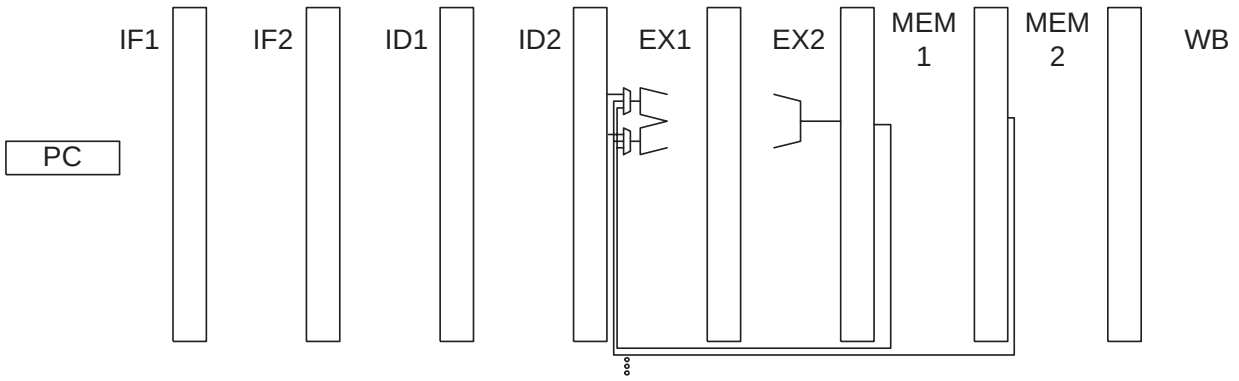
EE 4720 **Homework 4** Solution **Due: 22 April 2002**

To solve Problem 3 and the next assignment a paper has to be read. Do not leave the reading to the last minute, however try attempting the first problem below before reading the paper.

Problem 1: The pipeline below was derived from the five-stage statically scheduled MIPS implementation by splitting each stage (except writeback) into two stages. Each pair of stages (say `IF1` and `IF2`) does the same thing as the original stage (say `IF`), but because it is broken in to two stages it takes two rather than one clock cycle. The diagram shows only a few details. Bypass connections into the ALU are available from all stages from `MEM1` to `WB`.



The advantage of this pipeline is that the clock frequency can be doubled. (Actually not quite times two.) Perfect execution is shown in the diagram below:

```
add $1, $2, $3  IF1 IF2 ID1 ID2 EX1 EX2 ME1 ME2 WB
sub $4, $5, $6      IF1 IF2 ID1 ID2 EX1 EX2 ME1 ME2 WB
and $7, $8, $9          IF1 IF2 ID1 ID2 EX1 EX2 ME1 ME2 WB
```

(*a*) Suppose the old five-stage system ran at a clock frequency of 1 GHz and the new system runs at 2 GHz. How does the execution time compare on the new system when execution is perfect?

It's half! That is, performance scaled with clock frequency.

(*b*) Show a pipeline execution diagram of the code below on the new pipeline. Note dependencies through registers `$10` and `$11`.

```
add $10, $2, $3    IF1 IF2 ID1 ID2 EX1 EX2 ME1 ME2 WB
sub $4, $10, $6        IF1 IF2 ID1 ID2 --> EX1 EX2 ME1 ME2 WB
and $11, $8, $9            IF1 IF2 ID1 --> ID2 EX1 EX2 ME1 ME2 WB
or  $20, $21, $22              IF1 IF2 --> ID1 ID2 EX1 EX2 ME1 ME2 WB
xor $7, $11, $0                    IF1 --> IF2 ID1 ID2 EX1 EX2 ME1 ME2 WB
```

(*c*) In the previous part there should be a stall on the new pipeline that does not occur on the original pipeline. (It's not too late to change your answer!) How does that affect the usefulness of splitting pipeline stages?

Assuming not all adjacent instructions are truely dependent, splitting is still useful, but performance does not scale with clock frequency. (It never does.)

(*d*) (Optional, complete before reading paper.) To get that I'm-so-clever feeling answer the following: Suppose there is no way a 32-bit add can be completed in less than two cycles. Is there any way to perform addition so that results can be bypassed to an immediately following instruction, as

in the example above, but without stalling? The technique must work when adding any two 32-bit numbers. *Hint: The adder would have to be redesigned.* (A question in the next homework assignment revisits the issue.)

Split the ALU in to sixteen-bit parts and bypass the low and high parts separately.

Problem 2: *Note: The following problem is similar to one given in the Fall 2001 semester, see* `http://www.ece.lsu.edu/ee4720/2001f/hw03.pdf` *(the problem) and* `http://www.ece.lsu.edu/ee4720/2001f/hw03_sol.pdf` *(the solution). For best results do not look at the solutions until you're really stuck. This problem below uses MIPS instead of DLX and is for Method 3 instead of Method 1.* The code below executes on a dynamically scheduled four-way superscalar MIPS implementation using Method 3, physical register numbers.

- Loads and stores use the load/store unit, which consists of segments L1 and L2.
- The floating-point multiply unit is fully pipelined and consists of six segments, `M1` to `M6`.
- The usual number of instructions (for a 4-way superscalar machine) can be fetched, decoded, and committed per cycle.
- An unlimited number of instructions can complete (but not commit) per cycle. (Not realistic, but it makes the solution easier.)
- There are an unlimited number of reservation stations, reorder buffer entries, and physical registers.
- The target of a branch is fetched in the cycle after the branch is in ID, *whether or not the branch condition is available.* (We'll cover that later.)

(*a*) Show a pipeline execution diagram for the code below until the beginning of the fourth iteration. Show where instructions commit.

See diagram below.

(*b*) What is the CPI for a large number of iterations? *Hint: There should be less than six cycles per iteration.*

The CPI is $\frac{3}{6} = 0.5$.

(*c*) Show the entries in the ID and commit register maps for registers `f0` and `$1` for each cycle in the first two iterations. If several values are assigned in the same cycle show each one separated by commas.

(*d*) Show the values in the physical register file for `f0` and `$1` for the first two iterations. Use a "]" to show when a physical register is removed from the free list and use a "[" to show when it is put back in the free list.

See pipeline execution diagram on the next page.

```
 ! Solution
LOOP:  ! Instructions shown in dynamic order. (Instructions repeated.)
 ! Cycle          0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16
 ldc1 f0, 0($1)   IF ID Q  L1 L2 WC
 mul.d f0, f0, f2 IF ID Q        M1 M2 M3 M4 M5 M6 WC
 sdc1 0($1), f0   IF ID Q  L1                     L2 WC
 addi $1, $1, 8   IF ID Q  EX WB                     C
 bne $2, $0 LOOP     IF ID Q  B  WB                  C
 sub $2, $1, $3      IF ID Q  EX WB                  C
 # Cycle          0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17
 ldc1 f0, 0($1)            IF ID Q  L1 L2 WB              C
 mul.d f0, f0, f2          IF ID Q        M1 M2 M3 M4 M5 M6 WC
 sdc1 0($1), f0            IF ID Q  L1                     L2 WC
 addi $1, $1, 8            IF ID Q  EX WB                     C
 bne $2, $0 LOOP              IF ID Q  B  WB                  C
 sub $2, $1, $3               IF ID Q  EX WB                  C
 # Cycle          0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17
 ldc1 f0, 0($1)                     IF ID Q  L1 L2 WB             C
 mul.d f0, f0, f2                   IF ID Q        M1 M2 M3 M4 M5 M6 WC
 sdc1 0($1), f0                     IF ID Q  L1                     L2 WC
 addi $1, $1, 8                     IF ID Q  EX WB                     C
 bne $2, $0 LOOP                       IF ID Q     B  WB                C
 sub $2, $1, $3                        IF ID Q  EX WB                   C
 # Cycle          0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17
 ldc1 f0, 0($1)                           IF ID Q  L1 L2 WB
 mul.d f0, f0, f2                         IF ID Q        M1 M2 M3 M4 M5
 ...
 # Cycle          0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17
 ID Map
 f0 99               97,96    94,93    91,90
 $1 98               95       92       89
 # In cycle one first 97 is assigned to f0, then 96 (replacing 97).  The
 # same sort of replacement occurs in cycles 4 and 7.

 # Cycle          0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17
 Commit Map
 f0 99                         97               96    94 93    91 90
 $1 98                                             95       92       89
 ! Cycle          0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17

 # Cycle          0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17
 Physical Register File
 99               1.0           ]
 98               0x1000                        ]
 97                 [           10              ]
 96                 [                           11    ]
 95                 [         0x1008                        ]
 94                           [          20               ]
 93                           [                           2.2   ]
 92                           [        0x1010                          ]
 ! Cycle          0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17
```

The following is an introduction to the next few problems.

As mentioned several times in class many of the performance-enhancing microarchitectural features that came in to wide use in the closing decades of the twentieth century (I love the way that sounds!) are much easier to apply to RISC ISAs than CISC ISAs. Bound by golden handcuffs to the CISCish IA-32 ISA, Intel was forced to get RISC-level performance from IA-32. (Not just Intel, DEC [now Compaq, perhaps soon HP] faced the problem with the VAX ISA and IBM with 360.) The solution chosen by Intel (and also DEC) was to translate individual IA-32 instructions in to one or more μops (micro-operations). Each μop is something like a RISC instruction and so the parts of the hardware beyond the IA-32 to μop translation can employ the same techniques used to implement RISC ISAs.

The paper at `http://www.intel.com/technology/itj/q12001/articles/art_2.htm` and `http://www.ece.lsu.edu/ee4720/s/hinton_p4.pdf` (password needed off campus, will be given in class) describes the Pentium 4 implementation of IA-32, including μops (which are typeset using "u" instead of the Greek letter "μ", except occasionally in figures). This paper was not written for a senior-level computer architecture class four weeks from the end of the semester and so it will include material which we have not yet covered (caches and TLBs) and some material not covered at all. Some stuff in the paper is not explained (how they do branch prediction or what the Pentium 4 pipeline segments in Figure 3 mean), some of this can be figured out other things have to be found out elsewhere (but not for this assignment).

Read the paper and answer the question below. The next homework assignment will include additional questions on the paper. For this initial reading skip or lightly read material on the L2 cache, L1 data cache, and the ITLB. Questions on the cache material might be asked in a later assignment.

Problem 3: What does the paper call the following actions and components (that is, translate from the terminology used in class to the terminology used in the paper):

Commit – Retire

ID Register Map – Frontend RAT

Commit Register Map – Retirement RAT

Physical Register File – Physical Register File