# Subroutines

- A subroutine is a piece of code that can be called **by NAME** from one or more locations in a program whenever its functionality is required

```
*main program
...
<call> subroutine
<next instruction>
...
<call> subroutine
<next instruction>
...
*end main program
```

```
*subroutine
<first instruction>
...
<return>
*end of subroutine
```

# Subroutines

- A subroutine is a piece of code that can be called **by NAME** from one or more locations in a program whenever its functionality is required

```
*main program
...
<call> subroutine
<next instruction>
...
<call> subroutine
<next instruction>
...
*end main program
```

```
*subroutine
<first instruction>
...
<return>
*end of subroutine
```

# Subroutines

- A subroutine is a piece of code that can be called **by NAME** from one or more locations in a program whenever its functionality is required

```
*main program
...
<call> subroutine
<next instruction>
...
<call> subroutine
<next instruction>
...
*end main program
```

```
*subroutine
<first instruction>
...
<return>
*end of subroutine
```

# Subroutines

- A subroutine is a piece of code that can be called **by NAME** from one or more locations in a program whenever its functionality is required

```
*main program
...
<call> subroutine
<next instruction>
...
<call> subroutine
<next instruction>
...
*end main program
```

```
*subroutine
<first instruction>
...
<return>
*end of subroutine
```

# Subroutines

- A subroutine is a piece of code that can be called **by NAME** from one or more locations in a program whenever its functionality is required

```
*main program
...
<call> subroutine
<next instruction>
...
<call> subroutine
<next instruction>
...
*end main program
```

```
*subroutine
<first instruction>
...
<return>
*end of subroutine
```

# Subroutines

- A subroutine is a piece of code that can be called **by NAME** from one or more locations in a program whenever its functionality is required

```
*main program
...
<call> subroutine
<next instruction>
...
<call> subroutine
<next instruction>
...
*end main program
```

```
*subroutine
<first instruction>
...
<return>
*end of subroutine
```

# Subroutines

- A subroutine is a piece of code that can be called **by NAME** from one or more locations in a program whenever its functionality is required

```
*main program
...
<call> subroutine
<next instruction>
...
<call> subroutine
<next instruction>
...
*end main program
```

```
*subroutine
<first instruction>
...
<return>
*end of subroutine
```

# Subroutines

- A subroutine is a piece of code that can be called **by NAME** from one or more locations in a program whenever its functionality is required

```
*main program
...
<call> subroutine
<next instruction>
...
<call> subroutine
<next instruction>
...
*end main program
```

```
*subroutine
<first instruction>
...
<return>
*end of subroutine
```

# Subroutines

- A subroutine is a piece of code that can be called **by NAME** from one or more locations in a program whenever its functionality is required

```
*main program
...
<call> subroutine
<next instruction>
...
<call> subroutine
<next instruction>
...
*end main program
```

```
*subroutine
<first instruction>
...
<return>
*end of subroutine
```

# Subroutines

- A subroutine is a piece of code that can be called **by NAME** from one or more locations in a program whenever its functionality is required

```
*main program
...
<call> subroutine
<next instruction>
...
<call> subroutine
<next instruction>
...
*end main program
```

```
*subroutine
<first instruction>
...
<return>
*end of subroutine
```

# Subroutines

- A subroutine is a piece of code that can be called **by NAME** from one or more locations in a program whenever its functionality is required

```
*main program
...
<call> subroutine
<next instruction>
...
<call> subroutine
<next instruction>
...
*end main program
```

```
*subroutine
<first instruction>
...
<return>
*end of subroutine
```

# Subroutines

- A subroutine is a piece of code that can be called **by NAME** from one or more locations in a program whenever its functionality is required

```
*main program
...
<call> subroutine
<next instruction>
...
<call> subroutine
<next instruction>
...
*end main program
```

```
*subroutine
<first instruction>
...
<return>
*end of subroutine
```

# Call and Return using an Address Register

```
caller:
          lea next,a1
          jmp subroutine
next      ...
```

```
callee:

subroutine   ...
             jmp (a1)
```

# Call and Return using an Address Register

```
caller:
          lea next,a1
          jmp subroutine
next      ...
```

```
callee:

subroutine   ...
             jmp (a1)
```

# Call and Return using an Address Register

```
caller:
          lea next,a1
          jmp subroutine
next      ...
```

```
callee:

subroutine   ...
             jmp (a1)
```

# Call and Return using an Address Register

```
caller:
          lea next,a1
          jmp subroutine
next      ...
```

```
callee:

subroutine   ...
             jmp (a1)
```

# Call and Return using an Address Register

```
caller:
          lea next,a1
          jmp subroutine
next      ...
```

```
callee:

subroutine   ...
             jmp (a1)
```

# Call and Return using an Address Register

```
caller:
          lea next,a1
          jmp subroutine
next      ...
```

```
callee:

subroutine   ...
             jmp (a1)
```

- Pros
  - simple, fast
- Cons
  - forces programmer to keep track of return address
  - can't perform nested (or recursive) calls

# Call and Return Using a Mailbox

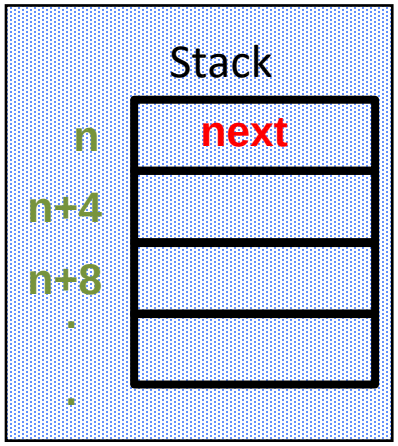
```
caller:

mailbox   ds.l    1
          move.l  #next,mailbox
          jmp     subroutine
next      ...
```

```
callee:

subroutine   ...
             movea.l mailbox,a1
             jmp (a1)
```

- Pros
  - allows for nested calls
- Cons
  - slow compared to register approach
  - can't perform recursive calls
  - Memory space grows with number of functions (not with level of nesting)

# Call and Return Using a Stack

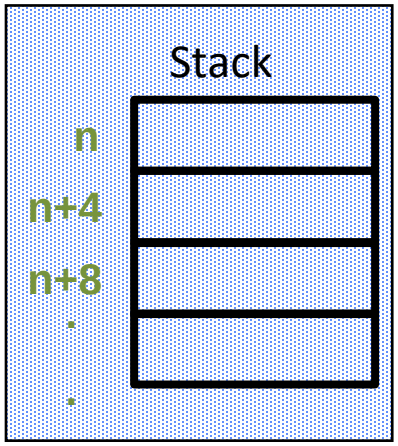
```
caller:

          move.l  #next,-(SP)
          jmp     subroutine
next      ...
```




```
callee:

subroutine   ...
             movea.l  (sp)+,a1
             jmp  (a1)
```

# Call and Return Using a Stack

```
caller:

          move.l  #next,-(SP)
          jmp     subroutine
next      ...
```




```
callee:

subroutine   ...
             movea.l (sp)+,a1
             jmp (a1)
```

- Pros
  - allows nested and recursive calls
  - space grows with level of nesting
- Cons
  - slow compared to register approach

# Jump to Subroutine Instruction

**JSR Jump to subroutine**

Syntax: `JSR <ea>`

Operation: SP = SP – 4
Memory[SP] = Program Counter
Program Counter = <ea>

| Dn | An | (An) | (An)+ | -(An) | d(An) | d(An,Xn) | ABS.W | ABS.L | Imm |
|---|---|---|---|---|---|---|---|---|---|
| | | ✓ | | | ✓ | ✓ | ✓ | ✓ | |

# Branch to Subroutine

## BSR Branch to Subroutine

Syntax: `BRS <label>`
`BRS <literal>`

| Name | Displacement | Machine Language | Operation Performed |
|---|---|---|---|
| BRS.S | 8-bit | 61XX | SP ← SP – 4<br>Memory[SP] = PC<br>PC ← PC + displacement |
| BRS.L | 16-bit | 6100 XXXX | |

# Return from Subroutine Instruction

**RTS Return from subroutine**

Syntax: `RTS`

Operation: Program Counter = Memory[SP]
SP = SP + 4

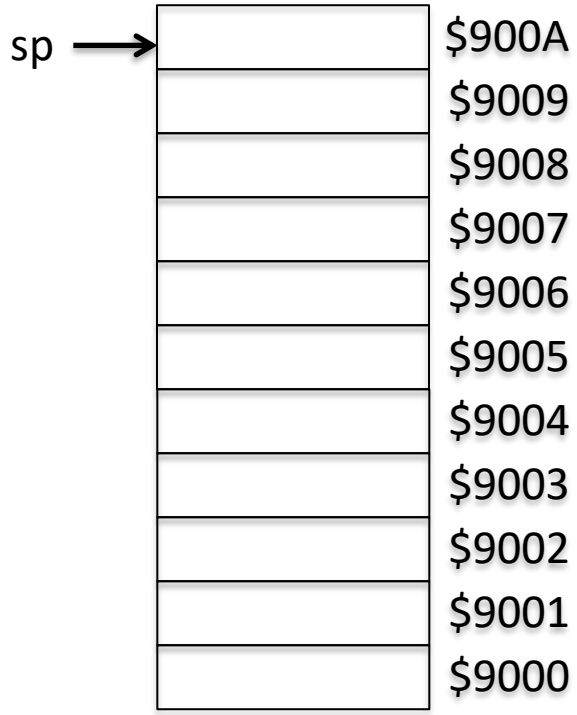
# Example

- Write a subroutine to calculate $2X^2$
  - Assumptions
    - Subroutine is to be called SQR
    - X is a 16-bit signed value
    - X is to be passed to the subroutine using D0
    - $2X^2$ is to be returned to the subroutine using D0

# Trace

Memory

```
8000            org   $8000
8000            move  #4,d0
8004            jsr   sqr
8008            nop


*** subroutine SQR ***

800A     sqr    muls   d0,d0
800C            asl.l  #1,d0
800E            rts
```

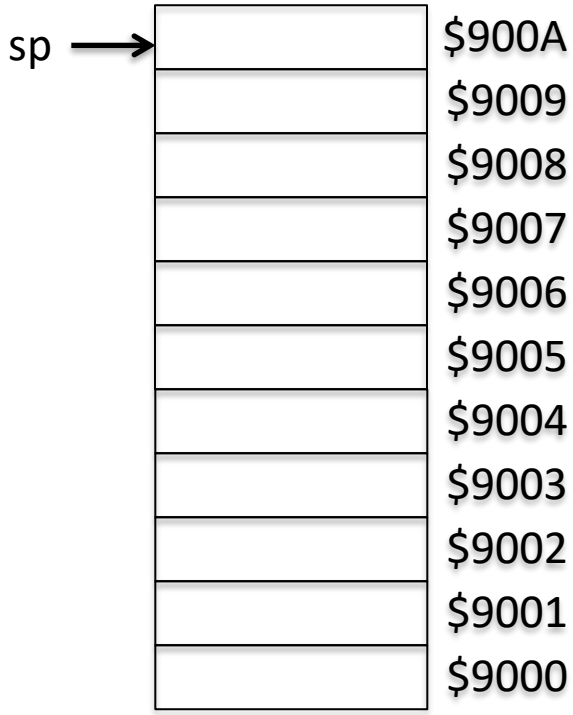



| | | | | |
|---|---|---|---|---|
| pc | 00 | 00 | 80 | 00 |
| d0 | XX | XX | XX | XX |
| a7 | 00 | 00 | 90 | 0A |

# Trace

Memory

```
8000            org    $8000
8000            move   #4,d0
8004            jsr    sqr
8008            nop


*** subroutine SQR ***

800A     sqr    muls    d0,d0
800C            asl.l   #1,d0
800E            rts
```

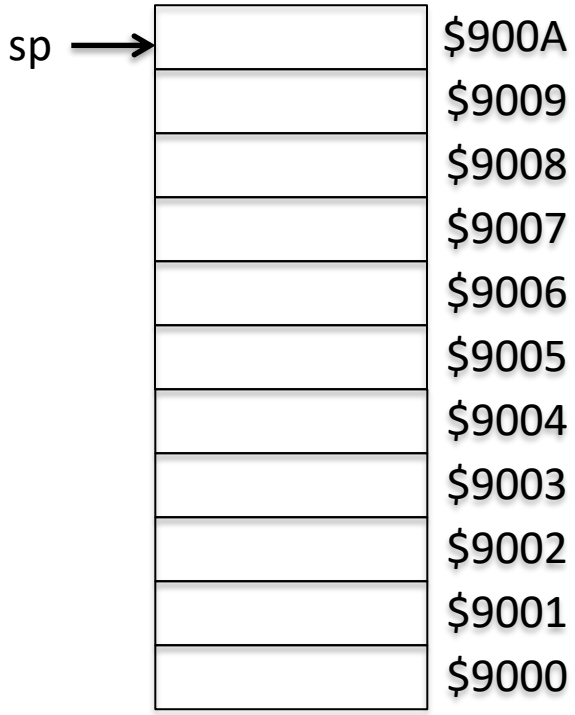



| | | | | |
|---|---|---|---|---|
| pc | 00 | 00 | 80 | 00 |
| d0 | XX | XX | XX | XX |
| a7 | 00 | 00 | 90 | 0A |

# Trace

Memory

```
8000            org   $8000
8000            move  #4,d0
8004            jsr   sqr
8008            nop


*** subroutine SQR ***

800A     sqr    muls   d0,d0
800C            asl.l  #1,d0
800E            rts
```

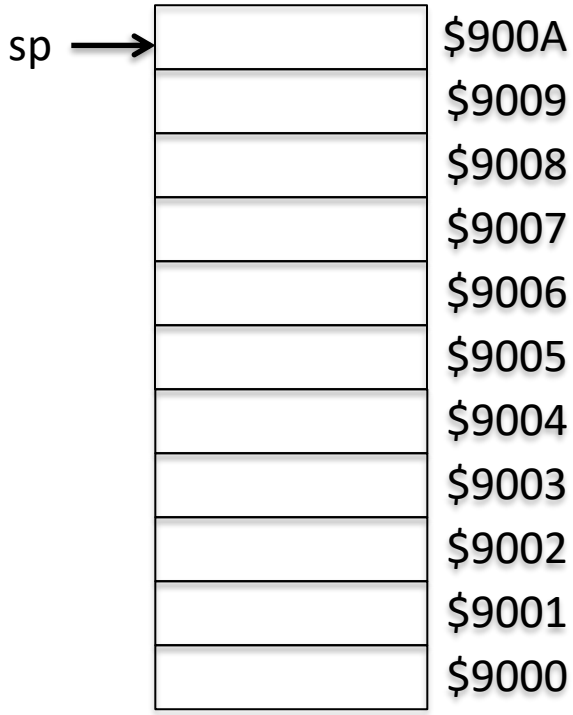



| | | | | |
|---|---|---|---|---|
| pc | 00 | 00 | 80 | 04 |
| d0 | XX | XX | 00 | 04 |
| a7 | 00 | 00 | 90 | 0A |

# Trace

Memory

```
8000            org   $8000
8000            move  #4,d0
8004            jsr   sqr
8008            nop


*** subroutine SQR ***

800A     sqr    muls   d0,d0
800C            asl.l  #1,d0
800E            rts
```




| | | | | |
|---|---|---|---|---|
| pc | 00 | 00 | 80 | 08 |

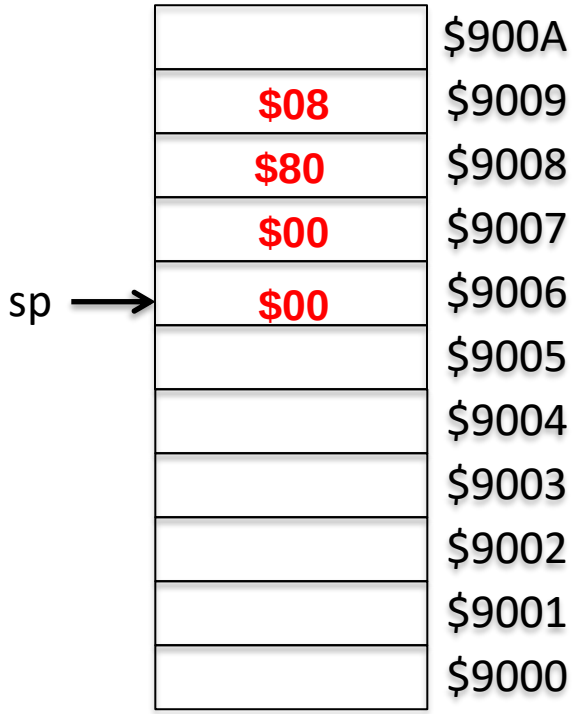
| | | | | |
|---|---|---|---|---|
| d0 | XX | XX | 00 | 04 |

| | | | | |
|---|---|---|---|---|
| a7 | 00 | 00 | 90 | 0A |

# Trace

Memory

```
8000            org   $8000
8000            move  #4,d0
8004            jsr   sqr
8008            nop


*** subroutine SQR ***

800A     sqr    muls   d0,d0
800C            asl.l  #1,d0
800E            rts
```

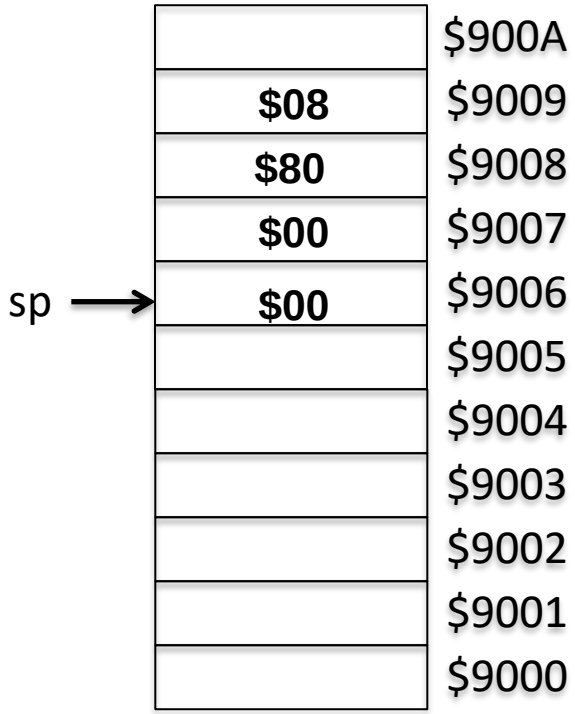



| | | | | |
|---|---|---|---|---|
| pc | 00 | 00 | 80 | 08 |
| d0 | XX | XX | 00 | 04 |
| a7 | 00 | 00 | 90 | 06 |

# Trace

Memory

```
8000            org   $8000
8000            move  #4,d0
8004            jsr   sqr
8008            nop


*** subroutine SQR ***

800A    sqr     muls   d0,d0
800C            asl.l  #1,d0
800E            rts
```

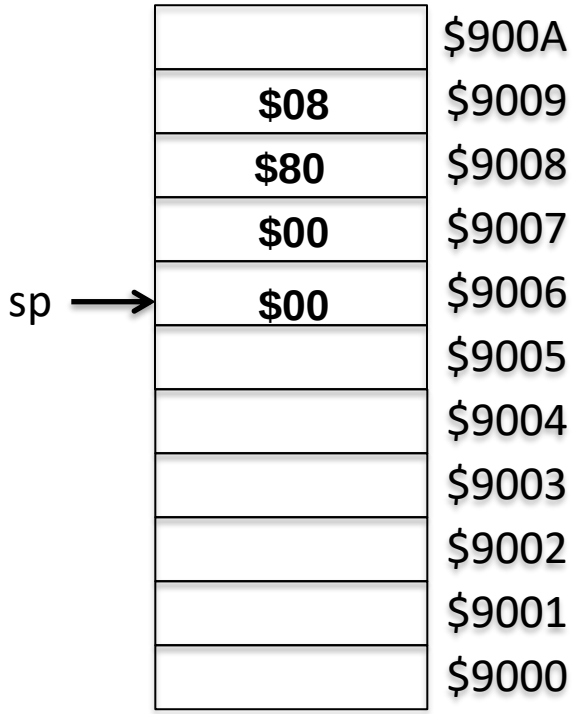



| | | | | |
|---|---|---|---|---|
| pc | 00 | 00 | 80 | 0A |
| d0 | XX | XX | 00 | 04 |
| a7 | 00 | 00 | 90 | 06 |

# Trace

Memory

```
8000            org   $8000
8000            move  #4,d0
8004            jsr   sqr
8008            nop


*** subroutine SQR ***

800A    sqr     muls   d0,d0
800C            asl.l  #1,d0
800E            rts
```

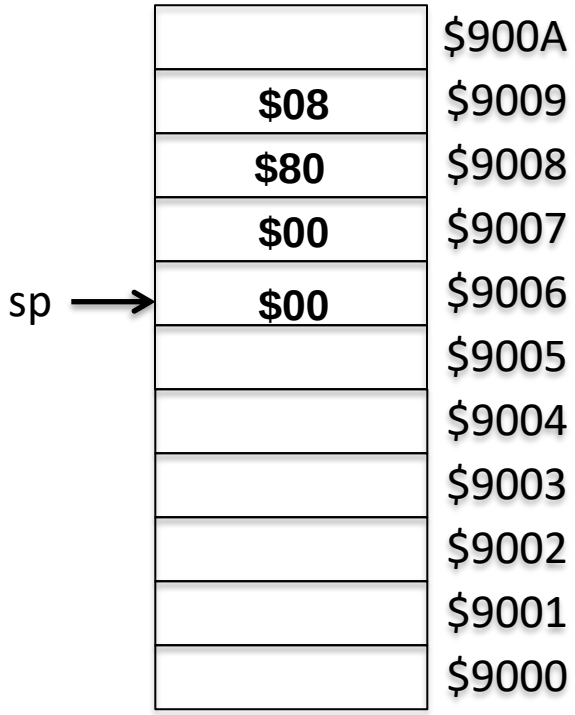



| | | | | |
|---|---|---|---|---|
| pc | 00 | 00 | 80 | 0A |
| d0 | XX | XX | 00 | 04 |
| a7 | 00 | 00 | 90 | 06 |

# Trace

Memory

```
8000            org    $8000
8000            move   #4,d0
8004            jsr    sqr
8008            nop


*** subroutine SQR ***

800A     sqr    muls    d0,d0
800C            asl.l   #1,d0
800E            rts
```

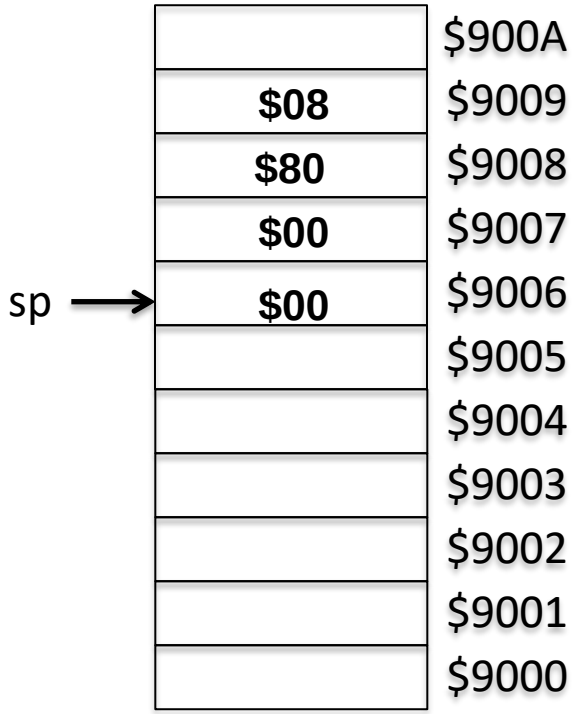



| | | | | |
|---|---|---|---|---|
| pc | 00 | 00 | 80 | 0A |
| d0 | 00 | 00 | 00 | 10 |
| a7 | 00 | 00 | 90 | 06 |

# Trace

Memory

```
8000            org    $8000
8000            move   #4,d0
8004            jsr    sqr
8008            nop


*** subroutine SQR ***

800A     sqr    muls    d0,d0
800C            asl.l   #1,d0
800E            rts
```




| pc | 00 | 00 | 80 | 0C |
|---|---|---|---|---|

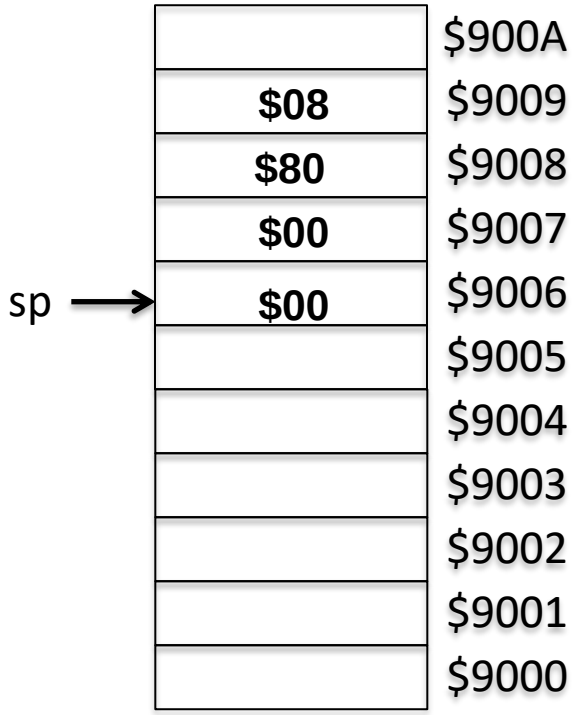
| d0 | 00 | 00 | 00 | 10 |
|---|---|---|---|---|

| a7 | 00 | 00 | 90 | 06 |
|---|---|---|---|---|

# Trace

Memory

```
8000            org    $8000
8000            move   #4,d0
8004            jsr    sqr
8008            nop


*** subroutine SQR ***

800A     sqr    muls    d0,d0
800C            asl.l   #1,d0
800E            rts
```




| pc | 00 | 00 | 80 | 0E |
|---|---|---|---|---|

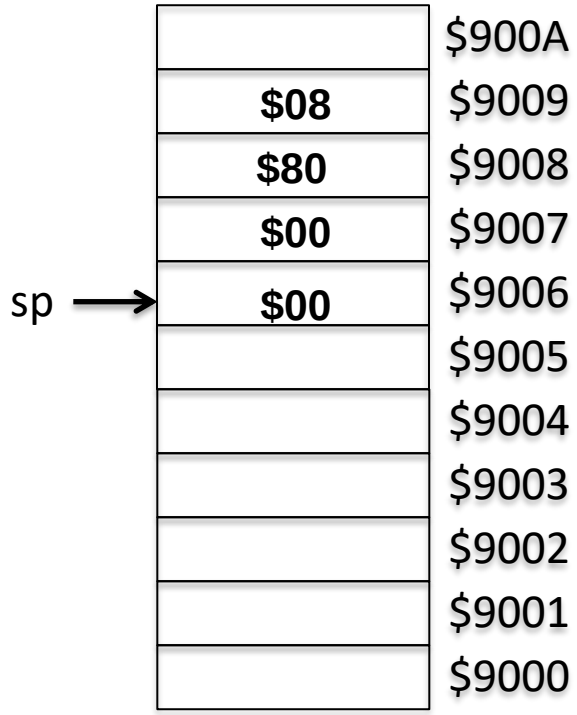
| d0 | 00 | 00 | 00 | 20 |
|---|---|---|---|---|

| a7 | 00 | 00 | 90 | 06 |
|---|---|---|---|---|

# Trace

Memory

```
8000            org    $8000
8000            move   #4,d0
8004            jsr    sqr
8008    nc      nop


*** subroutine SQR ***

800A    sqr     muls    d0,d0
800C            asl.l   #1,d0
800E            rts
```

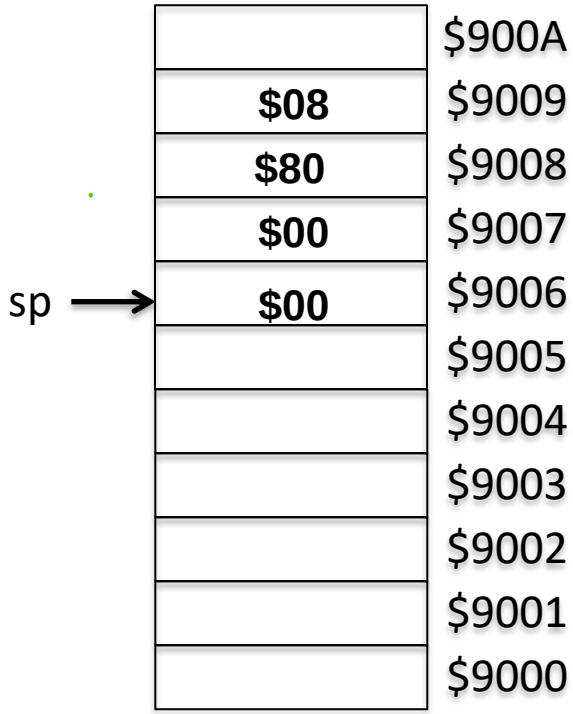



| | | | | |
|---|---|---|---|---|
| pc | 00 | 00 | 80 | 10 |
| d0 | 00 | 00 | 00 | 20 |
| a7 | 00 | 00 | 90 | 06 |

# Trace

Memory

```
8000            org    $8000
8000            move   #4,d0
8004            jsr    sqr
8008            nop


*** subroutine SQR ***

800A     sqr    muls    d0,d0
800C            asl.l   #1,d0
800E            rts
```




| pc | 00 | 00 | 80 | 08 |
|---|---|---|---|---|

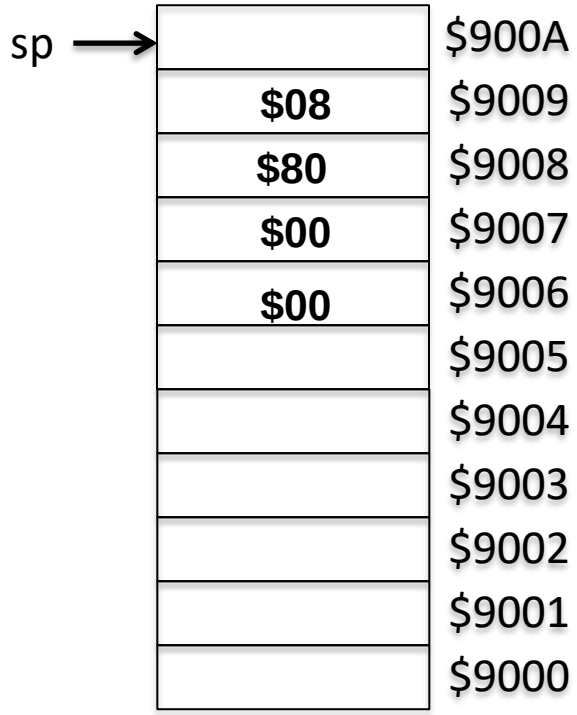
| d0 | 00 | 00 | 00 | 20 |
|---|---|---|---|---|

| a7 | 00 | 00 | 90 | 06 |
|---|---|---|---|---|

# Trace

Memory

```
8000            org   $8000
8000            move  #4,d0
8004            jsr   sqr
8008            nop


*** subroutine SQR ***

800A     sqr    muls   d0,d0
800C            asl.l  #1,d0
800E            rts
```

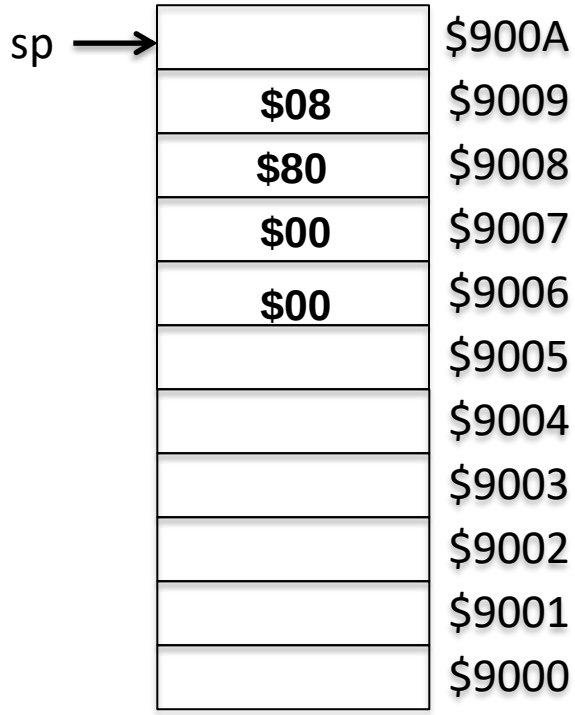



| | | | | |
|---|---|---|---|---|
| pc | 00 | 00 | 80 | 08 |
| d0 | 00 | 00 | 00 | 20 |
| a7 | 00 | 00 | 90 | 0A |

# Trace

Memory

```
8000            org   $8000
8000            move  #4,d0
8004            jsr   sqr
8008            nop


*** subroutine SQR ***

800A     sqr    muls   d0,d0
800C            asl.l  #1,d0
800E            rts
```




| pc | 00 | 00 | 80 | 08 |
|---|---|---|---|---|

| d0 | 00 | 00 | 00 | 20 |
|---|---|---|---|---|

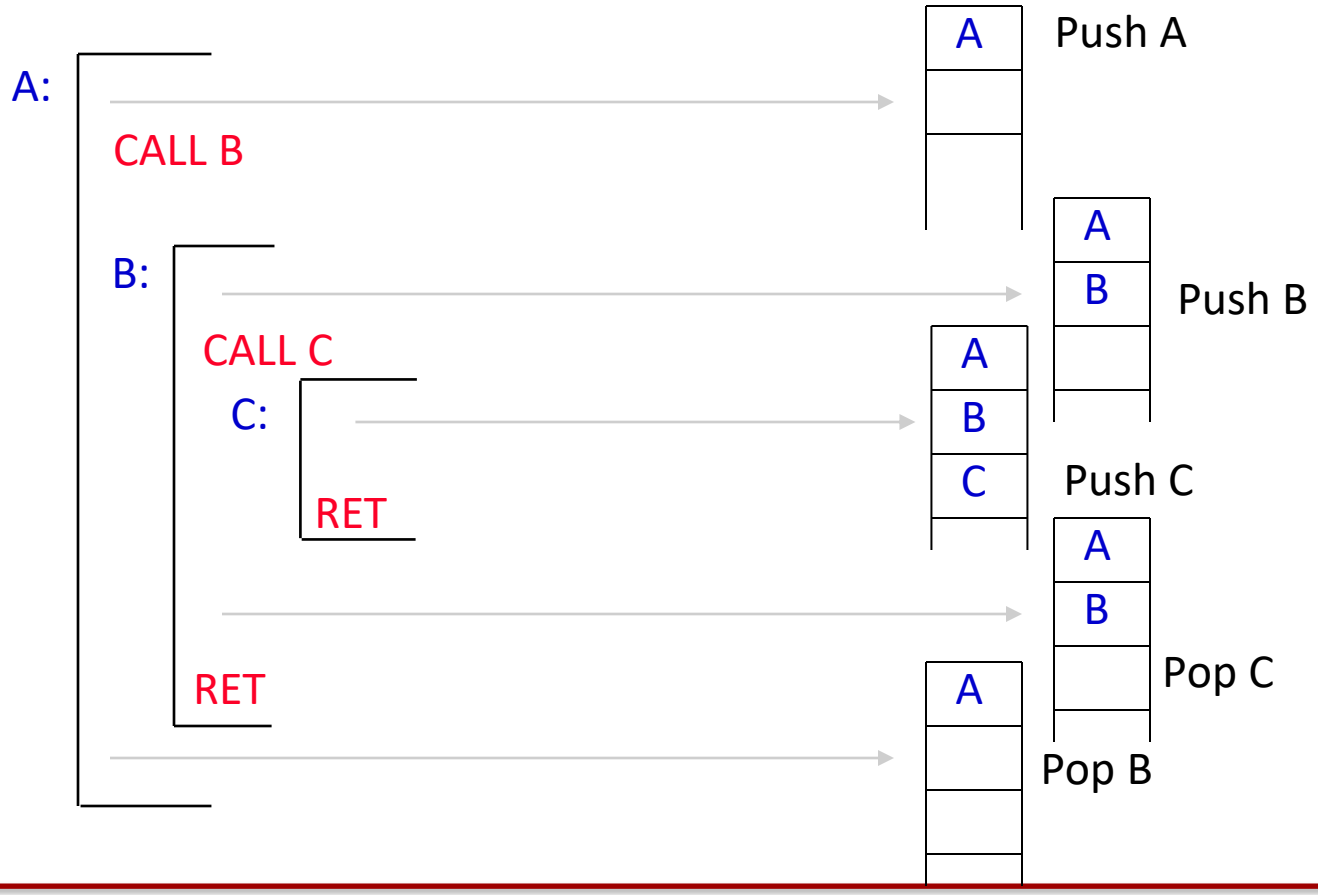
| a7 | 00 | 00 | 90 | 0A |
|---|---|---|---|---|

# Nested Subroutine (Stack) Calls

- The stack grows downward and shrinks upward to accommodate nested subroutine calls

# Nested Subroutine (Stack) Calls

- The stack grows downward and shrinks upward to accommodate nested subroutine calls



# Summary

- A subroutine is analogous to a C function
  - a piece of code that can be called **by NAME**
  - can be called from one or more locations in a program whenever its functionality is required
- Stack is preferred way to store return address
  - Allows for nested function calls
  - Allows for recursive function calls
  - Allows amount of memory for return address to grown proportional to number of nested/recursive calls
- JSR and BSR are used to call a subroutine
  - both have the same purpose, to transfer control to the beginning of the called subroutine and to save the return address on the runtime stack
  - Each differ in how they specify the starting address of the subroutine
- RTS are used to return to the calling code
  - Restores the PC from the runtime stack (but must be careful!)