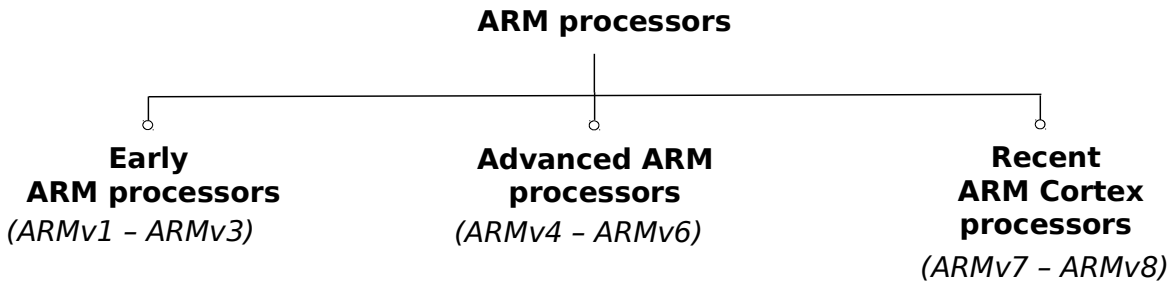
องค์ประกอบของเครื่องคอมพิวเตอร์
และภาษาแอสเซมบลี:
ARM และ RaspberryPi3

บทที่ 4 ภาษาแอสเซมบลีของ ARM เวอร์ชัน 32 บิท

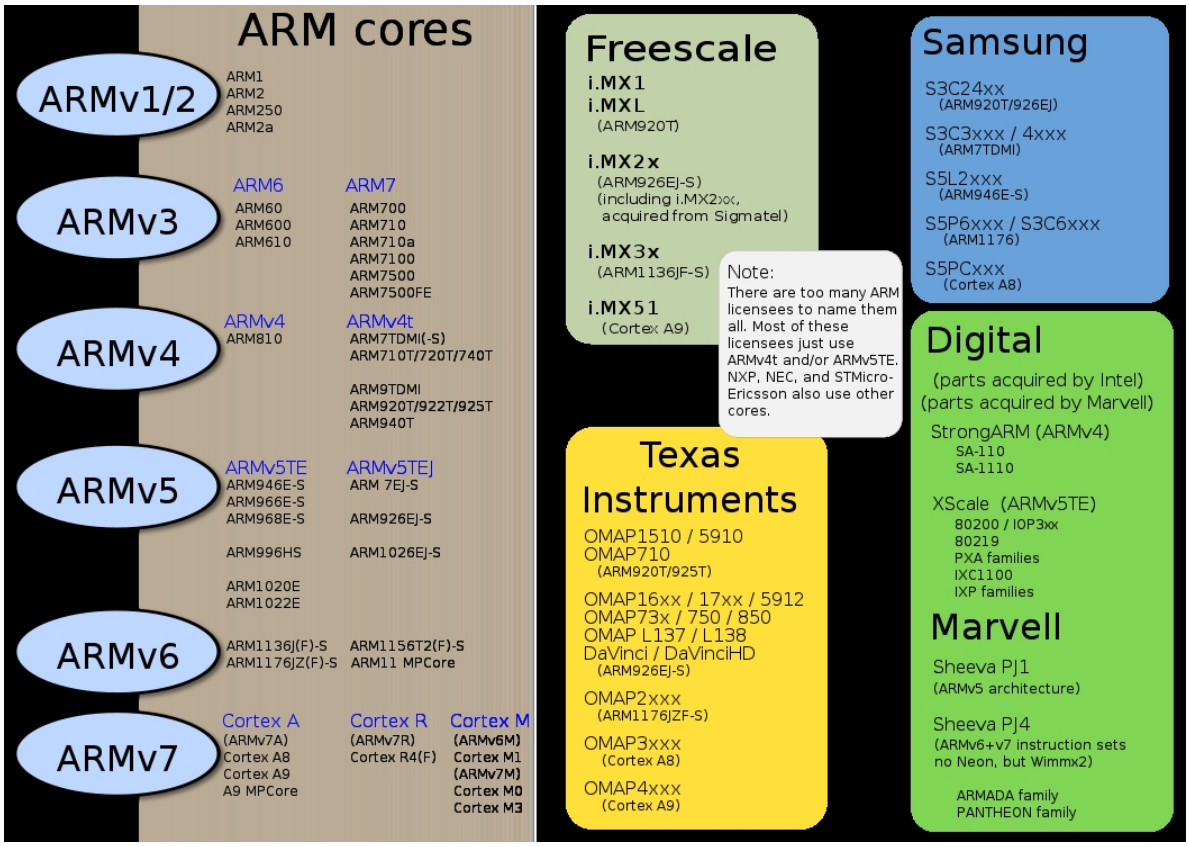
ผศ.ดร.สุรินทร์ กิตติธรกุล
ภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

สารบัญ

4.9 อุปกรณ์และวิวัฒนาการของชุดคำสั่ง ARM

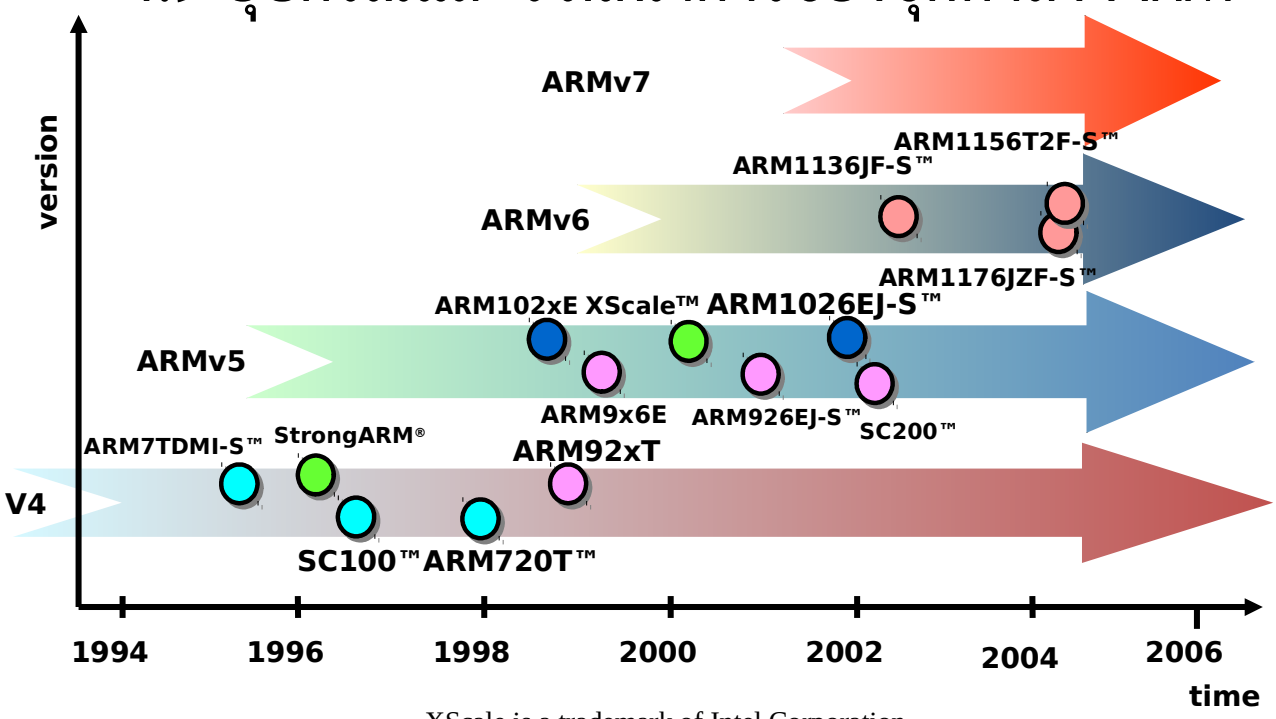


- ARM designed until now eight ISA versions, designated as ARMv1 – ARMv8, described in the related Architecture Reference Manuals.
- Subsequently, we give an overview of ARM's processor lines divided into three groups, according their underlying ISAs, as follows.



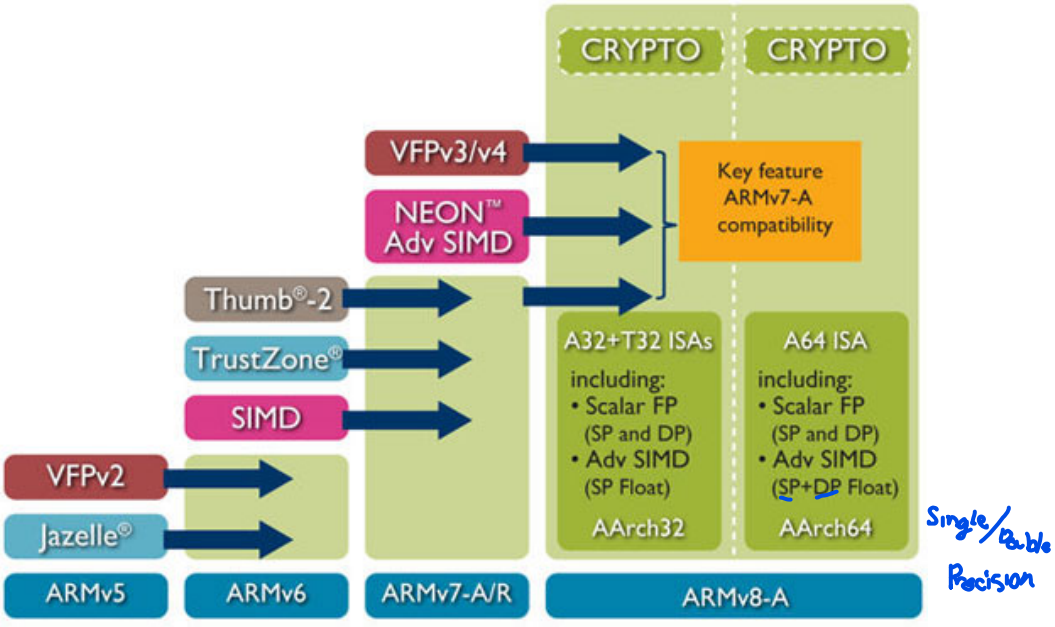
Early ARM processors (ARMv1 – ARMv3)

4.9 อุปกรณ์และวิวัฒนาการของชุดคำสั่ง ARM

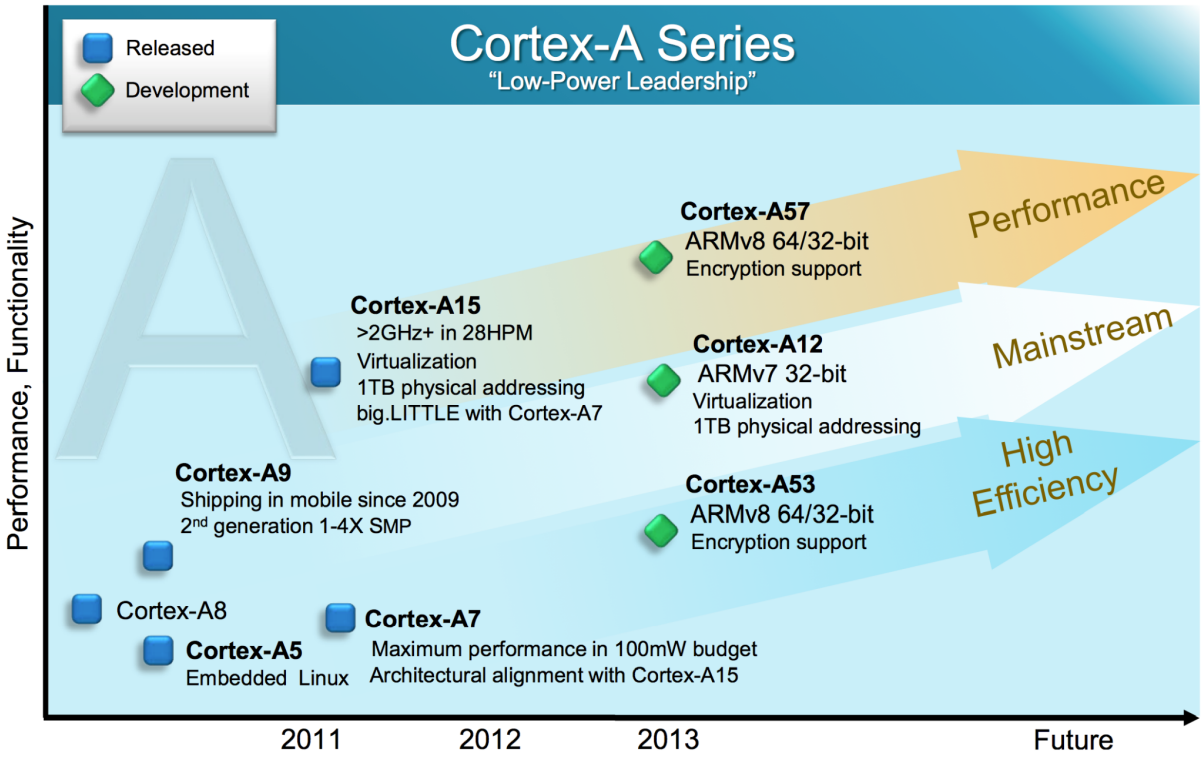


XScale is a trademark of Intel Corporation

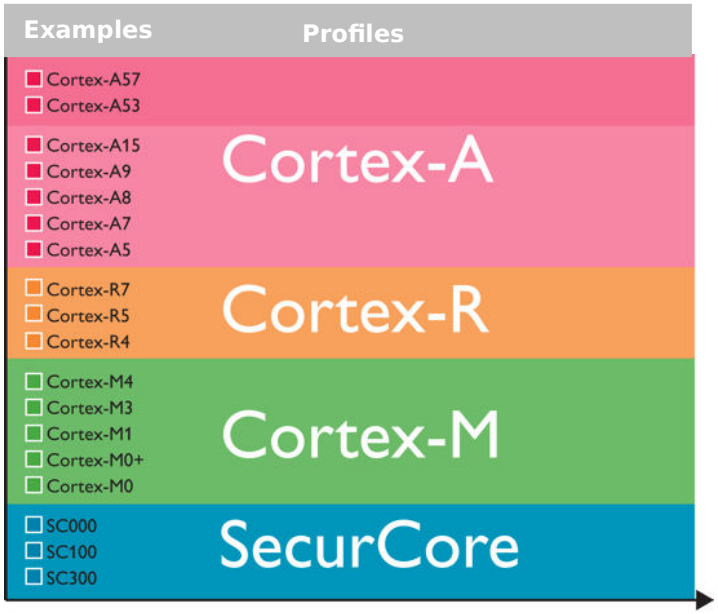
4.9 อุปกรณ์และวิวัฒนาการของชุดคำสั่ง ARM



Recent ARM Cortex-A processors



Recent ARM processors (ARMv7 and ARMv8)

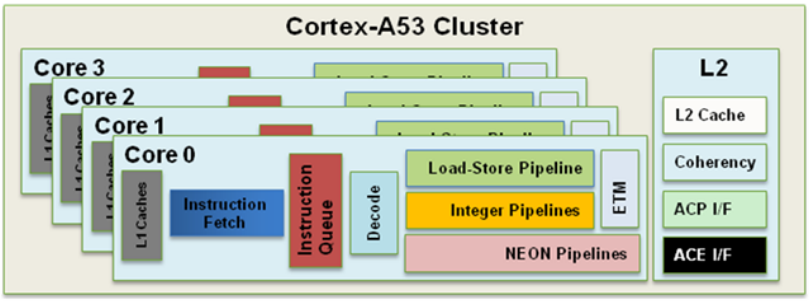


Cortex-A profile

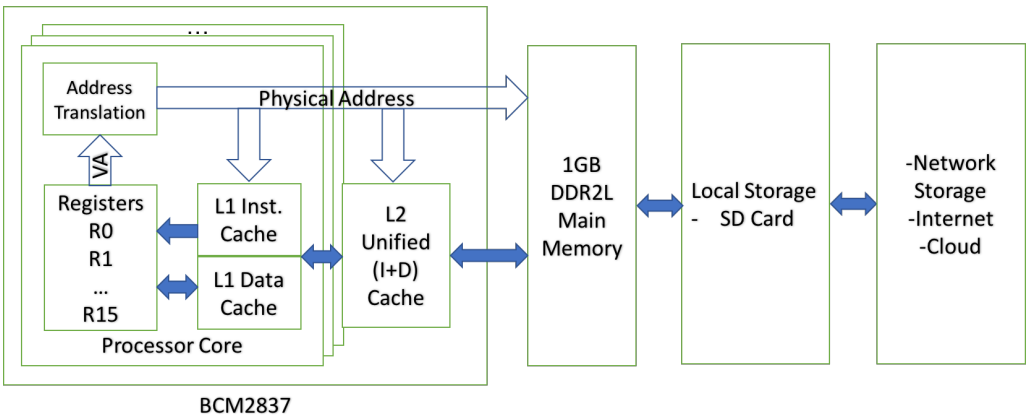
It aims at high-end applications running open and complex OSs, like smartphones, tablets, netbooks, eBook readers.

4.9 อุปกรณ์และวิวัฒนาการของชุดคำสั่ง ARM

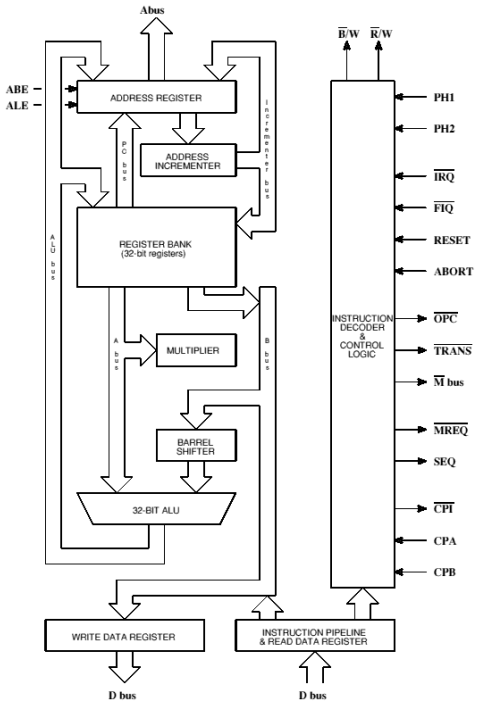
	Devices Shipped (Million of Units)	2010 Devices	Chips/ Device	TAM 2010 Chips	2010 ARM	2010 Share
Mobile	Smart Phone	280	2-5	1,200	1,100	90%
	Feature Phone	760	1-3	1,900	1,700	90%
	Low End Voice	570	1	570	540	95%
	Portable Media Players	150	1-3	300	220	70%
	Mobile Computing* (apps only)	230	1	230	25	10%
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%**

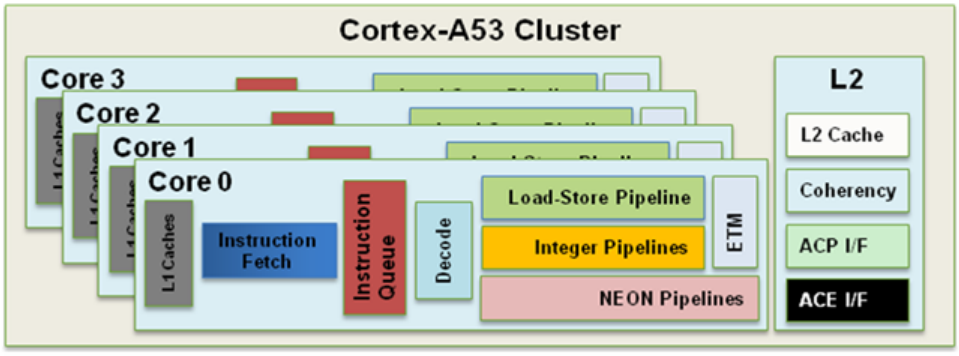


4.1 โครงสร้างของซีพียู ARM Cortex A53 ใน BCM2837



4.1 โครงสร้างของซีพียู ARM Cortex A53 ใน BCM2837

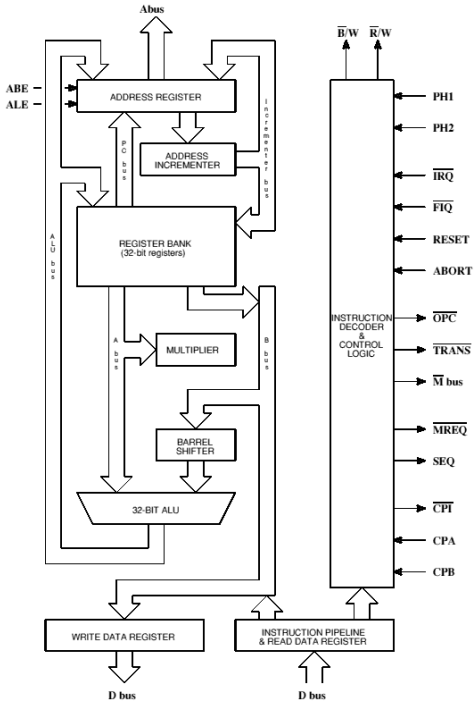


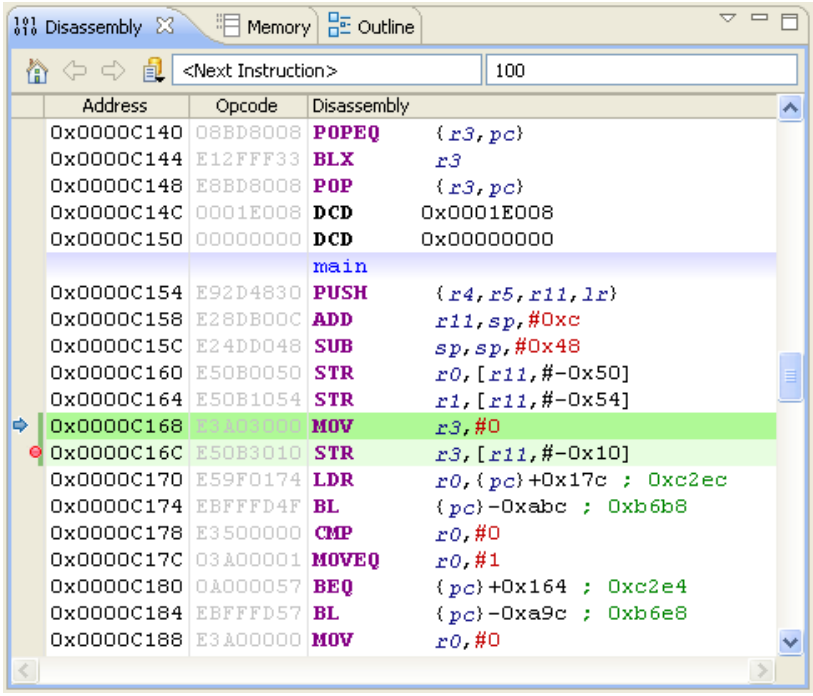


4.2 สถาปัตยกรรมชุดคำสั่งภาษาแอสเซมบลี (Assembly Instruction Set Architecture)

- Cortex A53 รองรับการทำงานในโหมด 32 และ 64 บิท นั่นคือ คำสั่งแอสเซมบลีความยาว 32 และ 64 บิทตามโหมดการทำงาน
- วิชานี้ จะอ้างอิงคำสั่งแอสเซมบลีเวอร์ชัน 32 บิทของ ARM เนื่องจากบอร์ดติดตั้งระบบปฏิบัติการ Raspbian ซึ่งทำงานในโหมด 32 บิท
-

4.3 ตัวอย่างคำสั่งภาษาเครื่องในหน่วยความจำ





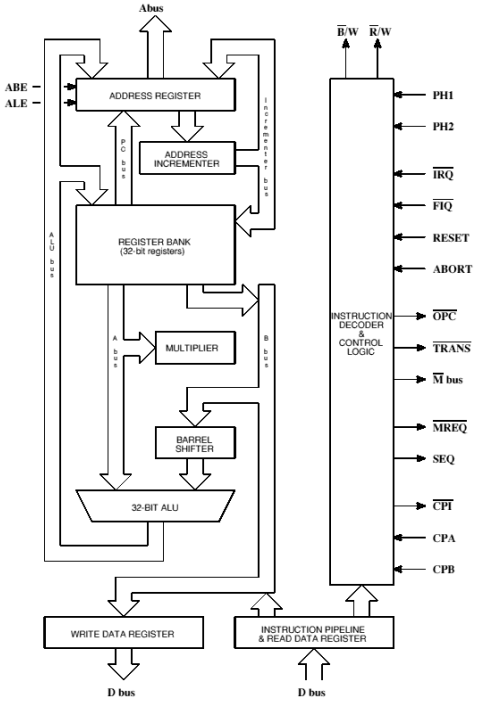
4.4 การประกาศและตั้งค่าตัวแปรในหน่วยความจำหลัก

#	เลเบล	คำสั่ง	คอมเมนท์
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

ใส่ค่าตัวแปร

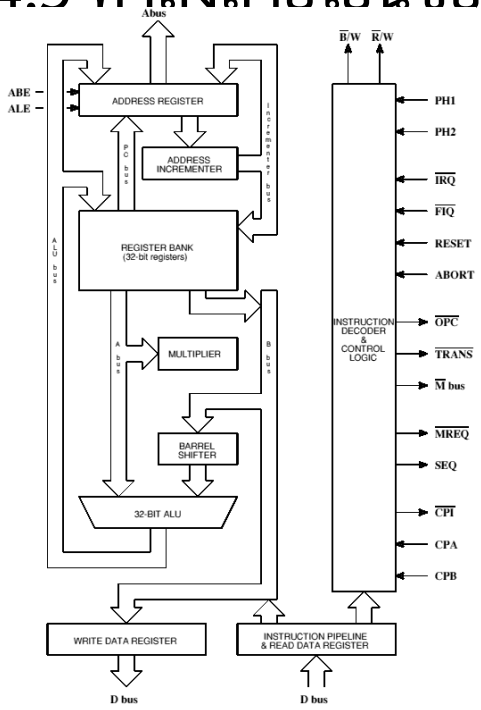
บรรทัดที่	เลเบล	คำสั่ง	รีจิสเตอร์ หรือ แอดเดรส หรือ เลเบล หรือ ค่าคงที่
1		.text	
2		.global main	
3	main:		
4		LDR	R1, =M
5		LDR	R1, [R1]
6		LDR	R2, =POINTR
7		MOV	R0, #0
8	LOOP:	LDR	R3, [R2], #4
9		ADD	R0, R0, R3
10		SUBS	R1, R1, #1
11		CMP	R1, #0
12		BGT	LOOP
13		LDR	R4, =SUM
14		STR	R0, [R4]
15	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

4.5 คำสั่งถ่ายโอนข้อมูลระหว่างหน่วยความจำและรีจิสเตอร์



รูปแบบ	ความหมาย (m และ n คือ หมายเลขรีจิสเตอร์มีค่าเท่ากับ 0-15)
LDR Rd [Rn]	Rd = Mem[Rn] (ก็อปปี้ข้อมูล 32 บิท)
STR Rd [Rn]	Mem[Rn] = Rd (ก็อปปี้ข้อมูล 32 bit)
LDRB Rd [Rn]	Rd = Mem[Rn] (ก็อปปี้ข้อมูล 8 บิทล่างสุด)
STRB Rd [Rn]	Mem[Rn] = Rd (ก็อปปี้ข้อมูล 8 บิทล่างสุด)
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

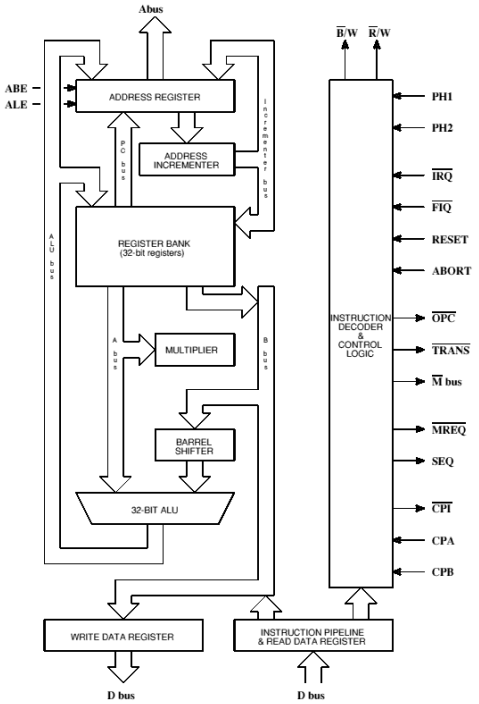
LDR Rd, [Rn, #Imm]!	Rd = Mem[Rn+ #Imm]
	Rn = Rn + #Imm

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)

4.6 คำสั่งประมวลผลข้อมูลในรีจิสเตอร์ (Register Data Processing Instructions)

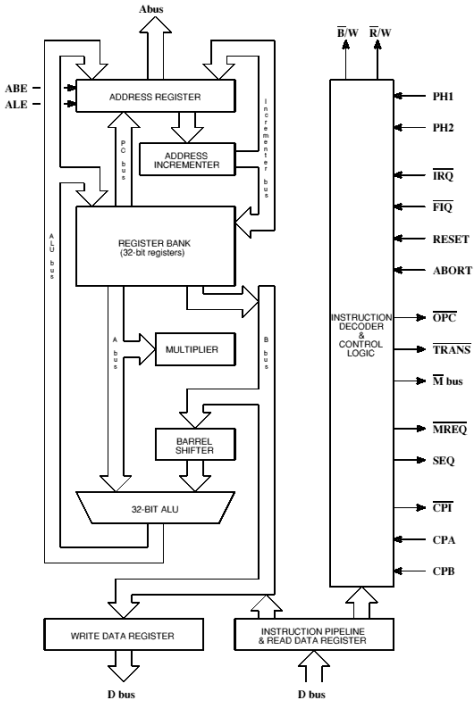
- **คำสั่งทางคณิตศาสตร์** เพื่อคำนวณเลขจำนวนเต็มชนิดมีและไม่มีเครื่องหมาย
- **คำสั่งเลื่อนบิท** เพื่อเลื่อนบิทข้อมูลไปทางซ้ายและขวา
- **คำสั่งทางคณิตศาสตร์และเลื่อนบิท** เพื่อคำนวณเลขจำนวนเต็มชนิดมีและไม่มีเครื่องหมาย หลังจากที่มีการเลื่อนบิทไปทางซ้ายหรือขวา
- **คำสั่งทางตรรกศาสตร์** เพื่อคำนวณค่าทางตรรกศาสตร์ของเลขจำนวนเต็มชนิดมีและไม่มีเครื่องหมาย

4.6 คำสั่งประมวลผลข้อมูลในรีจิสเตอร์ (Register Data Processing Instructions)



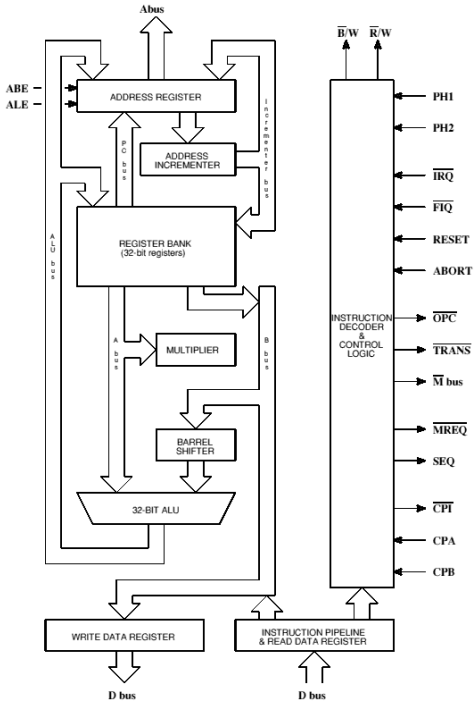
รูปแบบ	ความหมาย
ADD Rd, Rn, Rm	Rd = Rn + Rm
ADD Rd, Rn, #Imm	Rd = Rn + #Imm
SUB Rd, Rn, Rm	Rd = Rn - Rm
SUB Rd, Rn, #Imm	Rd = Rn - #Imm
RSB Rd, Rn, Rm	Rd = Rm - Rn (Reverse Subtract)
RSB Rd, Rn, #Imm	Rd = #Imm - Rn (Reverse Subtract)
MUL Rd, Rn, Rm	Rd = (Rn * Rm) (Only lower 32 bits)
UMULL Rhi, Rlo,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

4.6 คำสั่งประมวลผลข้อมูลในรีจิสเตอร์ (Register Data Processing Instructions)



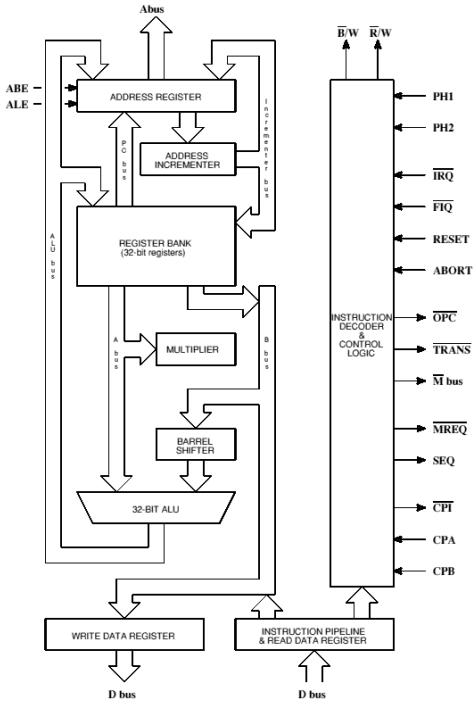
รูปแบบ	ความหมาย
LSL Rd, Rn, Rm	Rd = Rn « Rm (Logical Shift Left)
LSL Rd, Rn, #Imm	Rd = Rn « #Imm (Logical Shift Left)
LSR Rd, Rn, Rm	Rd = Rn » Rm (Logical Shift Right)
LSR Rd, Rn, #Imm	Rd = Rn » #Imm (Logical Shift Right)
ASR Rd, Rn, Rm	Rd = Rn » Rm (Arithmetic Shift Right)
ASR Rd, Rn, #Imm	Rd = Rn »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

4.6 คำสั่งประมวลผลข้อมูลในรีจิสเตอร์ (Register Data Processing Instructions)



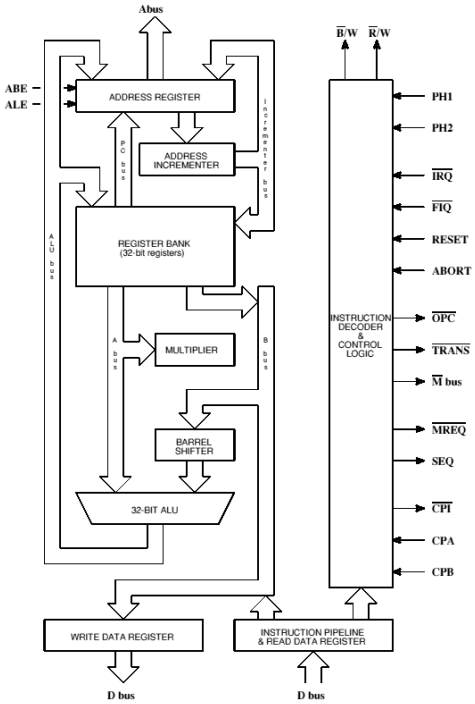
รูปแบบ	ความหมาย
ADD Rd, Rn, Rm LSL #shmt	Rd = Rn + (Rm « #shmt)
ADD Rd, Rn, Rm LSR #shmt	Rd = Rn + (Rm « #shmt)
ADD Rd, Rn, Rm ASR #shmt	Rd = Rn + (Rm » #shmt) (Signed)

4.6 คำสั่งประมวลผลข้อมูลในรีจิสเตอร์ (Register Data Processing Instructions)



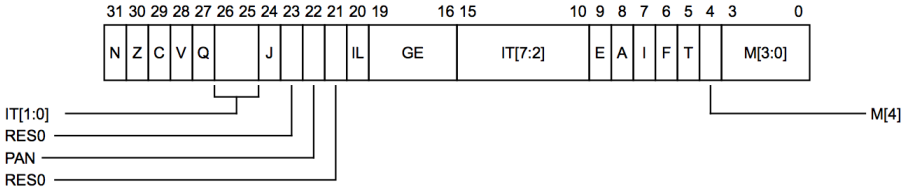
รูปแบบ	ความหมาย
AND Rd, Rn, Rm	Rd = Rn & Rm (bitwise AND)
AND Rd, Rn, #Imm	Rd = Rn & #Imm (bitwise AND)
ORR Rd, Rn, Rm	Rd = Rn Rm (bitwise OR)
ORR Rd, Rn, #Imm	Rd = Rn #Imm (bitwise OR)
MVN Rd, Rm	Rd = Rm (bitwise Inverse)
MVN Rd, #Imm	Rd = #Imm (bitwise Inverse)
EOR Rd, Rn, Rm	Rd = Rn ^ Rm (bitwise XOR)
EOR Rd, Rn, #Imm	Rd = Rn ^ #Imm (bitwise XOR)

4.6 คำสั่งประมวลผลข้อมูลในรีจิสเตอร์ (Register Data Processing Instructions)



Negative Zero Carry Overflow

รูปแบบ	ความหมาย
CMP Rn, Rm	NZCV ← Rn - Rm
CMP Rn, #Imm	NZCV ← Rn - #Imm



CPSR

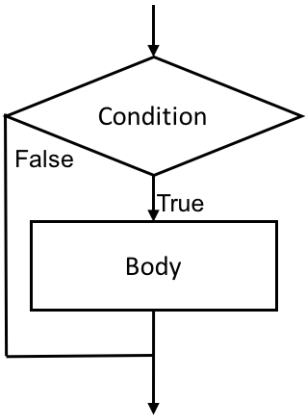
Current Prog Status Reg

4.7 คำสั่งควบคุมการทำงาน (Control Instructions)

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

4.7 คำสั่งควบคุมการทำงาน (Control Instructions)

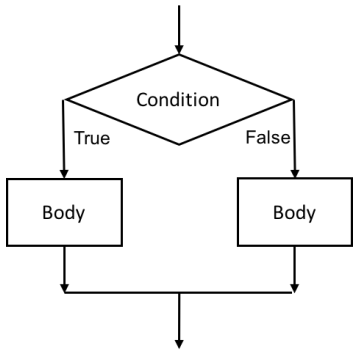
```
if ((a+b)>c) {
    x+=y;  /* Body */
}
```



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

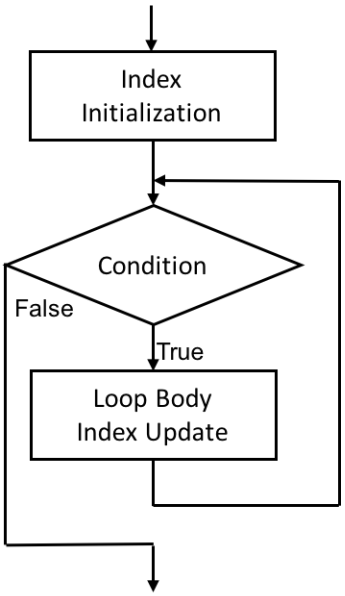
4.7 คำสั่งควบคุมการทำงาน (Control Instructions)

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



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

4.7 คำสั่งควบคุมการทำงาน (Control Instructions)

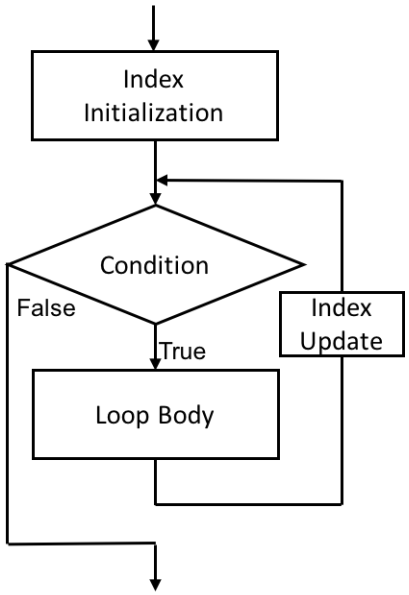


$$x = \sum_{i=1}^{10} i$$

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

#	เลเบล	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

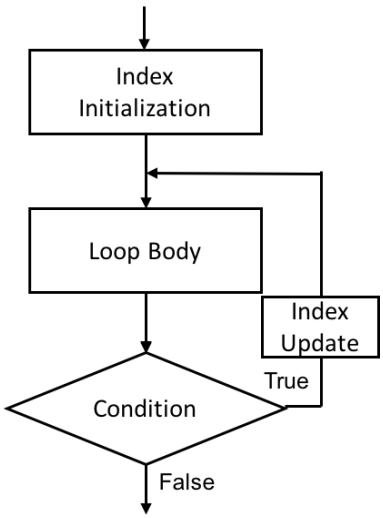
4.7 คำสั่งควบคุมการทำงาน (Control Instructions)




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

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

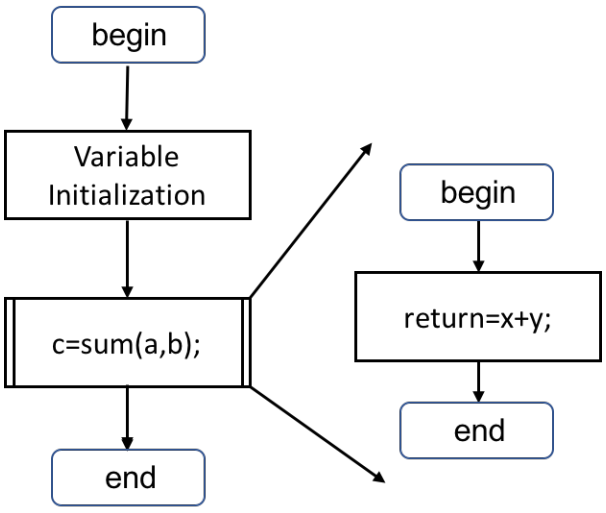
4.7 คำสั่งควบคุมการทำงาน (Control Instructions)




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

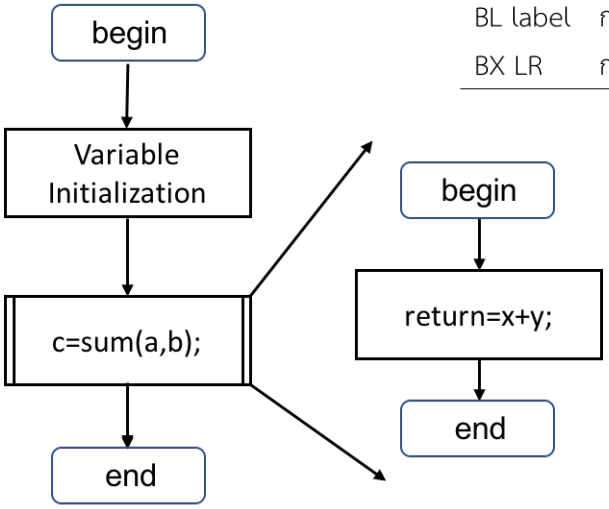
#	เลเบล	คำสั่ง	คอมเมนท์
1		...	@ Initialize R0 = 0
2		...	@ Initialize R1 = 0
3	do:	SUB R0, R0, R1	@ else subtract R1 from R2
4		ADD R1, R1, #1	@ R1 = R1+1
5		CMP R1, #10	@ Compute R1-10
6		BLE do	@ if less than or equal goto do
7	end:	...	@ End of do-while loop
8		...	@

4.8 การเรียกใช้ฟังค์ชัน (Function Call)




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

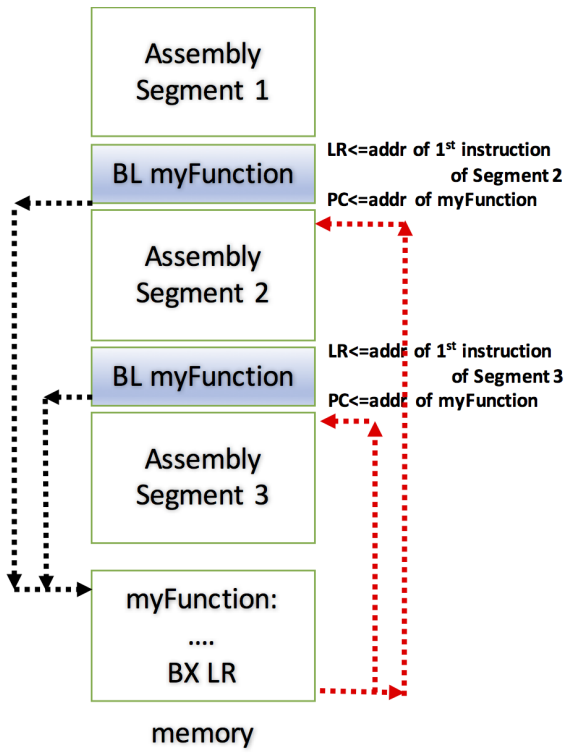
4.8 การเรียกใช้ฟังค์ชัน (Function Call)



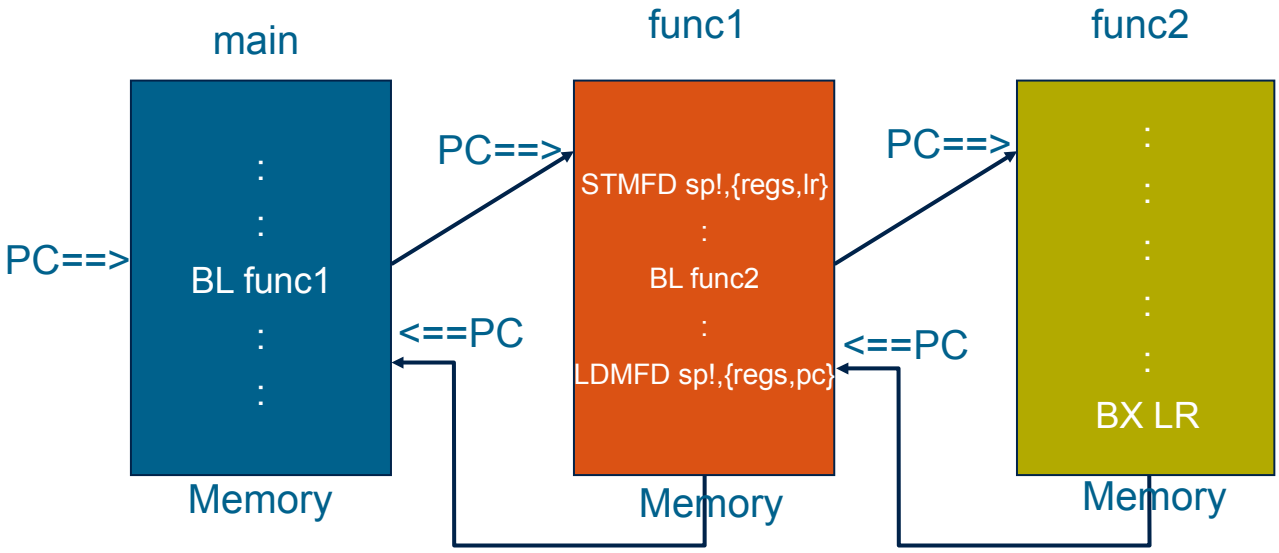
รูปแบบ	ความหมาย
BL label	กระโดดไปยัง label ซึ่งเป็นชื่อฟังค์ชัน label โดยไม่มีเงื่อนไข
BX LR	กระโดดกลับไปทำงานคำสั่งที่อยู่ถัดจากคำสั่ง BL label ก่อนหน้า โดยไม่มีเงื่อนไข

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

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



สรุปท้ายบท