Thumb

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

This document was produced by using LibreOffice.

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

ARM assembler in Raspberry Pi
Roger Ferrer Ibáñez

https://thinkingeek.com/arm-assembler-raspberry-pi/

Branch instructions

B,	**BL**,
BX,	**BLX**

BL and **BLX** copy the return address into **LR** (**R14**)

B,	**BL**,
BX,	**BLX**

BX and **BLX** can change the processor state

https://developer.arm.com/documentation/dui0489/c/arm-and-thumb-instructions/branch-and-control-instructions/b--bl--bx--blx--and-bxj

Branch instructions – changing the state

BLX *label* always changes the state.

ARM state → Thumb state
Thumb state → ARM state

BLX Rm changes the state depending on bit[0] of Rm:
BX Rm

Rm[0] = **0**, → ARM state

Rm[0] = **1**, → Thumb state

https://developer.arm.com/documentation/dui0489/c/arm-and-thumb-instructions/branch-and-control-instructions/b--bl--bx--blx--and-bxj

Branch and link operation (1)

Both the ARM and Thumb instruction sets
contain a primitive subroutine call instruction, **BL**,
which performs a branch-with-link operation.

LR ← the return address
the next value of the **PC**

PC ← the destination address

LR[0] ← 1 if the **BL** executed from Thumb state
LR[0] ← 0 if the **BL** executed from ARM state

The result is to transfer control to the destination address,
passing the return address in LR
as an additional parameter to the called subroutine

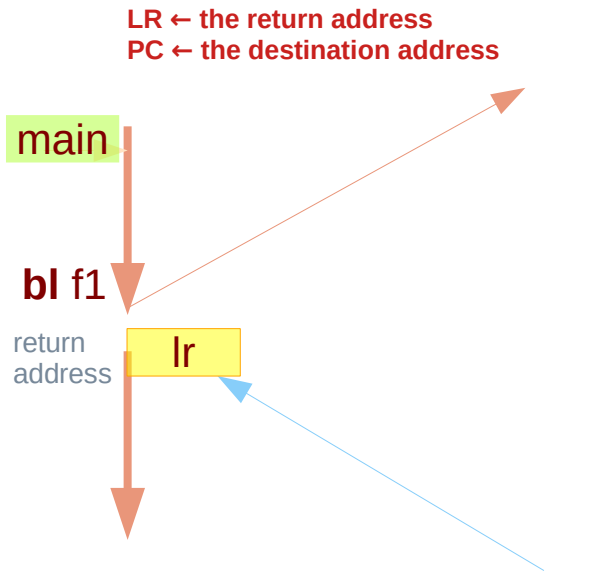
/IHI0042E_aapcs.pdf

Branch and link operation (2)

Control is returned to the instruction following the **BL**
when the return address is loaded back into the PC

A subroutine call can be synthesized
by any instruction sequence that has the effect:

LR[31:1] ← return address
LR[0] ← code type ***at*** return address
(0 ARM, 1 Thumb)
PC ← subroutine address ... return address:



LR[31:1] ← the return address
LR[0] ← 0 ARM state at the return address
LR[0] ← 1 Thumb state at the return address

/IHI0042E_aapcs.pdf

Branch and exchange operations

There are several ways to enter or leave the Thumb state properly.
The usual method is via the **Branch** and **Exchange** (**BX**) instruction.
See also **Branch**, **Link**, and **Exchange** (**BLX**) with version 5 architecture.

During the branch, the CPU examines
the least significant bit (LSb) of the destination address
to determine the new state.

Since all ARM instructions will align themselves
on either a 32- or 16-bit boundary,
the LSB of the address is not used in the branch directly.

However, if the LSB is 1 when branching from ARM state,
the processor switches to Thumb state
before it begins executing from the new address;

if 0 when branching from Thumb state,
back to ARM state it goes.

https://community.arm.com/developer/ip-products/processors/f/cortex-a-forum/5655/question-about-a-code-snippet-on-arm-thumb-state-change

32-bit / 16-bit alignment

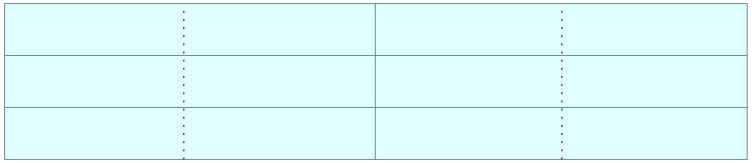
Since all ARM instructions have
either a 32- or 16-bit alignment

the LSB of the address is not used in the branch directly.

32-bit (4 bytes) - the least significant 2 bits of the target address
16-bit (2 bytes) - the least significatn 1 bit of the target address

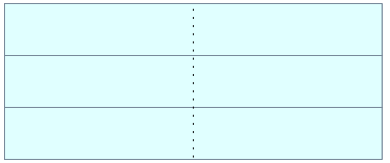
use the the least significant bit is used to change the state

data (32-bit word)



address (32-bit word)

data (16-bit halfword)



address (16-bit halfword)

https://www.cs.princeton.edu/courses/archive/fall13/cos375/ARMthumb.pdf

State changing example (1)

Change into Thumb state, then back

```
mov R0, #5                  ; argument to function is in R0
add  R1, PC, #1             ; Load address of SUB_BRANCH, Set for THUMB by adding 1
BX   R1                     ; R1 contains address of SUB_BRANCH+1

; Assembler-specific instruction to switch to Thumb

SUB_BRANCH:
BL   thumb_sub              ; Must be in a space of +/- 4 MB
add  R1, #7                 ; Point to SUB_RETURN with bit 0 clear
BX   R1

; Assembler-specific instruction to switch to ARM

SUB_RETURN:
```

thumb_sub:

https://community.arm.com/developer/ip-products/processors/f/cortex-a-forum/5655/question-about-a-code-snippet-on-arm-thumb-state-change

State changing example (2)

Change into Thumb state, then back

```
mov R0, #5
add  R1, PC, #1
BX   R1

; switch to Thumb

SUB_BRANCH:
BL   thumb_sub
add  R1, #7
BX   R1

; switch to ARM

SUB_RETURN:
```

In ARM mode, **PC** indicates 2 instructions ahead

PC of '**ADD R1,PC,#1**' is
the address of SUB_BRANCH

execution mode switch from **ARM** to **Thumb**
at the SUB_BRANCH and
the program will execute in **Thumb** mode.

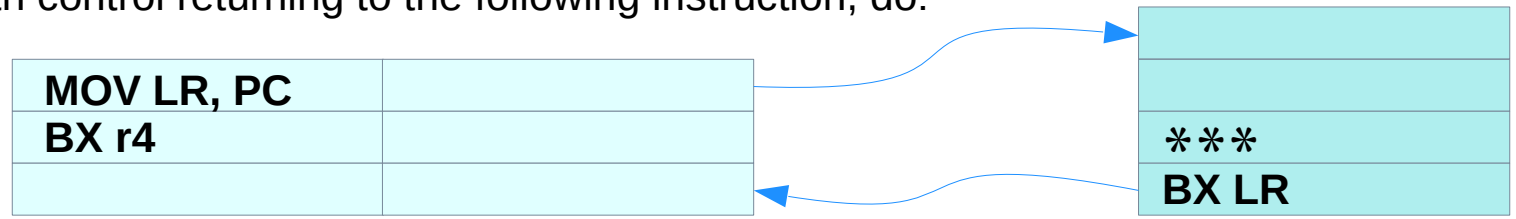
And **R1** is now 'SUB_BRANCH+1'
and by adding to 7
it will become 'SUB_BRANCH+8'.

'SUB_BRANCH+8' is
the address of 'SUB_RETURN' and
the program jumps to the address
of which LSB value is 0 and
the execution mode will become
from **Thumb** mode to **ARM** mode.

https://community.arm.com/developer/ip-products/processors/f/cortex-a-forum/

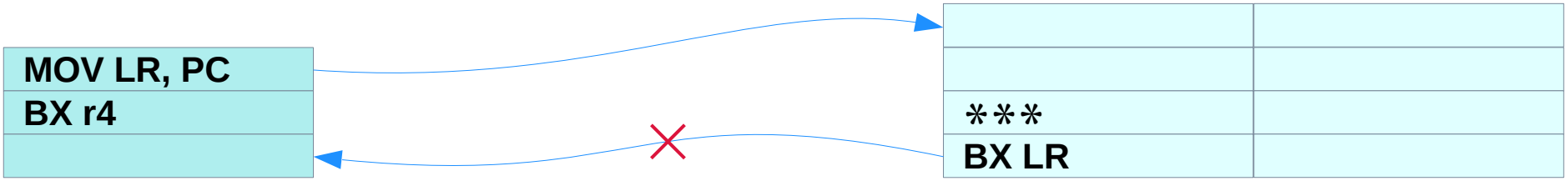
State changing example (3)

in ARM-state, to call a subroutine addressed by **r4**
with control returning to the following instruction, do:



no need explicit
LR[0] = 0, → **ARM state**

The equivalent sequence will not work from Thumb state
because the instruction that sets LR (**MOV LR, PC**)
does not copy the Thumb-state bit to **LR[0]**.



must ensure
LR[0] = 1, → **Thumb state**

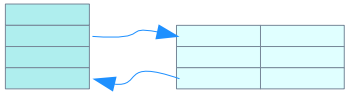
/IHI0042E_aapcs.pdf

BLX in ARM Architecture v5

In ARM Architecture v5
both ARM and Thumb state
provide a **BLX** instruction
that will call a subroutine addressed by a register
and correctly sets the return address
to the sequentially next value of the program counter.

/IHI0042E_aapcs.pdf

Thumb → ARM interworking call



to **BL** to an intermediate Thumb code segment
that executes the **BX** instruction.

the **BL** instruction loads the **link register**
immediately before the **BX** instruction is executed.

In addition, the Thumb instruction set version of **BL** sets **bit 0**
when it loads the **link register** with the **return address**.

When a Thumb-to-ARM interworking subroutine call returns
using a **BX LR** instruction, it causes the required state change
to occur automatically.

```
BL __call_via_r4
BX r4

Stop
BX r4

LR[0] = 0 → ARM state

BX LR
```

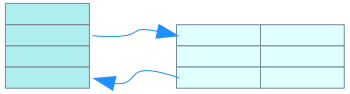
```
    CODE16
ThumbProg
    MOV    r0, #2
    MOV    r1, #3
    ADR    r4, ARMSubroutine

    BL     __call_via_r4
```

```
Stop
    MOV    r0, #0x18
    LDR    r1, =0x20026
    SWI    0xAB
__call_via_r4

    BX     r4
```

```
    CODE32
ARMSubroutine
    ADD    r0, r0, r1
    BX     LR

    END
```

https://developer.arm.com/documentation/dui0040/d/Interworking-ARM-and-Thumb/Basic-assembly-language-interworking/Implementing-interworking-assembly-language-subroutines

Thumb → ARM interworking call



If you always use the same register
to store the address of the ARM subroutine
that is being called from Thumb,
this segment can be used
to send an interworking call to any ARM subroutine.

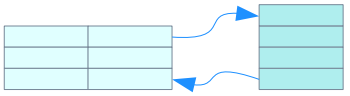
You must use a **BX LR** instruction
at the end of the ARM subroutine to return to the caller.

You cannot use the **MOV pc,lr** instruction
to return in this situation
because it does not cause
the required change of state.

```
ADR   r4, ARMSubroutine

      CODE16
ThumbProg
        ***
        ADR   r4, ARMSubroutine
        BL    __call_via_r4
        ***
__call_via_r4
        BX    r4


      CODE32
ARMSubroutine
        ***
        BX    LR
```

https://developer.arm.com/documentation/dui0040/d/Interworking-ARM-and-Thumb/Basic-assembly-language-interworking/Implementing-interworking-assembly-language-subroutines

ARM → Thumb interworking call



no need to set bit 0 of the **link register**
because the routine is returning to ARM state.

store the return address by copying **PC** into **LR**
with a **MOV lr,pc** instruction
immediately before the **BX** instruction.

Remember that the address operand to the **BX** instruction
that calls the Thumb subroutine must have bit 0 set
so that the processor executes in Thumb state on arrival.

As with Thumb-to-ARM interworking subroutine calls,
you must use a **BX** instruction to return.

LR[0] = 0 → ARM state

ADR r4, ThumbSub + 1
BX r4

```
        CODE16
        ADR         r4, ThumbSub + 1
        …
        MOV     lr, pc
        BX          r4
```

```
        CODE16
ThumbSub
        ADD     r0, r0, r1
        BX      LR
        END
```

https://developer.arm.com/documentation/dui0040/d/Interworking-ARM-and-Thumb/Basic-assembly-language-interworking/Implementing-interworking-assembly-language-subroutines

ARM → Thumb interworking call example code (1)

```
        AREA  ArmAdd,CODE,READONLY                    ; name this block of code.

        ENTRY                                         ; Mark 1st instruction to call.
                                                      ; Assembler starts in ARM mode.
main
        ADR     r2, ThumbProg + 1                     ; Generate branch target address and set bit 0,
                                                      ; hence arrive at target in Thumb state.
        BX      r2                                    ; Branch exchange to ThumbProg.
        CODE16                                        ; Subsequent instructions are Thumb.
ThumbProg
        MOV     r0, #2                                ; Load r0 with value 2.
        MOV     r1, #3                                ; Load r1 with value 3.
        ADR     r4, ARMSubroutine                     ; Generate branch target address, leaving bit 0
                                                      ; clear in order to arrive in ARM state.
        BL      __call_via_r4                         ; Branch and link to Thumb code segment that will
                                                      ; carry out the BX to the ARM subroutine.
                                                      ; The BL causes bit 0 of lr to be set.
Stop                                                  ; Terminate execution.
        MOV     r0, #0x18                             ; angel_SWIreason_ReportException
        LDR     r1, =0x20026                          ; ADP_Stopped_ApplicationExit
        SWI     0xAB                                  ; Angel semihosting Thumb SWI
__call_via_r4                                         ; This Thumb code segment will
                                                      ; BX to the address contained in r4.
        BX      r4                                    ; Branch exchange.
```

https://developer.arm.com/documentation/dui0040/d/Interworking-ARM-and-Thumb/Basic-assembly-language-interworking/Implementing-interworking-assembly-language-subroutines

ARM → Thumb interworking call example code (2)

```
        CODE32
ARMSubroutine
        ADD     r0, r0, r1          ; Add the numbers together
        BX      LR                  ; and return to Thumb caller
                                    ; (bit 0 of LR set by Thumb BL).
        END                         ; Mark end of this file.
```

https://developer.arm.com/documentation/dui0040/d/Interworking-ARM-and-Thumb/Basic-assembly-language-interworking/Implementing-interworking-assembly-language-subroutines

Thumb → ARM interworking call example code (1)

```
        AREA  ThumbAdd,CODE,READONLY     ; Name this block of code.
        ENTRY                            ; Mark 1st instruction to call.
                                         ; Assembler starts in ARM mode.
main
        MOV     r0, #2                   ; Load r0 with value 2.
        MOV     r1, #3                   ; Load r1 with value 3.
        ADR     r4, ThumbSub + 1         ; Generate branch target address and set bit 0,
                                         ; hence arrive at target in Thumb state.
        MOV     lr, pc                   ; Store the return address.
        BX      r4                       ; Branch exchange to subroutine ThumbSub.
Stop                                     ; Terminate execution.
        MOV     r0, #0x18                ; angel_SWIreason_ReportException
        LDR     r1, =0x20026             ; ADP_Stopped_ApplicationExit
        SWI     0x123456                         ; Angel semihosting ARM SWI

        CODE16                           ; Subsequent instructions are Thumb.
ThumbSub
        ADD     r0, r0, r1               ; Add the numbers together
        BX      LR                       ; and return to ARM caller.
        END                              ; Mark end of this file.
```

https://developer.arm.com/documentation/dui0040/d/Interworking-ARM-and-Thumb/Basic-assembly-language-interworking/Implementing-interworking-assembly-language-subroutines

References

[1] http://wiki.osdev.org/ARM_RaspberryPi_Tutorial_C
[2] http://blog.bobuhiro11.net/2014/01-13-baremetal.html
[3] http://www.valvers.com/open-software/raspberry-pi/
[4] https://www.cl.cam.ac.uk/projects/raspberrypi/tutorials/os/downloads.html