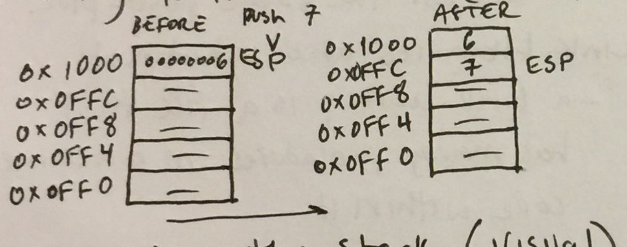
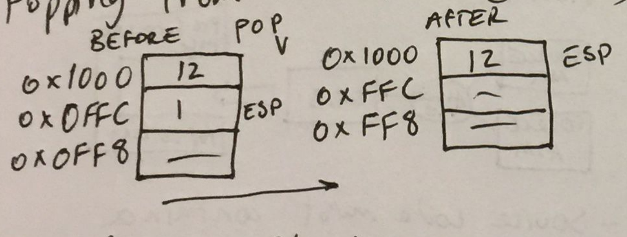
CS 271 WEEK 3
— Readings —

The Stack
- Like a stack of plates
- LIFO (Last In, First Out)
- Runtime Stack is supported by hadware within the CPU
- Runtime Stack uses <u>ESP</u> (stack pointer or stack pointer reg)
- ESP points to the hex offset of the most recently (~~topmost~~) pushed value
- In 32-bit mode, each stack loc. has 32-bits of space

Pushing to the Stack (Visual)


Popping from the Stack (Visual)


Uses for the Stack
- temporary storage for registers when their values are needed later, but need to be changed short-term.
- When a CALL is called, the CPU saves return addr on stack
- Stack is how you pass arguments to a subroutine.

PUSH and POP opcodes
- PUSH copies the source operand into the stack, decrementing ESP by (2 if 16-bit op. | 4 if 32-bit op)
- POP copies the current value pointed to by ESP into the "source" (dest) operand, then increments ESP by (2 if 16-bit | 4 if 32-bit).

PUSHFD, POPFD
- these opcodes push and pop the EFLAGS to/from the stack
- Pushfd is essentially "backing up" the flags, and Popfd essentially "restores from backup".
- It is best to use pushfd, then pop to a variable, then inverse to restore.

PUSHAD, POPAD, PUSHA, POPA
- PushAD and POP AD push or pop these registers on to/off of the Stack:

  EAX ECX EDX EBX ESP* EBP ESI EDI

  * before PUSHAD EXECUTION
- PUSHA and POPA are only for 16-bit progaming. they push/pop these registers to/from the stack:

  AX CX DX BX SP BP SI DI
- PUSHAD and POPAD <u>can</u> be used to "back up" all the registers, but beware of any overwritten values that will be lost in the process of POPAD!!

Procedures

- AKA Methods, functions, subroutines
- Must have a PROC and ENDP directive:

```
temp_proc PROC
   ...
   ...
   ret
temp_proc ENDP
```

- Labels can only be accessed within their own procedure, unless they are defined as global labels : label_name :: note double colon
- Should document Procedures with
  - T : TASK (goal of proc)
  - R : Recieves (inputs, pre-conditions)
  - R : Returns (outputs, post-con.)
  - R : Requires (Necessary settings/values)

Stack when Calling Procedures

PROC_Main PROC	Address		Procedure_1
CALL Procedure_1	0x20	0x40	mov ebx,eax
mov eax,ebx	0x25	0x45	ret

```
   ...
   ...
Proc_main ENDP
```

1] Call function moves 0x25 onto the stack, and moves 0x40 into EIP

2] When procedure_1 calls ret, 0x25 is popped off the stack, into EIP

Arguments for Procedures

- usually, we use general-purpose registers for this

USES Operator

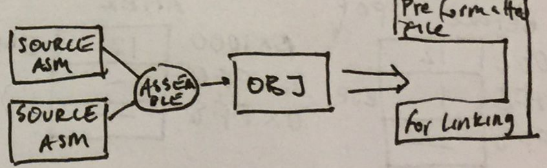
- when defining a process, the uses operator will "manage" all the pushing and popping for the registers listed as to ensure that their values are the same once the proc has completed, as when the proc started
- Ex:

```
temp PROC USES ecx, esi
   ...
   ret
temp ENDP
```

↳ USES will make sure the assembler will put a "push" for each register at the top of "temp" and a "pop" at the end, thus keeping the values of ecx, esi the same post-proc.

Link Libraries and Whatnot

- a link library is a file that has many procedures in machine code within it
- VISUAL!



- Source code must contain a PROTO to be able to have a file linked with the library derived from that source be able to use call opcode.
- When assembling, assembler leaves call's for the linker (sometimes...), and linker copies in relevant code and addresses.