

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.

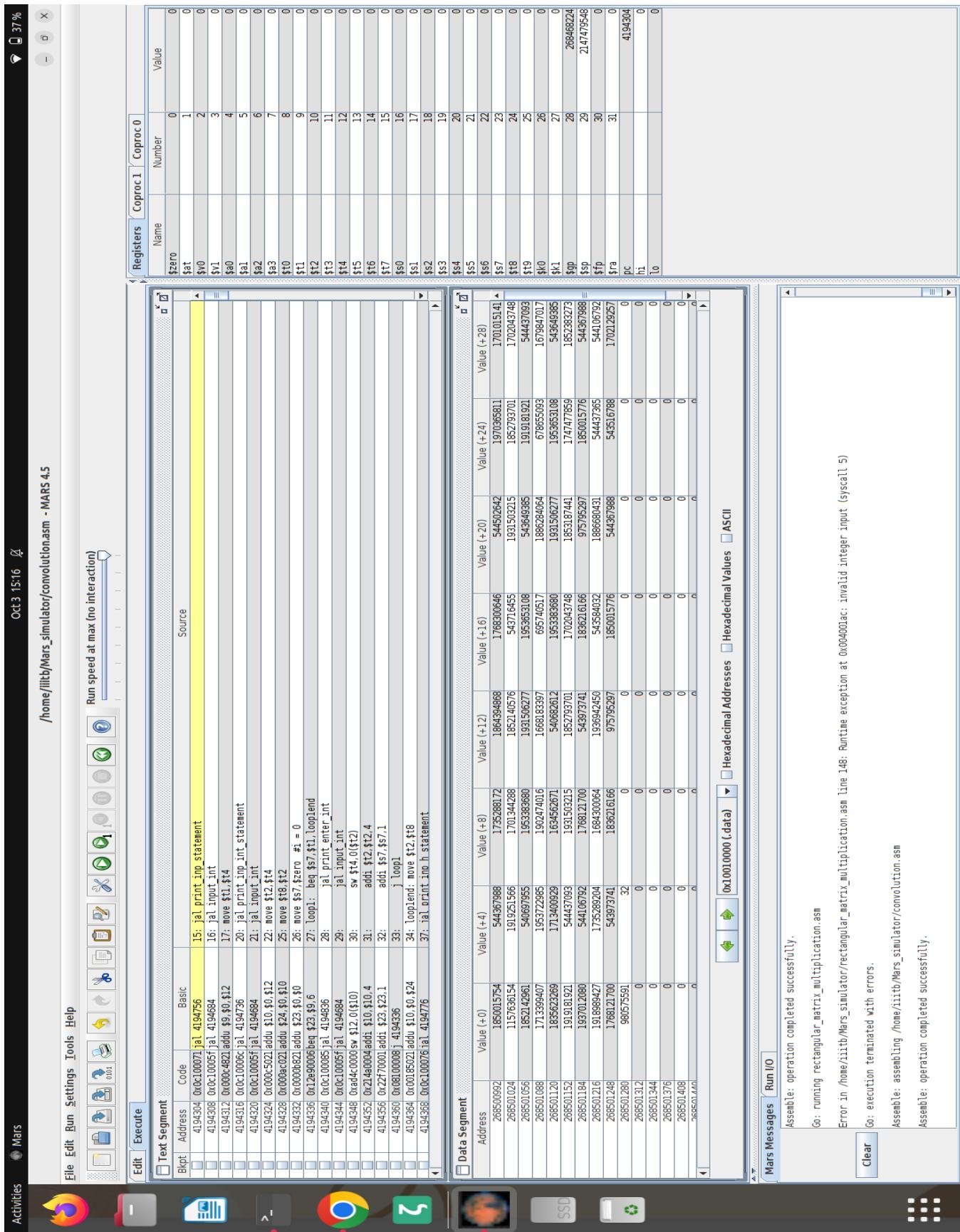


Figure 1.1. Initial State after assembly

Oct 3 15:20 4

/home/jilltb/Mars_simulator/convolution.asm - MARS 4.5

File Edit Run Settings Tools Help

Registers Coproc1 Coproc0

Name	Number	Value
\$zero		0
\$at		1
\$v0		2
\$v1		3
\$a0		4
\$a1		5
\$a2		6
\$a3		7
\$t0		8
\$t1		9
\$t2		10
\$t3		11
\$t4		12
\$t5		13
\$t6		14
\$t7		15
\$t8		16
\$t9		17
\$s0		18
\$s1		19
\$s2		20
\$s3		21
\$s4		22
\$s5		23
\$s6		24
\$s7		25
\$s8		26
\$s9		27
\$s10		28
\$s11		29
\$s12		30
\$s13		31
\$s14		32
\$s15		33
\$s16		34
\$s17		35
\$s18		36
\$s19		37
\$s20		38
\$s21		39
\$s22		40
\$s23		41
\$s24		42
\$s25		43
\$s26		44
\$s27		45
\$s28		46
\$s29		47
\$s30		48
\$s31		49

Text Segment

Bkpt	Address	Code	Basic
	4194304	0x0c100071jal 414956	15: jal print_im1_statement
	4194308	0x0c10005fjal 4149584	16: jal input_int
	4194312	0x0c10005fjal addiu \$0,\$0,\$12	17: move \$t1,\$14
	4194316	0x0c10005cjal 4149736	20: jal print_im1_int statement
	4194320	0x0c10005fjal addiu \$10,\$10,\$12	21: jal input_int
	4194324	0x0c0ac021addu \$24,\$0,\$10	22: move \$t2,\$12
	4194328	0x0c000821addu \$23,\$0,\$0	25: move \$t3,\$12
	4194332	0x12800006fleq \$57,\$t0,\$0	26: move \$t5,\$t0
	4194336	0x12800006fleq \$57,\$t1,\$0	27: loop: beq \$t5,\$t1,loopend
	4194340	0x0c00005fjal 414936	28: jal print_enter_int
	4194344	0x0c10005fjal 4149584	29: jal input_int
	4194348	0x0d4c0000sw \$12,0(\$t2)	30: sw \$t4,0(\$t2)
	4194352	0x214e0004addi \$10,\$10,4	31: addi \$t2,\$t2,4
	4194356	0x22f70001addi \$23,\$23,1	32: addi \$t7,\$t7,1
	4194360	0x08000008jal 419436	33: loop:
	4194364	0x0c00005fjal addiu \$10,\$0,\$24	34: loopend: move \$t2,\$t8
	4194368	0x0c000076jal 4149776	37: jal print_im1_statement
	4194372	0x0c10005fjal 4149584	38: jal input_int
	4194376	0x0c00005fjal addiu \$11,\$0,\$12	39: move \$t3,\$14
	4194380	0x0c100057jal 4149796	42: jal print_im1_int statement
	4194384	0x0c10005fjal 4149584	43: jal input_int
	4194388	0x0c00005fjal addiu \$13,\$0,\$12	44: move \$t5,\$14
	4194392	0x0c00005fjal addiu \$24,\$0,\$13	47: move \$t8,\$15
	4194396	0x0c00005fjal addiu \$23,\$0,\$0	48: move \$t7,\$zero #1 = 0
	4194400	0x12800006fleq \$23,\$11,6	49: loop2: beq \$t5,\$t3,loopend
	4194404	0x0c10005fjal 414936	50: jal print_enter_int
	4194408	0x0c00005fjal 4149584	51: jal input_int
	4194412	0x0d4c0000sw \$12,0(\$t3)	52: sw \$t4,0(\$t5)
	4194416	0x214b0004addi \$13,\$13,4	53: addi \$t5,\$t5,4
	4194420	0x22f70001addi \$23,\$23,1	54: addi \$t7,\$t7,1
	4194424	0x08000038jal 4194000	55: loop2
	4194428	0x0c00005fjal addiu \$13,\$0,\$24	56: loopend: move \$t5,\$18
	4194432	0x0c100058jal 4149816	59: jal print_out_int statement
	4194436	0x0c00005fjal 4149584	60: jal input_int
	4194440	0x0c000702jal addu \$14,\$0,\$12	61: move \$t6,\$14
	4194444	0x21480000addiu \$10,\$10,0	62: addi \$t0,\$12,#0
	4194448	0x214c0000addiu \$12,\$13,0	66: addi \$t4,\$15,0 #hi
	4194452	0x21490000addiu \$25,\$14,0	67: addi \$t9,\$16,0 #y
	4194456	0x012618820add \$1,\$9,\$11	68: add \$s1,\$t1,\$t3
	4194460	0x0202218822sub \$1,\$17,\$1	69: subi \$s1,\$s1,1
	4194464	0x200f0000addiu \$15,\$0,0	70: addi \$t7,\$0,0 #n=0
	4194468	0x20180000addiu \$24,\$0,0	71: addi \$t8,\$0,0 #k=0
	4194472	0x20180000addiu \$25,\$0,0	72: subi \$s1,\$19,\$t7,2
	4194476	0x000ff880s1,\$12,\$15,2	73: add \$t9,\$t9,\$t6
	4194480	0x032e6820add \$25,\$25,\$14	74: sw \$0,0(\$t9)
	4194484	0xa1200000sw \$0,\$25	77: slt \$s2,\$t7,\$s1 #t<1
	4194488	0x01f1902as1,\$1,\$15,17	78: beq \$s2,\$0,done
	4194492	0x12400022deq \$1,\$0,34	79: slt \$s2,\$t8,\$s1 #t<1
	4194496	0x0399902as1,\$1,\$24,\$9	80: bne \$s2,\$0,kloop #<1 => inner loop
	4194500	0x16400006bne \$1,\$0,6	81: addi \$t7,\$17,1 #n = n + 1
	4194504	0x21fe0001addiu \$15,\$15,1	82: addi \$t8,\$0,0 #reset k
	4194508	0x20180000addiu \$24,\$0,0	

Figure 1.2. Text segment

Oct 3 15:20 45 %

/home/jilltb/Mars_simulator/convolution.asm - MARS 4.5

Registers **Coproc 1** **Coproc 0**

Name	Number	Value
\$zero	0	0
\$at	1	26850992
\$v0	2	10
\$v1	3	0
\$a0	4	26850992
\$a1	5	0
\$t2	6	0
\$s3	7	0
\$t0	8	268501312
\$t1	9	7
\$t2	10	268501312
\$t3	11	4
\$t4	12	4
\$t5	13	268501344
\$t6	14	268501404
\$s0	15	0
\$s1	16	0
\$s2	17	7
\$s3	18	4
\$s4	19	4
\$s5	20	4
\$s6	21	0
\$s7	22	0
\$s8	23	7
\$s9	24	0
\$s10	25	0
\$s11	26	0
\$s12	27	0

Text Segment

Bkt	Address	Code	Basic	Source
	4194508	Or 2080000 addi \$24,\$0,0	82:	addi \$18,\$0,0 #reset k
	4194512	Or 00000f880 s1,\$125,\$15,\$2	83:	sl \$17,\$2,\$#14
	4194516	Or 0032e820 add \$25,\$14	84:	add \$t1,\$19,\$16,\$#14 + base address
	4194520	Or 00f200000 sw \$0,\$125)	85:	sw \$0,\$159,\$#14
	4194524	Or 0080002e l 4194688	86:	l.nloop
	4194528	Or 017f80022 sub \$18,\$15,\$24	89:	sub \$s5,\$17,\$18,\$#14
	4194532	Or 21730000 addi \$19,\$11,0	90:	addi \$s3,\$13,0
	4194536	Or 00278982a s1t \$19,\$19,\$18	91:	slt \$s5,\$s3,\$s2 #length 2 < n-k
	4194540	Or 2854fffff s1t \$18,-\$18,-1	92:	s1t \$s4,\$s2,-1 #n-k < 1
	4194544	Or 002d48825 or \$19,\$19,\$20	93:	or \$s3,\$s3,\$s4
	4194548	Or 23dffffc addi \$29,\$23,-4	94:	addi \$sp,\$sp,-4
	4194552	Or a1fb20000 sw \$18,0(\$29)	95:	lw \$s2,0(\$sp)
	4194556	Or 0001260003 beq \$19,\$0,3	96:	beq \$s3,0,\$update #if (n-k)<12 and (n-k)>-1
	4194560	Or 00f23810001 addi \$24,\$24,1	97:	addi \$s8,\$18,1 #k = k+1
	4194564	Or 002d0004 addi \$29,\$29,4	98:	addi \$sp,\$sp,4
	4194568	Or 0080002e l 4194688	99:	l.nloop
	4194572	Or 0f8fb20000 lw \$18,0(\$29)	102:	lw \$s2,0(\$sp)
	4194576	Or 000129980 s1,\$18,\$18,2	103:	sl \$1,\$s2,\$s2,2
	4194580	Or 002d49020 add \$18,\$18,\$12	104:	add \$s4,\$s2,\$t4
	4194584	Or 008e530000 lw \$19,0(\$18)	105:	lw \$s3,0(\$s2) # [n-k]
	4194588	Or 000189080 s1,\$18,\$124,2	106:	sl \$1,\$s5,\$18,2
	4194592	Or 002d480020 add \$18,\$18,\$8	107:	add \$s5,\$s2,\$t0
	4194596	Or 008e540000 lw \$20,0(\$18)	108:	lw \$s4,0(\$s2) # [n-k]
	4194600	Or 002d740008 mult \$19,\$120	109:	mult \$s3,\$s4 #h[n-k]*x[k]
	4194604	Or 000009812 lfto \$19	110:	mflo \$s3
	4194608	Or 00f20320000 lw \$18,0(\$25)	111:	lw \$s2,0(\$19) # [n]
	4194612	Or 002d39820 add \$18,\$18,\$19	112:	add \$s1,\$s2,\$s3 # [n-k]
	4194616	Or a1f330000 sw \$19,0(\$25)	113:	sw \$s3,0(\$t9)
	4194620	Or 002380001 addi \$24,\$24,1	114:	addi \$s8,\$18,1 #k = k+1
	4194624	Or 002d0004 addi \$29,\$29,4	115:	addi \$sp,\$sp,4
	4194628	Or 0080002e l 4194688	116:	l.nloop
	4194632	Or 001264820 add \$29,\$11	121:	add \$t1,\$t1,\$t3
	4194636	Or 002010001 addi \$18,\$10,1	122:	subi \$t1,\$t1,1
	4194640	Or 001214822 sub \$9,\$9,\$1	123:	move \$t7,\$zero #i = 0
	4194644	Or 000000821 addu \$23,\$0,\$0	124:	loop: beq \$s7,\$t1,end
	4194648	Or 0000000061neq \$23,\$9,6	125:	lv \$t4,0(\$t6)
	4194652	Or 0000000001 lw \$12,0(\$14)	126:	jal print_int
	4194656	Or 0000000003 jal 4194700	127:	jal print_line
	4194660	Or 0000000007 jal 4194716	128:	addi \$t6,\$16,4
	4194664	Or 0021ce0004 addi \$14,\$14,4	129:	addi \$t7,\$7,1
	4194668	Or 0000000001 addi \$23,\$23,1	130:	j.loop
	4194672	Or 0000260006 l 4194698	132:	end: li \$t0,10
	4194676	Or 002d000000 addiu \$2,\$10,10	133:	syscall
	4194680	Or 000000000c syscall	135:	input_int: li \$t0,5
	4194684	Or 002d000005 addiu \$2,\$0,5	136:	syscall
	4194688	Or 000000000c syscall	137:	move \$t4,\$t0
	4194692	Or 0000260021 addu \$12,\$0,\$2	138:	jl tra
	4194696	Or 00800008 lw \$31	140:	print_int: li \$t0,1
	4194700	Or 002d0001 addiu \$2,\$0,1	141:	move \$t0,\$t4
	4194704	Or 000002021 addu \$4,\$0,\$12	142:	syscall
	4194708	Or 000000000c syscall	143:	jr tra
	4194712	Or 0032e00008 lw \$31		

Figure 1.3. Text segment

Oct 3 15:20 45

/home/jilltb/Mars_simulator/convolution.asm - MARS 4.5

Text Segment

Bkt	Address	Code	Basic	Source
	4194658	0x12800006lreq \$23,\$9,6	124: loop: beq \$7,\$t1,end	
	4194652	0x80cc0000lw \$12,\$1\$14]	125: lw \$14,0(\$16)	
	4194656	0x0c000003jal \$1,40400	126: jal print_int	
	4194650	0x0c000007jal \$1,40416	127: jal print_line	
	4194654	0x21ce0004addi \$14,\$14,4	128: addi \$16,\$16,4	
	4194658	0x22770001addi \$23,\$23,1	129: addi \$7,\$7,1	
	4194672	0x08800056l,4194648	130: j loop	
	4194676	0x24020004addi \$2,\$0,10	132: end: li \$10,10	
	4194680	0x00000000c syscall	133: syscall	
	4194684	0x24020005addiu \$2,\$0,5	135: input_int: li \$v0,5	
	4194688	0x00000000c syscall	136: syscall	
	4194692	0x000026021addu \$12,\$0,\$2	137: mov \$14,\$v0	
	4194696	0x03ae0008ljr \$31	138: jr \$ra	
	4194700	0x24020001addiu \$2,\$0,1	140: print_int: li \$v0,1	
	4194704	0x00000001jal \$1,40412	141: move \$80,\$14	
	4194708	0x00000000c syscall	142: syscall	
	4194712	0x03ae0008ljr \$31	143: jr \$ra	
	4194716	0x24020004addiu \$2,\$0,4	145: print_line:li \$v0,4	
	4194720	0x3e010001lui \$1,4097	146: la \$a0,_next_line	
	4194724	0x3e010000ori \$4,\$1,0		
	4194728	0x00000000c syscall	147: syscall	
	4194732	0x03ae0008ljr \$31	148: jr \$ra	
	4194736	0x24020004addiu \$2,\$0,4	150: print_int_statement: li \$v0,4	
	4194740	0x3e010001lui \$1,4097	151: la \$a0,_mp_int_statement	
	4194744	0x3e040009lori \$4,\$1,73		
	4194748	0x00000000c syscall	152: syscall	
	4194752	0x03ae0008ljr \$31	153: jr \$ra	
	4194756	0x24020004addiu \$2,\$0,4	155: print_int_statement: li \$v0,4	
	4194760	0x3e010001lui \$1,4097	156: la \$a0,_mp_statement	
	4194764	0x3e040009lori \$4,\$1,2		
	4194768	0x00000000c syscall	157: syscall	
	4194772	0x03ae0008ljr \$31	158: jr \$ra	
	4194776	0x24020004addiu \$2,\$0,4	159: print_int_h_statement: li \$v0,4	
	4194780	0x3e010001lui \$1,4097	160: la \$a0,_mp_h_statement	
	4194784	0x3e040009lori \$4,\$1,35		
	4194788	0x00000000c syscall	161: syscall	
	4194792	0x03ae0008ljr \$31	162: jr \$ra	
	4194796	0x24020004addiu \$2,\$0,4	164: print_int_h_in_statement: li \$v0,4	
	4194800	0x3e010001lui \$1,4097	165: la \$a0,_mp_h_int_statement	
	4194804	0x3e040009lori \$4,\$1,145		
	4194808	0x00000000c syscall	166: syscall	
	4194812	0x03ae0008ljr \$31	167: jr \$ra	
	4194816	0x24020004addiu \$2,\$0,4	170: print_out_int_statement: li \$v0,4	
	4194820	0x3e010001lui \$1,4097	171: la \$a0,out_int_statement	
	4194824	0x3e040009lori \$4,\$1,218		
	4194828	0x00000000c syscall	172: syscall	
	4194832	0x03ae0008ljr \$31	173: jr \$ra	
	4194836	0x24020004addiu \$2,\$0,4	175: print_enter_int: li \$v0,4	
	4194840	0x3e010001lui \$1,4097	176: la \$a0,enter_int	
	4194844	0x3e040009lori \$4,\$1,274		
	4194848	0x00000000c syscall	177: syscall	
	4194852	0x03ae0008ljr \$31	178: jr \$ra	

Figure 1.4. Text segment

Oct 3 15:19 45% /home/lllb/Mars_simulator/convolution.asm - MARS 4.5

Registers

Name	Number	Value
\$zero	0	0
\$at	1	268503992
\$v0	2	10
\$v1	3	0
\$a0	4	268503992
\$a1	5	0
\$s2	6	0
\$s3	7	0
\$t0	8	268503132
\$t1	9	7
\$t2	10	268503132
\$t3	11	4
\$t4	12	4
\$t5	13	26850344
\$t6	14	14
\$t7	15	7
\$t8	16	0
\$t9	17	16
\$t10	18	0
\$t11	19	4
\$t12	20	4
\$t13	21	0
\$t14	22	0
\$t15	23	0
\$t16	24	0
\$t17	25	0
\$t18	26	0
\$t19	27	0
\$t20	28	0
\$t21	29	0
\$t22	30	0
\$t23	31	0
\$t24	32	0
\$t25	33	0
\$t26	34	0
\$t27	35	0
\$t28	36	0
\$t29	37	0
\$t30	38	0
\$t31	39	0

Text Segment

Bkpt	Address	Code	Basic	Source
	4194304	0x0c100071jal 414956	15. jal print_im1_statement	
	4194308	0x0c10005fjal 4149584	16. jal input_int	
	4194312	0x0c1000482jaldu \$0 \$0 \$12	17. move \$t1 \$14	
	4194316	0x0c10006cjal 4149736	20. jal print_im1_int statement	
	4194320	0x0c10005fjal 4149584	21. jal input_int	
	4194324	0x0c0c5021addu \$10 \$10 \$12	22. move \$t2 \$14	
	4194328	0x000ac021addu \$24 \$0 \$10	25. move \$t8 \$12	
	4194332	0x00000821addu \$23 \$0 \$0	26. move \$t7 \$zero #1 = 0	
	4194336	0x128900061beq \$23,\$19,6	27. loop: beq \$t5,\$1,t1,loopend	
	4194340	0x0c00005jal 4149536	28. jal print_enter_int	
	4194344	0x0c10005fjal 4149584	29. jal input_int	

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+12)	Value (+16)	Value (+20)	Value (+24)	Value (+28)
268503992	1880015754	543267988	1752388172	1964904898	178800646	545052642	1970365931	170101541
268503024	115756154	1919251566	1701344288	1852140576	153716455	1931503215	1852793701	1702047478
268503056	1882149861	540697955	195383680	1931563277	198563108	543692885	191981921	544431093
268503088	173395407	1553722865	1902471016	1688183597	69540517	188224064	1678555033	16796474016
268503120	185622269	1713400929	163452671	198338880	19331506277	1953653108	185384385	185283273
268503152	1919181921	544437093	1931503215	1852793701	1702047478	153187441	1747477859	1747477859
268503184	193701080	540106792	173528904	188621666	183642166	1850015776	1850015776	544437093
268503216	1918688427	188430064	1936442490	543584032	188660431	54443735	54443735	54443735
268503248	1789217700	543973741	1836216166	975795297	1850015776	544367988	544367988	544367988
268503280	980575591	32	0	0	0	0	0	0
268503312	1	2	3	4	0	0	0	0
268503344	1	2	2	1	0	0	0	0
268503376	1	4	9	15	16	11	4	0
268503408	0	0	0	0	0	0	0	0
268503440	0	0	0	0	0	0	0	0
268503472	0	0	0	0	0	0	0	0

Mars Messages

Enter length of first sequence: 4
 Enter starting address of first sequence(x) inputs(in decimal format): 2685031312
 Enter the integer: 1
 Enter the integer: 2
 Enter the integer: 3
 Enter the integer: 4
 Enter the length of second sequence: 4
 Enter starting address of second sequence(h) inputs(in decimal format): 2685031344
 Enter the integer: 1
 Enter the integer: 2
 Enter the integer: 3
 Enter the integer: 4
 Enter starting address of outputs (in decimal format): 2685031376
 Mars is finished running ...

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.

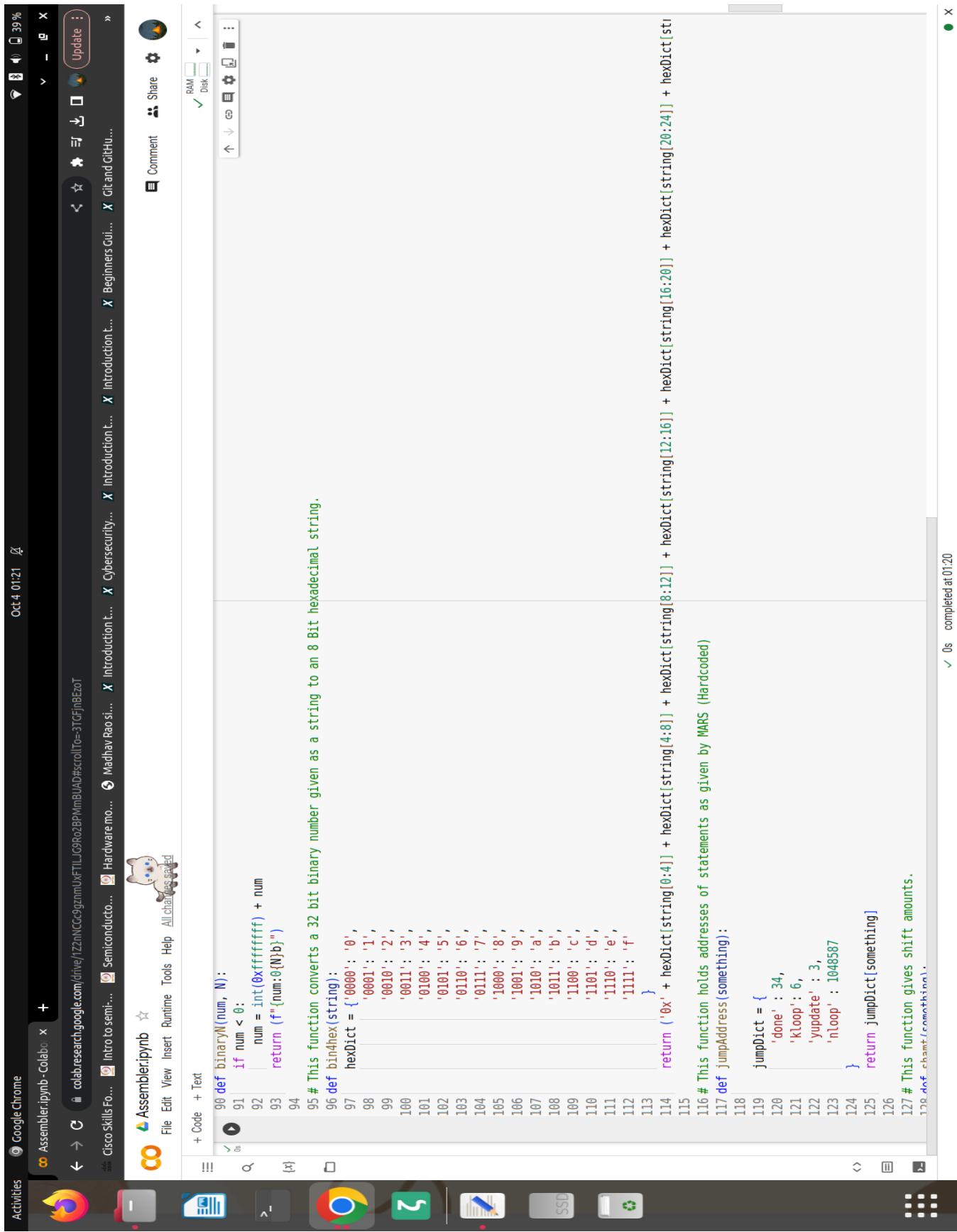
```

Activities
  Google chrome
  Assembleripyrb - Colab
Cisco Skills Fo... Intro to semi... Semiconducto... Hardwaredmo... Madhav/Rao si...
  Assembleripyrb
  File Edit View Insert Routine Tools Help All changes saved

+ Code + Text
Q x
  1 # Opcodes for r_format instructions. (All zero)
  2 r_format = { 'sll' : 0x00,
  3   'srl' : 0x00,
  4   'sra' : 0x00,
  5   'mult' : 0x00,
  6   'multu' : 0x00,
  7   'div' : 0x00,
  8   'divu' : 0x00,
  9   'sllv' : 0x00,
 10  'srlv' : 0x00,
 11  'srav' : 0x00,
 12  'add' : 0x00,
 13  'addu' : 0x00,
 14  'sub' : 0x00,
 15  'subu' : 0x00,
 16  'and' : 0x00,
 17  'or' : 0x00,
 18  'xor' : 0x00,
 19  'nor' : 0x00,
 20  'slt' : 0x00,
 21  'sltu' : 0x00,
 22  'mflo' : 0x00
}
  23
  24
  25 # Opcodes for i_format instructions.
  26 i_format = {'beq' : 0x04,
  27   'bne' : 0x05,
  28   'blez' : 0x06,
  29   'bgtz' : 0x07,
  30   'andi' : 0x08,
  31   'oridi' : 0x09,
  32   'slti' : 0x0A,
  33   'stiu' : 0x0B,
  34   'andi' : 0x0C,
  35   'ori' : 0x0D,
  36   'xori' : 0x0E,
  37   'lui' : 0x0F,
  38   'lb' : 0x20.
}
  39
  ✓ 0s completed at 01:20

```

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



```

Activities
  Google chrome
  Assembleripyrb - Colab
  Cisco Skills Fo...
  Intro to semi...
  Semiconductor...
  Hardware mo...
  Hardw...
  Beginner...
  Git and Gui...
  »
  Assembleripyrb
  File Edit View Insert Routine Tools Help All changes saved
  + Code + Text
  90 def binaryN(num, N):
  91     if num < 0:
  92         num = int(0xffffffff) + num
  93     return f"{num:{N}b}"
  94
  95 # This function converts a 32 bit binary number given as a string to an 8 Bit hexadecimal string.
  96 def bin4hex(string):
  97     hexDict = {'0000': '0',
  98                '0001': '1',
  99                '0010': '2',
 100               '0011': '3',
 101               '0100': '4',
 102               '0101': '5',
 103               '0110': '6',
 104               '0111': '7',
 105               '1000': '8',
 106               '1001': '9',
 107               '1010': 'a',
 108               '1011': 'b',
 109               '1100': 'c',
 110               '1101': 'd',
 111               '1110': 'e',
 112               '1111': 'f'
 113
 114     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[string[24:28]] + hexDict[string[28:32]])
  115
  116 # This function holds addresses of statements as given by MARS (Hardcoded)
  117 def jumpAddress(something):
  118
  119     jumpDict = {
 120         'done' : 34,
 121         'kloop' : 6,
 122         'yupdate' : 3,
 123         'nloop' : 1048587
 124     }
  125     return jumpDict[something]
  126
  127 # This function gives shift amounts.
  128 def chmod (command):
  ✓ 0s completed at 01:20

```

Figure 1.7. Branch target addresses (hardcoded)

Figure 1.8. File I/O

The screenshot shows a Linux desktop environment with several open windows. In the foreground, there is a terminal window titled "Text Editor" with the command "AssembleMatrix.py" running. The output of the script is displayed, listing integer values from 1 to 45. Above this terminal is another terminal window titled "AssemblerConvolution.py" which has run successfully, indicated by the message "convolution.asm". To the right of these terminals is a file manager window showing files like "machine.txt", "AssemblerConvolution.py", and "convolution.asm". The desktop bar at the bottom includes icons for a browser, file manager, terminal, and system status.

```

1. 0x21486000
2. 0x21a2c0000
3. 0x21d960000
4. 0x012bb8220
5. 0x280f90000
6. 0x281880000
7. 0x000fc880
8. 0x0122ec820
9. 0xaf7200000
10. 0x01f1902a
11. 0x12400022
12. 0x03990028
13. 0x16400006
14. 0x21ef0001
15. 0x281880000
16. 0x000fc880
17. 0x0122ec820
18. 0xaf7200000
19. 0x0810000b
20. 0x01ff89022
21. 0x217390000
22. 0x02729823
23. 0x2554ffff
24. 0x02749825
25. 0x23bdffff
26. 0xafb20000
27. 0x12666603
28. 0x23180001
29. 0x23bd0004
30. 0x0810000b
31. 0x8fb20000
32. 0x08129698
33. 0x024c5020
34. 0x8e536000
35. 0x001890000
36. 0x02489020
37. 0x8e546000
38. 0x027440018
39. 0x0009812
40. 0x8f326000
41. 0x02539820
42. 0xaf330000
43. 0x23180001
44. 0x23bd0004
45. 0x0810000b

```

Figure 1.9. Output text file

The screenshot shows a terminal window titled "dumpedhexconvolution.txt" with the path "/Downloads". The window title bar includes icons for volume, brightness, and battery, along with a "Save" button. The terminal interface has tabs for "PlainText" and "Ln 1, Col 1". The status bar at the bottom shows "Oct 4 02:28" and "18%".

```

Oct 4 02:28  ↵
dumpedhexconvolution.txt
-/Downloads

1 21480000
2 21ac0000
3 21d90000
4 612bb8820
5 20010001
6 032218822
7 2000f0000
8 20180000
9 0000fc880
10 032ee820
11 af200000
12 01f1902a
13 12400022
14 03099902a
15 16400006
16 21ef0001
17 20180000
18 000fc880
19 6325c820
20 af200000
21 0810000b
22 01ff9022
23 21739000
24 0277982a
25 2854ffff
26 02749825
27 23bdffff
28 afb20000
29 12600003
30 23180001
31 23bd0004
32 6010000b
33 8fb20000
34 00129880
35 024c9020
36 8e530000
37 60109000
38 02489020
39 8e540000
40 02749018
41 600099812
42 8f7320000
43 02539820
44 af330000
45 23180001
46 23bd0004
47 6810000b

Loading file "/home/jilbt/Downloads/dumpedhexconvolution.bcf" ...
Ln 1, Col 1      <  INS
Tab Width: 8      <  PlainText

```

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.

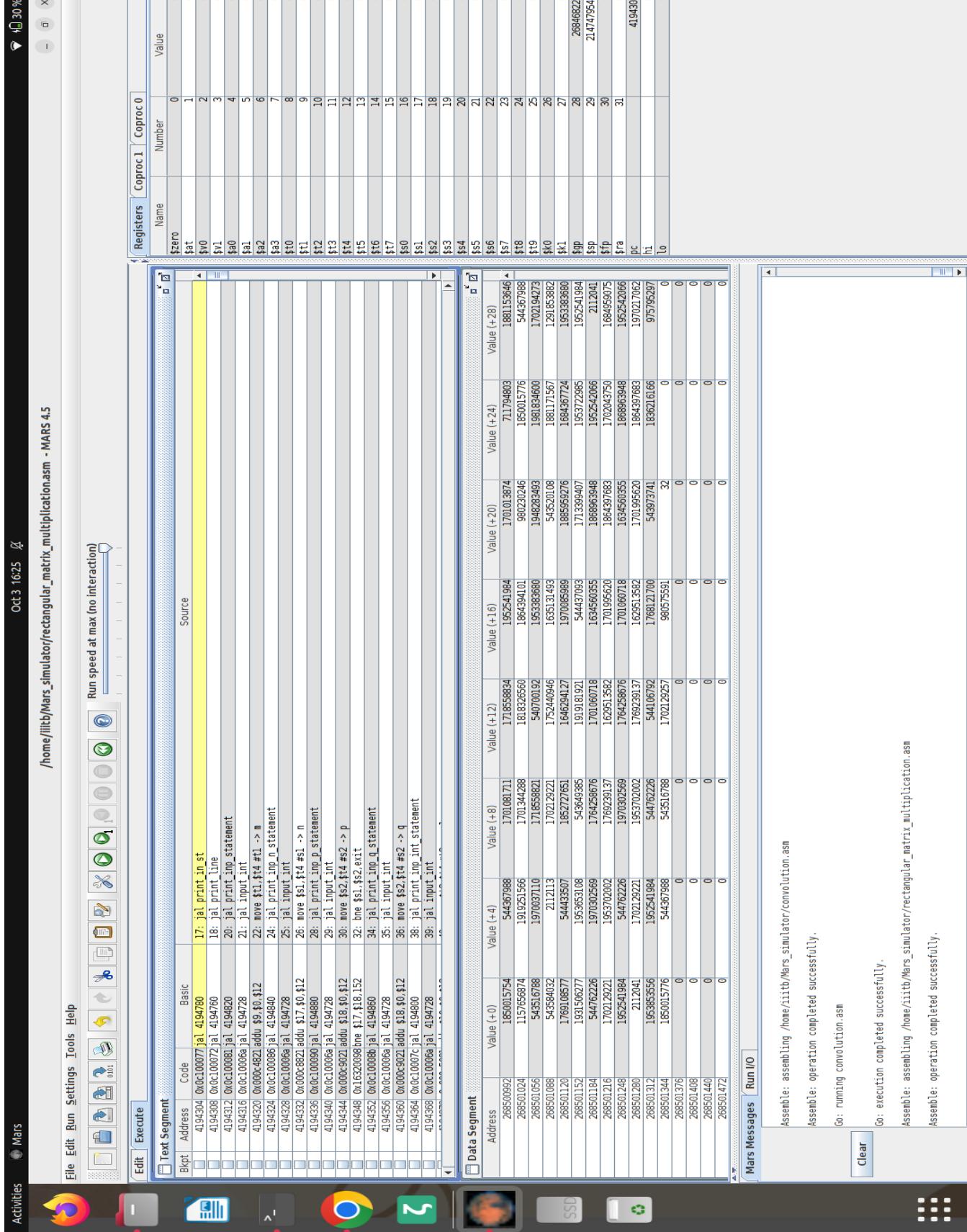


Figure 2.1. Initial State after assembly

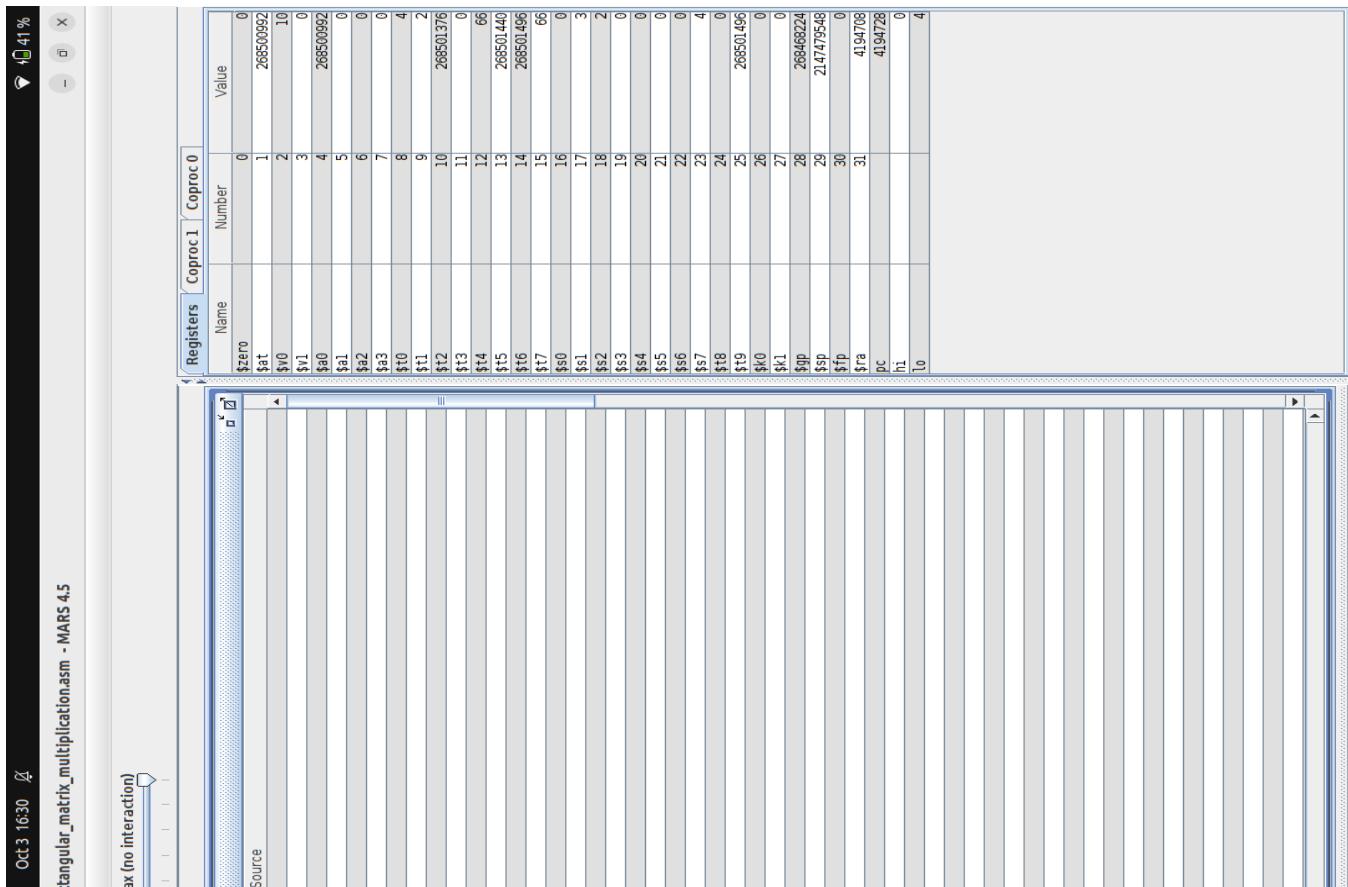
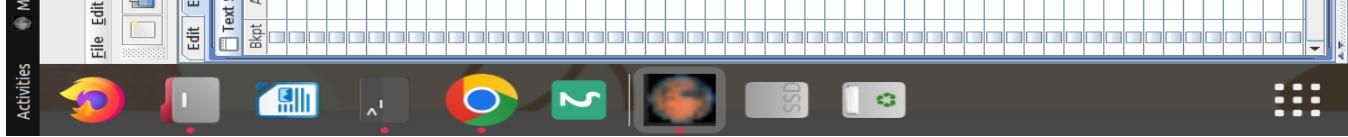


Figure 2.2. Text segment

Oct 3 16:31 41 %

/home/jilltb/Mars_Simulator/rectangular_matrix_multiplication.asm - MARS 4.5

Text Segment

Bkpt	Address	Code	Basic
	4194508	0x01200008 mult \$t0,\$t18	80: mult \$t0,\$t2 #t4q
	4194512	0x00000000 mflo \$25	80: mflo \$t9
	4194516	0x00000000 add \$25,\$11	82: add \$t9,\$t9 t#t4q
	4194520	0x00000000 sll \$1,15,25,2	83: sll \$t9,19,2 #t4q
	4194524	0x032e820 add \$25,\$25 #14	84: add \$t9,\$t9 t#(1t4q) + base address
	4194528	0x0f200000 sw \$0,\$t25	85: sw \$0,0(\$t25)
	4194532	0x0108c02a sll \$12,\$18,\$19	88: sll \$t0,\$t0,\$t1 #t1q
	4194536	0x13000033 beq \$t24,\$0,\$35	89: beq \$t8,\$0,done
	4194540	0x0172c02a sll \$24,\$11,\$18	90: sll \$t8,\$t3,\$t2 #t1q
	4194544	0x17000003 bne \$t24,\$0,\$3	91: bne \$t8,\$0,loop
	4194548	0x12100000 addi \$t0,\$18,1	92: addi \$t0,\$t0,1
	4194552	0x20000000 addi \$t1,\$0,0	93: addi \$t3,\$0,0
	4194556	0x00000039 1,4194532	94: lloop
	4194560	0x0108c02a sll \$24,\$12,\$17	97: sll \$t8,\$t4,\$t1
	4194564	0x17000009 bne \$t24,\$0,\$9	98: bne \$t8,\$0,loop
	4194568	0x12100000 addi \$t1,\$1,1	99: addi \$t3,\$t3,1
	4194572	0x01200008 mult \$t0,\$t18	100: mult \$t0,\$t2 #t1q
	4194576	0x00000000 mflo \$25	101: mflo \$t9
	4194580	0x032e820 add \$25,\$25 #11	102: add \$t9,\$t9 t#(1t4q)
	4194584	0x00000000 sll \$1,25,25,2	103: sll \$t8,\$t9,2 #t1q
	4194588	0x032e820 add \$25,\$25 #14	104: add \$t8,\$t9,\$t6 #(1t4q) + base address
	4194592	0x0f200000 sw \$0,\$t25	105: sw \$0,(\$t25)
	4194596	0x20000000 addi \$t2,\$0,0	106: addi \$t4,\$0,0
	4194600	0x0108c02a sll \$24,\$0,\$32	107: lloop
	4194604	0x01110010 mult \$t0,\$t17	110: mult \$t0,\$t1
	4194608	0x00000000 mflo \$24	111: mflo \$t8
	4194612	0x01030c020 add \$t2,\$24,\$12	112: add \$t8,\$t8 #t4
	4194616	0x00000000 sll \$1,24,2	113: sll \$t1,\$t8,2
	4194620	0x032ac020 add \$24,\$24,\$10	114: add \$t1,\$t8,\$t2
	4194624	0x10f7820 add \$15,\$15,\$13	115: lw \$t8,0(\$t18)
	4194628	0x01920008 mult \$12,\$18	116: mult \$t4,\$t2
	4194632	0x000007812 mflo \$15	117: mflo \$t7
	4194636	0x0108c020 add \$15,\$15,\$11	118: add \$t7,\$t7,\$t3
	4194640	0x00000000 sll \$15,15,2	119: sll \$t1,\$t7,2
	4194644	0x010f7820 add \$15,\$15,\$13	120: add \$t1,\$t7,\$t5
	4194648	0x12800000 beq \$t25,\$0,\$15	121: lw \$t7,0(\$t17)
	4194652	0x01f90018 mult \$t5,\$t24	122: mult \$t7,\$t8
	4194656	0x00000000 mflo \$24	123: mflo \$t8
	4194660	0x08f2f0000 lw \$15,0,\$t25	124: lw \$t7,0(\$t19)
	4194664	0x010f7820 add \$15,\$15,\$24	125: add \$t7,\$t7,\$t8
	4194668	0x0f2f0000 sw \$15,0,\$t25	126: sw \$t7,0(\$t19)
	4194672	0x12800000 beq \$t25,\$0,\$14	127: addi \$t4,\$t4,1
	4194676	0x08000040 l,4194644	128: lloop
	4194680	0x00000008 mult \$t5,\$t8	133: mult \$t1,\$t2
	4194684	0x00000402 mflo \$t8	134: mflo \$t0
	4194688	0x000000821 addu \$23,\$0,\$0	135: move \$t7,\$zero t#1 = 0
	4194692	0x12800006 beq \$t25,\$0,\$10	136: loop: beq \$t7,10, end
	4194696	0x080cc0000 lw \$12,0(\$t14)	137: lw \$t4,0(\$t16)
	4194700	0x00c10006 jal 4194744	138: jal print_int
	4194704	0x00000002 jal 4194760	139: jal print_line
	4194708	0x21fe0004 addi \$14,\$14,4	140: addi \$t6,\$t6,4
	4194712	0x22f70001 addi \$23,\$23,1	141: addi \$t7,\$t7,1

Figure 2.3. Text segment

Oct 3 16:31 41 %

/home/jilltb/Mars_Simulator/rectangular_matrix_multiplication.asm - MARS 4.5

Text Segment

Bkpt	Address	Code	Basic
	4194712	0x22f0001	addi \$3,\$23,1
	4194716	0x08000051	jr \$1,49492
	4194720	0x0200000a	addiu \$2,\$0,10
	4194724	0x0000000c	syscall
	4194728	0x24020005	addiu \$2,\$0,5
	4194732	0x0000000c	syscall
	4194736	0x00020002	addu \$12,\$0,\$2
	4194740	0x03000008	jr \$3,
	4194744	0x24020001	addiu \$2,\$0,1
	4194748	0x00000020	addiu \$4,\$0,\$12
	4194752	0x0000000c	syscall
	4194756	0x03000008	jr \$3,
	4194760	0x24020004	addiu \$2,\$0,4
	4194764	0x30010001	lui \$1,4097
	4194768	0x34240000	ori \$4,\$0
	4194772	0x0000000c	syscall
	4194776	0x03000008	jr \$3,
	4194780	0x24020004	addiu \$2,\$0,4
	4194784	0x30010001	lui \$1,4097
	4194788	0x34240002	ori \$4,\$1,2
	4194792	0x0000000c	syscall
	4194796	0x03000008	jr \$3,
	4194800	0x24020004	addiu \$2,\$0,4
	4194804	0x30010001	lui \$1,4097
	4194808	0x3424000d	ori \$4,\$1,157
	4194812	0x0000000c	syscall
	4194816	0x03000008	jr \$3,
	4194820	0x24020004	addiu \$2,\$0,4
	4194824	0x30010001	lui \$1,4097
	4194828	0x34240023	ori \$4,\$1,35
	4194832	0x0000000c	syscall
	4194836	0x03000008	jr \$3,
	4194840	0x24020004	addiu \$2,\$0,4
	4194844	0x30010001	lui \$1,4097
	4194848	0x3424003a	ori \$4,\$1,58
	4194852	0x0000000c	syscall
	4194856	0x03000008	jr \$3,
	4194860	0x24020004	addiu \$2,\$0,4
	4194864	0x30010001	lui \$1,4097
	4194868	0x34240051	ori \$4,\$1,81
	4194872	0x0000000c	syscall
	4194876	0x03000008	jr \$3,
	4194880	0x24020004	addiu \$2,\$0,4
	4194884	0x30010001	lui \$1,4097
	4194888	0x34240058	ori \$4,\$1,104
	4194892	0x0000000c	syscall
	4194896	0x03000008	jr \$3,
	4194900	0x24020004	addiu \$2,\$0,4
	4194904	0x30010001	lui \$1,4097
	4194908	0x34240060	ori \$4,\$1,224
	4194912	0x0000000c	syscall
	4194916	0x03000008	jr \$3,

Figure 2.4. Text segment

Oct 3 16:31 41 %

/home/jilltb/Mars_Simulator/rectangular_matrix_multiplication.asm - MARS 4.5

Text Segment

Bkt	Address	Code	Basic
	4194768	0x34240000ori \$4,\$1,0	159: syscall
	4194772	0x0000000c syscall	160: jr \$ra
	4194780	0x24020004addiu \$2,\$0,4	162: print_in_st:li \$10,4
	4194784	0x36010001lui \$1,4097	163: la \$a0,inp_st
	4194788	0x34240002ori \$4,\$1,2	164: syscall
	4194792	0x0000000c syscall	165: jr \$ra
	4194796	0x03800008ir \$32	167: print_inp_int_statement: li \$v0,4
	4194800	0x24020004addiu \$2,\$0,4	168: la \$a0,inp_int_statement
	4194804	0x36010001lui \$1,4097	169: syscall
	4194808	0x34240003ori \$4,\$1,157	170: jr \$ra
	4194812	0x0000000c syscall	172: print_inp_int_statement: li \$v0,4
	4194816	0x03800008ir \$32	173: la \$a0,inp_int_statement
	4194820	0x24020004addiu \$2,\$0,4	174: syscall
	4194824	0x36010001lui \$1,4097	175: jr \$ra
	4194828	0x34240023ori \$4,\$1,35	177: print_inp_n_statement: li \$v0,4
	4194832	0x0000000c syscall	178: la \$a0,inp_n_statement
	4194836	0x03800008ir \$32	179: syscall
	4194840	0x24020004addiu \$2,\$0,4	180: jr \$ra
	4194844	0x36010001lui \$1,4097	182: print_inp_q_statement: li \$v0,4
	4194848	0x34240033aori \$4,\$1,58	183: la \$a0,inp_q_statement
	4194852	0x0000000c syscall	184: syscall
	4194856	0x03800008ir \$32	185: jr \$ra
	4194860	0x24020004addiu \$2,\$0,4	187: print_inp_p_statement: li \$v0,4
	4194864	0x36010001lui \$1,4097	188: la \$a0,inp_p_statement
	4194868	0x34240051ori \$4,\$1,81	189: syscall
	4194872	0x0000000c syscall	190: jr \$ra
	4194876	0x03800008ir \$32	192: print_inp_h_int_statement: li \$v0,4
	4194880	0x24020004addiu \$2,\$0,4	193: la \$a0,inp_h_int_statement
	4194884	0x36010001lui \$1,4097	194: syscall
	4194888	0x34240068ori \$4,\$1,104	195: jr \$ra
	4194892	0x0000000c syscall	196: print_out_int_statement: li \$v0,4
	4194896	0x03800008ir \$32	198: la \$a0,out_int_statement
	4194900	0x24020004addiu \$2,\$0,4	199: syscall
	4194904	0x36010001lui \$1,4097	201: jr \$ra
	4194908	0x34240060ori \$4,\$1,224	203: print_enter_int: li \$v0,4
	4194912	0x0000000c syscall	204: la \$a0,enter_int
	4194916	0x03800008ir \$32	205: syscall
	4194920	0x24020004addiu \$2,\$0,4	206: jr \$ra
	4194924	0x36010001lui \$1,4097	207: print_out_statement: li \$v0,4
	4194928	0x34240124ori \$4,\$1,292	208: la \$a0,out_statement
	4194932	0x0000000c syscall	209: syscall
	4194936	0x03800008ir \$32	210: jr \$ra
	4194940	0x24020004addiu \$2,\$0,4	211: syscall
	4194944	0x36010001lui \$1,4097	212: la \$a0,out_statement
	4194948	0x34240162ori \$4,\$1,354	213: syscall
	4194952	0x0000000c syscall	214: jr \$ra
	4194956	0x03800008ir \$32	215: syscall
	4194960	0x24020004addiu \$2,\$0,4	216: la \$a0,out_statement
	4194964	0x36010001lui \$1,4097	217: syscall
	4194968	0x342400f1ori \$4,\$1,127	218: la \$a0,out_statement
	4194972	0x0000000c syscall	219: syscall

Figure 2.5. Text segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+12)	Value (+16)	Value (+20)	Value (+24)	Value (+28)
268501092	1850015754	544367988	1700687711	1718558834	1952541984	1701013874	711794803	1881153646
268501094	1157656874	191921566	1701344288	1818326560	1864394101	980230246	1850015776	544367988
268501096	545516788	1970067710	1718558821	540700192	1953388660	1948283493	1981834600	1702194273
268501098	545580432	2112113	1702129221	175244046	1635131493	543520108	1881171567	1291853882
268501120	1769106577	544493507	185227651	164629427	1970065988	1885559276	1684367724	1953388680
268501152	1931506277	1933653108	545649385	1919181921	544473093	1713399407	1953722905	195241984
268501184	544762226	1970302559	1764256676	1701660718	1634560355	1868863948	1952542066	2112041
268501216	1702129221	193372002	1769239137	1679513882	170195620	1864397683	1702043750	168459075
268501248	1052541984	544772226	1970302569	1764256676	1701660718	1634560355	1868935948	195242066
268501280	2112041	1702129221	1953702002	1769239137	1629513582	170195620	1864397683	1970217062
268501312	195385556	1952541984	544762226	544466792	176821700	54397341	186226166	97595297
268501344	1850015776	544367988	545516788	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

Mars Messages Run I/O

Enter order of matrices m*n p*q
Enter the value of m: 2
Enter the value of n: 3
Enter the value of p: 3
Enter the value of q: 2
Enter starting address of first matrix inputs(in decimal format): 268501376
Enter the integer: 1
Enter the integer: 5
Enter the integer: 0
Enter the integer: 3
Enter the integer: 7
Enter the integer: 9
Enter starting address of second matrix inputs(in decimal format): 268501440
Enter the integer: 2
Enter the integer: 7
Enter the integer: 2
Enter the integer: 0
Enter the integer: 0
Enter the integer: 5
Enter starting address of output matrix (in decimal format): 268501480
12
7
20
66

Clear

... program is finished running ...

Figure 2.7. Terminal

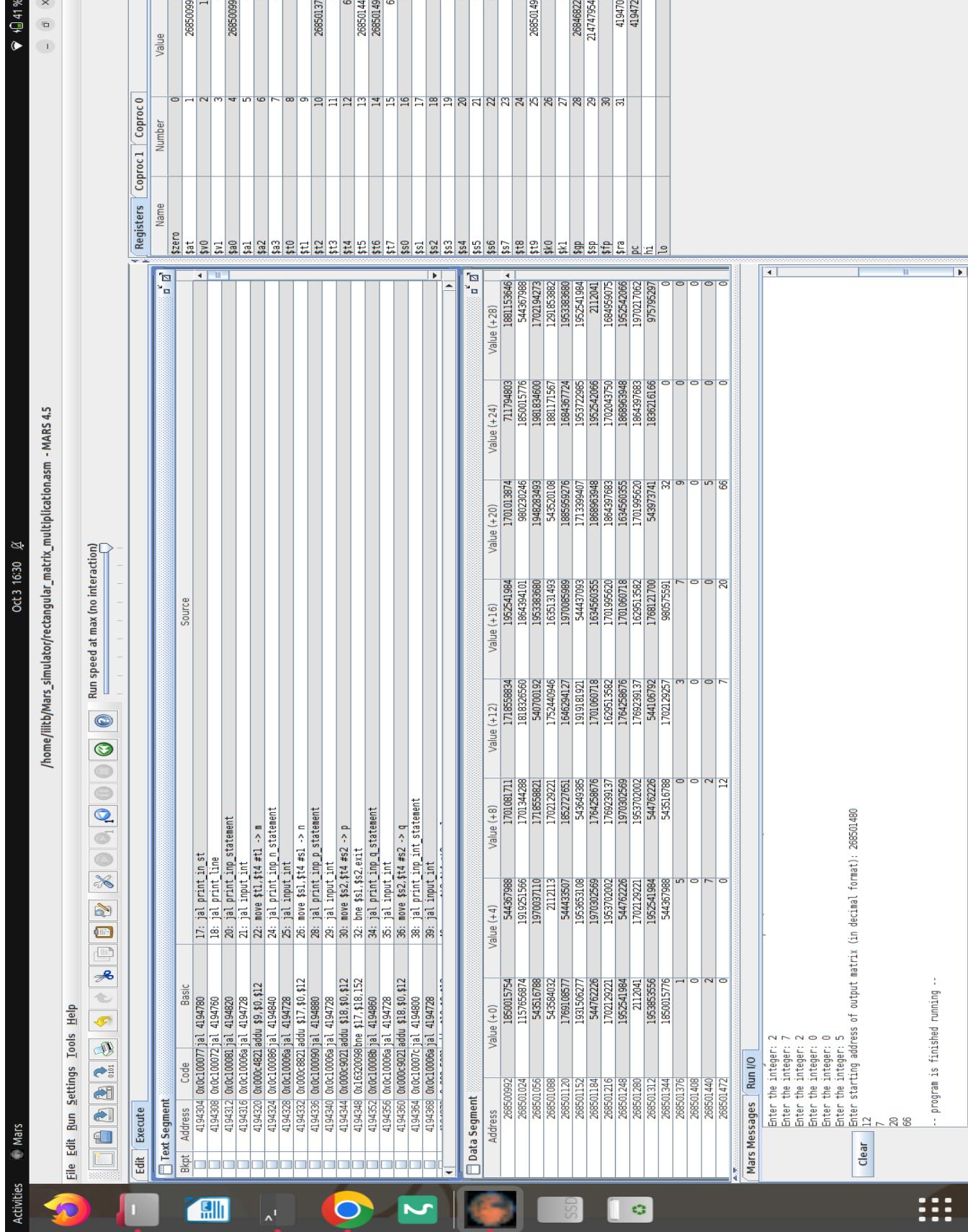


Figure 2.8. Output

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.

Oct 4 01:21

Activities

Google Chrome

AssembleyEditor

File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

1# Opcodes for r format instructions. (All zero)

1	2	r_format = { 'sll' : 0x00,
2	3	'srl' : 0x90,
3	4	'sra' : 0x90,
4	5	'mult' : 0x00,
5	6	'multu' : 0x00,
6	7	'div' : 0x00,
7	8	'divu' : 0x00,
8	9	'sllv' : 0x90,
9	10	'srlv' : 0x90,
10	11	'srav' : 0x90,
11	12	'add' : 0x00,
12	13	'addu' : 0x90,
13	14	'sub' : 0x00,
14	15	'subu' : 0x90,
15	16	'and' : 0x00,
16	17	'or' : 0x00,
17	18	'xor' : 0x00,
18	19	'nor' : 0x00,
19	20	'slt' : 0x00,
20	21	'situ' : 0x90,
21	22	'mflo' : 0x00
22	23	}
23	24	# Opcodes for i format instructions.
24	25	i_format = { 'beq' : 0x04,
25	26	'bne' : 0x05,
26	27	'blez' : 0x96,
27	28	'bgtz' : 0x97,
28	29	'addi' : 0x98,
29	30	'addiu' : 0x09,
30	31	'slti' : 0x9A,
31	32	'sltiu' : 0x0B,
32	33	'andi' : 0x9C,
33	34	'ori' : 0x0D,
34	35	'xori' : 0x9E,
35	36	'lui' : 0x0F,
36	37	'lbu' : 0x20,
37	38	}
38	39	completed at 01:20

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

The screenshot shows a terminal window with assembly code for a binary converter function. The code is as follows:

```
92 # This function converts an integer to binary with N total digits, using necessary padding with 0's.
93 def binary(N, num):
94     if num < 0:
95         num = int(0xffffffff) + num
96     return f"{num:{N}b}"
97
98 # This function converts a 32 bit binary number given as a string to an 8 Bit hexadecimal string.
99 def bin4hex(string):
100     hexDict = {'0000': '0',
101             '0001': '1',
102             '0010': '2',
103             '0011': '3',
104             '0100': '4',
105             '0101': '5',
106             '0110': '6',
107             '0111': '7',
108             '1000': '8',
109             '1001': '9',
110             '1010': 'a',
111             '1011': 'b',
112             '1100': 'c',
113             '1101': 'd',
114             '1110': 'e',
115             '1111': 'f'}
116
117     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[string[24:28]] + hexDict[string[28:32]])
118
119 # This function holds addresses of statements as given by MARS (Hardcoded)
120 def jumpAddress(something):
121
122     jumpDict = {
123         'done' : 35,
124         'kloop' : 9,
125         'jloop' : 3,
126         'lloop' : 1048885,
127
128     }
129     return jumpDict[something]
```

Figure 2.10. Function for shift and jump

```

Activities
Google chrome
Assembleripyrb - Colab
Cisco Skills Fo...
File Edit View Insert Runtime Tools Help All changes saved
+ Code + Text
131 # This function gives 'func' value in the machine code.
132 def shamt(something):
133     shifts = ['sll', 'srl', 'sra']
134     if (something in shifts):
135         return 10
136     else:
137         return 0
138
139 # This function gives the 'func' value in the machine code.
140 def func(something):
141     funcDict = {
142         'sll' : 0,
143         'srl' : 2,
144         'sra' : 3,
145         'mult' : 24,
146         'multu': 25,
147         'div' : 26,
148         'divu': 27,
149         'sllv': 4,
150         'srav': 6,
151         'sra' : 7,
152         'add' : 32,
153         'addu': 33,
154         'sub' : 34,
155         'subu': 35,
156         'and' : 36,
157         'or' : 37,
158         'xor': 38,
159         'nor': 39,
160         'slt' : 42,
161         'sltu': 43
162     }
163     return funcDict[something]
164
165
166 # This function returns the 32 bit machine code for the instruction it is passed.
167 def MachineCodeMaker(l):
168     i = 0
169     # Each conditional statement is made to filter different types of functions and make the necessary binary string.

```

Figure 2.11. Branch target addresses (hardcoded)

```

Activities Google chrome
Assembleripyrb - Colab + 
Cisco Skills Fo... Intro to semi... Semiconducto... Hardwaredemo... MadhavRao si... Cybersecurity... X Introduction t... X Beginner... X Git and GitHub...
File Edit View Insert Runtime Tools Help All changes saved
+ Code + Text
167 def MachineCodeDecoder(l1):
168     l = 0
169     # Each conditional statement is made to filter different types of functions and make the necessary binary string.
170     if processed[i] == 'sll':
171         return ('0000000000' + binaryN((registers[int(i+2)]),5) + binaryN((registers[int(i+1)]),5) + binaryN(int(l[i+3]),5) + '00000000')
172     elif processed[i] in ['lw', 'sw']:
173         return (binaryN(int(i_format[l1])),6) + binaryN((registers[int(i+3)]),5) + binaryN((registers[int(l[i+1])],5) + binaryN(int(l[i+2])),16)
174     elif processed[i] in ['beq', 'bne']:
175         return (binaryN(int(i_format[l1])),6) + binaryN((registers[int(i+1)]),5) + binaryN((registers[int(l[i+2])],5) + binaryN(jumpAddress(l[i+3])),16)
176     elif processed[i] in ['mult', 'multu']:
177         return ('000000' + binaryN((registers[int(i+1)]),5) + binaryN((registers[int(i+2)]),5) + binaryN(func(l[i])),6))
178     elif processed[i] == 'mflo':
179         return ('0000000000' + binaryN((registers[int(i+1)]),5) + '00000010010')
180     elif processed[i] in r_format(l1):
181         return (binaryN(int(r_format[l1])),6) + binaryN((registers[int(i+1)]),5) + binaryN((registers[int(l[i+2])],5) + binaryN(shamt(l[i+1]),5) + binaryN(shamt(l[i+1])),5) + binaryN(int(l[i+3])),16)
182     elif processed[i] in i_format.keys():
183         return (binaryN(int(i_format[l1])),6) + binaryN((registers[int(i+2)]),5) + binaryN((registers[int(l[i+1])],5) + binaryN(int(l[i+3])),16)
184     elif processed[i] in j_format.keys():
185         if processed[i] == 'j' and processed[i+1] == 'jloop':
186             return ('00000000000000000000000000000000')
187         return (binaryN(int(j_format[l1])),6) + binaryN(jumpAddress(l[i+1]),26))
188
189
190 # Open the file with the code written for matrix multiplication.
191 f = open('matrixcore.asm', 'r')
192
193 # Go through each line.
194 for line in f.readlines():
195     current = line[:1]
196     commentless = (current.split('#')) # Separate the comments
197     processed = (commentless[0].replace(' ', ' ').replace('\t', '')).split() # Split the line so we isolate the values
198
199 if len(processed) < 2: # Blank lines and statements omitted.
200     continue
201 if ((processed[0] in r_format.keys()) or (processed[0] in j_format.keys()) or (processed[0] in i_format.keys())):
202     code = bin4hex(MachineCodeDecoder(processed))
203     txt = open('machinematrix.txt', "a")
204     # for e in processed:

```

Figure 2.12. File I/O

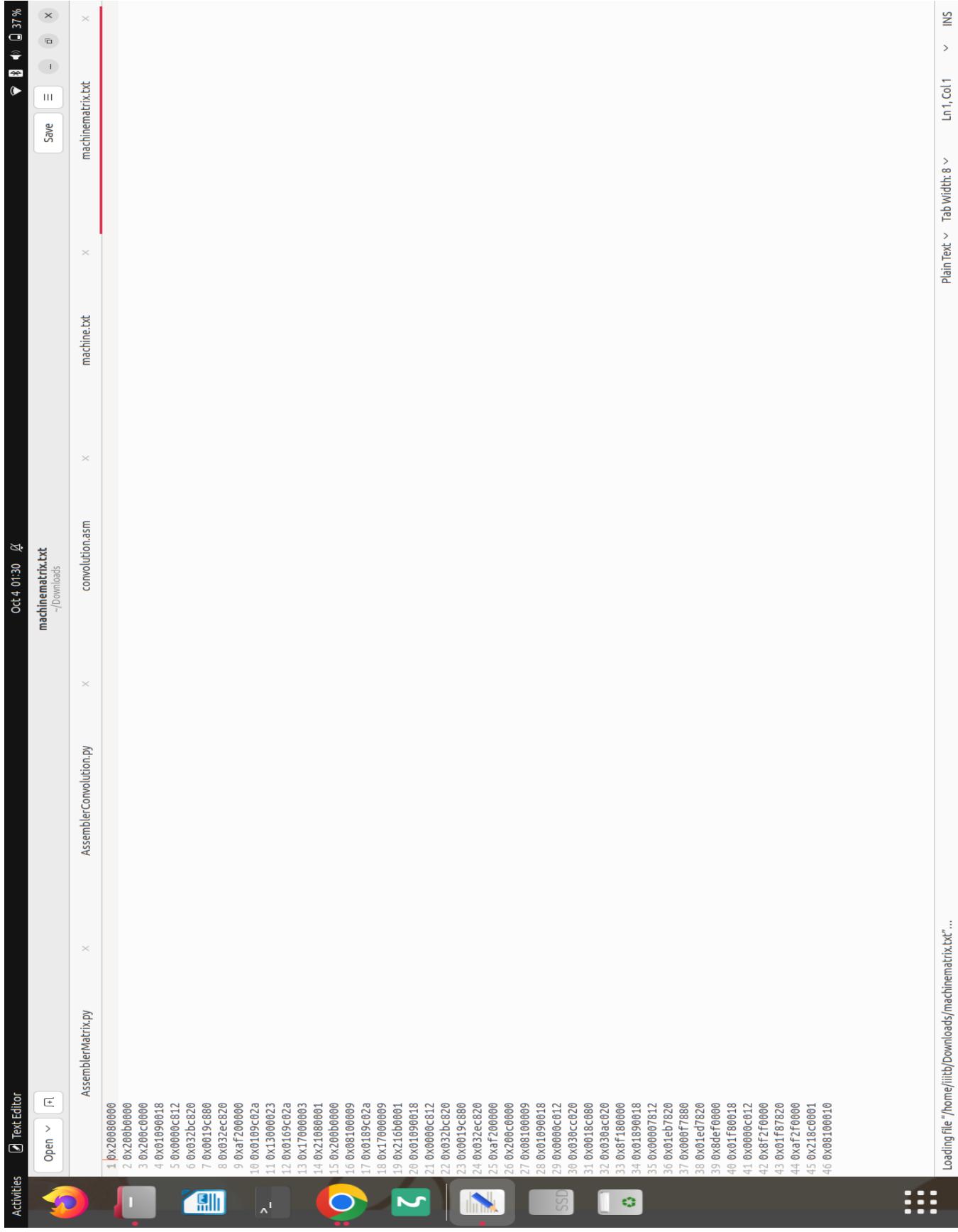


Figure 2.13. Output text file

The screenshot shows a mobile application interface for managing files. At the top, there's a status bar with the date "Oct-4 02:29" and a battery level of "18%". Below the status bar are two tabs: "dumpedhexmatrix.txt" and "dumpedhexconvolution.txt", both located in the directory "~/Downloads". Each tab has its own set of controls: "Save", "Open", and "X". The "dumpedhexmatrix.txt" tab is currently active, showing a large list of numerical values from 1 to 46. The "dumpedhexconvolution.txt" tab is also visible but appears to be empty or has very small text. At the bottom of the screen is a navigation bar with various icons: a circular arrow, a minus sign, a document icon, a browser icon, a green square icon, a pencil and paper icon, an SSD icon, a trash bin icon, and a recycling bin icon.

File	Content Preview
dumpedhexmatrix.txt	1 20000000 2 20000000 3 20000000 4 010900018 5 00000C812 6 632b0c820 7 0019c880 8 032cc020 9 0f780000 10 01090c02a 11 130000023 12 01690c02a 13 10000003 14 21000001 15 20000000 16 68100009 17 01890c02a 18 10000009 19 21600001 20 010900018 21 00000C812 22 632b0c820 23 0019c880 24 032cc020 25 af200000 26 20000000 27 68100009 28 010900018 29 00000C812 30 03000C020 31 0018c080 32 632b0c020 33 8f100000 34 018900018 35 000007812 36 01eb7820 38 01ed7820 39 8dfe0000 40 01f800018 41 00000C012 42 8f12f6000 43 01f87820 44 af2f6000 45 218c00001 46 081000010
dumpedhexconvolution.txt	File content is mostly illegible due to low resolution, appearing as small black dots.

Plain Text ▾ Tab Width: 8 ▾ Ln 25, Col 9 ▾ INS

Loading file "/home/lllbt/Downloads/dumpedhexmatrix.txt" ...

Figure 2.14. MARS machine code output text file