Topic 2: (Part B)

Microprocessors and Memory Hierarchy in Microprogrammed Embedded Systems

Advanced Use of the stack in Assembly:

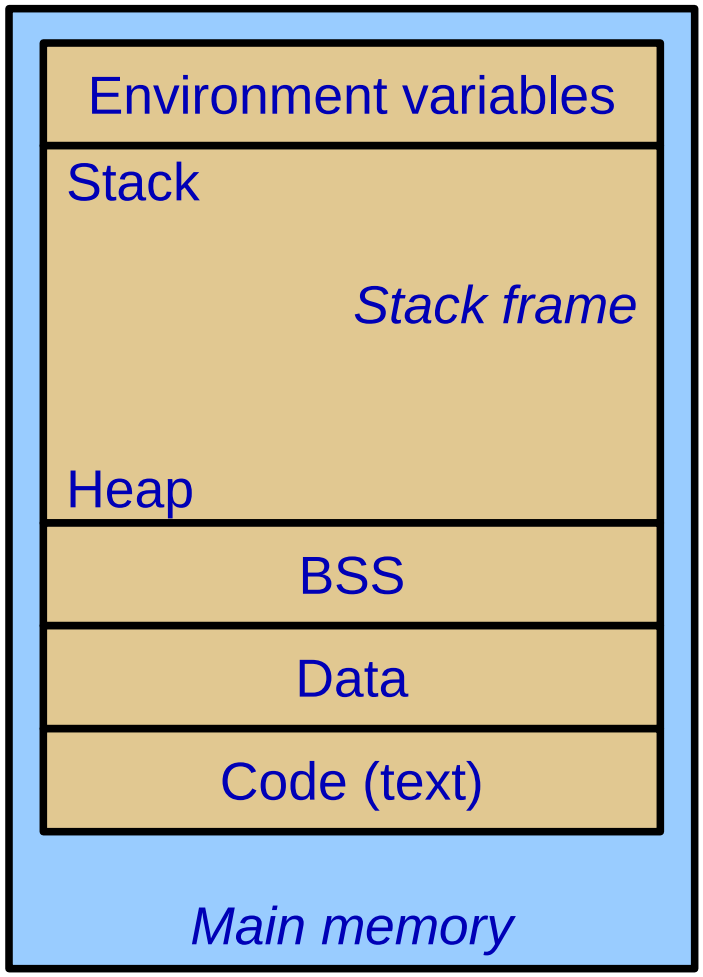
ARM/Thumb Procedure Call Standard

Systèmes Embarqués Microprogrammés

Prof. David Atienza Alonso, SEL-STI

Review: Memory regions in an ARM program

- A running program has 3 regions (both in C or in assembly)
  - Code
    - Code or instructions (text)
  - Data
    - Global data
    - BSS: data initialized to 0
  - Stack frame
    - Stack: automatic variables
    - Heap: explicitly managed dynamic variable



How parameters are passed in ARM Example: C program with a function

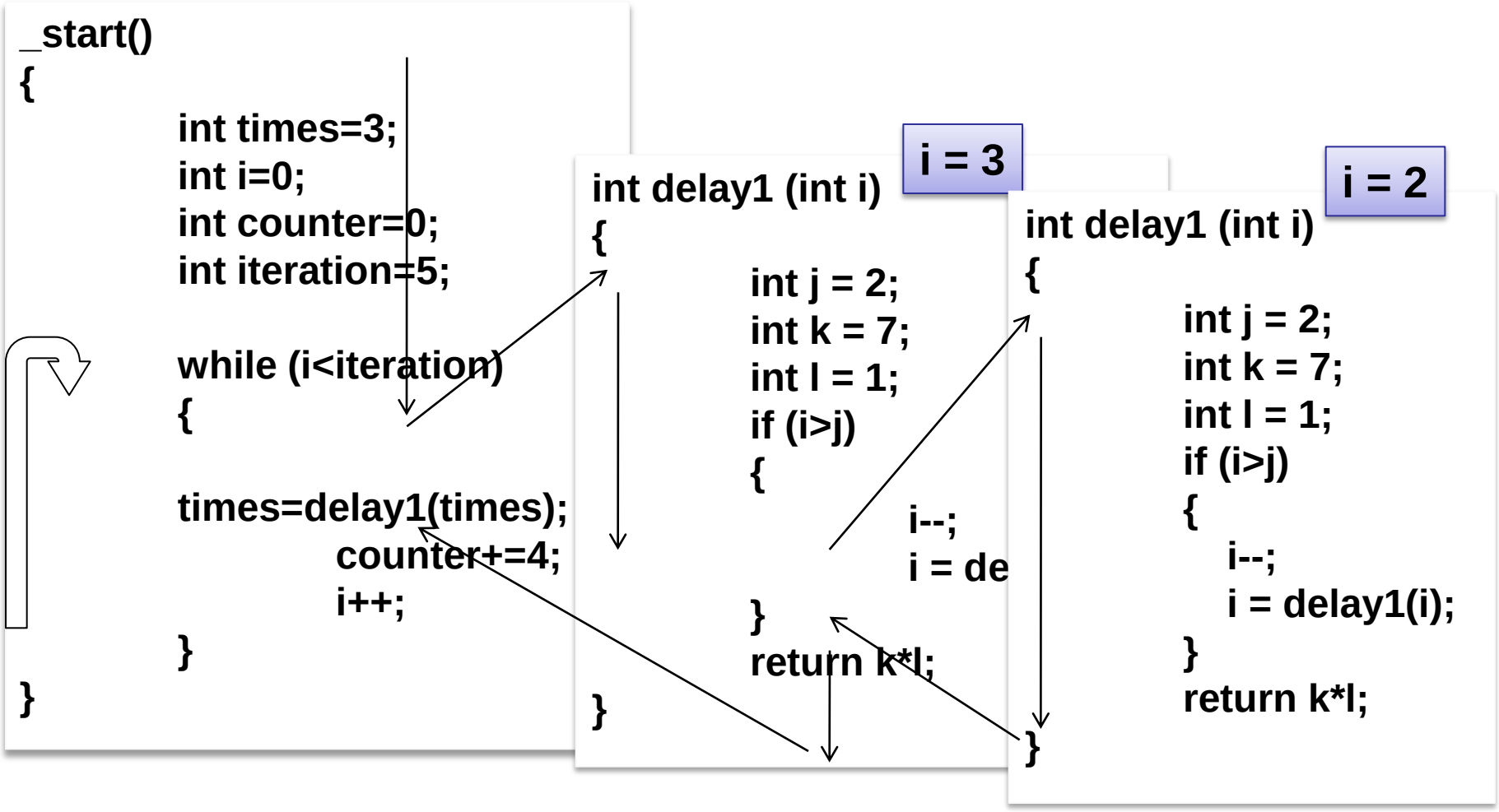
```
extern int delay1(int);
_start()
{
        int times=3;
        int i=0;
        int counter=0;
        int iteration=5;

        while (i<iteration)
        {

        times=delay1(times);
                counter+=4;
                i++;
        }
}
```

```
int delay1 (int i)
{
        int j = 2;
        int k = 7;
        int l = 1;
        if (i>j)
        {
                i--;
                i = delay1(i);
        }
        return k*l;
}
```

C Program flow: Passing arguments between functions



ARM Thumb Procedure Call Standard (ATPCS): sharing data between languages

- Each register has a purpose in the ARM microprocessors

Reg ARM	Name ATPCS	Description
R0-R3	a1 - a4	Parameters/results/register 1-4
R4-R6	v1 - v3	Register 1-3 of local variable
R7	v4 or wr	Register 4 of local variable, thumb-state work register
R8	v5	Register 5 of local variable
R9	v6 or sb	Reg. 6 of local variable, static base in shared library
R10	v7 or sl	Reg. 7 of local variable, stack limit pointer
R11	v8 or fp	Reg. 8 of local variable, ARM-state frame pointer
R12	ip	Temporal register for subroutine
R13	sp	Stack pointer
R14	lr	Link register
R15	pc	Program counter

Manage the stack

```
_start()
{
        int times=3;
        int i=0;
        int counter=0;
        int iteration=5;

        while (i<iteration)
        {

        times=delay1(time
                counter+=
                i++;
        }
}
```

```
mov     ip, sp
stmdb   sp!, {fp, ip, lr,
pc}
sub     fp, ip, #4
sub     sp, sp, #16
mov     r3, #3
str     r3, [fp, -#16]
mov     r3, #0
str     r3, [fp, -#20]
str     r3, [fp, -#24]
mov     r3, #5
str     r3, [fp, -#28]
ldr     r2, [fp, -#20]
ldr     r3, [fp, -#28]
```

SP=0x01000

IP=0x01000

0x00fb4	
0x00fb8	
0x00fbc	
0x00fc0	
0x00fc4	
0x00fc8	
0x00fcc	
0x00fd0	
0x00fd4	
0x00fd8	
0x00fdc	
0x00fe0	
0x00fe4	
0x00fe8	
0x00fec	
0x00ff0	
0x00ff4	
0x00ff8	
0x00ffc	
0x01000	

Manage the stack

```
_start()
{
        int times=3;
        int i=0;
        int counter=0;
        int iteration=5;

        while (i<iteration)
        {

        times=delay1(time
                counter+=
                i++;
        }
}
```

```
mov     ip, sp
stmdb   sp!, {fp, ip, lr, pc}
sub     fp, ip, #4
sub     sp, sp, #16
mov     r3, #3
str     r3, [fp, -#16]
mov     r3, #0
str     r3, [fp, -#20]
str     r3, [fp, -#24]
mov     r3, #5
str     r3, [fp, -#28]
ldr     r2, [fp, -#20]
ldr     r3, [fp, -#28]
```

0x00fb4	
0x00fb8	
0x00fbc	
0x00fc0	
0x00fc4	
0x00fc8	
0x00fcc	
0x00fd0	
0x00fd4	
0x00fd8	
0x00fdc	
0x00fe0	
0x00fe4	
0x00fe8	
0x00fec	
0x00ff0	0x0000 (FP)
0x00ff4	0x1000 (IP)
0x00ff8	0x0000 (LR)
0x00ffc	0x8010 (PC)
0x01000	

SP=0x00ff0

IP=0x01000

Manage the stack

```
_start()
{
        int times=3;
        int i=0;
        int counter=0;
        int iteration=5;

        while (i<iteration)
        {

        times=delay1(time
                counter+=
                i++;
        }
}
```

```
mov     ip, sp
stmdb   sp!, {fp, ip, lr, pc}
sub     fp, ip, #4
sub     sp, sp, #16
mov     r3, #3
str     r3, [fp, -#16]
mov     r3, #0
str     r3, [fp, -#20]
str     r3, [fp, -#24]
mov     r3, #5
str     r3, [fp, -#28]
ldr     r2, [fp, -#20]
ldr     r3, [fp, -#28]
```

0x00fb4	
0x00fb8	
0x00fbc	
0x00fc0	
0x00fc4	
0x00fc8	
0x00fcc	
0x00fd0	
0x00fd4	
0x00fd8	
0x00fdc	
0x00fe0	
0x00fe4	
0x00fe8	
0x00fec	
0x00ff0	0x0000 (FP)
0x00ff4	0x1000 (IP)
0x00ff8	0x0000 (LR)
0x00ffc	0x8010 (PC)
0x01000	

SP=0x00ff0

IP=0x01000

FP=0x00ffc

Manage the stack

```
_start()
{
        int times=3;
        int i=0;
        int counter=0;
        int iteration=5;

        while (i<iteration)
        {

        times=delay1(time
                counter+=
                i++;
        }
}
```

Leave space for 4 integer variables

```
mov     ip, sp
stmdb   sp!, {fp, ip, lr, pc}
sub     fp, ip, #4
sub     sp, sp, #16
mov     r3, #3
str     r3, [fp, -#16]
mov     r3, #0
str     r3, [fp, -#20]
str     r3, [fp, -#24]
mov     r3, #5
str     r3, [fp, -#28]
ldr     r2, [fp, -#20]
ldr     r3, [fp, -#28]
```

0x00fb4	
0x00fb8	
0x00fbc	
0x00fc0	
0x00fc4	
0x00fc8	
0x00fcc	
0x00fd0	
0x00fd4	
0x00fd8	
0x00fdc	
0x00fe0	
0x00fe4	
0x00fe8	
0x00fec	
0x00ff0	0x0000 (FP)
0x00ff4	0x1000 (IP)
0x00ff8	0x0000 (LR)
0x00ffc	0x8010 (PC)
0x01000	

SP=0x00fe0

IP=0x01000

FP=0x00ffc

Manage the stack

```
_start()
{
        int times=3;
        int i=0;
        int counter=0;
        int iteration=5;

        while (i<iteration)
        {

        times=delay1(time
                counter+=
                i++;
        }
}
```

Leave space for 4 integer variables

```
mov     ip, sp
stmdb   sp!, {fp, ip, lr, pc}
sub     fp, ip, #4
sub     sp, sp, #16
mov     r3, #3
str     r3, [fp, -#16]
mov     r3, #0
str     r3, [fp, -#20]
str     r3, [fp, -#24]
mov     r3, #5
str     r3, [fp, -#28]
ldr     r2, [fp, -#20]
ldr     r3, [fp, -#28]
```

0x00fb4	
0x00fb8	
0x00fbc	
0x00fc0	
0x00fc4	
0x00fc8	
0x00fcc	
0x00fd0	
0x00fd4	
0x00fd8	
0x00fdc	
0x00fe0	
0x00fe4	
0x00fe8	
0x00fec	0x0003(vec)
0x00ff0	0x0000 (FP)
0x00ff4	0x1000 (IP)
0x00ff8	0x0000 (LR)
0x00ffc	0x8010 (PC)
0x01000	

SP=0x00fe0

IP=0x01000

FP=0x00ffc

Manage the stack

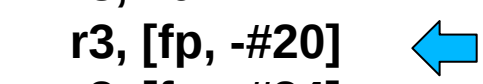
```
_start()
{
        int times=3;
        int i=0;
        int counter=0;
        int iteration=5;

        while (i<iteration)
        {

        times=delay1(time
                counter+=
                i++;
        }
}
```

Leave space for 4 integer variables

```
mov     ip, sp
stmdb   sp!, {fp, ip, lr, pc}
sub     fp, ip, #4
sub     sp, sp, #16
mov     r3, #3
str     r3, [fp, -#16]
mov     r3, #0
str     r3, [fp, -#20]
str     r3, [fp, -#24]
mov     r3, #5
str     r3, [fp, -#28]
ldr     r2, [fp, -#20]
ldr     r3, [fp, -#28]
```



0x00fb4	
0x00fb8	
0x00fbc	
0x00fc0	
0x00fc4	
0x00fc8	
0x00fcc	
0x00fd0	
0x00fd4	
0x00fd8	
0x00fdc	
0x00fe0	
0x00fe4	
0x00fe8	0x0000 (i)
0x00fec	0x0003(vec)
0x00ff0	0x0000 (FP)
0x00ff4	0x1000 (IP)
0x00ff8	0x0000 (LR)
0x00ffc	0x8010 (PC)
0x01000	

SP=0x00fe0

IP=0x01000

FP=0x00ffc

Manage the stack

```
_start()
{
        int times=3;
        int i=0;
        int counter=0;
        int iteration=5;

        while (i<iteration)
        {

        times=delay1(time
                counter+=
                i++;
        }
}
```

Leave space for 4 integer variables

```
mov     ip, sp
stmdb   sp!, {fp, ip, lr, pc}
sub     fp, ip, #4
sub     sp, sp, #16
mov     r3, #3
str     r3, [fp, -#16]
mov     r3, #0
str     r3, [fp, -#20]
str     r3, [fp, -#24]
mov     r3, #5
str     r3, [fp, -#28]
ldr     r2, [fp, -#20]
ldr     r3, [fp, -#28]
```

0x00fb4	
0x00fb8	
0x00fbc	
0x00fc0	
0x00fc4	
0x00fc8	
0x00fcc	
0x00fd0	
0x00fd4	
0x00fd8	
0x00fdc	
0x00fe0	
0x00fe4	0x0000 (cont)
0x00fe8	0x0000 (i)
0x00fec	0x0003(vec)
0x00ff0	0x0000 (FP)
0x00ff4	0x1000 (IP)
0x00ff8	0x0000 (LR)
0x00ffc	0x8010 (PC)
0x01000	

SP=0x00fe0

IP=0x01000

FP=0x00ffc

Manage the stack

Leave space for 4 integer variables. They are saved in the same order as declared in the code

```
_st
{
        int times=3;
        int i=0;
        int counter=0;
        int iteration=5;

        while (i<iteration)
        {

        times=delay1(time
                counter+=
                i++;
        }
}
```

```
mov     ip, sp
stmdb   sp!, {fp, ip, lr, pc}
sub     fp, ip, #4
sub     sp, sp, #16
mov     r3, #3
str     r3, [fp, -#16]
mov     r3, #0
str     r3, [fp, -#20]
str     r3, [fp, -#24]
mov     r3, #5
str     r3, [fp, -#28]
ldr     r2, [fp, -#20]
ldr     r3, [fp, -#28]
```

0x00fb4	
0x00fb8	
0x00fbc	
0x00fc0	
0x00fc4	
0x00fc8	
0x00fcc	
0x00fd0	
0x00fd4	
0x00fd8	
0x00fdc	
0x00fe0	0x0005 (iter)
0x00fe4	0x0000 (cont)
0x00fe8	0x0000 (i)
0x00fec	0x0003(vec)
0x00ff0	0x0000 (FP)
0x00ff4	0x1000 (IP)
0x00ff8	0x0000 (LR)
0x00ffc	0x8010 (PC)
0x01000	

Activation Frame

SP=0x00fe0

IP=0x01000

FP=0x00ffc

Manage the stack

```
_start()
{
        int times=3;
        int i=0;
        int counter=0;
        int iteration=5;

        while (i<iteration)
        {

        times=delay1(time
                counter+=
                i++;
        }
}
```

In *r0* the parameter "times" is stored

```
...
ldr     r2, [fp, -#20]
ldr     r3, [fp, -#28]
cmp     r2, r3
blt     0x8040
b       0x806c
ldr     r0, [fp, -#16]
bl      0x8078
0x00008048      mov  r3,
r0
...
```

0x00fb4	
0x00fb8	
0x00fbc	
0x00fc0	
0x00fc4	
0x00fc8	
0x00fcc	
0x00fd0	
0x00fd4	
0x00fd8	
0x00fdc	
0x00fe0	0x0005 (iter)
0x00fe4	0x0000 (cont)
0x00fe8	0x0000 (i)
0x00fec	0x0003(vec)
0x00ff0	0x0000 (FP)
0x00ff4	0x1000 (IP)
0x00ff8	0x0000 (LR)
0x00ffc	0x8010 (PC)
0x01000	

Activation Frame

SP=0x00fe0

IP=0x01000

FP=0x00ffc

Manage the stack

```
_start()
{
        int times=3;
        int i=0;
        int counter=0;
        int iteration=5;

        while (i<iteration)
        {

        times=delay1(time
                counter+=
                i++;
        }
}
```

We jump to the subroutine

```
...
ldr     r2, [fp, -#20]
ldr     r3, [fp, -#28]
cmp     r2, r3
blt     0x8040
b       0x806c
ldr     r0, [fp, -#16]
bl      0x8078
0x00008048      mov  r3,
r0
...
```

0x00fb4	
0x00fb8	
0x00fbc	
0x00fc0	
0x00fc4	
0x00fc8	
0x00fcc	
0x00fd0	
0x00fd4	
0x00fd8	
0x00fdc	
0x00fe0	0x0005 (iter)
0x00fe4	0x0000 (cont)
0x00fe8	0x0000 (i)
0x00fec	0x0003(vec)
0x00ff0	0x0000 (FP)
0x00ff4	0x1000 (IP)
0x00ff8	0x0000 (LR)
0x00ffc	0x8010 (PC)
0x01000	

Activation Frame

SP=0x00fe0

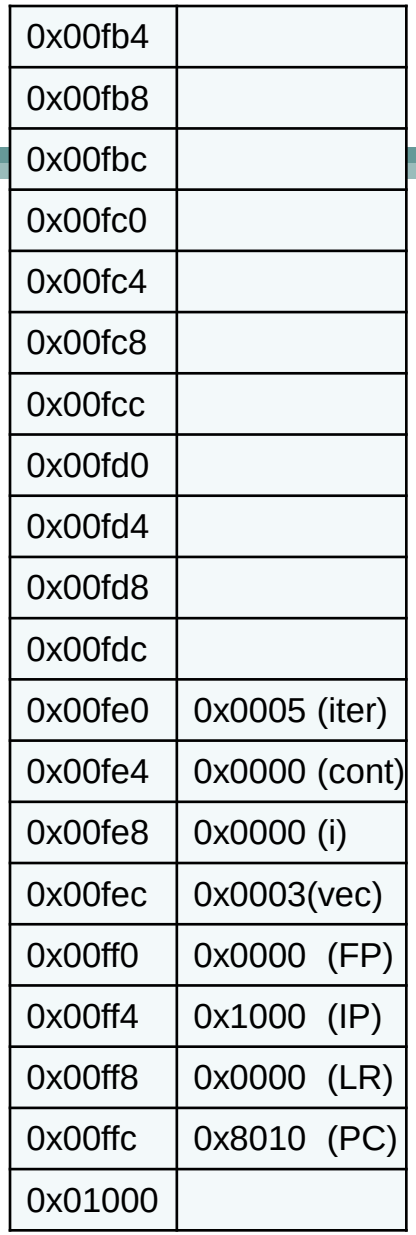
IP=0x01000

FP=0x00ffc

Manage the stack

```
int delay1 (int i)
{
        int j = 2;
        int k = 7;
        int l = 1;
        if (i>j)
        {
                i--;
                i = delay1
        }
        return k*l;
}
```

```
delay1:
0x8078  mov     ip, sp
stmdb   sp!, {fp, ip, lr, pc}
sub     fp, ip, #4
sub     sp, sp, #16
mov     r3, #3
str     r3, [fp, -#16]
mov     r3, #0
str     r3, [fp, -#20]
str     r3, [fp, -#24]
mov     r3, #5
str     r3, [fp, -#28]
ldr     r2, [fp, -#20]
ldr     r3, [fp, -#28]
```



SP=0x00fe0

IP=0x0fe0

FP=0x00ffc

Manage the stack

Saves *fp* of activation frame of call

Save return address after execution of function

```
int delay1 (int i)
{
        int j = 2;
        int k = 7;
        int l = 1;
        if (i>j)
        {
                i--;
                i = delay1
        }
        return k*l;
}
```

```
delay1:
0x8078  mov     ip, sp
stmdb   sp!, {fp, ip, lr, pc}
sub     fp, ip, #4
sub     sp, sp, #16
mov     r3, #3
str     r3, [fp, -#16]
mov     r3, #0
str     r3, [fp, -#20]
str     r3, [fp, -#24]
mov     r3, #5
str     r3, [fp, -#28]
ldr     r2, [fp, -#20]
ldr     r3, [fp, -#28]
```

0x00fb4	
0x00fb8	
0x00fbc	
0x00fc0	
0x00fc4	
0x00fc8	
0x00fcc	
0x00fd0	0x0ffc (FP)
0x00fd4	0x0fe0 (IP)
0x00fd8	0x8048 (LR)
0x00fdc	0x8088 (PC)
0x00fe0	0x0005 (iter)
0x00fe4	0x0000 (cont)
0x00fe8	0x0000 (i)
0x00fec	0x0003(vec)
0x00ff0	0x0000 (FP)
0x00ff4	0x1000 (IP)
0x00ff8	0x0000 (LR)
0x00ffc	0x8010 (PC)
0x01000	

Activation Frame

Save stack pointer address before the call

SP=0x00fd0

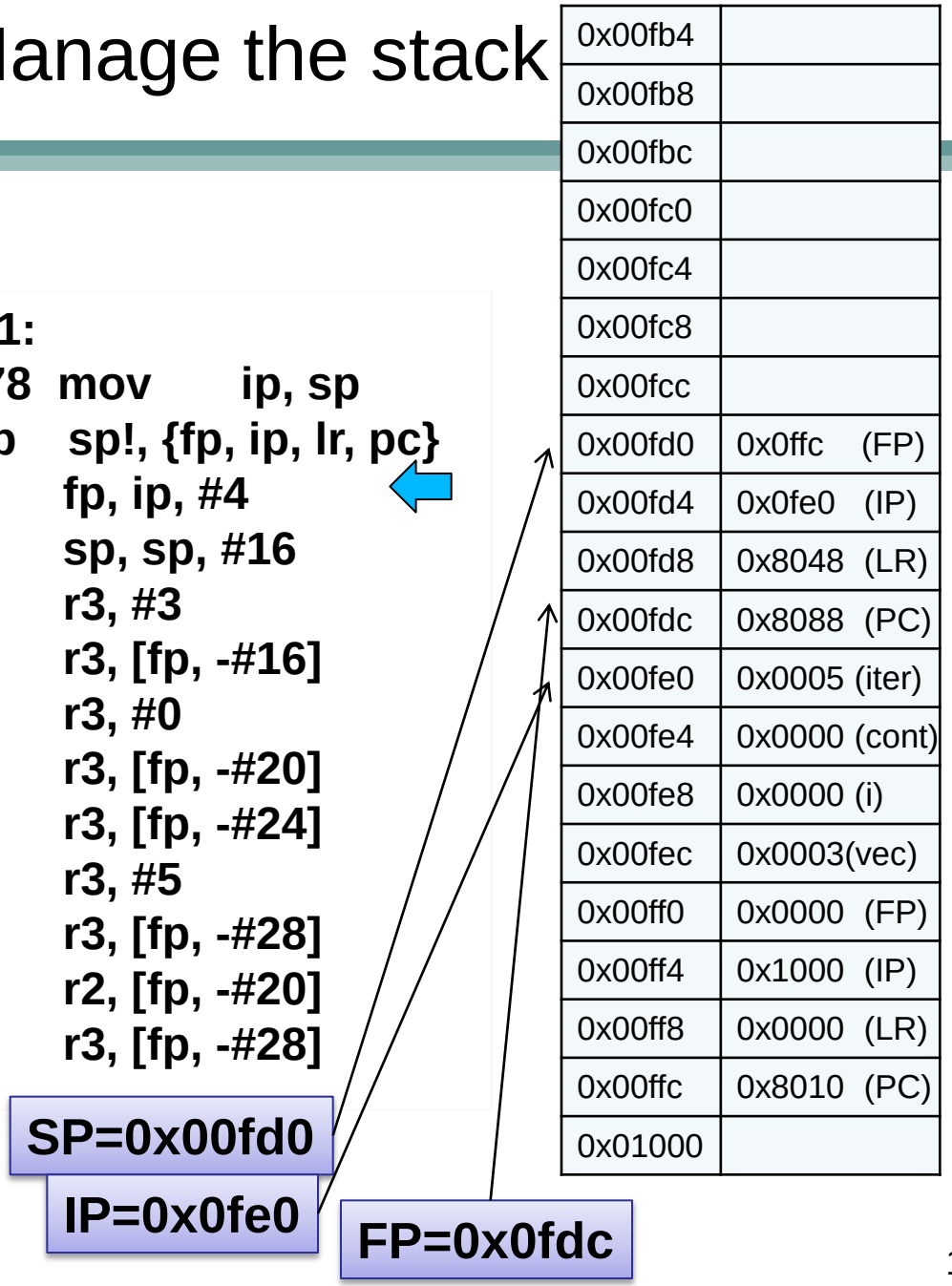
IP=0x0fe0

FP=0x0ffc

Manage the stack

```
int delay1 (int i)
{
        int j = 2;
        int k = 7;
        int l = 1;
        if (i>j)
        {
                i--;
                i = delay1
        }
        return k*l;
}
```

```
delay1:
0x8078  mov     ip, sp
stmdb   sp!, {fp, ip, lr, pc}
sub     fp, ip, #4
sub     sp, sp, #16
mov     r3, #3
str     r3, [fp, -#16]
mov     r3, #0
str     r3, [fp, -#20]
str     r3, [fp, -#24]
mov     r3, #5
str     r3, [fp, -#28]
ldr     r2, [fp, -#20]
ldr     r3, [fp, -#28]
```



Manage the stack

Leave space for 4 integer variables

```
int delay1 (int i)
{
        int j = 2;
        int k = 7;
        int l = 1;
        if (i>j)
        {
                i--;
                i = delay1
        }
        return k*l;
}
```

```
delay1:
0x8078  mov     ip, sp
stmdb   sp!, {fp, ip, lr, pc}
sub     fp, ip, #4
sub     sp, sp, #16
str     r0, [fp, -#16]
mov     r3, #2
str     r3, [fp, -#20]
mov     r3, #7
str     r3, [fp, -#24]
mov     r3, #1
str     r3, [fp, -#28]
```

0x00fb4	
0x00fb8	
0x00fbc	
0x00fc0	
0x00fc4	
0x00fc8	
0x00fcc	
0x00fd0	0x0ffc (FP)
0x00fd4	0x0fe0 (IP)
0x00fd8	0x8048 (LR)
0x00fdc	0x8088 (PC)
0x00fe0	0x0005 (iter)
0x00fe4	0x0000 (cont)
0x00fe8	0x0000 (i)
0x00fec	0x0003(vec)
0x00ff0	0x0000 (FP)
0x00ff4	0x1000 (IP)
0x00ff8	0x0000 (LR)
0x00ffc	0x8010 (PC)
0x01000	

Activation Frame

SP=0x00fc0

IP=0x0fe0

FP=0x0fdc

Manage the stack

Leave space for 4 integer variables

```
int delay1 (int i)
{
        int j = 2;
        int k = 7;
        int l = 1;
        if (i>j)
        {
                i--;
                i = delay1
        }
        return k*l;
}
```

```
delay1:
0x8078  mov     ip, sp
stmdb   sp!, {fp, ip, lr, pc}
sub     fp, ip, #4
sub     sp, sp, #16
str     r0, [fp, -#16]
mov     r3, #2
str     r3, [fp, -#20]
mov     r3, #7
str     r3, [fp, -#24]
mov     r3, #1
str     r3, [fp, -#28]
```

0x00fb4	
0x00fb8	
0x00fbc	
0x00fc0	
0x00fc4	
0x00fc8	
0x00fcc	0x0003 (i)
0x00fd0	0x0ffc (FP)
0x00fd4	0x0fe0 (IP)
0x00fd8	0x8048 (LR)
0x00fdc	0x8088 (PC)
0x00fe0	0x0005 (iter)
0x00fe4	0x0000 (cont)
0x00fe8	0x0000 (i)
0x00fec	0x0003(vec)
0x00ff0	0x0000 (FP)
0x00ff4	0x1000 (IP)
0x00ff8	0x0000 (LR)
0x00ffc	0x8010 (PC)
0x01000	

Activation Frame

SP=0x00fc0

IP=0x0fe0

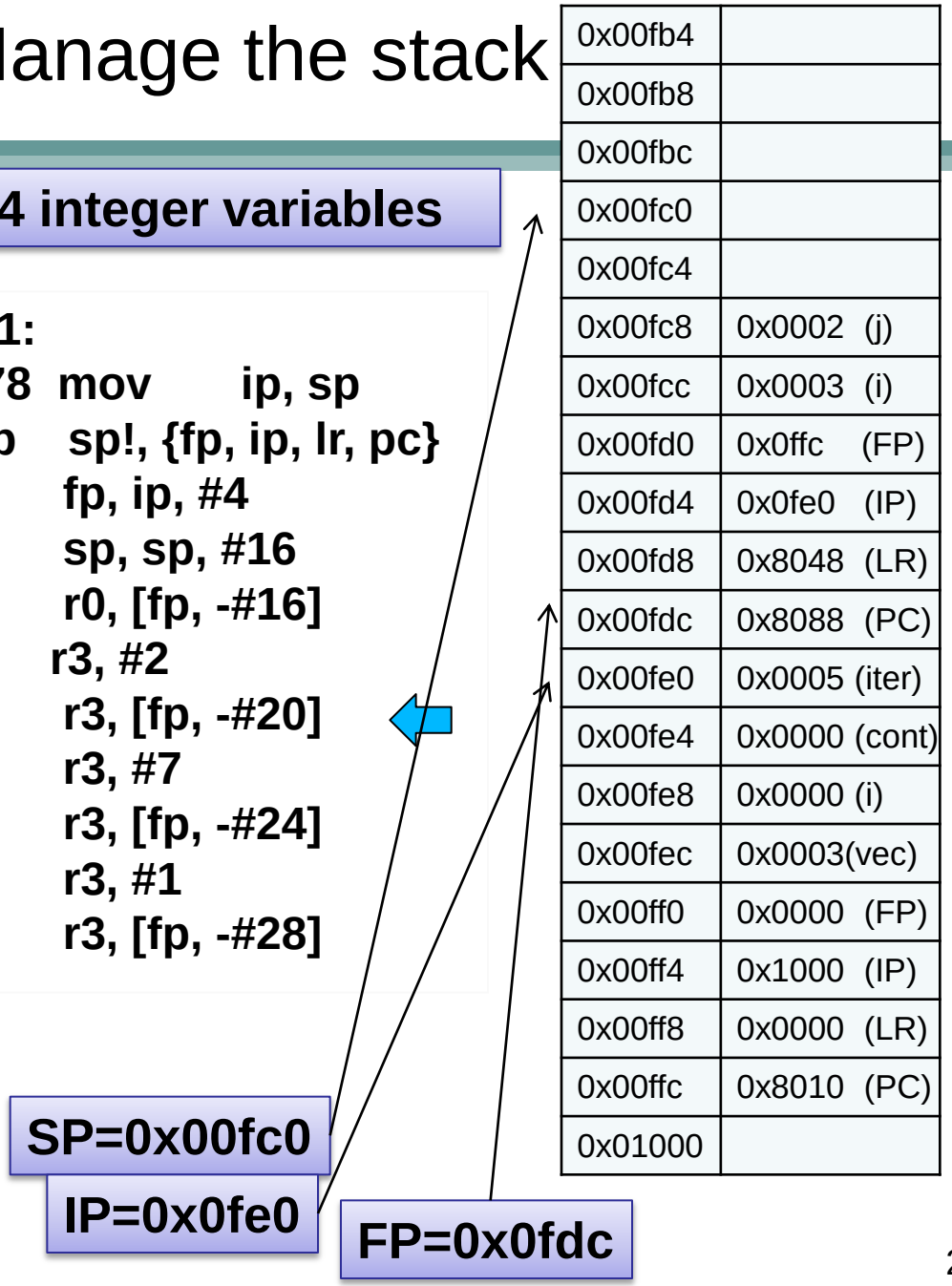
FP=0x0fdc

Manage the stack

Leave space for 4 integer variables

```
int delay1 (int i)
{
        int j = 2;
        int k = 7;
        int l = 1;
        if (i>j)
        {
                i--;
                i = delay1
        }
        return k*l;
}
```

```
delay1:
0x8078  mov     ip, sp
stmdb   sp!, {fp, ip, lr, pc}
sub     fp, ip, #4
sub     sp, sp, #16
str     r0, [fp, -#16]
mov     r3, #2
str     r3, [fp, -#20]
mov     r3, #7
str     r3, [fp, -#24]
mov     r3, #1
str     r3, [fp, -#28]
```

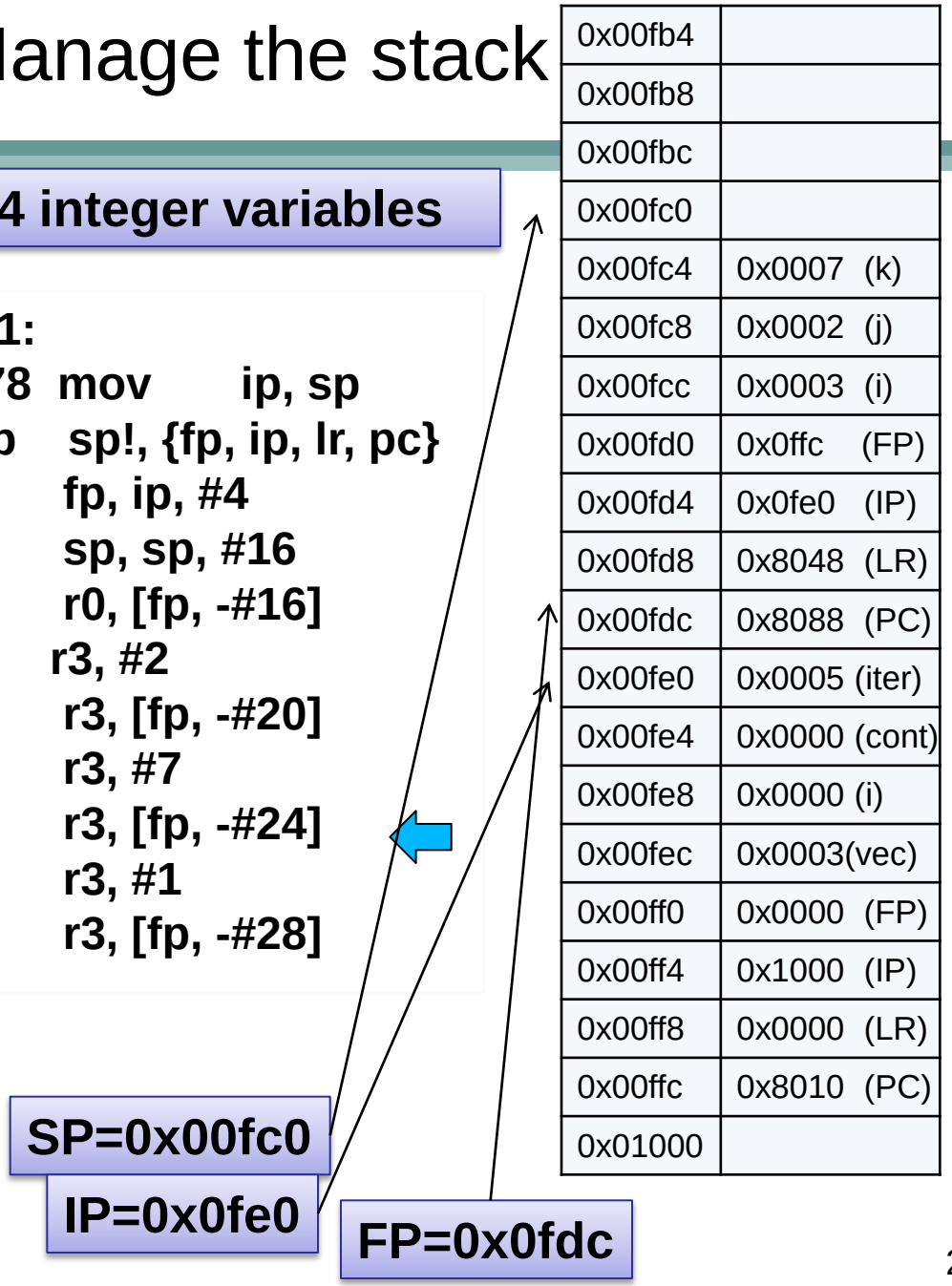


Manage the stack

Leave space for 4 integer variables

```
int delay1 (int i)
{
        int j = 2;
        int k = 7;
        int l = 1;
        if (i>j)
        {
                i--;
                i = delay1
        }
        return k*l;
}
```

```
delay1:
0x8078  mov     ip, sp
stmdb   sp!, {fp, ip, lr, pc}
sub     fp, ip, #4
sub     sp, sp, #16
str     r0, [fp, -#16]
mov     r3, #2
str     r3, [fp, -#20]
mov     r3, #7
str     r3, [fp, -#24]
mov     r3, #1
str     r3, [fp, -#28]
```

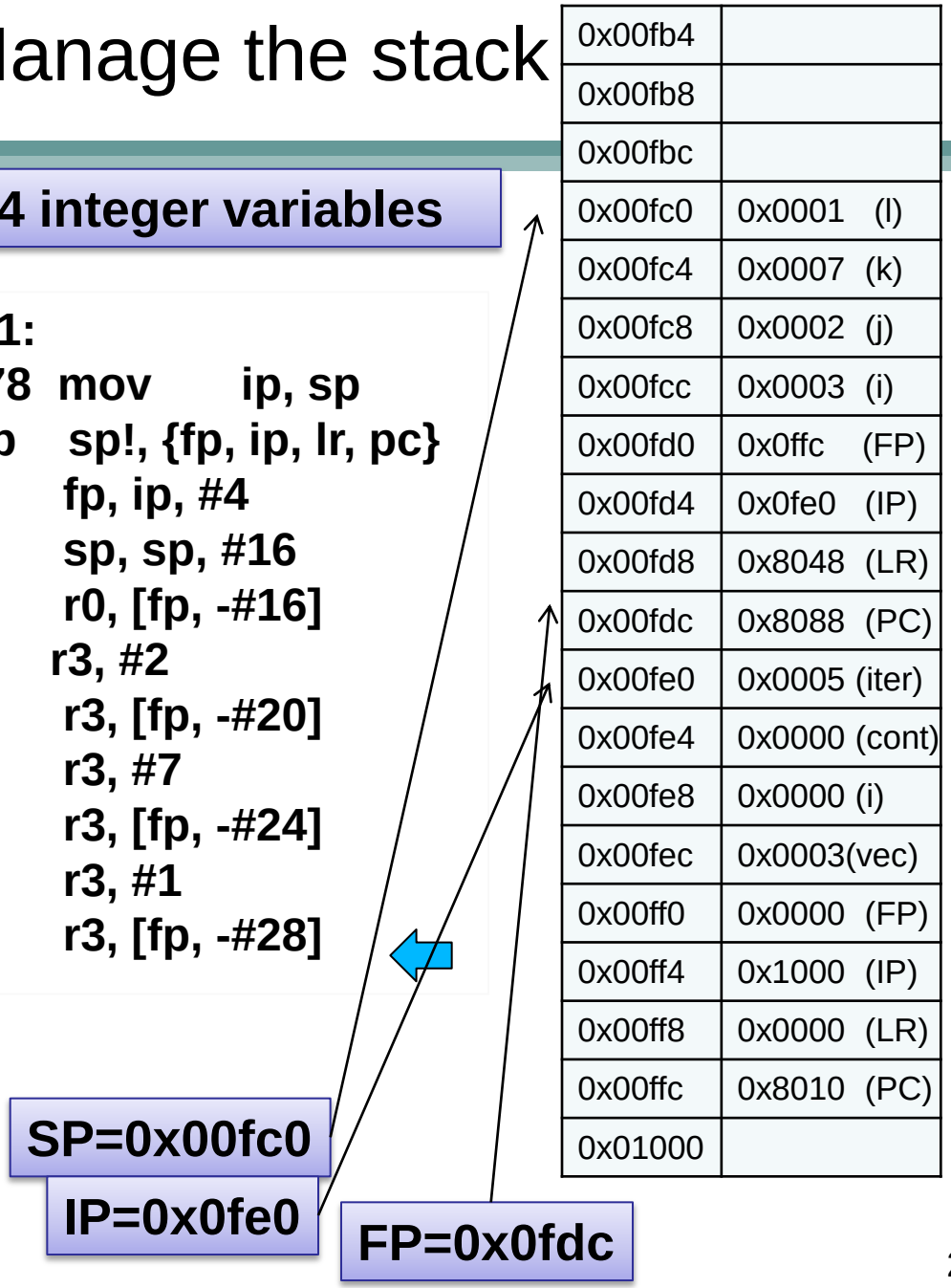


Manage the stack

Leave space for 4 integer variables

```
int delay1 (int i)
{
        int j = 2;
        int k = 7;
        int l = 1;
        if (i>j)
        {
                i--;
                i = delay1
        }
        return k*l;
}
```

```
delay1:
0x8078  mov     ip, sp
stmdb   sp!, {fp, ip, lr, pc}
sub     fp, ip, #4
sub     sp, sp, #16
str     r0, [fp, -#16]
mov     r3, #2
str     r3, [fp, -#20]
mov     r3, #7
str     r3, [fp, -#24]
mov     r3, #1
str     r3, [fp, -#28]
```



Manage the stack

Before calling delay1, we store the parameter in *r0*

```
int delay1 (int i)
{
        int j = 2;
        int k = 7;
        int l = 1;
        if (i>j)
        {
                i--;
                i = delay1(i);
        }
        return k*l;
}
```

```
...
ldr     r0, [fp, -#16]
bl      0x8078
0x000080c8      mov
r3, r0
...
```

0x00fb4	
0x00fb8	
0x00fbc	
0x00fc0	0x0001 (l)
0x00fc4	0x0007 (k)
0x00fc8	0x0002 (j)
0x00fcc	0x0002 (i)
0x00fd0	0x0ffc (FP)
0x00fd4	0x0fe0 (IP)
0x00fd8	0x8048 (LR)
0x00fdc	0x8088 (PC)
0x00fe0	0x0005 (iter)
0x00fe4	0x0000 (cont)
0x00fe8	0x0000 (i)
0x00fec	0x0003(vec)
0x00ff0	0x0000 (FP)
0x00ff4	0x1000 (IP)
0x00ff8	0x0000 (LR)
0x00ffc	0x8010 (PC)
0x01000	

Activation Frame

Activation Frame

SP=0x00fc0

IP=0x0fe0

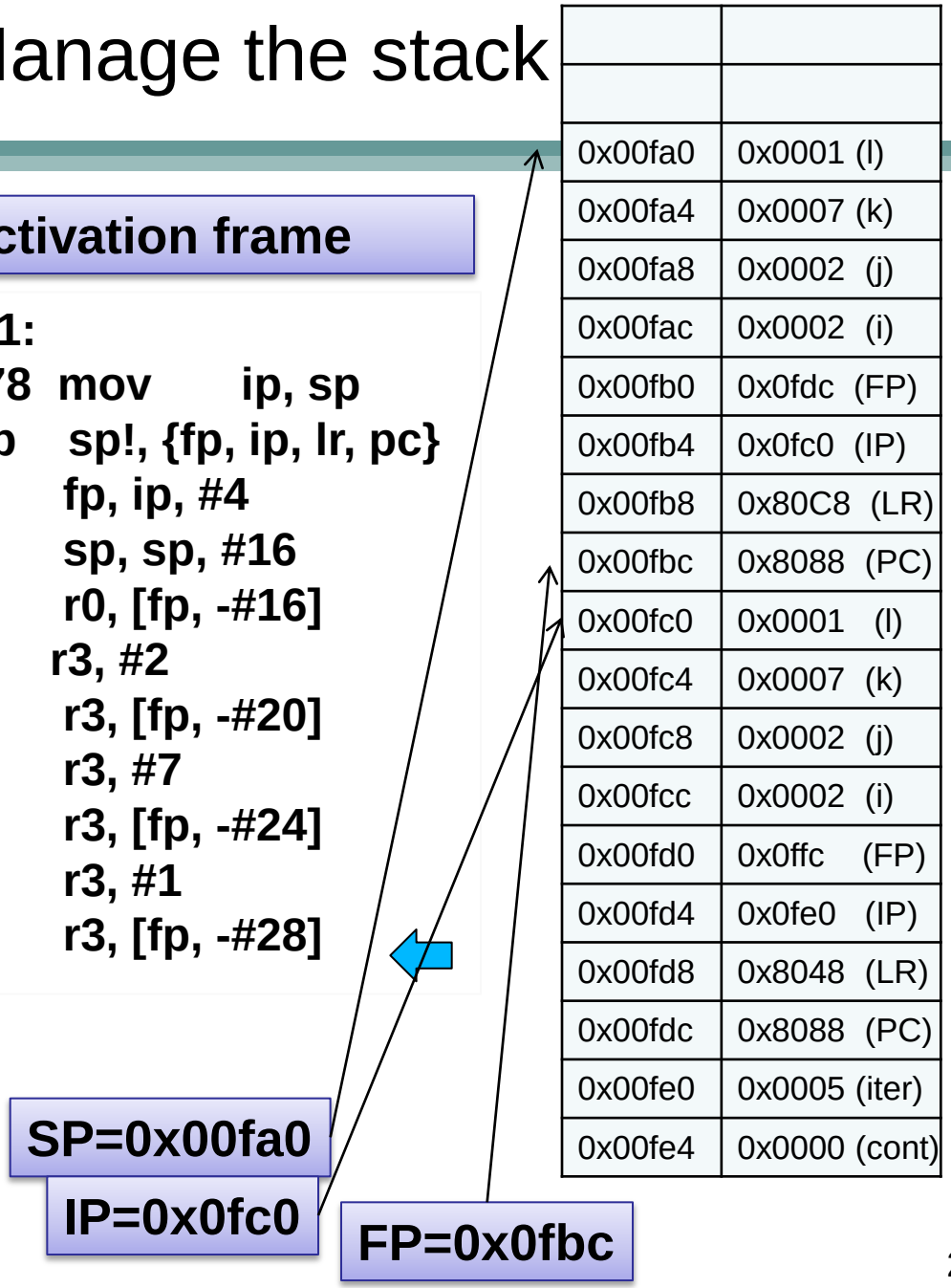
FP=0x0fdc

Manage the stack

Generate a new activation frame

```
int delay1 (int i)
{
        int j = 2;
        int k = 7;
        int l = 1;
        if (i>j)
        {
                i--;
                i = delay1
        }
        return k*l;
}
```

```
delay1:
0x8078  mov     ip, sp
stmdb   sp!, {fp, ip, lr, pc}
sub     fp, ip, #4
sub     sp, sp, #16
str     r0, [fp, -#16]
mov     r3, #2
str     r3, [fp, -#20]
mov     r3, #7
str     r3, [fp, -#24]
mov     r3, #1
str     r3, [fp, -#28]
```



Manage the stack

- After the second call we do not have any other recursive call. So, we return to the subroutine or function.

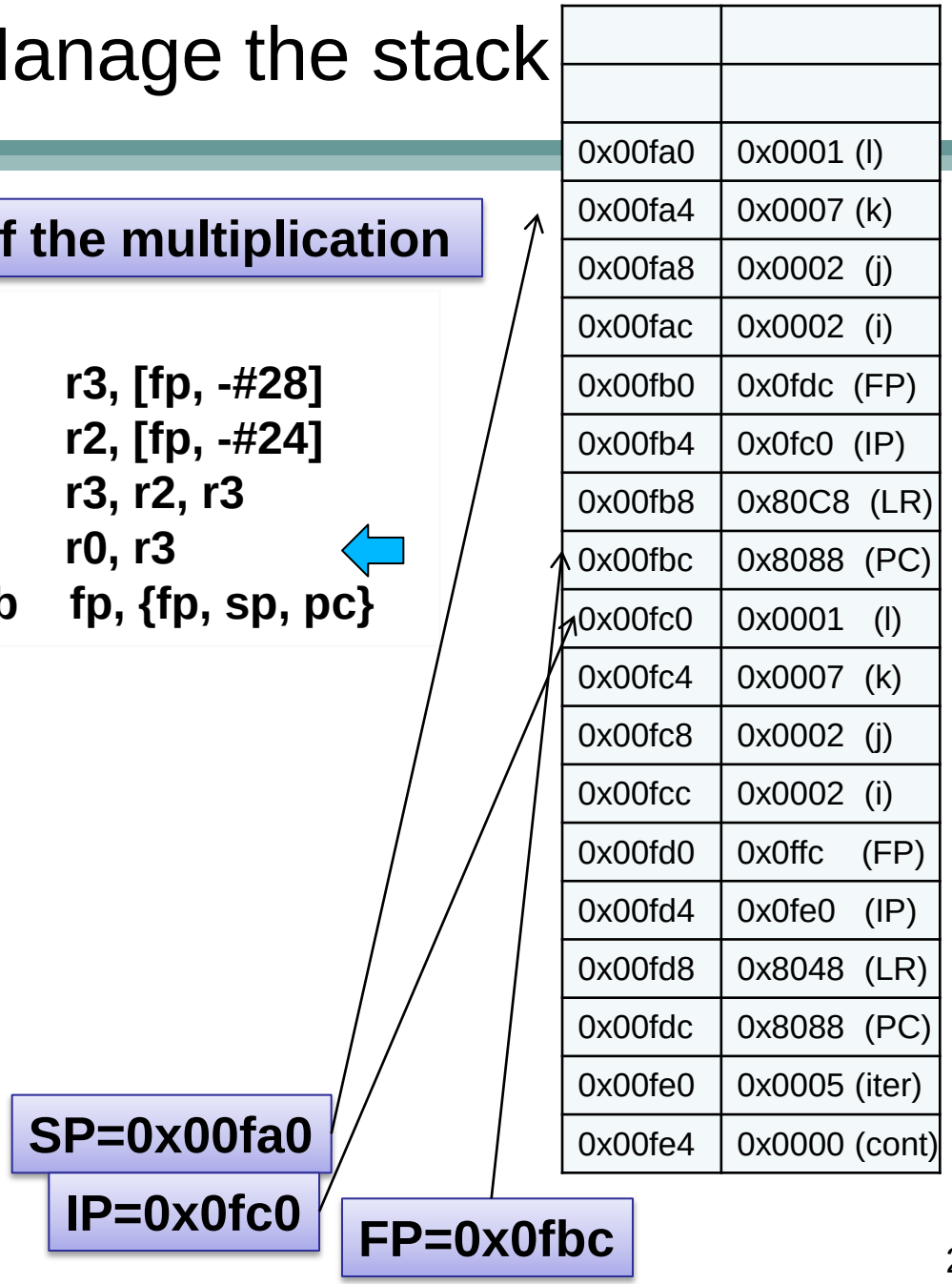
- In this situation, the return value is stored in ***r0*** and we start unstacking, how is this process done?

Manage the stack

Store in *r0* the result of the multiplication

```
int delay1 (int i)
{
        int j = 2;
        int k = 7;
        int l = 1;
        if (i>j)
        {
                i--;
                i = delay1(i);
        }
        return k*l;
}
```

```
…
ldr     r3, [fp, -#28]
ldr     r2, [fp, -#24]
mul     r3, r2, r3
mov     r0, r3
ldmdb   fp, {fp, sp, pc}
```



Manage the stack

Unstack: we read from FP, which had the value 0x0fbc. As we have a decrement before, we first decrease the content of the register

```
ir
{
        int j = 2;
        int k = 7;
        int l = 1;
        if (i>j)
        {
                i--;
                i = delay1(i);
        }
        return k*l;
}
```

```
ldr     r3, [fp, -#28]
ldr     r2, [fp, -#24]
mul     r3, r2, r3
mov     r0, r3
ldmdb   fp, {fp, sp, pc}
```

0x00fa0	0x0001 (l)
0x00fa4	0x0007 (k)
0x00fa8	0x0002 (j)
0x00fac	0x0002 (i)
0x00fb0	0x0fdc (FP)
0x00fb4	0x0fc0 (IP)
0x00fb8	0x80C8 (LR)
0x00fbc	0x8088 (PC)
0x00fc0	0x0001 (l)
0x00fc4	0x0007 (k)
0x00fc8	0x0002 (j)
0x00fcc	0x0002 (i)
0x00fd0	0x0ffc (FP)
0x00fd4	0x0fe0 (IP)
0x00fd8	0x8048 (LR)
0x00fdc	0x8088 (PC)
0x00fe0	0x0005 (iter)
0x00fe4	0x0000 (cont)

Activation Frame

Activation Frame

Return value of next instruc. to run

PC=0x080C8

SP=0x00fc0

IP=0x0fc0

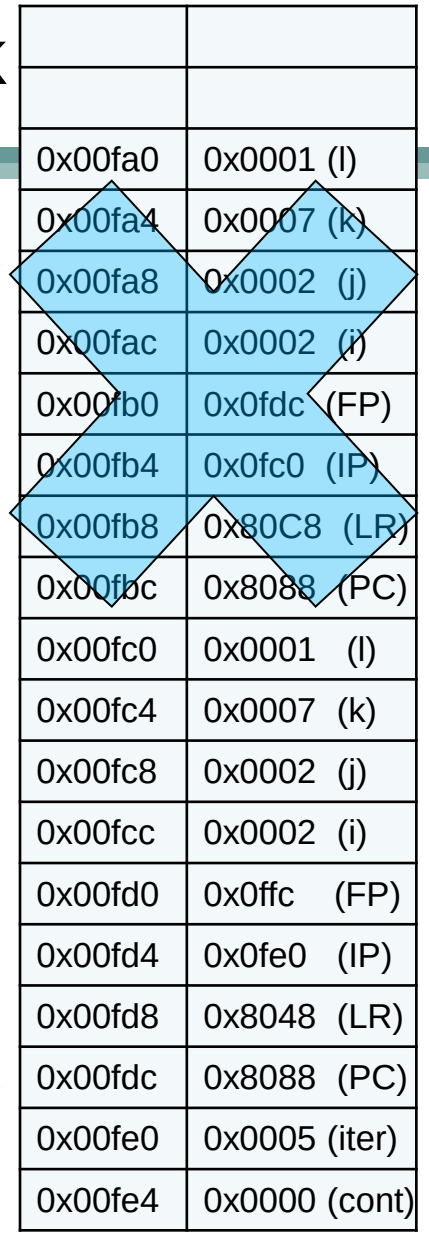
FP=0x0fdc

Manage the stack

We retrieve the return value in *r0*

```
int delay1 (int i)
{
        int j = 2;
        int k = 
        int l = 1;
        if (i>j)
        {
                i--;
                i = delay1(i);
        }
        return k*l;
}
```

```
…
0x000080c4      bl      0x8078
0x000080c8      mov     r3, r0
0x000080cc              str
        r3, [fp, -#16]
…
```



SP=0x00fc0

IP=0x0fc0

FP=0x0fdc

Manage the stack

And we store its local value

```
int delay1 (int i)
{
        int j = 2
        int k =
        int l = 1
        if (i>j)
        {
                i--;
                i = delay1(i);
        }
        return k*l;
}
```

```
…
0x000080c4      bl      0x8078
0x000080c8      mov     r3, r0
0x000080cc              str
        r3, [fp, -#16]
…
```

0x00fa0	0x0001 (l)
0x00fa4	0x0007 (k)
0x00fa8	0x0002 (j)
0x00fac	0x0002 (i)
0x00fb0	0x0fdc (FP)
0x00fb4	0x0fc0 (IP)
0x00fb8	0x80C8 (LR)
0x00fbc	0x8088 (PC)
0x00fc0	0x0001 (l)
0x00fc4	0x0007 (k)
0x00fc8	0x0002 (j)
0x00fcc	0x0007 (i)
0x00fd0	0x0ffc (FP)
0x00fd4	0x0fe0 (IP)
0x00fd8	0x8048 (LR)
0x00fdc	0x8088 (PC)
0x00fe0	0x0005 (iter)
0x00fe4	0x0000 (cont)

Activation Frame

SP=0x00fc0

IP=0x0fc0

FP=0x0fdc