Assignment 3: Gem5 with RISC-V and Adding Custom Instructions

ANANNYA MATHUR

Step 1:

We clone the riscv toolchain and add the definition of our custom instruction of combination formula to riscv-opcodes array present in riscv-gnu-toolchain/binutils/opcodes/riscv-opc.c:

```
const struct riscv_opcode riscv_opcodes[] =
{

/* fact, combination formula declaration : */
{"comb_form",      0, INSN_CLASS_I,       "d,s,t",         MATCH_COMBFORM, MASK_COMBFORM, match_opcode, 0 },
{"fact",      0, INSN_CLASS_I,       "d,a",         MATCH_FACT, MASK_FACT, match_opcode, 0 },
```

MATCH and MASK values have to be defined at riscv-gnu-toolchain/binutils/include/opcode/riscv-opc.h:

```
/* fact and combination formula declaration: */

#define MATCH_COMBFORM 0x6027
#define MASK_COMBFORM 0xfe00707f
#define MATCH_FACT 0x27
#define MASK_FACT 0x7f
```

```
/* fact and combination formula declaration*/
DECLARE_INSN(comb_form, MATCH_COMBFORM, MASK_COMBFORM)
DECLARE_INSN(fact, MATCH_FACT, MASK_FACT)
```

Once these changes are made, the riscv toolchain must be rebuilt.

Step 2: Writing the c code to evaluate the coefficients in the Binary Expansion series using nCr as a custom instruction-

For the sake of performance comparison, two c codes are maintained; one makes use of nCr as a custom instruction, and the other is a standard c code sans custom instruction.

comb_form.c (with nCr as a custom instruction):

```
#include <stdio.h>

int main()
{
   int c, A, X, n;

   A=100;
   X=10;
   n=10;

   printf ("Printing the equation \n");
   printf("(%u + %u)^ %u = ", A, X, n);

   for (int x=0; x<=n; x++)
   {
      asm
      (
         "comb_form   %[z], %[x], %[y]\n\t"
         : [z] "=r" (c)
         : [x] "r" (n), [y] "r" (x)

      );

      if (x == n)
      {
         printf("%u*%u^%u*%u^%u \n", c,A,(n-x),X,x);
      }
      else
      {
         printf("%u*%u^%u*%u^%u + ", c,A,(n-x),X,x);
      }
   }

  printf("_____ \n"); }
```

Upon executing
/opt/riscv/bin/riscv64-unknown-elf-gcc comb_form.c -o m1.o
/opt/riscv/bin/riscv64-unknown-elf-objdump -D m1.o > comb_form.dump,
we can check the presence of our custom instruction "comb_form" →

```
101f4:   00e7e7a7          comb_form   a5,a5,a4
101f8:   fcf42e23          sw a5,-36(s0)
101fc:   fec42783          lw a5,-20(s0)
10200:   873e              mv a4,a5
10202:   fe042783          lw a5,-32(s0)
10206:   2701              sext.w   a4,a4
10208:   2781              sext.w   a5,a5
1020a:   02f71963          bne   a4,a5,1023c <main+0x96>
1020e:   fe042783          lw a5,-32(s0)
```

Step 3: Configuring nCr custom instruction with gem5-

```
comb_form:
OPCODE5: 0x09
FUNCT3: 0x6
FUNCT7: 0x0

fact:
OPCODE5: 0x09
FUNCT3: 0x0
FUNCT7: 0x0
```

To define the function of our custom instruction, we make changes in the decoder.isa file present at gem5/src/arch/riscv/isa:

The instruction must be defined within the 0x3: decode OPCODE5{0x09: decode FUNCT3 {}} block:

```
    // Add the code of factorial:
          format ROp {
```

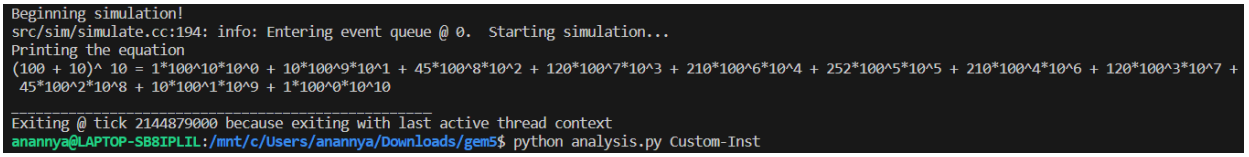
```
    0x0: decode FUNCT7 {
        0x0: fact({{

            int ans=1;
            int a=Rs1_sd;
            for (int i=1;i<=a;i++)
            {
                ans=ans*i;
            }

            Rd=ans;

        }});
    }

// Add the code of combination function:
  0x6: decode FUNCT7 {
    0x0: comb_form({{


        int n= Rs1_sd;
        int r= Rs2_sd;

           int term1=1;

            for (int i=1;i<=n;i++)
            {
                term1=term1*i;
            }

        int term2=1;

            for (int i=1;i<=(n-r);i++)
            {
                term2=term2*i;
            }

        int term3=1;

            for (int i=1;i<=(r);i++)
            {
                term3=term3*i;
            }
        int ans= term1/(term2*term3);

        Rd=ans;
```

```
                }});
            }
        ////////////////////////////////////////
            }
```

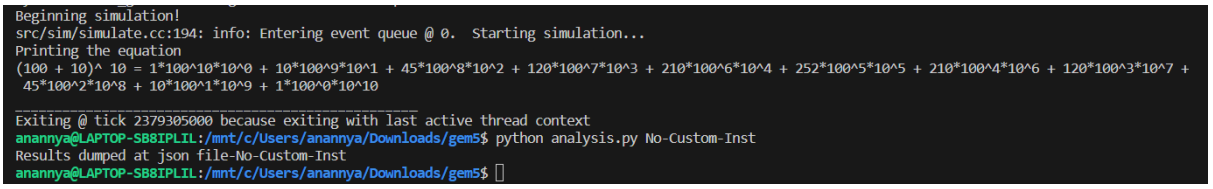
Gem5 can now be rebuilt.

Step 4: Running simulation

With nCr as a custom instruction:



Without any custom instruction,



gem5/configs/learning_gem5/part1/simple-riscv.py has been used for generating the results.

	With/Without nCr as custom inst	simSeconds
0	Custom-Inst	0.002145
1	No-Custom-Inst	0.002379

It can be observed that upon using nCr as a custom instruction, the simulation seconds go down to 0.002145s from 0.002379s, improving the code's performance.