Lecture 5
Subroutines and stack

Computing platforms

Novosibirsk State University
University of Hertfordshire

D. Irtegov, A.Shafarenko

2018

Stack

- Stack as a primitive (opaque type with predefined set of operations)
- Primitive means that we have semantic of the operations
- But do not know (or should not rely on) details of implementation.
- So we can change implementation without changing the semantics
- Two operations: push and pop
- Push stores data in some [internal] storage
- Pop retrieves them in LIFO (Last In First Out) order

Stack on CdM-8

- SP register (we discussed it during CocoIDE demonstration)
- Main memory pointed by SP register (*SP)
- Push rn
  - ((SP-1)→SP) then (rn → *SP)
- Pop rn
  - (*SP → rn) then ((SP+1)→SP)
- At CPU power on, SP==0
- First push makes SP==255, so stack starts from the top of the RAM
- Be careful!

How stack works

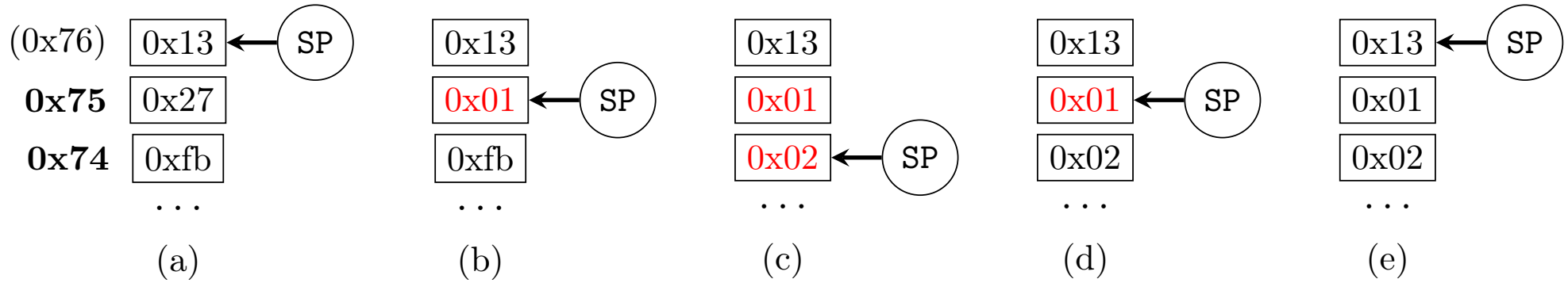


Figure 6.1: Stack behaviour: (a) initial state: the stack is empty; (b) after `0x01` has been pushed; (c) after `0x02` has been pushed; (d) after a pop; (e) after another pop, stack is empty again.

Be careful!

- If you push too many times, you can overwrite your program!
- If you pop more times than push, SP wraps over to 0 and you can overwrite your program again!
- Commercial CPU (x86, ARM) have hardware protection against this
  - We will discuss it in Operating System course
  - And this protection is not 100% bulletproof
    (you can mess your stack if you really want to)
- CdM-8, like most other 8-bit CPU, has no hardware protection
  (at least in basic configuration)

Wait, there is more!

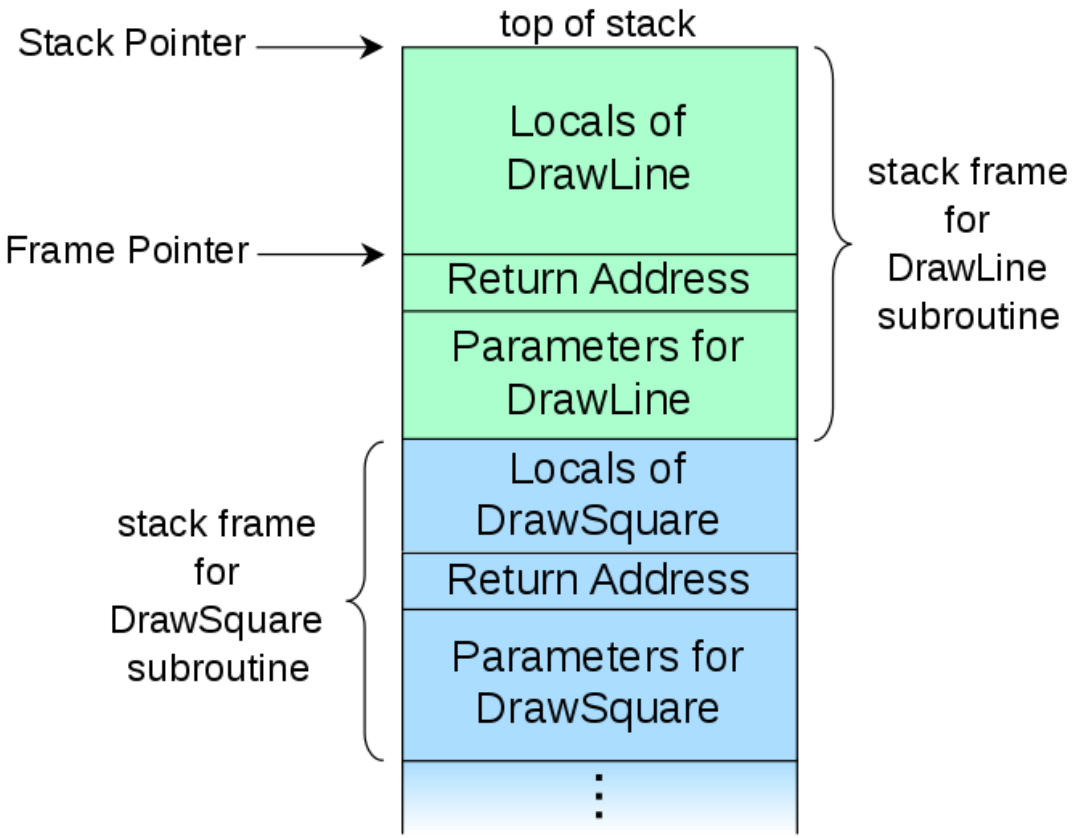
- Ldsa rn, offset
  - SP+offset → rn
  - Not in instruction-set.pdf (we’re working on this)
- Addsp n
  - SP=SP+n
- Ldsp rn, Stsp rn
  - Move SP to/from a GP register n

Subroutine call and return

- Jsr [const]
  - SP-1→SP, then PC → *SP, then const → PC
  - In most modern CPUs this instruction is called Call
  - Jsr mnemonic comes from IBM 360
- Rts
  - *SP → *PC, then SP+1→SP
- Jsrr rn
  - SP-1→SP, then PC → *SP, then rn → PC
  - You can implement function pointers!

Subroutine activation record

- Create a space for local variables
- New space for every new call
- Allows recursion
  - CdM-8 has no frame pointer
- Caller push param to stack
- Then jsr to callee
- Then callee addsp frame size
- And uses ldsa to access values



Special syntax for local variables (and structs!)

```
                    1         tplate  foo
00:                 2         dc      "abcde"
05:                 3  a:     ds      13
12:                 4         dc      "this is it"
1c:                 5  b:     ds      7
23:                 6
                    7         asect   0
                    8:  main:
00: c9 05           9         ldsa    r1,foo.a
02: ca 1c           10        ldsa    r2,foo.b
04: cb 23           11        ldsa    r3,foo._
              ….
```

What exactly tplate directive does?

- A template is a *named* absolute section that
  - starts at 0,
  - does not allocate any memory
    - dc parameters are only placeholders
  - is accessible in the whole source file,
  - the section's text can not be interrupted and continued later
- Each label defined within a template is absolute and must be referenced using the prefix name.

Calling conventions

- How to pass parameters
  - On registers?
    - Fast, but CdM-8 has too few registers
    - Cannot pass structures
  - On stack?
    - Relatively slow
  - Who cleans the stack after the call?
    - On CdM-8 it is hard for callee to clean the stack, but other CPU have means for that
    - Callee must know size of parameters to clean the stack (impossible in C)
- How to save registers?
  - Clean protocol (callee must save all registers before touching them)
  - Dirty protocol (callee can change any register)
  - Hybrid protocol (some registers must be saved, some are not)