CSEN 122L: Project (Fall 2025)

Design a Structural Model of a Pipelined CPU

Description: The project is to design a structural model of a pipelined CPU with 11 instructions using Verilog HDL. In this project, you will design a 32-bit pipelined CPU for the given SCU Instruction Set Architecture (SCU ISA). The SCU ISA is described below.

- Register file size: 64 registers, each register has 32 bits. The 64 register names are referred to as x0, `x1`, `x2`, ..., `x63`.
- PC: 32 bits
- Instruction format: Each instruction is 32-bit wide, and consists of five fields: `opcode` (operation code), `rd` (destination register), `rs` (first source register), `rt` (second source register), and `y` (immediate value). The format is as follows.

y: immediate value (10 bits)	rd (6 bits)	rs (6 bits)	rt (6 bits)	Opcode (4 bits)

The 11 instructions are defined in the following table (**X** = don't care / unused)

Instruction	Syntax	y	rd	rs	rt	Opcode	Function
No operation	`NOP`	X	X	X	X	0000	No operation
Save PC	`SVPC rd,y`	y	rd	X	X	1111	xrd <- PC + y (y signed extended)
Load	`LD rd,rs,y`	y	rd	rs	X	1110	xrd <- M[xrs+y] (y signed extended)
Store	`ST rt,rs,y`	y	X	rs	rt	0011	M[xrs+y] <- rt (y signed extended)
Add	`ADD rd,rs,rt`	X	rd	rs	rt	0100	xrd <- xrs + xrt
Increment	`INC rd,rs,y`	y	rd	rs	X	0101	xrd <- xrs + y (y signed extended)
Negate	`NEG rd,rs`	X	rd	rs	X	0110	xrd <- -xrs
Subtract	`SUB rd,rs,rt`	X	rd	rs	rt	0111	xrd <- xrs - xrt
Jump	`J rs`	X	X	rs	X	1000	PC <- xrs
Branch if zero	`BRZ rs`	X	X	rs	X	1001	PC <- xrs, if Z=1
Branch if negative	`BRN rs`	X	X	rs	X	1010	PC <- xrs, if N=1

SCU ISA CPU Characteristics

(1) Word addressing architecture: Unlike RISC-V, SCU ISA uses word addressing, not byte addressing. In other words, in the word addressing architecture, each individual address indicates a different word (32 bits). In byte addressing, on the other hand, each individual address indicates a different byte (8 bits).

Suppose that we have the following code with SCU ISA:

```
SVPC x5,1          // x5 = PC + 1
ADD  x6,x7,x8      // x6 = x7 + x8
SUB  x9,x10,x11    // x9 = x10 - x11
```

Assuming the current PC is 1000 executing the SVPC instruction "`SVPC x5,1`", then the x5 register will be updated to hold a value of 1001 that indicates the address of `ADD` ("`ADD x6,x7,x8`") instruction.

(2) Branch instructions use the zero/negative flag from the previous instruction executed immediately before. Our branch instructions, "Branch if Zero (`BRZ`)" and "Branch if Negative (`BRN`)", have only one operand that indicates the branch target address. For branch condition, the branch instruction needs to look at the Zero (`Z`) or Negative (`N`) flag from the instruction executed immediately before the branch instruction. The Zero flag is 1 if the previous ALU result was 0. The Negative flag is 1 if the previous ALU result was a negative value.

Suppose that we have the following code:

```
SVPC x5,1          // x5 <- PC + 1
ADD  x6,x7,x8      // x6 <- x7 + x8
SUB  x9,x10,x11    // x9 <- x10 - x11
BRN  x5            // PC <- x5, if x9 < 0
```

The branch instruction BRN will set the PC to indicate the ADD ("`ADD x6,x7,x8`") instruction if the SUB instruction ("`SUB x9,x10,x11`")'s result is a negative value.

Also, refer to the ALU block diagram and truth table in the last page of this document.

Assembly programming (the first Project homework)

Write an assembly program using the 11 base instructions (`NOP`, `SVPC`, `LD`, `ST`, `ADD`, `INC`, `NEG`, `SUB`, `J`, `BRZ`, `BRN`) of the SCU ISA to perform a 1-d median stencil program, as described in below. You can use a text editor or a word processor. Submit it to Camino.

1-d median stencil program

This program will take an input array `a` and an output array `b`, each of which has `n` number of elements. And each element's size is 4 bytes (integer). For each element of `b`, do the following computation:

$b_i = \text{median}(a_{i-1}, a_i, a_{i+1})$ for $i = 1, 2, 3, \ldots, n-2$

For boundary conditions, i.e., for `i=0` and `i=n-1`, just take the element value of `a`, i.e.,

$b_0 = a_0$
$b_{n-1} = a_{n-1}$

When you write the assembly code, you can assume the **problem size *n*** (the number of vector elements to add) and the **base addresses of array `a`** (`&a[0]`) **and `b`** (`&b[0]`) in some registers. Specify your assumptions in your submission and write some comments about your logic.

This program will also be used to test your CPU.

Zero register: If you want to use a specific register for hard-wired 0 value, you can do that, but specify that in your submission and also in your project report if you implemented your CPU to do that. You can also initialize a specific register with a value of 0, using a simple sub instruction, e.g.,

```
sub x0, x0, x0
```

Assignments and Submission:

1. **All students** submit their team information to **Dr. Cho (to the Lecture Camino section)** by **Monday of Week 6** (as already announced) (1 pt in CSEN122 Lecture).

2. The assembly codes, for the vector add program, to **Dr. Cho (to the Lecture Camino Section)** by **Wednesday of Week 7**. Please upload your solution to Camino CSEN 122 lecture. **One submission per team**. (6 pts in CSEN122 Lecture)

3. Your datapath and control (including the truth table) for the SCU ISA that supports the 11 SCU ISA instructions by **Friday of Week 8** to **Dr. Cho (to the Lecture Camino Section)**. Use the ALU attached in the last page. Upload your datapath to Camino CSEN 122 lecture. **One submission per team**. (8 pts in CSEN122 Lecture).

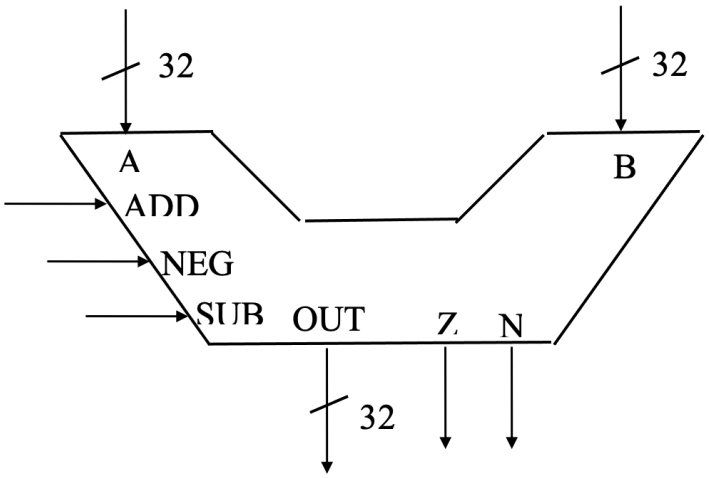
4. Report (30 pts in CSEN122 Lab): Submit a written final report to the **TA (to the Lab Camino section)** by **Tuesday of Week 11**, including the followings:

- Abstract (short description or outline of the project),
- Detailed description of the CPU design including the datapath and the truth table of your control.
- Test benchmarks/waveforms verifying the functions.
- The 1-d median stencil assembly code for the benchmark.
- Performance analysis: estimate CPI, express the instruction count, total number of cycles in terms of the problem size *n*, and verify your estimate with simulation/waveform. When you analyze the cycle time, you can use the following delay data: delay of memory (I and D memory): 2 ns., delay of register file: 1.5 ns., delay of ALU (adders): 2 ns. Ignore the delays of all other components. Use the ALU attached in the last page.

5. Codes & Demo (30 pts in CSEN122 Lab): You will submit your working program codes to the **TA (to the Lab Camino section)** by **Tuesday of Week 11**. And you will also demo your working programs to the **TA (during the Lab times or TA office hours)** by **Tuesday of Week 11**. During the demo, the TA will provide you *n* random numbers for the input array of the 1-d median stencil program and you will show the TA the result after running your program on your pipeline. You will also be asked to perform random but related tasks (for instance, change the address of data and demo the modified program), and be prepared to demo/answer related project questions. The purpose of these questions is to make sure that you understand the project thoroughly. All team members must be present for the demo to receive the points. The Week 11 Tuesday deadline already includes any extension, so there will be no further extension at all (and we need time for grading). Start the project early, and you should finish the demo by Week 10. There is no designated lab time in Week 11. In Week 11, we do not guarantee any extra TA time, other than 1 or 2 office hours offered by each TA.

Appendix (ALU block diagram and ALU truth table):

ALU Block Diagram



ALU Truth Table

ADD	NEG	SUB	OUT	Operation
0	0	0	A + B	Add
0	1	0	Don't Care	No Operation
1	0	1	A - B	Subtract
1	1	0	-A	2's Complement
1	1	1	A	Pass A

Z=1 if and only if OUT=0
N=1 if and only if OUT is negative.