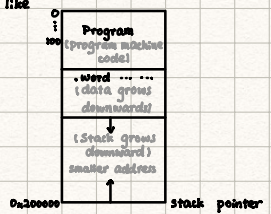
Subroutine doesn't know which registers were being used by the "calling" code.

⇒ in general it show "save" those registers that it wants to use (somewhere) & "restore" them to their original values afterwards.

→ we will save & restore registers on the system stack: Recall: a stack.

- Data structure for saving & restoring.
- Two operations - push an item on to a stack
  - pop an item off a stack.
- In ARM, register r13 is reserved to contain the stack pointer → it contains the address of the current item on the top of the stack. r13 = SP
- All of memory looks like



Basic Stack Operators

① Initialize the stack pointer MOV SP, #0x200000

② To "push" a single (4 byte) word onto stack

(i) decide stack pointer by 4 SUB SP,#4 ⇒ in one instruction str r1,[sp,#4]! ⇒ simpler push {r1}

(ii) store the value (say it is in r1) at that location str r1,[sp]

③ To pop the item on top of stack off & put into r2

(i) ldr r2,[SP] // item on top → r2 ⇒ ldr r2,[sp],#4

(ii) add sp,#4 // "remove" it from stack pop {r2}

Let's use the stack to save & restore registers in subroutines as above

```
_start: MOV SP, #0x200000   //initialize stack
        MOV r4, #6
        MOV r5, #9
        BL  MY_SUB          // LR←PC, PC ← MY_SUB
// MY_SUB will use r4 & r5 for something else
  and it will call another subroutine
  so I must save and restore r4, r5 & link register (r14)
MY_SUB: STR r4,[SP,#-4]!    // push r4
        STR r5,[SP,#-4]!    //  ...  r5   ⇒  PUSH {R4, R5, LR}
        STR LR,[SP,#-4]!    //  ...  LR
        MOV r4,#10
        MOV r5,#20
        BL  elsewhere       // calling a subroutine uses LR
        LDR LR,[SP],#4      // pop top of stack into LR → order matters! (order of push & pop is the same as pushed & poped)
        LDR r5,[SP],#4      // pop r5          ⇒ pop {r4,r5,LR}
        LDR r4,[SP],#4      // pop r4
        MOV PC,LR           // return from subroutine
```

ARM using the following "convention" for sending information to & from subroutines

① parameters passed to a subroutine should be placed in registers. R0 → R3

→ it is ok to change R0 → R3 (Don't have to save & restore)

→ if need more than 4 parameters → use stack

② it is **not** ok to change R4 → R12

If do, must save & restore them

③ A single result from a subroutine is put into R0, if more needed, use stack