

ARM Cortex-M4 Programming Model Flow Control Instructions

Textbook: Chapter 4, Section 4.9 (CMP,TEQ,TST)

Chapter 6

“ARM Cortex-M Users Manual”, Chapter 3

CPU instruction types

- **Data movement operations**
 - memory-to-register and register-to-memory
 - includes different memory “addressing” options
 - “memory” includes peripheral function registers
 - register-to-register
 - constant-to-register (or to memory in some CPUs)
- **Arithmetic operations**
 - add/subtract/multiply/divide
 - multi-precision operations (more than 32 bits)
- **Logical operations**
 - and/or/exclusive-or/complement (between operand bits)
 - shift/rotate
 - bit test/set/reset
- **Flow control operations**
 - branch to a location (conditionally or unconditionally)
 - branch to a subroutine/function
 - return from a subroutine/function

ARM comparison instructions

These instructions set flags in the PSR **without saving the result**.

“Set Status” is implied, and there is no “destination register”

- CMP : compare : $Op1 - Op2$
 - Sets Z, N, V and C flags
 - Use to test for signed and unsigned relationships
- TST : bit-wise AND : $Op1 \wedge Op2$
 - Sets Z and N flags; *C and V flags are unaffected**
 - Use to check selected bit(s) of a word
- TEQ : bit-wise XOR : $Op1 \text{ xor } Op2$
 - Sets Z and N flags; *C and V flags are unaffected**
 - Use instead of CMP to check for “equal” condition, if C and V flags are to be preserved (ex. for multi-precisions arithmetic)

* C flag affected if “shift” applied to Op2.

Avoid shifting Op2 if C flag must be preserved.

ARM flow control operations

- Unconditional branch: ***B label***
 - *Target address = PC ± displacement*
 - *Displacement embedded in instruction code*
 - *Target < ±32M(ARM), ±2K(Thumb), ±16M(Thumb2)*
- Conditional branch (**cc = true/false condition**):
Bcc label (Ex. *BNE label*)
Target < ±32M(ARM), -252..+258(T), ±1M(T2)
- Conditions that can be tested:
EQ, NE, CS, CC, MI, PL, VS, VC,
HI, LS, GE, LT, GT, LE
(see next slide)

Condition codes represented by PSR flags

<i>Suffix</i>	<i>Flags</i>	<i>Meaning</i>
EQ	$Z = 1$	Equal
NE	$Z = 0$	Not equal
CS or HS	$C = 1$	Higher or same, unsigned \geq
CC or LO	$C = 0$	Lower, unsigned $<$
MI	$N = 1$	Negative
PL	$N = 0$	Positive or zero
VS	$V = 1$	Overflow
VC	$V = 0$	No overflow
HI	$C = 1$ and $Z = 0$	Higher, unsigned $>$
LS	$C = 0$ or $Z = 1$	Lower or same, unsigned \leq
GE	$N = V$	Greater than or equal, signed \geq
LT	$N \neq V$	Less than, signed $<$
GT	$Z = 0$ <small>or</small> $N = V$	Greater than, signed $>$
LE	$Z = 1$ and $N \neq V$	Less than or equal, signed \leq
AL	Can have any value	Always. This is the default when no suffix specified

Text: Tables 6-2 to 6-5 (Arithmetic in Chap. 2)

Conditional Branch Instructions

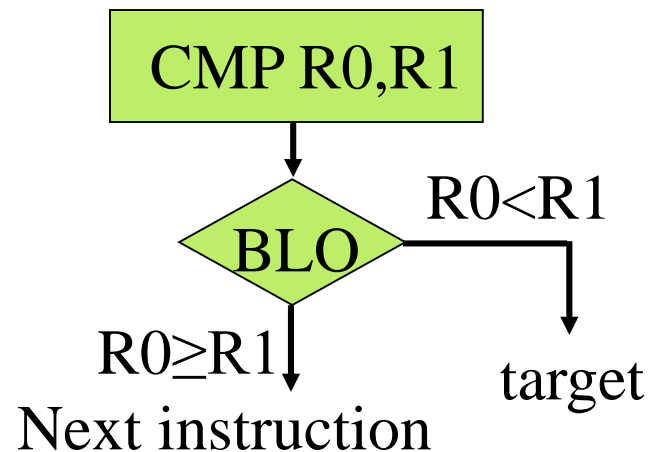
- Unsigned conditional branch
 - follow **SUBS** or **CMP**

BLO target ; Branch if unsigned less than (if C=0, same as **BCC**)

BLS target ; Branch if unsigned less than or equal to (if C=0 or Z=1)

BHS target ; Branch if unsigned greater than or equal to
(if C=1, same as **BCS**)

BHI target ; Branch if unsigned greater than (if C=1 and Z=0)



Conditional Branch Instructions

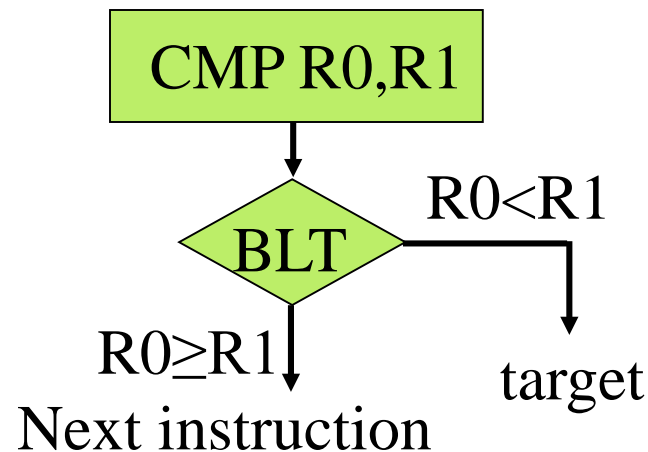
- Signed conditional branch
 - follow **SUBS**, **CMP**, or **CMN**

BLT target ; if signed less than (if $(\sim N \& V \mid N \& \sim V) = 1$ if $N \neq V$)

BGE target ; if signed greater than or equal to (if $(\sim N \& V \mid N \& \sim V) = 0$ if $N = V$)

BGT target ; if signed greater than (if $(Z \mid \sim N \& V \mid N \& \sim V) = 0$ if $Z = 0$ and $N = V$)

BLE target ; if signed less than or equal to
(if $(Z \mid \sim N \& V \mid N \& \sim V) = 1$ if $Z = 1$ and $N \neq V$)



Equality Test

Assembly code	C code
<pre>LDR R2, =G ; R2 = &G LDR R0, [R2] ; R0 = G CMP R0, #7 ; is G == 7 ? BNE next1 ; if not, skip BL GEqual7 ; G == 7 next1</pre>	<pre>unsigned long G; if(G == 7){ GEqual7(); }</pre>
<pre>LDR R2, =G ; R2 = &G LDR R0, [R2] ; R0 = G CMP R0, #7 ; is G != 7 ? BEQ next2 ; if not, skip BL GNotEqual7 ; G != 7 next2</pre>	<pre>if(G != 7){ GNotEqual7(); }</pre>

Program 5.1. Conditional structures that test for equality.

Can also use `TEQ R0, #7 ; Test if Equal (R0 xor #7 = 0)`
`BNE next1`

Unsigned Conditional Structures

Assembly code	C code
<pre> LDR R2, =G ; R2 = &G LDR R0, [R2] ; R0 = G CMP R0, #7 ; is G > 7? BLS next1 ; if not, skip BL GGreater7 ; G > 7 next1 </pre>	<pre> unsigned long G; if(G > 7){ GGreater7(); } </pre>
<pre> LDR R2, =G ; R2 = &G LDR R0, [R2] ; R0 = G CMP R0, #7 ; is G >= 7? BLO next2 ; if not, skip BL GGreaterEq7 ; G >= 7 next2 </pre>	<pre> if(G >= 7){ GGreaterEq7(); } </pre>
<pre> LDR R2, =G ; R2 = &G LDR R0, [R2] ; R0 = G CMP R0, #7 ; is G < 7? BHS next3 ; if not, skip BL GLess7 ; G < 7 next3 </pre>	<pre> if(G < 7){ GLess7(); } </pre>
<pre> LDR R2, =G ; R2 = &G LDR R0, [R2] ; R0 = G CMP R0, #7 ; is G <= 7? BHI next4 ; if not, skip BL GLessEq7 ; G <= 7 next4 </pre>	<pre> if(G <= 7){ GLessEq7(); } </pre>

Program 5.2. Unsigned conditional structures.

Bard, Gerstlauer,
Valvano, Yerraballi

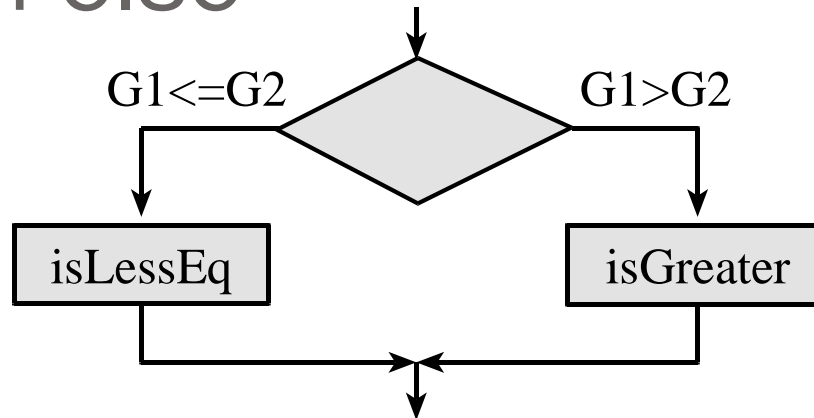
Signed Conditional Structures

Assembly code	C code
<pre>LDR R2, =G ; R2 = &G LDR R0, [R2] ; R0 = G CMP R0, #7 ; is G > 7? BLE next1 ; if not, skip BL GGreater7 ; G > 7 next1</pre>	<pre>long G; if(G > 7){ GGreater7(); }</pre>
<pre>LDR R2, =G ; R2 = &G LDR R0, [R2] ; R0 = G CMP R0, #7 ; is G >= 7? BLT next2 ; if not, skip BL GGreaterEq7 ; G >= 7 next2</pre>	<pre>if(G >= 7){ GGreaterEq7(); }</pre>
<pre>LDR R2, =G ; R2 = &G LDR R0, [R2] ; R0 = G CMP R0, #7 ; is G < 7? BGE next3 ; if not, skip BL GLess7 ; G < 7 next3</pre>	<pre>if(G < 7){ GLess7(); }</pre>
<pre>LDR R2, =G ; R2 = &G LDR R0, [R2] ; R0 = G CMP R0, #7 ; is G <= 7? BGT next4 ; if not, skip BL GLessEq7 ; G <= 7 next4</pre>	<pre>if(G <= 7){ GLessEq7(); }</pre>

Program 5.4. Signed conditional structures.

Bard, Gerstlauer,
Valvano, Yerraballi

If-then-else



```
LDR R2, =G1 ; R2 = &G1
LDR R0, [R2] ; R0 = G1
LDR R2, =G2 ; R2 = &G2
LDR R1, [R2] ; R1 = G2
CMP R0, R1 ; is G1 > G2 ?
BHI high ; if so, skip to high
low BL isLessEq ; G1 <= G2
B next ; unconditional
high BL isGreater ; G1 > G2
next
```

```
unsigned long G1,G2;
if(G1>G2){
    isGreater();
}
else{
    isLessEq();
}
```

Example: if-then-else statement

- C:

```
if (a > b) { x = 5; y = c + d; } else x = c - d;
```

- Assembler:

```
; compute and test condition
```

```
LDR r4,=a           ; get address for a
```

```
LDR r0,[r4]        ; get value of a
```

```
LDR r4,=b           ; get address for b
```

```
LDR r1,[r4]        ; get value for b
```

```
CMP r0,r1           ; compare a < b
```

```
BLE fblock        ; if a >= b, branch to false block
```

If statement, cont'd.

; true block

```
MOV r0,#5           ; generate value for x
LDR r4,=x           ; get address for x
STR r0,[r4]         ; store x
LDR r4,=c           ; get address for c
LDR r0,[r4]         ; get value of c
LDR r4,=d           ; get address for d
LDR r1,[r4]         ; get value of d
ADD r0,r0,r1        ; compute y
LDR r4,=y           ; get address for y
STR r0,[r4]         ; store y
B after           ; branch around false block
```

If statement, cont'd.

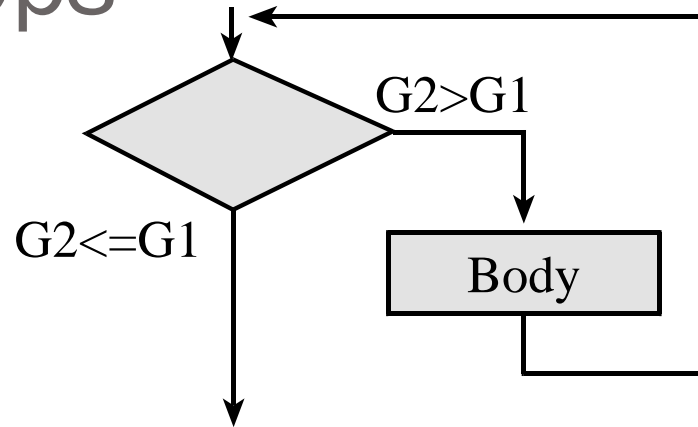
`; false block`

`fblock`

```
LDR r4,=c           ; get address for c
LDR r0,[r4]         ; get value of c
LDR r4,=d           ; get address for d
LDR r1,[r4]         ; get value for d
SUB r0,r0,r1        ; compute a-b
LDR r4,=x           ; get address for x
STR r0,[r4]         ; store value of x
```

`after ...`

While Loops



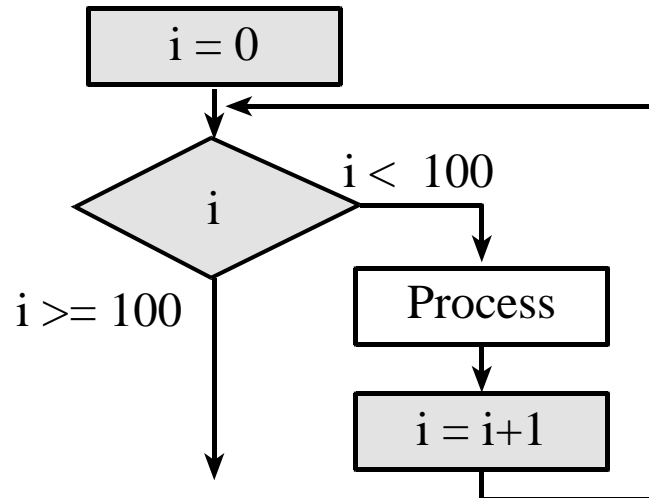
```
LDR R4, =G1    ; R4 -> G1
LDR R5, =G2    ; R5 -> G2
loop LDR R0, [R5] ; R0 = G2
LDR R1, [R4]   ; R1 = G1
CMP R0, R1    ; is G2 <= G1?
BLS next     ; if so, skip to next
BL Body      ; body of the loop
B loop
next
```

```
unsigned long G1,G2;
while(G2 > G1){
    Body();
}
```

For Loops

Count up

```
for(i=0; i<100; i++){  
    Process();  
}
```



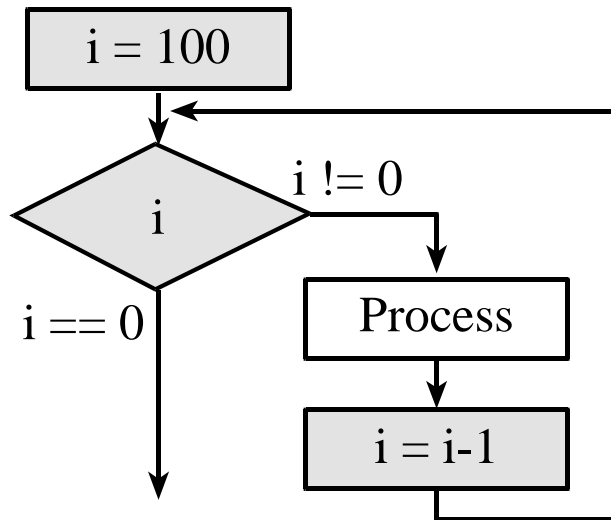
```
MOV R4, #0      ; R4 = 0  
loop CMP R4, #100 ; index >= 100?  
BHS done      ; if so, skip to done  
BL Process     ; process function*  
ADD R4, R4, #1 ; R4 = R4 + 1  
B loop
```

done

For Loops

Count down

```
for(i=100; i!=0; i--){  
    Process();  
}
```



```
MOV R4, #100 ; R4 = 0  
loop BL Process ; process function  
SUBS R4, R4, #1 ; R4 = R4 - 1  
BNE loop  
done
```

Thumb2 conditional execution

- (IF-THEN) instruction, IT, supports conditional execution in Thumb2 of up to 4 instructions in a “block”
 - Designate instructions to be executed for THEN and ELSE
 - Format: ITxyz condition, where x,y,z are T/E/blank

<i>if (r0 > r1) {</i>	<i>cmp r0,r1</i>	<i>;set flags</i>
<i> add r2,r3,r4</i>	<i>ITTEE GT</i>	<i>;condition 4 instr</i>
<i> sub r3,r4,r5</i>	<i>addgt r2,r3,r4</i>	<i>;do if r0>r1</i>
<i>} else {</i>	<i>subgt r3,r4,r5</i>	<i>;do if r0>r1</i>
<i> and r2,r3,r4</i>	<i>andle r2,r3,r4</i>	<i>;do if r0<=r1</i>
<i> orr r3,r4,r5</i>	<i>orrle r3,r4,r5</i>	<i>;do if r0<=r1</i>
<i>}</i>	<i>Thumb2 code</i>	

Pseudo-C

Example: C switch statement

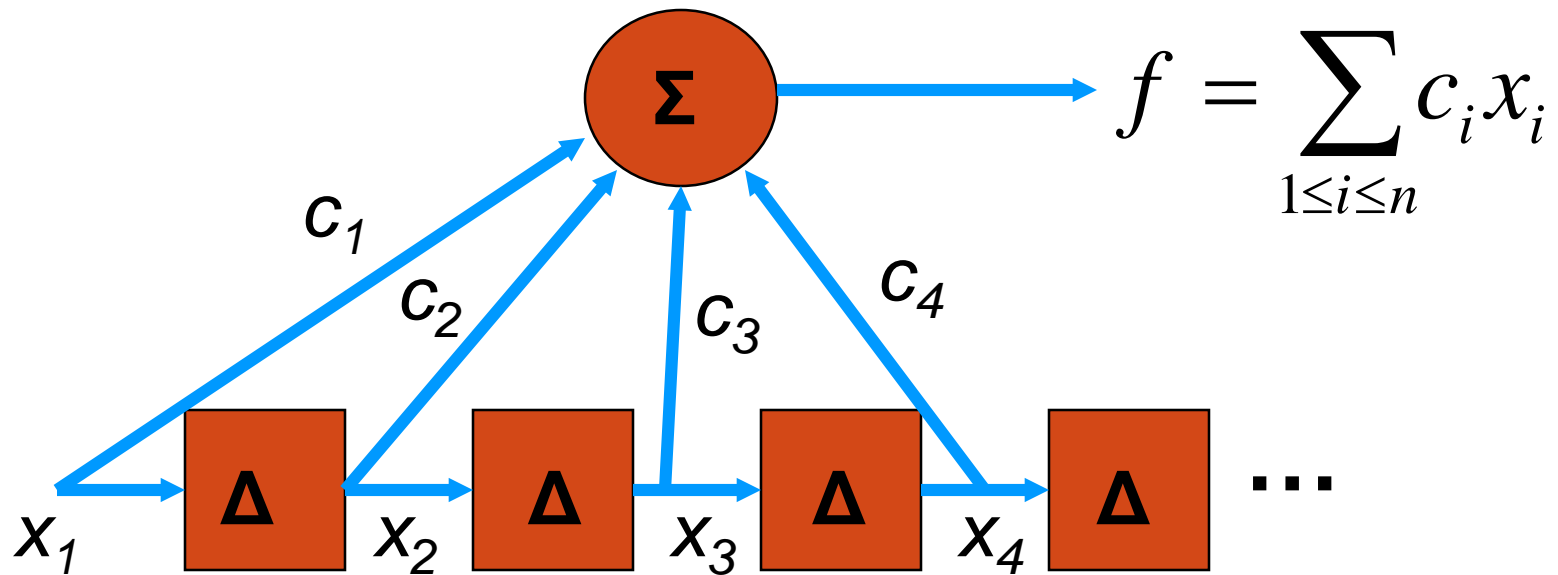
- C:

```
switch (test) { case 0: ... break; case 1: ... }
```

- Assembler:

```
LDR r2,=test           ; get address for test
LDR r0,[r2]            ; load value for test
ADR r1,switchtab      ; load switch table address
LDR r15,[r1,r0,LSL #2] ; index switch table
switchtab DCD case0    ;address of case0 routine
          DCD case1    ;address of case1 routine
          ...
```

Finite impulse response (FIR) filter



x_i 's are data samples
 c_i 's are constants

Example: FIR filter

- C:

```
for (i=0, f=0; i<N; i++)  
    f = f + c[i]*x[i];
```

- Assembler

i loop initiation code

```
MOV r0,#0           ; use r0 for I  
MOV r8,#0           ; use separate index for arrays  
LDR r2,=N           ; get address for N  
LDR r1,[r2]         ; get value of N  
MOV r2,#0           ; use r2 for f  
LDR r3,=c           ; load r3 with base of c  
LDR r5,=x           ; load r5 with base of x
```

FIR filter, cont'.d

; loop body

loop

```
LDR r4,[r3,r8]    ; get c[i]
LDR r6,[r5,r8]    ; get x[i]
MUL r4,r4,r6      ; compute c[i]*x[i]
ADD r2,r2,r4      ; add into running sum f
ADD r8,r8,#4      ; add word offset to array index
ADD r0,r0,#1      ; add 1 to i
CMP r0,r1         ; exit?
BLT loop          ; if i < N, continue
```

FIR filter with MLA & auto-index

```
AREA TestProg, CODE, READONLY
ENTRY
    mov     r0,#0           ;accumulator
    mov     r1,#3           ;number of iterations
    ldr     r2,=carray      ;pointer to constants
    ldr     r3,=xarray      ;pointer to variables
loop    ldr     r4,[r2],#4   ;get c[i] and move pointer
        ldr     r5,[r3],#4   ;get x[i] and move pointer
        mla     r0,r4,r5,r0  ;sum = sum + c[i]*x[i]
        subs   r1,r1,#1     ;decrement iteration count
        bne    loop        ;repeat until count=0
here    b      here
carray dcd    1,2,3
xarray dcd    10,20,30
END
```

Also, need “time delay” to prepare x array for next sample