O trabalho II compreenderá um arquitetura RISC implementada em FPGA. A CPU em questão será baseada nas CPUs MIPS. No entanto, não utilizará o mesmo *instruction set*. Este será modificado de acordo, para cada grupo.

OBRIGATORIAMENTE, UTILIZAR FPGA DA FAMÍLIA CYCLONE IV GX (qualquer uma da família que caiba o circuito, aconselho o modelo com o maior número de pinos da família).
OBRIGATORIAMENTE, SIMULAR EM GATE LEVEL.

Características Principais:
- A Word da arquitetura é definida em 32 bits ***Big Endian***;
- Todo o sistema é implementado em pipeline;
- Todas as instruções são formadas por 4 bytes;
- São 32 registros (r0 a r31) e o r0 é ***hard-wired*** em 0;
- A memória de programa tem 1kWord alocados a partir de ***Número do grupo * 120h*** **(**o módulo ***ADDRDecoding_Prog*** deverá fazer a decodificação de endereços apropriada para selecionar, apropriadamente, instruções da memória de programa interna ou externa);
- A memória de dados tem 1kWord alocados a partir de ***Número do grupo * 125h*** (o módulo ***ADDRDecoding*** deverá fazer a decodificação de endereços apropriada para selecionar, apropriadamente, dados da memória de dados interna ou externa)
- A cada *Reset*, o *Program Counter* sempre aponta para o endereço inicial **do programa**;

A figura 1a abaixo representa a arquitetura requerida sem considerar que nas FPGAs da Altera, as BRAM funcionam apenas de forma síncrona, tanto para leitura quanto para a escrita. De qualquer forma, é apresentada para entendimento geral do pipeline requisitado.

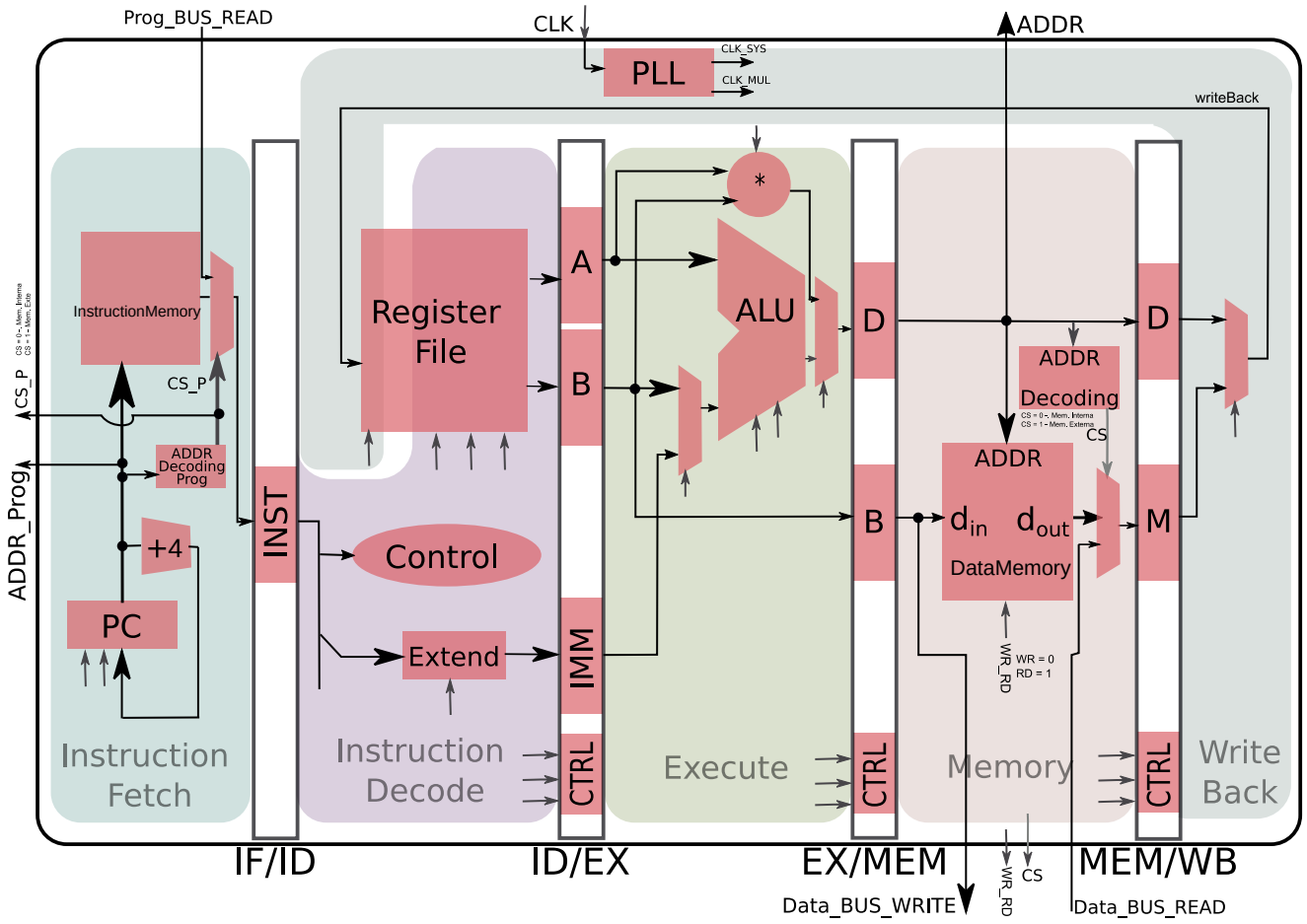

Fig. 1a – Arquitetura RISC Padrão de 5 Estágios

Já a figura 1b a seguir apresenta o pipeline requerido já ajustado para as características das FPGAs Altera (note que alguns registros de estágio foram removidos por já estarem presentes internamente nas memórias BRAM). Também são apresentados em verde os sinais necessários para os controles das instruções de desvio de fluxo. Essa figura apresenta exatamente o *pipeline* requisitado pelo trabalho e deverá ser seguida como base para sua implementação.

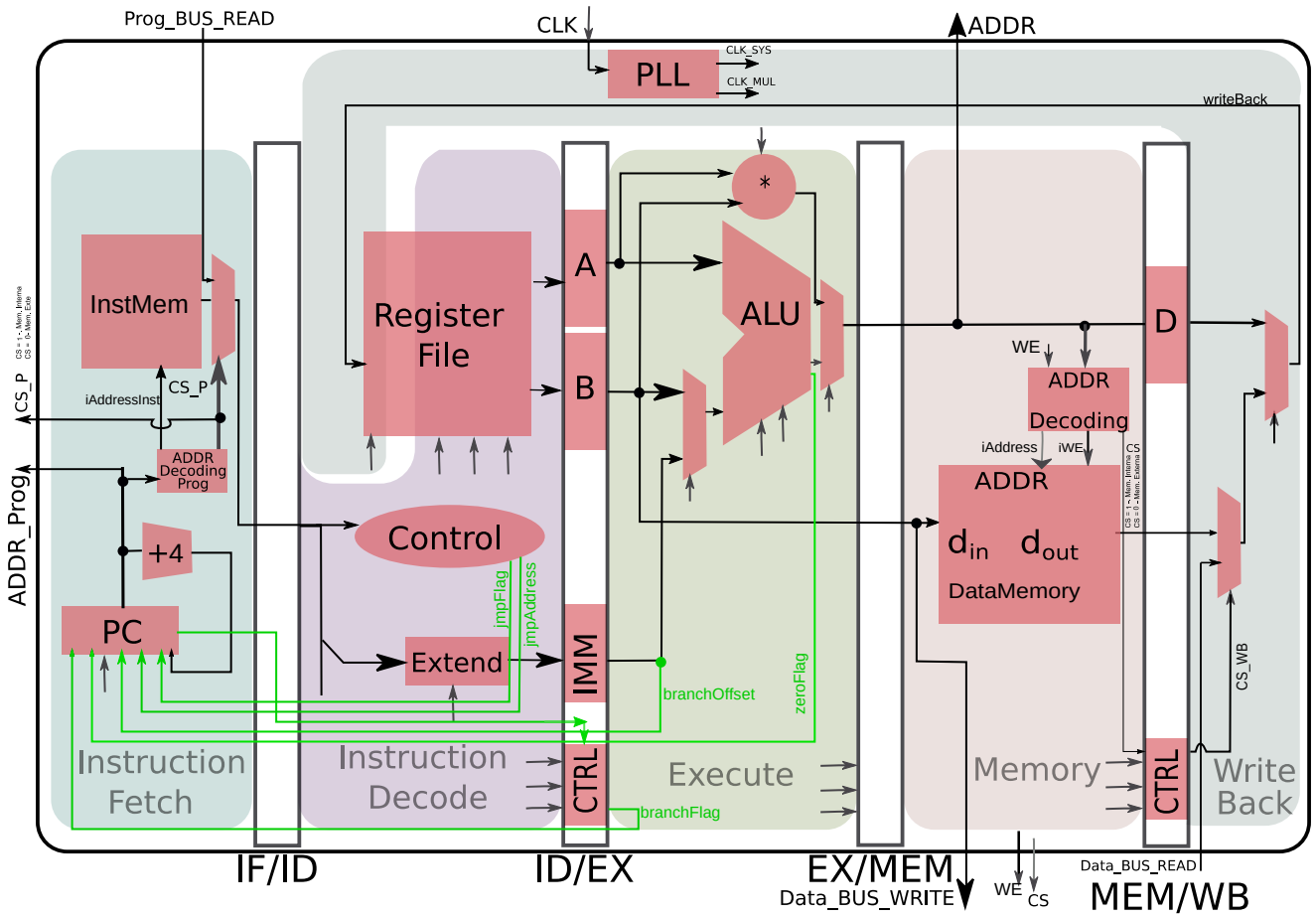

Fig. 1b – Arquitetura MIPS a ser Implementada já Modificada para Memórias e Banco de Registradores Síncronos. As linhas em verde representam sinais de controle para Instruções de BNE e JMP. Outras linhas de controle foram suprimidas para não poluir o desenho. Fica a cargo do aluno identificar e implementar essas linhas

O projeto deverá ser feito utilizando hierarquia, ou seja, cada módulo deve ser feito e testado separadamente com *TestBench* apropriado. A adoção de abordagem ascendente ou descendente fica a cargo do grupo. Uma descrição estrutural deverá conectar todos os módulos, formando a **MIPS_CPU** (hierarquia *top*), que também deverá ser testada. Nomes de módulos devem ser rigorosamente seguidos de acordo com a seção de *deliverables* deste roteiro.

Segue abaixo a definição do ISA proposto:

rs	Primeiro registrador Fonte
rt	Segundo registrador Fonte para instruções tipo R Ou Registro destino para instruções tipo I
rd	Registrador destino para Instruções tipo R

Tab. 1 – Legenda

Tamanho em Bits			**6**	**5**	**5**	**16**	
Load Word	LW	I	(Grupo+32)	rs	rt	offset	R[rt]=M[R[rs]+SignExtImm]
Store Word	SW	I	(Grupo+33)	rs	rt	offset	M[R[rs]+SignExtImm]=R[rt]
Branch on Not Equal	BNE	I	(Grupo+34)	rs	rt	offset	if(R[rs] != R[rt]) PC = PC + 4 + offset
Add Immediate	ADDI	I	(Grupo+35)	rs	rt	offset	R[rt]=R[rs]+SignExtImm
Or Immediate	ORI	I	(Grupo+36)	rs	rt	offset	R[rt]=R[rs] | SignExtImm

	Mnemonic	**Format**	**calc**						
Tamanho em Bits			**6**	**5**	**5**	**5**	**5**	**6**	
Add	ADD	R	Grupo+10	rs	rt	rd	10	32	R[rd]=R[rs]+R[rt]
Subtract	SUB	R	Grupo+10	rs	rt	rd	10	34	R[rd]=R[rs]-R[rt]
Multiplication	MUL	R	Grupo+10	rs	rt	rd	10	50	R[rd] = lowerHW_R[rs] * lowerHW_R[rt]
And	AND	R	Grupo+10	rs	rt	rd	10	36	R[rd]=R[rs]&R[rt]
Or	OR	R	Grupo+10	rs	rt	rd	10	37	R[rd]=R[rs]|R[rt]

Instruction name	**Mnemonic**	**Format**	**Encoding (10)**		
Tamanho em Bits			6	26	
Jump	JMP	J	2	JumpADDR	PC = JumpADDR

Tabela 2 – MIPS Instruction Set (ISA) Modificado

O Programa de teste da CPU deverá executar a seguinte expressão matemática, salvando o resultado na ***última posição da memória de dados interna***:

$$MemDados[\text{última posição}] \leftarrow (\sum_{i=0}^{n-1} Mem[i + \text{Número do grupo} * 0125\,h]) * 255(d) \quad ,$$

onde n = 32(d) e as n primeiras posições de memória devem ser preenchidas com 0, 1, 2, 3, … 31(d).
Em *Assembly*, o código fica:

```
                                        ;   Code presents data hazzard
Main:   ori $r30, 32($r0)               ;   r30 ← 32
        ori $r10, 0($r0)                ;   r10 ← 0
        ori $r31, 0($r0)                ;    r31 ← 0
Loop:   lw $r1, grupo*0x0125($r31)      ;    r1 ← Mem[i + Número do Grupo * 0250h]
        addi $r31, 1($r31)              ;    i = i + 1
        add $r10, $r1, $r10             ;    r10 ← r1 + r10
        bne $r31, $r30, Loop            ;    if r31 != r30 then PC ← Loop Address
        sw $r10, grupo*0x0125+1023($r15);    MemDados[ultima posição RAM] ←  [r10]
                                        ;
                                        ;    Code solves data hazzard by adding bubbles
        ori $r31, 0($r0)                ;    r31 ← 0
        ori $r10, 0($r0)                ;    r10 ← 0
        ori $r13, 0x00FF($r0)           ;    r13 ← 0x00FF
Loop1:  lw $r1, grupo*0x0125($r31)      ;    r1 ← Mem[i + Número do Grupo * 0250h]
        addi $r31, 1($r31)              ;    i = i + 1
                                        ;    NOP as Bubble
                                        ;    NOP as Bubble
        add $r10, $r1, $r10             ;    r10 ← r1 + r10
                                        ;     NOP as Bubble
                                        ;     NOP as Bubble
        bne $r31, $r30, Loop1           ;    if r31 != r30 then PC ← Loop Address
        mul $r20, $r10, $r13            ;    r20 ← sum * 0x00FF
                                        ;
                                        ;
        sw $r20, grupo*0x0125+1023($r15);    MemDados[ultima posição RAM] ←  [r10]
                                        ;
        jmp Main                        ;    PC ← Main Address
```

Obs.:

1. Gere o código binário a ser salvo na memória de programa (utilize como base a Tabela 2 ou o assembler no GitHub https://github.com/odutra00/assemblerRISC).
2. Linhas vazias são interpretadas pelo Assembler como uma instrução NOP, gerando uma instrução de 4 bytes zerados. O módulo ***control*** deverá gerar os sinais de controle apropriados para esse opcode);
3. A instrução de multiplicação operará em 16 bits, ou seja, os operandos correspondem aos 16 bits menos significativos do conteúdo de ***rs*** e ***rt***. Dessa forma, o resultado será de 32 bits, a ser armazenado em ***rd***. Obrigatoriamente, o hardware responsável pela multiplicação será o abordado no laboratório 4, modificado para operandos de 16 bits. Ele deverá, OBRIGATORIAMENTE, funcionar com um clock diferente do clock do sistema (CLK_SYS). Mesmo com a adição do multiplicador, o sistema ainda deverá operar com *throughput* de 1 instrução/clk;
4. O clock do sistema (***CLK_SYS***) e clock do multiplicador (***CLK_MUL***) devem ser saídas do bloco de propriedade intelectual (IP) ALTPLL.

Avaliação (responda em forma de comentários no início do módulo *top* MIPS_CPU):
Após a implementação e verificação do correto funcionamento do circuito, responda (respostas dentro do módulo MIPS_CPU como comentários):
a) Qual a latência do sistema?
b) Qual o *throughput* do sistema?
c) Qual a máxima frequência operacional entregue pelo *Time Quest Timing Analizer* para o multiplicador e para o sistema? (Indique a FPGA utilizada)
d) Qual a máxima frequência de operação do sistema? (Indique a FPGA utilizada)
e) Analisando a sua implementação de dois domínios de clock diferentes, haverá problemas com metaestabilidade? Por que?
f) A aplicação de um multiplicador do tipo utilizado, no sistema MIPS sugerido, é eficiente em termos de velocidade? Por que?
g) Cite **modificações cabíveis na arquitetura do sistema** que tornaria o sistema mais rápido (frequência de operação maior). Para cada modificação sugerida, qual a nova latência e *throughput* do sistema?

Deliverables

O que entregar? Arquivo zip com a estrutura hierárquica do projeto como a seguir (Apenas os arquivos indicados. Caso contrário, o arquivo zip ficará muito grande e não será possível enviá-lo pelo SIGAA):
(arquivos qws, qpf e qsf são gerados automaticamente ao se criar o projeto)
IMPORTANTE: Antes de enviar o arquivo compactado, extraia-o em outra pasta, resimule-o e garanta que o arquivo contenha a hierarquia funcional e pedida. NÃO IREI CORRIGIR TRABALHOS QUE NÃO ATENDAM O QUE ESTÁ SENDO PEDIDO.
TestBench mostrando saídas/entradas e sinais internos indicados por nomes na fig. 1b (*exatamente na forma como escrito na fig.1b – utilizar $init_signal_spy e (*keep=1*) para sinais internos*), rodando o programa anterior.

- → MIPS_CPU (pasta zipada)
  - → cpu.v (descrição estrutural ligando os módulos)
  - → TB.v (testbench da cpu)
  - → cpu.qws
  - → cpu.qpf
  - → cpu.qsf
  - → DataMemory
    - → datamemory.v
    - → datamemory_TB.v
    - → datamemory.qws
    - → datamemory.qpf
    - → datamemory.qsf
    - → Data.hex
  - → InstMem
    - → InstMem.v
    - → InstMem_TB.v
    - → InstMem.qws
    - → InstMem.qpf
    - → InstMem.qsf
    - → Code.hex
  - → MUX
    - → mux.v
    - → mux_TB.v
    - → mux.qws
    - → mux.qpf
    - → mux.qsf
  - → PC
    - → pc.v
    - → pc_TB.v
    - → pc.qws
    - → pc.qpf
    - → pc.qsf
  - → ALU
    - → alu.v
    - → alu_TB.v
    - → alu.qws
    - → alu.qpf
    - → alu.qsf

- Multiplicador
  - Multiplicador.v
  - Multiplicador_TB.v
  - Multiplicador.qws
  - Multiplicador.qpf
  - Multiplicador.qsf
    - **Adder**
      - Adder.qpf
      - Adder.qsf
      - Adder.v
      - Adder_TB.v
    - **CONTROL**
      - CONTROL.qpf
      - CONTROL.qsf
      - CONTROL.v
      - CONTROL_TB.v
    - **Counter**
      - Counter.qpf
      - Counter.qsf
      - Counter.v
      - Counter_TB.v
    - **ACC**
      - ACC.qpf
      - ACC.qsf
      - ACC.v
      - ACC_TB.v
- Control
  - control.v
  - control_TB.v
  - control.qws
  - control.qpf
  - control.qsf
- RegisterFile
  - registerfile.v
  - registerfile_TB.v
  - registerfile.qws
  - registerfile.qpf
  - registerfile.qsf
- Extend *(estende o offset de 16 bits para 32 bits nas instruções lw e sw).*
  - extend.v
  - extend_TB.v
  - extend.qws
  - extend.qpf
  - extend.qsf
- Register *(são os registros que separam cada estágio).*
  - register.v
  - register_TB.v
  - register.qws
  - register.qpf
  - register.qsf
- ADDRDecoding
  - ADDRDecoding.v
  - ADDRDecoding_TB.v
  - ADDRDecoding.qws
  - ADDRDecoding.qpf
  - ADDRDecoding.qsf
- ADDDecoding_Prog
  - ADDRDecoding_Prog.v
  - ADDRDecoding_Prog_TB.v
  - ADDRDecoding_Prog.qws
  - ADDRDecoding_Prog.qpf
  - ADDRDecoding_Prog.qsf
- PLL
  - Manter todos os arquivos da pasta raiz do projeto deletando apenas as pastas.

Dicas:

1) Todos os módulos que contiverem *Reset* devem *resetar* da mesma forma (parece brincadeira mas, quando vocês dividem as tarefas e cada um faz um módulo, acaba cada um fazendo de um jeito). E lembre-se, o tipo de *reset* tem suas implicações. É possível fazer funcionar de qualquer forma sem que haja interferência na arquitetura proposta.

2) Tenha certeza que o programa gravado na BRAM que funciona como *InstructionMemory* contém o programa codificado apropriadamente.

3) Tenha certeza que seu sistema funciona no *Testbench* rodado em *GateLevel*. Para isso, a estimativa de máxima frequência de operação deve estar correta e deve ser respeitada na definição do *clock* no *TestBench*. Além disso, o clock definido no testbench deve estar de acordo com o clock definido como entrada de clock na definição da PLL.

4) Tenha certeza que todos os sinais monitorados no *Testbench* são os pedidos no escopo do trabalho, inclusive respeitando o *case* dos caracteres.

5) A depuração deve sempre ser feita do fim para o início. Por exemplo, entendendo como uma instrução de ***Load Word*** funciona, no ciclo de ***Write Back***, a linha de ***writeBack*** deveria conter o dado presente na posição de memória lida pela instrução ***LW***. Se a mesma não apresentar o valor, pergunte-se o que pode estar errado. Perceba que se qualquer um dos sinais de controle estiver errado, de qualquer estágio anterior, acarretará o erro na linha de ***writeBack***. Vá monitorando os sinais, sempre do fim para o início até visualizar o que está errado. Vá consertando o que está errado e repetindo o procedimento.

6) O clock gerado no testbench para ser aplicado na PLL deve ser o mesmo clock definido como entrada na definição deste bloco no IP Catalog. Caso contrário, para diferenças grandes, a PLL pode não gerar os clocks CLK_SYS e CLK_MUL.

FAQ

1 Tamanho do .zip maior que 10MB

1.a Por favor, siga exatamente as instruções do guia do trabalho. A estruturas das pastas, com os arquivos que devem estar nelas, está definida no roteiro. Todo o restante deve ser deletado. Seguindo isso, vai caber com folga em 10Mb.

2 Dúvida simulação em *Gate Level*: Estamos finalizando o projeto e surgiu uma dúvida referente a simulação em Gate Level, já que na aula do Lab08 vi que essa simulação é feita quando não é possível usar o *timing* por ele não reconhecer o tempo dos multiplicadores, a simulação em gate level seria apenas dos blocos que não possuem registradores?

Aconselho a fazer/assistir a aula sobre *timing* e o lab05.

A simulação *Gate Level* deverá ser feita sempre que disponível pois a mesma considera os atrasos dos circuitos (*Gates*) e por isso se aproxima muito da realidade. A simulação RTL não se aproxima em nada da realidade, nunca poderá ser considerada como parâmetro para funcionamento no mundo real. A simulação RTL é apenas prova de conceito já que a mesma não considera os atrasos intrínsecos do hardware (*Gates*) e, por isso, é chamada de RTL (*register transfer level*). Ou seja, ela não considera o *timing* (que nós estudamos). Se quiser fazer um teste, qualquer frequência de *clock* que você colocar no *testbench* funcionará a simulação em RTL.

No trabalho eu pedi para vocês utilizarem como base a FPGA Cyclone IV GX (qualquer uma da família que caiba o circuito) pois a mesma, por se tratar de uma FPGA mais antiga, tem disponibilizado os arquivos de tecnologia que tornam possível a simulação em *Gate Level*. Com a família MAX10 isso não é possível pois não existem esses arquivos.

Além disso, em simulações RTL, para vários modelos de FPGA, dados inicializados em memória são desprezados, tornando impossível a simulação do processador já que, tanto o programa quanto os dados estarão inicializados nas memórias de programa e de dados, respectivamente.

Nada a ver com o descrito acima é o TimeQuest Timing Analyser. Ele funcionará sempre, e deverá ser feito sempre para determinação dos parâmetros de operação do circuito (frequência de *clock* , etc).

3 Duvida entre questões c) e d): Professor, a diferença entre as perguntas relativas à máxima frequência operacional e a máxima frequência de operação seria com relação à que no primeiro seria considerado as piores condições de operação do ambiente (alta temperatura) e a segunda seria considerado as condições ótimas do ambiente (baixas temperaturas) ? Ou seria o contrário?

Quando você fizer a análise no TimeQuest, o que será reportado é a máxima frequência, baseada no timing do caminho crítico (lembrando que um caminho é SEMPRE referenciado da ENTRADA DE UM FLIP-FLOP À ENTRADA DO PRÓXIMO FLIP-FLOP). Essa análise será feita para o multiplicador e para o sistema (MIPS) e os valores encontrados serão as respostas ao item c.

No entanto, quando vocês acoplarem o multiplicador ao sistema (MIPS), as coisas mudam de figura. O timeQuest não sabe que a arquitetura proposta compreende o multiplicador (que necessita de vários pulsos de clock para entregar o resultado - latência grande) dentro do estágio de execução do sistema (MIPS). Assim sendo, um período de clock do sistema (MIPS) deve ser capaz de acomodar diversos períodos de clock do multiplicador já que o multiplicador deve receber os operandos, processá-los e entregar a produto dentro do estágio de execução do MIPS. Assim, basicamente, a máxima frequência de operação do sistema MIPS total, basicamente, será a máxima frequência reportada pelo TimeQuest para o multiplicador dividido pela quantidade de pulsos de clock necessária para que o multiplicador processe os operando e entregue o produto (resposta a letra d).

Exemplo: Digamos que no item c você encontrou 340MHz para máxima frequência operacional do multiplicador e 80MHz para máxima frequência operacional do sistema (MIPS) sem o multiplicador.

Se você interligar o sistema total, com estas frequências, o mesmo não funcionará para a instrução da multiplicação pois a freq. do multiplicador é apenas 4,25x maior que a do sistema sem ele. E seriam necessários, baseado no trabalho do multiplicador, 2N+2 (onde N é o número de bits do multiplicando ou do multiplicador, considerando que ambos tenham o mesmo tamanho em bits). Assim, 2N+2 = 2*16+2 = 34. A frequência de clock do multiplicador deve ser 34x maior que a frequência de clock do sistema para que o estágio de execução acomode toda a latência do multiplicador. Como

a máxima frequência reportada pra o multiplicador pelo timeQuest é 340MHz, se fizermos 80MHz (máxima freq. reportada para o sistema sem o multiplicador) * 34 = 2720MHz. Ou seja, não podemos fazer assim, pois ultrapassamos a máxima frequencia reportada pelo timeQuest para o multiplicador.
Dessa forma, o que temos que fazer é utilizar como frequência do MIPS:
freq. Sistema MIPS= freq. Multiplicador / 34 = 340MHz / 34 = 10 MHz
Como essa freq. encontrada (10MHz é menor que 80MHz, a mesma pode ser utilizada como freq. do sistema total.
Caso fosse maior que a reportada pelo timequest para o sistema sem o multiplicador, utilizaríamos a máxima freq. reportada.
Essa análise já é, basicamente a resposta do item g. Perceba como a utilização da arquitetura proposta para este multiplicador não casa bem à arquitetura do MIPS. Devido sua latência muito grande (2N + 2), o clock do sistema total fica comprometido
Obs.:
1) Latência do multiplicador é 2N + 2 se implementado exatamente igual ao diagrama de estados do Laboratório 4. Se for omitido o sinal de done e, consequentemente o estado S3, será 2N + 1. Se também for omitido o estado S0 e o sinal de start, 2N.
2) Como o multiplicador e o sistema MIPS sem o multiplicador funcionam com clocks distintos, o timequest conseguirá forneceres as máximas frequências pedidas no item c mesmo sendo rodado pro sistema total (ele analisará o caminho crítico para ambos os domínios de clock). No entanto, é mais fácil fazer separado. Daí a importância de termos hierarquias separadas para cada componente, individualmente.

4 Memória de Programa: Professor, para o TestBench do teste do MIPS_CPU, a memória de programa deverá já estar gravada ou será necessário que o Testbench realize o *boot* das instruções na memória e posteriormente as execute?

Sintetize, ambas as memórias, já com informação. Memória de programa com as instruções codificadas e a memória de dados já com os dados.

5 Sinais de Controle do módulo control: Professor, na figura da arquitetura MIPS a ser implementada existem sinais de controle em alguns blocos, minha dúvida é, o numero de sinais de controle representados em cada bloco será o número de sinais de controle real que cada bloco irá receber? Por exemplo, o módulo Register File Receberá 4 sinais de controle, a ALU 2 sinais de controle e assim suscetivamente?

Entendendo o funcionamento vocês terão condições de terminar quantos e quais serão os sinais de controle. O diagrama não é uma representação fidedigna.

6 Memoria de dados e addr decoding: Professor, não entendi muito bem como vai funcionar a memória de dados. A memoria interna começa em 0000h, e a memoria externa começa em nº * 500h? Então meu grupo sendo 1, a memoria externa começa em 500h? ou 1500h?

Número do Grupo multiplicado por 500h. Exemplo: Se seu grupo é o 1, o resultado será 0500h.

7 Como consigo manter o nome de um sinal para usar a diretiva *$init_signal_spy* para monitoramento de sinais internos em simulações GateLevel?

8 Em simulações gate *level*, a *netlist* do circuito é modificado para poder acomodar os parasitas dos *gates*. Desta forma, o motor de síntese altera os nomes das ligações do circuito que descrevemos pois, na verdade, um novo circuito é feito, compreendendo os parasitas. Esses nomes são gerados por software e se tornam praticamente indistinguíveis de outros, dada a complexidade dos mesmos.
9 Para forçar a barra e dizer à ferramenta de síntese que uma determinada ligação/nó de nossa descrição deve ter o nome mantido, podemos usar a diretiva *keep*. Exemplo:
*(*keep=1*) reg nomeDoNó;*
*(*keep=1*) wire nomeDoNó;*