CSEN 122L: Project (Spring 2025)

Design a Structural Model of a Pipelined CPU

The project is to design a structural model of a pipelined CPU with 13 instructions using Verilog HDL.

Description:

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 as `x0`, `x1`, `x2`, ..., `x63`.
- PC: 32 bits
- Instruction format: Each instruction is 32-bit wide, and consists of five fields: `opcode`, `rd`, `rs`, `rt`, and `unused`. The format is as follows.

`Opcode (4 bits)`	`rd (6 bits)`	`rs (6 bits)`	`rt (6 bits)`	`unused(10 bits)`

The 13 instructions are defined in following table.

Instruction	Symbol	Opcode	rd	rs	rt	Function
No operation	**`NOP`**	`0000`	**X**	**X**	**X**	**`No operation`**
Save PC	**`SVPC rd,y`**	`1111`	`rd`		**y**	`xrd ← PC + y` (22 bits)
Load	**`LD rd,rs`**	`1110`	`rd`	`rs`	**X**	`xrd ← M[xrs]`
Store	**`ST rt,rs`**	`0011`	**X**	`rs`	`rt`	`M[xrs] ← rt`
Add	**`ADD rd,rs,rt`**	`0100`	`rd`	`rs`	`rt`	`xrd ← xrs + xrt`
Increment	**`INC rd,rs,y`**	`0101`	`rd`	`rs`	**y**	`xrd ← xrs + y` (16 bits)
Negate	**`NEG rd,rs`**	`0110`	`rd`	`rs`	**X**	`xrd ← - xrs`
Subtract	**`SUB rd,rs,rt`**	`0111`	`rd`	`rs`	`rt`	`xrd ← xrs - xrt`
Jump	**`J rs`**	`1000`	**x**	`rs`	**x**	`PC ← xrs`
Jump memory	**`JM rs`**	`1010`	**X**	`rs`	**X**	`PC ← M[xrs]`
Branch if zero	**`BRZ rs`**	`1001`	**X**	`rs`	**X**	`PC ← xrs`, **if Z=1**
Branch if negative	**`BRN rs`**	`1011`	**X**	`rs`	**X**	`PC ← xrs`, **if N=1**
MIN	**`MIN rd,rs,rt`**	`0001`	`rd`	`rs`	`rt`	**`See *`**

* `rd` = min(`mem[rs]`, `mem[rs+1]`, `mem[rs+2]`, `...`, `mem[rs+rt-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 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 of 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 two assembly programs (1) min (without using the MIN instruction) (2) vector addition, in the SCU ISA

Write two assembly programs: Min program and Vector addition program, using the 12 base instructions (`NOP`, `SVPC`, `LD`, `ST`, `ADD`, `INC`, `NEG`, `SUB`, `J`, `JM`, `BRZ`, `BRN`) in the SCU ISA.

You can use a text editor or a word processor. Submit it to Camino.

(1) min program

Write an assembly program that finds the minimum element value from a given array, without using the specialized MIN instruction,

$$a_{min} = min(a_0, a_1, a_2, a_3, ..., a_{n-1})$$

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

(2) vector add program

Write an assembly program that does vector addition, such as

$$c_i = a_i + b_i \text{ , where } i = 0, ... , n-1$$

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

These two programs will also be used to test your CPU.

Assignments and Submission:

1. The two assembly codes, for (1) min program (2) 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**. (9 pts in CSEN122 Lecture)

2. Your datapath and control (including the truth table) for the SCU ISA (excluding MIN instruction) by **Friday of Week 8** to **Dr. Cho (to the Lecture Camino Section)**. Upload your datapath to Camino CSEN 122 lecture. **One submission per team**. (15 pts in CSEN122 Lecture); Use the ALU attached in the last page.

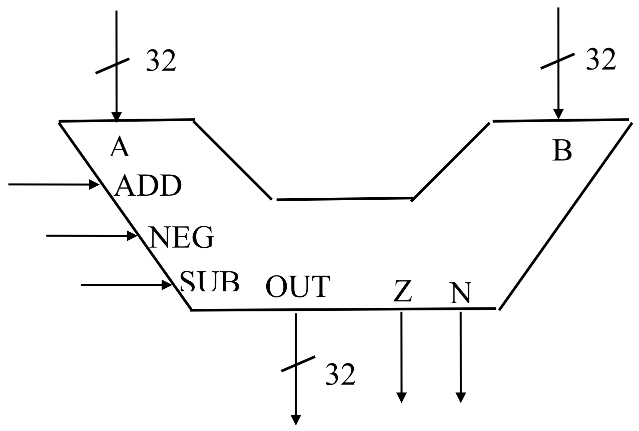
3. Report (30 pts in CSEN122 Lab):
   Submit a written final report to the **TA (to the Lab Camino section)** by **Tuesday of Week 11**, including the following
   - Abstract (short description or outline of the project),
   - Detailed description of the CPU design including the datapath and the truth table of your control.
   - Test benchmarks/waveforms verifying the functions,
   - Two assembly programs from Assignment (1) 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.

4. Codes & Demo (30 pts in CSEN122 Lab):
   You will submit your working program codes to the **TA (to the Lab Camino section)** by **Friday of Week 10**. And you will also demo your working programs to the **TA (during the Lab times or TA office hours)** by **Friday of Week 10**. During the demo, the TA will provide you *n* random numbers 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 answer related project questions. The purpose of these questions is to make sure that you understand the project thoroughly.

Appendix (ALU block diagram and ALU truth table):

ALU Block Diagram



ALU Truth Table

ADD	NEG	SUB	OUT	Operation
0	0	0	B+A	add
1	1	0	-B	2's complement
1	0	1	B-A	subtract
0	1	0	don't care	no operation
1	1	1	A	Pass A

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