Flow Control Instructions

Program Flow

Flow control instructions are used to divert the flow of the program.

These instructions are used to implement loops and subroutine calls.

The basic instruction is the Branch.

Conditional affixes can be added to the Branch instructions to enable choices.

Allow conditional execution of codes when coupled with the status bit operation instructions.

Branch Instructions

Branch instructions change the flow of a program

- by modifying the program counter (register r15)

The most basic branch instruction is

`B`: Branch

Example:

```
          B     _Label0 ; branch to _label0
          :
_Label0: MOV    r1, r2
```

The branch becomes more flexible when it is used together with the conditional codes.

Branches on ARM 7TDMI

`b`: Branch typically used with Condition Codes, `b<cond code>`

SPECIAL FORMS OF BRANCH:

`bx`: Branch and Exchange: Direct branch with registered value AND Switches from 32-bit ARM to 16-bit THUMB Instructions

`bl`: Branch and Link: `r14` (or `lr`) the Link Register holds a Return Address after Branch Instruction.
Used with subroutines
- Return Address (after `bl` Inst) is in `r14` (or `lr`) (proc stores)
- Value of `r14` (or `lr`) placed into `r15` (or `pc`) at end of subroutine (programmer must do this)

Condition Codes

Conditional codes (16 altogether) are used as affixes with the branch to enable choices depending on the settings of the various CPSR's status bits/flags (Z, N, C, V).

Some commonly used examples:

	CHECK	MEANING
EQ	Z=1	equal to Zero
NE	Z=0	not equal (Zero)
GE	N=V	greater than or equal to (signed)
LT	N!=V	less than (signed)
GT	Z=0 & N=V	greater than (signed)
LE	Z=1 | N!=V	less than or equal to (signed)
CS	C=1	carry set
CC	C=0	carry clear

Conditional Branch

A conditional branch is constructed by adding the conditional suffix to the basic branch instruction.
The branch will take effect if the condition is met.

Examples:

BEQ	Branch on equal
BNE	Branch on not equal
BGE	Branch when greater or equal to (signed)
BLT	Branch when less than (signed)

To use these instructions, they have to be preceded with the appropriate data processing instructions that set the respective status flags.

More Condition Codes

EQ	Z=1	equal to Zero
NE	Z=0	not Zero
GE	N >= V	greater or equal to (signed)
LT	N != V	less than (signed)
GT	Z=0 & N=V	greater than (signed)
LE	Z=1 | N!=V	less or equal to (signed)
CS/HS	C=1	carry set (unsigned higher or equal)
CC/LO	C=0	carry clear (unsigned lower)
MI	N=1	Negative
PL	N=0	Positive or Zero
VS	V=1	Overflow
VC	V=0	No Overflow
HI	C=1 & Z=0	Unsigned >
LS	C=0 | Z=1	Unsigned <=
AL	always	Default

Conditional Branch Example

Preceding the conditional branch with the status flag setting operating instructions.

Example:

```
          CMP   r0, #0    ; r0 = 0? set/reset Z
          BEQ   _label0   ; if equal (Z=1), branch
            :
_label0: LDR    r1, [r2] ; execute for r0 = 0
```

Equivalent code:

```
          SUBS  r0, r1, #0 ; r0 = r1 - 0
          BEQ   _label0    ; if Z = 1, branch
            :
_label0: LDR    r1, [r2]   ; execute for r0 = 0
```

Branch and Link

This instruction is used for executing a subroutine/function call, where the program has to return to the code after the branch instruction.

`BL`: Branch and link

The link register (`LR`, which is `r14`) saves the address of the next instruction after `BL` before executing the branch.

When the function completes its execution, it loads its `PC` with the `LR` register value to effect the return.

Subroutine Call

Example (use of BL)

```
     :
    BL    _subr; branch to subroutine, lr holds
          ; address of next instr. (mov)
    MOV   r1, r4       ; get the value in r4 into r1
     :

_subr:                 ; subroutine starts here
     :
    MOV   r4, #rslt    ; store return result in r4
    MOV   pc, lr       ; load PC with lr
                       ; cause a jump to the mov
                       ; instruction above
```

Branch and Exchange

This branch instruction provides the mechanism for the processor to change between the 32-bit ARM state and the 16-bit Thumb® state. (see more later)

It also takes a register as its argument (instead of a label as in the case of `B` and `BL`).

Example:

```
BX    r14     ; load the PC with the content of r14 & branch
```

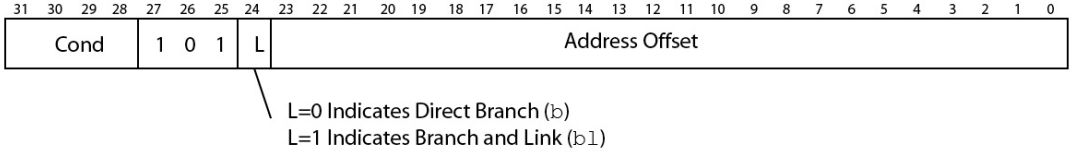
It is usually also used to return from function when changing states, in place of the instruction `MOV pc,lr`

Subroutine Call

Example (use of BX)

```
     :
    MOV   r1, r2       ; get value of r2 into r1
    BL    _subr; branch to subroutine, lr holds
          ; address of next instr. (MOV)
    MOV   r1, r4       ; get the value in r4 into r1
     :

_subr:                 ; subroutine starts here
    CMP    r1, #0      ; test r1 = 0?
    BXEQ   lr          ; if zero, change state & return
    MOV    r4, #rslt   ; store return result in r4
    BX     lr          ; load PC with lr & branch
                       ;      & change state
```

`b` And `bl` Formats



- After Instruction is Decoded, bits 23:0 are Added to the `pc` (`r15`)
- Before Adding, bits 23:0 are Left Shifted by 2 bits (Instructions are Aligned in Memory)
- Jump Ranges is then +/- 2^{25}
- Offset is Signed – 26 bits since MSb is Sign bit
- What if Farther Jump is Required?

Jumps Farther than 32k

- What if Farther Jump is Required?
- `r15` can be Treated as General Purpose Register

```
ldr    pc, =0xbe000000
```

OR

```
mov  pc, #0x04000000
```

OR

```
ldr  r3, =0x055000aa
bx   r3     ;r15 <- r3 & change state
```

`for` Loop

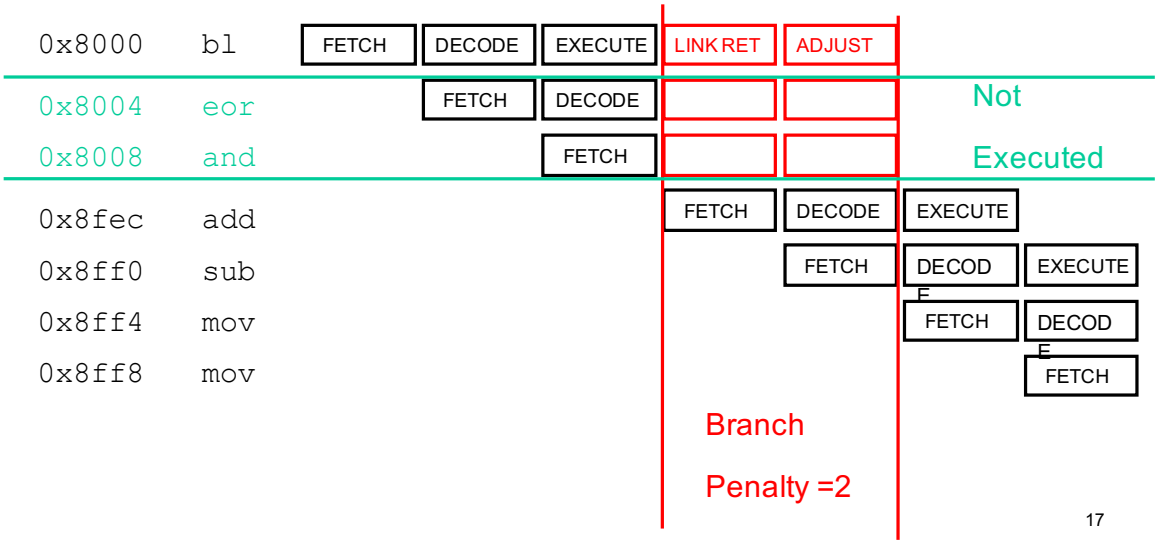
```
;       for (j=0; j<10; j++) {instructions}

        mov    r1, #0              ;j <- 0
LOOP    cmp    r1, #10             ;j<10 ?
        bge    DONE
        ;
        ; inner loop instructions here
        ;
        add    r1, r1, #1          ;increment j  (j++)
        b      LOOP
DONE
```


Instructions with Condition Codes

- Branch Penalties due to ***Control Hazards***
  - Can Cost Many Clock Cycles
- ARM's Conditional Instructions with Suffixes Can Reduce Frequent Pipeline Flushing
- When Condition not met, a `nop` is Executed Resulting in a Single Cycle
- No Need to Flush Instruction Pipeline

Control Hazards in the Pipeline

- Why Architects DON'T LIKE "jumps"
- ARM7TDMI 3-stage Instruction Pipeline in Datapath



for Loop

```
;       for (j=0; j<10; j++) {instructions}

        mov   r1, #0            ;j <- 0
LOOP    cmp   r1, #10           ;j<10 ?
        bge   DONE
        ;
        ; inner loop instructions here
        ;
        add   r1, r1, #1        ;increment j (j++)
        b     LOOP
DONE
```

POTENTIAL CONTROL HAZARDS HERE

`for` Loop – Better Way

```
;       for (j=0; j<10; j++) {instructions}

        mov   r1, #10           ;j <- 10
LOOP
        ;
        ; inner loop instructions here
        ;
        subs  r1, r1, #1        ;decrement j
        bne   LOOP
DONE
```

NOW ONLY HAVE A SINGLE BRANCH INSTRUCTION

Another Example of Loop

```
;       for (i=0; i<8; i++)
;           {
;           a[i] = b[7-i];
;           }
        AREA Prog8b, CODE, READONLY
BASE    EQU  0x8000
        ENTRY
        mov     r0, #0  ;i <- 0
        adr     r1, arrayb      ;load address of array (pseudo-inst)
        mov     r2, #BASE       ;a[i] starts here
LOOP    cmp     r0, #8          ;i=8 ?
        bge     DONE            ;if i<8, proceed
        rsb     r3, r0, #7      ;index <- 7-i
        ldrb    r5, [r1, r3]    ;load b[7-i]
        strb    r5, [r2, r0]    ;store into a[i]
        add     r0, r0, #1      ;increment i
        b       LOOP
DONE    b       DONE
        ALIGN
arrayb  DCB     0xA, 0x9, 0x8, 0x7, 0x6, 0x5, 0x4, 0x3
        END
```

Another Example of Loop

```
;       sum = 0;
;       for (i=0; i<6; i++)
;           {
;           sum += a[i];
;           }

          AREA Prog8c, CODE, READONLY
          ENTRY
          mov     r0, #0          ;sum <- 0
          mov     r1, #5          ;# of elements - 1
          adr     r1, arraya      ;load address of array
LOOP      ldr     r3, [r2, r1, LSL #2]   ;load value from memory
          add     r0, r3, r0      ;sum += a[i]
          subs    r1, r1, #1      ;i <- i -1
          bge     LOOP
DONE      b       DONE
          ALIGN
arrayb    DCB     -1, -2, -3, -4, -5, -6
          END
```


Euclid's Algorithm

- Method for Finding Greatest Common Divisor (GCD) of Two Values
- Key Element of Many Encryption and Other Arithmetic Algorithms
- GCD is Largest Value that Divides Two Numbers with a Zero-valued Remainder
- Example:

  GCD(9,12)=3

  GCD(252,105)=21

Euclid's Algorithm

- Idea is to Determine Largest of Two Values and Subtract Smaller from Larger
- Repeat Until One of the Values Becomes Zero (OR two arguments are equal)
- Example:

GCD(252,105)=GCD[(252-105),105]
=GCD(147,105)=GCD[(147-105),105)
=GCD(42,105)=GCD[(105-42),42]
=GCD(63,42)=GCD[(63-42),42]
=GCD(21,42)=GCD[(42-21),21]
=GCD(21,21)=GCD[(21-21),21]
=GCD(21,0)=21

Do While Example – Euclid's Alg

```
;      while (a != b) {
;         if (a>b) a = a - b;
;         else b = b - a;
;          }

GCD     cmp     r0, r1          ;a > b ?
        beq     DONE            ;if a=b, done
        blt     LESS            ;load address of array
        sub     r0, r0, r1      ;a <- a - b
        b       GCD             ;loop again
LESS    sub     r1, r1, r0      ;b <- b - a
        b       GCD
DONE    b       DONE
```

- This Code has Many Branches and Delay Penalties

Do While Example – Euclid's Alg

```
;      while (a != b) {
;         if (a>b) a = a - b;
;         else b = b - a;
;          }

GCD     cmp     r0, r1          ;a > b ?
        subgt   r0, r0, r1      ;a <- a-b if a>b
        sublt   r1, r1, r0      ;b <- b-a if a<b
        bne     GCD             ;loop if a != b
```

- This Code Only has the Loop Branch !!!

Conditional Example

```
;    if (char == '!' || char == '?') found++;


       teq   r0, #'!'     ;if {r0='!'} Z=1
       teqne r0, #'?'     ;if Z=0, check for
{r0='?'}
       addeq r1, r1, #1   ;if Z=1, r1 <- r1+1
```

- First Instruc XORs bit-by-bit, Sets Z
- Second Instruc Only Executes if Z=0
- Third Instruction Increments only if Found

Loop Unrolling

```
        mov     r1, #3                  ;j=3
LOOP    mla     r3, r2, r4, r5          ;r3 <- r2*r4 + r5
        subs    r1, r1, #1              ;j<-j-1 & set flags
        bne     LOOP                    ;if Z=0, branch
```

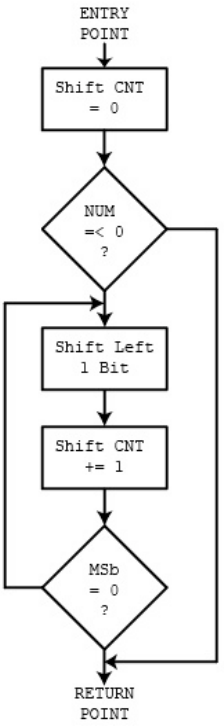
- aka Straight-Line Coding
- To Avoid Control Hazards

```
        mla     r3, r2, r4, r5          ;r3 <- r2*r4 + r5
        sub     r1, r1, #1              ;iteration 1
        mla     r3, r2, r4, r5          ;r3 <- r2*r4 + r5
        sub     r1, r1, #1              ;iteration 2
        mla     r3, r2, r4, r5          ;r3 <- r2*r4 + r5
        sub     r1, r1, #1              ;iteration 3
```


Normalization Example

- Many Arithmetic Operations Require the MSb in a Register to be “1” for maximum precision
- Floating Point Mantissa is Example – Allows for Maximizing Quantity Resolution
- ARM Instruction Set Incorporates a Special Instruction for this Purpose, `clz` in Version 5Te
- Not Present in Version 4 (ARM7TDMI)
- Must Write a Program to Perform this Operation

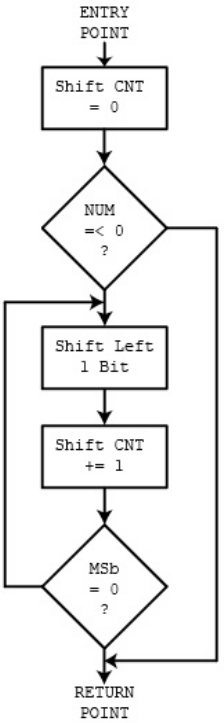
Normalization Example




```
        AREA Prog8a, CODE, READONLY
        ENTRY
Main    mov   r4, #0          ;clear shift count
        cmp   r3, #0          ;orig value <=0?
        ble   Finish          ;if yes, Finish
Loop    movs  r3, r3, LSL #1  ;shift 1 bit
        add   r4, r4, #1      ;incr. shift count
        bpl   Loop            ;branch if N=0
Finish  b     Finish          ;processing complete
        END
```

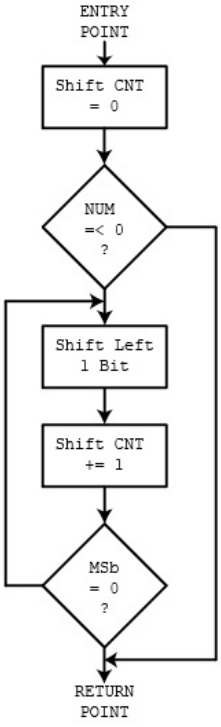
POTENTIAL CONTROL HAZARDS HERE

Normalization Example



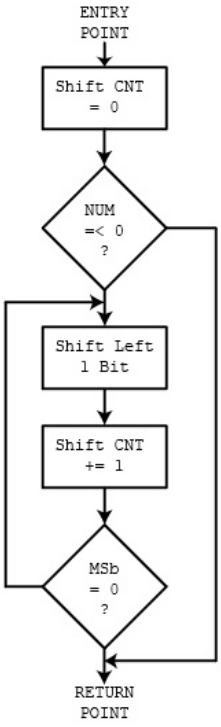
- Rewrite Normalization Code with Loop Unrolling
- First Check 8 MSbs
- Then Check 4 MSbs
- Then Check 2 MSbs
- Then Check 1 MSb

Normalization Example




```
; Normalization on ARM7TDMI, Argument in r0
; Shift count returned in r1
SHIFT   RN      r0              ;alias for r0
X       RN      r1              ;alias for r1
        AREA Prog8d, CODE, READONLY
        ENTRY
        mov     SHIFT,#0            ;SHIFT=0
        cmp     X,#1<<16            ;check (X-(1<<16))
        movcc   X, X, LSL#16        ;{X<-X<<16 if C=0;
        addcc   SHIFT, SHIFT, #16 ;SHIFT +=16 if C=0; }
        tst     X, #0xff000000      ;if(X<(1<<24))
        moveq   X, X, LSL#8         ;{ X<-X<<8 if Z=1;
        addeq   SHIFT, SHIFT, #8    ;SHIFT+=8 if Z=1; }
        tst     X, #0xf0000000      ;if(X<(1<<28))
        moveq   X, X, LSL#4         ;{ X<-X<<4 if Z=1;
        addeq   SHIFT, SHIFT, #4    ;SHIFT+=4 if Z=1; }
        tst     X, #0xc0000000      ;if(X<(1<<30))
        moveq   X, X, LSL#2         ;{ X<-X<<2 if Z=1;
        addeq   SHIFT, SHIFT, #2    ;SHIFT+=2 if Z=1; }
```


Normalization Example (cont)




```
; Normalization on ARM7TDMI, Argument in r0
; Shift count returned in r1
;
;
        tst     X, #0x80000000      ;if(X<(1<<31))
        addeq   SHIFT, SHIFT, #1  ;{SHIFT+=1 if Z=1;
        moveqs  X, X, LSL#1       ;X <<= 1 if Z=1;
        moveq   SHIFT, #32          ;if(X==0)SHIFT<-32 if Z=1;}
DONE    b       DONE
        END
```