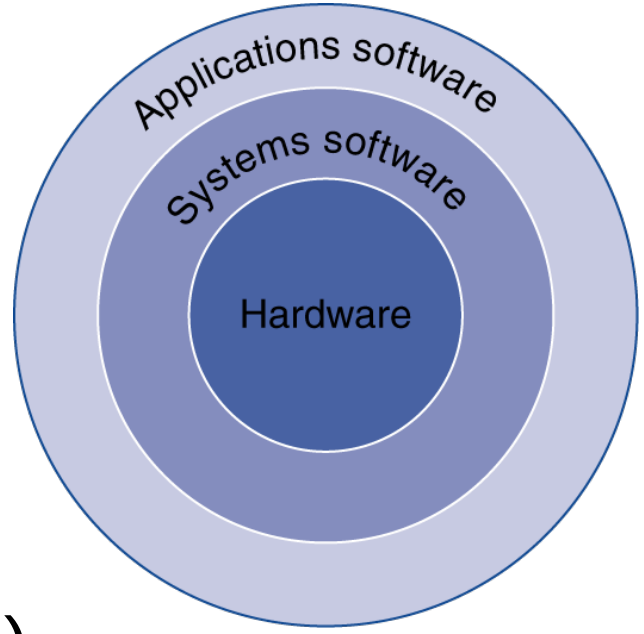
HW – SW – OS interface



- Executing Code (by datapath)
- Exceptions, traps and interrupts
- Memory-mapped I/O
- Interfacing with the OS (C-level) with `syscall`

Exceptions and Interrupts

Unexpected events (asynchronous interrupt) requiring change in **flow of control**

(different ISAs use the terms differently)

- "exception"
  - arises **within** the CPU

    e.g., undefined opcode, overflow, syscall, …
- "trap" (for monitoring, debugging)
- "interrupt"
  - From an **external** I/O controller

Dealing with them without sacrificing **performance** is hard (and requires extending the datapath)

Handling Exceptions (PH5 pp. A.33 - A.40)

In MIPS, exceptions are managed by a System Control **`CoProcessor`** (`CP0`)

`CP0` has its own registers: (access with `mfc0` and `mtc0`)

```
   Name      Register  Description

(*)BadVAddr   $8        memory address of offending memory access
   Count      $9        current timer; incremented every 10ms
   Compare   $11        timeout exception when Count == Compare
(*)Status    $12        controls which interrupts are enabled (vs.
masked)
(*)Cause     $13        exception type, and pending interrupts
(*)EPC       $14        PC where exception/interrupt occurred
```

(*)simulated by MARS

Registers | Coproc 1 | Coproc 0

Name	Number	Value
$8 (vaddr)	8	0x00000000
$12 (status)	12	0x0000ff11
$13 (cause)	13	0x00000000
$14 (epc)	14	0x00000000

Handling Exceptions

In MIPS, exceptions are managed by a System Control **`CoProcessor`** (`CP0`)

<u>When exception occurs</u>:

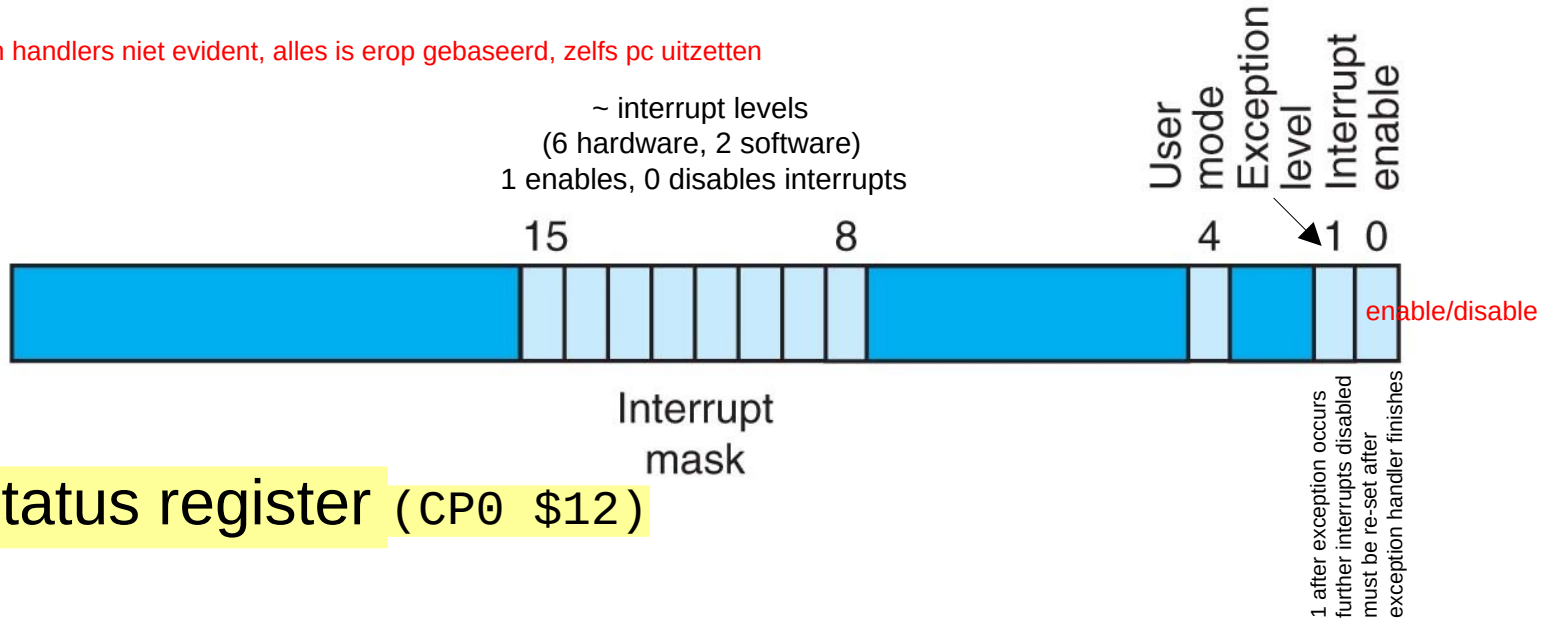
- **Save PC** of offending (or interrupted) instruction
  - in Exception Program Counter (**EPC**) register (`$14`)
- Save **indication of the problem**
  - in **cause** register (`$13`)

Exception Code in cause register ($13)

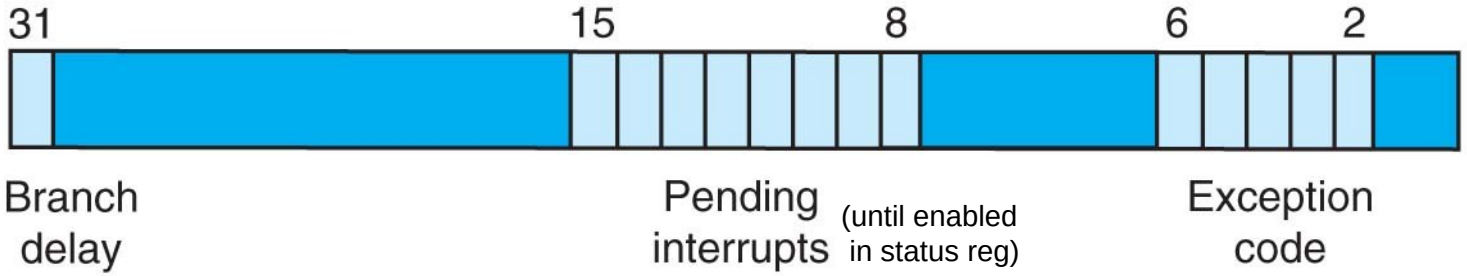
Number	Name	Cause of exception
0	Int	Hardware interrupt pending
4	AdEL	Address Error on Load or instruction fetch
5	AdES	Address Error on Store
6	IBE	Bus Error on Instruction fetch
7	DBE	Bus Error on Data load or store
8	Sys	Syscall exception
9	Bp	Breakpoint (usually used by debuggers)
10	CpU	Coprocessor Unimplemented
12	Ov	Arithmetic overflow
13	Tr	Trap
15	FPE	Floating Point Exception

Handling Exceptions

schrijven exception handlers niet evident, alles is erop gebaseerd, zelfs pc uitzetten



Status register `(CP0 $12)`



Cause register `(CP0 $13)`

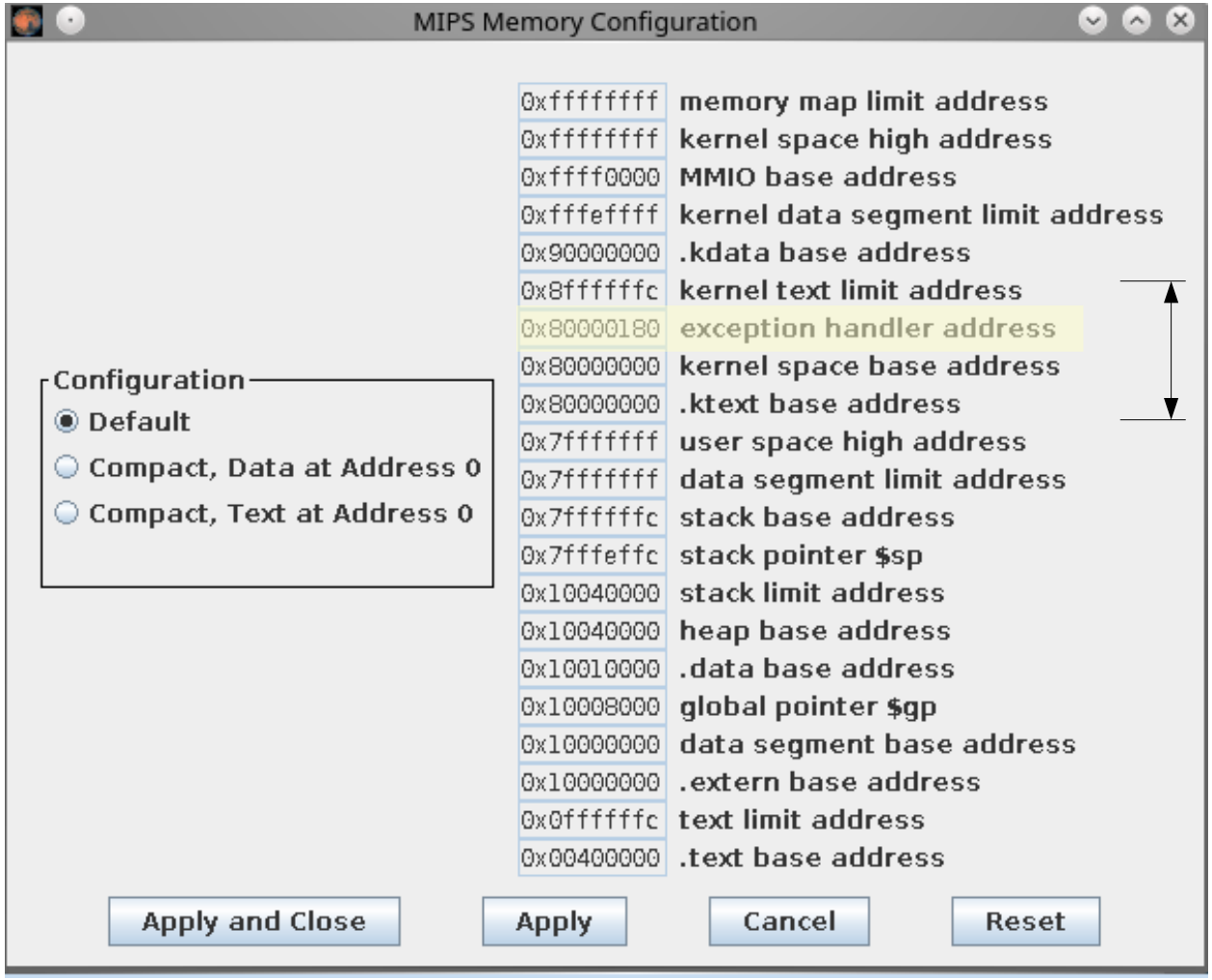
Handling Exceptions

In MIPS, exceptions are managed by a System Control **`CoProcessor`** (`CP0`)

<u>When exception occurs</u>:

- **Save PC** of offending (or interrupted) instruction
  - in Exception Program Counter (**EPC**) register (`$14`)
- Save **indication of the problem**
  - in **cause** register (`$13`)
- Jump to **handler** at `0x80000180`

Memory Layout



kernel

An Alternate Mechanism

Vectored Interrupts

- **Handler address** determined by the **cause**

- Example:
  - Undefined opcode: `0x8000 0000`
  - Arithmetic overflow: `0x8000 0020`
  - …: `0x8000 0040`

8 instructions

- Instructions (8) either
  - deal with the interrupt, or
  - jump to real handler

op verschillende addressen, verschillende error handlers etc

Handler Actions

- Read cause from `$13` (using `mfc0`), and transfer to relevant handler
- Determine action required (using `mfc0`)
- If restartable
  - take corrective action
  - use EPC to return to program

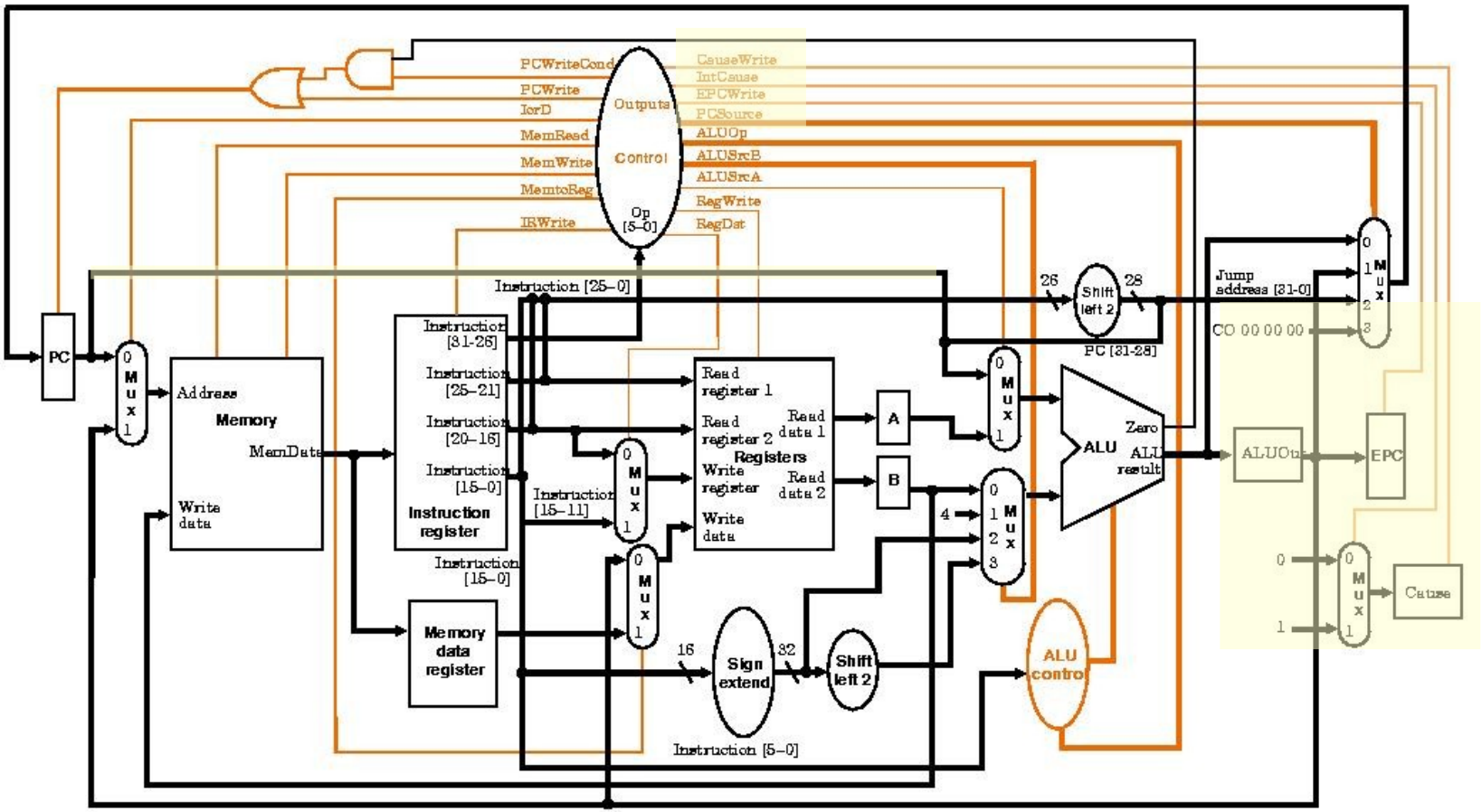
    `eret`
- If not re-startable
  - terminate program (cleanup)
  - report error (to OS) using EPC, cause, …

New instructions (to deal with CP0)

Instruction	Meaning
mfc0 Rd, C0src	Move from CoProcessor 0 register C0src into destination register Rd
mtc0 Rs, C0dst	Move to CoProcessor 0 register C0dst the value of register Rs
lwc0 C0dst, addr	Load a word from memory into CoProcessor 0 register C0dst
swc0 C0src, addr	Store CoProcessor 0 register C0src in memory
eret	return from exception Reset Exception Level to 0 (back to user mode) and return: PC = EPC
teq Rs, Rt	Raise the **trap** exception if register Rs is equal to register Rt
tne Rs, Rt	Raise the trap exception if register Rs is not equal to register Rt
tlt Rs, Rt	Raise the trap exception if register Rs is less than register Rt
. . . there are other trap instructions not listed here (see Appendix B)	
break code	Raise the **breakpoint** exception. code 1 is reserved for the debugger
syscall	Raise the **system call** exception. Service number is specified in $v0

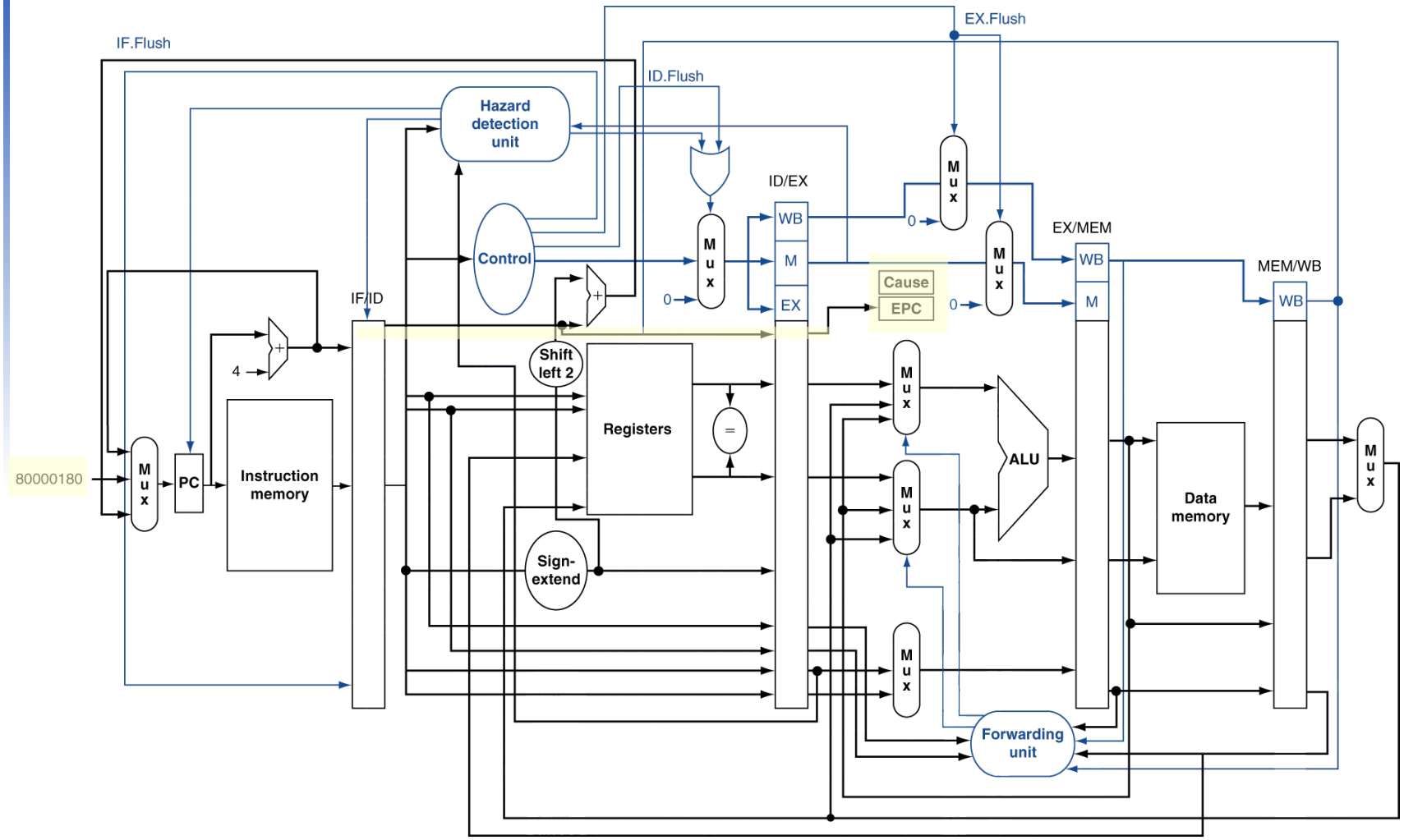
https://faculty.kfupm.edu.sa/coe/mudawar/coe301/lab/COE301_Lab_8_MIPS_Exceptions_and_IO.pdf

datapath with exceptions (partial)



https://people.cs.pitt.edu/~don/coe1502/current/Unit4a/Unit4a.html

datapath with exceptions (pipelined)



I/O: WHERE?

Communication with I/O devices

- Programmed or Instruction-Based I/O
- Memory-mapped I/O

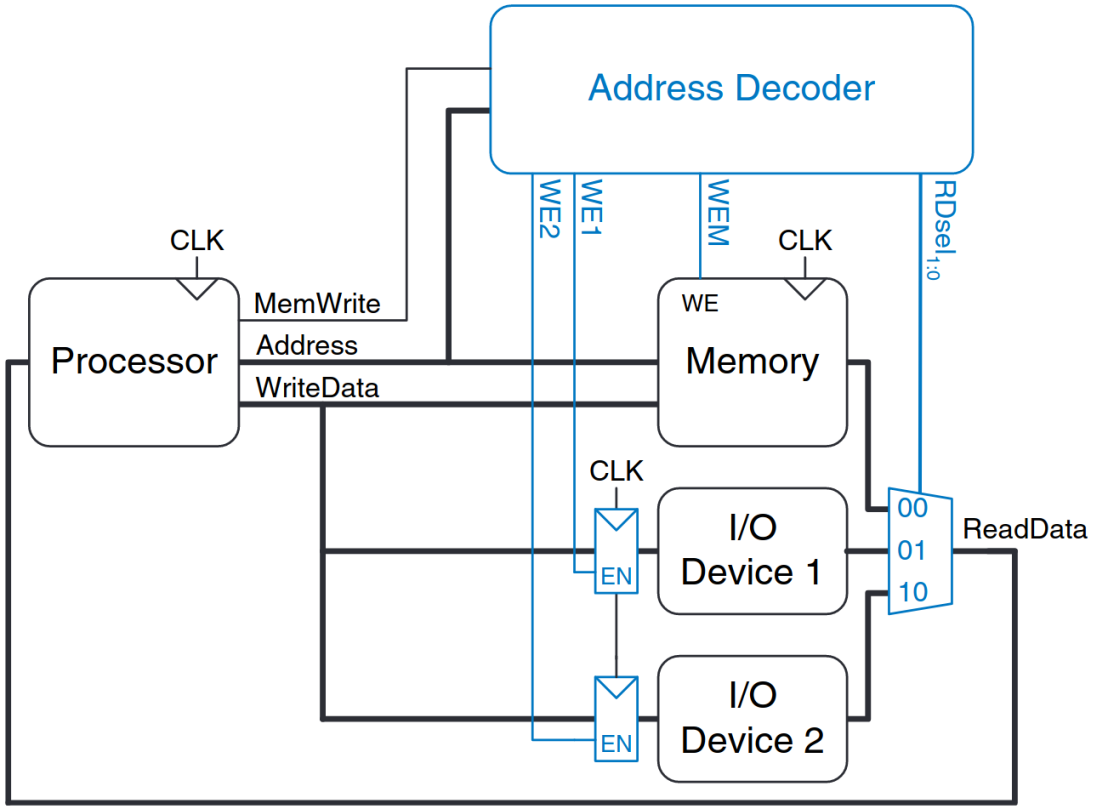
programmed or instruction-based I/O

Some architectures, most notably Intel x86, use **specialized instructions** to communicate with I/O devices. Also known as “isolated” or “port based” I/O. “Port” is the name of the address of the dedicated memory used for I/O.

These instructions take the following form, with `device1` and `device2` the unique ID of the I/O device:

```
lwio $t0, device1
swio $t0, device2
```

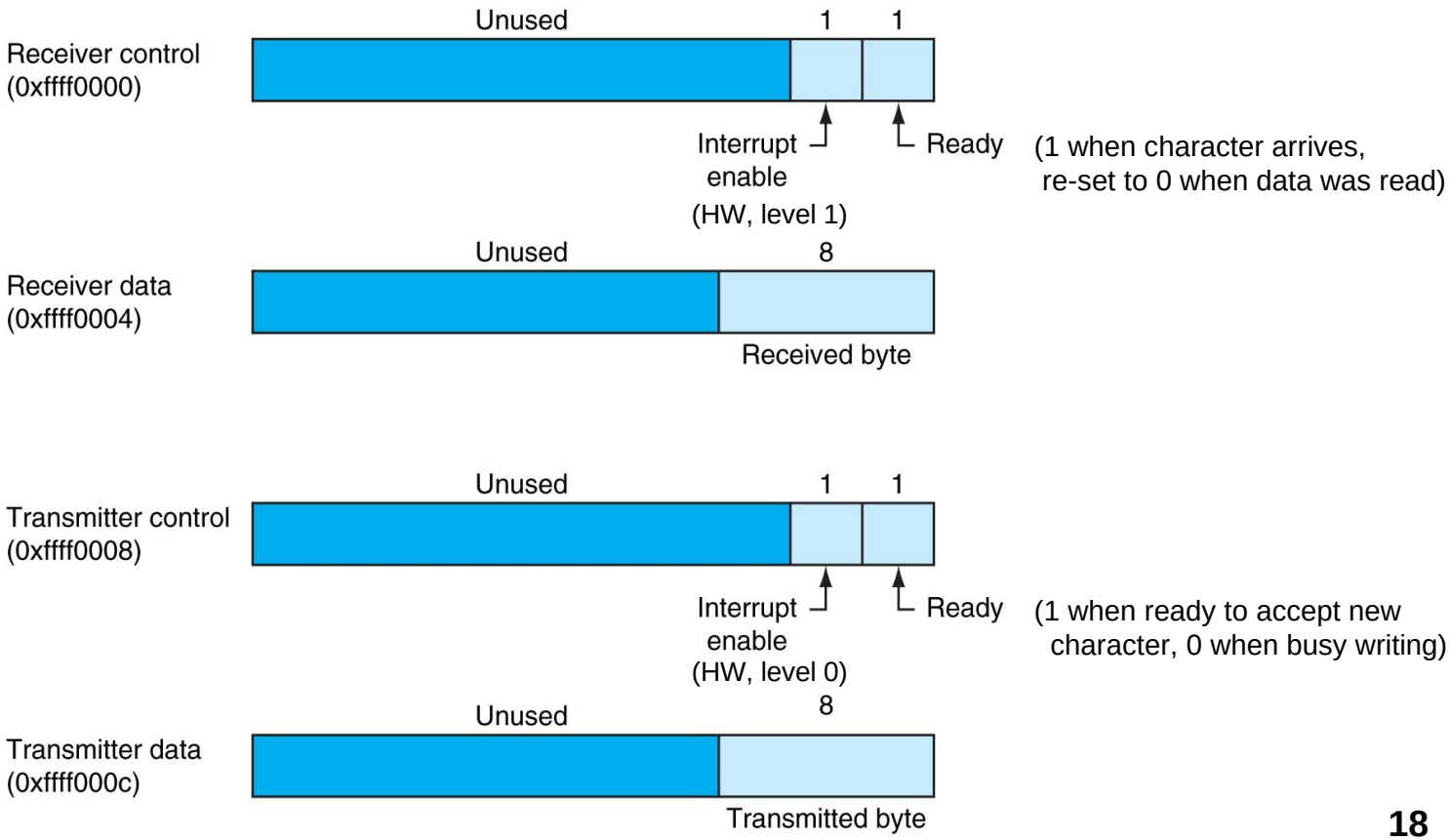
memory-mapped I/O



https://www.elsevier.com/books/digital-design-and-computer-architecture/harris/978-0-12-394424-5

memory-mapped I/O

- I/O via read/write of “device registers”
- Appear at special memory locations
- Accessed using **standard memory** `lw/sw` **operations**



I/O: WHEN?

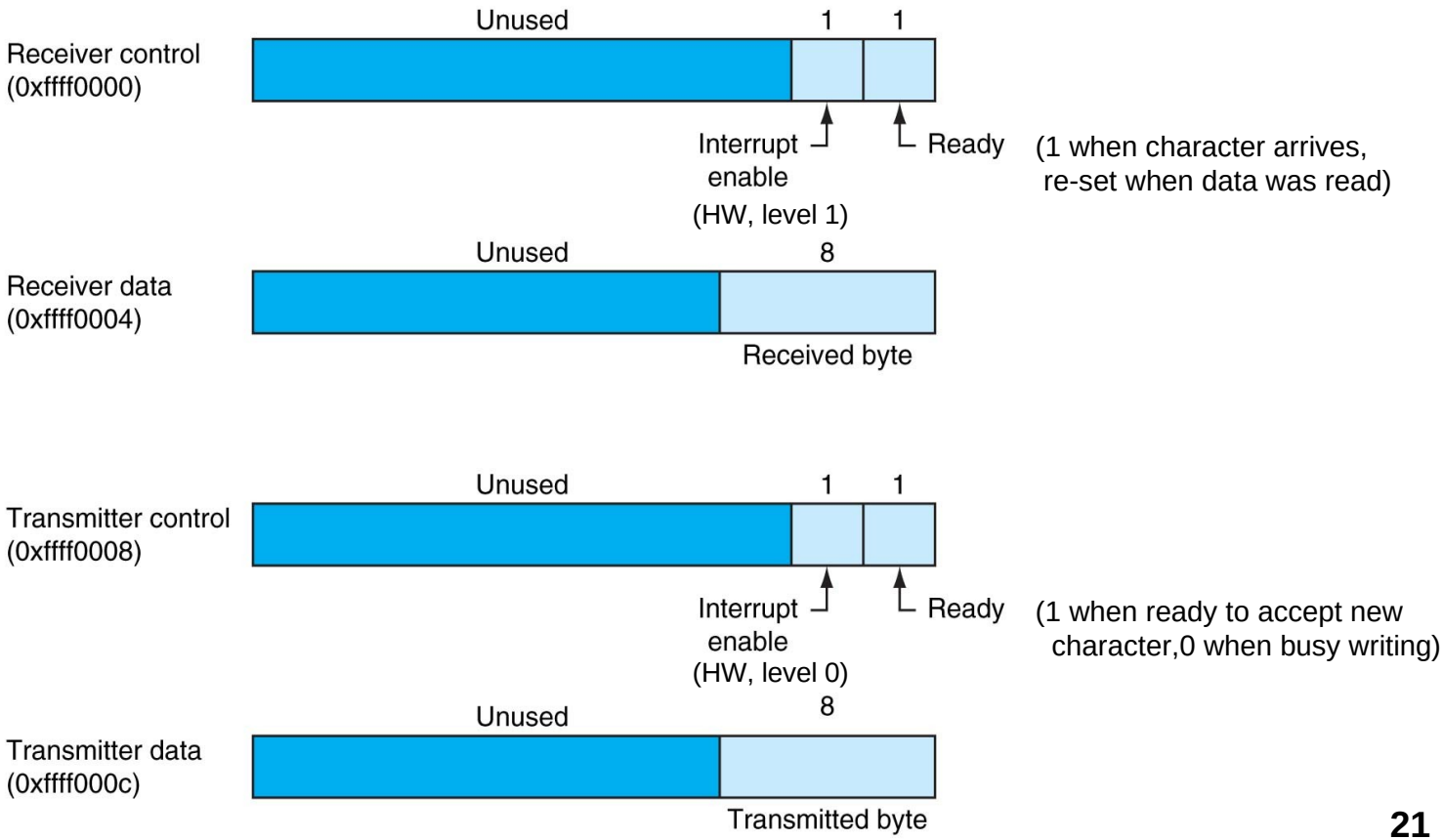
- **Polling**: busy loop inspecting status reg.

  → heavy use (waste) of processor time
  → OK in hardware implementation
  → predictable overhead (RT systems)

- **Interrupt-Driven**:
  a-synchronous reaction to device interrupt

memory-mapped I/O

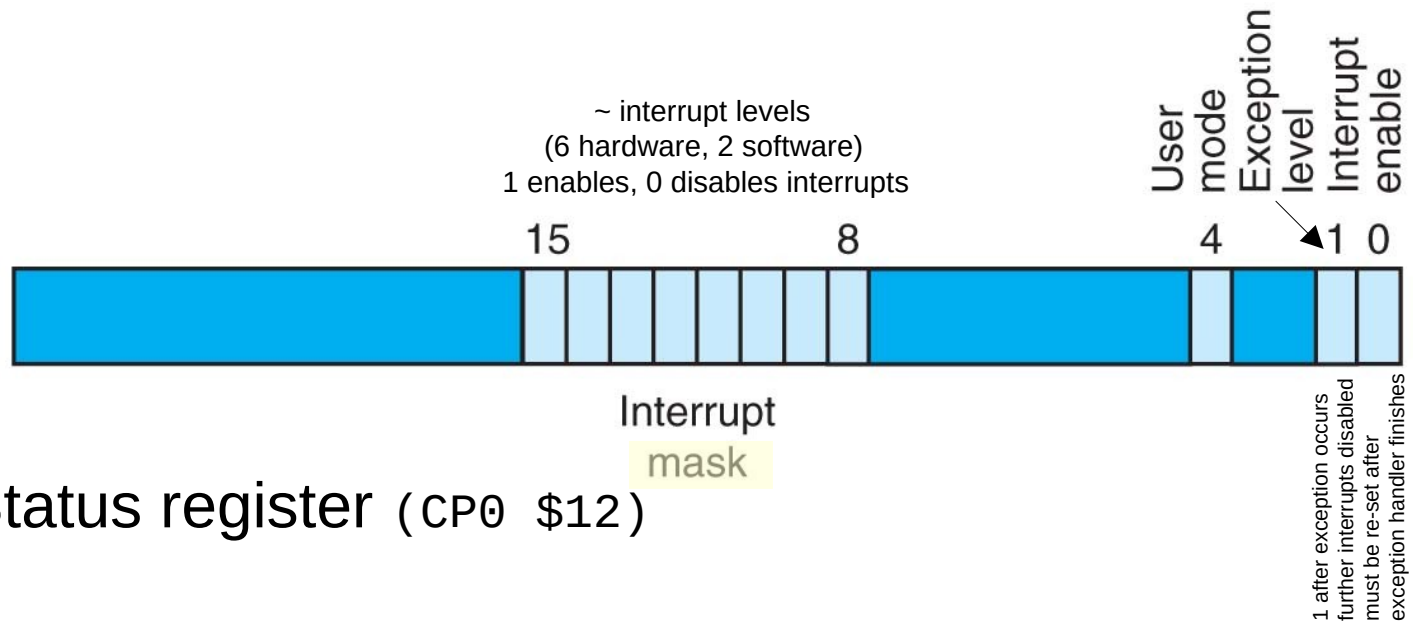
- I/O via read/write of “device registers”
- Appear at special memory locations
- Accessed using standard memory `lw/sw` operations



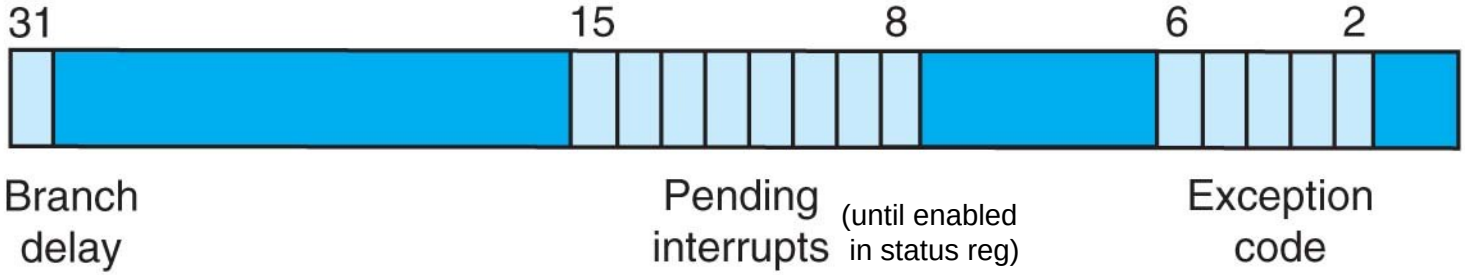
memory-mapped I/O, polling/busy wait

```
# printing a zero-terminated ASCII string

print_string: # $a0: address of zero-terminated ascii string (.asciiz)
              #      to print
    j ps_cond # jump to code to
              # * load next character to print
              # * check if end of string (loaded char is 0x00)

ps_loop:
    lw   $v0, 0xFFFF0008      # Transmitter control
    andi $v0, $v0, 0x01       # mask (select) Ready bit
    beq  $v0, $zero, ps_loop # busy loop until ready to print  <----------
    sw   $a1, 0xFFFF000C      # data (byte) to print into Transmitter data

ps_cond:
    lbu  $a1, ($a0)           # load character to print
    addi $a0, $a0, 1          # increment char pointer
    bne  $a1, $zero, ps_loop # loop as long as not EndOfString (0x00) found
    jr   $ra                  # return from subroutine
```

Handling Exceptions



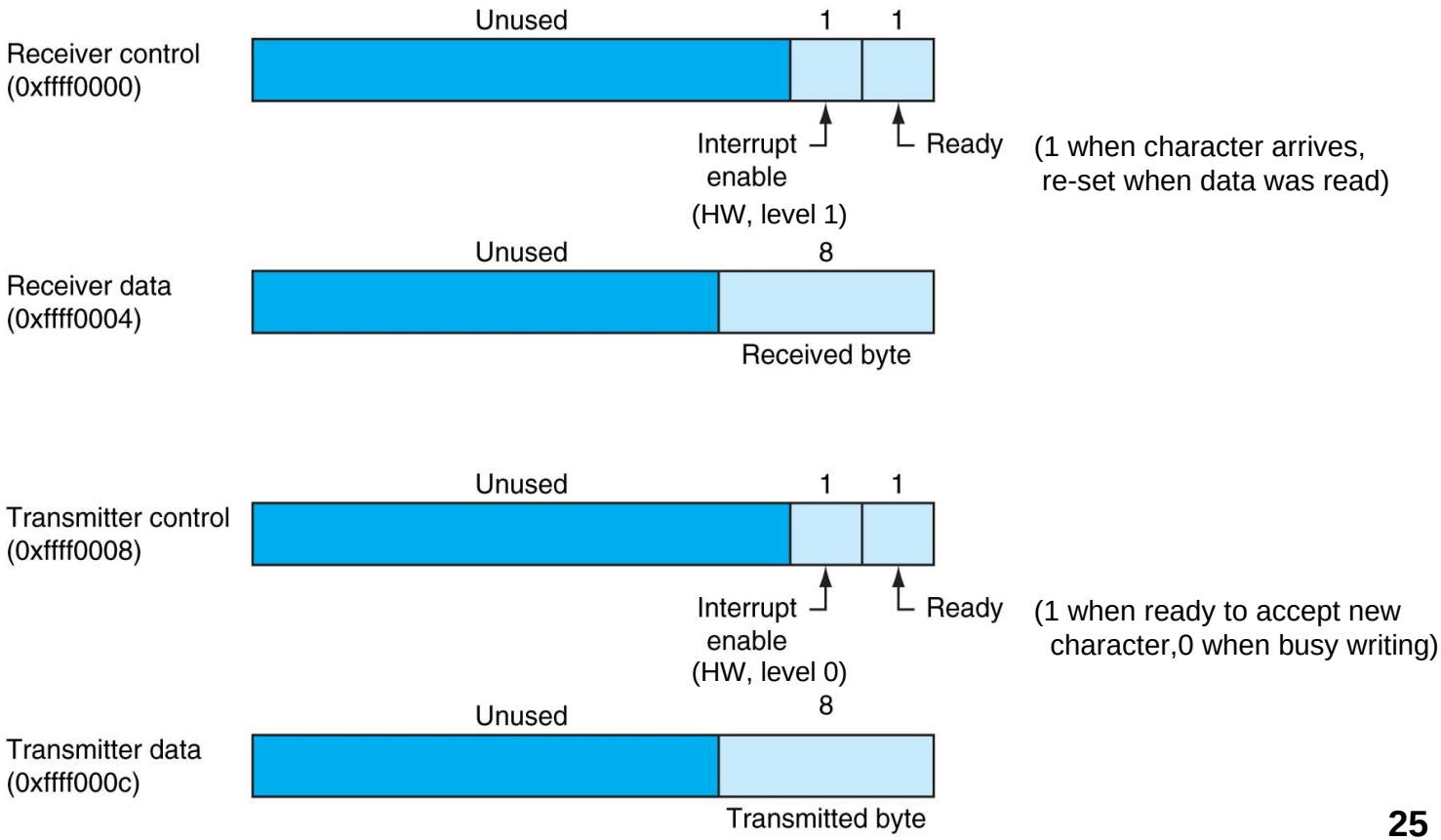
Status register `(CP0 $12)`



Cause register `(CP0 $13)`

memory-mapped I/O

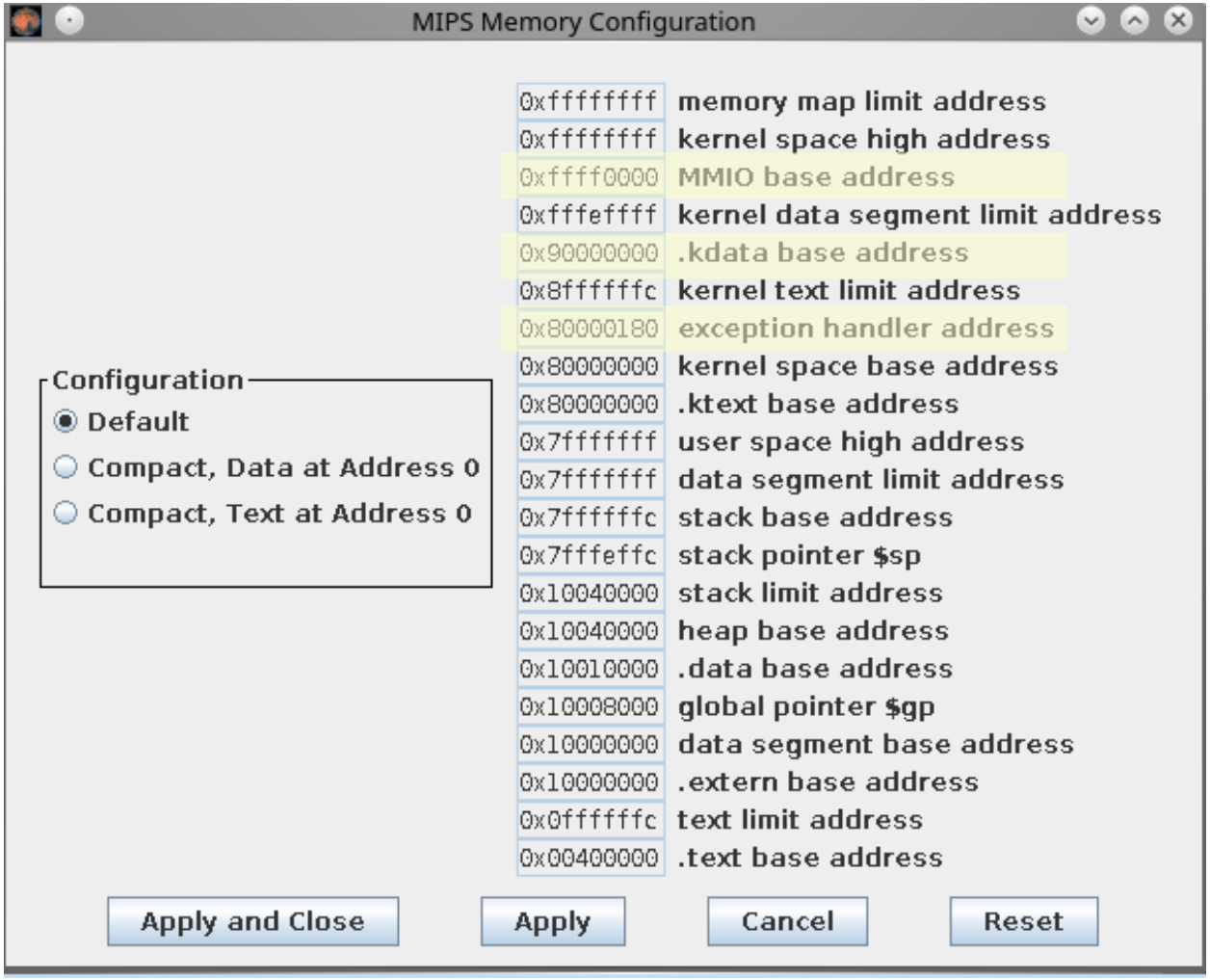
- I/O via read/write of “device registers”
- Appear at special memory locations
- Accessed using standard memory `lw/sw` operations



memory-mapped I/O, interrupt-driven

```
# Interrupt handler for keyboard (input) interrupt:

e_int_timer_end:
  mfc0 $v0, $13          # Cause
  andi $v0, $v0, 0x0100 # mask (select) pending interrupt bit 8 (HW interrupt)
  beq $v0, $zero, e_int_keyrecv_end # not keyboard interrupt

  # handle keyboard receive interrupt
  mfc0 $a0, $13      # coprocessor0 Cause register
  xor $a0, $a0, $v0 # set pending interrupt bit 8 to 0 (and keep other bits)
  mtc0 $a0, $13      # reset Cause (removing pending HW interrupt)

  la $a0, hw_int_keyboard # "keyboard input\n"
  jal print_string

  li $a0, 0xFFFF0004 # Receiver data address (interrupt based,
                     # so don't need to check Receiver control)
  lw $v0, 0($a0)  # Receiver data
  la $a0, char    # space for one character
  sb $v0, 0($a0)  # store Received data (key pressed)
                  # note: accessing data re-sets Ready bit
                  #       in Receiver control
  la $a0, key     # key pressed message/character
                  # remember that the following were declared:
                  #   key: .ascii "\t\tkey pressed: " # not .asciiz!
                  #   char: .ascii " "
                  #   nl: .asciiz "\n"
  jal print_string

e_int_keyrecv_end:
```

Memory Layout



Exception Handling and I/O

```
##################################
# Example of exception handling  #
# and memory-mapped I/O          #
##################################

##################################
# Must enable memory-mapped I/O! #
##################################


#########################
# Handler (Kernel) Data #
#########################

        .kdata
        .align  4
ktemp:  .space  16  # allocate 4 consecutive words, word aligned,
                    # with storage uninitialized, for temporary saving
                    # (stack can't be used as it may be corrupt
                    #  and may even be the cause of the exception)

hex:    .ascii  "0123456789ABCDEF"  # table for quick hex conversion

exc:    .ascii  "\texception type:" # not .asciiz!
spc:    .asciiz " "
epc:    .asciiz "EPC: "
status: .asciiz " Status: "
cause:  .asciiz " Cause: "
count:  .asciiz " Count: "
hw_int:          .asciiz "\tHardware Interrupt,  "
hw_int_timer:    .asciiz "timer\n"
hw_int_keyboard: .asciiz "keyboard input\n"
syscall_string:  .asciiz "\tsyscall "
timer:  .asciiz "\ttimer expired... and reset\n"
key:    .ascii  "\t\tkey pressed: " # not .asciiz!
char:   .ascii  " "
nl:     .asciiz "\n"
```

Exception Handling and I/O

```
# Handler Implementation (in Kernel) #

        # Overwrites existing exception handler
        .ktext 0x80000180

        mfc0   $k0, $12 # get status
                        # $k0 and $k1 are reserved for
                        # OS and Exception Handling
        andi   $k0, 0xfffffffe  # Disable interrupts while in interrupt handler
                                # by setting interrupt enable bit in status register to 0
        mtc0   $k0, $12         # update status

        .set   noat     # tell assembler not to use $at (assembler temporary)
                        # and hence not to complain when we do
        move   $k0, $at # save $at (used in pseudo-instructions) in $k0
                        # programmer should not use them, so not saved
        .set   at       # tell assembler it may use $at again

        la     $k1, ktemp   # address of temporary save area
                            # in exception handler. The stack can NOT be used
                            # as the stack pointer/stack content may be corrupt!
                            # Consequence: exception handler NOT re-entrant!
        sw     $a0, 0($k1)  # save $a0 as we'll use it
        sw     $a1, 4($k1)  # save $a1 as we'll use it
        sw     $v0, 8($k1)  # save $v0 as we'll use it
        sw     $ra, 12($k1) # save $ra as we'll use it

 # coprocessor0 (exception handling) registers
 #
 #    Name      Register Description    (*) simulated by MARS
 #
 # (*)BadVAddr  $8 offending memory reference
 #    Count      $9 current timer; incremented every 10ms
 #    Compare   $11 interrupt when Count == Compare
 # (*)Status    $12 controls which interrupts are enabled (vs. masked)
 # (*)Cause     $13 exception type, and pending interrupts
 # (*)EPC       $14 PC where exception/interrupt occured
```

Exception Handling and I/O

```
la    $a0, epc        # "EPC: "
jal   print_string    # (no print syscall interrupt from exception handler!)

mfc0  $a0, $14        # coprocessor0 EPC register:
                      # address of instruction that caused exception
jal   print_hex

la    $a0, status     # "Status: "
jal   print_string

mfc0  $a0, $12        # coprocessor0 Status register
jal   print_hex

la    $a0, cause      # "Cause: "
jal   print_string

mfc0  $a0, $13        # coprocessor0 Cause register
jal   print_hex

la    $a0, count      # "Count: "
jal   print_string

mfc0  $a0, $9         # coprocessor0 timer register
jal   print_hex

la    $a0, nl         # "\n"
jal   print_string

mfc0  $a0, $13        # coprocessor0 Cause register
andi  $v0, $a0, 0x7C  # Cause bits [6:2] contain Exception type
```

Exception Handling and I/O

```
# Exception type
#
# Number Name Description
#
#    0   Int  Hardware interrupt pending
#
#    4   AdEL Address error on load (or instruction fetch)
#    5   AdES Address error on store
#    6   IBE  Bus error on instruction fetch
#    7   DBE  Bus error on data load or store
#    8   Sys  syscall exception; in MARS, only for "new" syscall types ($v0 > 59)
#    9   Bp   breakpoint (usually used by debuggers, but also by div)
#
#   12   Ov   Arithmetic overflow

        la    $a0, exc       # "\texception type:"
        jal   print_string

        mfc0  $a0, $13       # coprocessor0 Cause register
        srl   $a0, $a0, 2    # Exception code starts at bit 2
        andi  $a0, $a0, 0x1F # mask the 5 exception code bits
        jal   print_hex

        la    $a0, nl        # "\n"
        jal   print_string

        # following two lines need to be re-done as $v0 (and $a0) got over-written in print_hex/print_string
        mfc0  $a0, $13       # coprocessor0 Cause register
        andi  $v0, $a0, 0x7C # Cause bits [6:2] contain Exception type

        beq   $v0, $zero, e_int  # handle hardware interrupt (exception type 0)
```

Exception Handling and I/O

```
# Program exception (i.e., not hardware interrupt)

# here: know what the cause was and could deal with it
# ...
# for example, when cause was 4 (AdEL) or 5(AdES)
# print offending memory address from coprocessor0 register $8 (BadVAddr)
# ...

# skip offending instruction
mfc0  $v0, $14    # EPC: address of instruction that caused exception
addiu $v0, $v0, 4 # next sequential instruction (caveat: delayed branch)
mtc0  $v0, $14    # update EPC (needed for "exception return" eret)

# following two lines need to be re-done as $v0 (and $a0)
# got over-written in print_hex/print_string
mfc0  $a0, $13       # coprocessor0 Cause register
andi  $v0, $a0, 0x7C # Cause bits [6:2] contain Exception type

# syscall exception
srl   $v0, $v0, 2
beq   $v0, 8, e_syscall # handle non-builtin syscall (in MARS, $v0 > 59)

j     e_int_end
```

Exception Handling and I/O

```
e_syscall: # handle syscall
    la    $a0, syscall_string # "syscall"
    jal   print_string

    la    $k1, ktemp   # address of temporary save area

    lw    $a0, 8($k1)  # saved $v0
    jal   print_hex

    la    $a0, spc     # " "
    jal   print_string

    la    $k1, ktemp   # address of temporary save area

    lw    $a0, 0($k1)  # saved $a0
    jal   print_hex

    la    $a0, nl      # "\n"
    jal   print_string

    j     e_int_end
```

Exception Handling and I/O

```
e_int:  # hardware (HW) interrupt handler

        la    $a0, hw_int # "\tHardware Interrupt,  "
        jal   print_string

        mfc0  $v0, $13                    # Cause
        andi  $v0, $v0, 0x8000            # mask (select) pending interrupt bit 15
        beq   $v0, $zero, e_int_timer_end # not timer interrupt

        # handle timer interrupt
        # note: timer not supported by MARS (but it is by SPIM)!

        mfc0  $a0, $13      # coprocessor0 Cause register
        xor   $a0, $a0, $v0 # set pending interrupt bit 15 to 0
        mtc0  $a0, $13      # reset Cause (removing pending HW interrupt)

        la    $a0, hw_int_timer # "timer\n"
        jal   print_string

        # reset timer to 0
        mtc0  $zero, $9     # set Count

        la    $a0, timer    # timer reset notice
        jal   print_string

        j     e_int_end
```

Exception Handling and I/O

```
e_int_timer_end:

        mfc0  $v0, $13                      # Cause
        andi  $v0, $v0, 0x0100              # mask (select) pending interrupt bit 8
        beq   $v0, $zero, e_int_keyrecv_end # not keyboard interrupt

        # handle keyboard receive interrupt

        mfc0  $a0, $13        # coprocessor0 Cause register
        xor   $a0, $a0, $v0   # set pending interrupt bit 8 to 0
        mtc0  $a0, $13        # reset Cause (removing pending HW interrupt)

        la    $a0, hw_int_keyboard # "keyboard input\n"
        jal   print_string

        li    $a0, 0xFFFF0004 # Receiver data address (interrupt based,
                              # so don't need to check Receiver control)
        lw    $v0, 0($a0)     # Receiver data
        la    $a0, char       # space for one character
        sb    $v0, 0($a0)     # store Received data (key pressed)
                              # note: accessing data re-sets Ready bit
                              # in Receiver control

        la    $a0, key        # key pressed message/character
        jal   print_string
```

Exception Handling and I/O

```
e_int_keyrecv_end:

e_int_end:

        # restore saved values
        la    $k1, ktemp
        lw    $a0, 0($k1)
        lw    $a1, 4($k1)
        lw    $v0, 8($k1)
        lw    $ra, 12($k1)

        .set  noat      # tell assembler not to use $at
                        # and hence not to complain when we do
        move  $at, $k0  # restore $at
        .set  at        # tell assembler it may use $at again

        mtc0  $zero, $13# re-set Cause, including all pending interrupts

        mfc0  $k0, $12  # Status
        ori   $k0, 0x01 # re-enable interrupts
        mtc0  $k0, $12  # update Status
        eret  # return from exception,
              # PC <- EPC after key pressed, continue where left off
              # PC <- EPC+4 after skipping offending instruction
```

Exception Handling and I/O

```
##############################
# print_string implementation #
##############################

print_string: # $a0: address of zero-terminated ascii string (.asciiz) to print
        j ps_cond                 # jump to code to
                                  #  * load next character to print
                                  #  * check if end of string (loaded char is 0x00)
ps_loop:
        lw    $v0, 0xFFFF0008     # Transmitter control
        andi  $v0, $v0, 0x01      # mask (select) Ready bit
        beq   $v0, $zero, ps_loop # (bus) loop until ready to print
        sw    $a1, 0xFFFF000C     # data (byte) to print into Transmitter data
ps_cond:
        lbu   $a1, ($a0)          # load character to print
        addi  $a0, $a0, 1         # increment char pointer
        bne   $a1, $zero, ps_loop # loop as long as not EndOfString (0x00) found
        jr    $ra                 # return from subroutine
```

Exception Handling and I/O

```
############################
# print_hex implementation #
############################

print_hex:  # $a0: word (32 bits long) to print
        la    $a1, hex            # address of hex conversion table
        li    $v0, 28             # printing a word (32 bits)
                                  # per nibble (4 bits = 1 hex character)
                                  # from leftmost to rightmost nibble
ph_loop:
        lw    $k1, 0xFFFF0008     # Transmitter control
        andi  $k1, $k1, 0x01      # mask (select only the) Ready bit
        beq   $k1, $zero, ph_loop # (busy) loop until ready to print

        srlv  $k1, $a0, $v0       # shift right logical variable (in reg) amount
        andi  $k1, $k1, 0x0f      # mask bits [3:0]
        add   $k1, $a1, $k1       # use $k1 as index in hex conversion table
        lbu   $k1, ($k1)          # load that character into $k1
        sw    $k1, 0xFFFF000C     # data (byte) to print into Transmitter data

        addi  $v0, $v0, -4        # next nibble (4 bits = 1 hex character)
        bge   $v0, $zero, ph_loop # loop until nothing left
        jr    $ra                 # return from subroutine
```

Exception Handling and I/O

```
#######################
# Program Entry Point #
#######################

        .text
        .globl main
main:
        li    $a0, 0xFFFF0000 # Receiver control
        lw    $t0, 0($a0)
        ori   $t0, 0x02       # set bit 1 to enable input interrupts
                              # such a-synchronous I/O (handling of keyboard input in this case)
                              # this is much more efficient than the "polling" we use for output
                              # In particular, it does not "block" the main program if there is no input
        sw     $t0, 0($a0)    # update Receiver control

        mfc0   $t0, $12   # load coprocessor0 Status register
        ori    $t0, 0x01  # set interrupt enable bit
        mtc0   $t0, $12   # move into Status register

        li     $t0, 100
        mtc0   $t0, $11   # coprocessor0 Compare register
                          # value is compared against timer
                          # interrupt when Compare ($11) and Count ($9) match
        mtc0   $zero, $9  # Count = 0
                          # Count (timer) will be incremented every 10ms
                          # hence, a timeout interrupt will occur after
                          # 100 x 10ms = 1s
                          # This should catch an infinite loop ...
                          # ... 1s = 1ns x 10^9

        # try I/O using own syscall
        li   $a0, 0xFFFF
        li   $v0, 60      # print integer in hexadecimal
        syscall
```

Exception Handling and I/O

```
        # divide by zero
        div     $t0, $t0, $zero
        move    $a0, $t0
        li      $v0, 60       # print integer in hexadecimal
        syscall
        move    $a0, $t1
        li      $v0, 60       # print integer in hexadecimal
        syscall

        # arithmetic overflow
        li      $t1, 0x7FFFFFFF
        addi    $t1, $t1, 1

        # non-existing memory address -- address error store
        sw      $t2, 124($zero)

        # non-aligned address -- address error load
        lw      $t2, 125($zero)

        # illegal instruction
        #.word 0xDEADBEEF  # hexspeak, "magic" value on some platforms :)

# infinite loop

forever:
        nop
        nop
        j forever
```

running in MARS

Edit | Execute

Text Segment

Bkpt	Address	Code	Basic	Source
	0x0040003c	0x0000000d	break	
	0x00400040	0x0100001a	div $8,$0	
	0x00400044	0x00004012	mflo $8	
	0x00400048	0x00082021	addu $4,$0,$8	346: move $a0, $t0
	0x0040004c	0x2402003c	addiu $2,$0,0x0000003c	347: li $v0, 60 # print integer in hexadecimal
	0x00400050	0x0000000c	syscall	348: syscall
	0x00400054	0x00092021	addu $4,$0,$9	349: move $a0, $t1
	0x00400058	0x2402003c	addiu $2,$0,0x0000003c	350: li $v0, 60 # print integer in hexadecimal
	0x0040005c	0x0000000c	syscall	351: syscall
	0x00400060	0x3c017fff	lui $1,0x00007fff	355: li $t1, 0x7FFFFFFF
	0x00400064	0x3429ffff	ori $9,$1,0x0000ffff	
	0x00400068	0x21290001	addi $9,$9,0x00000001	356: addi $t1, $t1, 1
	0x0040006c	0xac0a007c	sw $10,0x0000007c($0)	359: sw $t2, 124($zero)
	0x00400070	0x8c0a007d	lw $10,0x0000007d($0)	362: lw $t2, 125($zero)
	0x00400074	0x00000000	nop	370: nop
	0x00400078	0x00000000	nop	371: nop
	0x0040007c	0x0810001d	j 0x00400074	372: j forever
	0x80000180	0x401a6000	mfc0 $26,$12	47: mfc0 $k0, $12 # get status
	0x80000184	0x3c01ffff	lui $1,0xffffffff	48: andi $k0, 0xfffffffe # Disable interrupts while in interr
	0x80000188	0x3421fffe	ori $1,$1,0x0000fffe	
	0x8000018c	0x0341d024	and $26,$26,$1	
	0x80000190	0x409a6000	mtc0 $26,$12	49: mtc0 $k0, $12 # update status
	0x80000194	0x0001d021	addu $26,$0,$1	53: move $k0, $at # save $at in $k0
	0x80000198	0x3c019000	lui $1,0xffff9000	59: la $k1, ktemp # address of temporary save area
	0x8000019c	0x343b0000	ori $27,$1,0x00000000	
	0x800001a0	0xaf640000	sw $4,0x00000000($27)	63: sw $a0, 0($k1) # save $a0 as we'll use it
	0x800001a4	0xaf650004	sw $5,0x00000004($27)	64: sw $a1, 4($k1) # save $a1 as we'll use it
	0x800001a8	0xaf620008	sw $2,0x00000008($27)	65: sw $v0, 8($k1) # save $v0 as we'll use it
	0x800001ac	0xaf7f000c	sw $31,0x0000000c($27)	66: sw $ra, 12($k1) # save $ra as we'll use it
	0x800001b0	0x3c019000	lui $1,0xffff9000	79: la $a0, epc # "EPC: "
	0x800001b4	0x34240032	ori $4,$1,0x00000032	
	0x800001b8	0x0c0000da	jal 0x80000368	80: jal print_string # (no print syscall from exception hand
	0x800001bc	0x40047000	mfc0 $4,$14	82: mfc0 $a0, $14 # coprocessor0 EPC register:
	0x800001c0	0x0c0000e5	jal 0x80000394	84: jal print_hex
	0x800001c4	0x3c019000	lui $1,0xffff9000	86: la $a0, status # "Status: "
	0x800001c8	0x34240038	ori $4,$1,0x00000038	
	0x800001cc	0x0c0000da	jal 0x80000368	87: jal print_string
	0x800001d0	0x40046000	mfc0 $4,$12	89: mfc0 $a0, $12 # coprocessor0 Status register
	0x800001d4	0x0c0000e5	jal 0x80000394	90: jal print_hex
	0x800001d8	0x3c019000	lui $1,0xffff9000	92: la $a0, cause # "Cause: "
	0x800001dc	0x34240042	ori $4,$1,0x00000042	
	0x800001e0	0x0c0000da	jal 0x80000368	93: jal print_string
	0x800001e4	0x40046800	mfc0 $4,$13	95: mfc0 $a0, $13 # coprocessor0 Cause register
	0x800001e8	0x0c0000e5	jal 0x80000394	96: jal print_hex
	0x800001ec	0x3c019000	lui $1,0xffff9000	98: la $a0, count # "Count: "
	0x800001f0	0x3424004b	ori $4,$1,0x0000004b	
	0x800001f4	0x0c0000da	jal 0x80000368	99: jal print_string
	0x800001f8	0x40044800	mfc0 $4,$9	101: mfc0 $a0, $9 # coprocessor0 timer register
	0x800001fc	0x0c0000e5	jal 0x80000394	102: jal print_hex
	0x80000200	0x3c019000	lui $1,0xffff9000	104: la $a0, nl # "\n"

Registers | Coproc 1 | Coproc 0

Name	Number	Value
$8 (vaddr)	8	0x0000000
$12 (status)	12	0x0000ff1
$13 (cause)	13	0x0000000
$14 (epc)	14	0x0000000

Keyboard and Display MMIO Simulator, Version 1.4

Keyboard and Display MMIO Simulator

DISPLAY: Store to Transmitter Data 0xffff000c, cursor 1276, area 103 x 33

```
EPC: 00400034 Status: 0000FF12 Cause: 00000020 Count: 00000000
        exception type: 00000008
        syscall 0000003C 0000FFFF
EPC: 0040003C Status: 0000FF12 Cause: 00000024 Count: 00000000
        exception type: 00000009
EPC: 00400050 Status: 0000FF12 Cause: 00000020 Count: 00000000
        exception type: 00000008
        syscall 0000003C 00000000
EPC: 0040005C Status: 0000FF12 Cause: 00000020 Count: 00000000
        exception type: 00000008
        syscall 0000003C 00000000
EPC: 00400068 Status: 0000FF12 Cause: 00000030 Count: 00000000
        exception type: 0000000C
EPC: 0040006C Status: 0000FF12 Cause: 00000014 Count: 00000000
        exception type: 00000005
EPC: 00400070 Status: 0000FF12 Cause: 00000010 Count: 00000000
        exception type: 00000004
EPC: 00400074 Status: 0000FF12 Cause: 00000100 Count: 00000000
        exception type: 00000000
        Hardware Interrupt,  keyboard input
                key pressed: A
EPC: 0040007C Status: 0000FF12 Cause: 00000100 Count: 00000000
        exception type: 00000000
        Hardware Interrupt,  keyboard input
                key pressed: B
EPC: 0040007C Status: 0000FF12 Cause: 00000100 Count: 00000000
        exception type: 00000000
        Hardware Interrupt,  keyboard input
                key pressed: C
EPC: 00400074 Status: 0000FF12 Cause: 00000100 Count: 00000000
        exception type: 00000000
        Hardware Interrupt,  keyboard input
                key pressed: D
```

Font | ☑ DAD | Fixed transmitter delay, select using slider

KEYBOARD: Characters typed here are stored to Receiver Data 0xffff0004

ABCD

Tool Control

Disconnect from MIPS | Reset | Help | Close

Exception Code (in cause register)

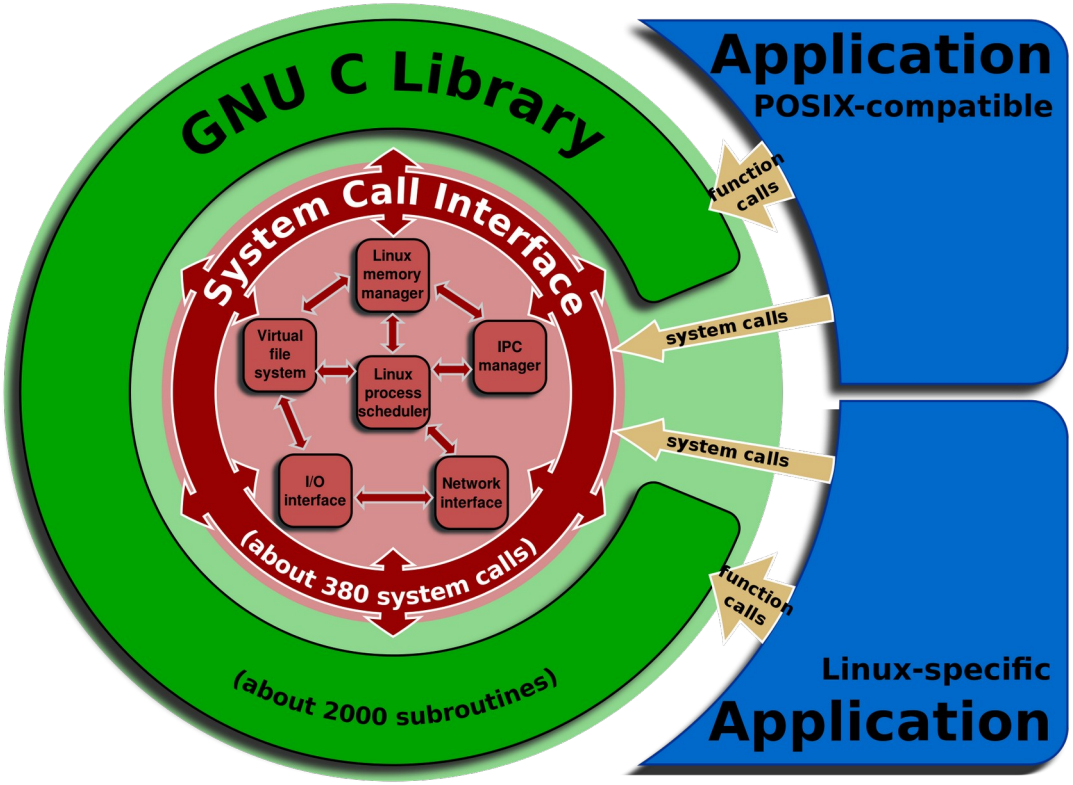
Number	Name	Cause of exception
0	Int	Hardware interrupt pending
4	AdEL	Address Error on Load or instruction fetch
5	AdES	Address Error on Store
6	IBE	Bus Error on Instruction fetch
7	DBE	Bus Error on Data load or store
8	Sys	Syscall exception (in MARS only for `$v0 > 59`)
9	Bp	Breakpoint (usually used by debuggers)
10	CpU	Coprocessor Unimplemented
12	Ov	Arithmetic overflow
13	Tr	Trap
15	FPE	Floating Point Exception

System Services (syscall)

Service	System call code	Arguments	Result
print_int	1	$a0 = integer	
print_float	2	$f12 = float	
print_double	3	$f12 = double	
print_string	4	$a0 = string	
read_int	5		integer (in $v0)
read_float	6		float (in $f0)
read_double	7		double (in $f0)
read_string	8	$a0 = buffer, $a1 = length	
sbrk	9	$a0 = amount	address (in $v0)
exit	10		
print_char	11	$a0 = char	
read_char	12		char (in $v0)
open	13	$a0 = filename (string), $a1 = flags, $a2 = mode	file descriptor (in $a0)
read	14	$a0 = file descriptor, $a1 = buffer, $a2 = length	num chars read (in $a0)
write	15	$a0 = file descriptor, $a1 = buffer, $a2 = length	num chars written (in $a0)
close	16	$a0 = file descriptor	
exit2	17	$a0 = result	

allocate memory on heap

`syscall`: link with OS



POSIX "Portable Operating System Interface [for Unix]" is the name of a family of related standards specified by the IEEE to define the application programming interface (API), along with shell and utilities interfaces for software compatible with variants of the Unix operating system, although the standard can apply to any operating system.

Link with OS

`syscall` **ASM-level vs. C-level (OS)**

Calling conventions

Register	use on input	use on output	Note
$at	—	(caller saved)	
$v0	syscall number	return value	
$v1	—	2nd fd only for pipe(2)	
$a0 ... $a2	syscall arguments	returned unmodified	O32
$a0 ... $a2, $a4 ... $a7	syscall arguments	returned unmodified	N32 and 64
$a3	4th syscall argument	$a3 set to 0/1 for success/error	
$t0 ... $t9	—	(caller saved)	
$s0 ... $s8	—	(callee saved)	
$hi, $lo	—	(caller saved)	

https://www.linux-mips.org/wiki/Syscall

syscall numbers

Compatibility ABIs

For compatibility ABIs Linux/MIPS obviously follows whatever the native OS is doing. This happens to be similar to what Linux/MIPS is doing or from a historical perspective, Linux/MIPS is following what other, earlier MIPS UNIX implementations were doing.

Syscall number ranges

OS flavor	First	Last
System V Release 4 flavored syscalls	0	999
System V syscalls. All flavors of IRIX use this number range as well.	1000	1999
BSD 4.3 syscalls	2000	2999
POSIX syscalls	3000	3999
Linux O32 syscalls	4000	4999
Linux N64 syscalls	5000	5999
Linux N32 syscalls	6000	6999

For the exact syscall numbers for the three Linux ABI, please see <uapi/asm/unistd.h> of your kernel.

https://www.linux-mips.org/wiki/Syscall

syscall numbers

path: root/arch/mips/include/uapi/asm/unistd.h

blob: f25dd1d83fb74700b33e4bf2387ebf89ac200f64 (plain)

```
/* SPDX-License-Identifier: GPL-2.0 WITH Linux-syscall-note */
/*
 * This file is subject to the terms and conditions of the GNU General Public
 * License.  See the file "COPYING" in the main directory of this archive
 * for more details.
 *
 * Copyright (C) 1995, 96, 97, 98, 99, 2000 by Ralf Baechle
 * Copyright (C) 1999, 2000 Silicon Graphics, Inc.
 *
 * Changed system calls macros _syscall5 - _syscall7 to push args 5 to 7 onto
 * the stack. Robin Farine for ACN S.A, Copyright (C) 1996 by ACN S.A
 */
#ifndef _UAPI_ASM_UNISTD_H
#define _UAPI_ASM_UNISTD_H

#include <asm/sgidefs.h>

#if _MIPS_SIM == _MIPS_SIM_ABI32

/*
 * Linux o32 style syscalls are in the range from 4000 to 4999.
 */
#define __NR_Linux                      4000
#define __NR_syscall                    (__NR_Linux +   0)
#define __NR_exit                       (__NR_Linux +   1)
#define __NR_fork                       (__NR_Linux +   2)
#define __NR_read                       (__NR_Linux +   3)
#define __NR_write                      (__NR_Linux +   4)
#define __NR_open                       (__NR_Linux +   5)
#define __NR_close                      (__NR_Linux +   6)
#define __NR_waitpid                    (__NR_Linux +   7)
#define __NR_creat                      (__NR_Linux +   8)
#define __NR_link                       (__NR_Linux +   9)
#define __NR_unlink                     (__NR_Linux +  10)
#define __NR_execve                     (__NR_Linux +  11)
#define __NR_chdir                      (__NR_Linux +  12)
#define __NR_time                       (__NR_Linux +  13)
#define __NR_mknod                      (__NR_Linux +  14)
```

```
#define __NR_statx                      (__NR_Linux + 366)
#define __NR_rseq                       (__NR_Linux + 367)
#define __NR_io_pgetevents              (__NR_Linux + 368)

/*
 * Offset of the last Linux o32 flavoured syscall
 */
#define __NR_Linux_syscalls             368

#endif /* _MIPS_SIM == _MIPS_SIM_ABI32 */
```

syscall from ASM

```
/*
 * hello-1.1/Makefile
 *
 * This file is subject to the terms and conditions of the GNU General Public
 * License.  See the file "COPYING" in the main directory of this archive
 * for more details.
 *
 * Copyright (C) 1995 by Ralf Baechle
 */
#include <asm/unistd.h>
#include <asm/asm.h>
#include <sys/syscall.h>

#define O_RDWR                02


        .set    noreorder
        LEAF(main)

#       fd = open("/dev/tty1", O_RDWR, 0);
        la      a0,tty
        li      a1,O_RDWR
        li      a2,0
        li      v0,SYS_open
        syscall

        bnez    a3,quit
        move    s0,v0                           # delay slot

#       write(fd, "hello, world.\n", 14);
        move    a0,s0
        la      a1,hello
        li      a2,14
        li      v0,SYS_write
        syscall

#       close(fd);
        move    a0,s0
        li      v0,SYS_close
        syscall

quit:
        li      a0,0
        li      v0,SYS_exit
        syscall

        j       quit
        nop

        END(main)

        .data
tty:    .asciz  "/dev/tty1"
hello:  .ascii  "Hello, world.\n"
```

recent version

```
#include <regdef.h>
#include <sys/asm.h>
#include <sys/syscall.h>

EXPORT(__start)


        .set    noreorder
        LEAF(main)
        li      a0,1
        la      a1,hello
        li      a2,12
        li      v0,__NR_write
        syscall

quit:
        li      a0,0
        li      v0,__NR_exit
        syscall

        j       quit
        nop

        END(main)

        .data

hello:  .ascii  "Hello world!\n"
```

https://www.linux-mips.org/wiki/Syscall

syscall from C

syscall C-level (OS) vs. ASM-level

```
SYSCALL(2)              Linux Programmer's Manual              SYSCALL(2)

NAME      top

       syscall - indirect system call

SYNOPSIS      top

       #include <sys/syscall.h>      /* Definition of SYS_* constants */
       #include <unistd.h>

       long syscall(long number, ...);

   Feature Test Macro Requirements for glibc (see
   feature_test_macros(7)):

       syscall():
           Since glibc 2.19:
               _DEFAULT_SOURCE
           Before glibc 2.19:
               _BSD_SOURCE || _SVID_SOURCE

DESCRIPTION      top

       syscall() is a small library function that invokes the system
       call whose assembly language interface has the specified number
       with the specified arguments.  Employing syscall() is useful, for
       example, when invoking a system call that has no wrapper function
       in the C library.

       syscall() saves CPU registers before making the system call,
       restores the registers upon return from the system call, and
       stores any error returned by the system call in errno(3).

       Symbolic constants for system call numbers can be found in the
       header file <sys/syscall.h>.

RETURN VALUE      top

       The return value is defined by the system call being invoked.  In
       general, a 0 return value indicates success.  A -1 return value
       indicates an error, and an error number is stored in errno.
```

Arch/ABI	Instruction	System call #	Ret val	Ret val2	Error	Notes
alpha	callsys	v0	v0	a4	a3	1, 6
arc	trap0	r8	r0	-	-	
arm/OABI	swi NR	-	r0	-	-	2
arm/EABI	swi 0x0	r7	r0	r1	-	
arm64	svc #0	w8	x0	x1	-	
blackfin	excpt 0x0	P0	R0	-	-	
i386	int $0x80	eax	eax	edx	-	
ia64	break 0x100000	r15	r8	r9	r10	1, 6
m68k	trap #0	d0	d0	-	-	
microblaze	brki r14,8	r12	r3	-	-	
mips	syscall	v0	v0	v1	a3	1, 6
nios2	trap	r2	r2	-	r7	

Arch/ABI	arg1	arg2	arg3	arg4	arg5	arg6	arg7	Notes
alpha	a0	a1	a2	a3	a4	a5	-	
arc	r0	r1	r2	r3	r4	r5	-	
arm/OABI	r0	r1	r2	r3	r4	r5	r6	
arm/EABI	r0	r1	r2	r3	r4	r5	r6	
arm64	x0	x1	x2	x3	x4	x5	-	
blackfin	R0	R1	R2	R3	R4	R5	-	
i386	ebx	ecx	edx	esi	edi	ebp	-	
ia64	out0	out1	out2	out3	out4	out5	-	
m68k	d1	d2	d3	d4	d5	a0	-	
microblaze	r5	r6	r7	r8	r9	r10	-	
mips/o32	a0	a1	a2	a3	-	-	-	1
mips/n32,64	a0	a1	a2	a3	a4	a5	-	

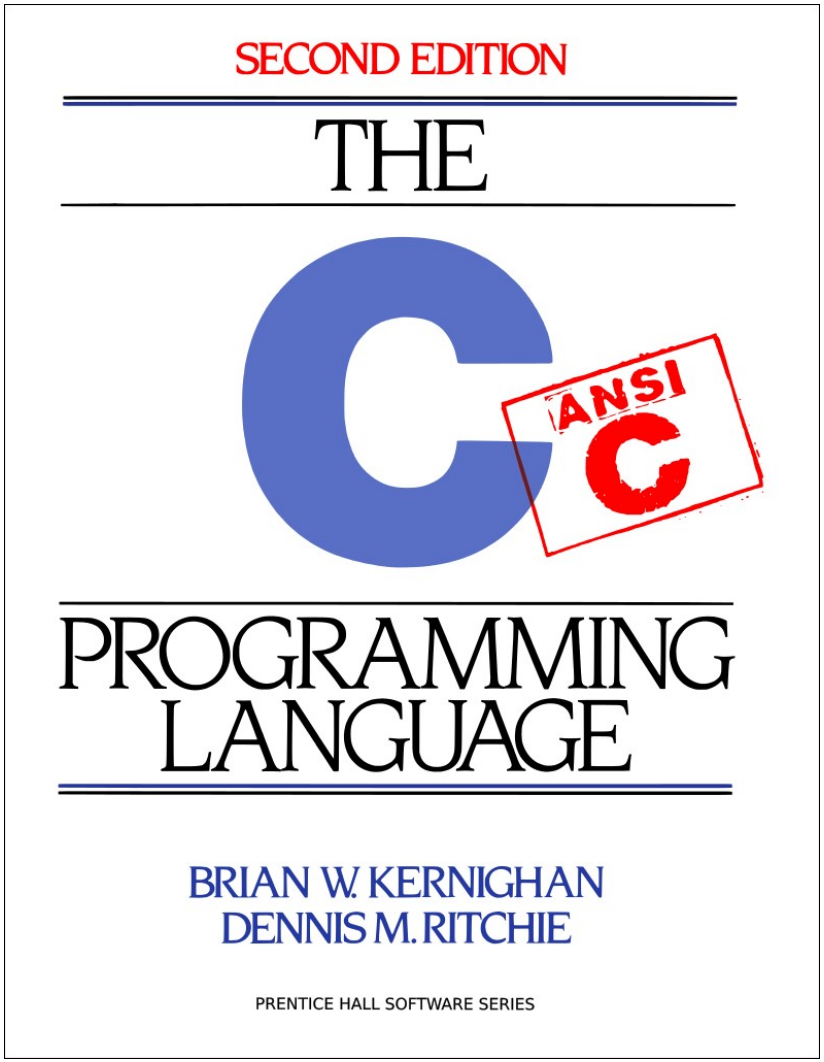
https://man7.org/linux/man-pages/man2/syscall.2.html

syscall from C (implementation in ASM)

```
/* MIPS syscall wrappers.
   Copyright (C) 2017-2022 Free Software Foundation, Inc.
   This file is part of the GNU C Library.

   The GNU C Library is free software; you can redistribute it and/or
   modify it under the terms of the GNU Lesser General Public
   License as published by the Free Software Foundation; either
   version 2.1 of the License, or (at your option) any later version.

   The GNU C Library is distributed in the hope that it will be useful,
   but WITHOUT ANY WARRANTY; without even the implied warranty of
   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
   Lesser General Public License for more details.

   You should have received a copy of the GNU Lesser General Public
   License along with the GNU C Library.  If not, see
   <https://www.gnu.org/licenses/>.  */

#include <sysdep.h>
#include <sys/asm.h>

        .text
        .set    nomips16

/* long long int __mips_syscall5 (long int arg1, long int arg2, long int arg3,
                                  long int arg4, long int arg5,
                                  long int number)  */

ENTRY(__mips_syscall5)
        lw      v0, 20(sp)
        syscall
        move    v1, a3
        jr      ra
END(__mips_syscall5)
libc_hidden_def (__mips_syscall5)
```

https://codebrowser.dev/glibc/glibc/sysdeps/unix/sysv/linux/mips/mips32/mips-syscall5.S.html

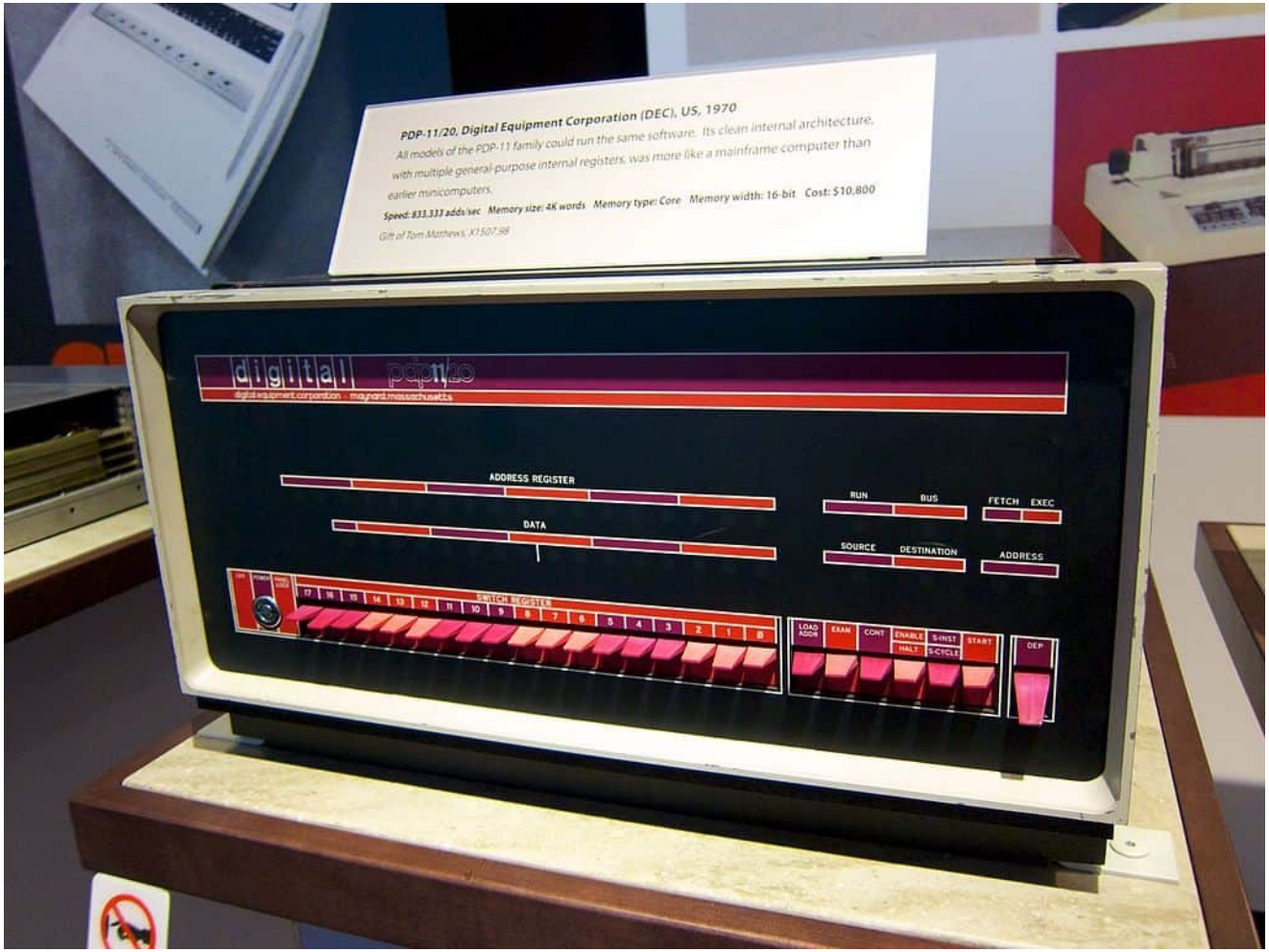
Link with OS: `(g)libc`



Link with OS: `(g)libc`

https://www.gnu.org/software/libc/manual/html_node/index.html

Link with OS: `booting`



https://digital.com/wp-content/uploads/1024px-DEC_PDP-11_20_computer_at_the_Computer_History_Museum.jpg
https://www.pdp-11.nl/pdp11-70startpage.html