Assignment - 1

Khadarabad Tahir Mohammed - IMT2022100, Khadarabad.Mohammed@iiitb.ac.in

Sasi Snigdha Yadavalli - IMT2022571, YS.Snigdha@iiitb.ac.in

October 3, 2023

1 Convolution

1.1 Explanation of MIPS Code

The MIPS code takes the length of the input sequence ($t1), length of impulse response sequence ($t3), the addresses where the sequences(x - ($t2), h - ($t5), y - ($t6)) must be stored along with the sequences as input. The code has 3 parts - nloop, kloop and yupdate.

Initialization - n = 0; k = 0; y[0] = 0
Memory Access -> for $x[i] = (i * 4) + baseaddress$

- nloop -
  - Checks the condition if $n < l$ where $l = (length1) + (length2) - 1$
  - Checks the condition if $k < length1$ and branches to kloop if it is
  - Else, increments n, resets k and stores 0 in y[n], loops again
- kloop -
  - Calculates $n - k$ and stores it on the stack
  - Checks if $(n - k) > -1$ and $(n - k) < length2$ and branches to yupdate if both are True
  - Else, increments k and deallocates stack
  - Calls nloop procedure
- yupdate -
  - Retrieves $n - k$ stored on the stack and loads $h[n - k]$
  - Loads $x[k]$
  - Calculates $y[n] = y[n] + x[k] * h[n - k]$
  - Increments k and calls nloop procedure

1.2 Code and Output Screenshots

The screenshots of the MIPS simulator, Data segment, Text segments, Registers, Terminal are below.

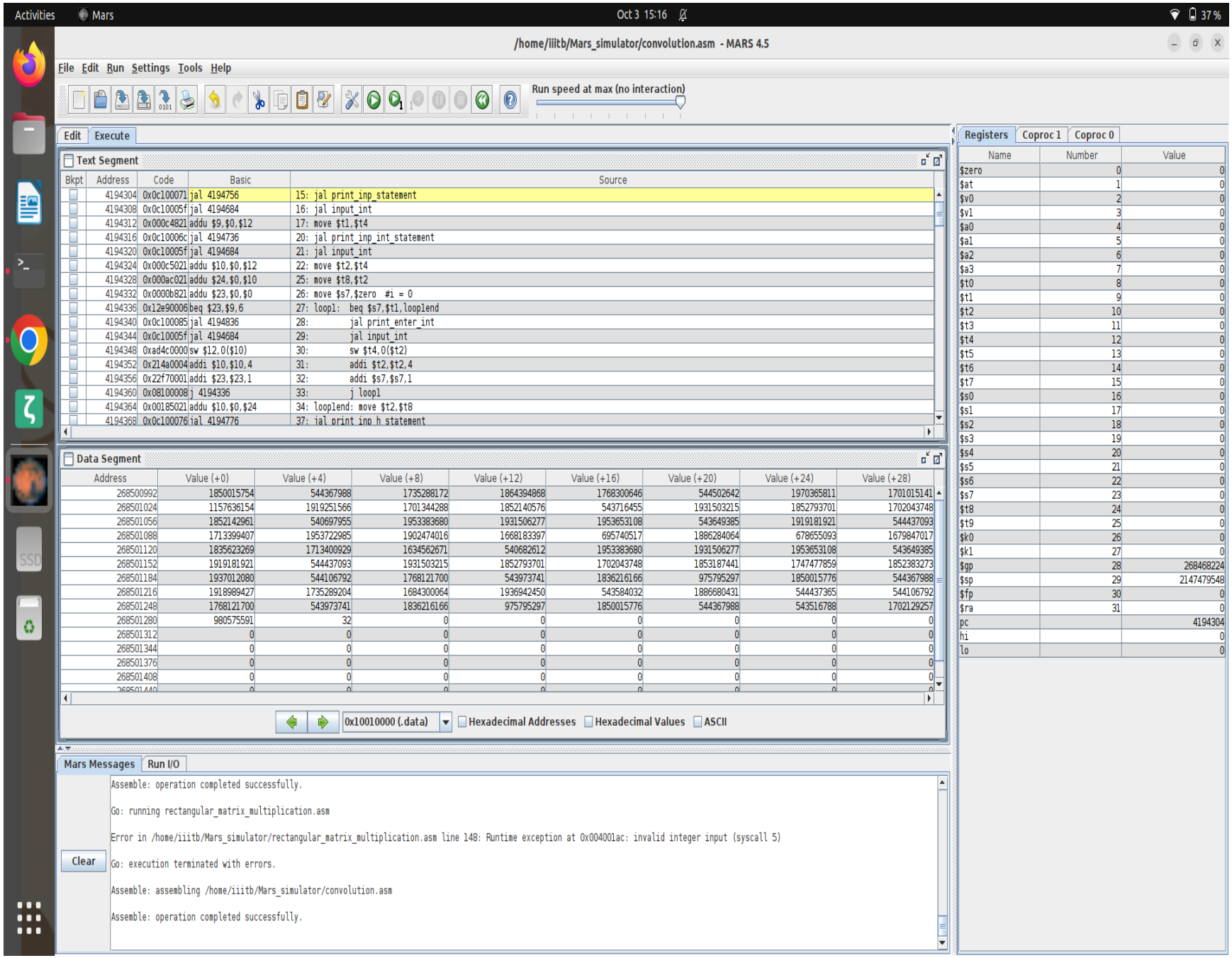


Figure 1.1. Initial State after assembly

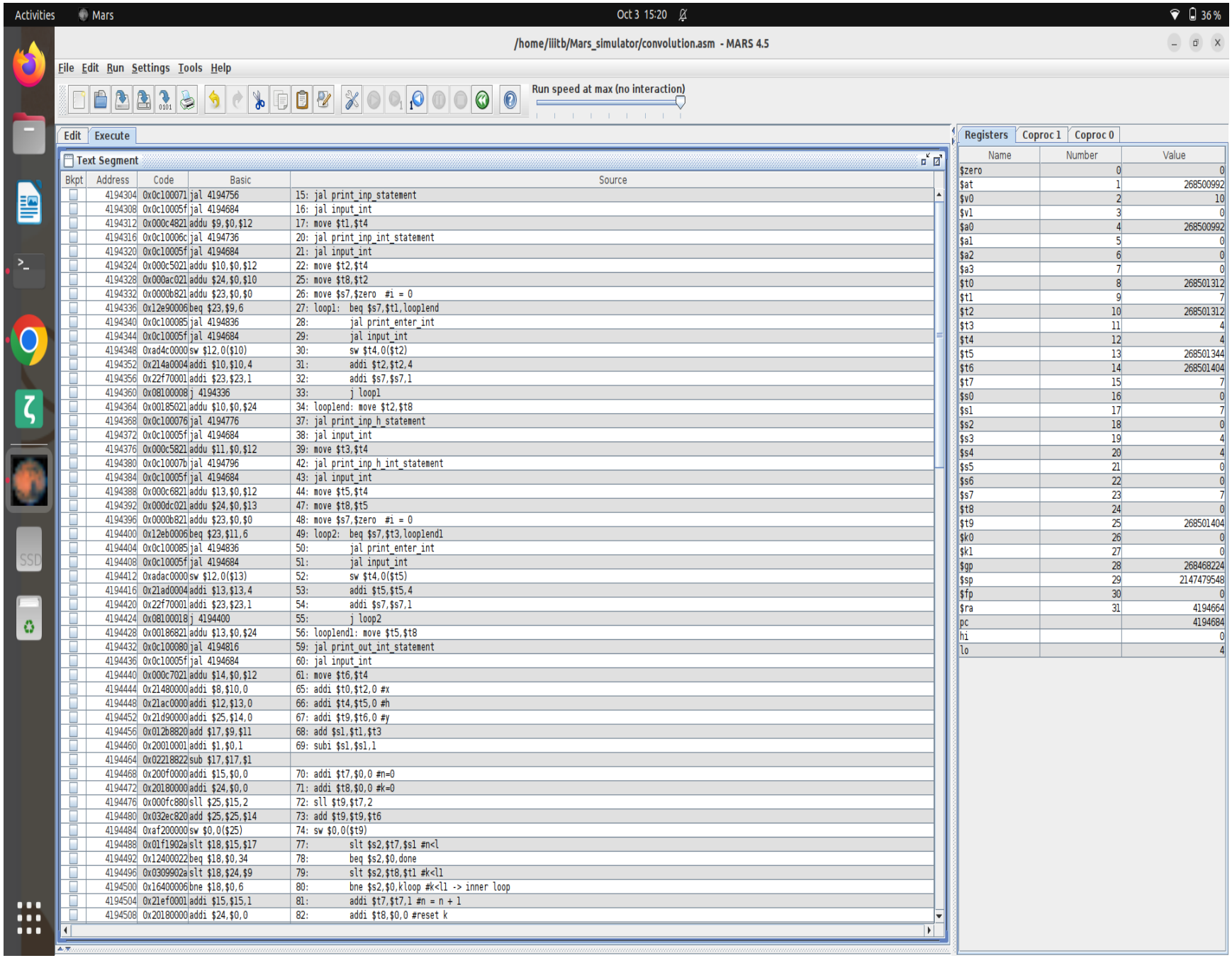


Figure 1.2. Text segment

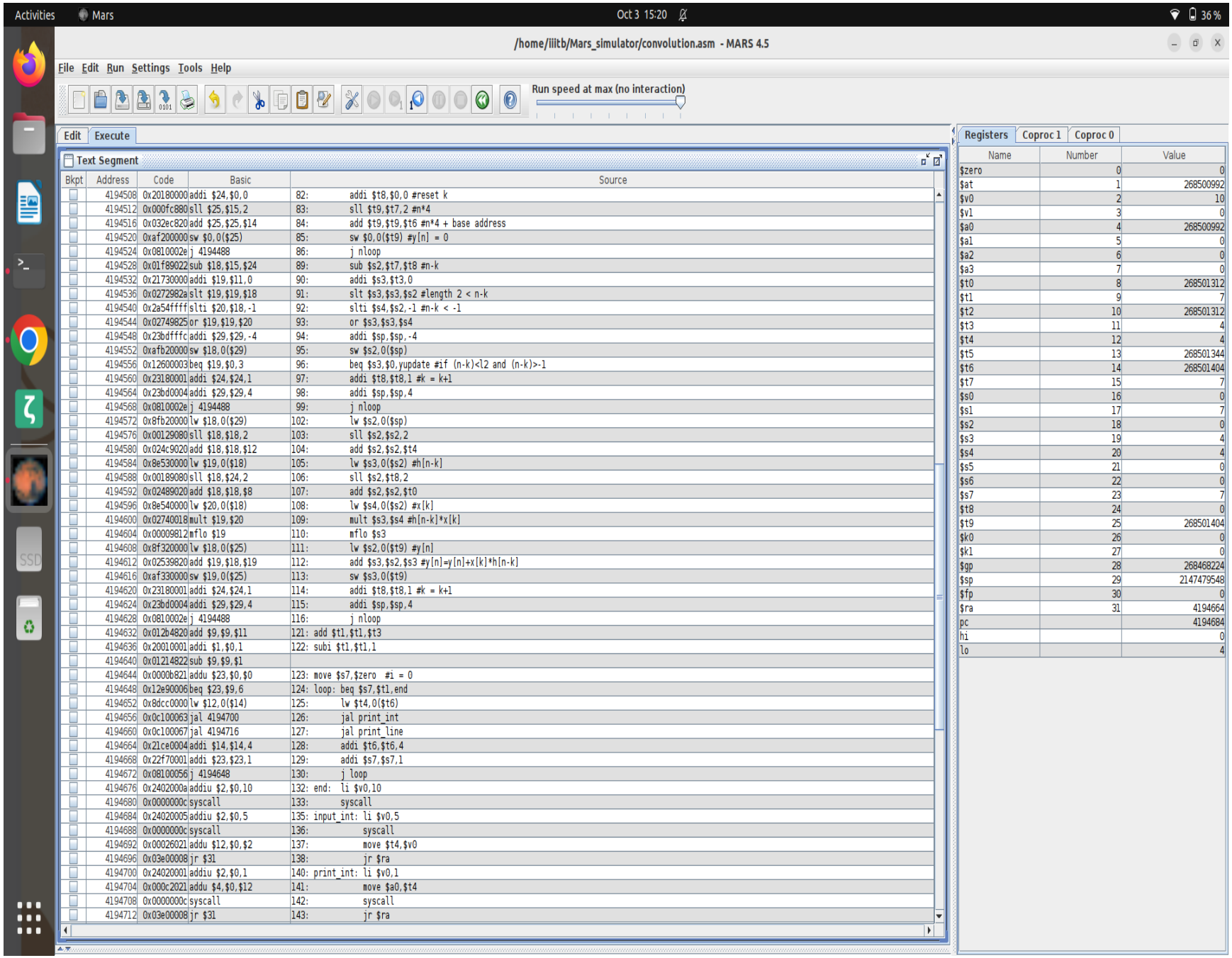


Figure 1.3. Text segment

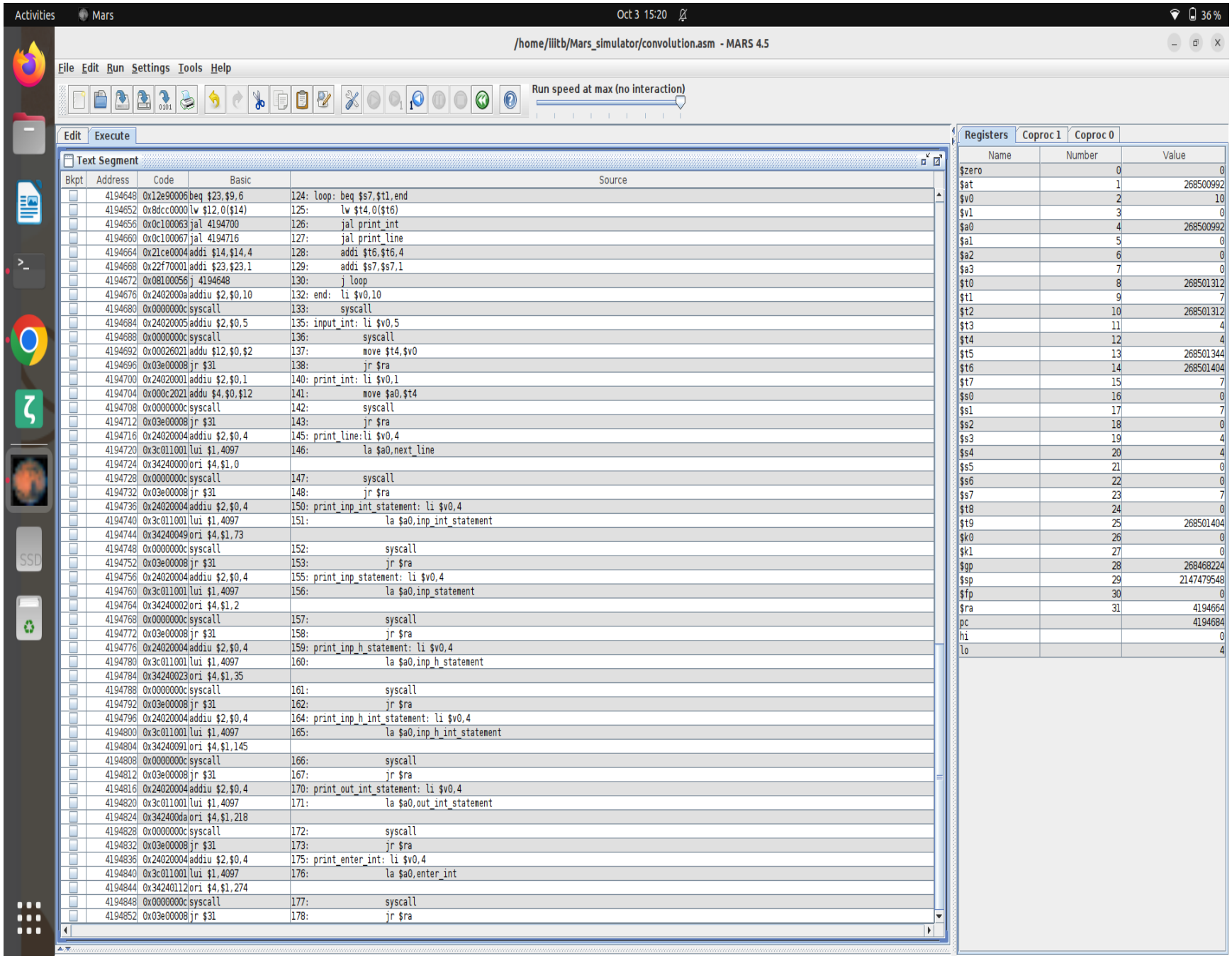


Figure 1.4. Text segment

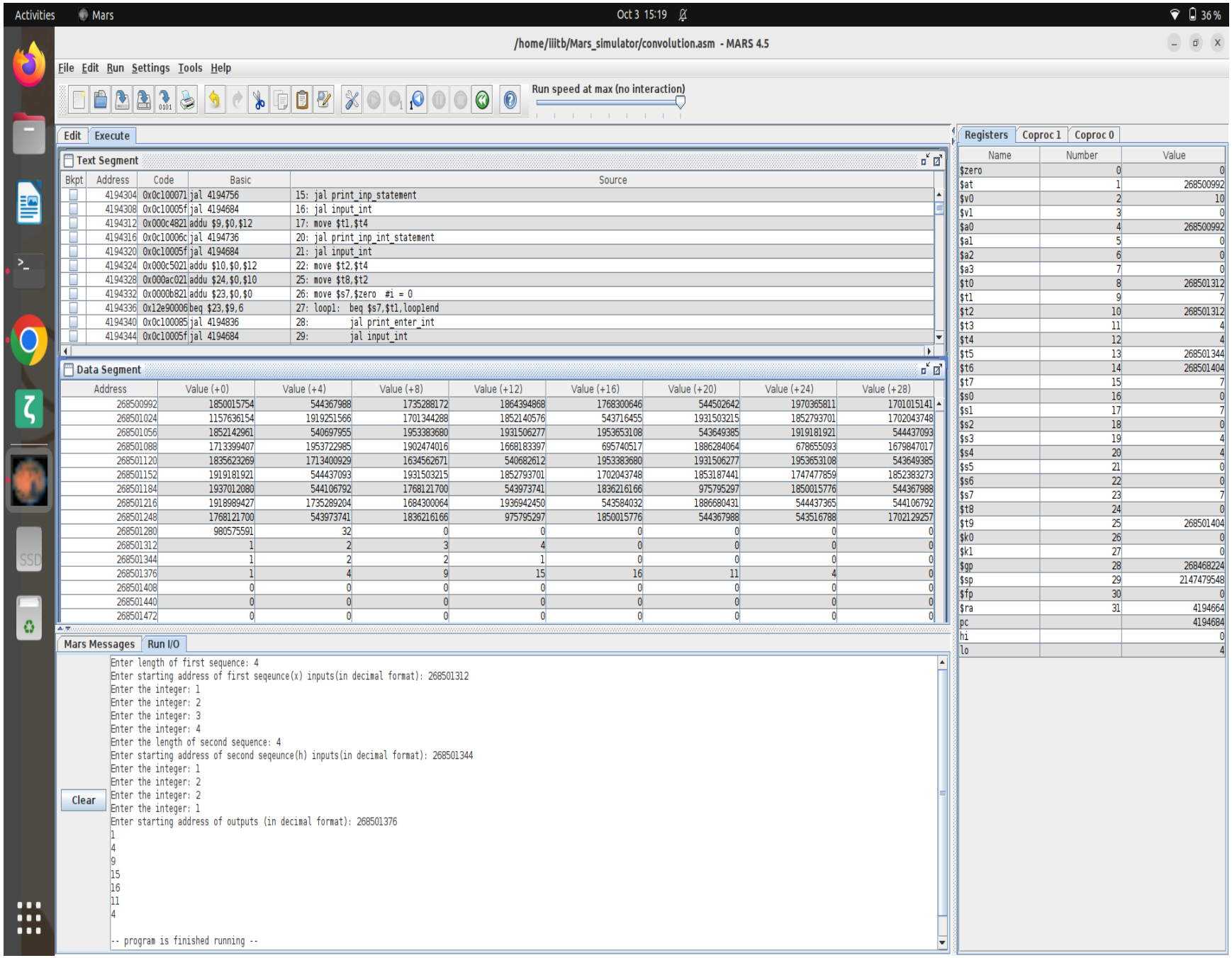


Figure 1.5. Output

1.3 Assembler(Python Code) Explanation

The r-format, j-format, i-format instructions, register numbers, function values for r-format instructions are declared as dictionaries. There are a total of 6 functions - binaryN, bin4hex, jumpaddress, shamt, func, MachineCodeMaker.

- binaryN - Takes an integer and the number of digits required in binary as arguments. Converts the integer into binary with N digits with signed padding.
- bin4hex - Takes a 32bit binary string as an argument and converts it into a hexadecimal 8bit string. (Dictionary to store hexadecimal values of 4 bit binary numbers)
- jumpaddress - Creates a dictionary with jump target addresses for procedures. (hard-coded for convolution based on MARS simulator)
- shamt - Used to identify shift functions and change the shamt field in r-type instructions accordingly.
- func - Returns the function value of r-type instruction. (Dictionary for all the function values of instructions)
- MachineCodeMaker - Converts each line of MIPS instruction into a 32 bit binary string; Considers sll, lw, sw, mult, mflo, beq, bne and other r,j,i format instructions differently and converts accordingly using above functions.
- Main file I/O - Reads the convolutioncore.asm file, processes input text and passes it to the MachineCodeMaker and binaryN to obtain and write into a file.

1.4 Code and Output screenshots

The screenshots of the Python assembler code and output text file that it generates which contains the machine code.

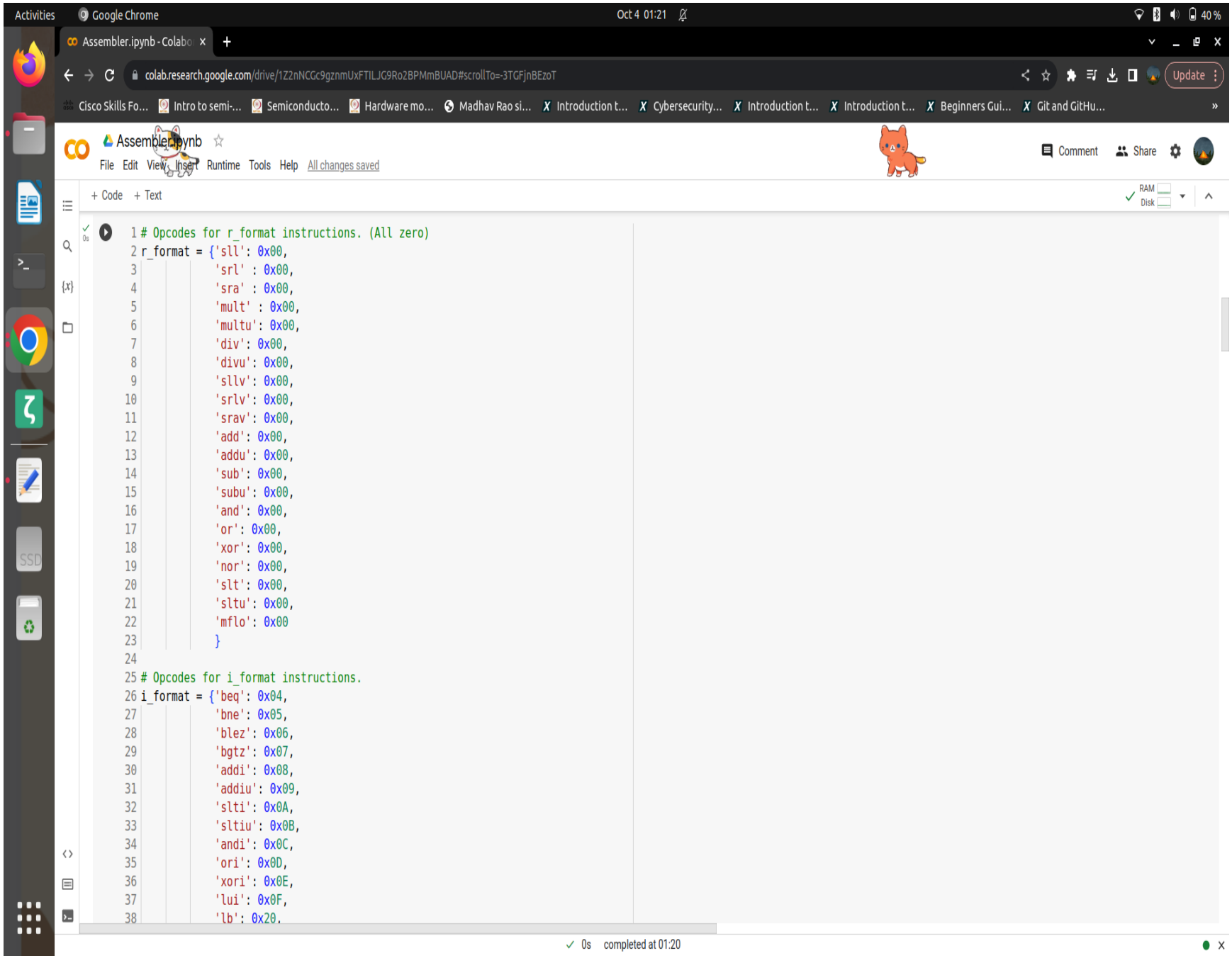
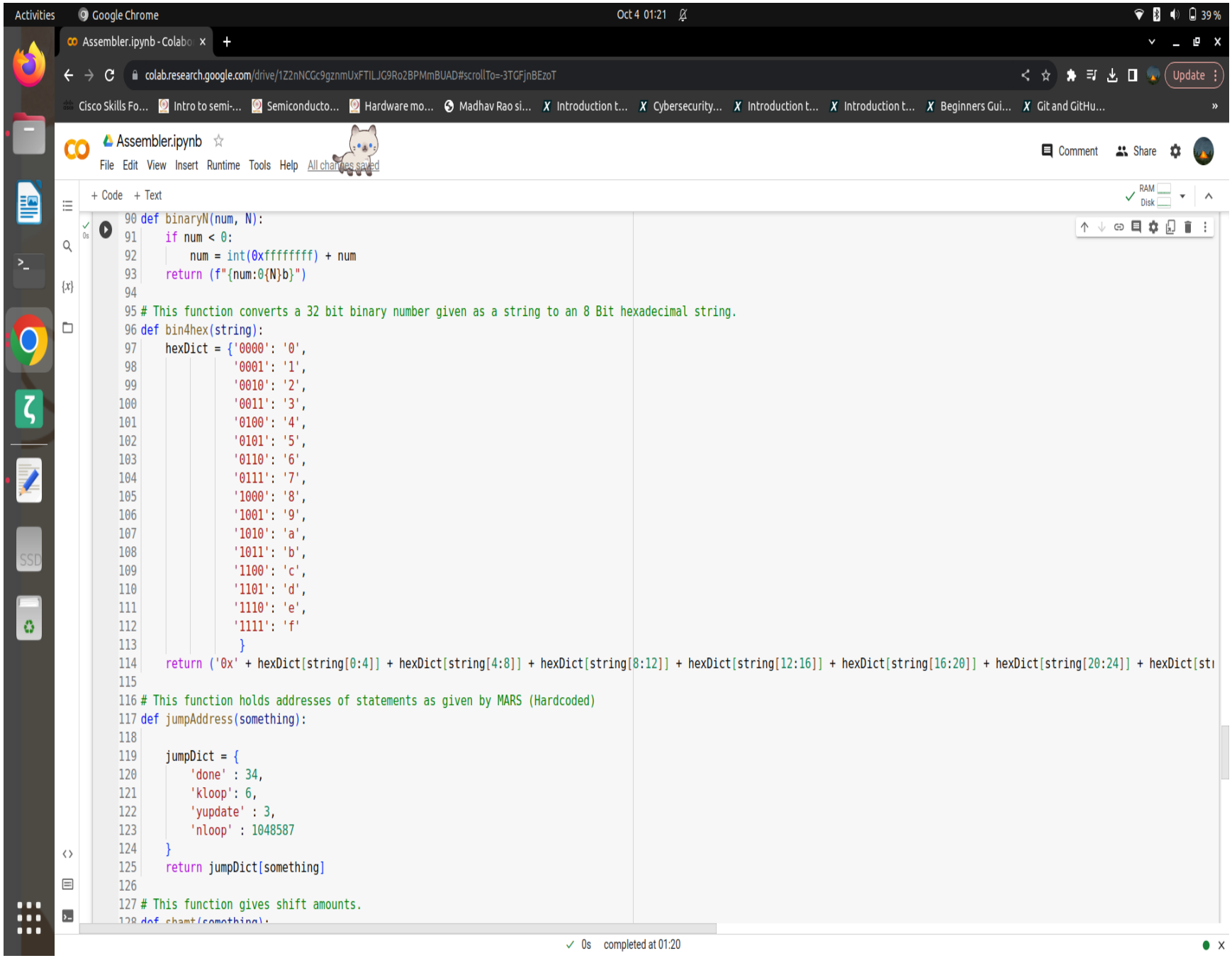


Figure 1.6. Dictionaries for each type of instruction, register numbers




```
def binaryN(num, N):
    if num < 0:
        num = int(0xffffffff) + num
    return (f"{num:0{N}b}")

# This function converts a 32 bit binary number given as a string to an 8 Bit hexadecimal string.
def bin4hex(string):
    hexDict = {'0000': '0',
               '0001': '1',
               '0010': '2',
               '0011': '3',
               '0100': '4',
               '0101': '5',
               '0110': '6',
               '0111': '7',
               '1000': '8',
               '1001': '9',
               '1010': 'a',
               '1011': 'b',
               '1100': 'c',
               '1101': 'd',
               '1110': 'e',
               '1111': 'f'
               }
    return ('0x' + hexDict[string[0:4]] + hexDict[string[4:8]] + hexDict[string[8:12]] + hexDict[string[12:16]] + hexDict[string[16:20]] + hexDict[string[20:24]] + hexDict[str

# This function holds addresses of statements as given by MARS (Hardcoded)
def jumpAddress(something):

    jumpDict = {
        'done' : 34,
        'kloop': 6,
        'yupdate' : 3,
        'nloop' : 1048587
    }
    return jumpDict[something]

# This function gives shift amounts.
def shamt(something):
```

Figure 1.7. Branch target addresses (hardcoded)

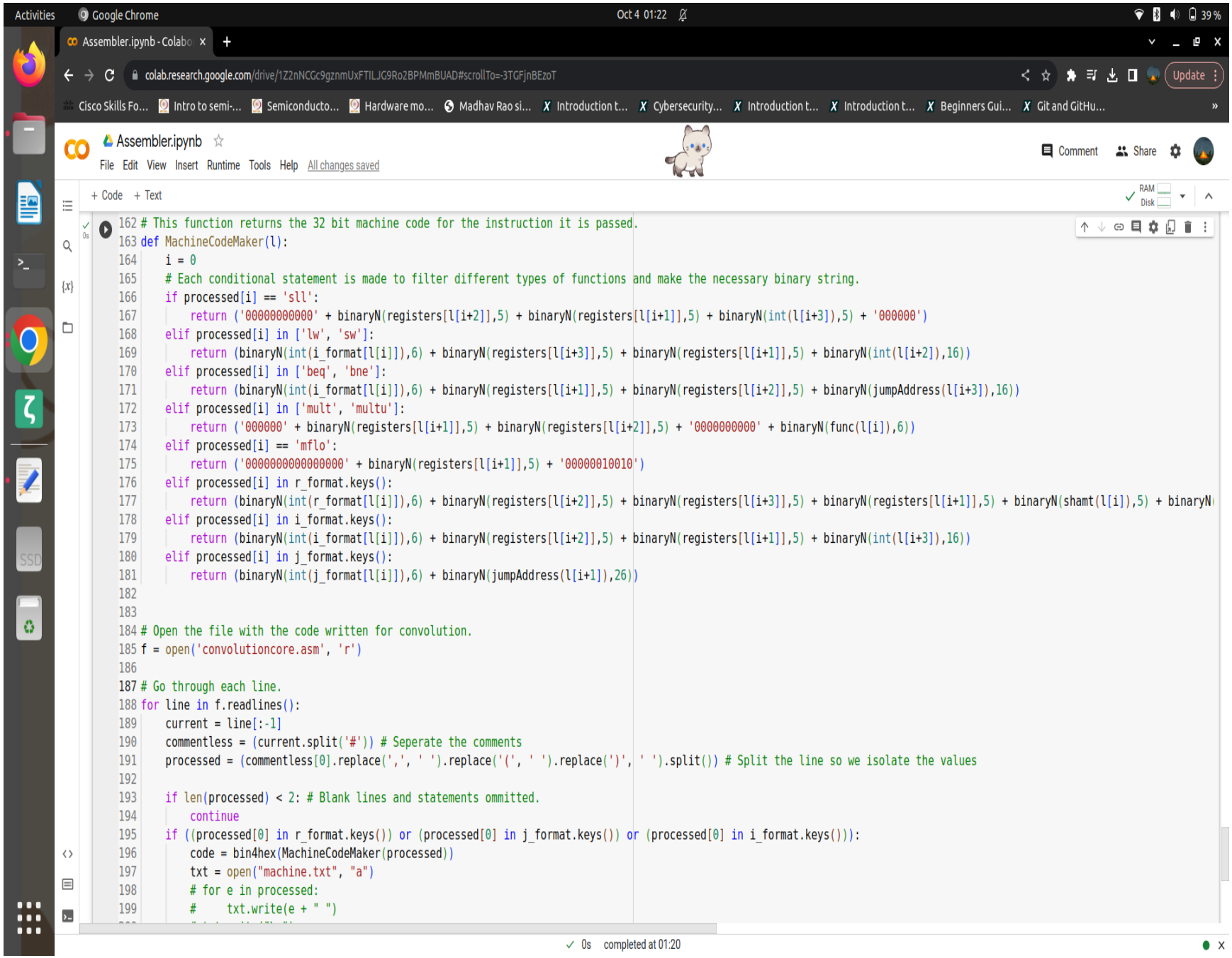


Figure 1.8. File I/O

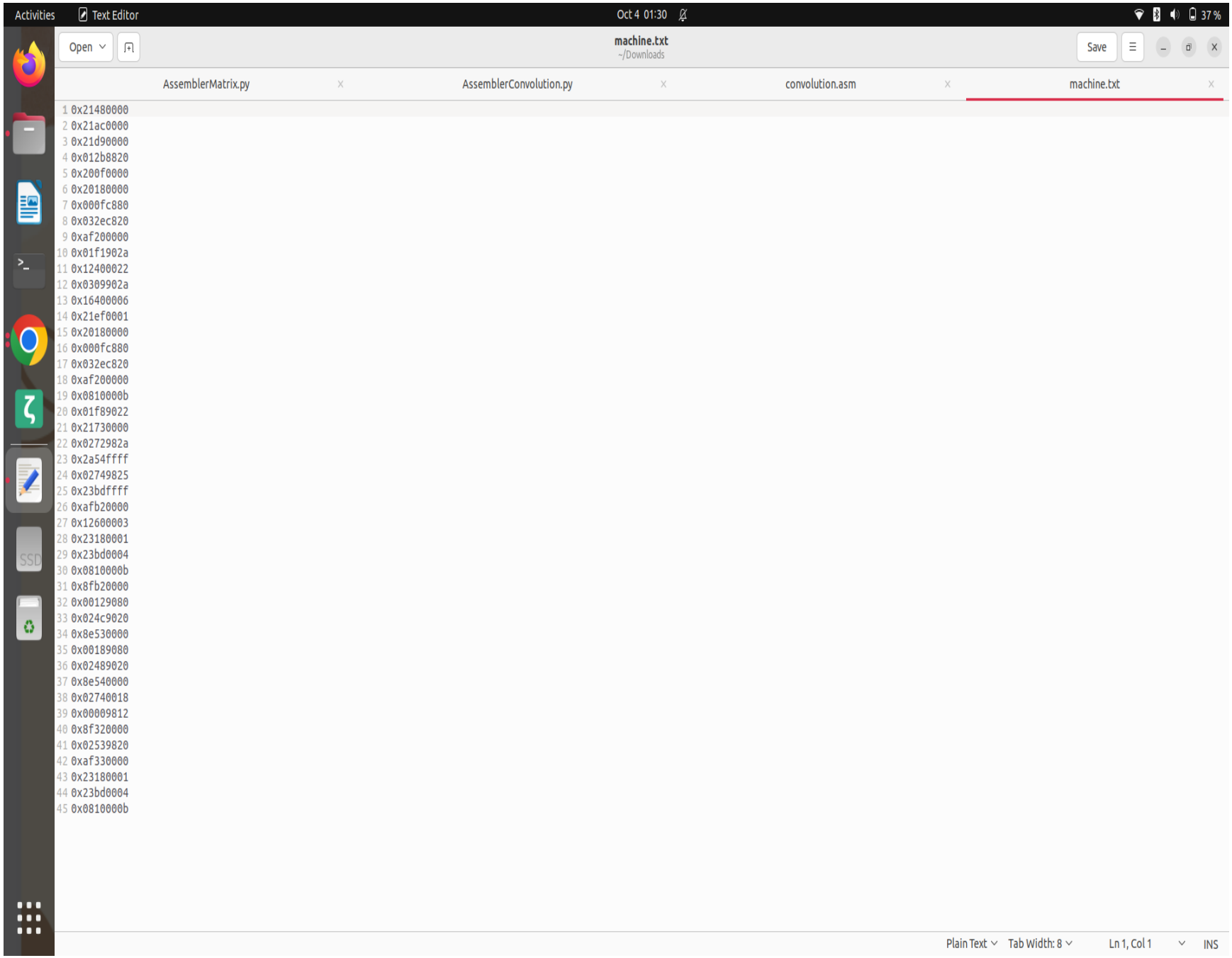


Figure 1.9. Output text file

Figure 1.10. MARS machine code output text file

2 Matrix Multiplication

2.1 Explanation of MIPS Code

The MIPS code takes the number of rows and columns of each matrix (A- m ($t1)*n($s1), B- p ($s2) * q ($s2)), input addresses (A- ($t2), B- ($t5)), matrices' elements, output address ($t6) as input. It checks whether the number of columns in the first matrix is equal to the number of rows in the second matrix and terminates if not equal. The code has 3 parts - iloop, jloop and kloop.

Initialization - i = 0; j = 0; k = 0; c[0][0] = 0

Memory Access -> for $x[i][j] = (i * n + j) * 4 + baseaddress$, where n = number of columns

- iloop -
  - Checks the condition if $i < m$
  - Checks the condition if $j < q$ and branches to jloop if it is
  - Else, increments i, resets j and loops again
- jloop -
  - Checks if $k < n$ and branches to yupdate if it is True
  - Else, increments j, resets k and stores 0 in c[i][j]
  - Calls iloop procedure
- kloop -
  - Loads $a[i][k]$ and $b[k][j]$
  - Retrieves $c[i][j]$
  - Calculates $c[i][j] = c[i][j] + a[i][k] * b[k][j]$
  - Increments k and calls jloop procedure

2.2 Code and Output Screenshots

The screenshots of the MIPS simulator, Data segment, Text segments, Registers, Terminal are below.

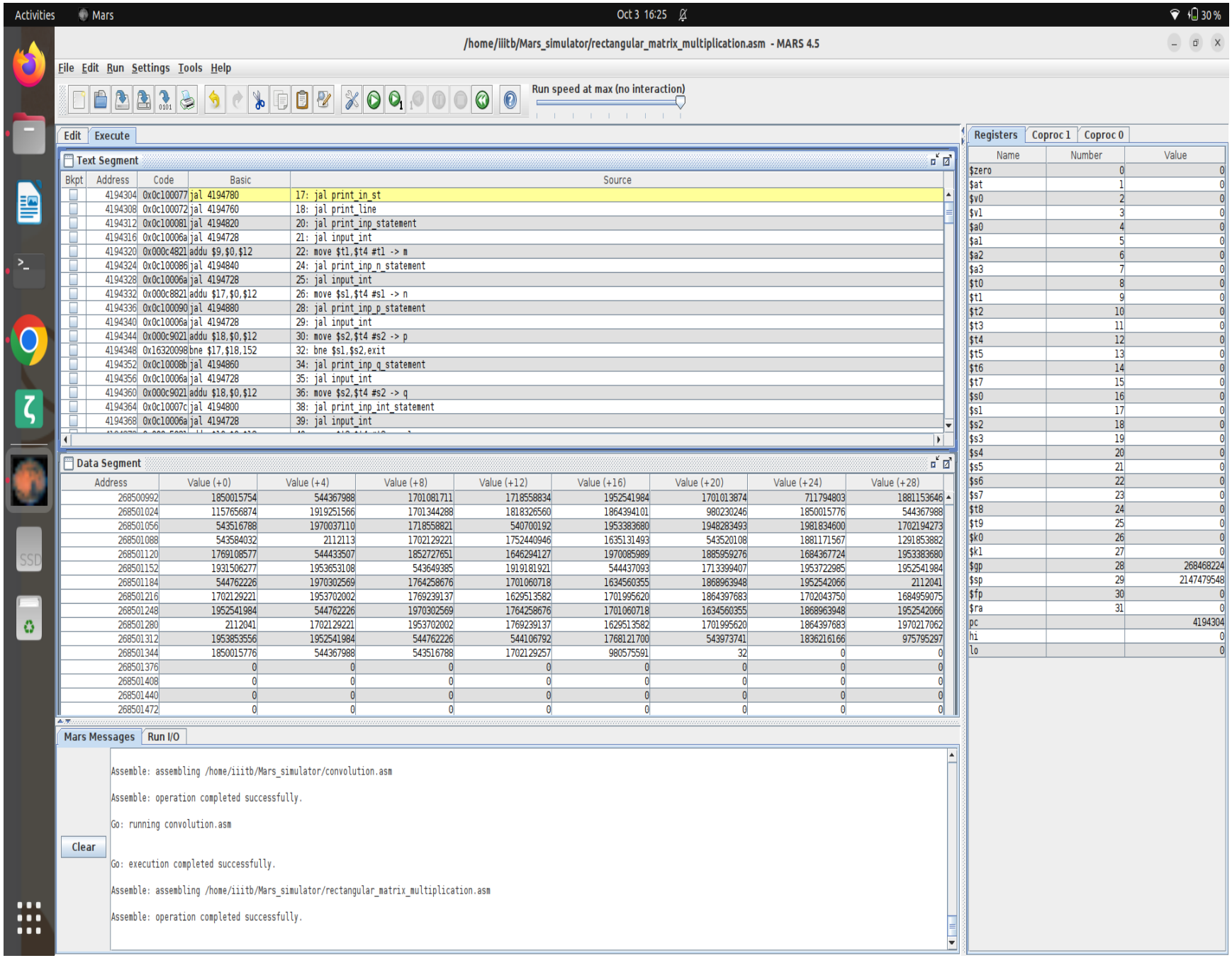


Figure 2.1. Initial State after assembly

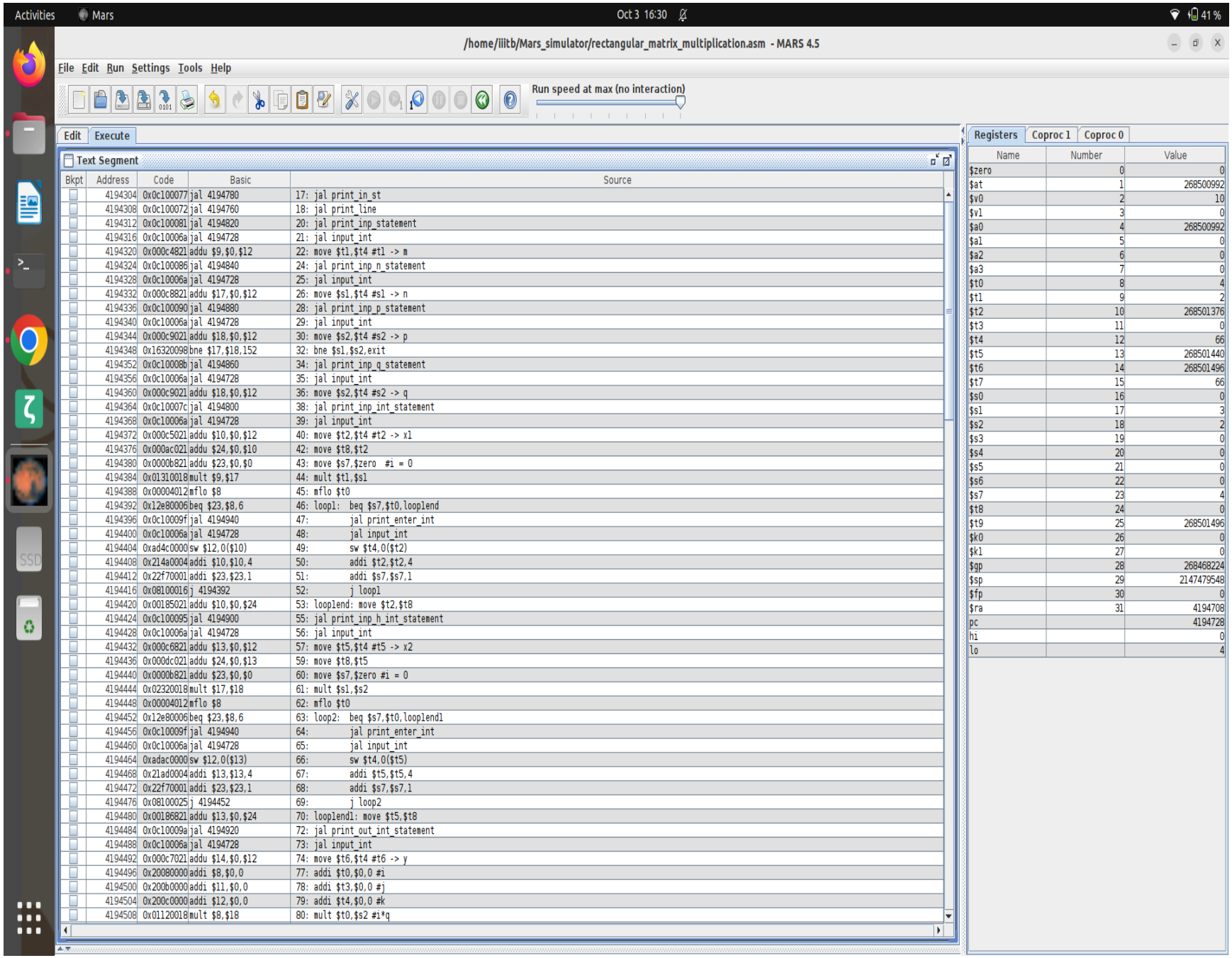


Figure 2.2. Text segment

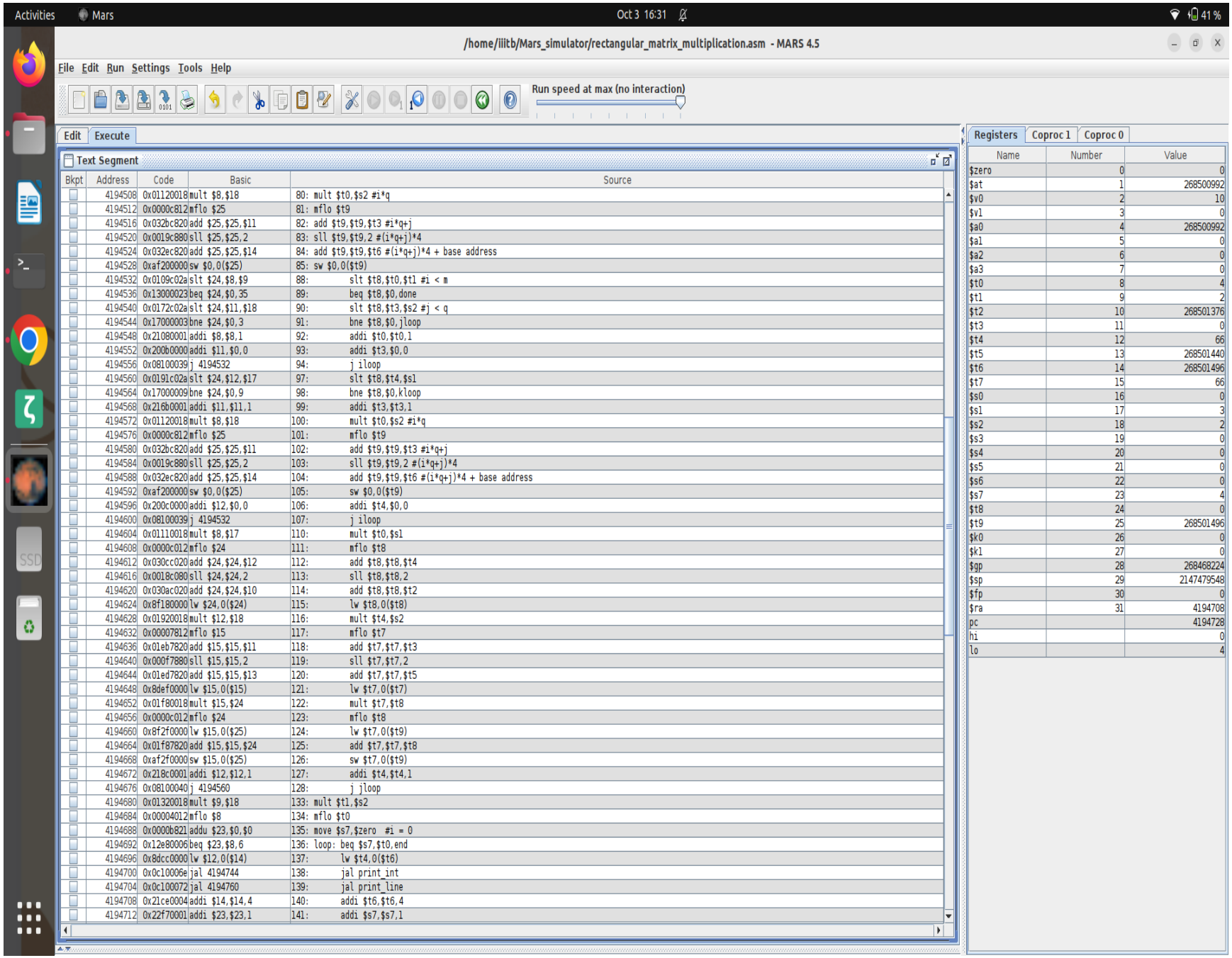


Figure 2.3. Text segment

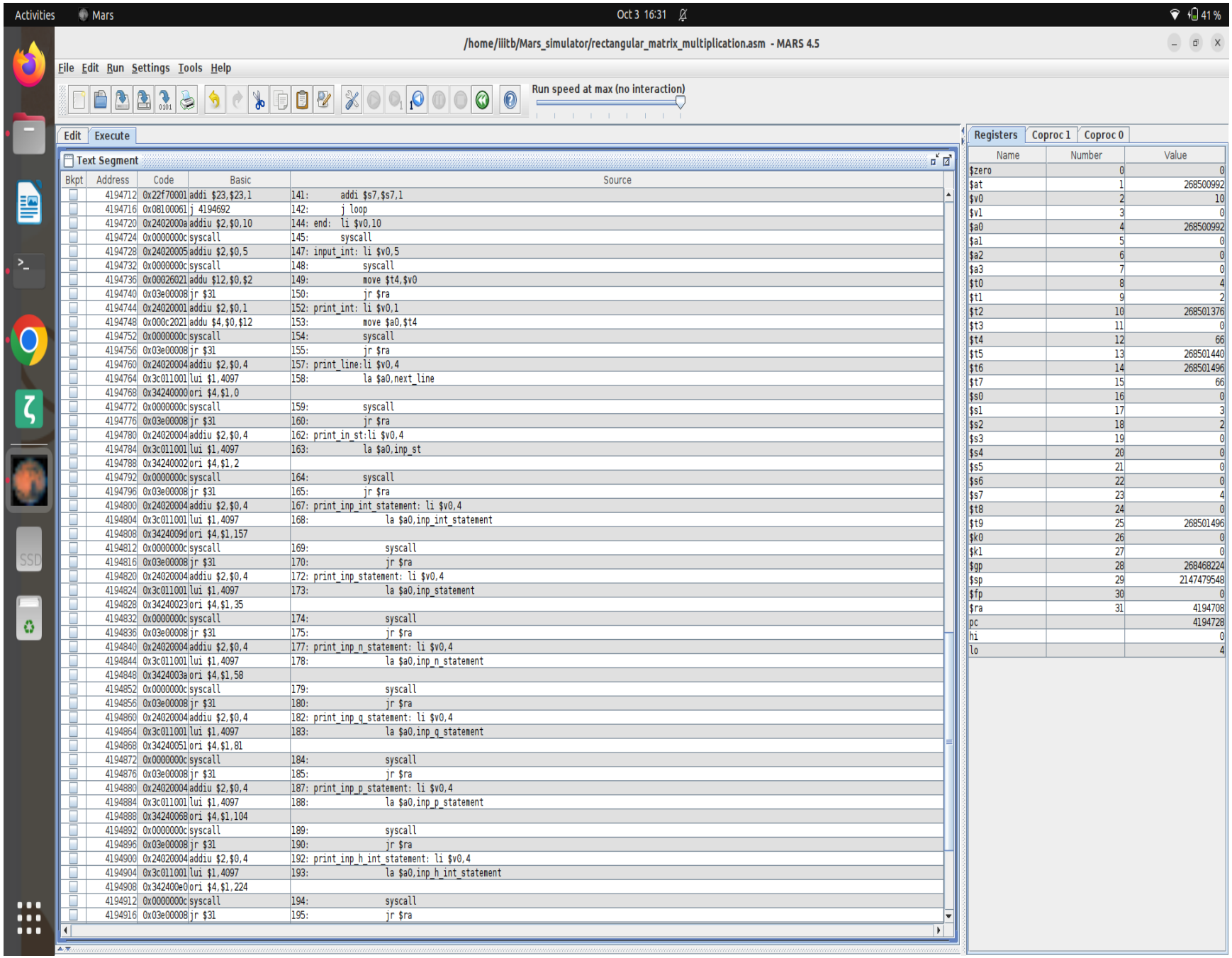


Figure 2.4. Text segment

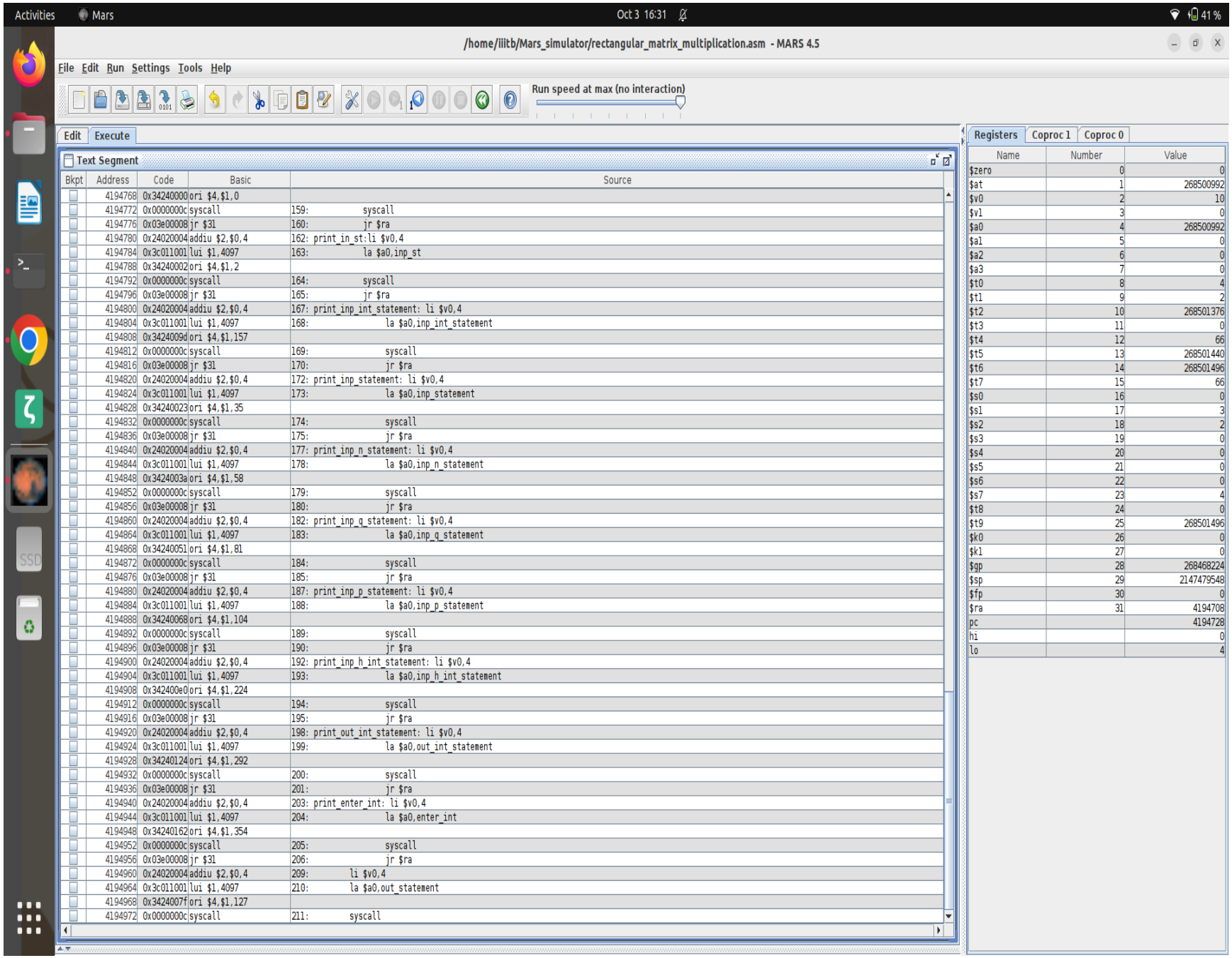
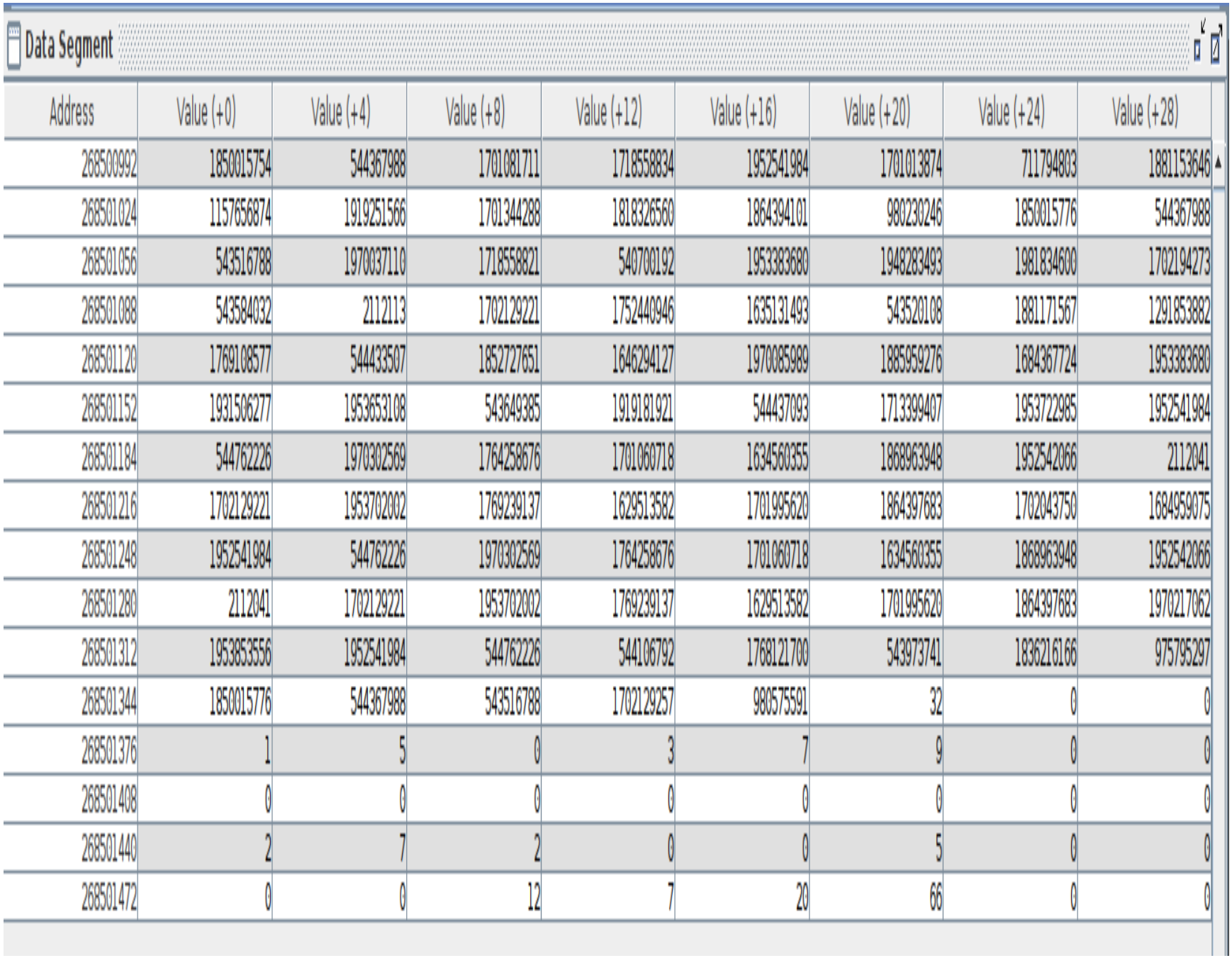


Figure 2.5. Text segment



Address	Value (+0)	Value (+4)	Value (+8)	Value (+12)	Value (+16)	Value (+20)	Value (+24)	Value (+28)
268500992	1850015754	544367988	1701081711	1718558834	1952541984	1701013874	711794803	1881153646
268501024	1157656874	1919251566	1701344288	1818326560	1864394101	980230246	1850015776	544367988
268501056	543516788	1970037110	1718558821	540700192	1953383680	1948283493	1981834600	1702194273
268501088	543584032	2112113	1702129221	1752440946	1635131493	543520108	1881171567	1291853882
268501120	1769108577	544433507	1852727651	1646294127	1970085989	1885959276	1684367724	1953383680
268501152	1931506277	1953653108	543649385	1919181921	544437093	1713399407	1953722985	1952541984
268501184	544762226	1970302569	1764258676	1701060718	1634560355	1868963948	1952542066	2112041
268501216	1702129221	1953702002	1769239137	1629513582	1701995620	1864397683	1702043750	1684959075
268501248	1952541984	544762226	1970302569	1764258676	1701060718	1634560355	1868963948	1952542066
268501280	2112041	1702129221	1953702002	1769239137	1629513582	1701995620	1864397683	1970217062
268501312	1953853556	1952541984	544762226	544106792	1768121700	543973741	1836216166	975795297
268501344	1850015776	544367988	543516788	1702129257	980575591	32	0	0
268501376	1	5	0	3	7	9	0	0
268501408	0	0	0	0	0	0	0	0
268501440	2	7	2	0	0	5	0	0
268501472	0	0	12	7	20	66	0	0

Figure 2.6. Data segment

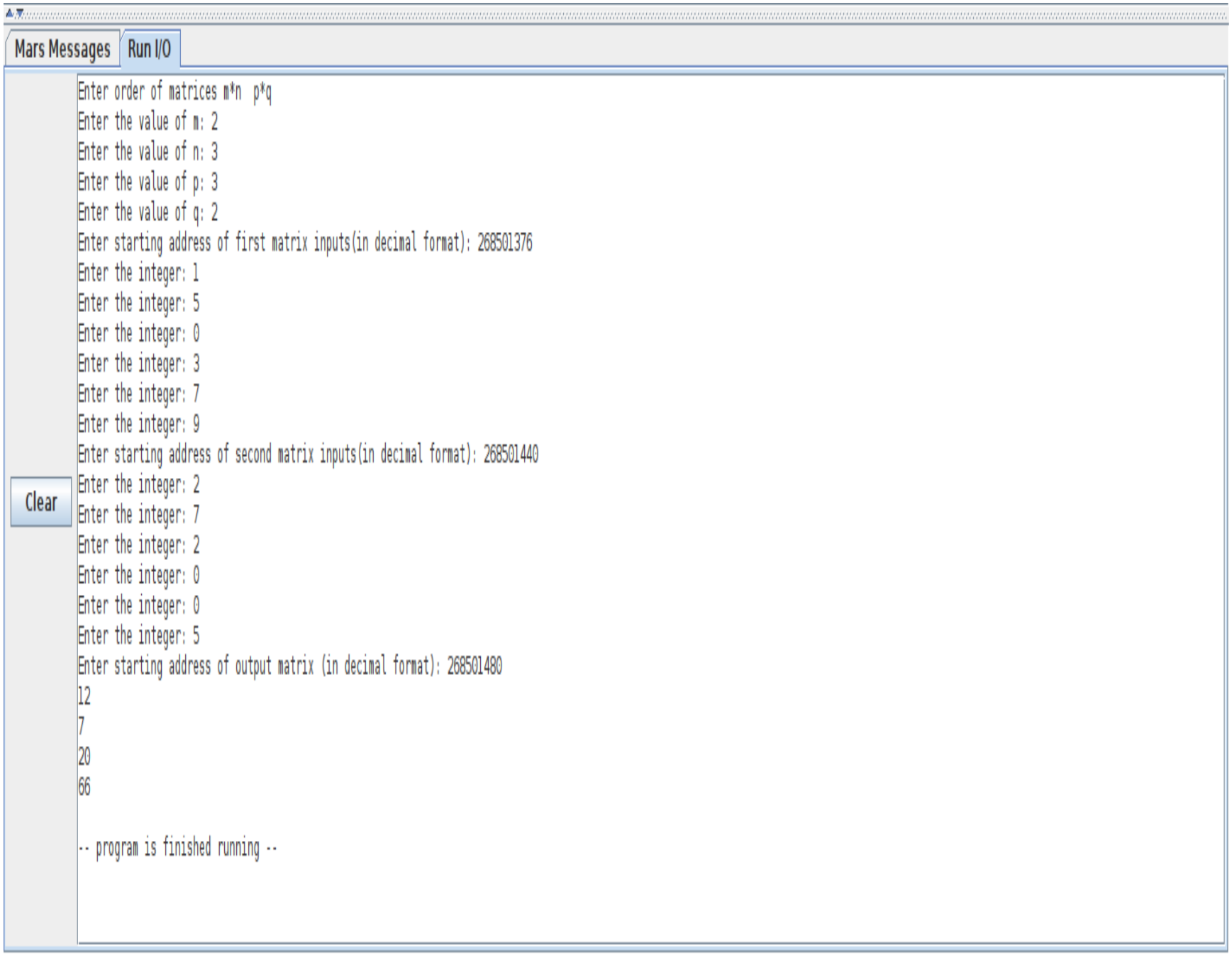


Figure 2.7. Terminal

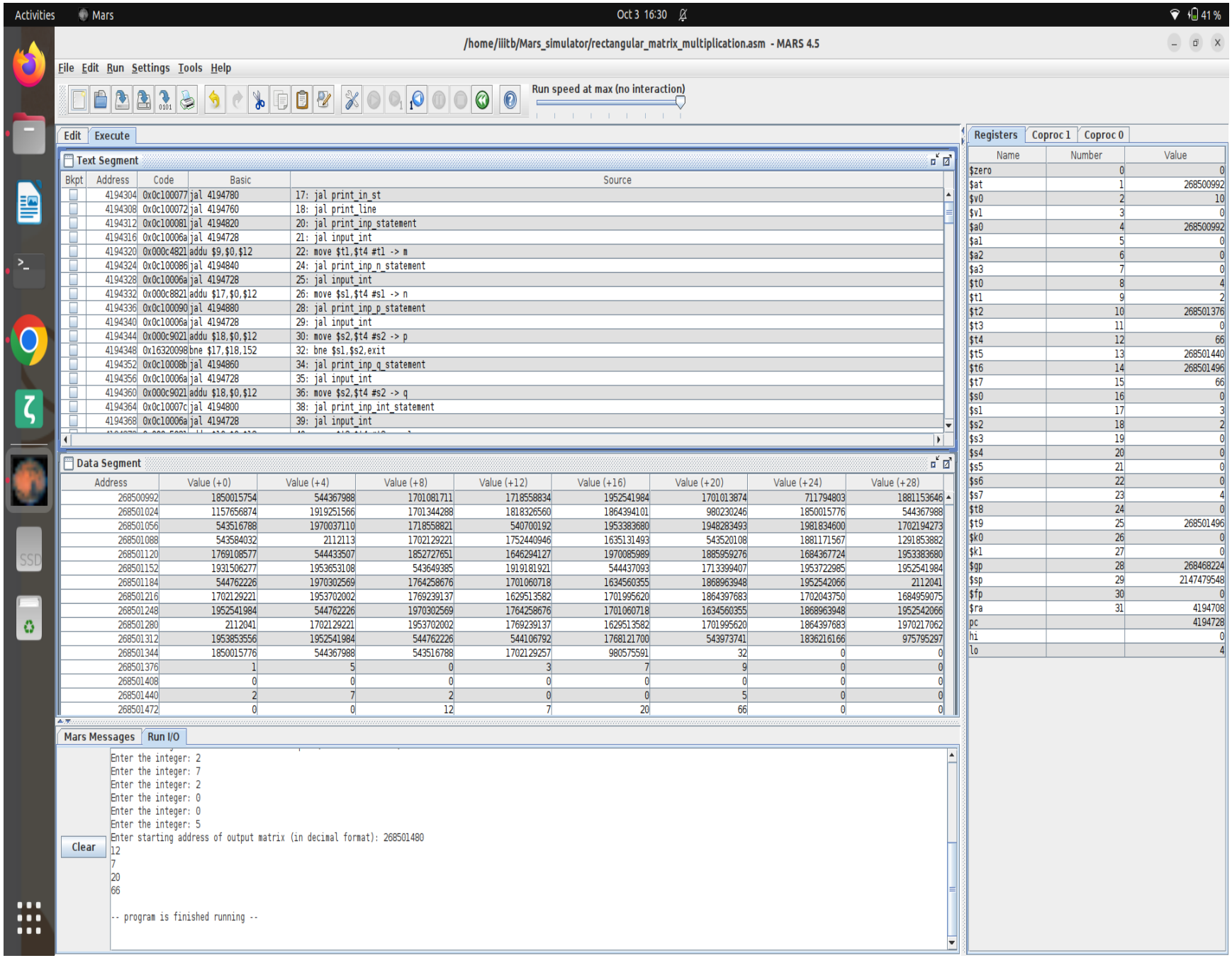


Figure 2.8. Output

2.3 Assembler(Python Code) Explanation

The r-format, j-format, i-format instructions, register numbers, function values for r-format instructions are declared as dictionaries. There are a total of 6 functions - binaryN, bin4hex, jumpaddress, shamt, func, MachineCodeMaker.

- binaryN - Takes an integer and the number of digits required in binary as arguments. Converts the integer into binary with N digits with signed padding.
- bin4hex - Takes a 32bit binary string as an argument and converts it into a hexadecimal 8bit string. (Dictionary to store hexadecimal values of 4 bit binary numbers)
- jumpaddress - Creates a dictionary with jump target addresses for procedures. (hard-coded for matrix based on MARS simulator)
- shamt - Used to identify shift functions and change the shamt field in r-type instructions accordingly.
- func - Returns the function value of r-type instruction. (Dictionary for all the function values of instructions)
- MachineCodeMaker - Converts each line of MIPS instruction into a 32 bit binary string; Considers sll, lw, sw, mult, mflo, beq, bne and other r,j,i format instructions differently and converts accordingly using above functions. (hard coded for matrix due to j, jloop)
- Main file I/O - Reads the matrixcore.asm file, processes input text and passes it to the MachineCodeMaker and binaryN to obtain and write into a file.

2.4 Code and Output screenshots

The screenshots of the Python assembler code and output text file that it generates which contains the machine code.

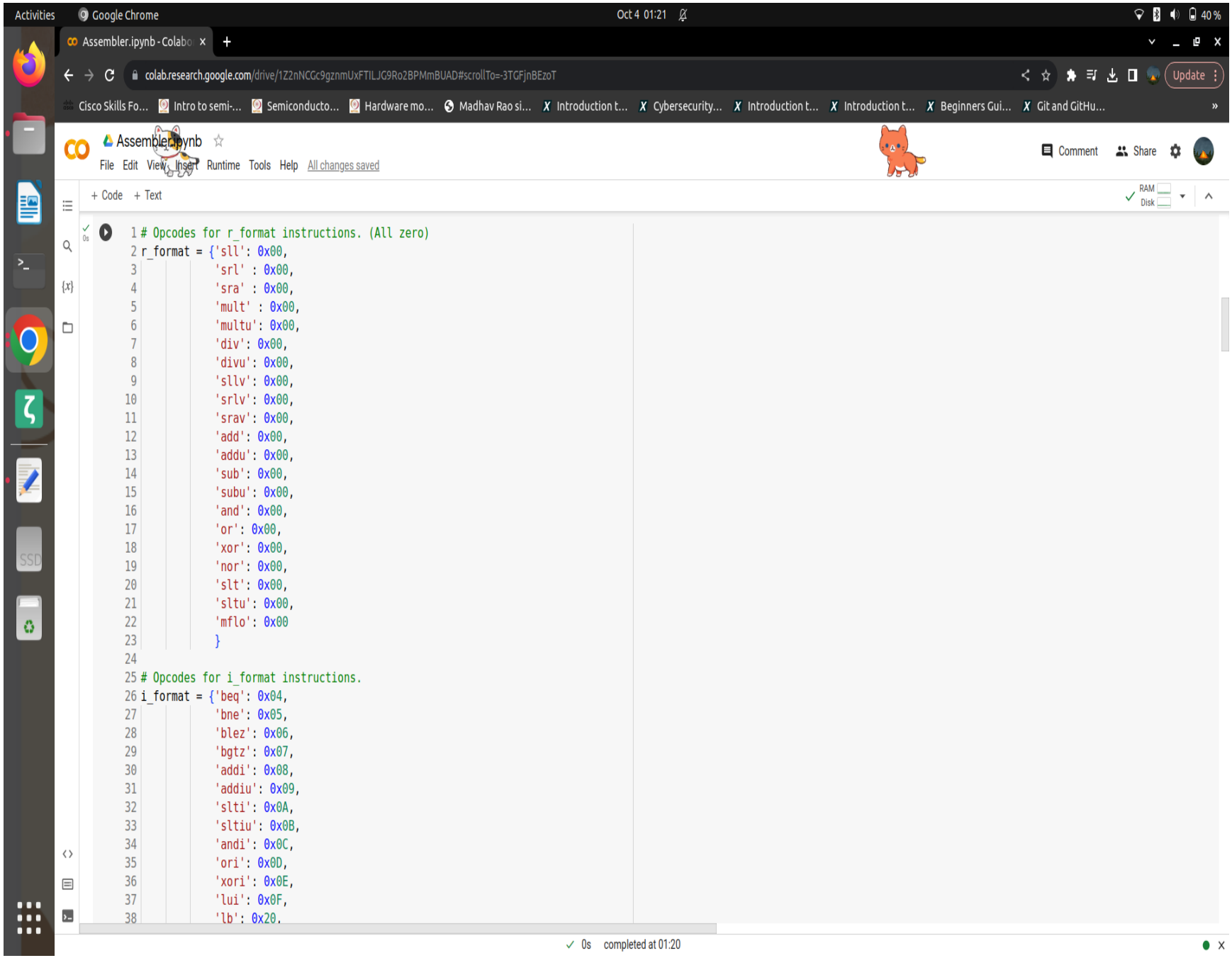


Figure 2.9. Dictionaries for each type of instruction, register numbers

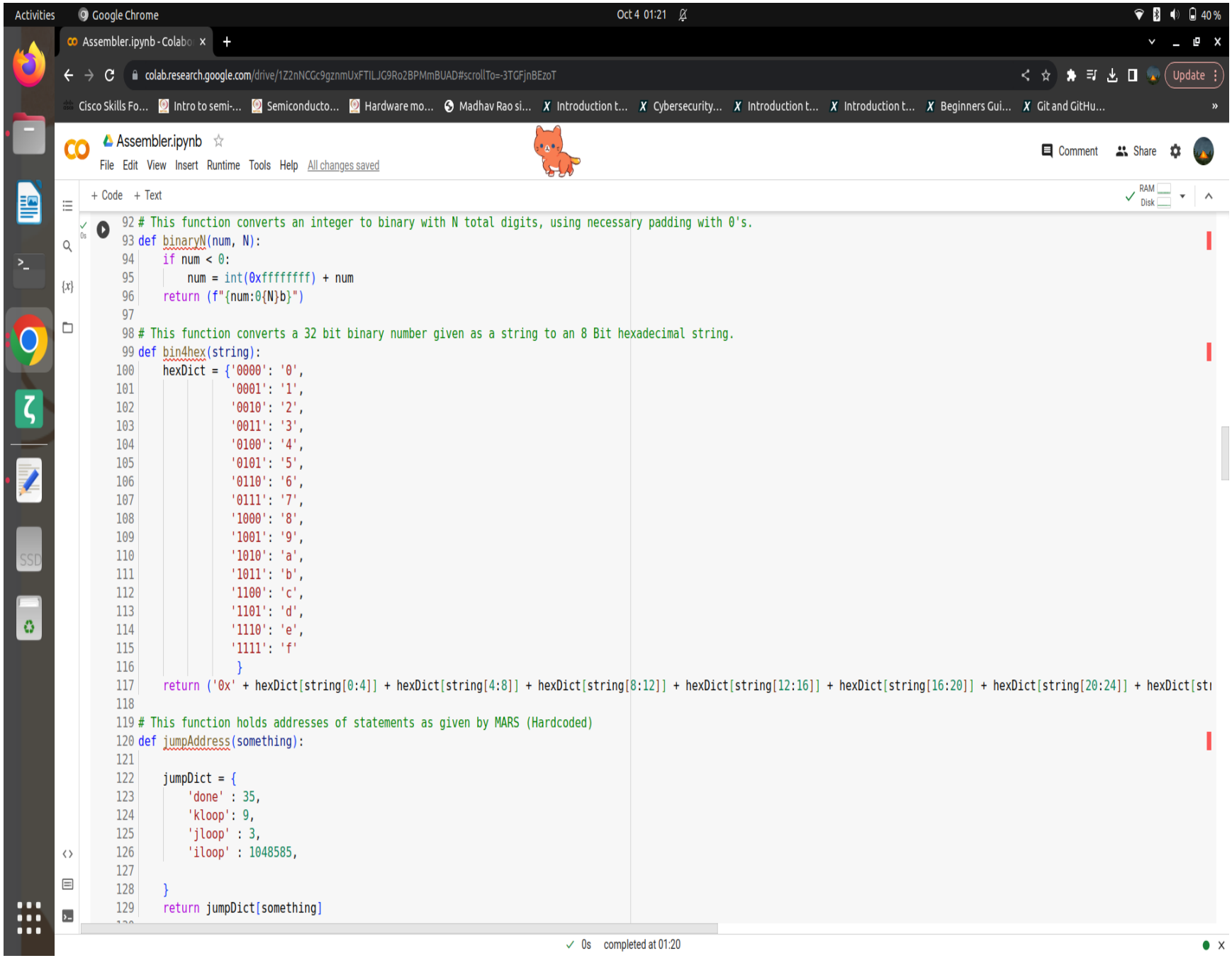


Figure 2.10. Function for shift and jump

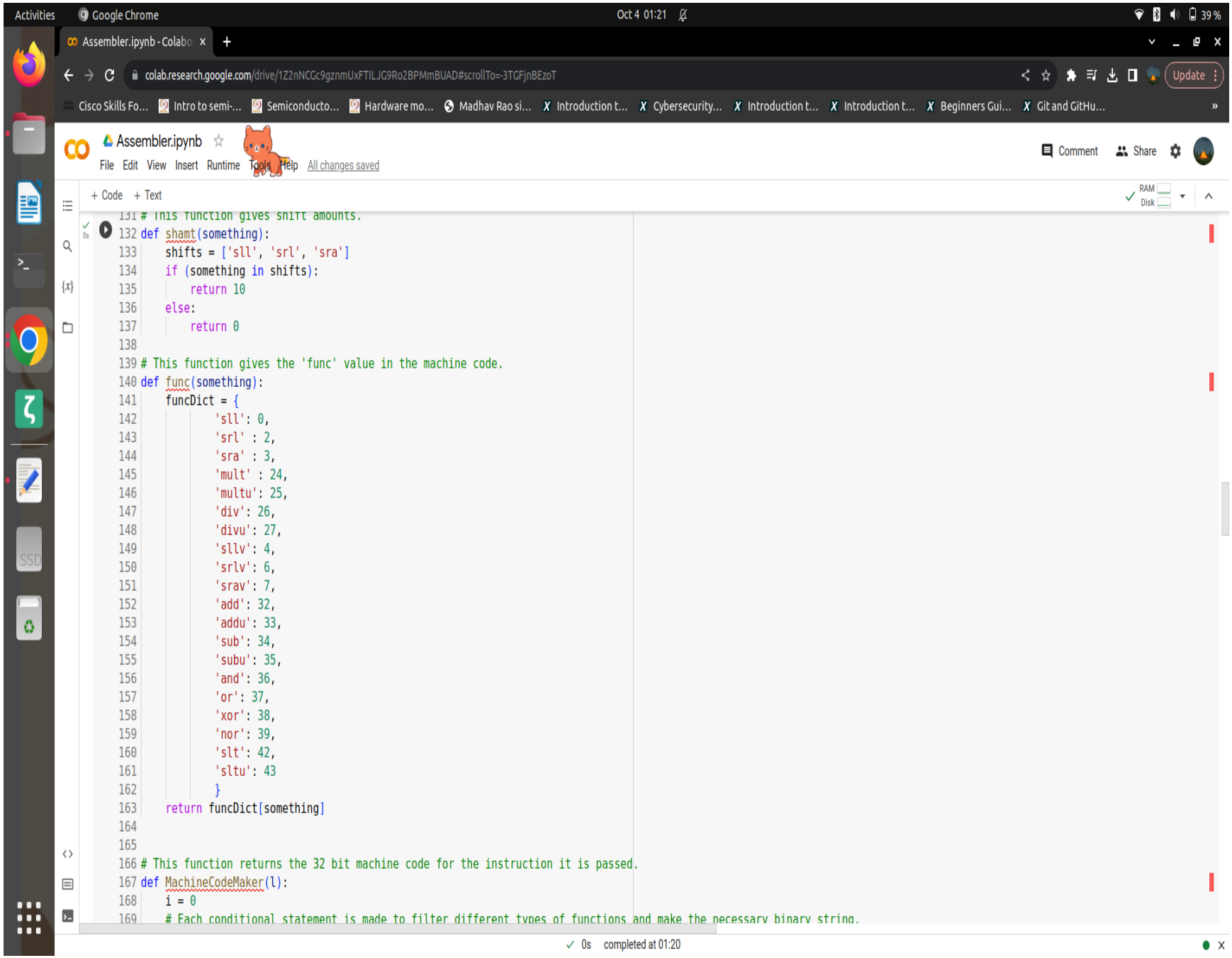


Figure 2.11. Branch target addresses (hardcoded)

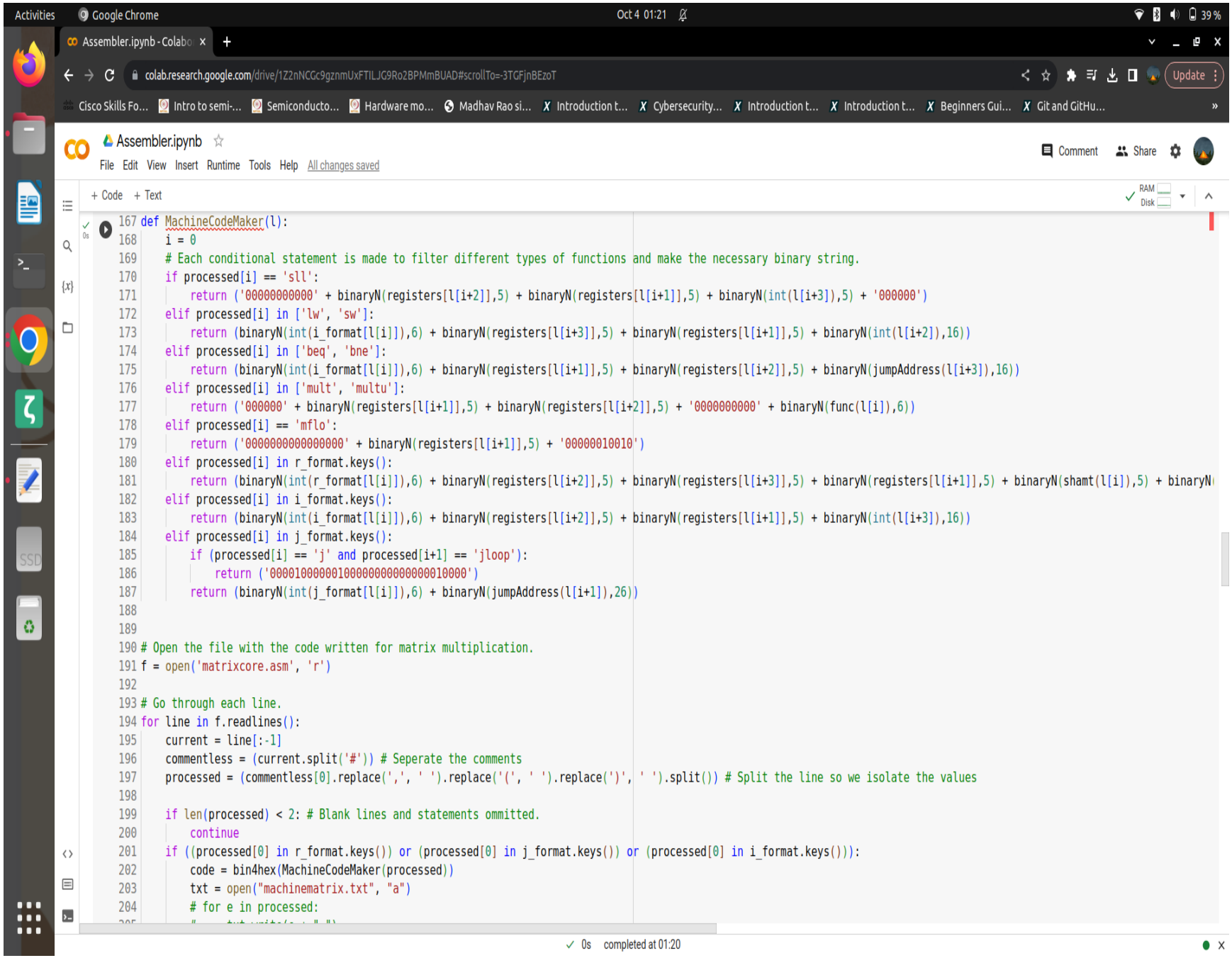


Figure 2.12. File I/O

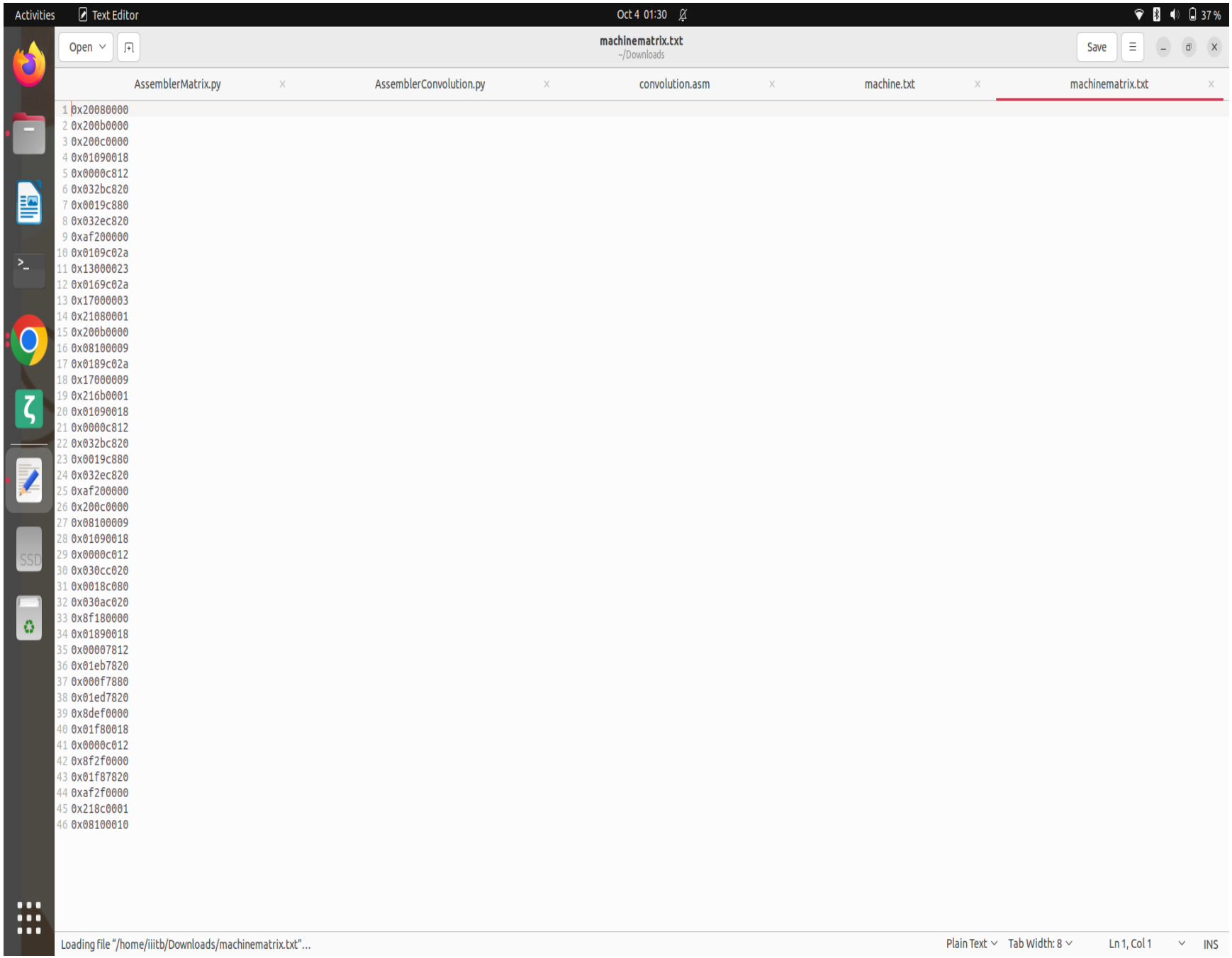


Figure 2.13. Output text file

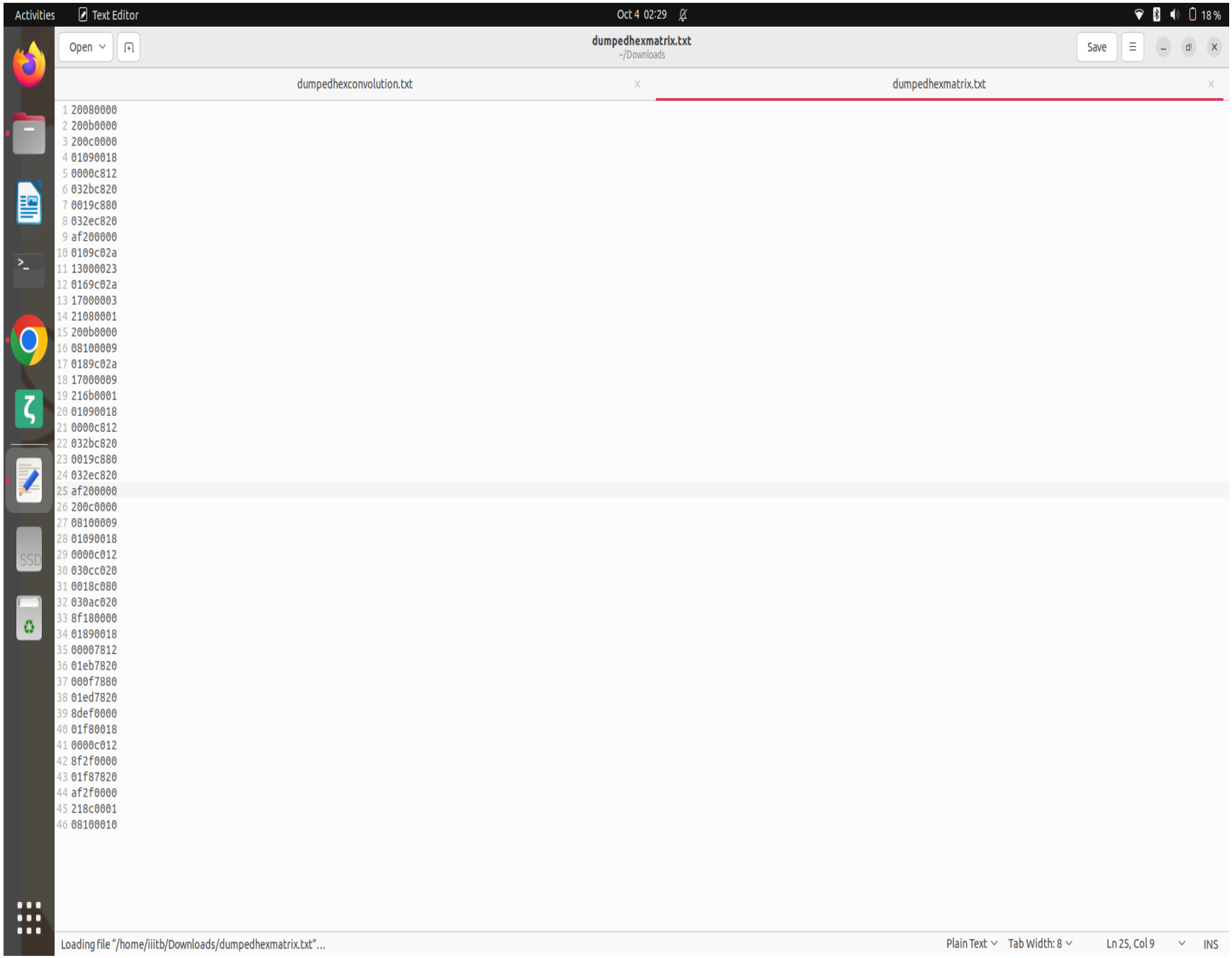


Figure 2.14. MARS machine code output text file