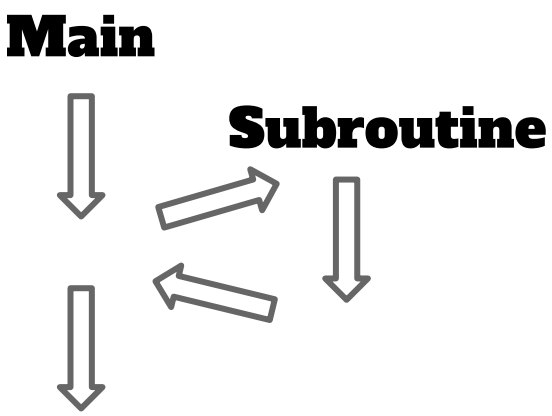
Subroutines

CSC 236

Subroutine



Subroutine

- Definition:
  - a sequence of program instructions that performs a specific task, packaged as a unit
- AKA
  - Function
  - Procedure
  - Method

Characteristics

- Self-contained (can be compiled/assembled separately)
- Called, executes, return
- Appears as a statement or expression to the caller
  - `y = sqrt(x)`
  - `printf( “Hello CSC236\n” );`

Advantages

- Breaks large problem into small pieces
- Modularity
- Potential for reuse

Prerequisites

- Standardization
- Protocol for communication

Protocols

- Needed for subroutine
- Define interface
  - Invoke
  - Initialize
  - Return
- Pass parameters
- Share resources
  - Registers
  - Memory (stack)

Registers

- Caller saves
  - Before invoking subroutine
    - Save all “live” data to memory
  - After
    - Restore live data from memory
- Callee saves
  - At beginning:
    - Save registers subroutine will use
  - Before returning
    - Restore saved registers

Hybrid

- Which is better?
  - Ideally, we'd like fewest saved registers
- Callee saved
  - For a simple subroutine, maybe we just have to save a few registers.
- Caller saved
  - If the caller is using few registers, subroutine overhead is reduced.
- All registers are in one class or the other:
  - Responsibility of the caller to save (if needed)
  - Responsibility of the callee to save (if used)

Parameter passing options

- Register
  - Fast
  - Limited size / number
- Global memory locations
  - Fast (but slower than registers)
  - Static (single/shared)
  - Hard to debug
- Stack
  - Practically unlimited size / number
  - Re-entrant not shared across calls
  - Permits recursion

Concept

```
;program: calls x=sqrt(y)
;main

mov ax,[y]
call sqrt
```

Assume:

- **In: ax=y (parameter)**
- **Out: ax=x (square root)**
- **Subroutine uses: ax, bx & cx**

Concept

```
;program: calls x=sqrt(y)
;main

mov ax,[y]
call sqrt
```

```
;sqrt(y)
sqrt:
  push bx          ;save bx
  push cx          ;save cx
```

Callee saved registers

Concept

```
;program: calls x=sqrt(y)
;main

mov ax,[y]
call sqrt
```

```
;sqrt(y)
sqrt:
  push bx          ;save bx
  push cx          ;save cx

  mov bx, 61       ;pretend
  mov cx, 11       ;pretend
```

Body of the subroutine

Concept

```
;program: calls x=sqrt(y)
;main

mov ax,[y]
call sqrt
```

```
;sqrt(y)
sqrt:
  push bx          ;save bx
  push cx          ;save cx

  mov bx, 61       ;pretend
  mov cx, 11       ;pretend

  mov ax, 25       ;set root
```

Put return value in ax

Concept

```
;program: calls x=sqrt(y)
;main

mov ax,[y]
call sqrt
```

```
;sqrt(y)
sqrt:
  push bx          ;save bx
  push cx          ;save cx

  mov bx, 61       ;pretend
  mov cx, 11       ;pretend

  mov ax, 25       ;set root
  pop  cx          ;restore cx
  pop  bx          ;restore bx
```

Restore saved registers

Concept

```
;program: calls x=sqrt(y)
;main

mov ax,[y]
call sqrt
```

```
;sqrt(y)
sqrt:
  push bx          ;save bx
  push cx          ;save cx

  mov bx, 61       ;pretend
  mov cx, 11       ;pretend

  mov ax, 25       ;set root
  pop  cx          ;restore cx
  pop  bx          ;restore bx
```

Restore saved registers

Concept

```
;program: calls x=sqrt(y)
;main

mov ax,[y]
call sqrt
```

```
;sqrt(y)
sqrt:
  push bx          ;save bx
  push cx          ;save cx

  mov bx, 61       ;pretend
  mov cx, 11       ;pretend

  mov ax, 25       ;set root
  pop  cx          ;restore cx
  pop  bx          ;restore bx
  ret              ;return
```

Concept

```
;program: calls x=sqrt(y)
;main

mov ax,[y]
call sqrt
```

```
;sqrt(y)
sqrt:
  push bx          ;save bx
  push cx          ;save cx

  mov bx, 61       ;pretend
  mov cx, 11       ;pretend

  mov ax, 25       ;set root
  pop  cx          ;restore cx
  pop  bx          ;restore bx
  ret              ;return
```

4 new instructions

- **call/ret**
- **push/pop**

Call / ret

Call

- Syntax: call <label>
- Semantics
  - Pushes current PC (IP) on stack
  - Jumps to label
    - Sets PC to label

Return

- Syntax: ret
- Semantics
  - Pops return PC off the stack
  - Jumps to return PC

Push / pop

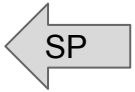
- Stack
  - Stack segment (SS)
  - Stack pointer (SP)
  - Word-sized elements
  - Grows **downward**
- Push
  - Syntax: push <reg|var>
  - Semantics
    - Decrement SP (by 2)
    - Copy word of data to [sp] (pointed to by sp)

Push / pop

- Stack
  - Stack segment (SS)
  - Stack pointer (SP)
  - Word-sized elements
  - Grows **downward**
- Push
  - Syntax: push <reg|var>
  - Semantics
    - Decrement SP (by 2)
    - Copy word of data to [sp] (pointed to by sp)

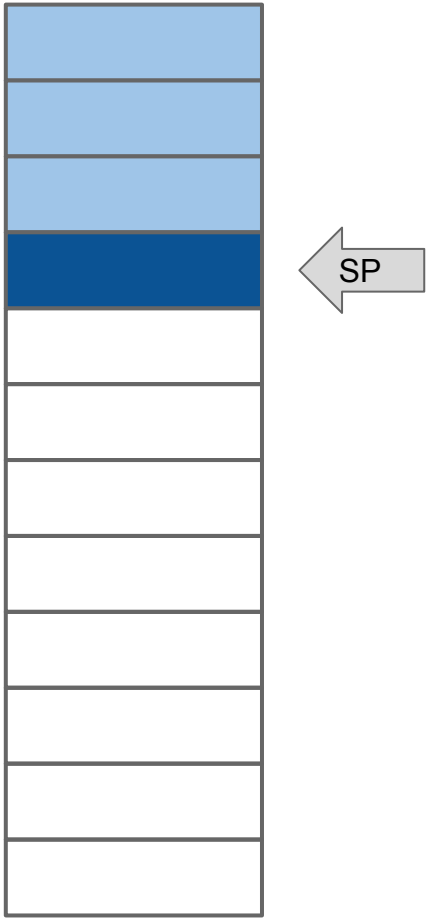
Push / pop

- Stack
  - Stack segment (SS)
  - Stack pointer (SP)
  - Word-sized elements
  - Grows **downward**
- Push
  - Syntax: push <reg|var>
  - Semantics
    - Decrement SP (by 2)
    - Copy word of data to [sp] (pointed to by sp)



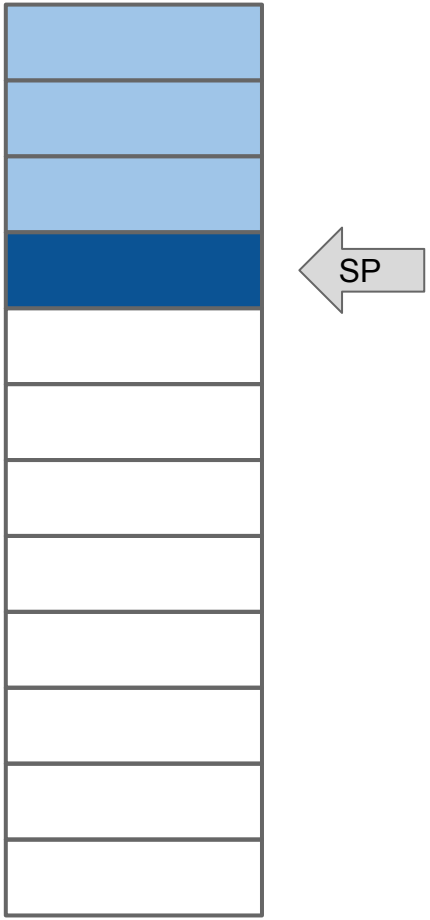
Push / pop

- Stack
  - Stack segment (SS)
  - Stack pointer (SP)
  - Word-sized elements
  - Grows **downward**
- Push
  - Syntax: push <reg|var>
  - Semantics
    - Decrement SP (by 2)
    - Copy word of data to [sp] (pointed to by sp)



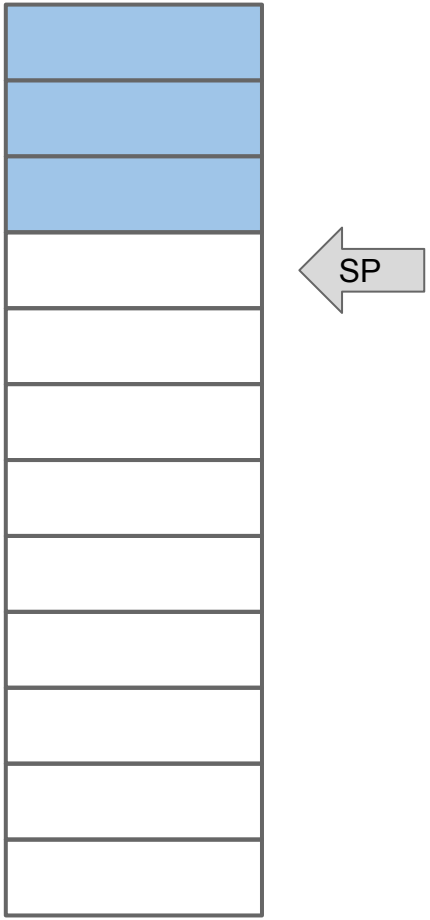
Push / pop

- Push
  - Syntax: push <reg|var>
  - Semantics
    - Decrement SP (by 2)
    - Copy word of data to [sp] (pointed to by sp)
- Pop
  - Syntax: pop <reg|var>
  - Semantics
    - Copy word of data from [sp]
    - Increment SP (by 2)



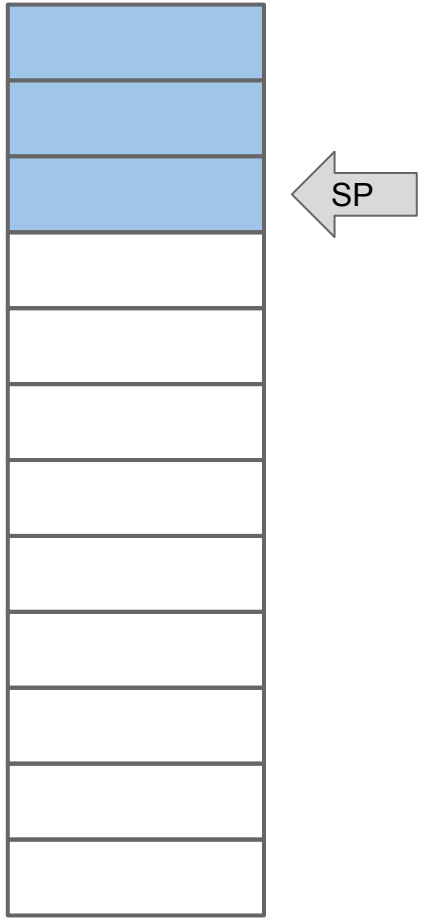
Push / pop

- Push
  - Syntax: push <reg|var>
  - Semantics
    - Decrement SP (by 2)
    - Copy word of data to [sp] (pointed to by sp)
- Pop
  - Syntax: pop <reg|var>
  - Semantics
    - Copy word of data from [sp]
    - Increment SP (by 2)



Push / pop

- Push
  - Syntax: push <reg|var>
  - Semantics
    - Decrement SP (by 2)
    - Copy word of data to [sp] (pointed to by sp)
- Pop
  - Syntax: pop <reg|var>
  - Semantics
    - Copy word of data from [sp]
    - Increment SP (by 2)



Non-reusable code

- Only works once
- Must re-load every time

```
  .data
count  dw  10
  .code
subr:

repit:
  dec   [count]
  jne   repit
  ret
```

Non-reusable code

- Only works once
- Must re-load every time

```
  .data
count   dw   10 0
  .code
subr:

repit:
  dec    [count]
  jne    repit
  ret
```

Serially reusable

- Code may be used again
- Sequentially

```
  .data
count  dw  ?
  .code
subr:

  mov   [count],10
repit:
  dec   [count]
  jne   repit
  ret
```

Serially reusable

- Code may be used again
- Sequentially

```
  .data
count  dw  10
  .code
subr:

  mov   [count],10
repit:
  dec   [count]
  jne   repit
  ret
```

Serially reusable

- Code may be used again
- Sequentially

```
  .data
count  dw  ~~10~~ 0
  .code
subr:

  mov   [count],10
repit:
  dec   [count]
  jne   repit
  ret
```

Reentrant

- May be called again, before it finishes
- May be used simultaneously by multiple tasks

Reentrant

- May be called again, before it finishes
- May be used simultaneously by multiple tasks

- Why is this useful?

Reentrant

- May be called again, before it finishes
- May be used simultaneously by multiple tasks

- Why is this useful?
  - Recursion
  - Parallelism

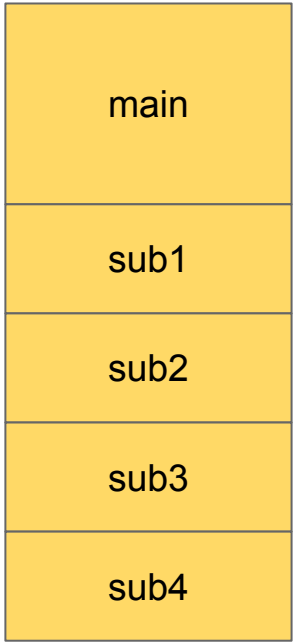
Reentrant

- May be called again, before it finishes
- May be used simultaneously by multiple tasks

- How
  - Unique variables for each activation
  - Dynamically built on stack

Single file

- Code for main
- Code for sub1
- Code for sub2
- Code for sub3
- Code for sub4
- Each may have code & data
- But variables/labels are global
- Main initializes the DS register
- One end directive (pointing to the start of main)



Single file

```
   .model small
   .8086
   .stack 256

   .data
n    dw   6
list dw   -3,7,100,-83,0,1000
   .code
start: mov ax,@data
       mov ds,ax

       mov cx,[n] = count
       mov si, offset list
       call set

exit:  mov ax,4c00h
       int 21h
;  end of the main data and code
```

```
   .data
zero dw      0
   .code
set:  push ax
      push si
      push cx
      . . .
      pop cx
      pop si
      pop ax
      ret

      end start
```

Separate files

main.asm

```
;main
...
call foo
...
```

sub.asm

```
;sub

foo:
    ...
```

- **Assemble separately**
- **Combined by linker**
- **No label “foo” in main**

Separate files

- Main file
  - .model .8086 .stack
  - Initializes DS register
  - end declaration
  - extrn — new directive
- Subroutine file
  - .model .8086
  - No stack
  - Do not initialize DS
  - No label for end declaration
  - public — new directive
- extrn
  - External
  - Label defined in different file
- Public
  - Identify label visible outside of file

Single file

```
   .model small
   .8086
   .stack 256

   .data
n    dw   6
list dw   -3,7,100,-83,0,1000
   .code
start: mov ax,@data
       mov ds,ax

       mov cx,[n] = count
       mov si, offset list
       call set

exit:  mov ax,4c00h
       int 21h
;  end of the main data and code
```

```
   .data
zero dw      0
   .code
set:  push ax
      push si
      push cx
      . . .
      pop cx
      pop si
      pop ax
      ret

      end start
```

Separate files

main.asm

```
        .model   small
        .8086
        extrn    set:proc
        .stack   256

        .data
n    dw     6
list dw     -3,7,100,-83,0,1000

        .code
start:  mov   ax,@data
        mov   ds,ax
        mov   cx,[n]
        mov   si, offset list
        call  set
exit:   mov   ax,4c00h
        int   21h
        end   start
```

sub.asm

```
        .model   small
        .8086
        public   set

        .data
zero dw      0

        .code
set:    push   ax
        push   si

        pop    si
        pop    ax
        ret
        end
```

Separate files

main.asm

```
        .model   small
        .8086
        extrn    set:proc
        .stack   256

        .data
n    dw     6
list dw     -3,7,100,-83,0,1000

        .code
start:  mov   ax,@data
        mov   ds,ax
        mov   cx,[n]
        mov   si, offset list
        call  set
exit:   mov   ax,4c00h
        int   21h
        end   start
```

link

sub.asm

```
        .model   small
        .8086
        public   set

        .data
zero dw      0

        .code
set:    push   ax
        push   si

        pop    si
        pop    ax
        ret
        end
```

Separate files

main.asm

```
        .model   small
        .8086
        extrn    set:proc
        .stack   256

        .data
n    dw     6
list dw     -3,7,100,-83,0,1000

        .code
start:  mov   ax,@data
        mov   ds,ax
        mov   cx,[n]
        mov   si, offset list
        call  set
exit:   mov   ax,4c00h
        int   21h
        end   start
```

stack ← | → no stack

sub.asm

```
        .model   small
        .8086
        public   set

        .data
zero dw      0

        .code
set:    push   ax
        push   si

        pop    si
        pop    ax
        ret
        end
```

Separate files

main.asm

```
        .model   small
        .8086
        extrn    set:proc
        .stack   256

        .data
n     dw     6
list dw      -3,7,100,-83,0,1000

        .code
start:  mov    ax,@data
        mov    ds,ax
        mov    cx,[n]
        mov    si, offset list
        call   set
exit:   mov    ax,4c00h
        int    21h
        end    start
```

stack

DS init

no stack

no DS init

sub.asm

```
        .model   small
        .8086
        public   set

        .data
zero dw      0

        .code
set:    push   ax
        push   si

        pop    si
        pop    ax
        ret
        end
```

Separate files

main.asm

```
        .model   small
        .8086
        extrn    set:proc
        .stack   256

        .data
n     dw     6
list dw      -3,7,100,-83,0,1000

        .code
start:  mov    ax,@data
        mov    ds,ax
        mov    cx,[n]
        mov    si, offset list
        call   set
exit:   mov    ax,4c00h
        int    21h
        end    start
```

stack

DS init

label

sub.asm

```
        .model   small
        .8086
        public   set

        .data
zero dw      0

        .code
set:    push   ax
        push   si

        pop    si
        pop    ax
        ret
        end
```

no stack

no DS init

no label