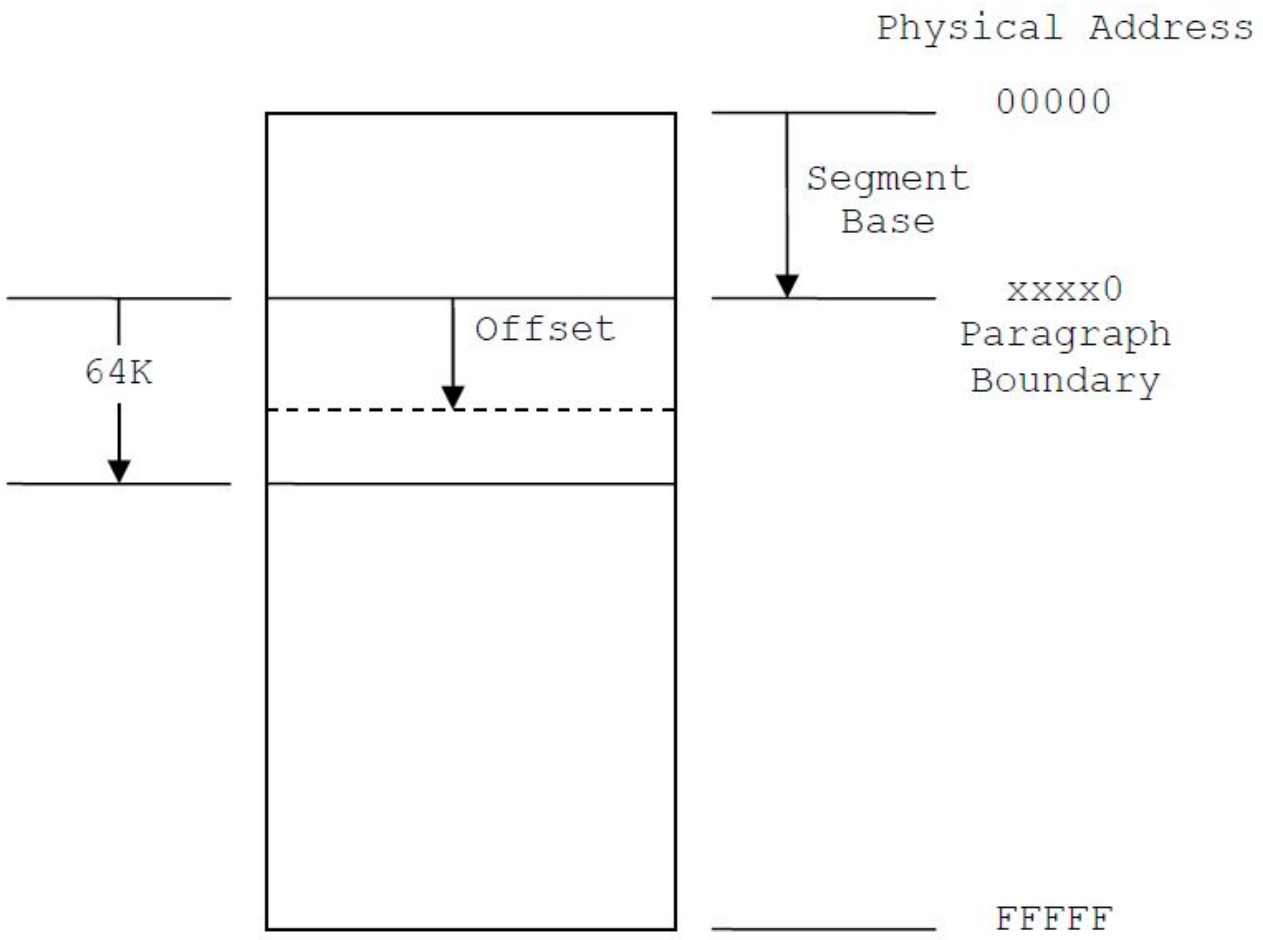
Data Transfer and Addressing

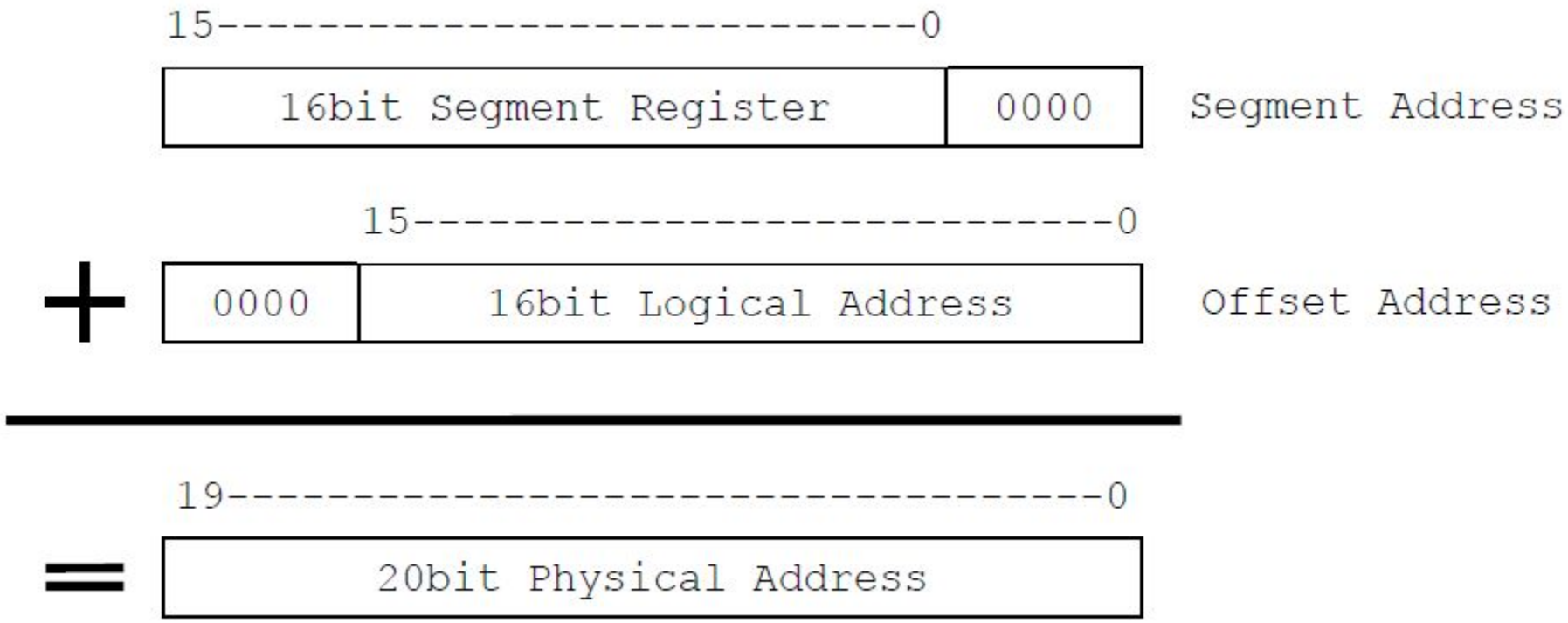
Contents

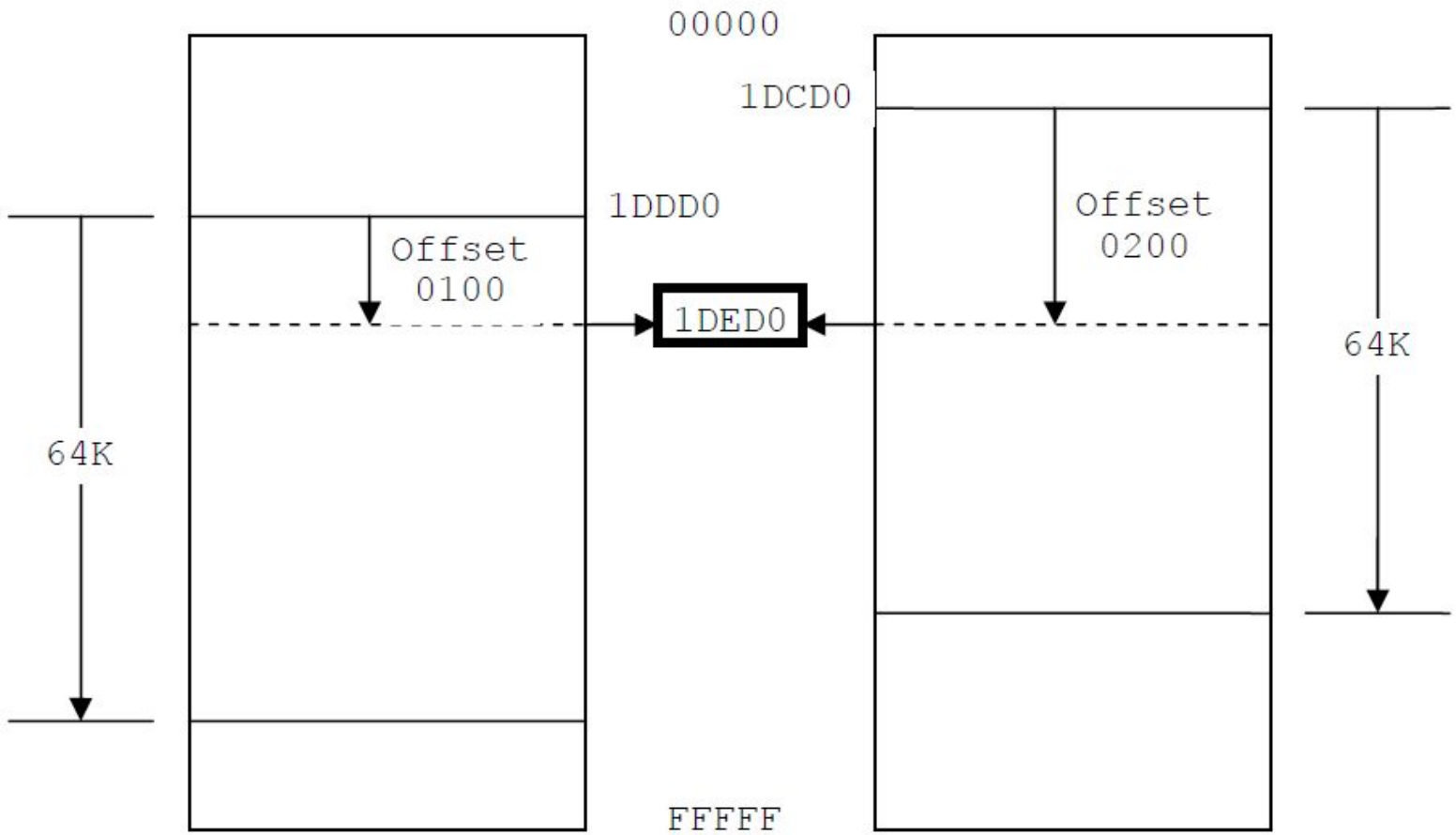
- Segmented Memory Model
- Data Transfer
- Data Declaration
- Addressing Modes
  - Direct
  - Indirect

Segmented Memory Model



Logical to Physical Address




00000
1DCD0
1DDD0
Offset
0100
1DED0
Offset
0200
64K
64K
FFFFF

Data Transfer

- The MOV instruction copies data from a source operand to a destination operand.
- Known as a data transfer instruction, it is used in virtually every program.
- Its basic format shows that the first operand is the destination and the second operand is the source:
  - ***MOV destination,source***
- It is equivalent to ***destination= source;*** in any high level language like in java or C++

Rules for MOV instruction

- MOV is very flexible in its use of operands, as long as the following rules are observed:
  - Both operands must be the same size.
  - Both operands cannot be memory operands.
  - The instruction pointer register (IP) cannot be a destination operand.

MOV instruction formats

- Here is a list of the standard MOV instruction formats:

MOV register, register
MOV register, immediate operand
MOV memory, register
MOV register, memory
MOV memory, immediate operand

Direct Addressing

Example 2.1

```
001         ; a program to add three numbers using memory variables
002         [org 0x0100]
003                     mov   ax, [num1]          ; load first number in ax
004                     mov   bx, [num2]          ; load second number in bx
005                     add   ax, bx              ; accumulate sum in ax
006                     mov   bx, [num3]          ; load third number in bx
007                     add   ax, bx              ; accumulate sum in ax
008                     mov   [num4], ax          ; store sum in num4
009
010                     mov   ax, 0x4c00          ; terminate program
011                     int   0x21
012
013         num1:       dw    5
014         num2:       dw    10
015         num3:       dw    15
016         num4:       dw    0
```

002	Originate our program at 0100. The first executable instruction should be placed at this offset.
003	The source operand is changed from constant 5 to [num1]. The bracket is signaling that the operand is placed in memory at address num1. The value 5 will be loaded in ax even though we did not specified it in our program code, rather the value will be picked from memory. The instruction should be read as "read the contents of memory location num1 in the ax register." The label num1 is a symbol for us but an address for the processor while the conversion is done by the assembler.
013	The label num1 is defined as a word and the assembler is requested to place 5 in that memory location. The colon signals that num1 is a label and not an instruction.

Listing file

- The first instruction of our program has changed from B80500 to A11700.
- The opcode B8 is used to move constants into AX, while the opcode A1 is used when moving data into AX from memory.
- The immediate operand to our new instruction is 1700 or as a word 0017 (23 decimal) and from the bottom of the listing file we can observe that this is the offset of num1.
- The assembler has calculated the offset of num1 and used it to replace references to num1 in the whole program. Also the value 0500 can be seen at offset 0017 in the file.
- We can say contents of memory location 0017 are 0005 as a word. Similarly num2, num3, and num4 are placed at 0019, 001B, and 001D addresses.
- When the program is loaded in the debugger, it is loaded at offset 0100, which displaces all memory accesses in our program. The instruction A11700 is changed to A11701 meaning that our variable is now placed at 0117 offset.
- The instruction is shown as mov ax, [0117]. Also the data window can be used to verify that offset 0117 contains the number 0005.

```
 1
 2                                  [org 0x0100]
 3 00000000 A1[1700]                            mov  ax, [num1]
 4 00000003 8B1E[1900]                          mov  bx, [num2]
 5 00000007 01D8                                add  ax, bx
 6 00000009 8B1E[1B00]                          mov  bx, [num3]
 7 0000000D 01D8                                add  ax, bx
 8 0000000F A3[1D00]                            mov  [num4], ax
 9
10 00000012 B8004C                              mov  ax, 0x4c00
11 00000015 CD21                                int  0x21
12
13 00000017 0500                   num1:        dw   5
14 00000019 0A00                   num2:        dw   10
15 0000001B 0F00                   num3:        dw   15
16 0000001D 0000                   num4:        dw   0
```

Example 2.2

```
001         ; a program to add three numbers accessed using a single label
002         [org 0x0100]
003                         mov  ax, [num1]          ; load first number in ax
004                         mov  bx, [num1+2]        ; load second number in bx
005                         add  ax, bx              ; accumulate sum in ax
006                         mov  bx, [num1+4]        ; load third number in bx
007                         add  ax, bx              ; accumulate sum in ax
008                         mov  [num1+6], ax        ; store sum at num1+6
009
010                         mov  ax, 0x4c00          ; terminate program
011                         int  0x21
012
013         num1:           dw   5
014                         dw   10
015                         dw   15
016                         dw   0
```

004 The second number is read from num1+2. Similarly the third number is read from num1+4 and the result is accessed at num1+6.

013-016 The labels num2, num3, and num4 are removed and the data there will be accessed with reference to num1.

Example 2.3

```
001     ; a program to add three numbers accessed using a single label
002     [org 0x0100]
003                     mov   ax, [num1]          ; load first number in ax
004                     mov   bx, [num1+2]        ; load second number in bx
005                     add   ax, bx              ; accumulate sum in ax
006                     mov   bx, [num1+4]        ; load third number in bx
007                     add   ax, bx              ; accumulate sum in ax
008                     mov   [num1+6], ax        ; store sum at num1+6
009
010                     mov   ax, 0x4c00          ; terminate program
011                     int   0x21
012
013     num1:           dw    5, 10, 15, 0
```

013 As we do not need to place labels on individual variables we can save space and declare all data on a single line separated by commas. This declaration will declare four words in consecutive memory locations while the address of first one is num1.

Example 2.4

```
01      ; a program to add three numbers directly in memory
02      [org 0x0100]
03                      mov   ax, [num1]          ; load first number in ax
04                      mov   [num1+6], ax        ; store first number in result
05                      mov   ax, [num1+2]        ; load second number in ax
06                      add   [num1+6], ax        ; add second number to result
07                      mov   ax, [num1+4]        ; load third number in ax
08                      add   [num1+6], ax        ; add third number to result
09
10                      mov   ax, 0x4c00          ; terminate program
11                      int   0x21
12
13      num1:           dw    5, 10, 15, 0
```

```
 1
 2                                          [org 0x0100]
 3 00000000 A1[1900]                                    mov  ax, [num1]
 4 00000003 A3[1F00]                                    mov  [num1+6], ax
 5 00000006 A1[1B00]                                    mov  ax, [num1+2]
 6 00000009 0106[1F00]                                  add  [num1+6], ax
 7 0000000D A1[1D00]                                    mov  ax, [num1+4]
 8 00000010 0106[1F00]                                  add  [num1+6], ax
 9
10 00000014 B8004C                                      mov  ax, 0x4c00
11 00000017 CD21                                        int  0x21
12
13 00000019 05000A000F000000          num1:             dw   5, 10, 15, 0
```

Size Mismatch Errors

Example 2.5

```
001     ; a program to add three numbers using byte variables
002     [org 0x0100]
003                     mov   al, [num1]          ; load first number in al
004                     mov   bl, [num1+1]        ; load second number in bl
005                     add   al, bl              ; accumulate sum in al
006                     mov   bl, [num1+2]        ; load third number in bl
007                     add   al, bl              ; accumulate sum in al
008                     mov   [num1+3], al        ; store sum at num1+3
009
010                     mov   ax, 0x4c00          ; terminate program
011                     int   0x21
012
013     num1:           db    5, 10, 15, 0
```

003	The number is read in AL register which is a byte register since the memory location read is also of byte size.
005	The second number is now placed at num1+1 instead of num1+2 because of byte offsets.
013	To declare data db is used instead of dw so that each data declared occupies one byte only.

Register Indirect Addressing

Example 2.6

```
001     ; a program to add three numbers using indirect addressing
002     [org 0x100]
003                     mov  bx, num1           ; point bx to first number
004                     mov  ax, [bx]           ; load first number in ax
005                     add  bx, 2              ; advance bx to second number
006                     add  ax, [bx]           ; add second number to ax
007                     add  bx, 2              ; advance bx to third number
008                     add  ax, [bx]           ; add third number to ax
009                     add  bx, 2              ; advance bx to result
010                     mov  [bx], ax           ; store sum at num1+6
011
012                     mov  ax, 0x4c00         ; terminate program
013                     int  0x21
014
015     num1:           dw   5, 10, 15, 0
```

003	Observe that no square brackets around num1 are used this time. The address is loaded in bx and not the contents. Value of num1 is 0005 and the address is 0117. So BX will now contain 0117.
004	Brackets are now used around BX. In iapx88 architecture brackets can be used around BX, BP, SI, and DI only. In iapx386 more registers are allowed. The instruction will be read as "move into ax the contents of the memory location whose address is in bx." Now since bx contains the address of num1 the contents of num1 are transferred to the ax register. Without square brackets the meaning of the instruction would have been totally different.
005	This instruction is changing the address. Since we have words not bytes, we add two to bx so that it points to the next word in memory. BX now contains 0119 the address of the second word in memory. This was the mechanism to change addresses that we needed.

Example 2.7

```
001     ; a program to add ten numbers
002     [org 0x0100]
003                     mov  bx, num1           ; point bx to first number
004                     mov  cx, 10             ; load count of numbers in cx
005                     mov  ax, 0              ; initialize sum to zero
006
007     l1:             add  ax, [bx]           ; add number to ax
008                     add  bx, 2              ; advance bx to next number
009                     sub  cx, 1              ; numbers to be added reduced
010                     jnz  l1                 ; if numbers remain add next
011
012                     mov  [total], ax        ; write back sum in memory
013
014                     mov  ax, 0x4c00         ; terminate program
015                     int  0x21
016
017     num1:           dw   10, 20, 30, 40, 50, 10, 20, 30, 40, 50
018     total:          dw   0
```

Register Indirect + Offset Addressing

Example 2.8

```
001     ; a program to add ten numbers using register + offset addressing
002     [org 0x0100]
003                     mov  bx, 0                  ; initialize array index to zero
004                     mov  cx, 10                 ; load count of numbers in cx
005                     mov  ax, 0                  ; initialize sum to zero
006
007     l1:             add  ax, [num1+bx]          ; add number to ax
008                     add  bx, 2                  ; advance bx to next index
009                     sub  cx, 1                  ; numbers to be added reduced
010                     jnz  l1                     ; if numbers remain add next
011
012                     mov  [total], ax            ; write back sum in memory
013
014                     mov  ax, 0x4c00             ; terminate program
015                     int  0x21
016
017     num1:           dw   10, 20, 30, 40, 50, 10, 20, 30, 40, 50
018     total:          dw   0
```

003	This time BX is initialized to zero instead of array base
007	The format of memory access has changed. The array base is added to BX containing array index at the time of memory access.
008	As the array is of words, BX jumps in steps of two, i.e. 0, 2, 4. Higher level languages do appropriate incrementing themselves and we always use sequential array indexes. However in assembly language we always calculate in bytes and therefore we need to take care of the size of one array element which in this case is two.

ADDRESSING MODES SUMMARY

- Direct
  - Direct + offset
- Indirect
  - Based Register Indirect (e.g. [BX] or [BP])
  - Index Register Indirect (e.g. [DI] or [SI])
  - Based Register Indirect + offset (e.g. [BX+100] or [BP+200])
  - Index Register Indirect +offset (e.g. [SI+100] or [DI+200])
  - Base + index (e.g. [BX+SI])
  - Base + index + offset (e.g. [BX+SI+300])
- Things that are not allowed:
  - Base register + base register (e.g [BX+BP])
  - Index register + index register (e.g. [SI+DI])
  - Base – index (e.g. [BX-SI])
  - Part of register cannot be used to access memory address (e.g [BH] or [BL])

Important thing to remember

- Programmer has a full control of memory.
- If you write any (valid) 16 bit address in square brackets you will be able to access it , either its in form of label/registers +/- offset or simple constant number.
- It is up to you to access it carefully without creating logical errors

Reading

- BH 2.1, 2.2, 2.3, 2.4, 2.5