Introduction to 8086 Assembly

Lecture 19

Introduction to Floating Point

How to represent rational/real numbers

- Decimal
  - $78.173 = 7 * 10^{1} + 8 * 10^{0} + 1 * 10^{-1} + 7 * 10^{-2} + 3 * 10^{-3}$
- Binary
  - $1001.1011 = ?$

How to represent rational/real numbers

- Decimal
  - $78.173 = 7 * 10^{1} + 8 * 10^{0} + 1 * 10^{-1} + 7 * 10^{-2} + 3 * 10^{-3}$
- Binary
  - $1001.1011 = 1 * 2^{3} + 0 * 2^{2} + 0 * 2^{1} + 1 * 2^{0} + 1*2^{-1} + 0 * 2^{-2} + 1 * 2^{-3} + 1 * 2^{-4}$

Convert decimal to binary

float1.c

```c
#include <stdio.h>


int main() {

  double x = 12.625;
  double y = 12.35;

  printf("x= %f\n", x);
  printf("y= %f\n", y);


  return 0;
}
```

Convert decimal to binary

float1.c

```c
#include <stdio.h>


int main() {

  double x = 12.625;
  double y = 12.35;

  printf("x= %f\n", x);
  printf("y= %f\n", y);


  return 0;
}
```

```
CS@kntu:lecture19$ gcc float1.c && ./a.out
x= 12.625000
y= 12.350000
```

Convert decimal to binary

float2.c

```c
#include <stdio.h>


int main() {

  double x = 12.625;
  double y = 12.35;

  printf("x= %.16f\n", x);
  printf("y= %.16f\n", y);


  return 0;
}
```

Convert decimal to binary

float2.c

```c
#include <stdio.h>


int main() {

  double x = 12.625;
  double y = 12.35;

  printf("x= %.16f\n", x);
  printf("y= %.16f\n", y);


  return 0;
}
```

```
CS@kntu:lecture19$ gcc float2.c && ./a.out
x= 12.6250000000000000
y= 12.3499999999999996
```

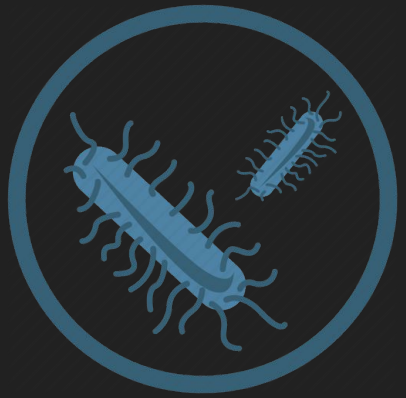
Fixed point representation

1231.2784

Fixed point representation

K. N. Toosi
University of Technology

https://goo.gl/EGnmXc



0.0000023718 m

https://goo.gl/yjPBnm

53.2843453 m

https://goo.gl/NSBgxj

12742345.23 m

Floating point representation

K. N. Toosi
University of Technology

https://goo.gl/EGnmXc

0.0000023718 m

$2.3718 * 10^{-6}$

https://goo.gl/yjPBnm

53.2843453 m

$5.3284 * 10^{1}$

https://goo.gl/NSBgxj

12742345.23 m

$1.2742 * 10^{7}$

IEEE 754 standard floating point representation

- single precision (32 bits)
- double precision (64 bits)

look at http://cs.boisestate.edu/~alark/cs354/lectures/ieee754.pdf

Floating point arithmetic

- add
- subtract
- multiply
- divide

floating point range

32 bit int: up to $2.1*10^9$

32 bit float: up to $+3.4*10^{38}$

?

Example

float3.c

```c
#include <stdio.h>

int main() {

  float x = 1e12;
  float y = 1e-12;
  float z = x+y;

  printf("x= %e\n", x);
  printf("y= %e\n", y);
  printf("z= %e\n", z);
  printf("%d\n", x==z);

  return 0;
}
```

Example

float3.c

```c
#include <stdio.h>

int main() {

  float x = 1e12;
  float y = 1e-12;
  float z = x+y;

  printf("x= %e\n", x);
  printf("y= %e\n", y);
  printf("z= %e\n", z);
  printf("%d\n", x==z);

  return 0;
}
```

```
b.nasihatkon@kntu:lecture19$ gcc float3.c && ./a.out
x= 1.000000e+12
y= 1.000000e-12
z= 1.000000e+12
1
```

Example

float4.c

```c
#include <stdio.h>

int main() {

  int a = 88888888;
  int b = 88888889;

  float x = a;
  float y = b;

  printf("x=%.10f\n", x);
  printf("y=%.10f\n", y);
  printf("%d\n", x==y);

  return 0;
}
```

Example

float4.c

```c
#include <stdio.h>

int main() {

  int a = 88888888;
  int b = 88888889;

  float x = a;
  float y = b;

  printf("x=%.10f\n", x);
  printf("y=%.10f\n", y);
  printf("%d\n", x==y);

  return 0;
}
```

```
b.nasihatkon@kntu:lecture19$ gcc float4.c && ./a.out
x=88888888.0000000000
y=88888888.0000000000
1
```

Example

float5.c

```c
#include <stdio.h>

int main() {

  float x = 88888888;
  printf("x=%f\n", x);

  for (int i = 0; i < 100000; i++)
    x++;

  printf("x=%f\n", x);

  return 0;
}
```

Example

float5.c

```c
#include <stdio.h>

int main() {

  float x = 88888888;
  printf("x=%f\n", x);

  for (int i = 0; i < 100000; i++)
    x++;

  printf("x=%f\n", x);

  return 0;
}
```

```
b.nasihatkon@kntu:lecture19$ gcc float5.c && ./a.out
x=88888888.000000
x=88888888.000000
```

Example: Newton's method

test_root1.c

```c
double h(double x) {
  return x*x*x*x*x - 1.25;

}

int main() {
  double x = newton(h, 2);

  if (fabs(h(x)) == 0)
    printf("root= %e\n", x);
  else
    printf("root not found!\n");

  return 0;
}
```

$$x_{t+1} = x_t - f(x_t) / f'(x_t)$$

test_root1.c

```c
double newton(double (*f)(double x), double x0) {
  double x = x0;
  const double delta = 1e-7;

  while (f(x) != 0 ) {
    double df_dx = (f(x+delta)-f(x))/delta;

    x = x - f(x) / df_dx;

    printf("%.10f, %e\n", x, f(x));
  }
  return x;
}
```

Example: Newton's method

test_root1.c

```c
double newton(double (*f)(double x), double x0) {
  double x = x0;
  const double delta = 1e-7;

  while ( f(x) != 0 ) {
    double df_dx = (f(x+delta)-f(x))/delta;

    x = x - f(x) / df_dx;

    printf("%.10f, %e\n", x, f(x));
  }
  return x;
}
```

test_root2.c

```c
double newton(double (*f)(double x), double x0) {
  double x = x0;
  const double delta = 1e-7;

  while ( fabs(f(x)) >= 1e-10 ) {
    double df_dx = (f(x+delta)-f(x))/delta;

    x = x - f(x) / df_dx;

    printf("%.10f, %e\n", x, f(x));
  }
  return x;
}
```

Example: Newton's method

test_root1.c

```
int main() {
  double x = newton(h, 2);

  if (fabs(h(x)) == 0)
    printf("root= %e\n", x);
  else
    printf("root not found!\n");

  return 0;
}
```

test_root2.c

```
int main() {
  double x = newton(h, 2);

  if ( fabs(h(x)) < 1e-10 )
    printf("root= %e\n", x);
  else
    printf("root not found!\n");

  return 0;
}
```

Example: Computing Convergent Series

$$e = \sum_{n=0}^{\infty} \frac{1}{n!} = \frac{1}{0!} + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \frac{1}{4!} + \cdots$$

Example: Convergent Series - forward summation

float6.c

```c
#include <stdio.h>
#include <math.h>

int main() {
  double sum = 1;
  double p_factorial = 1;
  int max_itr = 20;

  for (int p = 1; p <= max_itr; p++) {
    p_factorial *= p;
    sum += 1/p_factorial;
  }

  printf("sum= %.20f\n", sum);
  printf(" e = %.20f\n", M_E);
  printf("|sum-e|=%e\n", fabs(sum-M_E));

  return 0;
}
```

Example: Convergent Series - forward summation

float6.c

```c
#include <stdio.h>
#include <math.h>

int main() {
  double sum = 1;
  double p_factorial = 1;
  int max_itr = 20;

  for (int p = 1; p <= max_itr; p++) {
    p_factorial *= p;
    sum += 1/p_factorial;
  }

  printf("sum= %.20f\n", sum);
  printf(" e = %.20f\n", M_E);
  printf("|sum-e|=%e\n", fabs(sum-M_E));

  return 0;
}
```

```
b.nasihatkon@kntu:lecture19$ gcc float6.c && ./a.out
sum= 2.71828182845904553488
 e = 2.71828182845904509080
|sum-e|=4.440892e-16
```

Example: Convergent Series - backward summation

float7.c

```c
#include <stdio.h>
#include <math.h>

int main() {
  double p_factorial = 1;
  int max_itr = 21;

  for (int p = 1; p <= max_itr; p++)
    p_factorial *= p;

  double sum = 0;
  for (int p = max_itr; p >= 0; p--) {
    sum += 1/p_factorial;
    p_factorial /= p;
  }

  printf("sum= %.20f\n", sum);
  printf(" e = %.20f\n", M_E);
  printf("|sum-e|=%e\n", fabs(sum-M_E));

  return 0; }
```

Example: Convergent Series - backward summation

float7.c

```c
#include <stdio.h>
#include <math.h>

int main() {
  double p_factorial = 1;
  int max_itr = 21;

  for (int p = 1; p <= max_itr; p++)
    p_factorial *= p;

  double sum = 0;
  for (int p = max_itr; p >= 0; p--) {
    sum += 1/p_factorial;
    p_factorial /= p;
  }

  printf("sum= %.20f\n", sum);
  printf(" e = %.20f\n", M_E);
  printf("|sum-e|=%e\n", fabs(sum-M_E));

  return 0; }
```

```
b.nasihatkon@kntu:lecture19$ gcc float7.c && ./a.out
sum= 2.71828182845904509080
 e = 2.71828182845904509080
|sum-e|=0.000000e+00
```