The Thumb instruction set

Outline:

- the Thumb programmers’ model
- Thumb instructions
- Thumb implementation
- Thumb applications

☞ hands-on: writing Thumb assembly programs
The Thumb instruction set

Outline:

- the Thumb programmers’ model
  - Thumb instructions
  - Thumb implementation
  - Thumb applications

☞ hands-on: writing Thumb assembly programs
What is Thumb?

Thumb is:

- a compressed, 16-bit representation of a subset of the ARM instruction set
  - primarily to increase code density
  - also increases performance in some cases

It is not a complete architecture

- all ‘Thumb-aware’ cores also support the ARM instruction set
  - therefore the Thumb architecture need only support common functions
The Thumb bit

- The ‘T’ bit in the CPSR controls the interpretation of the instruction stream

  - switch from ARM to Thumb (and back) by executing BX instruction
  - exceptions also cause switch to ARM code
    - return symmetrically to ARM or Thumb code
  - Note: do not change the T bit with MSR!
The Thumb programmers’ model

- Hi registers
  - SP (r13)
  - LR (r14)
  - PC (r15)

- Lo registers
  - r0
  - r1
  - r2
  - r3
  - r4
  - r5
  - r6
  - r7
  - r8
  - r9
  - r10
  - r11
  - r12

- CPSR

Shaded registers have restricted access
The Thumb programmers’ model

- Thumb register use:
  - r0 - r7 are general purpose registers
  - r13 is used implicitly as a stack pointer
    - in ARM code this is a software convention
  - r14 is used as the link register
    - implicitly, as in the ARM instruction set
  - a few instructions can access r8 - r15
  - the CPSR flags are set by data processing instructions &
    control conditional branches
The Thumb programmers’ model

- Thumb-ARM similarities:
  - load-store architecture
    - with data processing, data transfer and control flow instructions
  - support for 8-bit byte, 16-bit half-word and 32-bit data types
    - half-words are aligned on 2-byte boundaries
    - words are aligned on 4-byte boundaries
  - 32-bit unsegmented memory
The Thumb programmers’ model

- Thumb-ARM differences:
  - most Thumb instructions are unconditional
    - all ARM instructions are conditional
  - most Thumb instructions use a 2-address format
    - most ARM instructions use a 3-address format
  - Thumb instruction formats are less regular
    - a result of the denser encoding
  - Thumb has explicit shift opcodes
    - ARM implements shifts as operand modifiers
The Thumb instruction set

Outline:

- the Thumb programmers’ model
- Thumb instructions
- Thumb implementation
- Thumb applications

hand-on: writing Thumb assembly programs
Thumb branch instructions

1. B<cond> <label>

2. B <label>

3. BL <label>

4. BX Rm

<table>
<thead>
<tr>
<th>15</th>
<th>12 11</th>
<th>8 7</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 1 0 1</td>
<td>cond</td>
<td>8-bit offset</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>15</th>
<th>12 11</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 1 1 0 0</td>
<td>11-bit offset</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>15</th>
<th>12 11 10</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 1 1 1</td>
<td>H</td>
<td>11-bit offset</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>15</th>
<th>12 11</th>
<th>7 6 5 3 2</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 0 0 0 1 1 1 0</td>
<td>H</td>
<td>0 0 0</td>
<td></td>
</tr>
</tbody>
</table>
Thumb branch instructions

- These are similar to ARM instructions except:
  - Offsets are scaled to half-word, not word
  - Range is reduced to fit into 16 bits
  - BL works in two stages:
    
    H=0: \( LR := PC + \text{signextend}(\text{offset} \ll 12) \)
    
    H=1: \( PC := LR + (\text{offset} \ll 1) \)
    
    \( LR := \text{oldPC} + 3 \)
  
  - The assembler generates both halves
  - LR bit[0] is set to facilitate return via BX
Thumb branch instructions

- **Branch and eXchange (BX)**

  - to return to ARM or Thumb caller:

    ```assembly
    BX   lr     ; replaces MOV pc, lr
    ```

- **Subroutine calls**

  - later ARMs support BLX instruction
  
  - to synthesize BLX or earlier ARM:

    ```assembly
    ADR  r0, subr + 1  ; "+ 1" to enter Thumb mode
    ADR  lr, return     ; save return address
    BX   r0             ; calls subr
    return  ...  ;
    ```
The Thumb SWI operates exactly like the ARM SWI

- the (interpreted) immediate is just 8 bits
  - Thumb Angel SWI uses value 0xAB
    r0 call value is exactly as in ARM code

- the SWI handler is entered in ARM code
  - the return automatically selects ARM or Thumb
Thumb data processing instructions

1. ADD|SUB Rd, Rn, Rm

2. ADD|SUB Rd, Rn, #imm3

3. MOV|CMP|ADD|SUB Rd/Rn, #imm8

4. LSL|LSR|ASR Rd, Rn, #shift
Thumb data processing instructions

(5) \( <\text{Op}> \text{ Rd/Rn, Rm/Rs} \)

(6) \( \text{ADD|CMP|MOV Rd/Rn, Rm} \)

(7) \( \text{ADD Rd, SP|PC, \#imm8} \)

(8) \( \text{ADD|SUB SP, SP, \#imm7} \)

\( \text{In case (6)}: \)

- MOV does not affect the flags
  (it can be distinguished using the mnemonic CPY after v6)
Thumb data processing instructions

Notes:

- in Thumb code shift operations are separate from general ALU functions
- in ARM code a shift can be combined with an ALU function in a single instruction
- all data processing operations on the ‘Lo’ registers set the condition codes
- those on the ‘Hi’ registers do not, apart from CMP which only changes the condition codes
Thumb single register data transfers

(1) LDR|STR{B} Rd, [Rn, #off5]

(2) LDRH|STRH Rd, [Rn, #off5]

(3) LDR|STR{S}{H|B} Rd, [Rn, Rm]

(4) LDR Rd, [PC, #off8]

(5) LDR|STR Rd, [SP, #off8]
Thumb multiple register data transfers

- These map directly onto the ARM forms:
  
  **PUSH:** STMFD SP!, {<regs>{, lr}}
  
  **POP:** LDMFD SP!, {<regs>{, pc}}

  - note restrictions on available addressing modes compared with ARM code
Unique Thumb mnemonics

Most significant differences from ARM:

- `PUSH ; STMFD sp!,{&}`
- `POP ; LDMFD sp!,{&}`
- `NEG ; RSB Rd, Rs, #0`
- `LSR ; MOV Rd, Rd, LSR <Rs | #5>`
- `ASR ; MOV Rd, Rd, ASR <Rs | #5>`
- `LSL ; MOV Rd, Rd, LSL <Rs | #5>`
- `ROR ; MOV Rd, Rd, ROR Rs`
Newer Thumb instructions (from v5)

- **BLX** works in two stages; (first is same as BL)

  15  12  11  10  1  0
  1  1  1  0  1  10-bit offset  0

  - H=0: LR := PC + signextend(offset << 12)
  - H=1: PC := (LR + (offset << 2)) AND FFFFFFFFC
  - LR := oldPC + 3
  - T flag := 0

- There is also a register-based **BLX**

  15  7  6  3  2  0
  0  1  0  0  0  1  1  1  Rm  0  0  0

  - (2) BLX Rm

- **BKPT** (Breakpoint)
Newer Thumb instructions (from v6)

- CPY
  - Mnemonic allowing register moves without affecting flags
- SXTB/SXTH/UXTB/UXTH
  - Sign extension (no shifts)
- REV/REV16/REVSH
  - Byte swaps
- SETEND
- CPSIE/CPSID
  - Interrupt enable/disables (no mode changes)

More about these in later ARM session.
ARM/Thumb interworking

- **BX** (Branch eXchange) moves to the mode specified by the address LSB (in register)

- **BLX** (Branch with Link and eXchange) moves to the *other* mode (common case)
  - the LSB of LR retains the ‘parent’ mode
  - BLX  Rm can move to either mode (like BX)

- The ‘correct’ subroutine return is:
  
  ```
  BX       LR
  ```
  
  - the routine can then be called from both ARM and Thumb code
ARM/Thumb interworking

- Calling procedures in other instruction set
  - ARM v5 or later
    - BLX procedure ; ARM or Thumb
  - ARM v4T
    - from ARM
      - ADR lr, return_addr ;
      - ADR r0, procedure + 1 ; + 1 sets ‘T’
      - BX r0 ;
      - return_addr ...
    - from Thumb
      - LDR r0, =procedure ;
      - MOV lr, pc ; ‘here’ + 4
      - BX r0 ;
      - ...
The Thumb instruction set

Outline:

- the Thumb programmers’ model
- Thumb instructions
- Thumb implementation
- Thumb applications

☞ hands-on: writing Thumb assembly programs
Thumb decoding

- The original Thumb implementation translated the opcodes into ARM opcodes.
  - This means the effect of Thumb and ARM instructions are the same
    - Thumb is more restricted (e.g. smaller offsets/immediates)
    - One or two new functions (e.g. BL details)

- Later implementations decode Thumb directly
Thumb - ARM instruction mapping

ADD Rd, #imm8

‘always’ condition

major opcode, format 3: MOV/CMP/ADD/SUB with immediate

minor opcode denoting ADD & set CC

destination and source register

zero shift

immediate value

1110001010010010 Rd 0 Rd 0000 imm8
The Thumb instruction set

Outline:

- the Thumb programmer’s model
- Thumb instructions
- Thumb implementation

→ Thumb applications

☞ hands-on: writing Thumb assembly programs
Thumb applications

- Thumb code properties:
  - 70% of the size of ARM code
    - 30% less external memory power
    - 40% more instructions
  - With 32-bit memory:
    - ARM code is 40% faster than Thumb code
  - With 16-bit memory:
    - Thumb code is 45% faster than ARM code
Thumb applications

- For the best performance:
  - use 32-bit memory and ARM code

- For best cost and power-efficiency:
  - use 16-bit memory and Thumb code

- In a typical embedded system:
  - use ARM code in 32-bit on-chip memory for small speed-critical routines
  - use Thumb code in 16-bit off-chip memory for large non-critical control routines
Hands-on: writing Thumb assembly programs

- Explore further the ARM software development tools
  - Write Thumb assembly programs
  - Check that they work as expected

☞ Follow the ‘Hands-on’ instructions