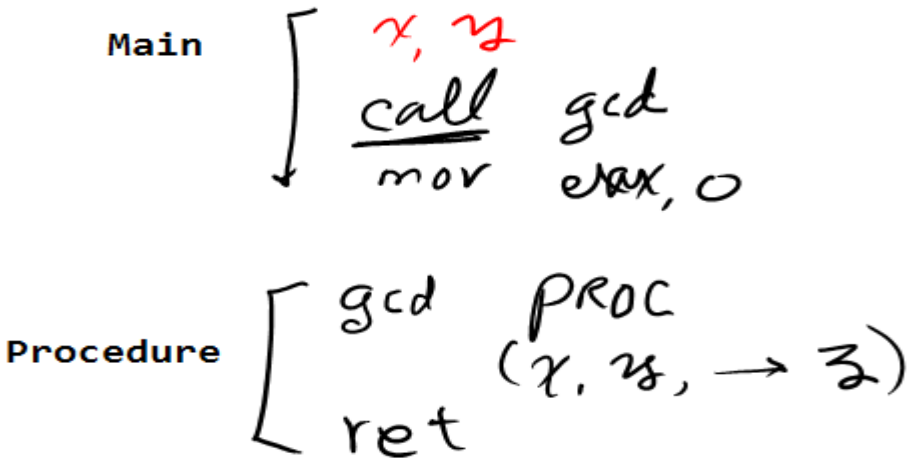
X86 Assembly Language Programming

(2) Procedures

Procedures with Value Parameters

- Main program **`call`**(s) a procedure
- Main Program transfers the parameter values
- Procedure receives (retrieves) them
- Procedure may do a task or it may return a value
  - value-returning procedure is sometimes called a *function*



Procedure Calling and Stack

- 3 concepts:
  - How to transfer control from a calling [main] program to a procedure and back
  - How to pass parameter values to a procedure and results back from the procedure
  - How to write procedure code that is independent of the calling program.
- Hardware stack is used to accomplish each of the above jobs.

80x86 Stack

- Hardware Stack
  - ESP holds the address of the "**first (lowest) byte above (or higher) "** of the **stack pointer**
  - Most access is indirect, through the stack point register ESP
    - Operating system initializes ESP to point to byte above stack pointer
    - As program executes, it points to the last item pushed on the stack
  - "Top" of stack is at the lowest address
  - Stack grows toward lower address

How `Call/Ret` Works

- **`call`**
  - The address of the instruction in the EIP register following the `call` ("the address of the next code line after the call instruction") is pushed on the stack
  - **so ESP has grown by 4, which means ESP address is lowered by 4**
  - Equivalent to `push EIP`
  - Then, the instruction pointer register EIP is loaded with the address of the first instruction in the procedure → jump to the procedure

How `Call/Ret` Works

- **`ret`**
  - The double-word (4 bytes) on the **top of the stack** is popped into the instruction pointer register EIP (**so ESP has increased by 4**
  - Equivalent to `pop EIP`
  - this is the address of the instruction right after the `call`, that instruction will be executed next [Return Address]
  - If the stack has been used for other values after the call, these must be removed before the `ret` instruction is executed

Alternative `Ret` Format

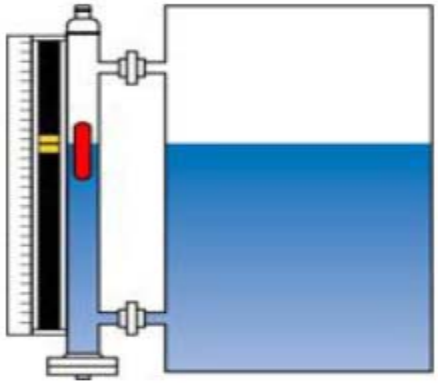
- **`ret` *`n`***
  - After the returned address is popped to EIP from the stack, *n* is added to ESP
  - [ESP] = [ESP + 8]
  - This is most often used to logically remove procedure parameters that have been pushed onto the stack
  - Used in Stdcall Protocol
- Protocol?
  - Transfer of control from calling program to procedure and back
  - Passing parameter values to procedure and results back from the procedure

Push Instruction

- Usual format: **`push`** `source`
  - –source can be memory, register or immediate
  - –Double-word or word pushed on the stack
- ESP decremented by size of operand
- Flags not changed
- **By Push, stack point goes lower in address (ESP)**

Push Instruction

	Stack size of 16		
	Stack Point = 0010		
	Stack filling starts from 000F (1 below the pointer)		
Stack		**Istruction**	**ESP**
			0010
		push 01020304	000C
		push 0A0B0C0D	**0008**
0000			
0001			
0002			
0003			
0004			
0005			
0006			
0007			
0008	0D	←	
0009	0C		
000A	0B		
000B	0A		
000C	04	←	
000D	03		
000E	02		
000F	01		
0010		←	



Stack push (filling) starts from 1 lower the ESP, and stays there

Stack pops from the ESP and moves 1 higher

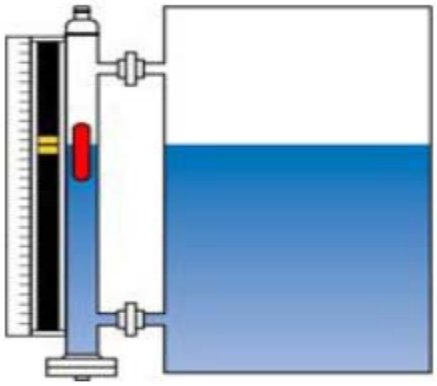
Push Example

- Pushd --- DWORD size operand

-240d → -F0h → 00000 F0
FFFFFF0F +1
FFFFFF10

Stack push (filling) starts from 1 lower the ESP, and stays there
Stack pops from the ESP and moves 1 higher

		push EAX	006001FC
		pushd -240	006001F8
006001F0			
006001F1			
006001F2			
006001F3			
006001F4			
006001F5			
006001F6			
006001F7			
006001F8	10		
006001F9	FF		
006001FA	FF		
006001FB	FF		
006001FC	A2		
006001FD	47		
006001FE	B5		
006001FF	83		
00600200			



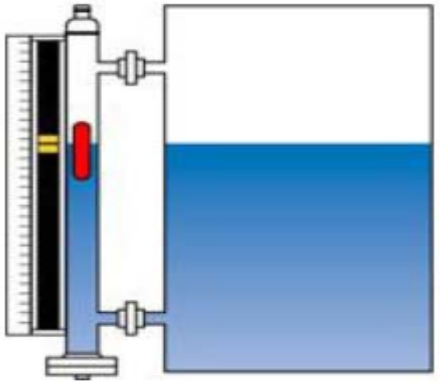
pop Instruction and Execution

- Usual format: `pop` *`destination`*
  - Double-word *destination* can be memory or register
- Operand stored in stack where ESP point is copied to destination
- ESP incremented by size of operand after the value is copied

pop Instruction and Execution

Stack push (filling) starts from 1 lower the ESP, and stays there
Stack pops from the ESP and moves 1 higher

Stack	Data	Istruction	ESP	EAX
			00600200	83B547A2
		push EAX	006001FC	
		pushd -240	006001F8	
		pop EAX	006001FC	FFFFFF10
006001F0				
006001F1				
006001F2				
006001F3				
006001F4				
006001F5				
006001F6				
006001F7				
006001F8	10			
006001F9	FF			
006001FA	FF			
006001FB	FF			
006001FC	A2			
006001FD	47			
006001FE	B5			
006001FF	83			
00600200				

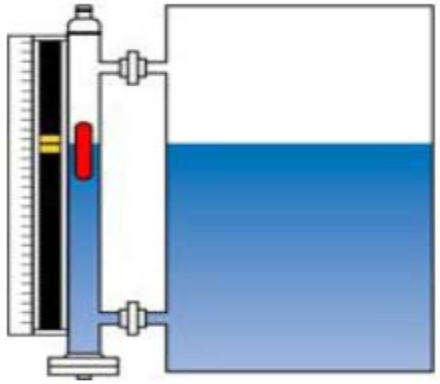


Pop Example [pop CX]

Stack push (filling) starts from 1 below the ESP, and stays there

Stack pops from the ESP and moves 1 above

Stack	Data	Istruction	ESP	ECX
			00600200	83B547A2
		push ECX	006001FC	
		pushd -240	006001F8	
		pop ECX	006001FC	FFFFFF10
		pop CX	006001FE	FFFF47A2
006001F0				
006001F1				
006001F2				
006001F3				
006001F4				
006001F5				
006001F6				
006001F7				
006001F8	10	←		
006001F9	FF			
006001FA	FF			
006001FB	FF			
006001FC	A2	← ←		
006001FD	47			
006001FE	B5	←		
006001FF	83			
00600200		←		



Push Exercise/Solution

- Before
  - [ESP]=06 00 10 00
  - [ECX]=01 A2 5B 74
- After **`push ECX`**
- After **`pushd 10`**
  - [STACK]= ?

Stack push (filling) starts from 1 lower the ESP, and stays there
Stack pops from the ESP and moves 1 higher

Stack filling starts from 1 below the ESP

Stack	Data	Istruction	ESP	ECX
			06001000	01A25B74
		push ECX	06000FFC	
		pushd 10	06000FF8	
06000FF7				
06000FF8	0A			
06000FF9	00			
06000FFA	00			
06000FFB	00			
06000FFC	74			
06000FFD	5B			
06000FFE	A2			
06000FFF	01			
06001000				

Push – Practice

- Before:
  - [ESP]=02 00 0B 7C
  - [EBX]=12 34 56 78
- Stack Diagram and [ESP]
  - After **`pushd 20`**
  - After **`push EBX`**

Stack push (filling) starts from 1 lower the ESP, and stays there
Stack pops from the ESP and moves 1 higher

Stack filling starts from 1 below the ESP

Stack	Data	Istruction	ESP	EBX
			02000B7C	12345678
		pushd 20		
		push EBX		
02000B73				
02000B74				
02000B75				
02000B76				
02000B77				
02000B78				
02000B79				
02000B7A				
02000B7B				
02000B7C				
02000B7D				

Push – Practice (SOLUTION)

- Before:
  - [ESP]=02 00 0B 7C
  - [EBX]=12 34 56 78
- Stack Diagram and [ESP]
  - After **`pushd 20`**
  - After **`push EBX`**

Stack push (filling) starts from 1 lower the ESP, and stays there
Stack pops from the ESP and moves 1 higher

Stack filling starts from 1 below the ESP

Stack	Data	Istruction	ESP	EBX
			02000B7C	12345678
		pushd 20	02000B78	
		push EBX	02000B74	
02000B73				
02000B74	78			
02000B75	56			
02000B76	34			
02000B77	12			
02000B78	14			
02000B79	00			
02000B7A	00			
02000B7B	00			
02000B7C				
02000B7D				

Push-Pop Practice

- Before:
  - [ESP]=00 10 F8 3A
  - [EAX]=12 34 56 78
- Stack Diagram, [EAX
  - After
    - **Push EAX**
    - **Pushd 30**
    - **Pop EAX**
    - **Pop EBX**

Stack push (filling) starts from 1 lower the ESP, and stays there
Stack pops from the ESP and moves 1 higher

Stack filling starts from 1 below the ESP

Stack	Data	Istruction	ESP	EAX	EBX
			0010F83A	12345678	????????
		push EAX			
		pushd 30			
		pop EAX			
		pop EBX			
0010F82F					
0010F830					
0010F831					
0010F832					
0010F833					
0010F834					
0010F835					
0010F836					
0010F837					
0010F838					
0010F839					
0010F83A					

Push-Pop Practice SOLUTION

- Before:
  - [ESP]=00 10 F8 3A
  - [EAX]=12 34 56 78
- Stack Diagram, [E
  - After
    - **Push EAX**
    - **Pushd 30**
    - **Pop EAX**
    - **Pop EBX**

Stack filling starts from 1 below the ESP

Stack	Data	Istruction	ESP	EAX	EBX
			0010F83A	12345678	????????
		push EAX	0010F836		
		pushd 30	0010F832		
		pop EAX	0010F836	0000001E	
		pop EBX	0010F83A		12345678
0010F82F					
0010F830					
0010F831					
0010F832	1E				
0010F833	00				
0010F834	00				
0010F835	00				
0010F836	78				
0010F837	56				
0010F838	34				
0010F839	12				
0010F83A					

Practice Example (with original ESP=10001FF0)

EECE416 Microcomputer
Individual Class Activity for POP and PUSH

Date:
Name:

STACK	Conetnts
10001FDD	
10001FDE	
10001FDF	
10001FE0	
10001FE1	
10001FE2	
10001FE3	
10001FE4	
10001FE5	
10001FE6	
10001FE7	
10001FE8	
10001FE9	
10001FEA	
10001FEB	
10001FEC	
10001FED	
10001FEE	
10001FEF	
10001FF0	
10001FF1	
10001FF2	

Original [ESP] = 10001FF0

Instruction		ESP	EAX	EBX	ECX	EDX
PUSH	11223344h					
PUSH	AABBCCDDh					
PUSH	FFEEFFEEh					
PUSH	01020304h					
POP	EAX					
POP	EBX					
PUSH	66778899h					
POP	ECX					
POP	EDX					

Push/Pop example code: Proc1.asm

Proc1.asm

```
TITLE Procedure Example        (Proc1.asm)


INCLUDE Irvine32.inc
.stack    4096

.data

.code
main PROC
      mov  EAX,0
      mov  EBX,0
      mov  ECX,0
      mov  EAX, 83B547A2h
      push EAX
      pushd -240     ;double word
      pushw 5; WORD Size
      pop  EAX
      pop  AX
      pop  EBX
      exit
main ENDP
```

100 %

- Pushw --- WORD size operand

Memory 1

Address: 0x0018FF80

```
0x0018FF80   00 00 00 00 00 00 00 00   ........
0x0018FF88   00 00 00 00 8a 33 56 76   ....Š3Vv
0x0018FF90   00 e0 fd 7e d4 ff 18 00   .àý~Ôÿ..
0x0018FF98   72 9f 44 77 00 e0 fd 7e   rŸDw.àý~
0x0018FFA0   38 f3 32 76 00 00 00 00   8ó2v....
0x0018FFA8   00 00 00 00 00 e0 fd 7e   .....àý~
0x0018FFB0   00 00 00 00 00 00 00 00   ........
0x0018FFB8   00 00 00 00 a0 ff 18 00   ....  ÿ..
```

Registers

```
EAX = 76563378 EBX = 7EFDE000
ECX = 00000000 EDX = 00401005
ESI = 00000000 EDI = 00000000
EIP = 00401010 ESP = 0018FF8C
EBP = 0018FF94 EFL = 00000246
```

Push/Pop example code: Proc1.asm

- push EAX

```
        mov   ECX,0
        mov   EAX, 83B547A2h
        push  EAX
        pushd -240      ;doubl
        pushw 5; WORD Size
        pop   EAX
        pop   AX
        pop   EBX
        exit
main ENDP
```

100 %

	A	B	C	D	E	F
1		Stack push (filling) starts from 1 lower the ESP, and stays there				
2		Stack pops from the ESP and moves 1 higher				
3	**Stack**	**Data**	**Istruction**	**ESP**	**EAX**	**EBX**
4				0018FF8c	83B547A2	????????
5	0018FF80		push EAX	0018FF88		
6	0018FF81		pushd -240	0018FF84		
7	0018FF82		pushw 5	0018FF82		
8	0018FF83		pop EAX	0018FF86	FF100005	
9	0018FF84		pop AX	0018FF88	FF10FFFF	
10	0018FF85		pop EBX	0010FF8C		83B547A2
11	0018FF86					
12	0018FF87					
13	0010FF88	A2				
14	0010FF89	47				
15	0010FF8A	B5				
16	0010FF8B	83				
17	0010FF8C					
18	0010FF8D					

Memory 1

Address: 0x0018FF80

```
0x0018FF80  00 00 00 00 00 00 00 00  ........
0x0018FF88  a2 47 b5 83 8a 33 56 76  ¢GµƒŠ3Vv
0x0018FF90  00 e0 fd 7e d4 ff 18 00  .àý~Ôÿ..
0x0018FF98  72 9f 44 77 00 e0 fd 7e  rŸDw.àý~
```

Registers

```
EAX = 83B547A2 EBX = 00000000
ECX = 00000000 EDX = 00401005
ESI = 00000000 EDI = 00000000
EIP = 00401025 ESP = 0018FF88
EBP = 0018FF94 EFL = 00000246
```

Push/Pop example code: Proc1.asm

- pushd -240

```
        push EAX
        pushd -240      ;double
        pushw 5; WORD Size
        pop   EAX
        pop   AX
        pop   EBX
        exit
main ENDP
```

100 %

	A	B	C	D	E	F
1		Stack push (filling) starts from 1 lower the ESP, and stays there				
2		Stack pops from the ESP and moves 1 higher				
3	**Stack**	**Data**	**Istruction**	**ESP**	**EAX**	**EBX**
4				0018FF8c	83B547A2	????????
5	0018FF80		push EAX	0018FF88		
6	0018FF81		pushd -240	0018FF84		
7	0018FF82		pushw 5	0018FF82		
8	0018FF83		pop EAX	0018FF86	FF100005	
9	0018FF84	10	pop AX	0018FF88	FF10FFFF	
10	0018FF85	FF	pop EBX	0010FF8C		83B547A2
11	0018FF86	FF				
12	0018FF87	FF				
13	0010FF88	A2				
14	0010FF89	47				
15	0010FF8A	B5				
16	0010FF8B	83				
17	0010FF8C					
18	0010FF8D					

Memory 1

Address: 0x0018FF80

```
0x0018FF80   00 00 00 00 10 ff ff ff   .....ÿÿÿ
0x0018FF88   a2 47 b5 83 8a 33 56 76   ¢GµƒŠ3Vv
0x0018FF90   00 e0 fd 7e d4 ff 18 00   .àý~Ôÿ..
0x0018FF98   72 9f 44 77 00 e0 fd 7e   rŸDw.àý~
0x0018FFA0   38 f3 32 76 00 00 00 00   8ó2v....
```

Registers

```
EAX = 83B547A2 EBX = 00000000
ECX = 00000000 EDX = 00401005
ESI = 00000000 EDI = 00000000
EIP = 0040102A ESP = 0018FF84
EBP = 0018FF94 EFL = 00000246
```

Push/Pop example code: Proc1.asm

- Pushw --- WORD size operand
  - pushw 5

```
        push EAX
        pushd -240      ;doubl
        pushw 5; WORD Size
        pop  EAX
        pop  AX
        pop  EBX
        exit
main ENDP
```

100 %

	A	B	C	D	E	F
1		Stack push (filling) starts from 1 lower the ESP, and stays there				
2		Stack pops from the ESP and moves 1 higher				
3	Stack	Data	Istruction	ESP	EAX	EBX
4				0018FF8c	83B547A2	????????
5	0018FF80		push EAX	0018FF88		
6	0018FF81		pushd -240	0018FF84		
7	0018FF82	05	pushw 5	0018FF82		
8	0018FF83	00	pop EAX			
9	0018FF84	10	pop AX			
10	0018FF85	FF	pop EBX			
11	0018FF86	FF				
12	0018FF87	FF				
13	0018FF88	A2				
14	0018FF89	47				
15	0018FF8A	B5				
16	0018FF8B	83				
17	0018FF8C					
18	0018FF8D					

Memory 1

Address: 0x0018FF80

```
0x0018FF80  00 00 05 00 10 ff ff ff  .....ÿÿÿ
0x0018FF88  a2 47 b5 83 8a 33 56 76  ¢GµƒŠ3Vv
0x0018FF90  00 e0 fd 7e d4 ff 18 00  .àý~Ôÿ..
0x0018FF98  72 9f 44 77 00 e0 fd 7e  rŸDw.àý~
0x0018FFA0  38 f3 32 76 00 00 00 00  8ó2v....
```

Registers

```
EAX = 83B547A2 EBX = 00000000
ECX = 00000000 EDX = 00401005
ESI = 00000000 EDI = 00000000
EIP = 0040102D ESP = 0018FF82
EBP = 0018FF94 EFL = 00000246
```

Push/Pop example code: Proc1.asm

- pop EAX

```
        push EAX
        pushd -240      ;c
        pushw 5; WORD Siz
        pop   EAX
        pop   AX
        pop   EBX
        exit
main ENDP
```

100 %

	A	B	C	D	E	F
1		Stack push (filling) starts from 1 lower the ESP, and stays there				
2		Stack pops from the ESP and moves 1 higher				
3	Stack	Data	Istruction	ESP	EAX	EBX
4				0018FF8c	83B547A2	????????
5	0018FF80		push EAX	0018FF88		
6	0018FF81		pushd -240	0018FF84		
7	0018FF82	05	pushw 5	0018FF82		
8	0018FF83	00	pop EAX	0018FF86	FF100005	
9	0018FF84	10	pop AX			
10	0018FF85	FF	pop EBX			
11	0018FF86	FF				
12	0018FF87	FF				
13	0018FF88	A2				
14	0018FF89	47				
15	0018FF8A	B5				
16	0018FF8B	83				
17	0018FF8C					
18	0018FF8D					

Memory 1

Address: 0x0018FF80

```
0x0018FF80  00 00 05 00 10 ff ff ff   .....ÿÿÿ
0x0018FF88  a2 47 b5 83 8a 33 56 76   ¢GµƒŠ3Vv
0x0018FF90  00 e0 fd 7e d4 ff 18 00   .àý~Ôÿ..
0x0018FF98  72 9f 44 77 00 e0 fd 7e   rŸDw.àý~
0x0018FFA0  38 f3 32 76 00 00 00 00   8ó2v....
```

Registers

```
EAX = FF100005  EBX = 00000000
ECX = 00000000  EDX = 00401005
ESI = 00000000  EDI = 00000000
EIP = 0040102E  ESP = 0018FF86
EBP = 0018FF94  EFL = 00000246
```

Push/Pop example code: Proc1.asm

- pop AX

```
        push EAX
        pushd -240      ;
        pushw 5; WORD Si
        pop   EAX
        pop   AX
        pop   EBX
        exit
    main ENDP
```

100 %

	A	B	C	D	E	F
1		Stack push (filling) starts from 1 lower the ESP, and stays there				
2		Stack pops from the ESP and moves 1 higher				
3	Stack	Data	Istruction	ESP	EAX	EBX
4				0018FF8c	83B547A2	????????
5	0018FF80		push EAX	0018FF88		
6	0018FF81		pushd -240	0018FF84		
7	0018FF82	05	pushw 5	0018FF82		
8	0018FF83	00	pop EAX	0018FF86	FF100005	
9	0018FF84	10	pop AX	0018FF88	FF10FFFF	
10	0018FF85	FF	pop EBX			
11	0018FF86	FF				
12	0018FF87	FF				
13	0018FF88	A2				
14	0018FF89	47				
15	0018FF8A	B5				
16	0018FF8B	83				
17	0018FF8C					
18	0018FF8D					

Memory 1

Address: 0x0018FF80

```
0x0018FF80  00 00 05 00 10 ff ff ff  .....ÿÿÿ
0x0018FF88  a2 47 b5 83 8a 33 56 76  ¢GµƒŠ3Vv
0x0018FF90  00 e0 fd 7e d4 ff 18 00  .àý~Ôÿ..
0x0018FF98  72 9f 44 77 00 e0 fd 7e  rŸDw.àý~
0x0018FFA0  38 f3 32 76 00 00 00 00  8ó2v....
```

Registers

```
EAX = FF10FFFF  EBX = 00000000
ECX = 00000000  EDX = 00401005
ESI = 00000000  EDI = 00000000
EIP = 00401030  ESP = 0018FF88
EBP = 0018FF94  EFL = 00000246
```

Push/Pop example code: Proc1.asm

- pop EBX

```
        mov   EAX, 83B547A2h
        push  EAX
        pushd -240      ;doub
        pushw 5; WORD Size
        pop   EAX
        pop   AX
        pop   EBX
        exit
main ENDP
```

	A	B	C	D	E	F
1		Stack push (filling) starts from 1 lower the ESP, and stays there				
2		Stack pops from the ESP and moves 1 higher				
3	Stack	Data	Istruction	ESP	EAX	EBX
4				0018FF8c	83B547A2	????????
5	0018FF80		push EAX	0018FF88		
6	0018FF81		pushd -240	0018FF84		
7	0018FF82	05	pushw 5	0018FF82		
8	0018FF83	00	pop EAX	0018FF86	FF100005	
9	0018FF84	10	pop AX	0018FF88	FF10FFFF	
10	0018FF85	FF	pop EBX	0010FF8C		83B547A2
11	0018FF86	FF				
12	0018FF87	FF				
13	0018FF88	A2				
14	0018FF89	47				
15	0018FF8A	B5				
16	0018FF8B	83				
17	0018FF8C					
18	0018FF8D					

```
mory 1
ldress: 0x0018FF80
0018FF80  00 00 05 00 10 ff ff ff   .....ÿÿÿ
0018FF88  a2 47 b5 83 8a 33 56 76   ¢GµƒŠ3Vv
0018FF90  00 e0 fd 7e d4 ff 18 00   .àý~Ôÿ..
0018FF98  72 9f 44 77 00 e0 fd 7e   rŸDw.àý~
```

```
Registers
EAX = FF10FFFF EBX = 83B547A2
ECX = 00000000 EDX = 00401005
ESI = 00000000 EDI = 00000000
EIP = 00401031 ESP = 0018FF8C
EBP = 0018FF94 EFL = 00000246
```