Exception Handling

- The 8087 detects six different types of exception conditions that occur during instruction execution. These will cause an interrupt if unmasked and interrupts are enabled.

1) INVALID OPERATION
2) OVERFLOW
3) ZERO DIVISOR
4) UNDERFLOW
5) DENORMALIZED OPERAND
6) INEXACT RESULT

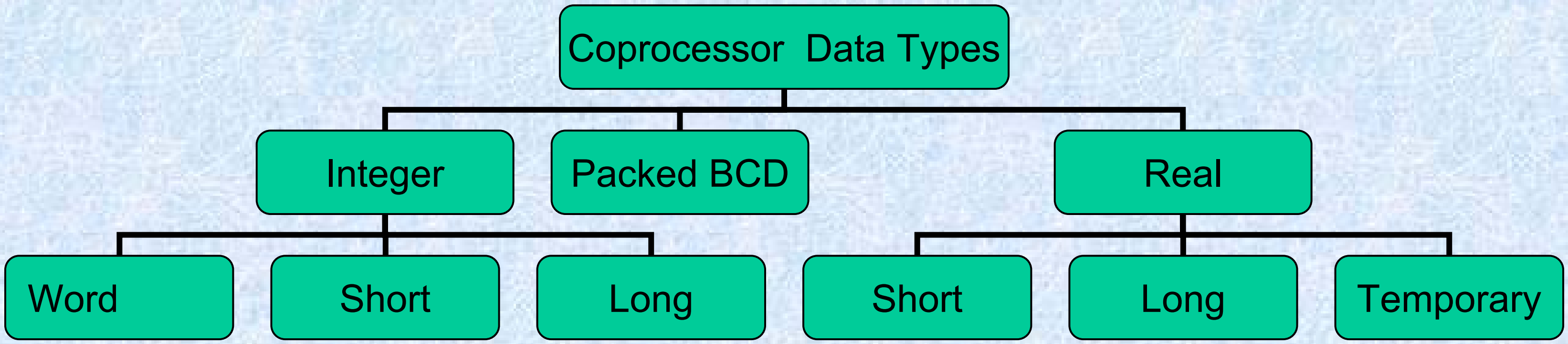
Data Types

- Internally, all data operands are converted to the 80-bit temporary real format.

We have 3 types.

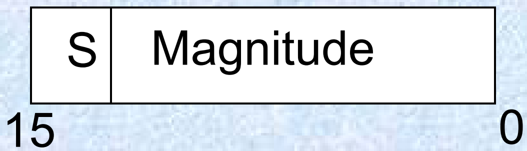
- Integer data type
- Packed BCD data type
- Real data type

Coprocessor data types



Integer Data Type

- Word integer 2 bytes

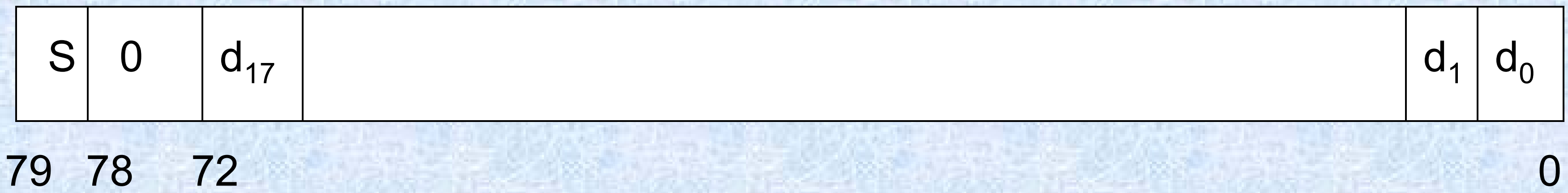


- Short integer 4 bytes

- Long integer 8 bytes

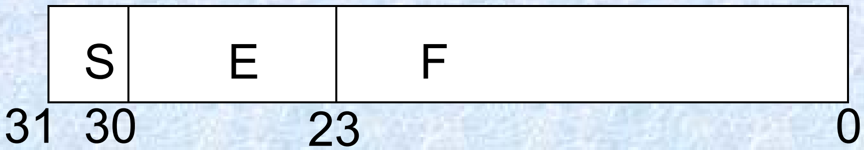
Packed BCD

- Packed BCD 10 bytes



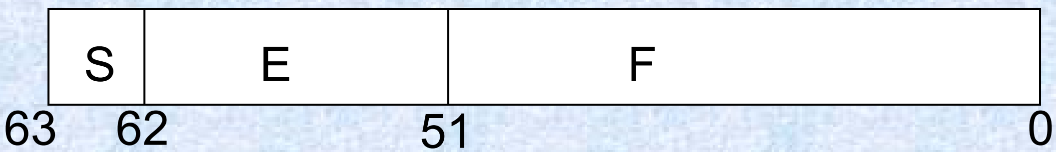
Real data type

- Short real 4 bytes 178.625 decimal



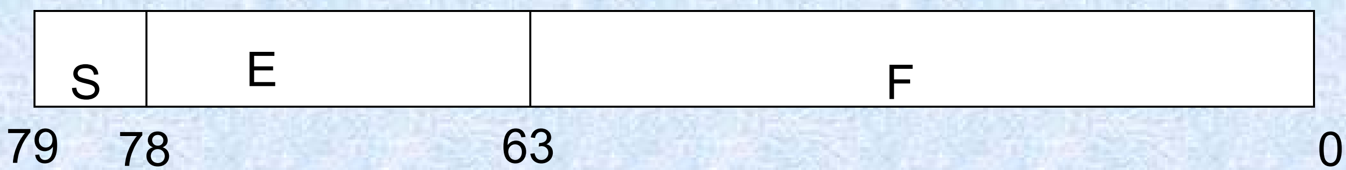
= 4332A000h

- Long real 8 bytes



= 4066540000000000h

- Temporary real 10 bytes



= 4006B2A0000000000000h

Example

- Converting a decimal number into a Floating-point number.

  1) Converting the decimal number into binary form.

  2) Normalize the binary number

  3) Calculate the biased exponent.

  4) Store the number in the floating-point format.

Example

Step	Result
1	100.25
2	1100100.01 = 1.10010001 * 2^6
3	110+01111111=10000101
4	Sign = 0 Exponent =10000101 Significand = 10010001000000000000000

- In step 3 the biased exponent is the exponent a 2^6 or 110,plus a bias of 01111111(7FH) ,single precision no use 7F and double precision no use 3FFFH.
- IN step 4 the information found in prior step is combined to form the floating point no.

Instruction Set

- The 8087 instruction mnemonics begins with the letter F which stands for Floating point and distinguishes from 8086.
- These are grouped into Four functional groups.
- The 8087 detects an error condition usually called an exception when it executing an instruction it will set the bit in its Status register.

Types

I. DATA TRANSFER INSTRUCTIONS.
II. ARITHMETIC INSTRUCTIONS.
III. COMPARE INSTRUCTIONS.
IV. TRANSCENDENTAL INSTRUCTIONS.
(Trigonometric and Exponential)

I. Data Transfers Instructions

- **REAL TRANSFER**

FLD	Load real
FST	Store real
FSTP	Store real and pop
FXCH	Exchange registers

- **INTEGER TRANSFER**

FILD	Load integer
FIST	Store integer
FISTP	Store integer and pop

I. Data Transfers Instructions (contd..)

- **PACKED DECIMAL TRANSFER(BCD)**

FBLD	Load BCD
FBSTP	Store BCD and pop

Example

- **FLD Source-** Decrements the stack pointer by one and copies a real number from a stack element or memory location to the new ST.

• FLD	ST(3)	;Copies ST(3) to ST.
• FLD	LONG_REAL[BX]	;Number from memory ;copied to ST.

- **FLD Destination-** Copies ST to a specified stack position or to a specified memory location .

• FST	ST(2)	;Copies ST to ST(2),and ;increment stack pointer.
• FST	SHORT_REAL[BX]	;Copy ST to a memory at a ;SHORT_REAL[BX]

Example (contd..)

- **FXCH Destination** – Exchange the contents of ST with the contents of a specified stack element.
- FXCH ST(5) ;Swap ST and ST(5)
- **FILD Source** – Integer load. Convert integer number from memory to temporary-real format and push on 8087 stack.
- FILD DWORD PTR[BX] ;Short integer from memory at ; [BX].
- **FIST Destination-** Integer store. Convert number from ST to integer and copy to memory.
- FIST LONG_INT ;ST to memory locations named ;LONG_INT.

Example (contd..)

- **FISTP Destination-**Integer store and pop. Identical to FIST except that stack pointer is incremented after copy.
- **FBLD Source-** Convert BCD number from memory to temporary- real format and push on top of 8087 stack.

II. Arithmetic Instructions.

- Four basic arithmetic functions:
  Addition, Subtraction, Multiplication, and Division.

- **Addition**

FADD	Add real
FADDP	Add real and pop
FIADD	Add integer

II. Arithmetic Instructions (contd..)

Subtraction

FSUB	Subtract real
FSUBP	Subtract real and pop
FISUB	Subtract integer
FSUBR	Subtract real reversed
FSUBRP	Subtract real and pop
FISUBR	Subtract integer reversed

II. Arithmetic Instructions (contd..)

Multiplication

FMUL	Multiply real
FMULP	Multiply real and pop
FIMUL	Multiply integer

II. Arithmetic Instructions (contd..)

Division

FDIV	Division real
FDIVP	Division real and pop
FIDIV	Division integer
FDIVR	Division real reversed
FDIVRP	Division real reversed and pop
FIDIVR	Division integer reversed

II. Arithmetic Instructions (contd..)

- **Advanced**

FABS	Absolute value
FCHS	Change sign
FPREM	Partial remainder
FPRNDINT	Round to integer
FSCALE	Scale
FSQRT	Square root
FXTRACT	Extract exponent and mantissa.

Example

- **FADD** – Add real from specified source to specified destination Source can be a stack or memory location. Destination must be a stack element. If no source or destination is specified, then ST is added to ST(1) and stack pointer is incremented so that the result of addition is at ST.

```
• FADD    ST(3), ST     ;Add ST to ST(3), result in ST(3)
• FADD    ST,ST(4)      ;Add ST(4) to ST, result in ST.
• FADD                  ;ST + ST(1), pop stack result at ST
• FADDP   ST(1)         ;Add ST(1) to ST. Increment stack
                        ;pointer so ST(1) become ST.
• FIADD   Car_Sold      ;Integer number from memory + ST
```

Example (contd..)

- **FSUB** - Subtract the real number at the specified source from the real number at the specified destination and put the result in the specified destination.
  - FSUB ST(2), ST ;ST(2)=ST(2) – ST.
  - FSUB Rate ;ST=ST – real no from memory.
  - FSUB ;ST=(ST(1) – ST)
- **FSUBP** - Subtract ST from specified stack element and put result in specified stack element .Then increment the pointer by one.
  - FSUBP ST(1) ;ST(1)-ST. ST(1) becomes new ST
- **FISUB** – Integer from memory subtracted from ST, result in ST.
  - FISUB Cars_Sold ;ST becomes ST – integer from ;memory

III. Compare Instructions

Comparison

FCOM	Compare real
FCOMP	Compare real and pop
FCOMPP	Compare real and pop twice
FICOM	Compare integer
FICOMP	Compare integer and pop
FTST	Test ST against +0.0
FXAM	Examine ST

IV. Transcendental Instruction

Transcendental

FPTAN	Partial tangent
FPATAN	Partial arctangent
F2XM1	$2^x - 1$
FYL2X	$Y \log_2 X$
FYL2XP1	$Y \log_2 (X+1)$

Example

- **FPTAN –** Compute the values for a ratio of Y/X for an angle in ST. The angle must be in radians, and the angle must be in the range of $0 < angle < \pi/4$.
- **F2XM1 –** Compute $Y=2^x-1$ for an X value in ST. The result Y replaces X in ST. X must be in the range $0 \leq X \leq 0.5$.
- **FYL2X - ** Calculate $Y(LOG_2X)$.X must be in the range of $0 < X < \infty$ any Y must be in the range $-\infty < Y < +\infty$.
- **FYL2XP1 –** Compute the function $Y(LOG_2(X+1))$.This instruction is almost identical to FYL2X except that it gives more accurate results when compute log of a number very close to one.

Constant Instructions

- **Load Constant Instruction**

FLDZ	Load +0.0
FLDI	Load+1.0
FLDPI	Load π
FLDL2T	Load $\log_2 10$
FLDL2E	Load $\log_2 e$
FLDLG2	Load $\log_{10} 2$
FLDLN2	Load $\log_e 2$

Algorithm

To calculate x to the power of y

- Load base, power.
- Compute (y)*($\log_2 x$)
- Separate integer(i) ,fraction(f) of a real number
- Divide fraction (f) by 2
- Compute $(2^{f/2}) * (2^{f/2})$
- $x^y = (2^x) * (2^y)$

Program

Program to calculate x to the power of y

```
.MODEL SMALL
    .DATA
    x           Dq      4.567   ;Base
    y           Dq      2.759   ;Power
    temp        DD
    temp1       DD
    temp2       DD              ;final  real result
    tempint     DD
    tempint1    DD              ;final integer result
    two         DW
    diff        DD
    trunc_cw    DW      0fffh
```

Program (contd..)

```
                .STACK 100h
                .CODE
start:          mov ax,@DATA        ;init data segment
                mov ds,ax

load:           fld y               ;load the power
                fld x               ;load the base
comput:         fyl2x               ;compute (y * log2(x))
                fst temp            ;save the temp result
```

Program (contd..)

```
trunc:      fldcw  trunc_cw        ;set truncation command
            frndint
            fld  temp              ;load real number of
fyl2x
            fist  tempint          ;save integer after
                                   ;truncation
            fld  temp        ;load the real number
getfrac:    fisub  tempint   ;subtract the integer
            fst   diff       ;store the fraction
```

Program (contd..)

```
fracby2:      fidiv two     ;divide the fraction by 2
twopwrx:      f2xm1         ;calculate the 2 to the
                            ;power fraction
              fst temp1     ;minus 1 and save the
   result
              fld1          ;load1
              fadd          ;add 1 to the previous
   result
              fst temp1     ;save the result
```

Program (contd..)

```
sqfrac:    fmul st(0),st(0)   ;square the result as fraction
           fst temp1          ;was halved and save the
                              ;result
           fild tempint       ;save the integer portion
           fxch               ;interchange the integer
                              ;and power of fraction.
```

Program (contd..)

```
scale:    fscale            ;scale the result in real  and
                            ;integer
          fst temp2         ;in st(1) and store
          fist tempint1     ;save the final result in real and
                            ;integer
 over:    mov ax,4c00h      ;exit to dos
          int 21h
          end start
```