جامعة المنارة
MANARA UNIVERSITY

كلية الهندسة المعلوماتية
بنيان حواسيب 2
الفصل الأول 2025-2024
المحاضرة الأولى

د كندة أبو فاسم

- مفردات المقرر

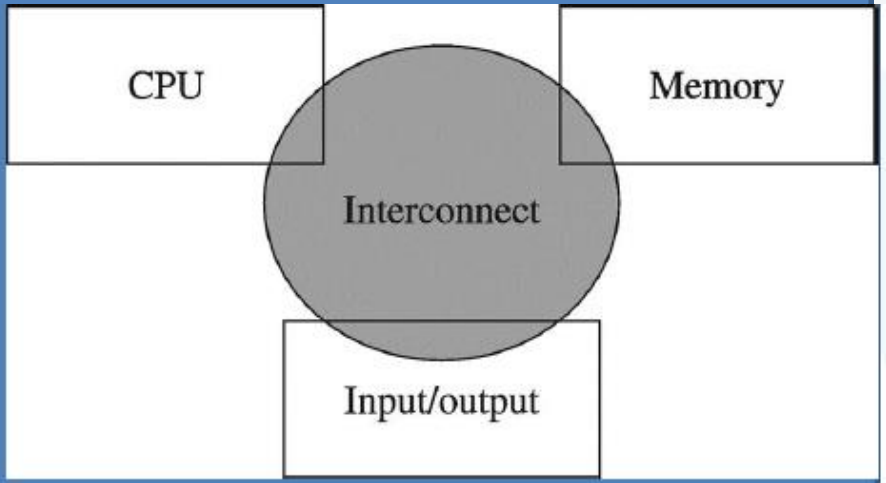
- المكونات الأساسية للحاسب Basic Computer Organization
- بنية المعالج 8086
- لغة التجميع assembly language
- مجموعة التعليمات Instruction Set
- أنماط العنونة addressing mode
- مجموعة تعليمات النقل
- مجموعة التعليمات الحسابيةarithmetic Instruction Set
- العمليات المنطقية
- تعليمات الإزاحةInstructions Shift
- تعليمات التحكم بالعملية Control Processor Instructions Conditional Jump
- تعليمات معالجة السلاسل string instruction
- بوابات الدخل/ الخرج I/O System Design
- أنماط المقاطعة interrupt mode

Basic Computer Organization
المكونات الأساسية للحاسب

- Basic components
- The processor
  * Execution cycle
  * System clock
- Number of addresses
  * 3-address machines
  * 2-address machines
  * 1-address machines
  * 0-address machines
  * Load/store architecture

- Memory
  * Basic operations
  * Types of memory
  * Storing multibyte data
- Input/Output

- Basic components of a computer system
  - * Processor
  - * Memory
  - * I/O
  - * System bus
    - » Address bus
    - » Data bus
    - » Control bus



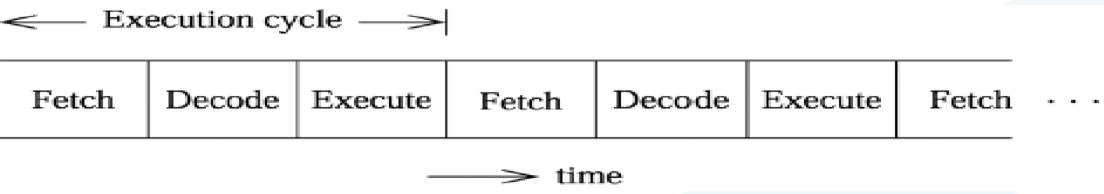
* المعالج
* الذواكر
* وحدات الدخل /خرج
* خطوط النقل
.. خطوط العنونة
.. خطوط المعطيات
.. خطوط التحكم

المكونات الأساسية

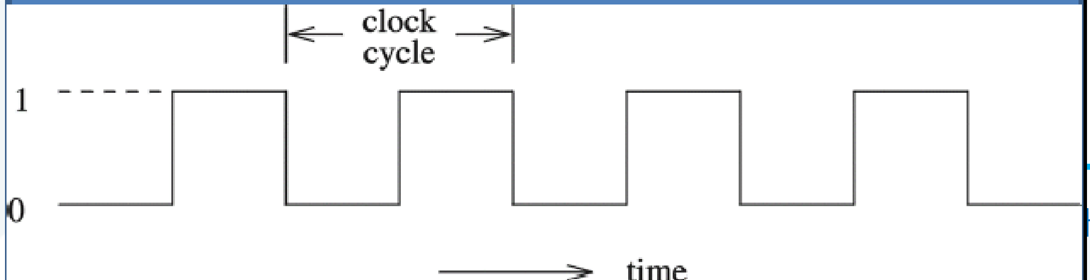
عمل المعالج processor

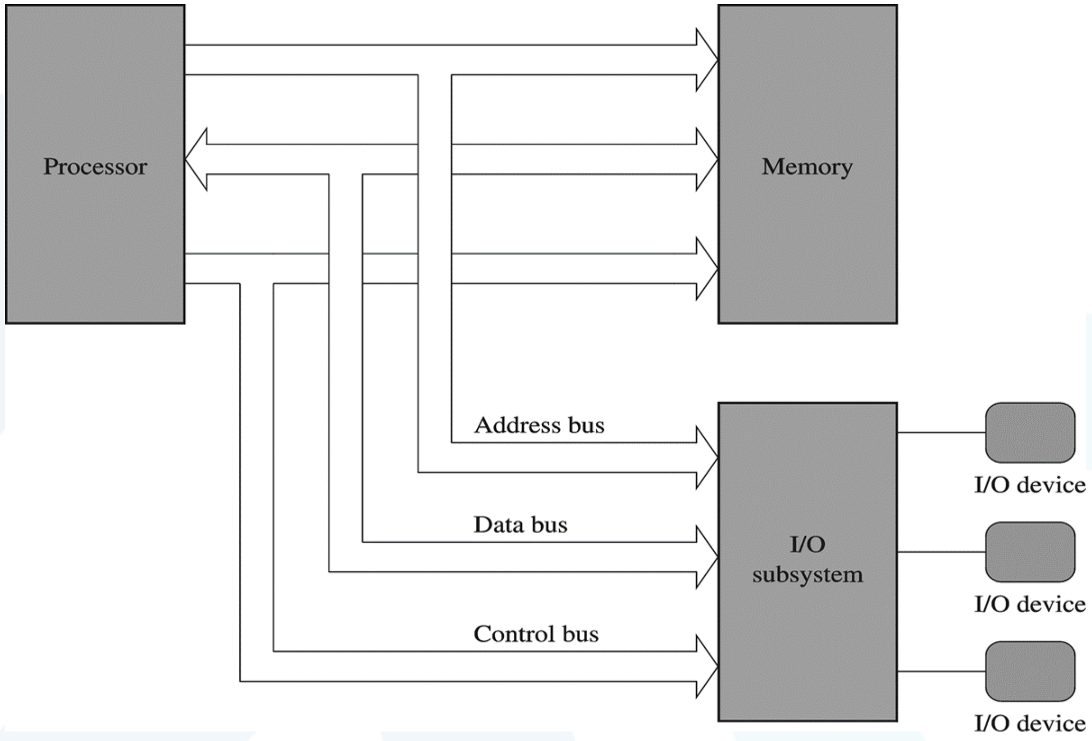
يقوم المعالج خلال دورة تنفيذ التعليمة بما يلي:

- مرحلة البحث عن التعليمة Fetch cycle
- مرحلة فك ترميز Decode
- مرحلة تنفيذ Execution



- System clock
  * Provides timing signal
  * Clock period = $\frac{1}{\text{Clock frequency}}$

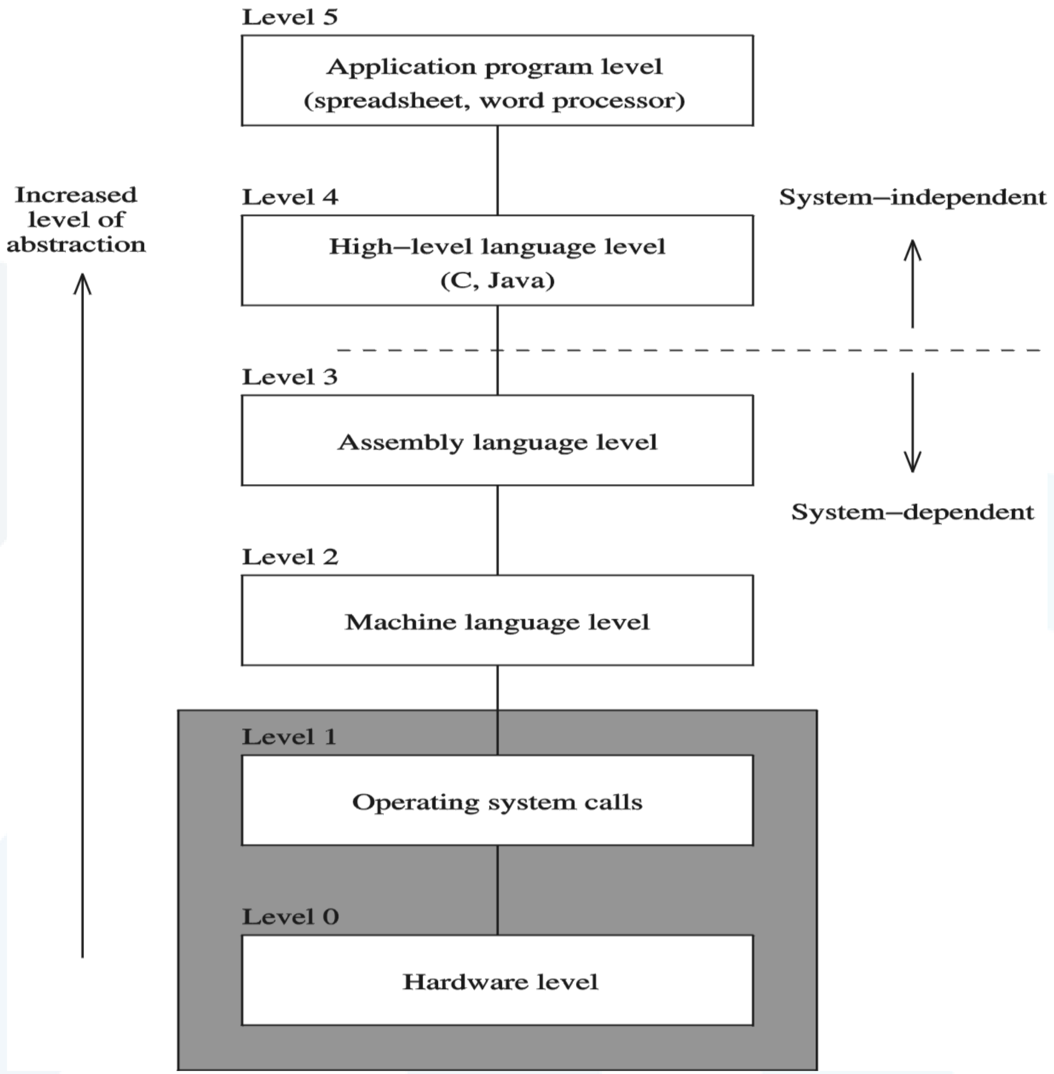




Why Program in Assembly Language?

- Two main reasons:
  * Efficiency
    » Space-efficiency
    » Time-efficiency
  * Accessibility to system hardware
- Space-efficiency
  * Assembly code tends to be compact
- Time-efficiency
  * Assembly language programs tend to run faster
    » Only a well-written assembly language program runs faster
      – Easy to write an assembly program that runs slower than its high-level language equivalent

View of Computer Systems



Typical Applications

- Accessibility to system hardware
  * System software typically requires direct control of the system hardware devices
    - » Assemblers, linkers, compilers
    - » Network interfaces, device drivers
    - » Video games
- Space-efficiency
  * Not a big plus point for most applications
  * Code compactness is important in some cases
    - – Portable and hand-held device software
    - – Spacecraft control software

- Application that need one of the three advantages of the assembly language
- Time-efficiency
  * Time-convenience
    - » Good to have but not required for functional correctness
      - – Graphics
  * Time-critical
    - » Necessary to satisfy functionality
    - » Real-time applications
      - – Aircraft navigational systems
      - – Process control systems
      - – Robot control software
      - – Missile control software

Number of Addresses

- Four categories
  - 3-address machines
    - 2 for the source operands and one for the result
  - 2-address machines
    - One address doubles as source and result
  - 1-address machine
    - Accumulator machines
    - Accumulator is used for one source and result
  - 0-address machines
    - Stack machines
    - Operands are taken from the stack
    - Result goes onto the stack

- Example

* C statement

```
A = B + C * D - E + F + A
```

* Equivalent code:

```
mult    T,C,D   ;T = C*D
add     T,T,B   ;T = B+C*D
sub     T,T,E   ;T = B+C*D-E
add     T,T,F   ;T = B+C*D-E+F
add     A,T,A   ;A = B+C*D-E+F+A
```

- Three-address machines
  * Two for the source operands, one for the result
  * RISC processors use three addresses
  * Sample instructions

```
add     dest,src1,src2
        ; M(dest)=[src1]+[src2]
sub     dest,src1,src2
        ; M(dest)=[src1]-[src2]
mult    dest,src1,src2
        ; M(dest)=[src1]*[src2]
```

Two-Address Machines

- Example
  - C statement

```
A = B + C * D – E + F + A
```

  - Equivalent code:

```
load    T,C   ;T = C
mult    T,D   ;T = C*D
add     T,B   ;T = B+C*D
sub     T,E   ;T = B+C*D-E
add     T,F   ;T = B+C*D-E+F
add     A,T   ;A = B+C*D-E+F+A
```

- Two-address machines
  - One address doubles (for source operand & result)
  - Last example makes a case for it
    - » Address T is used twice
  - Sample instructions

```
load    dest,src ; M(dest)=[src]
add     dest,src ; M(dest)=[dest]+[src]
sub     dest,src ; M(dest)=[dest]-[src]
mult    dest,src ; M(dest)=[dest]*[src]
```

One-Address Machines

- Example
- C statement

  A = B + C * D – E + F + A

- Equivalent code:

```
load   C  ;load C into accum
mult   D  ;accum = C*D
add    B  ;accum = C*D+B
sub    E  ;accum = B+C*D-E
add    F  ;accum = B+C*D-E+F
add    A  ;accum = B+C*D-E+F+A
store  A  ;store accum contents in A
```

- One-address machines

* Uses special set of registers called accumulators
  - » Specify one source operand & receive the result
* Called accumulator machines
* Sample instructions

```
load   addr ; accum = [addr]
store  addr ; M[addr] = accum
add    addr ; accum = accum + [addr]
sub    addr ; accum = accum - [addr]
mult   addr ; accum = accum *[addr]
```

- Example
  - C statement

```
A = B + C * D - E + F + A
```

  - Equivalent code:

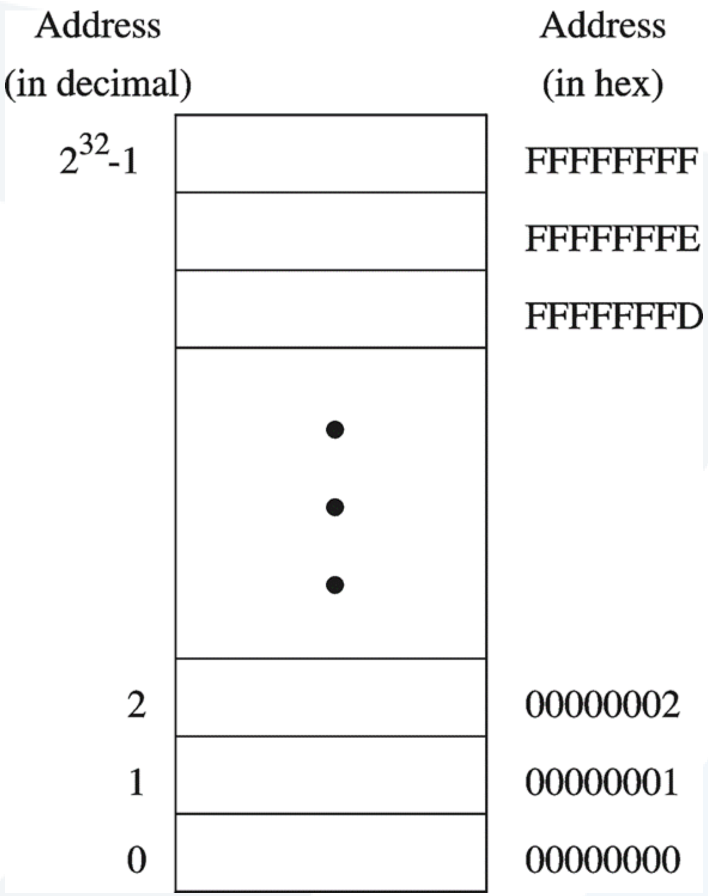
```
push   E          sub
push   C          push   F
push   D          add
Mult              push   A
push   B          add
add               pop    A
```

- Zero-address machines
  - Stack supplies operands and receives the result
    - Special instructions to load and store use an address
  - Called stack machines (Ex: HP3000, Burroughs B5500)
  - Sample instructions

```
push   addr   ; push([addr])
pop    addr   ; pop([addr])
add           ; push(pop + pop)
sub           ; push(pop - pop)
mult          ; push(pop * pop)
```

Memory الذواكر

- Memory can be viewed as an ordered sequence of bytes
- Each byte of memory has an address
  - * Memory address is essentially the sequence number of the byte
  - * Such memories are called *byte addressable*
  - * Number of address lines determine the memory address space of a processor



الذوا كر

تبدو الذاكرة كنسق من البايت المتتالية

لكل بايت عنوان

مثال تعنون الذاكرة من

$0 \longrightarrow 2^N - 1$

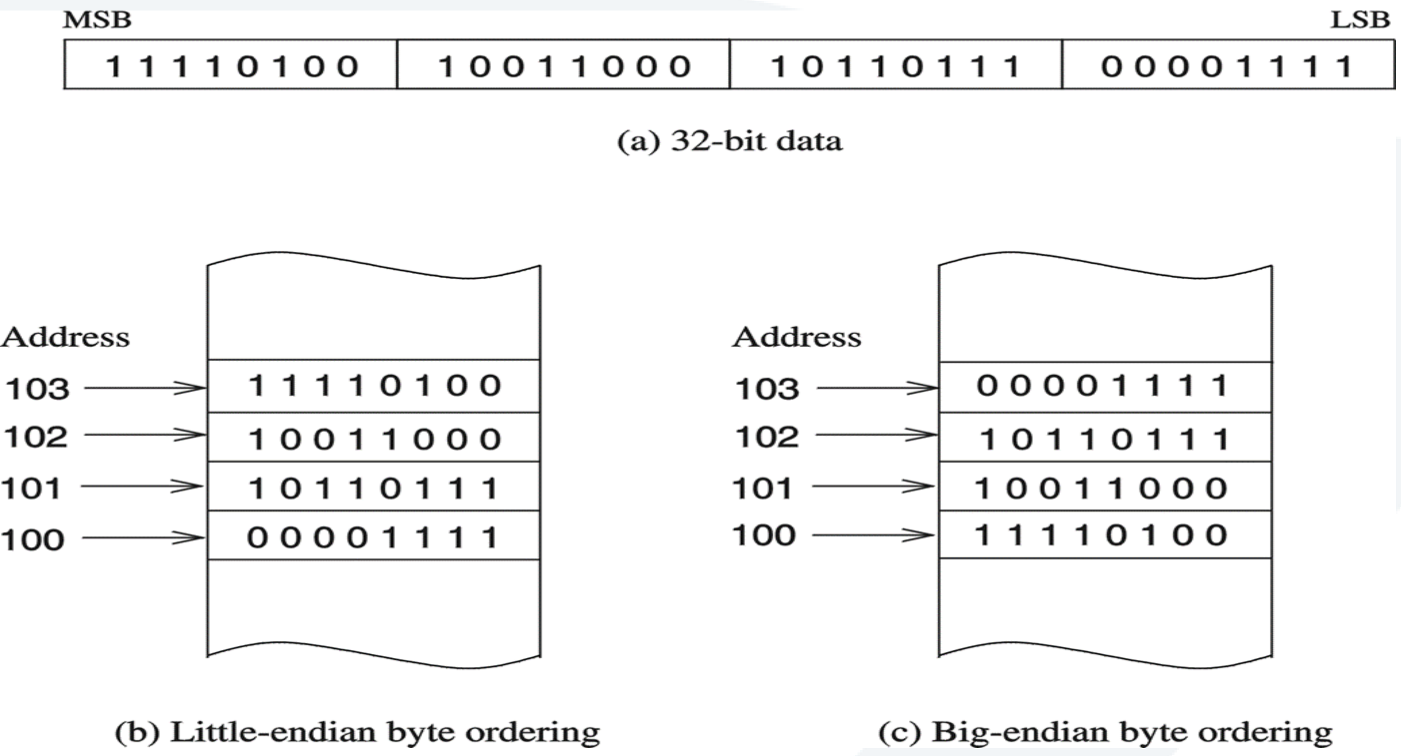
اذا كان عدد خطوط العنونة 32 bit

فيكون عدد مواقع الذاكرة

$2^N - 1 = 4\,G\,byte$

multibyte تخزين

Memory الذواكر



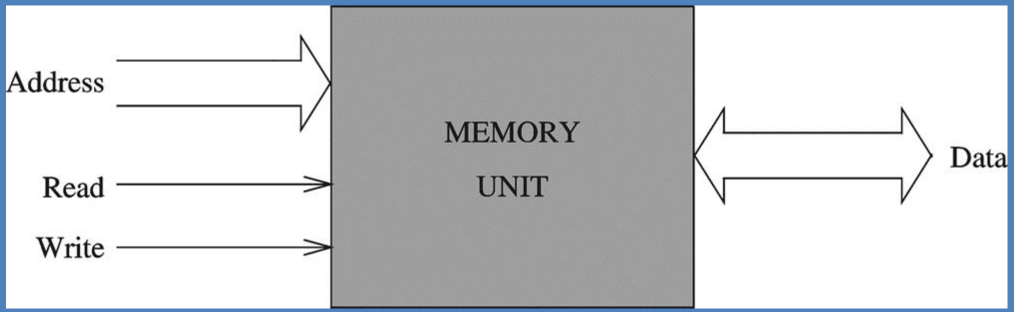
(a) 32-bit data

(b) Little-endian byte ordering

(c) Big-endian byte ordering

- Little endian
  - » Used by Intel IA-32 processors
- Big endian
  - » Used most processors by default

- Two basic memory operations
  - ∗ Read operation (read from memory)
  - ∗ Write operation (write into memory)
- Access time
  - » Time needed to retrieve data at addressed location
- Cycle time
  - » Minimum time between successive operations



الدخل والخرج input /output

- Processor and I/O interface points for exchanging data are called *I/O ports*
- Two ways of mapping I/O ports
  * Memory-mapped I/O
    - » I/O ports are mapped to the memory address space
      - – Reading/writing I/O is similar to reading/writing memory
        - ✈Can use memory read/write instructions
  * Isolated I/O
    - » Separate I/O address space
    - » Requires special I/O instructions (like `in` and `out` in Pentium)
    - » Intel 80x86 processors support isolated I/O

وحدات الادخال والإخراج I/O

تستخدم بوابات الاخال والإخراج لتبادل المعطيات بين المعالج وبوابات الدخل / خرج
يمكن تخطيط بوابات I/O
كوحدة من الذاكرة
* تحجز بوابات الدخل / خرج عناوين من الذاكرة الرئيسية كما تقوم بعملية القراءة والكتابة كما في الذواكر
*بوابات دخل/خرج معزولة
- تستخدم فضاء عناوين منفصلة
- تحتاج الى تعليمات خاصة للدخل والخرج
في معالجات بنتيوم مثلin , out
- تدعم معالجات انتل 80x68 وحدات دخل/خرج منفصلة

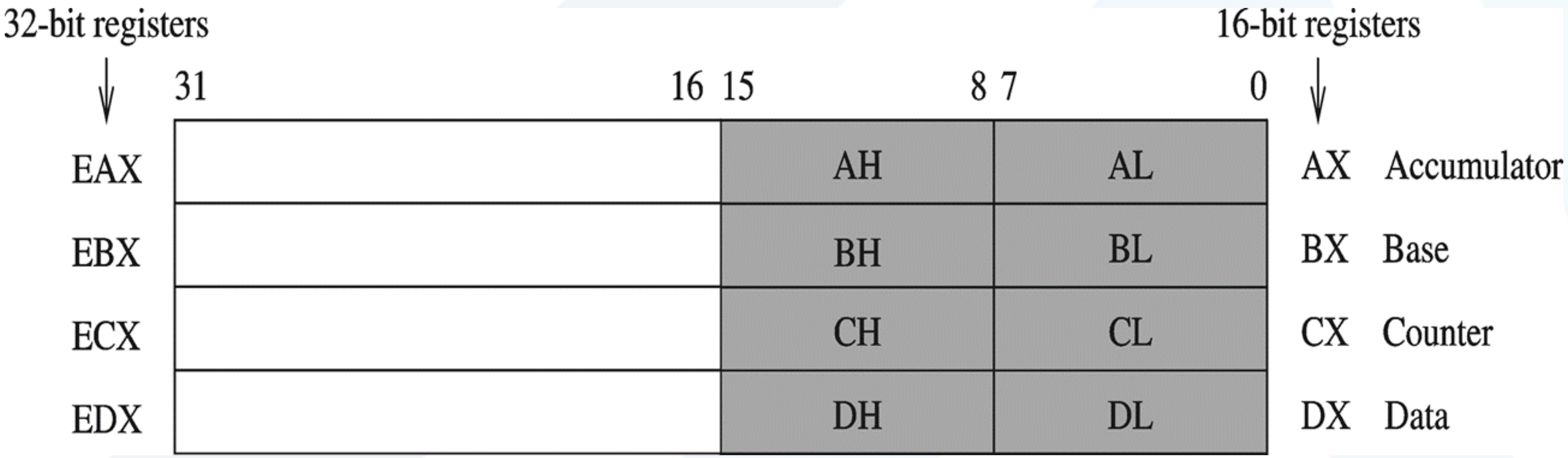
Input/Output الدخل والخرج

- Pentium I/O address space
  - ∗ Provides 64 KB I/O address space
  - ∗ Can be used for 8-, 16-, and 32-bit I/O ports
  - ∗ Combination cannot exceed the total I/O address space
    - » can have 64 K 8-bit ports
    - » can have 32 K 16-bit ports
    - » can have 16 K 32-bit ports
    - » A combination of these for a total of 64 KB
  - ∗ I/O instructions do not go through segmentation or paging
    - » I/O address refers to the physical I/O address

عنونة I/O في معالجات بنتيوم
تستخدم معالجات بنتيوم فضاء العناوين
- تزود بفضاء عنوان 64 k byte
- يمكن ان تستخدم بوابات من 8 أو16 أو 32
- تعليمات I/O لاتمر عبر المقاطع أو الصفحات

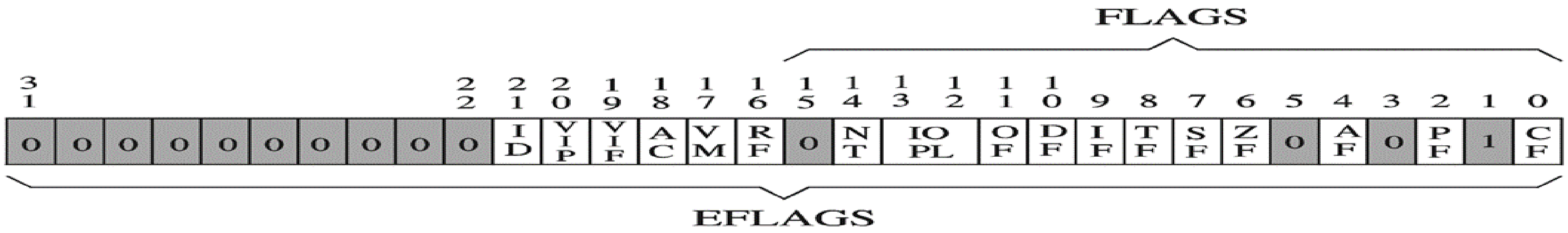
Pentium Registers مسجلات البنتيوم

- Four 32-bit registers can be used as
  * Four 32-bit register (EAX, EBX, ECX, EDX)
  * Four 16-bit register (AX, BX, CX, DX)
  * Eight 8-bit register (AH, AL, BH, BL, CH, CL, DH, DL)
- Some registers have special use
  * ECX for count in loop instructions



Pentium Registers مسجلات البنتيوم

Flags register



Status flags

CF = Carry flag
PF = Parity flag
AF = Auxiliary carry flag
ZF = Zero flag
SF = Sign flag
OF = Overflow flag

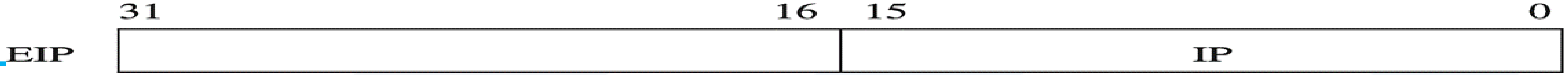
Control flags

DF = Direction flag

System flags

TF = Trap flag
IF = Interrupt flag
IOPL = I/O privilege level
NT = Nested task
RF = Resume flag
VM = Virtual 8086 mode
AC = Alignment check
VIF = Virtual interrupt flag
VIP = Virtual interrupt pending
ID = ID flag

Instruction pointer



Pentium Registers مسجلات البنتيوم

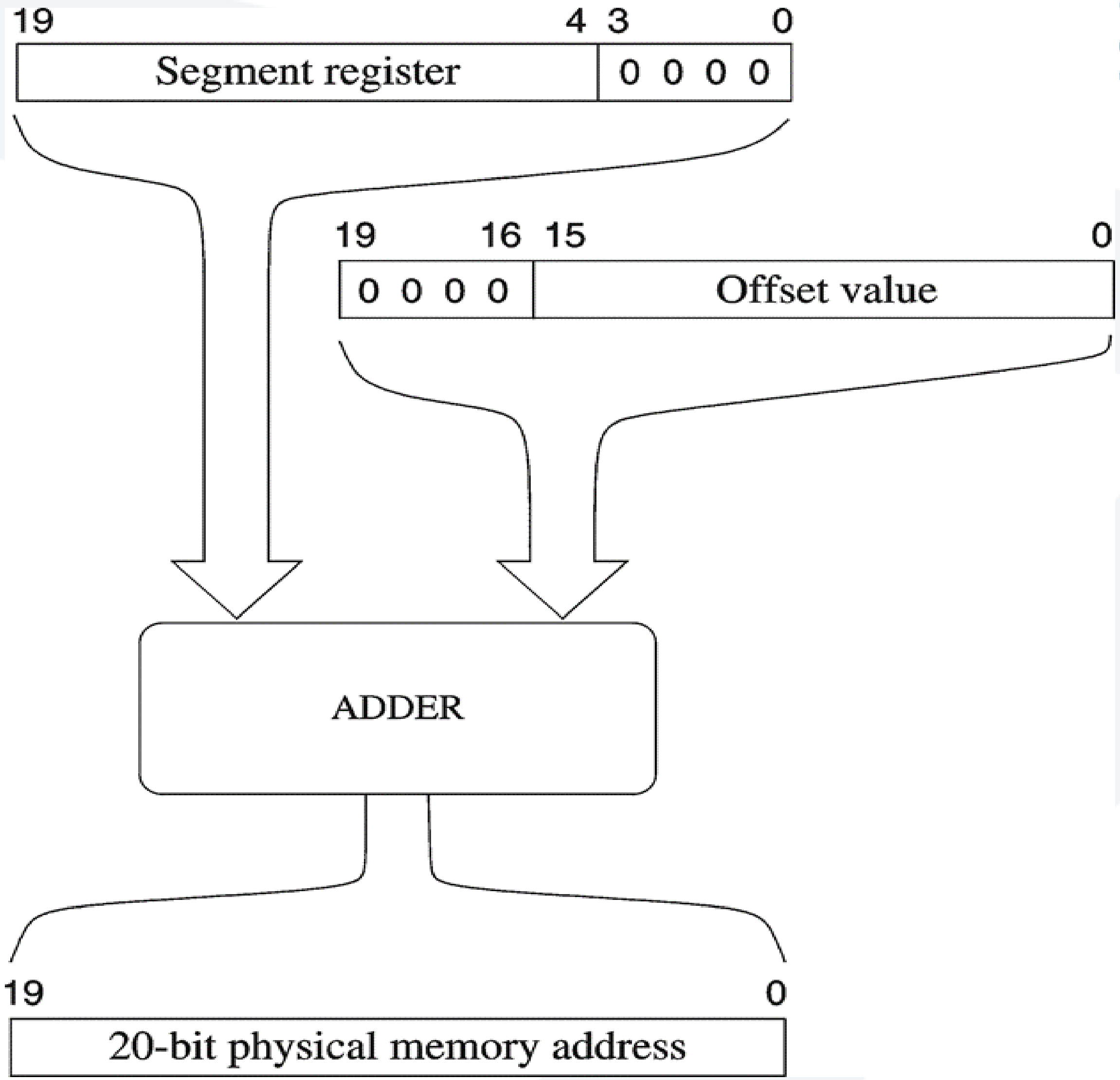
15 … 0	
CS	Code segment
DS	Data segment
SS	Stack segment
ES	Extra segment
FS	Extra segment
GS	Extra segment

- Segment register
  * Six 16-bit registers
  * Support segmented memory architecture
  * At any time, only six segments are accessible
  * Segments contain distinct contents
    » Code
    » Data
    » Stack

Protected Mode Architecture

- Pentium supports two modes
  - Protected mode
    - 32-bit mode
    - Native mode of Pentium
    - Supports segmentation and paging
  - Real mode
    - Uses 16-bit addresses
    - Runs 8086 programs
    - Pentium acts as a faster 8086

Real Mode نمط



- Conversion from logical to physical addresses

```
  11000  (add 0 to base)
+   450  (offset)
  11450  (physical address)
```

Real Mode نمط

- Programs can access up to six segments at any time
- Two of these are for
  - * Data
  - * Code
- Another segment is typically used for
  - * Stack
- Other segments can be used for
  - * data, code,..