Saving and Restoring Registers

Subroutines often save the current contents of registers on the stack before modifying them.

This is a good practice, because the original values can be restored just before the subroutine returns.

This ensures that the subroutine does not modify registers that are used by the caller, and that the caller's state is preserved.

The ideal time to save registers is just after setting EBP to ESP, and just before reserving space for local variables.

This is because the stack grows below EBP, so pushing registers does not affect the displacement from EBP of parameters already on the stack.

Here is an example of a subroutine that saves and restores registers:

```
381 MySub PROC
382   push ebp
383   mov ebp, esp
384   push ecx
385   push edx
386   ; ...
387   pop edx
388   pop ecx
389   pop ebp
390   ret
391 MySub ENDP
```

The subroutine first pushes the base pointer (EBP) onto the stack. This saves the current value of the stack pointer, which is used as the base of the stack frame for the subroutine. The subroutine then moves the stack pointer to EBP, which makes EBP the base of the stack frame for the subroutine.

The subroutine then pushes the ECX and EDX registers onto the stack. These are two commonly used registers, so it is a good practice to

save them before modifying them.
The subroutine then performs its work.

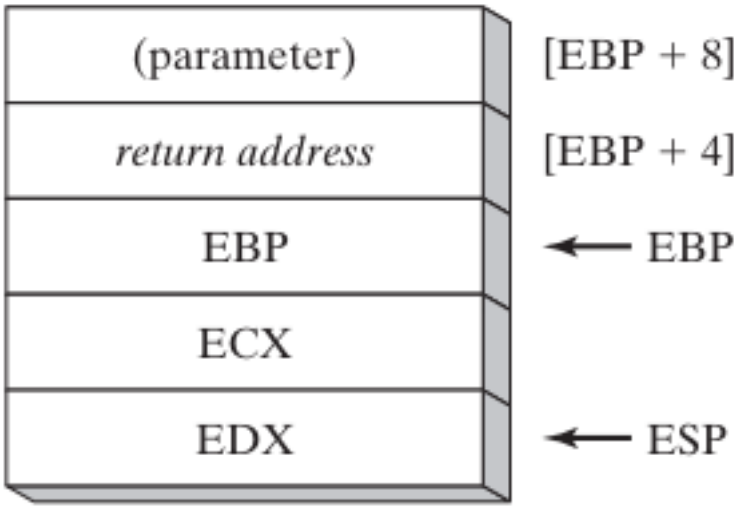
Before returning, the subroutine restores the ECX and EDX registers from the stack. It also pops the base pointer (EBP) from the stack, which restores the stack pointer to its original value.

Stack Frames

A stack frame is a region of the stack that is used to store the local variables and parameters of a subroutine. The stack frame is created when the subroutine is called, and it is destroyed when the subroutine returns.

The base pointer (EBP) register is used to point to the base of the stack frame. This allows the subroutine to access its local variables and parameters without having to keep track of the stack pointer explicitly.

Here is a diagram of a stack frame for the MySub procedure:



The **stack pointer (ESP)** is a register that points to the top of the stack. When a function is called, the caller pushes the function's parameters onto the stack and then calls the function. The called function then allocates space for its local variables on the stack.

In the case of MySub, the caller pushes two parameters onto the stack. When MySub is called, it first **saves the current value of the EBP register** onto the stack. This is because the EBP register is used to reference the stack frame for the current function.

MySub then **moves the ESP register into the EBP register.** This makes the EBP register point to the top of the stack frame for MySub. **MySub can then access its local variables** by using the EBP register as a base pointer.

The **first parameter to MySub is stored at [ESP + 8].** This is because the ESP register points to the top of the stack, and the first parameter was pushed onto the stack **before** the EBP register was pushed onto the stack.

The **return address for MySub is stored at [ESP + 4].** This is because the return address is the next instruction that will be executed after MySub returns. The return address is pushed onto the stack by the caller **before** the caller calls MySub.

The **EBP register** is stored at [ESP]. This is because the EBP register is used to reference the stack frame for the current function.

The **ECX and EDX registers are stored at [EBP - 4] and [EBP - 8],** respectively. This is because the ECX and EDX registers are **callee-saved registers**. This means that the caller is responsible for saving and restoring the values of these registers before and after calling MySub.

MySub can access its local variables by using the EBP register as a base pointer. For example, to access the first local variable, MySub would use the following instruction:

```
mov eax, [ebp + 12]
```

This instruction would copy the contents of the memory location at offset 12 from the EBP register into the EAX register.

When MySub is finished executing, it **pops the ECX and EDX registers off of the stack.** It then pops the EBP register off of the stack. This restores the EBP register to its previous value, which was the EBP register for the calling function.

Finally, **MySub executes the RET instruction.** This instruction returns to the caller and pops the return address off of the stack. The caller then executes the next instruction after the call to MySub.

Conclusion

Saving and restoring registers is a good practice for subroutines, because it ensures that the subroutine does not modify registers that are used by the caller, and that the caller's state is preserved. Stack frames are used to store the local variables and parameters of a subroutine.