[Chap.3-8] Machine-level Representation of Programs

Young Ik Eom (yieom@skku.edu, 031-290-7120)
Distributing Computing Laboratory
Sungkyunkwan University
http://dclab.skku.ac.kr

UNIVERSITY

Contents

- …
- Procedures
- Compound data structures
- Pointers
- GDB debugger
- Buffer overflow
- Floating-point codes

FP Operations

- **History**
  - Pentium/MMX @1997
    - Introduced MMX (media instructions)
      - ✓ Focused on allowing multiple operations to be performed in a parallel mode, called SIMD
      - ✓ SIMD (Single-Instruction Multiple-Data)
        - The same operation is performed on a number of different data values in parallel
    - MM registers (64-bits)

FP Operations

- **History**
  - Pentium III @1999
    - Introduced SSE
    - XMM registers (128-bits)
  - Pentium 4 @2000
    - Introduced SSE2
    - The media instructions have included ones to operate on scalar floating-point data, using single values in the low-order 32 or 64 bits of XMM registers

FP Operations

- **History**
  - Core i7 Sandy Bridge @2011
    - Introduced AVX
    - YMM registers (256-bits)
  - Now, AVX2 (Introduced with Core i7 Haswell @2013)
    - AVX2 code with the **gcc** command-line parameter **-mavx2**

FP Operations

- **FP registers (Media registers)**

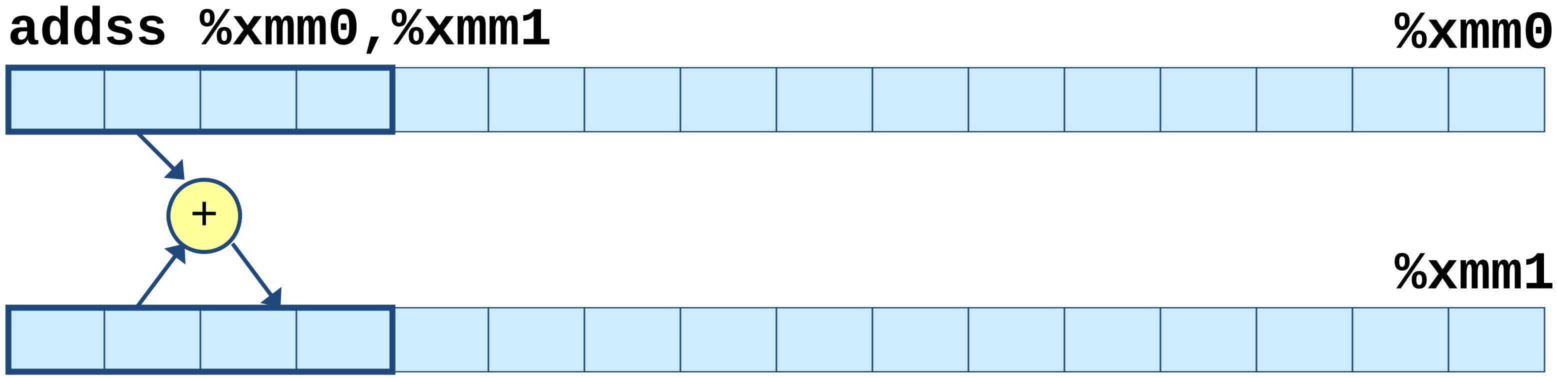
255	127 … 0	255	127 … 0
%ymm0	%xmm0	%ymm8	%xmm8
%ymm1	%xmm1	%ymm9	%xmm9
%ymm2	%xmm2	%ymm10	%xmm10
%ymm3	%xmm3	%ymm11	%xmm11
%ymm4	%xmm4	%ymm12	%xmm12
%ymm5	%xmm5	%ymm13	%xmm13
%ymm6	%xmm6	%ymm14	%xmm14
%ymm7	%xmm7	%ymm15	%xmm15

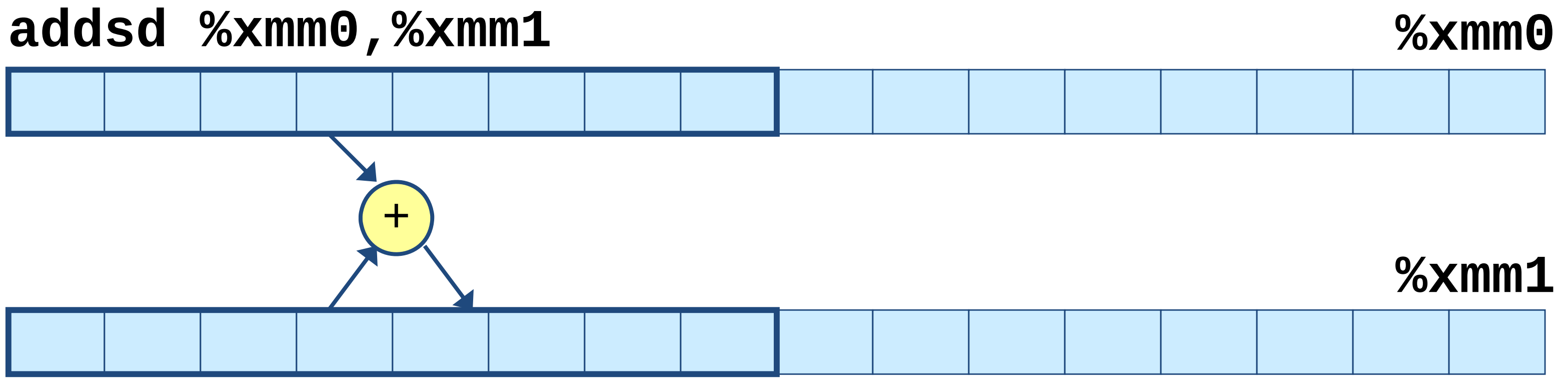
FP Operations

- **FP registers (Media registers)**
  - Each YMM register is 256 bits (32 bytes) long
  - When operating on scalar data, these registers only hold FP data, and only the low-order 32 bits (for float) or 64 bits (for double) are used
  - The assembly code refers to the registers by their SSE XMM register names %xmm0 ~ %xmm15, where each XMM register is the low-order 128 bits (16 bytes) of the corresponding YMM register

FP Operations

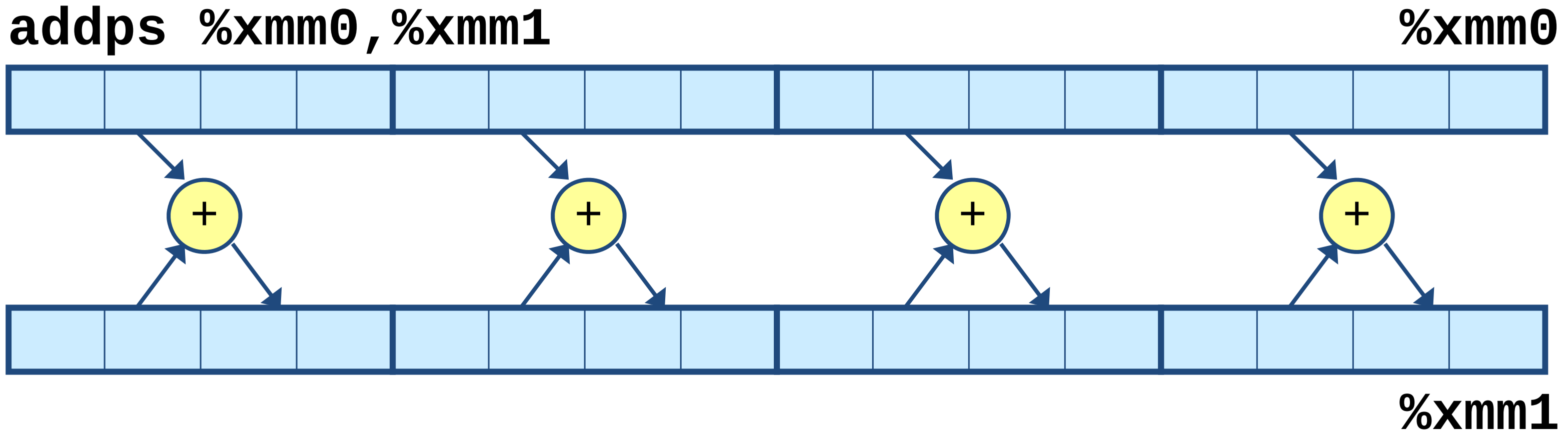
- **Scalar operations** and SIMD operations

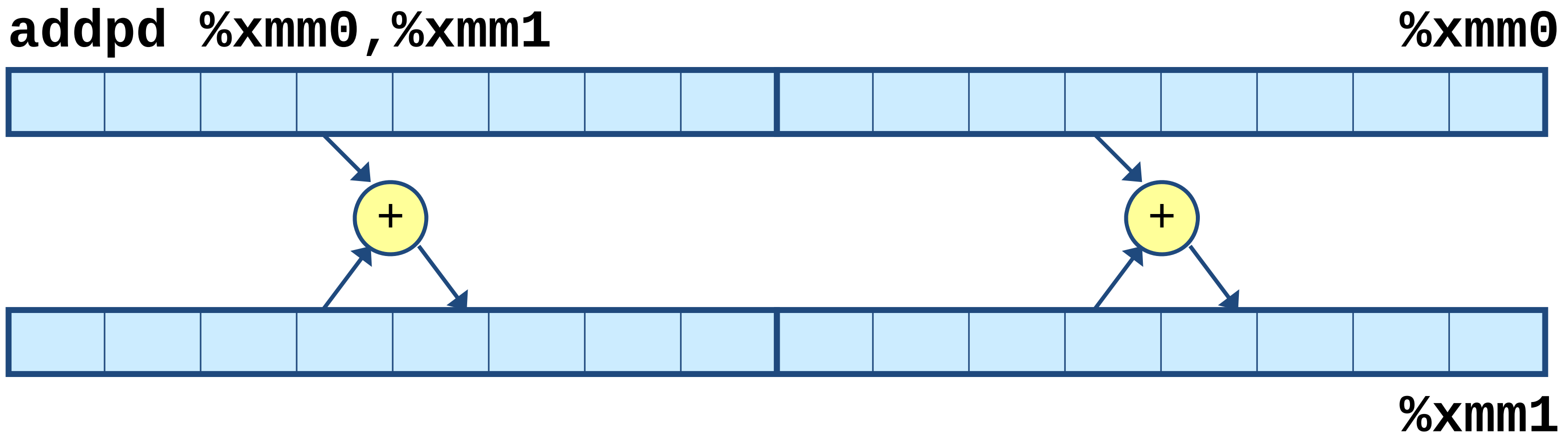




FP Operations

- Scalar operations and **SIMD operations**





FP Operations

FP movement

Instruction	Source	Destn	Description
`vmovss`	M_{32}	X	Move single precision
`vmovss`	X	M_{32}	Move single precision
`vmovsd`	M_{64}	X	Move double precision
`vmovsd`	X	M_{64}	Move double precision
`vmovaps`	X	X	Move aligned, packed single precision
`vmovapd`	X	X	Move aligned, packed double precision

X: XMM register, M_k: k-bit memory data (k = 32 or 64)

- Those instructions that reference memory are scalar instructions
- For transferring data between two XMM registers, it uses one of two different instructions for copying the entire contents of one XMM register to another

FP Operations

FP movement

- Example)

```
[C code]
float float_mov(float v1, float *src, float *dst) {
   float v2 = *src;
   *dst = v1;
   return v2;
}
```

```
[Assembly code]
   float float_mov(float v1, float *src, float *dst)
   v1 in %xmm0, src in %rdi, dst in %rsi
float_mov:
   vmovaps %xmm0,%xmm1      Copy vl
   vmovss (%rdi),%xmm0      Read v2 from src
   vmovss %xmm1,(%rsi)      Write vl to dst
   ret                      Return v2 in %xmm0
```

FP Operations

FP conversion (FP → Integer)

Instruction	Source	Destn	Description
`vcvttss2si`	X/M_{32}	R_{32}	Convert with truncation single precision to integer
`vcvttsd2si`	X/M_{64}	R_{32}	Convert with truncation double precision to integer
`vcvttss2siq`	X/M_{32}	R_{64}	Convert with truncation single precision to quad integer
`vcvttsd2siq`	X/M_{64}	R_{64}	Convert with truncation double precision to quad integer

R_k: k-bit integer register data (k = 32 or 64)

- All scalar instructions
- When converting FP values to integers
  - Perform truncation, rounding values toward 0

FP Operations

FP conversion (Integer → FP)

Instruction	Src1	Src2	Destn	Description
`vcvtsi2ss`	M_{32}/R_{32}	X	X	Convert integer to single precision
`vcvtsi2sd`	M_{32}/R_{32}	X	X	Convert integer to double precision
`vcvtsi2ssq`	M_{64}/R_{64}	X	X	Convert quad integer to single precision
`vcvtsi2sdq`	M_{64}/R_{64}	X	X	Convert quad integer to double precision

R_k: k-bit integer register data (k = 32 or 64)

- All scalar instructions
- 3-operand format with 2 sources and 1 destination
  - Now, we can ignore the 2nd operand

FP Operations

FP conversion

- Examples)
  - Convert long in **%rax** into double in **%xmm1**
    - ✓ `vcvtsi2sdq %rax,%xmm1,%xmm1`
  - Convert float in **%xmm0** into double in **%xmm0**
    - ✓ `vcvtss2sd %xmm0,%xmm0,%xmm0`
    - ✓ `vunpcklps %xmm0,%xmm0,%xmm0`
      `vcvtps2pd %xmm0,%xmm0`
  - Convert double in **%xmm0** into float in **%xmm0**
    - ✓ `vcvtsd2ss %xmm0,%xmm0,%xmm0`
    - ✓ `vmovddup %xmm0,%xmm0`
      `vcvtpd2psx %xmm0,%xmm0`

FP Operations

FP conversion

- Notes)
  - **`vunpcklps S1,S2,D`**
    - ✓ Used to interleave the values in 2 XMM registers and store them in a third XMM register
    - ✓ S1 = (~~s3~~,~~s2~~,s1,s0) and S2 = (~~t3~~,~~t2~~,t1,t0) → D = (s1,t1,s0,t0)
  - **`vcvtps2pd S,D`**
    - ✓ Expands the two low-order single-precision values in the source XMM register to be the two double-precision values in the destination XMM register
    - ✓ S = (~~s3~~,~~s2~~,s1,s0) → D = (ds1,ds0)

FP Operations

FP conversion

- Notes)
  - **`vmovddup S,D`**
    - ✓ Duplicate low-order double-precision value in a source XMM register
    - ✓ S = (ds1,ds0) → D = (ds0,ds0)
  - **`vcvtpd2psx S,D`**
    - ✓ Convert the 2 double-precision values in a source XMM register to single-precision, pack them into the low-order half of the register, and set the upper half to 0.0
    - ✓ S = (ds1,ds0) → D = (0.0,0.0,s1,s0)

FP Operations

FP conversion

- Example)

```
[C code]     rdi     rsi          rdx       rcx
double fcvt(int i, float *fp, double *dp, long *lp)
{
   float f = *fp; double d = *dp; long l = *lp;
   *lp = (long) d;
   *fp = (float) i;
   *dp = (double) l;
   return (double) f;
}
```

FP Operations

FP conversion

- Example)

```
[C code]
double fcvt(int i, float *fp, double *dp, long *lp)
{
   float f = *fp; double d = *dp; long l = *lp;
   *lp = (long) d;
   *fp = (float) i;
   *dp = (double) l;
   return (double) f;
}
```

```
[Assembly code]
   double fcvt(int i, float *fp, double *dp, long *lp)
   i in %edi, fp in %rsi, dp in %rdx, lp in %rcx
fcvt:
   vmovss (%rsi),%xmm0                Get f = *fp
   movq (%rcx),%rax                   Get l = *lp
   vcvttsd2siq (%rdx),%r8             Get d = *dp and convert to long
   movq %r8,(%rcx)                    Store at lp
   vcvtsi2ss %edi,%xmm1,%xmm1         Convert i to float
   vmovss %xmm1,(%rsi)                Store at fp
   vcvtsi2sdq %rax,%xmm1,%xmm1        Convert l to double
   vmovsd %xmm1,(%rdx)                Store at dp
   vunpcklps %xmm0,%xmm0,%xmm0        Convert f to double
   vcvtps2pd %xmm0,%xmm0              "
   ret                                Return f
```

FP Operations

FP code for procedures

- Up to 8 FP arguments can be passed in XMM registers **%xmm0** ~ **%xmm7**, in the order the arguments are listed
  - Additional FP arguments can be passed on the stack
- A function that returns a FP value does so in register **%xmm0**
- All XMM registers are caller saved

FP Operations

- **FP code for procedures**
  - Examples)
    - **`double f1(int x, double y, long z);`**
      - ✓ **x** in **%edi, y** in **%xmm0**, and **z** in **%rsi**
    - **`double f2(double y, int x, long z);`**
      - ✓ **x** in **%edi, y** in **%xmm0**, and **z** in **%rsi**
      - ✓ Same as the case of **`f1`**
    - **`double f3(float x, double *y, long *z);`**
      - ✓ **x** in **%xmm0, y** in **%rdi**, and **z** in **%rsi**

FP Operations

FP arithmetic operations

Single	Double	Effect	Description
`vaddss`	`vaddsd`	D ← S_2 + S_1	Floating-point add
`vsubss`	`vsubsd`	D ← S_2 - S_1	Floating-point subtract
`vmulss`	`vmulsd`	D ← S_2 * S_1	Floating-point multiply
`vdivss`	`vdivsd`	D ← S_2 / S_1	Floating-point divide
`vmaxss`	`vmaxsd`	D ← max(S_2,S_1)	Floating-point maximum
`vminss`	`vminsd`	D ← min(S_2,S_1)	Floating-point minimum
`sqrtss`	`sqrtsd`	D ← sqrt(S_1)	Floating-point square root

- Scalar FP instructions
- Each has 1 or 2 source operands and 1 destination operands
  - S_1: either an XMM register or a memory location
  - S_2 and D: XMM registers

FP Operations

- **FP arithmetic operations**
  - Example)

```
[C code]
double funct(double a, float x, double b, int i)
{
   return a*x - b/i;
}
```

```
[Assembly code]
   double funct(double a, float x, double b, int i)
   a in %xmm0, x in %xmm1, b in %xmm2, i in %edi
funct:
   vunpcklps %xmm1,%xmm1,%xmm1   Convert x to double
   vcvtps2pd %xmm1,%xmm1         "
   vmulsd %xmm0,%xmm1,%xmm0      Multiply a by x
   vcvtsi2sd %edi,%xmm1,%xmm1    Convert i to double
   vdivsd %xmm1,%xmm2,%xmm2      Compute b/i
   vsubsd %xmm2,%xmm0,%xmm0      Subtract from a*x
   ret                           Return
```

FP Operations

- **Defining and using FP constants**
  - AVX floating-point operations
    cannot have immediate values as operands
  - The compiler must allocate and initialize storage
    for any constant values and
    then the code reads the values from memory

FP Operations

- **Defining and using FP constants**
  - Example)

```
[C code]
double cel2fahr(double temp) {
   return 1.8 * temp + 32.0;
}
```

```
[Assembly code]
   double ce12fahr(double temp)
   temp in %xmm0
cel2fahr:
   vmulsd .LC2(%rip),%xmm0,%xmm0   Multiply by 1.8
   vaddsd .LC3(%rip),%xmm0,%xmm0   Add 32.0
   ret
.LC2:
   .long 3435973837                Low-order 4 bytes of 1.8
   .long 1073532108                High-order 4 bytes of 1.8
.LC3:
   .long 0                         Low-order 4 bytes of 32.0
   .long 1077936128                High-order 4 bytes of 32.0
```

FP Operations

FP bitwise operations

Single	Double	Effect	Description
`vxorps`	`vxorpd`	$D \leftarrow S_2$ ^ S_1	Bitwise XOR
`vandps`	`vandpd`	$D \leftarrow S_2$ & S_1	Bitwise AND

- These operations all act on packed data
  - Update the entire destination XMM register, applying the bitwise operation to all the data in the two source registers
- These operations often provides simple and convenient ways to manipulate FP values

FP Operations

FP bitwise operations

- Example)

```
[C code]
double simpfun(double x) {
   return –x;
}
```

```
[Assembly code]
   double simpfun(double x)
   x in %xmm0
simpfun:
   vmovsd .LC2(%rip),%xmm1          Get -0.0
   vxorpd %xmm1,%xmm0,%xmm0         Get -x
   ret
.LC2:
   .long 0                          Low-order 4 bytes of -0.0
   .long -2147483648                High-order 4 bytes of -0.0
   .long 0
   .long 0
```

FP Operations

FP comparison operations

Instruction	Based on	Description
`ucomiss` S_1, S_2	$S_2 - S_1$	Compare single precision
`ucomisd` S_1, S_2	$S_2 - S_1$	Compare double precision

- Similar to the CMP instruction
  - Compare S_2 and S_1
    - ✓ S_2 in XMM register, S_1 in XMM register or in memory
  - Set CCs (ZF, CF, PF)
    - ✓ PF is set when either operand is NaN

$S_2 : S_1$	CF	ZF	PF
Unordered	1	1	1
$S_2 < S_1$	1	0	0
$S_2 = S_1$	0	1	0
$S_2 > S_1$	0	0	0

Summary