arm

Armv7-A/R ISA Overview (Part 2)

Learning Outcomes

At the end of this module, you will be able to:

- Outline behaviors of conditional and branch instructions.
- Identify operations of coprocessor instructions.
- Compute the results of DSP instructions when executed, including saturated math and count lead zero.

Flow Control

- Arm state
  - Append a two-letter suffix to the mnemonic (conditional instruction), or
  - use a conditional branch instruction.
    - Almost all Arm instructions can be executed conditionally on the value of the condition flags in the Application Program Status Register (APSR).
    - Using conditional branch instructions can be more efficient when a series of instructions depends on the same condition.
- Thumb state
  - Use an IT (If-Then) instruction, or
  - use a conditional branch instruction.
    - IT is a 16-bit instruction that enables almost all Thumb instructions to be conditionally executed, based on the value of the condition flags and the condition code suffix specified.
    - Armv6T2 or later processors: instructions can also be conditionally executed by using CBZ and CBNZ.

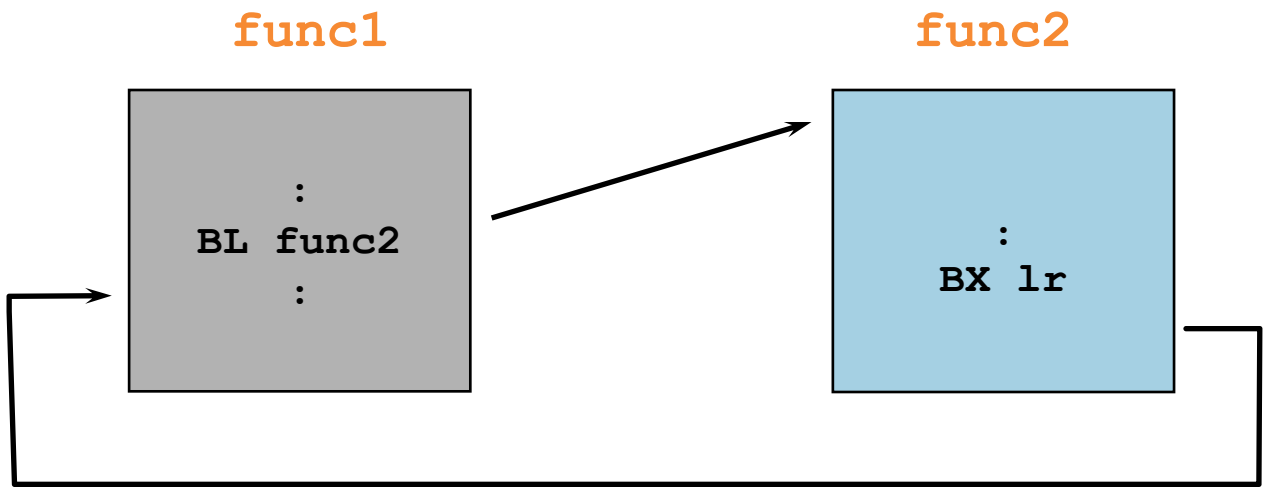
Branch Instructions

- Branch instructions have the following format:
- `B{<cond>} label`
  - Might not cause a pipeline flush (branch prediction)
  - Branch range depends on instruction set and width.

A `BL` instruction additionally generates a return address in r14 (lr).

  - Returning is performed by restoring the program counter (`pc`) from `lr`.

```
void func1 (void)
{
        :
        func2();
        :
}
```



Interworking

- Interworking can be carried out using the branch exchange instruction.

```
BX    Rn
BLX   Rn
```

- Bit 0 of `Rn` determines the exchange behavior.
  - Unset : change to (or remain in) Arm state.
  - Set: change to (or remain in) Thumb state.

Branch and link with exchange

- Used to branch to a subroutine, which is known to be in the opposite instruction set
- When branching to imported labels, use `BL`; the linker will substitute `BLX` if necessary.
-

```
BLX offset                  ; Arm/Thumb instruction which always
                            ; changes state (and sets LR)
```

All instructions that modify the `PC` can cause a state change.

- Depending on bit 0 of the result
- For data processing instructions, state changes only if S variant is snot used.

Compare and Branch if Zero

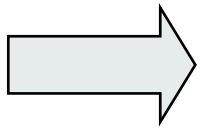
- Replaces a CMP followed by a branch
  - BUT does not affect condition code flags
- Syntax

  `CB{N}Z <Rn>, <label>`

  - `CBZ`: If Rn is equal to zero, branch to label
  - `CBNZ`: If Rn is not equal to zero, branch to label

- Can only branch forward between 4 and 130 bytes

```
   .
   CMP r0, #0
   BEQ exit
   .
exit
```



```
   .
   CBZ r0, exit
   .
   .
exit
```

Conditional Instructions

- Arm and Thumb instructions can execute conditionally on the condition flags set by a previous instruction.
- The conditional instruction can occur either:
  - Immediately after the instruction that updated the flags
  - After any number of intervening instructions that have not updated the flags
- The instructions that you can make conditional depends on whether the processor is in Arm state or Thumb state.
- To make an instruction conditional, you must add a condition code suffix to the instruction mnemonic.
  - The condition code suffix enables the processor to test a condition based on the flags.
  - If the condition test of a conditional instruction fails, the instruction:
    - Does not execute
    - Does not write any value to its destination register
    - Does not affect any of the flags
    - Does not generate any exception

Reference: Condition Codes and Flags

- The possible condition codes are listed below.
  - Note `AL` is the default and does not need to be specified.

Suffix	Description	Flags tested
EQ	Equal	Z=1
NE	Not equal	Z=0
CS/HS	Unsigned higher or same	C=1
CC/LO	Unsigned lower	C=0
MI	Minus	N=1
PL	Positive or Zero	N=0
VS	Overflow	V=1
VC	No overflow	V=0
HI	Unsigned higher	C=1 & Z=0
LS	Unsigned lower or same	C=0 or Z=1
GE	Greater or equal	N=V
LT	Less than	N!=V
GT	Greater than	Z=0 & N=V
LE	Less than or equal	Z=1 or N=!V
AL	Always	

By default, data processing instructions do not affect the condition flags but this can be achieved by placing an "S" between the instruction and any condition code.

Status Flag	Meaning
N	Negative
Z	Zero
C	Carry
V	Overflow

If-Then

- Thumb only, makes the next one to four instructions conditional
- Syntax
- `IT{T|E}{T|E}{T|E} <cond>`
  - Any condition code may be used.
  - Doesn't affect condition flags
  - 16-bit instructions in the IT block do not affect condition flags (except **CMP**, **CMN**, & **TST**).
  - 32-bit instructions do affect condition flags (normal rules apply).
  - No need to write this instruction: the assembler will insert it for you where necessary.
- Current if-then status stored in CPSR
  - Conditional block may be safely interrupted and returned to.
  - Not recommended to branch into or out of if-then block

```
; if (r0 == 0)
;   r0 = *r1 + 2;
; else
;   r0 = *r2 + 4;

; if
  CMP r0, #0
  ITTEE EQ
; then
  LDREQ r0, [r1]
  ADDEQ r0, #2
; else
  LDRNE r0, [r2]
  ADDNE r0, #4
```

Example: Conditional Execution

pseudocode

```
if (r0 == 0)
{
  r1 = r1 + 1;
}
else
{
  r2 = r2 + 1;
}
```

assembler instructions

inefficient branching

```
  CMP r0, #0
  IT NE
  BNE else
  ADD r1, r1, #1
  B end
else
  ADD r2, r2, #1
end
  ...
```

no branching

```
  CMP r0, #0
  ITE EQ
  ADDEQ r1, r1, #1
  ADDNE r2, r2, #1
  ...
```

Supervisor Call (SVC)

- Syntax
- `SVC{<cond>} <SVC number>`
  - Causes an SVC exception
  - The SVC handler can examine the SVC number to decide what operation has been requested.
    - But the core ignores the SVC number.
  - By using the SVC mechanism, an OS can implement a set of privileged operations, which applications running in user mode can request.

Coprocessor Instructions

- The instructions for each coprocessor occupy a fixed part of the Arm instruction set.
  - If the appropriate coprocessor is not present in the system, an Undefined Instruction exception occurs.
- There are three types of coprocessor instruction:
  - Data processing

    CDP: Initiate a coprocessor data processing operation.
  - Register transfer

    MRC: Move to Arm register from coprocessor register.

    MCR : Move to coprocessor register from Arm register.
  - Memory transfers

    LDC: Load coprocessor register from memory.

    STC: Store from coprocessor register to memory.
- Multiple register and other variants of these instructions are also available.

PSR Access

- MRS and MSR allow contents of CPSR to be transferred to/from a general-purpose register or take an immediate value.
  - MSR allows the whole status register, or just parts of it, to be updated.

```
MRS r0,CPSR      ; read CPSR into r0
BIC r0,r0,#0x80  ; clear bit 7 to enable IRQ
MSR CPSR_c,r0    ; write modified value to 'c' byte only
```

- **CPS** can be used to directly modify some bits in the CPSR.
- **SETEND** instruction selects the endianness of data accesses (BE8 or LE).
  - For use in systems with mixed-endian data (e.g., peripherals)

```
SETEND BE
LDR r0, [r7], #4     ; big-endian
SETEND LE
LDR r1, [r7], #4     ; little-endian
```

Miscellaneous Instructions

- Breakpoint instruction - **`BKPT <bkpt number>`**
  - Immediate value is ignored by the processor.
  - Execution of this instruction will either cause a Prefetch Abort or cause the processor to enter Debug state (depends on the core design and configuration).
  - Used by debug agents
- Wait for interrupt: **`WFI`**
  - Puts the core into Standby mode
  - Woken by an interrupt or debug event
  - Previously implemented as a CP15 operation
- No operation: **`NOP`**
  - Can be used as padding to align following instructions
  - May or may not take time to execute
- Wait for event (**`WFE`**) and send event (**`SEV`**)
  - Covered elsewhere in the course

DSP Instructions Overview

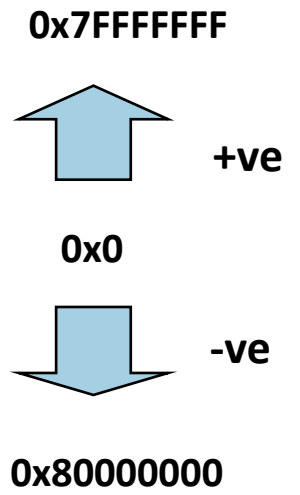
- These DSP instructions are single instruction multiple data (SIMD).
  - They operate on 8- or 16-bit quantities packed into words.
  - They allow more efficient access to packed structure types.
- Instruction groups
  - Data packing/unpacking
  - Data processing
  - Saturated maths
  - Addition/subtraction
  - Multiplication
  - Sum of absolute differences

Saturated Maths and CLZ

- Support for saturated mathematics
  - Targeted at DSP and control applications
  - Overflow sets Q flag (sticky) not V, and sets result to +/- max value.

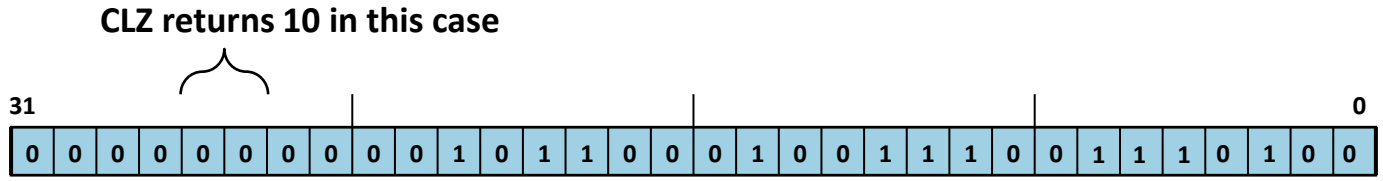
```
QSUB{cond} Rd, Rm, Rn       ; Rd = saturate(Rm - Rn)
QADD{cond} Rd, Rm, Rn       ; Rd = saturate(Rm + Rn)

QDSUB{cond} Rd, Rm, Rn      ; Rd = saturate(Rm - saturate(Rn×2))
QDADD{cond} Rd, Rm, Rn      ; Rd = saturate(Rm + saturate(Rn×2))
```



Count leading zeros

```
CLZ{cond} Rd, Rm
```



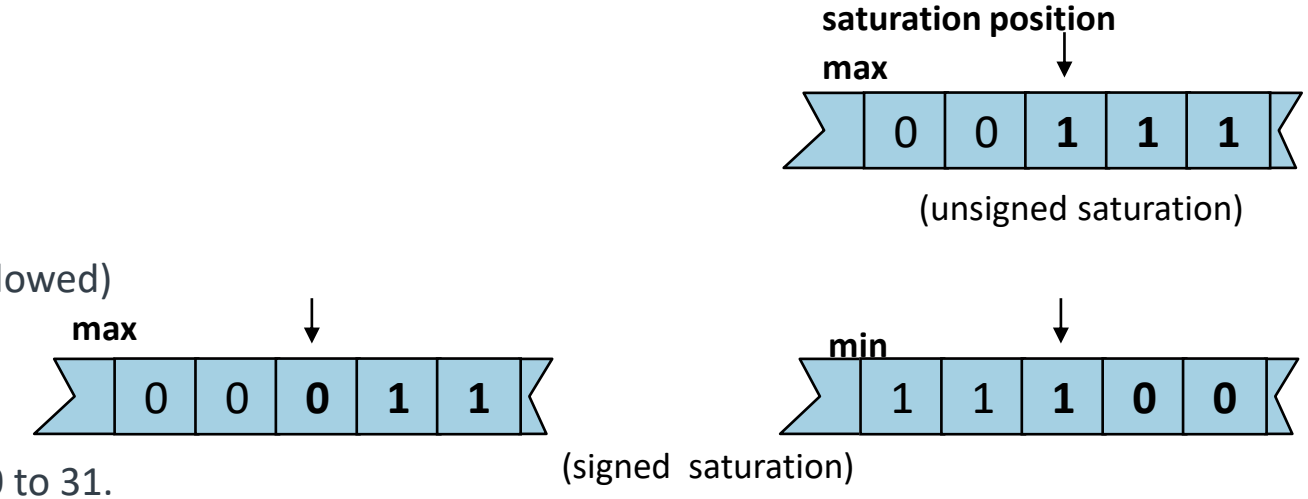
  - Returns number of unset bits before the most significant set bit

arm

Saturation

- Saturate a value to a specified bit position (effectively saturating to any power of 2).
  - USAT: Unsigned saturate 32-bit
  - Syntax: `USAT Rd, #sat, Rm {shift}`
  - Operation: `Rd = Saturate(Shift(Rm), #sat)`

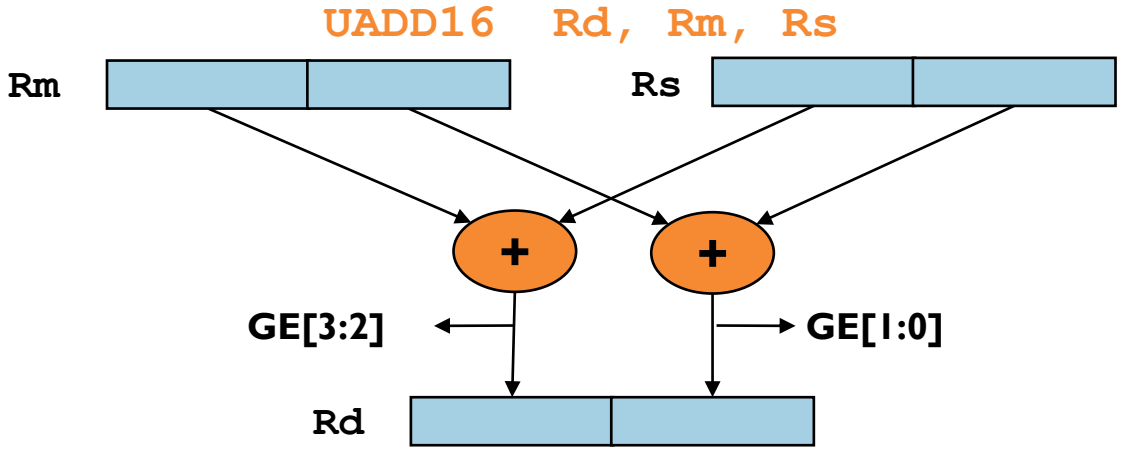
  - Variants
  - **SSAT**
  - signed saturation
  - **USAT16**
  - saturates two 16-bit unsigned halfwords (no rotation allowed)
  - **SSAT16**
  - signed saturation of two 16-bit halfwords (no rotation allowed)
  - `#sat` is specified as an immediate value in the range 0 to 31.
  - `{shift}` is optional and is limited to `LSL` or `ASR`.
  - Q flag is set if saturation occurs.



arm

SIMD

- Armv6 added a number of instructions that perform SIMD operations using Arm registers.
  - Includes instructions for addition, subtraction, multiplication, and sum of absolute differences
  - Instructions can work on four 8-bit quantities or two 16-bit quantities.
  - Signed/unsigned and saturating versions of many instructions available
  - CPSR GE bits used instead of normal ALU flags



  - There are instructions for packing (`PKHBT` and `PKHTB`) and unpacking (`UXTH` & `UXTB`) registers.

Appendix: Encoding Choice

- When assembling for a Thumb-2 processor, there is often a choice of 16-bit and 32-bit instruction encodings.
  - The assembler will normally generate 16-bit instructions.
- Thumb-2 instruction width specifiers
  - Allow you to determine which instruction width the assembler will use
  - Can be placed immediately after instruction mnemonics
  - `.W`: Forces a 32-bit instruction encoding
  - `.N`: Forces a 16-bit instruction encoding
  - Assembler will raise an error if the selected encoding is not possible.
- Disassembly rules
  - One-to-one mapping is defined to ensure correct reassembly.
  - `.W` or `.N` suffix used for cases when a bit pattern that doesn't follow the above rules is disassembled.