แล้วนำส่งคำสั่งไปให้แคชลำดับที่ต่างๆ และวงจร Fetch เป็นเลขฐานสองจำนวน 4-8 คำสั่ง เป็นความยาว 128-256 บิต ขึ้นอยู่กับความกว้างของแคชแต่ละ บล็อก

5.3 หลักการพื้นฐานของหน่วยความจำแคช (Cache)

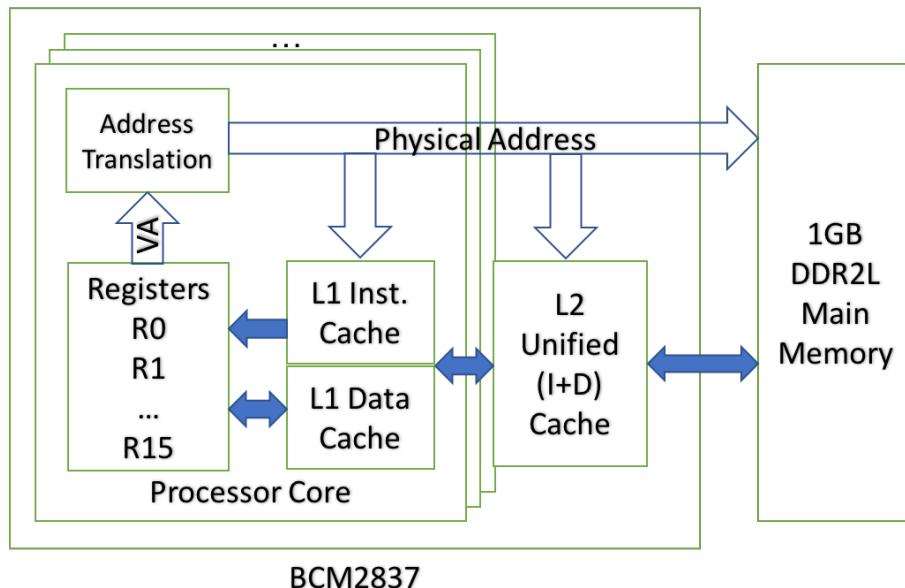


Figure 5.5: การเชื่อมต่อระหว่างรีจิสเตอร์ แคชลำดับที่ 1 และ 2 และหน่วยความจำหลัก (VA: Virtual Address) ภายในชิป BCM2837

รูปที่ 5.5 การเชื่อมต่อระหว่างรีจิสเตอร์ แคชลำดับที่ 1 และ 2 และหน่วยความจำหลัก ทำงานต่อเนื่องจากการแปลง VA ให้เป็น PA ประกอบด้วยลำดับที่ 1 แบ่งเป็น Instruction และ ข้อมูล (Data Only) เส้นสีขาว คือ แอ็ตเดรสภายภาพ เส้นสีน้ำเงิน คือ คำสั่งและข้อมูล ซึ่งจะอ่านคำสั่งจากแคชคำสั่งลำดับที่ 1 ก่อน หากไม่เจอก็ค้นคำสั่งนั้นจากแคชลำดับที่ถัดไป หากคำสั่งนั้น เป็นคำสั่งประเภท Load หรือ คำสั่ง STORE ระหว่างที่ซีพียูปฏิบัติตาม (Execute) จะค้นหา/เขียน ข้อมูลในแคชข้อมูล ลำดับที่ 1 ก่อน เช่นกัน แคชลำดับที่ 2 และลำดับที่ 3 เป็นแบบรวม (Unified Cache) ทำหน้าที่พักเก็บคำสั่งและข้อมูล

หน่วยความจำหลักซึ่งอาศัยหน่วยความจำชนิด DRAM ส่วนใหญ่จะอยู่ในระบบเป็นชิปแยกจากไมโครprocessor ในขณะที่ แคชจะมีอยู่บนชิป (On Chip) เดียวกัน ทำให้เกิดความล่าช้าระหว่าง การเคลื่อนที่ของข้อมูล ดังที่ได้กล่าวไปแล้วในหัวข้อที่ 5.5 การพักเก็บข้อมูลหรือคำสั่งที่มีการเรียกใช้บ่อยในแคช ทำให้ แคชเป็นประโยชน์ต่อระบบโดยภาพรวม เนื่องจากช่วยประหยัดเวลาในการรอการเดินของ ข้อมูลหรือคำสั่ง จากชิปสู่ชิป (Chip-Chip Delay) แคชจึงทำหน้าที่คล้ายกับบัฟเฟอร์ ในการพักเก็บข้อมูลหรือคำสั่ง ที่อ่านจากหน่วยความจำหลัก เพราะการทำงานของซอฟต์แวร์ส่วนหนึ่งจะเป็นการวนรอบ ทำให้ใช้คำสั่งเดิมๆ ซ้ำแล้วซ้ำอีก เรียกว่า **Temporal Locality**

การอ่าน/เขียนข้อมูลโดยเฉพาะข้อมูลชนิดอะเรย์ ซึ่งมีการจัดเรียงคล้ายกับรูปที่ 1.11 มีรูปแบบ (Pattern) ของความต่อเนื่องกันตามลำดับ เช่นเดียวกัน ดังนั้น ลักษณะนี้เรียกว่า **Spatial Locality** เชิงพื้นที่ (Space) ตรงกับการอ่าน/เขียนข้อมูลจาก DRAM แบบ Bulk Mode ทั้งสองชนิด เรียกรวมกันว่า **Principle of Locality**

แคชลำดับที่ 1 นิยมออกแบบด้วยแคชชนิด Direct Map แคชลำดับที่ 2 เป็นแบบรวมทำหน้าที่พักเก็บคำสั่งและข้อมูล จึงมีความจุมากกว่า แคชลำดับที่ 1 นิยมออกแบบด้วยแคชชนิด Set Associative ผู้ออกแบบนำเอาหลักการของสแตติคแรมมาทำหน้าที่เป็นแคชให้กับหน่วยความจำไดนามิกแรม มีวิธีการ

3 หลักการ แต่ละรุ่นนี้จะเปรียบเทียบเพียงแค่สองชนิดที่นิยมและเข้าใจง่าย โดยใช้ชุดคำสั่งเดียวกันในตารางที่ 5.1

Table 5.1: ตัวอย่างโปรแกรมภาษาแอสเซมบลีเพื่อประกอบการทำงานของแคช (PA: Physical Address)

แอ็ดเดรส ₁₀	Label	Code	Comment
0		LDR R1, [R3]	; get value of variable A
4		LDR R2, [R4]	; get value of variable B
8		BL Adder	; call A+B
12		B XYZ	; jump to XYZ
16			;
20			
24			
28			
32			
36			
40	Adder	ADD R1, R2, R1	; A=A+B
44		BX LR	
48			
52			
56	XYZ	...	
60			

5.3.1 แคชชนิดไดเร็คท์แมพ (Direct Map Cache)

รูปที่ 5.6 แสดงการทำงานของแคชชนิดไดเร็คท์แมพ ซึ่งนิยมใช้ออกแบบการทำงานของแคชลำดับที่ 1 (Level 1) ซึ่งมีความจุขนาดเล็ก เริ่ว เปรียบเทียบเหมือนหน่วยความจำบัฟเฟอร์เพื่อพักเก็บคำสั่งที่ใช้บ่อยๆ โดยในรูปแสดงให้เห็นว่าคำสั่งจากหน่วยความจำ 2 ตำแหน่งสามารถ ค้นหาในแคชตำแหน่งเดียวกัน เพราะหน่วยความจำมีขนาดเป็น 2 เท่าของขนาดแคชทำให้เกิดการแข่งขัน 2:1

- (a) เริ่มต้น แคชไม่มีข้อมูลใดๆ
- (b) คำสั่ง LDR R1, [R3] จะต้องถูกเฟลช (Fetch) หรืออ่านจากหน่วยความจำ แต่เนื่องจากแคชตำแหน่งที่ 0 ว่างเปล่า ทำให้ต้องนำคำสั่งนี้จากหน่วยความจำไปเก็บในแคชในตำแหน่งที่ 0 และนำไปถอดรหัสต่อไป
- (c) คำสั่ง LDR R2, [R4] จะต้องถูกเฟลชจากหน่วยความจำ แต่เนื่องจากแคชตำแหน่งที่ 4 ว่างเปล่า ทำให้ต้องนำคำสั่งนี้จากหน่วยความจำไปเก็บในแคชในตำแหน่งที่ 4 และนำไปถอดรหัสต่อไป
- (d) คำสั่ง BL Adder จะต้องถูกเฟลชจากหน่วยความจำ แต่เนื่องจากแคชตำแหน่งที่ 8 ว่างเปล่า ทำให้ต้องนำคำสั่งนี้จากหน่วยความจำไปเก็บในแคชในตำแหน่งที่ 8 และนำไปถอดรหัสต่อไป โดยจะต้องกระโดดไปยังหน่วยความจำตำแหน่งที่ 40 ต่อไป
- (e) คำสั่ง ADD R1, R2, R1 จะต้องถูกเฟลชจากหน่วยความจำ แต่เนื่องจากแคชตำแหน่งที่ 8 บรรจุคำสั่งจากหน่วยความจำตำแหน่งที่ 8 ดังนั้น ทำให้ต้องนำคำสั่ง ADD R1, R2, R1 จากหน่วยความจำตำแหน่งที่ 40 ไปเขียนในแคชตำแหน่งที่ 8 แทนและนำไปถอดรหัสต่อไป

Chapter 5. หน่วยความจำลำดับชั้น (Memory Hierarchy)

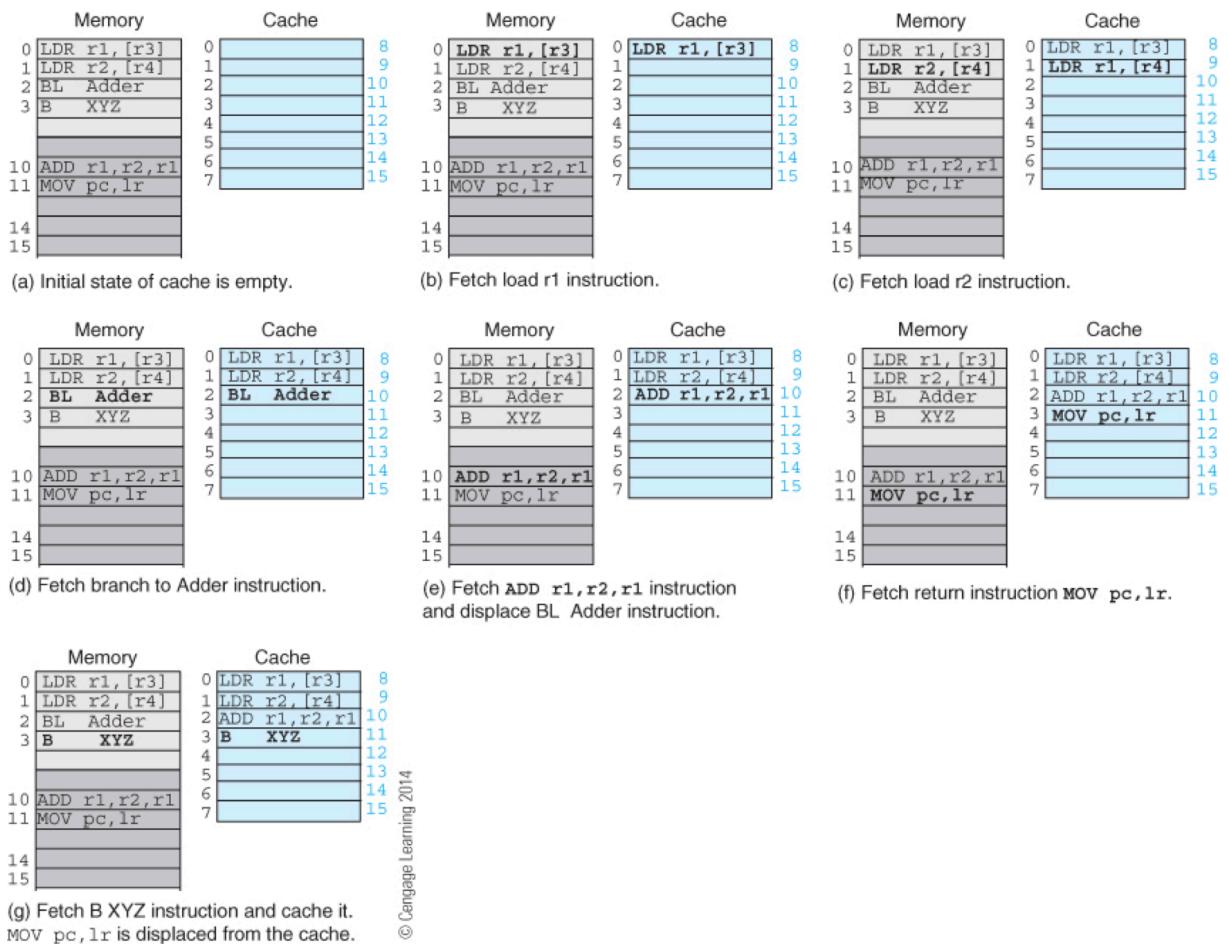


Figure 5.6: การทำงานของแคชคำสั่ง (Instruction Cache) ชนิด Direct Map ที่มา: [Clements \(2013\)](#)

(f) คำสั่ง MOV PC, LR จะต้องถูกเพลช์จากหน่วยความจำ แต่เนื่องจากแคชตำแหน่งที่ 12 ว่างเปล่า ทำให้ต้องนำคำสั่งนี้ไปเก็บในแคชในตำแหน่งที่ 12 และนำไปถอดรหัสต่อไป หลังจากนั้นจะต้องกระโดดกลับ (Return) ไปยังหน่วยความจำตำแหน่งที่ 12 ต่อไป นั่นคือคำสั่ง B XYZ

(g) คำสั่ง B XYZ จะต้องถูกเพลช์จากหน่วยความจำ แต่เนื่องจากแคชตำแหน่งที่ 12 บรรจุคำสั่ง MOV PC, LR ดังนั้น ทำให้ต้องนำคำสั่ง B XYZ จากหน่วยความจำตำแหน่งที่ 12 ไปเขียนในแคชตำแหน่งที่ 12 แทนและนำไปถอดรหัสต่อไป

ผู้อ่านจะสังเกตเห็นว่า แคชตำแหน่งใดๆ จะเป็นที่พักเก็บคำสั่งในหน่วยความจำ 2 ตำแหน่งเสมอ เช่น แคชตำแหน่งที่ 0 จะเป็นเป้าหมายของหน่วยความจำตำแหน่งที่ 0 และ 32 เป็นต้น ยิ่งไปกว่านั้น แคชลำดับที่ 1 ที่ใช้งานในซีพียูต่างๆ มีความจุน้อยกว่าหน่วยความจำหลักมาก เนื่องจากแคชอาศัยโครงสร้างพื้นฐานของสแตติกแรม จึงใช้พื้นที่บันชิพต่อกันๆ 1 บิตมากกว่า ทำให้เกิดการแย่งกัน (Contention) เป็นแบบ Many to One ดังนั้นมือแคชมีขนาดเล็ก อัตราการแย่ง (Contention Rate) จะยิ่งเพิ่มสูงมากขึ้น

5.3.2 แคชชนิดเซ็ตแอลโซซิเอทีฟ (Set Associative Cache)

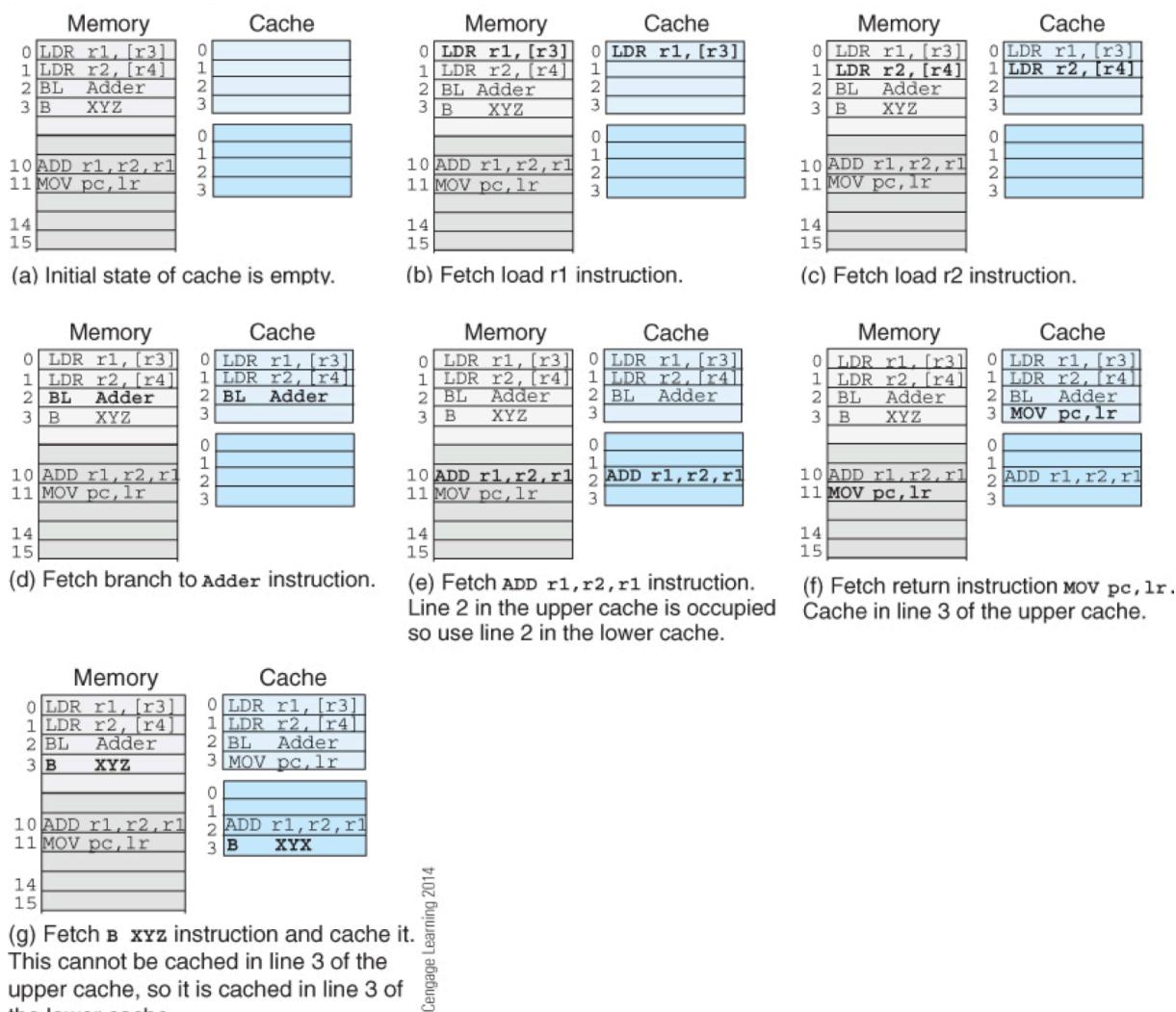


Figure 5.7: การทำงานของแคชลำดับที่ 1 แคชคำสั่ง (Instruction Cache) ชนิด Set Associative ที่มา: Clements (2013)

รูปที่ 5.7 แสดงการทำงานของแคชชนิด Set Associative Cache นิยมใช้ออกแบบแคชลำดับที่ 2 (Level 2) และเลเวล 3 (Level 3) ซึ่งมีความจุขนาดใหญ่

(a) เริ่มต้น แคชไม่มีข้อมูลใดๆ

(b) คำสั่ง LDR R1, [R3] จะต้องถูกเฟล์ชจากหน่วยความจำ แต่เนื่องจากแคชตำแหน่งที่ 0 ของทั้งแคชก้อนบนและก้อนล่างว่างเปล่า ทำให้ต้องนำคำสั่งนี้จากหน่วยความจำไปเก็บในแคชในตำแหน่งที่ 0 ของก้อนบน และนำไปถอดรหัสต่อไป

(c) คำสั่ง LDR R2, [R4] จะต้องถูกเฟล์ชจากหน่วยความจำ แต่เนื่องจากแคชตำแหน่งที่ 4 ว่างเปล่าทั้งบันและล่าง ทำให้ต้องนำคำสั่งนี้จากหน่วยความจำไปเก็บในแคชในตำแหน่งที่ 4 ของแคชก้อนบน และนำไปถอดรหัสต่อไป

(d) คำสั่ง BL Adder จะต้องถูกเฟล์ชจากหน่วยความจำ แต่เนื่องจากแคชตำแหน่งที่ 8 ว่างเปล่าทั้งสองก้อน ทำให้ต้องนำคำสั่งนี้จากหน่วยความจำไปเก็บในแคชในตำแหน่งที่ 8 ของก้อนบน และนำไปถอดรหัสต่อไป โดยจะต้องกระโดยไปยังหน่วยความจำตำแหน่งที่ 40 ต่อไป

(e) คำสั่ง ADD R1, R2, R1 จะต้องถูกเฟทซ์จากหน่วยความจำ แต่เนื่องจากแคชตำแหน่งที่ 8 ของก้อนบันบรรจุคำสั่งจากหน่วยความจำตำแหน่งที่ 8 แต่แคชก้อนล่างยังว่างอยู่ ดังนั้น ระบบจึงนำคำสั่ง ADD R1, R2, R1 จากหน่วยความจำตำแหน่งที่ 40 ไปเขียนในแคชตำแหน่งที่ 2 ของแคชก้อนล่างแทนและนำไปถอดรหัสต่อไป

(f) คำสั่ง MOV PC, LR จะต้องถูกเฟทซ์จากหน่วยความจำ แต่เนื่องจากแคชตำแหน่งที่ 12 ของก้อนล่างยังว่างเปล่า จึงนำคำสั่งนี้ไปเก็บในแคชในตำแหน่งที่ 12 ของก้อนล่างและนำไปถอดรหัสต่อไป หลังจากนั้นจะต้องعودةกลับ (Return) ไปยังหน่วยความจำตำแหน่งที่ 12 ต่อไป นั่นคือคำสั่ง B XYZ

(g) คำสั่ง B XYZ จะต้องถูกเฟทซ์จากหน่วยความจำ แต่เนื่องจากแคชตำแหน่งที่ 12 ของก้อนบันบรรจุคำสั่ง MOV PC, LR และตำแหน่งที่ 12 ของก้อนล่างยังว่างเปล่า ดังนั้น จึงนำคำสั่ง B XYZ จากหน่วยความจำตำแหน่งที่ 12 ไปเขียนในแคชตำแหน่งที่ 12 ของก้อนล่างและนำไปถอดรหัสต่อไป

ผู้อ่านจะสังเกตเห็นว่า แคชมีการทำงานคล้ายกับแคชชนิด DM แต่มีจำนวน 2 ทาง (2-Way) นั่นคือ ที่แคชตำแหน่งเดียวกันมี 2 ทางเลือก บน หรือ ล่าง ขึ้นอยู่กับว่า ก้อนใดว่าง หากไม่ว่างทั้งคู่ แคชจะตัดสินใจให้เขียนทับตำแหน่งที่ไม่ค่อยได้ใช้งาน

แคชชนิดนี้เป็นการขยายการทำงานของแคชชนิด DM ให้มีความยืดหยุ่นมากขึ้น ปัจจุบัน แคชชนิด SA ขนาด $n=8-16$ way ได้รับความนิยม

5.4 หน่วยความจำชนิด静态 RAM (Static RAM: SRAM)

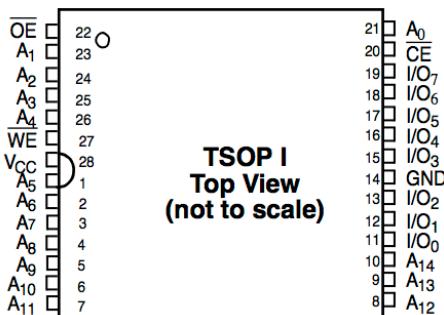


Figure 5.8: ตัวถังแบบ TSOP ของชิปหน่วยความจำชนิด static RAM Cypress 62256 ที่มา: [Cypress Semiconductor \(2002\)](#)

แคชลำดับที่ 1 และ 2 ในชิป BCM2837 สร้างจากหน่วยความจำ static RAM ซึ่งมีโครงสร้างที่ไม่ซับซ้อน และสามารถออกแบบให้ผลิตพร้อมกับวงจรท่านซิสเตอร์ในชิปปุ่ย นอกจากนี้ หน่วยความจำ static RAM นี้ ยังนิยมใช้งานเป็นหน่วยความจำหลักภายในชิปไมโครคอนโทรลเลอร์ ที่ต้องการสมรรถนะต่ำถึงปานกลาง โดยมีความจุหลายขนาด ตั้งแต่ 16 กิโลไบต์ ถึงหลายเมกะไบต์ ผู้เขียนขอใช้หน่วยความจำ static RAM (Static RAM: SRAM) หมายเลข CY62256 เป็นกรณีศึกษา ชิป CY62256 ใช้เทคโนโลยีการผลิตชนิด CMOS ในปี ค.ศ. 2002 เอกสารคุณลักษณะ (Datasheet) ของหน่วยความจำนี้ ที่มา: [Cypress Semiconductor \(2002\)](#) สามารถค้นเจอที่ ecee.colorado.edu ถึงแม้ว่าชิป CY62256 จะเป็นเทคโนโลยีที่เก่าแล้ว แต่ชิปมีลักษณะเรียบง่ายและโดยเด่น ดังนี้

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

ชิปตัวนี้สามารถลดการใช้พลังงานได้สูงสุดถึง 99.99 % และปิดการใช้งานได้อัตโนมัติ และสามารถเชื่อมต่อเพื่อขยายความจุ โดยใช้ขาสัญญาณ \overline{CE} (Chip Enable) สัญญาณ \overline{OE} (Output Enable) ซึ่งทำงานที่ลอจิคศูนย์ (Active Low) และใช้บัสข้อมูลร่วมกัน เพราะเชื่อมต่อกับชิปหน่วยความจำนี้ เพราะภายในชิปมี ไดรเวอร์สามสถานะ (Three State Driver)

ในเชิงโครงสร้าง ชิพอาศัยตัวถังแบบ TSOP (Thin Small Outline Package) เพื่อให้ เหมาะสมสำหรับ การบัดกรีเพื่อยืดตัวถังชิพบนแผ่นวงจรพิมพ์แบบ Surface Mount ผู้อ่านสามารถค้นคว้าเรื่องการห่อหุ้ม ชิพด้วยแพ็กเกจชนิด TSOP และอื่นๆ ได้ที่ [wikipedia](https://en.wikipedia.org) ซึ่งแต่ละชนิดมีข้อได้เปรียบและเสียเปรียบแตกต่าง กันไป

5.4.1 โครงสร้างภายในของชิป SRAM

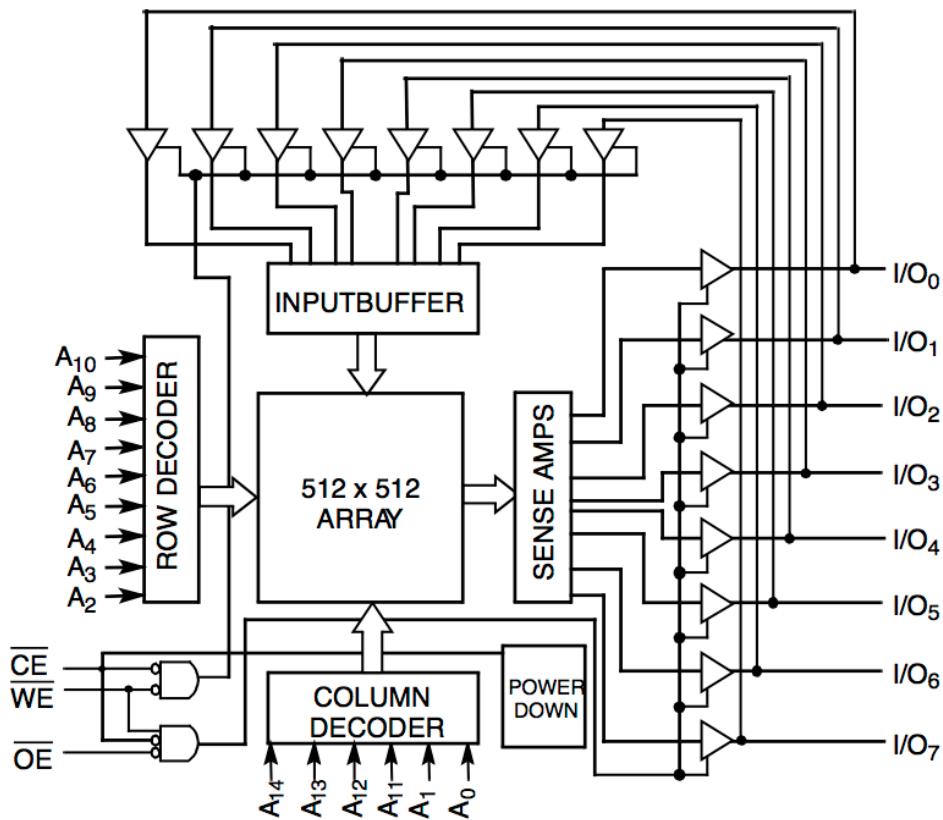


Figure 5.9: โครงสร้างของหน่วยความจำชนิดสแตติก Cypress 62256 ที่มา: [Cypress Semiconductor \(2002\)](#)

โครงสร้างของชิป CY62256 จัดเรียงเป็น $32K \times 8$ บิต หรือ $2^5 \times 2^{10} \times 2^3 = 2^{9+9}$ หรือ 512×512 บิต ซึ่งจะตรงกับรูปที่ 5.9

- การอ่านข้อมูลจะเกิดขึ้นโดยเปลี่ยนสัญญาณที่ขาตั้งต่อไปนี้
 - ขา $\overline{CE}=0$ และ $\overline{OE}=0$
 - ขาแอดเดรส A_0-A_{14} ทั้งหมด 15 ขา รับสัญญาณแอดเดรส เพื่อป้อนให้วงจร Decoder ทั้ง แนวอน (Row Decoder) และแนวคอลัมน์ (Column Decoder) ทำหน้าที่สร้างสัญญาณจำนวน 512 เส้นในแนวอนและแนวตั้งจำนวน 512 เส้น เพื่อเลือกบิตเซลล์ที่ต้องการในอะเรย์
 - โมดูล Sense Amplifier เป็นวงจรอนาล็อกขยายสัญญาณความไวสูง ใช้สำหรับขยายสัญญาณที่ส่งออกมาจากอะเรย์หน่วยความจำ เพราะสัญญาณที่ได้มีระดับโวลต์เจต้า รายละเอียดเพิ่มเติมที่ [wikipedia](#)
 - สัญญาณที่ผ่านการขยายและตรวจจับแล้วจะกลับเป็นข้อมูลดิจิทัลผ่านเกต Three State Buffer ออกไปยังขา I/O_0 จนถึง I/O_7 พร้อมกันทั้ง 8 บิต
- การเขียนข้อมูลจะเกิดขึ้นโดยเปลี่ยนสัญญาณที่ขาตั้งต่อไปนี้

- ขา $\overline{CE}=0$ $\overline{WE}=0$ และ $\overline{OE}=1$
- ข้อมูลจะไห่ลงเข้าทางขา I/O₀ จนถึง I/O₇ ผ่านเกต Three State Buffer มาพักใน
- โมดูล Input Buffer ใช้สำหรับพักข้อมูลชั่วคราว เพื่อรอเขียนในบิทเซลล์ที่ต้องการพร้อมกันทั้ง 8 บิต
- ขาแอดเดรส A0-A14 ทั้งหมด 15 ขา รับสัญญาณแอดเดรส เพื่อป้อนให้วงจร Decoder ทั้ง แนวอน (Row Decoder) และแนวคอลัมน์ (Column Decoder) ทำหน้าที่สร้างสัญญาณจำนวน 512 เส้นในแนวอนและแนวตั้งจำนวน 512 เส้น เพื่อเลือกบิทเซลล์ที่ต้องการเขียนใน อะเรย์ พร้อมกันทั้ง 8 บิต

แต่ละบิทเซลล์ในอะเรย์หน่วยความจำ ประกอบด้วยฟลิปฟล็อป (Flip-Flop) และเกต เพื่อควบคุมการอ่านและเขียน บิทเซลล์มีโครงสร้างที่ซับซ้อนและบริโภคพลังงานตลอดเวลา เนื่องจากมีการจัดเรียงเป็นลูป ป้อนกลับ (Feedback Loop) บิทเซลล์ใช้พื้นที่ซิลิกอน (Silicon Area) ต่อความจุข้อมูล 1 บิตคิดเป็นจำนวนทaranซิสเตอร์เฉลี่ย เท่ากับทaranซิสเตอร์ 6 ตัวขึ้นไป ทำให้การทำงานของหน่วยความจำชนิดสแตติคแรมรวดเร็ว ผู้อ่านสามารถค้นค่าวารายละเอียดเพิ่มเติม ได้ที่เว็บ [wikipedia](#)

5.4.2 การทำงานของสแตติคแรม: อ่านและเขียน

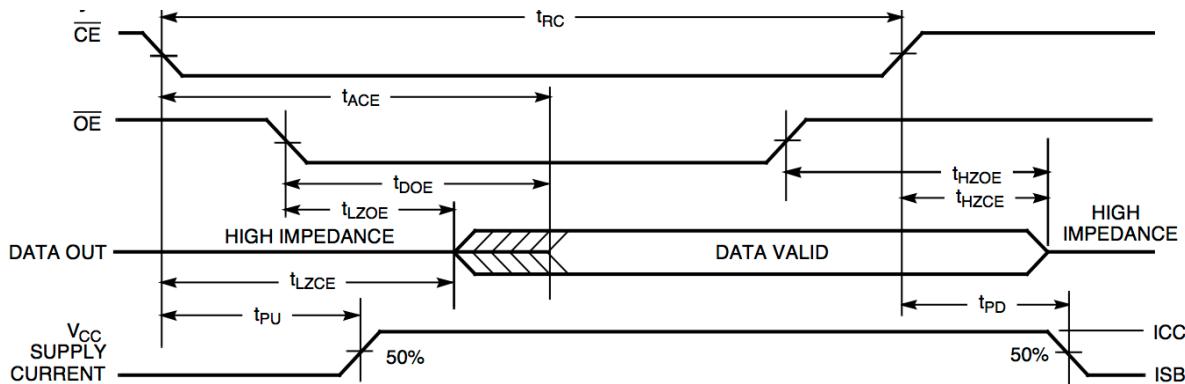


Figure 5.10: ไดอะแกรมเวลา (Timing Diagram) สำหรับการอ่าน (Read) ข้อมูลของหน่วยความจำชนิด สแตติคแรม ที่มา: [Cypress Semiconductor \(2002\)](#)

ไดอะแกรมเวลาของหน่วยความจำชนิดสแตติค ในรูปที่ 5.10 แกนนอนเป็นแกนเวลา เริ่มต้นจาก ซีพีyu ส่ง แอดเดรสไปยังหน่วยความจำ แล้วจึงเปลี่ยนขาสัญญาณ Chip Enable (\overline{CE}) และ Output Enable (\overline{OE}) จาก "1" เป็น "0" เพื่อกระตุ้นการทำงานของหน่วยความจำ หลังจากนั้น หน่วยความจำจะใช้เวลาอย่าง น้อย t_{ACE} ในการอ่านค่าข้อมูล ณ แอดเดรสหนึ่งๆ เพื่อส่ง ข้อมูลออกมาทางบัสข้อมูล ช่วงเวลาต่างๆ มีดังนี้

- t_{RC} เรียกว่า คาดเวลาที่สั้นที่สุดในการอ่านข้อมูลจากสแตติคแรม (Read Cycle Time) $t_{RC} > t_{ACE}$
- t_{ACE} เรียกว่า เวลาสำหรับการเข้าถึงข้อมูล หรือ Access Time โดยเริ่มนับจากเมื่อสัญญาณ CE 1->0 จนกว่าสามารถอ่านข้อมูลได้ถูกต้อง

- t_{HZOE} เรียกว่า เวลาที่จะรักษาข้อมูลไว้บนบัสข้อมูลเมื่อเปลี่ยนหน่วยแอดเดรสไปแล้ว
- t_{HZCE} เรียกว่า เวลาที่จะรักษาข้อมูลไว้บนบัสข้อมูลเมื่อขาสัญญาณ CE เปลี่ยนเป็น 1 แล้ว
- t_{PU} เรียกว่า Power Up Time เวลาที่กระแสไฟฟ้าเพิ่มขึ้นจาก 0% เป็น 50% โดยเริ่มนับจาก \overline{CE} เปลี่ยนจาก 1 เป็น 0
- t_{PD} เรียกว่า Power Down Time เวลาที่กระแสไฟฟ้าลดลงจาก 100% เป็น 50% โดยเริ่มนับจาก \overline{CE} เปลี่ยนจาก 0 เป็น 1

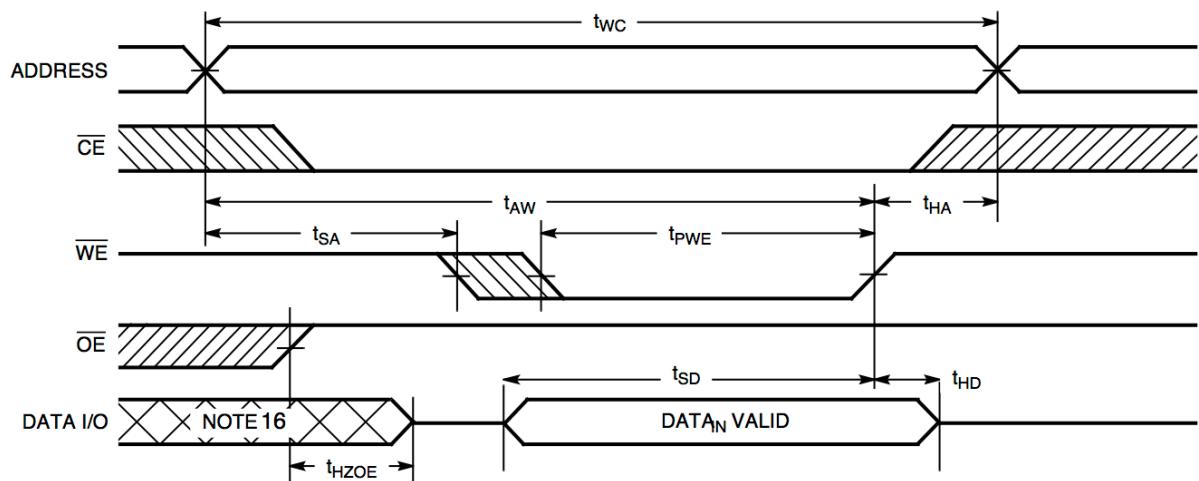


Figure 5.11: ไดอะแกรมเวลา (Timing Diagram) สำหรับการเขียน (Write) ข้อมูลของหน่วยความจำชนิดสแตติกแรม ที่มา: [Cypress Semiconductor \(2002\)](#)

Timing Diagram ของหน่วยความจำชนิดสแตติก ในรูปที่ 5.11

- t_{WC} เรียกว่า คาบเวลาที่สั้นที่สุดในการเขียนข้อมูลในสแตติกแรม (Write Cycle Time)
- t_{AW} เรียกว่า เวลาสำหรับการเขียน หรือ Access Write Time
- t_{HA} เรียกว่า เวลาสำหรับการตรึงแอดเดรส หรือ Hold Address Time
- t_{SD} และ t_{HD} เรียกว่า เวลาที่จะตรึงข้อมูลไว้ Data Stable Time เมื่อ Write Enable (\overline{WE}) หรือ Data Hold Time

5.5 หน่วยความจำหลักชนิดไดนามิกแรม (Dynamic RAM: DRAM)

ผู้ผลิตเครื่องคอมพิวเตอร์ในตบุค โทรศัพท์เคลื่อนที่สมาร์ทโฟน และอุปกรณ์พกพาต่างนิยมออกแบบติดตั้งชิพหน่วยความจำ DRAM บน เมนบอร์ด (Main Board) เช่นเดียวกับบอร์ด Pi3 เพื่อลดขนาดและปริมาตรของเครื่องให้มีขนาดเท่ากับบัตรเครดิต แต่ในเชิงเทคโนโลยีขั้นการผลิต DRAM ยังไม่สามารถรวมกับขั้นการผลิตไมโครโปรเซสเซอร์ได้ ผู้ผลิตจึงจำเป็นต้องผลิตชิพ DRAM แยกต่างหาก

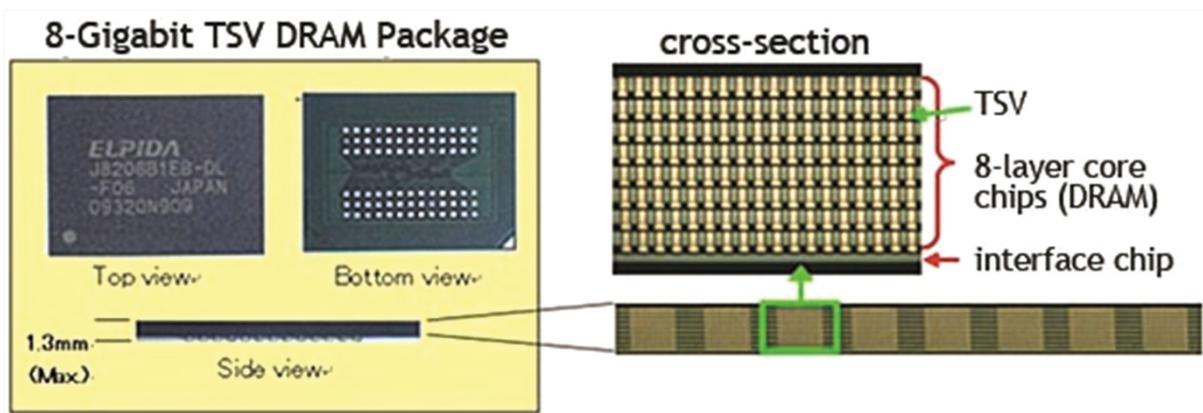


Figure 5.12: รูปถ่ายด้านบน (Top View) ด้านล่าง (Bottom View) ด้านข้าง (Side View) และภาพตัดขวาง (Cross Section) ของชิพ DRAM โดยใช้เทคโนโลยี TSV (Through Silicon Via) ผู้ผลิตบริษัท Elpida ที่มา: [Micron Technology \(2014\)](#)

ชิพหน่วยความจำ SDRAM ชนิด DDR2 ซึ่งติดยึดใต้บอร์ด Pi3 ตามรูปที่ 3.4 ช่วยประหยัดพื้นที่บนบอร์ดและสามารถเชื่อมต่อกับชิพ BCM2837 โดยใช้ระยะทางสั้นที่สุด ทำให้ชิพทั้งสองสามารถทำงานด้วยสัญญาณคลือกความถี่สูงสุดตามคุณลักษณะเฉพาะ (Specification) โดยรายละเอียดเบื้องต้นได้กล่าวไว้ไปบ้างแล้วในบทที่ 3 หัวข้อที่ 3.1.2 ชิพ DRAM มีความจุสูงมาก เนื่องจากใช้จำนวนทรานซิสเตอร์เฉลี่ยเพียง 1-2 ตัวต่อความจุ 1 กิกะไบต์ แต่มีข้อเสีย คือ ใช้เทคโนโลยีและขั้นการผลิตที่ซับซ้อน เนื่องจากชิพ DRAM มีความจุสูงมาก ในทางตรงกันข้าม ผู้ใช้ต้องการความเชื่อมั่นในตัวอุปกรณ์สูง และความถี่คลือกสูง เช่นกัน ทำให้ต้นทุนต่อความจุหนึ่งบิตของ DRAM สูงกว่าหน่วยเก็บรักษาข้อมูล

ชิพ DDR2 DRAM กรณีศึกษาที่ใช้บนบอร์ด Pi3 ผลิตโดยบริษัท Elpida ในปี ค.ศ. 2014 [Micron Technology \(2014\)](#) และต่อมาระยะต่อมาบริษัท ไมโครอน เทคโนโลยี ได้เข้าถือหุ้นแทน ชิพ DDR3 SDRAM นี้ใช้โครงสร้างแพ็คเกจและขาชนิด BGA (Ball Grid Array) จำนวน 168 ขา ด้านล่างเพียงอย่างเดียว ขนาด 12 มม. x 12 มม. x 0.8 มม. ผู้อ่านจะสังเกตได้ว่า แพ็คเกจชนิด BGA นี้จะใช้พื้นที่บนแผ่นวงจรพิมพ์เล็กกว่า เนื่องจากใช้เทคโนโลยีขั้นการผลิตที่ทันสมัยกว่า ในทางตรงกันข้าม ชิพ SRAM ชนิด TSOP ในรูปที่ 5.8 ซึ่งมีข่ายในออกมากด้านข้าง

รูปที่ 5.12 แสดงชิพ DRAM ขนาด 8 Gbit โดยบริษัท Elpida ที่มา: [electroiq.com](#) ภาพถ่ายด้านบน (Top view) มีสกรีนชื่อบริษัทผู้ผลิต หมายเลขอุปกรณ์ ประเภท และอื่นๆ ภาพถ่ายด้านล่าง แสดงให้เห็นขาสำหรับเชื่อมต่อเข้าสู่ภายในชิพ จำนวน สองแฉวๆ ละ 39 ขา และรอย balk ตรงมุมขวาล่าง เมื่อพลิกคว่ำลงจะตรงกับจุดวงกลมด้านซ้ายล่างของ ภาพถ่ายด้านข้างของชิพ (Side view) มีความสูงเท่ากับ 1.3

มีลลิเมตร เมื่อขยายภาพตัดขวาง (cross-section) ให้ใหญ่ขึ้น ผู้อ่านจะมองเห็นขาที่ยื่นออกมาจากตัวถัง ภาพตัดขวางที่ขยายทางด้านขวามีสุด จะมองเห็นเป็น 8 ชั้น (8-layer core) การเชื่อมต่อสัญญาณบน แผ่นซิลิกอนแต่ละชั้น จะเกิดขึ้นในแนวราบภายในแผ่นหรือชั้นเดียวกัน แต่การเชื่อมระหว่างชั้นของแผ่น ซิลิกอนเข้าด้วยกันในแนวตั้ง โดยใช้ TSV (Through Silicon Via, [wikipedia](#)) เพื่อเชื่อมสัญญาณควบคุม สัญญาณคล็อก ไฟเลี้ยง gravard และอื่นๆ เกิดการเชื่อมต่อสัญญาณครบทั้งสามมิติ ทำให้ประหยัดพื้นที่ วางแผ่นซิลิกอนในแนวราบ เรียกว่า อินเตอร์คอนเน็คท์ (Interconnect) ส่วนชั้นล่างสุดเป็นวงจรเชื่อมต่อ (Interface) ซึ่งอยู่ใกล้กับขาที่อยู่ใต้ชิป BCM2837

5.5.1 โครงสร้างภายในของชิป DRAM

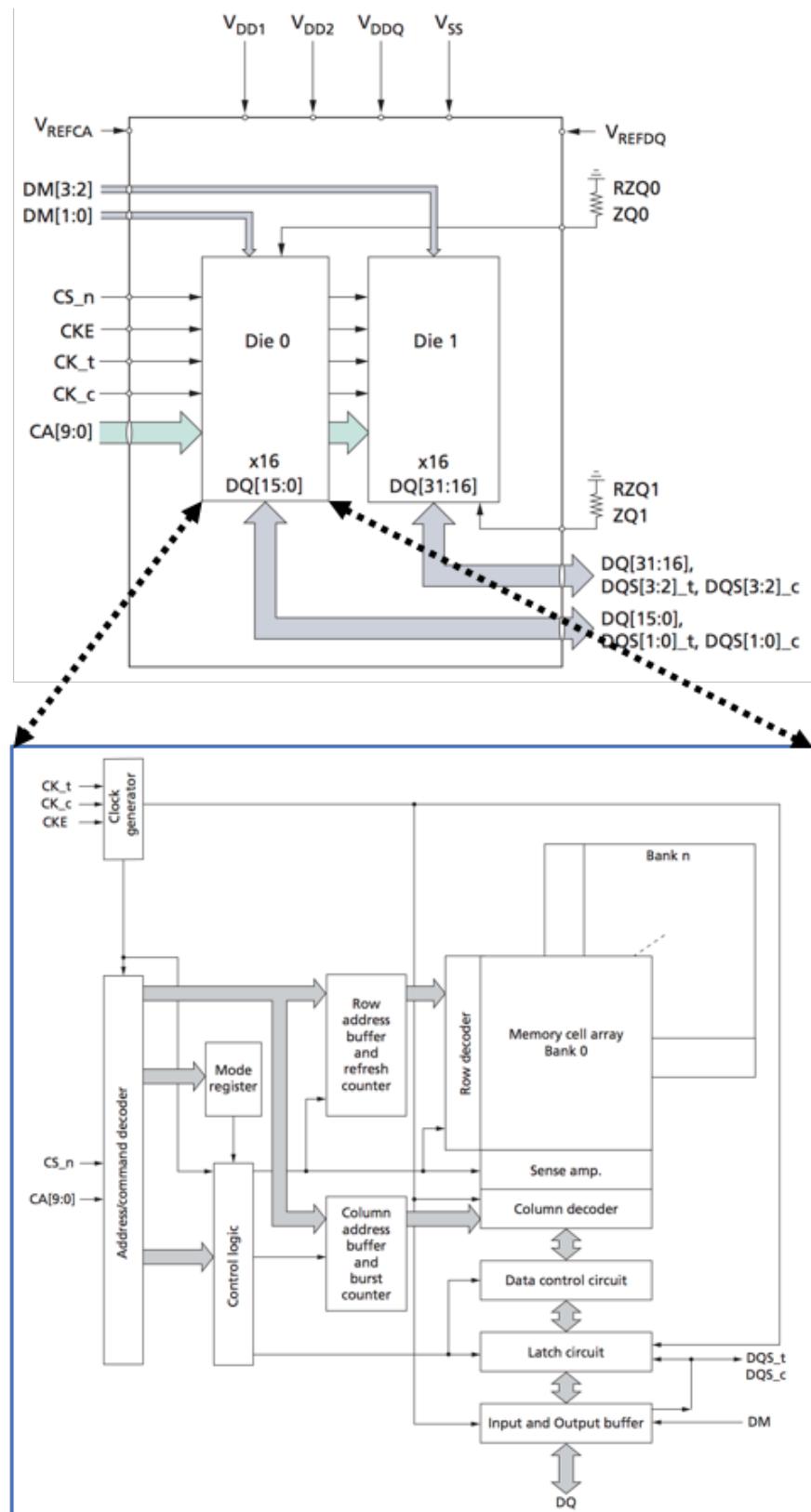


Figure 5.13: โครงสร้างภายในชิปหน่วยความจำ DRAM ชนิด DDR2 ประกอบด้วยดาย (Die) 2 ชิ้น แต่ละชิ้นมีบล็อกข้อมูลความกว้าง 16 บิต รวมเป็น 32 บิต ที่มา: [Micron Technology \(2014\)](#)

ในรูปที่ 5.13 ชิพ DRAM กรณีศึกษา โครงสร้างของหน่วยความจำชนิด DDR2 จำนวน 2 ดาย (die) แต่ละ ดายประกอบด้วย แบงอะเรย์ DRAM จำนวน 8 ชั้น หรือ 8 แบงค์ (Bank) ชั้นละ 32 เมกะเซลล์ \times 16 บิต คิดเป็น $2 \times 8 \times 32$ เมกะเซลล์ \times 16 บิต \times ต่อชิพ หรือ $2^1 \times 2^3 \times 2^5 \times 2^{20} \times 2^4 = 2^{33} = 2^3 \times 2^{30} = 8$ Gbits = 1 GByte

แบงค์ที่ 0 ถึง 7 แต่ละแบงค์ประกอบด้วยวงจรอุดตรหัสแอดเดรส (Address Decoder) ในแนวนอน (Row Decoder) และแนวตั้ง (Column Decoder) ภายในชิพประกอบด้วยขาสัญญาณต่างๆ เรียงตาม ลำดับความสำคัญ ดังนี้

- ***CS_n*** (Chip Select Not) หรือ \overline{CS} ใช้เปิด/ปิดการทำงานของชิพ เพื่อช่วยประหยัดพลังงาน
- **CKE (Clock Enable)** เมื่อสัญญาณ $\overline{CS}=0$ เพื่อให้ชิพทำงาน หลังจากนั้น สัญญาณ CKE ใช้สำหรับ เปิด/ปิดการทำงานของคลีอกที่จ่ายให้กับชิพ DRAM นี้ ซึ่งมีความสามารถควบคุมสัญญาณ CKE=0 เพื่อ พักการใช้งาน DRAM ชั่วคราวเพื่อช่วยประหยัดพลังงาน
- **CK (Clock)** และ CK# (Clock Not) คือ สัญญาณคลีอกสองสัญญาณที่มีตรังข้ามขั้วตรงข้ามกัน เรียกว่า **คู่ดิฟเฟอเรนเชียล** (Differential Pair) หน่วยเป็นเมกะเฮิรท์ สัญญาณคลีอกความถี่สูงสุด 400 เมกะเฮิรท์ ชิพ DDR4 สำหรับคอมพิวเตอร์ตั้งโต๊ะความถี่สูงสุดเท่ากับ 3,000 เมกะเฮิรท์ และ มีแนวโน้มเพิ่มขึ้นตามเทคโนโลยีการผลิตที่พัฒนาอย่างต่อเนื่อง ผู้อ่านสามารถค้นคว้าเพิ่มเติมได้ที่ [wikipedia](#)
- แอดเดรสคลอลัม CA[0:9] ขนาด 10 บิต ใช้สำหรับรับสัญญาณแอดเดรสและคำสั่ง (Command) โดยการมัลติเพล็กซ์ (Multiplex) จากซีพียู โดยใช้ขาจำนวน 10 ขาเพื่อรับคำสั่ง (Command) และ สัญญาณแอดเดรส (Address) ต่างหัวใจกัน การรับคำสั่งเกิดขึ้น ณ ขอบขาขึ้นและขอบขาลง ของแต่ละสัญญาณคลีอก
 - **สัญญาณแอดเดรสแกร** (Row Address) จำนวน 14 บิต และ **แอดเดรสคลอลัม** (Column Address) จำนวน 11 บิต จะพักเก็บในวงจรบัฟเฟอร์ เพื่อป้อนให้กับวงจรอุดตรหัส (Decoder) คล้ายกับการทำงานของหน่วยความจำ SRAM
 - ซีพียุจะส่งคำสั่งต่างๆ ดังนี้ Activate, Burst Read, Burst Write, Refresh, Power Down, Precharge และ Burst Terminate เป็นต้น เพื่อกำหนดโหมดการทำงานของหน่วยความจำ DRAM ได้แก่ Power Up, Deep Power Down, Active, Idle, Reading, Writing, Precharging, Refreshing เป็นต้น
- **ดาต้าสโตรบ DQS[0:3]** ขนาด 4 บิต ใช้สำหรับควบคุมการอ่านและการเขียนข้อมูล
- **ดาต้าบัส DQ[0:31]** จำนวน 32 บิต หรือ 4 ไบต์ สำหรับอ่านข้อมูลจากชิพ DRAM และเขียนข้อมูล ในชิพ ชิพ DDR2 นี้ สามารถกำหนดขนาดข้อมูลจากการอ่านเขียนต่อเนื่อง (Burst length) เป็น 4, 8 และ 16 ตำแหน่งต่อเนื่องกัน

Chapter 5. หน่วยความจำลำดับขั้น (Memory Hierarchy)

- ตามมาสก์ DM (Data Mask) [0:3] ขนาด 4 บิต ใช้สำหรับมาสก์ (Mask) หรือปิด เพื่อควบคุมการเขียนข้อมูลแต่ละบีท โดย DM[0] ควบคุมบีทที่ 0, DM[1] ควบคุมบีทที่ 1 ตามลำดับ ทำให้การเขียนข้อมูลแต่ละบีทเป็นอิสระจากกันได้

ผู้อ่านจะสังเกตเห็นว่า วงจรรอบๆ อะเรย์หน่วยความจำ (Cell Array) มีลักษณะที่คล้ายกับสแตติคแรม ดังนี้

- Row Decoder สร้างสัญญาณจำนวน $2^{14} = 16,384$ เส้น เรียกว่า bit line ในแนวราบ
- Column Decoder สร้างสัญญาณจำนวน $2^{11} = 2048$ เส้น เรียกว่า word line ในแนวตั้ง
- Sense Amplifier ใช้สำหรับขยายสัญญาณในขบวนการอ่าน เช่นเดียวกับการอ่านของ SRAM

จุดตัดระหว่าง Bit Line และ Word Line ทั้งสองเส้น คือ ตำแหน่งของบีทเซลล์ที่ต้องการจะอ่านหรือเขียน คล้ายกับการทำงานของหน่วยความจำสแตติคแรม แต่หน่วยความจำ DRAM มีโครงสร้างของบีทเซลล์ที่แตกต่าง กล่าวคือ บีทเซลล์แต่ละบีทของ DRAM ประกอบด้วยทรานซิสเตอร์ชนิด MOS จำนวน 1-2 ตัว โดยอาศัยหลักการมีประจุในขาเกต (Gate) ปทนาข้อมูล 1 หรือ ไม่มีประจุในขาเกต แทนข้อมูล 0 ผู้อ่านสามารถคลิกบนคำว่า [ไดนามิกแรมขนาด 4x4 บิต](#) เพื่อศึกษาโครงสร้างบีทเซลล์ของ DRAM เพิ่มเติม

5.5.2 การทำงานของไดนามิกแรม: อ่านและเขียน

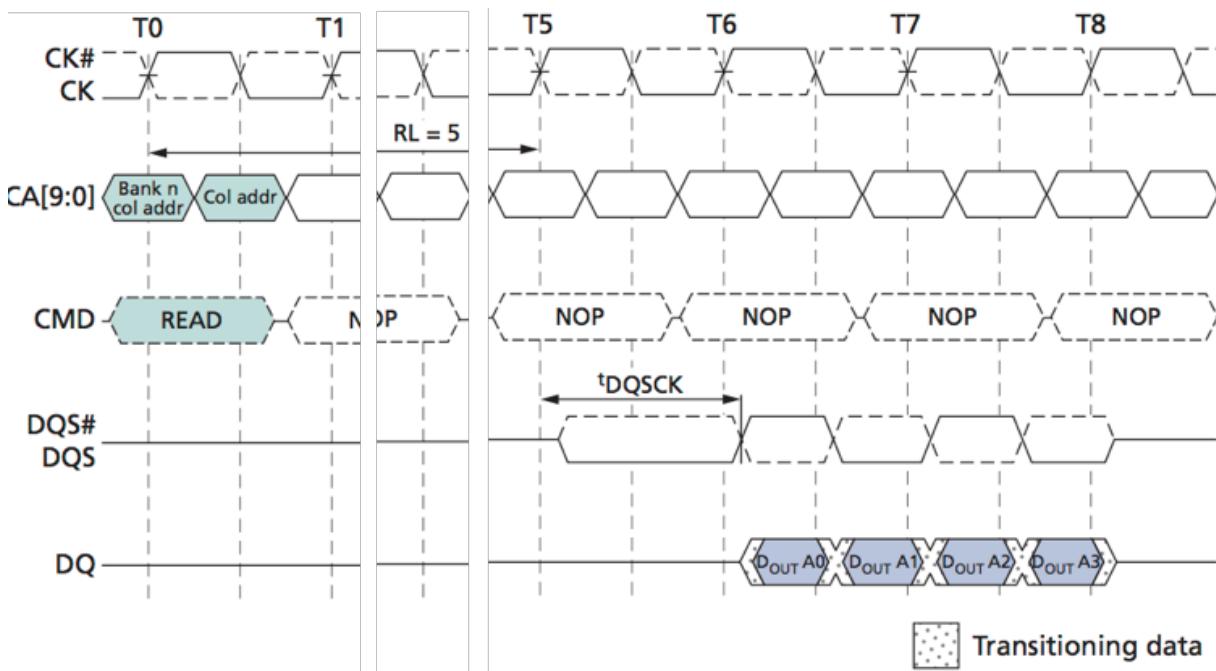


Figure 5.14: ไดอะแกรมเวลา (Timing Diagram) สำหรับการอ่านข้อมูลของหน่วยความจำ DRAM รุ่น Elpida B8132B4PB-8D-F คำสั่ง Burst READ โดย RL= 5 BL= 4 หมายเหตุ รูปมีการตัดต่อเพื่อเน้นบริเวณที่สำคัญ, NOP=No Operation

การทำงานของหน่วยความจำ DRAM มีความหลากหลาย โดยคำสั่งที่ใช้งานบ่อยที่สุด คือ คำสั่ง Burst READ และ คำสั่ง Burst Write ดังนี้

คำสั่ง Burst READ เริ่มต้นโดย ทำให้สัญญาณ $CS\#$ หรือ $\overline{CS}=0$, CA0=1, CA1=0 และ CA2 = 1 ณ ขอบขาขึ้นของสัญญาณคลีก็อก รูปที่ 5.14 แสดงถึงไดอะแกรมเวลาสำหรับการอ่านข้อมูลของหน่วยความจำ ชนิด DDR แบบซิงโครนัสซึ่งต้องทำงานด้วยขอบขาขึ้นของสัญญาณทั้งขาขึ้นและขาลงของสัญญาณคลีก็อก ในรูปเป็นการทำงานตามคำสั่ง Burst READ ด้วยค่า RL (Read Latency) = 5 ไซเคิล นับจาก T1-T5 โดยต้องรอเวลาโดยการนับจำนวนคลีก็อกเป็นจำนวน RL (Read Latency) ไซเคิล โดยเริ่มนับจากขอบขาขึ้นของคำสั่ง READ หมายเหตุ รูปมีการตัดต่อเพื่อเน้นบริเวณที่สำคัญ และอ่านข้อมูลต่อเนื่องเป็นจำนวน BL (Burst Length) = 4 ชุด เรียngติดกันโดยเริ่มจากแอดдресที่น้อยกว่า (A0) วงจรจะใช้สัญญาณ DQS (Data Strobe) เพื่อชิงครอในซิกกับการอ่านข้อมูล ที่มาสำหรับ [ดาวน์โหลดเอกสาร](#)

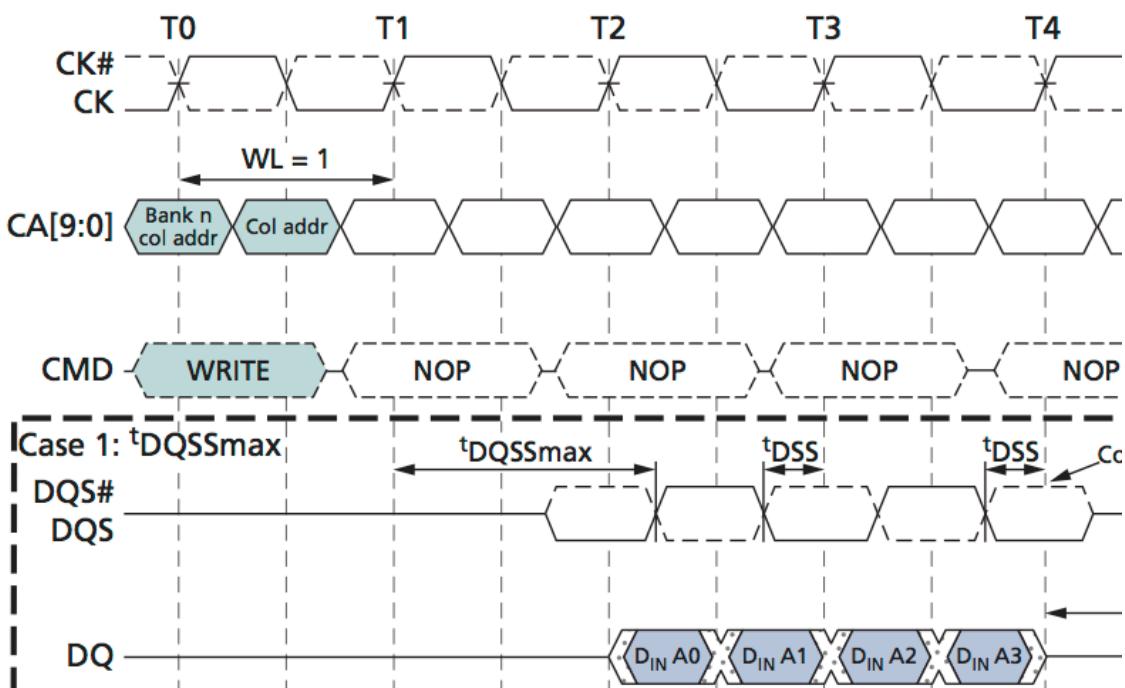


Figure 5.15: ไดอะแกรมเวลา (Timing Diagram) สำหรับการเขียนข้อมูลของหน่วยความจำ Elpida รุ่น B8132B4PB-8D-F ตามคำสั่ง Burst WRITE – WL = 1, BL = 4 หมายเหตุ รูปมีการตัดต่อเพื่อเน้นบริเวณที่สำคัญ ที่มา: [Micron Technology \(2014\)](#)

ต่อมาเป็นตัวอย่างการเขียนด้วยคำสั่ง Burst WRITE ในรูปที่ 5.15 โดย WL (Write Latency) = 1 ไซเคิลและ BL (Burst Length) = 4 ไซเคิล เรียngติดกันโดยเริ่มจากแอดdressที่น้อยกว่า (A0) วงจรจะใช้สัญญาณ DQS (Data Strobe) เพื่อชิงครอในซิกกับการเขียนข้อมูล ผู้อ่านจะสังเกตเห็นว่า ขอบสัญญาณ DQS จะเปลี่ยนแปลงตรงกลางข้อมูลสำหรับการเขียน แต่ในรูปที่ 5.14 ขอบสัญญาณ DQS จะเปลี่ยนแปลงตามข้อมูลสำหรับการอ่าน

5.5.3 การรีเฟรชข้อมูล

การรีเฟรช (Refresh) คือ การอ่านข้อมูลที่อยู่ในบิทเซลล์ต่างๆ แล้วเขียนชั้งไป เพื่อป้องกันไม่ให้ประจุที่เก็บอยู่ในบิทเซลล์ต่างๆ รั่วไหลหายไป เนื่องจากเซลล์ต่างๆ ที่ใช้เก็บข้อมูล ทำหน้าที่เก็บ ('1') หรือไม่เก็บ

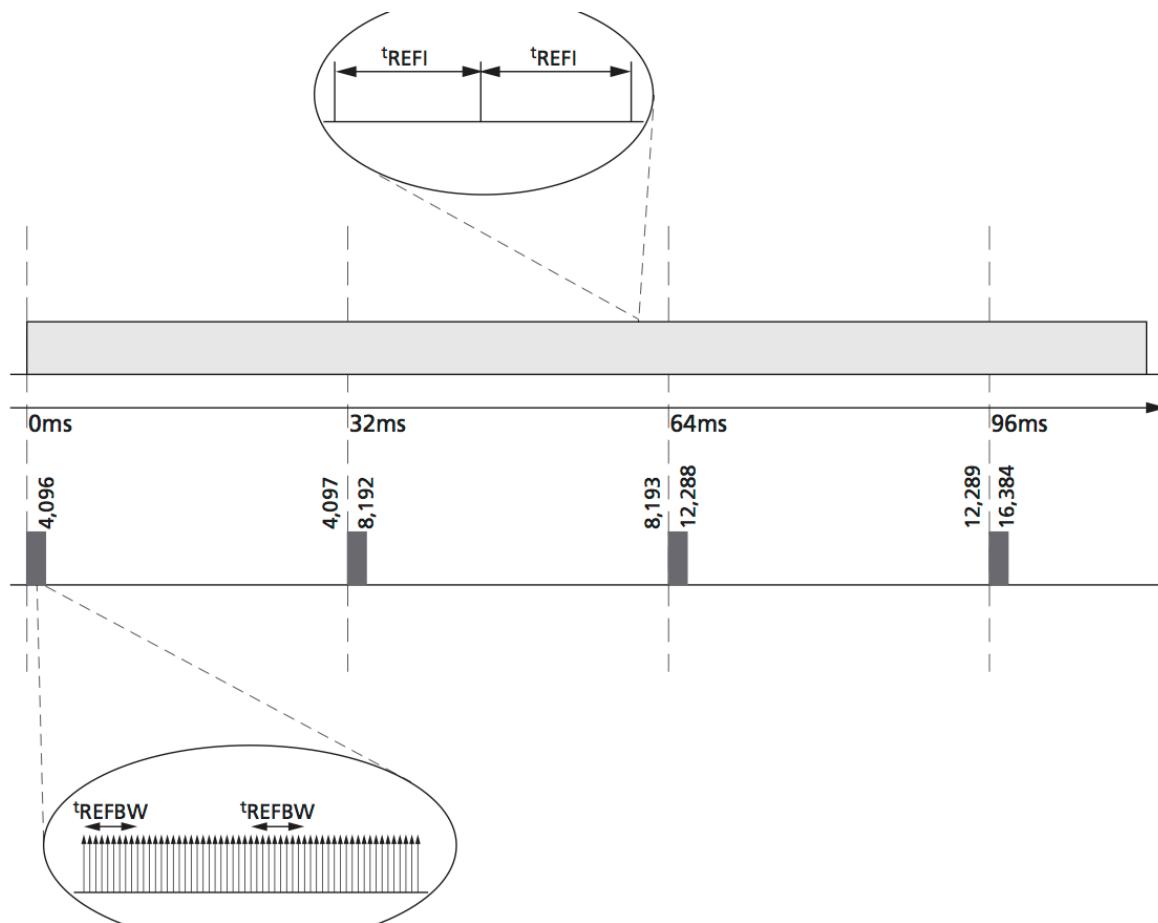


Figure 5.16: ไดอะแกรมเวลาสำหรับการรีเฟรชหน่วยความจำชนิด DRAM ขนาด 1 กิกะบิต ที่มา: [Micron Technology \(2014\)](#)

('0') ประจุไฟฟ้า เมื่อเวลาผ่านไปไม่กี่มิลลิวินาที จึงเกิดการร้าวไฟลของประจุเหล่านี้ เมื่อจำนวนประจุลดลง การแยกแยะระหว่างบิทเซลที่มีและไม่มีประจุจึงยากขึ้น ซึ่งอาจทำให้เกิดความผิดพลาดในการอ่าน

แทนนอนในรูปที่ 5.16 คือ แทนเวลา การรีเฟรชทุกๆ 32 มิลลิวินาที โดยส่งคำสั่ง Refresh 4096 คำสั่ง เป็นระยะเวลา $t_{REFW}=32$ มิลลิวินาที แต่ละคำสั่งจะห่างกันด้วยระยะเวลา Refresh Interval $t_{REFI}=7.8$ ไมโครวินาที หนึ่งคำสั่งจะทำการ Refresh เป็นชุดๆ แต่ละชุดด้วยระยะเวลา Refresh Burst Window $t_{REFBW}=4.16$ ไมโครวินาที โดยแต่ละชุด จะทำการรีเฟรชทีลแบงค์ด้วยระยะเวลา 60 นาโนวินาที

หากมีการรีเฟรชดำเนินการอยู่ การอ่านหรือเขียนหน่วยความจำจะต้องหยุดรอ เพื่อให้ขบวนการรีเฟรชนั้นเสร็จสิ้น ในทำนองเดียวกัน หากมีการอ่านหรือเขียนข้อมูลจริงอยู่ การรีเฟรชจะต้องหยุดรอ ก่อนเพื่อให้ขบวนการอ่านหรือเขียนนั้นเสร็จสิ้น

หน่วยความจำสแตติคแรมไม่ต้องมีการรีเฟรชข้อมูล เนื่องจากการจัดเก็บข้อมูลใช้วิธีการเก็บข้อมูล ที่แตกต่างกับหน่วยความจำ DRAM ทำให้ SRAM มีสมรรถนะ และประสิทธิภาพสูงกว่า แต่ต้องใช้จำนวนทรานซิสเตอร์มากกว่า จึงทำให้ใช้พื้นที่ซิลิคอนต่อความจุข้อมูล 1 บิต มากกว่าเซ่นกัน ทำให้ SRAM ใช้พลังงานไฟฟ้าต่อกำลัง 1 บิตโดยเฉลี่ยสูงกว่า ผู้อ่านสามารถค้นคว้าเพิ่มเติมได้ที่ [wikipedia](#)

5.6 สรุปท้ายบท

หน่วยความจำลำดับชั้นาอย่างการบริหารรวมกันของหน่วยความจำหลักและหลักข้ามกันเนื่องจากมีเทคโนโลยีที่แตกต่างกัน หน่วยความจำขนาดเล็กความเร็วสูง เช่น SRAM นำมาใช้งานเป็นริจิสเตอร์ และ แคช ทำหน้าที่เป็นตัวแทนของหน่วยความจำที่มีความจุมากกว่าแต่ความเร็วต่ำกว่า เช่น DRAM นำมาใช้งานเป็นหน่วยความจำหลัก และใช้เทคโนโลยี หน่วยความจำแฟลช และ ฮาร์ดดิสก์เป็นอุปกรณ์เก็บรักษาข้อมูล เพื่อให้คอมพิวเตอร์มีความจุเพียงพอและตอบสนองต่อความต้องการใช้งานระบบโดยเฉลี่ยได้รวดเร็วขึ้น โดยการผสานจุดเด่นของหน่วยความจำแต่ละชนิดเข้าด้วยกัน และช่วยประหยัดต้นทุนของระบบ

หลักการของแคมป์บากาทต่อประสิทธิภาพของระบบโดยองค์รวม แคมป์นิด N-way Set Associative ได้รับความนิยม เนื่องจากมีโครงสร้างเหมือนกับแคมป์นิด Direct Mapped และมีความสามารถคล้ายแคมป์นิด Fully Associative มีการนำหลักการ Principle of Locality มาประยุกต์ใช้กับหน่วยความจำประเภทต่างๆ และหลากหลาย หนึ่งในนั้น คือ หน่วยความจำเสมือน นั่นเอง หน่วยความจำเสมือนอาศัยการทำงานร่วมกันระหว่างฮาร์ดแวร์และระบบปฏิบัติการ เพื่อเพิ่มลำดับชั้นในการบริหารจัดการหน่วยความจำทุกลำดับชั้นดังได้กล่าวไปแล้ว

เนื้อหาในบทนี้ว่าด้วยเรื่องของหน่วยความจำผนวกกับความรู้ในบทก่อนหน้า ผู้เขียนจึงขออุปมาอุปมาภัยการทำงานของคอมพิวเตอร์โดยรวม ดังนี้

- ซอฟต์แวร์ คือ สูตรหรือขั้นตอนการประกอบอาหารอย่างละเอียด เพื่อให้ฟอร์มร่วมประกอบอาหารให้สุกและมีรสชาติดี
- ซีพียู คือ พอกรัวและเครื่องครัวที่ประกอบอาหารตามสูตรเท่านั้น ประกอบด้วย
 - ALU คือ อุปกรณ์เครื่องครัว เช่น มีด เขียง กระทะ หม้อ เป็นต้น
 - Register คือ งานหรือที่พักอาหารที่ปูรุ่งค้าง เพื่อรอทำให้เป็นอาหารที่ปูรุ่งสำเร็จต่อไป
 - วงจรอื่นๆ
- หน่วยความจำ คือ ชั้นวางวัตถุดิบ (ข้อมูล) ขนาดเล็ก ไม่สามารถเก็บไว้ต่ำๆได้นาน
- ข้อมูลดิบ คือ วัตถุดิบสำหรับประกอบอาหาร อาจมาจากแหล่งข้อมูลหลายๆ แหล่ง เช่น ผู้ใช้กรอกข้อมูลในโปรแกรม ไฟล์ข้อมูลในหน่วยสำรอง เป็นต้น
- ข้อมูลหรือสารสนเทศ คือ อาหารที่ประกอบและปูรุ่งสำเร็จแล้ว โดยอาศัยซีพียูและซอฟต์แวร์ที่กล่าวมาข้างต้น



Figure 5.17: การประกอบอาหารตามสูตร ที่มา: <https://www.pinterest.com>

- หน่วยเก็บรักษาข้อมูล คือ ตู้แช่เย็นขนาดใหญ่ที่สามารถบรรจุวัตถุติดไฟฟ้ามาก และระยะเวลานาน หากต้องการเก็บข้อมูลหรือวัตถุติดไฟไว้ทานต่อไป ข้อมูลติดไฟหรือวัตถุติดไฟมักมีปริมาณมากๆ วัตถุติดไฟเหล่านี้จะเก็บในรูปของไฟล์ข้อมูล ซอฟต์แวร์สามารถอ่านหรือเขียนข้อมูลจากไฟล์ที่มีลักษณะต้องการ
- หากวัตถุติดมาจากการต่างประเทศ จะมาจากเซิร์ฟเวอร์อื่นๆ บนเครือข่ายอินเทอร์เน็ต ก่อนจะนำมาปรุง ข้อมูลติดไฟเหล่านี้จะถูกอ่านขึ้นมาเก็บอยู่ในหน่วยความจำหลัก ในรูปของไฟล์ข้อมูล หรือ ในรูปของข้อมูลที่จะจัดกระจายมาทางเซิร์ฟเวอร์ต่างๆ ด้วยเทคโนโลยี IoT (Internet of Thing)
- หากข้อมูลมีปริมาณมากๆ การประมวลผลข้อมูลจำเป็นต้องอาศัยซีพียูและ/หรือจีพียุหลายๆ คอร์ เพื่อร่วมประมวลผลแบบขนาน (Parallel Computing) ทำให้การประมวลผลเสร็จสิ้นรวดเร็วขึ้น

5.7 คำถ้ามท้ายบท

1. หน่วยความจำในบทนี้ทั้งหมด สามารถเก็บข้อมูลได้หรือไม่หากไม่มีไฟเลี้ยง
2. เหตุใดเครื่องคอมพิวเตอร์จึงต้องมีหน่วยเก็บรักษาข้อมูล คู่กับ หน่วยความจำในบทนี้
3. จงเปรียบเทียบการอ่านของหน่วยความจำ SRAM และ DDR-SDRAM ในแง่มุมเหล่านี้
 - ขนาดของข้อมูลขั้นต่ำที่สามารถอ่านได้ หน่วยเป็นไบท์
 - ระยะเวลาที่ต้องรอ (Latency) เพื่อให้ข้อมูลปรากฏบนบัสข้อมูล
4. จงเปรียบเทียบการเขียนของหน่วยความจำ SRAM และ DDR-SDRAM ในแง่มุมเหล่านี้
 - ระยะเวลาที่ต้องรอ (Latency) เพื่อให้ข้อมูลเขียนสำเร็จ
 - ขนาดของข้อมูลขั้นต่ำและขั้นสูงที่สามารถเขียนได้
5. เหตุใดหน่วยความจำสารแตติกแรมจึงไม่ต้องรีเฟรชข้อมูล
6. เหตุใดหน่วยความจำไดนามิกแรมจึงต้องรีเฟรชข้อมูลภายใต้ระดับๆ

Chapter 6

อุปกรณ์/วงจรอินพุตและเอาท์พุต (Input and Output Devices/Circuit)

ผู้ใช้สามารถถอดอาศัยเครื่องคอมพิวเตอร์ติดต่อกับโลกภายนอกผ่านทางอุปกรณ์อินพุต/เอาท์พุต เช่น การเชื่อมต่อกับผู้ใช้ผ่านทางคีย์บอร์ด เม้าส์ หน้าจอสัมผัส การขยับ (Motion) / การเอียง (Tilt) / ความเร่ง (Acceleration) ของการเคลื่อนไหว การเชื่อมต่อกับเครือข่ายอินเตอร์เน็ต ผ่านทางเครือข่ายมีสายและเครือข่ายไร้สาย เป็นต้น ดังนั้น การเชื่อมต่อกับอุปกรณ์เหล่านี้จำเป็นต้องทำตามมาตรฐาน เพื่อลดความยุ่งยากในการออกแบบและต้นทุนโดยรวมของคอมพิวเตอร์

ระบบปฏิบัติการจะทำหน้าที่บริหารจัดการอุปกรณ์อินพุต/เอาท์พุต เพื่อให้โปรแกรมต่างๆสามารถใช้ทรัพยากรเหล่านี้ร่วมกัน โดยจะต้องมีการป้องกันและการจัดลำดับการใช้งาน (Protection and Scheduling) เนื่องจากการใช้งานอุปกรณ์อินพุต/เอาท์พุต ทำให้เกิดการอินเตอร์รัพท์ (Interrupt) แปลว่า ขัดจังหวะการทำงานของซีพียู ระบบปฏิบัติการจะต้องจัดเตรียมวิธีการเชื่อมต่อกับอุปกรณ์อินพุต/เอาท์พุต โดยมีชาร์ดแวร์ที่ควบคุมการทำงานของอุปกรณ์ต่างๆ แทนโดยตรง เพื่อให้ ซีพียูทำงานที่สำคัญได้อย่างต่อเนื่องโดยเน้นที่การประมวลผลข้อมูล ซึ่งวิธินี้จะเพิ่มประสิทธิภาพและการตอบสนองต่อผู้ใช้โดยรวมได้ และเพื่อให้โปรแกรมเมอร์สามารถพัฒนาโปรแกรมได้อย่างรวดเร็ว มีประสิทธิภาพและปลอดภัย บทที่ 2 มีวัตถุประสงค์ดังต่อไปนี้

- เพื่อให้เข้าใจสัญญาณการเชื่อมต่อกับอุปกรณ์อินพุต/เอาท์พุตชนิดต่างๆ ของบอร์ด Pi3
- เพื่อให้รู้จักโครงสร้างและการทำงานด้านอินพุต/เอาท์พุตของบอร์ด Pi3
- เพื่อให้เข้าใจกลไกการติดต่อกับอุปกรณ์อินพุต/เอาท์พุตชนิดต่างๆ
- เพื่อให้เข้าใจหลักการ Memory Mapped I/O, Interrupt และ Direct Memory Access

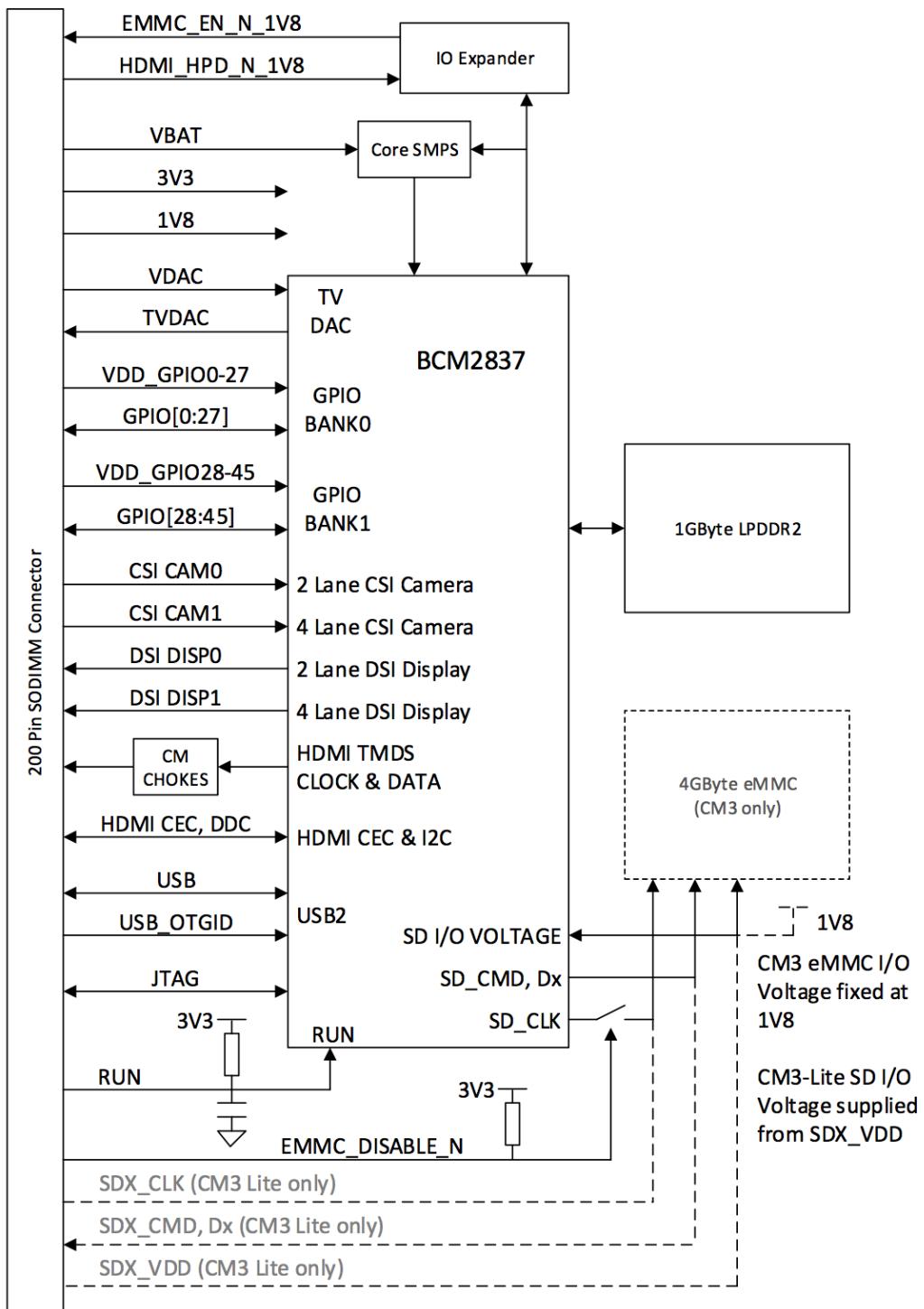


Figure 6.1: การเชื่อมโยงอุปกรณ์ต่างๆ บนบอร์ด Pi3 โดยมีชิป BCM2837 หรือ CM3 (Compute Module 3) เป็นศูนย์กลางด้วยขาจำนวน 200 ขา ที่มา: [Ltd \(2019\)](#)

เนื้อหาในบทนี้จะเน้นเรื่องของการเชื่อมต่อ BCM2837 หรือ CM3 (Compute Module 3) ภายใต้ชิป ARM Cortex A53 จำนวน 4 คอร์ ตัวชิปประกอบด้วยขาทั้งหมด 200 ขา เชื่อมต่อกับโลกภายนอก ซึ่งส่วนใหญ่จะเป็นขาเชื่อมต่ออุปกรณ์หรือวงจรอินพุตและเอาท์พุต ตามรูปที่ 2.1 เช่น HDMI, USB และอุปกรณ์เก็บรักษาข้อมูล กลไกการเชื่อมต่อที่ลึกลงไป จะปรากฏในการทดลอง การทดลองที่ 7 ภาคผนวก I การศึกษาและปรับแก้อินพุต/เอาท์พุตต่างๆ เป็นต้นไป

6.1 สัญญาณ HDMI สำหรับจอภาพ LCD ขนาดใหญ่

เนื้อหาในหัวข้อนี้ต่อเนื่องจาก HDMI (High-Definition Multimedia Interface) ในหัวข้อที่ 3.1.3.3 สำหรับเขื่อมต่อจอแสดงผลภายนอก HDMI คือ สัญญาณสำหรับการเชื่อมต่ออุปกรณ์ภาพและเสียง ซึ่งเข้ามาแทนที่การเชื่อมต่อรูปแบบเดิมๆ เช่น สัญญาณคอมโพสิตวีดีโอ และแบบ S-video

การเชื่อมต่อแบบ HDMI เป็นการถ่ายโอนสัญญาณแบบดิจิทัล สามารถส่งได้ทั้งสัญญาณภาพและสัญญาณเสียงไปพร้อมๆ กันด้วยอัตราบิทเรตสูง ระดับจิกะบิทต่อวินาที การเชื่อมต่อด้วยสัญญาณ HDMI หมายความว่าสำหรับการแสดงผลจากเครื่องคอมพิวเตอร์หรือเครื่องเล่นมีเดีย (Media Player) ไปยังจอภาพความละเอียดสูงระดับเอชดี (HD) หรือสูงกว่าสำหรับจอความละเอียด Ultra HD HDMI เวอร์ชันล่าสุด คือ 2.1 ซึ่งพัฒนาเพื่อรองรับวีดีโอที่ละเอียดสูงเฟรม率 8000 เส้นต่อ 60 เฟรมต่อวินาที (8K60) และเฟรม率 4000 เส้นต่อ 120 เฟรมต่อวินาที (4K120) และเพิ่มถึงเฟรม率 10,000 เส้น (10K) ซึ่งจะทำให้อัตราบิทเรตเพิ่มเป็น 48 จิกะบิทต่อวินาที

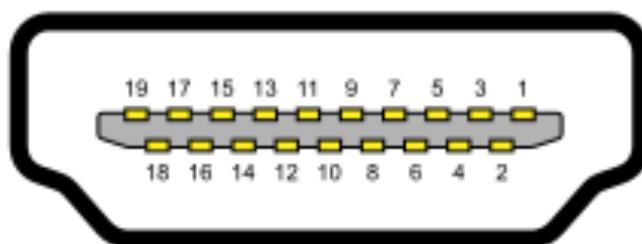


Figure 6.2: หัวเชื่อมต่อ HDMI ชนิด Female ประกอบด้วยขาสัญญาณทั้งหมด 19 ขา ที่มา: [wikipedia](#)

สาย HDMI ที่ใช้เชื่อมต่อส่วนใหญ่จะเป็นตัวผู้ (Male) ทั้งสองด้าน รูปที่ 2.2 แสดงภาพตัดขวางของหัวเชื่อมต่อ HDMI ชนิดตัวเมีย (Female) ประกอบด้วยขา 19 ขา มีรายชื่อตามตารางที่ 2.1 ตามหมายเลขขาซึ่ง และวัตถุประสงค์ของคอนเน็คเตอร์ชนิด HDMI เวอร์ชัน 1.4 ที่มา: ? รายละเอียดเพิ่มเติมที่: [wikipedia](#)

จากตารางที่ 2.1 ผู้อ่านจะสังเกตได้ว่า สัญญาณ HDMI มีช่องสื่อสาร 5 ช่องแยกจากกัน ได้แก่

- ช่อง TMDS (Transition-Minimized Differential Signaling) ช่อง TMDS จะส่งข้อมูลภาพวีดีโอเสียง และข้อมูลเป็นดิจิทัล ในรูปของแพ็กเก็ต (Packet) แต่ละแพ็กเก็ตข้อมูล ประกอบด้วย ช่วงส่งข้อมูลภาพ (Video Data Period) และช่วงส่งแพ็กเก็ตควบคุม (Control Period) สำหรับสัญญาณควบคุม ยกตัวอย่างเช่น สัญญาณ HSYNC (Horizontal Synchronization) และ VSYNC (Vertical Synchronization) เป็นต้น

รูปที่ 2.3 แสดงตัวอย่างการส่งแพ็กเก็ตข้อมูลจะใช้สัญลักษณ์สีเทา และแพ็กเก็ตควบคุมจะใช้สัญลักษณ์สีฟ้า ในช่วงเวลาต่างๆ สำหรับ การแสดงภาพด้วยความละเอียด 720x480 ต่อ 1 เฟรมแต่ละเฟรมใช้เวลา 33 มิลลิวินาทีเพื่อการแสดงผล หรือคิดเป็น 30 เฟรมต่อวินาที ภายในระยะเวลา 33 มิลลิวินาทีจะเกิดการส่งสัญญาณภาพเป็นจำนวน 525 เส้นๆ ละ 858 พิกเซลแต่แสดงผลให้เห็นเพียง 480 เส้นๆ ละ 720 พิกเซลในตำแหน่งที่มีสีเทาเข้มเท่านั้น ส่วนที่เกินเพื่อไว้สำหรับช่วงเวลา Vertical Blanking และ Horizontal Blanking ช่วงเวลาทั้งสองจะใช้ในการส่งสัญญาณควบคุมและข้อมูลอื่นๆ โดยแพ็กเก็ตข้อมูลสำหรับจุดภาพเส้น (Line) บนสุด ตำแหน่งจุดภาพซ้ายสุดไปจุดภาพขวาสุด

Table 6.1: หมายเลขขา ชื่อ และวัตถุประสงค์ของคอนเนคเตอร์ชนิด HDMI เวอร์ชัน 1.4 ที่มา: [wikipedia](#)

ขา	ชื่อ	วัตถุประสงค์
1	TMDS Data2+	สำหรับข้อมูล Control Period
2	TMDS Data2 Shield	ข้อมูลเลน 2 ขั้วบวก ชีล์ดสำหรับข้อมูลเลน 2
3	TMDS Data2-	ข้อมูลเลน 2 ขั้วลบ
4	TMDS Data1+	สำหรับข้อมูลเสียง
5	TMDS Data1 Shield	ข้อมูลเลน 1 ขั้วบวก ชีล์ดสำหรับข้อมูลเลน 1
6	TMDS Data1-	ข้อมูลเลน 1 ขั้วลบ
7	TMDS Data0+	สำหรับข้อมูลภาพ
8	TMDS Data0 Shield	ข้อมูลเลน 0 ขั้วบวก ชีล์ดสำหรับข้อมูลเลน 0
9	TMDS Data0-	ข้อมูลเลน 0 ขั้วลบ
10	TMDS Clock+	สัญญาณคล็อกขั้วบวก
11	TMDS Clock Shield	ชีล์ดสำหรับสัญญาณคล็อก
12	TMDS Clock-	สัญญาณคล็อกขั้วลบ
13	CEC	ช่อง Consumer Electronics Control
14	Reserved	สงวนไว้ใช้ในอนาคต
15	SCL	สัญญาณคล็อกสำหรับช่อง DDC
16	SDA	สายข้อมูลสำหรับช่อง DDC
17	Ground	กราวด์
18	+5.0 V	ไฟเลี้ยง 5.0 โวลท์
19	Plug Detect	ขาตรวจจับการเชื่อมต่อ

การส่งแพ็กเก็ตจะขับลงมา 1 เส้น แล้วเริ่มจากตำแหน่งจุดภาพซ้ายสุดไปจุดภาพขวาสุด เช่นเดิม การส่งแพ็กเก็ตจะขับลงมาเรื่อยๆ จนถึงเส้นภาพสิ้นสุด แล้วจึงเริ่มต้นภาพใหม่ด้วยเส้นภาพบนสุด เช่นเดิม

ในสาย HDMI หนึ่งเส้นประกอบด้วยเส้น TMDS จำนวน 3 เส้นสำหรับส่งแพ็กเก็ตข้อมูลพร้อมกัน รายละเอียดเพิ่มเติมที่ [หัวข้อ Transition Minimized Differential Signaling](#) ของ Wikipedia

- ช่อง DDC (Display Data Channel) ใช้สื่อสารกับเครื่อง Media Player ด้วยมาตรฐาน I2C เพื่อกำหนดรูปแบบของวิดีโอและเสียงที่ส่งไปยังจอภาพ และยังใช้คุ้มครองเนื้อหาดิจิทัลแบบดิจิทัล (High-bandwidth Digital Content Protection: HDCP) เช่น ภาพยนต์ที่มีลิขสิทธิ์ เป็นต้น
- ช่อง CEC (Consumer Electronics Control) ผู้ใช้งานสามารถควบคุมอุปกรณ์ต่อพ่วงผ่านช่อง CEC ได้มากถึง 15 ตัว เป้าหมายคือ อุปกรณ์สามารถทำงานร่วมกันและใช้เครือข่ายอินเตอร์เน็ต ไปพร้อมๆ กัน
- ช่อง ARC (Audio Return Channel) หรือช่องสัญญาณเสียงคืนกลับ เพื่อใช้สาย HDMI เชื่อมต่อกับตัว盒อัตรัสเสียงและเครื่องขยายเสียงผ่านทางช่อง ARC

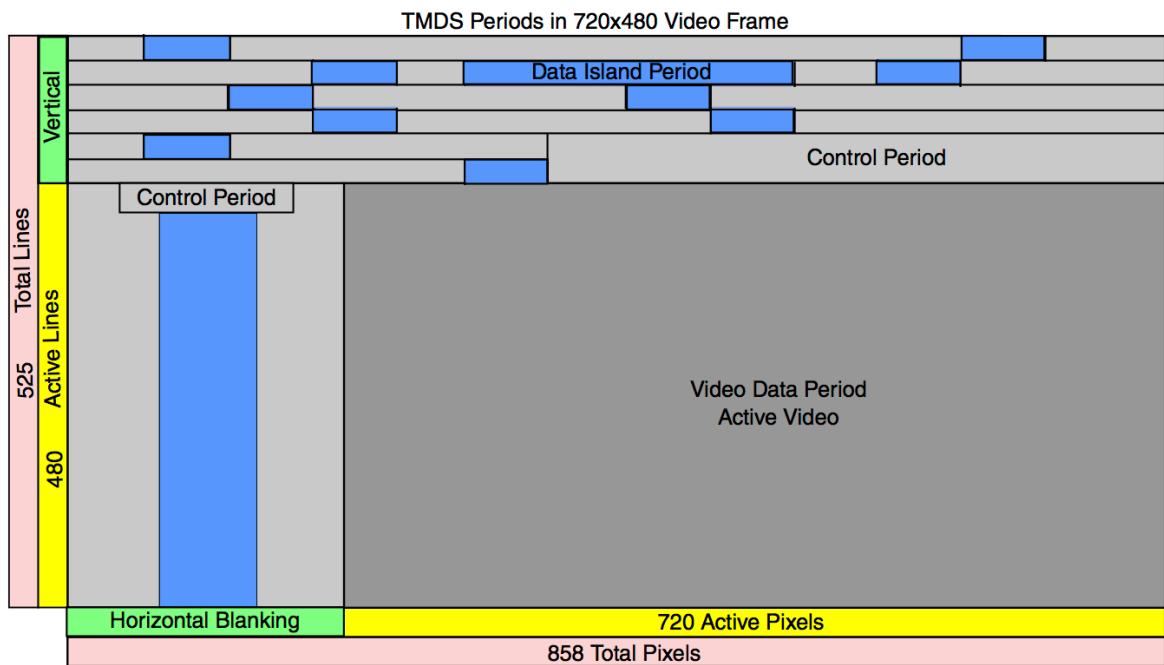


Figure 6.3: การส่งแพ็กเก็ตข้อมูลและควบคุมด้วยช่องสัญญาณ TMDS ในช่วงต่างๆ สำหรับความละเอียดในการแสดงผล 720x480 ต่อเฟรม ที่มา: freebsd.org

- HEC (HDMI Ethernet Channel) ใช้เชื่อมต่ออุปกรณ์อื่นๆ ผ่านทางสาย HDMI ตั้งแต่เวอร์ชัน 1.3 เป็นต้นมา

6.2 สัญญาณ DSI สำหรับจอภาพ LCD ขนาดเล็ก

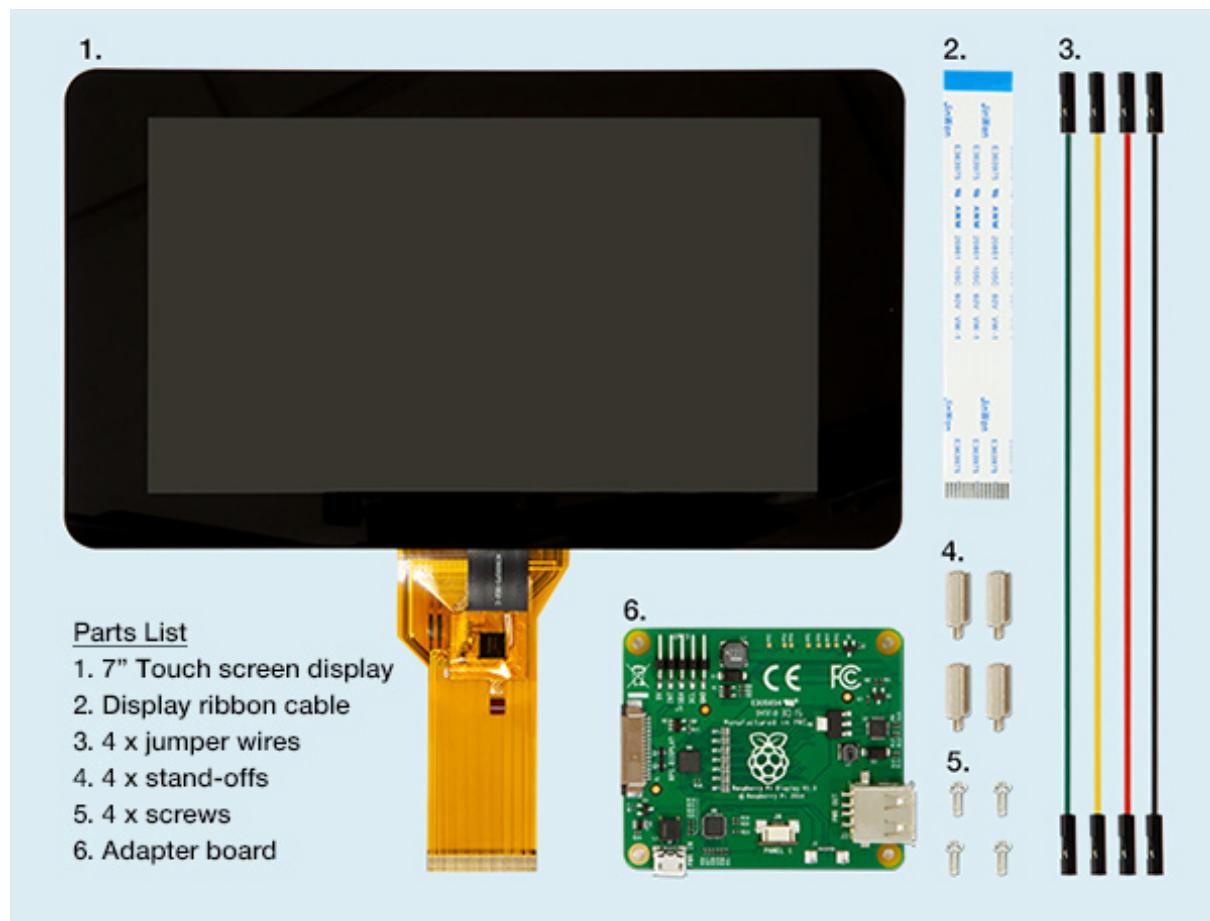


Figure 6.4: จอยแสดงผลสำหรับเชื่อมต่อระหว่างบอร์ด Pi3 ด้วยอินเตอร์เฟสการแสดงผลแบบอนุกรม (Display Serial Interface) ที่มา: element14.com

สัญญาณ DSI (Display Serial Interface) สำหรับเชื่อมต่อ กับชิปยูนอุปกรณ์ เคลื่อนที่ เช่น โทรศัพท์เคลื่อนที่ แท็บเล็ต คอมพิวเตอร์โน้ตบุค เป็นต้น เพื่อการแสดงผลในรูปของกราฟิก ใหม่ๆ สัญญาณ DSI นี้ถูกกำหนดเป็นมาตรฐานโดยองค์กรชื่อ MIPI (Mobile Industry Processor Interface)

สำหรับบอร์ด Pi3 คอนเนกเตอร์ S2 เป็นหัวเชื่อมต่อสัญญาณ DSI โดยใช้สายแพจำนวน 15 ชา โดยส่งข้อมูลพร้อมๆ กันจำนวน 2 เลน แต่ละเลนเป็นแบบอนุกรม (Serial) ตารางที่ 2.2 แสดงหมายเลข ชื่อและวัตถุประสงค์ของขาทั้ง 15 ชา โดย ชา 5 และ 6 จะถูกกำหนดให้เป็นเลนสัญญาณคลือก ชา 8 และ 9 จะถูกกำหนดให้เป็นเลนข้อมูลหมายเลข 0 ชา 2 และ 3 จะถูกกำหนดให้เป็นเลนข้อมูลหมายเลข 1 เป็นต้น

สัญญาณ DSI แบ่งเป็นชนิดเลนเดียว (Single Lane) และหลายๆ เลนตั้งแต่ 2 เลนขึ้นไป เพื่อกระจายการส่งข้อมูลแต่ละไบท์ข้อมูล ไปแต่ละเลนตามรูปที่ 2.5 ข้อมูลไบท์ที่ 0, 4, 8, ... จะส่งมาทางเลนหมายเลข 0 ข้อมูลไบท์ที่ 1, 5, 9, ... จะส่งมาทางเลนหมายเลข 1 ข้อมูลไบท์ที่ 2, 6, 10, ... จะส่งมาทางเลนหมายเลข 2 ข้อมูลไบท์ที่ 3, 7, 11, ... จะส่งมาทางเลนหมายเลข 3 และสลับกันไปแบบนี้เรื่อยๆ การส่งข้อมูลจำนวนหลายๆ เลนพร้อมกันทำได้เร็วขึ้น รองรับการแสดงผลที่ละเอียดมากขึ้น เปลี่ยนแปลงภาพต่อวินาทีได้มาก

Table 6.2: หมายเลข และหน้าที่ของสายสัญญาณ DSI สำหรับจอภาพ LCD ขนาดเล็กด้วยข้อมูลภาพจำนวน 2 เลน ที่มา: [wikipedia](#)

ขา	ชื่อ	หน้าที่
1	Ground	กราวด์
2	Data Lane 1-	ขาลงเลนข้อมูล 1
3	Data Lane 1+	ขาขึ้นเลนข้อมูล 1
4	Ground	กราวด์
5	Clock N	ขาลงคลีอก
6	Clock P	ขาขึ้นคลีอก
7	Ground	กราวด์
8	Data Lane 0-	ขาลงเลนข้อมูล 0
9	Data Lane 0+	ขาขึ้นเลนข้อมูล 0
10	Ground	กราวด์
11		
12		
13	Ground	กราวด์
14	+3.3 V	ไฟเลี้ยงขนาด 3.3 โวลท์
15	+3.3 V	ไฟเลี้ยงขนาด 3.3 โวลท์

ขั้น การเคลื่อนไหวของภาพจึงต่อเนื่องไม่กระตุก เพิ่มอรรถรสในการรับชมมากขึ้น เมื่อป้ายทางรับข้อมูลได้สำเร็จ วงจรรับจะนำข้อมูลเหล่านั้นมารวมกัน (Lane Merging Function)

ตามมาตรฐานสัญญาณ DSI เลนข้อมูลแต่ละเลนใช้หลักการส่งสัญญาณแบบ Differential Signalling คล้ายกับสัญญาณของ USB แต่ใช้ความต่างศักย์ในระดับ 200 มิลลิโวลท์ เรียกว่า Low Voltage Differential Signalling (LVDS) แต่ไม่ได้กำหนดรายละเอียดของการเชื่อมต่อกับอินพุทชนิดสัมผัส (Touch Screen Sensor) จอภาพ LCD จะทำงานในโหมดแสดงผล และโหมดรับคำสั่ง โหมดรับคำสั่ง ซึ่งจะกำหนดค่าต่างๆ ของรีจิสเตอร์ได้ เพื่อควบคุมการทำงานของจอตามที่ต้องการ โดยอาศัยการรับส่งข้อมูลและคำสั่งในเลนที่ 0 ส่วนเลนที่ 1 เป็นต้นไปจะทำหน้าที่ส่งข้อมูลเท่านั้น เลนหมายเลข 0 สามารถรับและส่งข้อมูลได้ทั้งสองทิศทาง (BiDirectional) ในขณะที่เลนหมายเลขอื่นๆ จะส่งข้อมูลไปยังจอภาพ LCD ได้เพียงทิศทางเดียวเท่านั้น

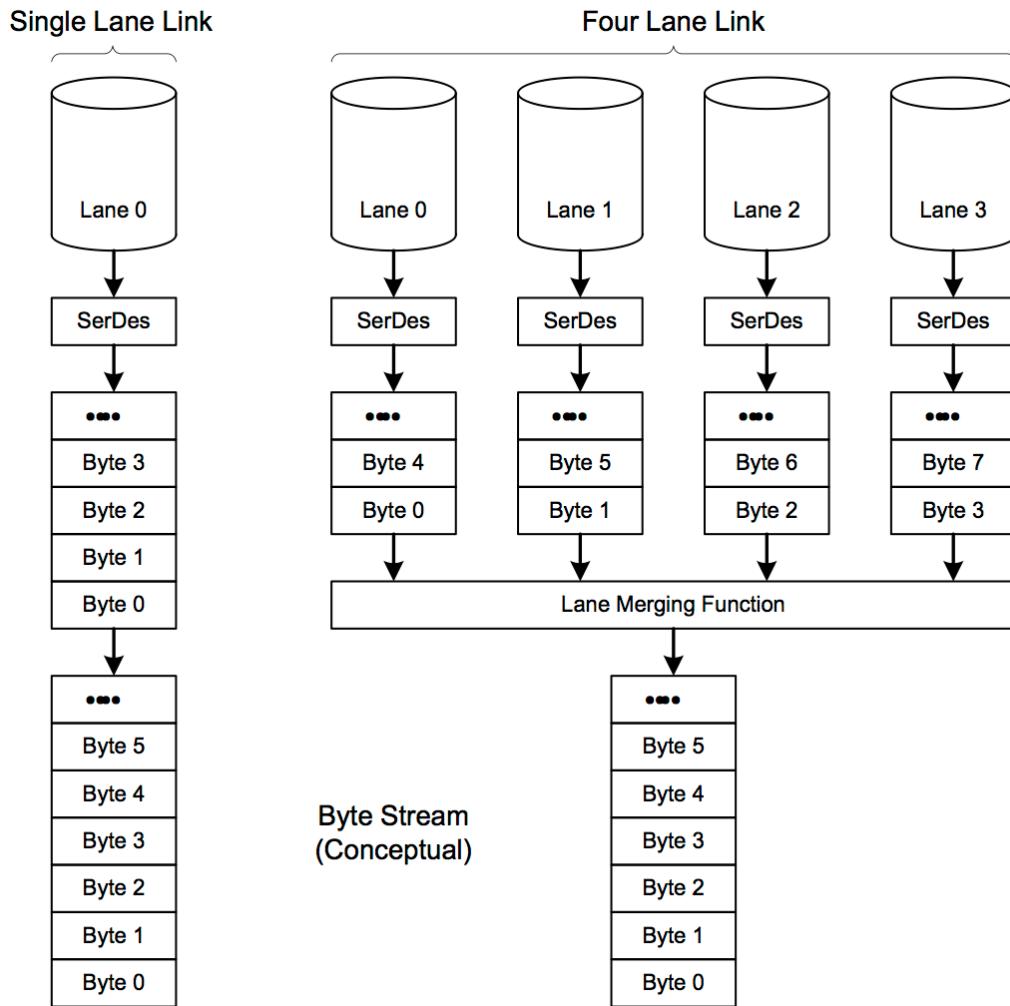


Figure 6.5: สัญญาณ DSI แบ่งเป็นชนิดหนึ่งเลน (Single Lane) และชนิด 4 เลน ที่มา: [wikipedia](#)

6.3 สัญญาณ CSI สำหรับเชื่อมต่อกล้องขนาดเล็ก



Figure 6.6: การเชื่อมต่อระหว่างบอร์ด Pi3 และกล้องขนาดเล็กด้วยอินเตอร์เฟสกล้องแบบอนุกรม (Camera Serial Interface) ที่มา: element14.com

บอร์ด Pi3 รองรับการเชื่อมต่อกล้องขนาดเล็ก ตามมาตรฐาน CSI: Camera Serial Interface ซึ่งมีความคล้ายคลึงกับสัญญาณ DSI ซึ่งกำหนดโดยองค์กรเดียวกัน คือ MIPI ข้อมูลภาพจากกล้องจะส่งผ่านสายด้วยเลขข้อมูลจำนวนหนึ่ง ตามรูปที่ 2.5 เพื่อไปรวมกันเป็นภาพเดียวที่ปลายทาง แต่ละเลนมีการส่งข้อมูลทีละบิท

กล้องจะเชื่อมกับบอร์ด Pi3 กับชีอกเก็ตหมายเลข S5 ซึ่ง เป็นชนิดยืดติดบนพื้นผิวของแผ่นวงจรพิมพ์ (Surface Mount) และไม่ต้องใช้แรงกด ZIF (Zero Insertion Force) ชนิด 15 ชา เชื่อมต่อกับสายแพ (Ribbon) โดย CSI ได้กำหนดให้สัญญาณที่ใช้สำหรับเลนข้อมูลเป็น Low Voltage Differential Signalling (SubLVDS) ซึ่งปรับปรุงมาจากมาตรฐานสัญญาณ IEEE1596.3 LVDS สำหรับอุปกรณ์ที่ใช้ไฟเลี้ยงต่ำประมาณ 1.2 โวลท์ เพื่อให้สามารถส่งข้อมูลต่อเลนได้สูงสุด 800-1,000 เมกะบิตต่อวินาที (Mbps) ตารางที่ 2.3 แสดงหมายเลขอารบิก และวัตถุประสงค์ของคอนเนคเตอร์ชนิด CSI ดังนี้

- CAM1_D0-/CAM1_D0+ และ CAM1_D1-/CAM1_D1+ คือ สัญญาณข้อมูลและบอกของเลนข้อมูลที่ 0 และ 1 ตามลำดับ โดยชิพในกล้องจะสร้างสัญญาณแล้วส่งผ่านสายแพให้กับวงจรที่รับภาพในชิพ BCM2837 ผ่านเลนข้อมูลทั้งสองเลนนี้ตามรูปที่ 2.5

ข้อมูลภาพสามารถกำหนดให้ส่งมาในรูปแบบต่าง ดังนี้ RGB (Red แดง Green เขียว และ Blue น้ำเงิน) RAW หรือข้อมูลภาพที่ไม่มีการปรับแต่งใดๆ YUV (Y: Luminance หรือ ความสว่าง U และ V คือ องค์ประกอบของสี Chrominance 2 ชนิด) เป็นต้น

Table 6.3: หมายเลข ชื่อ และวัตถุประสงค์ของคอนเน็คเตอร์ชนิด CSI [wikipedia](#)

ขา	ชื่อ	วัตถุประสงค์
1	Ground	กราวด์
2	CAM1_D0-	ขั้วลบข้อมูลภาพเลน 0
3	CAM1_D0+	ขั้วบวกข้อมูลภาพเลน 0
4	Ground	กราวด์
5	CAM1_D1-	ขั้วลบข้อมูลภาพเลน 1
6	CAM1_D1+	ขั้วบวกข้อมูลภาพเลน 1
7	Ground	กราวด์
8	CAM1_C-	ขั้วลบสัญญาณคลีอก
9	CAM1_C+	ขั้วบวกสัญญาณคลีอก
10	Ground	กราวด์
11	CAM_GPIO	ขา GPIO
12	CAM_CLK	ขาสัญญาณคลีอก
13	SCL0	สัญญาณคลีอกสำหรับ I2C
14	SDA0	สัญญาณข้อมูลสำหรับ I2C
15	+3.3 V	ไฟเลี้ยงขนาด 3.3 โวลท์

- CAM1_C- and CAM1_C+ คือ สัญญาณคลีอกขาลบและขาบวกตามลำดับ เพื่อให้การส่งข้อมูลภาพในเลนต่างๆ ที่ความเร็วสูงเป็นแบบบีจิ้งโครนัส
- SDA0 และ SCL0 SDA0 และ SCL0 คือ สัญญาณข้อมูลและคลีอกตามมาตรฐาน I2C ซึ่งผู้อ่านสามารถค้นคว้ารายละเอียดเพิ่มเติมที่ [wikipedia](#) เพื่อควบคุมการทำงานของกล้อง เช่น ความละเอียดของภาพ รูปแบบของข้อมูล เป็นต้น โดยการรับส่งจะซิงโครในซีกับสัญญาณคลีอก SCL0 ซึ่งมีความถี่ต่ำกว่าสัญญาณคลีอก CAM1_C

6.4 สัญญาณ PCM สำหรับสัญญาณเสียง

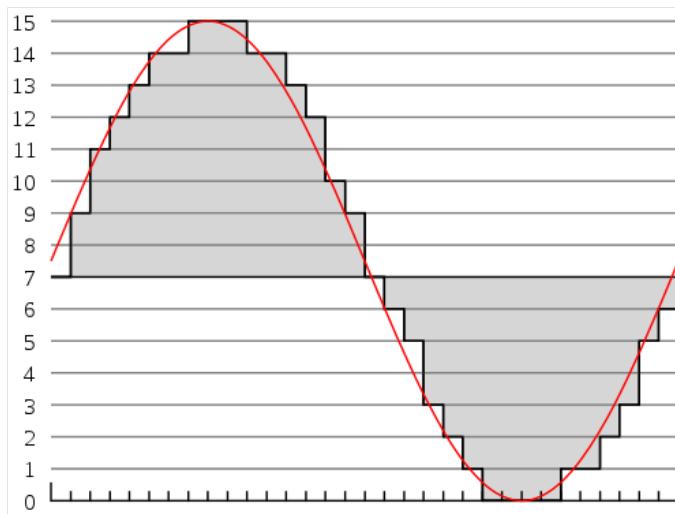


Figure 6.7: รูปคลื่นไอน์ (Sine Wave) และสัญญาณ PCM (Pulse Code Modulation) 16 ระดับ ที่มา: wolfcrow.com

สัญญาณชนิด PCM คือ สัญญาณดิจิทัลพื้นฐาน เกิดจากการแปลงสัญญาณอนาล็อกเป็นดิจิทัล (Analog to Digital: A2D) นิยมแพร่หลายในอดีตจนถึงปัจจุบัน และใช้กับแผ่นซีดี (Compact Disc) โทรศัพท์บ้านพื้นฐาน และอื่นๆ ชิป BCM2837 บนบอร์ด Pi3 สามารถแปลงข้อมูลเสียงที่ได้จากการประมวลผล ในรูปแบบ PCM และเอาท์พุตสัญญาณดิจิทัลให้เป็นสัญญาณอนาล็อกเพื่อส่งต่อให้กับลำโพงภายนอก

สัญญาณอนาล็อกรูปคลื่นไอน์จะโดนสุ่มด้วยระยะเวลาต่อเนื่องและสม่ำเสมอ ด้วยคาบเวลาที่สูงพอสมควร ขึ้นอยู่กับคุณภาพสัญญาณที่ต้องการ ยกตัวอย่าง เช่น

- ตัวอย่างที่ 1 สัญญาณเสียงสนนนาผ่านโทรศัพท์จะ สุ่มด้วยความถี่ 8,000 ครั้งต่อวินาที ซึ่งจะตรงกับคาบเวลา $1/8,000 = 125$ มิโครวินาที ด้วยระดับความดัง 256 ระดับ หรือ 8 บิต
- ตัวอย่างที่ 2 คือ สัญญาณเสียงเพลงคุณภาพระดับแผ่นซีดี จะสุ่มด้วย ความถี่ 44,100 ครั้งต่อวินาที ซึ่งจะตรงกับคาบเวลา $1/44,100 = 22.67$ มิโครวินาที ด้วยระดับความดัง 65,536 ระดับ เพื่อให้เป็นข้อมูลขนาด 16 บิตต่อการสุ่ม 1 ครั้ง ทั้งนี้ การสุ่มจะเกิดขึ้นกับช่องสัญญาณซ้ายและขวา รวมเป็นข้อมูล 32 บิตต่อการสุ่ม 1 ครั้ง
- ตัวอย่างที่ 3 รูปที่ 2.7 แสดง รูปคลื่นไอน์ (Sine Wave) และการสุ่มค่าของคลื่นไอน์นี้ ด้วยความถี่สูง เป็น 26 เท่าของความถี่เดิม และทำการแปลงเป็นสัญญาณดิจิทัลชนิด PCM (Pulse Code Modulation) จำนวน 16 ระดับให้กลับเป็นข้อมูลขนาด 4 บิตต่อการสุ่ม 1 ครั้ง

ภายในชิป BCM2837 มีโมดูล PCM ตามรูปที่ 2.8 ประกอบด้วย บัฟเฟอร์สำหรับส่ง (Transmit) และรับ (Receive) ขนาด 64×32 บิต จำนวน 2 ชุด โมดูล PCM เชื่อมอยู่บนบัส APB (ARM Peripheral Bus) เพื่อรับสัญญาณเสียงเข้า ผ่านโมดูลนี้เพื่อพักเก็บในหน่วยความจำหลักชั่วคราว เพื่อรอให้ ซีพียูและวีดีโอคอร์ประมวลผล และส่งสัญญาณเสียงไปยังลำโพง

PCM มีสายสัญญาณตามมาตรฐาน I2S (Inter IC Sound) อ่านว่า ไอสแควร์อีส จำนวน 4 เส้นเพื่อเชื่อมต่อ กับชิปภายในของ BCM2837 แบบอนุกรม ได้แก่

- PCM_CLK - สัญญาณคล็อกสำหรับส่งข้อมูลแต่ละบิตด้วยความถี่สูง
- PCM_DIN - สัญญาณข้อมูลเสียงขาเข้าหรือขารับ (Receive) จะรับสัญญาณตามความถี่ของ PCM_CLK โดยจะรับบิตสูง (Most Significant Bit) ก่อนและบิตสุดท้ายคือ บิตต่ำสุด (Least Significant Bit) เสมอ
- PCM_DOUT - สัญญาณข้อมูลเสียงขาออกหรือขาส่ง (Transmit) จะมีทิศทางที่ต่างกันข้ามกับ PCM_DIN
- PCM_FS - สัญญาณซิงค์เฟรมข้อมูล (Frame Sync) เพื่อใช้ประกาศการเริ่มต้นและสิ้นสุดเฟรม ข้อมูล โดยหนึ่งเฟรมสามารถกำหนดความยาวได้ ทั้งนี้ขึ้นอยู่กับขนาดจำนวนบิตข้อมูล และความถี่สุ่ม (Sampling Frequency)

รายละเอียดเพิ่มเติมที่ [wikipedia](#)

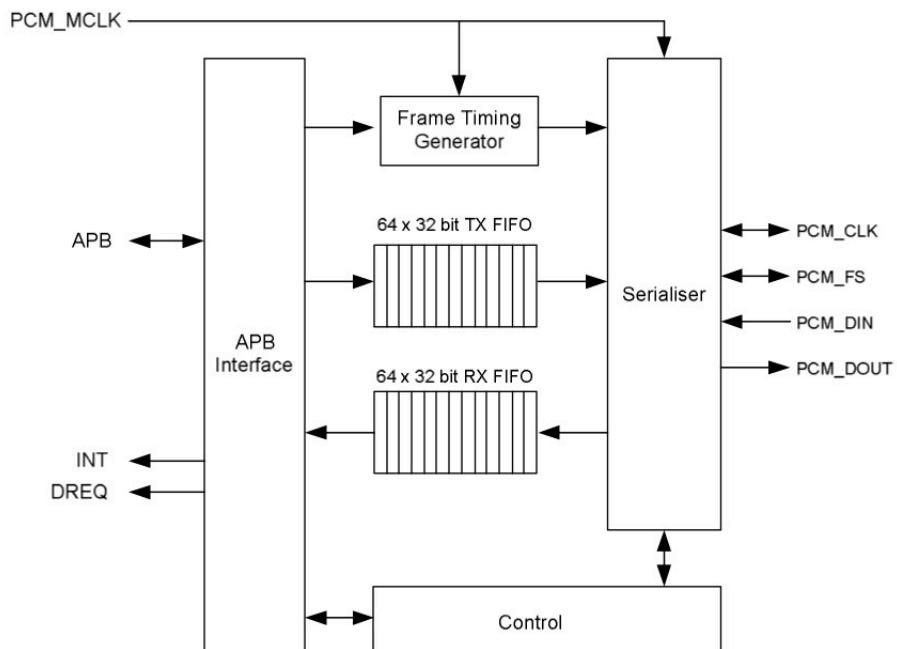


Figure 6.8: บัดฟเฟอร์สำหรับส่งและรับ ข้อมูลเสียงดิจิทัลชนิด PCM จากการเชื่อมต่อชนิด I2S ที่มา: [Broadcom \(2012\)](#)

ไม่ดูแลเรื่องรับการทำงานทั้งสามรูปแบบ คือ โพลลิ่ง (Polling) การขัดจังหวะ (Interrupt) รายละเอียดเพิ่มเติมในหัวข้อที่ [2.12](#) และ การเข้าถึงหน่วยความจำโดยตรง (Direct Memory Access) รายละเอียดเพิ่มเติมในหัวข้อที่ [2.13](#)

6.5 สัญญาณภาพและเสียงสำหรับจอทีวี



Figure 6.9: แจ็ค 3.5 มม. (กลาง) ชนิด 4 ขั้วสำหรับเสียบกับบอร์ด Pi3 (ขวา) เพื่อส่งสัญญาณภาพไปยังแจ็ค RCA (เหลือง) และสัญญาณเสียงไปยังแจ็ค RCA (แดงและขาว) ที่มา: stackexchange.com

ช่องเสียบสายในรูปที่ 2.9 ใช้เชื่อมต่อสัญญาณภาพและเสียงจากบอร์ด Pi3 เข้ากับจอทีวีหรือจอมอนิเตอร์ ที่มีช่องอินพุตเป็น คอมโพสิตวีดีโอ (Composite Video) และเสียงสเตอริโอิ หรือ โทรทัศน์บางเครื่องเรียกว่าช่อง AV In ซึ่งสามารถใช้ทดแทนจอภาพ LCD ได้ สำหรับโรงเรียนหรือผู้ใช้ที่ขาดแคลนทุนทรัพย์ แต่จะได้สัญญาณภาพที่ลักษณะต่างกว่าสัญญาณ HDMI โดยแจ็ค 3.5 มม. (กลาง) นี้เป็นชนิด 4 ขั้วสำหรับเสียบกับบอร์ด Pi3 (ขวา) เพื่อส่งสัญญาณภาพไปยังแจ็ค RCA (เหลือง) และสัญญาณเสียงไปยังแจ็ค RCA (แดงและขาว) ผู้อ่านสามารถหาซื้อสายสัญญาณสำเร็จรูปนี้ได้ทั่วไป หรือจะบัดกรีเชื่อมต่อสายเองได้เช่นกัน

สัญญาณภาพดิจิทัลจะสร้างโดยชิป BCM2837 แล้วแปลงเป็นสัญญาณภาพอนาล็อก โดยใช้ DAC (Digital to Analog Converter) ตามรูปที่ 3.2 เรียกว่าสัญญาณชื่อ TVDAC ในรูปที่ 2.1 ในขณะที่ สัญญาณเสียงดิจิทัลชนิด PCM ซึ่งได้กล่าวแล้วในหัวข้อที่ 2.4 จะถูกแปลงเป็นสัญญาณเสียงอนาล็อกโดยใช้ DAC ภายในชิป BCM2837 เช่นกัน เป็นสัญญาณเสียงซองซ้าย และซองขวาแบบสเตอริโอิ (Stereo) เพื่อผ่านฟิลเตอร์ (Filter) หรือตัวกรองความถี่ตามรูปที่ 3.2

6.6 สัญญาณ USB 2.0 สำหรับอุปกรณ์ต่อพ่วงต่างๆ

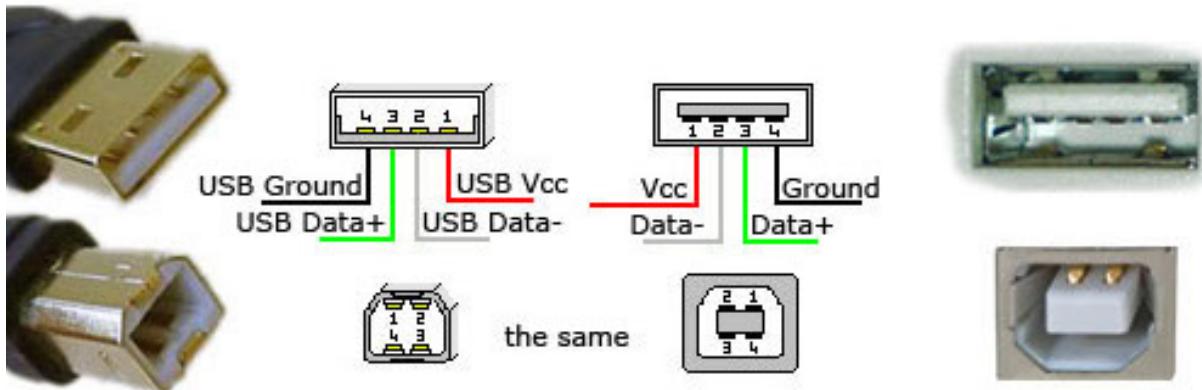


Figure 6.10: หัวเชื่อมต่อ USB ชนิด A (บน ฝั่งโฮสต์) และ B (ล่าง ฝั่งอุปกรณ์) ประกอบด้วยสัญญาณ 4 เส้น กราวด์ (GND) Data+ Data- และไฟเลี้ยง 5 โวลท์ ที่มา:

เนื้อหาในหัวข้อนี้ต่อเนื่องจากเรื่อง คีย์บอร์ด ในหัวข้อที่ 3.1.3.1 และมาส์ ในหัวข้อที่ 3.1.3.2 ซึ่งนิยมใช้เป็นแบบ USB เพื่อต่อเข้ากับบอร์ด Pi3 และคอมพิวเตอร์ทั่วไป เวอร์ชันปัจจุบันของ USB คือ 3.1 ซึ่งมีความสามารถสูงขึ้นและได้รับความนิยมเพิ่มขึ้น ในทำราลีมนี้จะกล่าวถึง USB เวอร์ชัน 2.0 ซึ่งเป็นพื้นฐานและมีคุณสมบัติ ดังนี้

- สามารถถ่ายข้อมูลทั่วไป สัญญาณเสียง และสัญญาณภาพได้สูงสุดถึง 1.5 (Low Speed) 12 (Full Speed) และ 48 (High Speed) เมกะบิตต่อวินาที
- สามารถจ่ายไฟเลี้ยงความต่างศักย์ 5 โวลท์ 0.5 แอม培ร์ให้แก่อุปกรณ์ขนาดเล็ก และสูงสุด 1 แอม培ร์สำหรับพอร์ตพิเศษ
- สายเคเบิลมีความยาวไม่เกิน 5 เมตร เนื่องจากความต้านทานของสายจะทำให้เกิดโวลตेजต่ำคร่อม (Voltage Drop) ในสาย จนทำให้ความต่างศักย์ไปเลี้ยงอุปกรณ์ไม่เพียงพอ
- ”Hot Swapping” รองรับการต่อเข้า/ออก ตลอดเวลา และรีเซ็ตอุปกรณ์ที่ต่ออยู่โดยไม่ต้องรีเซ็ตหรือรีบูตระบบโอเอส

รูปที่ 2.10 แสดง หัวเชื่อมต่อ USB ชนิด A (ฝั่งโฮสต์หรือคอมพิวเตอร์) และ B (ฝั่งอุปกรณ์) ในการเชื่อมต่อทางไฟฟ้าของ USB นั้นจะสายเคเบิลแบบ 4 คอร์ เพียง 1 เส้นต่อ 1 อุปกรณ์เท่านั้น ซึ่งมีตำแหน่งขาดังนี้

- ขา 1 เป็น 5V+ สำหรับจ่ายไฟเลี้ยงให้กับอุปกรณ์ขนาดเล็ก เช่น แฟลชไดร์ฟ กล้องเว็บแคม โทรศัพท์スマาร์ทโฟน เป็นต้น
- ขา 2 เป็น D- เป็นสายสัญญาณรับส่งข้อมูลแบบ Differential
- ขา 3 เป็น D+ เป็นสายสัญญาณรับส่งข้อมูลแบบ Differential

- ขา 4 เป็น GND เป็นขากราว์ดสำหรับไฟเลี้ยง 5 โวลท์

สายส่งข้อมูลของระบบ USB มี 2 สาย สำหรับ สัญญาณ D+ และ D- ในการส่งสัญญาณแบบ เป็นสาย สัญญาณรับส่งข้อมูลแบบ Differential คือ กรณีในการส่งสัญญาณ ”0” สายสัญญาณ D+ จะมีระดับแรงดันที่ต่ำกว่า D- อย่างน้อย 200 mV กรณีในการส่งสัญญาณ ”1” สายสัญญาณ D+ จะมีระดับแรงดันที่สูงกว่า D- อย่างน้อย 200 mV

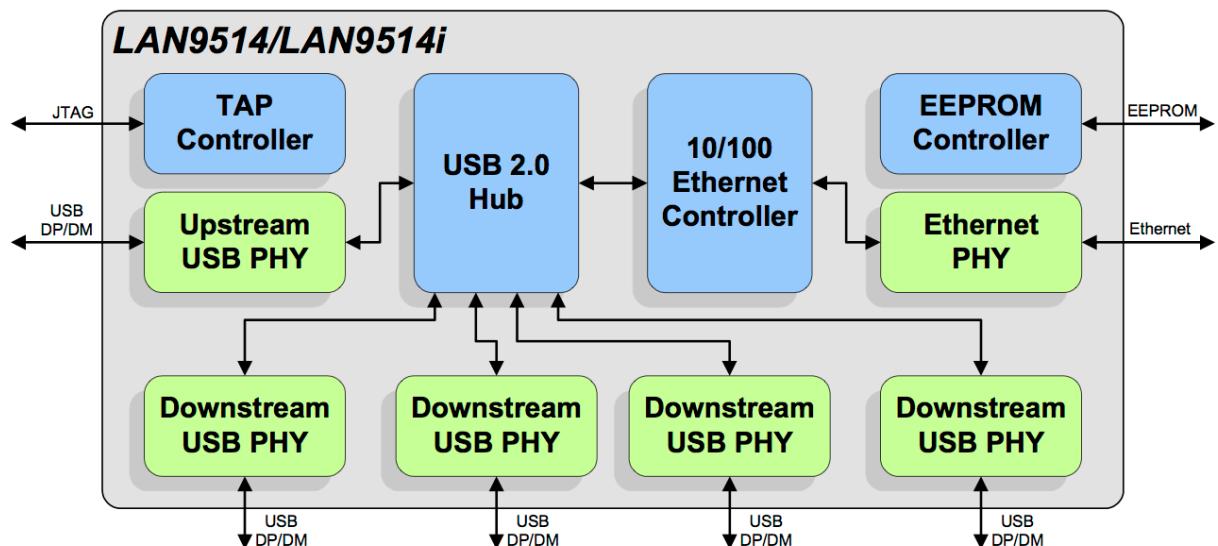


Figure 6.11: โครงสร้างของชิป LAN 9514 ภายในประกอบด้วยวงจร USB Hub และวงจร Ethernet ที่มา: [Microchip Technology \(2009\)](#)

จากรูปที่ 3.2 BCM2837 จะเชื่อมต่อกับไอซี LAN9514 เพื่อขยายเป็น 4 พอร์ต เนื่องจากภายในชิป BCM2837 จะมีรูหัสบ (Root Hub) เพียง 1 พอร์ต โครงสร้างของไอซี LAN9514 ตามรูปที่ 2.11 ถูกออกแบบให้ LAN9514 มี USB Hub (Upstream) จำนวน 1 พอร์ต เพื่อเชื่อมกับรูหัสบในชิป BCM2837 และขยายจำนวนพอร์ต (Downstream) เพิ่มเป็น 4 พอร์ต เพื่อต่อเข้ากับคีย์บอร์ด เม้าส์ และอุปกรณ์ USB อื่นๆ นอกจากนี้ ภายใน LAN9514 ยังมีโมดูล Ethernet สำหรับเชื่อมต่อกับเครือข่ายอินเตอร์เน็ต แบบใช้สาย ซึ่งจะได้กล่าวต่อไป ในทางปฏิบัติชิป LAN9514 มีพอร์ต IEEE 1149.1 TAP (Test Access Port) CONTROLLER เพื่อใช้สำหรับทดสอบวงจรภายใน

6.7 สัญญาณ Ethernet สำหรับสายเชื่อมต่อกับเครือข่าย อินเตอร์เน็ต



Figure 6.12: หัวเชื่อมต่อชนิด RJ45 (ช้ายสุด) สำหรับการเชื่อมต่อเครือข่ายท้องถิ่น (Local Area Network) แบบมีสาย Ethernet ที่มา:

อีเธอร์เน็ต (Ethernet) คือ เทคโนโลยีเครือข่าย LAN (Local Area Network) ที่นิยมใช้กันอย่างกว้างขวาง ในอดีตมานั่งปั๊จจุบัน เพราะเป็นการรับส่งข้อมูลแบบอนุกรมด้วยความเร็วสูง ใช้สายทองแดงในการติดตั้ง และต้นทุนไม่สูง เนื่องจากมีอุปกรณ์สนับสนุนเพื่อใช้งานมากที่สุด รูปที่ 2.12 แสดงตำแหน่งของพอร์ต เชื่อมต่อสาย Ethernet บริเวณมุมของบอร์ด Pi3 หัวเชื่อมต่อชนิด RJ45 แบบตัวเมีย (Female) โดยใช้สาย CAT5e ขึ้นไป

บอร์ด Pi3 นี้ รองรับการเชื่อมต่อ Ethernet ตามมาตรฐาน IEEE 802.3 ชนิด 10/100 BaseT ซึ่งเป็นที่นิยมทั้งในอดีตและปัจจุบัน ด้วยอัตรา 10/100 เมกะบิตต่อวินาที (Mbps) สายที่ใช้มีชื่อว่าสาย CAT5e หรืออาจจะใช้สายที่มาตรฐานสูงกว่าได้ เช่น CAT6 CAT6A เป็นต้น โดยจะต่อเชื่อมบอร์ดเข้ากับอุปกรณ์เครือข่าย ที่เรียกว่า Ethernet Switch ตามลำดับชั้น เพื่อสุดท้ายเชื่อมต่อกับเครือข่ายอินเตอร์เน็ต ดังนั้น การทดลองที่ 7 ภาคผนวก I การศึกษาและปรับแก้อินพุต/เอาท์พุตต่างๆ แสดงรายละเอียดของ Ethernet การตั้งค่า (Configuration) และอื่นๆ เพื่อให้บอร์ดทำหน้าที่เป็น เซิร์ฟเวอร์ (Server) เช่น เว็บเซิร์ฟเวอร์ FTP เซิร์ฟเวอร์ เป็นต้น อุปกรณ์ IoT จากการเชื่อมต่อกับเซ็นเซอร์ต่างๆ



Figure 6.13: การเข้าปลายนาย RJ45 ทั้งสองด้าน สำหรับการเชื่อมต่อระหว่างเครื่องคอมพิวเตอร์และอุปกรณ์สวิทช์ (Switch) ตามมาตรฐาน TIA T568B ที่มา:

การเชื่อมต่อเครือข่ายท้องถิ่น (Local Area Network) แบบมีสาย Ethernet รูปที่ 2.13 ซึ่งสามารถเชื่อมต่อเครือข่ายโดยใช้ช่องบันทึกการ CSMA/CD (Carrier Sense Multiple Access/Collision Detection) รายละเอียดเพิ่มเติม ผู้อ่านสามารถศึกษาเพิ่มเติมได้ที่ลิงค์ต่อไปนี้ [wikipedia](#) หรือในรายวิชาอื่นๆ เช่น การสื่อสารข้อมูล (Data Communication) เครือข่ายคอมพิวเตอร์ (Computer Network) เป็นต้น

6.8 สัญญาณ WiFi และ Bluetooth สำหรับการสื่อสารไร้สาย



Figure 6.14: รูปถ่ายและรูปขยายของชิป BCM 43438 สำหรับเชื่อมต่อเครือข่ายไร้สายท้องถิ่น (Wireless Local Area Network) หรือ WiFi และเครือข่ายไร้สายบลูทูธ (Bluetooth) ที่มา: [Corporation \(2017\)](#)

นอกเหนือจากการเชื่อมต่อแบบใช้สายแล้ว บอร์ด Pi3 ได้ถูกออกแบบให้หันสมัย และรองรับการเชื่อมต่อแบบไร้สายถึงสองชนิด คือ

- เครือข่ายไร้สายท้องถิ่น (Wireless Local Area Network) หรือ WiFi สำหรับเชื่อมต่อกับเครือข่ายอินเทอร์เน็ตแบบไร้สาย และ
- Bluetooth สำหรับเชื่อมต่อกับอุปกรณ์ระยะสั้นแบบไร้สาย เช่น โทรศัพท์เคลื่อนที่ นาฬิกา เป็นต้น

วงจรสำหรับการเชื่อมต่อแบบไร้สายทั้งสองรวมอยู่ในชิป BCM 43438 บนบอร์ด RPi3 ในรูปที่ 2.14

WiFi เป็นเทคโนโลยีการเชื่อมต่ออินเทอร์เน็ตแบบไร้สาย สำหรับอุปกรณ์ต่างๆ เช่น คอมพิวเตอร์ส่วนบุคคล เครื่องเล่นเกมส์ โทรศัพท์มาร์ทโฟน แท็บเล็ต กล้องดิจิทัลและเครื่องเสียงดิจิทัล โดยใช้คลื่นวิทยุที่ช่วงความถี่ 2.4 และ 5 จิกะเฮิร์تز อุปกรณ์ทั่วไปสามารถเชื่อมต่อกับอินเทอร์เน็ตได้ผ่านอุปกรณ์ที่เรียกว่า WiFi Router และเซสพอยต์ (Access Point) หรือ ฮอตสปอต (Hot Spot) และบริเวณที่ระยะทำการของแอคเซสพอยต์ครอบคลุมอยู่ที่ประมาณ 20 ม. ในอาคาร ถ้าเป็นที่โล่งแจ้งระยะทำการจะเพิ่มขึ้น

ชิป BCM 43438 บนบอร์ด RPi3 รองรับสัญญาณ IEEE 802.11b/g/n ที่ย่านความถี่คลื่น파ห์ 2.4 GHz เท่านั้น และสัญญาณ Bluetooth เวอร์ชัน 4.1 รวมถึงตัวรับสัญญาณวิทยุ FM Corporation (2017) บล็อกไดอะแกรมของชิป BCM 43438 ประกอบด้วยขาสัญญาณเชื่อมต่อเสาอากาศ และขาเชื่อมต่อกับไมโครคอนโทรลเลอร์ (Host Interface) ทั้งสองสัญญาณใช้สายอากาศ (Antenna) และความถี่พาร์ทหลักในย่านความถี่ 2.4 GHz จิจิเซิทซ์เดียวกัน บลูทูธใช้หลักการ Frequency Hopping และกำลังส่งที่ต่ำกว่าสัญญาณ WiFi ทำให้ไม่เกิดการรบกวนกัน

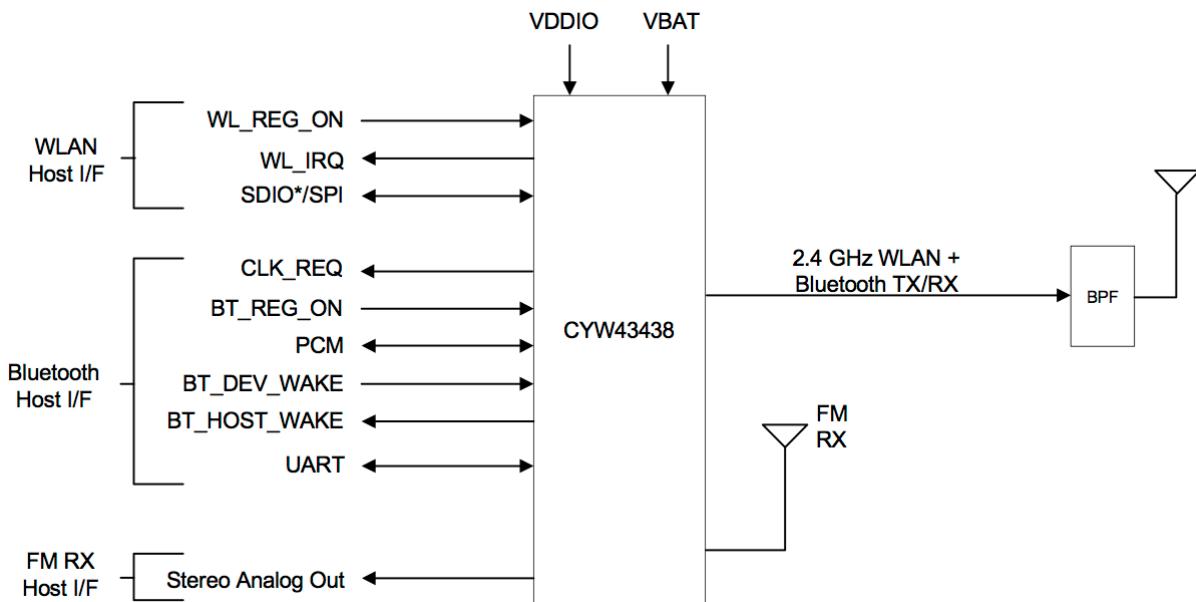


Figure 6.15: บล็อกไดอะแกรมของชิป BCM 43438 ประกอบด้วยขาสัญญาณเชื่อมต่อเสาอากาศ และขาเชื่อมต่อกับไมโครคอนโทรลเลอร์ ที่มา: Corporation (2017)

บลูทูธเชื่อมต่อกับอุปกรณ์รอบๆ ตัว เรียกว่า Personal Area Network (PAN). การเชื่อมต่อของ Bluetooth จะเป็นแบบ Master Slave โดย Master 1 ตัวจะสามารถรับ Slave ได้มากถึง 7 ตัว เรียกว่า Piconet เช่น โทรศัพท์สมาร์ทโฟน เป็น Master เชื่อมต่อกับ หูฟัง เป็น Slave หากมี Slave หลายตัวต่อ เชื่อมพร้อมกัน Master จะสื่อสารกับ Slave เหล่านั้นแบบ Round Robin นอกจากนี้ Master สามารถ Broadcast ข้อมูลไปยัง Slave ทุกด้วยตัวได้เช่นกัน

Bluetooth จะใช้ความถี่คลื่นพาร์ท ในย่าน 2.4 GHz. (จิจิเซิทซ์) จะใช้ช่วง 2.400 ถึง 2.4835 GHz. และแบ่งออกเป็น 79 ช่องสัญญาณ และจะใช้ช่องสัญญาณที่แบ่งนี้ เพื่อส่งข้อมูลสลับช่องไปมา (Frequency Hopping) 800 / 1,600 ครั้งต่อ 1 วินาที (1600 Hops/Sec) ระยะทำการของ Bluetooth ประมาณไม่เกิน 10 เมตร เป็นการป้องกันการดักสัญญาณระหว่างสื่อสาร โดยระบบจะสลับช่องสัญญาณไปมา สามารถค้นควาระละเอียดเพิ่มเติมที่ bluetooth.com

บลูทูธได้รับการออกแบบมาเพื่อใช้กับอุปกรณ์ที่มีขนาดเล็ก หรือสามารถเคลื่อนย้ายได้ง่าย เช่น โทรศัพท์เคลื่อนที่ แท็บเล็ต หูฟัง ลำโพง นาฬิกา รถยนต์ เป็นต้น เพื่อแอปพลิเคชันต่างๆ สามารถเชื่อมต่อกัน รับส่ง หรือถ่ายโอนไฟล์ภาพ, เสียง, ข้อมูล โดยการใช้งานบลูทูธจะต้องมีการ Pair up หรือจับคู่ เป็นกลไกรักษาความปลอดภัย โดยผู้ใช้อุปกรณ์บลูทูธทั้งสองฝ่ายจะต้องป้อนรหัสเดียวกันก่อนการเชื่อมต่อ โดยอาศัยโปรไฟล์ (Profile) สำคัญที่มีในอุปกรณ์ทั้งสอง เช่น

- Human Interface Device Profile (HID) สำหรับเชื่อมต่ออุปกรณ์ เช่น เม้าส์ คีย์บอร์ด ของเครื่องคอมพิวเตอร์
- Dial-up Networking Profile (DUN) สำหรับใช้ในการตั้งค่าเครือข่ายเครื่องคอมพิวเตอร์ เชื่อมต่อกับอินเทอร์เน็ตผ่านโทรศัพท์เคลื่อนที่
- Advanced Audio Distribution Profile (A2DP) สำหรับเชื่อมต่อหูฟังชนิดบลูทูธ

รายละเอียดเพิ่มเติม ผู้อ่านสามารถค้นคว้าได้ที่ [Wikipedia](#) การทดลองเพื่อแสดงรายละเอียดของ WiFi และ Bluetooth ในการทดลองที่ 7 ภาคผนวกที่ |

6.9 หลักการ Memory Mapped Input/Output

โปรแกรมต่างๆ อ้างอิงกับหน่วยความจำเสมือนเท่านั้นภายใต้ระบบปฏิบัติการลีนักซ์ โปรแกรมเหล่านี้จะไม่สามารถเข้าถึงแอดเดรสภายนอกโดยตรงได้เลย และจะเข้าถึงอุปกรณ์อินพุตเอาท์พุตผ่าน System Call Interface ของระบบปฏิบัติการเท่านั้น เช่น พิมพ์ชัน printf ในไฟล์ stdio.h เพื่อแสดงผลข้อมูลต่างๆ บนหน้าจอแสดงผล ในเชิงโครงสร้าง ระบบหน่วยความจำเสมือนเชื่อมโยงกับอินพุตเอาท์พุต ผ่านทาง MMU (Memory Management Unit) ตัวที่ 1 (ARM MMU) จะแปลงแอดเดรสเสมือนให้กลายเป็นแอดเดรสภายนอก ตามรูปที่ 5.3 และ MMU ตัวที่ 2 (VC/ARM MMU) จะแปลงแอดเดรสภายนอกให้กลายเป็นแอดเดรสบัสที่เข้มซึ่งกับหน่วยความจำและอุปกรณ์อินพุต/เอาท์พุตต่างๆ ตามรูปที่ 2.16

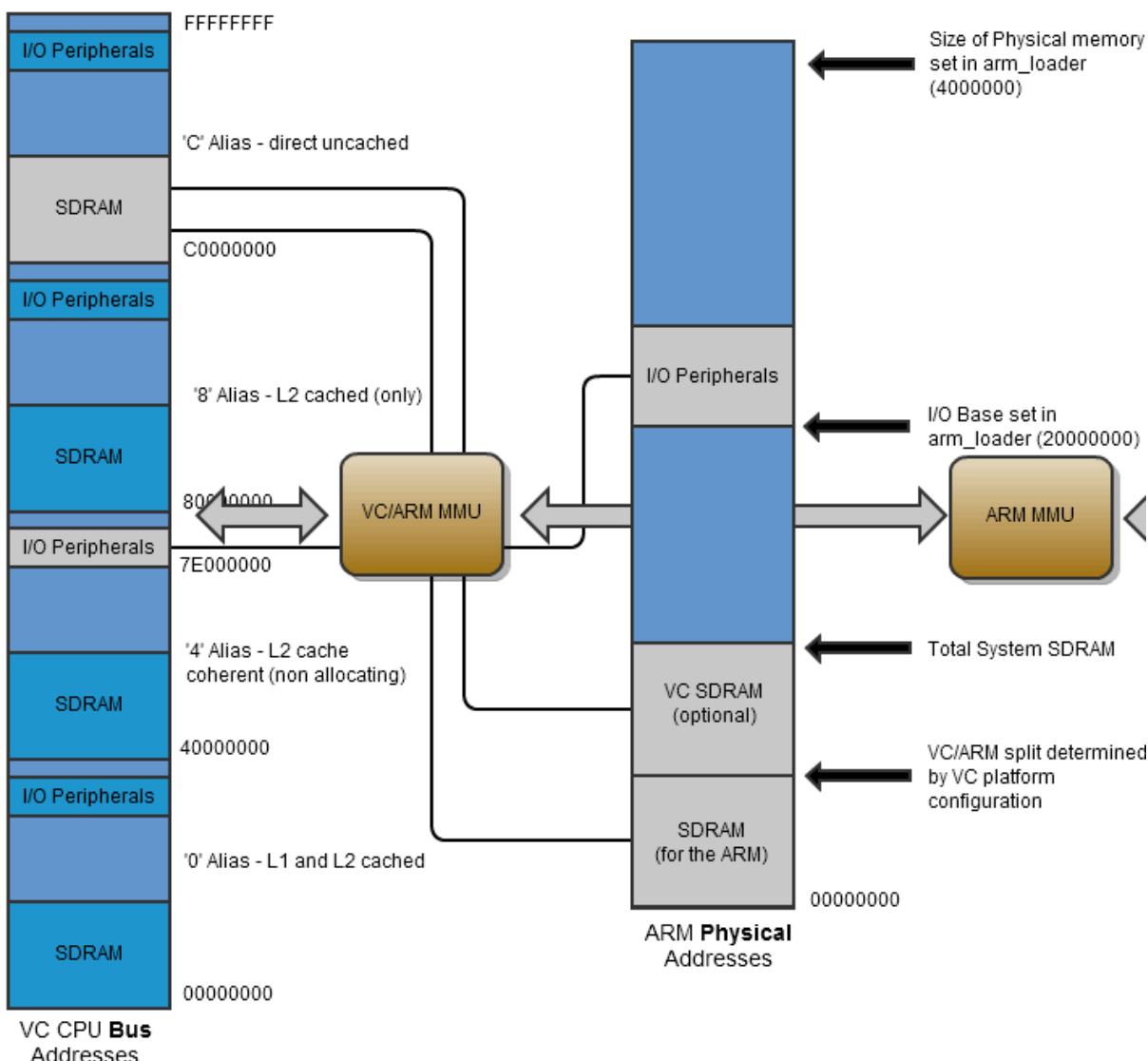


Figure 6.16: การแมปหน่วยความจำระหว่างแอดเดรสบัส (VC ชีพิชชูBus Address) และแอดเดรสภายนอกของ ARM ที่มา: [Broadcom \(2012\)](#) หมายเหตุ VC: Video Core

ในเชิงของฮาร์ดแวร์ TLB และตารางเพจภายในตัว ARM MMU ทำหน้าที่แปลงแอดเดรสเสมือนเป็นแอดเดรสภายนอก หลังจากนั้น VC (Video Core)/ARM MMU จึงแปลงแอดเดรสภายนอกเป็นแอดเดรสบัส (VC

ชีพียู Bus Address) ซึ่งเชื่อมต่อกับชีพียู วิดีโโคอร์ อุปกรณ์อินพุต/เอาท์พุต และหน่วยความจำ โดยในรูปที่ 2.16 แสดงเดรสเกยภาพ 0x20000000 หรือ 0x2000_0000 ถูกใช้ตั้งค่าตัวแปร I/O Base ในไฟล์ ARM Loader จะแปลงเป็น แอดเดรสบัส (Bus Address) 0x7E000000 หรือ 0x7E00_0000

ภายในชิป BCM283x มีวงจรหรืออุปกรณ์/อินพุตเอาท์พุต (I/O Peripherals) มีจำนวนมาก การเชื่อมต่อกับอุปกรณ์เหล่านี้จะใช้การตั้งค่าแอดเดรสบัส (Bus Address) เพื่ออ่านข้อมูลและเขียนข้อมูลเหล่านี้แล้วแมพ (Map) แอดเดรสบัสเหล่านี้ไปบน แอดเดรสหน่วยความจำภายในภาพ การอ่านหรือเขียนข้อมูลไปยังแอดเดรสเกยภาพเหล่านี้ ทำได้การใช้คำสั่ง LDR และ STR เมื่ອนกับหน่วยความจำปกติทั่วไป โดยผู้ผลิตหารดแวร์ได้กำหนดหมายเลขแอดเดรสของบัสตามตารางที่ 2.4 ต่อไปนี้ โดยแอดเดรสบัสเริ่มต้นที่หมายเลข 0x7E000000 หรือ 0x7E00_0000

Table 6.4: ตารางแอดเดรสบัสเริ่มต้นที่หมายเลข 0x7E00_0000 สำหรับอุปกรณ์อินพุตเอาท์พุต (I/O Peripherals) และหัวข้อ ที่มา: [Broadcom \(2012\)](#)

แอดเดรสบัส (Bus Address)	ชื่อ (Name)	รายละเอียด (Details)	หัวข้อ (Section)
0x7E00_0000	
0x7E00_1000	
0x7E00_2000	
0x7E00_3000	System Timer	วงจรจับเวลาสำหรับระบบปฏิบัติการ	-
0x7E00_7000	DMA Controller	การเข้าถึงหน่วยความจำโดยตรง	2.13
0x7E00_B000	Interrupt Register	การอินเตอร์รัฟท์	2.12
0x7E00_B400	Timer	วงจรจับเวลาสำหรับการใช้งานทั่วไป	
0x7E20_0000	General Purpose I/O	ขา GPIO ทั้งหมด	2.11
0x7E20_3000	Pulse Code Modulation	สัญญาณเสียง	2.4
0x7E21_5000	mini UART, SPI1, SPI2	การสื่อสารแบบอนุกรม	-
0x7E30_0000	External Mass Media Controller	อุปกรณ์สำรองข้อมูลภายนอก	7.3
0x7E98_0000	Universal Serial Bus	USB	2.6

นอกจาก I/O Peripherals แล้ว หน่วยความจำ SDRAM ณ แอดเดรสเกยภาพ 0x0000_0000 จะแมพเป็นแอดเดรสบัสที่ตำแหน่ง 0xC000_0000 ในพื้นที่สีเทา โดยจะมีตัวแปร VC/ARM Split เป็นตัวกำหนดขนาดหน่วยความจำที่จะแบ่งไว้ให้กับ GPU ซึ่งจะได้ทดลองเปลี่ยนแปลงค่าในการทดลองที่ 7 ภาคผนวก |

6.10 หัวเชื่อมต่อ 40 ขา (40-Pin Header)

เนื่องจากบอร์ดตระกูลนี้ถูกออกแบบให้มีราคาถูกและมีความหลากหลาย สามารถใช้ประกอบการเรียนการสอนวิชาต่างๆ ด้านคอมพิวเตอร์ได้ ผู้ออกแบบจึงเชื่อมต่อขาบางส่วนของชิป BCM283x มาให้ผู้อ่านได้เรียนรู้ ทางหัวเชื่อมต่อจำนวน 40 ขาในรูปที่ 3.2 ซึ่งประกอบด้วย ส่วนต่างๆ เหล่านี้

- ไฟเลี้ยงชนิด 3.3 โวลท์, 5.0 โวลท์ และกราวด์ (GND) สำหรับเลี้ยงวงจรภายนอกที่กินกระแสหนาอย่างมาก
- ขาเชื่อมต่อชนิด GPIO General Purpose *input/output* ได้แก่ ขา GPIO00-GPIO27 สำหรับใช้เชื่อมต่อกับวงจรที่ต้องการทดลอง
- ขาสำหรับการสื่อสารชนิด UART ได้แก่ ขา RXD0/TXD0, ขา RXD1/TXD1 สำหรับเชื่อมต่อกับบอร์ดอื่นๆ ผ่านทางพอร์ต UART ซึ่งมีอัตราบิตร率สูงสุด 115,000 บิตต่อวินาที
- ขาสำหรับการเชื่อมต่อกับชิปภายนอกชนิด SPI ได้แก่ ขา SPI1_CE0, SPI1_CE1, SPI1_CE2, SPI1_MISO, SPI1_MOSI, SPI1_SCLK สำหรับเชื่อมต่อกับบอร์ดอื่นๆ ผ่านทางพอร์ต SPI ซึ่งมีอัตราบิตร率สูงกว่า 115,000 บิตต่อวินาที
- ขาสำหรับการเชื่อมต่อสัญญาณเสียง ได้แก่ ขา PCM_DIN, PCM_DOUT, PCM_FS, PCM_CLK ตามมาตรฐาน I2S ในหัวข้อที่ 2.4
- ขาสำหรับการเชื่อมต่อกับอุปกรณ์ภายนอกชนิด PWM และ PPM ได้แก่ ขา PWM0, PWM1 สำหรับการทดลอง เช่น การควบคุมการหมุนของมอเตอร์ไฟฟ้า การควบคุมความสว่างของหลอดไฟ เป็นต้น
- ขาสำหรับการเชื่อมต่อกับชิปภายนอกตามมาตรฐาน I2C ได้แก่ ขา SDA0, SCL0, SDA1, SCL1
- ขาสำหรับการเชื่อมต่อกับ SD Card ภายนอก ได้แก่ ขา SD0_CMD, SD0_CLK, SD0_DAT0, SD0_DAT0-SD0_DAT3

ขาสัญญาณเหล่านี้สามารถเขียนโปรแกรมควบคุมสั่งการด้วยภาษาแอสเซมบลี C และ Python ในการทดลองต่างๆ

เนื่องจากชิป BCM2837 มีฟุตพรินท์ (Foot Print) ขนาดเล็กทำให้จำนวนขา (Leads) จำกัด เพียง 200 ขา ที่มา: [Ltd \(2019\)](#) จึงต้องใช้การมัลติเพล็กซ์ (Multiplex) หรือเลือก (Select) สัญญาณที่ต้องการเพียงสัญญาณเดียวมาที่ขา โดยโปรแกรมเมอร์จะต้องกำหนดว่าหมายเลขนี้มัลติเพล็กซ์เป็นสัญญาณอะไร ทำให้ขาเดียวกันมีสัญญาณหลายชื่อ ตามที่ได้สรุปในตารางที่ 2.5

Table 6.5: หมายเลขขา ชื่อขา ตัวเลือกที่ 0-5 ของหัวเชื่อมต่อสายทั้ง 40 ขา (GND: Ground) ที่มา: Broadcom (2012)

ขา	ชื่อ (ตัวเลือก0)	ตัวเลือก1	ตัวเลือก2	ตัวเลือก3	ตัวเลือก4	ตัวเลือก5
1	3.3V					
2	5.0V					
3	GPIO02	SDA1	SA3			
4	5.0V					
5	GPIO03	SCL1	SA2			
6	GND					
7	GPIO04	GPCLK0	SA1			
8	GPIO14	TxD0	SD6	TXD1		
9	GND					
10	GPIO15	RxD0	SD7			RXD1
11	GPIO17		SD9	RTS0	SPI1_CE1	RTS1
12	GPIO18	PCM_CLK	SD10	SDA	SPI1_CE0	PWM0
13	GPIO27				SD0_DAT3	ARM_TMS
14	GND					
15	GPIO22	SD14		SD1_CLK	ARM_TRST	
16	GPIO23		SD15	SD0_CMD	ARM_RTCK	
17	3.3V					
18	GPIO24		SD16	SD0_DAT0	ARM_TDO	
19	GPIO10	SPI0_MOSI	SD2			
20	GND					
21	GPIO09	SPI0_MISO	SD1			
22	GPIO25		SD17	SD0_DAT1	ARM_TCK	
23	GPIO11	SPI0_CLK	SD3			
24	GPIO08	SPI0_CE0	SD0			
25	GND					
26	GPIO07	SPI0_CE1_N	SWE_N			
27	GPIO00	SDA0	SA5			
28	GPIO01	SCL0	SA4			
29	GPIO05	GPCLK1	SA0			
30	GND					
31	GPIO06	GPCLK2	SOE_N			
32	GPIO12	PWM0	SD4			
33	GPIO13	PWM1	SD5			
34	GND					
35	GPIO19	PCM_FS	SD11	SCL	SPI1_MISO	PWM1
36	GPIO16		SD8	CTS0	SPI1_CE2	CTS1
37	GPIO26			SD1_DAT2	ARM_TDI	
38	GPIO20	PCM_DIN	SD12	SPI1_MOSI	GPCLK0	
39	GND					
40	GPIO21	PCM_DOUT	SD13	CE_N	SPI1_SCLK	GPCLK1

6.11 ขา GPIO (General Purpose Input Output)

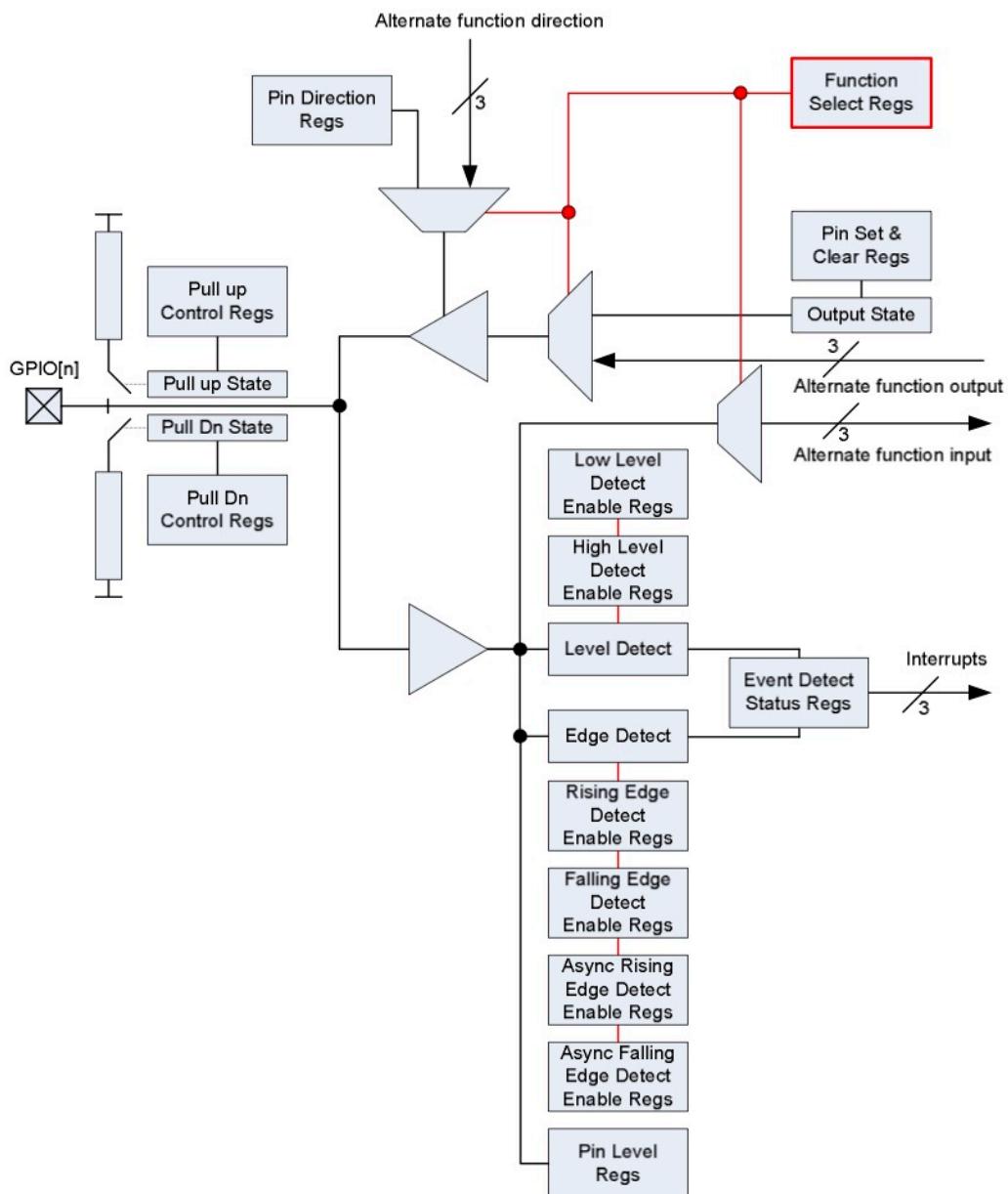


Figure 6.17: โครงสร้างภายในขา GPIO สำหรับชิปตรรกะเดียวกับ BCM2835 ที่มา: [Broadcom \(2012\)](#)

ขา GPIO ทุกขาหั้งหมด 54 ขา มีโครงสร้างภายในเหมือนกันหมดตามรูปที่ 2.17 ขา GPIO[n] คือ ขาที่ $n = 0$ ถึง $n = 53$ สำหรับชิปตรรกะ BCM283x ผู้อ่านสามารถทำความเข้าใจในรูปจากซ้ายไปขวา โดยจะเห็นว่าประกอบด้วย

- ตัวต้านทานสองตัว ซึ่งทำหน้าที่ Pull Up และ Pull Down การพูลอัพหรือพูลดาวน์ กำหนดค่าในรีจิสเตอร์พูลอัพ (Pull Up Reg) หรือพูลดาวน์ (Pull Down Reg) ผู้ใช้สามารถใช้งาน ตัวต้านทานภายในเพื่อพูลอัพ (Pull up) กับไฟเลี้ยง (Vdd) 3.3 โวลท์ หรือ พูลดาวน์ (Pull Down) ขากับกราวด์ โดยรีจิสเตอร์ GPPUD ขนาด 2 บิต เพื่อตั้งค่าของ รีจิสเตอร์ Pull Up Control และ รีจิสเตอร์ Pull Down Control

- Pin Direction Reg เพื่อกำหนดทิศทาง เข้า/ออก (Direction = Input) หรือ ขาออก (Direction = Output) หรือ Direction = Alternate นั่นคือ การสลับระหว่าง เข้า และ ออก ตามการควบคุมด้วย รีจิสเตอร์ Alternate Function Direction
 - วงจรผู้ส่งขาออก (Output) ซึ่งอยู่ด้านบนของรูป ประกอบด้วย รีจิสเตอร์ Pin Set & Clear ซึ่งเก็บค่าของเอาท์พุทไว้ โปรแกรมเมอร์สามารถตั้งค่ารีจิสเตอร์ตัวนี้ให้มีค่าลอจิค 1 สัญลักษณ์สี่เหลี่ยมคงที่ แทนอุปกรณ์มัลติเพล็กเซอร์ ทำงานที่เลือกสัญญาณเอาท์พุทจากอินพุทจำนวนหลายเส้น สัญลักษณ์สามเหลี่ยม แทนวงจรบัฟเฟอร์ซึ่งล็อกอิจิก O/p = i/p แต่ทำงานที่เพิ่มกระแสให้กับผู้ส่งเอาท์พุทที่เป็นด้านแหลม
 - วงจรผู้ส่งขาเข้า (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) เพื่อเก็บค่าลอจิคที่ได้จากการตรวจจับสัญญาณว่าตรวจจับเหตุการณ์ได้หรือไม่
 - รีจิสเตอร์ Pin Level เก็บค่าระดับสัญญาณที่รับได้ โดยใช้การสัมมตากับสัญญาณคลื่นอิเล็กทรอนิกส์

ผู้อ่านสามารถเรียนรู้การใช้งานขา GPIO เหล่านี้ในการทดลองที่ 8 และ 9 ตามลำดับ

BCM2837 เป็นล็อกิจิกชนิด CMOS ล็อกิก 1 ของขาทั้งหมดใช้ความต่างศักย์ 3.3 โวลท์เท่านั้น ในขณะที่ไฟเลี้ยง 5.0 โวลท์ใช้สำหรับจ่ายไฟให้กับซีพีอินๆ บางตัวที่เป็นล็อกิก CMOS นักพัฒนาจะต้องระวัง กรณี เชื่อมต่อวงจรทั้งสองชนิดเข้าด้วยกัน การใช้ขาทำงานในลักษณะ GPIO โปรแกรมเมอร์และนักพัฒนาจะ ต้อง เชื่อมต่อวงจรและเขียนโปรแกรมควบคุมทิศทางการไหลของสัญญาณว่า เป็น อินพุท หรือ เอาท์พุท ตามวงจรที่ต้องไว้ให้ถูกต้อง มิเช่นนั้นอาจทำให้ ชิป BCM2837 เสียหาย โดยเฉพาะการต่อไฟเลี้ยง 3.3 และ 5.0 โวลท์ กับกราวด์โดยไม่ได้ตั้งใจ

ตารางที่ 2.6 คือ รายละเอียดของแอดเดรสหมายเลข 0x7E20 0000 ในตารางที่ 2.4

Table 6.6: ตารางแอดเดรสในหน่วยความจำเริ่มต้นที่หมายเลข 0x7E20_0000 สำหรับ GPIO ที่มา: Broadcom (2012)

แอดเดรสบัส	รีจิสเตอร์	รายละเอียด	จำนวนบิต	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_0014	GPFSEL5	GPIO Function Select 5	12	R/W	50-53
0x7E20_001C	GPSET0	GPIO Pin Output Set 0	32	W	0-31
0x7E20_0020	GPSET1	GPIO Pin Output Set 1	22	W	32-53
0x7E20_0028	GPCLR0	GPIO Pin Output Clear 0	32	W	0-31
0x7E20_002C	GPCLR1	GPIO Pin Output Clear 1	22	W	32-53
0x7E20_0034	GPLEV0	GPIO Pin Level 0	32	R	0-31
0x7E20_0038	GPLEV1	GPIO Pin Level 1	22	R	32-53
0x7E20_0040	GPEDS0	GPIO Pin Event Detect Status 0	32	R/W	0-31
0x7E20_0044	GPEDS1	GPIO Pin Event Detect Status 1	22	R/W	32-53
0x7E20_004C	GPREN0	GPIO Pin Rising Edge Detect Enable 0	32	R/W	0-31
0x7E20_0050	GPREN1	GPIO Pin Rising Edge Detect Enable 1	22	R/W	32-53
0x7E20_0058	GPFEN0	GPIO Pin Falling Edge Detect Enable 0	32	R/W	0-31
0x7E20_005C	GPFEN1	GPIO Pin Falling Edge Detect Enable 1	22	R/W	32-53
0x7E20_0064	GPHEN0	GPIO Pin High Detect Enable 0	32	R/W	0-31
0x7E20_0068	GPHEN1	GPIO Pin High Detect Enable 1	22	R/W	32-53
0x7E20_0070	GPLENO	GPIO Pin Low Detect Enable 0	32	R/W	0-31
0x7E20_0074	GPLEN1	GPIO Pin Low Detect Enable 1	22	R/W	32-53
0x7E20_007C	GPAREN0	GPIO Pin Async. Rising Edge Detect 0	32	R/W	0-31
0x7E20_0080	GPAREN1	GPIO Pin Async. Rising Edge Detect 1	22	R/W	32-53
0x7E20_007C	GPAFEN0	GPIO Pin Async. Falling Edge Detect 0	32	R/W	0-31
0x7E20_0080	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 Function Select 0 เพื่อควบคุมการทำงานของขาที่ 0 - ขาที่ 9 ทั้งหมด 10 ขา ละ 3 บิตเท่ากับใช้ข้อมูลเพียง 30 บิต โดยขาที่ 0 ใช้งานบิตที่ 0-2 ตามลำดับจนถึงขาที่ 9 ใช้งานบิตที่ 28-30
 - การเขียนข้อมูลที่แอดเดรส 0x7E20_0004 - 0x7E20_0007 รวม 4 ไบต์ หรือ 32 บิต เท่ากับ เป็นการตั้งค่าของรีจิสเตอร์ Function Select 1 เพื่อควบคุมการทำงานของขาที่ 10 - ขาที่ 19 ทั้งหมด 10 ขา ละ 3 บิตเท่ากับใช้ข้อมูลเพียง 30 บิต โดยขาที่ 10 ใช้งานบิตที่ 0-2 ตามลำดับจนถึงขาที่ 19 ใช้งานบิตที่ 28-30
 - การเขียนข้อมูลที่แอดเดรส 0x7E20_0014 - 0x7E20_0015 รวม 2 ไบต์ หรือ 16 บิต เท่ากับ เป็นการตั้งค่าของรีจิสเตอร์ Function Select 5 เพื่อควบคุมการทำงานของขาที่ 50 - ขาที่ 53 ทั้งหมด 4 ขา ละ 2 บิตเท่ากับใช้ข้อมูลเพียง 8 บิต โดยขาที่ 50 ใช้งานบิตที่ 0-1 ตามลำดับจนถึงขาที่ 53 ใช้งานบิตที่ 28-29

53 ทั้งหมด 4 ขาฯ ละ 3 บิตเท่ากับใช้ข้อมูลเพียง 12 บิต โดยขาที่ 50 ใช้งานบิตที่ 0-2 ตามลำดับจนถึงขาที่ 53 ใช้งานบิตที่ 9 - 11

การใช้ข้อมูลจำนวน 3 บิตเพื่อเลือกการทำงานของ GPIO แต่ละชานั้น ใช้วิธีการกำหนดบิตทั้ง 3 ดังนี้

- 000 = ขาเข้า (Input)
- 001 = ขาออก (Output)
- 100 = ตัวเลือก 0 ในตารางที่ [2.5](#)
- 101 = ตัวเลือก 1 ในตารางที่ [2.5](#)
- 110 = ตัวเลือก 2 ในตารางที่ [2.5](#)
- 111 = ตัวเลือก 3 ในตารางที่ [2.5](#)
- 011 = ตัวเลือก 4 ในตารางที่ [2.5](#)
- 010 = ตัวเลือก 5 ในตารางที่ [2.5](#)
- การตั้งขาเป็นขาเอาท์พุต (000)
 - รีจิสเตอร์ GPSET0 (GPIO Pin Output Set) ใช้สำหรับตั้งค่าเอาท์พุตเป็นลอจิค 1
 - * แอดเดรส 0x7E20_001C - 0x7E20_001F จำนวน 4 ไบท์ หรือ 32 บิต ควบคุมขาที่ 0 - 31 จำนวน 32 ขาฯ ละ 1 บิต ขาที่ 0 ควบคุมโดยบิตที่ 0 จนถึงขาที่ 31 ควบคุมโดยบิตที่ 31 ตามลำดับ
 - * แอดเดรส 0x7E20_0020 - 0x7E20_0022 จำนวน 3 ไบท์ หรือ 24 บิต ควบคุมขาที่ 32 - 53 จำนวน 22 ขาฯ ละ 1 บิต ขาที่ 32 ควบคุมโดยบิตที่ 0 จนถึงขาที่ 53 ควบคุมโดยบิตที่ 21 ตามลำดับ
 - รีจิสเตอร์ GPCLR0 (GPIO Pin Output Clear) ใช้สำหรับตั้งค่าเอาท์พุตเป็นลอจิค 0
 - * แอดเดรส 0x7E20_0028 - 0x7E20_002B จำนวน 4 ไบท์ หรือ 32 บิต ควบคุมขาที่ 0 - 31 จำนวน 32 ขาฯ ละ 1 บิต ขาที่ 0 ควบคุมโดยบิตที่ 0 จนถึงขาที่ 31 ควบคุมโดยบิตที่ 31 ตามลำดับ
 - * แอดเดรส 0x7E20_002C - 0x7E20_002e จำนวน 3 ไบท์ หรือ 24 บิต ควบคุมขาที่ 32 - 53 จำนวน 22 ขาฯ ละ 1 บิต ขาที่ 32 ควบคุมโดยบิตที่ 0 จนถึงขาที่ 53 ควบคุมโดยบิตที่ 21 ตามลำดับ
- การตั้งขาเป็นขาอินพุต (001) จะมีความซับซ้อนมากกว่าเป็นขาเอาท์พุต โดยโปรแกรมเมอร์จะต้องรีจิสเตอร์ว่าเป็นชนิดตรวจจับระดับสัญญาณ (Level Detect) หรือ ชนิดตรวจจับขอบ (Edge Detect) รีจิสเตอร์ GPEDS event detection เกิดจากการตรวจจับระดับโวลต์เจ หรือ เกิดจากการตรวจจับขอบสัญญาณ

- การตรวจจับระดับสัญญาณ (Level Detect) แบ่งเป็น สัญญาณต่ำ หรือ โลจิก 0 (Low Level Detect) สัญญาณสูง หรือ โลจิก 1 (High Level Detect) โดยการตั้งค่าในรีจิสเตอร์ GPLEN (GPIO Pin Low Detect Enable) และรีจิสเตอร์ GPHEN (GPIO Pin High Detect Enable) ตัวใดตัวหนึ่งให้เท่ากับโลจิก 1 ที่เหลือเป็น 0 รีจิสเตอร์ Pin Level (GPLEV) เพื่อบันทึกค่าอินพุตที่รับเข้ามา
- การตรวจจับขอบสัญญาณ (Edge Detection) แบ่งเป็น
 - * ขอบขึ้น (Rising) และขอบขาลง (Falling) เมื่อตรวจจับได้ จะส่งสัญญาณอินเทอร์รัพท์ เข้าสู่ซีพียูต่อไป
 - * การตรวจจับสัญญาณแบ่งเป็น ชนิดซิงโครนัส (Synchronous) และ อะซิงโครนัส (Asynchronous) ความหมายคือ ซิงโครไนซ์ (Synchronize) กับสัญญาณคล็อกหรือไม่
 - . ชนิดซิงโครนัส การตรวจจับจะขึ้นกับความถี่ของสัญญาณคล็อก เหมาะกับเหตุการณ์ที่เกิดขึ้นบ่อย
 - . ชนิดอะซิงโครนัส การตรวจจับจะไม่ขึ้นกับความถี่ของสัญญาณคล็อก เหมาะกับเหตุการณ์ที่นานๆ จะเกิดขึ้น

โดยการควบคุมที่รีจิสเตอร์ทั้งสี่ชนิดนี้ ได้แก่

- . รีจิสเตอร์ GPREN (GPIO Synchronous Rising Edge Enable) และ
- . รีจิสเตอร์ GPFEN (GPIO Synchronous Falling Edge Enable)
- . รีจิสเตอร์ GPAREN (GPIO Asynchronous Rising Edge Enable) และ
- . รีจิสเตอร์ GPAFEN (GPIO Asynchronous Falling Edge Enable)

เกิดเป็นสัญญาณอินเทอร์รัพท์ร้องขอไปยังซีพียู รีจิสเตอร์ Pin Event Detect Status (GPEDS) จะเก็บค่าโลจิก 1 เมื่อมีการตรวจจับเหตุการณ์ที่รับเข้ามาได้ตามเงื่อนไขที่ต้องการ และจะถูกเปลี่ยนเป็นโลจิก 0 เมื่อซีพียูตอบสนองต่อการร้องขอเรียบร้อยแล้ว รายละเอียดเพิ่มเติมในหัวข้อที่ [2.12](#)

ผู้อ่านสามารถใช้งานขา GPIO เหล่านี้ใน การทดลองที่ 8 ภาคผนวก [J](#) โดยพัฒนาโปรแกรมประยุกต์ ด้วยภาษา C และแอสเซมบลีทำงานร่วมกับไลบรารี wiringPi และกิจกรรมท้ายการทดลองจะช่วยเสริมให้ผู้อ่านเข้าใจการตั้งค่ารีจิสเตอร์เหล่านี้ด้วย

6.12 การขัดจังหวะ (Interrupt)

การขัดจังหวะสำหรับ ARM Cortex A รุ่นต่างๆ อาศัยวงจรควบคุม เรียกว่า Generic Interrupt Controller (GIC) เพื่อรับการขัดจังหวะที่ซับซ้อน เนื่องจากชิปมีจำนวน ARM Cortex A ตั้งแต่ 2 คอร์ ขึ้นไป และอุปกรณ์ที่หลากหลาย โดยทั้งหมดเชื่อมต่อกันด้วย ARM Advanced eXtensible Interface (AXI) ซึ่งเป็นวงจรชนิดหนึ่ง มีโครงสร้างและโปรโตคอลที่ซับซ้อน ผู้อ่านสามารถศึกษารายละเอียดและวิวัฒนาการของ AXI เพิ่มเติมที่ [wikipedia](#)

แต่หัวข้อนี้จะอธิบายโครงสร้างภายในของ Generic Interrupt Controller (GIC) และการเชื่อมโยงกับโมดูลอื่นๆ ตามรูปที่ 2.18

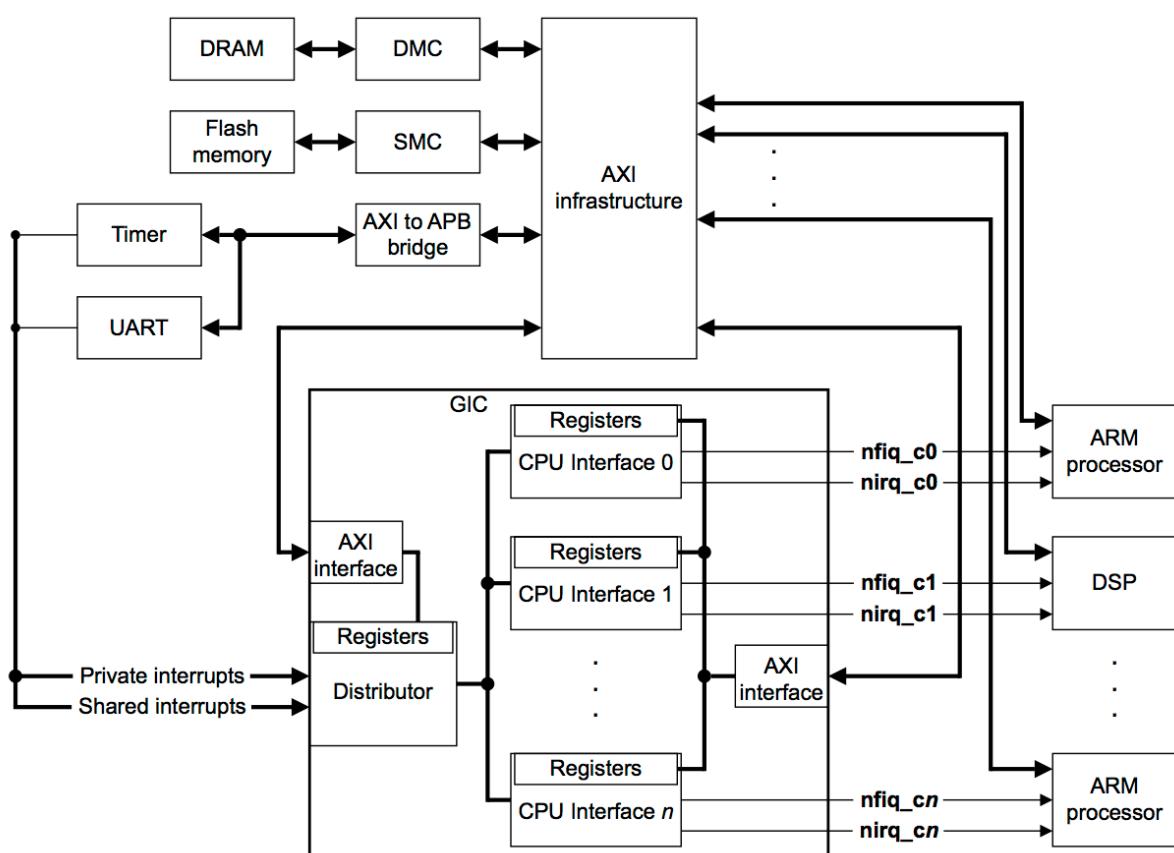


Figure 6.18: โครงสร้างภายในของ Generic Interrupt Controller (GIC) และการเชื่อมโยงกับโมดูลอื่นๆ กับ ARM Advanced eXtensible Interface (AXI) ที่มา: [arm.com](#), APB (ARM Peripheral Bus)

รูปที่ 2.19 แสดงให้เห็นว่า การทำงานและโหมดแกรมเวลาของ GIC เมื่อเกิดสัญญาณร้องขอ (Interrupt Request) คือ Interrupt M และ Interrupt N จำนวนสองสัญญาณ ในห่วงเวลาที่ใกล้เคียงกัน โดยที่สัญญาณที่มากายหลังอาจมีความสำคัญ (Priority) เหนือกว่าสัญญาณที่ร้องขอมาถึงก่อน การตอบสนองต่อการร้องขอที่เกิดขึ้นนั้นจะขึ้นกับความสำคัญเป็นหลัก (Higher Priority) และสามารถหยุดพักรอตอบสนองต่อการร้องขอที่มีความสำคัญต่ำกว่าชั่วคราว (Pending) เพื่อที่ซีพียูจะตอบสนองต่อการร้องขอที่สำคัญกว่าได้ เรียกว่า Nested Interrupt และซีพียูจะกลับไปตอบสนองจากที่พักไว้ต่อไปเมื่อเสร็จสิ้น

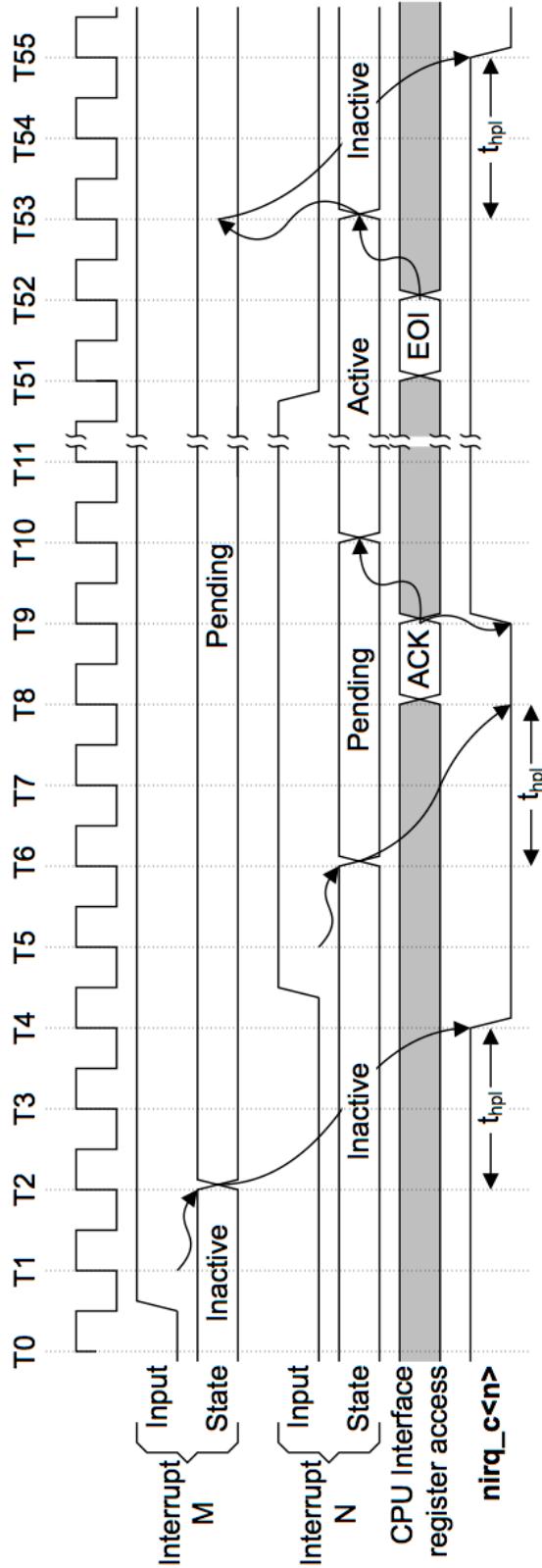


Figure 6.19: การทำงานและไดอะแกรมเวลาของ Generic Interrupt Controller (GIC) ที่มา: ?gic

คาดว่าต่างๆ ทำงานที่ขอบขั้นของสัญญาณคลื่อกระแทกนั้น ตามลำดับดังนี้

- คาดเวลา T1 ของ Distributor ตรวจจับการร้องขอของ Interrupt M

- คابเวลา T2 วงจร Distributor พักรการร้องขอจาก Interrupt M
- คابเวลา T4 วงจรเชื่อมต่อกับ ชีพียูตั้งค่า Interrupt M กับสัญญาณร้องขอการขัดจังหวะ `nirq_c<M>`
- คابเวลา T5 วงจร Distributor ตรวจจับการร้องขอของ Interrupt N ซึ่งมีความสำคัญสูงกว่า Interrupt M จึงทำให้ต้องหยุดการทำงานของ Interrupt M ในคابเวลาถัดไป
- คابเวลา T6 วงจร Distributor พักรการร้องขอจาก Interrupt N วงจรเชื่อมต่อกับชีพียูตั้งค่า INTID ของ Interrupt N กับสัญญาณร้องขอการขัดจังหวะ `nirq_c<n>` แทน
- คابเวลา T8 ชีพียูตอบสนองต่อสัญญาณร้องขอ ด้วยการส่งสัญญาณ ACK
- คابเวลา T9 ชีพียูอ่านค่า Interrupt Acknowledge Register และตอบสนองต่อ `nirq_c<n>` โดยเปลี่ยนค่าให้เป็น 1
- คابเวลา T10 วงจร Distributor ตั้งค่าสถานะของ Interrupt N เป็น Active และเปลี่ยนแปลงค่า Active Status Register
- คابเวลา T11-T50 ชีพียูตอบสนองโดยการทำงานที่ Interrupt Service Routine ของ Interrupt N หลังจากนั้นอุปกรณ์ยกเลิกการร้องขอ
- คابเวลา T51 ชีพียูบันทึกค่ารีจิสเตอร์ End of Interrupt (EOI) ด้วยหมายเลข INTID ของ Interrupt N
- คابเวลา T52 วงจร Distributor ตั้งค่าสถานะของ Interrupt N เป็น Inactive และเปลี่ยนแปลงค่า Active Status Register.
- คابเวลา T53 วงจร Distributor แจ้งวงจรเชื่อมต่อชีพียูว่า interrupt M ยังคงอยู่และมีความสำคัญสูงที่สุด ณ เวลานี้
- คابเวลา T55 วงจรเชื่อมต่อชีพียูส่งสัญญาณร้องขอ `nirq_c<n>` ให้กลับเป็น 0 เพื่อขัดจังหวะการทำงานของ ชีพียู เป็นลำดับถัดไป

ผู้อ่านสามารถใช้งานขา GPIO ทำงานร่วมกับการขัดจังหวะใน การทดลองที่ 9 ภาคผนวก K โดยพัฒนาโปรแกรมประยุกต์ด้วยภาษา C และแ Olsen เชมบลีทำงานร่วมกับไลบรารี `wiringPi` และกิจกรรมท้ายการทดลองจะช่วยเสริมให้ผู้อ่านเข้าใจการทำงานของอินเทอร์รัฟท์เพิ่มมากขึ้น

6.13 การเข้าถึงหน่วยความจำโดยตรง (Direct Memory Access)

การรับส่งข้อมูลผ่าน GPIO มีข้อจำกัดในเรื่องของความเร็ว จึงไม่เหมาะสมกับการถ่ายโอนข้อมูล จำนวนมาก ตั้งแต่ 16 ไบท์ขึ้นไป ดังนั้น ARM จึงได้ออกแบบขบวนการใหม่ เรียกว่า การเข้าถึงหน่วยความจำโดยตรง (Direct Memory Access) เพื่อทำงานร่วมกับการขัดจังหวะ

ยกตัวอย่างเช่น การรับส่งข้อมูลผ่านเครือข่าย Ethernet ในรูปของเฟรมข้อมูลที่มีความยาวขั้นต่ำ 64 ไบท์และไม่เกิน 1522 ไบท์ ดังนั้น การรับข้อมูลจากอุปกรณ์อินพุต DMA จะทำหน้าที่เคลื่อนย้ายข้อมูลจาก FIFO ของวงจรเพื่อต่อไปยังหน่วยความจำหลัก และการส่งข้อมูลไปยังอุปกรณ์เอาท์พุตต่างๆ DMA จะทำหน้าที่เคลื่อนย้ายข้อมูลจากหน่วยความจำหลัก ไปยัง FIFO ของวงจรนั้นๆ

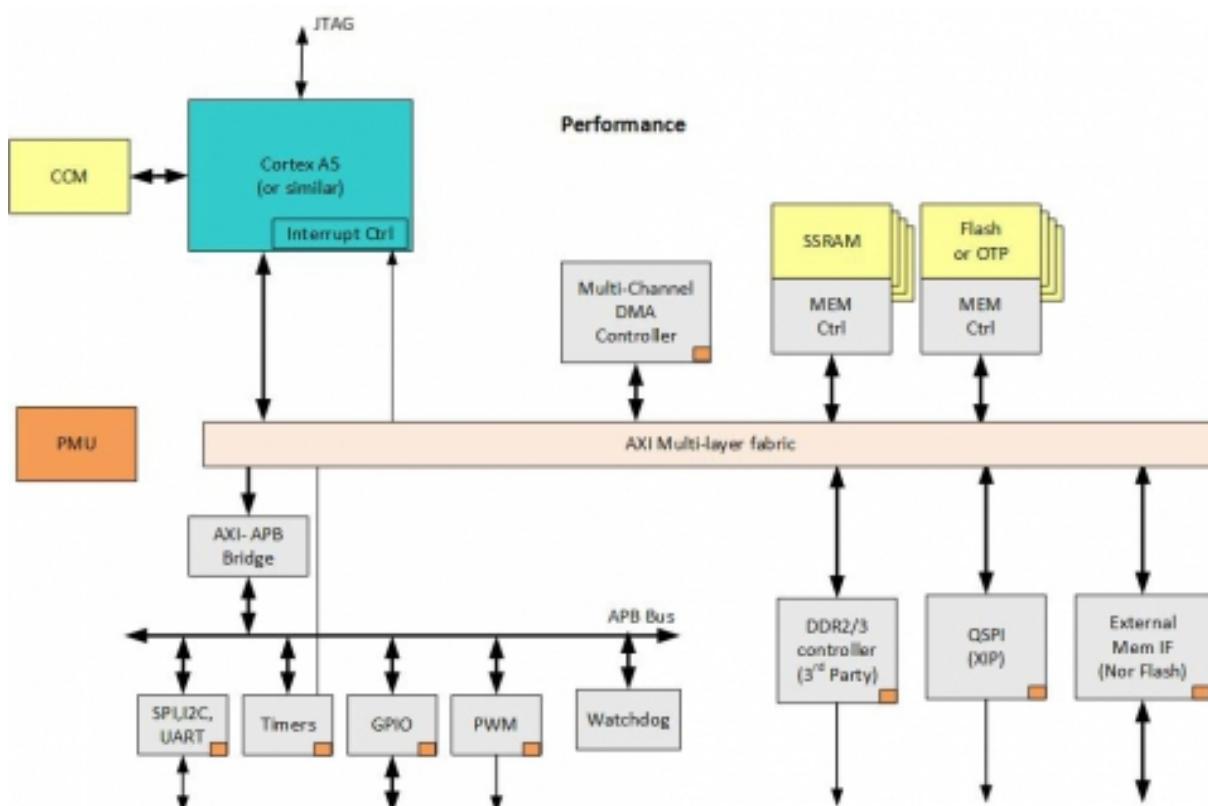


Figure 6.20: การเชื่อมต่อระหว่าง Cortex A5 และโมดูล DMA (Direct Memory Access) ด้วยบัส AXI (Advanced eXtensible Interface) ภายในชิป ที่มา: design-reuse.com

รูปที่ 2.20 แสดง การเชื่อมต่อระหว่าง Cortex A5 และโมดูล DMA (Direct Memory Access) ผ่าน วงจร AXI (Advanced eXtensible Interface) ภายในชิป โดยสามารถรองรับขบวนการได้พร้อมกัน 16 ช่อง แต่ละช่องสามารถทำงานได้อิสระจากกัน

ขบวนการ DMA เริ่มต้นจาก DMA Controller อ่านค่าการติดตั้งใน CB (Control Block) ในหน่วยความจำไปตั้งค่าริจิสเตอร์ CONBLK_AD ภายใน DMA ช่อง (Channel) ที่ยังว่างอยู่ หลังจากนั้น วงจร DMA ช่องนั้นจะเริ่มทำงานด้วยตนเอง เมื่อทำงานแล้วเสร็จ DMA จะส่งสัญญาณขัดจังหวะไปแจ้งเตือน ซึ่งหมายว่างานเสร็จสิ้นแล้ว ผ่าน GIC ตามที่ได้กล่าวไปแล้วในหัวข้อที่ 2.12

ข้อมูลการติดตั้งของแต่ละ CB ประกอบด้วยข้อมูลจำนวน 256 บิต หรือแบ่งเป็น 8 x 32 บิต ซึ่งตรง กับแอ็คเดรสของ DMA แต่ละช่องที่มีค่าห่างกัน 256 ไบท์

- 0x7E00_7000 สำหรับตั้งค่าการทำงานของ DMA ช่อง 0
- 0x7E00_7100 สำหรับตั้งค่าการทำงานของ DMA ช่อง 1
- 0x7E00_7200 สำหรับตั้งค่าการทำงานของ DMA ช่อง 2
- ...

การทำงานของ DMA แต่ละช่องจะมีลักษณะเหมือนกัน คือ ตั้งค่าการทำงานของรีจิสเตอร์ด้วยชุดข้อมูล Control Block (CB) แต่ละชุดจะเชื่อมโยงกันเป็นลิงค์ลิสต์ (Linked List) ในหน่วยความจำ เมื่อการตั้งค่า CB ชุดแรกเสร็จสิ้น ค่า CB ชุดต่อไปจะถูกอ่านเพื่อไปตั้งค่าการทำงานชุดต่อไป โดยอัตโนมัติ ทำให้ชีพียูไม่จำเป็นต้องเกี่ยวข้องและสามารถทำงานอื่นที่สำคัญกว่าได้

6.14 แหล่งจ่ายไฟ (Power Supply) ของบอร์ด Pi3

แหล่งจ่ายไฟของบอร์ดมาจากแดปเตอร์ (Adaptor) ที่แปลงไฟฟ้ากระแสสลับ 220 โวลท์เป็นไฟฟ้ากระแสตรงความต่างศักย์ 5.0 โวลท์ โดยใช้หัวคอนเนคเตอร์ Micro USB จะต้องจ่ายกระแสไฟฟ้าได้ไม่น้อยกว่า 1.0 ถึง 2.5 แอมป์ ทั้งนี้ เพื่อให้บอร์ด Pi3 สามารถทำงานได้เต็มประสิทธิภาพ กระแสจะไหลผ่านพิวส์ และไดโอด เพื่อจ่ายไฟให้กับบอร์ด Pi3 และจึงแปลงความต่างศักย์ 5 โวลท์ให้ลดลง (Step Down) เป็น 3.3 และ 1.8 โวลท์ด้วยชิป PAM2306 และ 1.2 โวลท์ด้วยชิป RT8088A ตามลำดับ

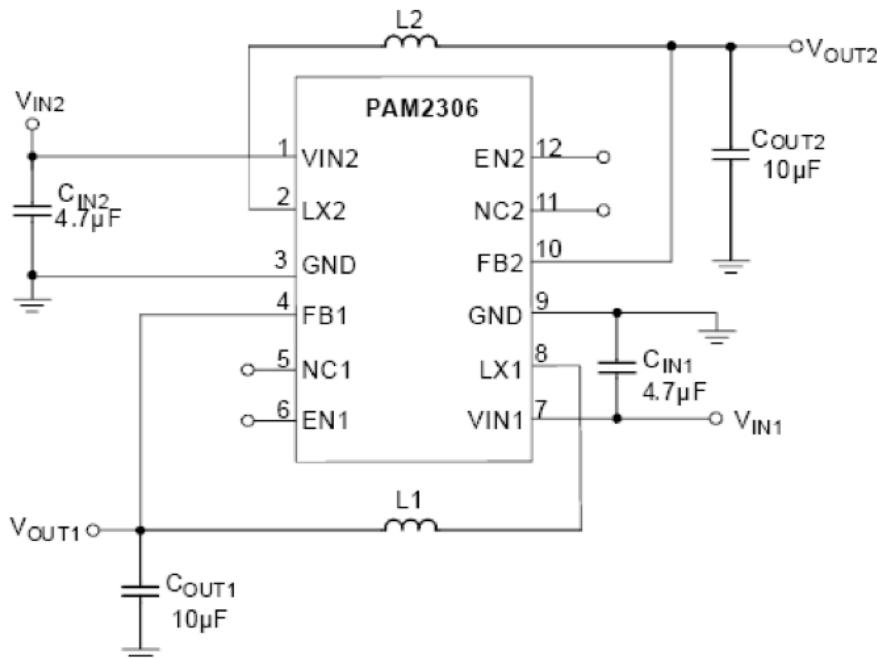


Figure 6.21: การแปลงไฟกระแสตรง 5 โวลท์เป็นไฟกระแสตรงด้วยไอซี PAM 2306 DC-DC Covertor คู่กำหนดค่าเอาท์พุตด้วยค่าตัวหนี่ยวนำ L_1 และ L_2 หน่วยเป็น ไมโครเอนรี ที่มา: [Diodes \(2012\)](#)

ชิป PAM2306 จะทำหน้าที่แปลงไฟกระแสตรง 5.0 โวลท์ ให้เป็นไฟกระแสตรงความต่างศักย์ 3.3 โวลท์ เพื่อเลี้ยงชิปต่างๆ และวงจรทั้งหมด PAM2306 รองรับอินพุตตั้งแต่ 2.5V ถึง 5.5V โดยสามารถแปลงเอาท์พุตโวลเทจได้เป็นสองระดับพร้อมๆ กัน ซึ่งผู้ออกแบบสามารถเลือกได้ดังนี้ 3.3V, 2.8V, 2.5V, 1.8V, 1.5V, 1.2V หรือใช้ความต้านทานปรับค่าได้ PAM2306 ทำงานในลักษณะ DC-DC converter ควบคุณการทำงานแบบ Step Down ด้วยกระแส การ PWM เพื่อให้เกิดเสถียรภาพ และรักษาอายุการใช้งานของแหล่งจ่ายไฟ เมื่อต้องจ่ายกระแสมากขึ้น จนถึง โดยใช้หลักการ Pulse-Skipping Modulation (PSM) เพื่อลดกระแสในขณะที่จ่ายกระแสสนับสนุน

ขา V_{IN1} จะถูกแปลงให้มีความต่างศักย์ลดลงเป็น V_{OUT1} ขา V_{IN2} จะถูกแปลงให้มีความต่างศักย์ลดลงเป็น V_{OUT2} โดยจะควบคุมการเปิดปิดด้วยขา EN1 (Enable1) และ EN2 (Enable2) ตามลำดับ ค่าโวลเทจทั้ง V_{OUT1} และ ค่าโวลเทจทั้ง V_{OUT2} ถูกกำหนดที่หมายเลขอ้วลังของไอซีมาจากการผลิต โดยผู้ออกแบบต้องเลือกค่าตัวหนี่ยวนำให้เหมาะสมกำหนดความต่างศักย์ของ V_{OUT1} และ V_{OUT2} กำหนดจากค่าตัวหนี่ยวนำ L_1 และ L_2 หน่วยเป็น ไมโครเอนรี ได้ตามตารางที่ 2.7

Table 6.7: ตารางเลือกค่า V_{OUT} และค่าตัวเหนี่ยวนำ L_1 และ L_2 ที่มา: [Diodes \(2012\)](#)

V_{OUT} (โวลท์)	L (μH)
1.2	2.2
1.5	2.2
1.8	2.2
2.5	4.7
3.3	4.7

เอกสารคุณสมบัติความต้องการด้านกำลังไฟสำหรับบอร์ด Pi3 ได้ระบุว่าบอร์ดต้องการกระแส 700 มิลลิแอมป์ ถึง 2.5 แอมป์ ขึ้นอยู่กับโมเดลที่ใช้และการใช้งานจริง ซึ่งแหล่งจ่ายไฟขนาด 5 โวลท์ 1 แอมป์ขึ้นไปสามารถใช้งานได้ดี โดยเชื่อมต่อกับอุปกรณ์ เช่น เม้าส์ คีย์บอร์ด และ WiFi ผ่าน USB แหล่งจ่ายไฟควรมีความต่างศักย์ $5 \pm 0.25\text{V}$ ที่กระแสกำหนดไว้บนอุปกรณ์ เช่น 1 แอมป์แพร์ หรือมากกว่า โดยผู้ใช้ไม่ต้องกังวลว่าการจ่ายกระแสมากเกินไป จะเป็นผลเสีย เพราะบอร์ดจะปริโภคกระแสเท่าที่จำเป็น และมีฟิวส์คุ้มครองช่วยจำกัดกระแส 2.5 แอมป์ นอกจากนี้ บอร์ด Pi3 และ Pi2 โมเดล B+ มีชิพหมายเลข APX803 ทำหน้าที่เฝ้าระวัง (Monitor) โวลต์เจ้า กรณีไฟเลี้ยงจาก USB มีความต่างศักย์อยู่ในช่วง 4.63 ± 0.07 โวลท์ ไฟ LED สีแดงบนบอร์ดจะสว่างขึ้น สัญญาณสายไฟและข้อความว่า Under Voltage ผ่านทางหน้าจอตรงมุมขวาบน เพื่อเตือนให้ผู้ใช้ทราบว่าไฟเลี้ยงของบอร์ดมีค่าต่ำไป แต่บอร์ดยังใช้งานได้ตามปกติ

ข้อควรระวังเรื่องแหล่งจ่ายไฟให้กับบอร์ด Pi3 คือ ขนาดและความยาวของสายไมโคร USB ที่จ่ายไฟเลี้ยง 5 โวลท์ สายที่มีขนาดเล็กเกินไปจะทำให้ความต้านทานของสายสูงขึ้น เมื่อกระแสไฟ流ผ่านจะทำให้เกิดโวลต์เจตกรคร่อมในสายสูงขึ้นตาม จนทำให้บอร์ดได้รับโวลต์เจไม่สูงพอนั่นคือ ต่ำกว่า 4.63 โวลท์ จนส่งผลถึงความต่างศักย์ด้านเอาท์พุตของ PAM2306 ที่อาจต่ำกว่า 3.3 โวลท์ และทำให้ชิพขาดเสื่อมร้าฟ

6.15 สรุปท้ายบท

โครงสร้างด้านอินพุต/เอาท์พุต และอุปกรณ์สำรองข้อมูลของเครื่องคอมพิวเตอร์เชื่อมด้วยสาย ระบบบัสซีพียูในปัจจุบันอาศัยกลไกการติดต่อกับอุปกรณ์อินพุต/เอาท์พุต เรียกว่า Memory Mapped I/O, กลไกการทำ Interrupt และการทำ Direct Memory Access ร่วมกับ อุปกรณ์เชื่อมต่อกับเครือข่ายและอุปกรณ์เก็บรักษาข้อมูล

6.16 คำนำท้ายบท

- BCM2837 มีจำนวนขาเพื่อเชื่อมต่อกับบอร์ดจำนวนกี่ขา
- จะบอกจำนวนกล้องชนิด CSI ที่ชิพ BCM2837 รองรับได้สูงสุด
- จะบอกจำนวนจอภาพชนิด DSI ที่ชิพ BCM2837 รองรับได้สูงสุด
- จะบอกชื่อขาที่ชิพ BCM2837 สำหรับเชื่อมต่อกับหน่วยความจำ SD ทั้งหมด
- จะบอกจำนวนเลนช์ข้อมูลของการเชื่อมต่อกับจอภาพชนิด HDMI ที่ชิพ BCM2837 รองรับได้สูงสุด

6. จงบอกจำนวนเล่นข้อมูลของการเชื่อมต่อกับกล้องชนิด CSI ที่ชิป BCM2837 รองรับได้สูงสุด
7. จงบอกจำนวนเล่นข้อมูลของการเชื่อมต่อกับจอภาพชนิด DSI ที่ชิป BCM2837 รองรับได้สูงสุด
8. การขัดจังหวะของ ซีพียูเกิดขึ้นจากอุปกรณ์
9. เหตุใดการเชื่อมต่อกับกล้องและจอภาพจึงต้องมีสัญญาณคลือกด้วย
10. เหตุใดการเชื่อมต่อกับ USB, Ethernet และอื่นๆ จึงไม่ต้องมีสัญญาณคลือกทั้งๆ ที่เป็นการส่งด้วยเทคนิค Differential Voltage Signaling เช่นเดียวกับกล้องและจอภาพ
11. นอกเหนือจากสัญญาณ WiFi และ Bluetooth ชิป BCM43438 รองรับสัญญาณใดเพิ่มเติม
12. จงอธิบายหลักการเชื่อมต่อกับ GPIO ของชิป BCM2837 และการขัดจังหวะ โดยตรวจจับสัญญาณ อินพุตที่ขอบขาลง (Falling Edge)
13. เหตุใดการบริโภคพลังงานของบอร์ดจึงต้องการความต่างศักย์ที่ 5 โวลท์แต่ต้องการกระแสไฟตั้งแต่ 700 mA - 2.5 A
14. เหตุใดบอร์ดจึงต้องแปลงไฟเลี้ยง 5 โวลท์เป็น 3.3 และ 1.8 โวลท์

อุปกรณ์เก็บรักษาข้อมูล (Data Storage Devices)

เทคโนโลยีที่ใช้สร้างอุปกรณ์เก็บรักษาข้อมูลที่ได้รับความนิยมในปัจจุบัน ได้แก่ เทคโนโลยีสารกึ่งตัวนำ เทคโนโลยีสารแม่เหล็ก และ เทคโนโลยีสารสะท้อนแสง ดังนั้น อุปกรณ์เก็บรักษาข้อมูลจึงแบ่งเป็นชนิดต่างๆ ตามสารที่ใช้ ดังนี้

- อุปกรณ์เก็บรักษาข้อมูลชนิดสารกึ่งตัวนำ ได้แก่ โซลิดสเตติดิสก์ และ หน่วยความจำการด SD ซึ่งอาศัยชิพหน่วยความจำชนิดแฟลช ทำงานที่สัญญาณคลื่อกความถี่สูงระดับ 100 เมกะเฮิร์ซ
- อุปกรณ์เก็บรักษาข้อมูลชนิดสารแม่เหล็ก ได้แก่ ฮาร์ดดิสก์อาศัยการหมุนของจานแม่เหล็กแข็ง ที่ความเร็วสูง
- อุปกรณ์เก็บรักษาข้อมูลชนิดสารสะท้อนแสง ได้แก่ แผ่นซีดีและแผ่นดีวีดี อาศัยการหมุนของจานสะท้อนแสง ที่ความเร็วปานกลาง

โปรดสังเกตว่าอุปกรณ์เหล่านี้ล้วนทำงานที่สัญญาณคลื่อกความถี่ต่ำกว่า หน่วยความจำ DRAM และ SRAM ซึ่งทำงานที่ 1000-2000 เมกะเฮิร์ซ แต่ต้องอาศัยไฟเลี้ยงเพื่อรักษาข้อมูลให้สูญหายหรือเสียหายตลอดเวลา เทคโนโลยีการเก็บรักษาข้อมูลเหล่านี้มีความแตกต่างกัน มีข้อได้เปรียบเสียเปรียบต่างกัน โดยบทนี้จะเน้นที่ อุปกรณ์เก็บรักษาข้อมูลชนิดสารกึ่งตัวนำ และ อุปกรณ์เก็บรักษาข้อมูลชนิดสารแม่เหล็ก โดยมีวัตถุประสงค์ต่อไปนี้

- เพื่อให้เข้าใจโครงสร้างและกลไกการทำงานของระบบไฟล์ (File System) ในระบบปฏิบัติการตระกูล Unix
- เพื่อให้รู้จักชนิดและกลไกการทำงานของอุปกรณ์เก็บรักษาข้อมูล ได้แก่ หน่วยความจำแฟลช การ์ด SD เป็นต้น

ผู้เขียนใช้คำว่า อุปกรณ์เก็บรักษาข้อมูล ในขณะที่กำราบงเล่ม เว็บไซต์บางที่เรียกว่า หน่วยความจำ หรือ หน่วยเก็บข้อมูล หรือ หน่วยบันทึกข้อมูล ทำหน้าที่บันทึกและเก็บไฟล์ทั้งหมดของระบบปฏิบัติการ ไฟล์โปรแกรม ไฟล์ข้อมูล และไฟล์อื่นๆ เพื่อให้เครื่องคอมพิวเตอร์สามารถทำงานได้ต่อเนื่องเมื่อเปิด

เครื่องใหม่อีกรอบหลังจากการซัพเดต การปิดเครื่องคอมพิวเตอร์เป็นการช่วยประหยัดพลังงาน หรือกรณีขุกเณนไฟฟ้าดับ หรือแบตเตอรี่หมด ซึ่งอาจทำให้ไฟล์ข้อมูลเสียหายหรือถึงขั้นสูญหาย ดังนั้น ผู้ใช้และซอฟต์แวร์ต้องมีหน้าที่บันทึก (Save) ข้อมูลลงในไฟล์ข้อมูลระหว่างการปฏิบัติงานด้วยเข่นกัน ยิ่งไปกว่านั้นกรณีผู้ใช้ไม่สามารถซัพเดตข้อมูลระบบอาจทำให้ระบบไฟล์และไฟล์ข้อมูลเสียหายได้ ทั้งนี้ขึ้นอยู่กับชนิดของระบบไฟล์ ผู้อ่านควรศึกษาและทดลองระบบไฟล์ของลีนุกซ์เพิ่มเติมในการทดลองที่ 10 ภาคผนวก L ซึ่งจะช่วยเสริมความเข้าใจเรื่องคำสั่งของระบบ Unix ใน การทดลองที่ 4 ภาคผนวก D

7.1 ระบบไฟล์ (File System)

ระบบบริหารจัดการไฟล์ที่สำคัญและเป็นที่นิยม ได้แก่

- ระบบ NTFS (File System) สำหรับระบบวินโดวส์ รายละเอียดเพิ่มเติมที่ ntfs.com
- ระบบ EXT4 สำหรับระบบลีนุกซ์ รายละเอียดเพิ่มเติมที่ wikipedia
- ระบบ Apple File System (APFS) สำหรับระบบ MAC OS รายละเอียดเพิ่มเติมที่ wikipedia

ตำราเล่มนี้จะเน้นที่พื้นฐานของระบบไฟล์ดังเดิมของระบบยูนิกซ์ ซึ่งระบบอื่นๆ ได้พัฒนาต่ออยอด

7.1.1 การใช้งานไฟล์ (File Operations)

ผู้ใช้และนักพัฒนาสามารถใช้งานไฟล์โดยการ สร้างไฟล์ (Create) เขียน/บันทึก/อ่านไฟล์ เปลี่ยนชื่อ (Rename) ไฟล์ ลบ (Remove/Delete) ไฟล์ ทำสำเนา (Copy) ย้ายตำแหน่ง (Move) ไฟล์ และกู้คืน (Recover) ไฟล์เมื่อเกิดปัญหา โดยไม่จำเป็นต้องรับรู้ว่าระบบไฟล์เป็นชนิดใด เนื่องจากระบบปฏิบัติการได้ซ่อนรายละเอียดไว้ เพื่อให้ผู้ใช้และนักพัฒนาโปรแกรม สามารถพัฒนาโปรแกรมได้สะดวกมากขึ้น

การเปิดไฟล์ คือ การจดหน่วยความจำเมื่อนจำนวนหนึ่งเพื่อทำหน้าที่เป็นบัฟเฟอร์ โดยเริ่มต้นค่าพอยท์เตอร์ หรือ แอดдресเริ่มต้นของบัฟเฟอร์คืน ตำแหน่งเริ่มต้นไฟล์ การเปิด แบ่งเป็น เพื่อการอ่านอย่างเดียว (Read Only) เพื่อการเขียนอย่างเดียว (Write Only) และเพื่ออ่านและเขียน (Read-Write) การปิดไฟล์ คือ การคืนพื้นที่บัฟเฟอร์ให้กับระบบ

การอ่านเขียนข้อมูลในหน่วยความจำอยู่ในรูปของการอ่านเขียนไฟล์แทน โดยที่โปรแกรมเมอร์ต้องเปิดไฟล์โดยใช้คำสั่ง fopen() ในภาษา C fread() สำหรับอ่านข้อมูลในไฟล์ fwrite() สำหรับเขียนข้อมูล และเมื่อต้องการเลิกใช้งานไฟล์ข้อมูลใดๆ โปรแกรมเมอร์จะต้องปิดไฟล์ด้วยคำสั่ง fclose() ระบบลีนุกซ์ ได้พัฒนาพังค์ชันเหล่านี้ให้กับโปรแกรมเมอร์ ซึ่งจะทำหน้าที่ติดต่อกับอุปกรณ์เก็บรักษาข้อมูลผ่านวงจรต่างๆ

การอ่านหรือเขียนไฟล์จะเขียนหรืออ่านข้อมูลในหน่วยความจำเมื่อตอน โดยบริเวณพื้นที่หน่วยความจำเมื่อนี้ที่ทำหน้าที่เป็นบัฟเฟอร์ เมื่อบัฟเฟอร์เต็ม โอเอสจะอ่านข้อมูลจำนวนหนึ่งมาแทนที่บริเวณบัฟเฟอร์นั้น หรือย้ายข้อมูลจากบัฟเฟอร์ไปเขียนลงในไฟล์ ในลักษณะของ Block File System ตามรูปที่ 3.14 ในหัวข้อที่ 3.2.5

UNIX File System Layout

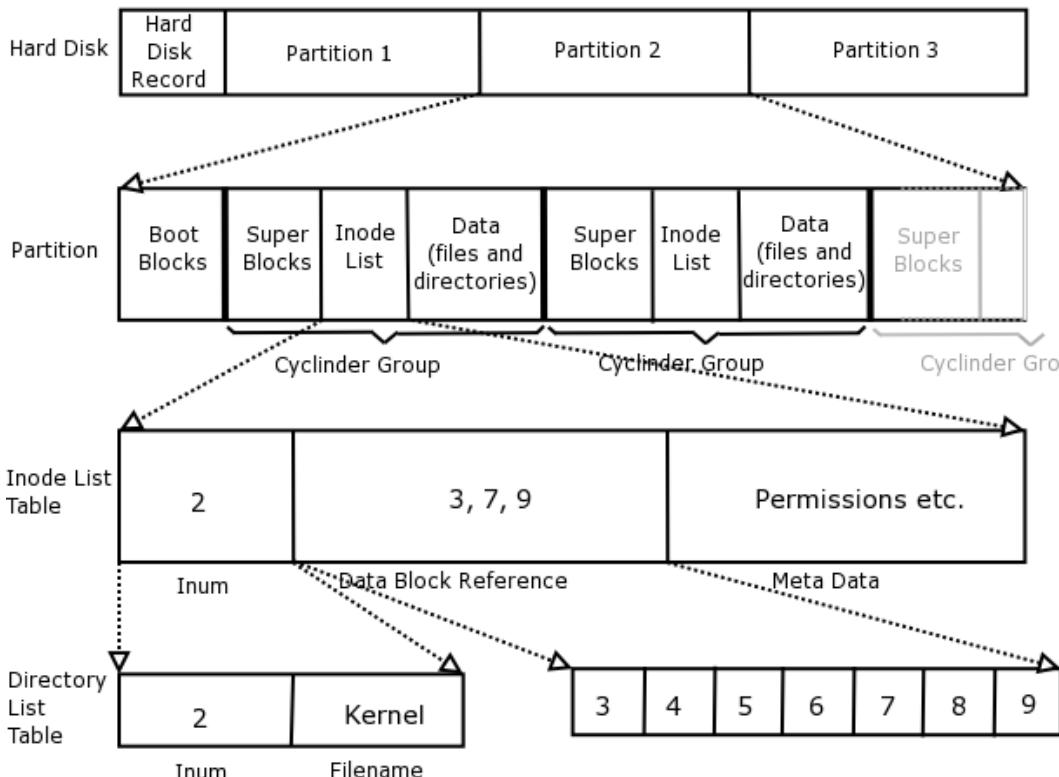


Figure 7.1: โครงสร้างของระบบไฟล์ในระบบปฏิบัติการตระกูล Unix ที่มา: [Demblon and Spitzner \(2004\)](#)

7.1.2 การแบ่งพาร์ติชัน (Partition)

เราจะจินตนาการว่าอุปกรณ์เก็บรักษาข้อมูล เช่น การ์ดหน่วยความจำ SD โซลิดสเตทไดส์ค ฮาร์ดดิสค์ งานหมุน เหล่านี้ มีลักษณะเป็นแบบข้อมูลที่มีความยาวตามขนาดความจุ และแบ่งແນບออกเป็นพื้นที่ย่อยๆ เรียกว่า พาร์ติชัน (Partition) ซึ่งอุปกรณ์รักษาข้อมูล 1 ตัวสามารถแบ่งเป็นหลายพาร์ติชัน (Partition) แต่ละพาร์ติชันต้องมีระบบบริหารจัดการไฟล์ (File System) ของตนเอง ตามรูปที่ 7.1

การแบ่งพาร์ติชัน คือ การแบ่งพื้นที่แบบข้อมูลออกเป็นส่วนต่างๆตามต้องการ การแบ่งพาร์ติชันสามารถตอบสนองความต้องการที่หลากหลาย ได้แก่ การบูรณาการปฏิบัติการได้หลากหลาย การจัดเก็บข้อมูลในบางพาร์ติชัน เป็นต้น ผู้ใช้สามารถติดตั้งระบบปฏิบัติการในแต่ละพาร์ติชันได้โดยอิสระจากกัน ที่มา [archlinux](#) และ [datadoctor.biz](#) โดยตำแหน่งเริ่มต้นจะเป็นพื้นที่สำหรับเก็บ ตารางพาร์ติชัน (Partition Table) โดยรายชื่อและข้อมูลประกอบของแต่ละพาร์ติชันจะถูกบันทึกลงในตารางแบ่งตามยุคสมัย ดังต่อไปนี้

- มาสเตอร์บูทเรคอร์ด (Master Boot Record: MBR) รายชื่อพาร์ติชัน ตำแหน่งเริ่มต้นหรือหมายเลข Logical Block ในฮาร์ดดิสค์ ไว้ในตารางพาร์ติชัน (Partition Table) ในอดีตักคอมพิวเตอร์เรียกว่า มาสเตอร์บูทเรคอร์ด ในอดีต พื้นที่ส่วนหนึ่งของมาสเตอร์บูทเรคอร์ดทำหน้าที่ กีบตารางพาร์ติชัน (Partition Table) ตั้งอยู่ในเข็คเตอร์แรก หรือเข็คเตอร์หมายเลข 0 โดยใช้พื้นที่ 64 ไบท์ ซึ่งดิสค์ 1 ลูก สามารถแบ่งพาร์ติชันได้มากที่สุดเป็นจำนวน 4 พาร์ติชัน รายละเอียดเพิ่มเติมที่ [wikipedia](#)

- Globally Unique Identification Partition Table: GUIDPT ในปัจจุบันตารางพาร์ติชันมีชื่อว่า Globally Unique Identification Partition Table หรือ GUIDPT เรียกย่อๆ ว่า GPT ทำหน้าที่คล้ายกับ MBR แต่รองรับจำนวนพาร์ติชันในฮาร์ดดิสก์ได้มากกว่า และไปอสหรือเฟิร์มแวร์ของเครื่องคอมพิวเตอร์นั้นๆ ตามมาตรฐาน เรียกว่า **Unified Extensible Firmware Interface: UEFI** รายละเอียดเพิ่มเติมเกี่ยวกับ GPT ที่ [wikipedia](#) เพื่อให้ตารางพาร์ติชันสามารถรองรับระบบปฏิบัติ 64 บิต และฮาร์ดดิสก์ที่มีความจุเพิ่มสูงขึ้นระดับเพتل่าไบท์ (Peta Byte) หรือ 10^{15} ไบท์ หรือ 1000 เทอร่าไบท์

7.1.3 โครงสร้างของระบบไฟล์ Unix (Structure of Unix File System)

หน้าที่ของระบบไฟล์ คือ รองรับการจัดเก็บไฟล์ต่างๆ บันทึกประวัติการเข้าถึง (Access) ไฟล์ การบริหารสิทธิ์ (Permission) การเข้าถึงไฟล์ต่างๆ จากผู้ใช้งานจำนวนมาก การถูไฟล์คืน การลบไฟล์ การใช้งานไฟล์ เป็นต้น ตามที่ผู้ใช้และซอฟต์แวร์ประยุกต์ต้องขอ

โครงสร้างของระบบไฟล์ในระบบปฏิบัติการตระกูล Unix จะมีโครงสร้าง และหน้าที่ของระบบไฟล์เป็นชั้นการจัดการ 3 ชั้น โดยเรียงลำดับ ดังนี้

- ชั้น Logical เป็นชั้นบนสุด ทำหน้าที่เชื่อมต่อกับซอฟต์แวร์ประยุกต์ โดยใช้คำสั่งเปิดไฟล์ ปิดไฟล์ อ่านไฟล์ เขียนไฟล์ เป็นต้น ซึ่งไฟล์ที่ซอฟต์แวร์ประยุกต์ต้องการใช้งาน โดยเนื้อหาในบทนี้จะเน้นที่การทำงานในชั้น Logical
- ชั้น Virtual ทำหน้าที่เชื่อมต่อระหว่างชั้น Logical และชั้น Physical รายละเอียดขึ้นอยู่กับวิธีการพัฒนาโปรแกรมของแต่ละระบบปฏิบัติการ
- ชั้น Physical เป็นชั้นล่างสุด ทำหน้าที่จัดการบล็อกข้อมูลบนอุปกรณ์ เพื่อตอบสนองต่อการร้องขอจากชั้น Virtual ซึ่งกลไกสำคัญคือ การบริหารบัฟเฟอร์ (Buffer) และหน่วยความจำหลัก การจัดวางตำแหน่งบล็อกข้อมูลให้ประสิทธิภาพการอ่านหรือเขียนดีขึ้น การเชื่อมต่อกับดีไวซ์ไดรเวอร์ของอุปกรณ์เก็บรักษาข้อมูล

เมื่อผู้ใช้จัดแบ่งพาร์ติชันสำเร็จแล้ว ระบบปฏิบัติการจะทำการเขียนโครงสร้างของพาร์ติชันตามที่ได้ออกแบบไว้ เรียกว่า การฟอร์แมท (Format) หลังจากนั้น การติดตั้งระบบปฏิบัติการลงบนพาร์ติชันที่ต้องการ โดยจะทำการฟอร์แมท (Format) คือ การจัดแบ่งพื้นที่ออกเป็นพื้นที่ต่างๆ ในรูปที่ 7.1 พาร์ติชันหนึ่งแบ่งเป็น Boot Blocks และ Cylinder Group จำนวนหนึ่ง โดยแต่ละ Cylinder Group แบ่งเป็นพื้นที่ส่วนย่อยๆ ดังนี้

- บล็อกพิเศษ (Superblock) ทำหน้าที่เก็บรายละเอียดต่างๆ ของระบบไฟล์ ขนาดของบล็อกข้อมูล รายละเอียดการใช้งานบล็อกข้อมูลต่างๆ เช่น สถานะว่างหรือใช้งานอยู่ เป็นต้น
- ตาราง Inode (Inode Table) หรืออาจเรียกว่า ไอโหนดอะเรย์ (Inode Array) หรือ ไอโหนดลิสต์ (Inode List) ทำหน้าที่เก็บโครงสร้างข้อมูล Inode ภายใต้ Cylinder Group นี้ รายละเอียดเพิ่มเติมในหัวข้อถัดไป

- บล็อกข้อมูล (Data Block) จำนวนหนึ่งแบ่งผันตามขนาดที่ผู้ติดตั้งระบบกำหนดระหว่างที่ทำการฟอร์แมท ขนาดของบล็อกข้อมูลแต่ละบล็อกเท่ากับ 2048 ไบต์ ทั้งนี้ขึ้นกับชนิดของเทคโนโลยีที่ใช้ เมื่อกำหนดขนาดของแต่ละบล็อกแล้ว จำนวนบล็อกจะแบ่งผันตามความจุของพาร์ติชันนั้นๆ

7.1.4 ไอโหนด (Inode) คืออะไร

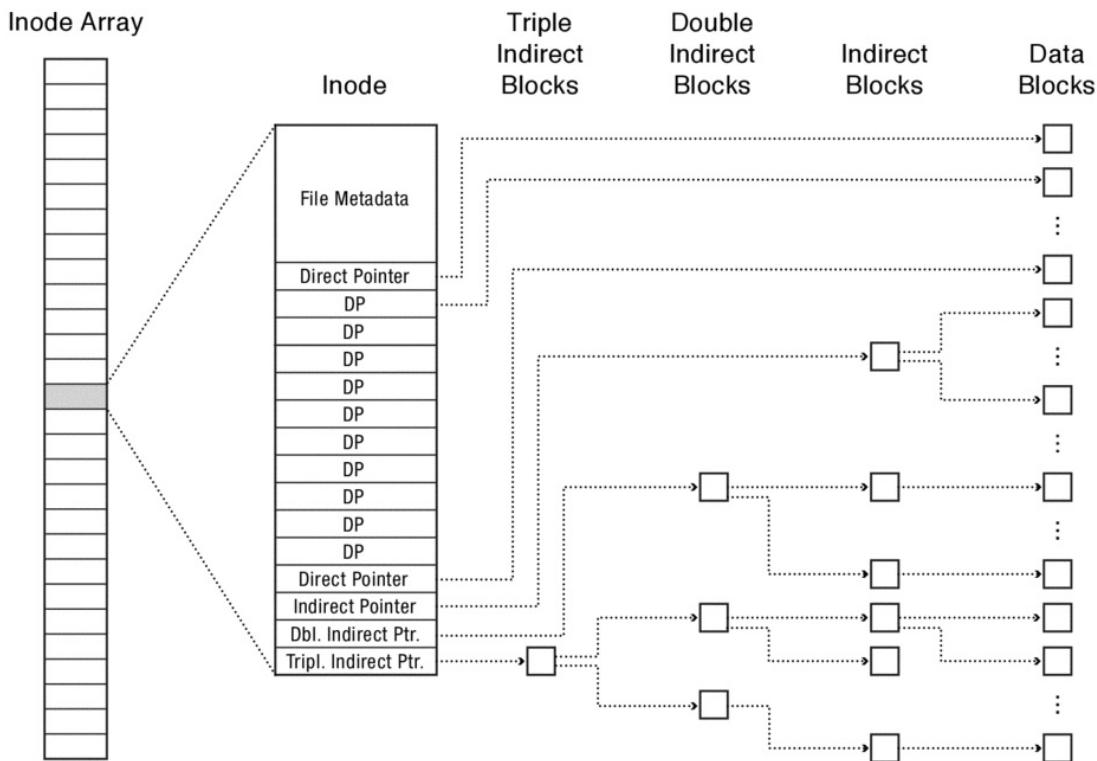


Figure 7.2: โครงสร้างของไอโหนดและการเชื่อมโยงกับบล็อกข้อมูลของไฟล์หนึ่งไฟล์ ที่มา: ?

ไอโหนด (Inode) คือโครงสร้างข้อมูล (Data Structure) ชนิดหนึ่งมีรายละเอียดและหมายเลขอีกับประจำตัว (Inode Number: Inum) ไอโหนดหนึ่งตัวจะทำหน้าที่แทนไฟล์ หรือ ไดเรคทอรี อย่างใดอย่างหนึ่ง โครงสร้างไอโหนด 1 ตัวต้องการพื้นที่ประมาณ 128 Bytes ขึ้นกับรายละเอียดของแต่ละระบบ ตาราง Inode Table หรือ Inode List จะจัดเรียง Inode ทั้งหมด โดยเรียงตามหมายเลข 0 เป็นต้นไป ตามโครงสร้างข้อมูลในรูปที่ 7.2 ที่มา: ?

การติดตั้งระบบปฏิบัติการสำหรับอุปกรณ์ ระบบไฟล์จะกำหนดจำนวนไอโหนดสูงสุด ตามขนาดหรือความจุของแต่ละพาร์ติชัน ซึ่งจะปรับเปลี่ยนตามขนาดของพาร์ติชันนั้นๆ ไอโหนดหนึ่งตัวทำหน้าที่จัดเก็บรายละเอียด ดังนี้

- หมายเลขอีโหนด (Inode Number: Inum): หมายเลขอีโหนดประจำตัวของไฟล์หรือไดเรคทอรี นั้นๆ เป็นเลขจำนวนเต็มชนิดไม่มีเครื่องหมาย
- การเชื่อมโยงไปยังบล็อกข้อมูล (Links to Data Blocks) ทำหน้าที่เก็บหมายเลขอีกบล็อกข้อมูล (บล็อกสี่เหลี่ยมจตุรัส) ซึ่งทำหน้าที่เก็บข้อมูลจริงๆ การเชื่อมโยงแบ่งเป็น 3 ชนิด

- หมายเลขบล็อกข้อมูลทางตรง (Direct Blocks หรือ Direct Pointers: DP) คือ หมายเลขบล็อกข้อมูลที่เก็บข้อมูลสำหรับไอโหนดนี้ สำหรับไฟล์ขนาดเล็ก ในรูปมีจำนวน DP รวมเท่ากับ 11 ตำแหน่ง สามารถเก็บหมายเลขบล็อกได้สูงสุด 11 หมายเลข
- หมายเลขบล็อกข้อมูลทางอ้อม (Indirect Blocks หรือ Indirect Pointers) คือ กรณีที่ไฟล์มีขนาดใหญ่ขึ้น ต้องการจำนวนบล็อกข้อมูลมากขึ้น จน Direct Pointer ไม่เพียงพอ ดังนั้นหมายเลขบล็อกข้อมูลทางอ้อม หรือ Indirect Pointer แบ่งเป็น 3 ระดับ
 - * Indirect Pointer จะอาศัยบล็อกข้อมูลหนึ่งบล็อกทำหน้าที่เก็บหมายเลขบล็อกข้อมูล เพื่อซึ้งบล็อกข้อมูลอีกชั้นหนึ่ง โดยสังเกตบล็อกสี่เหลี่ยมจตุรัสที่ขึ้นระหว่างช่อง Indirect Pointer และบล็อกสี่เหลี่ยมจตุรัส ในรูปที่ [7.2](#)
 - * Double Indirect Pointer เมื่อไฟล์มีขนาดใหญ่ขึ้นจน Indirect Pointer ไม่เพียงพอ ระบบไฟล์จะอาศัย Double Indirect Pointer ในโครงสร้างไอโหนด เพื่อซึ้ง Indirect Pointer อีกชั้นหนึ่ง โดยสังเกตบล็อกสี่เหลี่ยมจตุรัสที่ขึ้นระหว่างช่อง Indirect Pointer และบล็อกข้อมูล ในรูปที่ [7.2](#)
 - * Triple Indirect Pointer เมื่อไฟล์มีขนาดใหญ่ขึ้นจน Double Indirect Pointer ไม่เพียงพอ ระบบไฟล์จะอาศัย Triple Indirect Pointer ในโครงสร้างไอโหนดเพื่อซึ้ง Double Indirect Pointer อีกชั้นหนึ่ง โดยสังเกตบล็อกสี่เหลี่ยมจตุรัสที่ขึ้นระหว่างช่อง Double Indirect Pointer และบล็อกข้อมูล ในรูปที่ [7.2](#)
- ข้อมูลอื่นเกี่ยวกับไฟล์ (File Metadata) แบ่งเป็น
 - ชนิดไฟล์: ไฟล์ธรรมดา (regular file), ไดเรคทอรี (directory), ไปป์ (pipe) เป็นต้น
 - * ไฟล์ ตามที่เคยนิยามแล้วในหัวข้อที่ [3.2.6](#) ว่าเป็น การเรียงตัวกันของตัวเลขที่ลະไบท์โดยอาศัยพื้นที่จัดเก็บในอุปกรณ์รักษาข้อมูล เรียกว่า บล็อก ไฟล์เกิดจากการเรียงของบล็อกข้อมูล อย่างน้อย 1 บล็อกขึ้นไป เพื่อจัดเก็บข้อมูลหรือคำสั่งต่างๆ ลงในอุปกรณ์เก็บรักษาข้อมูล
 - * ไดเรคทอรี หรือ โฟลเดอร์ คือ ไฟล์ชนิดหนึ่งที่เก็บหมายเลขไอโหนดที่อยู่ภายใต้โครงสร้างนี้โดยไดเรคทอรีสามารถซ่อนกันได้ เพื่อความสะดวกในการจัดหมวดหมู่ของไฟล์ข้อมูล
 - หมายเลขผู้ใช้ (User ID) ซึ่งเป็นเจ้าของ
 - หมายเลขกรุ๊ป (Group ID) ซึ่งเจ้าของเป็นสมาชิกอยู่
 - บัญชีข้อมูลกำหนดการเข้าถึง (Access control list) เพื่อกำหนดสิทธิ์ของเจ้าของ กรุ๊ป และผู้ใช้คนอื่นๆ ทุกคน
 - ขนาดไฟล์ที่แท้จริง (หน่วยเป็นไบท์)
 - สิทธิ์การเข้าถึงไฟล์ (Permission) : rwx rwx rwx อ่าน (r: read), เขียน (w: write) และรัน (x: execute) จำนวน 9 บิต แบ่งเป็น 3 ชุด
 - * Owner คือ เจ้าของไฟล์ซึ่งล็อกอินเข้าระบบมาแล้วสร้างไฟล์นี้ โดยชื่อเข้าของไฟล์ คือชื่อ Username ที่ล็อกอิน

- * Group of Owner คือ กลุ่มของเจ้าของไฟล์ที่ Username นั้นเป็นสมาชิกอยู่
 - * everyone หมายถึง ผู้ใช้งานระบบใดๆ
- บันทึกเวลา (Time stamp): ประกอบด้วยบันทึกเวลาชนิดต่างๆ ดังนี้
 - * เวลาที่เข้าถึง (access time) หรือเวลาที่อ่านไฟล์
 - * เวลาที่ไฟล์เปลี่ยนแปลง (modification time)
 - * เวลาที่ Inode ถูกเปลี่ยนแปลง (change time)
- อื่นๆ

ตัวอย่างเช่น ไฟล์ชื่อ Kernel ในไดเรคทอรีรoot (/ หรือ root directory) ในรูปที่ 7.1 มี inode หมายเลข (Inum = 2) ซึ่งข้อมูลจะบันทึกอยู่ในบล็อกข้อมูลหมายเลข 3, 7, 9 มีรายละเอียดสิทธิ์ต่างๆ ตามมา เรียกรวมๆ ว่า MetaData ส่วนชื่อไฟล์ (Filename) Kernel จะเก็บบันทึกในตารางไดเรคทอรีอ้างอิงโดยใช้ Inum = 2 เป็นหมายเลขอ้างอิง

หัวข้อถัดไปจะอธิบายการทำงานของเทคโนโลยีหน่วยความจำชนิดต่างๆ โดยเริ่มจากหน่วยความจำแฟลช การ์ดหน่วยความจำ SD โซลิดสเตทไดส์ก และไฮร์ดดิสก์ ตามลำดับ

7.2 ชนิดความจำแฟลช (Flash Memory Chip)

หน่วยความจำแฟลช เดิมเรียกว่า **แฟลชร้อม** (Flash ROM) คำว่า ROM ย่อมาจากคำว่า Read-Only Memory วัตถุประสงค์เดิมของแฟลชร้อม ทำหน้าที่เป็นหน่วยความจำ ที่สร้างจากเทคโนโลยีสารภิค ตัวน้ำด้วยมีคุณสมบัติใช้เก็บคำสั่งและข้อมูลเพื่อการอ่านเป็นหลัก นักพัฒนาประยุกต์ใช้เก็บคำสั่งประจำเครื่อง หรือ **เฟิร์มแวร์ (Firmware)** เป็นหลัก ด้วยความนิยมในเทคโนโลยีชนิดนี้ หน่วยความจำแฟลชจึงถูกพัฒนาอย่างต่อเนื่อง จนสามารถเขียนหรือแก้ไขข้อมูลภายในแฟลชร้อมได้รวดเร็วขึ้นจนกลายเป็นอุปกรณ์เก็บรักษาข้อมูล เพราะแฟลชร้อมมีจุดเด่น คือ ส่วนประกอบการทำงานน้อยชิ้นกว่า น้ำหนักเบากว่า ปริมาตรที่เล็กกว่า ความเร็วในการอ่านหรือเขียนข้อมูลได้รวดเร็วกว่า ความจุที่เพิ่มมากขึ้น และสามารถเก็บข้อมูลได้ยาวนานขึ้น

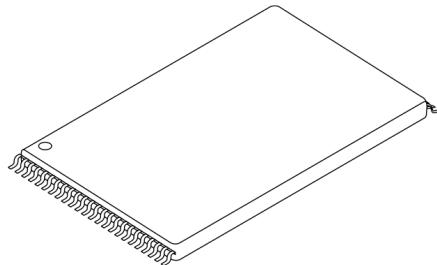


Figure 7.3: หน่วยความจำแฟลช NAND ผลิตโดยบริษัท Micron Technology โดยใช้ตัวถังชนิด TSOP (Thin Small Outline Package) จำนวน 48 ขา ที่มา: [Micron Technology \(2004\)](#)

แฟลชร้อมกรณีศึกษาของตราเล่มนี้ เป็น หน่วยความจำแฟลชชนิด NAND ผลิตโดยบริษัท Micron Technology ในปี ค.ศ. 20xx ใช้ตัวถังชนิด TSOP (Thin Small Outline Package) ในรูปที่ 7.3 ตัวชิปมีการผลิตออกแบบด้วยความจุที่แตกต่างกัน ขึ้นอยู่กับ จำนวนสาย จำนวนบล็อกข้อมูล และจำนวนสายต่อชิป โดยความจุข้อมูลต่อ 1 สายเริ่มต้นที่ ขนาด 2 จิกะบิท หรือ 2^{31} บิท ซึ่งประกอบด้วยบล็อก (Block) ข้อมูล จำนวน 2048 (2^{11}) บล็อก แต่ละบล็อกประกอบด้วย 64 (2^6) เพจ แต่ละเพจ (Page) มีความจุ 2048 (2^{11}) ไบท์ โดย 1 ไบท์ เท่ากับ 2^3 บิท รวมทั้งหมดคิดเป็นจำนวน $2^{11} \times 2^6 \times 2^{11} \times 2^3$ เท่ากับ 2^{31} บิท

หน่วยความจำแฟลช NAND จะมีประสิทธิภาพและความจุเพิ่มสูงขึ้นเรื่อยๆ เมื่อเทคโนโลยีพัฒนาไปเรื่อยๆ โดยทั่วไปประสิทธิภาพการอ่านข้อมูลจะดีกว่าการเขียน ประสิทธิภาพการเขียนจะดีกว่าการลบตามลำดับ โดยอายุการใช้งานชิปจะนับจากจำนวนการลบข้อมูล (Erase) ดังนี้

- ประสิทธิภาพการอ่านข้อมูล ขึ้นอยู่กับตำแหน่งและรูปแบบการอ่านและรูปแบบการเรียงตัวของข้อมูล เช่น การอ่านข้อมูลที่เรียงตัวกันอย่างต่อเนื่อง (Sequential Address) จะใช้เวลาอคอย (Latency) สั้นมากเพียง 30 นาโนวินาที การอ่านข้อมูลที่ไม่เรียงตัวต่อเนื่องและสุ่มตำแหน่ง (Random Address) จะใช้เวลาอ่อนนานขึ้นเป็น 25 ไมโครวินาที จะเห็นได้ว่าใช้เวลาเพิ่มขึ้นเกือบ 1000 เท่า เทียบกับการอ่านข้อมูลแบบเรียงตัวต่อเนื่อง
- ประสิทธิภาพการเขียนข้อมูล มีลักษณะเช่นเดียวกับการอ่านข้อมูล โดย การเขียนข้อมูลต้องเขียนครั้งละเพจ (Page Program) หรือ 2048 ไบท์ ซึ่งจะใช้เวลาอคอยยาวนานกว่าการอ่านข้อมูลเป็น 300 ไมโครวินาที ใช้เวลาเพิ่มขึ้นเป็น 10 เท่า เทียบกับการอ่านข้อมูลแบบสุ่ม

- ประสิทธิภาพการลบข้อมูล (Erase) จะใช้เวลาเพิ่มขึ้นสูงมากถึง 2 มิลลิวินาที จะเห็นได้ว่าใช้เวลาเพิ่มขึ้นเกือบ 100 เท่าเทียบกับการเขียนข้อมูลจำนวนหนึ่งเพียงครั้งเดียว
- อายุการใช้งานของชิพจะนับจากจำนวนครั้งที่เขียนข้อมูล หรือ จำนวนครั้งที่ลบข้อมูล ประมาณ 100,000 ครั้งเท่านั้น แต่ชิพสามารถรักษาข้อมูลได้เป็นระยะเวลายาวนานถึง 10 ปีตามที่แจ้งไว้ในเอกสารคุณสมบัติ

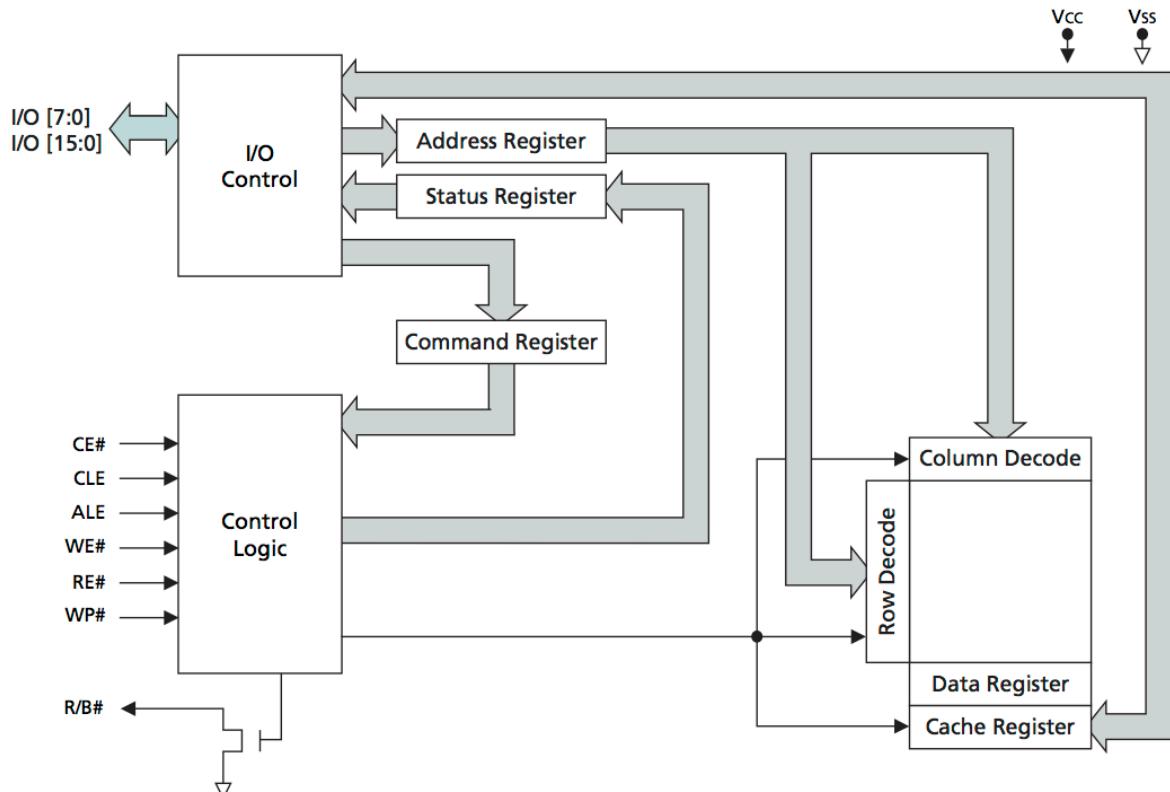


Figure 7.4: โครงสร้างภายในหน่วยความจำ Flash NAND ที่มา: [Micron Technology \(2004\)](#)

7.2.1 โครงสร้างภายในชิพหน่วยความจำแฟลชชนิด NAND

รูปที่ 7.4 แสดง โครงสร้างภายในหน่วยความจำแฟลช NAND ประกอบด้วย

- อะเรย์ของเซลล์หน่วยความจำด้านขวาล่าง
- วงจรควบคุม (Control Logic) การทำงานโดยรวม และ
- วงจรควบคุมอินพุตและเอาท์พุต (I/O Control) เพื่อเชื่อมต่อกับวงจรภายนอกตัวชิพ

โดยตัวชิพหน่วยความจำแฟลชมีขาสัญญาณควบคุม ขาสัญญาณแอ็ตเตรสและขาสัญญาณข้อมูล ดังนี้

- ขา $CE\#$ หรือ \overline{CE} คือ สัญญาณ Chip Enable 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 ทำงานเมื่อได้รับลอจิก 0
- ขา RE# หรือ \overline{RE} คือ สัญญาณ Read Enable ทำงานเมื่อได้รับลอจิก 0
- ขา WP# หรือ \overline{WP} คือ สัญญาณ Write Protect ทำงานเมื่อได้รับลอจิก 0
- ขา R/B# หรือ R/\overline{B} คือ สถานะ Read หากมีค่าเท่ากับลอจิก 1 หรือ Busy หากมีค่าเท่ากับลอจิก 0
- ขาสัญญาณ I/O[0:15] ทั้งหมด 16 ขา มีหน้าที่มัลติเพล็กซ์ (Multiplex) หรือใช้งานร่วมกันคนละช่วงเวลา เพื่อการรับแอดเดรส รับ/ส่งข้อมูล ส่งสถานะและรับคำสั่งไปยังตัวchip
 - สัญญาณแอดเดรส จะถูกเก็บพักในรีจิสเตอร์แอดเดรส (Address Register) เพื่อนำไปอ่านรหัส (Decode) โดยวงจร Column Decoder เพื่อสร้างสัญญาณ Bit Line และ Row Decoder เพื่อสร้างสัญญาณ Word Line ในรูปที่ 7.1
 - สัญญาณข้อมูลที่อ่านหรือเขียน จะถูกเก็บพักในรีจิสเตอร์แคช (Cache Register) ชั่วคราวแล้วจึงย้ายเข้าหรือออกจากรีจิสเตอร์ข้อมูล (Data Register)
 - สถานะของการทำงานของแฟลชจะเก็บพักในรีจิสเตอร์สถานะ (Status Register) เพื่อให้วงจรเขื่อมต่อ (Interface Circuit) รับทราบ ได้แก่ สถานะ เป็นต้น
 - คำสั่งจะเก็บพักในรีจิสเตอร์คำสั่ง (Command Register) เพื่อป้อนให้กับวงจรควบคุม ได้แก่
 - * คำสั่งอ่านข้อมูลเพจ (Page Read)
 - * คำสั่งอ่านข้อมูลสุ่ม (Random Data Read)
 - * คำสั่งเขียนข้อมูลในเพจ (Program Page)
 - * คำสั่งลบ (Block Erase) เป็นต้น

ผู้อ่านจะสังเกตได้ว่าโครงสร้างภายในชิปแฟลช NAND มีลักษณะคล้ายกับหน่วยความจำ SRAM ในรูปที่ 5.9 DRAM ในรูปที่ 5.13 คือมี อะเรย์ของเซลหน่วยความจำเป็นหลัก และวงจรควบคุมการอ่านหรือเขียนข้อมูลลงไปในอะเรย์นี้ จะเข้าถึงโดยใช้แอดเดรสแทга (เลขแทга) และแอดเดรสคลอลัมบ์ (เลขคลอลัมบ์) เพื่อบ่งชี้ตำแหน่งเซลนั้นๆ ในอะเรย์ แต่การทำงานของหน่วยความจำแฟลชมีความซับซ้อน เนื่องจากมีการอ่านหลายชนิด การเขียนหลายชนิดและการลบข้อมูลตามที่กล่าวไว้ก่อนหน้า ที่มา: [Micron Technology \(2004\)](#)

7.2.2 การอ่านข้อมูล

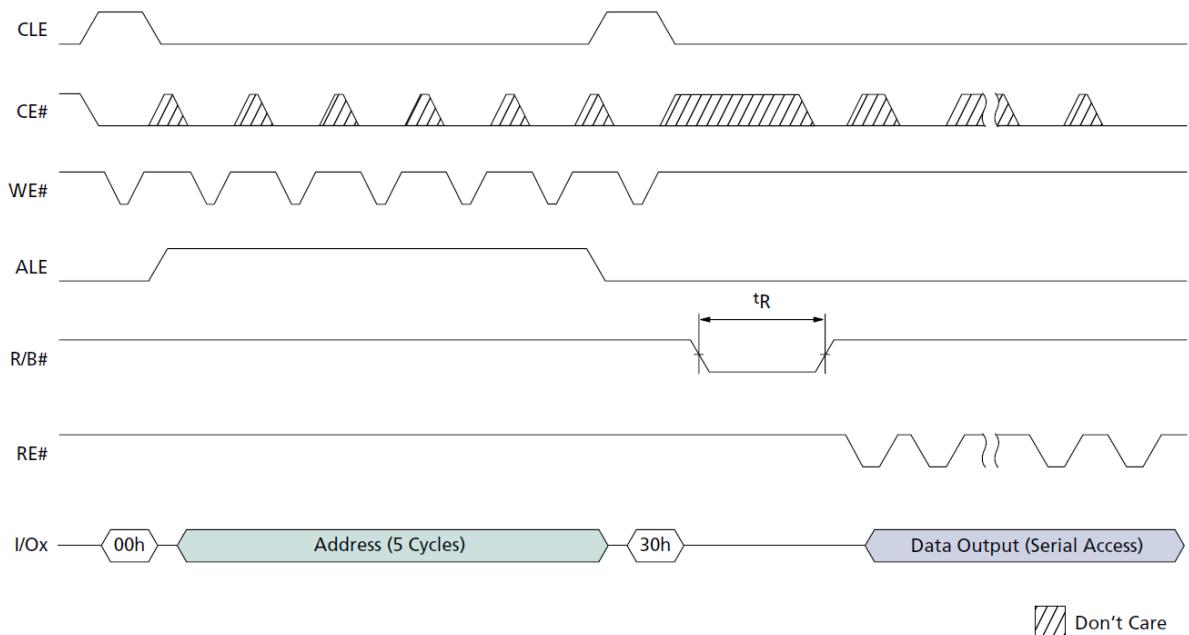


Figure 7.5: ไดอะแกรมเวลาของการอ่านข้อมูลแบบเพจ (Page Read) ของหน่วยความจำ Flash NAND ที่มา: [Micron Technology \(2004\)](#)

ในรูปที่ 7.5 การอ่านข้อมูลทั้งเพจ (Page Read) ซึ่งมีความประสิทธิภาพสูง เริ่มต้นโดยการเปลี่ยนสัญญาณเหล่านี้ แกนเวลาอยู่ในแนวนอนโดยเวลาเริ่มต้นจากทางซ้ายไปทางขวา

- สัญญาณ CLE เปลี่ยนจาก 0 เป็น 1 เพื่ออ่านค่าคำสั่งที่ปรากฏบนบัส I/O โดยคำสั่งจะมีรูปแบบ คำสั่ง 00h-Address (5 Cycles)-30h คือ การอ่านข้อมูลทั้งเพจ ณ แอดเดรสที่ได้รับ
- สัญญาณ CE# หรือ \overline{CE} เปลี่ยนจาก 1 เป็น 0 เพื่อส่งชิปให้เริ่มต้นทำงาน
- สัญญาณ WE# หรือ \overline{WE} จะมีการเปลี่ยนแปลงจาก 1 เป็น 0 สลับไปมา เพื่อเขียนคำสั่งและแอดเดรสทางขา I/Ox ไปเก็บพักในรีจิสเตอร์ต่างๆ
- สัญญาณ ALE จะเปลี่ยนแปลงจาก 0 เป็น 1 ในระหว่างที่บัส I/O รับแอดเดรสเป็นระยะเวลา 5 คากาเวลา (Cycles)
- สัญญาณ R/B# หรือ R/\overline{B} จะเปลี่ยนแปลงจาก 1 เป็น 0 ระยะเวลาหนึ่ง (t_R) เพื่อบ่งบอกว่าชิป มีสถานะยุ่ง (Busy) แล้วกลับไปเป็น 1 เพื่อป้องกันว่าชิปพร้อมแล้ว
- สัญญาณ RE# หรือ \overline{RE} จะเปลี่ยนแปลงจาก 1 เป็น 0 สลับไปมา เมื่อตรวจพบว่าสัญญาณ R/B# เปลี่ยนจาก 0 เป็น 1 เพื่ออ่านข้อมูลทางขา I/Ox โดยใช้ขอบขาขึ้นหรือขอบขาลงของสัญญาณ RE#

- สัญญาณ I/O[0:15] ทั้งหมด 16 ขา จะเปลี่ยนแปลงตามลำดับดังนี้
 - คำสั่ง 00h (00₁₆) จำนวน 1 ค疤เวลา
 - แอดдрес จำนวน 5 ค疤เวลา
 - คำสั่ง 00h (30₁₆) จำนวน 1 ค疤เวลา
 - ข้อมูล จำนวนหลายค疤เวลาตามขนาดของเพจข้อมูลโดยอ่านข้อมูลเรียงตามลำดับหมายเลขแอดdress (Serial Access)

7.2.3 หน่วยความจำแฟลชชนิดอื่นๆ

Table 7.1: ตารางเปรียบเทียบเซลล์หน่วยความจำ Flash ชนิด NAND (ซ้าย) และ NOR (ขวา) ตามโครงสร้างและเลเยอร์ของทรานซิสเตอร์ ที่มา: Choi (2010) หมายเหตุ F คือ ความกว้างของฟล็อกเกต (Float Gate) มีหน่วยเป็น นาโนเมตร

	NAND	NOR
Cell Array & Size	 Word line Unit Cell Source line $5F^2$	 Bit line Word line Contact Unit Cell Source line $10F^2$
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 แสดงการเปรียบเทียบเซลล์หน่วยความจำ Flash ชนิด NAND (ซ้าย) และ NOR (ขวา) เชิงโครงสร้าง ภาพแนวตัดขวาง (Cross Section) ภาพการจัดวางหรือเลเยอร์ (Lay Out) และขนาดของเซลล์ (Cell Size) ผู้อ่านจะเห็นได้ว่า แกะ Cell Array แสดงโครงสร้างการเชื่อมต่อของ Unit Cell เข้าด้วยกันโดยใช้การตัดกันของ Bit line และ Word line เป็นการระบุตำแหน่งของเซลล์ที่ต้องการอ่านหรือเขียน การเชื่อมต่อของ Unit cell มีลักษณะที่แตกต่างกัน คือ

- แฟลชชนิด NAND ต่อเซลล์เข้าด้วยกันแบบอนุกรม คล้ายกับวงจรเกท NAND ทำให้ประหยัดพื้นที่บนแผ่นซิลิโคนเท่ากับ $4F^2$ ต่อ Unit cell จึงทำให้ต้นทุนในการผลิตต่ำกว่าแฟลชชนิด AND และ NOR แต่ต้องใช้เวลาอ่านหรือเขียนหลาย Unit cell ตามลำดับ

- แฟลชชนิด NOR ต่อเซลล์เข้าด้วยแบบขนาน คล้ายกับวงจรเกท NOR โดยใช้ Word line แยกกัน การต่อ Unit cell แบบขนาน สามารถอ่านและเขียนข้อมูลแต่ละเซลล์ได้อิสระที่ลະเซลล์ แต่ใช้พื้นที่บน แผ่นซิลิกอนเท่ากับ $10F^2$ ต่อ Unit cell

ขนาดของพื้นที่เลเยอร์ (Layout) และภาพตัดขวาง (Cross Section) ของแต่ละเซลล์ สามารถคำนวณ ตามความกว้างคูณความยาวของแต่ละเซลล์ โดย F คือ ความกว้างของขาฟlot gate (Float Gate) ของ ทรานซิสเตอร์ ที่สามารถถูกดึงให้ใน การผลิตชิป ซึ่งมีขนาดเล็กลงเรื่อยๆ จาก 32 นาโนเมตร เหลือ 12 นาโน เมตร ที่มา: [wikipedia](#)

โครงสร้างของแต่ละ Unit cell ในแฟลชร้อมปัจจุบันพัฒนาให้เป็นเซลล์ชนิด 3D TLC (Triple Level Cell) และ QLC (Quad Level Cell) เพื่อเพิ่มความจุต่อเซลล์ให้มากขึ้น โครงสร้างชนิด TLC และ QLC สามารถเพิ่มความจุเป็นสามเท่า (Triple) และสี่เท่า (Quad) ตามลำดับ สามารถเก็บข้อมูลได้จำนวน 3 และ 4 บิตต่อเซลล์ข้อมูล 1 เซลล์ ผู้อ่านสามารถค้นควารายละเอียดเพิ่มเติมได้ที่ [embedded-computing.com](#) และ [wikipedia](#)

7.3 การ์ดหน่วยความจำ SD (Secure Digital)

เนื้อหาในหัวข้อนี้จะเสริมเพิ่มเติมและละเอียดมากขึ้น โดยอาศัยพื้นฐานจากหัวข้อที่ผ่านมา เนื่องจากโครงสร้างภายในของการ์ด SD เป็นหน่วยความจำแฟลชชนิด Nand และวงจรควบคุม โครงสร้างภายในของการ์ด SD ในรูปที่ 3.8 หัวข้อที่ 3.1.4 มีชิพหน่วยความจำแฟลชเป็นองค์ประกอบหลัก ที่มา: [wikipedia](#) การ์ดจะทำงานแบบซิงโครนัส (Synchronous) ตามสัญญาณคล็อกที่ไมโครโปรเซสเซอร์ส่งมา ซึ่งมักจะมีความถี่สูงสุดเป็นจำนวนเท่าของ 25 MHz เช่น 50 MHz 100 MHz เป็นต้น

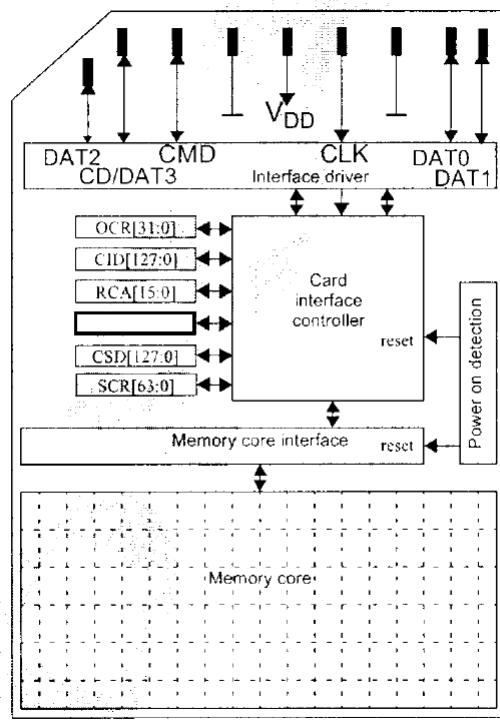


Figure 7.6: โครงสร้างภายในการ์ด SD Memory ที่มา: [SanDiskCorporation \(2003\)](#)

โครงสร้างภายในการ์ด SD Memory ในรูปที่ 7.6 ประกอบด้วย

- วงจรควบคุม Card Interface Controller เพื่อเชื่อมต่อกับไฮส์ท์ ทำงานร่วมกับรีจิสเตอร์ต่างๆ เหล่านี้
 - รีจิสเตอร์ CID[27:0] (Card Identification) ขนาด 28 บิต เพื่อเก็บหมายเลขประจำตัวการ์ด
 - รีจิสเตอร์ OCR[31:0] (Operation Condition Register) ขนาด 32 บิต เพื่อเก็บสถานะการทำงานของการ์ด
 - รีจิสเตอร์ RCA[15:0] (Relative Card Address) ขนาด 32 บิต เพื่อเก็บหมายเลขแอดเดรสของการ์ดซึ่งจะเปลี่ยนแปลงระหว่างที่ถูกดึงเข้าออกจากระบบ
 - รีจิสเตอร์ CSD[27:0] (Card Specific Data) ขนาด 28 บิต เพื่อเก็บสถานะจำเพาะของการ์ด
 - รีจิสเตอร์ SCR[63:0] (SD Configuration Register) ขนาด 64 บิต เพื่อเก็บการตั้งค่าพิเศษประจาระจำตัวการ์ด

- Memory Core Interface เพื่อเชื่อมต่อกับหน่วยความจำ
- อะเรย์เซลหน่วยความจำชนิดแฟลช ตามขนาดของความจุที่ต้องการใช้

การเชื่อมต่อ CPU กับการ์ด SD นี้สามารถเลือกได้โดยใช้การเชื่อมต่อโหมด SPI และโหมด SDIO การเชื่อมต่อกับการ์ด SDIO สามารถทำได้อีกสองโหมด ก็คือ การเชื่อมต่อโหมด SD 1 บิต และ การเชื่อมต่อโหมด SD 4 บิต ซึ่งมีประสิทธิภาพเร็วที่สุด เนื่องจากสามารถอ่าน/เขียนได้พร้อมกัน 4 บิต หัวข้อที่ [3.1.4](#) แล้ว

Table 7.2: หมายเลขขา ชื่อ และ วัตถุประสงค์ของ SD ในโหมดการทำงานโหมด SD 4 บิต ที่มา: [wikipedia](#)

ขา	ชื่อ	วัตถุประสงค์
1	DAT2	Data บิตที่ 2
2	DAT3/CD	Data บิตที่ 3 หรือ Card Detect
3	CMD	ขารับคำสั่ง (Command)
4	VDD	ขั้วบวกแหล่งจ่ายไฟ
5	CLK	สัญญาณคลื่อ
6	VSS	ขากราวด์แหล่งจ่ายไฟ
7	DAT0	Data บิตที่ 0
8	DAT1	Data บิตที่ 1

ตารางที่ [7.2](#) ชี้พิยุ ไมโครโปรเซสเซอร์ และ ไมโครคอนโทรลเลอร์ (MCU) จะควบคุมการทำงานของ SD โดยส่งคำสั่งต่างๆ ผ่านทางขา CMD ซึ่งประกอบด้วยคำสั่ง (CMD: Command) สำคัญๆ ในโหมด SD ดังต่อไปนี้

- CMD17: Read Single Block
- CMD18: Read Multiple Block
- CMD24: Write Single Block
- CMD25: Write Multiple Block
- CMD32: Erase Block Start
- CMD33: Erase Block End
- CMD38: Erase

7.3.1 การอ่านข้อมูลจากการ์ด

ไฮสท์ส่งโทเก็น Command CMD18 ไปทำการ์ด การ์ดจะตอบกลับด้วยโทเก็น Response ไปยังไฮสท์ บล็อกข้อมูล Data Block ไปยังไฮสท์ ไปเรื่อยๆ จนถึง Data Block สุดท้ายเมื่อครบตามจำนวนที่ต้องการ

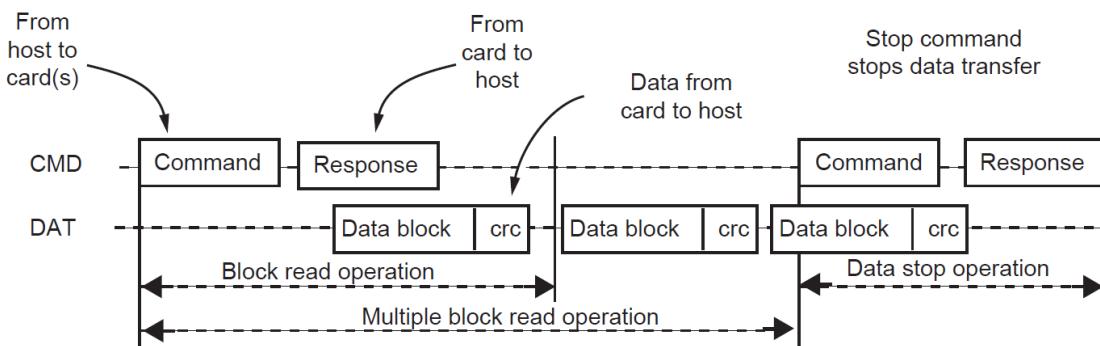


Figure 7.7: ໄດ້ອະແກນເວລາຂອງການອ່ານຂໍ້ມູນຈຳນວນຫລາຍບັນຈຸກຈາກກາຣົດ SD Memory ທີ່ມາ: [San-DiskCorporation \(2003\)](#)

හລັງຈາກນັ້ນ ໂອສທ໌ຈະສ່ວໂທເກີນ Stop Command ເພື່ອຫຼຸດຂບວນການ ແລະກາຣົດຈະຕອບກັບດ້ວຍໂທເກີນ Response ໄປຢັ້ງໂອສທ໌

หากເປັນ Command CMD17 ຈະໝາຍຖື່ງ ການອ່ານເພີ່ມບັນຈຸກເດືອຍ ແລະເສົ່າງສິນເຮົວກ່າວການອ່ານຈຳນວນຫລາຍບັນຈຸກ ໂດຍໄມ່ຕ້ອງມີກາຣໂຕ້ຕອບຮະຫວ່າງໂອສທ໌ແລກາຣົດອີກຮອບ

7.3.2 ການເຂີຍນ້ຳຂໍ້ມູນຈາກກາຣົດ

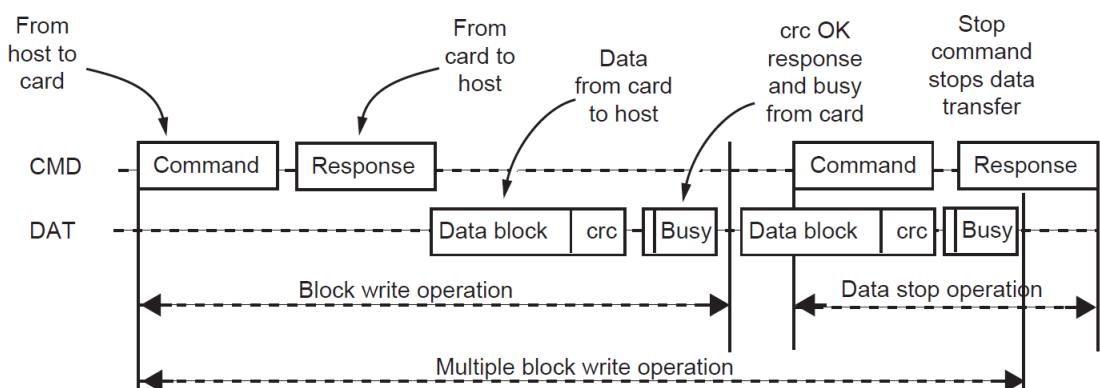


Figure 7.8: ໄດ້ອະແກນເວລາຂອງການເຂີຍນ້ຳຂໍ້ມູນຈຳນວນຫລາຍບັນຈຸກຈາກກາຣົດ SD Memory ທີ່ມາ: [San-DiskCorporation \(2003\)](#)

ໂອສທ໌ສ່ວໂທເກີນ Command CMD25 ໄປກາຣົດ ກາຣົດຈະຕອບກັບດ້ວຍໂທເກີນ Response ໄປຢັ້ງໂອສທ໌ ບັນຈຸກຂໍ້ມູນ Data Block ໄປຢັ້ງໂອສທ໌ ໄປເຮືອຍໆ ຈນຖື່ງ Data Block ສຸດທ້າຍເມື່ອຄຽບຕາມຈຳນວນທີ່ຕ້ອງການ ໄລັງຈາກນັ້ນ ໂອສທ໌ຈະສ່ວໂທເກີນ Stop Command ເພື່ອຫຼຸດຂບວນການ ແລະກາຣົດຈະຕອບກັບດ້ວຍໂທເກີນ Response ໄປຢັ້ງໂອສທ໌

หากເປັນ Command CMD24 ຈະໝາຍຖື່ງ ການເຂີຍນ້ຳເພີ່ມບັນຈຸກເດືອຍ ແລະເສົ່າງສິນເຮົວກ່າວການເຂີຍຈຳນວນຫລາຍບັນຈຸກ ໂດຍໄມ່ຕ້ອງມີກາຣໂຕ້ຕອບຮະຫວ່າງໂອສທ໌ແລກາຣົດອີກຮອບ

7.4 โซลิดสเตตดิสก์ (Solid-State Disk: SSD)

SSD Controller

SATA
and
Power

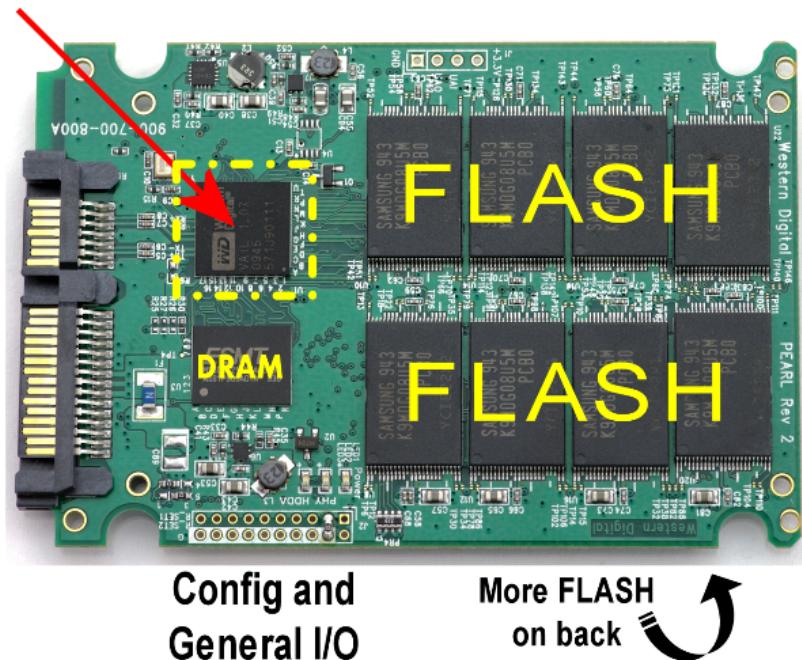


Figure 7.9: แผ่นวงจรพิมพ์ภายในอุปกรณ์ SSD ชนิด SATA III ประกอบด้วยชิพหน่วยความจำแฟลช ไดนามิกแรม คอนโทรลเลอร์สำหรับควบคุม ที่มา: thatoldnews.site

ความจุของ SSD มีแนวโน้มเพิ่มสูงขึ้น ทำให้ SSD มีขนาดเริ่มต้นตั้งแต่ 120-128 กิกะไบท์ขึ้นไป แนวโน้มของ SSD ต้นทุน/ความจุจะถูกลงเรื่อยๆ จนใกล้เคียงฮาร์ดดิสก์ในอนาคต โดยองค์ประกอบหลักที่สำคัญ คือ หน่วยความจำแฟลช NAND ที่มีความจุต่อชิปเพิ่มสูงขึ้นไปอีก และใช้หน่วยความจำ DRAM เพื่อทำหน้าที่เป็นแคช หรือ บัฟเฟอร์ เพื่อเพิ่มประสิทธิภาพการทำงานให้รวดเร็วขึ้น SSD ใช้ชื่อที่ใกล้เคียงกับฮาร์ดดิสก์ไดร์ฟ เพื่อความหมายที่ใกล้เคียงกัน

โครงสร้างภายในของ SSD ในรูปที่ 7.9 แผ่นวงจรพิมพ์ภายในอุปกรณ์ SSD ชนิด SATA III ส่วนเชื่อมต่อกับเครื่อง ไม่ครอบคลุม บัฟเฟอร์หรือแคช และหน่วยความจำแฟลช

- ชิพหน่วยความจำแฟลช NAND ความจุสูงเรียงตัวต่อเนื่องกันจนได้ความจุมากพอตามที่ผู้ซื้อต้องการ การจัดเรียงทั้งด้านบน (Channel 0) และด้านล่าง (Channel 1) ของแผ่นวงจรพิมพ์หลัก ของหน่วยความจำแฟลช และอาศัยชิพหน่วยความจำไดนามิกแรมทำหน้าที่เป็นแคช มีลักษณะตัวลังคล้ายกับของชิพหน่วยความจำแฟลชในรูปที่ 7.3
- วงจรควบคุม (Controller) นิยมใช้ไมโครคอนโทรลเลอร์และเฟิร์มแวร์ทำงาน โดยเฉพาะไมโครคอนโทรลเลอร์ระดับ ARM รุ่น Cortex R ซึ่ง R ย่อมาจากคำว่า Real Time เหมาะกับการควบคุมบางส่วนของรถยนต์ แขนหุ่นยนต์ ฮาร์ดไดร์ฟ เครื่องมือทางการแพทย์ อุปกรณ์สื่อสาร โมเด็ม (Modem) เป็นต้น ที่มา: anandtech.com

- การเชื่อมต่อกับไฮสตรีท (Host Interface) การเชื่อมต่อที่ได้รับความนิยม คือ SATA และ M.2 ส่วน เชื่อมต่อกับเครื่องมีสองชนิดหลัก คือ SATA III และ M.2 SATA III ได้รับความนิยมในระยะแรก เนื่องจากเป็นมาตรฐานของฮาร์ดดิสก์ ในขณะที่ M.2 ได้รับความนิยมเพิ่มมากขึ้นเรื่อยๆ เนื่องจากประสิทธิภาพสูงกว่า SATA III ศึกษาเพิ่มเติมได้ที่ pcworld.com

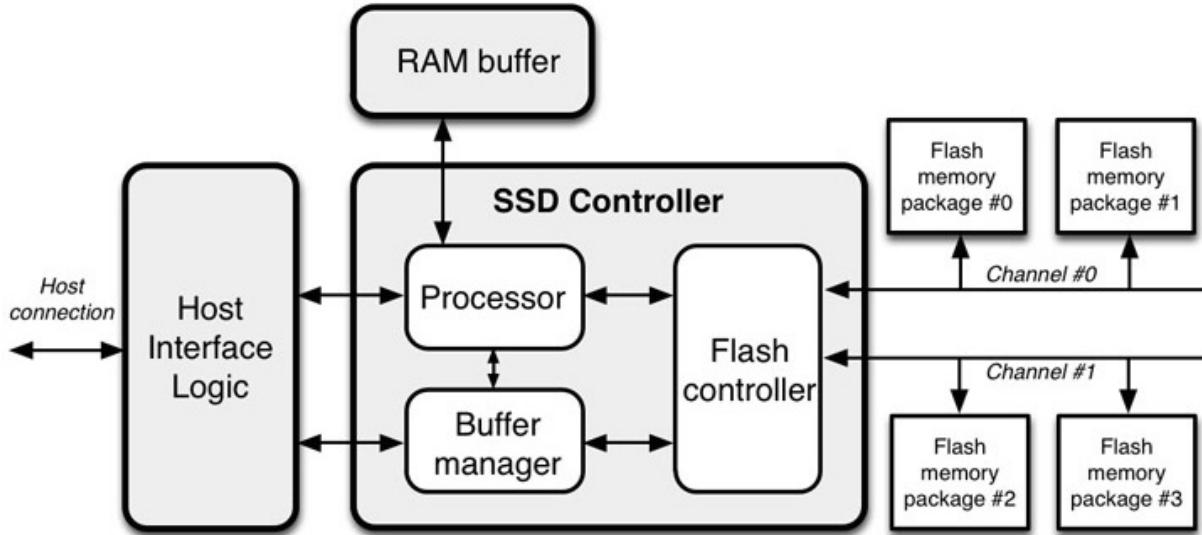


Figure 7.10: บล็อกไซด์แกรมภายใน SSD ประกอบด้วย ส่วนเชื่อมต่อกับเครื่อง ไมโครคอนโทรลเลอร์ บัฟเฟอร์ และหน่วยความจำแฟลช ที่มา: codecapsule.com

ผู้อ่านสามารถศึกษาบล็อกไซด์แกรมภายใน SSD เพิ่มเติมได้ในรูปที่ 7.10 ประกอบด้วย

- ไมโครprocessor สำหรับรับเฟิร์มแวร์ควบคุมการทำงานภายในและเชื่อมต่อกับคอมพิวเตอร์ไฮสตรีท ดังนั้น ผู้อ่านควรตรวจสอบผู้ผลิตเป็นระยะๆ ว่ามีการอัพเดทเฟิร์มแวร์ของ SSD รุ่นที่ใช้หรือไม่
- แฟลชคอนโทรลเลอร์ (Flash Controller) สำหรับควบคุมการอ่าน/เขียนข้อมูลหน่วยความจำแฟลช คล้ายกับการทำงานของหน่วยความจำ SD ซึ่งกล่าวในหัวข้อที่ 7.2
- บัฟเฟอร์ (Buffer) อาศัยหน่วยความจำ DRAM เป็นบัฟเฟอร์เพื่อพักเก็บข้อมูลชั่วคราวระหว่างที่ เชื่อมต่อกับคอมพิวเตอร์ไฮสตรีท ทำให้การอ่าน/เขียนมีค่าเวลาหน่วงต่ำลง
- การบริหารจัดการบัฟเฟอร์ (Buffer Management) เพื่อให้ใช้หน่วยความจำ DRAM ได้เต็มประสิทธิภาพ และใช้จำนวนคุ้มค่า ทำให้ต้นทุนการผลิตต่ำลง ประเด็นที่น่าสนใจ คือ การย้ายข้อมูลจากบัฟเฟอร์ไปเก็บในหน่วยความจำ แฟลช หากเกิดกรณีไฟดับกระแทกหนักโดยไม่มีหน่วยสำรองไฟ หรือ UPS

7.5 ฮาร์ดดิสก์ไดร์ฟ (Hard Disk Drive: HDD)

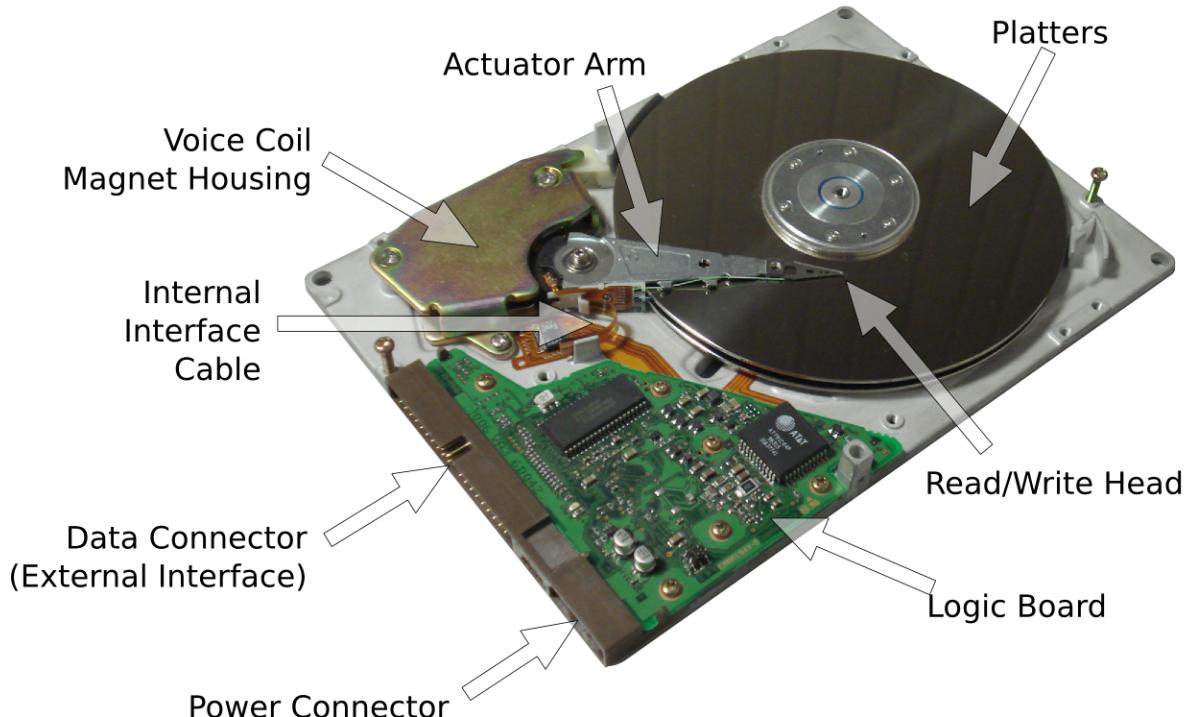


Figure 7.11: โครงสร้างและองค์ประกอบของอุปกรณ์ฮาร์ดดิสก์ไดร์ฟ (HDD)

อุปกรณ์เก็บรักษาข้อมูลที่ได้รับความนิยมจากในอดีต และพัฒนามาเป็นเวลาระยะนาน แผ่นจานแม่เหล็กหมุน มีเส้นผ่าศูนย์กลางสองขนาดที่นิยมผลิต คือ 2.5 นิ้ว และ 3.5 นิ้ว หมุนด้วยความเร็วสูงตั้งแต่ 5400 ถึง 10000 รอบต่อนาที จุดเด่นของหน่วยความจำนิดสารแม่เหล็ก คือ ความจุข้อมูลที่มากกว่า เริ่มต้นที่หลายร้อย吉ิกะไบท์จนถึงหลายเทอร่าไบท์ (Tera Byte) โดย 1 เทอร่าไบท์ ประมาณเท่ากับ 1000 吉ิกะไบท์ ทำให้ราคาต่อความจุต่ำลง ต้นทุนโดยรวมจึงถูกลง มีอายุการใช้งานที่ยาวนานพอสมควร

จุดอ่อน คือ ประสิทธิภาพด้านความเร็วที่ต่ำกว่า ฮาร์ดดิสก์ต้องการปริมาณและน้ำหนักสำหรับจัดเก็บมากกว่า ทำให้โครงสร้างใหญ่ขึ้น ตัวอุปกรณ์รองรับแรงกระแทกมากเมื่อเทียบกับการตกหล่นหรือแผ่นดินไหว ได้น้อยกว่า บริโภคพลังงานสูงกว่าโซลิดสเตติดิสก์ เนื่องจากมีการหมุนจานแม่เหล็กเกือบตลอดเวลา จึงเกิดความร้อนแ芳มากกว่า จึงต้องอาศัยอุปกรณ์อื่นๆ เช่น พัดลมช่วยระบายความร้อน หรือ เครื่องปรับอากาศในห้องคอมพิวเตอร์และศูนย์ข้อมูล

7.5.1 โครงสร้างของฮาร์ดดิสก์เชิงกายภาพ

โครงสร้างและองค์ประกอบของอุปกรณ์ฮาร์ดดิสก์ไดร์ฟ (HDD) เชิงกายภาพใน รูปที่ 7.11 ประกอบด้วย

- แผ่นจานแม่เหล็ก (Platter) ซ้อนกันหลายชั้น เก็บข้อมูลบนจานแม่เหล็ก เพื่อเพิ่มความจุโดยรวม และหมุนพร้อมกันด้วยความเร็ว 5,400 - 10,000 รอบต่อนาที 90-134 รอบต่อวินาที

- แขนกล (Actuator Arm) มีหัวอ่าน/เขียนข้อมูลสำหรับแต่ละจานยึดอยู่บนแขนกลเดียวกัน แผ่นละ 2 หัว แต่ละหัวมีหัวอ่าน/เขียน 1 คู่ ด้านบนและด้านล่าง โปรดสังเกตขนาดของหัวอ่าน/เขียนที่เล็กมาก
- แผ่นวงจรควบคุม (Controller Board หรือ Logic Board) ประกอบด้วยไมโครคอนโทรลเลอร์ เช่น ไมโครคอนโทรลเลอร์ ARM ตระกูล Cortex R 4 และอื่นๆ ทำหน้าที่ควบคุมการทำงานโดยรวม โดย ARM จะมีส่วนแบ่งการตลาดสูงมากจากการสำรวจในปี คศ 2010 ในตารางที่ 4.10
- การเชื่อมต่อ กับ Host Interface และเชื่อมต่อ กับเครื่องคอมพิวเตอร์ผ่านทาง External Interface ได้แก่ Parallel ATA ([wikipedia](#)) Serial ATA ([wikipedia](#)) NVM Express ([wikipedia](#)) เป็นต้น ขนาดบัฟเฟอร์หรือแคช หน่วยเป็น MB เพื่อพักเก็บข้อมูลล่าสุดที่ได้ทำการอ่านหรือเขียนไว้ก่อน ทำให้ระยะเวลาการอคุยสั้นลง รูปที่ 7.12

7.5.2 การจัดเรียงข้อมูลในฮาร์ดดิสก์

แผ่นจานแม่เหล็กแบ่งเป็นหลายๆ แทร็ค หรือ วงรอบ แต่ละแทร็คจะมีจำนวนเซ็คเตอร์เท่าๆ กัน พื้นที่หนึ่งเซ็คเตอร์ มีความจุ 512 ไบต์เสมอ โดยจะแบ่งพื้นที่ในแนวรัศมีจากจุดศูนย์กลาง นวยังขอบจานตามรูปที่ 7.12

ไซลินเดอร์ (Cylinder) คือ แทร็คที่อยู่บนจานแม่เหล็กต่างๆ และอยู่ห่างจากจุดศูนย์กลางเท่ากัน เรียngตัวกันเป็นทรงกระบอก โดยจะนับจากไซลินเดอร์หรือแทร็คหมายเลข 0 ซึ่งอยู่ขอบนอกสุดของ จานแม่เหล็ก โดยจะนับจากขอบนอกสุดเข้าสู่จุดศูนย์กลาง

บล็อก หรือ คลัสเตอร์ประกอบด้วย เซ็คเตอร์ที่ต่อเนื่องกัน จำนวน 2^n เซ็คเตอร์เสมอ เช่น 2 4 8 .. เซ็คเตอร์ ในแทร็คเดียวกัน โดยระบบปฏิบัติการจะกำหนดจำนวนเซ็คเตอร์ ที่ต้องการ ยกตัวอย่าง เช่น คลัสเตอร์ขนาด 4096 ไบต์ต้องการพื้นที่จำนวน 8 เซ็คเตอร์

หมายเลขบล็อก LBA: Logical Block Addressing คือ การตั้งหมายเลขบล็อกที่กล่าวมาก่อนหน้านี้ ให้เรียงตัวตามหมายเลขไซลินเดอร์ หมายเลขหัวอ่าน และหมายเลขแทร็ค ทำให้ง่ายต่อการบริหารจัดการ และเป็นพื้นฐานของระบบบริหารจัดการไฟล์ ที่มา: [wikipedia](#)

ระบบจะจองพื้นที่บนฮาร์ดดิสก์อย่างน้อย 1 บล็อกหรือ 1 คลัสเตอร์ให้กับไฟล์หนึ่งไฟล์ ยกตัวอย่าง เช่นไฟล์ขนาด 800 ไบต์ ต้องการใช้พื้นที่ 8 เซ็คเตอร์ เช่นเดียวกับไฟล์ขนาด 4096 ไบต์ ในอุปกรณ์เก็บรักษาข้อมูลใดๆ ผู้ใช้ทั่วไปสามารถเก็บข้อมูลหรือแอพพลิเคชันในรูปของไฟล์เท่านั้น ในด้านการบริหารจัดการไฟล์ ผู้ใช้สามารถคัดลอกหรือสำเนา (Copy) ลบ (Delete หรือ Remove) ย้าย (Move) คืน (Recover) ไฟล์ ต่างๆ ได้ผ่านระบบปฏิบัติการ ในด้านการบริหารจัดการพื้นที่ ผู้ใช้สามารถจัดหมวดหมู่ไฟล์โดยการ สร้างไฟลเดอร์ ตั้งชื่อ คัดลอก ย้าย ลบ คืนไฟลเดอร์ต่างๆ เช่นกัน

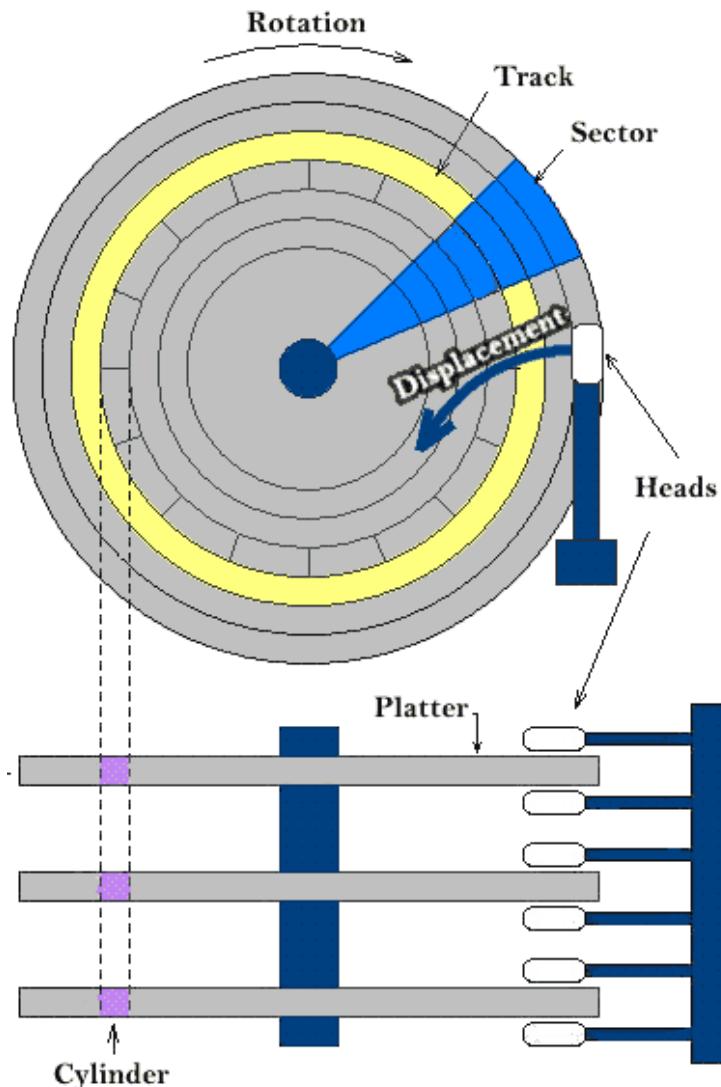


Figure 7.12: โครงสร้างของฮาร์ดดิสก์ไดร์ฟแบ่งเป็นไซลินเดอร์ แทร็คและเซกเตอร์ ที่มา: computable-minds.com

7.5.3 ความจุและประสิทธิภาพของฮาร์ดดิสก์

ความจุของฮาร์ดดิสก์ สามารถคำนวณได้จาก ผลคูณของ ความจุต่อหน่วยเซกเตอร์ จำนวนเซกเตอร์ต่อแทร็ค จำนวนไซลินเดอร์ และจำนวนหัวอ่าน ยกตัวอย่างเช่น 512 ไบท์ต่อเซกเตอร์ \times 63 เซกเตอร์ต่อแทร็ค \times 1024 ไซลินเดอร์ \times 256 หัว เท่ากับ 8,455,716,864 bytes หรือ 8.4×10^9 ไบท์โดยประมาณ ทำให้ผู้ผลิตใช้หน่วยเป็น 8.4 จิกะไบท์

อายุการใช้งานของฮาร์ดดิสก์วัดจากจำนวนชั่วโมงที่เปิดใช้งาน ซึ่งจะมีความแตกต่างจากการวัดอายุของหน่วยความจำแฟลชรีอมที่วัดจากจำนวนครั้งที่เขียนหรือลบข้อมูล ทำให้ฮาร์ดดิสก์มีต้นทุนการใช้งานต่ำกว่าความจุและต่ออายุที่คุ้มค่ากว่า

การวัดประสิทธิภาพของฮาร์ดดิสก์ อาศัยการวัดเวลาการเข้าถึง (T_{acc} , Access Time/Latency) หน่วยเป็น มิลลิวินาที ประกอบด้วย ช่วงเวลาการรอเฉลี่ยให้ตำแหน่งที่ต้องการอ่านหมุนมาอยู่หัวอ่าน เรียกว่า **Rotation Latency** T_{rotate} ช่วงเวลาการขยับหัวอ่านมายังแทร็คที่ต้องการ T_{head} และช่วงเวลาการ

ถ่ายโอนข้อมูล (T_{trans} , Transfer Time)

$$T_{acc} = T_{rotate} + T_{head} + T_{transf} \quad (7.1)$$

ดังนั้น เวลาการเข้าถึงจึงมีความสัมพันธ์กับความเร็วรอบในการหมุนของajanแม่เหล็ก

$$T_{rotate} = \frac{0.5}{V_{rotate}} \quad (7.2)$$

โดยแพร่องผันกับ V_{rotate} หรือ ความเร็วรอบในการหมุนของajan หน่วยเป็นรอบต่อวินาที (Round per Minute: RPM) และตำแหน่งของแทร็คที่ต้องขยับหัวอ่านไป ประสิทธิภาพการอ่านและการเขียน จะขึ้น กับตำแหน่งของข้อมูล เช่นเดียวกับการอ่านและเขียนข้อมูลของหน่วยความจำแฟลช ประสิทธิภาพการอ่านเขียนข้อมูลที่เรียงตัวต่อเนื่องจะดีกว่าการอ่านเขียนข้อมูลที่ไม่เรียงตัว ถ้ามีการจัดเรียงไฟล์ข้อมูลในแต่ละเซ็คเตอร์ให้ต่อเนื่องกัน หรือเรียกว่า **Defragmentation** จะทำให้เวลาการเข้าถึง (Access Time) สั้นลง รวมถึงการจัดเรียงลำดับการอ่านเขียนข้อมูลบนดิสก์ (Disk Scheduling) จะช่วยให้ประสิทธิภาพเพิ่มขึ้น ผู้อ่านสามารถศึกษาอัลกอริทึมได้ที่ geeksforgeeks.org องค์ประกอบสุดท้าย คือ T_{transf} ขึ้นอยู่ กับชนิดการซื้อมต่อ กับคอมพิวเตอร์ไฮส์ เน้น SATA จะมีการพัฒนาประสิทธิภาพสูงขึ้นเป็น SATA III ตามที่ได้กล่าวไว้ก่อนหน้า

7.6 สรุปท้ายบท

ระบบไฟล์และอุปกรณ์เก็บรักษาข้อมูลจะต้องประสานงานกัน เพื่อให้เครื่องเนล ผู้ใช้งานและแอพพลิเคชันอื่นๆ สามารถบริหารจัดการไฟล์โปรแกรม ไฟล์ข้อมูลต่างๆ ได้อย่างถูกต้อง และมีประสิทธิภาพสอดคล้องกับภารกิจของระบบคอมพิวเตอร์ นอกจากนี้ ผู้อ่านจะสังเกตเห็นว่า ขนาดหรือความจุของบล็อกข้อมูลในอุปกรณ์เก็บรักษาข้อมูล เน้น หน่วยความจำแฟลชรีอัม จะมีขนาดเท่ากับ 2^{11} หรือ 2048 ไบท์ ฮาร์ดดิสก์ จะมีขนาดเท่ากับ 2^9 หรือ 512 ไบท์ สอดคล้องกับ ขนาดของบล็อกข้อมูลในระบบบริหารจัดการไฟล์ และขนาดของเพจข้อมูลในหน่วยความจำสมเมื่อน ซึ่งจะมีขนาดเป็นจำนวนเท่าของสองเสมอ และสำหรับลีนุกซ์ มีขนาดเท่ากับ 4096 หรือ 2^{12} ไบท์

ตารางที่ 7.3 เป็นการเปรียบเทียบประสิทธิภาพของอุปกรณ์เก็บรักษาข้อมูล ประกอบด้วยชิพหน่วยความจำแฟลช NAND อุปกรณ์ SSD และอุปกรณ์ HDD ในด้านต่างๆ โดยผู้อ่านจะต้องใช้ปีที่ผลิตเป็นหลักยึดในการทำความเข้าใจ เนื่องจากปีที่ผลิตจะมีผลต่อประสิทธิภาพ ความจุและกำลังไฟฟ้าสูงสุด

อุปกรณ์เก็บรักษาข้อมูลจะเชื่อมต่อกับหน่วยความจำผ่านวงจรด้านอินพุท/เอาท์พุท โดยใช้ กลไก Direct Memory Access และ กลไกการทำ Interrupt ในชั้น Physical ร่วมกับกลไก Memory Mapped File ในระดับสูงขึ้น

7.7 คำถามท้ายบท

1. จงเปรียบเทียบการอ่านหรือเขียนของอุปกรณ์เก็บรักษาข้อมูลชนิดต่างๆ ได้แก่ การ์ด SD, SSD, HDD ในแต่ละอย่าง

Table 7.3: การเปรียบเทียบประสิทธิภาพของอุปกรณ์เก็บรักษาข้อมูลชนิดต่างๆ

อุปกรณ์	แฟลช NAND	SSD	HDD
ผู้ผลิต หมายเลขโมเดล ที่มา: ปี ค.ศ. ความจุ (จิกะไบท์)	Micron MT29F2G08AABWP micron.com 2004 0.25	Micron MTFDDAK120MAV micron.com 2013 120	Western Digital 5K1000 hgst.com 2016 1000
การอ่าน			
- ค่าเวลาเฉลี่ย	30 ns	160 μ s	5.5 ms
- ค่าเวลาสูงสุด	25 μ s	5 ms	-
การเขียน			
- ค่าเวลาเฉลี่ย	300 μ s	40 μ s	5.5 ms
- ค่าเวลาสูงสุด	2 ms	25 ms	-
โวลต์เต็มสูงสุด (โวลท์) กำลังไฟสูงสุด	4.6 23 มิลลิวัตต์	5.0 150 มิลลิวัตต์	5.0 1.6 วัตต์

- ขนาดของข้อมูลขึ้นตั้งแต่ที่สามารถอ่านหรือเขียนได้ หน่วยเป็นไบท์
- ระยะเวลาที่ต้องรอ (Latency) และช่วงเวลาอย่างอื่น

- จงเปรียบเทียบกำลังไฟสูงสุดของหน่วยความจำชนิดต่างๆ ได้แก่ Flash ROM, การ์ด SD, SSD ที่ความจุเท่ากัน เท่าที่จะหาข้อมูลได้
- เหตุให้การอ่านหรือเขียนข้อมูลในอุปกรณ์เก็บรักษาข้อมูลจึงต้องอาศัยกลไกของ DMA และการขัดจังหวะทำงานร่วมกัน
- จงเปรียบเทียบการเชื่อมต่อชนิด PATA และ SATA ในแง่ประสิทธิภาพ
- จงเปรียบเทียบการเชื่อมต่อชนิด SATA II และ SATA III ในแง่ประสิทธิภาพ
- จงเปรียบเทียบการเชื่อมต่อชนิด SATA III และ NVM Express ในแง่ประสิทธิภาพ
- จงเปรียบเทียบหน่วยความจำ SD Class ต่างๆ ในแง่ประสิทธิภาพ

Appendix A

การทดลองที่ 1 ข้อมูลและคณิตศาสตร์ในคอมพิวเตอร์

การทดลองนี้เป็นการทดลองความเข้าใจและแบบฝึกหัดเสริมของเนื้อหาในบทที่ 1 เนื่องจากจำนวนบิทข้อมูลที่ยาวขึ้นจำเป็นต้องใช้โปรแกรมคอมพิวเตอร์ช่วยคำนวณแทน โดยมีวัตถุประสงค์ ดังต่อไปนี้

- เพื่อให้เข้าใจการแปลงและคณิตศาสตร์สำหรับเลขจำนวนเต็มฐานสองชนิดไม่มีเครื่องหมายและมีเครื่องหมายแบบ 2-Complement
 - เพื่อให้เข้าใจการแปลงและคณิตศาสตร์สำหรับเลขทศนิยมฐานสองมาตรฐาน IEEE 754 Single Precision และ Double Precision
 - เพื่อให้เข้าใจรหัส ASCII และ Unicode สำหรับข้อมูลตัวอักษร
- นอกจากเนื้อหาในบทที่ 1 แล้ว ผู้อ่านสามารถศึกษาเว็บเพจเพิ่มเติม เพื่อทำความเข้าใจอย่างลึกซึ้งได้แก่
- https://www.tutorialspoint.com/cprogramming/c_data_types.htm
 - <https://www.ntu.edu.sg/home/ehchua/programming/java/datarrepresentation.html>

ผู้อ่านจะพบว่าเนื้อหาในเว็บที่สองเป็นการสอนพื้นภาษา Java ใช้งานข้อมูลเป็นเลขฐานสองเหมือนกับภาษา C/C++ ในเว็บที่สอง การทดลองจะครอบคลุมเนื้อหาตามทฤษฎี โดยจะเริ่มจากเลขจำนวนเต็ม เลขทศนิยม และตัวอักษรตามลำดับ

A.1 การแปลงและคณิตศาสตร์สำหรับเลขฐานสองจำนวนเต็ม

A.1.1 การทดลอง

เนื่องจากการแปลงเลขฐานสิบเป็นฐานสอง Unsigned สามารถแปลงได้ ด้วยเครื่องคิดเลขทางวิทยาศาสตร์ ทั่วไป ดังนั้น การทดลองนี้จะเน้นที่การแปลงเป็นเลขฐานสองชนิดมีเครื่องหมายแบบ 2 Complement สอดคล้องกับเนื้อหาในหัวข้อที่ 1.2 โดยผ่านเว็บเบราว์เซอร์ที่ผู้อ่านสนใจ กรอกหรือคลิกที่ ชื่อลิงค์ต่อไปนี้ http://www.free-test-online.com/binary/signed_converter.html เมื่อเว็บเพจปรากฏขึ้น ขอให้ผู้อ่านปฏิบัติตามการทดลอง ดังนี้

1. คลิกเลือกที่ปุ่ม Signed และวิจัยกรอกเลข -123 ลงในกล่องข้อความ ดังรูปที่ A.1

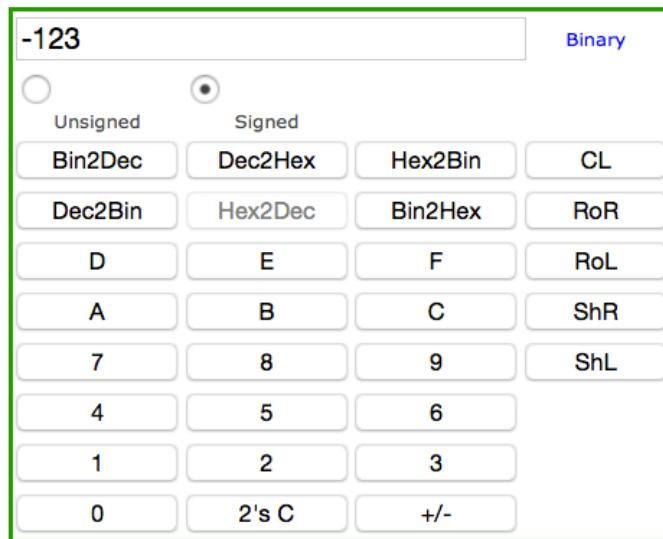


Figure A.1: กรอกเลข -123 ลงในกล่องข้อความ และคลิกเลือกที่ปุ่ม Signed เพื่อให้เป็นเลขฐานสองชนิด Signed

คล้ายเครื่องคิดเลข ประกอบด้วยปุ่มต่างๆ ดังนี้

- 'Bin2Dec' 'Dec2Bin' สำหรับแปลงเลขฐานสองเป็นฐานสิบไปและกลับ
- 'Dec2Hex' 'Hex2Dec' สำหรับแปลงเลขฐานสิบเป็นฐานสิบหกไปและกลับ
- 'Hex2Bin' 'Bin2Hex' สำหรับแปลงเลขฐานสองเป็นฐานสิบหกไปและกลับ
- ปุ่ม 0-9 และ A-F สำหรับกรอกตัวเลขฐานสิบและฐานสิบหก
- CL (Clear) สำหรับล้างค่าในกล่องข้อความให้เป็น 0
- RoR (Rotate Right) และ RoL (Rotate Left) สำหรับหมุนเลขที่อยู่ในกล่องข้อความ
- ShR (Shift Right) และ ShL (Shift Left) โดยป้อนเลข 0 เข้ามาแทน
- 2's C (2's Complement) สำหรับแปลงเลขฐานสองให้เป็นค่า 2's Complement

- +/- สำหรับกลับเครื่องหมายของตัวเลขฐานสิบในกล่องข้อความ
2. กดปุ่มเครื่องหมาย 'Bin2Dec' เพื่อให้เป็นเลขฐานสองชนิด Signed ดังรูปที่ A.2

The screenshot shows a binary calculator interface. At the top, there is a text input field containing the binary number **11111111111111110000101**. To the right of the input field, the word **Binary** is displayed. Below the input field, there is a radio button group with **Unsigned** and **Signed** options, where **Signed** is selected. Below the radio buttons is a grid of buttons labeled with letters (A-F), digits (0-9), and symbols (+/-). In the bottom-left corner of the grid, there is a button labeled **2's C**. Above the grid, there are four primary conversion buttons: **Bin2Dec**, **Dec2Hex**, **Hex2Bin**, and **CL**.

Figure A.2: ผลลัพธ์จากการแปลงเลข -123 ให้เป็นเลขฐานสองชนิด Signed 2-Complement ความยาว 24 บิต

3. กดปุ่มเครื่องหมาย 'Bin2Hex' เพื่อแปลงเลขฐานสองที่ได้ให้เป็นเลขฐานสิบหกชนิด Signed ตามรูปที่ A.3

The screenshot shows a binary calculator interface. At the top, there is a text input field containing the binary number **11111111111111110000101**. To the right of the input field, the word **HEX** is displayed. Below the input field, there is a radio button group with **Unsigned** and **Signed** options, where **Signed** is selected. Below the radio buttons is a grid of buttons labeled with letters (A-F), digits (0-9), and symbols (+/-). In the bottom-left corner of the grid, there is a button labeled **2's C**. Above the grid, there are four primary conversion buttons: **Bin2Dec**, **Dec2Hex**, **Hex2Bin**, and **CL**.

Figure A.3: ผลลัพธ์จากการแปลงเลข -123 ให้เป็นเลขฐานสิบหกชนิด Signed 2-Complement ความยาว 24 บิตหรือ 6 ตัวเลข

4. กดปุ่ม **Hex2Bin** เพื่อแปลงผลลัพธ์กลับไปเป็นฐานสอง แล้วเลือกตัวเลขฐานสองทั้งหมด แล้วทำการคัดลอก (Copy) หรือกดปุ่ม **Ctrl-C** พร้อมกัน
5. คลิกบนชื่อลิงค์ต่อไปนี้ เพื่อเปิดหน้าเว็บสำหรับ บวก/ลบ/คูณ/หาร เลขจำนวนเต็ม ทั้งชนิด Unsigned และ Signed http://www.free-test-online.com/binary/binary_calculator.html

Appendix A. การทดลองที่ 1 ข้อมูลและคณิตศาสตร์ในคอมพิวเตอร์

6. กดปุ่ม Signed ก่อนแล้วจึงทำการวาง (Paste) ลงในกล่องข้อความ เพื่อเปลี่ยนการทำงานให้อยู่ในโหมดตัวเลขฐานสองชนิดมีเครื่องหมายตามรูปที่ A.4

Figure A.4: หน้าต่างการแปลงเลข -123 ให้เป็นเลขฐานสองชนิด Signed 2-Complement ความยาว 32 บิต

7. กดปุ่ม '-' เพื่อทำการกระบวนการลบเลข แล้ววาง (Paste) เลข -123 อีกรอบในกล่องข้อความที่ว่างลง

8. กดปุ่ม '=' เพื่อแสดงผลลัพธ์

Figure A.5: ผลลัพธ์จากการแปลงเลข -123 ให้เป็นเลขฐานสองชนิด Signed 2-Complement ความยาว 32 บิต

ในรูปที่ A.5 แสดงให้เห็นว่า $-123 - (-123) = 0$

A.1.2 กิจกรรมท้ายการทดลอง

จงทำการทดลองและตอบคำถามต่อไปนี้ โดยแสดงวิธีทำการทดลองตามเนื้อหาในหัวข้อที่ [1.2.2](#) และตรวจคำตอบตามวิธีทำการทดลองที่ได้ทำไป

1. จงแปลงเลขฐานสิบต่อไปนี้ให้เป็นเลขฐานสองและฐานสิบหก ชนิดไม่มีเครื่องหมาย และนับจำนวนบิตที่เกิดขึ้น

ฐานสิบ	ฐานสอง	ฐานสิบหก
7	0111	7
8	1000	8
15	1111	F
16	10000	10
255	1111 1111	FF
256	1000000000	100
65535	1111 1111 1111 1111	FFFF
65536	1000000000000000	10000

2. จงแปลงเลขฐานสิบต่อไปนี้ให้เป็นเลขฐานสองและฐานสิบหกชนิด มีเครื่องหมายแบบ 2-Complement และนับจำนวนบิตที่เกิดขึ้น

ฐานสิบ	ฐานสอง	ฐานสิบหก
+1	1	1
-1	11	F
+15	1111	F
-16	10000	F0
+255	1111 1111	FF
-256	10000 0000	F00
+65535	1111 1111 1111 1111	FFFF
-65536	10000 0000 0000 0000	F0000

3. จงบวกเลข 2-Complement ต่อไปนี้ แล้วบันทึกผลลัพธ์เป็นฐานสอง ฐานสิบ ข้อผิดพลาดที่แจ้งเตือน และอธิบายเหตุผลว่าทำไม่สำเร็จไม่ตรงกัน

- $$10000000000000000000000000000000 + 0000000000000000000000000000000$$
 - ผลลัพธ์ฐานสอง 1000 0000 0000 0000 0000 0000 0000 0001
 - ผลลัพธ์ฐานสิบ $-2,147,483,648 + 1 = -2,147,483,647$
 - ข้อผิดพลาดที่แจ้งเตือน _____
 - เหตุผล _____
 - $$10000000000000000000000000000000 + 10000000000000000000000000000000$$
 - ผลลัพธ์ฐานสอง 0
 - ผลลัพธ์ฐานสิบ $-4.294,967,996$

Appendix A. การทดลองที่ 1 ข้อมูลและคณิตศาสตร์ในคอมพิวเตอร์

A.2 การแปลงและคณิตศาสตร์สำหรับ Floating-Point IEEE754

เพื่อให้เข้าใจรูปแบบของเลขฐานสอง และคณิตศาสตร์ แบ่งเป็น Single Precision และ Double Precision สอดคล้องกับเนื้อหาในหัวข้อ 1.6

A.2.1 การทดลองสำหรับ Single-Precision

การทดลองนี้จะเน้นที่การแปลงเลขจำนวนจริงให้เป็น เลขฐานสองทศนิยมชนิดลอยตัว สอดคล้องกับเนื้อหาในหัวข้อที่ 1.6 ในรูปแบบ Single Precision โดยผ่านเว็บเบราว์เซอร์ที่ผู้อ่านสนใจ กรอกหรือคลิกที่ช่องค์ต่อไปนี้

http://www.binaryconvert.com/convert_float.html

เมื่อเว็บเพจปรากฏขึ้น ขอให้ผู้อ่านปฏิบัติตามการทดลอง ดังนี้

- กรอกเลข 123 ลงในกล่องข้อความ แล้วกดปุ่ม Convert to binary ได้รูปที่ A.6

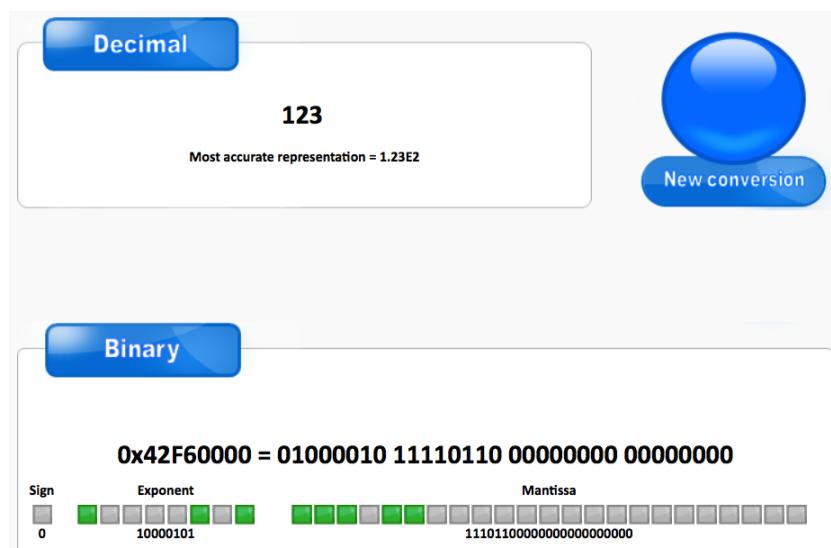


Figure A.6: ผลลัพธ์จากการแปลงเลข 123 ให้เป็นเลขฐานสองชนิด Single Precision

โปรดสังเกต กล่องสีเหลืองสีเขียวตรงกับบิตที่เป็น '1' กล่องสีเทาตรงกับบิตที่เป็น '0' หมายถึง เลขฐานสิบหาก

- กรอกเลข -123 ลงในกล่องข้อความ แล้วกดปุ่ม Convert to binary ได้รูปที่ A.7

Appendix A. การทดลองที่ 1 ข้อมูลและคณิตศาสตร์ในคอมพิวเตอร์

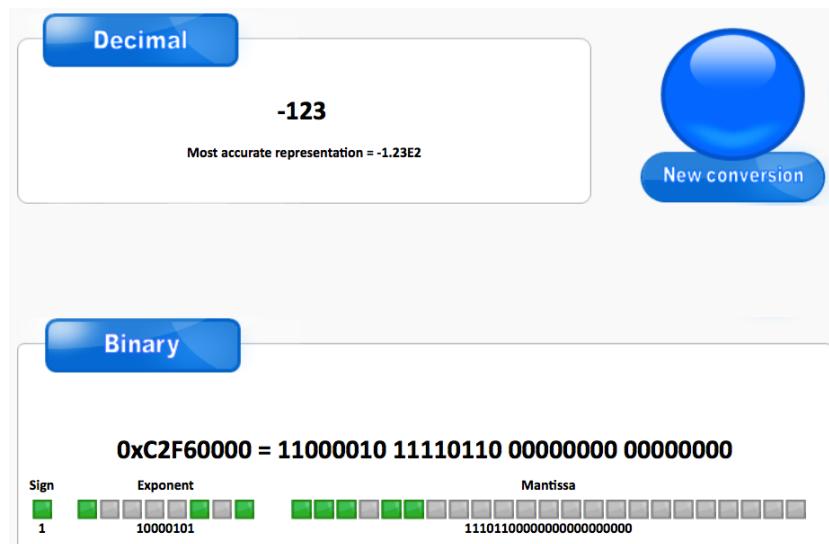


Figure A.7: ผลลัพธ์จากการแปลงเลข -123 ให้เป็นเลขฐานสองชนิด Single Precision

โปรดสังเกต ความแตกต่างที่ค่า Sign Exponent และ Mantissa ดังนั้น เราจะเห็นได้ว่าเฉพาะ Sign ที่มีการเปลี่ยนแปลง

- คลิกบนลิงค์นี้ เพื่อทดลองบวกและคูณเลขในรูปแบบ Single Precision ด้วยลิงค์ต่อไปนี้ <http://weitz.de/ieee/> เลื่อนหน้าเว็บลงไปด้านล่างสุด เพื่อค้นหาแถบเมนูตามรูปที่ A.8

[binary16](#) [binary32](#) [binary64](#) [binary128](#)

Figure A.8: เมนูด้านล่างสุดของหน้าเว็บ เพื่อเลือกเลขฐานสองชนิด Single Precision (Binary32) และ Double Precision (Binary64)

- เลื่อนหน้าเว็บกลับไปด้านบนสุดเพื่อกรอกเลข -123 ลงในกล่องข้อความซ้ายบน และ กรอกเลข 123 ลงในกล่องข้อความถัดลงมา และกดปุ่ม + และจะได้ผลลัพธ์ดังรูปต่อไปนี้

Appendix A. การทดลองที่ 1 ข้อมูลและคณิตศาสตร์ในคอมพิวเตอร์

IEEE 754 Calculator

(See info at bottom of page.)



Figure A.9: ผลลัพธ์จากการบวกเลข $-123+123$ ให้เป็นเลขฐานสองชนิด Single Precision

จะสังเกตเห็นว่า ผลลัพธ์ที่ได้เรียกว่า True Zero ตามตารางที่ 1.11

5. กดปุ่ม x (คูณ) แล้วจะได้ผลลัพธ์ดังรูปต่อไปนี้

IEEE 754 Calculator

(See info at bottom of page.)



Figure A.10: ผลลัพธ์จากการคูณเลข -123×123 ให้เป็นเลขฐานสองชนิด Single Precision

A.2.2 การทดลองสำหรับ Double-Precision

การทดลองนี้จะเน้นที่การแปลงเลขจำนวนจริงให้เป็น เลขฐานสองทศนิยมชนิดลอยตัว สอดคล้องกับเนื้อหาในหัวข้อที่ 1.6 ในรูปแบบ Double Precision โดยผ่านเว็บเบราว์เซอร์ที่ผู้อ่านสนใจ กรอกหรือคลิกที่ ชื่อลิงค์ต่อไปนี้

http://www.binaryconvert.com/convert_double.html

เมื่อเว็บเพจปรากฏขึ้น ขอให้ผู้อ่านปฏิบัติตามการทดลอง ดังนี้

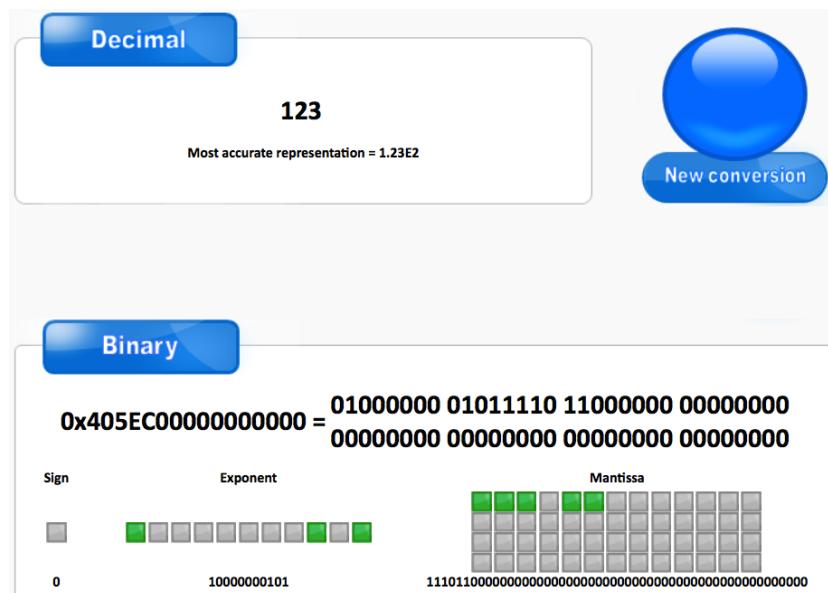


Figure A.11: ผลลัพธ์จากการแปลงเลข 123 ให้เป็นเลขฐานสองชนิด Double Precision

- กรอกเลข 123 ลงในกล่องข้อความ และกดปุ่ม Convert to binary ได้รูปที่ A.11

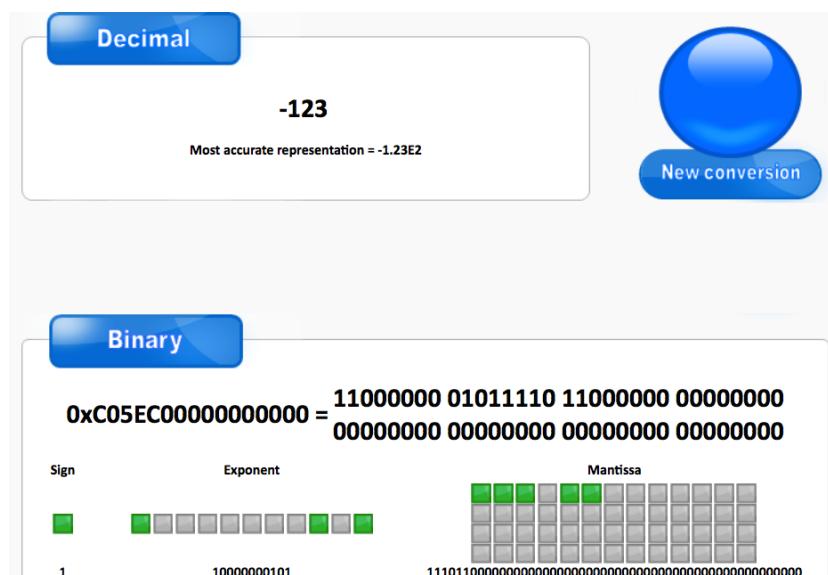


Figure A.12: ผลลัพธ์จากการแปลงเลข -123 ให้เป็นเลขฐานสองชนิด Double Precision

- คลิกบนลิงค์นี้ เพื่อทดลองบางและคุณเลขในรูปแบบ Double Precision ด้วยลิงค์ต่อไปนี้ <http://weitz.de/ieee/> และกดเลือกเมนู Binary64 ในรูปที่ A.8

Appendix A. การทดลองที่ 1 ข้อมูลและคณิตศาสตร์ในคอมพิวเตอร์

3. กรอกเลข -123 ลงในกล่องข้อความซ้ายบน และ กรอกเลข 123 ลงในกล่องข้อความด้านล่างมา แล้ว กดปุ่ม + แล้วจะได้ผลลัพธ์ดังรูปด้านล่างนี้

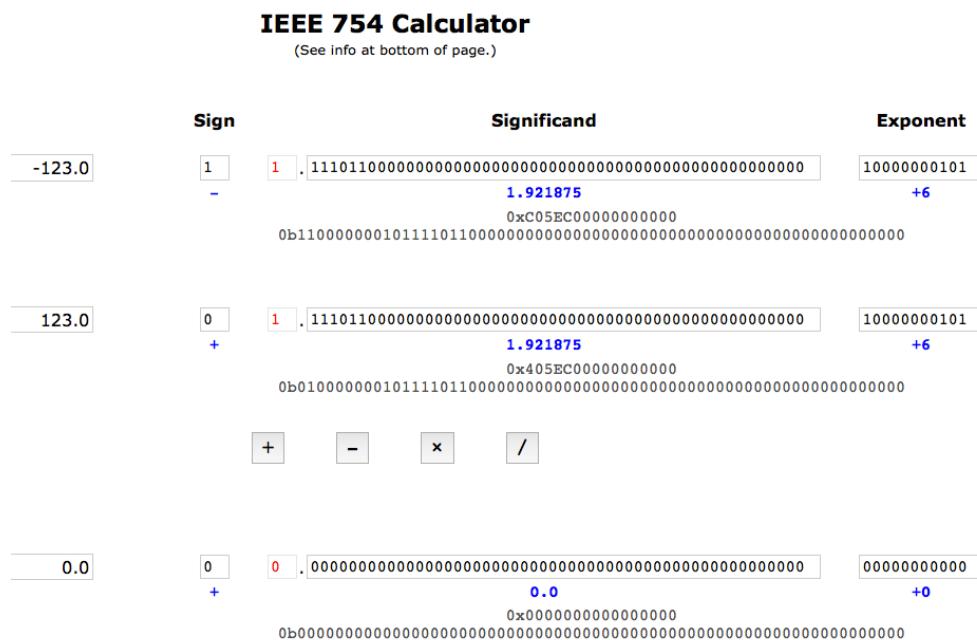


Figure A.13: ผลลัพธ์จากการบวกเลข -123+123 ให้เป็นเลขฐานสองชนิด Double Precision

จะสังเกตเห็นว่า ผลลัพธ์ที่ได้เรียกว่า True Zero ตามตารางที่ 1.11

4. กดปุ่ม × (คูณ) แล้วจะได้ผลลัพธ์ดังรูปด้านล่างนี้

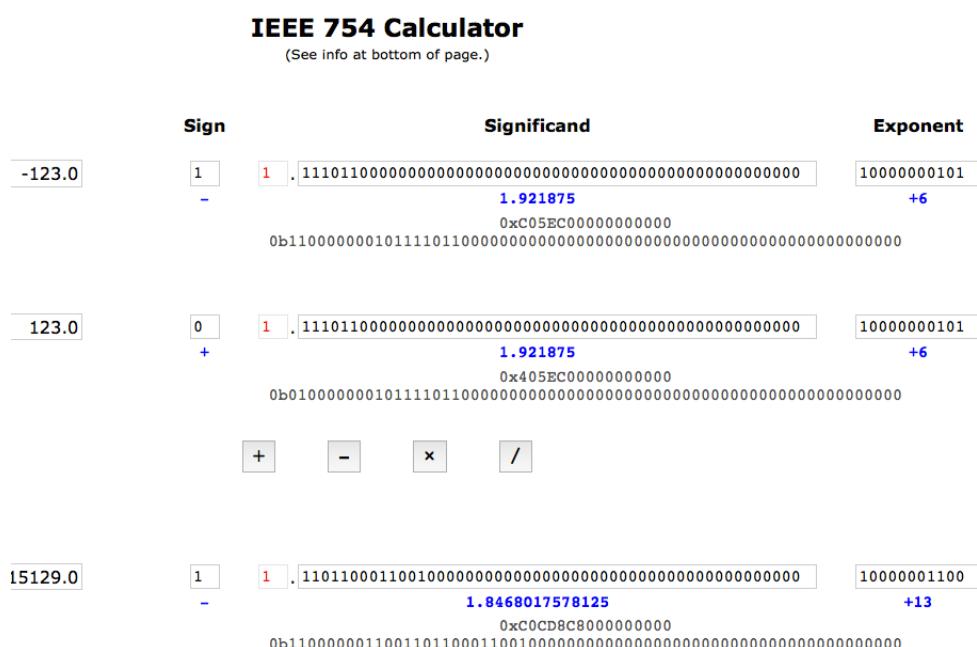


Figure A.14: ผลลัพธ์จากการคูณเลข -123 × 123 ให้เป็นเลขฐานสองชนิด Double Precision

A.2.3 กิจกรรมท้ายการทดลอง

จงใช้เว็บเพจลิงค์ต่อไปนี้ในการตอบคำถาม

<https://www.h-schmidt.net/FloatConverter/IEEE754.html>

Figure A.15: เว็บสำหรับการตอบคำถามเพื่อสร้างเลขหรือแปลงเลขฐานสิบด้วยมาตรฐาน IEEE 754 Single Precision การกดเลือกคือทำให้บันทึกไว้กับ '1'

โดยแสดงวิธีทำงานเนื้อหาในหัวข้อที่ 1.6 และตรวจคำตอบตามวิธีทำการทดลองที่ได้ทำไป

1. จงสร้างเลข -0.0_{10} โดยการกดเลือกปุ่มสีเหลืองในส่วน Sign เท่านั้น
Binary Representation=.....
Hexadecimal Representation=.....

2. จงสร้างเลข -1.0_{10} โดยการกดเลือกปุ่มสีเหลืองในส่วน Exponent เท่านั้น ต่อจากข้อที่แล้ว
Binary Representation=.....
Hexadecimal Representation=.....

3. จงสร้างเลข -1.5_{10} โดยการกดเลือกปุ่มสีเหลืองในส่วน Mantissa เท่านั้น ต่อจากข้อที่แล้ว
Binary Representation=.....
Hexadecimal Representation=.....

4. จงสร้างเลข 5.877472×10^{-39} ซึ่งอยู่ในรูป ดินอมัลไลซ์ (denormalized) เพราะมีค่าน้อยเกินไป
Binary Representation=.....

Hexadecimal Representation=.....**0x0040 0000**.....

5. จงแปลงเลข 32 บิตนี้ให้เป็น เลขจำนวนเต็ม โดยใช้ลิงค์ต่อไปนี้ http://www.binaryconvert.com/convert_signed_int.html เมื่อคัดลอกและวางเลขครบแล้ว ให้กดปุ่ม Convert to decimal

A.3 รหัสของข้อมูลตัวอักษร

A.3.1 การทดลอง

การทดลองในหัวข้อนี้จะเป็นการแปลงรหัสตัวอักษรภาษาอังกฤษและไทย เป็นรหัส ASCII และ Unicode ตามเนื้อหาในหัวข้อ 1.7 ผ่านทางเว็บไซต์ <https://www.branah.com/ascii-converter> ที่มีนักพัฒนาเพื่อเผยแพร่ความรู้เป็นวิทยาทานเช่นเดียวกับเว็บที่ได้ทดลองมา

1. เปิดเว็บตามลิงค์ต่อไปนี้ หรือ กดปุ่มซ้ายบนชื่อลิงค์ <https://www.branah.com/ascii-converter>

2. กรอกข้อความต่อไปนี้ ลงไปในกล่องข้อความ ASCII

ไทย กข ค a b c

โปรดสังเกต ระหว่างตัวอักษรมี ซ่องว่าง 1 ตัวอักษรเสมอ

3. กดปุ่ม Convert ซ้ายบนสุด จะได้ผลลัพธ์ดังรูปต่อไปนี้

ASCII Converter - Hex, decimal, binary, base64, and ASCII converter

The screenshot shows a web-based ASCII converter tool with four main tabs:

- ASCII (Example: a b c)**: Input: ไทย กข ค abc. Buttons: Add spaces, Remove spaces, Convert white space characters.
- Hex (Example: 0x61 0x62 0x63)**: Input: e44 e17 e22 e01 e02 e04 61 62 63. Buttons: Convert, Remove 0x.
- Decimal (Example: 97 98 99)**: Input: 3652 3607 3618 3585 3586 3588 097 098 099.
- Binary (Example: 01100001 01100010 01100011)**: Input: 111001000100 111000010111 111000100010 111000000001 111000000010 111000000100 01100001 01100010 01100011.

Figure A.16: ผลลัพธ์จากการกรอกและแปลงตัวอักษร ไทย กข ค เป็นรหัสต่างๆ

- กล่องข้อความ Hex จะแสดงค่า Unicode สำหรับภาษาไทย และ ASCII สำหรับภาษาอังกฤษ ในรูปผู้เขียนได้กดเลือก Remove 0x เพื่อความสะดวกในการอ่านค่า

A.3.2 กิจกรรมท้ายการทดลอง

- จงอธิบายวิธีการหาค่าฐานสิบ 0 - 9 จากรหัส ASCII ของตัวอักษร 0 - 9
นำข้อมูล 0 1 2 3 ลงในช่อง ASCII แล้วกด Convert เธิงเห็นตัว ascii เป็นฐาน 10 (ดูในช่อง decimal)
- จงอธิบายวิธีการหาค่าฐานสิบ 0 - 9 จากรหัส Unicode ของตัวอักษร 0 - 9
แนบชื่อหนึ่ง แต่ตัวที่ต้องการจะแปลงเป็น (264, 265, ..., 383) → จะเห็นตัวเลขตัวแรกเป็น Unicode
- จงเปิดเว็บที่มีข้อความภาษาไทย เช่น เว็บข่าว แล้วทดลองเปลี่ยนการนำเสนอบนจอเพื่อ View source เช่น Google Chrome ใช้เมนู Tool-> View Source และ Find หรือกดปุ่ม CTRL-F คำว่า charset ว่ามีค่าเท่ากับ utf-8 หรือไม่ เพราะเหตุใด
จะต้องตั้งค่า utf-8 เมื่อ utf-8 เป็นรหัสทางภาษาไทย จึงจะสามารถอ่านตัวอักษรไทยได้

Appendix **B**

การทดลองที่ 2 การประกอบและติดตั้งบอร์ด Pi3

การทดลองนี้เสริมสร้างประสบการณ์ให้ผู้อ่านได้มีโอกาสจับต้องบอร์ดและอุปกรณ์เสริม และสนับสนุนเนื้อหาของบทที่ 3 ในส่วนของhaarดแวร์ โดยมีวัตถุประสงค์ ดังต่อไปนี้

- เพื่อให้รู้จักโครงสร้างและรายละเอียดต่างๆ ของบอร์ด Pi3
- เพื่อให้เข้าใจลักษณะการระบายความร้อนของอุปกรณ์อิเลคทรอนิกส์
- เพื่อให้ทดสอบการเชื่อมต่อกับอุปกรณ์อินพุท/เอาท์พุทที่จำเป็น

การทดลองนี้ต้องการอุปกรณ์อื่นๆ ได้แก่ คอมอนิเตอร์ที่รองรับสัญญาณ HDMI หากไม่มีผู้อ่านสามารถใช้สายแปลงที่จำเป็น เช่น แปลงจากสัญญาณ HDMI เป็นสัญญาณ VGA หรือแปลงเป็นสัญญาณ DVI คีย์บอร์ด เม้าส์ กล้องสำหรับรับสัญญาณ และอุปกรณ์ระบายความร้อนหรือฮีทซิงค์

บอร์ด Raspberry Pi เป็นคอมพิวเตอร์ขนาดเล็กเท่ากับบัตรเครดิต เริ่มต้นออกแบบและผลิตในประเทศสหราชอาณาจักร โดยมูลนิธิ Raspberry Pi Foundation ซึ่งเริ่มต้นจากการเป็นคอมพิวเตอร์ราคาถูกสำหรับการศึกษา ต่อมา บอร์ดมีการประยุกต์ใช้งานเพิ่มขึ้นเรื่อยๆ ตั้งแต่ปี 2012 ทำให้มีการพัฒนามาก里斯ต์ จนเป็นเจนเนอเรชันที่ 3 ในปัจจุบัน โดยมีสองรุ่น คือ โมเดล A และ โมเดล B.

โมเดล A เป็นอุปกรณ์ที่เรียบง่าย ตันทุนมากกว่าโมเดล B ในขณะที่ โมเดล B เหมาะกับการใช้งานเป็นคอมพิวเตอร์ตั้งโต๊ะ และเซิร์ฟเวอร์ เนื่องจากมีการเชื่อมต่อแบบ Ethernet

B.1 รายการอุปกรณ์ฮาร์ดแวร์



Figure B.1: รูปแสดงรายการอุปกรณ์สำหรับประกอบบอร์ด

ผู้อ่านสามารถสั่งซื้อชุดหูลคิก (Toolkit) สำหรับบอร์ด Pi3 ในรูปที่ B.1 ประกอบด้วย รายการดังต่อไปนี้

ลำดับ	ชื่อและรายละเอียด	จำนวน	มี/ไม่มี
1	บอร์ด Pi 3 โมเดล B แรม 1 กิกะไบท์	1	
2	อะแดปเตอร์แปลงไฟกระแทกตรง 5.0 โวลท์ 2.5 แอม培ร์ (ปลายสายเป็นหัวไมโคร USB ชนิด B)	1	
3	สายเชื่อมต่อชนิด HDMI	1	
4	แผ่นวงจรโปรโตบอร์ด	1	
5	หัวขยายการเชื่อมต่อ GPIO สำหรับบอร์ดโมเดล B	1	
6	สายแพ GPIO 40 ขาสำหรับบอร์ดโมเดล B	1	
7	อิฐชิ้นขนาดเล็ก	2	
8	หน่วยความจำ MicroSD ขนาด 16GB	1	
9	กล่องสำหรับบอร์ด Pi3	1 ชุด	
10	หัวเชื่อมต่อขนาด 40 ขา	1	
11	สายแพขนาด 40 ขาชนิดตัวเมี้ยx2	1	
12	สายต่อวงจรชนิดตัวเมี้ยx2	1 ชุด	
13	สายแปลง HDMI เป็น VGA	1	

B.2 ประกอบบอร์ด Pi3 และกล่อง

การประกอบบอร์ด Pi3 เพื่อให้พร้อมใช้งานเพื่อการศึกษาและทดลอง จะต้องเสริมความแข็งแรง เพิ่มความสามารถในการระบายความร้อน ต่อขยายขาเข้า/ออกบนบอร์ดให้ยาวขึ้น

B.2.0.1 ประกอบฮีทซิงค์ (Heat Sink)

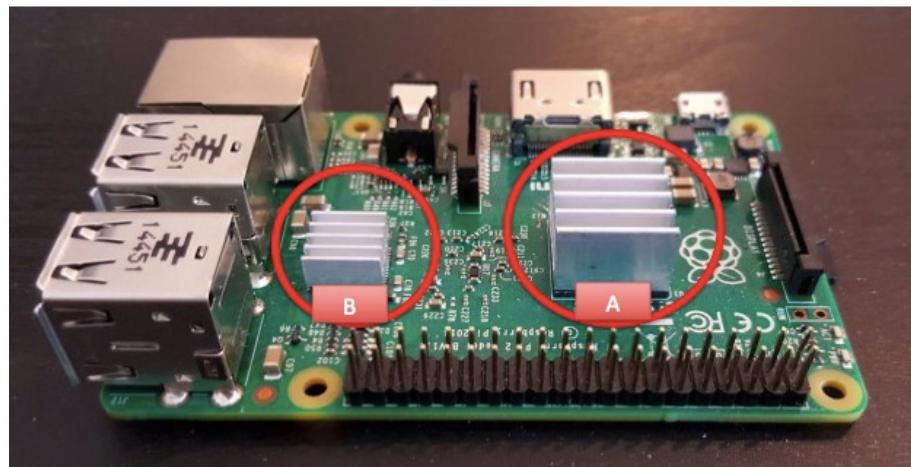


Figure B.2: บอร์ด Pi 3 เมื่อติดฮีทซิงค์เรียบร้อยแล้ว

- ตรวจสอบความเรียบร้อยของบอร์ด Pi3 ที่ได้รับมา หากมีร่องรอยชำรุด หรือ ไฟ ขอให้แจ้งกับผู้ดูแล
- หยับฮีทซิงค์ที่มีขนาดใหญ่ที่สุด แกะแผ่นเคลือบการสองหน้าออก วางด้านที่มีการลงบนชิป BCM2837 ตรงกลาง ณ ตำแหน่ง A ในรูปที่ [B.2](#)
- หยับฮีทซิงค์ที่มีขนาดกลาง แกะแผ่นเคลือบการสองหน้าออก วางด้านที่มีการลงบนชิป LAN9514 ณ ตำแหน่ง B ในรูปที่ [B.2](#)

ฮีทซิงค์ A ติดบนชิป BCM2837 ขนาดใหญ่กว่า เนื่องจากชิพทำงานที่ความถี่สัญญาณคลื่นสูงกว่า ซึ่งซ่อนมากกว่า ในขณะที่ฮีทซิงค์ B ติดบนชิป LAN9514 ซึ่งภายในคือ รูทับของ USB 2.0 และ Ethernet Controller 10/100

B.2.0.2 ประกอบกล่องใส่บอร์ด Pi 3

กล่องที่จำหน่วยมาพร้อมบอร์ดของบริษัทแห่งหนึ่ง ประกอบด้วยแผ่นพลาสติก 5 ชิ้น ตามรูปที่ [B.3](#)

Appendix B. การทดลองที่ 2 การประกอบและติดตั้งบอร์ด Pi3

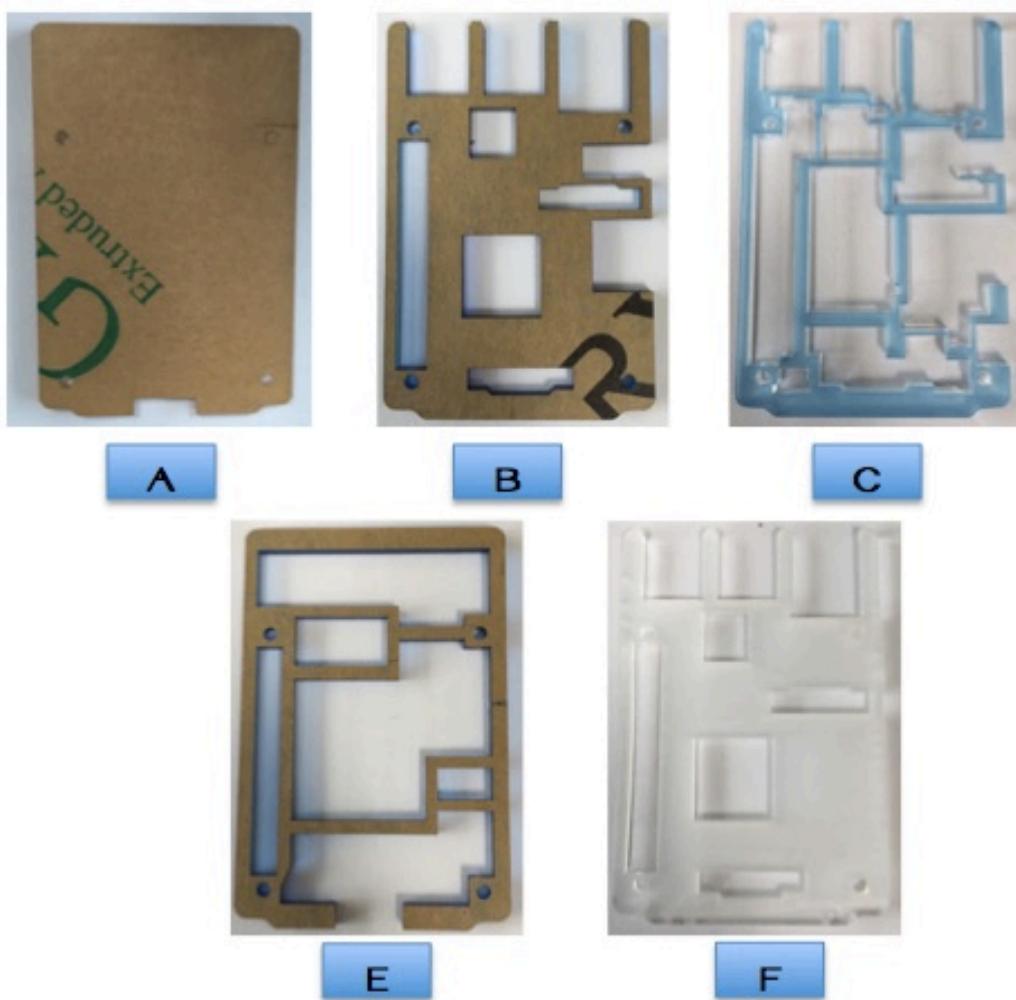


Figure B.3: แผ่นพลาสติก 5 ชิ้น สำหรับประกอบเป็นกล่องของบอร์ด Pi3

1. ประกอบแผ่นพลาสติกและบอร์ด Pi3 ด้วยกันตามรูปที่ [B.4](#) แผ่น B ประกอบกับบอร์ด Pi3 ด้านล่าง แล้วขอนบนแผ่น A ซึ่งอยู่ชั้นล่างสุด ประกอบแผ่น C D E เข้าด้วยกัน โดยแผ่น E อยู่ชั้นบนสุด และวางช้อนบนบอร์ด
2. ใช้สกรูและไขควงแผ่นพลาสติกและบอร์ด Pi3 เข้าด้วยกันทั้ง 4 มุม
3. ติดตั้งขาเชื่อมขยายชนิด 2x20 หรือ 2 แคลๆ ละ 20 ขาบนบอร์ด โดยสังเกตจากในรูปที่ [B.5](#) ว่าขาหมายเลข 1 อยู่ตรงมุมซ้ายบน

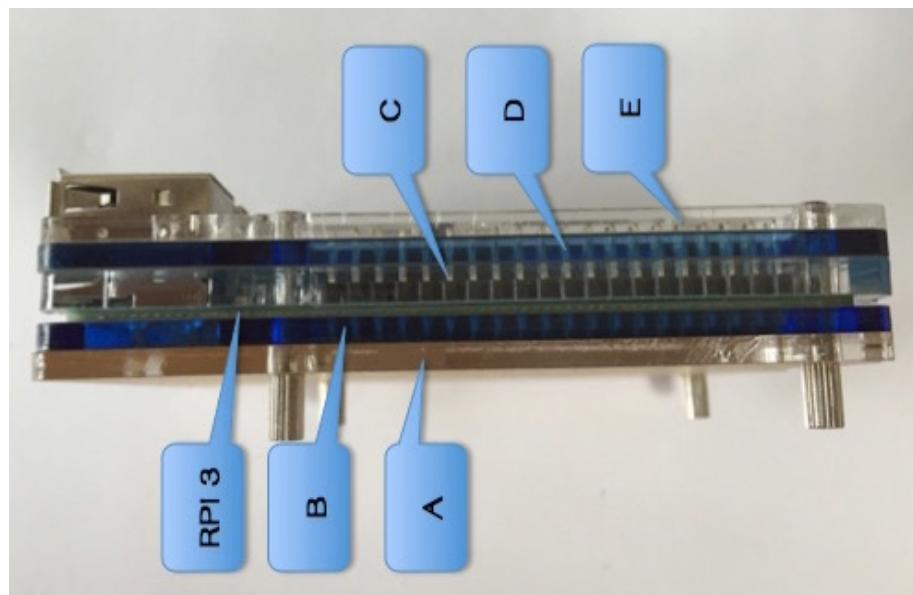


Figure B.4: บอร์ด Pi3 ที่มีกล่องประกอบเรียบร้อยแล้ว

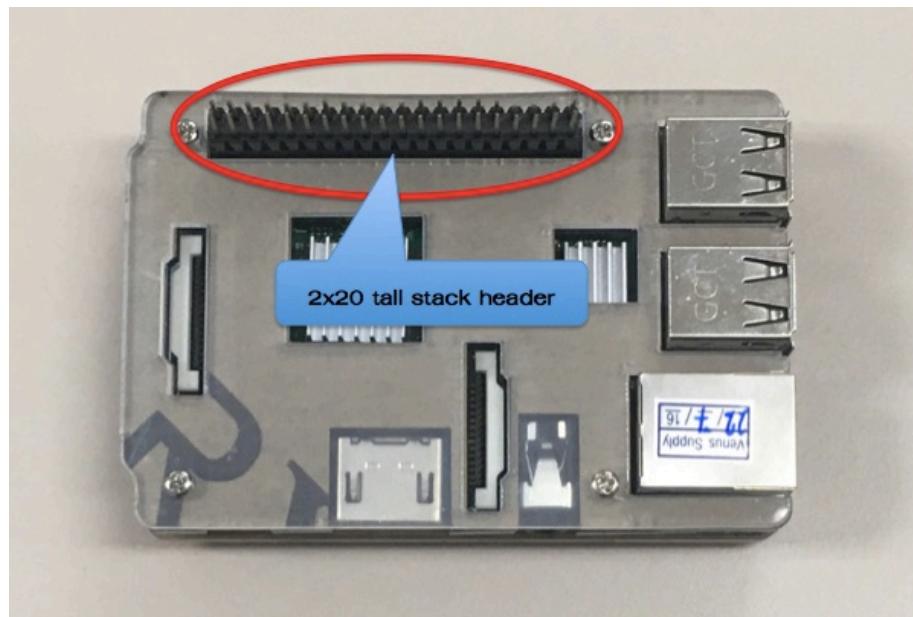


Figure B.5: บอร์ด Pi3 เมื่อประกอบขาเชื่อมขยาย 2x20

B.3 เครื่องคอมพิวเตอร์ส่วนบุคคลจากบอร์ด Pi 3 โมเดล B

1. ถอดสายไฟ 5 โวลท์ จากเต้าเสียบไฟ 220 โวลท์
2. เชื่อมต่อสายแปลง HDMI เป็น VGA กับบอร์ด Pi3 แล้วจึงเชื่อมสาย VGA กับจอคอมพิวเตอร์ เลือกอินพุตของจอเป็น Analog Input
3. เชื่อมต่อคีย์บอร์ดและมาส์กับช่องเสียบสาย USB บนบอร์ด Pi3 โปรดสังเกตสัญลักษณ์ของ USB บนหัวสายจะต้องหมายขึ้นดังรูปที่ B.6

Appendix B. การทดลองที่ 2 การประกอบและติดตั้งบอร์ด Pi3

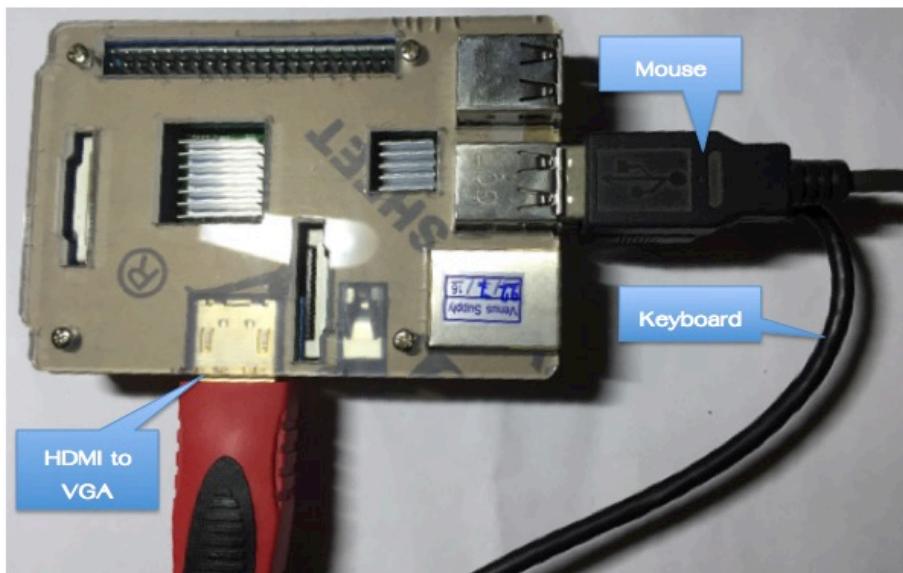


Figure B.6: การเชื่อมต่อคีย์บอร์ดและเม้าส์กับช่องเสียบสาย USB บนบอร์ด Pi3

4. เชื่อมต่อหัวไมโคร USB บนบอร์ด Pi3 ก่อน แล้วจึงเสียบอแดปเตอร์เข้ากับเต้ารับไฟ 220 โวลท์ โปรดสังเกตลักษณ์ของ USB บนหัวสายจะต้องหมายขึ้น ดังรูปที่ B.7

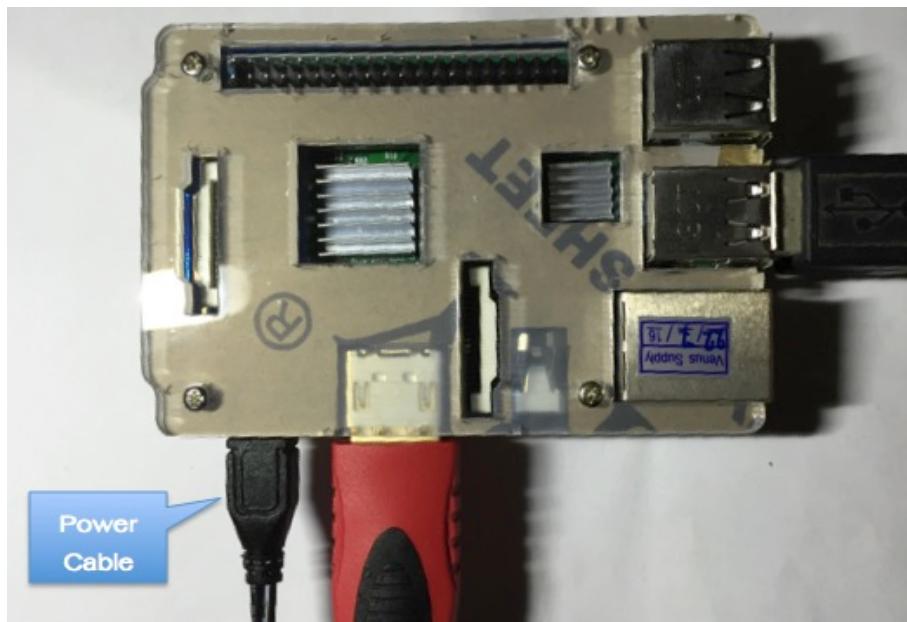


Figure B.7: การเชื่อมต่อไฟเลี้ยงจากอแดปเตอร์ทางหัวไมโคร USB กับบอร์ด Pi3

5. หลอดไฟ LED สีแดงจะสว่างขึ้นเมื่อไม่มีอะไรมีดพลาด หลังจากนั้น ภาพสีรุ่งจะปรากฏขึ้นบนจอ ด้านซ้ายบน ดังรูปที่ B.8

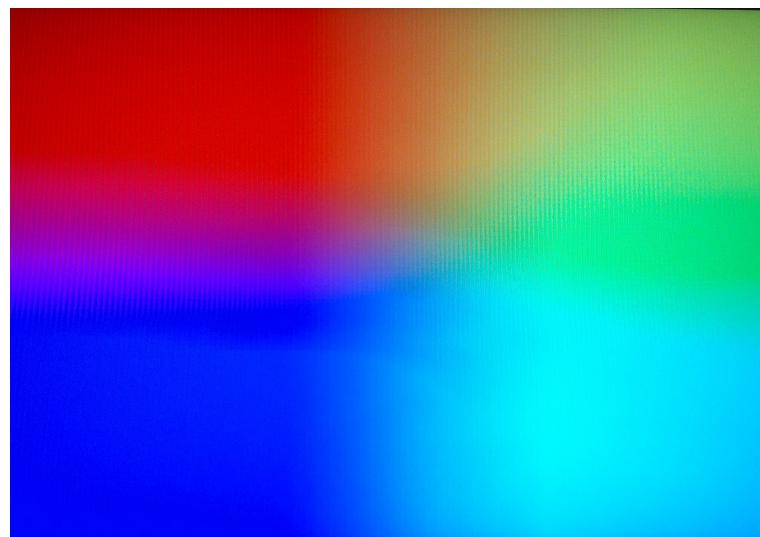


Figure B.8: ภาพสีรุ่งบนจออันเกิดจากบอร์ด Pi3

B.4 กิจกรรมท้ายการทดลอง

1. ทดสอบเดปเตอร์ออกจากเต้าเสียบ ประกอบสายแพกับหัวต่อเชื่อมกับໂປຣໂຕບອົດ ແລ້ວເຂົ້ມຕ່ອງສາຍມີເຕອຮືສີດຳກັບກາງດົກຂອງບອົດ
2. ເສີບອແດປເຕອຮົນເຕົາເສີບ ວັດຄວາມຕ່າງໆສັກຍົງຂອງໄຟ 5 Volt ແລະ 3.3 Volt ໂດຍໃຫ້ຫົວຄອນເນັ້ນຕົກເຕອຮົນຈາກຫົວຕ່ອງ 40 ຂາໄປບົນໂປຣໂຕບອົດ

การทดลองที่ 3 การติดตั้งระบบปฏิบัติการ Raspbian

การทดลองนี้เสริมสร้างประสบการณ์ให้ผู้อ่านได้มีโอกาสติดตั้งระบบปฏิบัติการของบอร์ด และโปรแกรมเสริม ผู้อ่านที่เคยติดตั้งและใช้งานระบบปฏิบัติการอื่นๆ เช่น ไมโครซอฟต์วินโดวส์ และ Mac OS โดยมองการ์ด microSD เป็นอุปกรณ์สำรองข้อมูลเช่นเดียวกับอุปกรณ์อื่นๆ การทดลองจะช่วยเสริมสร้างความเข้าใจเนื้อหาของบทที่ 3 ในส่วนของซอฟต์แวร์ โดยมีวัตถุประสงค์ ดังต่อไปนี้

- เพื่อให้รู้จักโครงสร้างและการฟอร์แมทหน่วยความจำ microSD เพื่อติดตั้งระบบปฏิบัติการ Raspbian
- เพื่อให้เข้าใจกลไกการติดตั้งระบบปฏิบัติ Raspbian การผ่านทางเครือข่ายอินเทอร์เน็ท
- เพื่อให้การใช้งานระบบปฏิบัติการ Raspbian หรือลีนุกซ์เบื้องต้น

C.1 การเตรียมการ์ดหน่วยความจำ MicroSD

ก่อนผู้อ่านจะติดตั้งระบบปฏิบัติการ Raspbian บนบอร์ด Pi3 ผู้อ่านจะต้องเตรียมการ์ดหน่วยความจำ microSD ให้เรียบร้อย โดยจะต้องฟอร์แมท (Format) การ์ด แล้วจึงติดตั้งโปรแกรม NOOBS ตามลงไว้ใน Windows:

1. ดาวน์โหลดโปรแกรมชื่อ SDFormatter สำหรับฟอร์แมท ตามลิงค์ต่อไปนี้

https://www.sdcard.org/downloads/formatter_4/eula_windows/

คลายการบีบอัด และรันไฟล์ Setup.exe เพื่อติดตั้งโดยทำตาม installShield Wizard จะติดตั้งแล้วเสร็จ

2. เปิดโปรแกรม SDFormatter
3. คลิกเมนู Option แล้วตั้งค่า FORMAT SIZE ADJUSTMENT ให้เป็น ON

Appendix C. การทดลองที่ 3 การติดตั้งระบบปฏิบัติการ Raspbian

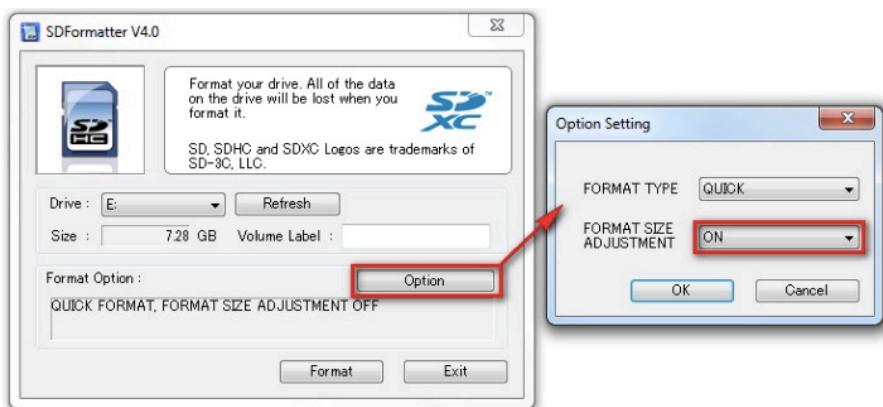


Figure C.1: การฟอร์แมทการ์ด microSD บนระบบ Windows

4. เลือกการ์ด microSD ที่ต้องการฟอร์แมทจากเมนู Drive ที่เลื่อนลงมา
5. เลือกไดร์ฟที่ต้องการและตรวจสอบความถูกต้อง
6. คลิกเมนู Format และคลิก OK เพื่อรอจนหน้าต่าง Drive Format complete! แสดงขึ้นมา

บนระบบปฏิบัติการ Mac OS X:

1. ดาวน์โหลดโปรแกรมชื่อ SDFormatter สำหรับฟอร์แมท ตามลิงค์ต่อไปนี้

https://www.sdcard.org/downloads/formatter_4/eula_mac/
2. เปิดโปรแกรม SDFormatter
3. เลือก Overwrite Format เพื่อฟอร์แมทและลบข้อมูลเดิมไปพร้อมๆ กัน ดังรูป

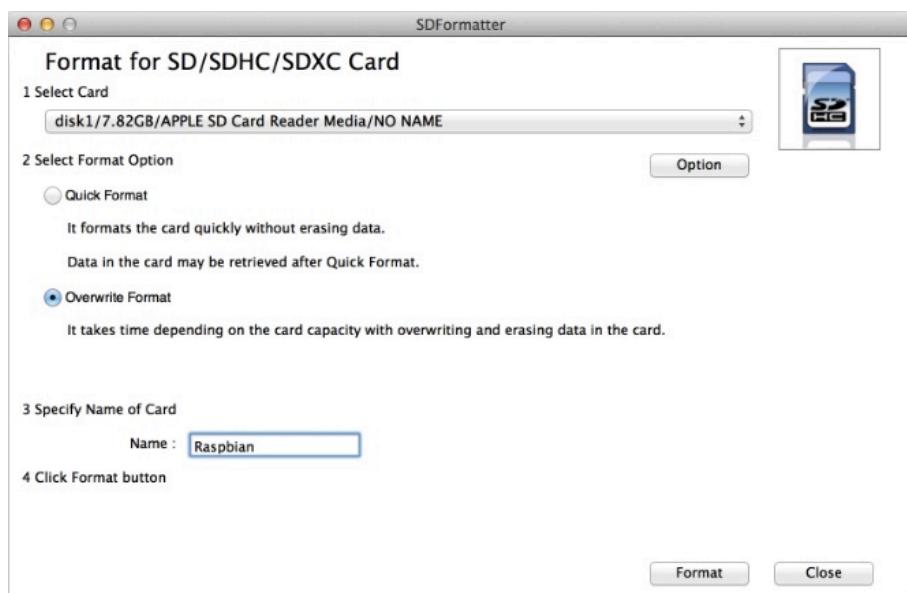


Figure C.2: การฟอร์แมทการ์ด microSD บนระบบ Mac OS

4. เลือกการ์ด microSD ที่ต้องการฟอร์แมทจากเมนู Drive ที่เลื่อนลงมา

5. เลือกไดร์ฟที่ต้องการและตรวจสอบความถูกต้อง
6. คลิกเมนู Format แล้วคลิก OK เพื่อรอจนหน้าต่าง Drive Format complete! แสดงขึ้นมา

C.2 การติดตั้ง NOOBS (โนอบส์) บนการ์ด microSD

ผู้อ่านควรจะติดตั้ง NOOBS (อ่านว่า โนอบส์) บนการ์ด microSD ผ่านเครื่องคอมพิวเตอร์ สำหรับผู้ใช้งานบนระบบปฏิบัติการไมโครซอฟต์วินโดวส์ หมายเหตุ ผู้อ่านควรเตรียมการ์ดแเปลงนขนาดไมโคร ให้เป็นการ์ด SD ขนาดปกติ

Windows:

1. ดาวน์โหลด New Out of Box Software (NOOBS) ตามลิงค์ต่อไปนี้:

<https://www.raspberrypi.org/downloads/noobs/>

2. คลายการบีบอัดไฟล์ที่ดาวน์โหลดในโฟลเดอร์สูงสุดของการ์ด
3. เมื่อแล้วเสร็จ การ์ดควรมีรายชื่อโฟลเดอร์และไฟล์ต่างๆ ในรูปที่ C.3

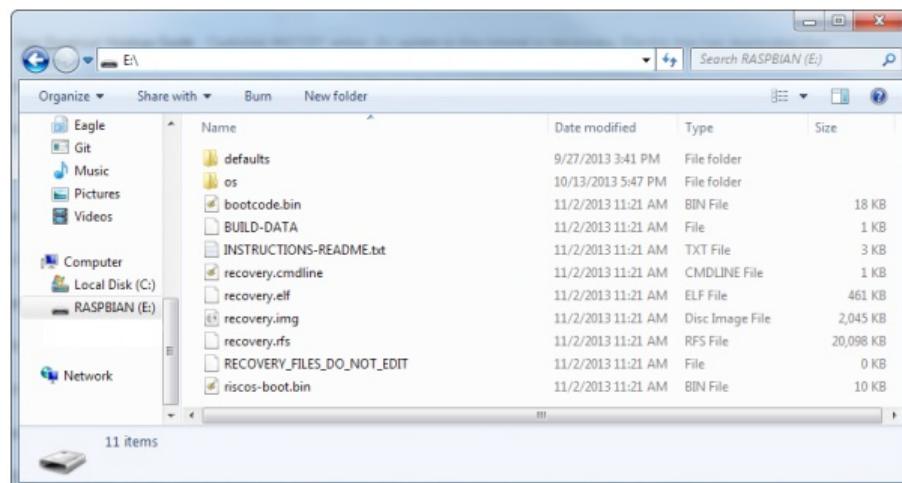


Figure C.3: โฟลเดอร์ในระบบปฏิบัติการวินโดวส์ หลังจาก Unzip ไฟล์ NOOBS.zip

โปรดสังเกตชื่อโฟลเดอร์หรือไดเรกทอรีที่มีรายชื่อไฟล์ของ NOOBS หากว่างไฟล์เหล่านี้ผิดโฟลเดอร์ จะทำให้บอร์ดทำงานไม่ได้

4. สั่งให้วินโดวส์ถอดการ์ด microSD ออกจากเครื่องด้วยการ Eject

Mac OS X:

1. กดเพื่อดาวน์โหลด Raspberry Pi's New Out of Box Software (NOOBS) จากลิงค์ต่อไปนี้:
<https://www.raspberrypi.org/downloads/noobs/>
2. คลายการบีบอัดไฟล์ที่ดาวน์โหลดในโฟลเดอร์สูงสุดของการ์ด

Appendix C. การทดลองที่ 3 การติดตั้งระบบปฏิบัติการ Raspbian

3. เมื่อแล้วเสร็จ การติดความมีรายชื่อไฟล์เดอร์และไฟล์ต่างๆ ในรูปที่ C.4 โปรดสังเกตชื่อไฟล์เดอร์หรือไดเรกทอรีที่มีรายชื่อไฟล์ของ NOOBS หากว่างไฟล์เหล่านี้มิได้ไฟล์เดอร์ จะทำให้บอร์ดทำงานไม่ได้

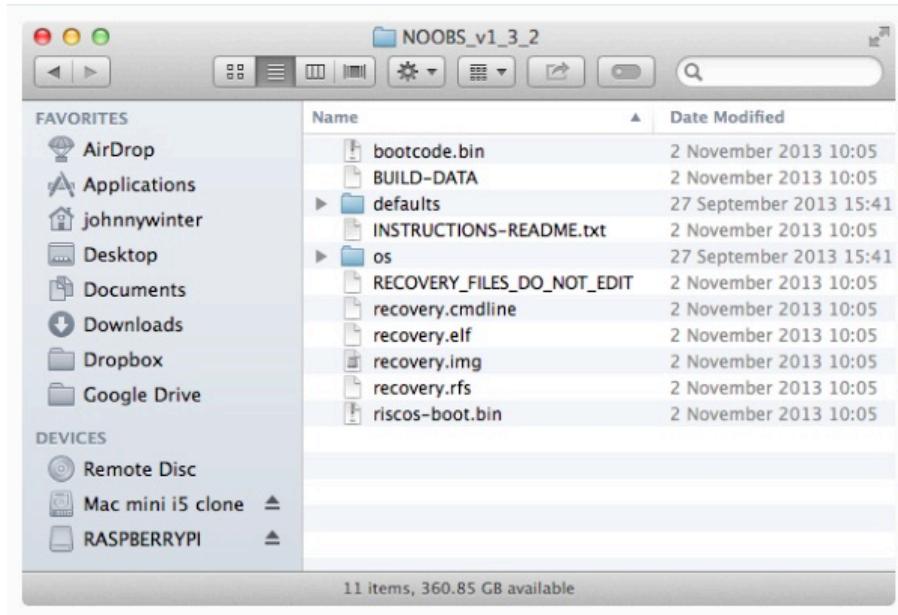


Figure C.4: ไฟล์เดอร์ในระบบปฏิบัติการ Mac OS X หลังจาก Unzip ไฟล์ NOOBS.zip

หมายเหตุ: ให้ผู้อ่านก็อปปี้ไฟล์จากไฟล์เดอร์นี้ลงในไฟล์เดอร์หลักของหน่วยความจำ SD โดยตรง

4. เมื่อติดตั้งแล้วเสร็จ จึงกดเลือก Eject หน่วยความจำ SD ออกจากช่องเสียบ

C.3 การติดตั้งระบบปฏิบัติการ Raspbian

1. สอดหน่วยความจำ microSD เข้าไปในสล็อตบนบอร์ด Pi3



Figure C.5: การสอดหน่วยความจำ microSD เข้าไปในสล็อตบนบอร์ด Pi3 โดยหมายบอร์ดขึ้นมา โปรดสังเกตการ์ดหน่วยความจำจะต้องหงายขึ้นดังรูป

2. ตรวจสอบว่าการ์ดหน่วยความจำเสียบถูกต้องแล้วจึงเสียบตัวจ่ายไฟเลี้ยงให้กับบอร์ด
3. ตรวจสอบว่าบอร์ดทำงานเมื่อจ่ายไฟให้ โดยเริ่มต้นโปรแกรม NOOBS จะเริ่มต้นแบ่งพาร์ติชันใหม่แล้วฟอร์แมทการ์ดหน่วยความจำ
4. เลือกติดตั้ง Raspbian โดยคลิกเลือกวิธีการติดตั้งที่ต้องการ แล้วจึงคลิกปุ่ม *Install*
5. เมื่อติดตั้งแล้วเสร็จ กดปุ่ม *OK* เพื่อให้บอร์ด Pi3 รีบูต หรือ เริ่มต้นใหม่ เพื่อทำการติดตั้งค่าต่างๆ โดยโปรแกรมชื่อ *raspi-config* ต่อไป
6. สำหรับผู้อ่านขั้นเริ่มต้น ผู้อ่านไม่ควรปรับแก้ใดๆ ระบบจะตั้งชื่อ username อัตโนมัติคือ *pi* โดยมีรหัสผ่าน (Password) คือ *raspberry* ซึ่งแนะนำว่า ผู้ติดตั้งควรเปลี่ยนเพื่อความปลอดภัย และจดบันทึกไว้เสมอ

Appendix C. การทดลองที่ 3 การติดตั้งระบบปฏิบัติการ Raspbian

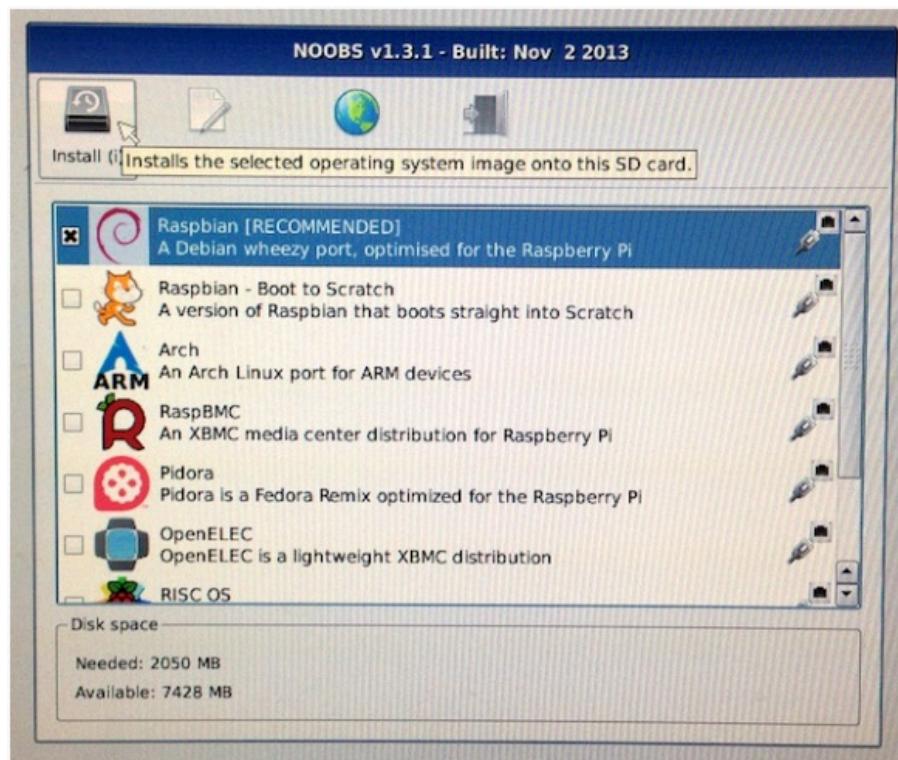


Figure C.6: การติดตั้งระบบ Raspbian

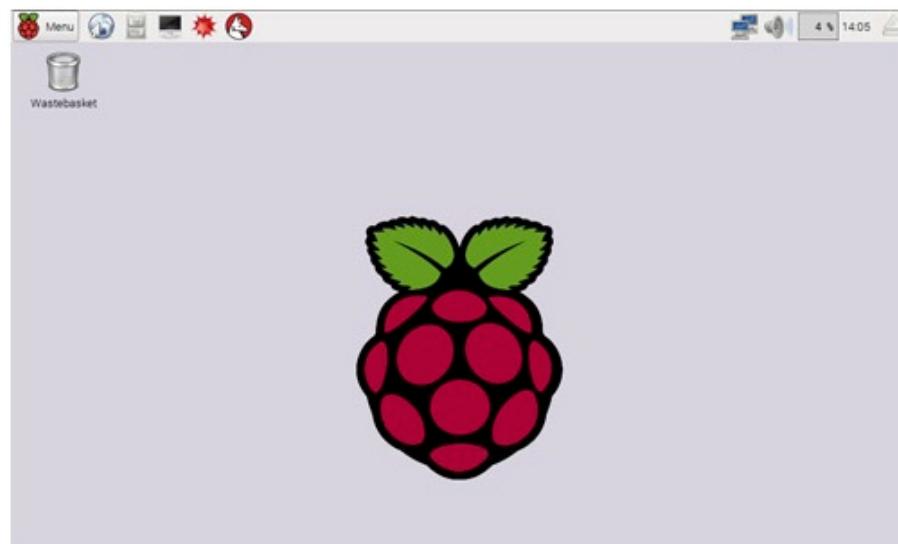


Figure C.7: Graphical User Interface ของระบบปฏิบัติการ Raspbian

C.4 การตั้งค่าบอร์ด Pi3 เพื่อใช้งาน

C.4.1 การตั้งชื่อและพาสเวิร์ด

ทำการตั้งชื่อผู้ใช้ และพาสเวิร์ด ซึ่งผู้อ่านควรใช้ชื่อ pi และพาสเวิร์ดที่ป้องกันแต่จำได้ขึ้นใจ เพื่อความปลอดภัยในอนาคต

ตั้งค่าความละเอียดของจอแสดงผล โดยเลือกใช้ค่า 1920x1080 ก่อน เพื่อทดสอบบุญสมบัติของจอที่ใช้ โดยผู้ใช้สามารถเปลี่ยนแปลงได้ภายหลังในการทดลองที่ 7 ภาคผนวก |

C.4.2 การตั้งค่า Wi-Fi เพื่อเชื่อมต่อกับอินเทอร์เน็ท

- หน้าจอหลักมุมขวาบน มองหาสัญลักษณ์ WiFi คลิกซ้ายบนรูปไอคอนนี้ เพื่ออ่านรายชื่อของสัญญาณ WiFi (SSID) ที่อยู่รอบๆ บริเวณบอร์ด ตามรูปที่ C.8



Figure C.8: แสดงรายชื่อสัญญาณ WiFi รอบๆ ที่บอร์ด Pi3 มองเห็น

- คลิกซ้ายเลือกรายชื่อสัญญาณที่ต้องการ ตามรูปที่ C.9

Appendix C. การทดลองที่ 3 การติดตั้งระบบปฏิบัติการ Raspbian



Figure C.9: รายชื่อสัญญาณ Wi-Fi ที่ต้องการเชื่อมต่อ

- หากสัญญาณ WiFi ที่ต้องการเชื่อมต่อนั้นมีการเข้ารหัสเพื่อความปลอดภัย ผู้ใช้จะต้องกรอกรหัสที่ทราบในหน้าต่างที่ปรากฏขึ้นมานี้ ตามรูปที่ C.10



Figure C.10: หน้าต่างสำหรับกรอกรหัส Wi-Fi ที่ต้องการเชื่อมต่ออย่างปลอดภัย

- เมื่อกรอกรหัสตามที่ทราบแล้ว ผู้ใช้ต้องกดปุ่ม OK เพื่อดำเนินการต่อ หากเชื่อมต่อสำเร็จ ชื่อสัญญาณจะปรากฏตรงมุมขวาบน ตามรูปที่ C.11
- ในหน้าต่าง Terminal พิมพ์คำสั่ง `sudo apt-get update` เพื่ออัพเดตระบบให้เป็นปัจจุบัน กรนีผู้อ่านดาวน์โหลดเวอร์ชันที่ไม่ล่าสุด
- ในหน้าต่าง Terminal พิมพ์คำสั่ง `iwconfig` เพื่อแสดงรายละเอียดต่างๆ ของสัญญาณ WiFi ที่เชื่อมต่ออยู่

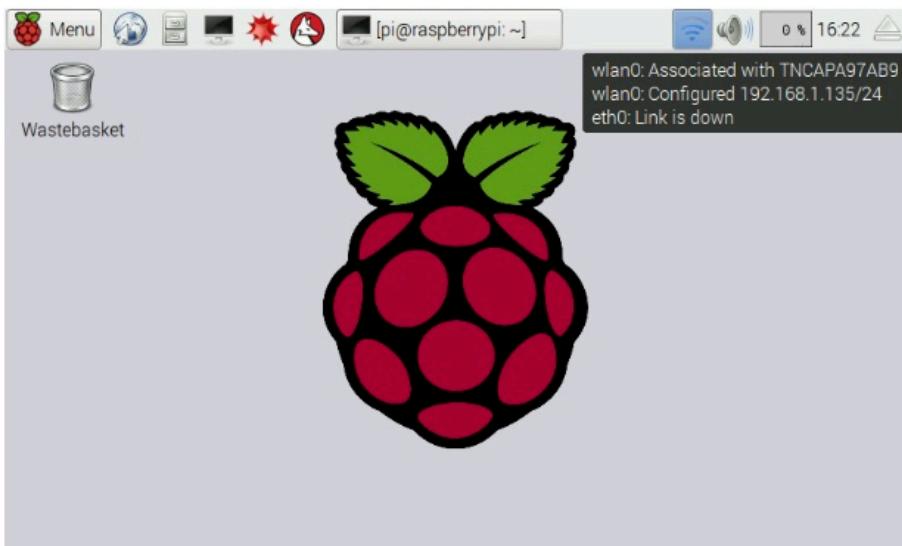


Figure C.11: ชื่อสัญญาณ Wi-Fi ที่ใช่อมต่อสำเร็จพร้อมกับหมายเลขและドレス IP ที่ได้รับมอบหมาย

C.4.3 การรีสตาร์ทและซัพเดต

การรีบูท หรือ รีสตาร์ทเครื่อง มักใช้เรียกเมื่อระบบต้องการหลังการอัพเดทซอฟต์แวร์ต่างๆ ที่จำเป็น หรือผู้ใช้ต้องการแก้อาการต่างๆ โดย

- ในหน้าต่าง Terminal พิมพ์คำสั่ง `sudo reboot` เพื่อรีบูทบอร์ด Pi3 และระบบปฏิบัติการ หรือ
- ในหน้าต่าง Terminal พิมพ์คำสั่ง `sudo shutdown -h now` เพื่อเตรียมพร้อมก่อนปิดเครื่อง ตามที่กล่าวในหัวข้อที่ [3.2.7](#)

C.4.4 กิจกรรมท้ายการทดลอง

1. การติดตั้งระบบจากไฟล์ config.txt เพื่อแจ้งให้ ARM Loader ทำการบูทรูบบตามรายละเอียดในไฟล์นั้น ผู้อ่านสามารถอ่านค่าโดยใช้คำสั่ง

```
cat /boot/config.txt
```

ขอให้ผู้อ่านสังเกตและบันทึกประโยชน์ที่ไม่เขียนด้วยสัญลักษณ์ # เพื่อค้นคว้าเพิ่มเติมใน Google

2. สำรวจส่วนต่างๆ ของหน้า Desktop และวัดตามคร่าวๆ พร้อมรายละเอียดสำคัญ
3. สำรวจเมนูหลัก และเมนูรองว่ามีรายละเอียดอะไรบ้าง และวัดเป็นแผนภูมิต้นไม้
4. ค้นหาวิธีการเพิ่มคีย์บอร์ดภาษาไทยเพื่อใช้งานบนระบบ Raspbian ทางเว็บไซต์

Appendix D

การทดลองที่ 4 การใช้งานระบบปฏิบัติการยูนิกซ์เบื้องต้น

ยูนิกซ์ เป็นระบบปฏิบัติลำดับต้นๆ ของโลก ที่เป็นต้นแบบการสร้างระบบปฏิบัติการต่างๆ รวมทั้ง ลีนุกซ์ และระบบปฏิบัติการ Raspbian ผู้อ่านสามารถเรียนรู้การใช้งานคำสั่งพื้นฐานด้วยการพิมพ์คำสั่งทางคีย์บอร์ด และกราฟิกไปพร้อมกัน โดยมีวัตถุประสงค์ดังต่อไปนี้

- เพื่อค้นคว้าข้อมูลขั้นสูงของบอร์ด Pi3
- เพื่อเปรียบเทียบการทำงานแบบคำสั่งทางคีย์บอร์ดและแบบกราฟิคโดยใช้เมาส์หรือทัชแพด (Touch Pad)
- เพื่อให้ผู้อ่านใช้คำสั่งเพื่อบริหารจัดการไฟล์ในไดเรกทอรีหรือโฟลเดอร์
- เพื่อวางแผนพื้นฐานการใช้งานระบบปฏิบัติการยูนิกซ์เบื้องต้นสำหรับพัฒนาโปรแกรมภาษาต่างๆ

ผู้อ่านที่คุ้นเคยกับระบบปฏิบัติการวินโดว์ส และการพิมพ์คำสั่งทางคีย์บอร์ด (Command Line) ของระบบปฏิบัติการดอส (DOS: Disk Operating System) ในอดีต จะคุ้นพบว่า คำสั่งเหล่านี้มีความใกล้เคียงกัน แต่ยูนิกซ์จะเข้มงวดกับตัวพิมพ์เล็กเป็นหลัก ขอให้ผู้อ่านปฏิบัติตามคำสั่งอย่างระมัดระวัง และสังเกตตัวพิมพ์อย่างละเอียดว่าเป็นตัวพิมพ์ใหญ่หรือเล็ก

D.1 การใช้งาน Unix ผ่านทาง GUI

D.1.1 หน้าจอหลัก (Desktop)

โครงสร้างปกติ ตารางเปรียบเทียบระหว่างไอคอนและปุ่มต่างๆ ของ Raspbian และวินโดว์ส ไอคอนเมนูคล้ายกับของปรับแก้ได้

Appendix D. การทดลองที่ 4 การใช้งานระบบปฏิบัติการยูนิกซ์เบื้องต้น

ปุ่ม	Raspbian	Windows
ปุ่มปิด (Close)		
ปุ่มย่อ (Minimize)		
ปุ่มขยาย (Maximize)		

D.1.2 ไฟล์เมเนจเจอร์ (File Manager)

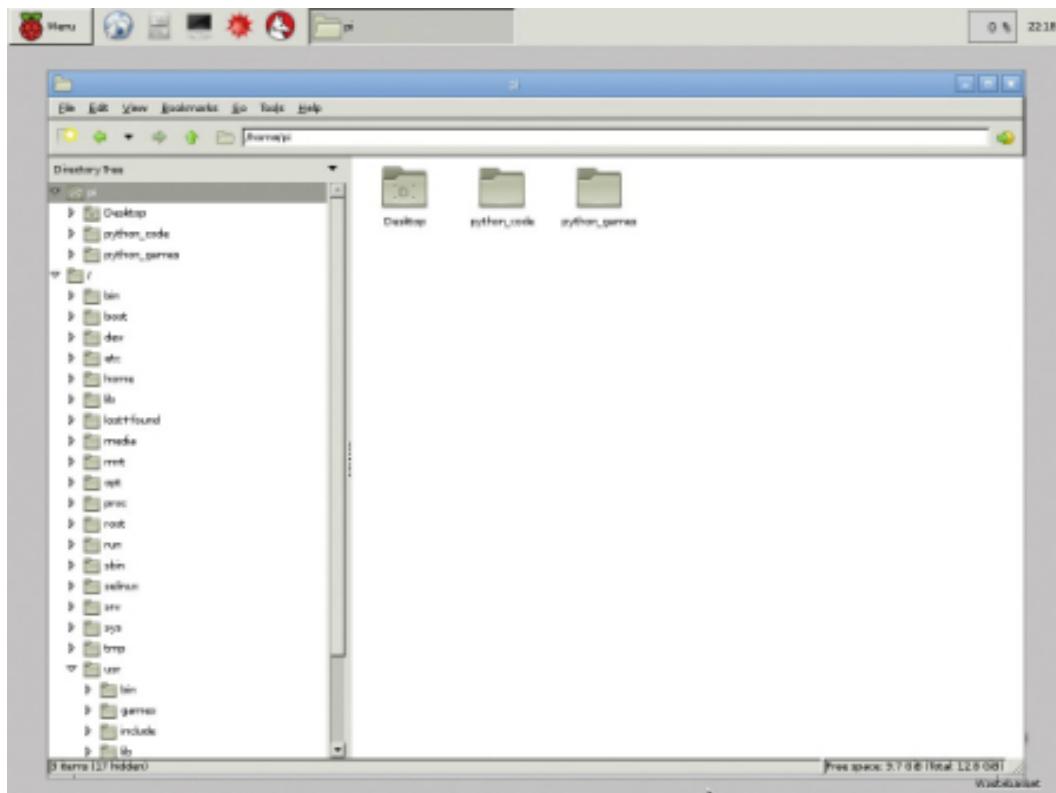


Figure D.1: หน้าต่างของไฟล์เมเนจเจอร์ (File Manager)

directory structure

Home directory

D.1.3 การชัตดาวน์ (Shutdown)

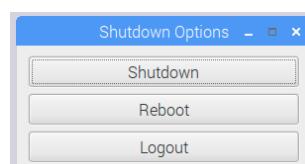


Figure D.2: Shutdown

Shortcut	Raspbian	Windows
Copy		
Cut		
Paste		

D.2 การใช้งาน Unix ผ่านทางคีย์บอร์ด



Figure D.3: Icon Terminal

คอมมานด์ไลน์ (Command Line)

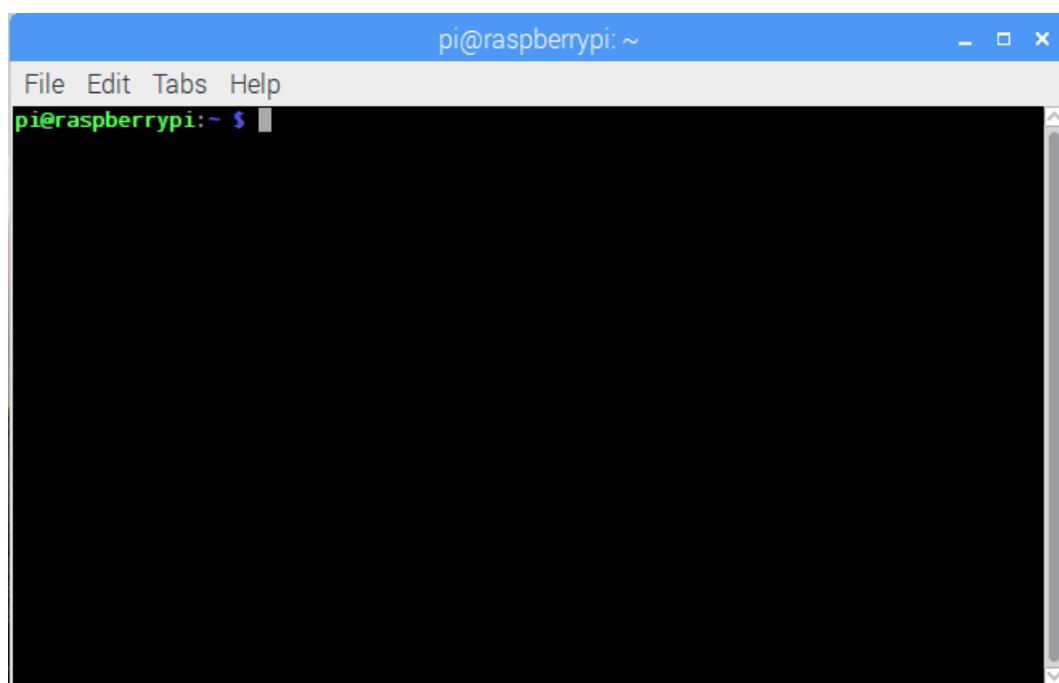


Figure D.4: Terminal

D.2.1 คำสั่งพื้นฐานของระบบ Unix

ผู้อ่านสามารถฝึกใช้คำสั่งเหล่านี้บนโปรแกรมเทอร์มินัล (Terminal) ตามตารางต่อไปนี้

Appendix D. การทดลองที่ 4 การใช้งานระบบปฏิบัติการยูนิกซ์เบื้องต้น

ลำดับที่	รายละเอียด	คำสั่ง
1	แสดงรายชื่อไฟล์และไดเรกทอรี	<code>ls <parameter></code>
	Ex.: \$ ls แสดงรายชื่อไฟล์และไดเรกทอรีในไดเรกทอรีปัจจุบัน	
	Ex.: \$ ls -l แสดงรายละเอียดต่างๆ ของไฟล์และไดเรกทอรีในไดเรกทอรีปัจจุบัน	
	Ex.: \$ ls -la แสดงรายละเอียดต่างๆ ของไฟล์และไดเรกทอรีทั้งหมดในไดเรกทอรีปัจจุบัน โปรดสังเกตสัญลักษณ์ต่อไปนี้บริเวณสอง端ของผลลัพธ์	
	”.” หมายถึง ไดเรกทอรีปัจจุบัน (current directory)	
	”..” หมายถึง ไดเรกทอรีที่อยู่เหนือขึ้นไป (parent directory)	
2	สร้างไฟล์เปล่า	<code>touch <file_name></code>
	Ex.: \$ touch test.txt สร้างไฟล์เปล่าชื่อ "text.txt"	
3	ทำไฟล์สำเนา	<code>cp <source_file> <destination_file></code>
	Ex.: \$ cp test.txt test2.txt	
4	เปลี่ยนชื่อไฟล์	<code>mv <source_file> <destination_file></code>
	Ex.: \$ mv test.txt test3.txt	
5	แสดงชื่อดireกทอรีปัจจุบัน	<code>pwd</code>
	Ex.: \$ pwd	
6	สร้างไดเรกทอรีใหม่	<code>mkdir <directory_name></code>
	Ex.: \$ mkdir /home/Pi/Lab สร้างไดเรกทอรีใหม่ชื่อ "Lab" ภายใต้ไดเรกทอรี "/home/Pi/"	
7	Change directory	<code>cd <destination></code>
	Ex.: \$ cd /home/Pi/Lab โปรดสังเกตสัญลักษณ์ต่อไปนี้ในประโยชน์ /home/Pi/Lab ”/” ตำแหน่งชี้ทางสุด หมายถึง ไดเรกทอรีราก (root directory) ”/” ตำแหน่งถัดมา หมายถึง สัญลักษณ์คั่นระหว่างชื่อดireกทอรี	

D.2.2 การขัดดาวน์ (Shutdown)

ผู้อ่านสามารถรีบูตหรือรีสตาร์ทบอร์ดใหม่ด้วยคำสั่ง

```
$ sudo shutdown -r now
```

โดย -r หมายถึง restart และ now หมายถึง ณ บัดนี้ ในทำนองเดียวกัน ผู้อ่านสามารถปิดการทำงานของบอร์ดด้วยคำสั่ง

```
$ sudo shutdown -h now
```

โดย -h หมายถึง halt แปลว่า หยุด ซึ่งนักคอมพิวเตอร์ส่วนใหญ่นิยมใช้ศัพท์คำนี้ในสั้นให้เครื่องคอมพิวเตอร์สิ้นสุดการทำงาน

D.3 ข้อมูลพื้นฐานของบอร์ด Pi3

การใช้งานทางคอมมานด์ไลน์มีประโยชน์หลายด้าน เนื่องจากรองรับคำสั่งเกือบทั้งหมดในระบบ ผู้อ่านควรจะฝึกให้หัดล่อง เพื่อเตรียมความพร้อมไปเป็นนักพัฒนาโปรแกรมและพัฒนาระบบท่อไป โดยการทดลองนี้จะอาศัยคำสั่งเพื่ออ่านค่าข้อมูลของชีพียูและข้อมูลขั้นสูงอื่นๆ

D.3.1 ข้อมูลของชีพียู

ผู้อ่านสามารถศึกษารายละเอียดเกี่ยวกับชีพียูที่ใช้งานอยู่บนบอร์ด โดยใช้คำสั่ง

```
$ cat /proc/cpuinfo
```

จดผลลัพธ์ที่ได้จากบอร์ด Pi3 ลงในช่องที่กำหนดให้

- Processor : ARMv -compatible processor rev ()
- BogoMIPS : .
- Features : _____
- CPU implementer : _____
- CPU architecture : _____
- CPU variant : 0x
- CPU part : 0x
- CPU revision :
- Hardware : BCM
- Revision :
- Serial : _____

D.3.2 ข้อมูลขั้นสูงของชีพิญและบอร์ด

ผู้อ่านสามารถสอบถามข้อมูลด้านฮาร์ดแวร์เชิงลึกจากคำสั่งต่อไปนี้

ลำดับที่	คำสั่ง	รายละเอียด
1	cat /proc/cpuinfo	รายละเอียดของชีพิญในการทดลองก่อนหน้า
2	cat /proc/version	รายละเอียดของระบบปฏิบัติการ
3	cat /proc/meminfo	รายละเอียดของหน่วยความจำ
4	cat /proc/partitions	รายละเอียดของการ์ด microSD
5	vcgencmd measure_temp	อ่านค่าอุณหภูมิ ณ จุดต่างๆ
6	vcgencmd measure_volts core	อ่านค่าโวลต์เจของชีพิญคอร์
7	vcgencmd measure_volts sdram_c	อ่านค่าโวลต์เจของ SD-RAM
8	vcgencmd measure_volts sdram_i	อ่านค่าโวลต์เจของ SD-RAM I/O

ยกตัวอย่าง เช่น ข้อมูลด้านหน่วยความจำถูกบันทึกในไฟล์ /proc/meminfo ผู้อ่านสามารถแสดงข้อมูลในไฟล์โดย

```
$ cat /proc/meminfo
```

จดผลลัพธ์ที่ได้จากบอร์ด Pi3

```
MemTotal: _____ kB
MemFree: _____ kB
Buffers: _____ kB
Cached: _____ kB
```

D.4 กิจกรรมท้ายการทดลอง

1. ใช้โปรแกรมไฟล์เมเนจ ero เพื่อทำการสำรวจโครงสร้างของไฟล์เดอร์ต่างๆ ในเครื่อง
2. จะเปรียบเทียบโครงสร้างของไฟล์เดอร์ต่างๆ กับรูปที่ 3.10 ว่าแตกต่างกันอย่างไร
3. จะใช้โปรแกรมไฟล์เมเนจ ero เพื่อทำการสำเนาหรือก็อปปี้ไฟล์ ลบไฟล์ สร้างไฟล์เดอร์ใหม่
4. จะใช้โปรแกรม Terminal และคำสั่งที่จำเป็น เพื่อทำการสำรวจโครงสร้างของไฟล์เดอร์ต่างๆ ในเครื่อง และเปรียบเทียบกับข้อที่แล้ว
5. ชิป BCM2837 มีจำนวนชีพิญกี่คอร์
6. ชิป BCM2708 เกี่ยวข้องกับ ชิปในตระกูล BCM283x อย่างไร
7. จะบอกหมายเลขรุ่นหรือรหัสของชีพิญ

Appendix D. การทดลองที่ 4 การใช้งานระบบปฏิบัติการยูนิกซ์เบื้องต้น

8. ในหัวข้อที่ [D.3.2](#) จงบอกค่าขนาดของหน่วยความจำ MemFree Buffers Cached เพื่อเปรียบเทียบกับ MemTotal ว่าแตกต่างกันหรือไม่ อย่างไร เพราะเหตุใด
9. จงบอกความต่างศักย์ของซีพียูคอร์ หน่วยความจำ และอินพุตเอาท์พุตและเปรียบเทียบกันว่าแตกต่างกันหรือไม่ อย่างไร เพราะเหตุใด
10. จงบอกอุณหภูมิของซีพียูและตำแหน่งอื่นๆ ว่าทำงานที่กีองศาเซลเซียส และเปรียบเทียบกันว่าแตกต่างกันหรือไม่ อย่างไร เพราะเหตุใด
11. จงบอกเวอร์ชันและรายละเอียดอื่นๆ ของระบบปฏิบัติการ Raspbian ที่ติดตั้ง

Appendix E

การทดลองที่ 5 การพัฒนาโปรแกรมด้วยภาษา C

การทดลองนี้ค่าด้วงผู้อ่านเคยเรียนการเขียนหรือพัฒนาโปรแกรมด้วยภาษา C มาบ้างแล้ว และมีความคุ้นเคยกับ IDE (Integrated Development Environment) จากพัฒนาโปรแกรมและการติดตั้งโปรแกรมด้วยภาษา C/C++ ดังนั้น การทดลองมีวัตถุประสงค์เหล่านี้

- เพื่อให้เข้าใจการพัฒนาซอฟต์แวร์ด้วย IDE ชื่อ Code::Blocks บนระบบปฏิบัติการ Raspbian/Linux/Unix
- เพื่อให้เข้าใจความแตกต่างระหว่างการพัฒนาโปรแกรมภาษา C ด้วย Integrated Development Environment และ Makefile
- เพื่อให้สามารถสร้าง Makefile เพื่อความสะดวกและพัฒนาศักยภาพการทำงานเป็นนักพัฒนาอาชีพ

E.1 การพัฒนาโดยใช้ IDE

โปรแกรมหรือแอ��พลิเคชัน IDE ย่อมาจาก Integrated Development Environment IDE ทำหน้าที่ช่วยเหลือโปรแกรมเมอร์ ทดสอบ และควบคุมซอฟต์แวร์ได้ให้เป็นปัจจุบัน ขั้นตอนการทดลองนี้เริ่มต้นโดย

- ติดตั้ง Code::Blocks ผู้อ่านต้องพิมพ์คำสั่งเหล่านี้ลงบนโปรแกรม Terminal

```
$ sudo apt-get install codeblocks
```

คำสั่ง sudo นำหน้าคำสั่งใดๆ นี้จะเป็นการเรียกใช้งานคำสั่งนั้นด้วยสิทธิ์ระดับ superuser การติดตั้งจะดาวน์โหลดโปรแกรมผ่านทางเครือข่ายอินเทอร์เน็ต จำเป็นต้องใช้สิทธิ์ระดับสูงสุดนี้

- พิมพ์คำสั่งนี้ เพื่อเริ่มต้นใช้งาน Code::Blocks

```
$ codeblocks
```

- การใช้งาน Code::Blocks ครั้งแรกจะเป็นการติดตั้งค่า compiler plug-ins เป็น GNU GCC compiler.

Appendix E. การทดลองที่ 5 การพัฒนาโปรแกรมด้วยภาษา C

4. หน้าต่างหลักจะปรากฏขึ้น หลังจากนั้น ผู้อ่านควรกด "Create a new project" เพื่อสร้างໂປຣເຈັກທີ່ໃໝ່ໃນหน้าต່າງ "New from template"

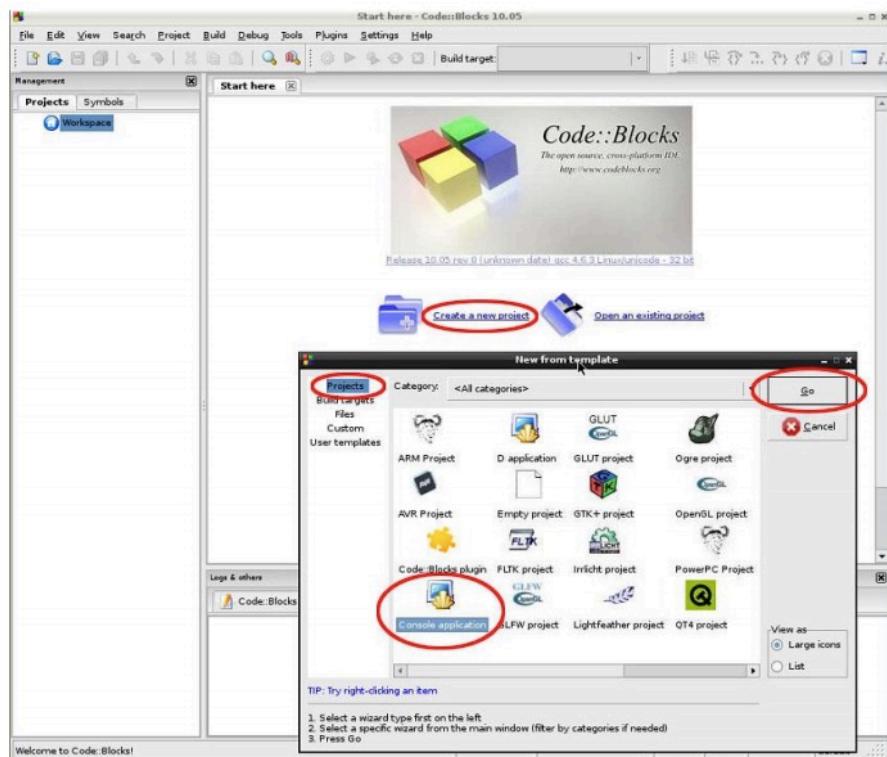


Figure E.1: หน้าต່າງເລືອກໜິດໂປຣເຈັກທີ່ຈະພັນນາເປັນໜິດ "Console application"

5. เลือก 'New Projects' ในช่องด้านซ้าย และเลือก "Console application" ในช่องด้านขวาเพื่อสร้างໂປຣແກມໃນรูปແບບເທົກໝາຍ (Text Mode) กดປຸ່ມ "Go" ตามຮູບທີ່ E.1

6. กดປຸ່ມ Next> เพื่อดໍາເນີນການຕ້ອ

7. หน้าต່າງ "Console application" จะปรากฏขຶ້ນ กดເລືອກພາສາ "C" เพื่อພັນນາໂປຣແກມກ່ອນແລ້ວກຳປຸ່ມ "Next>" ตามຮູບທີ່ E.2



Figure E.2: หน้าต่างเลือกภาษาสำหรับโปรเจ็คท์ที่จะพัฒนา

8. กรอกชื่อโปรเจ็คใหม่ชื่อ Lab5 ในช่อง Project title: และกรอกชื่อโฟลเดอร์ /home/pi/c/ ในช่อง Folder to create project in: โปรดสังเกตข้อความในช่อง Project filename: ว่าตรงกับ Lab5.cbp ใช่หรือไม่
9. กดปุ่ม "Next >" เพื่อดำเนินการต่อและสุดท้ายจะเป็นขั้นตอนการเลือกคอนฟิกกูเรชัน (Configuration) สำหรับคอมไพเลอร์ในรูปที่ E.3 โดย Debug เหมาะสำหรับการเริ่มต้นและแก้ไขข้อผิดพลาด แล้วจึงกดปุ่ม "Finish" เมื่อเสร็จสิ้น

Appendix E. การทดลองที่ 5 การพัฒนาโปรแกรมด้วยภาษา C

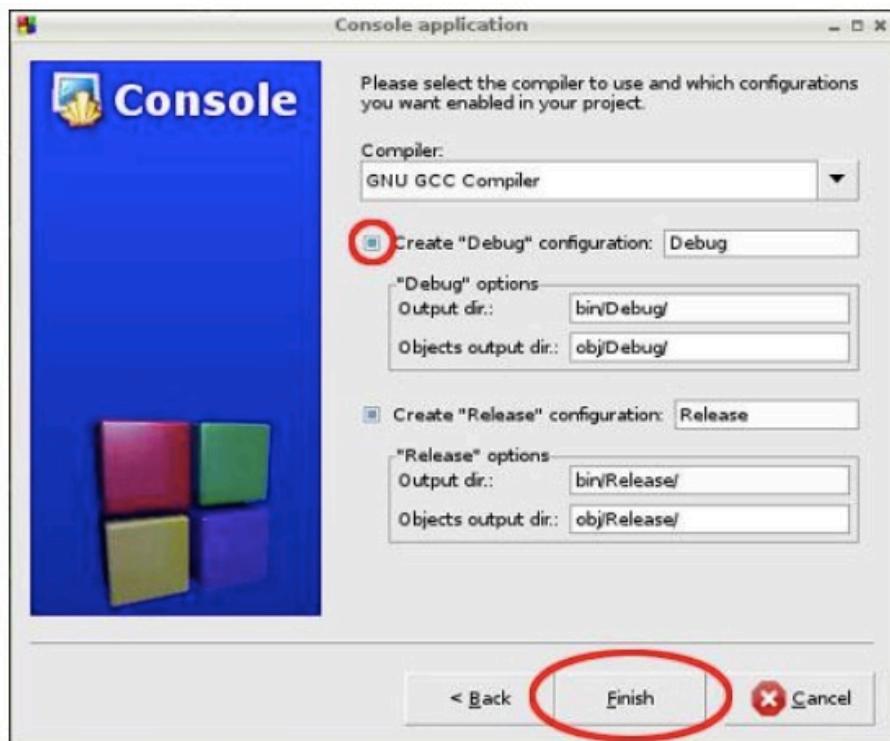


Figure E.3: การเลือกค่า Debug สำหรับคอมไพล์อร์ GNU GCC ในโปรเจ็คท์ Lab5

10. เพิ่มไฟล์เปล่าด้วยเมนูต่อไปนี้ File->New->Empty file ตามรูปที่ E.4



Figure E.4: การเพิ่มไฟล์เปล่าให้กับโปรเจ็คท์ Lab5 ที่สร้างขึ้น

11. ป้อนโปรแกรมนี้ลงในหน้าต่าง 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;
}
```

12. คอมไพล์และ Build โปรแกรม จนไม่มีข้อผิดพลาด โดยสังเกตจากหน้าต่างด้านล่างสุด

13. รันโปรแกรมเพื่อทดสอบการทำงาน

E.2 การดีบัก (Debugging) โดยใช้ IDE

การดีบักโปรแกรม คือ การตรวจสอบการทำงานของโปรแกรมอย่างละเอียด code::Blocks รองรับการดีบัก ผ่านเมนู Debug ผู้อ่านสามารถเริ่มต้นโดย

1. กด Debug บนเมนูแถบลูกปัด เลือก Active Debuggers ซึ่งค่าปัจจุบัน (Target's Default) คือ GDB/CDB Debugger
2. ตั้งเบรกพอยท์ (Break Point) ตรงประโยชน์ที่ต้องการศึกษา โดยเลื่อนเมาส์ไปยังบรรทัดนั้น แล้วกดปุ่ม F5 โปรดสังเกตต้นประโยชน์จะมีวงกลมสีแดงปรากฏขึ้น และเมื่อกด F5 อีกครั้งจะมีวงกลมสีแดงจะหายไป เรียกว่า การทือเกล (Toggle) เบรกพอยท์ กด F5 อีกครั้งเพื่อสร้างวงกลมสีแดง เพียงจุดเดียวเท่านั้น
3. กด F8 บนคีย์บอร์ดเพื่อรันโปรแกรม โปรแกรมจะรันไปจนหยุดตรงประโยชน์ที่มีวงกลมสีแดงนั้น โปรดสังเกตสัญลักษณ์สามเหลี่ยมสีเหลืองซ่อนทับกันอยู่ หลังจากนั้น กด F8 เพื่อดำเนินการต่อ
4. เลื่อนเมาส์ไปยังประโยชน์ที่มีวงกลมสีแดง กดปุ่ม F5 บนคีย์บอร์ดเพื่อปลดวงกลมสีแดงออก
5. เริ่มต้นใหม่เพื่อศึกษาการทำงานของ F4 (Run to cursor) โดยเลื่อนเมาส์ไปวางบนประโยชน์ที่สนใจ กดปุ่ม F4 และสังเกตว่า สามเหลี่ยมสีเหลืองจะปรากฏหน้าประโยชน์ เพื่อระบุว่าเครื่องรันมาถึงประโยชน์นี้แล้ว
6. กด F8 เพื่อรันต่อไป จนสิ้นสุดการทำงานของโปรแกรม

E.3 การพัฒนาโดยใช้ประโยชน์คำสั่งทีละขั้นตอน

ผู้อ่านควรเข้าใจคำสั่งพื้นฐานในการแปลงโปรแกรมภาษาแอลกอริتمบล็อกที่สร้างขึ้นใน Code::Blocks ก่อนหน้านี้ ตามขั้นตอนต่อไปนี้

1. เปิดโปรแกรม Terminal หน้าต่างใหม่ แล้วบันทึกไฟล์เดอร์ไปยัง /home/pi/c/Lab5 โดยใช้คำสั่ง cd

2. คอมpile ไฟล์ชอร์สโค๊ดให้เป็นไฟล์อ๊อบเจ็คท์ โดยเรียกใช้คอมไพล์เวอร์ชื่อ gcc ดังนี้

```
$ gcc -c main.c
```

ไฟล์ผลลัพธ์ ชื่อ main.o จะปรากฏขึ้น ผู้อ่านต้องตรวจสอบโดยใช้คำสั่ง ls -la เพื่อตรวจสอบวันที่และขนาดของไฟล์

Appendix E. การทดลองที่ 5 การพัฒนาโปรแกรมด้วยภาษา C

3. ทำการลิงค์และแปลงไฟล์อืบเจ็คที่เป็นไฟล์โปรแกรมโดย

```
$ gcc -g main.o -o Lab5
```

4. เรียกโปรแกรม Lab5 โดยพิมพ์

```
$ ./Lab5
```

5. เปรียบเทียบผลลัพธ์ที่ปรากฏขึ้นว่าตรงกับผลการรันใน IDE หรือไม่ อย่างไร

E.4 โครงสร้างของ Makefile

นอกเหนือจากการพัฒนาโปรแกรมด้วย IDE และ การพัฒนาด้วย Makefile จะช่วยให้นักพัฒนามีสมัครเล่นและมืออาชีพในการได้ถูกต้องและรวดเร็ว ไฟล์ชื่อ Makefile เป็นไฟล์อักขระหรือเท็กซ์ไฟล์ (text file) ง่ายๆ ที่อธิบายความสัมพันธ์ระหว่าง ไฟล์ซอร์สโค้ดต่างๆ ไฟล์อืบเจ็ค์ และไฟล์โปรแกรม แต่ละบรรทัดจะมีโครงสร้างดังนี้

```
target : prerequisites ...
<tab>recipe
<tab>      ...
<tab>...
```

- target หมายถึง ชื่อไฟล์ที่จะถูกสร้างขึ้น โดยอาศัยไฟล์ต่างๆ จากส่วนที่เรียกว่า prerequisites นอกจากชื่อไฟล์แล้ว คำสั่งง่าย เช่น 'clean' สามารถใช้เป็น target ได้ ซึ่งนิยมใช้สำหรับลบไฟล์ต่างๆ ที่ไม่ต้องการ
- recipe หมายถึง คำสั่งหรือการกระทำที่จะใช้รายชื่อไฟล์ใน prerequisites นั้นมาสร้างไฟล์ target ได้สำเร็จ โดยแต่ละบรรทัดจะต้องเริ่มต้นด้วยปุ่ม tab เสมอ

E.5 การพัฒนาโดยใช้ Makefile

ตัวอย่างนี้เป็นการสร้าง Makefile เพื่อคอมไพล์โปรแกรมเดิมที่เรามีอยู่ ผู้อ่านจะได้เข้าใจกลไกการทำงานที่ง่ายที่สุดก่อน หลังจากนั้นผู้อ่านสามารถศึกษาเพิ่มเติมด้วยตนเองได้จากเว็บไซต์หรือตัวอย่างโปรแกรม Open Source ที่ซับซ้อนขึ้นเรื่อยๆ ต่อไป

1. จงสร้าง makefile ในโฟลเดอร์ /home/pi/c/Lab5 โดยกรอกข้อความเหล่านี้ในไฟล์ใหม่

ឧបករណ៍ ក្រុមហ៊ុនកម្ពុជា

Lab5: main.o

gcc -g main.o -o Lab5 សាន្តិសារ

main.o: main.c

gcc -c main.c

clean :

rm *.o

2. เมื่อกรอกเสร็จแล้ว ให้ทำการบันทึก หรือ save โดยตั้งชื่อไฟล์ว่า Makefile โดยไม่มีนามสกุล หลังจากนั้น รันคำสั่งต่อไปนี้ใน Terminal ใน /home/pi/c/Lab5

3. พิมพ์คำสั่งนี้ใน Terminal

\$ make clean

เป็นการเรียกใช้คำสั่ง rm *.o ผ่านทาง Makefile เพื่อลบ (Remove) ไฟล์ที่มีนามสกุล .o ทั้งหมด

4. พิมพ์คำสั่งนี้ใน Terminal

\$ make Lab5

เป็นการเรียกใช้คำสั่ง gcc -c main.c และ gcc -g main.c -o Lab5 เพื่อสร้างไฟล์คำสั่ง Lab5 ที่จะทำงานตามชอร์สโค้ด main.c ที่กรอกไป โดยไฟล์ Lab5 ที่เกิดขึ้นใหม่จะมีโครงสร้างรูปแบบ ELF

5. พิมพ์คำสั่งนี้ใน Terminal

\$ ls -la

เพื่ออ่านค่าเวลาที่ไฟล์ Lab5 ที่เพิ่งถูกสร้าง โปรดสังเกตสิ่งชื่อไฟล์ต่างๆ ว่ามีสีอะไรบ้าง และบ่งบอกอะไรมานะ

6. พิมพ์คำสั่งนี้ใน Terminal

\$./Lab5

เป็นการเรียกใช้คำสั่ง Lab5 ให้ชีพួយปฏิบัติตาม

E.6 กิจกรรมท้ายการทดลอง

1. จงพัฒนาโปรแกรมภาษา C ให้สามารถอ่านไฟล์ Makefile เพื่อแสดงตัวอักษรในไฟล์ทีละตัวและค่ารหัส ASCII ฐานสิบหกของตัวอักษรนั้นบนหน้าจอ แล้วปิดไฟล์เมื่อเสร็จสิ้น
2. จงพัฒนาโปรแกรมภาษา C เพื่อสั่งพิมพ์เลขอนุกรม Fibonacci โดยรับค่าเลขเป้าหมาย n ซึ่งเกิดจาก $n = (n-1) + (n-2)$ และรายละเอียดเพิ่มเติมได้จาก [wikipedia](#) ตัวอย่างต่อไปนี้ $n=5$ และพิมพ์ผลลัพธ์ดังนี้

1 1 2 3 5

Appendix F

การทดลองที่ 6 การพัฒนาโปรแกรมภาษาแอลซีช์เมบลี

การทดลองนี้คาดว่าผู้อ่านเคยเรียนการเขียนหรือพัฒนาโปรแกรมด้วยภาษา C ในการทดลองที่ 5 ภาค พนวก E แล้ว และมีความคุ้นเคยกับ IDE จากพัฒนาโปรแกรมและการดีบักโปรแกรมด้วยภาษา C/C++ ดังนั้น การทดลองมีวัตถุประสงค์เหล่านี้

- เพื่อให้เข้าใจการพัฒนาและดีบัก (Debug) โปรแกรมภาษาแอลซีช์เมบลีด้วย IDE ชื่อ Code::Blocks บนระบบปฏิบัติการ Raspbian/Linux/Unix
- เพื่อให้เข้าใจความแตกต่างระหว่างการพัฒนาโปรแกรมภาษาแอลซีช์เมบลีด้วย IDE และ Makefile

F.1 การพัฒนาโดยใช้ IDE

- พิมพ์คำสั่งนี้ในโปรแกรม Terminal เพื่อเริ่มต้นใช้งาน Code::Blocks

```
$ codeblocks
```

- หน้าต่างหลักจะปรากฏขึ้น หลังจากนั้น ผู้อ่านสามารถสร้างโปรเจ็คใหม่โดยเลือก "Create a new project" ในช่องด้านซ้าย แล้วเลือก "Console application" ในช่องด้านขวาเพื่อสร้างโปรแกรม
- กรอกชื่อโปรเจ็คใหม่ชื่อ Lab6 ในช่อง Project title: และกรอกชื่อโฟลเดอร์ /home/pi/asm/ ในช่อง Folder to create project in: โปรดสังเกตข้อความในช่อง Project filename: ว่าตรงกับ Lab6.cbp ใช่หรือไม่
- โปรแกรม code blocks จะสร้างโฟลเดอร์ต่างๆ ภายใต้โฟลเดอร์ชื่อ /home/pi/asm/Lab6/
- กดปุ่ม "Next>" เพื่อดำเนินการต่อและสุดท้ายจะเป็นขั้นตอนการเลือกคอนฟิกกูเรชัน (Configuration) สำหรับคอมไฟเลอร์ เลือกออพชัน Debug เหมาะสำหรับการเริ่มต้นและแก้ไขข้อผิดพลาด แล้วจึงกดปุ่ม "Finish" เมื่อเสร็จสิ้น

Appendix F. การทดลองที่ 6 การพัฒนาโปรแกรมภาษาแอลเซมบลี

6. กดชื่อ Workspace ในหน้าต่างด้านซ้ายเพื่อขยายโครงสร้างโปรเจคท์เพื่อค้นหาไฟล์ "main.c" คลิกขวาบนชื่อไฟล์ แล้วเลือกเมนู "Remove file from project" ตามรูปที่ F.1

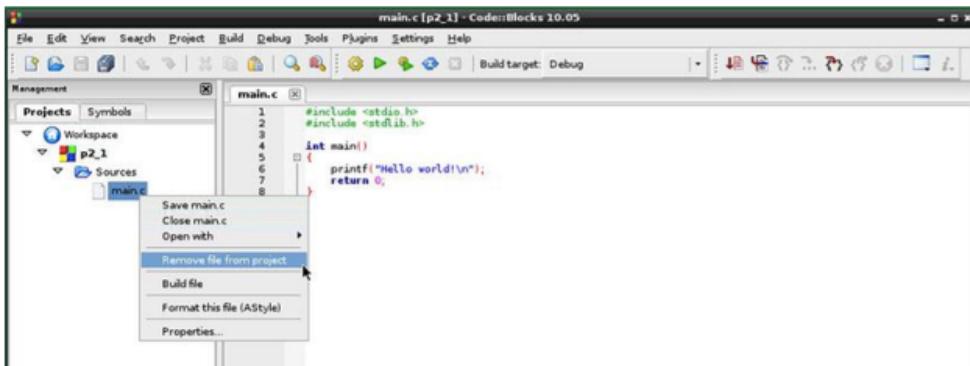


Figure F.1: การย้ายไฟล์ main.c ออกจากโปรเจคท์

7. เพิ่มไฟล์ใหม่ลงในโปรเจคโดยกดเมนู File->New->Empty file ตามรูปที่ F.2

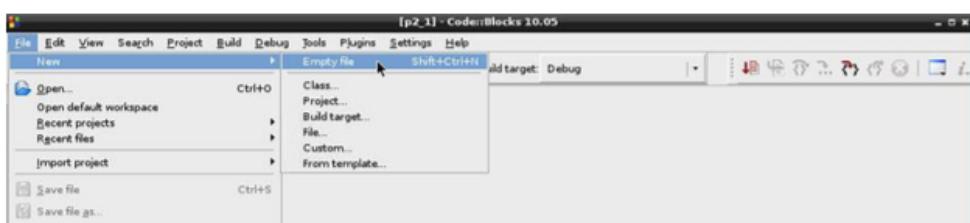


Figure F.2: การเพิ่มไฟล์ใหม่ลงในโปรเจคท์

8. กดปุ่ม "Yes" เพื่อยืนยันในรูปที่ F.3

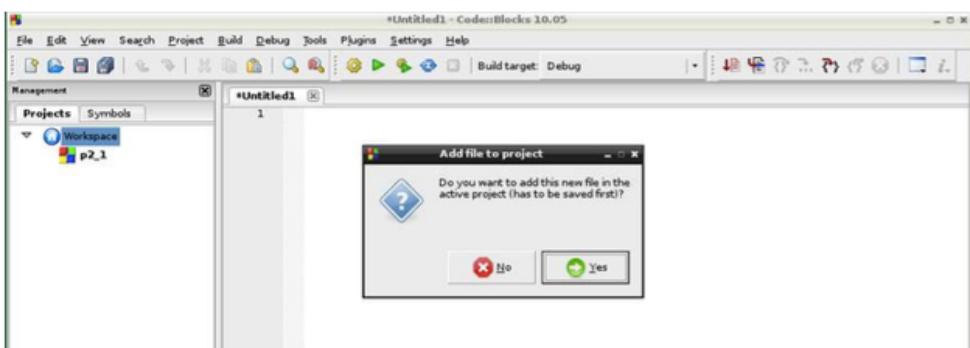


Figure F.3: หน้าต่างกดปุ่ม "Yes" เพื่อยืนยัน

9. หน้าต่าง "Save file" จะปรากฏขึ้น กรอกชื่อไฟล์ว่า main.s และจึงกดปุ่ม "Save" ดังรูปที่ F.4

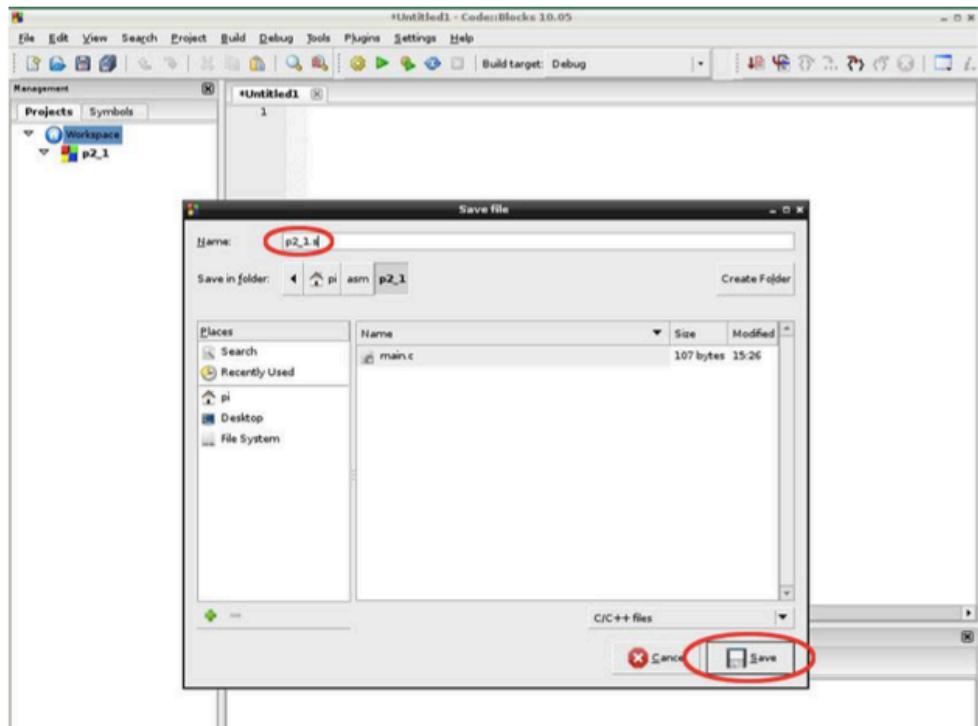


Figure F.4: หน้าต่าง Save File ชื่อไฟล์ว่า main.s

10. เพิ่ม (Add) ไฟล์ main.s เข้าไปในโปรเจ็คท์

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 เพื่อแปลงโปรแกรมที่เขียนให้เป็นโปรแกรมภาษาเครื่อง

13. เลือกเมนู Build->Run เพื่อรันโปรแกรม

14. อ่านและบันทึกประโยคที่เกิดขึ้นในหน้าต่าง Terminal ที่ผลลัพธ์มา

(ต่อจาก)

Process returned 6 (0x06) execution time : 0.10 s
Press Enter to continue

F.2 การดีบักโปรแกรมโดยใช้ IDE

1. เลื่อนปุ่มเมาส์ขวาไปบรรทัดที่มีคำสั่ง ORR R0, R1, R2 คลิกเมนู Debug->breakpoint หรือกดปุ่ม F5 ผู้อ่านจะสังเกตว่ากลมสีแดงปรากฏขึ้นด้านซ้าย
2. กดเมนู Debug->Debugging Windows->CPU Registers เพื่อแสดงค่าของ CPU register ในหน้าต่างที่ปรากฏขึ้นมาเพิ่มเติม
3. เมื่อพร้อมแล้ว ผู้อ่านสามารถเริ่มต้นการดีบักโดยกดเมนู Debug->Start/Continue หรือกดปุ่ม F8 โปรแกรมจะเริ่มต้นทำงานตั้งแต่ประযุกแกรужนหยุดที่บรรทัดที่มีคำสั่ง ORR R0, R1, R2
4. อ่านและบันทึกค่าของ R0 และ PC ในหน้าต่าง CPU Registers
5. ประมาณผลคำสั่งถัดไปโดยกดเมนู Debug->Next Instruction หรือปุ่ม Alt+F7
6. อ่านและบันทึกค่าของ R0 และ PC ในหน้าต่าง CPU Registers และสังเกตการเปลี่ยนแปลงที่เกิดขึ้น
 $\text{MOV R0, #0} \rightarrow \text{ห้าม R0 ไม่เป็น } 0$
 $\text{MOV R1, #2} \quad R_1 = 2$
 $\text{MOV R2, #4} \quad R_2 = 4$
 $\text{ORR R0, R1, R2} \quad R_1 \parallel R_2 = 6$
7. อธิบายว่าเกิดอะไรขึ้น

F.3 การพัฒนาโดยใช้ประโยชน์คำสั่งที่ลับซ่อน

ผู้อ่านควรเข้าใจคำสั่งพื้นฐานในการแปลงโปรแกรมภาษาแอสเซมบลีที่สร้างขึ้นใน Code::Blocks ก่อนหน้านี้ ตามขั้นตอนต่อไปนี้

1. ใช้โปรแกรมไฟล์เมเนจอร์เพื่อเบรนไฟล์ในไฟล์เดอร์ /home/pi/asm/Lab6
2. ดับเบิลคลิกบนชื่อไฟล์ main.s เพื่อเปิดอ่านไฟล์และเปรียบเทียบกับไฟล์ที่เขียนในโปรแกรม Code::Blocks
3. เปิดโปรแกรม Terminal หน้าต่างใหม่ แล้วนำไฟล์เดอร์ไปยัง /home/pi/asm/Lab6 โดยใช้คำสั่ง cd
4. แปลไฟล์ซอร์สโค้ดให้เป็นไฟล์อับเจคท์ โดยเรียกใช้คำสั่ง as (assembler) ดังนี้
 $\$ \text{as -o main.o main.s}$
5. ทำการลิงค์และแปลงไฟล์อับเจคท์เป็นไฟล์โปรแกรมโดย
 1. ลิงค์ไฟล์ main.o ด้วยคำสั่ง gcc
 2. แปลงไฟล์ main.o ให้เป็นไฟล์ ELF
 3. สร้างไฟล์ executable ด้วยคำสั่ง ./main
6. เรียกโปรแกรม Lab6 โดยพิมพ์

```
$ ./Lab6
```

```
$ echo $?
```

7. เปรียบเทียบหมายเลขที่ปรากฏขึ้นว่าตรงกับผลการรันใน IDE หรือไม่ อย่างไร

F.4 การพัฒนาโดยใช้ Makefile

การใช้ **makefile** สำหรับพัฒนาโปรแกรมภาษาแอสเซมบลี มีความเหมือนกับการทดลองที่ 5 ในภาค พนวก E

1. เปิดไปยังโฟลเดอร์ /home/pi/asm/Lab6 ด้วยโปรแกรมไฟล์เมเนจเจอร์
2. กดปุ่มขวาบนมาส์ในพื้นที่โฟลเดอร์เพื่อสร้างไฟล์เปล่าใหม่ (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
5. ลบไฟล์อ้อมเบ็คที่มีอยู่โดยใช้คำสั่ง clean ในโปรแกรม Terminal
6. ทำการแปลโดยใช้คำสั่ง make ในโปรแกรม Terminal
7. เรียกโปรแกรม Lab6 โดยพิมพ์

```
$ ./Lab6
```

```
$ echo $?
```

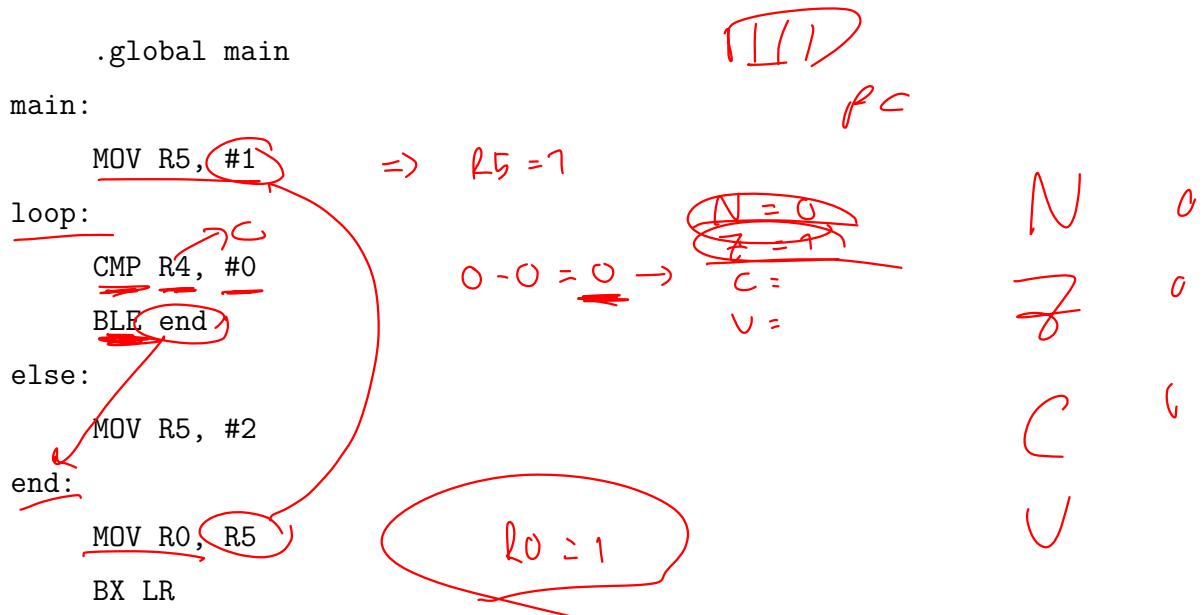
F.5 กิจกรรมท้ายการทดลอง

1. จงปรับแก้คำสั่ง ORR เป็นคำสั่ง AND ในโปรแกรม main.s ทดสอบเพื่อตรวจสอบการเปลี่ยนแปลง อธิบายผลการทดสอบ

0 เนื่องจากตัวสูตรค่า 0 2 & 4 = 0

2. จงปรับแก้โปรแกรมใน main.s ให้รีเทิร์นค่าอื่นแทน

3. จงปรับแก้โปรแกรมใน main.s เป็นดังนี้: $MOV R0, \#value$



จดบันทึกผลการทดสอบและอธิบาย.

4. จงปรับแก้โปรแกรมใน main.s เป็นดังนี้:

```

.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
    
```

value at [address] found in VR1[addr] is load to R2
value found in R1 is store to [addr] from in R2

*Appendix F. การทดลองที่ 6 การพัฒนาโปรแกรมภาษาอาเซียนบลี
จดบันทึกผลการทดสอบและอธิบาย*

Appendix G

การทดลองที่ 7 การสร้างฟังค์ชันในโปรแกรมภาษาแอสเซมบลี

ผู้อ่านควรจะต้องอ่านเนื้อหาของบทที่ 4 และ ทำการทดลองที่ 5 และการทดลองที่ 6 มา ก่อน โดยการทดลองนี้จะเสริมความเข้าใจของผู้อ่านให้เพิ่มมากขึ้น ดังนั้น การทดลองมีวัตถุประสงค์เหล่านี้

- เพื่อพัฒนาโปรแกรมภาษาแอสเซมบลีร่วมกับตัวแปรใน Data Segment
- เพื่อพัฒนาโปรแกรมแอสเซมบลีโดยใช้ตัวแปรอยู่ใน Data Segment
- เพื่อพัฒนาโปรแกรมภาษาแอสเซมบลีร่วมกับฟังค์ชันสำเร็จรูป

G.1 การใช้งานตัวแปรใน Data Segment

ตัวแปรต่างๆ ที่ประกาศโดยใช้ชื่อ เลเบล ต้องการพื้นที่ในหน่วยความจำสำหรับจัดเก็บค่าของมัน ตัวแปร มีสองชนิด คือ

- ตัวแปรชนิดโกลบอล (Global Variable) พื้นที่สำหรับเปิดให้ตัวแปรเหล่านี้ เรียกว่า ดาตาเช็กเมนท์ (Data Segment) ซึ่งผู้เขียนได้กล่าวไปแล้วในบทที่ 4 และ
- ตัวแปรชนิดโลคอล (Local Variable) อาศัยพื้นที่ภายในสแต็คเช็กเมนท์ (Stack Segment) ใน การจัดเก็บค่าชั่วคราว เนื่องจากฟังค์ชันคือโปรแกรมย่อที่ฟังค์ชัน main() เป็นผู้เรียกใช้ และเมื่อ ทำงานเสร็จสิ้น ฟังค์ชันใดๆ จะต้องรีเทิร์นกลับมาหาฟังค์ชัน main() ในที่สุด ดังนั้น ตัวแปรชนิดโลคอล จึงไม่จำเป็นต้องใช้พื้นที่ในดาตาเช็กเมนท์

G.1.1 การโหลดค่าตัวแปรจากหน่วยความจำ

- ย้ายไดเรคทอรีไปยัง \$ cd /home/pi/Assembly
- สร้างไดเรคทอรี Lab7 ภายใต้ \$ cd /home/pi/Assembly

Appendix G. การทดลองที่ 7 การสร้างฟังค์ชันในโปรแกรมภาษาแอสเซมบลี

3. ย้ายไดเรกทอรีเข้าไปใน Lab7

4. ตรวจสอบว่าไดเรกทอรีปัจจุบันโดยใช้คำสั่ง `pwd`

5. สร้างไฟล์ `Lab7_1.s` ตามโค้ดต่อไปนี้ ผู้อ่านสามารถข้ามประযุคคอมเม้นท์ได้ เมื่อทำการเข้าใจแต่ละคำสั่งแล้ว

```
.data
    .balign 4          @ Request 4 bytes of space
fifteen: .word 15      @ fifteen = 15

    .balign 4          @ Request 4 bytes of space
thirty:  .word 30     @ thirty = 30

.text
    .global main
main:
    LDR R1, addr_fifteen    @ R1 <- address_fifteen
    LDR R1, [R1]             @ R1 <- Mem[R1]
    LDR R2, addr_thirty    @ R2 <- address_thirty
    LDR R2, [R2]             @ R2 <- Mem[R2]
    ADD R0, R1, R2

end:
    MOV R7, #1
    SWI 0

addr_fifteen: .word fifteen
addr_thirty:  .word thirty
```

6. ทำการ `make` และรันโปรแกรมโดยใช้คำสั่ง

```
$ ./Lab7_1
$ echo $?
```

บันทึกผลและอธิบายผลที่เกิดขึ้น

7. สร้างไฟล์ **Lab7_2.s** ตามโค้ดต่อไปนี้จากไฟล์ **Lab7_1.s** ผู้อ่านสามารถข้ามประโยชน์คอมเม้นท์ได้ เมื่อทำความเข้าใจแต่ละคำสั่งแล้ว

```

.data
    .balign 4          @ Request 4 bytes of space
fifteen: .word 0           @ fifteen = 0
    .balign 4          @ Request 4 bytes of space
thirty: .word 0            @ thirty = 0

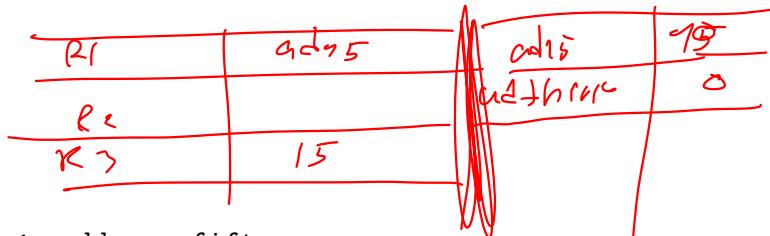
.text
.global main
main:
    LDR R1, addr_fifteen  @ R1 <- address_fifteen
    MOV R3, #15             @ R3 <- 15
    STR R3, [R1]             @ Mem[R1] <- R3
    LDR R2, addr_thirty    @ R2 <- address_thirty
    MOV R3, #30              @ R3 <- 30
    STR R3, [R2]             @ Mem[R2] <- R2

    LDR R1, addr_fifteen  @ Load address
    LDR R1, [R1]             @ R1 <- Mem[R1]
    LDR R2, addr_thirty    @ Load address
    LDR R2, [R2]             @ R2 <- Mem[R2]
    ADD R0, R1, R2

end:
    MOV R7, #1
    SWI 0

@ Labels for addresses in the data section
addr_fifteen: .word fifteen
addr_thirty: .word thirty

```



8. ทำการ make และรันโปรแกรมโดยใช้คำสั่ง

```

$ ./Lab7_2
$ echo $?

```

Appendix G. การทดลองที่ 7 การสร้างฟังค์ชันในโปรแกรมภาษาแอสเซมบลี

บันทึกผลและอธิบายผลที่เกิดขึ้นเพื่อเปรียบเทียบกับข้อที่แล้ว

G.1.2 การใช้งานตัวแปรอะเรย์ชนิดต่างๆ ใน Data Segment

ชนิดของตัวแปรจะกำหนดตามหลังชื่อตัวแปร เช่น .word, .hword และ .byte ใช้กำหนดขนาดของตัวแปรนั้นๆ ขนาด 32, 16 และ 8 บิตตามลำดับ ยกตัวอย่าง คือ:

```
numbers:    .word 1,2,3,4
```

เป็นการประกาศและตั้งค่าตัวแปรชนิดอะเรย์ของเวิร์ด ซึ่งต้องการพื้นที่ 4 ไบท์ต่อข้อมูลแต่ละค่า ซึ่งจะตรงกับประโยชน์ต่อไปนี้ในภาษา C

```
int numbers={1,2,3,4}
```

1. สร้างไฟล์ Lab7_3.s ตามโค้ดต่อไปนี้ ผู้อ่านสามารถข้ามประโยชน์คอมเม้นท์ได้ เมื่อทำการเข้าใจแต่ละคำสั่งแล้ว

```
.data  
primes:  
.word 2  
.word 3  
.word 5  
.word 7  
  
.text  
.global main  
  
main:  
    LDR R3, =primes    @ Load the address for the data in R3  
    LDR R0, [R3, #4]    @ Get the next item in the list  
  
end:  
    MOV R7, #1  
    SWI 0
```

2. รันโปรแกรม บันทึกและอธิบายผลลัพธ์

G.1.3 การใช้งานตัวแปรอะเรย์ชนิด Byte

คำสั่ง LDRB ทำงานคล้ายกับคำสั่ง LDR แต่เป็นการอ่านค่าของตัวแปรอะเรย์ชนิด Byte

- ป้อนคำสั่งต่อไปนี้

```
.data
numbers: .byte 1, 2, 3, 4, 5

.text
.global main
main:
    LDR R3, =numbers      @ Get address
    LDRB R0, [R3, #2]      @ Get next byte
end:
    MOV R7, #1
    SWI 0
```

- รันโปรแกรม บันทึกและอธิบายผลลัพธ์

G.2 การเรียกใช้ฟังค์ชันสำหรับรูปแบบตัวแปรชนิดประโยค

ฟังค์ชันสำหรับรูปแบบตัวแปรชนิดประโยคที่เข้าใจง่ายและใช้สำหรับเรียนรู้การพัฒนาโปรแกรมภาษา C เป็นต้น คือ ฟังค์ชัน printf ซึ่งถูกกำหนดอยู่ในไฟล์ヘดเดอร์ stdio.h ตามตัวอย่างซอฟต์แวร์สโค็ต ในรูปที่ 3.16 และการทดลองที่ 5 ภาค พนวก E ในการทดลองต่อไปนี้ ผู้อ่านจะสังเกตเห็นว่าการเรียกใช้ฟังค์ชัน printf ในภาษาแอสเซมบลี โดยอาศัยตัวแปรชนิดประโยค โดยใช้คำสำคัญ (Key Word) เหล่านี้ คือ .ascii และ .asciz ตัวแปรชนิด asciz จะมีตัวอักษรพิเศษ เรียกว่า อักขระ NULL ปิดท้ายประโยคเสมอ และอักขระ NUll จะมีรหัส ASCII เท่ากับ 00₁₆ ตามตารางรหัส ASCII ในรูปที่ 1.12

- กรอกคำสั่งต่อไปนี้ลงในไฟล์ชื่อ Lab7_4.s และทำความเข้าใจประโยคคอมเม้นท์แต่ละบรรทัด

```
.data
.balign 4
question: .asciz "What is your favorite number?"

.balign 4
message: .asciz "%d is a great number \n"

.balign 4
```

Appendix G. การทดลองที่ 7 การสร้างฟังก์ชันในโปรแกรมภาษาแอสเซมบลี

pattern: .asciz "%d"

.balign 4

number: .word 0

.balign 4

lr_bu: .word 0

.text

© Used by the compiler to tell libc where main is located

.global main

.func main

main:

© Keep the value inside Link Register

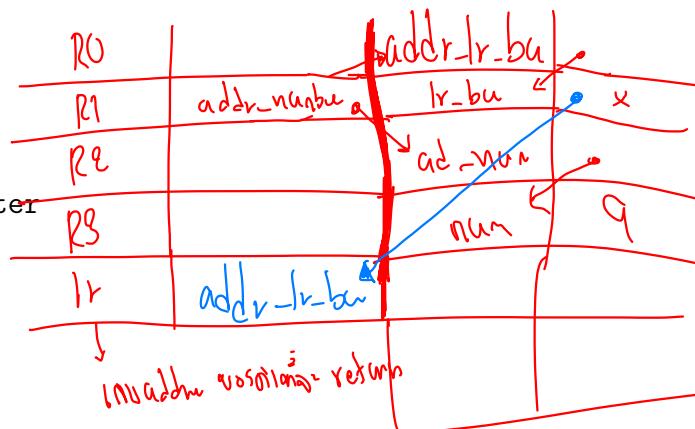
LDR R1, addr_lr_bu

STR lr, [R1] © Mem[R1] <- LR

© Load and print question

LDR R0, addr_question

BL printf



© Define pattern for scanf and where to store it

LDR R0, addr_pattern 1, q,

LDR R1, addr_number ๘๙๘๗

BL scanf

© Display the message together with number

LDR R0, addr_message

LDR R1, addr_number

LDR R1, [R1]

BL printf

© Restore the saved value to link register

LDR lr, addr_lr_bu

Appendix G. การทดลองที่ 7 การสร้างฟังค์ชันในโปรแกรมภาษาแอสเซมบลี

```
LDR lr, [lr]      @ LR <- Mem[addr_lr_bu]  
BX lr             @ Return to main function
```

@ Define addresses

```
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. สร้าง makefile ภายใต้เครื่องทอยี Lab7 และกรอกคำสั่งดังนี้

Lab7_4:

```
gcc -o Lab7_4 Lab7_4.s
```

3. เรียกใช้ make โปรดสังเกตความแตกต่างที่แสดงผลและใน makefile ที่ผ่านมา

G.3 การสร้างฟังค์ชันด้วยภาษาแอสเซมบลี

หัวข้อที่ 4.8 อธิบายไฟล์การทำงานของฟังค์ชัน โดยอาศัย การใช้งานรีจิสเตอร์ R0 - R12 ดังนี้

- รีจิสเตอร์ R0, R1, R2, และ R3 การส่งผ่านพารามิเตอร์ผ่านทางรีจิสเตอร์ R0 ถึง R3 ตามลำดับ หากฟังค์ชันบางตัวต้องการพารามิเตอร์จำนวนมากกว่า 4 ค่า โปรแกรมเมอร์สามารถส่งผ่านทางสเต็คโดยคำสั่ง PUSH หรือคำสั่งที่ใกล้เคียง
- รีจิสเตอร์ R0 สำหรับเริ่มต้นหรือส่งค่ากลับไปทางฟังค์ชันที่เรียกใช้มัน
- R4 - R12 สำหรับการใช้งานทั่วไป การใช้งานรีจิสเตอร์เหล่านี้ ควรตั้งค่าเริ่มต้นก่อนแล้วจึงสามารถนำค่าไปคำนวณต่อได้
- รีจิสเตอร์เฉพาะหน้าที่ ได้แก่ Stack Pointer (SP หรือ R13) Link Register (LR หรือ R14) และ Program Counter (PC หรือ R15) โปรแกรมเมอร์จะต้องบันทึกค่าของรีจิสเตอร์เหล่านี้เก็บไว้ โดยเฉพาะรีจิสเตอร์ LR ก่อนเรียกใช้ฟังค์ชันใดๆ และคืนค่า (Restore) ที่บันทึกเก็บไว้กลับไปให้รีจิสเตอร์ LR ก่อนจะเริ่มนกลับ

ผู้อ่านสามารถสำเนาซอฟต์แวร์สโคดูในการทดลองที่แล้วมาปรับแก้เป็นการทดลองนี้ได้

Appendix G. การทดลองที่ 7 การสร้างฟังก์ชันในโปรแกรมภาษาแอสเซมบลี

- ปรับแก้ Lab7_4.s ที่มีให้เป็น Lab7_5.s ดังต่อไปนี้

```

.data
@ Define all the strings and variables
.balign 4
get_val_1: .asciz "Number 1 :\n"

.balign 4
get_val_2: .asciz "Number 2 :\n"

@ printf and scanf use %d in decimal numbers
.balign 4
pattern: .asciz "%d"

@ variables: num_1 and num_2
.balign 4
num_1: .word 0

.balign 4
num_2: .word 0

@ Output format
.balign 4
output: .asciz "%d + %d = %d\n"

@ Back up the link register
.balign 4
lr_bu: .word 0
.balign 4
lr_bu_2: .word 0

.text
sum_vals:
    @ Save Link Register
    LDR R2, addr_lr_bu_2
    STR lr, [R2]    @ Mem[R2] <- LR
    B 266

    } ចានាបែងចាយនៅក្រោមនេះនឹងត្រូវបង្ហាញនៅក្នុង
        return function

```

Appendix G. การทดลองที่ 7 การสร้างฟังก์ชันในโปรแกรมภาษาแอลซีเมบลี

R_0 $R_0 + R_1$
 R_1 $R_0 + R_1$
 @ Sum values in R_0 and R_1 and return in R_0 $R_0 + R_1$ $| v_{-}lr$
ADD R_0, R_0, R_1 $| v_{-}lr$

R_0 $R_0 + R_1$
 R_1 $R_0 + R_1$
 @ Restore Link Register
LDR $lr, \text{addr_lr_bu_2}$ $\rightarrow \boxed{\text{addr_lr_bu_2}} | v_{-}lr$
LDR $lr, [lr]$ $\rightarrow \boxed{\text{addr_lr_bu_2}} | v_{-}lr$ $\downarrow lr = v_{-}lr$
BX lr $| v_{-}lr$

@ variable to back up Link Register

`addr_lr_bu_2: .word lr_bu_2`

@ Tell LIBC where main is

`.global main`

`main:`

@ Store Link Register
LDR $R_1, \text{addr_lr_bu}$
STR $lr, [R_1]$ $\rightarrow \text{Mem}[\text{addr_lr_bu}] \leftarrow LR$

@ Print out message to get 1st value

LDR $R_0, \text{addr_get_val_1}$
BL printf

@ Get num1 from user via keyboard

LDR $R_0, \text{addr_pattern}$
LDR $R_1, \text{addr_num_1}$
BL scanf

LDR $R_0, \text{addr_get_val_2}$
BL printf

@ Get num2 from user via keyboard

LDR $R_0, \text{addr_pattern}$
LDR $R_1, \text{addr_num_2}$

Appendix G. การทดลองที่ 7 การสร้างฟังก์ชันในโปรแกรมภาษาแอสเซมบลี

BL scanf

```
@ Pass by values entered to sum_vals  
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_vals
```

```
@ Keep returned value from sum_vals in R3  
MOV R3, R0
```

```
@ Pass the values to display  
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  
LDR lr, addr_lr_bu  
LDR lr, [lr]      @ LR <- Mem[addr_lr_bu]  
BX lr
```

```
@ Define pointer variables  
addr_get_val_1: .word get_val_1  
addr_get_val_2: .word get_val_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. ปรับแก้ makefile เพื่อแปลและรันโปรแกรม Lab7_6 แล้วสังเกตผลลัพธ์ที่ได้

3. ระบุชอร์สโค้ดใน Lab7_5.s ว่าตรงกับประโยชน์ภาษา C ต่อไปนี้

`int num1, num2`

4. ระบุชอร์สโค้ดใน Lab7_5.s ว่าตรงกับประโยชน์ภาษา C ต่อไปนี้

`sum = num1 + num2`

G.4 กิจกรรมท้ายการทดลอง

1. 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

2. จงพัฒนาโปรแกรมด้วยภาษา Assembly เพื่อรับตัวเลขจำนวน 2 ตัวจากผู้ใช้ผ่านทางคีย์บอร์ด เรียกว่า A และ B แล้วคำนวณด้วยคำสั่งภาษาแอสเซมบลี และแสดงผลลัพธ์ ตามตารางในข้อ 1

3. จงพัฒนาโปรแกรมด้วยภาษา C เพื่อรับตัวเลขจำนวน 2 ตัวจากผู้ใช้ผ่านทางคีย์บอร์ด เรียกว่า A และ B แล้วคำนวณหาค่า หرم หรือ หารร่วมมาก และแสดงผลลัพธ์ ตามตารางต่อไปนี้

Input	Output
5 2	1
18 6	6
49 42	7
81 18	9

4. จงพัฒนาโปรแกรมด้วยภาษา Assembly เพื่อรับตัวเลขจำนวน 2 ตัวจากผู้ใช้ผ่านทางคีย์บอร์ด เรียกว่า A และ B แล้วคำนวณหาค่า หرم หรือ หารร่วมมาก ด้วยคำสั่งภาษาแอสเซมบลีและแสดงผลลัพธ์ ตามตารางในข้อ 3

Appendix H

การทดลองที่ 8 การพัฒนาโปรแกรมภาษาแอลซ์ เชมบลีชันสูง

การพัฒนาโปรแกรมภาษาแอลซ์ เชมบลีชันสูง จะเน้นการพัฒนาร่วมกับภาษา C เพื่อเพิ่มศักยภาพของโปรแกรมภาษา C ให้ทำงานได้มีประสิทธิภาพยิ่งขึ้น โดยเฉพาะฟังค์ชันที่สำคัญและต้องเชื่อมต่อกับ hardware อย่างลึกซึ้ง และถ้ามีประสบการณ์การดีบักโปรแกรมภาษา C จะยิ่งทำให้ผู้อ่านเข้าใจการทดลองนี้ได้เพิ่มขึ้น ดังนั้น การทดลองมีวัตถุประสงค์เหล่านี้

- เพื่อฝึกการดีบักโปรแกรมภาษาแอลซ์โดยใช้โปรแกรม GDB แบบคอมมานด์ไลน์ (Command Line)
- เพื่อพัฒนาพัฒนาโปรแกรมแอลซ์โดยใช้ Stack Pointer
- เพื่อพัฒนาโปรแกรมภาษาแอลซ์ร่วมกับภาษา C

H.1 ดีบักเกอร์ GDB

ดีบักเกอร์เป็นโปรแกรมคอมพิวเตอร์ที่รันโปรแกรมที่กำลังพัฒนา เพื่อให้โปรแกรมเมอร์ตรวจสอบการทำงานได้ลึกซึ้งยิ่งขึ้น ทำให้โปรแกรมเมอร์สามารถเข้าใจการทำงานของโปรแกรมอย่างถ่องแท้ และหากโปรแกรมมีปัญหาหรือ บัก ที่บรรทัดไหน ตำแหน่งใด ดีบักเกอร์เป็นเครื่องมือที่จะช่วยแก้ปัญหานั้นได้ในที่สุด

GDB เป็นดีบักเกอร์มาตรฐานทำงานในระบบปฏิบัติการ Unix สามารถช่วยโปรแกรมเมอร์แก้ปัญหาของโปรแกรมที่พัฒนาจากภาษา C/C++ รวมถึงภาษาแอลซ์ของชิปยี่ห้อต่างๆ เช่น แอลซ์ เชมบลีชันของ ARM บนบอร์ด Pi3 นี้

ผู้อ่านสามารถย้อนกลับไปศึกษาการทดลองที่ 5 และ 6 ลักษณะ หัวข้อที่ F.2 เพื่อสังเกตรายละเอียดการสร้างโปรเจ็คท์ได้ว่า เราได้เลือกใช้ GDB เป็นดีบักเกอร์ ผู้อ่านสามารถเรียนรู้การดีบักโปรแกรมแอลซ์ เชมบลี พร้อมๆ กับทำความเข้าใจคำสั่งใน GDB ไปพร้อมๆ กัน ดังนี้

- เปิดโปรแกรม Terminal และย้ายไดเรกทอรีไปที่ `/home/pi/AssemblyLabs`

Appendix H. การทดลองที่ 8 การพัฒนาโปรแกรมภาษาแอสเซมบลีขั้นสูง

2. สร้างไดเรคทอรีใหม่ชื่อ **Lab8**

3. สร้างไฟล์ชื่อ **Lab8_1.s** ด้วยเทกซ์อิเดียตเตอร์ nano จากโปรแกรมต่อไปนี้

```
.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:
    MOV R7, #1
    SWI 0
```

4. สร้าง **makefile** และกรอกประযุคคำสั่งต่อไปนี้

```
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 เพื่อใช้ประกอบการทดลองถัดไป

7. ตั้งค่าเบรกพอยท์เพื่อยุดการรันโปรแกรมชั่วคราว และเปิดโอกาสให้โปรแกรมเมอร์สามารถตรวจสอบค่าของรีจิสเตอร์ต่างๆ ได้ โดยใช้คำสั่ง

(gdb) b 9

จะได้ผลตอบรับจาก GDB ดังนี้

Breakpoint 1, _continue_loop () at Lab8_1.s:10

8. รันโปรแกรม โดยพิมพ์คำสั่งต่อไปนี้ บันทึกและอธิบายผลลัพธ์

(gdb) run

9. โปรดสังเกตว่า (gdb) ปรากฏขึ้นแสดงว่าโปรแกรมหยุดที่เบรกพอยท์แล้ว พิมพ์คำสั่ง (gdb) info r เพื่อแสดงค่าภายในรีจิสเตอร์ต่างๆ ทั้งหมด และบันทึกค่าของรีจิสเตอร์เหล่านี้ r0, r1, r9, sp, pc, cpsr หลังรันโปรแกรม

(gdb) info r

			status (รูป)		
r0	0x0	0		r0	0x0 0
r1	0x1	1		r1	0x1 1
r2	0x7effefec	2130702316		r9	0x0 0
r3	0x10408	66568		sp	0x7efff2b8 0x7efff2b8
r4	0x10428	66600		pc	0x104e4 0x104e4
r5	0x0	0		cpsr	0x80000010 - 2147483632
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			

Appendix H. การทดลองที่ 8 การพัฒนาโปรแกรมภาษา C และเชมบลีชันสูง

จงตอบคำถามต่อไปนี้ประกอบความเข้าใจ

- อธิบายรายงานบนหน้าจอว่าคอลัมน์แต่ละคอลัมน์มีความหมายอย่างไร และแตกต่างกันหน้าจอของผู้อ่านอย่างไร
- เหตุใดเลขในคอลัมน์ขวัญจะมีค่าติดลบ

r0	0x0	1
r9	0x0	0
sp	0x7efff2b8	0x7efff2b8
pc	0x103e4	0x103e4
cpsr	0x80000000	-2147483632

10. พิมพ์คำสั่ง (gdb) c เพื่อรันโปรแกรมต่อไปจนกว่าจะวนรอบกลับมาที่เบรกพอยท์ที่ตั้งไว้

11. พิมพ์คำสั่ง (gdb) info r เพื่อแสดงค่าภายในรีจิสเตอร์ต่างๆ ทั้งหมด และบันทึกค่าของรีจิสเตอร์เหล่านี้ r0, r1, r9, sp, pc, cpsr เพื่อสังเกตการเปลี่ยนแปลง

12. เริ่มต้นการทดลองโดยพิมพ์คำสั่งต่อไปนี้เพื่อหาว่า เลเบล _loop ตรงกับหน่วยความจำตำแหน่งใด

(gdb) disassemble _loop

Dump of assembler code for function _loop:
0x000103dc <+0> add r0, r0, r1
End of assembler dump

บันทึกผลที่ได้โดย หมายเลขอ้ายสุด คือ แอดเดรสในหน่วยความจำ ที่คำสั่งนั้นบรรจุอยู่ หมายเลขอ้ายตำแหน่งถัดมา คือ จำนวนไปทั้งนับจากจุดเริ่มต้นของชื่อเลเบลนั้น แล้วตรวจสอบว่าเลเบล main อยู่ห่างจากตำแหน่งเริ่มต้นของโปรแกรมกี่เบท

Dump of assembler code for function _loop:

0x00010414 <+0>: add r0, r0, r1

End of assembler dump.

13. พิมพ์คำสั่ง (gdb) c เพื่อรันโปรแกรมต่อไปจนกว่าจะวนรอบกลับมาที่เบรกพอยท์ที่ตั้งไว้อีกรอบ

14. คำสั่ง x/ [count] [format] [address] แสดงค่าในหน่วยความจำ ณ ตำแหน่ง address เป็นต้นไป เป็นจำนวน /count ตาม format ที่ต้องการ ยกตัวอย่างเช่น x/10i main คือ แสดงค่าในหน่วยความจำ ณ ตำแหน่งเลเบล main จำนวน 10 ค่าตามรูปแบบ instruction ดังตัวอย่างต่อไปนี้

(gdb) x/10i main

0x10408 <main>: mov r0, #0

0x1040c <main+4>: mov r1, #1

0x10410 <main+8>: b 0x10418 <_continue_loop>

0x10414 <_loop>: add r0, r0, r1

0x10418 <_continue_loop>: cmp r0, #9

=> 0x1041c <_continue_loop+4>: ble 0x10414 <_loop>

0x10420 <end>: mov r7, #1

0x10424 <end+4>: svc 0x00000000

ก. จังหวะนี้ต่อไป

จุดที่จะนำไปต่อจากนั้น

```
0x10428 <__libc_csu_init>: push {r4, r5, r6, r7, r8, r9, r10, lr}
0x1042c <__libc_csu_init+4>: mov r7, r0
```

จงตอบคำถามต่อไปนี้

- เติมตัวอักษรที่เว้นว่างไว้จากหน้าจอของผู้อ่านในเครื่องหมาย <_> สองตำแหน่ง
- อธิบายว่า หมายเลขที่มาแทนที่ <_> ได้อย่างไร
- โปรดสังเกตและอธิบายว่าเครื่องหมายลูกศร => ด้านซ้ายสุดหน้าบรรทัดคำสั่ง หมายถึงอะไร

- s[tep] i ระหว่างที่เบรกการรันโปรแกรม ผู้ใช้สามารถสั่งให้โปรแกรมทำงานต่อเพียง 1 คำสั่งเพื่อตรวจสอบ
- n[ext] i ทำงานคล้ายคำสั่ง step i แต่ถ้าคำสั่งต่อไปที่จะทำงานเป็นการเรียกฟังค์ชัน คำสั่งนี้เรียกใช้ฟังค์ชันนั้นจนสำเร็จ แล้วจึงกลับมาให้ผู้ใช้ตรวจสอบ
- i[info] b[reak] เพื่อแสดงรายการเบรกพอยท์ทั้งหมดที่ตั้งไว้ก่อนหน้า

```
(gdb) i b
Num      Type            Disp Enb Address      What
1        breakpoint      keep y    0x0001041c Lab8_1.s:_
breakpoint already hit 3 times
```

ผู้อ่านจะต้องทำความเข้าใจรายงานที่ได้บนหน้าจอ โดยเฉพาะคอลัมน์ Address และ What โดยเติมตัวอักษรลงในช่องว่าง _ ทั้งสองช่อง

- คำสั่ง d[elete] b[reakpoints] number ลบการตั้งเบรกพอยท์ที่บรรทัด number ที่ตั้งไว้ก่อนหน้า หากผู้อ่านต้องการลบเบรกพอยท์ทั้งหมดพร้อมกันโดยพิมพ์

```
(gdb)d
Delete all breakpoints? (y or n)
```

แล้วตอบ y เพื่อยืนยัน

- พิมพ์คำสั่ง (gdb) c เพื่อรันโปรแกรมต่อไปจนเสร็จสิ้นจะได้ผลลัพธ์ต่อไปนี้

```
(gdb) c
Continuing.
[Inferior 1 (process 1688) exited with code 012]
```

20. พิมพ์คำสั่งต่อไปนี้เพื่อออกจากโปรแกรม GDB

(gdb) q

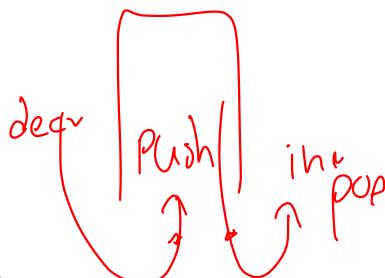
H.2 การใช้งานสแต็คพอยท์เตอร์ (Stack Pointer)

ตำแหน่งของหน่วยความจำบริเวณที่เรียกว่า สแต็คเซกเม้นท์ (Stack Segment) จากรูปที่ 3.12 สแต็คเซกเม้นท์ตั้งในบริเวณแอดเดรสสูง (High Address) หน้าที่เก็บข้อมูลของตัวแปรชนิดโลคอล (Local Variable) รับค่าพารามิเตอร์ระหว่างฟังค์ชัน กรณีที่มีจำนวนเกิน 4 ตัว พักเก็บค่าของรีจิสเตอร์ที่สำคัญ เช่น LR เป็นต้น

สแต็คพอยท์เตอร์ คือ รีจิสเตอร์ R13 มีหน้าที่เก็บแอดเดรสตำแหน่งบนสุดของสแต็ค (Top of Stack: TOS) ตำแหน่งบนสุดของสแต็คจะเป็นตำแหน่งที่เกิดการ PUSH (Store) และ POP (Load) ข้อมูลเข้าและออกจากสแต็คตามลำดับ โปรแกรมเมอร์สามารถจินตนาการได้ว่า **สแต็ค** คือ กองสิ่งของที่วางซ้อนกันโดยโปรแกรมเมอร์ สามารถหยิบสิ่งของออกหรือวางของที่ซึ้งบนสุดเท่านั้น สแต็คพอยท์เตอร์ คือ หมายเลขอันดับของซึ่งตำแหน่งจะเพิ่มขึ้นหรือลดลง เมื่อโปรแกรมเมอร์ใช้คำสั่ง PUSH/POP ตามลำดับ

คำสั่ง STM (Store Multiple) ทำหน้าที่ PUSH ข้อมูลลงบนสแต็ค คำสั่ง LDM (Load Multiple) ทำหน้าที่ POP ข้อมูลออกจากสแต็ค ตำแหน่งหรือแอดเดรสของสแต็คพอยท์เตอร์ สามารถเปลี่ยนแปลงได้สองทิศทาง คือ เพิ่มขึ้น (Ascending)/ลดลง (Descending). ดังนั้น คำสั่ง STM/LDM สามารถสมกับทิศทางได้ทั้งสิ้น 4 แบบ และก่อนหลัง รวมเป็น 8 แบบ ดังนี้

- LDMIA/STMIA : IA = Increment After
- LDMIB/STMIB : IB = Increment Before
- LDMDA/STMDA : DA = Decrement After
- LDMDB/STMDB : DB = Decrement Before



Increment/Decrement หมายถึง การเพิ่ม/ลดค่าของรีจิสเตอร์ที่เกี่ยวข้องโดยมักใช้งานร่วมกับ รีจิสเตอร์ SP **after/before** หมายถึง ก่อน/หลังการปฏิบัติตามคำสั่งนั้น ยกตัวอย่าง การใช้งานคำสั่งเพื่อ PUSH รีจิสเตอร์ลงในสแต็คโดยใช้ STMDB และ POP ค่าจากสแต็คจะคูณกับคำสั่ง LDMIA ความหมาย คือ สแต็คจะเติบโตในทิศทางที่แอดเดรสลดลง (Decrement Before) ซึ่งเป็นที่นิยมและตรงกับรูปการจัดวางหน่วยความจำเสมือนในรูปที่ 3.12 ผู้อ่านสามารถทบทวนเรื่องนี้ในหัวข้อที่ 5.2

1. สร้างไฟล์ Lab8_2.s ตามโค้ดต่อไปนี้ ผู้อ่านสามารถข้ามประযุคคอมเม้นท์ได้ เมื่อทำการเข้าใจแต่ละคำสั่งแล้ว

```
.global main
main:
    MOV R1, #1
```

MOV R2, #2

© Push (store) R1 onto stack, then subtract SP by 4 bytes
 © The ! (Write-Back symbol) updates the register SP

STR R1, [sp, #-4]!
 STR R2, [sp, #-4]!

© Pop (load) the value and add 4 to SP

LDR R0, [sp], #+4
 LDR R0, [sp], #+4

end:

MOV R7, #1
 breqp10 SWI 0

R0	0x1	1
R1	0x1	1
R2	0x2	2
R3	0x7	7
R4	0xffffffff	0xffffffff
R5	0x102e	0x102e
CPSR	0x60000010	16106129

2. รันโปรแกรม บันทึกและอธิบายผลลัพธ์

3. สร้างไฟล์ Lab8_3.s ตามโค้ดต่อไปนี้ ผู้อ่านสามารถข้ามประยุคคอมเม้นท์ได้ เมื่อทำความเข้าใจแต่ละคำสั่งแล้ว

```
.global main
main:
    MOV R1, #0
    MOV R2, #1
    MOV R4, #2
    MOV R5, #3
```

} ก่อนหน้า

© SP is subtracted by 8 bytes to save R4 and R5, respectively.

© The ! (Write-Back symbol) updates SP.

STMDB SP!, {R4, R5}

© Pop (load) the values and increment SP after that

LDMIA SP!, {R1, R2} $R1 = R4$, $R2 = R5$

ADD R0, R1, #0 $R0 = R4$

ADD R0, R0, R2 $R0 = R0 + R1$
 $= R4 + R5$

end:

MOV R7, #1 $R0 = R4$
 $= 2 + 3$
 $= 5$

R6	5
R7	2
R2	3
R4	2
R5	3

Appendix H. การทดลองที่ 8 การพัฒนาโปรแกรมภาษาแอสเซมบลีขั้นสูง

SWI 0

4. รันโปรแกรม บันทึกและอธิบายผลลัพธ์

H.3 การพัฒนาโปรแกรมภาษาแอสเซมบลีร่วมกับภาษา C

การพัฒนาโปรแกรมด้วยภาษา C สามารถเชื่อมต่อกับฮาร์ดแวร์ และทำงานได้รวดเร็วใกล้เคียง กับภาษาแอสเซมบลี แต่การเสริมการทำงานของโปรแกรมภาษา C ด้วยภาษาแอสเซมบลียังมีความจำเป็น โดยเฉพาะ โปรแกรมที่เรียกว่า **ดิไวซ์ไดรเวอร์** (Device Driver) ซึ่งเป็นโปรแกรมขนาดเล็กที่เชื่อมต่อกับฮาร์ดแวร์ที่ต้องการความรวดเร็วและประสิทธิภาพสูง การทดลองนี้จะแสดงให้ผู้อ่านเห็นการเขียนต่อฟังค์ชันภาษาแอสเซมบลีกับภาษา C อย่างง่าย

1. เปิดโปรแกรม Code::Blocks
2. สร้างโปรเจ็คท์ Lab8_4 ภายใต้โฟลเดอร์ /home/pi/Assembly/Lab8
3. สร้างไฟล์ชื่อ add_s.s และป้อนคำสั่งต่อไปนี้

```
.global add_s
add_s:
    ADD R0, R0, R1
    BX LR
```

4. เพิ่มไฟล์ add_s.s ในโปรเจ็คท์ Lab8_4 ที่สร้างไว้ก่อนหน้า
5. สร้างไฟล์ชื่อ main.c และป้อนคำสั่งต่อไปนี้

```
#include <stdio.h>
int main(){
    int a = 16;
    int b = 4;
    int i = add_s(a, b);
    printf("%d + %d = %d \n", a, b, i);
    return 0;
}
```

6. ทำการ Build และแก้ไขหากมีข้อผิดพลาดจนสำเร็จ
7. Run และสังเกตการเปลี่ยนแปลง

16
4

$$16 + 4 = 20$$

8. อธิบายว่าเหตุใดการทำงานจึงถูกต้อง ฟังค์ชัน `add_R` รับข้อมูลทางรีจิสเตอร์ตัวไหนบ้างและรีเทิร์นค่าที่คำนวนเสร็จแล้วทางรีจิสเตอร์อะไร

សປຕະ R0 R1 ទັນສາມາດ

ในทางปฏิบัติ การบวกเลขในภาษา C สามารถทำได้โดยใช้เครื่องหมาย + โดยตรง และทำงานได้รวดเร็วกว่า การทดลองตัวอย่างนี้เป็นการนำเสนอว่าผู้อ่านสามารถเขียนโปรแกรมอย่างไรที่จะบรรลุวัตถุประสงค์เท่านั้น ฟังค์ชันภาษาแอสเซมบลีที่จะลงค์เข้ากับโปรแกรมหลักที่เป็นภาษา C ควรจะมีอรรถประโยชน์มากกว่านี้ และเชื่อมโยงกับฮาร์ดแวร์โดยตรงได้กิ่งคำสั่งในภาษา C

H.4 กิจกรรมท้ายการทดลอง

- กิจกรรม
1. จะใช้โปรแกรม GDB เพื่อแสดงรายละเอียดของสแต็คระหว่างที่รันโปรแกรม Lab8_2 และบอกลำดับการ PUSH และการ POP ที่เกิดขึ้นภายในโปรแกรมจากแต่ละคำสั่ง
 2. จะใช้โปรแกรม GDB เพื่อแสดงรายละเอียดของสแต็คระหว่างที่รันโปรแกรม Lab8_3 และบอกลำดับการ PUSH และการ POP ที่เกิดขึ้นภายในโปรแกรมจากแต่ละคำสั่ง
 3. จะนำโปรแกรมภาษาแอสเซมบลีสำหรับคำนวนค่า Modulus ในการทดลองที่ 7 มาเรียกใช้ผ่านโปรแกรมภาษา C
 4. จะนำโปรแกรมภาษาแอสเซมบลีสำหรับคำนวนค่า GCD ในการทดลองที่ 7 มาเรียกใช้ผ่านโปรแกรมภาษา C
 5. จะดีบักโปรแกรมภาษา C บนโปรแกรม Codeblocks ที่พัฒนาในข้อ 2 และ 3 เพื่อบันทึกการเปลี่ยนแปลงของ PC ก่อน ระหว่าง และหลังเรียกใช้ฟังค์ชันภาษา Assembly ว่าเปลี่ยนแปลงอย่างไร และตรงกับทฤษฎีที่เรียนหรือไม่ อย่างไร

กิจกรรม ① int main(){
 int a=-5, b=3;
 int i = get_mod(a, b);
 printf("%d %.1f %.1f = %.1f", a, b, i);
 return 0}

R0 1
 R1 2130703300
 PC 0x104C4 (main+28)
 R0 12
 R1 0
 PC 0x104F8 (main+64)

```
.global get_mod
get_mod:
    CMP R0, #0
    BGE else
    loop:
        ADD R0, R0, R1
        CMP R0, #0
        BLT loop
    else:
        loop2:
            SUB R0, R0, R1
            CMP R0, #0
            BGT loop2
            CMP R0, #0
            BEQ exit
        else2:
            ADD R0, R0, R1
    exit:
        BX lr
```

R0 1
 R1 3
 PC 0x104E4 (main+44)

Q84) int main(){

int a=15, b=30;

int i = get_gcd(a,b);

printf("%d %d %d = %d", a, b, i);
return 0;

* ឧណុវត្តន៍សារមិនល្អ

```
.global get_gcd
get_gcd:
    CMP R0, R1
    BLT swap
    BGE process
swap:
    mov r3, r0
    mov r0, r1
    mov r1, r3
process:
    mov r2, #1
    find_multi:
        mul r3, r1, r2
        cmp r3, r0
        bgt subr0
        add r2, r2, #1
        b find_multi
subr0:
    sub r2, r2, #1
    mul r3, r1, r2
    sub r0, r0, r3
    b is_endr0
is_endr0:
    cmp r0, #0
    BEQ exit
    b swap
exit:
    mov r0, r1
    BX lr
```

r0 1
r1 213070330

r2 213070330

r3 0

pc 0x104f8<main+88>

r0 14

r1 0

r2 346339584

r3 346339584

pc 0x1051c<main+64>

r0 15
r1 75
r2 2
r3 30

pc 0x10508<main+44>

Appendix I

การทดลองที่ 9 การศึกษาและปรับแก้อินพุตและเอาท์พุตต่างๆ

การทดลองในภาคผนวกนี้ จะช่วยอธิบายเนื้อหาในบทที่ 2 ซึ่งเกี่ยวข้องกับอุปกรณ์อินพุต/เอาท์พุตที่หลากหลายบนเครื่องคอมพิวเตอร์ตั้งโต๊ะ โดยมีวัตถุประสงค์เหล่านี้

- เพื่อให้เข้าใจการปรับแก้อุปกรณ์อินพุตและเอาท์พุตนิดต่างๆ บนระบบปฏิบัติการ Raspbian
- เพื่อให้เข้าใจความแตกต่างระหว่างอุปกรณ์อินพุตและเอาท์พุตนิดต่างๆ บนบอร์ด Pi3
- เพื่อให้สามารถอ่านข้อมูลความแสดงรายละเอียดของอุปกรณ์อินพุตและเอาท์พุตนิดต่างๆ

หลักการและพื้นฐานความเข้าใจจะช่วยแนะนำทางให้ผู้อ่านสามารถศึกษาค้นคว้า อินพุต/เอาท์พุต ในชิปและบอร์ดได้เพิ่มเติม รวมไปถึงบันโหรส์เคลื่อนที่ แท็บเล็ตคอมพิวเตอร์ และอุปกรณ์อินเทอร์เน็ตสตรอร์สิ่ง (Internet of Things)

I.1 จอแสดงผลผ่านพอร์ต HDMI

หน่วยความจำสำหรับจอแสดงผลหรือ GPU (Graphic Processing Unit) ถูกแบ่งพื้นที่ออกจากหน่วยความจำ DRAM บนบอร์ด เพื่อใช้งานร่วมกันทำให้ประยุกต์ตันทุน แต่มีข้อเสียในด้านประสิทธิภาพลดลงบ้าง เมื่อผู้ใช้งานต้องการภาพที่มี อัตราการเปลี่ยนแปลง (Refresh Rate) หรืออัตราเฟรมเรท (Frame Rate) สูง เช่น ภาพเคลื่อนไหว เกมส์ 3 มิติ

I.1.1 การปรับแก้ขนาดหน่วยความจำของ GPU

ความละเอียดของจอแสดงผลขึ้นตรงกับขนาดของหน่วยความจำของ GPU ผู้อ่านสามารถปรับแก้ขนาดหน่วยความจำของ GPU ดังนี้

menu->Preferences->Raspberry Pi Configuration->Set Resolution->Performance

Appendix I. การทดลองที่ 9 การศึกษาและปรับแก้ในพุทธและเอาท์พุตต่างๆ

โดยหน้าต่างที่ปรากฏขึ้นมีลักษณะดังนี้ ผู้ใช้สามารถกำหนดขนาดที่ต้องการโดยขั้นต่ำคือ 64 MB เพื่อให้ระบบสามารถแสดงผลได้ หากผู้ใช้กำหนดสูงเกินไป จะทำให้บอร์ดมีหน่วยความจำไม่เพียงพอ

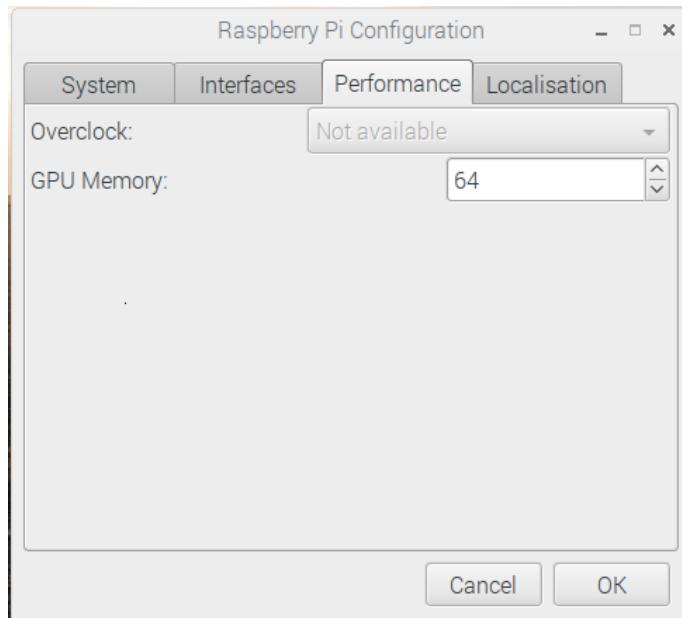


Figure I.1: หน้าต่างกำหนดขนาดหน่วยความจำสำหรับ GPU ที่ 64 เมกะไบต์

I.1.2 การปรับแก้ความละเอียดของจอแสดงผล

เมื่อขนาดหน่วยความจำของ GPU มีเพียงพอ ผู้ใช้สามารถปรับเพิ่มหรือลดความละเอียดของจอแสดงผลได้โดยกดปุ่มบนเมนูดังนี้

menu->Preferences->Raspberry Pi Configuration->Set Resolution

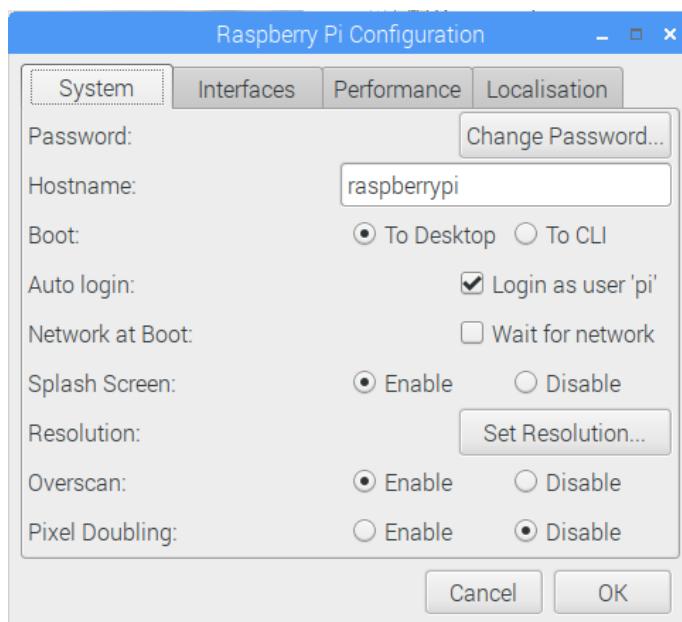


Figure I.2: หน้าต่าง Raspberry Pi Configuration แท็บ System สำหรับกำหนดความละเอียดหน้าจอแสดงผล (Resolution)

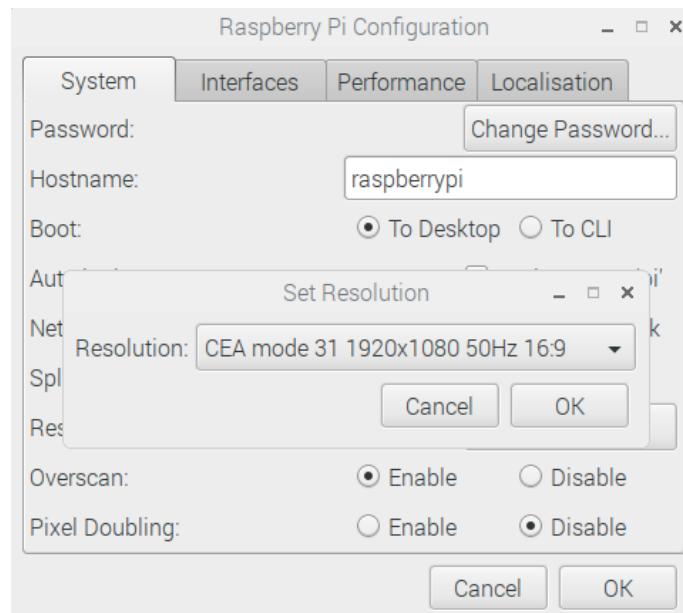


Figure I.3: หน้าต่าง Set Resolution สำหรับกำหนดความละเอียดหน้าจอที่ต้องการ

กดปุ่ม Set Resolution สำหรับกำหนดความละเอียดหน้าจอที่ต้องการ ในรูปที่ ผู้เขียนต้องการแสดงผลที่ความละเอียด CEA Mode 31 1920x1080 50Hz 16:9 หลังจากนั้นกดปุ่ม OK หน้าต่าง Reboot needed จะปรากฏขึ้น

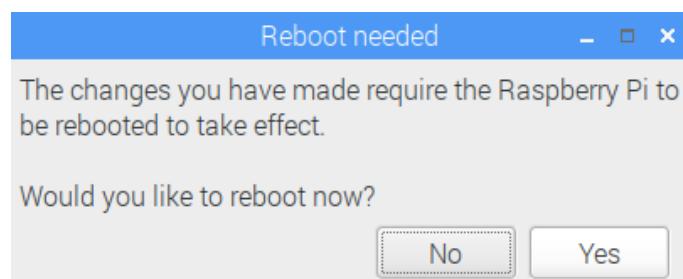


Figure I.4: หน้าต่าง Reboot needed กดปุ่ม Yes เมื่อต้องการรีบูต ณ เวลาใดก็ได้

I.2 ระบบเสียง

อุปกรณ์ด้านระบบเสียงที่ติดตั้งมาบนบอร์ด Pi3 จากโรงงาน ผู้ใช้สามารถเพิ่มเติมได้ผ่านพอร์ท USB และปรับแต่งระดับเสียงได้ เช่นกัน

I.2.1 รายชื่ออุปกรณ์ด้านระบบเสียง

ระบบเสียงในระบบปฏิบัติการ Linux ควบคุมการทำงานของเสียงผ่านระบบ ALSA (Advanced Linux Sound Architecture), ซึ่งจัดเตรียมมาให้โดยเดフォลต์ (Device Driver) สำหรับเสียงให้กับเครื่องเนต และอุปกรณ์ที่เกี่ยวข้องกับเสียงผ่านพอร์ท USB เช่น ไมโครโฟน, หูฟังพร้อมไมโครโฟน, เว็บแคม เป็นต้น ผู้อ่านสามารถแสดงรายชื่อไฟล์หรือโฟลเดอร์ที่เกี่ยวข้องกับระบบเสียงดังนี้

```
$ ls -l /proc/asound
```

```
lrwxrwxrwx 1 root root 5 Mar 26 20:59 ALSA -> card0
dr-xr-xr-x 4 root root 0 Mar 26 20:59 card0
-r--r--r-- 1 root root 0 Mar 26 20:59 cards
-r--r--r-- 1 root root 0 Mar 26 20:59 devices
-r--r--r-- 1 root root 0 Mar 26 20:59 modules
-r--r--r-- 1 root root 0 Mar 26 20:59 pcm
dr-xr-xr-x 2 root root 0 Mar 26 20:59 seq
-r--r--r-- 1 root root 0 Mar 26 20:59 timers
-r--r--r-- 1 root root 0 Mar 26 20:59 version
```

ผลลัพธ์คือ รายชื่ออุปกรณ์ที่เกี่ยวข้องกับเสียง โดยเฉพาะ ALSA ซึ่งได้แสดงไปก่อนหน้านี้ ผู้อ่านจะสังเกตได้ว่าโฟลเดอร์ /proc/asound/pcm จะเชื่อมโยงกับเนื้อหาในหัวข้อที่ 2.4 จะสังเกตเห็นว่ามีไฟล์เดอร์ชื่อ card0 อยู่สองตำแหน่งคือ ในແລງແຮກ และແຄວที่มีชื่อ ALSA -> card0 สัญลักษณ์ -> เเรียก ว่า Symbolic Link หมายความว่าชื่อ ALSA คือ card0 ผู้อ่านสามารถทดสอบโดยพิมพ์คำสั่งต่อไปนี้

```
$ cat /proc/asound/ALSA
$ cat /proc/asound/card0
```

จะได้ผลลัพธ์เดียวกัน

ผู้ใช้พิมพ์คำสั่งนี้ในโปรแกรม Terminal

```
$ cat /proc/asound/cards
```

โดยคำสั่ง cat ซึ่งได้อธิบายแล้วในการทดลองที่ 4 ภาคผนวก D สามารถอ่านไฟล์และแสดงข้อมูลภายในไฟล์ผ่านทางหน้าจอแสดงผล ผลลัพธ์ตัวอย่างอาจแตกต่างกันไปตามฮาร์ดแวร์ที่ใช้งาน ดังนี้

```
0 [ALSA] : bcm2835_alsa - bcm2835 ALSA
          bcm2835 ALSA
```

ผลลัพธ์ได้จากบอร์ด Pi3 ที่ใช้ชิปเซ็ตของ BCM2835 และ ไดรเวอร์เดียวกันกับ BCM2837 โดย หมายเลข 0 คือ หมายเลขของระบบเสียงที่ติดตั้งใช้งานเพียงระบบเดียว และตรงกับชื่อ card0

I.2.2 การควบคุมระดับเสียง

ผู้อ่านสามารถควบคุมระดับความดังของเสียงทั้งด้านอินพุตและเอาท์พุต โดยพิมพ์คำสั่งนี้

```
$ alsamixer
```

หน้าต่างต่อไปนี้จะปรากฏขึ้น ผู้อ่านสามารถกดปุ่มลูกศรขึ้น/ลง เพื่อเพิ่ม/ลด ระดับความดังได้

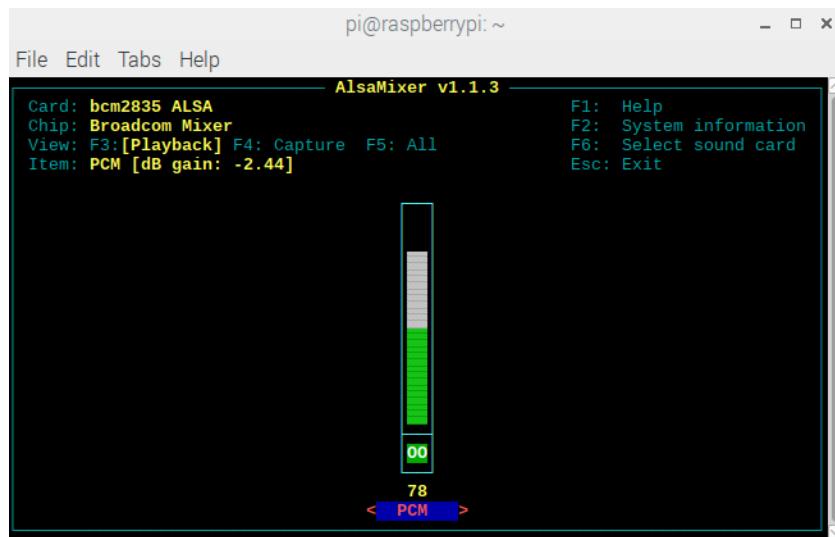


Figure I.5: โปรแกรม ALSA Mixer สำหรับควบคุมระดับเสียงบนบอร์ด Pi3

หมายเหตุ ผู้อ่านสามารถติดตั้งอุปกรณ์เสียงผ่านพอร์ท USB และใช้คำสั่งเหล่านี้เพื่อตรวจสอบและควบคุมการทำงาน

I.3 พอร์ทเชื่อมต่ออุปกรณ์ USB

I.3.1 รายชื่ออุปกรณ์กับพอร์ท USB

ในการทดลองนี้ ขอผู้อ่านให้ดึงหัวเชื่อมต่อ USB ของมาส์ที่เชื่อมต่ออยู่ออก และพิมพ์คำสั่งนี้ในหน้าต่าง Terminal

```
$ lsusb
```

เพื่อแสดงรายชื่ออุปกรณ์ USB ที่เชื่อมต่ออยู่ทั้งหมดในบอร์ด ดังตัวอย่างต่อไปนี้

```
Bus 001 Device 005: ID 413c:2003 Dell Computer Corp. Keyboard
```

```
Bus 001 Device 003: ID 0424:ec00 Standard Microsystems Corp.  
SMSC9512/9514 Fast Ethernet Adapter
```

```
Bus 001 Device 002: ID 0424:9514 Standard Microsystems Corp. SMC9514 Hub
```

```
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
```

ผู้อ่านจะเห็นรายชื่ออุปกรณ์ที่เชื่อมต่อกับพอร์ท USB เรียงลำดับย้อนกลับ จาก Device 005 - Device 001 โดย

- Device 005 คือ คีย์บอร์ดมีหมายเลข ID = 413c:2003 ผลิตโดย บริษัท Dell Computer Corp. ซึ่งคีย์บอร์ดของผู้อ่านจะแตกต่างจากที่ผู้เขียนบ้างก็ไม่ใช่ประเด็นสำคัญ
- Device 003 คือ วงจร Ethernet สำหรับเชื่อมต่อเครือข่ายชนิดสาย มีหมายเลข ID = 0424:ec00 ผลิตโดย บริษัท Standard Microsystems Corp. รุ่น SMSC9512/9514
- Device 002 คือ วงจร USB Hub สำหรับเชื่อมต่อพอร์ท USB เพิ่มเติม มีหมายเลข ID = 0424:9514 ผลิตโดย บริษัท Standard Microsystems Corp. รุ่น SMC9514
- Device 001 คือ วงจร Root Hub เป็นวงจรภายในชิป BCM2837 สำหรับเชื่อมต่อพอร์ท USB เพิ่มเติม มีหมายเลข ID = 1d6b:0002

ผู้อ่านต้องเสียบมาส์กับพอร์ท USB ใหม่อีกครั้ง และแสดงรายชื่ออุปกรณ์ USB ด้วยคำสั่ง lsusb เช่นเดิม โปรดสังเกตการเปลี่ยนแปลง

```
Bus 001 Device 005: ID 413c:2003 Dell Computer Corp. Keyboard
```

```
Bus 001 Device 006: ID 046d:c077 Logitech, Inc. M105 Optical Mouse
```

```
Bus 001 Device 003: ID 0424:ec00 Standard Microsystems Corp.  
SMSC9512/9514 Fast Ethernet Adapter
```

```
Bus 001 Device 002: ID 0424:9514 Standard Microsystems Corp. SMC9514 Hub
```

```
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
```

รายการที่เพิ่มขึ้น คือ

- Device 006 คือ เม้าส์เมาท์ ID = 046d:c077 ผลิตโดย บริษัท Logitech, Inc. รุ่น M105

ซึ่งลำดับที่ตั้งแต่ Device 003 เป็นต้นไปอาจเปลี่ยนแปลงตามรายการและลำดับที่อุปกรณ์ของผู้อ่านทำการเชื่อมต่อกับบอร์ด Pi3 และผู้ผลิตมาส์ของผู้อ่านจะแตกต่างจากที่ผู้เขียนบังกอกไม่ใช่ประเด็นสำคัญ

I.3.2 รายละเอียดการเชื่อมต่ออุปกรณ์กับพอร์ท USB

สำหรับการเชื่อมต่อไป คือ **dmesg** สามารถแสดงรายการทำงาน หรือ Log ของระบบปฏิบัติการว่าตั้งแต่เริ่มเปิดเครื่อง โดยคำว่า **dmesg** ย่อมาจากคำสั่ง “display message or display driver” ซึ่งเครื่องเนลได้บันทึกไว้ในบ퍼เฟอร์ชนิดวงแหวน (Ring Buffer) ซึ่งข้อมูลตอนต้นจะถูกเขียนทับเมื่อบ퍼เฟอร์เต็ม

```
$ dmesg
```

```
[0.000000] Booting Linux on physical CPU 0x0
[0.000000] Linux version 4.14.71-v7+ (dc4@dc4-XPS13-9333)
          (gcc version 4.9.3 (crosstool-NG crosstool-ng-1.22.0-88-g8460611))
          #1145 SMP Fri Sep 21 15:38:35 BST 2018
[0.000000] CPU: ARMv7 Processor [410fd034] revision 4 (ARMv7), cr=10c5383d
[0.000000] CPU: div instructions available: patching division code
[0.000000] CPU: PIPT / VIPT nonaliasing data cache,
          VIPT aliasing instruction cache
[0.000000] OF: fdt: Machine model: Raspberry Pi 3 Model B Rev 1.2
[0.000000] Memory policy: Data cache writealloc
[0.000000] cma: Reserved 8 MiB at 0x3ac00000
[0.000000] On node 0 totalpages: 242688
...
[0.000000] Memory: 940232K/970752K available (7168K kernel code, 576K rwdta,
          2076K rodata, 1024K init, 698K bss, 22328K reserved,
          8192K cma-reserved)
[0.000000] Virtual kernel memory layout:
          vector   : 0xfffff0000 - 0xfffff1000  (    4 kB)
          fixmap   : 0xfffc00000 - 0xfffff00000  (3072 kB)
          vmalloc  : 0xbb800000 - 0xff800000  (1088 MB)
          lowmem   : 0x80000000 - 0xbb400000  ( 948 MB)
          modules  : 0x7f000000 - 0x80000000  ( 16 MB)
          .bss    : 0x80c97f10 - 0x80d468b0  ( 699 kB)
          .data   : 0x80c00000 - 0x80c9017c  ( 577 kB)
```

Appendix I. การทดลองที่ 9 การศึกษาและปรับแก้อินพุตและเอาท์พุตต่างๆ

```
.init : 0x80b00000 - 0x80c00000 (1024 kB)
.text : 0x80008000 - 0x80800000 (8160 kB)
```

...

ผู้เขียนสามารถอธิบายผลลัพธ์ได้ดังต่อไปนี้ โดยเรียงลำดับตามเหตุการณ์ ซึ่งมีสัญลักษณ์ [xxxx.yyyyyy] แสดงลำดับที่เกิดขึ้นตามเวลา โดย xxxx คือเลขวินาทีตั้งแต่เครื่องเนลเริ่มทำงาน และ yyyyyy คือเศษวินาที ข้อความที่แสดงเป็น 0.000000 เนื่องจากเครื่องเนลอยู่ระหว่างการเริ่มต้น

- เริ่มต้นการบูรณาภิปัติการด้วยซีพียูคอร์ฟลายเลข 0
- แสดงรายละเอียดหมายเลขเวอร์ชันของลีนุกซ์
- แสดงรายละเอียดของ CPU ซึ่งใช้คำสั่งภาษาแอสเซมบลีเวอร์ชัน 7 (ARMv7)
- แสดงผลการตรวจจับว่าซีพียูรองรับคำสั่ง DIV
 - รายงานว่า แคชข้อมูล ทำงานแบบ nonaliasing PIPT (Physically indexed, physically tagged) หรือ VIPT (Virtually indexed, physically tagged) อย่างใดอย่างหนึ่ง และแคชคำสั่ง ทำงานแบบ aliasing VIPT แคชข้อมูลไม่สามารถแชร์ข้ามไฟรเซสได้เนื่องจากทำงานแบบ nonaliasing ในขณะที่แคชคำสั่งสามารถแชร์ข้ามไฟรเซสได้ เนื่องจากทำงานแบบ aliasing **ข้อมูลเพิ่มเติม**
- แสดงผลการตรวจจับว่าเป็นบอร์ด Raspberry Pi 3 Model B Rev 1.2
- การทำงานของแคชข้อมูลเป็นชนิด writealloc ย่อมาจาก Write Allocation ซึ่งซีพียูจะเขียนข้อมูลทั้งในแคชก่อน เมื่อแคชจะต้องถูกย้ายออกจึงค่อยเขียนในหน่วยความจำหลักภายหลัง **ข้อมูลเพิ่มเติม**
- cma (Contiguous Memory Allocator) สำหรับขบวนการ DMA เริ่มต้นที่แอดเดรส 0x3ac00000 ขนาด 8 เมกะไบต์
- ...
- พื้นที่การจัดวางหน่วยความจำเสมือนของเครื่องเนล (Virtual kernel memory layout) ผู้เขียนได้ทำการจัดเรียงใหม่ตามหมายเลขแอดเดรสที่ตำแหน่งมาก ไล่ลงมาจนถึงหมายเลขน้อย เพื่อให้ผู้อ่านมองเห็นภาพและเข้าใจง่ายขึ้น โดยแบ่งเป็นพื้นที่สำคัญๆ ตามลำดับดังนี้
 - จัดเก็บเวคเตอร์สำหรับการขัดจังหวะ (Interrupt Vector) ขนาด 4 กิโลไบต์ จากหมายเลข 0xffff_0000 ถึง 0xffff_1000
 - พื้นที่สำหรับจองหน่วยความจำเสมือน (vmalloc) ขนาด 1088 เมกะไบต์ จากหมายเลข 0xbb80_0000 ถึง 0xff80_0000
 - bss เช็คเม้นท์ (.bss) ขนาด 699 กิโลไบต์ จากหมายเลข 0x80c9_7f10 ถึง 0x80d4_68b0
 - ดาตาเช็คเม้นท์ (.data) ขนาด 577 กิโลไบต์ จากหมายเลข 0x80c0_0000 ถึง 0x80c9_017c

- init เช็คเมนท์ (.init) ขนาด 1024 กิโลไบท์ จากหมายเลข 0x80b0_8000 ถึง 0x80c0_0000
- เท็กซ์เช็คเมนท์ (.text) ขนาด 8160 กิโลไบท์ จากหมายเลข 0x8000_8000 ถึง 0x8080_0000

ในตัวอย่างนี้ ระบบสามารถตรวจสอบอุปกรณ์ USB และติดตั้งไดเรกอร์ได้อย่างถูกต้องปราศจาก ข้อผิดพลาด คำสั่งนี้จะแสดงรายละเอียดคร่าวๆ ของอุปกรณ์แต่ละตัวประมาณ 6-8 บรรทัด ผู้อ่านสามารถล้างบันทึกโดยใช้คำสั่ง ต่อไปนี้

```
$ sudo dmesg -C
```

โดย -C คือ Clear เป็นคำสั่งเพิ่มเติมให้ dmesg ล้างข้อมูลในบันทึกโดยการลบออก โปรดสังเกต ตัว C ใหญ่ หลังจากนั้น ผู้อ่านทดสอบโดยการกดเม้าส์ออก และเสียงกลับเข้าไปใหม่ ผู้อ่านสามารถแสดงข้อมูลที่เพิ่งเข้ามาในบันทึกโดยอีก โดยเรียกคำสั่ง dmesg อีกรอบ โดยข้อมูลเหล่านี้ เกิดจากผู้เขียนทดสอบและเสียงเม้าส์กลับเข้าไปใหม่อีกรอบ

```
[526.313715] usb 1-1.2: USB disconnect, device number 6
[527.653054] usb 1-1.2: new low-speed USB device number 7 using dwc_otg
[527.788253] usb 1-1.2: New USB device found, idVendor=046d, idProduct=c077
[527.788268] usb 1-1.2: New USB device strings: Mfr=1, Product=2,
                  SerialNumber=0
[527.788277] usb 1-1.2: Product: USB Optical Mouse
[527.788285] usb 1-1.2: Manufacturer: Logitech
[527.793119] input: Logitech USB Optical Mouse as /devices/platform/soc/
                  3f980000.usb/usb1/1-1/1-1.2/1-1.2:1.0/0003:046D:C077.0004/
                  input/input3
[527.793804] hid-generic 0003:046D:C077.0004: input,hidraw0: USB HID v1.11
                  Mouse [Logitech USB Optical Mouse] on usb-3f980000.usb-1.2/
                  input0
```

ผู้อ่านจะเห็นว่า อุปกรณ์ USB หมายเลข 6 ขาดการเชื่อมต่อ หลังจากนั้นเวลาผ่านไปประมาณ 1.3 วินาที และเชื่อมต่อใหม่เป็นอุปกรณ์หมายเลข 7 โดยระบบเก็บรายละเอียดทั้งหมดและพบว่า เม้าส์ USB นี้ มีหมายเลข idVendor=046d หมายเลข บริษัท Logitech, Inc. และ idProduct=c077 ซึ่งตรงกับเม้าส์ที่ได้จากการคำสั่ง lsusb

ในการเชื่อมต่อพอร์ท USB หากระบบแจ้งว่าอุปกรณ์โดยไม่มีข้อมูลใด พลาด แต่อุปกรณ์นั้นยังไม่สามารถทำงานได้ แสดงว่าอุปกรณ์ขาดซอฟต์แวร์ทำหน้าที่เป็นตัวไดเรกอร์ ขอให้ผู้อ่านค้นหาจากหมายเลขประจำตัวของผู้ผลิต (idVendor) หากผู้ผลิตมีไดเบิลเนชันซอฟต์แวร์ ผู้อ่านจำเป็นต้องดาวน์โหลดหรือคอมไพล์เองจากนักพัฒนารายอื่นแทน

I.4 พอร์ทเขื่อมต่อเครือข่าย WiFi และ Ethernet

I.4.1 รายชื่ออุปกรณ์เครือข่าย

ผู้อ่านสามารถตรวจสอบรายชื่ออุปกรณ์สำหรับเขื่อมต่อเครือข่ายได้จากคำสั่ง `ifconfig` ทางโปรแกรม Terminal ตัวอย่างผลลัพธ์เป็นดังนี้

```
$ ifconfig

eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
      inet 192.168.1.33 netmask 255.255.255.0 broadcast 192.168.1.255
      inet6 fe80::440b:2da7:638f:9061 prefixlen 64 scopeid 0x20<link>
        ether b8:27:eb:18:77:2d txqueuelen 1000 (Ethernet)
          RX packets 283 bytes 58857 (57.4 KiB)
          RX errors 0 dropped 2 overruns 0 frame 0
          TX packets 45 bytes 6515 (6.3 KiB)
          TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
      inet 127.0.0.1 netmask 255.0.0.0
      inet6 ::1 prefixlen 128 scopeid 0x10<host>
        loop txqueuelen 1000 (Local Loopback)
          RX packets 0 bytes 0 (0.0 B)
          RX errors 0 dropped 0 overruns 0 frame 0
          TX packets 0 bytes 0 (0.0 B)
          TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

wlan0: flags=4099<UP,BROADCAST,MULTICAST> mtu 1500
      ether b8:27:eb:4d:22:78 txqueuelen 1000 (Ethernet)
      RX packets 0 bytes 0 (0.0 B)
      RX errors 0 dropped 0 overruns 0 frame 0
      TX packets 0 bytes 0 (0.0 B)
      TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

โปรดสังเกตค่าเริ่มต้นในแต่ละรายการ ดังนี้

- eth0 ซึ่งหมายถึงอุปกรณ์เขื่อมต่อเครือข่ายสาย
- lo ซึ่งหมายถึงอุปกรณ์ Loopback สำหรับทดสอบการเชื่อมต่อขาสัญญาณ Tx ย้อนกลับมาที่ขา Rx และ

- wlan0 ซึ่งหมายถึงอุปกรณ์เชื่อมต่อเครือข่าย WiFi

I.4.2 การเปิดปิดอุปกรณ์เครือข่าย

ผู้อ่านสามารถใช้คำสั่ง ifconfig สำหรับเปิด หรือ ปิด อุปกรณ์ wlan0 ดังนี้

```
$ sudo ifconfig wlan0 down
$ ifconfig
$ sudo ifconfig wlan0 up
$ ifconfig
```

คำสั่ง \$ sudo ifconfig wlan0 down สำหรับสั่งปิดอุปกรณ์ ส่วนคำสั่งต่อมาใช้ทดสอบว่าอุปกรณ์ wlan0 ยังมีอยู่ในรายการหรือไม่ คำสั่ง \$ sudo ifconfig wlan0 up สำหรับสั่งเปิดอุปกรณ์ ส่วนคำสั่งสุดท้ายใช้ทดสอบว่าอุปกรณ์ wlan0 เปิดทำงานหรือยัง

ผู้อ่านสามารถเปิดปิดอุปกรณ์อื่นๆ โดยการพิมพ์ชื่อแทนที่ชื่ออุปกรณ์ wlan0 ได้ตามต้องการ เช่น

```
$ sudo ifconfig eth0 down
$ sudo ifconfig eth0 up
```

นอกเหนือจากการเปิดปิดอุปกรณ์เครือข่าย ผู้อ่านสามารถตรวจสอบรายชื่อเครือข่าย WiFi ที่บอร์ด เคยเชื่อมต่อสำเร็จได้จากไฟล์ wpa_supplicant.conf ซึ่งจะบันทึกรายละเอียดต่างๆ ของการเชื่อมต่อนั้นๆ รวมถึงพาสเวิร์ด (password) โดยพิมพ์คำสั่งต่อไปนี้ใน Terminal

```
$ cat /etc/wpa_supplicant/wpa_supplicant.conf
```

นี่เป็นตัวอย่างผลลัพธ์ที่ได้ โดย

```
network={
ssid="CE_ParaLab24"
psk="*****"
key_mgmt=WPA-PSK
}
```

- ssid หมายถึงชื่อเครือข่าย WiFi ซึ่งในตัวอย่าง คือ CE_ParaLab24
- psk ย่อมาจาก Public Shared Key โดยผู้เขียนได้ใส่ตัวอักษรอื่นแทนเพื่อความปลอดภัย
- key_mgmt คือ วิธีการเข้ารหัสและจัดการคีย์ หรือพาสเวิร์ด ซึ่งในตัวอย่าง คือ WPA-PSK ขึ้นกับผู้ดูแล (Administrator) ได้ติดตั้ง (Configure) อุปกรณ์ WiFi นั้น

I.4.3 การตรวจสอบการเชื่อมต่อกับเครือข่ายเบื้องต้น

เมื่อผู้อ่านเปิดและทำการเชื่อมต่อสำเร็จ แล้วจึงสามารถตรวจสอบการเชื่อมต่อในระดับชั้นเครือข่าย โดยใช้คำสั่ง ping ใน Terminal ดังนี้

```
$ ping <ip add or host name>
```

การตรวจสอบการเชื่อมต่อเบื้องต้น คือ การ ping ไปหาเราเตอร์ผ่านทางที่บอร์ดเชื่อมต่อ ผู้อ่านสามารถสืบค้นหมายเลขไอพีของเราเตอร์ที่ตั้งทาง โดยสังเกตที่ inet ของ eth0 หรือ wlan0 ว่าเริ่มต้นด้วยหมายเลข 192.168.x.y ซึ่งเราเตอร์ตั้งทางมักจะมีหมายเลข 192.168.x.1 หรือ 192.168.x.254

นี่เป็นตัวอย่างผลลัพธ์ที่ได้จากการคำสั่ง ping 192.168.1.1

```
PING 192.168.1.1 (192.168.1.1) 56(84) bytes of data.
```

```
64 bytes from 192.168.1.1: icmp_seq=1 ttl=64 time=2.03 ms
64 bytes from 192.168.1.1: icmp_seq=2 ttl=64 time=1.98 ms
64 bytes from 192.168.1.1: icmp_seq=3 ttl=64 time=25.3 ms
64 bytes from 192.168.1.1: icmp_seq=4 ttl=64 time=38.2 ms
64 bytes from 192.168.1.1: icmp_seq=5 ttl=64 time=53.3 ms
64 bytes from 192.168.1.1: icmp_seq=6 ttl=64 time=37.6 ms
64 bytes from 192.168.1.1: icmp_seq=7 ttl=64 time=18.9 ms
64 bytes from 192.168.1.1: icmp_seq=8 ttl=64 time=17.4 ms
64 bytes from 192.168.1.1: icmp_seq=9 ttl=64 time=6.99 ms
```

โดย 192.168.1.1 คือหมายเลขไอพีแอดเดรสของอุปกรณ์ที่คำสั่งจะส่งแพ็กเก็ต ICMP (Internet Control Message Protocol) ความยาว 64 ไบท์ไป แล้วรออุปกรณ์หมายเลขนี้ตอบกลับมายังบอร์ด Pi3 โดยจับเวลาตั้งแต่ส่งไปและรอตอบกลับมา ของแพ็กเก็ตลำดับที่ 1 (icmp_seq=1) เป็นระยะเวลา 2.03 มิลลิวินาที ส่วน ttl=64 ย่อมาจากคำว่า time to live หมายถึง เลขจำนวนเต็มที่ผู้ส่งกำหนดค่าอายุของแพ็คเก็ต ที่สามารถเดินทางผ่านเครือข่าย หากตั้งไว้น้อยจะทำให้แพ็คเก็ตข้อมูลนี้อายุสั้นและอาจเดินทางไปไม่ถึงปลายทางเนื่องจากหมดอายุก่อน โดย ttl=64 เป็นค่าปกติ

ผู้อ่านจะสังเกตเห็นว่า ระยะเวลาเมื่อค่าตั้งแต่ 1.98-53.3 มิลลิวินาที ขึ้นอยู่กับคุณภาพ ของสาย Ethernet หรือความแรงของสัญญาณ WiFi คุณภาพดีจะทำให้ระยะเวลาสั้นกว่า หลังจากตรวจสอบว่าบอร์ดสามารถเชื่อมต่อกับเราเตอร์ตั้งทางได้ตามตัวอย่างก่อนหน้า ผู้อ่านสามารถใช้ตรวจสอบการเชื่อมต่อได้ว่า เราเตอร์ตั้งทางสามารถเชื่อมต่อกับเครือข่ายอินเตอร์เน็ตได้สำเร็จหรือไม่ โดย Host name คือ ชื่อเซิร์ฟเวอร์ปลายทางที่จะทะเบียนโดเมนเนม (Domain Name) เรียบร้อยแล้ว เช่น ping www.google.com

I.5 กิจกรรมท้ายการทดลอง

1. จงค้นหาว่าความละเอียดของการแสดงผลผ่านพอร์ท HDMI ในหัวข้อที่ [I.1.2](#) เก็บบันทึกลงในไฟล์ ชื่ออะไร
2. ใช้คำสั่ง ifconfig ปิดอุปกรณ์ lo0 แล้วใช้คำสั่ง ping 127.0.0.1 ว่ามีการตอบสนองกลับมาหรือไม่ เปิดอุปกรณ์ lo0 แล้ว ping อีกรอบ จนอธิบายว่า 127.0.0.1 คือ อะไร
3. ใช้คำสั่ง ping เพื่อทดสอบเราเตอร์ที่อยู่ต้นทางของผู้อ่าน 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

การทดลองที่ 10 การเชื่อมต่อกับ GPIO

การทดลองนี้คาดว่าผู้อ่านเคยเรียนการเขียนหรือพัฒนาโปรแกรมด้วยภาษา C มาบ้างแล้ว และมีความคุ้นเคยกับ IDE (Integrated Development Environment) จากพัฒนาโปรแกรมและการดีบักโปรแกรมด้วยภาษา C/C++ และแօสเซมบลี ดังนั้น การทดลองมีวัตถุประสงค์เหล่านี้

- เพื่อปฏิบัติการเชื่อมต่อวงจรกับขา GPIO บนบอร์ด Pi3 ตามเนื้อหาในบทที่ 2 หัวข้อที่ 2.11
- เพื่อพัฒนาโปรแกรมภาษา C ควบคุมการทำงานของขา GPIO
- เพื่อพัฒนาโปรแกรมภาษา Assembly ควบคุมการทำงานของขา GPIO

โปรดสังเกตตัวอักษร W ที่คำว่า wiringPi ต้องเป็นตัวอักษรพิมพ์เล็ก

J.1 ไลบรารี wiringPi

ไลบรารี wiringPi เป็นฟังก์ชันที่พัฒนาด้วยภาษา C สำหรับบอร์ด Pi เป็น OpenSource ภายใต้ GNU LGPLv3 license สามารถเรียกใช้งานผ่านภาษา C and C++ รวมถึงแօสเซมบลี

เนื่องจากไลบรารีเป็นซอฟต์แวร์แบบ Open Source แจกให้แก่นักพัฒนาทั่วโลกผ่านทาง <https://github.com/WiringPi> และมีการปรับปรุงแก้ไขตลอดเวลาโดยทีมนักพัฒนา ดังนั้น ผู้อ่านควรต้องติดตั้งและปรับปรุงระบบปฏิบัติการให้ทันสมัยและติดตั้ง ตามขั้นตอนต่อไปนี้

- ผู้อ่านควรปรับปรุงระบบปฏิบัติการให้เป็นปัจจุบันก่อน โดยพิมพ์คำสั่งนี้บน Terminal โดยใช้สิทธิ์ของ SuperUser:

```
$ sudo apt-get update  
$ sudo apt-get upgrade
```

ขั้นตอนนี้จะใช้เวลานานและความอดทน รวมถึงการเชื่อมต่อกับเครือข่ายอินเทอร์เน็ตที่มีเสถียรภาพ

- ติดตั้ง wiringPi โดยพิมพ์คำสั่งนี้บน Terminal โดยใช้สิทธิ์ของ SuperUser:

Appendix J. การทดลองที่ 10 การเชื่อมต่อ กับ GPIO

```
$ sudo apt-get install wiringPi
```

คำสั่งนี้จะติดตั้งไลบรารีลงบนบอร์ด

3. เรียกคำสั่ง gpio -v เพื่อทดสอบการติดตั้งไลบรารี wiringPi และได้ผลลัพธ์ของการเรียกดังนี้

```
$ gpio -v
gpio version: 2.50
Copyright (c) 2012-2018 Gordon Henderson
This is free software with ABSOLUTELY NO WARRANTY.
For details type: gpio -warranty
```

Raspberry Pi Details:

```
Type: Pi 3, Revision: 02, Memory: 1024MB, Maker: Sony
* Device tree is enabled.
*--> Raspberry Pi 3 Model B Rev 1.2
* This Raspberry Pi supports user-level GPIO access.
```

4. เรียกคำสั่ง gpio readall เพื่อตรวจสอบและบันทึกผลลัพธ์ที่แสดงบนหน้าต่าง Terminal ลงในตารางหน้าถัดไป

```
$ gpio readall
```

จะเติมหมายเลขในคอลัมน์ wPi (wiringPi) ให้ตรงกับขาเชื่อมต่อ 40 ขางบนบอร์ด 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		CEO	_	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 กระแสฟริบ

1. รายการอุปกรณ์ที่ต้องใช้:

- หลอด LED จำนวน 3 หลอด
- ตัวต้านทาน (Resistor) ที่เตรียมไว้ให้จำนวน 3 ตัว
- แผ่นต่อวงจรprotoboard
- สายต่อวงจร

2. ซัดดาวน์และตัดไฟเลี้ยงออกจากบอร์ด Pi3 เพื่อความปลอดภัยในการต่อวงจร

3. ศึกษารูปที่ ?? ให้เข้าใจ และจึงต่อวงจรตามรูปที่ J.1

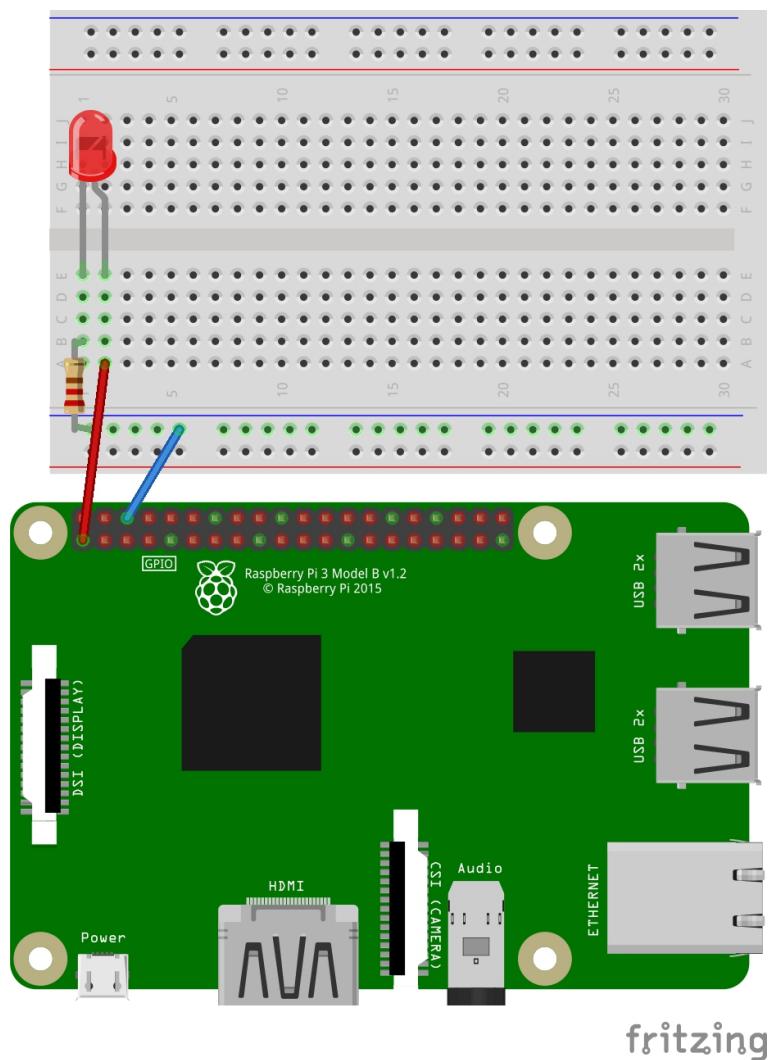


Figure J.1: วงจรเชื่อมต่อหลอด LED กับบอร์ด Pi3 ในการทดลองที่ 10 ที่มา: fritzing.org

4. ตรวจสอบความถูกต้อง โดยให้ผู้ควบคุมการทดลองตรวจสอบ

5. จ่ายไฟเลี้ยงให้กับบอร์ดแล้วสังเกตการเปลี่ยนแปลงที่หลอด LED

J.3 โปรแกรมไฟ LED กระพริบภาษา C

1. เรียกโปรแกรม Code::Blocks ผ่านทาง Terminal โดยใช้สิทธิ์ของ SuperUser ดังนี้

```
$ sudo codeblocks
```

2. สร้าง project ใหม่ชื่อ Lab10 จะเสร็จสิ้น

3. คลิกเมนู "Setting/Compiler..." เลือกแท็บ "Linker settings" และกดปุ่ม "Add"

4. ป้อนประโยชน์ "/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("wiringPi LED blinking\n");
    if (wiringPiSetup() == -1) {
        printf( "Setup problem ... Abort!" );
        exit (1);
    }
    pinMode(pin, OUTPUT);
    int i;
    for ( i=0; i<10; i++ ) {
        digitalWrite(pin, 1); /* LED On */
        delay(250);
        digitalWrite(pin, 0); /* LED Off */
        delay(250);
    }
    return 0;
}
```

7. ทำการ Build และแก้ไขหากมีข้อผิดพลาดจนสำเร็จ

8. Run และสังเกตการเปลี่ยนแปลงที่หลอดไฟ LED
9. จับเวลาช่วงเวลาที่หลอดสว่างและตั้งแต่เริ่มรันโปรแกรมจนเสร็จสิ้น เพื่อหาค่าเฉลี่ยของการสว่างตั้ง 1 รอบ

J.4 โปรแกรมไฟ LED กระพริบภาษา Assembly

1. เปิดโฟลเดอร์ `/home/pi/asm` ในโปรแกรมไฟล์เมเนเจอร์
2. สร้างโฟลเดอร์ใหม่ชื่อ `Lab10`
3. สร้างไฟล์ใหม่ชื่อ `Lab10.s` โดยใช้คำสั่ง `touch`
4. กรอกโปรแกรมภาษาแอสเซมบลีเหล่านี้ลงไป

```
#-----
# data segment
#-----
.data
.balign 4
intro: .asciz "wiringPi LED blinking\n"
errMsg: .asciz "Setup problem ... Abort!\n"
pin: .int 7
i: .int 0
duration:.int 250
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:

```

Appendix J. การทดลองที่ 10 การเชื่อมต่อ กับ GPIO

POP {ip, pc} @pop return address into pc

5. ทำการแปลและลิงค์ Lab10.s จนกว่าจะสำเร็จ:

```
$ as -o Lab10.o Lab10.s  
$ gcc -o Lab10 Lab10.o -lwiringPi
```

6. รันโปรแกรม Lab10 และสังเกตการเปลี่ยนแปลงที่หลอดไฟ LED

```
$ sudo ./Lab10
```

7. จับเวลาช่วงเวลาที่หลอดสว่างและดับตั้งแต่เริ่มรันโปรแกรมจนเสร็จสิ้น เพื่อหาค่าเฉลี่ยของการสว่างดับ 1 รอบ

J.5 กิจกรรมท้ายการทดลอง

1. สำรวจไฟล์ชื่อ wiringPi.c ในไดเรกทอรีชื่อ /home/pi/wiringPi/wiringPi/ เพื่อค้นหาตัวแปรชื่อ piGpioBase ว่า

- ใช้งานในฟังก์ชันชื่ออะไร
- ได้รับการตั้งค่าที่ฟังก์ชันชื่ออะไร และค่าเท่ากับเท่าไหร
- นำตัวแปร piGpioBase นี้ไปใช้ทำอะไรต่อได้อีก จงยกตัวอย่าง
- หมายเลขแอดเดรส 0x2000_0000 นี้เกี่ยวข้องกับหมายเลข 0x7E00_0000 ในตารางที่ [2.4](#) และรูปที่ [2.16](#) อย่างไร

2. จงตอบคำถามจากประโยชน์ต่อไปนี้

```
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 gpio นี้ไปใช้ทำอะไรต่อได้อีก จงยกตัวอย่าง
- จงอธิบายว่าตัวแปร gpio นี้เกี่ยวข้องกับหลักการ Memory Map IO อย่างไร

3. จงตอบคำถามจากประโยชน์ต่อไปนี้

```
GPIO_BASE = piGpioBase + 0x00200000 ;
```

- อยู่ในฟังค์ชันชื่ออะไร
 - ตัวแปร GPIO_BASE มีหน้าที่อะไร
 - เมื่อบอกแล้วได้ผลลัพธ์เป็นหมายเลขและตรวจสอบอะไร และเกี่ยวข้องกับหมายเลข 0x7E20_0000 ในตารางที่ [2.6](#) อย่างไร
 - นำตัวแปร GPIO_BASE นี้ไปใช้ทำอะไรต่อได้อีก จงยกตัวอย่าง
 - จะอธิบายว่าตัวแปร GPIO_BASE นี้เกี่ยวข้องกับขา gpio แต่ละขาอย่างไร
4. ต่อหลอด LED เพิ่มอีก 2 ดวงรวมเป็น 3 ดวงแล้วพัฒนาโปรแกรมภาษา C เดิมให้นับเลข 0-7 และแสดงผลทางหลอด LED เป็นเลขฐานสองวนไปเรื่อยๆ
 5. ใช้งานหลอด LED 3 ดวงที่มีอยู่และพัฒนาโปรแกรมภาษาแอสเซมบลีเดิมให้นับเลข 0-7 และแสดงผลทางหลอด LED เป็นเลขฐานสองวนไปเรื่อยๆ

Appendix K

การทดลองที่ 11 การเชื่อมต่อ กับ อินเทอร์รัพท์

การทดลองนี้คัดว่าผู้อ่านเคยเรียนการเขียนหรือพัฒนาโปรแกรมด้วยภาษา C และแอกซ์เพรสชันบลี ดังนั้น การทดลองมีวัตถุประสงค์เหล่านี้

- เพื่อพัฒนาการทำงานของอินเทอร์รัพท์ร่วมโปรแกรมภาษา C
- เพื่อศึกษาการทำงานของอินเทอร์รัพท์ร่วมกับขา GPIO ตามเนื้อหาในบทที่ 2 หัวข้อที่ 2.11

K.1 อินเทอร์รัพท์

สัญญาณร้องขอ อินเทอร์รัพท์ หรือ สัญญาณร้องขอ การขัดจังหวะ (Interrupt Request) คือ สัญญาณที่เกิดขึ้นจากอุปกรณ์อินพุท/เอาท์พุท จาก GPU และจากเหตุการณ์พิเศษ สัญญาณเหล่านี้จะทำให้ CPU หยุดพักโปรแกรมที่กำลังรันอยู่ เป็นการชั่วคราว และไปดำเนินการบางอย่างเพื่อตอบสนอง (Respond) หรือให้บริการ (Service) ต่อเหตุการณ์ที่เกิดขึ้น การตอบสนองหรือการบริการ นี้เรียกโดยรวมว่า **Interrupt Service Routine (ISR)** เมื่อซีพียูตอบสนองหรือบริการเสร็จสิ้น ซีพียูจะกลับไปทำงานที่หยุดพักนั้นต่อ

รายละเอียดการทำงานของอินเทอร์รัพท์ มีขั้นตอนดังนี้

- เหตุการณ์ อินเทอร์รัพท์ (Interrupt Event):** กดปุ่มต่างๆ การกดแป้นพิมพ์ การจับเวลา (Timer) เป็นต้น
- การร้องขอ การขัดจังหวะ (Interrupt Request):** ส่งสัญญาณร้องขอไปยังซีพียู
- ขั้นตอน การให้บริการ (Interrupt Service Routine เรียกว่า ISR):** พังค์ชันที่ซีพียูจะต้องปฏิบัติเพื่อให้บริการตามเหตุการณ์ที่ร้องขอ

เหตุการณ์ อินเทอร์รัพท์ มีความสำคัญ (Priority) แตกต่างกัน ISR สำหรับแต่ละเหตุการณ์มักจะเขียนในรูปแบบของฟังก์ชัน ที่ไม่มีพารามิเตอร์และไม่มีค่ารีเทิร์น หรือ void เหตุการณ์แบ่งเป็น 2 ชนิด คือ

- ฮาร์ดแวร์ อินเทอร์รัพท์ (Hardware interrupts):** เหตุการณ์ที่เกิดจากการทำงานร่วมกับอุปกรณ์ อินพุทและเอาท์พุท เช่น ปุ่มกดต่างๆ การรับส่งข้อมูลแบบอนุกรม (Serial communication) เช่น UART (Universal Asynchronous Receive Transmit), SPI (Serial Peripheral Interface)

Appendix K. การทดลองที่ 11 การเชื่อมต่อกับอินเทอร์รัพท์

เป็นต้น การเปลี่ยนแปลงของขา GPIO, ตัวจับเวลาถึงเวลาที่ตั้งไว้, การแปลงอนาล็อกเป็นดิจิทัล เสรีจสมบูรณ์, ตัวจับเวลาอัฟช็อก (Watchdog Timer) หมดเวลา (Timeout) เป็นต้น

- ซอฟต์แวร์อินเทอร์รัพท์ (**Software interrupts**): เหตุการณ์ที่เกิดจากการทำงานหรือสั่งการโดยซอฟต์แวร์ เช่น การเรียกใช้บริการจากโอเอส ความผิดพลาดของโปรแกรม เป็นต้น

K.2 การจัดการอินเตอร์รัพท์ (Interrupt Handling)

K.2.1 การจัดการอินเทอร์รัพท์ของ WiringPi

ไลบรารี wiringPi รองรับการทำอินเทอร์รัพท์ของ GPIO ได้ ทำให้โปรแกรมหลักสามารถทำงาน หลักได้ตามปกติ เมื่อเกิดสัญญาณอินเทอร์รัพท์ขึ้น ไม่ว่าจะเป็นสัญญาณจากการกดปุ่ม ทำให้เกิดขอบขั้นหรือขอบขาลงหรือทั้งสองขอบ โดยการเรียกใช้คำสั่ง

```
wiringPiISR(pin, edgeType, callback)
```

โดย pin หมายถึง เลขขาที่ wiringPi กำหนด edgeType กำหนดจากค่าคงที่ 4 ค่านี้

- INT_EDGE_FALLING,
- INT_EDGE_RISING,
- INT_EDGE_BOTH
- INT_EDGE_SETUP.

การกำหนดชนิดขอบขาเป็น 3 ชนิดแรก ไลบรารีจะตั้งค่าเริ่มต้น (Initialization) ให้โดยอัตโนมัติ หากกำหนดชนิดขอบเป็น INT_EDGE_SETUP ไลบรารีจะไม่ตั้งค่าเริ่มต้น (Initialization) ให้ เนื่องจากโปรแกรมเมอร์จะต้องทำเอง

พารามิเตอร์ callback คือ ชื่อฟังก์ชันที่จะทำหน้าที่เป็น ISR ฟังก์ชัน callback นี้จะเริ่มต้นทำงานโดยแจ้งต่อว่าจร Dispatcher ในหัวข้อที่ [2.12](#) ก่อนจะเริ่มต้นทำงาน โดยฟังก์ชัน callback จะสามารถอ่านหรือเขียนค่าของตัวแปรโกลบออนไลน์โปรแกรมได้ ซึ่งตัวอย่างการทำงานจะได้กล่าวในหัวข้อถัดไป

K.2.2 วงจรปุ่มกด Push Button เชื่อมผ่านขา GPIO

1. ซัดดาวน์และตัดไฟเลี้ยงออกจากบอร์ด Pi3 เพื่อความปลอดภัยในการต่อวงจร
2. ต่อวงจรตามรูปที่ [K.1](#)

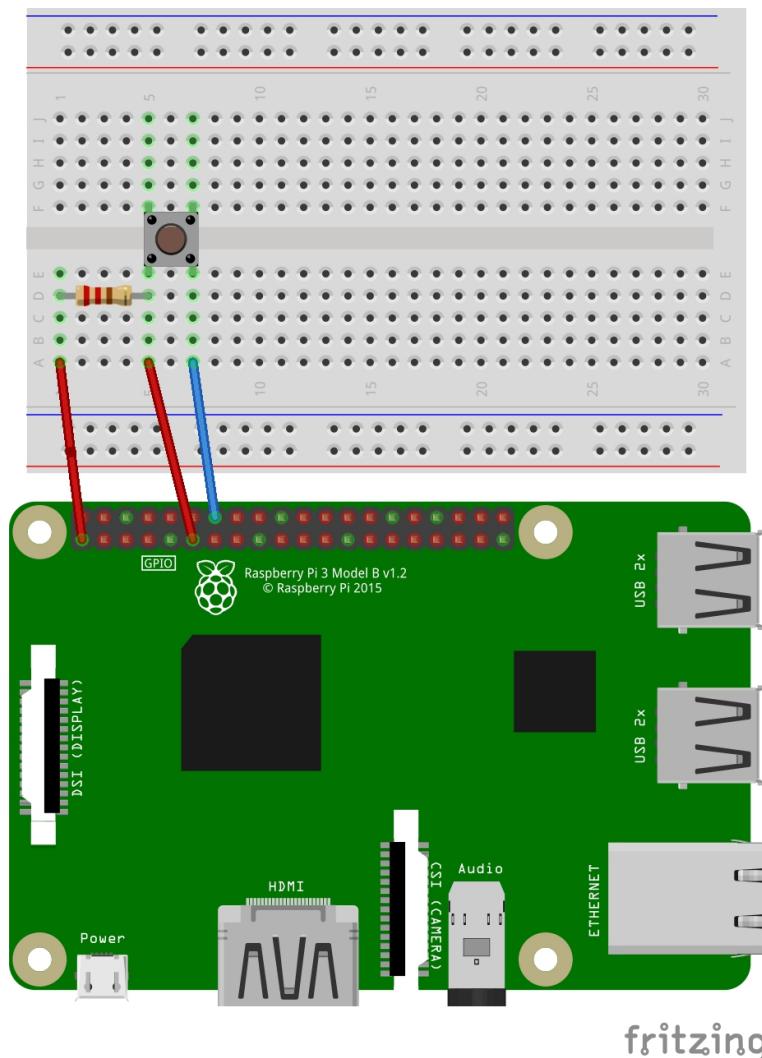


Figure K.1: วงจรกดปุ่มสำหรับทดลองการเขียนโปรแกรมอินเทอร์รัพท์ในการทดลองที่ 11 ที่มา: fritzing.org

3. ตรวจสอบความถูกต้อง โดยให้ผู้ควบคุมการทดลองตรวจสอบ
4. จ่ายไฟเลี้ยงให้กับบอร์ดแล้วสังเกตการเปลี่ยนแปลงที่หลอด LED
5. เรียกโปรแกรม Code::Blocks ผ่านทาง Terminal โดยใช้สิทธิของ SuperUser ดังนี้

```
sudo codeblocks
```

6. สร้าง project ใหม่ชื่อ Lab11.2 ภายใต้โฟลเดอร์ /home/pi/asm/Lab11.2 รายละเอียดบางอย่าง ต้องเปิดในการทดลองที่ 8 ภาคผนวก J

K.2.3 โปรแกรมภาษา C สำหรับทดสอบวงจรอินเทอร์รัฟท์

ผู้อ่านต้องทำความเข้าใจกับตัวโปรแกรมก่อนคอมไพล์หรือรันโปรแกรม เพื่อความเข้าใจสูงสุด โดยเฉพาะ ชื่อตัวแปร ชนิดของตัวแปร evenCounter การติดตั้งฟังค์ชัน wiringPiISR เพื่อเชื่อมโยงกับขา GPIO ชนิดของการตรวจจับ และชื่อฟังค์ชัน myInterrupt ซึ่งทำหน้าที่เป็น ISR หรือ ฟังค์ชัน callback

```
#include <stdio.h>
#include <errno.h>
#include <wiringPi.h>
#define BUTTON_PIN 0
// Use GPIO Pin 17, which is Pin 0 for wiringPi library

volatile int eventCounter = 0;

// myInterrupt: called every time an event occurs
void myInterrupt(void) {
    eventCounter++; // the event counter
}

int main(void) {
    if (wiringPiSetup () < 0) // check the existence of wiringPi library
    {
        printf ( "Unable to setup wiringPi: %s\n", strerror (errno));
        return 1;
    }
    // set wiringPi Pin 0 to generate an interrupt from 1-0 transition
    // myInterrupt() = ISR
    if ( wiringPiISR (BUTTON_PIN, INT_EDGE_FALLING, &myInterrupt) < 0)
    {
        printf ( "Unable to setup ISR: %s\n", strerror (errno));
        return 1;
    }
    // display counter value every second
    while ( 1 ) {
        printf( "%d\n", eventCounter );
        eventCounter = 0;
        delay( 1000 ); // wait 1 second
    }
}
```

```

    }
    return 0;
}

```

1. ป้อนโปรแกรมด้านบนใน main.c และคอมpile จนไม่เกิดข้อผิดพลาด
2. รันโปรแกรม ทดสอบการทำงานด้วยการกดปุ่มที่ต่อไว้ สังเกตผลลัพธ์ทางหน้าจอ Terminal ที่รัน
3. จงบอกความหมายและการประยุกต์ใช้งานตัวแปรชนิด volatile
4. ปรับแก้ volatile ออกเหลือแค่ int eventCounter = 0;
5. รันโปรแกรม ทดสอบการทำงานด้วยการกดปุ่มที่ต่อไว้ สังเกตผลลัพธ์ทางหน้าจอ Terminal ที่รัน
6. เปรียบเทียบการทำงานของโปรแกรมก่อนและหลังการปรับแก้ และหาเหตุผล

K.3 กิจกรรมท้ายการทดลอง

1. จงตอบคำถามจากประโยชน์ต่อไปนี้

```

if ( wiringPiISR (BUTTON_PIN, INT_EDGE_FALLING, &myInterrupt) < 0) {
}

```

- พังค์ชัน wiringPiISR ทำหน้าที่อะไร เหตุใดอยู่ในประโยชน์ของ if
- ตัวแปร &myInterrupt คืออะไร เหตุใดจึงมีสัญลักษณ์ & นำหน้า
- พังค์ชันนี้เชื่อมโยงกับตารางที่ [2.6](#) อย่างไร

2. จงใช้วงจรหลอด LED 3 ดวงและโปรแกรมจากการทดลองที่ 8 นับขึ้นจาก 0-7-0 โดยเพิ่มปุ่มกดในการทดลองนี้ และเพิ่มพังค์ชันการอินเทอร์รัฟท์จากโปรแกรม Lab11.2 นี้ เมื่อกดปุ่มแต่ละครั้งจะทำให้ความเร็วในการนับเพิ่มขึ้น หรือ delay สั้นลงครึ่งหนึ่ง เมื่อกดครั้งที่ 2 จะสั้นลงอีกครึ่งหนึ่ง เมื่อกดครั้งที่ 3 จะทำให้ Delay กลับไปเป็นค่าเริ่มต้น
3. จงใช้วงจรหลอด LED 3 ดวงและโปรแกรมจากการทดลองที่ 8 แต่นับลง จาก 7-0-7 โดยเพิ่มปุ่มกดในการทดลองนี้ และเพิ่มพังค์ชันการอินเทอร์รัฟท์จากโปรแกรม Lab11.2 นี้ เมื่อกดปุ่มแต่ละครั้ง จะทำให้ความเร็วในการนับลดลง หรือ delay เพิ่มขึ้นเท่าตัว เมื่อกดครั้งที่ 2 Delay เพิ่มขึ้นอีกเท่าตัว เมื่อกดครั้งที่ 3 จะทำให้ Delay กลับไปเป็นค่าเริ่มต้น

Appendix L

การทดลองที่ 12 การศึกษาอุปกรณ์เก็บรักษาข้อมูลและระบบไฟล์

การทดลองนี้อธิบายและเข้มโยงเนื้อหาความรู้ของทุกบทเข้าด้วยกัน แต่จะเน้นบทที่ 2 และบทที่ 7 เพื่อให้ผู้อ่านมองเห็นอุปกรณ์อินพุตและเอาท์พุตเมื่อไฟล์แต่ละไฟล์ โดยมีวัตถุประสงค์ดังนี้

- เพื่อให้เข้าใจขนาดของไฟล์และไฟล์เดอร์ในระบบไฟล์
- เพื่อให้รู้จักโครงสร้างและระบบไฟล์ของหน่วยความจำการ์ด microSD ที่ใช้งานในปัจจุบัน
- เพื่อให้เข้าใจระบบไฟล์ (File System) ชนิดต่างๆ บนบอร์ด Pi3
- เพื่อให้สามารถเข้มโยงอุปกรณ์อินพุตเอาท์พุตนิดต่างๆ กับระบบไฟล์

L.1 ขนาดของไฟล์และไดเรคทอรี

ผู้อ่านสามารถตรวจสอบขนาดของไฟล์ได้ฯ ชื่อ filename ที่แท้จริง หน่วยเป็นไบท์ ด้วยคำสั่งต่อไปนี้ du (Disk Usage)

- ย้ายไฟล์เดอร์ปัจจุบันไปที่ /home/pi ซึ่งเป็นไฟล์เดอร์หลักของผู้ใช้ชื่อ pi

```
$ cd /home/pi
```

- สร้างไฟล์ข้อความ test.txt ด้วยโปรแกรม nano ด้วยคำสั่งต่อไปนี้

```
$ nano test.txt
```

พิมพ์ข้อความ fdd ลงในไฟล์ ทำการ Write โดยกดปุ่ม Ctrl แซ่ตตามด้วยปุ่ม 0 ออกจากโปรแกรม โดยกดปุ่ม Ctrl แซ่ตตามด้วยปุ่ม x

- คำสั่ง ‘du -b filename’ จะแสดงผลขนาดเป็นจำนวนไบท์นำหน้าชื่อไฟล์นั้น

Appendix L. การทดลองที่ 12 การศึกษาอุปกรณ์เก็บรักษาข้อมูลและระบบไฟล์

```
$ du -b test.txt
4 test.txt
```

4 หมายถึง เลขจำนวนไบท์ที่คำสั่ง du แสดงผลมาตามพารามิเตอร์ b ที่ส่งไป เพื่อบอกค่าขนาดของไฟล์ test.txt เป็นจำนวน 4 ไบท์

- คำสั่ง ‘du -B1 filename’ ผู้อ่านสามารถตรวจสอบขนาดของไฟล์ใดๆ ชื่อ filename ที่จัดเก็บเป็นจำนวนเท่าของ 4096 ไบท์ ในอุปกรณ์เก็บรักษาข้อมูล SD ด้วยคำสั่งต่อไปนี้

```
$ du -B1 test.txt
4096 test.txt
```

4096 หมายถึง เลขจำนวนไบท์ที่คำสั่ง du แสดงผลมาตามพารามิเตอร์ B1 ที่ส่งไป โดยผู้อ่านจะสังเกตเห็นความแตกต่าง ถึงแม้ไฟล์มีข้อมูลจำนวนน้อยเพียงไม่ถึงไบท์ แต่การจองพื้นที่ในอุปกรณ์สำรองจะมีขนาดเป็นจำนวนเท่าของ 4096 ไบท์เสมอ

- คำสั่ง ‘du -h’ จะแสดงผลขนาดหรือจำนวนไบท์โดยใช้หน่วยเช่น K (Kilo) M (Mega) G (Giga) นำหน้าชื่อไฟล์เดอร์หรือไดเรกทอรีที่อยู่ใต้ไฟล์เดอร์ปัจจุบัน และจดบันทึก 10 รายการสุดท้ายลงในตาราง

```
$ du -h
```

Size	Folder Name
160e	./local/share/applications
400u	./local/share/gvfs-metadata
4.0k	./local/share/codelocks/plugins
4.0k	./local/share/codelocks/scripts
12u	./local/share/codelocks
4.0u	./local/share/desktop-directories
116e	./local/share
190k	./local
4.0k	./public
172M	.

L.2 ระบบไฟล์

- คำสั่ง df (Disk File System) สามารถแสดงรายละเอียดของอุปกรณ์เก็บรักษาข้อมูลในเครื่อง
 - คำสั่ง ‘df -h’ จะแสดงรายการ ดังต่อไปนี้ จดบันทึก 10 รายการแรกลงในตาราง

```
$ df -h
```

Filesystem	Size	Used	Avail	Use%	Mounted on
/dev/root	15G	3.6G	11G	26%	/
devtmpfs	459M	0	459M	0%	/dev
tmpfs	464M	0	464M	0%	/dev/shm
tmpfs	464M	24M	440M	6%	/run
tmpfs	5.0M	4.0M	5.0M	1%	/run/lock
tmpfs	464M	0	464M	0%	/sys/fs/cgroup
/dev/mmcblk0p1	953M	53M	900M	2%	/boot
tmpfs	93M	0	93M	0%	/run/user/1000

โดย Size จะแสดงผลขนาดหรือจำนวนไบท์โดยใช้หน่วยเช่น K (Kilo) M (Mega) G (Giga)

- คำสั่ง ‘df -T’ จะเพิ่มคอลัมน์ชนิด (Type) ของแต่ละรายการในการแสดงผล และขนาดเป็นจำนวนเท่าของ 1 กิโลไบท์ (1K) แทน จดบันทึก 5 รายการแรกลงในตาราง

```
$ df -T
```

- คำสั่ง ‘df -i’ จะแสดงรายการต่างๆ ดังนี้ จดบันทึก 10 รายการแรกลงในตาราง

Appendix L. การทดลองที่ 12 การศึกษาอุปกรณ์เก็บรักษาข้อมูลและระบบไฟล์

```
$ df -i
```

โดยคอลัมน์จะแสดงผลเป็นจำนวน inode แทน รายละเอียดเรื่อง inode ผู้อ่านสามารถค้นคว้าเพิ่มเติมได้ในบทที่ 7 และทาง [wikipedia](#)

- คำสั่ง stat แสดงรายละเอียดของไฟล์หรือไดเรคทอรี

```
$ stat asm
```

```
File: asm
Size: 4096          Blocks: 8           IO Block: 4096   directory
Device: b307h/45831d Inode: 521754        Links: 3
Access: (0755/drwxr-xr-x) Uid: ( 1000/      pi)  Gid: ( 1000/      pi)
Access: 2019-03-19 19:43:05.449401732 +0700
Modify: 2019-03-19 19:43:05.449401732 +0700
Change: 2019-03-19 19:43:05.449401732 +0700
Birth: -
```

ผู้เขียนขอรับคำแนะนำและคำติชมจากผู้เชี่ยวชาญด้านภาษาไทย

- ชื่อ asm
 - ขนาด 4096 ไบต์ ใช้พื้นที่จำนวน 8 Blocks เป็นไดเรกทอรี (directory)
 - มีหมายเลข Device = b307h/45831d หรือเท่ากับ $b307_{16}/45831_{10}$
 - มีหมายเลข Inode ที่ 521754 จำนวน 3 Links
 - สิทธิ์เข้าถึง (Access) ด้วยรหัส 0644 หรือ $011_2:100_2:100_2$ โดยผู้ใช้หมายเลข Uid (User ID)=1000 ชื่อผู้ใช้ (Username)=pi ในกรุํปหมายเลข Groupid=1000 ชื่อกรุํป pi
 - เข้าถึง (Access) ณ วันที่ 19 มีนาคม 2019 เวลา 19.43.05

- เปลี่ยนแปลง (Modify) ณ วันที่ 19 มีนาคม 2019 เวลา 19.43.05
- เวลาที่ Inode เปลี่ยนแปลง (Change) ณ วันที่ 19 มีนาคม 2019 เวลา 19.43.05

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: 341          Blocks: 8          IO Block: 4096   regular file
Device: b307h/45831d  Inode: 524766      Links: 1
Access: (0644/-rw-r--r--) Uid: ( 1000/      pi)    Gid: ( 1000/      pi)
Access: 2019-03-19 19:45:05.459401732 +0700
Modify: 2019-03-19 19:45:05.459401732 +0700
Change: 2019-03-19 19:45:05.459401732 +0700
Birth: -
```

ผู้เขียนอธิบายผลลัพธ์ที่ได้ตามลำดับดังนี้

- ชื่อ stat_asm.txt
- ขนาด 341 ไบท์ ใช้พื้นที่จำนวน 8 Blocks เป็นไฟล์ธรรมดा (regular File)
- มีหมายเลข Device = b307h/45831d หรือเท่ากับ b307₁₆/45831₁₀
- มีหมายเลข Inode ที่ 524766 จำนวน 1 Links
- สิทธิ์เข้าถึง (Access) ด้วยรหัส 0644 หรือ 011₂:100₂:100₂ โดยผู้ใช้หมายเลข Uid (User ID)=1000 ชื่อผู้ใช้ (Username)=pi ในกรุ๊ปหมายเลข Groupid=1000 ชื่อกรุ๊ป pi
- เข้าถึง (Access) ณ วันที่ 19 มีนาคม 2019 เวลา 19.45.05
- เปลี่ยนแปลง (Modify) ณ วันที่ 19 มีนาคม 2019 เวลา 19.45.05
- เวลาที่ Inode เปลี่ยนแปลง (Change) ณ วันที่ 19 มีนาคม 2019 เวลา 19.45.05

L.3 อุปกรณ์อินพุตและเอาท์พุตในระบบไฟล์

การทดลองในหัวข้อนี้จะเข้มต่อกับเนื้อหาในบทที่ 3 และ การทดลองที่ 4 ภาคผนวก D หลักการของระบบปฏิบัติการ Unix คือ การมาท์ (Mount) อุปกรณ์กับไฟล์เดอร์ด้วยระบบไฟล์ (File System) ที่แตกต่างกัน โดยใช้ชื่อไฟล์เดอร์ที่แตกต่างกัน โดยมีรูทไดเรกทอรีหรือไฟล์เดอร์ (Root Directory) เป็นตำแหน่งเริ่มต้น ผู้อ่านสามารถพิมพ์คำสั่งใน Terminal

```
$ mount
```

คำสั่งนี้จะแสดงรายชื่อการมาท์ หรือ ผูกยึด อุปกรณ์อินพุตเอาท์พุต เข้ากับไฟล์เดอร์ของระบบปฏิบัติการ ตัวอย่างผลลัพธ์และคำอธิบายต่อไปนี้

- /dev/mmcblk0p7 on / type ext4 (rw,noatime,data=ordered)
- devtmpfs on /dev type devtmpfs (rw,relatime,size=470116k,nr_inodes=117529,mode=755)
- sysfs on /sys type sysfs (rw,nosuid,nodev,noexec,relatime)
- proc on /proc type proc (rw,relatime)
- tmpfs on /dev/shm type tmpfs (rw,nosuid,nodev)
- devpts on /dev/pts type devpts (rw,nosuid,noexec,relatime,gid=5,mode=620,ptmxmode=000)
- ...

โดยมีชนิด (type) หรือระบบไฟล์ที่แตกต่างกัน เช่น

- ชนิด ext4 ซึ่งเป็นระบบไฟล์หลักของลีนุกซ์ ย่อมาจากคำว่า Fourth Extended File System เป็นเวอร์ชันที่ 4 พัฒนาจากชนิด ext3 [รายละเอียดเพิ่มเติมที่ wikipedia](#)
- ชนิด sysfs เป็นระบบไฟล์เสมือน (Virtual File System) [รายละเอียดเพิ่มเติมที่ wikipedia](#)
- ชนิด devfs เป็นระบบไฟล์เสมือน (Virtual File System) สำหรับอุปกรณ์อินพุตเอาท์พุตต่างๆ [รายละเอียดเพิ่มเติมที่ wikipedia](#)
- ชนิด tempfs ย่อมาจากคำว่า Temporary File System [รายละเอียดเพิ่มเติมที่ wikipedia](#)
- ชนิด proc เป็นระบบไฟล์เสมือน (Virtual File System) สำหรับระบบสำคัญต่างๆ เช่น CPU, โดยจะสร้างขึ้นเมื่อบูทเครื่อง และลบทิ้งเมื่อชี้ขาดว่าระบบ [รายละเอียดเพิ่มเติมที่ wikipedia](#)

รายชื่อต่อไปนี้ คือ ตัวเลือกคุณสมบัติ (Attribute) ที่สำคัญของระบบไฟล์ เช่น

- rw : read/write สามารถอ่านและเขียนได้
- noatime และ atime: No/ Access Time หมายถึง ไม่มี/มีการบันทึกเวลาในการสร้าง อ่านหรือเขียนไฟล์ทุกครั้ง

- relatime หมายถึง มีการบันทึกเวลาในการสร้าง อ่านหรือเขียนไฟล์ เมื่อเกิดการแก้ไขไฟล์ หรือ การอ่านหรือเข้าถึงไฟล์มากกว่าเวลาที่บันทึกไว้ก่อนหน้าอย่างน้อย 24 ชั่วโมง
- nosuid: No SuperUser ID เป็นการป้องกันไม่ให้ผู้ดูแลระบบ (SuperUser) กระทำการใดๆ ได้ เพื่อความมั่นคงปลอดภัย
- noexec: No Execution เพื่อตั้งค่าไม่ให้รันไฟล์ที่อยู่ในโฟลเดอร์นี้ได้ เช่น ไฟล์ที่เป็นไวรัสหรือมัลแวร์ (Malware) ที่แอบแฟ้มเข้ามา
- nodev: No Device หมายถึง การไม่อนุญาตให้สร้างหรืออ่านโนนด (Node) ซึ่งเป็นไฟล์ชนิดพิเศษ
- mode หมายถึง Group 3 บิต คือ บิทควบคุม Read Write Execute รวมทั้งหมด 9 บิต

ผู้อ่านสามารถแสดงรายชื่อไดร์กอทรีหรือไฟลเดอร์หรือชื่ออุปกรณ์ภายใต้ไฟลเดอร์ /dev โดยพิมพ์คำสั่งบนโปรแกรม Terminal

```
$ ls /dev
```

ผู้อ่านจะเห็นผลลัพธ์ที่ได้ทั้งหมดซึ่งมีจำนวนมากพอสมควร แต่ผู้เขียนได้พิมพ์ชื่ออุปกรณ์ที่สำคัญๆ ด้วยตัวหนา เพื่อให้ผู้อ่านมองเห็นชัดว่า **mmcblk0p7**, มีอยู่จริงและระบบได้ทำการเมท์เข้ากับไฟลเดอร์รูท (Root) นั่นคือ ไฟลเดอร์ / ด้วยชนิด ext4 ตามที่ได้แสดงในคำสั่งก่อนหน้าแล้ว

```
autofs block btrfs-control bus cachefiles char console cpu_dma_latency cuse disk
fb0 fd full fuse gpiochip0 gpiochip1 gpiochip2 spiomem, hidraw0 hidraw1 hwrng initctl
input kmsg log loop0 loop1 loop2 loop3 loop4 loop5 loop6 loop7 loop-control map-
per mem memory_bandwidth mmcblk0, mmcblk0p1 mmcblk0p2 mmcblk0p5 mmcblk0p6
mmcblk0p7 mqueue, net network_latency network_throughput null ppp ptmx pts ram0
ram1 ram10 ram11 ram12 ram13 ram14 ram15 ram2 ram3 ram4 ram5 ram6 ram7 ram8
ram9 random raw rfkill serial1 shm, snd stderr stdin stdout tty tty0 tty1 tty10 tty11 tty12
tty13 tty14 tty15 tty16 tty17 tty18 tty19 tty2 tty20 tty21 tty22 tty23 tty24 tty25 tty26 tty27
tty28 tty29 tty3 tty30 tty31 tty32 tty33 tty34 tty35 tty36 tty37 tty38 tty39 tty40 tty41
tty42 tty43 tty44 tty45 tty46 tty47 tty48 tty49 tty5 tty50 tty51 tty52 tty53 tty54 tty55 tty56
tty57 tty58 tty59 tty6 tty60 tty61 tty62 tty63 tty7 tty8 tty9 ttyAMA0 ttyprintk uhid uinput
urandom vchiq vcio vc-mem vcs vcs1 vcs2 vcs3 vcs4 vcs5 vcs6 vcs7 vcsa vcsa1 vcsa2 vcsa3
vcsa4 vcsa5 vcsa6 vcsa7 vcsm vhci watchdog watchdog0 zero
```

นอกจากนี้ อุปกรณ์สำคัญอื่นๆ เช่น stdin (standard input) stdout (standard output) และ stderr (standard error) นั้นเกี่ยวข้องกับโปรแกรม Terminal ซึ่งเชื่อมโยงกับประโยชน์ในภาษา C ในการทดลองที่ 5 ภาคผนวก E

```
#include <stdio.h>
```

เพลย์คลิป ที่โปรแกรมตัดต่อ เยี่ยงๆ กับกับชุดในภาษา C

แบบเรียบง่าย = 317

ต่างกันอย่างไรกัน

ภาษา C ภาษา C++

L.4 កិច្ចរម្យទាមការទាញ

1. ចងចាំថាគារទាញប្រព័ន្ធឌីជីថល ត្រូវបានដោះស្រាយជាផ្លូវការ ដើម្បីបង្កើតការណ៍ដែលបានបង្កើតឡើង និងបង្កើតឡើងនៃការណ៍ដែលបានបង្កើតឡើង។ ការណ៍ដែលបានបង្កើតឡើងនេះ ត្រូវបានបង្កើតឡើងជាផ្លូវការ ដើម្បីបង្កើតការណ៍ដែលបានបង្កើតឡើង។
2. សរាប់ផែនធានាអំពីការបង្កើតឡើងនៃការណ៍ដែលបានបង្កើតឡើង។
3. សរាប់ផែនធានាអំពីការបង្កើតឡើងនៃការណ៍ដែលបានបង្កើតឡើង។
4. សរាប់ផែនធានាអំពីការបង្កើតឡើងនៃការណ៍ដែលបានបង្កើតឡើង។
5. សរាប់ផែនធានាអំពីការបង្កើតឡើងនៃការណ៍ដែលបានបង្កើតឡើង។
6. សរាប់ផែនធានាអំពីការបង្កើតឡើងនៃការណ៍ដែលបានបង្កើតឡើង។
7. សរាប់ផែនធានាអំពីការបង្កើតឡើងនៃការណ៍ដែលបានបង្កើតឡើង។
8. សរាប់ផែនធានាអំពីការបង្កើតឡើងនៃការណ៍ដែលបានបង្កើតឡើង។
9. សរាប់ផែនធានាអំពីការបង្កើតឡើងនៃការណ៍ដែលបានបង្កើតឡើង។
10. សរាប់ផែនធានាអំពីការបង្កើតឡើងនៃការណ៍ដែលបានបង្កើតឡើង។
11. សរាប់ផែនធានាអំពីការបង្កើតឡើងនៃការណ៍ដែលបានបង្កើតឡើង។

Bibliography

- Allwinner Technology, C. (2015a). Allwinner a64 mobile application processor datasheet version 1.1.
- Allwinner Technology, C. (2015b). Allwinner a64 user manual version 1.0.
- Broadcom, C. (2012). Bcm2835 arm peripherals.
- Choi, K. (2010). Nand flash memory.
- Clements, A. (2013). *Computer Organization and Architecture: Themes and Variations* (1 ed.). Boston, USA: Cengage Learning.
- Corporation, C. S. (2017). Single-chip ieee 802.11ac b/g/n mac/baseband/radio with integrated bluetooth 4.1 and fm receiver.
- Cypress Semiconductor, C. (2002). 256k(32kx8) static ram cy62256.
- Demblon, S. and S. Spitzner (2004). *Linux Internals: (to the power of -1)* (1 ed.). The Shuttleworth Foundation.
- Diodes, I. (2012). Pam2306 dual high-efficiency pwm step-down dc-dc converter.
- Harris, D. and S. Harris (2013). *Digital Design and Computer Architecture* (1st ed.). USA: Morgan Kauffman Publishing.
- Ltd, R. P. T. (2019). *Raspberry Pi Compute Module 3+ and Raspberry Pi Compute Module 3+ Lite* (1 ed.). Raspberry Pi (Trading) Ltd.
- Microchip Technology, I. (2009). Lan9514/lan9514i usb 2.0 hub and 10/100 ethernet controller.
- Micron Technology, I. (2004). Nand flash memory: Mt29f2g08aabwp/mt29f2g16aabwp/mt29f4g08babwp/mt29f4g16babwp/mt29f8g08fabwp.

Bibliography

- Micron Technology, I. (2014). Embedded lpddr2 sdram edb8132b4pb-8d-f.
- Patterson, D. and J. Hennessy (2000). *Computer Organization: Design Approach* (4st ed.). USA: Morgan Kauffman Publishing.
- SanDiskCorporation (2003). Sandisk secure digital card, product manual version 1.9 document no. 80-13-00169.

[index]

Index

A

Audio

PCM 147-149, 158-159, 278

ALSA 278, 279

alsamixer 279

Analog to Digital, 147, 149, 215

ASCII 41-43, 80, 209-210, 257

B

Bluetooth, 48-50, 153-155

Branch

Branch and Link

Broadcom, 1

Software Development

Build

Compile

Make

Makefile 242-245

Cache 78, 110, 114, 116-120, 281

Direct Map 117-119

Set Associative 117, 120

Fully Associative 133

Tag 116

Dirty

Valid

char 8-10, 41-44, 209

unsigned 8-10, 44

Character

UTF

Composite Video, 1

CSI

D

DMA

DSI

Digital to Analog, 1

DSI

E

Executable

ELF 61-62, 68-73, 243

F

File

File System 174-176, 307, 310

Logical

Virtual

Physical

Fixed Point 27

Fraction 27

Floating Point 28, 35, 36

Exponent 28, 32, 204, 208

true 32

bias 32-34

Fraction 32

Mantissa

Significand 28, 32

Overflow

Index

Underflow	unsigned int
Function Call	Integer 7-8, 10, 13, 17, 27
Parameter	Signed 10
Return	Unsigned 10, 13
Debugging	2-Complement 13
GDB	Overflow
Gid	Sign-Magnitude 17
Git	Integer Overflow
GUIDPT	Unsigned
GPIO	Signed
GPU	IDE
H	Internet
Heat Sink	Internet Protocol
HDMI	IP Address
Loop	Interrupt 53, 111, 157, 165-167, 282
While	Hardware
Do-While	Software
For	Request 165, 299
Nested	Service Routine 167, 299
Hard Disk Drive	Input/Output 50, 52, 137
Cylinder	Memory Mapped 114, 156
Head	Polling 148
Platter	Java 9, 31, 57, 68, 105, 106, 197
Sector	Jazelle
I	L
IEEE 754, 31-45, 197, 208	Link 73
Single Precision, 31-40, 64, 70, 203-205, 208	M
Double Precision, 31-35, 64, 197, 204-207	Makefile
Inode, 176-179, 308-310	Memory
Instruction	Main Memory
Instruction Set	Secondary Memory
int	Virtual Memory
long int	Neon
long long int	Nested Loop
O	O
	Operationg System

Unix	Pointer
Linux	Top of Stack
Raspbian	Segment
Format	shutdown
Object File	61, 70-73, 102
Pipeline	36, 78-79
Integer	78-79
Floating Point	36, 78
Load-Store	78
Power Supply	170
Pointer	
Permission	
Read	SuperUser 237
Write	su
Execute	sudo 226, 237, 283, 285, 289, 290, 293
Q	V
Quotient	W
Register	WiFi 153-155, 225-226, 284-286
Link Return	Key Management
Program Counter	Pre-Shared Key
Stack Pointer	WPA
R	WEP
Main Memory	WiringPi 289-290, 293-296, 300-303
RAM (Random Access Memory)	X
Static	X
Dynamic	Y
Synchronous	Z
Refresh	ก
ROM (Read Only Memory)	ข
Flash ROM	ค
NAND Flash	คอมพิวเตอร์, 1
NOR Flash	คอมพิวเตอร์ตั้งโต๊ะ 1
S	คอมพิวเตอร์แม่ข่าย 1
Solid State Drive	คอมพิวเตอร์พกพา 2
Stack	คอมพิวเตอร์ฝังตัว 2
Frame	ง

Index

ๆ

ฉบับ

ชีว

ชิพ 3, 4

ไอซี 3

วงจรรวม 3

ไมโครชิพ 3

ชิสเต็มออนไลน์ชิพ 3, 48, 50, 283

ๆ

ร

ระบบปฏิบัติการ

ยูนิกซ์

ลินิกซ์

ราสเบียน

โพรเซสเซอร์ 4, 7, 8, 61, 110, 117

ไมโครคอนโทรลเลอร์

ไมโครโพรเซสเซอร์

ทرانซิสเตอร์

ฯ

ฯ

ฯ

ฯ