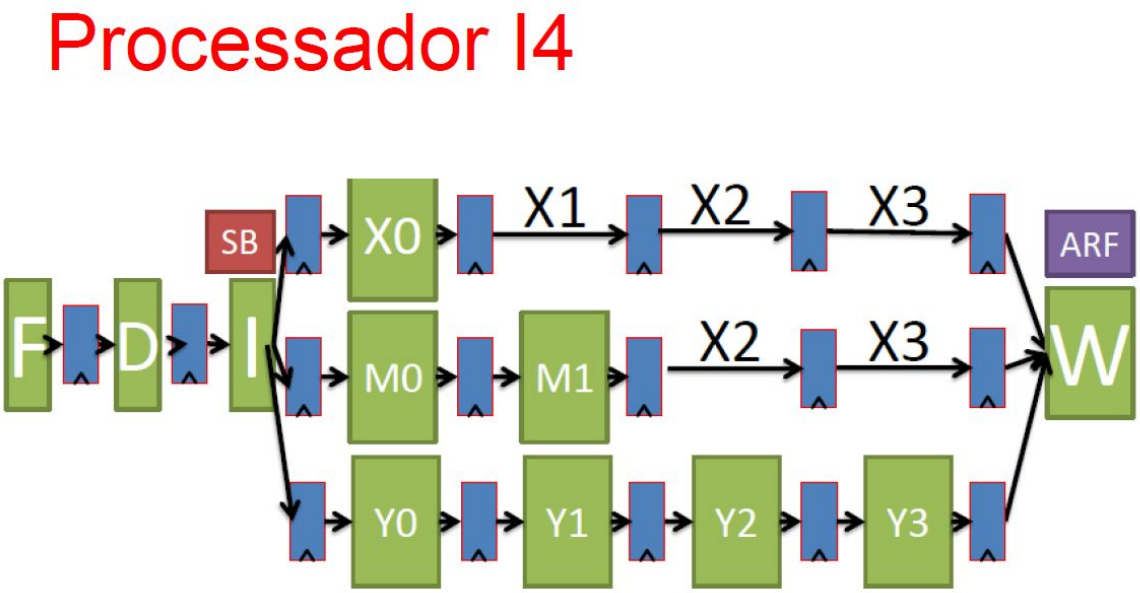
Descrição do TP 2

Agora que vocês já tem o MIPS pipeline funcionando, sem stall e sem encaminhamento, e já conhecem o Processador I4, o próximo trabalho será alterar o código que vocês tem do MIPS para que ele “vire” um I4. Segue imagem.



- ARF (Architecture Register File): Onde mantemos o estado da máquina;
- SB (ScoreBoard): Onde conseguimos as informações de onde os dados estão no pipeline.

O que vocês devem fazer:

- Criar o estágio “I” que irá encaminhar a instrução para o estágio adequado.
    - X0 é o que vocês já tem, executa soma, subtração, etc..
    - M0 e M1 para o Load e Store
        - M0 - cálculo do endereço de leitura/escrita
        - M1 - escrita/leitura na memória de dados
    - Y0, Y1, Y2 e Y3 para a multiplicação.
        - Y0 - Determina se o resultado da multiplicação será um valor positivo, negativo ou zero. (criar dois outputs, um para indicar se o resultado será positivo ou negativo e outro para indicar se o resultado será zero ou não).
        - Y1 - Mudança do sinal do número. Nosso circuito de multiplicação só opera com inteiros positivos. Fazer isso usando o complemento de 2 para

os números negativos. Seja o binário “1111”$_2$ = -1$_{10}$. Esse número tem que ser “0001”$_2$ = 1$_{10}$ para a multiplicação. O cálculo do complemento de 2 é feito da seguinte maneira: inverte os bits (0 vira 1 e 1 vira 0) e adiciona 1 ao resultado.

- Y2 - Realiza a multiplicação. C <= A * B por exemplo. No MIPS tradicional a instrução multiply salva o resultado da multiplicação em dois registradores, o HI (32 bits mais significativos) e o LO (32 bits menos significativos). Na implementação de vocês basta salvar os 32 bits menos significativos. A instrução é do formato R, com dois registradores operandos e 1 registrador destino, semelhante a instrução add. O OPCODE da nova instrução é “000000”$_2$ e o FUNCT é “011000”$_2$.
- Y3 - Detecção de overflow e mudança de sinal (se necessário). A multiplicação de dois números de 32 bits vai gerar um resultado com 64 bits. Nos interessa apenas os 32 bits menos significativos, mas se os 32 bits mais significativos forem maior que zero, o overflow tem que ser detectado. Se a multiplicação originalmente for gerar um número negativo, tem que ser feito o complemento de 2 no resultado final (mudança de sinal). Se houver overflow, o novo resultado não deve ser salvo no ARF.

Implementem uma unidade de detecção de hazard de dados e gerem stall no processador quando necessário.

Nesse trabalho não será necessário codificação na entrega, mas fora isso a correção será no mesmo estilo.

- Funcionamento das instruções load e store nos novos estágios (simulação no **MODELSIM** serve).
- Funcionamento da instrução mult nos novos estágios (simulação no **MODELSIM** serve).
- Funcionamento do ScoreBoard (simulação no **MODELSIM** serve).
- Funcionamento do projeto na placa.
    - Será necessário exibir o conteúdo dos registradores $0, $1, $2, $3, $4 nos diplays HEX0, HEX1, HEX2, HEX3 e HEX4 respectivamente (valores em decimal [3 bits menos significativos] ou em hexadecimal [4 bits menos significativos]).
    - Usar o input CLOCK_50 para a geração de clock na placa. Criem um contador de 32 bits que é incrementado em 1 no posedge do CLOCK_50 e usem o bit 24 do contador como clock do projeto. Dica: coloquem no LEDG0 esse bit para terem noção do período do clock.

Dica: usem o MARS para a geração de código de teste.