Control (5A)

Young Won Lim
2/15/20

Please send corrections (or suggestions) to youngwlim@hotmail.com.

This document was produced by using LibreOffice.

Young Won Lim
2/15/20

Based on

ARM System-on-Chip Architecture, 2nd ed, Steve Furber

Introduction to ARM Cortex-M Microcontrollers
– Embedded Systems, Jonathan W. Valvano

Digital Design and Computer Architecture,
D. M. Harris and S. L. Harris

Branch and Branch with Link (B, BL)

Branch and Branch with Link (B, BL)

B{L} {<cond>} <target address>

L : the branch and link

<cond> : condition codes, AL if omitted

<target address> : a label in the assembler code

The assembler will generate the offset

target address – branch instruction address + 8

```
B          <target address>
B<cond>    <target address>
BL         <target address>
BL<cond>   <target address>
```

Condition Code Suffixes <cond>

Suffix	Flags	Meaning
EQ	Z = 1	Equal
NE	Z = 0	Not equal
CS or **HS**	C = 1	Higher or same, unsigned
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
GE	N = V	Greater than or equal, signed
LT	N != V	Less than, signed
GT	Z = 0 and N = V	Greater than, signed
LE	Z = 1 and N != V	Less than or equal, signed
AL	any value	Always. This is the default when no suffix is specified.

https://community.arm.com/processors/b/blog/posts/condition-codes-1-condition-flags-and-codes

Conditional Flags

N=1	if the result is negative
Z=1	if the result is zero
C=1	the carry out of the ALU when the operation is arithmetic (ADD, ADC, SUB, SBC, RSB, RSC, CMP, CMN), or the carry out of the shifter (C is preserved when no shift)
V=1	if overflow is occurred during arithmetic operations only when an arithmetic operation has operands that are viewed as 2's complement signed value (V is preserved when non-arithmetic operations)

Conditional Flag Setting Instructions

Data Processing Instructions

<op> {<cond>} {**S**} Rd, Rn, #<32-bit immediate>

<op> {<cond>} {**S**} Rd, Rn, Rm, {<shift>}

Multiply Instructions

MUL {<cond>} {**S**} Rd, Rm, Rs

MLA {<cond>} {**S**} Rd, Rm, Rs, Rn

<mul> {<cond>} {**S**} RdHi, RdLo, Rm, Rs

UMULL, UMLAL, SMULL, SMLAL

<mul>	Meaning
MUL	Multiply (32-bit)
MLA	Multiply-Accumulate (32-bit)
UMULL	Unsigned Multiply Long
UMLAL	Unsigned Multiply Acc Long
SMULL	Singed Multiply Long
SMLAL	Unsigned Multiply Acc Long

<op>	Meaning
AND	Logical bit-wise AND
EOR	Logical bit-wise exclusive OR
SUB	Subtract
RSUB	Reverse subtract
ADD	Add
ADC	Add with carry
SBC	Subtract with carry
RSC	Reverse subtract with carry
TST	Test

Branch Conditions

B<cond>		
B	Unconditional	Always take this branch
BAL	Always	Always take this branch
BEQ	Equal	Comparison equal or zero result
BNE	Not equal	Comparison not equal or non-zero result
BPL	Plus	Result positive or zero
BMI	Minus	Result minus or negative
BCC	Carry clear	Arithmetic operation did not give carry-out
BLO	Lower	Unsigned comparison give lower
BCS	Carry set	Arithmetic operation gave carry-out
BHS	Higher or same	Unsigned comparison gave higher or same
BVC	Overflow clear	Signed integer operation; no overflow occurred
BVS	Overflow set	Signed integer operation; overflow occurred
BGT	Greater than	Signed integer comparison gave greater than
BGE	Greater or equal	Signed integer comparison gave greater or equal
BLT	Less than	Signed integer comparison gave less than
BLE	Less or equal	Signed integer comparison gave less than or equal
BHI	Higher	Unsigned comparison gave higher
BLS	Lower or same	Unsigned comparison gave low or same

Conditional Flag Setting Instructions

Branching
- Conditional Branch
- Jump

Conditional
- if
- if-else
- switch-case

Loop
- while
- for

Conditional Branch Instructions

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	if C=1 and Z=0

BLT target	; Branch if signed less than	if (~N&V | N&~V)=1 if N<>V
BGE target	; Branch if signed less than or equal to	if (~N&V | N&~V)=0 if N=V
BGT target	; Branch if signed greater than or equal to	if (Z|~N&V|N&~V)=0 if Z=0 and N=V
BLE target	; Branch if signed greater	if (Z|~N&V|N&~V)=1 if Z=1 and N<>V

Introduction to ARM Cortex-M Microcontrollers – Embedded Systems, Jonathan W. Valvano

if-then Instructions (1)

```
    LDR R2, =G              ; R2 = &G
    LDR R0, [R2]            ; R0 = G
    CMP R0, #7              ; is G == 7 ?
    BNS next1               ; if not, skip
    BL  GEqual7             ; G == 7
Next1:
```

```
uint32_t G;
if (G == 7) {
  GEqual7();
}
```

```
    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:
```

```
uint32_t G;
if (g != 7) {
  GNotEqual7();
}
```

if-then Instructions (2)

```
    LDR     R2, =G1          ; R2 = &G1
    LDRB    R0, [R2]         ; R0 = G1
    CMP     R0, #100         ; is G1 > 100?
    BLS     next             ; if not, skip to end
    MOV     R1, #1           ; R1 = 1
    LDR     R2, =G2          ; R2 = &G2
    STRB    R1, [R2]         ; G2 = 1
next:
```

```
uint32_t G1, G2;
if (G1 > 100) {
  G2 = 1;
}
```

if-then Instructions (3)

```
    LDR     R2, =G1          ; R2 = &G1
    LDRSB   R0, [R2]         ; R0 = G1 (signed)
    CMP     R0, #100         ; is G1 > 100?
    BLE     next             ; if not, skip to end
    MOV     R1, #1           ; R1 = 1
    LDR     R2, =G2          ; R2 = &G2
    STRB    R1, [R2]         ; G2 = 1
next:
```

```
uint32_t G1, G2;
if (G1 > 100) {
  G2 = 1;
}
```

if-then-else Instructions (1)

```
    LDR R2, =G1             ; R2 = &G1
    LDR R0, [R2]            ; R0 = G1
    LDR R2, =G2             ; R2 = &G2
    LDR R1, [R2]            ; R1 = G1
    BHI high                ; is G1 > G2 ?
low BL  is Less Eq          ; if so, skip to high
    B   next                ; G1 > G2
hi  BL  isGreater
next:
```

```
uint32_t G1, G2;
if (G1 > G2) {
  isGreater();
}
else {
  isLessEq();
}
```

Introduction to ARM Cortex-M Microcontrollers – Embedded Systems, Jonathan W. Valvano

if-then-else Instructions (2)

```
LDR R2, =G1              ; R2 = &G1
```

```
a = (b==1) ? 10 : 1;
```

```
if (b==1)
    a = 10;
else
    a = 1;
```

switch Instruction

```
uint32_t Last=10;          ; R2 = &G1

void OneStep(void) {
  uint32_t next;

  switch (Last) {
    case 10:   next =9; break;
    case 9:    next =5; break;
    case 5:    next =6; break;
    case 6:    next =10; break;
    default:   next =10;
  }
  GPIO_PORTD_DATA_R = next;
  Last = next;
}
```

```
a = (b==1) ? 10 : 1;

if (b==1)
    a = 10;
else
    a = 1;
```

while Loop Instructions

```
        LDR R4, =G1          ; R4 = &G1
        LDR R5, =G2          ; R5 = &G5
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
```

```
uint32_t G1, G2;

while (G2 > G1) {
  Body();
}
```

do-while Loop Instructions

```
        LDR  R1, =PF1       ; R1 = &PF1                 // toggle PF1 while
        LDR  R5, =PA5       ; R5 = &PA5                 // PA5 is low
loop    LDR  R0, [R1]       ; R0 = PF1
        EOR  R0, #2         ; toggle bit 1 of R0        do {
        STR  R0, [R1]       ; PF1 = R0                    PF1 = PF1 ^ 0x02;
        LDR  R2, [R5]       ; R2 = PA5                  } while (!(PA5&0x20));
        ANDS R2, #0x20      ; if bit 5 of R2 set?
        BEQ  loop           ; then loop
next
```

for Loop Instructions (1)

```
        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            ; go to loop
done
```

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

for Loop Instructions (2)

```
        MOV  R4, #0          ; R4 = 0
loop                         ; i
```

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

Conditional Flag Setting Instructions

```
        CMP r3, r4
        BNE Else
        ADD r0, r1, r2
        B   Exit
Else:   SUB r0, r1, r2
Exit:
```

Conditional Flag Setting Instructions

```
CMP R1, R2
BEQ L1

CMP R1, R2
BNE L1
```

Conditional Flag Setting Instructions

```
if (i==j) f = g+h;
else      f = g-h;
```

```
        CMP  r3, r4
        BNE Else          ; go to Else if I <> j

        ADD r0, r1, r2    ; f =g+h

        B Exit

Else:   SUB r0, r1, r2    ; f = g+h
Exit:
```

Conditional Flag Setting Instructions

```
while (save[i] == k)
    i += 1;
```

```
Loop:  ADD r12, r6, r3, LSL #2

       LDR r0, [r12, #0]

       CMP r0, r5
       BNE Exit

       ADD r3, r3, #1

       B Loop

Exit:
```

Conditional Flag Setting Instructions

```
CMP r0, r1

BLQ L1    ; unsigned branch
BLT L2    ; signed branch



CMP r1, r2
BHS L3    ; Index out of bounds
```

Switch – case

```
switch (amount) {
        case 20: fee = 2; break;
        case 50: fee = 3; break;
        case 100: fee = 5; break;
        default: fee = 0;
}

If (amount == 20) fee = 2;
else If (amount == 50) fee = 2;
else If (amount == 100) fee = 5;
else  fee = 0;
```

While Loop

```
int pow = 1;
int x = 0;

while (pow ~= 128)
{
  pow = pow * 2;
  x = x + 1;
}
```

For Loop (1)

```
int sum = 0;

for (i=0; I != 10; i=i+1) {
  sum = sum + I;
}
```

```
int sum = 0;
int i= 0;

while (I != 10) {
  sum = sum + I;
  i = I + 1;
}
```

For Loop (2)

```
int sum = 0;

for (i=0; I < 101; i=i*2) {
  sum = sum + I;
}
```

For Loop (2)

```
int sum = 0;

for (i=0; I < 101; i=i*2) {
  sum = sum + I;
}
```

References

[1] ftp://ftp.geoinfo.tuwien.ac.at/navratil/HaskellTutorial.pdf
[2] https://www.umiacs.umd.edu/~hal/docs/daume02yaht.pdf