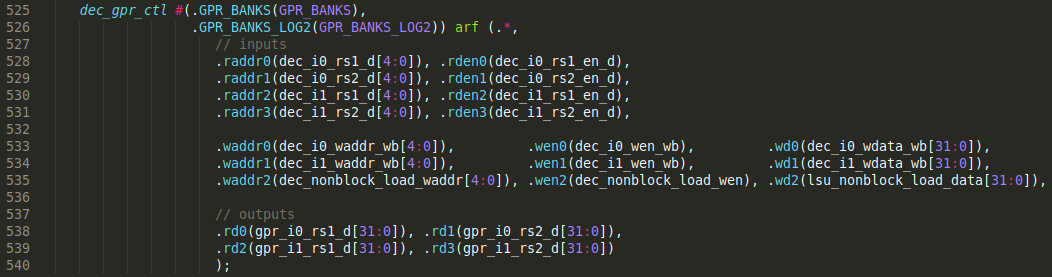
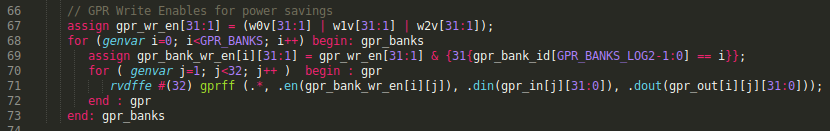
# 1. TAREFAS

**TAREFA:** O Register File é implementado no módulo **dec\_gpr\_ctl** e é instanciado no módulo **dec** (consulte a Figura 7). Analise o código Verilog e a simulação dos principais sinais do módulo **dec\_gpr\_ctl** (disponível no ficheiro *[RVfpgaPath]/RVfpga/src/SweRVolfSoC/SweRVEh1CoreComplex/dec/dec\_gpr\_ctl.sv*) para entender como ele funciona. Observe que o processador SweRV EH1 permite a inclusão de vários Register Files, mas a configuração usada no sistema RVfpga usa apenas um Register File (consulte a linha 402 do ficheiro *dec.sv*: localparam GPR\_BANKS = 1;).

**Instanciação no módulo dec**:

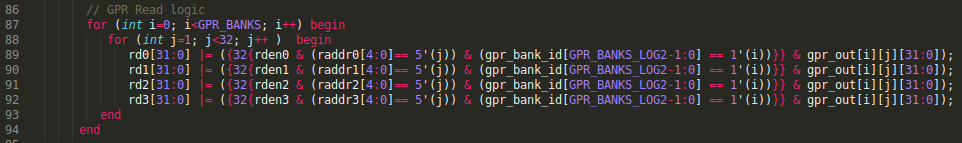


**Implementação dos 32 registos no módulo dec\_gpr\_ctl**:



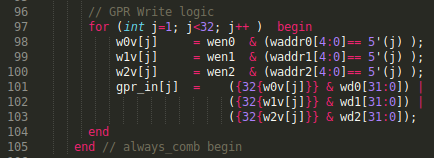
No nosso caso, apenas um banco é implementado. Para esse único banco, 31 registos são implementados instanciando 31 vezes o módulo **rvdffe** (que pode ser encontrado no ficheiro *[RVfpgaPath]/RVfpga/src/SweRVolfSoC/SweRVEh1CoreComplex/lib/beh\_lib.sv*). Observe que a largura de cada registo **rvdffe** é selecionada usando um parâmetro, que, no nosso caso, é de 32 bits🡪 rvdffe #(32). O registo 0 não é necessário, pois a arquitetura RISC-V força-o a ser sempre 0.

**Leitura de registos**:



São implementados 4 portos de leitura. Cada uma é atribuída com o valor do registo indicado pelos sinais raddr0/raddr1/raddr2/raddr3. Os sinais rden0/rden1/rden2/rden3 ativam/desativam a leitura. Observe que o valor inicial de *j* é 1, portanto, a leitura do registo 0 devolve sempre o valor 0.

**Escrita de registos**:



São implementados 3 portos de escrita. Cada registo é escrito com o valor fornecido nos sinais wd0/wd1/wd2, dependendo do endereço de registo waddr0/waddr1/waddr2. Os sinais wen0/wen1/wen2 ativam/desativam a escrita. Observe que o valor inicial de *j* é 1, portanto, não há escrita do registo 0.

**TAREFA:** Analise os bits de controle do multiplexer da Figura 8. Observe que os bits de controle estão no sinal e3d, que foi registado (pipelined) a partir do sinal dd, que foi gerado no andar Decode pela Unidade de Controle (consulte SweRVref.docx para obter descrições dos bits de controle).

* Se a instrução no DC3 for válida (e3d.i0v == 1) e for uma instrução load (e3d.i0load == 1), o valor proveniente do Pipe LSU será selecionado: i0\_result\_e3\_final = lsu\_result\_dc3.
* Se a instrução em EX3 for válida (e3d.i0v == 1) e for uma instrução mul (e3d.i0mul == 1), o valor proveniente do Multiplicador será selecionado: i0\_result\_e3\_final = exu\_mul\_result\_e3.
* Caso contrário, o valor proveniente do Pipe I0 é selecionado: i0\_result\_e3\_final = i0\_result\_e3.

**TAREFA:** Analise os bits de controle do multiplexer da Figura 9, que podem ser encontrados no módulo **dec\_decode\_ctl**.

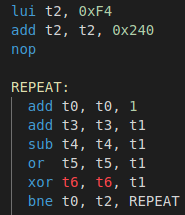
* Se o resultado em EX4 precisar ser selecionado na ALU secundária I0 (e4d.i0secondary == 1), o valor proveniente da ALU secundária I0 será selecionado: i0\_result\_e4\_final = exu\_i0\_result\_e4. Analisaremos a operação da ALU secundária no Lab 15.
* Se a instrução no DC4 for válida (e4d.i0v == 1) e for uma instrução load (e4d.i0load == 1), o valor proveniente do Pipe LSU será selecionado: i0\_result\_e4\_final = lsu\_result\_corr\_dc4.
* Caso contrário, o valor proveniente do Pipe I0 é selecionado: i0\_result\_e4\_final = i0\_result\_e4.

**TAREFA:** Replique a simulação da Figura 11 e da Figura 12 no seu computador seguindo estas etapas (conforme descrito em detalhes na Seção 7 do GSG):

* Se necessário, gere o binário de simulação (*Vrvfpgasim*).
* No PlatformIO, abra o projeto fornecido em: *[RVfpgaPath]/RVfpga/Labs/Lab11/ExampleProgram*.
* Defina o caminho correto para o binário de simulação do RVfpga (*Vrvfpgasim*) no ficheiro *platformio.ini*.
* Gere o trace da simulação com o Verilator (Generate Trace).
* Abra o trace usando o GTKWave.
* Use os ficheiros *test\_1.tcl* e *test\_2.tcl* (fornecidos em *[RVfpgaPath]/RVfpga/Labs/Lab11/ExampleProgram*) para abrir os mesmos sinais que os mostrados na Figura 11 e na Figura 12. Para isso, no GTKWave, clique em *File → Read Tcl Script File* e selecione o ficheiro *test\_1.tcl* ou *test\_2.tcl*.
* Clique em *Zoom In* ( ) várias vezes e vá para 48500ps (ou qualquer outra iteração do loop, exceto a primeira).

Solução fornecida no documento principal do Lab 11.

**TAREFA:** Execute o programa da Figura 13 na placa Nexys A7, conforme explicado no GSG. Você deve obter os resultados mostrados na Figura 14 para os quatro eventos medidos. Explique e justifique os resultados.



O programa é feito por um loop de 1000000 iterações que inclui 5 instruções aritméticas-lógicas e um salto condicional. Portanto, não há paragens devido a conflitos:

* + 6 \* 1000000 instruções são executadas
  + São executadas 2 instruções por ciclo, portanto: (6/2) \* 1000000 ciclos
  + 1000000 saltos são executados e quase todos eles acertam na previsão.

**TAREFA:** Medir outros eventos nos contadores em hardware para o programa da Figura 13. Para isso, você deve alterar no ficheiro *Test.c* a configuração dos eventos a serem medidos com a função pspPerformanceCounterSet. Observe que os diferentes eventos (mostrados na Tabela 1) podem ser referenciados usando as macros definidas no ficheiro PSP da WD: *.platformio/packages/framework-wd-riscv-sdk/psp/api\_inc/psp\_performance\_monitor\_eh1.h*. Por exemplo, se quiser medir o número de erros da I$ em vez do número de erros de salto, você deve substituir no ficheiro *Test.c* a linha: **pspPerformanceCounterSet**(D\_PSP\_COUNTER3, E\_BRANCHES\_MISPREDICTED);

para a linha: **pspPerformanceCounterSet**(D\_PSP\_COUNTER3, E\_I\_CACHE\_MISSES);

Solução não fornecida.

**TAREFA:** Proponha outros programas na função Test\_Assembly e verifique se os diferentes eventos fornecem os resultados esperados. Pode tentar outras instruções tais como leituras, escritas, multiplicações, divisões... bem como conflitos que provocam paradas no pipeline.

Solução não fornecida.