FUDAN UNIVERSITY, SOFTWARE SCHOOL

Computer Architecture
Project1 Part2 Design

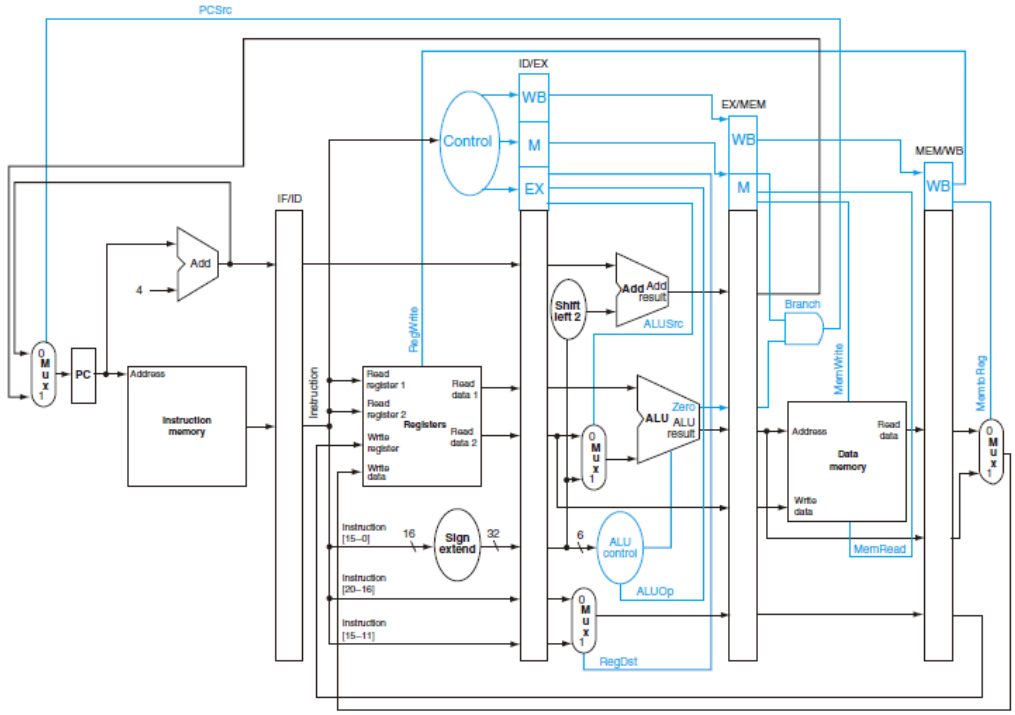
Wang Xin 10302010023

April 28, 2013

The Part2 of this project was to design and implement CPU pipelines both with and without stall. In order to get the full concept of the SimpleScalar simulation, we need to get a brief view of the source file.

1 PIPELINE WITH STALL

In Part1, we get the format of instruction implementation. In sim-pipe.[c/h], the simulator includes the file "machine.def"(in list 1). According to the five-stage pipeline defined in the textbook, we could get a overview of the pipeline design. The datapath and control logic was shown in the figure 1.1. What we need is to map the structure to the data structure defined in sim-pipe.h and the operation sequence defined in sim-pipe.c.



1 / 1

Figure 1.1: Pipeline with stall

Listing 1: instruction format used in sim-pipe.c

```
#define DEFINST(OP,MSK,NAME,OPFORM,RES,FLAGS,O1,O2,I1,I2,I3)\
  if (OP==de.opcode){\
    de.oprand.out1 = O1;\
    de.oprand.out2 = O2;\
    de.oprand.in1 = I1;\
    de.oprand.in2 = I2;\
    de.oprand.in3 = I3;\
    goto READ_OPRAND_VALUE;\
  }
#define DEFLINK(OP,MSK,NAME,MASK,SHIFT)
#define CONNECT(OP)
#include "machine.def"
```

1.1 Data Structure & Control Logic

Observing the data structure defined in sim-pipe.h, we could get a general view of the register file used in the five-stage pipeline. The point is to determine the control signal needed in each stage. I handle the five stages in reverse order so that we can handle the instruction sequence in normal order.

- IF/ID Register File. In this stage, the processor fetches the new instruction by PC and store the instruction in register file. In this case, some kind of PC calculation could be done in this stage(e.g. JUMP).
- ID/EX Register File. Most control signals were determined in this stage. With instruction decoded, we can get the instruction type and operation fields of a specific instruction(e.g. RegisterRs, RegisterRt, RegisterRd, Shamt, ExtendedImm) and most of the control signals(e.g. RegDst, MemtoReg, RegWrite, MemRead, MemWrite, Branch, Latch)
- EX/MEM Register File. Most operations was done in this stage. Write the program according to the The SimpleScalar Tool Set, Version 2.0. One thing to point out is that the definition of **"SLTI"** in this document is wrong.
- MEM/WB Register File. PCSrc, RegDst was not needed since it will not be passed to this stage.)

1.2 Stall Detection

The first pipeline was without forwarding technique, hence the pipeline should stall a cycle when detecting a hazard.

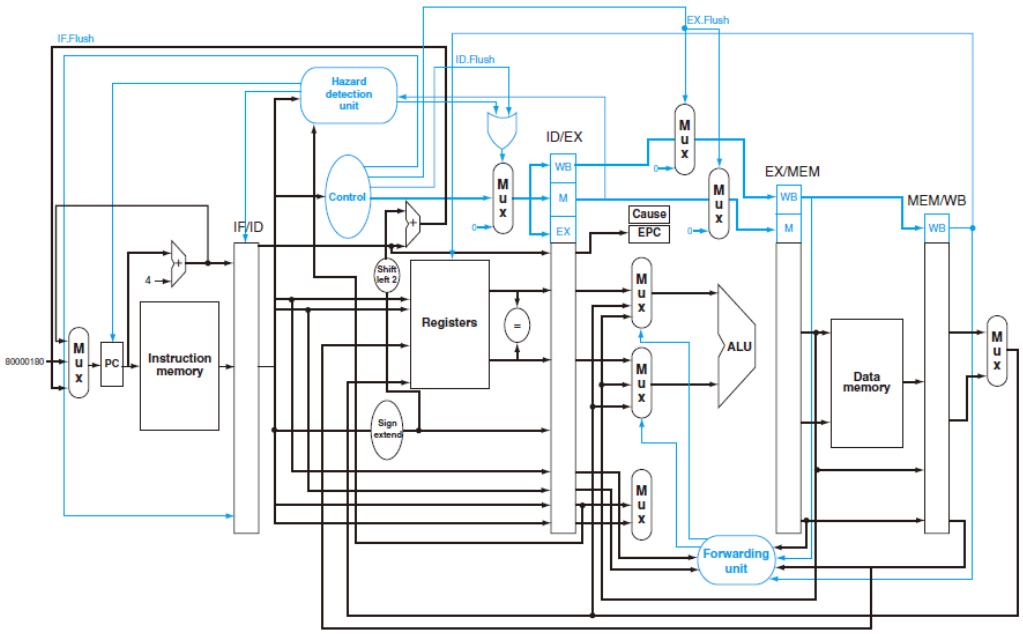
- Data Hazard. I determine the stall logic according to the Page229 in textbook. Mainly two types: EX Hazard and MEM Hazard. There are no forwarding thus the order is not important(listing 2).

Listing 2: Data Hazard Detection

```
if((em.RegWrite == 1 &&
(em.inst.a != NOP && em.oprand.out1 != DNA
&& (em.oprand.out1 == de.oprand.in1
|| em.oprand.out1 == de.oprand.in2)))//EX Hazard
||(mw.RegWrite == 1 &&
(mw.inst.a != NOP && mw.oprand.out1 != DNA
&& (mw.oprand.out1 == de.oprand.in1
|| mw.oprand.out1 == de.oprand.in2))))//MEM Hazard
```

- Control Hazard. This is a difficult point in this lab even though there is just one Branch instruction: BNE. Theoretically, the branch result should be determined as soon as possible. However, as there is no forwarding, the branch result could not be resolved before EXE stage. Moreover, when branch instruction is in decode stage, it has to stall the

pipeline for one cycle waiting for the comparison result calculated in ALU. Hence I set a field "Latch" in ID/EX Register File, whenever a Branch instruction is met, it should set the "Latch" signal and the3 Stall check int the next loop will not fetch a new instruction. Of course, we should release the lock carefully otherwise the pipeline will be locked forever.



1 / 1

Figure 2.1: Pipeline with forward

2 Pipeline with forwarding

In this part, we need to add some forwarding mechanism in the pipeline in order to reduce the stall overhead. The overall design is similar to the diagram described in textbook (Figure 2.1).

2.1 Forward Detection

The forward detection is designed according to the two types in textbook.

- EX Hazard Similar to Stall Detection, but in this data structure, the register could not be specified just by oprand. Hence we need to check the register number especially after the forwarding detection(Listing 3).

Listing 3: EX forwarding Detection

```
if(em.inst.a != NOP && !em.MemRead &&
     (de.oprand.in1 == em.oprand.out1
||de.oprand.in2 == em.oprand.out1)){
/*the value of out1 is not determined*/
if(em.oprand.out1 == de.RegisterRs){
```

```
        /*printf("ForwardA = 10\n");*/
        de.ReadData1 = em.ALUResult;
    }else if(em.oprand.out1 == de.RegisterRt){
        /*printf("ForwardB = 10\n");*/
        de.ReadData2 = em.ALUResult;
    }
```

- MEM Harzard. Similar to EX Hazard. There is a key point that the result of MEM Forwarding should never overwrite the result of EX Forwarding for the latter will provide the newest result. Therefore, we should use a if-else clause to avoid double forwarding when a register was modified consecutively(Listing 4).

Listing 4: MEM forwarding Detection

```
else if(mw.inst.a != NOP &&
     (de.oprand.in1 == mw.oprand.out1
    ||de.oprand.in2 == mw.oprand.out1)){
     if(mw.oprand.out1 == de.RegisterRs){
          /*printf("ForwardA = 01\n");*/
          de.ReadData1 = mw.MemtoReg == 1?
        mw.MemReadData:mw.ALUResult;
     }else if(mw.oprand.out1 == de.RegisterRt){
          /*printf("ForwardA = 01\n");*/
          de.ReadData2 = mw.MemtoReg == 1?
        mw.MemReadData:mw.ALUResult;
     }
}
```

2.2 Stall Detection

The load-use Hazard could not be solved even with forwarding for the pipeline could not get the value before MEM stage. A stall is still necessary(Listing 5).

Listing 5: Load-use Hazard Detection

```
if(em.inst.a != NOP && em.MemRead == 1 &&
            (em.oprand.out1 == de.oprand.in1
          || em.oprand.out1 == de.oprand.in2)){
          fd.inst = de.inst;
          fd.PC = de.PC;
          de.inst.a = NOP;
}
```

2.3 About Branch Instruction

As is mentioned in the previous section, the branch should be determine as soon as possible. With forwarding, we could get the value of two inputs in decode stage. However, the forwarding mechanism for branch is a little different with others. For the comparison result should

be resolved before the decode stage is over, we need an **extra** forwarding during this stage otherwise the next instruction will get into the decode stage and there still has to be a stall.