Practice Questions

===================================

QUESTIONS

===================================

Question 1: Which instruction pushes all of the 32-bit general-purpose registers on the stack?

Answer 1: The instruction that pushes all of the 32-bit general-purpose registers on the stack is PUSHA.

Question 2: Which instruction pushes the 32-bit EFLAGS register on the stack?

Answer 2: The instruction that pushes the 32-bit EFLAGS register on the stack is PUSHFD.

Question 3: Which instruction pops the stack into the EFLAGS register?

Answer 3: The instruction that pops the stack into the EFLAGS register is POPFD.

Question 4: Challenge: Another assembler (called NASM) permits the PUSH instruction to list multiple specific registers. Why might this approach be better than the PUSHAD instruction in MASM? Here is a NASM example: PUSH EAX EBX ECX

Answer 4: NASM's approach of allowing the PUSH instruction to list multiple specific registers can be better in some cases because it provides more flexibility. It allows you to choose which registers to push onto the stack, whereas PUSHA in MASM pushes all the general-purpose registers. This can save stack space and execution time when you only need to save a subset of registers.

Question 5: Challenge: Suppose there were no PUSH instruction. Write a sequence of two other instructions that would accomplish the same as push eax.

Answer 5: If there were no PUSH instruction, you could achieve the same result as PUSH EAX using the following two instructions:

```
sub esp, 4
mov [esp], eax
```

Question 6: (True/False): The RET instruction pops the top of the stack into the instruction pointer.

Answer 6: False. The RET instruction pops the return address from the stack into the instruction pointer (EIP).

Question 7: (True/False): Nested procedure calls are not permitted by the Microsoft assembler unless the NESTED operator is used in the procedure definition.

Answer 7: False. Nested procedure calls are permitted without the need for the NESTED operator in the Microsoft assembler.

Question 8: (True/False): In protected mode, each procedure call uses a minimum of 4 bytes of stack space.

Answer 8: False. In protected mode, each procedure call doesn't necessarily use a minimum of 4 bytes of stack space. The actual stack space used depends on the number of parameters and local variables.

Question 9: (True/False): The ESI and EDI registers cannot be used when passing 32-bit parameters to procedures.

Answer 9: False. The ESI and EDI registers can be used when passing 32-bit parameters to procedures.

Question 10: (True/False): The ArraySum procedure (Section 5.2.5) receives a pointer to any array of doublewords.

Answer 10: False. The ArraySum procedure from Section 5.2.5 expects a pointer to an array of doublewords specifically.

Question 11: (True/False): The USES operator lets you name all registers that are modified within a procedure.

Answer 11: True. The USES operator lets you specify all registers that are modified within a procedure.

Question 12: (True/False): The USES operator only generates PUSH instructions, so you must code POP instructions yourself.

Answer 12: True. The USES operator generates PUSH instructions, so you need to code the corresponding POP instructions yourself.

Question 13: (True/False): The register list in the USES directive must use commas to separate the register names.

Answer 13: True. The register list in the USES directive must use commas to separate the register names.

Question 14: Which statement(s) in the ArraySum procedure (Section 5.2.5) would have to be modified so it could accumulate an array of 16-bit words? Create such a version of ArraySum and test it.

Answer 14: To accumulate an array of 16-bit words, you would need to modify the mov eax, [esi] and add esi, 4 statements to work with 16-bit words, like this:

```
mov ax, [esi]
add esi, 2
```

Question 15: What will be the final value in EAX after these instructions execute? push 5 push 6 pop eax pop eax

Answer 15: EAX will equal 5 after these instructions execute. The second pop eax instruction will overwrite the previous value of EAX.

Question 16: Which statement is true about what will happen when the example code runs?

```
1: main PROC
2: push 10
3: push 20
4: call Ex2Sub
5: pop eax
6: INVOKE ExitProcess,0
7: main ENDP

10: Ex2Sub PROC
11: pop eax
12: ret
13: Ex2Sub ENDP
```

Answer 16: a. EAX will equal 10 on line 6.

Question 17: Which statement is true about what will happen when the example code runs?

```
1: main PROC
2: mov eax,30
3: push eax
4: push 40
5: call Ex3Sub
6: INVOKE ExitProcess,0
7: main ENDP

10: Ex3Sub PROC
11: pusha
12: mov eax,80
13: popa
14: ret
15: Ex3Sub ENDP
```

Answer 17: d. The program will halt with a runtime error on Line 11 because there's no matching pop for the pusha instruction.

Question 18: Which statement is true about what will happen when the example code runs?

```
1: main PROC
2: mov eax,40
3: push offset Here
4: jmp Ex4Sub
5: Here:
6: mov eax,30
7: INVOKE ExitProcess,0
8: main ENDP

10: Ex4Sub PROC
11: ret
12: Ex4Sub ENDP
```

Answer 18: b. The program will halt with a runtime error on Line 4 because there's no matching pop for the push instruction.

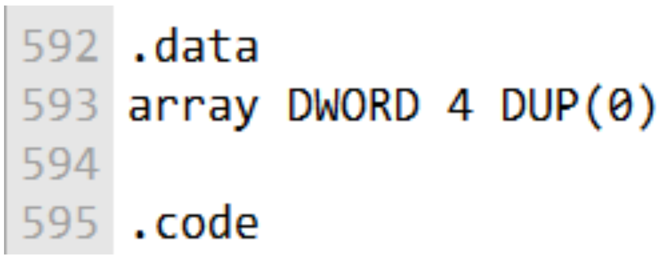
Question 19: Which statement is true about what will happen when the example code runs?

```
1: main PROC
2: mov edx,0
3: mov eax,40
4: push eax
5: call Ex5Sub
6: INVOKE ExitProcess,0
7: main ENDP

10: Ex5Sub PROC
11: pop eax
12: pop edx
13: push eax
14: ret
15: Ex5Sub ENDP
```

Answer 19: a. EDX will equal 40 on line 6.

Question 20: What values will be written to the array when the following code executes?



```
.data
array DWORD 4 DUP(0)

.code
```

In the provided code, you've declared an array named array with four double word (DWORD) elements, and you've initialized each element to 0 using 4 DUP(0). Therefore, when this code is executed, the array will contain the following values:

array[0] will be 0.
array[1] will be 0.
array[2] will be 0.
array[3] will be 0.

So, all elements in the array will have the value 0.