COMP1001

Computer Systems

Dr. Vasilios Kelefouras

Email: v.kelefouras@plymouth.ac.uk
Website:
https://www.plymouth.ac.uk/staff/vasilios-kelefouras

Outline

- x86-64 Assembly
  - Why use assembly?
  - Basic concepts
  - Different ways of using assembly

Main reasons for using assembly nowadays

- Security
  - Make viruses
  - Reverse engineering to identify software flaws
- Making compilers, hardware drivers, processors
- Optimize Software programs (develop more efficient software) in terms of
  - execution time
  - energy consumption

Main reasons for NOT writing assembly nowadays

- Development time
- Reliability and security
- Debugging
- Maintainability
- Portability

X86, X64 and IA-32

- What is **x86** and what **x64**?
  - **x86** is an Intel CPU architecture that originated with the 16-bit 8086 processor in 1978.
  - Today, the term "**x86**" is used generally to refer to any 32-bit processor compatible with the **x86** instruction set
  - **IA-32** (short for "Intel Architecture, 32-bit“), sometimes also called i386 is the 32-bit version of the **x86** instruction set architecture
  - **x86-64** or x64 is the general name of a series of 64-bit processors and their associated instruction set architecture. These processors are compatible with **x86**.
- What 32bit mean?
  - 32bit Data/address bus, registers, …

Introduction to x86 Assembly Programming

- There are many different assemblers : MASM, NASM, GAS, AS86, TASM, A86, Terse, etc. All use radically different assembly languages.
  - GNU Assembler (GAS)
    - AT&T syntax for writing the assembly language
  - Microsoft Macro Assembler (MASM)
  - Netwide Assembler (NASM)

A warming up example


```
.data
 ;This is a comment. this is the region where we define our variables. There are no variables in this example
.code
 ;This is a comment. Here we write our program
main function
    mov eax, 25 ; move the literal value 25 to eax register
    mov ebx, 30
    add eax, ebx ; eax=eax+ebx
INVOKE ExitProcess, 0 ; call exit function
Main ENDP ; exit main procedure
END main ; stop assembling
```

Let's open VS and debug this

This is our first assembly program

Reserved Words

- Predefined purpose, e.g. mov is a reserved word and an instruction
- These **cannot** be used in any other way, e.g. for variable names
- **Case-insensitive:** Mov ≡ mov ≡ MOV

MASM

```
.386
.MODEL FLAT, stdcall
.STACK 4096
ExitProcess PROTO,
dwExitCode:DWORD

.data
sum DWORD 0

.code
_main PROC
mov eax, 25
mov ebx, 50
add eax, ebx
mov sum, eax

INVOKE ExitProcess, 0
_main ENDP
END
```

Directives

- **Assembler specific commands: direct the assembler to do something**
- Examples
  - *answer DWORD 42 :*ask the assembler to reserve 32-bits of memory and write the literal value 42
  - ***.386*** Enables 80386 processor instructions
  - ***.model*** Sets the memory model. FLAT for 32-bit instructions, and stdcall for assembly instructions

MASM

```
.386
.MODEL FLAT, stdcall
.STACK 4096
ExitProcess PROTO,
dwExitCode:DWORD

.data
sum DWORD 0

.code
_main PROC
mov eax, 25
mov ebx, 50
add eax, ebx
mov sum, eax

INVOKE ExitProcess, 0
_main ENDP
END
```

Program sections (or segments)

- Special sections pre-defined by the assembler
- Common segments:
  - ***.data*** uninitialised and initialised variables
  - ***.code*** assembly instructions

MASM

```
.386
.MODEL FLAT, stdcall
.STACK 4096
ExitProcess PROTO,
dwExitCode:DWORD

.data
sum DWORD 0

.code
_main PROC
mov eax, 25
mov ebx, 50
add eax, ebx
mov sum, eax

INVOKE ExitProcess, 0
_main ENDP
END
```

Labels and Comments

- Labels allow us to partition code for programmatic (e.g. jumping and looping), or design purposes (e.g. clarity). Used in .code section.

```
userLoop:
            inc counter

otherLoop: inc counter2
```

- Comments are integral parts of coding: explain why and how (as opposed to what). Usually starts with ‘;’.

```
mov eax, counter ; Moves the counter value into eax
```

```
COMMENT !
     This is the section of the code where
     employee salaries are calculated. Note
     how the exclamation point is not in the
     text of the comment.
!
```

Instructions

- **Executable statements in a program**
- **Two basic parts:** *mnemonic and [operands]*
- *Mnemonic* is the instruction name as defined in the architecture's instruction set
- Some do not require operands, some one or more
- Common code examples:
  - ***inc eax :*** increments eax by one
  - ***mov eax, 5 :*** moves literal value 5 to eax register

Intel's x86 instruction set manuals comprise over 2900 pages – it is large and complex

MASM

```
.386
.MODEL FLAT, stdcall
.STACK 4096
ExitProcess PROTO,
dwExitCode:DWORD

.data
sum DWORD 0

.code
_main PROC
mov eax, 25
mov ebx, 50
add eax, ebx
mov sum, eax

INVOKE ExitProcess, 0
_main ENDP
END
```

Label:	Mnemonic	Operand(s)	;Comment

Literals


```
00011111b   ; b is the radix character for binary
31          ; decimal values do not need radix characters
31d         ; but you can specify d for decimal
1Fh         ; h is the radix character for hexadecimal
37o         ; o is the radix character for octal
```

Radix	Base
b	Binary (base-2)
d	Decimal (base-10)
h	Hexadecimal (base-16)
q, o	Octal (base-8)

```
0FFFF0342h  ; the actual value is FFFF0342 in hexadecimal
```

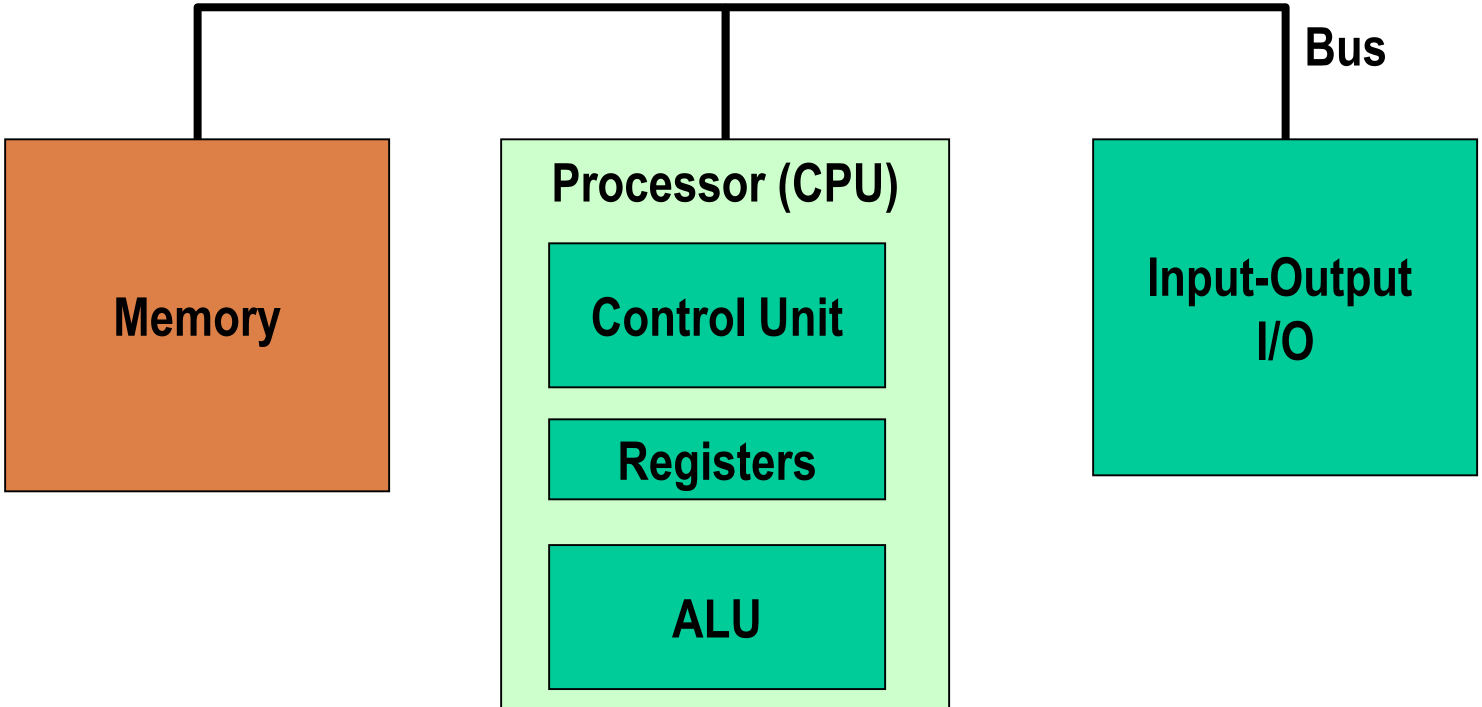
```
"I don't understand contractions."          ; strings that have one
'"Good job," said the father to his son.'   ; type of quotes on the
                                            ; outside and a different
                                            ; type on the inside
```

A more complicated example


```
.data
 sum DWORD 0 ; sum is a 32-bit variable stored somewhere in memory
 input DWORD 25
.code
 ;This is a comment. Here we write our program
main function
    mov eax, input ; load input variable from memory and store it to eax
    mov ebx, 30
    add eax, ebx ; eax=eax+ebx
    mov sum, eax ; mov eax to sum
INVOKE ExitProcess, 0 ; call exit function
Main ENDP ; exit main procedure
END main ; stop assembling
```

What is a variable and what is a register?

- **Registers** are very small (e.g. 8 bytes) and very fast dedicated memories. Their number is limited and each register has a different name, e.g., eax, ebx
- **Variables** are high level language constructs (not always used in assembly). Each variable is stored somewhere in memory



Data Types

- BYTE – 8bit unsigned integer
- SBYTE – 8bit signed integer
- WORD - 16bit unsigned integer
- SWORD - 16bit signed integer
- DWORD - 32bit unsigned integer
- SDWORD - 32bit signed integer
- QWORD – 64bit unsigned integer
- REAL4 – single precision floating point numbers (32bit)
- REAL8 - double precision floating point numbers (64bit)

Let's use Visual Studio

- Open the 'assembly programming I' pdf file and follow the steps

Further Reading

- Chapter 1 and Chapter 2 in ‘Modern X86 Assembly Language Programming’ , available at https://www.pdfdrive.com/download.pdf?id=185772000&h=3dfb070c1742f50b500f07a63a30c86a&u=cache&ext=pdf