Carlito

Universiteit Antwerpen
Faculteit Wetenschappen

Computersystemen en -architectuur

MIPS: Memory-mapped I/O

Tim Apers

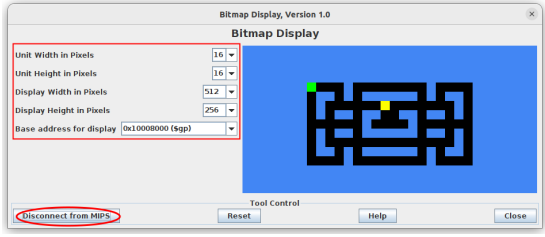
tim.apers@uantwerpen.be

Academiejaar 2024 – 2025

Memory-mapped I/O

- Toetsenbord input
- Bitmap output
- Bestanden inlezen
- Sleep-modus
- Functies

Bitmap output



- Elke `word` het in geheugen bevat één volledige pixel
- Row-major
- ! Gebruik de correcte instellingen
  - Unit width/height = 16
  - Display width = 512 pixels
  - Display height = 256 pixels
- ! "Connect to MIPS" knop

Bitmap output

Geheugen-layout

- Elke pixel = 32 bit kleurencode in aRGB-formaat
- Elke kleur is één byte (00 = 0, FF = 255)
- Voorbeeld:
  - **rood** = 0x00FF0000
  - **groen** = 0x0000FF00
  - **blauw** = 0x000000FF
  - **paars** = 0x00671D9D
- Voorstelling in geheugen (gebruik het $gp register):

0x10008000	0x10008004	0x10008008	0x1000800C	0x10008010
0x10008014	0x10008018	0x1000801C	0x10008020	0x10008024
0x10008028	0x1000802C	0x10008030	0x10008034	0x10008038

Bestanden inlezen

- Lezen van de file descriptor:
  1. Zet `$v0` op 13
  2. `$a0`: adres van string gevolgd door zero-byte (`.asciiz`) met de filename
  3. `$a1`: op 0 zetten (flags)
  4. `$a2`: op 0 zetten (mode)
  5. Roep `syscall` instructie op
  6. `$v0` bevat de file descriptor (of negatief getal in geval van error)
- Lezen van de file inhoud:
  1. Zet `$v0` op 14
  2. `$a0`: de file descriptor
  3. `$a1`: het adres van de buffer (= lege ruimte in geheugen)
     - ! Eerst deze ruimte vrijmaken d.m.v. `.space` directive.
  4. `$a2`: maximum aantal characters (=bytes) die gelezen worden
  5. Roep `syscall` instructie op
  6. `$v0` bevat lengte van de ingelezen file

Bestanden inlezen

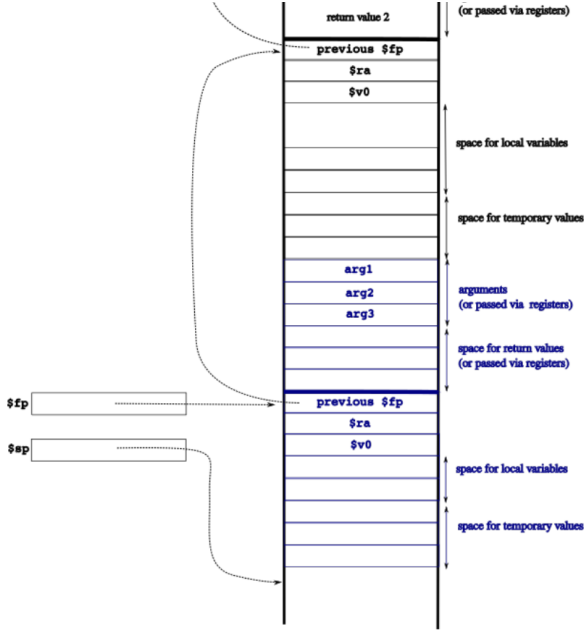
```
    .data
fin:    .asciiz "input.txt"     # filename for input
buffer: .space  2048        # space used as buffer

    .text
################################################################
# Open (for reading) a file
li $v0, 13      # system call for open file
la $a0, fin     # output file name
li $a1, 0       # Open for writing (flags are 0: read, 1: write)
li $a2, 0       # mode is ignored
syscall         # open a file (file descriptor returned in $v0)

move $s6, $v0   # save the file descriptor

################################################################
# Read from file to buffer
li $v0, 14      # system call for read from file
move $a0, $s6   # file descriptor
la $a1, buffer  # address of buffer to which to load the contents
li $a2, 2048    # hardcoded max number of characters (equal to size of buffer)
syscall         # read from file, $v0 contains number of characters read

################################################################
# Close the file
li $v0, 16      # system call for close file
move $a0, $s6   # file descriptor to close
syscall         # close file
################################################################
```

Sleep-modus

Laat toe om programma te pauzeren (bijv. time-steps in een videospel)

1. Zet `$v0` op 32
2. `$a0`: aantal milliseconden dat het programma moet pauzeren
3. Roep de `syscall` instructie op
4. Programma wacht nu een bepaalde tijd
5. Programma gaat na het wachten verder met de andere instructies

Functies en stackframes



Functies en stackframes

```
1
2  .text
3  ####### PROCEDURE to add two numbers ###################################
4  addtwonumbers:
5      sw  $fp, 0($sp) # push old frame pointer (dynamic link)
6      move    $fp, $sp    # frame pointer now points to the top of the stack
7      subu    $sp, $sp, 16    # allocate 16 bytes on the stack
8      sw  $ra, -4($fp)    # store the value of the return address
9      sw  $s0, -8($fp)    # save locally used registers
10     sw  $s1, -12($fp)
11
12     move $s0, $a0       # $s0 = first number to be added
13     move $s1, $a1       # $s1 = second number to be added
14
15     add $t0, $s0, $s1       # perform calculation
16     move    $v0, $t0        # place result in return value location
17
18     lw  $s1, -12($fp)   # reset saved register $s1
19     lw  $s0, -8($fp)    # reset saved register $s0
20     lw  $ra, -4($fp)    # get return address from frame
21     move    $sp, $fp        # get old frame pointer from current fra
22     lw  $fp, ($sp)  # restore old frame pointer
23     jr  $ra
24
25 ########################################################################
26     ... other code goes here ...
27     move    $a0, $t0    # Put procedure arguments
28     move    $a1, $t1    # Put procedure arguments
29     jal addtwonumbers   # Call procedure
```