Module 4 Assignment: Disassembler and Assembler

EN.605.204 Computer Organization

Introduction

As we delve more into the MIPS Language, the ability to create/breakdown and understand MIPS instructions is extremely important. This assignment will give you some experience doing just that. There will be two parts to this assignment. Part I you will build a basic disassembler of MIPS instructions in Python and in Part II you will play the role of assembler and interpret the instructions.

Some MIPS Reminders -

1. MIPS instructions are 32 bits
2. Each character used requires 1 byte of storage, each integer requires 4 bytes
3. Halfword = 2 bytes (16 bits), word = 4 bytes (32 bits)
4. Literals represent all numbers
5. Characters are enclosed in single quotes
6. Strings are enclosed in double quotes
7. There are 32 registers that are used, preceded by a '$'
8. Don't forget the Green Sheet!!

Part I – Disassembler in Python

A disassembler takes a machine code file in binary and converts it back into assembly code. Disassemblers are a key tool in the software reverse engineering process because it allows you to go from the raw binary all the way back to the original source code, if desired. You will be given a set of 32-bit binary strings. You'll write a basic disassembler in Python that determines the instruction type (R, I, J), splits the 32-bit string into its corresponding fields, converts them from binary into their string representation, then reorders the fields, resulting in the original assembly code instruction.

Here's an example:

000000 01010 01011 01001 00000 100000 (R-type)					
Opcode(6)	*rs(5)*	*rt(5)*	*rd(5)*	*shamt(5)*	*func(6)*
000000	01010	01011	01001	00000	1000000
	$t2	$t3	$t1		add
add $t1, $t2, $t3					

Module 4 Assignment: Disassembler and Assembler

EN.605.204 Computer Organization

Think about the steps you take for each row of the example. Those are the ones you should be implementing within your Python program. It may be helpful to build a Python dictionary that maps opcodes/function codes to instructions. The only instructions you'll have to disassemble come from the following instruction list:

add, addiu, addu, and, andi, j, jal, jr, lw, mult, nor, or, ori, sub, sw

```
10101111101111110000000000010100
10101111101001000000000000100000
10101111101001010000000000100100
00000011000011111100100000100001
10101111101110010000000000011000
10001111101001010000000000011000
00001100000000000001001011001100
00100100100001000000010000110000
10001111101111110000000000010100
00100111101111010000000000100000
00000011111000000000000000001000
00000001101011100101100000100100
10001101010010010000000000001000
00001000000000010010001101000101
00000010101010010101100000100010
00110101111100001011111011101111
00000010110011010101000000100000
```

Hints:

Start by putting the machine code instructions into a list. Build the dictionary of opcode/function codes. It might also be helpful to have a dictionary of all the registers as well.

```
this is I-instruction
funciton is:sw
binary code:10101111101111110000000000010100
MIPS assembly code:sw $sp,$ra 0x14
this is I-instruction
funciton is:sw
binary code:10101111101001000000000000100000
MIPS assembly code:sw $sp,$a0 0x20
this is I-instruction
funciton is:sw
```

```
binary code:10101111101001010000000000100100
MIPS assembly code:sw $sp,$a1 0x24
this is R instruction
Instruction: add
Function code (binary): 100001
binary code:00000011000011111100100000100001
MIPS assembly code: add $t9,$t8,$t7
this is I-instruction
funciton is:sw
binary code:10101111101110010000000000011000
MIPS assembly code:sw $sp,$t9 0x18
this is I-instruction
funciton is:lw
binary code:10001111101001010000000000011000
MIPS assembly code:lw $sp,$a1 0x18
this is J-instruction
function is:jal
binary code:00001100000000000001001011001100
MIPS assembly code:jal 0x12cc
this is I-instruction
funciton is:addiu
binary code:00100100100001000000010000110000
MIPS assembly code:addiu $a0,$a0 0x430
this is I-instruction
funciton is:lw
binary code:10001111101111110000000000010100
MIPS assembly code:lw $sp,$ra 0x14
this is I-instruction
funciton is:addiu
binary code:00100111101111010000000000100000
MIPS assembly code:addiu $sp,$sp 0x20
this is R instruction
Instruction: add
Function code (binary): 001000
binary code:00000011111000000000000000001000
MIPS assembly code: add $zero,$ra,$zero
this is R instruction
Instruction: add
Function code (binary): 100100
binary code:00000001101011100101100000100100
MIPS assembly code: add $t3,$t5,$t6
this is I-instruction
funciton is:lw
binary code:10001101010010010000000000001000
MIPS assembly code:lw $t2,$t1 0x8
this is J-instruction
function is:j
binary code:00001000000000010010001101000101
MIPS assembly code:j 0x12345
this is R instruction
```

```
Instruction: add
Function code (binary): 100010
binary code:00000010101010010101100000100010
MIPS assembly code: add $t3,$s5,$t1
this is I-instruction
funciton is:ori
binary code:00110101111100001011111011101111
MIPS assembly code:ori $t7,$s0 0xbeef
this is R instruction
Instruction: add
Function code (binary): 100000
binary code:00000010110011010101000000100000
MIPS assembly code: add $t2,$s6,$t5
```

Part II – Assembler

In this part you will play the role of the assembler. You will assemble the given MIPS instructions by hand and fill in the fields similar to the example provided in gray. Please note that you will also need to determine values for the program counter (PC) as you work through the instructions. The assembly process is reverse of what you implemented within Part I – Disassembler in Python.

1. Provide the integer values for each of the fields in the MIPS instruction
2. Convert those integer values into their machine code / binary equivalents (32-bits)
3. Convert the 32-bit representation into the hex equivalent
4. Update the program counter accordingly

						PC
Instruction ➔	addi	$v0	$zero	0		
Int Equivalent ➔	8	2	0	0		

Module 4 Assignment: Disassembler and Assembler

EN.605.204 Computer Organization

Machine Code ➔	001000 00000 00010 0000000000000000					
Hex Equivalent ➔	2002 0000					4
next:	lw	$t1	8	($t2)		
	35	9	8	10		
	100011 01010 01001 0000000000001000					
	0x8D49 0008					**0x08**
35=32+2+1=100011 in binary 9=8+1=01001 10=8+2=01010 8=0000000000001000 **1000/1101/0100/1001/0000/0000/0000/1000=8D490008**						

	sub	$t3	$s5	$t1		
	0/34	11	21	9		
	000000 10101 01001 01011 00000 100010					
	0x02A9 5822					**0x0C**
21=16+4+1=10101 **9=8+1=01001** **11=8+2+1=01011** **34=32+2=100010** **0000/0010/1010/1001/0101/1000/0010/0010**						
	addi	$v0	$t3	0x1042		

<table>
<tr><td></td><td>8</td><td>2</td><td>11</td><td>4162</td><td></td><td></td></tr>
<tr><td></td><td colspan="5">001000 01011 00010 0001000001000010</td><td></td></tr>
<tr><td></td><td colspan="5">0x2162 1042</td><td>0x10</td></tr>
<tr><td colspan="7">8=001000
2=00010
11=8+2+1=01011
0x1042=0001/0000/0100/0010
0010/0001/0110/0010/0001/0000/0100/0010=0x21621042</td></tr>
<tr><td></td><td>ori</td><td>$t7</td><td>$s0</td><td>0x128</td><td></td><td></td></tr>
<tr><td></td><td>13</td><td>15</td><td>16</td><td>296</td><td></td><td></td></tr>
<tr><td></td><td colspan="5">001101 10000 01111 0000000100101000</td><td>0x14</td></tr>
<tr><td></td><td colspan="5">0x360F 0128</td><td></td></tr>
<tr><td colspan="7">13=8+4+1=001101
16=10000
15=01111
0x128=0000/0001/0010/1000
0011/0110/0000/1111/0000/0001/0010/1000</td></tr>
<tr><td></td><td>bne</td><td>$t9</td><td>$zero</td><td>next</td><td></td><td></td></tr>
<tr><td>Assume not equal and branching to next:</td><td>5</td><td>25</td><td>0</td><td>8</td><td></td><td></td></tr>
<tr><td></td><td colspan="5">000101 11001 00000 0000000000001000</td><td></td></tr>
<tr><td></td><td colspan="5">0x1720 0008</td><td>0x24</td></tr>
<tr><td colspan="7">5=4+1=000101
25=16+8+1=11001
0=00000
8=0000/0000/0000/1000
0001/0111/0010/0000/0000/0000/0000/1000=1720 0008
PC=20+4=24</td></tr>
</table>

Module 4 Assignment: Disassembler and Assembler

EN.605.204 Computer Organization

BNE:PC=PC+4+Branchaddress=24+4+12=36=2*16+4=0x24

Deliverables

Part I – Disassembler

Please submit your Python source code in a file called <Your JHEDID>_disassembler.py as well as a PDF file called <Your JHEDID>_disassembler.pdf containing screenshots / output from your program showing all disassembled instructions.

Part II – Assembler

Please submit your assembler assignment showing all work in a PDF named <Your JHEDID>_assembler.pdf using the Assignment link on Canvas.

All documents must be submitted at the same time. You can select multiple documents by holding the <ctrl> key and clicking on the files to be submitted.