Module2: Machine Prog: Basic & Control

Machine-Level Programming I: Basics

Our Coverage:

Book covers x86-64

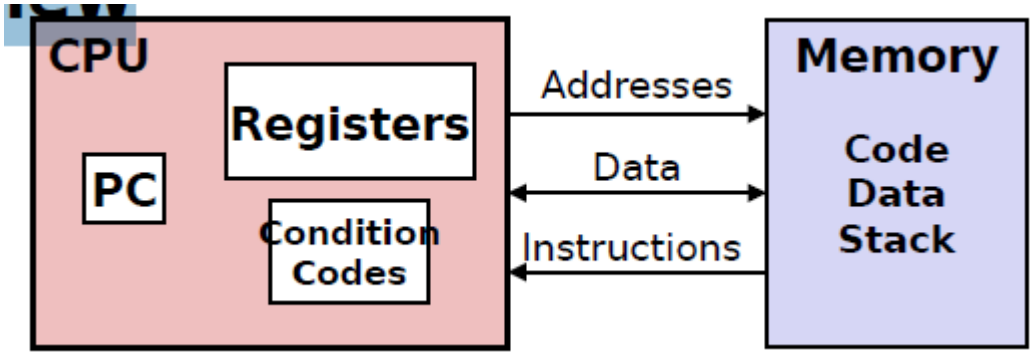
Web aside on IA32

We will only cover **x86-64**

Definitions

- **Architecture**: (also ISA: instruction set architecture) The parts of a processor design that one needs to understand for writing assembly/machine code.
    - Examples: instruction set specification, registers
- **Microarchitecture**: Implementation of the architecture
    - Examples: cache sizes and core frequency
- Code Forms:
    - Machine Code: The byte-level programs that a processor executes
    - Assembly Code: A text representation of machine code

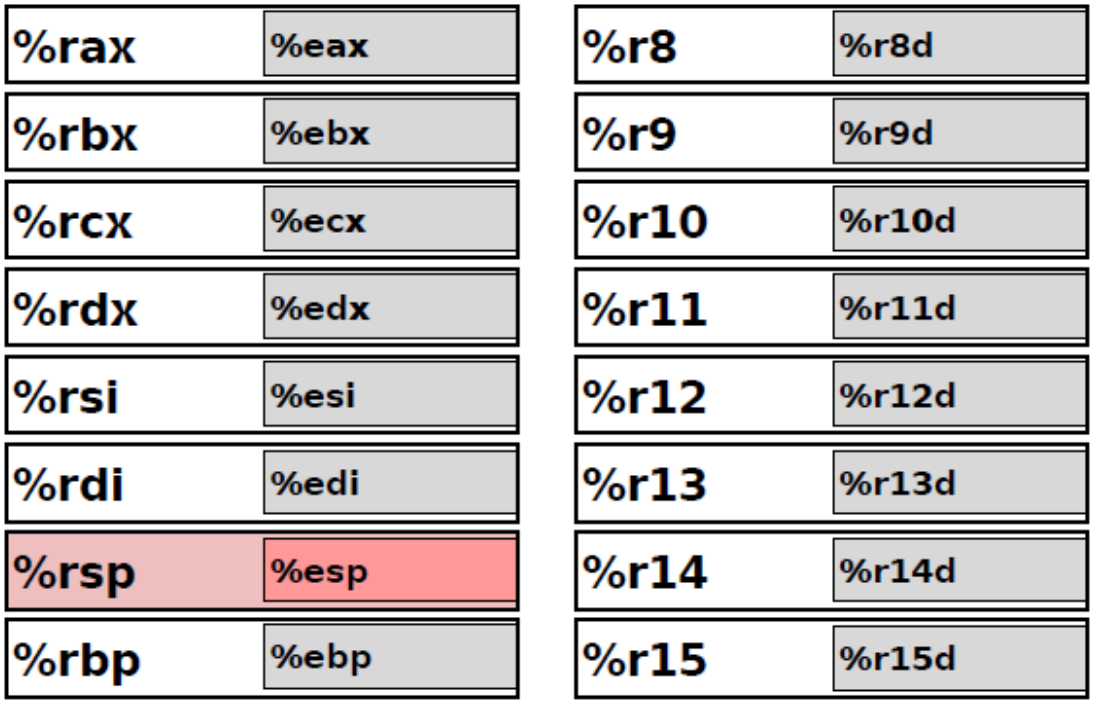
Assembly/Machine Code View：

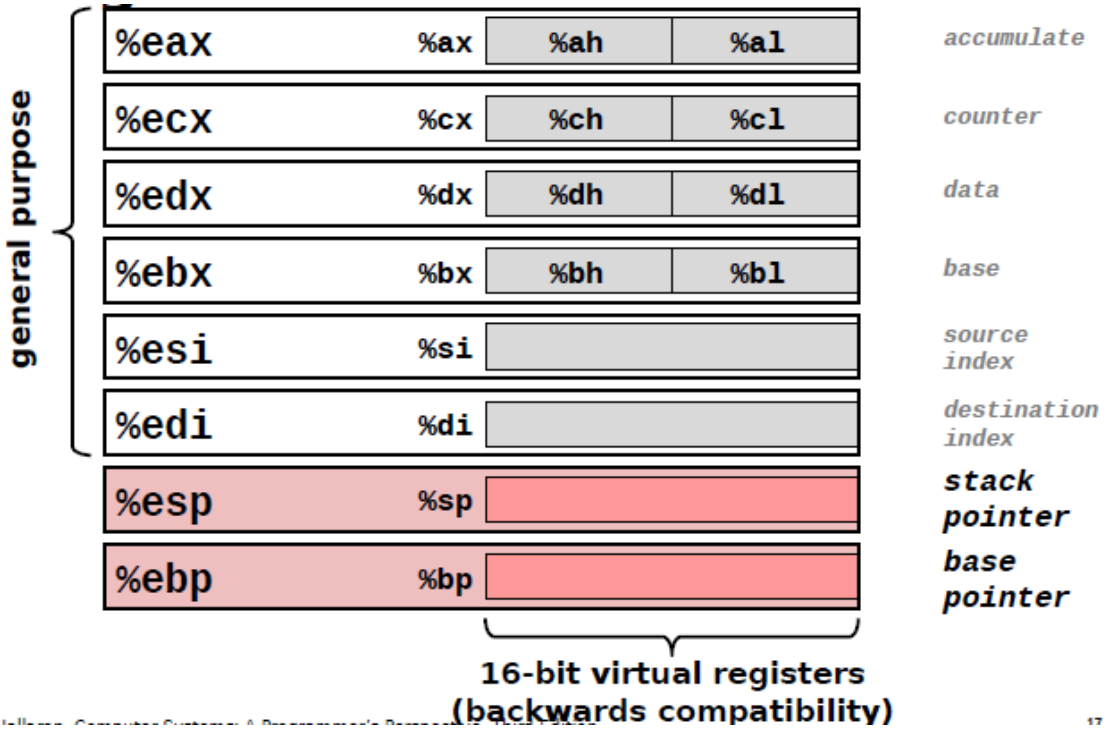


Assembly Characteristics: Data Types

- "Integer" data of 1, 2, 4, or 8 bytes
    1. Data values
    2. Addresses (untyped pointers)
- Floating point data of 4, 8, or 10 bytes
- Code: Byte sequences encoding series of instructions
- No aggregate types such as arrays or Structures

x86-64 Integer Registers





First four type: mainly used to save data;

%esi and %edi: index register;

%esp and %ebp: pointer register;

Moving Data Instruction:

movq Source Dest(or mov Source Dest)

• **Immediate**: Constant integer data

- Example: $0x400, $-533
- Like C constant, but prefixed with '$'
- Encoded with 1, 2, or 4 bytes

• **Register**: One of 16 integer registers
- Example: %rax, %r13
- But **%rsp** reserved for special use
- Others have special uses for particular instructions

• Memory: 8 consecutive bytes of memory at address given by register
- Simplest example: (%rax)
- Various other "addressing modes"

Simple Memory Addressing Modes:

• Normal **(R)**: **Mem[Reg[R]]** → movq (%rcx),%rax

• Displacement **D(R)**: **Mem[Reg[R]+D]** → movq 8(%rbp),%rdx

• Most General Form **D(Rb,Ri,S)** : **Mem[Reg[Rb]+S*Reg[Ri]+ D]**
- D: Constant "displacement" 1, 2, or 4 bytes
- Rb: Base register: Any of 16 integer registers
- Ri: Index register: Any, except for %rsp
- S: Scale: 1, 2, 4, or 8 (why these numbers?)

• Special Cases

(Rb,Ri): **Mem[Reg[Rb]+Reg[Ri]]**

D(Rb,Ri): Mem[Reg[Rb]+Reg[Ri]+D]

(Rb,Ri,S): Mem[Reg[Rb]+S*Reg[Ri]]

Example:

%rdx	0xf000
%rcx	0x0100

Expression	Address Computation	Address
0x8(%rdx)	0xf000 + 0x8	0xf008
(%rdx,%rcx)	0xf000 + 0x100	0xf100
(%rdx,%rcx,4)	0xf000 + 4*0x100	0xf400
0x80(,%rdx,2)	2*0xf000 + 0x80	0x1e080

Address Computation Instruction:

• **leaq Src, Dst**
- Src is address mode expression
- Set Dst to address denoted by expression

• Uses
- Computing addresses without a memory reference

- E.g., translation of p = &x[i];
- Computing arithmetic expressions of the form x + k*y (k = 1, 2, 4, or 8)

Arithmetic Operations Instructions:

Two Operand Instructions:

Format	Computation		
addq	Src,Dest	Dest = Dest + Src	
subq	Src,Dest	Dest = Dest – Src	
imulq	Src,Dest	Dest = Dest * Src	
salq	Src,Dest	Dest = Dest << Src	Also called shlq
sarq	Src,Dest	Dest = Dest >> Src	Arithmetic
shrq	Src,Dest	Dest = Dest >> Src	Logical
xorq	Src,Dest	Dest = Dest ^ Src	
andq	Src,Dest	Dest = Dest & Src	
orq	Src,Dest	Dest = Dest | Src	

One Operand Instructions

incq	Dest	Dest = Dest + 1
decq	Dest	Dest = Dest – 1
negq	Dest	Dest = – Dest
notq	Dest	Dest = ~Dest

Machine-Level Programming II: Controls

Processor State

Information about currently executing program:

- Temporary data(%rax)
- Location of runtime stack(%rsp)
- Location of current code control point(%rip)
- Status of recent tests(CF, ZF, SF, OF)

Condition Codes

- CF: Carry Flag(for unsigned)
- SF: Sign Flag(for signed)
- ZF: Zero Flag(for both)
- OF: Overflow Flag(for signed)

Implicitly set (as side effect) of arithmetic of arithmetic operations

Example: addq Src ,Dest ↔ t = a+b

CF if carry out from most significant bit (unsigned overflow)

ZF set ZF setZF set if t == 0

SF set if t < 0 (as signed)

OF set if two's complement (signed) overflow **(a>0 && b>0 && t<0) || (a<0 && b<0 && t>=0)**

Explicit Setting by Compare Instruction

cmpq Src2, Src1

cmpq b,a like computing like computing a-b without setting destination

CF set if carry out from most significant bit (used for unsigned comparisons)

ZF set if a == b

SF set if (a -b) < 0 (as signed)

OF set if two's complement (signed) overflow **(a>0 && b<a -b)<0) || (a<0 && b>a -b)>0)**

Explicit Setting by Test instruction

Testq Src2, Src1

testq b,a like computing a&b without setting destination

Sets condition codes based on value of **Src1&Src2**

Useful to have one of the operands be a mask

ZF set when a&b == 0

SF set when a&b < 0

SetX Instructions

- Set low-order byte of destination to 0 or 1 based on combinations of condition codes
- Does not alter remaining 7 bytes

SetX	Condition	Description
sete	ZF	Equal / Zero
setne	~ZF	Not Equal / Not Zero
sets	SF	Negative
setns	~SF	Nonnegative
setg	~(SF^OF)&~ZF	Greater (Signed)
setge	~(SF^OF)	Greater or Equal (Signed)
setl	(SF^OF)	Less (Signed)
setle	(SF^OF)|ZF	Less or Equal (Signed)
seta	~CF&~ZF	Above (unsigned)
setb	CF	Below (unsigned)

Jumping

jX Instructions

- Jump to different part of code depending on condition codes

jX	Condition	Description
jmp	1	Unconditional
je	ZF	Equal / Zero
jne	~ZF	Not Equal / Not Zero
js	SF	Negative
jns	~SF	Nonnegative
jg	~(SF^OF)&~ZF	Greater (Signed)
jge	~(SF^OF)	Greater or Equal (Signed)
jl	(SF^OF)	Less (Signed)
jle	(SF^OF)|ZF	Less or Equal (Signed)
ja	~CF&~ZF	Above (unsigned)
jb	CF	Below (unsigned)

Movzbl %al, %eax: zapped the rest bits to all 0

Switch-Structure

Code Blocks (x == 2, x == 3)

```
    long w = 1;
       . . .
    switch(x) {
       . . .
    case 2:
        w = y/z;
        /* Fall Through */
    case 3:
        w += z;
        break;
       . . .
    }
```

```
.L5:                    # Case 2
   movq    %rsi, %rax
   cqto
   idivq   %rcx         #  y/z
   jmp     .L6          #  goto merge
.L9:                    # Case 3
   movl    $1, %eax     #  w = 1
.L6:                    # merge:
   addq    %rcx, %rax   #  w += z
   ret
```

Register	Use(s)
%rdi	Argument x
%rsi	Argument y
%rdx	Argument z
%rax	Return value

Code Blocks (x == 5, x == 6, default)

```
  switch(x) {
    . . .
    case 5:  // .L7
    case 6:  // .L7
        w -= z;
        break;
    default: // .L8
        w = 2;
}
```

```
.L7:                # Case 5,6
  movl  $1, %eax    #  w = 1
  subq  %rdx, %rax  #  w -= z
  ret
.L8:                # Default:
  movl  $2, %eax    #  2
  ret
```

Register	Use(s)
%rdi	Argument **x**
%rsi	Argument **y**
%rdx	Argument **z**
%rax	Return value