µProcessador 4 "Calculadora Programável"

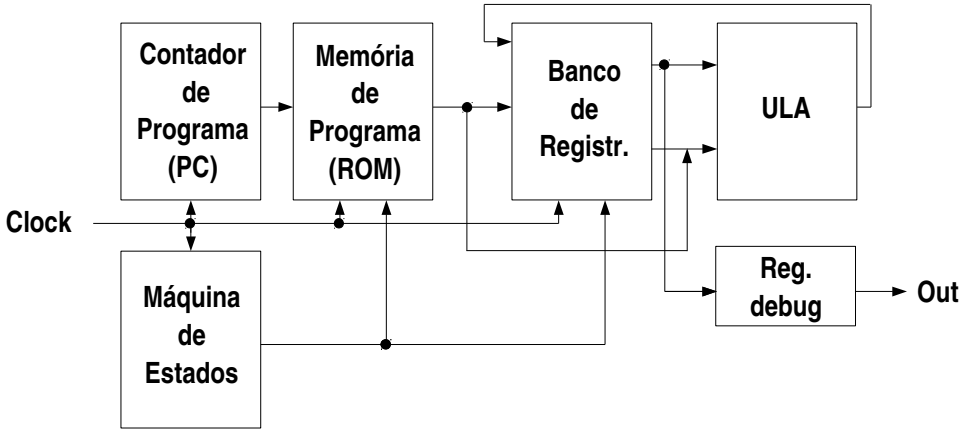
Partindo da ULA-Banco de Registradores dos laboratórios anteriores, adicionar uma memória de programa e executar uma lista de instruções aritméticas.

Deve obrigatoriamente ser multiciclo.

Implemente apenas instruções da ULA, de carga de constantes e transferência de valores entre registradores: outras instruções ficam para depois.

Sugestão de Circuito (apenas sugestão)

Observe que o bloco "Reg. debug" abaixo é um registrador à parte, para debug e avaliação, e suas saídas devem estar presentes no *top level*. Uma instrução *assembly* especial "peek $r" (ou "out R", tanto faz), sendo R a especificação de um registrador, deverá ser implementada para seu uso.



Perceba que os únicos pinos obrigatórios no *top-level* do projeto são Clock e Out, mas quaisquer outros adicionais podem ser incluídos. Em especial, o valor do PC e a saída da ROM (é o opcode da instrução) são valiosos para debug.

Codificação das Instruções

Uma codificação mínima deve ser usada neste estágio. É permitido mudar os formatos de instrução em laboratórios posteriores.

Para pensar sobre essa codificação, leia a seção "RISC vs. CISC" mais à frente. Pode usar algo similar aos formatos R e I do MIPS, mas há várias outras opções, de acordo com a escolha do *set* de instruções, incluindo usar instruções que ocupam dois ou mais de um endereço de memória (dê uma olhada no que a Intel faz).

Note particularmente que o seu *set* de instruções não precisa ser ortogonal e note também que é possível usar um registrador com constante zero, como o $zero, ou não usar isso. Também podem ser implementadas instruções específicas para para carga e movimentação de dados (*li* e *move*) sem passar pela ULA, se desejar.

É permitido usar memórias com *bus* de dados de largura diferente de 8 ou 16 bits, embora isso seja esquisito.

▶	Crie uma codificação de instruções para o seu processador. Anote-a num papel, arquivo texto ou planilha, explicitando os campos. Para cada programa que você for implementar, liste as instruções indicando os endereços de memória e os códigos de máquina *em hexadecimal ou binário*. Decimal é para os fracos.

Memória de Programa

Adicione ao circuito uma memória, um Contador de Programa (PC) explícito e uma unidade de controle mínima (como a das primeiras versões vistas do MIPS), que decodifica as instruções escolhidas e aciona os circuitos do Banco de Registradores e ULA de forma adequada. Neste estágio é fácil obter, decodificar e executar uma instrução usando duas rampas de clock.

Para colocar os programas de teste dentro da memória de programa é possível editar diretamente, salvando/carregando os dados no formato .mif (*memory initialization file*, da própria Altera). Para criar um arquivo .mif, vá em File->New->Memory Files; o conteúdo pode ser editado dentro do próprio Quartus. Duas opções de componentes de memória são a ROM: 1_PORT[1] do MegaWizard e a LPM_ROM.

Muito Importante: quando você for mudar o arquivo .mif, ele *não é atualizado automaticamente!* Para recarregá-lo, vá em *Processing => Update Memory Initialization File*

Multiciclo

Faça uma máquina de estados simples com dois ou três estados, sendo o primeiro o *fetch* da instrução. No *fetch*, o endereço a acessar (vindo do PC) é alimentado à memória de programa. A instrução lida apenas estará disponível à saída da ROM no clock seguinte.

Cuidado com máquinas assíncronas pois os estados transientes[2] podem bugar o circuito caso a decodificação das instruções ou o reset do contador sejam assíncronos. Sugiro fazer máquinas *síncronas,* ou seja, aquelas em que os flip-flops estão todos ligados a um clock comum (ao invés de estarem cascateados). Revise rapidamente eletrônica digital.

Nota: Quando se tenta fazer ciclo único (é possível), o circuito fatalmente acaba virando uma árvore de gambiarras em algum momento. Portanto estão vetados os processadores ciclo único.

Dicas e Comentários

- *Teste direito essa coisa!* O professor fará testes adicionais com o projeto, além do entregue.
- Não esqueça de incluir um pino de RESET no PC e na máquina de estados, que zera todos os flip-flops quando ativado. Todos os RESETs devem estar ligados ao mesmo pino de RESET global no *top-level.*
- Note que as memórias numa FPGA são *síncronas* e portanto a ROM requer um clock (!)
- Recomenda-se usar sempre clocks sensíveis a borda de subida para as operações.
- Se for preciso trocar o nome de um sinal, utilize o componente WIRE (é uma porta não inversora, como um buffer simples).
- Verifique sempre se as operações do circuito conseguem ser executadas *dentro de um ciclo de clock* (ou seja, o tempo de propagação pelas portas não estoura o período).
- Infelizmente só podemos debugar os sinais que estão no *top level*. Portanto, se houver necessidade de checar um sinal de *dentro* de um bloco, deve-se criar um pino no bloco para este sinal e ligá-lo em uma saída no *top level*, expondo-o para o *Waveform Simulator*.
- Se desejar, construa um montador simples que produz uma listagem hexa a partir de um programa assembly.
- Na próxima tarefa, o programa deverá começar a executar a partir do endereço zero. Isso não é necessário aqui.

▶	Determine aproximadamente a frequência máxima de operação do seu ciruito.

1 Para usar a ROM: 1-PORT, é necessário haver criado previamente um arquivo tipo .mif para inicialização. Também é preciso desmarcar a opção "q output port" na página 4 de 7 na criação, evitando que a saída seja latcheada.

2 Os contadores assíncronos geram transitórios durante a mudança de estado, produzindo valores de estado indesejáveis por um curto período, que podem prejudicar (muito) o circuito se não tratados adequadamente.

Entrega eletrônica (em beta até a copa)

Esta prática costuma gerar uns probleminhas simples mas enjoados de resolver. Como de hábito, ficamos para daqui a duas semanas, até 6a feira às 13h50.

O arquivo "**projeto.txt**" deverá conter linhas adicionais definindo o nome do sinal de clock e do pino de saída do registrador de debug:

```
TOP_LEVEL_ENTITY = banco # atualizar!
...                      # definições anteriores
DATA_REG2 = R2_Out       # última definição do lab uProc #3
CLOCK = Clock            # nome do sinal de clock
REG_DEBUG = Out          # nome dos pinos do registrador de debug
```

Escolha um dos seus registradores para ser o acumulador designado (A). Construa um programa que calcula a expressão `Reg debug <= ((3+4)-(5+2)) op 5`, onde "op" é a operação da ULA dada pelo professor à equipe. Grave este código na ROM usando um arquivo obrigatoriamente chamado "**programa.mif**".

Este programa deverá estar listado claramente, em papel ou num arquivo, com cada linha contendo o endereço de memória, o *opcode* em hexadecimal e os mnemônicos da instrução. Isto deve ser mostrado ao professor juntamente com a descrição (tabela, talvez) da codificação de instruções, que *tem que estar compreensível!*

Finalmente, especifique num arquivo "**opers.txt**" os seguintes trechos de opcodes hexadecimais para o seu processador *na ordem em que estão listados.* Use um opcode por linha e faça cada trecho ser separado por uma linha em branco:

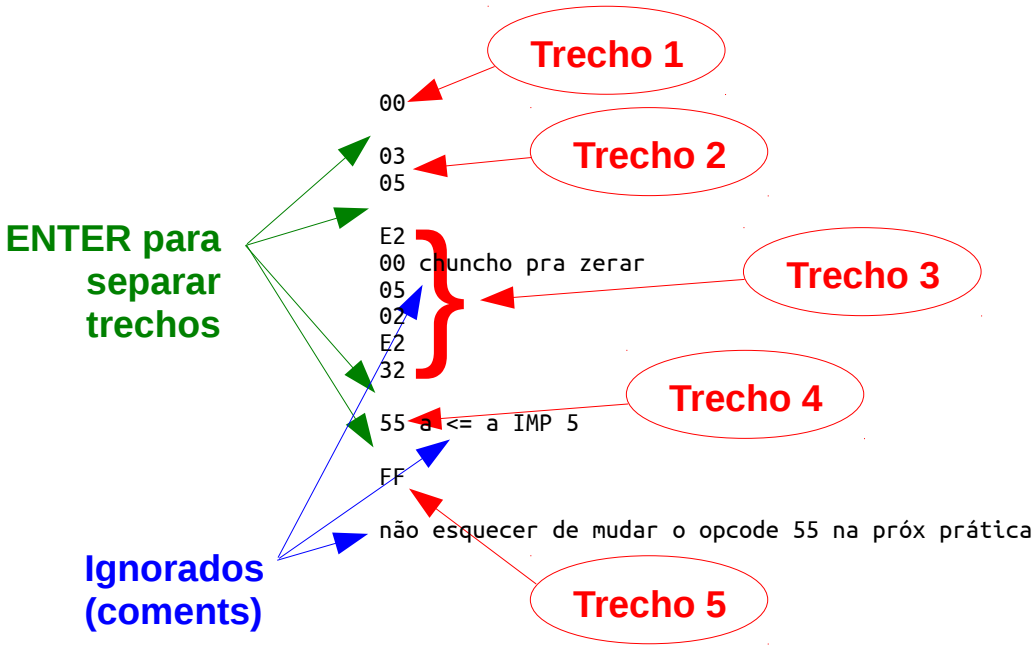
Trecho 1: opcode do nop
Trecho 2: opcodes que realizam A <= 3+5
Trecho 3: opcodes para o cálculo de A <= A-(5+2)
Trecho 4: lista de opcodes de A <= A op 5
Trecho 5: opcode que faz Reg debug <= A

Onde "nop" é uma instrução que não faz nada, "op" é a operação que o professor atribuiu à equipe e "out" é o registrador de saída de debug. Por exemplo, um arquivo poderia ser:

```
00

03
05

E2
00 chuncho pra zerar
05
02
E2
32

55 a <= a IMP 5

FF

não esquecer de mudar o opcode 55 na próx prática
```

Pode colocar comentários pra si mesmo após o opcode desde que deixe um espaço entre eles, e comentários após o final do código, deixando uma linha em branco depois do último opcode.

E agora uma figurinha colorida mais descritiva do mesmo arquivo:



Checklist de Entrega

Circuitaria:

- ✔ Não há pinos de entrada deixados em aberto (i.e., desconectados)
- ✔ Todos os flip-flops/contadores possuem pino de RESET
- ✔ Não há acentuação ou espaços nos componentes
- ✔ Não há acentuação ou espaços nos nomes dos arquivos nem no caminho das pastas

Arquivos anexados no email:

- ✔ O arquivo de projeto compactado **.qar** *atualizado*
- ✔ O **projeto.txt** com as configurações *atualizadas*
- ✔ O **opers.txt** como descrito anteriormente
- ✔ Os códigos de ROM para realizar o cálculo pedido, **programa.mif**
- ✔ Os testes **.vwf** que vocês utilizaram, para me ajudar caso preciso
- ✔ Os outros arquivos **.mif** com os programas testados nos .vwf

Apêndice: RISC vs. CISC

RISC (computador com conjunto reduzido de instruções) e CISC (computador com conjunto complexo de instruções) são os dois paradigmas básicos de ISAs. Atualmente, o modelo RISC é mais usado, embora os processadores nunca sigam exatamente o paradigma. Os CISCs são considerados ultrapassados e encontrados em microcontroladores (8051, 68HCxx) ou mainframes (o IBM z10) e modelam a ISA dos Intel Core. Já os RISCs vão de microcontroladores (AVR e PIC de 8 a 32 bits) a processadores de alto desempenho (ARM e MIPS), além de serem usados internamente nos Intel. Outro paradigma, pouco usado, é o VLIW (DSPs da Texas e Itanium).

A tabela a seguir resume as características típicas.

	RISC	CISC
Número de instruções	Pequeno (menor do que 100)	Grande (mais do que 400)
Desempenho	Mais rápido e otimizável	Difícil de otimizar e programar
Operandos	Instruções ortogonais*	Uso específico de registradores
Acesso à memória	Modelo Load/Store*	Endereçamento flexível
Formato de instruções	Bastante regular	Muito irregular
Tipos de dados	Apenas muito básicos	Estruturas mais complexas

- * *Ortogonalidade do Set de Instruções:* significa que as instruções podem acessar qualquer registrador disponível. O *add* do MIPS pode usar qualquer um dos 32 registradores em qualquer dos três operandos. Em contraste, no 8051 a soma é obrigatoriamente feita com o registrador A (ou Acc, o acumulador) como uma das fontes e como o único destino possível. No Z80, os endereços dos periféricos têm que estar no registrador C.
- Há a distinção de registradores de uso genérico (nos RISC) e registradores de uso específico (nos CISC).
- * *Modelo Load/Store:* significa que as instruções de acesso à memória apenas fazem a leitura (*Load*) de dados ou a escrita *(Store),* sem processamento adicional nenhum. A única forma de acessar a memória é com instruções *lw* e *sw* ou similares. Para os CISCs isso é diferente: a instrução *sub ax,[bx+1234]* da Intel vai ler o dado da posição de memória bx+1234 e subtraí-lo do acumulador *ax*, armazenando o resultado em *ax*.
- *Modos de endereçamento:* classificam a forma como os dados são acessados em um processador. No MIPS, para dados, temos o modo registrador (*add $4,$4,$4*), o modo imediato (*addi $7,$0,43*) e o modo indireto indexado (*lw $9,1024($8)*). No 8051, fazemos *mov a,50* (endereçamento direto: lê do endereço 50 para o acumulador). Na ISA Intel, temos uma instrução como *mov ax,[bx+di+10]* (indireto, "base-índice com deslocamento") e outra como *add ax, es:[bx+40]* (indireto com índice e segmento: *es* identifica outra área de memória).
- Na ISA x86 Intel, as instruções podem ter de um a 18 bytes de tamanho, um formato bastante irregular para decodificar.

Note que a arquitetura ARM (que compreende múltiplas versões e processadores bastante diferentes entre si, mas com um núcleo comum na ISA) se vende como RISC, embora tenha várias características CISC, em especial o acesso à memória com modos de endereçamento bastante complexos. Isso não impede o uso de *pipelining* e alto desempenho: apenas dificulta para o projetista.