Name Solution

Computer Architecture
EE 4720
Final Examination

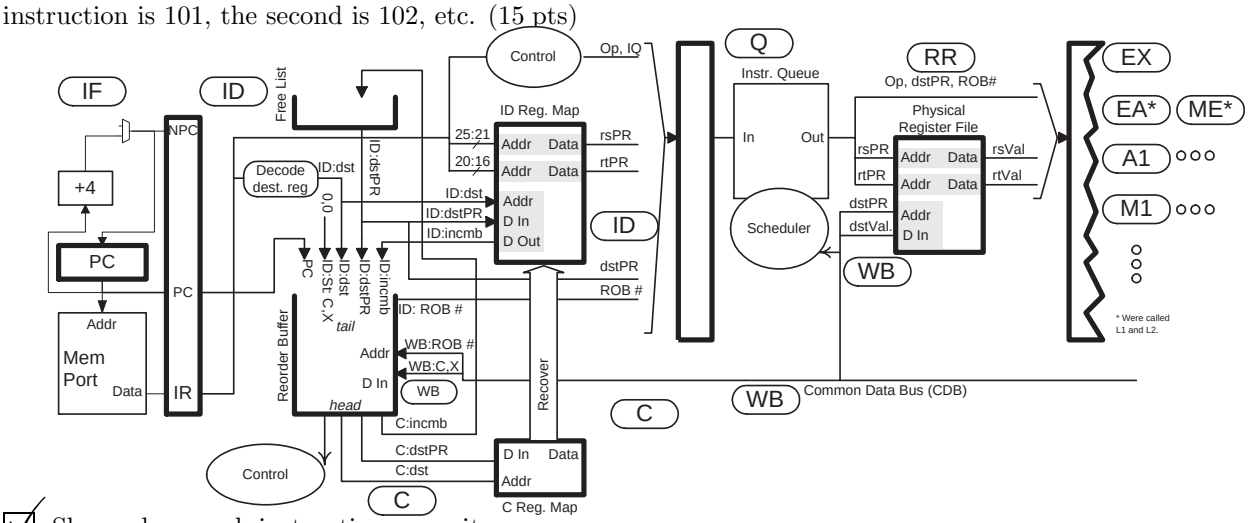
6 December 2004, 7:30–9:30 CST

Problem 1		(15 pts)
Problem 2		(25 pts)
Problem 3		(25 pts)
Problem 4		(10 pts)
Problem 5		(25 pts)
Exam Total		(100 pts)

Alias EE MV̄DCCXX

Good Luck!

Problem 1: The MIPS code below executes as shown on the illustrated dynamically scheduled scalar implementation. There are no exceptions or recoveries; the result (value to be written in `r2`) of the first instruction is 101, the second is 102, etc. (15 pts)



☑ Show where each instruction commits.
☑ The initial value of `r1` is 111 and the initial value of `r2` is 222. Fill in the tables to show these values.
☑ Complete the tables for the execution of the code. Take into account the results (see first paragraph).
☑ Show where registers are removed from and put back in the free list.

```
# Cycle              0   1   2   3   4   5   6   7   8   9   10  11  12
lw  r2, 0(r4)        IF  ID  Q   RR  EA  ME  WB  C
add r1, r2, r3           IF  ID  Q       RR  EX  WB  C
sw r1, 0(r4)                 IF  ID  Q   RR  EA      ME  WB  C
xor r2, r5, r3                   IF  ID  Q   RR  EX  WB          C
and r2, r2, r6                       IF  ID  Q       RR  EX  WB      C
# Cycle              0   1   2   3   4   5   6   7   8   9   10  11  12
# ID Map

Reg   Initial PR
r1  17                       66
r2  43                   63          83  88

# Cycle              0   1   2   3   4   5   6   7   8   9   10  11  12
# Commit Map

Reg   Initial PR
r1  17                                           66
r2  43                                       63              83  88

# Cycle              0   1   2   3   4   5   6   7   8   9   10  11  12
Physical Register File

PR    Initial State
17    111 r1                                         ]
43    222 r2                                     ]
63                       [r2                 101                 ]
66                           [r1                 102
83                                   [r2             104             ]
88                                       [r2                 105
# Cycle              0   1   2   3   4   5   6   7   8   9   10  11  12
```

The solution is shown on the previous page. This problem is uninteresting and so it would be a good first study problem for those reading this in the future.

Common Mistake: Allocating a register for `sw`. The `sw` instruction does not write a register and so it is not allocated a physical register.

Common Mistake: Freeing the wrong register. When an instruction commits the correct physical register to free is **not** the one assigned to the instruction but the incumbent, the previous one assigned to the same register. For example when the `add`, which was assigned physical register 66, commits it is physical register 17 which gets tossed back into the free list. The incumbent can be found in the commit (or ID) map to the left of the register assigned to the instruction.

Problem 2: An extended version of MIPS, called *MMMIPS*, includes memory-to-memory (MM) arithmetic instructions that can read the first source operand from memory and write a result to memory; the second source operand is always an immediate and the MM's are encoded in format I. Their mnemonics end with `.mm`, `.mr`, or `.rm` and they operate as described in the comments below. (25 pts)

```
lw     r1, 2(r3)        #  r1 = Mem[r3+2]  (Existing instruction, for your reference.)
add.mm (r4),(r5), 3     #  Mem[r4] = Mem[r5] + 3
sub.rm  r6, (r7), 3     #  r6 = Mem[r7] + 3
or.mr  (r8), r9,  3     #  Mem[r8] = r9 + 3
```

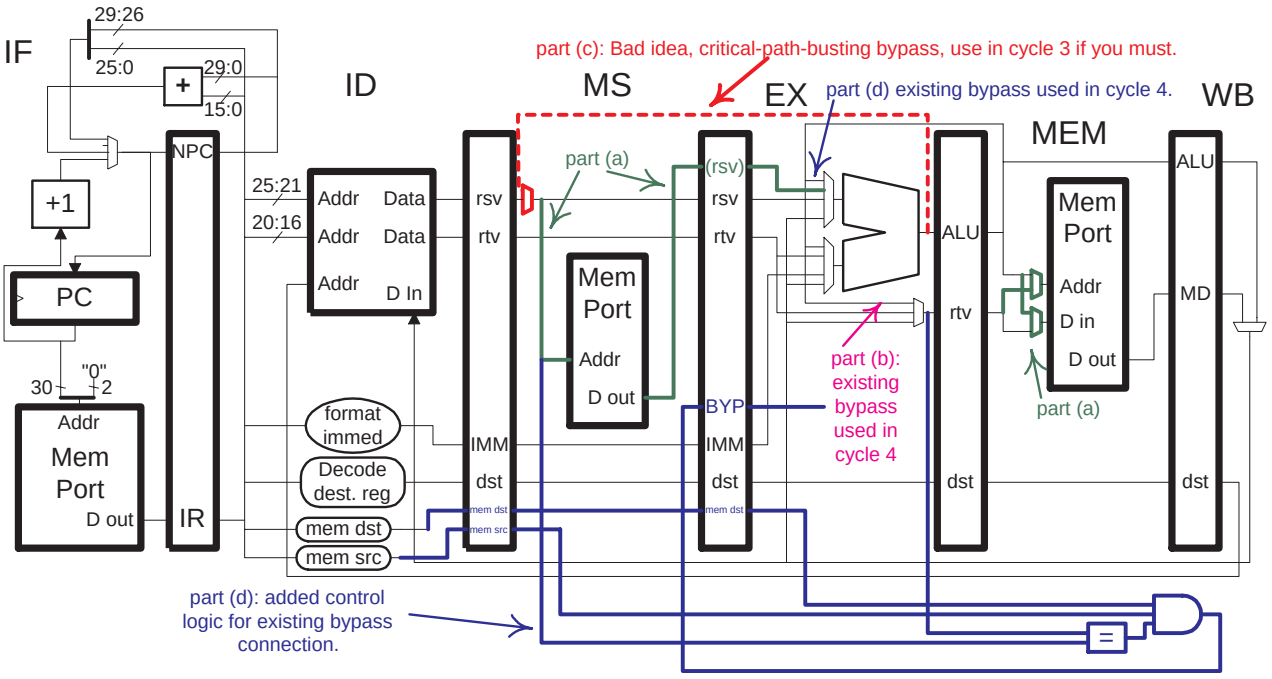
(*a*) Shown below is a partially completed implementation of MMMIPS. It includes a new stage, MS (the S is for source), that has a memory port for the source operand, shown unconnected. For use in a later part of this problem, boxes mem src and mem dst identify an instruction as having a memory source operand (output `1`) or a memory destination operand (output `1`) (an instruction can have both). An output of `0` means the respective operand is not from or to memory.

☑ Connect the MS-stage memory port for the MM instructions.

☑ Modify connections to the MEM-stage memory port for the MM instructions.

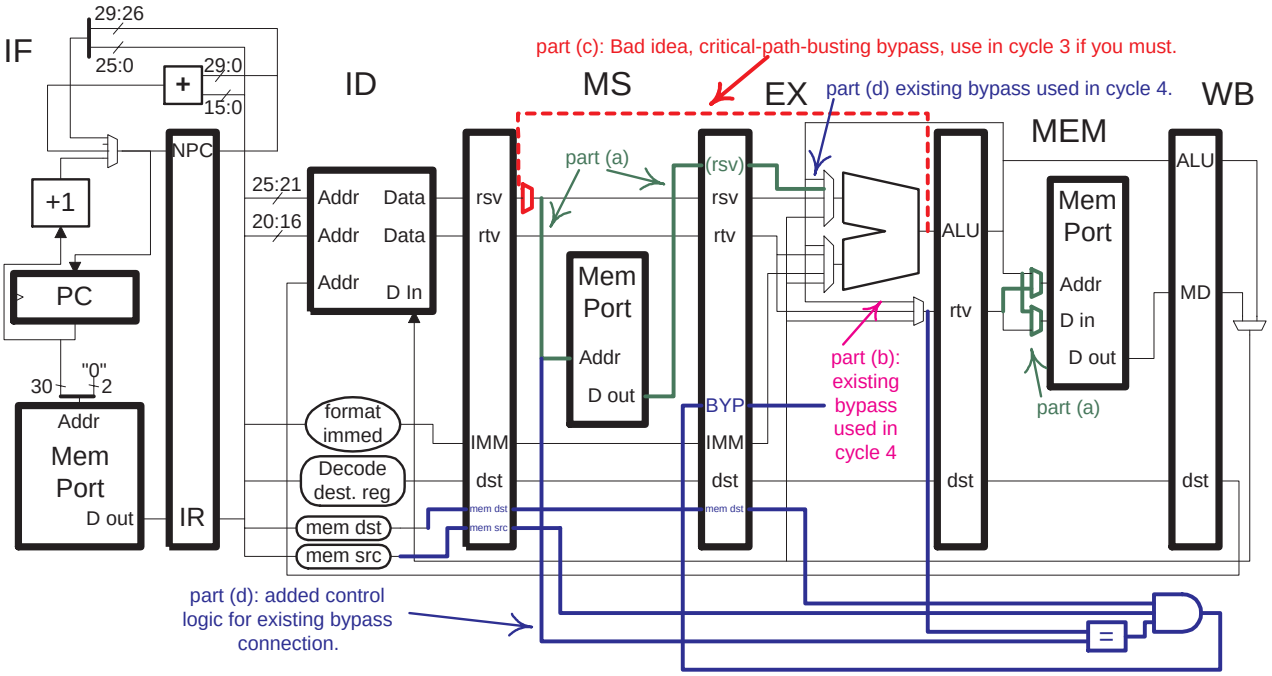
☑ Make sure existing MIPS load and store instructions continue to work correctly.

☑ For this part don't add bypassing or control logic.



Solution to this part appears above in green along with solution to other parts.

For the problems below use either the diagram below or the one on the previous page.



(*b*) Add a bypass connection so that the code below can execute without a stall or show which existing bypass connection can be used.

☑ Add or identify the connection.

☑ Show which cycle the bypass connection is used.

```
# Cycle              0  1  2  3  4  5  6
add     r1, r2, r3   IF ID MS EX ME WB
sub.mr (r1), r4, 5      IF ID MS EX ME WB
```

The value can use the exiting MEM-to-EX bypass connection, the bypassed value goes through the mux to EX/MEM.rtv. The value would be bypassed in cycle 4 for use in the memory write in cycle 5.

(*c*) Can a bypass connection be added for the code below? If yes, should it be added? Explain.

☑ Show how and/or explain why not.

```
# Cycle              0  1  2  3  4  5  6
add    r1, r2, r3    IF ID MS EX ME WB
sub.rm r6, (r1), 5      IF ID MS EX ME WB
```

Yes it can be added, but it shouldn't because of critical path impact. Added connection is shown above in red.

The subtract needs the address in the beginning of cycle 3, while it's in MS, but the value is computed at the end of cycle 3. If a bypass connection were provided then the cycle time would have to be long enough for both an add and a memory access, substantially lowering the clock frequency.

(*d*) Show the bypass connections (if any) and control logic so that the code below executes without a stall (and without causing other instructions to execute incorrectly, of course). The control logic should deliver a `BYPASS` signal to the EX stage at the right time, it should be `1` if a bypass of the type needed below is necessary. Do not connect it to anything.

☑ Show the control logic generating `BYPASS`.

☑ Add the bypass connection or show which existing one would be used.

```
# r1 = 0x1000, r5 = 0x1000
# Cycle              0  1  2  3  4  5  6
add.mr (r1), r2, 3   IF ID MS EX ME WB
sub.rm  r4, (r5), 6     IF ID MS EX ME WB
```

The bypass is only necessary if the two addresses are the same and so the control logic must compare the addresses as well as checking if the earlier instruction writes a memory location and the later one reads one. The control logic and existing bypass path are shown above in blue.

Problem 3: Answer each question below. Be sure to check each code fragment for dependencies. (25 pts)

(*a*) Show a pipeline execution diagram for the code fragment below running on the usual statically scheduled and bypassed scalar MIPS implementation. *Note: Bypassing and static scheduling not mentioned in the original exam.*

```
# Cycle         0    1    2    3    4    5   6   7   8

lw  r1, 0(r2)

add r3, r1, r4
```

```
# Solution:
# Cycle         0  1  2  3  4  5  6
lw  r1, 0(r2)   IF ID EX ME WB
add r3, r1, r4     IF ID -> EX ME WB
```

☑ Solve this easy problem.

Grading Note: The original test omitted the kind of implementation. Some did it on an unbypassed statically scheduled, a few did it on a dynamically scheduled system, and one on a superscalar statically scheduled implementation.

(*b*) The `sub.s` instruction below stalls in IF.

```
# Cycle           0  1  2  3  4  5  6  7  8  9  10 11
add.s f4, f5, f6  IF ID A1 A2 A3 A4 WF
sub.s f7, f4, f8     IF -------> ID A1 A2 A3 A4 WF
add.s f9, f10, f11                IF ID A1 A2 A3 A4 WF
```

☑ What's wrong about the stall?

☑ Show correct execution.

It's stalling because of the dependency through `f4`, but how could the implementation know that in cycle 1 to 4 if `sub.s` does not yet been decoded. The correct execution appears below.

```
# Solution
# Cycle           0  1  2  3  4  5  6  7  8  9  10 11
add.s f4, f5, f6  IF ID A1 A2 A3 A4 WF
sub.s f7, f4, f8     IF ID -------> A1 A2 A3 A4 WF
add.s f9, f10, f11      IF -------> ID A1 A2 A3 A4 WF
```

Problem 3, continued:

(*c*) The code below executes on a dynamically scheduled machine of the type used in the first problem and in class. The `mul.s` stalls in ID.

```
# ROB initially empty.
# Cycle           0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16
add.s f4, f5, f6  IF ID Q  RR A1 A2 A3 A4 WF C
sub.s f7, f4, f8     IF ID Q           RR A1 A2 A3 A4 WF C
mul.s f9, f10, f11      IF ID -------> Q  RR M1 M2 M3 M4 M5 M6 WF C
```

☑ What's wrong about the stall?

☑ Show correct execution.

This is a dynamically scheduled system and so `sub.s` waiting for an operand does not stall `mul.s`. Instruction `sub.s` moves out of ID and into the instruction Q, and so `mul.s` is not blocked.

☑ If the "`ROB initially empty.`" comment was not there then the execution above would be possible, albeit misleading. How would the stall be possible?

Dynamically scheduled systems stall when some resource is exhausted. The comment indicates that the three instructions have the entire processor to themselves and so it's unlikely that it has run out of anything. But without the comment there might be lots more instructions present and so the processor might have run out of something, for example, ROB slots or physical registers. If it runs out it must stall the instruction in ID. Unlike waiting in the instruction queue, this is a real stall of the fetch/decode pipeline that will block later instructions.

(*d*) Show a pipeline execution diagram for the code below executing on a 2-way statically scheduled superscalar MIPS implementation. *Note: In the original exam "statically scheduled" was omitted so an answer for either type of system would be correct.*

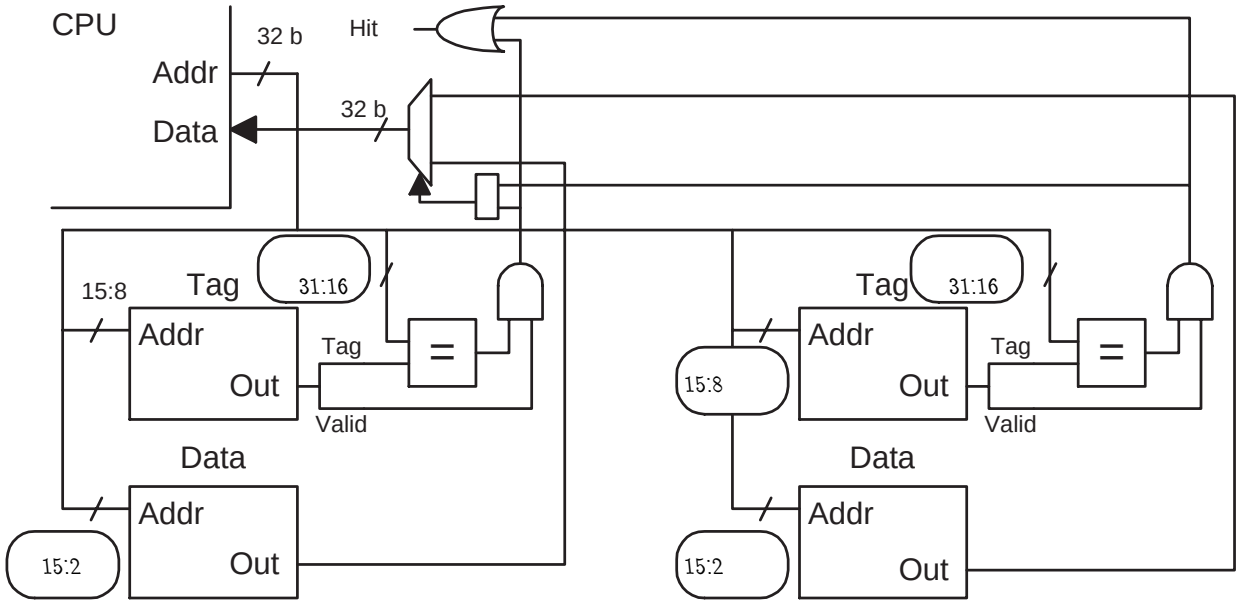
☑ Execution must be for decode logic of ordinary complexity. (See next item.)

☑ Execution must allow precise exceptions.

```
 # solution
 # Cycle         0  1  2  3  4  5  6  7
0x1000:
 lw  r4, 0(r5)   IF ID EX ME WB
 add r1, r2, r3  IF ID EX ME WB
 lh  r6, 0(r4)      IF ID -> EX ME WB
 xor r7, r1, r9     IF ID -> EX ME WB
 sll r10, r11, 12      IF -> ID EX ME WB
 srl r14, r15, 16      IF -> ID EX ME WB
```

Problem 4: (10 pts) The diagram below is for a two-way set-associative cache on a system with 8-bit characters. Answer the following, formulæ are fine as long as they consist of grade-time constants.

☑ Fill in the blanks in the diagram.



☑ Show the address bit categorization. Label the sections appropriately. (Alignment, Index, Offset, Tag.)

☑ Cache Capacity (Indicate Unit!!):
Capacity is 2×2^{16} characters (128 kiB).

☑ Memory Needed to Implement (Indicate Unit!!):

It's the cache capacity plus $2 \times 2^{16-8}(32 - 16 + 1)$ bits.

☑ Line Size (Indicate Unit!!):

Line size is 2^8 characters.

☑ Show the bit categorization for an eight-way cache with the same capacity and line size.

Problem 5: Answer each question below.

(*a*) The register renaming technique used in the dynamically scheduled processor (what the register maps do) solves a problem encountered when instructions execute out of order.(5 pts)

☑ What is the problem and how does it solve it?

☑ Provide an example showing what would go wrong without renaming.

The problem is that because instructions write back their results out of order and also read source registers out of order an instruction might read the wrong register value because (1) a later instruction already wrote the same register, overwriting the value that was supposed to be read, or (2) earlier instructions wrote the registers out of order. See the examples below. The solution is to assign a different physical register to each instruction that writes a destination so that there is no ambiguity. The ID register map is used to find which physical register is assigned to a given architected (from the ISA) register. Since the register map is read and updated in program order it will be correct.

```
# Case (1):  add reads wrong value of r2.
add  r1, r2, r3              ..RR..  # Ooops, got wrong r2.
sub  r2, r4, r5      ..WB..

# Case (2): xor reads wrong r1.

add  r1, r2, r3                ..WB..
...
or   r1, r6, r7      ..WB..
xor  r8, r1, r9                        ..RR..  # Ooops, got wrong r1.
```

(*b*) When it comes to caches one line size does not fit all.(5 pts)

☑ When is it better to have larger line sizes (while holding cache size constant)? Explain how the larger lines help.

When programs do alot of sequential access. The larger the line the more data is brought in on a cache miss. Since access is sequential that data will soon be accessed if the missing instruction is part of a piece of code accessing data sequentially.

☑ When is it better to have smaller line sizes (while holding cache size constant)? Explain how larger lines would hurt.

Suppose there is a miss on a system with large lines by a program doing no sequential access and with little spatial locality of any kind. The accessed data will be read but the other data on the line will not, wasting cache capacity. It would not hurt to have smaller line sizes since only the part which is first accessed is used, it would in fact help to have smaller lines because there would be more of them, increasing the hit ratio in many cases.

(*c*) Why would the SPEC CPU benchmark suite be less useful if it contained one integer program and one floating-point program? (5 pts)

☑ Why not just one of each?

Different programs have different characteristics making it difficult to choose one typical program. It might happen that the chosen program on a particular implementation does particularly well but that other programs don't do as well on that implementation. So if the suite had just one program it would not do a good job of predicting overall performance on a mix of programs.

(*d*) How would you answer a critic who said that the SPEC CPU benchmarks were rigged to make rich and powerful company I's products look good? (5 pts)

☑ They are not rigged because ...

... the benchmarks are chosen by SPEC members. Anyone can join SPEC and that includes I's competitors. If I were pushing benchmarks that favored its products the other members of SPEC, with their own hardware to sell, would object and the benchmarks would not be included in the suite.

Grading Note: The question is asking about the benchmarks themselves, some answered as though it were asking about the testing procedures.

Some answered that they are not rigged because SPEC is a non-profit who's mission is to provide fair tests. That does not make them immune to being influenced. The Washington, DC phone book would be alot lighter if one removed all non-profits (PACs, etc) that were not as impartial as they claim to be. What's important for SPEC is that its members have competing interests and so they will keep each other honest.

(*e*) What is the difference between a hardware interrupt, an exception, and a trap (as defined in class)? (5 pts)

☑ A hw interrupt's unique feature.

It has nothing to do with what's executing, it's triggered by a signal on a special processor port.

☑ An exception's unique feature.

It's tied to a particular instruction in which there was some problem with execution.

☑ A trap's unique feature.

It is an instruction, inserted to call a handler routine.