Solution to HW3 of CMSC611

1.a Here we assume that branch is solved at the MEM stage since no extra hardward is allowed.

Note: in the chart, the stall is represented by "st" clock cycle

instruction clock cycles

```
                  1  2  3   4     5   6   7      8      9  10  11    12 13  14    15    16  17  18  19 20 21  22
-------------------------------------------------------------------------------------
LW R1,0(R4) IF ID EX MEM WB
SLLI R1,R1,#3  IF  ID  st     st  EX MEM WB
LW R2,4(R4)       IF  st     st  ID  EX   MEM WB
ADD R1,R2,R1                  IF  ID    st     st EX MEM WB
SW R1,0(R4)                       IF     st     st ID  st     st  EX MEM WB
ADDI R4,R4,4                                 IF  st     st  ID EX    MEM WB
SUB R6,R3,R4                                       IF ID     st     st  EX MEM WB
BNZ R6, loop                                             IF      st     st  ID  st     st EX MEMWB
-----------
LW R1,0(R4)                                                                          IF
```

From above chart, we can see for one iteration, it will take 21 cycles since R4 = R3 - 196, then there are 196/4 = 49 iterations.

Total cycles = 21 * 48 + 22 = 1030

b. Here we assume that branch is solved at the ID stage.

instruction clock cycles

```
                  1 2   3   4     5   6     7      8     9    10     11    12     13  14  15
LW R1,0(R4)   IF ID EX MEM WB
SLLI R1,R1,#3    IF  ID  st     EX MEM WB
LW R2,4(R4)         IF  st     ID  EX    MEM WB
ADD R1,R2,R1               IF  ID    st     EX MEM WB
SW R1,0(R4)                    IF     st     ID  EX    MEM WB
ADDI R4,R4,4                              IF  ID   EX     MEM WB
SUB R6,R3,R4                                  IF    ID     EX     MEM WB
BNZ R6,loop                                         IF     ID     st     EX  MEM WB
---------------
LW R1,0(R4)                                                 IF
```

Total cycles = 12 * 48 + 15 = 591

c.instruction clock cycles

```
                   1  2  3  4     5     6     7     8     9       10    11  12
LW R1,0(R4)    IF ID EX MEM WB
LW R2,4(R4)       IF ID  EX    MEM
SLLI R1,R1,#3        IF   ID    EX  MEM WB
ADDI R4,R4,4              IF    ID  EX    MEM WB
SUB R6,R3,R4                  IF  ID    EX    MEM WB
ADD R1,R2,R1                      IF    ID    EX    MEM  WB
BNZ R6,loop                           IF    ID    EX    MEM WB
SW R1,-4(R4)                              IF    ID    EX    MEM WB
-------------
LW R1,0(R4)                                       IF
```

Total cycles = 8 * 48 + 12 = 396

C code:

```
Int A[49];
for(int i = 1; i<= 49; i++){
        A[i] = A[i] * 8 + A[i + 1];
}
```

2. a. IF: needs one adder to compute PC

RF: does not need adder

Alu1: needs one adder to compute address

Alu2: needs one adder for ADD operation

MEM: does not need adder

WB: does not need adder

So, we need three adders in this system.

b. We need one register read and one register write port, two memory read port and one memory write port. Because RF read register, WB write register, WB is at the first half of the circle, RF is at the second half of the cycle, so there is no confliction. Therefore, only one port for each is needed.

IF read memory, MEM read/write memory, IF and MEM can overlap, so we need to assign two ports for read, one port for write.

c. Data forwarding between ALUs:

pipeline register of source instruction	opcode of source instrution	pipeline register of source instruction	opcode of source instruction
ALU2/WB	r-r ALU	RF/ALU1	load, store, branch, ALUimm
ALU2/WB	ALUimm	RF/ALU1	load, store, branch, ALUimm
ALU2/WB	r-r ALU	MEM/ALU2	r-r ALU, ALUimm
ALU2/WB	ALUimm	MEM/ALU2	r-r ALU, ALUimm

d. Data forwarding between memory and ALU, memory and memory.

pipeline register of source instruction	opcode of source instrution	pipeline register of destination instruction	opcode of destination instruction
MEM/ALU2	load	RF/ALU1	load, store, branch, ALUimm,r-rALU
MEM/ALU2	load	MEM/ALU2	r-rALU, ALUimm
ALU2/WB	r-r ALU	ALU1/MEM	store, branch
ALU2/WB	ALUimm	ALU1/MEM	store, branch
MEM/ALU2	load	ALU1/MEM	store

e. The remaining hazard that involve at least one unit other than an ALU as the source or destination unit.

Source instruction	Destination instruction	length of hazard
Load	Load, store, ALUimm	one
ALUop	store,	one
Load	ALUop	one or two

f. Control hazard types. Since the pipeline initiate one instruction every cycle, and the branch results are known

only after ALU2, so if the branch is taken, then the successor instructions are wasted, if the branch is not taken, then there is no stall at all.

For branch taken:

```
branch          IF RF ALU1 MEM ALU2 WB
successor1        IF   RF   ALU1 MEM ALU2 WB
successor2             IF   RF    ALU1 MEM ALU2 WB
successor3                  IF     RF   ALU1 MEM ALU2 WB
successor4                        IF     RF   ALU1 MEM ALU2 WB
target                                 IF     RF   ALU1 ALU2 MEM WB
```

successor1 to 4 are wasted, the length of stall is 4 cycles.

For branch not taken:

```
branch          IF RF ALU1 MEM ALU2 WB
successor1        IF   RF   ALU1 MEM ALU2 WB
successor2             IF   RF    ALU1 MEM ALU2 WB
successor3                  IF     RF   ALU1 MEM ALU2 WB
successor4                        IF     RF   ALU1 MEM ALU2 WB
successor5                              IF     RF   ALU1 ALU2 MEM WB
```

There is no stall.

3. a. For 3-stage pipeline, the dependence is 1 cycle stall, the probability is 1/p; for 4-stage pipeline, the dependence between instruction i and i+1 is 2 cycle stall, the probability is 1/p, between i and i+2 is 1 cycle stall, the probability is 1/p2, then the average execution time per instruction for 3-stage pipeline is (1+1/p)*T, for 4-stage pipeline is (1+2/p+1/p2)*(T-d), to make it a profitable change, there is (1+1/p)*T>=(1+2/p+1/p2)*(T-d), then we get d>= T/(1+p), so the lower bound of d is **T/(1+p).**

   b. When forwarding is implemented, there is no data hazard, just consider control hazard. For 3-stage pipeline, taken branches have 2 cycle stall, not-taken branches have 1 cycle stall; for 4-stage pipeline, taken branches have 3 cycle stall, not-taken branches have 2 cycle stall. The frequency for taken branch is 60% of conditional branch, and not-taken branch is 40% of that. Let x be the percentage of conditional branches in the program. Then the average execution time per instruction for 3-stage pipeline is T*(1+2*60%*x+1*40%*x), for 4-stage pipeline is (T-d)(1+3*60%*x+2*40%*x), to make it better performance, there is T*(1+2*60%*x+1*40%*x)>=(T-d)(1+3*60%*x+2*40%*x), we get x<=d/(T- 6*d), let r=d/T, then the upper bound of x is r/(1-2.6*r). When r=10%, the maximum percentage is **13.51%.**

4. History file keeps track of the original values of registers. When an exception occurs and the state must be rolled back earlier than some instruction that completed out of order, the original value of the register can be restored from the history file.
   Future file keeps the newer value of a register, when all earlier instructions have completed, the main register file is updated from the future file. On an exception, the main register file has the precise values for the interrupted state.
   When choosing from them, we should know the complexity of implementation of each method, and the requirement of hardware; we also should know the frequency of exception, and implementation time difference of instructions. If frequency high or the difference is large, future file is better, since history file need to keep more results and the cost to roll back is high We can compute the frequency of exception, and the time to handle to quantify the cost of each method.
   Future file needs more register, since it needs to keep track of newer values of register, when the value is updated during the implementation of the instruction, but the control logic is less since the main register file need the new value all the time. while history file need less register, but the roll back logic is more complicated.

5. For the first method, when exception occurs, the exception instruction drain all the instruction following it, and wait until all the instruction preceding it finishes, then it restarts from an empty pipeline. Every 500 instructions, there is an exception. When exception occurs at IF, there is 4 cycle delay; for ID, it is 3 cycle; for EX, it is 2 cycle; for MEM, it is 1 cycle. Then the average delay is (60%*4+5%*3+10%*2+25%*1)/500= 3/500, then the execution time is IC*(1.2+3/500)/(500*106)=**0.0024*IC**.

For the second method, the pipeline keeps full, the exception instruction start one more time. So the CPI remains the same, while the IC changes. The execution time is (1.2*501*IC/500)/(475*106)=**0.0025*IC**.

From the calculation above, we get that the **first method results in a faster CPU**.

6. a. If the ALU stage is split into two cycles, and all possible forwarding paths are implemented, there are three additional data hazards, all are RAW hazards.
ALU2/MEM ID/ALU1, when instructioni+1 needs the ALU result of i;
MEM/WB ID/ALU1, when instructioni+1 needs the memory reference result of i;
WB ID , when instructioni+2 needs the result of i and i+1.

b. To estimate the penalty bound, we need to know the frequency of instructions, about 27% instructions are ALU, suppose half time is followed by such instruction, then it introduce 0.13 penalty, about 26% are loads, then it introduce 0.26 penalty.we can suppose that each instruction is dependent on the former one, then each causes 2 cycle stall, so the maximum penalty is 2 cycles stall.

7. If each stage is split into two stages that are half as long, the benefit is that the faster clock rate, and higher throughput. But the performance improvement gained by deeper pipeline stages are limited by some factors. And there are some drawbacks.

First, it will not improve the execution time for a specific program P. The original execution time for P = IC * CPIold * Clock Cycle Time, the new execution time for P = IC * CPI new * 0.5 Clock Cycle Time, in order to have improvement, we mush have 2 * CPIold > CPI new. But in fact, since the pipeline is twice as deep, if there is no any stalls, 2 * CPIold will be equal to CPInew, and if have stalls, the pipeline stall cycles per instruction maybe increased or decreased, but no matter which one is larger (2 * CPIold vs. CPInew), their difference will be not very larger. But we spend more hardware.

Also, it will make the pipeline design more difficult and complex and improve the cost to built a pipeline, but finally just get a little improvement or even no improvement. So, consider the tradeoff, it is not worthwhile to have such arrangement.