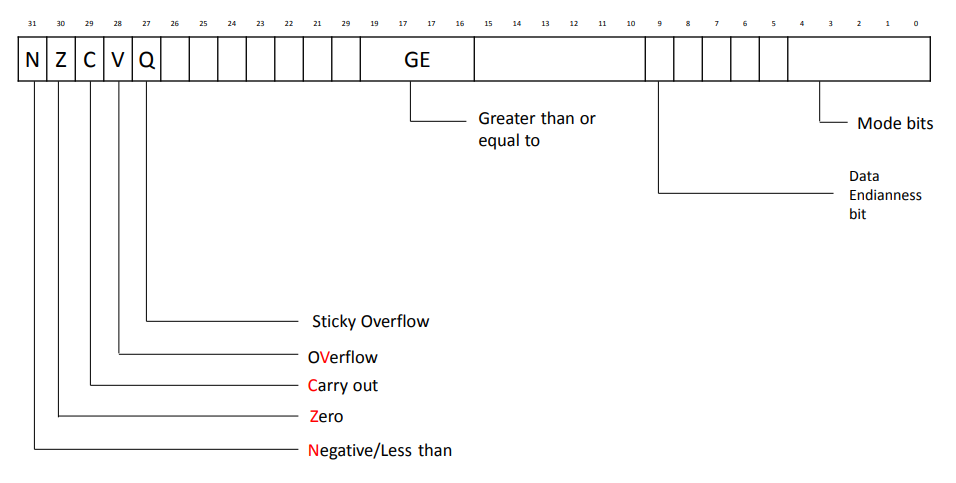
## ARM

Register: Internal CPU device that stores binary data and can be accessed quicker than RAM

ARM has

* + 13 general purpose registers (R0 – R12)
  + 1 stack pointer (R13)
  + 1 link register (R14)
  + 1 program counter (R15)
  + 1 current program status register

Process status register

* Contains: Condition flags that are set by arithmetic and logical instructions, used for conditional execution

ARM Instructions

* Written as opcode followed by 0+ operands
  + Which may be constaints, registers, or memory references
  + Opcode{cond}{flags} Rd, Rn, operand2
    - Cond = optional condition
    - Flag = optional flag
    - Rd = destination register
    - Rn = first source register
    - Operand2 = flexible second operand
* Operand 2
  + Passed through barrel shifter
  + Takes one of four forms:
    - Immediate value (8-bit number rotated right by even number
    - Register
    - Register shifted by 5-bit unsigned integer
    - Register shifted by bottom 8 bits of register
  + Add r0, r1, #3
  + Mov 01, r1, lsl #4
  + Add 45, r3, ror r0

Binary Numbers

Binary Representation and Logic

* Bit = Binary Digit: 0/1, Off / On
* Mathematical Diversion:
* Assume n-state device where cost of device is linear
* N = KN
* Number of devices required for representing an arbitrary number ‘x’ is logn(x)
* Thus, total cost to represent x = kn \* logn(x)
* Find n that minimuzes cost
* N = e = 2.718
* N bits have 2n permutations

Memory unit terminology

* Word
* Unit of memory access and/or size of registers
* 16 / 32 / 64 bits
* Byte
* Unit for character representation
* 8 bits
* Nibble
* Unit for binary-coded decimal digit
* BCD
* 4 bits

Terminology for bases:

* Decimal (Base 10, 0-9)
* Binary (Base 2, 0-1)
* Octal (Base 8, 0-7)
* Hexadecimal (Base 16, 0-F)

Conversions

* Decimal -> Binary
* While(number > 0)

1. Divide number by 2, getting quotient and remainder
2. Remainder is next binary digit
3. Number = quotient

* Binary -> Decimal
* Positional representation

1. 10110111 =
2. 1\*27 + 0\*26 + 1\*25 + 1\*24 + 0\*23 + 1\*22 + 1\*21 + 1\*20 =
3. 128 + 0 + 32 + 16 + 0 + 4 + 2 + 1 =
4. 128 + 32 + 16 + 4 + 2 + 1 =
5. 183 = 10110111

* Binary -> hexadecimal
* Using a table
* 1001  0010  1110  0001  1010
* 9         2        E (14)  1         A (10)
* Binary -> octal is same, just use groups of 3

Signed binary representation

* Sign-magnitude
* Put a 0 for positive, 1 for negative at very left of byte
* Computationally awkward
* Ones-complement
* If negative, flip all values
* Twos-complement
* If negative, flip all values and add 1

Barrel shifter

Functional unit when can be used in number of circumstances

Provides five types of shits and rotates that can be applied to operand2

* Lsl – logical shift left
* 0 enters from right, bits drop off left
* Multiplication by 2n
* Lsr – logical shift right
* 0 enters from left, drops off right end
* Unsigned multiplication by 2n
* Asr – arithmetic shift right
* Sign bit replicated on left, bits drop off right
* Signed division by 2n
* Ror – bitwise rotate right
  + Dropped off bits enter from left

### Field operation and Extraction

Rightmost field extraction with mask

* mov r3, #4 // m = 4
* mov r4, #0 // n = 0
* mov r2, #1
* lsl r2, r2, r3
* sub r2, r2, #1
* lsl r2, r2, r4

Field extraction with mask

* mov r3, #4 // m = 4
* mov r4, #4 // n = 4
* mov r2, #1
* lsl r2, r2, r3
* sub r2, r2, #1
* lsl r2, r2, r4

Field extraction without mask

* mov r2, #8 // starting position of desired field
* mov r3, #4 // width of field to extract
* rsub r2, r2, #32
* rsub r3, r3, #32

# Data in memory

Variable attributes

* Symbolic name
* Data type
* Storage class
  + Allocation region
  + Duration
  + Scope
  + Linkage
* Address in memory
* Alignment
* Byte order
* Initialization (optional)
* Current value

Data sizes:

* Byte - 8 bits = 1 byte -- character
* Halfword – 16 bits = 2 bytes – short
* Word – 32 bits = 4 bytes – integer, 32-bit long
* Doubleword – 64 bits = 8 bytes – double, 64-bit long

Alignment

* Memory access typically on word-by-word basis
  + “aligned”
* Restrictions on load/store instruction prevent multiple-byte units from spanning across two memory words, thus allowing individual instructions to make one and only one access
  + Makes hardware easier to build
* Unaligned access:
  + May be illegal (bus error)
  + May cause OS trap
  + Requires extra shifting network hardware and extra time to perform to access and insertion of bites
* Rules:
  + Bytes start anywhere
  + Halfwards start on halfword boundary
    - Address divisible by 2
  + Words on word boundary, 4
  + Doublewords on doubleword boundary, 8

Condition codes:

* Used for conditional execution
* N = negative
* Z = zero
* C = carry out
* V = overflow

Unsigned binary decimal:

* Does not specify sign, stores larger value
* 0 to 2n – 1

Hexadecimal:

2. range of an unsigned or signed bit field without a calculator.

* Unsigned: [0, 2^n – 1]
* 2s Complement: [-2n-1, 2n-1 -1 ]

3. convert numbers from decimal to hexadecimal or from hexadecimal or binary to decimal without a calculator.

* From decimal to hexadecimal:
  + Convert to binary, split into 4s, convert to hex
  + Or opposite for opposite

4. Be able to add and subtract binary or hexadecimal numbers using two's complement, without a calculator.

* + added with + giving - => signed overflow