Berner Fachhochschule
Haute école spécialisée bernoise
Bern University of Applied Sciences

CS Basics
3.1) Memory

E. Benoist & C. Grothoff

Fall Term 2018-19

Computer Science Division

Memory

- Introduction: memory?
- Real mode flat model
  - Registers
- The three assembly programming models
  - Real Mode Flat Model
  - Real Mode Segmented Model
  - Protected Mode Flat Model

Introduction: memory?

Location, Location, Location

- **Normal course in Assembly language**
  - Instruction set
  - Present machine instructions one by one
  - Very moronic
- **Even if you have learned every single instruction in an instruction set, you haven't learned assembly language**
  - Any idiot can learn machine instructions
- **The real job of CPU: locating the instructions and data in memory**
  - The skill of assembly language consits of a deep comprehension of memory addressing

Different memory models

- **Three major memory models for x86 computers**
  - Using Linux you will use only one model
  - Two of them are history
  - But we need history to understand the present
- **Real mode flat model**
  - Oldest and ancient memory model
  - Fossilized but straitforward
- **Real mode segmented model**
  - retired model
  - the most hateful thing you will ever learn in any kind of programming
  - DOS programming used it
- **Protected mode flat model**
  - The memory model behind modern OSes
  - Windows 2000/XP/Vista/7/8 and Linux
  - It works only on CPUs after 386 (not on 8086, 8088, 80286)

Strategy for this course

- **First present Real mode flat model**
  - Very easy
- **Then Real mode segmented model**
  - You need to understand it
- **Finally the Protected mode flat model**
  - Where real work is done nowadays

Real mode flat model

Back in history

- **1974: Intel invented 8080 CPU**
  - Was a 8 bits CPU
  - It had 16 address lines coming out of it
  - Most machines had 4kB to 8kB
  - 16 addresses lines can address up to 64k addresses
  - Each Byte had one address (true even on 16 bits, 32 or 64 bits machines)
- **OS used was CP/M-80**
  - The OS existed at the top of installed memory
  - Sometime to be contained in ROM
  - but mostly to get out of the way and allow a consistent memory starting point for transient programs
- **When CP/M-80 starts a program**
  - Read it from disk
  - load the program into low memory at address 0100H (i.e. 256 bytes from the very bottom of memory)
  - The first 256 bytes were the program segment prefix (PSP)

The 8080 memory model

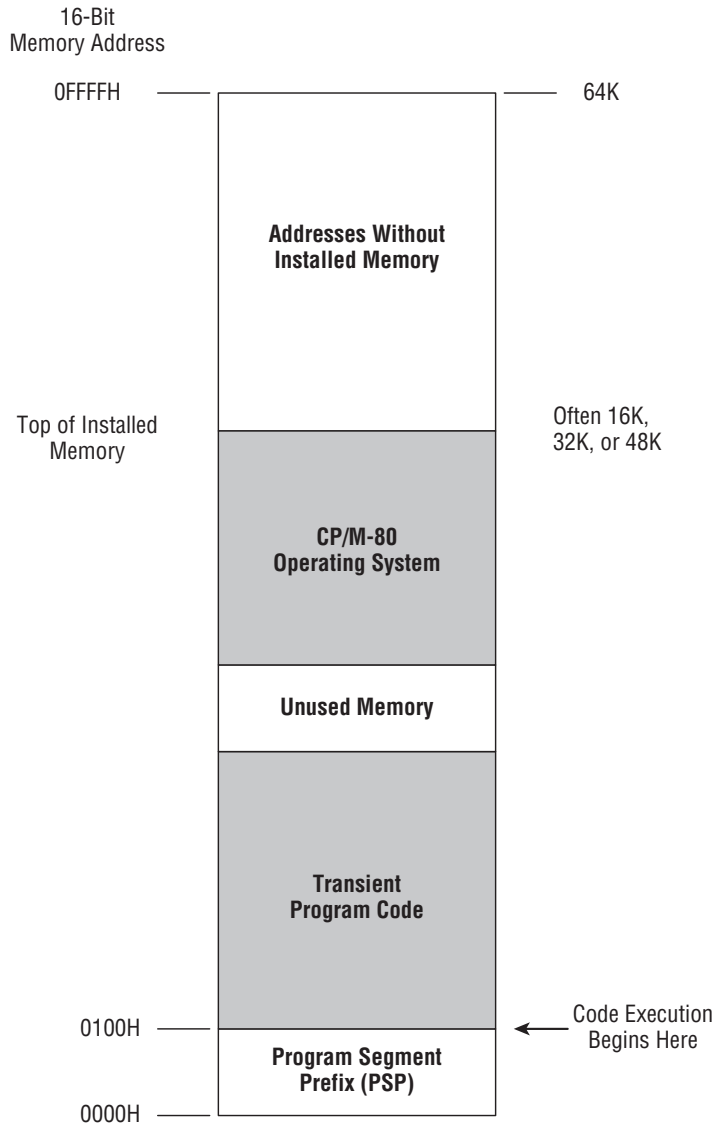

Figure 4-1: The 8080 memory model

8086 the first 16 bits CPU

- **Intel wanted to make it easy to translate programs from CP/M-80**
  - Was compatible with older code
- **8086 had 20 lines of addresses**
  - Could address 16 times more memory than 64 kB (i.e. 1MB)
- **Use of segment registers**
  - It points to a place in memory where things begin
  - Then the 64kB of memory can be addressated

The 8080 memory model inside a 8086 memory system

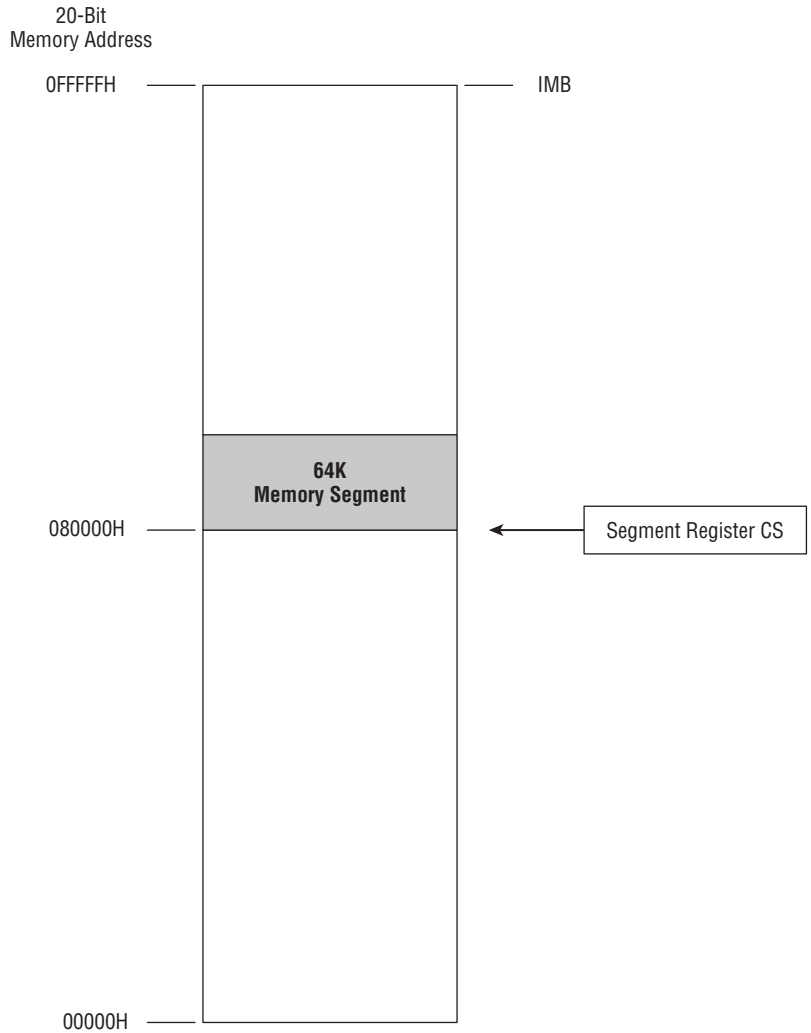

Figure 4-2: The 8080 memory model inside an 8086 memory system

One Megabyte of memory

- **One MB is 1 048 576 bytes**
  - Corresponds to 2^{20}
  - You had 20 address lines
  - You could address each byte in memory
- **16-Bits blinders**
  - the CPU can see a full megabyte
  - Must access to the megabyte using a 16 bits blinder
  - At one type it can only access 65 536 bytes

Seeing a megabyte through 64K blinders

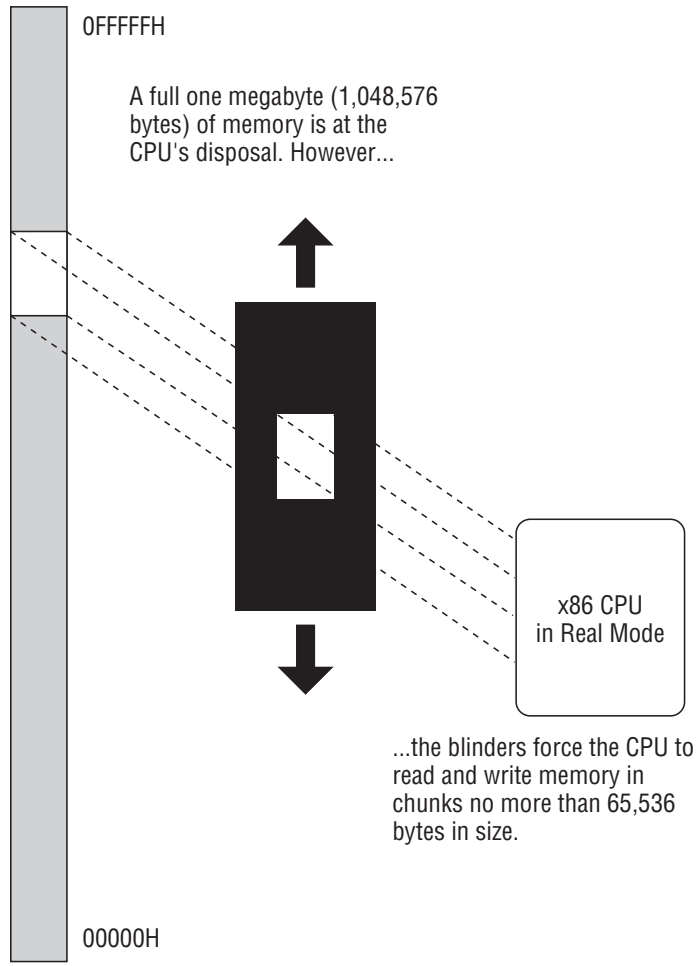

Figure 4-3: Seeing a megabyte through 64K blinders

Segments

- **A paragraph is a group of 16 bytes**
  - It is used to describe a place in memory where a segment begins
  - Every address in memory divisible by 16 is a *paragraph boundary*
  - It can be used to start a segment
- **Adresses**
  - first 0H
  - second 10H
  - third 20H
  - etc. (addresses ending with a 0 have the last four bits equal to 0)
- **Each segment can start at any paragraph boundary**
  - Can be each of the 64K (i.e. 65 535) locations
- **A segment can be 64kB long**
  - Not necessary, can be shorter

Making a 20-bit address out of 16-bit registers

- **Register**
  - Memory location inside the CPU
- **16-bit CPU's have registers with 16-bit**
  -
- **80386 and successors are 32-bit CPU's**
  - Nowadays, all CPU's are 64-bit
- **One big role for registers: holding addresses**
  - How to address 20 address pins if register contain 16 bit adresses?
- **Use two registers**
  - One for the segment
  - One for memory inside the segment

Example of access to the memory

- **Let us call a byte "MyByte"**
  - Its address is given 0001:0019
  - For segment 0001H, it is at position 0019H
  - As a convention we do not write H at the end of the two numbers
- **MyByte can have different addresses**
  - By changing the segment
  - 0:0029
  - 0002:0009

Segments and offsets

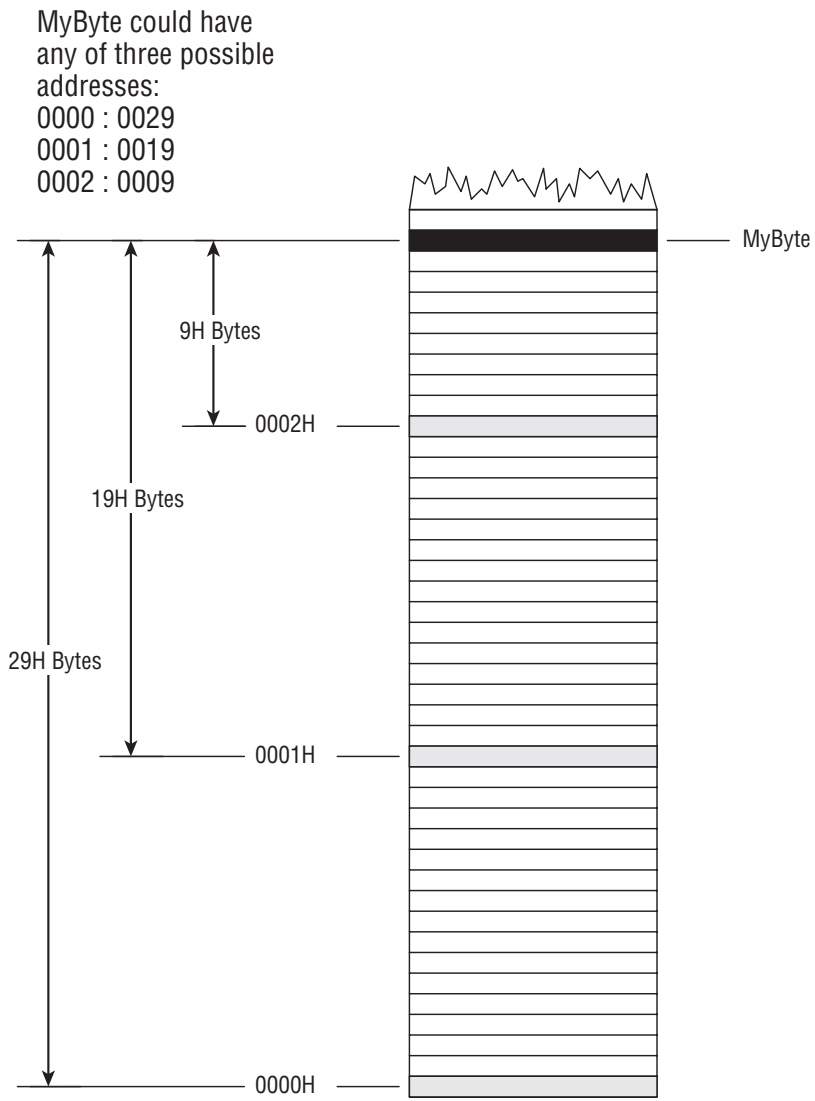

Figure 4-5: Segments and offsets

Registers

Registers for real mode segmented model

- **How to access elements in memory**
  - Move the blinder to one place
  - leave the blinder in place
  - Move inside the 64KB
- **Registers specifically design to hold segment addresses**
  - CS, DS, SS, ES, FS, and GS
  - All segment registers are 16 bits in size
  - Regardless of the CPU type (16, 32 or 64 bits)

Registers for segment addresses

- **CS stands for Code Segment**
  - segment address of the currently executing instruction
- **DS stands for Data Segment**
  - Variables or data are stored somewhere inside the segment
- **SS stands for Stack Segment**
  - The stack is a very important part of memory used for temporary storage
  - Stack has also a segment address
- **ES stands for Extra Segment**
  - Can be used to specify a location in memory
- **FS and GS are clones of ES**
  - Exist only in the 386 and later x86 CPUs

General Purpose registers

- **Segment registers are specialized**
  - Should not be used for something else
  - Plenty of other *General purpose registers*
- **Must be paired with segment addresses to pin down a location in memory**
  - Segment address + address in general register => address in memory
- **Can be used for other purposes**
  - Arithmetic manipulation
  - bit-shifting
  - many other things
- **Size of general purpose registers**
  - From 16 bits on 8086,8088,80286
  - to 32 bits since 386
  - and now 64 bits

General purpose registers

- **Memory location inside the CPU**
  - Like for segment registers
- **Are generalists**
  - share a large suite of capabilities
- **some registers have a "hidden agenda"**
  - Coming from limitations of the 16 bits processors
- **There are eight 16 bits registers**
  - AX, BX, CX, DX, BP, SI, DI, SP (SP is indeed not so general)
- **In 32 bits CPU, registers have 32 bits**
  - All 16 bits registers have been doubled
  - Prefixed with a E (Extended)
  - EAX, EBX, ECX, EBP, ESI, EDI, ESP

Extending 16-bit general purpose registers

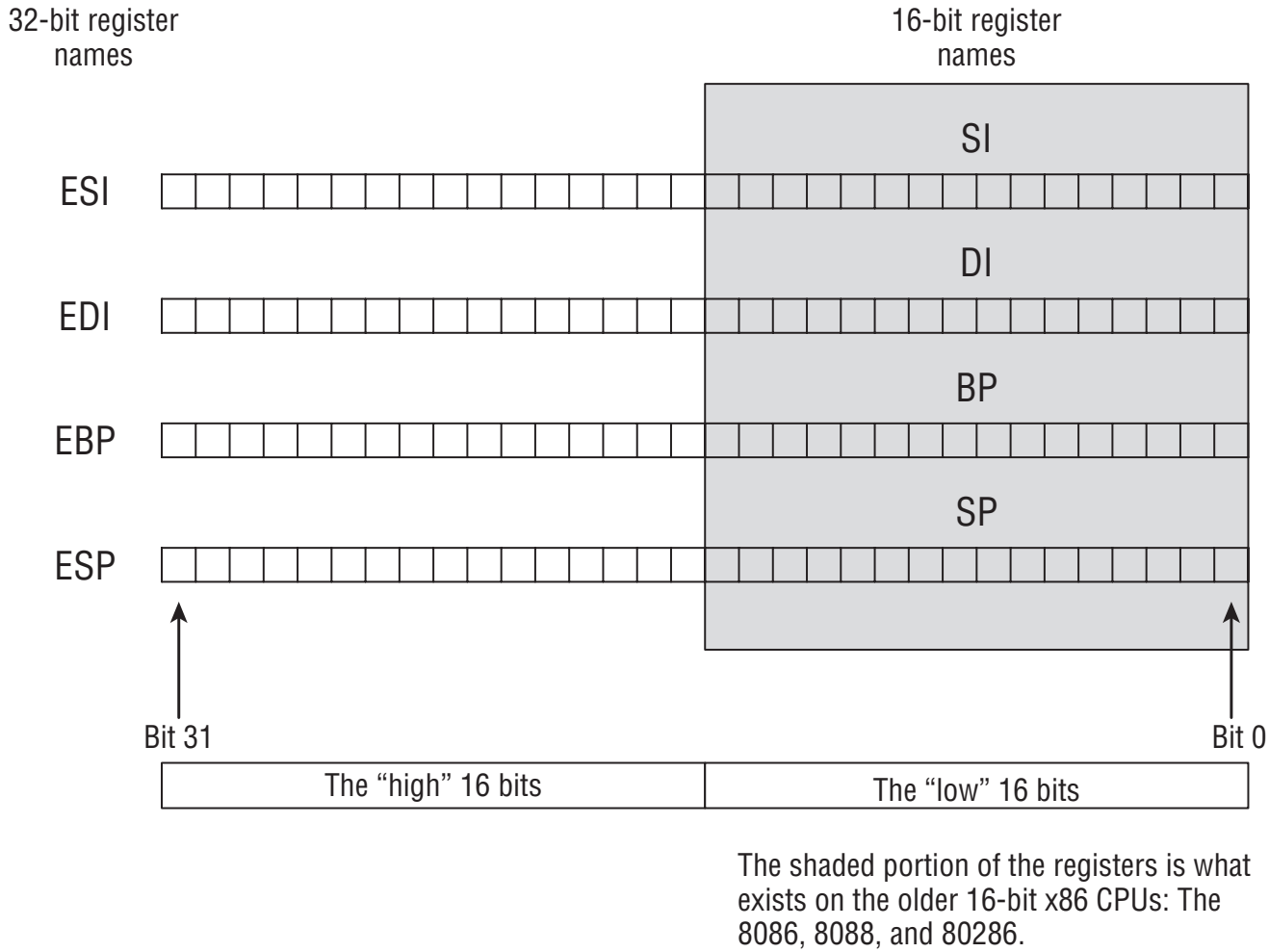

Figure 4-6: Extending 16-bit general-purpose registers

Register Halves

- **EAX, EBX, ECX and EDX have an additional twist**
  - The low 16 bits can also be divided in 8-bit registers
- **AX, BX, CX, and DX**
  - are the lower 16 bits of EAX, EBX, ECX, and EDX
  - They can also be devided in two halves
- **Special names for two halves**
  - A, B, C, and D remain
  - Postfix is H for high half
  - or L for low half
- **Example**
  - AX is composed of AH and AL
  - AH is the 8 high bits of AX
  - AL is the 8 low bits of AX

Registers Halves

- **You can change one half without touching the other**
  - Load 76E9H into register AX (16 bits)
  - You can read 76H from AH
  - E9H from AL
  - If you store 0AH in AL
  - Read AX, you get 760AH
- **Usefull**
  - Manipluating AL, AH, BL, BH, CL, CH, DL, DH as 8-bit registers
  - If you need a lot of 8-bit registers
  - Registers are much much fasters than memory access!
- **This system works only with general-purpose 16-bit registers**
  - AX, BX, CX, and DX
  - It does not work with SP, BP, SI and DI

Instruction Pointer

- **Called IP or EIP in 32-bit protected mode**
  - Is a specialist; can do only one thing
  - contains the offset of the address of the next machine instruction
  - In the current code segment
- **Code Segment**
  - The area (segment) in memory where machine instructions can be found
  - There may be many such code segments
  - Current code segment is stored in register CS
- **The CPU uses IP to keep track of where it is in the current code segment**
  - Each time an instruction is executed, IP is incremented by some number of bytes
  - The number of bytes of the instruction just executed
  - size of instructions varies from 1 to 6 bytes (sometime more)
- **CS and IP contain together the full address of the next instruction**

Size of the Instruction Pointer

- **16-bit CPUs**
  - 8088,8086,80286
  - IP is a 16-bit register
- **32-bit CPUs**
  - 386 and later
  - IP becomes EIP and is 32-bit
- **Real mode segmented model**
  - CS and IP give a 20-bit address
  - Access 1MB of memory
- **In the two flat models**
  - CS is set by the OS and held constant
  - IP does all the work
  - In 16-bit flat model (real mode flat model): IP can follow 64K (a segment)
  - In 32-bit flat model, 32 bits can address 4GB in memory
- **IP can neither be read from or written to**

Flags Register

- **FLAGS (16-bit) or EFLAGS (32-bit)**
  - Each bit can be used as a "bit-register" and be accessed separately
  - CF, DF, OF, ... (F is for Flags)
  - Each flag has a specific meaning for the CPU
- **Flags are used by machine instructions**
  - Many instructions will test the value of a flag
  - It influences their execution

The three assembly programmi models

Real Mode Flat Model

Real Mode Flat Model

- **CPU can see only one megabyte of memory**
  - segment:offset access to 20-bit addresses using two 16-bit registers
- **Program works inside one 64K memory block**
  - First version of Word Star
  - Turbo Pascal (including editor)
  - ...
- **Segment registers are all set to point to the begining of the 64K block of memory**
  - Set by the OS
  - They all point to the same place
  - You do not have to access segment registers

Real Mode flat model

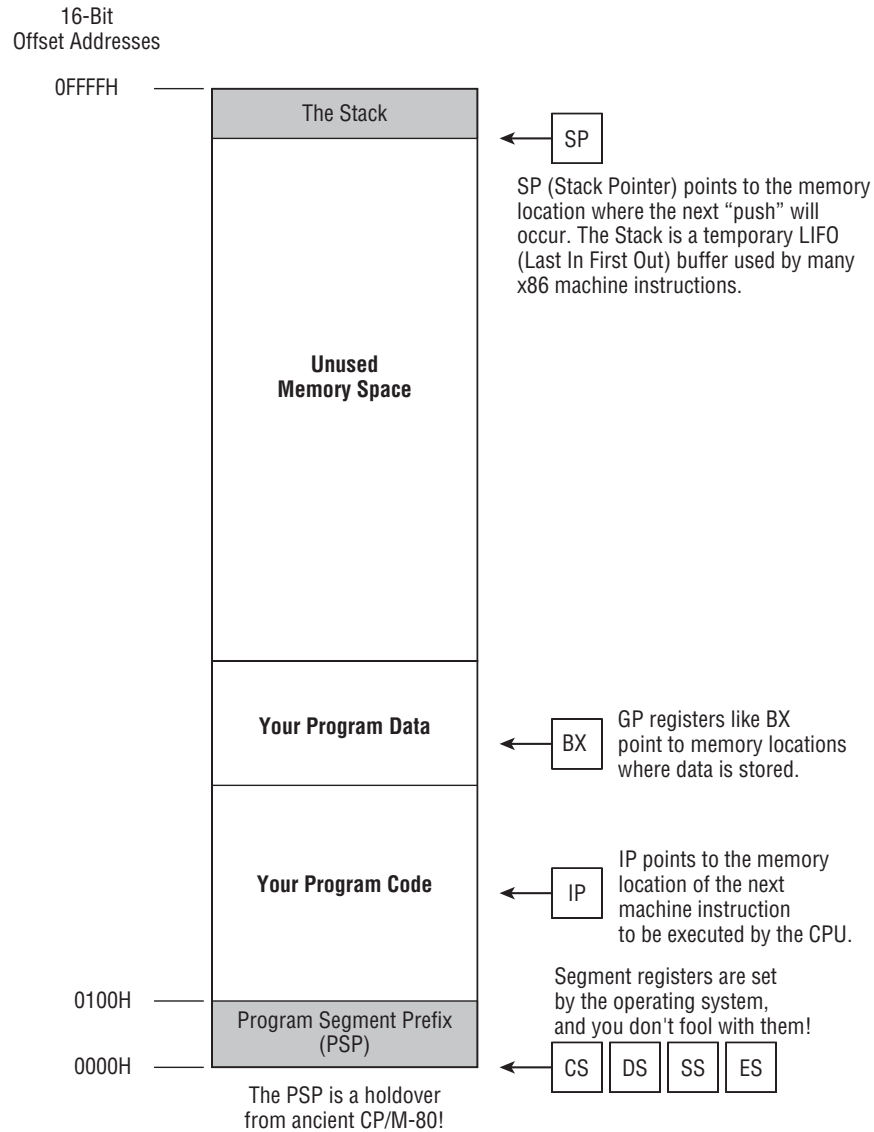

Figure 4-8: Real mode flat model

Real Mode Segmented Model

Real Mode Segmented Model

- **Was the mainstream for MS-DOS**
  - Complicated and uggly system
- **A program can see the full 1MB of memory**
  - combines 16-bit address and 16-bit segment
  - Generates a 20-bit address
  - The address of the first byte of a segment can be computed: 16 times the segment value
  - Segment 0000H starts with the address 0000H
  - Segment 0002H starts with the address 0002H * 10H = 0020H
- **Notation: segment register : offset register**
  - SS:SP
  - SS:BP
  - ES:DI
  - DS:SI
  - CS:BX

Real Mode Segmented Model

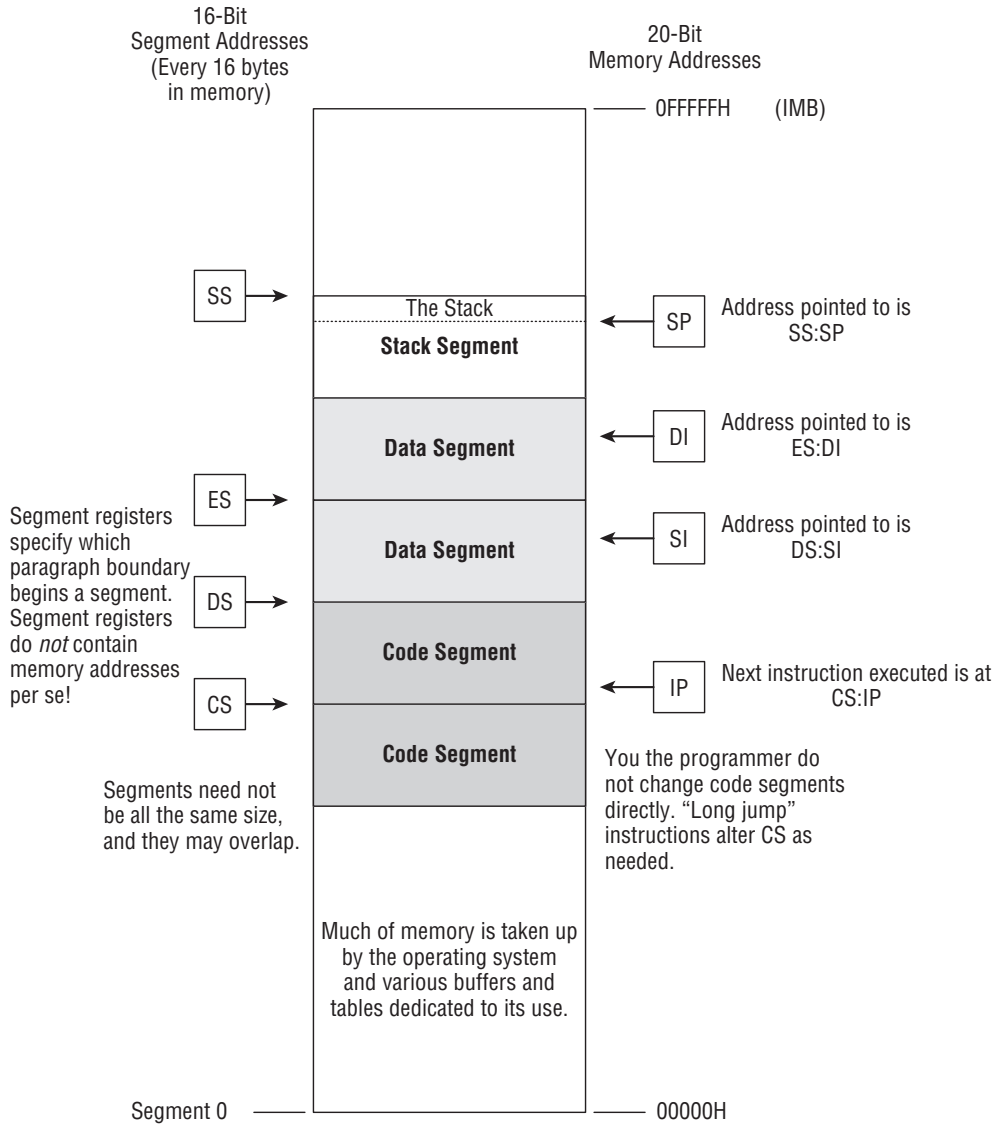

Figure 4-9: Real mode segmented model

Real Mode Segmented Model

- **Segments**
  - You can have any reasonable number of code and data segments
  - You can access two segments of each at the same time
- **Two data segments**
  - Two segment registers
  - DS, ES
  - FS and GS after 386
- **One single code segment register**
  - CS
  - You can not manipulate CS
  - Instructions called `jumps` will change the CS if necessary
- **One stack segment per program**
  - Start at SS
  - SP is the stack pointer, but goes down relative to SS

Problems with Real Mode Segmented Model

- **Pieces of the operating system are in memory**
  - Along with important system tables
- **You can destroy important part of the OS**
  - careless use of segment registers
- **New features needed:**
  - For CPU after 386
  - Protected mode for application programs
  - i.e. your programs
  - They can not destroy the OS
  - or other applications running on the system

Protected Mode Flat Model

Protected Mode Flat Model

- **Has been implemented by Intel since 386**
  - Was not supported by MS-DOS
  - Partially supported by Windows 95 et 98
  - Supported by Windows NT and successors
  - Supported by Linux from the begining
- **Programs are protected**
  - Your program sees a single 4GB memory block
  - The Segments are set by the OS, they are protected
- **All addresses have 32 bits**
  - All general purpose registers are 32-bit
  - They can only refere 2^{32} bytes of memory
  - They can only express 4'294'967'296 Bytes (i.e. 4 GB)

Protected Mode Flat Model

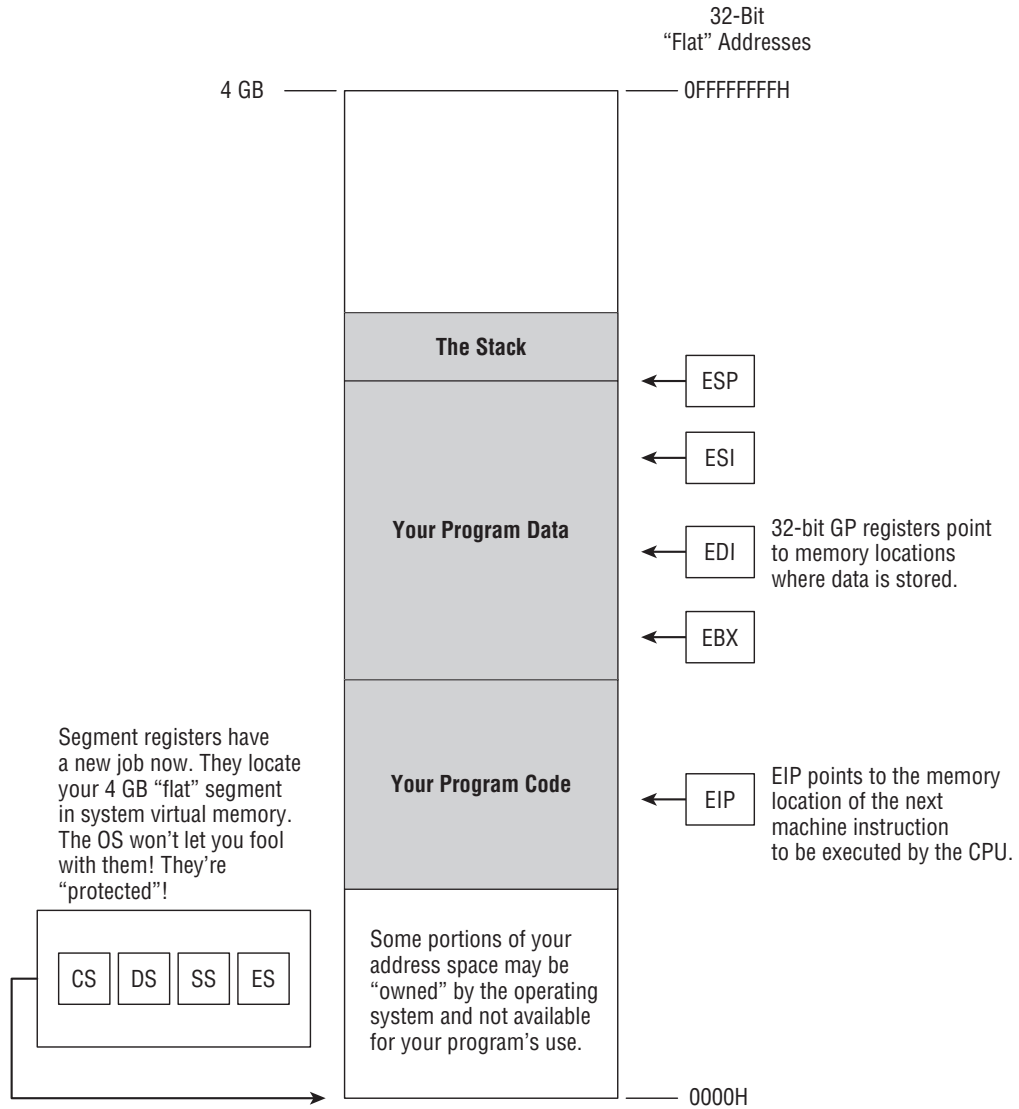

Figure 4-10: Protected mode flat model

Protected Mode Flat Model

- **You can not fool with Segment registers**
  - Only OS can
- **New job of segment register**
  - Define where the region of 4GB you can play with is
- **Each program receives maximum 4GB**
  - The Operating System can use "virtual memory" to extend the capacity of memory
  - It is a complex system where memory can be mapped on the disk
  - Not all the memory can be accessed (or even seen), depends on the OS

Limitations in protected mode

- **No access to Memory video**
  - It was possible to write in a place of memory, that was displayed on screen
  - Does not work anymore
  - Use OS and librearies instead
- **No access to Port hardware**
  - It was possible to write to memory location corresponding to serial and parallel ports
  - Or to read from it
  - We have to call the OS libraries to do so

64-bit “Long Mode”

- **Every CPU is now 64-bit**
  - Both windows and Linux have 64-bit versions of the OS
- **In long mode**
  - All registers are 64-bit
  - All machine instructions that act on 64-bit operands are available.
- **All registers in IA-32 have been extended to 64 bits in width**
  - Each register is renamed with a R
  - EAX becomes RAX,
  - EBX becomes RBX,
  - ...
- **8 new 64-bit general purpose registers**
  - R8, R9, ...R15

Memory with 64-bits

- **Registers for addresses are 64-bit**
  - can address 2^{64} bytes of memory
  - Not all memory pins are connected in the CPU
  - 48 address pins are implemented for virtual memory
  - 40 address pins for real memory
- **Sufficient to address 1TB of memory**

Conclusion

- **Information is stored in memory and registers**
  - Memory is slow
  - Registers are very fast
  - Memory is vast
  - Registers are rare
- **Different Memory models**
  - Protected Flat Model is used by Linux
  - Application can only access to a portion of the memory
  - The segment registers are manipulated by the OS
  - Memory can be accessed using registers:
  - RIP for the instructions
  - RSP for the stack
  - RBX, RDI, RSI, for data

Bibliography

- **This course corresponds to chapter 4 of the book:**
- Assembly Language Step by Step (3rd Edition)