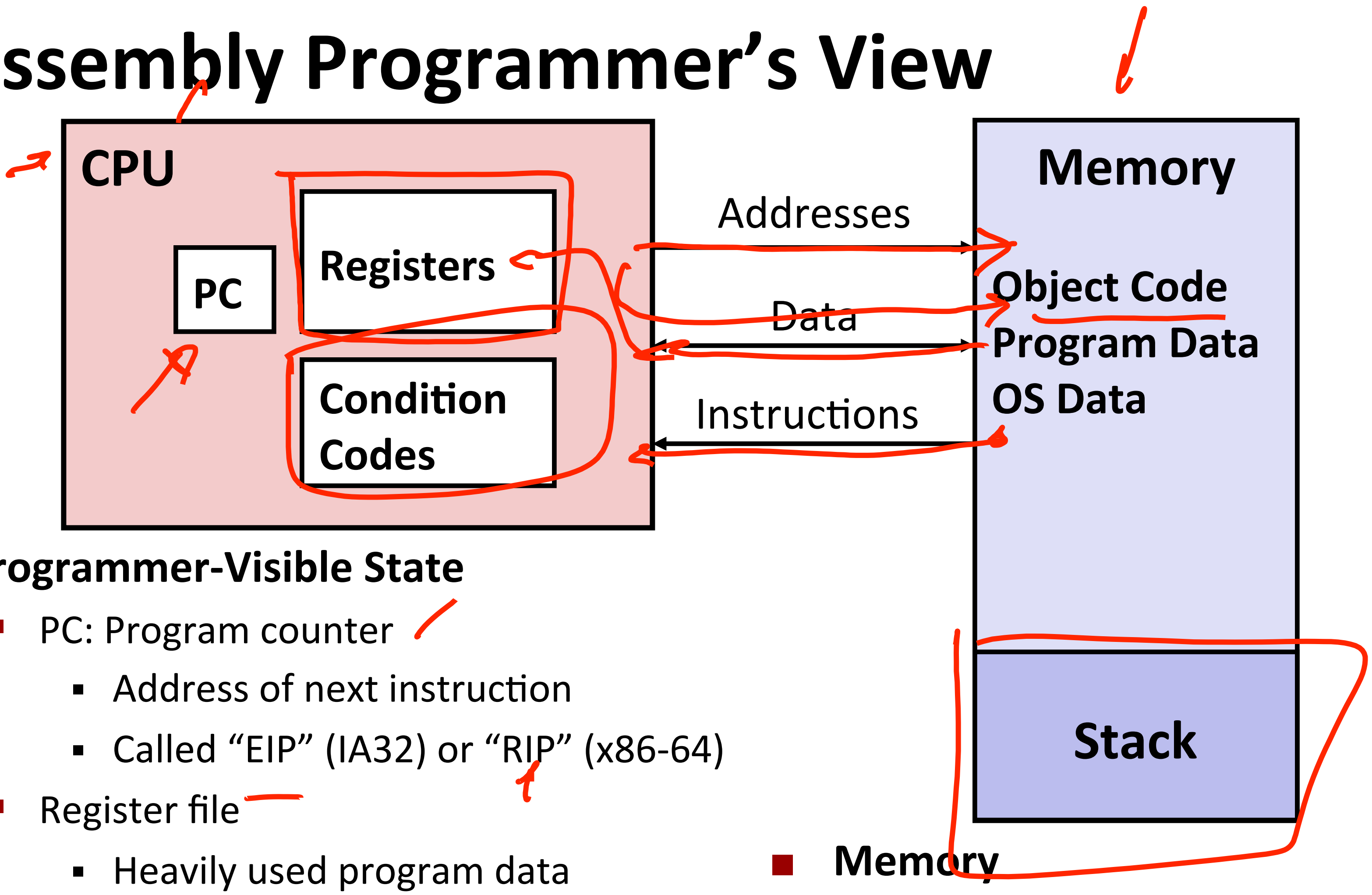
Definitions

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

- Is cache size "architecture"? no
- How about core frequency? ~~no~~ microarchitecture
- And number of registers? ISA, architecture

Assembly Programmer's View

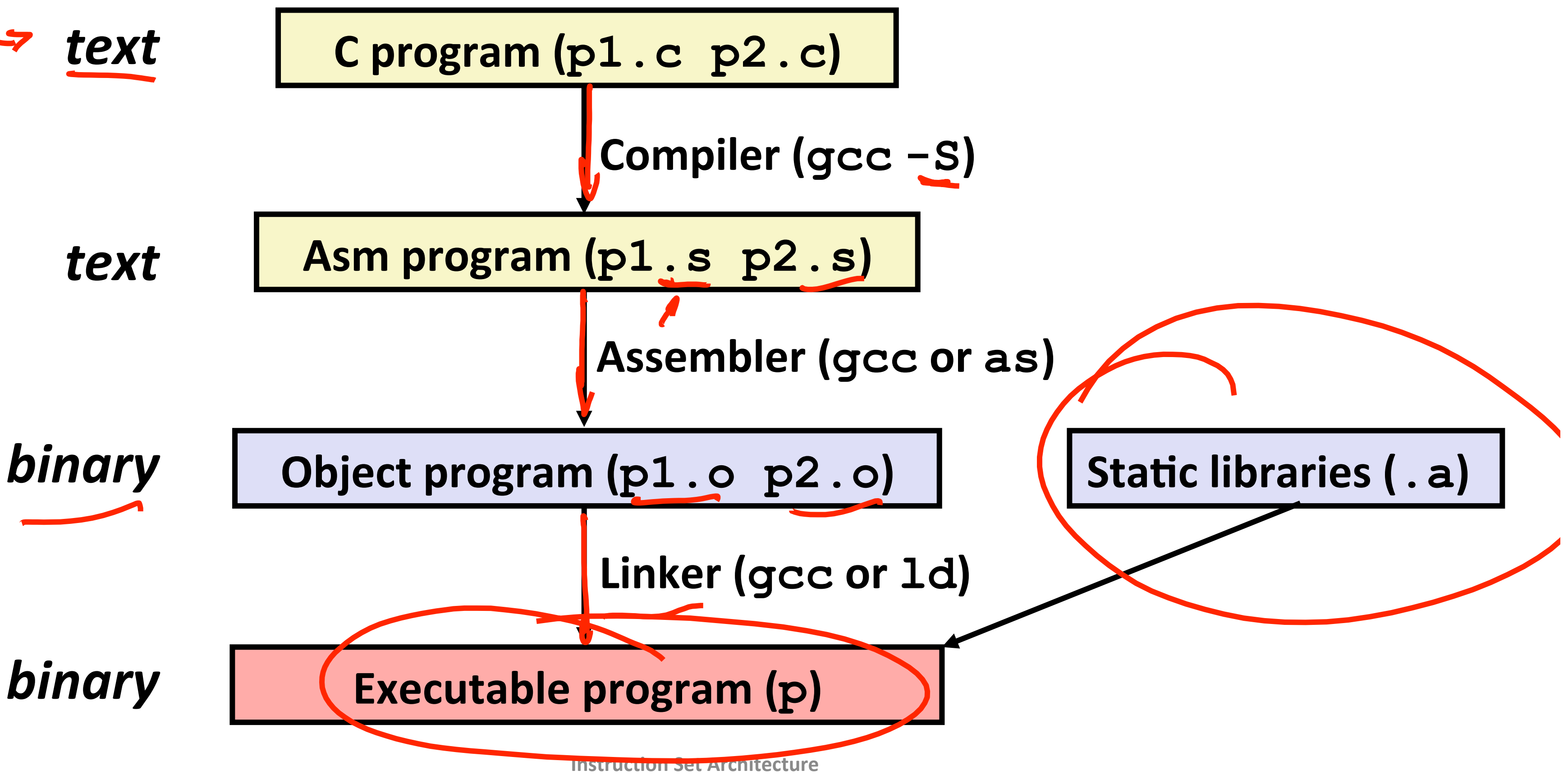


- **Programmer-Visible State**
  - PC: Program counter
    - Address of next instruction
    - Called "EIP" (IA32) or "RIP" (x86-64)
  - Register file
    - Heavily used program data
  - Condition codes
    - Store status information about most recent arithmetic operation
    - Used for conditional branching

- **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 -O1 p1.c p2.c -o p`
  - Use basic optimizations (`-O1`)
  - 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

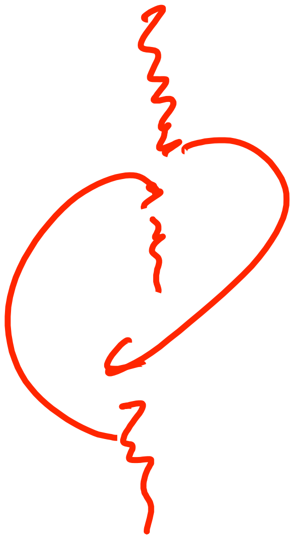
```
gcc -O1 -S code.c
```

Produces file `code.s`

Three Basic Kinds of Instructions

- **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

if (...)



Assembly Characteristics: Data Types

- "Integer" data of 1, 2, 4 (IA32), or 8 (just in x86-64) bytes
  - Data values
  - Addresses (untyped pointers)
- Floating point data of 4, 8, or 10 bytes
- What about "aggregate" types such as arrays or structs?
  - No aggregate types, just contiguously allocated bytes in memory

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`**
- **Not at all obvious where each instruction starts and ends**

- **Assembler**
  - Translates `.s` into `.o`
  - Binary encoding of each instruction
  - Nearly-complete image of executable code
  - Missing links between code in different files
- **Linker**
  - Resolves references between object files and (re)locates their data
  - 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 two 4-byte integers
    - "Long" words in GCC speak
    - 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
```

- **Disassembler**

  `objdump -d p`

  - Useful tool for examining object code (`man 1 objdump`)
  - Analyzes bit pattern of series of instructions (delineates instructions)
  - Produces near-exact rendition of assembly code
  - Can be run on either `p` (complete executable) or `p1.o` / `p2.o` file

Alternate Disassembly

Object

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

Disassembled

```
0x401040 <sum>:      push    %ebp
0x401041 <sum+1>:    mov     %esp,%ebp
0x401043 <sum+3>:    mov     0xc(%ebp),%eax
0x401046 <sum+6>:    add     0x8(%ebp),%eax
0x401049 <sum+9>:    mov     %ebp,%esp
0x40104b <sum+11>:   pop     %ebp
0x40104c <sum+12>:   ret
```

- **Within gdb debugger**

  `gdb p`

  `disassemble sum`

  (disassemble function)

  `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
```

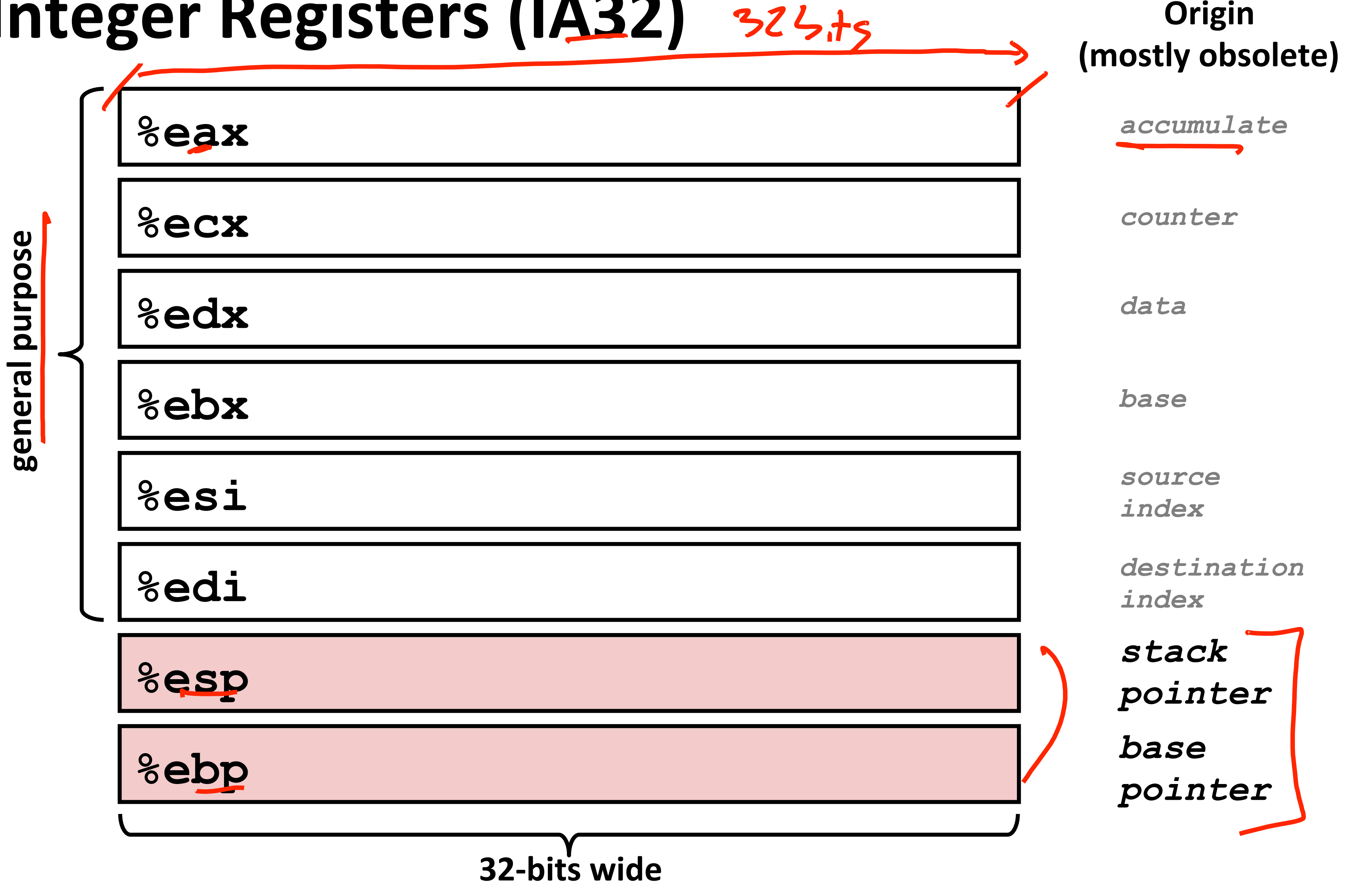
- **Anything that can be interpreted as executable code**
- **Disassembler examines bytes and reconstructs assembly source**

What Is A Register?

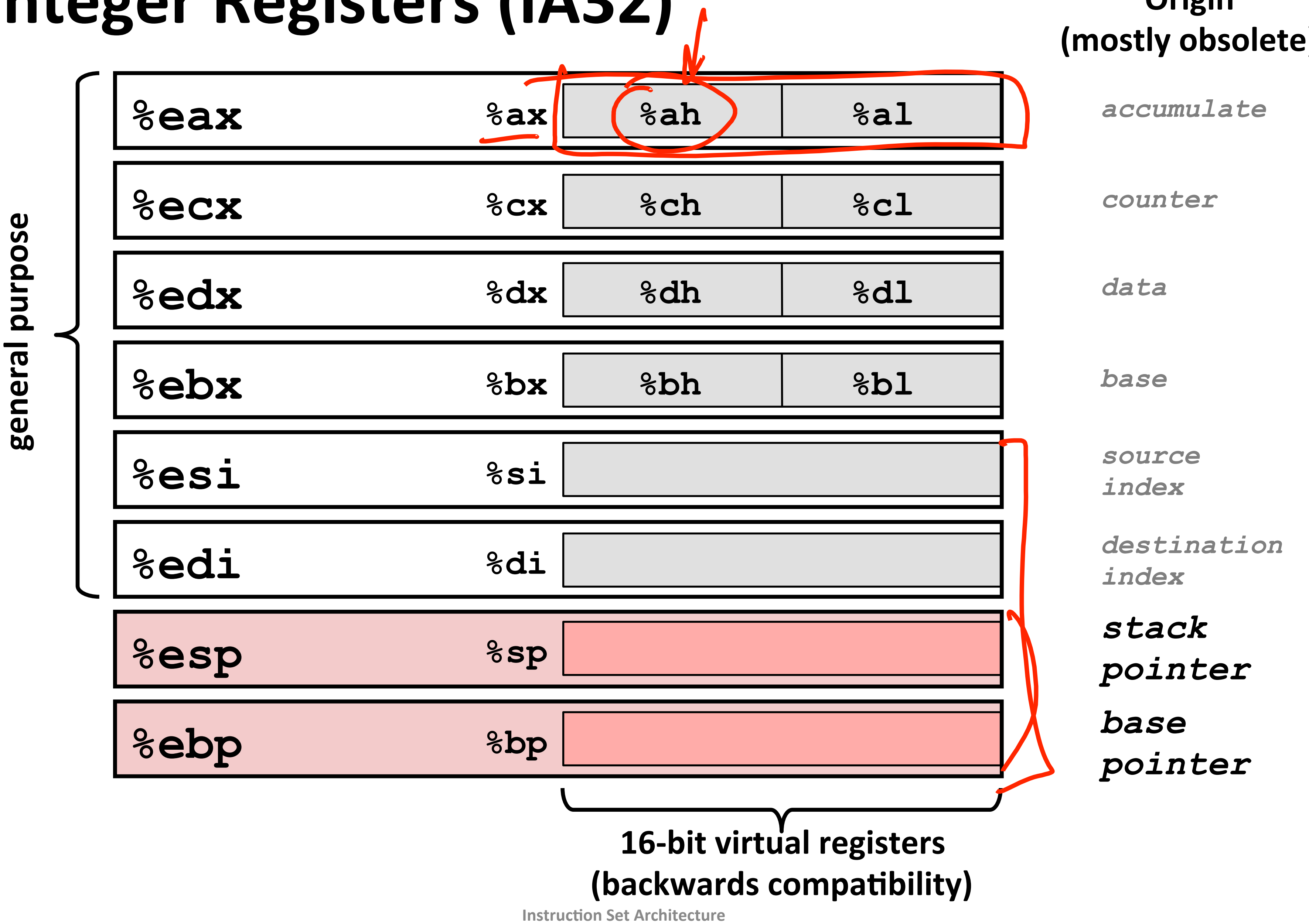
- **A location in the CPU that stores a small amount of data, which can be accessed very quickly (once every clock cycle)**

- **Registers are at the heart of assembly programming**
  - They are a precious commodity in all architectures, but *especially* x86

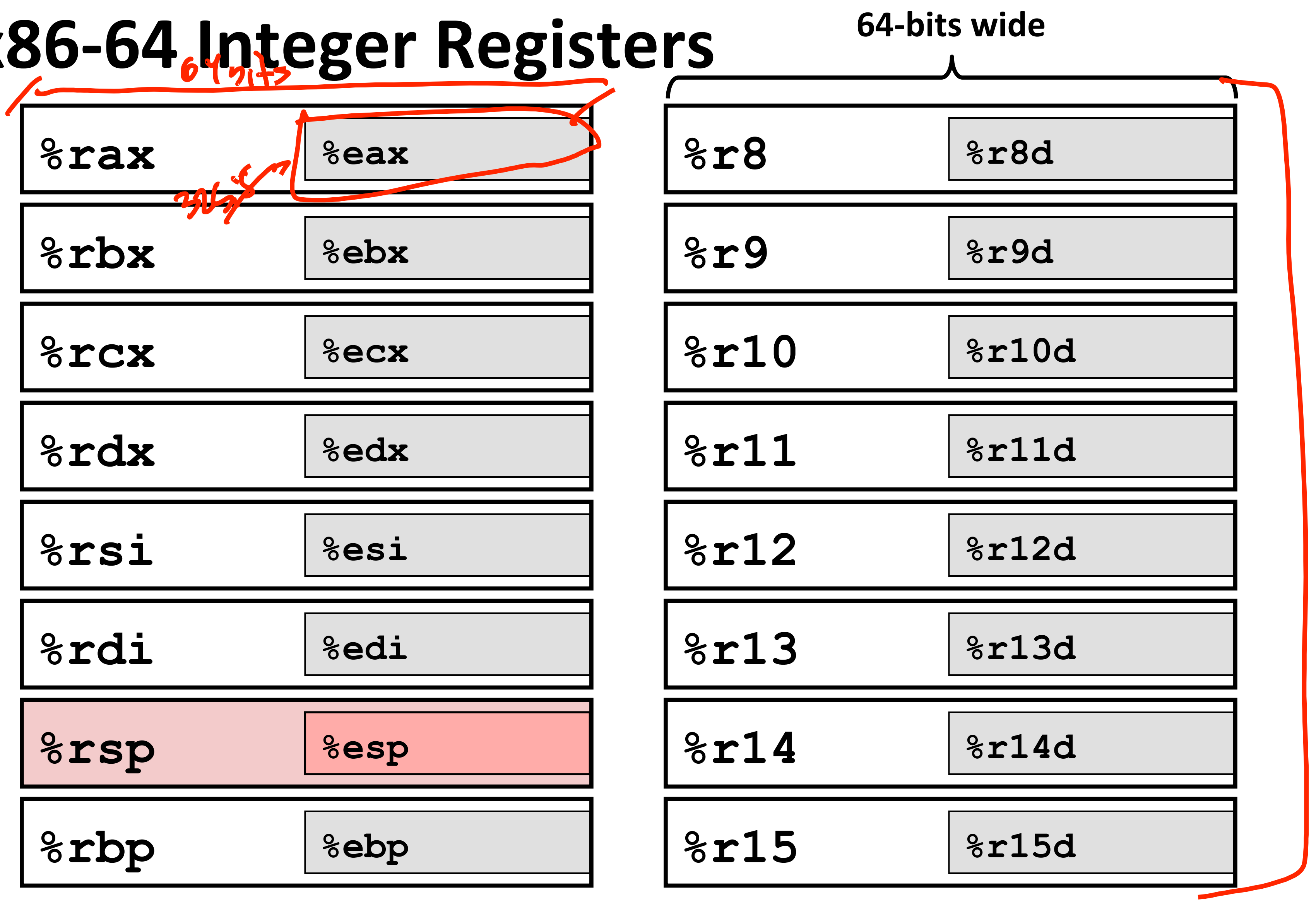
Integer Registers (IA32)



Integer Registers (IA32)



x86-64 Integer Registers



- Extend existing registers, and add 8 new ones; *all* accessible as 8, 16, 32, 64 bits.

Summary: Machine Programming

- **What is an ISA (Instruction Set Architecture)?** ✓
  - Defines the system's state and instructions that are available to the software
- **History of Intel processors and architectures** ✓
  - Evolutionary design leads to many quirks and artifacts
- **C, assembly, machine code** ✓
  - Compiler must transform statements, expressions, procedures into low-level instruction sequences
- **x86 registers**
  - Very limited number ✓
  - Not all general-purpose