

Software RE

Rev II



DR. TAREK A. TUTUNJI

REVERSE ENGINEERING
PHILADELPHIA UNIVERSITY, JORDAN
2015

References



- Reversing: Secrets of Reverse Engineering by Eldad Eilam. Published by Wiley Publishing 2005
- Reversing: Reverse Engineering, Recent Advances and Applications edited by Telea. Published by InTech 2012
- Reverse Engineering for Beginners by Dennis Yurichev 2015

Software is Everywhere



- PCs and Laptops
- Mobile phones
- Networks
- Washing machines
- Microwaves
- Automated Industry Controllers
- Automobiles
- Airplanes
- Spaceships

Software is Everywhere



We rely on too much software that we do not understand and do not know very well at all.

- We buy software packages.
- We run setup utilities that install numerous files, change system settings, delete or disable older versions , and modify critical registry files.
- We access websites that might interact with programs
- We purchase CD games
- We download programs , updates, and patches

S/W Reverse Engineering



- S/W RE is **the process of analyzing a system to identify its components and their interrelationships and to create representations of the system in another form or at a higher level of abstraction** -- IEEE 1990
- S/W RE is about opening up a program's "box" and looking inside
- S/W Reverse engineering is a critical **set of techniques and tools for understanding** what **software** is really all about.

S/W Reverse Engineering



- The techniques of analysis, and the application of automated tools for software examination, give us a reasonable way to comprehend the complexity of the software and to uncover its truth.
- Reverse engineering occurs every time someone looks at someone else's code.
- Reverse engineering is a discovery process.

What does this program do?



```
int A[SIZE] = {54,15,32,78,23,90,48,86,23,65};
int i, j, min, temp;

for (i = 0; i < SIZE - 1; i++){
    min = i;

    for (j = i+1; j < SIZE; j++){
        if (A[j] < A[min]){
            min = j;
        }
    }

    temp = A[i];
    A[i] = A[min];
    A[min] = temp;
}
```

S/W RE



- S/W RE involves skills:
 - Code breaking
 - Puzzle solving
 - Programming
 - Logical analysis

Applications



- There are **two main categories** of reverse engineering **applications**:
 - Security-related
 - Software development–related.

Security Applications



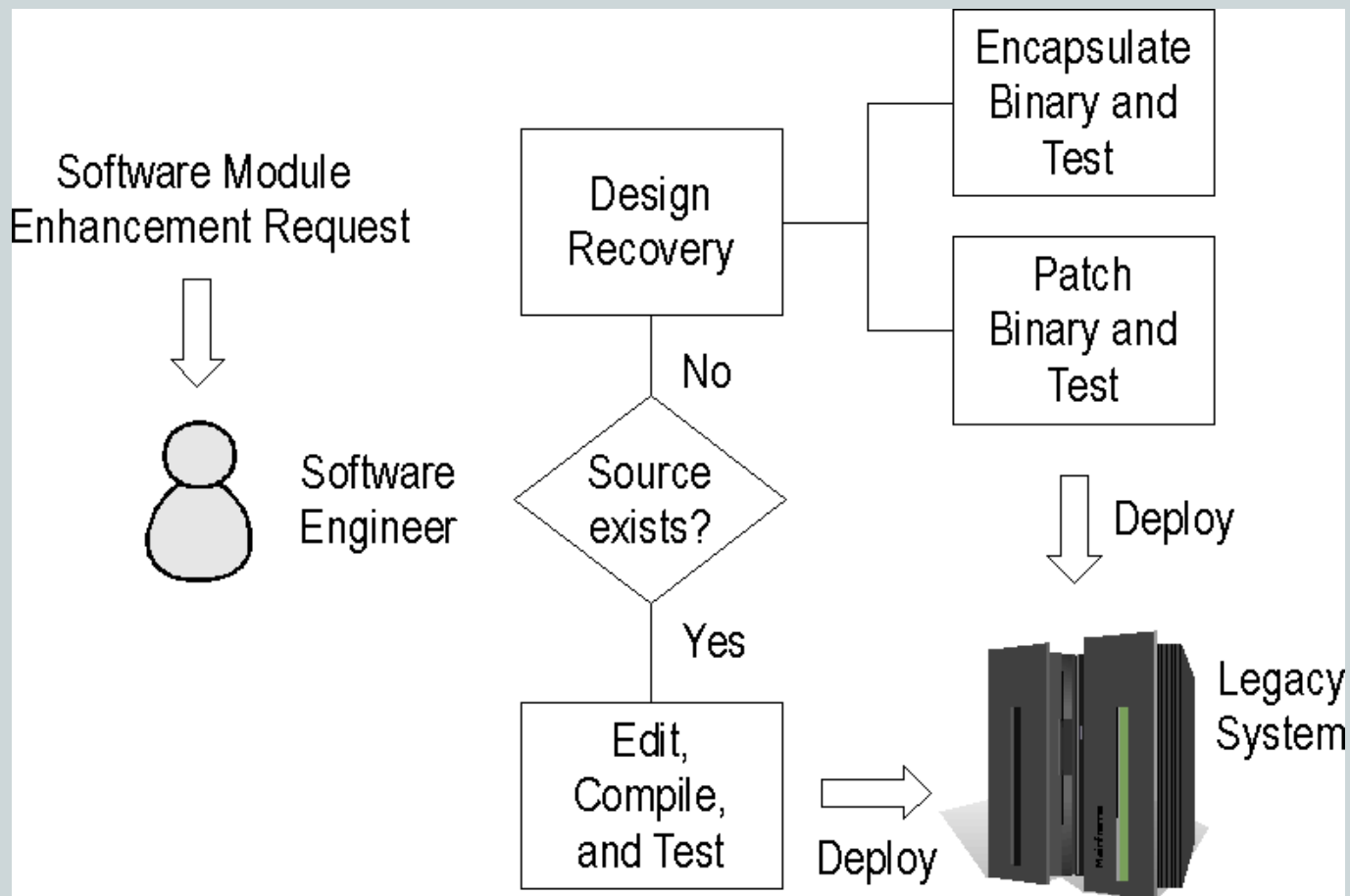
- **Finding malicious code**
 - RE can be used to detect viruses and worms by understanding how the code is structured and functions.
- **Discovering unexpected flaws and faults**
 - RE can help identify flaws and faults (bugs) in application software before they are released to the public.
- **Cracking**
 - To crack a program, means to trace and use a serial number or any other sort of registration information, required for the proper operation of a program.
 - RE can provide that information by decompiling a particular part of the program.

Software Development Applications



- **Learning from others' programs**
 - Re-use the code in other programs
 - Learn and build on a growing body of code knowledge.
 - RE techniques can enable the study of advanced software approaches.
- **Discovering features or opportunities**
 - Existing techniques can be reused in new contexts.
 - RE can lead to new discoveries about software and new opportunities for innovation.
- **Developing Competing Software**
 - Observe and understand competitor's design
 - Determine if another company used your code

Legacy Software Development Process

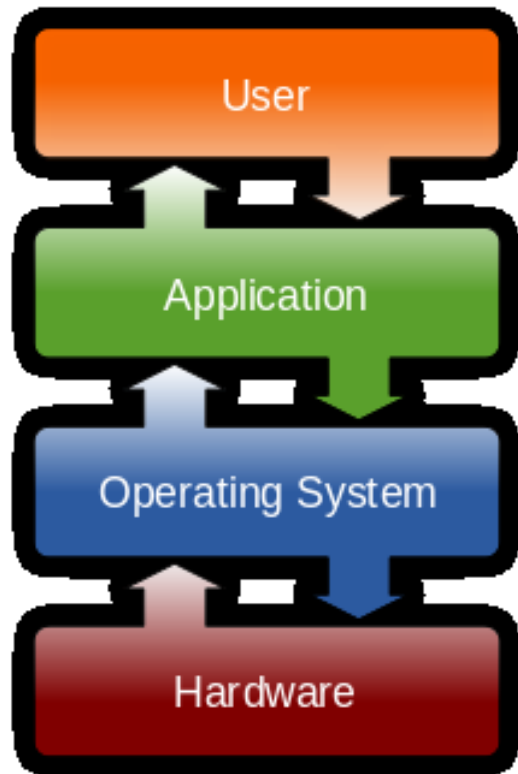


S/W RE Basics



- In order to perform software RE, a good understanding of the computer H/W and S/W is needed.
- The following slides provide some basics about
 - Assembly Language
 - Operating Systems

Software Layers



Hardware

lowest level, machine code
x86 / CISC / RISC / etc.

Assembly languages

software level, drivers
NASM / TASM

System languages

OS level, high performance
C / C++ / C# / Go

Application languages

productivity, portability, business logic
Java / PHP / Visual Basic

Low Level Software



- Computers and **software are built layers upon layers.**
 - At the bottom layer, there is the microprocessor
 - At the top layer, there are some elegant looking graphics, a keyboard, and a mouse—the user experience.
- Most software developers use high-level languages that take easily understandable commands and execute them.
- Reversing requires a solid understanding of the lower layers.
- Reversers must literally be aware of anything that comes between the program source code and the CPU.

Assembly Language



- The lowest level in the software chain
- If software performs an operation, it must be visible in the assembly language code.
- Assembly language is *the language of reversing*.
- Assembly language vs. Machine Code

Assembly Language



- An ***assembler program*** translates the textual assembly language code into binary code, which can be decoded by a CPU.
- A ***disassembler does the exact opposite***. *It reads object code and generates the textual mapping of each instruction in it.*
 - **Disassemblers are a key tool for reverse engineers**

Compilers



- A text file containing instructions that describe the program in a high-level language is fed into a *compiler*.
- A **compiler** is a program that takes a source file and generates a corresponding machine code file while **Decompilers** do the opposite
- Depending on the high-level compilers for programming languages (such as C and C++), decompilers generate machine-readable object code from the textual source code.

Operating Systems



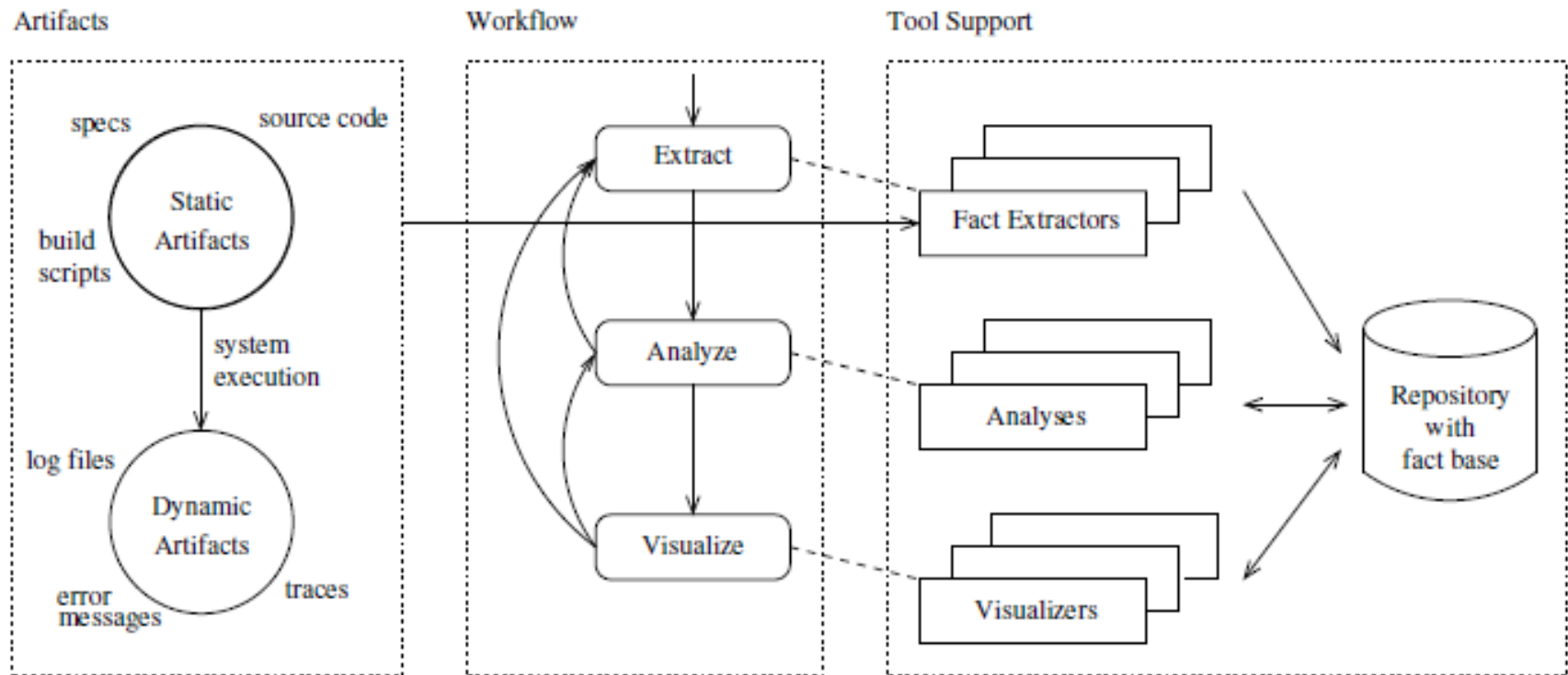
- An operating system is a program that **manages** the computer, including the **hardware** and **software applications**.
- An operating system takes care of many different tasks and can be seen as a kind of **coordinator between the different elements in a computer**.
- Operating systems are such a **key element** in a computer that any reverser **must have a good understanding of what they do and how they work**

RE Process



- RE can be divided into two separate phases.
 - ***System-level reversing techniques*** help determine the general structure of the program and sometimes even locate areas of interest within it.
 - ***Code-level reversing techniques*** provides detailed information on a selected code chunk

RE Process



Ref: Software RE in Domain of Complex Embedded Systems
by Kienle, Kraft and Muller

System Level RE



- **System-level reversing** involves running various tools on the program and utilizing various operating system services to obtain information, inspect program executables, track program input and output, and so forth.
- Most of this information comes from the operating system, because by definition every interaction that a program has with the outside world must go through the operating system.

Code Level RE



- **Code-level reversing** observes the code from a very low-level
- Software can be highly complex
 - Even with access to a program's well-written and properly-documented source code can be difficult to comprehend
- **Extracting design concepts** and algorithms **from a program binary** is a complex process that requires a mastery of reversing techniques along with a solid understanding of software development, the CPU, and the operating system.

RE Tools



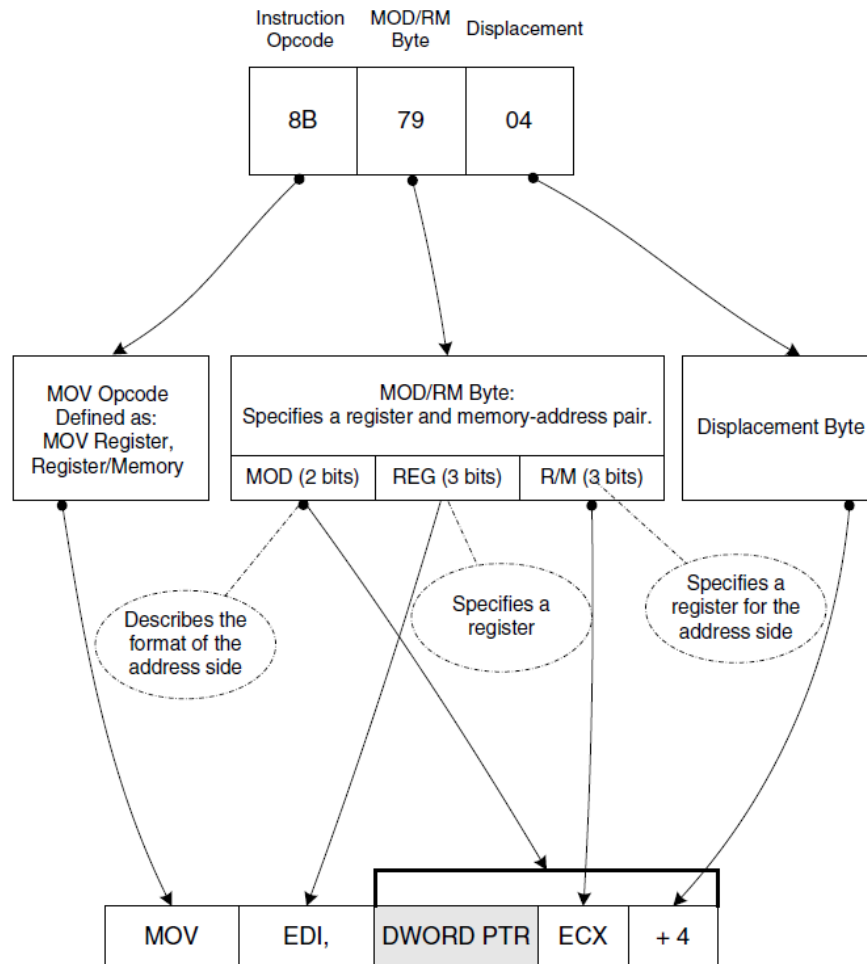
- Disassemblers
- Debuggers
- Decompilers
- System Monitoring Tools

Disassemblers



- A disassembler **decodes binary machine code into a readable assembly language** text.
- The disassembler process
 - Looks up the opcode in a translation table that contains the textual name of each instructions along with their formats.
 - Analyze which operands are used in a particular instruction.
- The specific instruction encoding format and the resulting textual representation are entirely **platform-specific**.

Disassemblers

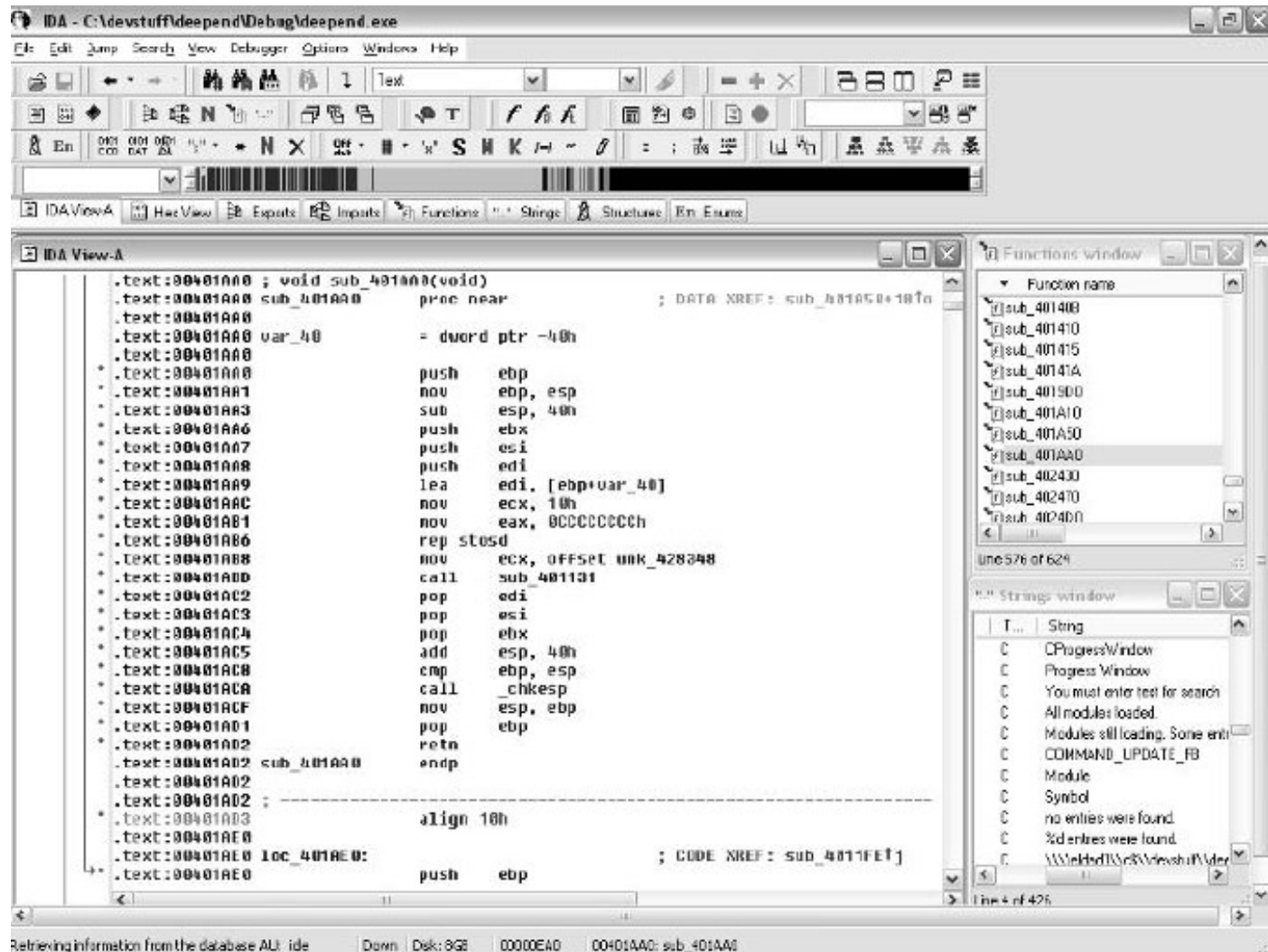


Example: IDA Pro



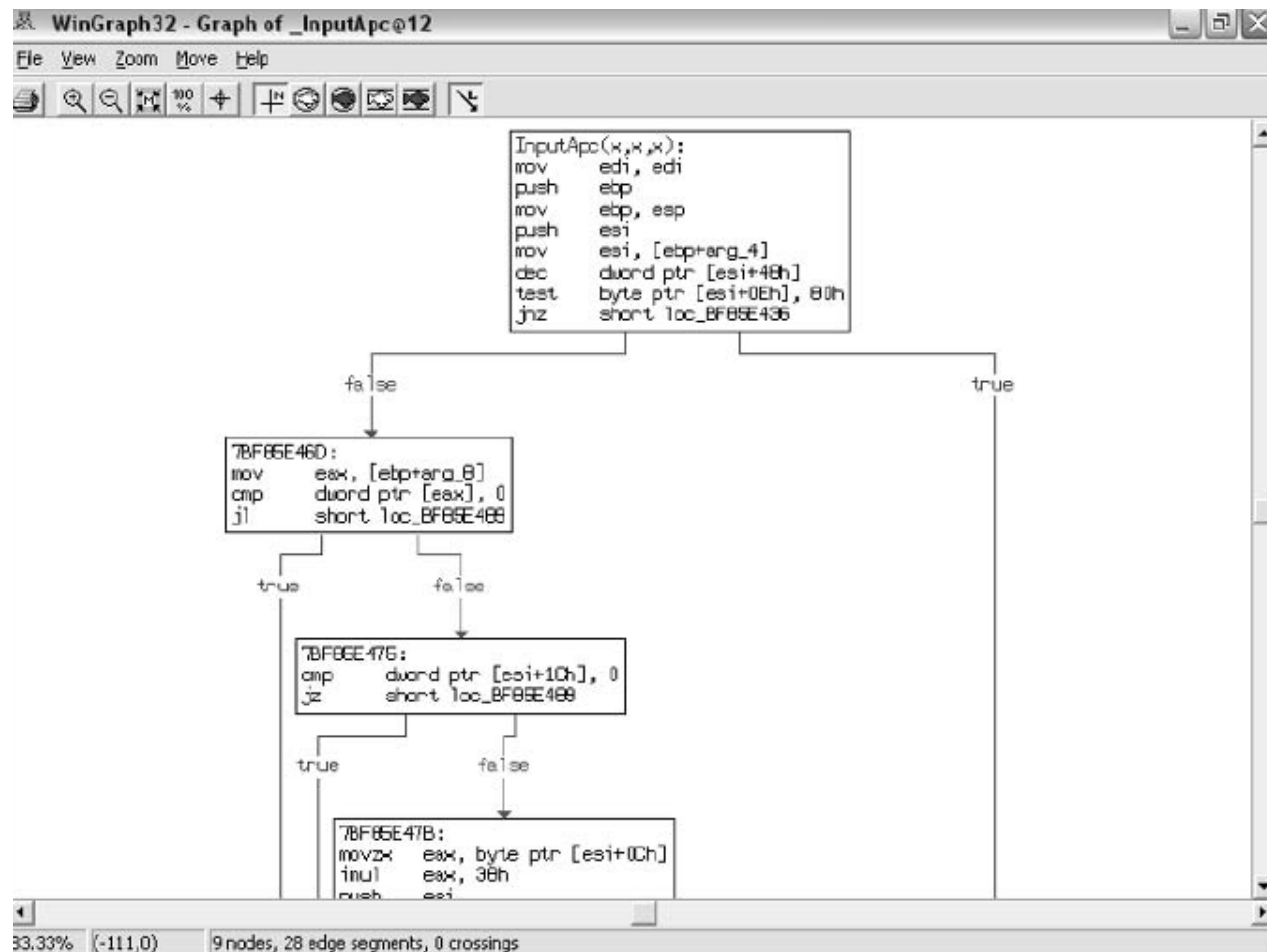
- IDA (Interactive Disassembler) by DataRescue is an extremely powerful disassembler that supports:
 - A variety of processor architectures : IA-32, IA-64 (Itanium), AMD64, and many others.
 - A variety of executable file formats, such as PE (Portable Executable, used in Windows), ELF (Executable and Linking Format, used in Linux), and even XBE, which is used on Microsoft's Xbox.
- IDA is capable of producing flowcharts for a given function. These are essentially logical graphs that show chunks of disassembled code and provide a visual representation of the program flow

Example: IDA Pro



IDA Pro screen showing: code disassembly, function and string lists

Example: IDA Pro



IDA-generated function flow

Debuggers



- Debuggers exist primarily to assist software developers with locating and correcting errors in their programs, but they can also be used as powerful reversing tools.
- The idea is that the debugger provides a disassembled view of the currently running function and allows the user to step through the disassembled code and see what the program does at every line.

Important Debugger Features



- **Powerful Disassembler**
 - View the code clearly, with cross-references that reveal which branch goes where and where a certain instruction is called from.
- **View of Registers and Memory**
 - Provide a good visualization of the important CPU registers and of system memory.
- **Process Information**
 - Most basic ones are a list of the currently loaded executable modules and the currently running threads

Important Debugger Features



- **Software and Hardware Breakpoints**
 - Software breakpoints are instructions added into the program's code by the debugger at runtime. These instructions make the processor pause program execution and transfer control to the debugger when they are reached during execution.
 - Hardware breakpoints are a special CPU feature that allow the processor to pause execution when a certain memory address is accessed, and transfer control to the debugger.

Example: OllyDbg



- OllyDbg includes a powerful disassembler
- Its code analyzer can identify loops, switch blocks, and other key code structures.
- It shows parameter names for all known functions and APIs, and supports searching for cross-references between code and data—in all possible directions.
- It supports a wide variety of views, including listing imports and exports in modules
- It also includes a built-in assembling and patching engine

Example: OllyDbg



OllyDbg - Defender.exe

File View Debug Plugins Options Window Help

Names in ntdll

Address	Name
7C92078F	RtlpGetLengthWithoutLastPath
7C92088F	RtlpDetermineDosPathNameType
7C920940	RtlpWin32NtRoot
7C920940	RtlpWin32NtRootSlash
7C920950	??_Cm_19MJCDBCEWY\$AA?2Y\$AA?2
7C92095F	RtlReadOutOfProcessMemoryStr
7C920A12	RtlStatMemoryStream12
7C920A7C	GUID_NULL
7C920A91	LdrAccessOutOfProcessResource
7C920AC0	RtlCreateActivationContext32
7C920C80	RtlValidateActivationContext
7C920E3A	RtlFinalReleaseOutOfProcessM
7C920ECE	RtlSetCurrentDirectory_0004
7C921129	LdrpCopyUnicodeString08
7C921190	RtlInitializeResource004
7C92124A	_wcsstr
7C92135A	RtlDestroyEnvironment04
7C921462	RtlDisplay04

CPU - main thread, module Defender

Address	Disassembly
00402E86	LEAVE
00402E87	RETN
00402E88	PUSH ECX
00402E89	MOV EAX,DWORD PTR FS:[30]
00402E8A	MOV DWORD PTR SS:[ESP],EAX
00402E8B	MOV EAX,DWORD PTR 3S:[ESP]
00402E8C	MOV EAX,DWORD PTR DS:[EAX+C]
00402E8D	MOV EAX,DWORD PTR DS:[EAX+C]
00402E8E	MOV EAX,DWORD PTR DS:[EAX]
00402E8F	MOV EAX,DWORD PTR DS:[EAX+18]
00402E90	POP ECX
00402E91	RETN
00402E92	MOV EAX,DWORD PTR DS:[406000]
00402E93	MOV ECX,EAX
00402E94	MOV EAX,DWORD PTR DS:[EAX]
00402E95	JMP SHORT Defender.00402ED9
00402E96	CMP EAX,1BF09AE
00402E97	JE SHORT Defender.00402EE4
00402E98	ADD ECX,8

Registers (FPU)

Register	Value
EAX	7FFDE000
ECX	0012FF00
EDX	7C90EB94 ntdll.RtlFastS
EBX	7FFDE000
ESP	0012FF08
EBP	0012FFC0
ESI	FFFFFFFF
EIP	7C910738 ntdll.7C910738
EIP	00402EAF Defender.00402EAF
C 0	ES 0020 32bit 0(FFFFFF)
P 1	CS 0010 32bit 0(FFFFFF)
A 0	SS 0020 32bit 0(FFFFFF)
Z 1	DS 0020 32bit 0(FFFFFF)
S 0	FS 0038 32bit 7FFDD000
T 0	GS 0000 NULL
D 0	
0 0	LastErr ERROR_SUCCESS
EFL	00000246 (NO,NO,E,DE,H)
ST0	empty -UNORM BDEC 0105

Stack SS:[0012FF08]-0012FF00

Address	32-bit long
00406000	00000000 00000000 00000000 00000000
00406010	00000000 00000000 00000000 00000000
00406020	00000000 00000000 00000000 00000000
00406030	00000000 00000000 00000000 00000000
00406040	00000000 00000000 00000000 00000000
00406050	00000000 00000000 00000000 00000000
00406060	00000000 00000000 00000000 00000000
00406070	00000000 00000000 00000000 00000000
00406080	00000000 00000000 00000000 00000000
00406090	00000000 00000000 00000000 00000000
004060A0	00000000 00000000 00000000 00000000
004060B0	00000000 00000000 00000000 00000000
004060C0	00000000 00000000 00000000 00000000
004060D0	00000000 00000000 00000000 00000000
004060E0	00000000 00000000 00000000 00000000
004060F0	00000000 00000000 00000000 00000000

Handles

Handle	Type	Refs	Access
00000000	Directory	108	00000003
00000014	Directory	76	000F000F
00000020	Event	3	001F0003
0000000C	File (dir)	2	00100020
00000004	KeyedEvent	74	000F0003
00000010	Port	3	001F0001
00000024	WindowStation	165	000F003F

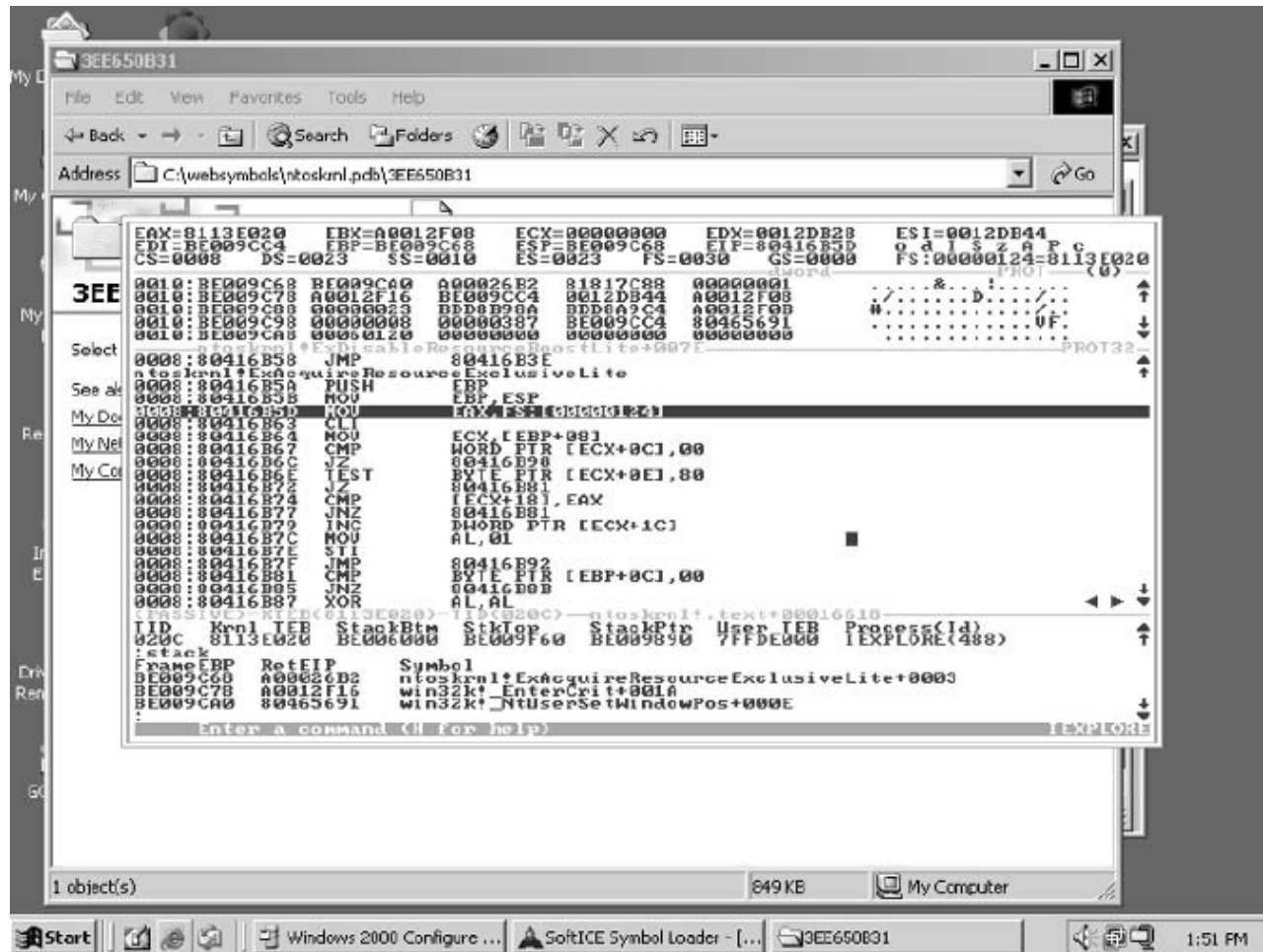
EDP-10

EDP-10	0012FF00
EDP-14	0040423E
EDP-18	7C910738
EDP-C	FFFFFFFF
EDP-8	0012FF00
EDP-4	0012FFB0
EDP -->	0012FFB0
EDP-4	7C910738
EDP-8	7C910738
EDP-C	FFFFFFFF
EDP-10	7FFDE000
EDP-14	854B038
EDP-18	0012FFC0
EDP-1C	88FAD0A8
EDP-20	FFFFFFFF
EDP-24	7C9099F0
EDP-28	7C916050

Paused

Typical OllyDbg Screen

Example: SoftICE



Decompilers



- Decompilers attempt to produce a high-level language source-code-like representation from a program binary.
- It is never possible to restore the original code in its exact form because the compilation process always removes some information from the program.

System Monitoring Tools



- System-monitoring tools is a general category of software tools that observe the various channels of I/O that exist between applications and the operating system.
- These are tools such as file access monitors that display every file operation (such as file creation, reading or writing to a file, and so on) made from every application on the system.
- This is done by hooking certain low-level components in the operating system and monitoring any relevant calls made from applications.

Example: Process Explorer



- Process Explorer is like a turbo-charged version of the built-in Windows Task Manager, and was actually designed to replace it.
- Process Explorer can show processes, DLLs loaded within their address spaces, handles to objects within each process, detailed information on open network connections, CPU and memory usage graphs, and the list just goes on and on.

Example: Process Explorer



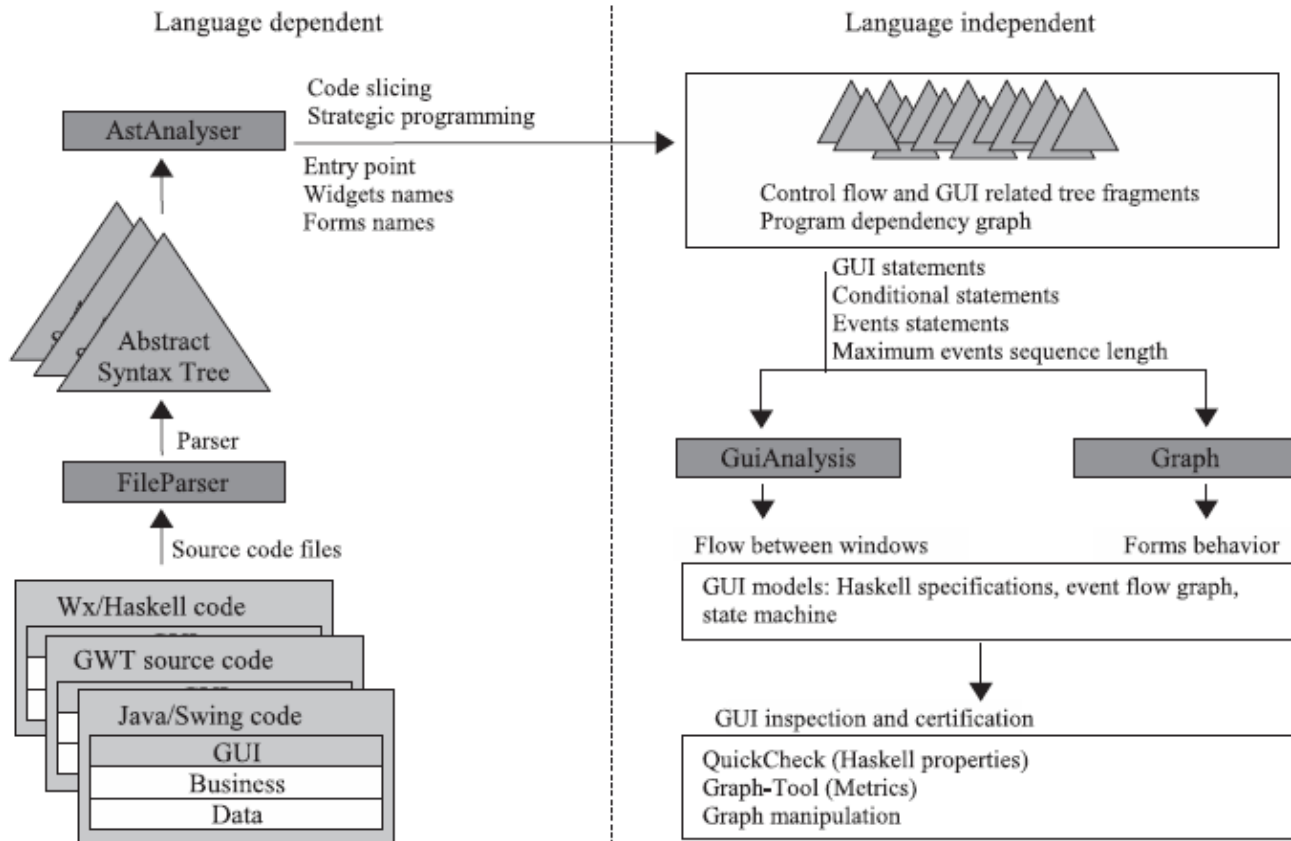
The screenshot displays the Process Explorer application from Sysinternals. The main window shows a list of processes with columns for Name, PID, CPU, Description, User Name, Priority, Handles, and Window Title. The 'System' folder is expanded, showing various system processes like smss.exe, csrss.exe, winlogon.exe, services.exe, lsass.exe, explorer.exe, and others.

Overlaid on the main window are two smaller windows:

- csrss.exe:684 Properties**: This window shows the 'Performance' tab, which includes a 'CPU' section and a 'Switch Data' section. The 'Switch Data' section lists various system threads and their addresses.
- Stack for thread 732**: This window shows the stack for a specific thread (ID: 732). It lists the thread ID, start time, state, kernel time, user time, and context switches. The stack itself is shown as a list of memory addresses and their corresponding values.

At the bottom of the Process Explorer window, the status bar indicates: CPU Usage: 100% | Commit Charge: 71.70% | Processes: 87 | Paused.

Other RE Tools: GUISURFER



Ref.: GUIsurfer: A Reverse Engineering Framework for User Interface Software
by Campos, Saraiva, and Silva



**FIRMWARE RE
FOR
PIC MICROCONTROLLER**

Microchip PICKit Debugger / Programmer



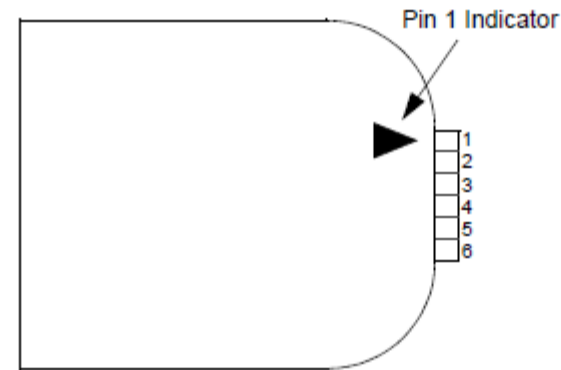
- The PICkit 3 debugger was developed for **programming and debugging embedded processors** with debug functions.
- The PICkit 3 features include:
 - Full-speed USB support using Windows standard drivers
 - Real-time execution
 - Processors running at maximum speeds
 - Built-in over-voltage/short circuit monitor
 - Low voltage to 5V (1.8-5V range)
 - Diagnostic LEDs (power, active, status)
 - **Read/write program and data memory of microcontroller**
 - Erasing of all memory types (EEPROM, ID, configuration and program) with verification
 - Peripheral freeze at breakpoint

PICKit Debugger



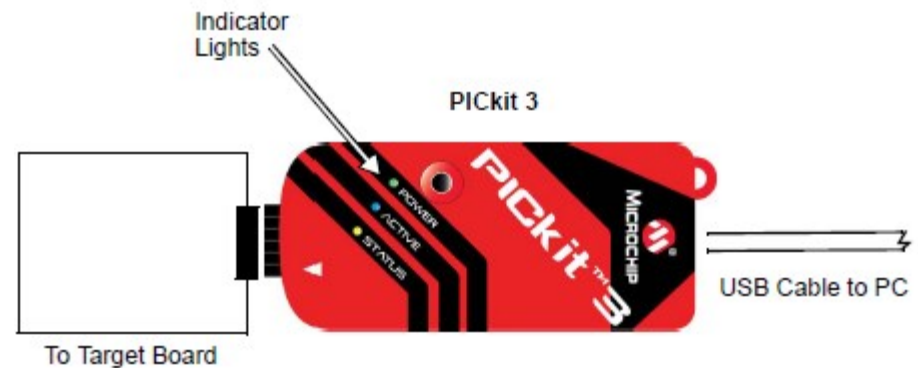
Legend:

- 1 – Lanyard Loop
- 2 – USB Port Connection
- 3 – Pin 1 Marker
- 4 – Programming Connector
- 5 – Indicator LEDs
- 6 – Push Button

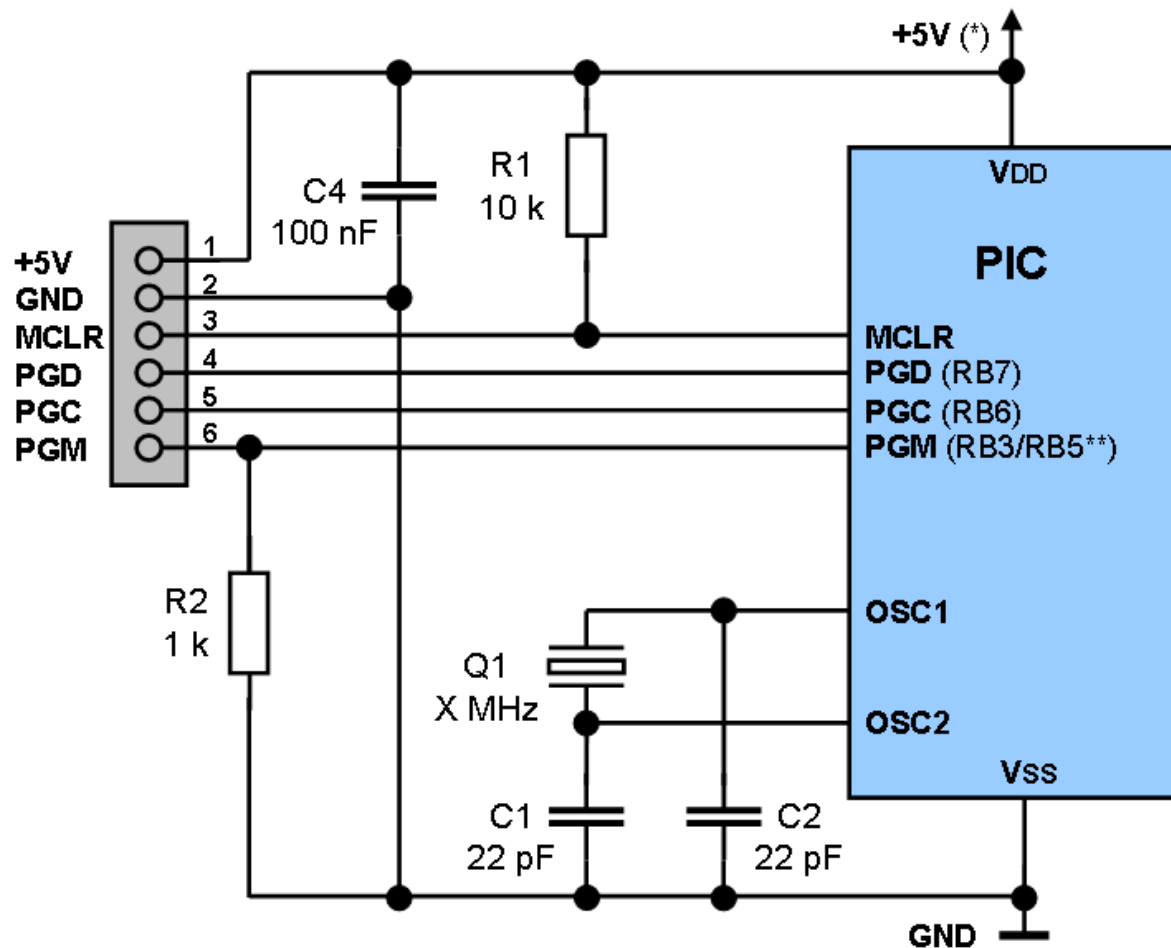


Pin Description*

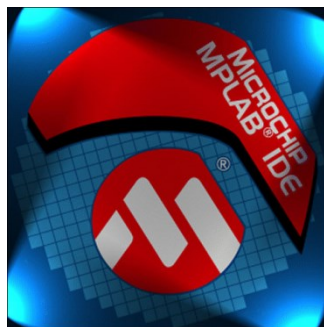
- 1 = $\overline{\text{MCLR}}/\text{VPP}$
- 2 = VDD Target
- 3 = VSS (ground)
- 4 = PGD (ICSPDAT)
- 5 = PGC (ICSPCLK)
- 6 = PGM (LVP)



PIC – Debugger Connection



MPLAB



MPLAB IDE

File Edit View Project Debugger Programmer Tools Configure Window Help

IntEEDATA.mcw

Source F
EED6
Main
Header F
Object F
Library F
Linker Sc
18F4
Other Fk

C:\V\Main.c

```

28 unsigned char Timeout;
29
30 //-----
31
32 void main ()
33 {
34     Timeout = 0;
35     INTCON = 0x20; //disable global and ena
36     INTCON2 = 0x84; //TMR0 high priority
37     RCONbits IPEN = 1; //enable priority levels
38     TMR0H = 0; //clear timer
39     TMR0L
40     T0CON
41     INTCON
42     TRISB =
43     TRISC =
44     EADR=
45
46 for (in
47
48 while
49 {
50 if (T
51 {
52     Timeout = 0; //clear timeout indi
53     EEWRITE(PORTC,EADR++);
54     Nop0;
55 }
56 }
57 }
58 }
59 }
60 }
61 }
62 }
63 }
64 }
65 }
66 }
67 }
68 }
69 }
70 }
71 }
72 }
73 }
74 }
75 }
76 }
77 }
78 }
79 }
80 }
81 }
82 }
83 }
84 }
85 }
86 }
87 }
88 }
89 }
90 }
91 }
92 }
93 }
94 }
95 }
96 }
97 }
98 }
99 }
100 }

```

Disassembly Listing

Address	OpCode	OpName	OpData	Comment
000162	52E6	MOVF	0xf6, F, ACCESS	59: movf
9EA6	BCF	0xfa6, 0x7		
94A6	BCF	0xfa6, 0x2		
9EF2	BCF	0xff2, 0x7		
0E55	MOVLW	0x55		
6EA7	MOVWF	0xfa7, ACC		
0EAA	MOVLW	0xaa		
6EA7	MOVWF	0xfa7, ACC		
82A6	BSF	0xfa6, 0x1		
8EF2	BSF	0xff2, 0x7		
0003	SLEEP			
84A6	BSF	0xfa6, 0x2		
0012	RETURN	0		

Special Function Registers

SFR Name	Hex	Binary
INDF1	--	-----
INDF2	--	-----
INTCON	A0	10100000
INTCON2	84	10000100
INTCON3	CO	11000000
IPR1	FF	11111111
IPR2	1F	00011111
LATA	00	00000000
LATB	00	00000000

Stopwatch

Synch Instruction Cycles 21511894 21511894

Zero Time (Secs) 4.302379 4.302379

Processor Frequency (MHz) 20.000000

Clear Simulation Time On Reset

Trace

Line	Ad	OpCode	OpName	OpData	Comment
23	00				
24	00				
25	00				
26	00				
27	00				
28	00				
29	00				
30	00				
31	00				
32	00E8	68BD	CLRF	Timeout, BANKED	
33	00EA	0E20	MOVLW	0x20	
34	00EC	6EF2	MOVWF	INTCON, ACCESS	
35	00EE	0EB4	MOVLW	0xB4	
36	00F0	6EF1	MOVWF	INTCON2, ACCESS	

Watch

Add SFR TOS Add Symbol PORTBbits

Address	Symbol Name	Value
008B	index	0x007B
0FC2	ADCON0	0x00
0F81	PORTB	0x00
0FFD	TOS	0x000126

Hardware Stack

TOS	Stack Level	Return Address	Location
	0	Empty	
	1	000044	_startup +
	2	000126	main + 0x40
	3	000198	.file
	4	000000	
	5	000000	
	6	000000	
	7	000000	
	8	000000	

MPLAB SIM PIC16F452 pc:0x198 W:0xaa N ov 2 dc c 0x850c

Machine Code vs. Assembly Language



<i>Memory address</i>	<i>Machine code</i>	<i>Assembly</i>	<i>Meaning</i>
000	3000	MOVLW 00	Load working register (W) with number 0
001	0066	TRIS 06	Store W in Port B direction code register
002	0186	CLRF 06	Clear Port B data register
003	0A86	INCF 06	Increment Port B data register
004	2803	GOTO 03	Jump back to address 0003 above

Conclusion



- S/W are used in many products and therefore S/W RE has gained much attention in the industry and research.
- S/W RE requires
 - In-depth knowledge in software construction (such as Assembly Language and O/S).
 - Great skills in puzzle solving and code breaking.
 - Knowledge and skills in identifying and using RE tools.