

Machine Programming I: Basics

- **History of Intel processors and architectures**
 - Intel processors ([Wikipedia](#))
 - Intel [microarchitectures](#)
- **C, assembly, machine code**
- **Assembly basics: registers, operands, move instructions**

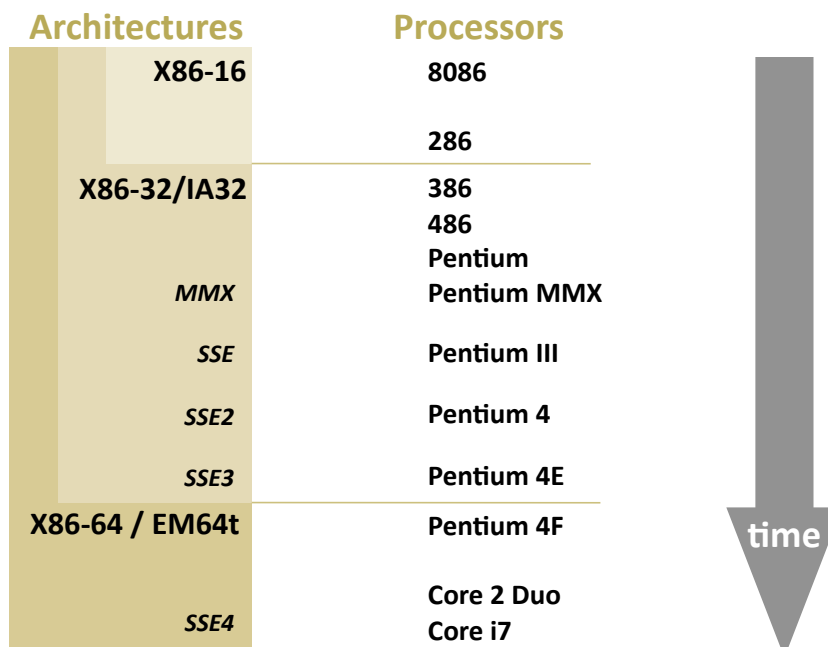
Intel x86 Processors

- **Totally dominate computer market**
- **Evolutionary design**
 - Backwards compatible up until 8086, introduced in 1978
 - Added more features as time goes on
- **Complex instruction set computer (CISC)**
 - Many different instructions with many different formats
 - But, only small subset encountered with Linux programs
 - Hard to match performance of Reduced Instruction Set Computers (RISC)
 - But, Intel has done just that!

Intel x86 Evolution: Milestones

<i>Name</i>	<i>Date</i>	<i>Transistors</i>	<i>MHz</i>
■ 8086	1978	29K	5-10
<ul style="list-style-type: none"> ▪ First 16-bit processor. Basis for IBM PC & DOS ▪ 1MB address space 			
■ 386	1985	275K	16-33
<ul style="list-style-type: none"> ▪ First 32 bit processor , referred to as IA32 ▪ Added “flat addressing” ▪ Capable of running Unix ▪ 32-bit Linux/gcc uses no instructions introduced in later models 			
■ Pentium 4F	2005	230M	2800-3800
<ul style="list-style-type: none"> ▪ First 64-bit processor ▪ Meanwhile, Pentium 4s (Netburst arch.) phased out in favor of “Core” line 			

Intel x86 Processors: Overview

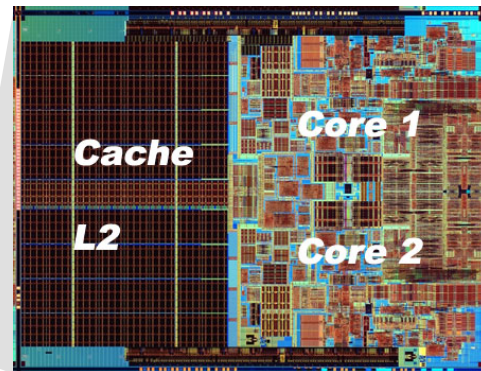


IA: often redefined as latest Intel architecture

Intel x86 Processors, contd.

■ Machine Evolution

■ 486	1989	1.9M
■ Pentium	1993	3.1M
■ Pentium/MMX	1997	4.5M
■ PentiumPro	1995	6.5M
■ Pentium III	1999	8.2M
■ Pentium 4	2001	42M
■ Core 2 Duo	2006	291M



■ Added Features

- Instructions to support multimedia operations
 - Parallel operations on 1, 2, and 4-byte data, both integer & FP
- Instructions to enable more efficient conditional operations

■ Linux/GCC Evolution

- Very limited

New Species: ia64, then IPF, then Itanium,...

<i>Name</i>	<i>Date</i>	<i>Transistors</i>
■ Itanium	2001	10M
<ul style="list-style-type: none"> ■ First shot at 64-bit architecture: first called IA64 ■ Radically new instruction set designed for high performance ■ Can run existing IA32 programs <ul style="list-style-type: none"> ▪ On-board “x86 engine” ■ Joint project with Hewlett-Packard 		
■ Itanium 2	2002	221M
<ul style="list-style-type: none"> ■ Big performance boost 		
■ Itanium 2 Dual-Core	2006	1.7B
■ Itanium has not taken off in marketplace		
<ul style="list-style-type: none"> ■ Lack of backward compatibility, no good compiler support, Pentium 4 got too good 		

x86 Clones: Advanced Micro Devices (AMD)

■ Historically

- AMD has followed just behind Intel
- A little bit slower, a lot cheaper

■ Then

- Recruited top circuit designers from Digital Equipment and other downward trending companies
- Built Opteron: tough competitor to Pentium 4
- Developed x86-64, their own extension to 64 bits

■ Recently

- Intel much quicker with dual core design
- Intel currently far ahead in performance
- em64t backwards compatible to x86-64

Intel's 64-Bit

■ Intel Attempted Radical Shift from IA32 to IA64

- Totally different architecture (Itanium)
- Executes IA32 code only as legacy
- Performance disappointing

■ AMD Stepped in with Evolutionary Solution

- x86-64 (now called "AMD64")

■ Intel Felt Obligated to Focus on IA64

- Hard to admit mistake or that AMD is better

■ 2004: Intel Announces EM64T extension to IA32

- Extended Memory 64-bit Technology
- Almost identical to x86-64!

■ Meanwhile: EM64t well introduced, however, still often not used by OS, programs

Our Coverage

- **IA32**
 - The traditional x86

- **x86-64/EM64T**
 - The emerging standard – we'll just touch on its major additions

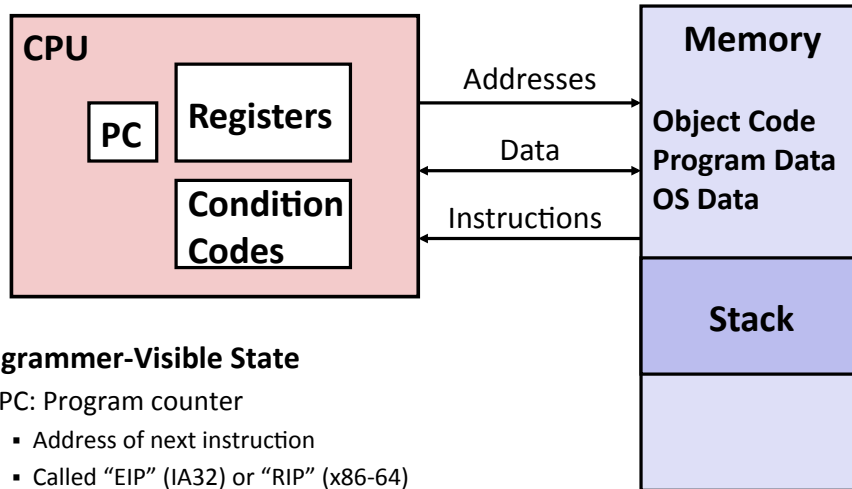
Definitions

- **Architecture:** (also instruction set architecture or ISA)
The parts of a processor design that one needs to understand to write assembly code
- **Microarchitecture:** Implementation of the architecture

- **Architecture examples:** instruction set specification, registers
- **Microarchitecture examples:** cache sizes and core frequency

- **Example ISAs (Intel):** x86, IA-32, IPF

Assembly Programmer's View



■ Programmer-Visible State

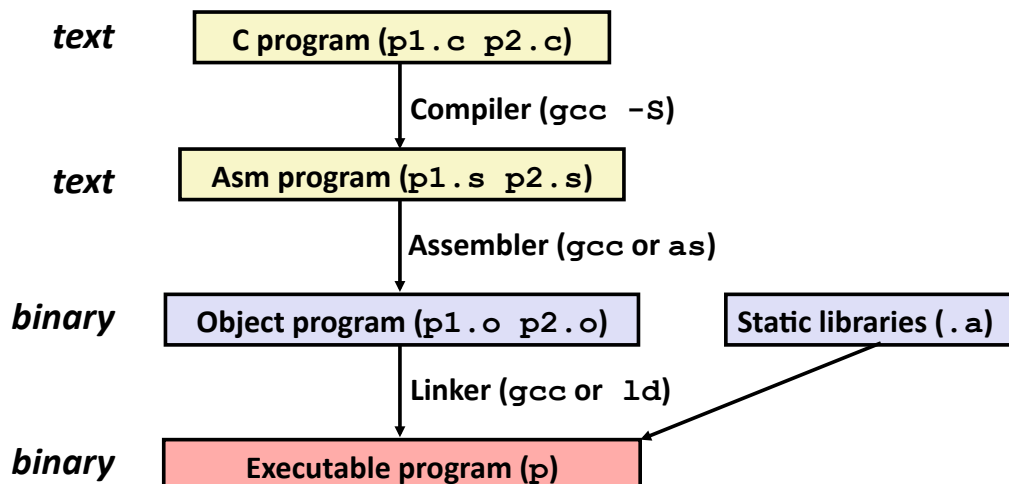
- PC: Program counter
 - Address of next instruction
 - Called "EIP" (IA32) or "RIP" (x86-64)
- Register file
 - Heavily used program data
- Condition codes
 - Store status information about most recent arithmetic operation
 - Used for conditional branching

■ Memory

- Byte addressable array
- Code, user data, (some) OS data
- Includes stack used to support procedures (we'll come back to that)

Turning C into Object Code

- Code in files `p1.c p2.c`
- Compile with command: `gcc -O p1.c p2.c -o p`
 - Use optimizations (`-O`)
 - Put resulting binary in file `p`



Compiling Into Assembly

C Code

```
int sum(int x, int y)
{
    int t = x+y;
    return t;
}
```

Generated IA32 Assembly

```
sum:
    pushl %ebp
    movl %esp,%ebp
    movl 12(%ebp),%eax
    addl 8(%ebp),%eax
    movl %ebp,%esp
    popl %ebp
    ret
```

Obtain with command

```
gcc -O -S code.c
```

Produces file code.s

Some compilers use single instruction "leave"

Assembly Characteristics: Data Types

- "Integer" data of 1, 2, or 4 bytes
 - Data values
 - Addresses (untyped pointers)
- Floating point data of 4, 8, or 10 bytes
- No aggregate types such as arrays or structures
 - Just contiguously allocated bytes in memory

Assembly Characteristics: Operations

- Perform arithmetic function on register or memory data
- Transfer data between memory and register
 - Load data from memory into register
 - Store register data into memory
- Transfer control
 - Unconditional jumps to/from procedures
 - Conditional branches

Object Code

Code for sum

```
0x401040 <sum>:
  0x55
  0x89
  0xe5
  0x8b
  0x45
  0x0c
  0x03
  0x45
  0x08
  0x89
  0xec
  0x5d
  0xc3
```

- Total of 13 bytes
- Each instruction 1, 2, or 3 bytes
- Starts at address 0x401040

■ Assembler

- Translates `.s` into `.o`
- Binary encoding of each instruction
- Nearly-complete image of executable code
- Missing linkages between code in different files

■ Linker

- Resolves references between files
- Combines with static run-time libraries
 - E.g., code for `malloc`, `printf`
- Some libraries are *dynamically linked*
 - Linking occurs when program begins execution

Machine Instruction Example

```
int t = x+y;
```

```
addl 8(%ebp),%eax
```

Similar to expression:

```
x += y
```

More precisely:

```
int eax;
int *ebp;
eax += ebp[2]
```

```
0x401046: 03 45 08
```

■ C Code

- Add two signed integers

■ Assembly

- Add 2 4-byte integers
 - “Long” words in GCC parlance
 - Same instruction whether signed or unsigned
- Operands:
 - x:** Register `%eax`
 - y:** Memory `M[%ebp+8]`
 - t:** Register `%eax`
- Return function value in `%eax`

■ Object Code

- 3-byte instruction
- Stored at address `0x401046`

Disassembling Object Code

Disassembled

```
00401040 <_sum>:
 0:      55                push  %ebp
 1:      89 e5             mov   %esp,%ebp
 3:      8b 45 0c         mov   0xc(%ebp),%eax
 6:      03 45 08         add  0x8(%ebp),%eax
 9:      89 ec             mov   %ebp,%esp
 b:      5d                pop   %ebp
 c:      c3                ret
 d:      8d 76 00         lea  0x0(%esi),%esi
```

■ Disassembler

```
objdump -d p
```

- Useful tool for examining object code
- Analyzes bit pattern of series of instructions
- Produces approximate rendition of assembly code
- Can be run on either a `.out` (complete executable) or `.o` file

Alternate Disassembly

Object	Disassembled
0x401040:	0x401040 <sum>: push %ebp
0x55	0x401041 <sum+1>: mov %esp,%ebp
0x89	0x401043 <sum+3>: mov 0xc(%ebp),%eax
0xe5	0x401046 <sum+6>: add 0x8(%ebp),%eax
0x8b	0x401049 <sum+9>: mov %ebp,%esp
0x45	0x40104b <sum+11>: pop %ebp
0x0c	0x40104c <sum+12>: ret
0x03	0x40104d <sum+13>: lea 0x0(%esi),%esi
0x45	
0x08	
0x89	
0xec	
0x5d	
0xc3	

■ Within gdb Debugger

```
gdb p
```

```
disassemble sum
```

- Disassemble procedure

```
x/13b sum
```

- Examine the 13 bytes starting at sum

What Can be Disassembled?

```
% objdump -d WINWORD.EXE

WINWORD.EXE: file format pei-i386

No symbols in "WINWORD.EXE".
Disassembly of section .text:

30001000 <.text>:
30001000: 55          push    %ebp
30001001: 8b ec      mov    %esp,%ebp
30001003: 6a ff      push   $0xffffffff
30001005: 68 90 10 00 30 push  $0x30001090
3000100a: 68 91 dc 4c 30 push  $0x304cdc91
```

- Anything that can be interpreted as executable code
- Disassembler examines bytes and reconstructs assembly source

Integer Registers (IA32)

