Counter-Controlled Loop Unit (CCLU) Verification Plan

CCLU Specifications

CCLU is a special processor's hardware unit that implements counter-controlled loop (CCL) instructions. CCL is a loop instruction whose number of iterations is known at compile time. These loops can be nested. The CCLU is a stack-based unit. There is a stack entry for each CCL instruction. The syntax of the CCL instruction is as follows:

```
ccl target, Rs
```

`target`: the loop target address, i.e., the memory address of the first instruction in the loop block (body).

`Rs:` the number of the register that contains the number of loop iterations.

Here is an example of a code snippet that uses **`ccl`**

```
L1:
     Instruction 1;
     Instruction 2;
     Instruction 3;
ccl L1, Rs; Rs value is considered only when the loop instruction
          ; entry is pushed on the stack at the first iteration
```

In addition to the **`ccl`** instruction, CCLU supports:

1. break instruction (**`break target`**), which terminates the current loop instruction prematurely. The target is the address of the next instruction right after the **`ccl`** instruction to be terminated prematurely.
2. continue instruction (**`continue target`**), which terminates the current loop iteration and transfers the control to the next iteration. The target is the address of the first instruction of the **`ccl`** instruction block (body). Here is an example:

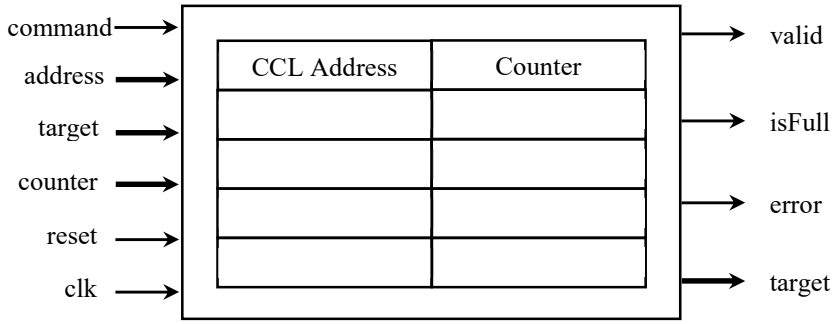
```
L1:
```

```
        Instruction 1;
        Instruction 2;
        L2:
              Instruction 21;
              continue L2;
              Instruction 22;
              Break L3;
              Instruction 23;
        ccl L2, R1
        L3: Instruction 3;
ccl L1, R0
```

The stack entry contains the following two fields. Assume that these two fields are unsigned 32-bit integers:

1. The CCL's address in memory (or part of the address). This address will be used as the instruction ID.
2. The CCL's counter, i.e., the number of the remaining loop iterations.

The figure below shows an abstract bock diagram of this CCLU.



The CCLU stack has a limited size, say **16** entries. Thus, if there is a nested loop that has more than 16 nests, then the stack becomes full when the number of entries reaches 16, and hence any extra CCL entry is pushed on the top of the stack extension in memory.

CCLU input ports:

1. **command:** it is a 2-bit signal. The table below shows the possible values of this signal and their effect

Command Value	Effect
00	Do nothing
01	A **`ccl`** or a **`continue`** instruction needs to be processed.
10	A **`break`** instruction that terminates the current **`ccl`** instruction prematurely. It requires removing the top element of the stack, and the output target address will be the input target address of the **`break`** instruction.
11	Error

2. **address**: the address of the current **`ccl`** instruction. This address is ignored in the case of a **`break`** instruction.
3. **target**: the target address of the current control instruction (**`ccl`**, **`break`**, or **`continue`**)
4. **counter**: the initial number of loop iterations. It is fetched from the register file. It is considered for the **`ccl`** instruction first iteration. It is ignored, otherwise.
5. **reset**: delete all elements of the stack.
6. **clk**: the system clock

CCLU output ports:

1. **valid:** "1" means the value of the output target address port is valid, otherwise this value is invalid.
2. **isFull:** indicates that the CCLU's stack is full, and the processor needs to use the stack extension in memory for additional **`ccl`** instructions.

3. **error:** "1" indicates that there is some error.
4. **target:** the target address of the current control instruction

CCLU Operation Details:

1. If the **`command`** input signal has the value of "00", nothing happens
2. If the **`command`** input signal has the value of "11", the error signal is asserted in the same clock cycle.
3. If the **`command`** input signal has the value of "01", this means a **`ccl`** or a **`continue`** instruction needs to be processed. the input address is compared to the CCL address in the top element of the stack:
   - If they are not equal, then this **`ccl`** instruction is new to the stack.
     - If the input counter is zero, the error signal is asserted.
     - If the input counter equals "1", then this loop has only one iteration, and hence there no need to add an entry on the top of the stack. The input target address becomes the output target address in the next clock cycle, and the output valid bit is asserted.
     - Otherwise, if the stack is full, then the error signal is asserted in the next clock cycle.
     - If the stack is not full, the input counter is decremented, and a new element is pushed on the top of the stack for this **ccl** instruction. Moreover, the input target address becomes the output target address in the next clock cycle, and the output valid bit is asserted.
   - If they are equal, this means that this **`ccl`** instruction already exists on the stack. If the counter field of the top element of the stack equals "1", this means that this is the last loop iteration, and hence the top element of the stack is removed. Otherwise, the counter field of the top element of the stack is decremented. Moreover, the input target address becomes the output target address in the next clock cycle, and the output valid bit is asserted.

4. If the **`command`** input signal has the value of “10”, this means a **`break`** instruction, then the top element of the stack is removed. the input target address becomes the output target address in the next clock cycle, and the output valid bit is asserted.
5. The **`reset`** signal empties the stack and it takes only one clock cycle.
6. If the **`reset`** has the value of “1” and the **`command`** input signal has a valid value in the same clock cycle, then the error signal is asserted.