# 109 - 2 Digital System Design Final Project

# Extension – Compressed Instructions

*Announced at May 20, 2021*

**TA Information**
**馬咏治**
**kane@access.ee.ntu.edu.tw**

## 1. Problem Description

A few decades ago, the memory is a rather limited resource as the process technology is not that advanced. Ideas of compressed instructions then emerge to save memory space. Though nowadays the memory is not as expensive and physically limited, demands on the memory space efficiency do not seem to decline, due to the flourish of a specific class of embedded systems. Wearable techs are notable examples of such. The applications can be enormously diverse, but they all share the common trend and demands: the decrease in both size and power consumption. Supports for compressed instructions can be effective aids to the code size reduction and power efficiency. As we can see, in addition to some basic supports, the newborn, elaborately designed RISC-V ISA also involves a compressed instruction set as one of the standard extension "C", which reflects its great potential. In this part, your challenge is to implement the following 16 C-instructions as the extension to your base RISC-V core.

| | | | |
|---|---|---|---|
| C.ADD | C.ANDI | C.LW | C.J |
| C.MV | C.SLLI | C.SW | C.JAL |
| C.ADDI | C.SRLI | C.BEQZ | C.JR |
| C.NOP | C.SRAI | C.BNEZ | C.JALR |

## 2. Specification of C-Instructions

Information of these compressed instructions are given in Appendix A to C. In addition, there are quick notes in "I_mem_compression" for convenience (See **Testbench**). For further details, please refer to the chapter 12 of *The RISC-V Instruction Set Manual*.

## 3. Implementation

To add a C module to the base, consider the followings:

**(a) Extract information encoded in C-instructions.**

Obviously, the decoder you have done in the baseline cannot correctly decode any compressed instructions. You have two choices: implement a dedicated decoder or a decompressor. As the name suggests, a decompressor expands compressed instructions into their 32-bit counterparts, which are later sent to the original decoder. This is a potentially neater way.

**(b) PC increment.**

PC is a special register indicating the location of the current instruction. Thus, if the current instruction is a compressed one, the next location should be PC+2 rather than the usual PC+4.

**(c) Address alignment issues.**

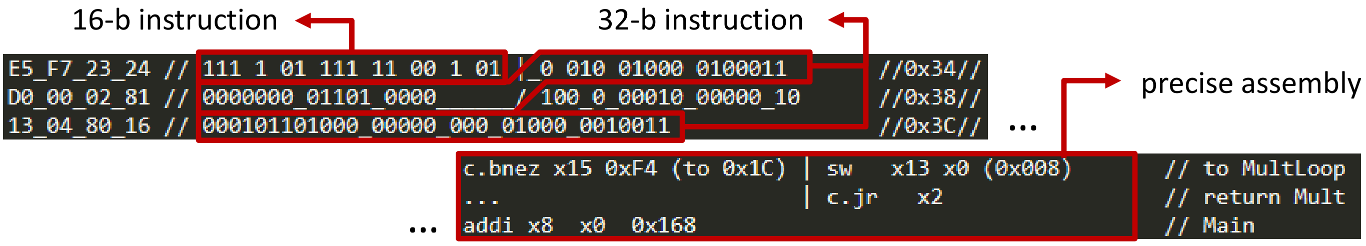
There is no constraint on the placement of instructions. Specifically, instructions can be located at unaligned addresses, i.e. PC = 4k+2, and thus a 32-bit one may cross a four-byte boundary. In this case, it may take two cycles to fetch a single instruction. Remember that usually read addresses specified by your processor are multiples of four. Hence, the PC does not necessarily equal the read address any longer. Of course, improvements to the microarchitecture, if any, may lead to additional problems. You should manage to handle them.

## 4. Testbench

"**I_mem_compression**", "**I_mem_decompression**", and "**TestBed_compression.v**" collectively form a set of test module, which you can find under "DEADxF625/". To be more specific, the I_mem_decompression is the pure-RV32I counterpart of the I_mem_compression. Hence, before going on and getting your hand dirty, it is highly recommended to check whether your baseline core pass the decompression.

Hint: +**define+(de)compression** in the simulation command to couple the testbench.

Here is the guide for "I_mem_compression":



For convenience, some notes are also included for quick reference:

```
Notes:
    Required supporting compression instructions with offset settings:
        c.nop                    : decode to addi x0 x0 0x000
        c.add  , c.mv            : no offset
        c.addi , c.andi*         : sign-extended offset[ 5:0] ( 6 bits)
        c.slli , c.srli*, c.srai*: shamt[5:0], shamt[5] must be 0
        c.sw*  , c.lw*           : zero-extended offset[ 6:2] ( 5 bits)
        c.beqz*, c.bnez*         : sign-extended offset[ 8:1] ( 8 bits)
        c.j    , c.jal^          : sign-extended offset[11:1] (11 bits)
        c.jr   , c.jalr^         : no offset
    *: limited register index (x8 ~ x15)
    ^: stores pc+2 to x1
```

**Test Program Generation.**

This part illustrates how to generate different test cases:

A. *Execute "compression_generate.py" in the directory "generate/":*
   -Python (version = 3.x)
   -Modify a, b
   -I_mem_(de)compression_ref & TestBed_compression_ref should be placed in the same folder
   **-I_mem_(de)compression** & **TestBed_compression** will be generated (Provided file in "DEADxF625/")

B. *+ define+(de)compression in ncverilog simulation command*

## 5. Requirements

**Design.** You should pass the given testbench "Final_tb.v" with the instruction memory and the testbed under the DEADxF625/ directory. **No need to check the validity of the input instructions (e.g. all-zero instruction) and the constraints of values of each field (e.g. src ≠ 0).** For simplicity, the validity of instructions in our testbenches is guaranteed.

**Report.** In the compression part of your final report "**Report.pdf**", we suggest you have a detailed discussion on the followings:

(a) What is the advantage of the C extension? Verify it with your simulation results.
(b) How do you design your chip to support this extension?
(c) Any improvement on the performance, especially on how you reduce the number of cycles to complete the simulation.

(d) What you have learned?

(e) Other detailed discussion will be appreciated

**Evaluation Metrics.**

$$A(compressed\ design - baseline\ design) \times T(given\ compressed\ testbench)$$

Don't worry about the performance evaluation. It is just one of the criteria. **Focus more on what you design to solve problems you face.**

## 6. Reference

[1] The RISC-V Instruction Set Manual Volume I: User Level ISA (Version 2.2)

# Appendix A.    Assembly
## Compressed Instructions with Their Unique 32-bit Expansions

| C-instr. | Base instruction (Assembly) |
|---|---|
| **C.LW** | lw rd’, offset[6:2](rs1’) |
| **C.SW** | sw rs2’ offset[6:2](rs1’) |
| **C.BEQZ** | beq rs1’, x0, offset[8:1] |
| **C.BNEZ** | bne rs1’, x0, offset[8:1] |
| **C.J** | jal x0, offset[11:1] |
| **C.JAL** | jal x1, offset[11:1] |
| **C.JR** | jalr x0, rs1, 0 |
| **C.JALR** | jalr x1, rs1, 0 |
| **C.ADDI** | addi rd, rd, nzimm[5:0] |
| **C.ANDI** | andi rd’, rd’, imm[5:0] |
| **C.SLLI** | slli rd, rd, shamt[5:0] |
| **C.SRLI** | srli rd’, rd’, shamt[5:0] |
| **C.SRAI** | srai rd’, rd’, shamt[5:0] |
| **C.MV** | add rd, x0, rs2 |
| **C.ADD** | add rd, rd, rs2 |
| **C.NOP** | addi x0, x0, 0 |

# Appendix B. RVC Instruction Subsets

| 15–13 | 12–10 | 9–7 | 6–5 | 4–2 | 1–0 | |
|---|---|---|---|---|---|---|
| 000 | 0 | | | 0 | 00 | *Illegal instruction* |
| 000 | nzuimm[5:4\|9:6\|2\|3] | | | rd′ | 00 | C.ADDI4SPN *(RES, nzuimm=0)* |
| 001 | uimm[5:3] | rs1′ | uimm[7:6] | rd′ | 00 | C.FLD *(RV32/64)* |
| 001 | uimm[5:4\|8] | rs1′ | uimm[7:6] | rd′ | 00 | C.LQ *(RV128)* |
| 010 | uimm[5:3] | rs1′ | uimm[2\|6] | rd′ | 00 | C.LW |
| 011 | uimm[5:3] | rs1′ | uimm[2\|6] | rd′ | 00 | C.FLW *(RV32)* |
| 011 | uimm[5:3] | rs1′ | uimm[7:6] | rd′ | 00 | C.LD *(RV64/128)* |
| 100 | — | | | | 00 | *Reserved* |
| 101 | uimm[5:3] | rs1′ | uimm[7:6] | rs2′ | 00 | C.FSD *(RV32/64)* |
| 101 | uimm[5:4\|8] | rs1′ | uimm[7:6] | rs2′ | 00 | C.SQ *(RV128)* |
| 110 | uimm[5:3] | rs1′ | uimm[2\|6] | rs2′ | 00 | C.SW |
| 111 | uimm[5:3] | rs1′ | uimm[2\|6] | rs2′ | 00 | C.FSW *(RV32)* |
| 111 | uimm[5:3] | rs1′ | uimm[7:6] | rs2′ | 00 | C.SD *(RV64/128)* |

| 15–13 | 12 | 11–10 | 9–7 | 6–5 | 4–2 | 1–0 | |
|---|---|---|---|---|---|---|---|
| 000 | 0 | 0 | | 0 | | 01 | C.NOP |
| 000 | nzimm[5] | rs1/rd≠0 | | nzimm[4:0] | | 01 | C.ADDI *(HINT, nzimm=0)* |
| 001 | imm[11\|4\|9:8\|10\|6\|7\|3:1\|5] | | | | | 01 | C.JAL *(RV32)* |
| 001 | imm[5] | rs1/rd≠0 | | imm[4:0] | | 01 | C.ADDIW *(RV64/128; RES, rd=0)* |
| 010 | imm[5] | rd≠0 | | imm[4:0] | | 01 | C.LI *(HINT, rd=0)* |
| 011 | nzimm[9] | 2 | | nzimm[4\|6\|8:7\|5] | | 01 | C.ADDI16SP *(RES, nzimm=0)* |
| 011 | nzimm[17] | rd≠{0, 2} | | nzimm[16:12] | | 01 | C.LUI *(RES, nzimm=0; HINT, rd=0)* |
| 100 | nzuimm[5] | 00 | rs1′/rd′ | nzuimm[4:0] | | 01 | C.SRLI *(RV32 NSE, nzuimm[5]=1)* |
| 100 | 0 | 00 | rs1′/rd′ | 0 | | 01 | C.SRLI64 *(RV128; RV32/64 HINT)* |
| 100 | nzuimm[5] | 01 | rs1′/rd′ | nzuimm[4:0] | | 01 | C.SRAI *(RV32 NSE, nzuimm[5]=1)* |
| 100 | 0 | 01 | rs1′/rd′ | 0 | | 01 | C.SRAI64 *(RV128; RV32/64 HINT)* |
| 100 | imm[5] | 10 | rs1′/rd′ | imm[4:0] | | 01 | C.ANDI |
| 100 | 0 | 11 | rs1′/rd′ | 00 | rs2′ | 01 | C.SUB |
| 100 | 0 | 11 | rs1′/rd′ | 01 | rs2′ | 01 | C.XOR |
| 100 | 0 | 11 | rs1′/rd′ | 10 | rs2′ | 01 | C.OR |
| 100 | 0 | 11 | rs1′/rd′ | 11 | rs2′ | 01 | C.AND |
| 100 | 1 | 11 | rs1′/rd′ | 00 | rs2′ | 01 | C.SUBW *(RV64/128; RV32 RES)* |
| 100 | 1 | 11 | rs1′/rd′ | 01 | rs2′ | 01 | C.ADDW *(RV64/128; RV32 RES)* |
| 100 | 1 | 11 | — | 10 | — | 01 | *Reserved* |
| 100 | 1 | 11 | — | 11 | — | 01 | *Reserved* |
| 101 | imm[11\|4\|9:8\|10\|6\|7\|3:1\|5] | | | | | 01 | C.J |
| 110 | imm[8\|4:3] | | rs1′ | imm[7:6\|2:1\|5] | | 01 | C.BEQZ |
| 111 | imm[8\|4:3] | | rs1′ | imm[7:6\|2:1\|5] | | 01 | C.BNEZ |

| 15–13 | 12 | 11–7 | 6–2 | 1–0 | |
|---|---|---|---|---|---|
| 000 | nzuimm[5] | rs1/rd≠0 | nzuimm[4:0] | 10 | C.SLLI *(HINT, rd=0; RV32 NSE, nzuimm[5]=1)* |
| 000 | 0 | rs1/rd≠0 | 0 | 10 | C.SLLI64 *(RV128; RV32/64 HINT; HINT, rd=0)* |
| 001 | uimm[5] | rd | uimm[4:3\|8:6] | 10 | C.FLDSP *(RV32/64)* |
| 001 | uimm[5] | rd≠0 | uimm[4\|9:6] | 10 | C.LQSP *(RV128; RES, rd=0)* |
| 010 | uimm[5] | rd≠0 | uimm[4:2\|7:6] | 10 | C.LWSP *(RES, rd=0)* |
| 011 | uimm[5] | rd | uimm[4:2\|7:6] | 10 | C.FLWSP *(RV32)* |
| 011 | uimm[5] | rd≠0 | uimm[4:3\|8:6] | 10 | C.LDSP *(RV64/128; RES, rd=0)* |
| 100 | 0 | rs1≠0 | 0 | 10 | C.JR *(RES, rs1=0)* |
| 100 | 0 | rd≠0 | rs2≠0 | 10 | C.MV *(HINT, rd=0)* |
| 100 | 1 | 0 | 0 | 10 | C.EBREAK |
| 100 | 1 | rs1≠0 | 0 | 10 | C.JALR |
| 100 | 1 | rs1/rd≠0 | rs2≠0 | 10 | C.ADD *(HINT, rd=0)* |

※ Some instructions in Quadrant 2 are not listed here.

## Appendix C.  RV32I Instruction Subset

| | | | | | | |
|---|---|---|---|---|---|---|
| imm[20\|10:1\|11\|19:12] | | | | rd | 1101111 | JAL |
| imm[11:0] | | rs1 | 000 | rd | 1100111 | JALR |
| imm[12\|10:5] | rs2 | rs1 | 000 | imm[4:1\|11] | 1100011 | BEQ |
| imm[12\|10:5] | rs2 | rs1 | 001 | imm[4:1\|11] | 1100011 | BNE |
| imm[12\|10:5] | rs2 | rs1 | 100 | imm[4:1\|11] | 1100011 | BLT |
| imm[12\|10:5] | rs2 | rs1 | 101 | imm[4:1\|11] | 1100011 | BGE |
| imm[12\|10:5] | rs2 | rs1 | 110 | imm[4:1\|11] | 1100011 | BLTU |
| imm[12\|10:5] | rs2 | rs1 | 111 | imm[4:1\|11] | 1100011 | BGEU |
| imm[11:0] | | rs1 | 000 | rd | 0000011 | LB |
| imm[11:0] | | rs1 | 001 | rd | 0000011 | LH |
| imm[11:0] | | rs1 | 010 | rd | 0000011 | LW |
| imm[11:0] | | rs1 | 100 | rd | 0000011 | LBU |
| imm[11:0] | | rs1 | 101 | rd | 0000011 | LHU |
| imm[11:5] | rs2 | rs1 | 000 | imm[4:0] | 0100011 | SB |
| imm[11:5] | rs2 | rs1 | 001 | imm[4:0] | 0100011 | SH |
| imm[11:5] | rs2 | rs1 | 010 | imm[4:0] | 0100011 | SW |
| imm[11:0] | | rs1 | 000 | rd | 0010011 | ADDI |
| imm[11:0] | | rs1 | 010 | rd | 0010011 | SLTI |
| imm[11:0] | | rs1 | 011 | rd | 0010011 | SLTIU |
| imm[11:0] | | rs1 | 100 | rd | 0010011 | XORI |
| imm[11:0] | | rs1 | 110 | rd | 0010011 | ORI |
| imm[11:0] | | rs1 | 111 | rd | 0010011 | ANDI |
| 0000000 | shamt | rs1 | 001 | rd | 0010011 | SLLI |
| 0000000 | shamt | rs1 | 101 | rd | 0010011 | SRLI |
| 0100000 | shamt | rs1 | 101 | rd | 0010011 | SRAI |
| 0000000 | rs2 | rs1 | 000 | rd | 0110011 | ADD |
| 0100000 | rs2 | rs1 | 000 | rd | 0110011 | SUB |
| 0000000 | rs2 | rs1 | 001 | rd | 0110011 | SLL |
| 0000000 | rs2 | rs1 | 010 | rd | 0110011 | SLT |
| 0000000 | rs2 | rs1 | 011 | rd | 0110011 | SLTU |
| 0000000 | rs2 | rs1 | 100 | rd | 0110011 | XOR |
| 0000000 | rs2 | rs1 | 101 | rd | 0110011 | SRL |
| 0100000 | rs2 | rs1 | 101 | rd | 0110011 | SRA |
| 0000000 | rs2 | rs1 | 110 | rd | 0110011 | OR |
| 0000000 | rs2 | rs1 | 111 | rd | 0110011 | AND |

## Appendix D.  Instruction types for RVC

| Format | Meaning | 15 14 13 | 12 | 11 10 | 9 8 7 | 6 5 | 4 3 2 | 1 0 |
|---|---|---|---|---|---|---|---|---|
| CR | Register | funct4 | | rd/rs1 | | rs2 | | op |
| CI | Immediate | funct3 | imm | rd/rs1 | | imm | | op |
| CSS | Stack-relative Store | funct3 | imm | | | rs2 | | op |
| CIW | Wide Immediate | funct3 | imm | | | | rd′ | op |
| CL | Load | funct3 | imm | | rs1′ | imm | rd′ | op |
| CS | Store | funct3 | imm | | rs1′ | imm | rs2′ | op |
| CB | Branch | funct3 | offset | | rs1′ | offset | | op |
| CJ | Jump | funct3 | jump target | | | | | op |