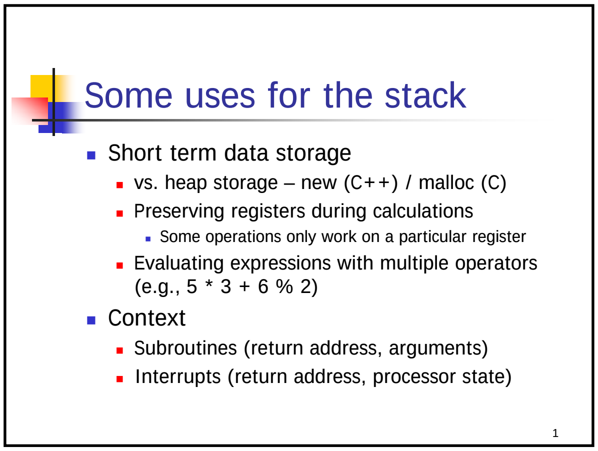
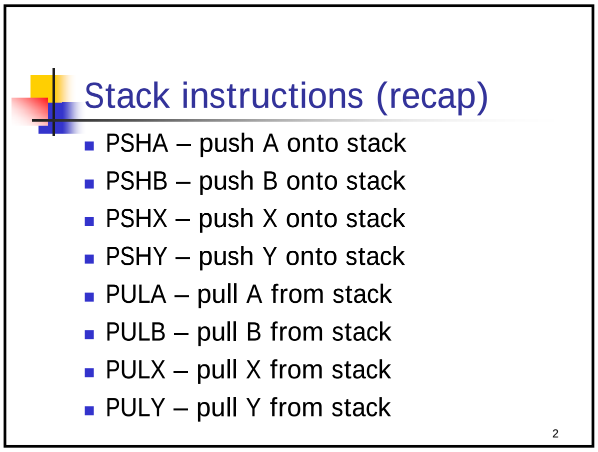


Some uses for the stack

- Short term data storage
  - vs. heap storage – new (C++) / malloc (C)
  - Preserving registers during calculations
    - Some operations only work on a particular register
  - Evaluating expressions with multiple operators (e.g., 5 * 3 + 6 % 2)
- Context
  - Subroutines (return address, arguments)
  - Interrupts (return address, processor state)

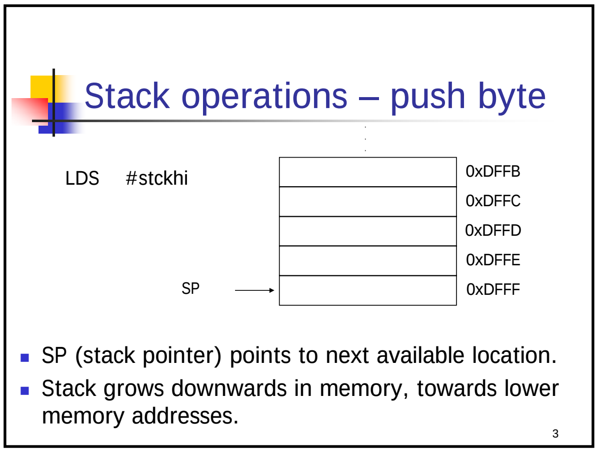
1



Stack instructions (recap)

- PSHA – push A onto stack
- PSHB – push B onto stack
- PSHX – push X onto stack
- PSHY – push Y onto stack
- PULA – pull A from stack
- PULB – pull B from stack
- PULX – pull X from stack
- PULY – pull Y from stack

2

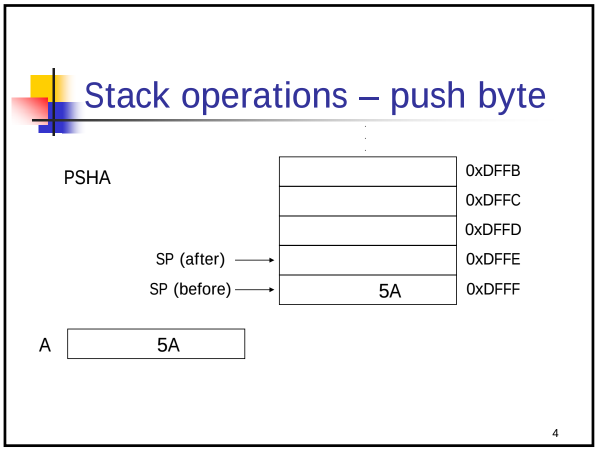


Stack operations – push byte

LDS #stckhi

- SP (stack pointer) points to next available location.
- Stack grows downwards in memory, towards lower memory addresses.

3

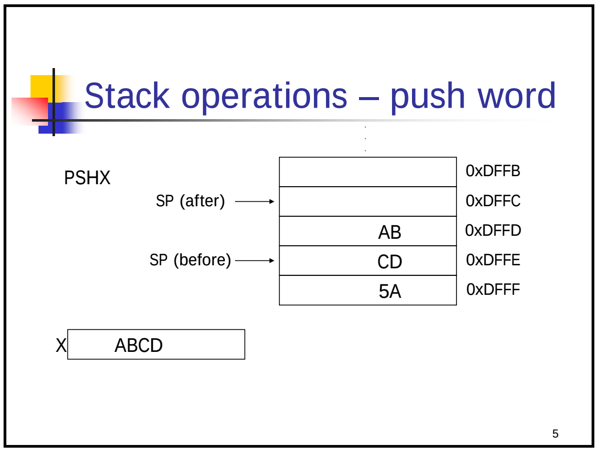


Stack operations – push byte

PSHA

		0xDFFB
		0xDFFC
		0xDFFD
SP (after) →		0xDFFE
SP (before) →	5A	0xDFFF

A 5A

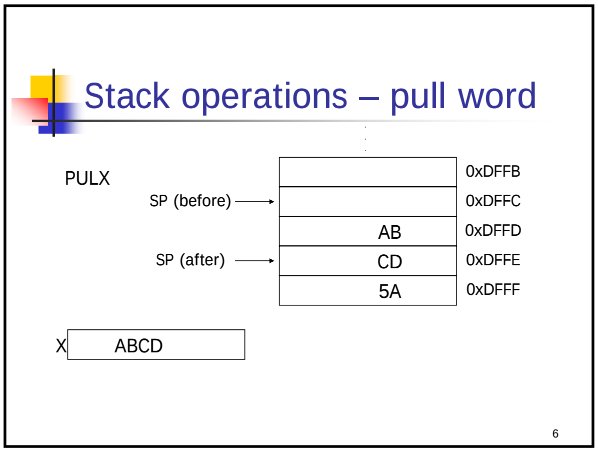


Stack operations – push word

PSHX

		0xDFFB
SP (after) →		0xDFFC
	AB	0xDFFD
SP (before) →	CD	0xDFFE
	5A	0xDFFF

X ABCD



Stack operations – pull word

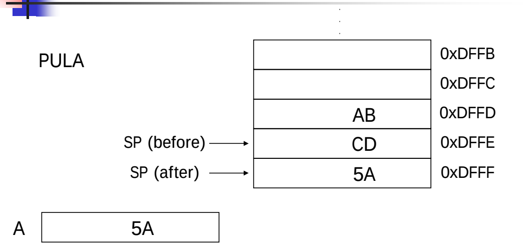
PULX

		0xDFFB
SP (before) →		0xDFFC
	AB	0xDFFD
SP (after) →	CD	0xDFFE
	5A	0xDFFF

X ABCD

Stack operations – pull byte

PULA



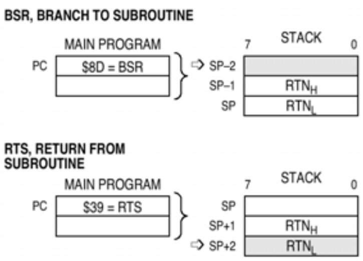
		0xDFFB
		0xDFFC
	AB	0xDFFD
SP (before) →	CD	0xDFFE
SP (after) →	5A	0xDFFF

A 5A

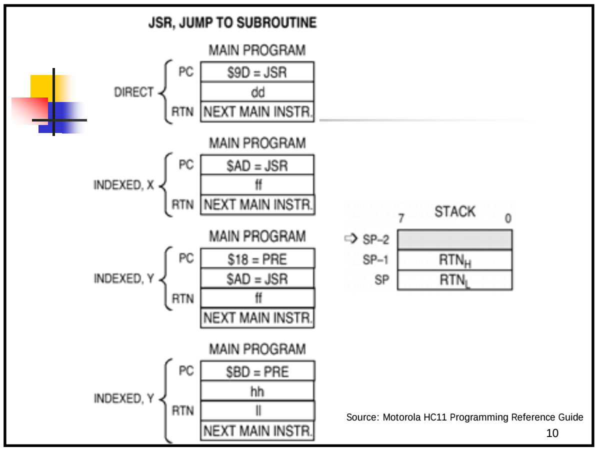
Stack pointer instructions

- LDS – load stack pointer
- STS – store stack pointer
- INS – increment stack pointer
- DES – decrement stack pointer
- TSX – transfer stack pointer+1 to X
- TSY – transfer stack pointer+1 to Y
- TXS – transfer X-1 to stack pointer
- TYS – transfer Y-1 to stack pointer

Subroutines

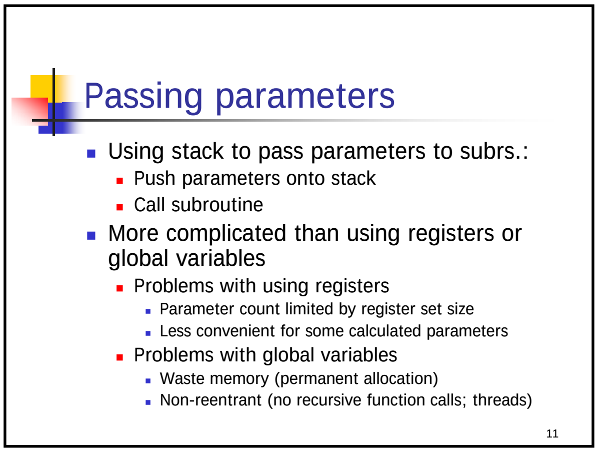


Source: Motorola HC11 Programming Reference Guide



Source: Motorola HC11 Programming Reference Guide

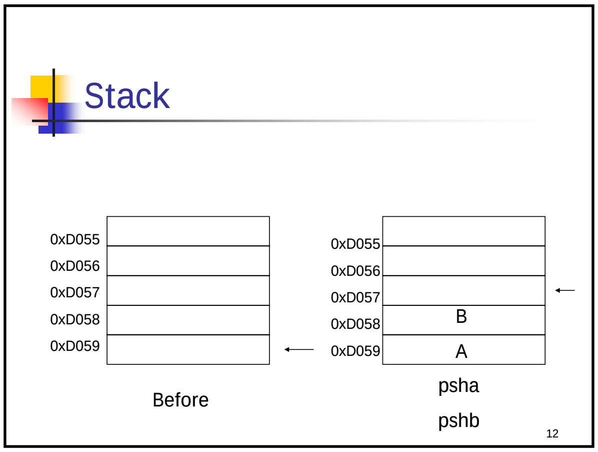
10



Passing parameters

- Using stack to pass parameters to subrs.:
  - Push parameters onto stack
  - Call subroutine
- More complicated than using registers or global variables
  - Problems with using registers
    - Parameter count limited by register set size
    - Less convenient for some calculated parameters
  - Problems with global variables
    - Waste memory (permanent allocation)
    - Non-reentrant (no recursive function calls; threads)

11



Stack

0xD055	
0xD056	
0xD057	
0xD058	
0xD059	

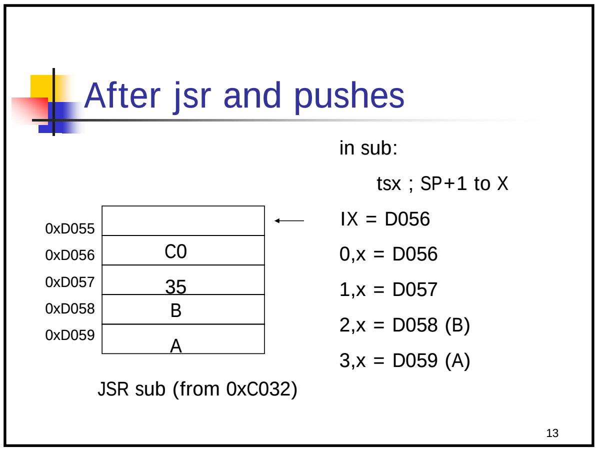
Before

0xD055	
0xD056	
0xD057	
0xD058	B
0xD059	A

psha

pshb

12

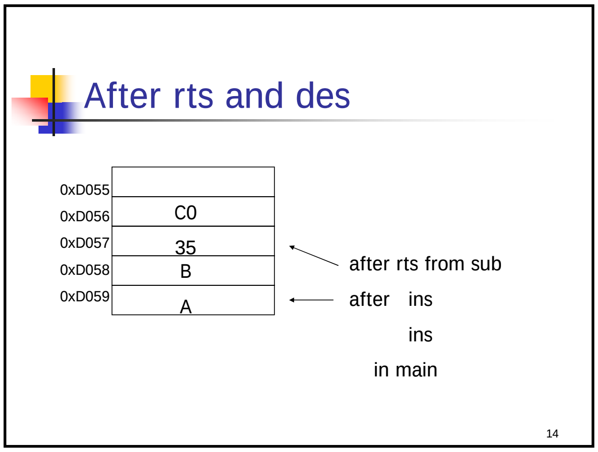


After jsr and pushes

0xD055	
0xD056	C0
0xD057	35
0xD058	B
0xD059	A

JSR sub (from 0xC032)

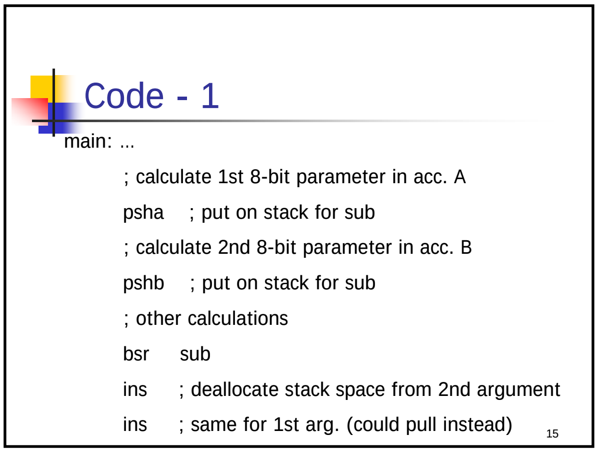
```
in sub:
      tsx ; SP+1 to X
IX = D056
0,x = D056
1,x = D057
2,x = D058 (B)
3,x = D059 (A)
```



After rts and des

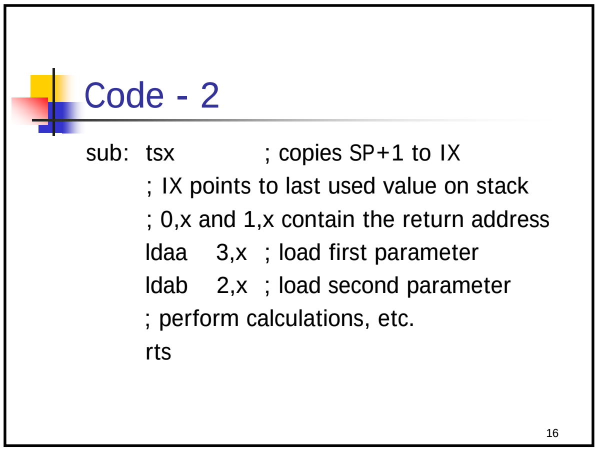
0xD055	
0xD056	C0
0xD057	35
0xD058	B
0xD059	A

```
after rts from sub
after   ins
        ins
    in main
```



Code - 1

```
main: ...
        ; calculate 1st 8-bit parameter in acc. A
        psha    ; put on stack for sub
        ; calculate 2nd 8-bit parameter in acc. B
        pshb    ; put on stack for sub
        ; other calculations
        bsr     sub
        ins     ; deallocate stack space from 2nd argument
        ins     ; same for 1st arg. (could pull instead)
```



Code - 2

```
sub:  tsx            ; copies SP+1 to IX
      ; IX points to last used value on stack
      ; 0,x and 1,x contain the return address
      ldaa   3,x   ; load first parameter
      ldab   2,x   ; load second parameter
      ; perform calculations, etc.
      rts
```