CSCI 331:
Introduction to Computer Security

Lecture 13: How C passes arguments

Instructor: Dan Barowy

Williams

Topics

Solution to Lab 4

How C passes arguments

Announcements

1. "The Ph.D. in CS: Getting There and Being Successful" on Monday, November 1 at 7 pm EST.
   Register in advance at https://bit.ly/3EjUyzw
2. TA applications due tomorrow.
   Please consider “giving back.”
3. Sandia National Labs
   Internships in Cybersecurity R&D
   https://cg.sandia.gov
   Job ID: 677929 (Albuquerque, NM)
   Job ID: 677896 (Livermore, CA)
   (American citizens only—sorry!)

Your to-dos

1. Project partner form, **due Sunday 10/31**.
2. Lab 5, **due Sunday 11/7**.
3. Project part 2, **due Sunday 11/14**.

~~Paper discussion~~

The program you examined in lab 4

```
void foo() {}

int main() {
  foo();
}
```

What does it do?

Why am I learning this?

Wise words from my favorite philosopher:

"I find it hard to remember things
I don't give a crap about." —House, M.D.

Why you should "give a crap" about assembly:

It's key to understanding control flow integrity attacks and defenses.

Caller vs callee

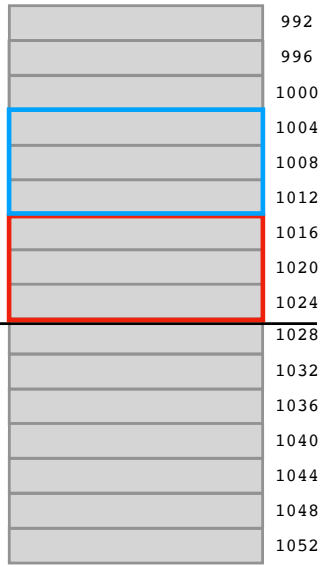
main calls foo

callee → foo

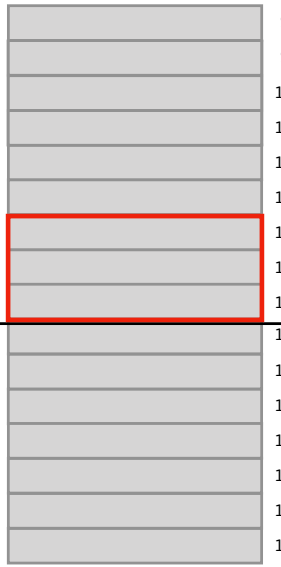
callee: the thing that is called

caller → main

caller: the thing that calls



ARM Calling Convention



sp : stack top

main

fp : stack base

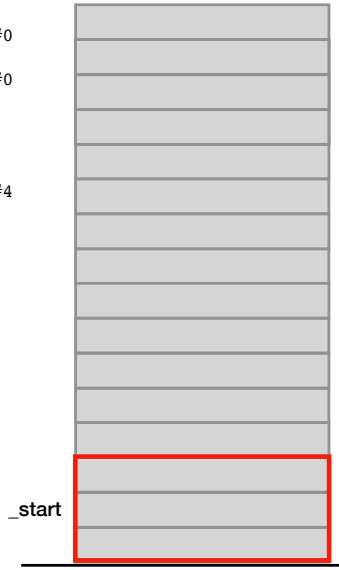
pc : instruction pointer

Class Activity

```
foo:
0  push  {fp}
4  add   fp, sp, #0
8  nop
12 add   sp, fp, #0
16 pop   {r11}
20 bx lr
main:
24 push  {fp, lr}
28 add   fp, sp, #4
32 bl foo
36 mov   r3, #0
40 mov   r0, r3
44 pop   {fp, pc}
_start:
…
48 bl    main
52 …
```

pc →

r0 = 0
r3 = 0
fp =1052
sp =1044
lr = 52
pc = 24

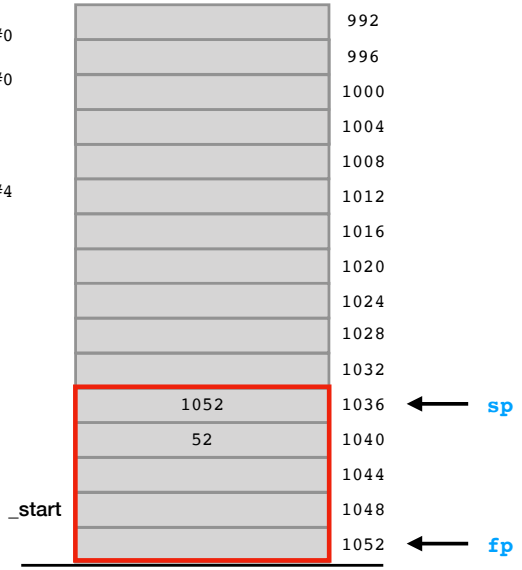


Class Activity

```
foo:
0  push  {fp}
4  add   fp, sp, #0
8  nop
12 add   sp, fp, #0
16 pop   {r11}
20 bx lr
main:
24 push  {fp, lr}
28 add   fp, sp, #4
32 bl foo
36 mov   r3, #0
40 mov   r0, r3
44 pop   {fp, pc}
_start:
…
48 bl    main
52 …
```

pc →

r0 = 0
r3 = 0
fp =1052
sp =1036
lr = 52
pc = 28

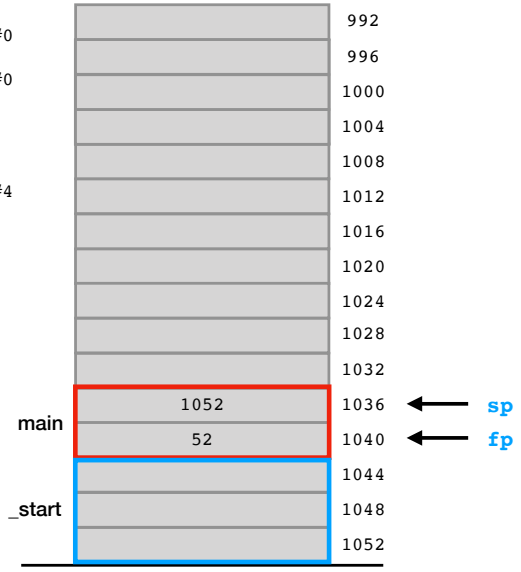


Class Activity

```
foo:
0  push  {fp}
4  add   fp, sp, #0
8  nop
12 add   sp, fp, #0
16 pop   {r11}
20 bx lr
main:
24 push  {fp, lr}
28 add   fp, sp, #4
32 bl foo
36 mov   r3, #0
40 mov   r0, r3
44 pop   {fp, pc}
_start:
…
48 bl    main
52 …
```

pc → 32 bl foo

r0 = 0
r3 = 0
fp =1040
sp =1036
lr = 52
pc = 32

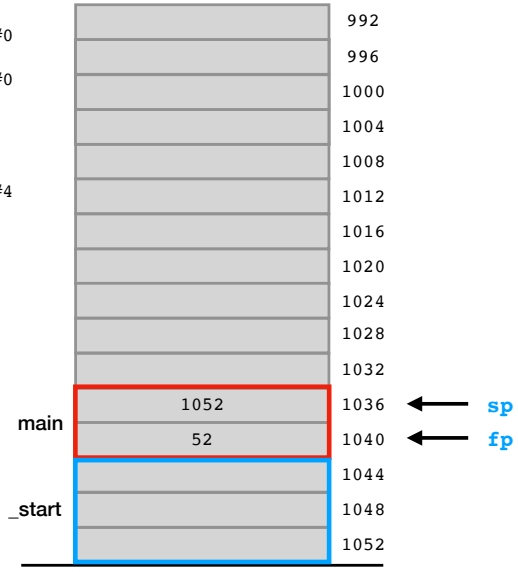


Class Activity

```
foo:
0  push  {fp}
4  add   fp, sp, #0
8  nop
12 add   sp, fp, #0
16 pop   {r11}
20 bx lr
main:
24 push  {fp, lr}
28 add   fp, sp, #4
32 bl foo
36 mov   r3, #0
40 mov   r0, r3
44 pop   {fp, pc}
_start:
…
48 bl    main
52 …
```

pc → 0 push {fp}

r0 = 0
r3 = 0
fp =1040
sp =1036
lr = 36
pc = 0

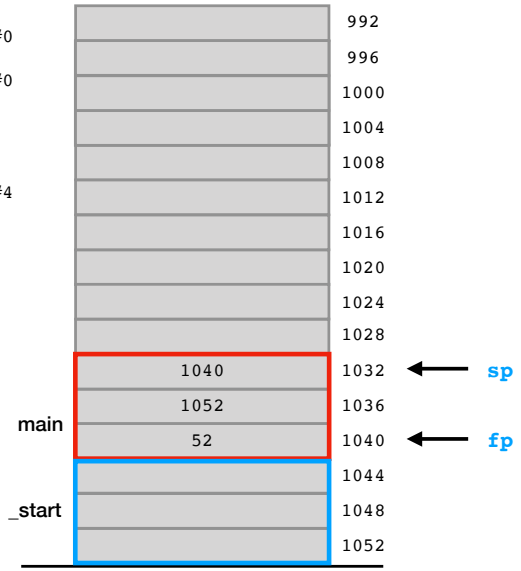


Class Activity

```
foo:
0  push  {fp}
4  add   fp, sp, #0
8  nop
12 add   sp, fp, #0
16 pop   {r11}
20 bx lr
main:
24 push  {fp, lr}
28 add   fp, sp, #4
32 bl foo
36 mov   r3, #0
40 mov   r0, r3
44 pop   {fp, pc}
_start:
…
48 bl    main
52 …
```

pc → 4 add fp, sp, #0

r0 = 0
r3 = 0
fp =1040
sp =1032
lr = 36
pc = 4

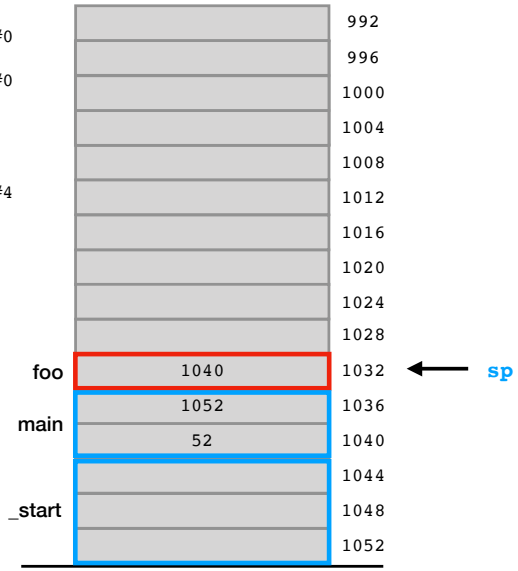


Class Activity

```
foo:
0  push  {fp}
4  add   fp, sp, #0
8  nop
12 add   sp, fp, #0
16 pop   {r11}
20 bx lr
main:
24 push  {fp, lr}
28 add   fp, sp, #4
32 bl foo
36 mov   r3, #0
40 mov   r0, r3
44 pop   {fp, pc}
_start:
…
48 bl    main
52 …
```

pc → 8 nop

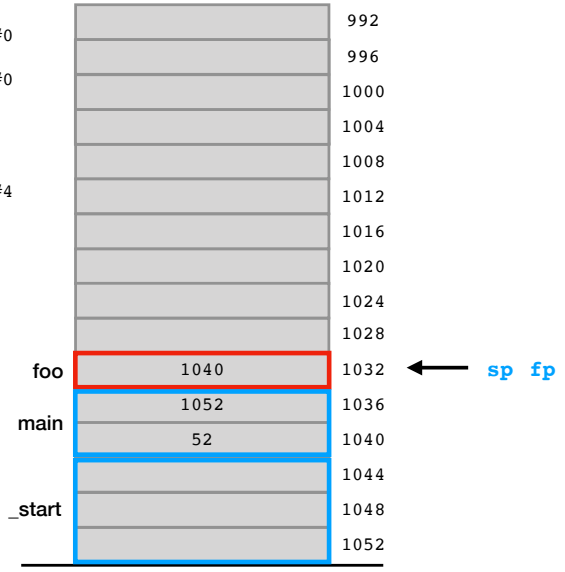
r0 = 0
r3 = 0
fp =1032
sp =1032
lr = 36
pc = 8



Class Activity

```
foo:
0  push  {fp}
4  add   fp, sp, #0
8  nop
12 add   sp, fp, #0
16 pop   {r11}
20 bx lr
main:
24 push  {fp, lr}
28 add   fp, sp, #4
32 bl foo
36 mov   r3, #0
40 mov   r0, r3
44 pop   {fp, pc}
_start:
…
48 bl    main
52 …
```

pc → 12

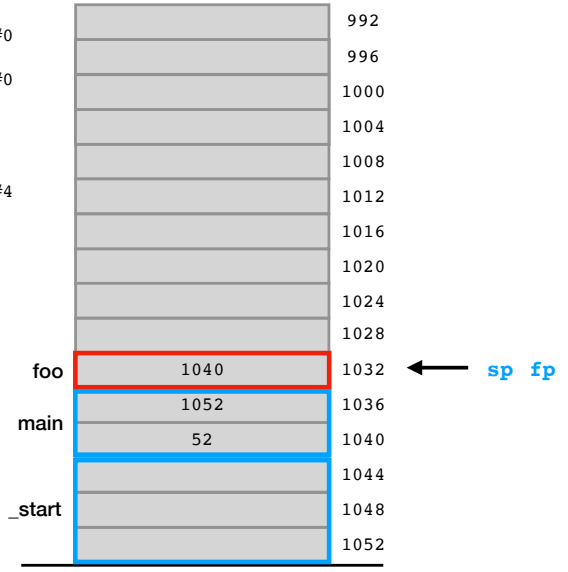


r0 = 0
r3 = 0
fp =1032
sp =1032
lr = 36
pc = 12

Class Activity

```
foo:
0  push  {fp}
4  add   fp, sp, #0
8  nop
12 add   sp, fp, #0
16 pop   {r11}
20 bx lr
main:
24 push  {fp, lr}
28 add   fp, sp, #4
32 bl foo
36 mov   r3, #0
40 mov   r0, r3
44 pop   {fp, pc}
_start:
…
48 bl    main
52 …
```

pc → 16

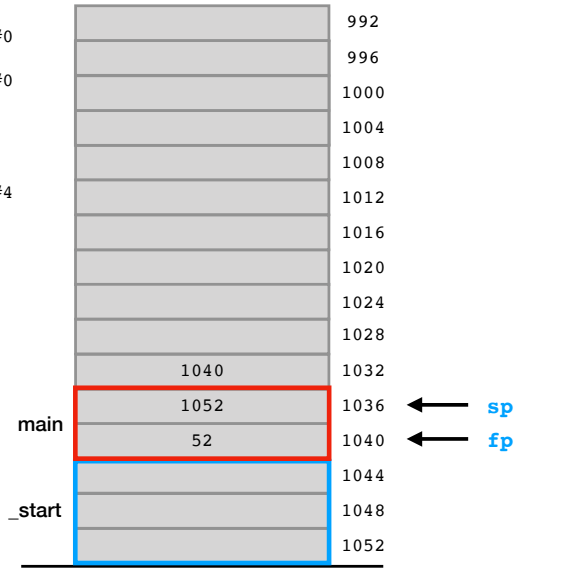


r0 = 0
r3 = 0
fp =1032
sp =1032
lr = 36
pc = 16

Class Activity

```
foo:
0  push  {fp}
4  add   fp, sp, #0
8  nop
12 add   sp, fp, #0
16 pop   {r11}
20 bx lr
main:
24 push  {fp, lr}
28 add   fp, sp, #4
32 bl foo
36 mov   r3, #0
40 mov   r0, r3
44 pop   {fp, pc}
_start:
…
48 bl    main
52 …
```

pc → 20

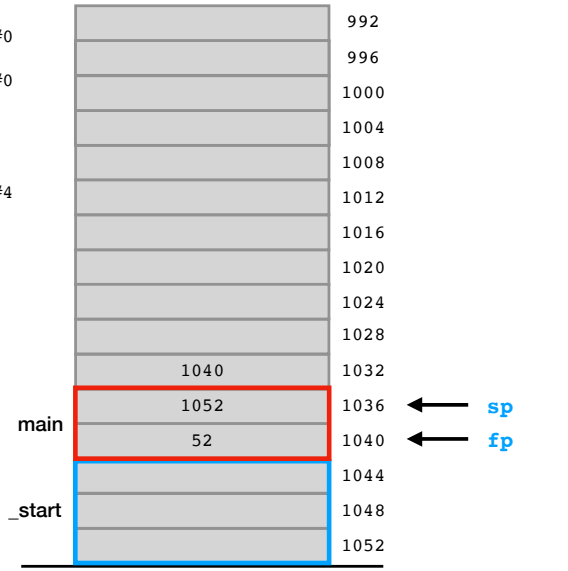


r0 = 0
r3 = 0
fp =1040
sp =1036
lr = 36
pc = 20

Class Activity

```
foo:
0  push  {fp}
4  add   fp, sp, #0
8  nop
12 add   sp, fp, #0
16 pop   {r11}
20 bx lr
main:
24 push  {fp, lr}
28 add   fp, sp, #4
32 bl foo
36 mov   r3, #0
40 mov   r0, r3
44 pop   {fp, pc}
_start:
…
48 bl    main
52 …
```

pc → 36



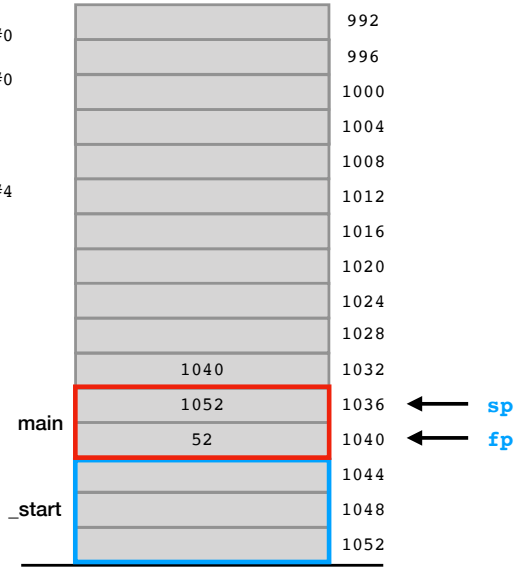
r0 = 0
r3 = 0
fp =1040
sp =1036
lr = 36
pc = 36

Class Activity

```
foo:
0  push  {fp}
4  add   fp, sp, #0
8  nop
12 add   sp, fp, #0
16 pop   {r11}
20 bx lr
main:
24 push  {fp, lr}
28 add   fp, sp, #4
32 bl foo
36 mov   r3, #0
40 mov   r0, r3
44 pop   {fp, pc}
_start:
…
48 bl    main
52 …
```

pc → 40

r0 = 0
r3 = 0
fp =1040
sp =1036
lr = 36
pc = 40

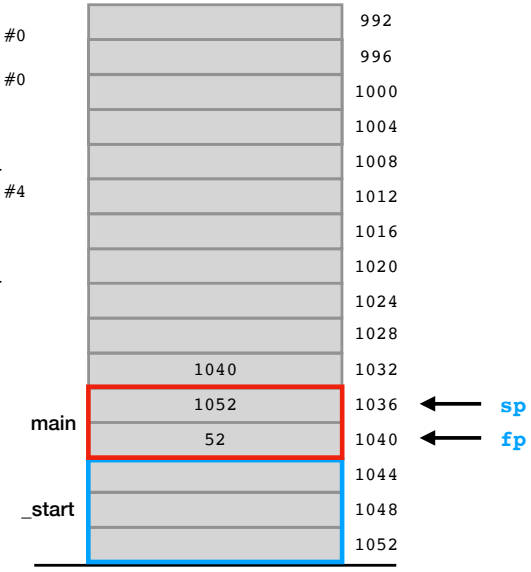


Class Activity

```
foo:
0  push  {fp}
4  add   fp, sp, #0
8  nop
12 add   sp, fp, #0
16 pop   {r11}
20 bx lr
main:
24 push  {fp, lr}
28 add   fp, sp, #4
32 bl foo
36 mov   r3, #0
40 mov   r0, r3
44 pop   {fp, pc}
_start:
…
48 bl    main
52 …
```

pc → 44

r0 = 0
r3 = 0
fp =1040
sp =1036
lr = 36
pc = 44



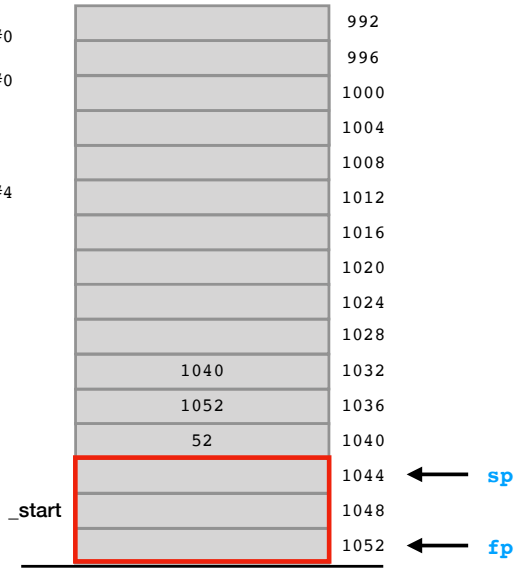
Class Activity

Everything is **back to where it started** except pc, which is advanced to 52.

```
foo:
0  push  {fp}
4  add   fp, sp, #0
8  nop
12 add   sp, fp, #0
16 pop   {r11}
20 bx lr
main:
24 push  {fp, lr}
28 add   fp, sp, #4
32 bl foo
36 mov   r3, #0
40 mov   r0, r3
44 pop   {fp, pc}
_start:
…
48 bl    main
52 …
```

pc → 52

r0 = 0
r3 = 0
fp =1052
sp =1044
lr = 36
pc = 52



Observations

- After a function is "torn down," **everything (that matters) is back where it was** before the call, **except** that `pc` is **advanced**.
- Notice that the `pc` saved on the stack is the **next instruction to run** after a return. All instructions except `b/bl/bx` (and a `pop` special case) advance `pc`.
- (You can't `push pc`!)
- Values are left on the stack. **Nobody cleans up!**
- Automatic variables: **only sort-of reclaimed**.
- Sometimes `gcc` **adds** `NOP` instructions. In general, these are added to align branches to 16-byte boundaries.

What are the parts of this program?

```
foo:
  push  {r11}
  add fp, sp, #0
  nop
  add sp, fp, #0
  pop {r11}
  bx  lr
main:
  push  {fp, lr}
  add fp, sp, #4
  bl  foo
  mov r3, #0
  mov r0, r3
  pop {fp, pc}
```

What are the parts of this program?

```
foo:
  push  {r11}
  add fp, sp, #0
  nop
  add sp, fp, #0
  pop {r11}
  bx  lr
main:
  push  {fp, lr}
  add fp, sp, #4
  bl  foo
  mov r3, #0
  mov r0, r3
  pop {fp, pc}
```

What are the parts of this program?

func. labels: where a function **starts**.

```
foo:
  push  {r11}
  add fp, sp, #0
  nop
  add sp, fp, #0
  pop {r11}
  bx  lr
main:
  push  {fp, lr}
  add fp, sp, #4
  bl  foo
  mov r3, #0
  mov r0, r3
  pop {fp, pc}
```

What are the parts of this program?

func. labels: where a function **starts**.

```
foo:
  push  {r11}
  add fp, sp, #0
  nop
  add sp, fp, #0
  pop {r11}
  bx  lr
main:
  push  {fp, lr}
  add fp, sp, #4
  bl  foo
  mov r3, #0
  mov r0, r3
  pop {fp, pc}
```

What are the parts of this program?

func. labels: where a function **starts**.

```
foo:
   push  {r11}
   add fp, sp, #0
   nop
   add sp, fp, #0
   pop {r11}
   bx  lr
main:
   push  {fp, lr}
   add fp, sp, #4
   bl  foo
   mov r3, #0
   mov r0, r3
   pop {fp, pc}
```

func. prologue: **callee** sets up stack for **itself**.

What are the parts of this program?

func. labels: where a function **starts**.

```
foo:
   push  {r11}
   add fp, sp, #0
   nop
   add sp, fp, #0
   pop {r11}
   bx  lr
main:
   push  {fp, lr}
   add fp, sp, #4
   bl  foo
   mov r3, #0
   mov r0, r3
   pop {fp, pc}
```

func. prologue: **callee** sets up stack for **itself**.

What are the parts of this program?

func. labels: where a function **starts**.

```
foo:
   push  {r11}
   add fp, sp, #0
   nop
   add sp, fp, #0
   pop {r11}
   bx  lr
main:
   push  {fp, lr}
   add fp, sp, #4
   bl  foo
   mov r3, #0
   mov r0, r3
   pop {fp, pc}
```

func. prologue: **callee** sets up stack for **itself**.

What are the parts of this program?

func. labels: where a function **starts**.

```
foo:
   push  {r11}
   add fp, sp, #0
   nop
   add sp, fp, #0
   pop {r11}
   bx  lr
main:
   push  {fp, lr}
   add fp, sp, #4
   bl  foo
   mov r3, #0
   mov r0, r3
   pop {fp, pc}
```

func. prologue: **callee** sets up stack for **itself**.

func. epilogue: **callee** restores stack & returns.

What are the parts of this program?

func. labels: where a function **starts**.

```
foo:
    push  {r11}
    add fp, sp, #0
    nop
    add sp, fp, #0
    pop {r11}
    bx  lr
main:
    push  {fp, lr}
    add fp, sp, #4
    bl  foo
    mov r3, #0
    mov r0, r3
    pop {fp, pc}
```

func. prologue: **callee** sets up stack for **itself**.

func. epilogue: **callee** restores stack & returns.

What are the parts of this program?

func. labels: where a function **starts**.


```
foo:
    push  {r11}
    add fp, sp, #0
    nop
    add sp, fp, #0
    pop {r11}
    bx  lr
main:
    push  {fp, lr}
    add fp, sp, #4
    bl  foo
    mov r3, #0
    mov r0, r3
    pop {fp, pc}
```


func. prologue: **callee** sets up stack for **itself**.

func. epilogue: **callee** restores stack & returns.

What are the parts of this program?

func. labels: where a function **starts**.


```
foo:
    push  {r11}
    add fp, sp, #0
    nop
    add sp, fp, #0
    pop {r11}
    bx  lr
main:
    push  {fp, lr}
    add fp, sp, #4
    bl  foo
    mov r3, #0
    mov r0, r3
    pop {fp, pc}
```


func. prologue: **callee** sets up stack for **itself**.

transfer of control: **caller** gives control to **callee**.

func. epilogue: **callee** restores stack & returns.

What are the parts of this program?

func. labels: where a function **starts**.


```
foo:
    push  {r11}
    add fp, sp, #0
    nop
    add sp, fp, #0
    pop {r11}
    bx  lr
main:
    push  {fp, lr}
    add fp, sp, #4
    bl  foo
    mov r3, #0
    mov r0, r3
    pop {fp, pc}
```


func. prologue: **callee** sets up stack for **itself**.

transfer of control: **caller** gives control to **callee**.

func. epilogue: **callee** restores stack & returns.

What are the parts of this program?

func. labels: where a function **starts**.

```
foo:
    push  {r11}
    add fp, sp, #0
    nop
    add sp, fp, #0
    pop {r11}
    bx  lr
main:
    push  {fp, lr}
    add fp, sp, #4
    bl  foo
    mov r3, #0
    mov r0, r3
    pop {fp, pc}
```

func. body: where **callee** does work (nothing here).

func. prologue: **callee** sets up stack for **itself**.

transfer of control: **caller** gives control to **callee**.

func. epilogue: **callee** restores stack & returns.

What are the parts of this program?

func. labels: where a function **starts**.

```
foo:
    push  {r11}
    add fp, sp, #0
    nop
    add sp, fp, #0
    pop {r11}
    bx  lr
main:
    push  {fp, lr}
    add fp, sp, #4
    bl  foo
    mov r3, #0
    mov r0, r3
    pop {fp, pc}
```

func. body: where **callee** does work (nothing here).

func. prologue: **callee** sets up stack for **itself**.

transfer of control: **caller** gives control to **callee**.

func. epilogue: **callee** restores stack & returns.

What are the parts of this program?

func. labels: where a function **starts**.

```
foo:
    push  {r11}
    add fp, sp, #0
    nop
    add sp, fp, #0
    pop {r11}
    bx  lr
main:
    push  {fp, lr}
    add fp, sp, #4
    bl  foo
    mov r3, #0
    mov r0, r3
    pop {fp, pc}
```

func. body: where **callee** does work (nothing here).

func. prologue: **callee** sets up stack for **itself**.

transfer of control: **caller** gives control to **callee**.

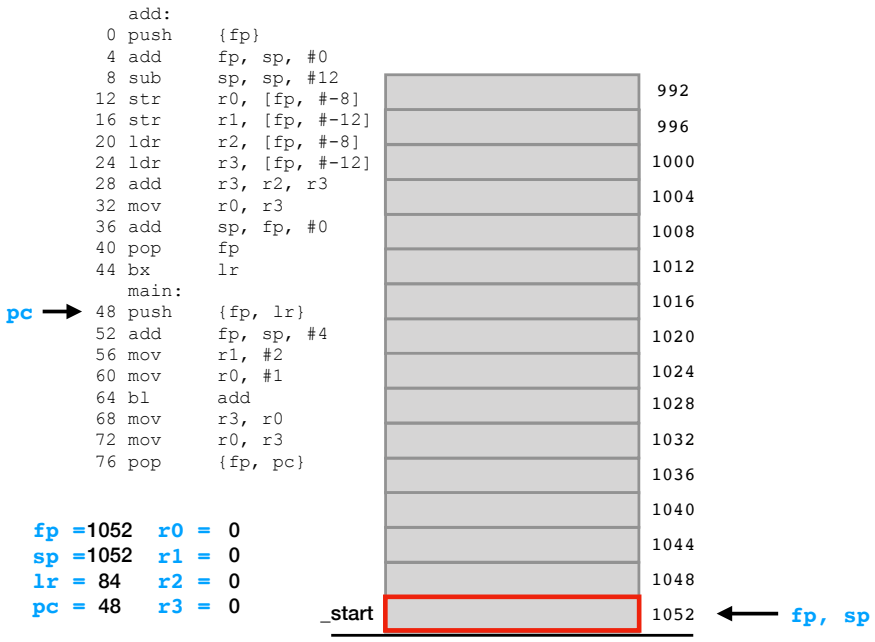
return value: **callee** prepares return value for **caller**.

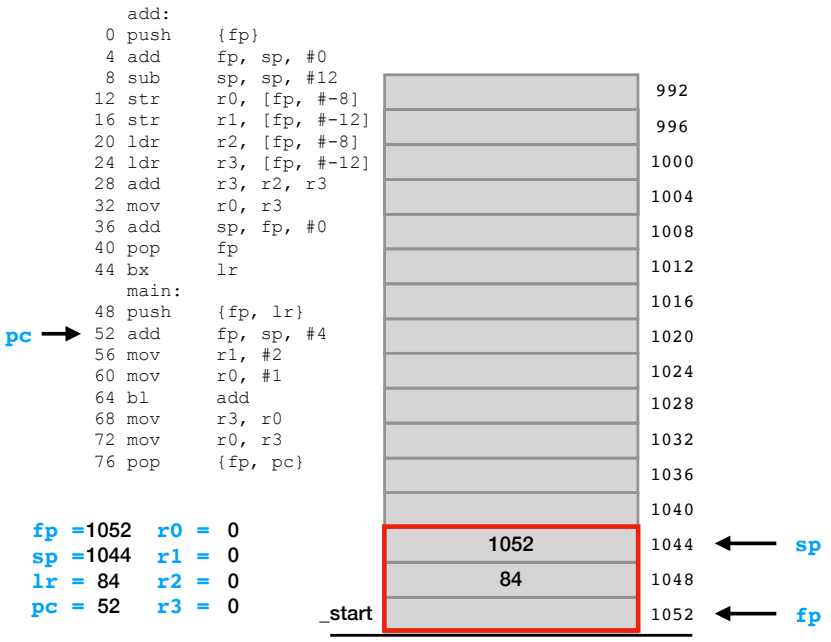
func. epilogue: **callee** restores stack & returns.

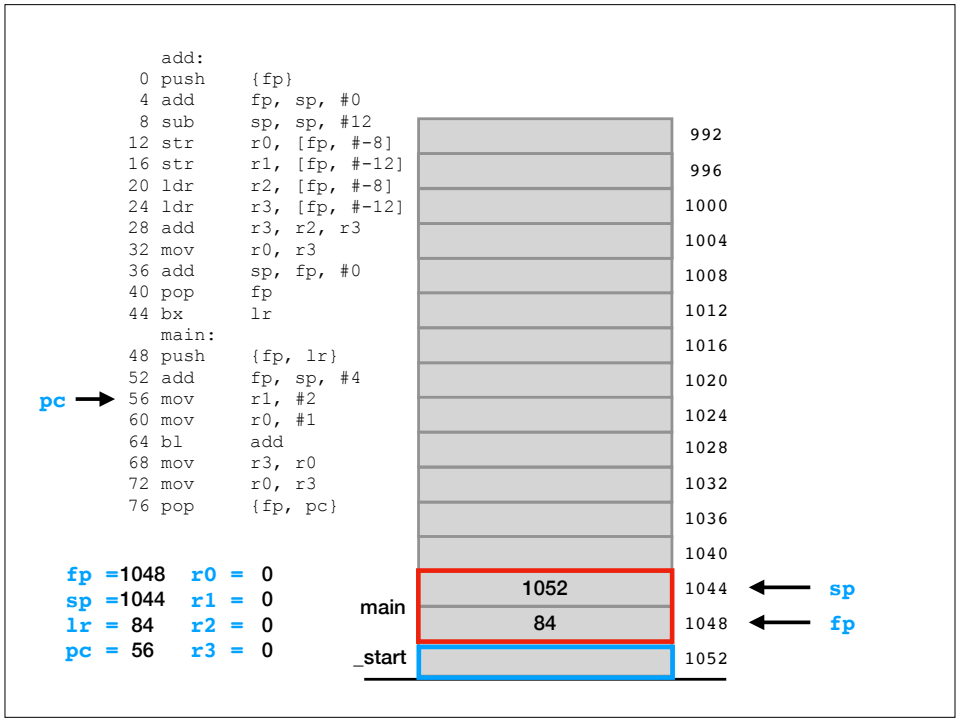
Arguments

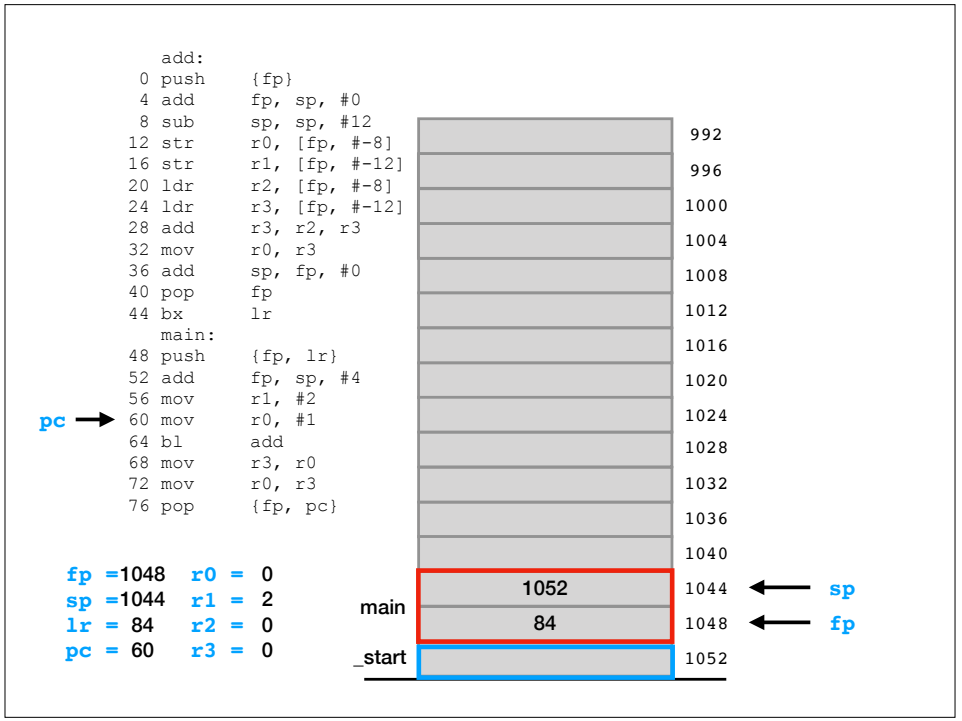
```c
int add(int a, int b) {
  return a + b;
}

int main() {
  return add(1, 2);
}
```

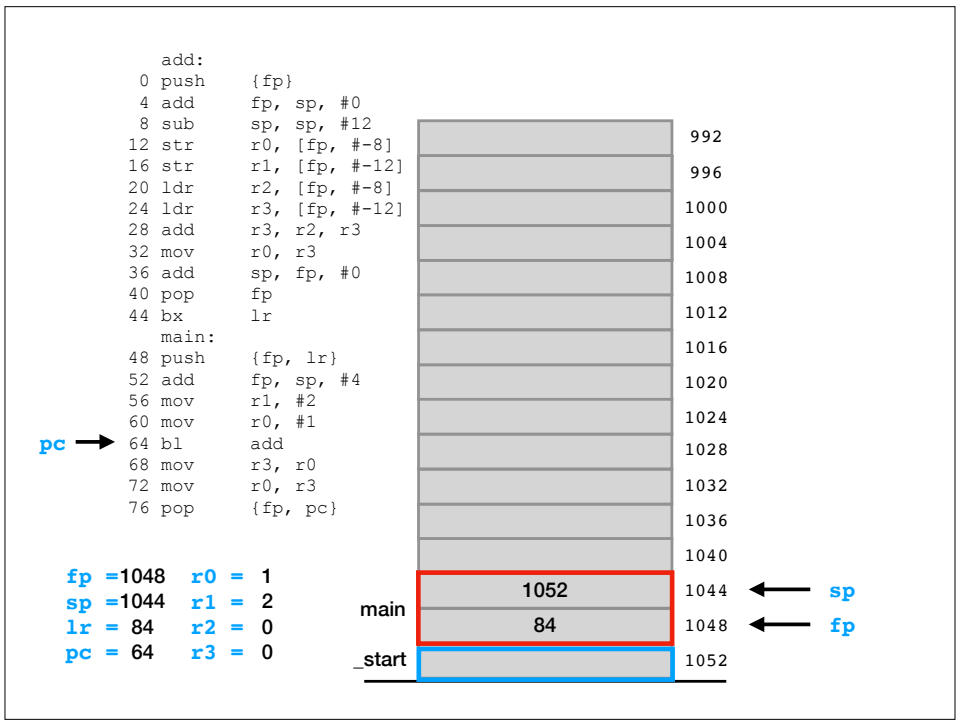
```
   add:
 0         push    {fp}
 4         add     fp, sp, #0
 8         sub     sp, sp, #12
12         str     r0, [fp, #-8]
16         str     r1, [fp, #-12]
20         ldr     r2, [fp, #-8]
24         ldr     r3, [fp, #-12]
28         add     r3, r2, r3
32         mov     r0, r3
36         add     sp, fp, #0
40         pop     fp
44         bx      lr
   main:
48         push    {fp, lr}
52         add     fp, sp, #4
56         mov     r1, #2
60         mov     r0, #1
64         bl      add
68         mov     r3, r0
72         mov     r0, r3
76         pop     {fp, pc}
```

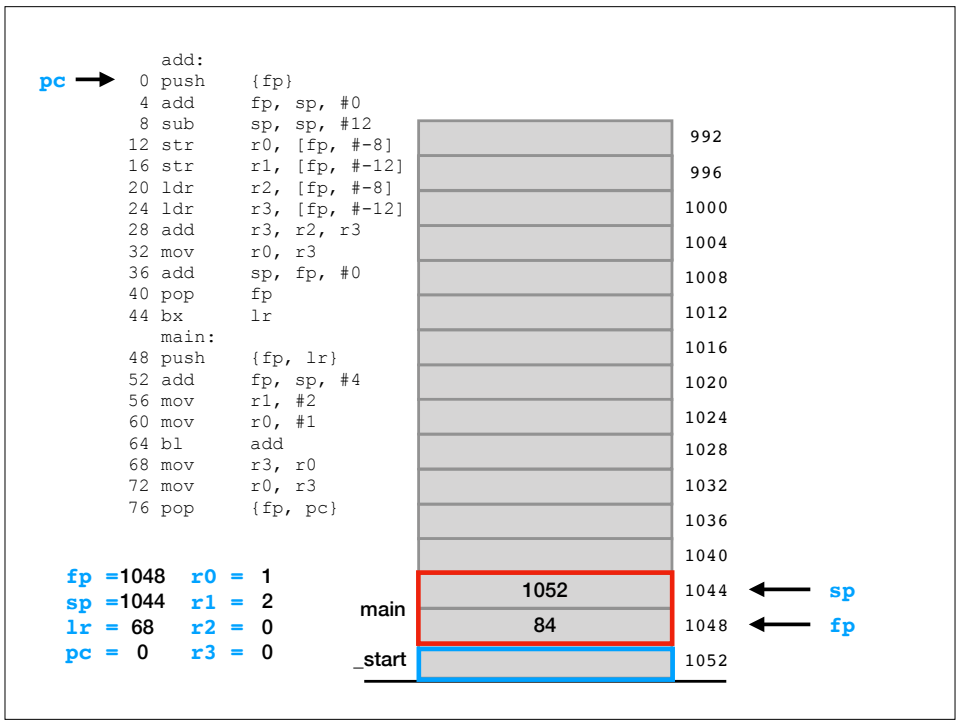


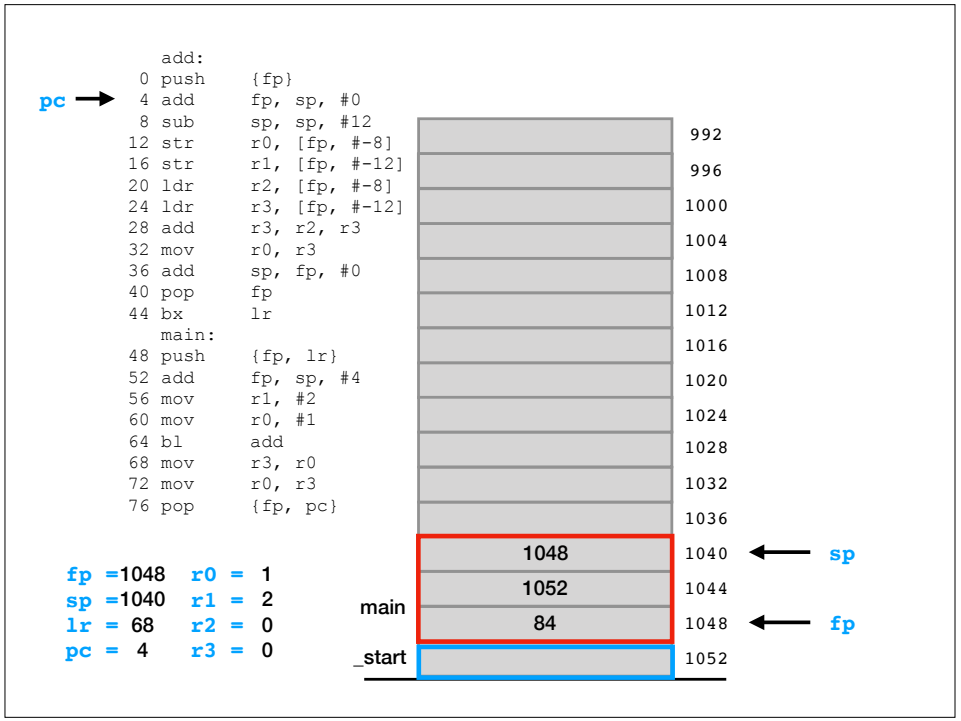


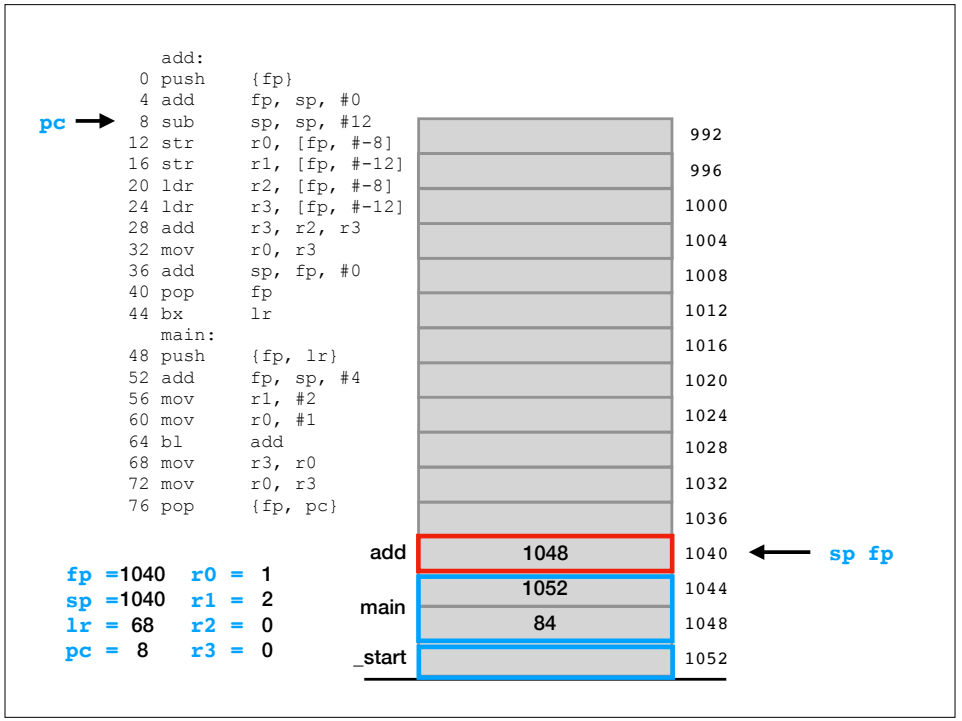

add:
0 push {fp}
4 add fp, sp, #0
8 sub sp, sp, #12
12 str r0, [fp, #-8]
16 str r1, [fp, #-12]
20 ldr r2, [fp, #-8]
24 ldr r3, [fp, #-12]
28 add r3, r2, r3
32 mov r0, r3
36 add sp, fp, #0
40 pop fp
44 bx lr
main:
48 push {fp, lr}
52 add fp, sp, #4
56 mov r1, #2
60 mov r0, #1
64 bl add
68 mov r3, r0
72 mov r0, r3
76 pop {fp, pc}
pc
fp =1048 r0 = 0
sp =1044 r1 = 0
lr = 84 r2 = 0
pc = 56 r3 = 0
992
996
1000
1004
1008
1012
1016
1020
1024
1028
1032
1036
1040
1052
84
1044
1048
1052
main
_start
sp
fp

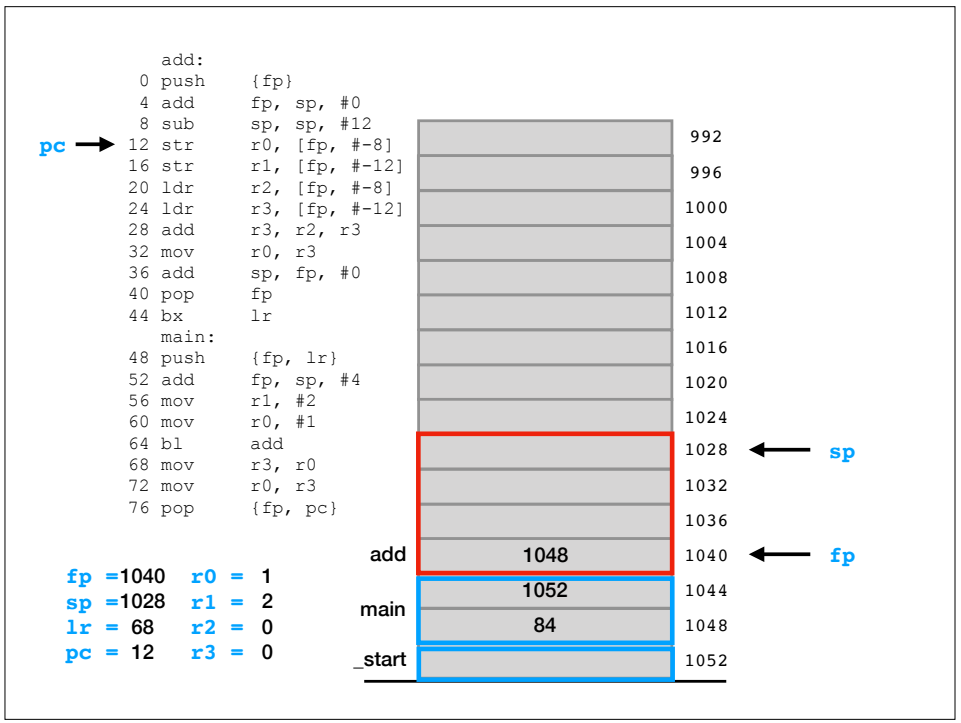

add:
0 push {fp}
4 add fp, sp, #0
8 sub sp, sp, #12
12 str r0, [fp, #-8]
16 str r1, [fp, #-12]
20 ldr r2, [fp, #-8]
24 ldr r3, [fp, #-12]
28 add r3, r2, r3
32 mov r0, r3
36 add sp, fp, #0
40 pop fp
44 bx lr
main:
48 push {fp, lr}
52 add fp, sp, #4
56 mov r1, #2
60 mov r0, #1
64 bl add
68 mov r3, r0
72 mov r0, r3
76 pop {fp, pc}
pc
fp =1048 r0 = 0
sp =1044 r1 = 2
lr = 84 r2 = 0
pc = 60 r3 = 0
992
996
1000
1004
1008
1012
1016
1020
1024
1028
1032
1036
1040
1052
84
1044
1048
1052
main
_start
sp
fp

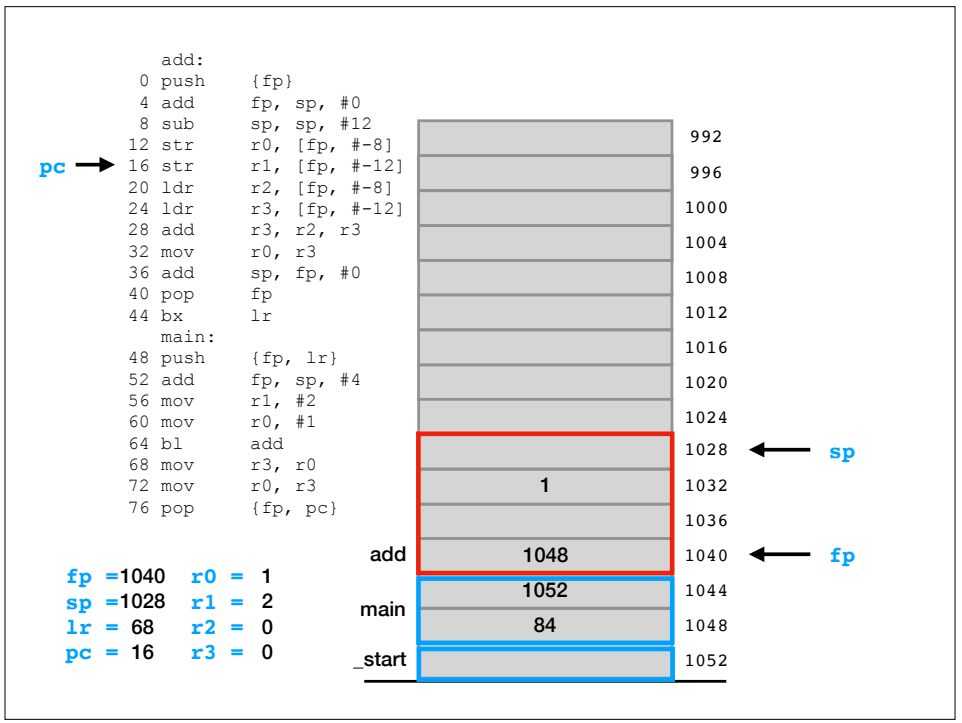

add:
0 push {fp}
4 add fp, sp, #0
8 sub sp, sp, #12
12 str r0, [fp, #-8]
16 str r1, [fp, #-12]
20 ldr r2, [fp, #-8]
24 ldr r3, [fp, #-12]
28 add r3, r2, r3
32 mov r0, r3
36 add sp, fp, #0
40 pop fp
44 bx lr
main:
48 push {fp, lr}
52 add fp, sp, #4
56 mov r1, #2
60 mov r0, #1
64 bl add
68 mov r3, r0
72 mov r0, r3
76 pop {fp, pc}
pc
fp =1048 r0 = 1
sp =1044 r1 = 2
lr = 84 r2 = 0
pc = 64 r3 = 0
992
996
1000
1004
1008
1012
1016
1020
1024
1028
1032
1036
1040
1052
84
1044
1048
1052
main
_start
sp
fp

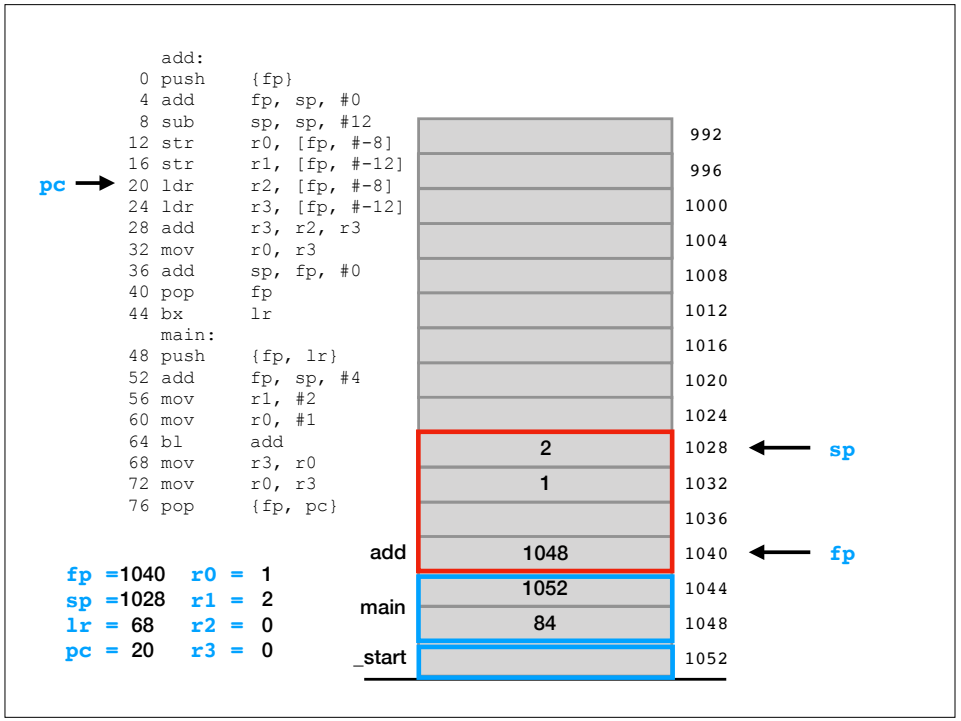

add:
0 push {fp}
4 add fp, sp, #0
8 sub sp, sp, #12
12 str r0, [fp, #-8]
16 str r1, [fp, #-12]
20 ldr r2, [fp, #-8]
24 ldr r3, [fp, #-12]
28 add r3, r2, r3
32 mov r0, r3
36 add sp, fp, #0
40 pop fp
44 bx lr
main:
48 push {fp, lr}
52 add fp, sp, #4
56 mov r1, #2
60 mov r0, #1
64 bl add
68 mov r3, r0
72 mov r0, r3
76 pop {fp, pc}
pc
fp =1048 r0 = 1
sp =1044 r1 = 2
lr = 68 r2 = 0
pc = 0 r3 = 0
992
996
1000
1004
1008
1012
1016
1020
1024
1028
1032
1036
1040
1052
84
1044
1048
1052
main
_start
sp
fp

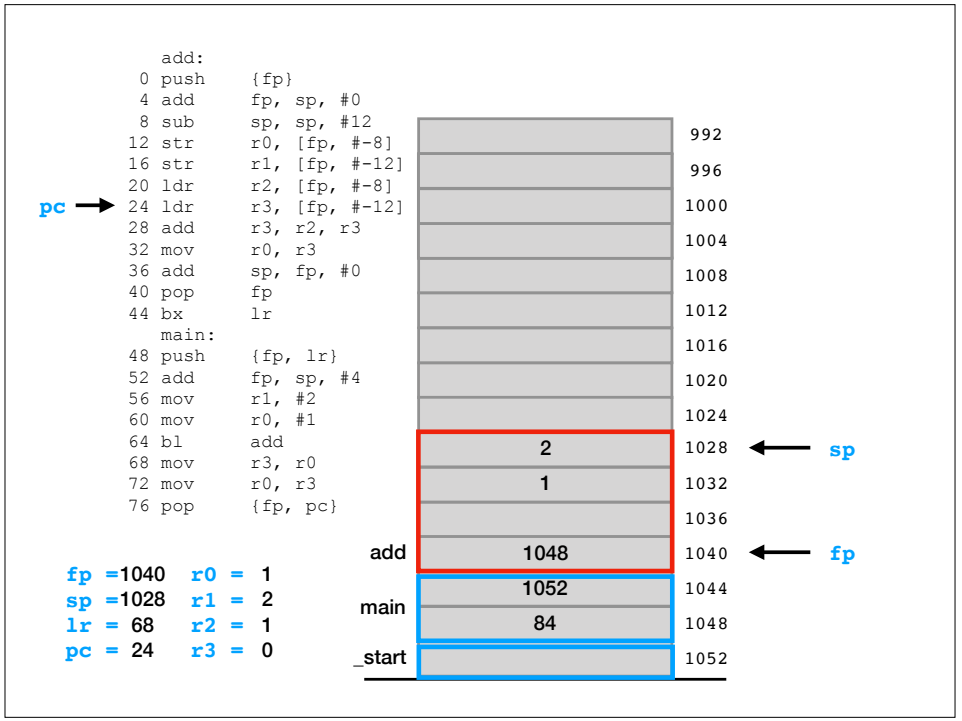

add:
0 push {fp}
4 add fp, sp, #0
8 sub sp, sp, #12
12 str r0, [fp, #-8]
16 str r1, [fp, #-12]
20 ldr r2, [fp, #-8]
24 ldr r3, [fp, #-12]
28 add r3, r2, r3
32 mov r0, r3
36 add sp, fp, #0
40 pop fp
44 bx lr
main:
48 push {fp, lr}
52 add fp, sp, #4
56 mov r1, #2
60 mov r0, #1
64 bl add
68 mov r3, r0
72 mov r0, r3
76 pop {fp, pc}
pc
fp =1048 r0 = 1
sp =1040 r1 = 2
lr = 68 r2 = 0
pc = 4 r3 = 0
992
996
1000
1004
1008
1012
1016
1020
1024
1028
1032
1036
1040
1044
1048
1052
1048
1052
84
main
_start
sp
fp

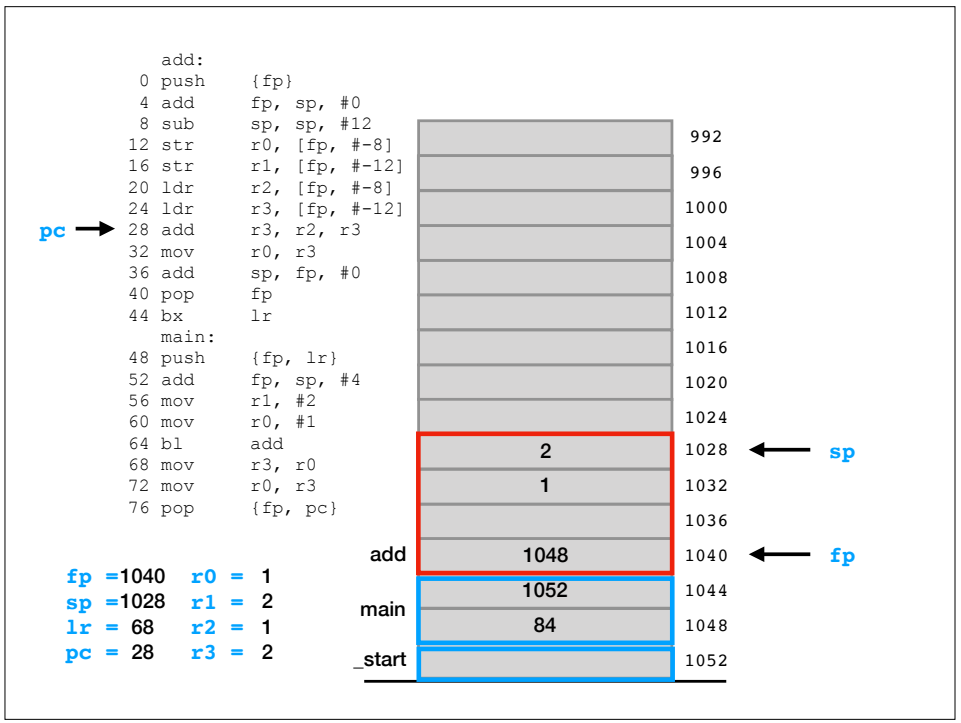

add:
0 push {fp}
4 add fp, sp, #0
8 sub sp, sp, #12
12 str r0, [fp, #-8]
16 str r1, [fp, #-12]
20 ldr r2, [fp, #-8]
24 ldr r3, [fp, #-12]
28 add r3, r2, r3
32 mov r0, r3
36 add sp, fp, #0
40 pop fp
44 bx lr
main:
48 push {fp, lr}
52 add fp, sp, #4
56 mov r1, #2
60 mov r0, #1
64 bl add
68 mov r3, r0
72 mov r0, r3
76 pop {fp, pc}
pc
fp =1040 r0 = 1
sp =1040 r1 = 2
lr = 68 r2 = 0
pc = 8 r3 = 0
992
996
1000
1004
1008
1012
1016
1020
1024
1028
1032
1036
1040
1044
1048
1052
add
1048
1052
84
main
_start
sp fp

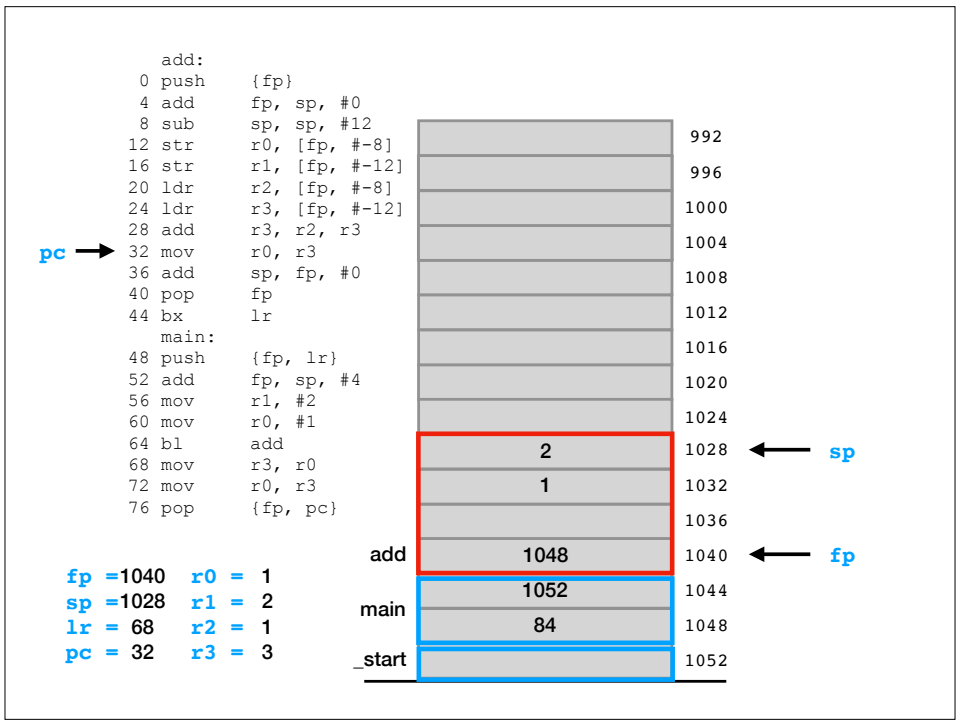

add:
0 push {fp}
4 add fp, sp, #0
8 sub sp, sp, #12
12 str r0, [fp, #-8]
16 str r1, [fp, #-12]
20 ldr r2, [fp, #-8]
24 ldr r3, [fp, #-12]
28 add r3, r2, r3
32 mov r0, r3
36 add sp, fp, #0
40 pop fp
44 bx lr
main:
48 push {fp, lr}
52 add fp, sp, #4
56 mov r1, #2
60 mov r0, #1
64 bl add
68 mov r3, r0
72 mov r0, r3
76 pop {fp, pc}
pc
fp =1040 r0 = 1
sp =1028 r1 = 2
lr = 68 r2 = 0
pc = 12 r3 = 0
992
996
1000
1004
1008
1012
1016
1020
1024
1028
1032
1036
1040
1044
1048
1052
add
1048
1052
84
main
_start
sp
fp

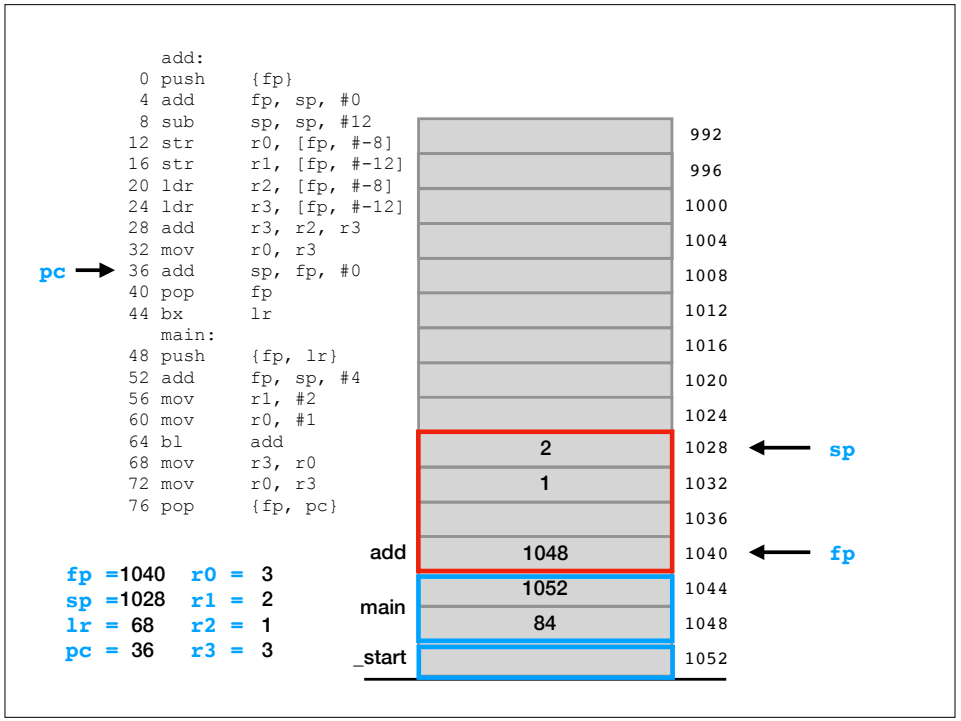

add:
0 push {fp}
4 add fp, sp, #0
8 sub sp, sp, #12
12 str r0, [fp, #-8]
16 str r1, [fp, #-12]
20 ldr r2, [fp, #-8]
24 ldr r3, [fp, #-12]
28 add r3, r2, r3
32 mov r0, r3
36 add sp, fp, #0
40 pop fp
44 bx lr
main:
48 push {fp, lr}
52 add fp, sp, #4
56 mov r1, #2
60 mov r0, #1
64 bl add
68 mov r3, r0
72 mov r0, r3
76 pop {fp, pc}
pc
fp =1040 r0 = 1
sp =1028 r1 = 2
lr = 68 r2 = 0
pc = 16 r3 = 0
992
996
1000
1004
1008
1012
1016
1020
1024
1028
1032
1036
1040
1044
1048
1052
1
add
1048
1052
84
main
_start
sp
fp

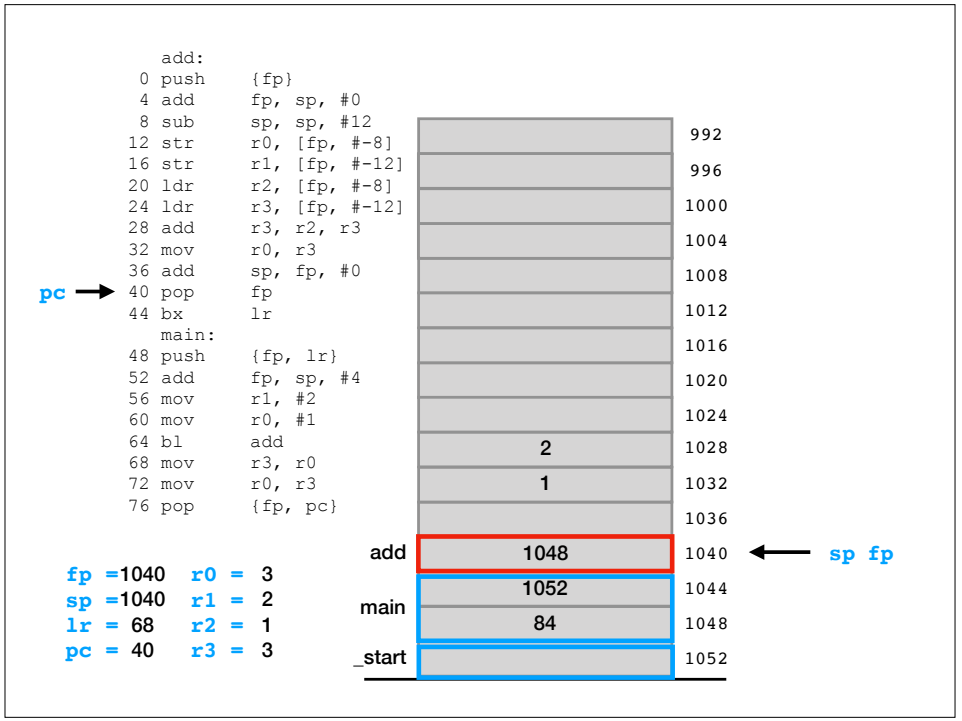

add:
0 push {fp}
4 add fp, sp, #0
8 sub sp, sp, #12
12 str r0, [fp, #-8]
16 str r1, [fp, #-12]
20 ldr r2, [fp, #-8]
24 ldr r3, [fp, #-12]
28 add r3, r2, r3
32 mov r0, r3
36 add sp, fp, #0
40 pop fp
44 bx lr
main:
48 push {fp, lr}
52 add fp, sp, #4
56 mov r1, #2
60 mov r0, #1
64 bl add
68 mov r3, r0
72 mov r0, r3
76 pop {fp, pc}
pc
fp =1040 r0 = 1
sp =1028 r1 = 2
lr = 68 r2 = 0
pc = 20 r3 = 0
992
996
1000
1004
1008
1012
1016
1020
1024
2
1028
sp
1
1032
1036
add
1048
1040
fp
1052
1044
main
84
1048
_start
1052

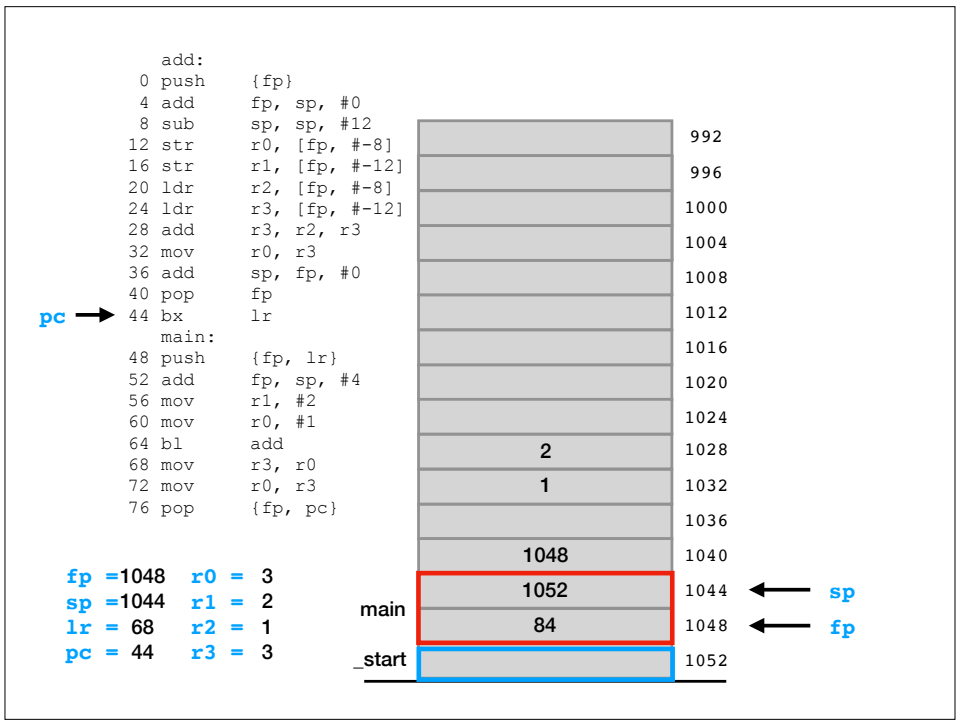

add:
0 push {fp}
4 add fp, sp, #0
8 sub sp, sp, #12
12 str r0, [fp, #-8]
16 str r1, [fp, #-12]
20 ldr r2, [fp, #-8]
24 ldr r3, [fp, #-12]
28 add r3, r2, r3
32 mov r0, r3
36 add sp, fp, #0
40 pop fp
44 bx lr
main:
48 push {fp, lr}
52 add fp, sp, #4
56 mov r1, #2
60 mov r0, #1
64 bl add
68 mov r3, r0
72 mov r0, r3
76 pop {fp, pc}
pc
fp =1040 r0 = 1
sp =1028 r1 = 2
lr = 68 r2 = 1
pc = 24 r3 = 0
992
996
1000
1004
1008
1012
1016
1020
1024
2
1028
sp
1
1032
1036
add
1048
1040
fp
1052
1044
main
84
1048
_start
1052

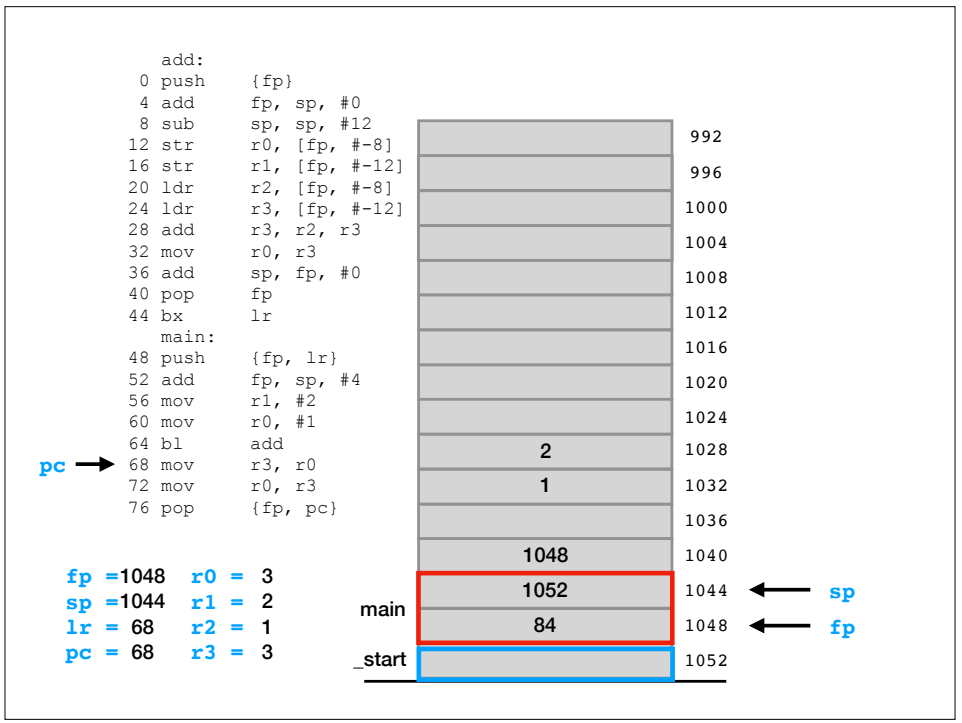

add:
0 push {fp}
4 add fp, sp, #0
8 sub sp, sp, #12
12 str r0, [fp, #-8]
16 str r1, [fp, #-12]
20 ldr r2, [fp, #-8]
24 ldr r3, [fp, #-12]
28 add r3, r2, r3
32 mov r0, r3
36 add sp, fp, #0
40 pop fp
44 bx lr
main:
48 push {fp, lr}
52 add fp, sp, #4
56 mov r1, #2
60 mov r0, #1
64 bl add
68 mov r3, r0
72 mov r0, r3
76 pop {fp, pc}
pc
fp =1040 r0 = 1
sp =1028 r1 = 2
lr = 68 r2 = 1
pc = 28 r3 = 2
992
996
1000
1004
1008
1012
1016
1020
1024
2
1028
sp
1
1032
1036
add
1048
1040
fp
1052
1044
main
84
1048
_start
1052

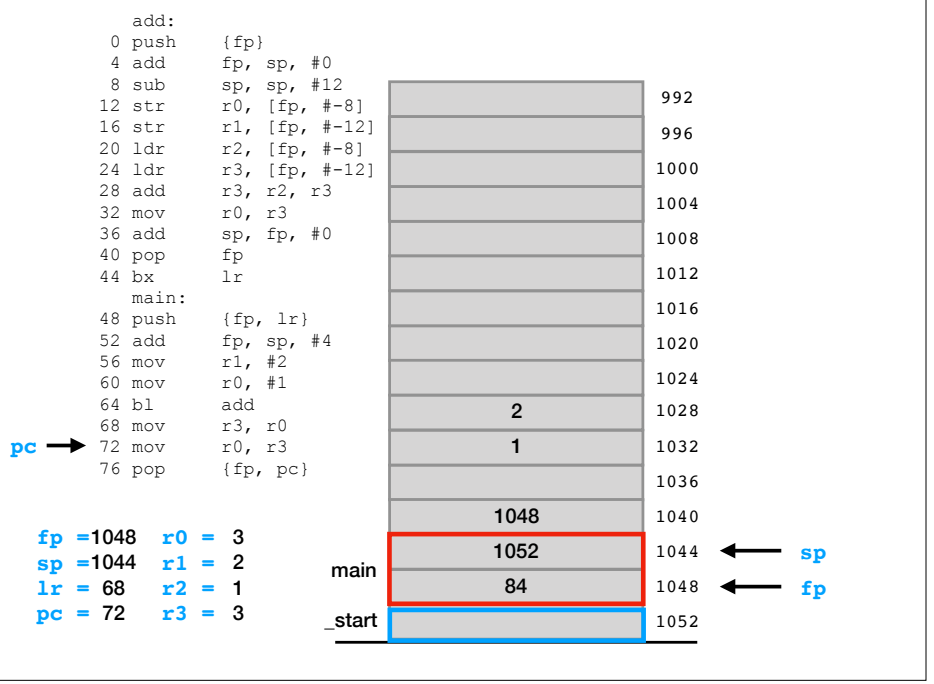

add:
0 push {fp}
4 add fp, sp, #0
8 sub sp, sp, #12
12 str r0, [fp, #-8]
16 str r1, [fp, #-12]
20 ldr r2, [fp, #-8]
24 ldr r3, [fp, #-12]
28 add r3, r2, r3
32 mov r0, r3
36 add sp, fp, #0
40 pop fp
44 bx lr
main:
48 push {fp, lr}
52 add fp, sp, #4
56 mov r1, #2
60 mov r0, #1
64 bl add
68 mov r3, r0
72 mov r0, r3
76 pop {fp, pc}
pc
fp =1040 r0 = 1
sp =1028 r1 = 2
lr = 68 r2 = 1
pc = 32 r3 = 3
992
996
1000
1004
1008
1012
1016
1020
1024
2
1028
sp
1
1032
1036
add
1048
1040
fp
1052
1044
main
84
1048
_start
1052

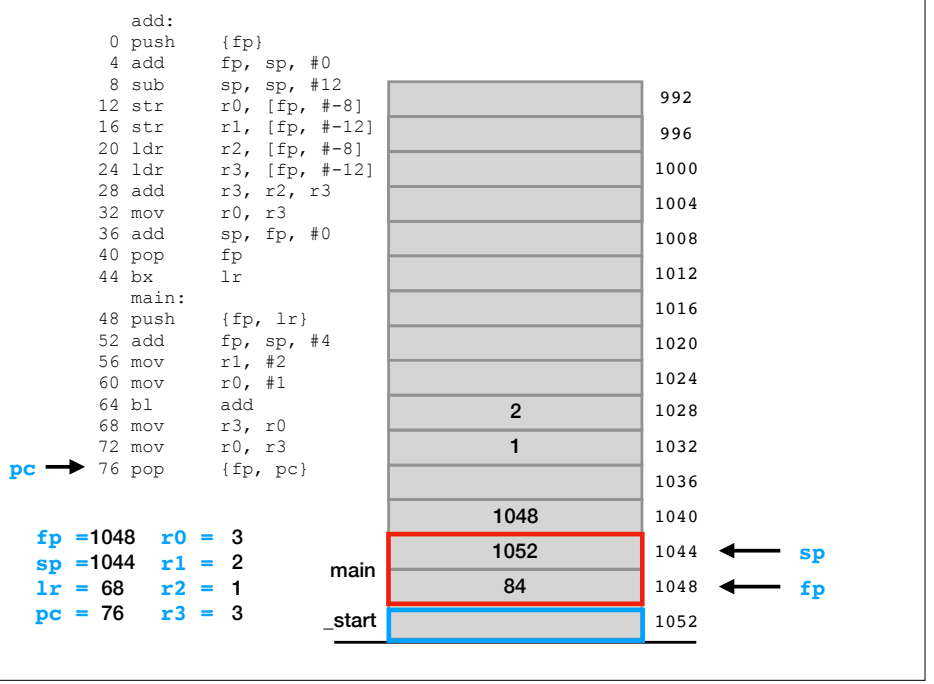

add:
0 push {fp}
4 add fp, sp, #0
8 sub sp, sp, #12
12 str r0, [fp, #-8]
16 str r1, [fp, #-12]
20 ldr r2, [fp, #-8]
24 ldr r3, [fp, #-12]
28 add r3, r2, r3
32 mov r0, r3
36 add sp, fp, #0
40 pop fp
44 bx lr
main:
48 push {fp, lr}
52 add fp, sp, #4
56 mov r1, #2
60 mov r0, #1
64 bl add
68 mov r3, r0
72 mov r0, r3
76 pop {fp, pc}
pc
fp =1040 r0 = 3
sp =1028 r1 = 2
lr = 68 r2 = 1
pc = 36 r3 = 3
992
996
1000
1004
1008
1012
1016
1020
1024
2
1028
sp
1
1032
1036
add
1048
1040
fp
1052
1044
main
84
1048
_start
1052

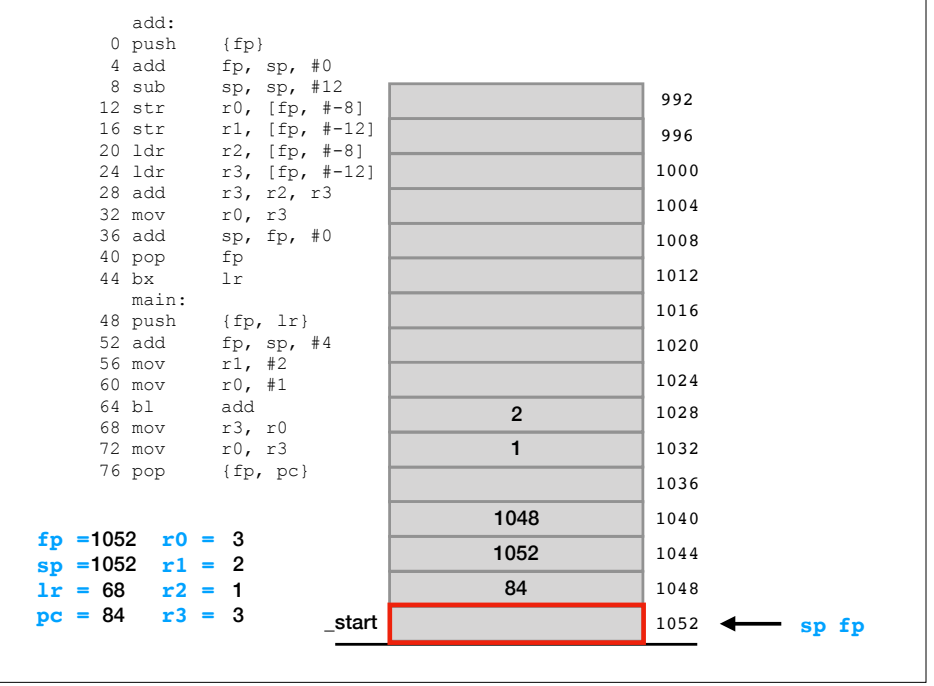

add:
0 push {fp}
4 add fp, sp, #0
8 sub sp, sp, #12
12 str r0, [fp, #-8]
16 str r1, [fp, #-12]
20 ldr r2, [fp, #-8]
24 ldr r3, [fp, #-12]
28 add r3, r2, r3
32 mov r0, r3
36 add sp, fp, #0
40 pop fp
44 bx lr
main:
48 push {fp, lr}
52 add fp, sp, #4
56 mov r1, #2
60 mov r0, #1
64 bl add
68 mov r3, r0
72 mov r0, r3
76 pop {fp, pc}
pc
fp =1040 r0 = 3
sp =1040 r1 = 2
lr = 68 r2 = 1
pc = 40 r3 = 3
992
996
1000
1004
1008
1012
1016
1020
1024
2
1028
1
1032
1036
add
1048
1040
sp fp
1052
1044
main
84
1048
_start
1052


add:
0 push {fp}
4 add fp, sp, #0
8 sub sp, sp, #12
12 str r0, [fp, #-8]
16 str r1, [fp, #-12]
20 ldr r2, [fp, #-8]
24 ldr r3, [fp, #-12]
28 add r3, r2, r3
32 mov r0, r3
36 add sp, fp, #0
40 pop fp
44 bx lr
main:
48 push {fp, lr}
52 add fp, sp, #4
56 mov r1, #2
60 mov r0, #1
64 bl add
68 mov r3, r0
72 mov r0, r3
76 pop {fp, pc}
pc
fp =1048 r0 = 3
sp =1044 r1 = 2
lr = 68 r2 = 1
pc = 44 r3 = 3
992
996
1000
1004
1008
1012
1016
1020
1024
2
1028
1
1032
1036
1048
1040
1052
1044
sp
main
84
1048
fp
_start
1052


add:
0 push {fp}
4 add fp, sp, #0
8 sub sp, sp, #12
12 str r0, [fp, #-8]
16 str r1, [fp, #-12]
20 ldr r2, [fp, #-8]
24 ldr r3, [fp, #-12]
28 add r3, r2, r3
32 mov r0, r3
36 add sp, fp, #0
40 pop fp
44 bx lr
main:
48 push {fp, lr}
52 add fp, sp, #4
56 mov r1, #2
60 mov r0, #1
64 bl add
68 mov r3, r0
72 mov r0, r3
76 pop {fp, pc}
pc
fp =1048 r0 = 3
sp =1044 r1 = 2
lr = 68 r2 = 1
pc = 68 r3 = 3
992
996
1000
1004
1008
1012
1016
1020
1024
2
1028
1
1032
1036
1048
1040
1052
1044
sp
main
84
1048
fp
_start
1052







Lab 3 walkthrough

Recap & Next Class

Today we learned:

How argument passing works

Next class:

globalthermonuclearwar

and other string vulnerabilities