COL-216 Assignment-3

Mohit Thakur 2019CS10373
Sidharth Agarwal 2019CS50661

March 13, 2021

Aim

Develop an interpreter for a subset of MIPS assembly language instructions.

Approach

1. We first declared and array integers for memory, registers, and frequency of instructions(vector in case of frequency). We also declared the integers to store current instruction number and upto which point does the memory has instructions.
2. We also realised that we need to store instructions in bits and thus will need some kind of encoding to store instructions.
3. We decided to split instructions in categories depending on the number of registers and the number of numbers it need to store
4. Then, we read instructions one-by-one and the mapped them to category and further encoded them in integer to store the information.
5. To run the code, we reset the current instruction to 0 and started traversing instruction until it do not crosses end of instruction.

Design Decisions

1. Instead of declaring memory as 1 << 20 bytes we declared it as 1 << 18 integers for ease of storing instructions.
2. Program terminates if it traverses all the instructions or it it is made to jump to instruction that is not a part of instruction set.
3. Categories of instructions are -
   (a) type_a : add,sub,mul,slt <type> $r1,$r2,$r3
   26-30 bits for unique instruction number assigned
   21-26 bits for register1
   16-21 bits for register2
   11-16 bits for register3
   (b) type_b : bne,beq,addi <type> $r1,$r2,<value>
   26-30 bits for unique instruction number assigned
   21-26 bits for register1
   16-21 bits for register2

0-16 bits used to store integer for instruction number in case of bne and beq and number to be added in addi

(c) type_c : j <type> <label/value>
26-30 bits for unique instruction number assigned
0-26 bits used to store integer for instruction number

(d) type_d : sw,lw <type> $r1,offset($r2)
26-30 bits for unique instruction number assigned
21-26 bits for register1
16-21 bits for register2
0-16 bits for offset

4. All instructions formats are very strict, i.e. code throw error when anything unexpected occurs in the file.
5. addi instruction can only add number that is strictly less then 16 bits i.e. 1<<16.
6. Implementation of sw and lw is bit different from that in other MIPS architectures. It reads/strores next bytes from ((4*register)+offset).

Testing

We tested the compiled code on manually written codes of subset language. Here are some some snippets and results(only concerned registers and statistics).

Input :

```
addi $t0,$ze,45
addi $t1,$ze,36
add $t2,$t1,$t0
mul $t3,$t1,$t0
sub $t4,$t1,$t0
sw $t1,2($t0)
lw $t5,2($t0)
slt $t6,$t1,$t0
beq $t6,$ze,15
bne $t6,$ze,11
mul $t3,$t2,$t1
j 15
sub $t3,$t3,$t3
```

Output:

```
$ze :0 $t0 :0 $t1 :0 $t2 :0 $t3 :0 $t4 :0 $t5 :0 $t6 :0

$ze :0 $t0 :2d $t1 :0 $t2 :0 $t3 :0 $t4 :0 $t5 :0 $t6 :0

$ze :0 $t0 :2d $t1 :24 $t2 :0 $t3 :0 $t4 :0 $t5 :0 $t6 :0

$ze :0 $t0 :2d $t1 :24 $t2 :51 $t3 :0 $t4 :0 $t5 :0 $t6 :0

$ze :0 $t0 :2d $t1 :24 $t2 :51 $t3 :654 $t4 :0 $t5 :0 $t6 :0

$ze :0 $t0 :2d $t1 :24 $t2 :51 $t3 :654 $t4 :fffffff7 $t5 :0 $t6 :0
```

```
$ze :0 $t0 :2d $t1 :24 $t2 :51 $t3 :654 $t4 :fffffff7 $t5 :0 $t6 :0

$ze :0 $t0 :2d $t1 :24 $t2 :51 $t3 :654 $t4 :fffffff7 $t5 :24 $t6 :0

$ze :0 $t0 :2d $t1 :24 $t2 :51 $t3 :654 $t4 :fffffff7 $t5 :24 $t6 :1

$ze :0 $t0 :2d $t1 :24 $t2 :51 $t3 :654 $t4 :fffffff7 $t5 :24 $t6 :1

$ze :0 $t0 :2d $t1 :24 $t2 :51 $t3 :654 $t4 :fffffff7 $t5 :24 $t6 :1

$ze :0 $t0 :2d $t1 :24 $t2 :51 $t3 :654 $t4 :fffffff7 $t5 :24 $t6 :1

Clock cycles Elapsed: 11
instruction at line_number 0 ran for 1 time(s)
instruction at line_number 1 ran for 1 time(s)
instruction at line_number 2 ran for 1 time(s)
instruction at line_number 3 ran for 1 time(s)
instruction at line_number 4 ran for 1 time(s)
instruction at line_number 5 ran for 1 time(s)
instruction at line_number 6 ran for 1 time(s)
instruction at line_number 7 ran for 1 time(s)
instruction at line_number 8 ran for 1 time(s)
instruction at line_number 9 ran for 1 time(s)
instruction at line_number 10 ran for 0 time(s)
instruction at line_number 11 ran for 1 time(s)
instruction at line_number 12 ran for 0 time(s)
```

Observe that first 2 instructions are to load t0 and t1 with 45 and 36 respectively. Instructions at line_number 2-4 are basic operations. Instruction at line_number 5 and 6 are for observing the functioning of lw and sw. slt at line_number 7 sets t6 to 1, beq branches if t6 and ze are equal, but they are not and thus code simply runs. Now as t6 and ze are not equal at line 9, bne branches it to line 11, and thus line 10 is skipped(observe in stats). j is jump statement and it points to a line number that is not in instructions and thus terminates the program.