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

43

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 s	size of 16		
	Stack F	Point = 00 ⁻	10	
	Stack f	illing starts	from 000F (1 below the pointer)
Stack		Istru	ction	ESP
				0010
		push	01020304	000C
		push	ØAØBØCØD	0008
0000				
0001				
0002				
0003				
0004				
0005				
0006				
0007				
0008	ØD	\leftarrow		
0009	0C			
000A	ØB		Stack pus	h (filling) starts fron
000B	ØA		-	
000C	04	←	Stack pop	s from the ESP and
000D	03			
000E	02			
000F	01			
DOLD		←		

Push Example

- 240 d -> Foh > 0000 00 For

FFFFFF

(FF-FF-FF

Pushd --- DWORD size operand •

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	
6001F0				
6001F1				
6001F2				
6001F3				ā
6001F4				1
6001F5				
6001F6				
6001F7				
6001F8	10			
6001F9	FF			
6001FA	FF			1
6001FB	FF			
6001FC	A2			
6001FD	47			
6001FE	B5			
6001FF	83			
600200				

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

	Data	rom the ESP and n		FAY
Stack	Data	Istruction	ESP	EAX
			00600200	83B547A2
		push EAX	006001FC	
		pushd -240	006001F8	
		pop EAX	006001FC	FFFFFF10
06001F0				
06001F0				
06001F1				
06001F3				
06001F4				
06001F5				
06001F6				
06001F7				
06001F8	10			
066001F9	FF			
066001FA	FF			
006001FB	FF			
006001FC	A2			
006001FD	47			
006001FE	B5			
06001FF	83			

аск рор	s from	the ESP and mov	/es 1 above	, and stays t	
Stack	Data	Istruction	ESP	ECX	
			00600200	83B547A2	
		push ECX	006001FC		
		pushd -240	006001F8		
		pop ECX	006001FC	FFFFF10	
		pop CX	006001FE	FFFF47A2	
006001F0					
006001F1					
06001F2					A A
06001F3					
066001F4					
006001F5					
006001F6					
006001F7					111
006001F8	10	←──			
006001F9	FF				
006001FA	FF				
006001FB	FF				
006001FC	A2	← ←			
006001FD	47				
06001FE	B5	←			
006001FF	83				
00600200		L			

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

н	D		U	L
	Stack f			
Stack	Data	Istruction	ESP	ECX
			06001000	01A25B74
		push ECX	06000FFC	
		pushd 10	06000FF8	
06000FF7				
06000FF8	ØA			
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

А	D		D	L				
	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				50				
				56				

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 f	illing starts from 1 I	below the ESP	
Stack	Data	Istruction	ESP	EBX
			02000B7C	12345678
		pushd 20	02000B78	
		push EBX	02000B74	
02000B73				
02000B74	78			
02000B75	56			
02000B76	34			
02000B77				
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, [EA>
 - After
 - Push EAX
 - Pushd 30
 - Pop EAX
 - Pop EBX

А	0			L	
	Stack f				
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					58

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

		~	0	U		L		
٦, 🛛	[E		Stack fill	ing starts from 1 belo	w the ESP			
, ,	• :	Stack	Data	Istruction	ESP	EAX	EBX	
	1				0010F83A	12345678	????????	
	÷			push EAX	0010F836			
/X	÷.,			pushd 30	0010F832			
30				pop EAX	0010F836	0000001E		
50				pop EBX	0010F83A		12345678	
۲.		0010F82F						
-		0010F830						
Z (C	0010F831						
	1	0010F832	1E					
	2	0010F833	00					
	3	0010F834	00					
	4	0010F835	00					
	5	0010F836	78					
	6	0010F837	56					
	7	0010F838	34					
	В	0010F839	12					
	Э	0010F83A						
	D							

Practice Example (with original ESP=10001FF0)

EECE416 Microcomputer

STACK

10001FDD

Individual Class Activity for POP and PUSH

Date: Name:

Original [ESP] = 10001FF0

		<u> </u>		1		1	1		
	Conetnts		Instruction E		ESP	EAX	EBX	ECX	EDX
			PUSH	11223344h					
			PUSH	AABBCCDDh					
			PUSH	FFEEFFEEh					
			PUSH	01020304h					
			POP	EAX					
			POP	EBX					
			PUSH	66778899h					
			POP	ECX					
			POP	EDX					
-									

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

Push/Pop example code: Proc1.asm

Proc1.asm ×									
TITLE	Procedui	re Exa	mple	(P	roc1.asm)				
INCLUD	E Irvine	e32.in	с						
.stack	4090	5							
.data									
.code							• Pus	shw WORD s	ize operand
main P	ROC								size operand
🔷 me	ov EAX	,0							
me	ov EBX	,0							
me	ov ECX								
		, 83B5	47A2h						
-	ush EAX								
	ushd -24		-	le word					
	ushw 5;		Size						
	op EAX								
	op AX								
	op EBX								
	xit								
main Fl	UDP								
Memory 1						-			
	5500					Registers	76562270		2
Address: 0x0018								EBX = 7EFDE00	
0x0018FF80		5.5 3.5	00 00					EDX = 0040100	
0x0018FF88			8a 33		Š3Vv			EDI = 00000000000000000000000000000000000	
0x0018FF90				18 00		X.10-10-5		ESP = 0018FF8	
0x0018FF98					rŸDw.àý~	EBb :	= 0018FF94	EFL = 0000024	0
0x0018FFA0			00 00	18 A. 19	8ó2v				60
0x0018FFA8			00 e0		àý~				62
0×0018FFB0			00 00						
0x0018FFB8	00 00	00 00	a0 ff	18 00	Ÿ				

Push/Pop example code: Proc1 asm

							1	4	B			C		D		E	F
						1			Stack p	oush (filli	ng) star	ts fro	om 1 lower th	ne ESF	, and	stays there
		_				2			Stack p	ops f	fron	n the ES	SP ar	nd moves 1 ł	nigher		
• r	ous	h	F	ΔΧ		3	Stac	k	Data	Ist	ruc	tion		ESP	EAX		EBX
	JUD				•	4								0018FF8c	83B5	47A2	???????
						5	0018	FF80		pusl	h E	AX		0018FF88			
			_			6	0018	FF81		pusl	hd	-240		0018FF84			
		ECX,				7	0018	FF82		pus	hw	5		0018FF82			
		-	83	B547	7A2h	8	0018	FF83		рор	EA	X		0018FF86	FF10	0005	
	push	EAX				9	0018	FF84		рор	AX	C		0018FF88	FF10	FFFF	
⇒	pushd	-24	10	;	; doub	1 10	0018	FF85		pop	EB	X		0010FF8C			83B547A2
	pushw	5;	WOR	D Si	ize	11	0018	FF86									
	рор	EAX				12	0018	FF87									
	pop /	AX				13	0010	FF88	A2								
	pop	EBX				14	0010	FF89	47								
	exit						0010		-								
main	ENDP						0010										
100 % 👻 🖣							0010										
Memory 1						18	0010	FF8D									
	00105500										- "	registers					
Address: 0x											4.			B547A2 EB)			
0x0018FF	80 00	00	00	00 e	90 00	00	00							000000 ED)			
0x0018FF	88 <mark>a2</mark>	47	b5	83 8	3a 33	56	76	¢G⊧	ı f Š3Vv			ESI =	000	000000 EDI	[= 0	00000	000
0x0018FF	90 00	e0	fd	7e d	14 ff	18	00	.àý	⁄∼Ôÿ			EIP =	004	401025 ESP	? = 0	018FF	88
0x0018FF	98 72	9f	44	77 0	00 e0	fd	7e	rŸD)w.àý∼			EBP =	00:	18FF94 EFI	_ = 0	00002	46
0004 0	AO 70	دع	22	76 0		00	00	011	··· ·								

Push/Pop example code: Proc1.asm

			A	В	С	D	E	F
		1		Stack p	ush (filling) starts fro	m 1 lower th	e ESP, and s	stays there
	pushd -240	2		Stack p	ops from the ESP an	d moves 1 h	igher	
	pushu = 240	з	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	рор АХ	0018FF88	FF10FFFF	
1 A A	push EAX		0018FF85	FF	pop EBX	0010FF8C		83B547A2
	pushd -240 ;double	11	0018FF86	FF				
⇒	pushw 5; WORD Size	12	0018FF87	FF				
	pop EAX	13	0010FF88	A2				
	pop AX	14	0010FF89	47				
	pop EBX	15	0010FF8A	B5				
	exit	16	0010FF8B	83				
main	ENDP	17	0010FF8C					
100 % 👻 🖣	1 10171	18	0010FF8D					

Memory 1											_ F	Registers				
Address:	0x0018F	F80										EAX =	83B547A2	EBX	= 00000	000
0x0018	FF80	00	00	00	00	10	ff	ff	ff	ÿÿÿ	П.	ECX =	00000000	EDX	= 00401	005
0x0018	FF88	a2	47	b5	83	8a	33	56	76	¢GµfŠ3V∨		ESI =	00000000	EDI	= 00000	000
x0018	FF90	00	e0	fd	7e	d4	ff	18	00	.àý∼Ôÿ		EIP =	0040102A	ESP	= 0018F	F84
x0018	FF98	72	9f	44	77	00	e0	fd	7e	rŸDw.àý~		EBP =	0018FF94	EFL	= 00000	246
0x0018	FFAØ	38	f3	32	76	00	00	00	00	8ó2v						

-				A	В	C	D	E	F
Pushw	WORD siz	e operand	1		Stack p	ush (filling) starts fr	om 1 lower th	e ESP, and s	stays the
		_	2		Stack p	ops from the ESP a	nd moves 1 hi	gher	
• pu	Shw	5	3	Stack	Data	Istruction	ESP	EAX	EBX
т ри			4				0018FF8c	83B547A2	????????
			5	0018FF80		push EAX	0018FF88		
			6	0018FF81		pushd -240	0018FF84		
			7	0018FF82	05	pushw 5	0018FF82		
			8	0018FF83	00	pop EAX			
pu	sh EAX		9	0018FF84	10	pop AX			
pus	shd -240	;doub]	10	0018FF85	FF	pop EBX			
		ORD Size	11	0018FF86	FF				
po			12	0018FF87	FF				
			13	0018FF88	A2				
pop			14	0018FF89	47				
pol			15	0018FF8A	B5				
ex:	it		16	0018FF8B	83				
main FNI)P		17	0018FF8C					
00 % 👻 🔹			18	0018FF8D					

viemory 1		Registers
Address: 0x00	18FF80	EAX = 83B547A2 EBX = 00000000
0x0018FF8	00 00 <mark>05</mark> 00 10 ff ff ffÿÿÿ	ECX = 00000000 EDX = 00401005
0x0018FF8	3 a2 47 b5 83 8a 33 56 76 ¢GµƒŠ3Vv	ESI = 00000000 EDI = 00000000
0x0018FF9	00 e0 fd 7e d4 ff 18 00 .àý~Ôÿ	EIP = 0040102D ESP = 0018FF82
0x0018FF9	3 72 9f 44 77 00 e0 fd 7e rŸDw.àý~	EBP = 0018FF94 EFL = 00000246
3x0018FFA	38 f3 32 76 00 00 00 00 8ó2v	

Push/Po		exa	mp	e co	de:	Proc1	l.asn)
		А	В	C		D	E	F
	1		Stack p	ush (filling)	starts fro	m 1 lower the	e ESP, and s	stays there
	2		Stack p	ops from the	e ESP and	d moves 1 hig	gher	
• pop EAX	3	Stack	Data	Istruction	า	ESP	EAX	EBX
	4					0018FF8c	83B547A2	<u> ????????</u>
	5	0018FF80		push EAX		0018FF88		
	6	0018FF81		pushd -240	9	0018FF84		
	7	0018FF82	05	pushw 5		0018FF82		
	8	0018FF83	00	pop EAX		0018FF86	FF100005	
push EAX	9	0018FF84	10	pop AX				
pushd -240 ;	10	0018FF85	FF	pop EBX				
pushw 5; WORD Si			FF					
pop EAX		0018FF87	FF					
pop AX		0018FF88	A2					
pop EBX		0018FF89	47					
exit		0018FF8A	B5					
main ENDP		0018FF8B	83					
100 % - 4	17							
	18	0018FF8D						
Memory 1					Registers			
Address: 0x0018FF80					EAX =	FF100005	EBX = 00	000000
0x0018FF80 00 00 05 00 1	ð f	f ff ff		ӱӱӱ	ECX =	00000000	EDX = 00	401005
0x0018FF88 a2 47 b5 83 8	а З	3 56 76	¢GµfŠ	3Vv	ESI =	00000000	EDI = 00	000000
0x0018FF90 00 e0 fd 7e d	4 f	f 18 00	.àý∼Ô		EIP =	0040102E	ESP = 00	18FF86
0x0018FF98 72 9f 44 77 0) e	0 fd 7e	rŸĎw.	-	EBP =	0018FF94	EFL = 00	000246
				-,				

	Ρ	ush/F	or	exa	mp	e code:	Proc1	asm	
				А	В	С	D	E	F
			1		Stack p	ush (filling) starts fi	rom 1 lower th	e ESP, and s	stays there
			2		Stack p	ops from the ESP a	nd moves 1 hi	gher	
• n	on	AX	3	Stack	Data	Istruction	ESP	EAX	EBX
P	ΥP		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	
	push		10	0018FF85	FF	pop EBX			
	pusho	-240	; 11	0018FF86	FF				
	pushv	5; WORD	Si 12	0018FF87	FF				
	рор	EAX	13	0018FF88	A2				
	рор	AX	14	0018FF89	47				
→	рор	EBX		0018FF8A		-			
	exit				83	-			
main	ENDP			0018FF8C					
100 % 👻 🖣			18	0018FF8D					

Memory 1		_ Registers
Address: 0x0018	F80	EAX = FF10FFFF EBX = 00000000
0x0018FF80	00 00 05 00 10 ff ff ffÿÿÿ	ECX = 00000000 EDX = 00401005
0x0018FF88	a2 47 b5 83 8a 33 56 76 ¢GµƒŠ3Vv	ESI = 00000000 EDI = 00000000
0x0018FF90	00 e0 fd 7e d4 ff 18 00 .àý∼Ôÿ	EIP = 00401030 ESP = 0018FF88
0x0018FF98	72 9f 44 77 00 e0 fd 7e rŸDw.àý~	EBP = 0018FF94 EFL = 00000246
0x0018FFA0	38 f3 32 76 00 00 00 00 8ó2v	67

	Push/Pop	e	xam	ple	code: F	Proc1.a	asm	
			А	В	С	D	E	F
		1		Stack p	ush (filling) starts	from 1 lower th	e ESP, and ຄ	stays there
		2		Stack p	ops from the ESP	and moves 1 hi	gher	
		3	Stack	Data	Istruction	ESP	EAX	EBX
• r	oop EBX	4				0018FF8c	83B547A2	???????
- F		5	0018FF80		push EAX	0018FF88		
		6	0018FF81		pushd -240	0018FF84		
		7	0018FF82	05	pushw 5	0018FF82		
	mov EAX, 83B547A2h	8	0018FF83	00	pop EAX	0018FF86	FF100005	
	push EAX	9	0018FF84	10	pop AX	0018FF88	FF10FFFF	
	pushd -240 ;doub	10	0018FF85		pop EBX	0010FF8C		83B547A2
		11	0018FF86	FF				
	pushw 5; WORD Size	12	0018FF87	FF				
	pop EAX	13	0018FF88	A2				
	pop AX		0018FF89	47				
-	pop EBX		0018FF8A	-				
	exit	16	0018FF8B	83				
main		17	0018FF8C					
- main 1% + 4	FNDF	18	0018FF8D					
					1		-	-

mory 1									Registers
ldress: 0x001	8FF80								EAX = FF10FFFF EBX = 83B547A2
0018FF80	00 00	ð Ø5	00	10	ff	ff	ff	ÿÿÿ	ECX = 00000000 EDX = 00401005
0018FF88	a2 4	7 b5	83	8a	33	56	76	¢GµfŠ3Vv	ESI = 00000000 EDI = 00000000
0018FF90	00 e	ð fd	7e	d4	ff	18	00	.àý∼Ôÿ	EIP = 00401031 ESP = 0018FF8C
0018FF98	72 9	F 44	77	00	e0	fd	7e	rŸDw.àý~	EBP = 0018FF94 EFL = 00000246