Procedure & Data

X86 family:

- 8086(1978, 29K)
  - The heart of the IBM PC & DOS (8088)
  - 16-bit, 1M bytes addressable, 640K for users
  - x87 for floating pointing
- 80286(1982, 134K)
  - More (now obsolete) addressing modes
  - Basis of the IBM PC-AT & Windows
- i386(1985, 275K)
  - 32 bits architecture, flat addressing model
  - Support a Unix operating system
- I486(1989, 1.9M)
  - Integrated the floating-point unit onto the processor chip
- Pentium(1993, 3.1M)
  - Improved performance, added minor extensions
- PentiumPro(1995, 5.5M)
  - P6 microarchitecture
  - Conditional mov
- Pentium II(1997, 7M)
  - Continuation of the P6

- Pentium III(1999, 8.2M)
    - New class of instructions for manipulating vectors of floating-point numbers(SSE, Stream SIMD Extension)
    - Later to 24M due to the incorporation of the level-2 cache
- Pentium 4(2001, 42M)
    - Netburst microarchitecture with high clock rate but high power consumption
    - SSE2 instructions, new data types (eg. Double precision)
- Pentium 4E: (2004, 125Mtransistors).
    - Added *hyperthreading*
        - run two programs simultaneously on a single processor
    - EM64T, 64-bit extension to IA32
        - First developed by Advanced Micro Devices (AMD)
        - x86-64
- Core 2: (2006, 291Mtransistors)
    - back to a microarchitecture similar to P6
    - multi-core (multiple processors a single chip)
    - Did not support hyperthreading
- Core i7: (2008, 781 M transistors).
    - Incorporated both hyperthreading and multi-core
    - the initial version supporting two executing programs on

each core

- Core i7: (2011.11, 2.27B transistors)
    - 6 cores on each chip
    - 3.3G
    - 6*256 KB (L2), 15M (L3)
- Advanced Micro Devices (AMD)
    - At beginning,
        - lagged just behind Intel in technology,
        - produced less expensive and lower performance processors
- In 1999
    - First broke the 1-gigahertz clock-speed barrier
- In 2002
    - Introduced x86-64
    - The widely adopted 64-bit extension to IA32

Stack operation:

- Stack is a special kind of data structure. It can store objects of the same type
- The top of the stack must be explicitly specified. It is denoted as top
- There are two operations on the stack. push and pop
- There is a hardware stack in x86. its bottom has high address number. its top is indicated by %esp

Data Movement Example:

```
int exchange(int *xp, int y)    {
      int x = *xp ;
      *xp = y ;
      return x ;
}
```

```
pushl    %ebp
movl      %esp, %ebp
movl      8(%ebp),    %eax
movl      12(%ebp), %edx
movl      (%eax), %ecx
movl      %edx,        (%eax)
movl      %ecx,      %eax
movl      %ebp, %esp
popl       %ebp
```

Condition codes:

A set of single-bit. Maintained in a condition code register. Describe attributes of the most recently arithmetic or logical operation

(EFLAGS)

CF: Carry Flag. The most recent operation generated a carry out of the most significant bit. Used to detect overflow for unsigned operations

OF: Overflow Flag. The most recent operation caused a two's complement overflow — either negative or positive

ZF: Zero Flag. The most recent operation yielded zero

SF: Sign Flag. The most recent operation yielded a negative value

Jump Instructions:

```
1.      movl 8(%ebp), %edx          get x
2.      movl 12(%ebp), %eax         get y
3.      cmpl %eax, %edx             cal x - y
4.      jl .L3                      if   x < y goto less
5.      subl %eax, %edx             compute x - y
6.      movl %edx, %eax             set return val
7.      jmp .L5                     goto done
8.  .   L3:                         less:
9.      subl %edx, %eax             compute y – x
10..    L5:                         done: Begin Completion code
```

x86-64 Linux Register Usage:

%rax:

Return value; Also caller-saved; Can be modified by procedure

%rdi , ... , %r9

Arguments; Also caller-saved; Can be modified by procedure

%r10, %r11

Caller-saved;

Can be modified by procedure

%rbx, %r12, %r13, %r14

- Callee-saved
- Callee must save & restore

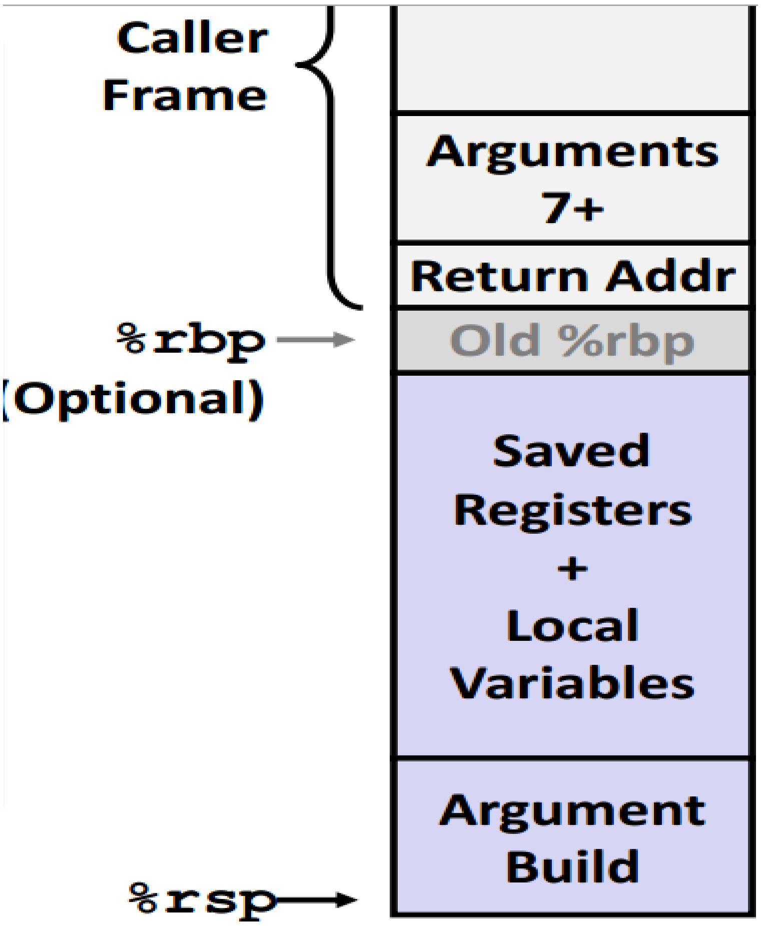
%rbp

- Callee-saved; Callee must save & restore
- May be used as frame pointer
- Can mix & match

%rsp

- Special form of callee save
- Restored to original value upon exit from procedure

x86-64 Procedure Summary:



Accessing Array：

Array elements can be accessed：Using an integer index ranging between 0 and N-1

Array element i is stored at address ： $X_A + \mathit{sizeof(T)} * i$

char a[12] ;

0	1	2	3	4	5	6	7	8	9	10	11

Xa Xa+4 Xa+8

char *b[5] ;

0	**4**	**8**	**12**	**16**

Xb Xb+4 Xb+8 Xb+12 Xb+16

double c[2] ;

0	**8**

Xc Xc+8

double *d[5] ;

0	**4**	**8**	**12**	**16**

Xd Xd+4 Xd+8 Xd+12 Xd+16

Pointer Arithmetic:

Expression	Type	Value	Assembly code
E	int *	x_E	movl %edx, %eax
E[0]	int	M[x_E]	movl (%edx), %eax
E[i]	int	M[x_E+4i]	movl (%edx, %ecx, 4), %eax
&E[2]	int *	x_E+8	leal 8(%edx,) %eax
E+i-1	int *	x_E+4i-4	lea -4(%edx, %ecx, 4), %eax
*(&E[i]+i)	int	M[x_E+4i+4i]	movl (%edx, %ecx, 8), %eax
&E[i]-E	int	i	movl %ecx, %eax

C operators:

Operators	**Associativity**
() [] -> . ++ --	left to right
! ~ ++ -- + - * & (type) sizeof	right to left
* / %	left to right
+ -	left to right
<< >>	left to right
< <= > >=	left to right
== !=	left to right
&	left to right
^	left to right
|	left to right
&&	left to right
||	left to right
?:	right to left
= += -= *= /= %= &= ^= != <<= >>=	right to left
,	left to right

Out-of-Bounds Memory References:

```
echo:
pushl    %ebp               Save %ebp on stack
movl     %esp, %ebp
pushl     %ebx              Save %ebx
subl        $20, %esp       Allocate 20 bytes on stack
leal        -12(%ebp), %ebx Compute buf as %ebp-12
movl        %ebx, (%esp)    Store buf at top of stack
call        gets            Call gets
movl        %ebx, (%esp)    Store buf at top of stack
call        puts            Call puts
addl        $20, %esp       Deallocate stack space
popl        %ebx            Restore %ebx
popl        %ebp            Restore %ebp
ret                              Return
```