Assembly Language

Data Alignment

GAS

```
.balign alignment      # .balign 2
```

MASM

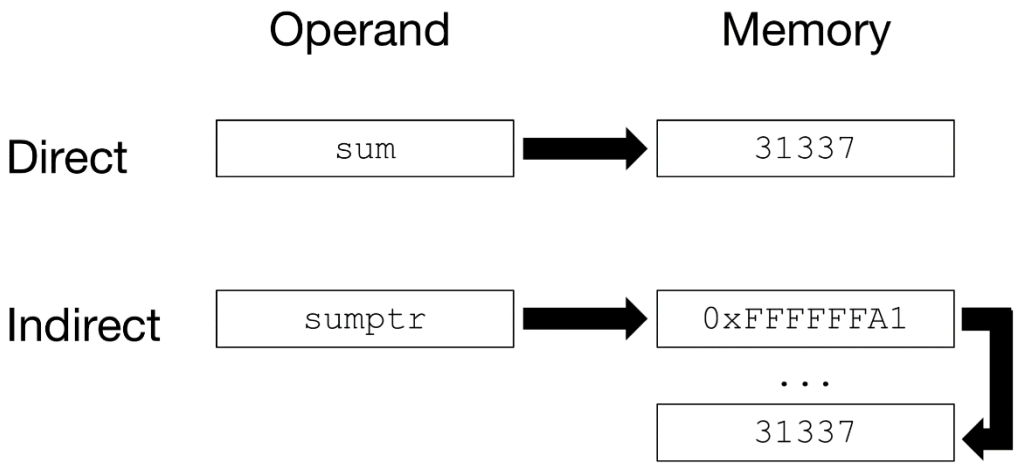
```
ALIGN alignment        ; ALIGN 2
```

NASM

```
SECTION .data
ALIGN alignment        ; ALIGN 2
```

```
SECTION .bss
ALIGNB alignment   ; ALIGNB 2
```

Data Addressing (Figure 4.1)



Storing an Operand's Address

GAS

```
MOVS $M, M/%R        # MOVL $sum, %esi
```

MASM

```
MOV M/R, OFFSET M        ; MOV esi, OFFSET sum
```

NASM

```
MOV SIZE [M]/R, M        ; MOV DWORD [sumaddr], sum
```

GAS

```
LEAS $M, %R          # LEAL sum, %esi
```

MASM

```
LEA R, M             ; LEA esi, sum
```

NASM

```
LEA R, [M]           ; LEA eax, [sum]
```

Arrays (Figure 4.2)

arrayA	
Address	**Value**
0x00000000	2
0x00000001	4
0x00000002	6
0x00000003	8

arrayB	
Address	**Value**
0x00000000	FFFFF
0x00000004	FFFFE
0x00000008	FFFFD
0x0000000C	FFFFC

Example 4.10 Arrays in MASM

```
arrayA BYTE 2, 4, 6, 8
arrayB DWORD 0FFFFFh, 0FFFFEh, 0FFFFDh, 0FFFFCh
```

Array Size

GAS

```
arrayA: .long 2, 4, 6, 8
.equ len, (. - arrayA)          # len = 16
```

MASM

```
arrayA DWORD 2, 4, 6, 8
len EQU ($ - arrayA)            ; len = 16
```

NASM

```
arrayA: DD 2, 4, 6, 8
len: EQU ($ - arrayA)           ; len = 16
```

Array Usage with Byte Offsets

GAS

```
LEAS M, %R                      # Store item 1 from arrayA in ebx
MOVS CONSTANT(%R), M/%R         # LEAL arrayA, %eax
                                # MOVL 1(%eax), %ebx
                                # %ebx = 4

                                # Store 10 into element 3 of arrayB
LEAS M, %R                      # LEAL arrayB, %eax
MOVS $L/%R, CONSTANT %R         # MOVL $10, 8(%eax)
                                # arrayB = 0xFFFFF,0xFFFFE,0xA,0xFFFFC
```

MASM

```
                                ; Store item 1 from arrayA in eax
MOV R, [M+CONSTANT]             ; MOV eax, [arrayA + 1]
                                ; eax = 4

                                ; Store 10 into item 3 of arrayB
MOV [M+CONSTANT], R             ; MOV [arrayB + 8], 10
                                ; arrayB = 0xFFFFF,0xFFFFE,0xA,0xFFFFC
```

NASM

```
                                ; Store item 1 from arrayA in eax
MOV R, [M+CONSTANT]             ; MOV eax, [arrayA + 1]
                                ; eax = 4

                                ; Store 10 into item 3 of arrayB
MOV SIZE [M+CONSTANT], L/R      ; MOV DWORD [arrayB + 8], 10
                                ; arrayB = 0xFFFFF,0xFFFFE,0xA,0xFFFFC
```

Array Usage with High-Level Indices

GAS

```
LEAS M, %R                    # LEAL arrayB, %eax
MOVS L, %R                    # MOVL $3, %edx
MOVS (%R, %R, L), %R          # MOVL (%eax, %edx, 4), %ebx
```

MASM/NASM

```
MOV R, L                      ; MOV edx, 3
MOV R, [M+R*L]                ; MOV eax, [arrayB + edx * 4]
```

Array Usage (MASM-specific)

MASM

```
MOV eax, TYPE arrayA            ; eax = 1
MOV ebx, LENGTHOF arrayA        ; ebx = 4
MOV ecx, SIZEOF arrayA          ; ecx = 4

MOV eax, TYPE arrayB            ; eax = 4
MOV ebx, LENGTHOF arrayB        ; ebx = 4
MOV ecx, SIZEOF arrayB          ; ecx = 16
```

Changing Data Sizes (Large -> Small)

GAS

```
MOVS M, %R        # MOVW val, %ax
```

MASM

```
MOV R, SIZE PTR M        ; MOV ax, WORD PTR val
```

NASM

```
MOV R, SIZE [M]          ; MOV ax, WORD [val]
```

Changing Data Sizes (Small-> Large)

GAS

```
MOVZSS M/%R, %R        # MOVZWL sum, %eax
MOVSSS M/%R, %R        # MOVSWL sum, %eax
```

MASM

```
MOVZX R, M/R           ; MOVZX eax, sum
MOVSX R, M/R           ; MOVSX eax, sum
```

NASM

```
MOVZX R, SIZE [M]/R     ; MOVZX eax, WORD [sum]
MOVSX R, SIZE [M]/R     ; MOVSX eax, WORD [sum]
```