

## Bit operations and traps in PeANUt

- Bitwise operations
- Traps
  - Trap concept
  - PeANUt traps
  - Trap handler and trap table
- Debugging
- References:
  - Specification of the PeANUt computer (2.8.3 and Appendix B)

1

COMP2300, 2006

## Bitwise operations in PeANUt (1)

- Need to get at bit-level of data in many applications
- Example: What does the following code do?

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

- Bit masks are useful:

```
Msk:  data    %00000 11111 00000; /* 2016 */
CMsk: data    %11111 00000 11111; /* -2017 */
tmp:  block 1      ; /* 'temporary variable' */
```

2

COMP2300, 2006

## Bitwise operations in PeANUt (2)

- To get a range of bits

```
AC:  xxxxx  yyyyyy  zzzzz
and  Msk
AC:  00000  yyyyyy  00000
store tmp
```

- And to set a range of bits

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

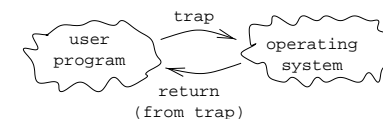
- Note: We can use **dvd #32** to **shift right** 5 bits and **mul #32** to **shift left** 5 bits (Care: Overflow!)

3

COMP2300, 2006

## Traps 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 OS and user program



4

COMP2300, 2006

## More on traps

- **Exceptions** are traps initiated via (hardware) event (**#4-8, 10, 11**)
- Interrupts are exceptions that may also be initiated by events outside the processor (like network, disk device; not available on PeANUt)
- Traps may also be initiated by (software) execution of the corresponding **trap** instruction – the effect is same

5

COMP2300, 2006

## PeANUt defined traps (1)

- PeANUt predefined traps (also virtual memory traps **#11-13**)
  - #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

6

COMP2300, 2006

## PeANUt defined traps (2)

- PeANUt predefined traps (continued)
  - #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

7

COMP2300, 2006

## Trap handler routine

- A **Trap Table Item** (TTI) is a 2-word record containing:
  - A trap number
  - A **handler routine** address (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         ; } /* T14Pr1() */
      ...          ;
      loada TT14   ; /* establish trap #14
      trap  #9     ; with handler T14Pr1 */
      ...          ;
      trap  #14   ; /* execute trap #14 */

```

8

COMP2300, 2006

## Trap handler routine – Example

- Ignore integer overflow

```

TT7I: data 7 ; /* TTI for ignore trap #7 */
      data -2 ;
TT7D: data 7 ; /* TTI for restore trap #7 */
      data -1 ;
      ...
      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 */
      ...
      loada TT7D ; /* restore trap #7 */
      trap #9 ; /* from here, default action (abort) */
      ...

```

- Question: What are other ways to ignore integer overflow on PeANUt?

## Software initiated traps (2)

```

...
; AC = AC
sub #'A' ; - 'A'
add #'a' ; + 'a'; /* AC = AC + 32 */
RFenf: ; } /* if */
; return ch;
ret ; } /* ReadFold() */
...
loada TT15 ; /* establish trap #15 */
trap #9 ; /* with handler ReadFold */
...
trap #15 ; NextC = ReadFold();
store NextC ;

```

## Software initiated traps (1)

- Software-initiated traps may also be used to implement simple procedures
- Example: Define a trap to read in characters with upper case converted to lower

```

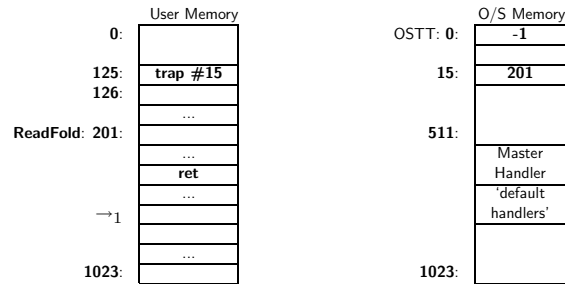
TT15: data 15 ;
      data ReadFold ;
; char ReadFold() {
; /* post: returns next char */
; /* read, folded to lower case */
; /* nb. RV passed via AC */
; char ch; /* also impl. via AC */
ReadFold:
;
trap #2 ; /* Get */
cmp #'Q' ; if ((AC >= 'A') &&
ble RFenf ;
cmp #'Z' ; (AC <= 'Z')) {
bgt RFenf ;
...

```

## Operating system and traps

- A 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

## Operating system and traps – Memory view



## Debugging

- Very 'small' errors can lead to very strange results
- Use **break points** in 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 suspected 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**

## 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
- 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)

## End of PeANUt module

- After semester break (after Anzac day, NO lecture on Monday 24th April)
  - Memory Systems and Modern Machines (Lecture M1 on Thursday 27th April)
  - Operating System Concepts
  - Interconnection Networks
  - Assignment 2 (due Wednesday 17th May)
  - No lab session in first week after the break
- Have a good and safe semester break... :-)