universidade de aveiro

Mestrado Integrado em Engenharia de Computadores e Telemática

Arquitetura de Computadores Avançada

Trabalho de Grupo 1

Implementação de unidades de forwarding e de stalls numa arquitetura pipeline

Grupo:
Bernardo Ferreira 67413
Bruno Silva 68535

Introdução

Para o trabalho de ACA do ano letivo 2015/2016 era pedido para usar o programa MIPS_SystemC de forma a poder implementar vários mecanismos usados em processadores. O trabalho consistia em 5 exercícios dos quais 4 eram de implementação e 1 era teórico. Os exercícios pedidos passavam por dividir a fase ID em duas, ID1 e ID2 respectivamente, alterar a resolução dos saltos condicionais para ID2, implementar as instruções bne, bnez, bgtz, blez, j e jr, implementar uma unidade de forwarding e uma unidade de controlo responsável por controlar a execução das instruções e os data hazards necessários.

Neste relatório estão descritos de forma pormenorizada os exercícios e as decisões tomadas na sua implementação. Assim como no exercício teórico estão descritos todos os casos de forwarding como pedido.

Exercício 1

No Exercício 1 era pedido para dividir a fase ID em duas fases, ID1 e ID2 respectivamente, por forma a satisfazer essa condição foi adicionado um registo ID1/ID2 construído de forma semelhante aos restantes registos mas apenas com os valores pretendidos. Este novo registo realizou a divisão da fase ID e possibilitou assim o aumento da frequência do processador. A divisão dos componentes de ID fez-se da seguinte forma, em ID1 ficou o decoder da instrução apenas, enquanto que em ID2 ficou a unidade de controlo, o mux que seleciona o writeReg e o ext que extende o sinal. O acesso aos registos, uma vez que é síncrono, ou seja funciona com o clock, encontra-se situado como parte do registo ID1/ID2, o que significa que em ID1 entram o numero dos registos a ser lidos e em ID2 já saem da unidade os valores dos mesmos registos. Para exemplificar o que foi feito pode-se olhar para a imagem ilustrativa do exercício 2 que embora já tenha mais componentes dá para compreender a divisão efetuada.

Dada esta nova arquitetura e de forma a que as instruções continuem a ser executadas de forma correta foi necessário alterar as condições para quando há stalls, ficou-se então com as seguintes condições:

```
rs_id2.read() != 0 && rs_id2.read() == WriteReg_exe.read() && RegWrite_exe.read() ==
true
||
rs_id2.read() != 0 && rs_id2.read() == WriteReg_mem.read() && RegWrite_mem.read() ==
true
||
rs_id2.read() != 0 && rs_id2.read() == WriteReg_wb.read() && RegWrite_wb.read() ==
true
||
rt_id2.read() != 0 && rt_id2.read() == WriteReg_wb.read() && RegWrite_wb.read() ==
true && MemRead.read()==false
||
rt_id2.read() != 0 && rt_id2.read() == WriteReg_exe.read() && RegWrite_exe.read() ==
true && MemRead.read() == false
||
rt_id2.read() !=0 && rt_id2.read() == WriteReg_mem.read() && RegWrite_mem.read() ==
true && MemRead.read() == false
```

Exercício 2

No exercício 2 eram pedidas varias tarefas como tal, em baixo descrevemos um pouco do que foi feito em cada uma delas. Contudo no final da descrição dos 3 sub exercícios está um esquema representativo das fases ID1 e ID2 no final deste exercício 2.

2.1

Neste execício era pedido que a resolução dos saltos condicionais fosse antecipada para a fase ID2 em vez de MEM. Desta forma quando os saltos fossem taken menos instruções eram descartadas e mais eficiente se torna a pipeline. Era também pedido para implementar a técnica de delay branch slot que faz com que a próxima instrução a seguir ao branch seja sempre executada independentemente do resultado do branch.

Para proceder as alterações em vez de mudar todos os componentes necessários a resolução do branch para id2, esses componentes foram apagados e substituídos por uma unidade de branch que seria responsável por realizar todos os cálculos necessários ao branch e produzir dois valores, uma flag (branchTaken) que diz se o branch é taken ou não, e outro que diz qual o endereço para onde saltar (branchTarget).

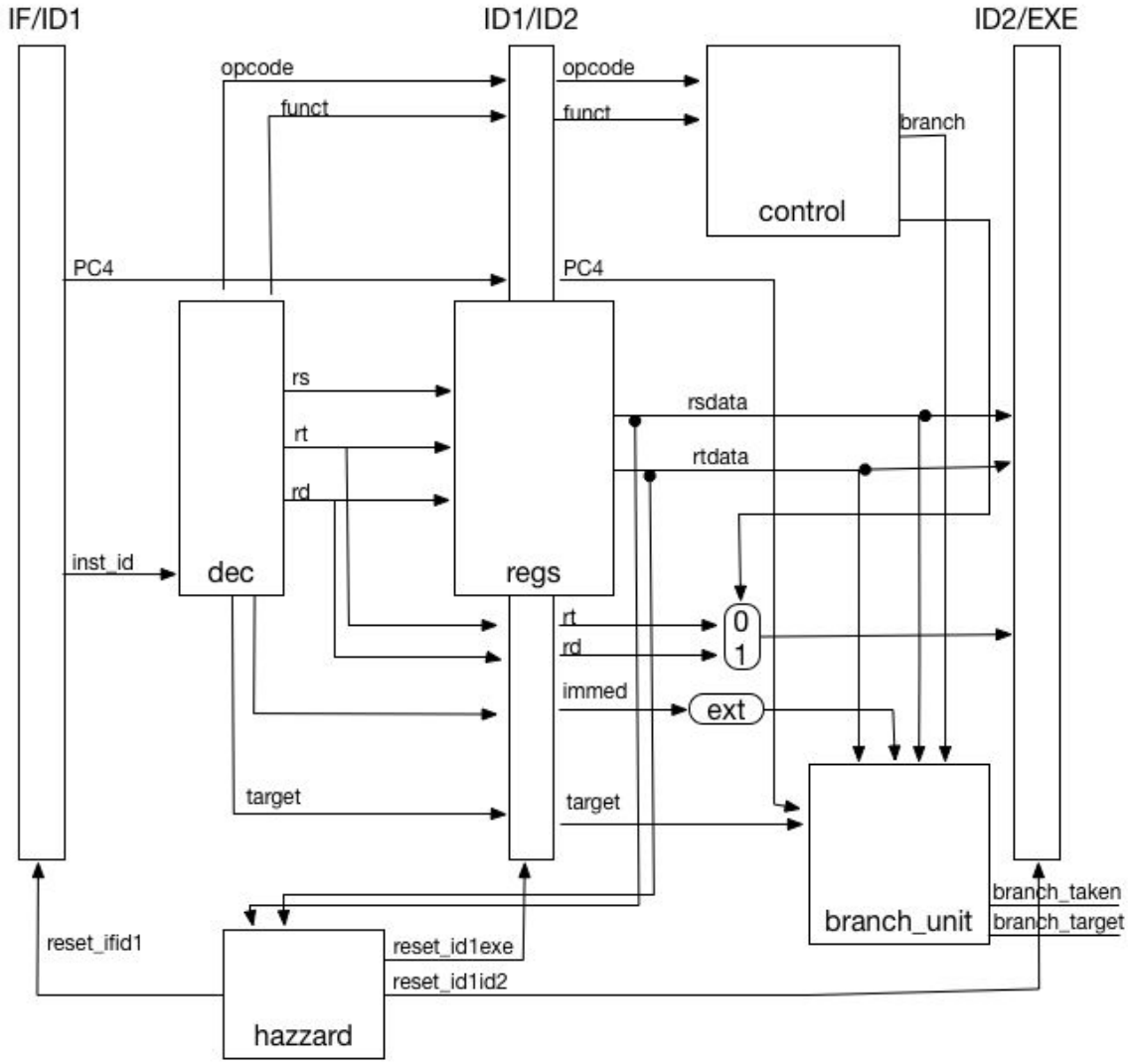
Na unidade de branch entra o sinal “branch” da unidade de controlo, os valores dos registos rs e rt, o PC4 e o imm. Para calcular o salto é visto o valor de branch de entrada e se for true, é calculado o branchTarget com a soma do PC4 com o imm*4.

2.2

No exercício 2.2 era pedido para implementar as instruções de bne, bgtz, blez, j e jr. O estado da pipeline manteve-se igual desde o exercício anterior com pequenas alterações em 3 unidades. Estas 3 unidades foram o decoder que agora passa a ter uma saída jtarget que envia os 26 bits menos significativos para a unidade de branch, a unidade de controlo teve o sinal de branch alterado que em vez de ser um sinal boleano passou a ser um sinal de 3 bits onde codificamos consoante o opcode, qual o tipo de salto a efetuar, e a unidade de branch que agora utiliza um switch case para dependendo do tipo de salto a efetuar produzir resultados diferentes nos sinais se saída.

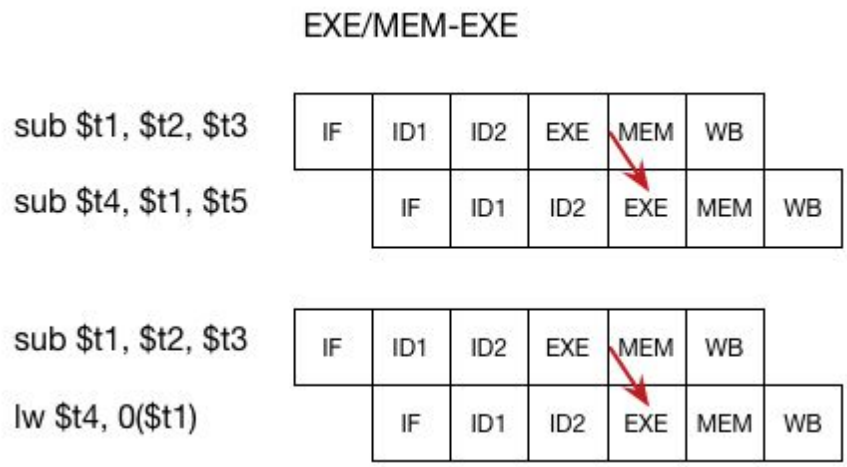
2.3

No exercício 2.3 era pedido apenas para descartar a instrução que entra erradamente na pipeline quando o branch é taken. Para isso na unidade de hazard entra o sinal branchTaken produzido na unidade de branch. A unidade de hazard usa esse sinal e caso seja true, faz reset a instrução em IF.


IF/ID1
ID1/ID2
ID2/EXE
opcode
funct
opcode
funct
branch
control
PC4
PC4
rs
rt
rd
rsdata
rtdata
inst_id
dec
regs
rt
rd
0
1
immed
ext
target
target
branch_unit
branch_taken
branch_target
reset_ifid1
reset_id1exe
reset_id1id2
hazzard

Exercício 3

- Tipos de Forwarding
  - EXE/MEM - EXE



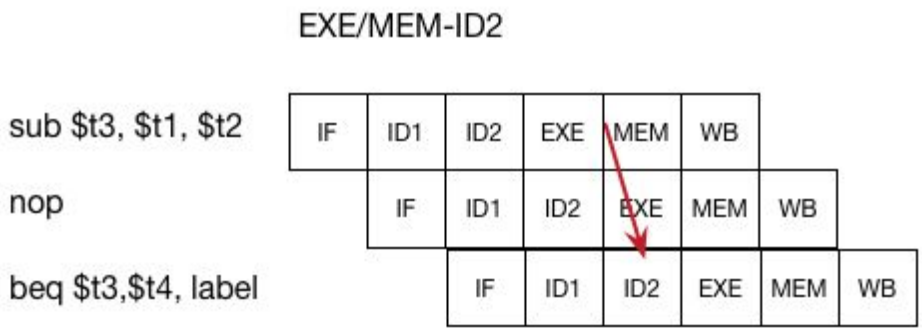
Este tipo de Forwarding acontece quando uma instrução precisa do valor de um registo que ainda não foi escrito numa instrução anterior. Para se detetar quando é necessário fazer este forwarding tem de se verificar as seguintes condições:

- Se os registos rs ou rt em EXE forem diferentes de 0.
- Se os registos rs ou rt na fase EXE forem iguais ao registo que está a ser escrito na fase MEM.
- RegWrite em MEM estar a *true*.

No caso de se verificarem estas condições, coloca-se o multiplexer do rs ou rt com o valor 1 para que o rs ou rt a ser selecionado para entrar na ALU seja igual ao da saída da ALU no registo EXE/MEM.

```
    // EXE/MEM --> EXE (rs)
    if ( rs_exe.read() != 0 && rs_exe.read() == WriteReg_mem.read() &&
RegWrite_mem.read() == true ){
        rs_mux_exe.write(1);
    }
    // EXE/MEM --> EXE (rt)
    if ( rt_exe.read() != 0 && rt_exe.read() == WriteReg_mem.read() &&
RegWrite_mem.read() == true && MemRead_exe.read() == false ){
        rt_mux_exe.write(1);
    }
```

- EXE/MEM - ID2



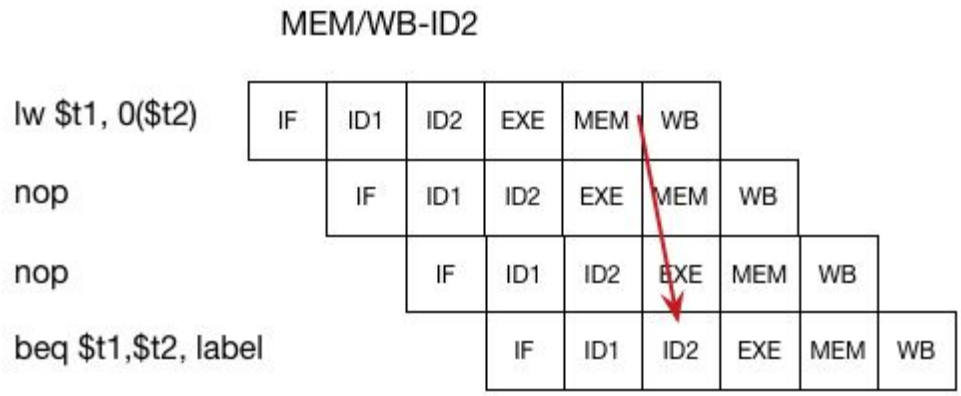
Este tipo de Forwarding acontece quando uma instrução de branch precisa de o valor de um registo em ID2 que esta a ser escrito por outra instrução. Para se detetar quando é necessário fazer este forwarding tem de se verificar as seguintes condições:

- Se os registos rs ou rt em ID2 forem diferentes de 0.
- Se os registos rs ou rt na fase ID2 forem iguais ao registo que está a ser escrito na fase MEM.
- RegWrite em MEM estar a *true*.
- Se o Branch for diferente de 0.

No caso de se verificarem estas condições, coloca-se, no multiplexer de ID2, o valor 1 para que o rs ou rt a ser selecionado seja o valor que vem do forward.

```
// EXE/MEM --> ID2 (rs - saltos)
       if  (  rs_id2.read()  !=  0  &&  rs_id2.read()  ==  WriteReg_mem.read()  &&
RegWrite_mem.read() == true && Branch.read() != 0 ){
       rs_mux_id2.write(1);
   }
// EXE/MEM --> ID2 (rt - saltos)
       if  (  rt_id2.read()  !=  0  &&  rt_id2.read()  ==  WriteReg_mem.read()  &&
RegWrite_mem.read() == true && Branch.read() != 0  && MemRead.read() == false ){
       rt_mux_id2.write(1);
   }
```

- MEM/WB - ID2



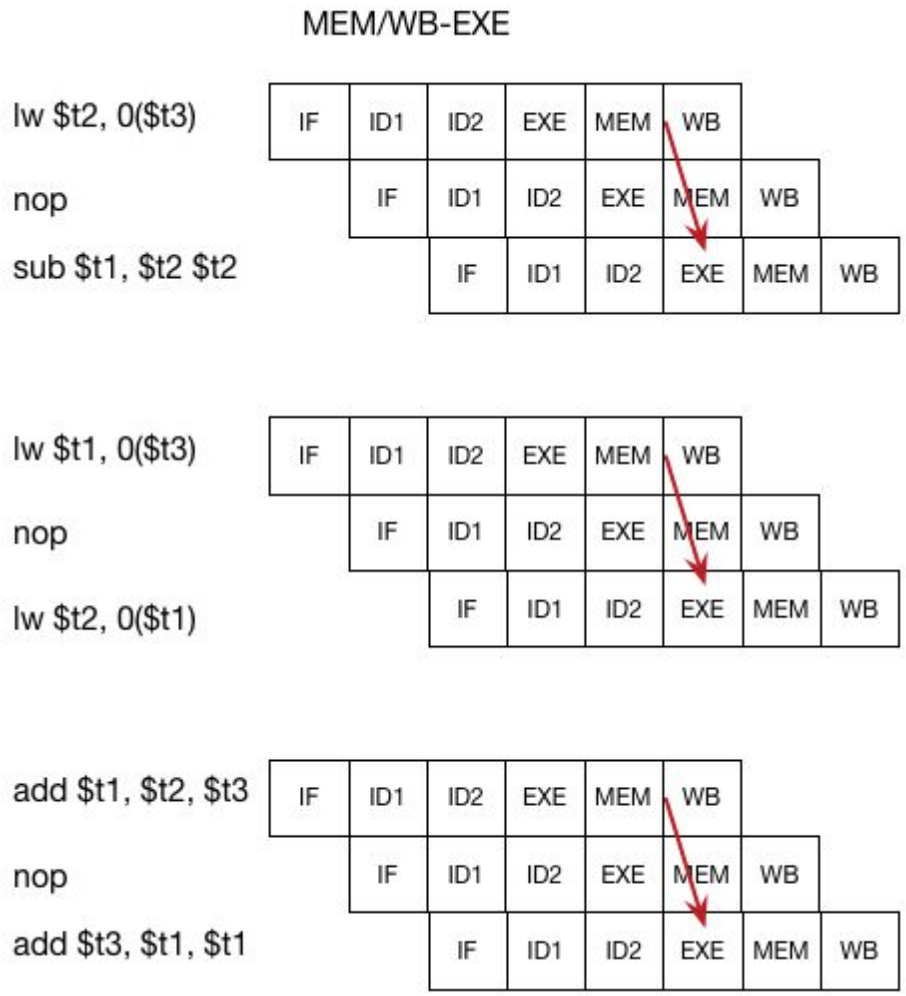
Este tipo de Forwarding acontece quando uma instrução de branch precisa de o valor de um registo em ID2 que esta a ser escrito por uma instrução lw. Para se detetar quando é necessário fazer este forwarding tem de se verificar as seguintes condições:

- Se os registos rs ou rt em ID2 forem diferentes de 0.
- Se os registos rs ou rt na fase ID2 forem iguais ao registo que está a ser escrito na fase WB.
- RegWrite em WB estar a *true*.

No caso de se verificarem estas condições, coloca-se, no multiplexer de ID2, o valor 2 para que o rs ou rt a ser selecionado seja o valor que vem do forward.

```
// MEM/WB --> ID2 (rs - saltos)
    if ( rs_id2.read() != 0 && rs_id2.read() == WriteReg_wb.read() && RegWrite_wb.read()
== true ){
        rs_mux_id2.write(2);
    }
// MEM/WB --> ID2 (rt - saltos)
    if ( rt_id2.read() != 0 && rt_id2.read() == WriteReg_wb.read() && RegWrite_wb.read()
== true && MemRead.read() == false ){
        rt_mux_id2.write(2);
    }
```

- MEM/WB - EXE



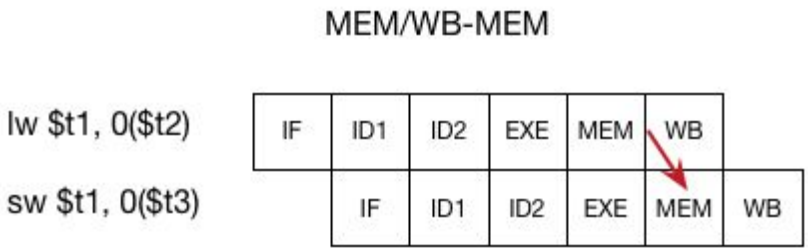
Este tipo de Forwarding acontece quando uma instrução precisa de o valor de um registo em EXE que esta a ser escrito por outra instrução. Para se detetar quando é necessário fazer este forwarding tem de se verificar as seguintes condições:

- Se os registos rs ou rt em EXE forem diferentes de 0.
- Se os registos rs ou rt na fase EXE forem iguais ao registo que está a ser escrito na fase WB.
- RegWrite em WB estar a *true*.

No caso de se verificarem estas condições, coloca-se, no multiplexer de EXE, o valor 2 para que o rs ou rt a ser selecionado seja o valor que vem do forward.

```
// MEM/WB --> EXE (rs)
            if  (  rs_exe.read()  !=  0  &&  rs_exe.read()  ==  WriteReg_wb.read()  &&
RegWrite_wb.read() == true ){
          rs_mux_exe.write(2);
     }
// MEM/WB --> EXE (rt)
            if  (  rt_exe.read()  !=  0  &&  rt_exe.read()  ==  WriteReg_wb.read()  &&
RegWrite_wb.read() == true && MemRead_exe.read() == false ){
         rt_mux_exe.write(2);  }
```

- MEM/WB - MEM



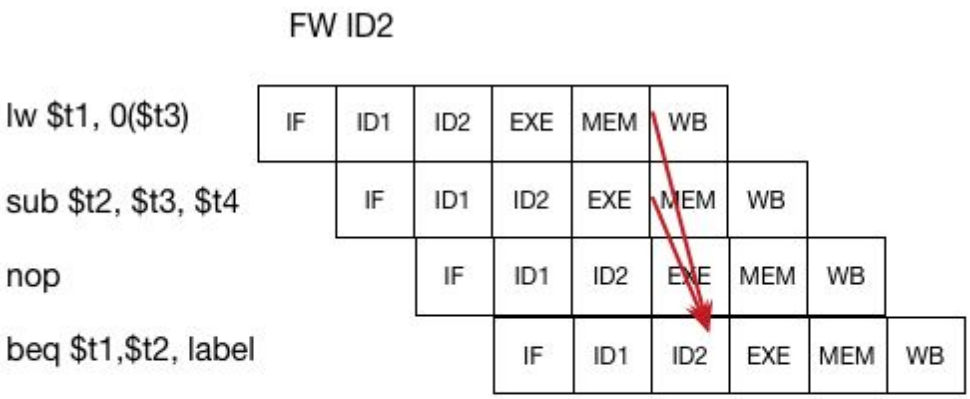
Este tipo de Forwarding acontece especificamente quando há uma instrução lw seguida de um sw em que a intrução de lw escreve no mesmo registo que a intrução de sw vai escrever Para se detetar quando é necessário fazer este forwarding tem de se verificar as seguintes condições:

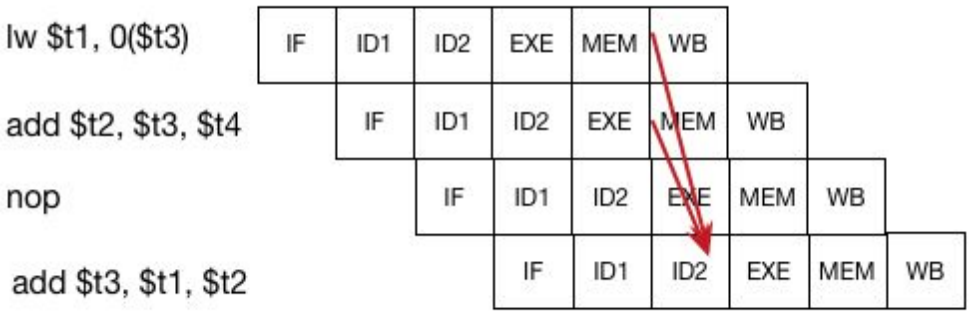
- Se o registo rt em MEM for diferente de 0.
- Se o registo rt na fase MEM for igual ao registo que está a ser escrito na fase WB.
- RegWrite em WB estar a *true*.
- MemRead em WB estar a *true*.
- MemWrite em MEM estar a *true*.

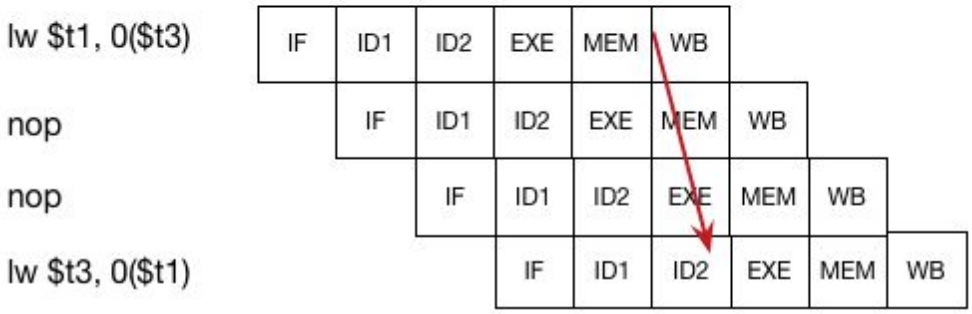
No caso de se verificarem estas condições, coloca-se, no multiplexer de MEM, o valor 1 para que o rs ou rt a ser selecionado seja o valor que vem do forward.

```
// MEM/WB --> EME (lw -> sw)
if ( rt_mem.read() != 0 && rt_mem.read() == WriteReg_wb.read() && RegWrite_wb.read()
== true && MemRead_wb.read() == true && MemWrite_mem.read() == true ){
     mux_mem.write(1);
}
```

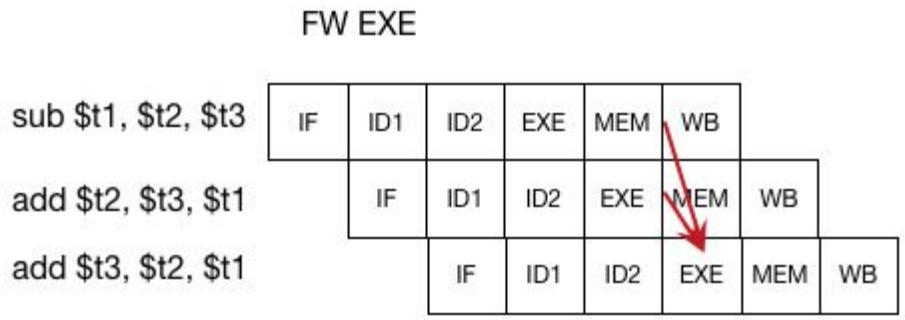
- Outras situações de Forwarding consideradas
  - Forwarding para ID2



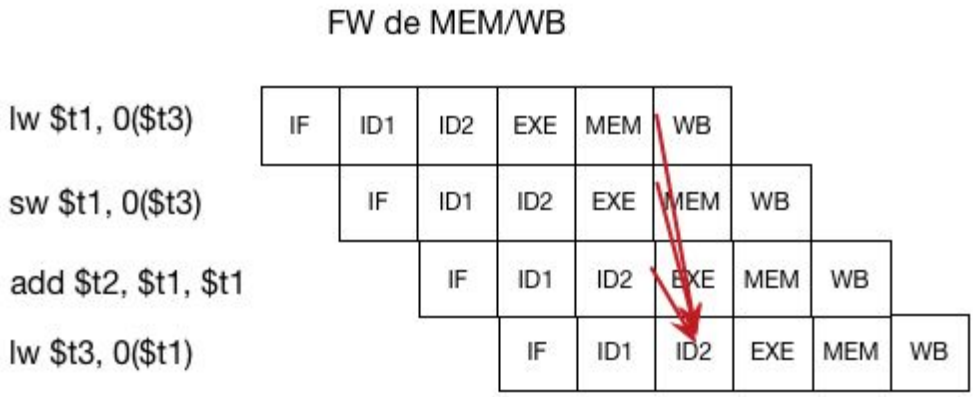




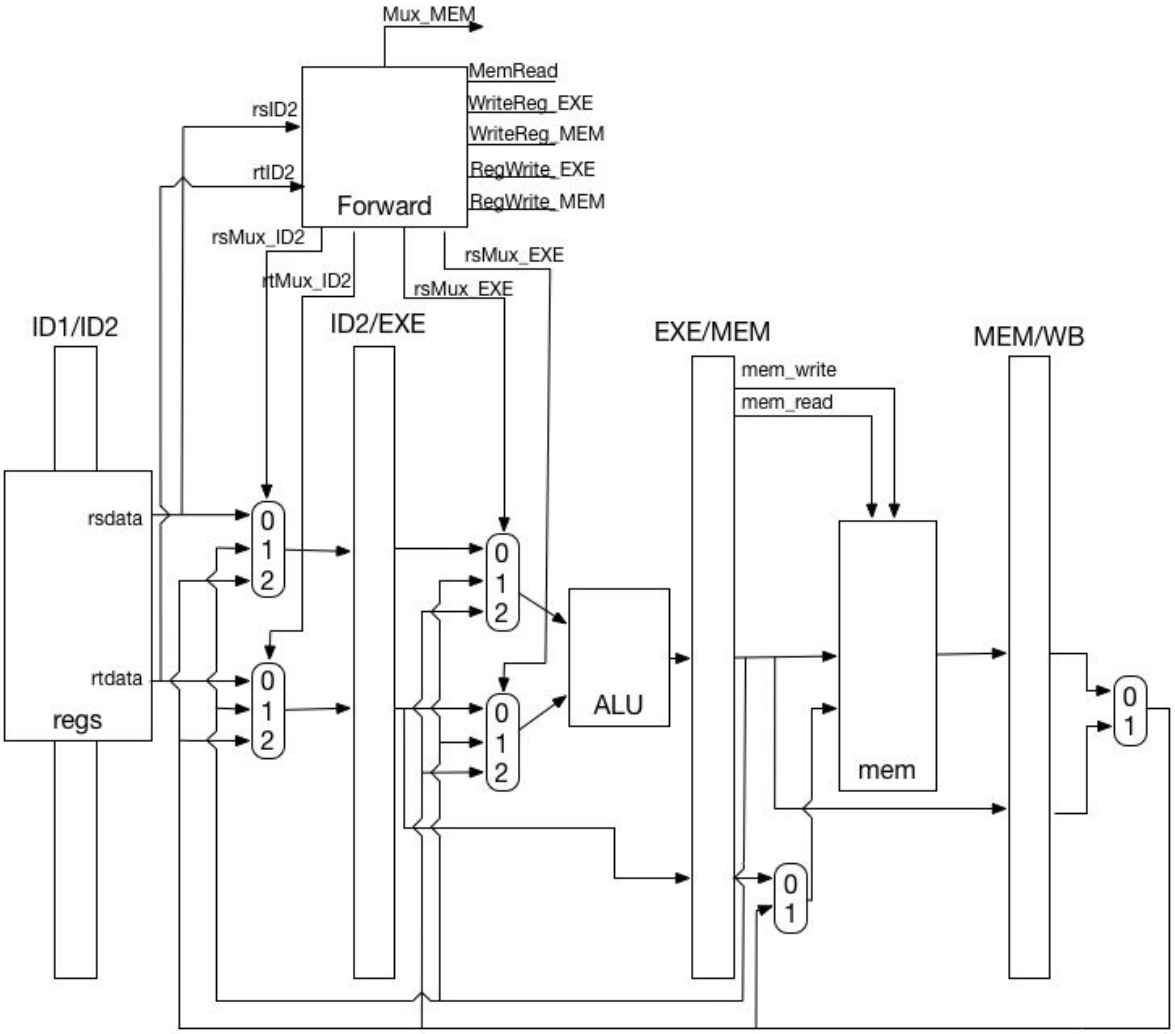
- Forwarding para EXE



- Forwarding de MEM/WB



Exercício 4

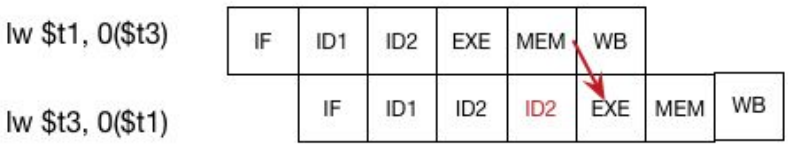


No exercício 4 era pedida a implementação dos tipos de forward identificados no exercício 3. Para isso criamos um novo módulo(*forwardunit.cpp*) e fizemos as alterações necessárias para a correta implementação das condições de forwarding.

Exercício 5

No exercício 5 foi pedido que identificássemos as situações em que apesar de haver forwarding, teriam de se inserir Stalls para o correto funcionamento dos programas.
As situações em que verificamos que existia esta necessidade foram as seguintes:

- 1ºCaso



Nestes casos, temos um situação de duas instruções seguidas em que a primeira escreve num registo que posteriormente vai ser usado para leitura. Nesta situação, para um correto funcionamento, tem de se inserir um stall em ID2 e fazer um forwarding de MEM/WB - EXE.

- 2ºCaso

Nesta situação, quando temos um branch que precisa do valor de um registo que só é calculado em EXE, portanto terá de ser fazer um stall em ID2 e um forward de EXE/MEM - ID2.

- 3ºCaso

Neste último caso, quando temos uma instrução de lw que escreve num registo que uma branch seguidamente utiliza para ler em ID2, tem de se inserir 2 stalls até ser possível fazer um forwarding de MEM/WB como o valor correto a ser usado pela instrução de branch.

Conclusão

Com este trabalho, conseguimos cimentar conhecimentos de arquitetura de computadores, bem como perceber melhor o funcionamento de uma arquitetura de processadores pipelined. Apesar de no exercício 2 termos falhado a implementação da instrução de jump(erro que foi corrigido nesta entrega final), consideramos que atingimos com sucesso todos os objetivos equacionados para este trabalho.