CPE 333 Lab 0

Matrix-Matrix Multiplication

Report by:
Ethan Vosburg (evosburg@calpoly.edu)
Isaac Lake (ilake@calpoly.edu)

April 6, 2024

Table of Contents

1 Introduction

In this lab, we will be writing an assembly program to multiply matrices together. In this case, we will be provided 2 nxn matrices of known dimensions and then perform the multiplication. Notably, we will be using the stack to hold values and pointers to preform operations.

2 Roles

Listed below are the group members

1. **Ethan:**
   Report, Version Control, Verification
2. **Isaac:**
   Multiplication Function, initialization, .data section
3. **Sam:**
   Matrix Traversal
4. **Victoria:**
   Row/Column Multiplication

3 Source Code

```
###############################################################################
######################## CPE 333 LAB 0 ########################################
###############################################################################
#Isaac Lake, Ethan Vosburg, Samuel Solano, and Victoria Asencio-Clemens


.data
A:
    # .word 0, 3, 2, 0, 3, 1, 0, 3, 2
    .word 1, 1, 1, 1, 1,
          2, 2, 2, 2, 2,
          3, 3, 3, 3, 3,
          4, 4, 4, 4, 4,
          5, 5, 5, 5, 5

B:
    # .word 1, 1, 0, 3, 1, 2, 0, 0, 0
    .word 1, 1, 1, 1, 1,
          2, 2, 2, 2, 2,
          3, 3, 3, 3, 3,
          4, 4, 4, 4, 4,
          5, 5, 5, 5, 5

```

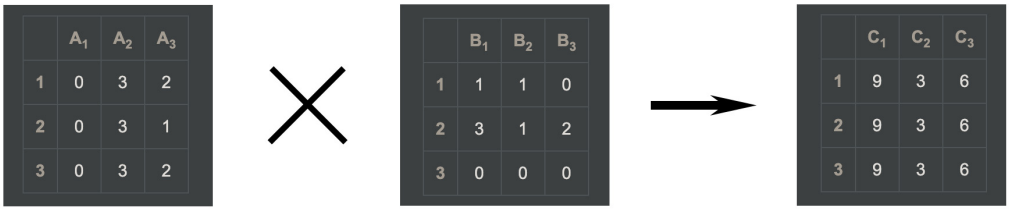
```
C:
    # .word 0, 0, 0, 0, 0, 0, 0, 0, 0
    .word 0, 0, 0, 0, 0,
          0, 0, 0, 0, 0,
          0, 0, 0, 0, 0,
          0, 0, 0, 0, 0,
          0, 0, 0, 0, 0



.text
################################################################################
####################### Initialization #########################################
################################################################################

    li    sp, 0x10000               # Initialize the stack pointer
    la    s0, A                     # Load the base address of A into s0
    la    s1, B                     # Load the base address of B into s1
    la    s2, C                     # Load the base address of C into s2
    # li    s3, 3                     # Load the dim_size into s3
    li    s3, 5                     # Load the dim_size into s3
    mv    a6, s3                    # Load the dim_size into a6
    mv    a7, s3                    # Load the dim_size into a7
    # Calsulate the number of elements in the matrix
    call  MULT                      # Call multi func to init a7
    mv    s5, a7                    # Store the number of elements in s5

################################################################################
####################### Matrix Traversal Function ##############################
################################################################################

#a0 row number
#a1 column number
#a2 return value


    addi  a0, x0, -1                # row number to input into RCMULT
    addi  a1, x0, 0                 # var for overall loop and - col number
    addi  t0, x0, 0                 # var for nested loop - row loop
    addi  t1, x0, 0                 # counter for the array (inc by 4)


LOOP:
    beq   t0, s3, DONE
    addi  a1, x0, 0                 # var for nested loop
    addi  a0, a0, 1
    addi  t0, t0, 1


NESTED:
    beq   a1, s3, LOOP
    call  RCMULT
    sw    a2, 0(s2)                 # put mult result into s4
    addi  s2, s2, 4                 # change s4 to store location
                    # (s2 (array start) + t1 (offset))
    addi  t1, t1, 4                 # update t1 which is the array offset
```

```
    addi  a1, a1, 1                 # update nested var
    j     NESTED

###############################################################################
#################### Row / Column Multiplication ##############################
###############################################################################

#Inputs: a0, a1
#Outputs: a2
#Registers Changed: t4, t5, t6, a3, a6, a7
RCMULT:
    addi  sp, sp, -4                # move stack pointer
    sw    ra, 0(sp)                 # push return address to stack
    mv    a6, a0                    # move row num to a6 for multiplication
    mv    a7, s3                    # move dim_size to a7 for muliplication
    call  MULT                      # multiply row number by dim_size
    slli  t4, a7, 2                 # row product x 4 = offset for first element
    add   t4, s0, t4                # add to pointer for matrix array A
    mv    t5, a1                    # move column number to t5
    slli  t5, t5, 2                 # multiply by 4 to get offset for first
   element
    add   t5, s1, t5                # add to pointer for matrix array B
    mv    t6, zero                  # initialize value for looping
    mv    a2, zero                  # initialize total sum
DOTPROD:
    bge   t6, s3, RCMULTEND         # branch when done with dot product
    lw    a6, 0(t4)                 # load first row value from A
    lw    a7, 0(t5)                 # load first column value from B
    call  MULT                      # multiply row and column values
    add   a2, a2, a7                # add product to total
    addi  t4, t4, 4                 # move to next row value
    slli  a3, s3, 2                 # a3 = dim_size x 4
    add   t5, t5, a3                # move to next column value
    addi  t6, t6, 1                 # increment number for looping
    j     DOTPROD                   # loop
RCMULTEND:
    lw    ra, 0(sp)                 # load return address
    addi  sp, sp, 4                 # return stack pointer to original value
    ret                             # return

###############################################################################
#################### Multiplication Function ##################################
###############################################################################
#Note only works for positive integers

#Inputs: a6, a7
#Outputs: a7
#Registers Changed: t0, t1, t2, t3
MULT:
    addi  sp, sp, -20               # Adjust stack pointer to make room for
                    # 5 registers (ra, t0, t1, t2, t3)
    sw    ra, 16(sp)                # Push ra to stack
    sw    t0, 12(sp)                # Push t0 to stack
    sw    t1, 8(sp)                 # Push t1 to stack
    sw    t2, 4(sp)                 # Push t2 to stack
    sw    t3, 0(sp)                 # Push t3 to stack
```

```
135
136     li    t0, 0                     # Running Total
137     li    t1, 1                     # Loop "Counter"
138     li    t2, -1                    # Starts at -1 Due to how we increment it
139
140 MULTI:
141     beqz  t1, MULTEND               # Checks if we are finished with the mult
142     and   t3, t1, a7                # Checks if we should add this loop
143     addi  t2, t2, 1
144     slli  t1, t1, 1
145     beqz  t3, MULTI
146     sll   t3, a6, t2
147     add   t0, t0, t3
148     j     MULTI
149
150 MULTEND:
151     mv    a7, t0
152
153     lw    t3, 0(sp)                 #Pop t3 from the stack
154     lw    t2, 4(sp)                 #Pop t2 from the stack
155     lw    t1, 8(sp)                 #Pop t1 from the stack
156     lw    t0, 12(sp)                #Pop t0 from the stack
157     lw    ra, 16(sp)                #Pop return address from the stack
158     addi  sp, sp, 20                #Adjust stack pointer back
159     ret
160
161 DONE:
162     nop
```

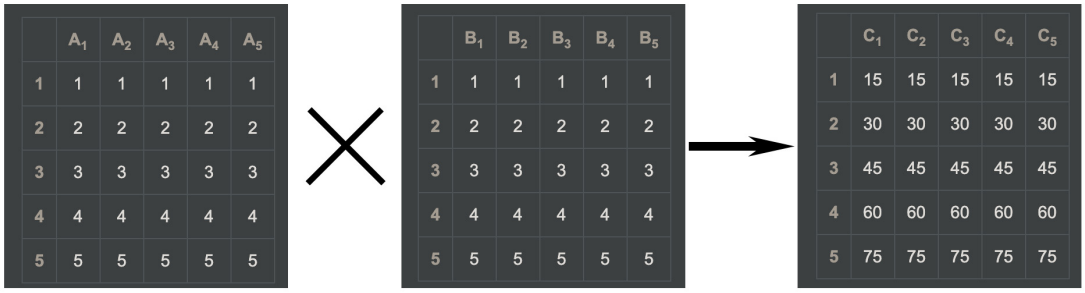
Listing 1: Matrix-Matrix Multiplication Assembly Code

4 Verification



Address	Value (...	Value (...	Value (...	Value (...	Value (...	Value (...	Value (...	Value (...
0x00...	0	3	2	0	3	1	0	3
0x00...	2	1	1	0	3	1	2	0
0x00...	0	0	9	3	6	9	3	6
0x00...	9	3	6	0	0	0	0	0
0x00...	0	0	0	0	0	0	0	0
0x00...	0	0	0	0	0	0	0	0
0x00...	0	0	0	0	0	0	0	0
0x00...	0	0	0	0	0	0	0	0
0x00...	0	0	0	0	0	0	0	0
0x00...	0	0	0	0	0	0	0	0

Figure 1: 3x3 Matrix Verification



Address	Valu...	Val...	Valu...	Val...	Value...	Val...	Valu...	Value (...
0x00006000	1	1	1	1	1	2	2	2
0x00006020	2	2	3	3	3	3	3	4
0x00006040	4	4	4	4	5	5	5	5
0x00006060	5	1	1	1	1	1	2	2
0x00006080	2	2	2	3	3	3	3	3
0x000060a0	4	4	4	4	4	5	5	5
0x000060c0	5	5	15	15	15	15	15	30
0x000060e0	30	30	30	30	45	45	45	45
0x00006100	45	60	60	60	60	60	75	75
0x00006120	75	75	75	0	0	0	0	0

Figure 2: 5x5 Matrix Verification