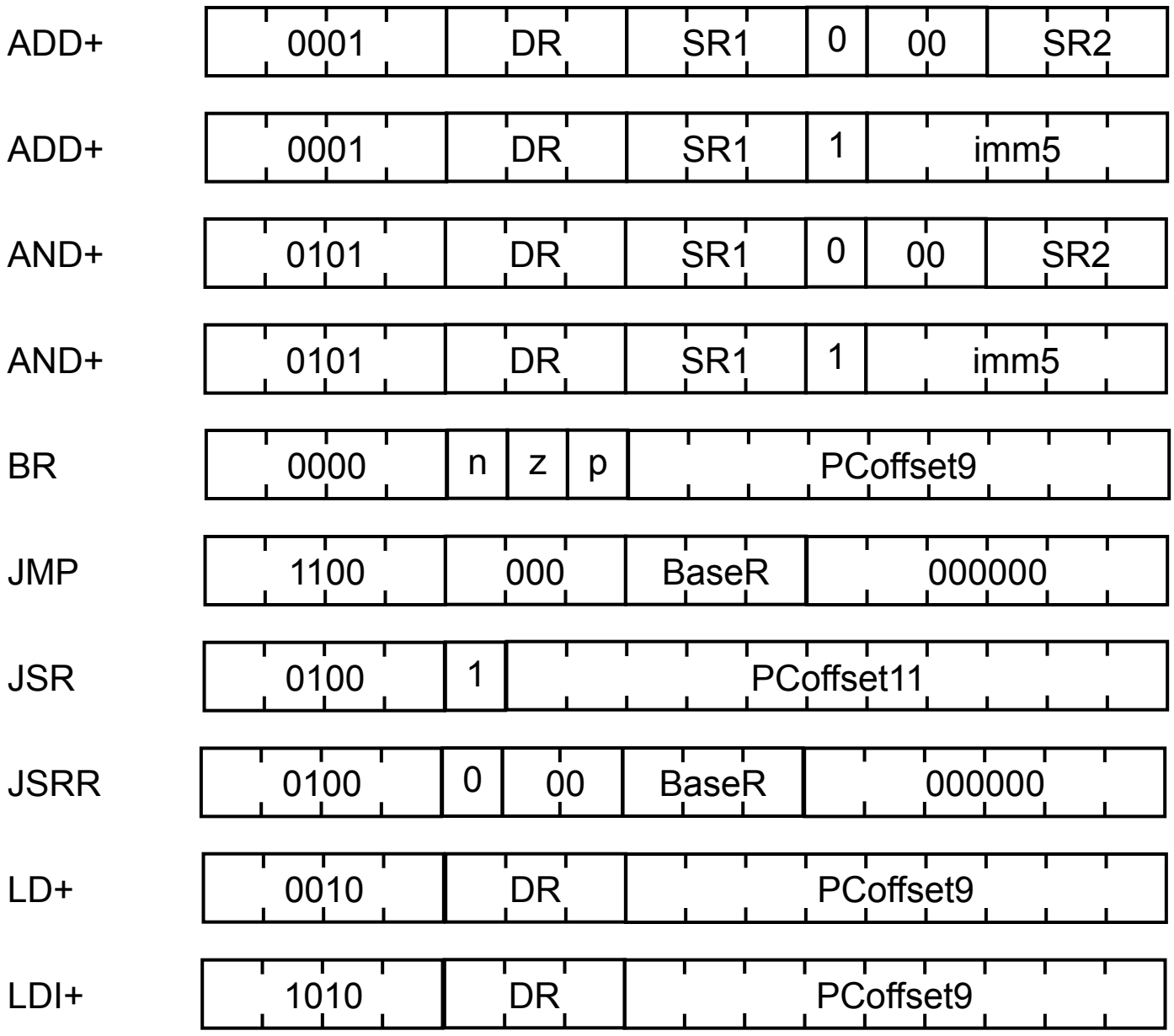
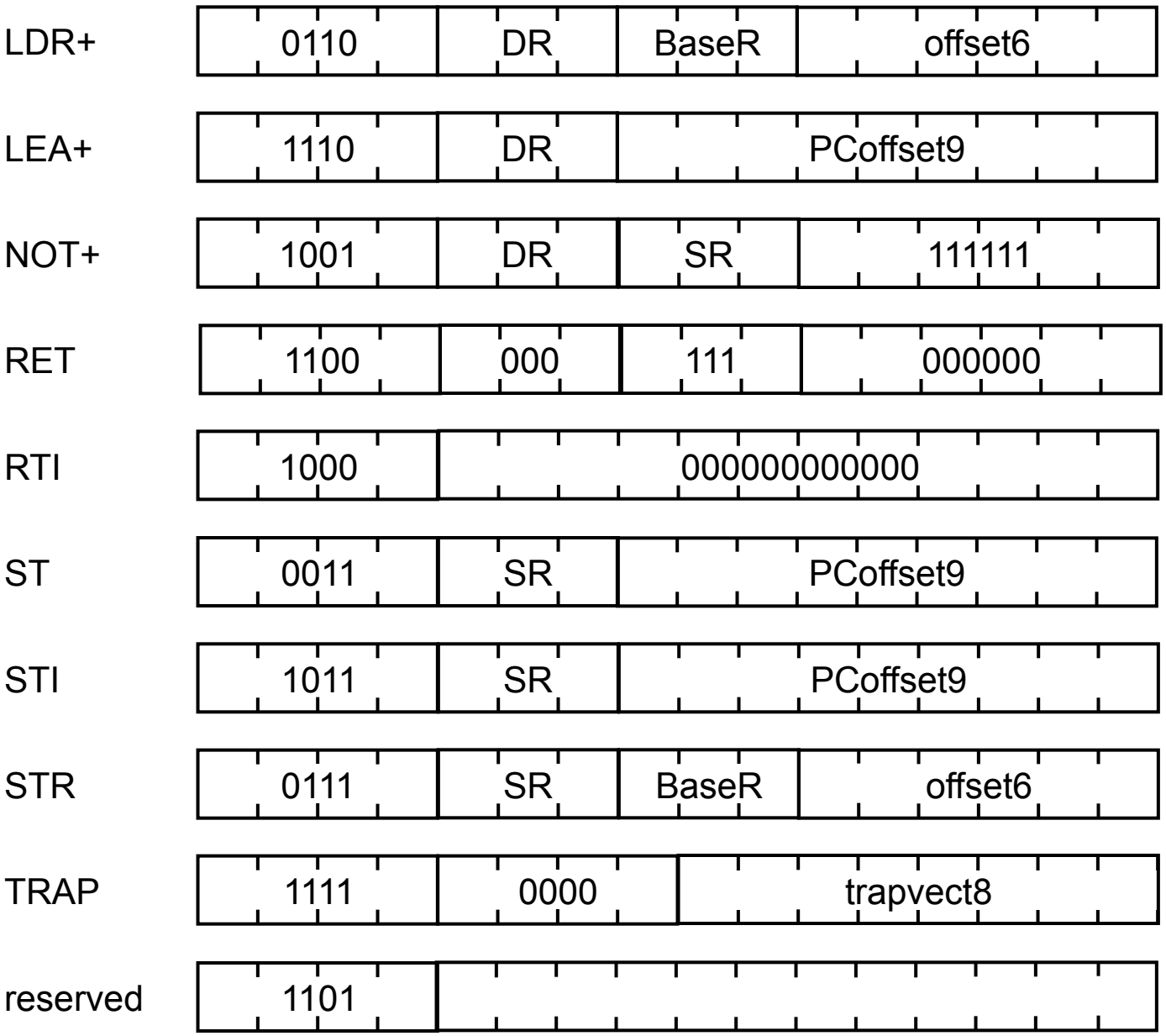
LC-3 Instructions, Control Structure and Calling Sequence Reference

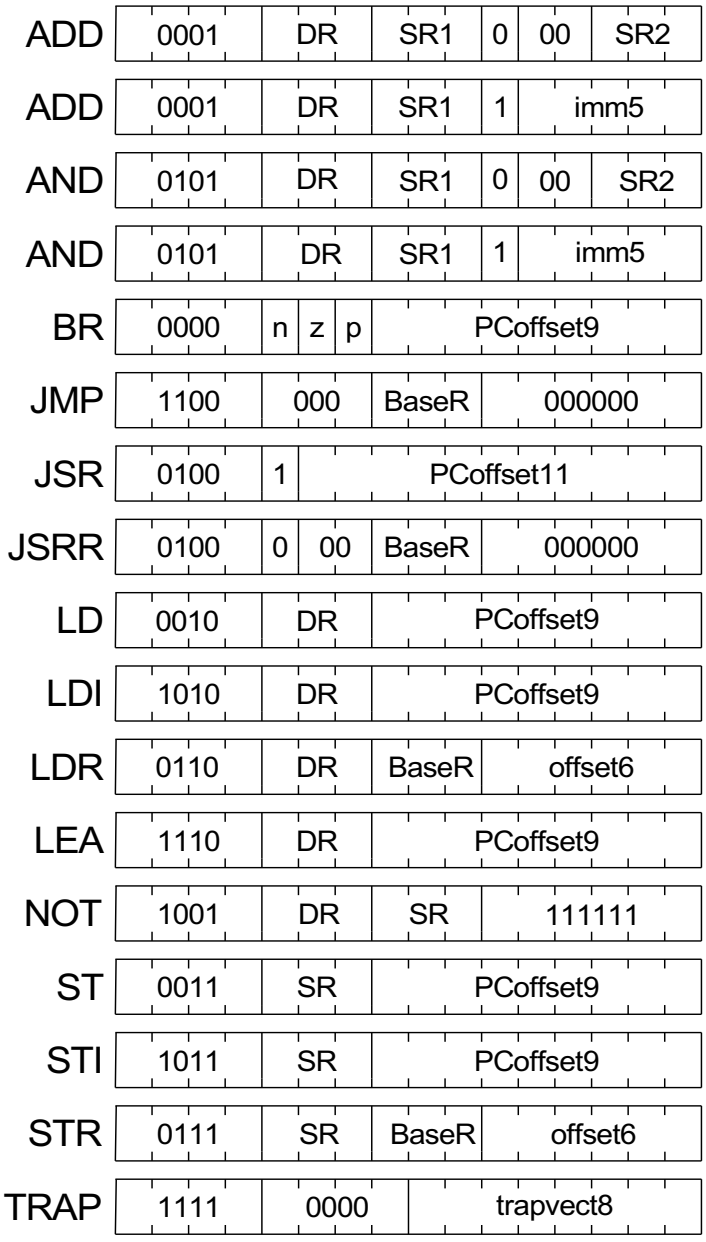


+ Indicates instructions that modify condition codes



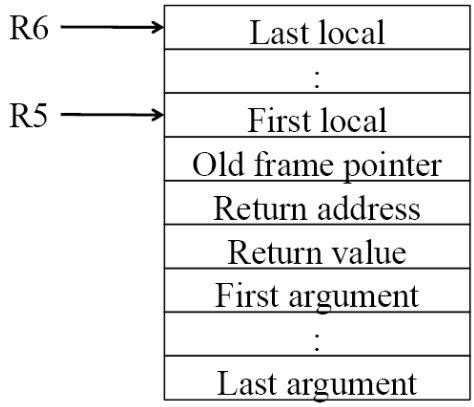
+ Indicates instructions that modify condition codes

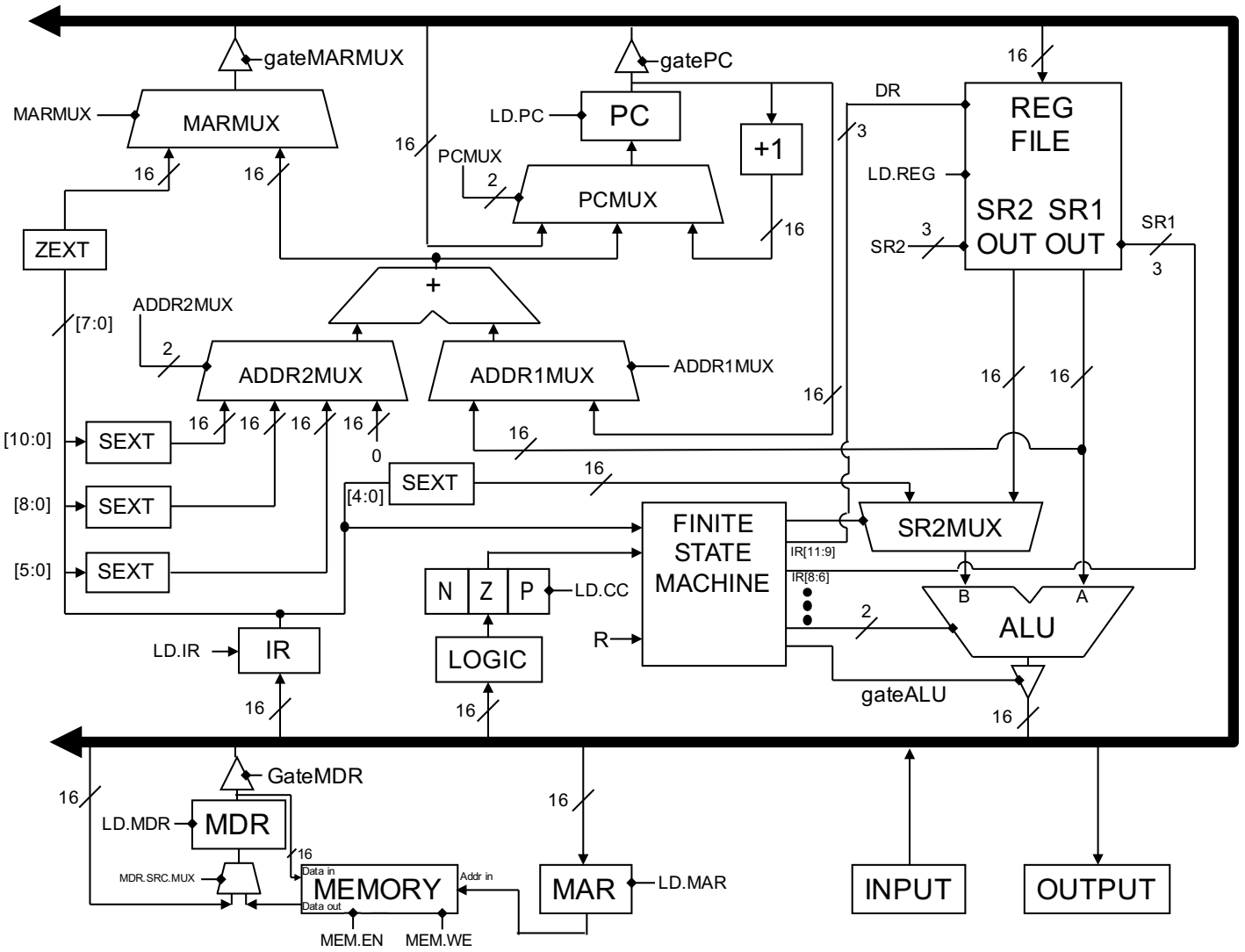
CS2110 REFERENCE SHEET



Trap Vector	*Assembler Name*
x20	GETC
x21	OUT
x22	PUTS
x23	IN
x25	HALT

Device Register	*Address*
Keybd Status Reg	xFE00
Keybd Data Reg	xFE02
Display Status Reg	xFE04
Display Data Reg	xFE06





Boolean Signals	
LD.MAR	GateMARMUX
LD.MDR	GateMDR
LD.REG	GatePC
LD.CC	GateALU
LD.PC	LD.IR
MEM.EN	MEM.WE

MUX Name	Possible Values
ALUK	ADD, AND, NOT, PASSA
ADDR1MUX	PC, BaseR
ADDR2MUX	ZERO, offset6, PCoffset9, PCoffset11
PCMUX	PC+1, BUS, ADDER
MARMUX	ZEXT, ADDER
MDR.SRC.MUX	BUS, MEM
SR2MUX	SR2, SEXT

if (R1>0) then .. else ..

```
        ADD     R1,R1,#0  ; if (R1>0) then
        BRNZ    ELSE1

    …[THEN part]…

        BRNZP   ENDIF1
ELSE1   NOP               ; else

    …[ELSE part]…

ENDIF1  NOP               ; endif
```

for (init; R1>0; reinit)

```
        …[init loop]…                   ; for (init;
FOR1    ADD     R1,R1,#0
        BRNZ    ENDF1   ; R1>0;

        …[FOR body]…

        …[reinit loop]…                 ; reinit)
        BRNZP   FOR1
ENDF1   NOP
```

while(R1>0)

```
WHILE1   ADD      R1,R1.#0 ; while(R1>0)
         BRNZ     ENDW1

        …[WHILE body]…

         BRNZP    WHILE1
ENDW1    NOP          ; endwhile
```

do … while(R1>0);

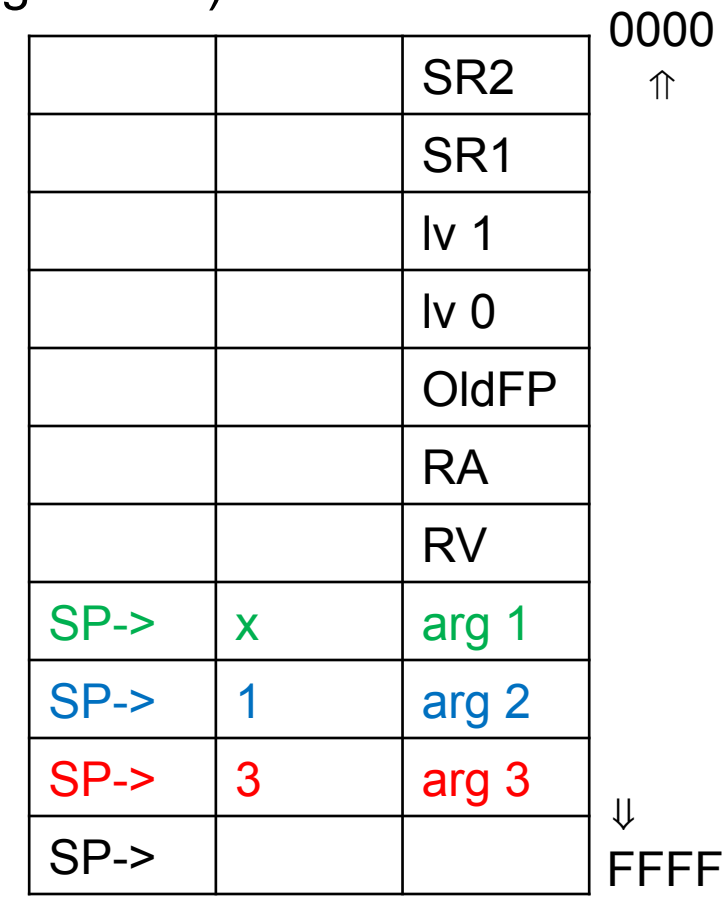
```
DO1     NOP                 ; do

        …[DO WHILE body]…

        ADD     R1,R1,#0
        BRP     DO1         ; while(x)
```

Boilerplate: Caller

before foo() returns - (3 arguments)

```
Caller  ADD R6, R6, -1  ; push(arg3)
        AND R0, R0, 0
        ADD R0, R0, 3
        STR R0, R6, 0
        ADD R6, R6, -1  ; push(arg2)
        AND R0, R0, 0
        ADD R0, R0, 1
        STR R0, R6, 0
        ADD R6, R6, -1  ; push(arg1)
        LD  R0, X
        STR R0, R6, 0
        JSR FOO         ; foo()
        ...
```

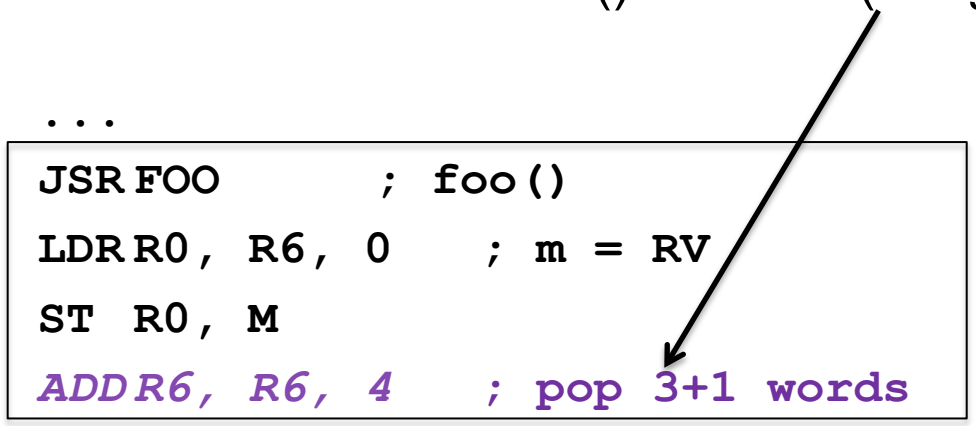
```
m = foo(x, 1, 3)
  = x + 1 - 3
```



Boilerplate: Caller

after foo() returns - (3 arguments)

```
...
JSR FOO        ; foo()
LDR R0, R6, 0    ; m = RV
ST  R0, M
ADD R6, R6, 4    ; pop 3+1 words

; and the function call is done!
```



	old R2	SR2
	old R1	SR1
		lv 1
		lv 0
	old R5	OldFP
	old R7	RA
~~SP->~~	x+1-3	RV
	x	arg 1
	1	arg 2
	3	arg 3
SP->		

0000 ⇑

⇓ FFFF

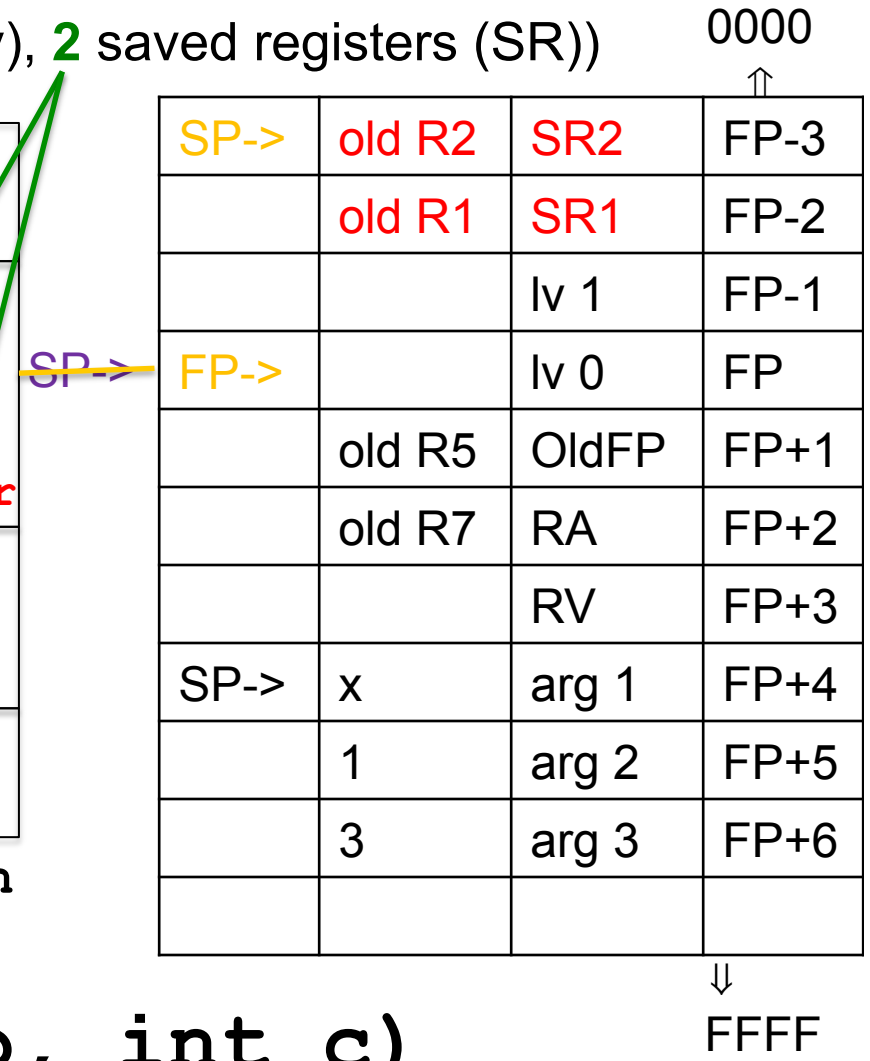
$$m = foo(x, 1, 3)$$

$$= x + 1 - 3$$

Boilerplate: Callee

(3 arguments, 2 local variables (lv), 2 saved registers (SR))

```
FOO   ADD R6, R6, -4  ; push 4 words
          ; for RV, RA, OldFP, lv0
                      ; set RV later
      STR R7, R6, 2   ; store RA
      STR R5, R6, 1   ; store OldFP
                ; set lv0 & 1 later
      ADD R5, R6, 0   ; FP = SP
      ADD R6, R6, -3  ; push 3 words
                      ; 2+2-1 (lv0)
      STR R1, R5, -2  ; save SR1
      STR R2, R5, -3  ; save SR2
      ... ; foo() implementation
```

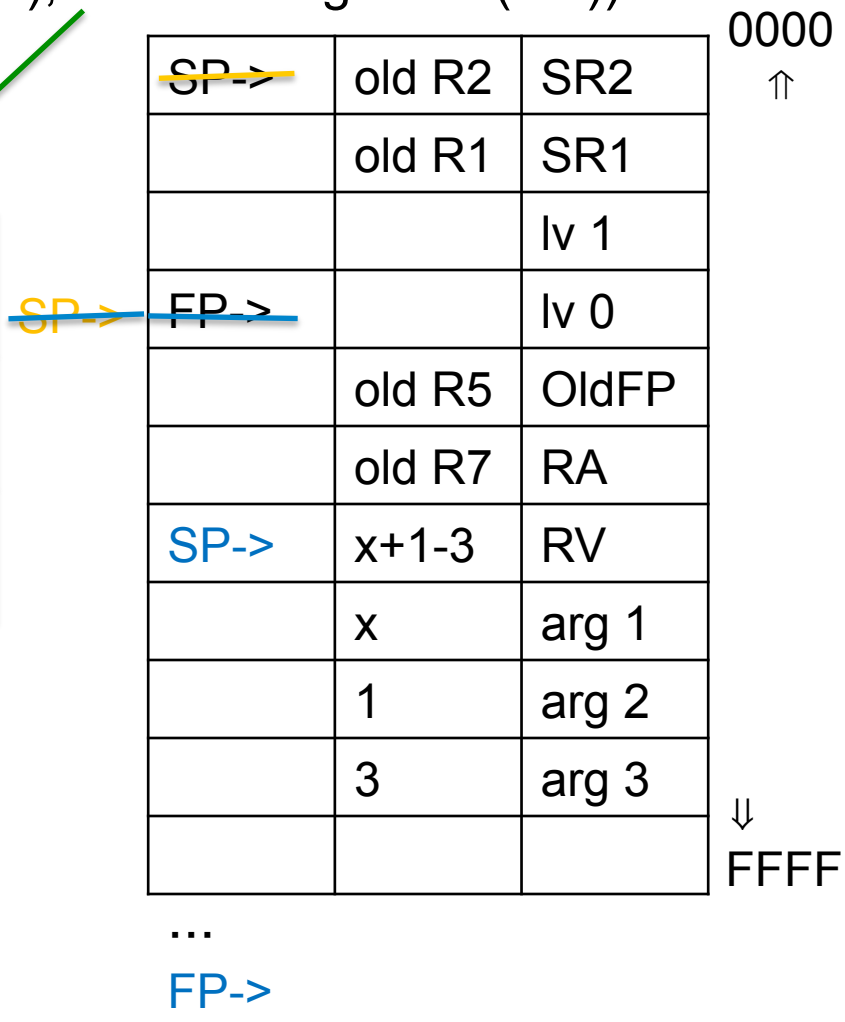



```
int foo(int a, int b, int c)
{
...
```

Boilerplate: Callee

(**3** arguments, **2** local variables (lv), **2** saved registers (SR))

```
... ;foo() implementation
; And now we're ready to return
LDR R1, R5, -2 ; restore SR1
LDR R2, R5, -3 ; restore SR2
ADD R6, R5, 0  ; pop lv0/1 & SR1/2
LDR R7, R5, 2  ; R7 = RA
LDR R5, R5, 1  ; FP = OldFP
ADD R6, R6, 3  ; pop 3 words
RET            ; foo() is done!
```




```
int foo(int a, int b, int c)...
```