16. Examples

October 24, 2018

- Review
  - Subroutines
  - Calling and Return
  - Passing Parameters
- Example
  - Rotate 4 Hex Digits

ANNOUNCEMENTS

HYUNSU CHANGED HER OFFICE HOURS -
TODAY (10/24) - 7:30-9:30 pm, PCL 2.430B

REVIEW SESSION
SUNDAY (10/28) 2:00-5:00PM - HERE
(EER 1.516)

Subroutines

A **subroutine** is a program fragment that:

- lives in user space
- performs a well-defined task
- is invoked (called) by another user program
- returns control to the calling program when finished

Like a service routine, but not part of the OS

- not concerned with protecting hardware resources
- no special privilege required

Reasons for subroutines:

- reuse useful (and debugged!) code without having to keep typing it in
- divide task among multiple programmers
- use vendor-supplied *library* of useful routines

Passing Information to/from Subroutines

Arguments

- A value passed in to a subroutine is called an argument.
- This is a value needed by the subroutine to do its job.
- Examples:
  - In 2sComp routine, R0 is the number to be negated
  - In OUT service routine, R0 is the character to be printed.
  - In PUTS routine, R0 is ***address*** of string to be printed.

Return Values

- A value passed out of a subroutine is called a return value.
- This is the value that you called the subroutine to compute.
- Examples:
  - In 2sComp routine, negated value is returned in R0.
  - In GETC service routine, character read from the keyboard is returned in R0.

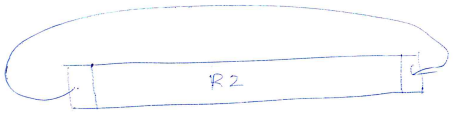
Using Subroutines

In order to use a subroutine, a programmer must know:

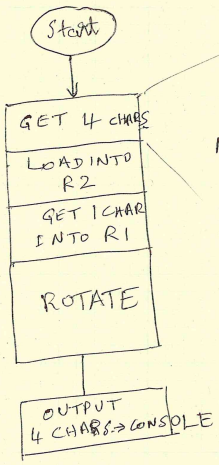
- its address (or at least a label that will be bound to its address)
- its function (what does it do?)
  - NOTE: The programmer does not need to know ***how*** the subroutine works, but what changes are visible in the machine's state after the routine has run.
- its arguments (where to pass data in, if any)
- its return values (where to get computed data, if any)

Example

- **Rotate 4 Hex Digits** LEFT
  - **Input digits from keyboard into a register (R2)** – ASCII → BINARY
  - **Input number of places to rotate from keyboard (R1)**
  - **Perform Rotation**
  - **Display digits after rotation on console** → BINARY → ASCII



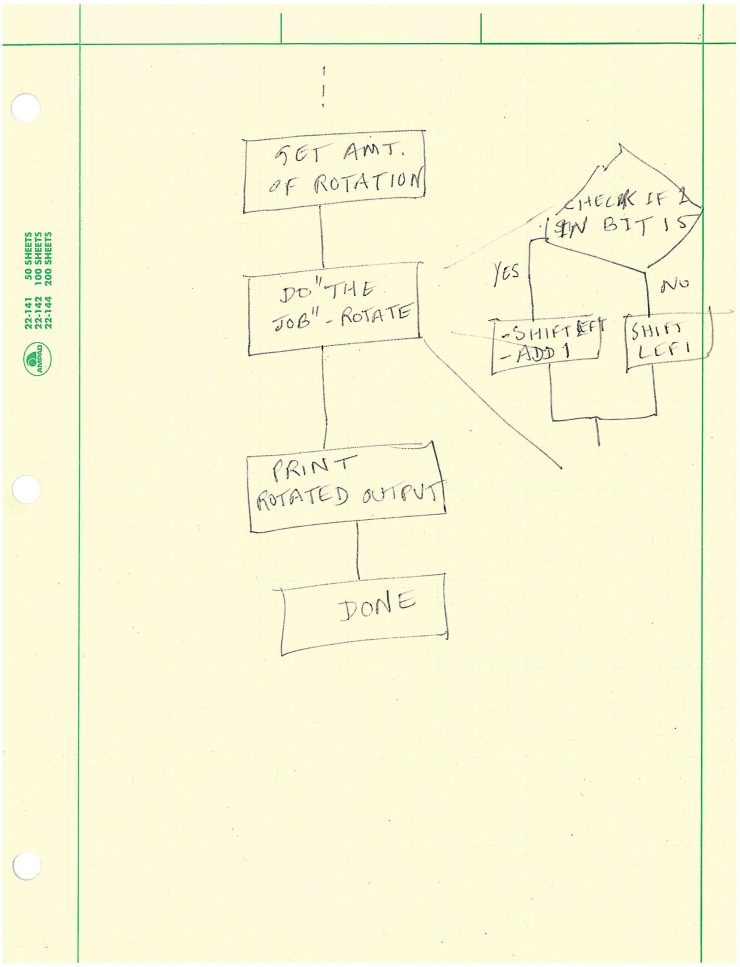
R1: # OF ROTATES (1-BIT)




```
        AND R2, R2, #0
        AND R5, R5, #0
        ADD R5, R5, #4
NEXT_IN BRZ  STEP 2
        LD  R0, LINEFEED
        OUT
        LEA R0, INBANNER
        PUTS
        GETC
        OUT
```

```
        4 DIGITS
LOAD INTO R2
        LD  R3, NEGZERO
        ADD R3, R3, R0
        BRn BAD     ; NOT A VALID CHAR.
        LD  R3, NEGNINE
        ADD R3, R3, R0
        BRp ATOF
; WE KNOW WE HAVE A DECIMAL DIGIT
        ADD R2, R2, R2  ; SHIFT R2
        ADD R2, R2, R2  ; 4 PLACES LEFT
        ADD R2, R2, R2
        ADD R2, R2, R2
;       AND R0, R0, xF  ; KEEP LAST
                        ; 4 BITS
        ADD R2, R2, R0
;       ADD R5, R5, #-1
        BR  NEXT_IN
; TEST FOR A-F
ATOF
```

```
ATOF    LD  R3, NEG_A
        ADD R3, R3, R0
        BRn BAD
        LD  R3, NEG_F
        ADD R3, R3, R0
        BRp BAD
; WE HAVE A HEX DIGIT IN R3
        ADD R2, R2, R2  ; SHIFT LEFT
        ADD R2, R2, R2
        ADD R2, R2, R2
        ADD R2, R2, R2
;       LD  R3, NEG_55  ; -#55
        ADD R0, R0, R3
        ADD R2, R2, R0
;       ADD R5, R5, #-1
        BR  NEXT_IN
BAD     - - - ; PRINT "BAD CHAR"
```




```
   .ORIG   x3000
; first step - input 4 hex digits, check them and load into R2
       AND R2, R2, #0          ; R2 will keep the 4 hex digits
       AND R5, R5, #0          ; R5 will keep track of how many digits left
       ADD R5, R5, #4          ; 4 digits to start with
;
NEXT_IN BRz STEP2               ; next step after getting 4 digits
;
       LD  R0, LINEFEED
       TRAP x21
       LEA R0, INBANNER
       TRAP X22                ; PUTS
       TRAP X20                ; GET HEX DIGIT
       TRAP X21                ; ECHO
; Start test for decimal digit
;
       LD  R3, NEGZERO
       ADD R3, R3, R0
       BRn BAD                 ; Could have gone to "bad" also
       LD  R3, NEGNINE
       ADD R3, R3, R0
       BRp ATOF
;
; Know we have a decimal digit
;
       ADD R2, R2, R2          ; Shift R2 4 places left to make room for one digit
       ADD R2, R2, R2
       ADD R2, R2, R2
       ADD R2, R2, R2
;
       AND R0, R0, xF          ; just keep last 4 bits which is the number we want
;
       ADD R2, R2, R0
;
       ADD R5, R5, #-1
       BR  NEXT_IN
; TEST FOR A THROUGH F
;
ATOF    LD  R3, NEG_A
       ADD  R3, R3, R0
       BRn  BAD
       LD   R3, NEG_F
       ADD  R3, R3, R0
       BRp  BAD
;
; We got a hex digit from A to F
;
               ADD  R2, R2, R2         ; Shift R2 4 left to make room
               ADD  R2, R2, R2
               ADD  R2, R2, R2
               ADD  R2, R2, R2

               LD   R3, NEG_55         ; ASCII code minus value = #55
               ADD  R0, R0, R3
               ADD  R2, R2, R0

               ADD  R5, R5, #-1
               BR   NEXT_IN

BAD             LD   R0, LINEFEED
               TRAP x21
               LEA  R0, NOTHEX
               TRAP x22
               BR   NEXT_IN
;
```

```
;
LINEFEED .FILL  x0A
INBANNER .STRINGZ "Input a HEX digit: "
NOTHEX   .STRINGZ "That was not a Hex digit!"
NEGZERO  .FILL xFFD0
NEGNINE  .FILL xFFC7
NEG_A    .FILL xFFBF
NEG_F    .FILL xFFBA
NEG_55   .FILL xFFC9
;
; Now get the amount of rotation (assume that input is correct!)
;
STEP2            AND  R1, R1, #0

               LD   R0,LINEFEED
               TRAP x21
               LEA  R0,ROTATION
               TRAP x22
               TRAP x20                ; We will assume this time input is correct
               TRAP x21                ; Echo input

               LD   R3, NEGNINE
               ADD  R3, R3, R0
               BRp  ROT_AF

               AND  R0,R0,xF
               ADD  R1,R1,R0
               BR   THEJOB

ROT_AF          LD   R3, NEG_55
               ADD  R0, R0, R3
               ADD  R1, R1, R0
               BR   THEJOB

ROTATION        .STRINGZ  "Input a Hex digit (Rotation): "

THEJOB          ADD R1, R1, #0
AGAIN           BRz DISPLAY_OUT
               ADD R2, R2, #0
               BRn NEG
               ADD R2, R2, R2
               BRnzp DEC
NEG             ADD R2, R2, R2
               ADD R2, R2, #1
DEC             ADD R1, R1, #-1
               BR AGAIN

; The output routine

DISPLAY_OUT     LD   R0,LINEFEED
               TRAP x21
               LEA  R4, ASCIITABLE
               AND  R5, R5, #0
               ADD  R5, R5, #4         ; R5 will keep track of how many left to output

NEXT_OUT        BRz  DONE

               AND   R1, R1, #0
               ADD   R1, R1, #4
AGAIN2          BRz   DISPLAY_CHAR
               ADD   R2, R2, #0
               BRn   NEG2
               ADD   R2, R2, R2
               BRnzp DEC2
NEG2            ADD   R2, R2, R2
```

```
               ADD   R2, R2, #1
DEC2            ADD   R1, R1, #-1
               BR    AGAIN2
;
; output routine
;
DISPLAY_CHAR    AND  R3, R2, xF         ; This extracts the high hex digit

               ADD  R3, R3, R4         ; R3 now points to the ASCII code of that digit
               LDR  R0, R3, #0         ; R0 now contains the ASCII code of that digit

               TRAP x21

               ADD  R5, R5, #-1
               BR   NEXT_OUT

DONE            TRAP x25

ASCIITABLE      .FILL x30
               .FILL x31
               .FILL x32
               .FILL x33
               .FILL x34
               .FILL x35
               .FILL x36
               .FILL x37
               .FILL x38
               .FILL x39
               .FILL x41
               .FILL x42
               .FILL x43
               .FILL x44
               .FILL x45
               .FILL x46
.END
```