UNIVERSITY OF CALIFORNIA, DAVIS
Department of Electrical and Computer Engineering

EEC 170 Introduction to Computer Architecture Winter 2021

Quiz 2 Solutions

All instructions in this exam, unless otherwise noted, are RISC-V RV32I instructions. This means that, unless otherwise noted, all registers are 32 bits wide on this exam. Recall that if the instruction writes to a register, that register appears first in the instruction. Unless otherwise indicated, you may not use any RISC-V pseudoinstructions on this exam, only actual RISC-V instructions.

1. The SPUR architecture (RISC-IV) contained an instruction called `extract`. An `extract` instruction takes a destination register argument (`rd`), a source register argument (`rs`), and an immediate argument (`imm`) (so, `extract rd, rs, imm`).

   `extract` does the following: `dest<07:00> ← one byte from rs selected by imm`.

   In English, this instruction takes the `imm`'th least-significant byte of `rs` and writes it into the least significant byte of `rd`, without changing the other bits of `rd`. Two examples, assuming that the initial values of the two registers are `rd = 0xabcdefgh` and `rs = 0xijklmnop`:

   - `extract rd, rs, 0` results in `rd ← 0xabcdefop`
   - `extract rd, rs, 2` results in `rd ← 0xabcdefkl`

   Assignment Project Exam Help
https://tutorcs.com
WeChat: cstutorcs

   (a) (2 points) If this instruction were implemented in RISC-V, **what type should it be?**

   > **Solution:** I-type. Like other I-type instructions, it has 1 register destination, 1 register source, and 1 immediate. Also, it is most similar in function to immediate-using logic instructions like `andi` or `ori`, both of which are I-type.

   (b) (12 points) **Write a minimal RISC-V (RV32I) code segment that implements `extract`.** Note that all registers are 32 bits wide. Use `rd` for the destination register, `rs` for the source register (do not change the value of this register), and `rt0`, `rt1`, etc. for any temporaries that you need.

   You will get some partial credit if you clearly write a short sentence or two that describes your strategy even if your code has errors.

   > **Solution:** Clear the low byte of the destination register, shift the correct byte of the source register into the low byte of a temporary, zero out the rest of the bytes of the temporary, and `or` the low byte of the temporary into the low byte of the destination.

```
andi rd, rd, -256 # this is 0xffffff00 and clears the low byte of rd
addi rt0, r0, imm # put the immediate in a register
slli rt0, rt0, 3  # multiply the immediate by 8
srl  rt1, rs, rt0 # shift the source register right to put the byte
                  # of interest into the low byte of rt1
andi rt1, rt1, 0xff # zero out all but the low byte of rt1
or   rd, rd, rt1  # put the low byte of rt1 into the low byte of rd
```

Let's watch how this works on the second example above:

```
                  # rd = 0xabcdefgh, rs = 0xijklmnop
andi rd, rd, -256 # rd = 0xabcdef00
addi rt0, r0, imm # rt0 = imm = 2
slli rt0, rt0, 3  # rt0 = imm * 8 = 16
                  #       (shift left by 3 == multiply by 8)
srl  rt1, rs, rt0 # rt1 = 0x0000ijkl
andi rt1, rt1, 0xff # rt1 = 0x000000kl
or   rd, rd, rt1  # rd = 0xabcdefkl
```

Assignment Project Exam Help
https://tutorcs.com

2. The following opcode map is for the ARM Thumb instruction set, sourced from the ARM7TDMI Technical Reference Manual. It lists 19 different instruction types (analogous to RISC-V's R-type, I-type, etc.) down the left side and their bit encodings in the table itself. ARM Thumb instructions are 16 bits wide and the opcode map shows how those 16 bits are used for each of the 19 instruction types. Note that any use of `R_` (`Rs`, `Rd`, etc.) indicates a register argument. Assume that any arithmetic immediates are unsigned (because they are). The "ALU operation" category includes all logical operations.

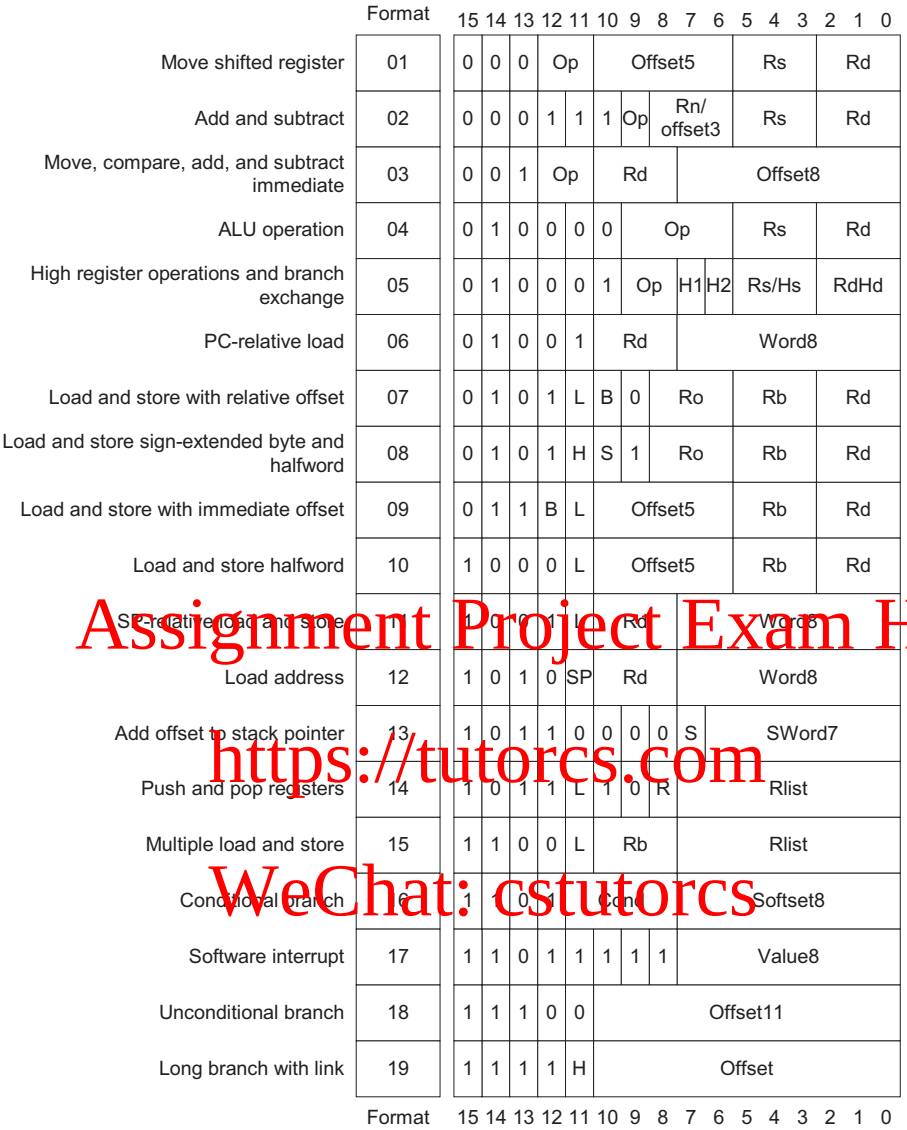
WeChat: cstutorcs

   For instance, one of the instruction types is "move shifted register". Instructions in this category always have bits 15–13 set to zero; bits 12–11 encode the specific instruction within this category; and the other instructions are used for two source registers and an immediate ("offset").

   Pay particular attention to:

   - How many registers can be addressed per instruction type
   - What it means if there are fewer registers for a particular instruction type than you might otherwise expect (hint: think of what x86 does)
   - How many bits are available for each register in the instruction encoding
   - If a field contains two possibilities (as indicated by a slash, `a/b`), you can assume that some variant of the instruction will use one of the possibilities (`a`) and another

variant of the same instruction will use the other possibility (`b`).



Format	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Move shifted register	01	0	0	0	Op		Offset5				Rs			Rd		
Add and subtract	02	0	0	0	1	1	1	Op	Rn/ offset3		Rs			Rd		
Move, compare, add, and subtract immediate	03	0	0	1	Op		Rd		Offset8							
ALU operation	04	0	1	0	0	0	0	Op			Rs			Rd		
High register operations and branch exchange	05	0	1	0	0	0	1	Op		H1	H2	Rs/Hs			RdHd	
PC-relative load	06	0	1	0	0	1	Rd		Word8							
Load and store with relative offset	07	0	1	0	1	L	B	0	Ro		Rb			Rd		
Load and store sign-extended byte and halfword	08	0	1	0	1	H	S	1	Ro		Rb			Rd		
Load and store with immediate offset	09	0	1	1	B	L	Offset5				Rb			Rd		
Load and store halfword	10	1	0	0	0	L	Offset5				Rb			Rd		
SP-relative load and store	11	1	0	0	1	L	Rd		Word8							
Load address	12	1	0	1	0	SP	Rd		Word8							
Add offset to stack pointer	13	1	0	1	1	0	0	0	0	S	SWord7					
Push and pop registers	14	1	0	1	1	L	1	0	R		Rlist					
Multiple load and store	15	1	1	0	0	L	Rb		Rlist							
Conditional branch	16	1	1	0	1	Cond	Softset8									
Software interrupt	17	1	1	0	1	1	1	1	1	Value8						
Unconditional branch	18	1	1	1	0	0	Offset11									
Long branch with link	19	1	1	1	1	H	Offset									
Format		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Assignment Project Exam Help
https://tutorcs.com
WeChat: cstutorcs

(a) (2 points) How many different registers can be addressed by an ARM Thumb instruction?

> **Solution:** Registers are encoded with 3 bits each, so $2^3 = 8$ registers.

(b) (10 points) **Write a minimal set of ARM Thumb assembly instructions to perform the following operation:**

`r1 = (r2 AND (-500 + r3 + r4)) OR r5`

where `r#` indicates a value in register `#`.

You may only change `r1` (you cannot change the value of any other register). You may assume `r1` is initialized to zero.

The instructional staff believes the opcode map contains all the information you need to solve this problem, but encourages you to explicitly write down any assumptions you are making.

The instructional staff understands that you've probably never seen the ARM or ARM Thumb instruction sets before. Use RISC-V notation (e.g., `add r1, r1, r1`) for the assembly code you're writing, but adhere to any limitations of the ARM Thumb instruction set.

Solution: We expect correct solutions to make two important realizations about this instruction set that differ from RISC-V:

- Most instructions take only 2 register arguments, except for add/subtract.
- The largest immediate available is 8 bits (255).

```
1 add  r1, r3, r4   # r1 = r3+r4
2 subi r1, r1, 250  # can't fit the whole immediate in one instr ...
3 subi r1, r1, 250  # ... now r1 = r3+r4-500
4 and  r1, r1, r2   # r1 = r2 AND (r3+r4-500)
5 or   r1, r1, r5   # r1 = (r2 AND (r3+r4-500)) OR r5
```

Assignment Project Exam Help
https://tutorcs.com
WeChat: cstutorcs

3. Eddie is trying to simplify the RISC-V instruction set. He makes the following suggestions. Explain why making the simplifications he suggests are likely to make the RISC-V implementation more complex or hurt performance (or, equivalently, why they are a bad idea).

   (a) (2 points) "Having an S-type instruction class seems silly to me. It has all the same information as an I-type instruction. Why not make all S-type instructions I-type instead?"

   Solution: In all RISC-V instructions, register reads come from registers specified from bits `<24:20>` and `<19:15>`. S-type instructions require reading from two registers, one for address and one for data. If we used bits `<11:7>` to specify a read register, we would have to add additional logic to handle this special case.

   (b) (2 points) "This idea of having some instructions with 7 total opcode bits, some with 10 (opcode + funct3), and some with 17 (opcode + funct7 + funct3) is really confusing. Since we have enough opcode space to encode all instructions with a uniform 10 bits of opcode per instruction, let's make all instructions have 10 opcode bits."

Solution: If we adopted Eddie's solution, U-type and UJ-type instruction classes (supporting instructions like `LUI`, `AUIPC`, `JAL`, and `JALR`) would not be able to support as many immediate bits as they do now. Consequently, some of these instructions that currently require all these bits would no longer be able to be expressed in a single instruction and we would have to use multiple instructions to encode them, increasing code size and decreasing performance.

Assignment Project Exam Help

https://tutorcs.com

WeChat: cstutorcs