Assignment 2 : EE675 Microprocessor Applications in Power Electronics

Shreyas Shyamsunder
113300025
Prashant Rupapara
11307R008
Group 10

Oct 09,2012

1. This question requires us to make 4 subroutines for getting and setting the required element in a matrix. The value of array address is placed in ARO, i in R0 and j in R1. i and j are the indexes required for accessing the corresponding element in the matrix. This question is dealt in 4 parts.

   (a) *getnum_*2 subroutine accepts the value of i and j in R0 and R1 for a two dimensional matrix and places the array element in R0. Even though the array is two dimensional, it is stored in contiguous memory locations.The appropriate element needs to be indexed after the initial offset of the array header is taken care of. The index of the element can be calculated as

   $$index = (j * R0) + R1 + 4 + R0$$

   where j is the number of columns in the matrix which is stored in the 4th memory location from the base address of the array. Once the index is known the corresponding element can be acccessed and placed in R0. If the value placed in R0 and R1 is more than the maximum allowable(which is placed in the array header), then the appropriate error bit is set. This error check is done at the beginning of the subroutine.

   (b) *getnum_*1 subroutine accepts the value of i in R0 for a one dimensional matrix and places the array element in R0. It does not matter whether the matrix is a row matrix or a column matrix, it will be stored in the same way in memory. This subroutine does not check whether matrix is one dimensional or not. It is assumed that the user has taken care of that problem. It was not mentioned in the question to tackle this issue. The appropriate element needs to be accessed appropriate element needs to be indexed after accounting for the offset due to the header. The index of the element can be calculated as

$$index = R0 + 4$$

where j is the number of columns in the matrix which is stored in the 4th memory location from the base address of the array. Once the index is known the corresponding element can be acccessed and placed in R0. If the value placed in R0 is more than the maximum allowable number of columns or rows(which is placed in the array header), then the appropriate error bit is set.This error check is done at the beginning of the subroutine. The value of R0 is checked with both the row or column entries in the header. This is done because the matrix could either be a row or a column matrix. As mentioned above it is assumed that atleast one of these values is 0, this subroutine does not address the issue when both number of rows as well as columns is a finite number in the header(denoting a 2 dimensional matrix).

(c) *setnum_2* subroutine accepts the value of i,j and the data in R0,R1 and R3 respectively and stores R3 in the corresponding position in the matrix. The appropriate index is calculated same as for the getnum_2.

$$index = (j * R0) + R1 + 4 + R0$$

where j is the number of columns in the matrix which is stored in the 4th memory location from the base address of the array. Once the index is known the value in R3 is placed in the corresponding matrix location. The error check is same as the one for getnum_2. This error check is done at the beginning of the subroutine.

(d) *setnum_1* subroutine accepts the value of i adn data in R0 and R3 and stores R3 in the corresponding position in the matrix. The appropriate index is calculated same as for the getnum_1.

$$index = R0 + 4$$

Once the index is known the value in R3 is placed in the corresponding matrix location. The error check is same as the one for getnum_1. This error check is done at the beginning of the subroutine.

The above subroutines were successfully tested with the TMS320VC33 board. All the subroutines are made global and stored in the file asubs.asm.

2. In this question we are required to perform arithmetic operations on the matrix. It is assumed that the matrices are already stored in memory locations. This question is dealt in 4 parts.

   (a) *mmult* This subroutine accepts the starting addresses of the two matrices in AR0 and AR1 respectively and calculates C = A*B. However first the dimension check is done. Two matrices can be multiplied if the number of columns of the first matrix is equal to the number of rows of the second matrix. This check is performed first as all the values is known(placed in the matrix header). If this is satisfied, the dimensions of the new matrix is written in the appropriate header position for C(given be base address AR2).

   Our subroutine uses the previous subroutines getnum_2 and setnum_2(written for question 1) to access and place the element in the corresponding positions in the matrix respectively.

   We know that a particular element in C can be given as

$$C[i,k] = \sum_{j=0}^{J} A(i,j) * B(j,k)$$

   where J is the number of columns of matrix1 or number of rows of matrix2.

   In our subroutine, we have generated the appropriate indexes for the three matrices using loops. We have made use of the conditional branch instructions to get the appropriate values of i,j and k. These values are then passed onto the subroutines to get the value of the matrix element. This is multiplied and accumulated and then stored into the index of the matrix C using the setnum_2 subroutine.

(b) *madd* This subroutine performs the operation C = A+B where the base addresses for A,B and C are stored in AR0, AR1 and AR2 respectively.Our subroutine first performs the check for dimensions. Two matrices can be added or subtractd only if their dimensions match. This check is performed and the appropriate error bit is set at the beginning of the subroutine. We know that

$$C[i, j] = A(i, j) + B(i, j)$$

In our subroutine, we have used a loop to calculated sum of each corresponding element of A and B. The result is stored in the appropriate address using AR2.

(c) *msub* This subroutine performs the operation C = A-B where the base addresses for A,B and C are stored in AR0, AR1 and AR2 respectively.Our subroutine first performs the check for dimensions.The check for dimensions is performed and the appropriate error bit is set. We know that

$$C[i, j] = A(i, j) - B(i, j)$$

In our subroutine, we have used a loop to calculated difference of each corresponding element of A and B. The result is stored in the appropriate address using AR2. (Please note that matrix elements are stored in contiguous memory locations).

(d) *mneg* This subroutine performs the operation C = -A where the base addresses for A and C are stored in AR0 and AR2 respectively.Our subroutine first performs the check for dimensions.The check for dimensions is performed and the appropriate error bit is set. We know that

$$C[i, j] = -A(i, j)$$

In our subroutine, we have used a loop to calculated difference of each corresponding element of C. The instruction negf is used to negate. The result is stored in the appropriate address using AR2. (Please note that matrix elements are stored in contiguous memory locations).

(e) *mtrn* This subroutine performs the operation $C = A^T$ where the base addresses for A and C are stored in AR0 and AR2 respectively.Our subroutine first performs the check for dimensions.The check for dimensions is performed and the appropriate error bit is set. We know that

$$C[i, j] = A(j, i)$$

The logic for calculating the transpose is as follows. (pseudo code)

```
IR0 = 4
for(R1=0,R1<(number of columns+1),R1++)
{
IR1 = R1 + 4
for(R2=0,R2<(number of rows+1),R2++)
{
C[IR0]=A[IR1]
IR1 = R1+R5+4
IR0 = IR0 + 1
}
}
```

The above subroutines were successfully compiled and checked on the TMS320VC33 board. The main calling program initializes the required matrices. Also all the operations were performed on floating point numbers with appropriate checks for overfow and underflow. This was done by monitoring the v bit and the UF bit in the status register.