Name Solution

Computer Architecture
EE 4720
Midterm Examination 2.1.0

Friday, 12 November 2004, 10:40–11:30 CST

Problem 1		(50 pts)
Problem 2		(30 pts)
Problem 3		(20 pts)
Exam Total		(100 pts)

Alias Sell Processors

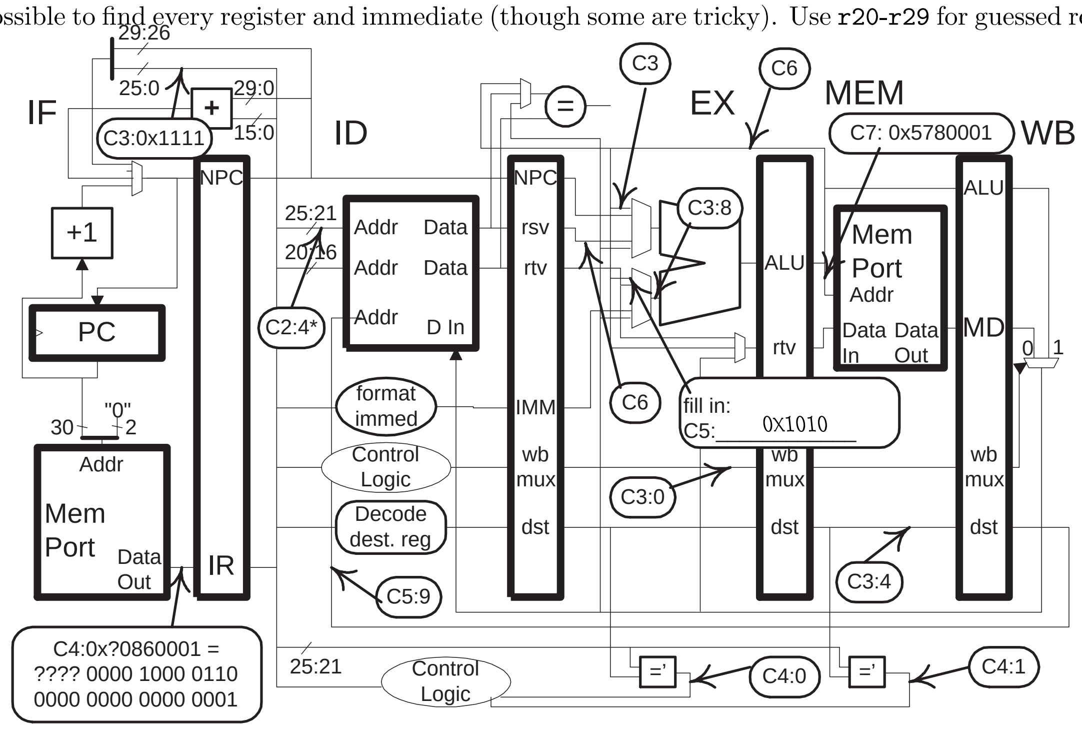
Good Luck!

Problem 1: In the diagram below some wires are labeled with cycle numbers and corresponding values. For example, $\boxed{\text{C2:4}}$ indicates that at cycle 2 the pointed-to wire will hold a 4. Other wires are labeled just with cycle numbers, indicating that the wire is used at that cycle. If a value on any labeled wire is changed the code would execute incorrectly. There are no stalls during the execution of the code. The first instruction (`lui`) is shown (but don't forget to finish it). In one of the C4's four bits are not shown, indicated by question marks. *Note:* $\boxed{\textit{C3:4}}$ *was not in the original exam; it conveys the same information as* $\boxed{\textit{C2:4*}}$. *See solution.* [50 pts]

☑ Write a program consistent with these labels.

☑ Fill in the blank next to $\boxed{\text{C5:}}$.

☑ It is possible to find every register and immediate (though some are tricky). Use `r20`-`r29` for guessed registers.



```
 Cycle                     0  1  2  3  4  5  6  7  8
0x1000: lui  r4, 0x578     IF ID EX ME WB
0x1004: lw   r9, 8(r4)        IF ID EX ME WB
0x1008: jal  0x4444              IF ID EX ME WB
0x100c: ADD  r6, r9, r31            IF ID EX ME WB
0x4444: sb   r6, 1(r4)                 IF ID EX ME WB
 Cycle                     0  1  2  3  4  5  6  7  8
```

The solution is shown above.

`lw`: This instruction has to be a load because the control signal for the WB-stage mux is set to 0, as indicated by $\boxed{\text{C3:0}}$ in EX. The $\boxed{\text{C3:8}}$ in EX tells us that the offset is 8. Because of the dependence with `lui` we know the effective address is `0x5780008` and so the load could be any size (it might be a `lb`, for example).

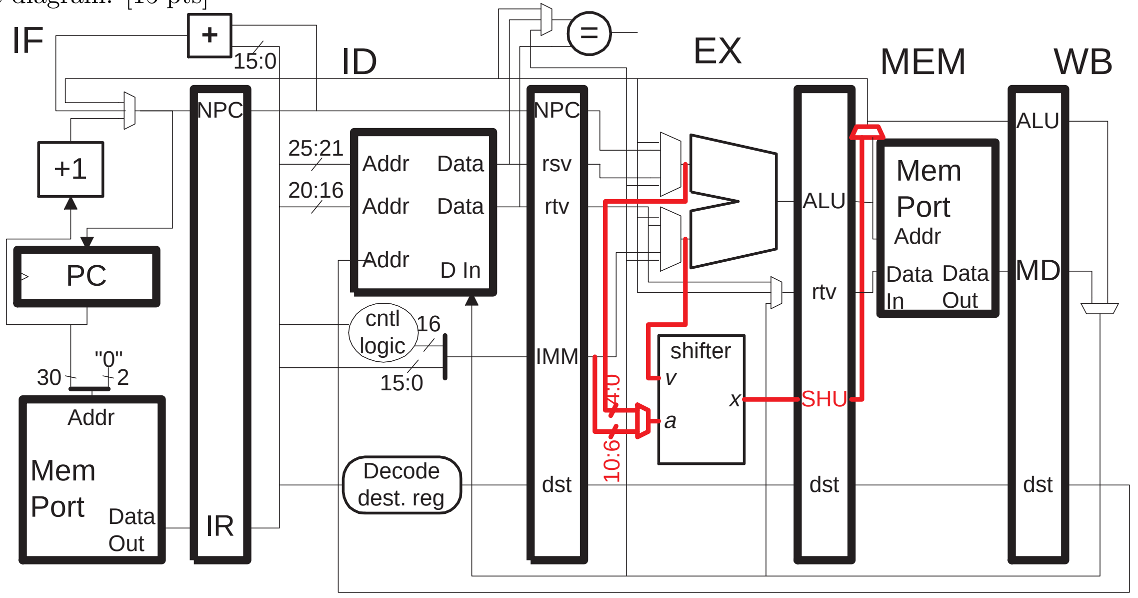
`jal`: The $\boxed{\text{C3:0x1111}}$ in ID (upper left) indicates that this is a `j` or `jal` since only those instructions construct a target using bits 25:0. The target is constructed by shifting the `ii` bits, 25:0, two bits to the left, yielding `0x4444` and putting the four high bits of NPC on (which are all zero), so the target is `0x4444`. The fill-in box indicates a dependency between the destination of this instruction and the `rt` source of the next one, and so this instruction must be a `jal` (instruction `j` does not write a destination). The destination of `jal` is the return address, which is `0x1010`, and that is the value to place in the fill-in box.

add: This could be any integer format-R instruction with two sources and a destination. The $\boxed{\text{C4:0}}$ and $\boxed{\text{C4:1}}$ along the bottom indicate that this instruction will bypass its `rs` value from the WB stage, which is how `r9` is determined.

sb: The $\boxed{\text{C7:0x5780001}}$ tells us that this is a memory instruction and that it is loading or storing a character. The $\boxed{\text{C6}}$ at the top on the bypass connection from MEM to EX tells us that the instruction uses a bypassed value and the $\boxed{\text{C6}}$ at the output of the EX `rsv` latch tells us it uses an unbypassed `rs` value. Therefore the bypassed value must be an `rt` value and so it can't be a load (because in a load `rt` is a destination). Therefore the instruction must be a `sb`. The register numbers and offset can be determined by examining $\boxed{\text{C4:0x?0860001}}$, which shows the encoded form of the store instruction (also shown in binary). The `rs` field is 4, the `rt` field is 6, and the immediate is 1, and so the instruction must be `sb r6, 1(r4)`.

*Change from original exam: In the original exam $\boxed{\text{C3:4}}$ in MEM was not present but $\boxed{\text{C2:4*}}$ in ID conveys the same information, that the `rs` register of the second instruction is `r4`. The problem stated that if a value on any pointed wire was changed execution would be incorrect. There are two ways to interpret "pointed wire." If the interpretation is "the portion of the wire that the arrow points at" then $\boxed{\text{C2:4*}}$ would be incorrect (and would be contradictory) because the `rs` value is bypassed and so it does not matter what is retrieved from the register file. If the interpretation is "anywhere on the wire" then $\boxed{\text{C2:4*}}$ would be correct because if those bits were changed the control logic would not set the ALU mux to bypass the correct value. Because of the way these problem are written the first interpretation makes more sense because the arrow is used to show the input that is using the value. Sorry about the problem.

Problem 2: The pipeline below includes a left shift unit (shifter) to be used by the left shift instructions. It performs the operation `x = v << a` (left shift `v` by `a` bits), where `x`, `v`, and `a` are the shifter ports shown in the diagram. [15 pts]



(*a*) Add connections to the shifter so that it can be used by shift left instructions. The code below should execute without a stall. Do not show control logic.

☑ The modifications should make efficient use of hardware.

☑ Be sure to show bit ranges on wires.

☑ Please check the code for dependencies.

Changes shown above in **red bold**. For `sll` instructions the shift amount is obtained from the `sa` field, bits 10:6. Those bits are available in EX in the `IMM` pipeline latch. For `sllv` the shift amount is the low five bits of the `rs` register value. Taking that from the upper ALU mux takes advantage of the already present bypassing hardware which can handle the bypasses in the code sample and other bypasses. For the same reason the item to shift, the `rt` register value, is taken from the lower ALU mux. The output of the shifter joins the path to WB in the MEM stage, so that existing bypass connections can be used. The cost of the new pipeline latch, SHU, could be saved by placing the added multiplexor in EX instead of MEM. That would stretch the critical path because the signal out of the ALU would have to pass through the added mux and so the added mux was placed in MEM.

```
# Cycle            0  1  2  3  4  5  6
sll   r1, r2, 10   IF ID EX ME WB
sllv  r4, r5, r1      IF ID EX ME WB
sllv  r6, r4, r1         IF ID EX ME WB
# Cycle            0  1  2  3  4  5  6
```

Hint: All of the shift instructions are Format R. The `rt` *register holds the value to be shifted. The* `v` *in* `sllv` *means the amount to shift is taken from a register.*

	Opcode	rs	rt	rd	sa	func
Format-R:	31 26	25 21	20 16	15 11	10 6	5 0

Problem 2, continued: The EX stage is the right stage for the shifter. [15 pts]

(*b*) Suppose the shifter were placed in the ID stage and correctly connected, including bypasses.

☑ What would be the impact on performance? Explain.

There would be two negative impacts on performance. First, the clock frequency would have to be lowered because the path through ID would include the register lookup and the time to shift. Performance would drop further because of stalls when there is a dependency with the immediately preceding instruction. In the example below the shifter needs `r1` when it is in ID but that is not available until the end of EX. (If the clock frequency were halved then the ALU could produce a result halfway through EX, avoiding a stall, if the ALU design were not changed. With half the clock frequency though a lower-cost and slower ALU might be substituted, if so a stall would be necessary.)

```
! Cycle         0  1  2  3  4  5
add r1, r2, r3  IF ID EX ME WB
sll r4, r1, 5      IF ID -> EX ME WB
```

(*c*) Suppose the shifter were placed in the MEM stage and correctly connected, including bypasses.

☑ What would be the impact on performance? Explain. *Hint: The explanation can use a code example.*

There would be a negative impact on performance because of stalls, in this case when there is a dependent instruction immediately following the shift. In the example below the `add` needs the result of the shift in the beginning of its EX, since the shift produces the result at the end of its ME the add must stall.

```
! Cycle         0  1  2  3  4  5
sll r1, r2, 5   IF ID EX ME WB
add r3, r1, r4     IF ID -> EX ME WB
```

Problem 3: Answer each question below.

(*a*) Consider a new ISA, MIPS-DS2, in which branches and other control transfers have two delay slots, not just one. [10 pts]

✓ Would the performance on something similar to the familiar (to us) five-stage implementation be higher, lower, or about the same as MIPS on its familiar five-stage implementation? Explain. (See next question before answering.)

The performance would be lower because the additional delay slot would not always be filled with a useful instruction—reducing performance—and even when the slot is filled there would be no performance benefit since the five-stage MIPS has no branch penalty.

☑ Describe an implementation that might favor MIPS-DS2 and explain how it favors MIPS-DS2.

One in which the branch direction and target are determined two stages after IF. With plain-old-MIPS an instruction would be squashed on a taken branch, not so on MIPS-DS2. (Though if the compiler couldn't fill the delay slot it would only be a `nop` that's not squashed.) In the examples below the new implementation has six stages, stages F1 and F2 do the same thing done by IF in the fix-stage implementation. When conventional MIPS runs on the six-stage implementation it will always squash an instruction when a branch is taken.

```
 # One delay slot, five-stage implementation.
 #
 add  r4, r5, r6    IF ID EX ME WB
 bneq r1, r2  TARG     IF ID EX ME WB
 or r7, r8, r9            IF ID EX ME WB
...
TARG:
 sub r10, r11, r12           IF ID EX ME WB

 # One delay slot, six-stage implementation.
 #
 add  r4, r5, r6    F1 F2 ID EX ME WB
 bneq r1, r2  TARG     F1 F2 ID EX ME WB
 or r7, r8, r9            F1 F2 ID EX ME WB
 xor r10, r11, r12           F1x (Squashed)
...
TARG:
 sub r10, r11, r12              F1 F2 ID EX ME WB

 # Two delay slots, six-stage implementation. (IF replaced by F1 and F2)
 #
 bneq r1, r2  TARG  F1 F2 ID EX ME WB
 add  r4, r5, r6       F1 F2 ID EX ME WB
 or r7, r8, r9            F1 F2 ID EX ME WB
...
TARG:
 sub r10, r11, r12           F1 F2 ID EX ME WB

 # Two delay slots, five-stage implementation.
 #
 bneq r1, r2  TARG  IF ID EX ME WB
 add  r4, r5, r6       IF ID EX ME WB
 or r7, r8, r9            IF ID EX ME WB
...
TARG:
 sub r10, r11, r12           IF ID EX ME WB

 # Two delay slots, six-stage implementation. (IF replaced by F1 and F2)
 #
 bneq r1, r2  TARG  F1 F2 ID EX ME WB
 add  r4, r5, r6       F1 F2 ID EX ME WB
 or r7, r8, r9            F1 F2 ID EX ME WB
...
TARG:
 sub r10, r11, r12           F1 F2 ID EX ME WB
```

☑ Other than the people that suggested the second delay slot and designed the hardware, if it works who gets the promotion, if it does not work who gets re-assigned to tech support (punished)? (Give a job description, not a name.) *Hint: That person would probably quit if it were a MIPS-DS9. Note: The parenthesized part and the hint were not on the original exam. This question was originally the second of these three questions and seemed to refer to the five-stage implementation. Since the question really applied to the new implementation, not the five-stage one, the class was told not to answer it.*

The compiler writer since it is his or her skill that is needed to fill the delay slots with extra instructions.

(*b*) For an instruction that raises a precise exception the handler can return and so the faulting instruction can be re-executed or skipped. [10 pts]

☑ Describe a situation in which the handler would need to either re-execute or skip. Mention what caused the exception and what the handler did about it.

The processor is decoding an instruction that is in the ISA but is not implemented. That would trigger an illegal instruction exception. The handler would perform in software what the instruction was supposed to do in hardware, and then return to the instruction immediately following the "illegal" instruction.

Another example is a load instruction in which the address is valid, adjustments need to be made to the memory system. The handler will make those adjustments and return to re-execute the load. Those adjustments, to be covered later in the semester, include a TLB (a cache used by hardware to translate virtual addresses to physical addresses) miss or a page fault (the area of address space containing the address is not present, perhaps swapped out to disk).

☑ Other times there is no need to return. Describe such a situation, mention what caused the exception and why the handler would not return.

The instruction really was illegal. There is no point returning because the program can not be expected to run correctly.