Microprocessors

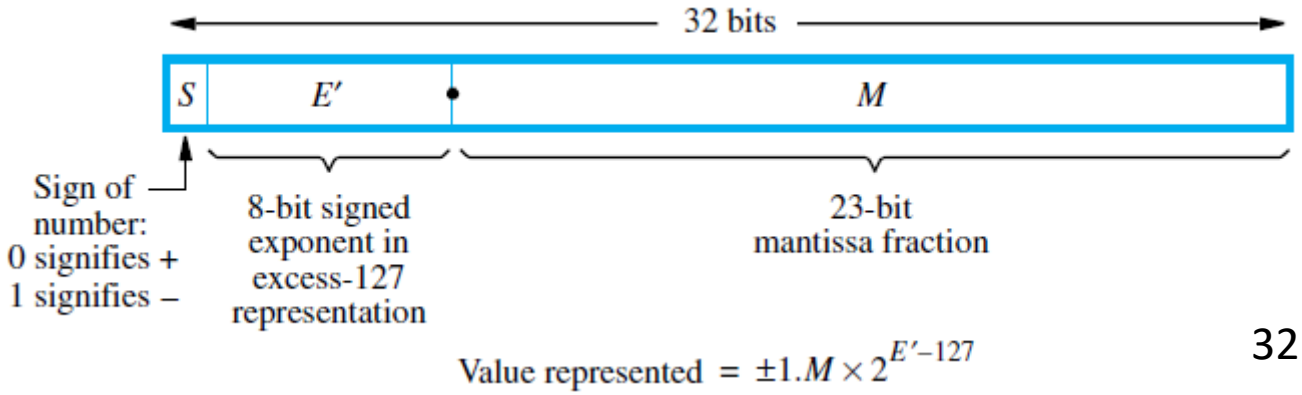
Tuba Ayhan

MEF University

Floating point arithmetic

CH9.7

IEEE Standard 754 (754-2008)

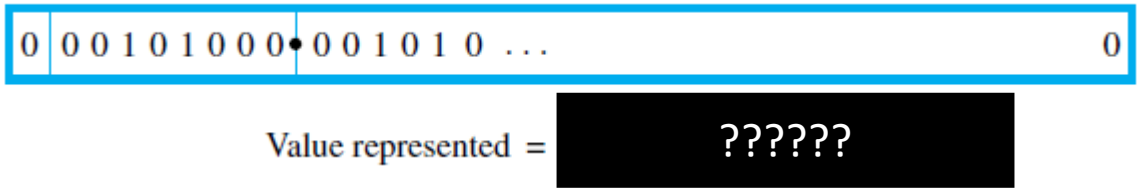


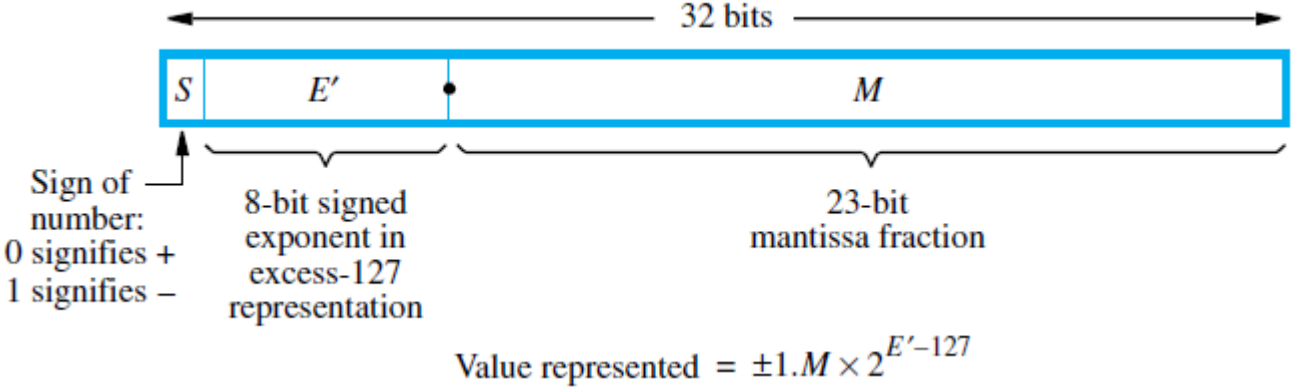
32 bit - Single precision

A binary floating-point number can be represented by

- A sign for the number
- Some significant bits
- A signed scale factor exponent for an implied base of 2

Single precision example

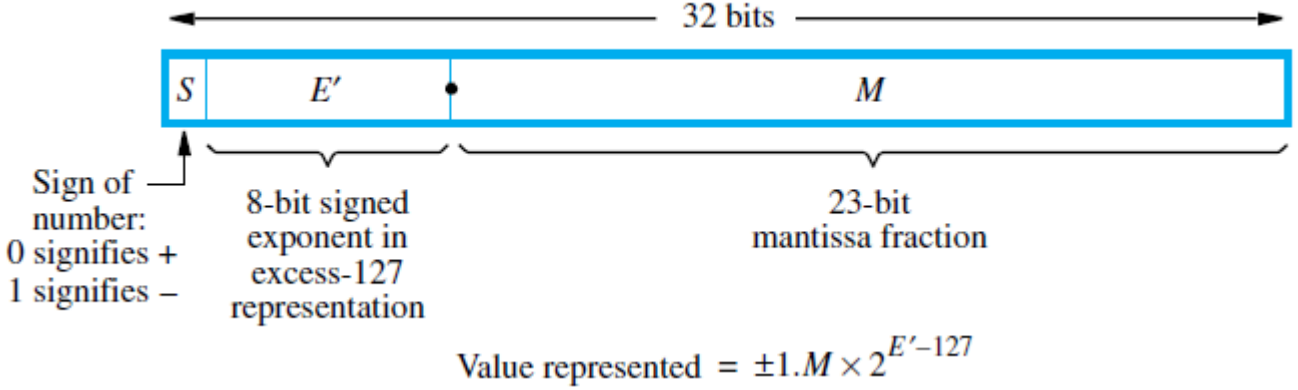




Excess-127 in single precision format

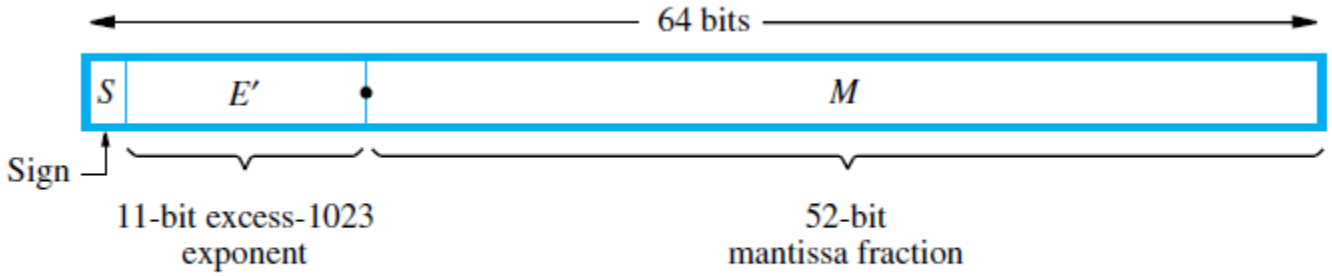
- E = E’ + 127
- 0 ≤ E’ ≤ 255
  - The end values of this range, 0 and 255, are used to represent special values.
- E, is in the range

  $-126 \leq E' \leq 127$; approximately equal to 10±38



- This representation simplifies comparison of the relative sizes of two floating-point numbers.

Double precision



$$\text{Value represented} = \pm 1.M \times 2^{E'-1023}$$

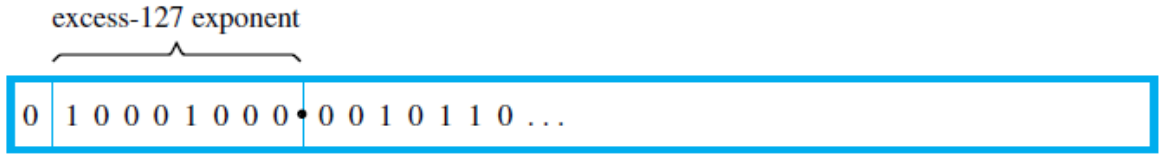
- increased exponent and mantissa ranges
- 11-bit excess-1023 exponent E’
- $-1022 \leq E \leq 1023$, → approximately $10^{\pm 308}$ d
- 53-bit mantissa provides a precision equivalent to about 16 decimal digits

IEEE Standard

- A computer must provide at least single-precision representation to conform to the IEEE standard.
- Double-precision representation is optional.
- Extended versions provide
  - increased precision
  - increased exponent range
- The use of extended formats helps to reduce the size of the accumulated round-off error in a sequence of calculations leading to a desired result. i.e.

1. Dot product of vectors involves accumulated products.
   - The input vector components are given in a standard precision (single or double),
   - The final answer (the dot product) is truncated to the same precision.
   - Intermediate calculations use **extended precision** to limit accumulation of errors.
2. Extended formats enhance the accuracy of evaluation of elementary functions such as sine, cosine.

Normalization



(There is no implicit 1 to the left of the binary point.)

Value represented = $+0.0010110\ldots\times 2^{9}$

(a) Unnormalized value

Value represented = $+1.0110\ldots\times 2^{6}$

(b) Normalized version

- Normalizing a number: by shifting the binary point and adjusting the exponent.
  - Unnormalized value, $0.0010110\ldots\times 2^{9}$
  - Normalized version$1.0110\ldots\times 2^{6}$
- A number that does not fall in the representable range of normal numbers:
- In single precision, its normalized representation requires an exponent
  - less than −126 → *underflow*
  - greater than +127 → *overflow*

Special Values

- **0 and ∞**:
  - E’ = 0 and the mantissa fraction M = 0 → the value 0 is represented.
  - E’ = 255 and M = 0, the value ∞ is represented, where ∞ is the result of dividing a normal number by 0.
  - The sign bit is still used in these representations, so there are representations for ±0 and ±∞.
- **Denormal numbers** are smaller than the smallest normal number. They allow for **gradual underflow**, providing an extension of the range of normal representable numbers.
  - $\pm 0.M \times 2^{-126}$
- **Not a Number (NaN)**:
  - When E = 255 and M is not 0.
  - i.e. 0/0

Exceptions

- A processor must set *exception* flags if any of these conditions arise:

  underflow, overflow, divide by zero, inexact, invalid.

- **Inexact** is the name for a result that requires rounding in order to be represented in one of the normal formats.
- An **invalid** exception : 0/0 or √−1 ...
- When an exception occurs, the result is set to one of the special values.
- If interrupts are enabled for any of the exception flags, system or user-defined routines are entered when the associated exception occurs.
- Alternatively, the application program can test for the occurrence of exceptions, as necessary, and decide how to proceed.

Add/Subtract Rule

1. Choose the number with the smaller exponent and shift its mantissa right a number of steps equal to the difference in exponents.
2. Set the exponent of the result equal to the larger exponent.
3. Perform addition/subtraction on the mantissas and determine the sign of the result.
4. Normalize the resulting value, if necessary.

Multiply Rule

1. Add the exponents and subtract 127 to maintain the excess-127 representation.
2. Multiply the mantissas and determine the sign of the result.
3. Normalize the resulting value, if necessary.

Divide Rule

1. Subtract the exponents and add 127 to maintain the excess-127 representation.
2. Divide the mantissas and determine the sign of the result.
3. Normalize the resulting value, if necessary.

Guard Bits and Truncation

- The mantissas of initial operands and final results are limited to 24 bits.
- During the intermediate steps, extra bits called **guard bits** are used.
- Final result → truncation is used to reduce the number of mantissa bits to 24.
- According to the standard, results of single operations must be computed to be accurate within half a unit in the LSB position.
- This means that **rounding** must be used as the **truncation method**.

Microprocessors

Tuba Ayhan

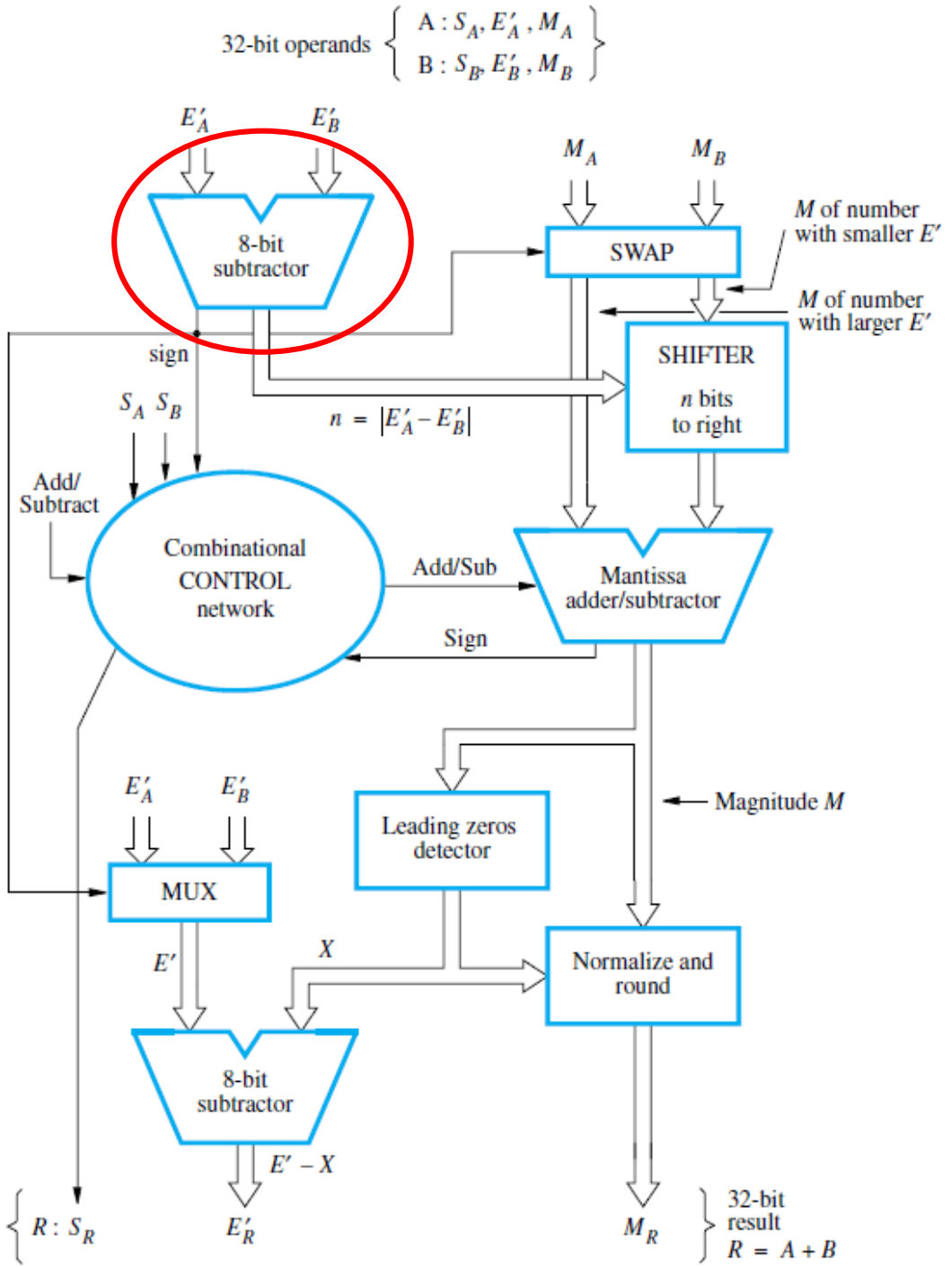
MEF University

Floating point arithmetic - Implementation

CH9.7

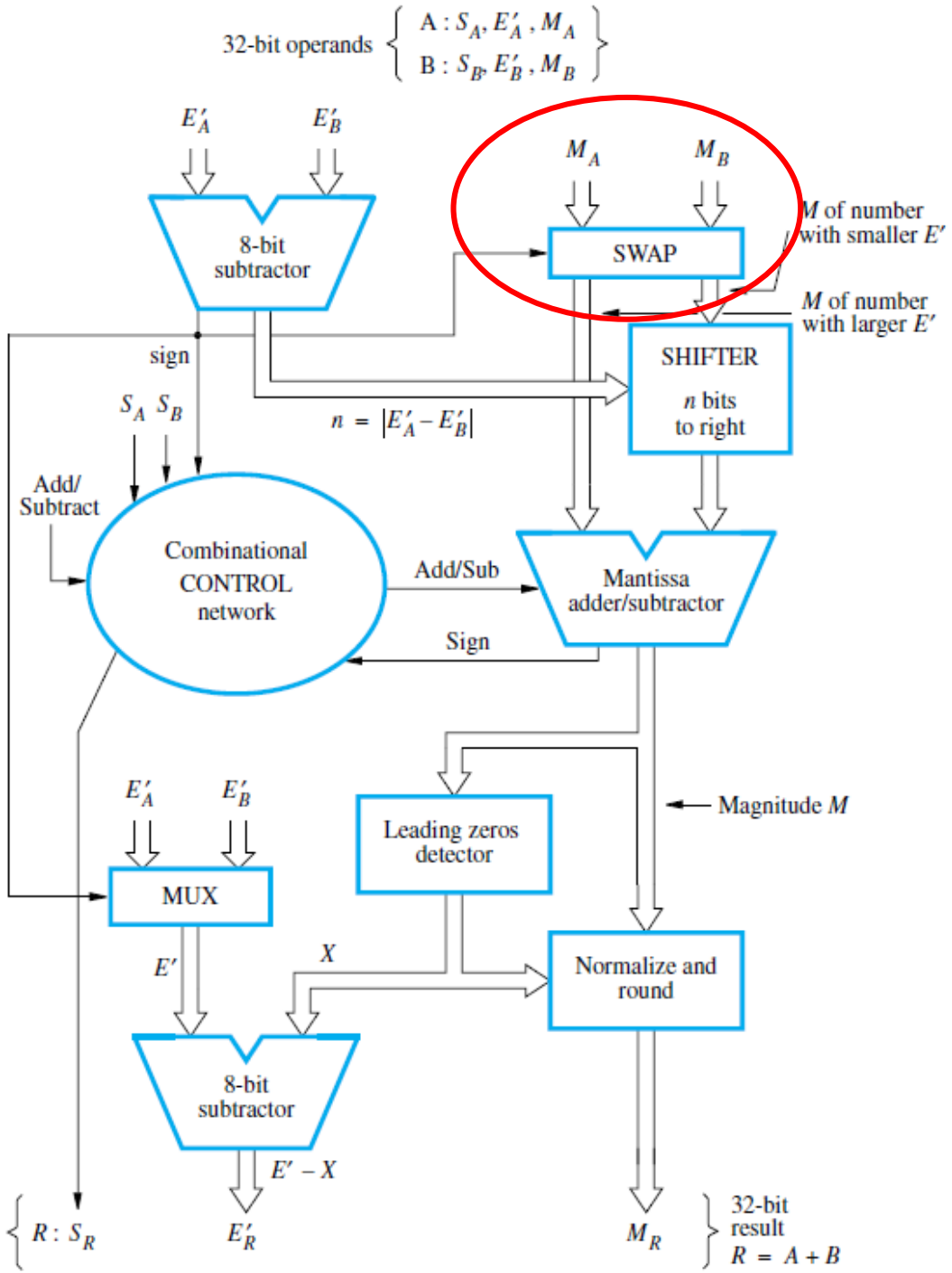
Implementing

- In many general-purpose processors, floating-point operations are available at the machine-instruction level, implemented in hardware.
- Step 1. compare exponents to determine how far to shift the mantissa of the number with the smaller exponent.
- Shift-count value, n, is determined by the 8-bit subtractor circuit. The magnitude of the difference E'A – E'B (n) is sent to the SHIFTER unit.



Implementing

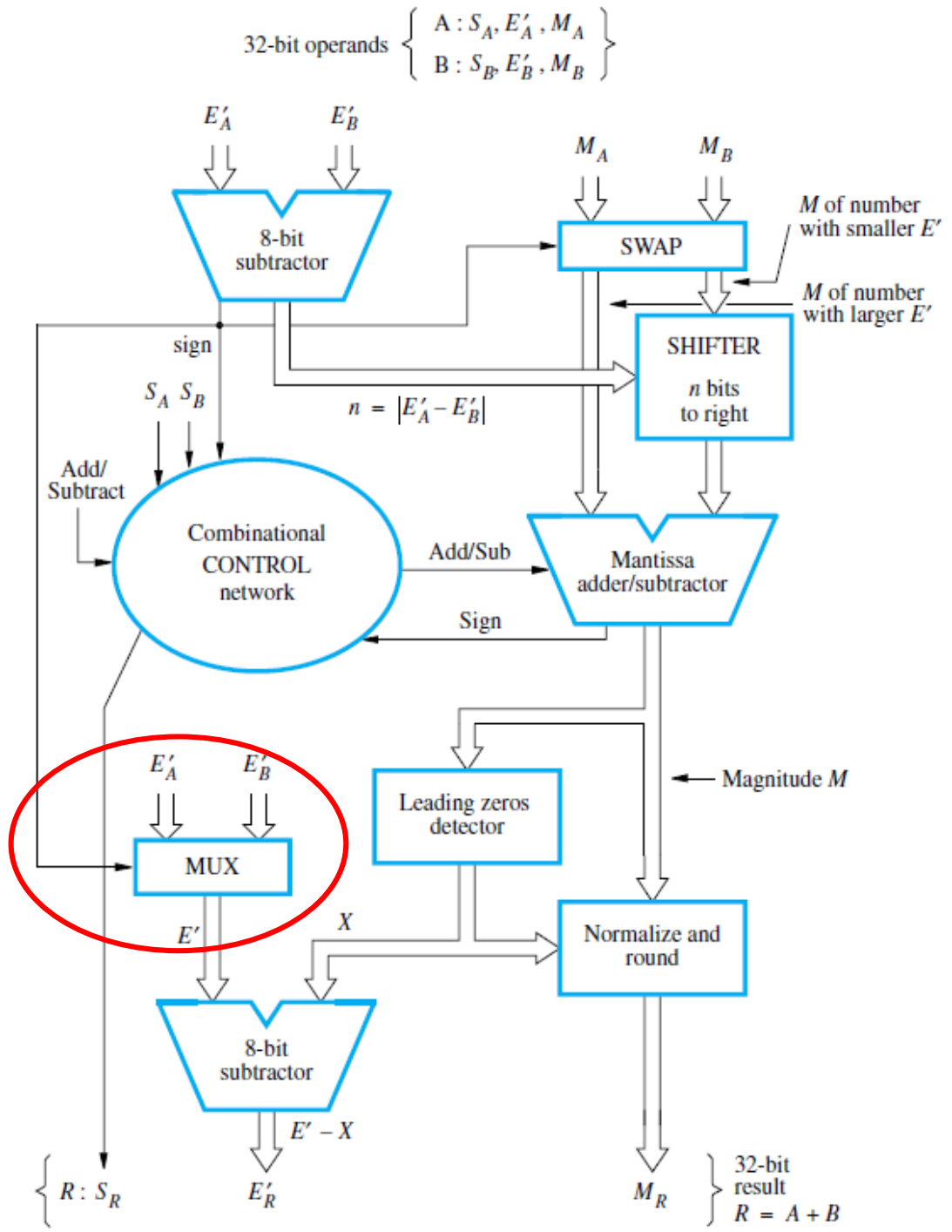
- The sign is sent to the SWAP.
  - If the sign is 0, then *EA* ≥ *EB* and the mantissas *MA* and *MB* are sent straight through the SWAP network.
  - If the sign is 1, then *EA* < *EB* and the mantissas are swapped before they are sent to the SHIFTER.
  - This results in *MB/A* is sent to the SHIFTER, to be shifted *n* positions to the right. The other mantissa, *MA/B*, is sent directly to the mantissa adder/subtractor.



Implementing

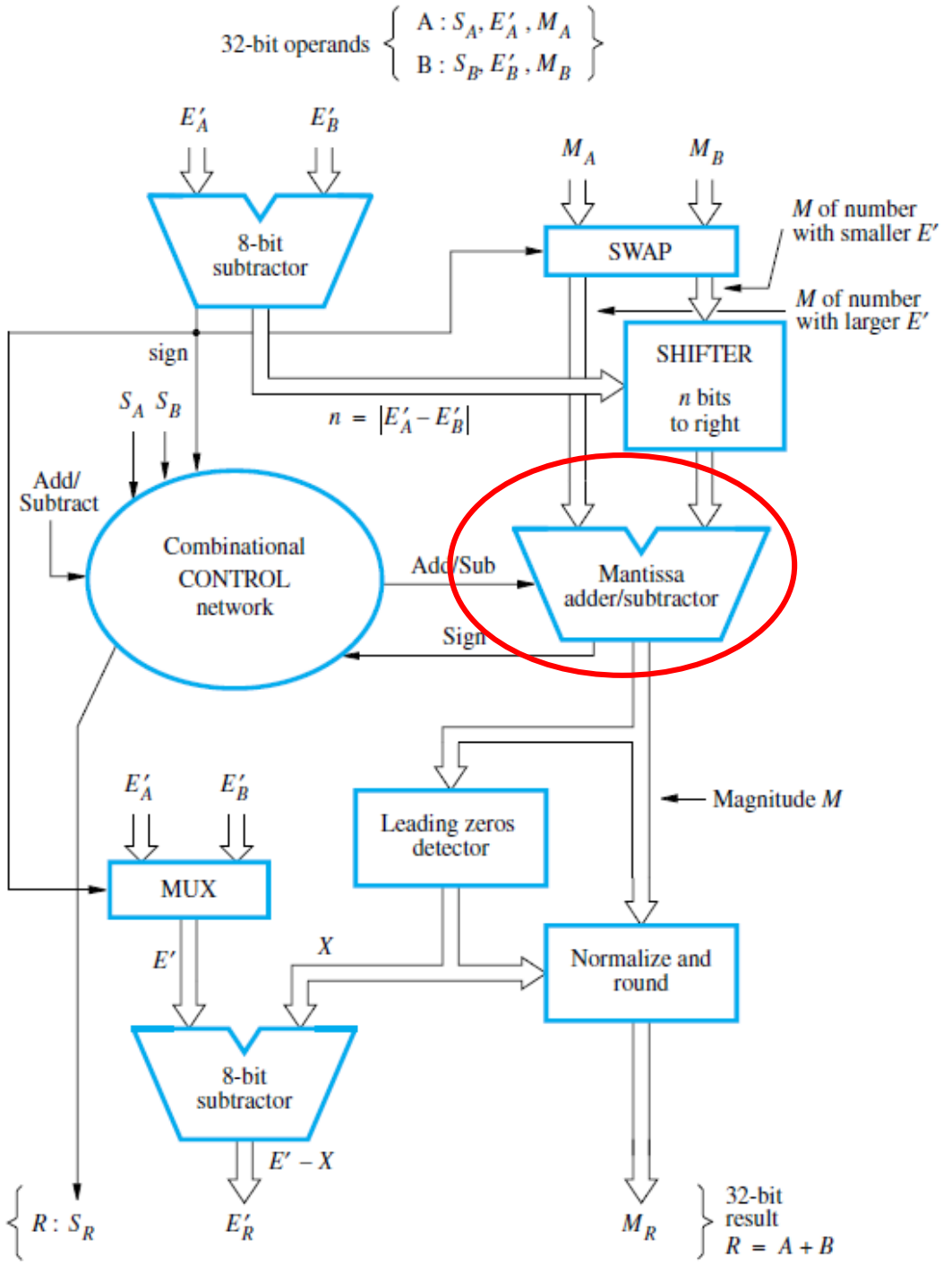
- Step 2 is performed by the two-way multiplexer, MUX.
- The exponent of the result, *E'*, is tentatively determined as
  - *E'A* if *E'A* ≥ *E'B*,
  - or *E'B* if *E'A* < *E'B*,

  based on the sign of the difference resulting from comparing exponents in step 1.



Implementing

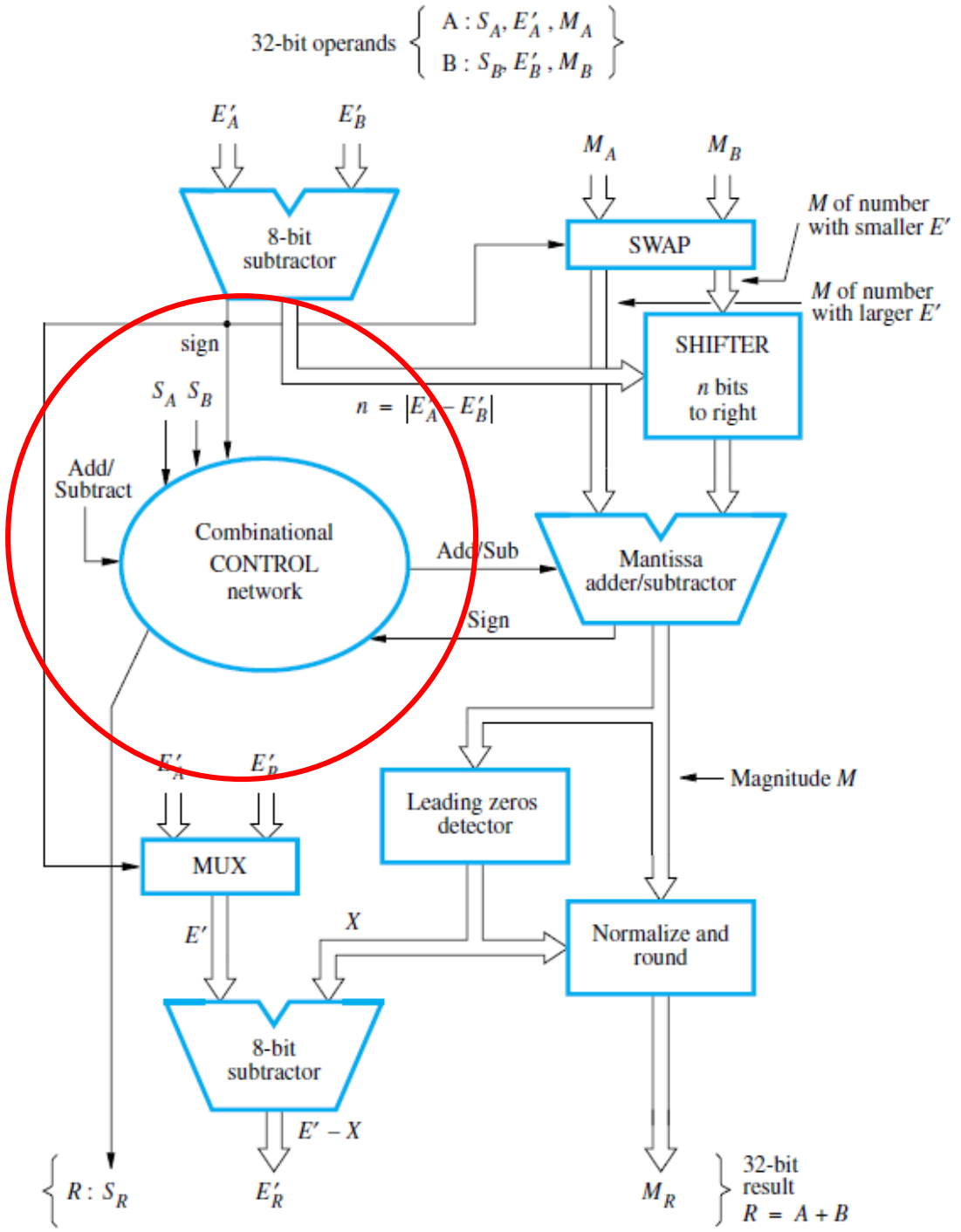
- Step 3 involves the major component, the mantissa adder/subtractor.
- The CONTROL logic determines
  - whether the mantissas are to be added or subtracted.
  - the sign of the result, *SR*.
- Sign for A – B:
- if *A* is negative (*SA* = 1), *B* is positive (*SB* = 0) → then the mantissas are added and the sign of the result is negative (*SR* = 1).
- if *A* and *B* are both positive → then the mantissas are subtracted. The sign of the result, *SR*, depends on the mantissa subtraction
  - if *E'A* > *E'B*, then *M* = *MA* − *(*shifted *MB*) and the resulting number is positive.
  - if *E'B* > *E'A*, then *M* = *MB* − *(*shifted *MA*) and the result is negative.
- When *E'A* = *E'B* and the mantissas are subtracted, the sign of the mantissa adder/subtractor output determines the sign of the result.



Implementing

- Step 3 involves the major component, the mantissa adder/subtractor.
- The CONTROL logic determines
  - whether the mantissas are to be added or subtracted.
  - the sign of the result, *SR*.
- Sign for A – B:
- if *A* is negative (*SA* = 1), *B* is positive (*SB* = 0) → then the mantissas are added and the sign of the result is negative (*SR* = 1).
- if *A* and *B* are both positive → then the mantissas are subtracted. The sign of the result, *SR*, depends on the mantissa subtraction
  - if *E'A* > *E'B*, then *M* = *MA* − *(*shifted *MB*) and the resulting number is positive.
  - if *E'B* > *E'A*, then *M* = *MB* − *(*shifted *MA*) and the result is negative.
- When *E'A* = *E'B* and the mantissas are subtracted, the sign of the mantissa adder/subtractor output determines the sign of the result.

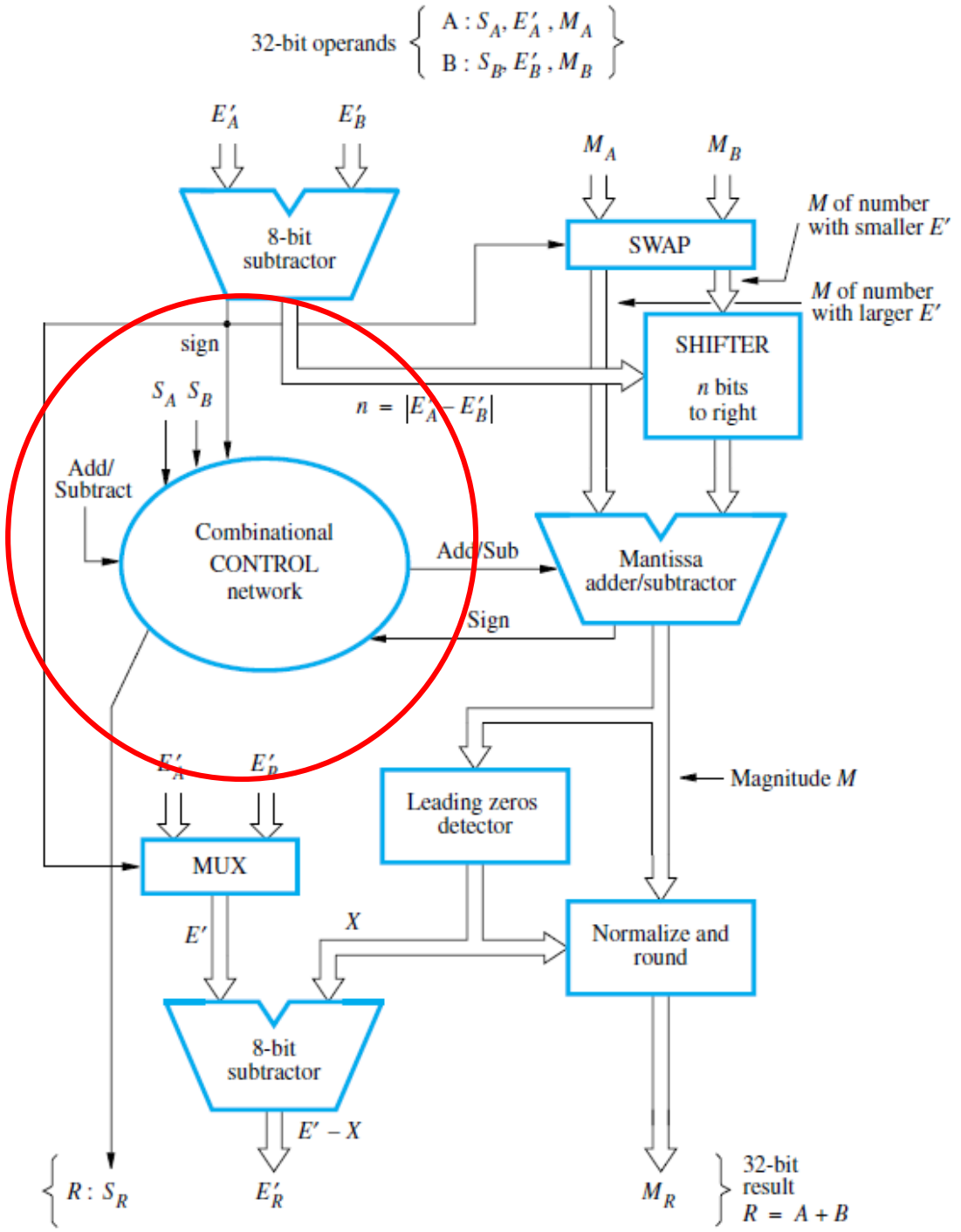
Question: design the control logic



Implementing

- Step 4 of the Add/Subtract rule consists of normalizing the result of step 3 by shifting M to the right or to the left, as appropriate.
- The number of leading zeros in M determines the number of bit shifts, X , to be applied to M.
- The normalized value is rounded to generate the 24-bit mantissa, MR.
- The value X is also subtracted from the tentative result exponent E' to generate the true result exponent, E'R.
- A single right shift might be needed, if

??????



Microprocessors

Tuba Ayhan

MEF University

Floating point co-processor

Cortex A Programmer’s guide

ARM VFP co-processor

- The VFP architecture is an optional co-processor extension to the ARM architecture.
- It provides both single-precision and double-precision floating-point arithmetic, conforming to the IEEE 754 standard.
- VFPv3 is an optional (but rarely omitted) extension to the instruction sets in the ARMv7-A architecture. It can be implemented with either thirty-two, or sixteen double-word registers.

Co-processor

It has

- A structure → basic processor unit
- Registers → control and status registers
- Instruction set
  - Data transfer instructions
  - Arithmetic and logic instructions
- Exchange data with main processor
  - Data
  - Flag information

VFP registers

Floating-Point System ID Register (FPSID)

- This can be read by system software to determine which floating-point features are supported in hardware.

Floating-Point Status and Control register (FPSCR) - User mode code can only access this

- This holds comparison results and flags for exceptions. Control bits select rounding options and enable floating-point exception trapping.

Floating-Point Exception Register (FPEXC)

- The FPEXC register contains bits which allow system software that handles exceptions to determine what has happened.

Media and VFP Feature registers 0 and 1 (MVFR0 and MVFR1)

- These registers allow system software to determine which Advanced SIMD and floating-point features are provided on the processor implementation.

Example code – VFP

```
VCMP d0, d1
VMRS APSR_nzcv, FPSCR
BNE  label
```

- The VCMP instruction performs a comparison the values in VFP registers d0 and d1 and sets FPSCR flags as a result.
- These flags must then be transferred to the integer processor APSR (*Application Program Status Register*), using the VMRS instruction.
- We can then conditionally execute instructions based on this.

Flags – ARM Cortex A9

MEF UNIVERSITY
Your Freedom in Learning

Table 6-1 Conditional execution suffixes

Suffix	Flags	Description
EQ	Z=1	zero (EQual to 0)
NE	Z=0	not zero (Not Equal to 0)
CS/HS	C=1	Carry Set / unsigned Higher or Same
CC/LO	C=0	Carry Clear / unsigned LOwer
MI	N=1	negative (MInus)
PL	N=0	positive or zero (PLus)
VS	V=1	sign overflow (oVerflow Set)
VC	V=0	no sign overflow (oVerflow Clear)
HI	C=1 AND Z=0	unsigned HIgher
LS	C=0 OR Z=1	unsigned Lower or Same
GE	N=V	signed Greater or Equal
LT	N != V	signed Less Than
GT	Z=0 AND N=V	signed Greater Than
LE	Z=1 OR N != V	signed Less or Equal
AL	-	ALways (default)

Flag meanings

- Floating-point values are always signed, so there is no need for unsigned comparisons
- Floating-point comparisons can result in the unordered result (meaning that one or both operands was NaN, or “not a number”).
- IEEE-754 defines four testable relationships between two floating-point values.

Table 18-2 ARM APSR flags

IEEE-754 relationship	ARM APSR flags			
	N	Z	C	V
Equal	0	1	1	0
Less Than (LT)	1	0	0	0
Greater Than (GT)	0	0	1	0
Unordered (At least one argument was NaN)	0	0	1	1

Interp

Code	Meaning (when set by vcmp)	Meaning (when set by cmp)	Flags tested
EQ	Equal to	Equal to	Z =1
NE	Unordered, or not equal to	Not equal to.	Z = 0
CS or HS	Greater than, equal to, or unordered	Greater than or equal to (unsigned).	C = 1
CC or LO	Less than.	Less than (unsigned).	C = 0
MI	Less than	Negative.	N = 1
PL	Greater than, equal to, or unordered	Positive or zero.	N = 0
VS	Unordered. (At least one argument was NaN.)	Signed overflow.	V = 1
VC	Not unordered. (No argument was NaN.)	No signed overflow.	V = 0
HI	Greater than or unordered	Greater than (unsigned).	(C = 1) && (Z = 0)
LS	Less than or equal to	Less than or equal to (unsigned).	(C = 0) || (Z = 1)
GE	Greater than or equal to.	Greater than or equal to (signed).	N==V
LT	Less than or unordered.	Less than (signed).	N!=V
GT	Greater than.	Greater than (signed).	(Z==0) && (N==V)
LE	Less than, equal to or unordered.	Less than or equal to (signed).	(Z==1) || (N!=V)
AL (or omitted)	Always executed.	Always executed.	None tested.

MEF UNIVERSITY
Your Freedom in Learning

Instructions

- VFP instructions are provided which perform
  - arithmetic and data processing,
  - load and stores to memory, and
  - register transfers (between VFP registers and to/from ARM registers).
- These instructions are encoded with ARM coprocessor instructions, but are typically viewed as part of the main instruction set, rather than as coprocessor operations. VFP offers all the common arithmetic operations, format conversions, a few complex arithmetic operations (for example, Multiply accumulate, VMLA, and square root, VSQRT), along with memory access instructions.

Instructions – Guide App. B.7

MEF UNIVERSITY
Your Freedom in Learning

- VABS
- VADD
- VCMP
- VCVT (between single-precision and double-precision)
- VCVT (between floating-point and integer)
- VCVT (between floating-point and fixed-point)
- VCVTB, VCVTT (half-precision extension)
- VDIV
- VFMA, VFNMA, VFMS, VFNMS
- VMOV
- VMOV
- VMUL, VMLA, VMLS, VNMUL, VNMLA, and VNMLS
- VNEG
- VSQRT
- VSUB

VADD

VADD adds the values in the operand registers and places the result in the destination register.

Syntax

```
VADD{cond}.F32 {Sd,} Sn, Sm
VADD{cond}.F64 {Dd,} Dn, Dm
```

where:

Sd, Sn, and Sm are the single-precision registers for the result and operands.

Dd, Dn, and Dm are the double-precision registers for the result and operands.

Enabling VFP

If an ARMv7 processor includes VFP hardware, boot code must explicitly enable its use. Several steps are required to do this:

- Access to CP10 and CP11 must be enabled in the Coprocessor Access Control Register (CP15.CACR)
- If access to VFP is required in the Non-Secure world, access to CP10 and CP11 must be enabled in the Non-Secure Access Control Register (CP15.NSACR)
- The EN bit in the FPEXC register must be set.

Advanced SIMD co-processor: NEON

- NEON is a combined 64- and 128-bit SIMD instruction set that provides standardized acceleration for media and signal processing applications.
  - It has a comprehensive instruction set, separate register files and independent execution hardware.
  - It supports 8-, 16-, 32- and 64-bit integer and single-precision (32-bit) floating-point data and SIMD operations for handling audio and video processing, graphics and gaming processing.
  - MP3 audio decoding on CPUs running at 10 MHz
  - GSM AMR speech codec at 13 MHz.
  - The SIMD supports up to 16 operations at the same time.

Flynn's Taxonomy