

# Bit Operations and Traps in PeANUt

- ref: [PeANUt Spec, sect 2.8.3 and Appendix B]
- bitwise operations
- traps:
  - concepts
  - in PeANUt;
    - ◆ predefined and user-definable
  - trap handler and trap table
- debugging
- other issues:
  - overall feedback on Assignment 1
  - a final point on C programming for the MSE
  - new! PeANUt@Home page
- revise lecture P8

## Bitwise Operations in PeANUt

- need to get at bit-level of data in many applications (recall AND and OR gates)
- example: what does the following code do?

```

load    x          ;
not     ; /*AC = not AC*/
add     #1         ;
store  y          ;
    
```

- bit masks are useful: (bitop-example.ass)

```

Msk:    data      %00000 111111 00000      ; int Msk = 2016;
CMsk:   data      %11111 000000 11111      ; int CMsk = ~Msk; /* -
tmp:    block     1                          ; int tmp;
    
```

- to get a range of bits:

```

AC: [xxxxx | yyyyyy | zzzzz]
    and Msk
AC: [00000 | yyyyyy | 00000]
    store tmp
    
```

- to set a range of bits:

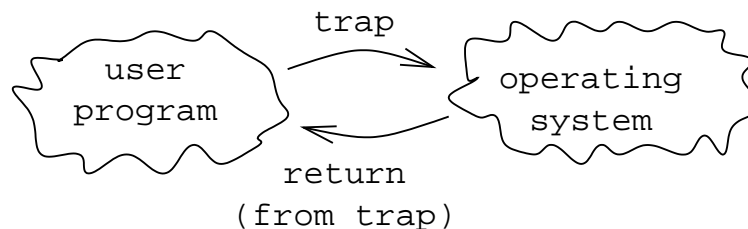
```

AC: [xxxxx | aaaaaa | zzzzz]
    and CMsk
AC: [xxxxx | 000000 | zzzzz]
    or tmp
AC: [xxxxx | yyyyyy | zzzzz]
    
```

- note: we can use `dvd #32` to shift right (`>>`) 5 bits  
and `mul #32` to shift left (`<<`) 5 bits (care: **overflow!**) (c.f.  $\div * 10^5$  in decimal)

## Traps and Exceptions in PeANUt

- input/output and other crucial transfers of control can only be safely performed by the operating system (OS)
- PeANUt has an operating system, only visible via traps
- the OS creates a user program; then both exist as interacting processes
- trap: an interaction with the OS and user program



- exceptions are traps initiated via a (hardware) event (#4–8, #10, #11)
- traps may also be initiated by (software) execution of the corresponding `trap` instruction – the effect is the same
- interrupts are exceptions that may also be initiated by events outside the processor (e.g. network, disk device; not available on PeANUt)

## PeANUt Predefined Traps

#1 Halt (return control to operating system, user program process then terminates)

#2 Get (operating system code will *wait* if needed)

#3 Put

#4 Data error (from Get/Put)

#5 Illegal Instruction

#6 Illegal Mode (e.g. from store #5)

#7 Integer Overflow (if EN=1 abort, else OV=1)

#8 Integer Divide by 0

#9 Establish Trap Routine (set new trap or modify existing one)

#10 Trapping Error (from e.g. trap #23 (if trap 23 not yet established), or error in handling established trap)

#11 Page Fault (for PeANUt virtual memory mode only; needs predefined handler at a46)

#12, #13 Swap Page In, Out (AC contains page number)

● note: the default action of traps #4, #5, #6, #8, #10 is to abort





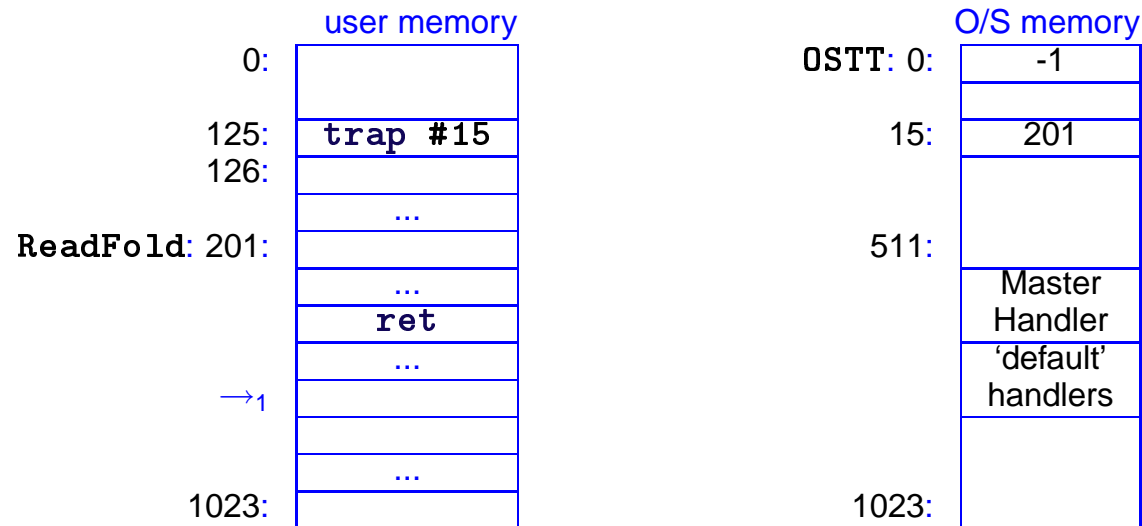
## Software Initiated Traps: softwaretrap-example.ass

- can be used to implement simple procedures, e.g. define trap 15 to read in chars. with upper case folded to lower case

```
TT15:    data    15        ;
         data    ReadFold; //ret. next char read, folded to lower
ReadFold:                                ; char ReadFold() { /*RV passed via AC*/
                                         ; char ch;          /*also impl. via AC*/
         trap    #2        ; scanf ("%c", &ch);
         cmp     #'@'      ; if ((ch >= 'A') &&
         ble     RFenf     ;          (ch <= 'Z')) {
         cmp     #'Z'      ;
         bgt     RFenf     ;
         sub     #'A'      ;     ch = ch - 'A'
         add     #'a'      ;         + 'a';
RFenf:   ;          }
         ;          return ch;
         ret     ;      } /* ReadFold() */
main:    ; int main() {
         loada   TT15     ;          /* establish trap #15 */
         trap    #9        ;          /* with handler ReadFold */
do1:     ; do {
         trap    #15     ;          char c = ReadFold(); /*impl. via AC*/
         trap    #3        ;          printf ("%c", c);
```

# Traps and the Operating System

- a simple model for the operating system role of traps:
  - user program sees only *half* of the PeANUt machine;  
operating system has own memory, special instructions and registers
  - operating system has 512-word trap table (**OSTT**) for current action of each trap  
and also code (handler routines) for the default actions
  - a trap is like a procedure call to an operating system Master Handler,  
which then uses the **OSTT** to call the corresponding handler routine



## Traps – Review

- establish traps via TTI (trap table item) and `trap #9`
  - traps work via a procedure-like interaction of the operating system and user program
  - the handler routine calling convention is different:
    - uses AC to pass parameter and return value (if any)
    - ◆ Q: how does program control return to correct point in user program?  
what about upon exceptions?
- require a fair bit of extra hardware  
(and then some more to perform raw input/output accesses etc.)
- virtual input/output is an important abstraction, and is usually implemented via traps  
(simplicity, security)
- “No [user program] is an island!”

## Debugging

- very 'small' errors can lead to very strange results!
- use break points in the PeANUt tool. Set a break point at:
  - each **call** instruction (check parameters)
  - head of each (unproven) loop (check index variables)
  - other critical points
  - are the values what you expected?
  - use **single step** between break points if suspect bug is there
  - map memory / PC locations to `.ass` code  
(compare with listing file `.lst` if not obvious)
  - check sequence of instructions, value of accumulator at **load / store**
- end of PeANUt Module; followup with 'real world' perspectives:
  - Memory Systems and Modern Machines (5)
  - Operating System Concepts (4)
  - Interconnection Networks (1)

## A Tale of 2 PeANUTs: an odyssey of survival

- the good old days: CSA02 (assembler only; debug via `execute -trace!`, code base in Modula-2...)
- 1992: DCS GUI implemented (in C)
  - 1993: the term “PeANUt” is coined
  - 1994: “port” code base to Garden Point Modula-2 compiler (!!!)
- 1996: MD/BH/PM PeANUt (mostly C; GUI uses Tcl/TK and Tix libraries)
  - 1998: adoption by DCS; only Solaris versions available
  - 2002: first ‘port’ to Linux (v2.13)
  - 2005: no longer able to run on Linux!
  - 2006: an investigation concludes GUI must be completely rewritten (i.e. Java or Python-based) to run on Linux
    - ◆ only old Solaris GUI working (cannot re-compile on Solaris)
  - 2007: `assemble` binary on Solaris no longer works!!! (‘edge of darkness’)
  - 2008: fixes to run GUI on Linux and other bug fixes (v2.14)