Death Machine CPU Manual

Alexander Jansiewicz
Clare O'Brien
Prof. Shudong Hao
CS 382-B
5 December, 2022

Table of Contents

I. Introduction to the Death Machine

Welcome to the Death Machine manual! In this guide, you will learn everything you need to know about the Death Machine. We'll start with the actual CPU itself; if you want to know more about writing and running a program, skip to page 3.

There are 8 supported instructions: ADD, SUB, LDR, STR, CBNZ, CBZ, BL, RET. The instructions have the same functions as they would in a typical ARMv8 assembly program. Please do not use any instructions other than these 8, as the CPU will not know what to do with them. For more information on formatting instructions, go to page 3.

The Death Machine has 8 registers, but only 6 are considered general purpose: that is, those registers can be written to without causing errors. X6 is the link register, which stores the return address of the correct instruction after using the BL instruction. It can be written to, but this is not advisable. X7 is the XZR register, the value of which remains 0 at all times. This register does not have write capabilities. X0 through X5 can be written to freely without causing any errors.

Since the INSTRUCTION_MEMORY has an address bit-width of 16 bits, 65,536 instructions can be written at a time. In any case where you have less than 65,536 instructions, any unused instructions will be filled with the dummy instruction ADD X7, X7, X7. Since X7 is XZR, and XZR does not have write capabilities, the instruction just adds 0 and 0 into an unwritable register. The DATA_MEMORY module also has 16 bits of address space, meaning you can have 65,536 16-bit data pieces at a time. Any unused data addresses are automatically filled with 0000.

There are 10 LED displays within the CPU itself. The ones labeled X0, X1, X2, X3, X4, X5, LR, and XZ display the value of each register at any given instruction in your program. The display at the top labeled "PC" displays the current value of the program counter. It is worth noting that the PC appears to count up in hexadecimal, so don't be alarmed when the PC goes from 9 to a. Registers also store data in hexadecimal, so keep this in mind when tracking calculations. Finally, the very rightmost display shows the current instruction type based on where the CPU is in the program.

On the left hand side, there are multiple circuits that we've used in the Death Machine. "main", "register_file", "control_unit", and "ALU" are pretty self explanatory. "display" was the circuit that we used to craft the LED displays. The circuit "hex_to_LED" takes a 16-bit hexadecimal number (such as the value in a register or in the program counter) and converts it into several 32-bit signals to be used by an LED display. "INSTR_LED" is a circuit that takes the opcode, converts it into its mnemonic (for use by an LED display), and then displays it.

Speaking of opcode, we will now break down how we convert an assembly instruction to machine code for the Death Machine CPU to read and calculate with. Each instruction will be converted to a 3-byte binary number (24 digits). The format of the machine code is shown below:

OPCODE [23:21]	→	Requires 3 bits since we have 8 instructions
TARGET [20:18]	→	Requires 3 bits since we have 8 registers
OPRND1 [17:15]	→	Requires 3 bits since we have 8 registers
IMMD12 [14:3]	→	Requires 12 bits for immediate headroom
OPRND2 [2:0	→	Requires 3 bits since we have 8 registers

The opcode for each instruction is written below:

ADD	= 000	**SUB**	= 001	**LDR**	= 010	**STR**	= 011
BL	= 100	**RET**	= 101	**CBZ**	= 110	**CBNZ**	= 111

Then the instruction needs to be converted to hexadecimal from binary so that instruction memory can read it. The following are some examples of how to take an instruction and translate it into machine code:

LDR X3, [X0, 2]	→	010 011 000 000000000010 000	→	4c0010
SUB X3, X3, X2	→	001 011 011 000000000000 010	→	2d8002
CBNZ X3, -2	→	111 011 000 111111111110 000	→	ec7ff0
BL 4	→	100 110 000 000000000100 000	→	980020
RET	→	101 110 000 000000000000 000	→	b80000

You won't need to do this, though; the assembler automatically handles the conversion process, converting your inputted instructions and data into an instruction.txt file and a data.txt file that the Death Machine can use. The next section will go over how to write the assembly instruction file to input to the assembler. Please make sure to *strictly* follow the rules listed below in order to allow for optimal Death Machine performance.

II. Crafting your .txt File: DOs and DON'Ts

DO: Name your .txt file "death_assembly.txt".

The assembler will only read from .txt files that have this *exact* name.

DO: Use only these specific instructions: ADD, SUB, LDR, STR, CBNZ, CBZ, BL, RET

Invalid instructions will throw an error. (Obviously.)

DO: Strictly follow this typing format for writing instructions:

ADD Rd, Rn, Rm SUB Rd, Rn, Rm LDR Rt, [Rn, imm12] STR Rt, [Rn, imm12] CBNZ Rt, imm12 CBZ Rt, imm12 BL imm12 RET

where Rd = Destination Register
Rn = First register operand
Rm = Second register operand
Rt = Target register
imm12 = 12 digit immediate (in binary)

Other formatting, such as using an immediate as an operand instead of a register for ADD/SUB or forgetting the brackets for STR/LDR will throw an error.

DO: Remember that instructions written in the .txt file are not case sensitive.

Instructions of the correct format will be accepted whether they are uppercase or lowercase.

DO: Use decimal immediates only within the range of [-2048, 2047].

Since the immediate only accepts up to 12 binary digits, using immediates outside of the specified range will throw an error.

DON'T: Add tabs or more / less than a single space in between instructions, registers or immediates.

DO: ADD X0, X0, X0

DON'T: ADD X0,X0, X0 // This will throw an error!

DON'T: Forget to add commas after registers.

DO: SUB X1, X1, X3

DON'T: SUB X1 X1 X3 // This will throw an error!

DON'T: Comment your code.

The assembler has no capacity for commenting; it will cause errors. Sorry.

DON'T: Write more than one instruction on each line.

DO: LDR X0, [X1, 0]

DON'T: LDR X1, [X0, 0] ADD X1, X1, X2 // This will throw an error!

DO: Remember that X7 is the register XZR and cannot be written to.

Register X7 is reserved for the XZR register, the value of which is strictly zero.

DON'T: Write "XZR" instead of "X7".

DO: ADD X7, X7, X7

DON'T: ADD XZR, XZR, XZR // This will throw an error!

DO: Remember that X6 is the link register and can be written to (carefully).

Register X6 is reserved for the linked address when using the BL instruction.

DON'T: Write "LR" instead of "X6".

DO: SUB X6, X6, X5

DON'T: SUB LR, LR, X5 // This will throw an error!

DO: Understand how to branch conditionally and unconditionally.

Instead of using labels, the Death Machine accepts immediates as branching offsets. Positive numbers will add to the program counter, moving forward (or downwards) in the code. Negative numbers subtract from the program counter, moving backward (or upwards) in the code. For example, if PC is at 2 and the instruction is BL 4, then PC will be updated to line 6, X6 will be updated to 3, and the RET instruction will use X6 to return to the line below the BL instruction. The same structure can be used for CBZ and CBNZ, but the register that comes before the immediate will be the one compared with 0.

DO: CBZ X3, 2

DO: CBNZ X1, -5

DO: BL 10

DO: Feel free to put empty lines in between instructions.

The assembler allows empty white space (only a newline) in between lines with instructions:

DO: LDR X1, [X0, 16]

SUB X1, X1, X2

DO: Specify a .data segment at the bottom of the .txt file after all the instructions.

If .data is placed at the top, it will not be read correctly and will cause errors.

DO: Use one 4-digit hexadecimal number per line when inputting data under the .data segment.

The first number will be in spot 0, the second will be in spot 1, and so on. See the example death_assembly.txt file on the last page of the manual to get an idea of what the .data segment should look like.

DO: Have fun!

DON'T: Die.

For legal reasons, this is a joke. The Death Machine has never actually killed anyone.

III. Instructions for Using the Death Assembler

1) Fire up your virtual machine! You'll be needing it in order to run death_assembler.cpp.
2) Created a shared folder with the virtual machine and name it whatever you want. We recommend "Death_Assembler"!
3) Place the death_assembler.cpp file into the shared folder.
4) Place your desired assembly code into a new text file called "death_assembly.txt." Make sure your "death_assembly.txt" assembly program is in the same shared folder as death_assembler.cpp and the given makefile, so that the assembler can read from the .txt file and translate the instructions. We have given you a demo program in the "death_assembly.txt" file that is ready to use (so that you don't have to write your own).
5) Navigate to the correct shared folder within the virtual machine. Open a terminal shell window. Type "make" into the command line and press enter to compile the program. Then type "./death_assembler" and press enter to run the program.
6) The program will print its status (reading, translating, or completed). Once the program says it has finished translating the instructions, navigate back to the shared folder, where there will be two new .txt files.
7) The .text segment of death_assembly.txt (the part containing the instructions) will be translated into a .txt file called "instruction.txt" full of machine code instructions that the Death Machine can work with. The .data segment will be translated into a .txt file called "data.txt".
8) Open the Death Machine in Logisim using the Death_Machine.circ file. Once there, right-click on the INSTRUCTION_MEMORY and select "Load image". Select instructions.txt to be loaded and click "Open".
9) Do the same thing with DATA_MEMORY and the data.txt file. You should have the translated instructions and data both loaded into the CPU.
10) Finally, run the simulation in Logisim and watch as the registers, program counter, and current instruction type update in real time using LED displays as the CPU runs your instructions!

IV. Example "death_assembly.txt" File

```
(on
LDR X1, [X0, 0]
LDR X2, [X0, 1]
LDR X3, [X0, 2]
ADD X3, X1, X2

SUB X3, X3, X2
STR X3, [X0, 3]
CBNZ X3, -2

BL 4
SUB X4, X4, X2
CBNZ X4, -1
CBZ X4, 4

ADD X4, X1, X2
ADD X4, X4, X4
RET
STR X4, [X0, 4]

.data:
0040
0010
0024
```

V. Job Descriptions

Building the Death Machine

Set up of CPU Components:	Alexander Jansiewicz
Register File Circuit:	Alexander Jansiewicz
ALU Circuit:	Alexander Jansiewicz and Clare O'Brien
Control Unit Circuit:	Alexander Jansiewicz and Clare O'Brien
Wiring:	Alexander Janziewicz and Clare O'Brien
LED Displays:	Alexander Jansiewicz and Clare O'Brien

Building the Assembler

Reading and Writing to a .txt File:	Alexander Jansiewicz
Separating .data and .text:	Alexander Jansiewicz
Translating instructions to machine code:	Clare O'Brien

Writing the Manual

Author:	Clare O'Brien
Editor:	Alexander Jansiewicz