STAM Center
SECURE, TRUSTED, AND ASSURED MICROELECTRONICS

Ira A. Fulton Schools of Engineering
Arizona State University

CSE 520 Computer Architecture II
Term: Spring 2026
Lead Instructor: **Prof. Michel A. Kinsy**

Review Problem Set 1

Posted Jan. 27th, 2026

---

http://ascslab.org/courses/cse520/index.html

This problem set is designed to give you practice and help you identify and address any gaps in your understanding.

Problem 1

The ***NOR*** instruction is not part of the RISC-V instruction set because the same functionality can be implemented using existing instructions. Write a short assembly code snippet that has the following functionality: ***s3 = s4 NOR s5***. Use as few instructions as possible.

Problem 2

Convert the following high-level code snippet into RISC-V assembly. Assume that the base address of ***array1*** and ***array2*** are held in ***t1*** and ***t2*** and that the ***array2*** array is initialized before it is used. Use as few instructions as possible. Clearly comment your code.

```
int i;
int array1[100]; int array2[100];
…
for (i = 0; i < 100; i = i + 1) array1[i] = array2[i];
```

Problem 3

Convert the following RISC-V assembly code into machine language. Write the instructions in hexadecimal.

```
add s7, s8, s9
srai t0, t1, 0xC
oti s3, s1, 0xABC
lw s4, 0x5C(t3)
```

Problem 4

Consider the RISC-V machine code snippet below.

a. Convert the machine code snippet into RISC-V assembly language.
b. Reverse engineer a high-level program that would compile into this assembly language routine and write it.
c. Explain in words what the program does. ***a0*** and ***a1*** are the inputs. a0 contains a 32-bit number and ***a1*** is the address of a 32-element array of characters.

```
0x01F00393
0x00755E33
0x001E7E13
0x01C580A3
0x00158593
```

```
0xFFF38393
0xFE03D6E3
0x00008067
```

Problem 5

Consider the following C code snippet.

```
// C code
void setArray(int num) {
      int i;
      int array[10];
      for (i = 0; i < 10; i = i + 1)
            array[i] = compare(num, i);
}

int compare(int a, int b) {
      if (sub(a, b) >= 0)
            return 1;
      else
            return 0;
}

int sub(int a, int b) {
      return a - b;
}
```

a. Implement the C code snippet in RISC-V assembly language. Use ***s4*** to hold the variable ***i***. Be sure to handle the stack pointer appropriately. The array is stored on the stack of the ***setArray*** function. Clearly comment your code.

b. Assume ***setArra***y is the first function called. Draw the status of the stack before calling ***setArray*** and during each function call. Indicate stack addresses and the names of registers and variables stored on the stack; mark the location of ***sp***; and clearly mark each stack frame. Assume that ***sp*** starts at ***0x8000***.

Problem 6

What is the maximum number of instructions that a branch instruction (like ***beq***) can branch backward (i.e., to lower instruction addresses)?

Problem 7

Consider how far ***jal*** instructions can jump.

a. How many instructions can a ***jal*** instruction jump forward (i.e., to higher addresses)?
b. How many instructions can a ***jal*** instruction jump backward (i.e., to lower addresses)?

Problem 8

Processor Control-Signal Assignment

For each of the five RISC-V instructions listed below, fill in the control-signal values for the single-cycle RISC-V.

Use the following control signals a canonical single-cycle CPU (Figure below):

- `PCSrc`
- `Branch`
- `MemRead`
- `MemWrite`
- `ALUSrc`
- `ALUOp (2 bits)`
- `RegWrite`
- `MemtoReg`
- `ImmGen Type (I, S, B, U, J, don't-care if unused)`

Instructions to Analyze

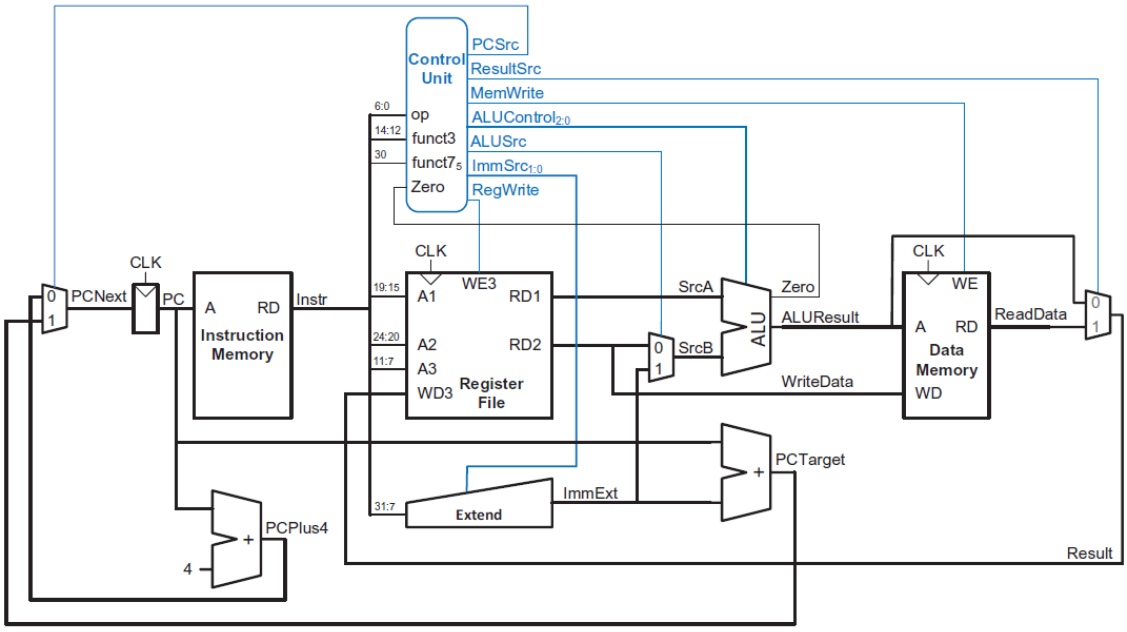
1. `add x5, x6, x7`
2. `lw x10, 12(x3)`
3. `sw x4, 8(x2)`
4. `beq x1, x2, Loop`
5. `ori x8, x9, 0xFF`

Main Control Signals Table

Instruction	PCSrc	Branch	MemRead	MemWrite	ALUSrc	ALUOp	RegWrite	MemtoReg	ImmGen Type
`add x5, x6, x7`									
`lw x10, 12(x3)`									
`sw x4, 8(x2)`									
`beq x1, x2, Loop`									
`ori x8, x9, 0xFF`									

ALU Control Signals Table

Instruction	ALUOp	Funct3	Funct7	ALU Control
`add x5, x6, x7`				
`lw x10, 12(x3)`				
`sw x4, 8(x2)`				
`beq x1, x2, Loop`				
`ori x8, x9, 0xFF`				


Control Unit
op
funct3
funct7₅
Zero
6:0
14:12
30
PCSrc
ResultSrc
MemWrite
ALUControl2:0
ALUSrc
ImmSrc1:0
RegWrite
CLK
PCNext
PC
Instruction Memory
A
RD
Instr
19:15
24:20
11:7
A1
A2
A3
WD3
WE3
RD1
RD2
Register File
SrcA
SrcB
ALU
Zero
ALUResult
WriteData
Data Memory
WE
WD
ReadData
PCPlus4
4
Extend
31:7
ImmExt
PCTarget
Result