Computer Architectures
2nd part labs – lab 1

1) Assembly program (program_1.s)

```
.data
vett: .word 1,200,3,4,5,6,7,8,9,10
result: .word 0
.code
        daddui r3, r0, 0        ; index
        daddui r1, r0, 10       ; count loop
        daddui r2, r0, 0        ; max
loop:   ld r4, vett(r3)         ; load a value from vett
        slt r5, r2, r4          ; if (r2 < r4) r5=1
        beqz r5, lower          ; if (r5 == 0) (r2 >= r4) (current max not defeated) skip
        daddui r2, r4, 0        ; update max
lower:  daddui r3, r3, 8        ; update index
        daddi r1, r1, -1        ; update counter
        bnez r1, loop           ; loop

        sd r2, result(r0)       ; save max
        halt
```

2) Execution

- All features disabled

```
        daddui r3, r0, 0      ;5 FDEMW
        daddui r1, r0, 10     ;1  FDEMW
        daddui r2, r0, 0      ;1   FDEMW

loop:   ld r4, vett(r3)       ;1    FDEMW
        slt r5, r2, r4        ;3     FD--EMW
        beqz r5, lower        ;3      F--D--EMW
        daddui r2, r4, 0      ;1         F--DEMW
lower:  daddui r3, r3, 8      ;1            FDEMW
        daddi r1, r1, -1      ;1             FDEMW
        bnez r1, loop         ;3              FD--EMW

        sd r2, result(r0)     ;1                F--DEMW
        halt                  ;1
```

Total clock cycles: 7 + 14*10 + 1 = 148

- Forwarding enabled

```
        daddui r3, r0, 0      ;5 FDEMW
        daddui r1, r0, 10     ;1  FDEMW
        daddui r2, r0, 0      ;1   FDEMW

loop:   ld r4, vett(r3)       ;1    FDEMW
        slt r5, r2, r4        ;2     FD-EMW
        beqz r5, lower        ;2      F-D-EMW
        daddui r2, r4, 0      ;1        F-DEMW
lower:  daddui r3, r3, 8      ;1          FDEMW
        daddi r1, r1, -1      ;1           FDEMW
        bnez r1, loop         ;2            FD-EMW

        sd r2, result(r0)     ;1              F-DEMW
        halt                  ;1
```

Total clock cycles: 7 + 11*10 + 1 = 118

- Forwarding and Branch Target Buffer enabled

Instructions beqz and bnez without BTB have a penalty of 1 CC in case the branch is taken, 0 otherwise. With BTB, penalties are the following:

Instruction in buffer	Prediction	Actual branch	Penalty
Yes	Taken	Taken	0
Yes	Taken	Not taken	2
No		Taken	2
No		Not taken	0

With the input data (1, 200, 3, 4, 5, 6, 7, 8, 9, 10) the stalls counts are the following (first row due to other stalls, second row caused by BTB penalties):

Instruction	1	2	3	4	5	6	7	8	9	10
beqz	1 (raw) 0 NT	1 (raw) 0 NT	1 (raw) 2 T	1 (raw) 0 T	1 (raw) 0 T	1 (raw) 0 T	1 (raw) 0 T	1 (raw) 0 T	1 (raw) 0 T	1 (raw) 0 T
bnez	1 (raw) 2 T	1 (raw) 0 T	1 (raw) 0 T	1 (raw) 0 T	1 (raw) 0 T	1 (raw) 0 T	1 (raw) 0 T	1 (raw) 0 T	1 (raw) 0 T	1 (raw) 2 NT

Total clock cycles: 118 (without BTB) – 8 (beqz Taken no BTB) + 1 (beqz BTB) - 9 (bnez Taken no BTB)+ 1 (bnez Taken) + 2 (bnez Not Taken) = 105 CC

- Forwarding and Delay Slot

Enabling delay slot always causes the execution of the instruction after branch. To get the correct final result the code has to be changed. After the `beqz` instruction I need to place an instruction that can always be executed: the only independent instruction is `daddui r3, r3, 8`, and after the `bnez` I have no independent instructions: I will let the `store` to happen every time: it won't affect final result, but is time wasting.

```
        daddui r3, r0, 0      ;5 FDEMW
        daddui r1, r0, 10     ;1  FDEMW
        daddui r2, r0, 0      ;1   FDEMW

loop:   ld r4, vett(r3)       ;1    FDEMW
        slt r5, r2, r4        ;2     FD-EMW
        beqz r5, lower        ;2      F-D-EMW
        daddui r3, r3, 8      ;1        F-DEMW
        daddui r2, r4, 0      ;1          FDEMW
lower:  daddi r1, r1, -1      ;1           FDEMW
        bnez r1, loop         ;2            FD-EMW

        sd r2, result(r0)     ;1             F-DEMW
        halt                  ;1
```

For the count of clock cycles I need to consider that if beqz branch is taken (8 times), the second daddui is not executed. Clock cycles: 7 + 10*8 (beqz taken) + 11*2 (beqz not taken) + 1 = 110 CC.

3) Program 2

```
.data
v1: .double 1,2,3,4,5,6,7,8,9,10
    .double 1,2,3,4,5,6,7,8,9,10
    .double 1,2,3,4,5,6,7,8,9,10
    .double 1,2,3,4,5,6,7,8,9,10
v2: .double 1,2,3,4,5,6,7,8,9,10
    .double 1,2,3,4,5,6,7,8,9,10
    .double 1,2,3,4,5,6,7,8,9,10
    .double 1,2,3,4,5,6,7,8,9,10
```

```
v3: .double 1,2,3,4,5,6,7,8,9,10
    .double 1,2,3,4,5,6,7,8,9,10
    .double 1,2,3,4,5,6,7,8,9,10
    .double 1,2,3,4,5,6,7,8,9,10
v4: .double 1,2,3,4,5,6,7,8,9,10
    .double 1,2,3,4,5,6,7,8,9,10
    .double 1,2,3,4,5,6,7,8,9,10
    .double 1,2,3,4,5,6,7,8,9,10
v5: .double 0,0,0,0,0,0,0,0,0,0
    .double 0,0,0,0,0,0,0,0,0,0
    .double 0,0,0,0,0,0,0,0,0,0
    .double 0,0,0,0,0,0,0,0,0,0
v6: .double 0,0,0,0,0,0,0,0,0,0
    .double 0,0,0,0,0,0,0,0,0,0
    .double 0,0,0,0,0,0,0,0,0,0
    .double 0,0,0,0,0,0,0,0,0,0
v7: .double 0,0,0,0,0,0,0,0,0,0
    .double 0,0,0,0,0,0,0,0,0,0
    .double 0,0,0,0,0,0,0,0,0,0
    .double 0,0,0,0,0,0,0,0,0,0
.code
        daddui r1, r0, 0
        daddui r20, r0, 40
loop:   l.d f1, v1(r1)
        l.d f2, v2(r1)
        l.d f3, v3(r1)
        l.d f4, v4(r1)
        mul.d f5, f1, f2
        div.d f6, f2, f3
        add.d f7, f4, f1
        s.d f5, v5(r1)
        s.d f6, v6(r1)
        s.d f7, v7(r1)
        daddui r1, r1, 8
        daddi r20, r20, -1
        bnez r20, loop
        halt
```

Vectors can't have 100 values. To run this code (40 elements for vector) I changed MIPS data address bus to 12 bits.

4) Clock cycles calculation

```
     daddui r1, r0, 0      ;5 FDEMW
        daddui r20, r0, 40    ;1  FDEMW
loop:   l.d f1, v1(r1)        ;1   FDEMW
        l.d f2, v2(r1)        ;1    FDEMW
        l.d f3, v3(r1)        ;1     FDEMW
        l.d f4, v4(r1)        ;1      FDEMW
        mul.d f5, f1, f2      ;8       FDxxxxxxxxMW
        div.d f6, f2, f3      ;5        FDddddddddddddMW
        add.d f7, f4, f1      ;0         FDaaaaMW
        s.d f5, v5(r1)        ;0          FDEssssSMW
        s.d f6, v6(r1)        ;1           FDssssssEsssSMW
        s.d f7, v7(r1)        ;1            FssssSDssssEMW
        daddui r1, r1, 8      ;1                  FssssDEMW
        daddi r20, r20, -1    ;1                       FDEMW
        bnez r20, loop        ;2                        FsDEMW
        halt                  ;1                          FDEMW
```

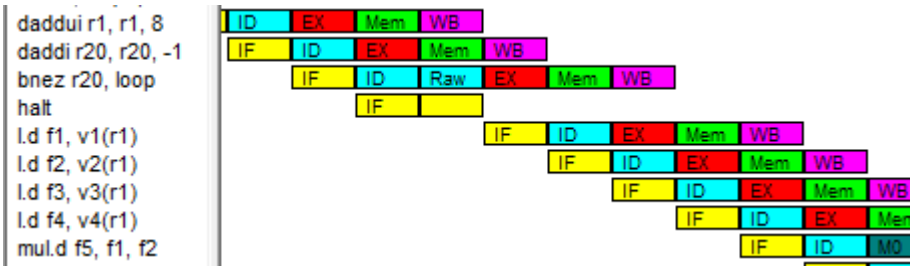
6 + 24*40 = 966

5) winMIPS64 verification

```
Execution

966 Cycles
523 Instructions
1.847 Cycles Per Instruction (CPI)


Stalls

280 RAW Stalls
0 WAW Stalls
0 WAR Stalls
120 Structural Stalls
39 Branch Taken Stalls
0 Branch Misprediction Stalls


Code size

64 Bytes
```

6) Comparation

Same values. But MIPS emulator puts bnez stall after decode instead of before.



7) Branch delay slot and static scheduling

```
        daddui r1, r0, 0      ;5 FDEMW
        daddui r20, r0, 40    ;1  FDEMW
loop:   l.d f2, v2(r1)        ;1   FDEMW
        l.d f3, v3(r1)        ;1    FDEMW
        l.d f1, v1(r1)        ;1     FDEMW
        div.d f6, f2, f3      ;12     FDddddddddddddMW
        l.d f4, v4(r1)        ;0       FDEMW
        mul.d f5, f1, f2      ;0        FDxxxxxxxxMW
        add.d f7, f4, f1      ;0         FDaaaaMW
        daddi r20, r20, -1    ;0          FDEMW
        s.d f7, v7(r1)        ;0           FDEsSMW
        s.d f5, v5(r1)        ;0            FDssEsSMW
        s.d f6, v6(r1)        ;1             FssDssESMW
        bnez r20, loop        ;1                FssDsEMW
        daddui r1, r1, 8      ;1                   FsDEMW
        halt                  ;1                     FDEMW
```

Clock cycles calculation: 6+18*39 + 19 = 727 instead of 966

```
Execution

727 Cycles
523 Instructions
1.390 Cycles Per Instruction (CPI)


Stalls

80 RAW Stalls
0 WAW Stalls
0 WAR Stalls
120 Structural Stalls
0 Branch Taken Stalls
0 Branch Misprediction Stalls


Code size

64 Bytes
```

8) Loop unrolling and register renaming

The v3 of this program needs some stalls to wait the results of the floating point units. In this new version I anticipated some load of the following loop cycle to use those wasted clocks.

Register renaming table:

Previous name	L1 name	L2 name	L3 name	L4 name
F1	F1	F8	F15	F22
F2	F2	F9	F16	F23
F3	F3	F10	F17	F24
F4	F4	F11	F18	F25
F5	F5	F12	F19	F26
F6	F6	F13	F20	F27
F7	F7	F14	F21	F28

```
.code
        daddui r1, r0, 0        ;  5 FDEMW
        daddui r20, r0, 40      ;  1  FDEMW
loop:   l.d f2, v2(r1)          ;1 1   FDEMW
        l.d f3, v3(r1)          ;1 1    FDEMW
        l.d f1, v1(r1)          ;1 1     FDEMW
        div.d f6, f2, f3        ;1 12     FDddddddddddddMW
        l.d f4, v4(r1)          ;1 0       FDEMW
        mul.d f5, f1, f2        ;1 0        FDxxxxxxxxMW
        add.d f7, f4, f1        ;1 0         FDaaaaMW
        l.d f9, v2+8(r1)        ;2 0          FDEMW
        l.d f10, v3+8(r1)       ;2 0           FDEMW
        s.d f7, v7(r1)          ;1 0            FDESMW
        l.d f8, v1+8(r1)        ;2 0             FDsEMW
        s.d f5, v5(r1)          ;1 0              FsDESMW
        s.d f6, v6(r1)          ;1 1                FDsESMW
        div.d f13, f9, f10      ;2 11                FsDddddddddddddMW
        l.d f11, v4+8(r1)       ;2 0                   FDEMW
        mul.d f12, f8, f9       ;2 0                    FDxxxxxxxxMW
        add.d f14, f11, f8      ;2 0                     FDaaaaMW
        l.d f16, v2+16(r1)      ;3 0                      FDEMW
        l.d f17, v3+16(r1)      ;3 0                       FDEMW
        s.d f14, v7+8(r1)       ;2 0                        FDESMW
        l.d f15, v1+16(r1)      ;3 0                         FDsEMW
        s.d f12, v5+8(r1)       ;2 0                          FsDESMW
```

```
s.d f13, v6+8(r1)      ;2 1
div.d f20, f16, f17    ;3 11
l.d f18, v4+16(r1)     ;3 0
mul.d f19, f15, f16    ;3 0
add.d f21, f18, f15    ;3 0
l.d f23, v2+24(r1)     ;4 0
l.d f24, v3+24(r1)     ;4 0
s.d f21, v7+16(r1)     ;3 0
l.d f22, v1+24(r1)     ;4 0
s.d f19, v5+16(r1)     ;3 0
s.d f20, v6+16(r1)     ;3 1
div.d f27, f23, f24    ;4 11
l.d f25, v4+24(r1)     ;4 0
mul.d f26, f22, f23    ;4 0
add.d f28, f25, f22    ;4 0
daddi r20, r20, -4     ;4 0
s.d f28, v7+24(r1)     ;4 0
s.d f26, v5+24(r1)     ;4 0
s.d f27, v6+24(r1)     ;4 1
bnez r20, loop         ;  1
daddui r1, r1, 32      ;  1
halt                   ;  1
```

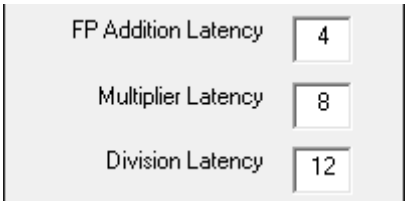
```
FDsESMW
 FsDddddddddddddMW
   FDEMW
    FDxxxxxxxxMW
     FDaaaaMW
      FDEMW
       FDEMW
        FDESMW
         FDsEMW
          FsDESMW
            FDsESMW
             FsDddddddddddddMW
               FDEMW
                FDxxxxxxxxMW
                 FDaaaaMW
                  FDEMW
                   FDEsSMW
                    FDssEsSMW
                     FssDssESMW
                        FssDsEMW
                           FsDEMW
                             FDEMW
```

Clock cycles calculation: 6 + 54*9 + 55 = 547 instead of 727

```
Execution

547 Cycles
433 Instructions
1.263 Cycles Per Instruction (CPI)


Stalls

20 RAW Stalls
0 WAW Stalls
0 WAR Stalls
120 Structural Stalls
0 Branch Taken Stalls
0 Branch Misprediction Stalls


Code size

184 Bytes
```

9) Amdahl's law verification

The program used is program_2.s. The starting processor architectural parameters are the following:



Execution time: 966 clock cycles. 523 instructions.

$$fraction_{FPdiv} = \frac{40}{523} \cong 0.0765$$

FP division latency	11	10
$ExecutionTime$	926	886
$speedup_{enhanced}$	12/11	12/10
$experimentalSpeedup$	$\frac{966}{926} = 1{,}043$	$\frac{966}{886} = 1.090$
$theoreticalSpeedup$	$\frac{1}{(1 - fraction_{enh}) + \frac{fraction_{enh}}{speedup_{enhanced}}} = 1{,}006$	$1{,}012$

The theoretical and experimental speedups are quite different. Maybe this program is too small.