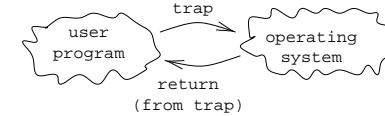


Bit Operations and Traps in PeANUt

- 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, Tute/Labs and the MSE (comments, marks)
 - wed11-13 tute/lab session arrangements for Apr 25

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)

Bitwise Operations in PeANUt

- need to get at bit-level of data in many applications
- 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!)

## 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

## Trap Handler Routine

- a Trap Table Item (TTI) is a 2-word record containing:
  - a trap number
  - address of a handler routine (procedure)
    - or action information:  $-2$  to ignore trap,  $-1$  to restore the default action

- example: establish trap 14 to execute handler routine **T14Pr1**

```

TT14: data 14 ;
 data T14Pr1 ;
T14Pr1:
... ; void T14Pr1() {
... ; /* code to handle trap #14 */
... ;
ret ; } /* T14Proc1() */
...
loada TT14 ; /* establish trap #14
trap #9 ; /* with handler T14Pr1 */
...
trap #14 ; /* execute trap #14 */

```

## Trap Handler Routine – Example **traphandle-example.ass**

- ignore integer overflow

```

TT7I: data 7 ; /* TTI for ignore trap #7 */
 data -2 ;
TT7D: data 7 ; /* TTI for restore trap #7 */
 data -1 ;
main:
loada TT7I ; /* redefine trap #7 to ignore */
trap #9 ; /* from here, ignore overflow */

load #511 ; /* generate trap #7 via */
mul #511 ; /* exception; sets OV only */

load #' \n ' ; /* print EOL, to show ... */
trap #3 ; /* ... abort did not occur */

loada TT7D ; /* restore trap #7 */
trap #9 ; /* from here, default action (abort) */

load #511 ; /* generate trap #7 via */
mul #511 ; /* exception; abort should happen */

```

- question: what are other ways to ignore integer overflow on PeANUt?

## 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:
... ; }
ret ; return ch;
main:
loada TT15 ; /* establish trap #15 */
trap #9 ; /* with handler ReadFold */
do1:
trap #15 ; do {
trap #3 ; char c = ReadFold(); /*impl. via AC
 ; 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

