System Architecture and Assembly

Systems Programming
(CST-210)

A. P. MAZUMDAR

Intel x86 Processors

- Totally dominate laptop/desktop/server market
- Evolutionary design
  - Backwards compatible up until 8086, introduced in 1978
  - Added more features as time goes on

Intel x86 Processors

- 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!
    - In terms of speed. Less so for low power.

Intel x86 Evolution: Milestones

Name	*Date*	*Transistors*	*MHz*
8086	**1978**	**29K**	**5-10**

- First 16-bit processor. Basis for IBM PC & DOS
- 1MB address space

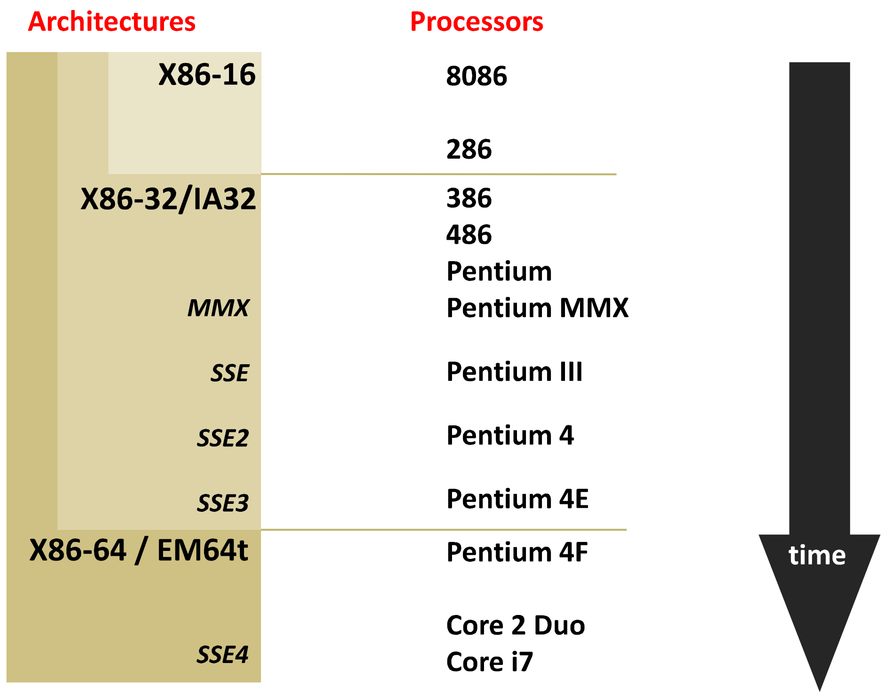
Name	*Date*	*Transistors*	*MHz*
386	**1985**	**275K**	**16-33**

- 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

Intel x86 Evolution: Milestones

Name	Date	Transistors	MHz
Pentium 4F	2004	125M	2800-3800
Core i7	2008	731M	2667-3333

- **Pentium 4F**
  - First 64-bit processor, referred to as x86-64
- **Core i7**
  - New machines



IA: often redefined as latest Intel architecture

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 Corp. and other downward trending companies
  - Built Opteron: tough competitor to Pentium 4
  - Developed x86-64, their own extension to 64 bits

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

Intel's 64-Bit

- 2004: Intel Announces EM64T extension to IA32
  - Extended Memory 64-bit Technology
  - Almost identical to x86-64!
- All but low-end x86 processors support x86-64
  - But, lots of code still runs in 32-bit mode

IA32 (Pentium) Processor Architecture

- **32 bit Processor**
- **1 WORD = 16bit**
- **32 bits = 2 WORDS = 1 Double Word   (DWORD)**

Processor modes

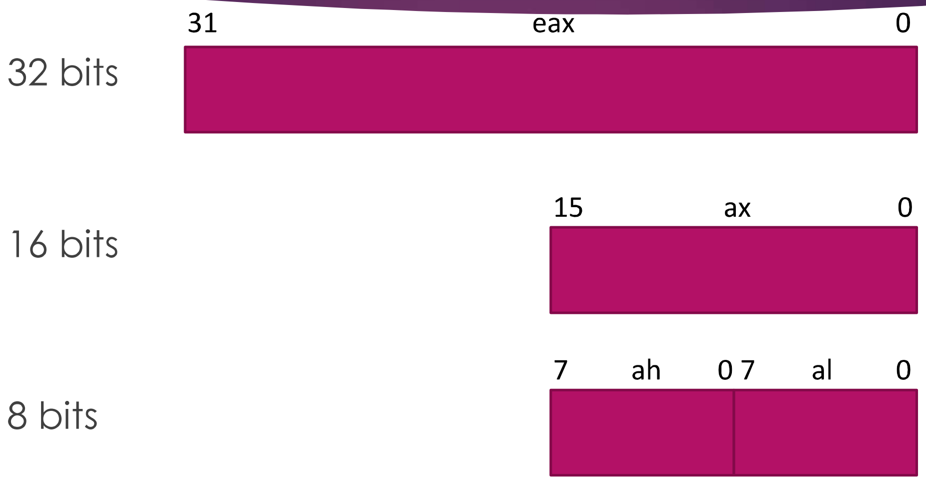
1. Protected (*important*)
   - 32-bit mode
   - 32-bit (4GB) address space
2. Virtual 8086 modes
3. Real mode
   - 1MB address space
4. System management mode

Registers

- 32-bit GPR's ("general" purpose registers):

eax	**ebp**
ebx	**esp**
ecx	**esi**
edx	**edi**
eflags	**eip**

e[a,b,c,d]x:



Note: eax is **one** register that can be viewed **four** different ways.

Registers

- Not really GPR’s.
  - eax - accumulator; multiplication and division
  - ecx - loop counter
  - esp - stack pointers; don’t use
  - esi, edi - for memory-to-memory transfer
  - ebp - used by HLL for local vars on stack

Registers

- Additional registers:
  - 16-bit segment registers
    - cs, es, ss, fs, ds, gs
    - don’t use
  - eip
    - instruction pointer / program counter (PC)
    - don‘t use

Registers

- CS
  - Address of current code segment
- SS
  - Address of current stack segment
- Others (DS, ES, FS, GS)
  - Address of data segments

Registers

- Additional registers:
  - eflags
    - contains results of operations
    - 32 individual bits
      - control flags
      - status flags:
        - **C = carry (unsigned)**
        - **O = overflow (signed); also called V**
        - **S = sign; also called N for negative**
        - **Z = zero**

Registers

- Additional registers:
  - floating point registers:
    - ST(0) … ST(7)
      - 80 bits
  - MMX has 8 64-bit regs
    - Translate segment address to Physical address
  - XMM has 8 128-bit regs

Compilation

- Two parts of a program: **p1.c and p2.c**
- To compile:
  - gcc -O1 -o p p1.c p2.c
- -o
  - Output file name, followed by the <name>
- -O1
  - Level of optimization: (higher level = faster execution, slower compilation)

Compilation

- ASM code generation:
  - ***gcc -O1 -S code.c***
- Output file code.s will be generated
- -S : gcc option to compile till assembly level

```c
int accum = 0;

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

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

Creating Object code

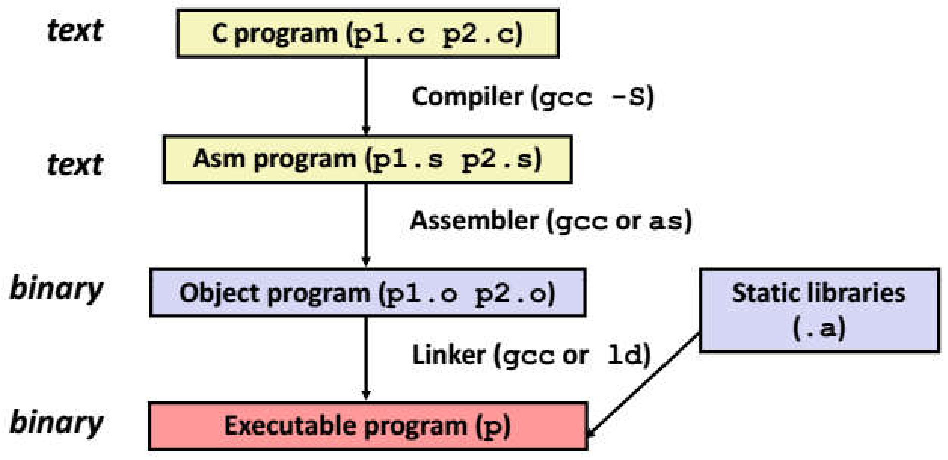
- **gcc –O1 -c code.c**
  - **-c:**   option for generating obj code
- For code.c   :

```
55 89 e5 8b 45 0c 03 45 08 01 05 00 00 00 00 5d c3
```

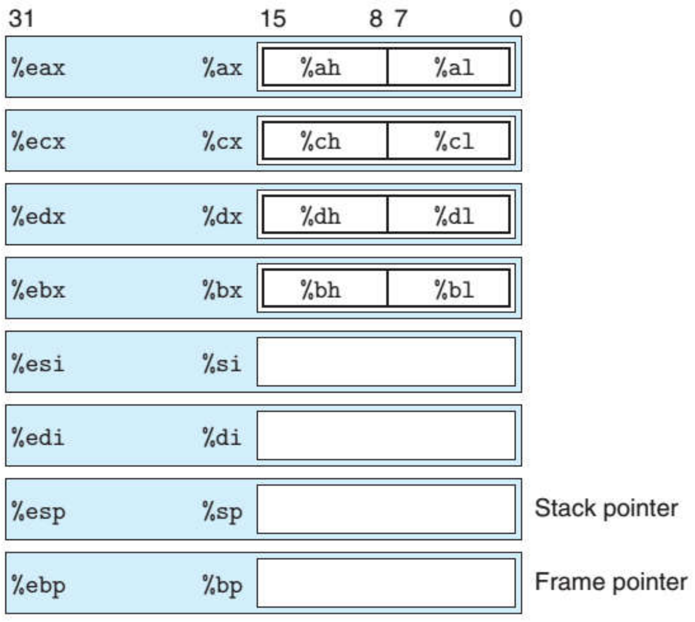
Generating Object code

- **gcc –O1 -c code.c**
- Use Disassembler to byte code length
  - In this case it is 17
- Use GNU debugging toll (GDB) on **code.o** to get the code
  - **(gdb) x/17xb sum**

Summary of compilation



Revisit IA32 Integer Registers


31
15
8 7
0
%eax
%ax
%ah
%al
%ecx
%cx
%ch
%cl
%edx
%dx
%dh
%dl
%ebx
%bx
%bh
%bl
%esi
%si
%edi
%di
%esp
%sp
Stack pointer
%ebp
%bp
Frame pointer