CS2422 Assembly Language & System Programming

October 3, 2006

Today’s Topics

• Section 4.3: Data-Related Operators
• Section 4.4: Indirect Addressing
  – Now you can handle array!
• Section 4.5: Jump and Loop
Data Related Operators

• Who are they?
  – OFFSET, PTR, TYPE, LENGTHOF, SIZEOF
• They are only understood by the assembler.
• They are not instructions! For example:
  MOV EDI, OFFSET Var1

Operand Sizes

• Operands may have the size of 1 byte, 2 bytes, or 4 bytes.
• Most of the time, we can tell the size from the register names or the variable definition.
  For examples:
  Var1 BYTE “Hello”
  MOV ECX, 13
  MOV AL, Var1
PTR

- But sometimes we cannot tell the size of operand, especially if indirect addressing (or pointer) is used.
- Or we may simply want to override the default.

_Some examples in next slide..._

```assembly
myDouble    DWORD 12345678h
MOV AL, myDouble ; error
MOV AL, BYTE PTR myDouble
MOV AX, WORD PTR myDouble
MOV AX, WORD PTR [myDouble+2]
MOV EAX, myDouble
```
What Else?

• TYPE returns the size (in bytes) of each element.
• LENGTHOF returns the number of elements.
• SIZEOF returns the size of the variable (the whole array).
  \[ \text{SIZEOF} = \text{LENGTHOF} \times \text{TYPE} \]

I lied when I said there was no array data type in assembly...

.data
byte1 BYTE 10, 20, 30
array1  WORD 30 DUP(?)

• Exercise: What is TYPE byte1? TYPE array1?
• LENGTHOF array1 is 30, SIZEOF array1 is 60.
Direct-Offset Addressing

• During last lecture, we discussed Direct-Offset operands:

• Problem: the offset is fixed.
  – Can’t handle array index, like \( A[i] \)

Indirect Addressing

• The solution? The memory address must be a variable too! \( \Rightarrow \) Store it in a register!

• Compare these:
  – MOV AL, [10000h]
  – MOV AL, [Var1+1]
  – MOV AL, [ESI]  \( \Leftarrow \) indirect addressing
OFFSET Operator

• But...How do we get the address?
  – For example: “MOV ESI, Var1” moves the value of var1, not its address.
• Answer: Use the OFFSET operator to obtain the address.
  – MOV ESI, OFFSET Var1

Array – An Example

.data
arrayB BYTE 10h, 20h, 30h
.code
    mov ESI,OFFSET arrayB
    mov AL, [ESI] ; first byte
    INC ESI
    add AL, [ESI] ; second byte
    INC ESI
    add AL, [ESI] ; third byte
Array Index

• So, can you modify the code in last slide to implement array index like `arrayB[i]`? (Assume `i` is stored in a register, e.g., `ECX`.)

*Wait! There is an easier way...*

Indexed Operands

A Few Examples:
• `[arrayB+ESI]` or simply `arrayB[ESI]`
• `[ESI+2], [ESI+4],…etc.`
Pointers

- Now we know that we can store an address in a register.
- Can we store it in a variable (in memory) too?

What Have We Learned So Far?

A bird’s-eye view:
- It’s a very different world from high-level languages.
- It’s important to access the data (in memory) precisely at our will.
- Flags to control the execution flow.
Where Do We Go from Here?

- Conditional Branches (similar to if…then…)
- And most importantly…

Get a feeling of how the low-level actions in assembly level become the fancy Windows (or Linux) operating system and applications.

Implementation of Loops

- JMP instruction: Unconditional Branch.
- LOOP instruction:
  - Step 1: Set ECX to n for a loop of n iterations.
  - Step 2: Use LOOP instruction at the end of loop.
  - Hidden action: DEC ECX
Example 1: Summation

• For I := 10 downto 1 \{Sum := Sum+I\}

```assembly
MOV ECX, 10
MOV EAX, 0
L1:    ADD EAX, ECX
      LOOP L1
```

Example 2: Summation

• For I := 1 to 10 \{Sum := Sum+I\}

```assembly
MOV ECX, 10
MOV EAX, 0
MOV EDX, 1
L1:    ADD EAX, EDX
       INC EDX
       LOOP L1
```
Your turn . . .

What will be the final value of AX?

10

How many times will the loop execute?

4,294,967,296

(=2^{32})

Example 3: Array Traversal

• Exercise: what is computed and stored at EAX?

MOV ECX, 10
MOV EAX, 0
MOV EDI, OFFSET var1
L1: ADD EAX, [EDI]
INC EDI
LOOP L1
Copying a String

The following code copies a string from source to target.

```
.data
source  BYTE  "This is the source string",0
target  BYTE  SIZEOF source DUP(0),0

.code
    mov  esi,0 ; index register
    mov  ecx,SIZEOF source ; loop counter
L1:
    mov  al,source[esi] ; get char from source
    mov  target[esi],al ; store it in the target
    inc  esi ; move to next character
    loop L1 ; repeat for entire string
```

*good use of SIZEOF*