Lecture 5
Subroutines and stack

Computing platforms

Novosibirsk State University
University of Hertfordshire

D. Irtegov, A.Shafarenko

2018

сохранение и восстановление содержимого регистров
СТЕК (stack)

save rn

restore rn

Правила использования:

1. Любая управляющая структура, возникающая между сохранением и соответствующим восстановлением, должна находиться в этой области, включая начальную и конечную части, например: if and fi, while and wend, etc.
2. Инструкции управления без области действия break, continue, is, stays и т. д. могут возникать только между save и соответствующим restore вместе (внутри) всей конструкции, которую они контролируют.
3. Если программист хочет восстановить сохраненное значение в другом регистре, это можно сделать, указав этот регистр в качестве аргумента восстановления, например восстановить.

```
do

    save r2      # free up r2
    save r0      # free up r0
    ldi r0,squares
   add r1,r0
   ld r0,r3 # r3=squares[r1]=b^2
   ldi r0,squares
   add r2,r0
   ld r0,r2 # r2=squares[r2]=a^2
   add r2,r3 # r3=a^2+b^2
 restore              # r0=c again

if

is cs # if unsigned OFL, not Pyth
shl r3 # ... ensure r3!=0
```

```
else
    ldi r2,squares        # otherwise calc the diff
    add r0,r2
    ld r2,r2              # r2=c^2
    sub r2,r3             # r3=c^2-a^2-b^2
 fi
restore        # r2=a as before

If                        # Pythagorean

is eq          # (the flags were set by sub)
break 3             # exit 3 loops at once

 fi

dec r2             # otherwise
 until eq           # next iteratio
 dec r1          # next iteration
```

Subroutines (Подпрограммы)

Циклы — это одна из форм повторяющихся вычислений; тело цикла повторяется определенное количество раз в зависимости от определенных условий, но все равно используется в программе один раз. Управление передается в цикл предыдущей инструкцией, а из цикла оно переходит в следующую инструкцию, следующую за ней. Даже если вычисление может быть повторено, все же невозможно повторить его в разных точках программы; если бы это было необходимо, нужно было бы включить туда копию цикла. Копирование кода для повторных вычислений имеет серьезные последствия на всех уровнях, от производительности платформы до удобства сопровождения программного обеспечения. Копии кода занимают адресное пространство, которое является важным ресурсом для небольших встроенных систем (таких как микроконтроллеры в автомобилях двигателей и стиральных машин). Копии идентичны только изначально, когда программа впервые написана. Всякий раз, когда инженеру нужно изменить код, он должен помнить о последовательном изменении всех копий. Это само по себе является источником ошибок, которые трудно обнаружить и исправить.

Subroutines

jsr name — имя является меткой общего сегмента кода, называемого подпрограммой
Имя в jsr — это метка, помещаемая в начале подпрограммы.

Инструкция jsr изменяет поток управления программой. Вместо того, чтобы передать управление следующей инструкции (т.е. той, которая занимает следующую ячейку памяти), она передает его точке, помеченной как имя. Подпрограмма будет выполняться, инструкция за инструкцией, пока платформа не встретит инструкцию.

rts

В более общем контексте мы обычно называем программу, которая переходит к подпрограмме, вызывающей программой, сам переход часто называют вызовом подпрограммы, и, следовательно, подпрограмму часто называют вызываемой программой.

```
myprog:
ldi r0,a                                  # compute e=a*b+c*d
ld r0,r0 # r0=a
ldi r1,b
ld r1,r1 # r1=b
jst mult      # r0=a, r1=b, a*b expected in r2
        move r2,r3 # save a*b in r3 for now
         ldi r0,c
         ld r0,r0 # r0=c
         ldi r1,d
         ld r1,r1 # r1=d
jst mult         # r0=c, r1=d, c*d expected in r2
         add r2,r3 # r3=a*b+c*d
         ldi r0,e
         st r0,r3 # e=a*b+c*d, job done
    halt
# subroutine mult: computers r2=r0*r1
mult:
          clr r2
          while
               dec r1
          stays ne
              add r0,r2
           wend
        rts
end
```

Subroutines

Расположение исходных данных для подпрограммы (а также ожидание того, что результат займет определенное место в памяти или регистрах) обычно называют соглашениями о вызовах программы.

Правила, определяющие, какие регистры могут использоваться в качестве рабочего пространства подпрограммы, называются дисциплиной регистров.

адрес, на который должно быть передано управление после завершения подпрограммы называется адресом возврата

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

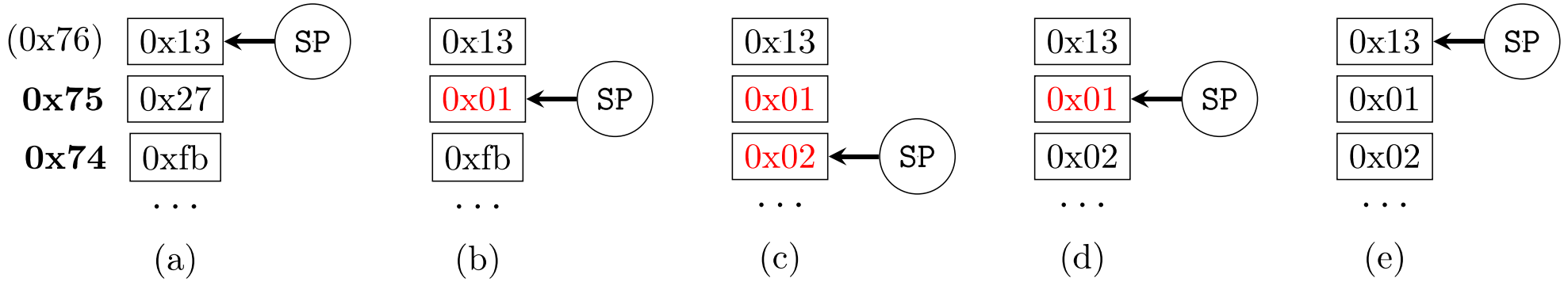


Figure 6.1: Stack behaviour: (a) initial state: the stack is empty; (b) after `0x01` has been pushed; (c) after `0x02` has been pushed; (d) after a pop; (e) after another pop, stack is empty again.

```
run myprog
# play with the stack
            run myprog
myprog:
            ldi r1,0x01
            ldi r2,0x02
    push r1
    push r2
    pop r0              # now r0=0x02
    pop r0              # now r0=0x01
halt                    # now the stack is empty
end
```

Be careful!

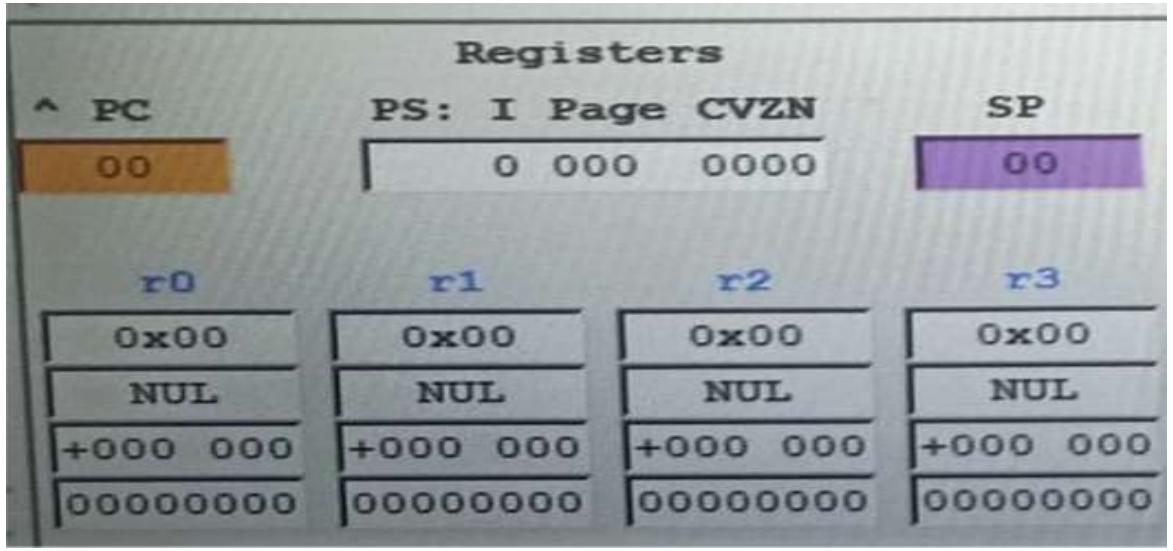
- If you push too many times, you can overwrite your program!
- If you pop more times than push, SP wraps over to 0 and you can overwrite your program again!
- Commercial CPU (x86, ARM) have hardware protection against this
  - We will discuss it in Operating System course
  - And this protection is 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

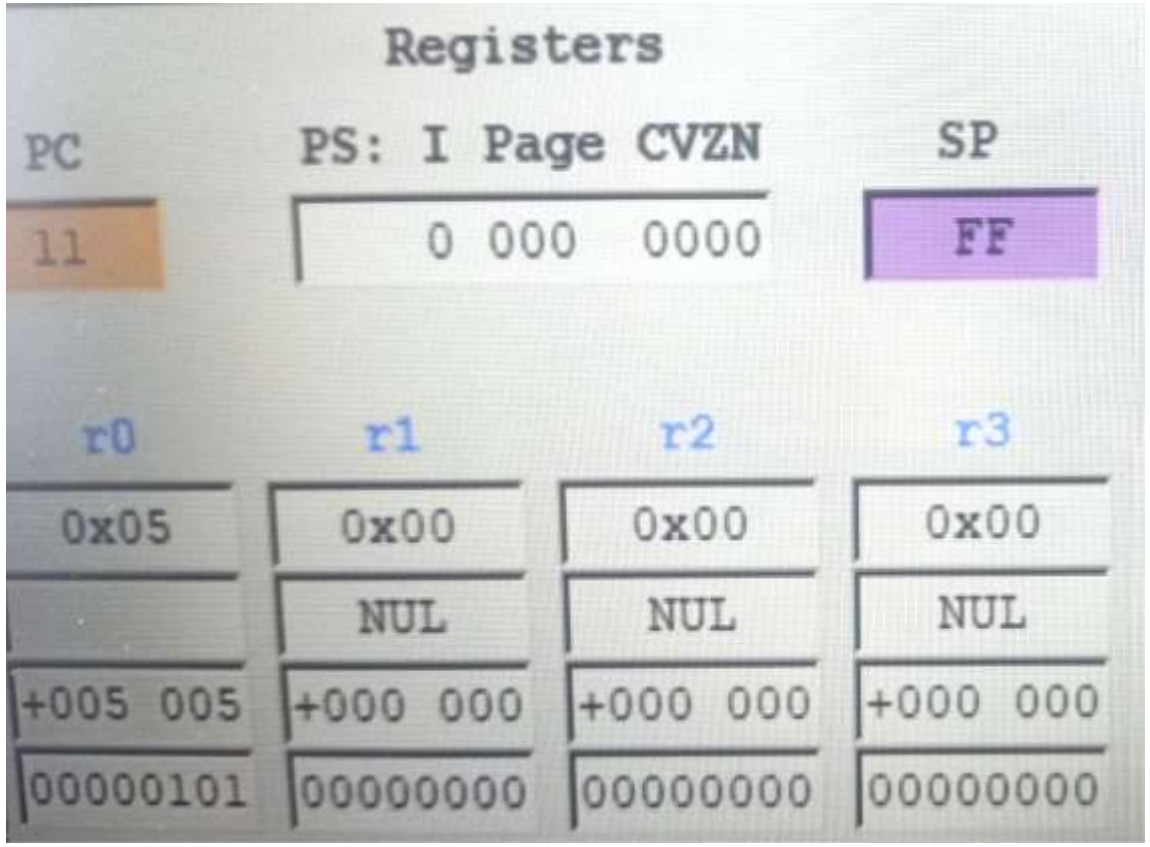
копирует содержимое указателя стека в rn, второй делает обратное: копирует содержимое регистра обратно в SP.

```
str: dc "abrakadabra",0          # 5 a's, 2 b's, ...

myprog: ldi r0,str               # str is the parameter
        jsr freq
        halt
```



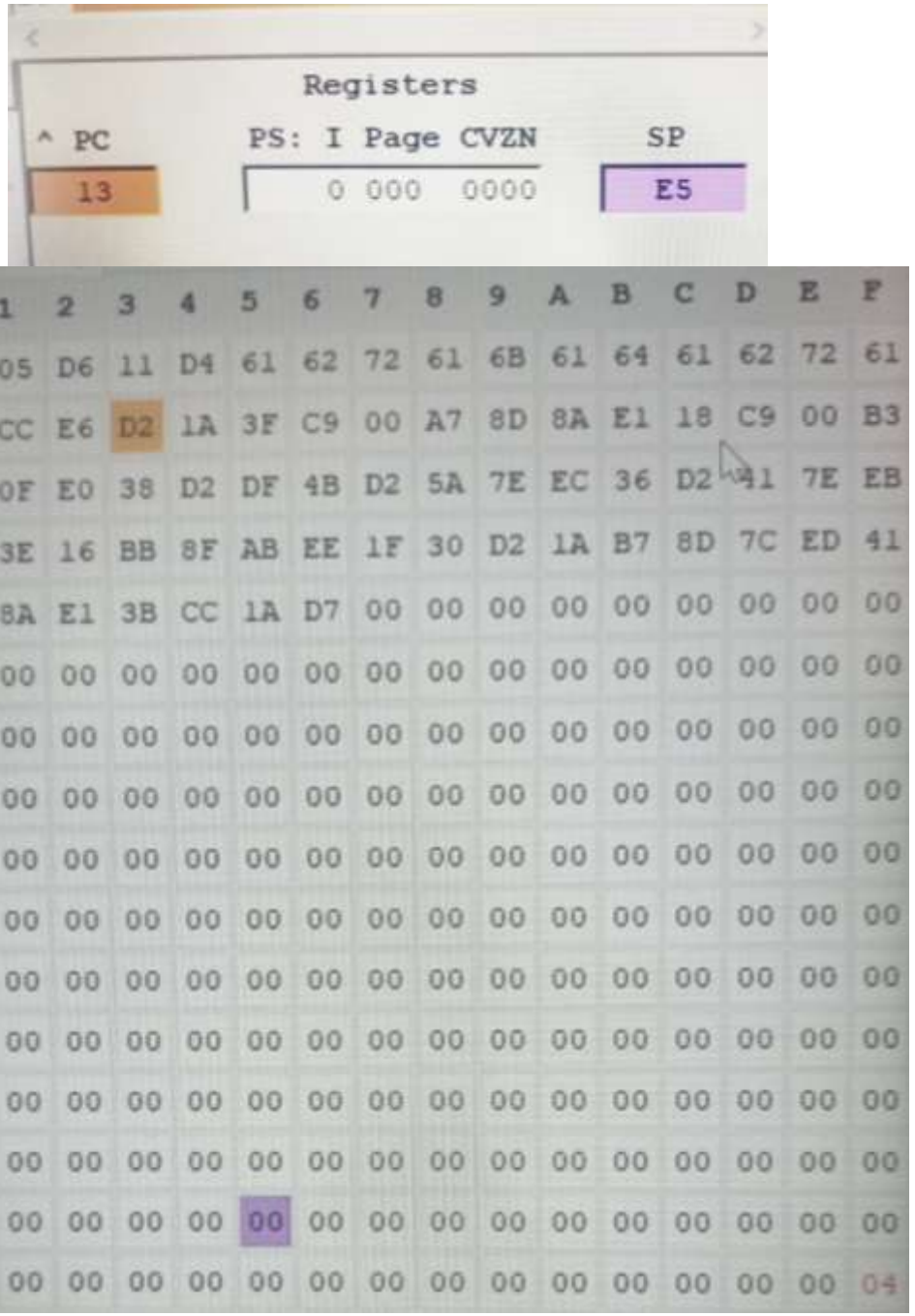
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	D0	05	D6	11	D4	61	62	72	61	6B	61	64	61	62	72	61
1	00	CC	E6	D2	1A	3F	C9	00	A7	8D	8A	E1	18	C9	00	B3
2	8C	0F	E0	38	D2	DF	4B	D2	5A	7E	EC	36	D2	41	7E	EB
3	36	3E	16	BB	8F	AB	EE	1F	30	D2	1A	B7	8D	7C	ED	41
4	0C	8A	E1	3B	CC	1A	D7	00	00	00	00	00	00	00	00	00
5	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
6	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
7	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
8	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
9	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
A	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
B	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
C	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
D	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
E	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
F	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	D0	05	D6	11	D4	61	62	72	61	6B	61	64	61	62	72	61
1	00	CC	E6	D2	1A	3F	C9	00	A7	8D	8A	E1	18	C9	00	B3
2	8C	0F	E0	38	D2	DF	4B	D2	5A	7E	EC	36	D2	41	7E	EB
3	36	3E	16	BB	8F	AB	EE	1F	30	D2	1A	B7	8D	7C	ED	41
4	0C	8A	E1	3B	CC	1A	D7	00	00	00	00	00	00	00	00	00
5	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
6	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
7	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
8	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
9	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
A	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
B	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
C	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
D	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
E	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
F	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	04



04 - адрес возврата

```
freq:
                        # allocate stack memory

    addsp -26       # make a table for 26 letters of the alphabet

                        # initialise table with 0s

     ldi r2,26              # r2 holds the downcount
    clr r3                 # r3 holds initial value
    ldsa r1,0              # r1->beginning of table

do

    st r1,r3            # initialise current cell
    inc r1              # r1->next cell
   dec r2              # decrement the counter
   until eq               # if we have done it 26 times, stop
  ldsa r1,0             # r1->beginning of table again
```

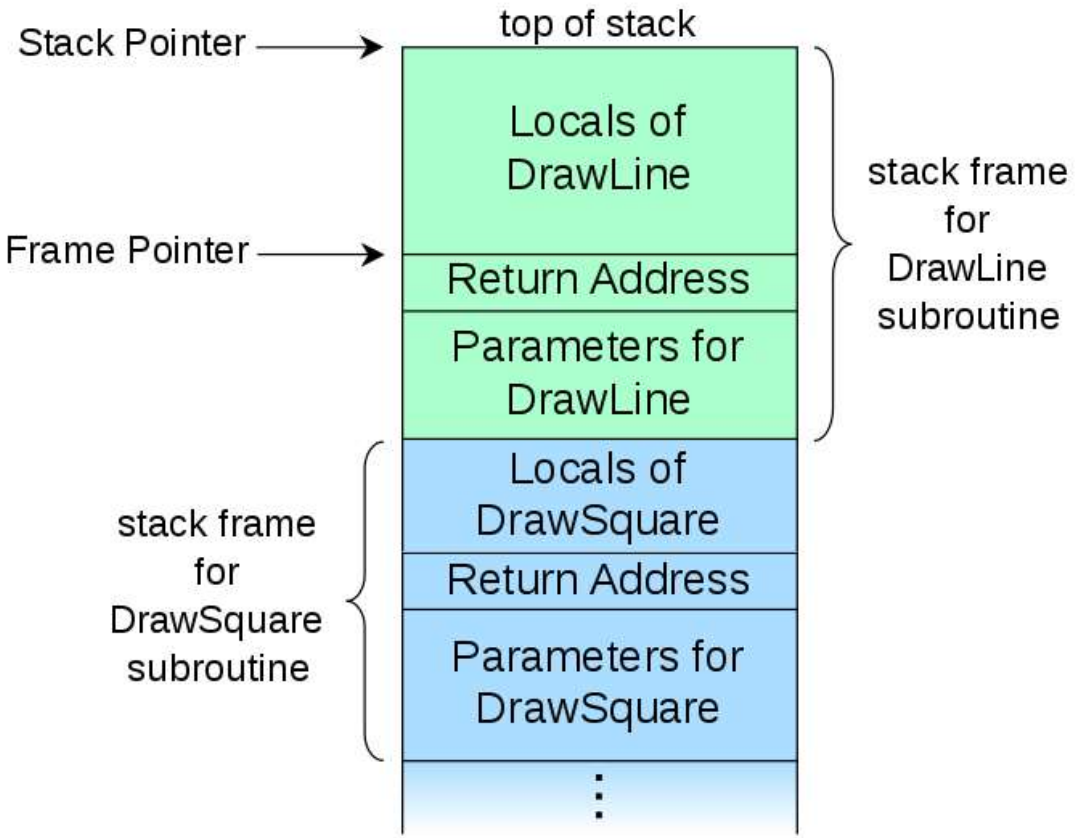


Subroutine call and return

- 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)