Tutorial4.2

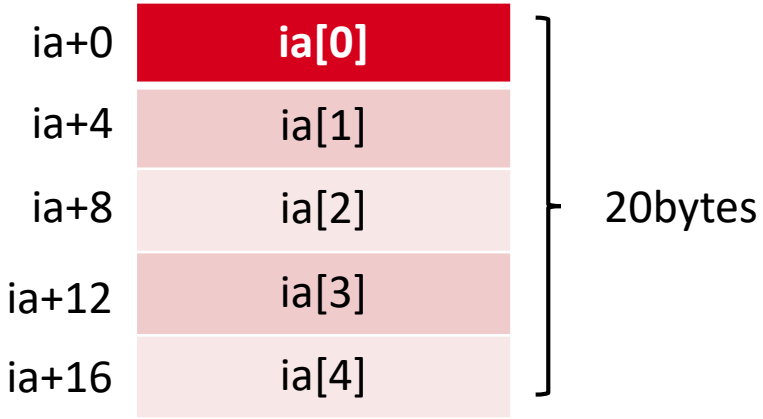
One Dimensional Arrays

Lei Wang

lei.wang2@ucalgary.ca

One dimensional Arrays

- Store consecutive sequence of variables
  - Block size = number of elements * element size
  - address of element i is:
    - Base address + (i * element size)
  - Eg: int ia[5];



UNIVERSITY OF CALGARY

One-Dimensional arrays

- Array is allocated in the function's stack frame like other variables
  - Eg: C code

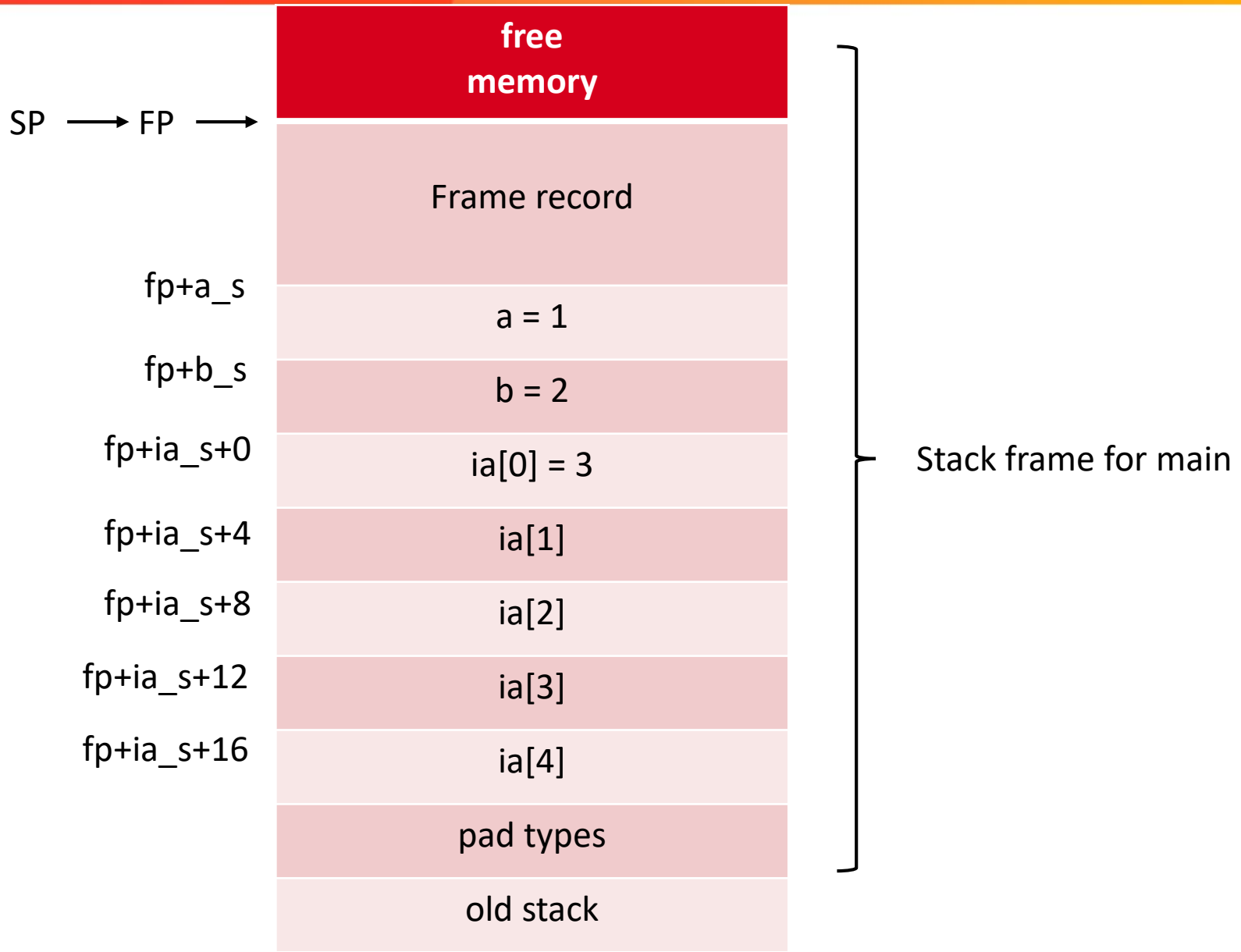
```
int main ()
{
    int a, b, ia[5];
    a = 1;
    b = 2;
    ia[0] = 3;

    …
}
```

Assembly code:

```
a_size = 4
b_size = 4
ia_size = 5*4
alloc = -(16 + a_size + b_size + ia_size) &-16
dealloc = -alloc
a_s = 16
b_s = 20
ia_s = 24
…
main:   stp     x29, x30, [sp, alloc]!
        mov     x29, sp
        mov     w22, 1
        str     w22, [x29, a_s]
        mov     w22, 2
        str     w22, [x29, b_s]
        mov     w22, 3
        str     w22, [x29, ia_s]
        …
        ldp     x29, x30, [sp], dealloc
        ret
```


UNIVERSITY OF CALGARY

Memory used in stack



UNIVERSITY OF CALGARY

access elements in array

- Array elements are accessed using load and store instructions
  - Eg: ia[2] = 13;

```
define(ia_base_r,  x19)
define(index_r,  x20)
define(offset_r,  x21)
…
main:   add     ia_base_r,  x29,  ia_s      //Calculate array base address, = address of ia[0]
        mov     index_r,  2                 //set index to 2
        lsl     offset_r,  index,  2        //offset = i * element size, index << 2  = index* 4 (4bytes)

        mov     w22,  13
        str     w22,  [ia_base_r,  offset_r] //ia[2] = 13
```


UNIVERSITY OF CALGARY

access elements in array

- Can be optimized by doing the LSL in the STR instruction:

  Eg:

```
move   w22,  13                          //Set  index to 2
str    w22,  [ia_base_r,  index_r,  LSL  2]    //ia[2] = 13
```

- if index is a *w* register, one must sign extend first:

  Eg:

```
define(index_r,  w20)
…
str    w22,  [ia_base_r,  index_r,  SXTW 2]  //ia[2] = 13
// here SXTW 2 means  sign_extend(w2) << n
```


UNIVERSITY OF CALGARY

addressing mode

STR/LDR X0, [X1]	Store to/Load from the address in X1
STR/LDR X0, [X1, #n]	Store to/Load from the address X1 + n
STR/LDR X0, [X1, X2]	Store to/Load from X1 + X2
STR/LDR X0, [X1, X2, LSL, #n]	Store to/Load from X1 + (X2<<n)
LDR X0, [X1, W2, SXTW]	Store to/Load from X1 + sign_extend(W2)
STR/LDR X0, [X1, W2, SXTW, #n]	Store to/Load from X1 + (sign_extend(w2) << n)

UNIVERSITY OF CALGARY

examine address in gdb

- use command x
  - **x** /[*Length*][*Format*][*Address expression*]
- x/15i main : examine 15 lines of instruction from address main
- x/d $fp+16 : examine 1 decimal value from address fp+16
- use gcc -g while compiling to get more info in gdb

UNIVERSITY OF CALGARY

Coding&debugging Pratice

```c
#include <stdio.h>
#define SIZE 10
//calculate the sum of 10 random numbers that is less than 256
int main()
{
    int array[SIZE];
    int i = 0;
    int sum = 0;

    for(i = 0; i < SIZE; i++){
        array[i] = rand() & 0xFF;
    }

    for(i = 0; i < SIZE; i++){
        sum  = sum + array[i];
        printf("array[%d] = %d(sum = %d)\n", i, array[i], sum);
    }
    return 0;
}
```

```
size = 10                                   //define the number of elements in the array
array_size = size * 4                       //define equates for size of array
i_size = 4                                  //size of i
sum_size = 4                                //size of sum
array_s = 16                                //Define equates for stack offset of array
i_s = array_s + array_size                  //stack offset of i
sum_s = i_s + i_size                        //stack offset of sum
var_size = array_size + i_size + sum_size   //Calculate memory needed for local variables
alloc = -(16 + var_size) & -16              //Calculate amount of memory to allocate and deallocate
dealloc = -allocate
fp      .req    x29                         //define register aliases
lr      .req    x30

fmt:    .string "array[%d] = %d (sum = %d)\n"

        .balign 4
        .global main
main:   stp     fp, lr, [sp, alloc]!        //allocate space for local variables
        mov     fp, sp                      //update FP to current SP

        mov     w19, 0
        str     w19, [fp, i_s]              //initialize i to 0
        mov     w19, 0
        str     w19, [fp, sum_s]            //initialize sum to 0
        b       test1                       //branch to test at bottom of loop1

loop1:  bl      rand                        //call rand() function
        and     w20, w0, 0xFF               //mod 256 the result
        add     x21, fp, array_s            //
        ldr     w19, [fp, i_s]              //get current i
        str     w20, [x21, w19, SXTW 2]     //store result w20 into array[i]
        ldr     w19, [fp, i_s]              //get current i
        add     w19, w19, 1                 //increment i by 1
        str     w19, [fp, i_s]              //store the current i
test1:  cmp     w19, size                   //loop while
        b.lt    loop1

        mov     w19, 0
        str     w19, [fp, i_s]              //set i to 0 again for the loop2
        b       test2                       //branch to test at bottom of loop2

loop2:  ldr     w19, [fp, i_s]              //get the current i
        add     x21, fp, array_s            //calculate array base address
        ldr     w20, [x21, w19, SXTW 2]     //get the value of array[i]
        ldr     w21, [fp, sum_s]            //get the current sum
        add     w21, w21, w20               //sum = sum + array[i]
        str     w21, [fp, sum_s]            //store the current sum
        adrp    x0, fmt                     //arg 0 high
        add     x0, x0, :lo12:fmt           //arg 0 low
        ldr     w1, [fp, i_s]               //arg 1: i
        add     x21, fp, array_s            //calculate array base address
        ldr     w2, [x21, w19, SXIW 2]      //arg 2: array[i]
        mov     w3, [fp, sum_s]             //arg 3: sum
        bl      printf                      //call printf
        add     w19, w19, 1                 //i++
        str     w19, [fp, i_s]              //store current i

test2:  cmp     w19, size
        b.lt    loop2

        mov     w0, 0
        ldp     fp, lr, [sp], dealloc       //deallocate stack memory
        ret
```