Quantifying Memory

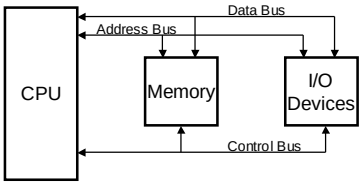
- Measured in the quantity of BInary digiT (BIT)

1 nybble	=	4 bits	Standard
1 byte	=	8 bits	Standard
1 word	=	16 bits	Machine Dependent (*8086*)
1 doubleword	=	32 bits	Machine Dependent (*8086*)
1 quadword	=	64 bits	Machine Dependent (*8086*)
1 paragraph	=	16 bytes	Machine Dependent (*8086*)
1 page	=	256 bytes	Machine Dependent (*8086*)
1 segment (max)	=	65,636 bytes	Machine Dependent (*8086*)

- Capacity Measures

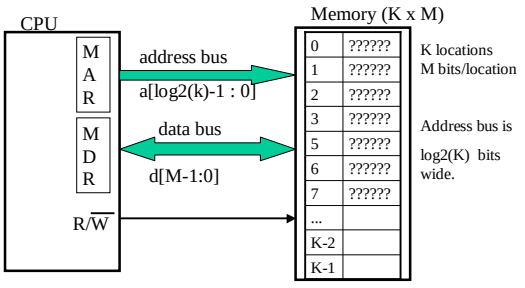
1 kilobyte	(kB)	=	2^{10} bytes
1 megabyte	(MB)	=	2^{20} bytes
1 gigabyte	(GB)	=	2^{30} bytes
1 terabyte	(TB)	=	2^{40} bytes

Modern Computer Model



- CPU - Central Processing Unit
  - arithmetic, logical, synchronization functions
- Memory - Stores Information
  - CPU instructions and Data
- I/O - Input/Output Devices - Peripherals
  - Interface to Outside World (humans, other machines)
- Bus - Set of Parallel wires
  - Transmit instructions/data between CPU/Memory/IO

Diagram of Memory



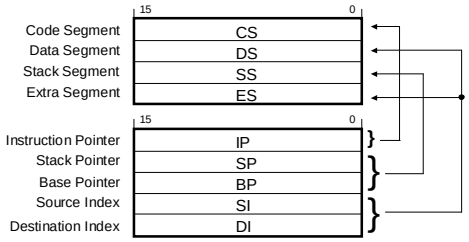
MAR: Memory Address Register, holds address of location being accessed.
MDR: Memory Data Register, holds incoming/outgoing data
R/$\overline{W}$: Read/$\overline{\text{Write}}$ control line, low when writing, high when Reading.

8086 Segmented Memory

- x86 Memory Partitioned into Segments
  - 8086: maximum size is 64K (16-bit index reg.)
  - 8086: can have 4 active segments (**CS, SS, DS, ES**)
  - 8086: 2-data; 1-code; 1-stack
  - x386: maximum size is 4GB (32-bit index reg.)
  - x386: can have 6 active segments (4-data; **FS, GS**)
- Why have segmented memory ????????

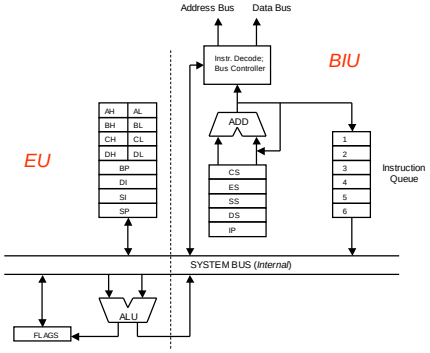
  Other microprocessors could only address 64K since they only had a single 16-bit MAR (or smaller). Segments allowed computers to be built that could use more than 64K memory (but not all at the same time).

8086/8088 Memory Access Registers

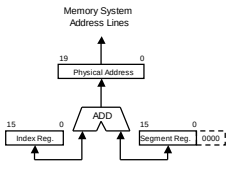


8086 Internal Architecture

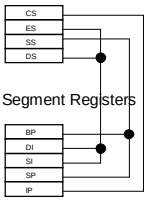
Processor Model



8086 Generating Physical Addresses



Portion of *BIU* Circuitry

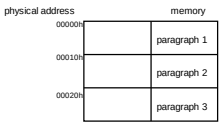


Dedicated Segment Registers

Dedicated Index Registers

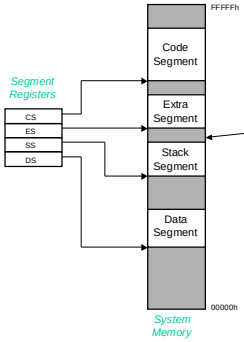
Segmented Addressing

- Each Segment must begin at Paragraph Boundary



- Each paragraph has phys. address that is multiple of 10h
- BIU is responsible for appending 0000 to Segment
  – only need 16-bit segment registers

Segmented Memory (x86 Style)

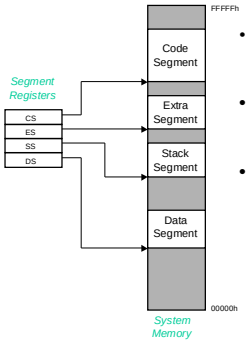


- Segment Registers:
  - Point to Base Address
- Index Registers:
  - *fragmentation*
  - Contain Offset Value
- Notation (*Segmented Address*):
  - `CS:IP`
  - `DS:SI`
  - `ES:DI`
  - `SS:BP`
  - `SS:SP`

Memory Storage Organization

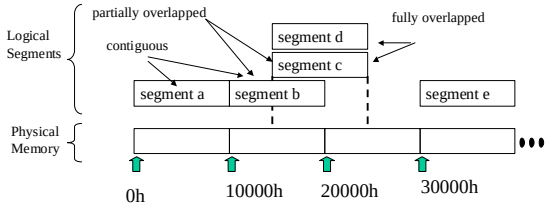
- Organized as *SEGMENTS*
  - Maximum segment size = 64KB
  - (Since 16 bit offsets: 2^{16} = 65,535 = 64KB)
- Maximum Memory Size:
  - 2^{20} = 1,048,576 = 1MB
- Newer Processors (386+) Can Utilize More Memory
  - Wider Address Registers 32 bits
  - 2^{32} = 4,294,967,296 = 4GB

Segmented Memory Example



- Logical, Segmented Address:
  0FE6:012Bh
- Offset, Index Address:
  012Bh
- Physical Address:
  0FE60h → 65120
  + 012Bh → 299
  0FF8Bh → 65149

Segment Locations in Physical Memory



Note that segments can overlap. This means that two different logical addresses can refer to the same physical address (aliasing).

Segmented Memory Aliasing

• Logical, Segmented Address 1:
DS:SI = 1234:4321

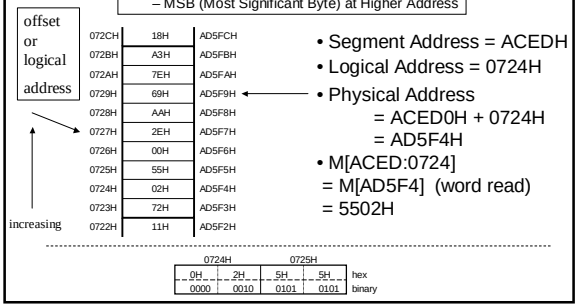
• Physical Address:
12340h → 74560
+ 4321h → 17185
16661h → 91745

• Logical, Segmented Address 2:
ES:DI = 1665:0011

• Physical Address:
16650h → 91728
+ 0011h → 00017
16661h → 91745

Memory Data Organization

• 1 Word = 16 bits
• Byte Addressable
• *Little Endian* Arrangement
– MSB (Most Significant Byte) at Higher Address



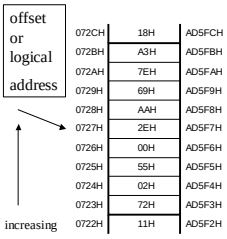
Register Transfer Language

• When describing memory/register operations, will use Register Transfer Language (RTL)

rega ← regb	transfer regb to rega (ax ← cx)
rega ← reg a *op* reg b	rega gets result of rega *op* regb
rega ← M[location N]	transfer contents of memory location N to reg a (mem read operation)
M[location N] ← rega	transfer reg a to memory location N (write operation)

Size of memory operation (byte, word, dword) depends on register size or instruction.

Memory Read Operations



072CH	18H	AD5FCH
072BH	A3H	AD5FBH
072AH	7EH	AD5FAH
0729H	69H	AD5F9H
0728H	AAH	AD5F8H
0727H	2EH	AD5F7H
0726H	00H	AD5F6H
0725H	55H	AD5F5H
0724H	02H	AD5F4H
0723H	72H	AD5F3H
0722H	11H	AD5F2H

Byte Read:
al ← M[0724h]
after: al = 02h

Word Read:
ax ← M[0724h]
after: ax = 5502h

Dword Read
eax ← M[0724h]
after: eax = 2E005502h

Memory Write Operations

Assume EAX = FAC4237B h

Byte Write: M[0724h] ← al

before:

072CH	18H	AD5FCH
072BH	A3H	AD5FBH
072AH	7EH	AD5FAH
0729H	69H	AD5F9H
0728H	AAH	AD5F8H
0727H	2EH	AD5F7H
0726H	00H	AD5F6H
0725H	55H	AD5F5H
0724H	02H	AD5F4H
0723H	72H	AD5F3H
0722H	11H	AD5F2H

after:

072CH	18H	AD5FCH
072BH	A3H	AD5FBH
072AH	7EH	AD5FAH
0729H	69H	AD5F9H
0728H	AAH	AD5F8H
0727H	2EH	AD5F7H
0726H	00H	AD5F6H
0725H	55H	AD5F5H
0724H	7BH	AD5F4H
0723H	72H	AD5F3H
0722H	11H	AD5F2H

Memory Write Operations (cont.)

Assume EAX = FAC4237B h

Word Write: M[0724h] ← ax

before:

072CH	18H	AD5FCH
072BH	A3H	AD5FBH
072AH	7EH	AD5FAH
0729H	69H	AD5F9H
0728H	AAH	AD5F8H
0727H	2EH	AD5F7H
0726H	00H	AD5F6H
0725H	55H	AD5F5H
0724H	02H	AD5F4H
0723H	72H	AD5F3H
0722H	11H	AD5F2H

after:

072CH	18H	AD5FCH
072BH	A3H	AD5FBH
072AH	7EH	AD5FAH
0729H	69H	AD5F9H
0728H	AAH	AD5F8H
0727H	2EH	AD5F7H
0726H	00H	AD5F6H
0725H	23H	AD5F5H
0724H	7BH	AD5F4H
0723H	72H	AD5F3H
0722H	11H	AD5F2H

Memory Write Operations (cont.)

Assume EAX = FAC4237B h

DWord Write: M[0724h] ← eax

before:

072CH	18H	AD5FCH
072BH	A3H	AD5FBH
072AH	7EH	AD5FAH
0729H	69H	AD5F9H
0728H	AAH	AD5F8H
0727H	2EH	AD5F7H
0726H	00H	AD5F6H
0725H	55H	AD5F5H
0724H	02H	AD5F4H
0723H	72H	AD5F3H
0722H	11H	AD5F2H

after:

072CH	18H	AD5FCH
072BH	A3H	AD5FBH
072AH	7EH	AD5FAH
0729H	69H	AD5F9H
0728H	AAH	AD5F8H
0727H	FAH	AD5F7H
0726H	C4H	AD5F6H
0725H	23H	AD5F5H
0724H	7BH	AD5F4H
0723H	72H	AD5F3H
0722H	11H	AD5F2H

Default Segment/Index Pairs

Type of Memory Reference	Default Segment Base	Alternate Segment Base	Offset
Instruction Fetch	CS	None	IP
Stack Operation	SS	None	SP
Variable (except following)	DS	CS, ES, SS	Effective Address
- String Source	DS	CS, ES, SS	SI
- String Destination	ES	None	DI
- BP used as Base Register	SS	CS, DS, ES	Effective Address
- BX Used as Base Register	DS	CS, ES, SS	Effective Address

80386 Operating Modes

Real Protected Virtual 8086

- Real
  - Initialized to this Mode at Boot Time
  - Looks Identical to 8086 Except FS and GS
- Protected
  - Multi-task Support
  - Segments can be up to 4GB in Size
  - Task Segments Defined by Descriptor Tables
  - Memory Protection and Task Privilege
  - Paging Support
- VM86
  - Allows for Multiple 8086 (Real) Tasks
  - Non-interacting
  - VM86 and Protected Simultaneously