PC Hardware and x86

Dr. Jun Zheng

CSE325 Principles of Operating Systems

8/23/2019

A PC

- How to make it do something userful?

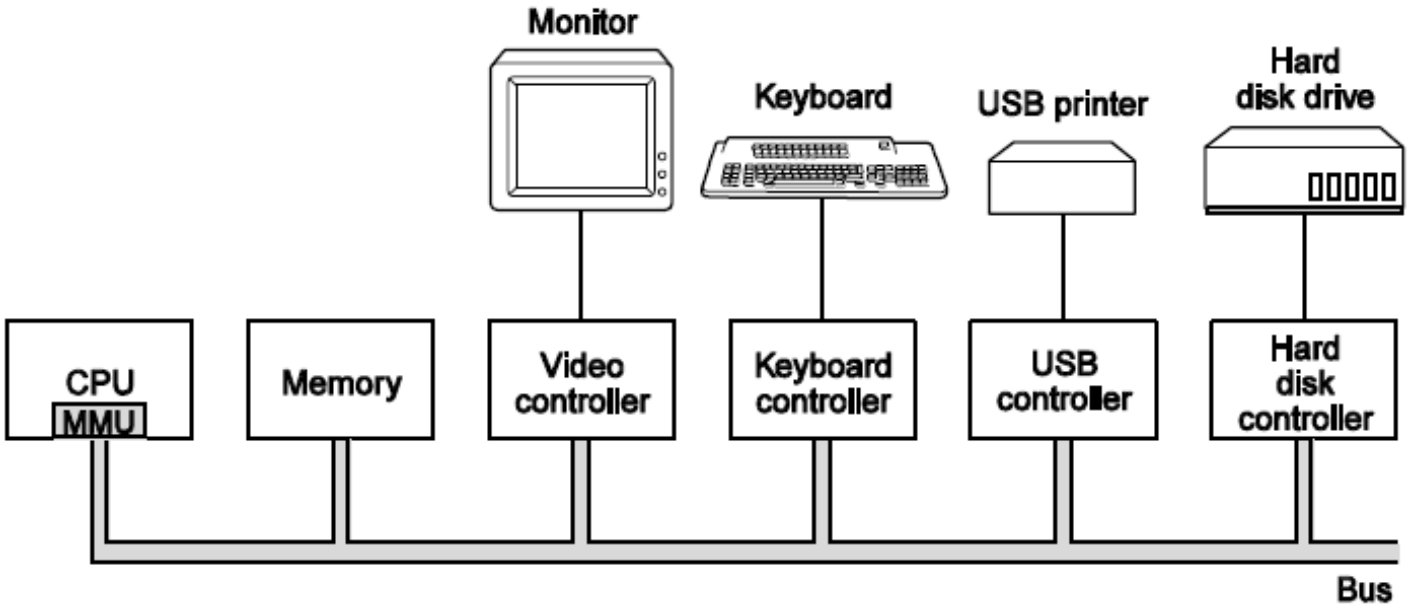
NEW MEXICO TECH
SCIENCE · ENGINEERING · RESEARCH · UNIVERSITY

Outline

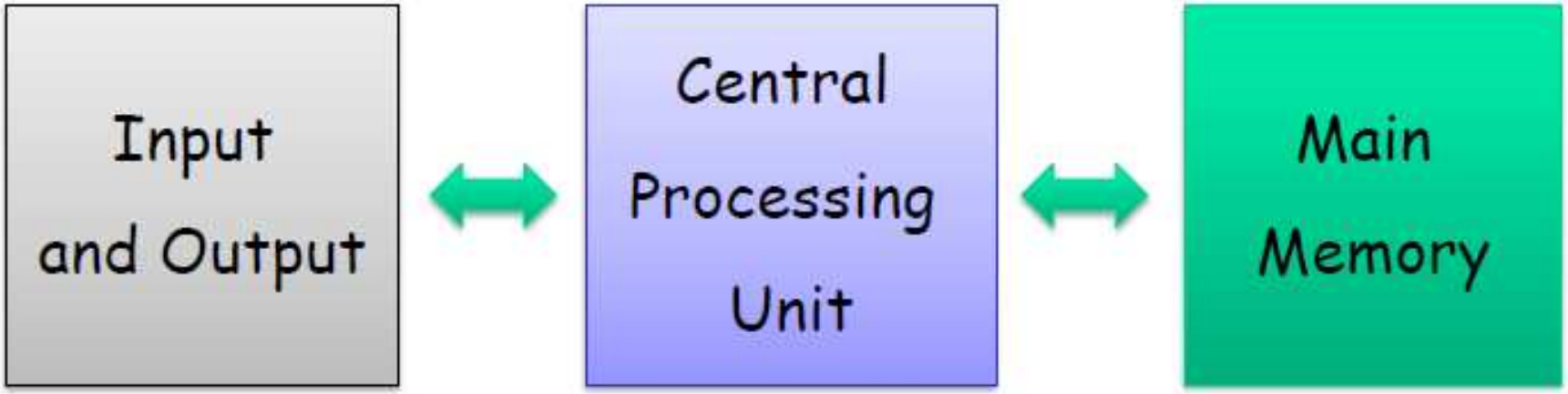
- PC architecture
- x86 Instruction set
- gcc calling conventions

PC Organization

- One or more CPUs, memory, and device controllers connect through system bus

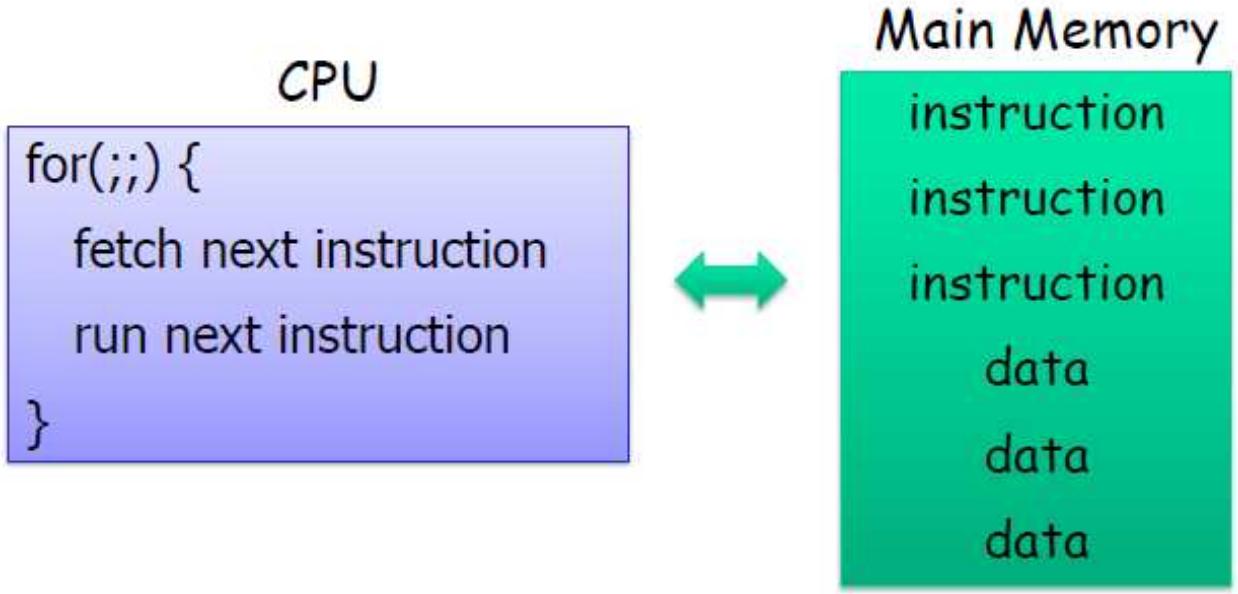


Abstract Model



- I/O: communicating data to and from devices
- CPU: digital logic for doing computation
- Memory: *N* words of *B* bits

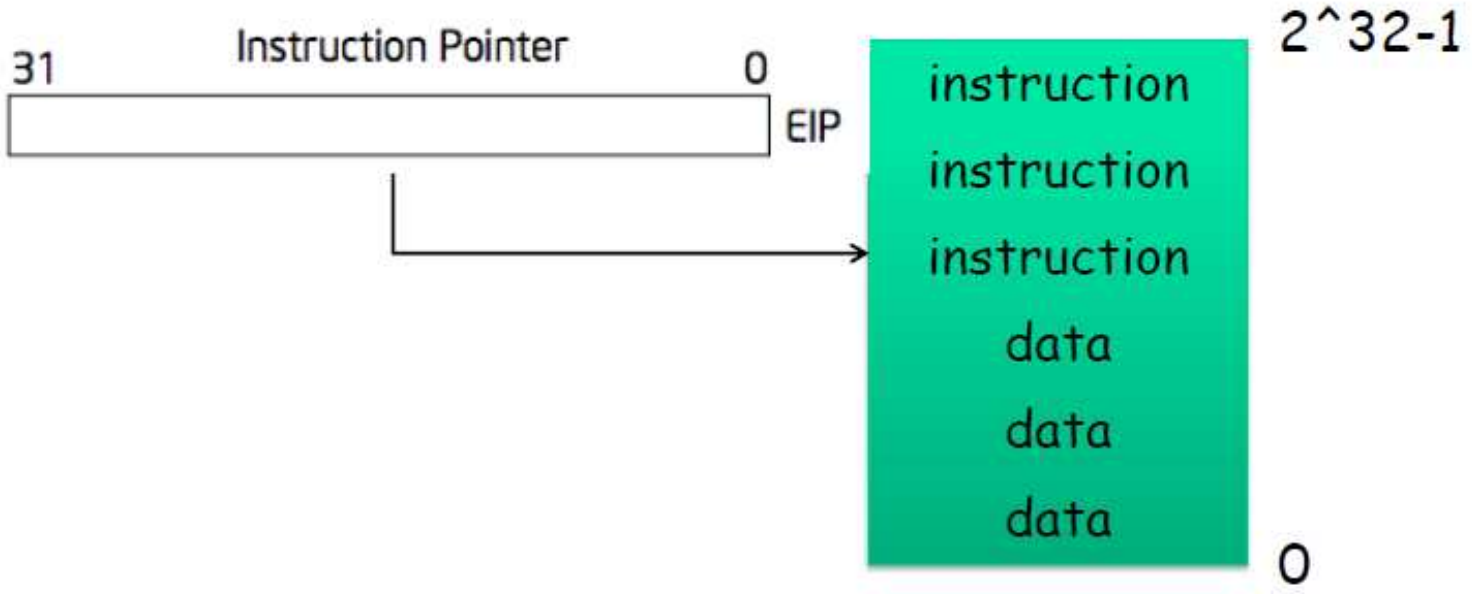
The Stored Prgram Computer



- Memory holds both instructions and data
- CPU interprets instructions
- Instructions read/write data

NEW MEXICO TECH
SCIENCE · ENGINEERING · RESEARCH · UNIVERSITY

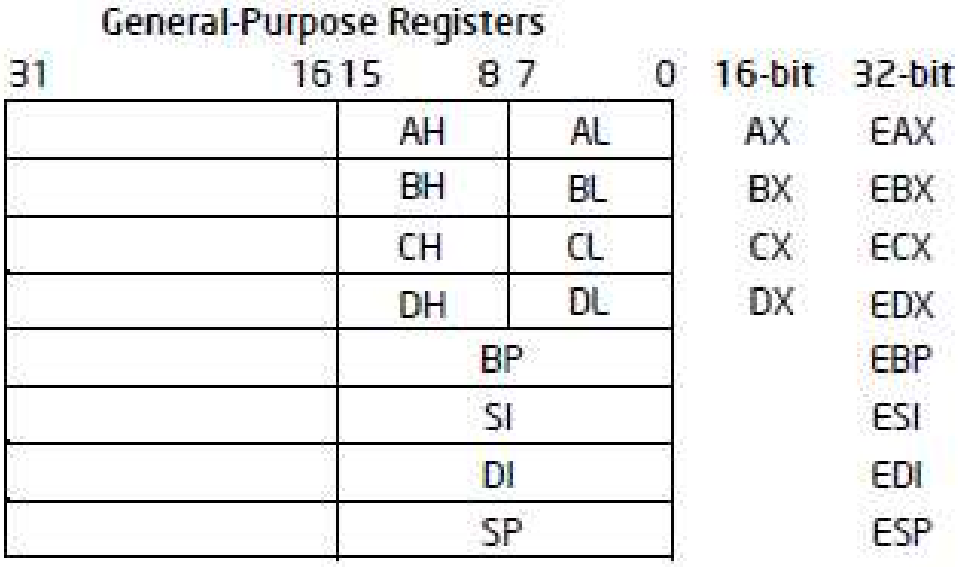
x86 Implementation



- `EIP` incremented after each instruction
- Variable length instructions
- `EIP` modified by `CALL`, `RET`, `JMP`, `conditional JMP`

NEW MEXICO TECH
SCIENCE · ENGINEERING · RESEARCH · UNIVERSITY

Registers: Work Space



ESP: stack pointer

EBP: frame base pointer

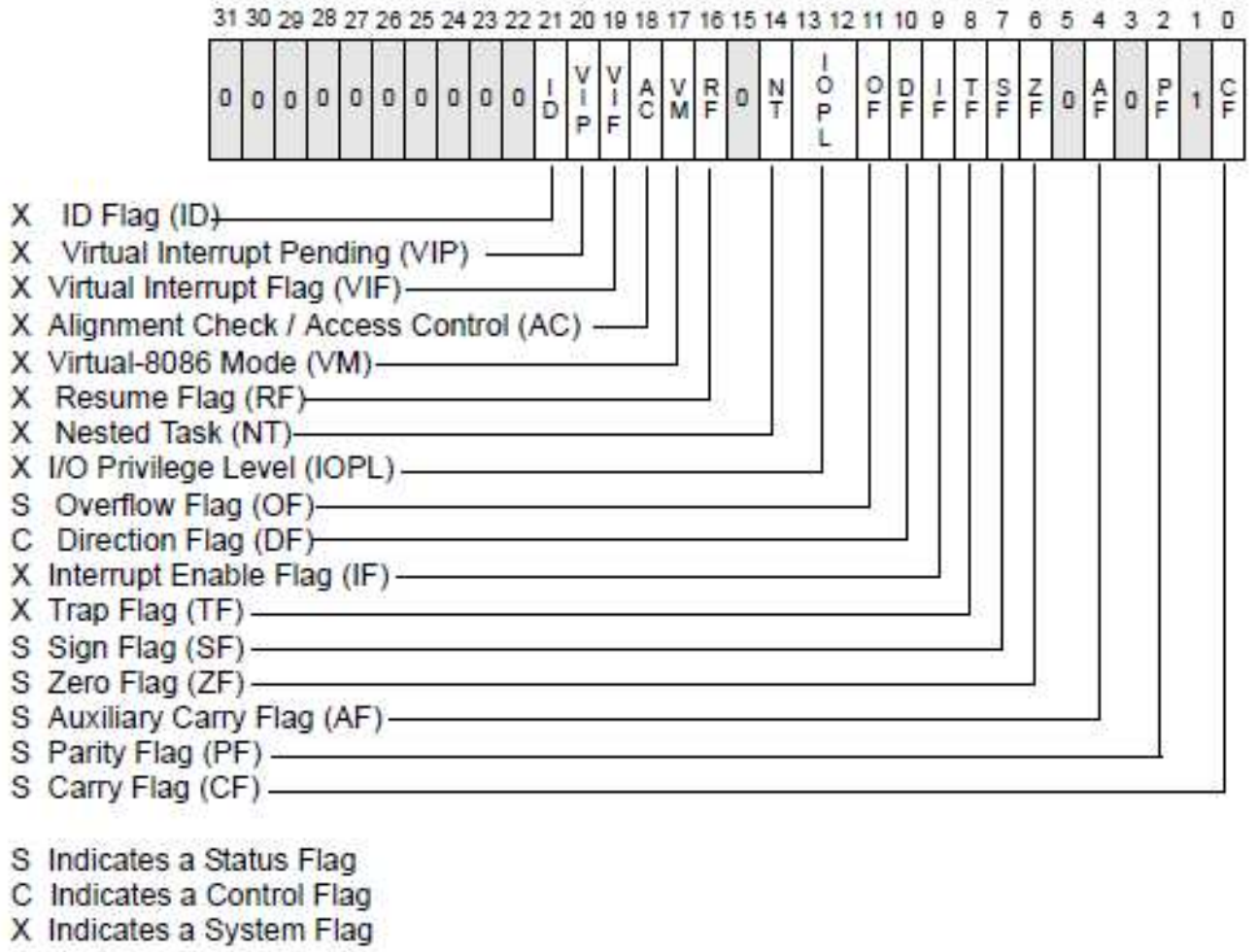
ESI: source index

EDI: destination index

- 8, 16, and 32 bit versions
- Example: `ADD EAX, 10`
  - More: `SUB`, `AND`, etc.
- By convention some for special purposes

NEW MEXICO TECH
SCIENCE · ENGINEERING · RESEARCH · UNIVERSITY

EFLAGS Register



- Track current CPU status

NEW MEXICO TECH
SCIENCE · ENGINEERING · RESEARCH · UNIVERSITY

x86 Instructon Set

- Instruction syntax
  - Intel manual Volumne 2: `op dst, src`
  - AT&T (gcc/gas): `op src, dst`
    - `op` uses suffix b, w, l for 8, 16, 32-bit operands
    - xv6, JOS
- Instruction classes
  - Data movement: `MOV, PUSH, POP`, ...
  - Arithmetic: `TEST, SHL, ADD, AND`, ...
  - I/O: `IN, OUT`, ...
  - Control: `JMP, JZ, JNZ, CALL, RET`
  - String: `MOVSB, REP`, ...
  - System: `INT, IRET`

NEW MEXICO TECH
SCIENCE · ENGINEERING · RESEARCH · UNIVERSITY

Memory: Addessing Examples

Address	Value	Operand	Value	Comment
0x100	0xFF	%eax	0x100	Value in the register
0x104	0xAB	0x104	0xAB	Vaue is at the address
0x108	0x13	$0x108	0x108	Value is the value ($ means immediate, i.e. constant, value)
0x10C	0x11	(%eax)	0xFF	value is at the address stored in the register -> GTV@(reg)
		4(%eax)	0xAB	GTV@(4 + reg)
Register	**Value**	9(%eax, %edx)	0x11	GTV@(9 + reg + reg)
%eax	0x100	0xFC(,%ecx, 4)	0xFF	GTV@(0xFC + 0 + reg*4)
%ecx	0x1	(%eax, %edx, 4)	0x11	GTV@(reg + reg*4)
%edx	0x3			

- Red means memory adress

NEW MEXICO TECH
SCIENCE · ENGINEERING · RESEARCH · UNIVERSITY

Memory: Addressing Modes

movl %eax, %edx	edx = eax;	*register mode*
movl $0x123, %edx	edx = 0x123;	*immediate*
movl 0x123, %edx	edx = *(int32_t*)0x123;	*direct*
movl (%ebx), %edx	edx = *(int32_t*)ebx;	*indirect*
movl 4(%ebx), %edx	edx = *(int32_t*)(ebx+4);	*displaced*

- Memory instructions: MOV, PUSH, POP, etc.
- More instructions can take a memory address

NEW MEXICO TECH
SCIENCE · ENGINEERING · RESEARCH · UNIVERSITY

Stack Memory + Operations

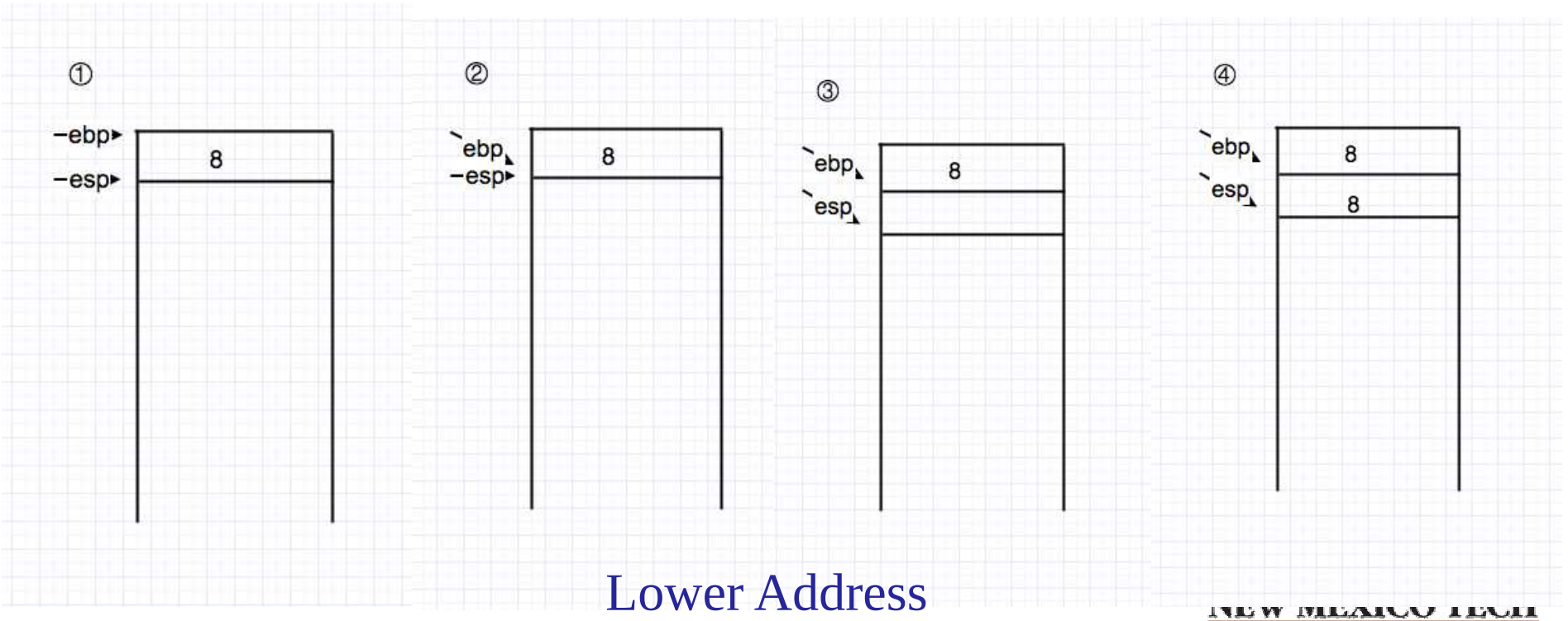
Example instruction	What it does
pushl %eax	subl $4, %esp movl %eax, (%esp)
popl %eax	movl (%esp), %eax addl $4, %esp
call 0x12345	pushl %eip (*) movl $0x12345, %eip (*)
ret	popl %eip (*)

- For implementing function calls
- Stack grows "down" on x86

NEW MEXICO TECH
SCIENCE · ENGINEERING · RESEARCH · UNIVERSITY

Stack Operation Example

```
pushl $8          ①
movl %esp, %ebp   ②
subl $4, %esp     ③
movl $8, (%esp)   ④
```



More Memory

- 8086 16-bit register and 20-bit bus addresses
- These extra 4 bits come from segment register
  - `CS`: code segment, for EIP
    - Instruction address: `CS * 16 + EIP`
  - `SS`: stack segment, for `ESP` and `EBP`
  - `DS`: data segment for load/store via other registers
  - `ES`: another data segment, destination for string ops
- Make life more complicated
  - Cannot directly use 16-bit stack address as pointer
  - For a far pointer programmer must specify segment reg
  - Pointer arithmetic and array indexing across seg bound

And More Memory

- 80386: 32 bit register and addresses (1985)
- AMD k8: 64 bit (2003)
  - `RAX` instead of `EAX`
  - x86-64, x64, amd64, intel64: all same thing
- Backward compatibility
  - Boots in 16-bit mode; `bootasm.S` switches to 32
  - Prefix `0x66` gets 32-bit mode instructions
    - `MOVW` in 32-bit mode = `0x66` + `MOVW` in 16-bit mode
  - `.code32` in `bootasm.S` tells assembler to insert `0x66`
- 80386 also added virtual memory addresses

I/O Space and Instructions

```
#define DATA_PORT     0x378
#define STATUS_PORT   0x379
#define    BUSY 0x80
#define CONTROL_PORT 0x37A
#define    STROBE 0x01
void
lpt_putc(int c)
{
  /* wait for printer to consume previous byte */
  while((inb(STATUS_PORT) & BUSY) == 0)
    ;

  /* put the byte on the parallel lines */
  outb(DATA_PORT, c);

  /* tell the printer to look at the data */
  outb(CONTROL_PORT, STROBE);
  outb(CONTROL_PORT, 0);
}
```

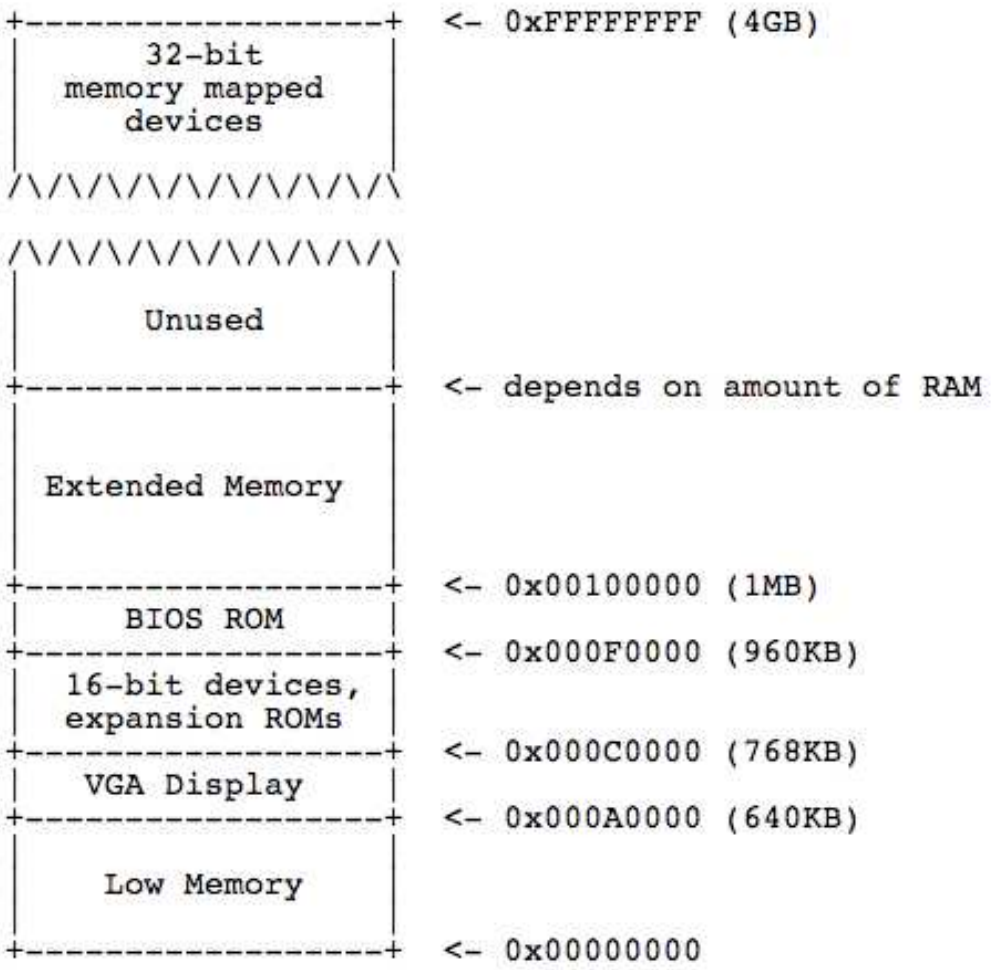
- 8086: only 1024 I/O addresses

NEW MEXICO TECH
SCIENCE · ENGINEERING · RESEARCH · UNIVERSITY

Memory-mapped I/O

- Use normal addresses for I/O
  - No special instructions
  - No 1024 limit
  - Hardware routes to appropriate device
- Works like “magic” memory
  - I/O device addressed and accessed like memory
  - However, reads and writes have “side effects”
  - Read result can change due to external events

Memory Layout



gcc Inline Assembly

- Can embed assembly code in C code
  - Many examples in xv6
- Basic syntax: asm ("assembly code")
  - e.g., `asm ("movl %eax, %ebx")`
- Advanced syntax:

```
asm ( assembler template
      : output operands /* optional */
      : input operands /* optional */
      : list of clobbered registers /* optional */ );
```

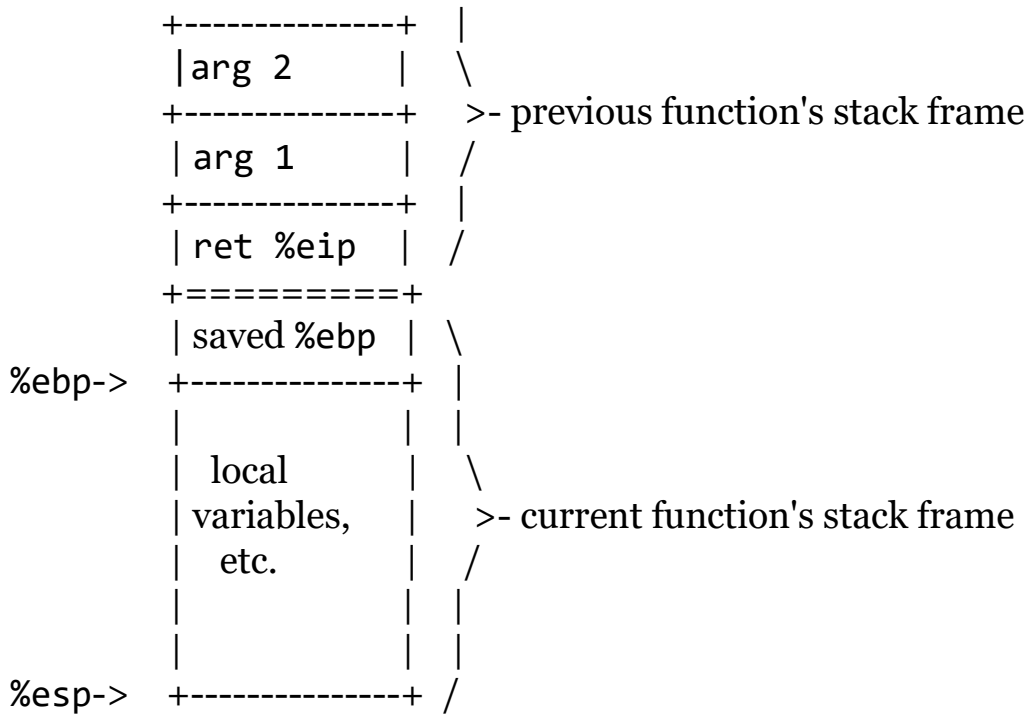
e.g., `int val;`

```
asm ("movl %%ebp,%0" : "=r" (val));
```

NEW MEXICO TECH
SCIENCE · ENGINEERING · RESEARCH · UNIVERSITY

gcc Calling Conventions

- Functions can do anything that doesn't violate contract. By convention, GCC does more:
  - each function has a stack frame marked by `%ebp, %esp`
  - `%esp` can move to make stack frame bigger, smaller
  - `%ebp` points at saved `%ebp` from previous function, chain to walk stack



NEW MEXICO TECH
SCIENCE · ENGINEERING · RESEARCH · UNIVERSITY

gcc Calling Conventions (cont.)

- `%eax` contains return value, `%ecx`, `%edx` may be trashed
- 64 bit return value: `%eax + %edx`
- `%ebp`, `%ebx`, `%esi`, `%edi` must be as before call
- Caller saved: `%eax, %ecx, %edx`
- Callee saved: `%ebp, %ebx, %esi, %edi`