

องค์ประกอบของเครื่องคอมพิวเตอร์
และภาษาแอสเซมบลี:
ARM และ RaspberryPi3

บทที่ 4 ภาษาแอสเซมบลีของ ARM เวอร์ชัน 32 บิต

ผศ.ดร.สุรินทร์ กิตติธรกุล
ภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

สารบัญ

- บทที่ 1 บทนำ
- บทที่ 2 ข้อมูลและคณิตศาสตร์ในคอมพิวเตอร์
- บทที่ 3 ยาร์ดแวร์และซอฟท์แวร์ของคอมพิวเตอร์
- บทที่ 4 ภาษาแอสเซมบลีของ ARM เวอร์ชัน 32 บิต
- บทที่ 5 หน่วยความจำล้ำตัวชี้
- บทที่ 6 อุปกรณ์/วงจรอินพุตและเอาท์พุต
- บทที่ 7 อุปกรณ์รักษาข้อมูลและระบบไฟล์

ពេលវេលាដើម្បីយកប័ត់លែង B (branch)

↳ ដើម្បី branch ទ្វាយត្រូវបាន binary, assembler ត្រួរពីនិរន្តរ offset តាម current instruction ហើយជាលើ destination Instruction

↳ ផ្តល់ offset តាម encode នីមួយៗ

↳ ការចូលរួមទៅលើកើតឡើ R15 + offset → ពារសំណើគំនួរបញ្ជូនកើតឡើ → make sense
រួមទៅការបង្កើត និង → end

↳ add ពីរបន្ទាន់ការបង្កើតការចូលរួម

↳ the address of the instruction immediately following by BL , arm save that into R14

↳ so to return from subroutine , the program simply has to move R14 back to R15

នឹងធ្វើដឹងទីនេះ ? → mreturn និង នាំវា

↳ ការចូលរួមនឹងបានបង្ហាញ arm នៃចុច្ចិនិងការចូលរួម
↳ នៅលើការបង្កើតការចូលរួម

4.9 อุปกรณ์และวิวัฒนาการของชุดคำสั่ง ARM

ARM processors

Early ARM processors
(ARMv1 – ARMv3)

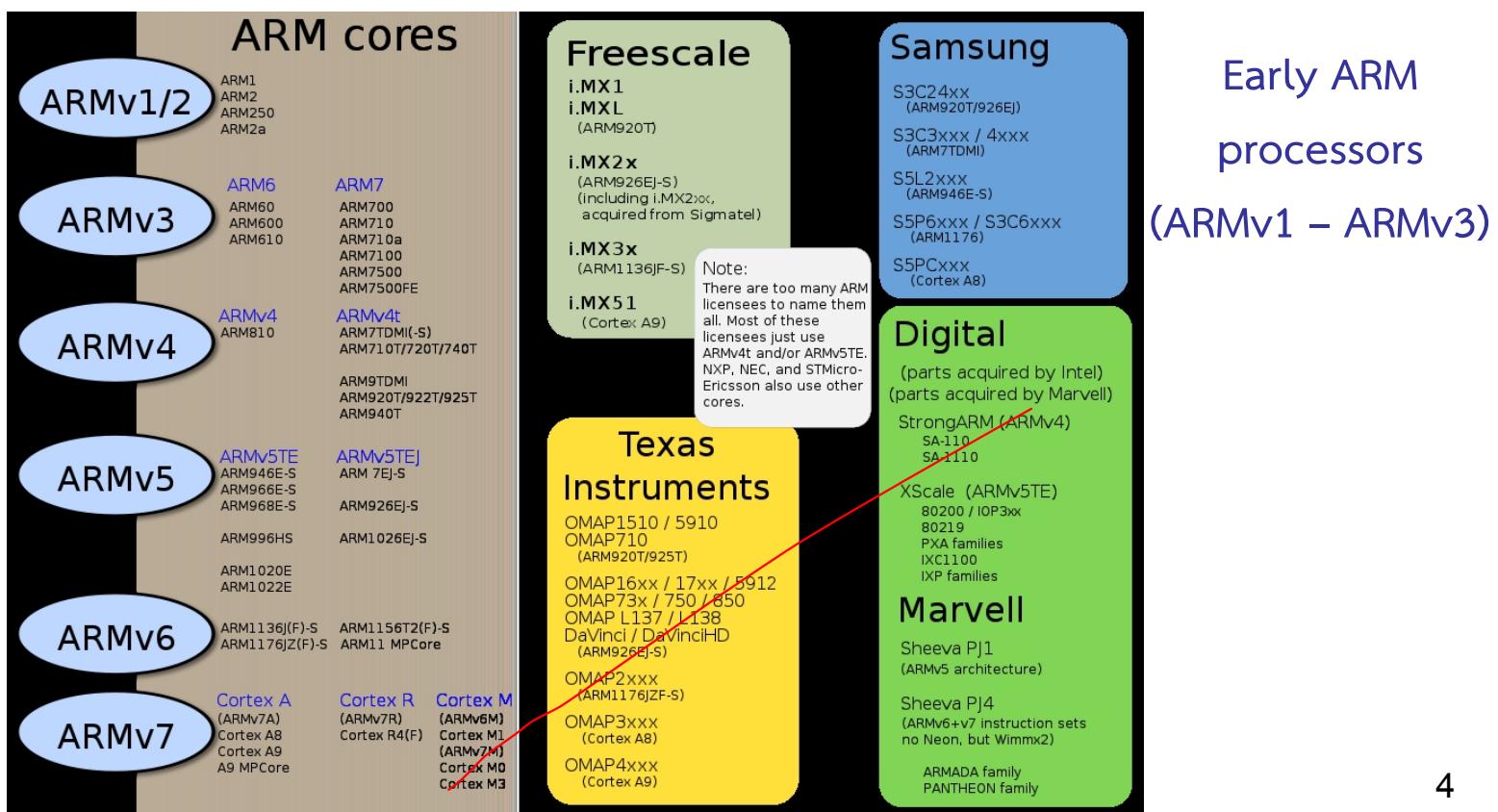
Advanced ARM processors
(ARMv4 – ARMv6)

Recent ARM Cortex processors
(ARMv7 – ARMv8)

- ARM designed until now **eight ISA versions**, designated as **ARMv1 – ARMv8**, described in the related Architecture Reference Manuals.
- Subsequently, we give an overview of ARM's processor lines divided into three groups, according their underlying ISAs, as follows.

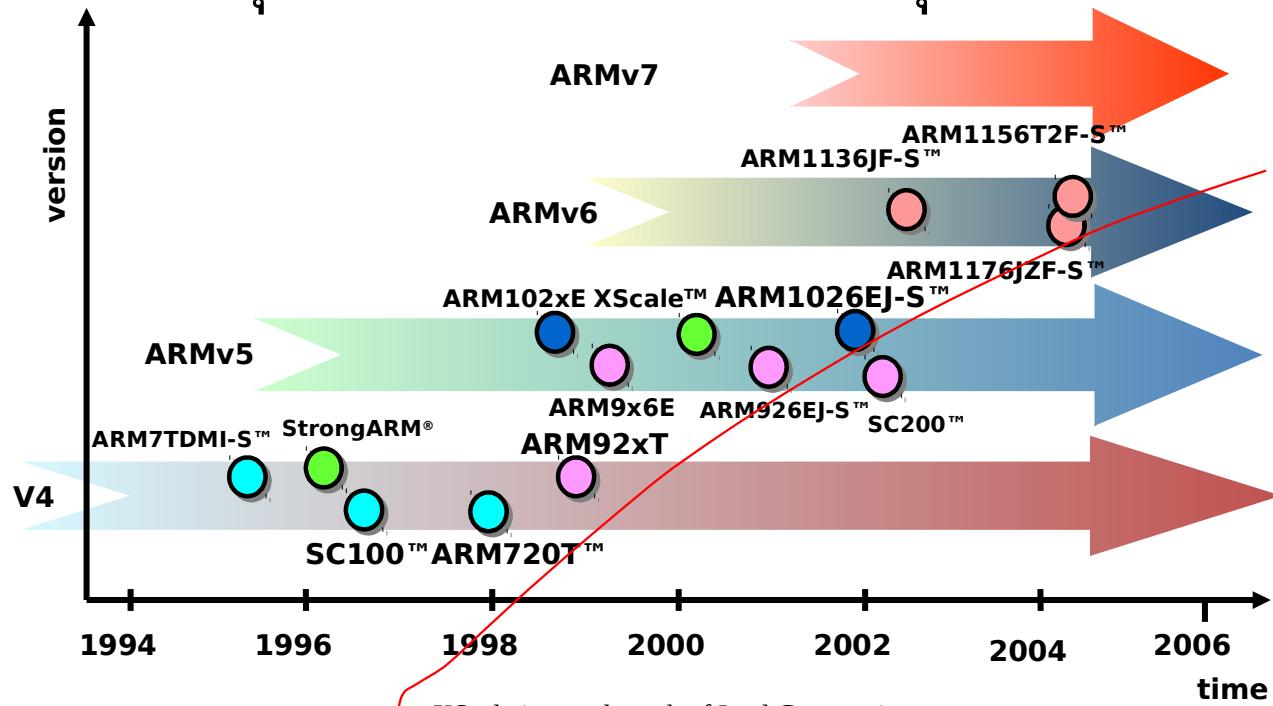
Computer Organization & ARM Assembly Language: RaspberryPi3, Surin K., KMITL

3



4

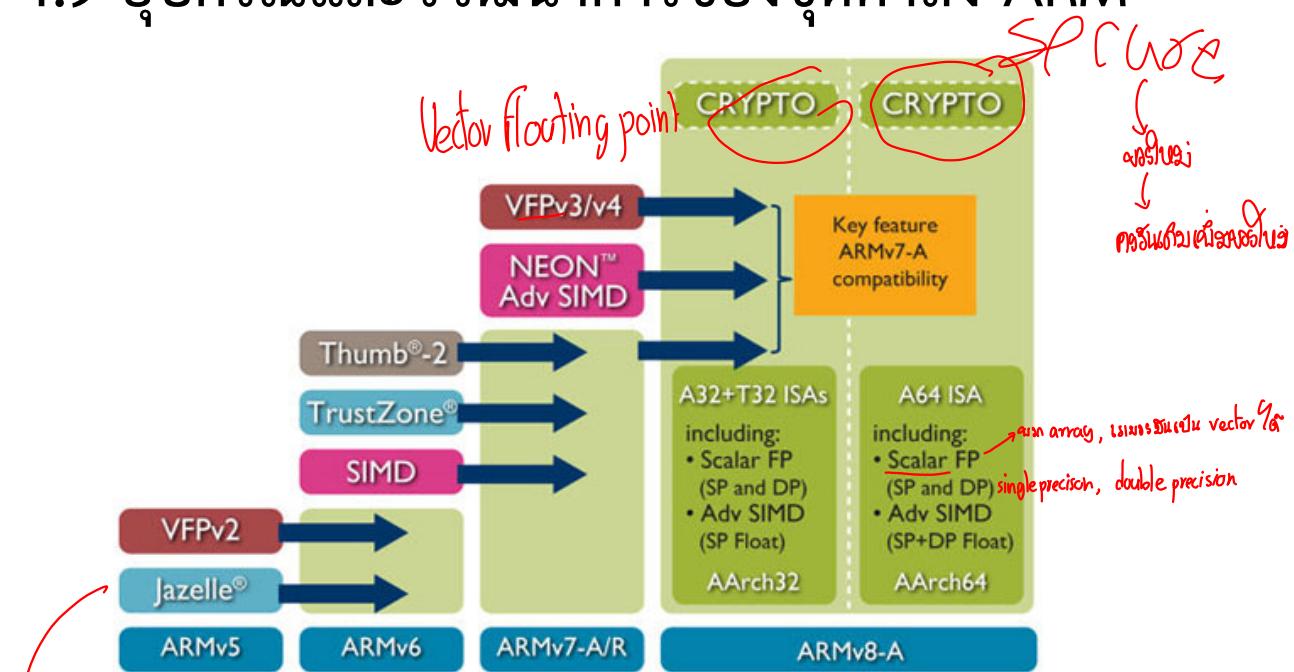
4.9 อุปกรณ์และวิวัฒนาการของชุดคำสั่ง ARM



Computer Organization & ARM Assembly Language: RaspberryPi3, Surin K., KMITL

5

4.9 อุปกรณ์และวิวัฒนาการของชุดคำสั่ง ARM

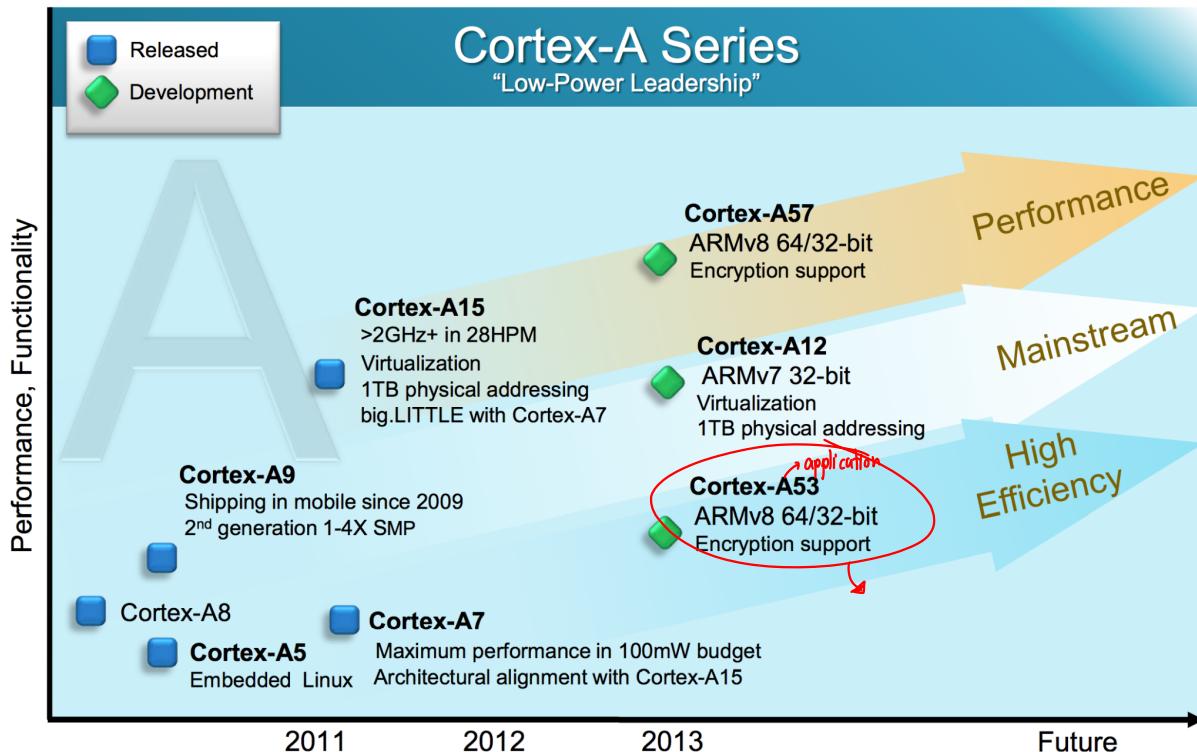


Computer Organization & ARM Assembly Language: RaspberryPi3, Surin K., KMITL

6

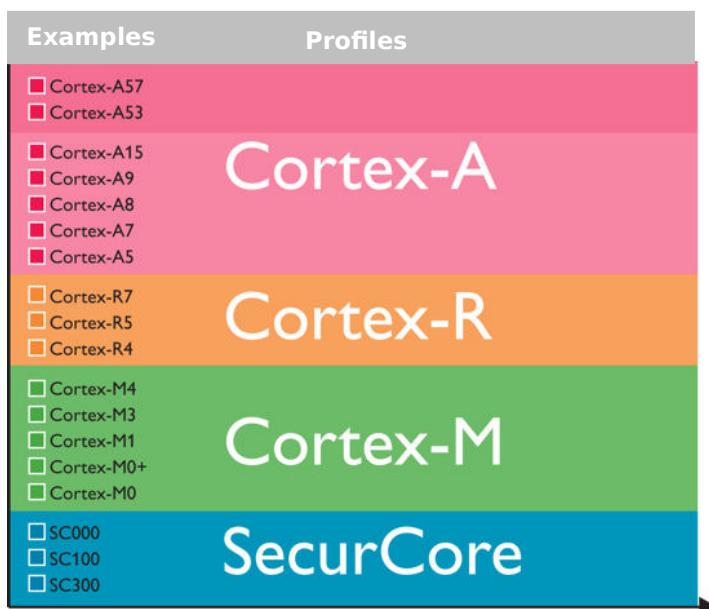
hardware ต้อง execute java ที่เป็น machine language ของ arm จึงทำได้

Recent ARM Cortex-A processors



7

Recent ARM processors (ARMv7 and ARMv8)



Cortex-A profile

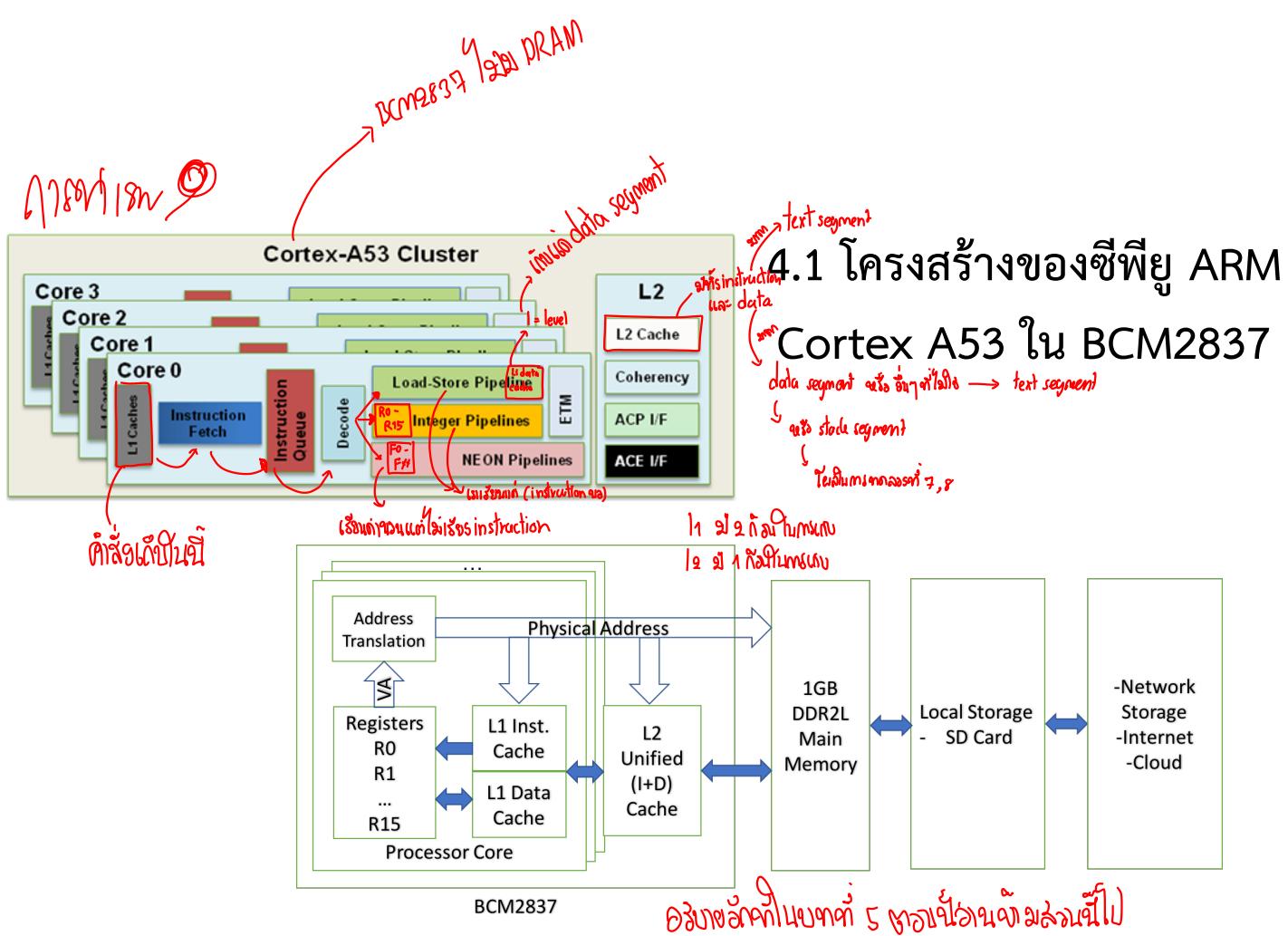
It aims at **high-end applications** running open and **complex OSs**, like smartphones, tablets, netbooks, eBook readers.

4.9 อุปกรณ์และวิวัฒนาการของชุดคำสั่ง ARM

	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%	

Computer Organization & ARM Assembly Language: RaspberryPi3, Surin K., KMITL

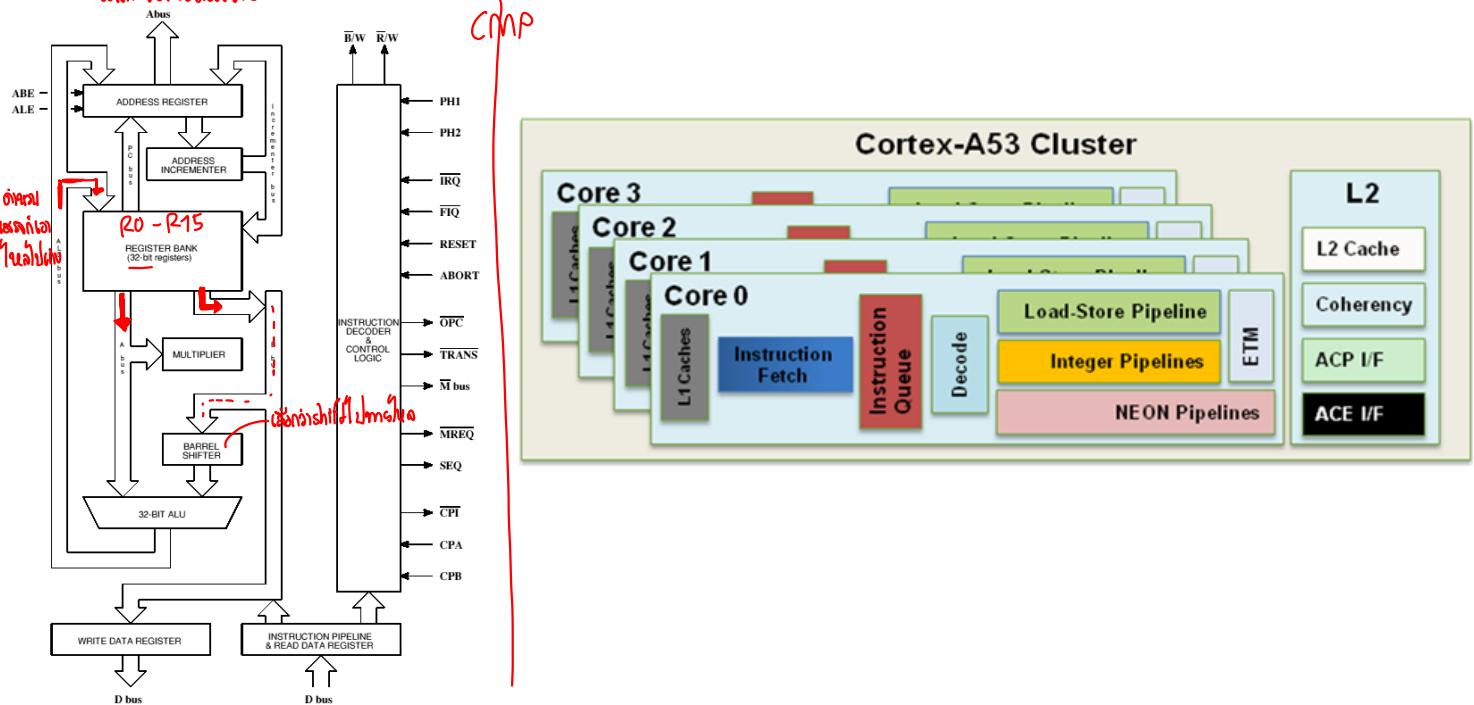
9



Computer Organization & ARM Assembly Language: RaspberryPi3, Surin K., KMITL

10

4.1 โครงสร้างของชิปปุย ARM Cortex A53 ใน BCM2837



Computer Organization & ARM Assembly Language: RaspberryPi3, Surin K., KMITL

11

↗ ผังนี้มี integer pipeline + load and store pipeline

4.2 สถาปัตยกรรมชุดคำสั่งภาษาแอสเซมบลี

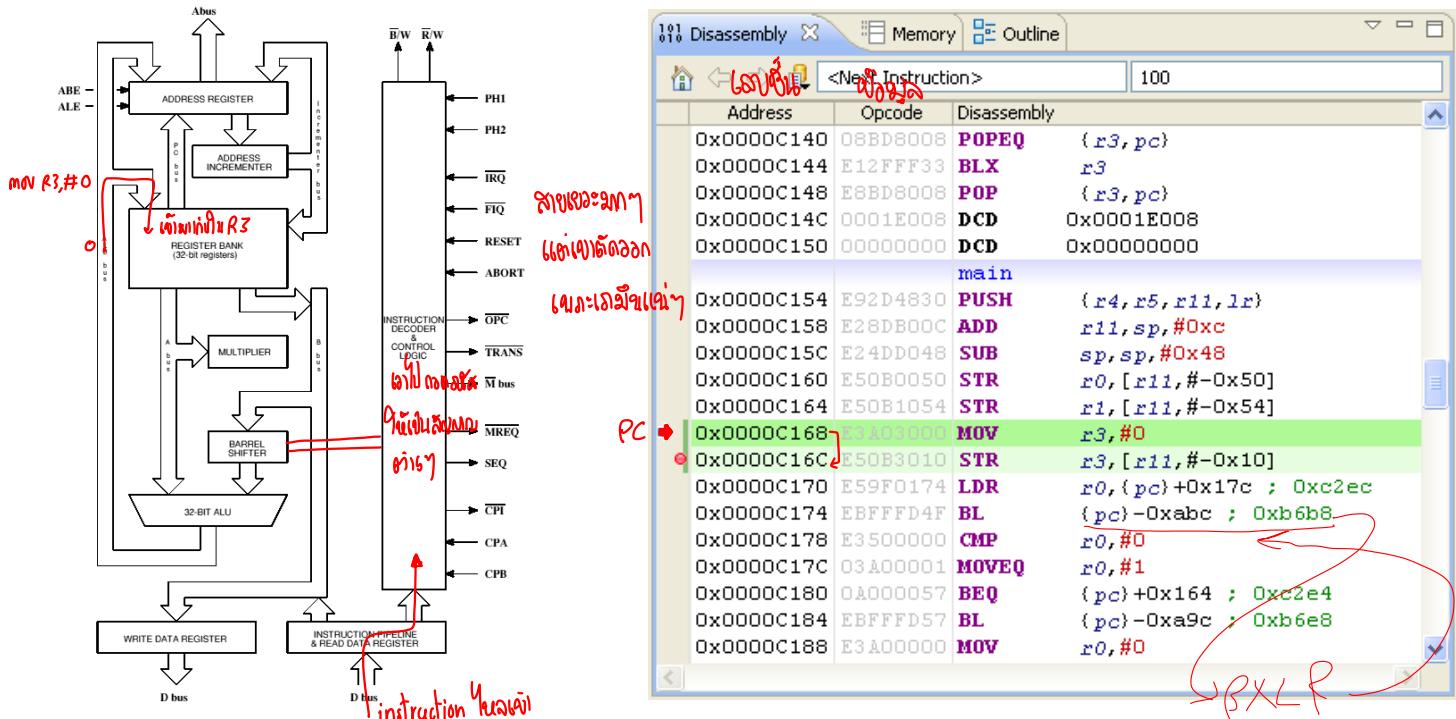
(Assembly Instruction Set Architecture)

- Cortex A53 รองรับการทำงานในโหมด 32 และ 64 บิต นั่นคือ คำสั่งแอสเซมบลีความยาว 32 และ 64 บิตตามโหมดการทำงาน
- วิชานี้ จะอ้างอิงคำสั่งแอสเซมบลีเวอร์ชัน 32 บิตของ ARM
เนื่องจากบอร์ดติดตั้งระบบปฏิบัติการ Raspbian ซึ่งทำงานในโหมด 32 บิต
-

Computer Organization & ARM Assembly Language: RaspberryPi3, Surin K., KMITL

12

4.3 ตัวอย่างคำสั่งภาษาเครื่องในหน่วยความจำ



Computer Organization & ARM Assembly Language: RaspberryPi3, Surin K., KMITL

13

Fetch → decode
↓
ปูร์กัน hacker

4.4 การประการคและตั้งค่าตัวแปรในหน่วยความจำหลัก

บรรทัดที่	เลbel	คำสั่ง	รีจิสเตอร์ หรือ แอดdress หรือ เลbel หรือ ค่าคงที่
1		.text	
2		.global main	
3	main:		
4		LDR R1, =M	ให้ address ของ M ใน R1
5		LDR R1, [R1]	รับ address ของ M ใน R1
6		LDR R2, =POINTR	; กับ address ของ pointer
7		MOV R0, #0	
8	LOOP:	LDR R3, [R2], #4	R3 = memory(R2), R2+4
9		ADD R0, R0, R3	เพิ่มตัวต้อง R0, R3
10		SUBS R1, R1, #1	ลดตัวต้อง R1 (7→6)
11		CMP R1, #0	ถ้า 0 = ให้ R1 status
12		BGT → ถ้า R1 > 0 ไปLOOP	
13		LDR R4, =SUM	รับ address ของตัว sum
14		STR R0, [R4]	value ตัวต้อง R0
15		BX LR	return
16		.data ห้าม	
17	SUM :	.word #0	
18	M:	.word #4	เริ่มต้น 4 byte
19	NUM:	.word 3, 5, 7, 9	ต้อง array
20	POINTR:	.word NUM	ให้ address ของตัว array num

เลbel คำสั่ง คอมเม้นท์

- 1 .data @Variable definition
- 2 .ballign 4 @Align variable to 4-byte space
- 3 wordvar1: .word 7 @wordvar1=7
- 4 .ballign 4 @Align variable to 4-byte space
- 5 wordvar2: .word 3 @wordvar2=3

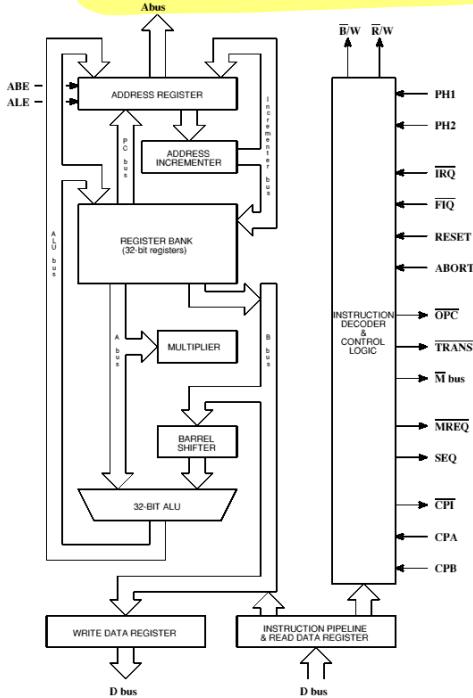
Computer Organization & ARM Assembly Language: RaspberryPi3, Surin K., KMITL

14

9 ตัว 1 word 4 byte = 32 bit

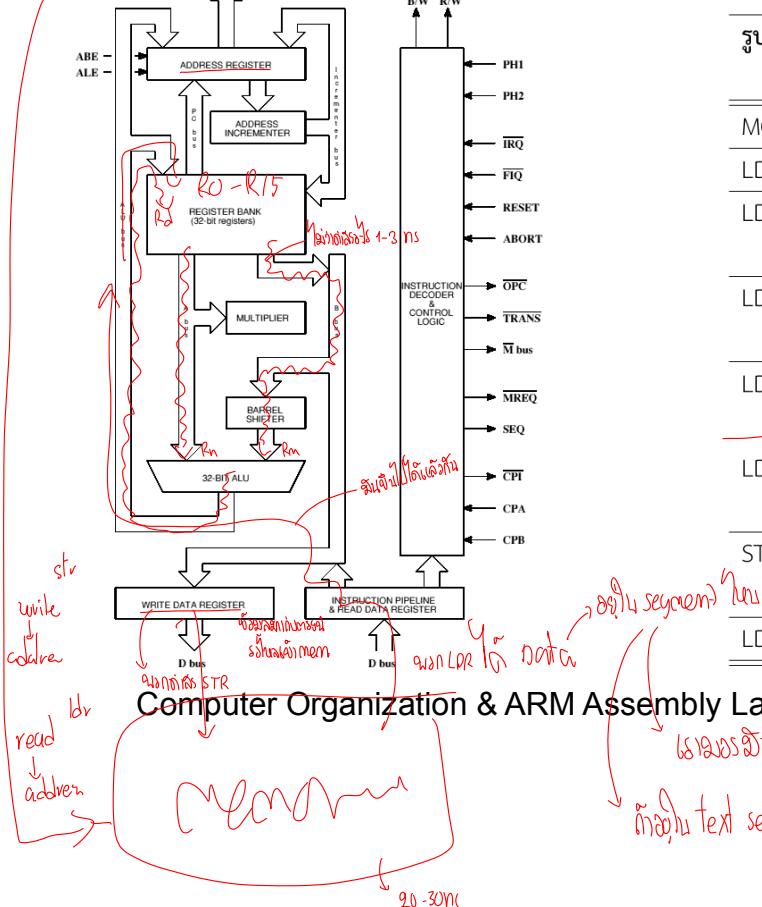
ดีที่	เลbel	คำสั่ง	รีจิสเตอร์ หรือ แอดเดรส หรือ เลbel หรือ ค่าคงที่
1		.text	
2		.global main	
3	main:		
4		LDR R1, =M	ใน address นี้เป็น M หรือ P
5		LDR R1, [R1]	ใน address นี้เป็น R1
6		LDR R2, =POINTR	; ให้ address นี้เป็น point
7		MOV R0, #0	ผู้ให้คำอธิบายต้องการ?
8	LOOP:	LDR R3, [R2], #4	R3 = memory(R2), R2+4
9		ADD R0, R0, R3	R0 = R0 + R3
10		SUBS R1, R1, #1	R1 = R1 - 1 (4-3)
11		CMP R1, #0	R1 = 0 = ไม่เท่ากัน
12		BGT LOOP	ถ้า R1 > 0 ไปLOOP
13		LDR R4, =SUM	pull ให้ address นี้เป็น sum
14		STR R0, [R4]	ให้ [R4] เก็บ R0
15		BX LR	LR = return
16		.data	ข้อมูล
17	SUM :	.word #0	เริ่มต้น 0
18	M:	.word #4	เริ่มต้น 4 byte
19	NUM:	.word 3, 5, 7, 9	ตัวแปร array
20	POINTR:	.word NUM	ใน address นี้เป็น array

4.5 คำสั่งถ่ายโอนข้อมูลระหว่างหน่วยความจำและรีจิสเตอร์



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

4.5 คำสั่งถ่ายโอนข้อมูลระหว่างหน่วยความจำและรีจิสเตอร์



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

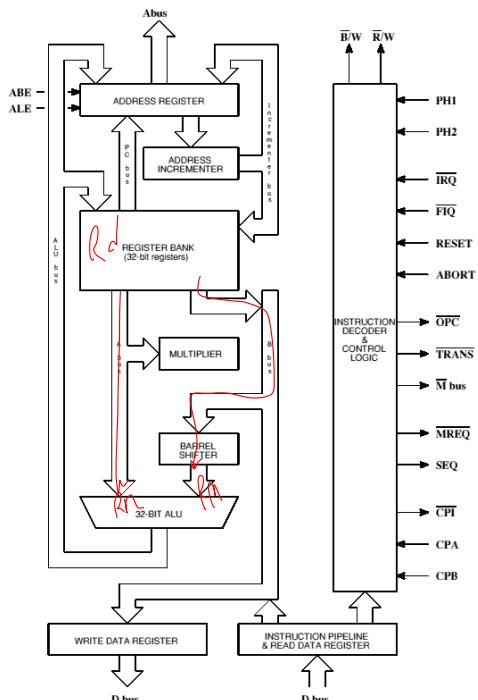
Computer Organization & ARM Assembly Language: RaspberryPi3, Surin K., KMITL

17

4.6 คำสั่งประมวลผลข้อมูลในรีจิสเตอร์ (Register Data Processing Instructions)

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

4.6 คำสั่งประมวลผลข้อมูลในรีจิสเตอร์ (Register Data Processing Instructions)



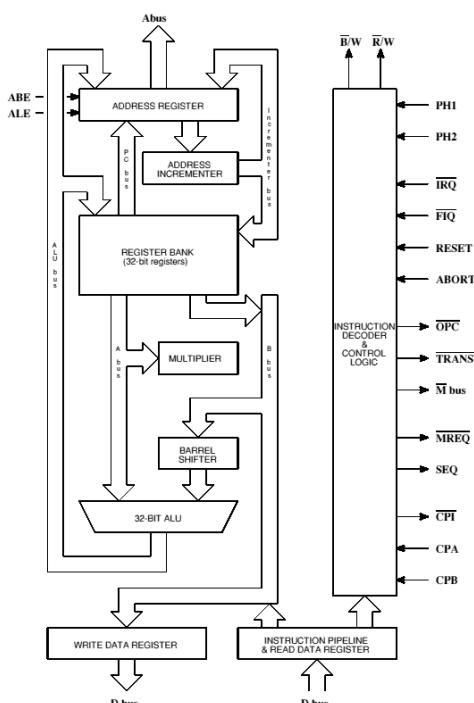
รูปแบบ	ความหมาย
ADD Rd, Rn, Rm	$Rd = Rn + Rm$
ADD Rd, Rn, #Imm	$Rd = Rn + \#Imm$
SUB Rd, Rn, Rm	$Rd = Rn - Rm$
SUB Rd, Rn, #Imm	$Rd = Rn - \#Imm$
RSB Rd, Rn, Rm	$Rd = Rm - Rn$ (Reverse Subtract)
RSB Rd, Rn, #Imm	$Rd = \#Imm - Rn$ (Reverse Subtract)
MUL Rd, Rn, Rm	$Rd = (Rn * Rm)$ (Only lower 32 bits)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

Computer Organization & ARM Assembly Language: RaspberryPi3, Surin K., KMITL

19

#immediate value part of instruction

4.6 คำสั่งประมวลผลข้อมูลในรีจิสเตอร์ (Register Data Processing Instructions)

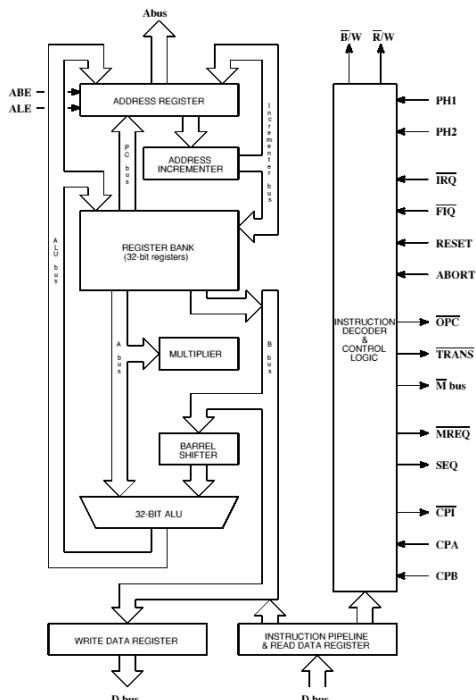


รูปแบบ	ความหมาย
LSL Rd, Rn, Rm	$Rd = Rn \ll Rm$ (Logical Shift Left)
LSL Rd, Rn, #Imm	$Rd = Rn \ll \#Imm$ (Logical Shift Left)
LSR Rd, Rn, Rm	$Rd = Rn \gg Rm$ (Logical Shift Right)
LSR Rd, Rn, #Imm	$Rd = Rn \gg \#Imm$ (Logical Shift Right)
ASR Rd, Rn, Rm	$Rd = Rn \gg Rm$ (Arithmetic Shift Right)
ASR Rd, Rn, #Imm	$Rd = Rn \gg \#Imm$ (Arithmetic Shift Right)

Computer Organization & ARM Assembly Language: RaspberryPi3, Surin K., KMITL

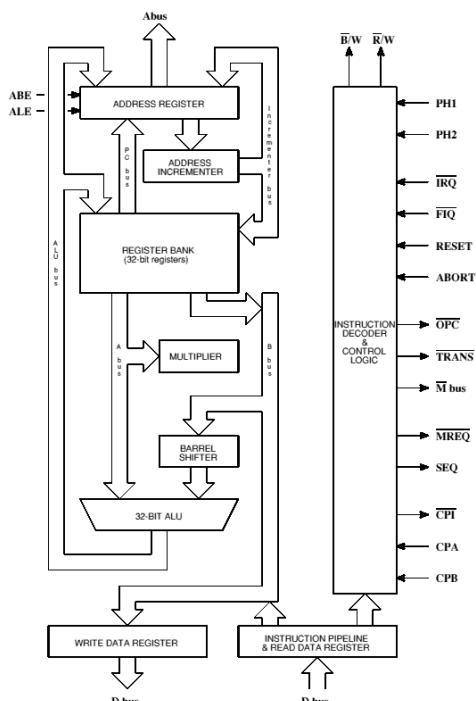
20

4.6 คำสั่งประมวลผลข้อมูลในรีจิสเตอร์ (Register Data Processing Instructions)



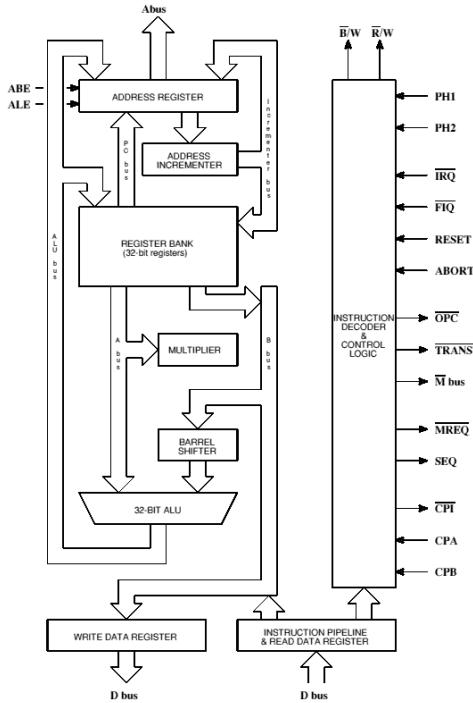
รูปแบบ	ความหมาย
ADD Rd, Rn, Rm LSL #shmt	$Rd = Rn + (Rm \ll \#shmt)$
ADD Rd, Rn, Rm LSR #shmt	$Rd = Rn + (Rm \ll \#shmt)$
ADD Rd, Rn, Rm ASR #shmt	$Rd = Rn + (Rm \gg \#shmt)$ (Signed)

4.6 คำสั่งประมวลผลข้อมูลในรีจิสเตอร์ (Register Data Processing Instructions)



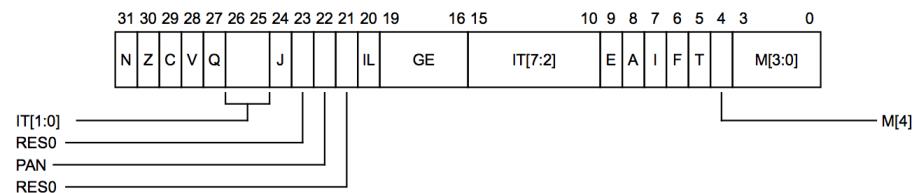
รูปแบบ	ความหมาย
AND Rd, Rn, Rm	$Rd = Rn \& Rm$ (bitwise AND)
AND Rd, Rn, #Imm	$Rd = Rn \& \#Imm$ (bitwise AND)
ORR Rd, Rn, Rm	$Rd = Rn \text{ OR } Rm$ (bitwise OR)
ORR Rd, Rn, #Imm	$Rd = Rn \text{ OR } \#Imm$ (bitwise OR)
MVN Rd, Rm	$Rd = \text{inverse}(Rm)$ (bitwise Inverse)
MVN Rd, #Imm	$Rd = \text{inverse}(\#Imm)$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

4.6 คำสั่งประมวลผลข้อมูลในรีจิสเตอร์ (Register Data Processing Instructions)



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

CPSR

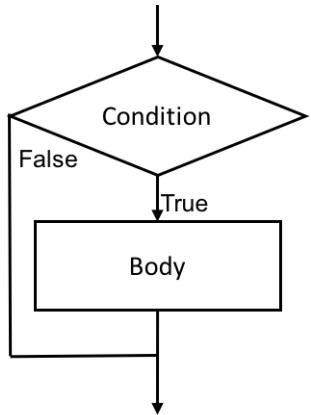


4.7 คำสั่งควบคุมการทำงาน (Control Instructions)

รูปแบบ	ความหมาย
B label	กระโดดไปยัง label (อย่างไม่มีเงื่อนไข)
BEQ label	กระโดดไปยัง label หากผลลัพธ์การเปรียบเทียบเท่ากัน $Z=1$
BNE label	กระโดดไปยัง label หากผลลัพธ์การเปรียบเทียบไม่เท่ากัน $Z=0$
BGT label	กระโดดไปยัง label หากผลลัพธ์การเปรียบเทียบมากกว่าจริง $N=0, Z=0$
BLT label	กระโดดไปยัง label หากผลลัพธ์การเปรียบเทียบน้อยกว่าจริง
BGE label	กระโดดไปยัง label หากผลลัพธ์การเปรียบเทียบมากกว่าหรือเท่ากับจริง
BLE label	กระโดดไปยัง label หากผลลัพธ์การเปรียบเทียบน้อยกว่าหรือเท่ากับจริง

4.7 คำสั่งควบคุมการทำงาน (Control Instructions)

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



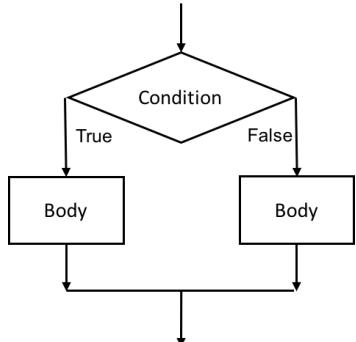
#	เลขบล	คำสั่ง	คอมเม้นท์
1		LDR R4, =a	@ get address of variable a
2		LDR R0, [R4]	@ get value of variable a
3		LDR R4, =b	@ get address of variable b
4		LDR R1, [R4]	@ get value of variable b
5		ADD R3, R0, R1	@ compute a+b
6		LDR R4, =c	@ get address of variable c
7		LDR R2, [R4]	@ get value of variable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

if (R120)

C

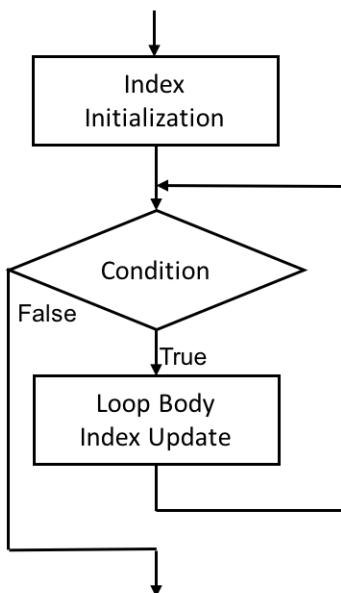
4.7 คำสั่งควบคุมการทำงาน (Control Instructions)

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



#	เลขบล	คำสั่ง	คอมเม้นท์
1		LDR R4, =a	ADD R5, R5, R6 @ x += y
2		LDR R0, [R4]	LDR R4, =x @ get address of va
3		LDR R4, =b	@ store value of va
4		LDR R1, [R4]	@ jump to exit label
5		ADD R3, R0, R1	B exit
6		LDR R4, =c	LDR R4, =x @ get address of va
7		LDR R2, [R4]	@ get value of variab
8		CMP R3, R2	@ get address of variab
9		BLE else	@ get value of variab
10		LDR R4, =x	@ x -= y @ get address of va
11		LDR R5, [R4]	@ store value of va
12		LDR R4, =y	@ get address of va
13		LDR R6, [R4]	@ get value of variab
14		ADD R5, R5, R6	@ label exit

4.7 คำสั่งควบคุมการทำงาน (Control Instructions)



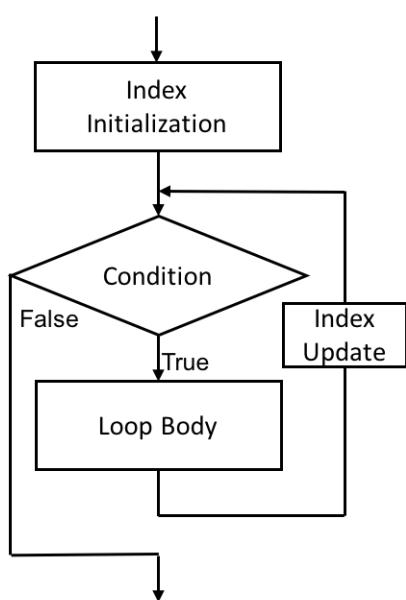
$$x = \sum_{i=1}^{10} i$$

```

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

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

4.7 คำสั่งควบคุมการทำงาน (Control Instructions)



$$i=1;$$

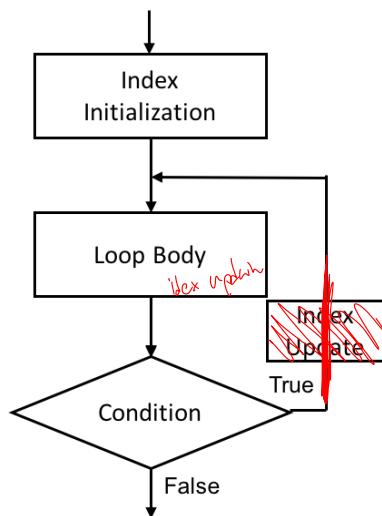
$$x=0;$$

```

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

#	เลขบล	คำสั่ง	คอมเมนท์
1		...	@ Initialize R0=0
2		...	@ Initialize R1=1
3	while:	CMP R1, #10	@ Compute R1-10
4		BGT end	@ if greater than goto end
5	ADD	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		...	@

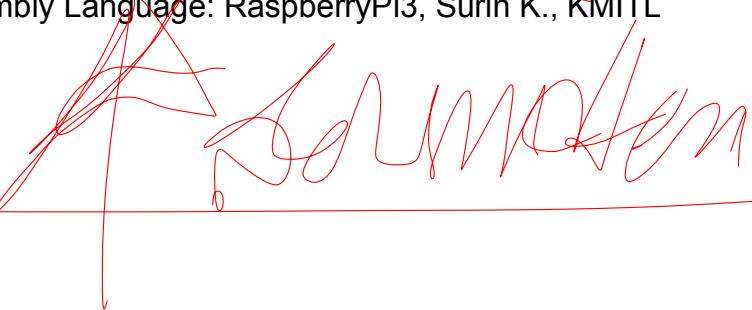
4.7 คำสั่งควบคุมการทำงาน (Control Instructions)



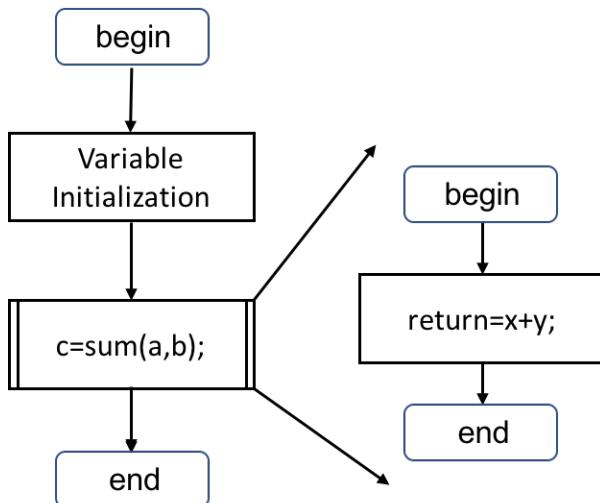
```

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

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



4.8 การเรียกใช้ฟังก์ชัน (Function Call)

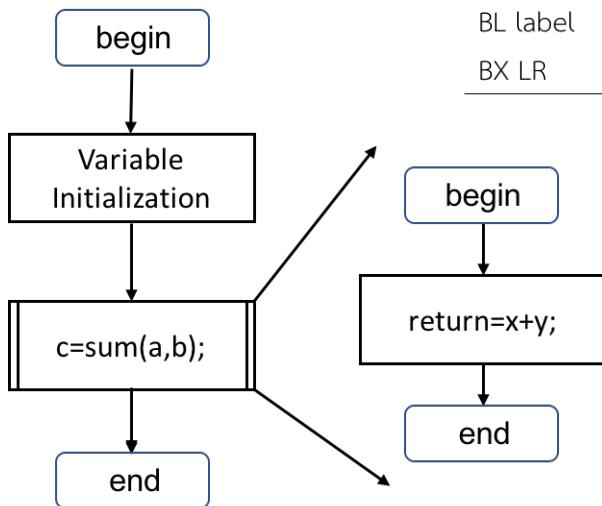


```

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

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

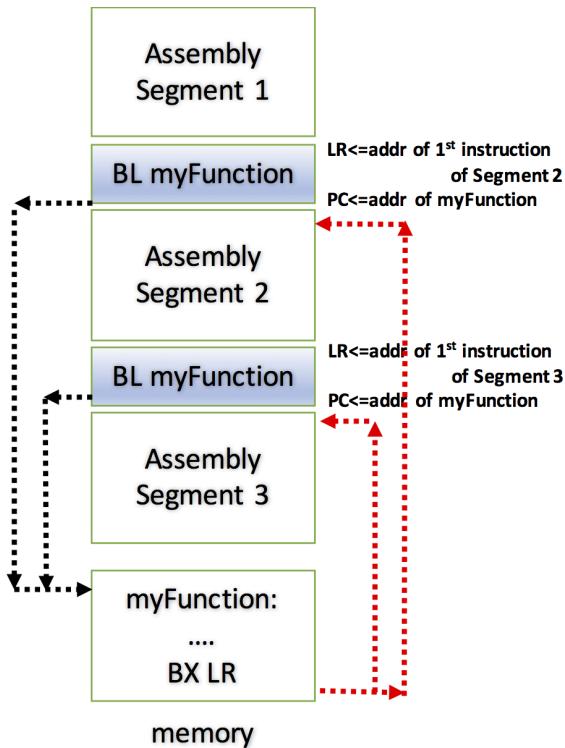
4.8 การเรียกใช้ฟังค์ชัน (Function Call)



รูปแบบ	ความหมาย
BL label	กระโดดไปยัง label ซึ่งเป็นชื่อฟังค์ชัน label โดยไม่มีเงื่อนไข
BX LR	กระโดดกลับไปทำงานคำสั่งที่อยู่ถัดจากคำสั่ง BL label ก่อนหน้า โดยไม่มีเงื่อนไข

#	เลขบล	คำสั่ง	คอมเม้นท์
1	main:	...	@ Initialize R4 (a)
2		...	@ Initialize R5 (b)
3		MOV R0, R4	@ Pass R0 to function sum
4		MOV R1, R5	@ Pass R1 to function sum
5		BL sum	@ Call function sum
6		...	@
7	sum:	ADD R0, R0, R1	@ entry point of function sum
8		BX LR	@ Return the result in R0

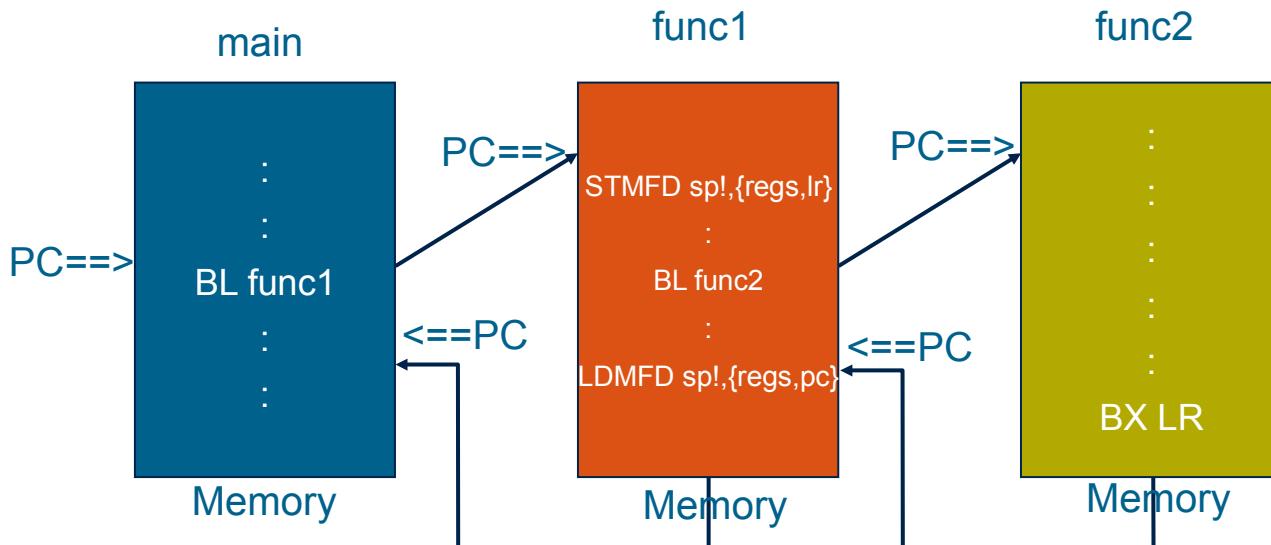
4.8 การเรียกใช้ฟังค์ชัน (Function Call)



● BL <func_name>

- Stores return address in LR
- Returning implemented by restoring the PC from LR
- For non-leaf functions, LR will have to be stacked

ក្រឡូលុយ បិតាគអូរ (នៅក្នុង)



Computer Organization & ARM Assembly Language: RaspberryPi3, Surin K., KMITL

33

ស្រួលបាយបញ្ហា

Computer Organization & ARM Assembly Language: RaspberryPi3, Surin K., KMITL

34