Trabalho 5 - parte 2

- Exceções
  - Eventos que mudam o fluxo normal de execução, diferentes dos desvios ou saltos (*branchs/jump*), Podem ser de dois tipos (jargão do MIPS)
    - Assíncronas
      - Geradas externamente através da(s) entrada(s) de interrupção do processador
      - Podem acontecer a qualquer momento, sem relação temporal alguma com o programa em execução
    - Síncronas
      - Geradas internamente pela execução de uma instrução (e.g. instrução inválida, divisão por zero, *overflow*)
      - Acontecem sempre na execução da mesma instrução, considerando um certo estado do sistema (memória e registradores)

Trabalho 5 - parte 2

- Exceções
  - As exceções síncronas (internas) podem também ser geradas a partir de instruções específicas com a finalidade de chamar funções do *kernel* do sistema operacional
    - Chamadas de sistemas (*system calls*)
    - Tipicamente as aplicações não tem acesso ao hardware. Quando necessitam de alguma operação de entradas/saída (e.g. ler teclado, mostrar algo na tela, imprimir) elas chamam funções do *kernel* (chamadas de sistema) para realizar tais operações. O sistema operacional tem acesso a todos os recursos de hardware de um sistema computacional
    - Em um sistema real, não se pode fazer uma chamada de sistema com instruções de salto (e.g. *jal, jalr*) a partir da aplicação, porque o endereço destino está fora da área de memória da aplicação
      - Funções do kernel podem ser chamadas apenas através de intruções específicas

Trabalho 5 - parte 2

- Terminologia mais genérica
  - Interrupções
    - Desvio de execução para o *kernel* com origem externa
    - Geradas externamente através da(s) entrada(s) de interrupção do processador
  - *Traps*
    - Qualquer tipo de desvio de execução para o *kernel* com origem interna
      - Chamada de sistema: desvio intencional (planejado)
      - Exceção: desvio inesperado (e.g. divisão por zero, *overflow*)
  - Outras arquiteturas usam termos diferentes
    - Intel chama tudo de interrupção, diferindo apenas na origem (interna ou externa)

Trabalho 5 - parte 2

- Exceções
  - O objetivo do trabalho é adicionar suporte à exceções síncronas (*traps*) ao processador MIPS
  - Sempre que uma exceção síncrona ocorrer, a execução deve desviar para um endereço fixo onde iniciará a execução da rotina de tratamento de exceções (ESR - *Exception Service Routine*)
    - Ao entrar nesta rotina, as interrupções externas devem ser desabilitadas via *hardware* e reabilitadas no final através da instrução ERET
    - O endereço de retorno deve ser armazenado no EPC
  - O endereço da ESR deve ser armazenado no registrador ESR_AD através da instrução MTC0
    - Adicionar o registrador ESR_AD no *data path* e mapear no registrador C0[30]

Trabalho 5 - parte 2

- Exceções
  - Deve-se adicionar também o registrador CAUSE (C0[13]) o qual registra o identificador da exceção
    - 1: *Invalid Instruction*
    - 8: SYSCALL (instrução)
    - 12: *Overflow em complemento de 2*
      - *Deve ser gerado apenas pelas* instruções ADD*,* ADDI e SUB (implementar)
    - 15: *Division-by-zero (*gerada pela instrução DIVU*)*
  - A "causa" da exceção deve ser escrita via *hardware* no registrador CAUSE
  - Para possibiliar a leitura deste registrador, deve-se implementar a instrução MFC0

Trabalho 5 - parte 2

- Exceções
  - Para cada uma das quatro exceções, deve-se ter um *handler*
  - A chamada de um *handler* deve seguir a mesma idéia das interrupções (*jump table*)
    - Definir um *array* com os endereços dos *handlers*
    - Indexar o *array* utilizando o identificador lido do registrador CAUSE

Trabalho 5 - parte 2

- Exceções
  - Estrutura da rotina de tratamento de exceções

```
ExceptionServiceRoutine:
    1. Salvamento de contexto no PCB (todos regs menos 0, 26
   e 27)
    2. Ler do reg CAUSE o número da exceção
    3. Indexar jump table e gravar em algum
       registrador o endereço do handler
    4. jalr rs (chama handler)
    5. Recuperação de contexto do PCB
    6. Retorno (ERET)

Hanlder1:
    ...
    jr $ra
...
```

Na arquitetura MIPS, o endereço para o tratamento de exceções é o mesmo, idependente do tipo. A rotina tem que verificar se é interna ou externa.
Se quisermos implementar assim, basta setar o mesmo endereço nos registradores ISR_AD e EXC_AD.

Trabalho 5 - parte 2

- Exceções
  - A tarefa do *handler* da exceção é decidir se o problema é recuperável ou não
    - Se o problema não pode ser recuperado, a aplicação é abortada
    - Caso contrário, o *handler* tenta solucionar o problema e retorna para a aplicação
  - Para este trabalho, no caso das exceções *invalid instruction, division by zero e overflow*, o *handler* deve apenas enviar uma mensagem de erro para o módulo TX indicando qual exceção ocorreu e o endereço da instrução que causou a exceção (hexadecimal). Em seguida retornar ao programa.
    - Implementar a função *char *IntegerToHexString (int n)*
    - Formato da *string*: 0xXXXXXXXX
  - Instruções que causam estas exceções não devem ser concluidas
    - Nenhum registrador do banco de registradores deve ser alterado

Trabalho 5 - parte 2

- Exceções
  - As exceções que fazem chamadas de sistema são geradas a partir da instrução *syscall* (implementar)
  - Neste trabalho teremos três chamadas de sistema
    - *0: PrintString*
    - *1: IntegerToString*
    - *2: IntegerToHexString*
  - Para identificar a função a ser chamada, deve-se utilizar o registrador v0

```
li $v0, 0          # PrintString
la $a0, msg
syscall            # Salta para a rotina de tratameto de exceção
```

  - O *handler* da exceção SYSCALL deve verificar a função solicitada ($v0) e chamar a função correspondente
    - *Jump table* (definir *array* com os endereços das três funções)

Trabalho 5 - parte 2

- Exceções
  - Estrutura do *handler* SYSCALL

```
SystemCallHandler:
    1. Ler do reg v0 o número da função solicitada
    2. Indexar jump table e gravar em algum
       registrador o endereço da função
    3. jalr rs (chama função)
    4. jr ra
```

Trabalho 5 - parte 2

- Aplicação
  - Teste de exceções

Tipo de ordenação: crescente ou decrescente

```
main:
    1. jal BubbleSort(order)
    - Imprimir array inicial (syscall funções da biblioteca)
    - Delay
    - Imprimir arrar final (syscall funções da biblioteca)
    2. Forçar exceção add
       - Delay
    3. Forçar exceção addi
       - Delay
    4. Forçar exceção sub
       - Delay
    5. Forçar exceção instrução inválida
       - Delay
    6. not order (inverter ordenação do bubble sort)
    7. j main
```

Fora do BubbleSort

Pode-se adicionar mais mensagens e *delays* a fim de facilitar o entendimento do que está acontecendo

Trabalho 5 - parte 2

- Aplicação
  - Exemplo de exceções

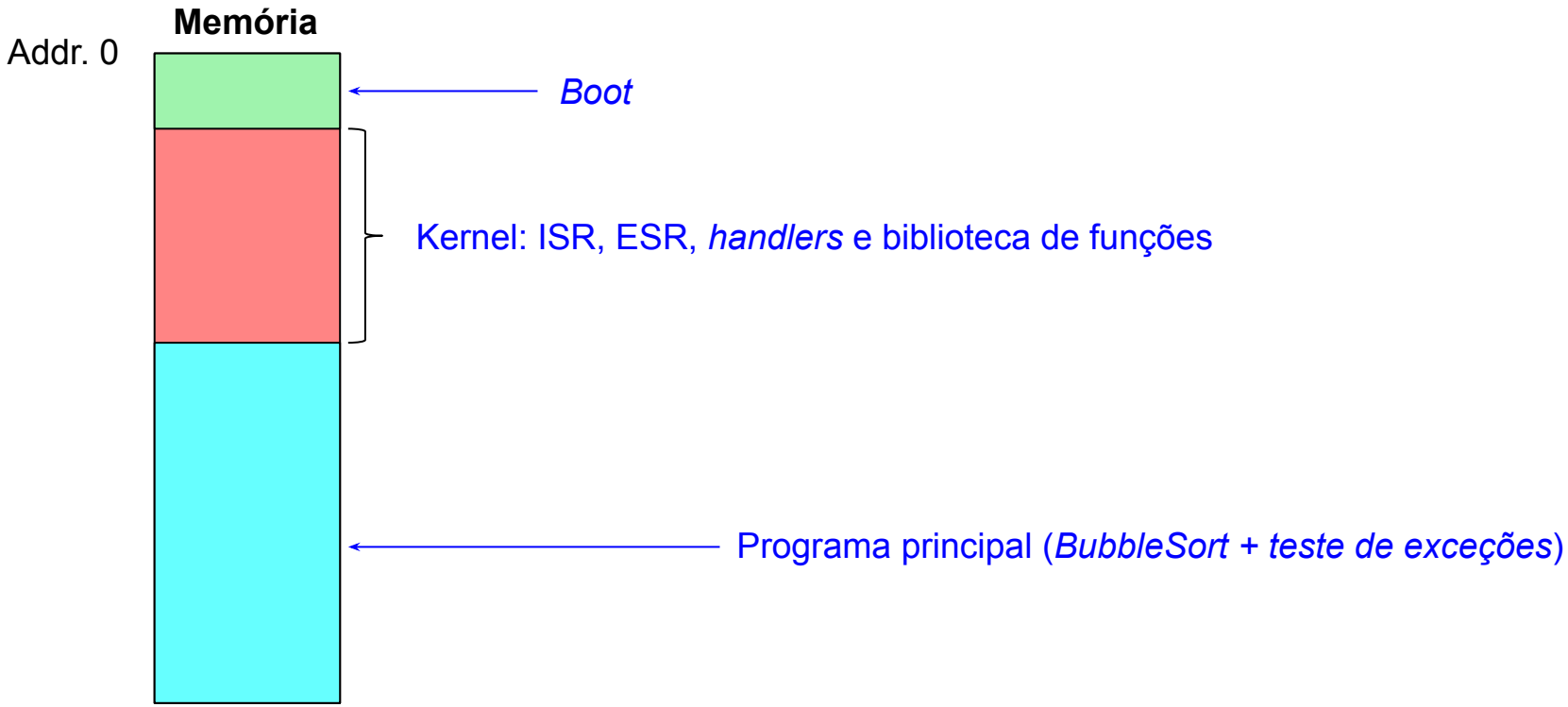
```
li $8, 0x7fffffff          # r8 <- greatest positive number (2's)
addiu $9, $8, 1            # Ignore overflow
addi $10, $8, 1            # Overflow!
li $8, 0x80000000          # r8 <- smallest negative number (2's)
addiu $9, $8, -1           # Ignore overflow
addi $10, $8, -1           # Overflow!
Adicionar instruções para testar SUB e ADD
divu $8, $0                # Division by 0
add.s $f0, $f1, $f2        # Invalid instruction
```

Trabalho 5 - parte 2

- Aplicação
  - O sistema deve ser prototipado utilizando a placa Nexys 3
    - Apenas o MIPS_uC (sem *CryptoMessages*)
  - Para visualização das mensagens geradas pelo sistema, deve-se utilizar algum terminal de comunicação serial como PuttY ou Hiper Terminal

Trabalho 5 - parte 2

- Estrutura do código na memória



Trabalho 5 - parte 2

- Chamadas de sistema tipicamente não são invocadas diretamente a partir de uma linguagem de alto nível porque dependem de instruções específicas que variam de acordo com o ISA do processador
  - MIPS: *syscall*; ARM: *swi*; Intel: *int*
- As linguagens de programação de alto nível disponibilizam sub-rotinas que possibilitam a invocação das chamadas de sistema
- Por exemplo, a chamada de sistema *write* (Linux, Mac OS X) é utilizada pelo kernel para escrever em dispositivos de saída (e.g. monitor, HD). Ela pode ser invocada através da função *write()* da *libc* padrão. Esta chamada de sistema é invocada por todas as funções de saída da *libc* como *printf(), fprintf(), putc(), fputc(), puts(), fputs()* através da função *write()*

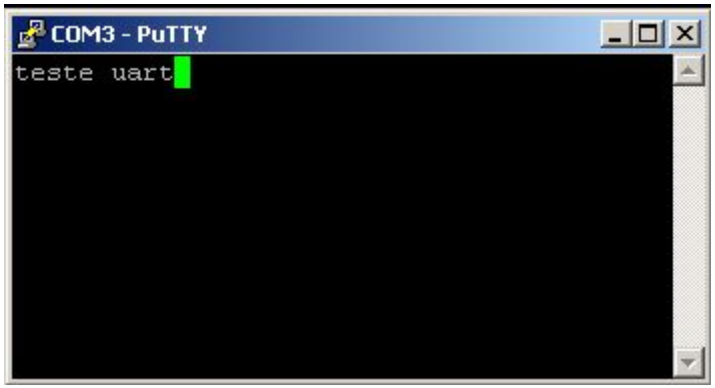
Trabalho 5 - parte 2

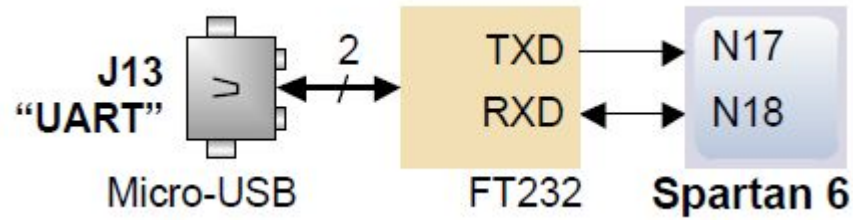
- write/read

```
#include <unistd.h>

int main() {

    char buffer[50];
    int nbytes;

    // printf("Hello world!\n");
    write(1,"Hello world!\n",13);
    // printf("Leitura de teclado: ");
    write(1,"Leitura de teclado: ",20);

    nbytes = read(0,buffer,50);  // gets(buffer);
    buffer[nbytes] = 0;          // Set the end-of-string
    write(1,buffer,nbytes);      // printf("%s",buffer);
    return 0;
}
```

Tarefa

- UART Terminal
  - Mostra no terminal caracter digitado no teclado
  - Utiliza comunicação serial segundo o padrão RS232
  - Moodle: uart_terminal.bit

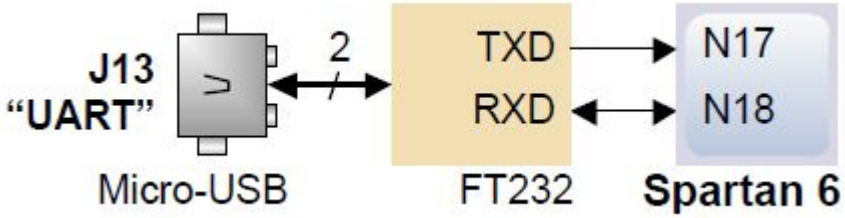


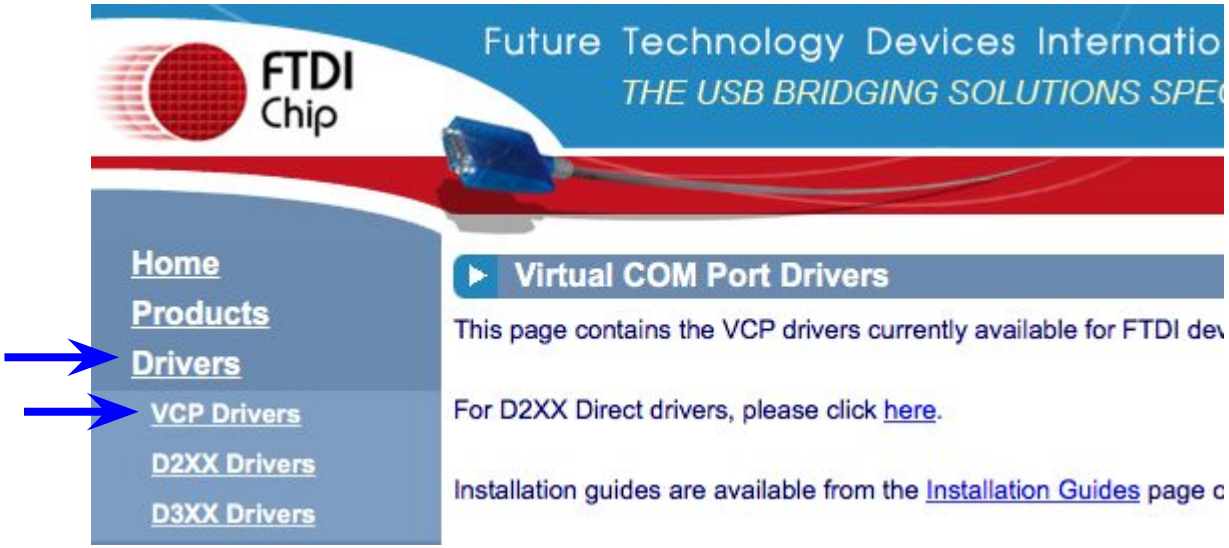


Necessita mais um cabo USB

Tarefa

- UART Terminal
  - Instalar *driver* USB-COM a fim utilizar a interface USB como uma porta serial
    - www.ftdichip.com
    - Mais detalhes no manual da placa

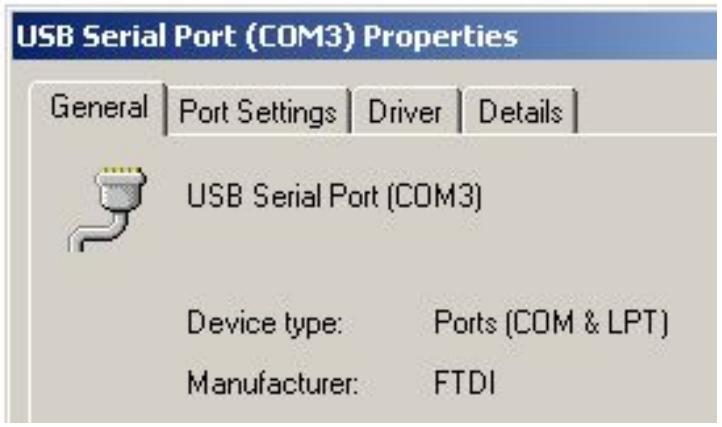




O window pode instalar o *driver* automaticamente ao plugar a placa no computador

Tarefa

- UART Terminal
  - Uma nova porta aparecerá no Gerenciador de Dispositivos do windows (*USB Serial Port*)
  - Deve-se verificar o número da porta COM que apareceu



Tarefa

- UART Terminal
  - Configurar terminal
    - PuTTY (www.putty.org)
    - Hyper Terminal