EE 4720 **Homework 4** Solution **Due: 17 April 2000**

Problem 1: The diagram below shows the execution of code on a dynamically scheduled machine that uses physical register numbers to name destination operands. Show the state of the ID register map, the commit register map, their free lists, and the physical register file for each cycle of the execution below. In the register maps and file show only values related to registers `f0` and `f3`. Initially, `f0`=0, `f1`=10, `f2`=20, etc. Initially, register `f0` is assigned to physical register 12 and `f3` is assigned to physical register 15 (ignore the other architected registers). Initially, both free lists contain physical register numbers $\{7, 8, 9, 10, 11\}$.

Note: As originally assigned the initial free lists did not contain register 11 and the pipeline execution diagram showed reservation station (RS) segments. Both were mistakes and have been corrected.

```
! Cycle             0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17 18
multf f0, f1, f2    IF ID Q           M0 M1 M2 M3 M4 M5 WC
addf  f3, f0, f2       IF ID Q                              A0 A1 WC
subf  f0, f4, f5          IF ID Q  A0 A1 WB                        C
addf  f3, f0, f5             IF ID Q        A0 A1 WB                  C
addf  f0, f2, f1                IF ID Q        A0 A1 WB                  C
```

The solution appears below. Blank entries in the tables below indicate that the value has not changed. The free lists (shown in braces, or curly brackets) are for the cycle in which the opening brace appears. For example, in cycle 3 the ID free list is `10,11` and the completion free list is `7,8,9,10,11` (because there was no change since cycle 0). The row in which a free list appears is not significant, there is only one ID free list and one completion free list.

```
! Cycle            0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17 18
multf f0, f1, f2    IF ID Q           M0 M1 M2 M3 M4 M5 WC
addf  f3, f0, f2       IF ID Q                            A0 A1 WC
subf  f0, f4, f5          IF ID Q  A0 A1 WB                      C
addf  f3, f0, f5             IF ID Q        A0 A1 WB                C
addf  f0, f2, f1                IF ID Q        A0 A1 WB                C

! Cycle            0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17 18
! ID Register Map
!               f0  12 7     9     11
!               f3  15    8     10
! ID Free List
!                   {7,8,9,10,11}  {}                  {12}     {12,15}
!                      {8,9,10,11}                                {12,15,7}
!                         {9,10,11}                                  {12,15,7,8}
!                            {10,11}                                    {12,15,7,8,9}
!                               {11}
! Commit Register Map
!               f0  12                                 7           9     11
!               f3  15                                                 8     10
! Commit Free List
!                   {7,8,9,10,11}                      {8,9,10,11,12}
!                                                                 {9,10,11,12,15}
!                                                                    {10,11,12,15,7}
!                                                                       {11,12,15,7,8}
!                                                                          {12,15,7,8,9}
!
! Cycle            0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17 18
!
! Physical Register File
!               7                                      200
!               8                                                220
!               9                        -10
!               10                                40
!               11                                   30
!               12  0
!               13
!               14
!               15  30
! Cycle            0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17 18
```

Problem 2: Repeat the problem above assuming that there is an exception in stage A1 of the execution of `addf f3, f0, f5`, as shown below: The solution can start at the cycle in which the tables will differ from the solution above.

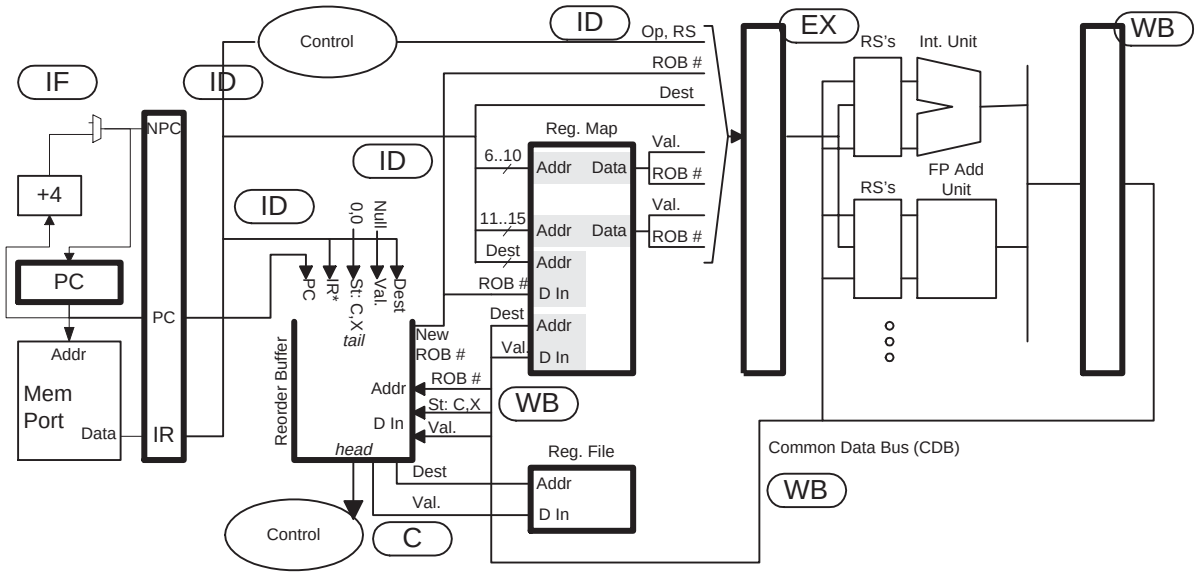
The solution appears below. The exception is not handled until the instruction reaches completion, at cycle 17. (So the solution below is identical to the one above up to cycle 17.) At cycle 17 the controller recovers from the exception by copying the completion map and completion free list to the ID map and free list. The diagram below shows this recovery being done in one cycle, but real system might take longer. Because the add encountered an exception the value it writes into the register file may not be valid, that is indicated by question marks.

```
! Cycle             0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17 18
multf f0, f1, f2    IF ID Q           M0 M1 M2 M3 M4 M5 WC
addf  f3, f0, f2       IF ID Q                              A0 A1 WC
subf  f0, f4, f5          IF ID Q  A0 A1 WB                        C
addf  f3, f0, f5             IF ID Q        A0*A1*WB                  Cx
addf  f0, f2, f1                IF ID Q        A0 A1 WB

! Cycle             0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17 18
! ID Register Map
!               f0  12 7     9     11                                  9
!               f3  15    8     10                                     8
! ID Free List
!                   {7,8,9,10,11}  {}                  {12}     {12,15}
!                      {8,9,10,11}                                {12,15,7}
!                         {9,10,11}                                  {10,11,12,15,7}
!                            {10,11}
!                               {11}
! Commit Register Map
!               f0  12                                 7           9
!               f3  15                                             8
! C Free List
!                   {7,8,9,10,11}                      {8,9,10,11,12}
!                                                               {9,10,11,12,15}
!                                                                  {10,11,12,15,7}
!
! Cycle             0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17 18
!
! Physical Register File
!               7                                      200
!               8                                               220
!               9                        -10
!               10                                ?40?
!               11                                   30
!               12  0
!               13
!               14
!               15  30
! Cycle             0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17 18
```

Problem 3: The diagram below, of a dynamically scheduled processor, omits hardware that checks whether the register map should be updated in the WB stage. (The hardware was described in class.) Add the hardware to the diagram (at the same level of detail as other parts of the diagram).

Solution diagram not yet available.



Problem 4: Draw a pipeline execution diagram for the DLX code below running on a dynamically scheduled 4-way superscalar implementation with the following characteristics:

- Dynamically scheduled using a reorder buffer to name registers (method 1).
- One load/store functional unit with stages L1 and L2.
- No dynamic (hardware) branch prediction, all branches are predicted not taken. Branch predictor uses the `B` functional unit and must wait for its operand like any other instruction.
- Four integer execution units.

Find the IPC for an execution of a large number of iterations. Show the execution for 14 cycles or until there is enough information to compute the IPC, which ever is shorter.

```
! Note: runs for many iterations.
 add  r3, r0, r0
LOOP:! LOOP = 0x1000
 lw   r1, 4(r2)
 add  r3, r3, r1
 lw   r2, 8(r2)
 bneq r2, LOOP
 xor  r0, r0, r0
```

The pipeline execution diagram is shown below. The misprediction is detected in cycle 7 and the correct path is fetched in cycle 8. The `xor` and following instructions get squashed (or flushed from the reorder buffer). Since the iteration that starts at cycle 8 will take the same number of cycles as the one that starts at cycle 1 the IPC is $\frac{4}{7} \approx 0.571$.

```
! Solution
! Cycle           0  1  2  3  4  5  6  7  8
 add  r3, r0, r0  IF ID EX WC
LOOP: ! LOOP = 0x1000
 lw   r1, 4(r2)      IF ID L1 L2 WC       IF ...
 add  r3, r3, r1     IF ID RS RS EX WC    IF ...
 lw   r2, 8(r2)      IF ID RS L1 L2 WC    IF ...
 bneq r2, LOOP       IF ID RS RS RS B  WC
                                          IF ...
 xor  r0, r0, r0        IF ID EX WB    x
```

Problem 5: Repeat the problem above when the branch is statically predicted as taken and the branch target is computed in the ID stage.

The pipeline execution diagram is shown below. Since the branch target is computed in ID the target instruction is fetched two cycles after the branch. (With a branch target buffer it would be fetched one cycle after the branch is fetched.) The hardware is able to fetch and decode instructions in this loop at the rate of 2 IPC, but the completion rate is lower due to dependencies between the loads. The second load must wait one cycle for the first load to move out of L1, as it does in cycle 3. The first load must wait for the second load from the previous iteration to enter WB, as it does in cycle 5. Because instructions are being fetched faster than they are begin committed some resource (such as reorder buffer slots or reservation stations) will be used up. When that happens (not shown below) instructions will stall in ID and fetch will drop to a rate of $\frac{4}{3}$ instructions per cycle. This is much faster than $\frac{4}{7}$ from the previous problem but still less than the 4 IPC that the processor is capable of.

```
! Solution
! Cycle           0  1  2  3  4  5  6  7  8  9  10 11 12 13
 add  r3, r0, r0  IF ID EX WC
LOOP: ! LOOP = 0x1000
 lw   r1, 4(r2)      IF ID L1 L2 WC
                        IF ID RS L1 L2 WC
                           IF ID RS RS L1 L2 WC
                              IF ID RS RS RS L1 L2 WC
! Cycle           0  1  2  3  4  5  6  7  8  9  10 11 12 13
 add  r3, r3, r1     IF ID RS RS EX WC
                        IF ID RS RS RS EX WC
                           IF ID RS RS RS RS EX WC
                              IF ID RS RS RS RS RS EX WC
 lw   r2, 8(r2)      IF ID RS L1 L2 WC
                        IF ID RS RS L1 L2 WC
                           IF ID RS RS RS L1 L2 WC
                              IF ID RS RS RS RS L1 L2 WC
 bneq r2, LOOP       IF ID RS RS RS B  WC
                        IF ID RS RS RS RS B  WC
                           IF ID RS RS RS RS RS B  WC
                              IF ID RS RS RS RS RS RS B  WC
 xor  r0, r0, r0        IF x  IF x  IF x  IF x
! Cycle           0  1  2  3  4  5  6  7  8  9  10 11 12 13
```

Problem 6: Repeat the superscalar problem when the branch is statically predicted taken and in which the address of LOOP it 0x1004.

```
! Note: runs for many iterations.
 add  r3, r0, r0
LOOP:! LOOP = 0x1004
 lw   r1, 4(r2)
 add  r3, r3, r1
 lw   r2, 8(r2)
 bneq r2, LOOP
 xor  r0, r0, r0
```

The pipeline execution diagram is shown below. Because of alignment the instructions for one iteration are fetched in two groups. (In the previous example the four instructions in an iteration neatly fit on one group.) This adds an extra cycle, so instructions are fetched at a rate of $\frac{4}{3}$ IPC, which is the same rate at which they are executed. So, even though instructions are fetched at a lower rate execution occurs at the same rate because of dependencies.

```
! Solution
! Cycle           0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15
 add  r3, r0, r0  IF ID EX WC
LOOP: ! LOOP = 0x1004
 lw   r1, 4(r2)   IF ID L1 L2 WC
                           IF ID L1 L2 WC
                                    IF ID L1 L2 WC
                                             IF ID L1 L2 WC
 add  r3, r3, r1  IF ID RS RS EX WC
                           IF ID RS RS EX WC
                                    IF ID RS RS EX WC
                                             IF ID RS RS EX WC
 lw   r2, 8(r2)   IF ID RS L1 L2 WC
                           IF ID RS L1 L2 WC
                                    IF ID RS L1 L2 WC
                                             IF ID RS L1 L2 WC
 bneq r2, LOOP       IF ID RS RS B  WC
                              IF ID RS RS B  WC
                                       IF ID RS RS B  WC
                                                IF ID RS RS B  WC
 xor  r0, r0, r0     IF IDx   IF IDx   IF IDx   IF IDx   IF IDx   IF IDx
! Cycle           0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15
```