

องค์ประกอบของเครื่องคอมพิวเตอร์และภาษาแอสเซมบลี:

บอร์ด RaspberryPi3

Computer Organization and Assembly Language:

RaspberryPi3 Board

ผศ.ดร.สุรินทร์ กิตติธรภุก‡

ผศ.ดร.ชัยวัฒน์ หนูทอง†

2 สิงหาคม 2561 ‡

* ภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

† วิทยาลัยนานาชาติ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

‡ First drafted on 2 สิงหาคม 2559

Contents

Contents	i
List of Tables	ii
List of Figures	iii
1 บทนำ	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 ขั้นตอนการผลิตไมโครโปรเซสเซอร์	3
1.4 ไมโครโปรเซสเซอร์ชนิดมัลติคอร์	4
2 ข้อมูลและคณิตศาสตร์ในคอมพิวเตอร์	7
2.1 ตัวแปรชนิดต่างในภาษา C/C++	8
2.2 เลขฐานสองชนิดจำนวนเต็ม	10
2.2.1 ชนิดไม่มีเครื่องหมาย (Unsigned Integer)	10
2.2.2 ชนิดมีเครื่องหมาย (Signed Integer) แบบ 2-Complement	13
2.2.3 ชนิดมีเครื่องหมาย (Signed Integer) แบบ Sign-Magnitude	17
2.3 คณิตศาสตร์เลขจำนวนเต็ม	19
2.3.1 ชนิดไม่มีเครื่องหมาย	19
2.3.2 ชนิดมีเครื่องหมายแบบ 2-Complement	24
2.4 เลขทศนิยมชนิด Fixed Point	26
2.5 เลขทศนิยมชนิด Floating Point	28
2.6 การบวกเลข Floating-Point	29
2.7 การคูณเลข Floating-Point	30
2.8 เลขทศนิยมชนิดจุดลอยตัวมาตรฐาน IEEE 754	32

2.9 รูปแบบของเลข IEEE 754	32
2.10 ค่าสูงสุด ต่ำสุดและค่าอื่นๆ ของเลข Floating Point	35
2.11 การบวกเลข Floating-Point ตามมาตรฐาน IEEE 754	36
2.12 การคูณเลข Floating-Point ตามมาตรฐาน IEEE 754	40
2.13 ตัวอักษร (Alphabet)	42
2.14 สรุป	44
3 Computer Hardware & Software	45
3.1 ฮาร์ดแวร์ของเครื่องคอมพิวเตอร์	45
3.1.1 หน่วยประมวลผลกลาง (CPU: Central Processing Unit)	47
3.1.2 หน่วยความจำหลัก (Main Memory)	49
3.1.3 อินพุท/เอาท์พุท (Input/Output)	50
3.1.4 หน่วยสำรองข้อมูล (Data Storage)	54
3.2 ซอฟต์แวร์หรือโปรแกรมคอมพิวเตอร์	55
3.2.1 การบูรณะระบบปฏิบัติการจากหน่วยสำรองข้อมูลเข้าสู่หน่วยความจำหลัก	56
3.2.2 การโหลดซอฟต์แวร์ประยุกต์จากไฟล์ ELF เข้าสู่หน่วยความจำหลัก	59
3.2.3 การอ่านคำสั่งของซอฟต์แวร์ประยุกต์จากหน่วยความจำหลักเพื่อไปปฏิบัติตาม	60
3.2.4 ซอฟต์แวร์ประยุกต์อ่าน/เขียนข้อมูลในหน่วยความจำเพื่อให้ซีพียูประมวลผล	62
3.2.5 การใช้งานอินพุทและเอาท์พุทต่างๆ	62
3.2.6 ซอฟต์แวร์ประยุกต์เขียนข้อมูลบันทึกในหน่วยสำรองข้อมูล	64
3.2.7 การชัตดาวน์ (Shut Down) ระบบปฏิบัติการก่อนหยุดจ่ายไฟ	65
3.3 โครงสร้างของซอฟต์โค้ด (Source Code) โปรแกรมคอมพิวเตอร์	66
3.3.1 โครงสร้างของซอฟต์โค้ดโปรแกรมภาษา C/C++	66
3.3.2 การคอมไพล์โปรแกรมภาษา C/C++	66
3.3.3 โครงสร้างของซอฟต์โค้ดโปรแกรมภาษา Assembly	69
3.3.4 ลิงค์เกอร์ (Linker)	71
3.3.5 โครงสร้างของคำสั่งภาษาเครื่อง (Machine Code)	71
4 ภาษา Assembly เวอร์ชัน 32 บิตของ ARM	75
4.1 โครงสร้างของซีพียู ARM Cortex A53	75
4.2 สถาปัตยกรรมชุดคำสั่ง (Instruction Set Architecture)	77
4.3 ตัวอย่างคำสั่งภาษาเครื่องในหน่วยความจำ	78
4.4 การประกاثและตั้งค่าตัวแปรในหน่วยความจำหลัก	80
4.5 คำสั่งถ่ายโอนข้อมูลระหว่างหน่วยความจำและรีจิสเตอร์	80
4.6 คำสั่งประมวลผลข้อมูลในรีจิสเตอร์ (Register Data Processing Instructions)	82
4.6.1 คำสั่งทางคณิตศาสตร์	82
4.6.2 คำสั่งเลื่อนบิตข้อมูล	83

4.6.3	คำสั่งคณิตศาสตร์และเลื่อนบิต	83
4.6.4	คำสั่งทางตรรกศาสตร์	84
4.7	คำสั่งควบคุมการทำงาน (Control Instructions)	84
4.7.1	การตัดสินใจ IF	86
4.7.2	การตัดสินใจ IF-ELSE	88
4.7.3	การวนรอบชนิด FOR	90
4.7.4	การวนรอบ WHILE	92
4.7.5	การวนรอบ DO-WHILE	93
4.7.6	การวนรอบชนิด FOR จำนวน 2 ชั้น	94
4.8	การเรียกใช้ฟังก์ชัน (Function Call)	95
4.9	อุปกรณ์และวิวัฒนาการของชุดคำสั่ง ARM	97
4.9.1	อุปกรณ์คอมพิวเตอร์ที่ใช้ชิปยู ARM	97
4.9.2	วิวัฒนาการของชุดคำสั่ง ARM	98
5	ลำดับชั้นหน่วยความจำ (Memory Hierarchy)	101
5.1	การจัดวางหน่วยความจำชนิดต่างๆ ในเครื่องคอมพิวเตอร์	101
5.2	หน่วยความจำสแตติคแรม	104
5.2.1	โครงสร้างภายในและบิตเซลของหน่วยความจำสแตติค	105
5.2.2	การทำงานของสแตติคแรม: อ่านและเขียน	106
5.3	หน่วยความจำไดนามิกแรม (Dynamic RAM: DRAM)	108
5.3.1	โครงสร้างภายในของชิป DRAM	110
5.3.2	การทำงานของไดนามิกแรม: อ่านและเขียน	112
5.3.3	การรีเฟรชข้อมูล	113
5.4	หลักการพื้นฐานของหน่วยความจำแคช (Cache)	115
5.4.1	แคชชนิดไดเร็คเมป (Direct Map Cache)	116
5.4.2	แคชชนิดเซ็ตแอสโซซิเอทีฟ (Set Associative Cache)	118
5.5	หลักการหน่วยความจำเสมือนชนิดเพจ (Paging Virtual Memory)	120
5.6	หน่วยความจำเสมือนของ Raspberry Pi3	124
5.7	สรุป	125
6	อินพุท/เอาท์พุท (Input/Output)	127
6.1	สัญญาณ HDMI สำหรับจอ LCD ขนาดใหญ่	129
6.2	สัญญาณ DSI สำหรับจอ LCD ขนาดเล็ก	131
6.3	สัญญาณ CSI สำหรับเชื่อมต่อกล้องขนาดเล็ก	134
6.4	สัญญาณ PCM สำหรับสัญญาณเสียง	136
6.5	สัญญาณภาพและเสียงสำหรับจอทีวี	138
6.6	สัญญาณ USB 2.0 สำหรับอุปกรณ์ต่อพ่วงต่างๆ	139

Contents

6.7	สัญญาณ Ethernet สำหรับสายเชื่อมต่อกับเครือข่ายอินเตอร์เน็ต	141
6.8	สัญญาณ WiFi และ Bluetooth สำหรับการสื่อสารไร้สาย	142
6.9	หลักการ Memory Mapped Input/Output	144
6.10	หัวเชื่อมต่อ 40 ขา (40-Pin Header)	146
6.11	ขา GPIO (General Purpose Input Output)	148
6.12	การขัดจังหวะ (Interrupt)	152
6.13	การเข้าถึงหน่วยความจำโดยตรง (Direct Memory Access)	155
6.14	ขา UART (Universal Asynchronous Receiver Trasmitter)	157
6.15	ขา SPI (Serial Peripheral Interface)	159
6.16	ขา I2C ((Inter-Integrated Circuit)	161
6.17	พัลซ์วิดจ์มอดูลีฟชัน (Pulse Width Modulation)	162
6.18	แหล่งจ่ายไฟ (Power Supply) ของบอร์ด Pi3	163
6.19	สรุป	164
7	อุปกรณ์สำรองข้อมูล (Data Storage Devices)	165
7.1	ชิปหน่วยความจำชนิดแฟลช (Flash Memory Chip)	165
7.2	การ์ดหน่วยความจำ SD (Secure Digital)	170
7.3	โซลิดสเตทไดสก์ (Solid-State disk)	172
7.4	ฮาร์ดไดสก์ (Hard disk)	174
7.5	สรุป	176
	Bibliography	177
A	การทดลองที่ 1 ข้อมูลและคณิตศาสตร์ในคอมพิวเตอร์	179
A.1	การแปลงและคณิตศาสตร์สำหรับ Integer	179
A.1.1	การทดลอง	179
A.1.2	Assignment	182
A.2	การแปลงและคณิตศาสตร์สำหรับ Floating-Point IEEE754	183
A.2.1	การทดลองสำหรับ Single-Precision	183
A.2.2	การทดลองสำหรับ Double-Precision	186
A.2.3	Assignment	188
A.3	รหัสของข้อมูลอักขระ	189
A.3.1	การทดลอง	189
A.3.2	Assignment	190
B	การทดลองที่ 2 การติดตั้งและใช้งานฮาร์ดแวร์	191
B.1	Hardware List	191
B.2	Board Assembly	192

B.3 Raspberry Pi 3 Model B Personal Computer	195
C การทดลองที่ 3 การติดตั้งระบบปฏิบัติการ Raspbian	199
C.1 Format the MicroSD Card	199
C.2 NOOBS Installation	200
C.3 Installing an Operating System on the Raspberry Pi	202
C.4 Setting up Wi-Fi via Graphical Interface	204
C.5 Board Configuration	205
D การทดลองที่ 4 การใช้งานระบบปฏิบัติการยูนิกซ์เบื้องต้น	207
D.1 CPU Information	207
D.2 User Accounts & Permissions	208
D.3 Partitions & File System	209
D.3.1 Partitions	209
D.3.2 File System	209
D.4 Folder	210
D.5 Useful Applications	211
D.6 Log out vs. Shutdown	211
E การทดลองที่ 5 การพัฒนาโปรแกรมด้วยภาษาซี	213
E.0.1 Assembly Programming for ARM	213
E.0.2 The ARM architecture	213
E.0.3 Assembler Language	214
E.0.4 Arithmetic-Logic Instructions	214
E.0.5 Branch Instructions	215
E.0.6 Load/Store Instruction	216
E.1 Integrated Development Environment	217
E.2 Makefile	220
E.2.1 Rule of Makefiles	220
E.2.2 A Simple Makefile (For C Language)	220
E.3 Debugging	222
E.4 Assignments	222
F การทดลองที่ 6 การพัฒนาโปรแกรมด้วยภาษาแอสเซมบลี	225
F.1 Build and Debug Assembly Program	225
F.2 First Assembly Program	231
F.3 Creating Makefiles	232
F.3.1 Creating Your Own Makefile	232

F.3.2 A GDB Debugger	234
F.4 C with Assembly Program and Interrupt	234
F.4.1 C Programming with Assembly Code	234
F.4.2 Summing Two Numbers	234
F.4.3 Summing Array	235
F.4.4 Assignments	236
F.4.5 Assignments	236
G การทดลองที่ 7 การศึกษาและปรับแก้ในพุทธ/AIoT พุทธต่างๆ	237
G.0.1 HDMI	238
G.0.2 USB	238
G.0.3 WiFi & Bluetooth	238
G.0.4 Ethernet	239
G.0.5 SD Memory Card	239
H การทดลองที่ 8 การเชื่อมต่อกับ GPIO	241
H.1 Hardware Setup for Blink LED	241
H.2 WiringPi Library	244
H.2.1 Download and Install	245
H.3 Blink LED with Assembly	246
H.4 Blink LED with C	248
H.5 Assignments	249
I การทดลองที่ 9 การเชื่อมต่อกับอินเทอร์รัพท์	251
I.1 Interrupts	251
I.2 การจัดการอินเตอร์รัพท์ (Interrupt Handling)	252
I.2.1 Interrupt with WiringPi	252
I.3 Hardware Setup: Push Button with GPIO	253
I.4 Software Setup	254
I.5 Assignments	255
J การทดลองที่ 10 การเชื่อมต่อผ่าน UART	257
J.1	257
K การทดลองที่ 11 การเชื่อมต่อด้วย SPI	259
K.1 SPI	259
L การทดลองที่ 12 การควบคุมกำลังไฟฟ้าด้วยสัญญาณ PWM	263

List of Tables

2.1	ชนิดตัวแปร จำนวนบิต ค่าต่ำสุด และค่าสูงสุดของตัวแปรแต่ละชนิดในภาษา C/C++	8
2.2	การแปลงค่าฐานสิบเป็นเลขฐานสองแบบไม่มีเครื่องหมายความยawa $n=8$ บิต ด้วยการหาร	12
2.3	การแปลงค่าฐานสิบเป็นเลขฐานสองแบบไม่มีเครื่องหมายความยawa $n=8$ บิต	12
2.4	รูปแบบ (Pattern) ของเลขฐานสองขนาด $n=4$ บิตทั้งหมด $2^4=16$ แบบและการตีความหมายให้เป็นค่าฐานสิบแบบมีเครื่องหมายและแบบไม่มีเครื่องหมาย	14
2.5	ค่าฐานสิบแบบมีเครื่องหมายและแบบไม่มีเครื่องหมายของเลขฐานสองความyawa $n=5$ บิตทั้งหมด $2^5=32$ แบบ	15
2.6	ค่าฐานสิบชนิดมีเครื่องหมายและไม่มีเครื่องหมายของเลขฐานสองความyawa $n=8$ บิตทั้งหมด $2^8=256$ แบบ	16
2.7	การแปลงค่าฐานสิบแบบมีเครื่องหมายให้เป็นเลขฐานสองความyawa $n=4$ บิต โดยแปลงให้เป็น ค่าฐานสิบโดยใช้สูตร $2^n + X_{10,s}$ กรณีที่ $X_{10,s} < 0$ ตามสมการที่ 2.32	17
2.8	ค่าฐานสิบแบบมีเครื่องหมาย Sign-Magnitude, แบบ 2-Complement, และแบบไม่มีเครื่องหมาย (Unsigned) ของเลขฐานสองความyawa $n=4$ บิตทั้งหมด $2^4=16$ แบบ	18
2.9	ตารางสรุปความแตกต่างระหว่างเลข Floating-Point ตามมาตรฐาน IEEE 754 ชนิด Single Precision (x หมายถึง ข้อมูล '0' หรือ '1')	35
2.10	ชนิด ความyawa ข้อมูล และการประยุกต์ใช้งานเลขฐานสองชนิดต่างๆ ในคอมพิวเตอร์	44
3.1	ตารางสรุปข้อมูลด้านอาร์ดแวร์และซอฟต์แวร์ของบอร์ด Pi3 โนเดล B และลิงค์เขื่อมไปยังหัวข้อที่แสดงรายละเอียด	48
3.2	ระดับการรัน (Run Level), โหมดการทำงานของระบบ และชื่อไฟล์เดอร์ที่บรรจุไฟล์โปรแกรมและไฟล์ข้อมูลสำหรับการบูท	57
3.3	ตัวอย่างโปรแกรมภาษาแอสเซมบลีเพื่อสร้างตัวแปรขนาด 32 บิต จำนวน 2 ตัวแปร	69
4.1	ตัวอย่างโปรแกรมภาษาแอสเซมบลีเพื่อสร้างตัวแปรขนาด 32 บิต จำนวน 2 ตัวแปร	80
4.2	ตัวอย่างโปรแกรมภาษาแอสเซมบลีเพื่ออ่านค่าตัวแปรจากหน่วยความจำ โดยการอ่านตำแหน่งของตัวแปร	81
4.3	ตัวอย่างโปรแกรมภาษาแอสเซมบลีเพื่อคำนวณประโยชน์ $x = (a + b) - c$	81
4.4	ตัวอย่างโปรแกรมตามประโยชน์คเงื่อนไข if	87
4.5	ตัวอย่างโปรแกรมตามประโยชน์คเงื่อนไข if-else	89

List of Tables

4.6 ตัวอย่างโปรแกรมตามประโยคุนรอบ For	91
4.7 ตัวอย่างโปรแกรมตามประโยคุนรอบ While	93
4.8 ตัวอย่างโปรแกรมตามประโยคุนรอบ Do-While	94
4.9 ตัวอย่างโปรแกรมเรียกใช้ฟังก์ชันด้วยคำสั่ง BL และ BX	96
4.10 ส่วนแบ่งการตลาดของบริษัท ARM ในปี ค.ศ.2010 ที่มา: https://www.zdnet.com/article/arm-holdings-2015-plan-grab-pc-server-share/	98
5.1 ตัวอย่างโปรแกรมภาษาแอสเซมบลีเพื่อประกอบการทำงานของแอดร์รัส (PA: Physical Address)	116
6.1 หมายเลขขา ซีอี และวัตถุประสงค์ของคอนเนคเตอร์ชานนิค HDMI เวอร์ชัน 1.4	131
6.2 หมายเลข และหน้าที่ของสายสัญญาณ DSI สำหรับจอ LCD ขนาดเล็ก	133
6.3 หมายเลข ซีอี และวัตถุประสงค์ของคอนเนคเตอร์ชานนิค CSI	135
6.4 หมายเลข ซีอี 1 และซีอี 2 ของหัวเชื่อมต่อสายทั้ง 40 ขา (GND: Ground) ที่มา: ?	147
6.5 ตารางแอดเดรสในหน่วยความจำเริ่มต้นที่หมายเลข 0xE20 0000 สำหรับ GPIO	150
6.6 ตารางเลือกค่า V_{OUT} และค่าตัวหนேี่ยวนำ L ที่มา: Diodes (2012)	164
7.1 ตารางเปรียบเทียบเซลล์หน่วยความจำ Flash ชนิด NAND (ซ้าย) และ NOR (ขวา) เชิงโครงสร้าง	168
7.2 หมายเลข ซีอี และวัตถุประสงค์ของ SD ในโหมดการทำงานแบบ SDIO	170
7.3 เปรียบเทียบประสิทธิภาพของหน่วยสำรองข้อมูลชนิดต่างๆ	176

List of Figures

1.1	ตัวอย่างเครื่องเซิร์ฟเวอร์ของ HP รุ่น Proliant ที่มา: https://www.datacenterknowledge.com/archives/2009/06/10/hp-scales-out-with-new-cloud-servers	2
1.2	ยอดขายเครื่องคอมพิวเตอร์ชนิดต่างๆทั่วโลกในปี 2005-2015 ที่มา: https://www.slideshare.net/kleinerperkins/2012-kpcb-internet-trends-yearend-update/25-Global_Smartphone_Tablet_Shipments_Exceeded	3
1.3	ขั้นตอนการผลิต (Process) ของชิป (Chip) ไมโครโปรเซสเซอร์และอื่นๆ ที่มา: https://slideplayer.com/slide/5218297/	4
1.4	ภาพถ่ายダイ (Die) โครงสร้างและรายละเอียดของชิป BCM 2835 บนบอร์ด Raspberry Pi ที่มา: https://www.raspberrypi.org/forums/viewtopic.php?t=63750	5
2.1	เลขจำนวนเต็มและเลขจำนวนจริงบนเส้นจำนวนในคณิตศาสตร์ ที่มา: https://math.tutorvista.com/number-system/real-number-line.html	8
2.2	การประกاثตัวแปรและตัวค่าเริ่มต้น (ที่มา: Harris and Harris (2013))	9
2.3	พื้นที่ในหน่วยความจำซึ่งบรรจุค่าเริ่มต้นของตัวแปรที่ได้ประกาศในรูปที่ 2.2 (ที่มา: Harris and Harris (2013))	9
2.4	เส้นจำนวนเปรียบเทียบข้อมูลขนาด 8 บิตในรูปแบบของเลขจำนวนเต็มชนิด Unsigned แบบ 2-Complement ที่มา: https://stackoverflow.com/questions/27527943/how-does-adding-min-value-compare-integers-as-unsigned	10
2.5	สัญลักษณ์ ALU (Arithmetic Logic Unit) สำหรับเลขจำนวนเต็มขนาด 32 บิต	19
2.6	วงจรคูณเลขขนาด $n = 32$ บิต ชนิดที่ 1 โดยใช้ ALU ขนาด $2n = 64$ บิต และรีจิสเตอร์ตัวตั้งขนาด $2n = 64$ บิต (ที่มา: Patterson and Hennessy (2000))	21
2.7	วงจรคูณเลขขนาด $n = 32$ บิต ชนิดที่ 2 โดยใช้ ALU ขนาด $n = 32$ บิต และรีจิสเตอร์ตัวตั้งขนาด $n = 32$ บิต (ที่มา: Patterson and Hennessy (2000))	22
2.8	ตัวเลขชนิดจำนวนทศนิยมโดยตัวตามมาตรฐาน IEEE 754 Double-Precision และ Single-Precision ที่มา: http://pybugs.com/single-precision-vs-double-precision/	32
2.9	เลขศนิยมโดยตัวชนิดนอมัลໄล์ต์ตั้งแต่ $\pm N_{min}$ ถึง $\pm N_{max}$ และชนิดที่ไม่สามารถเขียนแบบนอมัลໄล์ต์ ตั้งแต่ $\pm D_{min}$ ถึง $\pm D_{max}$	36
2.10	วงจรบวกเลข Floating-Point ตามมาตรฐาน IEEE 754 โดยรับค่าอินพุทจำนวน 2 ตัวด้านบน และเอาท์พุทผลลัพธ์ด้านล่าง (ที่มา: Patterson and Hennessy (2000))	37

List of Figures

2.11 วงศ์คุณเลข Floating-Point ตามมาตรฐาน IEEE 754 โดยรับค่าอินพุทจำนวน 2 ตัวด้านบน และเอาท์พุทผลลัพธ์ด้านล่าง (ที่มา: Patterson and Hennessy (2000))	40
2.12 การเก็บข้อความในหน่วยความจำในรูปของรหัส ASCII ด้วยตัวแปร str ซึ่งเป็นตัวแปรชนิด char[10] str="Hello!" ที่แอ็ดเดรสเริ่มต้น 0x50 (ที่มา: Harris and Harris (2013))	42
2.13 ตารางรหัสแอกซ์ก์ (ASCII) และ Unicode สำหรับตัวอักษรจำนวน $2^8=256$ ตัว ภาษาอังกฤษและภาษาไทย ที่มา: http://ascii-table.com/codepage.php?874	43
3.1 ตำแหน่งของอุปกรณ์ต่างๆ บนบอร์ด Pi3 ที่มา: http://www.raspberryhome.net/product/72	46
3.2 การเชื่อมโยงอุปกรณ์ต่างๆ บนบอร์ด Pi3 โดยมีชิพ BCM2837 เป็นศูนย์กลาง รายละเอียดเพิ่มเติมในบทที่ 6 ที่มา: https://xdevs.com/article/rpi3_oc/	47
3.3 บล็อกโดยรวมของชิพ AllWinner A64 (ที่มา: Allwinner Technology (2015a) และ Allwinner Technology (2015b)) ที่มีโครงสร้างคล้ายกับ BCM2837 บนบอร์ด Raspberry Pi3	49
3.4 หน่วยความจำไดนามิกแรมด้านล่างของบอร์ด Pi3 ไม่เดล B	50
3.5 ตำแหน่งและสแกนโค้ด (Scan Code) ของปุ่มบนคีย์บอร์ดชนิด 106 ปุ่ม ที่มา: http://ps-2.kev009.com/tl/techlib/manuals/adoclib/aixkybd/kybdtech/figures/kybdt3.jpg	51
3.6 โครงสร้างภายในแม่สัมบัติ Optical ประกอบด้วยหลอด LED ส่องแสงกระแทกกับพื้นผิว ด้านล่างแล้วสะท้อนกลับมายังตัวรับแสงชนิด CMOS https://www.pinterest.com/pin/693835886317197305/	52
3.7 โครงสร้างจอแสดงผลชนิด Color LCD ประกอบด้วยแหล่งกำเนิดแสง (Light Source) ปล่อยแสงทะลุชั้นต่างๆ มาแสดงผล ที่มา: https://techterms.com/definition/lcd	53
3.8 โครงสร้างของหน่วยสำรองข้อมูล SD ชนิด SDHC ความจุ 16 GB ประกอบด้วย Flash Memory วงจรควบคุม และวงจรเข้ามูกต่อ ที่มา: https://en.wikipedia.org/wiki/Secure_Digital	54
3.9 โครงสร้างของระบบปฏิบัติการ Linux ซึ่งแบ่งเป็น Kernel Space และ User Space ที่มา: http://www.linux-india.org/components-of-linux-system/	56
3.10 โครงสร้างของโฟลเดอร์ (Folder) หรือไดเรกทอรี (Directory) สำคัญๆ ในระบบปฏิบัติการ Linux ที่มา: https://freedompenguin.com/articles/how-to/learning-the-linux-file-system/	58
3.11 โครงสร้างไฟล์สำหรับเก็บชุดคำสั่งและข้อมูลในหน่วยสำรองข้อมูล ที่มา: https://en.wikipedia.org/wiki/Executable_and_Linkable_Format	59
3.12 โครงสร้างหน่วยความจำเสมือนเมื่อโหลดชุดคำสั่งและข้อมูล ที่มา: https://gabrieletolomei.wordpress.com/miscellanea/operating-systems/in-memory-layout/	60
3.13 ขบวนการอ่านข้อมูลจากหน่วยความจำหลักที่ตำแหน่ง 125 ที่มา: http://www.phatcode.net/res/260/files/html/SystemOrganization2.html	62

3.14 โครงสร้างของลีนุกซ์เคอร์แนล โดยเน้นส่วนที่เป็น I/O, Memory Management และ Process Management (https://en.wikipedia.org/wiki/Completely_Fair_Scheduler)	63
3.15 หลักการของ Memory Map File คือการແມປ່າຍຄວາມຈຳສໍາຮັບອ່ານຫຼືເຂົ້າໄຟລ໌ ດ້ວຍຫຼັກຫຼີ້າ ໂດຍໃຫ້ໂຄງຮັງຂອງໄຟລ໌ໃໝ່ເປີດຕະຫຼາດ ດ້ວຍຫຼັກຫຼີ້າ ທີ່ຈຸກຈອງເພື່ອໃໝ່ພັກຂໍ້ມູນໃນໄຟລ໌ ທີ່ມາ: https://www.safaribooksonline.com/library/view/linux-system-programming/0596009585/ch04s03.html	64
3.16 ຕ້ວອຍ່າງຂອ້ສໂດ້ດໍາການ C ພັກ໌ຂັ້ນຫຼັກ໌ຂໍ້ອ main() ເນື້ອທຳການເສົ້ນຈະຣີເທິຣົນຄ່າ 0	67
3.17 ໂຟລ່ວການພັດນາຂອົບຕົວຈຳກັດສູ່ໄຟລ໌ Executable (ELF) ຮີເປັນໂປຣແກຣມປະຢຸກຕົວ (Application Program) ຮີເປັນໂອງໆວ່າ ແອພ (ທີ່ມາ: https://slideplayer.com/slide/4804417/)	68
3.18 ຕ້ວອຍ່າງກາລິກົດ (Link) ຮີເປັນໄຟລ໌ Object ແລະ Library ເຂົ້າດ້ວຍກັນເປັນໄຟລ໌ໂປຣແກຣມ ຮີເປັນແອພພລິເຄື່ອນ ທີ່ມາ: https://sites.google.com/site/kmrvikash/home/tutorials/c-tutorials/compiler-assembler-linker-and-loader-a-brief-story	72
3.19 ຕ້ວອຍ່າງການແປ່ງຈາກຄໍາສັ່ງແອສເສມບລືຖາງດ້ານຂວາ ເປັນຄໍາສັ່ງການເຄື່ອງທາງດ້ານໜ້າ (ທີ່ມາ: Harris and Harris (2013))	72
 4.1 ໂຄງຮັງຂອງ ARM Cortex A53 ຈຳນວນ 4 ຄອർ ແຕ່ລະຄອນປະກອບດ້ວຍ CPU core ເພື່ອປະມວລຜລຕ້ວເລຂຈຳນວນເຕີມຂາດ 32 ແລະ 64 ບີຕ, Floating Point Unit, NEON SIMD engine, ແລະ L1 I-Cache ແລະ D-Cache ທີ່ມາ: https://www.tomshardware.com/reviews/snapdragon-810-benchmarks_4053-2.html	75
4.2 ໂຄງຮັງຂອງຊື່ປຶ້ງຈຳນວນ 1 ຄອർສໍາຮັບປະມວລຜລຕ້ວເລຂຈຳນວນເຕີມຂາດ 32 ບີຕ ໂດຍມີຮີສເຕອຣ R0-R15 ແລະຫຼັກ໌ຫຼັກ໌ (VA: Virtual Address)	77
4.3 ຄໍາສັ່ງຂອງໂປຣແກຣມທີ່ຈຸກອ່ານ (Load) ເຂົ້າສູ່ຫ່ວຍຄວາມຈຳແລະແສດງໃນຮູບຂອງການເສເສມບລື ບນໂປຣແກຣມຊົມເລເຕອຣ (Simulator) ທີ່ມາ: http://infocenter.arm.com/help/topic/com.arm.doc.dui0446c/graphics/disassembly_view.png	79
4.4 ຮີຈີສເຕອຣສໍາຮັບເກີບສັນນະຂອງຊື່ປຶ້ງ ປັຈຈຸບັນ (Current Program Status Register: CPSR) http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0344i/Beibijfb.html	83
4.5 ຕ້ວອຍ່າງຄໍາສັ່ງການບວກຄ່າໃນຮີສເຕອຣທີ່ໄດ້ຈາກການຊີຟທີ່ໄປທາງໜ້າຈຳນວນ 2 ບີຕ	84
4.6 ໂຟລ່ວ່າຮົດການທຳການຂອງໂຄງຮັງການເຂົ້າໄຟໂປຣແກຣມ IF	86
4.7 ໂຟລ່ວ່າຮົດການທຳການຂອງໂຄງຮັງການເຂົ້າໄຟໂປຣແກຣມ IF-ELSE	88
4.8 ໂຟລ່ວ່າຮົດການທຳການຂອງການວຽກຮັບໜິດ FOR	91
4.9 ໂຟລ່ວ່າຮົດການທຳການຂອງໂຄງຮັງການເຂົ້າໄຟລູບ While	93
4.10 ໂຟລ່ວ່າຮົດການທຳການຂອງໂຄງຮັງການເຂົ້າໄຟລູບ Do While	94
4.11 ໂຟລ່ວ່າຮົດການທຳການຂອງການວຽກຮັບ 2 ຊັ້ນ	95
4.12 ໂຟລ່ວ່າຮົດການທຳການຂອງການເຮັດໃໝ່ພັກ໌ຂັ້ນ sum(x,y)	96

List of Figures

4.13 Flow Chart Function Call	97
4.14 การควบรวมชุดคำสั่งภาษาแอสเซมบลีของ ARM ตั้งแต่เวอร์ชัน 5 ถึงเวอร์ชัน 8A โดยเวอร์ชันใหม่จะรวมเวอร์ชันเก่าเพื่อรองรับการทำงานซอฟต์แวร์เดิม ที่มา: https://community.arm.com/processors/b/blog/posts/programmer-s-guide-for-armv8-a	99
5.1 ลำดับขั้นของหน่วยความจำชนิดต่างๆ สำหรับชิป BCM2837, (VA: Virtual Address)	102
5.2 ตัวถังแบบ TSOP ของชิพหน่วยความจำชนิดสแตติก Cypress 62256	104
5.3 โครงสร้างของหน่วยความจำชนิดสแตติก Cypress 62256	105
5.4 ไดอะแกรมเวลา (Timing Diagram) สำหรับการอ่าน (Read) ข้อมูลของหน่วยความจำชนิดสแตติกแรม	106
5.5 ไดอะแกรมเวลา (Timing Diagram) สำหรับการเขียน (Write) ข้อมูลของหน่วยความจำชนิดสแตติกแรม	107
5.6 รูปถ่ายด้านบน (Top View) ด้านล่าง (Bottom View) ด้านข้าง (Side View) และภาพตัดขวาง (Cross Section) ของชิป DRAM โดยใช้เทคโนโลยี TSV (Through Silicon Via) ผู้ผลิตบริษัท Elpida	108
5.7 โครงสร้างภายในชิพหน่วยความจำ DRAM ชนิด DDR2 ประกอบด้วยダイ (Die) 2 ชิ้น แต่ละชิ้นมีบัสข้อมูลความกว้าง 16 บิต รวมเป็น 32 บิต	110
5.8 ไดอะแกรมเวลา (Timing Diagram) สำหรับการอ่านข้อมูลของหน่วยความจำ DRAM รุ่น Elpida B8132B4PB-8D-F คำสั่ง Burst READ โดย RL= 5 BL= 4 หมายเหตุ รูปมีการตัดต่อเพื่อเน้นบริเวณที่สำคัญ, NOP=No Operation	112
5.9 ไดอะแกรมเวลา (Timing Diagram) สำหรับการเขียนข้อมูลของหน่วยความจำ Elpida รุ่น B8132B4PB-8D-F ตามคำสั่ง Burst WRITE – WL = 1, BL = 4 หมายเหตุ รูปมีการตัดต่อเพื่อเน้นบริเวณที่สำคัญ, NOP=No Operation	113
5.10 ไดอะแกรมเวลาสำหรับการรีเฟรชหน่วยความจำชนิด DRAM ขนาด 1 กิกะบิต	114
5.11 การเข้มต่อระหว่างรีจิสเตอร์ แคชแลベル 1 และ 2 และหน่วยความจำหลัก (VA: Virtual Address) ภายในชิป BCM2837	115
5.12 การทำงานของแคช L1 แคชคำสั่ง (Instruction Cache) ชนิด Direct Map ที่มา: (Clements (2013))	117
5.13 การทำงานของแคช L1 แคชคำสั่ง (Instruction Cache) ชนิด Set Associative ที่มา: (Clements (2013))	118
5.14 การแมป (Map) แอดдресสมேือน (Virtual Address) ขนาด 2GB เป็นแอดdressกายภาพ (Physical Address) ขนาด 64MB ตามหลักการหน่วยความจำสมேือนชนิดเพจ (Paging Virtual Memory หมายเหตุ เส้นประ คือ รอยต่อระหว่างเพจของหน่วยความจำสมேือน) ที่มา: http://www.wikiwand.com/en/Page_table	121
5.15 โครงสร้างภายในของชิป ARM Cortex A7 ประกอบด้วย TLB และ แคช หลายระดับเพื่อรองรับการทำงานของหน่วยความจำสมேือน (Virtual Memory)	122

5.16 โครงสร้างของหน่วยความจำเสมือน (Virtual Memory) ของบอร์ด Pi3 โมเดล B ซึ่งใช้ชิป ตระกูล BCM283x, x=5, 6, 7 หมายเหตุ ขนาดของรูปไม่เป็นไปตามพื้นที่ตามความเป็นจริง, MMU: Memory Management Unit	124
5.17 การประกอบอาหารตามสูตร ที่มา: https://www.pinterest.com/pin/85779567881855495/ 126	
6.1 การเชื่อมโยงอุปกรณ์ต่างๆ บนบอร์ด Pi3 โดยมีชิป BCM2837 หรือ CM3 (Compute Module 3) เป็นศูนย์กลางด้วยขาจำนวน 200 ขา ที่มา: ?	128
6.2 การส่งแพ็กเก็ตข้อมูลและควบคุมด้วยขาจำนวน TMDS ในช่วงต่างๆ สำหรับความ ละเอียดในการแสดงผล 720x480 ต่อเฟรม ที่มา: https://people.freebsd.org/\protect\unhbox\voidb@x\penalty\@M\{}gonzo/arm/iMX6-HDMI.pdf . . .	129
6.3 หัวเชื่อมต่อ HDMI ชนิด Female ประกอบด้วยขาสัญญาณทั้งหมด 19 ขา ที่มา: https://en.wikipedia.org/wiki/HDMI	130
6.4 จอแสดงผลสำหรับเชื่อมต่อระหว่างบอร์ด Pi3 ด้วยอินเตอร์เฟสการแสดงผลแบบอนุกรม (Display Serial Interface) ที่มา: https://www.element14.com/community/docs/DOC-78156/1/raspberry-pi-7-touchscreen-display	132
6.5 สัญญาณ DSI แบ่งเป็นชนิดหนึ่งเลน (Single Lane) และหลายเลน ที่มา:	133
6.6 การเชื่อมต่อระหว่างบอร์ด Pi3 และกล้องขนาดเล็กด้วยอินเตอร์เฟสกล้องแบบอนุกรม (Camera Serial Interface) ที่มา: https://www.element14.com/community/docs/DOC-83171/1/raspberry-pi-3-model-b-with-camera-module-v2#documents	134
6.7 รูปคลื่นไอน์ (Sine Wave) และสัญญาณ PCM (Pulse Code Modulation) 16 ระดับ ที่มา: https://en.wikipedia.org/wiki/Pulse-code_modulation	136
6.8 บัฟเฟอร์สำหรับส่งและรับ ข้อมูลเสียงดิจิทัลชนิด PCM จากการเชื่อมต่อชนิด I2S ที่มา: Broadcom (2012)	137
6.9 แจ็ค 3.5 มม. (กลาง) ชนิด 4 ขั้วสำหรับเสียบกับบอร์ด Pi3 (ขวา) เพื่อส่งสัญญาณภาพไป ยังแจ็ค RCA (เหลือง) และสัญญาณเสียงไปยังแจ็ค RCA (แดงและขาว) ที่มา	138
6.10 หัวเชื่อมต่อ USB ชนิด A (บน ฝั่งไฮสต์) และ B (ล่าง ฝั่งอุปกรณ์) ประกอบด้วยสัญญาณ 4 เส้น กราวด์ (GND) Data+ Data- และไฟเลี้ยง 5 โวลท์	139
6.11 โครงสร้างของชิป LAN 9514 ภายในประกอบด้วยวงจร USB Hub และวงจร Ethernet Microchip Technology (2009)	140
6.12 หัวเชื่อมต่อชนิด RJ45 (ช้ายสุด) สำหรับการเชื่อมต่อเครือข่ายท้องถิ่น (Local Area Net- work) แบบมีสาย Ethernet	141
6.13 การเข้าปลายนาย RJ45 ทั้งสองด้าน สำหรับการเชื่อมต่อระหว่างเครื่องคอมพิวเตอร์และอุ ปกรณ์สวิทช์ (Switch) ตามมาตรฐาน TIA T568B	141
6.14 รูปถ่ายและรูปขยายของชิป BCM 43438 สำหรับเชื่อมต่อเครือข่ายไร้สายท้องถิ่น (Wire- less Local Area Network) หรือ WiFi และเครือข่ายไร้สายบลูทูธ (Bluetooth)	142

List of Figures

6.15 บล็อกไดอะแกรมของชิพ BCM 43438 ประกอบด้วยขาสัญญาณเชื่อมต่อเสาอากาศ และขาเชื่อมต่อกับปุ่มโคลคันโทรลเลอร์ ที่มา: Corporation (2017)	143
6.16 การแมปหน่วยความจำระหว่างแอดเดรสบัส (Bus Address) ของ Video Core (VC) และแอดเดรสภายใน ARM	144
6.17 ตารางแอดเดรสบัสเริ่มต้นที่หมายเลข 0x7E00 0000 สำหรับอุปกรณ์อินพุตเอาท์พุต (IO Peripherals) และหัวข้อ	145
6.18 โครงสร้างภายในขา GPIO สำหรับชิพระกูลเดียวกับ BCM2835 ที่มา: Broadcom (2012)	148
6.19 โครงสร้างภายในของ Generic Interrupt Controller (GIC) และการเชื่อมโยงกับโมดูลอื่นๆ กับ ARM Advanced eXtensible Interface (AXI) ที่มา: ?gic), APB (ARM Peripheral Bus),	152
6.20 การทำงานและไดอะแกรเวลาของ Generic Interrupt Controller (GIC) ที่มา: ?gic)	153
6.21 การเชื่อมต่อระหว่าง Cortex A5 และโมดูล DMA (Direct Memory Access) ด้วยบัส AXI (Advanced eXtensible Interface) ภายในชิพ ที่มา: https://www.design-reuse.com/sip/blockdiagram/37237/20150520111508-main-Performance_blk.jpg	155
6.22 การเชื่อมต่อ MCU (MicroController Unit) สองตัวด้วย UART ที่มา: ที่มา 1 และ ที่มา 2	157
6.23 การเชื่อมต่อแบบ SPI: Serial Peripheral Interface ประกอบด้วยมาสเตอร์ (Master) และสเลฟ (Slave) ที่มา: https://en.wikipedia.org/wiki/Serial_Peripheral_Interface_Bus	159
6.24 การส่งข้อมูลจาก SPI มาสเตอร์ (Master) และ สเลฟ (Slave) ด้วยการซิงไครโนซีกับสัญญาณ CS (Chip Select) และคลีก (CLK)	160
6.25 การเชื่อมต่อแบบ I2C (Inter-Integrated Circuit) ประกอบด้วยมาสเตอร์ (Master) และสเลฟ (Slave) ที่มา: http://www.thaimicrotron.com/CCS-628/Referrence/I2CBUS.htm	161
6.26 สัญญาณ PWM ที่ดิวตี้ไซเคิล D_{cycle} =10% 30% 50% และ 90% ที่มา: https://protostack.com.au/2011/06/atmega168a-pulse-width-modulation-pwm/	162
6.27 การแปลงไฟกระแทรง 5 โวลต์เป็นไฟกระแทรงด้วยไอซี PAM 2306 DC-DC Covertor คู่ กำหนดค่าเอาท์พุตด้วยค่าตัวเหนี่ยวนำ L ₁ และ L ₂ หน่วยเป็น ไมโคร哼รี ที่มา: Diodes (2012)	163
7.1 หน่วยความจำ NAND Flash หมายเลขผลิตโดยตัวถังชนิด TSOP (Thin Small Outline Package) จำนวน 48 ขา ที่มา: Micron Technology (2004)	165
7.2 โครงสร้างภายในหน่วยความจำ Flash NAND ที่มา: Micron Technology (2004)	167
7.3 Random Read หน่วยความจำ Flash NAND ที่มา: Micron Technology (2004)	168
7.4 SD Memory Card วงจรสัญญาณในโหมด SD	170
7.5 แผ่นวงจรพิมพ์ภายในอุปกรณ์ SSD ชนิด SATA III ประกอบด้วยชิพหน่วยความจำแฟลชไดนามิกแรม คอนโทรลเลอร์สำหรับควบคุม	172

7.6	โครงสร้างของ SSD ประกอบด้วย ส่วนเชื่อมต่อกับเครื่อง ไมโครคอนโทรลเลอร์ บัฟเฟอร์ และหน่วยความจำแฟลช	173
7.7	องค์ประกอบของอุปกรณ์ฮาร์ดดิสก์ไดร์ฟ (HDD)	174
7.8	โครงสร้างของฮาร์ดดิสก์ไดร์ฟแบ่งเป็น Sector, Track, Cylinder	175
A.1	กรอกเลข -123 ลงในกล่องข้อความ และคลิกเลือกที่ปุ่ม Signed เพื่อให้เป็นเลขฐานสองชนิด Signed	180
A.2	ผลลัพธ์จากการแปลงเลข -123 ให้เป็นเลขฐานสองชนิด Signed 2-Complement ความยาว 24 บิต	181
A.3	ผลลัพธ์จากการแปลงเลข -123 ให้เป็นเลขฐานสิบหกชนิด Signed 2-Complement ความยาว 24 บิตหรือ 6 ตัวเลข	181
A.4	วางแผนการแปลงเลข -123 ให้เป็นเลขฐานสองชนิด Signed 2-Complement ความยาว 32 บิต	182
A.5	ผลลัพธ์ของการแปลงเลข -123 ให้เป็นเลขฐานสองชนิด Signed 2-Complement ความยาว 32 บิต	182
A.6	ผลลัพธ์จากการแปลงเลข 123 ให้เป็นเลขฐานสองชนิด Single Precision	183
A.7	ผลลัพธ์จากการแปลงเลข -123 ให้เป็นเลขฐานสองชนิด Single Precision	184
A.8	เมนูด้านล่างสุดของหน้าเว็บ เพื่อเลือกเลขฐานสองชนิด Single Precision (Binary32) และ Double Precision (Binary64)	184
A.9	ผลลัพธ์จากการบวกเลข -123+123 ให้เป็นเลขฐานสองชนิด Single Precision	185
A.10	ผลลัพธ์จากการคูณเลข -123 x 123 ให้เป็นเลขฐานสองชนิด Single Precision	185
A.11	ผลลัพธ์จากการแปลงเลข 123 ให้เป็นเลขฐานสองชนิด Double Precision	186
A.12	ผลลัพธ์จากการแปลงเลข -123 ให้เป็นเลขฐานสองชนิด Double Precision	186
A.13	ผลลัพธ์จากการบวกเลข -123+123 ให้เป็นเลขฐานสองชนิด Double Precision	187
A.14	ผลลัพธ์จากการคูณเลข -123 x 123 ให้เป็นเลขฐานสองชนิด Double Precision	188
A.15	เว็บสำหรับการตอบคำถามเพื่อสร้างเลขหรือแปลงเลขฐานสิบด้วยมาตรฐาน IEEE 754 Single Precision การกดเลือกคือทำให้บิตนั้นเท่ากับ '1'	188
A.16	ผลลัพธ์จากการกรอกและแปลงตัวอักษร ໄ ຖ ຍ ก ຂ ค a b c เป็นรหัสต่างๆ	190
B.1	Layout of Raspberry Pi 3 Model B	192
B.2	Raspberry Pi 3 with heat sinks	193
B.3	Raspberry Pi 3 case	194
B.4	A Raspberry Pi 3 with case	195
B.5	Raspberry Pi 3 case with 2x20 tall stack header	195
B.6	Connect HDMI to VGA adapter, mouse and keyboard to the Raspberry Pi	196
B.7	Insert microSD card to the Raspberry Pi	196
B.8	Power cable connection	197

List of Figures

C.1	SDCard formatting (Windows)	200
C.2	SDCard formatting (Mac OS X)	201
C.3	Unzipped NOOBS file for Windows	201
C.4	Unzipped NOOBS file for Mac OS X	202
C.5	Install Raspbian Operating System	203
C.6	Graphical User Interface of Raspbian	203
C.7	List of available Wi-Fi networks	204
C.8	List of available Wi-Fi networks	204
C.9	List of available Wi-Fi networks	205
C.10	List of available Wi-Fi networks	205
E.1	New project using "Console application" wizard	218
E.2	Selecting C language for the project	218
E.3	Specify project folder location	219
E.4	Specify project folder location	219
F.1	New project using "Console application" wizard	226
F.2	Selecting C language for the project	227
F.3	Specify project folder location	228
F.4	Specify project folder location	229
F.5	Remove main.c from the project	229
F.6	Add an empty file to the project	229
F.7	Click "Yes" to add the file to the project	230
F.8	Save the file with a filename	230
H.1	Connecting Ground	242
H.2	Put LED on breadboard	243
H.3	Connecting the resistor	243
H.4	Complete the circuit	244
H.5	Mapping of WiringPi and GPIO pins	245
I.1	ARM Stack Frame ก่อนและระหว่างที่เกิดขบวนการ Interrupt	252
I.2	Push button for interrupt programming	254

บทนำ

คอมพิวเตอร์มีประโยชน์ต่อการใช้งานด้านการทำงาน การศึกษา รวมถึงชีวิตประจำวันมากขึ้น ทำราเล่มนี้สามารถใช้ประกอบการเรียนการสอนวิชาองค์ประกอบคอมพิวเตอร์ (Computer Organization) และวิชาสถาปัตยกรรมคอมพิวเตอร์ (Computer Architecture) กรณีศึกษา บอร์ด Raspberry Pi3 และ ARM Cortex A53

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)

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

1.1.2 คอมพิวเตอร์เซิร์ฟเวอร์หรือแม่ข่าย (Server computers)

คอมพิวเตอร์เซิร์ฟเวอร์หรือแม่ข่ายจะต้องเข้มต่อกับเครือข่ายตลอดเวลา เพื่อรับการใช้งานจากเครื่องคอมพิวเตอร์อื่นๆ จำนวนมากๆ โดยคอมพิวเตอร์แม่ข่ายจะมีความจุ สมรรถนะ และความเชื่อมั่นสูงเพื่อรับการใช้งาน ระบบปฏิบัติการที่ได้รับความนิยมสำหรับเครื่องเหล่านี้ ได้แก่ ลีนุกซ์ และไมโครซอฟต์วินโดวส์



Figure 1.1: ตัวอย่าง เครื่อง เซิร์ฟเวอร์ ของ HP รุ่น Proliant ที่มา: <https://www.datacenterknowledge.com/archives/2009/06/10/hp-scales-out-with-new-cloud-servers>

1.1.3 คอมพิวเตอร์พกพา (Portable Computers)

คอมพิวเตอร์พกพามีขนาดเล็กและพกพาง่าย ใช้กำลังไฟจากแบตเตอรี่เป็นหลัก ได้แก่ แท็บเล็ท และโทรศัพท์เคลื่อนที่สมาร์ทโฟน (Smart Phone) เป็นต้น ระบบปฏิบัติการที่ได้รับความนิยมสำหรับเครื่องเหล่านี้ ได้แก่ แอนดรอยด์ (Android) และ แอปเปิล iOS ทั้งสองระบบมีใช้บันโทรศัพท์สมาร์ทโฟนและแท็บเล็ตมากกว่า 90% ทั่วโลก ผู้ใช้สามารถติดตั้งแอนดรอยด์ซึ่งพัฒนาต่อจากระบบปฏิบัติการลินุกซ์บนบอร์ด Pi3 ได้เช่นกัน รายละเอียดเพิ่มเติมสามารถศึกษาได้จาก <https://howtoraspberrypi.com/install-android-raspberry-pi/>

1.1.4 คอมพิวเตอร์ฝังตัว (Embedded computers)

คอมพิวเตอร์ฝังตัวเป็นสมองกลที่ซ่อนอยู่ในระบบ เช่น รถยนต์ เครื่องบิน ทีวี ตู้เย็น ฯลฯ ทำงานอยู่ใต้เงื่อนไขของกำลังไฟ/สมรรถนะ/ราคา ปัจจุบัน คอมพิวเตอร์ฝังตัวได้รับความนิยมเพื่อทำหน้าที่เป็นเซ็นเซอร์ (Sensor) และแขนกล (Actuator) สำหรับงานประเภทต่างๆ มากขึ้นและเข้มต่อกับเครือข่ายอินเทอร์เน็ตได้อย่างแพร่หลาย ทำให้มีชื่อเรียกว่า Internet of Things หรือ IoT

1.2 แนวโน้มของจำนวนอุปกรณ์คอมพิวเตอร์ชนิดต่างๆ

กราฟในรูปที่ 1.2 แสดงแนวโน้มของจำนวนอุปกรณ์คอมพิวเตอร์ กลุ่มคอมพิวเตอร์ส่วนบุคคลตั้งโต๊ะ และกลุ่มคอมพิวเตอร์พกพาประเภทสมาร์ทโฟนและแท็บเล็ต ตั้งแต่ปี ค.ศ. 2005-2015 จะเห็นได้ว่า กลุ่มスマาร์ทโฟนและแท็บเล็ต มีจำนวนเพิ่มมากขึ้นแบบก้าวกระโดด ในขณะเดียวกัน กลุ่มพีซียังมีแนวโน้มลดลงไปเรื่อยๆ โดยรายละเอียด คือ สมาร์ทโฟนขนาดใหญ่ (Large Smartphone) จะได้รับความนิยมเพิ่มมากขึ้นเรื่อยๆ ในขณะที่สมาร์ทโฟนขนาดเล็ก (Small Smartphone) จะได้รับความนิยมมากกว่า แต่จะมีแนว

Global Smartphone + Tablet Shipments Exceeded PCs in Q4:10

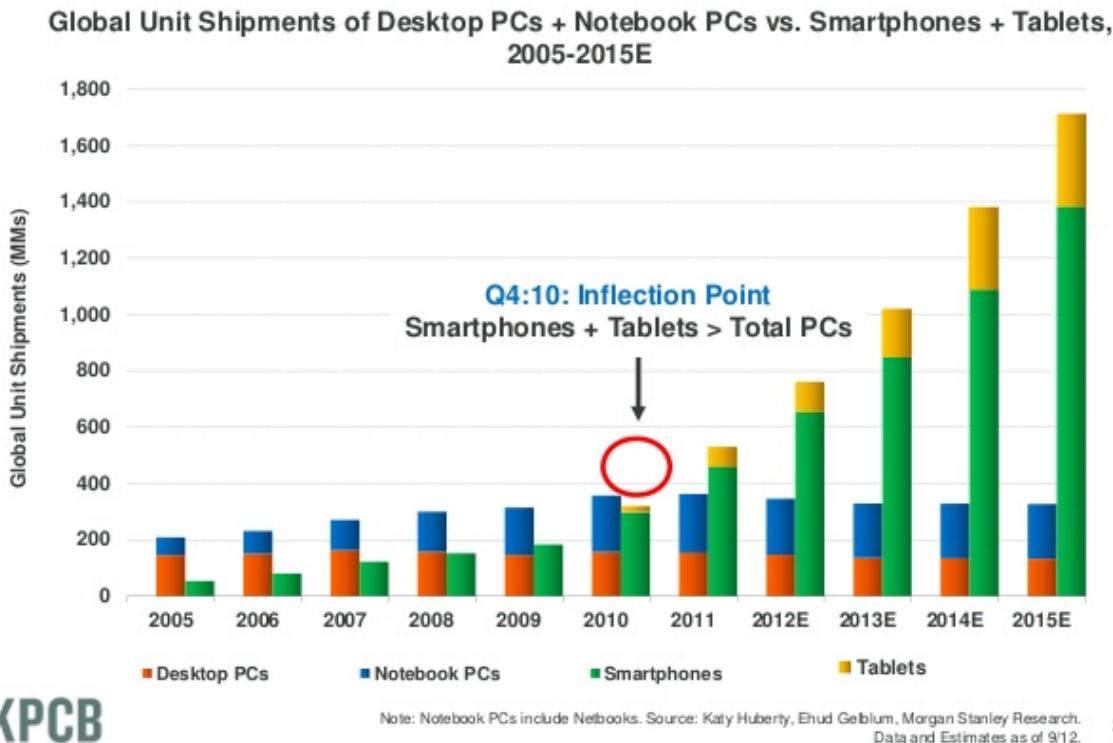


Figure 1.2: ยอดขายเครื่องคอมพิวเตอร์ชนิดต่างๆ ทั่วโลกในปี 2005-2015 ที่มา: https://www.slideshare.net/kleinerperkins/2012-kpcb-internet-trends-yearend-update/25-Global_Smartphone_Tablet_Shipments_Exceeded

โน้มลดลง แท็บเล็ทขนาดใหญ่ (Large Tablet) และแท็บเล็ทขนาดเล็ก (Small Tablet) มีแนวโน้มค่อนข้างคงที่

1.3 ขั้นตอนการผลิตไมโครโปรเซสเซอร์

ขั้นตอนการผลิต (Process) ของชิป (Chip) ไมโครโปรเซสเซอร์และอื่นๆ ในรูปที่ 1.3 มีลักษณะคล้ายการทำแผ่นวงจรพิมพ์แต่มีความซับซ้อนและเทคโนโลยีในการผลิตสูง โดยเริ่มต้นจากการนำแท่งซิลิกอน (Silicon ingot) บริสุทธิ์มาสไลซ์ (Slice) ให้เป็นแผ่นบาง เรียกว่า เวเฟอร์เปล่า (Blank Wafer) เมื่อนำแผ่นเวเฟอร์เปล่าเหล่านี้ไปผ่านกระบวนการปลูกถ่ายสาร (Doping) และอื่นๆ จำนวน 20-40 ขั้นตอน จนกลายเป็นแผ่นเวเฟอร์ที่มีลวดลาย (Patterned wafers) โดยเวเฟอร์หนึ่งแผ่นประกอบด้วย ダイ (Die) จำนวนหนึ่งกระจายในลักษณะตารางตั้งรูป หลังจากการทดสอบダイแต่ละตัวด้วยทดสอบเวเฟอร์ (Wafer tester) ダイตัวที่ไม่ผ่านการทดสอบจะถูกกากบาทเพื่อทำเครื่องหมายแล้ว แผ่นเวเฟอร์จะถูกตัดด้วยไดเซอร์ (Dicer) ให้ด้วยแยกตัวออกจากกันเป็นสี่เหลี่ยม ダイตัวที่ผ่านการทดสอบก็จะถูกบอนด์ (Bond) เข้ากับตัวถัง (Package) ให้ตรงตามลักษณะที่ต้องการ หลังจากการทดสอบด้วย Part tester ダイที่ไม่ผ่านก็จะถูกทำเครื่องหมาย และไม่ถูกส่ง (Ship) ไปยังลูกค้า

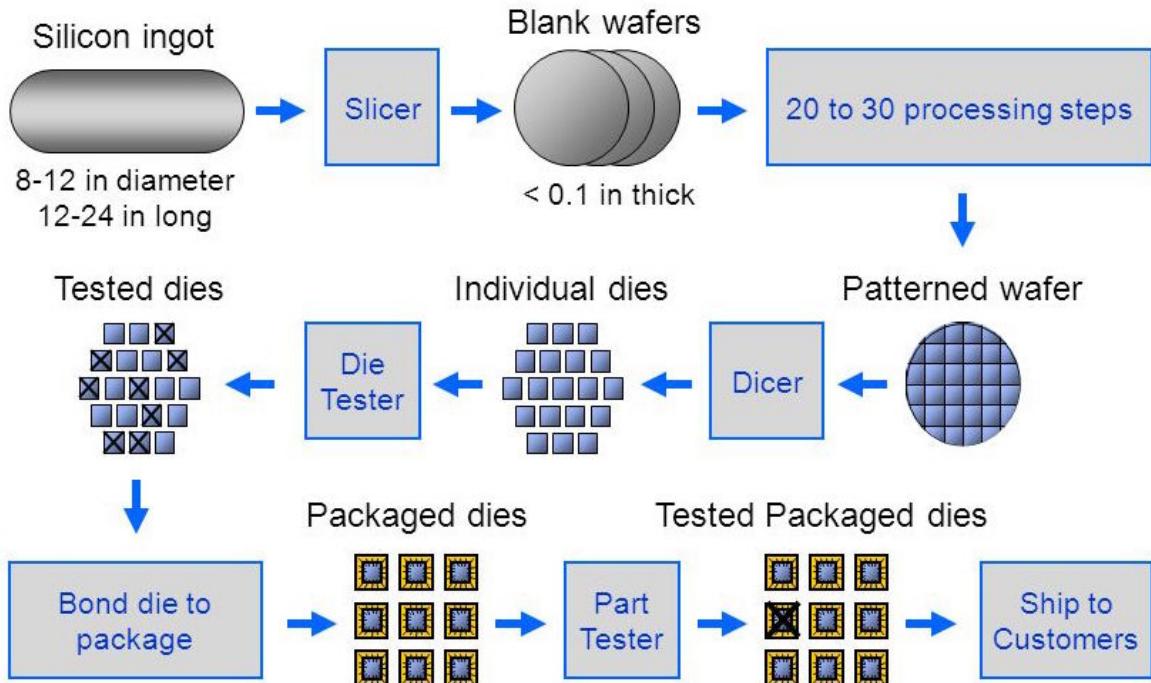


Figure 1.3: ขั้นตอนการผลิต (Process) ของชิป (Chip) ไมโครโปรเซสเซอร์และอื่นๆ ที่มา: <https://slideplayer.com/slide/5218297/>

ชิป (Chip) คือ ดaiyที่ห่อหุ้มด้วยแพ็คเกจเรียบร้อยแล้ว ซิลิกอน คือ ธาตุหมู่ที่ 4 มีวาเลนซ์อิเลคตรอน (Valence Electron) วงนอกสุดจำนวน 4 ตัว แผ่นซิลิกอนເວັບໂອ คือ การนำธาตุซิลิกอนที่สกัดและหลอมจนเป็นแท่ง (Silicon Ingot) มาแล่เป็นแผ่นบางๆ การໂດປາສາຣ คือ การเปลี่ยนແປງແຜ່ນເວັບໂອບາງ ຕໍາແໜ່ງເພື່ອໃກ້ລາຍເປັນ P ໂດຍການເພີ່ມໂຫຼດ (Hole) ແລະ ລາຍເປັນສາຣ N ໂດຍການເພີ່ມອີເລັດ ທຳໄໝ ເກີດກາຮູ້ເຊື່ອມໂຍງກັນເປັນ ທຣານຊີສເຕັອຣ 2 ຊົນດີ คือ N-MOS ແລະ P-MOS ລາຍເປັນ ວົງຈາລອຈິກເກຕ (Logic Gate) ຕ່າງໆ ແລະ ໂມໂຄດູລ ຕ່າງໆ ໂດຍໃຫ້ธาຕຸທອງແດງ ອີເລັດ ອຸລຸມີເນີຍ (Aluminum) ໃນກາຮູ້ເຊື່ອມໂຍງ ທັ້ງໃນແນວນອນ ເຮັດວຽກ ອິນເຕອົງຄອນເນັົກ (Interconnect) ແລະ ແນວດິງເຮັດວຽກ ວິເຢ (Via)

1.4 ไมโครโปรเซสเซอร์ชนิดມັລຕິຄອർ

รูปที่ 1.4 คือ ภาพถ่ายดาย (Die) ของโปรเซสเซอร์ BCM 2835 ซึ่งใช้สำหรับบอร์ด Raspberry Pi เกิดจาก การถอดแพ็คเกจพลาสติกสีดำออก โดยมีคุณลักษณะเฉพาะ (Specification) ดังนี้

- โปรเซสเซอร์ ARM1176JZF-S ทำงานที่สัญญาณนาฬิกาความถี่ 700 MHz แคชලেเวล 1 (L1) ขนาด 16 KB แคชලেเวล 2 (L2) ขนาด 128 KB
- หน่วยประมวลผลด้านกราฟิก ອີເລັດ GPU ຮຸນ VideoCoreIV ມາຍໃນชີພຕັວເດືອກກັນ

บอร์ด Pi3 ในปัจจุบัน ໃຊ້ເຕොໂລຢີທີ່ສູງຂຶ້ນ ສມරรถະແລະ ປະສິທິກາພເພີ່ມຂຶ້ນ ໂດຍມີຈຳນວນຄອർ (Core) ອີເລັດ 4 ຄອർ ຮາຍລະເອີຍດເພີ່ມເຕີມໃນບທທີ່ 3

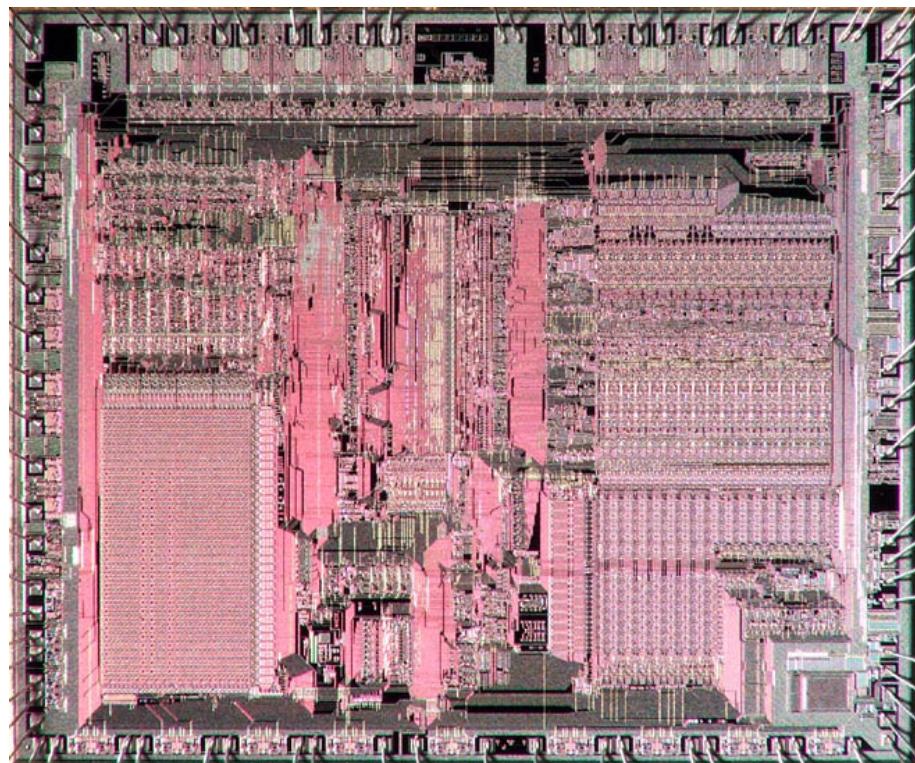


Figure 1.4: ภาพถ่ายดาย (Die) โครงสร้างและรายละเอียดของชิป BCM 2835 บนบอร์ด Raspberry Pi
ที่มา: <https://www.raspberrypi.org/forums/viewtopic.php?t=63750>

Chapter 2

ข้อมูลและคณิตศาสตร์ในคอมพิวเตอร์

วัตถุประสงค์

- เพื่อให้เข้าใจเลขจำนวนเต็มชนิดไม่มีเครื่องหมายและมีเครื่องหมาย (Integers: Unsigned and Signed) และคณิตศาสตร์ในเครื่องคอมพิวเตอร์
- เพื่อให้เข้าใจเลขจำนวนจริงชนิดจุดทศนิยมคงที่ (Fixed-point real numbers) และคณิตศาสตร์ในเครื่องคอมพิวเตอร์
- เพื่อให้เข้าใจเลขจำนวนจริงชนิดจุดทศนิยมลอยตัว (Floating-point real numbers) และคณิตศาสตร์ในเครื่องคอมพิวเตอร์
- เพื่อให้รู้จักรหัสเลขฐานสองตามมาตรฐาน ASCII และ Unicode สำหรับข้อมูลตัวอักษร

ไมโครโปรเซสเซอร์ตัวแรกของโลก คือ Intel 4004 ในปี ค.ศ. 1971 [Intel 4004](#) สามารถคำนวณเลขจำนวนเต็มได้เพียง 4 บิต ชนิด BCD (Binary Coded Decimal) ซึ่งใช้คำนวณเลขแต่ละตัวเป็นตัวอักษรในเลขฐานสิบ โดยสามารถนับเลขจาก 0 ถึง 9 ดังนั้น ไมโครโปรเซสเซอร์ในปัจจุบันจำเป็นต้องคำนวณตัวเลขทั้งจำนวนเต็มและทศนิยม เพื่อรับรองการใช้งานทางคณิตศาสตร์ได้หลากหลายและซับซ้อนมากขึ้น

ในปัจจุบัน การใช้งานเครื่องคอมพิวเตอร์ชนิดต่างๆ เพื่อประมวลผลข้อมูลและอักษรให้กลายเป็นสารสนเทศ (Information) ที่เป็นประโยชน์ต่อธุรกิจการค้า สื่อสังคม และวัตถุประสงค์อื่นๆ การประมวลผลข้อมูลและอักษรเหล่านี้ จำเป็นต้องใช้คณิตศาสตร์สำหรับข้อมูลพื้นฐานด้วยอาร์ดแวร์ โดยมุ่งเน้นย้ำเข้าใจข้อมูลเชิงจำนวนเป็นตัวเลขฐานสิบ แต่คอมพิวเตอร์จำเป็นต้องประมวลผลด้วยเลขฐานสองแทน

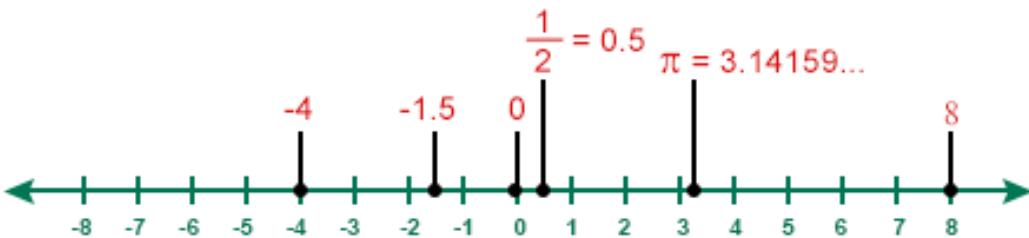


Figure 2.1: เลขจำนวนเต็มและเลขจำนวนจริงบนเส้นจำนวนในคณิตศาสตร์ ที่มา: <https://math.tutorvista.com/number-system/real-number-line.html>

ดังนั้น เพื่อให้คอมพิวเตอร์สามารถคำนวณหรือกราฟทำกระบวนการเชิงคณิตศาสตร์กับเลขฐานสิบที่เป็นจำนวนเต็ม หรือจำนวนจริงดังรูปเส้นจำนวน ในรูปที่ 2.1 ในรูปของเลขฐานสองได้ เลขฐานสองจึงแบ่งเป็นเลขจำนวนเต็ม (Integer) และเลขจำนวนจริง (Real number) คล้ายกับเลขฐานสิบ ส่วนตัวอักษรจะแต่ละตัวที่มีนุชย์อ่านเข้าใจ คือ เลขฐานสอง เช่นเดียวกัน

ทั้งนี้เนื่องจากข้อมูลและคำสั่งจะอยู่ในรูปของระดับโวლเตจ หรือ ทิศทางของกระแสไฟฟ้า เป็นเลข '1' หรือ '0' เรียกว่า บิต เพื่อแทนระดับโวลเตจสูง และ โวลเตจต่ำ ตามลำดับ ซึ่งการจัดเรียงบิตข้อมูลเหล่านี้หลายๆ บิต จึงกลายเป็นเลขฐานสอง และจำเป็นต้องใช้คณิตศาสตร์คอมพิวเตอร์ (Computer Arithmetic)

2.1 ตัวแปรชนิดต่างในภาษา C/C++

ผู้อ่านควรมีพื้นฐานการเขียนหรือพัฒนาโปรแกรมคอมพิวเตอร์ด้วยภาษา C หรือ C++ มาบ้าง เพื่อช่วยให้เข้าใจบทนี้และบทต่อๆ ไปได้ดีขึ้น ตารางที่ 2.1 แสดง ชนิดของตัวแปร ขนาด (บิต) ค่าต่ำสุด (Minimum) และค่าสูงสุด (Maximum) สำหรับภาษา C/C++ ยกตัวอย่าง เช่น ตัวแปรชนิด char นั้น ต้องการพื้นที่จำนวน 8 บิต หรือ 1 ไบท มีค่าต่ำสุดเท่ากับ -128 หรือ -2^7 มีค่าสูงสุดเท่ากับ +127 หรือ $+2^7 - 1$

Table 2.1: ชนิดตัวแปร จำนวนบิต ค่าต่ำสุด และค่าสูงสุดของตัวแปรแต่ละชนิดในภาษา C/C++

ชนิด	บิต	ค่าต่ำสุด	ค่าสูงสุด
char	8	-128	+127
unsigned char	8	0	255
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	$\pm 2^{-126}$	$\pm 2^{127}$
		$\pm 1.18 \times 10^{-38}$	$\pm 3.4 \times 10^{38}$
double	64	$\pm 2^{-1023}$	$\pm 2^{1022}$
		$\pm 1.23 \times 10^{-88}$	$\pm 1.23 \times 10^{88}$

ในขณะที่ตัวแปรชนิด int (integer) นั้น ไม่ครอบคลุมส่วนใหญ่ต้องการพื้นที่จัดเก็บตัวแปรชนิด int นี้ 1 ตัวเป็นพื้นที่ 32 บิต หรือ 4 ไบท โดยมีค่าต่ำสุดเท่ากับ -2^{31} หรือ $-2,147,483,648$ และมีค่าสูงสุด

เท่ากับ $+2^{31}-1$ หรือ +2,147,483,647

C Code Example eC.3 EXAMPLE DATA TYPES

```
// Examples of several data types and their binary representations
unsigned char x = 42;           // x = 00101010
short y = -10;                 // y = 11111111 11110110
unsigned long z = 0;            // z = 00000000 00000000 00000000 00000000
```

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

การเขียนโปรแกรมนั้น ผู้เขียนหรือนักพัฒนาจะต้องประกาศชื่อตัวแปร ก่อนจะตั้งค่าเริ่มต้น (Initialize) หรือเปลี่ยนแปลงค่าของตัวแปรนั้นจากการคำนวณหรือประมวลผลได้ ตัวอย่างในรูปที่ 2.2 การประกาศตัวแปรและตั้งค่าเริ่มต้น ให้

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

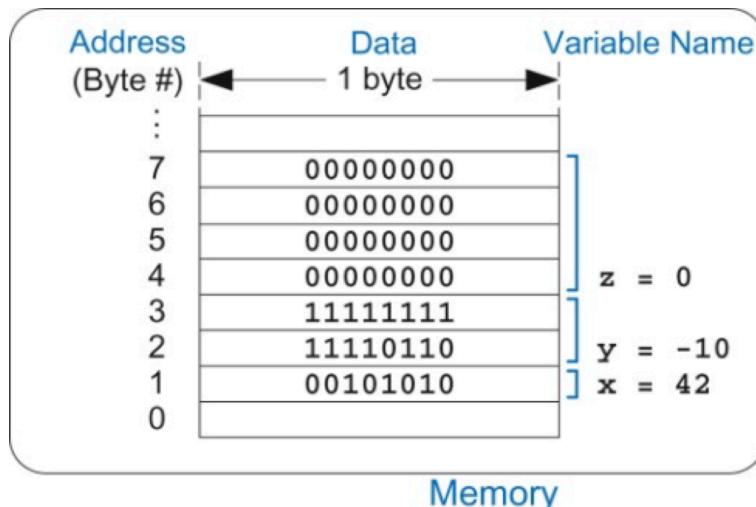


Figure 2.3: พื้นที่ในหน่วยความจำซึ่งบรรจุค่าเริ่มต้นของตัวแปรที่ได้ประกาศในรูปที่ 2.2 (ที่มา: [Harris and Harris \(2013\)](#))

การประกาศตัวแปรและตั้งค่าเริ่มต้น ในรูปที่ 2.2 ก่อนที่โปรแกรมจะเริ่มทำงาน ระบบปฏิบัติการจะจดจำพื้นที่ของหน่วยความจำในลักษณะเดียวกับ รูปที่ 2.3 ซึ่งหมายเลขไบท์ (Byte Number) ในหน่วยความจำหลัก (Main Memory) จะเรียกว่าเป็นชั้นๆ ตั้งแต่หมายเลขไบท์ที่ 0 จำกัด้านล่างขึ้นไปทางหมายเลขไบท์ที่สูงขึ้น คล้ายกับตึกสูง แต่เริ่มต้นนับจาก 0 และแต่ละชั้นบรรจุข้อมูลได้เพียง 1 ไบท์ หากต้องการพื้นที่มากกว่า จะต้องนำพื้นที่ของชั้นถัดไปมาใช้งานด้วย ยกตัวอย่างเช่น ตัวแปร y และ z ในรูปที่ 2.3 ต้องการพื้นที่ 2 และ 4 ไบท์ ตามลำดับ

Chapter 2. ข้อมูลและຄณิตศาสตร์ในคอมพิวเตอร์

ในบทนี้ ผู้อ่านจะได้ทำความเข้าใจกับข้อมูลชนิดจำนวนเต็มก่อน เพราะเป็นพื้นฐานของข้อมูลชนิดอื่นๆ และทำความเข้าใจกับเนื้อหาด้วยการปฏิบัติตามการทดลองที่ 1 ภาคผนวก A

2.2 เลขฐานสองชนิดจำนวนเต็ม

เลขฐานสองชนิดจำนวนเต็ม (Integer) แบ่งเป็นเลขจำนวนเต็มไม่มีเครื่องหมาย (Unsigned Integer) และจำนวนเต็มมีเครื่องหมาย (Signed Integer) ซึ่งต้องมีกระบวนการการบวกและลบ การคูณและหาร และการตรวจจับโอเวอร์เฟล์ (Overflow) เนื่องจากการคำนวณด้วยวงจรดิจิทัลภายใน

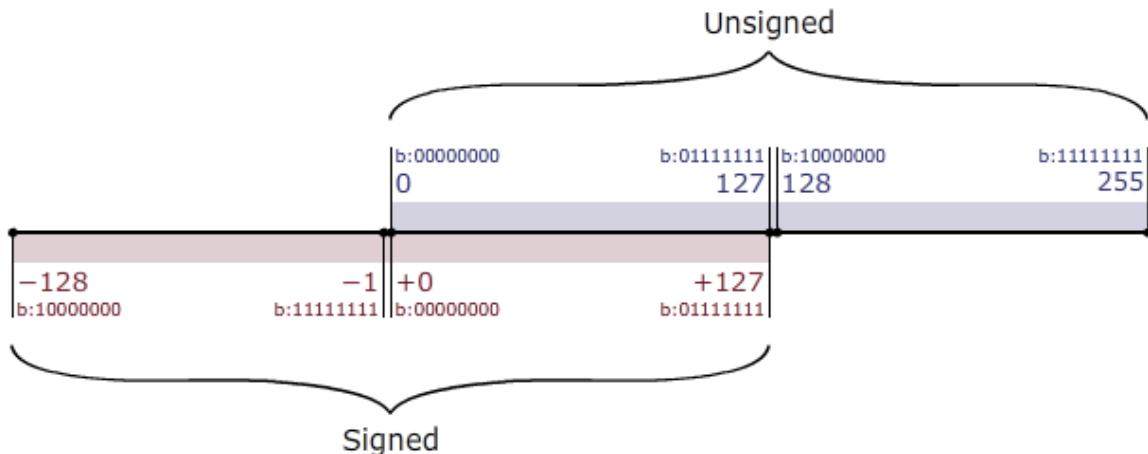


Figure 2.4: เส้นจำนวนเปรียบเทียบข้อมูลขนาด 8 บิตในรูปแบบของเลขจำนวนเต็มชนิด Unsigned แบบ 2-Complement ที่มา: <https://stackoverflow.com/questions/27527943/how-does-adding-min-value-compare-integers-as-unsigned>

รูปที่ 2.4 เส้นจำนวนเปรียบเทียบข้อมูลขนาด 8 บิตในรูปแบบของเลขจำนวนเต็มชนิด Unsigned และชนิด Signed แบบ 2-Complement

2.2.1 ชนิดไม่มีเครื่องหมาย (Unsigned Integer)

เลขจำนวนเต็มชนิดไม่มีเครื่องหมายในคอมพิวเตอร์ หมายความว่าการใช้ชันบจำนวนสีงของต่างๆ ซึ่งจำนวนจะมีค่าเริ่มต้นตั้งแต่ 0 จนถึงจำนวนที่อาร์ดแวร์และซอฟต์แวร์รองรับ

นิยามที่ 2.2.1. กำหนดให้ เลขฐานสอง $X_{2,u}$ เป็นเลขจำนวนเต็มชนิดไม่มีเครื่องหมาย (Unsigned Integer) เขียนอยู่ในรูป

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

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

การแปลงเลขฐานสองเป็นฐานสิบ

จากนิยามที่ 2.2.1 ค่าฐานสิบ $X_{10,u}$ 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 ค่าของเลขฐานสิบ $X_{10,u}$ อยู่ในช่วง 0 ถึง $+2^n - 1$

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

ค่าฐานสิบของ $X_{2,u}$ คือ

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

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

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

เลขจำนวนเต็มขนาด $n = 4$ บิตจะมีค่าฐานสิบอยู่ในช่วง 0 ถึง $+15$

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

ค่าฐานสิบของ $X_{2,u}$ คือ

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

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

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

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

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

ค่าฐานสิบของ $X_{2,u}$ คือ

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

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

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

เลขจำนวนเต็มขนาด $n = 16$ 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

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

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

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

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

ผู้อ่านจะเห็นได้จากตัวอย่างที่ 2.2.1 ถึง 2.2.4 เลขฐานสิบค่าเดียวกัน เมื่อนำไปคำนวณในวงจรดิจิทัล ที่มีความยาวต่างกัน

การแปลงเลขฐานสิบเป็นฐานสอง

การแปลงเลขฐานสิบเป็นฐานสองชนิดไม่มีเครื่องหมาย (Unsigned) แบ่งเป็น 2 วิธี คือ การหาร และ การลับ

- การหาร เป็นวิธีที่เข้าใจง่ายและซับซ้อนน้อยกว่าวิธีการลับ

ตัวอย่างที่ 2.2.5. การแปลง 123 เป็นเลขฐานสองด้วยวิธีหาร

Table 2.2: การแปลงค่าฐานสิบเป็นเลขฐานสองแบบไม่มีเครื่องหมายความยาว $n=8$ บิต ด้วยการหาร

บิตที่	เลขฐานสิบ	ผลหารฐานสิบ	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 โดยผู้ใช้ควรจะจำเลขิกกำลังสองได้เหมาะสำหรับผู้อ่าน ที่มีประสบการณ์กับการใช้เลขฐานสองบ่อยๆ

ตัวอย่างที่ 2.2.6. การแปลง 123 เป็นเลขฐานสองด้วยวิธีลับ

Table 2.3: การแปลงค่าฐานสิบเป็นเลขฐานสองแบบไม่มีเครื่องหมายความยาว $n=8$ บิต

บิตที่ (i)	ค่าประจำ ตำแหน่ง (2^i)	ตัวตั้ง- 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

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

ตัวอย่างที่ 2.2.5 และ ตัวอย่างที่ 2.2.6 แสดงการแปลงเลขฐานสิบ เป็นเลขฐานสองชนิดไม่มีเครื่องหมาย ทั้งสองวิธี

2.2.2 ชนิดมีเครื่องหมาย (Signed Integer) แบบ 2-Complement

เลขจำนวนเต็มชนิดมีเครื่องหมายในคอมพิวเตอร์ เหมาะสำหรับการใช้แทนข้อมูลที่มีค่าทั้งบวกและลบ บน เส้นจำนวน ซึ่งจำนวนจะมีค่าเริ่มต้น จนถึงจำนวนที่ฮาร์ดแวร์และซอฟต์แวร์รองรับ

นิยามที่ 2.2.2. กำหนดให้ เลขฐานสอง $X_{2,s}$ เป็นเลขจำนวนเต็มชนิดมีเครื่องหมาย (Unsigned Integer) แบบ 2-Complement เขียนอยู่ในรูป

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

เมื่อ x_{n-1} ทำหน้าที่เป็นบิตเครื่องหมาย (Sign bit) และ x_i คือค่า “1” หรือ “0” ในตำแหน่งที่ i และ ตำแหน่งขวามีสุดคือตำแหน่งที่ $i = 0$

การแปลงเลขฐานสองเป็นฐานสิบ

การแปลงเลขฐานสองแบบ 2-Complement จากนิยามที่ 2.2.2 ให้เป็นค่าฐานสิบแบบมีเครื่องหมาย สามารถทำได้โดย

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

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

ตัวอย่างที่ 2.2.7. เมื่อ $n = 4$ บิต เลขฐานสองแบบ 2 Complement $X_{2,s} = 1011_2$ มีค่าฐานสิบเท่ากับ เท่าไหร่

ค่าฐานสิบของ $X_{2,s}$ คือ

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

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

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

เลขจำนวนเต็มขนาด $n = 4$ บิตจะมีค่าฐานสิบอยู่ในช่วง -8 ถึง $+7$

จากตัวอย่างที่ 2.2.7 เลขฐานสองความยาว $n=4$ บิตสามารถตีความแบบมีเครื่องหมายและแบบไม่มี เครื่องหมาย ตามสมการที่ 2.16 และ สมการที่ 2.2 ตามลำดับ ในตารางที่ 2.4

Table 2.4: รูปแบบ (Pattern) ของเลขฐานสองขนาด $n=4$ บิตทั้งหมด $2^4=16$ แบบและการตีความหมายให้เป็นค่าฐานสิบแบบมีเครื่องหมายและแบบไม่มีเครื่องหมาย

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

เลขฐานสองความยາ $n=5$ บิตสามารถตีความแบบมีเครื่องหมายและแบบไม่มีเครื่องหมาย ตามสมการที่ 2.16 และ 2.2 ตามลำดับ ในตารางที่ 2.5

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

ค่าฐานสิบของ $X_{2,s}$ คือ

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

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

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

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

ตัวอย่างที่ 2.2.9. เมื่อ $n = 8$ บิต เลขฐานสองแบบ 2 Complement $X_{2,s} = 11111011_2$ มีค่าฐานสิบเท่ากับเท่าไหร่

ค่าฐานสิบของ $X_{2,s}$ คือ

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

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

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

Table 2.5: ค่าฐานสิบแบบมีเครื่องหมายและแบบไม่มีเครื่องหมายของเลขฐานสองความยาว $n=5$ บิต
ทั้งหมด $2^5=32$ แบบ

เลขฐานสอง $n=5$ บิต	$X_{10,s}$ ค่าฐานสิบ 2-Complement	$X_{10,u}$ ค่าฐานสิบ Unsigned
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

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

เลขฐานสองมีเครื่องหมายแบบ 2-Complement เลขฐานสองความยาว $n=8$ บิตสามารถถือความแบบมีเครื่องหมายและแบบไม่มีเครื่องหมาย ตามสมการที่ 2.16 และ 2.2 ตามลำดับ คล้ายกับกรณี $n=5$ บิต ในตารางที่ 2.6

ผู้อ่านจะสังเกตเห็นได้ว่า 1111 ด้านซ้ายของ 1111 1011₂ ในตารางนี้ ทำหน้าที่เหมือน Sign bit โดยเราเรียกปรากฏการณ์นี้ว่า Sign Extension ซึ่งสามารถประยุกต์ใช้กับเลขมีเครื่องหมายได้ทุกความยาว

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

ค่าฐานสิบของ $X_{2,s}$ คือ

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

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

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

เลขจำนวนเต็มขนาด $n = 16$ บิตจะมีค่าฐานสิบอยู่ในช่วง -32,768 ถึง +32,767

Table 2.6: ค่าฐานสิบชนิดมีเครื่องหมายและไม่มีเครื่องหมายของเลขฐานสองความยาว $n=8$ บิตทั้งหมด $2^8=256$ แบบ

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

ค่าฐานสิบของ $X_{2,s}$ คือ

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

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

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

เลขจำนวนเต็มขนาด $n = 32$ บิตจะมีค่าฐานสิบอยู่ในช่วง $-2,147,483,648$ ถึง $+2,147,483,647$

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

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 2.32

Table 2.7: การแปลงค่าฐานสิบแบบมีเครื่องหมายให้เป็นเลขฐานสองความยาว $n=4$ บิต โดยแปลงให้เป็นค่าฐานสิบโดยใช้สูตร $2^n + X_{10,s}$ กรณีที่ $X_{10,s} < 0$ ตามสมการที่ 2.32

$X_{10,s} < 0$	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 เลขฐานสอง $X_{2,sm}$ เป็นเลขจำนวนเต็มชนิดมีเครื่องหมาย (Signed Integer) แบบ Sign-Magnitude 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 และตำแหน่งของ x_0 มีอสุดคือตำแหน่งที่ $i = 0$

การแปลงเลขฐานสองแบบ Sign-Magnitude ให้เป็นค่าฐานสิบแบบมีเครื่องหมายสามารถทำได้โดย

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

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

จากนิยามที่ 2.2.1 และนิยามที่ 2.2.3 เลขฐานสองชนิดมีเครื่องหมาย แบบ Sign Magnitude สามารถเขียนใหม่ในรูปของเลขฐานสองไม่มีเครื่องหมายเป็น

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

ที่ความยาว $n - 1$ บิต

ตัวอย่างที่ 2.2.12. เมื่อ $n = 4$ บิต $X_{2,sm} = 1011_2$ 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 $n=4$ บิตสามารถถือความแบบมีเครื่องหมายและแบบไม่มีเครื่องหมาย ตามสมการที่ 2.35, 2.16 และ 2.2 ตามลำดับ ในตารางที่ 2.8 ตรงกับรูปที่ 2.4 เช่นกัน

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

ตารางนี้ตรงกับรูปที่ 2.4 รูปแบบของ Sign-Magnitude นิยมใช้งานร่วมกับข้อมูลชนิดเลขทศนิยม ซึ่งจะกล่าวต่อไปในหัวข้อที่ 2.4 และหัวข้อที่ 2.5

2.3 คณิตศาสตร์เลขจำนวนเต็ม

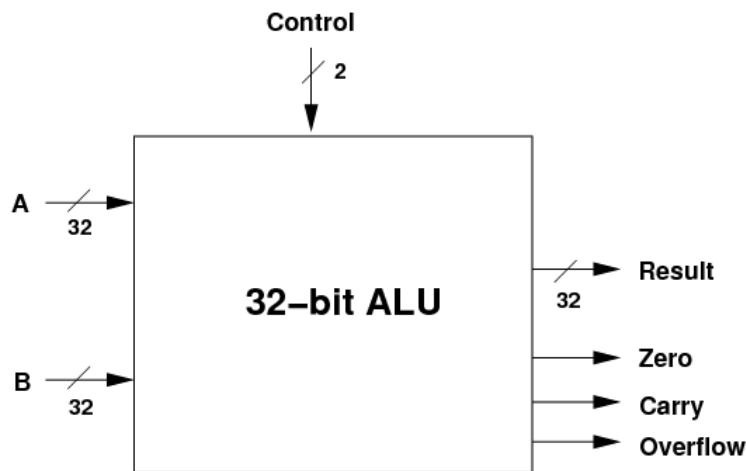


Figure 2.5: สัญลักษณ์ ALU (Arithmetic Logic Unit) สำหรับเลขจำนวนเต็มขนาด 32 บิต

รูปที่ 2.5 แสดงสัญลักษณ์ ALU (Arithmetic Logic Unit) สำหรับเลขจำนวนเต็มขนาด 32 บิต มีอินพุท A และ B ขนาด 32 บิต สำหรับนำข้อมูลชนิดจำนวนเต็มจำนวน 2 ตัว สัญญาณ Control เพื่อควบคุมการทำงาน เช่น บวก ลบ คูณ หาร เป็นต้น มีเอาท์พุท คือ C ขนาด 32 บิต ขาเครื่องหมาย N (Negative) ขา Carry c_n ขา Overflow (V)

ขาสัญญาณเหล่านี้จะบันทึกลงในรีจิสเตอร์สถานะ สำหรับให้โปรแกรมเมอร์ตรวจสอบ ด้วยภาษาแอกซ์เเชมบลี ในบทที่ 4

2.3.1 ชนิดไม่มีเครื่องหมาย

เลขจำนวนเต็มฐานสองชนิดไม่มีเครื่องหมาย $X_{2,u}$ และ $Y_{2,u}$ ความยาว n บิต ตามนิยามที่ 2.2.1 สามารถบวกกันได้ การบวกเลขขนาด 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	

ในการใช้คณิตศาสตร์ การบวกเลข 3 บิตจะทำให้เกิดผลลัพธ์จำนวน 2 บิต เรียกว่าติดกัน 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 กันเขิงตຽรรคศาสตร์ ในเขิงตຽรรคศาสตร์

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

เมื่อ \oplus คือ กระบวนการ Exclusive-OR

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

เมื่อ & คือ กระบวนการ AND และ | คือ กระบวนการ OR วงจรสามารถตรวจจับการเกิดโอเวอร์โฟล์วได้โดย

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

การบวกเลขสองจำนวนและการตรวจจับโอเวอร์โฟล์ว

การบวกเลขชนิดไม่มีเครื่องหมาย 2 จำนวน ผลลัพธ์ที่ได้จะไม่มีเครื่องหมายด้วยเช่นกัน แต่การบวกเลขขนาดใหญ่ที่เข้าใกล้ค่าสูงสุด สามารถเกิดความผิดพลาดได้ เรียกว่า การเกิดโอเวอร์โฟล์ว (Overflow) ในสมการที่ 2.43 ซึ่งเป็นผลลัพธ์เนื่องจากวงจรดิจิทัลที่สามารถประมวลผลได้จำกัด ตามจำนวนบิตข้อมูลสูงสุดที่ทำได้ ในตัวอย่างการแปลงเลขฐานสองเป็นฐานสิบที่ได้แสดงไปแล้ว ยกตัวอย่าง เช่น การวนรอบหรือวนลูป (Loop) เพิ่มค่าอย่างต่อเนื่องโดยไม่ระวัง ตามประโยคในภาษา C/C++ i++ หรือ i=i+1 นี้ หาก i มีค่าเพิ่มขึ้นเรื่อยๆ จะเข้าใกล้ค่าสูงสุด การบวกเพิ่มอีก 1 ไปเรื่อยๆ โดยไม่มีการตรวจจับล่วงหน้า จะทำให้เกิดโอเวอร์โฟล์วในที่สุด ซึ่งอาจทำให้เกิดผลลัพธ์ตามมาอย่างรุนแรง

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

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

	c_4	c_3	c_2	c_1	c_0	OVF/Valid
	0	0	0	1	0	No
$X=5$		0	1	0	1	
$+Y=9$		1	0	0	1	
$Z=14$		1	1	1	0	Valid

ตัวอย่างที่ 2.3.2. จงคำนวณหาค่าของ $7 + 9$ ด้วยเลขจำนวนเต็มชนิดไม่มีเครื่องหมาย 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	OVF/Valid
	1	1	1	1	0	$OVF=c_n=1$
$X=7$		0	1	1	1	
$+Y=9$		1	0	0	1	
$Z=16$		0	0	0	0	Invalid

สาเหตุของการเกิด Overflow เนื่องจากผลลัพธ์มีค่าอยู่นอกช่วงที่เป็นไปได้ โดยสามารถตรวจสอบอย่างง่ายดายโดย $c_4 = 1$ (OVF: Overflow) เมื่อเกิดโอเวอร์โฟล์ว ผลลัพธ์ที่ได้จึงมีค่าไม่ถูกต้อง (Invalid)

การคูณเลขสองจำนวน

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

การคูณเลขมีความซับซ้อนสูง جداเป็นต้องใช้วงจรดิจิทัลและเวลาในคำนวณ การคูณเลขที่ง่ายที่สุดคือการบวกเลขแล้ววนบวกซ้ำ ตามวงจรที่จะอธิบายในรูปที่ 2.6 และ 2.7 ต่อไปนี้

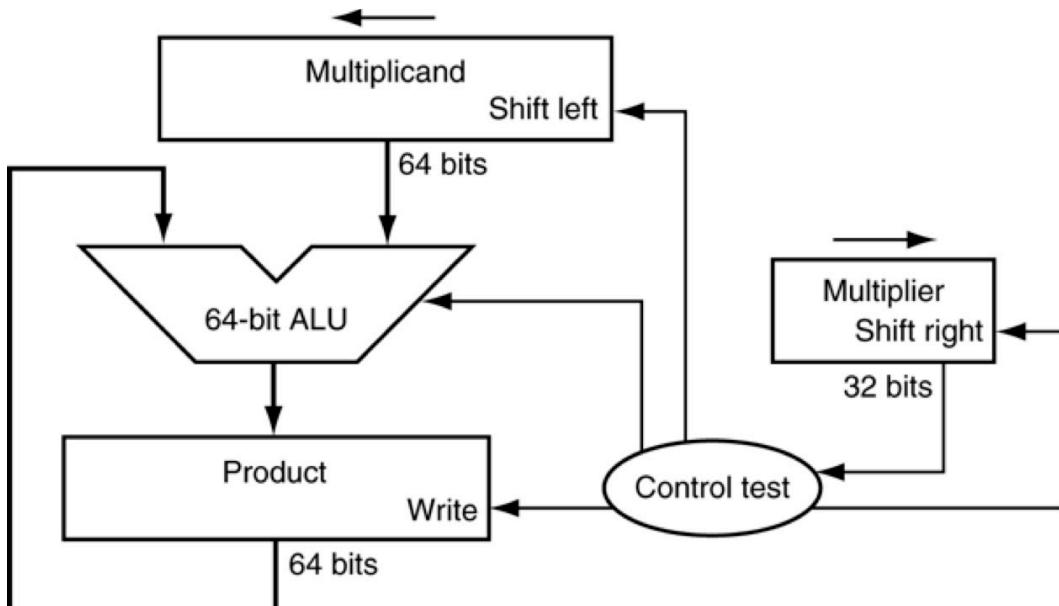


Figure 2.6: วงจรคูณเลขขนาด $n = 32$ บิต ชนิดที่ 1 โดยใช้ ALU ขนาด $2n = 64$ บิต และรีจิสเตอร์ตัวตั้งขนาด $2n = 64$ บิต (ที่มา: Patterson and Hennessy (2000))

วงจรในรูปที่ 2.6 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 = -)
3. เลื่อน (Shift) ข้อมูลในรีจิสเตอร์
 - เลื่อนรีจิสเตอร์ตัวตั้งไปทางซ้าย 1 บิต โดยป้อน 0 เข้าทางขวาไปแทนที่บิตที่ถูกเลื่อนไปทางขวา
 - 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 2.6

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

รอบ Iteration	ตัวตั้ง Multicand	ตัวคูณ Multiplier	ผลคูณ Product ₂	ผลคูณ Product ₁₀	Action
-	0000 1010 ₂	1101 ₂	0000 0000 ₂	0 ₁₀	Initialized
0	0000 1010 ₂	1101 ₂	0000 1010 ₂	10 ₁₀	Add
	0001 0100 ₂	0110 ₂	0000 1010 ₂	10 ₁₀	Shift
1	0001 0100 ₂	0110 ₂	0000 1010 ₂	10 ₁₀	-
	0010 1000 ₂	0011 ₂	0000 1010 ₂	10 ₁₀	Shift
2	0010 1000 ₂	0011 ₂	0011 0010 ₂	50 ₁₀	Add
	0101 0000 ₂	0001 ₂	0011 0010 ₂	50 ₁₀	Shift
3	0101 0000 ₂	0001 ₂	1000 0010 ₂	130 ₁₀	Add
	1010 0000 ₂	0000 ₂	1000 0010 ₂	130 ₁₀	Shift

วงจรคูณเลขชนิดที่ 2

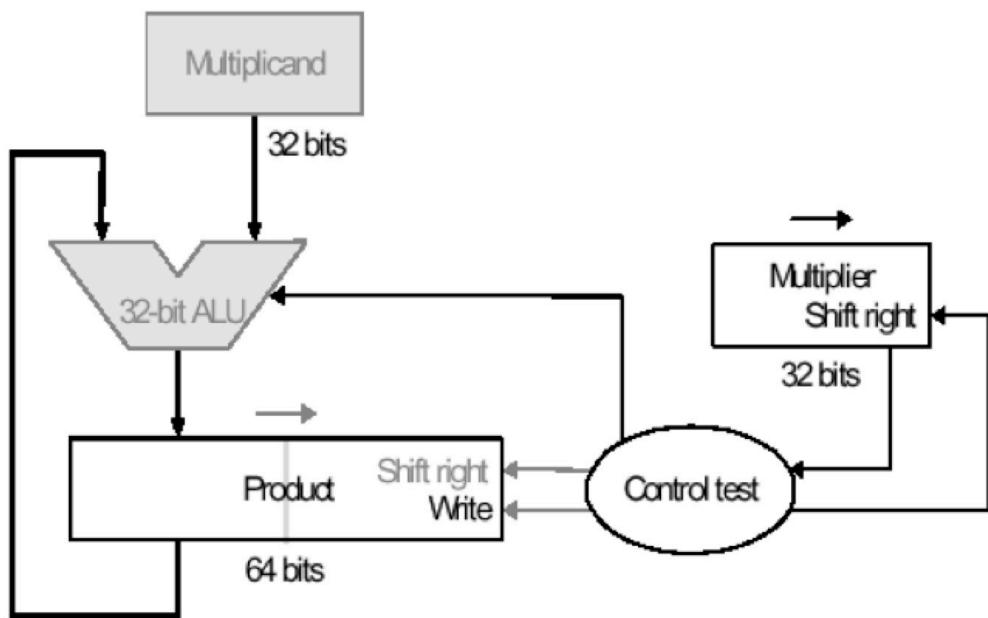


Figure 2.7: วงจรคูณเลขชนิด $n = 32$ บิต ชนิดที่ 2 โดยใช้ ALU ขนาด $n = 32$ บิต และรีจิสเตอร์ตัวตั้งขนาด $n = 32$ บิต (ที่มา: Patterson and Hennessy (2000))

วงจรในรูปที่ 2.7 ให้รีจิสเตอร์ตัวตั้ง (Multiplicand) และตัวคูณ (Multiplier) เป็นเลขจำนวนเต็มไม่음ีเครื่องหมายขนาด n บิตทั้งคู่ และผลคูณ (Product) จะมีขนาด $2n$ บิต มีการทำงานเป็นขั้นตอนดังนี้

- ตั้งค่าเริ่มต้นของรีจิสเตอร์ทุกตัว
 - ตั้งค่ารีจิสเตอร์ตัวตั้ง (Multiplicand) ในรีจิสเตอร์ขนาด n บิต

- ตั้งค่ารีจิสเตรอร์ตัวคูณ (Multiplier) ในรีจิสเตรอร์ขนาด n บิต
 - ตั้งค่ารีจิสเตรอร์ผลคูณ (Product) ขนาด $2n$ บิตให้เป็น 0 ทุกบิต
2. ตรวจสอบค่าบิตที่ 0 (ขวาสุด) ของรีจิสเตรอร์ตัวคูณ
 - หากมีค่าเป็น 1 บวกค่าตัวตั้งกับค่าผลคูณเข้าด้วยกัน (Action = Add)
 - หากมีค่าเป็น 0 ไม่ต้องบวก (Action = -)
 3. เลื่อน (Shift) ข้อมูลในรีจิสเตรอร์
 - เลื่อนรีจิสเตรอร์ตัวคูณและรีจิสเตรอร์ผลคูณไปทางขวา 1 บิต
 - โดยป้อน 0 เข้าทางซ้ายของรีจิสเตรอร์ตัวคูณ เพื่อไปแทนที่บิตที่ถูกเลื่อนไปทางขวา
 - แต่ป้อนค่า c_n กลับเข้ามาทางซ้ายของรีจิสเตรอร์ผลคูณ เพื่อไปแทนที่บิตที่ถูกเลื่อนไปทางขวา
 4. กลับไปทำข้อ 2 จนครบ n รอบ

ตัวอย่างที่ 2.3.4. จงคูณ 1010_2 (10_{10}) ด้วย 1101_2 (13_{10}) ตามวงจรที่ 2.7

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

2.7

รอบ Iteration	ตัวตั้ง Multicand	ตัวคูณ Multiplier	ผลคูณ Product ₂	ผลคูณ Product ₁₀	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
1	1010_2	0110_2	0101 0000 ₂	80_{10}	-
	1010_2	0011_2	0010 1000 ₂	40_{10}	Shift
2	1010_2	0011_2	1100 1000 ₂	200_{10}	Add
	1010_2	0001_2	0110 0100 ₂	100_{10}	Shift
3	1010_2	0001_2	1 0000 0100 ₂	260_{10}	Add
	1010_2	0000_2	1000 0010 ₂	130_{10}	Shift

จะเห็นได้ว่าที่รอบที่ 3 บิตที่เกิดขึ้นทางซ้ายสุดของผลคูณ จะถูกป้อนกลับเข้ามาทางซ้าย เพื่อให้ได้ผลลัพธ์ที่ถูกต้อง

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

2.3.2 ชนิดมีเครื่องหมายแบบ 2-Complement

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

ในเชิงคณิตศาสตร์

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

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

ในเชิงตรรกศาสตร์

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

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 วิธีสามารถตรวจสอบจับการเกิดโอเวอร์โฟล์วได้โดย

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

ขาสัญญาณเหล่านี้จะบันทึกลงในรีจิสเตอร์สถานะ สำหรับให้โปรแกรมเมอร์ตรวจสอบ ด้วยภาษาแอลซีดี ในการบวกเลขขนาด 4 บิต $7 - 6 = 7 + (-6)$ ดังนั้น ในเครื่องคอมพิวเตอร์ขนาด 4 บิต สามารถคำนวณได้ดังนี้

	c_4	c_3	c_2	c_1	c_0	OVF/Valid
	1	1	1	0	0	No
$X=7$		0	1	1	1	
$+ -Y=-6$		1	0	1	0	
$Z=1$		0	0	0	1	Valid

ไม่เกิด Overflow เมื่อผลลัพธ์มีค่าอยู่ในย่านที่เป็นไปได้ และ $c_4 \text{ xor } c_3 = 1 \text{ xor } 1 = 0$ (No Overflow) ผลลัพธ์ที่ได้จึงมีค่าถูกต้อง (Valid)

ตัวอย่างที่ 2.3.6. 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	OVF/Valid
	1	0	0	0	0	โอเวอร์โฟล์
$X = -3$		1	1	0	1	
$+ -Y = -6$		1	0	1	0	
$Z = 7$		0	1	1	1	Invalid

สาเหตุของการเกิด Overflow เนื่องจากผลลัพธ์มีค่าอยู่นอกย่านที่เป็นไปได้ โดยสามารถตรวจสอบ อย่างง่ายดายโดย $c_4 \text{ xor } c_3 = 1 \text{ xor } 0 = 1$ (OVF: Overflow) เมื่อเกิดโอเวอร์โฟล์ ผลลัพธ์ที่ได้จึงมีค่าไม่ ถูกต้อง (Invalid)

2.4 เลขทศนิยมชนิด Fixed Point

เลขฐานสองชนิดจำนวนจริงแบ่งเป็นชนิดจุดทศนิยมคงที่ (Fixed-Point Real Number) และ ชนิดจุดทศนิยมลอยตัว (Floating-Point Real Number) ในหัวข้อนี้ ผู้อ่านจะได้ทำความเข้าใจเรื่องของรูปแบบ (Representation) ชนิดจุดทศนิยมคงที่ ส่วนการบวกและลบ เลขทศนิยมชนิดนี้จะมีความคล้ายกับเลขจำนวนเต็ม ชนิด Sign-Magnitude

นิยามที่ 2.4.1. กำหนดให้ เลขฐานสอง $F_{2,u}$ เป็นเลขทศนิยมชนิด Signed Magnitude เขียนอยู่ในรูป

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

นิยามให้ชนิดขนาด-เครื่องหมาย ประกอบด้วย 3 ส่วน คือ บิตเครื่องหมาย (Sign bit: s) ค่าจำนวนเต็ม (Integer: $X_{2,u}$) มีความยาว n บิต และค่าทศนิยม (Fragment: Y_2) ยาว m บิต

โดยค่าฐานสิบ F_{10} ของเลข 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. จงแปลงเลขฐานสิบต่อไปนี้เป็นเลขฐานสอง Fixed-Point

$$+0.75_{10} = 0\dots00.11_2 \quad (2.51)$$

$$+3.00_{10} = 0\dots11.00_2 \quad (2.52)$$

$$-3.75_{10} = 10..11.11_2 \quad (2.53)$$

เลขทศนิยมชนิดจุดคงที่ สามารถเขียนอยู่ในรูปของเลขฐานสองสามารถเขียนในรูปแบบนี้

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

ค่าแฟร์กชัน (Fraction: Y_2) มีความยาว m บิต เขียนเป็นสัญลักษณ์ได้ดังนี้

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

ยิ่งยาวมาก ยิ่งมีความละเอียดเพิ่มขึ้น ตามตัวอย่างต่อไปนี้

ตัวอย่างที่ 2.4.2. จงแปลงเลขฐานสอง Fixed-Point ต่อไปนี้เป็นเลขฐานสิบ

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

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

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

การบวก/ลบเลข การคูณเลข Fixed Point มีความคล้ายกับเลขจำนวนเต็ม

2.5 เลขทศนิยมชนิด Floating Point

เลขทศนิยมชนิด Floating Point เหมาะสำหรับข้อมูลที่มีพิสัย (Range) กว้างและเลขทศนิยมที่ต้องการความละเอียดสูง สำหรับการคำนวณทางวิทยาศาสตร์ (Scientific) ดังนี้

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

นิยามที่ 2.5.1. เลขทศนิยมชนิด Floating Point ฐานสองที่อยู่ในรูปน่อมัลไล์ซ์มีลักษณะดังนี้

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

ประกอบด้วย 3 ส่วน คือ บิตเครื่องหมาย (Sign bit: s) ค่านัยสำคัญ (Significant: S_2) และค่ายกกำลัง (Exponent: E_2) เป็นเลขจำนวนเต็มฐานสองชนิด sign-magnitude ความยาว n บิต ดังนี้

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

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

เลขฐานสองที่น่อมัลไล์ซ์แล้วสามารถคำนวณหาค่าที่ตรงกันในเลขฐานสิบ ได้ดังนี้

$$F_{10} = \pm\left[1 + \frac{y_{-1}}{2} + \frac{y_{-2}}{4} + \frac{y_{-3}}{8} + \dots + \frac{y_{-m}}{2^m}\right] \times (2^{\pm E_2}) \quad (2.61)$$

จากนิยามที่ 2.59 ค่านัยสำคัญ (Significant: S_2) ความยาว $m+1$ บิต สามารถเขียนใหม่ได้ดังนี้

$$S_2 = [1.y_{-1}y_{-2}y_{-3}\dots y_{-m}]_2 \quad (2.62)$$

ซึ่งมีความสำคัญต่อรูปแบบการเขียน เนื่องจาก วงจรจะต้องทำการน่อมัลไล์ซ์เพลลัพท์ที่ได้การคำนวณเสมอ

ตัวอย่างที่ 2.5.1. จงแปลงเลขฐานสองแบบ Floating-Point ที่น่อมัลไล์ซแล้วให้เป็นเลขฐานสิบ วิธีทำ

สมมติว่ามีเลขฐานสองชนิด Floating Point แบบน่อมัลไล์ซขนาด $n=4$ บิต (4-bit)

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

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

2. แปลงค่าเลขฐานสองชนิดที่เลื่อนตำแหน่งแล้วให้เป็นฐานสิบ
 $-\{1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 + 1 \times 2^{-1}\} = -10.5$

ตัวอย่างที่ 2.5.2. จงแปลงเลขฐานสองแบบ Floating-Point ที่น้อมัลไล์ซ์แล้วให้เป็นเลขฐานสิบ วิธีทำ

สมมติว่ามีเลขฐานสองชนิด Floating Point ที่น้อมัลไล์ซขนาด $n=4$ บิต (4-bit)

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

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

2. แปลงค่าเลขฐานสองชนิดที่เลื่อนตำแหน่งแล้วให้เป็นฐานสิบ

$$\begin{aligned} & (1 \times 2^{-3}) + (1 \times 2^{-4}) + (1 \times 2^{-6}) \\ & = 0.125 + 0.0625 + 0.015625 \\ & = 0.203125_{10} \end{aligned}$$

2.6 การบวกเลข Floating-Point

เพื่อให้ผู้อ่านเข้าใจกลไกการบวกเลขฐานสองแบบ Floating-Point โดยใช้ตัวอย่างจากเลขฐานสิบก่อน แล้ว จึงนำขั้นตอนที่เหมือนกันไปประยุกต์ใช้กับเลขฐานสองในตัวอย่างต่อไป

ตัวอย่างที่ 2.6.1. กำหนดให้เลขฐานสิบทั้งสองตัวมีขนาดไม่เกิน 4 หลัก (4-digit) จงบวกเลขฐานสิบแบบ Floating-Point ที่น้อมัลไล์ซแล้ว ดังต่อไปนี้ $9.999 \times 10^1 + 1.610 \times 10^{-1}$

วิธีทำ

- เลื่อนตำแหน่งจุดศูนย์และปรับเลขยกกำลังที่มีค่าน้อยกว่า ให้ตรงกับเลขยกกำลังของเลขที่มีค่ามากกว่า

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

- บวกค่า significands ที่เลื่อนตำแหน่งแล้ว

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

- น้อมัลไล์ซค่าผลลัพธ์และตรวจเช็คการเกิดโอเวอร์โฟล์วหรืออันเดอร์โฟล์ว
 1.0015×10^2

- ปัดค่าให้เหลือ 4 หลักและอาจต้องน้อมัลไล์ซเมื่อจำเป็น

$$1.002 \times 10^2$$

จะเห็นได้ว่า ผลลัพธ์ที่ได้จะเกิดความคลาดเคลื่อนจากค่าที่ควรจะเป็น เนื่องจากมีการจำกัดจำนวนดิจิท

ของผลลัพธ์ให้เหลือ 4 หลักตามโจทย์

ตัวอย่างที่ 2.6.2. จงบวกเลขฐานสองแบบ Floating-Point ที่น้อมลัลเลอร์แล้ว ดังต่อไปนี้ วิธีทำ

สมมติว่ามีเลขฐานสองขนาด 4 บิต (4-bit)

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

- เลื่อนตำแหน่งจุดศูนย์ และปรับเลขยกกำลังที่มีค่าน้อยกว่า ให้ตรงกับเลขยกกำลังของเลขที่มีค่ามากกว่า

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

- บวกค่า significands ที่เลื่อนตำแหน่งแล้ว

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

- น้อมลัลเลอร์ค่าผลลัพธ์และตรวจเช็คการเกิดโอเวอร์โฟล์วหรืออันเดอร์โฟล์ว

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

- ปัดค่าให้เหลือ 4 บิตและอาจต้องน้อมลัลเลอร์เมื่อจำเป็น

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

2.7 การคูณเลข Floating-Point

เพื่อให้ผู้อ่านเข้าใจกลไกการคูณเลขฐานสองแบบ Floating-Point เราจะใช้ตัวอย่างจากเลขฐานสิบก่อนแล้วจึงนำขั้นตอนที่เหมือนกันไปประยุกต์ใช้กับเลขฐานสองในตัวอย่างต่อไป

ตัวอย่างที่ 2.7.1. จงคูณเลขฐานสิบแบบ Floating-Point ที่น้อมลัลเลอร์ ดังต่อไปนี้

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

- บวก exponents เข้าด้วยกัน

$$\text{ค่า exponent ใหม่ } = 10 + -5 = 5$$

- คูณค่า significands เข้าด้วยกัน

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

- น้อมลัลเลอร์ค่าผลลัพธ์ และตรวจเช็คการเกิดโอเวอร์โฟล์วหรืออันเดอร์โฟล์ว

$$1.0212 \times 10^6$$

4. ปัดค่าให้เหลือ 4 หลักและอาจต้องนำมัลไล์ซ์เมื่อจำเป็น

$$1.021 \times 10^6$$

5. ปรับเครื่องหมายของผลลัพธ์ให้ถูกต้องจากตัวตั้งและตัวคูณ

$$+1.021 \times 10^6$$

จะเห็นได้ว่า ผลลัพธ์ที่ได้จะเกิดความคลาดเคลื่อนจากค่าที่ควรจะเป็น เนื่องจากมีการจำกัดจำนวนดิจิท ของผลลัพธ์ให้เหลือ 4 หลักตามโจทย์

ตัวอย่างที่ 2.7.2. จงคูณเลขฐานสองแบบ Floating-Point ขนาด 4 บิต (4-bit) ที่นำมัลไล์ซ์ 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. บวก exponents เข้าด้วยกัน

ค่า exponent ใหม่: $-1 + -2 = -3$

2. คูณค่า significands เข้าด้วยกัน

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

3. นำมัลไล์ซ์ค่าผลลัพธ์ และตรวจเช็คการเกิดโอเวอร์โฟล์วหรืออันเดอร์โฟล์ว

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

4. ปัดค่าให้เหลือ 4 หลักและอาจต้องนำมัลไล์ซ์เมื่อจำเป็น

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

5. ปรับเครื่องหมายของผลลัพธ์ให้ถูกต้องจากตัวตั้งและตัวคูณ

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

2.8 เลขทศนิยมชนิดจุดลอยตัวมาตรฐาน IEEE 754

มาตรฐานของเลข Floating Point ได้ถูกกำหนดโดย IEEE Standard 754 ในปี ค.ศ.1985 เพื่อให้โปรแกรมสามารถคำนวณค่าเลขทศนิยมบนเครื่องที่ใช้ชิปปิคิก็ได้โดยให้ผลลัพธ์เดียวกัน 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 2008

เมื่อผู้อ่านทำความเข้าใจทฤษฎีและตัวอย่างการคำนวณเบื้องต้นแล้ว ผู้อ่านสามารถทำการทดลองเพิ่มเติมใน การทดลองที่ 1 ภาคผนวก A ข้อมูลและคณิตศาสตร์ในคอมพิวเตอร์ หัวข้อที่ A.2

2.9 รูปแบบของเลข IEEE 754

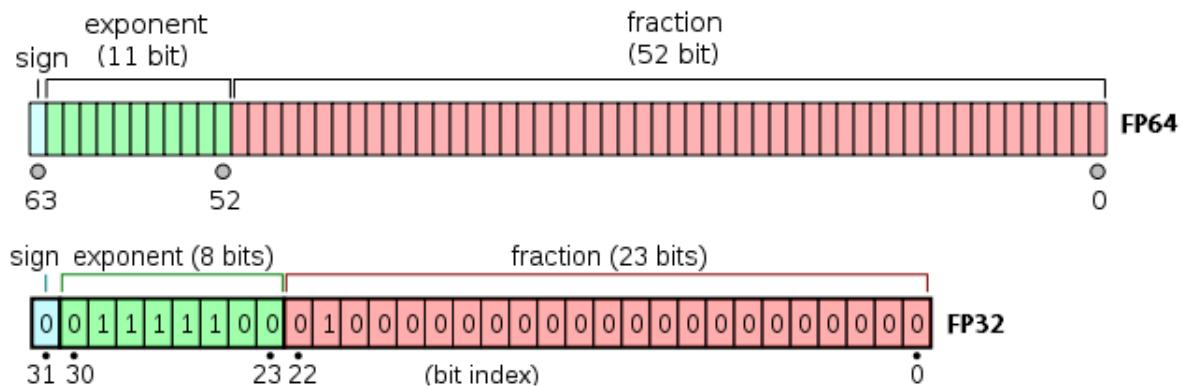


Figure 2.8: ตัวเลขชนิดจำนวนทศนิยมโดยตัวตามมาตรฐาน IEEE 754 Double-Precision และ Single-Precision ที่มา: <http://pybugs.com/single-precision-vs-double-precision/>

นิยามที่ 2.9.1. เลขทศนิยมชนิดจุดลอยตัวที่น้อมล้ำเลี้ยว สามารถเขียนอยู่ในรูปของเลขฐานสองสามารถเขียนในรูปแบบนี้

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

โดยค่า Fraction: Y_2 มีความยาว $m=23$ และ 51 บิต ตามชนิด Single Precision และ Double Precision ตามลำดับ สามารถเขียนเป็นสัญลักษณ์คล้ายกับเลขทศนิยมชนิดจุดคงที่ 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 2.62 ในหัวข้อที่ 2.5

ค่ายกกำลัง เป็นเลขจำนวนเต็มชนิดไม่มีเครื่องหมาย ยาว $n=8$ และ 11 บิต ตามชนิด Single Precision และ Double Precision ตามลำดับ โดยมีลักษณะคล้ายกับเลขทศนิยมชนิดจุดลอยตัวในสมการที่ 2.60 แต่

ต่างกันที่เลขยกกำลังของ IEEE 754 มีค่าเป็นเลขไม่มีเครื่องหมาย

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

โดยรายละเอียดเป็นดังนี้

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

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

- เพื่อปรับให้ Exponent เป็นเลขที่ไม่มีเครื่องหมาย (unsigned)
 - ชนิด Single Precision ค่า $E_{bias} = 127_{10}$
 - ชนิด Double Precision ค่า $E_{bias} = 1023_{10}$

เลขฐานสองชนิดทศนิยม IEEE 754 ในสมการที่ 2.63 สามารถแปลงเป็นค่าที่ตรงกัน ในรูปของเลขฐานสิบ ได้ดังนี้

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

ตัวอย่างที่ 2.9.1. หมายเลข FP-32 ที่เติมในรูปที่ 2.8 คือ เลขทศนิยมมาตรฐาน IEEE 754 ชนิด Single Precision ดังต่อไปนี้

$$0][0111\ 1110\ 0][010\ 0000\ 0000\ 0000\ 0000]_2$$

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

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

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

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

$$= 0.125_{10} + 0.03125_{10} = 0.15625_{10} \quad (2.72)$$

ตัวอย่างที่ 2.9.2. จงแปลงเลข -0.75_{10} เป็นเลขฐานสอง Floating-Point ตามมาตรฐาน IEEE 754 ทั้งสองชนิด

วิธีทำ

$$-0.75_{10} = (-1)^1 \times 0.11_2 \times 2^0$$

ทำการนอเมลไลซ์ ตามสมการที่ 2.59

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

ดังนั้น $s = 1$

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

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

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

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

ดังนั้น -0.75_{10} เขียนในรูปของเลขศนิยมชนิดลอยตัวตามมาตรฐาน IEEE 754 ชนิด

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

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

ตัวอย่างที่ 2.9.3. เลข IEEE 754 Floating-Point ชนิด Single-Precision ต่อไปนี้ มีค่าเท่าไรในฐานสิบ

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

วิธีทำ

แปลงจากเลขฐานสิบหากให้เป็นฐานสองและองค์ประกอบต่างๆ ได้ดังนี้

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

จะพบว่าบิตเครื่องหมาย $s = 1$

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

$$\text{ค่าของ Fraction } Y_2 = 010\ 0000\ 0000\ 0000\ 0000_2$$

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

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

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

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

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

ตัวอย่างที่ 2.9.4. เลข IEEE 754 Floating-Point ชนิด Single-Precision ต่อไปนี้ มีค่าเท่าไรในฐานสิบ

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

วิธีทำ

แปลงจากเลขฐานสิบหากให้เป็นฐานสองและองค์ประกอบต่างๆ ได้ดังนี้

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

จะพบว่าบิตเครื่องหมาย $s = 0$

ค่าของ True Exponent $E_{true} = 1000\ 0011_2 - E_{bias} = 131 - 127 = 4$

ค่าของ Fraction $Y_2 = 011\ 0000\ 0000\ 0000\ 0000_2$

$$F_{10, IEEE} = (-1)^0 \times (1 + .011_2)2^4 \quad (2.78)$$

$$= (-1)^0 \times (1.011_2) \times 2^4 \quad (2.79)$$

$$= (-1)^0 \times (10110.0_2) \quad (2.80)$$

$$= (+1) \times (16 + 4 + 2) \quad (2.81)$$

$$= 22.0_{10} \quad (2.82)$$

2.10 ค่าสูงสุด ต่ำสุดและค่าอื่นๆ ของเลข Floating Point

การคำนวณในเครื่องคอมพิวเตอร์สามารถรองรับการคำนวณที่หลากหลาย โดยใช้เลขทศนิยม Floating-Point ตามมาตรฐาน IEEE 754 แต่เนื่องจากเลขทศนิยมในโลกนี้มีจำนวนอนันต์ (Infinite) เลขทศนิยม ตามมาตรฐาน IEEE 754 ไม่สามารถรองรับได้ทั้งหมด และมีข้อจำกัดทางคณิตศาสตร์ สรุปในตารางที่ 2.9

Table 2.9: ตารางสรุปความแตกต่างระหว่างเลข Floating-Point ตามมาตรฐาน IEEE 754 ชนิด Single Precision (x หมายถึง ข้อมูล '0' หรือ '1')

Sign s (1 บิต)	Exponent E_2 (8 บิต)	Fraction Y_2 (23 บิต)	ความหมาย
±	0000 0001 ₂ - 1111 1110 ₂	xx.. ₂	เลขฐานสิบห้าไป (นอมัลไลซ์)
±	0000 0000 ₂	xx.. ₂	เลขฐานสิบห้าอยมาก แต่ไม่เท่ากับศูนย์ (ดีนมัลไลซ์)
0	0000 0000 ₂	0...0 ₂	0.0 ₁₀
±	1111 1111 ₂	0...0 ₂	±∞
0	1111 1111 ₂	xx.. ₂	Nan (Not a Number)

โดย E_2 คือ Exponent หรือค่ายกกำลัง

Y_2 คือ Fraction เป็นส่วนประกอบของ significand หรือค่านัยสำคัญ

- ❖ แவบนสุด คือ ย่านของเลขทศนิยมที่มาตรฐานที่สามารถแสดงค่าได้

โดยมีค่าอยู่ระหว่างค่าต่ำสุดที่เข้าใกล้ศูนย์ จนถึง ค่าที่มากแต่ยังไม่ใช่ค่าอินฟินิตี้ ซึ่งได้กล่าวไว้แล้ว

- ❖ แவที่สอง คือ เลขทศนิยมที่มาตรฐานนี้ไม่สามารถแสดงค่าได้ เนื่องจากมีค่าน้อยมาก
- ❖ แவที่สาม คือ เลขทศนิยม 0.0₁₀ หรือเรียกว่า True zero
- ❖ แவที่สี่ คือ ค่าบวกและลบอินฟินิตี้ สามารถใช้แทนผลลัพธ์ที่มีค่าสูงเกินไปจากการคำนวณทางคณิตศาสตร์
- ❖ แવที่ห้า คือ NaN ออกเสียงว่า แนน เป็นสัญลักษณ์ซึ่งไม่สามารถแทนได้ด้วยตัวเลข ย่อมาจากคำว่า Not a Number แสดงถึงความผิดพลาดที่อาจเกิดจากการเขียนโปรแกรมเพื่อใช้ตัวแปรหารค่าตัวตั้ง แต่ตัวแปรที่ใช้มีค่าที่เปลี่ยนแปลงไปจนกลายเป็น 0.0₁₀ เมื่อนำไปหารจึงเกิดสัญลักษณ์นี้ขึ้น หรืออาจเกิดจากผลลัพธ์ทางคณิตศาสตร์ที่มีค่าต่ำเกินไป เป็นต้น

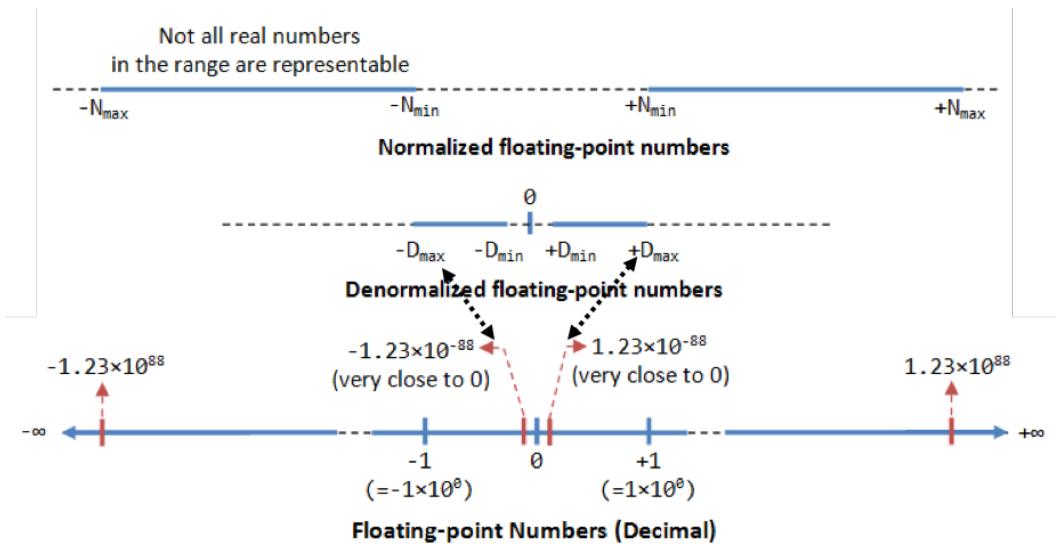


Figure 2.9: เลขทศนิยมโดยตัวชนิด omnal ไลซ์ตั้งแต่ $\pm N_{min}$ ถึง $\pm N_{max}$ และชนิดที่ไม่สามารถเขียนแบบ omnal ไลซ์ ตั้งแต่ $\pm D_{min}$ ถึง $\pm D_{max}$

รูปที่ 2.9 แสดงค่าบันเด็นจำนวนของ แควร์ต่างๆ ในตารางที่ 2.9 ยกเว้น NaN เส้นด้านบนของรูปแสดงเลขต่างๆ ที่เขียนในรูป omnal ไลซ์ได้ เส้นด้านล่างแสดงเลขขนาดเล็กมากๆ ที่ไม่สามารถเขียนในรูป omnal ไลซ์ได้ หรือ เลขดีโนมัล ไลซ์ รูปที่ 2.9 แสดงเลขต่างๆ ที่เขียนในรูป omnal ไลซ์ได้จากซ้ายสุดถึงขวาสุด ของเส้นจำนวน สำหรับ Double Precision แต่สำหรับ Single Precision มีค่าตั้งแต่ -3.4×10^{38} ถึง -1.175×10^{-38} และ $+1.175 \times 10^{-38}$ ถึง $+3.4 \times 10^{38}$

2.11 การบวกเลข Floating-Point ตามมาตรฐาน IEEE 754

การบวกเลขด้วยความเร็วสูง จำเป็นต้องอาศัยวงจรอยาร์ดแวร์ หรือ ALU สำหรับเลขทศนิยม (Floating Point ALU) ช่วยคำนวณ แบบไบพาสไลน์ (Pipeline) ซึ่งมีหลักการคล้ายกับ โรงงานประกอบรถยนต์ ทำให้ ซีพียู 1 คอร์ร์สามารถคำนวณการบวกเลขทศนิยมได้ในหลัก กิกะฟล็อปส์ (Giga Floating Point Operations Per Second) หรือ พันล้านหรือ 10^9 ครั้งต่อวินาที ซึ่งในอดีตจนถึงปัจจุบัน การวัดสมรรถนะ (Performance) ของซีพียู รวมไปถึงซูเปอร์คอมพิวเตอร์ (Super Computer) ยังคงใช้หน่วยวัด จิกะฟล็อปส์ และสูงถึง เพตาฟล็อปส์ (Peta Flops) หรือ 10^{15} ครั้งต่อวินาที ผู้อ่านสามารถศึกษาเรื่อง ฟล็อปส์ ได้ที่ <https://en.wikipedia.org/wiki/FLOPS>

ดังนั้น การทำงานของวงจรบวกเลขทศนิยม จึงอาศัยหลักการเดียวกันกับ อัลกอริทึมการบวก ซึ่ง มีความซับซ้อนใกล้เคียงกับการคูณเลขทศนิยม และทั้งคู่มีความซับซ้อนมากกว่าการบวกและการคูณ เลขจำนวนเต็มที่จำนวนบิตเท่ากัน

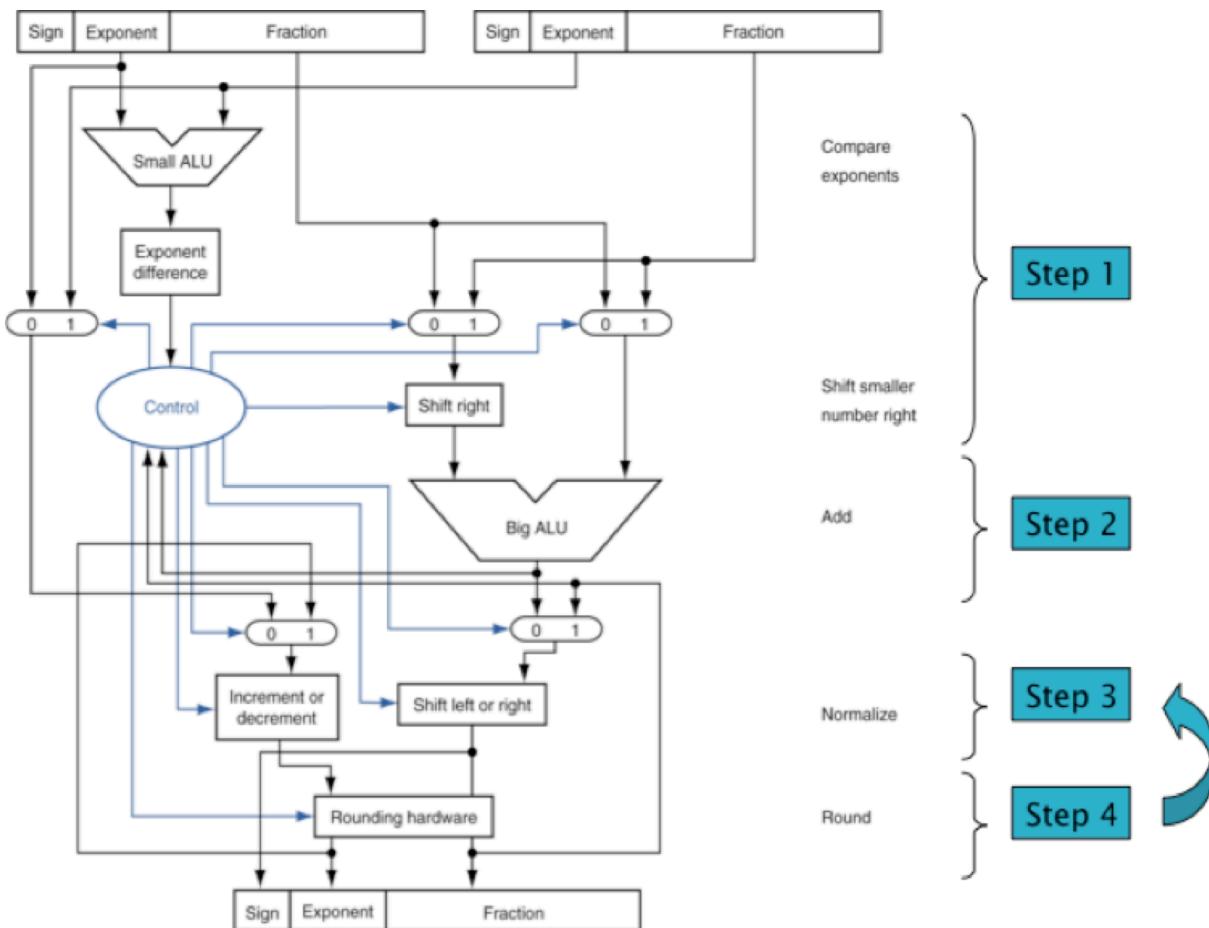


Figure 2.10: วิจารบกเลข Floating-Point ตามมาตรฐาน IEEE 754 โดยรับค่าอินพุทจำนวน 2 ตัวด้านบน และเอาท์พุทผลลัพธ์ด้านล่าง (ที่มา: Patterson and Hennessy (2000))

วงจรบกเลขฐานสองแบบ Floating-Point ในรูปที่ 2.10 มีความซับซ้อนใกล้เคียงกับตัวอย่างที่แสดงในหัวข้อที่ 2.5 ที่ผ่านมา ดังนี้

- ขั้นตอนที่ (Step) 1 คือ ปรับจุดศูนย์ให้ตรงกันโดยเลื่อนตำแหน่งเลขที่มีค่าน้อยกว่า โดยใช้การลบกันของค่ายกกำลังจากเลขทั้งสองตัว เมื่อได้ผลต่างก็จะทราบว่าเลขตัวใดมีค่ายกกำลังมากกว่าเท่าใด และตัวใดที่มีค่ายกกำลังน้อยก็จะถูกเลื่อน (Shift) ไปทางขวาเป็นจำนวนบิตเท่ากับค่าสัมบูรณ์ของผลต่าง
- ขั้นตอนที่ (Step) 2 คือ ทำการบวก/ลบค่า significand ที่เลื่อนตำแหน่งแล้วเข้าด้วยกัน โดยก่อนหน้านั้นค่า Fraction ของเลขทั้งสองตัวจะต้องบวกกับ 1 ที่ซ่อนอยู่ เพื่อปรับให้เป็นค่า significand ก่อน ตัวที่มีค่ายกกำลังน้อยกว่าจะถูกเลื่อนค่า significand ไปทางขวาเสมอ ส่วนค่า significand ที่มีค่ายกกำลังมากกว่าจะไม่เปลี่ยนแปลง
- ขั้นตอนที่ (Step) 3 คือน้อมลัลลีซ์ค่าผลลัพธ์ และตรวจเช็คการเกิดโอเวอร์โฟล์วหรืออันเดอร์โฟล์ว ซึ่งผลลัพธ์ที่ได้จากการบวกอาจมีค่าเพิ่มขึ้น หรือ ลดลงขึ้นอยู่กับเครื่องหมายและขนาดของเลขทั้งสองตัว ทำให้ต้องน้อมลัลลีอีกรอบ ระหว่างนั้น เมื่อค่าสัมบูรณ์ของผลลัพธ์เพิ่มมากขึ้นจนเกินค่าสูงสุด เรียกว่าเกิดโอเวอร์โฟล์ว หากค่าสัมบูรณ์ของผลลัพธ์มีค่าเข้าใกล้ศูนย์จนน้อยกว่าค่าต่ำสุดในตารางที่ 2.9 เรียกว่าเกิดอันเดอร์โฟล์ว จึงต้องใช้สัญลักษณ์ NaN แทนค่าผลลัพธ์
- ขั้นตอนที่ (Step) 4 คือ ปัดค่า Fraction ให้เหลือ 23 บิตและอาจต้องน้อมลัลลีเมื่อจำเป็น

ตัวอย่างที่ 2.11.1. การบวกเลขฐานสองชนิด Floating-Point (Single Precision)

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

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

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

วิธีทำ

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

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

1. เลื่อนตำแหน่งจุดศูนย์นิยมและปรับเลขยกกำลังที่มีค่าน้อยกว่า 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. บวกค่า significands ที่เลื่อนตำแหน่งแล้ว

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

3. น้อมัลไลซ์ค่าผลลัพธ์ และตรวจสอบการเกิดโอเวอร์โฟล์วหรืออันเดอร์โฟล์ว

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

4. ปัดค่าให้เหลือ 23 บิตและอาจต้องน้อมัลไลซ์เมื่อจำเป็น

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

$$s = 1$$

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

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

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

ตัวอย่างที่ 2.11.2. การบวกเลขฐานสองชนิด Floating-Point (Single Precision)

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

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

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

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

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

1. 1. เลื่อนตำแหน่งจุดทศนิยมและปรับเลขยกกำลังที่มีค่าน้อยกว่า ให้ตรงกับเลขยกกำลังของเลขที่มีค่ามากกว่า

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

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

2. บวกค่า significands ที่เลื่อนตำแหน่งแล้ว

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

3. น้อมัลไลซ์ค่าผลลัพธ์ และตรวจเช็คการเกิดโอเวอร์โฟล์วหรืออันเดอร์โฟล์ว

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

4. ปดค่าให้เหลือ 23 บิตและอาจต้องน้อมัลไลซ์เมื่อจำเป็น

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

$$s = 0$$

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

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

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

2.12 การคูณเลข Floating-Point ตามมาตรฐาน IEEE 754

การคูณเลขด้วยความเร็วสูง จำเป็นต้องอาศัยวงจรอาร์ดแวร์ แต่การทำงานของวงจรนี้ในหลักการเดียวกัน กับ อัลกอริทึมการคูณ ดังนั้น ผู้อ่านจำเป็นต้องศึกษาอัลกอริทึมให้เข้าใจอย่างถ่องแท้ก่อน

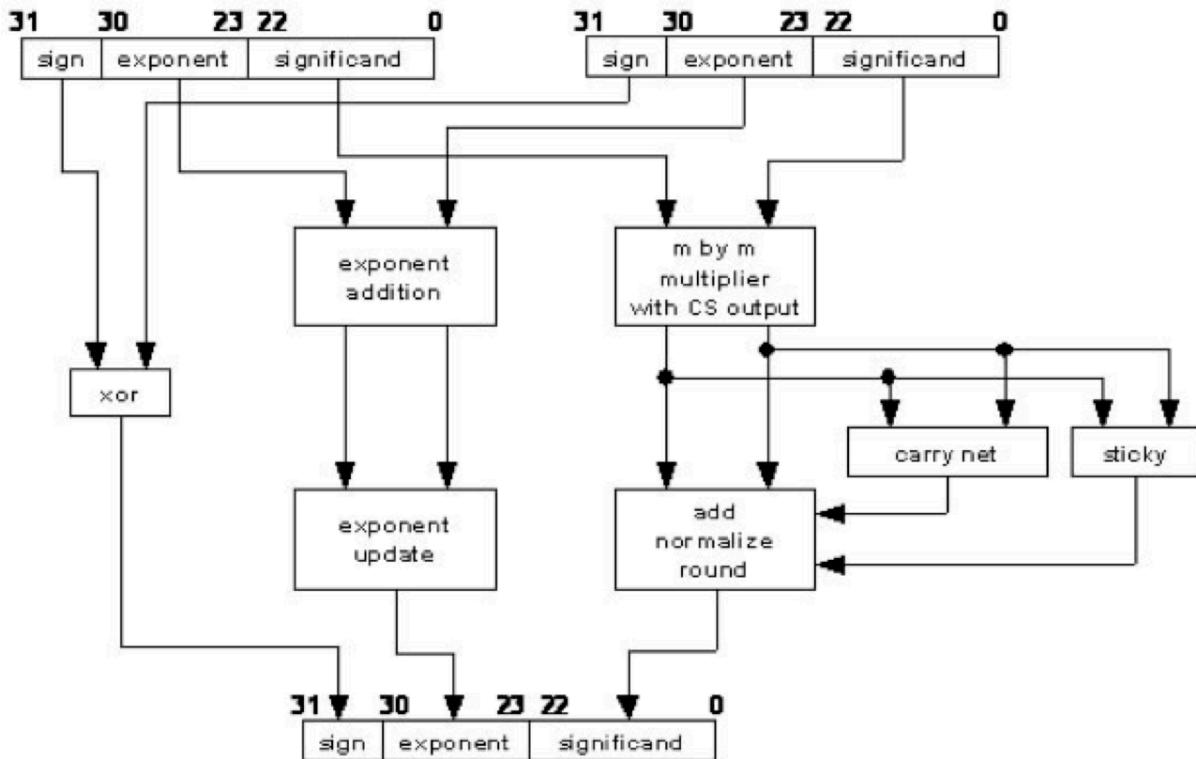


Figure 2.11: วงจรคูณเลข Floating-Point ตามมาตรฐาน IEEE 754 โดยรับค่าอินพุทจำนวน 2 ตัวด้านบน และเอาท์พุทผลลัพธ์ด้านล่าง (ที่มา: Patterson and Hennessy (2000))

วงจรคูณเลขฐานสองแบบ Floating-Point ในรูปที่ 2.11 มีความซับซ้อนมากกว่า วงจรคูณเลขจำนวนเต็มในรูปที่ 2.6 และ 2.7 ดังนี้

- ขั้นตอนที่ (Step) 1 คือ คูณส่วนที่เป็น Fraction ทั้งสองค่า
- ขั้นตอนที่ (Step) 2 คือ บวกค่า Exponent ทั้งสองเข้าด้วยกันแล้วลบค่าไบอัสหนึ่งครั้ง
- ขั้นตอนที่ (Step) 3 คือ น้อมลัลไซซ์ค่าผลลัพธ์ และตรวจเช็คโอล์ฟล์วหรืออันเดอร์ฟล์ว ซึ่งผลลัพธ์ที่ได้จากการคูณอาจมีค่าเพิ่มขึ้น หรือ ลดลงขึ้นอยู่กับขนาด (Magnitude) และ Exponent ของเลขทั้งสองตัว ทำให้ต้องน้อมลัลไซซ์อีกรอบ ระหว่างนั้น เมื่อค่าสัมบูรณ์ของผลลัพธ์เพิ่มมากขึ้นจนเกินค่าสูงสุด เรียกว่าเกิดโอล์ฟล์ว และ หากค่าสัมบูรณ์ของผลลัพธ์มีค่าเข้าใกล้ศูนย์จนน้อยกว่าค่าต่ำสุดในตารางที่ 2.9 เรียกว่าเกิด อันเดอร์ฟล์ว
- ขั้นตอนที่ (Step) 4 คือ ปัดค่า Fraction ให้เหลือ 23 บิตและอาจต้องน้อมลัลไซซ์อีกรอบเมื่อจำเป็น

ตัวอย่างที่ 2.12.1. จงคูณเลขฐานสองแบบ Floating-Point Single Precision ดังต่อไปนี้

$$\text{EX: } -5_{10} \times -0.75_{10} = \text{C0A0 } 0000_{16} \times \text{BF40 } 0000_{16} = ?$$

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

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

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

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

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

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

4. คูณค่านัยสำคัญ (significands) เข้าด้วยกัน

$$\underline{1.010\ 0000\ 0000\ 0000\ 0000}_2 \times$$

$$\underline{1.100\ 0000\ 0000\ 0000\ 0000}_2$$

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

5. น้อมลําเลซ์ค่าผลลัพธ์ และตรวจเช็คการเกิดโอเวอร์ฟอล์ว์หรืออันเดอร์ฟอล์ว์

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

6. ปัดค่าให้เหลือ 23 หลักและอาจต้องน้อมลําเลซ์เมื่อจำเป็น

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

7. ปรับเครื่องหมายของผลลัพธ์ให้ถูกต้องจากตัวตั้งและตัวคูณ

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

$$S = 0$$

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

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

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

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

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

ตัวอย่างที่ [2.12.1](#)

2.13 ตัวอักษร (Alphabet)

รูปที่ 2.12 แสดง การเก็บข้อความในหน่วยความจำในรูปของรหัส ASCII ด้วยตัวแปร str ซึ่งเป็นตัวแปรชนิดตัวเรียกของตัวอักษร (Array of Character)

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

ที่แอดдресเริ่มต้น 0x50 การเรียงไปทีตัวขอเริ่มต้นจากไปทีที่แอดdressต่อไปแอดdressที่สูงขึ้น เริ่มต้นที่แอดdress 0x50 เก็บรหัส ASCII หมายเลข 0x48 ซึ่งตรงกับตัวอักษร 'H' ไปจนถึงแอดdress 0x56 ซึ่งเก็บตัวขอ 0x00 หรืออักขระ NULL ซึ่งเป็นอักขระพิเศษแสดงว่าจบประโยค รูปที่ 2.12 ส่วนไปทีที่ 0x57, 0x58, 0x59 จะมีค่าเป็น unknown เพราะไม่มีผลต่อค่าของประโยคที่เก็บ ตัวแปร str นี้สามารถเก็บตัวอักษรความยาวสูงสุด 10 ตัวและตัวสุดท้ายจะต้องเป็นตัวขอ NULL เท่านั้น ยกตัวอย่างเช่น ประโยค "012345678" ประโยค "abcdefghi" เป็นต้น

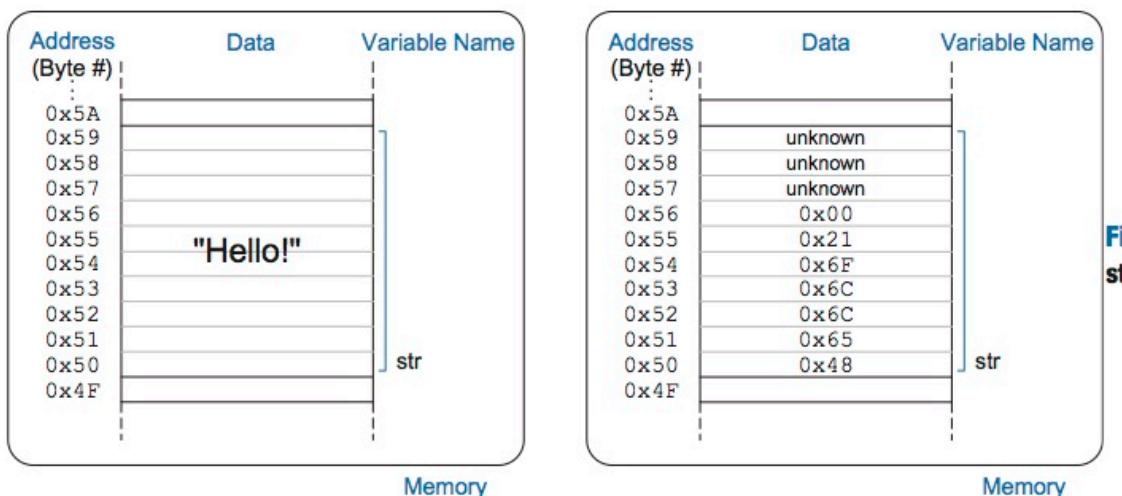


Figure 2.12: การเก็บข้อความในหน่วยความจำในรูปของรหัส ASCII ด้วยตัวแปร str ซึ่งเป็นตัวแปรชนิด char[10] str="Hello!" ที่แอดdressเริ่มต้น 0x50 (ที่มา: [Harris and Harris \(2013\)](#))

รหัส ASCII และรหัส Unicode

ตารางที่ 2.13 คือ ตารางรหัสแอสกี้ (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} แทนอักขระ NULL อ่านว่า นัล

ไมโครซอฟต์พัฒนารหัสภาษาไทยของตนเอง 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	່	ຈ	້	ໜ	ໜ	ໝ	ໝ	ໝ	ໝ
B-	0E10	ໜ	ໝ	໘	ດ	ດ	ດ	ກ	ກ	ນ	ບ	ປ	ຜ	ຜ	ພ	ພ
C-	0E20	ກ	ມ	ຢ	ຮ	ຖ	ຄ	ກ	ວ	ສ	ໜ	ໜ	ໜ	ໜ	ໜ	ໜ
D-	0E30	ິ	ີ	ື	ຸ	ົ	ົ	ົ	ົ	ົ	ົ	ົ	ົ	ົ	ົ	ົ
E-	0E40	ເ	ແ	ໂ	ຈ	ີ	ກ	ງ	ັ	ັ	ັ	ັ	ັ	ັ	ັ	ັ
F-	0E50	ອ	ອ	ໂ	ຕ	ແ	ແ	ແ	ຳ	ຳ	ຳ	ຳ	ຳ	ຳ	ຳ	ຳ

Figure 2.13: ตารางรหัสแอ็สกี้ (ASCII) และ Unicode สำหรับตัวอักษรจำนวน $2^8=256$ ตัว ภาษาอังกฤษและภาษาไทย ในรูปที่ 2.13 ภายในแต่ละตัวอักษร มีค่ารหัส Unicode กำกับอยู่ด้วย ยกตัวอย่างเช่น ตัวอักษรไทย เริ่มต้นที่ รหัส A1 คือ ก เลขฐานสองขนาด 8 บิต เพื่อแทน

ตารางรหัสแอ็สกี้ (ASCII) และ Unicode สำหรับตัวอักษรจำนวน $2^8=256$ ตัว ภาษาอังกฤษและภาษาไทย ในรูปที่ 2.13 ภายในแต่ละตัวอักษร มีค่ารหัส Unicode กำกับอยู่ด้วย ยกตัวอย่างเช่น ตัวอักษรไทย เริ่มต้นที่ รหัส A1 คือ ก เลขฐานสองขนาด 8 บิต เพื่อแทน

รหัส Unicode จะใช้พื้นที่ 16 บิต ต่อ 1 ตัว Chr ซึ่งทำให้สามารถใช้เลขฐานสองจำนวน 216 หรือ 65,536 แบบมาแทนตัว Chr ได้เกือบทุกภาษาในโลกนี้ การนำตารางรหัส ASCII เดิม แล้วเพิ่มอักษรต่างๆ เข้าไปต่อท้ายตาราง โดยตำแหน่งเริ่มต้นของตารางรหัส Unicode จะเหมือนกับตารางรหัส ASCII

การทดลองที่ 1 หัวข้อที่ A.3.1 จะเปิดโอกาสให้ผู้อ่านได้ทดลองแปลงตัวอักษรต่างๆ ให้เป็นรหัส ASCII และรหัส Unicode ด้วยตนเอง เพื่อสร้างความเข้าใจที่ลึกซึ้งมากขึ้น

2.14 สรุป

การคำนวณทางคณิตศาสตร์ของเลขจำนวนเต็มชนิดมีเครื่องหมาย และไม่มีเครื่องหมายจำเป็นต้องตรวจจับการเกิดโอเวอร์ฟอล์ว์ เนื่องจากอาร์ดแวร์หรือโปรเซสเซอร์มีจำนวนบิตในการคำนวณที่จำกัด ปัจจุบันคือ 32 64 และ 128 บิต ในทำนองเดียวกัน การคำนวณโดยใช้เลขทศนิยมชนิดจุดทศนิยมคงที่ หรือ Fixed Point และloyตัว หรือ Floating-point จำเป็นต้องตรวจจับการเกิดโอเวอร์ฟอล์ว์/อันเดอร์ฟอล์ว์ และกรณีอื่นๆ ด้วยเหตุผลที่ซับซ้อนกว่าเลขจำนวนเต็ม ทำให้ใช้ทรัพยากร้านอาร์ดแวร์และเวลาหรือจำนวนคลีอกมากกว่า

ชนิดและความยาวของข้อมูลที่หากหลายตามที่สรุปในตารางที่ 2.10 ของซอฟต์แวร์คอมพิวเตอร์ ต้องการความสามารถของไมโครโปรเซสเซอร์หรืออาร์ดแวร์ให้รองรับตามเช่นกัน เพื่อให้ประสบการณ์การใช้งานของผู้ใช้ (User Experience) ดีขึ้นเรื่อยๆ ยกตัวอย่างเช่น การใช้ GPU มาช่วยคำนวณ การคำนวณข้อมูลที่จัดเรียงตัวกันแบบเวกเตอร์ (Vector) เพื่อเพิ่มประสิทธิภาพ เป็นต้น

Table 2.10: 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 จำนวนดาวเทาฯ
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	± นำ้มือ	นำ้มือกอนุภาคเล็กๆ

Computer Hardware & Software

วัตถุประสงค์

- เพื่อให้เข้าใจโครงสร้างโดยองค์รวมของคอมพิวเตอร์ กรณีศึกษาบอร์ด Raspberry Pi3
- เพื่อให้เข้าใจโครงสร้างและการทำงานของด้านฮาร์ดแวร์ กรณีศึกษา ARM Cortex A53 ภายในชิป BCM2837
- เพื่อให้เข้าใจโครงสร้างและการทำงานของซอฟต์แวร์ระบบปฏิบัติการลีนูกซ์
- เพื่อให้เข้าใจการประสานงานระหว่างฮาร์ดแวร์และซอฟต์แวร์ เข้าใจซอฟต์แวร์ การดำเนินงาน และการสั่งงานให้ฮาร์ดแวร์ทำงาน
- เพื่อให้เข้าใจขบวนการพัฒนาซอฟต์แวร์ด้วยภาษา C และภาษาแอสเซมบลี

3.1 ฮาร์ดแวร์ของเครื่องคอมพิวเตอร์

Raspberry Pi3 https://wikidevi.com/wiki/RPF_Raspberry_Pi_3_Model_B และ https://elinux.org/RPi_Hardware เป็นคอมพิวเตอร์ขนาดเล็กที่ได้รับความนิยม เนื่องจากราคาไม่แพง การประสานงานของฮาร์ดแวร์และซอฟต์แวร์ที่ลงตัว รวมไปถึงมีซอฟต์แวร์ประยุกต์ หรือ แอพพลิเคชัน (Application) ที่หลากหลาย กรณีศึกษา Raspberry Pi3 ในตำราเล่มนี้จะอ้างอิงกับบอร์ด Pi3 โมเดล B ในรูปที่ 3.1 นอกจากโมเดล B บอร์ด Pi3 โมเดล A มีลักษณะต่างกันไม่มากแต่จะเน้นการทำงานแบบระบบฝังตัว (Embedded System) เป็นหลัก

บอร์ด Raspberry Pi3 (Pi3) โมเดล B เป็นคอมพิวเตอร์บอร์ดเดียว (Single Board) ราคาเพื่อการศึกษา เป็นรุ่นถัดจาก Raspberry Pi 2 (RPi 2) โมเดล B ออกแบบและพัฒนาโดยองค์กรที่มีชื่อว่า Raspberry Pi Foundation บอร์ด Pi2 และ Pi3 มีขนาดเท่ากัน ($86\text{mm} \times 56\text{mm} \times 21\text{mm}$) ทำให้ใช้กล่อง (Case) ขนาดเดิมได้ ตำแหน่งของพอร์ตและكونเนกเตอร์คล้ายกัน แต่บอร์ด RPi3 มีความสามารถในการประมวลผลที่สูงขึ้นและมีประสิทธิภาพดีกว่า (ตัวอย่างการเปรียบเทียบ เช่น A Comprehensive Raspberry Pi3 Benchmark)

บอร์ด Pi3 สามารถนำไปประยุกต์ใช้ได้หลากหลาย เช่น Low cost PC/tablet/laptop Media center Industrial/Home automation Print server Web camera Server/cloud server Wireless access

point Security monitoring Environmental sensing/monitoring IoT applications Robotics

การทดลองเพื่อประกอบบอร์ด Pi3 กับอุปกรณ์ต่างๆ เป็นหลัก เริ่มต้นทั้ง การติดตั้งระบบปฏิบัติการ การใช้งานซอฟต์แวร์ที่จำเป็น การพัฒนาโปรแกรมด้วยภาษา C/C++ การพัฒนาโปรแกรมด้วยภาษา Assembly เป็นต้น

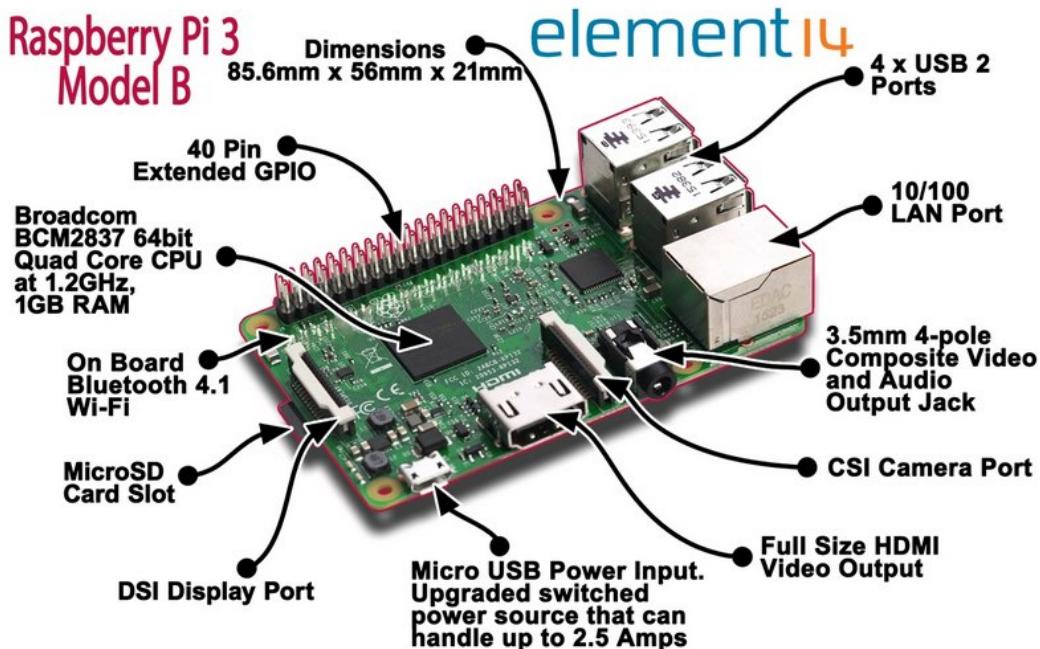


Figure 3.1: ตำแหน่งของอุปกรณ์ต่างๆ บนบอร์ด Pi3 ที่มา: <http://www.raspberryhome.net/product/72>

แผ่นวงจรพิมพ์ของบอร์ด Raspberry Pi3 โมเดล B ในรูปที่ 3.1 ประกอบด้วยชิป BCM2837 ผลิตโดยบริษัท Broadcom Corp. ประเทศสหรัฐอเมริกา หน่วยความจำหลักชนิด DDR2 (Double Data Rate 2) SDRAM (Synchronous Dynamic Random Access Memory) ขนาดความจุ 1 กิกะไบท์ หน่วยสำรองข้อมูลชนิด SD (Secure Digital) Memory card ขนาด 8-16 กิกะไบท์ เชื่อมต่ออินเตอร์เน็ตผ่านระบบสื่อสารไร้สาย WiFi และสาย Ethernet LAN ซ่องเสียบ USB สำหรับเชื่อมต่อคีย์บอร์ดและเมาส์ เป็นต้น

องค์ประกอบสำคัญของบอร์ด คือ ชิป BCM2837 เป็น SoC ย่อมาจาก System on Chip ที่มีชิปยุ่ง ARM Cortex A53 ARMv8 แบบ Quad-core ขนาด 64 บิต ใช้ความถี่สัญญาณคลื่น 1.2GHz ซึ่งแตกต่างจากกรณีของบอร์ด Pi2 ที่ใช้ BCM2836 ซึ่งเป็น SoC ที่มีชิปยุ่ง Cortex-A7 ARMv7 ขนาด 32 บิต ทำงานที่ความถี่ 900MHz แม้ว่าจะเป็นแบบ Quad-core เช่นกัน แต่บอร์ด RPi3 มีหน่วยความจำ 1GB DDR2 RAM ซึ่งมากกว่าขนาดความจำในบอร์ด RPi 2 นอกจากนั้น บอร์ด Pi3 ยังมีชิปสำหรับเชื่อมต่อ Wi-Fi IEEE 802.11 และ Bluetooth 4.1 ได้โดยตรง ในขณะที่บอร์ด Pi2 ต้องซื้ออุปกรณ์มาต่อเพิ่ม

จุดเด่นของบอร์ด Pi3 โมเดล B คือ การเชื่อมตอกับอุปกรณ์อินพุท/เอาท์พุทที่หลากหลาย รูปที่ 3.2 แสดงการเชื่อมโยงอุปกรณ์ต่างๆ บนบอร์ด Pi3 โดยมีชิป BCM2837 SoC (System on Chip) เชื่อมตอกับอุปกรณ์ภายนอกเท่าที่จำเป็น เพื่อให้ระบบสมบูรณ์ ได้แก่ หน่วยความจำหลัก SDRAM ขนาด 1GB ด้วยวงจรภายในชิป หน่วยสำรองข้อมูลชนิด Micro SD ผ่านสาย SDIO (Secure Digital Input/Output) บอร์ด Pi3 รองรับการเชื่อมตอกับจอแสดงผลได้ 3 ชนิด คือ จอโทรทัศน์ด้วยสัญญาณคอมโพสิตวีดีโอ จอด

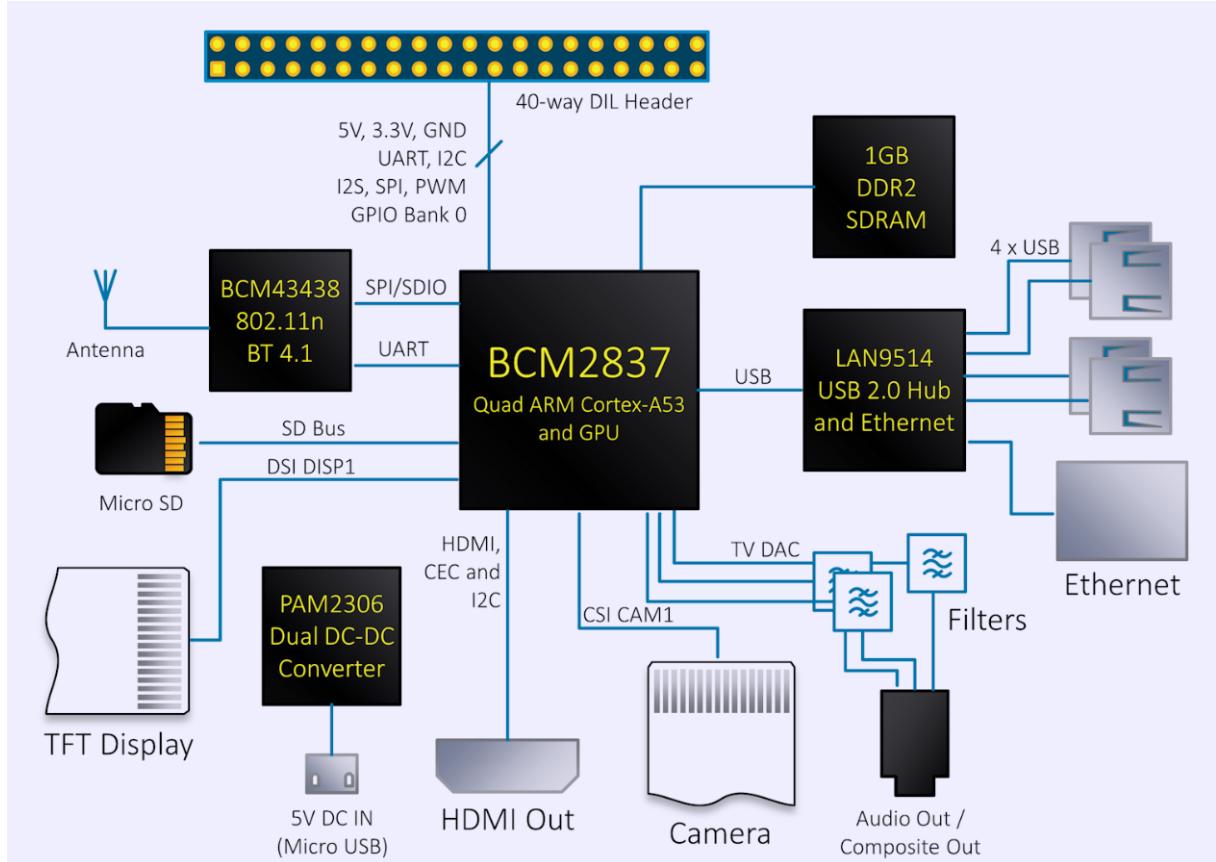


Figure 3.2: การเชื่อมโยงอุปกรณ์ต่างๆ บนบอร์ด Pi3 โดยมีชิป BCM2837 เป็นศูนย์กลาง รายละเอียดเพิ่มเติมในบทที่ 6 ที่มา: https://xdevs.com/article/rpi3_oc/

LCD ขนาดใหญ่ผ่านสาย HDMI และ จอ LCD ขนาดเล็กผ่านสาย SDI บนบอร์ดขยายจำนวนพอร์ตด้วย USB 2.0 ชิ้น (Hub) จำนวน 4 พอร์ต เชื่อมต่ออินเทอร์เน็ตด้วยสาย 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 หน่วยประมวลผลกลาง (CPU: Central Processing Unit)

ชิป BCM2837 เป็นชิปที่ผลิตโดย Broadcom (<https://www.broadcom.com>) ประกอบด้วย

- ชิปปี้ ARM Cortex A53 จำนวน 4 คอร์ (Quad core) ใช้สัญญาณนาฬิกาความถี่ 1.2 GHz รองรับคำสั่งภาษาแอสเซมบลี (Assembly) ของ ARM เวอร์ชัน 8A
- จีปีปี้ (GPU: Graphic Processing Unit) เป็นหน่วยประมวลผลด้านกราฟิก จำนวน 2 คอร์ ใช้สัญญาณนาฬิกาความถี่ 400 MHz และรองรับ OpenGL ES เวอร์ชัน 2.0 และ OpenVG (<https://en.wikipedia.org/wiki/OpenVG>)
- โมดูลจัดการวีดีโอ (Video Engine) สามารถดรอหัสภาพวีดีโอที่ความละเอียด 1080p30 ตามมาตรฐาน H.264 ด้วยอัตรา 1 พันล้านพิกเซลต่อวินาที และโมดูล DMA

Table 3.1: ตารางสรุปข้อมูลด้านฮาร์ดแวร์และซอฟต์แวร์ของบอร์ด Pi3 โมเดล B และลิงค์เชื่อมไปยังหัวข้อที่แสดงรายละเอียด

องค์ประกอบ	รายละเอียด	ในหัวข้อที่
BCM2837	SoC ผลิตโดยบริษัท Broadcom ประกอบด้วย	3.1.1
CPU	Quad (4)-Core ARM Cortex-A53 ความถี่ 1.2 GHz	
GPU	Dual (2) VideoCoreIV ความถี่ 400 MHz Multimedia Co-Processor, OpenGL ES 2.0 1080p 30fps H.264 High-Profile Decoder	
Input/Output	ไม่ดูแลภายในชิป BCM2837	
จอขนาดใหญ่	สาย HDMI เวอร์ชัน 1.3 & 1.4 (ภาพและเสียง)	6.1
จอ LCD ขนาดเล็ก	สาย Display Serial Interface (DSI) 15 ขา ประกอบด้วยสัญญาณข้อมูล 2 คู่ สัญญาณคลื่น 1 คู่	6.2
กล้องขนาดเล็ก	สาย MIPI Camera Serial Interface (CSI-2) 15-ขา	6.3
สัญญาณเสียง	แจ็ค 3.5 มม ชนิด 4 ขั้ว	6.4
จอทีวี	สัญญาณคอมโพสิตวิดีโอ PAL/NTSC	6.5
GPIO	ขั้วต่อชนิด 2.54 มม 40 ขา ประกอบด้วย GPIO 27 ขา +3.3 และ +5V โวลท์	6.11
Input/Output	ไม่ดูแลภายนอกชิป BCM2837	
USB	ชิป USB 2.0 จำนวน 4 พอร์ต	6.6
อินเตอร์เน็ต	ชิป Ethernet 10/100 Mbps	6.7
อินเตอร์เน็ตไร้สาย	ชิป WiFi 802.11 b/g/n สูงสุด 150Mbps	6.8
เชื่อมต่อไร้สาย	ชิป Bluetooth 4.1 (ปกติและประหยัดพลังงาน)	6.8
หน่วยความจำหลัก	ชิป DDR2 SDRAM ความจุ 1 GB ชนิดประยุกต์พลังงาน	3.1.2, 5.3
หน่วยสำรองข้อมูล	สล็อตใส่การ์ด MicroSD upto x GB	3.1.4, 7.2
ระบบปฏิบัติการ	บูทจากการ์ด MicroSD รองรับ Raspbian Linux และ Windows 10 IoT	3.2.1
แหล่งจ่ายไฟ	ซีอกเก็ตชนิด ไมโคร USB 5 โวลท์ 2.5 แอมเปอร์	6.18

- วงจรเชื่อมต่อหน่วยความจำภายนอก ชนิด DDR2 SDRAM และ SD เวอร์ชัน 2.0

- ไม่ดูแลเชื่อมต่ออินพุตและเอาท์พุต ซึ่งรายละเอียดในตารางที่ 3.1

เนื่องจากรายละเอียดโครงสร้างภายในของ BCM2837 มีไม่มากนัก ผู้เขียนจึงขอใช้ข้อมูลจากชิปที่มีคุณสมบัติใกล้เคียงกันแทน

รูปที่ 3.3 บล็อกไดอะแกรมของชิป A64 จากบริษัท AllWinner (Allwinner Technology (2015a) และ Allwinner Technology (2015b)) ที่มีโครงสร้างคล้ายกับ BCM2837 บนบอร์ด Raspberry Pi3 แต่ต่างกันในโมดูลสำคัญๆ เช่น GPU, Ethernet เป็นต้น

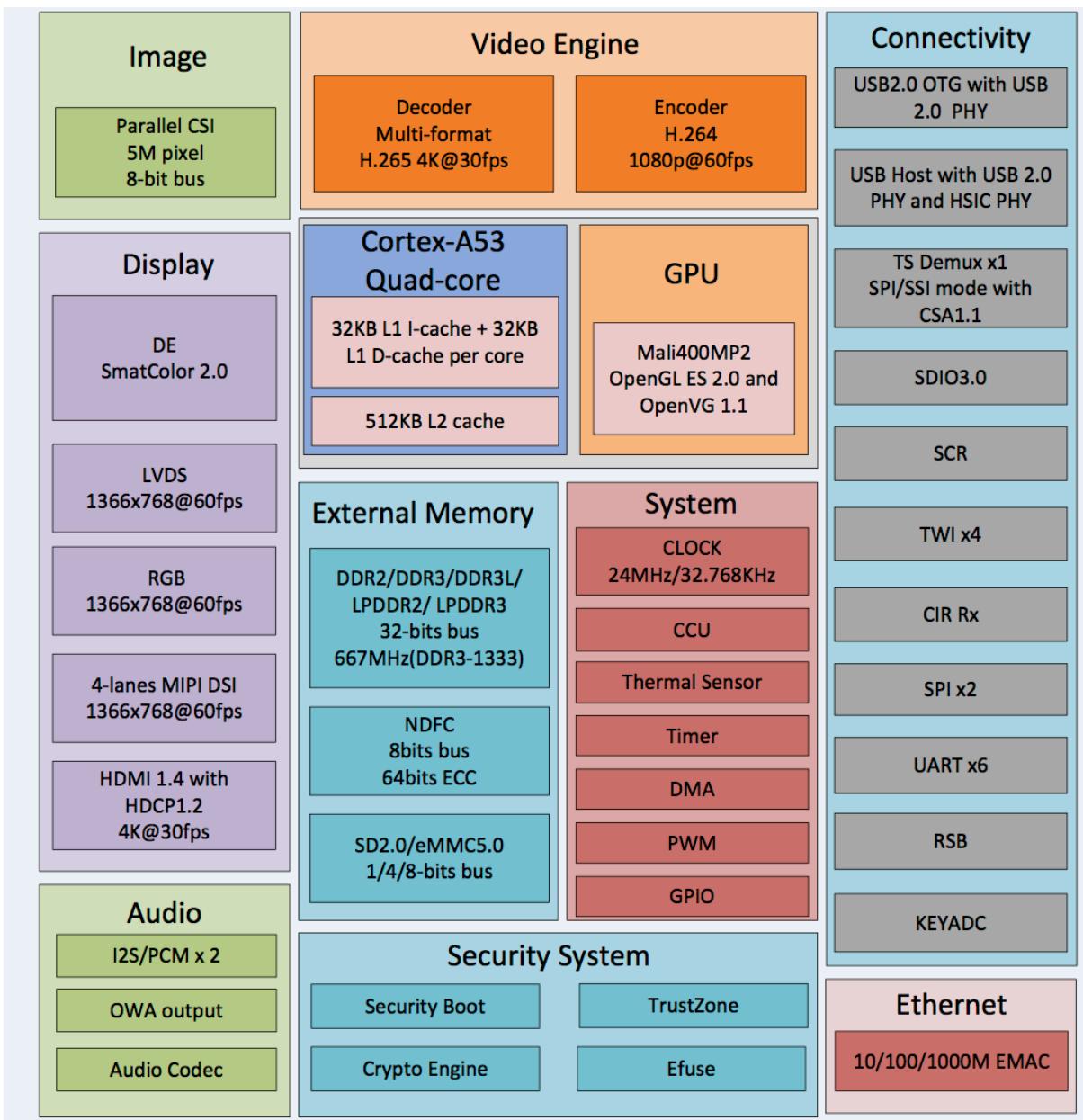


Figure 3.3: บล็อกໄດ້ອະແກນຂອງຊີບ AllWinner A64 (ທີ່ມາ: [Allwinner Technology \(2015a\)](#) ແລະ [Allwinner Technology \(2015b\)](#)) ທີ່ມີໂຄຮສ້າງຄລ້າຍກັບ BCM2837 ບນບອົດ Raspberry Pi3

3.1.2 ໜ່ວຍຄວາມຈໍາຫລັກ (Main Memory)

ເປັນຊີບທີ່ນ່ວຍຄວາມຈໍາໄດ້ນາມືກແຣມແບບສິງໂຄຣນັສ ຮູ່ອ SDRAM (Synchronous Dynamic RAM) ຊົນດ DDR2 (Double Data Rate 2) ເພີ້ມສໍາຫຼັບໃຈ່ງນັນອຸປະນົມເຄລື່ອນທີ່ (Mobile) ເພີ້ມເປັນຮຸ່ນປະຫຍັດ ພັງງານ ຊົນດ LP (Low Power) ແລະ ຈຳນວນຂານ້ອຍເພື່ອປະຫຍັດພື້ນທີ່ບັນບອົດ ໂດຍໃໝ່ຂາຈຳນວນ 10 ຂາ ຮ່ວມກັນເພື່ອຮັບຄໍາສັ່ງ (Command) ແລະ ສັງຄູານແອດເດຣສ (Address) ຮ່ວມກັນ ໂດຍຮັບຄໍາສັ່ງທີ່ຂອບຂາໜຶນ ແລະ ຂອບຂາລັງຂອງແຕ່ລະ ສັງຄູານຄລື້ອກ ຮອງຮັບສັງຄູານຄວາມຄືສູງສຸດ 400 MHz ທຳໄໝເກີດຄວາມເວົ້ວສູງສຸດ 800 ເມກະບີຕ/ວິນາທີ ເນື່ອງດ້ວຍຄວາມຄືສັງຄູານຄລື້ອກທີ່ສູງ ຕ້ວັນມີໜາດເລີກ ຄວາມຮ້ອນຈຶງເປັນອຸປະກອດຕ່ອງ ການທຳງານ ດັ່ງນັ້ນ ຜູ້ອ່ານຄວາມຕິດຕັ້ງອີ່ຫົງຫົກ (Heat Sink) ເພື່ອຮະບາຍຄວາມຮ້ອນຂອງຊີບໄດ້ນາມືກແຣມນີ້ ຕາມ ການທົດລອງທີ່ 2 ການພັນກວກ B ການຕິດຕັ້ງແລະ ໃຊ້ງານຢາർດແວງ



Figure 3.4: หน่วยความจำไดนามิกแรมด้านล่างของบอร์ด Pi3 โมเดล B

ชิป SDRAM ผลิตโดย Elpida รุ่น B8132B4PB-8D-F วางอยู่ด้านล่างของบอร์ด Pi3 โมเดล B ดังรูปที่ 3.4 เพื่อลดขนาดฟุตพري้ท (Foot Print) บนแผ่นวงจรพิมพ์ ขาดภัยภูมิต่างๆ จึงช่วยให้ชิป โดยภายในชิปประกอบด้วย แผงอะเรย์ DRAM จำนวน 2 ชุดๆ ละ 8 ชิ้นๆ ละ 16×32 เมกะเซลล์ คิดเป็น 32 เมกะเซลล์ $\times 16$ บิต $\times 8$ ชิ้น $\times 2$ ดาย (die) ต่อชิป ตามวิธีการคำนวณต่อไปนี้

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

ปัจจุบันนี้ บริษัท Elpida ได้เปลี่ยนผู้ถือหุ้นหลักเป็น บริษัท Micron Technology และ ดังนั้น ได้ นำมิคแรมหมายเลขรุ่น EDB8132B4PB จึงสามารถค้นคว้าเพิ่มเติมได้ตามลิงค์ต่อไปนี้ <https://www.micron.com/parts/dram/mobile-ddr2-sdram/edb8132b4pb-8d-f>

หนังสือเล่มนี้จะอธิบาย การทำงานและรายละเอียดเพิ่มเติมของหน่วยความจำ SDRAM นี้และชนิดต่างๆ ในบทที่ 5 ในหัวข้อที่ 5.3.1

3.1.3 อินพุท/เอาท์พุท (Input/Output)

หัวข้อนี้จะกล่าวเฉพาะอินพุทและเอาท์พุทที่จำเป็นก่อน ได้แก่ คีย์บอร์ด เม้าส์ จอモニเตอร์ ส่วนอินพุท และเอาท์พุทนั่นๆ ผู้อ่านสามารถอ่านเพิ่มเติมในบทที่ 6 ได้อย่างละเอียด

คีย์บอร์ด (Keyboard)

คีย์บอร์ดชนิด 106 ปุ่มนี้ เป็นคีย์บอร์ดที่ออกแบบโดยบริษัท IBM และพัฒนาต่อเนื่องมา มีลักษณะคล้ายกับปุ่มพิมพ์บนเครื่องพิมพ์เดิม โดยปกติจะประกอบด้วย

- ปุ่มตัวอักษร ประกอบด้วยอักษรสำหรับการป้อนข้อมูลที่มีทั้งตัวอักษร A-Z และ a-z ตัวเลข 0-9 และยังรวมพิเศษ เช่น @, #, \$, % เป็นต้น
- ปุ่มป้อนข้อมูลตัวเลข โดยจะมีสแกนโค้ดเท่ากับ 90-109 สำหรับการป้อนข้อมูลที่เป็นตัวเลขเรียงตัวกันทางขวาสุดของคีย์บอร์ด เพื่อความสะดวกต่อการใช้งานสำหรับนักบัญชี หรือ ผู้ที่กรอกข้อมูลบ่อยๆ

106-Key Keyboard Position Codes

Figure 3.5: ตำแหน่ง และ สแกน โค้ด (Scan Code) ของ ปุ่ม บน คีย์บอร์ด ชนิด 106 ปุ่ม
 ที่มา: <http://ps-2.kev009.com/tl/techlib/manuals/adoclib/aixkybd/kybdtech/figures/kybdt3.jpg>

- ปุ่มฟังก์ชัน 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) นิยมใช้เคลื่อนหรือเปลี่ยนตำแหน่งของเมาเซอร์ (Cursor) บนหน้าจอแสดงผลทั้งในโหมดกราฟิกและตัวคัลชัน โดยจะมีสแกนโค้ดเท่ากับ

รูปที่ 3.5 ตำแหน่งและสแกนโค้ด (Scan Code) ของปุ่มบันคីយបอร์ดชนิด 106 ปุ่ม ผ่านสายไปยังพอร์ต USB เป็นหลัก เมื่อผู้ใช้กดปุ่มตั้งแต่ 1 ปุ่มขึ้นไป คីយបอร์ดจะส่งรหัสของปุ่มที่ถูกกดนั้น วงจรติดจิหัลภายในตัวคីយបอร์ดจะส่งสแกนโค้ดของปุ่มที่ได้นกัดตามที่ได้ออกแบบไว้ หรืออาจจะเป็นคីយបอร์ดไร้สาย เช่น ชนิดคลื่นวิทยุความถี่ต่ำ ชนิดคลื่นบลูทูธ เป็นต้น ไปยังระบบปฏิบัติ เพื่อให้โปรแกรมตอบสนองต่อรหัสนั้นๆ เมื่อผู้ใช้กดปุ่มใดๆ ก็ครั้งกีต้าม ระบบปฏิบัติการจะรับรู้ได้ผ่านกลไกการเกิดอินเตอร์รัพท์ (Interrupt) ซึ่งมีรายละเอียดเพิ่มเติมในการทดลองที่ 9 ภาคผนวก |

มาส์ (Mouse)

มาส์เปลี่ยนการเคลื่อนที่ของแขนงผู้ใช้ในแกน 2 มิติเป็นการเคลื่อนที่ของพอยท์เตอร์ (POINTR) การเคลื่อนที่ของมาส์จะสัมพัทธ์กับตำแหน่งของพอยเตอร์บนจอแสดงผล ทั้งนี้ขึ้นอยู่กับความละเอียดของมาส์เอง ความละเอียดของหน้าจอแสดงผล และแอปพลิเคชันนั้นๆ เช่น เกมส์ โปรแกรมวัดและตัดต่อภาพ จะต้องการความละเอียดของมาส์สูงกว่ามาส์ปกติ เป็นต้น

มาส์ส่วนใหญ่ไม่จำนวนปุ่มคลิกทางซ้ายและขวา ขึ้นอยู่กับการออกแบบของระบบโอลีอิส มี 1-3 ปุ่มสำหรับการใช้งานปัจจุบัน มีการเพิ่มลูกกลิ้งสำหรับการเคลื่อนแบบพิเศษ เป็นมิติต่างๆ การคลิกปุ่มซ้าย ปุ่มขวา การคลิกปุ่มซ้ายซ้ำ และ การสัมผัสที่หน้าจอลงมาส์บนปุ่ม จะทำให้เกิดแอคชันต่างๆ ตามระบบปฏิบัติ

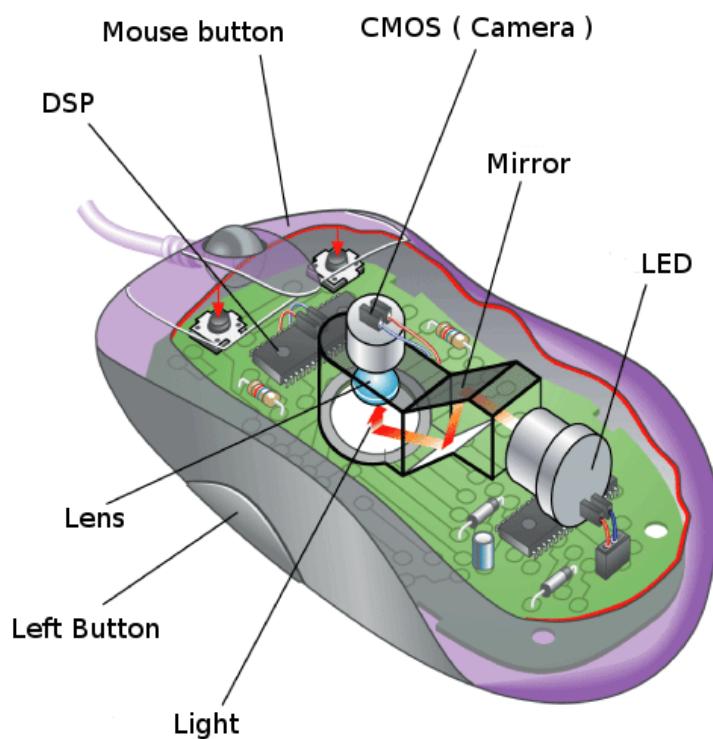


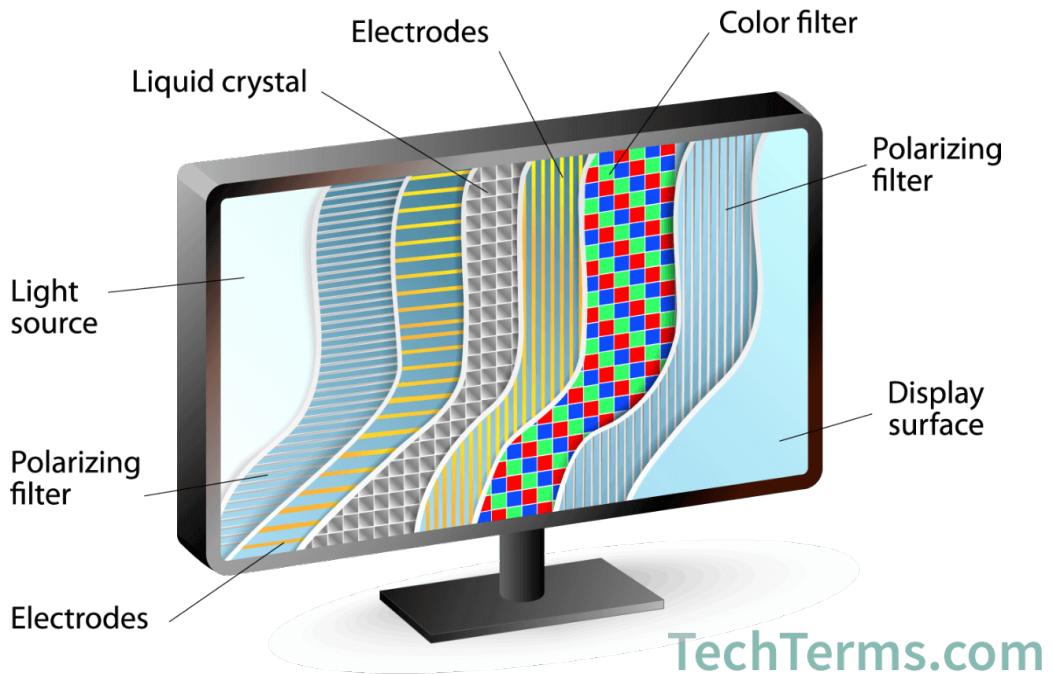
Figure 3.6: โครงสร้างภายในเมาส์ชนิด Optical ประกอบด้วยหลอด LED ส่องแสงและทบกับพื้นผิวด้านล่างแล้วสะท้อนกลับมายังตัวรับแสงชนิด CMOS <https://www.pinterest.com/pin/693835886317197305/>

การและโปรแกรมที่ใช้งาน ดังนั้น ผู้ใช้จะต้องใช้ความรู้ ความเข้าใจ ตลอดจนถึงความจำ เพื่อให้สามารถใช้งานโปรแกรมเดียวกันแต่ทำงานบนระบบปฏิบัติการที่ต่างกันได้

รูปที่ 3.6 แสดง โครงสร้างภายในเมาส์ชนิดออพติคอล (Optical) ประกอบด้วยหลอดแอลอีดี (LED) ส่องแสงและทบกับพื้นผิวด้านล่างก่อน แล้วจึงสะท้อนกลับมายังตัวรับแสงชนิด CMOS แสงจากหลอด LED มีลักษณะที่แตกต่างกันไปตามชนิดและราคาของเมาส์ เช่น หลอด LED เป็นสีแดงหรือขาวซึ่งผู้ใช้มองเห็น และหลอด LED และเลเซอร์ (Laser) ซึ่งมองไม่เห็นด้วยตาเปล่า เพื่อประหยัดการใช้แรงขับของผู้ใช้ และเพิ่มความแม่นยำในการเคลื่อนที่ของเมาส์ ซึ่งแสงเลเซอร์จะให้ความละเอียดในการใช้งานสูงกว่า เหมาะกับซอฟต์แวร์ด้านกราฟิก และเกมส์ ด้วยความละเอียด 800-6000 DPI (Dot Per Inch)

นอกจากเมาส์ อุปกรณ์อื่นๆ ที่ใกล้เคียงสามารถแทนการใช้เมาส์ได้ เช่น ทัชแพด (Touch Pad) จะแสดงผลระบบสัมผัส เสียงพูด ท่าทางของมือและนิ้วที่ตรวจจับโดยกล้องวิดีโอ เป็นต้น ทำให้การใช้งานคอมพิวเตอร์ง่ายลง และได้ประสบการณ์เป็นธรรมชาติมากขึ้น สาขาด้าน Human Computer Interface จึงมีความสำคัญเพิ่มมากขึ้นเรื่อยๆ

เมื่อผู้ใช้กดปุ่มใดๆ หรือขับเมาส์ ระบบปฏิบัติการจะรับรู้ได้ผ่านกลไกการเกิดอินเทอร์รัพท์ (Interrupt) เช่นเดียวกับคีย์บอร์ด ซึ่งมีรายละเอียดเพิ่มเติมในการทดลองที่ 9 ภาคผนวก |



TechTerms.com

Figure 3.7: โครงสร้างจอแสดงผลชนิด Color LCD ประกอบด้วยแหล่งกำเนิดแสง (Light Source) ปล่อยแสงที่ลุ้นต่างๆ มาแสดงผล ที่มา: <https://techterms.com/definition/lcd>

จอแสดงภาพ (LCD Display)

จอแสดงภาพทำหน้าที่แสดงผลตัวอักษรและการพิมพ์ทั้งในรูปของ ภาพนิ่ง ภาพเคลื่อนไหว และเกมส์ เพื่อโต้ตอบและสร้างปฏิสัมพันธ์กับผู้ใช้งาน ซึ่งในอดีตจอแสดงภาพของคอมพิวเตอร์สามารถแสดงผลได้เฉพาะตัวอักษรเท่านั้น

โครงสร้างจอแสดงผลชนิด Color LCD ประกอบด้วยแหล่งกำเนิดแสง (Light Source) ด้านหลังของจอ ปล่อยแสงที่ลุ้นต่างๆ มาแสดงผลทางด้านหน้า ตามรูปที่ 3.7 ความละเอียดในการแสดงผล นับตามจำนวนจุดหรือพิกเซล (Pixel) แหล่งกำเนิดแสงชนิด LED ความสว่าง และการจัดเรียงตัวของหลอด LED การเรียงตัวของแม่สี (RGB) ในแต่ละพิกเซล ขนาดของพื้นผิวน้ำจอ และองศาของมุมมองภาพ (Angle) การเชื่อมต่อ กับบอร์ดชนิด HDMI

จุดภาพ 1 จุดบนจอ LCD ประกอบด้วย ข้าไฟฟ้าชนิดโปร่งแสง (Transparent Electrode) ส่องข้าว และการเรียงตัวของ ผลึกคริสตัล (Crystal) เมื่อมีประจุไฟฟ้าที่ข้าวจะทำให้ผลึกเกิดการบิดตัวคลื่นแสงจากเลนส์โพลาไรซ์ด้านหลัง ที่ลุ่นเลนส์โพลาไรซ์ด้านหน้า ทำให้ผู้ใช้มองเห็นความสว่างของจุดนั้น เมื่อไม่มีประจุไฟฟ้า ผลึกจะไม่บิดเกลี้ยง ทำให้แสงหลักไปไม่ได้ ผู้ใช้จะเห็นจุดนั้นเป็นสีดำ สีที่เกิดขึ้นบนจอเกิดจากการผสมกันของจุดภาพ 3 จุดๆ ละสี คือ แดง เอียว น้ำเงิน ด้วยระดับความสว่างที่ไม่เท่ากัน ทำให้เกิดการผสมของสีที่หลากหลายได้ตามจำนวนบิตข้อมูลสี ซึ่งปัจจุบัน ใช้ข้อมูลสีอย่างน้อย 8 บิต ทำให้เกิดการผสมของสีทั้งหมดคิดเป็น $2^{3 \times 8} = 16.7$ ล้านสีโดยประมาณ

การทดลองที่ 2 ภาคผนวก B การติดตั้งและใช้งานฮาร์ดแวร์



Figure 3.8: โครงสร้างของหน่วยสำรองข้อมูล SD ชนิด SDHC ความจุ 16 GB ประกอบด้วย Flash Memory วงจรควบคุม และวงจรเชื่อมต่อ ที่มา: https://en.wikipedia.org/wiki/Secure_Digital

3.1.4 หน่วยสำรองข้อมูล (Data Storage)

หน่วยความจำ SD มี 3 ขนาด ขนาดปกติ มินิ และ ไมโคร โดยมีตัวแปลงให้เป็นขนาดปกติได้ แบ่งเป็นสีรุ่น ได้แก่ Standard-Capacity (SDSC), High-Capacity (SDHC), eXtended-Capacity (SDXC), และ SDIO ซึ่งรวมฟังก์ชัน input/output กับการบันทึกข้อมูลเข้าด้วยกัน

โครงสร้างทางกายภาพของหน่วยสำรองข้อมูลชนิด SD Card นี้ จะมีกระเตื้องเพื่อช่วยล็อกกับสล็อตที่เสียบ โดยราคาอยู่ตามความจุ ความจุของ SD มีแนวโน้มเพิ่มขึ้น เป็นที่นิยม อายุการใช้งานยาวนาน ความเร็วในการอ่านหรือเขียนสูง ขนาดเล็ก น้ำหนักเบา

บอร์ด Pi3 โมเดล B เพื่อใช้บรรจุระบบปฏิบัติการ บันทึกและสำรองข้อมูลต่างๆ กรณีที่ต้องปิดเครื่อง หรือรีสตาร์ทเครื่อง บอร์ด Pi3 รองรับระบบปฏิบัติการหลายระบบ เช่น Raspbian (Linux), Ubuntu Linux, Microsoft Windows 10 IoT เป็นต้น

SD (Secure Digital) Card พัฒนาให้เป็นมาตรฐานโดยองค์กร ชื่อ SD Card Association (SDA) สำหรับใช้งานกับอุปกรณ์เคลื่อนที่ต่างๆ เช่น กล้องถ่ายรูป โทรศัพท์ เครื่องเล่นเพลง เป็นต้น ในเดือน สิงหาคม 1999 โดยความร่วมมือกันของ บริษัท SanDisk Panasonic และ Toshiba

โครงสร้างพื้นฐานของหน่วยความจำ SD แสดงในรูปที่ 3.8 ชิพภายใน คือ หน่วยความจำ Flash ผลิตโดยบริษัท Sasung หนังสือเล่มนี้จะกล่าวอธิบาย รายละเอียดของ Flash Memroy ในบทที่ 7 หัวข้อที่ 7.1 หน่วยความจำ SD หัวข้อที่ 7.2 และ SSD หัวข้อที่ 7.3 การเชื่อมต่อ CPU กับ SD card นี้สามารถเลือกได้โดยใช้การเชื่อมต่อแบบ SPI และแบบ SDIO การติดตั้ง (Install) ระบบปฏิบัติการ Raspbian ในการทดลองที่ 3 ภาคผนวก C

3.2 ซอฟต์แวร์หรือโปรแกรมคอมพิวเตอร์

โครงสร้างทางซอฟต์แวร์ของคอมพิวเตอร์ แบ่งเป็น

- ซอฟต์แวร์ระบบ (System software) เป็นซอฟต์แวร์พื้นฐานที่เครื่องคอมพิวเตอร์ต้องมี ประกอบด้วย ระบบปฏิบัติการ (Operating System) ทำหน้าที่ดำเนินการด้านอินพุตและเอาท์พุต บริหารจัดการหน่วยความจำ และหน่วยสำรองข้อมูล จัดลำดับการทำงานและการใช้งานทรัพยากรของโปรแกรม เพื่อให้ซอฟต์แวร์ประยุกต์ทำงานได้อย่างถูกต้อง ปลอดภัย และมีประสิทธิภาพ
- ซอฟต์แวร์ประยุกต์ (Application software) เป็นซอฟต์แวร์ที่ผู้ใช้นำมาประยุกต์ในการทำงาน โดยผู้พัฒนาซอฟต์แวร์ (Software Developer) พัฒนาหรือเขียนขึ้นด้วยภาษาระดับสูง (High-level language)

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

เมื่อผู้ใช้กดปุ่มเปิดเครื่องเพื่อจ่ายไฟเลี้ยงให้กับคอมพิวเตอร์ ระบบฮาร์ดแวร์จะเริ่มทำงาน เพื่อตรวจสอบและเตรียมความพร้อม หลังจากนั้น ขั้นตอนการทำงานของซอฟต์แวร์จะแบ่งเป็น

- การบูรณาการปฏิบัติการจากหน่วยสำรองข้อมูลเข้าสู่หน่วยความจำหลัก ในหัวข้อที่ 3.2.1
- การโหลดไฟล์ซอฟต์แวร์แอพพลิเคชันจากหน่วยสำรองข้อมูลเข้าสู่หน่วยความจำหลัก 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/เขียนข้อมูลระหว่างหน่วยความจำหลักและหน่วยสำรองข้อมูล ในหัวข้อที่ 3.2.6
- การซัพเดต (Shut Down) ระบบปฏิบัติการก่อนปิดเครื่อง ในหัวข้อที่ 3.2.7

ขั้นตอนดังกล่าวจะแตกต่างกันออกไปตามรายละเอียดของฮาร์ดแวร์และระบบโอเอสที่ใช้ แต่หลักการดังกล่าวข้างต้นจะมีความคล้ายคลึงกัน

ตัวอย่างเช่นในระบบปฏิบัติการ Linux ซึ่งมีโครงสร้างคร่าวๆ ตามรูปที่ 3.10 ประกอบด้วย Kernel Space และ User Space Kernel Space ประกอบด้วย

- Kernel ตัวโปรแกรมหลัก
- Architecture Dependent Kernel Code สำหรับ Kernel เข้ามายัง Kernel ผ่านช่องทาง Kernel Call Interface และ User Space Kernel Space
- System Call Interface สำหรับ Kernel เข้ามายัง Kernel ผ่านช่องทาง Kernel Call Interface และ User Space Kernel Space

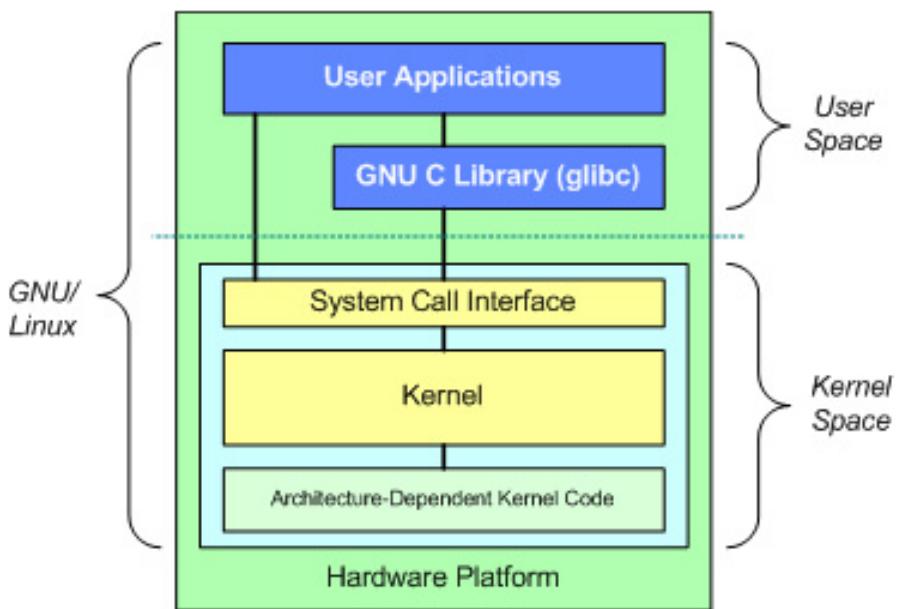


Figure 3.9: โครงสร้างของระบบปฏิบัติการ Linux ซึ่งแบ่งเป็น Kernel Space และ User Space ที่มา: <http://www.linux-india.org/components-of-linux-system/>

User Space ประกอบด้วย User Application หรือซอฟต์แวร์ประยุกต์ต่างๆ ได้แก่ โปรแกรมที่ใช้งานประเภทต่างๆ เช่น เว็บเบราว์เซอร์ (Web Browser) เช่น โมซิลาไฟร์ฟ็อกซ์ (Mozilla Firefox) เกมประเภทต่างๆ เป็นต้น ซึ่งโปรแกรมเมอร์พัฒนาซอฟต์แวร์ประยุกต์เหล่านี้แจกในรูปของฟรีแวร์ (Freeware) และฟรีแวร์บางตัวมีการเปิดเผยแพร่โค้ด เรียกว่า Open Source เพื่อให้โปรแกรมเมอร์ต่างๆ ทั่วโลก สามารถพัฒนาเสริมเพิ่มเติมได้ ตามเงื่อนไข เช่น GPL (GNU General Profile License), LGPL (Lesser GPL) เป็นต้น

ซอฟต์แวร์ประยุกต์เกือบทุกตัวจะทำงานใน User Space ที่กำลังล็อกอินโดย ซอฟต์แวร์ประยุกต์ จะทำงานได้ต้องอาศัยพื้นที่ในหน่วยความจำหลักและความยินยอม (Permission) จากระบบปฏิบัติการ เว็บเบราว์เซอร์สามารถเชื่อมต่อกับเครือข่ายและทรัพยากรอื่นๆ ผ่านทาง System Call Interface เพื่อให้ Kernel ติดต่อกับทรัพยากรเหล่านั้นแทน การทำงานนี้เรียกว่า Kernel Space

การทดลองที่ 3 ภาคผนวก C จะแนะนำการติดตั้งโอเอสชื่อว่า Raspbian ซึ่งเป็น ลีนุกซ์ที่พัฒนาสำหรับใช้งานบนบอร์ด Pi3 โมเดล B นี้ การทดลองที่ 4 ภาคผนวก D จะแนะนำการใช้งานยูนิกซ์เบื้องต้นและคำสั่งพื้นฐาน ซึ่งเป็นโอเอสที่มีประวัติศาสตร์ยาวนานและเป็นต้นแบบของโอเอสต่างๆ ในโลกคอมพิวเตอร์ การทดลองที่ 5 จะแนะนำการพัฒนาโปรแกรมด้วยภาษา C สำหรับยูนิกซ์

3.2.1 การบูทรับบปฏิบัติการจากหน่วยสำรองข้อมูลเข้าสู่หน่วยความจำหลัก

การบูทรับบโอเอส คือ การเริ่มต้นใช้งานเครื่องคอมพิวเตอร์ โดยหลักการคร่าวๆ ระบบจะอ่านไฟล์ kernel จากหน่วยสำรองข้อมูลเข้าสู่หน่วยความจำหลัก ขั้นตอนและรายละเอียดการบูทจะแตกต่างกันไปตามเงื่อนไขของ硬件แวร์และระบบปฏิบัติการ สำหรับระบบปฏิบัติการ Raspbian บนบอร์ด Pi3 มีขั้นตอนโดยละเอียด ดังนี้

- เมื่อเปิดเครื่อง หรือ จ่ายไฟให้บอร์ด Pi3 ซีพียู ARM Cortex A53 จะยังไม่ทำงาน แต่ GPU จะใช้

ซีพียูคอร์ขนาดเล็กภายในอ่านคำสั่งที่บรรจุใน ROM เรียกว่า บูทโลดเดอร์ (Boot Loader) ภายในชิป BCM2837

2. GPU สั่งงานอาร์ดแวร์ที่ควบคุมหน่วยความจำ SD Card เพื่อมองหาเฟิร์ติชันซึ่งจะต้องฟอร์แมทด้วยรูปแบบ FAT32 ภายในการ์ดหน่วยความจำ SD
3. GPU อ่านชุดคำสั่งจากไฟล์ชื่อ bootcode.bin ซึ่งทำหน้าที่เป็น Boot Loader จากไฟล์เดอร์ /bin ในหน่วยความจำ SD โดยในระหว่างนี้จะยังไม่มีการใช้งานหน่วยความจำหลัก SDRAM บนบอร์ด Pi3
4. GPU เริ่มต้นอ่านไฟล์ start.elf ในหน่วยความจำ SD ไปเก็บใน RAM เพื่อให้ซีพียูเริ่มทำงาน โดยอาศัยไฟล์ fixup.dat ซึ่งมีข้อมูลการจัดแบ่งพาร์ติชันภายในระหว่าง GPU และซีพียู
5. ซีพียูเริ่มต้นทำงานตามคำสั่งภายใน start.elf แล้วจึงอ่าน kernel.img ไปบรรจุใน RAM ตามรายละเอียดในไฟล์ config.txt เพื่อติดตั้งค่าของระบบ
6. ซีพียูเริ่มต้นรัน 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 และชื่อไฟล์เดอร์ที่บรรจุไฟล์โปรแกรมและไฟล์ข้อมูลสำหรับการบูท

Table 3.2: ระดับการรัน (Run Level), โหมดการทำงานของระบบ และชื่อไฟล์เดอร์ที่บรรจุไฟล์โปรแกรมและไฟล์ข้อมูลสำหรับการบูท

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/

เมื่อบูทระบบสำเร็จ kernel จะครอบครองอาร์ดแวร์ทั้งหมด และทำงานร่วมกับไฟล์ในไฟล์เดอร์ต่างๆ ของหน่วยสำรองข้อมูล SD ไฟล์เดอร์หรือไดเรกทอรีในระบบปฏิบัติการลีนุกซ์ หรือ ยูนิกซ์ ดังรูปที่ [3.10](#) เรียงตามลำดับความสำคัญดังนี้

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

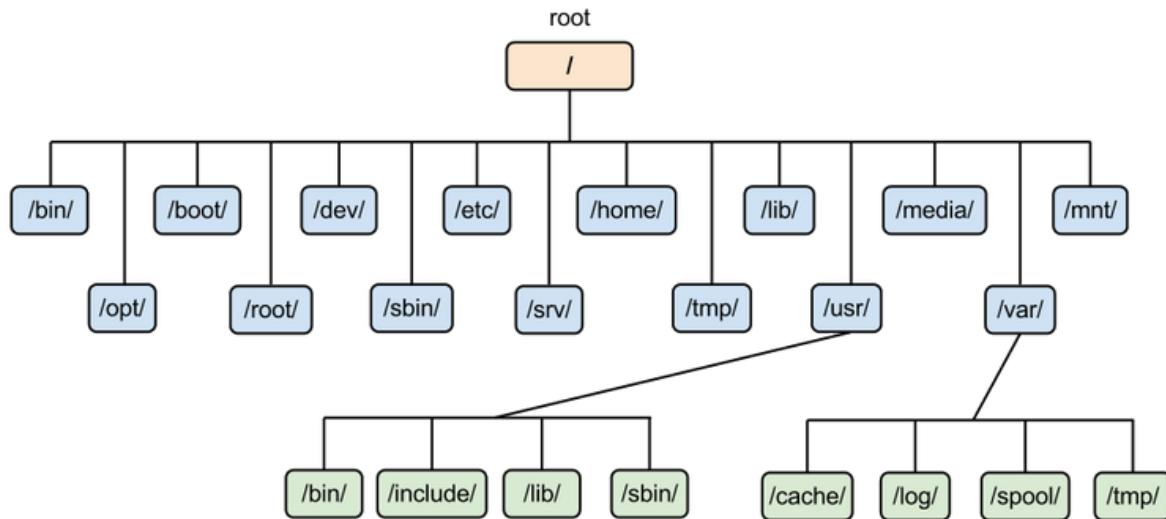


Figure 3.10: โครงสร้างของ โฟลเดอร์ (Folder) หรือ ไดเรกทอรี (Directory) สำคัญ ในระบบปฏิบัติการ Linux ที่มา: <https://freedompenguin.com/articles/how-to-learning-the-linux-file-system/>

6. /sys: เป็นระบบไฟล์เสมือน (Virtual filesystem) เพื่อใช้สำหรับให้โปรแกรมเมอร์เข้าถึงอุปกรณ์ชาร์ดแวร์ เช่น ขา GPIO ต่างๆ ซึ่งจะกล่าวในการทดลองที่ 8 ภาคผนวกที่ H
7. /dev: ถูกสร้างใหม่ทุกครั้งที่บูตเครื่องและโอลอส เพื่อกีบไฟล์ที่เป็นตัวแทนของอุปกรณ์อินพุต/เอาท์พุตต่างๆ ใน ยูสเซอร์สเปช รายละเอียดเพิ่มเติมสามารถอ่านได้จากคำสั่ง df
8. /etc: ใช้กีบไฟล์สำหรับบรรจุไฟล์คอนฟิกของระบบ และแอพพลิเคชันต่างๆ
9. /home: เป็นพื้นที่ส่วนตัวสำหรับผู้ใช้แต่ละคน สำหรับกีบไฟล์ต่างๆ โดยใช้ชื่อโฟลเดอร์ตามชื่อผู้ใช้ภายในโฟลเดอร์ผู้ใช้แต่ละคนแบ่งเป็นพื้นที่หรือโฟลเดอร์ต่างๆ เช่น Documents Desktop Downloads เพื่อใช้เป็นที่จัดเก็บโฟลเดอร์และไฟล์ส่วนตัวของผู้ใช้แต่ละคน ซึ่งจะแยกตามชื่อ ล็อกอิน เช่น /home/user1 /home/user2 เป็นต้น โดย user1 และ user2 คือ ชื่อล็อกอิน
10. /var: เป็นโฟลเดอร์สำหรับเก็บล็อก และข้อมูลอื่นๆ ที่เกิดขึ้นระหว่างการรันระบบ
11. /media: เป็นจุดมาท์หลักสำหรับสื่อเคลื่อนที่ (Removable device) ต่างๆ เช่น แฟลชไดร์ ซีดี/ดีวีดีรอม เป็นต้น
12. /mnt: เป็นจุดมาท์ ชั่วคราว เช่น การแชร์ไฟล์ผ่านเครือข่าย
13. /opt: เป็นโฟลเดอร์สำหรับติดตั้งแอพพลิเคชันจากผู้ขาย
14. /proc: เป็นเวอร์ชวลไดเรกทอรี สำหรับแต่ละprocess ในระบบ
15. /run: เป็นโฟลเดอร์ชั่วคราวสำหรับแอพพลิเคชันในระหว่างที่รันอยู่
16. /tmp: ใช้สำหรับบรรจุไฟล์ชั่วคราวจากแอพพลิเคชันและระบบ

3.2.2 การโหลดซอฟต์แวร์ประยุกต์จากไฟล์ ELF เข้าสู่หน่วยความจำหลัก

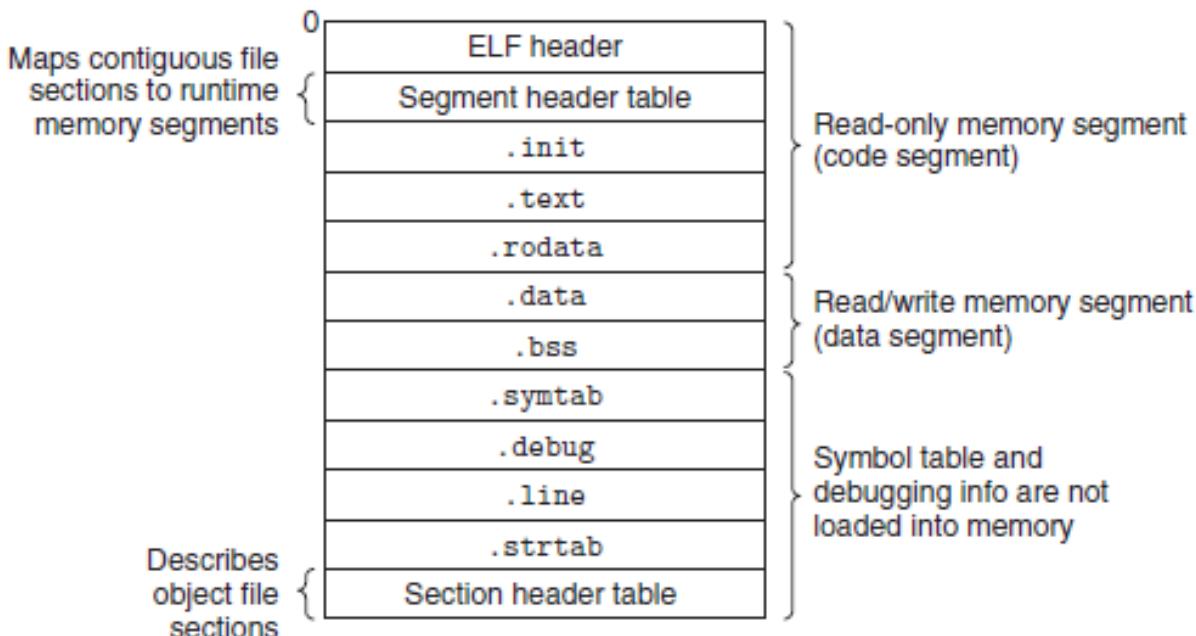


Figure 3.11: โครงสร้างไฟล์สำหรับเก็บชุดคำสั่งและข้อมูลในหน่วยสำรองข้อมูล ที่มา: https://en.wikipedia.org/wiki/Executable_and_Linkable_Format

เมื่อเครื่องคอมพิวเตอร์พร้อมใช้งานหรือบูทเครื่องสำเร็จแล้ว ผู้ใช้จึงสามารถเริ่มต้นเรียกใช้งานซอฟต์แวร์ประยุกต์ได้ ซึ่งสามารถทำได้ตามคำแนะนำใน การทดลองที่ 4 ภาคผนวก D คำสั่งหรือแอพพลิเคชันเหล่านี้ คือ ไฟล์ที่เก็บชุดคำสั่งและข้อมูลประกอบการทำงานเอาไว้ ซึ่งในตระกูลยูนิกซ์ ไฟล์จะอยู่เขียนตามรูปแบบ ELF ในรูปที่ 3.11 ส่วนตระกูล Microsoft Windows ไฟล์จะอยู่เขียนตามรูปแบบ EXE ไฟล์เหล่านี้อยู่ในหน่วยสำรองข้อมูล ดังนั้น ก่อนที่โอเอสจะรันคำสั่งหรือแอพพลิเคชันได้ โอเอสโหลดซอฟต์แวร์ประยุกต์จากหน่วยสำรองข้อมูลเข้าสู่หน่วยความจำหลัก

ELF ย่อมาจาก Executable and Linkable Format เป็นชนิดและโครงสร้างของไฟล์โปรแกรม (Executable) และไฟล์อ็อบเจ็คท์ (Object) ที่ได้จากการคอมไพล์ ELF มีความยืดหยุ่น ขยายเพิ่มเติมได้ง่าย และรองรับการข้ามแพลฟอร์ม (Cross Platform) ระหว่างไมโครโปรเซสเซอร์ และสถาปัตยกรรมของชุดคำสั่งภาษาแอสเซมบลี โครงสร้างของไฟล์ ELF ตามรูปที่ 3.11 ประกอบด้วยพื้นที่หลายส่วนและเซกเมนต์ (Segment) ต่างๆ คือ

- ส่วนหัว (ELF Header) บ่งบอกว่าจะใช้หน่วยความจำขนาด 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

- เท็กซ์เช็กเมนท์ (.text) ใช้สำหรับเก็บชุดคำสั่งหรือแมชีนโคเด็ต ชุดคำสั่งเปรียบเหมือนขั้นตอนการประกอบอาหาร
- ดาต้าเช็กเมนท์ (.rodata) ใช้สำหรับเก็บข้อมูลที่เป็นค่าคงที่ เปรียบเหมือนเครื่องเทศ เครื่องปรุง น้ำตาล เกลือ เป็นต้น
- ดาต้าเช็กเมนท์ (.data และ .data1) ใช้สำหรับเก็บข้อมูลหรือชื่อตัวแปรที่มีค่าตั้งต้น (Initialized Data) และอาจเปลี่ยนแปลงได้เมื่อทำงาน เปรียบเหมือนองค์ประกอบหลักของอาหาร เช่น เนื้อ สัตว์ ผัก เป็นต้น
- ตาราง Section header table อธิบายเซกชัน (Section) ต่างๆ ในหน่วยความจำ ว่าซึ่ออะไร มีความยาวเท่าไหร่

โครงสร้างของไฟล์ หน่วยความจำเพื่อจัดเก็บคำสั่งและข้อมูลต่างๆ รายละเอียดและข้อมูลเพิ่มเติมสามารถสืบค้นได้ทางลิงค์ต่อไปนี้ https://en.wikipedia.org/wiki/Executable_and_Linkable_Format

3.2.3 การอ่านคำสั่งของซอฟต์แวร์ประยุกต์จากหน่วยความจำหลักเพื่อไปปฏิบัติตาม

เมื่อโอเอสอ่านไฟล์แอพพลิเคชันตามรูปแบบ ELF เข้าไปแล้ว โอเอสจะจงพื้นที่ในหน่วยความจำเสมือนให้กับแอพพลิเคชันนั้น ตรงตามชื่อของเช็กเมนท์ต่างๆ ในไฟล์ ELF ในรูปที่ 3.12 เมื่อโหลดไฟล์สำเร็จแล้ว หลังจากนั้น โอเอสจะเปิดโอกาสให้แอพพลิเคชันนั้น ใช้งานซึ่งพิจารณา Permission ที่ได้รับ

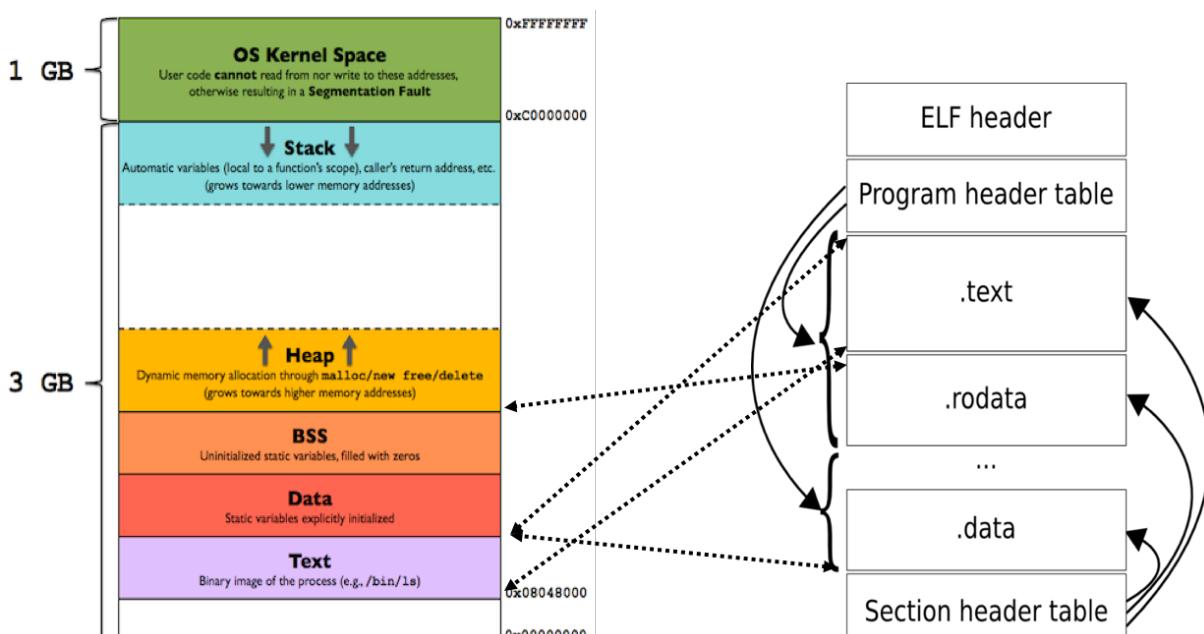


Figure 3.12: โครงสร้าง หน่วย ความ จำ เสมือน เมื่อ โหลด ชุด คำ สั่ง และ ข้อมูล ที่มา: <https://gabrieletolomei.wordpress.com/miscellanea/operating-systems/in-memory-layout/>

โดยตัวราเม่นี้จะครอบคลุมเนื้อหาของระบบปฏิบัติการขนาด 32 บิตเท่านั้น โอเอสจะสามารถสร้างหน่วยความจำเสมือนขนาด 2^{32} หรือ 4 GB เพราะ $2^{32}-1$ kernel เองเป็นโพรเซส ใช้พื้นที่หน่วยความจำเสมือน 1 GB ตั้งแต่ 0xC000 0000 ถึง 0xFFFF FFFF แต่ 0x0000 0000 to 0xBFFF FFFF จะเป็นพื้นที่ 3GB สำหรับ User Space แบ่งเป็นเซกเมนท์ต่างๆ ดังนี้

- เซกเมนท์ BSS (“Block Started by Symbol”) คือ พื้นที่ส่วนหนึ่งในหน่วยความจำเสมือน สำหรับจัดเก็บค่าของตัวแปรชนิด global และ static ที่โปรแกรมเมอร์ยังไม่ได้ตั้งค่าเริ่มต้น แต่โอเอสจะทำการตั้งค่าเริ่มต้นให้กลับเป็น 0 โดยอัตโนมัติ ตำแหน่งของ BSS จะเริ่มต้นต่อเนื่องจากตำแหน่งสุดท้ายของデータเซกเมนท์
- เซกเมนท์สเต็ค (Stack) คือ พื้นที่ส่วนหนึ่งในหน่วยความจำเสมือน สำหรับให้โปรแกรมเมอร์ใช้เก็บค่าของตัวแปรชนิดโลคอล (Local) บางระบบปฏิบัติการจะใช้พื้นที่ของสเต็คเก็บ ค่าตัวแปรในฟังค์ชันเพื่อส่งผ่าน (Pass) และคืนค่า (Return) พารามิเตอร์ระหว่างฟังค์ชันผู้เรียก (Caller Function) และผู้ถูกเรียก (Callee Function) ซึ่งพื้นที่สำหรับใช้งานในแต่ละฟังค์ชัน เรียกว่า สเต็คเฟรม (Stack Frame) มีสแต็คพอยเตอร์ (Stack POINTR) เก็บค่าแอดдресหน่วยความจำชั้นบนสุด เพื่อความสะดวกในการบริหารจัดการ

การบริหารสแต็คเฟรมของแต่ละฟังค์ชัน จะมีลักษณะเป็น LIFO (Last In First Out) นั่นคือ ในตอนเริ่มต้นรันแอปพลิเคชัน สแต็คเซกเมนท์ของโปรแกรมนี้ยังว่างเปล่า เมื่อมีการเรียกใช้ฟังค์ชันโปรแกรมเมอร์หรือคอมไพล์เตอร์ จะสร้าง หรือ Push สแต็คเฟรมลงไปในบริเวณสแต็คเซกเมนท์ ณ ตำแหน่งสแต็คพอยเตอร์ปัจจุบัน (Current Stack POINTR) ทำให้สแต็คพอยเตอร์ย้ายตำแหน่งไปที่ตำแหน่งใหม่ โปรแกรมเมอร์สามารถใช้งานพื้นที่ภายในสแต็คเฟรมนี้ ตามที่กล่าวไปแล้ว เมื่อฟังค์ชันทำงานเสร็จสิ้น โปรแกรมเมอร์จะต้องรีเทิร์นออกจากฟังค์ชัน โดยคืนพื้นที่ (Pop) สแต็คเฟรมออกจากหน่วยความจำ เพื่อให้สแต็คว่างลง

หากมีการเรียกฟังค์ชันภายในฟังค์ชัน (Nested Function) สแต็คเฟรมจะถูกวางเรียงต่อกันไป โดยเฉพาะการเรียกฟังค์ชันแบบรีเครอเรสีฟ(Recursive) หรือแบบเรียกตัวเอง สแต็คเฟรมจะเรียงต่อกันไปเรื่อยๆ จนกว่าจะรีเทิร์นกลับ หากไม่มีการรีเทิร์นกลับ พื้นที่ของสแต็คเซกเมนท์จะโตขึ้นจนถึงขีดจำกัด เรียกว่า RLIMIT_STACK หรือ จนสแต็คพอยเตอร์超出了ทับกับตำแหน่งของฮีพ และทำให้เกิด Exception เพื่อให้ระบบปฏิบัติการมาบริหารจัดการต่อไป

- ฮีพ (Heap) คือ เซกเมนท์หนึ่งในหน่วยความจำเสมือน อนุญาตให้โปรแกรมเมอร์จดหน่วยความจำเพิ่มเติม เช่น ลิงค์ลิสต์ (Linked List) และโครงสร้างข้อมูลชนิดต่างๆ โดยใช้คำสั่ง malloc ในภาษา C และคำสั่ง new ในภาษา C++ ฮีพสามารถขยายขนาดเพิ่มเติมได้ จนถึงตำแหน่งบนสุดของสแต็คพอยเตอร์ (Stack POINTR) โปรแกรมเมอร์สามารถคืนหน่วยความจำที่จองได้ โดยใช้คำสั่ง free ในภาษา C และคำสั่ง delete ในภาษา C++

รายละเอียดเพิ่มเติมสามารถศึกษาและทดลองได้จากการทดลองที่ 4 ภาคผนวกที่ D และการทดลองอื่นตามลำดับ

3.2.4 ซอฟต์แวร์ประยุกต์อ่าน/เขียนข้อมูลในหน่วยความจำเพื่อให้ชีพิญประมวลผล

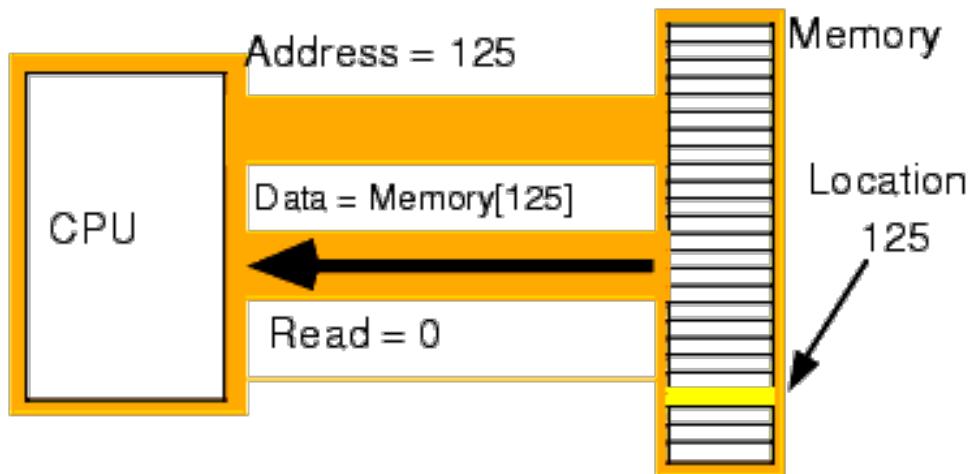


Figure 3.13: ขบวนการ อ่าน ข้อมูล จาก หน่วย ความ จำหลัก ที่ ตำแหน่ง 125 ที่มา: <http://www.phatcode.net/res/260/files/html/SystemOrganizationa2.html>

ข้อมูลนิดค่าคงที่ เช่น ค่า π หรือ ตัวแปรต่างๆ อาศัยพื้นที่ในหน่วยความจำหลักในการเก็บค่าตามชนิดของข้อมูลนั้น ตามตารางที่ 2.1 เมื่อต้องการค่าตัวแปรเหล่านี้ไปประมวลผล ชีพิญต้องอ่านค่า (Load) ไปพักในรีจิสเตอร์ โดยแบ่งเป็น รีจิสเตอร์เลขจำนวนเต็ม และ รีจิสเตอร์เลขทศนิยม รีจิสเตอร์เลขจำนวนเต็ม จะใช้เก็บค่าตัวแปรชนิด char, int, unsigned int, long เป็นต้น ส่วนรีจิสเตอร์เลขทศนิยม จะใช้เก็บค่าตัวแปรชนิด IEEE Single Precision และ Double Precision ทั้งนี้ขึ้นอยู่กับยาร์ดแวร์

เมื่อประมวลผลเสร็จสิ้น ชีพิญต้องนำค่าในรีจิสเตอร์เก็บ (Store) หรือเขียนกลับเข้าสู่หน่วยความจำหลัก ณ ตำแหน่งหรือแอดเดรสของตัวแปรนั้นๆ ตัวอย่างโปรแกรมในตารางที่ 3.3 คำสั่งที่บอกให้ชีพิญอ่านหรือเขียนข้อมูล เรียกว่าคำสั่งประเภท Load/Store ซึ่งจะได้กล่าวในรายละเอียดในบทที่ 4

ในด้านความปลอดภัย ซอฟต์แวร์หรือแอพพลิเคชันใดๆ อ่านหรือเขียนข้อมูล ได้เฉพาะในเซกเมนท์ที่ โอล์สอยอนุญาต คือ data และ stack แอพพลิเคชันไม่สามารถเขียนหรือแก้ไขหน่วยความจำในเซกเมนท์ .text ได้ หากกระทำเช่นนั้น โอล์สจะสามารถตรวจสอบจับได้

3.2.5 การใช้งานอินพุตและเอาท์พุตต่างๆ

ภายในแกนหรือเครื่องเนลของโอล์ส 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 Device Driver ซึ่งกำหนดรายละเอียดเอาไว้สำหรับผู้ผลิตและอุปกรณ์แต่ละรุ่น การเชื่อมต่อลักษณะนี้ ได้แก่ Terminal

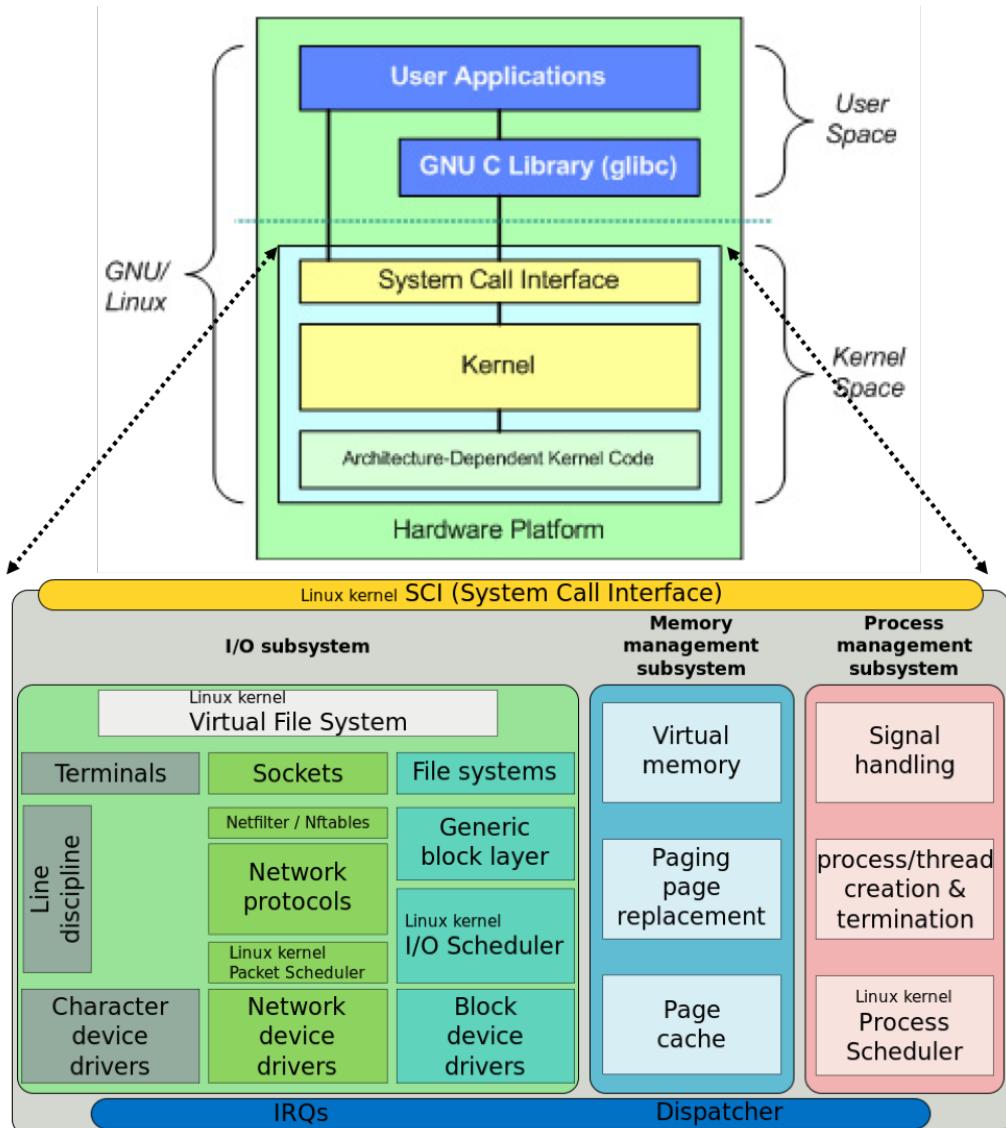


Figure 3.14: โครงสร้างของลีนุกซ์เครื่องเนล โดยเน้นส่วนที่เป็น I/O, Memory Management และ Process Management (https://en.wikipedia.org/wiki/Completely_Fair_Scheduler)

การเชื่อมต่อกับเครือข่ายอินเทอร์เน็ตผ่านอุปกรณ์สื่อสารชนิดสายและไร้สาย จะอยู่ในรูปของ Network Interface ซึ่งจำเป็นต้องอาศัยโปรแกรมดีไวซ์ไดรเวอร์ (Device Driver) เช่นกัน การเชื่อมต่อลักษณะนี้ มีชื่อเฉพาะเรียกว่า ซ็อกเก็ต (Socket)

การใช้งานอินพุตและเอาท์พุต เช่น หน่วยสำรองข้อมูล เครื่องอ่านซีดี/ดิวีดีรอม (CD/DVD Rom Drive) แฟลชไดร์ (Flash Drive) เป็นต้น จะอาศัยการเชื่อมต่อกับอุปกรณ์เหล่านี้ในลักษณะของ Block ข้อมูลขนาดตั้งแต่ 512 ไบท์ขึ้นไป การอ่านหรือเขียนข้อมูลจากไฟล์ จะต้องผ่านกระบวนการนี้ เช่นกัน ซึ่งโอลีสจะเป็นผู้บริหารจัดการ เปิดไฟล์ อ่าน/เขียนข้อมูล และปิดไฟล์ การเชื่อมต่อลักษณะนี้ มีชื่อเฉพาะเรียกว่า File Systems การใช้งานอุปกรณ์อินพุต/เอาท์พุต ทั้งสามรูปแบบ เพื่อเชื่อมต่ออุปกรณ์ต่างๆ กับระบบปฏิบัติการ ในรูปแบบของ Virtual File System

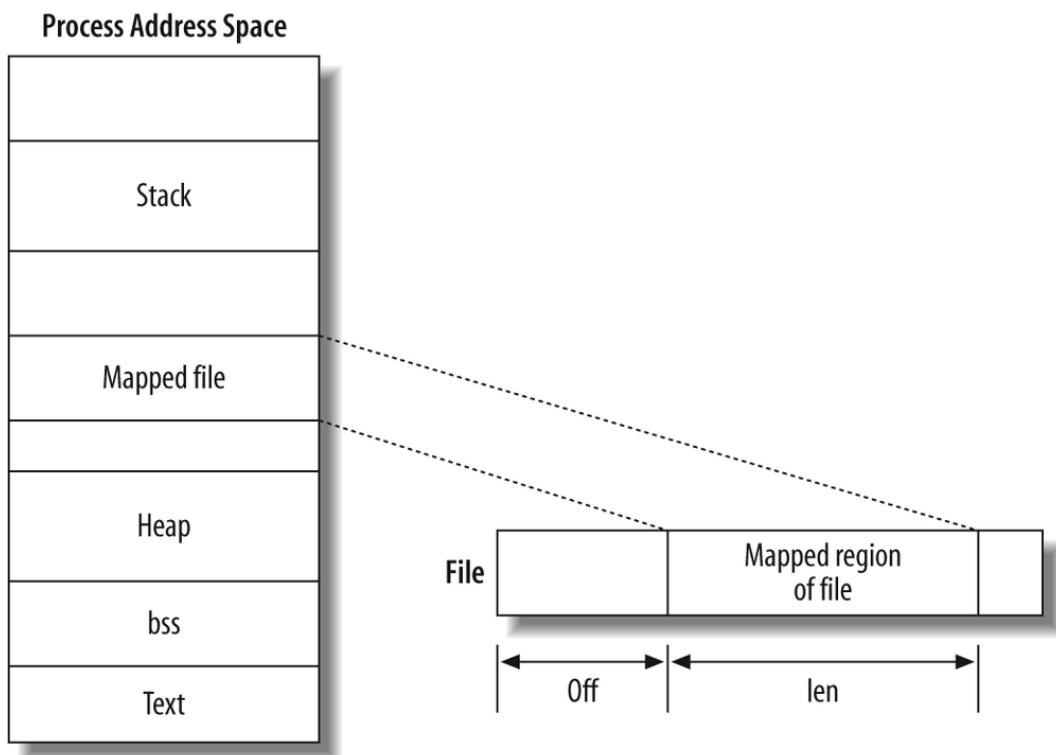


Figure 3.15: หลัก การ ของ Memory Map File คือ การ แมป หน่วย ความ จำ สำหรับ อ่าน หรือ เขียน ไฟล์ ด้าน ขวา คือ โครงสร้าง ของ ไฟล์ ใน เชิง ตรรกะ ด้าน ซ้าย คือ หน่วย ความ จำ ที่ ถูก จง เพื่อ ใช้ พก ข้อมูล ใน ไฟล์ ที่มา: <https://www.safaribooksonline.com/library/view/linux-system-programming/0596009585/ch04s03.html>

3.2.6 ซอฟต์แวร์ประยุกต์เขียนข้อมูลบันทึกในหน่วยสำรองข้อมูล

ไฟล์ คือ ข้อมูล หรือ คำสั่ง ที่ เรียง ตัว ต่อ กัน ตั้งแต่ ต้น ไฟล์ ไป จน สิ้น สุด ไฟล์ เมื่อ เจอ สัญลักษณ์ EOF (End of File) ซึ่ง เป็น เลข จำนวนเต็ม มี ค่า เท่า กับ -1 การ ใช้งาน ไฟล์ จะ ต้อง เริ่ม ด้วย การ เปิด ไฟล์ เสมอ การ เปิด ไฟล์ คือ การ จง หน่วย ความ จำ ให้ กับ ไฟล์ ที่ ต้อง การ เพื่อ อ่าน หรือ เขียน เมื่อ เริ่ม ต้น เปิด ไฟล์ ข้อมูล หรือ คำสั่ง ใน ไฟล์ จะ ถูก อ่าน ข้าม พก เก็บ ใน หน่วย ความ จำ ที่ ถูก จง ตาม หลัก การ ที่ เรียกว่า Memory Map File คือ การ แมป หน่วย ความ จำ สำหรับ อ่าน หรือ เขียน ไฟล์ รูปที่ 3.15 การ เปิด ไฟล์ มี หลาย ทาง เลือก ได้ แก่ เปิด ไฟล์ เพื่อ อ่าน เพื่อ อ่าน และ เขียน เป็น ต้น

ไฟล์ มี หลาย ชนิด เช่น ไฟล์ ตัว อักษร เรียกว่า Text File ไฟล์ ข้อมูล ไฟล์ บรรจุ โปรแกรม คำสั่ง (Executable File) ตาม รูปแบบ ELF ที่ กล่าว ใน หัวข้อ ที่ 3.2.2 เป็น ต้น

เมื่อ คำสั่ง หรือ แอพ ประมวล ผล ข้อมูล ตาม ที่ ได้ รับ มอบ หมาย เรียบร้อย แอพพลิเคชัน สามารถ บันทึก ข้อมูล หรือ สารสนเทศ เก็บ ลง ใน ไฟล์ ภายใน หน่วย ความ จำ สำรอง ตาม รูปที่ 3.13 ทั้งนี้ ราย ละเอียด โครงสร้าง ไฟล์ ข้อมูล แต่ ละ ชนิด จะ แตกต่าง กัน ไป เช่น ไฟล์ รูปภาพ JPEG (.jpg) ไฟล์ รูปภาพ PNG (.png) ไฟล์ เสียง WAV (.wav) ไฟล์ ภาพเคลื่อน ไหว MPEG4 (.mp4) เป็น ต้น

การ อ่าน ไฟล์ คือ การ อ่าน ข้อมูล จาก หน่วย ความ จำ บริเวณ ที่ ตรง กับ ไฟล์ โดย ข้อมูล จาก ไฟล์ ใน หน่วย สำรอง ข้อมูล จะ ถูก อ่าน ขึ้น มา บรรจุ ใน หน่วย ความ จำ ตาม ที่ กำหนด

เมื่อ คำสั่ง หรือ แอพ ประมวล ผล ข้อมูล ตาม ที่ ได้ รับ มอบ หมาย เรียบร้อย แอพ สามารถ บันทึก ข้อมูล หรือ สารสนเทศ เก็บ ลง ใน หน่วย ความ จำ สำรอง การ เขียน ไฟล์ คือ การ เขียน ข้อมูล ไป ยัง หน่วย ความ จำ บริเวณ ที่

ตรงกับไฟล์ โดยหน่วยความจำตำแหน่งนั้นๆ โดยระบบปฏิบัติการจะทยอยเขียนข้อมูลลง ในหน่วยสำรองข้อมูลจริง ตามขนาดของบัฟเฟอร์ ดังนั้น เมื่อใช้งานไฟล์เสร็จสิ้นแม้ว่าจะเพียงแค่อ่านไฟล์ โปรแกรมเมอร์ จะต้องปิดไฟล์เสมอ การปิดไฟล์ คือ การนำข้อมูลที่ยังคงในหน่วยความจำเพื่อย้ายข้อมูลเหล่านั้นไปเขียนต่อในหน่วยสำรองข้อมูลจริง และคืนหน่วยความจำที่จองไว้กลับให้ระบบปฏิบัติการ

3.2.7 การขัดดาวน์ (Shut Down) ระบบปฏิบัติการก่อนหยุดจ่ายไฟ

เป็นการสั่งให้ระบบปฏิบัติการอพเดทข้อมูลต่างๆ ของเครื่องในรูปของไฟล์ต่างๆ กลับลงในหน่วยสำรองข้อมูล และปิดไฟล์ที่เปิดค้างไว้ การปิดเครื่องโดยไม่สั่งขัดดาวน์จะทำให้การบุหเครื่องครั้งต่อไปมีปัญหาและจะทำให้ระบบปฏิบัติการทำงานได้ไม่สมบูรณ์

`shutdown -h ขัดดาวน์ส่งสัญญาณ (Signalling)` ไปยัง init เพื่อเปลี่ยนระดับการรัน (Run Level) ให้เป็นระดับ 0 (Halt) เพื่อปิดระบบ ซึ่งปกติจะอยู่ระดับ 3 หรือระดับ 5 เป็นค่าดีฟอลต์ (Default)

`shutdown -r` คือการรีสตาร์ทเครื่อง เปลี่ยนระดับการรันเป็น 6 นิยมใช้กับระบบฟังตัว เพื่อเริ่มต้นการทำงานใหม่ ลักษณะหน่วยความจำที่ไม่ได้ใช้ โปรแกรมเมอร์สามารถเขียนสคริปต์ (Script) ระบบ Microsoft Windows ทำได้ เช่น กัน แต่เรียกว่า การเขียนคำสั่งแบบที่ (Batch Command)

3.3 โครงสร้างของซอฟต์แวร์สโคด์ (Source Code) โปรแกรม

คอมพิวเตอร์

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 มีตัวอย่างโปรแกรมที่นักพัฒนาทั่วโลกเปิดเผยแพร่สโคด์ผ่านทางเครือข่ายอินเทอร์เน็ต ผู้อ่านสามารถค้นคว้าเพิ่มเติมได้ที่ลิงค์ต่อไปนี้ <https://www.raspberrypi.org/documentation/usage/python/> เป็นต้น

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

3.3.1 โครงสร้างของซอฟต์แวร์สโคด์โปรแกรมภาษา C/C++

ภาษาระดับสูง (High-level language) การเขียนโปรแกรมด้วยภาษาระดับสูงเพื่อแก้โจทย์หรือปัญหา โปรแกรมเมอร์สามารถเขียนโปรแกรมได้ง่ายและสร้างสรรค์ผลงานได้ง่ายกว่าภาษาระดับล่าง เช่น ภาษาแอลซีเมบลี และภาษาเครื่อง เนื่องจากภาษาระดับสูงมีลักษณะใกล้เคียงกับประโยคภาษาอังกฤษ ภาษาระดับสูงจะไม่ยึดติดกับเครื่องหรือฮาร์ดแวร์ (Machine Independent) ในทางตรงกันข้าม ภาษาระดับล่าง เช่น ภาษาแอลซีเมบลีและภาษาเครื่องจะใช้มโนye โดยตรงกับฮาร์ดแวร์หรือจรวจหรือไมโครโปรเซสเซอร์ ภายในเครื่องคอมพิวเตอร์นั้นๆ (Machine Specific) โปรแกรมเมอร์สามารถใช้คำสั่งภาษาเหล่านี้ในการสั่งงานฮาร์ดแวร์ของเครื่องคอมพิวเตอร์ ในตำรานี้จะใช้ภาษา C/C++ เป็นหลัก เพื่อเข้าถึงภาษาแอลซีเมบลี และภาษาเครื่อง ไปจนถึงฮาร์ดแวร์ได้ลึกซึ้งมากกว่า

ตัวอย่างซอฟต์แวร์สโคด์ภาษา C พังค์ชันหลัก ในรูปที่ 3.16 ชื่อไฟล์ main.c พังค์ชันหลักชื่อ main() เป็นพังค์ชันที่โปรแกรมเริ่มต้นทำงาน เปรียบได้กับการทำงานโดยไม่แสดงรายละเอียด เมื่อทำงานเสร็จสิ้นจะรีเทิร์นค่า 0 ซึ่งเป็นเลขจำนวนเต็มค่าหนึ่ง ซึ่งมักใช้ตรวจสอบว่าเป็นการทำงานเสร็จสิ้นโดยไม่มีปัญหาหรือข้อผิดพลาด และรีเทิร์นค่าอื่นๆ ที่ไม่ใช่ 0 เพื่อบอกรหัสความผิดพลาดได้

3.3.2 การคอมไพล์โปรแกรมภาษา C/C++

ภาษาระดับสูง (High-level language) มีความใกล้เคียงกับโจทย์หรือปัญหา โปรแกรมเมอร์สามารถเขียนโปรแกรมได้ง่ายและสร้างสรรค์ผลงานได้ง่ายกว่า

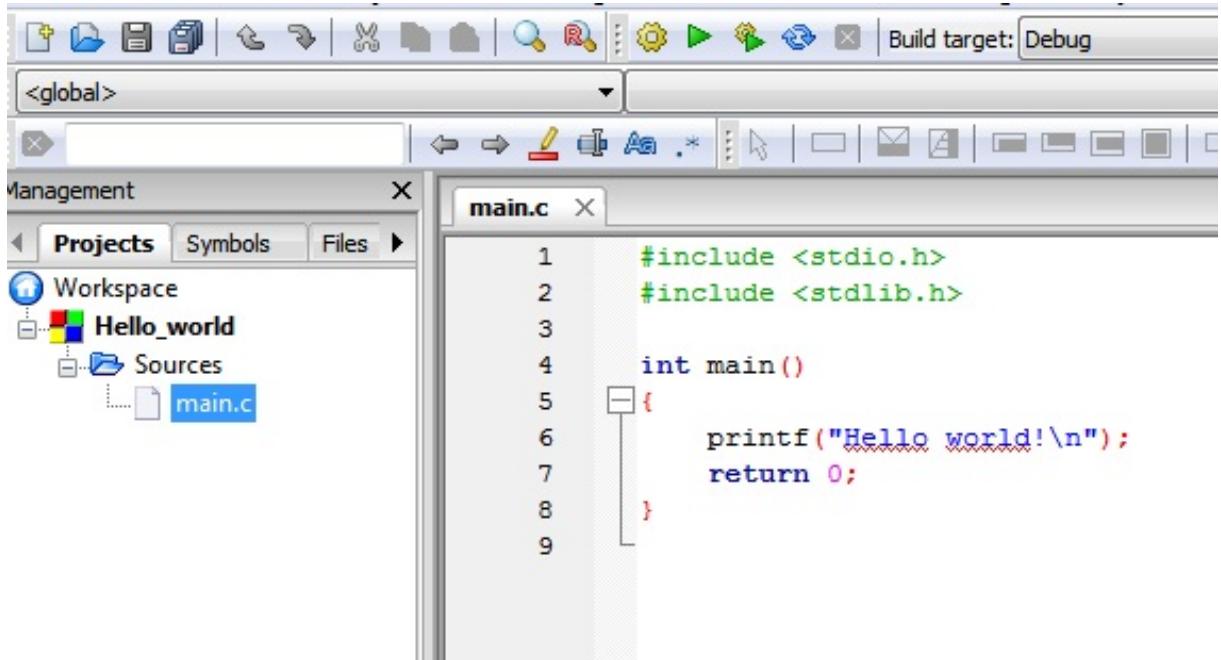


Figure 3.16: ตัวอย่างซอฟต์แวร์ C ที่แสดงไฟล์ main.c ซึ่งมีโค้ดพื้นฐานพิมพ์คำว่า Hello world!

Source code รูปที่ 3.17 โฟล์กการพัฒนาซอฟต์แวร์ จากซอฟต์แวร์ Executable หรือโปรแกรมบนระบบปฏิบัติการยูนิกซ์ คอมไพล์เป็นโปรแกรมคอมพิวเตอร์ชนิดหนึ่งที่ทำหน้าที่แปลงซอฟต์แวร์ภาษาสูงให้กลายเป็น คำสั่งภาษาแอสเซมบลีและคำสั่งภาษาเครื่องในที่สุด และจึงจัดเรียงคำสั่งภาษาเครื่องให้มีโครงสร้างตามไฟล์ ELF ซึ่งอธิบายแล้วในหัวข้อที่ 3.2.2

คำสั่งภาษาเครื่อง คือ เลขฐานสอง ในขณะที่ข้อมูลเป็นเลขฐานสองเช่นกัน ดังนั้น เช็คเมนท์ต่างๆ ของไฟล์ ELF จะเป็นสิ่งกำกับว่า เลขฐานสองแต่ละค่า คือ คำสั่ง หรือ ข้อมูล ยกตัวอย่างเช่น

- เลขฐานสองที่จัดเรียงอยู่ใน Data เช็คเมนท์ คือ ข้อมูล เท่านั้น
- เลขฐานสองที่จัดเรียงอยู่ใน Text เช็คเมนท์ คือ คำสั่ง เท่านั้น

คอมไابل์ภาษา Java และ C# จะแปล Data Flow Graph ให้กลายเป็น Intermediate Code ยกตัวอย่างเช่น

- Java bytecode ในระบบ Java ผู้อ่านสามารถศึกษาเพิ่มเติมได้ที่ https://en.wikipedia.org/wiki/Java_bytecode
- CIL (Common Intermediate Language) ในระบบ .NET ผู้อ่านสามารถศึกษาเพิ่มเติมได้ที่ https://en.wikipedia.org/wiki/Common_Intermediate_Language

Common Intermediate Code คือ ชุดคำสั่งและข้อมูลที่จะแปลอีกรอบก่อนที่จะรัน ยกตัวอย่าง เช่น โค้ดที่ได้จากการคอมไابل์โปรแกรม C# ด้วย .NET คอมไابل์ เรียกว่า CIL และ CLR (Common Language Runtime) จะแปลโค้ด CIL อีกรอบก่อนที่จะรัน ให้เป็น Native Machine Code ก่อนการทำงานจริง การคอมไابل์แบบนี้ว่า JIT ย่อมาจากคำว่า Just In Time แปลว่า ทันทีที่ต้องการ โดยส่วนที่เรียกว่า Run Time Environment ของระบบปฏิบัติการจะเป็นผู้ดำเนินการ สำหรับภาษา Java จะถูก

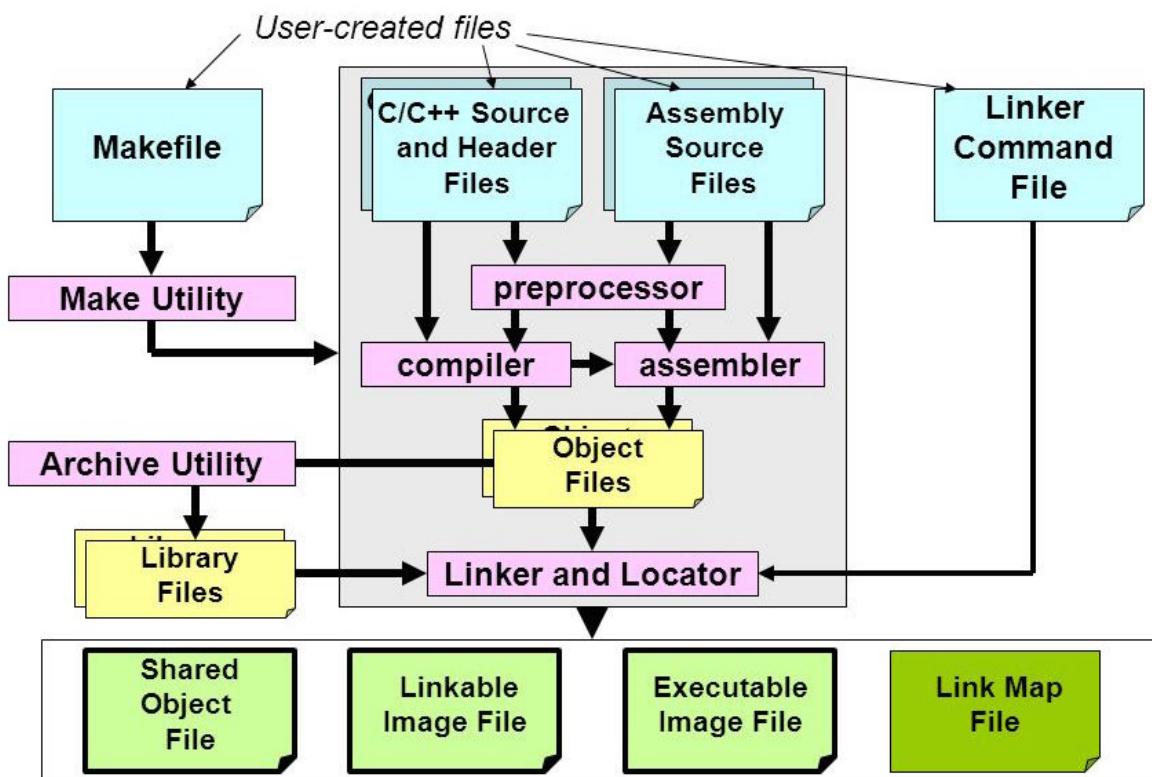


Figure 3.17: โฟล์กการพัฒนาซอฟต์แวร์ จากชอร์สโค้ดสู่ไฟล์ Executable (ELF) หรือโปรแกรมประยุกต์ (Application Program) หรือย่อๆว่า แอพ (ที่มา: <https://slideplayer.com/slide/4804417/>)

เรียกว่า JRE สำหรับภาษา .NET จะเรียกว่า .NET Framework ซึ่งมีภาระงานจะเปลี่ยนความหมายของคำ สั่งภาษาเครื่อง หรือ Native Machine Code ในขณะที่ เวอร์ชวลแมชชีนจะแปลง Intermediate Code ให้กลายเป็นคำสั่งภาษาเครื่องอีกทوذันนึง

การคอมไพล์สามารถสร้างผลลัพธ์ได้ 4 รูปแบบ ตามที่โปรแกรมเมอร์ต้องการ คือ

- Executable Image File (ชื่อไฟล์ a.out) ซึ่งโครงสร้างของ Executable Image File หรือ ELF ในรูปที่ 3.11
- Shared Object File (*.o)
- Linkable Image File (*.lib)
- Link Map File (*.?)

คอมไพล์เลอร์ต้องทำความเข้าใจ ด้วยการอ่านไฟล์ชอร์สโค้ด สแกนตัวอักษรที่เป็นเนื้อโปรแกรม จัดกลุ่มคำ (Token) ตรวจคำสะกด แปลความหมายของคำ สร้างแผนภูมิต้นไม้ (Abstract Syntax Tree) เพื่อตรวจสอบไวยากรณ์และ ความหมาย (Semantic Analysis) ของประโยคต่างๆ เมื่อปราศจากข้อผิดพลาด คอมไпал์เลอร์จะดำเนินการต่อ เพื่อปรับปรุงการทำงานของโปรแกรมที่จะคอมไпал์ให้ดีขึ้น เรียกว่า การอปติไมซ์ (Optimize) โดยการสร้าง Data-Flow graph เพื่อประกอบการวิเคราะห์ หลังจากนั้น คอมไпал์เลอร์จะสร้างแมชชีนโค้ด (Machine Code) ที่ได้จากการที่ผ่านการอปติไมเซชัน (Optimization) แล้ว ผู้อ่านสามารถศึกษารายละเอียดเพิ่มเติม ได้ที่ <https://en.wikipedia.org/wiki/Compiler>

โปรแกรมเมอร์จะพัฒนาหรือสร้างไฟล์ชอร์สโค้ดต่างๆ กำหนดความสัมพันธ์หรือความเชื่อมโยงระหว่างชอร์สโค้ดเหล่านี้ใน Makefile ในการทดลองที่ 5 ในภาคผนวก E การทดลองที่ 5 ภาคผนวก E การทดลองเพื่อพัฒนาโปรแกรมภาษา C บนบอร์ด Pi3 และระบบปฏิบัติการ Linux

3.3.3 โครงสร้างของชอร์สโค้ดโปรแกรมภาษา Assembly

ภาษาแอสเซมบลี (Assembly language) เป็นคำสั่งภาษาเครื่องที่อยู่ในรูปของตัวอักษรย่อ (Mnemonic) ทำให้โปรแกรมเมอร์เข้าใจง่ายกว่าตัวเลขฐานสองที่เป็นคำสั่งภาษาเครื่อง (Machine Instruction) ในตำรานี้จะใช้ภาษาแอสเซมบลีของ ARM Cortex A เวอร์ชัน 32 บิตเป็นหลัก รายละเอียดในบทที่ 4

Table 3.3: ตัวอย่างโปรแกรมภาษาแอสเซมบลีเพื่อสร้างตัวแปรขนาด 32 บิต จำนวน 2 ตัวแปร

#	Label	Operation	Registers or Addresses or Labels or Constants
1		Area	CODE
2		ENTRY	
3		LDR	R1, M
4		LDR	R2, POINTR
5		MOV	R0, #0
6	LOOP	LDR	R3, [R2], #4
7		ADD	R0, R0, R3
8		SUBS	R1, R1, #1
9		BGT	LOOP
10		STR	R0, SUM
11			
12		AREA	DATA
13	SUM	DCD	0
14	M	DCD	4
15	POINTR	DCD	NUM
16	NUM	DCD	3, 5, 7, 9
17		END	

ในภาพรวม ตารางที่ 3.3 แสดงให้เห็นถึง โครงสร้างหลักของชอร์สโค้ดภาษา Assembly ของ ARM แบ่งเป็น Area CODE และ Area DATA หมายถึง Code เซกเมนท์ และ Data เซกเมนท์ ตามลำดับ ส่วนคำสั่งภาษาแอสเซมบลี (Assembly language) ในบรรทัดอื่นๆ อยู่ในรูปของตัวหนังสือที่โปรแกรมเมอร์สามารถทำความเข้าใจได้ และมีความใกล้เคียงกับคำสั่งภาษาเครื่อง ประกอบด้วย พัคซัน main ประกอบด้วย 3 คอลัมน์ ดังนี้

- คอลัมน์ซ้าย ใช้กำหนดค่า Label ต่างๆ
- คอลัมน์กลาง ใช้กำหนด Operation ว่าต้องการทำอะไร
- คอลัมน์ขวา ใช้ระบุค่า รีจิสเตอร์ หรือ แอดเดรส หรือ เลเบล หรือ ค่าคงที่สำหรับ Operation ในคอลัมน์ตรงกลาง

ในภาพลักษณ์ เอียง ตัวอย่างโปรแกรมในตารางที่ 3.3 สามารถอธิบายตามหมายเหตุบรรทัดได้ดังนี้

1. AREA CODE หมายถึง ตำแหน่งเริ่มต้นของ CODE เช็คเมนท์
2. ENTRY หมายถึง ตำแหน่งเริ่มต้นของ CODE ซึ่งจะเป็นพังค์ชันให้อโศกเริ่มทำงานโปรแกรม
3. คำสั่ง LDR ย่อมาจาก Load Data Register ดังนั้น LDR R1, M คือ การโหลดค่าของตัวแปร N มาเก็บในรีจิสเตอร์ R1
4. คำสั่ง LDR R2, POINTR คือ การโหลดค่าของตัวแปร POINTR ซึ่งเก็บแอดдресของตัวแปร NUM มาเก็บในรีจิสเตอร์ R2
5. คำสั่ง MOV ย่อมาจาก MOVE ดังนั้น MOV R0, #0 คือ การตั้งค่าของรีจิสเตอร์ R0 ให้มีค่าเป็น 0
6. คำสั่ง LDR R3, [R2], #4 คือ การโหลดค่าของตัวแปร NUM[0] มาเก็บในรีจิสเตอร์ R3 และจึงเพิ่มค่า R2 เป็น $R2+4$ โดยคำสั่งนี้จะกำหนดให้มี Label ชื่อ LOOP ซึ่งต้องการจะให้เกิดการทำซ้ำ
7. คำสั่ง ADD หมายถึง การบวกค่าภายในรีจิสเตอร์ โดยบวกค่า $R0+R3$ และเก็บผลลัพธ์ใน R0
8. คำสั่ง SUBS ย่อมาจาก Subtract และอัพเดทผลลัพธ์ในรีจิสเตอร์สถานะ (CPSR: Current Program Status Register) ซึ่งรายละเอียดจะกล่าวในบทที่ 4 ดังนั้น R1 จะลดลง 1 หน่วย
9. คำสั่ง BGT ย่อมาจาก Branch Greater Than คำสั่งนี้จะตรวจสอบผลของคำสั่งก่อนหน้า โดย
 - หาก $R1-1$ และค่ามากกว่า 0 และ การทำงานจะกลับไปเริ่มที่คำสั่งที่ตั้งกับเลเบล LOOP ดังนั้น โปรแกรมนี้จะวนรอบ $M=4$ ครั้ง เพื่อบวกค่าของ NUM[0] จนถึง NUM[3]
 - หาก $R1-1$ และน้อยกว่าหรือเท่ากับ 0 ซึ่งหมายความว่าทำงานที่คำสั่งต่อไป คือ STR R0, SUM เพราะวนครับจำนวน $M=4$ รอบแล้ว
10. คำสั่ง STR ย่อมาจาก Store Register คือ การนำผลบวกในรีจิสเตอร์ R0 ไปเก็บไว้ที่ตัวแปร SUM ซึ่งอยู่ในหน่วยความจำหลัก
11. เว้นวรรค เพื่อความสวยงาม
12. AREA DATA หมายถึง ตำแหน่งเริ่มต้นของ CODE เช็คเมนท์
13. SUM ซึ่งของตัวแปรชนิดจำนวนเต็ม (Integer) ถูกกำหนดค่าเริ่มต้น ให้มีค่าเป็น 0
14. M คือ ซึ่งของตัวแปรชนิดจำนวนเต็ม (Integer) ถูกกำหนดค่าเริ่มต้น ให้มีค่าเป็น 4
15. NUM คือ ซึ่งของตัวแปรชนิดของเรียกของเลขจำนวนเต็มที่จะถูกกำหนดค่าเริ่มต้น ให้มีค่าเป็น 3, 5, 7, 9 ตามลำดับ โดย $NUM[0] = 3$ และ $NUM[3] = 9$

16. POINTR ชื่อของตัวแปรชนิดพอยท์เตอร์ที่จะถูกกำหนดค่าเริ่มต้น ให้มีค่าเป็นแอ็ดเดรสของ NUM หากไม่มีการกำหนดค่าเริ่มต้นด้วยคำสั่ง DCD ค่าของตัวแปรชนิดพอยท์เตอร์จะเก็บแอ็ดเดรส NULL?
DCD = การกำหนดค่าเริ่มต้นให้กับชื่อตัวแปรทางข้ามือ ด้วยพื้นที่ขนาด 1 Word หรือเท่ากับ 4 ไบต์
17. END บอกจุดสิ้นสุดของซอร์สโค้ด เมื่อโปรแกรมที่เขียนนี้เสร็จสิ้นการทำงานจะรีเทิร์นกลับไปหา ไออีส

3.3.4 ลิงค์เกอร์ (Linker)

ลิงค์เกอร์ (Linker) เป็นซอฟต์แวร์สำคัญในกระบวนการพัฒนาโปรแกรมในรูปที่ 3.17 เพราะเป็นการขยายขีดความสามารถของโปรแกรมที่พัฒนาใหม่จากไฟล์ Object และ Library ที่มีอยู่เดิม การทำงานร่วมกันสามารถทำได้ในรูปของ การเรียกใช้ฟังค์ชัน (Function Call) ที่มีผู้พัฒนาไว้แล้ว ตรงตามหลักการที่สำคัญ พัฒนาซอฟต์แวร์ที่ดี เรียกว่า โมดูลาริตี้ (Modularity) ไฟล์ไลบรารีมาตรฐานของแต่ละภาษา คือ รวบรวมฟังค์ชันสำหรับที่ถูกพัฒนาอย่างต่อเนื่องจนน่าเชื่อถือ ไฟล์ออบเจ็คท์ คือ การรวมรวมฟังค์ชันที่นักพัฒนาเก็บไว้ส่วนตัว รายละเอียดสามารถศึกษาเพิ่มเติมจากการทดลองที่ 5 ภาคผนวก E

ในรูปที่ 3.18 ตัวอย่างการลิงค์ (Link) หรือรวมไฟล์ Object และ Library เข้าด้วยกันให้เป็นไฟล์ โปรแกรมหรือแอพพลิเคชัน กล่องสีเหลี่ยมในรูป คือ ไฟล์ที่บรรจุคำสั่งและข้อมูลเป็นเลขฐานสอง แต่เพื่อให้ผู้อ่านเข้าใจได้ง่าย คำสั่งจึงแสดงเป็นภาษา Assembly แทน คำอธิบายเพิ่มเติมในหัวข้อที่ 4.8 เรื่องการเรียกใช้ฟังค์ชัน

3.3.5 โครงสร้างของคำสั่งภาษาเครื่อง (Machine Code)

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

คำสั่ง SUB R5, R7, R1 จะถูกแปลงตามบิตต่อๆ

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

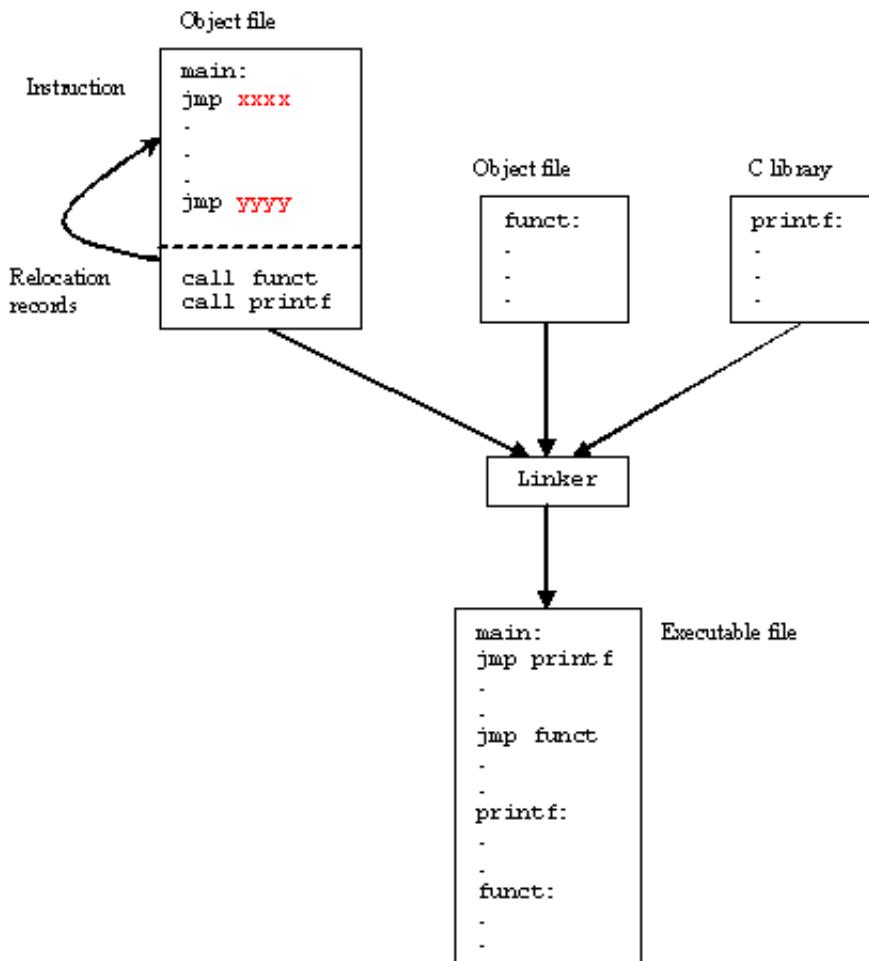


Figure 3.18: ตัวอย่างการลิงค์ (Link) หรือรวมไฟล์ Object และ Library เข้าด้วยกันเป็นไฟล์โปรแกรม หรือแอพพลิเคชัน ที่มา: <https://sites.google.com/site/kmrvikash/home/tutorials/c-tutorials/compiler-assembler-linker-and-loader-a-brief-story>

- บิตที่ 0-3 ความยาว 4 บิต คือตำแหน่งของ Rm มีค่าเท่ากับ 1_{10} หรือ 0001₂ 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			
1110 ₂	00 ₂	0	2	0	7	5	0	0	0	1			
cond	op	I	cmd	S	Rn	Rd	shamt5	sh			SUB R5, R7, R1		
31:28	27:26	25:20	19:16	15:12	11:0							LDR R9, [R4, #16]	
1110 ₂	01 ₂	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] จะถูกแปลงตามบิตต่างๆ

- บิตที่ 28-31 ความยาว 4 บิต คือตำแหน่งของ cond (Condition) คือเงื่อนไขของการทำงานคำสั่งปัจจุบันนี้ ซึ่ง 1110_2 หมายถึง ไม่มีเงื่อนไข รายละเอียดเพิ่มเติมในหัวข้อที่ [4.7](#)
- บิตที่ 26-27 ความยาว 2 บิต คือตำแหน่งของ OpCode (Operation Code) ของคำสั่ง LDR มีค่าเท่ากับ 01_2
- บิตที่ 20-25 ความยาว 6 บิต คือตำแหน่งของ IPUWL ของคำสั่ง LDR มีค่าเท่ากับ 25_{10}
- บิตที่ 15-19 ความยาว 4 บิต คือตำแหน่งของ Rn มีค่าเท่ากับ 4_{10} หรือ 0100_2 หมายถึง R4
- บิตที่ 12-15 ความยาว 4 บิต คือตำแหน่งของ Rd มีค่าเท่ากับ 9_{10} หรือ 1001_2 หมายถึง R9
- บิตที่ 0-11 ความยาว 12 บิต คือตำแหน่งของ imm12 มีค่าเท่ากับ 16_{10} หรือ $0000\ 0001\ 0000_2$ หมายถึง #16

ตัวเลขในตำแหน่งบิตต่างๆ เหล่านี้ ฮาร์ดแวร์จะตีความ (Decode) ได้ว่า คำสั่งที่อ่านเข้ามาเป็นคำสั่งอะไร ต้องการใช้รีจิสเตอร์ตัวไหน กำหนดค่าคงที่ (Immediate) เป็นเลขฐานสิบหรือไบนารี เป็นบวกหรือลบ ซึ่งรายละเอียดในการออกแบบหรือฮาร์ดแวร์ได้ในวิชาสถาปัตยกรรมคอมพิวเตอร์ (Computer Architecture) โดยผู้อ่านสามารถศึกษาเพิ่มเติมได้ที่ https://en.wikipedia.org/wiki/Computer_architecture ส่วนการออกแบบพัฒนาวงจรนั้นสามารถทำได้โดยใช้ภาษา Hardware Description Language (HDL) เช่น

- ภาษา VHDL (Very High Speed Integrated Circuit HDL)
<https://en.wikipedia.org/wiki/VHDL> และ
- ภาษา Verilog HDL
<https://en.wikipedia.org/wiki/Verilog> บนอุปกรณ์
- FPGA (Field Programmable Gate Array)
https://en.wikipedia.org/wiki/Field-programmable_gate_array

ภาษา Assembly เวอร์ชัน 32 บิตของ ARM

วัตถุประสงค์

- เข้าใจโครงสร้างด้าน硬件แวร์ภายในของชีพียุ ARM Cortex A53
- เข้าใจภาษา Assembly เวอร์ชัน 32 บิตของ ARM
- เข้าใจรูปแบบคำสั่งภาษา Assembly ประเภทสำคัญของ ARM
- รับรู้วิวัฒนาการภาษา Assembly ของ ARM

4.1 โครงสร้างของชีพียุ ARM Cortex A53

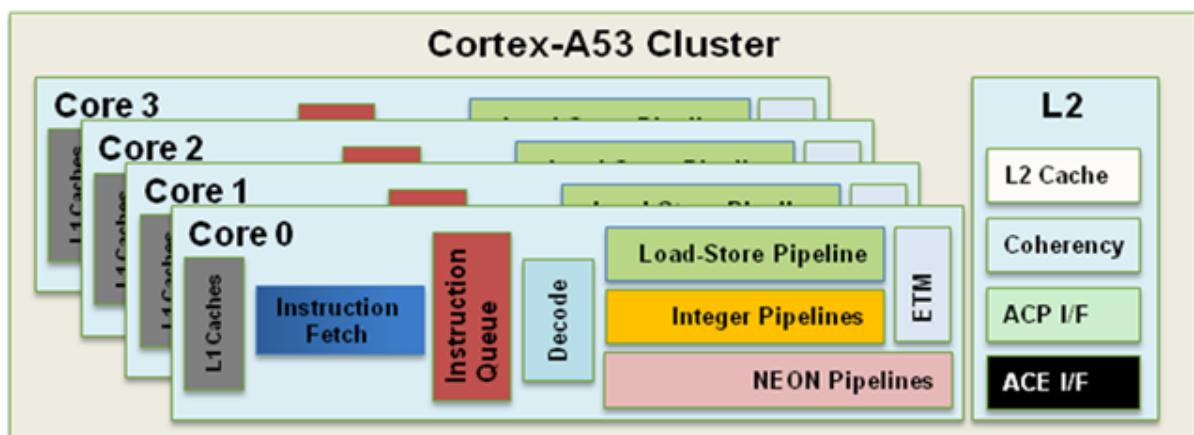


Figure 4.1: โครงสร้างของ ARM Cortex A53 จำนวน 4 คอร์ แต่ละคอร์ประกอบด้วย CPU core เพื่อประมวลผลตัวเลขจำนวนเต็มขนาด 32 และ 64 บิต, Floating Point Unit, NEON SIMD engine, แคช L1 I-Cache และ D-Cache ที่มา: https://www.tomshardware.com/reviews/snapdragon-810-benchmarks_4053-2.html

ในบทที่ 3 ผู้อ่านได้เรียนรู้โครงสร้างของชิป BCM2837 ซึ่งเป็นศูนย์กลางของบอร์ด Pi3 โดยมี CPU เป็นองค์ประกอบหลักภายในชิป ประกอบด้วย ARM Cortex A53 จำนวน 4 คอร์ ตามรูปที่ 3.3 ในบทที่ 3 โดยรูปที่ 4.1 แสดงรายละเอียดของ แต่ละคอร์ประกอบด้วยวงจรจากซ้ายไปขวา ARM Cortex A53 มี

โครงสร้างที่ไม่ซับซ้อน มีการทำงานแบบไปปีลน์ รายละเอียดเพิ่มเติม: https://en.wikichip.org/wiki/arm_holdings/microarchitectures/cortex-a53

- แคชคำสั่งลำดับที่ 1 หรือแคช L1 Instruction Cache หรือ L1 I-Cache จะเก็บคำสั่งล่าสุด ซึ่งคำสั่งเหล่านี้มาจากเซกเมนต์ Text ของหน่วยความจำเมモรี่ การทำงานเบื้องต้นของแคชคำสั่งคล้ายกับหน่วยความจำขนาดเล็กเพื่อ เก็บคำสั่งล่าสุดที่อ่านมาเพื่อมาถอดรหัส (Decode) และมีขนาดเล็กประมาณ 16-64 กิโลไบต์ หรือ เท่ากับ 4,096-16,536 คำสั่ง (4 ไบต์ต่อคำสั่ง)
- โมดูลหรือวงจรเฟลช (Fetch) ทำหน้าที่ส่งแอดเดรสของคำสั่งที่ต้องการไปยังแคชคำสั่งลำดับที่ 1 แล้วรอรับคำสั่งมาพักเก็บในคิว (Instruction Queue)
- คิวคำสั่งทำหน้าที่เป็นคิวเก็บพักคำสั่งจำนวน 8-16 คำสั่ง เพื่อส่งต่อให้วงจรถอดรหัส
- วงจรถอดรหัส (Decode) ทำหน้าที่แปลความหมายของเลขฐานสองที่ส่งมาจากคิว คล้ายกับรูปที่ 3.19 เพื่อส่งต่อไปยังวงจรไปปีลน์ที่เหมาะสมสมต่อไป
- วงจรไปปีลน์ (Pipeline) มี 3 ชนิด ประกอบด้วย
 - วงจรไปปีลน์สำหรับการอ่าน/เขียนข้อมูลกับหน่วยความจำหลัก (Load-Store Pipeline) เพื่ออ่าน/เขียนข้อมูลกับหน่วยความจำผ่านทาง แคชข้อมูล (Data Cache หรือ D-Cache) ลำดับที่ 1 หรือ L1 D-Cache และแคชลำดับที่ 2 (Level 2)
 - แคชข้อมูล หรือ Data Cache หรือ L1 D-Cache มีการทำงานเบื้องต้น คล้ายกับบีฟเฟอร์ขนาดเล็ก เพื่อเก็บข้อมูลล่าสุดที่อ่านหรือเขียนบ่อยๆ ซึ่งข้อมูลเหล่านี้มาจากเซกเมนต์ต่างๆ ที่ไม่ใช่เซกเมนต์ Text การทำงานเบื้องต้นของแคชข้อมูลคล้ายกับ แคชคำสั่ง ต่างกันตรงที่ซีพียูสามารถเปลี่ยนแปลงค่าของข้อมูลในแคชข้อมูลได้
 - วงจรไปปีลน์สำหรับการประมวลผลเลขจำนวนเต็ม (Integer Pipeline) ขนาด 32 และ 64 บิต การประมวลผล คณิตศาสตร์และตรรกศาสตร์ ข้อมูลแบบสเกลาร์ (Scalar) หรือ ข้อมูลเชิงเดี่ยว
 - วงจรไปปีลน์สำหรับการประมวลผลเลขจำนวนจริงชนิดศนนิยมโดยตัว (NEON Pipeline) รองรับข้อมูลเชิงเดี่ยวและแบบเวคเตอร์ (Vector) ข้อมูลแบบเวคเตอร์ ได้แก่ ข้อมูลตำแหน่ง ในเกม 3 มิติ ข้อมูลภาพและเสียง โดยจะสามารถใช้คำสั่งเดี่ยวประมวลผลข้อมูลพร้อมๆ กันหลายตัว เรียกว่า SIMD (Single Instruction Multiple Data)
- แคชลำดับที่ 2 หรือ L2 โดยทั้งสี่คอร์จะใช้งานแคช L2 ด้วยกัน หากคอร์ใดหาคำสั่งหรือข้อมูลในแคชลำดับที่ 1 (L1) ของตนเองไม่เจอ จะต้องค้นหาคำสั่งหรือข้อมูลในแคชลำดับที่ 2 ต่อไป หากไม่เจออีกจึงค้นต่อในหน่วยความจำหลัก บทที่ 5 จะอธิบายการทำงานของแคชทั้งสองลำดับ

ในเชิงโครงสร้าง ซีพียูจำนวน 1 คอร์ของ ARM Cortex A53 ในรูปที่ 4.1 ประกอบด้วย รีจิสเตอร์ R0-R15 แคชต่างๆ และหน่วยความจำหลัก เพื่อให้สามารถประมวลผลตัวเลขจำนวนเต็มขนาด 32 บิต โดยอาศัยวงจร ALU (Arithmetic Logic Unit) ตามที่ได้อธิบายไปแล้วในบทที่ 2

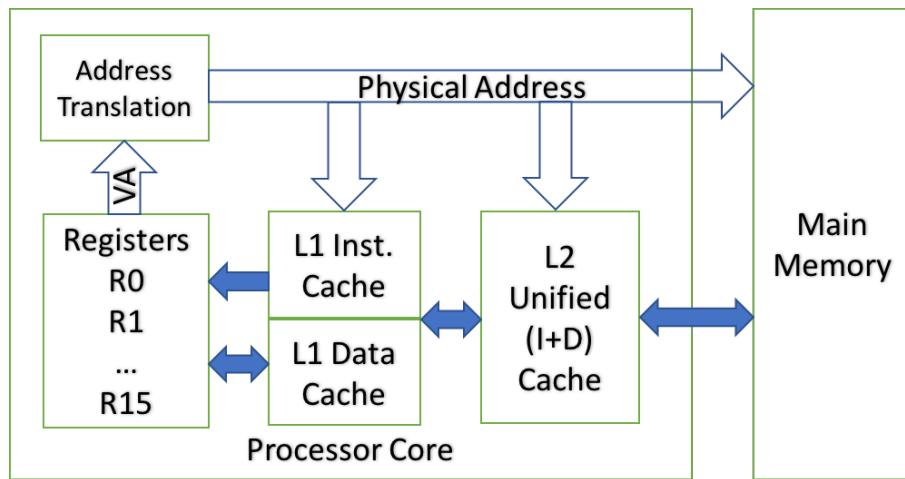


Figure 4.2: โครงสร้างของชีพิญจำนวน 1 คอร์สำหรับประมวลผลตัวเลขจำนวนเต็มขนาด 32 บิต โดยมีรีจิสเตอร์ R0-R15 แคชต่างๆ และหน่วยความจำหลัก, (VA: Virtual Address)

ตัวแปรชื่อต่างๆ นั้นถูกกำหนดพื้นที่ไว้ใน Data Segment ของหน่วยความจำเมื่อเริ่ม แล้วหน่วยความจำหลัก แต่การประมวลผลค่าของตัวแปรเหล่านี้ ชีพิญจะต้องโหลดค่ามาพักเก็บในรีจิสเตอร์ก่อน เมื่อประมวลผลแล้วเสร็จ จึงนำค่าเก็บไว้ในรีจิสเตอร์ เพื่อพักเก็บค่าหรือรอประมวลผลต่อ ด้วยเหตุผลด้านประสิทธิภาพ ซึ่งจะอธิบายในบทที่ 5

ในด้านของความจุ หน่วยความจำหลัก มีความจุขนาดใหญ่แต่ใช้เวลานานกว่า ในขณะที่แคชมีความจุรองลงมา และรีจิสเตอร์มีความจุน้อยที่สุดแต่มีความเร็วสูงสุด การประมวลผลทางคณิตศาสตร์และตรรกศาสตร์ข้อมูลเหล่านี้ จะต้องอาศัยขั้นการอ่าน (Load) ข้อมูลจากแคชหรือหน่วยความจำมาพักเก็บในรีจิสเตอร์ก่อน แล้วจึงนำค่าในรีจิสเตอร์ไปประมวลผลแล้วพักเก็บเพื่อประมวลผลต่อไป เมื่อคำนวณแล้วเสร็จจึงทำการเขียน (Store) ค่าไปเก็บในแคชและหน่วยความจำ ดังนั้น เราจึงเรียกการทำงานลักษณะนี้ว่า สถาปัตยกรรมอ่าน/เขียน (Load/Store Architecture)

สถาปัตยกรรมอ่าน/เขียนของ ARM แตกต่างจากการทำงานสถาปัตยกรรมชนิดอื่นๆ เช่น สถาปัตยกรรมสแต็ค (Stack Architecture) ในภาษา Java ไบท์โค้ด (Java Byte Code) สถาปัตยกรรมเมม莫รี/เมโมรี (Memory/Memory Architecture) ในภาษาแอสเซมบลีของ Intel 80x86 หรือเรียกย่อๆ ว่า x86 ซึ่งใช้การอ่านค่าจากหน่วยความจำเพื่อประมวลผล แล้วจึงเขียนผลลัพธ์กลับไปยังหน่วยความจำ

รายละเอียดเพิ่มเติมสามารถศึกษาเพิ่มเติมได้ในหัวข้อที่ 5.5 การแปลง Virtual Address เป็น Physical Address เพื่อนำไปอ้างถึงข้อมูลที่อยู่ในหน่วยความจำหลัก คือ Operation พื้นฐานสำหรับคำสั่งนิด LOAD และคำสั่ง STORE

4.2 สถาปัตยกรรมชุดคำสั่ง (Instruction Set Architecture)

สถาปัตยกรรมชุดคำสั่งของ ARM เวอร์ชัน 32 บิต ประกอบด้วย

- ชนิดและขนาดของคำสั่ง

- คำสั่งประมวลผลตัวแปร

- คำสั่งการถ่ายโอนข้อมูลในตัวแปรกับรีจิสเตอร์
 - คำสั่งประมวลผลข้อมูลคณิตศาสตร์และตรรกศาสตร์
 - คำสั่งการควบคุมการทำงาน
- รีจิสเตอร์สำหรับพักข้อมูล จำนวน 16 ตัว เพื่อรอการประมวลผลด้วยคำสั่งทางคณิตศาสตร์ ตรรกศาสตร์ และอื่นๆ รีจิสเตอร์เหล่านี้ ประกอบด้วย R0, R1, ..., R15 ทุกตัวมีความยาว 32 บิต โดย
 - R15 เรียกว่า Program Counter (PC) คือ รีจิสเตอร์สำหรับเก็บแอดเดรสของคำสั่งที่ต้องการจะ執行มา
 - R14 เรียกว่า Link Register (LR) คือ รีจิสเตอร์สำหรับเก็บแอดเดรสของคำสั่งที่ต้องการจะเรียกกลับ
 - R13 เรียกว่า Stack Pointer (SP) คือ รีจิสเตอร์สำหรับเก็บแอดเดรสบนสุดของสแต็ค
 - R12-R3 ใช้งานทั่วไป
 - R1-R0 นิยมใช้สำหรับการรับ/ส่งข้อมูลระหว่างฟังค์ชัน
 - ชนิดและขนาดของข้อมูล พื้นที่และขนาดของหน่วยความจำ (Memory Space) เทียบเคียงตารางที่ 2.1 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, int เป็นต้น
 - Doubleword 64 บิต เหมาะสำหรับตัวแปรชนิดจำนวนเต็ม เช่น unsigned long long เป็นต้น

4.3 ตัวอย่างคำสั่งภาษาเครื่องในหน่วยความจำ

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

- คอลัมน์ซ้าย Address ระบุหมายเลขไปที่ที่เริ่มต้นของ Opcode ทางขวา
- คอลัมน์กลาง Opcode คือ คำสั่งภาษาเครื่องในรูปของเลขฐานสิบหก ความยาว 32 บิต หรือ 4 byte
- คอลัมน์ขวา (Disassembly) คือ การแปลคำสั่งภาษาเครื่องให้กลับเป็นภาษาแอสเซมบลี ARM

ยกตัวอย่างเช่น หน่วยความจำตำแหน่งที่ 0x0000_C140 จำนวน 4 byte บรรจุ Opcode ค่า 0x08BD_8008 ซึ่งตรงกับคำสั่ง POPEQ r3,pc ชีดล่าง '_' ที่ผู้เขียนใส่เพิ่มทำให้ผู้อ่านสามารถอ่านหมายเลขที่เรียงติดกันได้ง่ายขึ้น

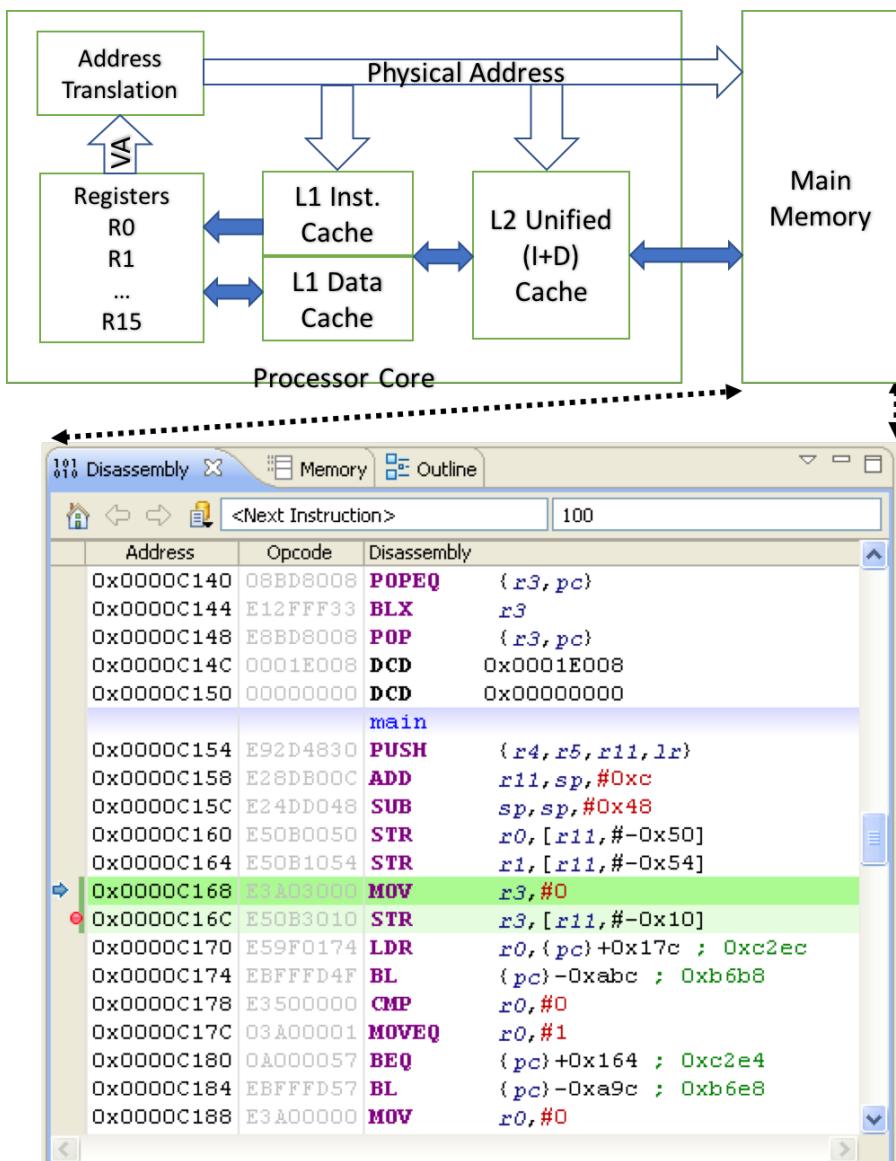


Figure 4.3: คำสั่งของโปรแกรมที่ถูกอ่าน (Load) เข้าสู่หน่วยความจำและแสดงในรูปของภาษาแอสเซมบลี บนโปรแกรมซิมูเลเตอร์ (Simulator) ที่มา: http://infocenter.arm.com/help/topic/com.arm.doc.dui0446c/graphics/disassembly_view.png

ตำแหน่งถัดไปคือ 0x0000_C144 จำนวน 4 ไบต์บรรจุ Opcode ค่า 0xE12F_FF33 ซึ่งตรงกับคำสั่ง BLX r3 คำสั่ง POPEQ หมายถึง คำสั่ง POP เมื่อผลการคำนวนเปรียบเทียบก่อนหน้ามีค่าเท่ากัน (EQ: EQUAL) การ POP r3, pc คือ การอ่านข้อมูลจากหน่วยความจำที่ pc เก็บ ไปบรรจุใน r3 เมื่อประมวลผลคำสั่งที่แอดдрес 0x0000_C144 สำเร็จแล้ว pc จะเพิ่มขึ้น 4 ไบต์เป็น 0x0000_C148 เพื่อนำ Opcode ค่า 0xE12F_FF33 ซึ่งตรงกับคำสั่ง BLX r3 ต่อไป

pc คือ R15 รีจิสเตอร์สำหรับเก็บแอดdressในหน่วยความจำของคำสั่งปัจจุบัน โดยสังเกตได้จากลูกศรสินะเงิน เพื่อให้ซึ่งสามารถอ่านคำสั่งนั้น ไปประมวลผล ซึ่งในรูป คือ หมายเลข 0x0000_C168 บรรจุ Opcode ค่า 0xE3A0_3000 ซึ่งตรงกับคำสั่ง MOV r3, #0

เมื่อประมวลผลคำสั่งที่แอดdress 0x0000_C168 สำเร็จแล้ว pc จะเพิ่มขึ้น 4 ไบต์เป็น 0x0000_C16C เพื่อนำ Opcode ค่า 0xE50B_3010 ซึ่งตรงกับคำสั่ง STR r3, [r11, #-0x10] ต่อไป แต่ในรูป ผู้ใช้ได้ใส่

Break Point โดยสังเกตได้จากวงกลมสีแดงทางซ้ายสุด Break Point จะทำให้โปรแกรมหยุดทำงานชั่วขณะ เพื่อให้มีความสามารถศึกษาความเป็นของคำสั่งที่แล้วได้

4.4 การประกาศและตั้งค่าตัวแปรในหน่วยความจำหลัก

ผู้อ่านต้องระลึกไว้เสมอว่าตัวแปรต่างๆ อยู่ในหน่วยความจำหลักเสมอ (Variables are always in main memory.) รีจิสเตอร์เป็นแค่ที่เก็บพักข้อมูลชั่วคราวสำหรับการประมวลผล เมื่อคำนวณเสร็จแล้ว ข้อมูลในรีจิสเตอร์จะถูกถ่ายโอนกลับไปยังหน่วยความจำ

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

Table 4.1: ตัวอย่างโปรแกรมภาษาแอสเซมบลีเพื่อสร้างตัวแปรขนาด 32 บิต จำนวน 2 ตัวแปร

#	Label	Code	Comment
1		.data	;Variable definition
2		.balign 4	
3	wordvar1:	.word 7	
4		.balign 4	
5	wordvar2:	.word 3	

ตารางที่ 4.1 ตัวอย่างโปรแกรมภาษาแอสเซมบลีเพื่อสร้างตัวแปรขนาด 32 บิต จำนวน 2 ตัวแปร บรรทัดที่ 1 คำสั่ง .data คือ การบ่งบอกถึงการเริ่มต้นประกาศตัวแปร คำสั่ง .balign คือการสั่งให้การจองข้อมูลในหน่วยความจำ ตรงกับพื้นที่ 4 ไบต์ เพื่อให้บริหารจัดการได้ง่ายขึ้น ตัวแปรชนิดจำนวนเต็ม ความยาว 1 Word ต้องการพื้นที่ 4 ไบต์ 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

คำสั่งโอนถ่ายข้อมูลระหว่างหน่วยความจำและรีจิสเตอร์ (Memory-Register Transfer Instructions) และโหมดการอ้างแอดเดรสชนิดต่างๆ (Addressing Mode)

รูปแบบ	ความหมาย
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
MOV Rd, #Imm	Rd = #Imm

Table 4.2: ตัวอย่างโปรแกรมภาษาแอสเซมบลีเพื่ออ่านค่าตัวแปรจากหน่วยความจำ โดยการอ่านตำแหน่งของตัวแปร

#	Label	Code	Comment
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 <= var1 - var2

ตัวอย่างการเขียนโปรแกรม

x = (a + b) - c;

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

#	Label	Code	Comment
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	; a+b
6		LDR R4, =c	; get address of variable c
7		LDR R2, [R4]	; get value of variable c
8		SUB R3, R2, R3	; x = (a+b)-c
9		LDR R4, =x	; get address of variable x
10		STR R3, [R4]	; store value of variable x

4.6 คำสั่งประมวลผลข้อมูลในรีจิสเตอร์ (Register Data Processing Instructions)

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

- คำสั่งทางคณิตศาสตร์
- คำสั่งเลื่อนบิต
- คำสั่งทางคณิตศาสตร์และเลื่อนบิต
- คำสั่งทางตรรกศาสตร์

4.6.1 คำสั่งทางคณิตศาสตร์

คำสั่งสำหรับเลขจำนวนเต็มนิดไม่มีเครื่องหมาย บวก ลบ คูณ กระทำโดยวงจรอาร์ดแวร์ เรียกว่า ALU (Arithmetic and Logic Unit) ซึ่งประกอบด้วยวงจรบวก/ลบเลข คูณเลข ตามหลักการที่ได้กล่าวมาในหัวข้อที่ [2.3](#)

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	$Rd = Rm - Rn$ (Resverse Subtract)
RSB Rd, Rn, #Imm	$Rd = \#Imm - Rn$ (Resverse 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)

การตรวจสอบโอเวอร์โฟลว์ เครื่องหมาย ศูนย์ และตัวทดบิทสุดท้าย ผู้อ่านสามารถศึกษารายละเอียดของการตรวจจับโอเวอร์โฟลว์ได้ในหัวข้อที่ [2.3.1](#) และ [2.3.2](#) Status Register จะเปลี่ยนแปลงตามผลลัพธ์ที่ ALU คำนวณตามชนิดของข้อมูลและ Opcode 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 (oVerflow) = 1 ใช้ตรวจสอบว่าผลการคำนวณเกิดโอเวอร์โฟลว์

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

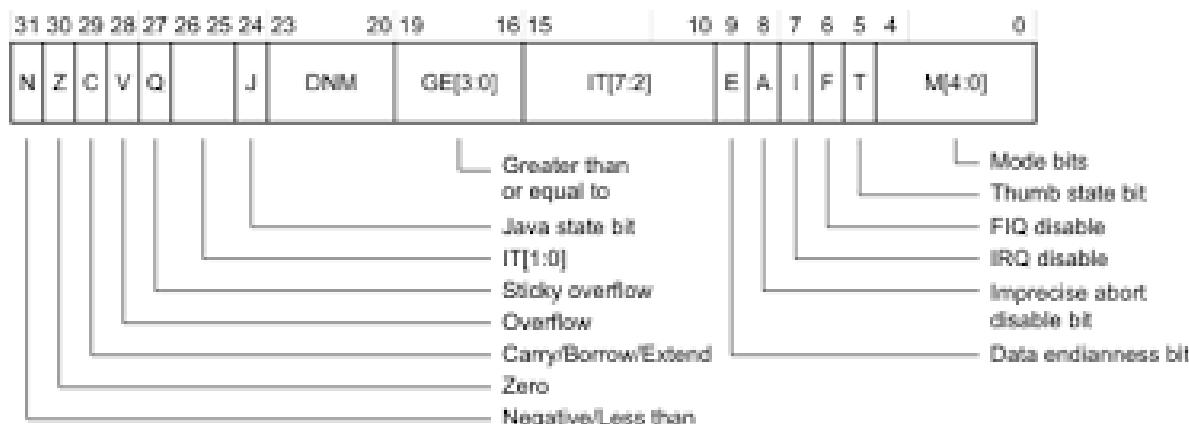


Figure 4.4: รีจิสเตอร์สำหรับเก็บสถานะของซีพียู ณ ปัจจุบัน (Current Program Status Register: CPSR) <http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0344i/Beibijfb.html>

4.6.2 คำสั่งเลื่อนบิตข้อมูล

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)

4.6.3 คำสั่งคอมมิตรัตเตอร์และเลื่อนบิต

การซิฟท์ข้อมูลภายในคำสั่งพื้นฐาน

รูปแบบ	ความหมาย
ADD Rd, Rn, Rm LSL #shmt	Rd = Rn + (Rm « #shmt)
ADD Rd, Rn, Rm LSR #shmt	Rd = Rn + (Rm « #shmt)
ADD Rd, Rn, Rm ASR #shmt	Rd = Rn + (Rm » #shmt) (Signed)

สามารถใช้งานคำสั่ง MOV, SUB และ RSB ร่วมกับ การซิฟต์ภายใน
ตัวอย่างคำสั่งการบวกค่าในรีจิสเตอร์ที่ได้จากการซิฟท์ไปทางซ้ายจำนวน 2 บิต

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

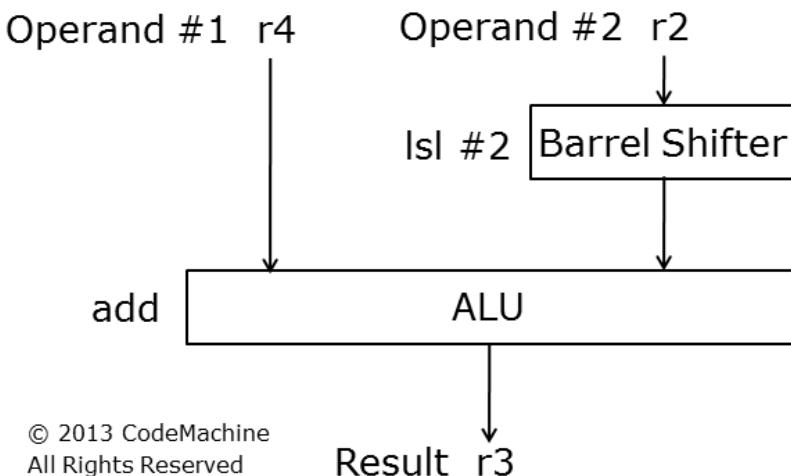


Figure 4.5: ตัวอย่างคำสั่งการบวกค่าในรีจิสเตอร์ที่ได้จากการซิฟท์เพลาท์ชัยจำนวน 2 บิต

4.6.4 คำสั่งทางตรรกศาสตร์

คำสั่งสำหรับเลขจำนวนเต็มชนิดไม่มีเครื่องหมาย ได้แก่ กระบวนการ AND OR NOT ประมวลผลพร้อมกันทุกบิต โดยบิตข้อมูลตัวตั้งจะกระทำกับบิตที่ตรงกันของตัวกระทำเท่านั้น

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 จะตรงกับคำว่า Branch เพื่อย้ายการทำงานไปยังคำสั่งเป้าหมาย ซึ่งถูกกำหนดโดย Label ตามรูปที่ 3.3 ตรงกับคำสั่ง Jump ในภาษาแอลกอริทึม หรือ การกระโดด go-to ในภาษา C/C++

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

ผลลัพธ์ที่ได้จากการรันคำสั่ง CMP คือ ค่าในรีจิสเตอร์ CPSR โดยเฉพาะบิต NZCV ในหัวข้อที่ ?? จะเกิดการเปลี่ยนแปลง โปรแกรมเมอร์สามารถตรวจสอบเงื่อนไขต่างๆ เหล่านี้ได้จาก บิต NZCV ดังนี้ สำหรับการเปรียบเทียบของเลขจำนวนเต็ม

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 เกิดโอเวอร์ฟล์ว
8. VC (Overflow Clear): ไม่เกิด Overflow คือการตรวจสอบว่า $V=0$ หรือไม่ นั่นคือ ไม่เกิดโอเวอร์ฟล์ว
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 ผลการลงมากกว่าหรือเท่ากับ 0
12. LT (Less Than): Negative != Overflow คือการตรวจสอบว่า $N!=V$ หรือไม่ นั่นคือ ผลการลงน้อยกว่า 0
13. GT (Greater Than): !Zero && Negative = Overflow คือการตรวจสอบว่า $Z=0$ และ $N=V$ หรือไม่ นั่นคือ ผลการลงมากกว่า 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 ซีพียูจะปฏิบัติตามประโภคคำสั่งที่โปรแกรมเมอร์ต้องการ หากเท็จ ซีพียูจะทำคำสั่งอื่นๆ ต่อไป
ตัวอย่างการเขียนโปรแกรม ประโภค IF ในภาษา C/C++

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

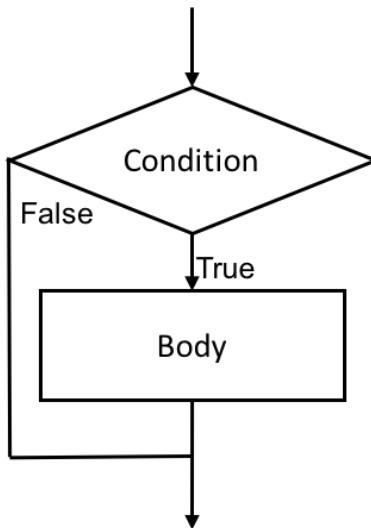


Figure 4.6: โพลว์ชาร์ตการทำงานของโครงสร้างการเขียนโปรแกรม IF

จากประโภค IF นี้ คอมไපิลเลอร์สามารถแปลเป็นชุดคำสั่งภาษา Assembly ของ ARM ได้ในตารางที่ 4.4 ผู้อ่านสามารถทำความเข้าใจการทำงานได้จากคำอธิบายภายใต้คอลัมน์ Comment และคำอธิบายเป็นภาษาไทยตามหมายเลขอรรถด์ 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

Table 4.4: ตัวอย่างโปรแกรมตามประโยคเงื่อนไข if

#	Label	Code	Comment
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 the condition is false
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

6. คือ การโหลดตำแหน่งของตัวแปร c ไปบรรจุใน R4
7. คือ การโหลดค่าของตัวแปร c ไปบรรจุใน R4
8. คือ การเปรียบเทียบค่าของ R2 และ R3
9. คือ หากเงื่อนไข Less Than or Equal (LE) เป็นจริง ซึ่งจะกระโดดไปทำงานคำสั่งที่ขึ้นต้นด้วย exit หากไม่เป็นจริง ซึ่งจะทำงานประโยคที่ 10 ต่อไป
10. คือ การโหลดตำแหน่งของตัวแปร x ไปบรรจุใน R4
11. คือ การโหลดค่าของตัวแปร x ไปบรรจุใน R5
12. คือ การโหลดตำแหน่งของตัวแปร y ไปบรรจุใน R4
13. คือ การโหลดค่าของตัวแปร y ไปบรรจุใน R6
14. คือ การคำนวนค่า a + b ไปบรรจุใน R5
15. คือ การโหลดตำแหน่งของตัวแปร x ไปบรรจุใน R4
16. คือ การสโตร์ค่าของ R5 ไปบรรจุตำแหน่งของตัวแปร x
17. คือ คำสั่งที่ขึ้นต้นด้วย exit

4.7.2 การตัดสินใจ IF-ELSE

การตัดสินใจ IF-ELSE ขึ้นอยู่กับเงื่อนไขของประYoic IF ว่าผลลัพธ์ที่ได้จะเป็นจริงหรือเท็จ เมื่อจริง ชีพิญ จะปฏิบัติตามประYoic คำสั่งที่โปรแกรมเมอร์ต้องการ เมื่อแล้วเสร็จชีพิญจะกระโดดข้ามส่วน ELSE เพื่อ ประมวลผลต่อ หากเท็จ ชีพิญจะทำคำสั่งส่วนที่ ELSE กำหนด เมื่อแล้วเสร็จจะประมวลผลต่อ ตามโฟล์ ชาร์ตในรูปที่ 4.7

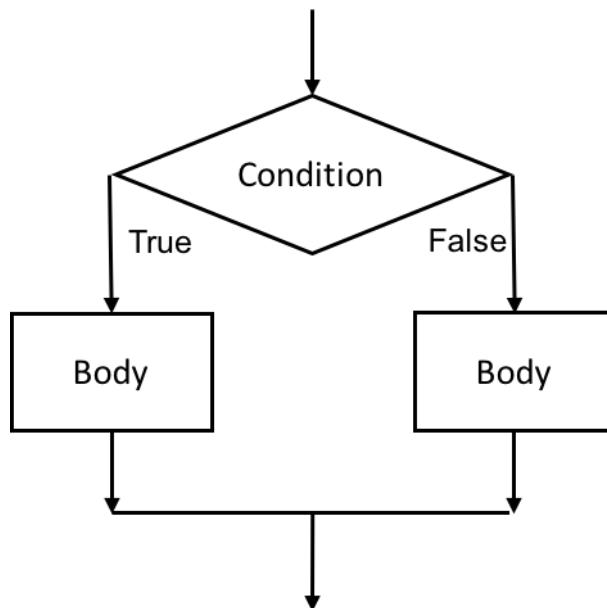


Figure 4.7: โฟล์ชาร์ตการทำงานของโครงสร้างการเขียนโปรแกรม IF-ELSE

ตัวอย่างการเขียนโปรแกรม ประYoic IF-ELSE ในภาษา C/C++

```

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

จากประYoic IF-ELSE นี้ คอมไพล์เลอร์สามารถแปลเป็นชุดคำสั่งภาษา Assembly ของ ARM ได้ใน ตารางที่ 4.5 ผู้อ่านสามารถทำความเข้าใจการทำงานได้จากคำอธิบายภายใต้คอลัมน์ Comment และคำ อธิบายเป็นภาษาไทยตามหมายเลขอรรถทัศ 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

Table 4.5: ตัวอย่างโปรแกรมตามประโยคเงื่อนไข if-else

#	Label	Code	Comment
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, R3	; compute (a+b)-c
9		BLE else	; jump to else if the condition is false
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	...	;

5. คือ การคำนวณค่า $a + b$ ไปบรรจุใน R3
6. คือ การโหลดตำแหน่งของตัวแปร c ไปบรรจุใน R4
7. คือ การโหลดค่าของตัวแปร c ไปบรรจุใน R4
8. คือ การเบรียบเทียบค่าของ R2 และ R3
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 การสโตรค่าของ R5 ไปบรรจุตำแหน่งของตัวแปร x
17. คือ การบังคับให้ซิพิยูกระโดยดีไปทำงานคำสั่งที่ขึ้นต้นด้วย exit
18. คือ การโหลดตำแหน่งของตัวแปร x ไปบรรจุใน R4
19. คือ การโหลดค่าของตัวแปร x ไปบรรจุใน R5
20. คือ การโหลดตำแหน่งของตัวแปร y ไปบรรจุใน R4
21. คือ การโหลดค่าของตัวแปร y ไปบรรจุใน R6
22. คือ การคำนวนค่า $x = x - y$ ไปบรรจุใน R5
23. คือ การโหลดตำแหน่งของตัวแปร x ไปบรรจุใน R4
24. คือ การสโตรค่าของ R5 ไปบรรจุตำแหน่งของตัวแปร x
25. คือ คำสั่งที่ขึ้นต้นด้วย exit

4.7.3 การวนรอบชนิด FOR

ในอดีต การพัฒนาโปรแกรมคอมพิวเตอร์มุ่งเน้นที่การคำนวนแก็บปัญหาทางคณิตศาสตร์ ยกตัวอย่างเช่น

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

สมการที่ 4.1 คือ การบวกเลขตั้งแต่ค่า $i=1$ จนถึง $i=10$ ในรูปแบบทางคณิตศาสตร์ อย่างง่าย โปรแกรมเมอร์สามารถใช้ภาษาใดภาษาหนึ่ง เช่น ภาษา C เขียนการบวกนี้ในรูปของประโยค 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 แล้วนำผลลัพธ์ที่ได้เก็บค่าไว้ ในตัวแปร x เพื่อที่จะวนกลับมาทำประโยคนี้อีก โดย i จะถูกเพิ่มค่าเป็น $i+1$ เช่นกัน ประโยค $x=x+1$ จะทำงานทั้งหมด 10 รอบตามเงื่อนไข $i <= 10$

เราเรียก i ว่า เป็นตัวแปรวนรอบ (Index) เนื่องจากเป็นตัวแปรที่ใช้นับเลขรอบ ประโยค $i=1$ เรียกว่า Index Initialization คือ การตั้งค่าเริ่มต้นให้กับตัวแปรวนรอบ ซึ่งเป็นองค์ประกอบหนึ่งของประโยค for ประโยค $x=x+1$ เรียกว่า Loop Body ตามโฟล์ชาร์ตในรูปที่ 4.8 ประโยคเงื่อนไข $i <= 10$ เรียกว่า

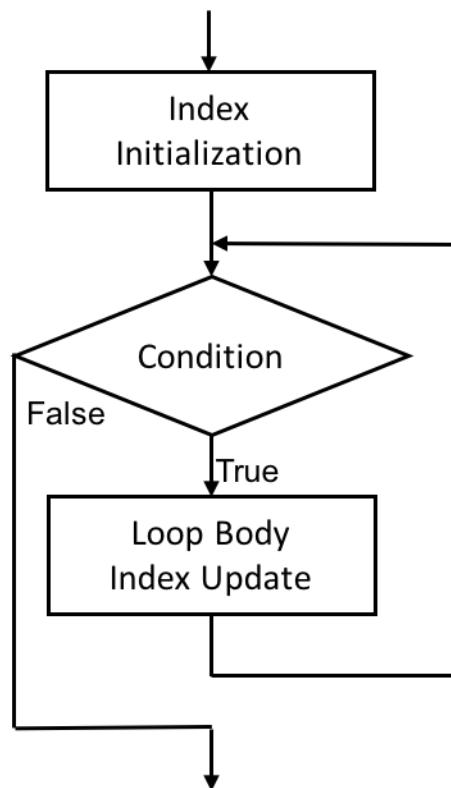


Figure 4.8: โฟล์ชาร์ตการทำงานของการวนรอบชนิด FOR

ว่า Condition เพื่อตรวจสอบว่าเป็นจริงหรือเท็จ เมื่อ i มีค่าเท่ากับ 1 ถึง 10 เงื่อนไขจะให้ผลลัพธ์เป็นจริง (True) เมื่อ i มีค่าเท่ากับ 11 ผลลัพธ์จะกลับเป็นเท็จ (False) และจะไม่เกิดการวนรอบ เพื่อทำงานประโยชน์ที่ต่อจากประโยชน์ for ประโยชน์ $i=i+1$ เรียกว่า Index Update คือ การเปลี่ยนแปลงค่าตัวแปรวนรอบให้สอดคล้องกับเงื่อนไข ตามที่กล่าวมาข้างต้น

Table 4.6: ตัวอย่างโปรแกรมตามประโยชน์วน For

#	Label	Code	Comment
1		...	; Initialize R0=0
2		...	; Initialize R1=1
3	for:	CMP R1, #10	;
4		BGT end	; if less 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 Comment และคำอธิบายเป็นภาษาไทยตามหมายเลขอรรถที่ด ดังนี้

1. คือ การตั้งค่าเริ่มต้นให้กับ $R0 \leq 0$
2. คือ การตั้งค่าเริ่มต้นให้กับ $R1 \leq 1$

3. คือ คำสั่งที่ขึ้นต้นด้วย for เพื่อทำการเปรียบเทียบ R1 กับค่าคงที่ 10_{10}
4. คือ หากเงื่อนไข Less Than or Equal (LE) เป็นจริง ซึ่งจะกระโดดไปทำงานคำสั่งที่ขึ้นต้นด้วย end หากไม่เป็นจริง ซึ่งจะทำงานโดยอิคที่ 5 ต่อไป
5. คือ การคำนวนค่า $a + b$ ไปบรรจุใน R3
6. คือ การโหลดคำแนะนำของตัวแปร c ไปบรรจุใน R4
7. คือ การโหลดค่าของตัวแปร c ไปบรรจุใน R4
8. คือ การเปรียบเทียบค่าของ R2 และ R3
9. คือ การบังคับให้ซึ่งจะกระโดดไปทำงานคำสั่งที่ขึ้นต้นด้วย end
10. คือ คำสั่งที่ขึ้นต้นด้วย end

โดยทั่วไป ตัวอย่างนี้เป็นการวนลูปชนิด For ในรูปที่ 4.8 การตั้งค่าเริ่มต้น (Initialization) ให้กับตัวแปรนลูป (Index) หลังจากนั้น โปรแกรมจะตรวจสอบเงื่อนไข Condition ของลูป หากเป็นจริง โปรแกรมจะทำงานตามคำสั่งจำนวนหนึ่ง ที่เรียกว่า Body เมื่อแล้วเสร็จ โปรแกรมจะเพิ่ม (Increment) หรือ ลด (Decrement) ตัวแปรลูป (Index) แล้วจึงกลับไปบรรทัดที่มีประโยชน์เงื่อนไขซ้ำแล้วซ้ำอีก จนเงื่อนไขเป็นเท็จ (False) ลูป FOR จะสิ้นสุดการทำงาน แล้วจึงย้ายการทำงานไปคำสั่งอื่นภายนอกลูป

4.7.4 การวนรอบ WHILE

การวนรอบ WHILE คล้ายกับการวนรอบ FOR คือ การวนรอบจะเกิดขึ้นอยู่กับเงื่อนไขของประโยชน์ WHILE ว่าผลลัพธ์ที่ได้จะเป็นจริงหรือเท็จ เมื่อจริง ซึ่งจะปฏิบัติตามประโยชน์คำสั่งที่โปรแกรมเมอร์ต้องการ แล้วกลับไปตรวจสอบเงื่อนไขอีกรอบ หากเท็จ ซึ่งจะทำการคำสั่งอื่นๆ ต่อไป ในรูปที่ 4.9

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

ตัวอย่างนี้ ทำงานเหมือนกับประโยชน์ FOR ทุกประการ ความแตกต่างคือ การตั้งค่าตัวแปรเริ่มต้น การตรวจสอบเงื่อนไขในวงเล็บของประโยชน์ WHILE และการ Update ตัวแปร i ภายใต้ลูป ดังนั้น ตัวอย่างการเขียนโปรแกรม ในตารางที่ 4.7 จึงมีความใกล้เคียงกันมากกับประโยชน์ FOR

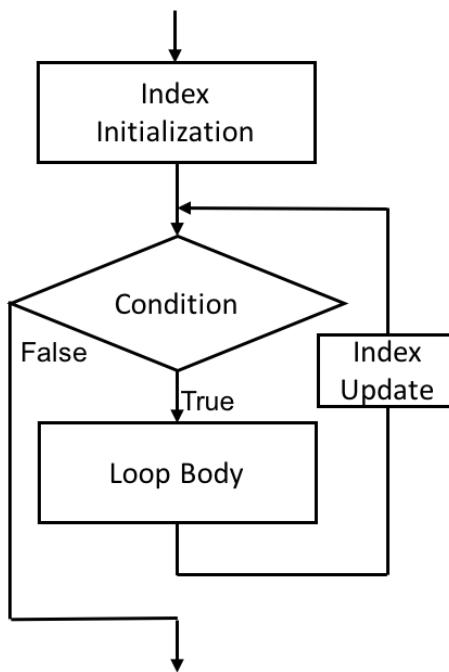


Figure 4.9: โฟล์ชาร์ตการทำงานของโครงสร้างการเขียนลูป While

Table 4.7: ตัวอย่างโปรแกรมตามประโยชน์ควบคุม While

#	Label	Code	Comment
1		...	; Initialize R0=0
2		...	; Initialize R1=1
3	while:	CMP R1, #10	;
4		BGT end	; if less 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 การวนรอบ DO-WHILE

การวนรอบอีกชนิดคือ การวนรอบ DO-WHILE มีโครงสร้างการทำงานตามรูปที่ 4.10 การวนรอบ DO-WHILE คล้ายกับการวนรอบ WHILE คือ ซีพียูจะปฏิบัติตามประโยชน์คำสั่งที่โปรแกรมเมอร์ต้องการอย่างน้อย 1 ครั้งก่อน หลังจากนั้น ซีพียูจะตรวจสอบเงื่อนไขของประโยชน์ WHILE ว่าผลลัพธ์ที่ได้จะเป็นจริงหรือเท็จ หากจริง ซีพียูจะปฏิบัติตามประโยชน์อีกรอบ และวิ่งตรวจสอบเงื่อนไข หากเท็จ ซีพียูจะทำคำสั่งอื่นๆ ต่อไป

```
i=1;
x=0;
do {
    x+=i; /* Body */
}
```

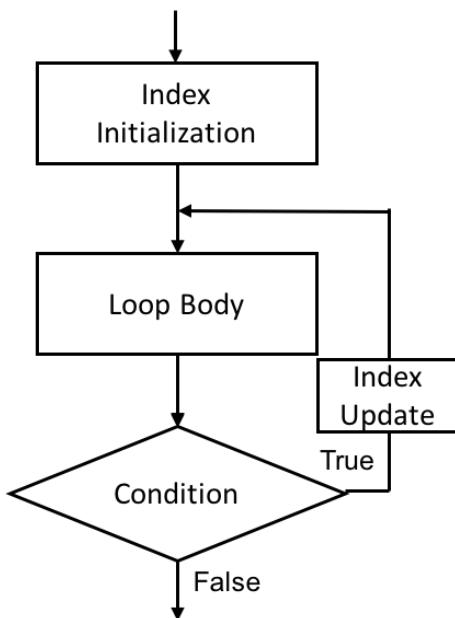


Figure 4.10: โฟลว์ชาร์ตการทำงานของโครงสร้างการเขียนลูป Do While

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

ตัวอย่างการเขียนโปรแกรม ในตารางที่ ??

Table 4.8: ตัวอย่างโปรแกรมตามประโยชน์ควบคุม Do-While

#	Label	Code	Comment
1		...	; Initialize R0 = 0
2		...	; Initialize R1 = 0
3	do:	SUB R0, R0, R1	; else subtract R1 from R2
4		ADD R1, R1, #1	;
5		CMP R1, #10	;
6		BLE do	; if less than goto end
7	end:	...	; End of do-while loop
8		...	;

4.7.6 การวนรอบชนิด FOR จำนวน 2 ชั้น

โฟลว์ชาร์ตนี้เป็นการวนลูป 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

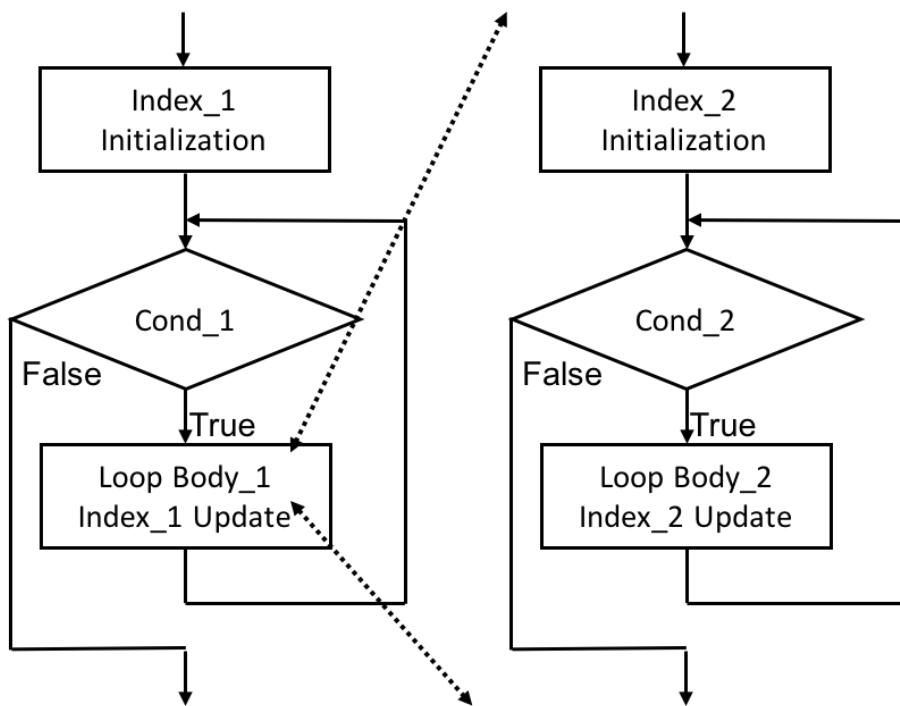


Figure 4.11: โฟลว์ชาร์ตการทำงานของการวนรอบ 2 ชั้น

4. หากเป็นจริง (True) โปรแกรมจะทำงานตาม Statements หรือคำสั่งจำนวนหนึ่ง
5. เมื่อแล้วเสร็จ โปรแกรมจะเพิ่ม (Increment) หรือ ลด (Decrement) ตัวนับลูปที่ 2 (Loop Counter 2)
6. จนเมื่อไก Condition_2 ของลูปชั้นใน เป็นเท็จ (False)
7. โปรแกรมจะเพิ่ม (Increment) หรือ ลด (Decrement)
8. ตัวนับลูปที่ 1 (Loop Counter 1) กลับไปบรรทัดที่ x
9. แต่หากเมื่อไกเป็นเท็จ (False) ลูปชั้นนอกจะสิ้นสุดการทำงาน

4.8 การเรียกใช้ฟังก์ชัน (Function Call)

ตัวอย่างการเขียนโปรแกรมโดยการเรียกใช้ฟังก์ชันในภาษา 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 กลไกนี้จะเกิดขึ้นเมื่อมีการทำงานเป็นโมดูล สามารถนำฟังค์ชันไปเผยแพร่เป็นฟังค์ชันมาตรฐานให้กับ ยกตัวอย่าง เช่น ฟังค์ชัน printf ในไฟล์ stdio.h ซึ่งย่อมาจากชื่อ Standard I/O ในรูปที่ 3.16 ฟังค์ชันมาตรฐานเหล่านี้ จะต้องนำมาลิงค์ตามหลักการในรูปที่ 3.18 ซึ่งเรียกว่า ไลบรารี (*.lib) หรือนามสกุลอื่นๆ ที่ใกล้เคียง ส่วนฟังค์ชันที่โปรแกรมเมอร์พัฒนาเอง จะอยู่ในรูปของไฟล์ Object (*.o) หรือนามสกุลอื่นๆ ที่ใกล้เคียง

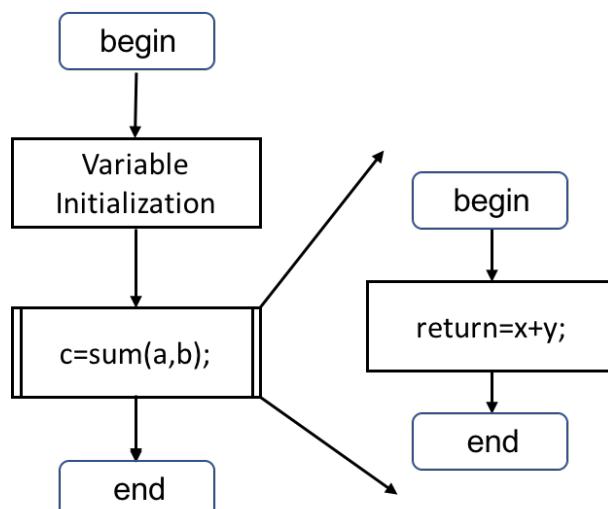


Figure 4.12: โฟล์ชาร์ตการทำงานของการเรียกใช้ฟังค์ชัน sum(x,y)

ภาษาแอสเซมบลีต่างๆ มีศักยภาพที่ใกล้เคียงกับภาษาสูง เช่น C/C++ ที่โปรแกรมเมอร์และคอมไพล์เลอร์สามารถสร้างฟังค์ชันของตนเองได้ คำสั่งภาษาแอสเซมบลีของ ARM ที่รองรับการเรียกใช้ฟังค์ชัน คือ BL function_name โดยคำว่า function_name คือ label ทำหน้าที่เป็นชื่อฟังค์ชันที่ต้องการเรียกใช้ BL ย่อมาจากคำว่า Branch and Link การเรียกกลับไปยังคำสั่งถัดไปเมื่อสิ้นสุดการเรียกใช้ฟังค์ชัน และคำสั่ง BX LR จะเป็นการเรียกกลับมาอ้างตำแหน่ง address ที่รีจิสเตอร์ LR เก็บค่าไว้ ค่านี้จะเก็บไว้ก่อนซีพียูจะทำงานคำสั่งแรกของฟังค์ชัน

การทำงานของโปรแกรมจะย้ายจากคำสั่ง BL myFunction ไปยังคำสั่งแรกสุดของ myFunction ชื่อยู่ 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

Table 4.9: ตัวอย่างโปรแกรมเรียกใช้ฟังค์ชันด้วยคำสั่ง BL และ BX

#	Label	Code	Comment
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	;
6		...	;
7	sum:	ADD R0, R0, R1	; function sum
8		BX LR	; Return the result in R0

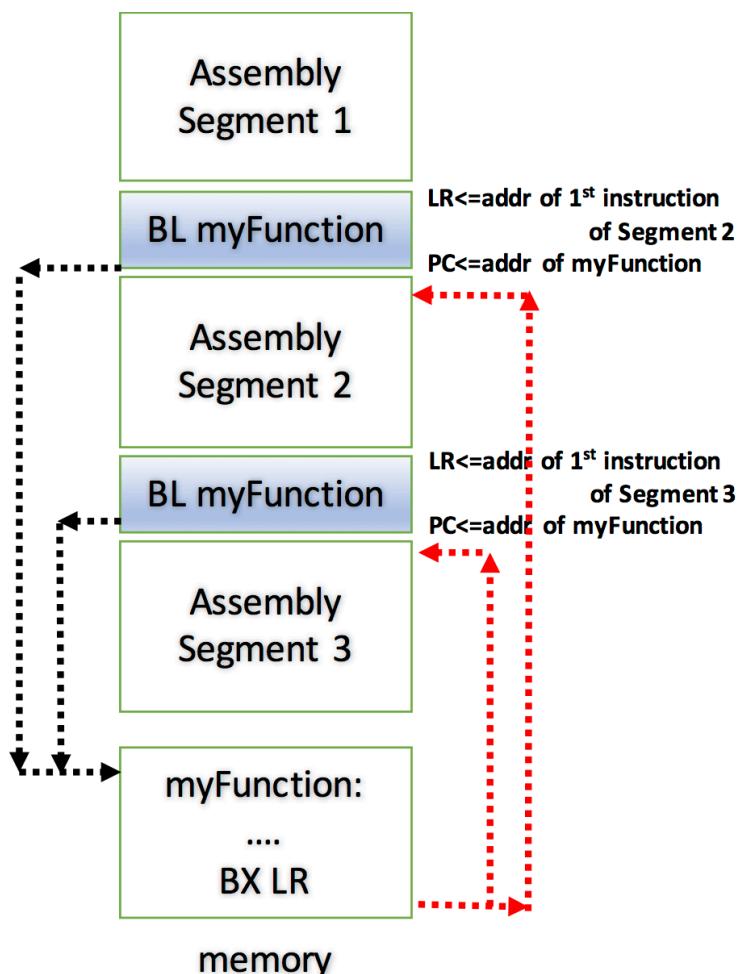


Figure 4.13: Flow Chart Function Call

4.9 อุปกรณ์และวิวัฒนาการของชุดคำสั่ง ARM

4.9.1 อุปกรณ์คอมพิวเตอร์ที่ใช้ชีพียู ARM

ตารางที่ 4.10 แสดงให้เห็นภาพรวมว่า ในปี 2010 ARM มีส่วนแบ่งการตลาดทั่วโลก 28% คิดเป็นชิปจำนวน 6,100 ล้านชิป จากทั้งหมด 22,000 ล้านชิป ในอุปกรณ์ทั้งหมด 19,000 ล้านตัว ตลาดของ

Table 4.10: ส่วนแบ่งการตลาดของบริษัท ARM ในปี ค.ศ.2010 ที่มา: <https://www.zdnet.com/article/arm-holdings-2015-plan-grab-pc-server-share/>

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

ARM แบ่งเป็นอุปกรณ์ที่เป็นโทรศัพท์เคลื่อนที่ (Mobile) และอื่นๆ (Non-Mobile) ตลาดโทรศัพท์เคลื่อนที่ ประกอบด้วย Smart Phone, Feature Phone, Low End Voice, Portable Media Players และ Mobile Apps โดยอุปกรณ์ที่ ARM มีส่วนแบ่งการตลาดสูงที่สุด (95%) คือ Low End Voice รองลงมาคือ Smart Phone และ Feature Phone ตามลำดับ โดย ARM คาดว่าตลาดของ Smart Phone จะเติบโตและได้รับความนิยมเพิ่มสูงขึ้นเรื่อยๆ รวมถึงตลาดของโมบายล์แอพ

ในส่วนของตลาด Non-Mobile ประกอบด้วย อุปกรณ์ที่หลากหลายกว่า โดยอุปกรณ์ที่ ARM มีส่วนแบ่งการตลาดสูงที่สุด (85%) คือ Hard Disk & Solid State Drives รองลงมา คือ Digital Camera และ Printer ตามลำดับ โดย ARM คาดว่าตลาดของ HDD & SSD จะเติบโตและได้รับความนิยมเพิ่มสูงขึ้นเรื่อยๆ โดยเฉพาะ SSD นอกจากนี้ ตลาดของ Digital TV & Set Top Box และ Microcontroller น่าจะมีการขยายตัวในปี 2015 เช่นกัน

4.9.2 วิวัฒนาการของชุดคำสั่ง ARM

ในอดีตที่ผ่านมา ARM ได้พัฒนาภาษาอักมาห์ลายเวอร์ชัน โดย ARMv5 และ ARMv6 คือ ภาษาแอสเซมบลีเวอร์ชัน 5 และ 6 ตามลำดับ ARMv7 คือ ภาษาแอสเซมบลีเวอร์ชัน 7 ซึ่งรองรับการทำงานของโอลิส 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

นอกเหนือจากภาษา Assembly เวอร์ชันต่างๆ ที่กล่าวมาข้างต้น จุดขายของ ARM ยังมีภาษา As-

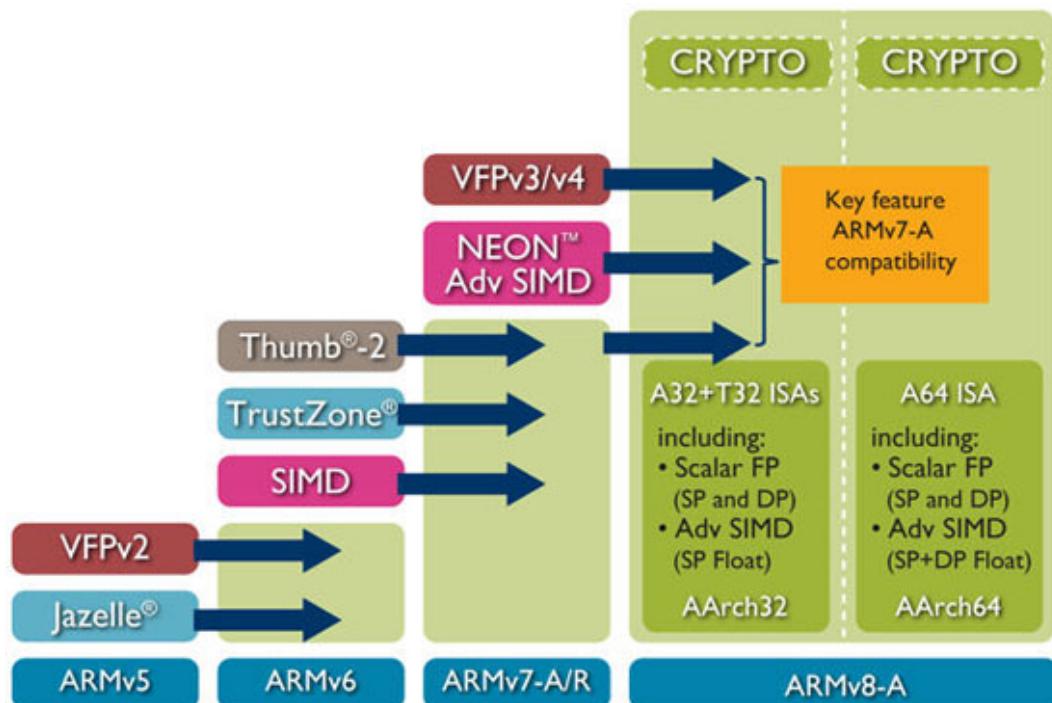


Figure 4.14: การรวมชุดคำสั่งภาษาและซีมบลีของ ARM ตั้งแต่เวอร์ชัน 5 ถึงเวอร์ชัน 8A โดยเวอร์ชันใหม่จะรวมเวอร์ชันเก่าเพื่อรองรับการทำงานของพัฒนาเมืองที่มา: <https://community.arm.com/processors/b/blog/posts/programmer-s-guide-for-armv8-a>

assembly อีกเวอร์ชัน ซึ่งว่า Thumb, Thumb-2, T32 นี้คือ ชุดคำสั่งภาษาและซีมบลีของ ARM ที่มีความยาว 16 และ 32 บิต ซึ่งนิยมใช้สำหรับอุปกรณ์ที่มีหน่วยความจำขนาดเล็ก รายละเอียดเพิ่มเติม https://en.wikipedia.org/wiki/ARM_architecture#Thumb

การคำนวนปกติจะกระทำกับข้อมูลเชิงเดี่ยวตามที่ได้อธิบายไปแล้ว ซึ่งไม่หมายความว่าข้อมูลมักติดมีเดีย ดังนั้น ARM จึงได้ออกแบบชุดคำสั่งภาษา Assembly ที่ประมวลผลข้อมูลหลายๆ ตัวพร้อมกัน เรียกว่า SIMD, Adv SIMD (Advanced SIMD) หรือ NEON เหมาะสำหรับข้อมูลภาพและเสียง รวมไปถึงเกม ซึ่งเป็นข้อมูลที่เรียงตัวกันจำนวนหนึ่ง เรียกว่า เวคเตอร์ (Vector) รายละเอียดเพิ่มเติม [https://en.wikipedia.org/wiki/ARM_architecture#Advanced SIMD_\(NEON\)](https://en.wikipedia.org/wiki/ARM_architecture#Advanced SIMD_(NEON))

จุดขายที่น่าสนใจของ ARM เรียกว่า Jazelle คือ การรองรับการประมวลผลคำสั่งและซีมบลีของภาษา Java เรียกว่า Java ByteCode รายละเอียดเพิ่มเติม <https://en.wikipedia.org/wiki/Jazelle> นอกจากนี้จากที่ได้กล่าวมาแล้ว ผู้อ่านสามารถค้นควารายละเอียดอื่นๆ เพิ่มเติมได้ที่ https://en.wikipedia.org/wiki/ARM_architecture

ลำดับชั้นหน่วยความจำ (Memory Hierarchy)

หน่วยความจำลำดับชั้น (Memory Hierarchy) อาศัยการทำงานร่วมกันของหน่วยความจำหลายชนิด ต้นทุนที่หลากหลาย และหลายความจุเข้าด้วยกัน เพื่อผสานจุดเด่นของหน่วยความจำแต่ละชนิดเข้าด้วยกัน และช่วยประหยัดต้นทุนโดยรวมของระบบ บทนี้มีวัตถุประสงค์ดังนี้

- เพื่อให้เข้าใจถึงลำดับชั้นและการทำงานร่วมกันระหว่างหน่วยความจำชนิดต่างๆ
- เพื่อให้เข้าใจถึงความแตกต่างระหว่างหน่วยความจำชนิดต่างๆ
- เพื่อให้เข้าใจถึงการทำงานของแคชชนิด Direct Map และ Set Associative
- เพื่อให้เข้าใจถึงการทำงานของหน่วยความจำเสมือนชนิดเพจ (Paging Virtual Memory)
- เพื่อให้เข้าใจถึงการทำงานของหน่วยความจำเสมือนของบอร์ด Pi3

บทนี้จะอธิบายลำดับชั้นของหน่วยความจำ (Memory Hierarchy) ว่าเหตุใดคอมพิวเตอร์ จึงต้องประกอบด้วยหน่วยความจำหลายชนิด ซึ่งต่อเนื่องจากหัวข้อที่ 3.1.2 เนื้อหาในบทนี้อาศัยการทำงานของหน่วยความจำในบอร์ด Pi3 ภาคผนวก D การใช้งานระบบปฏิบัติการยูนิกซ์ ซึ่งจะทำให้ผู้อ่านเข้าใจหลักการและรายละเอียดมากขึ้น

5.1 การจัดวางหน่วยความจำชนิดต่างๆ ในเครื่องคอมพิวเตอร์

หน่วยความจำมีหลายชนิดตามหลักการออกแบบ ความซับซ้อน/เทคโนโลยีการผลิต ความจุของหน่วยความจำแต่ละชนิด มีความหลากหลาย ความเร็วหรือสมรรถนะในการอ่านหรือเขียนข้อมูล และ ต้นทุนต่อความจุหนึ่งหน่วย ทำให้การจัดวางมีผลต่อประสิทธิภาพโดยรวม หน่วยความจำบางชนิดสามารถบรรจุในชิป (On Chip) ในขณะที่บางชนิดจำเป็นต้องอยู่คู่กับชิปภายนอก (Off Chip) ทำให้การจัดวางหน่วยความจำชนิดต่างๆ ของบอร์ด Pi3 ซึ่งใช้ชิป BCM2837 เป็นกรณีศึกษาลำดับชั้นและการจัดวางหน่วยความจำชนิดต่างๆ

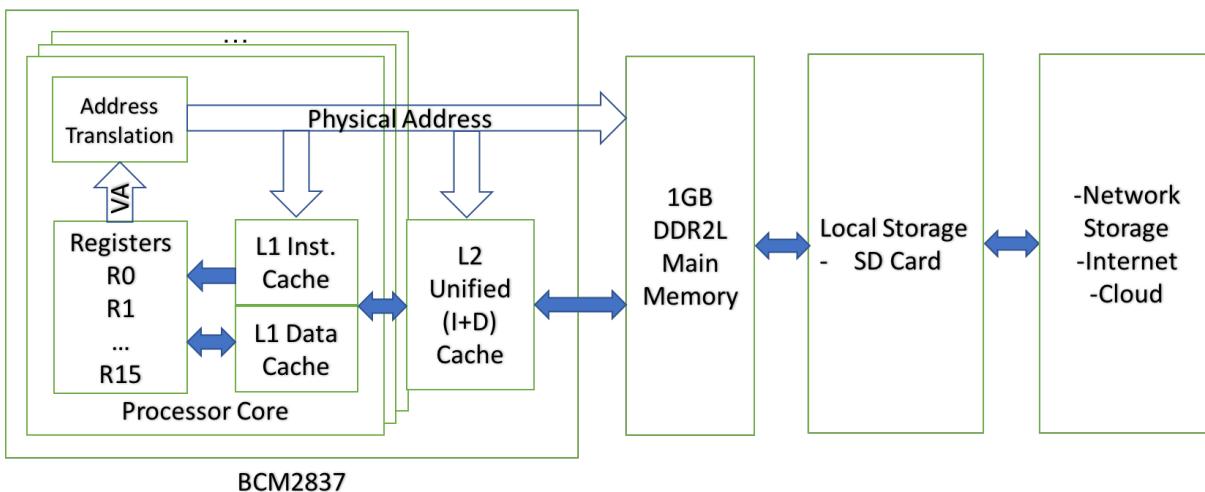


Figure 5.1: ลำดับชั้นของหน่วยความจำชนิดต่างๆ สำหรับชิป BCM2837, (VA: Virtual Address)

บอร์ด Pi3 ประกอบด้วย หน่วยความจำบางส่วนอยู่ในชิปเดียวกับไมโครโปรเซสเซอร์ (On Chip) และอยู่นอกชิป (Off Chip) ในรูปที่ 5.1 หน่วยความจำเหล่านี้อยู่ในลำดับชั้น (Hierarchy) ต่างๆ ทั้งใกล้และไกลจาก ALU (Arithmetic Logic Unit) ประกอบด้วย

- รีจิสเตอร์ (Register) ในสถาปัตยกรรมคำสั่ง ARMv7 ซึ่งเป็นเวอร์ชัน 32 บิต ประกอบด้วยรีจิสเตอร์ R0, R1, ... R15 ซึ่งสร้างมาจากหน่วยความจำชนิดสแตติกแรม (Static RAM: SRAM) สามารถเข้าถึงข้อมูลได้ภายในเวลาสั้นๆ หน่วยเพียง 1-2 นาโนวินาที (Nano Second)
- แคชเลเวล 1 (Level 1) ซึ่งแบ่งเป็นแคชคำสั่ง I (Instruction) Cache และแคชข้อมูล D (Data) Cache ซึ่งใช้เทคโนโลยีสแตติกแรม มีขนาดประมาณ 16-64 กิโลไบท์ (KiloByte: KB) ซึ่งมีแคชทั้งสองจะอยู่ด้วยกัน
- แคชเลเวลสามตัวด้วยกัน ซึ่งใช้เทคโนโลยีสแตติกแรม เช่น กิโลไบท์ 64 KB จนถึงหลักเมกะไบท์ (MegaByte: MB) แคชต่างมีการประสานงานร่วมกัน เพื่อให้แคชเลเวล 2 ทำหน้าที่คล้ายหน่วยความจำขนาดเล็กของแคชเลเวล 1 และในไมโครโปรเซสเซอร์จากผู้ผลิตบางราย ได้ออกแบบให้มีแคชเลเวล 3 ที่มีความจุใหญ่ขึ้น ทำหน้าที่คล้ายหน่วยความจำขนาดเล็กของแคชเลเวล 2 ซึ่งต้นทุนในการผลิตชิปเหล่านี้แปรผันตรงกับความจุของแคชเช่นกัน
- หน่วยความจำหลัก (Main Memory) หรือบางคราวใช้คำว่า Primary Memory หรือ หน่วยความจำปั๊มภูมิ ซึ่งใช้เทคโนโลยีไดนามิกแรม (Dynamic RAM: DRAM) มีขนาด 512 เมกะไบท์ ไปจนถึง吉igaByte (GigaByte: GB) รีจิสเตอร์ แคช และหน่วยความจำหลักไม่สามารถเก็บรักษาข้อมูลได้ หากผู้ใช้ปิดเครื่อง ไฟฟ้าขัดข้อง ไฟฟ้ากระชากหรือแรงดันตกช่วงขณะ หรือแหล่งจ่ายไฟหมดพลังงาน เช่น ในคอมพิวเตอร์พกพาซึ่งต้องอาศัยพลังงานจากแบตเตอรี่ ทำให้เครื่องคอมพิวเตอร์จะต้องมีหน่วยสำรองข้อมูลในลำดับชั้นถัดไป ในทำนองเดียวกัน เครื่องคอมพิวเตอร์ที่ต้องเปิดทำงานนานๆ ควรมีแหล่งจ่ายไฟสำรองหรือ UPS (Uninterrupted Power Supply) เช่นเดียวกัน

- หน่วยสำรองข้อมูล (Secondary Memory) หรือเรียกว่า หน่วยความจำทุติยภูมิ ซึ่งมีทางเลือกจาก หಲายเทคโนโลยี เพื่อเก็บรักษาข้อมูลไม่ให้สูญหาย ถึงแม้ผู้ใช้จะปิดเครื่องหรือแหล่งจ่ายไฟหมด พลังงาน เช่นใน คอมพิวเตอร์พกพาซึ่งต้องอาศัยพลังงานจากแบตเตอรี่ ชนิดของหน่วยสำรองข้อมูล โดยจะกล่าวรายละเอียดเพิ่มเติมในบทที่ 7

- ฮาร์ดดิสก์ (Hard Disk) ซึ่งใช้เทคโนโลยีหน่วยความจำแผ่นแม่เหล็ก (Magentic Disc) หมุน ด้วยความเร็วสูงคงที่ประมาณ 5,400-10,000 รอบต่อนาที มีขนาดหดใหญ่จิกไบท์จนถึงหดใหญ่ เทราไบท์ (TeraByte: TB)
- หน่วยความจำ SD ซึ่งสร้างจากเทคโนโลยีแฟลช (Flash) ทำให้มีความจุมาก ระดับ 16 จิกไบท์ขึ้นไป หน่วยความจำแฟลชความเร็วสูงกว่าเมื่อเทียบกับฮาร์ดดิสก์ พกพาได้สะดวก เพราะขนาดเล็กและน้ำหนักเบา จึงนิยมใช้กับกล้องดิจิทัล โทรศัพท์สมาร์ทโฟน และอื่นๆ
- โซลิดสเตทดิสก์ (Solid State Disk: SSD) ซึ่งใช้เทคโนโลยีหน่วยความจำแฟลช (Flash Memory) มีขนาด 128 จิกไบท์ขึ้นไปจนถึงหดใหญ่เทราไบท์ และคาดว่าจะทดแทนฮาร์ดดิสก์ใน ปัจจุบันและอนาคต
- หน่วยสำรองข้อมูลผ่านเครือข่าย (Network Storage) ซึ่งมีรูปแบบต่างๆ เช่น Network Attached Storage (NAS), Storage Area Network (SAN) Cloud Storage เป็นต้น

กล่าวโดยสรุปคือ หน่วยความจำในรูปของริจิสเตอร์อยู่ใกล้กับ ALU ใช้เวลาเข้าถึง (Access Time) สั้น กว่าเสมอ แต่จะมีความจุน้อย เนื่องจากความซับซ้อนและต้นทุนต่อความจุที่สูงกว่า การจัดแบ่งระดับชั้น ต่างๆ ของหน่วยความจำอาศัยหลักการใช้งานขึ้นในแกนเวลา (Temporal Locality) และในพื้นที่ใกล้เคียง (Spatial Locality) สามารถทำให้ผู้ใช้มองเห็นหน่วยความจำขนาดใหญ่ที่มีต้นทุนถูก แต่มีความเร็วสูงสุด เท่าที่จะเป็นไปได้

5.2 หน่วยความจำสแตติคแรม

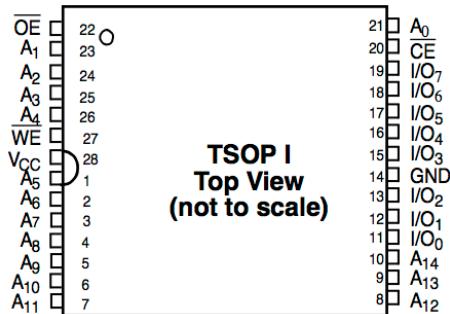


Figure 5.2: ตัวถังแบบ TSOP ของชิพหน่วยความจำชนิดสแตติค Cypress 62256

หน่วยความจำสแตติกนิยมใช้งานร่วมกับไมโครคอนโทรลเลอร์ ที่ต้องการสมาร์ตนาฬิกา หรือต้องการความเร็วสูง มีความจุหลายขนาด ตั้งแต่ 16 กิโลไบต์ ถึงหลายเมกะไบต์ ผู้เขียนขอใช้ หน่วยความจำสแตติคแรม (Static RAM: SRAM) หมายเลข CY62256 ใช้เทคโนโลยีการผลิตชนิด CMOS ผลิตในปี ค.ศ. 2002 เป็นชิปกรณีศึกษาถึงแม้ว่า จะเทคโนโลยีที่เก่าแล้ว เอกสารคุณลักษณะ (Datasheet) ของหน่วยความจำนี้ ([Cypress Semiconductor \(2002\)](http://ecee.colorado.edu/~mcclure1/Cypress_SRAM_CY62256.pdf)) สามารถค้นเจอที่ http://ecee.colorado.edu/~mcclure1/Cypress_SRAM_CY62256.pdf แต่ชิป SRAM มีลักษณะเด่น คือ

- 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) ซึ่งทำงานที่ลอจิกศูนย์ (Active Low) และใช้บัสข้อมูลร่วมกัน เพราะเชื่อมต่อกับชิพหน่วยความจำนี้ เพราะภายในชิปมี ไดรเวอร์สามสถานะ (Three State Driver)

ในเชิงโครงสร้าง ชิพอาศัยตัวถังแบบ TSOP (Thin Small Outline Package) เพื่อให้ เหมาะสมสำหรับ การบัดกรีเพื่อยืดตัวถังชิพบนแผ่นวงจรพิมพ์แบบ Surface Mount ผู้อ่านสามารถค้นคว้าเรื่องการห่อหุ้มชิพด้วยแพ็กเกจชนิด TSOP และอื่นๆ ได้ที่ https://en.wikipedia.org/wiki/Thin_Small_Outline_Package ซึ่งแต่ละชนิดมีข้อได้เปรียบและเสียเปรียบแตกต่างกันไป

5.2.1 โครงสร้างภายในและบิตเซลของหน่วยความจำสแตติค

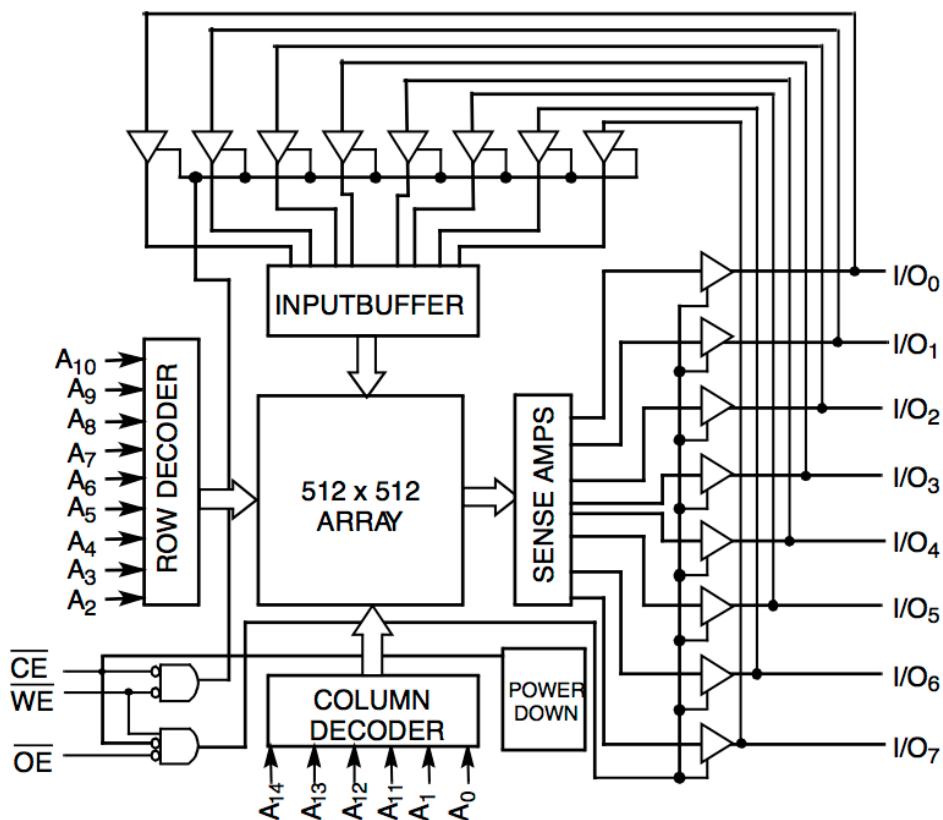


Figure 5.3: โครงสร้างของหน่วยความจำนิิดสแตติค Cypress 62256

โครงสร้างของชิป CY62256 จัดเรียงเป็น $32K \times 8$ บิต หรือ $2^5 \times 2^{10} \times 2^3 = 2^{9+9}$ หรือ 512×512 บิต ซึ่งจะตรงกับรูปที่ 5.3

- การอ่านข้อมูลจะเกิดขึ้นโดยเปลี่ยนสัญญาณที่ขาตั้งต่อไปนี้
 - ขา $\overline{CE}=0$ และ $\overline{OE}=0$
 - ขาแอดдрес A0-A14 ทั้งหมด 15 ขา รับสัญญาณแอดдрес เพื่อป้อนให้วงจร Decoder ทั้ง วนวนบน (Row Decoder) และวนคอลัมน์ (Column Decoder) ทำหน้าที่สร้างสัญญาณจำนวน 512 เส้นในวนวนบนและวนตั้งจำนวน 512 เส้น เพื่อเลือกบิตเซลล์ที่ต้องการในอะเรย์
 - โมดูล Sense Amplifier เป็นวงจรอนาล็อกขยายสัญญาณความไวสูง ใช้สำหรับขยายสัญญาณที่ส่งออกมาจากอะเรย์หน่วยความจำ เพราะสัญญาณที่ได้มีระดับโวลต์เจต้า รายละเอียดเพิ่มเติมที่ https://en.wikipedia.org/wiki/Sense_amplifier
 - สัญญาณที่ผ่านการขยายและตรวจจับแล้วจะกล้ายเป็นข้อมูลดิจิทัลผ่านเกต Three State Buffer ออกไปยังขา I/O₀ จนถึง I/O₇ พร้อมกันทั้ง 8 บิต
- การเขียนข้อมูลจะเกิดขึ้นโดยเปลี่ยนสัญญาณที่ขาตั้งต่อไปนี้
 - ขา $\overline{CE}=0$ $\overline{WE}=0$ และ $\overline{OE}=1$

- ข้อมูลจะไหลเข้าทางขา I/O_0 จนถึง I/O_7 ผ่านเกต Three State Buffer มาพักใน
- โมดูล Input Buffer ใช้สำหรับพักข้อมูลชั่วคราว เพื่อรอเขียนในบิตเซลล์ที่ต้องการพร้อมกันทั้ง 8 บิต
- ขาแอดเดรส A0-A14 ทั้งหมด 15 ขา รับสัญญาณแอดเดรส เพื่อป้อนให้วงจร Decoder ทั้ง วนวนอน (Row Decoder) และแนวคอลัมน์ (Column Decoder) ทำหน้าที่สร้างสัญญาณจำนวน 512 เส้นในวนวนอนและแนวตั้งจำนวน 512 เส้น เพื่อเลือกบิตเซลล์ที่ต้องการเขียนใน อาร์ย พร้อมกันทั้ง 8 บิต

แต่ละบิตเซลล์ ประกอบด้วยฟลิปฟล็อป (Flip-Flop) และเกตเพื่อควบคุมการอ่านและเขียน มีโครงสร้างที่ซับซ้อนและบริโภคพลังงานตลอดเวลา เนื่องจากมีการจัดเรียงเป็นลูปป้อนกลับ (Feedback Loop) บิตเซลล์ใช้พื้นที่ซิลิคอน (Silicon Area) ต่อความจุข้อมูล 1 บิตคิดเป็นจำนวนทรานซิสเตอร์เฉลี่ย เท่ากับ ทรานซิสเตอร์ 6 ตัวขึ้นไป ผู้อ่านสามารถค้นค่าวารายละเอียดเพิ่มเติมได้ที่เว็บ https://en.wikipedia.org/wiki/Static_random-access_memory

5.2.2 การทำงานของสแตติกแรม: อ่านและเขียน

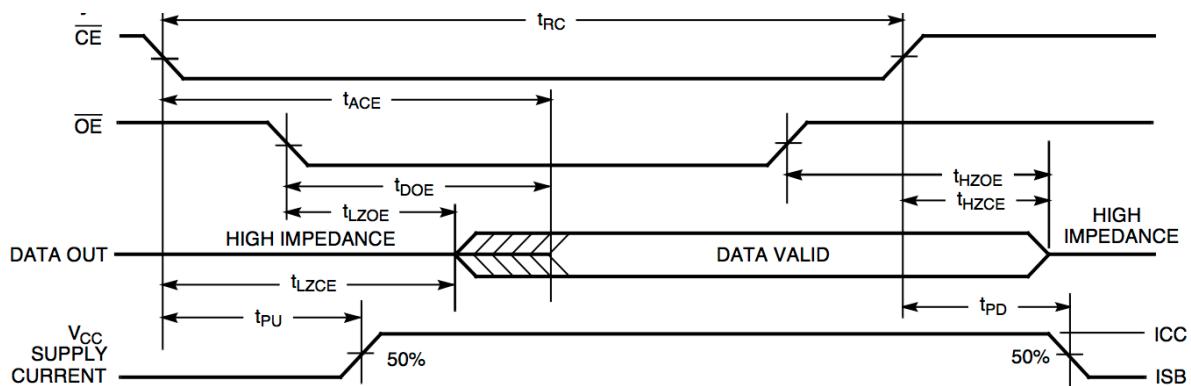


Figure 5.4: ไดอะแกรมเวลา (Timing Diagram) สำหรับการอ่าน (Read) ข้อมูลของหน่วยความจำชนิด สแตติกแรม

ไดอะแกรมเวลาของหน่วยความจำชนิดสแตติก ในรูปที่ 5.4 แกนแนวนอนเป็นแกนเวลา เริ่มต้นจาก ซีพียู 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 คาบเวลาที่สั้นที่สุดในการอ่านข้อมูลจากสแตติกแรม (Read Cycle Time) $t_{RC} > t_{ACE}$
- t_{ACE} เรียกว่า เวลาสำหรับการเข้าถึงข้อมูล หรือ Access Time โดยเริ่มนับจากเมื่อสัญญาณ CE 1->0 จนวงจรสามารถอ่านข้อมูลได้ถูกต้อง
- t_{HZCE} เรียกว่า เวลาที่จะรักษาข้อมูลไว้บนบัสข้อมูลเมื่อเปลี่ยนหมายแอดเดรสไปแล้ว

- 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

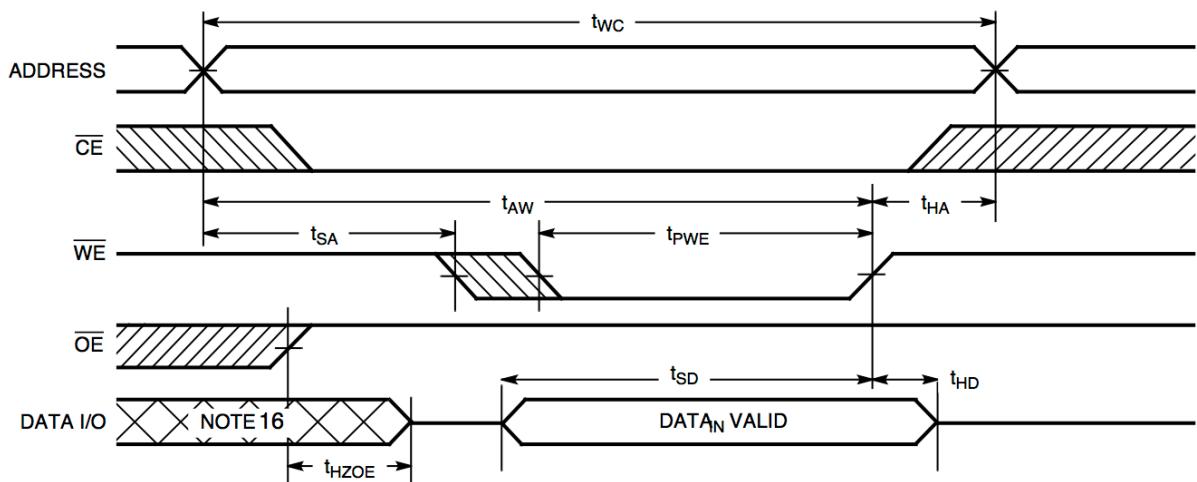


Figure 5.5: ไดอะแกรมเวลา (Timing Diagram) สำหรับการเขียน (Write) ข้อมูลของหน่วยความจำชนิดสแตติกแรม

Timing Diagram ของหน่วยความจำชนิดสแตติก ในรูปที่ 5.5

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

5.3 หน่วยความจำไดนามิกแรม (Dynamic RAM: DRAM)

ผู้ผลิตเครื่องคอมพิวเตอร์โน้ตบุ๊ก โทรศัพท์เคลื่อนที่สมาร์ทโฟน และอุปกรณ์พกพาต่าง นิยมออกแบบติดตั้ง ชิพหน่วยความจำ DRAM บน เมนบอร์ด (Main Board) เช่นเดียวกับบอร์ด RPi เพื่อลดขนาดและปริมาตร ของเครื่องลง ขบวนการผลิต DRAM ยังไม่สามารถกับขบวนการผลิตไมโครโปรเซสเซอร์ได้ ผู้ผลิตจึง จำเป็นต้องผลิตชิพ DRAM แยกต่างหาก ในทางกลับกัน ไมโครโปรเซสเซอร์ และ ไมโครคอนโทรลเลอร์ สามารถบรรจุ SRAM แทน เพื่อทำหน้าที่เป็นรีจิสเตอร์ แคช และหน่วยความจำหลัก ตามความต้องการและ ต้นทุนที่หลายหลาย เทคโนโลยีปัจจุบันของ DRAM คือ เทคโนโลยี DDR4 https://en.wikipedia.org/wiki/DDR4_SDRAM ซึ่งทันสมัยกว่าที่จะใช้เป็นกรณีศึกษาในตำราเล่มนี้

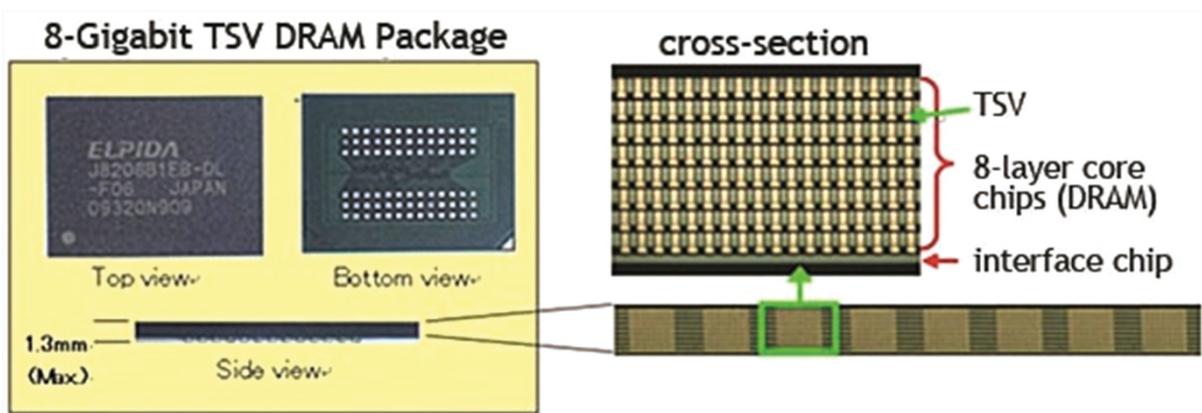


Figure 5.6: รูปถ่ายด้านบน (Top View) ด้านล่าง (Bottom View) ด้านข้าง (Side View) และภาพตัด ขวาง (Cross Section) ของชิพ DRAM โดยใช้เทคโนโลยี TSV (Through Silicon Via) ผู้ผลิตบริษัท Elpida

รูปที่ 3.4 แสดงให้เห็นภาพชิพหน่วยความจำ DDR2 DRAM ซึ่งติดยืดติดบอร์ด Pi3 โดยรายละเอียด เป็นตัวตั้งได้กล่าวไว้บ้างแล้วในบทที่ 3 หัวข้อที่ 3.1.2 ชิพ DRAM มีความจุสูงมาก เนื่องจากใช้จำนวน ทรานซิสเตอร์เฉลี่ยเพียง 1-2 ตัวต่อความจุ 1 บิตซึ่งน้อยกว่าสแตติคแรม แต่มีข้อเสีย คือ ใช้เทคโนโลยีและ ขบวนการผลิตที่ซับซ้อน เนื่องจากชิพ DRAM มีความจุสูงมาก ในทางตรงกันข้าม ผู้ใช้ต้องการความเร็ว มั่นในตัวอุปกรณ์สูง และความถี่คลื่อกสูงเช่นกัน ทำให้ต้นทุนต่อความจุหนึ่งบิตของ DRAM สูงกว่าหน่วย สำรองข้อมูล

ชิพ DDR2 DRAM กรณีศึกษาที่ใช้บนบอร์ด Pi3 ผลิตโดยบริษัท Elpida ในปี ค.ศ. 2014 [Micron Technology \(2014\)](#) และต่อมาระบิษัท ไมโครคอน เทคโนโลยี ได้เข้าถือหุ้นแทน ใช้โครงสร้างแพ็คเกจและขาชนิด BGA (Ball Grid Array) จำนวน 168 ขาด้านล่างเพียงอย่างเดียว ขนาด 12 มม. x 12 มม. x 0.8 มม. ผู้อ่าน จะสังเกตได้ว่า แพ็คเกจชนิด BGA นี้จะใช้พื้นที่บนแผ่นวงจรพิมพ์เล็กกว่า ในทางตรงกันข้าม ชิพ SRAM ชนิด TSOP ในรูปที่ 5.2 ซึ่งมีขายืนอกมาด้านข้าง

ชิพ DDR2 นี้ สามารถกำหนดขนาดข้อมูลจากการอ่านเขียนต่อเนื่อง (Burst length) เป็น 4, 8 และ 16 ตำแหน่งต่อเนื่องกัน สัญญาณคลื่อกความถี่สูงสุด 400 เมกะเฮิร์ตซ์ ซึ่งปัจจุบัน ปี 2018 ความถี่สูงสุด คือ 2,133 เมกะเฮิร์ตซ์ ในชิพ DDR4 สำหรับคอมพิวเตอร์ตั้งโต๊ะ โดยชิปกรณีศึกษาใช้เทคโนโลยีการผลิต เดียวกัน และมีลักษณะเชิงกายภาพคล้ายกับชิพ DRAM ในรูปที่ 5.6 ต่อไปนี้

รูปที่ 5.6 แสดงชิป DRAM ขนาด 8 Gbit โดยบริษัท Elpida ที่มา: <https://electroiq.com/2009/09/elpida-stacks-8-drams/> ภาพถ่ายด้านบน (Top view) มีส่วนซึ่งมีบริษัทผู้ผลิต หมายเลขอุตสาหกรรม 39 ของประเทศไทย และอื่นๆ ภาพถ่ายด้านล่าง แสดงให้เห็นขาสำหรับเชื่อมต่อเข้าสู่ภายในชิป จำนวน สองเสา ละ 39 ขา และรอยบางครั้งมุมขวาล่าง เมื่อพิจารณาว่าล่างจะตรงกับจุดวงกลมด้านซ้ายล่างของ ภาพถ่ายด้านข้างของชิป (Side view) มีความสูงเท่ากับ 1.3 มิลลิเมตร เมื่อขยายภาพตัดขวาง (cross-section) ให้ใหญ่ขึ้น ผู้อ่านจะมองเห็นขาที่ยื่นออกมาจากตัวถัง ภาพตัดขวางที่ขยายทางด้านขวาไม่อุด จะมองเห็นเป็น 8 ชั้น (8-layer core) การเชื่อมต่อสัญญาณบนแผ่นซิลิกอนแต่ละชั้น จะเกิดขึ้นในแนวราบภายในแผ่นหรือชั้นเดียวกัน แต่การเชื่อมระหว่างชั้นของแผ่นซิลิกอนเข้าด้วยกันในแนวตั้ง โดยใช้ TSV (Through Silicon Via, https://en.wikipedia.org/wiki/Through-silicon_via เพื่อเชื่อมสัญญาณควบคุม สัญญาณคลีอก ไฟเลี้ยง กราวด์ และอื่นๆ เกิดการเชื่อมต่อสัญญาณครบทั้งสามมิติ ทำให้ประหยัดพื้นที่ทางแผ่นซิลิกอนในแนวราบ ส่วนชั้นล่างสุดเป็นวงจรเชื่อมต่อ (interface) ซึ่งอยู่ใกล้กับขาที่อยู่ใต้ชิป

5.3.1 โครงสร้างภายในของชิป DRAM

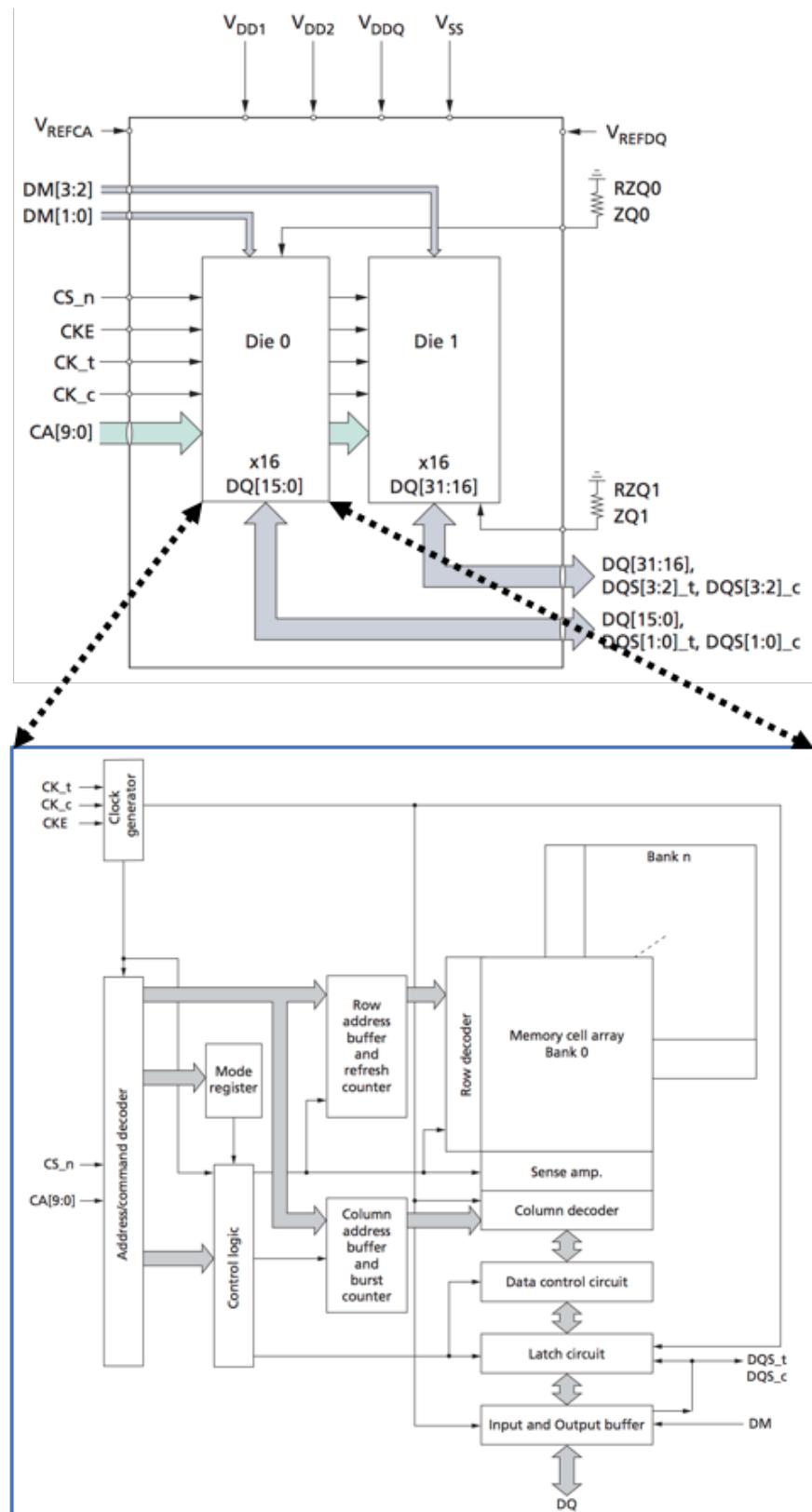


Figure 5.7: โครงสร้างภายในชิปหน่วยความจำ DRAM ชนิด DDR2 ประกอบด้วยดาย (Die) 2 ชิ้น แต่ละชิ้นมีบัสข้อมูลความกว้าง 16 บิต รวมเป็น 32 บิต

ในรูปที่ 5.7 ชิป DRAM กรณีศึกษา โครงสร้างของหน่วยความจำชนิด DDR2 จำนวน 2 ดาย (die) แต่ละ ดายประกอบด้วย แบงอะเรย์ DRAM จำนวน 8 ชั้น หรือ 8 แบงค์ (Bank) ชั้นละ 32 เมกกะเซลล์ x 16 บิต คิดเป็น $2 \times 8 \times 32$ เมกกะเซลล์ x 16 บิต x ต่อชิป 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 แต่ละแบงค์ประกอบด้วยวงจรจดจำรหัสแอดเดรส (Address Decoder) ในแนวนอน (Row Decoder) และแนวตั้ง (Column Decoder) ภายในชิปประกอบด้วยขาสัญญาณต่างๆ เรียงตาม ลำดับความสำคัญ ดังนี้

- CS_n (Chip Select Not) หรือ \overline{CS} ใช้เปิด/ปิดการทำงานของชิป เพื่อช่วยประหยัดพลังงาน
- CKE (Clock Enable) เมื่อสัญญาณ $\overline{CS}=0$ เพื่อให้ชิปทำงาน หลังจากนั้น สัญญาณ CKE ใช้เปิด/ปิดการทำงานของคลีอก กรณีพักการใช้งานชั่วคราวเพื่อช่วยประหยัดพลังงาน
- CK (Clock) และ CK# (Clock Not) คือ คุณสมบัติที่ต้องข้ามกัน เรียกว่า คุณภาพเฟอร์เนเชียล (Differential) หน่วยเป็นเมกะเฮิรท์ และมีแนวโน้มเพิ่มขึ้น ปัจจุบันความถี่สูงสุดเท่ากับ 3,000 เมกะเฮิรท์
- แอ็ตเตอร์สบส CA[0:9] ขนาด 10 บิต ใช้สำหรับรับสัญญาณแอดเดรสและคำสั่ง (Command) โดย การมัลติเพล็กซ์ (Multiplex) จากชิป
 - สัญญาณแอดเดรสแคว (Row Address) จำนวน 14 บิต และแอ็ตเตอร์สคอลัมน์ (Column Address) จำนวน 11 บิต จะพักเก็บในวงจรบัฟเฟอร์ เพื่อป้อนให้กับวงจรจดจำ (Decoder) คล้ายกับการทำงานของหน่วยความจำ SRAM
 - โหมดการทำงาน ได้แก่ Power Up, Deep Power Down, Active, Idle, Reading, Writing, Precharging, Refreshing เป็นต้น คำสั่งต่างๆ ดังนี้ Activate, Burst Read, Burst Write, Refresh, Power Down, Precharge และ Burst Terminate เป็นต้น
- ดาต้าสตรอบ DQS[0':3] ขนาด 4 บิต ใช้สำหรับควบคุมการอ่านและการเขียนข้อมูล
- ดาต้าบัส DQ[0:31] จำนวน 32 บิต หรือ 4 ไบท์ สำหรับอ่านข้อมูลจากชิป DRAM และเขียนข้อมูล ในชิป
- DM (Data Mask) [0:3] ขนาด 4 บิต ใช้สำหรับมาสก์ (Mask) เพื่อควบคุมการเขียนข้อมูลแต่ละไบท์ โดย DM[0] ควบคุมไบท์ที่ 0, DM[1] ควบคุมไบท์ที่ 1 ตามลำดับ ทำให้การเขียนข้อมูลแต่ละไบท์ เป็นอิสระจากกันได้

ผู้อ่านจะสังเกตเห็นว่า วงจรรอบๆ อะเรย์หน่วยความจำ (Cell Array) มีลักษณะที่คล้ายกับสแตติคแรม คือ

- 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

จุดตัดระหว่างทั้งสองสีคือ ตำแหน่งของบิตเซลล์ที่จะอ่านหรือเขียน คล้ายกับการทำงานของ แต่โครงสร้างของบิตเซลล์แตกต่างกันมาก บิตเซลล์แต่ละบิตประกอบด้วยทรานซิสเตอร์ชนิด MOS อย่างน้อยจำนวน 1 ตัว โดยอาศัยหลักการมีประจุในขาเกต (Gate) หรือ ไม่มีประจุในขาเกต เมื่อที่ขาเกตมีประจุจะทำให้ทรานซิสเตอร์ไม่สามารถนำกระแสจากขา Source ไปขา Drain ได้ และในทางตรงกันข้าม เมื่อขาเกตไม่มีประจุเนื่องจากทำให้ทรานซิสเตอร์นำกระแสได้ ผู้อ่านสามารถคลิกบนคำว่า **บิตเซลล์ของไดนามิกแรม** เพื่อค้นคว้าโครงสร้างบิตเซลล์ของ DRAM ทั่วไป

5.3.2 การทำงานของไดนามิกแรม: อ่านและเขียน

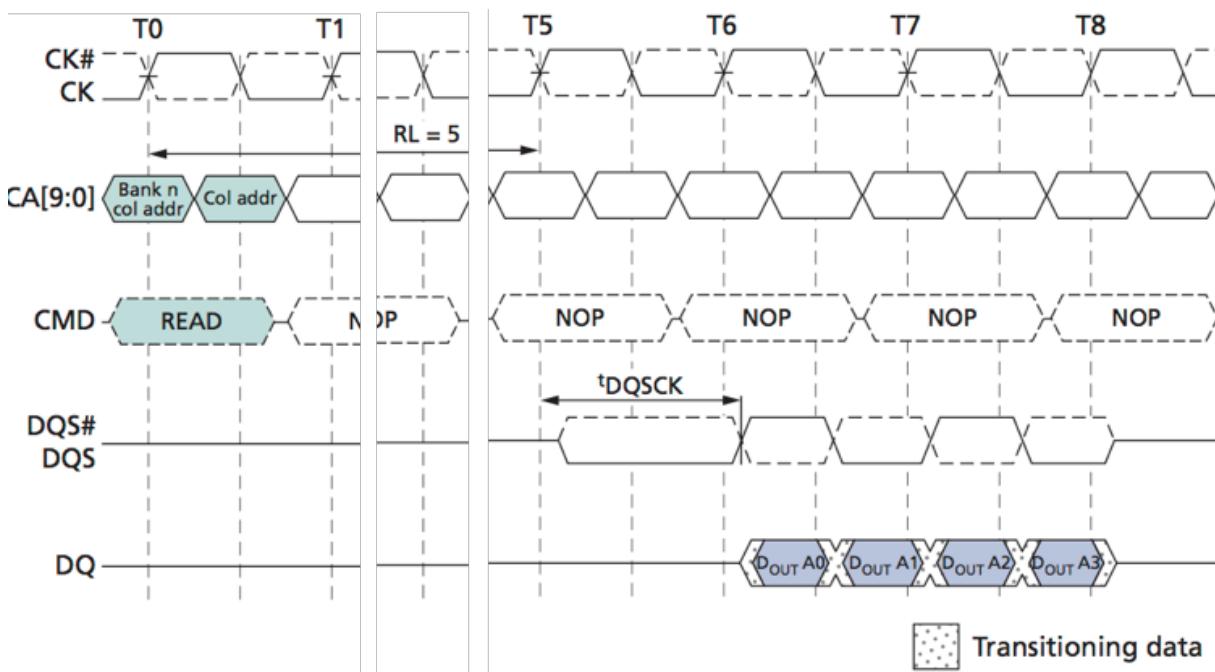


Figure 5.8: ไดอะแกรมเวลา (Timing Diagram) สำหรับการอ่านข้อมูลของหน่วยความจำ DRAM รุ่น Elpida B8132B4PB-8D-F คำสั่ง Burst READ โดย RL= 5 BL= 4 หมายเหตุ รูปมีการตัดต่อเพื่อเน้นบริเวณที่สำคัญ, NOP=No Operation

ตัวอย่าง คำสั่ง Burst READ เริ่มต้นโดย ทำให้สัญญาณ $CS\#=0$, $CA0=1$, $CA1=0$ และ $CA2=1$ ณ ขอบขาขึ้นของสัญญาณคล็อก รูปที่ 5.8 แสดงถึงไดอะแกรมเวลาสำหรับการอ่านข้อมูลของหน่วยความจำ ชนิด DDR แบบชิงโคนนัสซึ่งต้องทำงานด้วยขอบขาขึ้นและขาลงของสัญญาณคล็อก ในรูปเป็นการทำงานตามคำสั่ง Burst READ ด้วยค่า RL (Read Latency) = 5 ไบคิล นับจาก T1-T5 โดยต้องรอเวลาโดยการนับจำนวนคล็อกเป็นจำนวน RL (Read Latency) ไบคิล โดยเริ่มนับจากขอบขาขึ้นของคำสั่ง READ หมายเหตุ รูปมีการตัดต่อเพื่อเน้นบริเวณที่สำคัญ และอ่านข้อมูลต่อเนื่องเป็นจำนวน BL (Burst Latency) = 4 ชุด เรียงติดกันโดยเริ่มจากแอดдресที่น้อยกว่า (A0) วงจรจะใช้สัญญาณ DQS (Data Strobe) เพื่อชิงโคนนัสกับการอ่านข้อมูล ที่มาสำหรับ **ดาวน์โหลดเอกสาร**

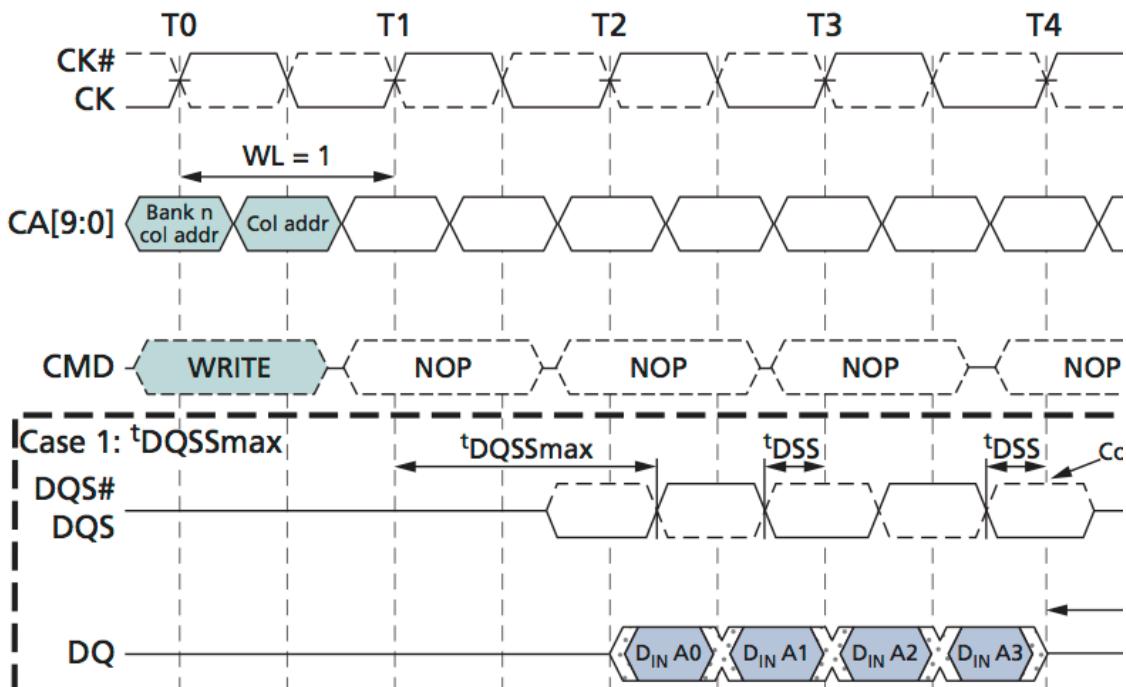


Figure 5.9: ไดอะแกรมเวลา (Timing Diagram) สำหรับการเขียนข้อมูลของหน่วยความจำ Elpida รุ่น B8132B4PB-8D-F ตามคำสั่ง Burst WRITE – WL = 1, BL = 4 หมายเหตุ รูปมีการตัดต่อเพื่อเน้นบริเวณที่สำคัญ, NOP=No Operation

ต่อมาเป็นตัวอย่างการเขียนด้วยคำสั่ง Burst WRITE ในรูปที่ 5.9 โดย WL (Write Latency) = 1 ไซเดลและ BL (Burst Latency) = 4 ชุด เรียงติดกันโดยเริ่มจากแอดдресที่น้อยกว่า (A0) วงจรจะใช้สัญญาณ DQS (Data Strobe) เพื่อซิงโครไนซ์กับการเขียนข้อมูล ผู้อ่านจะสังเกตเห็นว่า ขอบสัญญาณ DQS จะเปลี่ยนแปลงตรงกลางข้อมูลสำหรับการเขียน แต่ในรูปที่ 5.8 ขอบสัญญาณ DQS จะเปลี่ยนแปลงตามข้อมูลสำหรับการอ่าน

5.3.3 การรีเฟรชข้อมูล

การรีเฟรช (Refresh) คือ การอ่านข้อมูลที่อยู่ในบิตเซลต่างๆ แล้วเขียนซ้ำลงไป เพื่อป้องกันไม่ให้ประจุที่เก็บอยู่ในบิตเซลต่างๆ รั่วไหลหายไป เนื่องจากเซลต่างๆ ที่ใช้เก็บข้อมูล ทำหน้าที่เก็บ ('1') หรือไม่เก็บ ('0') ประจุไฟฟ้า เมื่อเวลาผ่านไปไม่กี่มิลลิวินาที จึงเกิดการรั่วไหลของประจุเหล่านี้ เมื่อจำนวนประจุลดลง การแยกแยะระหว่างบิตเซลที่มีและไม่มีประจุจึงยากขึ้น ซึ่งอาจทำให้เกิดความผิดพลาดในการอ่าน

แกนนอนในรูปที่ 5.10 คือ แกนเวลา การรีเฟรชทุกๆ 32 มิลลิวินาที โดยใช้ Refresh คอมมานด์ 4096 คำสั่ง เป็นระยะเวลา $t_{REFW}=32$ มิลลิวินาที แต่ละคอมมานด์จะห่างกันด้วยระยะเวลา Refresh Interval $t_{REFI}=7.8$ ไมโครวินาที หนึ่งคอมมานด์จะทำการ Refresh เป็นชุดๆ แต่ละชุดด้วยระยะเวลา Refresh Burst Window $t_{REFBW}=4.16$ ไมโครวินาที โดยแต่ละชุด จะทำการรีเฟรชทีลังแบงค์ด้วยระยะเวลา 60 นาโนวินาที

การอ่านหรือเขียนหน่วยความจำจะต้องหยุดรอ หากมีการรีเฟรชดำเนินกร้อยู่ ในทำนองเดียวกัน หากมีการอ่านหรือเขียนข้อมูลจริง การรีเฟรชจะต้องหยุดรอ เพื่อให้กระบวนการอ่านหรือเขียนนั้นเสร็จสิ้นก่อน

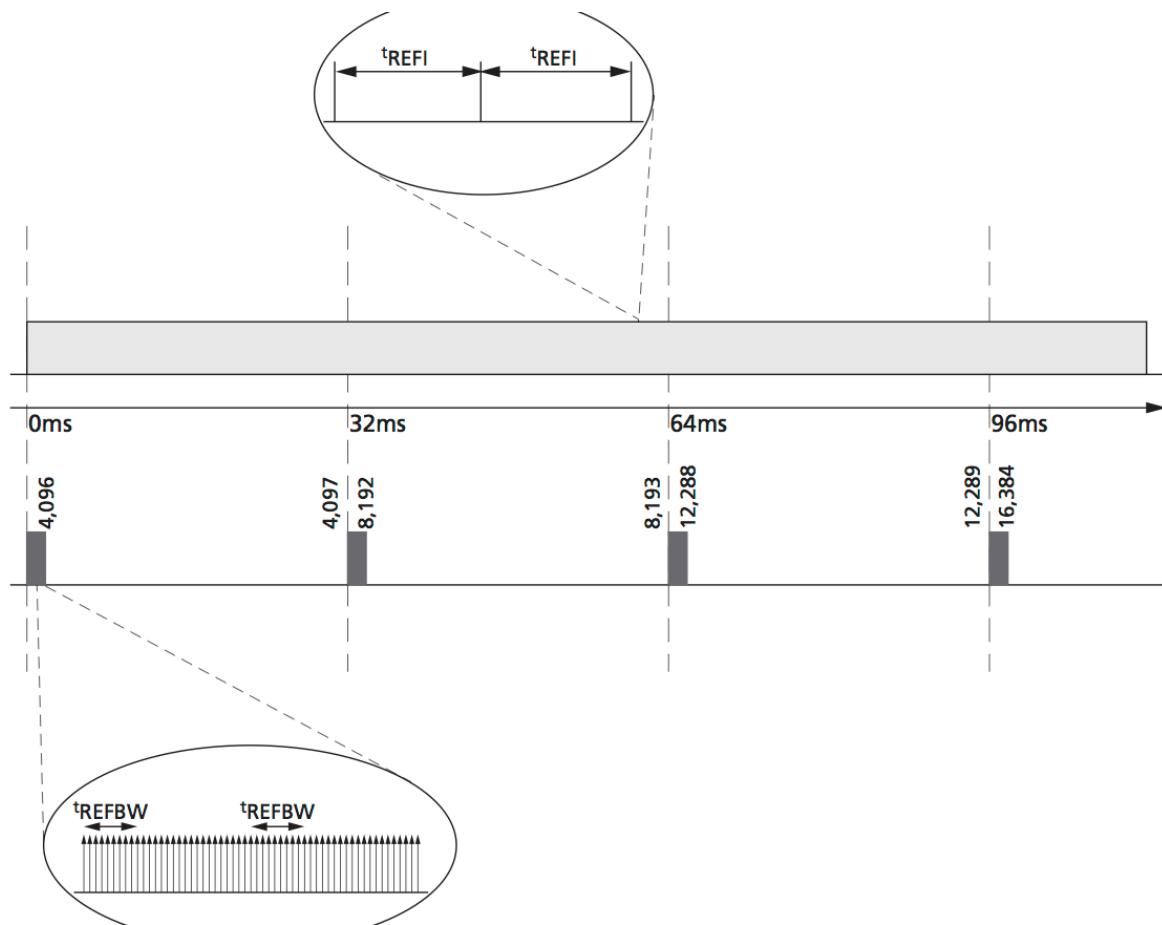


Figure 5.10: ไดอะแกรมเวลาสำหรับการรีเฟรชหน่วยความจำชนิด DRAM ขนาด 1 กิกะบิต

สเตติคแรมไม่ต้องมีการรีเฟรชข้อมูล เนื่องจากการจัดเก็บข้อมูลใช้วิธีการเก็บข้อมูล ที่แตกต่างกับหน่วยความจำ DRAM ทำให้ SRAM มีสมรรถนะ และประสิทธิภาพสูงกว่า แต่ต้องใช้จำนวนทรานซิสเตอร์มากกว่า จึงทำให้ใช้พื้นที่ซิลิโคนต่อความจุข้อมูล 1 บิต มากกว่าเซ็นกัน ทำให้ SRAM ใช้พลังงานไฟฟ้าต่อความจุ 1 บิตโดยเฉลี่ยสูงกว่า ผู้อ่านสามารถค้นคว้าเพิ่มเติมได้ที่ https://en.wikipedia.org/wiki/Dynamic_random-access_memory

5.4 หลักการพื้นฐานของหน่วยความจำแคช (Cache)

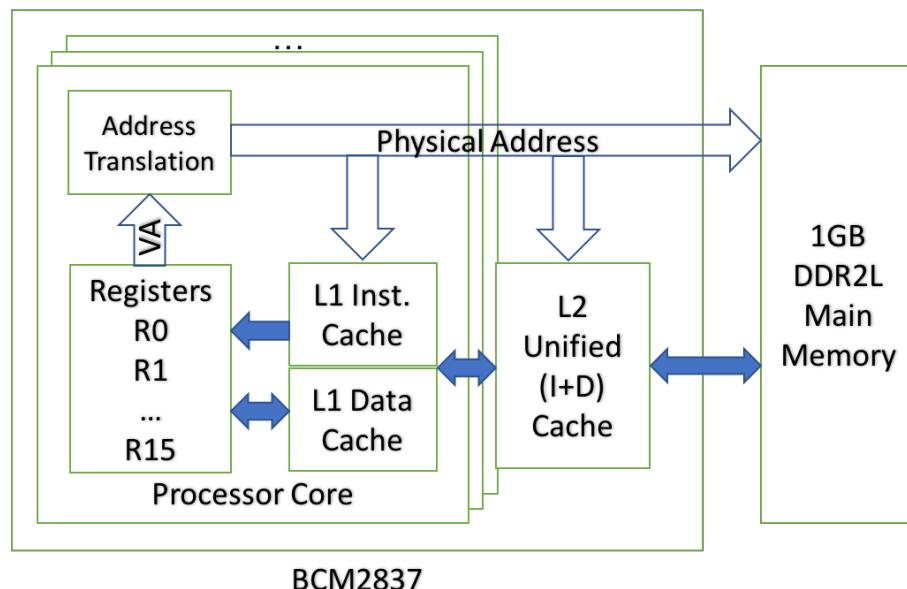


Figure 5.11: การเชื่อมต่อระหว่างรีจิสเตอร์ แคชเลเวล 1 และ 2 และหน่วยความจำหลัก (VA: Virtual Address) ภายในชิป BCM2837

รูปที่ 5.11 การเชื่อมต่อระหว่างรีจิสเตอร์ แคชเลเวล 1 และ 2 และหน่วยความจำหลัก ทำงานต่อเนื่องจากการแปลง VA ให้เป็น PA ประกอบด้วย L1 แบ่งเป็น Instruction และ ข้อมูล (Data Only) เส้นสีขาว คือ แอดเดรสสายภาพ เส้นสีน้ำเงิน คือ คำสั่งและข้อมูล ซึ่งจะอ่านคำสั่งจากแคชคำสั่งเลเวล 1 ก่อน หากไม่เจอก็จะค้นคำสั่งนั้นจากแคชเลเวลถัดไป หากคำสั่งนั้น เป็นคำสั่งประเภท Load หรือ คำสั่ง STORE ระหว่างที่ซีพียูปฏิบัติตาม (Execute) จะค้นหา/เขียน ข้อมูลในแคชข้อมูล เลเวล 1 ก่อน เช่นกัน แคช L2 และ L3 เป็นแบบรวม (Unified Cache) ทำหน้าที่พักเก็บคำสั่งและข้อมูล

หน่วยความจำหลักซึ่งอาศัยหน่วยความจำชนิด DRAM ส่วนใหญ่จะอยู่อิสระเป็นชิปแยกจากไมโครโปรเซสเซอร์ ในขณะที่ แคชจะมีอยู่บนชิป (On Chip) เดียวกัน ทำให้เกิดความล่าช้าระหว่าง การเคลื่อนที่ของข้อมูล ดังที่ได้กล่าวไว้แล้วในหัวข้อที่ 5.3 การพักเก็บข้อมูลหรือคำสั่งที่มีการเรียกใช้บ่อยในแคช ทำให้ แคชเป็นประโยชน์ต่อระบบโดยภาพรวม เนื่องจากช่วยประหยัดเวลาในการรอการเดินของ ข้อมูลหรือคำสั่งจากชิปสู่ชิป (Chip-Chip Delay) แคชจึงทำหน้าที่คล้ายกับบัฟเฟอร์ ในการพักเก็บข้อมูลหรือคำสั่ง ที่อ่านจากหน่วยความจำหลัก เพราะการทำงานของซอฟต์แวร์ส่วนหนึ่งจะเป็นการวนรอบ ทำให้ใช้คำสั่งเดิมๆ ซ้ำแล้วซ้ำอีก เรียกว่า Temporal Locality

การอ่าน/เขียนข้อมูลโดยเฉพาะข้อมูลชนิดอะเรย์ ซึ่งมีการจัดเรียงคล้ายกับรูปที่ 2.12 มีรูปแบบ (Pattern) ของความต่อเนื่องกันตามลำดับเข็นเดียวกัน ดังนั้น ลักษณะนี้เรียกว่า Spatial Locality เชิงพื้นที่ (Space) ตรงกับการอ่าน/เขียนข้อมูลจาก DRAM แบบ Bulk Mode ทั้งสองชนิด เรียกรวมกันว่า Principle of Locality

แคช L1 นิยมออกแบบด้วยแคชชนิด Direct Map แคช L2 เป็นแบบรวมทำหน้าที่พักเก็บคำสั่งและข้อมูล จึงมีความจุมากกว่าแคช L1 นิยมออกแบบด้วยแคชชนิด Set Associative ผู้ออกแบบนำเอาหลัก

การของสแตติคแรมมาทำหน้าที่เป็นแคชให้กับหน่วยความจำไดนามิกแรม มีวิธีการ 3 หลักการ แต่ต่างๆ เล่นี้จะเปรียบเทียบเพียงแค่สองชนิดที่นิยมและเข้าใจง่าย โดยใช้ชุดคำสั่งเดียวกันในตารางที่ 5.1

Table 5.1: ตัวอย่างโปรแกรมภาษาแอสเซมบลีเพื่อประกอบการทำงานของแคช (PA: Physical Address)

PA	Label	Code	Comment
0		LDR R1, [R3]	; get value of variable A
1		LDR R2, [R4]	; get value of variable B
2		BL Adder	; call A+B
3		B XYZ	; jump to XYZ
4			;
5			
6			
7			
8			
9			
10	Adder	ADD R1, R2, R1	; A=A+B
11		BX LR	
12			
13			
14	XYZ	...	
15			

5.4.1 แคชชนิดไดเร็คแมป (Direct Map Cache)

รูปที่ 5.12 การทำงานของแคชชนิด นิยมใช้ออกแบบแคชเลเวล 1 (Level 1) ซึ่งมีความจุขนาดเล็ก เร็ว เก้าอี้ดนตรี คำสั่งที่หน่วยความจำหลายๆ ตำแหน่ง แยกกันมาอยู่ในแคชตำแหน่งเดียวกัน ซึ่งในรูป หน่วยความจำมีขนาดเป็น 2 เท่าของขนาดแคช ทำให้เกิดการแบ่งขั้น 2:1

(a) เริ่มต้น แคชไม่มีข้อมูลใดๆ

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

(c) คำสั่ง LDR R2, [R4] จะต้องถูกเฟล์ชจากหน่วยความจำ แต่เนื่องจากแคชตำแหน่งที่ 1 ว่างเปล่า ทำให้ต้องนำคำสั่งนี้จากหน่วยความจำไปเก็บในแคชในตำแหน่งที่ 1 และนำไปถอดรหัสต่อไป

(d) คำสั่ง BL Adder จะต้องถูกเฟล์ชจากหน่วยความจำ แต่เนื่องจากแคชตำแหน่งที่ 2 ว่างเปล่า ทำให้ต้องนำคำสั่งนี้จากหน่วยความจำไปเก็บในแคชในตำแหน่งที่ 2 และนำไปถอดรหัสต่อไป โดยจะต้องกระโดดไปยังหน่วยความจำตำแหน่งที่ 10 ต่อไป

(e) คำสั่ง ADD R1, R2, R1 จะต้องถูกเฟล์ชจากหน่วยความจำ แต่เนื่องจากแคชตำแหน่งที่ 2 บรรจุคำสั่งจากหน่วยความจำตำแหน่งที่ 2 ดังนั้น ทำให้ต้องนำคำสั่ง ADD R1, R2, R1 จากหน่วยความจำตำแหน่งที่ 10 ไปเขียนในแคชตำแหน่งที่ 2 แทนและนำไปถอดรหัสต่อไป

(f) คำสั่ง MOV PC, LR จะต้องถูกเฟล์ชจากหน่วยความจำ แต่เนื่องจากแคชตำแหน่งที่ 3 ว่างเปล่า ทำให้ต้องนำคำสั่งนี้ไปเก็บในแคชในตำแหน่งที่ 3 และนำไปถอดรหัสต่อไป หลังจากนั้นจะต้องกระโดด

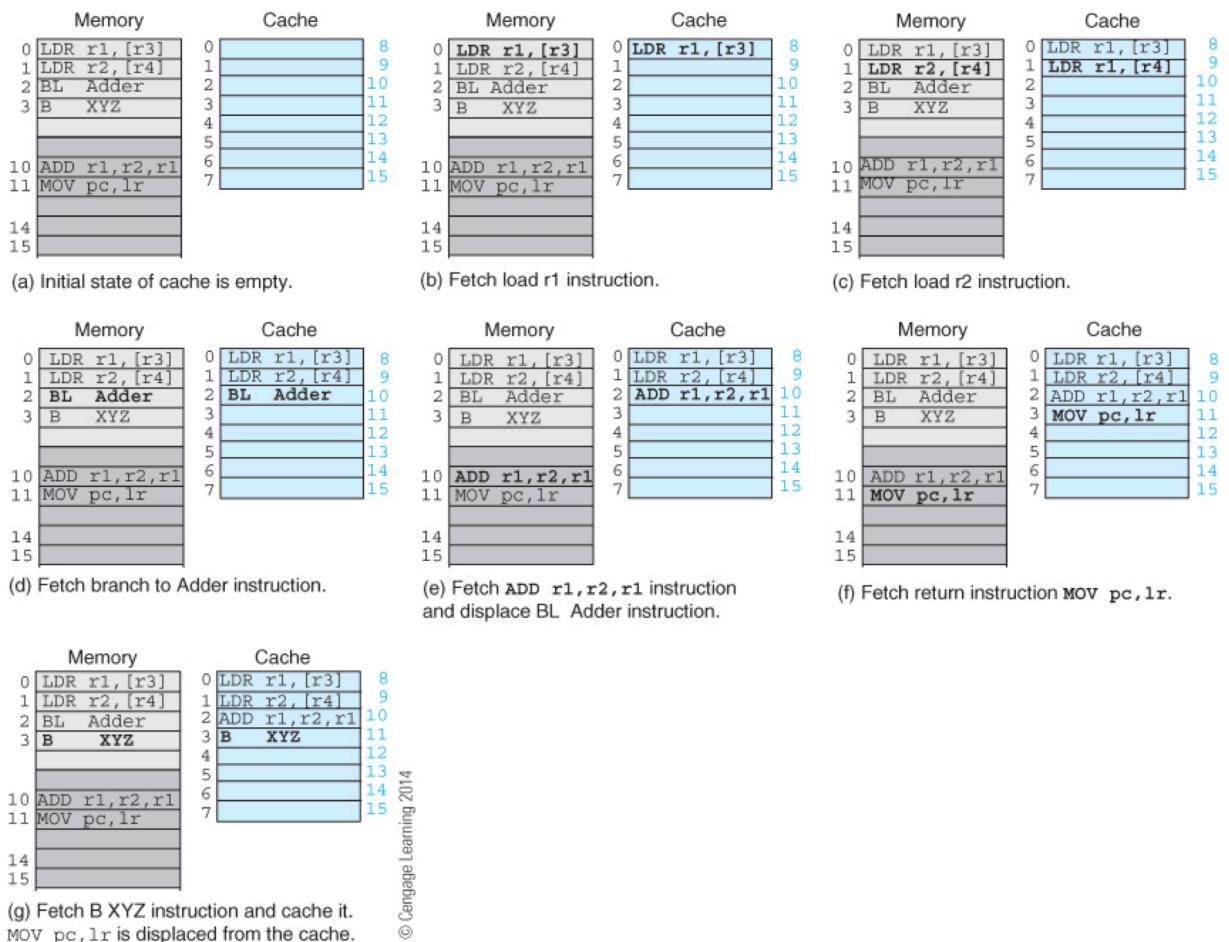


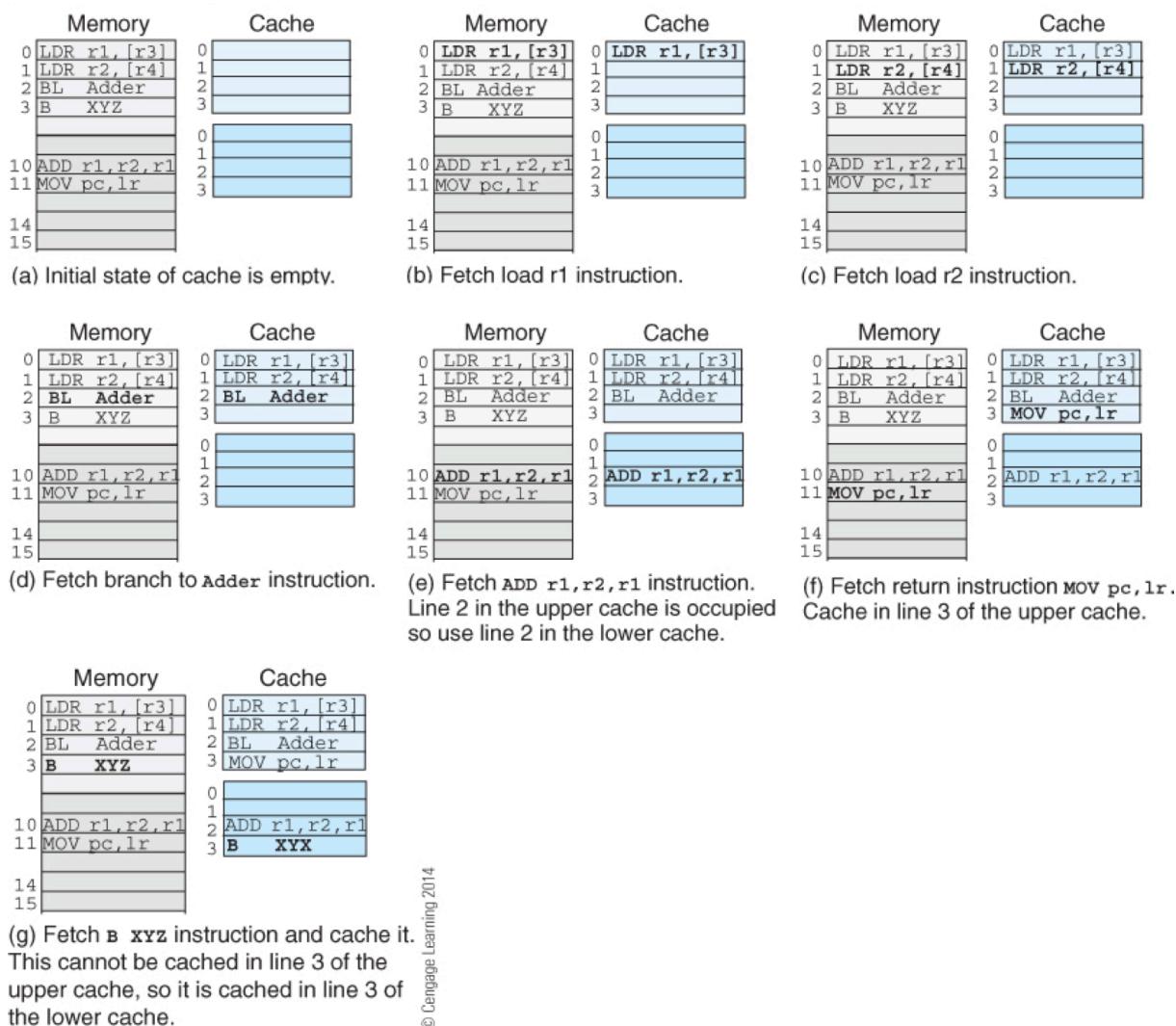
Figure 5.12: การทำงานของแคช L1 แคชคำสั่ง (Instruction Cache) ชนิด Direct Map ที่มา: [Clements \(2013\)](#)

กลับ (Return) ไปยังหน่วยความจำตำแหน่งที่ 3 ต่อไป นั่นคือคำสั่ง B XYZ

(g) คำสั่ง B XYZ จะต้องถูกเฟลชจากหน่วยความจำ แต่เนื่องจากแคชคำสั่งตำแหน่งที่ 3 บรรจุคำสั่ง MOV PC, LR ดังนั้น ทำให้ต้องนำคำสั่ง B XYZ จากหน่วยความจำตำแหน่งที่ 3 ไปเขียนในแคชตำแหน่งที่ 3 แทน และนำไปอุดรหัสต่อไป

ผู้อ่านจะสังเกตเห็นว่า แคชตำแหน่งใดๆ จะเป็นที่พักเก็บคำสั่งในหน่วยความจำ 2 ตำแหน่งเสมอ เช่น แคชตำแหน่งที่ 0 จะเป็นเป้าหมายของหน่วยความจำตำแหน่งที่ 0 และ 8 เป็นต้น ยิ่งไปกว่านั้น แคชจริง มีความจุน้อยกว่าหน่วยความจำหลักจริงมาก เนื่องจากแคชอาศัยโครงสร้างพื้นฐานของสแตติกแรม จึงใช้พื้นที่บนชิปมากกว่าต่อความจุ 1 บิตเท่ากัน ทำให้เกิดการแย่งกัน (Contention) เป็นแบบ Many to One ดังนั้นเมื่อแคชมีขนาดเล็ก อัตราการแย่ง (Contention Rate) จะยิ่งเพิ่มสูงมากขึ้น

5.4.2 แคชชนิดเซ็ตแอลโซซิเอทีฟ (Set Associative Cache)



© Cengage Learning 2014

Figure 5.13: การทำงานของแคช L1 แคชคำสั่ง (Instruction Cache) ชนิด Set Associative ที่มา: (Clements (2013))

รูปที่ 5.13 การทำงานของแคชชนิด Set Associative Cache นิยมใช้ออกแบบแคชlevel 2 (Level 2) และlevel 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 1 ว่างเปล่าทั้งบนและล่าง ทำให้ต้องนำคำสั่งนี้จากหน่วยความจำไปเก็บในแคชในตำแหน่งที่ 1 ของแคชก้อนบน และนำไปถอดรหัสต่อไป

(d) คำสั่ง BL Adder จะต้องถูกเฟล์ชจากหน่วยความจำ แต่เนื่องจากแคชตำแหน่งที่ 2 ว่างเปล่าทั้งสองก้อน ทำให้ต้องนำคำสั่งนี้จากหน่วยความจำไปเก็บในแคชในตำแหน่งที่ 2 ของก้อนบน และนำไปถอดรหัสต่อไป โดยจะต้องกระโดดไปยังหน่วยความจำตำแหน่งที่ 10 ต่อไป

(e) คำสั่ง ADD R1, R2, R1 จะต้องถูกเฟทซ์จากหน่วยความจำ แต่เนื่องจากแคชตำแหน่งที่ 2 ของก้อนบันบรรจุคำสั่งจากหน่วยความจำตำแหน่งที่ 2 แต่แคชก้อนล่างยังว่างอยู่ ดังนั้น ระบบจึงนำคำสั่ง ADD R1, R2, R1 จากหน่วยความจำตำแหน่งที่ 10 ไปเขียนในแคชตำแหน่งที่ 2 ของแคชก้อนล่างแทนและนำไปถอดรหัสต่อไป

(f) คำสั่ง MOV PC, LR จะต้องถูกเฟทซ์จากหน่วยความจำ แต่เนื่องจากแคชตำแหน่งที่ 3 ของก้อนล่างยังว่างเปล่า จึงนำคำสั่งนี้ไปเก็บในแคชในตำแหน่งที่ 3 ของก้อนล่างและนำไปถอดรหัสต่อไป หลังจากนั้นจะต้องกระโดยกลับ (Return) ไปยังหน่วยความจำตำแหน่งที่ 3 ต่อไป นั่นคือคำสั่ง B XYZ

(g) คำสั่ง B XYZ จะต้องถูกเฟทซ์จากหน่วยความจำ แต่เนื่องจากแคชตำแหน่งที่ 3 ของก้อนบันบรรจุคำสั่ง MOV PC, LR และตำแหน่งที่ 3 ของก้อนล่างยังว่างเปล่า ดังนั้น จึงนำคำสั่ง B XYZ จากหน่วยความจำตำแหน่งที่ 3 ไปเขียนในแคชตำแหน่งที่ 3 ของก้อนล่างและนำไปถอดรหัสต่อไป

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

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

5.5 หลักการหน่วยความจำเสมือนชนิดเพจ (Paging Virtual Memory)

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

- เพื่อรองรับความต้องการความจุของหน่วยความจำหลักที่เพิ่มสูงขึ้น เนื่องจากระบบปฏิบัติการ 64 บิต ที่สามารถอ้างอิงหน่วยความจำหลักได้เกิน 4 GB
- เพื่อบริหารจัดการความเร็วที่แตกต่างระหว่างหน่วยความจำหลักและหน่วยสำรองข้อมูล ถึงแม้หน่วยความจำแฟลชจะมีความเร็วสูงขึ้นและพร่าหอยามากขึ้น

การทำงานของแต่ละโปรแกรม ให้มีหน่วยความจำเสมือนของตนเอง ดังรูปที่ 5.15 ข้อดี คือ โปรแกรมของผู้ใช้สามารถมีขนาดใหญ่กว่าหน่วยความจำจริงได้ทำให้ผู้เขียนโปรแกรมทำการเขียน โปรแกรมได้อย่างอิสระมากขึ้น ไม่ต้องกังวลถึงขนาดของหน่วยความจำ การเข้ามต่อ กับหน่วยความจำภายในภาพ รูปที่ 5.14 หน่วยความจำเสมือนจะถูกแบ่งเป็นเพจ หรือ พื้นที่ขนาด 1 ถึง 8 กิโลไบท์ ขึ้นอยู่กับฮาร์ดแวร์และโอเอส ที่จะตกลงกัน พื้นที่เหล่านี้จะแบ่งเป็นส่วนที่เก็บคำสั่ง (Text Segment) ส่วนเก็บตัวแปร (Data Segment) ส่วนสำหรับสร้างสเต็ค (Stack) และ ไฮพ์ (Heap) ในพื้นที่ส่วนที่เหลือเรียกว่า สเต็คเช็คเม้นท์ (Stack Segment) ไฮพ์ คือ พื้นที่สำหรับจองพื้นที่ตัวแปรในขณะที่โปรแกรมกำลังทำงาน เช่น ตัวแปรโครงสร้างข้อมูล (Data Structure) ชนิด Linked List, Tree, Graph และอื่นๆ

เมื่อโอเอสได้จองพื้นที่ขนาดเท่ากับหนึ่งเพจในหน่วยความจำภายในภาพ เรียกว่า เพรม (Frame) และทำการแมป (Map) หรือ แปลงเลขเพจเสมือน (Virtual Page Number) ให้ตรงกับเลขเฟรมภายในภาพ (Physical Frame Number) สังเกตได้จากเฟรมที่เป็นสีม่วงซึ่งมาจากprocessorเดียวกันนี้ ตารางการแมปนี้ ระบบปฏิบัติการเรียกว่า ตารางเพจ (Page Table) ตารางเพจ คือ ตารางสำหรับจดบันทึกความสัมพันธ์ระหว่างเลขเพจเสมือน กับ เลขเฟรม ต้องอยู่ริเวณส่วนของ Kernel Space ของหน่วยความจำภายในภาพ

ตามรูปที่ 5.14 ซึ่งที่เป็นสีชมพูในหน่วยความจำภายในภาพ คือ เพรมที่ว่างอยู่ หรือ ถูกจับจองโดยprocessor อื่นๆ ด้วย ซึ่งโอเอสที่รองรับการทำงานแบบมัลติโปรแกรมมิ่ง (Multi Programming) ในปัจจุบัน เปิดโอกาสให้ processor หลายๆ ตัวใช้งานหน่วยความจำภายในภาพได้พร้อมๆ กัน ส่วนเฟรมซึ่งที่เป็นสีขาว หมายถึง เพรมที่ยังว่างอยู่ หมายเหตุ เส้นประ คือ รอยต่อระหว่างเพจของหน่วยความจำเสมือน

เนื่องจากหน่วยความจำหลัก เช่น ROM/RAM ยังมีต้นทุน (ต่อพื้นที่หนึ่งหน่วย) สูงกว่าเมื่อเทียบกับหน่วยสำรองข้อมูล จึงจำเป็นต้องใช้หน่วยความจำหลักทำหน้าที่เป็นแคชของหน่วยสำรองข้อมูล การอ่านหน่วยความจำสำรอง จำเป็นต้องอ่านหรือเขียนครั้งละมากๆ เนื่องจากเวลาหน่วง (Latency) ที่ต้องรอเพื่อ โอนย้าย (อ่านหรือเขียน) ข้อมูลหรือคำสั่ง ระหว่างหน่วยความจำหลัก และหน่วยความจำสำรอง รายละเอียดสามารถอ่านเพิ่มเติมในบทที่ 7 ดังนั้น การแบ่งหน่วยความจำเสมือนและหน่วยความจำหลัก ออกเป็นเพจ จึงเป็นเรื่องที่เหมาะสม และง่ายต่อการบริหารจัดการ

ARM Cortex A7 เป็นต้นแบบการพัฒนาของ Cortex A53 ซึ่งทั้งคู่จัดเป็นชั้นระดับต้น (Entry Level) สำหรับโทรศัพท์เคลื่อนที่สมาร์ทโฟน ลิงค์สำหรับที่มาของรูปที่ 5.15 ประกอบด้วยการทำงานร่วมกันของ

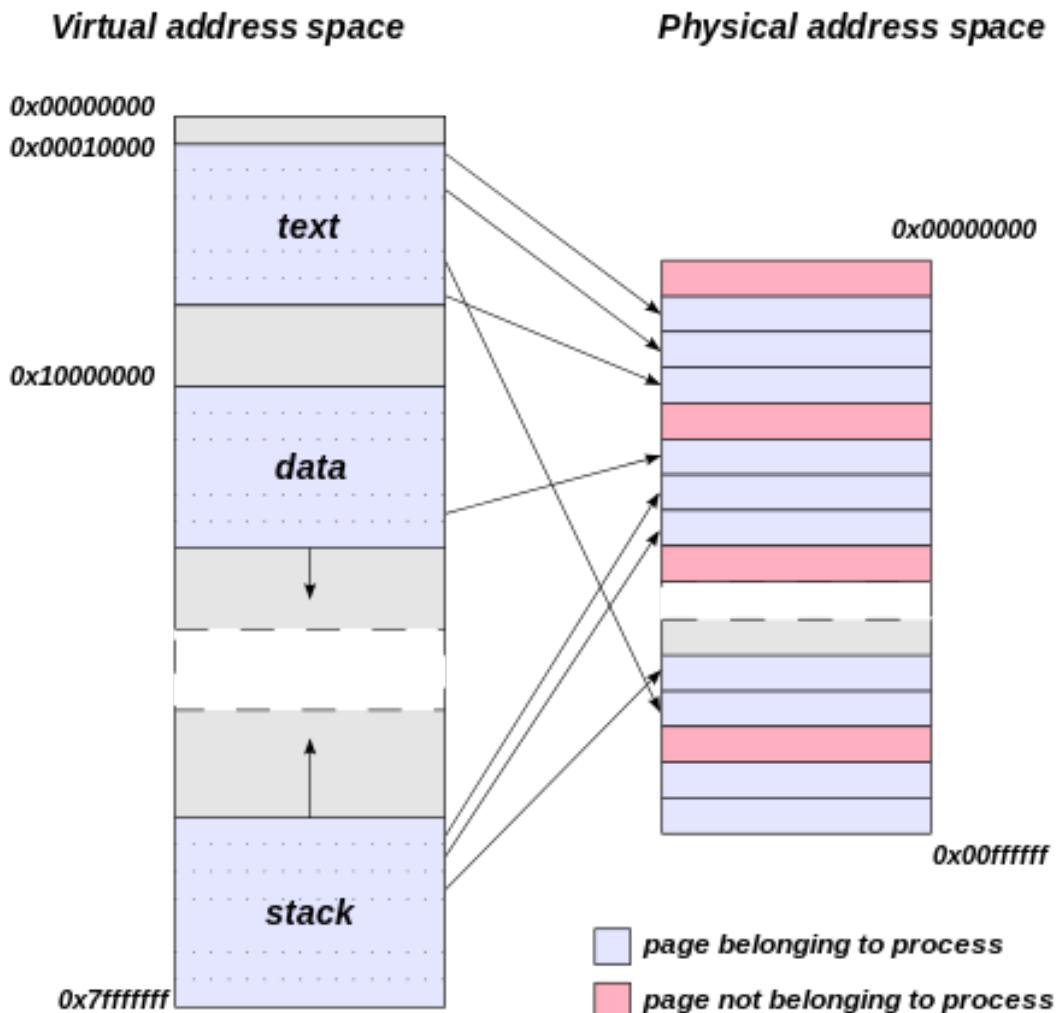


Figure 5.14: การแมป (Map) แอดเดรสเสมือน (Virtual Address) ขนาด 2GB เป็นแอดเดรஸกายภาพ (Physical Address) ขนาด 64MB ตามหลักการหน่วยความจำเสมือนชนิดเพจ (Paging Virtual Memory หมายเหตุ เส้นประ คือ รอยต่อระหว่างเพจของหน่วยความจำเสมือน) ที่มา: http://www.wikiwand.com/en/Page_table

หน่วยความจำหลายชนิด รูปที่ 5.15 ที่มา: การแปลง Virtual Address เป็น Physical Address จะต้องอาศัยตารางเพจ (Page Table) ในการเก็บหมายเลขเพจเสมือน (Virtual Page Number) และหมายเลขเพจกายภาพ (Physical Page Number) โดยมี TLB ทำหน้าที่เป็นแคชให้กับหน่วยความจำบริเวณ ตารางเพจ (Page Table) ที่แอลบี (TLB: Translation Look Aside Buffer) ซึ่งแบ่งเป็น I (Instruction) TLB และ D (Data) TLB ซึ่งใช้เทคโนโลยีสแตติคแรมเช่นกัน

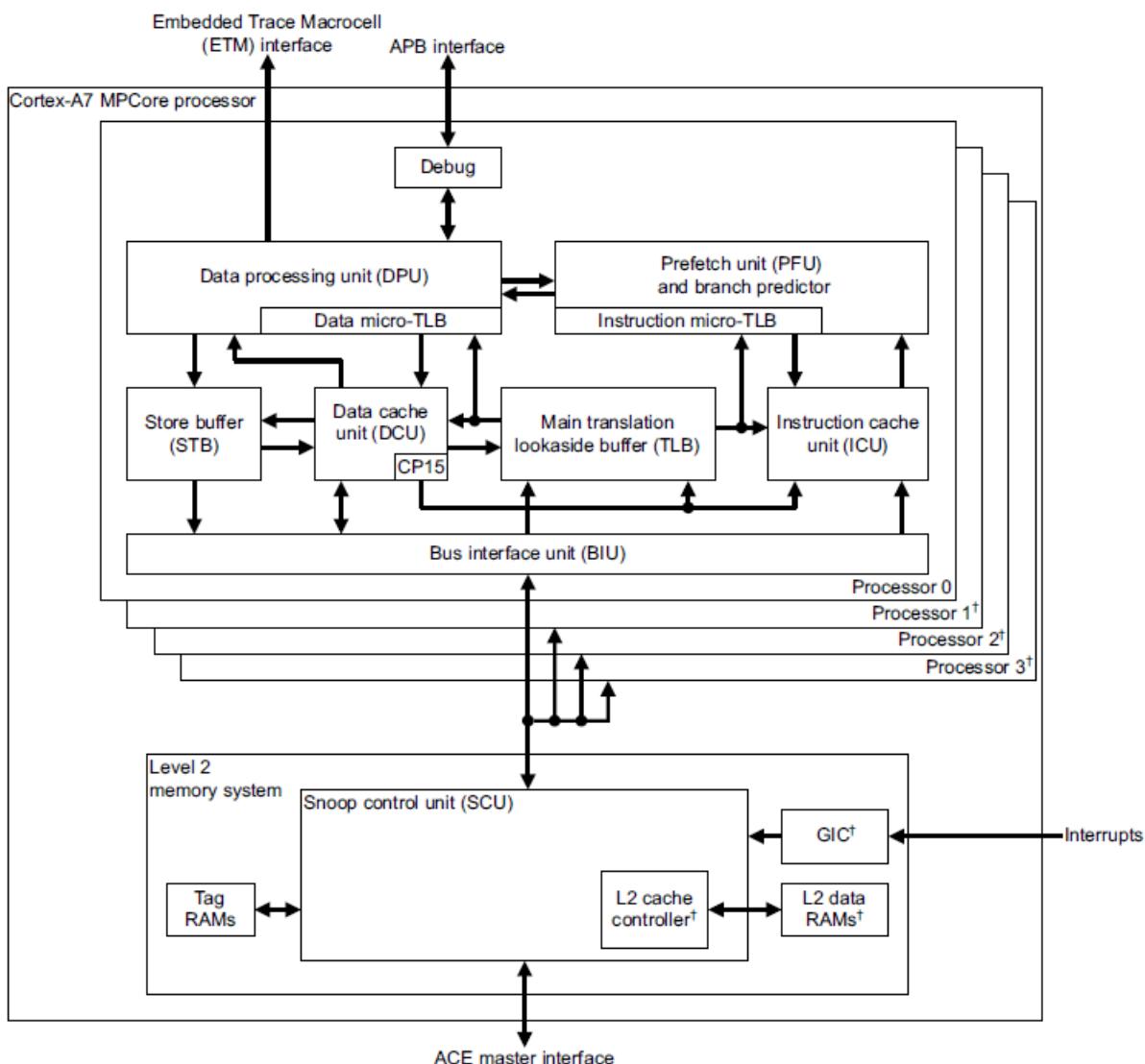


Figure 5.15: โครงสร้างภายในของชิป ARM Cortex A7 ประกอบด้วย TLB และ แคช หลายระดับเพื่อรองรับการทำงานของหน่วยความจำเสมือน (Virtual Memory)

การทำงานของระบบหน่วยความจำโดยรวมซึ่งประกอบด้วย Instruction micro-TLB, Data micro-TLB, Main TLB, L1 Instruction Cache Unit (ICU), L1 DCache, L2 Cache, L3 Cache และหน่วยความจำหลัก DRAM ลำดับการทำงานของรูปที่ 5.15 เป็นการแสดงลำดับการอ่าน (Fetch) คำสั่งเพื่อนำไป Decode/Execute ในไปป์ไลน์ ดังนี้

1. นำค่าแอดเดรสเสมือนใน PC (Program Counter) ส่วนที่เป็น Virtual Page# ไป สืบคันหาค่าเลข เพจฟิสิกัล (Physical Page#) ใน Instruction micro-TLB
2. นำค่า Physical Page# จะนำไปตรวจเช็คค่า Tag ว่าตรงกันหรือไม่ ถ้าตรงกันคือ TLB ฮิต (Hit) ถ้าไม่ตรงกัน คือ TLB มิส (Miss) และจึงนำหมายเลข Virtual Page# ไปสืบคันใน Main TLB ต่อไป หากมิสอีกรอบ จะต้องสืบคันในตารางเพจเป็นลำดับสุดท้าย
3. นำค่า Virtual Page Offset จาก PC มาเชื่อมกับ Physical Page# เพื่อนำไปสืบคันคำสั่งใน ICU หากพบว่าตรงกันเรียกว่า เกิดแคชฮิตที่ L1 (L1 Cache Hit) และนำคำสั่งส่งกลับไปยังวงจร Fetch

ต่อไป หากพบว่าไม่ตรงกันเรียกว่า เกิดแคชเมิสที่ L1 (L1 Cache Miss)

4. นำแอดเดรสภายในไปค้นหาคำสั่งในแคช L2 หากพบว่าตรงกันเรียกว่า เกิดแคชฮิตที่ L2 (L2 Cache Hit) แล้วนำคำสั่งกลับไปยังวงจร Fetch ต่อไป หากพบว่าไม่ตรงกันเรียกว่า เกิดแคชเมิสที่ L2 (L2 Cache Miss)
5. หากระบบมีแคช L3 จะนำแอดเดรสภายในไปค้นหาคำสั่งในแคช L3 หากระบบไม่มีแคช L3 จะนำแอดเดรสภายในไปค้นหาคำสั่งในหน่วยความจำหลัก (DRAM) ผ่านทาง Memory Interface ต่อไป
6. หน่วยความจำหลักได้รับแอดเดรสภายใน แล้วนำส่งคำสั่งไปให้แคชเลเวลต่างๆ และวงจร Fetch เป็นเลขฐานสองจำนวน 4-8 คำสั่งเป็นความยาว 128-256 บิต ซึ่งอยู่กับความกว้างของแคชแต่ละบล็อก

5.6 หน่วยความจำเสมือนของ Raspberry Pi3

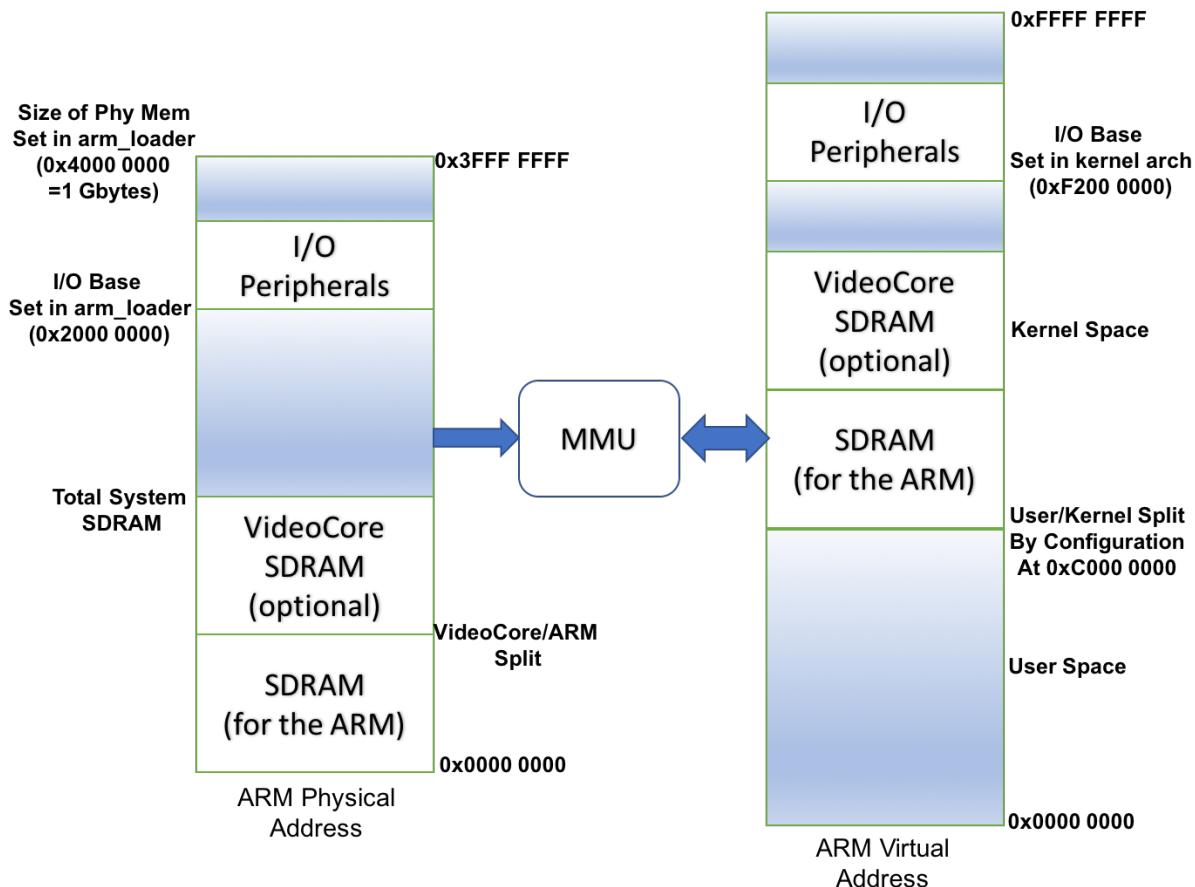


Figure 5.16: โครงสร้างของหน่วยความจำเสมือน (Virtual Memory) ของบอร์ด Pi3 โมเดล B ซึ่งใช้ชิปตระกูล BCM283x, x=5, 6, 7 หมายเหตุ ขนาดของรูปไม่เป็นไปตามพื้นที่ตามความเป็นจริง, MMU: Memory Management Unit

การแปลงและเดรสเสมือนของชิปตระกูล BCM2835-BCM2837 จะเหมือนกัน ในรูปที่ 5.11 โดยมี MMU (Memnory Management Unit) ทำหน้าที่แปลง (Address Translation) และเดรสเสมือนเป็น แอ็ดเดรสภายในภาพ ซึ่งได้แก้ไขรายละเอียดไปแล้วในหัวข้อก่อนหน้า รูปที่ 5.16 การแปลงโดยละเอียด พื้นที่ของหน่วยความจำเสมือนทั้งหมด 4 จิกะไบท์ จะแบ่งเป็น 1 จิกะไบท์สำหรับเก็บคำสั่งและข้อมูลของ เครื่อง(Linux) ในการบริหารจัดการเครื่อง เรียกว่า Kernel Space และเดรสเสมือนของ Kernel เริ่มต้นที่หมายเลข 0xC000 0000 มีพื้นที่ขนาด 1 GB โดยประมาณ โดยพื้นที่ 1 จิกะไบท์นี้ procress ทุกตัว จะเห็นเป็นพื้นที่เดียวกัน และ 3 จิกะไบท์สำหรับเก็บคำสั่งและข้อมูลของprocress แต่ละตัว เรียกว่า User Space ซึ่งคล้ายกับรูปที่ 3.12 และเดรสเสมือนของ User เริ่มต้นจากหมายเลข 0x0000 0000 จนถึง 0xBFFF FFFF. ขนาดเท่ากับ 3 GB

ในพื้นที่ของ Kernel หน่วยความจำของ ARM และของ GPU หรือ VideoCore จะถูกกำหนดโดยไฟล์ start.elf ใน บูตพาร์ติชั่นซึ่งฟอร์แมตด้วย FAT32 โดยหน่วยความจำที่กำหนดให้ GPU ขั้นต่ำที่สุด ขนาด 32 เมกะไบท์ แต่ความละเอียด 1080p30 ต้องการหน่วยความจำมากกว่านี้ สำหรับหน้าที่เป็นบัฟเฟอร์ สำหรับแสดงผล

แอดเดรสภายในภาพเริ่มต้นที่หมายเลข 0x0000 0000 โดยแบ่งออกเป็น

- พื้นที่สำหรับการแสดงผล ด้วยพื้นที่ขั้นต่ำ 32 MB
- พื้นที่สำหรับเชื่อมต่อ กับ อุปกรณ์อินพุตเอาท์พุต ซึ่งไม่ต้องใช้งานร่วมกับแคช (Uncached) starting at 0xC000 0000.

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

5.7 สรุป

หน่วยความจำลำดับชั้นอาศัยการบริหารรวมกันของหน่วยความจำหลายชนิดและหลายขนาดเข้าด้วยกันเนื่องจากมีเทคโนโลยีที่แตกต่างกัน หน่วยความจำขนาดเล็กความเร็วสูง เช่น SRAM มาใช้งานเป็นรีจิสเตอร์ และ แคช ทำหน้าเป็นตัวแทนของหน่วยความจำที่จุมากกว่าแต่ความเร็วต่ำกว่า เช่น DRAM มาใช้งานเป็นหน่วยความจำหลัก และใช้เทคโนโลยีฮาร์ดดิสก์และแฟลชเป็นหน่วยสำรองข้อมูล เพื่อให้คอมพิวเตอร์มีความจุเพียงพอและทำงานโดยเฉลี่ยได้รวดเร็วขึ้นต่อความต้องการใช้งานระบบ โดยผ่านจุดเด่นของหน่วยความจำแต่ละชนิดเข้าด้วยกัน และช่วยประหยัดต้นทุนของระบบ

หลักการของแคชมีบทบาทต่อประสิทธิภาพของระบบโดยองค์รวม แคชชนิด N-way Set Associative ได้รับความนิยม เนื่องจากมีโครงสร้างเหมือนกับแคชชนิด Direct Mapped และมีความสามารถคล้ายแคชชนิด Fully Associative มีการนำหลักการ Principle of Locality มาประยุกต์ใช้กับหน่วยความจำประเภทต่างๆ และหลากหลาย หนึ่งในนั้น คือ หน่วยความจำเสมือน นั่นเอง หน่วยความจำเสมือนอาศัยการทำงานร่วมกันระหว่างฮาร์ดแวร์และระบบปฏิบัติการ เพื่อเพิ่มลำดับชั้นในการบริหารจัดการหน่วยความจำทุกลำดับชั้นดังได้กล่าวไปแล้ว

เนื้อหาในบทนี้ว่าด้วยเรื่องของหน่วยความจำผนวกกับความรู้ในบทก่อนหน้า ผู้เขียนจึงขออุปมาอุปมาภัยการทำงานของคอมพิวเตอร์โดยรวม ดังนี้

- ซอฟต์แวร์ คือ สูตรหรือขั้นตอนการประกอบอาหารอย่างละเอียด เพื่อให้ผู้ครัวประกอบอาหารให้สุกและมีรสชาติดี
- ชีพิญ คือ พ่อครัวและเครื่องครัวที่ประกอบอาหารตามสูตรเท่านั้น ประกอบด้วย
 - ALU คือ อุปกรณ์เครื่องครัว เช่น มีด เครื่อง กระทะ หม้อ เป็นต้น
 - Register คือ งานหรือที่พักอาหารที่ปูรุกค้าง เพื่อรอทำให้เป็นอาหารที่ปูรุกสำเร็จต่อไป
 - วงจรอื่นๆ
- หน่วยความจำ คือ ชั้นวางวัตถุดิบ (ข้อมูล) ขนาดเล็ก ไม่สามารถเก็บวัตถุดิบได้นาน
- ข้อมูลดิบ คือ วัตถุดิบสำหรับประกอบอาหาร อาจมาจากแหล่งข้อมูลหลายๆ แหล่ง เช่น ผู้ใช้กรอกข้อมูลในโปรแกรม ไฟล์ข้อมูลในหน่วยสำรอง เป็นต้น

Chapter 5. ลำดับชั้นหน่วยความจำ (Memory Hierarchy)

- ข้อมูลหรือสารสนเทศ คือ อาหารที่ประกอบและปรุงสำเร็จแล้ว โดยอาศัยซีพียูและซอฟต์แวร์ที่กล่าวมาข้างต้น

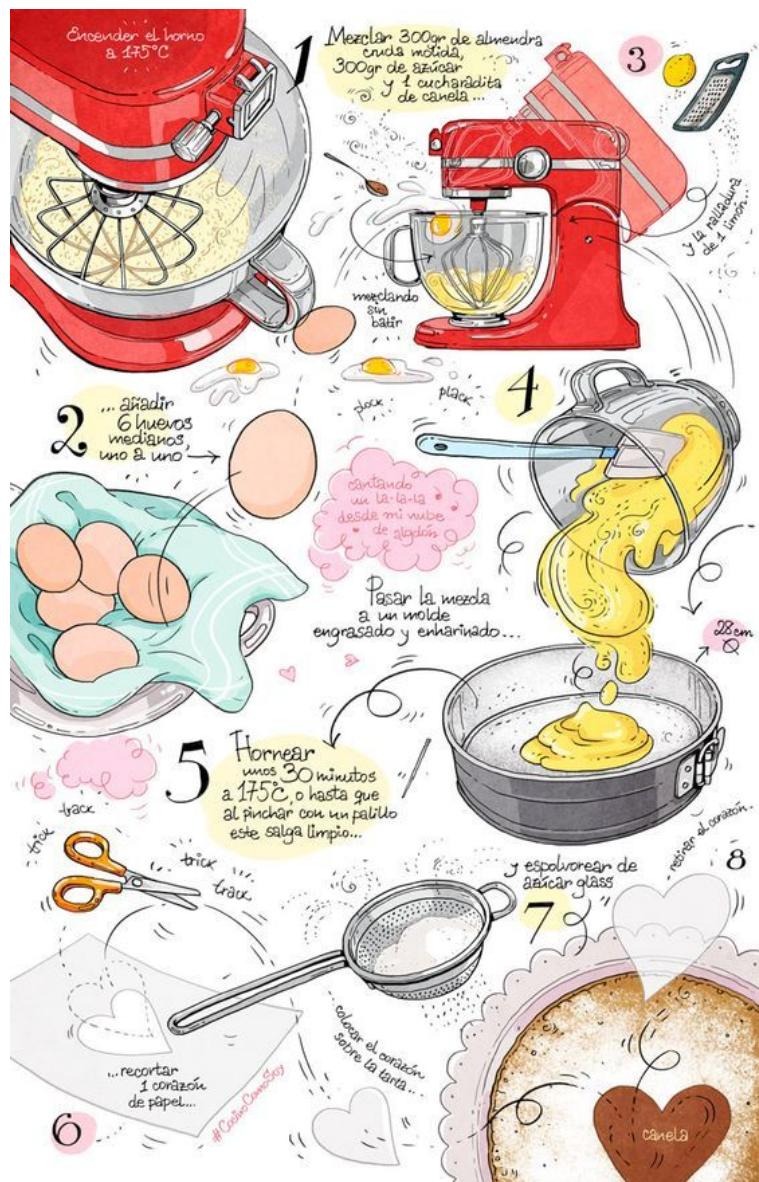


Figure 5.17: การ ประกอบ อาหาร ตาม สูตร ที่มา: <https://www.pinterest.com/pin/85779567881855495/>

- หน่วยสำรองข้อมูล คือ ตู้แข็งเย็นขนาดใหญ่ที่สามารถบรรจุวัตถุติดไฟได้ปริมาณมาก และระยะเวลานาน หากต้องการเก็บข้อมูลหรือวัตถุติดไฟไว้ทานต่อไป ข้อมูลติดไฟจะมีปริมาณมากๆ วัตถุติดไฟเหล่านี้จะเก็บในรูปของไฟล์ข้อมูล ซอฟต์แวร์สามารถอ่านหรือเขียนข้อมูลจากไฟล์ที่มีลักษณะต้องการ
- หากวัตถุติดไฟมาจากต่างประเทศ จะมาจากเซิร์ฟเวอร์อื่นๆ บนเครือข่ายอินเตอร์เน็ต ก่อนจะนำมาปรุง ข้อมูลติดไฟเหล่านี้จะถูกอ่านขึ้นมาเก็บอยู่ในหน่วยความจำหลัก ในรูปของไฟล์ข้อมูล หรือ ในรูปของข้อมูลที่จะจัดระจัดตามทางเซิร์ฟต่างๆ ด้วยเทคโนโลยี IoT (Internet of Thing)
- หากข้อมูลมีปริมาณมากๆ การประมวลผลข้อมูลจำเป็นต้องอาศัยซีพียูและ/หรือจีพียุหลายๆ คอร์ เพื่อร่วมประมวลผลแบบขนาน (Parallel Computing) ทำให้การประมวลผลเสร็จสิ้นรวดเร็วขึ้น

Chapter 6

อินพุท/เอาท์พุท (Input/Output)

วัตถุประสงค์

- เพื่อให้รู้จักโครงสร้างและการทำงานด้านอินพุท/เอาท์พุทของบอร์ด Pi3
- เพื่อให้เข้าใจสัญญาณการเชื่อมต่อกับอุปกรณ์อินพุท/เอาท์พุทนิดต่างๆ
- เพื่อให้เข้าใจกลไกการติดต่อกับอุปกรณ์อินพุท/เอาท์พุทนิดต่างๆ
- เพื่อให้เข้าใจหลักการ Memory Mapped I/O, Interrupt และ Direct Memory Access

ในเครื่องคอมพิวเตอร์ที่มีระบบปฏิบัติการ OS จะมีหน้าที่บริหารจัดการอุปกรณ์อินพุท/เอาท์พุท เพื่อให้โปรแกรมต่างๆสามารถใช้ทรัพยากรเหล่านี้ร่วมกัน โดยจะต้องมีการป้องกันและการจัดลำดับการใช้งาน (Protection and Scheduling) เนื่องจากการใช้งานอุปกรณ์อินพุท/เอาท์พุท ทำให้เกิดการอินเทอร์รัพท์ (Interrupt) แปลว่า ขัดจังหวะการทำงานของซีพียู ระบบปฏิบัติการจะต้องจัดเตรียมวิธีการเชื่อมต่อกับอุปกรณ์อินพุท/เอาท์พุท โดยมีฮาร์ดแวร์ที่จะควบคุมการทำงานของอุปกรณ์ต่างๆ แทนโดยตรง เพื่อให้ CPU ทำงานที่สำคัญได้อย่างต่อเนื่องโดยเน้นที่การประมวลผลข้อมูล ซึ่งวิธีนี้จะเพิ่มประสิทธิภาพและการตอบสนองต่อผู้ใช้โดยรวมได้ และเพื่อให้โปรแกรมเมอร์สามารถพัฒนาโปรแกรมได้อย่างรวดเร็ว มีประสิทธิภาพและปลอดภัย

การติดต่ออินพุท/เอาท์พุทโดยรวม เพิ่มความสามารถให้กับระบบคอมพิวเตอร์ มากขึ้นไปอีก เช่น การเชื่อมต่อกับเครือข่ายอินเตอร์เน็ต ผ่านทางเครือข่ายมีสายและเครือข่ายไร้สาย การเชื่อมต่อกับผู้ใช้ ผ่านทางปุ่มกด หน้าจอสัมผัส การขับ (Motion) / การเอียง (Tilt) / ความเร่ง (Acceleration) ของการเคลื่อนไหว เป็นต้น ดังนั้น การเชื่อมต่อกับอุปกรณ์เหล่านี้จำเป็นต้องทำตามมาตรฐาน เพื่อลดความยุ่งยากในการออกแบบและตั้งค่าโดยรวมของคอมพิวเตอร์

รูปที่ 3.2 ในบทที่ 3 ได้นำเสนอการเชื่อมโยงอุปกรณ์ต่างๆ บนบอร์ด Pi3 โดยมีชิป BCM2837 เป็นศูนย์กลาง แต่ละอุปกรณ์เป็น device แบบไหน Char หรือ Block memory mapped I/O ในบทที่ 5 Peripheral I/O อยู่ในพื้นที่ของ OS BCM เป็น SoC ในรูปที่ 3.3 ประกอบด้วย ARM Cortex A53 เชื่อมต่อกับโลกภายนอก ภายในคือ ARM Cortex A53 จำนวน 4 คอร์ ดังรูปที่ ตามรายละเอียดเบื้องต้นในบทที่ 3 บทนี้จะเน้นเรื่องของการเชื่อมต่อ BCM2837 หรือ CM3 (Compute Module 3) ซึ่งมีขาทั้งหมด 200

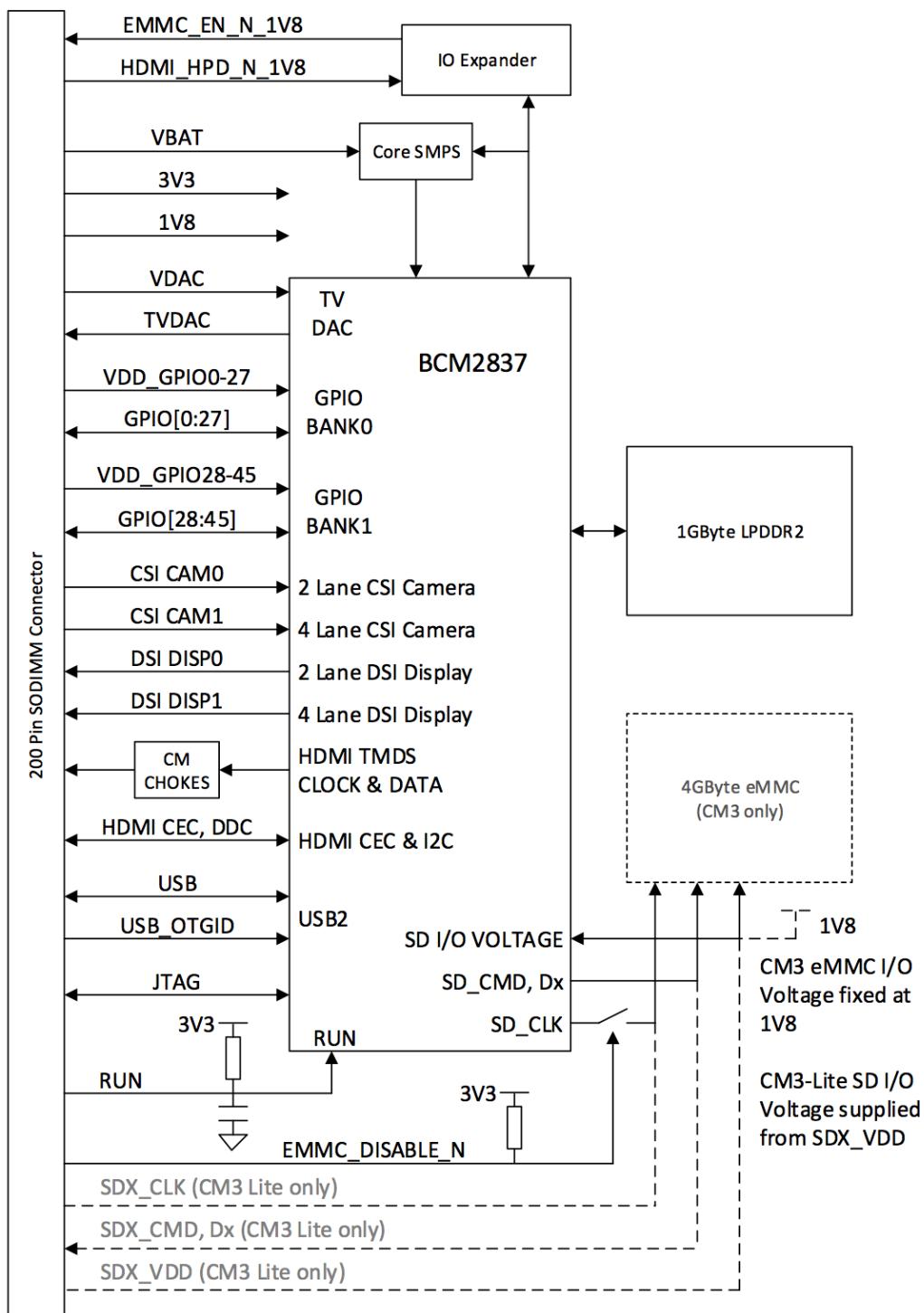


Figure 6.1: การเชื่อมโยงอุปกรณ์ต่างๆ บนบอร์ด Pi3 โดยเมจิพ BCM2837 หรือ CM3 (Compute Module 3) เป็นศูนย์กลางด้วยขาจำนวน 200 ขา ที่มา: ?

หากับอุปกรณ์ภายนอกอื่นๆ รูปที่ 6.1 เช่น HDMI, USB และอุปกรณ์สำรองข้อมูล กลไกการเชื่อมต่อที่ลึกลงไป จะปรากฏในการทดลอง การทดลองที่ 7 ภาคผนวก G การศึกษาและปรับแก้อินพุท/เอาท์พุทต่างๆ

6.1 สัญญาณ HDMI สำหรับจอ LCD ขนาดใหญ่

เนื้อหาในหัวข้อนี้ต่อเนื่องจาก HDMI (High-Definition Multimedia Interface) ในหัวข้อที่ 3.1.3 สำหรับ เชื่อมต่อจอแสดงผลภายนอก HDMI คือ สัญญาณสำหรับการเชื่อมต่ออุปกรณ์ภาพและเสียง ซึ่งเข้ามาแทนที่การเชื่อมต่อรูปแบบเดิมๆ เช่น สัญญาณคอมโพสิตวิดีโอ และแบบ S-video

การเชื่อมต่อแบบ HDMI เป็นการถ่ายโอนสัญญาณแบบดิจิทัล สามารถส่งได้ทั้งสัญญาณภาพและสัญญาณเสียงไปพร้อมๆ กันด้วยอัตราบิตเตตสูง ระดับจิกะบิตต่อวินาที การเชื่อมต่อด้วยสัญญาณ HDMI หมายความว่าสำหรับการแสดงผลจากเครื่องคอมพิวเตอร์หรือเครื่องเล่นมีเดีย (Media Player) ไปยังจอภาพความละเอียดสูงระดับเอชดี (HD) หรือสูงกว่าสำหรับความละเอียด Ultra HD HDMI เวอร์ชันล่าสุดคือ 2.1 ซึ่งพัฒนาเพื่อรองรับวิดีโอที่ละเอียดสูงเฟรมละ 8000 เส้นที่ 60 เฟรมต่อวินาที (8K60) และเฟรมละ 4000 เส้นที่ 120 เฟรมต่อวินาที (4K120) และเพิ่มลีน์เฟรมละ 10,000 เส้น (10K) ซึ่งจะทำให้อัตราบิตเตตเพิ่มเป็น 48 จิกะบิตต่อวินาที

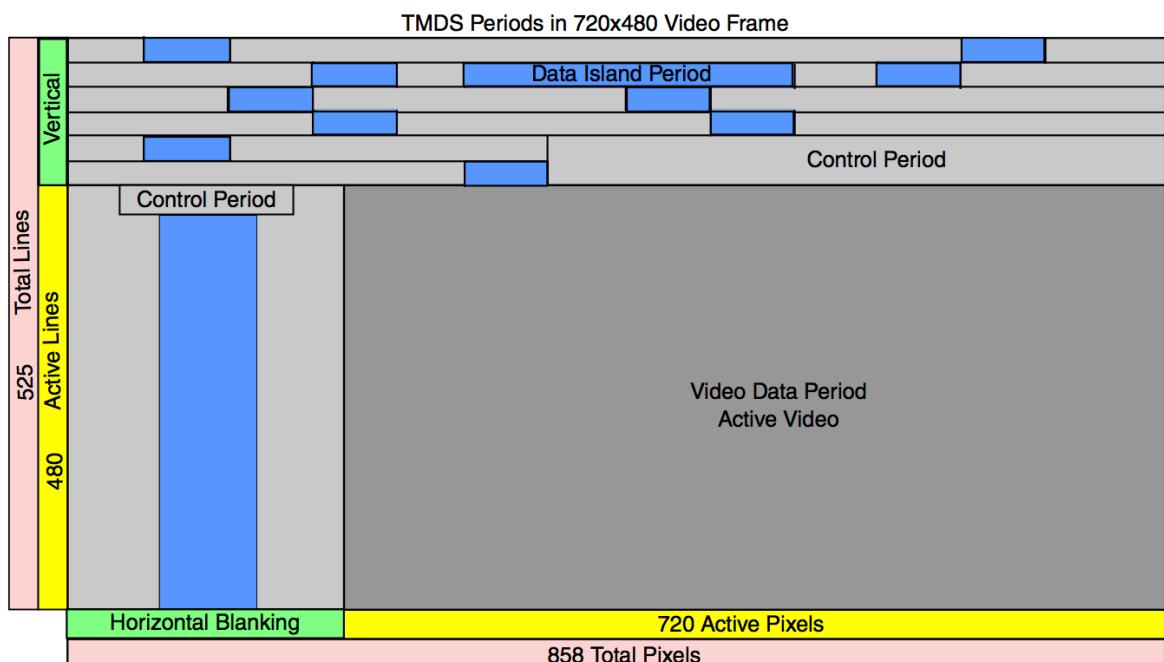


Figure 6.2: การส่งแพ็คเก็ตข้อมูล และควบคุมด้วยช่องสัญญาณ TMDS ในช่วงต่างๆ สำหรับความละเอียดในการแสดงผล 720x480 ต่อเฟรม ที่มา: <https://people.freebsd.org/~gonzo/arm/iMX6-HDMI.pdf>

สัญญาณ HDMI มีช่องสื่อสาร 5 ช่องแยกจากกัน ได้แก่

- ช่อง TMDS (Transition-Minimized Differential Signaling) ช่อง TMDS จะส่งข้อมูลภาพวิดีโอเสียง และข้อมูลเป็นดิจิทัล ในรูปของแพ็คเก็ต (Packet) แต่ละแพ็คเก็ตข้อมูล ประกอบด้วย ช่วงส่งข้อมูลภาพ (Video Data Period) และช่วงส่งแพ็คเก็ตควบคุม (Control Period) สำหรับสัญญาณควบคุม ยกตัวอย่างเช่น สัญญาณ HSYNC (Horizontal Synchronization) และ VSYNC (Vertical Synchronization) เป็นต้น

รูปที่ 6.2 แสดงตัวอย่างการส่งแพ็กเก็ตข้อมูลจะใช้สัญลักษณ์สีเทา และแพ็กเก็ตควบคุมจะใช้สัญลักษณ์สีฟ้า ในช่วงเวลาต่างๆ สำหรับ การแสดงภาพด้วยความละเอียด 720x480 ต่อ 1 เฟรมแต่ละเฟรม ใช้เวลา 33 มิลลิวินาทีเพื่อการแสดงผล หรือคิดเป็น 30 เฟรมต่อวินาที ภายในระยะเวลา 33 มิลลิวินาทีจะเกิดการส่งสัญญาณภาพเป็นจำนวน 525 เส้นๆ ละ 858 พิกเซลแต่แสดงผลให้เห็นเพียง 480 เส้นๆ ละ 720 พิกเซลในตำแหน่งที่มีสีเทาเข้มเท่านั้น ส่วนที่เกินเผื่อไว้สำหรับช่วงเวลา Vertical Blanking และ Horizontal Blanking ช่วงเวลาทั้งสองจะใช้ในการส่งสัญญาณควบคุมและข้อมูลอื่นๆ โดยแพ็กเก็ตข้อมูลสำหรับจุดภาพเส้น (Line) บนสุด ตำแหน่งจุดภาพซ้ายสุดไปจุดภาพขวาสุด การส่งแพ็กเก็ตจะขยับลงมา 1 เส้น แล้วเริ่มจากตำแหน่งจุดภาพซ้ายสุดไปจุดภาพขวาสุด เช่นเดิม การส่งแพ็กเก็ตจะขยับลงมาเรื่อยๆ จนถึงเส้นภาพสุดล่างสุด แล้วจึงเริ่มต้นภาพใหม่ด้วยเส้นภาพบน สุดเช่นเดิม

ในสาย HDMI หนึ่งเส้นประกอบด้วย LEN TMDS จำนวน 3 เลนสำหรับส่งแพ็กเก็ตข้อมูลพร้อมกัน รายละเอียดเพิ่มเติมที่ [หัวข้อ Transition Minimized Differential Signaling ของ Wikipedia](#)

- ช่อง DDC (Display Data Channel) ใช้สื่อสารกับเครื่อง Media Player ด้วยมาตรฐาน I2C เพื่อกำหนดรูปแบบของวีดีโอและเสียงที่ส่งไปยังจอภาพ และยังใช้คุ้มครองเนื้อหาดิจิทัลแบบดิวอร์ชูน (High-bandwidth Digital Content Protection: HDCP) เช่น ภาพนั้นที่มีลิขสิทธิ์ เป็นต้น
- ช่อง CEC (Consumer Electronics Control) ผู้ใช้งานสามารถควบคุมอุปกรณ์ต่อพ่วงผ่านช่อง CEC นี้ได้มากถึง 15 ตัว เป้าหมายคือ อุปกรณ์สามารถทำงานร่วมกันและใช้เครือข่ายอินเตอร์เน็ตไปพร้อมๆ กัน
- ช่อง ARC (Audio Return Channel) หรือช่องสัญญาณเสียงคืนกลับ เพื่อใช้สาย HDMI เชื่อมต่อกับตัว盒หรือหัวเสียงและเครื่องขยายเสียงผ่านทางช่อง ARC
- HEC (HDMI Ethernet Channel) ใช้เชื่อมต่ออุปกรณ์อื่นๆ ผ่านทางสาย HDMI ตั้งแต่เวอร์ชัน 1.3 เป็นต้นมา

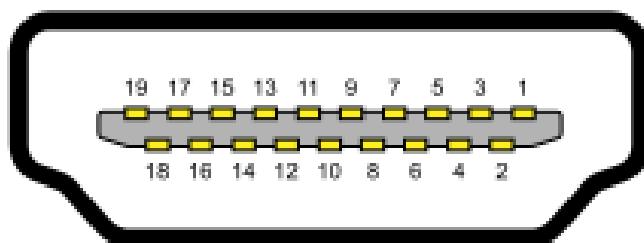


Figure 6.3: หัวเชื่อมต่อ HDMI ชนิด Female ประกอบด้วยขาสัญญาณทั้งหมด 19 ขา ที่มา: <https://en.wikipedia.org/wiki/HDMI>

สาย HDMI ที่ใช้เชื่อมต่อส่วนใหญ่จะเป็นตัวผู้ (Male) ทั้งสองด้าน รูปที่ 6.3 แสดงภาพตัดขวางของหัวเชื่อมต่อ HDMI ชนิดตัวเมีย (Female) ประกอบด้วยขา 19 ขา มีรายชื่อตามตารางที่ 6.1 ตามหมายเลขขา ซึ่ง แล้วแต่ประสิทธิ์ของคอนเนคเตอร์ชนิด HDMI เวอร์ชัน 1.4 ที่มา: (?) รายละเอียดเพิ่มเติมที่: <https://en.wikipedia.org/wiki/HDMI>

Table 6.1: หมายเลขขา ชื่อ และวัตถุประสงค์ของคอนเน็คเตอร์ชนิด HDMI เวอร์ชัน 1.4

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

6.2 สัญญาณ DSI สำหรับจอ LCD ขนาดเล็ก

สัญญาณ DSI (Display Serial Interface) สำหรับเชื่อมต่อกับจอ LCD ขนาดเล็ก กับซีพียูบนอุปกรณ์เคลื่อนที่ เช่น โทรศัพท์เคลื่อนที่ แท็บเล็ต คอมพิวเตอร์โน๊ตบุ๊ก เป็นต้น เพื่อการแสดงผลในรูปของกราฟิก ใหม่ด้วย สัญญาณ DSI นี้ถูกกำหนดเป็นมาตรฐานโดยองค์กรชื่อ MIPI (Mobile Industry Processor Interface)

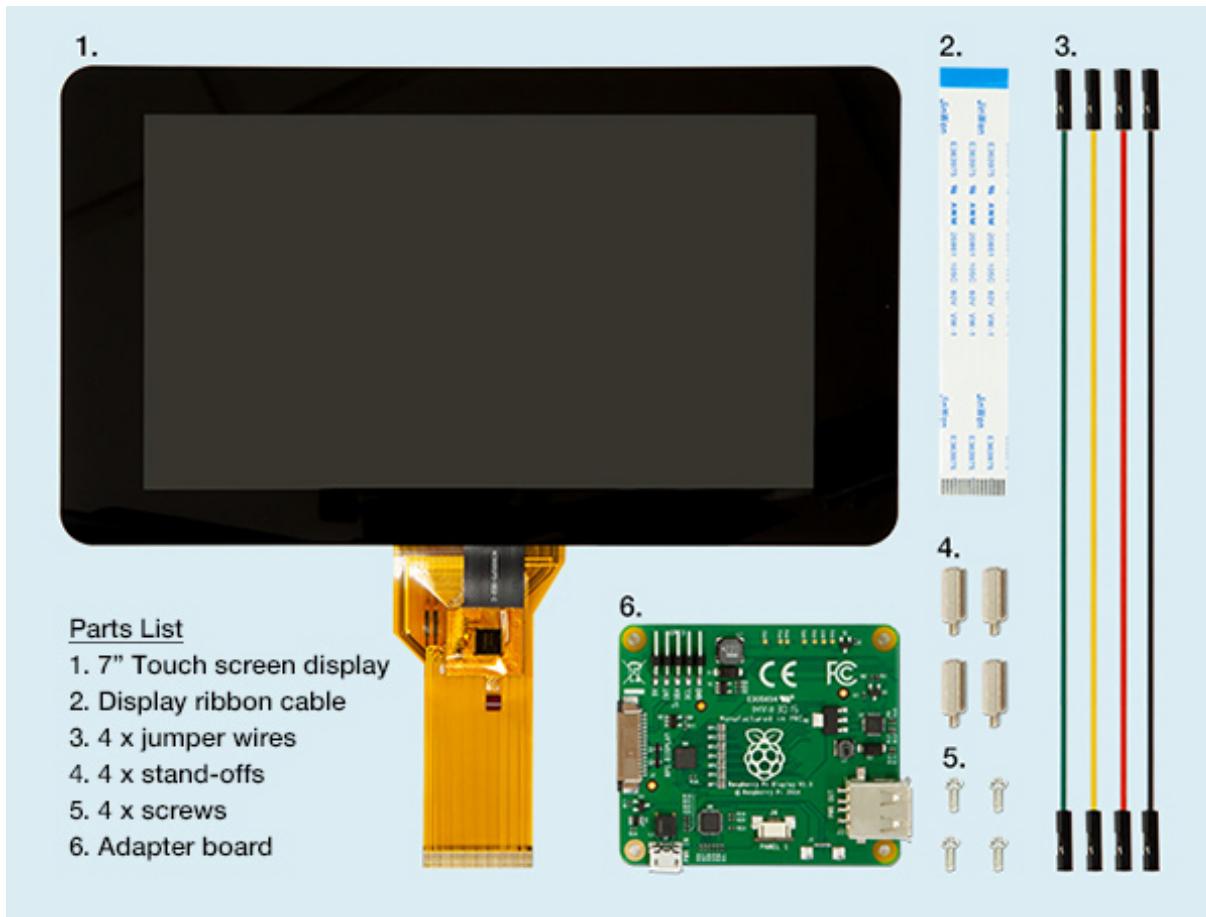


Figure 6.4: จอยแสดงผลสำหรับเชื่อมต่อระหว่างบอร์ด Pi3 ด้วยอินเตอร์เฟสการแสดงผลแบบอนุกรม (Display Serial Interface) ที่มา: <https://www.element14.com/community/docs/DOC-78156/1/raspberry-pi-7-touchscreen-display>

สัญญาณ DSI แบ่งเป็นชนิดเดียว (Single Lane) และหลายๆ เลนตั้งแต่ 2 เลนขึ้นไป เพื่อกระจายการส่งข้อมูลแต่ละใบที่ข้อมูล ไปแต่ละเลนตามรูปที่ 6.5 ข้อมูลใบที่ 0, 4, 8, ... จะส่งมาทางเลนหมายเลข 0 ข้อมูลใบที่ 1, 5, 9, ... จะส่งมาทางเลนหมายเลข 1 ข้อมูลใบที่ 2, 6, 10, ... จะส่งมาทางเลนหมายเลข 2 ข้อมูลใบที่ 3, 7, 11, ... จะส่งมาทางเลนหมายเลข 3 และสลับกันไปแบบนี้เรื่อยๆ การส่งข้อมูลจำนวนหลายๆ เลนพร้อมกันทำได้เร็วขึ้น รองรับการแสดงผลที่ละเอียดมากขึ้น เปลี่ยนแปลงภาพต่อวินาทีได้มากขึ้น การเคลื่อนไหวของภาพจึงต่อเนื่องไม่กระตุก เพิ่มอรรถรสในการรับชมมากขึ้น เมื่อปลายทางรับข้อมูลได้สำเร็จ จะรับจะนำข้อมูลเหล่านั้นรวมกัน (Lane Merging Function)

ตามมาตรฐานสัญญาณ DSI เลนข้อมูลแต่ละเลนใช้หลักการส่งสัญญาณแบบ Differential Signalling คล้ายกับสัญญาณของ USB แต่ใช้ความต่างศักย์ในระดับ 200 มิลลิโวลท์ เรียกว่า Low Voltage Differential Signalling (LVDS) แต่ไม่ได้กำหนดรายละเอียดของการเชื่อมต่อ กับอินพุทชนิดสัมผัส (Touch Screen Sensor) จอย LCD จะทำงานในโหมดแสดงผล และโหมดรับคำสั่ง โดยการรับส่งข้อมูลและคำสั่งในเลนที่ 0 ส่วนเลนที่ 1 เป็นต้นไปจะทำหน้าที่ส่งข้อมูลเท่านั้น เลนหมายเลข 0 สามารถรับและส่งข้อมูลได้ทั้งสองทิศทาง (BiDirectional) ในขณะที่เลนหมายเลขอื่นๆ จะส่งข้อมูลไปยังจอย LCD ได้เพียงทิศทางเดียวเท่านั้น

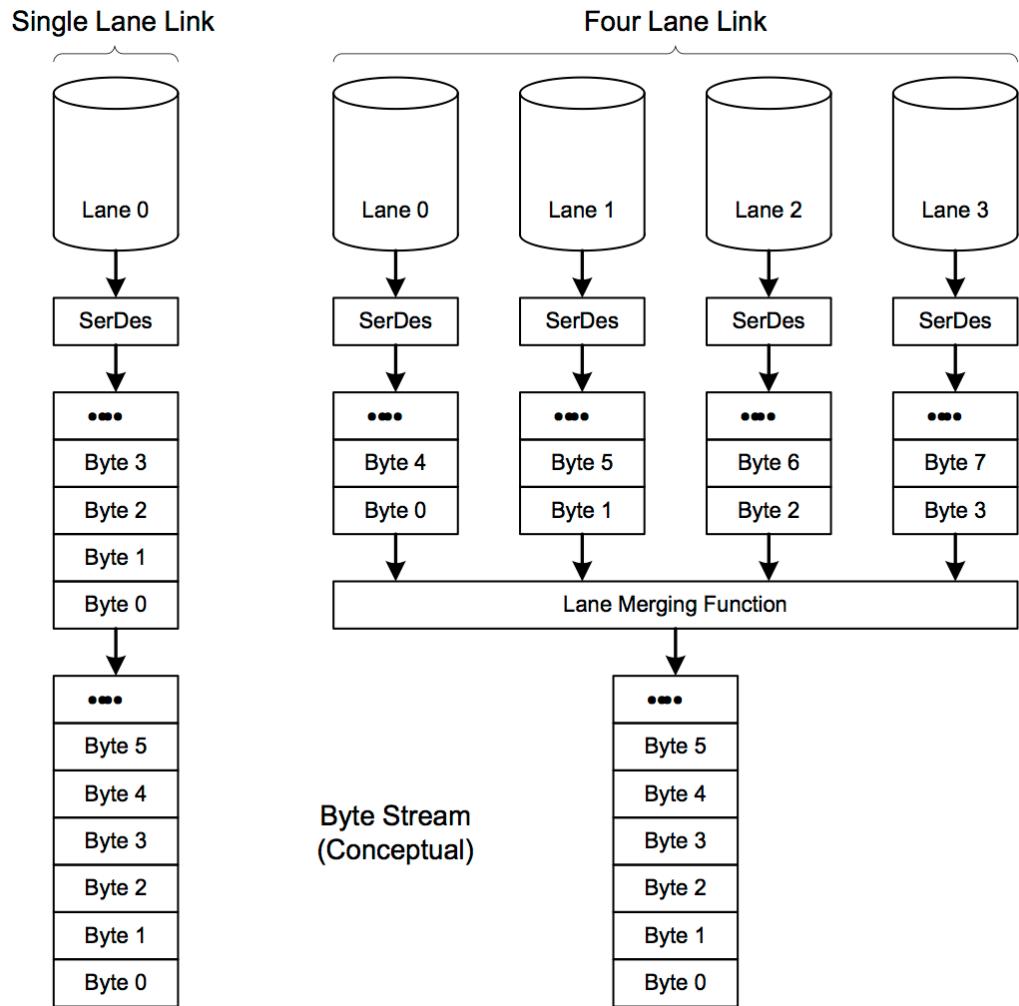


Figure 6.5: สัญญาณ DSI แบ่งเป็นช่องหนึ่งเลน (Single Lane) และหลายเลน ที่มา:

Table 6.2: หมายเลข และหน้าที่ของสายสัญญาณ DSI สำหรับจอ LCD ขนาดเล็ก

ขा	ชื่อ	หน้าที่
1	Ground	กราวด์
2	Data Lane 1-	ขาลงเลนข้อมูล 1
3	Data Lane 1+	ขาขึ้นเลนข้อมูล 1
4	Ground	กราวด์
5	Clock N	ขาลงคล็อก
6	Clock P	ขาขึ้นคล็อก
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

สำหรับบอร์ด Pi3 คอนเนคเตอร์ S2 เป็นหัวเชื่อมต่อสัญญาณ DSI โดยใช้สายแพ็จำนวน 15 ขา โดยส่งข้อมูลพร้อมๆ กันจำนวนหลายเลน แต่ละเลนเป็นแบบอนุกรม (Serial) ตารางที่ 6.2 แสดงหมายเลข ซึ่งอ่อนตุ่นประสิทธิภาพของขาทั้ง 15 ขา โดย ขา 5 และ 6 จะถูกกำหนดให้เป็นเลนสัญญาณคลีก ขา 8 และ 9 จะถูกกำหนดให้เป็นเลนข้อมูลหมายเลข 0 ขา 2 และ 3 จะถูกกำหนดให้เป็นเลนข้อมูลหมายเลข 1 เป็นต้น

6.3 สัญญาณ CSI สำหรับเชื่อมต่อกล้องขนาดเล็ก



Figure 6.6: การเชื่อมต่อระหว่างบอร์ด Pi3 และกล้องขนาดเล็กด้วยอินเตอร์เฟสกล้องแบบอนุกรม (Camera Serial Interface) ที่มา: <https://www.element14.com/community/docs/DOC-83171/1/raspberry-pi-3-model-b-with-camera-module-v2#documents>

บอร์ด Pi3 รองรับการเชื่อมต่อกล้องขนาดเล็ก ตามมาตรฐาน CSI: Camera Serial Interface ซึ่งมีความคล้ายคลึงกับสัญญาณ DSI ซึ่งกำหนดโดยองค์กรเดียวกัน คือ MIPI ข้อมูลภาพจากกล้องจะส่งผ่านสายด้วยเลนข้อมูลจำนวนหนึ่ง ตามรูปที่ 6.5 เพื่อไปรวมกันเป็นภาพเดียวที่ปลายทาง แต่ละเลนมีการส่งข้อมูลทีละบิต

กล้องจะเชื่อมกับบอร์ด Pi3 กับช่องอกเก็ตหมายเลข S5 ซึ่ง เป็นชนิดยืดติดบนพื้นผิวของแผ่นวงจรพิมพ์ (Surface Mount) และไม่ต้องใช้แรงกด ZIF (Zero Insertion Force) ชนิด 15 ขา เชื่อมต่อกับสายแพ (Ribbon) โดย CSI ได้กำหนดให้สัญญาณที่ใช้สำหรับเลนข้อมูลเป็น Low Voltage Differential Signalling (SubLVDS) ซึ่งปรับปรุงมาจากมาตรฐานสัญญาณ IEEE1596.3 LVDS สำหรับอุปกรณ์ที่ใช้ไฟเลี้ยงต่าประมาณ 1.2 โวลท์ เพื่อให้สามารถส่งข้อมูลต่อเลนได้สูงสุด 800-1,000 เมกะบิตต่อวินาที (Mbps)

Table 6.3: หมายเลขขา ชื่อ และวัตถุประสงค์ของคอนเนคเตอร์ชนิด CSI

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

ตารางที่ 6.3 แสดงหมายเลข ชื่อ และวัตถุประสงค์ของคอนเนคเตอร์ชนิด CSI ดังนี้

- CAM1_D0-/CAM1_D0+ และ CAM1_D1-/CAM1_D1+ คือ สัญญาณขั้วลบและบวกของเลนข้อมูลที่ 0 และ 1 ตามลำดับ โดยซิพในกล้องจะสร้างสัญญาณแล้วส่งผ่านสายแพให้กับวงจรที่รับภาพในซิพ BCM2837 ผ่านเลนข้อมูลทั้งสองเลนนี้ตามรูปที่ 6.5

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

- CAM1_C- and CAM1_C+ คือ สัญญาณคลีอกขาลบและขาบวกตามลำดับ เพื่อให้การส่งข้อมูลภาพในเลนต่างๆ ที่ความเร็วสูงเป็นแบบซิงโครนัส
- SDA0 และ SCL0 SDA0 และ SCL0 คือ สัญญาณข้อมูลและคลีอกตามมาตรฐาน I2C ซึ่งรายละเอียดเพิ่มเติมอยู่ในหัวข้อที่ 6.16 เพื่อควบคุมการทำงานของกล้อง เช่น ความละเอียดของภาพ รูปแบบของข้อมูล เป็นต้น โดยการรับส่งจะซิงโครในรูปแบบที่กำหนด SCL0 ซึ่งมีความถี่ต่ำกว่าสัญญาณคลีอก CAM1_C

6.4 สัญญาณ PCM สำหรับสัญญาณเสียง

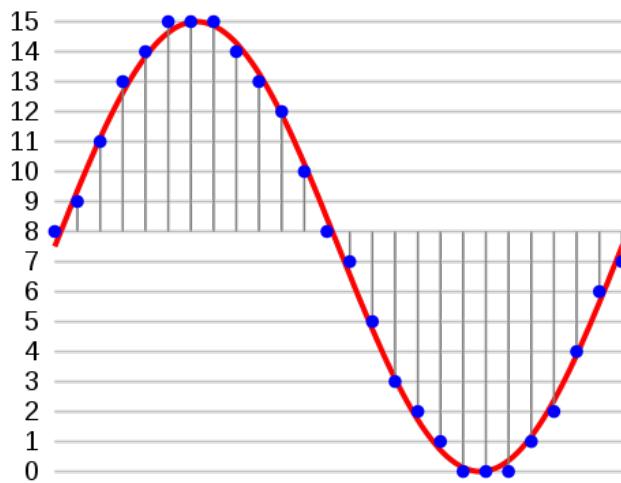


Figure 6.7: รูปคลื่นไซน์ (Sine Wave) และสัญญาณ PCM (Pulse Code Modulation) 16 ระดับ ที่มา: https://en.wikipedia.org/wiki/Pulse_code_modulation

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

สัญญาณอนาล็อกรูปคลื่นไซน์จะโดนสุ่มด้วยระยะเวลาต่อเนื่องและสม่ำเสมอ ด้วยความเวลาที่สูงพอสมควร ขึ้นอยู่กับคุณภาพสัญญาณที่ต้องการ ยกตัวอย่าง เช่น

- ตัวอย่างที่ 1 สัญญาณเสียงสนทนากับโทรศัพท์จะ สุ่มด้วยความถี่ 8,000 ครั้งต่อวินาที ซึ่งจะตรงกับความเวลา $1/8,000 = 125$ มิโครวินาที ด้วยระดับความดัง 256 ระดับ หรือ 8 บิต
- ตัวอย่างที่ 2 คือ สัญญาณเสียงเพลงคุณภาพระดับแผ่นซีดี จะสุ่มด้วย ความถี่ 44,100 ครั้งต่อวินาที ซึ่งจะตรงกับความเวลา $1/44,100 = 22.67$ มิโครวินาที ด้วยระดับความดัง 65,536 ระดับ เพื่อให้เป็นข้อมูลขนาด 16 บิตต่อการสุ่ม 1 ครั้ง ทั้งนี้ การสุ่มจะเกิดขึ้นกับช่องสัญญาณซ้ายและขวา รวมเป็นข้อมูล 32 บิตต่อการสุ่ม 1 ครั้ง
- ตัวอย่างที่ 3 รูปที่ 6.7 แสดง รูปคลื่นไซน์ (Sine Wave) และการสุ่มค่าของคลื่นไซน์นี้ ด้วยความถี่สูง เป็น 26 เท่าของความถี่เดิม และทำการแปลงเป็นสัญญาณดิจิทัลชนิด PCM (Pulse Code Modulation) จำนวน 16 ระดับให้กลายเป็นข้อมูลขนาด 4 บิตต่อการสุ่ม 1 ครั้ง

ภายในชิป BCM2837 มีโมดูล PCM ตามรูปที่ 6.8 ประกอบด้วย บัฟเฟอร์สำหรับส่ง (Transmit) และรับ (Receive) ขนาด 64×32 บิต จำนวน 2 ชุด โมดูล PCM เชื่อมอยู่บนบัส APB (ARM Peripheral Bus) เพื่อรับสัญญาณเสียงเข้า ผ่านโมดูลนี้เพื่อพกเก็บในหน่วยความจำหลักชั่วคราว เพื่อรอให้ ซีพียูและวิดีโอ คอร์ประมวลผล และส่งสัญญาณเสียงไปยังลำโพง

PCM มีสายสัญญาณตามมาตรฐาน I2S (Inter IC Sound) จำนวน 4 เส้นเพื่อเชื่อมต่อ กับชิปภายนอก BCM2837 แบบอนุกรม ได้แก่

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

รายละเอียดเพิ่มเติมที่ [https://en.wikipedia.org/wiki/I²S](https://en.wikipedia.org/wiki/I%20S)

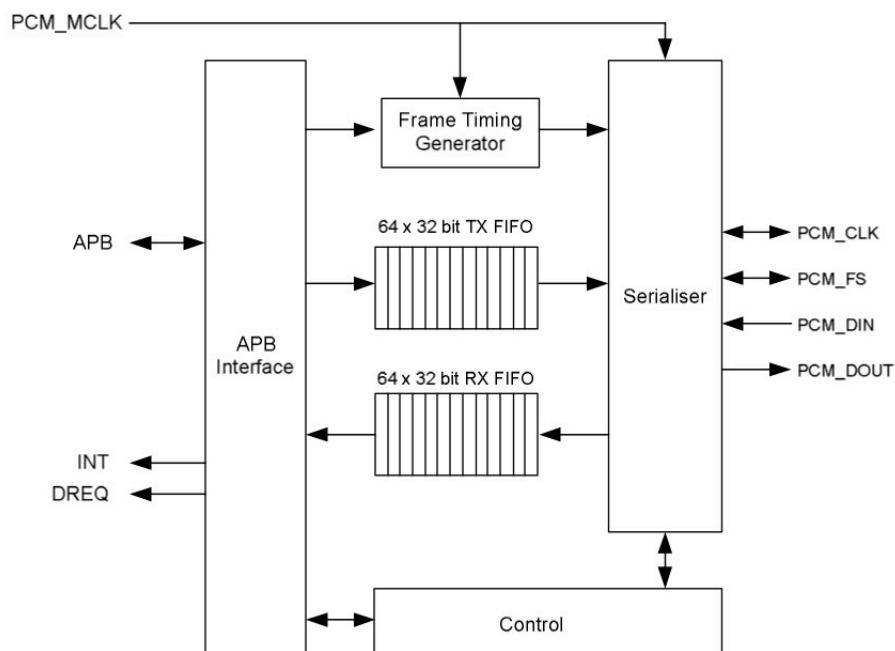


Figure 6.8: บีฟเฟอร์สำหรับส่งและรับ ข้อมูลเสียงดิจิทัลชนิด PCM จากการเชื่อมต่อชนิด I2S ที่มา: [Broadcom \(2012\)](https://www.broadcom.com)

ไม่ดูแลนี้รองรับการทำงานทั้งสามรูปแบบ คือ โพลลิ่ง (Polling) การขัดจังหวะ (Interrupt) รายละเอียดเพิ่มเติมในหัวข้อที่ 6.12 และ การเข้าถึงหน่วยความจำโดยตรง (Direct Memory Access) รายละเอียดเพิ่มเติมในหัวข้อที่ 6.13

6.5 สัญญาณภาพและเสียงสำหรับจอทีวี



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

ช่องเสียบสายในรูปที่ 6.9 ใช้เชื่อมต่อสัญญาณภาพและเสียงจากบอร์ด Pi3 เข้ากับจอทีวีหรือจอมอนิเตอร์ที่มีช่องอินพุทเป็น คอมโพสิตวีดิโอ (Composite Video) และเสียงสเตอโรโนว์ หรือ โทรทัศน์บางเครื่องเรียกว่าช่อง AV In ซึ่งสามารถใช้ทดแทนจอ LCD ได้ สำหรับโรงเรียนหรือผู้ใช้ที่ขาดแคลนทุนทรัพย์ แต่จะได้สัญญาณภาพที่ละเอียดต่ำกว่าสัญญาณ HDMI โดยแจ็ค 3.5 มม. (กลาง) นี้เป็นชนิด 4 ขั้วสำหรับเสียบกับบอร์ด Pi3 (ขวา) เพื่อส่งสัญญาณภาพไปยังแจ็ค RCA (เหลือง) และสัญญาณเสียงไปยังแจ็ค RCA (แดงและขาว) ผู้อ่านสามารถหาชื่อสายสัญญาณสำเร็จรูปนี้ได้ทั่วไป หรือจะบัดกรีเชื่อมต่อสายเองได้ เช่นกัน

สัญญาณภาพดิจิทัลจะสร้างโดยชิป BCM2837 แล้วแปลงเป็นสัญญาณภาพอนาล็อก โดยใช้ DAC (Digital to Analog Converter) ตามรูปที่ 3.2 เรียกว่าสัญญาณชื่อ TVDAC ในรูปที่ 6.1 ในขณะที่ สัญญาณเสียงดิจิทัลชนิด PCM ซึ่งได้กล่าวแล้วในหัวข้อที่ 6.4 จะถูกแปลงเป็นสัญญาณเสียงอนาล็อกโดยใช้ DAC ภายในชิป BCM2837 เช่นกัน เป็นสัญญาณเสียงซองซ้าย และซองขวาแบบสเตอโรโนว์ (Stereo) เพื่อผ่านฟิลเตอร์ (Filter) หรือตัวกรองความถี่ตามรูปที่ 3.2

6.6 สัญญาณ USB 2.0 สำหรับอุปกรณ์ต่อพ่วงต่างๆ

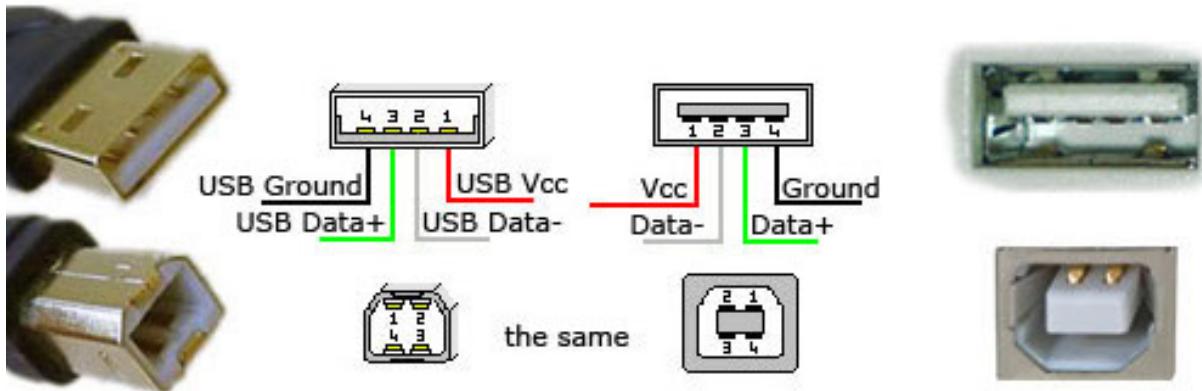


Figure 6.10: หัวเชื่อมต่อ USB ชนิด A (บน ฝั่งไฮสท์) และ B (ล่าง ฝั่งอุปกรณ์) ประกอบด้วยสัญญาณ 4 เส้น กราวด์ (GND) Data+ Data- และไฟเลี้ยง 5 โวลท์

เนื้อหาในหัวข้อนี้ต่อเนื่องจากเรื่อง คีย์บอร์ด ในหัวข้อที่ 3.1.3 และมาส์ ในหัวข้อที่ 3.1.3 ซึ่งนิยมใช้เป็นแบบ USB เพื่อต่อเชื่อมกับบอร์ด Pi3 และคอมพิวเตอร์ทั่วไป เวอร์ชันปัจจุบันของ USB คือ 3.1 ซึ่งมีความสามารถสูงขึ้นและได้รับความนิยมเพิ่มขึ้น ในตำราเล่มนี้จะกล่าวถึง 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 แอมเบร็ฟสำหรับพอร์ตพิเศษ
- สายเคเบิลมีความยาวไม่เกิน 5 เมตร เนื่องจากความต้านทานของสายจะทำให้เกิดโวลเทจตกรคร่อม (Voltage Drop) ในสาย จนทำให้ความต่างศักย์ไปเลี้ยงอุปกรณ์ไม่เพียงพอ
- ”Hot Swapping” รองรับการต่อเชื่อม ถอนออก และรีเซ็ตอุปกรณ์ที่ต่ออยู่โดยไม่ต้องรีเซ็ตหรือรีบูตระบบโอเอส

รูปที่ 6.10 แสดง หัวเชื่อมต่อ USB ชนิด A (ฝั่งไฮสท์หรือคอมพิวเตอร์) และ B (ฝั่งอุปกรณ์) ในการเชื่อมต่อทางไฟฟ้าของ USB นั้นจะสายเคเบิลแบบ 4 คอร์ เพียง 1 เส้นต่อ 1 อุปกรณ์เท่านั้น ซึ่งมีตำแหน่งขาดังนี้

- ขา 1 เป็น 5V+ สำหรับจ่ายไฟเลี้ยงให้กับอุปกรณ์ขนาดเล็ก เช่น แฟลชไดร์ กล้องเว็บแคม โทรศัพท์スマาร์ทโฟน เป็นต้น
- ขา 2 เป็น D- เป็นสายสัญญาณรับส่งข้อมูลแบบ Differential
- ขา 3 เป็น D+ เป็นสายสัญญาณรับส่งข้อมูลแบบ Differential
- ขา 4 เป็น GND เป็นขากราวด์สำหรับไฟเลี้ยง 5 โวลท์

สายส่งข้อมูลของระบบ USB มี 2 สาย สำหรับ สัญญาณ D+ และ D- ในการส่งสัญญาณแบบ เป็นสายสัญญาณรับส่งข้อมูลแบบ Differential คือ

กรณีการส่งสัญญาณ ”0” สายสัญญาณ D+ จะมีระดับแรงดันที่ต่ำกว่า D- อย่างน้อย 200 mV
กรณีการส่งสัญญาณ ”1” สายสัญญาณ D+ จะมีระดับแรงดันที่สูงกว่า D- อย่างน้อย 200 mV

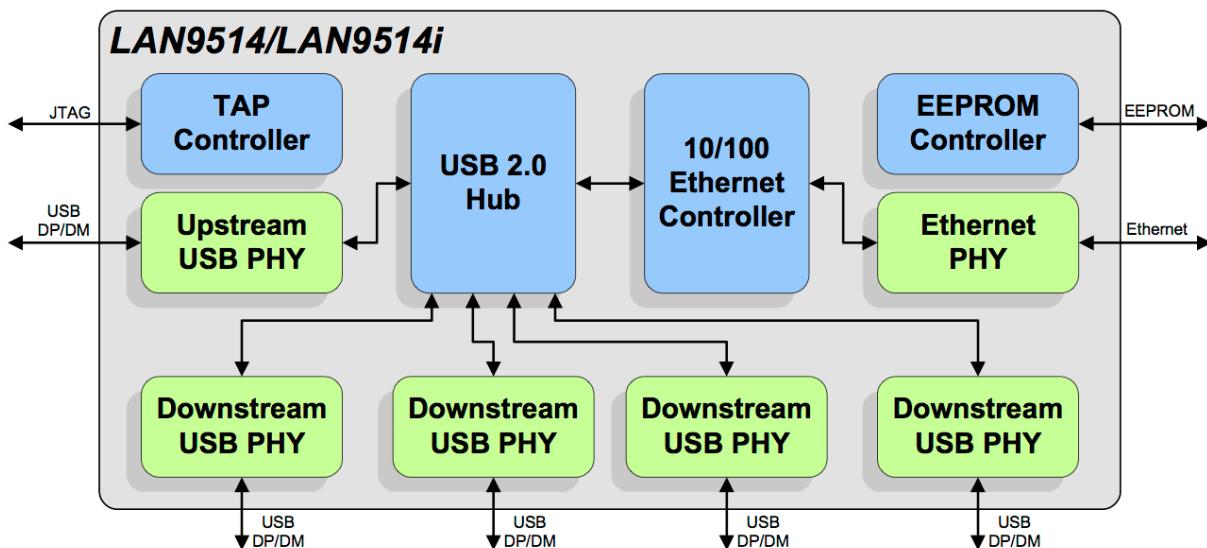


Figure 6.11: โครงสร้างของชิป LAN 9514 ภายในประกอบด้วยวงจร USB Hub และวงจร Ethernet [Microchip Technology \(2009\)](#)

จากรูปที่ 3.2 BCM2837 จะเขื่อมต่อกับไอซี LAN9514 เพื่อขยายเป็น 4 พอร์ต เนื่องจากภายในชิป BCM2837 จะมีรูทฮับ (Root Hub) เพียง 1 พอร์ต โครงสร้างของไอซี LAN9514 ตามรูปที่ 6.11 ถูกออกแบบให้ LAN9514 มี USB Hub (Upstream) จำนวน 1 พอร์ต เพื่อเขื่อมกับรูทฮับในชิป BCM2837 และขยายจำนวนพอร์ต (Downstream) เพิ่มเป็น 4 พอร์ต เพื่อต่อเขื่อมกับคีย์บอร์ด 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 สำหรับสายเชื่อมต่อกับเครือข่ายอินเตอร์เน็ต



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

อีเธอร์เน็ต (Ethernet) คือ เทคโนโลยีเครือข่าย LAN (Local Area Network) ที่นิยมใช้กันอย่างกว้างขวาง ในอดีตมานานปัจจุบัน เพราะเป็นการรับส่งข้อมูลแบบอนุกรมด้วยความเร็วสูง ใช้สายทองแดงในการติดตั้ง และต้นทุนไม่สูง เนื่องจากมีอุปกรณ์สนับสนุนเพื่อใช้งานมากที่สุด รูปที่ 6.12 แสดงตำแหน่งของพอร์ต เชื่อมต่อสาย Ethernet บริเวณมุมของบอร์ด Pi3 หัวเชื่อมต่อชนิด RJ45 แบบตัวเมีย (Female) โดยใช้สาย CAT5e ชี้ไป

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 โดยจะต่อเข้าบอร์ดเข้ากับอุปกรณ์เครือข่าย ที่เรียกว่า Ethernet Switch ตามลำดับชั้น เพื่อสุดท้ายเชื่อมต่อกับเครือข่ายอินเตอร์เน็ต ดังนั้น ในการทดลองที่ 7 ภาคผนวก G การศึกษาและปรับแก้อินพุท/เอาท์พุทต่างๆ แสดงรายละเอียดของ Ethernet การตั้งค่า (Configuration) และอื่นๆ เพื่อให้บอร์ดทำหน้าที่เป็นเซิร์ฟเวอร์ (Server) เช่น เว็บเซิร์ฟเวอร์ FTP เซิร์ฟเวอร์ เป็นต้น อุปกรณ์ IoT จากการเชื่อมต่อกับเซ็นเซอร์ต่างๆ



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

การเชื่อมต่อเครือข่ายท้องถิ่น (Local Area Network) แบบมีสาย Ethernet รูปที่ 6.13 ซึ่งสามารถเชื่อมต่อเครือข่ายคอมพิวเตอร์จำนวนหลายเครื่องบนเครือข่ายเดียวกัน โดยอาศัยหลักการ CSMA/CD (Carrier Sense Multiple Access/Collision Detection) รายละเอียดเพิ่มเติม ผู้อ่านสามารถศึกษาเพิ่มเติมได้ที่ลิงค์ต่อไปนี้ <https://en.wikipedia.org/wiki/Ethernet> 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)

นอกจากการเชื่อมต่อแบบไร้สายแล้ว บอร์ด Pi3 ได้ถูกออกแบบให้ทันสมัย และรองรับการเชื่อมต่อแบบไร้สายลีสสองชนิด คือ

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

วงจรสำหรับการเชื่อมต่อแบบไร้สายทั้งสองรวมอยู่ในชิป BCM 43438 บนบอร์ด RPi3 ในรูปที่ 6.14

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

ชิป BCM 43438 บนบอร์ด RPi3 รองรับสัญญาณ IEEE 802.11b/g/n ที่ย่านความถี่คลื่นพาร์ท 2.4 GHz เท่านั้น และสัญญาณ Bluetooth เวอร์ชัน 4.1 รวมถึงตัวรับสัญญาณวิทยุ FM Corporation (2017) บล็อกไดอะแกรมของชิป BCM 43438 ประกอบด้วยขาสัญญาณเชื่อมต่อเสาอากาศ และขาเชื่อมต่อกับไมโครคอนโทรลเลอร์ (Host Interface) ทั้งสองสัญญาณใช้สายอากาศ (Antenna) และความถี่พาร์ทหลัก ในย่านความถี่ 2.4 GHz จิกะเฮิร์ซเดียว กับ บลูทูธใช้หลักการ Frequency Hopping และกำลังส่งที่ต่ำกว่า สัญญาณ WiFi ทำให้ไม่เกิดการรบกวนกัน

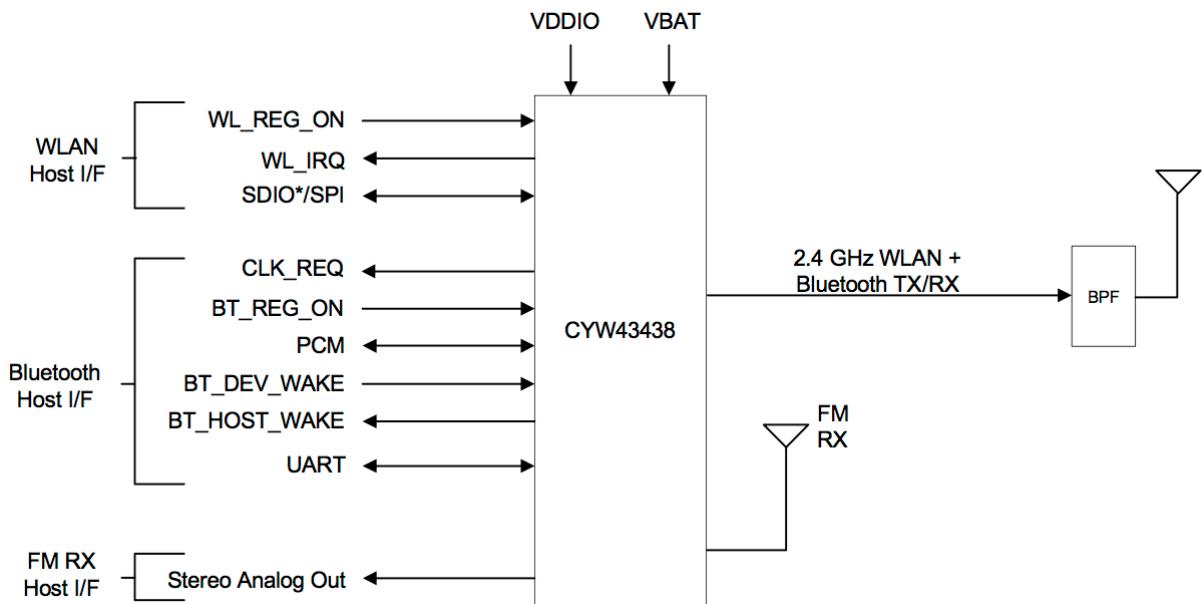


Figure 6.15: บล็อกไดอะแกรมของชิป BCM 43438 ประกอบด้วยขาสัญญาณเชื่อมต่อเสาอากาศ และขาเชื่อมต่อกับไมโครคอนโทรลเลอร์ ที่มา: Corporation (2017)

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 ทุกตัวได้เช่นกัน

Bluetooth จะใช้ความถี่ความถี่คลื่นพาร์ท ในย่าน 2.4 GHz. (จิกะเฮิร์ซ) 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 เป็นการป้องกันการตักสัญญาณระหว่างสื่อสาร โดยระบบจะสลับช่องสัญญาณไปมา สามารถค้นคว้ารายละเอียดเพิ่มเติมที่ <https://www.bluetooth.com>

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

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

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

6.9 หลักการ Memory Mapped Input/Output

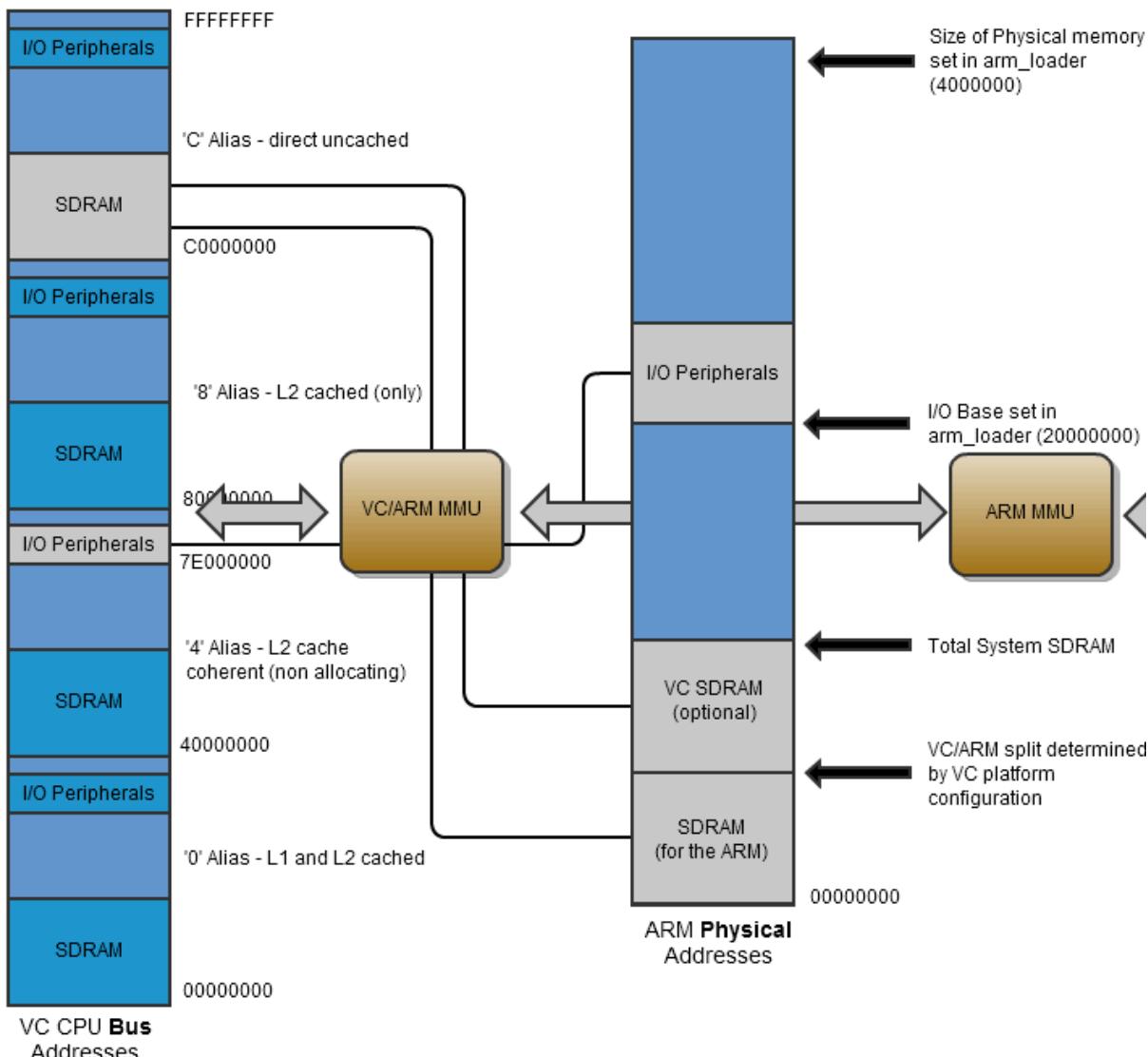


Figure 6.16: การแมปหน่วยความจำระหว่างแอดเดรสบัส (Bus Address) ของ Video Core (VC) และ แอดเดรสภายในของ ARM

โปรแกรมต่างๆ อ้างอิงกับหน่วยความจำเสมือนภายใต้ระบบปฏิบัติการลีนักซ์ โปรแกรมเหล่านี้จะเข้าถึงอุปกรณ์อินพุทเอาท์พุทผ่าน System Call Interface ของระบบปฏิบัติการเท่านั้น เช่น ฟังค์ชัน printf ในไฟล์ stdio.h เพื่อแสดงผลข้อมูลต่างๆ บนหน้าจอแสดงผล ในเชิงโครงสร้าง ระบบหน่วยความจำเสมือนเชื่อมโยงกับอินพุทเอาท์พุท ผ่านทาง MMU ตัวที่ 1 (ARM MMU) จะแปลงแอ็ดเดรสเสมือนให้กลายเป็นแอ็ดเดรสภายในภาพ ตามรูปที่ [5.16](#) และ MMU ตัวที่ 2 (VC/ARM MMU) จะแปลงแอ็ดเดรสภายในภาพให้กลายเป็นแอ็ดเดรสบนบัสหน่วยความจำ ตามรูปที่ [6.16](#)

การใช้แอ็ดเดรสของหน่วยความจำแปลงเป็นการเข้าถึงรีจิสเตอร์ของอุปกรณ์อินพุทเอาท์พุท แบ่งเป็นรีจิสเตอร์สำหรับ การตั้งค่าเพื่อการควบคุมทำงาน การอ่านข้อมูล การเขียนข้อมูล การอ่าน/เขียนข้อมูล โดยในรูปที่ [6.16](#) จาก I/O Base 0x2000 0000 ตั้งค่าใน ARM Loader เป็น หน่วยความจำเริ่มต้น 0x7E00 0000

แปลงแอ็ดเดรสเสมือนให้เป็นภายในภาพแล้ว ชิป BCM ยังต้องแปลงแอ็ดเดรสภายในภาพให้เป็นแอ็ดเดรสบนบัส (Bus) ซึ่งซึ่อมต่อ กับวิดีโอดีไซด์ และหน่วยความจำ โดยใช้โมดูล VC/ARM MMU

IO Peripherals อุปกรณ์อินพุทเอาท์พุท มีรีจิสเตอร์สำหรับตั้งค่า อ่านข้อมูล และเขียนข้อมูล การอ่านหรือเขียนข้อมูลไปยังรีจิสเตอร์เหล่านี้ ทำได้การใช้คำสั่ง LDR และ STR เมื่อกับหน่วยความจำปกติทั่วไป โดยผู้ผลิตอาร์ดแวร์ได้กำหนดหมายเลขแอ็ดเดรสของหน่วยความจำ ว่าตรงกับรีจิสเตอร์ของอุปกรณ์ตัวใดตามตารางที่ [6.17](#) ต่อไปนี้ โดยแอ็ดเดรสบัสในหน่วยความจำเริ่มต้นที่หมายเลข 0x7E00 0000 ซึ่งตรงกับตัวเลขในรูปที่ [6.16](#)

Figure 6.17: ตารางแอ็ดเดรสบัสเริ่มต้นที่หมายเลข 0x7E00 0000 สำหรับอุปกรณ์อินพุทเอาท์พุท (IO Peripherals) และหัวข้อ

แอ็ดเดรสบัส (Bus Address)	ชื่อ (Name)	รายละเอียด (Details)	หัวข้อ (Section)
0x7E00 3000	System Timer	วงจรจับเวลาสำหรับระบบปฏิบัติการ	-
0x7E00 7000	DMA Controller	การเข้าถึงหน่วยความจำโดยตรง	6.13
0x7E00 B000	Interrupt Register	การอินเตอร์รัพท์	6.12
0x7E00 B400	Timer	วงจรจับเวลาสำหรับการใช้งานทั่วไป	
0x7E20 0000	General Purpose I/O	ขา GPIO ทั้งหมด	6.11
0x7E20 1000	Universal Async. Rx Tx	การสื่อสารแบบอนุกรม	6.14
0x7E20 3000	Pulse Code Modulation	สัญญาณเสียง	6.4
0x7E20 4000	SPI0	Serial Peripheral Interface	6.15
0x7E20 5000	Serial Controller (I2C)	สัญญาณ I2C	6.16
0x7E21 4000	SPI/BSC Slave	สลิฟ Serial Peripheral Interface/ Serial Controller (I2C)	6.15 6.16
0x7E21 5000	mini UART, SPI1, SPI2	การสื่อสารแบบอนุกรม	-
0x7E30 0000	External Mass Media Controller	หน่วยสำรองข้อมูลภายนอก	7.2
0x7E98 0000	Universal Serial Bus	USB	6.6

6.10 หัวเชื่อมต่อ 40 ขา (40-Pin Header)

ขาเชื่อมต่อบนบอร์ดสำหรับเชื่อม BCM2837 สู่โลกภายนอก ในรูปที่ 3.2 ประกอบด้วย ส่วนต่างๆ เหล่านี้

- ไฟเลี้ยงชนิด 3.3 โวลท์, 5.0 โวลท์ และกราวด์ (GND)
- ขาเชื่อมต่อชนิด GPIO General Purpose input/output ได้แก่ ขา GPIO00-GPIO27
- ขาสำหรับการสื่อสารชนิด UART ได้แก่ ขา RXD0/TXD0, ขา RXD1/TXD1
- ขาสำหรับการเชื่อมต่อกับชิพภายนอกชนิด SPI ได้แก่ ขา SPI1_CE0, SPI1_CE1, SPI1_CE2, SPI1_MISO, SPI1_MOSI, SPI1_SCLK
- ขาสำหรับการเชื่อมต่อสัญญาณเสียง ได้แก่ ขา PCM_DIN, PCM_DOUT, PCM_FS, PCM_CLK
- ขาสำหรับการเชื่อมต่อกับอุปกรณ์ภายนอกชนิด PWM และ PPM ได้แก่ ขา PWM0, PWM1
- ขาสำหรับการเชื่อมต่อกับชิพภายนอกชนิด I2C ได้แก่ ขา SDA0, SCL0, SDA1, SCL1
- ขาสำหรับการเชื่อมต่อกับ SD Card ภายนอก ได้แก่ ขา SD0_CMD, SD0_CLK, SD0_DAT0, SD0_DAT0-SD0_DAT3

ขาสัญญาณเหล่านี้สามารถเขียนโปรแกรมควบคุมสั่งการด้วยภาษา Assembly, C หรือ Python ในการทดลองต่างๆ

เนื่องจากชิป BCM2837 มีจำนวนขาที่จำกัด เพียง 200 ขา ที่มา: ทำให้ต้องใช้ขาเดียวกัน เพื่อมัลติเพล็กซ์ (Multiplex) หรือเลือกสัญญาณที่จะใช้ ก่อนใช้งานขาเหล่านั้น โปรแกรมเมอร์จะต้องกำหนดว่าขาหมายเลขนี้มัลติเพล็กซ์เป็นสัญญาณอะไร ทำให้ขาเดียวกันมีสัญญาณหลายชื่อ ตามที่ได้สรุปในตารางที่ 6.4

Table 6.4: หมายเลขขา ชื่อ 1 และชื่อ 2 ของหัวเชื่อมต่อสายทั้ง 40 ขา (GND: Ground) ที่มา: ?

ขา	ฟังค์ชันที่ 1	ฟังค์ชันที่ 2	ฟังค์ชันที่ 3	ฟังค์ชันที่ 4	ฟังค์ชันที่ 5
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	FL1	SD9	SPI1_CE1	RTS1
12	GPIO18	PCM_CLK	SD10	SPI1_CE0	PWM0
13	GPIO27	SD0_DAT3			
14	GND				
15	GPIO22	SD0_CLK			
16	GPIO23	SD0_CMD			
17	3.3V				
18	GPIO24	SD0_DAT0			
19	GPIO10	SPI0_MOSI	SD2		
20	GND				
21	GPIO09	SPI0_MISO	SD1		
22	GPIO25	SD0_DAT1			
23	GPIO11	SPI0_CLK			
24	GPIO08	SPI0_CE0			
25	GND				
26	GPIO07	SPI0_CE1			
27	GPIO00	SDA0	SA5	PCLK	
28	GPIO01	SCL0	SA4	DE	
29	GPIO05	GPCLK1	SA0		
30	GND				
31	GPIO06	GPCLK2			
32	GPIO12	PWM0			
33	GPIO13	PWM1			
34	GND				
35	GPIO19	PCM_FS		SPI1_MISO	PWM1
36	GPIO16	FL0	CTS0	SPI1_CE2	CTS1
37	GPIO26				
38	GPIO20	PCM_DIN		SPI1_MOSI	GPCLK0
39	GND				
40	GPIO21	PCM_DOUT		SPI1_SCLK	GPCLK1

6.11 ขา GPIO (General Purpose Input Output)

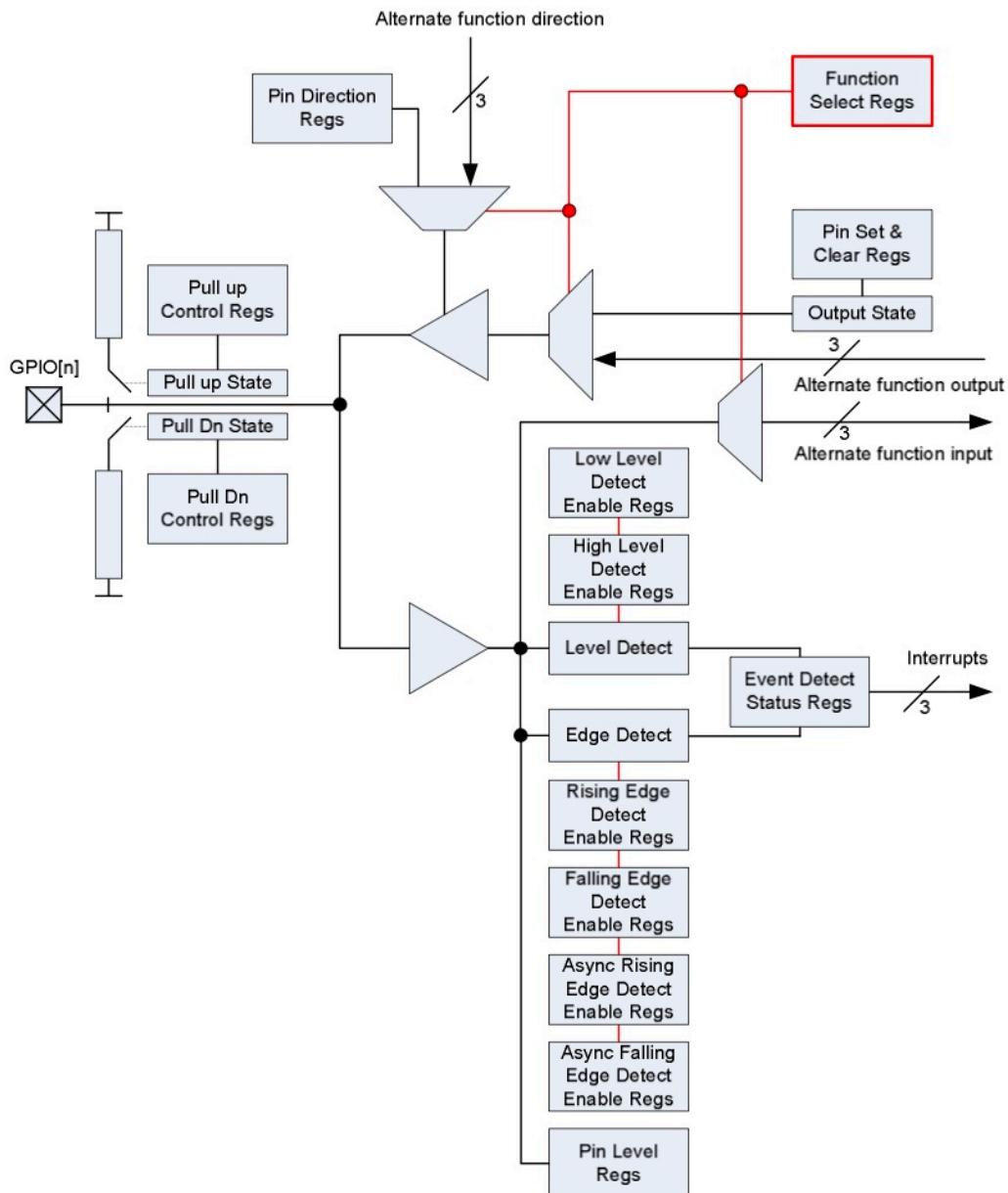


Figure 6.18: โครงสร้างภายในขา GPIO สำหรับชิปตระกูลเดียวกับ BCM2835 ที่มา: [Broadcom \(2012\)](#)

โครงสร้างของขา GPIO[n] สำหรับชิปตระกูล BCM2837 ในรูปที่ 6.18 คือ ขาที่ n จากจำนวนทั้งหมด 54 ขา เมื่อทำการเข้าใจจากซ้ายไปขวา จะเห็นว่าขาแต่ละข้าประกอบด้วย

- 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 หรือ Alternate นั่นคือ การสลับระหว่าง เข้า และ ออก ตามการควบคุมด้วย สัญญาณ Alternate Function Direction
- วงจรฝั่งขาออก (Output) ซึ่งอยู่ด้านบนของรูป ประกอบด้วย รีจิสเตอร์ Pin Set & Clear สัญลักษณ์ สี่เหลี่ยมคงที่ แทนอุปกรณ์ มัลติเพล็กเซอร์ สามเหลี่ยม แทนวงจรบัฟเฟอร์เพื่อเพิ่มกระแสให้กับฝั่งเอาท์พุท
- วงจรฝั่งขาเข้า (Input) ซึ่งอยู่ด้านล่างของรูป ประกอบด้วย
 - วงจรอรจับระดับสัญญาณ (Level Detect) ประกอบด้วย รีจิสเตอร์ High/Low Level Enable
 - วงจรอรจับขอบสัญญาณ (Edge Detect) ประกอบด้วย รีจิสเตอร์ Sync/Async Rising/Falling Edge Detect Enable
 - รีจิสเตอร์สถานะตรวจจับเหตุการณ์ (Event Detect Status Register)
 - รีจิสเตอร์ Pin Level

BCM2837 เป็นล็อกอิกชนิด CMOS ล็อกิก 1 ของขาทั้งหมดใช้ความต่างศักย์ 3.3 โวลท์เท่านั้น ในขณะที่ไฟเลี้ยง 5.0 โวลท์ใช้สำหรับจ่ายไฟให้กับชิปอื่นๆ บางตัวที่เป็นล็อกอิก CMOS นักพัฒนาจะต้องระวัง กรณีเชื่อมต่อวงจรทั้งสองชนิดเข้าด้วยกัน การใช้ขาทำงานในลักษณะ GPIO โปรแกรมเมอร์และนักพัฒนาจะต้อง เชื่อมต่อวงจรและเยี่ยนโปรแกรมควบคุมทิศทางการให้ของสัญญาณว่า เป็น อินพุท หรือ เอาท์พุท ตามวงจรที่ต่อไว้ให้ถูกต้อง มิเช่นนั้นอาจทำให้ ชิป BCM2837 เสียหาย โดยเฉพาะการต่อไฟเลี้ยง 3.3 และ 5.0 โวลท์ กับกราวด์โดยไม่ได้ตั้งใจ

ตารางที่ [6.5](#) คือ รายละเอียดของแอดเดรสหมายเลข 0x7E20 0000 ในตารางที่ [6.17](#)

- GPFSEL0 (GPIO Function Select 0) ควบคุมขาที่ 0 - 31 ชาละ 3 บิต ทั้งหมด 54 ชา สำหรับตั้งค่าของรีจิสเตอร์ Function Select ดังนี้ ขาที่ 0-9 ใช้ข้อมูลในรีจิสเตอร์จำนวน 30 บิต ของแอดเดรส 0x7E20 0000 โดยขาที่ 0 ใช้งานบิตที่ 0-2 ตามลำดับจนถึงขาที่ 9 ใช้งานบิตที่ 28-30 ส่วนขาที่ 10-19 ใช้ข้อมูลในรีจิสเตอร์จำนวน 30 บิต ของแอดเดรส 0x7E20 0004 โดยขาที่ 10 ใช้งานบิตที่ 0-2 ตามลำดับจนถึงขาที่ 19 ใช้งานบิตที่ 28-30 ไปเรื่อยๆ ตามลำดับ จนถึงขาที่ 50-53 ใช้ข้อมูลในรีจิสเตอร์จำนวน 12 บิต ของแอดเดรส 0x7E20 0014 โดยขาที่ 50 ใช้งานบิตที่ 0-2 ตามลำดับจนถึงขาที่ 54 ใช้งานบิตที่ 9-11
- เอาท์พุท
 - GPSET0 (GPIO Pin Output Set) ให้ขาที่ถูกควบคุมเป็นล็อกอิก 1 โดยควบคุมขาที่ 0 - 31 ชาละ 1 บิต ขาที่ 0 ควบคุมโดยบิตที่ 0 ตามลำดับ ดังนี้ ทั้งหมด 32 ชา ใช้ข้อมูลในรีจิสเตอร์จำนวน 32 บิต ของแอดเดรส 0x7E20 001C ส่วนขาที่ 32 - 53 ชาละ 1 บิต ขาที่ 32 ควบคุมโดยบิตที่ 0 ตามลำดับ ดังนี้ ทั้งหมด 22 ชา ใช้ข้อมูลในรีจิสเตอร์จำนวน 22 บิต ของแอดเดรส 0x7E20 0020

Table 6.5: ตารางแอดเดรสในหน่วยความจำเริ่มต้นที่หมายเลข 0x7E20 0000 สำหรับ GPIO

แอดเดรส	ชื่อ	รายละเอียด	บิต	R/W	ขา
0x7E20 0000	GPFSEL0	GPIO Function Select 0	32	R/W	0-9
0x7E20 0004	GPFSEL1	GPIO Function Select 1	32	R/W	10-19
...
0x7E20 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	GPARSENO	GPIO Pin Async. Rising Edge Detect 0	32	R/W	0-31
0x7E20 0080	GPARSEN1	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

- GPCLR0 (GPIO Pin Output Clear) ให้ขาที่ถูกควบคุมเป็นลอจิค 0 โดยควบคุมขาที่ 0 - 31 ขั้ล 1 บิต ขาที่ 0 ควบคุมโดยบิตที่ 0 ตามลำดับ ดังนั้น ทั้งหมด 32 ขา ใช้ข้อมูลในรีจิสเตอร์จำนวน 32 บิต ของแอดเดรส 0x7E20 0028 ส่วนขาที่ 32 - 53 ขั้ล 1 บิต ขาที่ 32 ควบคุมโดยบิตที่ 0 ตามลำดับ ดังนั้น ทั้งหมด 22 ขา ใช้ข้อมูลในรีจิสเตอร์จำนวน 22 บิต ของ แอดเดรส 0x7E20 002C

- การตั้งขาเป็นขาอินพุท จะมีความซับซ้อนมากกว่าเป็นขาเอาท์พุท ตั้งค่าตรวจจับระดับสัญญาณ (Level Detect) หรือ ตรวจจับขอบ (Edge Detect) GPEDS event detection เกิดจากการตรวจจับระดับโวลาเตจ หรือ เกิดจากการตรวจจับขอบสัญญาณ

- การตรวจจับระดับสัญญาณ แบ่งเป็นลอจิค 0 และลอจิค 1 โดยการตั้งค่าในรีจิสเตอร์ GPLEN (GPIO Pin Low Detect Enable) และรีจิสเตอร์ GPHEN (GPIO Pin High Detect Enable) ตามลำดับ ตรวจจับระดับสัญญาณ (Level Detect) แบ่งเป็น สัญญาณต่ำ หรือ ลอจิค 0 (Low Level Detect) สัญญาณสูง หรือ ลอจิค 1 (High Level Detect)
- การตรวจจับขอบสัญญาณ (Edge Detection) แบ่งเป็น

- * ขอบขาขึ้น (Rising) และขอบขาลง (Falling) เมื่อตรวจจับได้ จะส่งสัญญาณอินเทอร์รัพท์ เข้าสู่ซีพียูต่อไป
- * การตรวจจับสัญญาณทั้งขอบขาขึ้นและขอบขาลง รองรับชนิดซิงโครนัสและอะซิงโครนัส ชนิดซิงโครนัส (Sync) และอะซิงโครนัส (Async.) โดยการควบคุมที่รีจิสเตอร์ทั้งสี่ชนิด นี้ ได้แก่ รีจิสเตอร์ GPREN (GPIO Synchronous Rising Edge) และ รีจิสเตอร์ GPFEN (GPIO Synchronous Falling Edge) รีจิสเตอร์ GPAREN (GPIO Asynchronous Rising Edge) และ รีจิสเตอร์ GPAFEN (GPIO Asynchronous Falling Edge) เกิดเป็นสัญญาณ Interrupt รีจิสเตอร์ Pin Level (GPLEV) เพื่อบันทึกค่าอินพุทที่รับเข้ามา รีจิสเตอร์ Pin Event Detect Status (GPEDS) เพื่อบันทึกเหตุการณ์ที่รับเข้ามา

การทดลองที่ 8 ภาคผนวก H และ การทดลองที่ 9 ภาคผนวก I

6.12 การขัดจังหวะ (Interrupt)

การขัดจังหวะสำหรับ ARM Cortex A รุ่นต่างๆ อาศัยวงจรควบคุม เรียกว่า Generic Interrupt Controller (GIC) เพื่อรับการขัดจังหวะที่ซับซ้อน เนื่องจากชิปมีจำนวน ARM Cortex A ตั้งแต่ 2 คอร์ขึ้นไป และอุปกรณ์ที่หลากหลาย โดยทั้งหมดเชื่อมต่อกันด้วย ARM Advanced eXtensible Interface (AXI) ซึ่ง เป็นวงจรชนิดหนึ่ง มีโครงสร้างและโปรโตคอลที่ซับซ้อน ผู้อ่านสามารถค้นควารายละเอียดและวิธีการของ AXI เพิ่มเติมที่ https://en.wikipedia.org/wiki/Advanced_Microcontroller_Bus_Architecture

แต่หัวข้อนี้จะอธิบายโครงสร้างภายในของ Generic Interrupt Controller (GIC) และการเชื่อมโยงกับโมดูลอื่นๆ ตามรูปที่ 6.19

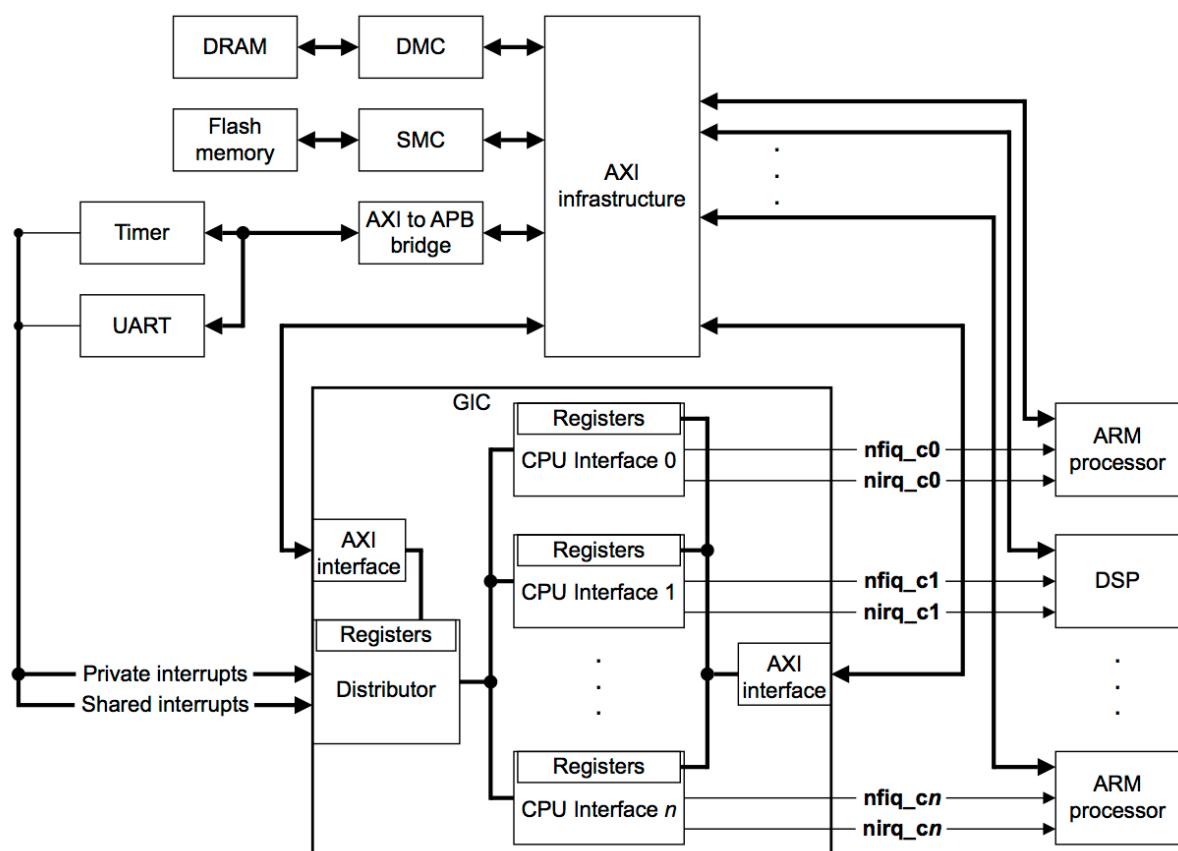


Figure 6.19: โครงสร้างภายในของ Generic Interrupt Controller (GIC) และการเชื่อมโยงกับโมดูลอื่นๆ กับ ARM Advanced eXtensible Interface (AXI) ที่มา: ?gic), APB (ARM Peripheral Bus),

รูปที่ 6.20 แสดงให้เห็นว่า การทำงานและโหมดแกรเวลาของ GIC เมื่อเกิดสัญญาณร้องขอ (Interrupt Request) คือ Interrupt M และ Interrupt N จำนวนสองสัญญาณ ในห้วงเวลาที่ต่อเนื่องกัน โดยที่ สัญญาณที่มากยิ่งมีความสำคัญ (Priority) เหนือกว่าสัญญาณที่ร้องขอ ก่อน

ควบเวลาต่างๆ ทำงานที่ขอบขั้นของสัญญาณคลื่นออกเท่านั้น

- T1 วงจร Distributor ตรวจจับการร้องขอของ Interrupt M
- T2 วงจร Distributor พักการร้องขอจาก Interrupt M

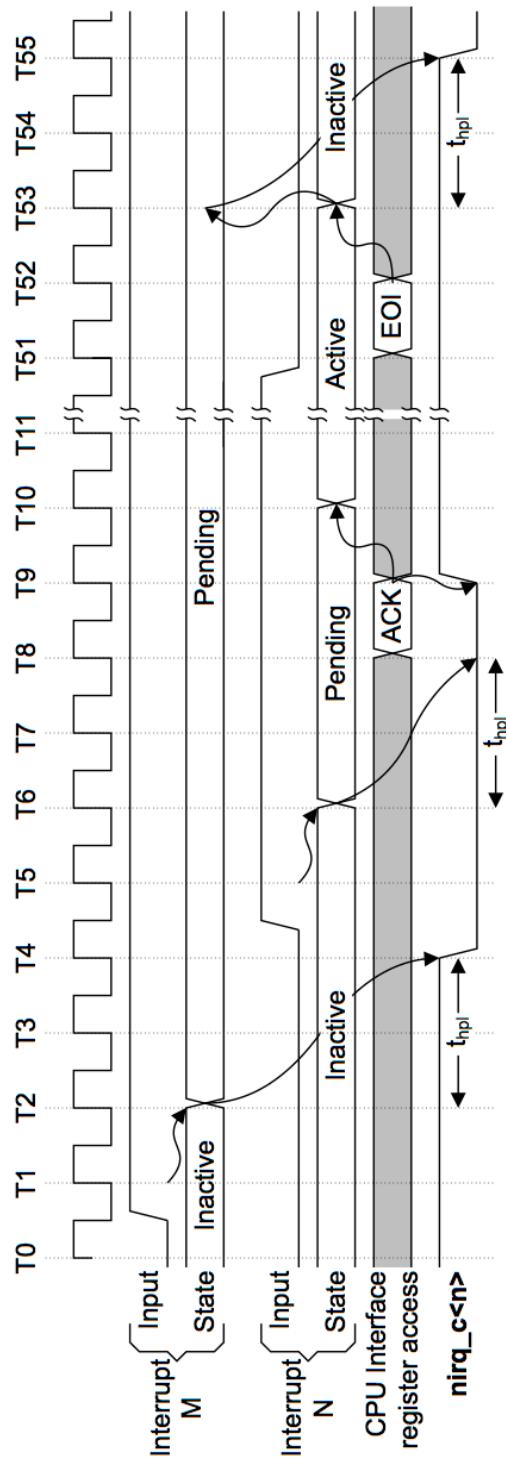


Figure 6.20: การทำงานและไดอะแกรฟเวลาของ Generic Interrupt Controller (GIC) ที่มา: ?gic

- T4 วงจรเชื่อมต่อกับ CPU ตั้งค่า Interrupt M กับสัญญาณร้องขอการขัดจังหวะ `nirq_c<n>`
- T5 วงจร Distributor ตรวจจับการร้องขอของ Interrupt N ซึ่งมีความสำคัญสูงกว่า Interrupt M จึงทำให้ต้องหยุดการทำงานของ Interrupt M ในcabเวลาถัดไป
- T6 วงจร Distributor พักการร้องขอจาก Interrupt N วงจรเชื่อมต่อกับ CPU ตั้งค่า INTID ของ Interrupt N กับสัญญาณร้องขอการขัดจังหวะ `nirq_c<n>` แทน

- T8 CPU ตอบสนองต่อสัญญาณร้องขอ ด้วยการส่งสัญญาณ ACK
- T9 CPU อ่านค่า Interrupt Acknowledge Register และตอบสนองต่อ $nirq_c< n>$ โดยเปลี่ยนค่าให้เป็น 1
- T10 วงจร Distributor ตั้งค่าสถานะของ Interrupt N เป็น Active และเปลี่ยนแปลงค่า Active Status Register
- T11-T50 CPU ตอบสนองโดยการทำงานที่ Interrupt Service Routine ของ Interrupt N หลังจากนั้นอุปกรณ์ยกเลิกการร้องขอ
- T51 CPU บันทึกค่ารีจิสเตอร์ End of Interrupt (EOI) ด้วยหมายเลข INTID ของ Interrupt N
- T52 วงจร Distributor ตั้งค่าสถานะของ Interrupt N เป็น Inactive และเปลี่ยนแปลงค่า Active Status Register.
- T53 วงจร Distributor แจ้งวงจรเข้มต่อ CPU ว่า interrupt M ยังคงอยู่และมีความสำคัญสูงที่สุดในเวลานี้
- T55 วงจรเข้มต่อ CPU ส่งสัญญาณร้องขอ $nirq_c< n>$ ให้กลับเป็น 0 เพื่อขัดจังหวะการทำงานของ CPU n เป็นลำดับถัดไป

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 การรับข้อมูลจากอุปกรณ์อินพุท DMA จะทำหน้าที่เคลื่อนย้ายข้อมูลจาก FIFO ของวงจรเชื่อมต่อไปยังหน่วยความจำหลัก และการส่งข้อมูลไปยังอุปกรณ์เอาท์พุทต่างๆ DMA จะทำหน้าที่เคลื่อนย้ายข้อมูลจากหน่วยความจำหลัก ไปยัง FIFO ของวงจรนั้นๆ

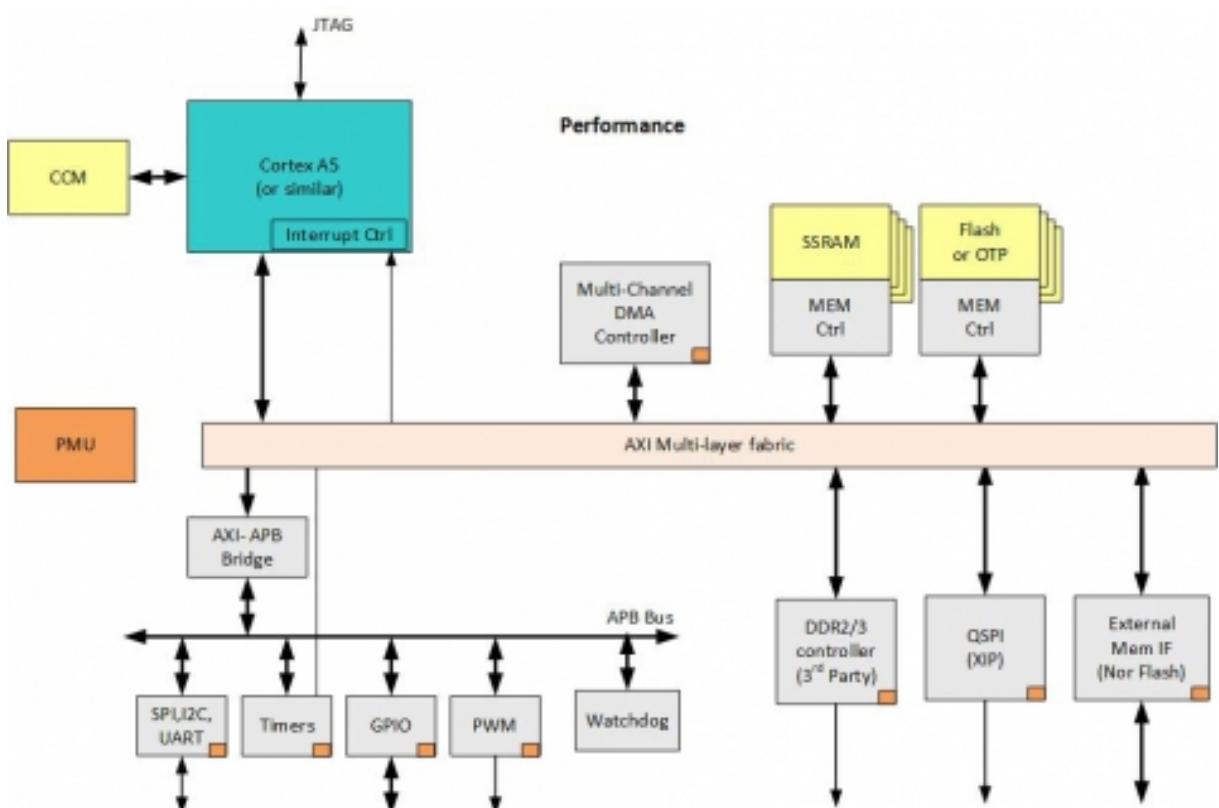


Figure 6.21: การเชื่อมต่อระหว่าง Cortex A5 และโมดูล DMA (Direct Memory Access) ด้วยบัส AXI (Advanced eXtensible Interface) ภายในชิป ที่มา: https://www.design-reuse.com/sip/blockdiagram/37237/20150520111508-main-Performance_blk.jpg

รูปที่ 6.21 แสดง การเชื่อมต่อระหว่าง Cortex A5 และโมดูล DMA (Direct Memory Access) ผ่าน บัส AXI (Advanced eXtensible Interface) ภายในชิป โดยสามารถรองรับขบวนการได้พร้อมกัน 16 ช่อง แต่ละช่องสามารถทำงานได้อิสระจากกัน

ขบวนการ DMA เริ่มต้นจาก DMA Controller อ่านค่า CB (Control Block) ในหน่วยความจำไปตั้ง ค่าริจิสเตอร์ CONBLK_AD ภายใน DMA ซึ่งที่ว่างอยู่ หลังจากนั้น วงจร DMA ซึ่งอยู่จะเริ่มทำงานด้วย ตนเอง เมื่อทำงานแล้วเสร็จ DMA จะส่งสัญญาณขัดจังหวะไปแจ้งเตือน CPU ว่างานเสร็จสิ้นแล้ว ผ่าน GIC ตามที่ได้กล่าวไว้แล้วในหัวข้อที่ 6.12

ข้อมูลแต่ละ CB ประกอบด้วยข้อมูลจำนวน 256 บิต หรือแบ่งเป็น 8×32 บิต ซึ่งตรงกับแอดเดรสของ DMA แต่ละช่องที่มีค่าห่างกัน 256 ไบต์

- 0x7E00 7000 สำหรับ DMA ช่อง 0
- 0x7E00 7100 สำหรับ DMA ช่อง 1
- 0x7E00 7200 สำหรับ DMA ช่อง 2
- ...

DMA แต่ละช่อง จะทำงานโดยการตั้งค่ารีจิสเตอร์ด้วย ชุดข้อมูล Control Block (CB) ภายในหน่วยความจำ CB แต่ละชุดจะเชื่อมโยงกันเป็นลิงค์ลิสต์ (Linked List) เมื่อการตั้งค่า CB ชุดแรกเสร็จสิ้น ค่า CB ชุดต่อไปจะถูกอ่านเพื่อไปตั้งค่าการทำงานชุดต่อไป โดยซีพียูไม่จำเป็นต้องเกี่ยวข้อง

6.14 ขา UART (Universal Asynchronous Receiver Trasmitter)

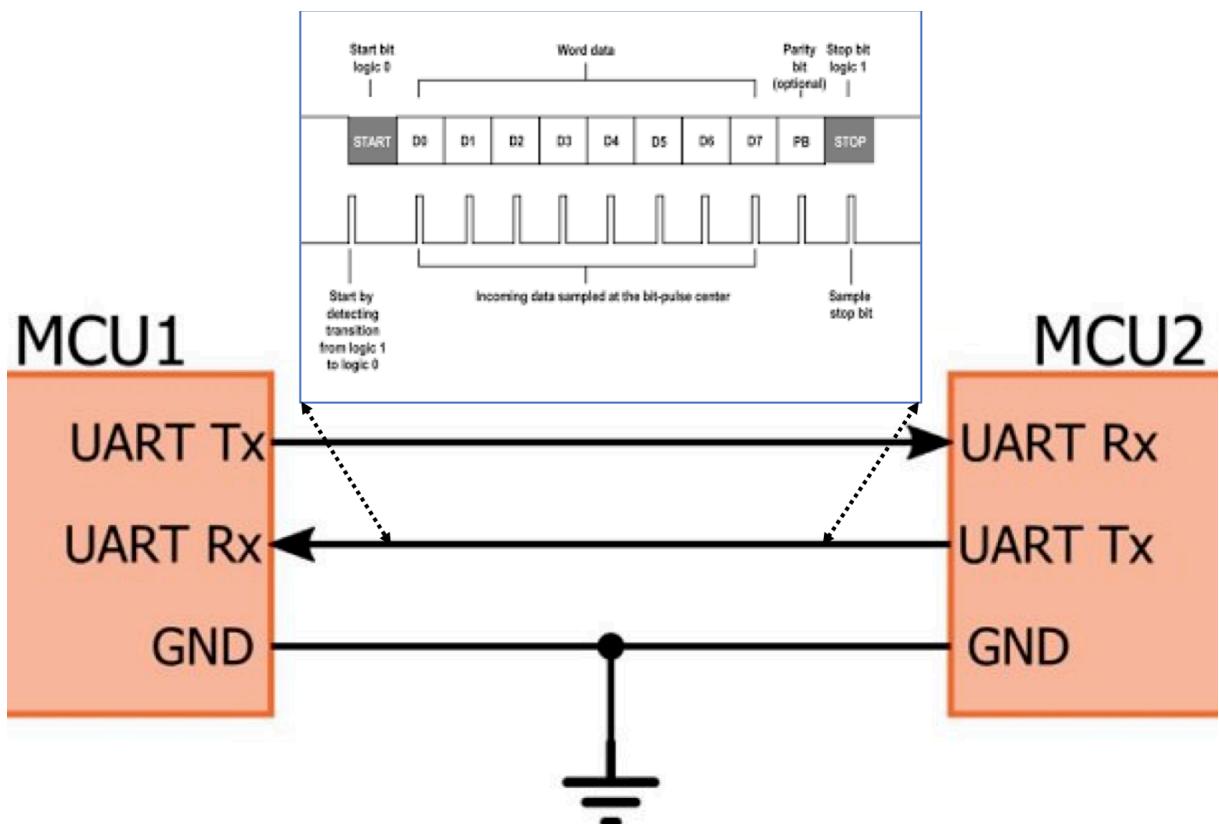


Figure 6.22: การเชื่อมต่อ MCU (MicroController Unit) สองตัวด้วย UART ที่มา: [ที่มา 1](#) และ [ที่มา 2](#)

UART คือ Universal Asynchronous Receiver Transmitter หมายถึง รูปแบบการส่งข้อมูล ที่ถูกกำหนดขึ้นมาเพื่อใช้รับส่งข้อมูลแบบ Asynchronous ซึ่งเป็นส่วนหนึ่งในการสื่อสารอนุกรม ข้อดีของการใช้ Asynchronous คือ เป็นการส่งข้อมูลที่ไม่ต้องใช้สัญญาณ Clock มาเป็นตัวกำหนดจังหวะการรับส่งข้อมูล แต่ใช้รูปแบบ Format ของสัญญาณที่ใช้ใน การรับส่งข้อมูลแทน

การรับส่งแบบ Async จะต้องกำหนด อัตราบิตเรตของการรับและส่งที่เท่ากัน เรียกว่า อัตราเร็วในการรับส่ง เรียกว่า อัตราบอตเดต (baud rate) มีหน่วยเป็น บิตต่อวินาที (Bits Per Second) การส่งจะต้องเริ่มจากการส่ง START ('0') บิตก่อนด้วยระยะเวลาเท่ากับความยาว 1 บิตเท่านั้นตามด้วย บิตข้อมูล LSB (Least Significant Bit) จะส่งก่อน ตามด้วยบิตที่สูงขึ้น จนถึงบิตข้อมูล MSB (Most Significant Bit) ซึ่งจำนวนบิตของข้อมูลว่าเป็น 5 - 8 บิต ต่อหนึ่งตัวอักษร พาริตี้บิต (Parity) ว่าเป็น None (ไม่มี), Odd (คี่) หรือ Even (คู่) Parity และปิดท้ายด้วย STOP ('1') บิต โดยจะมีความยาว 1, 1.5 หรือ 2 บิต ตามที่กำหนดไว้

โมดูล UART 2 ตัว ในชิป BCM2837 ประกอบด้วย mini UART ซึ่งเป็นโมดูล UART สำหรับใช้งานอย่างง่าย และ PL011 UART ซึ่งเป็นโมดูลเต็มประสิทธิภาพ ออกแบบโดยบริษัท ARM มีบีฟเฟอร์ทำหน้าที่ (FIFO: First In First Out) ด้านส่งขนาด 16 คูณ 8 ตัวอักษร ด้านรับขนาด 16 คูณ 12 ตัวอักษร สามารถกำหนดอัตราบอตได้ตามสมการที่คำนวณ สามารถตรวจสอบจับ สถา๊ร์ทบิต ที่ไม่ปกติได้ สามารถกำหนดจำนวน

บิตของข้อมูลได้ตั้งแต่ 5-8 บิต ชนิดของพาริตีบิต ดังนี้ ไม่มีพาริตีบิต (No Parity) มีพาริตีบิตโดยเลือกชนิด พาริตี้คู่ (Even Parity) หรือ พาริตี้คี่ (Odd Parity) อย่างไดอย่างหนึ่ง ความยาวของสต็อปบิต ขนาด 1 หรือ 2 บิต

mini UART เหมาะกับการเชื่อมต่อเทอร์มินัลหรือคอนโซล (Console) ง่ายๆ ขนาดหมายเลข 8 และ 10 ตามตารางที่ [6.4](#) สามารถกำหนดจำนวนบิตของข้อมูลได้เพียง 7 หรือ 8 บิตเท่านั้น สต็อปบิตมีความยาว เท่ากับ 1 บิต ไม่มีพาริตีบิต มีบัฟเฟอร์เพียง 8 ตัวอักษรเท่านั้น อัตราบอตเรตสามารถคำนวณได้จากความ ถี่สัญญาณคลือกของระบบ ตามสมการที่ [6.1](#)

$$f_{baud} = \frac{f_{CLK}}{8 \times (r + 1)} \quad (6.1)$$

โดย f_{baud} คือ อัตรา baud rate หน่วยเป็น bps f_{CLK} คือ ความถี่สัญญาณคลือกของบอร์ด Pi3 หน่วย เป็น เฮิรทซ์ r คือค่าตัวหารที่เปลี่ยนแปลงเพื่อกีบใน รีจิสเตอร์

การทดลองที่ 10 ภาคผนวก [J](#)

6.15 ขา SPI (Serial Peripheral Interface)

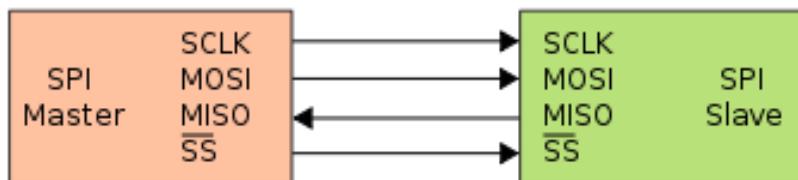


Figure 6.23: การเชื่อมต่อแบบ SPI: Serial Peripheral Interface ประกอบด้วยมาสเตอร์ (Master) และ-slave (Slave) ที่มา: https://en.wikipedia.org/wiki/Serial_Peripheral_Interface_Bus

รูปที่ 6.23 ในบอร์ด Pi3 ผู้พัฒนาทำการเชื่อมสัญญาณจากชิป BCM2837 มาที่ขาหมายเลข 19, 21, 23, 24 และ 26 ตามตารางที่ 6.4 สำหรับเชื่อมต่อซีพียูกับไอซีภายนอก เช่น EEPROM, จอ LCD ที่ต้องการรับส่งข้อมูลความเร็วสูง จังหวะการรับและส่งข้อมูลแต่ละ比特จะซิงค์ครอนซ์กับสัญญาณคล็อก

เป็นการเชื่อมต่อสื่อสารแบบอนุกรมโดยอาศัยสัญญาณนาฬิกาเป็นตัวกำหนดจังหวะการรับส่งข้อมูล (Synchronous) ที่สามารถส่งข้อมูลไปยังปลายทางและรับข้อมูลจากปลายทางกลับมาในครั้งเดียวกัน (Full Duplex)

SPI แบ่งอุปกรณ์ออกเป็น 2 ฝั่ง คือ Master เป็นตัวควบคุมการรับส่งข้อมูลโดยในที่นี้คือไมโครคอนโทรลเลอร์ กับ Slave เป็นอุปกรณ์ที่รอรับคำสั่ง โดย Master 1 ตัวมี Slave ได้มากกว่า 1 ตัว SPI ใช้สายสัญญาณทั้งหมด 4 เส้นดังนี้

1. MOSI (Master Out Slave In) ใช้สำหรับให้ Master ส่งข้อมูลไปให้ Slave
2. MISO (Master In Slave Out) ใช้สำหรับให้ Slave ส่งข้อมูลไปให้ Master
3. SCLK (Clock) ใช้สำหรับส่งสัญญาณคล็อกจาก Master ไปยัง Slave
4. CS (Chip Select) หรือ SS (Slave Select) ใช้สำหรับส่งสัญญาณกระตุ้นจาก Master ไปยัง Slave

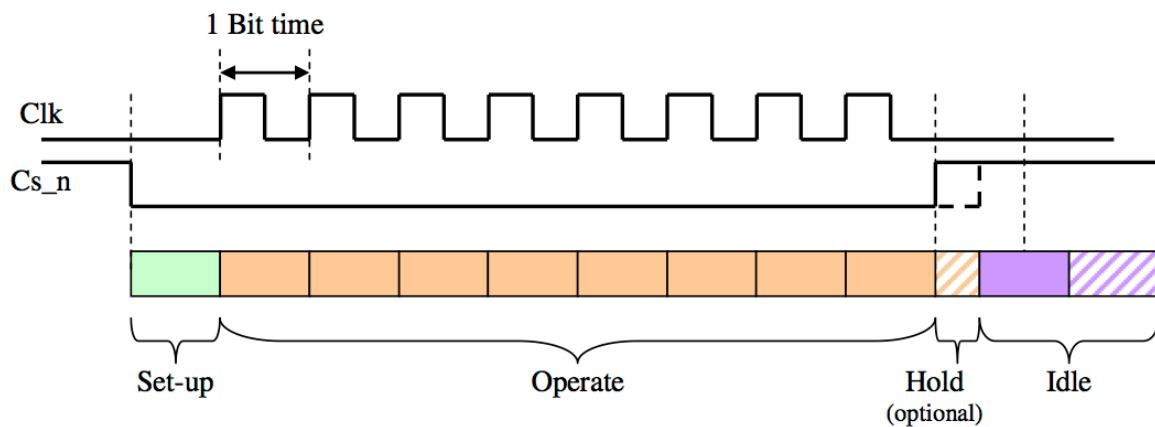


Figure 6.24: การส่งข้อมูลจาก SPI มาสเตอร์ (Master) และ สเลฟ (Slave) ด้วยการซิงโครไนซ์กับสัญญาณ CS (Chip Select) และคลีอก (CLK)

รูปที่ 6.24 การส่งข้อมูลจาก SPI มาสเตอร์ (Master) และ สเลฟ (Slave) ด้วยการซิงโครไนซ์กับสัญญาณ CS (Chip Select) และคลีอก (CLK) หลังจากที่ $CS = > 0$ คือการเตรียมตัว (Set up) คลีอก ลูกที่ 1 จะเริ่มนับเมื่อคลีอกเกิดการเปลี่ยนแปลง ข้อมูลบิตแต่ละบิตจะถูกส่งออกไป จนบิตสุดท้าย ในช่วง Operate จำนวนคลีอกจึงหยุดเปลี่ยนแปลง หลังจากนั้น สัญญาณ CS = > 1 เข้าสู่ช่วง Idle

สัญญาณ SPI ใช้สำหรับการเชื่อมต่อระหว่างชิปแบบอนุกรมระยะทางสั้น ด้วยความเร็วสูง ตามความต้องการของสัญญาณคลีอกที่ใช้จันถึงหลายร้อยเมกะเอิร์ตซ์ การรับส่งข้อมูลอาศัยการซิฟต์ข้อมูลบิตที่มีค่าประจำตำแหน่งสูง (MSB) ก่อนจันถึงบิตที่มีค่าประจำตำแหน่งน้อยที่สุด (LSB) มีลักษณะการรับส่งแบบ Master-Slave เท่านั้น โดย Master จะเริ่มต้นการทำงานด้วยการกำหนดให้สัญญาณ Slave Select หรือ Chip Select = 0 เพื่อกระตุ้นการทำงานของ Slave หรือ Chip ตัวนั้น หลังจากนั้น คลีอกจะเกิดการเปลี่ยนแปลงเพื่อให้เกิดการรับส่งข้อมูล ตามขอบขาขึ้น หรือ ขอบขาลง

Master สามารถเชื่อมต่อกับ Slave ได้มากกว่า 1 ตัว โดยทุกตัวจะใช้ขา MOSI MISO และ SCLK ร่วมกัน แล้ว Master จะส่งสัญญาณที่ขา CS เพื่อเลือกว่าในขณะนั้น Master ติดต่อกับ Slave ตัวใด ในการรับ/ส่งข้อมูล 1 รอบ จะทำได้ตั้งแต่ 1-32 บิต สามารถควบคุม Slave ได้พร้อมๆ กันจำนวน 3 ตัว

By default, the SPI master drive is disabled, but it can be enabled via the command
`raspi -config`

การทดลองที่ 11 ภาคผนวก K

6.16 ขา I2C ((Inter-Integrated Circuit))

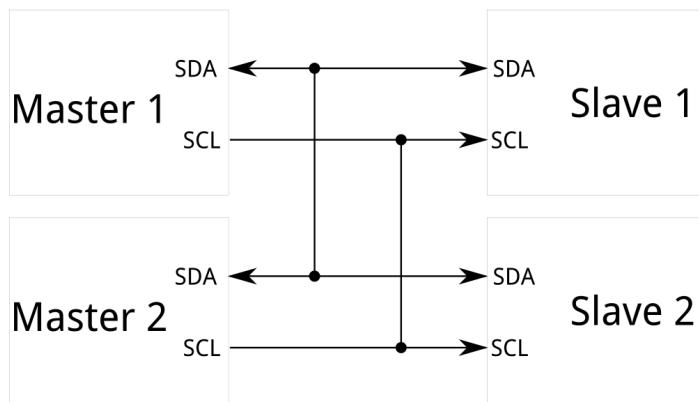


Figure 6.25: การเชื่อมต่อแบบ I2C (Inter-Integrated Circuit) ประกอบด้วยมาสเตอร์ (Master) และ สลave (Slave) ที่มา: <http://www.thaimicrotron.com/CCS-628/Referrence/I2CBUS.htm>

มาตรฐานสัญญาณ I2C หรือ Inter-Integrated Circuit คือ วิธีการสื่อสารระหว่างไอซี หรือ ไมโครคอนโทรเลอร์ ถูกคิดค้นโดยบริษัท Phillips Semiconductor ซึ่งปัจจุบันคือบริษัท NXP Semiconductor โดยคิดค้นค้างแต่ปี 1982 และได้มีการพัฒนามาเรื่อยๆ จนกลายเป็นprotoคอลมาตราฐานที่ใช้ในการติดต่อกันระหว่างอุปกรณ์ในปัจจุบัน I2C อ่านว่า ไอสแควร์ซี หรือ ไอทูซี I2C เป็นวิธีการส่งข้อมูลกันระหว่างมาสเตอร์และสลaveคล้ายกับการเชื่อมต่อ SPI ไมโครคอนโทรเลอร์ คือ มาสเตอร์สามารถติดต่อกับสลaveได้หลายตัว แต่มาสเตอร์จะต้องต่อกับสลaveตัวใดตัวหนึ่งเท่านั้น ในบอร์ด Pi3 ผู้พัฒนาขยายการเชื่อมต่อจากชิป BCM2837 มาที่ขาหมายเลข 3 และ 5 ตามตารางที่ 6.4 ไปยังไอซีภายนอกด้วยสัญญาณ I2C

มีใช้ในสัญญาณ HDMI, DSI, CSI, I2S ในหัวข้อที่ผ่านมา

I2C Bus ย่อมาจาก Inter Integrate Circuit Bus (IIC) นิยมเรียกว่า BUS (ไอ-แสគ-ซี-บัส) เป็นการสื่อสารอนุกรม แบบซิงโครนัส (Synchronous) โดยใช้สายสัญญาณเพียง 2 เส้นเท่านั้น คือ serial data (SDA) และสาย serial clock (SCL) ซึ่งสามารถ เชื่อมต่ออุปกรณ์ จำนวนหลายตัว เข้าด้วยกันได้

การรับ-ส่งข้อมูลแบบ I2C BUS ไมโครคอนโทรลเลอร์จะเริ่มต้นการส่งข้อมูล

- สถานะเริ่มต้น (START Conditions) เพื่อแสดงการขอใช้บัส
- แล้วตามด้วย รหัสควบคุม (Control Byte) ซึ่งประกอบ ด้วยรหัส ประจำตัวอุปกรณ์ Device ID ,Device Address ,และ Mode ในการเขียนหรืออ่านข้อมูล
- เมื่ออุปกรณ์ รับทราบว่า MCU ต้องการ จะติดต่อด้วยก็ต้องส่งสถานะรับรู้ (Acknowledge) หรือแจ้งให้ MCU รับรู้ว่าข้อมูลที่ได้ส่งมามีความถูกต้อง
- และเมื่อสิ้นสุดการส่งข้อมูล MCU จะต้องส่ง สถานะสิ้นสุด (STOP Conditions) เพื่อบอกกับอุปกรณ์ว่า สิ้นสุดการใช้บัส

6.17 พัลซ์วิดจ์มอดูเลชัน (Pulse Width Modulation)

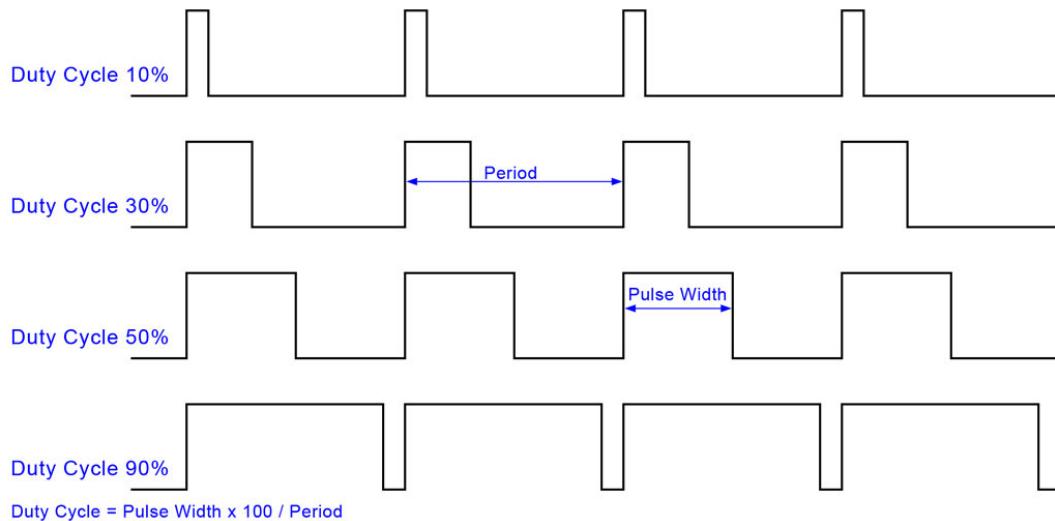


Figure 6.26: สัญญาณ PWM ที่ดิวตี้ไซเคิล $D_{cycle}=10\% 30\% 50\%$ และ 90% ที่มา: <https://protostack.com.au/2011/06/atmega168a-pulse-width-modulation-pwm/>

การทดลองที่ 12 ภาคผนวก L การควบคุมกำลังไฟฟ้าด้วยสัญญาณ PWM พัลซ์วิดจ์มอดูเลชัน เรียกย่อๆ ว่า PWM มาจากคำว่า Pulse Width Modulation. ใช้สำหรับควบคุมกำลังไฟฟ้าที่จะจ่ายให้กับอุปกรณ์ใดๆ เพื่อควบคุมความเร็ว ซึ่งนิยมใช้สัญญาณรูปพัลซ์หรือสัญญาณคลื่น ซึ่งมี $D_{cycle} = 50\%$ ในรูปที่ 6.26 อัตราส่วนระหว่าง Pulse Width และ คาบเวลา (Period) เรียกว่า ดิวตี้ไซเคิล (Duty Cycle: D_{cycle}) $=10\% 30\% 50\%$ และ 90%

$$D_{cycle}(\%) = \frac{T_{on}}{T_{on} + T_{off}} \times 100\% \quad (6.2)$$

T_{on} = ความกว้างของพัลซ์ (Pulse Width)

$T_{on} + T_{off}$ = คาบเวลาของพัลซ์ $= 1/f_{pwm}$ ความถี่ที่สร้างขึ้น

ที่ 100% เป็นการจ่ายกำลังไฟฟ้าได้สูงสุด และลดลงตามลำดับ หากจ่ายให้กับหลอด LED หลอดจะให้ความสว่างเต็มที่ และหรี่ลงตามลำดับ หากควบคุมเวลาของ PWM จะเห็นเป็นหลอด LED กระพริบได้ เมื่อ D_{cycle} ต่ำลง

ผู้อ่านสามารถใช้คำสั่งในไลบรารี **wiringPi** เพื่อสร้างสัญญาณ PWM ได้โดยใช้ขา GPIO ในการทดลองที่ 12 ภาคผนวก L

The PWM controller incorporates the following features:

- Two independent output bit-streams, clocked at a fixed frequency.
- Bit-streams configured individually to output either PWM or a serialised version of a 32-bit word.
- PWM outputs have variable input and output resolutions.
- Serialise mode configured to load data to and/or read data from a FIFO storage block, which can store up to eight 32-bit words.
- Both modes clocked by `clk_pwm` which is nominally 100MHz, but can be varied by the clock manager.

6.18 แหล่งจ่ายไฟ (Power Supply) ของบอร์ด Pi3

แหล่งจ่ายไฟของบอร์ดได้จากการรับไฟฟ้ากระแสตรงความต่างศักย์ 5.0 โวลท์ จากหัวคอนเนคเตอร์ Micro USB จะต้องจ่ายกระแสไฟฟ้าได้ไม่น้อยกว่า 1.0 แอมป์ ทั้งนี้ เพื่อให้บอร์ด Pi3 สามารถทำงานได้เต็มประสิทธิภาพ กระแสจะไหลผ่านพิวส์และไดโอด เพื่อจ่ายไฟให้กับบอร์ด Pi3 ความต่างศักย์ 5 โวลท์ แล้วจึงแปลงให้ลดลง (Step Down) เป็น 3.3 และ 1.8 โวลท์ด้วย PAM2306 และ 1.2 โวลท์ด้วย RT8088A

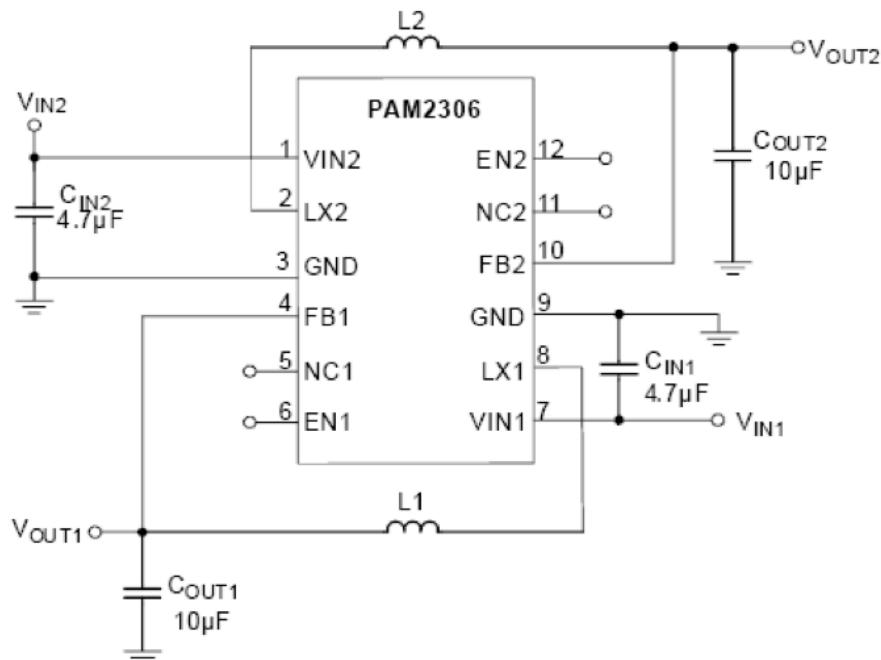


Figure 6.27: การแปลงไฟกระแสตรง 5 โวลท์เป็นไฟกระแสตรงด้วยไอซี PAM 2306 DC-DC Covertor คู่กำหนดค่าเอาท์พุทด้วยค่าตัวหนี่ยวนำ L_1 และ L_2 หน่วยเป็น ไมโครhenri ที่มา: [Diodes \(2012\)](#)

ไอซี PAM2306 จะทำหน้าที่แปลงไฟกระแสตรง 5.0 โวลท์ ให้เป็นไฟกระแสตรงความต่างศักย์ 3.3 โวลท์ เพื่อเลี้ยงชิพต่างๆ และวงจรทั้งหมด PAM2306 รองรับอินพุตตั้งแต่ 2.5V ถึง 5.5V โดยสามารถแปลงเอาท์พุทโวลเทจได้เป็นสองระดับพร้อมๆ กัน ซึ่งผู้ออกแบบสามารถเลือกได้ดังนี้ 3.3V, 2.8V, 2.5V, 1.8V, 1.5V, 1.2V หรือใช้ความต้านทานปรับค่าได้ PAM2306 ทำงานในลักษณะ DC-DC converter ควบคุมการทำงานแบบ Step Down ด้วยกระแส PWM เพื่อให้เกิดเสถียรภาพ และรักษาอายุการใช้งานของแหล่งจ่ายไฟ เมื่อต้องจ่ายกระแสมากขึ้น จนถึง โดยใช้หลักการ Pulse-Skipping Modulation (PSM) เพื่อลดกระแสในขณะที่จ่ายกระแสสูง

ขา V_{IN1} จะถูกแปลงให้มีความต่างศักย์ลดลงเป็น V_{OUT1} ขา V_{IN2} จะมีลักษณะในทำนองเดียวกัน โดยควบคุมการเปิดปิดด้วยขา EN1 (Enable1) และ EN2 (Enable2) ตามลำดับ ค่าโวลเทจทั้ง V_{OUT1} และ ค่าโวลเทจทั้ง V_{OUT2} ถูกกำหนดที่หมายเลขตัวถังของไอซีมาจากโรงงานผลิต แต่ผู้ออกแบบต้องเลือกค่าตัวหนี่ยวนำให้เหมาะสม กำหนดจากค่าตัวหนี่ยวนำ L_1 และ L_2 หน่วยเป็น ไมโครhenri ได้ตามตารางที่ [6.6](#)

Table 6.6: ตารางเลือกค่า V_{OUT} และค่าตัวเหนี่ยวนำ L ที่มา: [Diodes \(2012\)](#)

V_{OUT} (โวลต์)	L (μH)
1.2	2.2
1.5	2.2
1.8	2.2
2.5	4.7
3.3	4.7

ความต้องการด้านกำลังไฟสำหรับบอร์ด Pi3 มูลนิจได้ระบุว่าบอร์ดต้องการกระแส 700 มิลลิแอมป์ ถึง 2.5 แอมป์ ขึ้นอยู่กับโมเดลที่ใช้ และ การใช้งานจริง ซึ่งแหล่งจ่ายไฟขนาด 5 โวลท์ 1 แอมป์สามารถใช้งานได้ดี โดยเข้มต่อกับอุปกรณ์ เช่น เม้าส์ คีย์บอร์ด และ WiFi ผ่าน USB แหล่งจ่ายไฟควรมีความต่างศักย์ $5 \pm 0.25\text{V}$ ที่กระแสกำหนดไว้บนอุปกรณ์ เช่น 1 แอมป์ หรือมากกว่า ผู้ใช้ไม่ต้องกังวลว่าการจ่ายกระแสมากเกินไป จะเป็นผลเสีย เพราะบอร์ดจะบริโภคกระแสเท่าที่จำเป็น และมีพิวร์ส์ค้อยช่วยจำกัดกระแส 2.5 แอมป์ นอกจากนี้ บอร์ด Pi3 และ Pi2 โมเดล B+ มีชิพที่ทำหน้าที่เฝ้าระวังโวลเตจ หมายเลข APX803 กรณีที่ความต่างศักย์อยู่ในช่วง 4.63 ± 0.07 โวลท์ไฟ LED สีแดงจะสว่างขึ้น แต่หากความต่างศักย์ตกหรือเกินช่วงนี้ ไฟ LED สีแดงจะดับเพื่อเตือนให้ผู้ใช้ทราบ ผ่านทางหน้าจอ แต่ยังใช้งานได้ตามปกติ

ข้อควรระวังเรื่องแหล่งจ่ายไฟของบอร์ด Pi3 คือ ขนาดและคุณภาพของสายและหัวไมโคร USB สายที่มีขนาดเล็กเกินไปจะทำให้ความต้านทานสูง เมื่อกระแสไหลผ่านจะทำให้เกิด โวลเตจตกคร่อมในสายสูงขึ้น จนทำให้บอร์ดได้รับโวลเตจไม่สูงพอ จนส่งผลถึงศักย์ด้าน เอาท์พุทที่ 3.3 โวลท์ของ PAM2306

6.19 สรุป

โครงสร้างด้านอินพุท/เอาท์พุท และหน่วยสำรองข้อมูลของเครื่องคอมพิวเตอร์เข้มด้วยสาย ระบบบัส ก่อให้การติดต่อกับอุปกรณ์อินพุท/เอาท์พุท นิยมใช้ Memory Mapped I/O, ก่อให้การทำ Interrupt และการทำ Direct Memory Access ร่วมกับ หน่วยสำรองข้อมูลที่สำคัญตั้งแต่อดีตจนถึงปัจจุบัน คือ ฮาร์ดดิสก์ ซึ่งมีประสิทธิภาพสูงขึ้น หมุนด้วยความเร็วสูงขึ้น อ่านเขียนได้เร็วขึ้น และมีความเชื่อมั่นเพิ่มขึ้น ด้วยการใช้ RAID หน่วยความจำชนิด SSD มีแนวโน้มเป็นทางเลือกทดแทนฮาร์ดดิสก์ เพราะ