LSU EE 4720 Homework 4Due: 19 Mar 2025 at 09:30 CDT

Formatted 15:21, 17 March 2025

Collaboration Rules

Each student is expected to complete his or her own assignment. It is okay to work with other students and to ask questions in order to get ideas on how to solve the problems or how to overcome some obstacle (be it a question of MIPS or assembler syntax, interpreting error messages, how a part of the problem might be solved, etc.) It is also acceptable to seek out assembly language resources for help on MIPS, etc. It is okay to make use of AI LLM tools such as ChatGPT and Copilot to generate sample code. (Do not assume LLM output is correct. Treat LLM output the same way one might treat legal advice given by a lawyer character in a movie: it may sound impressive, but it can range from sage advice to utter nonsense.)

After availing oneself to these resources **each student is expected to be able to complete the assignment alone**. Test questions will be based on homework questions and **the assumed time needed to complete the question will be for a student who had solved the homework assignment on which it was based**.

Student Expectations

Some of the problems require thought, and students are expected to persevere until they find a solution. It is each student's duty to him or herself to resolve frustrations and roadblocks quickly, hopefully helped along by the satisfaction of making progress. There are plenty of old problems and solutions to look at. One way to resolve issues is to ask Dr. Koppelman or others for help.

Resources

See old homework and exams. There are a few questions about VAX in past assignments. There are question about RISC-V in many of the more recent assignments.

Problem 1: Remember that VAX is one of the few examples of a good CISC ISA. CISC ISAs are not considered suitable for current implementation technology, but those who do not learn by history are doomed to repeat it, so look over the summary of the VAX instruction set which can be found in Chapter 2 of the VAX 11/780 Architecture Handbook Volume 1, 1977-78. Focus on Section 2.4, which summarizes the instruction set. Consider item 5 in that section, which starts "Instructions provided specifically for high-level language constructs." Three examples of such instructions are given, `ACB`, `CALLS`, and `CASE`. As guided by the check boxes below, explain how a register-only version of suitable each instruction is for implementation in a RISC ISA. The instruction descriptions in the architecture handbook use metasyntactic symbols `rx`, `mx`, and `wx` to sources and destinations. (In MIPS `rs`, `rt`, and `rd` are metasyntactic symbols.) Symbol `rx` is used for a read (source) operand (signified by the `r`) that can come from a register, immediate, or memory (signified by the `x`). Similarly the `w` in `wx` signifies an argument that is written (a destination), and the `m` in `mx` signifies an argument that is read and then written. The questions below ask about hypothetical *register-only* versions of these instructions in which arguments `rx`, `mx`, and `wx` refer only to register arguments.

The instructions are explained in the architecture manual, but feel free to seek out other references. The description of `ACB` is fairly straightforward. The `CALLS` instruction is clear but may be difficult to understand for those who are less familiar with bit masks or bit vectors. In addition to the Architecture Handbook, see VAX MACRO and Instruction Set Reference Manual for a description of the `CASE` instruction and an example of its use. Note that for `CASES` the table (`displ`) is in memory immediately after the instruction. The operation performed by the `CASE` instruction is similar to the MIPS assembly code for the dense switch statement presented in the class control flow demo code. Of course, `CASE` does most of that with one instruction.

☐ A register-and-displacement-operand-only version of the `ACB` instruction ◯ *is definitely not suitable for a RISC ISA*, ◯ *arguably possible for a RISC ISA*, ◯ *fits well into a RISC ISA*.

☐ Explain. In your explanation consider how easy it would be to ☐ encode in a RISC ISA (allow some flexibility) and how easy it would be ☐ to implement in a five-stage pipeline.

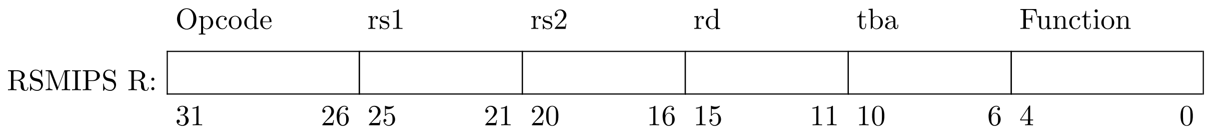
☐ The `CALLS` instruction ◯ *is definitely not suitable for a RISC ISA*, ◯ *arguably possible for a RISC ISA*, ◯ *fits well into a RISC ISA*.

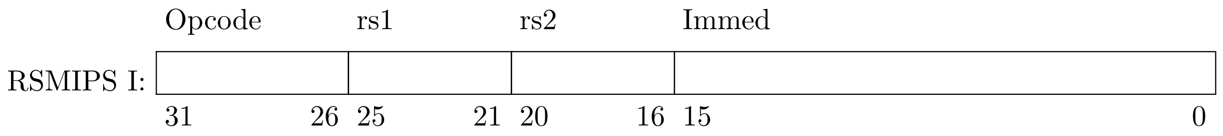
☐ Explain. In your explanation consider how easy it would be to ☐ encode in a RISC ISA (allow some flexibility) and how easy it would be ☐ to implement in a five-stage pipeline.

☐ A register-operand-only version of the `CASE` instruction ◯ *is definitely not suitable for a RISC ISA*, ◯ *arguably possible for a RISC ISA*, ◯ *fits well into a RISC ISA*.

☐ Explain. In your explanation consider how easy it would be to ☐ encode in a RISC ISA (allow some flexibility) and how easy it would be ☐ to implement in a five-stage pipeline.

Problem 2: RSMIPS is a hypothetical ISA with similarities to MIPS. Appearing below is RSMIPS' instruction format R, which is identical to MIPS' format R (except for the names of the source fields). Unlike MIPS, in RSMIPS all instructions that write a result to a register use the `rd` field for the register number (and the `rd` field is always in bits 15:11). Yes, RSMIPS is Real Strict about source and destination register fields, hence the name. Also notice that different from MIPS the RSMIPS source fields are named `rs1` and `rs2`. Remember that in MIPS, `rt` can be used as either a source or destination, depending on the instruction.





Because of this Real Strict provision, something like MIPS' format I can't be used for instructions such as `addi` and `lw`, but format I can be used for instructions such as `sw` and `beq`.

(*a*) In RSMIPS *format DI* is used for immediate instructions that write a result. Show a possible format DI. This is easy for those that understand what an instruction format is. (Note that RISC-V also follows this Real Strict philosophy, but the answer to this question is not an exact copy of a RISC-V instruction format.)

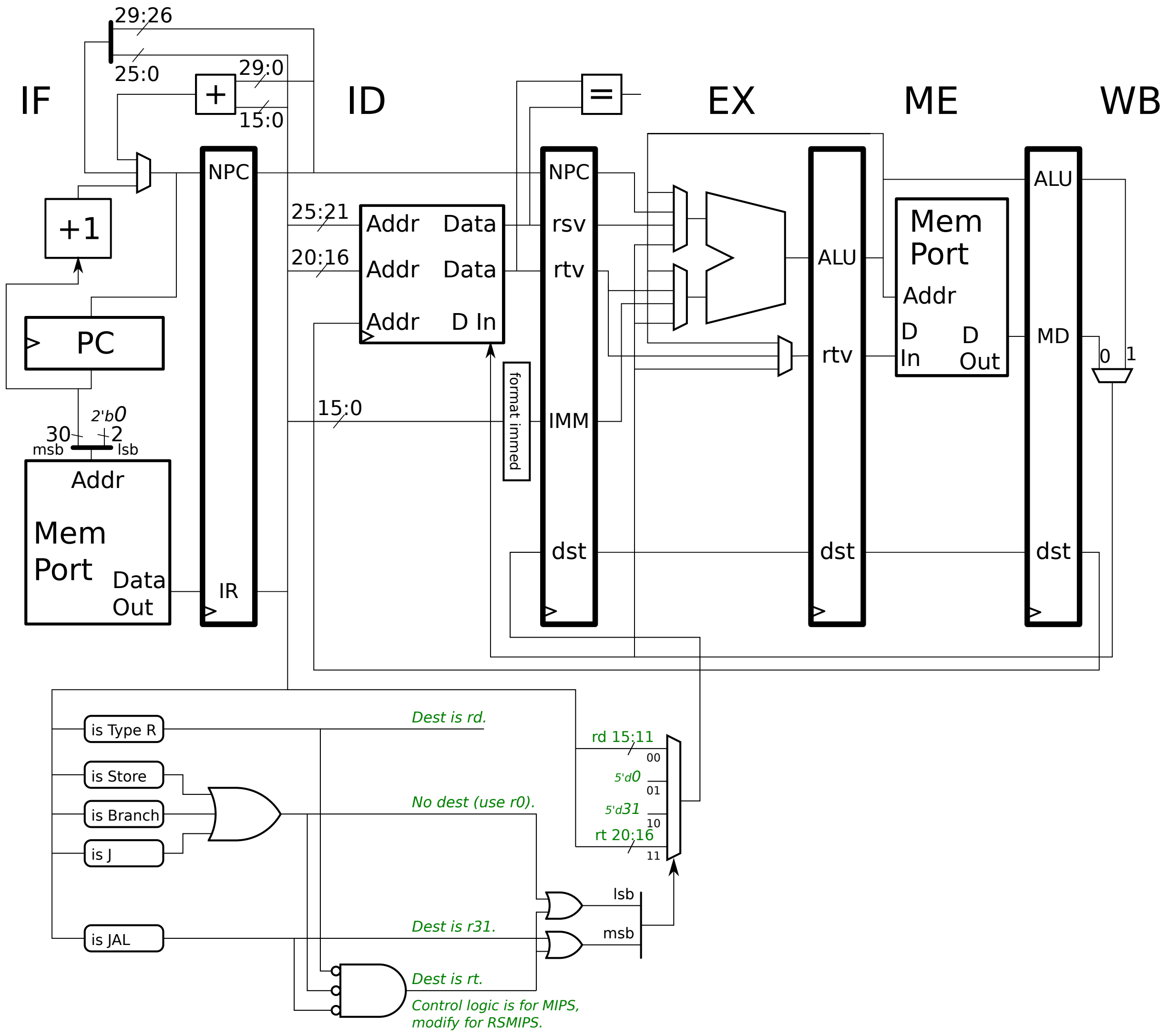
☐ Show a possible format DI.

(*b*) Convert the MIPS implementation below into an RSMIPS that works with format DI, format I, and format R RSMIPS instructions as requested in the checkbox items below. The illustration in SVG format can be found at `https://www.ece.lsu.edu/ee4720/2025/hw04-rsmips.svg`. It can be modified with your favorite SVG editor, even if it's not Inkscape.

☐ Modify the control logic to extract the correct destination register.

☐ Modify the datapath and control logic to provide the correct immediate.

☐ Be sure that the logic works with RSMIPS' format I, DI, and R instructions.



IF
ID
EX
ME
WB
29:26
25:0
29:0
15:0
+
=
NPC
+1
PC
2'b0
30
2
msb
lsb
Addr
Mem Port
Data Out
IR
25:21
20:16
Addr
Data
Addr
Data
Addr
D In
15:0
format immed
NPC
rsv
rtv
IMM
dst
ALU
rtv
dst
Mem Port
Addr
D In
D Out
ALU
MD
dst
0
1
is Type R
is Store
is Branch
is J
is JAL
Dest is rd.
No dest (use r0).
Dest is r31.
Dest is rt.
Control logic is for MIPS, modify for RSMIPS.
rd 15:11
5'd0
5'd31
rt 20:16
00
01
10
11
lsb
msb