[Chap.3-2] Machine-level Representation of Programs

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

SKK UNIVERSITY

Contents

- Introduction
- Program encodings
- Data formats
- Intel processors
- Accessing information
- Primitive instructions
- Data movement instructions
- Arithmetic and logic instructions
- Control instructions
- Procedures
- …

Intel Processors

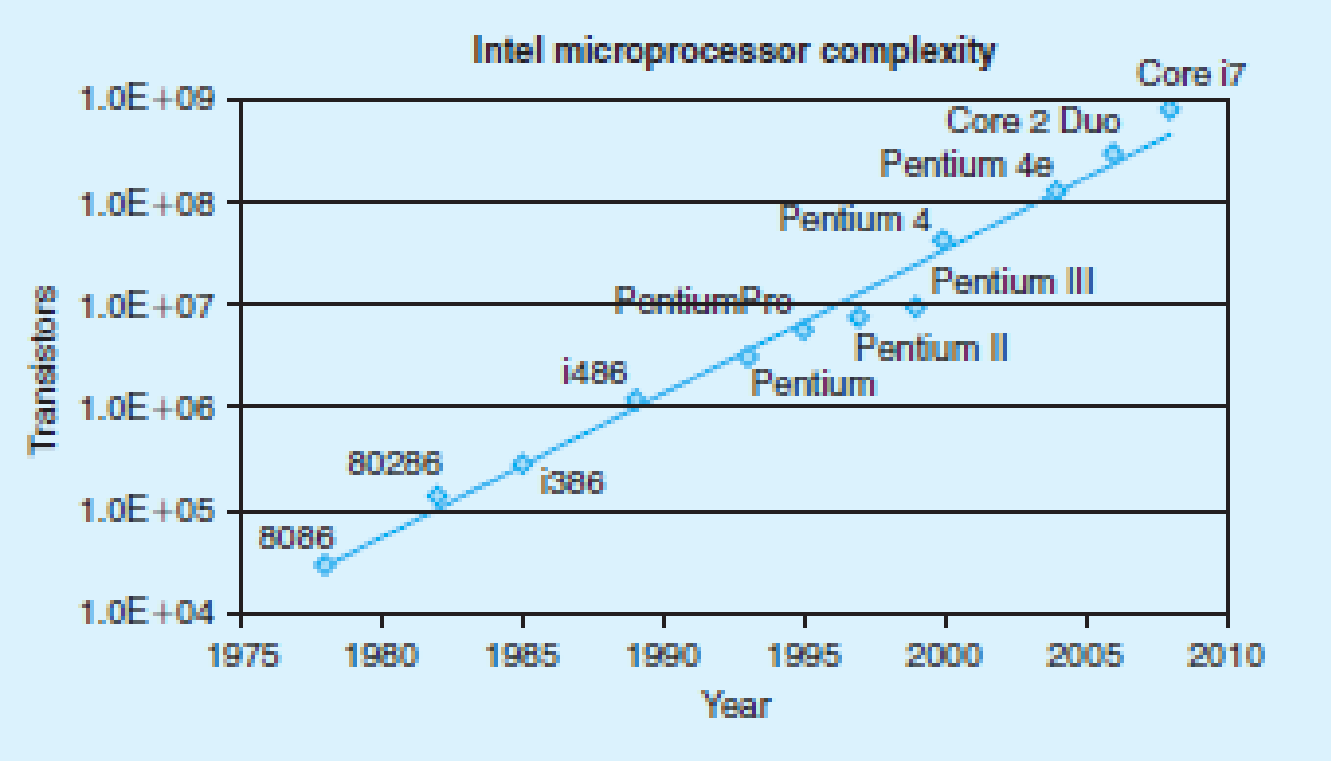
Intel processors

- CISC (Complex Instruction Set Computer)
  - Many different instructions with many different formats
  - Hard to match performance of RISC machines
    - ✓ RISC (Reduced Instruction Set Computer)
  - But, Intel has done just that!

- Evolutionary design
  - Starting in 1978 with 8086
  - Added more features as time goes on
  - Still support old features, although obsolete
  - Totally dominates computer market

Intel Processors

- Models of Intel processors



Intel Processors

Models of Intel processors

1978	8086	x86 is born
1980	8087	x87 is born
1985	i386	IA32
1995	Pentium Pro	PAE
1997	Pentium MMX	MMX
1999	Pentium III	SSE
2000	Pentium 4	SSE2
2004	Pentium 4E	Hyperthreading, 64-bit extension of IA32
2005	Pentium 4 662	Intel VT
2006	Core 2	SSE4
2008	Core i7 Nehalem	Hyperthreading + Multicore
2011	Core i7 Sandy Bridge	AVX(extension of SSE), Support of 256-bit vectors
2013	Core i7 Haswell	AVX2

Intel Processors

- **Models of Intel processors**
  - 8086 (1978, 29K transistors)
    - 16-bit microprocessor
    - MS-DOS by Microsoft
    - 8088 (8-bit version)
    - 8087 (FP coprocessor)
  - 80286 (1982, 134K transistors)
    - More addressing modes
    - IBM PC/AT
    - MS Windows

Intel Processors

- **Models of Intel processors**
  - i386 (1985, 275K transistors)
    - 32-bit architecture (IA32)
    - Flat addressing model
    - Unix OS support
  - i486 (1989, 1.2M transistors)
    - Improved performance
    - Integrated the FP unit onto the processor chip
  - Pentium (1993, 3.1M transistors)
    - Improved performance
    - Minor extensions to the instruction set

Intel Processors

Models of Intel processors

- Pentium Pro (1995, 5.5M transistors)
  - P6 micro architecture (new processor design)
  - "Conditional move" instructions
- Pentium/MMX (1997, 4.5M transistors)
  - New class of instructions for manipulating vectors of integers
- Pentium II (1997, 7M transistors)
- Pentium III (1999, 8.2M transistors)
  - Introduced SSE (Streaming SIMD Extensions)
    - ✓ A class of instructions for manipulating vectors of integer/FP data
- Pentium 4 (2000, 42M transistors)
  - Extended SSE to SSE2
    - ✓ Supports new data types such as double-precision FP

Intel Processors

- **Models of Intel processors**
  - Pentium 4E (2004, 125M transistors)
    - Hyperthreading
    - x86-64 (EM64T, 64-bit extension to IA32)
  - Core 2 (2006, 291M transistors)
    - Returned back to P6-like microarchitecture
    - Multi-core microprocessor
    - No hyperthreading
  - Core i7 Nehalem (2008, 781M transistors)
    - Incorporated both hyperthreading and multi-core
    - Up to 4 cores on each chip

Intel Processors

Models of Intel processors

- Core i7 Nehalem (2008, 781M transistors)
  - Incorporated both hyperthreading and multi-core
  - Up to 4 cores on each chip
- Core i7 Sandy Bridge (2011, 1.17G transistors)
  - Introduced AVX(Advanced Vector Extensions), an extension of SSE
    - ✓ Support 256-bit vectors
- Core i7 Haswell (2013, 1.4G transistors)
  - Extended AVX to AVX2
    - ✓ More instructions and instruction formats

Intel Processors

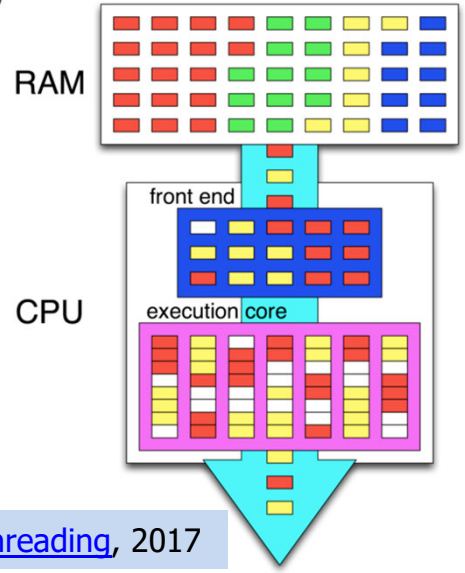
Notes

- SSE (Streaming SIMD Extensions)
  - SIMD instruction set extension to the x86 architecture
    - ✓ SIMD instructions can greatly increase performance when exactly the same operations are to be performed on multiple data objects
    - ✓ Typical applications are digital signal processing and graphics processing
  - Designed by Intel and introduced in 1999 in their Pentium III series processors as a reply to AMD's 3DNow
  - Contains 70 new instructions, most of which work on single precision FP data

Intel Processors

Notes

- HTT (HyperThreading Technologies)
  - Intel's proprietary simultaneous multithreading (SMT) implementation used to improve parallelization of computations performed on x86 microprocessors
  - For each processor core that is physically present, the OS addresses two virtual or logical cores, and shares the workload between them when possible
  - With HTT, one physical core appears as two processors to the OS, allowing concurrent scheduling of two processes per core

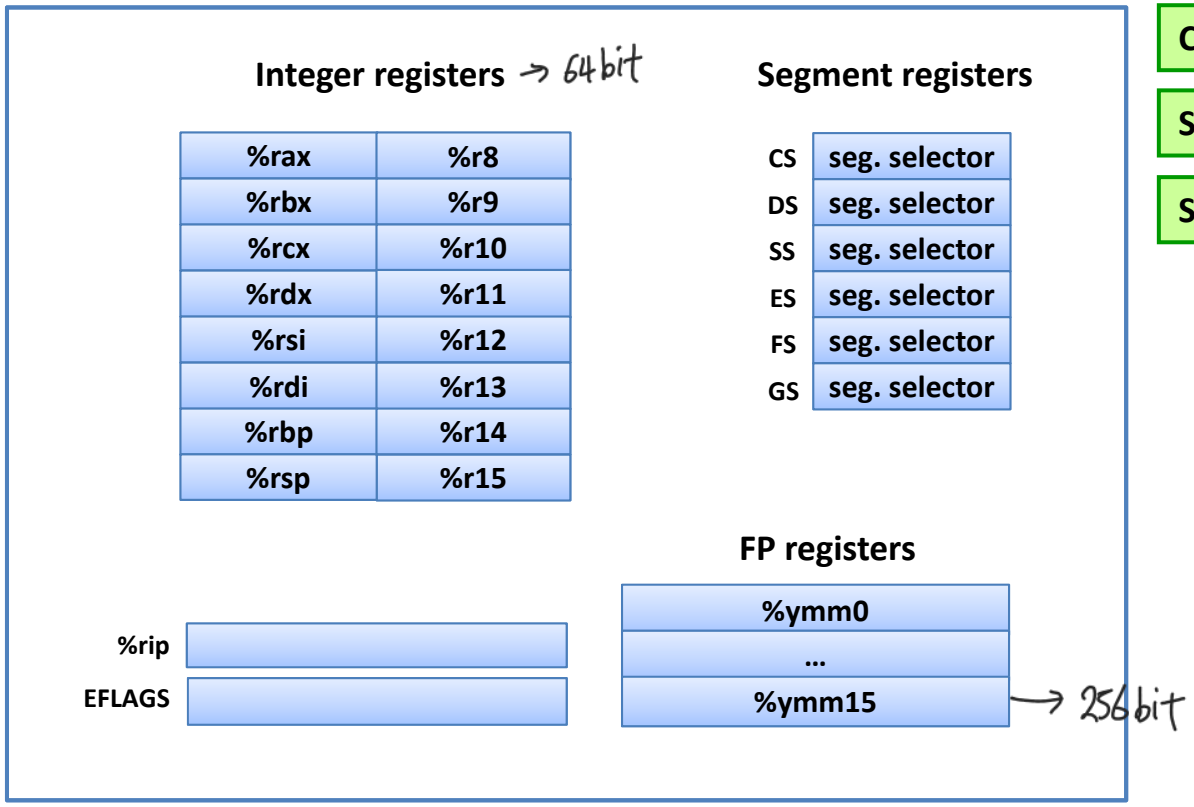


https://en.wikipedia.org/wiki/Hyper-threading, 2017

Accessing Information

Basic execution environment

Application Programming Registers



- Control registers
- System Table Registers
- System Segment Registers

Accessing Information

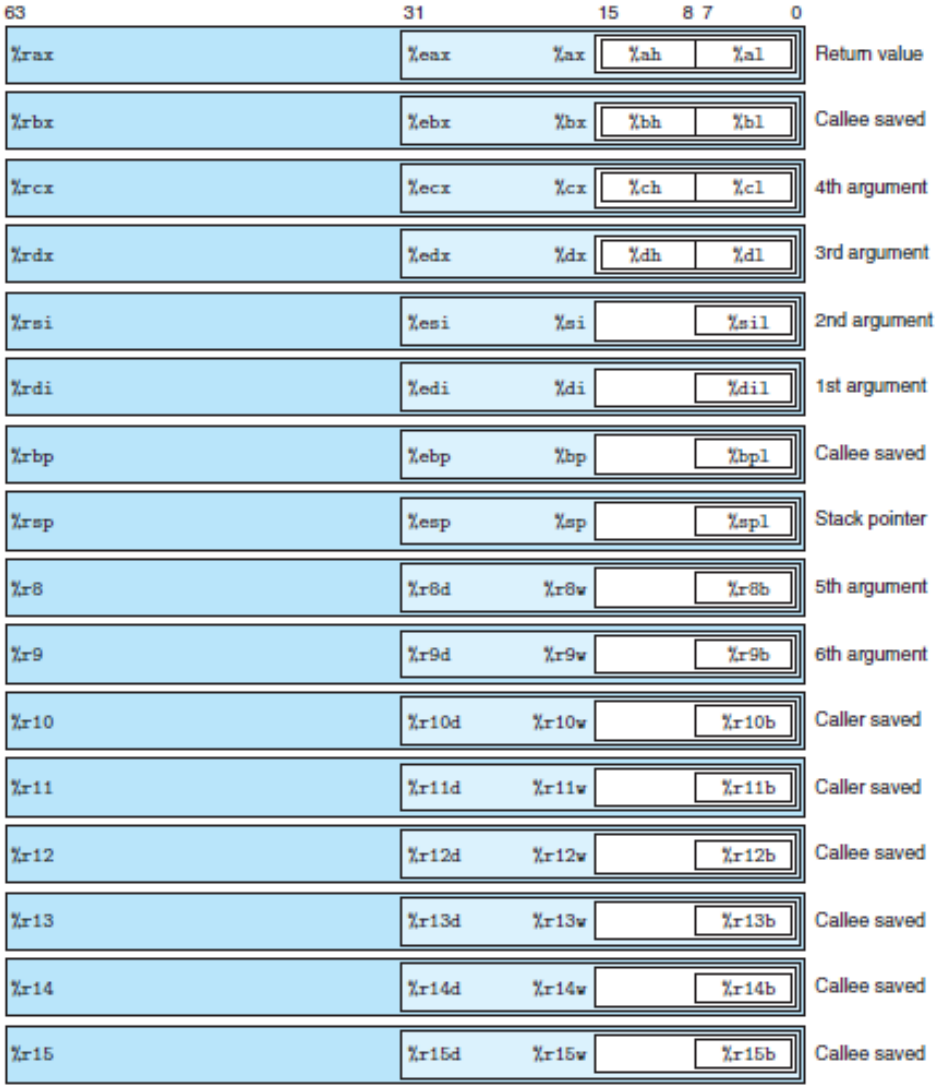
Integer registers

63	31	0
%rax	%eax	
%rbx	%ebx	
%rcx	%ecx	
%rdx	%edx	
%rsi	%esi	
%rdi	%edi	
%rbp	%ebp	
%rsp	%esp	

63	31	0
%r8	%r8d	
%r9	%r9d	
%r10	%r10d	
%r11	%r11d	
%r12	%r12d	
%r13	%r13d	
%r14	%r14d	
%r15	%r15d	

Accessing Information

- Integer registers



Accessing Information

Integer registers

- A set of 16 general-purpose registers storing 64-bit values
  - Used to store integer data as well as pointers
  - %rax ~ %rsp, %r8 ~ %r15

- Note)
  - 8086
    - ✓ Eight 16-bit registers (%ax ~ %sp)
  - IA32
    - ✓ Eight 32-bit registers (%eax ~ %esp)

Accessing Information

Special rules on using registers

- Instructions can operate on data of different sizes stored in the low-order bytes of the 16 registers
  - Byte-level operations can access the least significant byte
  - 16-bit operations can access the least significant 2 bytes
  - 32-bit operations can access the least significant 4 bytes
  - 64-bit operations can access entire registers
- Instructions can use registers as destinations with two conventions for what happens to the remaining bytes in the register for instructions that generate less than 8 bytes
  - Those that generate 1- or 2-byte quantities leave the remaining bytes unchanged
  - Those that generate 4-byte quantities set the upper 4 bytes of the register to zero

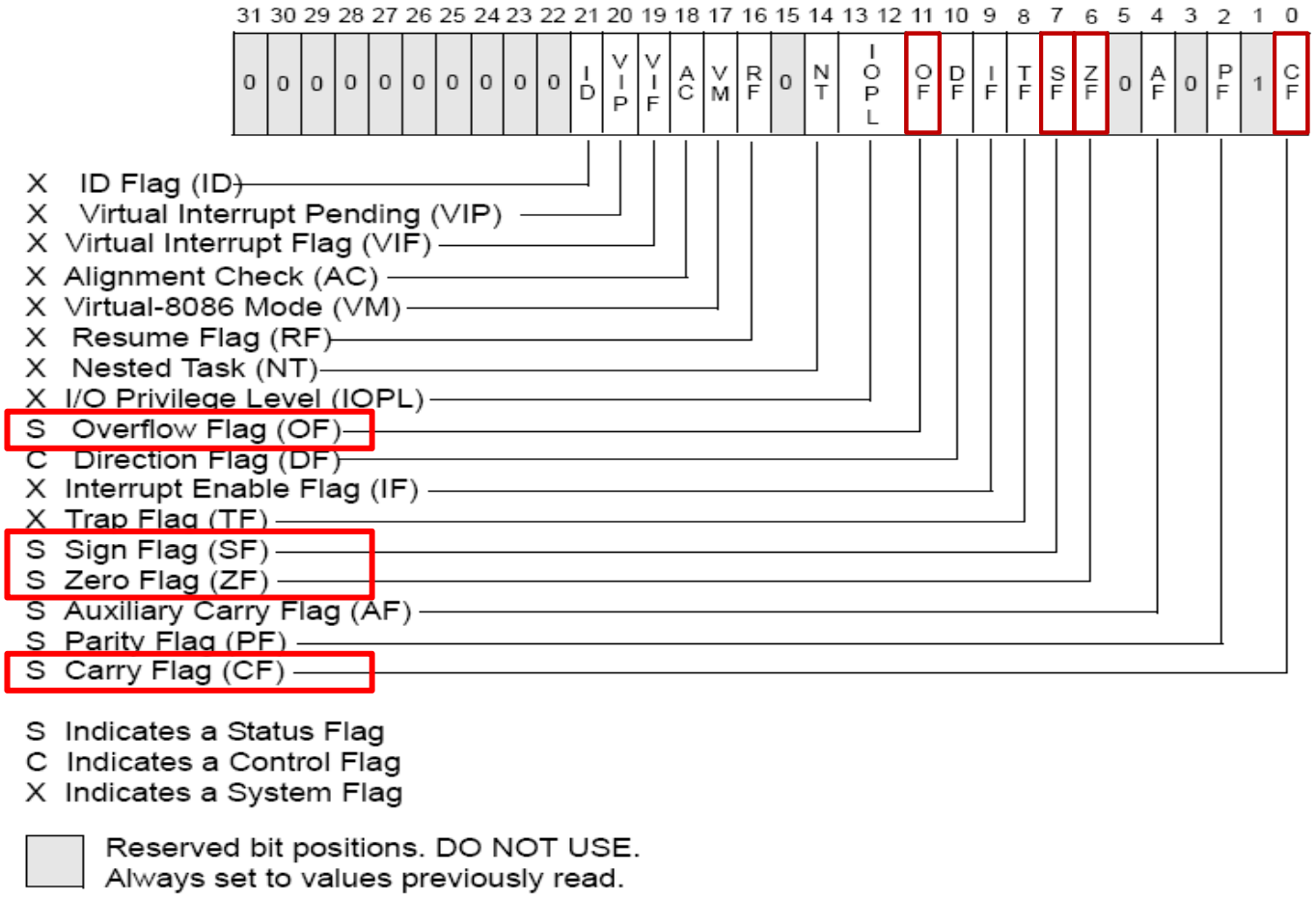
Accessing Information

Special rules on using registers

- Different registers serve different roles
  - `%rsp` (stack pointer)
    - ✓ Used to indicate the top position of the stack
  - Some instructions make specific use of certain registers
    - ✓ Passing function arguments
    - ✓ Returning values from functions
    - ✓ Storing local and temporary data
    - ✓ Etc…

Accessing Information

EFLAGS register



Accessing Information

EFLAGS register: status flags

Flags	Description
CF (Carry)	Set if an arithmetic operation generates a carry or a borrow; Indicates an overflow condition for unsigned-integer arithmetic
PF (Parity)	Set if the least-significant byte of the result contains an even number of 1 bits
AF (Adjust)	Set if an arithmetic operation generates a carry or a borrow out of bit 3 of the result; Used in binary-coded decimal (BCD) arithmetic
ZF (Zero)	Set if the result is zero
SF (Sign)	Set equal to the most-significant bit of the result
OF (Overflow)	Set if the integer result is too large a positive number or too small a negative number to fit in the destination operand; Indicates an overflow condition for signed-integer arithmetic
DF (Direction)	Setting the DF causes the string instructions to auto-decrement; Set and cleared by STD/CLD instructions

Accessing Information

- **`%rip` register**
  - Contains the offset in the current code segment for the next instruction to be executed
    - Advances from one instruction boundary to the next in straightline code, or
    - Moves ahead or backwards by instructions such as `JMP`, `JCC`, `CALL`, `RET`, and `IRET`
  - Cannot be accessed directly by software
    - **`%rip`** is controlled implicitly by control transfer instructions, interrupts, and exceptions
  - Because of instruction prefetching, an instruction address read from the bus may not match the value in the **`%rip`** register

Accessing Information

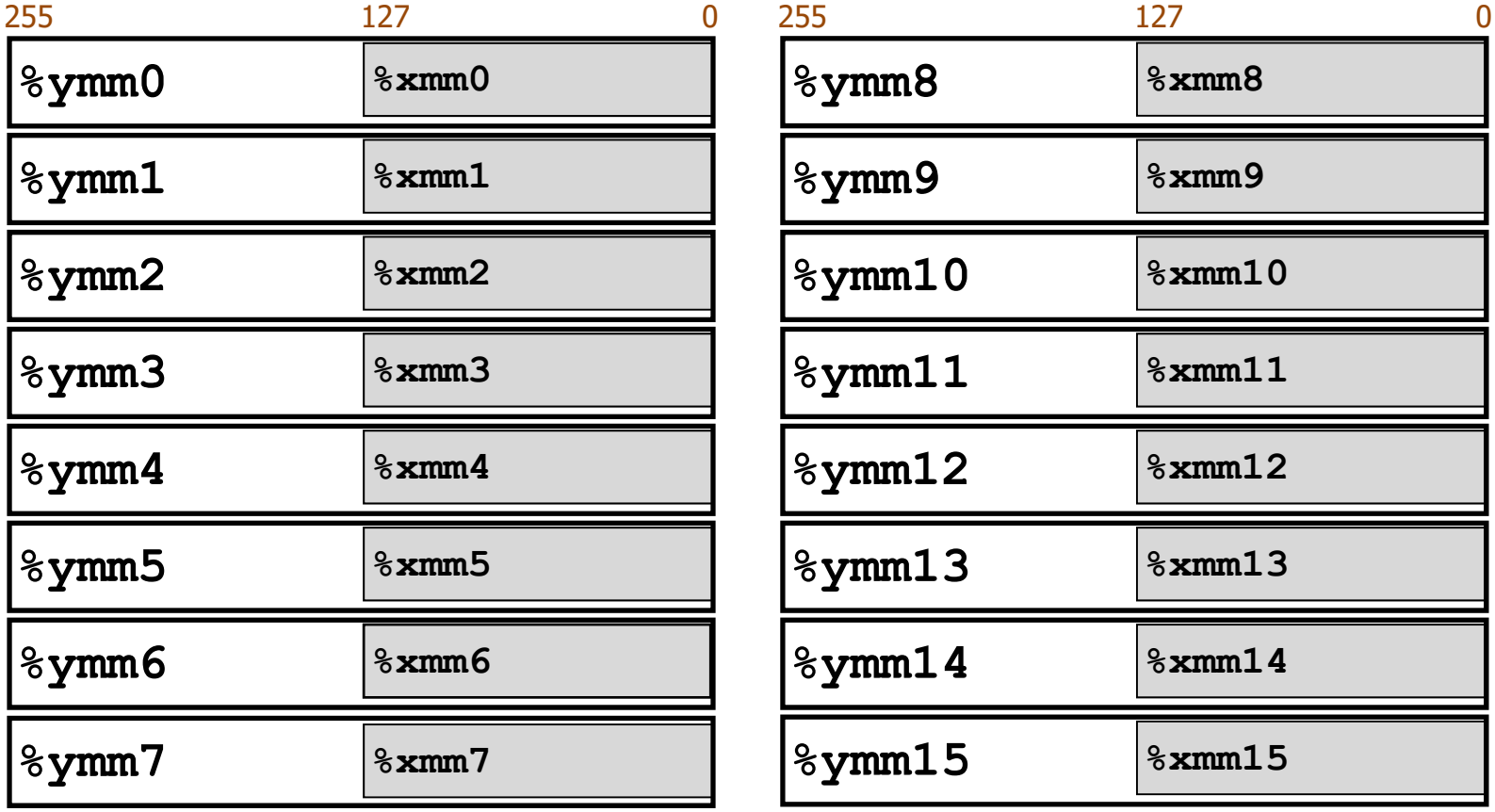
FP registers

- For SSE
  - 16 registers, all 128-bits long
  - **%xmm0 ~ %xmm15**
  - Can hold four 32-bit values or two 64-bit values

- For AVX
  - 16 registers, all 256-bits long
  - **%ymm0 ~ %ymm15**
    - ✓ Includes **%xmm0~%xmm15** as low-order 128-bits of each register
  - Can hold eight 32-bit values or four 64-bit values

Accessing Information

FP registers



Primitive Instructions

Manual

- x86-64 and IA32 Architectures Software Developer Manuals
  - Volume 1: Basic Architecture
  - Volume 2A, 2B: Instruction Set Reference
  - Volume 3A, 3B: System Programming Guide
- Available online:
  - http://www.intel.com/products/processor/manuals/

Primitive Instructions

- **Types of primitive operations**
  - Transfer data between memory and register
    - Load data from memory into register
    - Store register data into memory
  - Perform an arithmetic/logical function on register or memory data
  - Transfer control
    - Unconditional jumps (branches)
    - Conditional jumps (branches)
    - Procedure calls and returns
  - Etc…

Primitive Instructions

Operand specifiers

- 3 types of operands
  - **Immediate**
    - ✓ Prefix $
  - **Register**
    - ✓ Register name
  - **Memory**
    - ✓ Most general form **Imm(E_b,E_i,s)**
    - ✓ The effective address is obtained by **Imm + R[E_b] + s·R[E_i]**
      where **Imm**: offset (displacement),
      E_b: base register,
      E_i: index register,
      s: scale factor

Primitive Instructions

Operand specifiers

Type	Form	Operand value	Name
Immediate	$\$Imm$	Imm	Immediate
Register	E_a	$R[E_a]$	Register
Memory	Imm	$M[Imm]$	Absolute
Memory	(E_a)	$M[R[E_a]]$	Indirect
Memory	$Imm(E_b)$	$M[Imm + R[E_b]]$	Base + displacement
Memory	(E_b, E_i)	$M[R[E_b] + R[E_i]]$	Indexed
Memory	$Imm(E_b, E_i)$	$M[Imm + R[E_b] + R[E_i]]$	Indexed
Memory	$(, E_i, s)$	$M[R[E_i] \cdot s]$	Scaled indexed
Memory	$Imm(, E_i, s)$	$M[Imm + R[E_i] \cdot s]$	Scaled indexed
Memory	(E_b, E_i, s)	$M[R[E_b] + R[E_i] \cdot s]$	Scaled indexed
Memory	$Imm(E_b, E_i, s)$	$M[Imm + R[E_b] + R[E_i] \cdot s]$	Scaled indexed

Primitive Instructions

Operand specifiers: Example

%rdx	0x0000f000
%rcx	0x00000100

Expression	Computation	Address
0x8(%rdx)	0xf000 + 0x8	0x0000f008
(%rdx,%rcx)	0xf000 + 0x0100	0x0000f100
(%rdx,%rcx,4)	0xf000 + 4*0x0100	0x0000f400
0x80(,%rdx,2)	2*0xf000 + 0x80	0x0001e080

Primitive Instructions

Operand specifiers: Example

Address	Value	Register	Value
0x100	0xFF	%rax	0x100
0x104	0xAB	%rcx	0x1
0x108	0x13	%rdx	0x3
0x10C	0x11		

Operand	Value
%rax	
0x104	
$0x108	
(%rax)	
4(%rax)	
9(%rax,%rdx)	
260(%rcx,%rdx)	
0xFC(,%rcx,4)	
(%rax,%rdx,4)	

Summary