Segmentation

- Memory address access/manipulation is a fundamental processor operation.
- It must be speeded up as much as possible
- Registers have very, very fast response times

$\leadsto$ Store Memory addresses in registers as much as possible

- Problem:
  1. Registers are 16 bits or 2 bytes
  2. Memory addresses are 20 bits wide
- Solution:
  - Could use 2 registers, but that would give us 2^{32} addresses; far more than are possible (2^{20})
  - Intel's address calculation is a little bizarre but very easy to calculate (for us and for the computer)

Address Calculation

- Intel does break up the address into two pieces (stored in two registers)
  1. Segment
  2. Offset
- Rule

  To calculate an address given its segment and offset, multiply the segment by 16_{10} and add the offset.

  This is the Intel Segmentation Scheme
- Easy to calculate:
  1. In binary, simply append 4 zeros to segment and add offset
  2. In Hex, Even more simply, append a single zero and add offset
- In Intels's 8086/80286 Scheme, two 16 bit numbers team up to form a 20 bit address

$\boxed{\text{Examples}}$

1. $1234 : 1ABC \rightarrow 12340 + 1ABC = 13DFC$
2. $ABCD : 0011 \rightarrow ABCD0 + 0011 = ABCE1$
3. $A79C : 4321 \rightarrow A79C0 + 4321 = ABCE1$
4. $FFFF : FFFF \rightarrow FFFF0{+}FFFF = 10FFEF$

   Too large? $10FFEF - 10000 = 0FFEF$

- Segments are ranges of memory that spread across the 1Mbyte address space. Maximum length of a segment is 64Kbytes
  1. Segment A = 40000 to 4FFFF
  2. Segment B = 7A000 to 82741
  3. Segment C = 1C400 to 28880
  4. Segment D = 17000 to 1FF00

     Segments C and D overlap
  5. Segment E = 1C400 to 28880
  6. Segments C and E occupy the exact same addresses

Four Segments that matter

Paragraphs: Segment Addresses must be divisible by 16: Last digit of a segment's 16bit hex address must be 0. That is, Segments must start at paragraph boundaries. How Conveenient.

- You only need to worry about 4 segments
- Those that correspond to the 80x86's segment registers
  1. CS: the code segment register
  2. DS: the data segment register
  3. SS: the stack segment register
  4. ES: the extra segment register
- During most of this course, we won't even consider what four segments are in use. The 64K bytes reachable in a single segment is a lot of storage capacity. typically a program "sets and forgets" segment register during initialization and only works with offsets.

The Processor architecture

- Two parts (refer to fig 2.2 in text)
  1. The Execution Unit (EU)
  2. The Bus Interface Unit (BIU)

EU : Actual execution of instructions

BIU : Sending/Receiving information on Data Bus

A **bus** is just a collection of electrical conductors that transmit signals through the machine

An 8-bit bus can transmit/receive 8 bits in parallel (at a time)

- All information; code and data, get to the processor thru the BIU
- Address Adder: Combines 16-bit segment with 16-bit offset into a 20 bit address

- BIU stores code and data in a **Prefetch Queue**
- If prefetch queue full; BIU must wait for EU to process some bytes
- If prefetch queue empty; EU must wait for BIU to fetch some bytes
- Address and Data Multiplexing: At different times the signals on these lines are interpreted as data or addresses

- 4 groups of registers
  1. 4 Data registers
  2. 4 Index registers
  3. 4 Segment registers
  4. Instruction Pointer and Flags Register

$\boxed{\text{The registers}}$

- DATA/COMPUTATIONAL Registers: AX, BX, CX, DX
- These 4 16-bit registers are for general purpose calculations and storage
- ML programs read numbers into these registers from memory, perform operations on them and write the results back into memory.
- The low-order 8-bits and high-order 8-bits can be used separately
- When used as 8-bit storage cells the registers are called

  AH, AL, BH, BL, CH, CL, DH, DL

  H → **H**igh-order byte, L → **L**ow-order byte, esp useful for working with 1byte ASCII...hint
- We can, for example, read a byte from memory into BL, without affecting the contents of BH.

AX : Accumulator, many operations have a shorter (faster) form that specifies AX implicitly

BX : Base Register, and can function as an **index** register [BX] $\rightarrow$ implies use contents of BX as an offset

CX : Count Register, does certain types of counting automatically

DX : Data Register, DX stores MSW of 32 bit ops

- Except for their minor areas of specialization, these 4 registers are generally interchangeable.

The Index group

- SI, DI, BP, SP
- Special register used for indexing or pointing
- That is, their contents are usually interpreted as addresses

SI : Source Index

DI : Destination Index

BP : Base Pointer

SP : Stack Pointer

- These registers can also be used for general purpose computation
- SI and DI are very useful in moving large sequences of bytes around memory

The Segment group: CS, DS, SS, ES

- Special registers used for Intel's addressing scheme
- Contents interpreted as the segment part of a 20 bit address.
- Whenever the processor reads/writes memory one of these registers provides the segment part of the memory address
- Each of these registers defines a segment (a 64K block in memory)

CS : Code Segment, instruction fetches are from the code segment

DS : Data Segment, data writes/reads are from the data segment

SS : Stack Segment, determines segment for stack operations

ES : Extra Segment, freelancer: usually a pointer to an extra data segment.

- These registers **CANNOT** be used for anything else

IP and Flags

Instruction Pointer (IP) also called the Program Counter **always** contains the address of the Next Instruction to be Executed.

- The processor contains logic circuits to adjust IP, so that it always points to the next instruction.
- The IP is always interpreted as an **Offset** to the Code Segment (CS)

$\boxed{\text{Flags Register (FL)}}$ (A flag is an especially noteworthy bit)

- An oddball register, contains a collection of unrelated boolean facts
- bit 0 is the Carry flag: signals unsigned overflow
- bit 2 is the Parity flag: later...
- bit 4 is the Auxiliary flag: later...maybe
- bit 6 is the Zero flag: If arithmetic op results in a 0, flag is set, else reset. Comparisons (subtract two quantities and set flag)
- bit 7 is Sign flag: Set if signed arithmetic operation results in a negative number else reset
- bit 8 is Trap flag: later
- bit 9 is Interrupt flag: turns interrupts off
- bit 10 is Direction flag:String ops direction
- bit 11 is Overflow flag: Set if Signed Overflow, else reset
- TVC shows state of each flag

Other Registers that TVC shows in "big" mode

- **IR: Instruction register:** 1 byte register that holds the current instruction code to be decoded to figure out what to do
- **EA: Effective address register:** Another 1 byte register also used in conjunction with IR
- **X, Y: temporary storage**
- **SR, OR, AD: Segment, Offset registers and Address register:** SR always contains the same value as one of the Segment registers, SR * 16 + OR = AD (20 bits) current 20 bit address in memory.
- **Z: data latch, processors window to the world.** no numbers enter or leave processor without going thru Z. ZH, ZL are the byte versions, bytes enter/leave thru ZL, words thru Z
- **AD, DB: Address and Data Bus:** During writes/reads of memory AD holds address and DB the data to be stored/read from memory at that address

Byte Order in Memory

- Multi-byte data stored with LSbyte at lowest memory address
- Bytes of increasing significance occupy successively higher addresses
- example: low mem → high mem

1. Byte: EB

E	B

2. Word: F21C

1	C	F	2

3. Double Word: 1234ABCD

C	D	A	B	3	4	1	2

4. An address: FFFE:1AC2

C	2	1	A	F	E	F	F