ASC CURS 11

Modular architectures

* Modular programming
  + It’s not to be confused with what’s “modular programming” (the idea of splitting my source code into many files. Keeping them like that for the sake of reuse. This idea came from the need to reuse prewritten code. This is the principle of factorization (write once used many times)”. It is very dangerous to keep the code in source text form because it can be easily exposed to accidental changes
  + Any program written by you or an application with a reasonal size contains a graphical base, a data base and some computing libraries from which you take certain functions. One thing is to keep them in the source code and another way is to keep them in another way
  + How can we reuse code in the “text” form? The ‘include’ directive. This is the command given to the preprocessor. It will take the source code, work with it and provide a text. In assembly language we use ‘%’. Include takes the name of the file and incorporate it in the present file. Therefore, the size of our source code will grow, so it’s not very efficient
  + It is best to have binary files (already compiled files). Existence of separate binary files implies SEPARATE COMPILATION!!
  + SEPARATE COMPILATION = in a project composed of n text files (source code files) every one of them is compiled SEPARATELY at different moments in time. The final executable file will be obtained as a final step LINKING TOGETHER these binary .obj files using the LinkEditor as a tool.

Techniques and tools

* Using directive %include DOES NOT MEAN SEPARATE COMPILATION ⬄ YOU WILL NOT OBTAIN MULTI-MODULE PROGRAMMING
* The linkeditor may obtain a static or dynamic library instead of an .exe file
* Assemblers and compilers check the syntactic validity and generates bytes turning them into an obj files (?).
* THE LINKEDITOR DOESN’T CARE FROM WHERE YOU GET THE OBJ FILE.

Why put together 2 modules ? We want to cooperate. How can they cooperate? We need some data exchange. How is the data exchange performed? We will have import/export mechanism. For these reasons we have 2 directives: global and extern

Global - we want them to be public.

Extern – is an importing mechanism. U tell the assembler to generate … .

In c the exporting mechanism when we have multi module programming is implicit. For importing is the same extern import.

* Calling a function implies a lot of preparation. Maybe it is more efficient to perform a jump, or to multiply the source code in many places
* Ex: recursive calling of a procedure is not efficient. It might be better to go again and again to a source code without the preparation of calling, starting, etc.
* In assembly language using macros is useful because we can’t pass parameters to functions and procedures bc we don’t have them. You have to simulate them by jumps and loops. In case of a macro, you can parameterize it.

In main.asm and sub.asm (the long code)

What resources are shared between the 2 asm files?

Answer: Sirfinal, concatenare, EAX, EBX (in eax and ebx you put sir1 si sir2. Technically sir1 and sir2 are not shared, only where they are stored. THE REGISTERS ARE ALWAYS SHARED RESOURCES)

IN PRINCIPLE, registers are shared. But in this particular case ONLY EAX AND REGISTERS ARE SHARED.

Usually the stack is shared only if you use push or pop. But in an explicit way the stack is always shared because the return location is saved on the stack. IN THESE CASES BOTH ANSWERS ARE CORECT.

3 CATEGORIES OF SHARED RESOURCES

* Symbols
* Stack
* Registers

NASM + HIGH LEVEL LANGUAGES

* What should we be aware of in regard of the relation between and assembly module and a high level language? To build a new stack frame (?)
* If your functional procedure has parameters you need to transfer them. How many ways are there to pass the parameters? 9, but we only need the first 2:
  + Call by value
  + Call by reference
  + Call by text (?)
  + Returning the result

INTERFACING WITH HIGH-LEVEL LANGUAGES

Call by value – you work with a copy (good when it comes to safety)

Calling conventions

* refer to how parameters are passed
* What resources are volatile (you lose them)
  + EAX is a volatile resource
  + ECX
  + EDX (usually you have multiplication or division)
  + EFLAGS
* Non-volatile resources
* Where the result is stored in case of a function
  + Ex: AL or AX or DX:AX or EAX
* What cleanup actions are required after the call
* Conventions
  + STDCALL
  + CDECL
  + 2 differences
    - How many parameters we have: Only CDECL functions can have a variable (flexible) number of variables
    - Who is responsible for cleanup: In CDECL it is the caller, in STDCALL is the callee

WHY IS THE STACK SO IMPORTANT? WHY DO 3 REGISTERS DEAL WITH THE STACK? The stack reflects the order in which the program views the procedures and functions.

IMPLEMENTATION OF SUBROUTINES IN PROGRAMMING

Final goal: to understand the behavior of procedures and functions and how their relation works

Any relationship that involves calling another subroutine has to follow these 3 steps (phases)

* Call phase
  + It will have an associated call code
  + The compiler will generate in an automatic way the ‘call code’
* Entry phase
  + Involves an entry code
  + The compiler will generate in an automatic way the ‘entry code’
* Exit phase
  + Involves an entry code
  + The compiler will generate in an automatic way the ‘exit code’

Only the high level compiler generates these 3 codes automatically. When we want the assembly to be combined, WE NEED TO WRITE THEM BY SIMULATING THEM IN ASSEMBLY LANGUAGE. We have to obey the c rules

CALL CODE

* Save volatile resources – EAX, ECX, EDX, EFLAGS (Compiler saves them automatically, but if you are in assembly case, if they do not involve these resources, we can ignore them. In the case it affects 2 of the resources, we can only save 2)
  + Always make sure ESP is Aligned and DF is 0
* Passing parameters
* Perform the call
  + Save the returning address and jump to the initial instruction of the new function

YOU DON’T HAVE A CALL PHASE ENTRY PHASE EXIT PHASE when you have the combination of 2 .asm modules. All we are talking about implies high level programming language because that is what Assembly wants: to link with a high level programming language

Example presented:

the interaction with c is in printf

entry phase and exit phase are automatically generated by c compiler IN THIS CASE

In this case, our volatile resources are EAX AND ECX AND WE ONLY SAVE THEM because printf modifies nothing else, and we use eax and ecx for printing the 10 digits. You CAN save them all if you want

Passing the parameter: The number we have to display (the second ‘push eax’)

We always need to know who is responsible for creating which steps

|  |
| --- |
| Returning Adress |
| Parameters |
| Volatile Resources |
| Rest of the program  … |

ENTRY CODE

* It is the responsibility of the callee
* In the last example, printf is responsible for generating
* Steps:
  + Configuring/building the new CURRENT stack frame
    - Push EBP
    - Mov EBP, ESP
  + Reserving space for local variables
    - Sub ESP, nr\_bytes
  + Save the non-volatile resources which I know that will be affected

EBP

ESP

|  |
| --- |
|  |
|  |
| EBP |
|  |
| … |

EXAMPLE:

Changedir function

Nonvolatile resources: EBX, ESI, and we can consider that it’s even EBP

Recoursive Example:

THE INITIAL CALLING IS MADE BY C Module, but the 2nd, 3rd, fourth are done by the assembler

|  |
| --- |
|  |
|  |
|  |
| ESP -> non-volatile Registers |
| Local variables |
| EBP -> Calls EBP |
| Returning Adress |
| Parameters |
| Volatile Resources |
| Non-volatile Registers |
| Local variables |
| Calls EBP |
| Returning Adress |
| Parameters |
| Volatile Resources |
| Rest of the program  … |

EXIT CODE:

* Steps
  + Restore altered non-volatile resources (reverse of last operation of entry)
  + Release the local variables of the function (kind of revers of penultimate operation of entry)
    - This is kinda useless, it might never happen because we deallocate
  + Deallocate the stack frame
  + Returning step
    - Returning from the function
    - Releasing the arguments

subiect examen? Care is fazele, in ce consista si cand trebuie programatorul sa le implementeze in mod direct