UFPR – Bacharelado em Ciência da Computação
CI064 – Software Básico
Trabalho Semestral: Driver para a Interface Serial — 2019-1 31 de maio de 2019

Sua tarefa é implementar um *driver* para a interface serial do cMIPS. O *driver* deve funcionar por interrupções nos dois sentidos, transmissão (tx) e recepção (rx).

A unidade remota enviará para o cMIPS uma sequência de inteiros entre 1 e 50. Para cada inteiro i recebido, seu programa deve transmitir para a unidade remota o i-ésimo número primo.

Para tanto, você deve escrever um programa `main()` (em C) que contém um vetor estático com os 50 primeiros (menores) números primos. Seu programa recebe uma série de inteiros, indexa o vetor de primos, e envia o número primo correspondente a cada inteiro recebido.

Após inicializar as estruturas de dados, seu programa dispara a comunicação com a remota (fazendo RTS=1), e então recebe pela interface serial uma sequência de *strings*, cada *string* representando um inteiro. Para cada *string* recebida, seu programa transmite, também pela interface serial, a *string* com o número primo que lhe corresponde.

O circuito da interface serial (*Universal Asynchronous Receiver-Transmitter* ou UART) é aquele descrito nas Seções 8.2 e 8.3 das notas de aula desta disciplina. Estude o texto com atenção.

A UART do cMIPS opera com *double buffering* nos dois sentidos de transmissão.

***Driver* para a UART** A Figura 1 mostra um diagrama de blocos de um *driver* simplificado para a UART. Este *driver* é dividido em duas partes: um tratador de interrupções (*handler*) e um conjunto de funções que permitem ao programador enviar e receber através da interface serial. A este conjunto de funções é que chamaremos de *driver*.

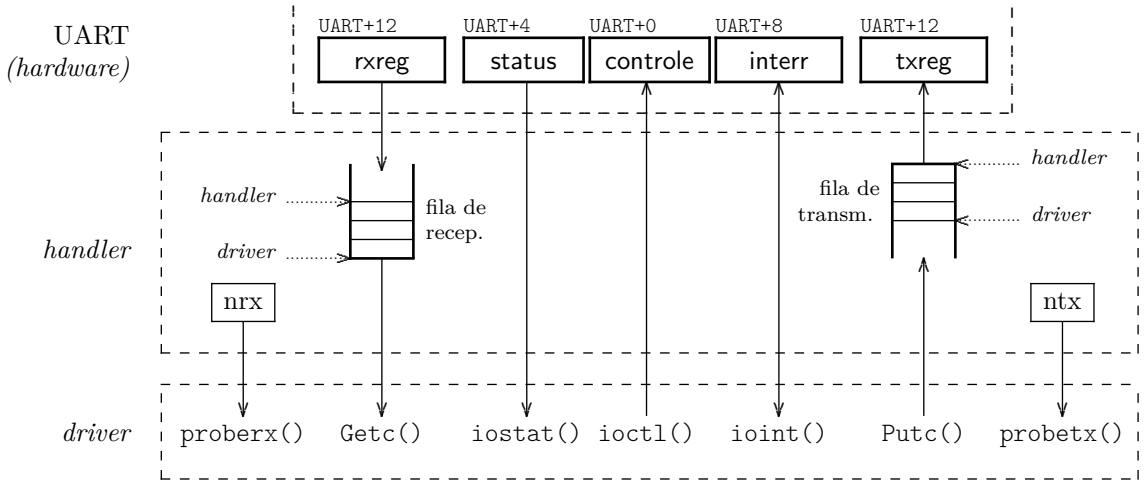

Figura 1: Diagrama de blocos do *driver* da UART.

O tratador de interrupções gerencia duas filas de octetos, uma associada à recepção, e outra à transmissão, cada uma com capacidade para 16 octetos. O tratador mantém dois contadores, `nrx` indica o número de octetos disponíveis na fila de recepção, enquanto que `ntx` indica o número de espaços na fila de transmissão. O tratador de interrupções deve ser escrito em *assembly* e seu código pode ser baseado nas notas de aula, e em `include/handlers.s`.

As funções `proberx()` e `probetx()` retornam os valores de `nrx` e `ntx`, respectivamente. A função `iostat()` retorna o conteúdo do registrador de status da UART, a função `ioctl()` permite escrever no seu registrador de controle, e `ioint()` permite atualizar o registrador de interrupções.

A função `initUART()` inicializa as estruturas de dados e variáveis do *driver* da interface serial.

A função `Getc()` retorna o octeto que está na cabeça da fila de recepção e decrementa `nrx`, ou `EOF` se a fila estiver vazia.

A função `Putc()` insere um octeto na fila de transmissão e decrementa `ntx`.

Estas funções devem estar contidas em um único arquivo, que contém a função `main()`. Os protótipos são listados abaixo. As funções compõem uma 'biblioteca' do SO que esconde do programador todos os detalhes sórdidos do uso da interface serial. Em `main()`, não pode haver nenhuma menção à estrutura `Ud`, e esta não pode ser declarada como uma variável global. Todas as referências à `Ud` devem estar escondidas nas funções da 'biblioteca'.

As definições para os tipos que modelam os registradores da UART estão em `include/uart_defs.h`.

```
void initUART(void);    // inicializa a UART e o driver
int  proberx(void);     // retorna nrx
int  probetx(void);     // retorna ntx
Tstatus iostat(void);   // retorna inteiro com status no byte menos sign.
void ioctl(Tcontrol);   // escreve byte menos sign no reg. de controle
int  ioint(Tinterr);    // escreve byte menos sign no reg. de controle
char Getc(void);        // retira octeto da fila, decrementa nrx
void Putc(char);        // insere octeto na fila, decrementa ntx
```

As funções `enableInterr()` e `disableInterr()` estão definidas em `include/handlers.s` e habilitam e desabilitam as interrupções; as duas retornam o conteúdo do registrador Status do processador *após* a alteração do bit Status.intEn. Estas funções devem ser usadas para impedir a execução concorrente do *driver* e do *handler*, quando o *driver* atualiza os ponteiros das filas circulares, `nrx` e `ntx`.

```
int  enableInterr(void);  // habilita interrupções, retorna STATUS
int  disableInterr(void); // desabilita interrupções, retorna STATUS
```

A Figura 2 mostra um diagrama com o fluxo dos dados do *driver* mais o tratador. A unidade remota lê o conteúdo do arquivo `serial.inp` e o transmite para a UART do cMIPS. No sentido contrário, o circuito de recepção da remota recebe os octetos enviados pela UART, e os exibe na saída padrão do simulador.

O tratador da interrupção, na recepção, lê um octeto do registrador de dados da UART (rxreg) e o insere na fila de recepção. Quando uma cadeia vazia for recebida, a entrada terminou.

A função `main()` remove um octeto da fila de recepção (`Getc()`) e o insere na fila de entrada. Quando um inteiro 'completo' é recebido, este é usado para indexar o vetor de primos, o número primo é convertido para uma *string* ($P(i)$), e esta é então transmitida, octeto a octeto através da fila de transmissão, com invocações sucessivas de `Putc()`.

O tratador de interrupção, na transmissão, remove um octeto da fila de transmissão e o escreve no registrador de dados da UART (txreg).

Interrupção de transmissão Se a fila de transmissão estiver vazia (`ntx=16`), e status.txEmpty=1, então `Putc()` deve inserir um octeto na fila de transmissão e provocar uma interrupção de transmissão, usando `ioint()`.

Esta interrupção provocará uma segunda interrupção logo após o primeiro octeto ter sido copiado para o registrador de transmissão da UART (txreg → transmit), por causa do *double buffering* no circuito de transmissão da UART.

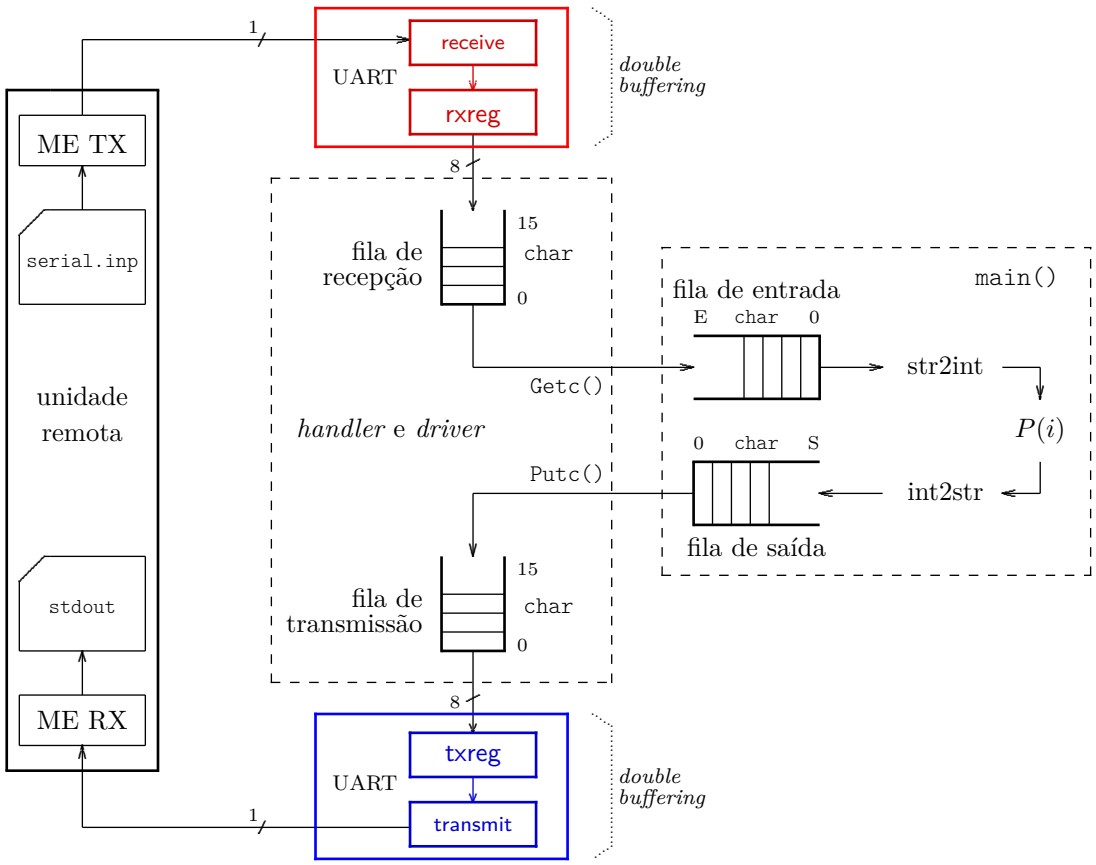

Figura 2: Fluxo de dados no *driver* da UART e programa de aplicação.

Das *strings* A entrada do seu programa consiste de uma sequência com 1 a 10 *strings*, cada *string* representa um inteiro de 1 a 50, mais um `'\n'`.

Neste trabalho as *strings* são terminadas por `'\n'` ao invés de por `'\0'`.

Uma cadeia vazia `"'\n'\n'"` indica o final dos dados.

A entrada é de somente dígitos (mais `'\n'`), representados por caracteres ASCII.

Das filas de octetos em `main()` Se um único octeto na entrada for perdido, seu programa produzirá resultados errados. Portanto, a recepção deve ter prioridade máxima. Isso é possível porque a especificação não impõe nenhuma restrição temporal quanto à produção dos resultados – mas entradas e saídas devem ser concorrentes, na medida do possível, e isso é possível.

A cada *string* recebida da entrada, a *string* correspondente a $P(s)$ deve ser inserida na fila de saída. Os octetos desta fila serão transmitidos com `Putc()` na medida do possível, por que a entrada tem maior prioridade que a saída.

Quais as capacidades das filas de entrada (`E`) e de saída `S`? Esta é uma pergunta fundamental.

Para minimizar o (looooongo) tempo de simulação, a taxa de transmissão é elevada, e só há tempo para executar de 300 a 500 instruções entre duas interrupções. Essa é uma restrição séria quanto à quantidade de trabalho que seu programa pode efetuar entre duas interrupções consecutivas.

Por causa da alta frequência de interrupções, a espera ocupada por novos octetos, e a espera ocupada por espaço para transmitir, devem ficar em `main()` para que seja fácil perceber em quais atividades o programa está gastando e/ou desperdiçando mais tempo.

A velocidade mínima de transmissão é obtida com `SPEED=5`.

Da implementação É recomendável que você descomente o `.include` em `include/handlers.s` e trabalhe diretamente com `tests/handlerUART.s` porque este arquivo não faz parte do controle de versões e portanto não há risco de você perder seu trabalho quando ocorrer alguma alteração no modelo do cMIPS, nos programas de teste, ou no código do 'SO' primitivo do cMIPS.

As variáveis, filas e ponteiros, dos caminhos de recepção e de transmissão estão declaradas no arquivo `include/handlers.s`. O código no arquivo com `main()` deve declarar estas variáveis com atributo **`extern`**. O vetor `_uart_buff` é o espaço reservado para salvar os registradores alterados pelo tratador da interrupção, e só é visível ao *handler* e não ao *driver*.

O Programa 1 mostra o leiaute da estrutura de dados na memória, como declarado em `include/handlers.s`.

Programa 1: Alocação da estrutura de dados para o *driver* da UART (versão *assembly*).

```
        .global Ud
Ud:
rx_hd:  .space 4        # index to queue head
rx_tl:  .space 4        # index to queue tail
rx_q:   .space 16       # reception queue
tx_hd:  .space 4        # index to queue head
tx_tl:  .space 4        # index to queue tail
tx_q:   .space 16       # transmission queue
nrx:    .space 4        # characters in RX_queue
ntx:    .space 4        # spaces left in TX_queue

_uart_buff: .space 16*4 # up to 16 registers to be saved here
```

Com base no Programa 1, a estrutura de dados que deve ser ser usada no *driver* é mostrada no Programa 2 e está declarada em `include/uart_defs.h`. Note que há uma correspondência exata entre `Ud` e `UARTdriver` – estas são duas versões, ou dois pontos de vista, sobre exatamente a mesma estrutura de dados.

Programa 2: Definição da estrutura de dados para *driver* da UART (versão C).

```
typedef struct UARTdriver {
   int      rx_hd;     // reception queue head index
   int      rx_tl;     // reception queue tail index
   char     rx_q[16];  // reception queue
   int      tx_hd;     // transmission queue head index
   int      tx_tl;     // transmission queue tail index
   char     tx_q[16];  // transmission queue
   int      nrx;       // number of characters in rx_queue
   int      ntx;       // number of spaces in tx_queue
} UARTdriver;

extern UARTdriver Ud;
```

O compilador e demais ferramentas que geram código para o MIPS estão em `/home/soft/linux/mips/cross/bin` .

Este caminho é acrescentado ao seu `PATH` automagicamente em `compile.sh` e em `assemble.sh`.

Se for conveniente instalar as ferramentas de compilação em seu computador pessoal, siga as instruções em `cMIPS/docs/installCrosscompiler`. A instalação demora cerca de uma hora, mais o tempo para fazer o *download* dos 5 pacotes necessários.

Da separação entre *driver* e `main()` O programa `main()` não pode acessar nenhuma variável ou estrutura de dados do *driver*. As únicas interações permitidas ocorrem através das funções da 'biblioteca', tais como `initUART()`, `ioctl()`, `Putc()`, etc. Qualquer desvio desta regra incorrerá em descontos significativos na atribuição de nota.

Especificação:

1. O trabalho pode ser efetuado em duplas;
2. o arquivo com os produtos deve ser nomeado `xx-yy.tgz` sendo `xx` e `yy` os *usernames* dos membros da dupla. Os arquivos relevantes devem estar abaixo do diretório `xx-yy`;
3. o arquivo com os produtos deve ser enviado por e-mail para `rhexsel@gmail.com` e conter somente os arquivos fonte com `main()` e `handlerUART.s`. Projetos com arquivos faltando e que não possam ser testados receberão nota zero. Todos os programas serão recompilados antes de simulados na avaliação, e os arquivos de teste serão alterados para a verificação do trabalho;
4. PLÁGIO NÃO SERÁ TOLERADO. É interessante que os alunos conversem sobre o projeto mas cada grupo deve escrever seu próprio código.

Produtos:

1. Relatório, com letras em 11 pontos, espaço simples, formatação simples, com os nomes dos componentes do grupo, e o código do tratador de interrupções em *assembly*.
2. o código novo em C e *assembly* do programa de comunicação deve ser entregue no *tarball*.
3. presença dos membros do grupo na data e hora marcadas para a apresentação.

Sugestões:

1. Assegure-se de que entendeu a especificação antes de iniciar o projeto do *driver*;
2. assegure-se de que entendeu as notas de aula sobre a UART antes de escrever qualquer linha de código;
3. escreva as funções de recepção (use o código em `include/handlers.s` como modelo); isso pronto, escreva as funções para a transmissão; isso pronto, escreva um programa de testes que faz eco e somente repete na saída a entrada; com isso funcionando corretamente, e só então, escreva o código das funções de conversão de cadeias para inteiros;
4. as simulações são demoradas, e por isso faça seus testes com o arquivo de entrada `serial.inp` contendo 3 ou 4 *strings*. As simulações podem facilmente gerar um arquivo `.vcd` com $\geqslant 1$ Gbytes e demorar mais do que 5 minutos no meu *desktop*.

Histórico das Revisões:

(31mai) adição ao FAQ de `data bus error`;
(22mai) separação entre *driver* e `main()`;
(21mai) primos em vetor estático ao invés de computados;
(16mai) atualização no diagrama do driver;
(14mai) publicação.

Referências

[RH12] *Sistemas Digitais e Microprocessadores*, R.A.Hexsel, 2012, Editora da UFPR.

[RH01] *Redes de Dados: Tecnologia e Programação*, R.A.Hexsel, 2001, Relatório Técnico do Depto. de Informática da UFPR, RT-DInf 005-2001, `http://www.inf.ufpr.br/roberto/rt005_2001.pdf`

FAQ

1) Posso mandar as mensagens de erro em uma foto?
Óbvia, definitiva e terminantemente, não.

2) Para que servem as mensagens de erro?
Servem para avisar de que algo está errado, e ademais, para apontar o quê está errado.

3) Para que serve o diagrama de tempos?
Serve para mostrar tudo o que está ocorendo com o sistema, no processador (topo) com eventos na escala de ciclos/instruções, e na interface serial (base) com eventos na escala de centenas de ciclos ou caracteres/mensagens.

4) Como descubro as opções de linha de comando do script `run.sh`?
`run.sh -h`

5) A simulação termina antes que a transmissão tenha completado. O que eu faço?
`run.sh -h`

6) Posso simular sem gerar `cMIPS.vcd`?
`run.sh -h`

7) Minha quota se esgotou. O que eu faço?
Remova `cMIPS.vcd`. Este arquivo é gigantesco. Apague todo o lixo que está armazenado em `~/Mailbox`, `~/Downloads`, `~/Videos`, `~/Music`.

8) O que significa a mensagem `addr error load (cause, epc, badAddr)`?
Seu programa executou um *load* desalinhado no endereço apontado no terceiro número da mensagem. Veja a definição de `UARTdriver`.

9) O que significa a mensagem `addr error store (cause, epc, badAddr)`?
Seu programa executou um *store* desalinhado no endereço apontado no terceiro número da mensagem. Veja a definição de `UARTdriver`.

10) O que significa a mensagem `data bus error (cause, epc, badAddr)`?
Seu programa executou uma referência (*load* ou *store*) num endereço no qual não existe nem RAM e nem periféricos. Os endereços base e os tamanhos das áreas de RAM e E/S estão próximos do topo do arquivo `vhdl/packageMemory.vhd`.

11) O que significa a mensagem `exception (cause, epc, badAddr)`?
Veja o primeiro número da mensagem, que é o conteúdo do registrador Cause no momento da excessão, e então a Tabela 7.1 nas notas de aula. Uma vez conhecida a causa da excessão, você consegue 'detetivar' a causa da causa da excessão.

12) Meu programa só funciona de vez em quando. O que eu faço?

(a) tente a saída honrosa: *Seppuku*;
(b) falhando a sugestão anterior, verifique se algum registrador que é usado no *handler* não foi salvo;
(c) falhando a sugestão anterior, verifique se algum registrador é usado no *handler* em duas funções diferentes mas o valor não é atualizado 'entre' a troca de função;
(d) falhando a sugestão anterior, releia a especificação;
(e) falhando a sugestão anterior, releia as notas de aula;
(f) falhando a sugestão anterior, reduza a velocidade de transmissão – use `SPEED` maior (≤ 5);
(g) falhando a sugestão anterior, inicialize **todos** os campos de `Ud`;
(h) falhando a sugestão anterior, repita o item (a) com uma espátula de pedreiro.

13) Para que serve o `EOT`?
Serve para a unidade remota avisar que todos os caracteres já foram enviados.

14) Posso alterar a velocidade de transmissão no meio do programa?
Não. Muito provavelmente o simulador deixará de funcionar a contento.

15) As interrupções funcionam e depois param. Por que?
Muito cuidado e atenção com o conteúdo de `uart.interr`. Os bits `uart.interr.intTX` e `uart.interr.intRX` só podem ser ligados na inícialização da UART e depois nunca mais modificados. Quando algum dos bits de `uart.interr` deve ser alterado, este registrador deve ser lido, o bit apropriado modificado (`ori` ou `andi`) e então o novo conteúdo – já com o bit trocado – deve ser escrito em `uart.interr`.

16) O que significa a mensagem de erro na compilação `Error: symbol 'xxxx' is already defined`?
Seu programa está incluindo algum arquivo de código ou de cabeçalho mais de uma vez.

17) Já tentei todas as sugestões acima e ainda não funciona. O que eu faço?

(a) Tente novamente *seppuku*, desta vez usando uma serra de cortar pão;

(b) compile seu programa sem a opção `-v` (verboso) e veja se há algum erro de compilação;

(c) preciso ver o `cMIPS.vcd`; compartilhe-o comigo no google-drive porque o arquivo é enorme. **Não envie uma foto da tela**.

18) Para que servem as notas de aula?
Sem estudar a parte teórica, é **impossível** completar o trabalho. Esta é a principal lição a tirar desta disciplina. Para quem olha na superfície, a teoria é completamente isolada do trabalho mas isso não é verdade. Uma das funções principais do trabalho é fazer com que vocês liguem a teoria com a prática. Outra é *aprender paciência*.

19) Para que servem as aulas de laboratório?
Servem para reduzir a derivada da curva de aprendizagem. Ignore-os e retorne no próximo semestre.

20) Para que serve esta disciplina? É uma gincana? É uma corrida de obstáculos?
Não serve para nada; servirá para uma vida profissional de 30 anos. Não. Não.

21) O trabalho cai na prova?
Sim.

22) O que eu faço após terminar o trabalho?
Assista aos filmes de Akira Kurosawa, nesta ordem: Sete Samurais, Kagemusha, Ran, Dersu Uzala (meu predileto), Yume q (a tradução em Português é 'Sonhos'). Assista qualquer outro Kurosawa fora desta lista. Isso feito, ignore a sugestão sobre *seppuku*.