SOFTWARE SYNTHESIS FOR EMBEDDED PROCESSORS

Lochana Abhayawardana

lochana.abhayawardana@stud.hshl.de

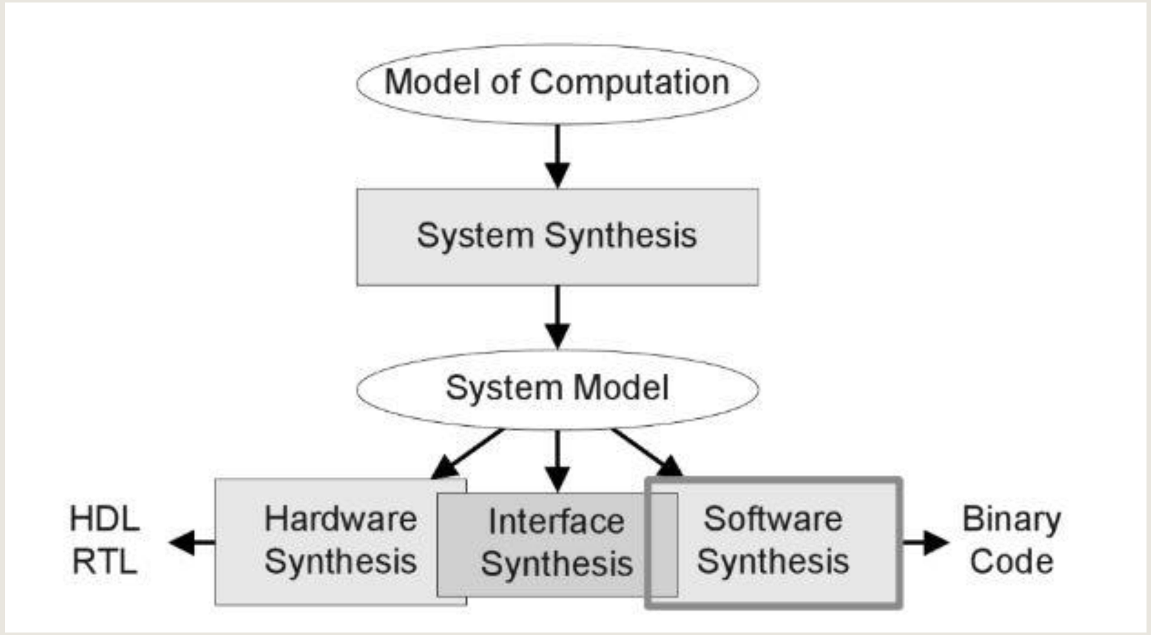
AGENDA

INTRODUCTION

- Embedded system in now days
- Software Synthesis – a critical role in reducing the design complexity
- Different approaches of the software synthesis
    1. High level synthesis: High-level programming languages play a critical role when it comes to programming nowadays.
    2. Register Transfer level synthesis: process of compiling high-level languages to computer-understandable assembly code
    3. Instruction set synthesis: interface between software and hardware, which facilitates the execution of diverse programs written in various programming languages
    4. principles of RISC architecture and ARM architecture are discussed

INTRODUCTION TO SOFTWARE SYNTHESIS FOR EMBEDDED SYSTEMS

- Software synthesis is a part of component synthesis together with Hardware synthesis and Interface synthesis.
- The system synthesis produces the system model to describe the system's components and their communication.
- Each part below System model uses it as an input and generates an implementation for each Hardware, interface and software synthesis.
- The Software Synthesis produces binary code for programmable processing elements.



[1]

APPROACHES TO SOFTWARE SYNTHESIS

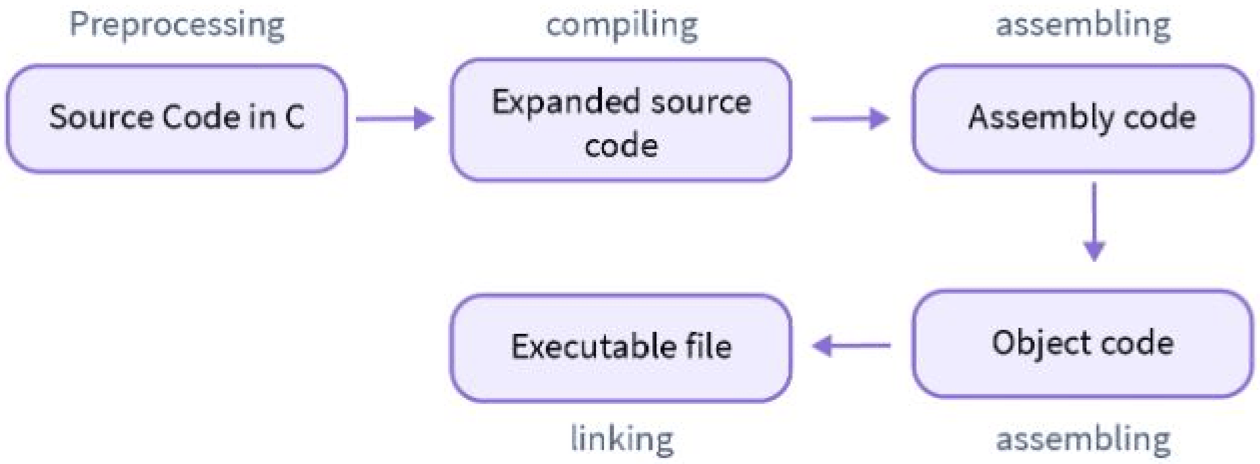
- Most of the levels of embedded systems are a part of the physical control process such as airbags in cars.
- Embedded software can implement a control loop of the physical process through getting readings from the sensors.
- Therefore, the software is specific to the underlying hardware, in order to achieve that embedded systems use specialized components (hardware accelerators, processors, DPSs) with communication schemes.

PROGRAMMING IN HIGH LEVEL-LANGUAGES

- Languages that are close to human languages and that can also understand by computers.
- Although they are a bit different from human languages when it comes to syntax and semantics.
- To write a program in high-level language we need an IDE or a text editor. E.g.: Visual Studio.
- For a high-level language, we can use C language.
- Even though using high-level language makes the process easier and more understandable for humans, computers do not understand high-level languages.
- computers can only understand binary numbers.
- Therefore, we need translators to lower the level of programming language to the assembly level.

COMPILERS

- The high-level languages must be translated into low-level in order to compute by the computers.
- The compiler can be used to create an executable binary directly, object files .o , also known as executable machine code
- Assembly code .S,
- Linker operations such as creating executable programs (.exe, .out, .bin, .elf),
- Libraries (.a, .dill) [2]

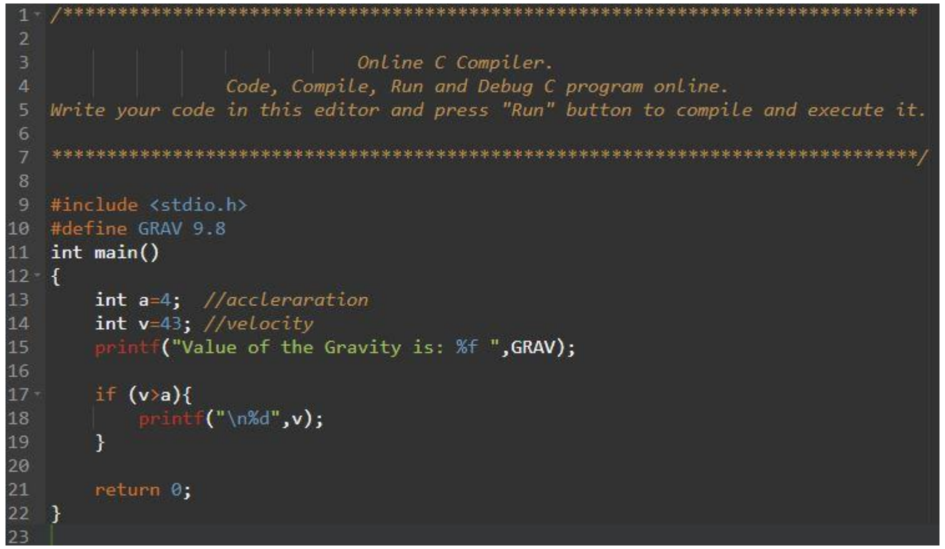


[2]

PRE-PROCESSING

- Pre-processing is the first step in the compilation process in C
- It is performed by the pre-processor tool (A pre-written program invoked by the system).
- All the statements with the symbol of # in a C program are processed by the preprocessor into an intermediate file with no # symbols.
- Steps, the pre-processing tasks are performed.

1. Comments removal : Comments are used to give a general idea of the code and its function to the humans. In synthesis there is no reason to keep them to use by the computers. Comments are represented by // or /* */in c language.



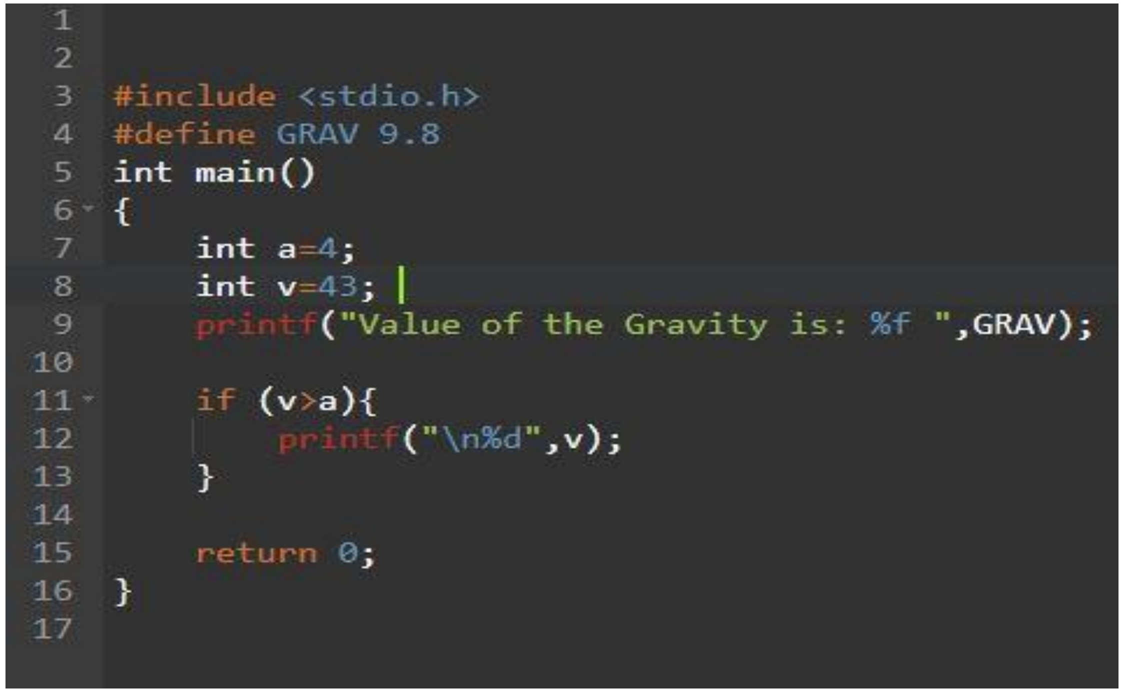
```
/******************************************************************************

                            Online C Compiler.
                Code, Compile, Run and Debug C program online.
Write your code in this editor and press "Run" button to compile and execute it.

*******************************************************************************/

#include <stdio.h>
#define GRAV 9.8
int main()
{
    int a=4;  //accleraration
    int v=43; //velocity
    printf("Value of the Gravity is: %f ",GRAV);

    if (v>a){
        printf("\n%d",v);
    }

    return 0;
}
```

2. Macros expansion:

- Marcos are constant values or definitions defined by the #define directives in C Language. [3]
- The pre-processor creates an intermediate file where the pre-written assembly-level instructions replaced the defined expressions or constants, and a '+' sign is added to every macros expanded statement.

Examples: Defining a value #define GRAV 9.8

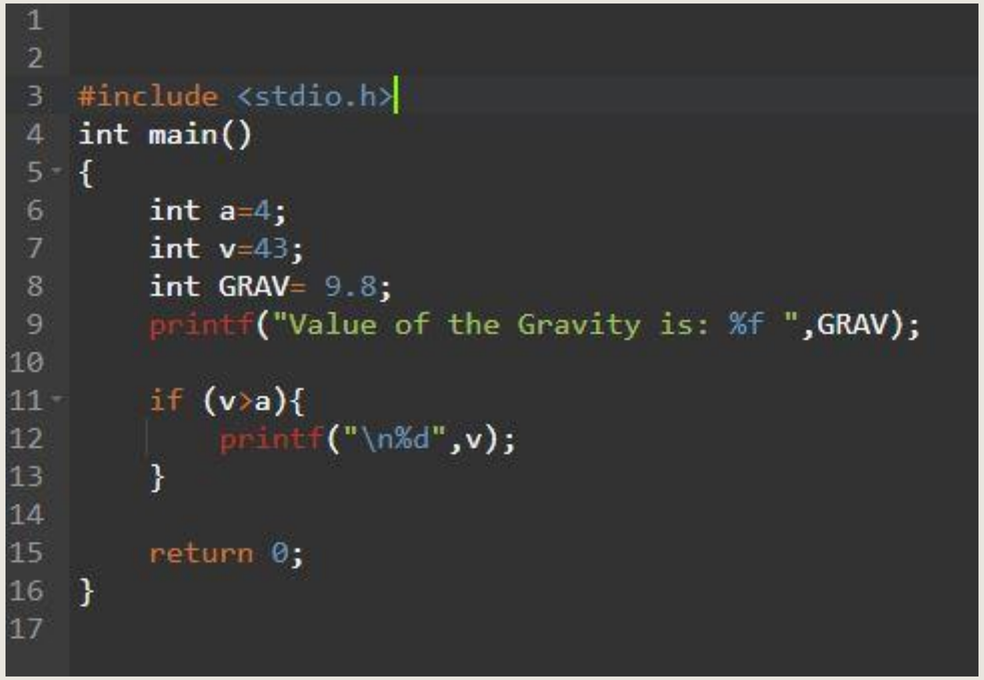
Defining an expression #define SUM(a,b) (a + b)



```c


#include <stdio.h>
#define GRAV 9.8
int main()
{
    int a=4;
    int v=43;
    printf("Value of the Gravity is: %f ",GRAV);

    if (v>a){
        printf("\n%d",v);
    }

    return 0;
}

```

3. File inclusion:

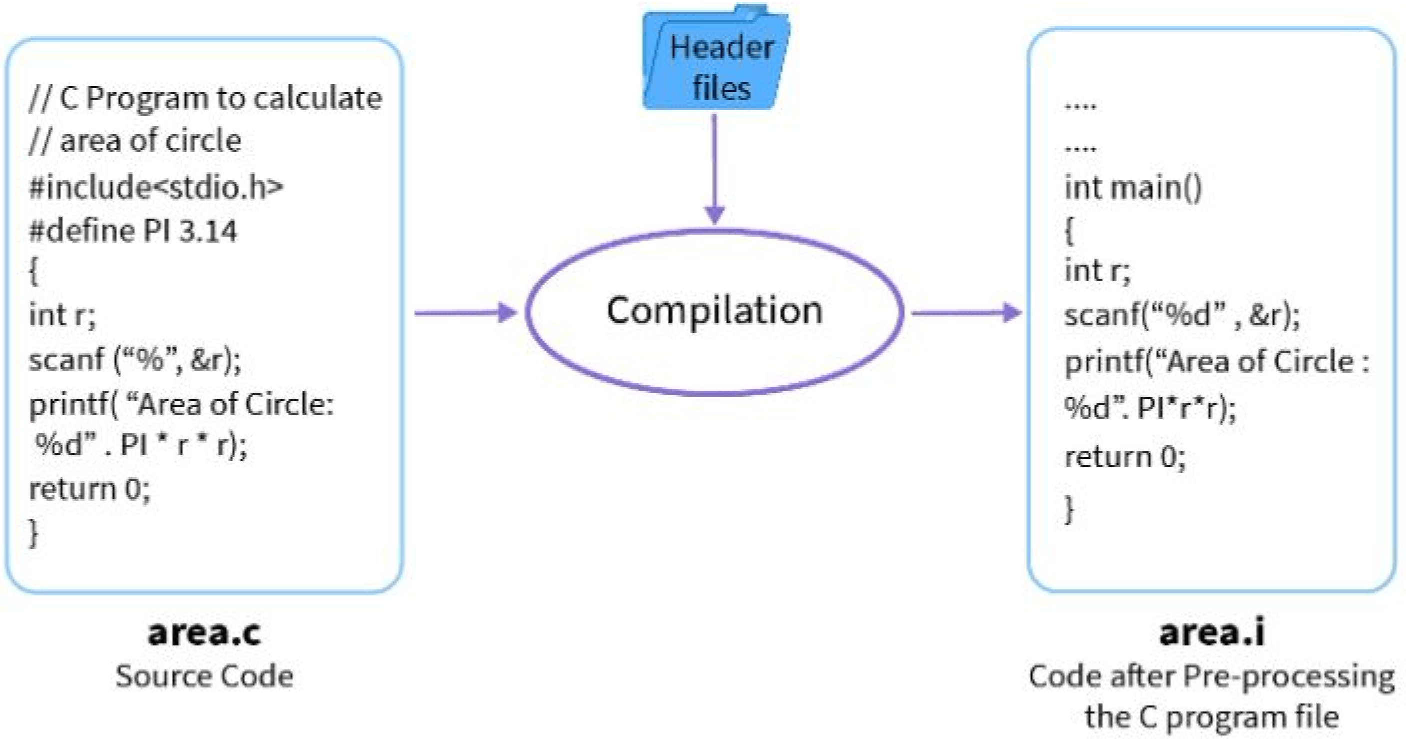
- The process of adding another file containing pre-written code into our C program during the pre-processing.
- This is done using the #include directive.
- If the functions like pritf() and scanf() are used, the C program must include the pre-defined standard input output header file as, #include .



```c


#include <stdio.h>
int main()
{
    int a=4;
    int v=43;
    int GRAV= 9.8;
    printf("Value of the Gravity is: %f ",GRAV);

    if (v>a){
        printf("\n%d",v);
    }

    return 0;
}

```

3. Conditional compilation:

- Like in the previous Marcos expansion here in Conditional compilations,
- It checks all the conditional compilation directives with some pre-defined assembly code and passes a newly expanded file to the compiler.
- This is performed using commands like #ifde, #endif, #ifndef, #if, #else and #endif in C programming. [3]



```
// C Program to calculate
// area of circle
#include<stdio.h>
#define PI 3.14
{
int r;
scanf ("%", &r);
printf( "Area of Circle:
 %d" . PI * r * r);
return 0;
}
```

area.c
Source Code

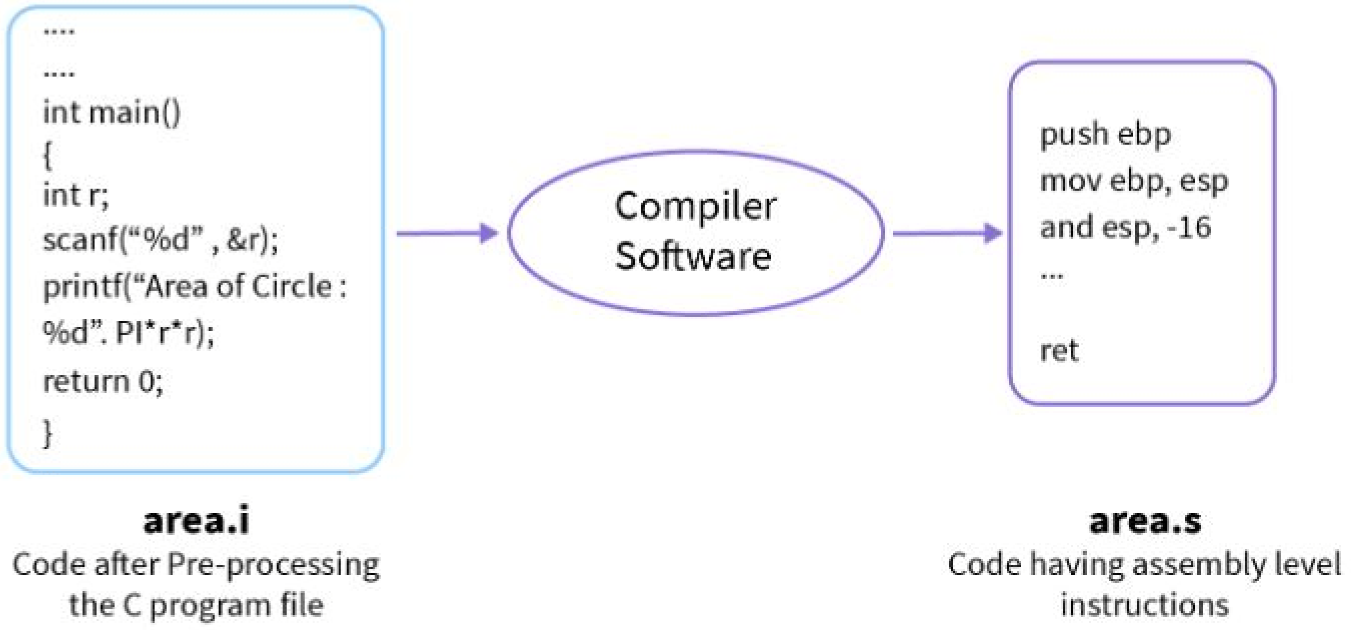
```
....
....
int main()
{
int r;
scanf("%d" , &r);
printf("Area of Circle :
%d". PI*r*r);
return 0;
}
```

area.i
Code after Pre-processing the C program file

[2]

5. Compiling:

- C uses an inbuilt compiler software to convert the intermediate .i files into Assembly files .s which include low-level code.
- This process tells us about any syntax errors or warnings present in the source code through the terminal.

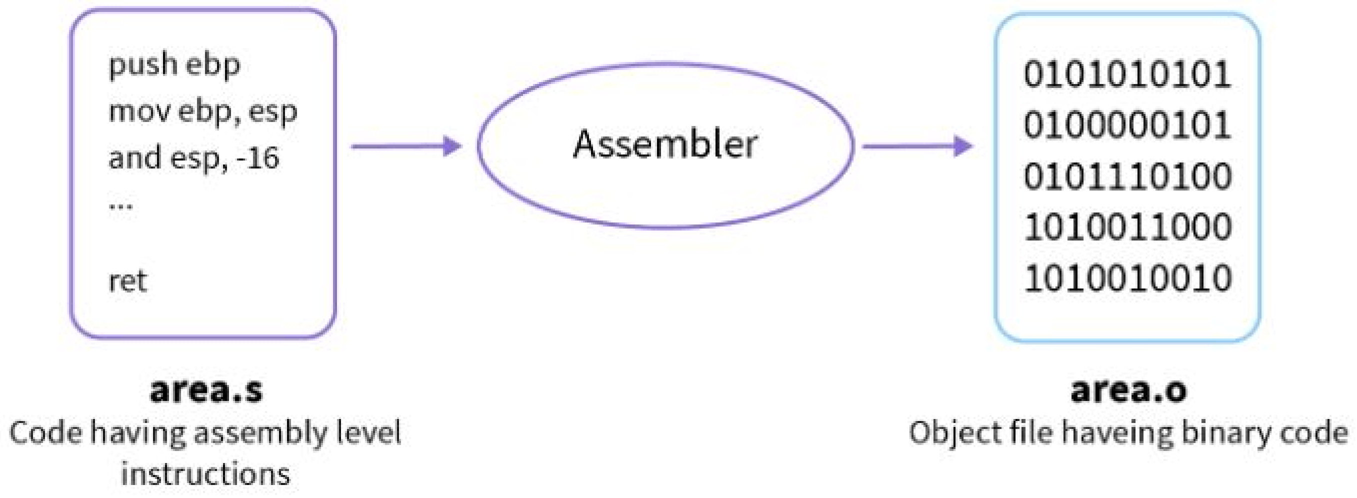

```
....
....
int main()
{
int r;
scanf("%d" , &r);
printf("Area of Circle :
%d". PI*r*r);
return 0;
}
```

```
push ebp
mov ebp, esp
and esp, -16
...

ret
```
Compiler Software
area.i
Code after Pre-processing the C program file
area.s
Code having assembly level instructions

[2]

6. Assembling:

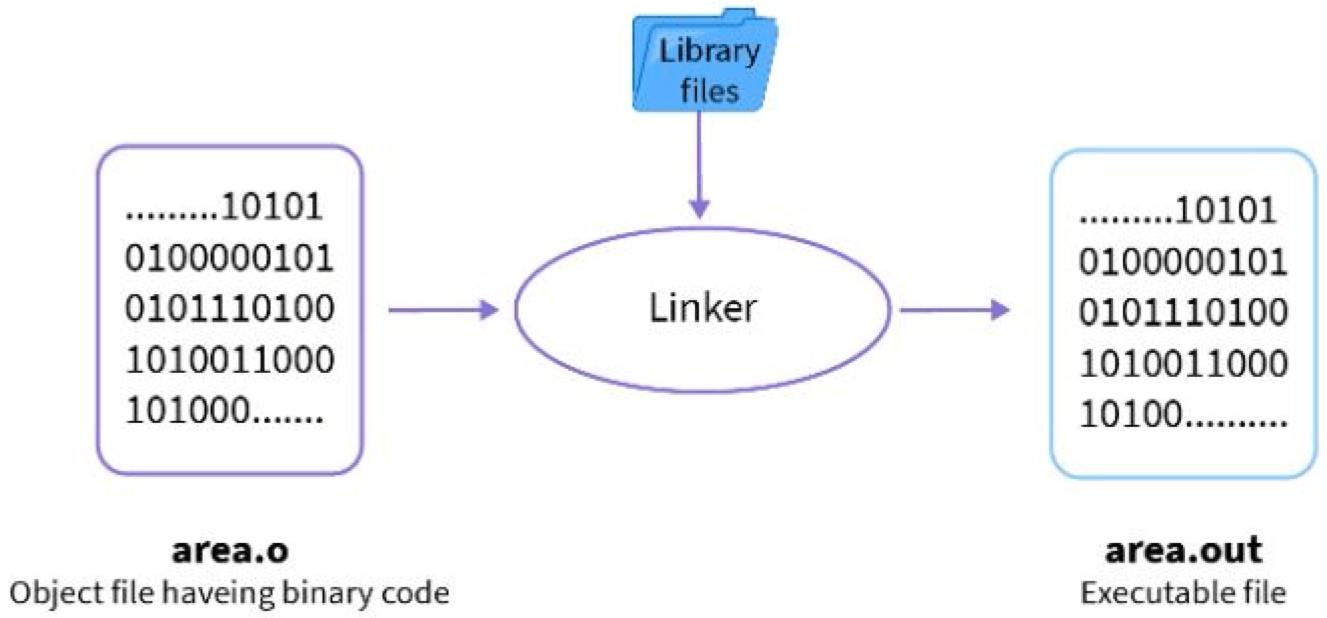
- converts the assembly level code to machine-understandable code using an assembler.
- Assembler is a pre-written program that translates assembly code into machine code.
- The generated file includes the same file name with the extension of .obj in DOS and .o in UnIX OS. [3]



[2]

7. Linking

- Process of including the library files into a program.
- The library file contains the definition of the machine language functions and the files containing the extension of .lib.
  (like when we read a book, we refer the unknown words with the use of a dictionary.)
- The use of the Library gives the meanings of unknown statements of object files.
- This process is an executable file with the extension of .exe for DOS and verb—.out— in Unix OS.



[2]

INSTRUCTION SET ARCHITECTURE (ISA)

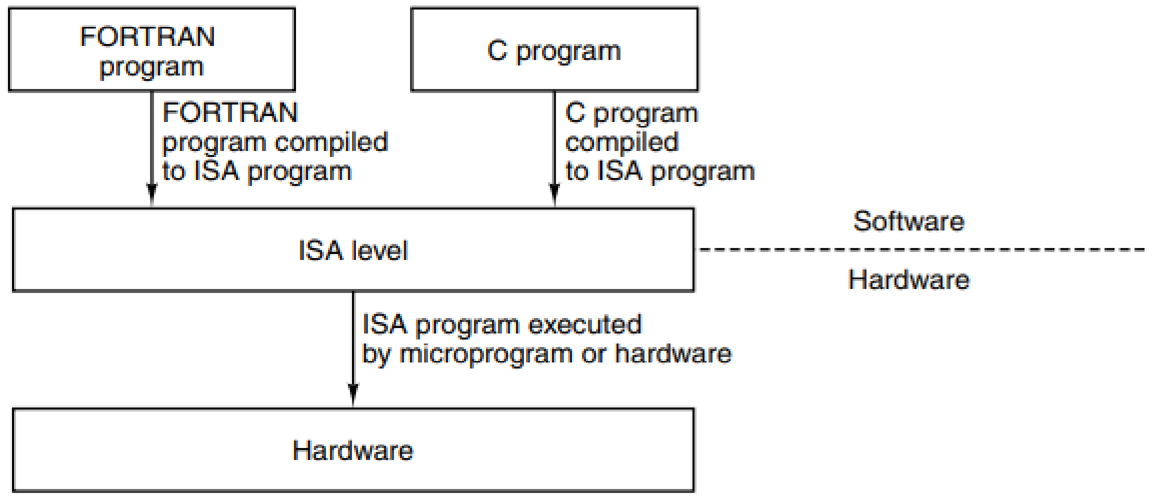
- When designing computers engineers had to produce a way to connect software to hardware.
- The ISA defines the type of instructions to be followed by the processor [4].
- In the following sections, the MIPS ISA is used as an example to discuss the process since it Is the most used due to its simplicity.
- Based on the type of operations, the instructions are classified into three types,

1. Arithmetic /logic instructions: Performs Arithmetic and logical operations on one or more operands.

2. Data Transfer instructions: Responsible for the transfer of instructions from memory to the processor and back.
3. Branch and jump instructions: Responsible for breaking the sequential flow of instructions and jumping instructions in the implementation of the function and conditional statements. The ISA also defines the maximum length of each type of instruction. Since the MIPS is using 32-bit ISA, each and every instruction must be accommodated within 32 bits. [3]

ISA

- The ISA level is important for the system architects as it is the interface between the software and hardware. [5]
- It might be possible to write hardware directly to execute programs written in C, C++, Java or some other high-level language, but it would not be a great idea.
- Most computers must have the ability to execute programs written in multiple languages.
- Therefore, the most suitable way to execute more programming languages is to convert them into a common intermediate form and build hardware which can execute ISA-level programs directly. [5]
- The process can be shown in the bellow diagram Reduced Instruction Set Computer (RISC) and Arm CPU architecture are mostly used in the industry.


FORTRAN program
C program
FORTRAN program compiled to ISA program
C program compiled to ISA program
ISA level
Software
Hardware
ISA program executed by microprogram or hardware
Hardware

REDUCED INSTRUCTION SET COMPUTER (RISC)

- At the start of RISC, the people who were working on microprogramming tools began to rethink the architectural design principles trying to close the “semantic gap” in high-level programming languages and microinstructions.
- But this approach introduced a “performance gap” and was not successful in achieving the desired performance goal. Therefore, a new design philosophy emerged, with the idea of optimizing compilers that can be used to compile high-level languages into instructions.
- These instructions are simpler, executed in one cycle, and have a reduced set compared to the previous set of instructions used. [6]

RISC

- Key principles of RISC

1) Functions should be simple, introducing a new operation that increases the cycle time must be justified by a reduction in the number of cycles required.
2) Microinstructions should not be faster than simple instructions, since the cache memory is implemented using the same technology as the writable control store. Simple instructions should operate at the same speed as microinstructions.
3) Moving software into microcode does not improve its performance or functionality. Everything that can perform on micro coded machine can also be performed in assembly language on a simpler machine. The run-time library architecture includes characteristics of microcode but is easier to modify.
4) The RISC architecture prioritizes simple instruction decoding and pipeline execution. Instructions are broken into parallelizable parts making the new instructions to start executing every cycle. This approach improves performance.
5) Use of compiler technology to simplify instructions.
6) RISC compilers keep operands in registers for simple register-to-register instruction, enabling shorter instruction formats for memory-based operands. [6]

ARM CPU ARCHITECTURE

- ARM is a RISC architecture.
- The ARM ISA is a loadstore one, instructions that operate only on registers and are separated from instructions that access memory.
- All ARM instructions are 32-bit long and mostly they have three-operand encoding.
- ARM architecture is equipped with a large register file with 16 general-purpose registers, allowing pipelining of the ARM architecture.

A. Registers:

- ARM provides 16 general-purpose registers in the user mode.
- Register 15 is the program counter or can change a general-purpose register as well.
- Register 14 is used as branchand-link instruction.
- Register 13 is used as a stack pointer. The current program status registers four 1-bit condition flags (‘Negative’, ‘zero’, ‘Carry’, ‘overflow’) and four fields reflecting the execution state of the processor.
- The ‘I’ and ‘F’ flags enable normal and fast interrupts respectively. The ‘mode’ field selects one of seven execution modes.
- Undefined instruction mode is entered when the processor attempts to execute an instruction that is supported neither by the main integrated core nor coprocessors. This mode is used to implement coprocessor emulation.
- System mode is used to run privileged operating system tasks.
- Abort mode is used when responding to memory faults. [7]

B. ARM architecture exceptions

- ARM architecture exceptions Reset starts the processor from a known state and renders all other pending exceptions irrelevant.
- Data abort exception is raised when a load or store instruction violates memory access permissions.
- Fast interruption exception is raised when the processor receives a signal from a fast interrupt source.
- Normal interrupt exception occurs when the processor receives a signal from any non-fast interrupt source.
- Perfect abort occurs when memory access permissions are violated during an instruction fetch.
- Software interrupt occurs on requests to an operating system service.
- Undefined instruction exception is generated when trying to decode an instruction that is supported neither by the main integer core nor by one of the coprocessors. [7]

CONCLUSION

- software synthesis plays a critical role in embedded software development.
- This report discusses various approaches to software synthesis, including high-level synthesis register transfer level synthesis, and instruction set architecture.
- High-level programming enables human-readable code, then translates to assembly-level instructions through compilers.
- Instruction Set Architecture (ISA) acts as an interface between software and hardware, enabling the execution of programs written in different languages.
- RISC architecture, with its simplicity, optimizing compilers and efficient execution.
- ARM architecture adapts the above-mentioned characteristics.
- **Using these technologies and concepts engineers can develop advanced embedded systems for various applications from mobile phones to automotive and various electronic applications.**

REFERENCES

[1] G. C. Buttazzo, Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications. Springer Science & Business Media, 2011.

[2] E. Artistry. Lesson: Compilers. [Online]. Available: https://embeddedartistry.com/lesson/compilers/

[3] Scaler. Compilation process in c. [Online]. Available: https://www.scaler.com/topics/c/compilation-process-in-c/

[4] GeeksforGeeks. Microarchitecture and instruction set architecture. [Online]. Available: https://www.geeksforgeeks.org/microarchitectureand-instruction-set-architecture/

[5] A. S. Tanenbaum and T. Austin, Structured Computer Organization. Pearson, 2012.

[6] D. A. Patterson, “Reduced instruction set computers,” Communications of the ACM, vol. Volume 28, 1985.

[7] L. Ryzhyk, “The arm architecture,” 2006.

THANK YOU

Lochana Abhayawardana

Lochana.abhayawardana@stud.hshl.de