Operating Systems

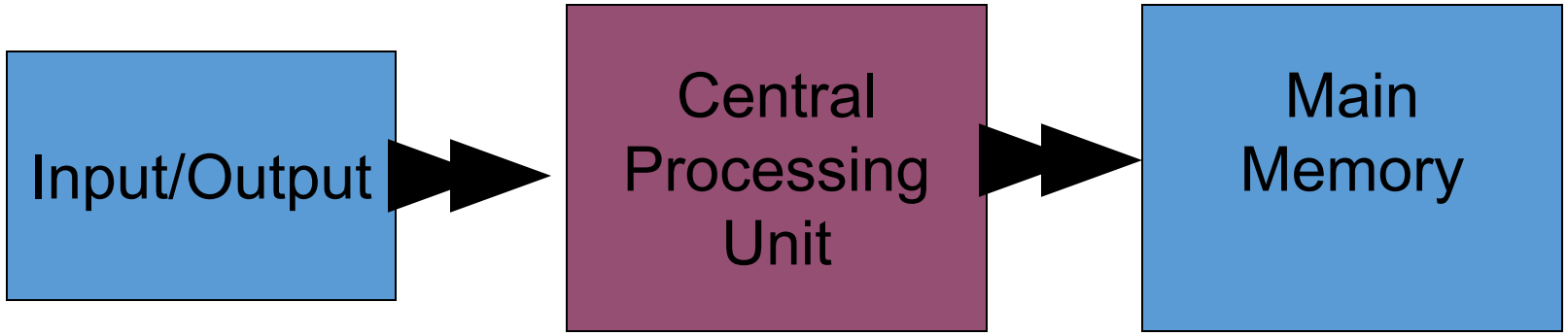
CSE 231 (Section B)

Instructor: Sambuddho Chakravarty

(Semester: Monsoon 2020)

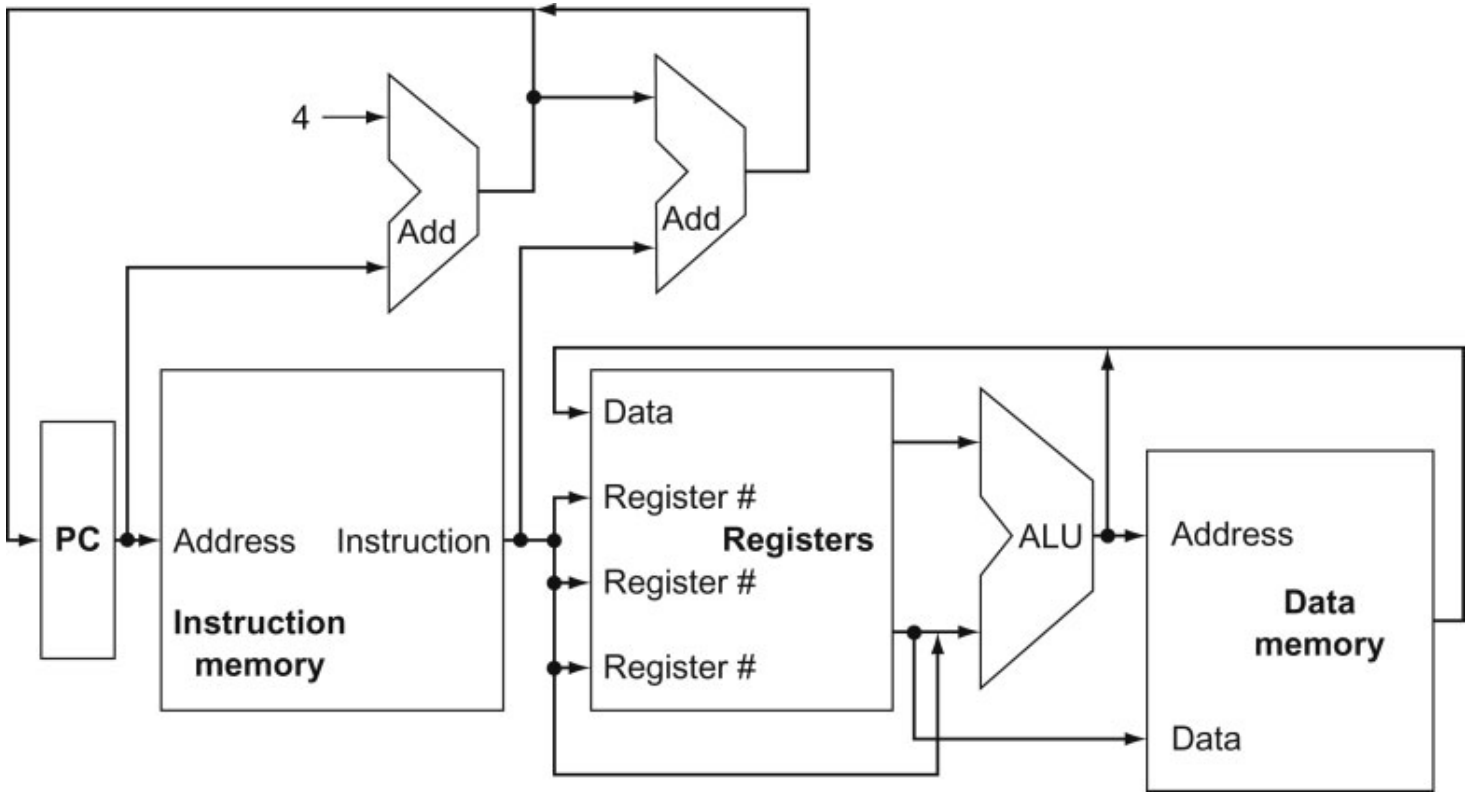
Week 2: Aug 31 – Sept. 6

Abstract model

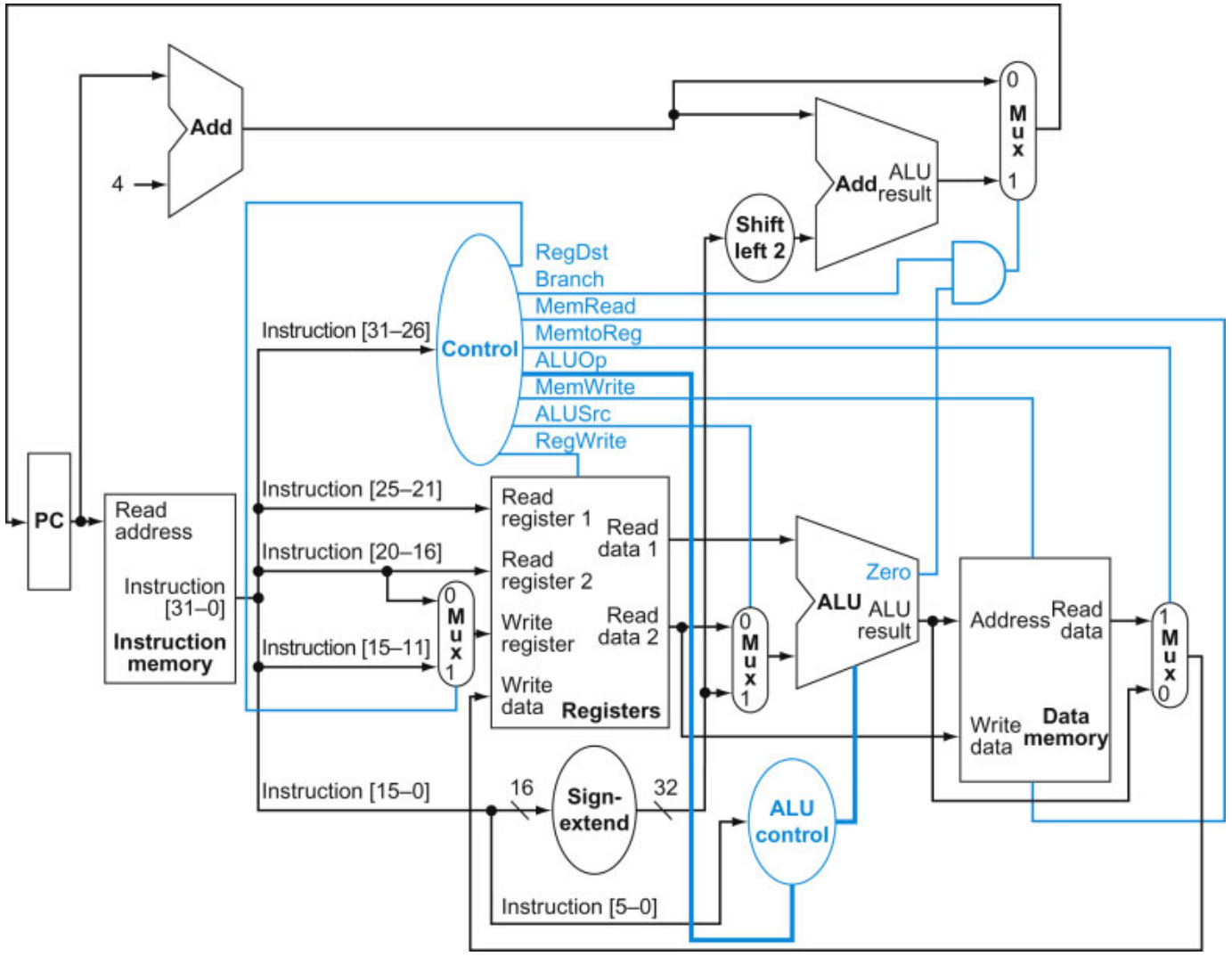


- I/O: communicating data to and from devices
- CPU: digital logic for performing computation
- Memory: *N* words of *B* bits

View from 30,000 Feet

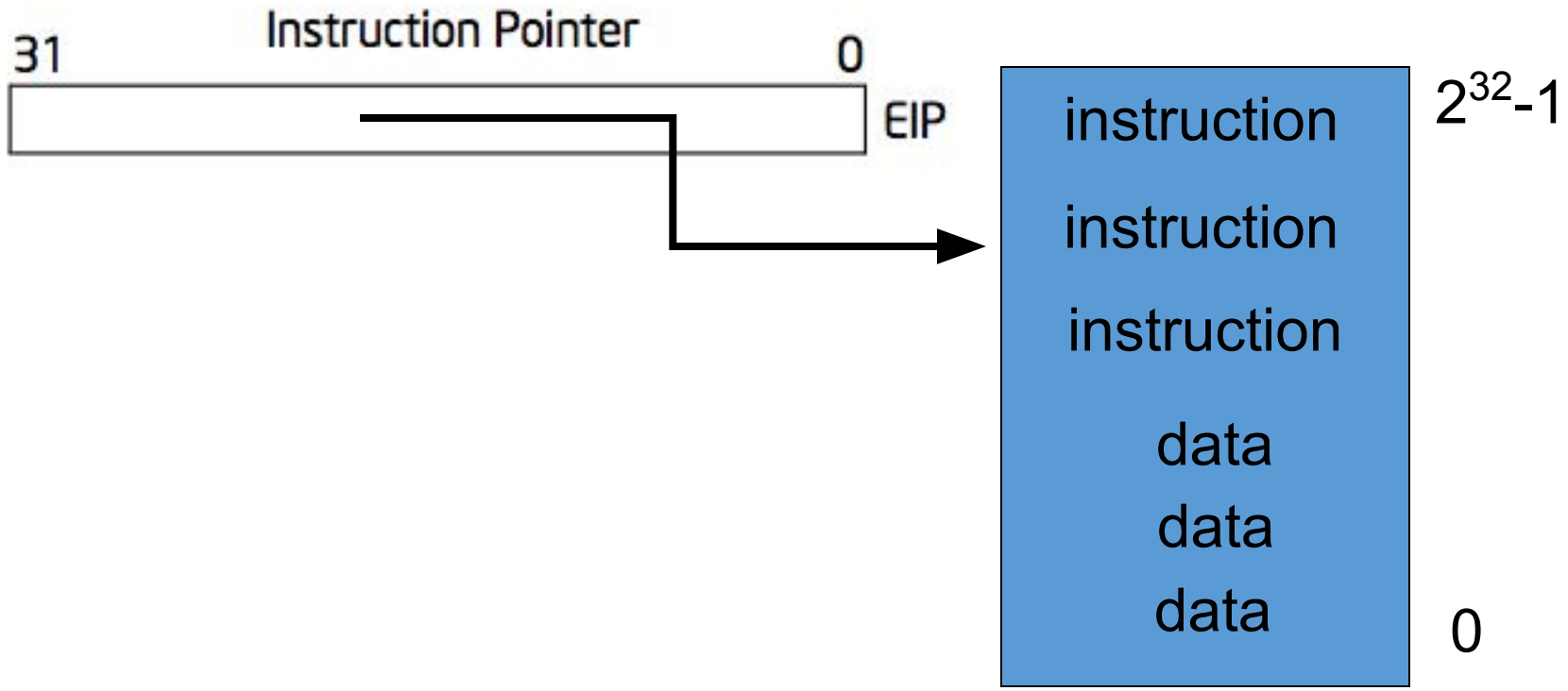


View from 5,000 Feet



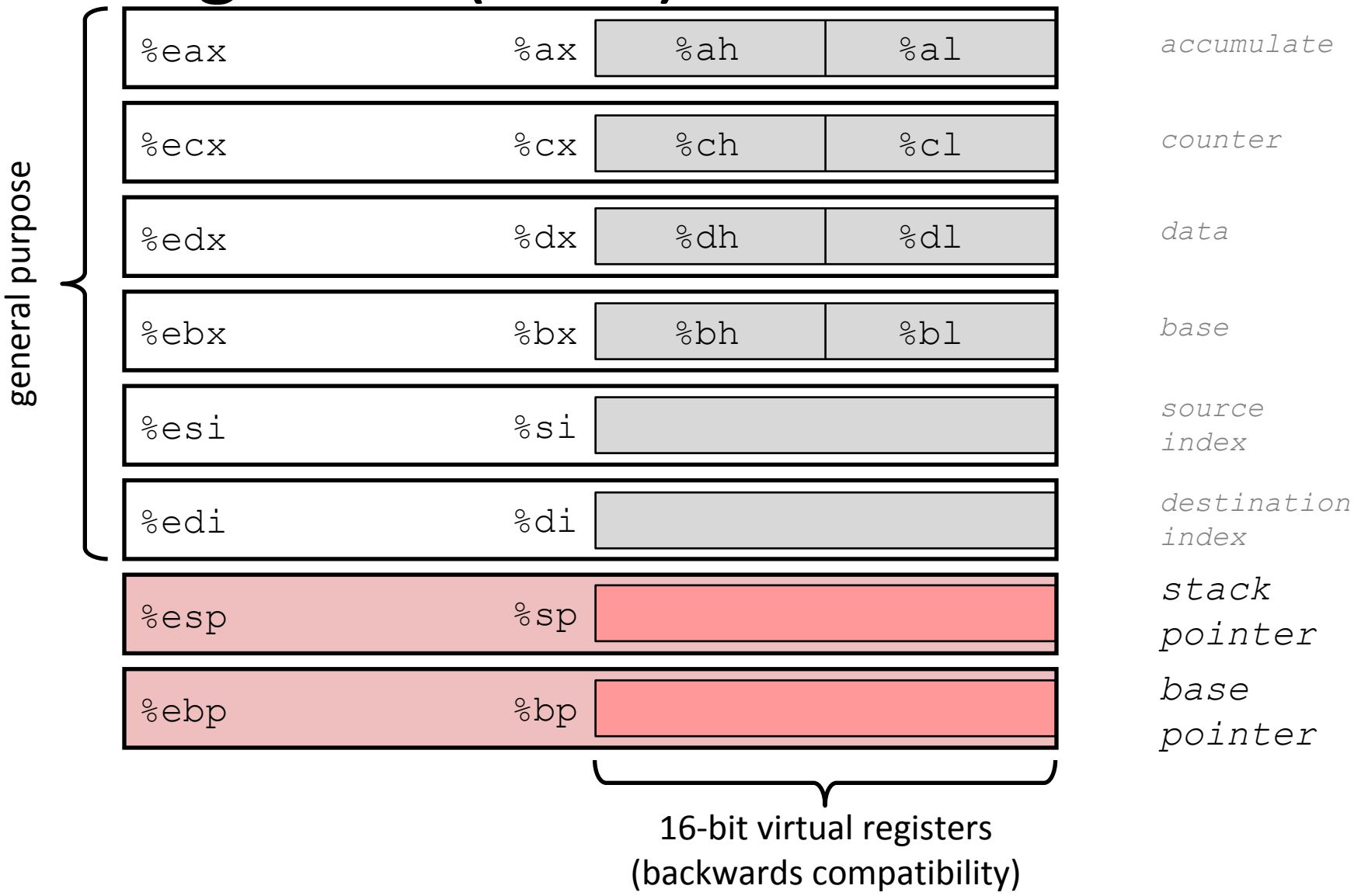
Source: H&P textbook

x86 implementation

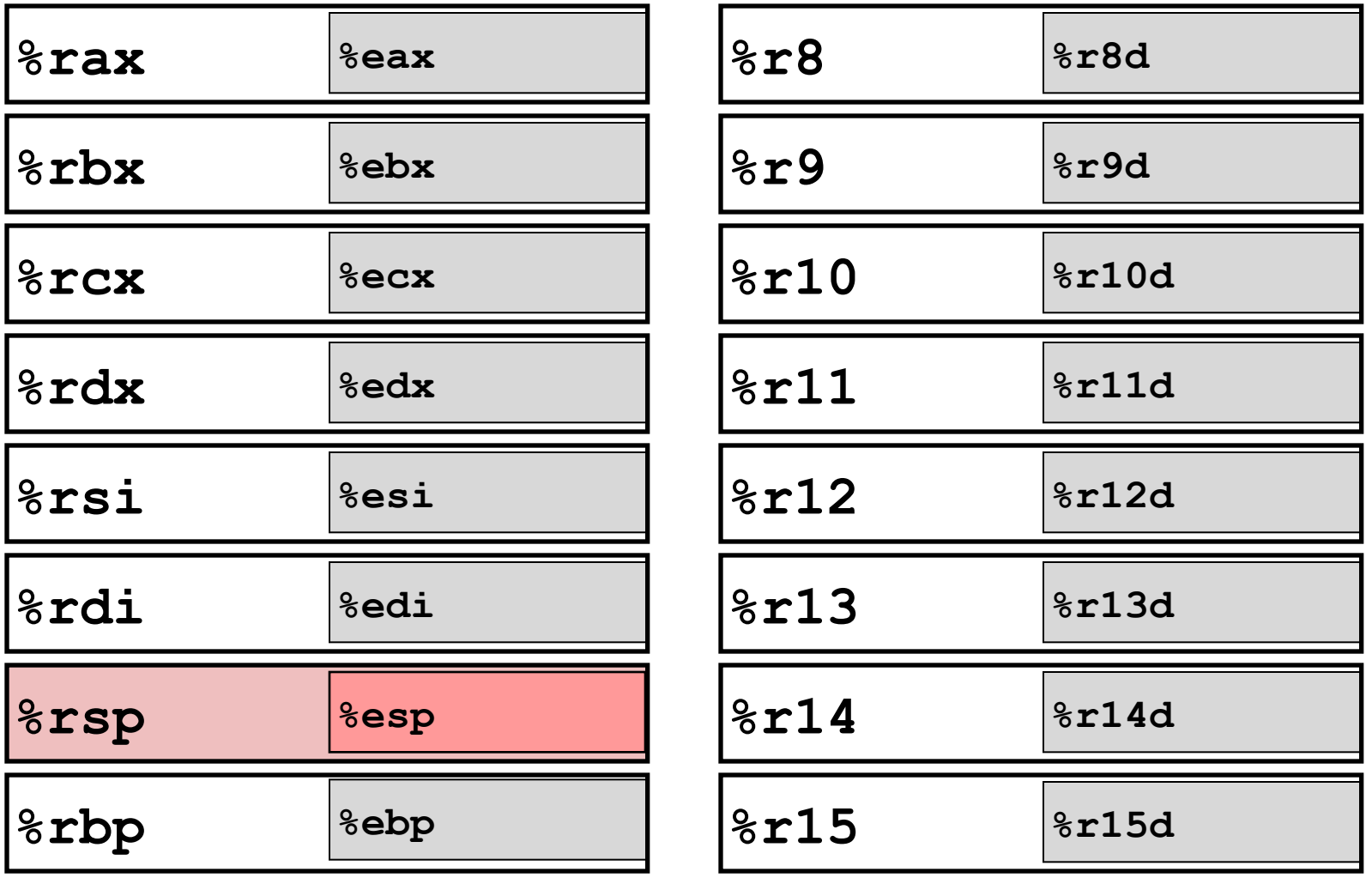


- EIP is incremented after each instruction
- Instructions are different length
- EIP modified by CALL, RET, JMP, and conditional JMP

Integer Registers (IA32)

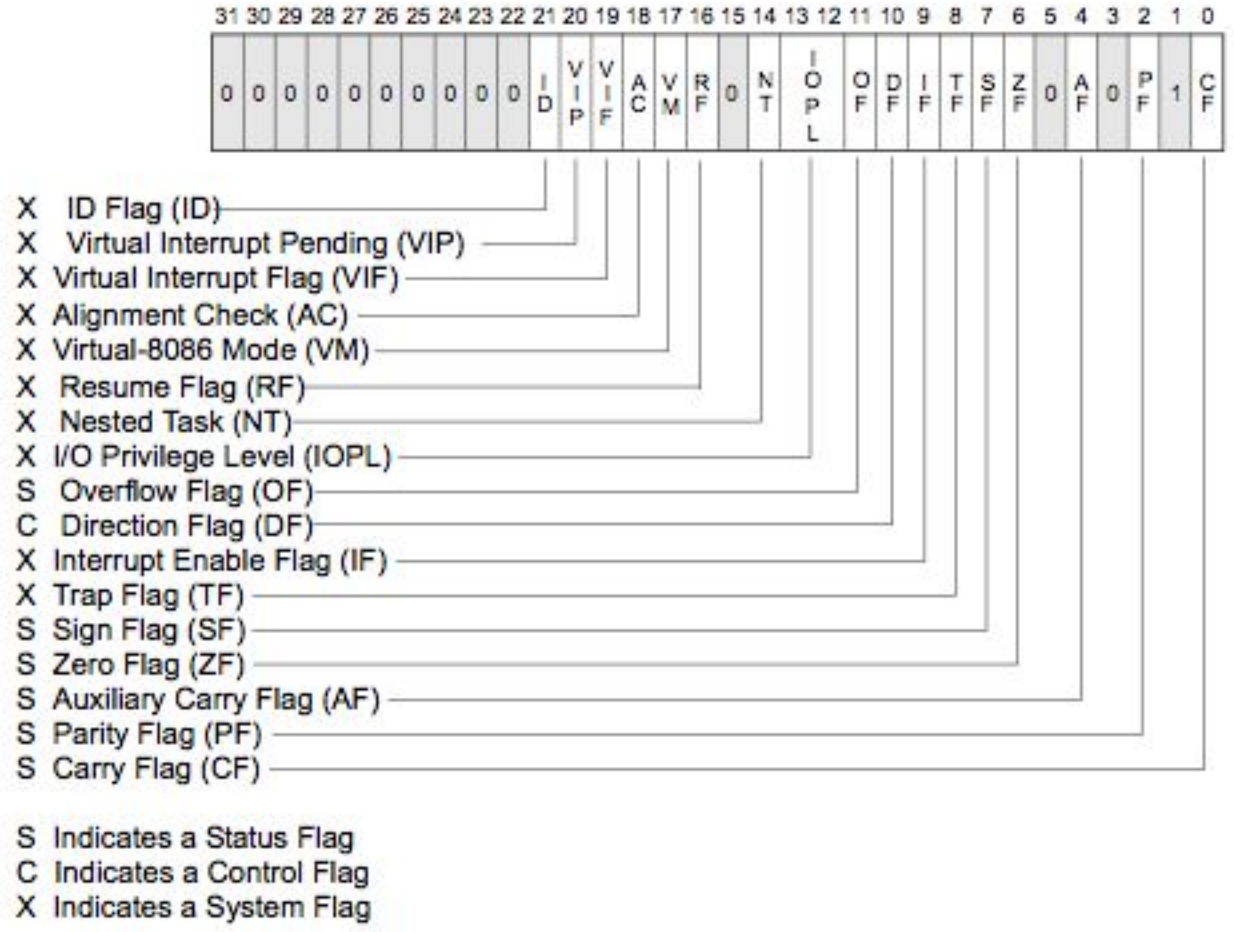


x86-64 Integer Registers



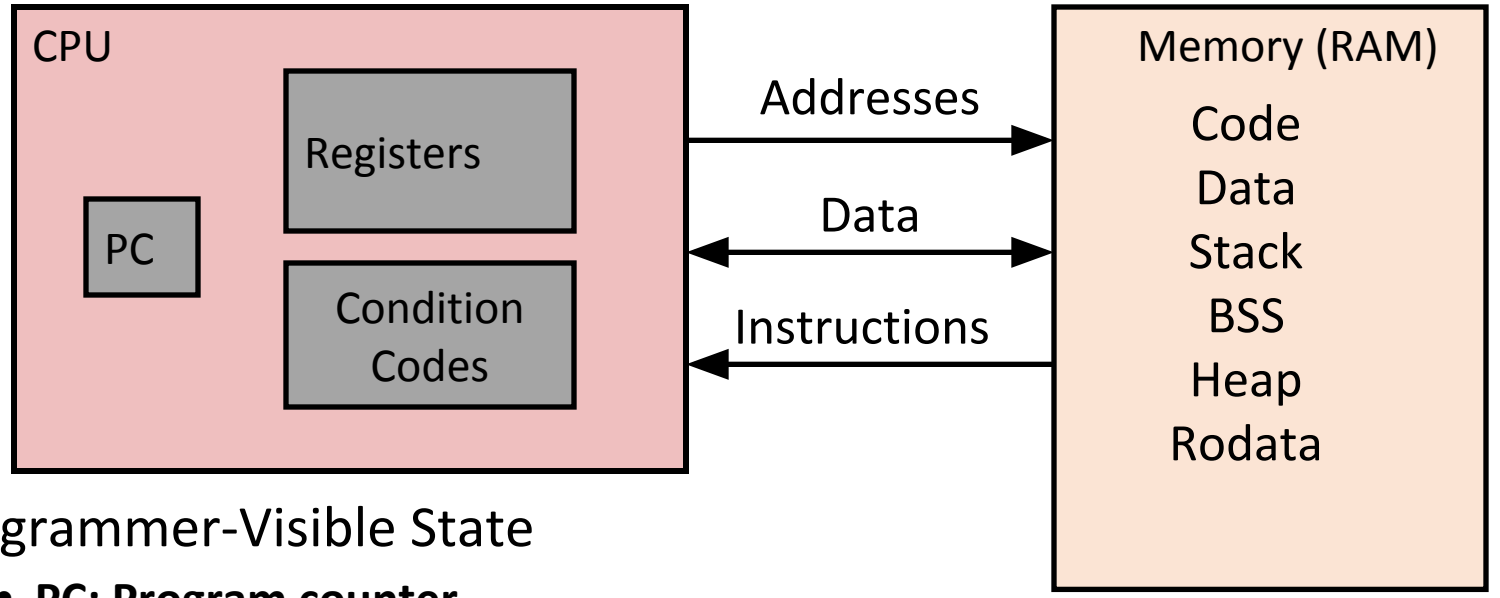
- Extend existing registers. Add 8 new ones.
- Make **%ebp/%rbp** general purpose

EFLAGS register



- Test instructions: TEST EAX, 0
- Conditional JMP instructions: JNZ address

Assembly Programmer's View

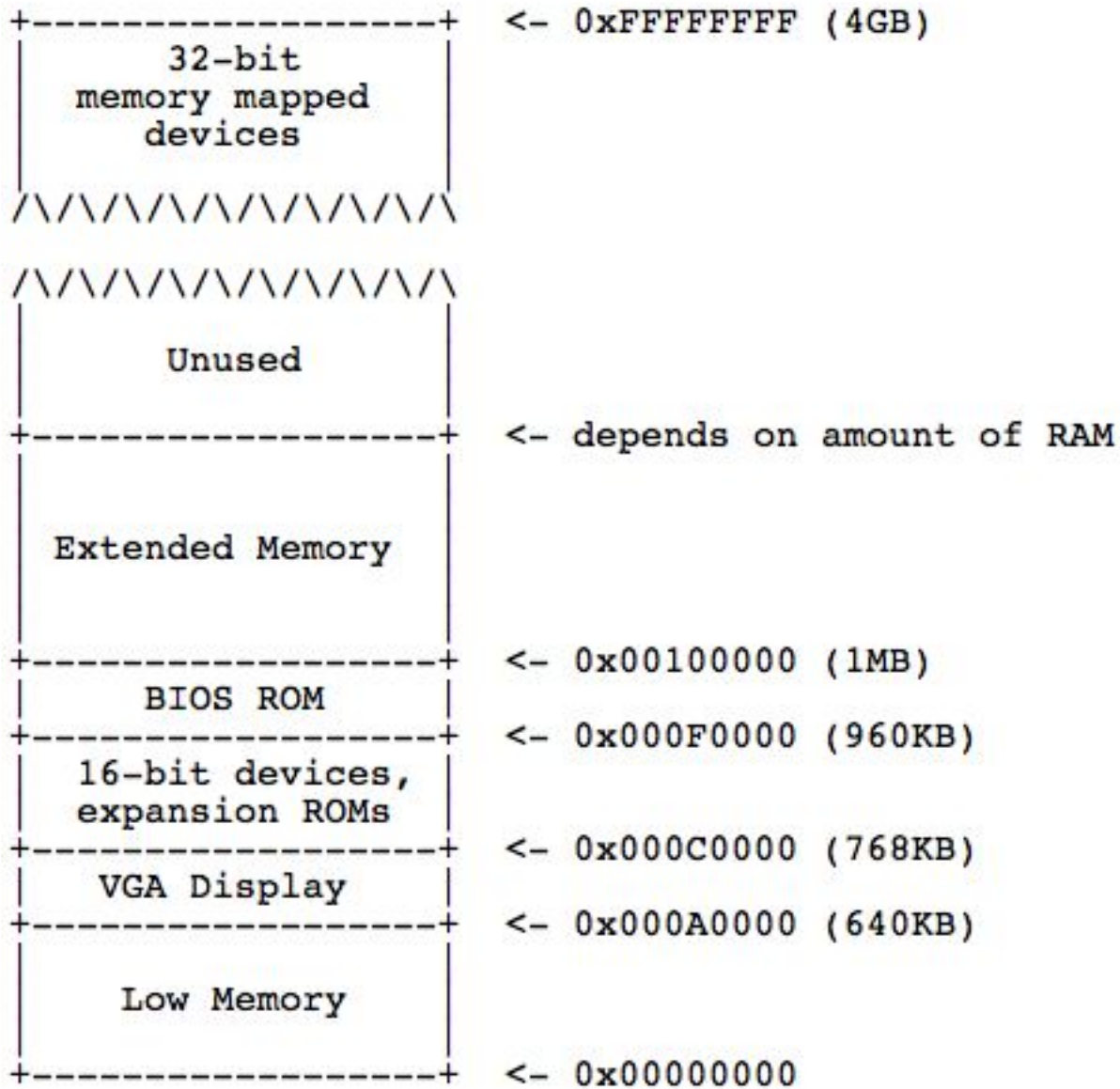


Programmer-Visible State

- **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 and user data
  - Stack to support procedures
  - Heap: Dynamic memory allocation (malloc(), calloc() etc.)

Physical memory layout



x86 instruction set

- Instructions classes:
  - Data movement: MOV, PUSH, POP, LEA…
  - Arithmetic: TEST, SHL, ADD, MUL…
  - I/O: IN, OUT, …
  - Control: JMP, JZ, JNZ, CALL, RET
  - String: MOVSB, …
  - System: IRET, INT, SYSCALL

x86 instruction set (Data movement)

- MOV , LEA
  - MOV: Move (byte/word/long/long long) from Reg. to mem. / mem. to Reg. / Imm. to reg. / Imm. to mem.

    Corresponding C syntax:

```
fun(){
    int a=10;      //Imm. to mem.
    ….
   }
```

  - LEA: Load effective address to register (byte/word/long/long long) of mem. location to Reg.

    Corresponding C syntax:

```
fun(){
    int a=10;
int b;
     b=&a;        // effective address of ‘a’ on the stack.

     ….
   }
```

x86 instruction set (Arithmetic)

- ADD, SUB, DIV, MUL, IMUL, etc.

  - Arithmetic operations (evident from the names).
  - Can use any of the general purposes registers or mem. loc.
    as the target (accumulator), where the result is saved.
  - Flags register (EFLAGS/RFLAGS) bits set to indicate, carry, overflow, sign etc.

```
add <reg>,<reg>          addl %rax, %rbx          add rbx, rax
add <reg>,<mem>          addw (%rcx), %ax         add ax, [rcx]
add <mem>,<reg>          addq  %rax, (%rsi)       add  [rsi],rax
add <reg>,<imm>
add <imm>,<con>
```

x86 instruction set (I/O)

- IN, OUT

  - Read/write bytes to hard ware device identified with address.

  IN Read from a port
  OUT Write to a port
  INS/INSB Input string from port/Input byte string from port
  INS/INSW Input string from port/Input word string from port
  INS/INSD Input string from port/Input doubleword string from port
  OUTS/OUTSB Output string to port/Output byte string to port
  OUTS/OUTSW Output string to port/Output word string to port
  OUTS/OUTSD Output string to port/Output doubleword string to port

x86 instruction set (Control)

- JMP, JZ, JNZ, CALL, RET
  - Unconditional / conditional jump to an address, call a function (procedure) using its address (in code memory).

Instruction	**Description**
JMP *rel8*	Jump short, RIP = RIP + 8-bit displacement sign extended to 64-bits
JMP *rel16*	Jump near, relative, displacement relative to next instruction. Not supported in 64-bit mode.
JMP *rel32*	Jump near, relative, RIP = RIP + 32-bit displacement sign extended to 64-bits
JMP *r/m16*	Jump near, absolute indirect, address = zero-extended *r/m16.* Not supported in 64-bit mode.
JMP *r/m32*	Jump near, absolute indirect, address given in *r/m32.* Not supported in 64-bit mode.
JMP *r/m64*	Jump near, absolute indirect, RIP = 64-Bit offset from register or memory

x86 instruction set (Control)

- CALL, RET

CALL <mem>/<reg>

Equivalent to:

push eip
jmp <mem>/<reg>

RET

Equivalent to:

pop <reg_1>
jmp <reg_1>

x86 instruction set (System)

- IRET
  -Return from an interrupt handler
  -During interrupt handling, the flags are saved on the stack.
  IRET restores the flags from the stack.

- INT
  -Software interrupt instruction (aka *trap*).
  -Emulates interrupt and jumps to interrupt handler.

  INT <int_num>

- SYSCALL
  -System call interrupt handler (added in X86_64).
  - Software interrupt specifically for system call interrupt (aka *trap*).
  - System call interrupt handler number identified via registers.