# CMPE 220

Class 5 – Assembly Language & Assemblers

# Using Pseudocode

**Fibonacci Sequence**

- Each number in the sequence is the sum of the previous two:
  - 0, 1, 1, 2, 3, 5, 8, 13, 21, 34…

- Write a short program to print a sequence of Fibonacci numbers
  - Also, indicate numbers that are evenly divisible by 5

# Output

**Fibonacci Sequence**

0: 0 Divisible by 5!
1: 1
2: 1
3: 2
4: 3
5: 5 Divisible by 5!
6: 8
7: 13
8: 21
9: 34
10: 55 Divisible by 5!
11: 89
12: 144
13: 233
14: 377
15: 610 Divisible by 5!
16: 987
17: 1597
18: 2584
19: 4181
20: 6765 Divisible by 5!
21: 10946
22: 17711
23: 28657
24: 46368
25: 75025 Divisible by 5!
26: 121393
27: 196418
28: 317811
29: 514229
30: 832040 Divisible by 5!

# Pseudocode

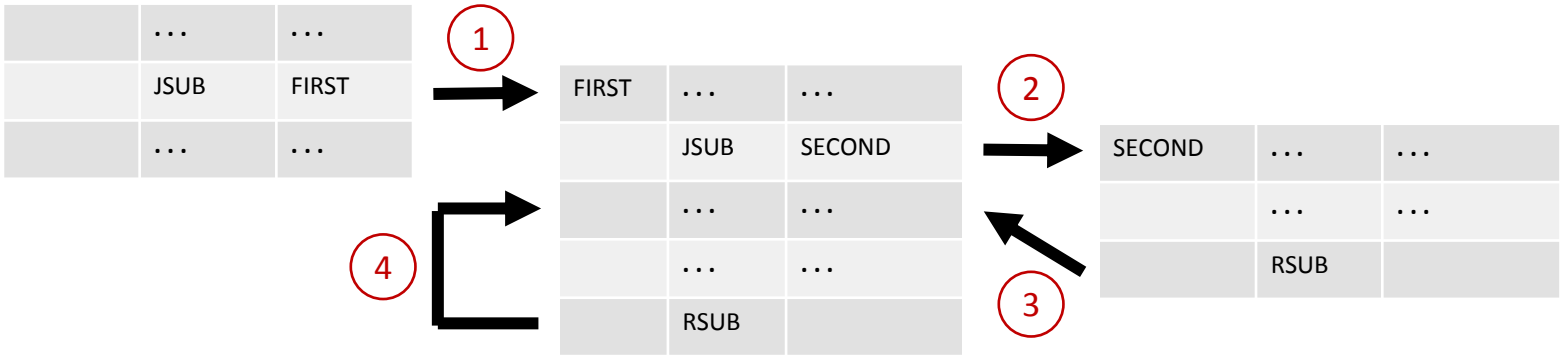
```
PRINT "Fibonacci Sequence”
linenumber = 0;
prev_prev = -1;
prev      = 1;
WHILE (linenumber <= 30)
{
   fib = prev_prev + prev;
   PRINT linenumber and fib
   IF (fib mod 5 is 0) PRINT “divisible by 5”
   PRINT new line
   i = i + 1;
   prev_prev = prev;
   prev = fib;
}
```

# Compilers Emit Assembly Language Code

- Sometimes length and complex instruction sequences are required to do things that are not built into the machine architecture

# EXAMPLE: Nesting Subroutines

- The SIC instruction set has a subroutine call – JSUB
- Because there is only one return address register (L), subroutines cannot be nested:

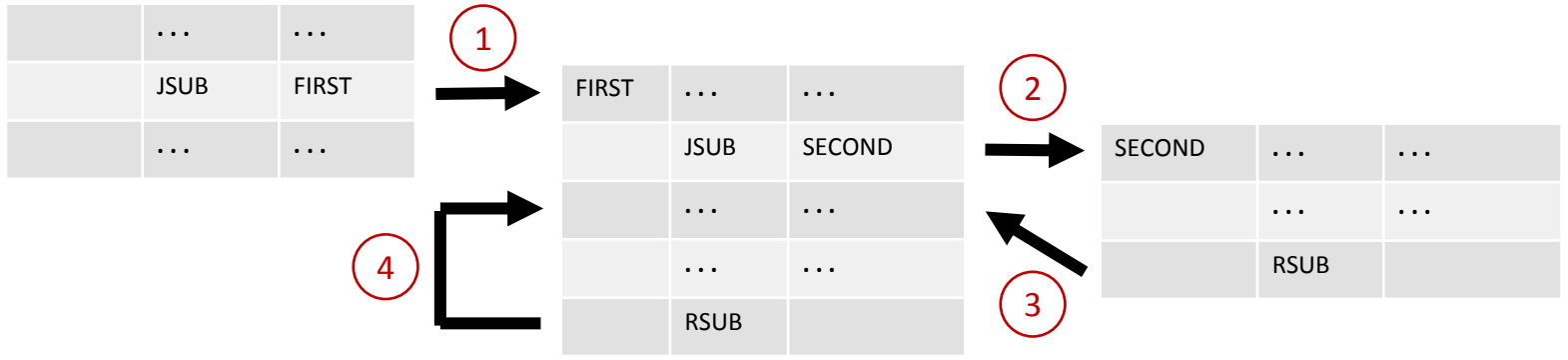


# A Subroutine Call Stack

- SIC/XE has JSUB and RSUB instructions, but only one L register
- Nested subroutine calls don’t work!

# EXAMPLE: Nesting Subroutines

- The SIC instruction set has a subroutine call – JSUB
- Because there is only one return address register (L), subroutines cannot be nested:



# Storage Declarations

| | | | |
|---|---|---|---|
| SSTACK | RESW | 10 | Allow 10 nested subroutine calls |
| SINDEX | WORD | 0 | Index |
| SMAX | WORD | 30 | Stack Maximum |
| SAVEA | RESW | 1 | Space to save A register (we'll see why) |
| SAVEX | RESW | 1 | Space to save X register (we'll see why) |

# Subroutine Call

| | | |
|---|---|---|
| JSUB | *+3 | Get address into L register |
| STA | SAVEA | Save the A register |
| STX | SAVEX | Save the X Register |
| LDA | SINDEX | Get the stack index in A |
| COMP | SMAX | See if we're at the end of stack |
| JEQ | ERROR | |
| RMO | L, A | Move return address into A register |
| ADD | 35 (?) | Increment address to point past this code |
| LDX | SINDEX | Get the stack index in X |
| STL | SSTACK,X | Use indexed addressing to store return address on stack |
| LDA | #3 | Increment stack index |
| ADDR | X, A | |
| STX | SINDEX | |
| LDA | SAVEA | Restore the A register |
| LDX | SAVEX | Restore the X register |
| J | SUB | Jump to the subroutine |
| • • • | • • • | This is where we will return |

# Subroutine Return

| | | |
|---|---|---|
| LDA | SINDEX | Decrement stack index |
| SUB | #3 | |
| STA | SINDEX | |
| LDX | SINDEX | |
| LDL | SPTR,X | Indexed load to get the return address into L register |
| LDA | SAVEA | Restore the A register |
| LDX | SAVEX | Restore the X register |
| RSUB | | |

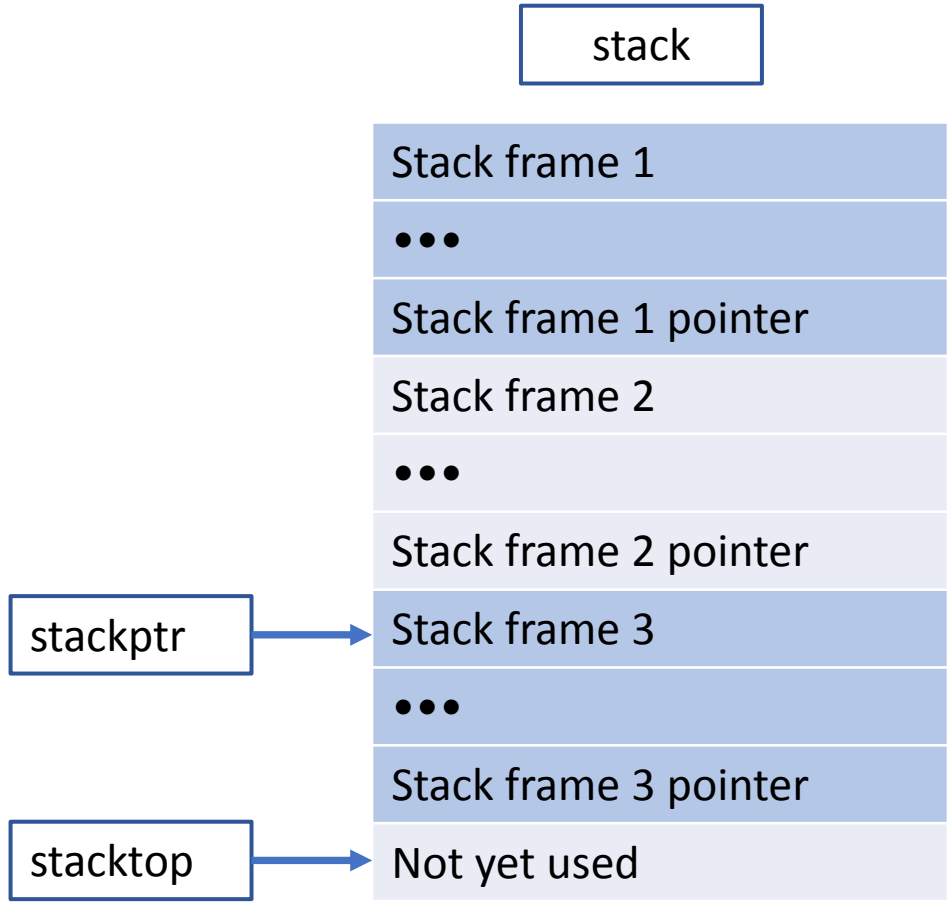
# Managing a Runtime Stack

- Many compiled languages use a runtime stack
- A “frame” is pushed onto the stack each time a subroutine is called
- The frame contains the subroutine parameters, and the local variables
- Calling sequence:
  - Load A with the number of bytes required
  - `jsb stkpush`
  - Returns the address of the stack frame in A
  - If the stack is full, returns 0 in A

- To “Pop” the frame off the stack
  - `jsb stkpop`

# Memory Management

| | | | |
|---|---|---|---|
| stack | RESW | 1000 | ; space for stack |
| endstack | WORD | 0 | ; end of stack |
| stacktop | WORD | #stack | ; next unused word |
| stackptr | RESW | 1 | ; stack address pointer |
| savea | RESW | 1 | ; place to save A register |
| savex | RESW | 1 | ; place to save X register |



# Memory Management

```
stkpush  STX   savex     ; save the X register
         LDX   stacktop  ; get the current top of stack
         STX   stackptr  ; save it as next stack pointer
         ADD   stackptr  ; add space needed to stack pointer
         ADD   #3        ; add space for previous stack pointer
         COMP  #endstack ; check for stack overflow
         JLT   stkOK
         CLEAR A         ; return error condition if stack is full
         RSUB
stkOK    STA   stacktop  ; update top of stack
         SUB   #3        ; get address to store prev stack ptr
         RMO   X, A
         LDA   stackptr  ; get stack pointer and put in current stack frame
         STA   0, X
         LDX   savex     ; restore X
         RSUB            ; return to caller
```

# Memory Management

| | | | |
|---|---|---|---|
| stkpop | STA | savea | ; save the A register |
| | STX | savex | ; save the X register |
| | LDA | stacktop | ; Get the address of the bottom of the previous stack frame |
| | SUB | # | |
| | RMO | X, A | ; move it into X |
| | LDA | 0, X | ; get the address of the previous stack frame |
| | STA | stacktop | ; make it the new top of stack |
| | LDA | savea | ; restore A |
| | LDX | savex | ; restore X |
| | RSUB | | ; return to caller |

# Printing

- The compiler emits code to print; this is usually a call to a library routine
- As an example, here is a very simple function to print a string
- Calling sequence:
  - Store the I/O device number in `outdev`
  - Load X with the address of a null-terminates string to print
  - `jsb pstr`

# Printing

```
outdev  BYTE  5       ; Output device number
savea   RESW  1       ; Space to save the A register
                      
pstr    STA   savea   ; Save A
        LDCH  0, X    ; A = character to output
        COMP  #0      ; Check for null character (string terminator)
        JGT   pstr1   ; If not null, go print the character
        LDA   savea   ; If null, all done printing:  return A.
        RSUB          ; Return to caller
pstr1   TD    outdev  ; Test to see if output device is ready
        JEQ   pstr1   ; Device not ready
        WD    outdev  ; Output A register to output device
        TIX   #0      ; Increment X to point to next character
        J     pstr    ; loop
```

# Assignment 2

- Short assignment in Canvas – due next Wednesday
- Pseudocode
- SIC/XE Assembly Language