Arithmetic ve Logical Operations

- ALU (Arithmetic Logical Unit): CPU nun Aritmetik ve logic islemlerinin yapildigi kismina denir.
  - Temel iki operation
    - Addition (Toplama)
    - Negation (NOT islemi)
- Islemler sayilarin temsil edilme sekline gore degisiklik gosterirler.
- Hangi sayi temsil sekli islemleri daha cok kolaylastiriyorsa o bilgisyar tasariminda kullanilmaktadir.

Overflow

- Bilgisayar mimarisi sabit uzunluktaki veriler uzerinde islem yaparlar.
  - 32-bit / 64-bit islemci
- Overflow
  - Eger islemin sonucu o islem icin ayrilan alana sigmazsa buna overflow denir
  - Overflow detection (tesbiti) onemli
    - Aksi halde yanlis sonuclar kullanilarak islemler yapilir.

Boolean Operations

- Bir boolean variable iki degerden birini alabilir.
  - False (0)
  - True (1)

input	zero	one	invert	same
0	0	1	1	0
1	0	1	0	1

Unary Boolean Operations

Boolean Operations

- Binary boolean operations
  - Iki variable uzerinde islem yapilir

a	b	and	or	nand	nor	xor	xnor
0	0	0	0	1	1	0	1
0	1	0	1	1	0	1	0
1	0	0	1	1	0	1	0
1	1	1	1	0	0	0	1

not	x, y
and	x, y, z
or	x, y, z
nand	x, y, z
nor	x, y, z
xor	x, y, z
xnor	x, y, z

SAL daki Logical Operations

Maskeleme İşlemi

- Boolean islemlerinde birden fazla değişken ayni bellek gozune yazildiginda kullanilirlar.
- Değişkenlerin bellek gozunden cikarilmasi islemine masking denir.

```
cell:      .word   0x43686172
mask1:     .word   0xff000000
mask2:     .word   0x00ff0000
mask3:     .word   0x0000ff00
mask4:     .word   0x000000ff

and    result, cell, mask1
```

Shift Operations (Oteleme Islemleri)

- Logical Shift
  - Logical right shift:
    - Bitler 1 pozisyon saga otelenir
    - En sagdaki bit (LSB) atilir
    - En soldaki (MSB) bite 0 atanir.

0 1 1 0 0 1 1 1

0 0 1 1 0 0 1 1

0 0 0 1 1 0 0 1

Shift Operations

- Logical Left Shift
  - Bitler bir pozisyon sola otelenirler
  - En soldaki bit (MSB) atilir
  - En sagdaki bit (LSB) 0 olur

0 1 1 0 0 1 1 1

1 1 0 0 1 1 1 0

1 0 0 1 1 1 0 0

Rotate

- Rotate Right
  - Bitler saga dogru bir pozisyon kaydirilir
  - LSB, MSB olur

0 1 1 0 0 1 1 0

0 0 1 1 0 0 1 1

1 0 0 1 1 0 0 1

Rotate

- Rotate Left
  - Bitler bir pozisyon sola otelenir
  - MSB, LSB olur

0 1 1 0 0 1 1 1

1 1 0 0 1 1 1 0

1 0 0 1 1 1 0 1

Arithmetic Shift

- Arithmetic left shift
- Logical left shift le ayni
  - Bitler bir pozisyon sola otelenirler
  - En soldaki bit (MSB) atilir
  - En sagdaki bit (LSB) 0 olur

0	1	1	0	0	1	1	1
1	1	0	0	1	1	1	0
1	0	0	1	1	1	0	0

Arithmetic Left Shift

- Sayinin arithmetic 1 bit otelenmesi o sayinin 2 ile carpimi anlamina gelir.

1	0	0	1	1	1	0	1	-99
0	0	1	1	1	0	1	0	58
0	1	1	1	0	1	0	0	116

Arithmetic Right Shift

- Logical right shift gibi. Tek farki sign bit extended (MSF sayinin sign bitiyle ayni)
- Bir sayinin 1 bit arithmetic saga otelenmesi o sayinin 2 ile bolunmesi anlamina gelir.

0	1	1	0	0	1	1	1	103
0	0	1	1	0	0	1	1	51
0	0	0	1	1	0	0	1	25

Addition (Toplama) / Subtraction (Cikarma)

Unsigned Integers:

$$
\begin{array}{rl}
 & 1 \\
 & 010010\ (18_{10}) \\
+ & 011000\ (24_{10}) \\
\hline
 & 101010\ (42_{10})
\end{array}
$$

c_i	x_i	y_i	z_i	c_{i+1}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Unsigned integers

a	b	a-b
0	0	0
0	1	borrow
1	0	1
1	1	0
10	1	1

$$\begin{array}{r} 0\,1\,0\,0\,0 \\ -\;0\,0\,1\,1\,0 \\ \hline 0\,0\,0\,1\,0 \end{array}$$

Sign magnitude

```
  1 00100          0 00110               10010
+ 1 00101        + 1 10010      ==>    - 00110
---------        ---------             -------
  1 01001                              1 01100
```

Two's Complement Addition/Subtraction

- Isaret bitlerine bakilmaksizin addition icin ayni algoritma uygulanir.
- Substraction additive inverse alinip addition algoritmasi kullanilmak suretiyle gerceklestirilir.
- Unsigned number larin toplami icin kullanilan ayni devre two's complementi icin toplama ve cikarmada kullanilabilir

$$
\begin{array}{rr}
 & 0010\ 0010 \\
+ & 1001\ 1000 \\
\hline
 & 1011\ 1010
\end{array}
\qquad
\begin{array}{r}
34_{10} \\
-104_{10} \\
-70
\end{array}
$$

Overflow

carry	$\underline{1}$ $\underline{1}$	
	1111 1000	-8_{10}
+	1111 1000	-8_{10}
	1111 0000	-16_{10}

dogru sonuc
no overflow

carry	$\underline{0}$ $\underline{1}$	
	0111 1110	126_{10}
+	0110 0000	96_{10}
	1101 1110	-34_{10}

yanlis sonuc
overflow

Multiplication

multiplicand → 1 1 0 1 = -3_{10}

multiplier → 0 1 1 0 = 6_{10}

×

```
           1 1 1 1 1 1 0 1 = -3₁₀
         × 0 0 0 0 0 1 1 0 =  6₁₀
         -----------------
           0 0 0 0 0 0 0 0
         1 1 1 1 1 1 0 1
       1 1 1 1 1 1 0 1
     0 0 0 0 0 0 0 0
   0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
+ 0 0 0 0 0 0 0 0
---------------------------------
             1 1 1 0 1 1 1 0 = -18
```

$$
\begin{array}{r}
1\;1\;1\;1\;1\;0\;0\;0 = -8_{10} \\
\times\; 1\;1\;1\;1\;1\;0\;0\;0 = -8_{10} \\
\hline
0\;0\;0\;0\;0\;0\;0\;0 \phantom{{}= -8_{10}} \\
0\;0\;0\;0\;0\;0\;0\;0 \phantom{{}= -8_{10}} \\
0\;0\;0\;0\;0\;0\;0\;0 \phantom{{}= -8_{10}} \\
1\;1\;1\;1\;1\;0\;0\;0 \phantom{{}= -8_{10}} \\
1\;1\;1\;1\;1\;0\;0\;0 \phantom{{}= -8_{10}} \\
1\;1\;1\;1\;1\;0\;0\;0 \phantom{{}= -8_{10}} \\
1\;1\;1\;1\;1\;0\;0\;0 \phantom{{}= -8_{10}} \\
+\;1\;1\;1\;1\;1\;0\;0\;0 \phantom{{}= -8_{10}} \\
\hline
0\;1\;0\;0\;0\;0\;0\;0 = 64
\end{array}
$$

X ve Y nin carpim programi

```
                .data
        X:      .word                   ms_sum:     .word   0
        Y:      .word                   bitsum:     .word
        ls_sum  .word   0               mask:       .word   0x1
        test:   .word

        .text
__start:and     test, X, mask           # strip off appropriate multiplier bit
        beqz    test, shift             # skip addition if multiplier is zero

        add     ms_sum, ms_sum, Y       # add partial sum
shift:  and     bitsum, ms_sum, 1       # determine lsb of ms_sum
        or      ls_sum, ls_sum, bitsum  # place lsb of ms_sum in lsb of ls_sum
        ror     ls_sum, ls_sum, 1       # shift ls_sum, moving new bit into msb
        sra     ms_sum, ms_sum, 1       # shift ms_sum, maintaining sign
        sll     mask, mask, 1           # update index
        bnez    mask, __start           # branch if not last iteration
        done
```

Floating Point (FP) Arithmetic

- Bilgisayar designinda floating point sayilarinmin gosterimi onemli bir yer tutar.
- Duyarliligi yuksek olmasi istenilen islemlerde floating point islemlerinin hizli olmasi istenir
- Floating Point Operations Per Second (FLOPS)
  - Scientific bilgisayarlarin performans karsilastirimlarinda kullanilir.
- Floating Point Operations Integer Operationlarindan daha yavastir

Hardware versus Software Calculatiuons

- Hesaplamalar nasil yapilmali
  - Hardware Implementation: Devreler (circuits) FP islemlerini yapar.
    - Hizli
    - Pahali
  - Software
    - Ucuz (devre acisindan)
    - Yavas (10 un bir kac kuvveti kadar yavas)

2.25 + 134.0625

0	1000	0000	(1) 001	0000	0000	0000	0000	0000
0	1000	0110	(1) 000	0110	0001	0000	0000	0000

	0	1000	0110	(0) 000	0010	0100	0000	0000	0000
+	0	1000	0110	(1) 000	0110	0001	0000	0000	0000
	0	1000	0110	(1) 000	1000	0101	0000	0000	0000

255.0625 + 134.0625

```
0  1000  0110  (1) 111  1111  0001  0000  0000  0000
0  1000  0110  (1) 000  0110  0001  0000  0000  0000
```

```
   0  1000  0110  (1) 111  1111  0001  0000  0000  0000
   0  1000  0110  (1) 000  0110  0001  0000  0000  0000
+
--------------------------------------------------------
   0  1000  0110(11)  000  0101  0010  0000  0000  0000
```

```
0  1000  0111  (1) 100  0010  1001  0000  0000  0000
```

2.25 + -134.0625

```
0  1000  0110  (0)000  0010  0100  0000  0000  0000
1  1000  0110  (1)000  0110  0001  0000  0000  0000
```

```
     0000  0000   0000  0010  0100  0000  0000  0000
     1111  1111   0111  1001  1111  0000  0000  0000
+
---------------------------------------------------
     1111  1111   0111  1100  0011  0000  0000  0000
```

```
(1)000  0011  1101  0000  0000  0000
```

```
1  1000  0110  (1)000  0011  1101  0000  0000  0000
```

```
                .data
FloatX:         .float  134.0625
FloatY          .float  2.25
Float_X_plus_Y: .float
X:              .word
Y:              .word
X_F:            .word
X_E:            .word
Y_F:            .word
Y_E:            .word
X_time_Y:       .word
X_plus_Y:       .word
X_plus_Y_F:     .word
X_plus_Y_E:     .word
X_plus_Y_S:     .word
small_F:        .word
diff:           .word
F_mask:         .word   0x007fffff
E_mask:         .word   0x7f800000
S_MASK:         .word   0x80000000
Hidden_one:     .word   0x00800000
zero:           .word   0
max_F:          .word   0x01000000
```

```
            .text
            # Extract E (exponent) and F (significand).
__start:    move    X, FloatX
            and     X_F, X, F_Mask              # get X_F
            or      X_F, X_F, Hidden_one        # add hidden bit
            bgtz    X, DoX_E                    # skip if positive
            sub     X_F, zero, X_F              # convert to 2's comp.
DoX_E:      and     X_E, X, E_mask              # get X_E
            srl     X_E, X_E, 23                # align
            sub     X_E, X_E, 127               # convert to 2's comp.

            move    Y, FloatY
            and     Y_F, Y, F_Mask              # get Y_F
            or      Y_F, Y_F, Hidden_one        # add hidden bit
            bgtz    Y, DoY_E                    # skip if positive
            sub     Y_F, zero, Y_F              # convert to 2's comp.
DoY_E:      and     Y_E, Y, E_mask              # get Y_E
            srl     Y_E, Y_E, 23                # align
            sub     Y_E, Y_E, 127               # convert to 2's comp.
```

```
                # Determine which input is smaller

                sub         diff, Y_E, X_E
                bltz        diff, X_bigger
                move        X_plus_Y_E, Y_E
                move        X_plus_Y_F, Y_F
                move        small_F, X_F
                b           LittleF
X_bigger:       move        X_plus_Y_E, X_E
                move        X_plus_Y_F, X_F
                move        small_F, Y_F
                sub         diff, zero, diff
LittleF:        sra         small_F, small_F, diff      # denormalize little F
```

```
            add     X_plus_Y_F, small_F, X_plus_Y_F     # add Fs
            and     X_plus_Y_S, X_plus_Y_F, S_mask
            beqz    X_plus_Y_F, Zero
            bgez    X_plus_Y_F, L1                      # skip if positive
            sub     X_plus_Y_F, zero, X_plus_Y_F        # convert to sign/mag
L1:         move    X_plus_Y_E, X_plus_Y_E
            blt     X_plus_Y_F, max_F, NotTooBig        # skip if no overflow
            srl     X_plus_Y_F, X_plus_Y_F, 1           # divide F by 2
            add     X_plus_Y_E, X_plus_Y_E, 1           # adjust E
            b       normalized
Zero:       move    Float_X_plus_Y, 0
            b       Finished
TooSmall:   sll     X_plus_Y_F, X_plus_Y_F, 1           # multiply F by 2
            sub     X_plus_Y_E, X_plus_Y_E, 1           # adjust E
NotTooBig:  blt     X_plus_Y_F, Hidden_one, TooSmall    # check if still too big
normalized: sub     X_plus_Y_F, X_plus_Y_F, Hidden_one # delete hidden one
            add     X_plus_Y_E, X_plus_Y_E, 127         # convert to bias-127
            sll     X_plus_Y_E, X_plus_Y_E, 23          # align properly
            or      X_plus_Y, X_plus_Y_E, X_plus_Y_F    # merge E, F
            or      X_plus_Y, X_plus_Y, X_plus_Y_S      # merge S
            move    Float_X_plus_Y, X_plus_Y            # move to floating point
Finished:   done
```

Multiplication

- Floating Addition dan daha basit
- 4 adim
  - Mantissa lar uzerinde unsigned multiplication yap
  - Exponents lere ekle
  - Sonucu normalize hale getir
  - Sonucun isaret bitini belirle

18.0 * 9.5

0	1000	0011	(1) 001	0000	0000	0000	0000	0000
0	1000	0010	(1) 001	1000	0000	0000	0000	0000

```
   1000  0011   4
 + 1000  0010   3
 ----------------
   1000  0110   7
```

```
                     1001  0000
                   × 1001  1000
                   ------------
                     0000  0000
                 0   0000  000
                00   0000  00
               100   1000  0
              1001   0000
          0   0000   000
         00   0000   00
      + 100   1000   0
      -------------------------
        101   0101   1000  0000
```

0	1000	0110	(1) 010	1011	0000	0000	0000	0000

Division

- Multiplication a benzer
- 4 adim
  - Mantissalar uzerinde unsigned division yap
  - Divisorun exponentini dividend in exponentinden cikar
  - Sonucu normalize yap
  - Sonucun isaret bitini belirle.

Overflow ve Underflow

- Overflow
  - Normalized sonucun exponenti (biased-127) kendine ayrilan yere sigmadiginda olusur
- Underflow
  - Sonucun temsil edilemeyecek kadar 0 ya yakin olmasiyle olusur