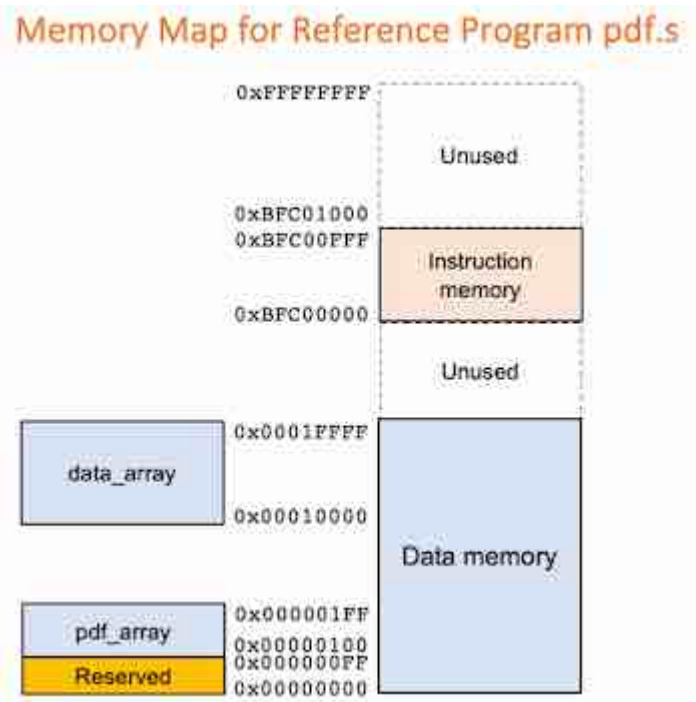# EIE2 Instruction Set Architecture & Compiler (IAC)

Reference Program for Team Project

Peter Cheung, V1.0 - 9 Dec 2022

This assembly language program, **pdf.s** is a common test for team's CPU design. The program reads 8-bit data from a data memory array starting at 0x10000 and produces a probability distribution function (PDF) in a pdf array at 0x100.

New Memory Map

To run this program, you need to increase your data memory space according to the memory map shown here. The data memory space from 0x00000000 to 0x0001FFFF is used in the following ways:

- 0x00000000 to 0x000000FF is 256 bytes reserved for other purpose (not used here)
- 0x00000100 to 0x000001FF is 256 bytes to store the probability distribution function (pdf array)
- 0x00010000 to 0x0001FFFF is a 65,536-bytes array that stores the data the PDF of which is to be calculated

The rest of the data memory space is not used by this program.

How is probability distribution function calculated?

To find the PDF of an array of data (e.g. signals), we need to reserve a block of memory to store the distribution. Each entry is a "bin", storing how many times a data value occurred. Since the data is 8-bit

wide, the range of data value is from 0 to 255. We therefore only need an array of size 256.

To start, the count in each bin of the data array must be initialised to zero. We then read the data one byte at a time. If a data value is v, we increment by one the "frequency count" in bin base_address + v. We stop building the PDF array if any bin reaches a count of 255.

The Reference Program *pdf.s*

Here is the reference program with line number.

```
1    .text
2    .equ base_pdf, 0x100
3    .equ base_data, 0x10000
4    .equ max_count, 200
5    main:
6        JAL       ra, init  # jump to init, ra and save position to ra
7        JAL       ra, build
8    forever:
9        JAL       ra, display
10       J         forever
11
12   init:         # function to initialise PDF buffer memory
13       LI        a1, 0x100              # loop_count a1 = 256
14   _loop1:                              # repeat
15       ADDI      a1, a1, -1             #     decrement a1
16       SB        zero, base_pdf(a1)  #     mem[base_pdf+a1) = 0
17       BNE       a1, zero, _loop1    # until a1 = 0
18       RET
19
20   build:        # function to build prob dist func (pdf)
21       LI        a1, base_data         # a1 = base address of data array
22       LI        a2, 0                 # a2 = offset into of data array
23       LI        a3, base_pdf          # a3 = base address of pdf array
24       LI        a4, max_count         # a4 = maximum count to terminate
25   _loop2:                             # repeat
26       ADD       a5, a1, a2            #     a5 = data base address + offset
27       LBU       t0, 0(a5)             #     t0 = data value
28       ADD       a6, t0, a3            #     a6 = index into pdf array
29       LBU       t1, 0(a6)             #     t1 = current bin count
30       ADDI      t1, t1, 1             #     increment bin count
31       SB        t1, 0(a6)             #     update bin count
32       ADDI      a2, a2, 1             #     point to next data in array
33       BNE       t1, a4, _loop2     # until bin count reaches max
34       RET
35
36   display:      # function send PDF array value to a0 for display
37       LI        a1, 0                 # a1 = offset into pdf array
38       LI        a2, 255               # a2 = max index of pdf array
39   _loop3:                             # repeat
40       LBU       a0, base_pdf(a1)   #   a0 = mem[base_pdf+a1)
41       addi      a1, a1, 1             #   incr
42       BNE       a1, a2, _loop3     # until end of pdf array
43       RET
44
```

It consists of 4 segments:

1. The main program loop,
2. The init function,
3. The build function,
4. The display function.

There are a number of important issues to note:

- Lines 1-4 are assembler directives. **.text** means what follows is a program section. **.equ** defines symbolic constants.

- Lines 5-10 are the main program which initialize the pdf array bin counts to zero, then build the pdf function, and finally go into an infinite loop that keep outputing the pdf array to a0, which can be displayed on Vbuddy in the testbench.

- I have used pseudoinstructions throughout to make the program more readable. For example, line 21

```
LI  a1, base_data
```

is translated to the real instruction:

```
LUI    a1, 0x10
```

The 32-bit constant **base_data** is 0x10000, which is larger than 12 bits allowed by I-type instructions. Therefore, we need to use the LUI instruciton. This requires the programmer to understand how the LUI instruction works. The LI pseudoinstruction hides such details from the programmable. Of course, as a hardware architect, you NEED to know that this instruction is translated to a LUI instruction!

- To help you mapping all pseudoinstruction to real instructions, I have provided you with a Makefile in the reference folder, which not only assembles the pdf.s source code to a hex file ready to be loading into instruction memory, but also produces the assembly listing in a .asm file.

- To **make** everything, simply type the command:

```
make reference
```

- Provided in the repo are four data files each contains 65,536 bytes of signal data. They are:

    1. sine.mem - a sinewave signal.
    2. triangle.mem - a triangular wave signal.
    3. gaussian.mem - a noise signal with a gaussian distribution.
    4. noisy.mem - a noisy sinewave signal.

- You can read any of these data into the data memory data_array at the base address 0x10000 with the SystemVerilog directive:

```
$readmemh('xxxx.mem', data_array, 0x10000);
```

- This program has data or control hazards if run on a pipelined version of the RISC-V processor. You must add NOP yourself to get round the hazards.

- You are free to modify the pdf.s code but you should tell me what you have done and why.

- Evidence of this program working would be the plot of the PDF on Vbuddy. This can be done by plotting the value of top->a0 AFTER the function which builds the distribution is finished. Do not plot a0 value every cycle - this will take too long.