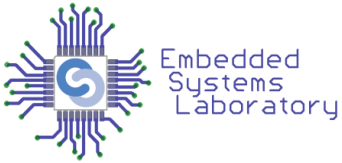


PROIECTAREA CU MICROPROCESOARE

ARHITECTURA X86

Facultatea de Automatică și Calculatoare
Universitatea Politehnica București

Contents

- Overview
- Hardware architecture
- Software architecture

Overview

- The name **x86 refers to instruction set** which is used in Intel's 8086 CPU
- This name is used in processors from Intel, AMD, VIA, and other processors that compatible with the instruction set
- Modern processors have many addition to the original x86 but still fully backward compatible with it.

Chronology (only Intel)

- 1978: 8086, 8088
  - 16-bit CPU with 20-bit address
- 1982: 80186, 80188
- 1982: 80286
  - 16-bit CPU with 24-bit address
- 1985: i386 (80386)
  - 32-bit CPU with 32-bit address, renamed to IA-32
- 1989: i486
  - Pipelining with integrated FPU(floating point unit)

Chronology (cont.)

- 1993: Pentium, Pentium MMX
  - 64-bit data bus, superscalar (2 ALUs)
- 1995: Pentium Pro
  - 36-bit address bus (Physical Address Extension, PAE), RISC core
- 1997: Pentium II, Pentium III
  - SSE
- 2000: Pentium IV
  - Deeply pipelining, SSE2, Hyperthreading
- 2003: Pentium M

Chronology (cont.)

- 2003: Athlon64 (from AMD)
  - First 64-bit CPU in x86 family
- 2004: Pentium 4 Prescott
  - Very deep pipeline (20 stages), SSE3, 64-bit
- 2006: Core 2
  - Multicore, SSE4
- 2008: Atom
  - Deep pipeline, very low power

Chronology

- 2008-2009: Intel Nehalem
  - 44-bit address, SSE4a, HyperTransport3, L3-cache on dye
- 2010: Intel Sandy Bridge
  - SSE5

Basic properties

- CISC design
- Primarily two-address with limitation
- Little endian
- Allow unaligned memory-address access for 16 and 32 bits
- Instruction set is mostly inherited from the previous generation Intel’s CPU (8085)

Current implementation

- Since Pentium Pro, the core unit is RISC
- An extra decoding unit asynchronously splits most x86 instructions into smaller pieces, which then are executed by many execution units
- The extra decoding unit operates like an emulator

Operation mode

- Many extensions have been introduced since the first generation of x86
- The original x86 architecture has many limitations and cannot support modern architecture such as protection and virtual memory
- How to support new requirements while maintain backward compatibility?

Operation mode

- x86 CPUs have many operation modes
  - Real mode
  - 16-bit Protected mode
  - 32-bit Protected mode
    - Virtual-8086 mode
  - Long mode (64-bit Protected mode)
- Different modes have different limitations
- After power up, CPU operates in Real mode.
- Switching between modes occurs by software (OS)

Real mode

- Addressing is fully compatible with the original x86 CPU
- Can access memory up to 1MB
- Uses segmentation
- Although the memory reference is limited to the original 16-bit CPU, data processing can utilize 32-bit instructions

Segmentation in Real mode

- The original x86 is 16 bits which has 16-bit registers.
- 64 kB memory was nearly not enough
- To extend memory size, Intel decided to use two 16-bit number to point to a physical address
- That two 16-bit number is referred to in the form of **Segment:Offset**
- The combination of segment and offset generates 20-bit address (1 MB)

Calculating address from Segment:Offset

- The segment must be stored in one of the “segment” registers
  - CS, DS, SS, ES
- Given a 16-bit selector and a 16-bit offset, the 20-bit address is computed as follows
  - Multiply the selector by 16
    - This simply transforms XXXX into XXXX0 (in base-16)
  - Add the offset

Example

- Given the segment and offset as 10DE:12A3, what is the physical address?
  - Physical = Segment * 16_{10} + offset
  - = 10DE0 + 12A3
  - Physical = 12083

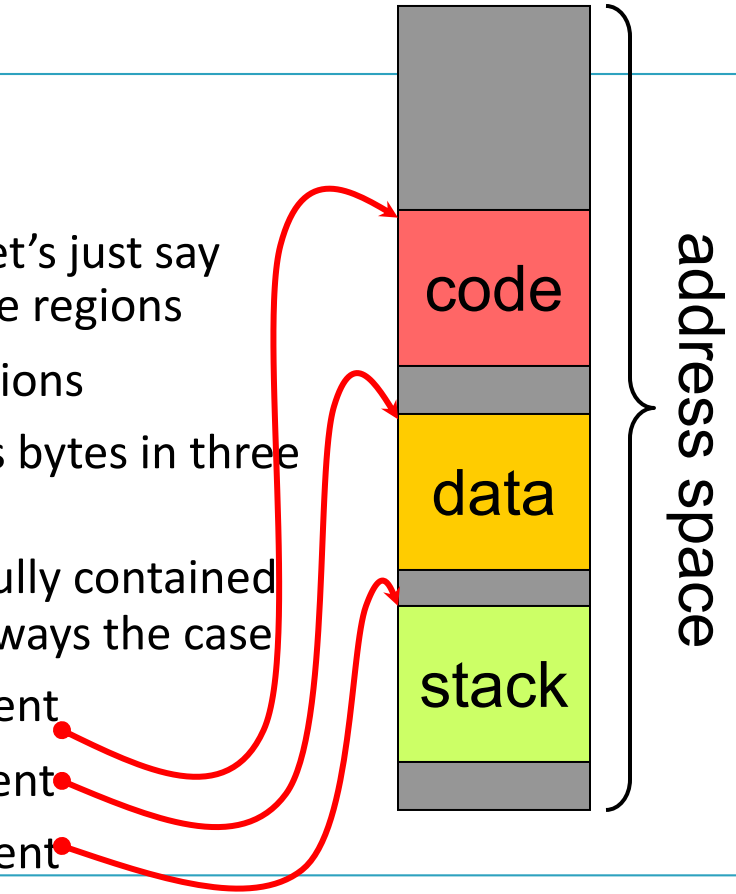
Example

- Consider the byte at address 13DFE with the segment value 10DE. What is its offset?

  - 13DDE = 10DE * 16_{10} + offset
  - offset = 13DFE - 10DE0
  - offset = 301E (a 16-bit quantity)

Code, Data, Stack

- Although we'll discuss these at length later, let's just say right for now that the address space has three regions
- A program constantly references all three regions
- Therefore, the program constantly references bytes in three different segments
  - For now let's assume that each region is fully contained in a single segment, which is in fact not always the case
- CS: points to the beginning of the code segment
- DS: points to the beginning of the data segment
- SS: points to the beginning of the stack segment



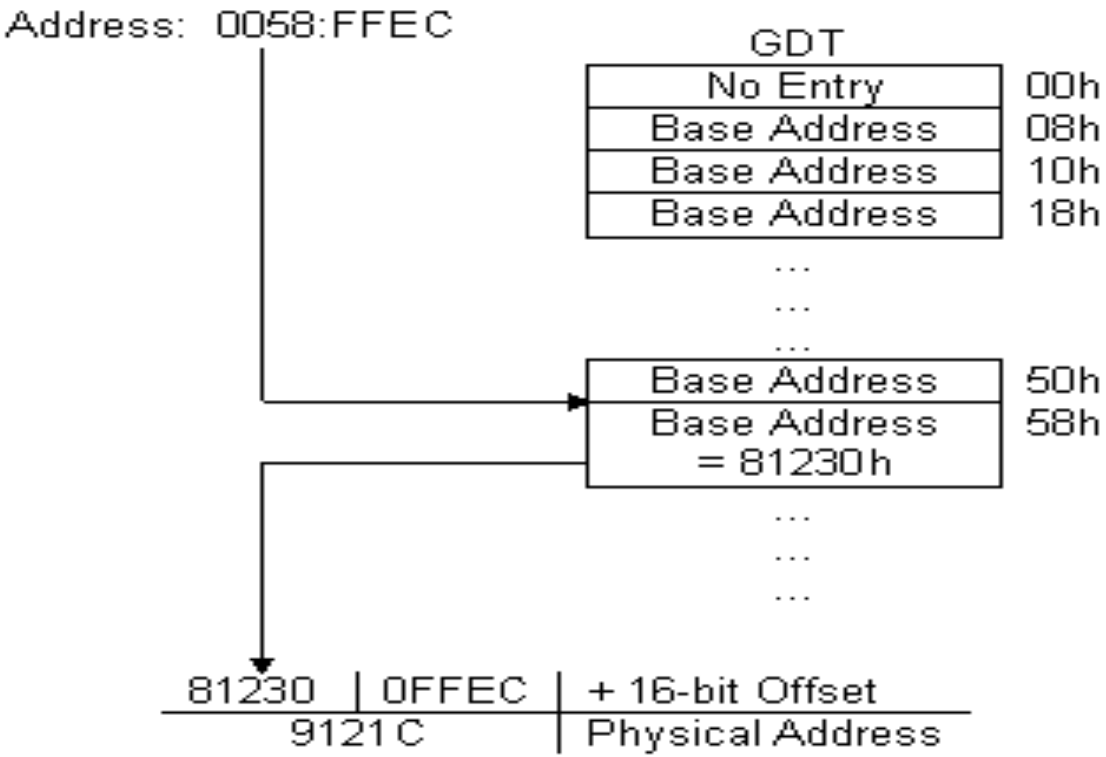
The trouble with segments

- It is well-known that programming with segmented architectures is really a pain
  - You constantly have to make sure segment registers are set up correctly
- What happens if you have data/code that's more than 64K?
  - You must then switch back and forth between selector values, which can be really awkward
- Something that can cause complexity also is that two different (selector, offset) pairs can reference the same address
  - Example: (a,b) and (a-1, b+16)
  - Concrete example: A000:5677 and A111:4567

Protected mode

- Introducing the term “logical address space” to x86
- Each position of memory has two address types:
  - Logical address – seen and used by program (also programmer)
  - Physical address – The actual address generated by memory management unit (embedded within the CPU) from a given virtual address
- Size of logical space can be different from those of physical space

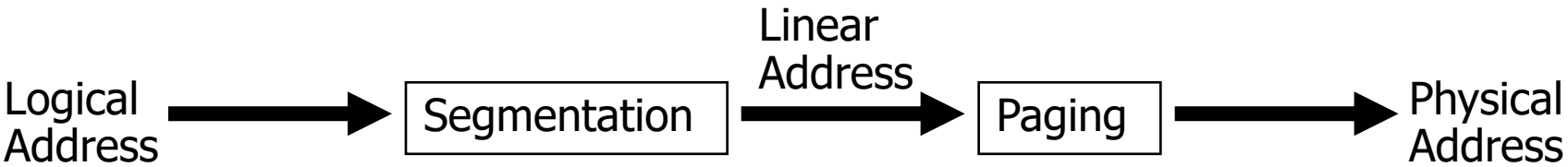
Logical -> Physical address



Difference of 286 and 386 address translation

- 286

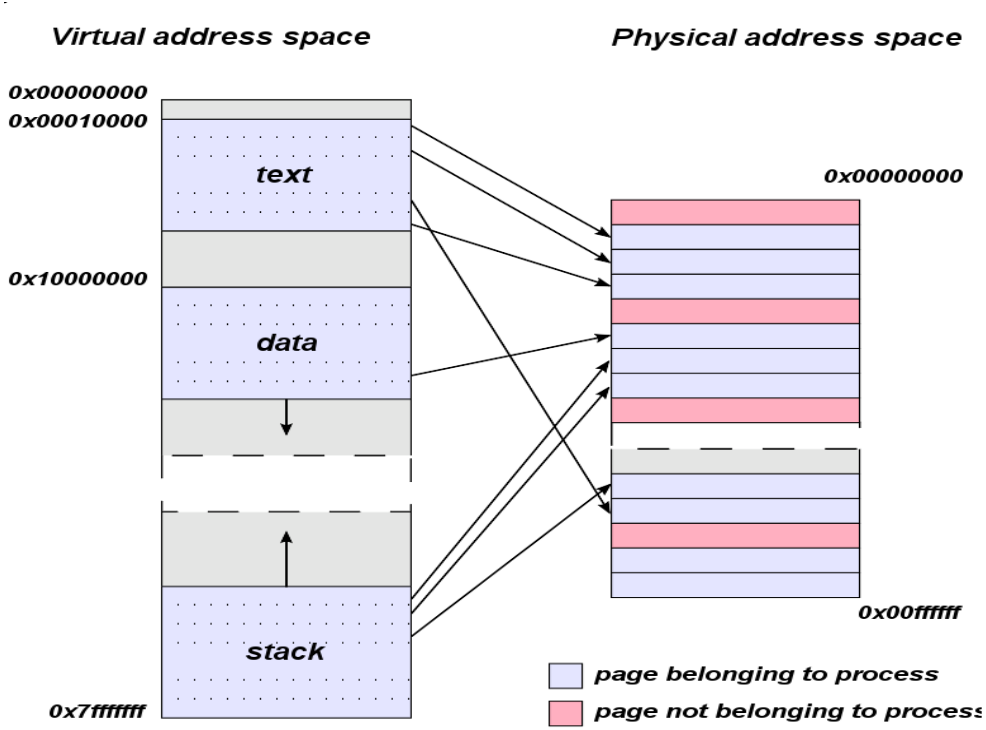
- 386



Virtual memory

- Paging in i386 introducing the world of virtual memory to x86
- A memory page can be absent from the physical memory
- If program refers to an absent page, paging unit generates a special interrupt (page fault) then OS can manage to fill the page with the required data

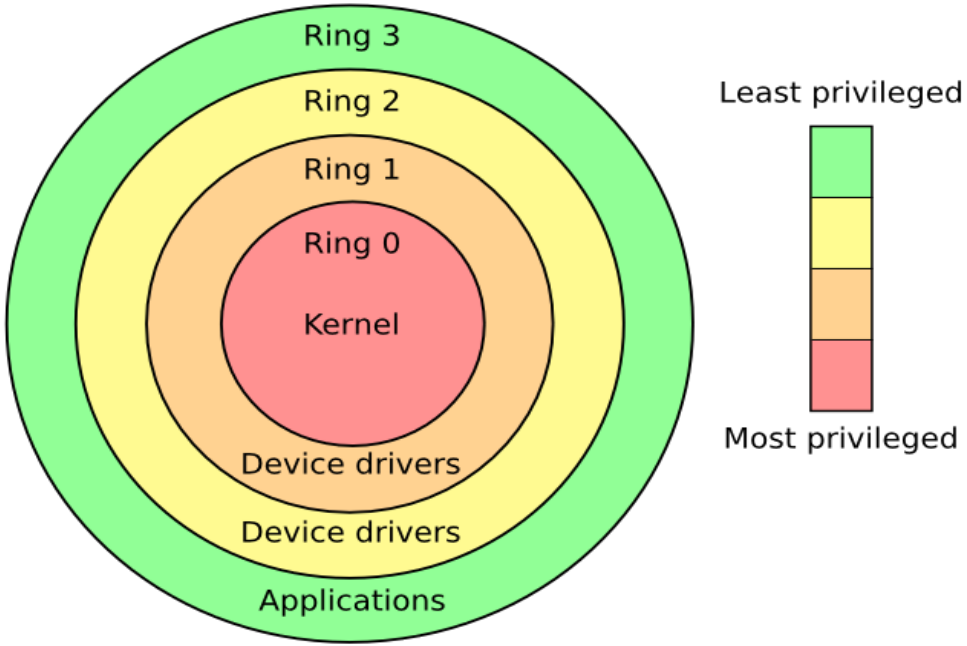
Virtual and physical space



Process privilege

- In protected mode, a process (or a task) has privilege (aka. Priority) level.
- There are 4 privilege levels (or rings) numbers from 0 to 3 (ring 0 is the highest privilege and ring 3 is the lowest)
- Usually, OS runs in ring 0 and all user applications operate in ring 3

Privileged ring



User applications cannot:

- Segment arithmetic
- Privileged instructions
- Direct hardware access
- Writing to a code segment
- Executing data
- Overlapping segments
- Use of BIOS functions, due to the BIOS interrupts being reserved
- **However, most legacy programs violate these rules!!!! To solve this problem, Virtual-8086 was introduced in i386**

Virtual 8086 mode

- As the segmentation in protected mode is far different from that in real mode, all legacy applications cannot run in protected mode.
- At the time of designing 80386, these old programs were still popular though the protected mode in 80286 had been around for sometime.
- Virtual-8086 mode is actually a sub-operation in 32-bit protected mode

Difference between protected and virtual-8086 modes

- Virtual-8086 mode is activated per process not for the whole system. Therefore, CPU is still in 32-bit protected mode and Virtual-8086 application is running in ring 3.
- In Virtual-8086, CPU uses real-mode segmentation instead of normal protected-mode segmentation
- Paging is still active so virtual memory and protection still exist.
- I/O operation is controlled by I/O permission map to allow/disallow the I/O change

Long mode

- Normally, applications cannot use 64-bit features such as 64-bit registers. Long mode enables these features.
- There is no virtual-8086 and segmentation in Long mode. However, paging is still active.
- Processor should enter 32-bit protected mode before starting Long mode.
- In Long mode, a task can be classified to either of two sub-mode:
  - 64-bit mode – For new 64-bit applications (old applications require re-compilation)
  - Compatibility mode – For 32-bit applications

x86 Registers

- The original 8086 has 14 registers. All of them are 16-bit wide.
  - 4 General purpose register (GPR): AX, BX, CX, DX
  - 2 Index register: SI, DI
  - 2 Pointer register: BP, SP
  - 4 Segment register: CS, DS, SS, ES
  - Flag
  - Instruction Pointer
- Each GPR can be accessed as two separate bytes (i.e., BX's high byte can be accessed as BH and low byte as BL)

x86 Registers (cont.)

- In 32-bit platforms, all GPR, flag, index, and pointer registers are expanded to 32-bit. (Except the segment registers)
- All 32-bit registers are represented by prefixing “E” to the 16-bit register name (e.g., to access 32-bit AX, we use EAX)
- Two new segment registers (FS and GS) were added (They are still 16-bit)

x86 Registers (cont.)

- In 64-bit platforms (also in 64-bit Long mode), all 32-bit registers are expanded to 64-bit in a similar way that 32-bit expansion did before but with the prefix “R” (e.g., AX become RAX)
- 8 additional general registers were added (R8, R9, .., R15)

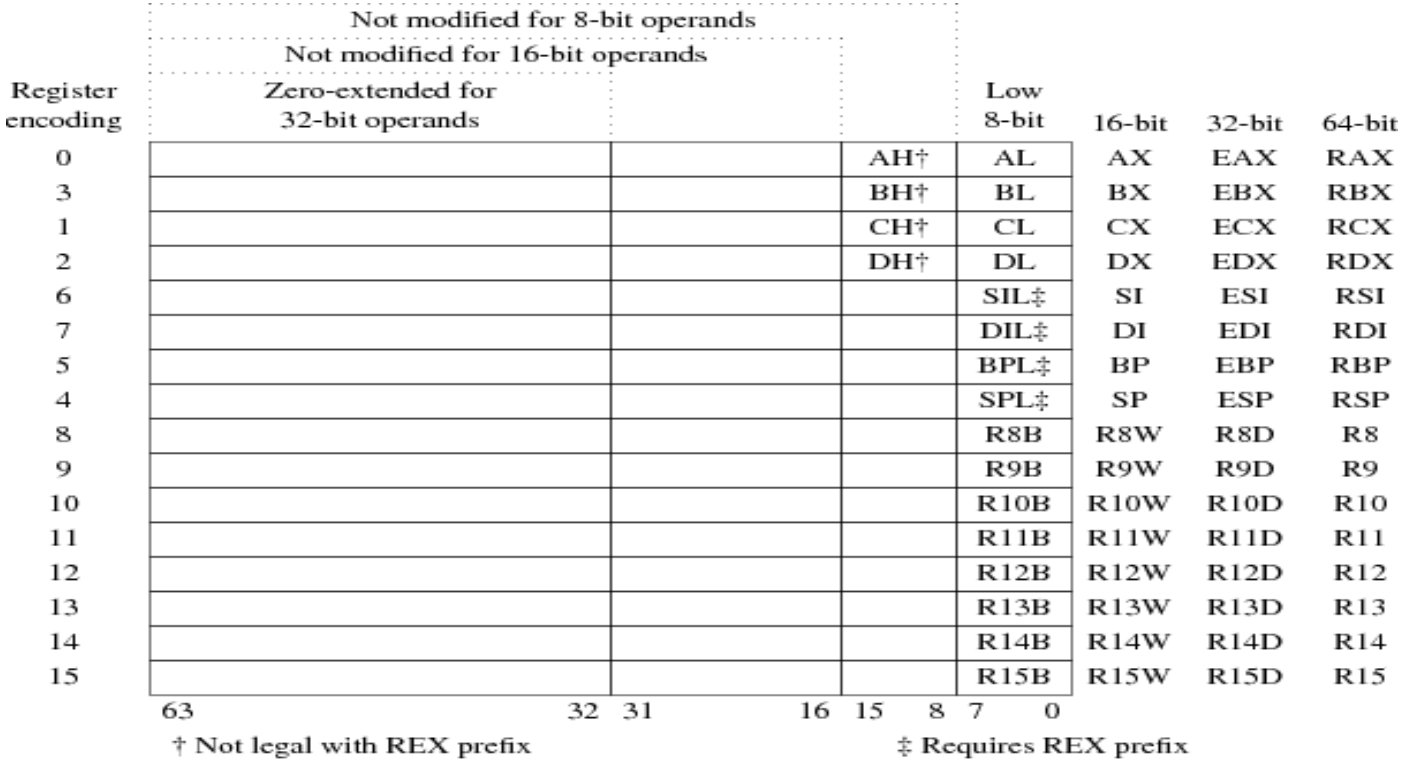
Floating-point Registers

- Floating-point unit has its own register groups separate from all operation of the other units
  - 8 80-bit floating-point registers: all are named ST
  - 3 16-bit word registers: Control word, Status word, Tag word
- Operation on these registers as well as the floating point functions require another set of instructions because this unit was originally a separate processor

Special registers

- Apart from all registers mentioned before, x86 also has various special/miscellaneous registers. They are mainly used by the OS
  - 5 Control registers: CR0 through 4
  - 6 Debug registers: DR0 through 3, plus 6 and 7
  - 4 Test registers: TR7 through 7
  - Descriptor registers: GDTR, LDTR, IDTR
  - Test register: TR

x86 Registers



MMX

- MMX is a SIMD instruction set
- MMX adds 8 new “registers” named MM0 to MM7. They are **not** actual registers. The are just aliases to the existing floating-point registers
  - Change MMx also change ST
- Each of MM registers is 64-bit. Therefore, the highest 16-bit in 80-bit floating point register is unused and filled with all 1.
- Each MM register can contain one 64-bit integer, two 32-bit integer, four 16-bit integer, or eight 8-bit integer.
- A MMX instruction calculates the results of these packed data simultaneously

3DNow

- Introduced by AMD in 1997
- Very similar architecture as MMX
- Each 64-bit register, can contain two single-precision floating point data. Hence, the operations can be performed only with real numbers

Streaming SIMD Extensions (SSE)

- SSE discarded all legacy connections to the FPU registers
- 8 new actual registers were added named XMM0 – XMM7 (in AMD64 the number of register has been increased from 8 to 16)
- Like 3DNow, SSE supports only floating-point operations
- Each XMM has 128 bits.
  - SSE1 each register can pack four single-precision floating-point data
  - SSE2 and later each register can pack four single-precision or two double-precision floating-point data
- SSE3 and SSE4 have same register structure with additional instructions

Physical Address Extension (PAE)

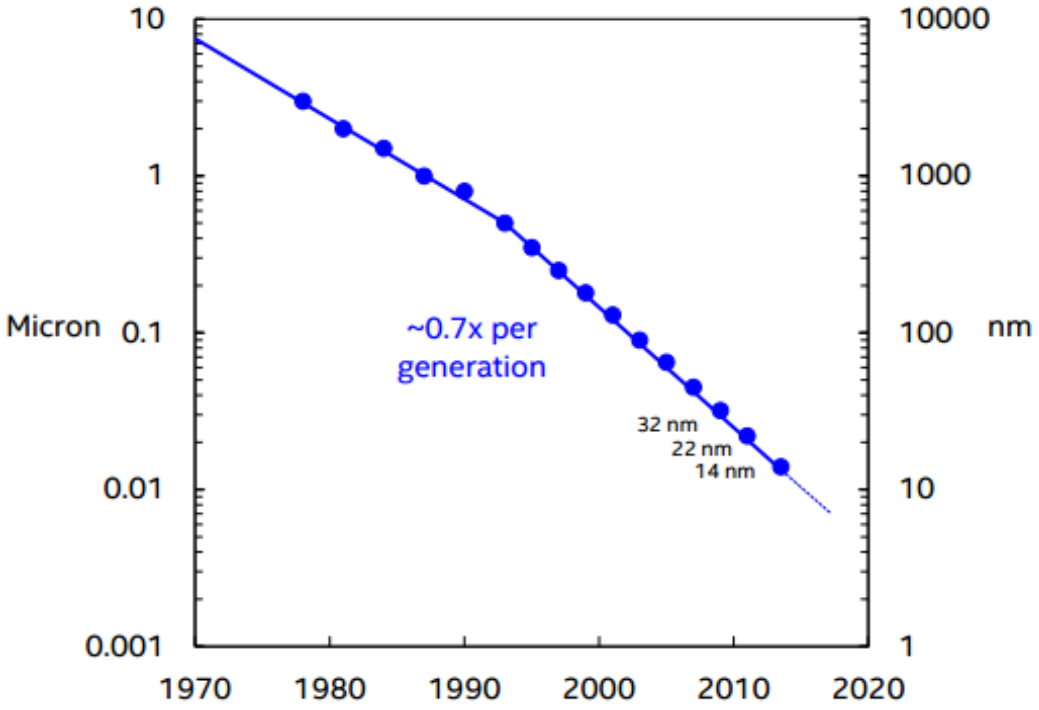
- Valid only in 32-bit protected mode
- The size of physical address is extended by 4 bit. Therefore, we can have maximum 64GB of physical memory
- The extension occurs in paging unit; hence, logical address still has 32 bits
  - Means that the size of memory that a program can access at a time is still 4GB

Data: Mark Bohr, Intel Sr. Fellow, IDF 2014

Intel Scaling Trend

22 nm | 14 nm | 10 nm | 7 nm

Manufacturing | Development | Research



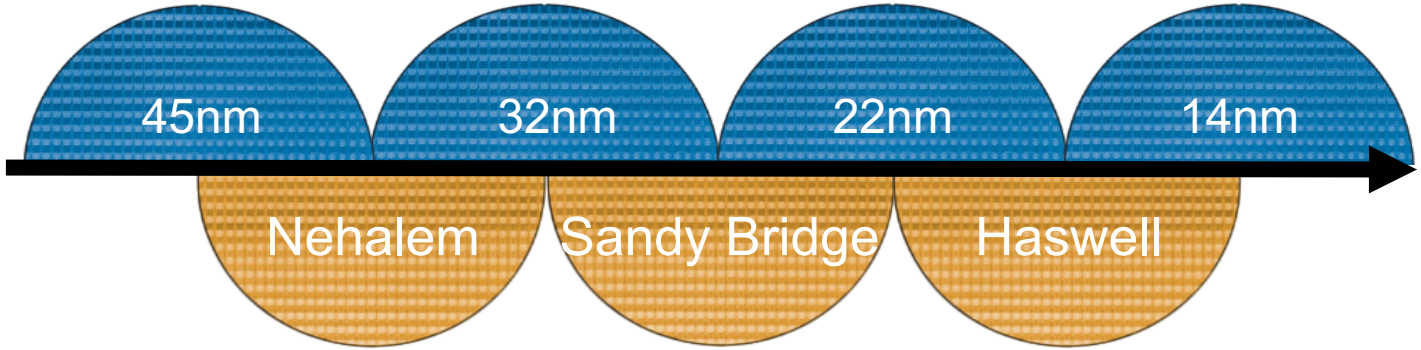
Scaled transistors provide:

- Higher performance
- Lower power
- Lower cost per transistor

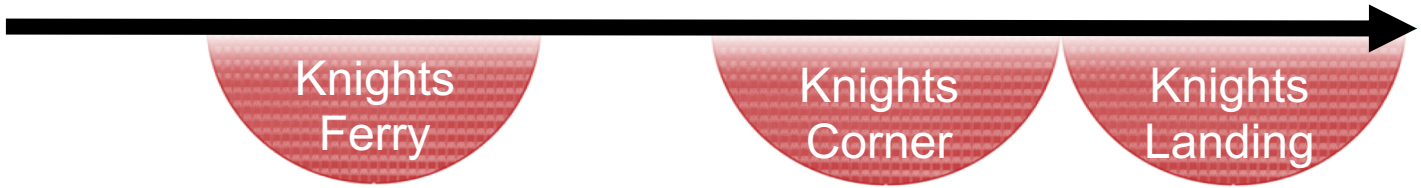
Moore's Law continues!

Tick-Tock Model (now defunct)

Scalable Performance
Energy Efficient
Microarchitecture



Highly Parallel
Energy Efficient
Architecture



NetBurst

- First released in 2000
- Hyper Pipelined Technology
  - 20-stage pipeline (compared to 10 in PIII) except for Prescott
  - Prescott has 31 stages!!!
  - Introducing of Hyper threading
  - Deep pipeline has high penalty in branching
- Rapid Execution Engine
  - ALU run at twice speed of other units
  - Speedup integer operations
  - Difficult to optimize operations among units
- Execution Trace Cache
  - When execute a previously visited instruction, CPU can use the previously decoded micro-ops instead of fetching and decoding the instruction again

Core Architecture

- First released in 2006
- Redesigned from ground up with the philosophy of Pentium M
- Only 14-stage pipeline but with wider execution unit.
- Micro-op Fusion
  - Combine two x86 instructions together and execute them simultaneously
- Multiple core CPUS with linked L1 cache and shared L2 cache
- Lower power consumption
  - 20% more performance for Merom at the same power level (compared to Dual Core)
  - 40% more performance for Conroe at 40% less power (compared to Pentium D)
  - 80% more performance for Woodcrest at 35% less power (compared to the original dual-core Xeon)

Nehalem

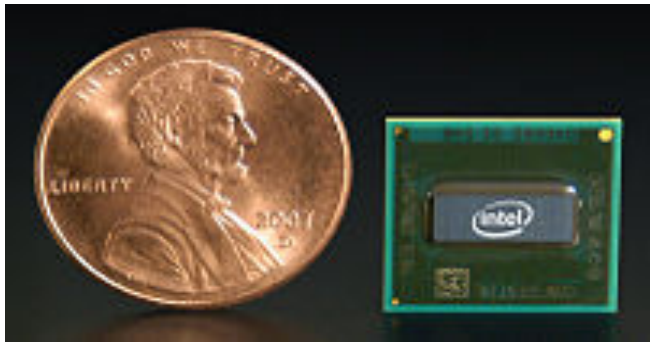
- Released 2008
- 2, 4, or 8 cores
  - 731 million transistors for the quad core variant
- 45 nm manufacturing process
- Integrated memory controller supporting DDR3 RAM and between 1 and 6 memory channels
- Integrated graphics processor (IGP) located off-die, but in the same CPU package
- A new point-to-point processor interconnect, the Intel QuickPath Interconnect, replacing the legacy front side bus
- Simultaneous multithreading, which enables two threads per core.
- Native (monolithic, i.e. all processor cores on a single die) quad and octo (8) core processors
- Linked L1 and L2 cache. Shared 8MB L3 cache
- 33% more in-flight micro-ops than Conroe
- 2nd level branch predictor and 2nd level Translation Lookaside Buffer
- Modular blocks of components such as cores that can be added and subtracted for varying market segments

Sandy Bridge

- Released in 2010
- 4 GHz clock speed.
- 4 to 8 out-of-order cores.
- Without SSE: 8 DP GFLOPS/core (2 DP FP/clock), 32-64 DP GFLOPS/processor.
- With SSE: 28 DP GFLOPS/core (7 DP FP/clock), 112-224 DP GFLOPS/processor.
- 32 KB L1 cache/core, (3 clocks).
- 512 KB L2 cache/core, (9 clocks).
- 2-3 MB L3 cache/core (8-24 MB total) (33 clocks), most likely pooled and dynamically allocated among the cores.
- 64 bytes cache line width.
- 256 bytes/cycle Ring bus bandwidth. The ring bus connects the cores.
- 0-512 MB GDDR / fast DRAM.
- 64 GB/s GDDR / fast DRAM memory bandwidth.
- 17 GB/s memory bandwidth per QuickPath link with 50 ns latency.

Atom

- Aims for ultra-low power mobile devices.
- No instruction re-ordering
- No instruction fusing
- Simpler micro-ops decoding than Nehalem (Less performance but smaller unit)
- Single core with hyper-threading



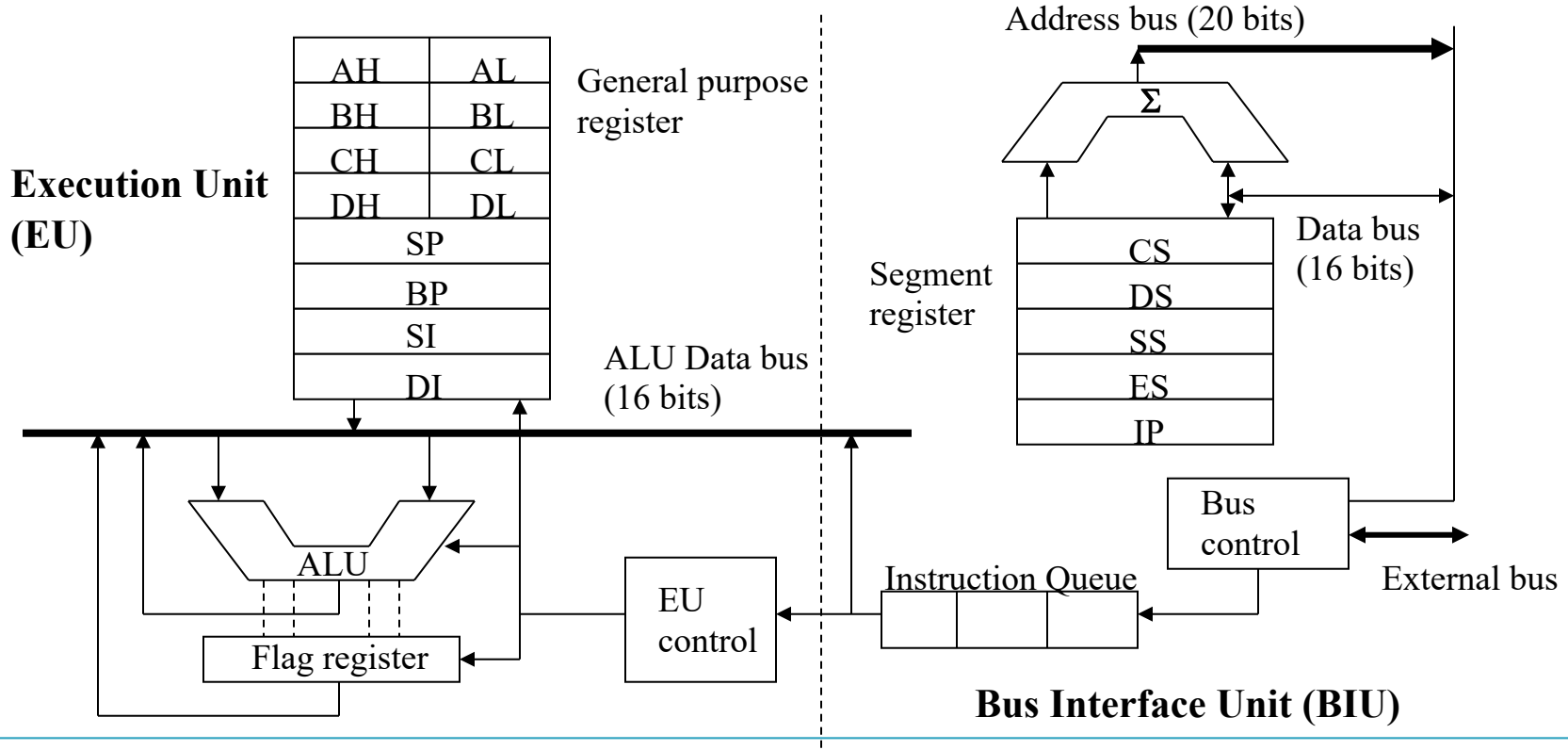
Haswell

- Introduced 2013
- 14 stage pipeline
- 4 cores, SMT machine
- In order issue, Out of Order execution, In order commit.
- Wider data paths and extra Store AGU to provide more bandwidth in AVX2 computations
- LLC/Ring is the point of coherence and distributed arbitration of requests.
- Intel TSX
  - Added support for Restricted Transaction Memory
- Integrated Graphics and Improved Power Management
  - Power Efficiency is a huge emphasis

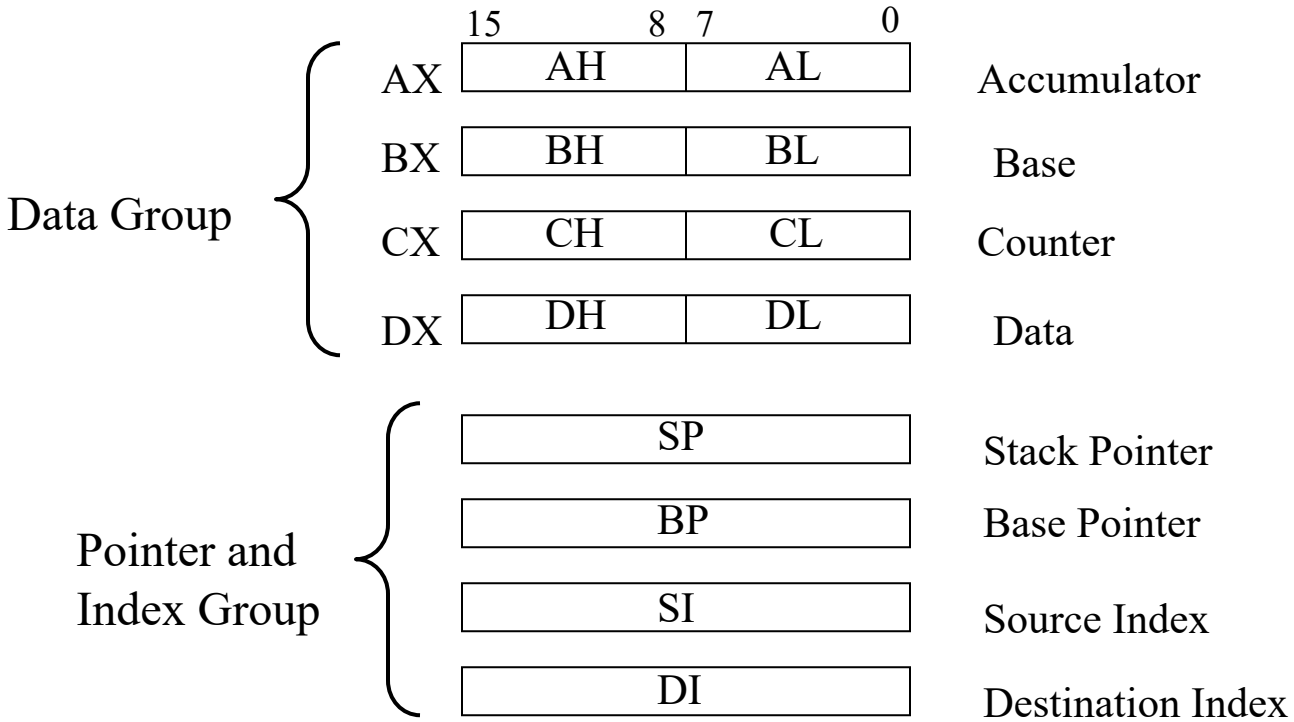
Skylake

- 6th generation Core uArch
- Released 2015
- Improved performance, particularly on mobile
- Better graphics support
- Power efficiency (4.5W at the low end)
- Better OS support (especially Win 10)

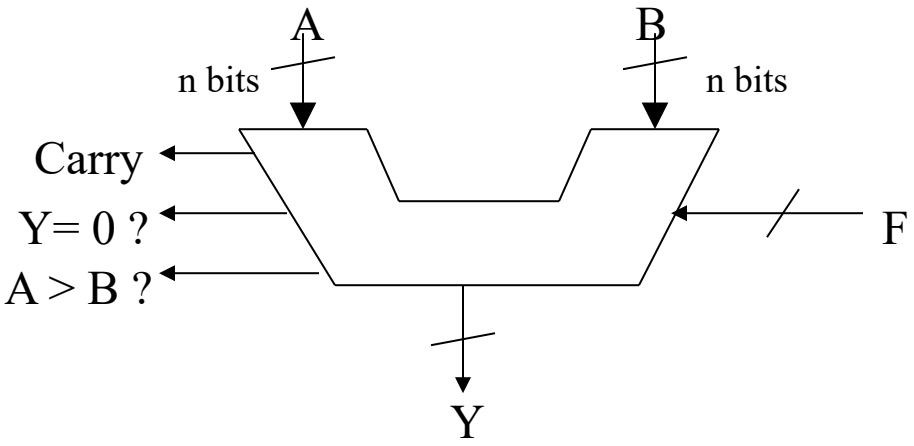
Organization of 8086



General Purpose Registers



Arithmetic Logic Unit (ALU)

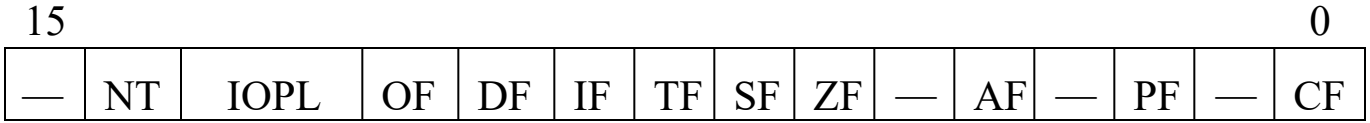


F	Y
0 0 0	A + B
0 0 1	A - B
0 1 0	A - 1
0 1 1	A ***and*** B
1 0 0	A ***or*** B
1 0 1	***not*** A
• • •	• • •

- Signal F controls which function will be conducted by ALU.
- Signal F is generated according to the current instruction.
- Basic arithmetic operations: *addition, subtraction, etc.*
- Basic logic operations: *and, or, xor, shifting, etc.*

Flag Register

❑ Flag register contains information reflecting the current status of a microprocessor. It also contains information which controls the operation of the microprocessor.



➢ Control Flags

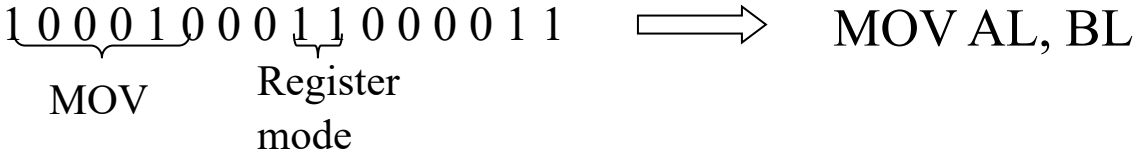
IF: Interrupt enable flag
DF: Direction flag
TF: Trap flag

➢ Status Flags

CF: Carry flag
PF: Parity flag
AF: Auxiliary carry flag
ZF: Zero flag
SF: Sign flag
OF: Overflow flag
NT: Nested task flag
IOPL: Input/output privilege level

Instruction Machine Codes

- Instruction machine codes are binary numbers
  - ***For Example:***



- Machine code structure

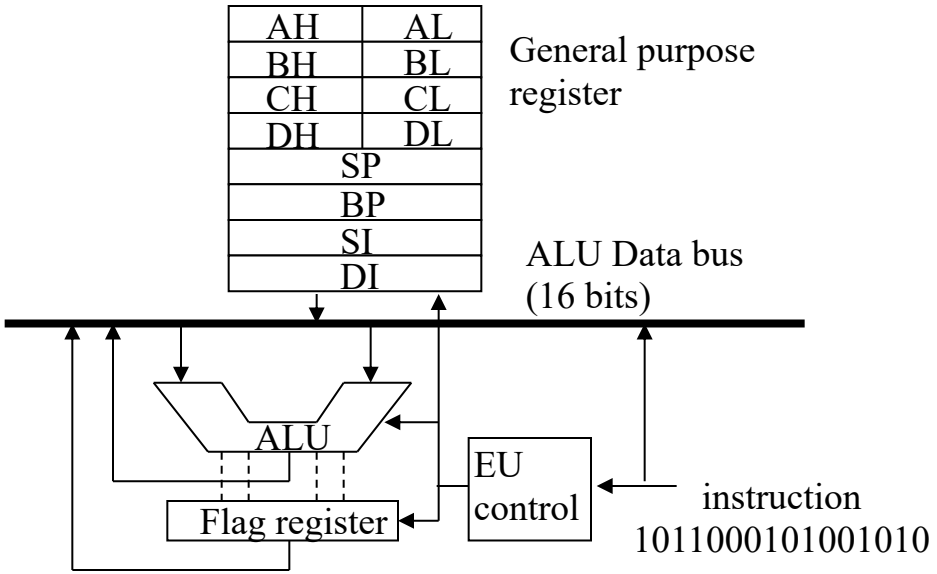
Opcode	Mode	Operand1	Operand2

  - Some instructions do not have operands, or have only one operand
  - Opcode tells what operation is to be performed.

    (EU control logic generates ALU control signals according to Opcode)
  - Mode indicates the type of a instruction: *Register type, or Memory type*
  - Operands tell what data should be used in the operation. Operands can be addresses telling where to get data (or where to store results)

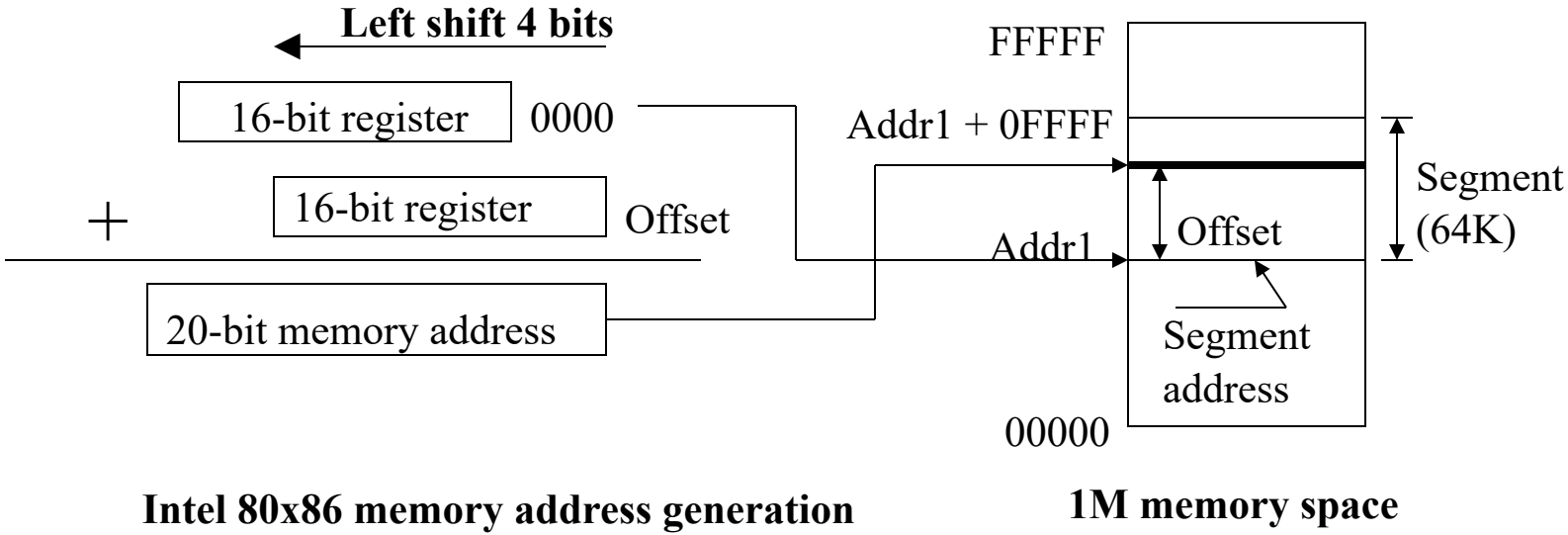
EU Operation

1. Fetch an instruction from instruction queue
2. According to the instruction, EU control logic generates control signals.
   (*This process is also referred to as instruction decoding)*
3. Depending on the control signal, EU performs one of the following operations:
   - An arithmetic operation
   - A logic operation
   - Storing data into a register
   - Moving data from a register
   - Changing flag register



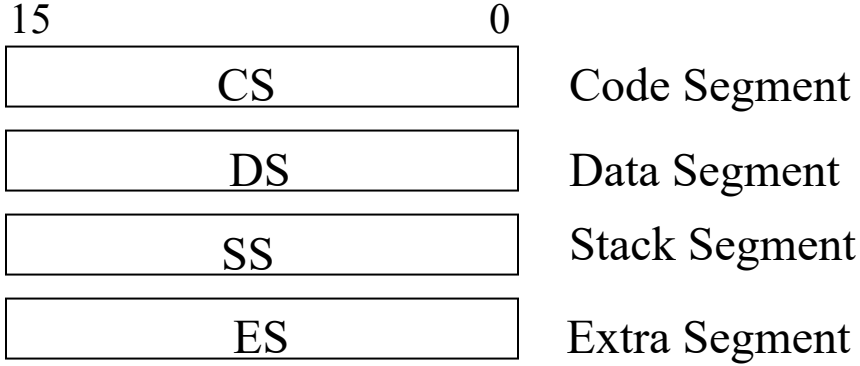
Generating Memory Addresses

- How can a 16-bit microprocessor generate 20-bit memory addresses?



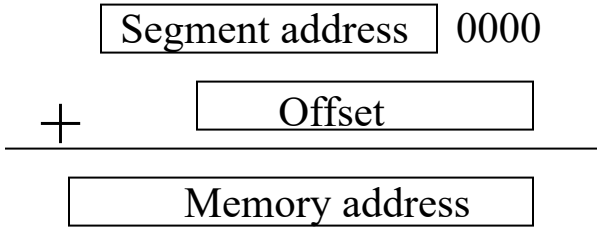
Memory Segmentation

- A segment is a 64KB block of memory starting from any 16-byte boundary
  - For example: 00000, 00010, 00020, 20000, 8CE90, and E0840 are all valid segment addresses
  - The requirement of starting from 16-byte boundary is due to the 4-bit left shifting
- Segment registers in BIU



Memory Address Calculation

- Segment addresses must be stored in segment registers
- Offset is derived from the combination of pointer registers, the Instruction Pointer (IP), and immediate values
- Examples



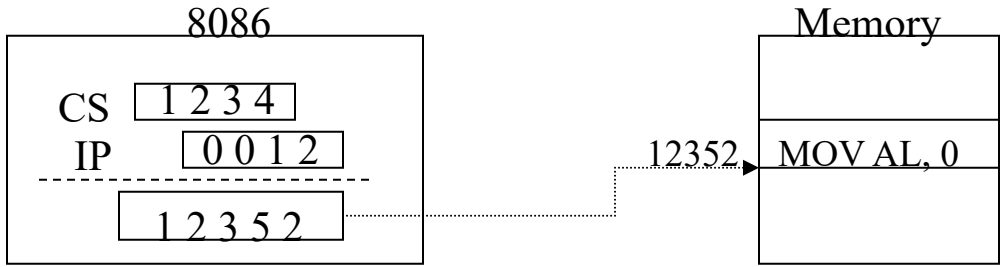
CS	3	4	8	A	0
IP +		4	2	1	4
Instruction address	3	8	A	B	4

SS	5	0	0	0	0
SP +		F	F	E	0
Stack address	5	F	F	E	0

DS	1	2	3	4	0
DI +		0	0	2	2
Data address	1	2	3	6	2

Fetching Instructions

- Where to fetch the next instruction?



- Update IP
  - After an instruction is fetched, Register IP is updated as follows:

IP = IP + Length of the fetched instruction

  - For Example: the length of **MOV AL, 0** is 2 bytes. After fetching this instruction, the IP is updated to 0014

Accessing Data Memory

- There is a number of methods to generate the memory address when accessing data memory. These methods are referred to as **Addressing Modes**
- Examples:
  - *Direct addressing:* **MOV AL, [0300H]**

DS	1	2	3	4	0	*(assume DS=1234H)*
		0	3	0	0	
Memory address	1	2	6	4	0	

  - *Register indirect addressing:* **MOV AL, [SI]**

DS	1	2	3	4	0	*(assume DS=1234H)*
		0	3	1	0	*(assume SI=0310H)*
Memory address	1	2	6	5	0	

Reserved Memory Locations

- Some memory locations are reserved for special purposes. Programs should not be loaded in these areas

  - Locations from FFFF0H to FFFFFH are used for system reset code

  - Locations from 00000H to 003FFH are used for the interrupt pointer table
    - It has 256 table entries
    - Each table entry is 4 bytes

    $256 \times 4 = 1024$ = memory addressing space From 00000H to 003FFH